



ICE ClusterWare Documentation

Release 12.4.2

Penguin Computing

Mar 14, 2025

CONTENTS

1	ICE ClusterWare Overview	12
1.1	Cluster Architecture Overview	12
1.2	The ClusterWare Database	16
1.3	Provisioning Compute Nodes	17
2	Quickstart	18
2.1	Prerequisites	18
2.2	Create Administrator	18
2.3	Install ClusterWare	19
2.4	Configure Boot Image and Job Scheduler	20
3	Install	22
3.1	Supported Distributions and Features	22
3.2	Required and Recommended Components	24
3.3	Install ICE ClusterWare	25
3.3.1	Download the ICE ClusterWare Install Script and Related Files	26
3.3.2	Execute the ICE ClusterWare Install Script	27
3.4	scylld-install	31
3.5	scylld-tool-config	32
3.6	scylld-cluster-conf	33
3.7	Securing the Cluster	35
3.7.1	Authentication	35
3.7.1.1	Assign Temporary Permissions	36
3.7.2	Role-Based Access Controls	36
3.7.3	Changing the Database Password	38
3.7.3.1	Changing the etcd Password	38
3.7.4	Compute Node Remote Access	39
3.7.5	Compute Node Host Keys	39
3.7.6	Encrypting Communications	39
3.7.6.1	Configure Encrypted Communication between Head and PXE Compute Nodes	40
3.7.6.2	Configure Encrypted Communication between Head and Diskful Compute Nodes	41
3.7.6.3	Configure Client Authentication between Head and Compute Nodes	41
3.7.7	Security-Enhanced Linux (SELinux)	41
3.7.7.1	SELinux On Compute Nodes	41
3.7.7.2	SELinux On Head Nodes	42
3.7.7.3	MLS Policy On Head Nodes	42
3.7.8	Security Technical Implementation Guides (STIG)	42
3.8	Services, Ports, Protocols	43
3.8.1	Apache	43

3.8.2	Chrony	43
3.8.3	DHCP	43
3.8.4	DNS	43
3.8.5	etcd	44
3.8.6	iSCSI	44
3.8.7	OpenSSH	44
3.8.8	Telegraf / Telegraf-Relay / InfluxDB	44
3.8.9	TFTP	44
3.9	Common Additional Configuration	44
3.9.1	Configure Hostname	44
3.9.2	Managing Databases	44
3.9.3	Configure Administrator Authentication	45
3.9.4	Disable/Enable Chain Booting	46
3.9.5	scylid-nss Name Service Switch (NSS) Tool	46
3.9.6	Firewall Configuration	46
3.9.7	Configure IP Forwarding	47
3.9.8	Status and Health Monitoring	47
3.9.9	Install Name Service Cache Daemon (nscd)	47
3.9.10	Install jq Tool	47
3.10	Additional Software	49
3.10.1	Adding 3rd-party Software	49
3.10.2	Job Schedulers	50
3.10.2.1	Slurm	51
3.10.2.2	OpenPBS	53
3.10.2.3	PBS TORQUE	55
3.10.3	Kubernetes	56
3.10.3.1	Bootstrap Kubernetes Control Plane	57
3.10.3.2	Checking Deployment Status	58
3.10.3.3	Additional Configuration	58
3.10.3.4	Adding Workers	59
3.10.4	OpenMPI, MPICH, and/or MVAPICH	62
4	Administration	64
4.1	Introduction	64
4.2	ICE ClusterWare Graphical User Interface	64
4.3	ICE ClusterWare Command Line Tools	66
4.3.1	--all and --ids	66
4.3.2	--config	66
4.3.3	--base-url and --user	67
4.3.4	--show-uids, --human, --json, --pretty/--no-pretty	67
4.3.5	--csv, --table, --fields	67
4.4	Common Subcommand Actions	68
4.4.1	list (ls)	68
4.4.2	create (mk)	68
4.4.3	clone (cp)	68
4.4.4	update (up)	68
4.4.5	replace (re)	68
4.4.6	delete (rm)	68
4.5	Files in database objects	68
4.6	The then argument	69
4.7	The --content argument	69
4.8	Variable Substitution	71
4.8.1	Node Attributes, Hardware, and Status	71
4.8.2	Head Node Substitutions	71

4.8.3	Kickstarting From A Repo	72
4.9	Manage Cluster	72
4.9.1	Cluster Overview Page	72
4.9.2	Cluster Settings	73
4.9.2.1	Cluster ID	73
4.9.2.2	Cluster Name	73
4.9.2.3	Accepting Unknown Nodes	73
4.9.2.4	Time Offset	73
4.9.3	scylid-clusterctl	73
4.9.4	scylid-nssctl	79
4.9.5	IP Forwarding Issues	79
4.9.6	managedb	80
4.9.7	ICE ClusterWare Log Files	82
4.9.8	Creating Diagnostic Test Images	82
4.9.9	scylid-sysinfo	83
4.10	Create Login Nodes	84
4.11	Update and Upgrade	87
4.11.1	Updating ICE ClusterWare Software	87
4.11.1.1	Updating head nodes	87
4.11.1.2	Updating compute nodes	88
4.11.1.3	Updating ClusterWare 11 to ClusterWare 12	88
4.11.2	Updating Firmware	89
4.11.3	Updating Base Distribution Software	89
4.12	Backup and Restore	90
4.12.1	Backup and Restore of ICE ClusterWare Software	90
4.12.2	Backup and Restore of the Database	90
4.12.2.1	take-snapshot	90
4.13	Interacting with Compute Nodes	92
4.13.1	scylid-nodectl	92
4.14	Nodes Page	98
4.14.1	Node Filtering	99
4.14.2	Node Grid Display	100
4.14.3	Node List Display	101
4.14.4	Create Compute Node	102
4.14.5	Edit Compute Node	102
4.14.6	Delete Compute Node	103
4.14.7	Related Links	103
4.15	Executing Commands	103
4.16	Create Nodes	104
4.16.1	Node Creation with Known MAC address(es)	104
4.16.2	Node Creation with Unknown MAC address(es)	105
4.16.3	Support for Diskful Nodes	106
4.16.3.1	Pre-Installer Script	106
4.16.3.2	Installer Scripts	106
4.16.3.3	Installation Logs	107
4.16.3.4	Head Node Preparation	107
4.16.3.5	RPM and DEB Installations	107
4.16.3.6	TAR Installations	107
4.16.4	Compute Node Fields	108
4.16.5	Compute Nodes IPMI Access	108
4.17	Boot Nodes	109
4.17.1	Compute Node Initialization Scripts	109
4.17.2	Booting From Local Storage Cache	110
4.17.2.1	Failing To Boot From Local Storage	111

4.17.3	Booting Diskful Compute Nodes	111
4.17.3.1	Installing the clusterware-node Package	112
4.17.3.2	Additional Support for Diskful Nodes	112
4.17.4	scylid-reports	112
4.18	Manage Nodes	114
4.18.1	Changing IP Addresses	114
4.18.2	Node Name Resolution	114
4.18.3	Command-Line Monitoring of Nodes	115
4.18.4	Managing Node Failures	116
4.18.4.1	Replacing Failed Nodes	116
4.18.5	Soft Power Control Failures	117
4.18.6	Managing Large Clusters	117
4.18.6.1	Improve Scaling of Node Booting	117
4.18.7	Hostnames Page	117
4.18.7.1	Create a Hostname	118
4.18.7.2	Edit Hostname	118
4.18.7.3	Delete Hostname	119
4.18.7.4	Related Links	119
4.18.8	Manage Non-ICE ClusterWare Entities	119
4.19	Attribute Groups	119
4.19.1	Database Objects Fields and Attributes	119
4.19.2	Node Attributes Page	120
4.19.2.1	Create Attribute Group	120
4.19.2.2	Edit Attribute Group	121
4.19.2.3	Delete Attribute Group	121
4.19.2.4	Change Default Attribute Group	121
4.19.2.5	Related Links	121
4.19.3	Node Attributes	121
4.19.4	Dynamic Groups Page	123
4.19.4.1	Create Dynamic Group	123
4.19.4.2	Edit Dynamic Group	123
4.19.4.3	Filter Nodes by Dynamic Group	123
4.19.4.4	Delete Dynamic Group	124
4.19.4.5	Related Links	124
4.19.5	Attribute Groups and Dynamic Groups	124
4.19.6	scylid-attribctl	126
4.19.7	Reserved Attributes	128
4.19.7.1	_aim_status	128
4.19.7.2	_altmacs	128
4.19.7.3	_ansible_pull	128
4.19.7.4	_ansible_pull_args	129
4.19.7.5	_ansible_pull_now	129
4.19.7.6	_bmc_pass	129
4.19.7.7	_bootloader	129
4.19.7.8	_bootnet	129
4.19.7.9	_busy	130
4.19.7.10	_boot_config	130
4.19.7.11	_boot_rw_layer	130
4.19.7.12	_boot_style	131
4.19.7.13	_boot_tmpfs_size	131
4.19.7.14	_coreos_ignition_url	131
4.19.7.15	_coreos_install_dev	131
4.19.7.16	_disk_cache	132
4.19.7.17	_disk_root	133

4.19.7.18	_disk_wipe	133
4.19.7.19	_domain	133
4.19.7.20	_gateways	133
4.19.7.21	_hardware_plugins	134
4.19.7.22	_hardware_secs	134
4.19.7.23	_health	134
4.19.7.24	_health_check	135
4.19.7.25	_health_plugins	135
4.19.7.26	_health_secs	135
4.19.7.27	_health_check_secs	135
4.19.7.28	_hostname	136
4.19.7.29	_hosts	136
4.19.7.30	_ignition	136
4.19.7.31	_ips	136
4.19.7.32	_ipxe_sanboot	137
4.19.7.33	_macs	137
4.19.7.34	_no_boot	137
4.19.7.35	_preferred_head	137
4.19.7.36	_remote_pass	138
4.19.7.37	_remote_user	138
4.19.7.38	_sched_extra	138
4.19.7.39	_sched_full	139
4.19.7.40	_sched_state	139
4.19.7.41	_status_cpuset	139
4.19.7.42	_status_hardware_secs	139
4.19.7.43	_status_packages_secs	140
4.19.7.44	_status_plugins	140
4.19.7.45	_status_secs	140
4.19.7.46	_telegraf_omit_pattern	140
4.19.7.47	_telegraf_plugins	141
4.19.7.48	_tpm_owner_pass	141
4.19.7.49	_wipe_all	141
4.20	Naming Pools Page	141
4.20.1	Create a Naming Pool	142
4.20.2	Edit Naming Pool	142
4.20.3	Delete Naming Pool	142
4.20.4	Change Default Naming Pattern	142
4.20.5	Related Links	142
4.21	Node Names and Pools	142
4.21.1	Node Indexing and Grouping in Naming Pools	143
4.21.2	Secondary Naming Pools	145
4.21.2.1	Configuration File	145
4.21.2.2	Command Line Tools	145
4.22	Boot Configurations Page	145
4.22.1	Create Boot Configuration	146
4.22.2	Edit Boot Configuration	146
4.22.3	Delete Boot Configuration	147
4.22.4	Related Links	147
4.23	scyl-d-add-boot-config	147
4.24	scyl-d-* Wrapper Scripts	148
4.25	Boot Configurations	148
4.25.1	Create Local Repo	150
4.25.2	Create Boot Configuration with Kickstart	150
4.25.3	scyl-d-mkramfs	151

4.25.4	scylld-bootctl	152
4.25.5	Freezing a Boot Configuration	155
4.25.6	Deleting Boot Configurations	155
4.25.7	Exporting and Importing Boot Configurations Between Clusters	155
4.25.8	Using Kickstart	156
4.25.8.1	Kickstart Files	156
4.25.8.2	Kickstart Failing	157
4.25.9	Using RHCOS	158
4.26	Software Images	159
4.26.1	Images	159
4.26.1.1	Images Page	159
4.26.1.2	Creating Images	160
4.26.1.3	Recreating the Default Image	161
4.26.1.4	Repos and Distros	162
4.26.1.5	Modifying Images	164
4.26.1.6	Caching in scylld-modimg	166
4.26.1.7	Updating the Kernel in an Image	167
4.26.1.8	Updating Drivers Inside Images	167
4.26.1.9	Capturing and Importing Images	168
4.26.1.10	Automating Common Image Tasks	169
4.26.1.11	Deploying Images Using Ignition	169
4.26.1.12	Freezing an Image	171
4.26.1.13	Deleting Unused Images	171
4.26.2	scylld-imgctl	171
4.26.3	scylld-modimg	173
4.26.4	Failing PXE Network Boot	176
4.26.5	Creating Local Repositories without Internet	177
4.26.6	Validating ClusterWare ISOs	179
4.26.6.1	make-iso	179
4.27	Image Sources Page	180
4.27.1	Repos	181
4.27.1.1	Create a Repo	182
4.27.1.2	Edit a Repo	182
4.27.1.3	Delete a Repo	182
4.27.2	Distros	182
4.27.2.1	Create a Distro	182
4.27.2.2	Edit a Distro	183
4.27.2.3	Delete a Distro	183
4.27.3	Related Links	183
4.28	Git Repositories	183
4.28.1	Initial Preparation	183
4.28.2	Locally Hosted Repositories	184
4.28.3	Mirroring Upstream Resources	185
4.28.4	Public Access	186
4.29	Git Repos Page	187
4.29.1	Create Git Repo	187
4.29.2	Edit Git Repo	187
4.29.3	Delete Git Repo	187
4.29.4	Related Links	188
4.30	State Maps	188
4.31	Grafana Telemetry Dashboard	189
4.31.1	Introduction to Grafana and InfluxDB	189
4.31.1.1	InfluxDB	189
4.31.1.2	Grafana	191

4.31.2	Grafana Setup Script	192
4.31.2.1	Arguments	192
4.31.2.2	Example	192
4.31.3	Grafana Login	193
4.31.4	Grafana Cluster Monitoring	193
4.31.4.1	Grafana General Page	194
4.31.4.2	Grafana Node Monitoring	195
4.31.4.3	Grafana Alerts	196
4.32	Workload Management	198
4.32.1	Monitoring Scheduler Info	198
4.32.1.1	sched-watcher Deployment	199
4.32.1.2	Verify Data	200
4.32.1.3	Config Settings	200
4.32.1.4	Notes	202
4.32.2	Applications Report Excessive Interruptions and Jitter	203
4.33	Role-Based Access Control System	203
4.33.1	Permissions	204
4.33.2	Roles	204
4.33.3	Modifying the Role-Permissions Mapping	205
4.34	Users and Groups Page	206
4.34.1	Add Administrator	207
4.34.2	Edit Administrator	207
4.34.3	Delete Administrator	207
4.34.4	Related Links	207
4.35	Configure Additional Cluster Administrators	207
4.35.1	scyl-d-adminctl	208
4.36	User Impersonation	210
4.37	Integrating Keycloak with ICE ClusterWare for RBAC	210
4.37.1	Installation	210
4.37.2	Select a Realm	210
4.37.3	Create a New Client	211
4.37.4	Add Users	211
4.37.5	Select or Create Roles	212
4.37.6	Configuring ClusterWare Software	212
4.37.6.1	Production Operations	213
4.37.7	User Management	213
4.37.8	Logging and Auditing	214
4.37.9	Access Token Lifespan	214
4.38	Integrating FreeIPA with ICE ClusterWare	214
4.38.1	Installation	214
4.38.2	Identify a Group for ClusterWare Users	215
4.38.3	Identify an Admin Account for Keycloak	215
4.38.4	Configure Keycloak	215
4.38.5	Verifying the Integration	216
4.39	Head Nodes Page	216
4.40	Important Files on Head Nodes	218
4.40.1	The ~/.scyl-dcw/ Folder	218
4.40.1.1	auth_tkt.cookie	218
4.40.1.2	logs/	219
4.40.1.3	workspace/	219
4.40.1.4	parse_failures/	219
4.40.2	The /opt/scyl-d/clusterware/ Folder	219
4.40.2.1	/opt/scyl-d/clusterware/bin/	219
4.40.2.2	/opt/scyl-d/clusterware/conf/	220

4.40.2.3	/opt/scyld/env/, modules/, and src/	220
4.40.2.4	/opt/scyld/clusterware/parse_failures/	220
4.40.2.5	/opt/scyld/clusterware/storage/	220
4.40.2.6	/opt/scyld/clusterware/workspace/	220
4.41	Managing Multiple Head Nodes	221
4.41.1	Adding a Head Node	221
4.41.1.1	Join a non-ClusterWare server	221
4.41.1.2	Join a ClusterWare head node	221
4.41.1.3	After a Join	222
4.41.1.4	Cleaning up From Join Failures	222
4.41.2	Removing a Joined Head Node	223
4.41.2.1	Peer Downloads	224
4.41.3	Booting With Multiple Head Nodes	225
4.42	headctl	226
4.43	Troubleshooting Head Nodes	226
4.43.1	Head Node Filesystem Is 100% Full	226
4.43.1.1	Verify Excessive Storage is Related to ClusterWare Software	227
4.43.1.2	Remove Unnecessary Objects from the ClusterWare Database	227
4.43.1.3	Investigate InfluxDB Retention of Telegraf Data	227
4.43.1.4	Remove Unnecessary Images and Repos	228
4.43.1.5	Move Large Directories	228
4.43.2	Head Nodes Disagree About Compute Node State	228
4.43.2.1	Head Node Failure	228
4.43.3	etcd Database Exceeds Size Limit	229
4.44	Networks Page	229
4.44.1	Create a Network	230
4.44.2	Edit Network	230
4.44.3	Delete Network	230
4.44.4	Related Links	231
4.45	Open Network Ports	231
4.46	Providing DHCP to Additional Interfaces	232
4.47	Exceeding System Limit of Network Connections	233
4.48	Managing Zero-Touch Provisioning (ZTP)	233
5	Articles	236
5.1	ICE ClusterWare Plugin System	236
5.1.1	Status Plugins	237
5.1.2	Hardware Plugins	239
5.1.3	Health-Check Plugins	240
5.1.4	Telegraf Plugins	241
5.1.5	Creating New Plugins	244
5.1.5.1	Creating Status Plugins	244
5.1.5.2	Creating Hardware Plugins	245
5.1.5.3	Creating Health-Check Plugins	245
5.1.5.4	Creating Telegraf Plugins	246
5.2	Using Ansible	247
5.2.1	Using Node Attributes with Ansible	250
5.2.2	Applying Ansible Playbooks to Images	250
5.3	Using Singularity	251
5.4	Using Docker for Compute Nodes	253
5.5	Using Kubernetes	254
5.5.1	Using a Single Non-ClusterWare System as a Control Plane	254
5.5.2	Using Multiple ClusterWare Nodes as a Control Plane	256
5.5.3	Using Multiple Non-ClusterWare Systems as a Control Plane	257

5.6	Creating Arbitrary Rocky Images	258
5.6.1	Using Version-Specific ISO File	259
5.6.2	Using Publicly Available Repositories	260
5.7	Creating Arbitrary RHEL Images	260
5.8	Creating Ubuntu and Debian Images	263
5.8.1	UBUNTU	263
5.8.2	DEBIAN	263
5.9	Converting CentOS 8 to Alternative Distro	263
5.10	Using Docker for Head Nodes	264
5.10.1	Install the Foundational Packages	264
5.10.2	Download and Load the ClusterWare Docker Image	265
5.10.3	Start the Container	265
5.10.4	Configure the Container	265
5.10.5	Stopping and Restarting the Container	267
5.10.6	The Container Storage Area	267
5.10.7	Known Issues	267
6	API Reference	269
6.1	Authentication	270
6.1.1	Username/Password Authentication	271
6.1.2	Token Refresh	272
6.1.3	Alternate Authentication Methods	272
6.2	Basic Operations	272
6.2.1	List Objects	273
6.2.2	Create New Object	273
6.2.3	Get Object Info	273
6.2.4	Update Object	274
6.2.5	Delete Object	274
6.2.6	Metadata Information	274
6.3	Admin Objects	275
6.3.1	Data Fields	275
6.3.2	Additional Endpoints	275
6.3.3	Example	276
6.4	Node Objects	277
6.4.1	Data Fields	277
6.4.2	Additional Endpoints	278
6.4.3	Example	280
6.5	Attribute-Group Objects	281
6.5.1	Data Fields	281
6.5.2	Additional Endpoints	282
6.5.3	Example	282
6.6	Boot Config Objects	284
6.6.1	Data Fields	284
6.6.2	Additional Endpoints	285
6.6.3	Example	286
6.7	Image Objects	287
6.7.1	Data Fields	287
6.7.2	Additional Endpoints	287
6.7.3	Example	288
6.8	Dynamic Group Objects	289
6.8.1	Data Fields	289
6.8.2	Additional Endpoints	289
6.8.3	Example	289
6.9	Naming Pool Objects	290

6.9.1	Data Fields	290
6.9.2	Additional Endpoints	291
6.9.3	Example	291
6.10	Software Repository Objects	292
6.10.1	Data Fields	292
6.10.2	Additional Endpoints	293
6.11	Software Distribution Objects	293
6.11.1	Data Fields	293
6.11.2	Additional Endpoints	293
6.12	State Set Objects	294
6.12.1	Data Fields	294
6.12.2	Additional Endpoints	294
6.12.3	Example	294
6.13	Network Objects	295
6.13.1	Data Fields	295
6.13.2	Additional Endpoints	296
6.14	Git Repository Objects	296
6.14.1	Data Fields	296
6.14.2	Additional Endpoints	297
6.15	Hostname Objects	298
6.15.1	Data Fields	298
6.15.2	Additional Endpoints	299
6.15.3	Example	299
6.16	Cluster-wide Endpoints	299
6.17	Head Node Endpoints	301
6.17.1	Data Fields	301
6.17.2	Additional Endpoints	301
6.18	Boot-time Support Endpoints	302
6.18.1	Client Download Endpoints	304
7	Release Notes, Changelog, and Known Issues	305
7.1	Release Notes	305
7.1.1	12.4.2 Release Notes	305
7.1.2	12.4.1 Release Notes	305
7.1.3	12.4.0 Release Notes	305
7.2	Changelog	306
7.2.1	12.4.2-g0000 - March 17, 2025	306
7.2.2	12.4.1-g0000 - March 7, 2025	306
7.2.3	12.4.0-g0000 - February 3, 2025	306
7.2.4	12.3.0-g0000 - October 4, 2024	308
7.2.5	12.2.0-g0000 - July 26, 2024	308
7.2.6	12.1.1-g0000 - January 23, 2024	309
7.2.7	12.1.0-g0000 - December 28, 2023	310
7.2.8	12.0.1-g0000 - July 24, 2023	311
7.2.9	12.0.0-g0000 - April 21, 2023	311
7.3	Known Issues And Workarounds	312
8	Frequently Asked Questions (FAQ)	314
8.1	Software Install/Update	314
8.2	Cluster Management	314
8.3	Manipulating Compute Node Images	315
8.4	Issues with Interacting with Compute Nodes	315
9	License Agreements	316

9.1	End-User License Agreement	316
9.2	Third-Party License Agreements	319
10	Feedback	326
10.1	Finding Further Information	326
10.2	Contacting Penguin Computing Support	326

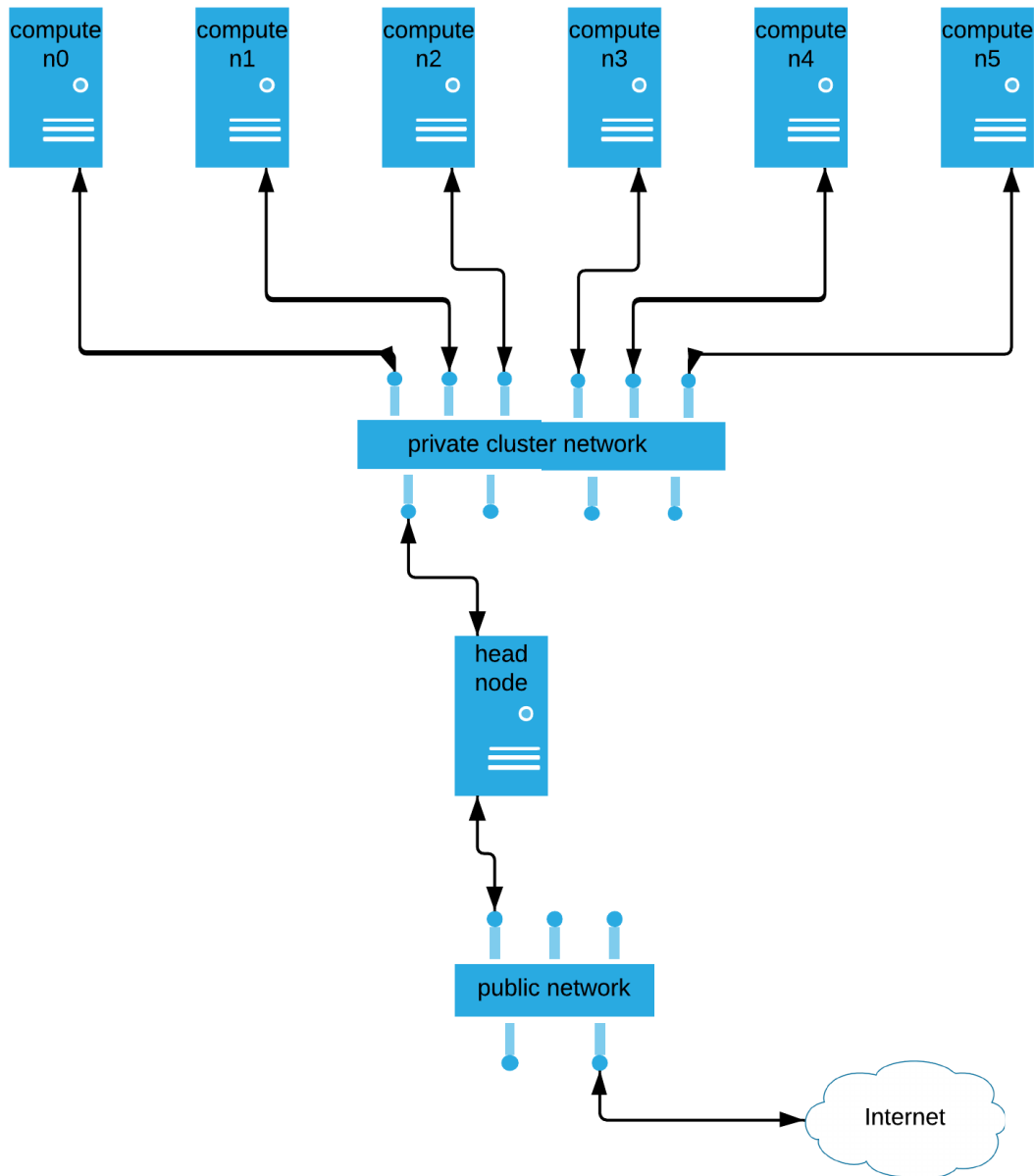
ICE CLUSTERWARE OVERVIEW

The ICE ClusterWare™ platform provides the tools (commonly named with prefix `scyld-`) and services (such as the key-value database) for a cluster administrator to install, administer, and monitor a Beowulf-style cluster. A cluster administrator commonly employs a shell on a head node to perform these functions. The ClusterWare platform additionally distributes packages for an administrator to install an optional job manager (for example, Slurm, OpenPBS, TORQUE), Kubernetes, and several varieties of OpenMPI-family software stacks for user applications. The *Administration* describes this with much greater detail.

1.1 Cluster Architecture Overview

A minimal ClusterWare cluster consists of a *head node* and one or more *compute nodes*, all interconnected via a *private cluster network*. User applications generally execute on the compute nodes, are often multithreaded across multiple compute nodes, and are usually coordinated by a job scheduler.

Basic ClusterWare Architecture



The head node is responsible for provisioning compute nodes, beginning with responding to a compute node's DHCP request for an IP address, and then (depending upon the compute node's BIOS settings) the compute node either boots from its local storage, or the compute node makes PXEboot requests for the kernel, initrd, and root filesystem images.

A ClusterWare head node usually also functions as a server for:

- The distributed *Key/Value Database*, which is implemented by the ClusterWare database and is accessed through

the REST API via command line tools or graphical user interface. It is the repository for information such as:

- The MAC address to IP address and node number mappings.
- The locations of the storage for the kernel, initrd, and root filesystem images.
- Compute node attributes, basic hardware and status, and configuration details.
- The storage for the images themselves.
- The compute node status information, which can be visualized by shell commands or by graphical tools.
- The compute node monitoring information, which is implemented by *Grafana*, *Telegraf*, and *InfluxDB*.
- Optional network storage, such as an NFS server.

A ClusterWare head node expects to execute in a Red Hat RHEL or CentOS 8.0 or later, Oracle Linux 8.3 to 8.6, or Rocky 8.4 or later environment.

Visit https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/ to view the Red Hat Enterprise Linux RHEL8 *Release Notes* and other useful documents, including the *Migration Planning Guide* and *System Administrator's Guide*.

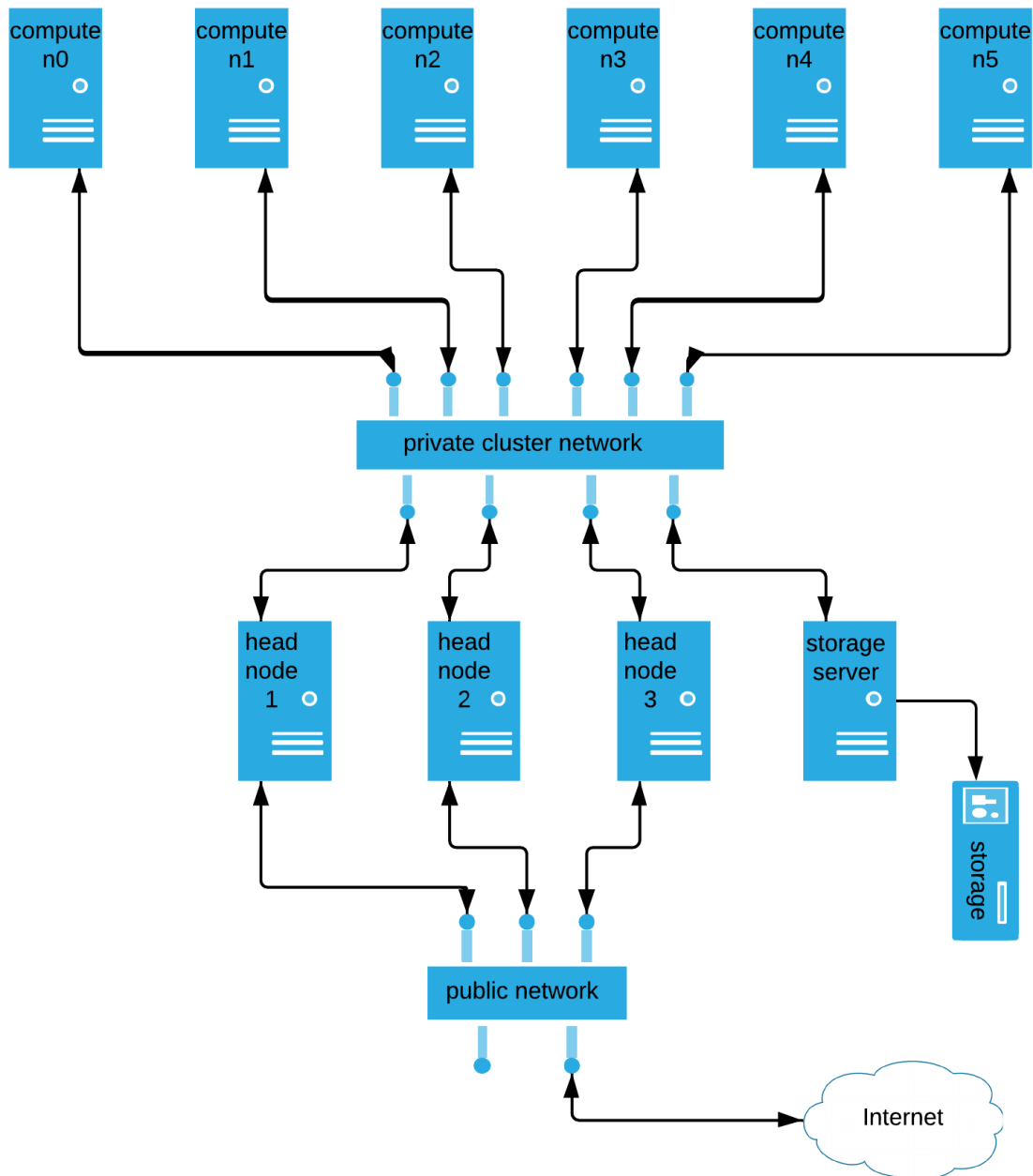
A more complex cluster can be High Availability (HA), consisting of multiple head nodes where each head node has access to the distributed database and shared image storage. In an active-active(-active....) relationship, each head node can manage any compute node, providing it with boot files and forwarding its status information into the shared database. Since no particular head node is specifically necessary to manage an individual compute node, then any head node can take over responsibility for the compute nodes that were previously communicating with a now-failed head node.

Note

Some network protocols, such as iSCSI, do not easily handle this sort of handoff, and any clusters using these protocols may experience additional difficulties on head node failure.

A complex cluster can also employ separate servers for the network storage, the compute node status information, and the boot images storage. For example:

Basic HA ClusterWare Architecture



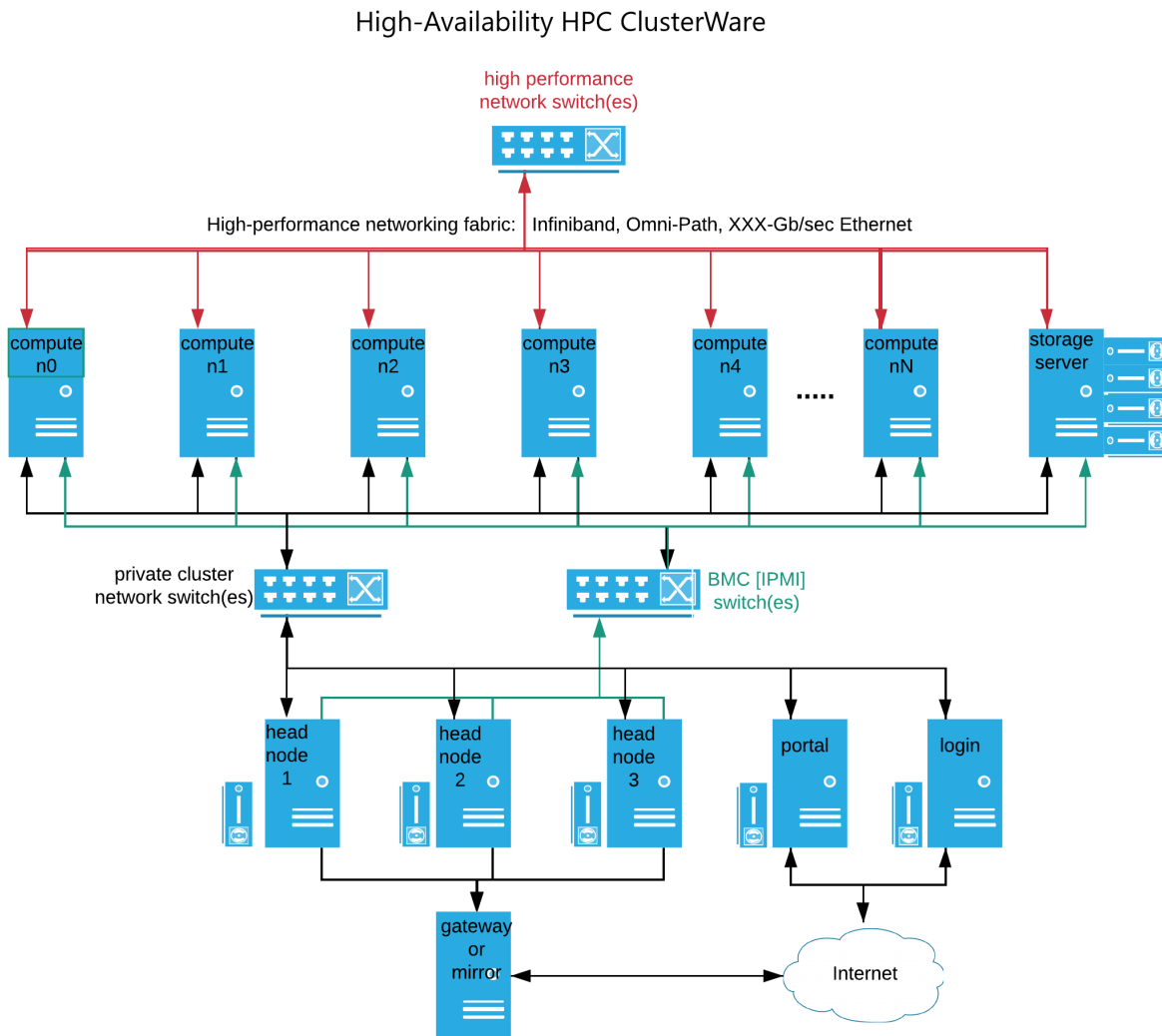
An even more complex cluster may employ high-performance networking, such as Infiniband, Omni-Path, or even 40GB/sec or faster Ethernet, in addition to the typical 1Gb/sec or 10Gb/sec Ethernet that commonly interconnects nodes on the private cluster network. This faster (and more expensive) network fabric typically interconnects the

compute nodes and commonly also shared cluster-wide storage.

The head node(s) commonly also have IPMI access to each compute node's Base Management Controller (BMC), which provides for command-line or programmatic access to the compute nodes at a more basic hardware level, allowing for remote control of power, forcing a reboot, viewing hardware state, and more.

Some complex clusters connect head nodes to the public internet via a gateway, for example, to allow a cluster administrator to use *yum* to install or update software from internet-accessible websites. Other complex clusters provide no head node access to the internet and keep software hosted on a cluster-internal mirror server, where the local cluster administrator has precise control over updates.

For example:



1.2 The ClusterWare Database

The ClusterWare database is stored as JSON content within a replicated document store distributed among the ClusterWare head nodes. This structure protects against the failure of any single head node.

The module API is intended to present a consistent experience regardless of the backend database, although some

details, such as failure modes, will differ.

The server side (head node) responses to specific steps in the PXE boot process are controlled by the cluster configuration stored as JSON documents (aka objects) in the database. The following sections will follow the order of the boot steps described above to explore the definition and use of these database objects.

Internally, database objects are identified by unique identifiers (UIDs). These UIDs are also used to identify objects in ClusterWare command line and GUI tools, although as these strings tend to be cumbersome, an administrator should also assign a name and an optional description to each object. Even when objects are listed by name, the UID is available in the *uid* field returned by the object query tools.

Database objects generally consist of name-value pairs arranged in a JSON dictionary and referred to here as *fields*. These fields can be set via using the `update` argument of the appropriate `scyld-*` command line tools or by editing object details through the GUI. Field names are all lower case with underscores separating words. Not all fields on all objects will be editable - for example, node names that are assigned based on the naming pool and node index.

Whenever a name-value pair is updated or added, a *last_modified* field in the mapping is also updated. These *last_modified* fields can be found scattered throughout the database objects.

1.3 Provisioning Compute Nodes

A principal responsibility of a head node is to provision compute nodes as they boot. A compute node's BIOS can be configured to boot from local storage (such as a harddrive) or to "PXEboot" by downloading the necessary images from a head node.

Each compute node is represented by a uniquely identified node object in the ClusterWare database. This object contains the basics of node configuration, including the node's index and the MAC address that is used to identify the node during the DHCP process. An administrator can also set an explicit IP address in the *ip* field. This IP address should be in the DHCP range configured during head node installation, although if none is specified, then a reasonable default will be selected based on the node index.

Each compute node is associated with a specific *boot configuration*, each stored in the ClusterWare database. A boot configuration ties together a *kernel* file, an *initramfs* file, and a *cmdline*, together with a reference to a root file system *rootfs* image. This *rootfs* is also known as a *boot image*, *root image*, or *node image*. A boot configuration also includes a configurable portion of the kernel command line that will be included in the iPXE boot script.

For a PXEboot, after the DHCP reply establishes the compute node's IP address, the node requests a loader program, and the ClusterWare head node responds by default with the Open Source iPXE loader, and a configuration file that identifies the *kernel* and *initramfs* images to download, and a *kernel command line* to pass to the booting kernel.

This kernel executes and initializes itself, then launches the *init* user program (provided by *dracut*), which in turn executes various scripts to initialize networking and other hardware, and eventually executes a ClusterWare *mount_rootfs* script, which downloads the *rootfs* image and sets up the node's root filesystem.

The *mount_rootfs* script may download and unpack a root filesystem image file, or alternatively may mount an iSCSI device or an image cached on a local harddrive, and then switch the node's root from the *initramfs* to this final root image. Other than when unpacking a root filesystem into RAM, images are shared and compute nodes are restricted to read-only access. In these cases compute nodes must use a writable overlay for modifiable portions of the file system. This is done toward the end of the *mount_rootfs* script via either the *rwtab* approach (for example, see https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/managing_file_systems/setting-read-only-permissions-for-the-root-file-system_managing-file-systems) or more commonly using an *overlayfs* (see <https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt>).

QUICKSTART

The following is a brief example of creating a minimal, yet functional, ICE ClusterWare™ cluster. No attempt is made here to explain the full breadth and depth of the ClusterWare platform, which is extensively discussed in the remainder of the documentation. This Quickstart assumes the reader is familiar with administering Red Hat RHEL or CentOS servers. For readers who are unfamiliar with clusters, see *Cluster Architecture Overview*.

2.1 Prerequisites

Prerequisites for this minimal cluster:

- A minimum of two x86_64 servers, and preferably three: one becomes a ClusterWare head node and the remainder become ClusterWare compute nodes. This Quickstart example uses three servers.
- The head node can be a bare metal server, although there is greater flexibility if it is a virtual machine. It should have a minimum of 4GB of RAM and 16GB of storage, and it must have an Ethernet controller connected to a private cluster network to communicate with the compute node(s). These are *minimal* requirements. See *Required and Recommended Components* for recommendations for a production cluster.
- The head node must be running Red Hat RHEL or CentOS 8.0 (or newer) or an equivalent distribution (see *Supported Distributions and Features*). It must have access to a repo for that base distribution so that the ClusterWare platform can `yum install` additional packages.
- If you do not already have a ClusterWare repo for ClusterWare packages, then the `scylld-install` installer prompts the user for an appropriate userid/password authentication and builds the ClusterWare repo in `/etc/yum.repos.d/clusterware.repo`. Typically access to the ClusterWare packages is through a second Ethernet controller connected to the "outside world", that is, the internet.
- The compute node(s) must have their BIOS configured to PXEboot by default, using either "Legacy" or "UEFI" mode. They should have a minimum of 4GB of RAM and one Ethernet controller that is also physically connected to the same private cluster network. See *Required and Recommended Components* for recommendations for a production cluster.

2.2 Create Administrator

A ClusterWare cluster administrator needs root privileges. Common practice is to create non-root administrators and give them `sudo` capability. For example, create an administrator user `admin1`:

```
useradd admin1      # create the user
passwd admin1      # and give it a password

# Give "admin1" full root sudo privileges
echo "admin1 ALL=(root) NOPASSWD: ALL" >> /etc/sudoers
```

(continues on next page)

(continued from previous page)

```
# Now execute as that user "admin1"
su - admin1
# And add SSH key pairs (defaulting to /home/admin1/.ssh/id_rsa)
ssh-keygen
```

2.3 Install ClusterWare

Create a cluster configuration text file that names the interface to the private cluster network, the IP address on that network for the first compute node, and a list of MAC addresses to use as compute nodes. For example:

```
cat <<-EOF >/tmp/cluster-conf
iprange 10.54.60.0          # starting IP address of node 0
node 52:54:00:a6:f3:3c     # node 0 MAC address
node 40:8d:5c:fa:ea:c3     # node 1 MAC address
EOF
```

Now install the *clusterware-installer* package using the ClusterWare `/etc/yum.repos.d/clusterware.repo` repo file:

```
sudo yum install clusterware-installer
```

That package contains the `scyld-install` script that you execute to install the software and create the *DefaultImage*, which consists of a basic compute node file system image, and the *DefaultBoot* config file, which references that *DefaultImage* and contains various boot-time information such as a kernel commandline to pass to a booting node.

For a simple installation:

```
# Reminder: you should be executing as user "admin1"
scyld-install --config /tmp/cluster-conf
```

By default the *DefaultImage* contains a kernel and roots software from the same base distribution installed on the head node, although if the head node executes RHEL8, then no *DefaultImage* and *DefaultBoot* are created.

Alternatively, for more flexibility (especially with a RHEL8 head node), execute the installer with an additional option that identifies the base distribution to be used for the *DefaultImage*:

```
scyld-install --config /tmp/cluster-conf --os-iso <ISO-file>
```

where `<ISO-file>` is either a pathname to an ISO file or a URL of an ISO file of a specific base distribution release, for example, `--os-iso rhel-8.5-x86_64-dvd.iso`. That ISO can match the head node's base distribution or can be any distribution supported by Penguin Computing (see *Supported Distributions and Features*).

Now you have a basic 2-node cluster that should PXEboot compute nodes. The installer has created a *DefaultImage* that contains basic compute node software and a *DefaultBoot* config file for booting that image, and has initialized every node to PXEboot using the *DefaultBoot*. Validate your current setup by rebooting both compute nodes, and check the status of the nodes as they boot and connect to the head node:

```
scyld-nodectl status --refresh
# Use ctrl-c to exit this display
```

which initially shows:

```
Node status [ date & time ]
-----
n[0-1] new
```

for the nodes *n0* and *n1*, and automatically updates as each node's status changes from *booting* to *up*. The per-node transition from *new* to *booting* consumes a minute or more doing hardware initialization, PXEboot provisioning, and early software init. The transition from *booting* to *up* consumes another minute or more. If the nodes do not boot, then see [Failing PXE Network Boot](#).

You can view information about the *up* nodes by executing:

```
scyld-nodectl ls -L
# which is shorthand for `scyld-nodectl list --long-long`

scyld-nodectl status -L
```

2.4 Configure Boot Image and Job Scheduler

Now enhance the functionality of the compute node software by installing the Slurm job scheduler and an OpenMPI software stack into the image that PXEboots. Best practice is to retain the original *DefaultImage* and *DefaultBoot* as a pristine starting point for future additional software enhancements, so copy these *Default* objects and modify just the copies:

```
scyld-imgctl -i DefaultImage clone name=NewImage
scyld-bootctl -i DefaultBoot clone name=NewBoot
# The NewBoot clone is initially associated with the DefaultImage,
# so change that:
scyld-bootctl -i NewBoot update image=NewImage
# Instruct all compute nodes to use "NewBoot" (instead of "DefaultBoot"):
scyld-nodectl --all set _boot_config=NewBoot
```

Add the head node to `/etc/hosts` and then restart the `clusterware-dnsmasq` service. Suppose the head node's private cluster network IP address is "10.54.0.60":

```
echo "10.54.0.60 $(hostname)" | sudo tee -a /etc/hosts
sudo systemctl restart clusterware-dnsmasq
```

Now install and configure Slurm, which is one of the ClusterWare-supported job schedulers. The `scyld-install` installer has disabled the ClusterWare repo (for an explanation, see [Additional Software](#)) so we must explicitly enable the repo:

```
sudo yum install slurm-scyld --enablerepo=scyld*

# Perform the Slurm initialization
slurm-scyld.setup init

# Add Slurm client software to the NewImage
slurm-scyld.setup update-image NewImage

# Reboot the nodes and view their status as they boot
scyld-nodectl --all reboot
scyld-nodectl status --refresh
# And ctrl-c when both rebooting nodes are again "up"
```

(continues on next page)

(continued from previous page)

```
# Check the job scheduler status
slurm-scyld.setup status
# If the Slurm daemon and munge are not both executing, then:
slurm-scyld.setup cluster-restart
# And check status again
slurm-scyld.setup status
sinfo
```

Configure the cluster to support OpenMPI multi-threaded communication between compute nodes using the `ssh` transport mechanism, which requires user `uid/gid` and passphrase-less key-based access. For this Quickstart we will continue to use `admin1` as the user. Add `admin1`'s authentication to the *NewImage*:

```
/opt/scyld/clusterware-tools/bin/sync-uids \
    -i NewImage --create-homes \
    --users admin1 --sync-key admin1=/home/admin1/.ssh/id_rsa.pub
```

Install OpenMPI 4.0 into *NewImage* using `chroot`:

```
scyld-modimg -i NewImage --chroot --no-discard --overwrite --upload
# Inside the chroot you are executing as user root
yum install openmpi4.0

# Set up access to Slurm and OpenMPI for "admin1"
echo "module load slurm" >> /home/admin1/.bashrc
echo "module load openmpi" >> /home/admin1/.bashrc

# Build an example OpenMPI application
cd /opt/scyld/openmpi/*/gnu/examples
yum install make
module load openmpi
make hello_c
# For simplicity, copy the executable to /home/admin1/hello_c
cp hello_c /home/admin1/hello_c

exit # from the chroot
```

Reboot the nodes with the updated *NewImage*:

```
scyld-nodectl --all reboot
# Observe the node status changes
scyld-nodectl status --refresh
# And ctrl-c when both rebooting nodes are again "up"
```

From the head node, verify this by using Slurm to execute programs on the compute nodes:

```
module load slurm

# Verify basic Slurm functionality by executing a simple command on each node
srun -N 2 hostname

# Use Slurm to execute one "Hello World" program on each of the two nodes
srun -N 2 hello_c
```

INSTALL

The *Overview* describes the ICE ClusterWare™ system architecture and design and basic terminology necessary to properly configure and administer a ClusterWare cluster.

This Install Guide is intended for use by ClusterWare administrators. As is typical for any Linux-based system, the administrator must have root privileges (if only via *sudo*) to perform many of the tasks described in this document.

This guide provides specific information about tools and methods for setting up the cluster, security considerations, and optional tools that can be useful in administrating your cluster.

This guide is written with the assumption that the administrator has a background in a Unix or Linux operating environment; therefore, the document does not cover basic Linux system administration. If you do not have sufficient knowledge for using or administering a Linux system, we recommend that you first study other resources, either in print or online.

When appropriate, this document refers the reader to other parts of the documentation set for more detailed explanations for various topics, such as the *Administration* Guide, which provides greater details about cluster administration, commands, and GUI features.

3.1 Supported Distributions and Features

Unless otherwise noted, the ICE ClusterWare™ platform is principally supported for the x86_64 architecture.

It has been additionally tested on the aarch64 architecture using Rocky 8.5. If you are interested in a cluster using that architecture or a mix of architectures, please contact Penguin Computing.

Entires marked as **probable** are RHEL clones and probably work, although they are not explicitly tested by Penguin Computing.

PENGUIN-VERIFIED DISTROS		Head Nodes	Compute Nodes		
Distro	Version		Node Image	Kickstart	Local Install
RHEL/CentOS	7.0 - 7.9 ¹	no	yes	yes	yes
RHEL	8.0 - 8.10	yes	yes	yes	yes
RHEL	9.0 - 9.5	yes	yes	yes	yes
CentOS	8.0 - 8.5 ²	yes	yes	yes	yes
Rocky	8.4 - 8.10	yes	yes	yes	yes
Rocky	9.0 - 9.5	yes	yes	yes	yes
CentOS Stream	8 ³	yes	yes	yes	yes
CentOS Stream	9 ⁴	yes	yes	yes	yes
Oracle	8.3 - 8.6	probable	probable	yes	yes
AlmaLinux	8.4 - 8.7	probable	probable	yes	yes
AlmaLinux	9.0 - 9.1	probable	probable	yes	yes
OpenSUSE	Leap 15.2 - 15.4	no	yes	yes	yes
Ubuntu	18 - 24 LTS	no	yes	no	yes
Debian	stable, testing	no	yes	no	yes

Footnotes

- [1] Since RHEL / CentOS 7 have reached end of maintenance they are no longer supported as head nodes. These versions are still temporarily supported as compute node images.
- [2] CentOS 8 can be converted to an alternative distro. See "Appendix: Converting CentOS 8 to Alternative Distro".
- [3] CentOS Stream 8 was confirmed supported as of December 26, 2023.
- [4] CentOS Stream 9 was confirmed supported as of December 26, 2023.

RHEL9-clone compute node boot images **cannot** be built by RHEL7-clone head nodes.

Entries marked **both** indicate that Penguin Computing tests and supports both SELinux *Targeted* and *MLS* policies.

PENGUIN-VERIFIED SECURITY		Head Nodes FIPS Mode	SELinux	Compute Nodes	
OS Distro	Version(s)			FIPS Mode	SELinux
RHEL/CentOS	7.0 - 7.5	no	no	yes	both
RHEL/CentOS	7.6 - 7.9	yes	both	yes	both
RHEL	8.0 - 8.9	yes	both	yes	both
RHEL	9.0 - 9.3	yes	both	yes	both
CentOS	8.0 - 8.7 ¹	yes	both	yes	both
Rocky	8.4 - 8.9	yes	both	yes	both
Rocky	9.0 - 9.3	yes	both	yes	both
CentOS Stream	8 ²	probable	probable	yes	both
CentOS Stream	9 ³	probable	probable	yes	both
Oracle	7.9	probable	probable	yes	both
Oracle	8.3 - 8.7	probable	probable	yes	both
AlmaLinux	8.4 - 8.7	probable	probable	yes	both
AlmaLinux	9.0 - 9.1	probable	probable	yes	both

Footnotes

- [1] CentOS 8 can be converted to an alternative distro. See "Appendix: Converting CentOS 8 to Alternative Distro".

(continues on next page)

(continued from previous page)

- [2] CentOS Stream 8 was confirmed supported as of December 26, 2023.
 [3] CentOS Stream 9 was confirmed supported as of December 26, 2023.

CLUSTERWARE-DISTRIBUTED SCHEDULERS			
OS Distro	PBS TORQUE	Slurm	OpenPBS
RHEL/CentOS 6			
RHEL/CentOS 7	6	18-23	
RHEL/CentOS 8		20-23	20
RHEL/CentOS 9		20-23	20

CLUSTERWARE-DISTRIBUTED MIDDLEWARE			
OS Distro	OpenMPI	MPICH	MVAPICH
RHEL/CentOS 6			
RHEL/CentOS 7	1, 2, 3, 4	4.1	2.3
RHEL/CentOS 8	3, 4	4.1	2.3
RHEL/CentOS 9	4	4.1	2.3

3.2 Required and Recommended Components

ICE ClusterWare™ head nodes are expected to use x86_64 processors running a Red Hat RHEL, Rocky, or similar distribution. See *Supported Distributions and Features* for specifics.

Important

ClusterWare head nodes currently require a Red Hat RHEL or Rocky 8.4 (or later) or CentOS Stream 8 (or later) base distribution environment due to dependencies on newer *selinux* packages. This requirement only applies to head nodes, not compute nodes.

Important

By design, ClusterWare compute nodes handle DHCP responses on the private cluster network (bootnet) by employing the base distribution's facilities, including *NetworkManager*. If your cluster installs a network file system or other software that disables this base distribution functionality, then *dhclient* or custom static IP addresses, and potentially additional workarounds, must be configured.

ClusterWare head nodes should ideally be "lightweight" for simplicity and contain only software that is needed for the local cluster configuration. Non-root users typically do not have direct access to head nodes and do not execute applications on head nodes.

Head node components for a production cluster:

- x86_64 processor(s) are required, with a minimum of four cores recommended.
- 8GB RAM (minimum) is recommended.
- 100GB fast storage (minimum) is recommended. All storage should be backed by NVMe or other performant technology.

The largest storage consumption contains packed images, uploaded ISOs, et al. Its location is set in the file `/opt/scyld/clusterware/conf/base.ini` and defaults to `/opt/scyld/clusterware/storage/`.

The directory `/opt/scyld/clusterware/git/cache/` consumes storage roughly the size of the git repos hosted by the system.

Other than the above `storage/` and `cache/`, the directory `/opt/scyld/` consumes roughly 300MB.

Each administrator's `~/ .scyldcw/workspace/` directory contains unpacked images that have been downloaded by an administrator for modification or viewing.

- One Ethernet controller (required) that connects to the private cluster network which interconnects the head node(s) with all compute nodes.
- A second Ethernet controller (recommended) that connects a head node to the Internet.

Multiple Ethernet or other high-performance network controllers (for example, Infiniband, Omni-Path) are common on the compute nodes, but do not need to be accessible by the head node(s).

We recommend employing virtual machines, hosted by "bare metal" hypervisors, for head nodes, login nodes, job scheduler servers, etc., for ease of management. Virtual machines are easy to resize and easy to migrate between hypervisors. See https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/ for basic Red Hat documentation.

Note

A bare metal hypervisor host must contain the aggregated resources required by each hosted virtual server, and ideally the aggregated recommended resources, plus several additional CPUs/cores and RAM resources devoted to the hypervisor functionality itself.

Note

The `nmcli` connection add tool can be used to create network bridges and to add physical interfaces to those newly created bridges. Once appropriate bridges exist, the `virt-install` command can attach the virtual interfaces to the bridges, so that the created virtual machines exist on the same networks as the physical interfaces on the hypervisor.

A High Availability ("HA") cluster requires a minimum of three "production" head nodes, each a virtual machine hosted on a different bare metal hypervisor. Even if an HA cluster is not required, we recommend a minimum of two head nodes - one functioning as the production head node, and the other as a development head node that can be used to test software updates and configuration changes prior to updating the production node to the validated final updates.

Compute nodes are generally bare metal servers for optimal performance. See *Supported Distributions and Features* for a list of supported distributions.

See *ICE ClusterWare Overview* for more details.

3.3 Install ICE ClusterWare

The ICE ClusterWare™ `scyld-install` script installs the necessary packages from the ClusterWare yum repositories, and installs dependency packages as needed from the base distribution (for example, Red Hat RHEL or Rocky) yum repositories.

Important

Do not install the ClusterWare platform as an upgrade to an existing ClusterWare 6 or 7 installation. Instead, install the ClusterWare platform on a non-ClusterWare system that ideally is a virtual machine. (See *Required and Recommended Components*.)

Important

The head node(s) must use a Red Hat RHEL- or CentOS-equivalent base distribution release 8.4 or later environment, due to dependencies on newer *selinux* packages.

Note

Clusters commonly employ multiple head nodes. The instructions in this section describe installing the ClusterWare platform on the first head node. To later install on additional head nodes, see *Managing Multiple Head Nodes*.

`scylld-install` anticipates being potentially executed by a non-root user, so ensure that your userid can execute `sudo`. Additionally, if using `sudo` behind a proxy, then because `sudo` clears certain environment variables for security purposes, the cluster administrator should consider adding several lines to `/etc/sudoers`:

```
Defaults    env_keep += "HTTP_PROXY http_proxy"
Defaults    env_keep += "HTTPS_PROXY https_proxy"
Defaults    env_keep += "NO_PROXY no_proxy"
```

Important

Various commands that manipulate images execute as user root, thereby requiring that the commands internally use `sudo` and requiring that user root must have access to the administrator's workspace which contains the administrator's images. Typically the per-user workspace is `~/.scylldcw/workspace/`. If that directory is not accessible to the command executing as root, then another accessible directory can be employed, and the administrator can identify that alternative pathname by adding a `modimg.workspace` setting to `~/.scylldcw/settings.ini`.

Important

`scylld-install` uses the `yum` command to access the ClusterWare platform and potentially various other repositories (for example, Red Hat RHEL or Rocky) that by default normally reside on Internet websites. However, if the head node(s) do not have Internet access, then the required repositories must reside on local storage that is accessible by the head node(s). See *Creating Local Repositories without Internet*.

3.3.1 Download the ICE ClusterWare Install Script and Related Files

Most commonly, first download a ICE ClusterWare™ yum repo configuration file that is already customized for your cluster, containing an appropriate *authentication token* granting access to the various ClusterWare yum repo directories. That *authentication token* is the cluster serial number provided by Penguin Computing.

- Login to the Penguin Computing Support Portal at <https://www.penguinsolutions.com/computing/support/technical-support/>.

- Click on the *Assets* tab, and then select a specific *Asset Name*.
- In the *Asset Detail* section, click on *YUM Repo File*, which downloads an asset-specific `clusterware.repo` file.
- Move that downloaded file to `/etc/yum.repos.d/clusterware.repo`.
- Verify `clusterware.repo` permissions and ownership by installing the `clusterware-installer` package, which contains the `scyld-install` script.

For example:

```
cd /tmp
# Expecting the desired clusterware.repo file to now reside in /tmp
sudo chmod 644 clusterware.repo
sudo chown root:root clusterware.repo
sudo cp -a clusterware.repo /etc/yum.repos.d/clusterware.repo
sudo yum install clusterware-installer
```

Alternatively, if Penguin Computing has transmitted (e.g., by email) a custom `clusterware.repo` file to you, then as described above, move that file to `/etc/yum.repos.d/clusterware.repo`, install the `clusterware-installer` RPM, and then execute the `/usr/bin/scyld-install` script contained in that RPM.

Less commonly, download the `scyld-install` script directly from the Penguin Computing yum repository. When executed, that script queries the user for the appropriate *authentication token* (cluster serial number) provided by Penguin Computing, and uses that to create an appropriate `/etc/yum.repos.d/clusterware.repo`. For example, download and prepare the `scyld-install` script:

```
cd /tmp
wget https://updates.penguincomputing.com/clusterware/12/installer/scyld-install
# or download with *curl* or equivalent
chmod +x scyld-install
```

3.3.2 Execute the ICE ClusterWare Install Script

If `/etc/yum.repos.d/clusterware.repo` exists, then `scyld-install`'s subsequent invocations of `yum` will employ that configuration file. If `/etc/yum.repos.d/clusterware.repo` does not exist, then `scyld-install` prompts the user for an appropriate *authentication token* and uses that to build a `/etc/yum.repos.d/clusterware.repo` that is customized to your cluster.

`scyld-install` accepts an optional argument specifying a cluster configuration file that contains information necessary to set up the DHCP server. For example:

```
cat <<-EOF >/tmp/cluster-conf
interface enp0s9          # names the private cluster interface
nodes 4                  # max number of compute nodes
iprange 10.10.32.45      # starting IP address of node 0
node 08:00:27:f0:44:35   # node 0 MAC address
node 08:00:27:f0:44:45   # node 1 MAC address
node 08:00:27:f0:44:55   # node 2 MAC address
node 08:00:27:f0:44:65   # node 3 MAC address
EOF
```

where the syntax of this cluster configuration file is:

`domain <DOMAIN_NAME>`

Optional. Defaults to "cluster.local".

interface <INTERFACE_NAME>

Optional. Specifies the name of head node's interface to the private cluster network, although that can be determined from the specification of the <FIRST_IP> in the *iprange* line.

nodes <MAX_COUNT>

Optional. Specifies the max number of compute nodes, although that can be determined from the *iprange* if both the <FIRST_IP> and <LAST_IP> are present. The max will also adjust as-needed if and when additional nodes are defined. For example, see *Node Creation with Known MAC address(es)*.

iprange <FIRST_IP> [<LAST_IP>]

Specifies the IP address of the first node (which defaults to n0) and optionally the IP address of the last node. The <LAST_IP> can be deduced from the <FIRST_IP> and the *nodes* <MAX_COUNT>. The <FIRST_IP> can include an optional netmask via a suffix of /<BIT_COUNT> (e.g., /24) or a mask (e.g., /255.255.255.0).

<FIRST_INDEX> <FIRST_IP> [<LAST_IP>] [via <FROM_IP>] [gw <GATEWAY_IP>]

This is a more elaborate specification of a range of IP addresses, and it is common when using DHCP relays or multiple subnets. <FIRST_INDEX> specifies that the first node in this range is node n<FIRST_INDEX> and is assigned IP address <FIRST_IP>; optionally specifies that the range of nodes make DHCP client requests that arrive on the interface that contains <FROM_IP>; optionally specifies that each DHCP'ing node be told to use <GATEWAY_IP> as their gateway, which otherwise defaults to the IP address (on the private cluster network) of the head node.

For example: `128 10.10.24.30/24 10.10.24.100 via 192.168.65.2 gw 10.10.24.254` defines a DHCP range of 71 addresses, the first starting with 10.10.24.30, and assigns the first node in the range as n128; watches for DHCP requests arriving on the interface containing 192.168.65.2; and tells these nodes to use 10.10.24.254 as the their gateway.

node [<INDEX>] <MAC> [<MAC>]

One compute node per line, and commonly consisting of multiple *node* lines, where each DHCP'ing node is recognized by its unique MAC address and is assigned an IP address using the configuration file specifications described above. Currently only the first <MAC> is used. An optional <INDEX> is the index number of the node that overrides the default of sequentially increasing node number indices and thereby creates a gap of unassigned indices. For example, a series of eight *node* lines without an <INDEX> that is followed by `node 32 52:54:00:c4:f7:1e` creates a gap of unassigned indices n8 to n31 and assigns this node as n32.

Note

ICE ClusterWare™ yum repositories contain RPMs that duplicate various Red Hat EPEL RPMs, and these ClusterWare RPMs get installed or updated in preference to their EPEL equivalents, even if `/etc/yum.repos.d/` contains an EPEL .conf file.

Note

The ClusterWare platform employs `userid/groupid 539` to simplify communication between the head node(s) and the backend shared storage where it stores node image files, kernels, and `initramfs` files. If the `scyl-d-install` script detects that this `uid/gid` is already in use by other software, then the script issues a warning and chooses an alternative new random `uid/gid`. The cluster administrator needs to set the appropriate permissions on that shared storage to allow all head nodes to read and write all files.

The ClusterWare database is stored as JSON content within a replicated document store distributed among the ClusterWare head nodes. This structure protects against the failure of any single head node.

For example, using the `cluster-config` created above, install the ClusterWare platform from a yum repo:

```
scyld-install --config /tmp/cluster-conf
```

By default `scyld-install` creates the *DefaultImage* that contains a kernel and rootfs software from the same base distribution installed on the head node, although if the head node executes RHEL8, then no *DefaultImage* and *DefaultBoot* are created.

Alternatively, for more flexibility (especially with a RHEL8 head node), execute the installer with an additional option that identifies the base distribution to be used for the *DefaultImage*:

```
scyld-install --config /tmp/cluster-conf --os-iso <ISO-file>
```

where `<ISO-file>` is either a pathname to an ISO file or a URL of an ISO file. That ISO can match the head node's distribution or can be any supported distribution.

`scyld-install` unpacks an embedded compressed payload and performs the following steps:

- Checks for a possible newer version of the *clusterware-installer* RPM. If one is found, then the script will update the local RPM installation and execute the newer `scyld-install` script with the same arguments. An optional argument `--skip-version-check` bypasses this check.
- An optional argument `--yum-repo /tmp/clusterware.repo` re-installs a yum repo file to `/etc/yum.repos.d/clusterware.repo`. This is unnecessary if `/etc/yum.repos.d/clusterware.repo` already exists and is adequate.
- Checks whether the *clusterware* RPM is installed.
- Confirms the system meets various minimum requirements.
- Installs the *clusterware* RPM and its supporting RPMs.
- Copies a customized *Telegraf* configuration file to `/etc/telegraf/telegraf.conf`
- Enables the `tftpd` service in `xinetd` for PXE booting.
- Randomizes assorted security-related values in `/opt/scyld/clusterware/conf/base.ini`
- Sets the current user account as a ClusterWare administrator in `/opt/scyld/clusterware/conf/base.ini`. If this is intended to be a production cluster, then the system administrator should create additional ClusterWare administrator accounts and clear this variable. For details on this and other security related settings, including adding ssh keys to compute nodes, please see [Securing the Cluster](#).
- Modifies `/etc/yum.repos.d/clusterware.repo` to change `enabled=1` to `enabled=0`. Subsequent executions of `scyld-install` to update the ClusterWare platform will temporarily (and silently) re-enable the ClusterWare repo for the duration of that command. This is done to avoid inadvertent updates of ClusterWare packages if and when the clusterware administrator executes a more general `yum install` or `yum update` intending to add or update the base distribution packages.

Then `scyld-install` uses `systemd` to enable and start `firewalld`, and opens ports for communication between head nodes as required by `etcd`. See [Services](#), [Ports](#), [Protocols](#) for details.

Once the ports are open, `scyld-install` initializes the ClusterWare database and enables and starts the following services:

- *httpd*: The Apache HTTP daemon that runs the ClusterWare service and proxies *Grafana*.
- *xinetd*: Provides network access to *tftp* for PXE booting.
- *telegraf*: Collects head node performance data and transmits to telegraf-relay service.

- *telegraf-relay*: Forwards telegraf data to *InfluxDB* and to telegraf-relay services running on head nodes
- *influxdb*: Stores node performance and status data for visualization in *Grafana*.
- *grafana-server*: Displays the head node and compute node status data through a web interface.

The script then:

- Opens ports in `firewalld` for public access to HTTP, HTTPS, TFTP, iSCSI, and incoming *Telegraf* UDP messages.

Note

UDP message sending is deprecated as of the ClusterWare 12.4.0 release.

- Installs and configures the cluster administrator's *clusterware-tools* package (unless it was executed with the `--no_tools` option).
- Configures the cluster administrator's `~/scylcdc/settings.ini` to access the newly installed ClusterWare service using the `scyld-tool-config` tool.
- Creates an initial simple boot image *DefaultImage*, boot config *DefaultBoot*, and attributes *DefaultAttribs* using the `scyld-add-boot-config` tool.
- Loads the cluster configuration specified on the command line using the `scyld-cluster-conf load` command.
- Restarts the *httpd* service to apply the loaded cluster configuration.

Important

See the *Boot Configurations* for details about how to modify existing boot images, create new boot images, and associate specific boot images and attributes with specific compute nodes. We strongly recommend not modifying or removing the initial *DefaultImage*, but rather cloning that basic image into a new image that gets modified further, or just creating new images from scratch.

Important

If you wish to ensure that the latest packages are installed in the image after the `scyld-install`, then execute `scyld-modimg -i DefaultImage --update --overwrite --upload`.

Important

See *Common Additional Configuration* for additional optional cluster configuration procedures, e.g., installing and configuring a job scheduler, installing and configuring one of the MPI family software stacks.

Important

If this initial `scyld-install` does not complete successfully, or if you want to begin the installation anew, then when/if you re-run the script, you should cleanse the partial, potentially flawed installation by adding the `--clear` argument, e.g., `scyld-install --clear --config /tmp/cluster-conf`. If that still isn't sufficient, then


```
scyld-install --clear-all --config /tmp/cluster-conf
```

does a more complete clearing, then reinstalls all the ClusterWare packages.

Due to licensing restrictions, when running on a Red Hat RHEL system, the installer will still initially create a Rocky compute node image as the DefaultImage. If after this initial installation a cluster administrator wishes to instead create compute node images based on RHEL, then use the `scyld-clusterctl repos` tool as described in *Creating Arbitrary RHEL Images*, and create a new image (e.g., *DefaultRHELimage*) to use as a new default.

3.4 scyld-install

NAME

scyld-install -- Tool to install the ClusterWare platform and perform initial basic configuration of a head node, and to update an existing head node installation.

USAGE

scyld-install

```
[-h] [-v] [--config CONF_FILE] [--token TOKEN] [--dnf-repo REPO_FILE] [--yum-repo
REPO_FILE] [-u | --update] [-l | --load DATABASE_FILE] [-s | --save DATABASE_FILE]
[--without-files] [--iso PATH] [--os-iso PATH] [--clear] [--clear-all] [--no-tools]
[--join HEAD_IP] [--skip-version-check] [--database-passwd PASSWD] [--non-interactive]
```

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- config CONF_FILE** Specify a cluster configuration file to load and to initialize the DHCP server for private cluster network.
- token TOKEN** Specify a cluster serial number or other authentication to use in the yum repository file.
- dnf-repo REPO_FILE** Provide a complete dnf repository file.
- yum-repo REPO_FILE** Provide a complete yum repository file for the ClusterWare platform. Alias for --dnf-repo.
- u, --update** If the ClusterWare platform is already installed, then by default `scyld-install` asks for a confirmation that the intention is to update software, not to perform a new install. This optional argument explicitly directs `scyld-install` to update the ClusterWare installation.

DATABASE LOAD/SAVE OPTIONS

- l, --load DATABASE_FILE** Load the ClusterWare database with the specified *DATABASE_FILE*.
- s, --save DATABASE_FILE** Save the ClusterWare database to the specified *DATABASE_FILE*.
- without-files** Do NOT include the contents of images and boot files when loading or saving.

ADVANCED OPTIONS

- iso PATH** Install or update the ClusterWare platform using the named *PATH*, which is either the path of a ClusterWare ISO file or the URL of a remote ClusterWare ISO file.
- os-iso PATH** Create the DefaultImage and DefaultBoot using the named *PATH*, which is either the path of a base distribution ISO file or the URL of a remote base distribution ISO file.

- clear** THIS IS DEPRECATED - PLEASE USE `--clear-all`.
- clear-all** Clear the ClusterWare database, which **DELETES ALL IMAGES AND BOOT CONFIGURATIONS!** Remove all ClusterWare RPMs, except for *clusterware-installer*) and *libcouchbase*. Delete directories `/opt/couchbase/`, `/opt/scyld/clusterware*/`, and `/var/log/clusterware/`, and delete root's `~/ .scyldcw/` and current admin's `~/ .scyldcw/` (but not any other admin's `~/ .scyldcw/`) for everything except logs, then optionally reinstall the ClusterWare platform.
- no-tools** Don't install the ClusterWare tools. The default is to install the tools.
- join EXISTING_HEAD_IP** Join this head node to the *EXISTING_HEAD_IP* IP address of an existing head node.
- skip-version-check** Use this installer and skip the online checking for a newer version.
- database-passwd PASSWD** Specify the database administrative password. Warning: influxdb2 requires a minimum of 8 characters.
- reconfigure** During an update, most steps that alter the head node OS will be skipped by default, but this option overrides and updates the base distribution.
- non-interactive** Execute the installer non-interactively, choosing default answers to the interactive questions in a way that most users would do. This is appropriate for using the installer in a script.

EXAMPLES

```
scyld-install --clear
```

Clear the database, leaving it empty, and undo any existing ClusterWare installation.

```
scyld-install --clear --config cluster-conf
```

Clear the database, leaving it empty, and undo any existing ClusterWare installation, then reset the database to the specified cluster-conf parameters.

RETURN VALUES

Upon successful completion, **scyld-install** returns 0. On failure, an error message is printed to `stderr` and **scyld-install** returns 1.

3.5 scyld-tool-config

NAME

scyld-tool-config -- "Command line tool" for the ClusterWare platform

USAGE

scyld-tool-config

```
[-h] [-v] [-q] [-c | --config CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]  
[--yes] [--example]
```

DESCRIPTION

The generic command line tool for the ClusterWare platform.

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.

- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.
- yes** Answer yes or to defaults to all questions.
- example** Generate an example file (implies **--yes**).

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]**
Masquerade as user *USER* with optional colon-separated password *PASSWD*.

EXAMPLES

RETURN VALUES

Upon successful completion, **scyld-tool-config** returns 0. On failure, an error message is printed to `stderr` and **scyld-tool-config** returns 1.

3.6 scyld-cluster-conf

NAME

scyld-cluster-conf -- load or save the cluster configuration file.

USAGE

```
scyld-cluster-conf
  [-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
  {load, save} ...
```

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]**
Masquerade as user *USER* with optional colon-separated password *PASSWD*.

ACTIONS

load *CLUSTER_CONFIG*

Load *CLUSTER_CONFIG* as the new configuration file, optionally loading only nodes.

- dry-run** Parse the file, but do not alter the database.
- nets-only** Ignore other settings and only load networks.
- nodes-only** Ignore other settings and only load nodes.

save *CLUSTER_CONFIG*

Save the current configuration file to file *CLUSTER_CONFIG*.

CLUSTER CONFIGURATION FILES

The `scylld-cluster-conf` command is primarily used to load a cluster configuration into the ClusterWare platform, including the PXE boot network definition(s) and the node definitions. A minimal useful configuration file consists of at least an `iprange` and one or more nodes:

```
iprange 10.10.24.100
node 08:00:27:A2:3F:C9
```

The first IP address in the `iprange` will be used to identify a local interface on the head node in order to find networking details such as the network mask. The DHCP range will be assumed to cover from the first IP up to the network broadcast address, but a "last" address can also be provided to limit that range:

```
nodes 10
iprange 10.10.24.100/24 10.10.24.199
node 08:00:27:A2:3F:C9
node
node 08:00:27:A2:E4:A2
```

Note that the node count can be provided in the file and a warning will be printed if more than that many nodes are defined in the file. The netmask can also be supplied as shown in the `iprange` line. Nodes will be numbered in order starting with index 0 but a line with no MAC address will act as a placeholder meaning this file would define nodes `n0` and `n2`.

Important

If multiple MAC addresses are included for a single node, only the first will be used.

Alternatively network definitions can specify where the node numbering actually starts:

```
1 10.10.24.100/24 10.10.24.199
node 08:00:27:A2:3F:C9
node
node 08:00:27:A2:E4:A2
```

This configuration file still defines a DHCP range of 100 IP addresses, now the nodes will be numbered starting with `n1`. In more complicated network configurations compute nodes may be split among multiple subnets:

```
1 10.10.24.100/24
node 08:00:27:A2:3F:C9
node
node 08:00:27:A2:E4:A2

21 10.10.25.100/24 10.10.25.199 via 10.10.24.4 gw 10.10.25.254
node 08:00:27:FE:A3:22
```

The first network definition will be limited to 20 IP addresses based on the first index of the second network definition. For networks that are not locally accessible to the head node(s), such as `10.10.25.0/24` in this case, the configuration file can also specify an optional route and compute node gateway. The route is specified through the `via` keyword and is only used to identify the appropriate interface for the DHCP server to listen to at run time. The gateway (`gw`) should be on the compute node network and will be provided to the booting nodes such that they can reach the head node cluster. A DHCP relay should be configured to forward DHCP traffic from the remote compute nodes to the head

nodes and vice versa, and should populate the `giaddr` field of the DHCP request with an address on the compute node subnet. For directions on configuring DHCP relays, please see your switch or operating system documentation.

When defining multiple networks they must be defined in order of node indexing. Node indexes and IP addresses are assigned based on the most recently defined network so the above example defines 3 nodes, `n1`, `n3`, and `n20`. Additional nodes added dynamically will be assigned the lowest available index and the corresponding IP address.

Important

Note that loading a cluster configuration will completely overwrite any existing configuration, including deleting all previously defined nodes.

Important

We suggest restarting the clusterware service on all head nodes after loading a new cluster configuration.

EXAMPLES

```
scyld-cluster-conf save /root/cluster-conf-bak
```

Save a copy of the current network configuration and node list.

```
scyld-cluster-conf load /root/cluster-conf-new
```

Replace the existing node definitions with ones loaded from the `/root/cluster-conf-new` file.

RETURN VALUES

Upon successful completion, **scyld-cluster-conf** returns 0. On failure, an error message is printed to `stderr` and **scyld-cluster-conf** returns 1.

3.7 Securing the Cluster

This section discusses cluster security issues that are exclusive to the ICE ClusterWare™ platform. We assume that the cluster administrator is familiar with security issues that are not solely related to ClusterWare clusters, such as securing the cluster from outside access, optionally enabling various Red Hat RHEL/CentOS functionalities for logging and auditing access to nodes and storage and for managing SELinux.

3.7.1 Authentication

The ICE ClusterWare™ cluster administrator authentication method is controlled in the `/opt/scyld/clusterware/conf/base.ini` file by the `plugins.auth` variable and is initially set to "appauth". The `scyld-install` installation adds the current user to the `auth.tmpadmins` variable in that same file (unless passed the `--no-tools` argument). The comma-separated list of user names, corresponding to system accounts on the head node, are allowed in without additional authentication checks. The `auth.tmpadmins` variable is only intended to be used during early installation, for small experimental clusters, or when recovering from some sort of failure, and is commented out by the installer during the installation process.

After installation, any administrator can add additional administrators through the `scyld-adminctl` command whose arguments match the other `scyld-*ctl` commands as described in *ICE ClusterWare Command Line Tools*. See *Configure Additional Cluster Administrators* for details. In the event of recovery, we suggest that administrators add accounts for themselves through this tool, and thereafter comment out or clear the `auth.tmpadmins` variable.

The "appauth" plugin executes the command defined in the `appauth.app_path` variable as user `root`. The default implementation of that command is provided by `/opt/scyld/clusterware/bin/pam_authenticator`. This implemen-

tation interfaces with the PAM authentication system using the `/etc/pam.d/cw_check_user` configuration file. The contents of this file initially use local system authentication, although this can be modified to authenticate against any mechanism available through the PAM system. For details, see the PAM documentation provided by your distro, the main PAM project, and the Red Hat https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system-level_authentication_guide/pluggable_authentication_modules documentation.

Administrators can provide authentication methods beyond PAM by implementing a script or application and providing it via the `appauth.app_path` variable. Appropriate applications should start with no arguments, read a username and password separated by a newline from `stdin`, and reply with either `yes` or `no` followed by a newline on `stdout`. For example, a test run of `pam_authenticator` looks like:

```
[example@head ~] sudo /opt/scyld/clusterware/bin/pam_authenticator
tester
not_the_password
no
```

3.7.1.1 Assign Temporary Permissions

If an issue occurs and you need to grant temporary administrative permissions to a user or list of users:

1. SSH into a head node.
2. Modify the `base.ini` file:

```
auth.tmpadmins = <comma-separated list of usernames>
```

3. Restart the `clusterware` service for the change to take effect:

```
systemctl restart clusterware
```

The list of users should have FullAdmin privileges.

4. Fix the issue.
5. Modify the `base.ini` file to remove the list of users who should not have FullAdmin privileges.
6. Restart the `clusterware` service for the change to take effect.

3.7.2 Role-Based Access Controls

Note

The ICE ClusterWare™ system supports the Keycloak authentication system that may be helpful in linking to existing enterprise identity management systems. For more info, see *Integrating Keycloak with ICE ClusterWare for RBAC*

Prior to version 12.2.0, the ClusterWare platform only had the concept of an "admin" who could perform any action on any of the ClusterWare database entries. The ClusterWare platform now has Role-Based Access Controls (RBAC) to allow different "classes" of administrators, some of whom may have significantly reduced sets of actions that they may perform. There are 6 system-provided Roles plus a "No Access" pseudo-role, described below. A given admin user can be assigned one or more Roles, with each Role granting a specific set of Permissions (roles are additive; if one role grants a permission, then, in general, another role cannot remove it).

By default, the first admin user added to the system will be assigned to the "Full Admin" Role and thereby gain permission to take any action within the system. When creating additional admin users, by default, they will inherit the same roles as their creator. This means that a Full Admin will create other Full Admins - the same as the default on prior versions of the ClusterWare platform.

The current ClusterWare roles are:

- **Authenticated User**
 - Any admin account will have this role which simply grants read-only access to the system. Auth-Users can read nearly any object in the ClusterWare database, with the exception of reserved attributes. This role may be useful for admins who need to report on cluster status and configuration, or perhaps for "power users" who may have some knowledge of cluster operations but should not be able to modify or change the configuration.
- **Onsite Engineer**
 - The Onsite Engineer role is aimed at technical staff who may need to interact with the machines physically, e.g. to power on/off the machines, or to rack/unrack the machines. In addition to the "read" permissions from Auth-User, the Onsite Engineer is able to issue power-control commands through `scylld-nodectl`, and to write/update node information.
- **Imaging Engineer**
 - The Imaging Engineer is primarily responsible for the creation of new images or modification of existing ones. To better assist them in that role, they are also given permissions to read reserved attributes (to see what nodes are booting into what images).
- **Production Engineer**
 - For most day-to-day operations, the Production Engineer role might be the most useful. It grants several permissions: to update/modify nodes, including power-control functionality; to update regular and reserved attributes; to create/modify images and boot configs; to create/modify naming pools; and to create/modify git repositories.
- **Manager**
 - In addition to the Auth-User (read-only) permissions, a Manager can create new admins in the system and view reserved attributes. This role may be useful in larger organizations where the creation and removal of admin accounts could be managed by less technical staff.
- **Full Admin**
 - The Full Admin role grants all possible permissions to the admin so they can read/write any object, do full power-control on nodes, update, or delete any object, etc.
 - Permissions that are only granted to Full Admins: control over distros and repos; control over dynamic groups and state sets; control over head nodes; control over hostnames and networks; and control over the cluster configuration itself.
- **No Access**
 - While not a "role" per se, the ClusterWare platform uses the No Access pseudo-role to enforce a block on an account. If an admin has the No Access role, then they will not be able to perform any actions, regardless of any other role they may have.
 - It is certainly best practice to remove such blocked accounts from the ClusterWare platform entirely, and to also remove them from any authentication system that feeds the software (such as Keycloak). However, it is sometimes helpful to have "helper" accounts that can be enabled and disabled quickly. For example, in larger sites, there may be contractors or vendor support staff that need access to a system to help with a problem; but when the problem is resolved, those accounts can be deactivated with the "No Access" role but left in-place in case another support event arises later.

To assign roles when a new account is created, use the `scylld-adminctl` tool with a comma-separated list of roles:

```
scylld-adminctl create name=charlie roles=ImagingEngineer,OnsiteEngineer
```

To assign roles after an account has been created, simply update the admin record with the new list of roles:

```
scyld-adminctl -isally update roles=Manager
```

Note

For organizations not wishing to utilize RBAC, simply assign every admin user to the "Full Admin" Role; this is the default and will then mimic the behavior of previous ClusterWare versions.

For more information, including a full description of the Roles and Permissions, see *Role-Based Access Control System*.

3.7.3 Changing the Database Password

The `scyld-install` installation configures the ICE ClusterWare™ database with a randomly generated password. This password is used when joining a new head node to the cluster and must be provided either through a command line or on request during the installation of the new head node. This password is stored in the `database.admin_pass` variable in the `/opt/scyld/clusterware/conf/base.ini` file. The details of changing this password depend on the specific database the cluster is using.

Important

Once this password is changed within the database, change the `database.admin_pass` variable in `base.ini` and restart the `clusterware` service on each head node.

3.7.3.1 Changing the etcd Password

Use the `etcdctl` tool for the rare occasion that you need to change the etcd password.

1. Change the etcd password using the `etcdctl` wrapper:

```
sudo /opt/scyld/clusterware-etcd/bin/etcdctl user passwd root
```

The wrapper code provides the existing password from the `base.ini` file to the real `etcdctl` application. That command then requests the new password and confirmation of the new password. Once this command completes, the `clusterware` service on all head nodes will stop working and all `scyld-*ctl` commands will not work until all steps are completed.

2. Update the `database.admin_pass` variable in `/opt/scyld/clusterware/conf/base.ini` file on each head node.
3. Restart the `clusterware` service on all head nodes:

```
systemctl start clusterware
```

After restarting the `clusterware` service on each head node, all `scyld-*ctl` commands will work again.

Warning

If this procedure is not followed, the cluster can become unusable and may require more extreme intervention to recover the etcd database contents. Please contact Penguin Solutions if such recovery is necessary.

3.7.4 Compute Node Remote Access

By default, remote access to compute nodes is provided through SSH using key-based authentication, although administrators may also enable password-based SSH in the compute node image by configuring a password for the root user. Every head node generates a public/private key pair and places these files in directory `/opt/scyld/clusterware/.ssh/` using the names `id_rsa.clusterware` and `id_rsa.clusterware.pub`. These keys are used by the head nodes to execute commands on the compute nodes. All head node public keys are downloaded by compute nodes at boot time by the `update_keys.sh` script and appended to `/root/.ssh/authorized_keys`. This allows any head node to execute a command on any compute node. The `/opt/scyld/clusterware/.ssh/id_rsa.clusterware` key can be used by system administrators as an "automation" key for tasks like cron jobs. It is also useful in recovery situations where an administrator may need to use this private key to directly access compute nodes

This same script that downloads the head node public keys also downloads the public keys attached to every cluster administrator account. These accounts are created using the `scyld-adminctl` tool as follows:

```
scyld-adminctl create name=admin keys=@~/.ssh/id_rsa.pub
```

This allows anyone with the corresponding `id_rsa` to SSH into the root account on any compute node booted after the key was added. The key can also be added as a string or updated for an existing administrator. For example:

```
scyld-adminctl -i admin update keys='ssh-rsa AAAAB3NzaC1yc2EAAAADA...'
```

Cluster administrators are also welcome to add SSH keys to compute node images in small private clusters. Although adding administrator accounts with public keys simplifies management of larger clusters with multiple node images or cluster administrators, administrator accounts stored in the database or listed in the `base.ini` use the same authentication mechanisms described in the previous section.

3.7.5 Compute Node Host Keys

In most computer systems the SSH `sshd` daemon uses unique host keys to identify itself to clients, and host keys are not created during image creation. This means that each compute node will generate its own host keys during boot. Since the compute node changes are discarded on reboot, a new set of keys will be generated with each boot.

In an appropriately protected cluster, some administrators prefer for all compute nodes to share host keys. This can be achieved by storing host keys in the compute node image. For example, to generate host keys and repack the Default-Image, an administrator can run:

```
scyld-modimg -i DefaultImage --exec sshd-keygen --overwrite --upload
```

All nodes that boot using this image after this change will use identical host keys, so ideally you should reboot the nodes with each node's updated image. To remove the host keys from an image, an administrator needs to delete the `/etc/ssh/ssh_host_*` files from the compute node image.

3.7.6 Encrypting Communications

The ICE ClusterWare™ platform provides an internal cluster certificate authority that can provide signed certificates to secure communications between head nodes as well as between head nodes and compute nodes. If you prefer to provide your own certificates or certificate authority, Apache can be configured to use those certificates and compute nodes can verify those signatures during communications. Apache configuration files are located in `/opt/scyld/clusterware/conf/httpd/`. The Apache VirtualHost definition can be found in `vhost.conf`. The proxy definition in that file needs to be included into the HTTPS VirtualHost. For details about how to properly enable HTTPS on the Apache server, see the documentation provided by your distro.

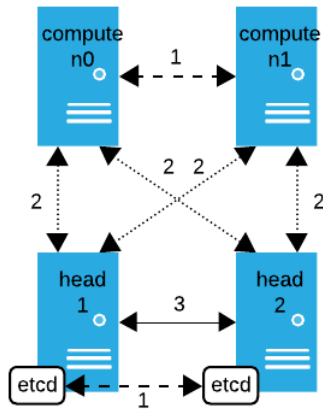
Once HTTPS is enabled, the `~/.scyldcw/settings.ini` file of any existing ClusterWare tool installation should be updated. In `~/.scyldcw/settings.ini`, update the protocol of the `client.base_url` variable to use https. It should be safe to leave HTTP enabled for localhost-only access and, in that case, local tool installations can continue to use the original localhost-based URL.

The internal cluster certificate authority is implemented using a certificate stored in the backend database, accessible to all head nodes, and can be used to sign certificates used by assorted services on the head node by executing:

```
/opt/scyld/clusterware/bin/generate_head_certs
```

The `scyld-install` script runs this command during installation or updates to generate separate certificates for supported services on the head node. The keys and certificates are stored in `/opt/scyld/clusterware/ca_certs/` along with the CA public key.

Communication between devices in your cluster happens with different levels of encryption and authentication. The following diagram illustrates the use of TLS in communications involving head and compute nodes with all possible configurations:



1. TLS off - not encrypted

- Between compute nodes: Communication between compute nodes while chain booting is not encrypted; however, the compute nodes do confirm the size and checksum of the received files based on information from the head node.
- Between etcd on head nodes: Communication between etcd on one head node and etcd on another head node is not encrypted or authenticated. All communication between etcd should occur on a secure network.

2. TLS on - encrypted, but not authenticated by default

- *Configure Encrypted Communication between Head and PXE Compute Nodes*
- *Configure Encrypted Communication between Head and Diskful Compute Nodes*
- *Configure Client Authentication between Head and Compute Nodes*

3. TLS on - encrypted and authenticated

Between head nodes: All backend communication between head nodes is encrypted and authenticated.

3.7.6.1 Configure Encrypted Communication between Head and PXE Compute Nodes

By default, communication between ClusterWare head and PXE compute nodes is encrypted using SSL/TLS. Leaving this on is highly recommended for security. If necessary, you can turn off SSL/TLS by setting `head.prefer_ssl = False` in the `/opt/scyld/clusterware/conf/base.ini` file on the head node(s). When this is enabled, compute nodes send node status information back to the ClusterWare platform over HTTPS, but do not authenticate the head node.

3.7.6.2 Configure Encrypted Communication between Head and Diskful Compute Nodes

To enable TLS encryption for diskful compute nodes, modify the `base_url` in `/etc/clusterware/node.sh` to use HTTPS. The `sslverify` option in the same file defaults to "no", but if the cluster CA certificate is installed on the diskful node, setting `sslverify=yes` in `node.sh` enables host authentication.

See *Installing the clusterware-node Package* for details about the `node.sh` file.

3.7.6.3 Configure Client Authentication between Head and Compute Nodes

To enable encryption with client authentication, enable one-time Certificate Signing Requests (CSR) for specific nodes using the following command:

```
scyld-clusterctl certs enable -i <nodes>
```

where `<nodes>` is a comma-separated list of node IDs.

Reboot all specified nodes after running the command.

This functionality requires a TPM on each compute node because the compute nodes create a private key in their TPM on boot and use it to sign a CSR. The CSR is sent to the head node. If CSR signing is enabled for a specific node, the head node responds with a signed client certificate. The node stores the signed client certificate in non-volatile TPM storage.

If the client finds a client certificate in its TPM on the same or subsequent boots, it uses that certificate and the private key in the TPM to authenticate itself to the server. Successful authentication to the server results in an encrypted TLS connection with client authentication.

Note that future releases will allow encryption with host authentication by automating the process of adding the cluster CA certificate to compute nodes booting via UEFI.

3.7.7 Security-Enhanced Linux (SELinux)

Security-Enhanced Linux (*SELinux*) is a set of patches to the Linux kernel and various utilities that provide mandatory access control to major subsystems of a node. See https://en.wikipedia.org/wiki/Security-Enhanced_Linux for general discussion of SELinux.

The ICE ClusterWare™ platform supports SELinux on the head nodes and compute nodes.

3.7.7.1 SELinux On Compute Nodes

For Red Hat RHEL and CentOS compute nodes, the root file systems created by the `scyld-modimg` tool include SELinux support as part of the installation of the `@core` yum group. During the boot process the `mount_rootfs` script will, like the standard dracut based `initramfs`, load the SELinux policy before switching root. Note that the default cmdline in the boot configurations created through `scyld-add-boot-config` (including the DefaultBoot configuration) will contain `enforcing=0`, thereby placing all compute nodes in SELinux "permissive" mode. Only remove this option once you have completed testing to confirm that your applications will run as expected with SELinux in "enforcing" mode.

SELinux on compute nodes may be disabled in the standard ways through command line arguments or by changing the contents of the node's `/etc/selinux/config` configuration file. For details please refer to appropriate distro-provided documentation.

In addition to the default "targeted" SELinux policy provided by RHEL and CentOS, the ClusterWare platform also supports the Multi-Level Security (MLS) policy for compute nodes. Enabling the MLS policy inside an image is done the same way as it would be done on a locally installed system. After entering the image chroot using `scyld-modimg`, first install the `selinux-policy-mls` package, and then modify the `/etc/selinux/config` file to reference the newly installed policy. Because the `clusterware-node` SELinux policy module is installed at image creation time, it may need to be re-installed after switching to the MLS policy:

```
semodule --install /opt/scyld/clusterware-node/clusterware-node.pp.bz2
```

The `semodule` command can also be used to check if the policy is loaded:

```
semodule --list | grep clusterware
```

When exiting the `chroot`, the ClusterWare platform automatically relabels the file system based on the policy referenced in `/etc/selinux/config`.

Important

Fully configuring a cluster for MLS requires significant effort, including labeling objects on shared storage and defining additional policy around user workflows and tools. Please refer to your operating system documentation, as such details are beyond the scope of this document. Note that ClusterWare-provided schedulers, MPI implementations, and 3rd party applications may need additional custom permissions not covered here in order to configure a functional MLS cluster.

When creating boot configuration for an MLS enabled image, please be aware that the MLS policy, by default, does not allow the root user to log into the compute node via `ssh`. Because `ssh` is used by the ClusterWare soft power commands, please either enable the root login functionality or use the `_remote_user` node attribute to configure login as a user with `sudo shutdown` permission. The root login permission can be enabled through the `setsebool` command, and the boolean is named `ssh_sysadm_login`.

3.7.7.2 SELinux On Head Nodes

On head nodes, SELinux is detected to be in "enforcing" mode at both installation and service run time. To switch SELinux from "enforcing" to "permissive" mode, please see the documentation for your operating system. If this switch is made while the `clusterware` service is running, please restart that service:

```
sudo systemctl restart clusterware
```

3.7.7.3 MLS Policy On Head Nodes

For head nodes enforcing the MLS policy, the SELinux user `sysadm_u` should be used to install the ClusterWare platform and run administrative tools.

To map a Linux user to the `sysadm_u` SELinux user, you can run:

```
sudo semanage login --add linux_user --seuser sysadm_u
```

By default, the `sysadm_u` user should run with the `sysadm_t` domain.

3.7.8 Security Technical Implementation Guides (STIG)

STIG security hardening implements compliance with the Defense Information Systems Agency (DISA) guidelines described in the Security Technical Implementation Guides (STIGs) (https://csrc.nist.gov/glossary/term/security_technical_implementation_guide). Certain high-security clusters may require STIG compliance.

The ICE ClusterWare™ platform provides basic STIG support for kickstarted nodes by adding the following snippet to your kickstart `*.ks` file:

```
%addon org_fedora_oscap
  content-type = scap-security-guide
```

(continues on next page)

(continued from previous page)

```
profile = xccdf_org.ssgproject.content_profile_stig
%end
```

To configure a STIG head node, add the snippet to your kickstart config file and reboot the node using that *.ks file to enable STIG. Then *Install ICE ClusterWare* on the STIG-enabled node in the usual way.

The ClusterWare software provides an example file `/opt/scyld/clusterware/kickstarts/basic-stig.ks` with that snippet appended for administrators who would like to kickstart infrastructure nodes or additional head nodes with that STIG applied at install time.

3.8 Services, Ports, Protocols

3.8.1 Apache

Apache serves the ICE ClusterWare™ REST API via HTTP on port 80 using `mod_wsgi` through the `httpd` systemd service aliased as `clusterware`. HTTPS Encryption over port 443 can be enabled through standard Apache and operating system procedures. Apache is Open Source, and Penguin Computing contributes the REST API. The log files are `/var/log/clusterware/api_access_log` and `/var/log/clusterware/api_error_log`.

The ClusterWare GUI is also served through Apache from the `/var/www/clusterware/front/` directory.

3.8.2 Chrony

Chrony is used to keep time synchronized across the cluster, including synchronization to upstream network time-servers and to all nodes within the cluster itself. The systemd service name is `chronyd` and it uses port 123 for its time-keeping communications, and port 323 for receiving commands from the `chronyc` management tool. This service is configured, started, and stopped by the ClusterWare service based on the cluster configuration. The configuration file is generated from a template located at `/opt/scyld/clusterware-chrony/chrony.conf.template`.

3.8.3 DHCP

DHCP provides dynamic host configuration, with a systemd service name `clusterware-dhcpd` and using port 68. The log file is `var/log/clusterware/isc-dhcpd.log`. This service is configured, started, and stopped by the ClusterWare service based on the cluster configuration. The configuration file is generated from a template located at `/opt/scyld/clusterware-iscdhcp/dhcpd.conf.template`.

Important

Never directly try to control DHCP on a ClusterWare head node, it will not work. Services that start with `clusterware-` are managed by the ClusterWare service.

3.8.4 DNS

DNS provides name- and ip-address-lookup services, with a systemd service name `clusterware-dnsmasq` and using port 53. This service is configured, started, and stopped by the ClusterWare service based on the cluster configuration. The configuration file is generated from a template located at `/opt/scyld/clusterware-dnsmasq/dnsmasq.conf.template`.

Important

Never directly try to control DNS on a ClusterWare head node, it will not work. Services that start with `clusterware-` are managed by the ClusterWare service.

3.8.5 etcd

The replicated configuration key/value store etcd has the systemd service name *clusterware-etcd*. Log files are found in */var/log/clusterware/*. etcd uses port 52380 to communicate with other head nodes.

ii Important

Never directly try to control etcd on a ClusterWare head node, it will not work. Services that start with *clusterware-* are managed by the ClusterWare service.

3.8.6 iSCSI

iSCSI optionally serves root filesystems to compute nodes and uses port 3260. Serving root file systems via iSCSI is configured by the ClusterWare service using the *targetcli* command line tool.

3.8.7 OpenSSH

OpenSSH provides services to remotely execute programs and to transfer files, with a systemd service name *sshd* and using port 22. Encryption is SSH. The log file is */var/log/messages*.

3.8.8 Telegraf / Telegraf-Relay / InfluxDB

All head node and compute node performance data collected by telegraf is sent to a systemd service named *telegraf-relay* on one of the head nodes over HTTP(S). Telegraf-Relay replicates and relays this data to every other telegraf-relay over HTTPS. Each telegraf-relay finally sends the data to its locally hosted InfluxDB for storage.

3.8.9 TFTP

The TFTP Server provides downloads for early iPXE boot files, with a systemd service name *xinetd* and using port 69. This service can be replaced by appropriate network card firmware. The log file is */var/log/messages*.

3.9 Common Additional Configuration

Following a successful initial install or update of ICE ClusterWare™, or as local requirements of your cluster dictate, you may need to make one or more configuration changes.

3.9.1 Configure Hostname

Verify that the head node *hostname* has been set as desired for permanent, unique identification across the network. In particular, ensure that the *hostname* is not *localhost* or *localhost.localdomain*.

3.9.2 Managing Databases

The ClusterWare platform currently only supports the etcd database.

On head nodes with multiple IP addresses the current ClusterWare etcd implementation has no way to identify the correct network for communicating with other head nodes. By default the system will attempt to use the first non-local IP. Although this is adequate for single head clusters and simple multihead configurations, a cluster administrator setting up a multihead cluster should specify the correct IP. This is done by setting the *etcd.peer_url* option in the */opt/scyld/clusterware/conf/base.ini* file. A correct peer URL on a head node with the IP address of 10.24.1.1, where the 10.24.1.0/24 network should be used for inter-head communications might look like:

```
etcd.peer_url = http://10.24.1.1:52380
```

If this value needs to be set or changed on an existing cluster, it should be updated on a single head node, then `managedb recover` run on that head node, and then other heads (re-)joined to the now correctly configured one. The `etcd.peer_url` setting should only be necessary on the first head as the proper network will be communicated to new heads during the join process.

The ClusterWare `etcd` implementation does not allow the second-to-last head node in a multihead cluster to leave or be ejected. See [Removing a Joined Head Node](#) for details, and [Managing Multiple Head Nodes](#) for broader information about multiple headnode management.

Important

Prior to any manipulation of the distributed database, whether through `managedb recover`, joining head nodes to a cluster, removing head nodes from a cluster, or switching from Couchbase to `etcd`, the administrator is strongly encouraged to make a backup of the ClusterWare database using the `managedb` tool. See [managedb](#).

The `etcdctl` command provides scriptable direct document querying and manipulation. The ClusterWare platform provides a wrapped version of `etcdctl` located in the `/opt/scyld/clusterware-etcd/bin/` directory. The wrapper should be run as root and automatically applies the correct credentials and connects to the local `etcd` endpoint. Note that direct manipulation of database JSON documents should only be done when directed by Penguin support.

3.9.3 Configure Administrator Authentication

ClusterWare administrator authentication is designed to easily integrate with already deployed authentication systems via PAM. By default cluster administrators are authenticated through the `pam_authenticator` tool that in turn uses the PAM configuration found in `/etc/pam.d/cw_check_user`. In this configuration, administrators can authenticate using their operating system password as long as they have been added to the ClusterWare system using the `scyld-adminctl` command. For example, to add username "admin1":

```
scyld-adminctl create name=admin1
```

If a ClusterWare administrator is running commands from a system account on the head node by the same name (i.e. ClusterWare administrator *fred* is also head node user *fred*), the system will confirm their identity via a Unix socket based protocol. Enabled by default, this mechanism allows the `scyld` tools to connect to a local socket to securely set a dynamically generated one-time password that is then accepted during their next authentication attempt. This takes place transparently, allowing the administrator to run commands without providing their password. The client code also caches an authentication cookie in the user's `.scyldcw/auth_tkt.cookie` for subsequent authentication requests.

Managing cluster user accounts is generally outside the scope of the ClusterWare platform and should be handled by configuring the compute node images appropriately for your environment. In large organizations this usually means connecting to Active Directory, LDAP, or any other mechanism supported by your chosen compute node operating system. In simpler environments where no external source of user identification is available or it is not accessible, the ClusterWare platform provides a `sync-uids` tool. This program can be found in the `/opt/scyld/clusterware-tools/bin` directory and can be used to push local user accounts and groups either to compute nodes or into a specified image. For example:

```
# push uids and their primary uid-specific groups:
sync-uids --users admin1,tester --image SlurmImage

# push uid with an additional group:
sync-uids --users admin1 --groups admins --image SlurmImage
```

The above pushes the users and groups into the compute node image for persistence across reboots. Then either reboot the node(s) to see these changes, or push the IDs into running nodes with:

```
sync-uids --users admin1,tester --nodes n[1-10]
```

The tool generates a shell script that is then executed on the compute nodes or within the image `chroot` to replicate the user and group identifiers on the target system. This tool can also be used to push ssh keys into the `authorized_keys` files for a user onto booted compute nodes or into a specified image. Please see the tool's `--help` output for more details and additional functionality, such as removing users or groups, and controlling whether home directories are created for injected user accounts.

3.9.4 Disable/Enable Chain Booting

The default ClusterWare behavior is to perform *chain booting* for more efficient concurrency for servicing a flood of PXEbooting nodes that are requesting their large *rootfs* file. Without chain booting, the head node(s) serve the *rootfs* file for all PXEbooting nodes and thus become a likely bottleneck when hundreds of nodes are concurrently requesting their file. With chain booting, the head node(s) serve the *rootfs* files to the first compute node requesters, then those provisioned compute nodes offer to serve as a temporary *rootfs* file server for other requesters.

In the event that the cluster administrator wishes to disable chain booting, then the cluster administrator executing as user `root` should edit the file `/opt/scyld/clusterware/conf/base.ini` to add the line:

```
chaining.enable = False
```

To reenable chain booting, either change that `False` to `True`, or simply comment-out that `chaining.enable` line to revert back to the default enabled state.

3.9.5 scyld-nss Name Service Switch (NSS) Tool

The *scyld-nss* package provides a Name Service Switch (NSS) tool that translates a hostname to its IP address or an IP address to its hostname(s), as specified in the `/etc/scyld-nss-cluster.conf` configuration file. These hostnames and their IP addresses (e.g., for compute nodes and switches) are those managed by the ClusterWare database, which automatically provides that configuration file at startup and thereafter if and when the cluster configuration changes.

Note

scyld-nss is currently only supported on head nodes.

Installing *scyld-nss* inserts the *scyld* function in the `/etc/nsswitch.conf` *hosts* line, and installs the ClusterWare `/lib64/libnss_scyld*` libraries to functionally integrate with the other NSS `/lib64/libnss_*` libraries.

Benefits include an expanded functionality of ClusterWare hostname resolution and increased performance of NSS queries for those hostnames. Install the *nscd* package for additional performance improvement of hostname queries, especially on clusters with very high node counts.

The *scyld-nss* package includes a `scyld-nssctl` tool allowing a cluster administrator to manually `stop` or `start` the service by removing or reinserting the *scyld* function in `/etc/nsswitch.conf`. Any user can employ `scyld-nssctl` to query the current status of the service. See *scyld-nssctl* for details.

3.9.6 Firewall Configuration

If you are not using the *torque-scyld* or *slurm-scyld* packages, either of which will transparently configure the firewall on the private cluster interface between the head node(s), job scheduler servers, and compute nodes, then you need to configure the firewall manually for both the head node(s) and all compute nodes.

3.9.7 Configure IP Forwarding

By default, the head node does not allow IP forwarding from compute nodes on the private cluster network to external IP addresses on the public network. If IP forwarding is desired, then it must be enabled and allowed through each head node's `firewalld` configuration.

On a head node, to forward internal compute node traffic through the `<PUBLIC_IF>` interface to the outside world, execute:

```
firewall-cmd --zone=external --change-interface=<PUBLIC_IF>
# confirm it was working at this point then make it permanent
firewall-cmd --permanent --zone=external --change-interface=<PUBLIC_IF>
```

Appropriate routing for compute nodes can be modified in the compute node image(s) (see *scyld-modimg* tool). Limited changes may also require modifying the DHCP configuration template `/opt/scyld/clusterware-iscdhcp/dhcpd.conf.template`.

3.9.8 Status and Health Monitoring

The ClusterWare platform provides a set of status and monitoring tools out-of-the-box, but admins can also use the plugin system to add or modify the list of status, hardware, health-check, and monitoring (Telegraf) plugins. Some of these plugins will be built into the disk image and cannot be removed without modifying that image manually; others may be added or removed on-the-fly through several node attributes:

```
[admin1@head]$ scyld-nodectl ls -L
Nodes
  n0
    attributes
      _boot_config: DefaultBoot
      _status_plugins=chrony,ipmi
      _hardware_plugins=infiniband,nvidia
      _health_plugins=rasmem,timesync
      _telegraf_plugins=lm-sensors,nvidia-smi
    domain: cluster.local
    . . .
```

The `scyld-nodectl` tool can be used to apply these attributes to individual or groups of nodes, or admins can create attribute-groups with `scyld-attrbctl` and then join nodes to those groups.

See *ICE ClusterWare Plugin System* for more details on the ClusterWare Plugin System.

3.9.9 Install Name Service Cache Daemon (nscd)

The Name Service Cache Daemon (*nscd*) provides a cache for most common name service requests. The performance impact for very large clusters is significant.

3.9.10 Install jq Tool

The *jq* tool (`/usr/bin/jq`) is installable from the standard Linux distribution repositories and provides a command-line parser for JSON output.

For example, for the `--long` status of node `n0`:

```
[sysadmin@head1 /]$ scyld-nodectl -i n0 ls --long
Nodes
  n0
```

(continues on next page)

(continued from previous page)

```

attributes
  _boot_config: DefaultBoot
  _no_boot: 0
  last_modified: 2019-06-05 23:44:48 UTC (8 days, 17:09:55 ago)
groups: []
hardware
  cpu_arch: x86_64
  cpu_count: 2
  cpu_model: Intel Core Processor (Broadwell)
  last_modified: 2019-06-06 17:15:59 UTC (7 days, 23:38:45 ago)
  mac: 52:54:00:a6:f3:3c
  ram_total: 8174152
index: 0
ip: 10.54.60.0
last_modified: 2019-06-14 16:54:39 UTC (0:00:04 ago)
mac: 52:54:00:a6:f3:3c
name: n0
power_uri: none
type: compute
uid: f7c2129860ec40c7a397d78bba51179a

```

You can use *jq* to parse the JSON output to extract specific fields:

```

[sysadmin@head1 /]$ scyld-nodectl --json -i n0 ls -l | jq '.n0.mac'
"52:54:00:a6:f3:3c"

[sysadmin@head1 /]$ scyld-nodectl --json -i n0 ls -l | jq '.n0.attributes'
{
  "_boot_config": "DefaultBoot",
  "_no_boot": "0",
  "last_modified": 1559778288.879129
}

[sysadmin@head1 /]$ scyld-nodectl --json -i n0 ls -l | jq '.n0.attributes._boot_config'
"DefaultBoot"

```

All of the *scyld-** tools can produce JSON data, so similar techniques can be applied to images, boot configuration, and so on. For example, use the the following to see the sizes of different images:

```

[sysadmin@head1 /]$ scyld-imgctl --json ls -L | jq '.[].content.cwsquash.size'
1277071360
1467298823

[sysadmin@head1 /]$ scyld-imgctl --json ls -L | jq '.[] | "\(.name) \(.content.cwsquash.
↪size)'"
"DefaultImage 1277071360"
"NewImage 1467298823"

```

In this example, *jq* takes all of the top-level items (*. []*) and then creates a text string for each item (the double quotes), using the *.name* and *.content.cwsquash.size* fields separated by a space. The *\(. .)* notation selects the fields.

Further information, including tutorials and user manuals, can be found at <https://jqlang.github.io/jq/>.

3.10 Additional Software

`scyl-d-install` installs and updates the basic ICE ClusterWare™ software. Additional software packages are available in the ClusterWare repository.

`scyl-d-install` manipulates the `/etc/yum.repos.d/clusterware.repo` file to automatically enable the `scyl-d` repos when the tool executes and disable the repos when finished. This is done to avoid inadvertent updating of ClusterWare packages when executing a simple `yum update`.

Note

If the cluster administrator has created multiple `/etc/yum.repos.d/*.repo` files that specify repos containing ClusterWare RPMs, then this protection against inadvertent updating is performed only for `/etc/yum.repos.d/clusterware.repo`, not for those additional repo files.

Accordingly, the `--enablerepo=scyl-d*` argument is required when using `yum` for listing, installing, and updating these optional ClusterWare packages on a head node. For example, these optional installable software packages can be viewed using `yum list --enablerepo=scyl-d* | grep scyl-d`. After installation, any available updates can be viewed using `yum check-update --enablerepo=scyl-d* | grep scyl-d`.

Specific install and configuration instructions for various of these packages, e.g., job managers and OpenMPI middle-ware, are detailed in this chapter.

3.10.1 Adding 3rd-party Software

An existing compute node image may need to contain additional software (e.g., a driver and perhaps the driver's associated software) that has been downloaded from a 3rd-party vendor in the form of an RPM or a tarball.

Suppose a tarball named `driver-tarball.tgz` has been downloaded into the head node `/tmp/` directory, and you need to install its contents into an image. A cautious first step is to clone an existing image and add the new software to that clone, which leaves the existing image unmodified. For example, clone a new image:

```
scyl-d-imgctl -i DefaultImage clone name=UpdatedImage
```

Now enter the new *UpdatedImage* in a chroot environment:

```
scyl-d-modimg -i UpdatedImage --chroot
```

Suppose your administrator user name is `admin1`. Inside the chroot you are always user `root`. Copy the downloaded tarball from the head node into your chroot with a simple command from inside the chroot:

```
scp -r admin1@localhost:/tmp/driver-tarball.tgz /tmp
```

Unpack `/tmp/driver-tarball.tgz` and examine the contents, where you will likely find a script that manages the tarball-specific software installation.

Important

Carefully read the instructions provided by the 3rd-party software vendor before executing the script, and carefully read the output produced when executing the script.

There are several factors to keep in mind when executing the 3rd-party install script:

- A 3rd-party installation that involves a new kernel module requires linking that module to the kernel in the chroot. This requires the presence of the `kernel-devel` package that matches that kernel. If that RPM is not

currently installed in the chroot, then inside the chroot manually `yum install` it, naming the specific kernel version, e.g.,:

```
yum install kernel-devel-3.10.0-957.27.2.el7.x86_64
```

to match `kernel-3.10.0-957.27.2.el7.x86_64`.

- Some 3rd-party install scripts use the `uname` command to determine the kernel against which to link a new kernel module. However, when the `uname` command executes inside a chroot, it actually reports the kernel version of the host system that executes the `scyld-modimg --chroot` command, **not** the kernel that has been installed inside the chroot. This `uname` behavior only works properly for module linking purposes if the chroot contains only one kernel and if that kernel matches the kernel on the `scyld-modimg --chroot-executing` server. To specify an alternate kernel, either name that kernel as an optional argument of a `--chroot` argument, e.g.,:

```
scyld-modimg -i NewImage --chroot 3.10.0-1160.45.1.el7.x86_64
```

or as a `KVER` variable value using the `--exec` argument, e.g., for a script inside the image that initializes a software driver module and links that module to a specific kernel:

```
scyld-modimg -i NewImage --execute 'KVER=3.10.0-1160.45.1.el7.x86_64 /path/to/script  
↪'
```

Otherwise, hopefully the 3rd-party install script supports an optional argument that specifies the intended kernel version, such as:

```
/path/to/install-script -k 3.10.0-1160.45.1.el7.x86_64
```

- If the 3rd-party install script encounters a missing dependency RPM, then the script reports the missing package name(s) and fails. You must manually `yum install` those missing RPM(s) within the chroot and reexecute the script.
- Some 3rd-party install scripts replace RPMs that were installed from the base distribution, e.g., Infiniband, OFED. If any currently installed ICE ClusterWare™ packages declare these base distribution packages as dependencies, then the install script's attempt to replace those packages fails. You must then uninstall the specified ClusterWare package(s) (e.g., `openmpi3.1`, `openmpi3.1-intel`), then retry executing the install script. In some cases the 3rd-party tarball contains packages that replace the ClusterWare package(s). In other cases you can reinstall these ClusterWare package(s) after the 3rd-party install script successfully completes.

Finally, `exit` the chroot and specify to *Keep changes*, *Replace local image*, *Upload image*, and *Replace remote image*.

3.10.2 Job Schedulers

The default ICE ClusterWare™ installation for RHEL/Rocky 8 includes support for the optional packages Slurm and OpenPBS. These optional packages can coexist on a scheduler server, which may or may not be a ClusterWare head node. However, if job schedulers are installed on the same server, then only one at a time should be enabled and executing on that given server.

All nodes in the job scheduler cluster must be able to resolve hostnames of all other nodes as well as the scheduler server hostname. The ClusterWare platform provides a DNS server in the `clusterware-dnsmasq` package, as discussed in *Node Name Resolution*. This `dnsmasq` will resolve all compute node hostnames, and the job scheduler's hostname should be added to `/etc/hosts` on the head node(s) in order to be resolved by `dnsmasq`. Whenever `/etc/hosts` is edited, please restart the `clusterware-dnsmasq` service with:

```
sudo systemctl restart clusterware-dnsmasq
```

Installing and configuring a job scheduler requires making changes to the compute node software. When using image-based compute nodes, we suggest first cloning the *DefaultImage* or creating a new image, leaving untouched the *DefaultImage* as a basic known-functional pristine image.

For example, to set up nodes n0 through n3, you might first do:

```
scyld-imgctl -i DefaultImage clone name=jobschedImage
scyld-bootctl -i DefaultBoot clone name=jobschedBoot image=jobschedImage
scyld-nodectl -i n[0-3] set _boot_config=jobschedBoot
```

When these nodes reboot after all the setup steps are complete, they will use the *jobschedBoot* and *jobschedImage*.

See <https://slurm.schedmd.com/rosetta.pdf> for a discussion of the differences between PBS TORQUE and Slurm. See <https://slurm.schedmd.com/faq.html#torque> for useful information about how to transition from OpenPBS or PBS TORQUE to Slurm.

The following sections describe the installation and configuration of each job scheduler type.

3.10.2.1 Slurm

See *Job Schedulers* for general job scheduler information and configuration guidelines. See <https://slurm.schedmd.com> for Slurm documentation.

Note

As of Clusterware 12, the default slurm-scyld configuration is Configless. This reduces the admin effort needed when updating the list of compute nodes. See https://slurm.schedmd.com/configless_slurm.html for more information.

Install Slurm

First, install Slurm software on the job scheduler server.

- For RHEL/Rocky 8:

```
sudo yum install slurm-scyld --enablerepo=scyld* --enablerepo=powertools
```

- For RHEL/Rocky 9:

```
sudo yum install slurm-scyld --enablerepo=scyld* --enablerepo=crb
```

- For all other systems:

```
sudo yum install slurm-scyld --enablerepo=scyld*
```

Note

An additional RPM package, *slurm-scyld-slurmrestd*, is available. See <https://slurm.schedmd.com/slurmrestd.html> for details. The *slurm-scyld-slurmrestd* package is not installed by default. To install the package, run `yum --enablerepo=scyld* install slurm-scyld-slurmrestd`.

Next, use a helper script `slurm-scyld.setup` to complete the initialization and setup the job scheduler and the compute node image(s).

```
slurm-scyld.setup init
```

The `slurm-scyld.setup` script performs the `init`, `reconfigure`, and `update-nodes` actions (described below) by default against all `up` nodes. Those actions optionally accept a node-specific argument using the syntax `[--ids|-i <NODES>]` or a group-specific argument using `[--ids|-i %<GROUP>]`. See *Attribute Groups and Dynamic Groups* for details.

`init` first generates `/etc/slurm/slurm.conf` by trying to install `slurm-scyld-node` and run `slurmd -C` on 'up' nodes. By default configless slurm is enabled by "SlurmctldParameters=enable_configless" in `/etc/slurm/slurm.conf`, and a DNS SRV record called `slurmctld_primary` is created. To see the details about the SRV, run: `scyld-clusterctl hosts -i slurmctld_primary ls -l`.

Note

For clusters with a backup Slurm controller, create a `slurmctld_backup` DNS SRV record:

```
scyld-clusterctl --hidden hosts create name=slurmctld_backup port=6817 \
    service=slurmctld domain=cluster.local target=backuphostname \
    type=srvrec priority=20
```

However if there are no 'up' nodes or `slurm-scyld-node` installation fails for some reason, then no node is configured in `slurm.conf` during `init`. Later you can use `reconfigure` to create a new `slurm.conf` or `update-node` to update the nodes in an existing `slurm.conf`. `init` also generates `/etc/slurm/cgroup.conf` and `/etc/slurm/slurmdbd.conf`, starts `munge`, `slurmctld`, `mariadb`, `slurmdbd`, and restarts `slurmctld`. At last, `init` tries to start `slurmd` on nodes. In an ideal case if the script succeeds to install `slurm-scyld-node` on compute nodes, `srun -N 1 hostname` works after `init`.

The `slurmd` installation and configuration on 'up' nodes do not survive after nodes reboot, unless on diskful compute nodes. To make a persistent slurm image:

```
slurm-scyld.setup update-image slurmImage
```

By default `update-image` does not include slurm config files into `slurmImage` if configless is enabled, otherwise includes config files into `slurmImage`. You can overwrite this default behavior by appending an additional arg `--copy-configs` or `--remove-configs` after `slurmImage` as in above command.

Reboot the compute nodes to bring them into active management by Slurm. Check the Slurm status:

```
slurm-scyld.setup status
```

If any services on controller (`slurmctld`, `slurmdbd` and `munge`) or compute nodes (`slurmd` and `munge`) are not running, you can try to use `systemctl` to start individual service, or use `slurm-scyld.setup cluster-restart`, `slurm-scyld.setup restart` and `slurm-scyld.setup start-nodes` to restart slurm cluster-wide, controller only and nodes only.

Note

The above restart or start do not effect `slurmImage`.

The `update-image` is necessary for persistence across compute node reboots.

Working with Slurm

Generate new slurm-specific config files with:

```
slurm-scyld.setup reconfigure      # default to all 'up' nodes
```

Add nodes by executing:

```
slurm-scyld.setup update-nodes    # default to all 'up' nodes
```

or add or remove nodes by directly editing the `/etc/slurm/slurm.conf` config file.

Note

With Configless Slurm, the `slurmImage` does NOT need to be reconfigured after new nodes are added -- Slurm will automatically forward the new information to the `slurmd` daemons on the nodes.

Inject users into the compute node image using the `sync-uids` script. The administrator can inject all users, or a selected list of users, or a single user. For example, inject the single user `janedoe`:

```
/opt/scyld/clusterware-tools/bin/sync-uids \
    -i slurmImage --create-homes \
    --users janedoe --sync-key janedoe=/home/janedoe/.ssh/id_rsa.pub
```

See [Configure Administrator Authentication](#) and `/opt/scyld/clusterware-tools/bin/sync-uids -h` for details.

To view the Slurm status on the server and compute nodes:

```
slurm-scyld.setup status
```

The Slurm service can also be started and stopped cluster-wide with:

```
slurm-scyld.setup cluster-stop
slurm-scyld.setup cluster-start
```

Slurm executable commands and libraries are installed in `/opt/scyld/slurm/`. The Slurm controller configuration can be found in `/etc/slurm/slurm.conf`, and each node caches a copy of that `slurm.conf` file in `/var/spool/slurmd/conf-cache/`. Each Slurm user must set up the `PATH` and `LD_LIBRARY_PATH` environment variables to properly access the Slurm commands. This is done automatically for users who login when Slurm is running via the `/etc/profile.d/scyld.slurm.sh` script. Alternatively, each Slurm user can manually execute `module load slurm` or can add that command line to (for example) the user's `~/.bash_profile` or `~/.bashrc`.

For a traditional config-file-based Slurm deployment, the admin will have to push the new `/etc/slurm/slurm.conf` file out to the compute nodes and then restart `slurmd`. Alternately, the admin can modify the boot image to include the new config file, and then reboot the nodes into that new image.

3.10.2.2 OpenPBS

OpenPBS is only available for RHEL/CentOS 8 clusters.

See [Job Schedulers](#) for general job scheduler information and configuration guidelines. See <https://www.openpbs.org> for OpenPBS documentation.

First install OpenPBS software on the job scheduler server:

```
sudo yum install openpbs-scyld --enablerepo=scyld*
```

Use a helper script to complete the initialization and setup the job scheduler and config file in the compute node image(s).

Note

The `openpbs-scyld.setup` script performs the `init`, `reconfigure`, and `update-nodes` actions (described below) by default against all *up* nodes. Those actions optionally accept a node-specific argument using the syntax `[--ids|-i <NODES>]` or a group-specific argument using `[--ids|-i %<GROUP>]`. See *Attribute Groups and Dynamic Groups* for details.

```
openpbs-scyld.setup init                # default to all 'up' nodes
openpbs-scyld.setup update-image openpbsImage # for permanence in the image
```

Reboot the compute nodes to bring them into active management by OpenPBS. Check the OpenPBS status:

```
openpbs-scyld.setup status

# If the OpenPBS daemon is not executing, then:
openpbs-scyld.setup cluster-restart

# And check the status again
```

This `cluster-restart` is a manual one-time setup that doesn't affect the *openpbsImage*. The `update-image` is necessary for persistence across compute node reboots.

Generate new openpbs-specific config files with:

```
openpbs-scyld.setup reconfigure        # default to all 'up' nodes
```

Add nodes by executing:

```
openpbs-scyld.setup update-nodes      # default to all 'up' nodes
```

or add or remove nodes by executing `qmgr`.

Any such changes must be added to *openpbsImage* by reexecuting:

```
openpbs-scyld.setup update-image openpbsImage
```

and then either reboot all the compute nodes with that updated image, or additionally execute:

```
openpbs-scyld.setup cluster-restart
```

to manually push the changes to the *up* nodes without requiring a reboot.

Inject users into the compute node image using the `sync-uids` script. The administrator can inject all users, or a selected list of users, or a single user. For example, inject the single user *janedoe*:

```
/opt/scyld/clusterware-tools/bin/sync-uids \
    -i openpbsImage --create-homes \
    --users janedoe --sync-key janedoe=/home/janedoe/.ssh/id_rsa.pub
```

See *Configure Administrator Authentication* and `/opt/scyld/clusterware-tools/bin/sync-uids -h` for details.

To view the OpenPBS status on the server and compute nodes:

```
openpbs-scyld.setup status
```

The OpenPBS service can also be started and stopped cluster-wide with:

```
openpbs-scyld.setup cluster-stop
openpbs-scyld.setup cluster-start
```

OpenPBS executable commands and libraries are installed in `/opt/scyld/openpbs/`. Each OpenPBS user must set up the `PATH` and `LD_LIBRARY_PATH` environment variables to properly access the OpenPBS commands. This is done automatically for users who login when OpenPBS is running via the `/etc/profile.d/scyld.openpbs.sh` script. Alternatively, each OpenPBS user can manually execute `module load openpbs` or can add that command line to (for example) the user's `~/.bash_profile` or `~/.bashrc`.

3.10.2.3 PBS TORQUE

PBS TORQUE is only available for RHEL/CentOS 7 clusters. See *Job Schedulers* for general job scheduler information and configuration guidelines. See <https://www.adaptivecomputing.com/support/documentation-index/torque-resource-manager-documentation> for PBS TORQUE documentation.

First install PBS TORQUE software on the job scheduler server:

```
sudo yum install torque-scyld --enablerepo=scyld*
```

Now use a helper script `torque-scyld.setup` to complete the initialization and setup the job scheduler and config file in the compute node image(s).

Note

The `torque-scyld.setup` script performs the `init`, `reconfigure`, and `update-nodes` actions (described below) by default against all *up* nodes. Those actions optionally accept a node-specific argument using the syntax `[--ids|-i <NODES>]` or a group-specific argument using `[--ids|-i %<GROUP>]`. See *Attribute Groups and Dynamic Groups* for details.

```
torque-scyld.setup init                # default to all 'up' nodes
torque-scyld.setup update-image torqueImage # for permanence in the image
```

Reboot the compute nodes to bring them into active management by TORQUE. Check the TORQUE status:

```
torque-scyld.setup status

# If the TORQUE daemon is not executing, then:
torque-scyld.setup cluster-restart

# And check the status again
```

This `cluster-restart` is a manual one-time setup that doesn't affect the *torqueImage*. The `update-image` is necessary for persistence across compute node reboots.

Generate new torque-specific config files with:


```
torque-scyld.setup reconfigure      # default to all 'up' nodes
```

Add nodes by executing:

```
torque-scyld.setup update-nodes    # default to all 'up' nodes
```

or add or remove nodes by directly editing the `/var/spool/torque/server_priv/nodes` config file. Any such changes must be added to *torqueImage* by reexecuting:

```
torque-scyld.setup update-image slurmImage
```

and then either reboot all the compute nodes with that updated image, or additionally execute:

```
torque-scyld.setup cluster-restart
```

to manually push the changes to the *up* nodes without requiring a reboot.

Inject users into the compute node image using the `sync-uids` script. The administrator can inject all users, or a selected list of users, or a single user. For example, inject the single user *janedoe*:

```
/opt/scyld/clusterware-tools/bin/sync-uids \  
    -i torqueImage --create-homes \  
    --users janedoe --sync-key janedoe=/home/janedoe/.ssh/id_rsa.pub
```

See *Configure Administrator Authentication* and `/opt/scyld/clusterware-tools/bin/sync-uids -h` for details.

To view the TORQUE status on the server and compute nodes:

```
torque-scyld.setup status
```

The TORQUE service can also be started and stopped cluster-wide with:

```
torque-scyld.setup cluster-stop  
torque-scyld.setup cluster-start
```

TORQUE executable commands are installed in `/usr/sbin/` and `/usr/bin/`, TORQUE libraries are installed in `/usr/lib64/`, and are therefore accessible by the default search rules.

3.10.3 Kubernetes

ICE ClusterWare™ administrators who want to use Kubernetes as a container orchestration layer across their cluster can either choose to install Kubernetes manually following directions found online or use scripts provided by the *clusterware-kubeadm* package to install and bootstrap Kubernetes clusters.

The provided scripts are based on the `kubeadm` tool and inherit both the benefits and limitations of that tool. If you prefer to use a different tool to install Kubernetes, follow appropriate directions available online from your chosen Kubernetes provider.

ClusterWare nodes and non-ClusterWare systems can be joined into the same Kubernetes cluster when the servers are on the same network.

- To use *clusterware-kubeadm* scripts on ClusterWare nodes, install the *clusterware-kubeadm* package on a server that a ClusterWare admin can use to access those nodes from `scyld-nodectl`. Use the following command to install:

```
sudo yum --enablerepo=scyld* install clusterware-kubeadm clusterware-tools
```

- To use *clusterware-kubeadm* scripts on non-ClusterWare servers, install the *clusterware-kubeadm* package on all of those servers. The scripts will be run from each of those servers locally. Use the following command to install:

```
sudo yum --enablerepo=scyld* install clusterware-kubeadm
```

After installing the software, a ClusterWare admin or a root user on a non-ClusterWare system can use the *scyld-kube* tool to install the Kubernetes cluster. The default kubernetes version is hardcoded in `/opt/scyld/clusterware-kubeadm/files/core/etc/yum.repos.d/kubernetes.repo.default` and has been tested. If you want to install any other version of Kubernetes, you can append a specific version (major.minor.patch) argument to *scyld-kube*. For example:

```
scyld-kube --version 1.31.1
```

Two Kubernetes control plane configuration options are supported: a single Kubernetes control plane node or a High Available (with HAProxy and Keepalived) Kubernetes control plane with a first and additional control nodes. Both configurations can have additional workers (non-controller nodes).

Important

For a server to function as a Kubernetes control plane or worker, swap must be turned off. Verify current status with `swapon -s`. Use `swapoff -a -v` to disable swap. You should not use a RAM-booted or otherwise ephemeral compute node as Kubernetes control plane.

The following sections include an example of a single ClusterWare control pane node plus ClusterWare nodes as workers. See [Using Kubernetes](#) for additional examples, including non-ClusterWare systems and multiple control plane nodes.

3.10.3.1 Bootstrap Kubernetes Control Plane

Initialize the control plane node(s).

For a single node control plane:

- For a single ClusterWare node control plane, use the following command:

```
scyld-kube -i <control plane node ID> --init
```

- For a single non-ClusterWare node control plane, use the following command:

```
scyld-kube --init
```

For a multi-node control pane:

- For a ClusterWare multi-node control plane, use the following commands on a ClusterWare admin node:

```
$scyld-kube --prepare-lb <unused IP> <first control plane node ID>:<node IP>,
↔<additional control plane node ID>:<node IP>,<additional control plane node ID>:
↔<node IP>
$scyld-kube -i <first control plane node ID> --init-ha
```

- For non-ClusterWare multi-node control plane, use the following commands on the first control plane system:

```
$scyld-kube --prepare-lb <unused IP> <first control plane node ID>:<node IP>,
↪<additional control plane node ID>:<node IP>,<additional control plane node ID>:
↪<node IP>
$scyld-kube --init-ha
```

Example

Run the following command to initialize ClusterWare node n0 (10.154.1.100) as a control plane node:

```
scyld-kube -i n0 --init
```

Messages about joining ClusterWare NODES/IMAGE and non-ClusterWare system as workers to this ClusterWare control plane are printed out after a successful initialization. For example:

```
...
To join ClusterWare NODES/IMAGE as worker to this Clusterware control plane:
scyld-kube -i NODES --join --cluster n0
scyld-kube --image IMAGE --join --cluster n0

To join non ClusterWare system as worker to this ClusterWare control plane:
scyld-kube --join --token yp6lxa.wcb6g48ud3f2cwng --cahash_
↪sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --cluster 10.
↪154.1.100
...
```

3.10.3.2 Checking Deployment Status

Verify that Kubernetes is ready on each system after the first initialization. Verify again after each control plane node or worker node joins.

- For a ClusterWare control plane, use the following command:

```
scyld-nodectl -i <node ID> exec kubectl get nodes -o wide
```

- For a non-ClusterWare control plane, use the following command:

```
kubectl get nodes -o wide
```

Example

The following example shows the Kubernetes cluster has ClusterWare n0 as a working control plane.

```
[admin@cwhead ~]$ scyld-nodectl -i n0 exec kubectl get nodes -o wide
NAME                STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP_
↪ OS-IMAGE                KERNEL-VERSION   CONTAINER-RUNTIME
n0.cluster.local    Ready    control-plane  1d    v1.31.2   10.154.1.100  <none>
↪ Rocky Linux 8.10 (Green Obsidian)  4.18.0-553.el8_10.x86_64   containerd://1.6.32
```

3.10.3.3 Additional Configuration

Depending on your ClusterWare cluster configuration, the INTERNAL-IP of the ClusterWare control plane may not match the IP address known to the ClusterWare platform. If they are different, replace the `--cluster` value with the INTERNAL-IP value when using the printed out messages to join additional control plane nodes and workers. In the

example, the INTERNAL-IP of the ClusterWare control plane is 10.154.1.100, which is the same as n0's IP address known to the ClusterWare platform.

If you are joining additional control plane nodes, you may need to generate a new certificate key because the one printed in the output expires in 2 hours.

For a ClusterWare control plane, use the following command to generate a new key:

```
[adminr@cwhead ~]$ scyld-nodectl -i <node ID> exec kubeadm init phase upload-certs --
↳upload-certs
[upload-certs] Storing the certificates in Secret "kubeadm-certs" in the "kube-system"
↳Namespace
[upload-certs] Using certificate key:
ad556dcd5c795a42321be46b0a3cf8a52d7a1c7fef6e0bd96c65525569c39105
```

On a non-ClusterWare control plane, use the following command:

```
[root@kube1 ~]$ kubeadm init phase upload-certs -upload-certs
```

Replace the `--certificate-key` value with the new certificate key you just generated when using the output messages to join additional control plane nodes.

3.10.3.4 Adding Workers

1. Using the messages output after initialization as a guide, join workers to the control plane.

To join ClusterWare nodes as workers to a ClusterWare control plane:

```
scyld-kube -i n[<node IDs>] --join --cluster <control plane node ID>
```

To join ClusterWare nodes as workers to a non-ClusterWare control plane:

```
scyld-kube -i n[<node IDs>] --join --token <token value> --cahash <cahash value> --
↳cluster <control plane IP>
```

To join non-ClusterWare systems as workers to a ClusterWare control plane:

```
scyld-kube --join --token <token value> --cahash <cahash> --cluster <control plane
↳IP>
```

To join non-ClusterWare systems as workers to a non-ClusterWare control plane:

```
scyld-kube --join --token <token value> --cahash <cahash value> --cluster <control
↳plane IP>
```

2. For ClusterWare workers, use the following commands to create a Kubernetes worker node image and then boot the nodes with the node image as workers:

```
$ scyld-bootctl -i DefaultBoot clone name=<boot name>
$ scyld-imgctl -i DefaultImage clone name=<image name>
$ scyld-kube --image <image name> --join --cluster <control plane node ID>
$ scyld-bootctl -i <boot name> up image=<image name>
$ scyld-nodectl -i n[5-10] set _boot_config=<boot name>
$ scyld-nodectl -i n[5-10] reboot
```

Example

For the single ClusterWare control plane example, the following messages are printed out after the control plane initialization:

```
...
To join ClusterWare NODES/IMAGE as worker to this Clusterware control plane:
scyld-kube -i NODES --join --cluster n0
scyld-kube --image IMAGE --join --cluster n0

To join non ClusterWare system as worker to this ClusterWare control plane:
scyld-kube --join --token yp6lxa.wcb6g48ud3f2cwng --cahash
↳ sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --cluster 10.
↳ 154.1.100
...
```

1. Using the message output after initialization as a guide, join ClusterWare nodes (n[1-4]) as workers to the control plane node n0 with the following command:

```
$ scyld-kube -i n[1-4] --join --cluster n0
```

2. Create a Kubernetes worker node image and then boot n[5-10] with the node image as workers to control plane n0:

```
$ scyld-bootctl -i DefaultBoot clone name=KubeWorkerBoot
$ scyld-imgctl -i DefaultImage clone name=KubeWorkerImage
$ scyld-kube --image KubeWorkerImage --join --cluster n0
$ scyld-bootctl -i KubeWorkerBoot up image=KubeWorkerImage
$ scyld-nodectl -i n[5-10] set _boot_config=KubeWorkerBoot
$ scyld-nodectl -i n[5-10] reboot
```

3. Verify that Kubernetes is ready on each system using the following command:

```
$ scyld-nodectl -i n0 exec kubectl get nodes -o wide
NAME                STATUS    ROLES    AGE   VERSION   INTERNAL-IP  └
↳ EXTERNAL-IP      OS-IMAGE                KERNEL-VERSION  └
↳ CONTAINER-RUNTIME
n0.cluster.local    Ready    control-plane    1d    v1.31.2   10.154.1.100  <none> └
↳ Rocky Linux 8.10 (Green Obsidian)  4.18.0-553.el8_10.x86_64 └
↳ containerd://1.6.32
n1.cluster.local    Ready    <none>          1d    v1.31.2   10.154.1.101  <none> └
↳ Rocky Linux 8.10 (Green Obsidian)  4.18.0-553.el8_10.x86_64 └
↳ containerd://1.6.32
```

The example output shows the Kubernetes cluster has ClusterWare n0 as a working control plane and n1 as a worker.

scyld-kube

NAME

scyld-kube -- Tool for installing Kubernetes control plane and worker nodes.

USAGE

scyld-kube

```
[-h | --help]  [--init]  [--join]  [--init-ha]  [--join-ha]  [--version VERSION]
```

```

[--prepare-lb OPTIONS]  [--cluster [IP | NODE]]  [--image IMAGE]  [--token TOKEN]
[--cahash CAHASH]  [--certificate-key KEY]  [[-i | --ids] NODES]  [--all]  [--up]
[--core-inst]

```

DESCRIPTION

To administer Kubernetes in a cluster, install the *clusterware-kubeadm* package on either a ClusterWare admin node, a full-install ClusterWare compute node, or a separate non-ClusterWare server. This package contains the *scyld-kube* tool.

STANDARD OPTIONS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
 - init** Initialize the **only** control plane node.
 - join** Add worker node.
 - init-ha** Initialize the first control plane node for High Availability (HA).
 - join-ha** Add additional control plane node(s) for High Availability (HA).
 - version VERSION** Optionally specify a *major.minor.patch* version for an `--init`, `--init-ha`, `--join`, or `--join-ha`.
- prepare-lb *APISERVER_VIP[:APISERVER_PORT:ROUTER_ID:AUTH_PASS]*
FIRST_CONTROL_PLANE_NODE_ID:FIRST_CONTROL_PLANE_NODE_IP, **ADDI-**
*ADDITIONAL_CONTROL_PLANE_NODE_ID1:ADDITIONAL_CONTROL_PLANE_NODE_IP1,...***
- Re-generate High Availability (HA) load balancer config files from local template.
- APISERVER_VIP***
 Required. The virtual IP address of the Kubernetes API server within the network subnet.
- APISERVER_PORT***
 Optional. An unused port for the API server, default 4200.
- Note: All three optional values must be specified if non-Default values are to be used.
- ROUTER_ID:***
 Optional. Default 51.
- Note: All three optional values must be specified if non-Default values are to be used.
- AUTH_PASS:***
 Optional. Default 42
- Note: All three optional values must be specified if non-Default values are to be used.
- FIRST_CONTROL_PLANE_NODE_ID:FIRST_CONTROL_PLANE_NODE_IP,* *ADDI-*
*ADDITIONAL_CONTROL_PLANE_NODE_ID1:ADDITIONAL_CONTROL_PLANE_NODE_IP1,...***
- A comma- and a colon-separated list of first and additional hosts' unique ID (can be host-names) and IP addresses.
- cluster [*IP* | *NODE*]**
 Specify the cluster (first control plane node); use with `--join` or `--join-ha`.
 - image IMAGE** Optionally modify the specified image for persistence across compute node reboots, so that nodes with this *IMAGE* auto-join when booted.
 - token TOKEN** Optionally specify first control plane node's token; use with `--join` or `--join-ha`.

--cahash CAHASH Optionally specify first control plane node's discover-token-ca-cert-hash; use with `--join` or `--join-ha`.

--certificate-key KEY Optionally specify first control plane node's certificate-key; use with `--join-ha`.

CLUSTERWARE NODE SELECTION OPTIONS

-i, --ids NODES A comma-separated list of nodes or an admin-defined group of nodes to act upon.

--all Configure all nodes (rare).

--up Configure all *up* nodes (rare).

ADVANCED OPTIONS

--core-inst Only install core packages.

EXAMPLES

RETURN VALUES

Upon successful completion, **scylld-kube** returns 0. On failure, an error message is printed to `stderr` and **scylld-kube** returns 1.

3.10.4 OpenMPI, MPICH, and/or MVAPICH

The ICE ClusterWare™ platform distributes several versions of OpenMPI, MPICH, and MVAPICH2, and other versions are available from 3rd-party providers. Different versions of the ClusterWare packages can coexist, and users can link applications to the desired libraries and execute the appropriate binary executables using `module load` commands. Typically one or more of these packages are installed in the compute node images for execution, as well as on any other server where OpenMPI (and similar) applications are built.

View the available ClusterWare versions using:

```
yum clean all      # just to ensure you'll see the latest versions
yum list --enablerepo=scylld* | egrep "openmpi|mpich|mvapich" | egrep scylld
```

The OpenMPI, MPICH, and MVAPICH packages are named by their major-minor version numbers, e.g., 4.0, 4.1, and each has one or more available major-minor "point" releases, e.g., *openmpi4.1-4.1.1* and *openmpi4.1-4.1.4*.

A simple `yum install` will install the latest "point" release for the specified major-minor version, e.g.:

```
sudo yum install openmpi4.1 --enablerepo=scylld*
```

installs the default GNU libraries, binary executables, buildable source code for various example programs, and man pages for *openmpi4.1-4.1.4*. The *openmpi4.1-gnu* packages are equivalent to *openmpi4.1*.

Alternatively or additionally:

```
sudo yum install openmpi4.1-intel --enablerepo=scylld*
```

installs those same packages built using the Intel oneAPI compiler suite. These compiler-specific packages can co-exist with the base GNU package. Similarly you can additionally install *openmpi4.1-nvhpc* for libraries and executables built using the Nvidia HPC SDK suite, and *openmpi-aocc* for libraries and executables built using AMD Optimizing C/C++ and Fortran Compilers. Additionally *openmpi4.1-hpcx_cuda- $\{compiler\}$* rpms sets are built against Nvidia HPC-X and cuda software packages and with `gnu`, `intel`, `nvhpc` and `aocc` compilers.

Note

The ClusterWare platform provides `openmpi` packages that are built with third party software and compilers with best effort. However if an `openmpi` rpm of a certain combination of compiler, software, OpenMPI version and distro is missing, that is because that combination failed to build or package failed to run. Also, the third party software and compilers that are needed for those OpenMPI packages must be installed in addition to clusterware installation.

Important

The ClusterWare yum repo includes various versions of `openmpi*` RPMs which were built with different sets of options by different compilers, each potentially having requirements for specific other 3rd-party packages. In general, avoid installing `openmpi` RPMs using a wildcard such as `openmpi4*scyld` and instead carefully install only specific RPMs from the ClusterWare yum repo together with their specific required 3rd-party packages.

Suppose `openmpi4.1-4.1.1` is installed and you see a newer "point" release `openmpi4.1-4.1.4` in the repo. If you do:

```
sudo yum update openmpi4.1 --enablerepo=scyld*
```

then `4.1.1` updates to `4.1.4` and removes `4.1.1`. Suppose for some reason you want to retain `4.1.1`, install the newer `4.1.4`, and have both "point" releases coexist. For that you need to download the `4.1.4` RPMs and install (not update) them using `rpm`, e.g.,

```
sudo rpm -iv openmpi4.1-4.1.4*
```

You can add OpenMPI (*et al*) environment variables to a user's `~/.bash_profile` or `~/.bashrc` file, e.g., add `module load openmpi/intel/4.1.4` to default a simple OpenMPI command to use a particular release and compiler suite. Commonly a cluster uses shared storage of some kind for `/home` directories, so changes made by the cluster administrator or by an individual user are transparently reflected across all nodes that access that same shared `/home` storage.

For OpenMPI, consistent user uid/gid and passphrase-less key-based access is required for a multi-threaded application to communicate between threads executing on different nodes using `ssh` as a transport mechanism. The administrator can inject all users, or a selected list of users, or a single user into the compute node image using the `sync-uids` script. See [Configure Administrator Authentication](#) and `/opt/scyld/clusterware-tools/bin/sync-uids -h` for details.

To use OpenMPI (*et al*) without installing either `torque-scyld` or `slurm-scyld`, then you must configure the firewall that manages the private cluster network between the head node(s), server node(s), and compute nodes. See [Firewall Configuration](#) for details.

ADMINISTRATION

The *Overview* describes the ICE ClusterWare™ system architecture and design and basic terminology necessary to properly configure and administer a ClusterWare cluster.

This ClusterWare Administration Guide is intended for use by ClusterWare administrators and advanced users. As is typical for any Linux-based system, the administrator must have root privileges (if only via *sudo*) to perform many of the administrative tasks described in this document.

Note

This guide is written with the assumption that the administrator has a background in a Unix or Linux operating environment; therefore, the document does not cover basic Linux system administration. If you do not have sufficient knowledge for using or administering a Linux system, we recommend that you first study other resources, either in print or online.

4.1 Introduction

This guide provides specific information about tools and methods for maintaining the ICE ClusterWare™ cluster, ways to control cluster usage, methods for batching jobs and controlling the job queue, how load balancing is handled in the cluster, and optional tools that can be useful in administrating your cluster.

Both the ClusterWare graphical user interface (GUI) and the command-line interfaces employ the same underlying interfaces to the ClusterWare database. When applicable, instructions are given for completing tasks in both the command line and the GUI.

Tip

The ClusterWare CLI Cheat Sheet provides a quick reference for common commands and arguments. See <https://docs.ice.penguinsolutions.com/clusterware-cli-cheatsheet-12.4.pdf> for details.

4.2 ICE ClusterWare Graphical User Interface

The ICE ClusterWare™ graphical user interface (GUI) is available on all head nodes using a browser to access `http://<HEADNODE_IP>`. The default authentication is done through PAM, so cluster administrators can use their existing credentials for the head node.

The ClusterWare GUI opens to an **Overview** page. Use the left navigation panel to access detailed pages for nodes, boot configurations, user management, and so on.

The following pages are available:

- [Cluster Overview Page](#)
- [Head Nodes Page](#)
- [Dynamic Groups Page](#)
- [Nodes Page](#)
- [Node Attributes Page](#)
- [Naming Pools Page](#)
- [Networks Page](#)
- [Hostnames Page](#)
- [Boot Configurations Page](#)
- [Images Page](#)
- [Image Sources Page](#)
- [Git Repos Page](#)
- [Grafana Login](#)
- [Users and Groups Page](#)

The Grafana Monitoring Dashboard (see [Grafana Telemetry Dashboard](#)) is also available on all head nodes, either by clicking on the **Telemetry Dashboard** link in the left navigation pane or by accessing `http://<HEADNODE_IP>/grafana`. The Grafana default credentials are the username "admin" and the `database.admin_pass` from the `base.ini`:

```
sudo grep admin_pass /opt/scyld/clusterware/conf/base.ini
```

Administrators can change the password within the Grafana graphical interface.

4.3 ICE ClusterWare Command Line Tools

This section describes the commonly used arguments and subcommands used by the various ICE ClusterWare™ tools. These tools can be used by the cluster administrator and are not intended for use by the ordinary user.

Tip

The ClusterWare CLI Cheat Sheet provides a quick reference for common commands and arguments. See <https://docs.ice.penguinolutions.com/clusterware-cli-cheatsheet-12.4.pdf> for details.

Certain arguments are shared among nearly all the `scyld-*ctl` tools, and instead of repeatedly describing these arguments, we will cover them here. Many of these arguments control the general operation of the tools, i.e. by printing help (`--help` or `-h`), selecting targets (`--all` or `-a`, `--ids` or `-i`), changing the verbosity or client configuration (`--verbose` or `-v`, `--quiet` or `-q`, `--config` or `-c`), allowing a user to override basic connection details (`--base-url`, `--user` or `-u`), or changing output formatting (`--show-uids`, `--human`, `--json`, `--pretty` or `--no-pretty`). Many of these arguments are self-explanatory, but others are described below:

4.3.1 `--all` and `--ids`

Tools that accept the `--all` (short name `-a`) and `--ids` (short name `-i`) arguments operate on corresponding database objects. For instance, `scyld-nodectl` is used for manipulating node objects in the database, and `scyld-attribctl` is used for manipulating attribute groups.

As one might expect, `--all` can be used to make an alteration to all of a given class of objects at once. For example, to remove a given attribute such as `_boot_style` from all attribute groups, e.g.:

```
scyld-attribctl --all clear _boot_style
```

Alternatively, an administrator can specify objects by name, or UID, or truncated UID (at least the first 5 characters of the UID are required to reduce the chance of accidental selection). Certain object types can also be selected based on some core fields, e.g. MAC, IP, or index for nodes. Further, nodes can be selected using the node query language, e.g.:

```
scyld-nodectl --ids n[0-5] --ids 08:00:27:F0:44:35 ls
```

For convenience, many tools can be executed without explicitly selecting any objects. Specifically, query tools such as `list` will default to `--all` if no selection arguments are used, and many other tools will operate on a single object if only one object of the expected type exists in the system.

4.3.2 `--config`

All client tools accept a `--config` argument which can be used to specify a client INI file. By default several locations are checked for configuration INI files with each able to override variables from the previous files. The client configuration search order is:

- `/etc/scyldcw/settings.ini`
- `/etc/scyldcw/${TOOL}.ini`
- `~/scyldcw/settings.ini`
- `~/scyldcw/${TOOL}.ini`
- the `--config` specified path

- command line arguments

These configuration files should be INI formatted, and the `[ClusterWare]` section can contain the following variables:

```
client.base_url = http://localhost/api/v1
client.sslverify = True
client.authuser = $USER
client.authpass = None
client.format = human
client.pretty = False
```

The `base_url` specifies the URL that the tools should use to connect to the head node's REST API and defaults to connecting to the standard location (`http://localhost/api/v1`) on the local machine. If the `base_url` specifies an HTTPS URL, then a client can disable SSL verification, but this is strongly discouraged as it bypasses the protections provided by HTTPS against impersonation and man-in-the-middle attacks. The `authuser` and `authpass` can be included to simplify authentication to the service, but be aware that specifying the `authpass` here may not be secure, depending on your environment.

The `format` argument affects the output format of data returned by the tools. The default value of "human" causes the tools to output an indented format with various computed values augmented with human-readable summaries. The alternative value of "json" will output the results as JSON formatted text, and the `pretty` argument can be used to turn on indentation for that JSON output.

4.3.3 --base-url and --user

Since an administrator may want to periodically connect to different head nodes or as a different user, command line arguments are provided to override those configuration settings. For example, the entire string passed to the `--base-url` argument is treated as a URL and is passed to the underlying Python requests library.

Any string passed to the `--user` argument will be split at its first colon, and the remainder of the string will be treated as the user's password. Providing a password this way is convenient, especially during testing, but is generally discouraged as the password could then be visible in `/proc` while the tool is running. If no password is provided either through command line or client configuration, then one will be requested when needed.

4.3.4 --show-uids, --human, --json, --pretty/--no-pretty

These arguments are used to change the tool output format, much like the corresponding client configuration variables described above. The `--human` and `--json` arguments override the `client.format` variable, and `--pretty` and `--no-pretty` can be used to override the `client.pretty` variable.

By default, tool output will show an object's name when referring to a named object, and the UID (or shortened UID) only if no name is defined. Using the `--show-uids` argument forces the display of full UIDs in place of more human-readable options. This is uglier, but occasionally useful to be absolutely certain about what object is being referenced.

4.3.5 --csv, --table, --fields

For ease of reading and automated parsing, the scyld tools can also produce output as CSV or in a table. Use the `--fields` argument to select fields to display and select from `--csv` or `--table` to print in your preferred format:

```
$ scyld-nodectl --fields "mac,Assigned IP=ip,BootConfig=attributes._boot_config" \
  --table ls -l
Nodes |          mac | Assigned IP | BootConfig
-----+-----+-----+-----
n0 | 08:00:27:f0:44:35 | 10.10.24.100 | DefaultBoot
n1 | 08:00:27:a2:3f:c9 | 10.10.24.101 | DefaultBoot
n2 | 08:00:27:e5:19:e5 | 10.10.24.102 | DefaultBoot
```

The above demonstrates how to both assign column names and select nested values such as individual attributes.

4.4 Common Subcommand Actions

In addition to the above arguments, some subcommand actions are common among the `scyld-*ctl` tools as well: `list`, `create`, `clone`, `update`, `replace`, `delete`. The precise details of what additional arguments these subcommand actions accept may differ between tools, but the generally supported arguments are discussed here.

4.4.1 list (ls)

List the requested object names, and optionally with `--long` or `-l` will display object details. The `--raw` option will display the actual JSON content as returned by the ClusterWare API call.

4.4.2 create (mk)

Create a new object using name-value pairs provided either on the command line or passed using the `--content` argument described below.

4.4.3 clone (cp)

Copy existing objects to new UIDs and names. Individual fields in the new objects can be overridden by name-value or a `--content` argument described below.

4.4.4 update (up)

Modify existing objects altering individual fields in name-value pairs or a `--content` argument described below.

4.4.5 replace (re)

Much like `update`, but completely replace the existing objects with new objects from fields defined in name-value pairs or a `--content` argument described below.

4.4.6 delete (rm)

Delete objects.

4.5 Files in database objects

In the ClusterWare database, boot configurations and images both contain references to files, either a kernel and an `initramfs`, or a root file system. The files themselves are not stored in the database but instead are referenced by the system on backend storage through plugins, such as the `local_files` plugin that works with locally mounted storage through the POSIX API.

When listing the details of a database object containing a file reference, the reference will be shown as a dictionary containing the file size, modification time, checksum, and an internal UID. To explore this we will start by listing a boot configuration created earlier in this document:

```
$ bin/scyld-bootctl ls -l
Boot Configurations
  TestBoot
    initramfs
      checksum: aa1161aa52b98287a3eac4677193c141a3648ebc
      mtime: 2019-02-09 21:13:08 UTC (0:00:52 ago)
```

(continues on next page)

(continued from previous page)

```

size: 20.0 MiB (20961579 bytes)
uid: d247e4aa1fde4ac3853e78c0f7683947
kernel
  chksum: 5a464d2a82839dac21c0fb7350d9cb0d055f8fed
  mtime: 2019-02-09 21:08:34 UTC (0:05:26 ago)
  size: 6.3 MiB (6639808 bytes)
  uid: fc96da5531b94038b38d7ef662b34947
last_modified: 2019-02-09 21:08:34 UTC (0:05:26 ago)
name: TestBoot
release: 3.10.0-957.1.3.el7.x86_64
uid: 4978077e53b944b38c4cda007b9b97b7

```

Files have been uploaded to both the `initramfs` and `kernel` fields. The `chksum` fields are the SHA-1 output and are used to detect data corruption, not as a security feature. The `mtime` is the UTC timestamp of the last time the underlying file was modified, and the `size` field is the size of the file in bytes. The `uid` field is how the object is referenced within the ClusterWare system and is the name passed to whatever plugin is interfacing with underlying storage. In the case of the `local_files` plugin, this is used as the name of the file on disk.

Because this output was generated for human readability, some fields (`last_modified`, `mtime`, `size`) have been augmented with human readable representations. Also, the `release` field was determined by examining the contents of the kernel file when it was uploaded.

4.6 The `then` argument

Various tools (most commonly `scyld-nodectl`, although also `scyld-adminctl`, `scyld-attribctl`, `scyld-bootctl`, and `scyld-imgctl`) accept the `then` argument, which serves as a divider of a serial sequence of multiple subcommands for a single invocation of the `scyld-*` tool.

For example,

```
scyld-nodectl -i n0 reboot then waitfor up then exec uname -r
```

that initiates a reboot of node `n0`, waits for the node to return to an "up" state, and then executes `uname -r` on the node.

If any subcommand in the sequence fails, then the tool reports the error, skips any subsequent subcommands, and terminates.

4.7 The `--content` argument

The `--content` argument can be passed to several of the tools described earlier and is always paired with an argument to accept name-value pairs that can override content values. The `--content` argument can be followed by a JSON string or by a file containing JSON formatted data, INI formatted data, or a text file where each object is represented by rows of name-value pairs. If the argument to `--content` is a filename, it must be prefixed with an '@' symbol.

For example, an administrator could create a new boot configuration as follows:

```
scyld-bootctl create --content \
  '{"name": "TestBoot", "kernel": "@/boot/vmlinuz-3.10.0-957.1.3.el7.x86_64"}'
```

Of course, a boot configuration also requires an `initramfs`:

```
cat > content.ini <<EOF
[BootConfig]
```

(continues on next page)

(continued from previous page)

```
initramfs: @initramfs-3.10.0-957.1.3.el7.x86_64.img
EOF

scylid-bootctl -iTestBoot update --content @content.ini
```

Adding nodes to the database one at a time is tedious for large clusters, and the `--content` argument can streamline this process. Below are examples of three different files that could be passed via the `--content` argument to add nodes with explicit indices to the database:

JSON:

```
[
  { "mac": "00:11:22:33:44:55", "index": 1 },
  { "mac": "00:11:22:33:44:66", "index": 2 },
  { "mac": "00:11:22:33:44:77", "index": 3 },
]
```

INI:

```
[Node0]
mac: 00:11:22:33:44:55
index: 1

[Node1]
mac: 00:11:22:33:44:66
index: 2

[Node2]
mac: 00:11:22:33:44:77
index: 3
```

Text:

```
mac=00:11:22:33:44:55 index=1
mac=00:11:22:33:44:66 index=2
mac=00:11:22:33:44:77 index=3
```

Although providing multiple objects at once makes sense for the `create` subcommand, the `clone`, `update`, and `replace` subcommands require a list of fields to alter and will collapse multiple objects into one set of variables. For example:

```
[
  { "name": "TestBoot" },
  { "kernel": "@/boot/vmlinuz-3.10.0-957.1.3.el7.x86_64" },
  { "name": "AnotherBoot" }
]
```

when passed to `scylid-bootctl` would result in the selected boot configuration(s) being renamed to "AnotherBoot" and assigned the `/boot/vmlinuz-3.10.0-957.1.3.el7.x86_64` kernel.

4.8 Variable Substitution

Data that is downloaded or processed by a ICE ClusterWare™ head node can often include per-node modifications through a variable substitution framework. Much like a templating system, an admin can create a baseline file, e.g. a kickstart file, and have the ClusterWare platform automatically replace entries with node- or head-specific data. Thus, one can dynamically include the node's name or IP address, or a head-node's base URL or key information.

```
# example kickstart fragment

# Perform a SOL friendly text-based install.
text

# Pull some basics from the head node.
url --url <root_url()>
lang <head[lang]>
keyboard <head[keymap]>
timezone <head[timezone]>

# include another file
<include(partial.ks)>
```

The ClusterWare platform currently allows variable substitution in kickstart files, ZTP scripts and configuration files, and the `power_uri` field for a node. The goal of these substitutions is to provide mechanisms to generalize the files for simpler configuration with fewer node-specific files or commands.

The `<include(partial.ks)>` tag shown in the example includes another file into the output, performing variable substitution on the included content as well. The `include` tag allows a cluster administrator to break larger files into manageable hunks that can then be included into a top-level kickstart file, much like a C or C++ `#include <filename>`.

4.8.1 Node Attributes, Hardware, and Status

Any node's attribute can be referenced as `<a[name]>` or `<attributes[name]>` and that text will be replaced with the value corresponding to the “name”. Similarly, values from the hardware section of `scylid-nodectls -L` can be referenced as `<h[name]>` or `<hardware[name]>` and status information is referenced as `<s[name]>` or `<status[name]>`. Fields outside of attributes and hardware, such as index, ip, or MAC can be referenced by `<n[name]>` or `<node[name]>`.

4.8.2 Head Node Substitutions

In addition to compute node-specific fields, a few head node-specific fields are also available in kickstart files:

<code>head[keymap]</code>	Kickstart keymap arguments like <code>--vckeymap=X --xlayouts=Y</code>
<code>head[lang]</code>	System locale
<code>head[timezone]</code>	Time zone

When used within downloadable text files, such as kickstart and ZTP files, a few other parameters are available as well:

<code>param[X]</code>	X is a parameter provided as part of the requesting URL
<code>cw[base_ur]</code>	A base URL that should be functional for the requestor
<code>cw[head]</code>	IP address of the parent head node extracted from the base URL
<code>cw[keys]</code>	System-wide authorized keys list including head node keys and cluster administrator keys. Suitable for appending to <code>.ssh/authorized_keys</code>

4.8.3 Kickstarting From A Repo

When downloading a kickstart file from a repo-based boot configuration, a URL parameter is added that references the repo, allowing for additional substitutions. In this case the substitutions are implemented as functions with a slightly different syntax:

<code><root_url()></code>	Link to the root of the full ISO file if the repo contains one
<code><iso_url()></code>	Link to the full ISO file if the repo contains one
<code><repo_url()></code>	Rarely if ever used. Refers to a specific URL in a ClusterWare repo

Note that all of these functions accept a repo name or UID as an argument but will use the automatically provided URL parameter if no repo is explicitly specified. The `<repo_url()>` function also includes a second optional integer argument to specify the index of the URL in the referenced repo.

4.9 Manage Cluster

4.9.1 Cluster Overview Page

After a successful login, the **Overview** page displays and presents the basic cluster health and status in summary panels. You can return to the **Overview** page using **Cluster > Overview** in the left navigation panel or by clicking the logo in the header bar.

At the top of the **Overview** page is a link to the locally installed ICE ClusterWare™ HTML documentation. The upper right of each panel includes an "i" icon, which takes you to the relevant contextual documentation for that panel.

A link at the lower right of each panel (labeled *Manage <panel-name>*), or the panel title in the left sidebar takes you to that panel's detailed information.

The **Refresh** control is available on all pages and controls how often the GUI retrieves database contents. You can change the time interval or disable automatic updates altogether.

The Overview's ClusterWare Disk Usage panel's *Manage Head Nodes* link (see [Head Nodes Page](#)) takes you to the **Head Nodes** page with disk usage and head node details.

Most of the more detailed pages show various entries (the lists of nodes, boot configurations, administrators, Git repositories, etc.) listed in a table. Table actions are available in the **More** menu, reachable by clicking on the ellipsis (...).

4.9.2 Cluster Settings

ICE ClusterWare™ cluster settings are managed via the *scyld-clusterctl* command line tool or via the **Cluster > Settings** page in the ClusterWare graphical user interface (GUI).

4.9.2.1 Cluster ID

The cluster ID is a serial number provided by Penguin Solutions when you purchase a ClusterWare license. The cluster ID is used to access the YUM repositories to download and install the ClusterWare product. You can pull your cluster ID from the repository on the **Settings** page of the ClusterWare GUI.

4.9.2.2 Cluster Name

You can provide a cluster name on the **Settings** page of the ClusterWare GUI. The name is for display purposes only and is not used in the command line tools.

4.9.2.3 Accepting Unknown Nodes

You can configure ClusterWare head nodes to either accept or deny access to new nodes that were not created with the ClusterWare command line or GUI. This functionality is disabled by default. To allow node creation with unknown MAC addresses, toggle the **Accept unknown nodes?** switch on the **Settings** page or run the following command:

```
scyld-clusterctl --set-accept-nodes True
```

For details, including how simultaneous node requests are handled, see *Node Creation with Unknown MAC address(es)*.

4.9.2.4 Time Offset

The **Settings** page displays a calculation of the time offset between the browser and the head node. A large time offset may require additional investigation. The time offset is not displayed in the command line, but may be manually calculated by reviewing the current time of a head node.

4.9.3 scyld-clusterctl

NAME

scyld-clusterctl -- Tool for manipulating global cluster settings.

USAGE

scyld-clusterctl

```
[-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
[--human | --json | --csv | --table] [--pretty | --no-pretty] [--fields FIELDS]
[--summary] [--get-group | --set-group ATTRIB_GROUP] [--get-naming | --set-naming
PATTERN] [--get-influx-token | --set-influx-token PATTERN] [--get-accept-nodes
| --set-accept-nodes T|F ] [--get-distro | --set-distro DISTRO] [--image-formats
] [--get-auth-config | --set-auth-config AUTHCFG] {repos, distros, heads, pools,
dyngroups, gitrepos, certs} ...
```

DESCRIPTION

Query and modify global cluster settings. This tool also includes commands for modifying the repositories and distributions used when making images, as well as commands to interact with cluster head nodes.

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]** Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

- human** Format the output for readability (default).
- json** Format the output as JSON.
- csv** Format the output as CSV.
- table** Format the output as a table.
- pretty** Indent JSON or XML output, and substitute human readable output for other formats.
- no-pretty** Opposite of `--pretty`.
- fields FIELDS** Select individual fields in the result or error.

CLUSTER-WIDE SETTINGS AND COMMANDS

- summary** Print a summary of the ClusterWare configuration.
- get-group** Print the default attribute group id.
- set-group ATTRIB_GROUP** Set the default attribute group.
- get-naming** Print the default node naming pattern.
- set-naming PATTERN** Set the default node naming pattern.
- get-influx-token** Print the InfluxDB API token.
- set-influx-token TOKEN** Set the InfluxDB API token.
- get-accept-nodes** Display whether or not unknown nodes should be automatically added.
- set-accept-nodes** Set whether unknown nodes should be automatically added (T=true) or not (F=false).
- get-distro** Get the current default distro.
- set-distro DISTRO** Set the default distro to *DISTRO*.
- image-formats** List image formats supported by the head node(s).
- get-auth-config** Print the authentication config.
- set-auth-config AUTHCFG** Set the authentication config.

DATABASE QUERYING AND MODIFICATION, SELECT A CLASS OF DATABASE OBJECT

repos *{list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm, download}*

Manipulate available repos using a subcommand:

list (ls): List information about repo(s).

create (mk): Add a repo.

clone (cp): Copy repo to new identifier.

update (up): Modify repo fields.

replace (re): Replace all repo fields. Deprecated in favor of "update".

delete (rm): Delete repo(s).

download: Download named files (any of 'iso').

-i, --ids REPOS A comma-separated list of repos to query or modify.

-a, --all Interact with all repos (default for *list*).

distros *{list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm, import}*

Manipulate available distros using a subcommand:

list (ls): List information about distro(s).

create (mk): Add a distro.

clone (cp): Copy distro to new identifier.

update (up): Modify distro fields.

replace (re): Replace all distro fields. Deprecated in favor of "update".

delete (rm) [-r, --recurse]: Delete distro(s).

-r, --recurse Optionally also delete any referenced repo.

import --name NAME [--release REL] FILE ...: Import one or more *FILE* repos into a distro *NAME*, and *REL* is an optional release string.

-i, --ids DISTROS A comma-separated list of distros to query or modify.

-a, --all Interact with all distros (default for *list*).

heads *[-i HEADS] [-a | --all] {list,ls, clean, service, delete,rm}*

Interact with cluster head nodes using a subcommand.

If multiple head nodes, then:

```
-i HEADS  A comma-separated list of head nodes to query or modify.
-a, --all Interact with all head nodes (default for *list* and *clean*).
```

list (ls): List information about services on the head node(s).

clean [ACTION]: Clean unreferenced objects from head node database, where *ACTION* is:

```
--all:      Trigger all implemented cleaning.
--files:    Delete any unknown files from storage.
--heads     Remove out-of-date head nodes.
--database  Scrub the database for broken references.
--dry-run   Take no action, but display what would be done.
(default)  --all --dry-run
```

delete (rm): Delete head nodes.

service [NAMES] [ACTION]: Interact with ClusterWare services (default: *list*), where *ACTION* is:

```
--start:   Start the service(s) NAMES.
--stop:    Stop the service(s) NAMES.
--restart: Restart the service(s) NAMES.
--enable:  Enable the service(s) NAMES.
--disable: Disable the service(s) NAMES.
```

pools *{list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm}*

Manipulate compute node name pools using a subcommand:

list (ls): List information about the name pools.

create (mk): Add a name pool.

clone (cp): Copy name pools to new identifiers.

update (up): Modify name pool fields.

replace (re): Replace all name pool fields. (Deprecated - use *update*.)

delete (rm): Delete name pools.

-i, --ids NAMINGPOOLS A comma-separated list of name pools to query or modify.

-a, --all Interact with all name pools (default for *list*).

dyngroups *{list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm, nodes}*

Manipulate dynamic groups using a subcommand:

list (ls): List information about the dynamic groups.

create (mk): Add a dynamic group.

clone (cp): Copy dynamic groups to new identifiers.

update (up): Modify dynamic group fields.

replace (re): Replace all dynamic group fields. (Deprecated - use *update*.)

delete (rm): Delete dynamic groups.

nodes: List nodes that currently meet the same selector.

-i, --ids DYNGROUPS A comma-separated list of dynamic groups to query or modify.

-a, --all Interact with all dynamic groups (default for *list*).

gitrepos *{list,ls, create,mk, clone,cp, update,up, delete,rm}*

Manipulate git repos using a subcommand:

list (ls): List information about the git repos.

create (mk): Add a git repo.

clone (cp): Copy git repos to new identifiers.

update (up): Modify git repo fields.

delete (rm): Delete git repos.

-i, --ids GITREPOS A comma-separated list of git repos to query or modify.

-a, --all Interact with all git repos (default for *list*).

certs *{list,ls, create,mk, clone,cp, update,up, delete,rm, assign}*

Manipulate certificate sources using a subcommand:

list (ls): List information about the certificate sources.

create (mk): Add a certificate source.

clone (cp): Copy certificate sources to new identifiers.

update (up): Modify certificate source fields.

delete (rm): Delete certificate sources.

assign: Assign the certificate sources to nodes, and create the certificates.

-i, --ids CERTS A comma-separated list of certificate sources to query or modify.

-a, --all Interact with all certificate sources (default for *list*).

hosts *[--show-uids] [-i HOSTNAMES | -a] {list,ls, create,mk, clone,cp, update,up, delete,rm}*

Manipulate hostname information for DNS, DHCP services using a subcommand:

list (ls): List information about the hostname.

create (mk): Add a hostname.

clone (cp): Copy hostnames to new identifiers.

update (up): Modify hostname fields.

delete (rm): Delete hostnames.

--show-uids Do not try to make the output more human readable.

-i, --ids HOSTNAMES A comma-separated list of hostnames to query or modify.

-a, --all Interact with all hostnames (default for *list*).

nets *{list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm}*

Manipulate available networks using a subcommand:

list (ls): List information about networks.

create (mk): Add a network.

clone (cp): Copy network to new identifiers.

update (up): Modify network fields.

delete (rm): Delete network(s).

-i, --ids NETWORKS A comma-separated list of networks to query or modify.

-a, --all Interact with all networks (default for *list*).

providers *{list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm, resources, alloc, release, attach}*

Manipulate available providers using a subcommand:

list (ls): List information about providers.

create (mk): Add a provider.

clone (cp): Copy providers to new identifiers.

update (up): Modify provider fields.

delete (rm): Delete provider(s).

resources: Show available resources, including unattached allocated systems.

alloc: Allocate machines according to the configuration.

release: Release machines according to the configuration.

attach: Attach existing machines to ClusterWare nodes.

-i, --ids PROVIDERS A comma-separated list of providers to query or modify.

-a, --all Interact with all providers (default for *list*).

EXAMPLES

```
scyld-clusterctl heads --help
```

Show the available subcommands: *list* (*ls*), *clean*, *service*, *delete* (*rm*).

```
scyld-clusterctl heads clean --help
```

Show the resources that can be cleaned: *--all*, *--files*, *--heads*, *--database*, *--dry-run*.

```
scyld-clusterctl heads service
```

Display the names of all ClusterWare system services and their states for a solo head node cluster.

```
scyld-clusterctl heads --all service
```

Display the names of all ClusterWare system services and their states for all head nodes.

```
scyld-clusterctl heads -i head02 service
```

Display the names of all ClusterWare system services and their states for head node *head02*.

```
scyld-clusterctl heads service --help
```

Show all the available actions on services: *--start*, *--stop*, *--restart*, *--enable*, *--disable*.

```
scyld-clusterctl heads --all clean --all
```

Clean everything on all head nodes.

```
scyld-clusterctl pools --help
```

Show the available subcommands: *list* (*ls*), *create* (*mk*), *clone* (*cp*), *update* (*up*), *delete* (*rm*)

```
scyld-clusterctl pools create name=infiniband_nodes pattern=ib{} first_index=0
scyld-nodectl -i n[64-127] update naming_pool=infiniband_nodes
```

Create a node name group "infiniband_nodes" for nodes named "ibX", beginning with "ib0", and associate those names with nodes n64 to n127.

```
scyld-clusterctl nets create first_ip=10.1.1.100 mask_bits=24 ip_count=100 gateway_ip=10.
1.1.10 router_ip=10.1.1.1
```

Create a network that includes 100 IP addresses, 10.1.1.100 to 10.1.1.199, with a network mask of 24 bits. The network mask does not have to align with the *first_ip* field, nodes are numbered starting at the *first_ip*. The network gateway and router are optional, but are explicitly given in this example.

RETURN VALUES

Upon successful completion, **scyld-clusterctl** returns 0. On failure, an error message is printed to *stderr* and **scyld-clusterctl** returns 1.

4.9.4 scyld-nssctl

NAME

scyld-nssctl -- Manage the *scyld-nss* service.

USAGE

```
scyld-nssctl
  [-h] [-v] [start] [stop], [status]
```

DESCRIPTION

A basic tool to start, stop, or show the status of the *scyld-nss* functionality without affecting the systemd status. The 'start' and 'stop' actions must be executed by user *root*.

OPTIONAL ARGUMENTS:

-h, --help	Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
-v	Increase verbosity.

start

Insert *scyld* in the `/etc/nsswitch.conf` *hosts* line to enable (or reenable) *scyld-nss* functionality.

stop

Disable *scyld-nss* functionality by removing *scyld* in the `/etc/nsswitch.conf` *hosts* line.

status

Display the current status of *scyld-nss* functionality. (The default if no argument is supplied.)

EXAMPLES

```
scyld-nssctl
  Display the current state of scyld-nss.
```

```
scyld-nssctl status
  Display the current state of scyld-nss.
```

```
scyld-nssctl stop
  Disable scyld-nss functionality.
```

```
scyld-nssctl start
  Enable scyld-nss functionality.
```

RETURN VALUES

Upon successful completion, **scyld-nssctl** returns 0. On failure, an error message is printed to `stderr` and **scyld-nssctl** returns nonzero.

4.9.5 IP Forwarding Issues

If IP forwarding is desired and is not working, then search for the line containing "net.ipv4.ip_forward":

```
grep net.ipv4.ip_forward /etc/sysctl.conf
grep net.ipv4.ip_forward /etc/sysctl.d/*
```

If that line exists and the assigned value is set to zero, then IP forwarding is disabled.

See *Changing IP Addresses* for details.

4.9.6 managedb

NAME

managedb -- Directly manipulate the database.

USAGE

managedb

```
[-h] [-v] [-q] [[-c | --config] CONFIG] [--print-options] [--as-ini] {join IP,
leave, eject IP, clear, update, recover, maintain, save ARCHIVE, load ARCHIVE, merge
ARCHIVE}
```

DESCRIPTION

This is a low-level tool that directly manipulates the database, generally only executed by other scyld-* tools.

The tool resides in `/opt/scyld/clusterware/bin/managedb` and must be executed by user `root`.

ACTIONS

join *IP*

Join this head node (referenced by IP address) to an existing cluster.

--purge Entirely delete the exiting database(s).

leave

Remove this head node from the cluster.

eject *IP*

Remove the specified head node IP address from the cluster.

clear

Reset the data back to a fresh, empty state.

--reinit Reinitialize the database server.

--purge Entirely delete the exiting database(s).

update

Update the internal database format.

recover

Attempt to recover the local database.

maintain

Perform a maintenance tasks.

--compact Force a compaction, regardless of current size.

--defrag Shrink the database after compacting.

save *ARCHIVE*

Save the database and optionally the various cluster files to an *ARCHIVE* file or directory. Default is to save only the database, i.e., no other files.

--with-boots Include boot files.

--with-images Include root file system images.

--with-isos Include ISO images uploaded for kickstarting.

--with-gits Include git archives.

--with-all Include all files (noted above).

--format *FMT*

Select a file format: *zip* (default), *dir*, *tar*

load *ARCHIVE*

Load the database from an *ARCHIVE* file or directory.

- without-boots** Exclude boot files.
- without-images** Exclude root file system images.
- without-isos** Exclude ISO images uploaded for kickstarting.
- without-gits** Exclude git archives.
- without-all** Exclude all files, i.e., only import the database.

--format *FMT*

Select a file format: *zip* (default), *dir*, *tar*

merge *ARCHIVE*

Merge the contents of an *ARCHIVE* file or directory into the database.

- without-boots** Exclude boot files.
- without-images** Exclude root file system images.
- without-isos** Exclude ISO images uploaded for kickstarting.
- without-all** Exclude all files, i.e., only import the database.

--format *FMT*

Select a file format: *zip* (default), *dir*, *tar*

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config *CONFIG*** Specify a client configuration file *CONFIG*.
- print-options** Print all backend options, then exit.
- as-ini** Use ini format when printing options.

EXAMPLES

(Reminder: `managedb` resides in `/opt/scyld/clusterware/installer/managedb`)

```
sudo managedb leave
```

Detach the current head node from the cluster.

```
sudo managedb eject 10.54.0.2
```

Eject head node at IP address 10.54.0.2 from the cluster.

RETURN VALUES

Upon successful completion, **managedb** returns 0. On failure, an error message is printed to `stderr` and **managedb** returns 1.

4.9.7 ICE ClusterWare Log Files

The `/var/log/clusterware/` folder contains several log files that may help diagnose problems. Additionally, the ICE ClusterWare™ database service may have useful information in its logs.

For `etcd`, see `/var/log/clusterware/etcd.log`.

On a typical head node the `/var/log/clusterware/` folder contains `api_access_log` and `api_error_log` files. These are the Apache logs for the service providing the REST API. The log level available in this file is controlled by the Pyramid logging configuration in the `/opt/scyld/clusterware/conf/pyramid.ini` file. The Pyramid project documentation contains details of the pertinent variables <https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/logging.html>

A selection of log statements from the `api_error_log` are also logged to the ClusterWare database and then copied to the logging folder on each head node. A separate log file is created for each head node and is named based on the head node UID, i.e. `head_293aafd3f635448e9aaa76fc998ebc0c.log`. This should allow a cluster administrator to diagnose many problems without needing to contact every head node individually. The log level for this file is controlled by the `logging.level` variable in each head node's `/opt/scyld/clusterware/conf/base.ini` file. The default log level of `WARNING` should be useful but not overly verbose. The options from most terse to most verbose are `AUDIT`, `ERROR`, `WARNING`, `INFO`, `DEBUG`.

The various `/var/log/clusterware/*` logfiles are periodically rotated, as directed by the `/etc/logrotate.d/clusterware`, `/etc/logrotate.d/clusterware-dnsmasq`, and `/etc/logrotate.d/clusterware-iscdhcp` configuration files that distributed distributed in the `clusterware`, `clusterware-dnsmasq`, and `clusterware-iscdhcp` RPMs, respectively.

Note

If the local cluster administrator modifies the `/etc/logrotate.d/clusterware` file, then a subsequent update of `clusterware` RPM will install a new version as `/etc/logrotate.d/clusterware.rpmnew`. The cluster administrator should merge this `clusterware.rpmnew` into the local customized `/etc/logrotate.d/clusterware`. Similar treatment of `clusterware-dnsmasq` and `clusterware-iscdhcp` is advised.

4.9.8 Creating Diagnostic Test Images

In uncommon situations a cluster administrator may wish to execute a self-contained diagnostic program as a compute node image. "Self-contained" means the diagnostic program itself functions as a kernel and does not need an `initrd`.

An example is the `memtest86+` memory diagnostic, which can be downloaded from www.memtest.org:

```
# Download the latest compressed Linux 64-bit ISO: mt86plus_6.01_64.iso.zip

# Uncompress the downloaded file to expose the ISO:
unzip mt86plus_6.01_64.iso.zip

# Mount the ISO
sudo mkdir -p /mnt/mt86plus_6.01_64
sudo mount mt86plus_6.01_64.iso /mnt/mt86plus_6.01_64 -o loop

# Create a new boot configuration that consists of just the efi binary:
scyld-bootctl create name=Memtest_6.01_Boot kernel=@/mnt/mt86plus_6.01_64/EFI/BOOT/
↳ bootx64.efi

# Configure the desired node (e.g., n123) to execute that new boot configuration:
scyld-nodectl -i n123 set _boot_config=Memtest_6.01_Boot
```

(continues on next page)

(continued from previous page)

```
# Ensure that you have a method to view serial output from that node.
# For example, if serial output for node n123 uses ttyS1:
scyld-bootctl -i Memtest_6.01_Boot update cmdline="console=ttyS1,115200"

# And then reboot that node.
scyld-nodectl -i n123 reboot
```

Note

memtest86+ version 6.01 works for both *legacy* and *uefi* PXE booting.

4.9.9 scyld-sysinfo

NAME

scyld-sysinfo -- Capture the system state information.

USAGE**scyld-sysinfo**

```
[-h] [-V] [--no-tar] [--no-save BLACKLIST] [--up | -i NODES] [-d DIR_SUBSTR] [-m MESSAGE]
```

DESCRIPTION

The tool works best when executed by a cluster administrator who is either user root or a user with sudo rights. The executing user must have write access to the current working directory.

The tool captures elements of the current system state into a subdirectory of the current working directory with the name `sysinfo-$(hostname)-YY-MM-DD` (using a 2-digit Year-Month-Day). This "capture" subdirectory is compressed by default into a gzip'ed tarball; alternatively, the optional `--no-tar` argument skips that compression and allows the administrator to explore the "capture" subdirectory to view exactly what information the tool has captured.

The administrator can employ a *blacklist* file containing a list of files and directories to **not** capture, passing this *blacklist* path to the tool with the `no-save` argument. The administrator can also use `--no-tar` and manually delete captured files and subdirectories within `sysinfo-$(hostname)-YY-MM-DD`, then manually compress the final captured information for archival or for sending the file to others for examination.

The tool also optionally captures sysinfo state for compute nodes, for either all *up* nodes or a specific node or list of nodes.

If the optional `-d DIR_SUBSTR` string is specified, then the directory name contains that alphanumeric string, e.g., `sysinfo-DIR_SUBSTR-$(hostname)-YY-MM-DD.tar.gz`.

If `-m MESSAGE` is specified, then the *MESSAGE* string is retained as the contents of the file *DESCRIPTION* at the top of the output directory. If `-m MESSAGE` is not specified, then the script queries the user for optional multi-line input that is retained as file *DESCRIPTION* in the output directory.

In the rare event that the tool aborts while capturing data, note that a partial capture is still available as the subdirectory `sysinfo-$(hostname)-YY-MM-DD` in the current working directory.

OPTIONAL ARGUMENTS:

- | | |
|-------------------|--|
| -h, --help | Print usage message and exit. Ignore trailing args, parse and ignore preceding args. |
| -V | Print the scyld-sysinfo version and ClusterWare package versions. |

- no-save BLACKLIST** Do not save files/directories listed in file *BLACKLIST*.
- no-tar** Leave the output as a subdirectory, not as a gzip'ed tarball.
- up** Optionally capture the state of all *up* compute nodes.
- i NODES** Optionally capture the state of a specific node or nodes.
- d DIR_SUBSTR** Insert the alphanumeric string *DIR_SUBSTR* into the output directory/tarball name.
- d MESSAGE** If specified, then the *MESSAGE* string is retained as the contents of file *DESCRIPTION* at the top of the output directory.

EXAMPLES

`scyld-sysinfo`

Capture the state of the current node into a gzip'ed tarball, executed as user *root*.

`scyld-sysinfo --no-tar`

Capture the state of the current node into a human-readable subdirectory of the current working directory.

`scyld-sysinfo -I -d UMich`

The output directory name for the head node "headnode1" is "sysinfo-UMich-headnode1-YY-MM-DD".

`scyld-sysinfo -m "dhcpd fails with network error"`

The output directory contains the file *DESCRIPTION* that contains the specified string.

`scyld-sysinfo --up`

Capture the state of the current head node and all the *up* compute nodes.

`scyld-sysinfo -i n0-10`

Capture the state of the current head node and compute nodes *n0* through *n10*.

`scyld-sysinfo -i n0,n2,n100`

Capture the state of the current head node and compute nodes *n0*, *n2*, and *n100*.

RETURN VALUES

Upon successful completion, **scyld-sysinfo** returns 0. On failure, an error message is printed to `stderr` and **scyld-sysinfo** returns nonzero.

4.10 Create Login Nodes

In many cluster environments, especially in traditional HPC environments, end users require access to a login node to interface with the system. The end user can log into that node to submit jobs, check on job status, and examine the results of completed jobs. After a complex computation, the results can be quite large and users may prefer to visualize the results on the cluster rather than downloading the content to visualize it on their local machine. Some clusters provide individual login nodes to specific end users whereas others allocate larger shared systems accessible to all end users, or use a mixture of both approaches.

Login nodes also provide end users with access to their home directory. The home directory is usually provided on a shared file system so that the same files and paths are available on the compute nodes during jobs. On a small cluster that shared file system may be provided by a simple NFS mount, but in more complex environments a more performant and resilient parallel file system (WekaIO, GPFS, etc.) is preferable. Configuring the shared file system is beyond the scope of this document. For NFS sharing, consult the relevant documentation for your chosen operating system.

Access control to login nodes is highly cluster-specific and also not covered in this document. The usual approaches consist of installing an authentication client into the login node image and/or configuring PAM within the image to utilize a site-wide identity provider. In addition to providing a single point of control for who can submit jobs on the cluster, that identity provider is useful to keep UIDs and GIDs consistent across the cluster.

The following files referenced in this example can be found in `/opt/scyld/clusterware-tools/examples`.

- `deploy/virsh.sh`

Files in the `deploy/` subdirectory are example scripts to install packages into images or locally installed nodes. These scripts assume that the head node has access to the internet. You may need to make modifications to the scripts to install from non-standard sources.

- `partitions.butane`

The `partitions.butane` file provides a Butane (<https://coreos.github.io/butane>) configuration to minimally partition a local `/dev/sda` drive for deployment. That script will likely require modifications to match the hardware used for deployment.

- `wipe-disks.sh`

The `wipe-disks.sh` script can be used to entirely remove all partitions from a node; however, extreme caution should be used and the script must be modified before it is run to prevent accidental execution.

The ICE ClusterWare™ platform implements “providers” plugins to help cluster administrators allocate and manage login nodes. The following example uses the simplest provider plugin, `virsh`, to mark a ClusterWare node as a hypervisor, install the appropriate `libvirt` packages, allocate a virtual login node on that hypervisor, and deploy the newly created login image to the virtual machine’s local disk.

1. Create an image that includes the necessary `libvirt` packages.

1. Clone the `DefaultImage` to a new name:

```
scyld-imgctl -iDefaultImage clone name=HypervisorImage
```

 **Tip**

Although not necessary, defining the hypervisor node with a memorable name will make organizing the cluster easier and also provide a place to set several variables necessary for persistent image deployment.

2. Create a boot configuration that uses the new image:

```
scyld-bootctl -iDefaultBoot clone name=HypervisorBoot image=HypervisorImage
```

3. Deploy `virsh` packages into the new image:

```
scyld-modimg -iHypervisorImage --deploy deploy/virsh.sh \
--discard-on-error --upload --overwrite
```

2. Configure the attributes and naming pool for the login node.

1. Create an attribute group and include attributes used for deploying the "HypervisorBoot" as a persistent installation on the local disk:

```
scyld-attribctl create name=hypers
scyld-attribctl -ihypers set _boot_config=HypervisorBoot \
_ignition=partitions.butane _bootloader=grub \
_boot_style=disked _disk_root=LABEL=root
```

2. Create the “hypers” naming pool and define a node using it:

```
scyld-clusterctl pools create name=hypers pattern=hyper{} group=hypers
```

3. Create the *hyper0* hypervisor node:

```
scyld-nodectl create mac=00:28:50:34:0f:ce \  
power_uri=ipmi:///root:password@10.110.10.35 naming_pool=hypers
```

4. Deploy that image to *hyper0* by rebooting *hyper0* and forcing a PXE Boot to start deploying the image:

```
scyld-nodectl -ihyper0 reboot then power setnext pxe then waitfor up
```

This command may take several minutes depending on how long it takes *hyper0* to boot. During this time, the local disk is partitioned, the image is unpacked onto those partitions, grub is installed, and then *hyper0* reboots to the local disk.

5. Once *hyper0* boots, define a provider instance associated with that node and report the resources on that hypervisor.

1. Create a provider pointing at *hyper0*:

```
scyld-clusterctl providers create name=hyper0 type=virsh \  
spec='{“server”: “hyper0”}'
```

2. Report the resources to confirm the connection works:

```
scyld-clusterctl providers -ihyper0 resources
```

6. Create a login node image.

1. Clone the *DefaultImage*:

```
scyld-imgctl -iDefaultImage clone name>LoginImage
```

2. Create a boot configuration that uses the new image:

```
scyld-bootctl -iDefaultBoot clone name>LoginBoot image>LoginImage
```

Tip

Although this example only clones the *DefaultImage*, additional libraries and applications may make sense depending on the cluster hardware and expected workloads. This is also the time to configure the image to work with a site-wide identity provider.

7. Define attributes and naming for the login node(s).

1. Create an attribute group and naming pool for login nodes:

```
scyld-attribctl create name=logins  
scyld-attribctl -ilogins set _boot_config>LoginBoot
```

2. Create the *logins* naming pool:

```
scyld-clusterctl pools create name=logins pattern=login{:02d} group=logins
```

8. Create a virtual machine using the previously defined provider instance.
 1. Allocate a virtual machine (VM) and attach it to the *logins* naming pool:

```
scyld-clusterctl providers -ihyper0 alloc --attach logins \
  --cpus 4 --memory 8G --disk 20G
```

2. Wait for the new login node to boot:

```
scyld-nodectl -ilogin00 waitfor up
```

The created virtual machine can be accessed just like any other ClusterWare node.

4.11 Update and Upgrade

4.11.1 Updating ICE ClusterWare Software

From time to time, updates and add-ons to the ICE ClusterWare™ platform are released. Customers on active support plans can access these updates on the Penguin Computing website. Visit <https://www.penguincomputing.com/computing/support/technical-support/> for details. That website offers answers to common technical questions and provides access to application notes, software updates, and product documentation.

ClusterWare release versions follow the traditional three dot-separated numbers format: *<major>.<minor>.<patch>*. Updating to a newer *major* release should be done with care. *Updating ClusterWare 11 to ClusterWare 12* requires an awareness of specific issues that are discussed later in this section.

The *Release Notes* contains brief notes about the latest release, and the *Changelog* provides a history of significant changes for each software release and a list of *Known Issues And Workarounds*.

4.11.1.1 Updating head nodes

The `scyld-install` tool is used to update ClusterWare software on a head node, just as it was used to perform the initial installation. This tool first determines if a newer *clusterware-installer* package is available, and if so will update *clusterware-installer* and then restart `scyld-install`.

Important

A simple `yum update` will **not** update ClusterWare packages on a head node, as the `scyld-install` tool has disabled `/etc/yum.repos.d/clusterware.repo` in order to prevent `yum update` from inadvertently updating the ClusterWare software. Instead, Penguin Computing strongly recommends using the `scyld-install` tool to perform updates of the basic ClusterWare packages that were originally installed by `scyld-install`. To install or update any optional ClusterWare packages described in *Additional Software*, you must use `sudo yum <install-or-update>--enablerepo=scyld* <packages>`.

Important

`scyld-install` uses the `yum` command to access the ClusterWare software and potentially various other repositories (for example, Red Hat RHEL or Rocky) that by default normally reside on Internet websites. However, if the head node(s) do not have Internet access, then the required repositories must reside on local storage that is accessible by the head node(s). See *Creating Local Repositories without Internet*.

Note

Executing `scyld-install` with no arguments presupposes that the ClusterWare platform is not yet installed. If the ClusterWare platform is currently installed, then the tool asks for positive confirmation that the user does intend to update existing software. You can avoid this interaction by providing the `-u`` or `--update` arg. That same degree of caution occurs if executing `scyld-install --update` on a server that does not currently have the ClusterWare platform already installed: the tool asks for positive confirmation that the user does intend to install the ClusterWare platform as a fresh install.

Important

Updating from 12.0.1 and earlier to 12.1.0 requires reconfiguration of the Influx/Telegraf monitoring stack. The following command can be used to update the necessary config files: `/opt/scyld/clusterware/bin/influx_grafana_setup --tele-env`, followed by `systemctl restart telegraf`. All data will persist through the upgrade.

The `scyld-install` tool only updates basic ClusterWare head node software that was previously installed by the tool, plus any other dependency packages. After the ClusterWare software is updated, you can execute `yum check-update --enablerepo=scyld* | grep scyld` to view the optional ClusterWare packages that were previously installed using `yum install --enablerepo=scyld*`, and then use `sudo yum update --enablerepo=scyld* <PACKAGES>` to update (or **not**) as appropriate for your local head node.

You can also execute `yum check-update` to view the non-ClusterWare installed packages that have available updates, and then use `sudo yum update <PACKAGES>` to selectively update (or **not**) as appropriate for your local head node.

Alternatively, `scyld-install --clear-all` empties the database and clears the current installation. Just like during an initial installation, after a `--clear-all` the database should be primed with a cluster configuration. The cluster configuration can be loaded at the same time as the `--clear-all` using the `--config /path/to/cluster-conf` argument. This will use the `scyld-cluster-conf` tool to load the cluster configuration's initial declaration of private cluster interface, max number of nodes, starting IP address, and MAC address(es), as described in *Execute the ICE ClusterWare Install Script*. See *scyld-cluster-conf* for more details about the `scyld-cluster-conf` tool.

Similar to using `scyld-install` on a non-ClusterWare server to perform a fresh install or to join another head node on an existing cluster, executing `scyld-install --clear-all --config /path/to/cluster-conf` will invoke the `scyld-add-boot-config` script to create a new default boot image.

4.11.1.2 Updating compute nodes

A compute node can be dynamically updated using a simple `yum update`, which will use the local `/etc/yum.repos.d/*repo` file(s). If the compute node is executing a ClusterWare created image, then these changes (and any other changes) can be made persistent across reboots using `scyld-modimg` and performing the `yum install` and `yum update` operations inside the chroot. See *Modifying Images* for details.

4.11.1.3 Updating ClusterWare 11 to ClusterWare 12

ClusterWare version 11 updates cleanly to version 12, albeit retaining the CW11-built boot configurations and images.

Important

A cluster using the ClusterWare Couchbase database must first switch that database to `etcd`.

Important

You must examine `/etc/yum.repos.d/clusterware.repo` and potentially edit that file to reference the ClusterWare version 12 repos. If the `baseurl=` contains the string `clusterware/11/`, then change 11 to 12. If the `gpgkey` contains `RPM-GPG-KEY-PenguinComputing`, then change `PenguinComputing` to `scyld-clusterware`.

CW11-based compute nodes are compatible with CW12 parent head nodes. However, to make use of the full additional functionality of CW12, after updating the CW11 head node(s) you should also update CW11 images to CW12 with at least the newest version of `clusterware-node`. See [Updating compute nodes](#), above.

4.11.2 Updating Firmware

A cluster contains hardware components that employ writeable firmware, such as the BIOS on a server, device controller firmware, and "smart switch" firmware. The management of cluster firmware and the identification of a need to update that firmware is beyond the scope of this document. Contact Penguin Computing Support for guidance.

4.11.3 Updating Base Distribution Software

The decision about if and when to update RHEL-clone base distribution software is complex and needs to be made by a local cluster administrator, ranging from never updating anything on the cluster to updating frequently.

Production clusters in constant use commonly have regular update schedules, typically ranging from weekly to quarterly. The cluster administrator should track the RHEL-clone release notifications as well as the ICE ClusterWare™ release notifications to determine which security fixes, bug fixes, and feature enhancements merit disrupting normal cluster operations in order to perform an update or a group of updates.

Base distribution software updates on a schedule managed by the distributor (e.g., Red Hat). RHEL-clone versioning consists of two dot-separated numbers that define a major release and minor release. See [Supported Distributions and Features](#) for details about what the ClusterWare platform supports on head nodes and compute nodes.

Various package releases can occur for a given *major.minor* release, and those *patch* releases are generally compatible with other software in the same *major.minor* release, which means a node can generally update *patch* release packages as desired. However, a kernel or device driver update may potentially require relinking of 3rd-party software. These *patch* releases typically include security fixes, bug fixes, and backward-compatible feature enhancements. See the distributors' documentation for details.

A minor release update generally entails a larger number of packages, and those packages need to be updated as a group in order to guarantee interoperability. Before updating to a new minor release the cluster administrator should confirm that the 3rd-party software intended to be in use is compatible with that particular major.minor base distribution. A minor release commonly includes a new kernel with substantial changes, and that commonly requires 3rd-party device software to be relinked. See the distributors' documentation for details.

A major release update always entails a large number of packages and usually changes of some degree to user and application interfaces. Commonly there is no simple updating from one major release to another, and an administrator commonly needs to perform a fresh install of the new distribution. Before updating to a new major release the cluster administrator should confirm that the intended 3rd-party software is supported by that major release. See the distributors' documentation for details.

Important

The ClusterWare build version (e.g., e17, e18, or e19, combined with x86_64 or aarch64) must match the base distribution's major release and hardware platform.

4.12 Backup and Restore

4.12.1 Backup and Restore of ICE ClusterWare Software

The `scyld-install` script can also be used to back up and restore all cluster-specific data, including the cluster configuration, images, and node details. To back up the cluster:

```
scyld-install --save /path/to/backup.zip
```

By default the produced ZIP archive can be quite large, as it will contain all boot files and root file system images. If these files are archived by other means, e.g. as part of a backup solution for cluster-wide shared storage, then system administrators may want to include the `--without-files` option. The resulting ZIP file will contain only the ICE ClusterWare™ database. Be aware that this option should only be used if those files are separately archived or when providing a copy of your ClusterWare database to Penguin Computing technical support.

A previously produced archive can also be loaded by the `scyld-install` script:

```
scyld-install --load /path/to/backup.zip
```

Important

Loading a ZIP backup will erase all data and all images and replace them with the corresponding contents from the archive.

During save and load, the `scyld-install` script is actually using the `managedb` tool that provides additional options and capabilities. For details see [managedb](#).

4.12.2 Backup and Restore of the Database

Cluster administrators may wish to capture the database state in the event that a database restore operation is desired in the future. This can be done by manually executing the `/opt/scyld/clusterware/bin/take-snapshot` tool, or more preferably by setting up a cronjob to periodically execute that tool.

For details see [take-snapshot](#).

4.12.2.1 take-snapshot

NAME

take-snapshot -- Perform a database backup

USAGE

```
take-snapshot
```

DESCRIPTION

This is a low-level tool that performs a database backup, typically executed periodically by cron. The tool uses optional "backups" configuration settings found in `/opt/scyld/clusterware/conf/base.ini`.

The tool resides in `/opt/scyld/clusterware/bin/take-snapshot` and must be executed by user `root`.

The base.ini optional settings and examples:

backups.user = CWADMIN

This setting optionally specifies the user name `CWADMIN` of a ClusterWare administrator. If unspecified, then the default is user `root`, although in that case `root` must be previously declared (e.g., via `scyld-adminctl create name=root`) as a ClusterWare administrator.

backups.path = ~CWADMIN/.scyldcw/database-backups

This setting optionally specifies the path to the directory into which the backups and associated files reside. If unspecified, then the default is `~CWADMIN/.scyldcw/database-backups` for the `CWADMIN` user in effect, whether explicitly specified or whether using the default `root`. For example, if `backups.user` is unspecified, then `CWADMIN` defaults to `root` and the default `backups.path` defaults to `~root/.scyldcw/database-backups`. The `take-snapshot` tool creates the directory with owner `CWADMIN`.

Within the `backups` directory there is a subdirectory `files` that contains the various raw `content`, `kernel`, and `initramfs` files from the database, a `database-backups.log` logfile, and one or more `snap-<timestamp>` directories of database snapshots, each created by an execution of the `take-snapshot` tool. Within each of these snapshot directories is a managedb-generated zipfile of files other than the various raw image files in the `files` subdirectory, and symlinks with "pretty" names such as "DefaultBoot.kernel" and "DefaultImage.content" that point to specific raw files in the `files` subdirectory.

backups.retention = 1h/24h,1d/7d,1w/4w,1040w

This setting optionally specifies the four retention tiers, which are comma-separated *block* and *span* time values separated by a '/'. A time value is a nonzero positive integer with a single letter suffix of *h* for hours, *d* for days, or *w* for weeks.

The above values are the default values for the tiers and specify:

```
Tier1: For the most recent 24 hours ("24h"), retain a max of one snapshot per hour (
↪ "1h").
Tier2: Then for the previous 7 days ("7d") prior to that Tier1 24 hour span,
      retain a max of one snapshot per day ("1d").
Tier3: Then for the previous 4 weeks ("4w") prior to that Tier2 7 day span,
      retain a max of one snapshot per week ("1w").
Tier4: Then for the previous 1040 weeks ("1040w", or about 20 years), prior
      to that Tier3 4 week span, retain a max of one snapshot per 4 weeks ("4w").
```

Any snapshots older than the Tier4 "span" are simply discarded.

backups.clean = 14d

This setting optionally specifies a interval between scans of the `snap-<timestamp>` directories to determine which of the raw files in the `files` subdirectory, if any, are no longer referenced by any `snap-<timestamp>`. If unspecified, then the default is once every 14 days.

EXAMPLES

(Note that `take-snapshot` resides in `/opt/scyld/clusterware/bin/`)

```
sudo take-snapshot
```

Manually perform a single database backup.

```
sudo cat /var/spool/cron/root
```

A sample crontab to execute the tool once an hour at five minutes past the hour:

```
SHELL=/bin/bash
PATH=/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
MAILTO=root@localhost

05 * * * * /opt/scyld/clusterware/bin/take-snapshot
```

RETURN VALUES

Upon successful completion, `take-snapshot` returns 0. On failure, an error message is printed to `stderr` and `take-snapshot` returns 1.

4.13 Interacting with Compute Nodes

The primary tool for interacting with ICE ClusterWare™ nodes from the command line is `scyld-nodectl`. This tool is how an administrator would add a node, set or check configuration details of a node, see the basic node hardware, see basic status, cause a node to join or leave attribute groups, reboot or powerdown a node, or execute commands on the node.

In this section we will show a number of examples and discuss what information an administrator can both get and set through the `scyld-nodectl` tool, as well as reference other resources for further details.

Nodes are named by default in the form of `nX`, where `X` is a numeric zero-based index. More complicated clusters may benefit from more flexible naming schemes. See *Node Names and Pools* for details.

4.13.1 `scyld-nodectl`

NAME

`scyld-nodectl` -- Query and modify nodes for the cluster.

USAGE

`scyld-nodectl`

```
[-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
 [--human | --json | --csv | --table] [--pretty | --no-pretty] [--show-uids] [-a |
-i NODES] | --up | --down | --booting] {clear, clone,cp, create,mk, delete,rm, exec,
hardware, join, leave, list,ls, ping, power, reboot, replace,re, scp, script, set, shutdown,
sol, ssh, status, update,up, waitfor}
```

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.
- show-uids** Do not try to make the output more human readable.
- a, --all** Interact with all nodes (default for list).
- i, --ids NODES** A comma-separated list of nodes or an admin-defined group of nodes to act upon.
- up** Interact with all "up" nodes.
- down** Interact with all "down" nodes.
- booting** Interact with all "booting" nodes.
- selector TEXT, -s TEXT** Node selection string.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]** Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

- human** Format the output for readability (default).
- json** Format the output as JSON.

--csv	Format the output as CSV.
--table	Format the output as a table.
--pretty	Indent JSON or XML output, and substitute human readable output for other formats.
--no-pretty	Opposite of --pretty .
--fields FIELDS	Select individual fields in the result or error.

ACTIONS

clear [-a | --all | *NAME* ...]

Delete attribute name(s) and their value(s).

-a, --all Delete all attributes.

clone (cp) [--content *JSON* | *INI_FILE*] [*NAME=VALUE* ...]

Copy node with new *NAME/VALUE* identifier pairs.

--content *JSON* | *INI_FILE*

Overwrite fields in the cloned node.

create (mk) [--content *JSON* | *INI_FILE*] [*NAME=VALUE* ...]

Add a node, commonly by specifying its MAC address (e.g., *mac=MACaddr*, that assigns the next available node number and associated IP address).

--content *JSON* | *INI_FILE*

Load this content into the database as a node.

delete (rm)

Delete node(s).

--release Release the machine back to the provider.

exec [--grouped] [--in-order] [--label] [--stdin *IN*] [--binary] [--stdout *OUT*] [--stderr *ERR*] *CMD*

Execute the *CMD* (double-quotes are optional) on node(s). The `scyld-nodectl exec` command passes its current stdin, stdout, and stderr to the remote command, or uses the `--stdin`, `--stdout`, and/or `--stderr` arguments to override the default(s) with a file.

When run via an `ssh` command (e.g. `ssh cwhead scyld-nodectl --up exec uptime`), that stdin should be provided and closed with `Ctrl-d`, or `ssh` should be passed the `-t` argument to force tty allocation. Otherwise the command will detect stdin is a pipe and wait for end-of-file.

Commands executed on multiple nodes will execute in parallel. The degree of fan-out can be controlled through the `ssh_runner.fanout` configuration variable in `base.ini`. Because these commands execute in parallel, their output may be interleaved or not in node index order. Override this with `grouped` or `--in-order` arguments.

For `sshpass` functionality, see `_remote_pass` in the *Reserved Attributes* section of the ClusterWare documentation.

--grouped Results are locally buffered and printed grouped by node.

--in-order Output is printed in node index order, implies `--grouped`.

--label Force output labeling, even if a single node is selected.

--stdin *IN* Provide `@file` or input string as stdin for the *CMD*.

--binary Treat *CMD* output as binary data.

- stdout OUT** Provide a filename *OUT* for the *CMD* stdout output. Any {} in the filename gets translated to the node name (see **EXAMPLES**).
- stderr ERR** Provide a filename *ERR* for the *CMD* stderr output. Any {} in the filename gets translated to the node name (see **EXAMPLES**). An *ERR* value consisting of the string *STDOUT* will merge stderr into stdout.

hardware

Show the "hardware" information subset of `scylid-nodectl ls -L`.

join GROUP ...

Append *GROUP(S)* to the node group lists.

leave [-a | --all | GROUP ...]

Remove *GROUP(S)* from the node group lists.

- a, --all** Remove node(s) from all groups (other than the global default).

list (ls) [--long | --long-long | --raw]

Show information about nodes.

- l, --long** Show a subset of all optional information for each node.
- L, --long-long** Show all optional information for each node.
- raw** Display the raw JSON content from the database.

ping [COUNT]

ping the specified node(s) with *COUNT* packets (default 1).

power {on | off | cycle | status | setnext BOOTDEV}

Display or control the node power state through the plugin defined by the node's *power_uri*, usually *ipmi*. The options *on*, *off*, *cycle*, and *status* correspond to *ipmitool* actions.

power {on|off|cycle} [--force]

- force** Perform the power control action regardless of *_no_boot*.

The option *setnext* specifies the boot device or method to use for the next node boot. *BOOTDEV* choices are *none*, *pxe*, *disk*, and *bios*.

power setnext BOOTDEV

Boot device from: *none*, *pxe*, *disk*, *bios*

reboot [--soft | --hard] [--kexec] [--force] [--timeout SECS]

Reboot node(s) using either "soft" (using *ssh*) or "hard" (using *ipmi*) or *kexec* methods. If none is specified, then the default behavior is to initially attempt a "soft" reboot; and if after a short delay (default 5 seconds) the node does not appear to begin a reboot, then perform a "hard" power cycle. Ignore the reboot if the node's *_no_boot* is set to true (or *t*, *1*, *yes*, *y*) or if *_busy* is set to true (or *t*, *1*, *yes*, *y*), unless an overriding **--force** argument is supplied.

- soft** Reboot node(s) using *ssh* methods.
- hard** Reboot node(s) using *ipmi* methods.
- kexec** Boot directly into a new kernel without a full reboot which would include Power On Self Test (POST) and hardware initialization. See `man kexec` for details. This is implemented on a compute node using the ClusterWare `reboot-kexec` tool which installs from the *clusterware-node* package.
- bootconfig BOOTCONFIG** Kexec into a specific boot configuration.
- force** Override the node's *_no_reboot* attribute value when set to 1.

--timeout SECS Wait a non-default *SECS* seconds between "soft" and "hard" methods.

scp

Copy files to or from node(s). See *scylld-nodectl exec* in **EXAMPLES**, below.

script *SCRIPT*

Execute the specified ClusterWare *SCRIPT* (distributed in the *clusterware-node* package) on the specified compute node(s). The script name *list* (or *ls*) displays names of the available scripts, which generally execute automatically at boot time to facilitate various node initializations and have limited usefulness for later execution by a cluster administrator. However, the scripts *fetch_hosts* (re-download the list of head nodes) and *update_keys* (update SSH keys) may be useful in rare circumstances for a booted node.

set [--content JSON | *INI_FILE*] [*NAME=VALUE*] ...

Set attribute value(s).

--content JSON | *INI_FILE*

Import the *NAME/VALUE* pairs from the file into the node attributes.

shutdown [--soft | --hard] [--timeout *SECS*] [--force]

Shutdown node(s) using either soft (using ssh) or hard (using ipmi) methods. If neither **--soft** nor **--hard** is specified, then the default behavior is to first attempt a soft shutdown; if after a short delay the node does not appear to begin a shutdown, then perform a hard power off.

--hard Shutdown node(s) using ipmi methods.

--soft Shutdown node(s) using ssh methods.

--timeout SECS Wait *SECS* seconds between "soft" and "hard" methods.

--force Perform the power control action regardless of *_no_boot*.

sol [--enable *ID*] [--steal]

Start a serial-over-lan connection using the local ipmitool.

--enable *ID* If SOL payload is disabled, then attempt to enable for *ID* and retry.

--steal If an SOL session is currently active for that node, then deactivate that session and retry.

ssh [--pubkey *FILE*]

Create an SSH connection to the specified node as the user root. This is done using a local SSH key that is temporarily copied to the compute node through the head node and removed after the command completes. The user can provide their own public key, or one will be generated and stored in *~/ .scylldcw/tempauth.key*.

--pubkey *FILE* Specify a file containing a public key to use for this connection.

status [--long] [--long-long] [--health | --aim | --no-aim] [--refresh] [--raw] [--counts]

Show node status.

--health Show status based on *_health* attribute.

--aim Show status based on *_aim_status* attribute.

--no-aim Opposite of **--aim**.

-l, --long Show a subset of all optional information for each node.

-L, --long-long Show all optional information for each node.

--raw Display the raw JSON content from the database.

- refresh** Show basic node states, refreshing for any state change.
- counts** Include node counts.

update (up) [--content JSON | *INI_FILE*] [*NAME=VALUE*] ...

Modify node *NAME* field(s) with new value(s).

--content JSON | *INI_FILE*

Overwrite this content into the database for a node.

waitfor [*Options*] *COND*

Complete when one or more of the specified nodes meet the condition *COND*, which is either an expression or a '@'-prefixed file name. If no nodes are specified, then defaults to --all.

- failure *COND*** Also complete if the failure condition becomes true.
- timeout *SECS*** Complete after *SECS* seconds if condition(s) never become true.
- name *NAME*** Use the currently defined *COND* state known as *NAME*, or define a new *COND* and remember it as *NAME*.
- load** Just save the state sets into the database.
- delete *NAME*** Delete an existing state set *NAME*.
- show [*NAME*]** Show a list of all state sets, or optionally just the details of one.
 - stream** Stream back ongoing results instead of returning the first result and exiting.
 - skip** Do not use or print the initial node states.
 - one-per** Stream node state changes with one node per line.
 - this-head** Only return state changes handled by the current head.

EXAMPLES

```
scyld-nodectl list
```

List all node names.

```
scyld-nodectl status
```

Shows the basic state of each node.

```
scyld-nodectl status
```

Shows the basic state of node n5.

```
scyld-nodectl -i n5 ls -L
```

Shows full information available for node n5.

```
scyld-nodectl -i %groupx ls -l
```

Shows an expanded information available for each node joined to the admin-defined group *groupx*.

```
scyld-nodectl create mac=00:25:90:0C:D9:3C
```

Add a new node to the end of the current list of nodes.

```
scyld-nodectl create mac=00:25:90:0C:D9:3C index=10
```

Add a new node beyond the end of the current list of nodes as node n10.

```
scyld-nodectl -i n3 update mac=40:25:88:0C:B9:2C
```

Replace the current MAC address for node n5 with a new MAC address.

```
scyld-nodectl -i n20 update power_uri=ipmi:///admin:passwd@10.2.255.37
```

Replace the current power_uri (defaults to "none") to an ipmitool authentication and BMC IP address.

```
scyld-nodectl -in2 ssh
```

Use ssh to open a shell on node n2.

```
scyld-nodectl -i n2 exec ls /var/log
```

Execute `ls /var/log` on node n2, directing stdout and stderr to scyld-nodectl's stdout and stderr, respectively.

```
scyld-nodectl -i n2 exec --stdout /tmp/n2.var.log ls /var/log
```

Execute `ls /var/log` on node n2, directing stdout to the head node file `/tmp/n2.var.log`.

```
scyld-nodectl -i n[2-4] exec --stderr STDOUT --stdout /tmp/{}.var.log ls /var/log
```

Execute `ls /var/log` on nodes n2, n3, and n4, directing both stderr and stdout to the head node files `/tmp/n2.var.log`, `/tmp/n3.var.log`, and `/tmp/n4.var.log`, respectively.

```
scyld-nodectl --up exec --stderr STDOUT --stdout /tmp/{}.var.log ls /var/log
```

Perform the same action as above, although this time for all the "up" nodes.

```
scyld-nodectl -in5 exec --stdout /tmp/n5-log.tar.gz tar -czf- /var/log
```

Execute `tar -czf- /var/log` on node n5, directing the stdout of the packed result into the head node file `/tmp/n5-log.tar.gz`.

```
scyld-nodectl -in5 exec --stdin=@/tmp/n5-log.tar.gz tar -C /root -xzf-
```

Send the local file `/tmp/n5-log.tar.gz` as the stdin to node n5 as it executes `tar -C /root -xzf-` to unpack the stdin contents at `/root`.

```
scyld-nodectl -in3 scp check-health.sh r:/opt/scyld/clusterware-node/bin/check-health.sh
```

Copy the local `check-health.sh` file to node n3 as file `/opt/scyld/clusterware-node/bin/check-health.sh`.

```
scyld-nodectl -in3 scp check-health.sh r:/opt/scyld/clusterware-node/bin/
```

Copy the local `check-health.sh` file to node n3 directory `/opt/scyld/clusterware-node/bin/`. The trailing `/` in the remote path is mandatory to differentiate copying a file vs. copying a directory.

```
scyld-nodectl -in3 scp r:/opt/scyld/clusterware-node/bin/check-health.sh /tmp/
```

Copy the remote n3 `check-health.sh` file to the head node's `/tmp/` directory.

```
scyld-nodectl -in3 scp /opt/scyld/clusterware-tools/examples/ r:/tmp/
```

Copy the directory `/opt/scyld/clusterware-tools/examples` to node n3 directory `/tmp/`. The trailing `/` in the remote path is mandatory to differentiate copying a file vs. copying a directory.

```
scyld-nodectl -i n4 reboot ; scyld-nodectl -i n4 waitfor 's[state] == "up"'
```

Reboot node n4, then wait until the node returns to the "up" state.

```
scyld-nodectl -i n4 reboot ; scyld-nodectl -i n4 waitfor up
```

Reboot node n4, then wait until the node returns to the "up" state. Another supported shorthand is the conditional "down".

```
scyld-nodectl -i n0 waitfor @/opt/scyld/clusterware-tools/examples/node-states.ini
```

For node `n0` establish a `waitfor` state condition described in that specified `examples` file, in which the state condition is named `status`. If no `-i <NODE(s)>` is specified, then defaults to `--all`.

```
scyld-nodectl waitfor --name status
```

For all nodes re-establish a `waitfor` state condition for the previously defined state named `status`. When the condition is true for any node, write the state to `stdout` and `exit`.

```
scyld-nodectl waitfor --name status --stream
```

For all nodes re-establish a `waitfor` state condition for the previously defined state named `status`. When the condition is true for any node, write the state change to `stdout` and continue executing.

```
scyld-nodectl -i n0 reboot then waitfor up then exec uname -r
```

Initiate a reboot of node `n0`, wait for the node to return to an "up" state, and then execute `uname -r` on the node.

```
scyld-nodectl -i n0 ssh
```

Start a `ssh` session on node `n0` as user `root` (by default) or whatever user is specified in the node's `_remote_user` attribute.

RETURN VALUES

Upon successful completion, **scyld-nodectl** returns 0. On failure, an error message is printed to `stderr` and **scyld-nodectl** returns 1.

4.14 Nodes Page

To display the **Nodes** page, click **Nodes > Compute Nodes** in the left navigation panel or, from the **Overview** page, click the *Manage Nodes* link in the **Nodes** panel.

At the top of the Nodes page is a **Node Filtering** subpanel (see [Node Filtering](#)). Below that is a summary of the total node count, the counts of Up vs Down nodes, and the counts of healthy vs unhealthy nodes. These summary lines are clickable links that update the page to show just the nodes applicable to that summary link. For example, clicking "7 unhealthy" selects just the nodes currently reported as "unhealthy". Clicking one of these links also updates the **Selector** field in the **Node Filtering** subpanel.

If you are using a job scheduler, such as Slurm, you can toggle the **Sched** switch to show scheduler status. For details on installing job schedulers, see [Job Schedulers](#). For details on configuring job schedulers to work with the ICE ClusterWare™ platform, see [Monitoring Scheduler Info](#).

Below the **Node Filtering** subpanel and summary links is the **Node Grid/Node List** subpanel showing each node in a potentially filtered display. Each node is color-coded to indicate status and health to quickly highlight any anomalies.

The **Create nodes** text box is available at the bottom of the page (see [Create Compute Node](#)).

4.14.1 Node Filtering

Node Filtering allows you to view, create, and manage subsets of compute nodes. Changing the filtering criteria updates the set of nodes displayed in the **Node Grid** or **Node List**.

A filter can be defined and saved as a Dynamic Group, which is dynamic in the sense that a specific filtering criteria may show different nodes when applied at different times. You can also create a dynamic group from the [Dynamic Groups Page](#) and it will be available to use as a filter on the **Nodes** page.

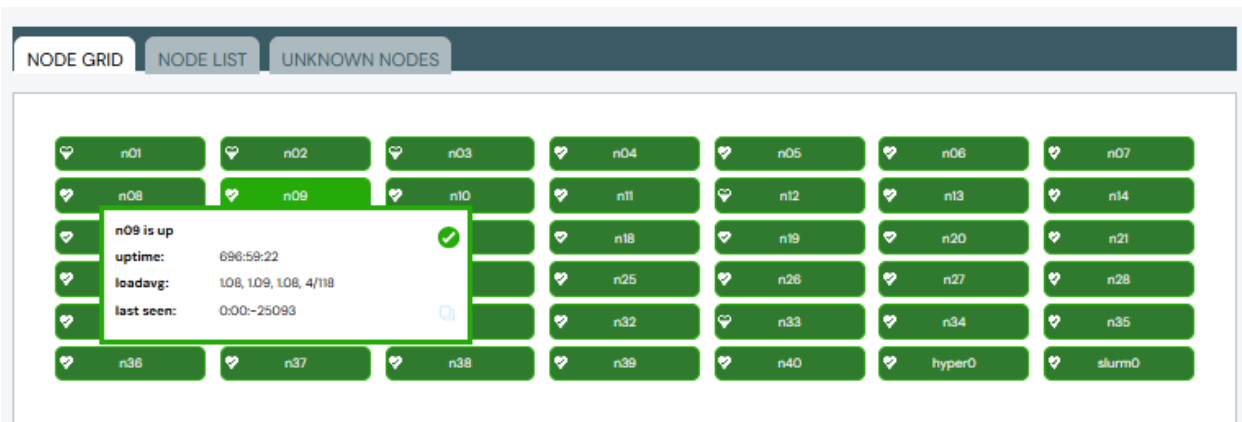
The following actions are available in the Node Filtering section:

- Select a dynamic group - Selecting an existing dynamic group from the **Dynamic Group** menu updates the list of displayed nodes per the **Selector** field.
- Enter a selector expression - Once a valid selector is entered into the **Selector** field, the list of displayed nodes updates.

You are given the option to save this selector expression as a dynamic group by clicking the **Save** button and entering a name and optional description. After creating the new dynamic group, it appears in the **Dynamic Group** menu and on the **Dynamic Groups** page.

- Select **Manage Dynamic Groups** from the **Dynamic Group** menu - Opens the **Dynamic Groups** page.
- Select **Clear** from the **Dynamic Group** menu - Resets node filtering.

4.14.2 Node Grid Display



The **Node Grid** is a row/column grid display where each node is represented by a cell. Above the grid display is a panel containing node status summary information and display options. Each node/cell conveys data with its visual properties:

- Label - This is the node name that appears on the cell. Long names are truncated. The full name is visible in the Node cell.
- Node status - The background color of the cell indicates node status: green=UP, red=DOWN, orange=ALERT, gray=NEW.
- Scheduler status indicator - This indicator is present only if Slurm is configured. Possible values: UNKNOWN, READY/IDLE, BUSY, UNUSABLE/ERROR. If the indicator is ERROR, there may be an accompanying error message. This is displayed when hovering over the Scheduler Status Indicator.

You can interact with cells in two ways:

- Hovering: Opens a pop-up containing additional identification and status information about the node.
- Clicking: Opens a pop-up window containing node details and an option to edit the node.

4.14.3 Node List Display

The screenshot shows the 'Node List' interface. At the top, there are three tabs: 'NODE GRID', 'NODE LIST', and 'UNKNOWN NODES'. Below the tabs is an 'Actions' menu with a dropdown arrow and the text 'Nodes selected: 0'. The main area is a table with the following columns: Name, Status, Health, Boot Config, and Attributes. The table contains 10 rows of node information. Nodes n01, n02, and n03 are marked as 'unhealthy' with a green 'UP' status icon. Nodes n04 through n10 are marked as 'healthy' with a green 'UP' status icon. Each row has a checkbox on the left and an ellipsis (...) on the right. At the bottom of the table, there are page number selectors (1, 2, 3, ..., 5) and a dropdown menu set to '10 per page'.

Name	Status	Health	Boot Config	Attributes
n01	UP	unhealthy	Prod-202502	{ "_boot_config": "Prod-202502", "_health": "Scratch space is not mounted", "location": "Rack1", "sched_state": "allocated" }
n02	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "Scratch space is not mounted", "location": "Rack1", "sched_state": "allocated" }
n03	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "Scratch space is not mounted", "location": "Rack1", "sched_state": "allocated" }
n04	UP	healthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "healthy", "location": "Rack1", "sched_state": "allocated" }
n05	UP	healthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "healthy", "sched_state": "allocated", "location": "Rack1" }
n06	UP	healthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "healthy", "location": "Rack1", "sched_state": "allocated" }
n07	UP	healthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "healthy", "location": "Rack1", "sched_state": "allocated" }
n08	UP	healthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "healthy", "sched_state": "allocated", "location": "Rack1" }
n09	UP	healthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "healthy", "location": "Rack1", "sched_state": "allocated" }
n10	UP	healthy	Prod-202402	{ "_boot_config": "Prod-202402", "location": "Rack1", "sched_state": "allocated", "reserved_for": "John", "hello": "world", "_health": "healthy" }

The **Node List** is a list of compute nodes in table format where each row is a single compute node. Like the **Node Grid** display, the **Node List** indicates node status using color-coding in the **Status** column. The list of nodes can be sorted by Name, Status, or Boot Configuration.

An **Actions** menu is available above the table. Use the actions on one or more nodes by selecting the checkbox for the node row(s) and picking an action from the menu. The action is applied to all selected nodes. For example, you can power on all Down nodes. Actions for individual nodes are available by clicking the ellipsis (...) on the far right of the row.

Tip

The **Nodes selected** indicator to the right of the **Actions** menu shows how many nodes are selected, which can be useful if the list of nodes spans multiple pages.

The example **Node List** display above shows an unfiltered list of nodes. You can adjust the number of nodes shown on each page using the drop-down at the lower right corner of the table. Switch between pages using the page number selectors at the lower left.

The example cluster has 7 "unhealthy" nodes. Filter for those nodes by clicking the "unhealthy" link above the **Node List** table.

Dynamic Group: EDITED Selector: SAVE

Unhealthy nodes

42 nodes total → 42 up / 0 down Health: 35 healthy / 7 unhealthy / 0 checking
 7 nodes chosen → 7 up / 0 down Sched No scheduler data available

NODE GRID NODE LIST UNKNOWN NODES

Actions: Nodes selected: 0

<input type="checkbox"/>	Name	Status	Health	Boot Config	Attributes
<input type="checkbox"/>	n01	UP	unhealthy	Prod-202502	{ "_boot_config": "Prod-202502", "_health": "Scratch space is not mounted", "location": "Rack1", "sched_state": "allocated" }
<input type="checkbox"/>	n02	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "Scratch space is not mounted", "location": "Rack1", "sched_state": "allocated" }
<input type="checkbox"/>	n03	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "_health": "Scratch space is not mounted", "location": "Rack1", "sched_state": "allocated" }
<input type="checkbox"/>	n12	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "location": "Rack2", "sched_state": "idle", "_boot_style": "roram", "reserved_for": "John", "hello..." }
<input type="checkbox"/>	n16	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "location": "Rack2", "sched_state": "allocated", "_boot_style": "roram", "reserved_for": "John", "... }
<input type="checkbox"/>	n17	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "location": "Rack2", "sched_state": "allocated", "_boot_style": "roram", "reserved_for": "John", "... }
<input type="checkbox"/>	n33	UP	unhealthy	Prod-202402	{ "_boot_config": "Prod-202402", "hello": "world", "location": "Rack4", "sched_state": "allocated", "_health": "Scratch space is n..." }

The node list updates to show only the seven unhealthy nodes and the **Selector** field is updated with the filter criteria. The **Attributes** column for each node shows why the health checking script has decided these are unhealthy: "Scratch space is not mounted".

4.14.4 Create Compute Node

To create a new node, enter the node's MAC address in the text box at the bottom of the page.

To create multiple nodes, supply a JSON string or upload a JSON file.

For example, the following string creates five compute nodes:

```
[ {"mac": "FF:11:22:33:44:55"}, {"mac": "11:22:33:a4:fe:66"}, {"mac": "22:33:44:55:66:77"}, {"mac": "00:99:88:77:66:55"}, {"mac": "ef:45:fd:43:23:54"} ]
```

You can also create a node with multiple fields. For example:

```
{ "mac": "11:22:33:44:55:66", "attributes": { "_boot_config": "DefaultBoot" } }
```

4.14.5 Edit Compute Node

To edit a node from the **Node Grid** display:

1. Click the node cell. A pop-up window displays with the current node details.
2. Click the edit pencil next to the node name at the top of the pop-up window.
3. Make updates to the node.
4. Click **Save** to save your changes.

To edit a node from the **Node List** display:

1. Open the node detail window using one of the following options:
 - Click the node name.
 - Click the ellipsis (...) on the far right of the row and select the **Edit** action.
2. Click the edit pencil next to the node name at the top of the pop-up window.
3. Make updates to the node.
4. Click **Save** to save your changes.

4.14.6 Delete Compute Node

You can only delete nodes from the **Node List** display.

To delete a node, click the ellipsis (...) on the far right of the row and select the **Delete** action.

To delete one or more nodes, select the node(s) from the list, then select **Delete** from the **Actions** menu.

4.14.7 Related Links

- [scylld-nodectl](#)
- [Node Creation with Known MAC address\(es\)](#)
- [Command-Line Monitoring of Nodes](#)

4.15 Executing Commands

A cluster administrator can execute commands on one or more compute nodes using the `scylld-nodectl` tool. For example:

```
scylld-nodectl -i n0 exec ls -l /
```

passes the command, e.g. `ls -l /`, to the head node, together with a list of target compute nodes. The head node will then `ssh` to each compute node using the head node's SSH key, execute the command, and return the output to the calling tool that will display the results. Note that this relay through the REST API is done because the ICE ClusterWare™ tools may be installed on a machine that is not a head node and is not able to directly access the compute nodes.

Note that even if DNS resolution of compute node names is not possible on the local machine, `scylld-nodectl exec` will still work because it retrieves the node IP addresses from the ClusterWare database via the head node. Further, once an administrator has appropriate keys on the compute nodes and has DNS resolution of compute node names, they are encouraged to manage nodes either directly using the `ssh` or `pdsh` commands or at a higher level with a tool such as `ansible`.

Commands executed through `scylld-nodectl exec` are executed in parallel across the selected nodes. By default 64 nodes are accessed at a time, but this is adjustable by setting the `ssh_runner.fanout` to a larger or smaller number. This variable can be set in an administrator's `~/scylldcw/settings.ini` or can be set in `/opt/scylld/clusterware/conf/base.ini` on a head node. Setting the `ssh_runner.fanout` variable to a value less than or equal to 1 causes all commands to be executed serially across the nodes.

Some limited support is also provided for sending content to the `stdin` of the remote command. That content can be provided in a file via an option, e.g.:

```
scylld-nodectl -i n0 exec --stdin=@input.txt dd of=/root/output.txt
```

or the content can be provided directly:


```
scyld-nodectl -i n0 exec --stdin='Hello World' dd of=/root/output.txt
```

or the content can be piped to `scyld-nodectl`, and this time optionally using redirection on the compute node to write to the output file:

```
echo 'Hello world' | scyld-nodectl -i n0 exec cat > /root/output.txt
```

When a command is executed on a single node, the command's `stdout` and `stderr` streams will be sent unmodified to the matching file descriptor of the `scyld-nodectl` command. This allows an administrator to include remote commands in a pipe much like `ssh`. For example:

```
echo 'Hello world' | scyld-nodectl -i n0 exec tr 'a-z' 'A-Z' > output.txt
```

will result in a the local file `output.txt` containing the text "HELLO WORLD". The `scyld-nodectl exec` exit code will also be set to the exit code of the underlying command. When a command is executed on multiple nodes, the individual lines of the resulting output will be prefixed with the node names:

```
[admin@virthead]$ scyld-nodectl -in[0-1] exec ls -l
n0: total 4
n0: -rw-r--r--. 1 root root 13 Apr  5 20:39 output.txt
n1: total 0
```

When executing a command on multiple nodes, the exit code of the `scyld-nodectl exec` command will only be 0 if the command exits with a 0 on each node. Otherwise the tool return code will match the non-zero status of the underlying command from one of the failing instances.

The mechanism for passing `stdin` should not be used to transfer large amounts of data to the compute nodes, as the contents will be forwarded to the head node, briefly cached, and copied to all compute nodes. Further, if the data was passed as a stream either through piping to the `scyld-nodectl` command or passing the path to a large file via the `--stdin=@/path/to/file` mechanism, the nodes will be accessed serially, not in parallel, so that the stream can be rewound between executions. This is supported for convenience when passing small payloads, but is not efficient in large clusters. A more direct method such as `scp` or `pdcp` should be used when the content is more than a few megabytes in size. Also note that even when communicating with a single compute node, this is not truly interactive because all of `stdin` must be available and sent to the head node before the remote command is executed.

4.16 Create Nodes

4.16.1 Node Creation with Known MAC address(es)

When a new node's MAC address is known to the cluster administrator, the simplest method is add the node to the cluster is to use `scyld-nodectl create` action and supply that node's MAC address:

```
scyld-nodectl create mac=11:22:33:44:55:66
```

and the node is assigned the next available node index and associated IP address.

The administrator can also add the node at an index other than the next available index, e.g., to add a node `n10`:

```
scyld-nodectl create mac=11:22:33:44:55:66 index=10
```

Of course, if a node already exists for the specified MAC or index, then an error is returned and no node is created.

Adding nodes one at a time would be tedious for a large cluster, so an administrator can also provide JSON formatted content to the `create` action. For example,

```
scyld-nodectl create --content @path/to/file.json
```

where that `file.json` contains an array of JSON objects, each object describing a single node, e.g., for two nodes:

```
[
  { "mac": "11:22:33:44:55:66" },
  { "mac": "00:11:22:33:44:55" }
]
```

The `content` argument can also directly accept JSON, or an INI formatted file, or a specially formatted text file. Details of how to use these alternative formats are available in *ICE ClusterWare Command Line Tools*.

4.16.2 Node Creation with Unknown MAC address(es)

A reset or powercycle of a node triggers a DHCP client request which embeds the node's MAC address. A head node with an interface that is listening on that private cluster network and which recognizes that MAC address will respond with an IP address that is associated with that MAC, unless directed to ignore that node. A ICE ClusterWare™ head node can be so directed to ignore the known-MAC node by using a `_no_boot` attribute (see `_no_boot`).

By default, a ClusterWare DHCP server that does not recognize the incoming MAC will ignore the incoming DHCP client request. To override this default, either enable the **Accept unknown nodes?** switch on the *Cluster Settings* page or run the following command:

```
scyld-clusterctl --set-accept-nodes True
```

Any head node that shares the same database will add that new MAC to the shared ClusterWare database, assign to it the next available node index and associated IP address, and attempt to boot the node.

When all new nodes with previously unknown MAC addresses are merged into the ClusterWare cluster, return to the default functionality by disabling the **Accept unknown nodes?** switch or by running the following command:

```
scyld-clusterctl --set-accept-nodes False
```

If multiple new nodes concurrently initiate their DHCP client requests, then the likely result is a jumbled assignment of indices and IP addresses. Cluster administrators often prefer nodes in a rack to have ordered indices and IP addresses. This ordered assignment can be accomplished by performing subsequent carefully crafted `scyld-nodectl update` actions. For example:

```
scyld-nodectl -i n10 update index=100
scyld-nodectl -i n11 update index=101
scyld-nodectl -i n12 update index=102
scyld-nodectl -i n10,n11,n12 reboot # at a minimum, reboot the updated nodes
```

Note

Desired ordering can more easily be accomplished by performing the initial node resets or powercycling for each individual node in sequence, one at a time, and allowing each node to boot and get added to the database before initiating the next node's DHCP request.

4.16.3 Support for Diskful Nodes

Important

This software is a TECHNOLOGY PREVIEW that is being rolled out with limited features and limited support. Customers are encouraged to contact Penguin with suggestions for improvements, or ways that the tool could be adapted to work better in their environments.

The ICE ClusterWare™ head nodes can be used to host packages that are needed during the installation of the compute nodes. In particular, the `clusterware-node` package must be installed if the node is to be fully integrated with the ClusterWare platform. Other tools can also be staged on the head nodes for use on the compute nodes. For example, to integrate a compute node with the ClusterWare monitoring system, the `telegraf` and `clusterware-telegraf` packages are needed.

Each head node can offer a set of RPM, DEB, and TAR packages through an API end-point. Clients can then download those packages with a tool like `curl`.

RPM files can be automatically converted to TAR files for those systems that may not support other packaging systems. Note that while DEB packages may be provided by the head nodes, they cannot be converted to TAR files.

Note

The examples will use an IP address for the head node since, if the ClusterWare platform is not yet installed, the node may not be able to resolve a head node's hostname.

4.16.3.1 Pre-Installer Script

In cases where the underlying system features are unknown, a "pre-installer" script can be downloaded from a head node. The script runs standard tests to determine the CPU-architecture, the base OS, and the type of packaging system available. The script provides the head node with the collected information and finally downloads and runs a customized installer that is better suited to that system.

To get a copy of the pre-installer for use in a custom script, download it through the API:

```
curl -o cw-preinst http://10.10.1.4/api/v1/install/client/installer
```

If the system configuration is already known, directly download and run a suitable installer script rather than starting with the pre-installer script.

4.16.3.2 Installer Scripts

After the pre-installer completes its tests, it posts the data to the head node and downloads the actual installer script. The pre-installer automatically runs this new script to complete the installation.

While the pre-installer will attempt to find the relevant system information, it may not be able to determine some features and will mark them as "unknown". The head node uses the known features to determine the best installer. Even in the case where all features are properly detected, the head node may not have a fully customized installer for that particular combination of attributes. In those cases, it may reply with a more "generic" script that should work on a wide range of related or similar systems. For example, if the CPU-architecture is unknown but RPM support is detected, then the installer may leverage RPM-based installation of packages since it can assume that the RPM tooling will detect and install suitable packages.

While the installer does enable various Systemd services, it does not start those services. It is often easiest to reboot the system so that Systemd starts everything in the correct order, but you can also start the `cw-status-updater` service manually.

To download a copy of the installer to use as a starting point for a custom script, use the same API endpoint as before, but append the optional information to the URL:

```
curl -o cw-installer http://10.10.1.4/api/v1/install/client/installer?arch=x86_64&
os=rhel&pkg=rpm
```

where:

- `arch` is one of: `x86_64` or `aarch64`
- `os` is one of: `rhel` (includes RedHat, Rocky, Centos), `debian`, `cumulus`, or `sonic` (the latter two are network operating systems)
- `pkg` is one of: `rpm`, `deb`, or `tar`

The TAR installer should work for many Linux systems and can be downloaded with:

```
curl -o cw-installer http://10.10.1.4/api/v1/install/client/installer?pkg=tar
```

4.16.3.3 Installation Logs

For both the pre-installer and installer scripts, logging output is produced as the script executes to enable error triage if necessary. Both log files are written to the `/tmp` directory, named as `CLUSTERWARE_PREINSTALL_LOG` and `CLUSTERWARE_INSTALL_LOG`.

If the pre-install script is skipped (for example, if the TAR-based installer is directly selected through the API), then only the `CLUSTERWARE_INSTALL_LOG` is produced.

4.16.3.4 Head Node Preparation

As of 12.3.0, head nodes contain a "client packages" directory, `/opt/scyld/clusterware/clientpkgs`, with sub-directories for the installers as well as DEB, RPM, and TAR packages. By default, the package directories are empty, but admins can populate them in a variety of ways:

- By copying files from the ClusterWare ISO (the `ScyldPackages` directory)
- By running `dnf` commands to download relevant packages
- By manually downloading RPMs from other sources

Any RPM files should be put in the `rpm` directory and DEB files in the `deb` directory. The `tar` directory can be left empty as the system will auto-convert RPMs to TAR files when requested by a node.

4.16.3.5 RPM and DEB Installations

The RPM and DEB installers install packages with the related tools: `rpm` or `apt`. This ensures that any dependencies are met, pre- or post-installation scripts are run, cross-package trigger scripts are run, etc.

To download the client packages inside a custom script, use the "download" endpoint along with the package type:

```
curl -o mypkg.rpm http://10.10.1.4/api/v1/install/client/download/rpm/mypkg
```

4.16.3.6 TAR Installations

To support a wider range of systems, the ClusterWare platform provides an RPM-to-TAR conversion process for any RPMs in the head node's client-package repository. The TAR installer automatically downloads the related TAR file and unpacks it into `/opt/scyld/<package-name>`. Any files that should be installed in other directories are located in the `./pkg` directory along with any pre- or post-installation scripts. The TAR installer automatically copies relevant files from `./pkg` into `/usr`, `/lib`, and `/etc` as needed. It also automatically runs any post-install scripts.

The RPM-to-TAR conversion extracts any cross-package trigger scripts that were present in the original RPM, but the TAR installer does NOT automatically run those scripts. RPM refers to these as "triggerin" scripts and they often modify the installation process based on the presence of other packages on the system. Admins should check for any such trigger scripts and determine if they are needed for their particular system.

To download the client packages inside a custom script, use the "download" endpoint along with the TAR package type:

```
curl -o mypkg.tar.gz http://10.10.1.4/api/v1/install/client/download/tar/mypkg
```

The head node automatically converts RPM to TAR if it cannot find a pre-existing TAR package. Once converted, the TAR package is cached on the head node for use by other compute nodes.

4.16.4 Compute Node Fields

Use the following commands to view various compute node fields (example commands for node n0):

- View the full list of fields using long-form arguments:

```
scyld-nodectl -i n0 list --long-long
```

- View the full list of fields using shorthand arguments:

```
scyld-nodectl -i n0 ls -L
```

- View the abbreviated list of fields using long-form arguments:

```
scyld-nodectl -i n0 list --long
```

- View the abbreviated list of fields using shorthand arguments:

```
scyld-nodectl -i n0 ls -l
```

The *type* field is currently set to "compute", although future updates to ICE ClusterWare™ may add additional values.

The *groups* and *attributes* fields are described in more detail in [Node Attributes](#) and in the commands *scyld-nodectl* and *scyld-attribctl*.

Prior to a node booting, the system informs the DHCP server of MAC-to-IP address mappings for nodes known to the system. Changes to node indices, IP, or MAC addresses may affect these mappings and cause updates to be sent to the DHCP server within a few seconds. When a node makes a DHCP request, the DHCP server maps that node's MAC address to the correct IP and provides additional options to the booting node, including where to find the correct boot files. These boot files are linked in *boot configurations* stored in the database.

4.16.5 Compute Nodes IPMI Access

ipmi tool is a hardware management utility that supports the Intelligent Platform Management Interface (IPMI) specification v1.5 and v2.0.

IPMI is an open standard that defines the structures and interfaces used for remote monitoring and management of a computer motherboard (baseboard). IPMI defines a micro-controller, called the "baseboard management controller" (BMC), which is accessed locally through the managed computer's bus or through an out-of-band network interface connection (NIC).

The *root* can use *ipmitool* for a variety of tasks, such as:

- Inventory a node's baseboards to determine what sensors are present
- Monitor sensors (fan status, temperature, power supply voltages, etc.)
- Read and display values from the Sensor Data Repository (SDR)

- Read and set the BMC's LAN configuration
- Remotely control chassis power
- Display the contents of the System Event Log (SEL), which records events detected by the BMC as well as events explicitly logged by the operating system
- Print Field Replaceable Unit (FRU) information, such as vendor ID, manufacturer, etc.
- Configure and emulate a serial port to the baseboard using the out-of-band network connection known as serial over LAN (SOL)

Several dozen companies support IPMI, including many leading manufacturers of computer hardware. You can learn more about OpenIPMI from the OpenIPMI project page at <http://openipmi.sourceforge.net>, which includes links to documentation and downloads.

The node's *power_uri* field in the database is optional and informs the head node(s) how to control the power to a given node. A plugin interface allows for different forms of power control, currently supporting IPMI for bare metal nodes, and KVM (virsh) or VirtualBox (vbox) for different types of virtual nodes. For example, a *power_uri* for a VirtualBox virtual node might be:

```
vbox://192.168.56.1/CW_Compute0
```

Production system compute nodes are generally bare-metal nodes that can be controlled via the `ipmitool` command that communicates with the node's Baseboard Management Controller (BMC) interface. Set a *power_uri* with the appropriate BMC IP address and username/password access credentials for these nodes. For example:

```
ipmi:///admin:password@172.45.88.1
```

With *power_uri*, the head node communicates with that compute node's BMC located at 172.45.88.1 using the username "admin" and password "password" to perform a `scylld-nodectl power on`, `power off`, `power cycle`, `shutdown --hard`, or `reboot --hard`.

If for any reason only a specific remote machine can execute `ipmitool` to control a node, then add that server name, and an optional username and password, to the *power_uri*. The local head node will ssh to that remote server and execute the `ipmitool` command from there. For example, the *power_uri*:

```
ipmi://remote_server/admin:password@172.45.88.1
```

sends the `ipmitool` command details to server "remote_server" for execution.

The `scylld-nodectl "soft" shutdown --soft` and `reboot --soft` commands do not use the *power_uri*. Rather, they ssh to the compute node to execute the local `/usr/sbin/shutdown` or `/usr/sbin/reboot` command with appropriate arguments. A simple `scylld-nodectl -i <NODE> reboot` (or `shutdown`) first attempts a "soft" action if the node is "up" and the head node can communicate with the node. If the "soft" action is not possible, or does not complete within a reasonable time, then the `scylld-nodectl` resorts to a "hard" action using the *power_uri* connection.

4.17 Boot Nodes

4.17.1 Compute Node Initialization Scripts

All compute node images should include the `clusterware-node` package. This package includes systemd services used for periodically reporting node status back to the head node as well as initialization scripts run as the node is booting.

At the end of the boot process described in *Boot Configurations*, the `mount_rootfs` script hands control of the machine over to the standard operating system initialization scripts when it switches to the newly mounted root. Shortly after networking is established on the booting node, it contacts the parent head node, the compute node begins periodic pushes of status information to the parent, which stores that information in the ICE ClusterWare™ database. The first

data push includes detected hardware information, while subsequent data only contains the more ephemeral node status information. With each status update the node also retrieves its attribute list and stores this list as an INI file at `/opt/scyld/clusterware-node/etc/attributes.ini`. Code running on the compute node can use the contents of this file to customize the node configuration. A simple `attributes.ini` file:

```
[Node]
UID = c1bf15749d724105bce9e07a3d79cb69

[Attributes]
_boot_config = DefaultBoot
```

The `[Node]` section will include node-specific details, while the `[Attributes]` section contains the node attributes as determined from the node's groups using the process described in *Node Attributes*. The `clusterware-node` package also contains a symlink at `/etc/clusterware` pointing to `/opt/scyld/clusterware-node/etc/`.

Shortly after the first status push, a series of shell scripts are executed on the node to perform ClusterWare-specific node initialization. These scripts are linked in `/opt/scyld/clusterware-node/scripts-enabled` and located in `/opt/scyld/clusterware-node/scripts-available`.

All such scripts should include `/opt/scyld/clusterware-node/functions.sh` for common variables and functions, and should use the `attributes.ini` described previously to determine what actions are necessary. Cluster administrators are invited to enable and disable these scripts in their root file system images as they see fit and to contribute improved or added scripts back to the ClusterWare developers for the continuing improvement of the product.

4.17.2 Booting From Local Storage Cache

Cluster designers sometimes include storage on compute nodes as scratch space or to fulfill the requirements of other cluster technologies such as caching in high speed storage systems. If a cluster administrator is able to partition off some of that space, the ICE ClusterWare™ platform can be configured to take advantage of this local storage. This can free up RAM that would otherwise be used to store the operating system and libraries, and in some circumstances of a very large node count may decrease boot-time network load for nodes which have local storage.

When a node boots using the *disked boot style*, it checks two other attributes: `_disk_cache` and `_disk_root`. Each attribute should be set to a value that can be passed as a device to the `mount` command. This includes explicit partition paths such as `/dev/sda2` or `/dev/nvme0n1p4` as well as `LABEL=X` or `UUID=Y` aliases. Because UUIDs are randomly generated during partitioning or file system creation, they are less suitable for cluster use since every node would require a different value. Similarly, a heterogeneous cluster may have different physical disk configurations requiring a cluster administrator to specify different partition paths for different classes of nodes. For these reasons we encourage cluster administrators to label the target partitions using a tool appropriate to the file system, e.g. `e2label`. Because the `_disk_cache` and `_disk_root` attributes are ignored by other boot styles, setting nodes to the *disked* style can be used as a flag to enable and disable booting from local storage without otherwise altering the node's boot configuration.

Early in the boot process a *disked* node will attempt to mount the partition specified by the `_disk_cache` attribute. If this attribute does not exist or if the partition specified cannot be mounted, an error will be logged and booting will continue without local caching. Shortly after the cache is mounted, the `mount_rootfs` script will attempt to mount the specified `_disk_root` partition. If this partition is not provided or cannot be mounted, an error is logged and booting continues in a *rwram* or *roram* style depending on the type of disk image downloaded. Log messages from this early boot process can be found in `/var/log/messages` on the node, and ClusterWare-specific early boot messages are also captured in the `/opt/scyld/clusterware-node/atboot/cw-dracut.log` file.

If the disk cache is successfully mounted, then prior to downloading any image the compute node will check if the image is already present in the cache. If the image is present, then the `mount_rootfs` script will compare the local file size and checksum to values provided by the head node. If both match, then the image download is skipped and the local copy will be used. Alternatively, if the image is not present in the cache or there is a size or checksum mismatch, then any local copy will be deleted and a fresh copy of the image will be downloaded into the cache partition.

During subsequent boots the booting node will confirm the cached image is valid and use the local copy whenever

possible. Note that if the cache partition is large enough to hold several compressed images, then the local cache can provide a somewhat faster means to switch between images on consecutive boots. If the cache ever fills, thereby causing an image download to fail, then the cache will be cleared and the node will reboot to try again.

Important

Please note that a cache partition must be large enough to hold at least the compressed compute node image plus a few megabytes, though ideally should be sized to hold a handful of compressed images.

If the disk root is successfully mounted, then when the image would usually be unpacked into RAM, the `mount_roofs` script will instead delete the contents of the disk root and unpack the image into the now empty partition. Booting will then continue with that partition as the system root. Note that any changes made to the contents of this partition are intentionally discarded during the next `disked` boot. This is done to prevent cluster administrators from inadvertently creating a heterogeneous cluster with unexpected and unpredictable behavior.

Important

Root partitions must be large enough to hold the uncompressed image in addition to files that may be installed after boot. A rough minimum estimate is to provide 2.5 times the space required by the compressed image. We encourage administrators to err on the side of providing excess space, as storage is usually inexpensive.

In order to reduce the chances of automating destructive mistakes, the ClusterWare platform does not provide tools to automatically partition compute node disks based on node attributes. Cluster administrators can manually partition disks in individual nodes for very small clusters and should research parallel management tools such as `ansible` when managing disk partitions on larger clusters: https://docs.ansible.com/ansible/latest/modules/parted_module.html.

4.17.2.1 Failing To Boot From Local Storage

If a compute node is configured to boot from local storage, and yet after successfully booting it is actually instead using a RAM root filesystem, then the problem may be that the `initramfs` image does not contain a needed kernel module to mount the root filesystem on local storage. Examine `/opt/scyld/clusterware-node/atboot/cw-dracut.log` on the compute node to determine if the mount failed and why. If the problem is a missing kernel module, then add that to the `initramfs`. For example, add the `virtio_blk` module, and rebuild the boot config:

```
scyld-mkramfs --update DefaultBoot --kver 3.10.0-957.27.2.el7.x86_64 --drivers virtio_blk
```

4.17.3 Booting Diskful Compute Nodes

In addition to booting diskless clients, the ICE ClusterWare™ platform can also integrate with "diskful" compute nodes that boot from full installations on local disk drives. See *Using Kickstart* for examples.

Add a locally installed node to the cluster using the same mechanisms as a diskless node. For example, if the following are true:

- The new node's network interface is physically connected to the private cluster network shared by existing head node(s) and compute nodes
- The interface is configured to use a dynamic IP address assigned by DHCP at boot time

then execute the following command on the head node:

```
scyld-nodectl create mac=00:11:22:33:44:55
```

The ClusterWare platform assigns the next available IP address to that MAC address.

Alternatively, if the new node has a static IP address, first ensure that static address is in the defined ClusterWare DHCP range, then specify that static address in your command:

```
scyld-nodect1 create mac=00:11:22:33:44:55 index=100 ip=10.10.42.100
```

4.17.3.1 Installing the clusterware-node Package

For the new node to fully integrate as a ClusterWare compute node, install the *clusterware-node* package and dependencies on the new node as the root user. The node does not use the *initramfs* provided by a ClusterWare boot configuration for diskless nodes. As a result, after installing *clusterware-node*, update the configuration file `/opt/scyld/clusterware-node/etc/node.sh` on the node. The configuration file instructs the node how to communicate with a head node. This file must define the *base_url* of a head node and optionally the *iface* network interface name that assigned the MAC address used in the `scyld-nodect1 create` command that added the node to the database.

For example, for interface *eth0* that connects to head node *head-01*, make the following edits:

```
[root@newnode]$ cat /opt/scyld/clusterware-node/etc/node.sh
# Specify a base_url for any head node
base_url=http://head-01/api/v1

# Specify the network interface used to reach the head node(s)
iface=eth0

# Optionally force (sslverify=yes) or disable (sslverify=no) SSL
# certificate checking. Leaving this option blank allows for HTTPS
# without certificate checking.
#sslverify=
```

On the node, run the following command:

```
/usr/bin/update-node-status --hardware --upload
```

This command sends the initial node hardware information to the head node. You can then reboot the node to fully integrate it into the cluster.

The newly booted compute node is controlled through the customary `scyld-nodect1 reboot`, `shutdown`, and `exec` commands. To support `--hard` mode, configure the node's *power_uri* field to provide appropriate `ipmitool` authentication and an IP address of the node's Baseboard Management Controller (BMC). For example: `ipmi:///admin:password@172.45.88.1`. See *Compute Nodes IPMI Access* and *Database Objects Fields and Attributes* for details.

4.17.3.2 Additional Support for Diskful Nodes

ClusterWare head nodes provide additional support for diskful node installation through a set of pre-installer scripts, installer scripts, and a repository of packages that compute nodes can use to download and install node packages. Additionally, the head node repository can automatically convert some packages to a TAR file that should be installable on a wide variety of platforms.

See *Support for Diskful Nodes* for details, including the preparation steps needed on the head nodes.

4.17.4 scyld-reports

NAME

scyld-reports -- Manage and generate cluster reports.

USAGE

scyld-reports

```
[-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
[--human | --json | --csv | --table] [--pretty | --no-pretty] {setup, usage, unknown}
...
```

DESCRIPTION

This tool manages and generates cluster reports.

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]** Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

- human** Format the output for readability (default).
- json** Format the output as JSON.
- csv** Format the output as CSV.
- table** Format the output as a table.
- pretty** Indent JSON or XML output, and substitute human readable output for other formats.
- no-pretty** Opposite of `--pretty`.

ACTIONS**setup [--accountant [ACCOUNTANT]]**

Specify the accountant URL with credentials. Contact Penguin Computing Support or Professional Services for assistance.

usage [--start START] [--end END] [--users USERS] [--queues QUEUES]

Display cluster usage. This requires a prior *scyld-reports setup* of the accountant connection.

- start START** Specify a starting date. Defaults to first of current month.
- end END** Specify a duration in days or an end date. Defaults to days until the end of the month.
- users USERS** Filter the results using comma-separated list of users.
- queues QUEUES** Filter the results using the comma-separated list of queues.

unknown [--newer-than SECS] [--columns COLS] [--sort COLS] [--lookup]

[--as-creates [POOL]] [--flush]

Display MAC addresses of unknown nodes that have attempted to boot.

- newer-than SECS** Only show MACs with contact within the last *SECS* seconds.

- columns COLS** Display the columns in the desired comma-separated name order. Default is display all columns.
- sort COLS** Sort entries based on the provided column names.
- lookup** Attempt to identify MAC OUIs.
- as-creates [POOL]**
Print a specific `scyld-nodectl create` command for each unknown MAC, optionally associating a new node with a specific *POOL*.
- flush** Flush the current unknown node list.

EXAMPLES

`scyld-reports unknown`

Display the full list of unknown nodes that have attempted to boot.

`scyld-reports unknown --newer-than 1200 --columns mac,seen`

Display only nodes which have made contact in the last 1200 seconds (20 minutes), showing each node's MAC address and last seen timestamp.

`scyld-reports unknown --newer-than 600 --as-creates`

Display only nodes which have made contact in the last 600 seconds (10 minutes), and for each show the `scyld-nodectl create mac=<MACADDR>` command that would add that node as a compute node.

RETURN VALUES

Upon successful completion, **scyld-reports** returns 0. On failure, an error message is printed to `stderr` and **scyld-reports** returns 1.

4.18 Manage Nodes

4.18.1 Changing IP Addresses

To change IP addresses on a cluster, generate a configuration file of the currently state of the nodes with their current IP addresses, edit the file to change one or more IP addresses as desired, re-load the file, and trigger the head node to recompute the new addresses and update the database. For example:

```
scyld-cluster-conf save new_cluster.conf
# manually edit new_cluster.conf to change IP addresses
scyld-cluster-conf load new_cluster.conf
scyld-nodectl -i <NODES_THAT_CHANGE> update ip=
```

The new addresses are not seen by compute nodes until they reboot or perform a dhcp renewal.

4.18.2 Node Name Resolution

The `scyld-install` script installs the `clusterware-dnsmasq` package which provides resolution services for head node names. Similar to the `clusterware-iscdhcp`, this package depends on a standard OS provided service, but runs a private instance of that service, configuring it through the templated configuration file `/opt/scyld/clusterware-dnsmasq/dnsmasq.conf.template`. Within that file, fields like "`<DOMAIN>`" are substituted with appropriate values from the cluster network configuration, and the resulting file is rewritten.

Specifically, the "domain" field (defaulting to `.cluster.local`) is appended to compute node names (`n0`, `n1`, etc.) to produce a fully-qualified domain name. That default value can be overridden in the cluster configuration provided at installation time or loaded via the `scyld-cluster-conf` command. Multiple domains can be defined in that configuration file and are applied to any subsequently defined network segments until a later line sets a new domain value. Note

that when changing this value on an established cluster, the cluster administrator may want to only load the networking portion of the cluster configuration instead of recreating already configured compute nodes:

```
scyld-cluster-conf load --nets-only cluster.conf
sudo systemctl restart clusterware
```

By default, any hosts listed in the `/etc/hosts` file on the head node will also resolve on the compute nodes through `dnsmasq` as will names added through the `scyld-clusterctl hosts` command. The `localise-queries` keyword in the template file is provided because head nodes commonly have multiple addresses on different networks and `dnsmasq` should reply with the IP appropriate to the requestor. Commenting out `localise-queries` will cause `dnsmasq` to reply with all IPs for a queried name. To entirely prevent `dnsmasq` being populated with head node IPs set `leases.register_heads = False` in `/opt/scyld/clusterware/conf/base.ini` and restart the `clusterware` service. These `dnsmasq` behaviors and many others can be changed in the aforementioned configuration template.

An administrator may modify the template file to completely remove the domain or to otherwise modify the `dnsmasq` configuration. Please see the `dnsmasq` project documentation for details of the options that service supports. Similarly, the `dhcpd` configuration template is located at `/opt/scyld/clusterware-iscdhcp/dhcpd.conf.template`, although as that service is much more integral to the proper operation of the ClusterWare platform, changes should be kept to an absolute minimum. Administrators of more complicated clusters may add additional "options" lines or similarly remove the "option domain-name" line depending on their specific network needs. Additional DNS servers can also be provided to compute nodes through the "option domain-name-servers" lines. As with `dnsmasq`, please see the ISC DHCP documentation for supported options.

During compute node boot, `dracut` configures the bootnet interface of the node with the DNS servers and other network settings. These settings may be changed by cluster administrators in startup scripts as long as the head node(s) remain accessible to the compute nodes and vice versa.

During initial installation, the `scyld-install` script attempts to add the local `dnsmasq` instance (listening on the standard DNS port 53) as the first DNS server for the head node. If this is unsuccessful, DNS resolution will still work on compute nodes, although the administrator may need to add local DNS resolution before `ssh` and similar tools can reach the compute nodes. Please consult your Linux distribution documentation for details. Note that DNS is not used for compute node name resolution within the REST API or by the ClusterWare administrative tools; rather, the database is referenced in order to map node ids to IP addresses.

4.18.3 Command-Line Monitoring of Nodes

The ICE ClusterWare™ platform provides two primary methods to monitor cluster performance and health: the command line `scyld-nodectl status` tool and a more extensive graphical user interface (see [Grafana Telemetry Dashboard](#)).

More basic node status can be obtained through the `scyld-nodectl` command. For example, a cluster administrator can view the status of all nodes in the cluster:

```
# Terse status:
[admin@virthead]$ scyld-nodectl status
n[0] up
n[1] down
n[2] new

# Verbose status:
[admin@virthead]$ scyld-nodectl status --long
Nodes
  n0
    ip: 10.10.24.100
    last_modified: 2019-04-16 05:02:26 UTC (0:00:02 ago)
    state: up
```

(continues on next page)

(continued from previous page)

```
uptime: 143729.68

n1
  down_reason: boot timeout
  ip: 10.10.42.102
  last_modified: 2019-04-15 09:03:20 UTC (19:59:08 ago)
  last_uptime: 59.61
  state: down

n2: {}
```

From this sample output we can see that n0 is up and has recently (2 seconds earlier) sent status information back to the head node. This status information is sent by each compute node to its parent head node once every 10 seconds, although this period can be overridden with the `_status_secs` node attribute. The IP address shown here is the IP reported by the compute node and should match the IP provided in the node database object unless the database has been changed and the node has not yet been rebooted.

Compute node n1 is currently down because of a "boot timeout". This means that the node attempted to boot, and the node's initial "up" status message to the head node was not received. This could happen due to a boot failure such as a missing network driver, a networking failure preventing the node from communicating with the head node, or if the `cw-status-updater` service provided by the `clusterware-node` package is not running on the compute node. Other possible values for `down_reason` include "node stopped sending status" or "clean shutdown".

There is no status information about n2 because it was added to the system and has never been booted. Additional node status can be viewed with `scyld-nodectl status -L` (an abbreviation of `--long-long`) that includes the most recent full hostname, kernel command line, loaded modules, loadavg, free RAM, kernel release, and SELinux status. As with other `scyld-*ctl` commands, the output can also be provided as JSON to simplify parsing and scripting.

For large clusters the `--long` (or `-l`) display can be unwieldy, so the status command defaults to a summary. Each row of output corresponds to a different node status and lists the nodes in a format that can then be passed to the `--ids` argument of `scyld-nodectl`. Passing an additional `--refresh` argument will cause the tool to start an ncurses application that will display the summary in the terminal and periodically refresh the display:

```
scyld-nodectl status --refresh
```

This mode can be useful when adding new nodes to the system by booting them one at a time as described in [Node Creation with Unknown MAC address\(es\)](#).

4.18.4 Managing Node Failures

In a large cluster the failure of individual compute nodes should be anticipated and planned for. Since many compute nodes are diskless, recovery should be relatively simple, consisting of rebooting the node once any hardware faults have been addressed. Disked nodes may require additional steps depending on the importance of the data on disk. Refer to your operating system documentation for details.

A compute node failure can unexpectedly terminate a long running computation involving that node. We strongly encourage authors of such programs to use techniques such as application checkpointing to ensure that computations can be resumed with minimal loss.

4.18.4.1 Replacing Failed Nodes

Since nodes are identified by their MAC addresses, replacing a node in the database is relatively simple. If the node (n23 in the following example) was repaired, but the same network interface is still being used, then no changes are necessary. However, if it was the network card that failed and it was replaced, then the node's MAC address can be updated with one command:

```
scyld-nodectl -i n23 update mac=44:22:33:44:55:66
```

If the entire node was replaced, you can use the `_no_boot` attribute to temporarily remove the node from ICE ClusterWare™. You can also update the description of the node with details such as the RMA number or anticipated replacement timeline. When the new node arrives, run the command above to modify the MAC address to match the replacement and then remove the `_no_boot` attribute to rejoin the node to the ClusterWare cluster.

If the entire node was replaced, then you may prefer to clear the node status and any history associated with that node instead of just updating the MAC address. To do this, delete and recreate the failed node:

```
scyld-nodectl -i n23 delete
scyld-nodectl create index=23 mac=44:22:33:44:55:66
```

Note

Deleting a node from the ClusterWare cluster removes all of the configuration and settings for the node.

4.18.5 Soft Power Control Failures

If the `scyld-nodectl reboot` or `shutdown` commands always fall back on hard power control, the shutdown process on the compute node may be taking too long. When this happens the `scyld-nodectl reboot` or `shutdown` commands will pause for several seconds waiting for the soft power change to take place before falling back to direct power control through the `power_uri`. A common cause for this is a network file system that is slow to unmount. The cluster administrator should address the problem delaying shutdown, but if it is unavoidable, then the `reboot` and `shutdown` commands accept options to adjust the timeout (`--timeout <seconds>`), or you can specify to use only the soft reboot (`--soft`) without falling back to direct power control.

4.18.6 Managing Large Clusters

ICE ClusterWare™ head nodes generally scale well out-of-the-box, at least from the perspective of software, since the compute nodes' demands on a head node are primarily during node boot, and thereafter nodes generate regular, modest *Telegraf* networking traffic to the *InfluxDB* server to report node status, and generate sporadic networking traffic to whatever cluster filesystem(s) are employed for shared storage.

Very large clusters may exhibit scaling limitations due to hardware constraints of CPU counts, RAM sizes, and networking response time and throughput. Those limitations are visible to cluster administrators using well known monitoring tools.

4.18.6.1 Improve Scaling of Node Booting

The *clusterware* service is a multi-threaded Python application started by the Apache web server. By default, each head node will spawn up to 16 worker threads to handle incoming requests, but for larger clusters (hundreds of nodes per head node) this number can be adjusted as needed by changing the `thread=16` value in `/opt/scyld/clusterware/conf/httpd_wsgi.conf` and restarting the *clusterware* service.

4.18.7 Hostnames Page

Hostnames are entities that are not created by the ICE ClusterWare™ platform, but that the platform should be aware of. A hostname could be a file server, database server, network gateway, or other system that interacts with one or more ClusterWare nodes.

Use the **Hostnames** page to create a DNS record for the external entity. The page is available via **Network > Hostnames** in the left navigation panel.

Hostnames

Refresh Interval (seconds): 10

Hostname entries allow for the definition of additional DNS records with optional DHCP information.

Name ▲	Type ▲	IP Address ▲	MAC Address ▲
slurmctld_primary	srvrec		

ADD HOSTNAME

4.18.7.1 Create a Hostname

To create a hostname:

1. Click **Add Hostname**.
2. Add details about the hostname.
 - **Name:** Required.
 - **Description:** Optional.
 - **Type:** Select either **A Record** or **SRV Record**. The remaining fields change based on your selection.

For a hostname with a DNS A Record:

- **IP Address:** Required. Supports IPv4 addresses.
- **MAC Address:** Optional.

For a hostname with a DNS SRV Record:

- **Weight:** Required.
- **Domain:** Required.
- **Priority:** Required.
- **Port:** Required.
- **Proto:** Required.
- **Service:** Required.
- **Target:** Required.

3. Click **Add Host** to save your changes.

The new hostname appears in the list at the top of the page.

4.18.7.2 Edit Hostname

To edit a hostname:

1. Click the ellipsis (...) on the far right of the row and select the **Edit** action. The **Edit Hosts** pane populates with the hostname details.
2. Make updates to the hostname.
3. Click **Edit Host** to save your changes.

4.18.7.3 Delete Hostname

To delete a hostname, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.18.7.4 Related Links

- [Manage Non-ICE ClusterWare Entities](#)
- [scyld-clusterctl](#)

4.18.8 Manage Non-ICE ClusterWare Entities

You can configure ICE ClusterWare™ to be aware of entities outside of those created by the ClusterWare platform, such as file servers, database servers, power distribution systems, or network gateways that interact with ClusterWare nodes. For example, a compute node could mount a file system from server fs001, but it needs to know the IP address of fs001. Use [scyld-clusterctl](#) with the `hosts` subcommand to manage these entities.

The `hosts` subcommand can take on a few different forms.

1. Create a DNS A record for a host with an IPv4 address:

```
scyld-clusterctl hosts create name=fs001 ip=10.99.88.77 type=arec
```

2. Create a DNS SRV record for a service running on a specified host and port:

```
scyld-clusterctl hosts create name=slurmctld_backup port=6817 proto=tcp
↪service=slurmctld domain=cluster.local target=backuphostname type=srvrec
↪priority=20
```

In this example, a name is provided for the database record (`slurmctld_backup`) and separately the names of the service (`slurmctld`) and the target (`backuphostname`) are specified separately.

3. Create a DNS A record and a DHCP record for a host with an IPv4 address:

```
scyld-clusterctl hosts create name=fs002 ip=10.99.88.78 type=arec
↪mac=11:22:33:44:55:66
```

In this example, a DNS entry is made for the host (as with #1) and a separate DHCP record is made to match the MAC address to the name and IP address. This allows the fs002 server to boot from the CW DHCP server, picking up the relevant network information that it needs to connect to the network.

The `hosts` subcommand supports both IPv4 and IPv6 addresses. Per the DNS specs, A records are for IPv4 addresses and AAAA records are used for IPv6 addresses. Also per the DNS specs, an SRV record should be fully resolvable by the DNS server itself and should not return names for which it does not have an A/AAAA record for. Practically speaking, this means that the “target” field of an SRV record should either be an IP-address or should be specified by another `scyld-clusterctl hosts` entry for the A/AAAA record.

4.19 Attribute Groups

4.19.1 Database Objects Fields and Attributes

Various ICE ClusterWare™ database objects (nodes, boot configurations, image configurations, administrators, attributes, etc.) each carry with them detailed descriptors called *fields*. Each field consists of a name-value pair and is relevant for its database object type. Fields are predefined by the ClusterWare platform. The cluster administrator uses the *update* action to change a field value.

For instance, a compute node object for each node has fields *mac* with the node's MAC address, *name* with the node's alphanumeric name, and *power_uri* with a value denoting how to communicate via ipmi to that node. For example, the command `scylld-nodedctl -i n0 ls -l` displays all the defined fields' name-value pairs for node n0.

Compute node and Attribute Groups object types have special fields called *attributes*, where an attribute is a collection of one or more attribute name-value pairs. Attribute names that begin with an underscore "_" are called *reserved attributes* or *system attributes*. The cluster administrator uses the *set* action to change an attribute value. See [Reserved Attributes](#) for details.

Additional attributes can be added by a cluster administrator as desired, each with a custom name and value defined by the administrator. Any script on a compute node can access the local file `/etc/clusterware/attributes.ini` and find that node's attributes. On the node there are helper functions in `/opt/scylld/clusterware-node/functions.sh` for reading attributes, specifically the function *attribute_value*.

4.19.2 Node Attributes Page

The **Node Attributes** page displays the list of attribute groups as well as a dropdown menu to change the default attribute group. The page is available via **Nodes > Node Attributes** in the left navigation panel.

{:} Node Attributes Refresh Interval (seconds): 10

Manage groups of attributes that can be assigned to nodes. These attribute groups allow administrators to change the behavior of predefined groups of nodes more efficiently.

{:}Name ▲	Description ▲	Attributes ▲
DefaultAttribs	N/A	_boot_config: Prod-202402, last_modified: 171770... ...
GpuGroup	N/A	hello: world, last_modified: 1717789079.7834268, la... ...

Default attribute group: [DefaultAttribs](#) ▼

[ADD ATTRIBUTE GROUP](#)

4.19.2.1 Create Attribute Group

To create an attribute group:

1. Click **Add Attribute Group**.
2. Add details about the attribute group. **Name** and **Attributes** are required fields. When adding multiple attributes, use a JSON structure with "key1":"val1" pairs. For example:

```
{
  "_boot_config": "GpuBoot",
  "_telegraf_plugins": "gpu-smi.conf"
}
```

3. Click **Add/Edit Attribute Group** to save your changes.

The new attribute group appears in the list at the top of the page.

4.19.2.2 Edit Attribute Group

To edit an attribute group:

1. Click the ellipsis (...) on the far right of the row and select the **Edit** action. The **Add/Edit Attribute Group** pane populates with the attribute group's details.
2. Make updates to the attribute group.
3. Click **Add/Edit Attribute Group** to save your changes.

4.19.2.3 Delete Attribute Group

To delete an attribute group, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.19.2.4 Change Default Attribute Group

The Default Attribute Group is used to provide a basic boot configuration for nodes that do not already have one specified. At system installation time, a default attribute group called *DefaultAttribs* is created and contains a single attribute, *_boot_config*, set to *DefaultBoot*. Use the **Default attribute group** dropdown to select a different default attribute group.

4.19.2.5 Related Links

- [Attribute Groups and Dynamic Groups](#)
- [scyl-d-attribctl](#)

4.19.3 Node Attributes

The names and uses of the fields associated with each database object are fixed, although nodes may be augmented with attribute lists for more flexible management. These attribute lists are stored in the *attributes* field of a node and consist of names (ideally legal Javascript variable names) and textual values. Attribute names prefixed with an underscore such as *_boot_config* or *_boot_style* are reserved for use by the ICE ClusterWare™ platform. These attributes may be referenced or modified by administrator defined scripting, but changing their values will modify the behavior of the ClusterWare platform.

Beyond their internal use, e.g. for controlling boot details, attributes are intended for use by cluster administrators to mark nodes for specific purposes, record important hardware and networking details, record physical rack locations, or whatever else the administrator may find useful. All attributes for a given node are available and periodically updated on the node in file `/opt/scyld/clusterware-node/etc/attributes.ini`. This directory `/opt/scyld/clusterware-node/etc/` is also symlinked to `/etc/clusterware`.

Attributes can also be collected together into *attribute groups* that are stored separately from the node database objects. Administrators can then assign nodes to these groups and thereby change the attributes for a selection of nodes all at once.

Each node has a list of groups to which it belongs, and *the order of this list is important*. Attribute groups appearing later in the list can override attributes provided by groups earlier in the list. For any given node there are two special groups: the global default group and the node-specific group. The global default group, which is defined during the installation process and initially named "DefaultAttribs", is always applied first, and the node-specific group contained in the node database object is always applied last. Any attribute group can be assigned to be the default group through the `scyld-clusterctl` command, e.g.,

```
scyld-clusterctl --set-group GroupNameOrUID
```

An example should clarify how attributes are determined for a node. Immediately after installation the "DefaultAttribs" group contains a single value:

```
[example@head ~]$ scyld-attribctl ls -l
Attribute Groups
  DefaultAttribs
    attributes
      _boot_config: DefaultBoot
```

Note that fields extraneous to this example have been trimmed from this output. Looking at two nodes on this same cluster:

```
[example@head ~]$ scyld-nodectl ls -l
Nodes
  n0
    attributes:
      _boot_config: DefaultBoot
    groups: []

  n1
    attributes:
      _boot_config: DefaultBoot
    groups: []
```

By default no attributes are defined at the node level, although all nodes inherit the *_boot_config* value from the "Default-Attribs" group. If an administrator creates a new boot configuration (possibly by using the `scyld-add-boot-config` script mentioned earlier) and calls it "AlternateBoot", then she could assign a single node to that configuration using the `scyld-nodectl` tool, e.g.,

```
scyld-nodectl -i n0 set _boot_config=AlternateBoot
```

Examining the same nodes after this change would show:

```
[example@head ~]$ scyld-nodectl ls -l
Nodes
  n0
    attributes:
      _boot_config: AlternateBoot
    groups: []

  n1
    attributes:
      _boot_config: DefaultBoot
    groups: []
```

Of course, managing nodes by changing their individual attributes on a per-node basis is cumbersome in larger clusters, so a savvy administrator can create a group and assign nodes to that group:

```
scyld-attribctl create name=AltAttribs
scyld-attribctl -i AltAttribs set _boot_config=ThirdBoot
```

Assigning additional nodes to that group is done by "joining" them to the attribute group:

```
scyld-nodectl -i n[11-20] join AltAttribs
```

After the above changes, node `n0` is assigned to the "AlternateBoot" configuration, `n11` through `n20` would boot using the "ThirdBoot" configuration, and any other nodes in the system will continue to use "DefaultBoot". This approach

allows administrators to efficiently aggregate a set of nodes in anticipation of an action against the entire set, for example when testing new images, or if some nodes need specific configuration differences due to hardware differences such as containing GPU hardware.

For a more technical discussion of setting and clearing attributes as well as nodes joining and leaving groups, see *Attribute Groups and Dynamic Groups*, *scylld-attribctl*, and *scylld-nodectl*.

4.19.4 Dynamic Groups Page

The **Dynamic Groups** page displays the list of dynamic groups. The page is available via **Cluster > Dynamic Groups** in the left navigation panel.

Dynamic Groups are a way to group nodes based on shared or common attributes, status, or hardware information. Dynamic Groups also do not alter a node's attributes, so they are most useful in targeting an action at a group of similar nodes.

Name ▲	Description ▲	Selector ▲
UpNodes	Only show up nodes.	status[state] == "up" ...
Index	index > 10

[ADD DYNAMIC GROUP](#)

A dynamic group provides a way to group nodes based on shared or common attributes, status, or hardware information. They are most useful when targeting an action, such as a reboot, to a group of similar nodes.

4.19.4.1 Create Dynamic Group

To create a dynamic group:

1. Click **Add Dynamic Group**.
2. Add details about the dynamic group. **Name** and **Selector** are required fields. For details about specifying a selector, see *Attribute Groups and Dynamic Groups*.
3. Click **Add Dynamic Group** to save your changes.

The new dynamic group appears in the list at the top of the page.

4.19.4.2 Edit Dynamic Group

To update a dynamic group:

1. Click the dynamic group name to open the edit panel for that group.
2. Make updates to the dynamic group.
3. Click **Edit Dynamic Group** to save your changes.

4.19.4.3 Filter Nodes by Dynamic Group

There are two ways to filter existing nodes by dynamic group:

- On the **Nodes** page, select an existing dynamic group from the **Dynamic Group** drop-down list. The list of nodes updates to show only nodes within that dynamic group.

- On the **Dynamic Groups** page, click the ellipsis (...) on the far right of the row for the dynamic group and select the **Apply to nodes** action. The **Nodes** page appears with the dynamic group applied and the list of nodes filtered to show only nodes within that dynamic group.

4.19.4.4 Delete Dynamic Group

To delete a dynamic group, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.19.4.5 Related Links

- *Attribute Groups and Dynamic Groups*
- *scyld-attribctl*

4.19.5 Attribute Groups and Dynamic Groups

The `scyld-install` script creates a default attribute group called *DefaultAttribs*. That group can be modified or replaced, although all nodes are always joined to the default group. The cluster administrator can create additional attribute groups, e.g.,:

```
scyld-attribctl create name=dept_geophysics
scyld-attribctl create name=dept_atmospherics
scyld-attribctl create name=gpu
```

and then assign or remove one or more groups to specific nodes, e.g.,:

```
scyld-nodectl -i n[0-7] join dept_geophysics
scyld-nodectl -i n[8-11] join dept_atmospherics
scyld-nodectl -i n[0-3,7-9] join gpu
scyld-nodectl -i n7 leave gpu
```

These group assignments can be viewed either by specific nodes:

```
scyld-nodectl -i n0 ls -l
scyld-nodectl -i n[4-7] ls -l
```

or as a table:

```
[admin]$ scyld-nodectl --fields groups --table ls -l
Nodes | groups
-----+-----
n0 | ['dept_geophysics', 'gpu']
n1 | ['dept_geophysics', 'gpu']
n2 | ['dept_geophysics', 'gpu']
n3 | ['dept_geophysics', 'gpu']
n4 | ['dept_geophysics']
n5 | ['dept_geophysics']
n6 | ['dept_geophysics']
n7 | ['dept_geophysics']
n8 | ['dept_atmospherics', 'gpu']
n9 | ['dept_atmospherics', 'gpu']
n10 | ['dept_atmospherics']
n11 | ['dept_atmospherics']
n12 | []
n13 | []
```

(continues on next page)

(continued from previous page)

```
n14 | []
n15 | []
```

Commands that accept group lists can reference nodes by their group name(s) (expressed with a % prefix) instead of their node names, e.g.,:

```
scyld-nodectl -i %dept_atmospherics
scyld-nodectl -i %gpu
scyld-nodectl -i %dept_geophysics status -L
```

Both the Kubernetes `scyld-kube --init` command and the Job Scheduler `scyl{{jobsched}}-scyld.setup init`, `reconfigure`, and `update-nodes` actions accept `--ids %<GROUP>` as well as `--ids <NODES>`. For details, see [Kubernetes](#).

In addition to attribute groups, the ICE ClusterWare™ platform also supports admin-defined *dynamic* groups using a query language that allows for simple compound expressions. These expressions can reference individual attributes, group membership, hardware fields, or status fields. For example, suppose we define attribute groups "dc1" and "dc2":

```
scyld-attribctl create name=dc1 description='Data center located in rear of building 1'
scyld-attribctl create name=dc2 description='Data center in building 2'
```

and then add nodes to appropriate groups:

```
scyld-nodectl -i n[0-31] join dc1
scyld-nodectl -i n[32-63] join dc2
```

and for each node, identify its rack number in an attribute:

```
scyld-nodectl -i n[0-15] set rack=1
scyld-nodectl -i n[16-31] set rack=2
scyld-nodectl -i n[32-47] set rack=1
scyld-nodectl -i n[48-63] set rack=2
```

Note that all attribute values are saved as strings, not integers, so that subsequent selector expressions must enclose these values in double-quotes.

Now you can query a list of nodes in a particular rack of a particular building using a `--selector` (or `-s`) expression, and perform an action on the results of that selection:

```
scyld-nodectl -s 'in dc1 and attributes[rack] == "2"' status
# or use 'a' as the abbreviation of 'attributes'
scyld-nodectl -s 'in dc1 and a[rack] == "2"' set _boot_config=TestBoot

# Show the nodes that have 32 CPUs.
# These hardware _cpu_count values are integers, not strings, and are
# not enclosed in double-quotes.
scyld-nodectl -s 'hardware[cpu_count] == 32' ls
# or use 'h' as the abbreviation of 'hardware'
scyld-nodectl -s 'h[cpu_count] == 32' ls

# Show the nodes that do not have 32 CPUs
scyld-nodectl -s 'h[cpu_count] != 32' ls
```

You can also create a *dynamic group* of a specific selector for later use:

```

scyld-clusterctl dyngroups create name=b1_rack1 selector='in dc1 and a[rack] == "1"'
scyld-clusterctl dyngroups create name=b1_rack2 selector='in dc1 and a[rack] == "2"'

# Show the nodes in building 1, rack 2
scyld-nodectl -i %b1_rack2 ls

# Show only those %b1_rack2 nodes with 32 CPUs
scyld-nodectl -i %b1_rack2 -s 'h[cpu_count] == 32' ls

```

You can list the dynamic groups using `scyld-clusterctl`:

```

# Show the list of dynamic groups
[admin1@headnode1 ~]$ scyld-clusterctl dyngroups ls
Dynamic Groups
  b1_rack1
  b1_rack2

```

And show details of one or more dynamic group. For example:

```

# Show the selector associated with a specific dynamic group
[admin1@headnode1 ~]$ scyld-clusterctl dyngroups -i b1_rack1 ls -l
Dynamic Groups
  b1_rack1
    name: b1_rack1
    selector: in dc1 and a[rack] == "1"

# Or show the selector associated with a specific dynamic group in full detail
[admin1@headnode1 ~]$ scyld-clusterctl dyngroups -i b1_rack1 ls -L
Dynamic Groups
  b1_rack1
    name: b1_rack1
    parsed: ((in "dc1") and (attributes["rack"] == "1"))
    selector: in dc1 and a[rack] == "1"

```

The *parsed* line in the above output can be useful when debugging queries to confirm how ClusterWare parsed the provided query text.

4.19.6 scyld-attribctl

NAME

scyld-attribctl -- Query and modify attribute groups for the cluster.

USAGE

scyld-attribctl

```

[-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
[--human | --json | --csv | --table] [--pretty | --no-pretty] [--show-uids] [[-i |
--ids] -i ATTRIBS | -a | --all] {list,ls, create,mk, clone,cp, update,up, replace,re,
delete,rm, get,set,clear}

```

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.

- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.
- show-uids** Do not try to make the output more human readable.
- a, -all** Interact with all attribute groups (default for list).
- i, --ids ATTRIBS** A comma-separated list of attribute groups to query or modify.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]**
Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

- human** Format the output for readability (default).
- json** Format the output as JSON.
- csv** Format the output as CSV.
- table** Format the output as a table.
- pretty** Indent JSON or XML output, and substitute human readable output for other formats.
- no-pretty** Opposite of --pretty.

ACTIONS ON SPECIFIED ATTRIBUTE GROUP(S)

list (ls)

List information about attribute group(s).

create (mk) name=NAME

Add an attribute group *NAME*.

clone (cp) name=NAME

Copy attribute group to new identifier *NAME*.

update (up)

Modify attribute group fields.

replace (re)

Replace all attribute group fields.

delete (rm)

Delete attribute groups.

get

Get attribute values.

set

Set attribute values.

clear

Clear attribute values.

EXAMPLES

```
scyld-attribctl create name=iScsi
```

Add a new attribute group.

```
scyld-attribctl -i iScsi set _boot_config=RebelBoot _boot_style=iscsi
```


Configure attributes to boot nodes using RebelBoot using iSCSI for root file system access.

RETURN VALUES

Upon successful completion, `scylid-attribctl` returns 0. On failure, an error message is printed to `stderr` and `scylid-attribctl` returns 1.

4.19.7 Reserved Attributes

Within the ICE ClusterWare™ attribute system, administrators are encouraged to store whatever information they find useful for labeling and customizing nodes. For ease of use, attributes names should be valid Javascript variable names, i.e., meaning that they may begin with any uppercase or lowercase letter, followed by letters, digits, or underscores. Names that start with an underscore are used by the ClusterWare software and should be set by administrators to affect the behavior of the system. These will be referred to as *system attributes* throughout this discussion.

Attributes are stored internally as a Javascript dictionary mapping strings to strings, otherwise known as name-value pairs. Administrator-defined attribute values should be strings and relatively small in size. The ClusterWare backend database enforces some document size constraints, and collections of node attributes should be no more than tens to hundreds of kilobytes in size. Individual attributes can be any length as long as the overall attribute group or node object size does not exceed this limits. Generally, if a cluster configuration is approaching these sizes, a cluster administrator pursue moving data from the database into shared storage locations referenced by database entries.

Attributes can be applied directly to nodes, but may also be collected into groups, and then these groups applied to sets of nodes. Attributes passed to nodes through groups are treated no differently than those applied directly to a node. Attribute groups help cluster administrators create more scalable and manageable configurations. See *Node Attributes* for more details.

The remainder of this section is a list of system attributes describing their use and allowed values.

4.19.7.1 `_aim_status`

Default: none

Values: AIM status (qual, prod, fail, service, other) with an optional explanation string

Depends: none

The Assured Infrastructure Module uses this attribute to specify what a node is doing or to provide more detailed status for node failures.

4.19.7.2 `_altmacs`

Default: none

Values: comma-separated list of MAC addresses

Depends: none

Nodes with bonded interfaces may send DHCP requests across different legs of the bond with no obvious pattern. Alternative MAC addresses in this attribute will be added to the DHCP server to receive identical responses.

4.19.7.3 `_ansible_pull`

Default: none

Values: reference to an ansible git repo and a playbook in that repo

Depends: none

See *Using Ansible* for details about format and usage.

4.19.7.4 `_ansible_pull_args`

Default: none

Values: optional arguments for `_ansible_pull`

Depends: using `_ansible_pull`

Specify optional arguments for an `_ansible_pull`. See *Using Ansible* for details about format and usage.

4.19.7.5 `_ansible_pull_now`

Default: none

Values: reference to an ansible git repo and a playbook in that repo

Depends: none

The cluster administrator must `systemctl enable cw-ansible-pull-now` and `systemctl start cw-ansible-pull-now`. See *Using Ansible* for details about format and usage.

4.19.7.6 `_bmc_pass`

Default: none

Values: password to the node BMC

Depends: none

This attribute is not directly used within the ClusterWare software but is meant to be referenced in a *power_uri* such as `ipmi:///admin:<attributes[_bmc_pass]>@10.10.10.10`. This attribute is masked by default so the password will not be printed to the terminal by `scylld-nodectl` without the `--no-pretty` argument.

4.19.7.7 `_bootloader`

Default: none

Values: bootloader to install, currently only "grub"

Depends: `_boot_style=disked`

Setting this attribute while using a *disked* boot style will trigger code in the `initramfs` to install the requested bootloader to the disk containing the partition that contains the `/boot` directory, append necessary entries into `/etc/fstab` based on then-mounted partitions, set the `_boot_style` to *sanboot*, and reboot the system.

This option is commonly coupled with the `_ignition` attribute to provide partitioning and filesystem creation. Using these attributes together allows for deploying images as persistent installations for infrastructure nodes.

4.19.7.8 `_bootnet`

Default: `bootnet`

Values: name for the system boot interface

Depends: none

By default, a system will rename whatever interface is used for network booting to "bootnet" but that name can be changed by setting this attribute. Be careful not to use a name already in use on the target system.

4.19.7.9 `_busy`

Default: undefined (see below)

Values: boolean (case-insensitive 1/0, on/off, y/n, yes/no, t/f, true/false)

Depends: none

This attribute explicitly controls the behavior of the compute node's *cw-status-updater* service which periodically gathers node state information every `_status_secs` seconds (default 10) and reports that information to its parent head node.

The *cw-status-updater* service can function in one of two ways:

- The default manner that gathers frequently changing state (e.g., `uptime` and load average) and occasionally gathers (albeit more expensively) infrequently changing state information (e.g., what hardware is present and which ClusterWare packages are currently installed), or
- A "busy mode" manner that severely reduces the scope of what information is gathered and reported. The service in "busy mode" is minimally invasive to performance of real-time (especially multi-node) applications that are sensitive to interruptions and to "jitter".

If `_busy` is undefined, then "busy mode" can be enabled or disabled by the presence or absence of `/opt/scyld/clusterware-node/etc/busy.flag`, which can be created in a job scheduler prologue and delete in an epilogue, or can be manually created and deleted.

If neither `_busy` and `busy.flag` are employed, then the compute node may itself heuristically determine on its own whether or not to execute in "busy mode".

A compute node in "busy mode" reports that with `scyld-nodectl status -l` showing "busy: True".

A busy compute node requires `--force` to reboot.

4.19.7.10 `_boot_config`

Default: none

Values: boot configuration identifier

Depends: none

The `_boot_config` attribute defines what boot configuration a given node should use. For a detailed discussion of boot configurations and other database objects, please see *Boot Configurations*.

A boot configuration identifier may be a, possibly truncated, UID or a boot configuration name.

4.19.7.11 `_boot_rw_layer`

Default: `overlayfs`

Values: `overlayfs`, `rwtab`

Depends: `_boot_style == roram` or `iscsi`

Use `_boot_rw_layer` to control the type of overlay used to provide read/write access to an otherwise read-only root file system image. The *overlayfs* provides a writable overlay across the entire file system, while the *rwtab* approach only allows write access to the locations defined in `/etc/rwtab` or `/etc/rwtab.d` in the node image.

Note that prior to kernel version 4.9, `overlayfs` does not support SELinux extended attributes and so cannot be used for compute nodes with SELinux in enforcing mode. The *rwtab* option does work with SELinux, but two additional changes need to be made when enabling *rwtab*. First, the cluster administrator must modify the `/etc/sysconfig/readonly-root` file in the node image to ensure `READONLY` is set to "yes":

```
READONLY=yes
```

Second, the kernel cmdline in the appropriate boot configuration must include "ro":

```
cmdline: enforcing=1 ro
```

4.19.7.12 `_boot_style`

Default: `rwram`

Values: `rwram`, `roram`, `iscsi`, `disked`, `next`, `sanboot`, `live`

Depends: none

Root file system images can be supplied to nodes through a variety of mechanisms, and this can be controlled on a per-node basis through the `_boot_style` attribute. In both the `rwram` and `roram` modes, the node will download the entire image into RAM and either unpack it into a tmpfs RAM file system (`rwram`) or apply a writable overlay (`roram`). These boot styles have the advantage of post-boot independence from the head node, meaning that the loss of a head node will not directly impact booted compute nodes.

The `iscsi` option uses less RAM as the boot image is not downloaded into node RAM, but depends on the head node even after the node is fully booted. Due to this dependence a head node crash may cause attached compute nodes to hang and lose work. This approach requires a writable overlay, as the images may be shared between multiple nodes.

With the `disked` option, the node boots with images read from local storage. See *Booting From Local Storage Cache* for details.

Use the `next` option to exit the boot loader and allow the BIOS to try the next device in the BIOS boot order. Since this process depends on support in the BIOS, it may not work on every server model.

The `sanboot` option causes the booting node to boot using the iPXE `sanboot` command and defaults to booting the first hard disk. Please see the `_ipxe_sanboot` attribute for more details.

The `live` option only works for ISO-based configurations, e.g., those used for kickstart. For supported ISOs (e.g., RHEL-based) the node boots into the live installer, and the administrator needs to interact with it via the (likely graphical) system console.

4.19.7.13 `_boot_tmpfs_size`

Default: half of RAM

Values: 1g, 2g, etc.

Depends: `_boot_style == rwram` or `_boot_rw_layer == overlayfs`

During the node boot process, a tmpfs is used to provide a writable area for diskless compute nodes. For the `rwram` boot style this attribute controls the size of the root file system where the image is unpacked. When booting with `overlayfs` on a `roram` or `iscsi` style, this attribute controls the size of the writable overlay.

4.19.7.14 `_coreos_ignition_url`

Default: none

Values: The URL of a RHCOS `*.ign` ignition file.

Depends: none

Both `_coreos_ignition_url` and `_coreos_install_dev` are attributes that must be set to fill in variables in the associated boot config's `cmdline`. See *Using RHCOS*.

4.19.7.15 `_coreos_install_dev`

Default: none

Values: The device on the target node into which the image is installed.

Depends: none

Both `_coreos_ignition_url` and `_coreos_install_dev` are attributes that must be set to fill in variables in the associated boot config's `cmdline`. See *Using RHCOS*.

4.19.7.16 `_disk_cache`

Default: none

Values: local partition name + optional encryption

Depends: none

Specifies a persistent location where the node can store downloaded images. This location should be a local partition with sufficient size to hold a handful of compressed images.

If the specified location exists, then the node will retain there a copy of the downloaded image. During subsequent boots the node will first compare the checksum of a file previously saved with the expected checksum provided by the head node in order to avoid unnecessary downloads.

If the specified partition does not exist, then an error will be logged, although the node will download the image to RAM and still boot. If the partition exists but cannot be mounted, then it will be reformatted.

Optionally Linux Unified Key Setup (LUKS) encryption can also be specified for the partition. Append `:encrypt` to the partition name to encrypt with a random key, or append `:encrypt=KEY` to specify an encryption key.

If no key is specified, encryption is performed using standard LUKS tools with 128 bytes of data from `/dev/urandom` stored in a key file used as the passphrase. This key file is only briefly stored in RAM and deleted shortly before an Ext4 file system is created on the newly encrypted partition.

Alternatively, if the specified key is TPM then the random key will be stored in the booting system's Trusted Platform Module (TPM) and deleted out of RAM shortly before the file system is created. The key can also be bound to specific TPM Platform Configuration Register (PCR) values meaning that the TPM will not later reveal the key unless those PCRs hold the same values. Since these values include hashes of the BIOS code, configuration, kernel, and other boot-time binaries access to the encrypted partition can be restricted to specific boot-time configurations. If the TPM has an owner password set it must be provided in the `_tpm_owner_pass` attribute.

Note

The `cryptsetup-luks` package must be installed in the image being booted.

Specifying a `KEY` is essentially necessary for `_disk_cache` because without that after a subsequent reboot the partition contents will be lost as they were encrypted with an unknown random key.

For example:

```
scyld-nodect1 -i n[0-63] set _disk_cache=/dev/nvme0n1p2:encrypt=Penguin
```

If `_disk_cache` is present but no `_disk_root` is provided, then if a `roram`-compatible image is downloaded, then the node will boot directly from the cached image with a writable overlay.

Important

Any data in the partition specified as a `_disk_cache` may be destroyed at boot time!

Similar to `/etc/fstab`, partitions can be identified by device path, UUID, PARTLABEL, or PARTUUID.

4.19.7.17 `_disk_root`

Default: none

Values: local partition name + optional encryption

Depends: ignored unless `_boot_style == disked`

Specifies a persistent location into which at boot time the node can unpack the root image. This will delete the contents of the partition before unpacking the root image. If the specified partition does not exist, then an error will be logged, although the node will still boot using the image unpacked into RAM.

Similar to `_disk_cache`, append `:encrypt` to the partition name to encrypt with a random key, or `:encrypt=KEY` to specify the encryption key. For `_disk_root` a random key is preferable, as the `_disk_root` contents are intended to be ephemeral across boots.

Important

All data in the partition specified as a `_disk_root` will be destroyed at boot time!

Similar to `/etc/fstab`, partitions can be identified by device path, UUID, PARTLABEL, or PARTUUID.

4.19.7.18 `_disk_wipe`

Default: none

Values: comma-separated list of local partition names + optional encryption

Depends: none

The listed partitions will be reformatted at every boot with an Ext4 file system. Similar to `_disk_cache`, append `:encrypt` to the partition name to enable "encryption at rest", or `:encrypt=KEY` to specify the encryption key. Like `_disk_root` the random key is preferable to ensure `_disk_wipe` partition contents are not retrievable from a physically removed storage device.

4.19.7.19 `_domain`

Default: none

Values: a DNS domain name

Depends: none

Booting compute nodes will use this attribute when constructing their full names unless their `_hostname` attribute already includes their domain.

4.19.7.20 `_gateways`

Default: The default gateway for the node's interfaces

Values: `<ifname>=IPaddress`

Depends: None

Override the interface `ifname`'s current gateway value with an alternative IP address. For example, `_gateways=enpls0f0=10.20.30.40,enpls0f1=10.20.40.40`.

4.19.7.21 `_hardware_plugins`

Default: 300

Values: Comma-separated list of hardware plugin modules

Depends: None

Specifies a list of hardware plugins that are added to the list that might be built into the disk image. If a plugin is listed twice, the second listing will be silently ignored; if a plugin does not exist, it will be silently ignored; if a plugin returns an error or outputs no data, it will be silently ignored.

Hardware information is assumed to be changing less frequently and results may be cached to further reduce the load of the monitoring system.

4.19.7.22 `_hardware_secs`

Default: 300

Values: seconds between checking for status hardware changes

Depends: none

A node sends its hardware state (viewed with `scyld-nodectl list --long` and `list --long-long`) as a component of its larger basic status information. See `_status_secs` below. This hardware component is typically only sent once at boot time. However, the node periodically reevaluates its hardware state every `_status_hardware_secs` seconds, and in the rare event that something has changed since it last communicated its hardware state to its parent head node, then the node includes the updated hardware information in its next periodic basic status message.

Changes to this value are communicated to an up node without needing to reboot the node.

4.19.7.23 `_health`

Default: none

Values: node health status

Depends: none

Cluster administrators can use a health check tool that periodically executes on a compute node (see [_health_check](#)) and relays the result back to the head node as a value of `_health`. The health check tool is expected to return a `_health` result string in one of three forms: an integer value of seconds-since-epoch (generated by `date +%s`), a no-problems-detected value of "healthy", or some other string that provides more details about a problem or problems encountered.

The `scyld-nodectl` tool can display the literal `_health` value doing:

```
scyld-nodectl -i n42 --fields attributes._health ls -l
```

Alternatively,:

```
scyld-nodectl status --health [--refresh]
```

and the ClusterWare GUI display a simple summary of the literal `_health` value. The seconds-since-epoch value is displayed as "checking", the "healthy" value is displayed as "healthy", and any other value is displayed as "unhealthy".

The health check tool can set a custom `_health` value to provide more detailed information about the problem was discovered, e.g.,

```
_health="Sent back to Penguin with RMA #123456"
```

or

```
_health="GPU2 is unhealthy"
```

4.19.7.24 `_health_check`

Default: `/opt/scyld/clusterware-node/bin/check-health-basic.sh`

Values: path to health check tool on node

Depends: none

Cluster administrators use the `_health_check` attribute to specify the path to a script or binary executable that implements the health check for display in the `_health` attribute (see `_health`). The default `/opt/scyld/clusterware-node/bin/check-health-basic.sh` tool is duplicated on a head node as `/opt/scyld/clusterware-tools/examples/check-health-basic.sh` to provide a prototype for the cluster administrator to copy and modify as desired, and then deploy to compute node(s), or to install into an image file or files, and then set `_health_check` for specified nodes to point to the path of this alternative tool.

When a health check tool begins executing on the node, it should initially return a `_health` value of the current seconds since epoch, e.g.,

```
set-node-attrs _health=$(date +%s)
```

The ClusterWare GUI and `scyld-nodectl status --health` both interpret this seconds-since-epoch value as "checking". At completion of the health check, the "healthy", "unhealthy", or more elaborate string result should be sent back to the head node in the using the same `set-node-attrs _health=<value>` mechanism.

4.19.7.25 `_health_plugins`

Default: None

Values: Comma-separated list of health plugin modules

Depends: None

Specifies a list of health plugins that are added to the list that might be built into the disk image. If a plugin is listed twice, the second listing will be silently ignored; if a plugin does not exist, it will be silently ignored; if a plugin returns an error or outputs no data, it will be silently ignored.

Health checks are assumed to be changing much less frequently and results may be cached to further reduce the load of the monitoring system.

4.19.7.26 `_health_secs`

Default: 300

Values: Number of seconds between health-check runs

Depends: None

Specifies the time between successive health update cycles. During each cycle, every health plugin will be run.

4.19.7.27 `_health_check_secs`

default: 300

Depends: none

Values: seconds between executing the health check program

Depends: none

Specifies the interval in seconds for executing a health check program specified by the `_health_check` attribute. See [_health_check](#).

4.19.7.28 `_hostname`

Default: none

Values: Hostname or fully-qualified domain name

Depends: none

Booting compute nodes will assign the value of `_hostname` as their hostname using the `hostnamectl` command. If the attribute value is a simple name (without periods), then the cluster domain will be appended to construct a FQDN. Changes to this variable take effect on the next status update.

4.19.7.29 `_hosts`

Default: blank

Values: download

Depends: none

During the compute node boot process, a list of known hosts is downloaded from the head node and is appended to the compute node's `/etc/hosts`. By default this will only append a list of head nodes to ensure that each compute node can resolve all head nodes without DNS. If the `_hosts` attribute is set to 'fetch', then all compute node names and IP addresses will be appended to `/etc/hosts`.

4.19.7.30 `_ignition`

Default: none

Values: A filename in the `kickstarts/` folder

Depends: none

Compute node disks can be partitioned early in the boot process using the included `ignition` tool. Setting the `_ignition` attribute instructs the booting node to download the `ignition` binary from the head node and then use it to download the configuration named by the `_ignition` attribute. At different points in the boot process, `ignition` will execute its `fetch`, `kargs`, `disks`, `mount`, and `files` stages. The `ignition` configuration file is not meant to be human readable or editable so administrators are expected to write YAML files in the format that the `butane` tool can translate. If the provided configuration file ends in `.butane` it will be converted automatically by the ClusterWare backend at the time of download.

The `ignition` tool provides several other capabilities including the ability to mount the created partitions. Using the `_disk_root` and `_ignition` attributes together an administrator can configure a node using the `disked` boot style with directories such as `/var`, `/usr`, etc. on different partitions as required by government STIGs.

Additional documentation about `ignition` can be found at: <https://coreos.github.io/ignition/>

4.19.7.31 `_ips`

Default: none

Values: comma-separated IP assignments

Depends: none

Compute nodes commonly define additional high-speed network interfaces other than the PXE boot network. These interfaces are commonly defined by `ifcfg-XXX` files located in `/etc/sysconfig/network-scripts` and differ between nodes only in the assigned IP address. Use the `_ips` attribute to specify what IP address should be assigned to an individual node on one or more interfaces. For example, a value of `_ips=eno0=10.10.23.12,ib0=192.168.24.12` would cause

the `prenet/write_ifcfg.sh` startup script to replace any `IPADDR=` line in `/etc/sysconfig/network-scripts/ifcfg-ib0` with `IPADDR=192.168.24.12` and would similarly modify the adjacent `ifcfg-en0` file, replacing any IP assignment in that file with `IPADDR=10.10.23.12`.

You can also specify a `bmc=` value as a special accepted interface. The BMC interface is not otherwise listed on a system, but is a recognized value for the `_ips` attribute. For details, see *Providing DHCP to Additional Interfaces*.

4.19.7.32 `_ipxe_sanboot`

Default: none

Values: local disk or partition

Depends: `_boot_style == sanboot`

Use this attribute to cause a node to boot using the iPXE `sanboot` command. This is most commonly used to boot a locally installed disk, although administrators are cautioned to be extremely careful with stateful compute nodes as they will retain modifications from previous boots, leading to an unexpectedly heterogeneous cluster.

Nodes with this attribute set will not download an image from the head node and will instead boot based on the URL or other iPXE `sanboot` arguments provided. Please see the iPXE documentation for the details of what iPXE provides: <http://ipxe.org/cmd/sanboot>

In addition to the arguments and URLs supported by iPXE, the ClusterWare platform also accepts a shorter URL for booting local disks of the form `local://0xHH` where 'HH' is a hexadecimal value specifying a local hard disk. The first disk is identified as 0x80, the second is 0x81, and so on. The provided hexadecimal value is then used in a `sanboot --no-describe --drive 0xHH` call.

4.19.7.33 `_macs`

Default: The default MAC address for each of the node's interfaces

Values: `<ifname>=<MACaddress>`

Depends: None

Override the interface `ifname`'s current MAC address with an alternative value. For example, `_macs=bond0=aa:bb:cc:dd:ee:ff`. Generally only used for bonded interfaces. Ignored for the booting interface `bootnet`.

You can also specify a `bmc=` value as a special accepted interface. The BMC interface is not otherwise listed on a system, but is a recognized value for the `_macs` attribute. For details, see *Providing DHCP to Additional Interfaces*.

4.19.7.34 `_no_boot`

Default: false

Values: boolean equivalents (0 / 1, true / false, t / f, yes / no, y / n)

Depends: none

The `_no_boot` attribute controls whether information about a node is provided to the DHCP server. Any node with `_no_boot` set to true will not receive DHCP offers from any ClusterWare head node. This allows an administrator to temporarily remove a node from the cluster.

4.19.7.35 `_preferred_head`

Default: none

Values: head node UID

Depends: none

In a multihead configuration any head node can provide boot files to any compute node in the system. In most cases this is a desirable feature because the failure of any given head node will not cause any specific set of compute nodes to fail to boot. In some cases the cluster administrator may want to specify a preference of which head node should handle a given compute node. By setting a compute node's `_preferred_head` attribute to a specific head node's UID, all head nodes will know to point that node toward the preferred head node. This is implemented during the boot process when the iPXE script is generated and passed to the compute node. This means that any head node can still supply DHCP, the iPXE binaries, and the iPXE boot script, but the subsequent kernel, initramfs, and root file system files will be provided by the preferred head node, and thereafter the node's boot status information will be sent to that `_preferred_head`.

4.19.7.36 `_remote_pass`

Default: none

Values: node account password for `_remote_user` attribute

Depends: none

Supports an alternative to the customary ClusterWare ssh-key functionality. It is useful to support `scyld-nodectl exec` to non-ClusterWare compute nodes which do not have `clusterware-node` installed, but which do accept user/password authentication.

To use, install the `sshpass` RPM on the head node. Set the `_remote_pass` attribute to the password of the `_remote_user` attribute user name (default `root`). Subsequent executions of `scyld-nodectl exec` to nodes that are set up with this attribute will employ this user/password pair to authenticate access to those target node(s).

Note

Use of `sshpass` is discouraged and is not a best practice. A clear text password is a significant security risk.

4.19.7.37 `_remote_user`

Default: root

Values: node account name

Depends: none

The `_remote_user` attribute controls what account is used on the compute node when executing the `scyld-nodectl reboot/shutdown` commands. Please ensure the specified account can execute `sudo shutdown` without a password or soft power control will not work. Similarly the `scyld-nodectl exec` and `scyld-nodectl ssh` commands will also use the specified remote user account and the boot-time script that downloads head node keys will store those keys in the `_remote_user`'s `authorized_keys` file.

4.19.7.38 `_sched_extra`

Default: None

Values: short line of text with a bit more information on the state of the scheduler

Depends: `sched_watcher` service must be running on the cluster

Gives one line of information on the current state of the node with respect to the scheduler. E.g. if a node is down, it may include whether the node could be pinged, or whether the scheduler-daemon on the node was found.

Note: `_sched_extra` is a "technology preview" and may change or be replaced in the future.

4.19.7.39 `_sched_full`

Default: None

Values: JSON table of information

Depends: `sched_watcher` service must be running on the cluster

Specifies all the information known about the current state of the node with respect to the scheduler. E.g. it may report the number of CPUs or memory seen by the scheduler; or any additional resources (like GPUs) that were found on the system.

Note: `_sched_full` is a "technology preview" and may change or be replaced in the future.

4.19.7.40 `_sched_state`

Default: None

Values: one of `unknown`, `down`, `idle`, or `allocated`

Depends: `sched_watcher` service must be running on the cluster

Specifies the current state of the node with respect to the scheduler, e.g. Slurm.

Note: `_sched_state` is a "technology preview" and may change or be replaced in the future.

4.19.7.41 `_status_cpuset`

Default: all available CPUs

Values: list of one or more CPU numbers

Depends: none

When a compute node boots, the `status-updater` and related child processes can execute by default on any of the node's CPUs, as chosen by the kernel's scheduler. The administrator may instead choose to restrict which CPUs these processes use to be a subset of all CPUs, or even to just a single CPU, in order to minimize the impact that these processes may have on a time-critical application(s) executing on the other CPUs.

The `_status_cpuset` value is a list of CPUs to use. For example, set `_status_cpuset=0` restricts the processes to just CPU 0, set `_status_cpuset="0-1"` restricts to CPUs 0 and 1, and set `_status_cpuset="0-1,4"` restricts to CPUs 0, 1, and 4. See `man 7 cpuset` for details.

4.19.7.42 `_status_hardware_secs`

Default: 300

Values: seconds between checking for status hardware changes

Depends: none

A node sends its *hardware* state (viewed with `scyld-nodectl list --long` and `list --long-long`) as a component of its larger basic status information. See `_status_secs` above. This *hardware* component is typically only sent once at boot time. However, the node periodically reevaluates its hardware state every `_status_hardware_secs` seconds, and in the rare event that something has changed since it last communicated its *hardware* state to its parent head node, then the node includes the updated *hardware* information in its next periodic basic status message.

Changes to this value are communicated to an *up* node without needing to reboot the node.

4.19.7.43 `_status_packages_secs`

Default: 0

Values: seconds between checking for installed packages changes

Depends: none

The time interval in seconds between the relatively expensive search for what ClusterWare packages are installed. This value times 10 is the time interval between the even more expensive calculations of a sha256sum hash of the sorted list of names of all installed packages, distilled into a single hexadecimal value. These values are seen by executing `scyld-nodectl -i<nodes> status -L` on the head node.

A non-zero value should be longer than the `_status_secs` value, described below.

If the value is zero, then these packages searches and calculations are done just at node boot time, and additionally when (and if) the administrator executes `/usr/bin/update-node-status --hardware` on a compute node. Such run-time changes to a node's installed packages are relatively rare, so the default value is zero to minimize the performance impact of these operations.

Changes to this value are communicated to an *up* node without needing to reboot the node.

4.19.7.44 `_status_plugins`

Default: None

Values: Comma-separated list of status plugin modules

Depends: None

Specifies a list of status plugins that are added to the list that might be built into the disk image. If a plugin is listed twice, the second listing will be silently ignored; if a plugin does not exist, it will be silently ignored; if a plugin returns an error or outputs no data, it will be silently ignored.

4.19.7.45 `_status_secs`

Default: 10

Values: seconds between status updates

Depends: none

Booted compute nodes periodically send basic status information to their parent head node. This value controls how often these messages are sent. Although the messages are relatively small, clusters with more compute nodes per head node will want to set this to a longer period to reduce load on the compute nodes.

Changes to this value are communicated to an *up* node without needing to reboot the node.

4.19.7.46 `_telegraf_omit_pattern`

Default: (`_sched_full|_telegraf_omit_pattern`)

Values: regex matching pattern (awk rules)

Depends: None

Specifies a pattern to match for the Telegraf `cw-attrs` plugin. Any compute node attributes or fields that match the pattern will be omitted from the Telegraf data-stream (all remaining fields and attributes will be included).

4.19.7.47 `_telegraf_plugins`

Default: None

Values: Comma-separated list of Telegraf plugin modules

Depends: None

Specifies a list of Telegraf plugins that are added to the list that might be built into the disk image. If a plugin is listed twice, the second listing will be silently ignored; if a plugin does not exist, it will be silently ignored.

The Telegraf/Grafana system is used for whole system monitoring and trending, and is not directly integrated with other Clusterware tools (`scylld-nodectl` will not report on Telegraf data).

Changing `_telegraf_plugins` will cause a restart of Telegraf on the node.

4.19.7.48 `_tpm_owner_pass`

Default: none

Values: Owner password for the compute node TPM

Depends: none

Certain TPM commands require authentication using the "owner" TPM password. This means that the clear-text password must be provided to systems using the TPM for disk encryption via this attribute.

4.19.7.49 `_wipe_all`

Default: none

Values: Three character string "now"

Depends: none

Set this attribute to "now" to trigger a complete wipe of all local disks. This will be done early in the initramfs before network disks are mounted. This attribute is most useful to ensure that all remnants of previous deployments are removed from the local drives before ignition is executed. **ALL DATA ON LOCAL DRIVES WILL BE LOST.**

4.20 Naming Pools Page

Naming pools are used to support multiple name groupings within the ICE ClusterWare™ platform. Naming pools are useful in complex clusters where you want to identify compute nodes based on core capabilities. For details, see [Node Names and Pools](#).

Use the **Naming Pools** page to create and manage naming pools. The page is available via **Nodes > Naming Pools** in the left navigation panel.

Naming Pools Refresh Interval (seconds): 10

Naming Pools are used to assign consecutive hostnames to groups of nodes.

Pool ▲	Pattern ▲	First Index ▲
hypers	hyper{}	0
slurm	slurm{}	0

[ADD NAMING POOL](#)

4.20.1 Create a Naming Pool

To create a naming pool:

1. Click **Add Naming Pool**.
2. Add details about the naming pool.
 - **Name:** Required.
 - **Description:** Optional.
 - **Pattern:** Optional.
 - **First Index:** Optional.
 - **Parent Group:** Optional. Select a parent group from the list of available groups.
 - **IP Base:** Available if a parent group is selected.
 - **Network:** Available if no parent group is selected.
 - **Group:** Available if no parent group is selected.
 - **Offset:** Available if no parent group is selected.
3. Click **Add Naming Pool** to save your changes.

The new naming pool appears in the list at the top of the page.

4.20.2 Edit Naming Pool

To edit a naming pool:

1. Click the naming pool name to open the details panel.
2. Make updates to the naming pool.
3. Click **Edit Naming Pool** to save your changes.

4.20.3 Delete Naming Pool

To delete a naming pool, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.20.4 Change Default Naming Pattern

If no other naming pool is associated with a node, the default naming pattern is used. At system installation time, the default naming pattern is `n{}`. Use **Default naming pattern** to update the cluster settings with a different default naming pattern.

4.20.5 Related Links

- *Node Names and Pools*
- *scylld-clusterctl*

4.21 Node Names and Pools

By default all compute nodes are named `nX`, where `X` is a numeric zero-based node index. This pattern can be changed using "nodename" lines found in a cluster configuration file. For example, a line `nodename compute{}` early in such a file will change the default node naming to `computeX`. This changes both the default node hostnames as well as the names recognized by the `scylld-nodectl` command.

For homogeneous clusters where all compute nodes are essentially the same, this is usually adequate, although in more complex environments there is utility in quickly identifying core compute node capabilities reflected by customized hostnames. For example, high memory nodes and general purpose GPU compute nodes could be named "hmX" and "gpgpuX". These names can be assigned via the `_hostname` attribute as described in *Reserved Attributes*, although the `scyld-nodectl` command will still refer to them as "nX".

To support multiple name groupings within the `scyld-*ctl` tools, the ICE ClusterWare™ system includes the concept of a *naming pool*. These pools are defined and modified through the `scyld-clusterctl pools` command line interface. Once the appropriate pools are in place, then compute nodes can be added to those pools. For example:

```
scyld-clusterctl pools create name=high_mem pattern=hm{} first_index=1
scyld-clusterctl pools create name=general_gpu pattern=gpgpu{} first_index=1
scyld-nodectl -in[37-40] update naming_pool=high_mem
scyld-nodectl -in[41,42] update naming_pool=general_gpu
```

After these changes the `scyld-nodectl status` and `scyld-nodectl ls` output includes the specified nodes as "hm[1-4]" and "gpgpu[1-2]". Any commands that previously used "nX" names now accept "hmX" or "gpgpuX" names to refer to those renamed nodes. The `first_index=` field of the naming pool forces node numbering to begin with a specific value, defaulting to 0. Any nodes not explicitly attached to a naming pool use the general cluster naming pattern controlled through the `scyld-clusterctl --set-naming PATTERN` command. This can be considered the default naming pool.

4.21.1 Node Indexing and Grouping in Naming Pools

Important

When moving multiple compute nodes from one naming pool to another, the node order may not be preserved. Instead, moving them individually, or specifying their MAC addresses in a cluster configuration file, may be more predictable.

When moving a node from one naming pool to another via the `scyld-nodectl` command, the node index is reset to the next available index in the destination pool. Using an explicit `index=X` argument allows the cluster administrator to directly control the node renumbering. Note that nodes in different naming pools may have the same index, so in this configuration the index is no longer a unique identifier for individual nodes. Further, the `--up`, `--down`, `--all` node selectors are *not* restricted to a single naming pool and will affect nodes in all pools that match the selection constraint. Nodes in `scyld-nodectl` output are ordered by index within their naming pool, although the order of the naming pools themselves is not guaranteed. For example:

```
[admin@head clusterware]$ scyld-nodectl ls
Nodes
  n1
  n2
  n3
  n4
  n5
 login6
 login7
 login8
 login9
```

Similarly, the nodes are grouped by naming pool in `scyld-cluster-conf save` output with "nodename" lines and explicit node indices inserted as needed:


```
[admin@head clusterware]$ scyld-cluster-conf save -
# Exported Scyld ClusterWare Configuration file
#
# This file contains the cluster configuration.
# Details of the syntax and semantics are covered in the
# Scyld ClusterWare Administrators Guide.
#
nodename n{}

# 10.10.24.0/24 network
domain cluster.local
1 10.10.24.101/24 10.10.24.115
node 1 00:00:00:00:00:01 # n1
node 00:00:00:00:00:02 # n2
node 00:00:00:00:00:03 # n3
node 00:00:00:00:00:04 # n4
node 00:00:00:00:00:05 # n5
nodename login{}
node 6 00:00:00:00:00:06 # login6
node 00:00:00:00:00:07 # login7
node 00:00:00:00:00:08 # login8
node 00:00:00:00:00:09 # login9
```

The organization of node naming pools is intentionally independent of node networking considerations. The cluster administrator can combine these concepts by creating separate naming pools for each network segment, although this is not necessary.

Secondary DNS names can also be defined using "nodename":

```
nodename <pattern> <ip> [pool_name]
```

A "nodename" line containing an IP address (or IP offset such as "0.0.1.0") can define a name change at an offset within the IP space or define a secondary DNS name depending on whether the IP is within a defined network. For example:

```
iprange 10.10.124.100/24 10.10.124.250
node
node 08:00:27:F0:44:35 # n1 @ 10.10.124.101

nodename hello{}/5 10.10.124.105
node 08:00:27:A2:3F:C9 # hello5 @ 10.10.124.105

nodename world{}/10 10.10.124.155
node 12 08:00:27:E5:19:E5 # world12 @ 10.10.124.157

nodename n%N-ipmi 10.2.255.37 ipmi
# world12 maps to n2-ipmi @ 10.2.255.39

nodename world%N-ipmi/10 10.2.254.37 ipmi
# world12 maps to world12-ipmi @ 10.2.254.39
```

Note that the "<pattern>/X" syntax defines the lowest node index allowed within the naming pool.

4.21.2 Secondary Naming Pools

The ClusterWare platform provides a mechanism to include name resolution for non-management interfaces with predictable IP addresses on existing pools of nodes. You can create secondary naming pools via a configuration file or via the ClusterWare command line tools.

4.21.2.1 Configuration File

The following configuration file uses the default *n{}* node naming convention, but also includes a secondary naming pool labelled *ib0*.

```
iprange 10.54.50.0/24
node 52:54:00:c6:c3:0a
node 52:54:00:c4:f7:1e
node 52:54:00:46:fe:99
node 52:54:00:dc:f2:e5
node 52:54:00:de:b3:5e
node 52:54:00:17:e7:be
node 52:54:00:48:05:f4
node 52:54:00:88:0f:82

nodename n{-ib0} 0.+1.0.0 ib0
```

When this configuration is loaded, the system resolves the following through both `dnsmasq` and `scylid-nss`:

- Names *n0* through *n7*
- Names *n0-ib0* through *n7-ib0* where those *-ib0* addresses are offset into the 10.55.50.0/24 network based on the provided specification (0.+1.0.0)

For example, *n0* resolves to 10.54.50.0 and *n0-ib0* resolves to 10.55.50.0.

4.21.2.2 Command Line Tools

Secondary naming pools can also be created via the following command:

```
scylid-clusterctl pools create name=ib1 parent= pattern=n{-ib1} ip_base=10.55.50.0
```

Where:

- The `parent` argument should generally be the name of a primary naming pool. However, the default cluster naming is not recorded as a primary naming pool. Instead, it is stored and retrieved using `scylid-clusterctl --set-naming <PATTERN> / --get-naming`. Specifying an empty `parent` value in this command refers to that default cluster naming.
- The `pattern` argument is formatted the same as any other ClusterWare naming pattern.
- The `ip_base` argument denotes the first IP in the pool and must be explicitly defined rather than using an offset as is supported in configuration files.

After creating the secondary naming pool, the system resolves offset addresses for the secondary interface corresponding to any nodes in the referenced parent pool.

4.22 Boot Configurations Page

The **Boot Configurations** page is available by clicking **Provisioning + SW > Boot Configs** in the left navigation panel. The page shows a summary list of each boot configuration, its associated kernel release, and image name, the command line to send to each booting kernel, and optionally the repo name and kickstart name.

Boot Configurations

 Refresh Interval (seconds):

Deployable boot configurations link together the kernel, initramfs, and kernel command line to control how nodes boot. They can also reference kickstart files to automate installation to node-local disks.

Name ▲	Release ▲	Image or Repo ▲	Command Line ▲
CentOS-7-x86_64-Everything-2009	3.10.0-1160.el7.x86_64	CentOS-7-x86_64-Everything-2009	enforcing=0 ...
DefaultBoot	4.18.0-513.5.1.el8_9.x86_64	DefaultImage	enforcing=0 ...
Prod-202402	4.18.0-513.5.1.el8_9.x86_64	Prod-202402	enforcing=0 ...
Prod-202502	4.18.0-553.36.1.el8_10.x86_64	Prod-202502	enforcing=0 ...
Rocky_iso	4.18.0-513.5.1.el8_9.x86_64	N/A	N/A ...

ADD BOOT CONFIGURATION

4.22.1 Create Boot Configuration

To create a boot configuration:

1. Click **Add Boot Configuration**.
2. Add details about the boot configuration.
 - **Name:** Required.
 - **Description:** Optional.
 - **Image:** Optional. Name of an existing image.
 - **cmdline:** Optional. Specify the command provided to the kernel as it is booting. The value can be any string, though `key=value` pairs are typical. For example, provide a command to modify power management settings or serial console settings.
 - **Kickstart:** Optional. Provide a Kickstart file to configure how the node is initialized at boot time. See *Using Kickstart* for details.
 - **Repo:** Optional. Select an existing repository from the list. If this value is set, the **Kernel** and **Initramfs** files are pulled from that repo.
 - **Frozen:** Optional. When enabled, the boot configuration cannot be modified via the GUI or command line. See *Freezing a Boot Configuration* for details.
 - **Kernel:** Optional. Typically used with **Initramfs** and not when **Repo** is set. The file can be added after creating the boot configuration.
 - **Initramfs:** Optional. Typically used with **Kernel** and not when **Repo** is set. The file can be added after creating the boot configuration.
3. Click **Add Boot Configuration** to save your changes.

The new boot configuration appears in the list at the top of the page.

4.22.2 Edit Boot Configuration

To edit a boot configuration:

1. Click the ellipsis (...) on the far right of the row and select the **Edit** action. The **Edit Boot Configuration** pane populates with the boot configuration's details.

2. Make updates to the boot configuration.
3. Click **Edit Boot Configuration** to save your changes.

4.22.3 Delete Boot Configuration

To delete a boot configuration, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.22.4 Related Links

- *Create Boot Configuration with Kickstart*
- *scylld-add-boot-config*

4.23 scylld-add-boot-config

NAME

scylld-add-boot-config -- Tool for creating ClusterWare boot configurations.

USAGE

scylld-add-boot-config

```
[-h] [-v] [-q] [-c | --config CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
[--make-defaults] [--distro NAME] [--image NAME] [--iso PATH] [--boot-config NAME]
[--attrib-group NAME] [--batch] [--interactive]
```

OPTIONAL ARGUMENTS

- | | |
|----------------------------|--|
| -h, --help | Print usage message and exit. Ignore trailing args, parse and ignore preceding args. |
| -v, --verbose | Increase verbosity. |
| -q, --quiet | Decrease verbosity. |
| -c, --config CONFIG | Specify a client configuration file <i>CONFIG</i> . |
| --make-defaults | If there are no attribute groups on this system, then automatically build an attribute group referencing a new boot configuration referencing a new image. |
| --distro NAME | Select the pre-existing distro <i>NAME</i> to use when creating an image. |
| --iso PATH | Create a repo and distro from the local or remote base distribution ISO, where <i>PATH</i> is a pathname or a URL. |
| --image NAME | Select the pre-existing image <i>NAME</i> this command should use. |
| --boot-config NAME | Name the boot configuration as <i>NAME</i> . |
| --attrib-group NAME | Name the new attribute group as <i>NAME</i> . |
| --batch | Run this command using the default options. |
| --interactive | Run this command interactively. |

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- | | |
|---------------------------------|---|
| --base-url URL | Specify the base URL of the ClusterWare REST API. |
| -u, --user USER[:PASSWD] | Masquerade as user <i>USER</i> with optional colon-separated password <i>PASSWD</i> . |

EXAMPLES

This tool is used internally by the `scyld-install` tool to populate the initial (or cleared) database with the objects necessary to boot compute nodes. When run on a database with no attribute groups defined and passed the `--auto-first` argument this script will not ask the user any questions and will use default values. This should not be necessary for an administrator to run unless they have manually cleared the database using the `managedb clear` command:

```
scyld-add-boot-config --make-defaults
```

Rebuild the `DefaultImage` and `DefaultBoot`.

```
scyld-add-boot-config --iso Rocky-9.5-x86_64-dvd.iso
```

Use the named ISO file to build a distro and repo named `Rocky-9.5-x86_64-dvd`, and manually accept defaults that create a boot image and boot configuration, all named `Rocky-9.5-x86_64-dvd`.

```
scyld-add-boot-config --iso Rocky-9.5-x86_64-dvd.iso \
  --image Rocky-9.5-Image --boot-config Rocky-9.5-boot --batch
```

Use the named ISO file in hands-off batch mode to build a repo and distro, both named `Rocky-9.5-x86_64-dvd`, a boot image named `Rocky-9.5-Image` and a boot config named `Rocky-9.5-boot`.

RETURN VALUES

Upon successful completion, `scyld-add-boot-config` returns 0. On failure, an error message is printed to `stderr` and `scyld-add-boot-config` returns 1.

4.24 scyld-* Wrapper Scripts

When creating a new boot image, it's common to create a corresponding boot configuration and assign that configuration to a set of nodes. The `scyld-add-boot-config` tool wraps `scyld-modimg`, `scyld-mkramfs`, and the appropriate `scyld-*ctl` tools to perform the necessary steps. The tool also optionally displays the required steps so you can learn about the usage of the underlying tools.

When executed with no arguments, the `scyld-add-boot-config` script asks a series of questions to define the various fields of the boot configuration, image, and attribute group that are being created. Default values are provided where possible.

Important

The default kernel command line sets SELinux on the compute nodes to permissive mode.

4.25 Boot Configurations

The `scyld-install` script creates a basic boot configuration named *DefaultBoot* that references the initial *DefaultImage* and is initially associated with all compute nodes. After installation, you can customize that configuration and/or create additional boot configurations and compute node images.

Modify configuration fields using the `scyld-bootctl` tool. For example, you can change the name and description of the newly created boot configuration on a freshly installed system using the `update` argument:

```
scyld-bootctl -i DefaultBoot update name="NewName" description="New description"
```

The kernel and `initramfs` are also set using the same command, although their paths must be prefixed with `@` (which signifies that what follows is a local file path). For example:

```
scyld-bootctl -i DefaultBoot update kernel=@/boot/vmlinuz-3.10.0-862.el7.x86_64
```

Other database objects (nodes, images, etc.) are modified using similarly named tools (`scyld-nodectl` and `scyld-imgctl`). Each node is associated with a specific boot configuration through its `_boot_config` attribute. Like other attributes, this field may be inherited from an attribute group (including the global default attribute group) or set directly on the node. Learn more about manipulating node attributes in *Interacting with Compute Nodes*.

Boot configurations also contain two more fields: `release` and `boot_style`. The `release` field is not editable and is populated by the system whenever the kernel file is uploaded, based on the Linux `file` command output. The `boot_style` dictates how the nodes receive the root file system, although it can be overridden by the `_boot_style` attribute set at the node level or in any attribute groups used by the node. See *Reserved Attributes* for details. The possible values for `boot_style` are `rwrpm` (default), `roram`, `iscsi`, `disked`, `live`, `next`, and `sanboot`.

- `rwrpm` is the default value. It instructs the system to download the compressed image into compute node RAM where the `mount_rootfs` script unpacks it during the boot process.
- If the `roram` option is provided, the script downloads a squashfs image into compute node RAM, combines this with a writable tmpfs via overlays, and boots using that combined file system.
- The `iscsi` option instructs the node to mount a read-only image via iSCSI and apply a writeable overlay.
- The `disked` option allows a node with local storage to both employ a node-local persistent cache to retain downloaded images and unpack images onto a node-local partition. Using a cache avoids the need to download images at boot time, and booting from a local partition frees the RAM that would otherwise hold the compute node image. See *Booting From Local Storage Cache* for details.
- The `live` and `next` options are most useful when kickstarting locally installed nodes. The `live` option can be applied to a boot configuration that points to a repo based on an uploaded Rocky or RHEL ISO. Nodes booted `live` from such a configuration use the kernel and `inittarfs` from the ISO with an `inst.repo` kernel option to boot into the ISO's Anaconda-based installer. Given access to the node console, you can manually install to the local disk, thereby generating a kickstart file that can be used to reinstall this or similar nodes at a later time. The BIOS of such kickstarted nodes should be configured to boot from the network and then from local disk. In this configuration the `next` boot style should cause the compute node(s) to initially attempt to PXE boot, but then fail and try to boot their local disk. See *Using Kickstart* for details about kickstarting locally installed nodes.

When booting a compute node into either a kickstart or live configuration, certain Anaconda options can be provided on the command line through the `cmdline` field in the boot config or node. For example, if the `inst.sshd` option is included on the `cmdline` when a node uses a boot configuration made from an ISO-based repo, then you can log into the node during a "live" boot or during the node kickstart process. Be aware that there is no root password required by default, but it can be set in a kickstart file.

Similarly, the `inst.vnc` Anaconda argument tells the booting node to start a VNC server to monitor the kickstart process or click through a manual install.

See <https://anaconda-installer.readthedocs.io/en/latest/boot-options.html> for documentation and additional options.

- Depending on BIOS details, some locally installed systems will not properly handle the `next` boot style and will halt instead of failing over to another boot device. In that case, the `sanboot` option can be used to trigger booting of the first partition of the first disk. The `sanboot` option behavior is customized using the `_ipxe_sanboot` attribute described in *Reserved Attributes*.

You can override the `boot_style` setting for an individual or group of nodes by assigning a `_boot_style` attribute. Similarly, to avoid overlays and use the `rwtab` approach to providing write capabilities to read-only root file systems, set the `_boot_rw_layer` attribute of a node or attribute group to `rwtab`.

4.25.1 Create Local Repo

ISO images of the installation DVDs for RHEL-family (see *Supported Distributions and Features*) systems can be downloaded from their respective websites and imported into the ICE ClusterWare™ software as repos using the `scyld-clusterctl repos` command. For example:

```
scyld-clusterctl repos create name=Rocky8repo iso=@Rocky-8.7-x86_64-dvd1.iso
```

Once the upload completes, the ISO will be automatically forwarded to all head nodes and will be locally mounted on each. Below are the repository details immediately after the upload completes:

```
[cadmin@virthead]$ scyld-clusterctl repos -i Rocky8repo ls -l
Repos
  Rocky8repo
    iso
      chksum: e47d5ca236a3152d63814b32081a1a3261dd1cf4
      filename: 4aceb9db1aef4670be82bf49855b514a
      mtime: 2023-01-31 23:07:36 UTC (0:08:31 ago)
      size: 11.3 GiB (12129927168 bytes)
      isolabel: Rocky-8-7-x86_64-dvd
      keys: []
      name: Rocky8repo
      urls
        <BASE_URL>/isomount/Rocky8repo/BaseOS/
        <BASE_URL>/isomount/Rocky8repo/AppStream/
```

The ClusterWare platform displays URLs for the repositories identified on the ISO. For example, there is a repository in the root of the uploaded Rocky-8 ISO, accessible at `<BASE_URL>/isomount/Rocky8repo/BaseOS/`. The `<BASE_URL>` tag is used as a placeholder to signify that any head node can provide access. When using such a URL, replace the `<BASE_URL>` with the actual head node's base URL, e.g., `http://10.20.30.30/api/v1/isomount/Rocky8repo/BaseOS/<target-file>`

The ISO can also be downloaded using the `scyld-clusterctl` command:

```
scyld-clusterctl repos -i Rocky8repo download iso
```

Just like URL defined repos, repos created using ISOs can be referenced in distros. See *Creating Images in Boot Configurations* in this guide for details about using repos and distros to create compute node images.

4.25.2 Create Boot Configuration with Kickstart

In addition to providing content for distros, repos based on RHEL-family ISO images can also be used to kickstart locally installed compute nodes. To prepare a kickstart configuration, create a boot configuration that references the repo directly:

```
scyld-bootctl create name=Rocky8boot repo=Rocky8repo kickstart=basic.ks
```

The resulting boot configuration will automatically locate the kernel and initramfs on the ISO and default to using no image:

```
[cadmin@virthead]$ scyld-bootctl -i Rocky8boot ls -l
Boot Configurations
  Rocky8boot
    image: none
    initramfs: repo:images/pxeboot/initrd.img
```

(continues on next page)

(continued from previous page)

```
kernel: repo:images/pxeboot/vmlinuz
kickstart: basic.ks
last_modified: 2023-01-31 23:27:08 UTC (0:00:13 ago)
name: Rocky8boot
release: 4.18.0-425.3.1.el8.x86_64
repo: Rocky8repo
```

Initially this boot configuration can be used to boot a disked node with the *live* boot style assigned either by the boot configuration *boot_style* field or a *_boot_style* node attribute. When live booting a node, the cluster administrator will need to access the node's console to proceed through the operating system installation steps. To use the serial-over-lan BMC feature, the administrator may need to provide an appropriate *console=* cmdline, e.g.:

```
scyld-bootctl -i Rocky8boot update cmdline=console=ttyS0,115200
```

The specific details of the console and other command line arguments depend on the target hardware and are beyond the scope of this document. Once the installation process is complete, the compute node should use a *next* boot style in order to skip the PXE boot process and instead boot from the next boot device. Cluster administrators are encouraged to configure the BIOS of locally installed compute nodes to attempt PXE boot first and then boot from the local disk so that the *next* boot style works as intended.

4.25.3 scyld-mkramfs

NAME

scyld-mkramfs -- Tool to create an initial root file system image.

USAGE

scyld-mkramfs

```
[-h] [OPTION] ... `` ``-o, -output PATH
```

STANDARD OPTIONS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- update, -u BOOT** Space-separated list of boot configurations to update.
- output, -o PATH** Where to write the initramfs.
- image, -i IMGID** Uses `scyld-modimg` to install the *clusterware-tools* package inside the image *IMGID*, and then executes `scyld-mkramfs` inside the image to create the root file system image.
- kver VERSION** Specify the kernel version to use, overriding the default of the head node's current kernel (viewed with `uname -r`).

ADVANCED OPTIONS

- stripped** Exclude all network drivers not loaded on some node.
- drivers NAMES** Space-separated list of additional kernel drivers to include.
- modules NAMES** Space-separated list of additional dracut modules to include.
- ramfs-conf PATH** Use a config file from a non-standard location *PATH*.
- base-url URL** Specify the base URL of the ClusterWare REST API.

EXAMPLES

```
scyld-mkramfs --update OpenMPI-Slurm-Boot
```


Rebuild the initramfs used by the OpenMPI-Slurm-Boot boot configuration.

```
scyld-mkramfs --update OpenMPI-Slurm-Boot --drivers mlx4_core
```

Rebuild the initramfs used by the OpenMPI-Slurm-Boot boot configuration after adding the `mlx4_core` driver (and its dependencies).

```
scyld-mkramfs --update OpenMPI-Slurm-Boot --kver 3.10.0-1160.45.1.el7.x86_64
```

Rebuild the initramfs in the boot config after an administrator installs a new kernel into the image that the boot config is using. The `--kver` argument is needed if there are multiple kernels installed in the image.

RETURN VALUES

Upon successful completion, `scyld-mkramfs` returns 0. On failure, an error message is printed to `stderr` and `scyld-mkramfs` returns 1.

4.25.4 scyld-bootctl

NAME

`scyld-bootctl` -- Query and modify boot configurations for the cluster.

USAGE

`scyld-bootctl`

```
[-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
[--human | --json | --csv | --table] [--pretty | --no-pretty] [--show-uids] [-a |
-i BOOTGROUPS] {list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm,
download, export, import, mkiso}
```

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.
- show-uids** Do not try to make the output more human readable.
- a, --all** Interact with all boot configurations (default for list).
- i, --ids BOOTGROUPS** A comma-separated list of boot configurations to query or modify.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]** Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

- human** Format the output for readability (default).
- json** Format the output as JSON.
- csv** Format the output as CSV.
- table** Format the output as a table.
- pretty** Indent JSON or XML output, and substitute human readable output for other formats.

--no-pretty Opposite of --pretty.

ACTIONS ON BOOT CONFIGURATION(S)

list (ls) [--long | --long-long | --raw]

List information about boot configurations.

-l, --long Show a subset of all optional information for each node.

-L, --long-long Show all optional information for each node.

--raw Display the raw JSON content from the database.

create (mk) [--content [JSON | INI_FILE]] [NAME=VALUE] ...

Add a boot configuration with optional *JSON* or *INI_FILE* content or with optional *NAME / VALUE* pairs.

--content [JSON | INI_FILE]

Load this *JSON* or *INI_FILE* content into the database as a boot config.

clone (cp) [--content [JSON | INI_FILE]] [NAME=VALUE] ...

Copy boot configuration with optional *JSON* or *INI_FILE* content or with optional *NAME / VALUE* identifiers.

--content [JSON | INI_FILE]

Overwrite fields in the cloned boot config.

update (up) [--content [JSON | INI_FILE]] [NAME=VALUE] ...

Modify boot configuration *NAME* field(s) with new value(s).

--content [JSON | INI_FILE]

Overwrite this content into the database for a boot config.

replace (re) [--content [JSON | INI_FILE]] [NAME=VALUE] ...

Replace all boot configuration fields.

--content [JSON | INI_FILE]

Overwrite this content into the database for a boot config.

delete (rm) [-r, --recurse]

Delete boot configuration(s).

-r, --recurse Optionally also delete the referenced image or iso-based repo.

download [--dest DIR] FILENAME ...

Extract named file(s) (any of "initramfs", "kernel") from boot config and download to current working directory (or to directory *DIR*).

export [--no-recurse] [PATH]

Export the specified boot configuration *NAME* to the file *NAME.export* in the current working directory or in destination *PATH*.

--no-recurse Do not recurse through and include dependencies.

import [--no-recurse] [--boot-config NAME_BOOT] [--image NAME_IMG] NAME.export

Import the *NAME.export* file into a local boot configuration (default embedded in *NAME.export*, or optionally renamed *NAME_BOOT*) and associated compute node image (or optionally renamed *NAME_IMG*).

mkiso [--image NAME] [--output PATH]

Create a bootable ISO from the boot configuration.

--image NAME Use a different image with this boot configuration.
--output PATH Where to save the resulting ISO.

EXAMPLES

```
scyld-bootctl create name=Fed29Boot \  
kernel=@/boot/vmlinuz-4.20.6-200.fc29.x86_64 \  
initramfs=@cw-ramfs-4.20.6-200.fc29.x86_64
```

Create a boot configuration with a premade kernel and initramfs.

```
scyld-bootctl -iFed29Boot download kernel
```

Download the kernel previously uploaded to the Fed29Boot configuration.

```
scyld-bootctl -iFed29Boot update \  
initramfs=@new-ramfs-4.20.6-200.fc29.x86_64 \  
description="Ramfs created Fed24
```

Replace the initramfs with a new one.

```
scyld-bootctl -i DefaultBoot ls -l
```

Display details about the DefaultBoot configuration.

```
scyld-bootctl -i DefaultBoot update cmdline="enforcing=0 console=ttyS0,115200"
```

Update the *cmdline* that is passed to a booting kernel to a new value. Note that `update` changes the entire *cmdline*, so to append a new substring to an existing *cmdline*, first view the full boot config (as noted in the example above), then form a new *cmdline* string with existing pieces you wish to retain.

```
scyld-bootctl -i SlurmBoot export  
mv SlurmBoot.export ExportSlurmBoot
```

Export the boot config SlurmBoot and associated image as file SlurmBoot.export, and rename that file to ExportSlurmBoot. Note that the boot config name is embedded in ExportSlurmBoot as "SlurmBoot".

```
scyld-bootctl import ExportSlurmBoot
```

Import the ExportSlurmBoot contents to a different cluster as a new SlurmBoot boot config and associated compute node image.

```
scyld-bootctl import --boot-config Slurm19Boot ExportSlurmBoot
```

Import the ExportSlurmBoot contents to a different cluster as a new Slurm19Boot boot config and associated compute node image.

```
scyld-bootctl import --boot-config Slurm19Boot --image Slurm19Image ExportSlurmBoot
```

Import the ExportSlurmBoot contents to a different cluster as a new Slurm19Boot boot config and associate compute node image with new name Slurm19Image.

```
scyld-bootctl import --boot-config Slurm19Boot --no-recurse ExportSlurmBoot
```

Import the ExportSlurmBoot contents to a different cluster as a new Slurm19Boot boot config without including the embedded image.

RETURN VALUES

Upon successful completion, `scyld-bootctl` returns 0. On failure, an error message is printed to `stderr` and `scyld-bootctl` returns 1.

4.25.5 Freezing a Boot Configuration

You can "freeze" a boot configuration to block future changes to it by setting the `frozen` field to true (example boot configuration named `xyzBoot`):

```
scyld-bootctl -i xyzBoot update frozen=true
```

This blocks updates to any field in the boot configuration, stops any field data from being erased, and prevents the boot configuration from being deleted.

To re-enable changes, set `frozen` back to false (the default):

```
scyld-bootctl -i xyzBoot update frozen=false
```

Anyone who can set `frozen=true` can also set it to `false` and thus this mechanism primarily protects against *accidental* changes to "known good" boot configurations. It does not provide significant protection against malicious attacks.

4.25.6 Deleting Boot Configurations

Boot configurations contain only a kernel and `initramfs` and consume only a few tens of megabytes. Permanently delete an unwanted boot configuration with the following command (example boot configuration named `xyzBoot`):

```
scyld-bootctl -i xyzBoot delete
```

4.25.7 Exporting and Importing Boot Configurations Between Clusters

A multiple head node cluster contains cooperating head nodes that share a replicated database and transparent access to peer boot configurations, kernel images, and `initramfs` files. See *Managing Multiple Head Nodes* for details. There is no need to manually copy boot configs between these head nodes.

However, it may be useful to copy boot configurations from a head node that controls one cluster to another head node that controls a separate cluster, thereby allowing the same boot config to be employed by compute nodes in the target cluster. On the source head node the administrator "exports" a boot config to create a single all-inclusive self-contained file that can be copied to a target head node. On the target head node the administrator "imports" that file into the local cluster database, where it merges with the local head node's existing configs, images, and files.

Important

Prior to exporting/importing a boot configuration, you should determine if the boot config and kernel image names on the source cluster already exist on the target cluster. For example, for a boot configuration named `xyzBoot`, execute `scyld-bootctl -i xyzBoot ls -l` on the source head node to view the boot config name `xyzBoot` and note its image name, e.g., `xyzImage`. Then on the target head node execute `scyld-bootctl ls -l | egrep "xyzBoot|xyzImage"` to determine if duplicates exist.

If any name conflict exists, then either (1) on the source head node create or clone a new uniquely named boot config associated with a uniquely named image, then export that new boot config, or (2) on the target head node import the boot config using optional arguments, as needed, to assign unique name or names.

To export the boot configuration `xyzBoot`:

```
scyld-bootctl -i xyzBoot export
```

which creates the file `xyzBoot.export`. If there is no name conflict(s) with the target cluster, then on the target head node import with:

```
scyld-bootctl import xyzImage.export
```

If there is a name conflict with the image name, then perform the import with the additional argument to rename the imported image:

```
scyld-bootctl import xyzImage.export --image uniqueImg
```

or import the boot config without importing its embedded image at all (and later associate a new image with this imported boot config):

```
scyld-bootctl import xyzImage.export --no-recurse
```

If there is a name conflict with the boot config name itself, then add:

```
scyld-bootctl import xyzImage.export --boot-config uniqueBoot
```

Associate a new image name to the imported boot config if desired, then associate the boot config with the desired compute node(s):

```
scyld-nodectl -i <NODES> set _boot_config=xyzBoot
```

4.25.8 Using Kickstart

The *Boot Configurations* section discusses the creation and modification of compute node images and how to add locally installed nodes to the system. The ICE ClusterWare™ platform provides support for an additional method of dynamically provisioning compute nodes: Kickstart.

4.25.8.1 Kickstart Files

A kickstart file configures how a kickstarted node is initialized at boot time. The files are stored on the head node in the `/opt/scyld/clusterware/kickstarts/` folder. In a multihead configuration this folder must currently be manually synced between head nodes. These kickstart files typically contain a limited set of variables that will be substituted at download time.

For example, the ClusterWare package includes the `basic.ks` kickstart file that begins with the following contents:

```
# Node fields, attributes, status, and hardware are available in
# dictionaries referenced by name or initial. Some examples:
#
# <node[ip]>
# <a[_boot_config]>
# <hardware[mac]>

# Perform a SOL friendly text-based install.
text

# Pull some basics from the head node.
url --url <root_url()>
lang <head[lang]>
```

(continues on next page)

(continued from previous page)

```
keyboard <head[keymap]>
timezone <head[timezone]>
```

The angle-bracket notation is used to substitute head-node or compute-node information into the kickstart file, essentially turning it into a kickstart template. When a node begins the kickstart process, it will query the head node for the file and all variables will automatically be substituted as the file is downloaded. This allows admins an opportunity to insert node-specific data into the kickstart. In addition to the `<head[name]>` syntax, node attributes may be referenced with `<a[name]>`, and similar template expansions are available for a node's hardware and status information, as well as other head- and system-specific substitutions. By templating the kickstart file, admins can better generalize those files for a simpler configuration process with fewer node-specific files or commands. For more information, including a complete list of template variables, see *Variable Substitution*.

The simple `basic.ks` performs a boot time install of both the base distribution core packages (e.g., from Rocky8repo) and the associated `clusterware-node` package which is appropriate for that particular base distribution.

Associate the boot configuration with a kickstart file:

```
scyld-bootctl -i Rocky8boot update kickstart=basic.ks
```

and associate a compute node to boot this boot configuration:

```
scyld-nodectl -i n0 set _boot_config=Rocky8boot
```

Finally, reboot node `n0` to initiate the kickstart, which will take a few minutes to complete.

Once the freshly installed node has booted, there will be a `/root/anaconda-ks.cfg` file that can be used as a starting point for creating a more generalized kickstart file. If the cluster administrator would like to reinstall the node the exact same way, the simplest thing to do is copy that `anaconda-ks.cfg` file to the head node's kickstart directory and assign it to be used in the boot configuration:

```
# Copy the compute node's /root/anaconda-ks.cfg to the head node,
# and then copy to the head node's kickstart files folder.
cp anaconda-ks.cfg /opt/scyld/clusterware/kickstarts
# And update the boot configuration to use the file.
scyld-bootctl -i Rocky8boot update kickstart=anaconda-ks.cfg
```

After that file is in place, any compute node booted from that boot configuration without the `next` or `live` boot style will boot using the kernel and `initramfs` from the ISO, and a URL to the kickstart file will be added to the kernel command line. Keep in mind that once a node starts the kickstart process, it is a good idea to change its boot style to `next` so that it does not reboot at the end of the install process and immediately reinstall. Configuring the kickstart process to end with a `shutdown` command (see your operating system documentation) is the current best practice.

If a cluster administrator wants to use a different kernel and/or `initramfs` for kickstarting instead of the ones found on the ISO, those can be replaced just like in any other boot configuration through the `update` action. Updating them with an empty string will reset them back to the detected paths:

```
scyld-bootctl -i Rocky8boot update kernel= initramfs=
```

4.25.8.2 Kickstart Failing

If a node has been configured to kickstart using a boot configuration provided by a repo created from an ISO file but is failing, then check the console output for the node. If the node is entering the "Dracut Emergency Shell" from the dracut timeout scripts, then you will need to retry and see what messages were on screen prior to the "Warning: dracut-initqueue timeout" messages that flood the screen. One common error is "Warning: anaconda: failed to fetch stage2 from <URL>", where the URL points to a repo on the head node. If this message occurs, please check that you have uploaded the correct ISO into the system.

For CentOS and RHEL, the "boot" ISO files such as `CentOS-8.1.1911-x86_64-boot.iso` do contain the files necessary to initiate the kickstart process, but do not contain the full package repositories. The cluster administrator must provide appropriate URLs in the kickstart file, or must upload a more complete ISO such as `CentOS-8.1.1911-x86_64-dvd1.iso` using the `scyld-clusterctl` command. For example, to replace the ISO originally uploaded into a newly created `centos8_repo` repo:

```
scyld-clusterctl repos -i centos8_repo update iso=@CentOS-8.1.1911-x86_64-dvd1.iso
```

4.25.9 Using RHCOS

The ICE ClusterWare™ platform provides support for installing and using RHCOS.

First create a repo from a RHCOS ISO file. For example:

```
scyld-clusterctl repos create name=rhcos iso=@rhcos-4.10.3-x86_64-live.x86_64.iso
```

Once the repo is created, the ISO will be automatically forwarded to all head nodes and will be locally mounted on each. Below are the repository details immediately after the upload completes:

```
[cwwadmin@virthead]$ scyld-clusterctl repos -i rhcos ls -l
Repos
  rhcos
    iso
      checksum: ee4f06946822b55c81c8aa95e21df4f02b9699e8
      filename: 7e9666b05e914b85b59be21f23ce9136
      mtime: 2022-06-17 18:20:52 UTC (0:16:20 ago)
      size: 999.0 MiB (1047527424 bytes)
      isolabel: rhcos-410.84.202201251210-0
      keys: []
      name: rhcos
      urls: []
```

Now create a boot config that uses this repo:

```
scyld-bootctl create name=rhcosBoot repo=rhcos
```

Examine the details of the boot config:

```
scyld-bootctl -i rhcosBoot ls -l
Boot Configurations
  rhcosBoot
    cmdline: coreos.live.rootfs_url=<BASE_URL>/repo/rhcos/content/images/pxeboot/rootfs.
    ↪img coreos.inst.install_dev=<attributes[_coreos_install_dev]> coreos.inst.ignition_url=
    ↪<attributes[_coreos_ignition_url]>
    image: none
    initramfs: repo:images/pxeboot/initrd.img
    kernel: repo:images/pxeboot/vmlinuz
    last_modified: 2022-06-17 18:22:31 UTC (0:03:42 ago)
    name: rhcosBoot
    release: 4.18.0-305.34.2.el8_4.x86_64
    repo: rhcos
```

Note the `_coreos_install_dev` and `_coreos_ignition_url` attributes in the `cmdline`. These attributes are set by the `scyld-nodectl` tool for the specific node(s) that use the `rhcosBoot` boot config.

For example:

```
scylld-nodectl -in0 set _boot_config=rhcosBoot \
    _coreos_ignition_url=http://10.20.30.40/path/path/ignition.ign \
    _coreos_install_dev=/dev/sda
```

For further information and examples about ignition files, see <https://cloud.redhat.com/blog/provision-red-hat-coreos-rhcos-machines-with-custom-v3-ignition-files>.

This variable replacement scheme is similar to the variable replacement in kickstart *.ks files.

4.26 Software Images

4.26.1 Images

An important concept is *local* image versus *remote* image.

The ICE ClusterWare™ backend retains the official copy of file system images, which are termed *remote* images. When a compute node PXE boots, it first downloads the Linux kernel and initramfs provided as part of the boot configuration assigned to the node. Next, the node downloads the *remote* image referenced in the boot configuration from a head node or peer node.

When a tool such as `scylld-modimg` creates or manipulates image contents, the tool manipulates a cached local version of the *remote* image. Per-administrator cache(s) are `~/scylldcw/workspace/`. The tool first downloads a *remote* image into the cache if it doesn't already exist there. Typically a new or modified cached *local* image is uploaded to the database when the creation or modification is complete.

See [Deleting Unused Images](#) for details about how to delete *local* or *remote* images.

4.26.1.1 Images Page

An initial image, *DefaultImage*, is created during initial ICE ClusterWare™ installation. You can create additional images from an ISO or network-accessible package repository.

Use the **Images** page to create and manage images. The page is available via **Provisioning + SW > Images** in the left navigation panel.

Images Refresh Interval (seconds): 10

An Image object represents a file system that will be deployed onto a compute node. It is not intended to have all software that an end-user might want, but a somewhat minimal set of drivers, configuration files, tools, and middleware to bring up the node.

Name ▲	Description ▲
CentOS-7-x86_64-Everything-2009	...
DefaultImage	...
Prod-202402	...
Prod-202502	...

ADD IMAGE

Create an Image

To create an image:

1. Click **Add Image**.
2. Add details about the image.
 - **Name:** Required.
 - **Description:** Optional.
 - **Parent:** Optional. Use to track the origin of the image. No configuration or attributes are inherited when using this field.
 - **Distros:** Optional. Use this to associate the image with the distro used for creation.
 - **Frozen:** Freezing an image prevents future changes, including updates to image details, removing data, or deleting the image. For details, see *Freezing an Image*.
3. Click **Add Image** to save your changes.

The new image appears in the list at the top of the page.

Edit Image

To edit an image:

1. Click the ellipsis (...) on the far right of the row and select the **Edit** action. The **Edit Image** pane populates with the image details.
2. Make updates to the image.
3. Click **Edit Image** to save your changes.

Delete Image

To delete an image, click the ellipsis (...) on the far right of the row and select the **Delete** action.

Related Links

- *Images*
- *scyl-d-modimg*

4.26.1.2 Creating Images

Important

Various commands that manipulate images execute as user root, thereby requiring that the commands internally use `sudo` and requiring that user root must have access to the workspace that contains the administrator's images. Typically the per-user workspace is `~/ .scyl-dcw/workspace/`. If that directory is not accessible to the command executing as root, then another accessible directory can be employed. You can identify that alternative path by adding a `modimg.workspace` setting to `~/ .scyl-dcw/settings.ini`.

The `scyl-d-install` script creates an initial image with the default name *DefaultImage* based on publicly available repositories that match the head node operating system. If these repositories are not accessible, the `scyl-d-add-boot-config` tool can be run later with locally accessible repositories as described in *Creating Local Repositories without Internet*. Once the *DefaultImage* is created, use `scyl-d-modimg` to modify it directly. A safer

approach is to use `scyld-imgctl` to clone the *DefaultImage* to new name, and then use `scyld-modimg` to modify that cloned image, leaving the *DefaultImage* untouched. See [Modifying Images](#) for details. You can also recreate the *DefaultImage*. See [Recreating the Default Image](#) for details.

You can also create a new image from an ISO or network accessible package repository. When doing that, consider the source of the components (packages) for the new image. A *distro* ties together a list of *repos* (package repositories) and an optional *release*. The *package_manager* is determined during image creation, but can be overridden in the *distro*. The initial default *distro* matches the original head node's version, uses *package_manager* `yum`, and downloads packages from a one item *repos* list containing "Rocky_base":

```
[admin@virthead]$ scyld-clusterctl distros ls -L
Distros
Rocky
  name: Rocky
  packaging: rpm
  release: 8
  repos
    Rocky_appstream
    Rocky_base

[admin@virthead]$ scyld-clusterctl repos ls -L
Repos
Rocky_appstream
  keys: []
  name: Rocky_appstream
  urls
    http://dl.rockylinux.org/pub/rocky/$releasever/AppStream/$basearch/os/

Rocky_base
  keys: []
  name: Rocky_base
  urls
    http://dl.rockylinux.org/pub/rocky/$releasever/BaseOS/$basearch/os/
```

Use the following command to create a new image named "NewImg" using the default distro that downloads packages from the latest Rocky yum repo:

```
scyld-modimg --create --set-name NewImg
```

To create a Rocky image that contains something other than the latest Rocky release, see [Creating Arbitrary Rocky Images](#). To create a RHEL image, see [Creating Arbitrary RHEL Images](#).

4.26.1.3 Recreating the Default Image

If you want to recreate the *DefaultImage* that was built by the `scyld-install` tool, first delete the components of the existing image and boot config:

```
scyld-attribctl -i DefaultAttribs rm
scyld-bootctl -i DefaultBoot rm
scyld-imgctl -i DefaultImage rm
```

Then create a new default. If there are no attribute groups defined on this cluster (see [Node Attributes](#)), then run:

```
scyld-add-boot-config --make-defaults
```

Otherwise clear the attributes before running the command.

4.26.1.4 Repos and Distros

One of the steps in the `scyld-install` script is to run the `scyld-clusterctl` tool to define a *distro* prior to creating the first image. The `scyld-modimg` tool only creates images based on defined distros. A *distro* associates one or more repos together with their package manager and an optional release string. If no release string is provided, then any supplied URL should not include the string "\$releasever", as that variable is not be defined during image creation. On a Rocky or RHEL system, the default *repo* and *distro* are created by:

```
scyld-clusterctl repos create name=Rocky_base \  
    urls=http://dl.rockylinux.org/pub/rocky/$releasever/BaseOS/$basearch/os/  
scyld-clusterctl repos create name=Rocky_appstream \  
    urls=http://dl.rockylinux.org/pub/rocky/$releasever/AppStream/$basearch/os/  
scyld-clusterctl distros create name=Rocky packaging=rpm release=8 \  
    repos=Rocky_appstream,Rocky_base
```

Together with the local `/etc/yum.repos.d/clusterware.repo` file, this information is used at image creation time to generate a `/etc/yum.repos.d/clusterware-node.repo` file for the image containing sections referring to both the head node's ICE ClusterWare™ repository and to the *distro*'s repos.

You can create additional repos and distros to make node images based on different upstream sources. To do this, provide multiple comma-separated URLs to the `scyld-clusterctl repos create` command or multiple repos to the `scyld-clusterctl distros create` command. Distros can also be imported from an existing yum repo files. For example:

```
scyld-clusterctl distros import --name Rocky \  
    /etc/yum.repos.d/Rocky-BaseOS.repo /etc/yum.repos.d/Rocky-AppStream.repo
```

The `import` action creates repos based on the contents of the provided yum repo file(s) and then associates all of them with a newly created Rocky9 distro. Any string passed to `--release` is saved into the distro *release* field and is used by yum to replace any occurrences of "\$releasever" in the repo file.

See the the *scyld-clusterctl* `repos` and `distros` actions and the *scyld-modimg* command that is used to actually create and modify images.

Using ISO Releases

Many distributions are distributed in ISO form. Use the `scyld-clusterctl` tool to create an image from an ISO. For example, for an ISO named `Rocky-9.5-x86_64-dvd.iso`, first create a *repo*:

```
scyld-clusterctl repos create name=rocky_9.5_iso \  
    iso=@/path/to/Rocky-9.5-x86_64-dvd.iso
```

Next, create a *distro* that references the new repo:

```
scyld-clusterctl distros create name=rocky_9.5_distro repos=rocky_9.5_iso
```

Finally, create an image using that *repo* and *distro*:

```
scyld-modimg --create rocky_9.5_distro --set-name rocky_9.5_image
```

When this image is booted, the ISO-based repo may not be accessible, and the `/etc/yum.repos.d/clusterware-node.repo` file needs to be modified to use a more permanent repo location.

Using Archived Releases

Many distributions archive individual releases after they are superseded by a newer release, but for this discussion we will examine CentOS. The CentOS project provides packages and updates on their various mirror sites for the most recent release, but deprecates all previous point releases. This means that at the URL where a mirror would nominally keep the previous release, a readme file is provided explaining that the release has been deprecated and pointing users to the CentOS vault for packages. The packages located in the vault are unchanged from when they were "current". The CentOS project also deprecates the release that is two major releases back. For example, as of the release of version 7, version 5 was deprecated. In this way there are always two currently supported versions of CentOS, the latest and the most recent of the previous major release.

What this means for ClusterWare administrators is twofold:

1. To create an image of an archived version of CentOS, create the correct repo and distro objects in the ClusterWare database.
2. After creating an image from the vault, manually modify the yum repo files present in the image.

To create an image based on an archived version of CentOS (7.3 in this example), the steps are:

```
scyld-clusterctl repos create name=CentOS-vault \
    urls=http://vault.centos.org/\$releasever/os/\$basearch/
scyld-clusterctl distros create name=CentOS_7.3 repos=CentOS-vault release=7.3.1611
scyld-modimg --create CentOS_7.3 --set-name CentOS_7.3_img
```

The first command creates a repo called CentOS-value pointing at the generic vault URL. The second command creates a distro that references the CentOS-vault repo and defining the release string. Once the distro exists, it can be referenced by name in the third command to actually create a new image.

Unfortunately, because the CentOS vault packages are identical to when they were the current release, the yum repo files located in the `/etc/yum.repos.d/` directory contain references to `mirror.centos.org` instead of `vault.centos.org`. Manually modify these files after image creation and before running yum commands directly or through the `scyld-modimg --install`, `--uninstall`, `--update`, or `--query`. The above `scyld-modimg --create` command displays an error referring back to this documentation:

```
[admin@virthead]$ scyld-modimg --create CentOS_7.3 --set-name CentOS_7.3_img

Executing step: Create
Preparing the chroot...
...done.
Initializing the chroot...
elapsed: 0:01:11.4
...initialized.
Installing core packages...
elapsed: 0:00:01.0
ERROR: One or more repositories in the newly created image are invalid. This
can happen when installing older versions of Linux distributions such as CentOS.
Please consult the Administrator's Guide for more information.
WARNING: The command will be retried with unknown repositories disabled.
elapsed: 0:02:39.9
fixing SELinux file labels...
...done.
step completed in 0:04:13.6
```

To manually modify the yum repo files, use the `scyld-modimg --chroot` command on an already created image as follows:

```
[admin@virthead]$ scyld-modimg -i CentOS_7.3_img --chroot
Checksumming image 6a8947156e08402ba2ad6e23a7642f4f
  elapsed: 0:00:01.0
Unpacking image 6a8947156e08402ba2ad6e23a7642f4f
  100.0% complete, elapsed: 0:00:29.6 (62.2% compression)
Checksumming...
  elapsed: 0:00:01.0
Executing step: Chroot
Dropping into a /bin/bash shell.  Exit when done.
[root@virthead /]# exit
exit
  fixing SELinux file labels...
(K)eep changes or (d)iscard? [kd]
```

When you exit the shell, the tool confirms that you want to keep the changes made and offers to upload the modified image to head node storage.

Installing Software With Subscriptions

For distributions requiring subscriptions for access to updated packages, note that subscription information in an image is used by all nodes unless removed before upload:

```
hostname nodeTemplate
subscription-manager register --username=$RHUSER --password=$RHPASS
subscription-manager attach --pool=$POOL_ID
yum upgrade -y
yum install $REQUIRED_PACKAGE
subscription-manager remove --all
subscription-manager unregister
subscription-manager clean
```

4.26.1.5 Modifying Images

Once you have an existing image, you can install additional RPMs into that image. A recommended best practice is to rarely and only very carefully modify *DefaultImage* and *DefaultBoot*. Instead, use them as stable baselines from which you clone new images and boot configurations.

The `scyld-modimg` tool supports a rich collection of options. See *scyld-modimg* for details.

For example:

```
scyld-imgctl -i DefaultImage clone name=mpiImage
scyld-add-boot-config --image mpiImage --boot-config mpiBoot
scyld-modimg -i mpiImage --install openmpi3.1
```

Suppose you want to create a new boot config *mpiAltBoot* that references the same *mpiImage*, though is otherwise different than *mpiBoot*. For instance, if you want *mpiAltBoot* to have a different *cmdline*:

```
scyld-bootctl -i mpiBoot clone name=mpiAltBoot

# Note that an updated cmdline replaces the entire existing cmdline,
# so examine the current cmdline:
scyld-bootctl -i mpiAltBoot ls -l | grep cmdline
```

(continues on next page)

(continued from previous page)

```
# and perhaps the current cmdline is "enforcing=0", which you add to a new cmdline:
scyld-bootctl -i mpiAltBoot update cmdline="enforcing=0 console=ttyS1,115200"
```

You can manually customize an image, including installing or removing RPMs and modifying configuration files, by operating on the image inside a chroot:

```
scyld-modimg -i mpiImage --chroot
```

You can also combine commands, ending inside a chroot:

```
scyld-modimg --create --set-name mpiImage --install openmpi3.1 --chroot
```

If `scyld-modimg --chroot` detects a problem accessing or manipulating the *local* image, delete the *local* image (see *Deleting Unused Images*), and then the retry of the operation will download a fresh copy of the *remote* image into the cache. Alternatively, execute `scyld-modimg` and add the `--freshen` argument, which ignores the current cached *local* image and downloads a fresh copy.

Inside the chroot, execute as user `root` and manually add, update, or remove rpms with `yum` (or other appropriate package manager), modify configuration files, etc. When you `exit` the chroot, you are asked if you want to discard or keep the changes. If you keep the changes, then you are asked whether or not you want to replace the *local* image, to upload the *local* image, and to replace the *remote* image.

Note

Keep in mind that several directories in the image do **not** get repacked and saved into the image file after an `exit`. Among them are `/tmp/`, `/var/tmp/`, and `/var/cache/yum`.

If your intention is to answer yes to all the questions following your `exit`, then you can skip those questions by adding more arguments to the original command line:

```
scyld-modimg --create --set-name mpiImage --install openmpi3.1 --chroot \
--no-discard --overwrite --upload
```

Examine the RPM contents of an image without going into a chroot by doing a simple query:

```
# Display the version of 'clusterware-node' in the image
scyld-modimg -i mpiImage --query clusterware-node

# Display the version of all RPMs in the image
scyld-modimg -i mpiImage --query
```

Finally, you must set the `_boot_config` attribute for specific nodes, or for all nodes, as desired to use this new boot config. For example, to have nodes `n0-n15` use the `mpiBoot` boot config:

```
scyld-nodectl -i n[0-15] set _boot_config=mpiBoot
```

The `scyld-modimg` command prompts you about whether to overwrite an existing image or create a new one. It also prompts about whether to upload the resulting file to the head node, optionally overwriting the image stored on the ICE ClusterWare™ head node. This tool operates on a local cache of the image and cannot be used to delete an image from the head nodes or to directly modify the name or description of an image on the head node. To modify these sorts of fields, use the `scyld-imgctl` tool.

Images are stored in the head node's `/opt/scyld/clusterware/storage/` directory in *cwsquash* format, which consists of a squashfs image offset inside a pseudo-disk image. This format is suitable for exporting via iSCSI.

Small homogeneous clusters may use a single node image across all compute nodes, although larger clusters that include compute nodes with differing hardware will require additional customization that may not be applicable to all nodes. Although you might find that node attributes (discussed in more detail in *Interacting with Compute Nodes*) and customized boot-time scripting provide adequate image customization, it may be useful (or necessary) to create additional boot configurations and root file systems that meet specific hardware and/or software needs.

Customization can involve more than adding software drivers to support node-specific hardware and adding applications and their associated software stacks. It can also involve customizing configuration files in an image to deal with a non-standard networking environment. For example, if the compute node needs to use a networking route that is not the gateway defined in the head node's `/opt/scyld/clusterware-iscdhcp/dhcpd.conf.template`, then you can edit that file to modify the default `option routers <GATEWAY>;` line, or edit the compute node image's appropriate `/etc/sysconfig/network-scripts/ifcfg-*` script to insert the desired GATEWAY IP address. For more details see the documentation for your base distribution.

4.26.1.6 Caching in `scyld-modimg`

To provide the best performance, the `scyld-modimg` command keeps a local cache of images within the `~/ .scyldcw/ workspace` directory. This directory contains a `manifest.json` file that lists the cached images. An image is added to the list whenever a cluster administrator downloads it. When the `scyld-modimg` command executes, it checks the workspace directory for images that are older than an hour and identical to the same images, matched by UID and name, stored by the head node. Images matching these criteria are evicted from the cache.

When you run the `scyld-modimg` command, the tool checks the local cache for a copy of the image based on the identifier provided, usually the image name. If a match is found, that image is unpacked for modification (to avoid a fresh download from the head node). You can modify the image in stages without losing changes in between, even if you do not upload the image to the cluster at each step.

This behavior can cause unexpected conflicts in very specific circumstances. If you modify an image and then delete that image from the cluster using the `scyld-imgctl` or `scyld-bootctl` commands, the local cache still contains a copy. This means that the next time the `scyld-modimg` command executes it will see the locally cached image and, regardless of the age, not delete it because an identical image is no longer available from the head node.

If you then create a new image by the same name, perhaps through cloning an existing image using the `scyld-imgctl` command, and attempt to modify that image using `scyld-modimg`, you will actually be modifying the local cache due to the name match. When you attempt to upload this modified image, the upload will fail because of a UID mismatch. For example:

```
ERROR: No image found for ID=d67501a26509486ebaad00827d7fac23
```

The simplest way to resolve this problem is to delete the locally cached image:

```
scyld-modimg -i <IMAGENAME> --delete
```

Then re-run the `scyld-modimg` command to modify the image. Since the local cache no longer contains an image with a matching name, the tool will download a fresh copy from the head node and record the correct UID in `manifest.json`. This is almost always the desired solution since you have likely modified the wrong image without realizing it.

If you do want to keep the changes, the simplest approach is to identify the new image UID:

```
scyld-imgctl --show-uids ls <IMAGENAME>
```

Then replace the image UID in the `~/ .scyldcw/ workspace/ manifest.json` file, rename the image file itself to the new UID, and reattempt the upload:

```
scyld-modimg -i <IMAGENAME> --upload
```


Once this command completes, the old image content overwrites the new image content and the changes made previously are preserved.

4.26.1.7 Updating the Kernel in an Image

Compute nodes that boot over the network download their kernel and initramfs at boot time from their parent head node (the first head node to respond to their DHCP request). The required kernel, initramfs, command line, and a reference to an image are combined within a ICE ClusterWare™ boot configuration that can be assigned to the nodes.

To update kernels or other packages within an image, named `Prod202404` in this example, run either:

```
scyld-modimg -i Prod202404 --chroot --overwrite --upload
```

Or

```
scyld-modimg -i Prod202404 --update --overwrite --upload
```

The first is interactive and requires running the `dnf` or `yum` update commands inside the `chroot`, whereas the second attempts the updates, but may suppress some of the errors. If the kernel inside the image is updated, then the boot configuration also needs to be updated. Assuming the newly installed kernel's version is `5.14.0-362.24.1.el9_3.x86_64` and the boot configuration is called `ProdBoot202404`, use the `scyld-mkramfs` command to update the boot configuration:

```
scyld-mkramfs --kver 5.14.0-362.24.1.el9_3.x86_64 --update ProdBoot202404
```

Note

This new ClusterWare *initramfs* file is not the same as a similarly named "initramfs" file in the head node `/boot/` directory, which is associated with a kernel in the `/boot/` directory. This ClusterWare *initramfs* file is associated with a specific image and boot config and it contains custom ClusterWare scripts that execute at boot time.

Without the `--kver <KVER>` options, the tool attempts to select the most recent kernel based on version directories found in `/lib/modules` within the image, so explicitly selecting the version is not required. This command examines the boot configuration, extracts the kernel from the image, uses `dracut` within the image to build a new `initramfs`, extracts that as well, and then uploads both into the boot configuration.

Alternatively, you may want to add a new boot configuration. For example, you may want to boot different kernels within the image, or you chose to upgrade a cloned copy of the original image. In this case, the `scyld-add-boot-config` command can be used:

```
scyld-add-boot-config --image Prod202404Cloned -boot-config ProdBoot202404New
```

This command uses `scyld-mkramfs` internally to extract the kernel and generate the `initramfs`, then uses those files to construct the new boot configuration with a default command line and a reference to the named image, `Prod202404Cloned` in this example.

Check the `release` field of the boot configuration after it is created to see the kernel version. Any nodes assigned to use this boot configuration will boot using that kernel. If a node is then booted and does not use the correct kernel, confirm that the node's `_boot_config` attribute references the correct boot configuration.

4.26.1.8 Updating Drivers Inside Images

The ICE ClusterWare™ platform uses images to provision compute nodes. Because of this, any drivers, applications, or libraries required to run the compute node hardware or jobs need to be available to the running compute node, not the head node(s). To assure availability, software needs to be installed into the image or onto some form of cluster

shared storage. Drivers are more commonly installed into the image while applications and libraries are installed to shared storage and accessed through the module command.

When installing software into an image there are two approaches available. The most common is installing into the image via the `scyld-modimg` command, commonly via the `--chroot` option. In rare cases, some software can only be installed on a running node. In these cases, the image be captured using the `scyld-modimg --capture` command.

For example, to install a package called `prod-install.sh` within an image named `Prod202404` using the `chroot` method, run:

```
scyld-modimg -iProd202404 --copyin prod-install.sh /root --chroot --upload --overwrite
```

The tool unpacks the image into a local workspace directory within your home and `chroot` into it after bind mounting necessary system paths. Once inside the `chroot`, the `prod-install.sh` file is copied into `/root` and you can complete the necessary steps to install the software.

Some types of software try to build kernel modules for the currently running kernel. Within a `scyld-modimg --chroot`, that may be incorrect because the current kernel is actually the host kernel and may not match the kernel running on the booted compute node. Most installers provide some command line option to allow you to specify the target kernel, but for installers that do not, the kernel version can be specified immediately after the `--chroot` argument:

```
scyld-modimg -iProd202404 --copyin prod-install.sh /root --chroot <KVER> --upload --
↪overwrite
```

Specifying the kernel version causes the ClusterWare software to replace the `uname` command inside the `chroot` with a wrapper that outputs the specified kernel version in place of the one detected by the actual `uname` command. This is usually adequate to trick even stubborn installers into using the correct kernel. In the rare case of an installer that still fails, `ssh` into a running node, install the software there, and then capture the file system to a new image via:

```
scyld-modimg --capture <NODE> --set-name <IMAGE> --chroot --upload
```

This command uses `ssh` to connect to the running node and run scripts on the node. These will copy the contents of the local file systems, unpack them into a local directory, and then `chroot` into that directory. Within that `chroot`, you can make further changes before the captured image is uploaded. Note that capturing a running node does run the risk of capturing node-specific details, so installing software within the `chroot` is preferable.

4.26.1.9 Capturing and Importing Images

You can modify the files on a booted compute node and use the `scyld-modimg --capture` command to capture those changes into the image. You can capture the node into an existing image or into a new image. First, confirm that the node being captured is idle to reduce the chance of capturing an image in some intermediate state, then run the capture command. For example, to capture node `n0`, run the following command:

```
scyld-modimg --capture n0 --set-name NewImage
```

This process may take several minutes. During that time the `scyld-pack-node` tool is executed on the compute node via the `scyld-nodect1 exec` mechanism. The result is streamed back to the `scyld-modimg` command that then uploads it to the head node, potentially replacing an existing `NewImage` contents. The `scyld-pack-node` tool captures all files on the node's `/` mount, but does not walk other mounted file systems to ensure that any shared storage is not accidentally captured.

You also need to create a boot config for this captured image. For example:

```
scyld-add-boot-config --image NewImg --boot-config NewBoot
```

Manual work is likely required to generalize the captured image as the process may capture details specific to the compute node. Due to this hazard, future ICE ClusterWare™ releases may expand what files are excluded during image capture.

RHEL 7 clones use a version of RPM too old to properly interpret RHEL 9 packages, so if you are trying to create an image, you may want to kickstart a diskful node and then use `scyld-modimg --capture` to create the image. You must comment out or delete the node-specific lines in `/etc/fstab` created during the kickstarted installation.

4.26.1.10 Automating Common Image Tasks

The `scyld-modimg --run` command allows cluster administrators to collect command line arguments into a file and execute those commands as a very simple script. Any step listed in the “image mutations” section of `scyld-modimg` can be included in a run script using the full step name, minus the leading dashes, with any necessary arguments included on the same line.

Although the command and script does not allow for flow control statements including conditionals and loops, this functionality can be useful in automating repetitive steps in image creation. For example, the file below could be used to copy in a more complicated script, execute it, and copy the results out of the image into the local directory.

```
copyin test-script.sh /tmp
execute /tmp/test-script.sh
copyout /tmp/results.log .
```

Use the following command to run the script:

```
scyld-modimg -i SlurmImage --run @commands.modimg --discard
```

The command is a shorter equivalent of the following command:

```
scyld-modimg -i SlurmImage --copyin test-script.sh /tmp --exec /tmp/test-script.sh --
↪copyout /tmp/results.log . --discard
```

4.26.1.11 Deploying Images Using Ignition

To partition nodes during their boot process, the ICE ClusterWare™ platform leverages tools called Ignition and Butane. Both originate from the Red Hat CoreOS project, but are generally applicable for “first boot” actions such as provisioning local storage. The Ignition tool accepts a machine parsable JSON formatted configuration file while the Butane tool translates a human readable YAML format and produces the JSON file that ignition needs. Details of these projects can be found at:

- Butane: <https://coreos.github.io/butane/>
- Ignition: <https://coreos.github.io/ignition/>

The functionality provided by Ignition enables the ClusterWare platform to partition and format local disks and deploy a root file system image to the newly prepared drives. This bypasses the previously used kickstart process to deploy locally-installed systems. The behavior of Ignition is controlled through a configuration file and node attributes. Another attribute triggers the installation of the GRUB2 bootloader, resulting in a fully provisioned and ready-to-boot system. By default these systems still use the ClusterWare platform for DHCP and report status back to the head nodes, but post-install changes can reconfigure the networking or disable the ClusterWare and Telegraf agents.

To utilize Ignition during provisioning, create an image that contains the desired operating system and software tools. Note that older ClusterWare images may not contain all the tools, such as `sgdisk` or `mkfs.xfs`, necessary for Ignition to properly partition or format the drives. Ignition is also relatively new and may rely on versions of tools that are not available on older operating systems.

Once the image exists, create a boot configuration using the image:

```
scylld-add-boot-config --image <IMAGENAME> --boot-config <BOOTNAME>
```

Several node attributes must also be set on the target node:

Attribute	Example	Notes
<code>_boot_config</code>	DeployBoot	Name of the boot configuration to deploy
<code>_boot_style</code>	disked	Must be "disked"
<code>_disk_root</code>	LABEL=root	Reference to the new root partition / file system
<code>_ignition</code>	config.butane	Name of the Butane configuration file ending in .butane
<code>_bootloader</code>	grub	Must be "grub"

Once these attributes are set, reboot the node. The node will boot using the kernel and initramfs from the specified boot configuration, and within the initramfs the system will execute the various Ignition stages during the normal image download and unpacking process as follows:

- After networking is established and the node has downloaded its initial `attributes.ini` file, it will detect the `_ignition` attribute and download the ignition binary from the head node.
- After downloading the root file system, the node will run the “fetch”, “kargs”, and “disks” stages to download the configuration, check the kernel arguments, and partition local disks.
- Immediately before unpacking the file system, the node will run the “mount” stage and then proceed to unpack the image into the mounted partitions.
- After unpacking the image, the node will run the “files” stage to manipulate the unpacked file system.

Once Ignition completes, the `_bootloader` attribute will trigger installation of the GRUB2 bootloader as a separate step. A successful installation of grub will also update `_boot_style` to ensure nodes do not continually loop through deployment. Finally, a reboot and relabel of the file system is triggered.

The Butane configuration needs to be copied to the `/opt/scyld/clusterware/kickstarts` directory on all head nodes. As long as the file name ends with “.butane” the file will be translated by the Butane command at download time. A file with any other extension must be a valid Ignition configuration file, as it will be passed directly to the Ignition command without translation.

Examples of Butane configuration files can be found in the Butane project documentation:

```
https://coreos.github.io/butane/examples/
```

The `clusterware-tools` package also includes a very simple Ignition example file at `/opt/scyld/clusterware-tools/examples/partitions.butane`. That example file will partition the local drive (`/dev/vda` on a VM) into two partitions, `/` and `/boot` where `/boot` is 1024 MB in size and `/` encompasses the rest of the drive.

Ignition provides functionality beyond simple disk partitioning and file system creation, including software RAID or LUKS configuration, and the creation of systemd unit files. However, these features are considered experimental within the ClusterWare system. Additional functionality will be confirmed in future releases.

An `ignition.log` file is produced during the deployment process and can be found in `/opt/scyld/clusterware-node/atboot` on the booted node. In the case of errors the node will likely not be able to boot. In that case, a cluster administrator can log into the initramfs shell during the 20 second “press any key” countdown and examine the log file located at `/atboot/ignition.log`.

One substantial difference between using Ignition directly and using it via the ClusterWare platform is that Ignition is intended to either produce exactly the desired result or nothing at all. However, when triggered by the `_ignition` attribute, the tool will log failures and proceed. Future releases may change this behavior to more closely match the standard Ignition approach.

4.26.1.12 Freezing an Image

You can "freeze" an image to block future changes to it by setting the `frozen` field to `true` (example image named `xyzImage`):

```
scyld-imgctl -i xyzImage update frozen=true
```

This blocks updates to any field in the image, stops any field data from being erased, and prevents the image from being deleted.

To re-enable changes, set `frozen` back to `false` (the default):

```
scyld-imgctl -i xyzImage update frozen=false
```

Anyone who can set `frozen=true` can also set it to `false` and thus this mechanism primarily protects against *accidental* changes to "known good" images. It does not provide significant protection against malicious attacks.

4.26.1.13 Deleting Unused Images

Compute node images consume significant storage space. *Remote* images are replicated among cooperating head nodes and are the files downloaded by PXEbooting compute nodes. A *local* image is a cached copy of a remote image that was downloaded when an administrator viewed or modified the image. Deleting a local image does not affect its remote version and merely causes it to be re-downloaded from the head node if and when an administrator subsequently views or modifies it.

To view the list of *local* and *remote* images, run:

```
scyld-modimg ls
```

To delete a local cached image `xyzImage`, run:

```
scyld-modimg -i xyzImage --delete
```

To delete all cached images, run:

```
scyld-modimg --all --delete
```

Neither of these commands deletes or otherwise affects the *remote* images.

To permanently delete an unwanted *remote* image, run:

```
scyld-imgctl -i xyzImage delete
```

4.26.2 scyld-imgctl

NAME

scyld-imgctl -- Query and modify images for compute nodes.

USAGE

scyld-imgctl

```
[-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
[--human | --json | --csv | --table] [--pretty | --no-pretty] [--show-uids] [-a | -i
IMAGES] {list,ls, create,mk, clone,cp, update,up, replace,re, delete,rm, download,
stat, capture, lock}
```

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.
- show-uids** Do not try to make the output more human readable.
- a, --all** Interact with all node images (default for `list`).
- i, --ids IMAGES** A comma separated list of node images to query or modify.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.
- u, --user USER[:PASSWD]** Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

- human** Format the output for readability (default).
- json** Format the output as JSON.
- csv** Format the output as CSV.
- table** Format the output as a table.
- pretty** Indent JSON or XML output, and substitute human readable output for other formats.
- no-pretty** Opposite of `--pretty`.
- fields FIELDS** Select individual fields in the result or error.

ACTIONS ON IMAGE(s)

list (ls)

List information about node images.

create (mk)

Add node image.

clone (cp)

Copy node image to new identifiers.

update (up)

Modify node image fields.

replace (re)

Replace all node image fields.

delete (rm)

Delete node image(s) from the remote cache.

download FILES

Download named files *FILES* (any of "content").

- dest DIR** Optional destination for the downloaded files. (Default is current working directory.)

stat

Print the recorded file stats for an image.

capture [--save *FILE*] [--node *NODE*] [--exclude *PATHS*] [--content *JSON/INI_FILE*] ...

Replace or create an image captured from a running system, adding optional name=value pairs.

--save *FILE*

Save the image locally instead of uploading.

-n, --node *NODE* Select the node to capture.

--exclude *PATHS* Exclude additional paths during image capture, specifying either pathnames or a file @*FILE* that contains a list of pathnames.

--content [*JSON* | *INI_FILE*]

Overwrite fields in the specified image(s).

lock [--release | --lock-msg *MSG*] [--force]

Lock or unlock images

--release Unlock images.

--lock-msg *MSG* Provide a message when locking images.

--force Force lock or unlock images.

EXAMPLES

```
scyld-imgctl -i DefaultImage download content
```

Download the previously uploaded image named DefaultImage.

```
scyld-imgctl -i DefaultImage stat
```

Print the last modified time and size of the previously uploaded image.

```
scyld-imgctl -i DefaultImage clone name=NewImage
```

Clone the DefaultImage to a new NewImage.

RETURN VALUES

Upon successful completion, **scyld-imgctl** returns 0. On failure, an error message is printed to `stderr` and **scyld-imgctl** returns 1.

4.26.3 scyld-modimg

NAME

scyld-modimg -- Tool for manipulating image contents.

USAGE

scyld-modimg

```
[ -h ] [ -v ] [ -q ] [ [ -c | --config ] CONFIG ] [ --base-url URL ] [ [ -u | --user ] USER[:PASSWD] ]
[ --human | --json | --csv | --table ] [ --pretty | --no-pretty ] [ --show-uids ] [ --fields
FIELDS ] [ [ -a | --all ] | [ [ -i | --image ] IMAGE ] ] [ --lock-msg MSG ] [ --freshen |
--download-only [ PATHNAME ] ] [ --overwrite | --no-overwrite ] [ --upload | --no-upload ]
[ --discard | --no-discard ] [ --discard-on-error ] [ --shell SHELL ] [ --pkgmgr CONF ] [ --run
SCRIPT ] [ --clean-local ] [ --register-all ] [ --set-name NAME ] [ --set-description DESC ]
[ --chroot [KVER] ] [ --create [DISTRO] ] [ --delete ] [ --import FILE ] [ --capture NODE ]
[ --install PKGS ] [ --update [PKGS] ] [ --uninstall PKGS ] [ --query [PKGS] ] [ --unpack TARGZ ]
[ --copyin SRC DEST ] [ --copyout SRC DEST ] [ --execute COMMAND ] [ --mount PATH ] [ --unmount
PATH ] [ --write-repos [DISTRO] ] {list,ls}
```

ACTIONS

list (ls)

List information about node images.

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.
- a, --all** Select all local images (default).
- i, --image IMAGE** Or select an image by its name *IMAGE*.

--download-only [PATH]

Download a new local copy and then exit. If *PATH* is provided, then it is overwritten; Otherwise any cached changes are lost.

- lock-msg MSG** Provide a message for locking the image.
- freshen** Discard any cached changes.
- overwrite** Keep the same UID after modifications and overwrite any existing image on upload.
- no-overwrite** Opposite of **--overwrite**.
- upload** Upload the final version. NOTE: This must follow all image manipulations options.
- no-upload** Opposite of **--upload**.
- discard** Discard image changes.
- no-discard** Opposite of **--discard**.
- discard-on-error** Discard if a step fails.
- pkgmgr CONF** Specify a package config file (using the '@' prefix), or pass the config contents as a string *CONF*, to override the default config example seen in */opt/scyld/clusterware-tools/examples/pkgmgr.ini*. A cluster administrator wishing to customize *pkgmgr.ini* should copy that example to another location, then add, delete, and/or modify that copy as desired.
- shell SHELL** Select the shell to use in the image for the mutation operations (default */bin/bash*).
- show-uids** Do not try to make the output more human readable.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

- base-url URL** Specify the base URL of the ClusterWare REST API.

-u, --user USER[:PASSWD]

Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

- human** Format the output for readability (default).
- json** Format the output as JSON.
- csv** Format the output as CSV.

--table	Format the output as a table.
--pretty	Indent JSON or XML output, and substitute human readable output for other formats.
--no-pretty	Opposite of --pretty .
--fields FIELDS	Select individual fields in the result or error.

CACHE MANIPULATIONS

Make changes to the local image cache.

--clean-local	Delete local images not found in the manifest and any temporary files or directories.
--register-all	Record information about all locally stored images.
--set-name NAME	Set the name of the image.
--set-description DESC	Set the description for the image.

IMAGE MUTATIONS

The following steps are performed on a selected image. Any failure terminates execution.

--run SCRIPT	Run a list of <code>scyld-modimg</code> commands.
---------------------	---

--chroot [KVER]

Chroot into the unpacked image to allow for manual modifications. Optionally specify *KVER*, which is the version of a kernel inside the image, which informs a `uname -r` inside the chroot to identify the specific kernel version if/when configuring software needing to link against that kernel. (Otherwise a `uname -r` inside a chroot names the kernel of the host system executing the `scyld-modimg`, not the kernel installed inside the chrooted image.) See **--execute** *COMMAND* and *EXAMPLES*.

--create [DISTRO]

Create a new image from scratch, optionally specifying a non-default distro name *DISTRO*.

--delete	Delete the selected image(s) from the local cache.
--import FILE	Import an existing tar, squashfs, or singularity image.

--capture NODE [--set-name IMAGE]

Capture image from a booted node. If the optional **--set-name** *IMAGE* is not supplied, then the tool prompts the user for an *IMAGE* name to create or overwrite.

--install PKGS	Install packages <i>PKGS</i> into the image.
-----------------------	--

--query [PKGS]

Query package versions from the image (default=ALL).

--update [PKGS]

Update specified packages in the image (default=ALL).

--uninstall PKGS	Uninstall packages from the image.
--unpack TARGZ	Unpack a tar.gz file into the image.
--execute COMMAND	Execute a command in the unpacked image. Note that this <i>COMMAND</i> can include <i>KVER=<kernelVersion></i> , thereby overriding the default behavior of a <code>uname -r</code> executing inside the image. See --chroot <i>KVER</i> and <i>EXAMPLES</i> .

--copyin SRC DEST

Copy files or directories from *SRC* into the image as *DEST*.

--copyout SRC DEST

Copy files or directories *SRC* out of the image to destination *DEST*.

--mount PATH Unpack the image into *PATH* and bind-mount various folders as if preparing for `--chroot`. After the mount the image can be customized by other commands, such as `ansible`, before being repacked.

--unmount PATH Repack the image from a previously mounted *PATH*.

--write-repos [DISTRO]

Write ClusterWare repository file(s) into the image.

EXAMPLES

```
scyld-modimg -i NewImage --query kernel,clusterware-node
```

Display the *kernel* and *clusterware-node* RPM versions installed in the image.

```
scyld-modimg -i NewImage --query
```

Display all RPMs installed in the image.

```
scyld-modimg -i NewImage --chroot
```

Examine and/or modify the contents of the image using `chroot`.

```
scyld-modimg -i NewImage --chroot 3.10.0-1160.45.1.el7.x86_64
```

Explicitly override the `uname -r` output when executed inside the image.

```
scyld-modimg -i NewImage --execute 'KVER=3.10.0-1160.45.1.el7.x86_64 uname -r'
```

Explicitly control the `uname -r` output inside the image when executing commands, e.g., in this case the `uname -r` command.

```
scyld-modimg --capture n8 --set-name CapturedN8image --upload
```

Capture the image executing on node `n8`, give it the name "CapturedN8image", and upload it.

QUIRKS

Note that when exiting a `--chroot`, several directories do not get repacked and saved into the image, including `/tmp/`, `/var/tmp/`, `/var/cache/yum`.

RETURN VALUES

Upon successful completion, **scyld-modimg** returns 0. On failure, any changes are discarded, an error message is printed to `stderr`, and **scyld-modimg** returns 1.

4.26.4 Failing PXE Network Boot

If a compute node fails to join the cluster when booted via PXE network boot, there are several places to look, as discussed below.

Rule out physical problems. Check for disconnected Ethernet cables, malfunctioning network equipment, etc.

Confirm the node's MAC is in the database. Search for the node by MAC address to confirm it is registered with the ICE ClusterWare™ system:

```
scyld-nodectl -i 00:11:22:33:44:55 ls -l
```

Check the system logs. Specifically look for the node's MAC address in the `api_error_log` and `head_*.log` files. These files will contain AUDIT statements whenever a compute node boots, e.g.,

```
Booting node (MAC=08:00:27:f0:44:35) as iscsi using boot config b7412619fe28424ebe1f7c5f3474009d.
```

Booting node (MAC=52:54:00:a6:f3:3c) as rram using boot config f72edc4388964cd9919346dfef21cd2c.

If there are no "Booting node" log statements, then the failure is most likely happening at the DHCP stage, and the head nodes' `isc-dhcpd.log` log files may contain useful information.

As a last resort, check if the head node is seeing the compute node's DHCP requests, or whether another server is answering, using the Linux `tcpdump` utility. The following example shows a correct dialog between compute node 0 (10.10.100.100) and the head node.

```
[root@cluster ~]# tcpdump -i eth1 -c 10
Listening on eth1, link-type EN10MB (Ethernet),
  capture size 96 bytes
18:22:07.901571 IP master.bootpc > 255.255.255.255.bootps:
  BOOTP/DHCP, Request from .0, length: 548
18:22:07.902579 IP .-1.bootps > 255.255.255.255.bootpc:
  BOOTP/DHCP, Reply, length: 430
18:22:09.974536 IP master.bootpc > 255.255.255.255.bootps:
  BOOTP/DHCP, Request from .0, length: 548
18:22:09.974882 IP .-1.bootps > 255.255.255.255.bootpc:
  BOOTP/DHCP, Reply, length: 430
18:22:09.977268 arp who-has .-1 tell 10.10.100.100
18:22:09.977285 arp reply .-1 is-at 00:0c:29:3b:4e:50
18:22:09.977565 IP 10.10.100.100.2070 > .-1.tftp: 32 RRQ
  "booting::loader" octet tsize 0
18:22:09.978299 IP .-1.32772 > 10.10.100.100.2070:
  UDP, length 14
10 packets captured
32 packets received by filter
0 packets dropped by kernel
```

Verify that [SCW-SHORT] services are running. Check the status of *clusterware* services with the commands:

```
systemctl status clusterware
systemctl status clusterware-dhcpd
systemctl status clusterware-dnsmasq
```

Restart *clusterware* services from the command line using:

```
sudo systemctl restart clusterware
```

Check the switch configuration. If the compute nodes fail to boot immediately on power-up but successfully boot later, the problem may lie with the configuration of a managed switch.

Some Ethernet switches delay forwarding packets for approximately one minute after link is established, attempting to verify that no network loop has been created ("spanning tree"). This delay is longer than the PXE boot timeout on some servers.

Disable the spanning tree check on the switch. The parameter is typically named "fast link enable".

4.26.5 Creating Local Repositories without Internet

When `scylld-install` (and its underlying use of the `yum` command) do not have access to repositories that are accessible via the Internet, then repositories must be set up on local storage.

First ensure that the appropriate base distribution repositories (i.e., Red Hat RHEL or CentOS) are also accessible locally without requiring Internet access. An initial install of ICE ClusterWare™ has dependencies on various base

distribution packages, and a subsequent ClusterWare update may have dependencies on new or updated base distribution packages.

Next you need a ClusterWare ISO file that contains the desired software. **The easiest way to obtain an ISO file is to download a pre-built ISO from the ClusterWare online repository.** To do this, log into <https://updates.penguincomputing.com/clusterware/12/> using your ClusterID as the username, leaving the password field blank. Once logged in, select the EL7, EL8 or EL9 folder as desired. Within these folders are "iso" folders, where pre-built ISOs can be downloaded.

Alternatively, it's possible to build the ISO on a local server that has access to the Internet. To build the ISO locally, you need a `clusterware.repo` file that contains a valid customer authentication token that allows access to Penguin Computing's ClusterWare yum repo, then:

```
# Download the ClusterWare `make-iso` script:
curl -O https://updates.penguincomputing.com/clusterware/12/installer/make-iso

# Execute the `make-iso` script to create either an ISO named "clusterware.iso":
./make-iso --yum-repo ./clusterware.repo
# Or to create an arbitrarily named ISO:
sudo ./make-iso --yum-repo ./clusterware.repo clusterware-12.1.0.iso

# Note: `./make-iso --from-yum` is equivalent to
#       `./make-iso --yum-repo /etc/yum.repos.d/clusterware.repo`
```

Once an ISO file is obtained, whether via download or the `make-iso` command, the ISO file needs to be mounted. Suppose the ISO file `clusterware-12.1.0.iso` contains ClusterWare release 12.1.0:

```
# Mount the ClusterWare ISO, if not already mounted:
sudo mount -o loop clusterware-12.1.0.iso /mnt/cw12.1.0
```

For an initial install, use a cluster configuration file (e.g., named `cluster-conf`) that is described in *Install ICE ClusterWare*, and execute the `scyld-install` script that is embedded in the ISO to perform the basic first install of the ClusterWare platform and create `/etc/yum.repos.d/clusterware.repo`, which points at the software in the ISO:

```
/mnt/cw12.1.0/scyld-install --config cluster-conf
```

Once the head node software has been installed, then subsequent ClusterWare commands need to find a base distribution defined repo and distro. See *Creating Arbitrary Rocky Images* (or *Creating Arbitrary RHEL Images*) for examples.

Suppose the base distribution ISO is accessible at `http://<baseOSserver>/<baseOSiso>`:

```
scyld-clusterctl repos create name=<baseOSrepo> iso=@</path/to/baseOSiso>
scyld-clusterctl distros create name=<baseOSdistro> repos=<baseOSrepo>
```

Now finish the setup. The following expects to find a single *distro* and one or more *repo* repositories:

```
scyld-add-boot-config --make-defaults
```

For a software update of an existing install, rename the current `/etc/yum.repos.d/clusterware.repo`, then execute the script (which recreates `clusterware.repo` with the appropriate values):

```
(cd /etc/yum.repos.d; sudo mv -f clusterware.repo clusterware.repo.bak)
/mnt/cw12.1.0/scyld-install
```

Important

If the local repo has been created in a manner other than what is described above, then it is possible that `/etc/yum.repos.d/clusterware.repo` uses `baseurl` of the form `file:///` (e.g., `baseurl=file:///var/www/html/cw12.1.0`). This may cause future problems when attempting to create an image, so the administrator should edit this to a functionally equivalent form `http://` (e.g., `baseurl=http://localhost/cw12.1.0`).

4.26.6 Validating ClusterWare ISOs

To validate a downloaded ICE ClusterWare™ ISO file, first import the gpg key that was used to sign the RPMs and ISOs:

```
curl -sSL https://updates.penguincomputing.com/RPM-GPG-KEY-scyld-clusterware | gpg --
↳import -
```

Then download the CHECKSUM.asc file from the repo, e.g.:

```
wget https://<AUTHENTICATION_TOKEN>@updates.penguincomputing.com/clusterware/12/el8/iso/
↳CHECKSUM.asc
```

and verify the CHECKSUM.asc file:

```
[admin@head]$ gpg --verify CHECKSUM.asc
gpg: Signature made Thu 05 Jan 2023 07:01:37 PM PST using DSA key ID 0A1E1108
gpg: Good signature from "Penguin Computing <support@penguincomputing.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: AEFA 2C55 EB4A 88EF BE71 022B 0722 4B0A 0A1E 1108
```

Confirm that the downloaded ISO is named in CHECKSUM.asc. For example, for `clusterware-11.9.2-g00000.el8.x86_64.iso`:

```
grep clusterware-11.9.2-g00000.el8.x86_64.iso CHECKSUM.asc
```

should find the ISO. Now compare the checksum of the ISO with the ISO named in CHECKSUM.asc:

```
diff <(sha256sum clusterware-11.9.2-g00000.el8.x86_64.iso) \
    <(grep      clusterware-11.9.2-g00000.el8.x86_64.iso CHECKSUM.asc)
```

and expect to see no differences.

4.26.6.1 make-iso

NAME

make-iso -- Create an ISO file from a yum repo.

USAGE**make-iso**

```
[-h] <RPM-SOURCE> [ISOFILE]
```

DESCRIPTION

This is a low-level tool that creates an ISO file, optionally named *ISOFILE*, from a yum repo file or from collection of RPMs from *RPM-SOURCE*.

The tool resides in `/opt/scyld/clusterware-installer/make-iso`.

<RPM-SOURCE> OPTIONS

- from-yum** Mirror RPMs from the baseurl(s) in `/etc/yum.repos.d/clusterware.repo`.
- rpm-dir DIR** Copy the RPMs from the directory *DIR*.
- yum-repo REPOFILE** Parse a specific repo file *REPOFILE* for RPM sources.

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.

EXAMPLES

(Note that `make-iso` resides in `/opt/scyld/clusterware/installer/`)

```
make-iso --yum-repo /tmp/clusterware.repo
```

Use the RPMs identified by the yum repo file `/tmp/clusterware.repo` to create an ISO named `clusterware.iso`.

```
make-iso --from-yum
```

Equivalent to `make-iso --yum-repo /etc/yum.repos.d/clusterware.repo`.

```
make-iso --yum-repo /tmp/clusterware.repo cw12.1.iso
```

Use the RPMs identified by the yum repo file `/tmp/clusterware.repo` to create an ISO named `cw12.1.iso`.

```
make-iso --rpm-dir /mnt/clusterware/12.0/e17 cw12.0.iso
```

Use the RPMs found in `/mnt/clusterware/12.0/e17/` to create an ISO named `cw12.0.iso`.

RETURN VALUES

Upon successful completion, **make-iso** returns 0. On failure, an error message is printed to `stderr` and **make-iso** returns 1.

4.27 Image Sources Page

The Image Sources page shows the defined repositories and distributions. The page is available via **Provisioning + SW > Image Sources** in the left navigation panel.

Image Sources

Refresh Interval (seconds): 10

Software Distribution objects are used to collect information about a software distribution and are comprised of one or more software repositories. Software Repository objects are used to collect information about an upstream or local repository for software packages.

Repos

Name ▲	URLs ▲
scyldiso	<BASE_URL>/isomount/scyldiso/ ...
Rocky_base	http://dl.rockylinux.org/pub/rocky/\$releasever/BaseOS/\$basearch/os/ ...
Rocky_appstream	http://dl.rockylinux.org/pub/rocky/\$releasever/AppStream/\$basearch/os/ ...
Rocky_iso	<BASE_URL>/isomount/Rocky_iso/BaseOS/<BASE_URL>/isomount/Rocky_iso/AppStream/ ...
CentOS-7-x86_64-Everything-2...	<BASE_URL>/isomount/CentOS-7-x86_64-Everything-2009/ ...
Rocky-9.3-x86_64	<BASE_URL>/isomount/Rocky-9.3-x86_64/BaseOS/<BASE_URL>/isomount/Rocky-9.3-x86_64/AppStream/ ...

Distros

Name ▲	Repo ▲	Description ▲
Rocky	Rocky_base . Rocky_appstream	...
Rocky_iso	Rocky_iso	...
CentOS-7-x86_64-Everything-2...	CentOS-7-x86_64-Everything-2009	...
Rocky-9.3-x86_64	Rocky-9.3-x86_64	...

ADD REPO

ADD DISTRO

4.27.1 Repos

The **Repos** panel shows existing repos and allows you to create or modify repos.

Repos

Name ▲	URLs ▲
scyldiso	<BASE_URL>/isomount/scyldiso/ ...
Rocky_base	http://dl.rockylinux.org/pub/rocky/\$releasever/BaseOS/\$basearch/os/ ...
Rocky_appstream	http://dl.rockylinux.org/pub/rocky/\$releasever/AppStream/\$basearch/os/ ...
Rocky_iso	<BASE_URL>/isomount/Rocky_iso/BaseOS/<BASE_URL>/isomount/Rocky_iso/AppStream/ ...
CentOS-7-x86_64-Everything-2...	<BASE_URL>/isomount/CentOS-7-x86_64-Everything-2009/ ...
Rocky-9.3-x86_64	<BASE_URL>/isomount/Rocky-9.3-x86_64/BaseOS/<BASE_URL>/isomount/Rocky-9.3-x86_64/AppStream/ ...

4.27.1.1 Create a Repo

To create a Repo:

1. Click **Add Repo**.
2. Add details about the repo.
 - **Name:** Required.
 - **Description:** Optional.
 - **Full Name:** Optional.
 - **Keys:** Optional.
 - **URLs:** Required.
 - **ISO File:** Optional.
3. Click **Add Repo** to save your changes.

The new repo appears in the list at the top of the panel.

4.27.1.2 Edit a Repo

To edit a repo:

1. Click the repo name to open the **Edit Repo** panel.
2. Make updates to the repo.
3. Click **Edit Repo** to save your changes.

4.27.1.3 Delete a Repo

To delete a repo, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.27.2 Distros

A distro is a collection of repositories. The **Distros** panel shows existing distros and allows you to create or modify distros.

Distros

Name ▲	Repo ▲	Description ▲
Rocky	Rocky_base . Rocky_appstream	...
Rocky_iso	Rocky_iso	...
CentOS-7-x86_64-Everything-2...	CentOS-7-x86_64-Everything-2009	...
Rocky-9.3-x86_64	Rocky-9.3-x86_64	...

4.27.2.1 Create a Distro

To create a distro:

1. Click **Add Distro**.
2. Add details about the distro.

- **Name:** Required.
- **Description:** Optional.
- **Repos:** Optional.
- **Packaging:** Optional.
- **Release:** Optional.

3. Click **Update** to save your changes.

The new distro appears in the list at the top of the page.

4.27.2.2 Edit a Distro

To edit a distro:

1. Click the distro name to open the **Edit Distro** panel.
2. Make updates to the distro.
3. Click **Edit Distro** to save your changes.

4.27.2.3 Delete a Distro

To delete a distro, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.27.3 Related Links

- *Repos and Distros*
- *scylld-modimg*

4.28 Git Repositories

ICE ClusterWare™ head nodes can host Git repositories containing code or configuration files needed by compute nodes. These Git repositories can be used to enable compute nodes to run Ansible during boot up or to pull down version-controlled config files. The ClusterWare platform can directly host the Git repository on the head node, storing the files locally and allowing for modifications by local developers. Alternatively, the ClusterWare head nodes can be leveraged as a synchronization point for tracking upstream repositories. In the latter case, development work is primarily done on the upstream Git repository and ClusterWare tools can be used to sync or pull any changes down to the local copy. In both cases, the Git repositories are exported into the cluster for use with Ansible, Puppet, or as repositories of configuration files for initialization and boot-up.

4.28.1 Initial Preparation

To facilitate interaction with the ClusterWare Git system, first add your personal public key to your ClusterWare admin account. This key is populated into the root user's (or `_remote_user`'s) `authorized_keys` file for a newly booted compute node. The key is then used to provide SSH access to the Git repository.

1. To add the key to the admin record, run the following command:

```
scylld-adminctl up keys=@/full/path/.ssh/id_rsa.pub
```

2. Add the localhost's host keys to a personal `known_hosts` file with the following command:

Note

Adding the localhost's host key is not strictly necessary, but adding it avoids an SSH warning that can interrupt scripting.

```
ssh-keyscan localhost >> ~/.ssh/known_hosts
```

4.28.2 Locally Hosted Repositories

The *scyl-d-clusterctl* tool is used to create, update, and delete *gitrepos* objects in the ClusterWare database.

To create a new *gitrepos* object:

```
scyl-d-clusterctl gitrepos create name=gtest
```

To view the details of the *gitrepos* object:

```
scyl-d-clusterctl gitrepos ls -l
Git Repos
gtest
  archive
  content
    checksum: sha1:b0b06c1d5acbcfc1e5a38314f31af6f3d8b4b63b
    filename: 50a2cc42303f4bf98c71c09b9e0d2adf
    mtime: 2024-08-02 18:41:43 UTC (0:42:47 ago)
    size: 12.0 KiB (12288 bytes)
  last_modified: 2024-08-02 18:41:43 UTC (0:42:47 ago)
  name: gtest
  sshgit: cwgit@<HEAD>:gtest
  url: <BASE_URL>/git-http/gtest
```

The main field of interest is *sshgit*, which can be used in Git operations. For example, to clone a copy of the *gitrepos* object:

```
git clone cwgit@192.168.122.88:gtest
Cloning into 'gtest'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
```

By default, a single file is added in the otherwise empty repository: *CwReadMe.txt*. At this point, the repository on the local disk can be used like any other Git repository. For example, to create a new file in the directory:

```
cd gtest/
ls
CwReadMe.txt
echo "new file" > NewFile.txt
git add NewFile.txt
git commit -m "add new file"
[main 89d5f44] add new file
```

(continues on next page)

(continued from previous page)

```
1 file changed, 1 insertion(+)
create mode 100644 NewFile.txt
```

Since authentication occurs via SSH keys, you can push the changes back to the ClusterWare-hosted Git repos:

```
git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 286 bytes | 286.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To 192.168.122.88:gtest
 d5a1a0b..89d5f44 main -> main
```

The “url” includes an HTTP-based URL that can also be used for Git download operations, but does NOT allow new data to be pushed up to the ClusterWare head node.

4.28.3 Mirroring Upstream Resources

In many cases, ClusterWare Git repositories are used to mirror upstream repos. Developers can work with standard tools and interact with an enterprise Git service, such as GitLab or GitHub, without needing direct contact to the cluster. The ClusterWare platform can synchronize the locally-hosted copy with the upstream repository.

When creating the gitrepos object, specify a name, an upstream source, and a mapping for local-to-upstream branches:

```
scyld-clusterctl gitrepos create name=gtest2 \
  upstream=http://192.168.122.88/api/v1/git-http/gtest \
  branch_map=main:main
```

In this example, only one branch is defined and it shares the same name (main) in the local and upstream repos. For more complex configurations, a comma-separated list of local-to-upstream mappings can be provided, and the local/upstream names do not need to be the same. For example: “main:main,cw_dev:up_dev” defines two mappings. The first maps the local “main” to upstream “main” and the second maps local “cw_dev” to the upstream “up_dev”.

Since the ClusterWare platform creates a minimal Git repository by default (with a CwReadMe.txt file), it is often necessary to “sync” and “sync reset” so that the local repos downloads the relevant metadata from the upstream repos, and then pulls down the current upstream contents. For example:

```
scyld-clusterctl gitrepos -igtest2 sync
Git Repos
 gtest2
 Branch 'main' is 1 commit ahead of and 3 commits behind branch 'upstream/main'

scyld-clusterctl gitrepos -igtest2 sync reset
Git Repos
 gtest2
 Branch 'main' reset to branch 'upstream/main'
```

Note

This extra “reset” is caused by the placeholder content that ClusterWare v.12.4.0 creates. This may change in future releases.

When new upstream changes are made, pull the changes to the local copy using the following command:

```
scyld-clusterctl gitrepos -igtest2 sync pull
Git Repos
gtest2
Branch 'main' reset to branch 'upstream/main'
```

Next, re-sync any clients that might need that new data (for example, all of the compute nodes):

```
git pull
```

4.28.4 Public Access

For development and testing of the config files and code in a Git repository, authenticated access is needed so that the `git push` commands work. The ClusterWare platform can also provide public access to any `gitrepos` hosted on the head nodes.

The base URL for public access is found in the `scyld-clusterctl` output:

```
scyld-clusterctl gitrepos ls --fields url -l
Git Repos
gtest
url: <BASE_URL>/git-http/gtest
```

For the above example, public access can be found at:

```
git clone http://192.168.122.88/api/v1/git-http/gtest
Cloning into 'gtest'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
```

In this example, if another file is added and pushed back to the ClusterWare `gitrepos` object, the push will fail. Even if SSH keys have been loaded, this is using a public URL, which disallows upload of new content:

```
echo "another file" > AnotherFile.txt
git add AnotherFile.txt
git commit -m "adding another file"
[main b900bf1] adding another file
1 file changed, 1 insertion(+)
create mode 100644 AnotherFile.txt
$ git push
fatal: unable to access 'http://192.168.122.88/api/v1/git-http/gtest/': The requested
↪URL returned error: 403
```

If public access to a `gitrepos` needs to be disabled for some reason, setting the “public” field to `False` blocks access:

```
scyld-clusterctl gitrepos -igtest update public=False
```

4.29 Git Repos Page

The **Git Repos** page lists each locally hosted Git repository with a clickable link to details, whether it is public or not, the URL to the bare repo (and contents), and the argument used to clone the repository via `ssh`. The page is available via **Provisioning + SW > Git Repos** in the left navigation panel.

Cluster head nodes can host public git repositories for use with the `_ansible_pull` and `_ansible_pull_now` node attributes. All repositories can be accessed via SSH and public repositories can be accessed over HTTP.

Name	Description	Public	URL	Cloning
cluster-settings		<input checked="" type="checkbox"/>	<BASE_URL>/git-http/cluster-settings (browse)	<code>cwgit@<HEAD>:cluster-settings</code> ...
admin-tools		<input checked="" type="checkbox"/>	<BASE_URL>/git-http/admin-tools (browse)	<code>cwgit@<HEAD>:admin-tools</code> ...

[ADD GIT REPO](#)

Git repositories hosted by ICE ClusterWare™ head nodes are useful for automated configuration systems. For example, a Git repository hosted by a head node could contain an Ansible playbook. For an example implementation, see *Using Ansible*.

4.29.1 Create Git Repo

To create a Git repository:

1. Click **Add Git Repo**.
2. Add details about the Git repository. The following fields are available:
 - **Name:** Required.
 - **Description:** Optional.
 - **Public:** Indicate whether the Git repository is public or private. Set to Public by default.
 - **Upstream:** Optional. Provide the URL for the Git repository to define a mirror of the upstream.
3. Click **Add Git Repo** to save your changes.

The new Git repository appears in the list at the top of the page.

4.29.2 Edit Git Repo

To edit a Git repository:

1. Click the Git repository name to open the **Edit Git Repo** panel.
2. Make updates to the Git repository.
3. Click **Edit Git Repo** to save your changes.

4.29.3 Delete Git Repo

To delete a Git repository, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.29.4 Related Links

- *Git Repositories*
- *scylld-clusterctl*
- *Using Ansible*

4.30 State Maps

A common task for a cluster administrator is identifying specific nodes that are out of compliance in some way and executing actions to solve such issues. These actions often involve temporarily removing the node(s) from production while performing testing, reprovisioning, and requalification. As problem nodes are identified, new nodes are added, or nodes are transferred from one configuration to another, the cluster administrator must have some means to keep track of the progress of each node through these processes. After all, these processes involve multiple stages, likely spanning multiple reboots or even reimaging. ICE ClusterWare™ node attributes can be leveraged for persistently storing this progress information.

For example, when a node health check detects a memory issue on a GPU, other tasks may dictate that power cycling the node for row remapping cannot occur immediately. Instead, the health checking code could set an attribute noting what was detected. Then, a separate process could see that attribute and initiate the steps of removing the node from production, rebooting it, triggering requalification tests, and moving it back into production if all goes well.

Of course, this simple detection and mitigation process only covers one type of failure and one possible resolution. The GPU or other hardware could fail in a myriad of ways, each requiring different mitigation strategies. This means that a node's health check results or progress through requalification may be stored across multiple node attributes.

The ClusterWare node selection language allows a cluster administrator to identify nodes that match possibly complex criteria by matching attribute values, detected status, or hardware details using basic comparators and logical operators. See the above section *Attribute Groups and Dynamic Groups* for dynamic groups for examples of node selectors.

Polling the ClusterWare service for attribute status frequently or across many nodes is inefficient and in extreme cases can impact the head node performance. To alleviate this, the ClusterWare platform provides a `scylld-nodectl waitfor` mechanism. One common use is to wait for a node to boot before proceeding with additional steps in an overall command, for example:

```
scylld-nodectl -in10 reboot then waitfor up then exec uptime
```

The `up` is shorthand for a longer selector, specifically `status[state] == "up"` and can be replaced with more complicated selectors if, for example, the administrator is not rebooting the node but executing a command that will modify a node attribute when it completes. This sort of command chaining with `"then"` allows for simple automation, but more complex automation will deal with multiple nodes at different stages. For that case, the ClusterWare platform allows administrators to provide a set of selectors referred to as a state map.

Using a state map, a cluster administrator can track nodes through scenarios including: * Error detection, handling, and requalification * A rolling firmware update process * Idle-time performance testing

State maps provide a general purpose mechanism to select groups of nodes based on their status and configuration, trigger actions, and observe the resulting changes. An example state map is provided as part of the `clusterware-tools` package:

```
$ cat /opt/scylld/clusterware-tools/examples/node-states.ini
[status]
up = status[state] == "up"
down = status[state] == "down"
booting = status[state] == "booting"
```

This INI format defines a state map named “status” containing 3 states named “up”, “down”, and “booting”. The selector that defines each state is provided to the right of the name, after the equal sign. This file can also be written in JSON format as:

```
{
  "name": "status",
  "states": {
    "up": "status[state] == \"up\"",
    "down": "status[state] == \"down\"",
    "booting": "status[state] == \"booting\""
  }
}
```

The cluster administrator can load the state map through the `scyld-nodectl` command:

```
scyld-nodectl waitfor --load-only @node-status.json
```

Once the state map is loaded, the `waitfor` command can also be used to see what nodes match what selectors by referencing the loaded map name, i.e. “status” in this example:

```
$ scyld-nodectl waitfor --name status
Nodes
  n[5-8,10]: up
```

Additional arguments are available to allow for streaming state transitions or simplifying the output for easier parsing:

```
$ scyld-nodectl waitfor --stream --name status
n[1-4] in down
n[5-10] in up

n[5] left up
n[5] entered down
n[5] left down
n[5] entered booting
n[5] left booting
n[5] entered up
```

In the above example, a single node in a 10 node cluster was rebooted and state transitions were emitted as the node progressed from “up” to “down” to “booting” and back to “up”.

4.31 Grafana Telemetry Dashboard

4.31.1 Introduction to Grafana and InfluxDB

InfluxDB is a database that is optimized for time-series data and analytics.

Grafana is a powerful and flexible dashboard and visualization tool from Grafana Labs (<https://grafana.com/>). It is available as Open-Source software, but Grafana Labs does offer commercial support as well.

4.31.1.1 InfluxDB

While InfluxDB is technically a separate tool, the Telegraf data collection tool is created by the same company and provides optimized "plugins" which can push a variety of metrics into InfluxDB. Both tools are Open-Source but commercial support is available through InfluxData Inc. (<https://www.influxdata.com/>). There is a vibrant community of

users and plugin developers, and the official community forum at <https://community.influxdata.com/> often has answers directly from the InfluxData team.

The InfluxDB database can currently be queried through the Flux language, a powerful data scripting and processing language. Documentation for Flux can be found at <https://docs.influxdata.com/flux/v0/>. As a simple example, a query to find the CPU usage data for all nodes in the cluster is shown below:

```
from(bucket: "telegraf")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "cpu")
  |> filter(fn: (r) => r["_field"] == "usage_system")
```

This example illustrates a typical approach to querying and filtering data in InfluxDB/Flux. First, a stream of records is pulled from a "bucket" (similar to a table in relational databases). That stream may include multiple metrics stored in multiple measurements and fields. A given Telegraf data collection cycle, for example, may include CPU and memory measurements, and each of those may have multiple fields of information: the CPU measurement may have per-CPU-core data, the memory measurement may have total, used, and free memory values. Each of those measurement-field combinations will be a separate record in the stream.

It is often useful to reduce the time-range early in the query, otherwise all records from all time will be processed by the stages of the query. The "earlier" the stream's data is reduced, the greater a speed-up will be seen in the query process. In this example, the range is reduced to whatever has been selected by the GUI, with the "v." notation being used to reference one of the dashboard-shared "variables". Since a measurement can include multiple fields, it is useful to filter on measurement next since it will often provide another large reduction in the amount of stream data. Additional filtering on specific field information can be used to further refine the data.

Data Retention

By default, the ICE ClusterWare™ platform retains 7 days of InfluxDB data. You can adjust the retention time and frequency of downsampling. A primary concern in adjusting these intervals is available disk space.

To update the InfluxDB data retention:

1. Obtain the bucket-id by running:

```
influx bucket list
```

2. Update the InfluxDB bucket retention by running:

```
influx bucket update -i <bucket-id> -r <retention period with units>
```

To adjust downsampling, you need both a source and destination bucket. Then you create a task that runs a script to downsample data after a certain period of time. See the following InfluxData documentation for details:

- Creating a new bucket: <https://docs.influxdata.com/influxdb/v2/admin/buckets/create-bucket/>
- Creating a task: <https://docs.influxdata.com/influxdb/v2/process-data/manage-tasks/create-task/>
- Downsampling data requirements and example: <https://docs.influxdata.com/influxdb/v2/process-data/common-tasks/downsample-data/>

If you want to visualize your downsampled data with Grafana, reference the appropriate bucket in your queries.

Learn More

The official documentation from InfluxData is very good, and includes examples and videos.

- Getting started: <https://docs.influxdata.com/flux/v0/get-started/>

- InfluxDB University includes training materials and videos: <https://www.influxdata.com/university/> and <https://university.influxdata.com/>
 - The "Data Querying" section includes Basic, Beginner, and Intermediate courses in the Flux language

It should be noted that in August 2023, InfluxData announced plans to de-prioritize their future investments in Flux and to put the project into "maintenance mode". They explicitly state that Flux is not going End-Of-Life, and that they do anticipate supporting customers for some time to come. InfluxData also offers an SQL-like language, InfluxQL, and has plans for full SQL in version 3 of the InfluxDB database, so there will be a path forward regardless.

- Future of Flux: <https://docs.influxdata.com/flux/v0/future-of-flux/>
- SQL-like alternatives: <https://docs.influxdata.com/influxdb/v2/query-data/influxql/> and <https://docs.influxdata.com/influxdb/clustered/query-data/>

The ClusterWare platform is fully committed to providing a robust monitoring and alerting platform. As InfluxData's roadmap becomes more firm, the ClusterWare platform will update its roadmap accordingly.

4.31.1.2 Grafana

Besides drawing charts and graphs, Grafana can provide alerting capabilities and, through the Loki plugin, may be used for log aggregation and analysis.

At a high level, a Grafana dashboard is a set of "panels" where each panel is a query against a "data source" along with a visualization for the resulting data. When the ClusterWare platform is installed, the *InfluxDB* data-source is configured, which connects to the InfluxDB/Telegraf data; it also comes with a cluster-wide and single-node dashboards. Either of those can be copied and then customized to meet any local needs.

When adding panels to a dashboard, note that it will show up as a blank panel of some default size. The corners of the panel can easily be clicked and dragged to be any size, but to change the query or visualization type requires editing the panel through the menu on the top-bar of the panel (look for the small triangle symbol). Keep in mind that since the query will, by default, be empty, there may be no data for the panel to display. As such, it may be helpful to start with the "table" visualization while working on the query since this will give more direct feedback even if the query itself is in error or if it's producing different kind of data than expected. Once the query is refined, then a proper visualization type can be selected for that data.

Note that since the ClusterWare installation configures the *InfluxDB* data-source to use the Flux language, all queries in Grafana must also use the Flux language.

Extensive GrafanaLabs documentation is available on-line:

- General overview of dashboards: <https://grafana.com/docs/grafana/latest/dashboards/>
- Panels and visualizations: <https://grafana.com/docs/grafana/latest/panels-visualizations/>
 - Grafana includes scatter, line, bar, and pie charts, tables, gauges, tables, and more.
 - All of the visualization types can be customized in terms of color, font, etc. And many of them have thresholding features to highlight only data above (or below) some threshold.
- Panels and dashboards can be made interactive through the use of "variables" that are shared across a dashboard, and through "data links" which enable hyperlinks from data values to other dashboards or panels.
 - Dashboard variables: <https://grafana.com/docs/grafana/latest/dashboards/variables/>
 - Panel data-links: <https://grafana.com/docs/grafana/latest/panels-visualizations/configure-data-links/>
 - * For example, clicking on a node's name in the ClusterWare Cluster Overview dashboard will link to the node-specific dashboard.
- Alerting: <https://grafana.com/docs/grafana/latest/alerting/>

- Alerts are essentially a query (Flux language) that is run periodically and if any records emitted by that query are above a threshold, then an alert is generated. The output can be as simple as an email sent to one or more recipients, or a connection to one or more external "contact points" such as PagerDuty, Sensu, Slack, etc.
- The main tutorials, including videos and quick-start guides: <https://grafana.com/tutorials/>
- Grafana Labs has a set of "Grafana for Beginners" videos at YouTube: <https://www.youtube.com/playlist?list=PLDGkOdUX1Ujo27m6qiTPPCpFHVfyKq9jT>
 - The whole series is a broad overview of observability and monitoring, the dashboard and visualization information in Episodes 8 and 9 may be of particular interest.

4.31.2 Grafana Setup Script

The `influx_grafana_setup` script is used during the install or upgrade process to set up, update, and manage the ICE ClusterWare™ monitoring tools. When the script is run, InfluxDB, Telegraf, and Grafana are installed or updated.

The script includes steps for:

- Creating the InfluxDB bucket and API token
- Installing and setting up Telegraf to write to InfluxDB
- Installing Grafana
- Setting up the Grafana admin password and API key
- Connecting the InfluxDB datasource

Running this script outside of the install or upgrade process is not common, but may be needed to clear the InfluxDB database contents or reset a password back to a known state.

The script is located at `/opt/scyld/clusterware/bin/influx_grafana_setup`

4.31.2.1 Arguments

- `--purge`: Provides a complete monitoring reset by clearing all telegraf data from InfluxDB and resetting Grafana configuration (password included) to the ClusterWare default.
- `--reset-grafana`: Rebuilds the Grafana sqlite database to base ClusterWare configuration. This includes resetting the admin password, reconnecting the InfluxDB datasource, and clearing any custom configurations including Alerts, Dashboards, API keys, and AdHoc users.
- `--tele-env`: Adds the InfluxDB API token as an environment variable. The environment variable is stored in the `/etc/default/telegraf` directory.
- `--tele-input`: This is a legacy/backwards compatible option to insert the InfluxDB API token directly into the telegraf.conf file. Requires both `--tele-input` and `--tele-output`.
- `--tele-output`: This is a legacy/backwards compatible option to insert the InfluxDB API token directly into the telegraf.conf file. Requires both `--tele-input` and `--tele-output`.

4.31.2.2 Example

Clear existing telemetry data and Grafana configurations and reset to a fresh install configuration:

```
/opt/scyld/clusterware/bin/influx_grafana_setup --purge
```

Clear Grafana configurations and return to a fresh install configuration while maintaining telemetry data:

```
/opt/scyld/clusterware/bin/influx_grafana_setup --reset-grafana
```

4.31.3 Grafana Login

The ICE ClusterWare™ Monitoring graphical interface employs the Open Source Grafana, InfluxDB, and Telegraf to collect data from compute nodes and head nodes and present the data visually to authorized users. The basic initialization directs InfluxDB to retain data for one week. The retention period can be modified:

```
TELEGRAF_BUCKET_ID=$(sudo influx bucket list | grep telegraf | awk '{print $1}')
sudo influx bucket update --id ${TELEGRAF_BUCKET_ID} --retention <new-period>
```

where *<new-period>* is an integer concatenated with a one-letter abbreviation of a time period, e.g., "7d" or "1w" for one week, "14w" for 14 weeks, "12h" for 12 hours, "1y" for one year. The longer the retention period means the greater the size of retained data. See <https://docs.influxdata.com/influxdb/v2.6/reference/internals/data-retention/> for details.

Access the Monitoring GUI through the **Health + Monitoring > Telemetry Dashboard** link in the ClusterWare left navigation panel or directly using `http://<HEADNODE_IP>/grafana`.

Note

The URL `http://<HEADNODE_IP>/grafana` may differ if the cluster administrator has switched to HTTPS or otherwise modified the Apache configuration.

When the home page is loaded for the first time, login with username "admin" and the *database.admin_pass* from the `base.ini` (`sudo grep pass /opt/scyld/clusterware/conf/base.ini`). After that, you can change the user name and/or the password as you wish by clicking on the colored icon in the lower left above the "?" question mark to expose a menu allowing you to view or change "Preferences", "Change Password", or "Sign out".

Typically after the initial "admin" *database.admin_pass* login the user should first edit the Preferences to change the user's Name, Email address, and the Username to use for subsequent logins. Then click on "Change Password" and change the password you wish to use for those subsequent logins.

A basic Grafana Monitoring capability is installed preconfigured in the ClusterWare software. You can further modify this configuration to suit your local cluster needs. New dashboards can be created, or new display panels added to the existing ones to show more customized information. Grafana includes a suite of visualization tools like scatter, line, bar, and pie charts, as well as tables, gauges, and histograms. Since the underlying ClusterWare monitoring database is InfluxDB, any valid Flux-language query can be used to filter or process the data. For more information on InfluxDB and Grafana customization, including links to tutorials, see *InfluxDB* and *Grafana*.

To facilitate monitoring of compute node GPU activity, first install into the GPU compute node image(s) the NVidia System Management Interface utility (`nvidia-smi`), which ships with NVidia GPU drivers. See <https://developer.nvidia.com/nvidia-system-management-interface> for details of that utility, and see <https://www.cyberciti.biz/faq/how-to-install-nvidia-driver-on-centos-7-linux/> for a description of how to install NVidia drivers. Then in the compute node image(s) copy `/etc/telegraf/telegraf.d/nvidia-smi.conf.example` (distributed in the *clusterware-node* RPM) to `/etc/telegraf/telegraf.d/nvidia-smi.conf`.

4.31.4 Grafana Cluster Monitoring

The **ClusterWare - Cluster Monitoring** dashboard displays a summary of current activity on the head node and all compute nodes and is shown upon initial login.

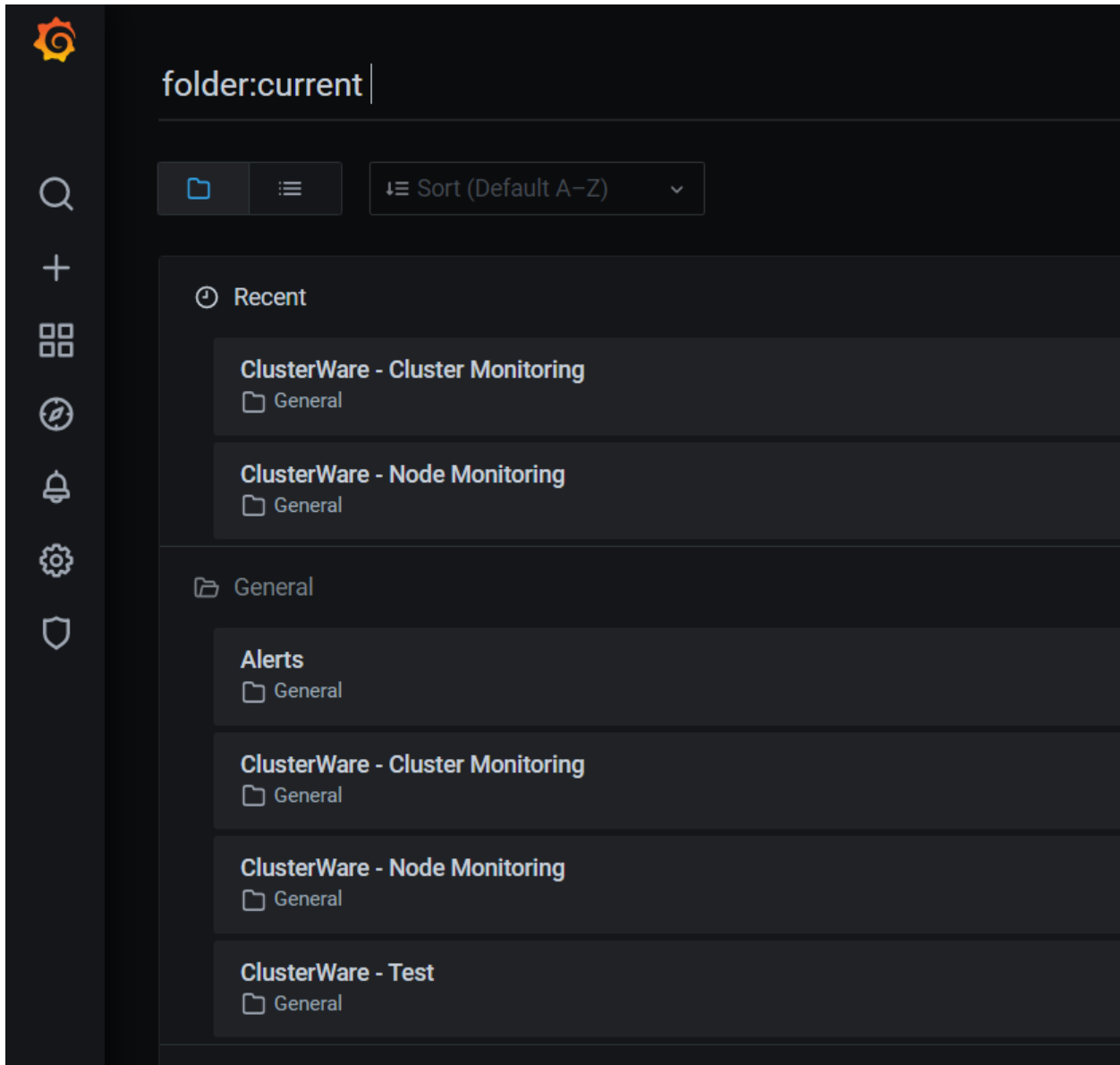
The following example shows the head node and first several nodes of a 49-node CPU-only cluster.



For clusters that include GPUs, the dashboard includes an additional column displaying GPU usage.

4.31.4.1 Grafana General Page

Click **General / ClusterWare - Cluster Monitoring** at the top of the page to display a list of the available dashboards.



The menu lists the **Recent** dashboards as well as the full **General** list. The list of dashboards varies based on your cluster hardware. Click **ClusterWare - Node Monitoring** to display detailed state and activity data for individual nodes. If your cluster includes NVidia GPUs, click **NVidia GPU Dashboard** to display details about GPU temperature, power draw, CUDA version, clock speeds, and so on.

4.31.4.2 Grafana Node Monitoring

The default **Node Monitoring** display shows details for individual nodes, beginning with the head node.



Click the drop-down list with the current node name at the top left of the dashboard to select a different node in the cluster.

For example, select "n02.cluster.local":



4.31.4.3 Grafana Alerts

You can define an **Alerts** dashboard with configurable panels and alert notifications.

1. Click the **Alerting** menu item (bell icon) in the left navigation panel.
2. On the **Alert Rules** tab, click **New alert rule**.

- Define the conditions or events about which you want to receive alerts as well as how those alerts should be delivered to you.

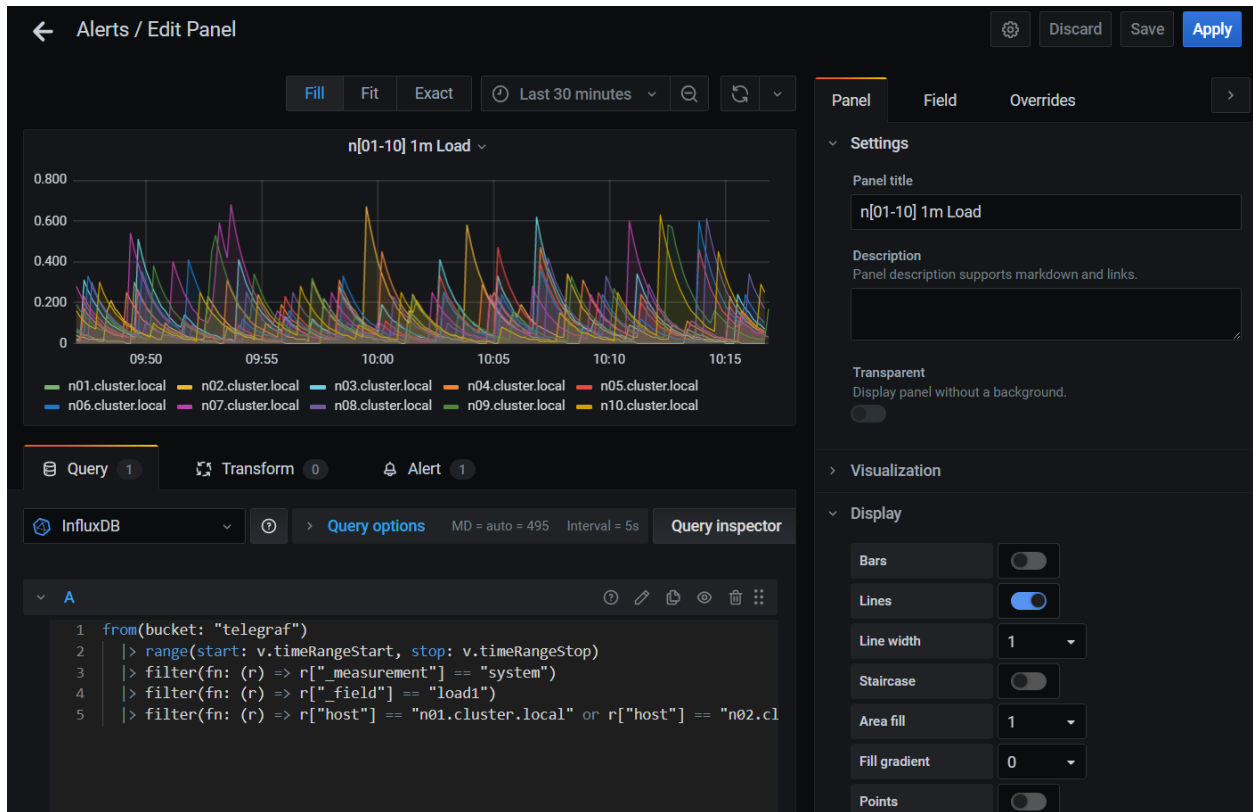
Consult the GrafanaLabs documentation for additional details.

An example Alerts dashboard is:



The first panel displays the CPU load levels for the first 10 compute nodes. The second panel displays the disk usage for one head node.

Alerts can be edited by clicking the title bar and selecting **Edit** from the drop-down menu. In the example below, the **Query** tab defines what gets shown in the panel. The **Alert** tab defines what values trigger an alert, what to send in an alert message, and where to send the message.



See the GrafanaLabs documentation for details about setting up alerts.

4.32 Workload Management

4.32.1 Monitoring Scheduler Info

Important

This software is a TECHNOLOGY PREVIEW that is being rolled out with limited features and limited support. Customers are encouraged to contact Penguin with suggestions for improvements, or ways that the tool could be adapted to work better in their environments.

The ICE ClusterWare™ platform provides a separate daemon process that reads data from one or more inputs and pushes that data into one or more endpoints. The supported inputs are the Slurm batch scheduler and ClusterWare itself. The supported outputs are ClusterWare, InfluxDB, or an archive file. By selecting appropriate inputs and outputs, one could read data from the ClusterWare software and write it into InfluxDB, or read from Slurm and write into the ClusterWare software.

The scheduler data will show up in the ClusterWare node attributes and can then be viewed with:

```
$ scyld-nodectl ls -l
Nodes
  n0
    attributes
      _boot_config: DefaultBoot
      _sched_state: idle
```

(continues on next page)

(continued from previous page)

```
_sched_extra: Node is idle
_sched_full: { ... full JSON blob }
```

Loosely speaking, `_sched_state` is a one-word summary of the state of the node (allocated, idle, down); `_sched_extra` is a one-line summary, potentially giving basic info on why the node might be in that state (e.g. not responding to pings might lead to a "down" indicator); and `_sched_full` is a JSON dump of all the information the scheduler provided for that node.

With the data in the nodes' attributes, admins can then use those attributes to select groups of nodes and target them with an action. For example, to list all nodes where Slurm is idle:

```
$ scyld-nodectl -s "attributes[_sched_state]==idle" ls
Nodes
  n0
  n1
  n2
```

One could similarly reboot all "down" nodes, or remotely execute a command to restart `slurmd` on any problematic nodes.

Note

At present, only the Slurm scheduler is supported, though other batch schedulers will likely be supported in the future.

4.32.1.1 sched-watcher Deployment

`sched_watcher` runs as a daemon process on a machine that has network access to both the batch system controller, e.g. `slurmctld`, and to the ClusterWare head node. While one could run `sched_watcher` directly on a head node, it is a better practice to run it on a ClusterWare management node to fully isolate any network or CPU load that it might generate.

For the `sched_watcher` server, the command-line tools for the batch scheduler must be installed, and it will be helpful to have the ClusterWare tools as well:

```
yum install clusterware-tools slurm-scyld-node
```

Copy the files from a head node:

```
scp -r /opt/scyld/clusterware/src/sched_watcher/* \
  mywatcher:/path/to/dest
```

On the `sched_watcher` server, prepare the SystemD service:

```
cp sched_watcher.service /etc/systemd/system/.
```

Modify the `sched_watcher.conf` file as needed (see below).

Create an authentication token using the `scyld-adminctl` tool:

```
scyld-adminctl token --lifetime 10d --outfile /tmp/cw_bearer_token
```

The default config file assumes `/tmp/cw_bearer_token` but any filename and path could be used. It is also possible to generate this token elsewhere and `scp` it to the `sched-watcher` server.

Enable and start the service:

```
systemctl enable sched_watcher
systemctl start sched_watcher
```

4.32.1.2 Verify Data

Once the `sched_watcher` tool is running, it should quickly push data to the ClusterWare platform and InfluxDB. On a ClusterWare head or management node, try:

```
scylld-nodectl ls -l
```

and verify that the `_sched_state` and other attributes are now populated.

Similarly, one can look in the monitoring GUI and the same data should be visible there.

Note

By default, the update cycle is every 30 seconds.

4.32.1.3 Config Settings

There are three main sections to the config file: one for `sched_watcher` (main) settings, and one for each of the output options (currently `slurm` and `influxdb`).

For `sched_watcher`, the following options are supported:

- `token_file_path = /tmp/cw_bearer_token`
 - Path to the authentication token file.
- `token_duration = 1h`
 - Since the auth-token will have some lifespan embedded within it, `sched_watcher` will periodically re-read the file assuming that it will be refreshed prior to expiration. `token_duration` sets the time-frame for re-reading the file.
- `polling_interval = 30`
 - Sets the time between sending of updates to the ClusterWare software. A longer interval can potentially reduce the load on the system, but the data will be more out-of-date.
- `sched_type = slurm`
 - Sets the "type" of batch scheduler to retrieve data from. At present, only `slurm` is supported.
- `debug_level = 1`
 - Enables debugging output.
- `input = slurm`
 - A comma-separated list of input modules that will be used. At present, `slurm` and `clusterware` are supported.
- `output = clusterware, influxdb`
 - A comma-separated list of output modules that should be used. It can include one or more of: `clusterware`, `influxdb`, or `archive`. If admins do not wish to use InfluxDB/Telegraf monitoring, it can be removed from this list.

For the ClusterWare platform, only one option is currently supported:

- `base_url = http://parent-head-node/api/v1/`
 - Sets the base URL for the ClusterWare platform. Best practice would be to run `sched_watcher` on a ClusterWare management node, so `parent-head-node` will be kept up-to-date and will always point at a valid head node.

For Slurm, only one option is currently supported:

- `base_path = /opt/scyld/slurm/bin`
 - Sets the base path to all of the Slurm command-line tools. This is where `sched_watcher` will look for the `sinfo` and `squeue` tools.

For InfluxDB, the options are:

- `base_url = udp://parent-head-node:8094`
 - Sets the base URL for the InfluxDB service. Best practice would be to not run `sched_watcher` on a ClusterWare management node, so `parent-head-node` will be updated to point at a valid head node.
- `include_sched = true`
 - For reduced data size, `sched_watcher` can enable or disable the `_sched_state` information.
- `include_extra = true`
 - By default, `sched_watcher` will only push the `_sched_state` information. Setting this to `true` will also push the `_sched_extra` (one line) summary into InfluxDB. At this time, there is no support for sending the full JSON data into InfluxDB.
- `include_cw_data = false`
 - Indicates if the data from the ClusterWare platform should be written to the InfluxDB endpoint. For example, one might want to archive the ClusterWare data, but not send it to InfluxDB.
- `drop_cw_fields = *`
 - A simple filter system to allow some ClusterWare fields to be dropped, keeping all the others. The `*` is a wildcard that matches any number of any character.
- `keep_cw_fields = a.*`
 - A simple filter system that will keep certain ClusterWare fields even if they were otherwise selected by the `drop_cw_fields` filter. The `*` is a wildcard that matches any number of any character

For the archive file output, the options are:

- `output_file = /tmp/cw_archive`
 - The full path to the archive file.
- `rotate_interval = 1d`
 - How often the archive file should be rotated (can use `h` for hours, `d` for days).
- `zip_prog = /usr/bin/gzip`
 - If given, the rotated (old) archive files will be compressed with the given tool to reduce storage requirements.
- `drop_cw_fields = *`
 - A simple filter system to allow some ClusterWare fields to be dropped, keeping all the others. The `*` is a wildcard that matches any number of any character.
- `keep_cw_fields = a.*`
 - A simple filter system that will keep certain ClusterWare fields even if they were otherwise selected by the `drop_cw_fields` filter. The `*` is a wildcard that matches any number of any character

4.32.1.4 Notes

- The code currently runs as root so that it can read the config file in `/opt/scyld/clusterware` and also read the admin-created token file
- The `sched_watcher` tool cannot refresh the auth-token that it's been given, so there must be some other (out-of-band) process to refresh that bearer token before it expires.
 - e.g. One could run a weekly cron job that executes the `scyld-adminctl token` command.
- Restart/Reload `sched_watcher` after any changes to the config file.
 - To reload the config and auth-token files: `systemctl reload sched_watcher`
 - To completely restart the system: `systemctl restart sched_watcher`
- The archived data is in a straightforward key=value format. Each line includes `time=<Unix timestamp>` and `host=<hostname>` followed by the data fields for that node.
 - The raw data is "flattened" into a single-level set of dotted keys, so `clusterware.attributes.foo` would become `c.a.foo=value` * `attributes` becomes `a`, `status: s`, `hardware: h`
- There is simple filtering available with some outputs modules: `keep_cw_fields` and `drop_cw_fields`.
 - Both can be comma-separated lists of fields that should be included or excluded, and both can include a trailing wildcard (`*`)
 - `keep_cw_fields` are retained in the output even if there is a matching `drop_cw_fields` key
 - `drop_cw_fields=* and keep_cw_fields=a.*`
 - * drop all fields except for `a.*` fields (attributes)
 - `drop_cw_fields=(empty) and keep_cw_fields=*`
 - * retain all fields

Example Config

```
[sched_watcher]
token_file_path = /tmp/cw_bearer_token
token_duration = 1h
polling_interval = 30
sched_type = slurm
debug_level = 1
# available inputs = clusterware, slurm
input = slurm
# available outputs = archive, clusterware, influxdb
output = clusterware

[clusterware]
base_url = http://parent-head-node/api/v1/

[slurm]
base_path = /opt/scyld/slurm/bin

[influxdb]
base_url = udp://parent-head-node:8094
include_sched = true
include_sched_extra = false
include_cw_data = false
```

(continues on next page)

(continued from previous page)

```
# drop most fields ...
drop_cw_fields = *
# but keep these ...
keep_cw_fields = a.*

[archive]
output_file = /tmp/cw_archive
rotate_interval = 1d
zip_prog = /usr/bin/gzip
# drop_cw_fields = *
keep_cw_fields = *
```

4.32.2 Applications Report Excessive Interruptions and Jitter

In certain circumstances that are more common with real-time, performance sensitive multi-node applications, applications may occasionally suffer noticeable unwanted interruptions or "jitter" that affects the application's stability and predictability.

Some issues may be remedied by having the affected compute nodes execute in "busy mode", during which the node's *cw-status-updater* service severely reduces the scope of what information it periodically gathers and reports to the node's parent. That service's normal operation may exhibit an infrequent 1-2 second computation stall, which in a cluster with hundreds or thousands of nodes may affect a multi-node real-time application's otherwise rapid periodic syncing.

"Busy mode" can be enabled in one of three ways:

- Set the node's boolean *_busy* reserved attribute to True with a case-insensitive 1, "on", "y", "yes", "t", or "true". See the *_busy* attribute in *Reserved Attributes* for details. Turn off "busy mode" by setting *_busy* to False with 1, "off", "n", "no", "f", or "false", or by clearing the *_busy* attribute completely.
- Execute `touch /opt/scyld/clusterware-node/etc/busy.flag` on the node in a job scheduler prologue to enable and `rm` that file in an epilogue to disable. This *busy.flag* method is ignored if the node's *_busy* attribute is explicitly set to True or False.
- The node's *cw-status-updater* service may heuristically decide on its own to execute in "busy mode". This method is overridden by the presence of *busy.flag* or by an explicit *_busy* attribute setting.

An additional approach is to employ cpusets to execute specific applications on specific node cores in order to minimize contention. See the *_status_cpuset* attribute in *Reserved Attributes* for details about how to do this for the *cw-status-updater* service, and consult your Linux distribution or job scheduler documentation for how to do this for your applications.

4.33 Role-Based Access Control System

As described in *Role-Based Access Controls*, the ICE ClusterWare™ Role-Based Access Control system has 6 primary roles plus the "No Access" pseudo-role. A role is a set of 1 or more permissions, usually grouped according to an employee's job-related needs. E.g. there is an Onsite Engineer role that is designed to capture all the actions a "rack-and-cable" technician might need in their daily job. The ClusterWare platform ships with a set of default roles, but admins can also create new roles or modify the existing ones.

4.33.1 Permissions

Every database object type has at least a Read and Write permission, and many have additional permissions specific to that object type. As an example, the Boot Configs-Read permission which allows an admin to read the information about any Boot Configuration in the system. There is also an Image-Write permission that allows an admin to create or overwrite/modify any Image in the system.

Object Type	Permissions	
Admin	AdminsRead	AdminsWrite
Attributes	AttribGroupsRead	AttribGroupsWrite
Groups	AttribGroupsReadReserv	AttribGroupsWriteReserv
Boot Configs	BootConfigsRead	BootConfigsWrite
Certificates	CertsRead	CertsWrite
Distros	DistrosRead	DistrosWrite
Dynamic Groups	DynGroupsRead	DynGroupsWrite
Git Repos	GitReposRead	GitReposWrite
Head Nodes	HeadsRead	HeadsWrite
Hostnames	HostnamesRead	HostnamesWrite
Images	ImagesRead	ImagesWrite
Naming Pools	NamingPoolsRead	NamingPoolsWrite
Networks	NetworksRead	NetworksWrite
Nodes	NodesRead	NodesWrite
	NodesExecCommand	NodesPowerControl
	NodesReadReserv	NodesWriteReserv
	NodesWriteGroups	
Repos	ReposRead	ReposWrite
State Sets	StateSetsRead	StateSetsWrite

Note that all Write permissions include the ability to create, modify, and delete objects of that type. All Read permissions include the ability to list all objects of that type and to see their details.

- Nodes and Attribute Groups have ReadReserv and WriteReserv permissions which allow reading/writing of the reserved attributes, including those attributes that control how the nodes boot (see *Reserved Attributes*).
- Nodes have an additional permission to allow execution of code and ssh access, NodesExecCommand, and another one to allow power control through the BMC or IPMI, NodesPowerControl.
- Nodes also have NodesWriteGroups to allow changing which group(s) a given node is joined to (this permission allows addition and removal of groups).

4.33.2 Roles

The default set of roles provided by the ClusterWare software are: Authenticated User, Onsite Engineer, Imaging Engineer, Production Engineer, Manager, and Full Admin. The "No Access" pseudo-role can be used to quickly and easily block access to a user. See *Role-Based Access Controls* for a description of each role.

Note

The No Access role supercedes all other roles and will block even Full Admins from performing any action.

Role	Permission Set
AuthUser	all Read permissions for all Object types
FullAdmin	all Read and Write permissions for all Object types; all special permissions for all Object types
NoAccess	all permissions are revoked; account is blocked
ImageEngineer	all Read permissions for all Object types, plus: ImagesWrite, NodesReadReserv, AttribGroupsReadReserv
Manager	all Read permissions for all Object types, plus: AdminsWrite, NodesReadReserv, AttribGroupsReadReserv
OnsiteEngineer	all Read permissions for all Object types, plus: NodesPowerControl, NodesWrite
ProductionEngineer	all Read permissions for all Object types, plus: AttribGroupsReadReserv, AttribGroupsWriteReserv NodesWrite, NodesWriteGroups, NodesWriteReserv, NodesPowerControl, AttribGroupsWrite, ImagesWrite NamingPoolsWrite, BootConfigsWrite, GitReposWrite, NodesPowerControl

4.33.3 Modifying the Role-Permissions Mapping

The role-to-permissions mapping is stored in a configuration file on the head nodes -- `/opt/scyld/clusterware/src/cw_common/files/roles.ini`. This is a standard INI file format with only one section named "Roles". Within that section, admins can define one or more role using lines like:

```
role_name = permission1, permission2
```

Alternately, permissions can be listed one per line (without commas).

Note that all roles automatically have all Read permissions to all object types, and the Authenticated User and Full Admin roles are always pre-defined and do not need to be manually created. An example file might include:

```
[Roles]
# custom On-site engineer that adds exec and power-control
MyOnsiteEngineer = NodesPowerControl
                   NodesWrite
                   NodesExecCommand
                   NetworksWrite
```

Once the `roles.ini` file has been modified, it must be copied to every head node in the cluster and must overwrite the current `/opt/scyld/clusterware/src/cw_common/files/roles.ini` file. Once all head nodes have the new file, the `clusterware` service must be restarted on every head node. To avoid any disruption in cluster services, admins should restart the `clusterware` service on one head node at a time, waiting for it to become fully operational before issuing the restart on the next head node.

Note

The ClusterWare platform may change how these configuration settings are stored in the future. The `roles.ini` file

may be deprecated in favor of storing the values in the main ClusterWare database.

Note

Admins are strongly encouraged to work with ClusterWare staff when looking to modify this file since misconfiguration could lead to non-functioning cluster head nodes.

4.34 Users and Groups Page

The **Users and Groups** page lists all administrator users, their permissions, and the date and time of their last login. Use this page to add and edit ICE ClusterWare™ administrators. The page is available via **User Management > Users + Groups** in the left navigation panel.

Users and Groups

Refresh Interval (seconds):

Manage user accounts. Administrator accounts defined in base.ini using the auth.tmpadmins setting are flagged. Roles are read-only on this page and must be managed in Keycloak. You may configure role mapping in Settings.

► Role descriptions

Name ▲	Roles ▲	Last Login ▲
██████████	fulladmin	2025-03-04 23:35:07
██████████	authuser fulladmin	2024-10-16 16:13:10
██████████	fulladmin	2025-03-05 01:23:40
██████████	authuser fulladmin	2025-03-05 04:47:42
██████████	fulladmin	2025-02-21 02:06:37
██████████	authuser fulladmin	2025-02-20 04:23:35
██████████	fulladmin	2024-05-02 20:16:47
██████████	authuser fulladmin	2025-01-29 21:44:42

ADD ADMINISTRATOR

Note

All users listed in the **Users** table can be assigned administrator roles and permissions.

Administrators granted temporary access via "auth.tmpadmins" as defined in the /opt/scyld/clusterware/conf/base.ini file are flagged in the list. See [Authentication](#) for details about enabling temporary permissions.

4.34.1 Add Administrator

Note

If you configured the ClusterWare platform with Keycloak, users must be added in both the Keycloak and ClusterWare softwares. See *Integrating Keycloak with ICE ClusterWare for RBAC* for details.

To create an administrator:

1. Click **Add Administrator**. The **Add Administrator** pane opens.
2. Add details about the administrator. **Name** and **RBAC Roles** are required fields. Default roles and descriptions are available at the top of the page by expanding the **Role descriptions** section. See *Roles* for a list of permissions granted by each default role.
3. Click **Add Administrator** to save your changes. The new administrator is added to the list.

4.34.2 Edit Administrator

Note

If you configured the ClusterWare platform with Keycloak, modifying roles must be done in both the Keycloak and ClusterWare softwares. See *Integrating Keycloak with ICE ClusterWare for RBAC* for details.

To edit an administrator:

1. Click the ellipsis (...) on the far right of the row and select the **Edit** action. The **Edit Administrator** pane appears and populates with the administrator's details.
2. Make updates to the administrator.
3. Click **Edit Administrator** to save your changes.

4.34.3 Delete Administrator

To delete an administrator, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.34.4 Related Links

- *Role-Based Access Controls*
- *Role-Based Access Control System*
- *Configure Additional Cluster Administrators*
- *scyld-adminctl*

4.35 Configure Additional Cluster Administrators

The ICE ClusterWare™ administrator's command-line tools are found in the *clusterware-tools* package, which is installed by default on the head node by `scyld-install`. It can be additionally installed on any system that has HTTP (or HTTPS, see *Securing the Cluster*) access to a ClusterWare head node in the cluster.

To install these tools on a machine other than the head node, login to that other system, copy `/etc/yum.repos.d/clusterware.repo` from a head node to the same location on this system, then execute:


```
sudo yum install clusterware-tools
```

Once the tools are installed, each administrator must configure a connection to the ClusterWare service, which is controlled by variables in the user's `~/ .scyldcw/settings.ini` file. The `scyld-tool-config` tool script is provided by the `clusterware-tools` package. The contents of the `settings.ini` file are discussed in the *ICE ClusterWare Command Line Tools*. Running that tool and answering the on-screen questions will generate a `settings.ini` file, although administrators of more advanced cluster configurations may need to manually add or edit additional variables.

Once the `settings.ini` is created, you can test your connection by running a simple node query:

```
scyld-nodectl ls
```

This query may complain at this time that no nodes exist or no nodes are selected, although such a complaint does verify that the requesting node can properly communicate with a head node database. However, if you see an error resembling the one below, check your `settings.ini` contents and your network configuration:

```
Failed to connect to the ClusterWare service. Please check that the
service is running and your base_url is set correctly in
/home/adminuser/.scyldcw/settings.ini or on the command line.
```

The connection URL and username can also be overridden for an individual program execution using the `--base-url` and `--user` options available for all `scyld-*` commands. The `settings.ini` file generated by `scyld-install` will contain a blank `client.authpass` variable. This is provided for convenience during installation, though for production clusters the system administrator will want to enforce authentication restrictions. See details in *Securing the Cluster*.

4.35.1 scyld-adminctl

NAME

scyld-adminctl -- Query and modify administrators for the cluster.

USAGE

scyld-adminctl

```
[-h] [-v] [-q] [[-c | --config] CONFIG] [--base-url URL] [[-u | --user] USER[:PASSWD]]
[--human | --json | --csv | --table] [--pretty | --no-pretty] [--fields FIELDS]
[--show-uids] [[-i | --ids] ADMINS | -a | --all] {list,ls, create,mk, clone,cp,
update,up, replace,re, delete,rm, token}
```

DESCRIPTION

This tool does not control the details of authentication. For that, please consult *Securing the Cluster* in the documentation.

OPTIONAL ARGUMENTS

- h, --help** Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
- v, --verbose** Increase verbosity.
- q, --quiet** Decrease verbosity.
- c, --config CONFIG** Specify a client configuration file *CONFIG*.
- show-uids** Do not try to make the output more human readable.
- a, --all** Interact with all administrators (default for list).
- i, --ids ADMINS** A comma-separated list of administrators to query or modify.

ARGUMENTS TO OVERRIDE BASIC CONFIGURATION DETAILS

--base-url URL Specify the base URL of the ClusterWare REST API.

-u, --user USER[:PASSWD]

Masquerade as user *USER* with optional colon-separated password *PASSWD*.

FORMATTING ARGUMENTS

--human Format the output for readability (default).

--json Format the output as JSON.

--csv Format the output as CSV.

--table Format the output as a table.

--pretty Indent JSON or XML output, and substitute human readable output for other formats.

--no-pretty Opposite of `--pretty`.

--fields FIELDS Select individual fields in the result or error.

ACTIONS ON SPECIFIED ADMINISTRATOR(S)**list (ls)**

List information about administrator(s).

create (mk) name=NAME

Add an administrator *NAME*.

clone (cp) name=NAME

Copy administrator to new identifier *NAME*.

update (up)

Modify administrator fields.

replace (re)

Replace all administrator fields. Deprecated in favor of "update".

delete (rm)

Delete administrator(s).

token

Create a new bearer token with selected properties.

EXAMPLES

```
scyld-adminctl create name=hsolo
```

Add new administrator "hsolo".

```
scyld-adminctl -i hsolo clone name=cbaca
```

Copy the administrator properties for "hsolo" to a new administrator "cbaca".

RETURN VALUES

Upon successful completion, **scyld-adminctl** returns 0. On failure, an error message is printed to `stderr` and **scyld-adminctl** returns 1.

4.36 User Impersonation

ICE ClusterWare™ users with the AdminWrite permission can impersonate another user, which allows them to complete actions like creating a ticket that appears to originate from another user. See *Role-Based Access Control System* for details about default roles and permissions.

Run the following command to generate a token for the user you want to impersonate:

```
scyld-adminctl -i <otheruser> token
```

Where <otheruser> is replaced by the user ID of the user you want to impersonate.

For example, if user admin1 runs the following command, a token will be generated for user user1:

```
scyld-adminctl -i user1 token
```

Note

To use this feature with Keycloak, you first need to enable fine-grained permissions and the "impersonate" feature. Contact Penguin Computing for assistance.

4.37 Integrating Keycloak with ICE ClusterWare for RBAC

4.37.1 Installation

Keycloak is a powerful, flexible authentication and authorization system with the ability to directly store user credentials and to link to other authentication providers. The ICE ClusterWare™ platform's use of Keycloak does not necessarily use all of its features, and the full configuration process for all of Keycloak is beyond the scope of this document. For more in-depth information, admins can refer to the Keycloak website and, particularly, its documentation link:

<https://www.keycloak.org/documentation>

Assuming Keycloak is installed and operational, the process to integrate it with the ClusterWare platform involves selecting or creating a realm, creating a new client within that realm, adding users to the realm, assigning or creating roles that will map to ClusterWare roles, and finally, configuring and restarting the ClusterWare software.

Installation instructions can be found in Keycloak's documentation. This includes how-to guides for several different types of installations: bare metal, Docker, Podman, Kubernetes, and OpenShift.

<https://www.keycloak.org/guides#getting-started> <https://www.keycloak.org/operator/basic-deployment>

4.37.2 Select a Realm

In Keycloak, a "realm" is an administrative domain that includes users, groups, roles, and client application information. A fresh installation of Keycloak will only have one realm, called "master", and it is highly recommended that admins create at least one other realm to use in production. The "master" realm should not be used for daily operations. In an enterprise organization, there may already be a suitable realm available. Since client-specific information can be compartmentalized even within a realm, there shouldn't be any concern about the ClusterWare platform's use of that realm interfering with other operations.

Note

When creating a new realm, the name that is given will be used as the unique identifier for the realm, and will show up in the URL.

More information can be found in Keycloak's Server Administration documentation:

https://www.keycloak.org/docs/latest/server_admin/index.html

Configuring realms:

https://www.keycloak.org/docs/latest/server_admin/index.html#configuring-realms

4.37.3 Create a New Client

Inside the selected realm, admins should create a new client – a client is an administrative container for information related to a specific application or use-case within this realm. In this case, the ClusterWare platform will be given its own client-container so that it does not interfere with any other applications that may be using the same Keycloak instance.

When creating the client, note that the "client ID" acts as the unique identifier and as a "short name" for the client - it will be displayed on many of the Keycloak webpages and it will show up in the URL.

Specific settings to be verified:

- The "Client authentication" and "Authorization" checkboxes must be enabled.
- In the "Authentication flow" section, the "Standard flow" and "Direct access grants" checkboxes must be enabled.
- The "Use refresh tokens" checkbox should be enabled (in the web-UI, this is under the "Advanced" tab).

Take note of the following information as it will be needed later during the ClusterWare configuration process:

- The server connection information, including the server address or name and any port information
- The realm name and client-ID
- The "client secret" – in the web-UI, this can be found in the Credentials tab; the "Client Authenticator" should be "Client Id and Secret", and under it should be "Client Secret"

More information can be found in Keycloak's Server Administration documentation:

https://www.keycloak.org/docs/latest/server_admin/index.html https://www.keycloak.org/docs/latest/server_admin/index.html#_oidc_clients

4.37.4 Add Users

After setting up these basic "containers" for interacting with the ClusterWare platform, the next step is to create any users that will need ClusterWare admin roles. For enterprise organizations, these accounts may already be created and can be re-used for ClusterWare roles.

The user creation process is straightforward – each user needs a unique username (unique within the realm), and can optionally be given first/last names and an email address. If desired, users can be configured to be required to change their password at first login, or required to verify their email address, etc. If Keycloak was configured for other organizational uses, it may require additional or different user information, e.g. a physical address or an employee ID number.

It may be worth considering Keycloak's "Groups" feature to categorize and organize users to simplify user-to-role mapping. E.g. a group could be defined for HPC-Admins and that group can then be granted a role of `hpc.fulladmin`; any user added to that group will thus become a Full Admin. This can greatly simplify the process of adding roles to large numbers of users.

More details can be found in Keycloak's Server Administration documentation:

https://www.keycloak.org/docs/latest/server_admin/index.html

User management:

https://www.keycloak.org/docs/latest/server_admin/index.html#assembly-managing-users_server_administration_guide

4.37.5 Select or Create Roles

For a given user, the ClusterWare platform will look at the Keycloak-provided list of roles to determine what that user is or is not allowed to do. The configuration process (see below) will need to map each of the ClusterWare roles to a Keycloak-provided role. For enterprise organizations, there may be existing roles that can be re-used for ClusterWare roles, but otherwise, admins will have to create any needed roles.

The ClusterWare roles are:

- On-site Engineer
- Imaging Engineer
- Production Engineer
- Manager
- Full Admin
- Authenticated User (this role is also granted to anyone in any of the other roles)

Once the roles are defined, users can then be assigned to those roles either directly or through a group.

Note

Admins may choose to not provide a mapping for a given ClusterWare role. In that case, then that role will simply be ignored. For example, if there is no mapping for the Manager role, then no one will ever be able to be assigned that role.

More information can be found in Keycloak's Server Administration documentation:

https://www.keycloak.org/docs/latest/server_admin/index.html

Roles and Groups:

https://www.keycloak.org/docs/latest/server_admin/index.html#assigning-permissions-using-roles-and-groups

4.37.6 Configuring ClusterWare Software

With the Keycloak configuration from above, admins can now configure the ClusterWare software to match. In particular, the ClusterWare software needs to know the basic configuration and the role-mapping details.

The Keycloak configuration info is stored in the ClusterWare database with the `authctl` tool, found at `/opt/scyld/clusterware/bin/authctl`. When using `authctl`, replace the values with info from the actual Keycloak instance as noted above:

```
authctl set keycloak base_url https://192.168.122.33:8080
authctl set keycloak client clusterware
authctl set keycloak realm penguin
authctl set keycloak secret abc123def456ghi789
```

Next, the role-mapping needs to be configured. The mapping is given as a comma-separated list of `keycloak_role=clusterware_role` settings. For example:

```
authctl set keycloak role_mapping "kc_fulladmin=role.fulladmin,kc_manager=role.manager"
```

Note that the `authctl` tool can also be used to double-check the data in the ClusterWare database:

```
authctl show
```

Finally, the main ClusterWare configuration needs to be modified to swap out the standard authentication system (the `appauth` module) with the Keycloak module. Edit `/opt/scyld/clusterware/conf/base.ini` and look for the `plugins.auth` section:

```
plugins.auth = appauth
```

Change this to:

```
plugins.auth = keycloakauth
```

Note that it may be helpful to add a user or two to the `auth.tmpadmins` list in `base.ini` as well. If anything goes wrong with the Keycloak integration process, most users will not be able to authenticate – only the `tmpadmins` will be allowed into the system, and they will have full admin privileges.

When those file modifications are complete, save the file and restart the ClusterWare service:

```
systemctl restart clusterware
```

Important

For multi-head clusters, the `base.ini` file should be modified on all heads, then the service on each head node should be restarted.

4.37.6.1 Production Operations

Once the steps in the Installation process are completed and the ClusterWare service is restarted, the cluster should now be looking to Keycloak for authentication credentials and roles. For production operations, there are several important changes to how admins interact with the system.

4.37.7 User Management

Adding new users and modifying roles must now be done in both the Keycloak and ClusterWare systems. The instructions for adding a new user to Keycloak are shown in the "Add Users" and "Select or Create Roles" sections above. Separately, the user must be added to the ClusterWare platform through the `scyld-adminctl` command:

```
scyld-adminctl create name=username
```

Note

When using Keycloak, no roles should be assigned in the ClusterWare platform – all roles must be assigned through Keycloak.

To allow admins to create new tokens using `scyld-adminctl token`, several features must be enabled when starting Keycloak: `token-exchange` and `admin-fine-grained-authz`. Note that these features are marked by the Keycloak team as "Preview" features and may not be suitable for production environments. The features may be set inside the config file or on the command line.

4.37.8 Logging and Auditing

Keycloak includes configurable logging and auditing settings that admins can use to directly track user behavior on the cluster, or alternatively, to feed into another system for intrusion detection or log analysis.

By default, the log file will be `keycloak.log` and will be found in the root directory where keycloak was installed, i.e. the `<kc-install-root>/data/log` directory. It should include all log messages except DEBUG level messages (so INFO, WARNING, ERROR, etc. should all be logged). The file follows standard Unix/Linux syslog formatting and should be easily interpretable by other tools. The documentation links below include examples of forwarding the data to Graylog and ELK log aggregation systems.

There are several sections of the Keycloak documentation that may be of interest:

- <https://www.keycloak.org/server/logging>
- https://www.keycloak.org/docs/latest/server_admin/index.html#configuring-auditing-to-track-events

4.37.9 Access Token Lifespan

When an admin logs in to a ClusterWare cluster, the ClusterWare server will pass the information to Keycloak, which will verify the credentials and return a set of tokens (JWT, Java Web Tokens). Those tokens will be cached in each admin's `.scyldcw` directory, in the `auth_token.hdr` file. That file will include the current "access token" that can be used to perform actions in the ClusterWare system, as well as a "refresh token" that can be used to refresh the access token if or when it expires. If the refresh token does expire, the user will be prompted for their password, which will grant them a new set of tokens. The ClusterWare tools will automatically use the tokens in a way that should reduce the number of times the admin has to enter their password.

Current best practice is to limit the access token to a lifetime of approximately 5-10 minutes while allowing the refresh token to have a longer duration, such as 30-60 minutes. If a credential is stolen (or the file copied), then the Keycloak admins can potentially invalidate the refresh token and limit exposure to the compromised tokens. Token expiration settings for both access and refresh tokens will now be managed in Keycloak, so admins may re-configure those settings to reduce the frequency that passwords need to be re-entered.

4.38 Integrating FreeIPA with ICE ClusterWare

4.38.1 Installation

FreeIPA is a powerful, open-source identity management system that can export an LDAP directory of user credentials which can then be ingested into Keycloak and other authentication systems. The ICE ClusterWare™ platform's use of FreeIPA does not necessarily use all of its features, and the full configuration process for all of FreeIPA is beyond the scope of this document. For more in-depth information, admins can refer to the FreeIPA website and, particularly, its documentation link:

<https://www.freeipa.org/page/Documentation.html>

Assuming FreeIPA is installed and operational, the process to integrate it with Keycloak involves creating a new "User Federation" connection, including an optional "User LDAP Filter" to reduce the user accounts to only those in a certain group. Once configured, any users that match the filter will be available inside Keycloak and can then be assigned roles and added to the ClusterWare software.

Installation instructions can be found in FreeIPA's documentation. FreeIPA provides RPM and DEB packages, as well as container deployment options.

https://www.freeipa.org/page/Quick_Start_Guide <https://www.freeipa.org/Downloads.html>

4.38.2 Identify a Group for ClusterWare Users

In a larger enterprise, it may make sense to select or create a group that will be used to identify users who have administrative access to the ClusterWare system. This can help reduce the number of accounts that are being sync'ed between Keycloak and FreeIPA. If a group already exists for those users, the full DN for the group should be noted. Otherwise, a new group should be created and any ClusterWare admins should be added to that group.

By default, FreeIPA places user groups in a DN of the form:

```
cn=keycloak-allowed,cn=groups,cn=accounts,dc=<companyname>,dc=<com>
```

The DC components at the end will be dependent on the domain name as it is configured inside FreeIPA.

4.38.3 Identify an Admin Account for Keycloak

Keycloak will need to authenticate in order to access FreeIPA, so it will need an admin-level account to do so. If an account already exists, the full DN for the account should be noted. Otherwise, a new account should be created.

By default, FreeIPA places users in a DN of the form:

```
uid=<kcadmin>,cn=users,cn=accounts,dc=<companyname>,dc=<com>
```

Again, the DC components will reflect the domain name, and the *kcadmin* component will be the username.

4.38.4 Configure Keycloak

In Keycloak, switch to the realm that is being used by the ClusterWare system and look for the "User federation" tab in the left menu. When creating a new User-federation connection, select the "LDAP" option.

Several settings need to be configured:

- For the "Vendor", select "Red Hat Directory Services"
- Connection URL will be the URL to get to the FreeIPA server. Note that this should start with a prefix of `ldap://` or `ldaps://`.
 - Use the "Test connection" button to verify that Keycloak can reach FreeIPA over the network. If any problems show up, it may indicate firewall or other network issues.
- Bind type is "simple", and use the DN for the admin user selected or created above. E.g. `uid=<kcadmin>,cn=users,cn=accounts,dc=<companyname>,dc=<com>`. Enter the admin password into the "Bind credential" text box.
 - Use the "Test authentication" button to verify that the username and password are working correctly, and that FreeIPA is responding properly to Keycloak's requests.
- Edit mode should be "READ_ONLY"
- For the default FreeIPA settings, the Users DN field should be `cn=users,cn=accounts,dc=<companyname>,dc=<com>`.
- Username LDAP attribute and RDN LDAP attribute should both be set to `uid`.
- UUID LDAP attribute should be `uidNumber`.
- Although optional, admins are encouraged to set a User LDAP filter to reduce the number of user accounts that are downloaded and sync'ed from FreeIPA to Keycloak.
 - For a newly created group in a default FreeIPA system, a suitable filter might be: `(memberOf=cn=<keycloak-allowed>,cn=groups,cn=accounts,dc=<companyname>,dc=<com>)` where `keycloak-group` is the name of the group that contains all ClusterWare admins.

LDAP offers a power filtering syntax that can allow for one or more user-groups, or even selecting users by one or more roles. See <https://ldap.com/ldap-filters/> for more information.

For all other settings, the defaults should work. Larger enterprises may want to think through the Periodic full sync and Periodic changed users sync settings to limit the load on the FreeIPA servers; note that those settings are defined in number of seconds between synchronization events.

Note that one must click the "Save" button at the bottom of the webpage or else any changes will be lost!

4.38.5 Verifying the Integration

Once the User federation connection is created, the connection can be verified by going to the Users page in the Keycloak web-UI. Since there is a federation defined, the Users webpage will only present a search box, not a list of known users. To see all users, simply search for *. Depending on the number of accounts that need to be downloaded or sync'ed, it may take a few seconds before results are rendered.

Once the page updates, all available users should be shown -- all users defined inside Keycloak as well as those defined in FreeIPA.

Note

Keycloak and FreeIPA accounts can co-exist, so it may be useful to create a few Keycloak-only users just in case the FreeIPA connection goes down.

In addition to the "Test" buttons in the Keycloak web-UI, one can always go directly to the Keycloak and FreeIPA web-UIs to do some "debugging".

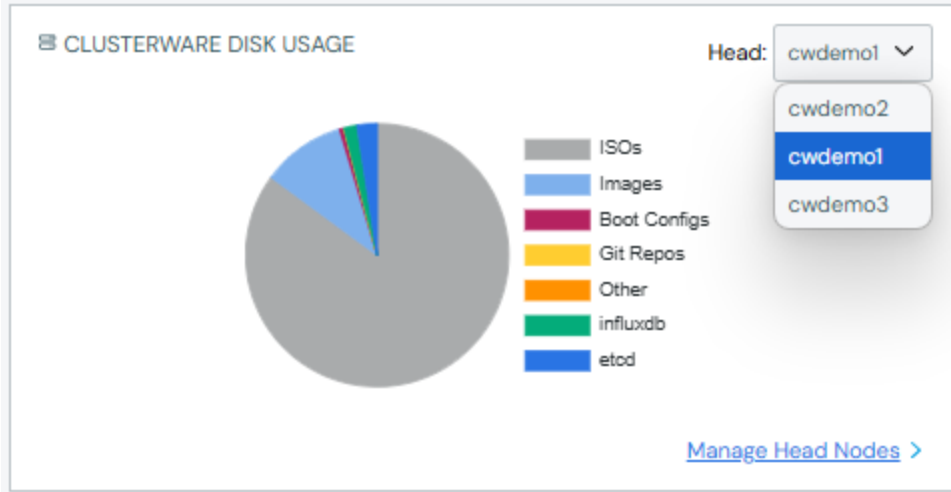
- The FreeIPA login page should be at <https://<freeipa-base-url>/ipa/ui/> (be sure to logout from any current sessions). This can be useful in verifying that a given username and password work at all. If FreeIPA does not allow the account, then Keycloak will never see it and the problem is likely inside FreeIPA.
- The Keycloak login page for a realm should be at: <https://<keycloak-base-url>:8080/realms/<realm-name>/account/> (be sure to logout from any current sessions). If Keycloak does not allow the account, then it could be a problem inside Keycloak.

Note

When using Keycloak and FreeIPA, no roles should be assigned in the ClusterWare software – all roles must be assigned through Keycloak.

4.39 Head Nodes Page

The **Head Nodes** page is available via the left navigation panel under **Cluster > Head Nodes**. You can also access the **Head Nodes** page for a specific ICE ClusterWare™ head node via the **Overview** page's **ClusterWare Disk Usage** panel. The panel has a pulldown menu on the upper right where you can select a cluster head node to display.



In example above, there are three headnodes. The selected head node displays details for the selected head node's disk usages for each of the various types of ClusterWare entities: ISOs, images, boot configurations, Git repositories, "other", influxdb data, and the etcd database.

Note

These are just the proportional disk usages for ClusterWare entities, not for the node as a whole.

In the **ClusterWare Disk Usage** panel, click the **Manage Head Nodes** link in the lower right to open the **Head Nodes** page, which displays details about every cluster head node.

Head Nodes

Refresh Interval (seconds): 10

Head nodes provide services to the compute nodes and must be added through joining installed heads to an existing head node.

HEAD cwdemo2.demo.local

Status	ACTIVE
ClusterWare Version	12.4.1-g0000.el8.x86_64
Public Key	ssh-rsa AAAAB3NzaC1yc2EAAAADAQAB...

HEAD cwdemo1.demo.local

Status	ACTIVE
ClusterWare Version	12.4.1-g0000.el8.x86_64
Public Key	ssh-rsa AAAAB3NzaC1yc2EAAAADAQAB...

HEAD cwdemo3.demo.local

Status	ACTIVE
ClusterWare Version	12.4.1-g0000.el8.x86_64
Public Key	ssh-rsa AAAAB3NzaC1yc2EAAAADAQAB...

The **Head Nodes** page provides a view of basic head node status and allows users to copy information about the ClusterWare version or the public key that the head node uses to connect to compute nodes. The version string is useful when reporting an issue to Penguin Computing support.

Although each head node's pie chart is likely to display small differences in the sizes of the pie components, you can hover the cursor over specific components to see their absolute sizes and note that the shared objects (ISOs, Images, Boot Configs) show the identical sizes across the head nodes.

The "Other" category consists of files in the ClusterWare storage directory that are not recognized by the system. These are usually files left behind during partial uploads, interrupted image cloning, or other failure cases. If the cluster is working as expected though there is space consumed by this "Other" category, those files can be removed via the "Clean storage/" option in the **More** (...) menu at the top of each head node panel. This action is equivalent to executing the following command:

```
scyld-clusterctl heads -i <HEADNODE> clean --files
```

4.40 Important Files on Head Nodes

4.40.1 The ~/.scyldcw/ Folder

As described elsewhere in this document, ICE ClusterWare™ administrator tools read some configuration details from the user's ~/.scyldcw/settings.ini file. This section describes the other common contents of the ~/.scyldcw/ folder. Although this is included in the *Important Files on Head Nodes* documentation, please note that this folder exists in the home directory of any user who executes the ClusterWare tools, and that these tools are intended to be installed not just on the head node, but also wherever an administrator finds convenient and has appropriate HTTP or HTTPS access to the head node.

4.40.1.1 auth_tkt.cookie

Whenever a user authenticates to the REST API running on a head node, an authentication cookie is generated and used for subsequent requests in the same session. Even though sessions typically end when the executed tool completes, the command line tools caches the authentication cookie in the ~/.scyldcw/auth_tkt.cookie file to allow for faster tool start times. A summary of the network requests are logged at the DEBUG level:

```
[sysadmin@virthead ~]$ scyld-nodectl -vv ls
DEBUG: GETing /node/{uid} through /mux
INFO: No value provided for global option 'client.auth_tkt'.
DEBUG: Starting new HTTP connection (1): localhost:80
DEBUG: http://localhost:80 "GET /api/v1/whoami HTTP/1.1" 200 74
DEBUG: Starting new HTTP connection (1): localhost:80
DEBUG: http://localhost:80 "GET /api/v1/whoami HTTP/1.1" 200 109
INFO: Loaded authentication cookie from previous run.
DEBUG: http://localhost:80 "POST /api/v1/mux?log=GET-/node/UID HTTP/1.1" 200 7995
DEBUG: 0.0946: transaction prepared in 0.017, completed in 0.033
INFO: Expanded '*' into 2 nodes.
Nodes
  n0
  n1
DEBUG: Saved authentication cookie instead of logging out.
DEBUG: 0.0959: exiting, waited for 0.033 seconds
```

As can be seen in the above log, the authentication token from a previous run was loaded and used for the duration of command execution and then re-cached for later use.

4.40.1.2 logs/

The command line tools also log their arguments and some execution progress in the `~/ .scylcdcw/logs/` folder. By default each tool keeps logs of its previous five runs, though this number can be adjusted in the `settings.ini` file by resetting the `logging.max_user_logs` value. Set this value to zero to discard all logs, and set to a negative number to preserve logs indefinitely. Administrators may be asked to provide these logs (usually via the `scyld-sysinfo` tool) when requesting assistance from Penguin Computing technical support.

4.40.1.3 workspace/

The `~/ .scylcdcw/workspace/` folder is used by the `scyld-modimg` tool to store, unpack, and manipulate image contents. Root file system images are large, which means this local image cache can grow large. Administrators are encouraged to delete unneeded entries in the cache using the `scyld-modimg --delete` command, either with the `-i` (or `--image`) argument to name specific images, or with `--all` to delete all local images. This will not delete the remote copies of images stored on the head nodes, just delete the local cache. Within this folder, the `manifest.json` file contains JSON formatted information about the cached images, while the images themselves are stored as individual packed files with names based upon their UID. If the cached images are ever out of sync with the manifest, i.e. a file is missing or an extra file is present, then the `scyld-modimg` tool will print a warning:

```
WARNING: Local cache contains inconsistencies. Use --clean-local
to delete temporary files, untracked files, and remove missing
files from the local manifest.
```

This warning can be automatically cleared by running the tool with the `--clean-local` option. This is not done automatically in case some useful image or other data might be lost. Alternatively, if the `manifest.json` is somehow lost, a new file can be constructed for a collection of images using the `--register-all` option. See the command documentation for more details.

The location of the workspace folder can be controlled on the `scyld-modimg` command line or by the `modimg.workspace` variable in the `settings.ini` file.

4.40.1.4 parse_failures/

Several ClusterWare tools execute underlying Linux commands, such as `rpm` or `yum`, and parse their output to check for details of success or failure. During execution and parsing, the `stdout` and `stderr` of the Linux commands are cached in the `~/ .scylcdcw/parse_failures/` folder. If the parsing completes, regardless of the command success or failure, these files will be deleted, but when a tool crashes or parsing fails, these files will be left behind. Though not generally useful to an administrator during normal operation, these output files could be useful for debugging problems and may be requested by Penguin Computing technical support. Much like files in the `~/ .scylcdcw/logs/` folder, these parse failures can be periodically purged if no problems are encountered, though be aware that useful debugging information may be lost.

4.40.2 The `/opt/scyld/clusterware/` Folder

The `/opt/scyld/clusterware` folder exists only on a head node and contains the core ClusterWare installation. Selected contents are described below.

4.40.2.1 `/opt/scyld/clusterware/bin/`

Tools located in the `bin/` folder are intended to be run as root only on head nodes and are rarely executed directly. This is where the `manageddb` tool is located, as well as the `pam_authenticator` application (described in [Configure Administrator Authentication](#) and [Authentication](#)) and the `randomize_ini` script executed during initial installation.

4.40.2.2 /opt/scyld/clusterware/conf/

The `conf/` folder contains the principal configuration files for ClusterWare REST API. In that folder the `httpd_*.conf` files are used in the actual Apache configuration, while the INI files control the behavior of the Python Pyramid-based service. Modifications to any of these files requires the administrator to restart the `clusterware` service. Also note that modifications to these files only affect the one head node and may need to be replicated to other head nodes in multihead configurations. Because of this, future releases may move selected variables from the `base.ini` file into the ClusterWare database to provide a cluster-wide effect.

Many aspects of the REST service can be tweaked through changes to variables in the `base.ini`, and these are discussed throughout this documentation. To list all available variables please use the `managedb` tool:

```
sudo /opt/scyld/clusterware/bin/managedb --print-options
```

This command will list all options registered with the configuration system, and although many of these options are for internal use only, Penguin Computing technical support may suggest changes in individual cases. The specific variables available and their effects may change in future releases.

The variable names take a general form of `SUBSYSTEM.VARIABLE` or `PLUGIN.VARIABLE`. For example, the `plugins` subsystem is controlled through these variables, and a specific authentication plugin is selected by the `plugins.auth` variable. Further, what application the `appauth` plugin uses is controlled by the `appauth.app_path` variable. For a description of this specific plugin, see *Securing the Cluster*. Other variables in the `base.ini` file follow similar patterns.

Variables in the `production.ini` file are used to control aspects of the Python Pyramid framework, specifically logging. Variables in this file are also for internal use and should not be modified except by the suggestion of Penguin Computing technical support.

4.40.2.3 /opt/scyld/env/, modules/, and src/

The `env/`, `modules/`, and `src/` folders contain the Python virtual environment, including the libraries required by the `scyld-*` and other tools.

4.40.2.4 /opt/scyld/clusterware/parse_failures/

Similar to the individual administrator `~/scyldcw/parse_failures/`, files in this folder will accumulate any parsing failures found while running underlying Linux commands and should generally be empty. If files are accumulating here, it is safe to delete them, but the ClusterWare developers should be informed and may request a sample of the files to diagnose the underlying failure.

4.40.2.5 /opt/scyld/clusterware/storage/

The `storage/` folder is the default location used by the `local_files` plugin to store kernels, `initramfs` files, and packed root file systems. The actual location of this folder is controlled by the `local_files.path` variable in the `base.ini` configuration file.

This folder can grow relatively large depending on the size and quantity of root file systems in the cluster. Most organizations will want to include the `storage` folder in their backup planning along with the database contents obtained through `scyld-install --save` or the `managedb save` command. See *Backup and Restore* for additional discussion of backup up the database contents.

4.40.2.6 /opt/scyld/clusterware/workspace/

The REST service running on each head node requires a location to hold temporary files. This location is controlled by the `head.workspace` variable and defaults to `/opt/scyld/clusterware/workspace/`. Like the `storage/` directory, `workspace/` can grow to relatively large size, but unlike `storage/` does not need to be backed up. Any files or directories found in this folder are temporary and should be deleted when the service is shut down or restarted. If files or folders accumulate in this folder, they are safe to remove, although this must be done carefully or when the REST

service is stopped. If files do accumulate here, please notify Penguin Computing developers so that we may diagnose the underlying issue.

4.41 Managing Multiple Head Nodes

The ICE ClusterWare™ platform supports optional active-active(-active....) configurations of multiple cooperating head nodes that share a single replicated database. Such multi-headnode configurations allow any head node to provide services for any compute node in the cluster. These services include cluster configuration using `scyld-*` tools, compute node booting and power control, and compute node status collection.

The ClusterWare `etcd` database requires a minimum of three cooperating head nodes to support full High Availability ("HA") in the event of head node failures. The `etcd` HA works in a limited manner with just two head nodes. The ClusterWare command-line tools for head node management are intended to cover the majority of common cases.

4.41.1 Adding a Head Node

After installing the first head node as described in *Install ICE ClusterWare*, additional head nodes can be installed and joined with the other cooperating head nodes using the same `scyld-install` tool or using `curl`.

On an existing head node view its database password:

```
sudo grep database.admin_pass /opt/scyld/clusterware/conf/base.ini
```

4.41.1.1 Join a non-ClusterWare server

A non-ClusterWare server can use `scyld-install` to join another head node (identified by its IP address `IP_HEAD`) that may itself already be joined to other head nodes. You can download `scyld-install` from the Penguin repo <https://updates.penguincomputing.com/clusterware/12/installer/scyld-install> or <https://updates.penguincomputing.com/clusterware/12/installer-el8/scyld-install> or <https://updates.penguincomputing.com/clusterware/12/installer-el9/scyld-install> without needing a cluster ID, or if you already have a `/etc/yum.repos.d/clusterware.repo` installed, then you can download the `clusterware-installer` package, which includes `scyld-install`. Then:

```
scyld-install --database-passwd <DBPASS> --join <IP_HEAD>
```

where `DBPASS` is `IP_HEAD`'s database password, as described above. If no `--database-passwd` is provided as an argument, then `scyld-install` queries the administrator interactively for `IP_HEAD`'s database password.

`scyld-install` doing a join will install the ClusterWare platform using the same `clusterware.repo` and database type being used by the head node at `IP_HEAD`.

A cluster configuration file is not required when joining a server to a head node because those settings are obtained from the existing head node's cluster database after the join successfully completes.

4.41.1.2 Join a ClusterWare head node

A "solo" ClusterWare head node can use `scyld-install` to join another head node (identified by its IP address `IP_HEAD`) that may itself already be joined to other head nodes.

Important

The join action discards the "solo" head node's current images and boot configs, then finishes leaving the "solo" head node with access to just the cooperating head nodes' images and boot configs. If you want to save any images or configs, then first use `scyld-bootctl export` (*Exporting and Importing Boot Configurations Between Clusters*) or `managedb save`.

For example, to join a ClusterWare head node:

```
scyld-install --update --database-passwd <DBPASS> --join <IP_HEAD>
```

When a ClusterWare 12 head node joins another ClusterWare 12 head node, `scyld-install` performs a mandatory update of the current ClusterWare software using `IP_HEAD`'s `clusterware.repo` prior to joining that `IP_HEAD`. This ensures that ClusterWare (and `scyld-install`) are executing compatible ClusterWare software.

However, the ClusterWare 11 version of `scyld-install` will **not** automatically perform this mandatory update of 11 to 12 and will just update the joining head node to the newest version of ClusterWare 11. Penguin Computing recommends first updating the ClusterWare 11 head node to 12 (following the guidance of [Updating ClusterWare 11 to ClusterWare 12](#)), and then using the ClusterWare 12 `scyld-install` to perform the join.

Just as when joining a non-ClusterWare server, if no `--database-passwd` is provided as an argument, then `scyld-install` queries the administrator interactively for `IP_HEAD`'s database password.

4.41.1.3 After a Join

Important

Every head node must know the hostname and IP address of every other head node, either by having those hostnames in each head node's `/etc/hosts` or by having their common DNS server know all the hostnames. Additionally, if using head nodes as default routes for the compute nodes, as described in [Configure IP Forwarding](#), then ensure that all head nodes are configured to forward IP traffic preferably over the same routes.

Important

Every head node should use a common network time-sync protocol. The Red Hat RHEL default is `chronyd` (found in the `chrony` package), although `ntpd` (found in the `ntp` package) continues to be available.

Subsequent head node software updates are also accomplished by executing `scyld-install -u`. We recommend that all cooperating head nodes update to a common ClusterWare release. In rare circumstance a newer ClusterWare release on the head nodes also requires a compatible newer `clusterware-node` package in each compute node image. Such a rare coordinated update will be documented in the [Release Notes](#) and [Changelog](#).

4.41.1.4 Cleaning up From Join Failures

The `scyld-install --join <IP>` command ensures that the `database.admin_pass` variable is properly set in `/opt/scyld/clusterware/conf/base.ini` and logs errors into `~/scyldcw/logs/install_<TIMESTAMP>.log`. If the join fails, check that log file for details. Please report those errors to Penguin, but there are several approaches to retrying the join.

Before retrying the join, the cluster administrator should ensure that the existing cluster has not been negatively affected by the failed join. Appropriate recovery depends on the initial number of head nodes in the cluster.

For a single head node:

The HA mechanisms expect at least 3 head nodes, so if joining fails while adding the second head node to the cluster, then `etcd` on the initial head node may be left in a bad state. Check for this by running:

```
/opt/scyld/clusterware/bin/managedb --heads
```

If that command results in an error then the existing head node requires recovery via:


```
/opt/scyld/clusterware/bin/managedb recover
```

Once recovery has completed then `managedb --heads` will show a single head node and `scyld-*ctl` commands will work as expected. At this point the other head nodes can be joined.

When joining to a cluster with more than one head node:

The complete join process consists of two stages. In the first stage, the joining head node reaches out to the existing head node to retrieve the etcd peer URLs and adds itself as an etcd member. In the second stage, the joining head node reconfigures its local etcd instance to communicate with the existing etcd instances.

If a failure happens during the first stage, existing head nodes will not be modified and the join can be retried without additional interventions. If a failure occurs during the second stage then running `managedb --heads` on the existing heads will show the joining head node and that remnant needs to be ejected before proceeding. To do that run a command on an existing head node:

```
/opt/scyld/clusterware/bin/managedb eject <JOINING_HEAD_IP>
```

After cleaning:

Once the appropriate steps are complete the cluster administrator can retry the head node join process. Possible reasons for join failure are:

Networking issues

- If the head nodes cannot reach each other at all then we expect a failure in stage one and the join can be retried once the issue is resolved.
- If the head nodes can reach the API but not the etcd port (default 52380), usually due to a firewall configuration, then we expect a failure in the second stage and cleanup will be required before retrying.

Duplicate head node UIDs

- If a head node is created by cloning an existing VM and the cluster administrator does not replace the `head.uid` in the `base.ini`, we expect a failure in the second stage and cleanup is required before fixing the problem and retrying.

If the joining process fails multiple times and the joining head node was previously a member of this or some other cluster, it can be useful to provide a `--purge` argument to the join process. This causes the joining head node to completely remove any local database content before attempting to join the cluster:

```
/opt/scyld/clusterware/bin/managedb join -purge <IP>
```

4.41.2 Removing a Joined Head Node

A list of connected head nodes can be seen with:

```
sudo /opt/scyld/clusterware/bin/managedb --heads
```

Head nodes also store status information in the ClusterWare database. That content can be viewed via:

```
scyld-clusterctl heads ls -l
```

Sometimes a cluster administrator will need to temporarily or permanently remove a head node from the cluster during the course of an RMA, upgrade, or other maintenance task. For a cluster with three or more head nodes you can remove one of the head nodes by running `managedb leave` on that head node:

```
sudo /opt/scyld/clusterware/bin/managedb leave
```


Or if that head node is shut down or otherwise unavailable, then it can be ejected by another head node in the cluster by running:

```
sudo /opt/scyld/clusterware/bin/managedb eject <IP_HEAD_TO_REMOVE>
```

Note that this command will attempt to stop some services on the targeted head node and that step may fail. That does not mean that the eject has necessarily failed. The now-detached head node will no longer have access to the shared database and will be unable to execute any `scyld-*` command, as those require a database. Either re-join the previous cluster:

```
sudo /opt/scyld/clusterware/bin/managedb join <IP_HEAD>
```

or join another cluster after updating the local `/opt/scyld/clusterware/conf/base.ini` `database.admin_pass` to the other cluster's database password and then joining to a head node in that other cluster. After joining a cluster by directly using the `managedb join` command the `clusterware` service should be restart on the joining head node.

The node can also be wiped and reinstalled:

```
scyld-install --clear-all --config <CLUSTER_CONFIG>
```

However, for a cluster with only **two** head nodes you cannot `managedb eject` or `managedb leave`, and instead must execute `managedb recover`, thereby "ejecting" the **other** head node:

```
sudo /opt/scyld/clusterware/bin/managedb recover
```

This command will reduce the head node cluster to a single head node, specifically the one where that command is executed, severing the connection to the shared database. If this command is executed on one of three or more head nodes the remaining head nodes will continue to operate as an independent cluster.

Important

Keep in mind that if `managedb recover` is run on both head nodes then both head nodes will have their own copies of the now-severed database that manages the same set of compute nodes, which means that both will compete for "ownership" of the same booting compute nodes.

To avoid both head nodes competing for the same compute nodes, either execute `sudo systemctl stop clusterware` on one of the head nodes, or perform one of the steps described above to re-join this head node to the other head node that previously shared the same database, or join another head node, or perform a fresh ClusterWare install.

4.41.2.1 Peer Downloads

Multi-head clusters replicate data between head nodes on local storage to preserve a copy of each uploaded or requested file. The storage location is defined in `/opt/scyld/clusterware/conf/base.ini` by the `local_files.path` variable, and it defaults to `/opt/scyld/clusterware/storage/`.

Whenever a ClusterWare head node is asked for a file such as a kernel, the expected file size and checksum are retrieved from the database. If the file exists in local storage and has the correct size and checksum, then that local file will be provided. However, if the file is missing or incorrect, then the head node attempts to retrieve the correct file from a peer.

Note that local files whose checksums do not match will be renamed with a `.old.NN` extension, where `NN` starts at `00` and increases up to `99` with each successive bad file. This ensures that in the unlikely event that the checksum in the database is somehow corrupted, the original file can be manually restored.

Peer downloading consists of the requesting head node retrieving the list of all head nodes from the database and contacting each in turn in random order. The first peer that confirms that it has a file with the correct size provides that file to the requesting head node. The checksum is computed during the transfer, and the transferred file is discarded if that checksum is incorrect. Contacted peers will themselves not attempt to download the file from other peers in order to avoid having a completely missing file trigger a cascade.

After a successful peer download, the original requester receives the file contents after a delay due to the peer download process. If the file cannot be retrieved from any head node, then the original requester will receive a *HTTP 404* error.

This peer download process can be bypassed by providing shared storage among head nodes. Such storage should either be mounted at the storage directory location prior to installation, or the `/opt/scyld/clusterware/conf/base.ini` should be updated with the non-default pathname immediately after installation of each head node. Remember to restart the *clusterware* service after modifying the `base.ini` file by executing `sudo systemctl restart clusterware`, and note that the `systemd clusterware.service` is currently an alias for the `httpd.service`.

When a boot configuration or image is deleted from the cluster, the deleting head node will remove the underlying file(s) from its local storage. That head node will also temporarily move the file's database entry into a deleted files list that other head nodes periodically check and delete matching files from their own local storage. If the *clusterware* service is not running on a head node when a file is marked as deleted, then that head node will not be able to delete the local copy. When the service is later restarted, it will see its local file is now no longer referenced by the database and will rename it with the `.old.NN` extension described earlier. This is done to inform the administrator that these files are not being used and can be removed, although cautious administrators may wish to keep these renamed files until they confirm all node images and boot configurations are working as expected.

4.41.3 Booting With Multiple Head Nodes

Since all head nodes are connected to the same private cluster network, any compute node's DHCP request will receive offers from all the head nodes. All offers will contain the same IP address by virtue of the fact that all head nodes share the same MAC-to-IP and node index information in the replicated database. The PXE client on the node accepts one of the DHCP offers, which is usually the first received, and proceeds to boot with the offering head node as its "parent head node". This parent head node provides the kernel and `initramfs` files during the PXE process, and provides the root file system for the booting node, all of which should also be replicated in `/opt/scyld/clusterware/storage/` (or in the alternative non-default location specified in `/opt/scyld/clusterware/conf/base.ini`).

On a given head node you can determine the compute nodes for which it is the parent by examining the head node `/var/log/clusterware/head_*` or `/var/log/clusterware/api_error_log*` files for lines containing "Boot-ing node". On a given compute node you can determine its parent by examining the node's `/etc/hosts` entry for `parent-head-node`.

Once a node boots, it asks its parent head node for a complete list of head nodes, and then thereafter the node sends periodic status information to its parent head node at the top of the list. If at any point that parent head node does not respond to the compute node's status update, then the compute node chooses a new parent by rotating its list of available head nodes by moving the unresponsive parent to the bottom of the list and moving the second node in the list up to the top of the list as the new parent.

The administrator can force compute nodes to re-download the head node list by executing `scyld-nodectl script fetch_hosts` and specifying one or more compute nodes. The administrator can also refresh the SSH keys on the compute node using `scyld-nodectl script update_keys`.

Clusters of 100 nodes or more benefit from having each head node being a parent to roughly the same number of compute nodes. Each head node periodically computes the current mean number of nodes per head, and if a head node parents significantly more (e.g., >20%) nodes than the mean, then the head node triggers some of its nodes to use another head node. Care is taken to avoid unnecessary shuffling of compute nodes. The use of the `_preferred_head` attribute may create an imbalance that this rebalancing cannot remedy.

4.42 headctl

NAME

headctl -- Manage head node network communication settings.

USAGE

headctl

```
[-h | --help] [--status] [--prefer-http] [--prefer-https] [--enable-https]
[--disable-https] [--enable-xsendfile] [--disable-xsendfile]
```

DESCRIPTION

This is a low-level tool that directly manipulates configuration settings for head node network communication. When controlling HTTP/HTTPS settings, it modifies `/opt/scyld/clusterware/conf/base.ini` and two Apache configuration files in `/etc/http/conf.d/`: `ssl.conf` and `clusterware.conf`. When enabling XSendfile support, the tool may install necessary RPMs as well as update variables in the `base.ini`.

Since the earliest boot steps cannot use encrypted communications, DHCP and PXE booting are not affected by these settings. Communications starting with `intrafms` execution will use HTTP or HTTPS as instructed by this command.

The tool resides in `/opt/scyld/clusterware/bin/headctl` and must be executed by user `root`.

OPTIONAL ARGUMENTS

-h, --help	Print usage message and exit. Ignore trailing args, parse and ignore preceding args.
--status	Report the configuration of the ClusterWare service.
--prefer-http	Instruct compute nodes to use HTTP.
--prefer-https	Instruct compute nodes to use HTTPS where possible.
--enable-https	Proxy through <code>/etc/http/conf.d/ssl.conf</code> .
--disable-https	Do not proxy through <code>/etc/http/conf.d/ssl.conf</code> .
--minimal-ipv6	Confirm the <code>radvd</code> is running on our interfaces by optionally installing <code>radvd</code> , adding blocks to the <code>radvd.conf</code> , and restarting the service if necessary.
--enable-xsendfile	Use the Apache XSendfile header when clients download files.
--disable-xsendfile	Do not use the Apache XSendfile header.

EXAMPLES

RETURN VALUES

Upon successful completion, **headctl** returns 0. On failure, an error message is printed to `stderr` and **headctl** returns 1.

4.43 Troubleshooting Head Nodes

4.43.1 Head Node Filesystem Is 100% Full

If a head node filesystem(s) that contains ICE ClusterWare™ data (typically the root filesystem) is 100% full, then the administrator cannot execute `scyld-*` commands and ClusterWare cluster operations will fail.

4.43.1.1 Verify Excessive Storage is Related to ClusterWare Software

First determine whether or not the problem is due to ClusterWare-related files. Investigate with:

```
sudo du -sh /opt/* ; sudo du -sh /opt/scyld/*
sudo du -sh /var/lib/*

# For each cluster administrator with a home directory on local storage:
sudo du -sh /home/*/.scyldcw/workspace
```

and look for excessive storage consumption. If the ClusterWare software is not the problematic consumer, then broaden the search across the filesystem(s) for non-ClusterWare storage that can be reduced.

For ClusterWare storage, continue:

4.43.1.2 Remove Unnecessary Objects from the ClusterWare Database

Remove any unnecessary objects in the database that may be lingering after an earlier aborted operation:

```
sudo systemctl stop clusterware
sudo rm /opt/scyld/clusterware/storage/*.old.00
sudo systemctl start clusterware
```

If that does not release enough space to allow the `scyld-*` commands to execute, then delete the entire local cache of database objects:

```
sudo systemctl stop clusterware
sudo rm -fr /opt/scyld/clusterware/workspace/*
sudo systemctl start clusterware
```

4.43.1.3 Investigate InfluxDB Retention of Telegraf Data

If you continue to see `influxdb` messages in `/var/log/messages` that complain "no space left on device", or if the size of the `/var/lib/influxdb/` directory is excessively large, then InfluxDB may be retaining too much Telegraf time series data, aka *shards*. Examine with:

```
sudo systemctl restart influxdb

# View the summation of all the Telegraf shards
sudo du -sh /var/lib/influxdb/data/telegraf/autogen/

# View the space consumed by each Telegraf shard
sudo du -sh /var/lib/influxdb/data/telegraf/autogen/*
```

If the `autogen` directory or any particular `autogen` subdirectory *shard* consumes a suspiciously large amount of storage, then examine the retention policy with the `influx` tool:

```
sudo influx
```

and now within the interactive tool you can execute `influx` commands:

```
> show retention policies on telegraf
```

The current ClusterWare defaults are a *duration* of 168h0m0s (save seven *shards* of Telegraf data) and a *shardGroup-Duration* of 24h0m0s (each spanning one 24-hour day). You can reduce the current retention policy, if that makes sense for your cluster, with simple command. For example, reduce the above 7-shard *duration* to five, thereby reducing the number of saved *shards* by two:

```
> alter retention policy "autogen" on "telegraf" duration 5d
```

You can also delete individual unneeded *shards*. View the *shards* and their timestamps:

```
> show shards
```

and selectively delete any unneeded *shard* using its *id* number, which is found in the `show` output's first column:

```
> drop shard <shard-id>
```

When finished, exit the `influx` tool with:

```
> exit
```

See <https://docs.influxdata.com/influxdb/v1.8/> for more documentation.

4.43.1.4 Remove Unnecessary Images and Repos

If `scyld-*` commands can now execute, then view information for all images and repos, including their sizes:

```
scyld-imgctl ls -l
scyld-clusterctl repos ls -l
```

Consider selectively deleting unneeded images with:

```
scyld-imgctl -i <imageName> rm
```

and consider selectively deleting unneeded repos with:

```
scyld-clusterctl repos -i <repoName> rm
```

4.43.1.5 Move Large Directories

If `scyld-*` commands still cannot execute, and if your cluster really does need all its existing images, boot configs, *telegraf* history, and other non-ClusterWare filesystem data, then consider moving extraordinarily large directories (e.g., `/opt/scyld/clusterware/workspace/`, as specified in `/opt/scyld/clusterware/conf/base.ini`) to another filesystem or even to another server, and/or add storage space to the appropriate filesystem(s).

4.43.2 Head Nodes Disagree About Compute Node State

If two linked head nodes disagree about the status of the compute nodes, this is usually due to clock skew between the head nodes. The appropriate fix is to ensure that all head nodes are using the same NTP / Chrony servers. The shared database includes the last time each compute node provided a status update. If that time is too far in the past, then a compute node is assumed to have stopped communicating and is marked as "down". This mark is *not* recorded in the database, but is instead applied as the data is returned to the calling process such as `scyld-nodectl status`.

4.43.2.1 Head Node Failure

To avoid issues like an Out-Of-Memory condition or similarly preventable failure, head nodes should generally not participate in the computations executing on the compute cluster. As a head node plays an important management role, its failure, although rare, has the potential to impact significantly more of the cluster than the failure of individual compute nodes. One common strategy for reducing the impact of a head node failure is to employ multiple head nodes in the cluster. See *Managing Multiple Head Nodes* for details.

4.43.3 etcd Database Exceeds Size Limit

The etcd database has a hard limit of 2GB. If exceeded, then all `scyld-*` commands fail and `/var/log/clusterware/api_error_log` will commonly grow in size as each node's incoming status message cannot be serviced. The ClusterWare `api_error_log` may also contain the following text:

```
etcdserver: mvcc: database space exceeded
```

Normally a head node thread executes in the background that triggers the discarding of database history (called *compaction*) and triggers database defragmentation (called *defrag*) if that is deemed necessary. In the rare event that this thread stops executing, then the etcd database grows until its size limit is reached.

This problem can be solved with manual intervention by an administrator. Determine if the etcd database really does exceed its limit. For example:

```
[admin@head1]$ sudo du -hs /opt/scyld/clusterware-etcd/
2.1G    /opt/scyld/clusterware-etcd
```

shows a size larger than 2GB, so you can proceed with the manual intervention.

First determine the current database revision. For example:

```
[admin@head1]$ sudo /opt/scyld/clusterware-etcd/bin/etcdctl get --write-out=json does_
↪not_exist
{"header":{"cluster_id":9938544529041691203,"member_id":10295069852257988966,"revision
↪":4752785,"raft_term":7}}
```

Subtract two or three thousand from the *revision* value 4752785 and compact to that new value:

```
[admin@head1]$ sudo /opt/scyld/clusterware-etcd/bin/etcdctl compaction 4750000
compacted revision 4750000
```

and trigger a defragmentation to reclaim space:

```
[admin@head1]$ sudo /opt/scyld/clusterware-etcd/bin/etcdctl defrag
Finished defragmenting etcd member[http://localhost:52379]
```

Then clear the alarm and reload the *clusterware* service:

```
[admin@head1]$ sudo /opt/scyld/clusterware-etcd/bin/etcdctl alarm disarm
[admin@head1]$ sudo systemctl reload clusterware
```

This restarts the head node thread that executes in the background and checks the etcd database size. Everything should now function normally.

4.44 Networks Page

If your cluster has multiple networks, the ICE ClusterWare™ platform initially creates a network record for only the network that was used during installation. You can add other networks to manage DHCP/DNS entries on ClusterWare nodes for those networks as well.

Use the **Networks** page to define available node IPs, what interfaces the DHCP server listens on, and parameters used during the early compute node boot process. The page is available via **Network > Networks** in the left navigation panel.

Networks

Refresh Interval (seconds):

Networks define available node IPs, what interfaces the DHCP server will listen on, and parameters used during the early boot process.

Name	First IP / Mask bits	IP count	Router	Gateway	Domain
10.54.2.1/16	10.54.2.1/16	200	10.54.1.1		cluster.local
192.168.122.1/24	192.168.122.1/24	1			cluster.local

ADD NETWORK

4.44.1 Create a Network

To create a network:

1. Click **Add Network**.
2. Add details about the network.
 - **Name:** Required.
 - **Description:** Optional.
 - **First IP:** Required.
 - **Mask Bits:** Required.
 - **IP Count:** Required.
 - **First Index:** Optional.
 - **Domain:** Optional.
 - **Router IP:** Optional.
 - **Gateway IP:** Optional.
 - **Node Interface:** Optional.

3. Click **Add Network** to save your changes.

The new network appears in the list at the top of the page.

4.44.2 Edit Network

To edit a network:

1. Click the network name to open the edit panel for that network.
2. Make updates to the network.
3. Click **Edit Network** to save your changes.

4.44.3 Delete Network

To delete a network, click the ellipsis (...) on the far right of the row and select the **Delete** action.

4.44.4 Related Links

- *scyld-clusterctl*

4.45 Open Network Ports

The ICE ClusterWare™ platform needs a number of network ports to be open so that critical services can be reached. Which ports are needed depends on the function of a given node. Head nodes serve out many of the cluster infrastructure services, like DNS and DHCP, and hence need a number of ports open. Login or management nodes need fewer ports open since, while admins can run ClusterWare commands on those nodes, they do not host any services themselves. Compute nodes also do not host any services and thus need few ports to be open.

Note

MPI and other communication libraries may have additional requirements on open network ports; these requirements will be highly application-specific, please refer to the vendor's documentation for more information.

It is often convenient to simply open up the internal cluster network to allow all traffic so that compute nodes and heads can easily "talk" to each other. One method is to create a "rich rule" for the cluster subnet that allows all traffic:

```
firewall-cmd --permanent --zone=public \
  --add-rich-rule='rule family=ipv4 source address=192.168.100.0/24 accept'
```

For clusters with multiple networks, multiple "rich rules" will need to be created.

In more secure environments, admins may want to lock down the network more tightly. The following table shows what ports need to be open on different "types" of nodes. "Open" indicates that the port is required to be open for proper functioning of the cluster. "Opt" indicates a port that may be open or blocked, but note that if it is blocked, then that service may not function fully. As an example, the Chrony tool uses port 123 to keep the system's time in sync; but port 323 is only needed if admins want to do further control of Chrony through the *chronyc* command-line tool.

Service	Port	HEAD	LOGIN	NODE
Apache (httpd)	80 ¹	open		
	443	open		
Chrony	123	open		
	323 ²	opt	opt	opt
DHCP	68	open		
DNS	53	open		
etcd	52380	open		
Grafana	52391 ³	local		
InfluxDB	8086 ⁴	local	opt	
iSCSI	3260 ⁵	opt		
SSH	22	open	open	open
Telegraf	8094	open		
TFTP	69	open		
Slurm	3306 ⁶	open		
	6817 ⁷	open	open	open
	6818 ⁸	open		open
	6819 ⁹	open		

Footnotes

- [1] Use of the insecure HTTP protocol on port 80 is deprecated; admins should switch to HTTPS on port 443.
- [2] Port 323 is used by the ``chronyc`` command-line tool and by the ``chrony`` status plugin; if those tools are not being used, then the port can be disabled. When used, the traffic should only be on localhost.
- [3] The Grafana port must be open for local traffic on the heads.
- [4] The InfluxDB port must be open for local traffic on the heads. On heads and login nodes, it can be opened for debugging or low-level access to InfluxDB.
- [5] iSCSI support is optional; if iSCSI booting is not being used, then the port can be disabled.
- [6] Slurm is an optional package; port 3306 is used by ``slurmdbd`` to talk to the SQL database which generally resides on the same host.
- [7] Slurm is an optional package; port 6817 is used by various tools to communicate with ``slurmctld`` on the Slurm controller node.
- [8] Slurm is an optional package; port 6818 is used by ``slurmd`` on the compute-nodes to receive job information from the ``slurmctld`` on the Slurm controller node.
- [9] Slurm is an optional package; port 6819 is used by ``slurmdbd`` to talk to ``slurmctld`` which generally resides on the same host.

4.46 Providing DHCP to Additional Interfaces

Some specialized cluster configurations require additional network-related ICE ClusterWare™ node attributes. For example, when booting compute nodes that are configured to use a bonded administrative interface, it can be difficult to predict what physical interface will successfully send DHCP requests.

In most clusters the compute node BMC addresses are statically assigned to ensure that the BMCs are always accessible regardless of the state of other infrastructure. However, there are times when dynamically assigning reserved addresses to the BMC interfaces is useful.

The ClusterWare platform supports dynamic assignment via the following *Reserved Attributes*:

- `_altmacs`
- `_ips` with a special `bmc=` entry
- `_macs` with a special `bmc=` entry

For example, a node has a primary MAC address of `aa:bb:cc:dd:ee:f0`, a primary IP address of `10.54.0.100`, and the following attributes defined:

```
_altmacs=aa:bb:cc:dd:ee:f1,aa:bb:cc:dd:ee:f2
_ips=bmc=10.10.10.100,ib0=10.55.10.100
_macs=bmc=11:22:33:44:55:66
```

The ClusterWare DHCP subsystem sees the `_altmacs` attribute and creates reservations of the `10.54.0.100` IP address for MAC addresses `aa:bb:cc:dd:ee:f0` (the primary MAC address), `aa:bb:cc:dd:ee:f1`, and `aa:bb:cc:dd:ee:f2`. Requests from the physical network adapters with any of those MAC addresses receive the same `10.54.0.100` IP address. Nodes

should proceed with the boot process once any network adapter successfully receives the IP address and that network adapter is marked as the bootnet.

Additionally, while examining the values of the `_ips` and `_macs` attributes, the system will find the `bmc=` sections and create a reservation for the 10.10.10.100 IP address to the 11:22:33:44:55:66 MAC address. If the node BMC is configured to dynamically request an address via DHCP and the BMC network is connected to the head node, then the BMC receives that reserved IP address.

The `ib0=10.55.10.100` section of the `_ips` attribute does not result in any DHCP reservations, but is substituted into an existing `ifcfg-ib0` file within the image at boot time. This is done by `preboot` scripts within the image provided by the `clusterware-node` package. The scripts perform similar replacements in NetworkManager connection files on newer operating systems or Netplan configuration files on Ubuntu.

4.47 Exceeding System Limit of Network Connections

Clusters with a large number of nodes (e.g., many hundreds or more) may observe a problem when executing a workload that attempts to communicate concurrently with many or most of the nodes, such as `scylld-nodectl --up exec` or `mpirun` executing a multi-threaded, multi-node application. The problem exhibits itself with an error message that refers to being unable to allocate a TCP/IP socket or network connection, or `arp_cache` reporting a "neighbor table overflow!" error.

A possible solution is to increase the number of available "neighbor" entries. These are managed by a coordinated increase of `gc_thresh1`, `gc_thresh2`, and `gc_thresh3` values. See <https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt> for the semantics of these variables. See the current values with:

```
sysctl net.ipv4.neigh.default.gc_thresh1
sysctl net.ipv4.neigh.default.gc_thresh2
sysctl net.ipv4.neigh.default.gc_thresh3
```

Default CentOS/RHEL values are 128, 512, and 1024, respectively. Experiment with higher values until your workloads are all successful. For example:

```
sudo sysctl -w net.ipv4.neigh.default.gc_thresh1=2048
sudo sysctl -w net.ipv4.neigh.default.gc_thresh2=4096
sudo sysctl -w net.ipv4.neigh.default.gc_thresh3=8192
```

See `man sysctl.conf` for how to make the successful values persistent across a reboot by putting them in a new `/etc/sysctl.d/` file.

4.48 Managing Zero-Touch Provisioning (ZTP)

Important

Currently only supported for Cumulus switches.

The ICE ClusterWare™ platform supports ZTP (Zero-Touch Provisioning) of ONIE and related switches. Note that ZTP by itself does not provide a full, end-to-end control plane for cluster networking, but it is the first step in that direction, allowing for server-provided scripts to alter the configuration of connected switches.

Since the ZTP-capable switches are essentially Linux management systems attached to the switches, the ClusterWare platform treats them as another node in the cluster. You can add them to the cluster using `scylld-nodectl create` and specifying the switch's MAC address. For example:

```
scyld-nodectl create mac=aa:bb:cc:00:11:22
```

which simplistically creates a new (switch) node in the default naming-pool and default group. This may not be the desirable approach, since it assigns a generic name like "n12" which is superficially indistinguishable from compute nodes "n0" through "n11". A better approach is to utilize the ClusterWare naming-pool and attribute-group functionality to assign a more self-identifying name and permit more efficient management of this and other ZTP-capable switches:

```
scyld-clusterctl pools create name=ztpswitch pattern="switch{}"  
scyld-nodectl create mac=aa:bb:cc:00:11:22 naming_pool=ztpswitch
```

which creates a new naming pool "ztpswitch" and configures the new node inside that pool with the name "switch0". Subsequent ZTP-capable switches can use the same naming-pool, which names them "switch1", "switch2", etc.

The cluster administrator can then use:

```
scyld-nodectl -i switch2 <action>  
scyld-nodectl -i switch* <action2>
```

to perform an action on a specific switch or a common action on all switches in that naming-pool.

Configure each ZTP node to boot using a ZTP boot script. A boot script may be written in Bash or Python. As with other scripts, the first line should be `#!/path/to/interpreter`, e.g. `#!/bin/bash`. Some switches also allow Perl, Ruby, or a vendor-specific language. These scripts execute as user *root* on the switch and can execute commands supported by the switch, including triggering Puppet or Ansible runs, downloading files via `wget` or `curl` and manipulating or moving them on the switch, and more. After a successful execution, the script **must** return status 0.

ZTP boot scripts reside in `/opt/scyld/clusterware/kickstarts/`. Configure the boot script `ztp_config.sh` for the node `switch0` using the specific prefix "ztp":

```
scyld-nodectl -i switch0 set _boot_config="ztp:ztp_config.sh"
```

Since switch nodes are ClusterWare nodes, you can use attribute groups to configure this as well:

```
scyld-attribctl create name=ZtpSwitches  
scyld-attribctl -i ZtpSwitches set _boot_config="ztp:ztp_config.sh"  
scyld-nodectl -i switch0 join ZtpSwitches
```

which creates an attribute group "ZtpSwitches" and joins "switch0" into it. All members of that attribute group will boot the same `ztp_config.sh` script.

In a multi-headnode cluster, every head node should have the same ZTP boot script installed. Currently this must be done manually.

At boot time the ZTP-enabled node `switch0` executes a DHCP query. The server sees the query, identifies the node using the client's MAC address in the DHCP request, recognizes the client as a ZTP-enabled node and the node's `_boot_config`'s "ztp:", then builds a DHCP response that includes a URL of the form `http://*<SERVER_IPADDR>*/boot/ztp_config.sh`. The switch then uses standard web protocols to read the URL to download the script and execute it.

Per the Cumulus Linux guidelines, the script **must** include the phrase "CUMULUS-AUTOPROVISIONING", usually in a comment, in order to execute at ZTP boot. Other switch or NOS vendors may require similar keywords.

While the system may provide some limited logging that the ZTP script was run, it may make sense to log any/all command outputs to a known file for easier debugging and triage. A line such as `exec >> /var/log/autoprovision 2>&1` in a bash script writes output to that log file for subsequent commands in the script.

Once a ZTP-switch has been successfully configured and the script returns status 0, it will **not** execute the ZTP boot script again, not even at the next reboot of the switch node. To force the switch to re-execute the boot script on the next

reboot, ssh to the switch and execute `sudo ztp --reset`.

The following articles provide instructions for advanced ICE ClusterWare™ cluster configurations or scenarios.

5.1 ICE ClusterWare Plugin System

The ICE ClusterWare™ Plugin System allows administrators to more quickly change the status and monitoring system across the entire cluster or on subsets of nodes.

There are 4 types of plugins:

- *Status Plugins*
 - These default to an update every 10 sec, so these are generally sensors or readings that change somewhat frequently; e.g. the free RAM on a node, or the current CPU load;
- *Hardware Plugins*
 - Called less often than “regular” status plugins, usually every 300 sec, these are sensors and readings that change less frequently and/or are tied to the hardware itself; e.g. the total RAM on a node, or the CPU architecture;
- *Health-Check Plugins*
 - Called less often than “regular” status plugins, usually every 300 sec, these are sensors and readings that can be thought of as answering the question “is this compute node operating correctly?”; the values will usually be “healthy” or “unhealthy”, but may include a time-stamp (indicating that the plugin is still calculating the value) or a longer message such as “unhealthy; some text on why the node is unhealthy”;
- *Telegraf Plugins*
 - These are really smaller, more granular Telegraf config files that the ClusterWare platform can individually enable/disable.

If admins know that they want some plugins permanently enabled, they can build those plugins into the disk images that the node boots from. These “built-in” plugins are always enabled and they cannot be disabled later except by changing the disk image.

For information that may only be needed some of the time, admins can add arbitrary plugins to a node using the `_status_plugins` list (with similar attributes for hardware, health, and telegraf). These on-the-fly plugins can be turned on and off at any time simply by setting, overwriting, or clearing that node attribute. E.g.:

```
scylld-nodectl -all set _status_plugins=chrony,ipmi
```

would enable the `chrony` and `ipmi` status plugins. While:

```
scylld-nodectl -all set _status_plugins=chrony
```

would keep `chrony` enabled but disable `ipmi` (since it is not listed anymore).

Best Practices

While the ability to enable/disable plugins in an ad-hoc fashion can be powerful, basic best practices still hold:

- It may be helpful to consider the ClusterWare software as “the management tool” and Telegraf as “the monitoring tool”:
 - Information which you may want to *take action* on should be included in the status, hardware, or health updates, and should be permanently enabled in `scripts-enabled` so that they are available when that action is needed later on;
 - Information that might be useful for *long-term analyses and trends* should be stored in Telegraf.
- Frequent changes to plugins may make the underlying data less useful. If a parameter exists at some times and not at others, it will be difficult to make future decisions based on any changes seen in that parameter.
- Any data that will be viewed through Grafana should be built into the image (in `telegraf-enabled`), otherwise the data may not exist and any Grafana dashboards may produce empty charts.
- For more security-conscious admins, any security-relevant plugins should be enabled in the `scripts-enabled` or `telegraf-enabled` directories, making them permanently enabled and more resistant to tampering.

For more information:

5.1.1 Status Plugins

Status plugins are used to report useful information about the compute node back to the server, e.g. CPU load, memory usage, available disk space, etc. This may be used for simple status monitoring to get an overall sense of how loaded the cluster is, for example:

```
scyld-nodectl --all status -L
```

And since this data is stored in the ICE ClusterWare™ database, it can also be used to target actions against groups of nodes:

```
scyld-nodectl -s "status[ram_free] < 1GB" ls
```

This would select (`-s`) all nodes with less than 1GB of free memory and then call the “`ls`” command on those nodes (i.e. listing those nodes); any other node command could also be executed here (power on/off, reboot, etc.).

Default frequency is data collection every 10 seconds but this may be overridden on a node-by-node basis with attribute `_status_secs`.

For larger-scale management and control, one can set the `_status_plugins` and/or `_status_secs` attributes inside an attribute group and then join nodes to that group.

Building Status Plugins into an Image

The current approach creates new directories in the `clusterware-node` package that must be installed on all nodes: `scripts-available/status` and `scripts-enabled/status`. The `scripts-available/status` directory is populated with Penguin-provided scripts that will provide useful status information in a variety of categories. An admin can copy or symlink those scripts into `scripts-enabled/status` to permanently add them to the image.

For example, to include the `timedatectl` plugin to the image:

```
% scyld-modimg -iNewImage -chroot -upload -overwrite
Downloading and unpacking image f21b65b1.....0aafef663
 100.0% complete, elapsed: 0:00:02.2 remaining: 0:00:00.0
elapsed: 0:00:06.0
```

(continues on next page)

(continued from previous page)

```
Executing step: Chroot
Dropping into a /bin/bash shell. Exit when done.
[CW:NewImage /]# cd /opt/scyld/clusterware-node/scripts-enabled/status/
[CW:NewImage status]# ln -s ../../scripts-available/status/timedatectl.sh .
[CW:NewImage status]# exit
```

Upon reboot, any nodes using NewImage will have `timedatectl` enabled automatically.

Note that scripts added to `scripts-enabled/status` are permanently enabled and cannot be disabled later without rebuilding and redeploying the disk image.

Admins can also create their own scripts (see *Creating New Plugins*) and add them to either `scripts-available` or `scripts-enabled`. Placing them in `scripts-available` would allow for future on-the-fly enabling/disabling of that script.

On-The-Fly Plugins

An admin can enable and disable “on-the-fly” plugins by adding or removing them from the `_status_plugins` attribute.

If `_status_plugins=nvidia`, then the system will look for the script in `scripts-available/status/nvidia.sh`.

Available status plugins

- `corestatus`
 - provides basic information about the server: uptime, load average, free RAM, current time measurement, loaded modules and kernel command line, OS release, and ssh keys
 - Note that `ram_free` is reported in KiB, not bytes; so `ram_free=1000` indicates that 1024000 bytes of memory is currently free
- `corenetwork`
 - provides basic network information about the server; for each network device: IP address(es) and MAC
- `selinux`
 - provides basic information on whether selinux is running and what mode it is in, also reports on FIPS if it can determine that too.
- `ipmi`
 - provides basic information that it can find through IPMI
- `virt`
 - provides basic information if the system is running on a virtual machine
- `chrony`
 - provides basic status on the Chrony (time-sync) daemon. Many queueing systems, as well as ClusterWare itself, require well-synchronized clocks across the cluster, so this status information could be useful in triaging, e.g., Slurm start-up issues.
- `timedatectl`
 - provides basic status on the time and date system in the kernel. Again, many parts of a cluster need accurate time-sync to work properly.

Note

Changes to `_status_plugins` will be processed on the next status-update cycle, usually every 10 seconds unless changed by the admin.

5.1.2 Hardware Plugins

Hardware plugins are similar to status plugins but are called less frequently since they are assumed to be more static kinds of information, e.g. the motherboard serial number or CPU vendor information. They are stored as a sub-field of the node's information and can be viewed with:

```
scyld-nodectl --all ls -L
```

As with status plugins, `scripts-available/hardware` is populated with Penguin-provided scripts covering a range of hardware options.

Default frequency is data collection every 300 seconds but this may be overridden on a node-by-node basis with attribute `_hardware_secs`.

For larger-scale management and control, one can set the `_hardware_plugins` and/or `_hardware_secs` attributes inside an attribute group and then join nodes to that group.

Building Hardware Plugins into an Image

As with status plugins, admins can sym-link or add scripts to the `scripts-enabled/hardware` directory inside a disk image.

On-The-Fly Plugins

The `_hardware_plugins` attribute is used for enabling/disabling hardware plugins on-the-fly.

If `_hardware_plugins=infiniband`, then the system will look for the script in `scripts-available/hardware/infiniband.sh`.

Available Hardware Plugins

- `corehardware`
 - provides basic hardware information about the server: total RAM, CPU count, CPU architecture, CPU model, BIOS/UEFI mode, boot style, vendor and product information, BIOS version/date/vendor
 - Note that `ram_total` is reported in KiB, not bytes; so `ram_total=1000` indicates that 1024000 bytes of total memory is available
- `infiniband`
 - provides basic information about any Infiniband network devices found on the server
- `storage`
 - provides basic information about any storage devices found, including NVME
- `nvidia`
 - provides basic information about an NVIDIA GPUs or accelerators

Note

Changes to `_hardware_plugins` will likely be *detected* on the next status cycle, usually every 10 seconds, but will not be processed until the next hardware-update cycle, usually every 300 seconds unless changed by the admin.

5.1.3 Health-Check Plugins

Health-check plugins are intended for more straightforward information about a node's health; information that should not be changing as frequently. They are stored as a sub-field of the "status" information of a node and can be viewed with:

```
scylid-nodectl --all status -L
```

As with status plugins, `scripts-available/health` is populated with Penguin-provided scripts on a variety of topics.

Default frequency is data collection every 300 seconds but this may be overridden on a node-by-node basis with attribute `_health_secs`.

For larger-scale management and control, one can set the `_health_plugins` and/or `_health_secs` attributes inside an attribute group and then join nodes to that group.

Building Health Plugins into an Image

As with status plugins, admins can sym-link from `scripts-available/health` into `scripts-enabled/health` inside a disk image.

On-The-Fly Health Plugins

Similar to status plugins, admins can set the attribute `_health_plugins` to indicate a list of on-the-fly health plugins. If `_health_plugins=rasmem`, then the system will look for the script in `scripts-available/health/rasmem.sh`.

Available Health Plugins

- disk
 - provides a basic disk-usage check based on a threshold. If the storage on the node is greater than the threshold, the node is considered "unhealthy".
 - Set the attribute `_hc_disk_avail_threshold` to set the threshold (can be done at the node- or group-level); can be "123" (amount in KB) or "75%" (for percentage based calculations).
- mem
 - provides a basic "memory health" check based on a threshold. If the current memory used is greater than the threshold, the node is considered "unhealthy".
 - Set the attribute `_hc_mem_avail_threshold` to set the threshold (can be done at the node- or group-level); can be "500" (amount in KB) or "75%".
- pingtest
 - provides a basic "network health" check based on whether the node can successfully ping one or more servers. Each server is pinged 3 times and if any of the pings fail, the node is considered "unhealthy". If the servers can be pinged but the average ping time is greater than the threshold, the node is also considered to be "unhealthy".
 - Set the attribute `_hc_ping_servers` to give a comma-separated list of servers to ping (defaults to the parent head node).
 - Set the attribute `_hc_ping_msecs` to identify the average ping threshold (default is 5 msec).
- rasmem
 - uses the `ras-mc-ctl` tool to provide a basic "memory hardware" check.
- timesync

- provides a basic "time sync" check based on whether the `timedatectl` tool considers the node to be synchronized to upstream time servers. If the tool reports that it is not synchronized, then the node is considered "unhealthy".

Note

Changes to `_health_plugins` will likely be *detected* on the next status cycle, usually every 10 seconds, but will not be processed until the next health-update cycle, usually every 300 seconds unless changed by the admin.

5.1.4 Telegraf Plugins

Where the status plugins are small scripts that are run during the periodic status-update cycle, Telegraf plugins are small configuration files that can be enabled/disabled by the HPC-admin. A telegraf plugin is usually targeted at one particular kind of data - e.g. CPU usage or memory usage.

The `cluserware-telegraf` package can be installed on either a compute node or a head node, but the on-the-fly plugin system currently only works on compute nodes.

For larger-scale management and control, one can set the `_telegraf_plugins` attribute inside an attribute group and then join nodes to that group.

Building Telegraf Plugins into an Image

Similar to status plugins, there is another directory:

```
/opt/scyld/clusterware-telegraf/telegraf-available
```

that contains Penguin-provided config files. Those can be sym-linked into `./telegraf-enabled` inside a disk Image.

On-The-Fly Telegraf Plugins

On compute nodes, an admin can enable/disable "on-the-fly" plugins by setting or clearing out that node's `_telegraf_plugins` attribute.

Note

Changes to `_telegraf_plugins` will force a full restart of the Telegraf daemon, so frequent changes could cause performance degradation.

Available status plugins

- `amd-rocm-smi`
 - provides information from AMD ROCm-based GPUs, including GPU and memory usage.
- `chrony`
 - provides information from Chrony on the time synchronization of the system.
- `cpu`
 - gives aggregate and per-CPU-core utilization data.
- `cw-attrs`
 - injects ICE ClusterWare™ node attributes and fields into the Telegraf/Influx data stream. This can be helpful for customizing dashboards for different "types" of nodes.

- By default, all node attributes and fields will be sent to Telegraf. This can be modified with the reserved attribute, `_telegraf_omit_pattern`. The pattern is an awk regex, usually of the form `(word1|word2)` and any matching fields will be omitted.
- disk
 - provides disk space utilization (used and free; both inode and capacity utilization).
- disk_head
 - disk utilization, customized for head nodes; includes more disk types in its data.
- diskio
 - gathers per-device disk I/O rates.
- hddtemp
 - reports data from hddtemp daemons.
- infiniband
 - gathers information for all Infiniband devices and ports on the system.
- intel-powerstat
 - provides data from Intel's Powerstat features.
- interrupts
 - reports data from IRQs, including interrupts and soft-interrupts.
- ipmisensor
 - collects data from the IPMI system. Note that more configuration may be required for this plugin to provide useful data (e.g. full URL, username, password, etc.)
- kernel
 - gathers kernel statistics from `/proc/stat`.
- kernel-vmstat
 - gathers kernel statistics from `/proc/vmstat`.
- lm-sensors
 - collects sensor information from the `lm-sensors` package.
- lustre
 - gathers job-level data on Lustre file system usage. Note that more configuration may be required for this plugin to provide useful data.
- mem
 - reports memory statistics for the system, including free and used amounts as well as `vmalloc`, `cache`, `high-memory` areas.
- net
 - provides aggregate and per-device network information including bytes sent and received, packets sent and received, errors, and more.
- netresponse
 - reports the network response time to contact the parent head node.
- netstat

- gathers TCP metrics from `lsof`, including established connections, time-wait, and socket counts.
- `nfsclient`
 - collects per-mount-point statistics for any NFS file systems (`/proc/self/mountstats`).
- `nvidia-smi`
 - provides information from NVIDIA GPUs, including GPU and memory usage, temperatures, and more.
- `ping`
 - records ping times back to the current parent head node.
- `processes`
 - reports information on the number of processes on the system that are in different states (zombie, sleeping, running, etc.).
- `rsyslog`
 - provides a local listening port which can receive syslog-formatted data (e.g. from rsyslog forwarding).
 - Note that by default the ClusterWare platform does not forward compute-node data. In addition to the rsyslog telegraf plugin, admins must also have configured rsyslog forwarding. The ClusterWare platform does include a "disabled" rsyslog config file: `/etc/rsyslog.d/cw_local_telegraf.conf.disabled`. This can be enabled by simply removing the `.disabled` from the filename and restarting rsyslog.
- `swap`
 - provides information about swap memory usage.
- `temp`
 - collects temperature data from sensors on the system.

Head Node Functionality

The ClusterWare-telegraf plugin system has reduced functionality on head nodes. Since head nodes do not currently have attributes, there is no way to do on-the-fly changes to the Telegraf plugins and so adding entries to `telegraf-enabled` is the only way to add plugins to the system.

Additionally, while the compute nodes will automatically detect changes and start/restart Telegraf automatically, changes to head nodes must be handled manually.

Once the `telegraf-enabled` directory is ready on the head-node, admins should run `reconfig-telegraf.sh` to push the enabled plugins into production (this will also restart Telegraf).

```
/opt/scyld/clusterware-telegraf/bin/reconfig-telegraf.sh
```

Note

Changes to `_telegraf_plugins` will be processed by the ClusterWare platform on the next status-update cycle, usually every 10 seconds unless changed by the admin. However, it may take several seconds before Telegraf actually restarts, and it then has to go through its own “data refresh” cycle (again, usually every 10 seconds, unless changed by the admin). So there could be a non-trivial delay (30-40 sec) before a new plugin's data is actually visible on a dashboard.

5.1.5 Creating New Plugins

The ICE ClusterWare™ platform provides a flexible plugin system for collecting a variety of node information – status, hardware, health, and telegraf monitoring. There are 4 types of plugins:

- *Status Plugins*
 - These are called every status cycle, usually every 10 sec; can return any datatype: e.g. free RAM is given as a float, distro is a string, etc.; one script can return multiple “sensor” readings;
- *Hardware Plugins*
 - These called less often than “regular” status plugins, usually every 300 sec; can return any datatype; one script can return multiple “sensor” readings;
- *Health-Check Plugins*
 - These are called less often than “regular” status plugins, usually every 300 sec; returns “healthy”, “un-healthy”, or a time-stamp (indicating that the plugin is still calculating the value); one script can return multiple “sensor” readings;
- *Telegraf Plugins*
 - These are really smaller, more granular Telegraf config files that the ClusterWare platform can individually enable/disable; each script is a self-contained piece of Telegraf configuration which contains settings for a given Telegraf “input” module.

Note

While plugins do run as root on the compute nodes, they can still be restricted by SELinux.

5.1.5.1 Creating Status Plugins

Broadly speaking, a status plugin is a shell-script that returns one or more JSON-like blobs of data for a named sensor reading:

```
“<sensor_name>”: <JSON_data>
```

For a single plugin that emits multiple sensor readings, they should be separated by a newline:

```
“<sensor_name1>”: <JSON_data>
“<sensor_name2>”: <JSON_data>
```

The script runs as root and can use any tool, or sequence of tools, on the system in order to collect the information and format it properly.

While the script could reach across the network to run commands on other systems, this would potentially take significantly longer to process and so it is discouraged, especially in status plugins, although pinging other systems or doing more complicated communication checks may be appropriate in health checks.

The first line of the script should be a `#!` line giving the shell or interpreter to use. Every plugin is sent the path to the `clusterware-node` package as the first argument since there is a useful “library” of routines in the `functions.sh` script. While less useful for the usually-dynamic status information, every script is sent a cache directory as the second argument, usually `/opt/scyld/clusterware-node/etc/status.d`. This directory can be used to store information for the next cycle rather than recomputing some lengthy calculation.

The sensor name can be any quoted string, though it is recommended to not have spaces or odd characters to avoid later problems when trying to use the data, we suggest using alphanumeric characters and underscores (A-Za-z0-9_). The JSON-data portion can be any JSON data: a string, number, boolean, list, or dictionary.

As the plugins are just scripts, to debug them, simply execute them manually and verify that the output is correct. Note that the plugin system does very little error-checking on the scripts, and since it is executing so often, it does not do a formal check of the JSON. If bad data is returned, it will cause failures in the status-update cycle. On the compute node, the failure will likely be silent (unless `_status_debug=1`) so it is best to check the head node logs to get more information on the error.

Note

Changes to `_status_plugins` will be processed on the next status-update cycle, usually every 10 seconds unless changed by the admin.

5.1.5.2 Creating Hardware Plugins

Broadly speaking, a hardware plugin is also just a shell-script that returns one or more JSON-like blobs of data for a named sensor reading:

```
"<sensor_name>": <JSON_data>
```

For a single plugin that emits multiple sensor readings, they should be separated by a newline. As with status plugins, the script can run any tool, or sequence of tools, on the system.

The first line of the script should be a `#!` line giving the shell or interpreter to use.

Since hardware information is usually more static, it may make sense for hardware plugins to calculate information once and then cache it to a file on disk so as to reduce future computational effort needed. Every script is sent the cache directory as the second argument, usually `/opt/scyld/clusterware-node/etc/hardware.d`. Every hardware plugin is also sent the path to the clusterware-node package as the first argument since there is a useful “library” of routines in the `functions.sh` script.

The sensor name can be any quoted string, though it is recommended to not have spaces or odd characters to avoid later problems when trying to use the data, we suggest using alphanumeric characters and underscores (A-Za-z0-9_). The JSON-data portion can be any JSON data: a string, number, boolean, list, or dictionary.

As the hardware plugins are just scripts, to debug them, simply execute them manually and verify that the output is correct. As with the status plugins, there is very little error-checking done on the scripts.

Note

Changes to `_hardware_plugins` will likely be *detected* on the next status cycle, usually every 10 seconds, but will not be processed until the next hardware-update cycle, usually every 300 seconds unless changed by the admin.

5.1.5.3 Creating Health-Check Plugins

A health plugin is a shell-script that returns one or more strings for a named sensor reading:

```
"<sensor_name>": "<string>"
```

where the string is usually either “healthy” or “unhealthy”, but any string can be sent. If the first word in the string is “healthy”, then it is assumed that the health-check passed; any other string is considered unhealthy (not just the word “unhealthy”). One can optionally communicate more specific information in the string, e.g. any errors or issues that were detected. I.e. `heathy: ping time was 0.164 ms` would also indicate a healthy check; `unhealthy: could not ping server` or simply `could not ping server` would also indicate an unhealthy check (since it does not start with `healthy`) and also give a possible path to help triage the problem. If the plugin emits more than one sensor reading, they should be separated by newlines.

The sensor name can be any quoted string, though it is recommended to not have spaces or odd characters to avoid later problems when trying to use the data, we suggest using alphanumeric characters and underscores (A-Za-z0-9_).

As with status plugins, the health-check script can run any tool, or sequence of tools, on the system. The first line of the script should be a `#!` line giving the shell or interpreter to use.

Every script is given the cache directory as the second argument, usually `/opt/scyld/clusterware-node/etc/health.d`. Lengthier calculations can potentially be run once and stored there for future use. Every health-check plugin is also sent the path to the clusterware-node package as the first argument since there is a useful “library” of routines in the `functions.sh` script.

Prior to sending the data to the head node, the compute node will assess all the health-check values and assign the node a global “health” status in the node's `_health` attribute. If any health check reports unhealthy, then the node is considered unhealthy; if all checks report healthy, then the node is considered healthy.

As the health-check plugins are just scripts, to debug them, simply execute them manually and verify that the output is correct. As with the status plugins, there is very little error-checking done on the scripts.

Note

Changes to `_health_plugins` will likely be *detected* on the next status cycle, usually every 10 seconds, but will not be processed until the next health-update cycle, usually every 300 seconds unless changed by the admin.

5.1.5.4 Creating Telegraf Plugins

Where the status plugins are small scripts that are run during the periodic status-update cycle, Telegraf plugins are small configuration files that can be enabled/disabled by the HPC-admin.

“Developing” Telegraf plugins for the ClusterWare platform really means the creation of config files, often by splitting up one of the larger Telegraf sample configuration files into smaller, self-contained pieces. Developing brand-new Telegraf plugins is outside the scope of this document.

A list of Telegraf plugins can be found at: <https://docs.influxdata.com/telegraf/v1/plugins/>

Note

head nodes can only use the telegraf-enabled directory as there is no `_telegraf_plugins` attribute. Admins must run the `/opt/scyld/clusterware-telegraf/bin/reconfig-telegraf.sh` script manually to push any changes into production.

Examples

As an example, if a cluster was running the Ceph file system, then one might want to use the Telegraf “inputs.ceph” module:

<https://github.com/influxdata/telegraf/blob/release-1.28/plugins/inputs/ceph/README.md>

To do so, create a new plugin file: `/opt/scyld/clusterware-telegraf/telegraf-available/ceph.conf` with the following:

```
[[inputs.ceph]]
interval = '1m'
ceph_binary = "/usr/bin/ceph"
socket_dir = "/var/run/ceph"
mon_prefix = "ceph-mon"
osd_prefix = "ceph-osd"
```

(continues on next page)

(continued from previous page)

```

mds_prefix = "ceph-mds"
rgw_prefix = "ceph-client"
socket_suffix = "asok"
ceph_user = "client.admin"
ceph_config = "/etc/ceph/ceph.conf"
gather_admin_socket_stats = true
gather_cluster_stats = false

```

With that file stored in `telegraf-available`, the admin can either sym-link it into `telegraf-enabled` or add it to the `_telegraf_plugins` attribute:

```
scylld-nodectl -all set _telegraf_plugins=ceph
```

Once the plugin is enabled through either method, the Telegraf daemon on the compute nodes will begin sending Ceph data to the head nodes.

Another example would be to ping several remote servers instead of just the parent head node. This could be useful, for example, to detect issues with a node's connection to the primary storage server and a network gateway, for example. In this case, we could edit the `ping.conf` file in `telegraf-available` and include the storage server name in the list of URLs to ping:

```

[[inputs.ping]]
## List of urls to ping
urls = ["parent-head-node", "storage-server", "gateway-server"]
## number of pings to send per collection (ping -c <COUNT>)
count = 1

```

The documentation on the Ping plugin:

<https://github.com/influxdata/telegraf/blob/release-1.28/plugins/inputs/ping/README.md>

shows a number of other options that admins may find useful to configure. If an existing file is modified, a restart of Telegraf will be necessary to have the system re-read the configuration file (`systemctl restart telegraf`).

Note that Telegraf's `exec` and `execd` modules allow for arbitrary scripts or executables to be run, with the output being ingested into Telegraf. This can be a very powerful tool for admins to write custom scripts that might be very specific to their cluster.

See the Influxdata documentation for more information:

- The `exec` plugin launches a new shell every update cycle when it runs the script <https://github.com/influxdata/telegraf/blob/release-1.28/plugins/inputs/exec/README.md>
- The `execd` plugin creates one shell during Telegraf start-up, and that shell is assumed to launch a daemon process which repeatedly sends data <https://github.com/influxdata/telegraf/blob/release-1.28/plugins/inputs/execd/README.md>

5.2 Using Ansible

A compute node can be configured to execute an Ansible playbook at boot time or after the node is *up*. In the following example, the cluster administrator creates a git repository hosted by the ICE ClusterWare™ head nodes, adds an extremely simple Ansible playbook to that git repository, and assigns a compute node to execute that playbook.

Install the `clusterware-ansible` package into the image (or images) that you want to support execution of an Ansible playbook:


```
scyld-modimg -i DefaultImage --install clusterware-ansible --upload --overwrite
```

The administrator should amend their PATH variable to include the git binaries that are provided as part of the *clusterware* package in `/opt/scyld/clusterware/git/`. This is not strictly necessary, though the `git` in that subdirectory is often significantly more recent than the version normally provided by a base distribution:

```
export PATH=/opt/scyld/clusterware/git/bin:${PATH}
```

The administrator should add their own personal public key to their ClusterWare admin account. This key will be populated into root user's (or *_remote_user*'s) `authorized_keys` file on newly booted compute nodes. See [Compute Node Remote Access](#) for details. In addition, this provides simple SSH access to the git repository:

```
scyld-adminctl up keys=@/full/path/.ssh/id_rsa.pub
```

Adding the localhost's host keys to a personal `known_hosts` file is not strictly necessary, though it will avoid an SSH warning that can interrupt scripting:

```
ssh-keyscan localhost >> ~/.ssh/known_hosts
```

Now create a ClusterWare git repository called "ansible". This repository will default to *public*, meaning it is accessible read-only via unauthenticated HTTP access to the head nodes and therefore should not include unprotected sensitive passwords or keys:

```
scyld-clusterctl gitrepos create name=ansible
```

Note that being unauthenticated means the HTTP access mechanism does not allow for `git push` or other write operations. Alternatively the repository can be marked *private* (`public=False`), although it then cannot be used for a client's `ansible-pull`.

Initially the repository will include a placeholder text file that can be deleted or discarded.

Now clone the git repo over an SSH connection to localhost:

```
git clone cwgit@localhost:ansible
```

The administrator could also create that clone on any machine that has the appropriate private key and can reach the SSH port of a head node.

Finally, create a simple Ansible playbook to demonstrate the functionality:

```
cat >ansible/HelloWorld.yaml <<EOF
---
- name: This is a hello-world example
  hosts: n*.cluster.local
  tasks:
    - name: Create a file called '/tmp/testfile.txt' with the content
      copy:
        content: hello world
        dest: /tmp/testfile.txt
EOF
```

and add that playbook to the "ansible" git repo:

```
bash -c "\
cd ansible; \
git add HelloWorld.yaml; \
```

(continues on next page)

(continued from previous page)

```
git -c user.name=Test -c user.email='<test@test.test>' \
    commit --message 'Adding a test playbook' HelloWorld.yaml; \
git push; \
"
```

Multiple playbooks can co-exist in the git repo.

In a multiple-head node cluster an updated git repository will be replicated to other head nodes in the cluster, so any client `ansible_pull` to any cluster head node will see the same playbook and the same commit history. This replication can require several seconds to complete.

With the playbook now available in the git repo, configure the compute node to execute `ansible-pull` to download it at boot time:

```
scyld-nodectl -i n1 set _ansible_pull=git:ansible/HelloWorld.yaml
```

Alternatively, to download the playbook from an external git repository on the server named *gitserver*:

```
scyld-nodectl -i n1 set _ansible_pull=http://gitserver/path/to/repo/root:HelloWorld.yaml
```

Either format can optionally end with "`@<gitrev>`", where `<gitrev>` is a specific commit, tag, or branch in the target git repo.

Note the colon between the `root` and `HelloWorld.yaml` components of the URL path in the previous example. That colon is required to denote the root of the repository within the URL. Any path components found after that colon is assumed to exist within the repository and the entirety of the URL preceding the colon must point to a cloneable git repository.

Use the `_ansible_pull_args` attribute to specify any arguments to the underlying `ansible-pull` command.

You may now reboot the node and wait for it to boot to an *up* status after the playbook has executed:

```
scyld-nodectl -i n1 reboot then waitfor up
```

You can verify that the `HelloWorld.yaml` playbook executed:

```
scyld-nodectl -in1 exec cat /tmp/testfile.txt ; echo
```

Note that during playbook execution the node remains in the *booting* status, changing to an *up* status after the playbook completes, assuming the playbook is not fatal to the node. That status may timeout to *down* (with no ill effect) when executing a lengthy playbook before switching to *up* after playbook completion. Administrators are advised to log the ansible progress to a known location on the booting node, such as `/var/log/ansible.log`.

The `clusterware-ansible` package supports another attribute, `_ansible_pull_now`, which uses the same syntax as `_ansible_pull`. Prior to first use, the administrator must enable the `cw-ansible-pull-now` service inside the chroot image:

```
systemctl enable cw-ansible-pull-now
```

and then on a running compute node, start the service:

```
systemctl start cw-ansible-pull-now
```

When the attribute is present and the service has been enabled and started, the node will download and execute the playbook during the node's next status update event, which occur every 10 seconds by default. Once the node completes execution of the playbook, it directs the head node to prepend "done" to the `_ansible_pull_now` attribute to ensure the script does not run again.

5.2.1 Using Node Attributes with Ansible

Admins can also change how playbooks run by reading ClusterWare node attributes into Ansible variables. The `clusterware-node` package includes a library of shell functions that can be used, in particular, `attribute_value` reads an attribute out of nodes configuration.

Inside the playbook, one can register a variable using the output of a command, and that command can reference the `attribute_value` function:

```
- name: Read the slurm_server attribute
  shell:
    executable: /bin/bash
    cmd: "source /opt/scyld/clusterware-node/functions.sh && attribute_value slurm_server
  register: slurm_server
```

This snippet would set an Ansible variable called `slurm_server` that would read the node attribute of the same name. Any ClusterWare or user-defined attribute can be referenced in this way. If a default value is needed, it can be given as a second argument: `attribute_value attrname defaultvalue`.

5.2.2 Applying Ansible Playbooks to Images

Cluster administrators commonly create and deploy a golden image containing all of the necessary libraries, tools, and applications. Given the frequent nature of software updates, the golden image can be out of date soon after it is created. With this in mind, many production clusters collect required changes into an Ansible playbook and then use the `_ansible_pull` functionality to deploy that playbook to ClusterWare nodes at boot time, or even to booted nodes using the `_ansible_pull_now` functionality.

Applying changes from an Ansible playbook adds a delay between when the node begins booting and when the node is ready to accept jobs after fully booting. Eventually this delay becomes cumbersome and the cluster administrator will want to flush the changes out of the playbook and into the image. The `scyld-modimg -deploy <PATH>` command supports executing a local playbook into the `chroot`.

Using this functionality requires that the `clusterware-ansible` package is installed on the head node and that the `community.general` Ansible Galaxy collection is installed for the `chroot` connection type. The following pair of commands installs the package on the system and installs the Ansible collection for the root user:

```
sudo dnf install --assumeyes --enablerepo=scyld\* clusterware-ansible
sudo -E /opt/scyld/clusterware-ansible/env/bin/ansible-galaxy \
  collection install community.general
```

The collection needs to be available to root because the `ansible-playbook` command is executed using `sudo` to allow full write permissions to all files within the `chroot`.

The `scyld-modimg` command assumes that any path that ends with `.yaml` is an Ansible playbook and uses the configured software to execute that playbook within the `chroot`.

```
scyld-modimg -iDefaultImage --deploy HelloWorld.yaml \
  --progress none --upload --overwrite --discard-on-error
```

The new `--discard-on-error` argument prevents the tool from asking for user confirmation before uploading. It assumes that the user wants to keep the result of a successful run but stop if an error was encountered. The following is an example of the expected output from the previous command:

```
[admin@cwhead ~]$ scyld-modimg -iDefaultImage --deploy HelloWorld.yaml \
  --progress none --upload --overwrite --discard-on-error
Treating HelloWorld.yaml as an ansible playbook
```

(continues on next page)

(continued from previous page)

```

Downloading and unpacking image DefaultImage
Executing step: Ansible ['/opt/scyld/clusterware-ansible/bin/ansible-playbook',
↳ 'HelloWorld.yaml']
  DefaultImage : ok=2    changed=1    unreachable=0    failed=0    skipped=0    ↳
↳ rescued=0    ignored=0
  step completed in 0:00:06.2
Executing step: Upload
Repacking DefaultImage
  fixing SELinux file labels...
  done.
Checksumming...
Cleaning up.
Checksumming image DefaultImage
Replacing remote image.
  step completed in 0:09:33.7

```

The `scyld-modimg --deploy <PATH>` command recognizes when the provided path is an `_ansible_pull` compatible URL. If the `clusterware-ansible` package is installed within the image, then the `scyld-modimg` command triggers execution within the unpacked image via the same mechanisms as setting the `_ansible_pull` reserved attribute. Note that an unpacked image will not include files that are normally generated during the boot process (such as `attributes.ini`, `heads.cache`, or other files in `/opt/scyld/clusterware-node/etc`), so scripts run this way must not rely on such files.

5.3 Using Singularity

Singularity is available in the ICE ClusterWare™ platform by installing the `singularity-scyld` RPM, which is built from source developed by Sylabs Inc., or by installing the `singularity` RPM found in the EPEL yum repository. See <https://www.sylabs.io/docs> for their extensive documentation.

The following example creates a Singularity container `openmpi.sif` containing `openmpi3.1`, and placing that container in a bootable image.

First create the `openmpi.def` Singularity definition file, then use that file to create the container:

```

# Use quoted "EOF" for bash to avoid % and $ expansions; just EOF for sh.
cat <<-"EOF" >openmpi.def
BootStrap: yum
OSVersion: 7
MirrorURL: http://mirror.centos.org/centos-%{OSVERSION}/%{OSVERSION}/os/$basearch/
Include: yum

%files
  /etc/yum.repos.d/clusterware.repo /etc/yum.repos.d/clusterware.repo

%environment
  PATH=/opt/scyld/openmpi/3.1.3/gnu/bin:$PATH
  LD_LIBRARY_PATH=/opt/scyld/openmpi/3.1.3/gnu/lib:/opt/scyld/slurm/lib64:$LD_LIBRARY_
↳ PATH
  MPI_HOME=/opt/scyld/openmpi/3.1.3/gnu
  MPI_LIB=/opt/scyld/openmpi/3.1.3/gnu/lib
  MPI_INCLUDE=/opt/scyld/openmpi/3.1.3/gnu/include
  MPI_SYSCONFIG=/opt/scyld/openmpi/3.1.3/gnu/etc

%post

```

(continues on next page)

(continued from previous page)

```

# IMPORTANT:
#   If instead using "OSVersion: 6" instead of "OSVersion 7" above,
#   then for any subsequent `rpm` or `yum`, add:
# rpm --rebuilddb
echo "Installing openmpi3.1-gnu rpm"
yum -y install openmpi3.1-gnu
exit 0
EOF

# Create the Singularity chroot "/tmp/openmpi" in which updates can be made.
sudo singularity build --sandbox /tmp/openmpi openmpi.def

# Make a sample update: build an openmpi test program inside the chroot.
sudo singularity exec -w /tmp/openmpi \
    mpicc -o /usr/bin/ring /opt/scyld/openmpi/3.1.3/gnu/examples/ring_c.c

# Finalize the sandbox chroot into the Singularity container "openmpi.sif".
sudo singularity build openmpi.sif /tmp/openmpi

```

Create a bootable image that hosts the Singularity container and can execute openmpi applications:

```

# Clone a new image instead of modifying an existing image.
scyld-imgctl -i DefaultImage clone name=SingularityImage

# Install needed packages inside the new image.
scyld-modimg -i SingularityImage --freshen --overwrite --no-discard \
    --install singularity-scyld,openmpi3.1-gnu --upload

# Now get into the chroot of the Singularity image.
scyld-modimg -i SingularityImage --chroot --overwrite --upload --no-discard

# Inside the root, add your userid (e.g., "myuserid") if necessary, which
# creates a /home/myuserid/ directory, and import the Singularity container file.
useradd myuserid
scp myuserid@localhost:/home/myuserid/openmpi.sif /home/myuserid/
exit

```

Boot nodes n0 and n1 with SingularityImage:

```

scyld-bootctl -i DefaultBoot clone name=SingularityBoot
scyld-bootctl -i SingularityBoot update image=SingularityImage
scyld-nodectl -i n[0-1] set _boot_config=SingularityBoot
# Now reboot nodes n0 and n1
scyld-nodectl -i n[0-1] reboot

```

When the nodes are *up*, then initialize passphrase-less key-based access, as described in [OpenMPI](#), [MPICH](#), and/or [MVAPICH](#).

Now you can run the ring program from n0 (or n1):

```

# logged into n0, or using a job scheduler
mpirun -np 2 --host n0,n1 singularity exec openmpi.sif /usr/bin/ring

```

Or from the head node:

```
# If not already installed
sudo yum install singularity-scyld openmpi3.1-gnu --enablerepo=scyld*

module load openmpi/gnu/3.1.3
mpirun -np 2 --host n0,n1 singularity exec openmpi.sif /usr/bin/ring
```

5.4 Using Docker for Compute Nodes

The ICE ClusterWare™ platform supports Docker, which is available from CentOS.

The following example shows Docker being used to execute the pre-built Docker "Hello World" image. First preferably create a new image:

```
# Clone a new image instead of modifying an existing image.
scyld-imgctl -i DefaultImage clone name=DockerImage

# Install needed packages inside the new image.
# NOTE: This uses the default _boot_style=rwram and _boot_rw_layer=overlayfs
scyld-modimg -i DockerImage --freshen --overwrite --no-discard \
             --install docker --exec "systemctl enable docker" --upload
```

Alternatively, the administrator may choose to use a *_boot_style* of *roram* or *iscsi* for nodes using this *DockerImage*. To accomplish this, more must be done to the *DockerImage* image and to all the nodes that use that image. For example:

```
# Additionally create file /etc/rwtab.d/docker in the image.
scyld-modimg -i DockerImage --freshen --overwrite --no-discard \
             --install docker --exec "systemctl enable docker" \
             --exec "echo 'empty /var/lib/docker' >/etc/rwtab.d/docker" --upload

scyld-nodectl -i <NODES> set _boot_style=roram _boot_rw_layer=rwtab
# Or use scyld-attribctl if the <NODES> are in a group.
```

You will also need to set up IP forwarding on the head node(s) for the node to access the external Internet, which may likely involve using `scyld-modimg` to add appropriate nameserver entries to the node's `/etc/resolv.conf`. See [Configure IP Forwarding](#) for details.

Now boot node `n0` with the new `DockerImage`:

```
scyld-bootctl -i DefaultBoot clone name=DockerBoot
scyld-bootctl -i DockerBoot update image=DockerImage
scyld-nodectl -i n0 set _boot_config=DockerBoot
# Now reboot node n0
scyld-nodectl -i n0 reboot
```

When node `n0` is *up*, you can initialize passphrase-less key-based access to allow your current administrator user to ssh to the node. See [OpenMPI, MPICH, and/or MVAPICH](#). Alternatively, you can simply login as root:

```
sudo ssh n0

# Now as user root on n0, and if n0 can access external Internet websites:

[root@n0] docker run hello-world
Unable to find image 'hello-world:latest' locally
```

(continues on next page)

(continued from previous page)

```
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world
1b930d010525: Pull complete
Digest: sha256:41a65640635299bab090f783209c1e3a3f11934cf7756b09cb2f1e02147c6ed8
Status: Downloaded newer image for docker.io/hello-world:latest
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Note the Hello from Docker! line in the above output.

5.5 Using Kubernetes

This section provides examples of setting up Kubernetes clusters with ICE ClusterWare™ and non-ClusterWare systems.

See *Kubernetes* for a general explanation of how to install and initialize a Kubernetes cluster and *scyl-d-kube* for details about the command and related arguments.

Note

All examples assume you have root user or ClusterWare administrator access and that the *clusterware-kubeadm* package is installed.

5.5.1 Using a Single Non-ClusterWare System as a Control Plane

1. On the non-ClusterWare system where *clusterware-kubeadm* is installed, kube1 (10.154.3.1), initialize the local system as a control plane:

```
scyl-d-kube --init
```

The following messages are printed out after the control plane initialization:

```

...
To join ClusterWare NODES/IMAGE as worker to this non ClusterWare control plane:
scyl-d-kube -i NODES --join --token nfg0ku.73f1gre8gxzco1qx --cahash_
↪sha256:cc999d4001c018a3423238773614bb8d6d8ad720e1f31a8b0e862052a67262da --cluster_
↪10.154.3.1
scyl-d-kube --image IMAGE --join --token nfg0ku.73f1gre8gxzco1qx --cahash_
↪sha256:cc999d4001c018a3423238773614bb8d6d8ad720e1f31a8b0e862052a67262da --cluster_
↪10.154.3.1

To join non ClusterWare system as worker to this non Clusterware control plane:
scyl-d-kube --join --token nfg0ku.73f1gre8gxzco1qx --cahash_
↪sha256:cc999d4001c018a3423238773614bb8d6d8ad720e1f31a8b0e862052a67262da --cluster_
↪10.154.3.1
...

```

2. Verify the deployment status by running the following command:

```
kubectl get nodes -o wide
```

3. Using the messages at the end of step 1 as a guide, join ClusterWare nodes (n[11-14]) as workers with explicit --token, --cahash, and --cluster arguments to the control plane kube1 (10.154.3.1):

```

scyl-d-kube -i n[11-14] --join nfg0ku.73f1gre8gxzco1qx --cahash_
↪sha256:cc999d4001c018a3423238773614bb8d6d8ad720e1f31a8b0e862052a67262da --cluster_
↪10.154.3.1

```

4. Create a Kubernetes worker node image with explicit --token, --cahash, and --cluster arguments then boot n[15-20] with the node image as workers to control plane kube1 (10.154.3.1):

```

$ scyl-d-bootctl -i DefaultBoot clone name=KubeWorkerBoot2
$ scyl-d-imgctl -i DefaultImage clone name=KubeWorkerImage2
$ scyl-d-kube --image KubeWorkerImage2 --join --token nfg0ku.73f1gre8gxzco1qx --
↪cahash sha256:cc999d4001c018a3423238773614bb8d6d8ad720e1f31a8b0e862052a67262da --
↪cluster 10.154.3.1
$ scyl-d-bootctl -i KubeWorkerBoot2 up image=KubeWorkerImage2
$ scyl-d-nodectl -i n[15-20] set _boot_config=KubeWorkerBoot2
$ scyl-d-nodectl -i n[15-20] reboot

```

5. On EACH non-ClusterWare system that you want to join as a worker and where *clusterware-kubeadm* is installed, join the local system to control plane kube1 (10.154.3.1) with explicit --token, --cahash, and --cluster arguments:

```

scyl-d-kube --join --token nfg0ku.73f1gre8gxzco1qx --cahash_
↪sha256:cc999d4001c018a3423238773614bb8d6d8ad720e1f31a8b0e862052a67262da --cluster_
↪10.154.3.1

```

6. Verify the deployment status by running the following command:

```
kubectl get nodes -o wide
```

You should see kube1 as the control plane and both the ClusterWare and non-ClusterWare systems you joined as workers in the output.

5.5.2 Using Multiple ClusterWare Nodes as a Control Plane

1. Create High Available (HAProxy and Keepalived) configure files with ClusterWare node n21 (10.154.1.121) as the first control plane node and n22 (10.154.1.122) and n23 (10.154.1.123) as additional control plane nodes:

```
scyld-kube --prepare-lb 10.154.2.0 n21:10.154.1.121,n22:10.154.1.122,n23:10.154.1.123
```

Note

10.154.2.0 is an unused IP within the cluster network. It will be the apiserver virtual IP for these Kubernetes control planes.

2. Initialize the first control plane node on n21:

```
scyld-kube -i n21 --init-ha
```

The following message is printed out from a successful initialization:

```
...
To join ClusterWare NODES as control planes to this ClusterWare control plane:
scyld-kube -i NODES --join-ha --certificate-key_
↪1271738c2ee3cda4dc022a9bef8a3166550a608e80d000cdf0dfbe3defb03776 --cluster n21
...
```

Note

There will also be messages about joining non-ClusterWare systems as workers to this ClusterWare control plane.

3. Verify the first control plane node is ready and note the `--cluster` value with INTERNAL-IP. See [Checking Deployment Status](#). If it is more than 2 hours since the first control plane node was initialized, generate a new certificate key. See [Additional Configuration](#).
4. Join n22 and n23 as additional control plane nodes to the first control plane node (n21):

```
scyld-kube -i n[22-23] --join-ha --certificate-key_
↪1271738c2ee3cda4dc022a9bef8a3166550a608e80d000cdf0dfbe3defb03776 --cluster n21
```

5. Verify all control plane nodes are ready. See [Checking Deployment Status](#).
6. Using the messages at the end of step 2 as a guide, join ClusterWare nodes (n[1-4]) as workers to the control plane node n21:

```
scyld-kube -i n[1-4] --join --cluster n21
```

7. Create a Kubernetes worker node image and then boot n[5-10] with the node image as workers to the control plane node n21:

```
$ scyld-bootctl -i DefaultBoot clone name=KubeWorkerBoot
$ scyld-imgctl -i DefaultImage clone name=KubeWorkerImage
$ scyld-kube --image KubeWorkerImage --join --cluster n21
$ scyld-bootctl -i KubeWorkerBoot up image=KubeWorkerImage
```

(continues on next page)

(continued from previous page)

```
$ scyld-nodectl -i n[5-10] set _boot_config=KubeWorkerBoot
$ scyld-nodectl -i n[5-10] reboot
```

- On EACH non-ClusterWare system that you want to join as a worker and where *clusterware-kubeadm* is installed, join the local system to the control plane node n21 with explicit `--token`, `--cahash`, and `--cluster` arguments:

```
scyld-kube --join --token yp6lxa.wcb6g48ud3f2cwng --cahash_
→sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --cluster_
→10.154.2.0
```

5.5.3 Using Multiple Non-ClusterWare Systems as a Control Plane

- On EACH non-ClusterWare system where *clusterware-kubeadm* is installed, create High Available (HAProxy and Keepalived) configure files with kube2 (10.154.3.2) as the first control plane node and kube3 (10.154.3.3) and kube4 (10.154.3.4) as additional control plane nodes:

```
scyld-kube --prepare-lb 10.154.4.0 kube2:10.154.3.2,kube3:10.154.3.3,kube4:10.154.3.
→4
```

Note

10.154.4.0 is an unused IP within the cluster network. It will be the apiserver virtual IP for these Kubernetes control planes.

- Initialize the control plane on kube2:

```
scyld-kube --init-ha
```

The following message is printed out from a successful initialization:

```
...
To join non ClusterWare system as control plane to this non ClusterWare control_
→plane:
scyld-kube --join-ha --token ka8y8y.enwcyfsk4hblayz5 --cahash_
→sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --
→certificate-key 86ae5340eb592759debd51ab9a03c9f9005a5027e7900d3a2fff687de473e2be -
→-cluster 10.154.4.0
...
```

Note

There will also be messages about joining ClusterWare NODES/IMAGE as workers to this non-ClusterWare control plane.

- On kube2, verify the first control plane node is ready. See *Checking Deployment Status*. If it is more than 2 hours since the first control plane node was initialized, generate a new certificate key. See *Additional Configuration*.
- On kube3, create the same High Available (HAProxy and Keepalived) configure files as on kube2 and then join kube3 as an additional control plane node:

```
$ scyld-kube --prepare-lb 10.154.4.0 kube2:10.154.3.2,kube3:10.154.3.3,kube4:10.154.
↪3.4
$ scyld-kube --join-ha --token ka8y8y.enwcyfsk4hb1ayz5 --cahash_
↪sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --
↪certificate-key 86ae5340eb592759debd51ab9a03c9f9005a5027e7900d3a2fff687de473e2be -
↪-cluster 10.154.4.0
```

- Repeat step 4 on kube4.
- Verify all control planes nodes are ready. See *Checking Deployment Status*.
- Using the messages at the end of step 2 as a guide, join ClusterWare nodes (n[11-14]) as workers with explicit `--token`, `--cahash`, and `--cluster` arguments to the control plane node kube2 (10.154.4.0):

```
scyld-kube -i n[11-14] --join --token ka8y8y.enwcyfsk4hb1ayz5 --cahash_
↪sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --cluster_
↪10.154.4.0
```

- Create a Kubernetes worker node image with explicit `--token`, `--cahash`, and `--cluster` arguments and then boot n[15-20] with the node image as workers to the control plane node kube2 (10.154.4.0):

```
$ scyld-bootctl -i DefaultBoot clone name=KubeWorkerBoot2
$ scyld-imgctl -i DefaultImage clone name=KubeWorkerImage2
$ scyld-kube --image KubeWorkerImage2 --join --token ka8y8y.enwcyfsk4hb1ayz5 --
↪cahash sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --
↪cluster 10.154.4.0
$ scyld-bootctl -i KubeWorkerBoot2 up image=KubeWorkerImage2
$ scyld-nodectl -i n[15-20] set _boot_config=KubeWorkerBoot2
$ scyld-nodectl -i n[15-20] reboot
```

- On EACH non-ClusterWare system that you want to join as a worker and where `clusterware-kubeadm` is installed, join the local system to the control plane node kube2 (10.154.4.0) with explicit `--token`, `--cahash`, and `--cluster` arguments:

```
scyld-kube --join --token ka8y8y.enwcyfsk4hb1ayz5 --cahash_
↪sha256:413a6267bac67ff749734749dc8b5f60323a68c64bf7fc8e99292dd9b29040b2 --cluster_
↪10.154.4.0
```

5.6 Creating Arbitrary Rocky Images

Creating Images describes how to create images from the latest Rocky repos. This section describes how to create images from older Rocky repos. These examples use Rocky 9.5 and Rocky 8.5, but other versions will work similarly.

Note

To create a CentOS/Rocky 7 image on a RHEL or Rocky 8 head node, you must disable FIPS mode.

Important

If the Rocky image built is subsequently updated using `yum update`, then by default that updates packages to the latest minor release level, **not** to newer versions at the image's current minor release level. Also, `yum install` of

additional packages may update dependency packages from their current minor release version level to the latest minor level. Such actions may result in a mixture of packages from different minor releases, which may have unintended consequences.

5.6.1 Using Version-Specific ISO File

The first option is to use the ISO for the targeted version of Rocky. The ISO is used to create an ISO-based repo that is the basis for a distro. `scyld-modimg` is used to create the final image.

For example, use `scyld-clusterctl repos` and `scyld-clusterctl distros` with the `Rocky-9.5-x86_64-dvd.iso` ISO file to create a repo and distro for Rocky version 9.5, then use `scyld-modimg` to create an image and a boot config.

You can also use `scyld-add-boot-config` to perform the same result in fewer steps. Execute the following and accept all the defaults:

```
scyld-add-boot-config --iso /mnt/isos/Rocky-9.5-x86_64-dvd.iso
```

This creates a distro and repo both named `Rocky-9.5-x86_64-dvd`, and an image and boot config both named `Rocky-9.5-x86_64-dvd`.

Avoid the manual acceptance of the defaults by specifying desired names and running the command in batch mode:

```
scyld-add-boot-config --iso /mnt/isos/Rocky-9.5-x86_64-dvd.iso \
--image Rocky-9.5-Image --boot-config Rocky-9.5-Boot
```

View the result, which shows the default repo and the new repo as well as the default boot config and the new boot config:

```
[admin@head]$ scyld-clusterctl distros ls -L
Distros
  Rocky
    name: Rocky
    packaging: rpm
    release: 9
    repos
      Rocky_appstream
      Rocky_base

[admin@head]$ scyld-bootctl ls -l
Boot Configurations
  Rocky-9.5-Boot
    cmdline: enforcing=0
    image: Rocky-9.5-Image
    initramfs
      chksum: a85b01e91c26c52ebf549066c6c5fce544f3c75b
      filename: 3684fbadf53f4c8bb8a3dea24ecf778d
      mtime: 2025-01-18 16:49:06 UTC (1:14:23 ago)
      size: 33.4 MiB (34978448 bytes)
    kernel
      chksum: 73872862a49ee024bf44c4d796c96bed4d52ee43
      filename: 1d6888add971485395d943df191645c4
      mtime: 2025-01-18 16:49:07 UTC (1:14:23 ago)
      size: 6.4 MiB (6734016 bytes)
    last_modified: 2025-01-18 16:49:07 UTC (1:14:23 ago)
```

(continues on next page)

(continued from previous page)

```
name: Rocky-9.5-Boot
release: 3.10.0-1062.el7.x86_64
```

Note

Creating images from some older ISOs may produce an error message beginning with `ERROR: One or more repositories in the newly created image are invalid or unreachable`. The `scyld-modimg` tool automatically retries the image creation, and if there is no subsequent error reported, then you can assume that the resulting image is useable.

5.6.2 Using Publicly Available Repositories

To create the image from publicly available repositories, embed the specific version in the repo URLs. Note that the repositories for older versions of Rocky are now in the `/vault/` directory.

For example, to create a Rocky 8.5 image, use the following commands:

```
scyld-clusterctl repos create name=Rocky85_baseos urls=http://dl.rockylinux.org/vault/
↳rocky/8.5/BaseOS/$basearch/os/
scyld-clusterctl repos create name=Rocky85_appstream urls=http://dl.rockylinux.org/vault/
↳rocky/8.5/AppStream/$basearch/os/
scyld-clusterctl distros create name=Rocky85 repos=Rocky85_baseos,Rocky85_appstream
```

After the distro is created, use `scyld-modimg` to create the image. Alternatively, run the following command to create both the image and boot configuration:

```
scyld-add-boot-config --distro Rocky85
```

5.7 Creating Arbitrary RHEL Images

The *Creating Arbitrary Rocky Images* describes how to create and update images using arbitrary Rocky repos. This section describes how to create arbitrary RHEL images from older releases and register (or re-register) them to Red Hat. The repo is most commonly built from an ISO file that represents a specific RHEL *major.minor* version.

For this example we build a RHEL 9.0 image and boot config using the `rhel-computenode-9.0-x86_64-dvd.iso` ISO file. Older releases will work similarly.

Note

To create a RHEL 7 image on a RHEL or Rocky 8 head node, you must disable FIPS mode.

Use `scyld-clusterctl repos` and `scyld-clusterctl distros` to create a repo and distro for this RHEL version 9.0, then use `scyld-modimg` to create an image and a boot config.

More simply, use `scyld-add-boot-config` to perform the same result in fewer steps. Execute the following and accept all the defaults:

```
scyld-add-boot-config --iso /mnt/isos/rhel-computenode-9.0-x86_64-dvd.iso
```

This creates a distro and repo both named `rhel-server-9.0-x86_64`, and an image and boot config both named `rhel-server-9.0-x86_64`.

Alternatively, you can avoid the manual acceptance of the defaults by specifying desired names and running the command in batch mode:

```
scyld-add-boot-config --iso=/mnt/isos/rhel-server-9.0-x86_64-dvd.iso \
--image RHEL-9.0-Image --boot-config RHEL-9.0-Boot
```

View the result, which shows the default repo and the new repo, and the default boot config and the new boot config:

```
[admin@head]$ scyld-clusterctl distros ls -l
Distros
Rocky
  name: Rocky
  packaging: rpm
  release: 9
  repos
    Rocky_base

rhel-server-9.0-x86_64
  name: rhel-server-9.0-x86_64
  packaging: rpm
  release: none
  repos
    rhel-server-9.0-x86_64

[admin@head]$ scyld-bootctl ls -l
Boot Configurations
DefaultBoot
  cmdline: enforcing=0
  image: DefaultImage
  initramfs
    chksum: a623be752272166f47896d648689789359239ebf
    filename: b51e6d31a84a4f069c6a4a484b5b5264
    mtime: 2025-01-18 19:20:19 UTC (0:37:22 ago)
    size: 33.4 MiB (35046202 bytes)
  kernel
    chksum: 2b0b0737e80596021ef71da44dbac6b335fcf0e3
    filename: db392537a1f6445d8c821d9a89ea5d8c
    mtime: 2025-01-18 19:20:19 UTC (0:37:22 ago)
    size: 6.5 MiB (6777448 bytes)
  last_modified: 2025-01-18 19:20:19 UTC (0:37:22 ago)
  name: DefaultBoot
  release: 3.10.0-1160.59.1.el7.x86_64

RHEL-9.0-Boot
  cmdline: enforcing=0
  image: RHEL-9.0-Image
  initramfs
    chksum: 0d824541ab9bc9452dbec07e8486f443472327f9
    filename: 905309b474f54c629ac8befd76150f8b
    mtime: 2025-01-18 19:43:39 UTC (0:14:02 ago)
    size: 33.4 MiB (35046065 bytes)
  kernel
    chksum: b5d0b67026d6ae5829d929dcd7b6ba52619de7fb
    filename: 222a7267c85849979a8908bdc72277b1
```

(continues on next page)

(continued from previous page)

```

mtime: 2022-03-18 19:43:39 UTC (0:14:02 ago)
size: 6.4 MiB (6762800 bytes)
last_modified: 2022-03-18 19:43:39 UTC (0:14:02 ago)
name: RHEL-9.0-Boot
release: 3.10.0-1127.el7.x86_64

```

Important

If the cluster administrator wants to enable FIPS, then follow the directions provided by the base distribution provider. The Red Hat RHEL or Rocky repo **must** include `@core`, and any subsequently created compute node image must contain several additional packages, including `dracut-fips`. Verify the presence of `@core` by successfully executing `yum groupinfo core`.

To boot the new image, assign RHEL-9.0-Boot to node `n0`, and reboot `n0`:

```

[admin@head]$ scyld-nodectl -i n0 set _boot_config=RHEL-9.0-Boot
Results
n0
  success: True

[admin@head]$ scyld-nodectl -i n0 reboot
Nodes
n0: Soft reboot succeeded

```

A RHEL compute node can automatically register (or re-register) with Red Hat at boot time by adding the file `/etc/clusterware/rhel-vars.sh` to the image. That file must contain two lines that define values for the variables "RHEL_USER" and "RHEL_PASS". The booting RHEL node executes `/opt/scyld/clusterware-node/scripts-available/up/register_rhel.sh` (distributed in the `clusterware-node` package) which opens `/etc/clusterware/rhel-vars.sh` (if that exists) and parses the "RHEL_USER=" username and "RHEL_PASS=" password, then executes:

```
subscription-manager register --username <username> --password <password>
```

On a successful first-time registration, the node transmits the resulting `consumerid` to its parent head node, which in turn stores that value into the node's `_rhel_consumerid` attribute in the ClusterWare database.

If a specific Pool ID is required, then add the attribute `_rhel_poolid`.

Important

If the RHEL image thus built is subsequently updated using `yum update`, then by default that updates packages to the latest minor release level, **not** to newer versions at the image's current minor release level. Also, `yum install` of additional packages may update dependency packages from their current minor release version level to the latest minor level. Such actions may result in a mixture of packages from different minor releases, which may have unintended consequences.

5.8 Creating Ubuntu and Debian Images

The following examples create Ubuntu and Debian images and associated boot configurations using the public Internet-accessible Ubuntu and Debian repos, both of which contain multiple releases.

5.8.1 UBUNTU

First an example of building an Ubuntu 20.04 LTS (Focal Fossa) image. Specify a local repo, arbitrarily naming it *ubuntu*, that serves as a shorthand reference to the public Ubuntu repo:

```
scyld-clusterctl repos create name=ubuntu urls=http://archive.ubuntu.com/ubuntu/
```

Next, specify a particular distribution within that Ubuntu repo. For this example we specify *focal*, which is the Focal Fossa 20.04 LTS release, and give this local distro the name *ubuntu_20.04*:

```
scyld-clusterctl distros create name=ubuntu_20.04 repos=ubuntu release=focal
↳ packaging=deb
```

Now create an image from the distro *ubuntu_20.04* and name it *UbuntuImg-20.04*:

```
scyld-modimg --create ubuntu_20.04 --set-name UbuntuImg-20.04 --upload
```

And create a boot configuration named *UbuntuBoot-20.04* that provides the necessary files to PXE boot that image:

```
scyld-add-boot-config --image UbuntuImg-20.04 --boot-config UbuntuBoot-20.04
```

The image thus created contains basic Ubuntu 20.04 LTS software. You can add software and modify configuration files in this image as needed, keeping in mind that Ubuntu's package manager expects software distributed as **.deb* files, not **.rpm* files.

For example, use `--chroot`:

```
scyld-modimg -i UbuntuImg-20.04 --chroot
```

and manipulate files and packages within the image.

5.8.2 DEBIAN

You can employ similar steps to create a Debian image, in this example creating an image from the *stable* distro:

```
scyld-clusterctl repos create name=debian urls=http://deb.debian.org/debian/
scyld-clusterctl distros create name=debian-stable repos=debian release=stable
↳ packaging=deb
scyld-modimg --create debian-stable --set-name debian-stable-img --upload
scyld-add-boot-config --image debian-stable-img --boot-config debian-stable-boot
```

5.9 Converting CentOS 8 to Alternative Distro

CentOS 8 has reached its official End Of Life phase and now exists only in archived form at <https://vault.centos.org/>. To access software updates that track RHEL8 to one degree or another, you should convert to an alternative distribution.

Some alternatives choices are:

Red Hat RHEL 8

RHEL is the original source base of every CentOS release, so RHEL 8 is an obvious alternative to CentOS 8. Accessing the RHEL 8 repositories requires a paid subscription. Additionally, some RPMs found in the CentOS repository for a particular release are only found in the Red Hat EPEL repository. Contact Red Hat for details.

CentOS Stream 8

The CentOS Project recommends transitioning CentOS 8 to CentOS Stream 8. The CentOS Project defines that repository as containing RPMs that are in a development phase between a RHEL clone and the somewhat more experimental Fedora, i.e., as a form of "beta" release candidates for the next RHEL release.

Rocky 8

Penguin Computing currently suggests considering Rocky 8 as a distribution similar to CentOS, i.e., tracking every new RHEL 8 release within days. See <https://rockylinux.org/> for details.

See <https://github.com/rocky-linux/rocky-tools/tree/main/migrate2rocky> for details about a bash script that performs the conversion.

Note however that updating CentOS RPMs to Rocky RPMs will likely install updates of various configuration files, which will leave various `*.rpmsave` and `*.rpmnew` files that require the administrator to examine and potentially merge local changes that were made when running CentOS.

5.10 Using Docker for Head Nodes

A RHEL/CentOS-clone server can use the `scyld-install` tool to install ICE ClusterWare™ to become a ClusterWare head node. This appendix describes an alternative approach that allows a server to use a ClusterWare Docker container that Penguin Computing has already built as a basic head node image, thereby avoiding the need for a cluster administrator to install and configure the ClusterWare system from scratch.

Note

The ClusterWare platform also supports containers running on the compute nodes, allowing each node to act as a Docker host for running containers. See *Using Docker for Compute Nodes*.

5.10.1 Install the Foundational Packages

The ClusterWare Docker container approach first requires installing the `docker` and `clusterware-tools` packages. For the latter package you need to set up `/etc/yum.repos.d/clusterware.repo` in order to access the Penguin Computing repo. Instructions for how to do that can be found at the beginning of the same chapter (*Install ICE ClusterWare*) that describes how to perform an initial install of a full ClusterWare cluster.

Once `clusterware.repo` is in place, then you can install the packages necessary for the Docker container approach:

```
sudo yum install docker clusterware-tools
```

The `clusterware-tools` package contains the various `scyld-` commands, including `/usr/bin/scyld-containerctl`, which is referenced below. Knowledgeable Docker administrators may wish to use the standard Docker tools.

Note

The `podman` container management system can be used in place of `docker` if desired.

5.10.2 Download and Load the ClusterWare Docker Image

First download a copy of a pre-built Docker image onto the server that is appropriate for the head node you wish to create. For example, visit <https://updates.penguincomputing.com/clusterware/12/el8/container/> (with appropriate authentication) and view the available containers compatible with a RHEL/CentOS 8 base distribution that is already installed on the Docker host server. Suppose you choose `clusterware-12.1.0-g0000` to download. You can validate the downloaded file using the same general method used to validate a downloaded ISO (see *Validating ClusterWare ISOs*).

Load the downloaded image into the Docker image registry:

```
scyld-containerctl img-load clusterware-12.1.0-g0000
```

which will show several progress bars such as "Loaded image: localhost/clusterware:12.1.0-g0000". After loading, see just the ClusterWare image using:

```
scyld-containerctl img-ls
```

or see all Docker images using:

```
docker image list
```

5.10.3 Start the Container

Start the ClusterWare head node container on the Docker host server:

```
scyld-containerctl start
```

which creates a new storage directory for persisting the data (by default named `cw_container_storage`), then creates the container itself and starts it executing. You can verify that the container is executing using:

```
scyld-containerctl status
```

which will show only `clusterware` containers. To see all Docker containers:

```
docker ps
```

5.10.4 Configure the Container

The container needs to contain at least one admin account. For an admin account already defined on the Docker host, you can directly reference that admin's ssh key file with `@` prepended to the admin's public key file name, e.g.,:

```
scyld-containerctl add-admin admin1 @/home/admin1/.ssh/id_dsa.pub
```

For an admin not defined on the Docker host, you will need a copy of the admin's `id_dsa.pub` file contents. You should include that `<ssh-key>` string on the command line enclosed in quotes to ensure that spaces and other characters are sent appropriately. For example, for admin `admin2`:

```
scyld-containerctl add-admin admin2 'ssh-rsa AAA..1A2B3C='
```

Note that the ssh key should end with an equals (=) sign and an optional email address.

It may be helpful to set the root password of the container to a new, known value -- this would allow access to the web-UI, for example. Use the `root-pass` action:

```
scyl-d-containerctl root-pass
```

The system will prompt for a new password, and ask for it a second time to confirm. The `root-pass` action will also print out the database password which would be needed for configuring Grafana monitoring (see [Grafana Login](#)).

Now configure the `clusterID` in the container with the customer's cluster authentication token so that it has access to the ClusterWare repo:

```
scyl-d-containerctl cluster-id <AUTH_TOKEN>
```

Now configure the use of ClusterWare tools using:

```
[root@rocky4 ~]# scyl-d-containerctl tool-config
```

which will attempt to find a "good" IP address for this Docker host to communicate with the private cluster network, although the tool may be confused if there are multiple network interfaces.

The tool writes results to stdout; for example:

```
ClusterWare tools will attempt to contact ssh-agent to get the
user's authentication key. It may be worthwhile for users to run:
    eval `ssh-agent` ; ssh-add
```

A potential `.scyl-dcw/settings.ini` file is below:

```
[ClusterWare]
client.base_url = https://10.54.0.123/api/v1
client.authuser = root
client.sslverify = quiet
```

Validate the proposed `settings.ini` lines, modify if needed, and write to `~/scyl-dcw/settings.ini`. This user's `settings.ini` file can be sent to each admin that has been added to the container, who can use that file for their own `~/scyl-dcw/settings.ini` after modifying the `client.authuser = <username>` line with their own username.

Each user will need to execute `ssh-agent` on the Docker host server at login to allow the ClusterWare platform to authenticate that user's access to the `scyl-d-*` tools:

```
eval `ssh-agent` ; ssh-add
```

With `ssh-agent` running, an admin user can now execute ClusterWare commands. First try:

```
scyl-d-nodectl ls
```

If that authentication was successful, then because initially there are no nodes configured for the container, the above command should report `ERROR: No nodes found, nothing was done` and thus verifies the admin's proper access.

Since the container initially has no images or boot configurations by default, we can create them as with any other ClusterWare installation by executing:

```
scyl-d-add-boot-config --make-defaults
```

Similarly, the container initially has no defined networks or nodes defined, so those also need to be defined. For example, create a cluster config file called `cluster.conf`:

```
cat <<-EOF >cluster.conf
iprange 192.168.122.100/24
```

(continues on next page)

(continued from previous page)

```
node 52:54:00:00:00:01
node 52:54:00:00:00:02
EOF
```

which defines a cluster network of 192.168.122.100/24 that services two compute nodes on that network with the given MAC addresses. Now configure the head node with that config file:

```
scyld-cluster-conf load cluster.conf
```

You can confirm the configuration with `scyld-nodectl ls -l`, which should return node names `n0` and `n1` with IP addresses in the specified range.

5.10.5 Stopping and Restarting the Container

To stop the ClusterWare container:

```
scyld-containerctl stop
```

The output will give the name of the storage directory and image-version information. It will also give an example command to restart this container without loss of data, e.g., by executing:

```
scyld-containerctl start cw_container_storage clusterware:12.0.0-g0000
```

Note

The ClusterWare container may take more time to shutdown than Docker usually expects and may show a time-out warning. This is just a warning. The container will in fact be stopped, which you can confirm with `scyld-containerctl status` or `docker ps`.

If you are using the default storage location `cw_container_storage` and image version name, then you can restart the head node without loss of data by using the shorter:

```
scyld-containerctl start
```

From an admin account, tools like `scyld-nodectl ls` should now work, and any nodes that were previously defined will still be present.

5.10.6 The Container Storage Area

The container storage directory will become populated with copies of several directories from inside the container. Most of this data will be opaque and should *not* be tampered with. The `logs/` directory, however, might be of use in helping to debug or triage problems.

5.10.7 Known Issues

Depending on how the container manager is configured, the ClusterWare container may need extra networking privileges. In particular, user-created containers may not be allowed to access network ports below 1024. If `syslog` shows messages like:

```
httpd: Permission denied: AH00072: make_sock: could not bind to address [::]:80
```

then admins may need to configure the container-host machine to allow users to access lower-numbered ports. One can insert a new config file into `/etc/sysctl.d` to permanently lower the starting point for "unprivileged" ports. Since the ClusterWare platform needs access to DNS/port 53, the following will create the necessary file:

```
echo net.ipv4.ip_unprivileged_port_start = 53 | sudo tee /etc/sysctl.d/90-unprivileged_port_start.conf
```

A reboot of the container-host will be needed to load the new `sysctl` configuration.

API REFERENCE

The ICE ClusterWare™ Application Programming Interface (API) enables cluster management, monitoring, and provisioning using a series of HTTP requests. While this web API is typically accessed by using the provided web portal and command line tools, users can also create their own customized tools by calling the web API in their own scripts.

Aside from a couple of endpoints related to long-running background jobs, the ClusterWare API is stateless, meaning there is no need for the client to remember any previous state information.

The API is organized around the major ClusterWare objects:

- Nodes - `/nodes`
- Attribute Groups - `/attribs`
- Admins - `/admins`
- Login/Authentication endpoints
- Boot Configurations - `/bootconfigs`
- Images - `/images`
- Networks - `/nets`
- Hostnames - `/hosts`
- Dynamic Groups - `/dyngroups`
- Naming Pools - `/namingpools`
- Head nodes - `/heads`, `/database`
- Software Repositories - `/repos`
- Software Distributions - `/distros`
- Git Repositories - `/gitrepos`
- State Sets - `/nodes/waitfor`
- Cluster-wide information - `/cluster`

Each ClusterWare object type has several method endpoints for getting or setting data that can be accessed using an HTTP call to the correct endpoint. For example, an HTTP GET call to the `/nodes` endpoint will return a list of all known nodes.

Some method endpoints also accept a unique identifier (UID) as part of the URL. For example, an HTTP GET call to `/node/<UID>` will return details about the node with the matching UID. A UID is a 32-character string of lower-case letters and digits, such as `bf0f61d24ce84064a8c7c7e872332c07` or `e54e420c42214101918584e27382e8f5`.

Many method endpoints that accept a UID will also accept other identifiers in place of a UID. For example, calls to Admins endpoints can use a username in place of the UID and calls to the Nodes endpoints can accept a node name, MAC address, or IP address in place of the UID.

Data being sent to web API endpoints is expected to be encoded in JSON format. Usually this data will be an JSON object (hash-table or dictionary), though some endpoints accept a list or string instead.

The HTTP response from the server will usually be a JSON object. When an HTTP request is successful, the response will have an HTTP status code of 200, the JSON object will contain a "success" entry with a value of `true`, and any additional data can be found in the "data" entry. For example:

```
{ "success": true, "data": "4e75fa48...0f005bd" }
```

If the HTTP response fails, then a 400-series status code will be returned (e.g. 401 for invalid credentials; 403 for denied access; 404 for object not found, etc.), the "success" entry will have a value of `false`, and a "reason" entry will have a description of why the action failed. For example:

```
{ "success": false, "reason": "No node found for ID=n123" }
```

6.1 Authentication

The majority of ICE ClusterWare™ API endpoints require that an authentication token is included in the request. This authentication token will usually come from a call to the `login` or `token_refresh` endpoints. There are several authentication options for logging in:

- Username / Password authentication. This leverages the operating system's login credentials.
- SSH key authentication. This uses a public / private key pair. The private keys must be set inside the ClusterWare software.
- Local-socket authentication service. This only works if a user is logged in to the head node where the request is made.

When the login is successful, two JSON Web Tokens (JWTs; see <https://jwt.io/>) will be returned: an access token and a refresh token. The refresh token can be presented to the `token-refresh` endpoint to make a new access token.

Note

The ClusterWare command-line tools use the access token when possible and will automatically use the refresh token when the access token has expired. Admins writing their own tools will have to detect errors due to token expiration and manually initiate a refresh.

While there are ways to unpack and interpret the data inside a JWT, the tokens can be considered opaque. A client simply receives a JWT upon login, and then sends that same JWT inside an authorization header on future requests:

```
Header: "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
```

The access token has an embedded expiration time (which is also sent by the ClusterWare platform during the login process) and after that time, the token will be rejected by the head node. The refresh token has its own expiration time, usually much longer than the access token. So the expectation is that when the access token expires, the refresh token will still be valid and can be used to make a new access token.

Note

While the refresh token is ONLY usable for making a new access token, it cannot be used to interact with the other endpoints.

In a multi-head cluster, every head node may be configured to use different underlying “secrets” that are used when making new tokens. Thus, after authenticating to one head node, the tokens that are returned may not be usable on other head nodes. In the rare case that a client needs to interact with multiple head nodes, it should login to each one separately and ensure that future requests use the correct token with each corresponding head node.

Where authentication asserts that “this user is who they say they are”, the ClusterWare platform uses role-based access control (RBAC) to limit what specific actions a given user can take. For more information on ClusterWare’s RBAC roles, see *Role-Based Access Controls* and *Role-Based Access Control System*.

Note

As with other token-based authentication systems, if anyone intercepts or acquires a copy of a token, they can fully impersonate the user who created it. Care must be taken to protect the tokens from other users. Consider using OS or file-system based access control mechanisms.

6.1.1 Username/Password Authentication

The traditional username/password process is provided through a PAM connector. When a login request is made, the system is queried to see if the given username and password are valid. If they are, the ClusterWare platform will make a new token and return it to the caller.

The data is sent as a JSON object with "user" and "pass" keys:

```
curl -X POST https://head1.cluster.local/api/v1/login --data '{"user":"admin1", \
"pass":"password"}'
```

Note that these example curl commands are interacting with an HTTP endpoint, but they are also available using HTTPS. Further, the examples may show passwords on the command line, but putting a password on the command line is not recommended as it may be visible to other users via ps or similar commands.

The returned value will include the access and refresh tokens, as well as expiration values for both:

```
{"success": true, "data": {"user": "admin1", "token": {"access_token": \
"eyJhbGciOiJIUzI1Ni...", "expires_in": 1200, "refresh_token": \
"eyJhbGciOiJIUzI1Ni...", "refresh_expires_in": 2592000}}}
```

The expiration times are in seconds from when the server issued the token. For long-running actions or scripts, admins may want to err on the side of caution and refresh prior to the actual expiration of the access token.

Note

While the head nodes synchronize data amongst themselves, the underlying operating systems do not synchronize unless configured to do so via some other mechanism. The head nodes could have their /etc/passwd and /etc/shadow files synchronized through scp or sftp, or through an external directory system (LDAP, NIS), but that is outside the scope of this document.

6.1.2 Token Refresh

The access token can be used in subsequent requests up until the expiration time. Once it has expired, the refresh token can be used to acquire a new access token through the token-refresh endpoint:

```
curl -X GET https://head1.cluster.local/api/v1/newtoken -H "Authorization: \
Bearer eyJhbGciOiJIUzI1Ni..."
```

Again, note that this should be the refresh token that is sent to this endpoint – sending an access token will result in a failure.

Upon success, a new set of tokens will be returned:

```
{"success": true, "data": {"authorized_by": "admin1", "userid": "admin1", "uid": \
  "735367a122664db8ae8ba3ec113f1643", "token": {"access_token": \
  "eyJhbGciOiJIUzI1Ni...", "expires_in": 1200, "refresh_token": \
  "eyJhbGciOiJIUzI1Ni...", "refresh_expires_in": 2592000}}}
```

If the authentication fails, the "success" field will be false and a "reason" field will contain more information.

Note

The default access token duration is 20 minutes; the default refresh token duration is 30 days.

6.1.3 Alternate Authentication Methods

There are two other authentication methods that can be used during login – a one-time password (OTP) sent to a local socket, and SSH-key authentication. For more information, please contact Penguin Solutions.

Since head nodes are assumed to be “locked down” and protected from end-user access, when a client is running on the head node, they can send an OTP to a known Unix socket and then re-send that OTP during the login process. The same POST /login endpoint is used, but with “user” and “secret” keys. If successful, the ClusterWare platform will return a set of access and refresh tokens.

For information on the SSH-key login process, please contact Penguin Solutions.

6.2 Basic Operations

For most ICE ClusterWare™ objects, standard HTTP actions are supported for manipulating the entries. Using the Node objects as an example:

- List objects - GET /nodes
- Create new object - POST /nodes
- Get object info - GET /node/<UID>
- Update some info - PATCH /node/<UID>
- Update all info - PUT /node/<UID>
- Delete object - DELETE /node/<UID>
- Metadata - GET /nodes/meta

Endpoints that refer to a whole class of objects are pluralized (end with ‘s’), such as /nodes or /images. Endpoints that refer to one specific object are singular and include a UID, such as /node/<UID> or /image/<UID>.

6.2.1 List Objects

Issuing an HTTP GET to an object-class (pluralized) endpoint will return information about all the objects of that class. Since this endpoint will only return a list of UIDs for the objects, any detailed information will have to be found with a subsequent call to the GET endpoint.

The return value will be a JSON object with two main fields: success and data. The data field will also be a JSON object, with the keys being set to object UIDs:

```
{ "success": true, "data": { "7ad57...73e4a1f4": {... first object's data ...}, \
  "ab89c...2501817": {... second object's data ...}}}
```

If the request is unsuccessful, then a reason field will be returned with more information on why the request failed.

```
{ "success": false, "reason": "No node found for ID=n123" }
```

6.2.2 Create New Object

To create a new object, a POST is issued to the object-class (pluralized) endpoint with the relevant data. Most object types accept a name and "description" field, but most fields are object-specific (see below for details).

To create a new compute-node object:

```
curl -X POST https://head1.cluster.local/api/v1/nodes --data \
  '{"mac":"11:22:33:44:55:66"}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
```

On success, the ClusterWare platform will return a UID for the newly create node:

```
{ "success": true, "data": "4e75fa48...0f005bd" }
```

Upon failure, the returned data will include a "reason" field with more information on why the request failed:

```
{ "success": false, "reason": "Node with MAC=52:54:00:00:00:03 already exists." }
```

6.2.3 Get Object Info

Issuing an HTTP GET to an endpoint will return information about a specific object. Depending on the object type, the UID may be the true UID for the object, the name of the object, or some other object-specific unique identifier such as MAC address (for nodes).

The return value will be a JSON object with two main fields, success and data. The data field will also be a JSON object with a single key-value pair. The key will usually be the object's UID, but if the request was made with the object's name, then the key will also be the name.

```
{ "success": true, "data": { "7ad57...73e4a1f4": { ... object data ... }}}
```

By default, GET operations will return all of the object's information: the UID, the name, and all object-specific fields and sub-fields. The client is expected to provide any filtering of data that might be needed.

If the request is unsuccessful, then a reason field will be returned with more information on why the request failed.

```
{ "success": false, "reason": "No node found for ID=n123" }
```

6.2.4 Update Object

There are two related update methods: PATCH and PUT. When using the PATCH method, the request is assumed to modify only those fields that are sent in the request – any other fields in the object will be left as-is. The PUT method assumes that the entire object should be overwritten by the new data – any fields in the request will be overwritten, and any fields not in the request will be removed or set to default values. Not all object-types allow for PUT operations, and note that a bad PUT call could irreversibly damage the object if some fields are wiped out.

To update the "description" field of a node with PATCH:

```
curl -X PATCH https://head1.cluster.local/api/v1/node/UID --data \  
 '{"description":"this is a new description"}' -H "Authorization: Bearer \  
 eyJhbGciOiJIUzI1Ni..."
```

The return value will simply indicate success or failure:

```
{"success": true}
```

In this example, success would indicate that ONLY the "description" field has been modified.

On failure, a "reason" field will give more information on why the request failed:

```
{"success": false, "reason": "Unhandled parameter(s): ..."}
```

6.2.5 Delete Object

The DELETE method will permanently remove an object from the ClusterWare system – there is no way to recover the object's data once it has been deleted.

To delete an Image from the system:

```
curl -X DELETE https://head1.cluster.local/api/v1/image/DefaultImage -H \  
 "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
```

The return value will simply indicate success or failure:

```
{"success": true}
```

On failure, a "reason" field will give more information on why the request failed:

```
{"success": false, "reason": "No image found for ID=DefaultImage"}
```

6.2.6 Metadata Information

All objects have a metadata endpoint off of their object-class (pluralized) endpoint (for example, /nodes/meta). This endpoint will return more detailed information about the fields available in that object-class.

Note that this is a public endpoint and does not need any authentication token to be accessed.

```
curl -X GET https://head1.cluster.local/api/v1/nodes/meta
```

It will return a JSON object that includes a list of fields. Each field in the list will include a path field (the chain of keys to get to that field), summary, source (where the data comes from), and a format (what kind of data it is).

6.3 Admin Objects

Admin objects are used to give explicitly specified users access to ICE ClusterWare™ functionality. Regular HPC users do not need to have Admin objects created for them - this is solely for those who will do administrative or managerial tasks on the cluster.

To create an Admin, the only required data is a name. This must match the username of the user on the underlying operating system. If additional user-data is required, like a full name or department affiliation, it should be stored inside the "description" field.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field. Thus, once a user, user1, has been created, it can be referenced through /admin/user1 or /admin/<UID>.

6.3.1 Data Fields

Admin objects have several fields:

name	Required: The name of the user on the underlying system description Optional: A text string with descriptive information
roles	Optional: A comma-separated list of ClusterWare roles; while optional, if a user does not have any roles assigned, they will not be able to take any actions; note that PATCH operations on the "roles" field will overwrite the entire field, there is no way to append or extend the list of roles
keys	Optional: A list of one or more ssh keys
gui_settings	Optional: Used internally by the ClusterWare GUI; it should not be modified by end-users

6.3.2 Additional Endpoints

Several endpoints can be used to modify the gui_settings field. Where the standard PATCH actions can update just the gui_settings field, it must update the entire object; these actions allow finer-grained updating of individual keys in that object. Note that this field is used internally by the ClusterWare GUI and admins should not need to modify it directly.

```
GET /admin/<UID>/gui_settings
PATCH /admin/<UID>/gui_settings
DELETE /admin/<UID>/gui_settings
```

Several endpoints can be used to update or delete the list of SSH keys that have been stored for the admin.

```
POST /admin/<UID>/keys
PUT /admin/<UID>/keys
```

(continues on next page)

(continued from previous page)

```
DELETE /admin/<UID>/keys
```

Similar to the token-refresh process, there is an endpoint that allows an admin to make a new token for use by other tools or automated processes; optional fields can be included in the request:

```
POST /admin/<UID>/newtoken timeout: integer; duration for the newly made token
```

6.3.3 Example

First, create a new admin:

```
curl -X POST https://head1.cluster.local/api/v1/admins --data '{"name":"admin2"}' \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "d763705495c3423083ae35f0850da018"}
```

Get the details on that Admin record:

```
curl -X GET https://head1.cluster.local/api/v1/admin/d763705495c3423083ae35f0850da018 \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"name": "admin2", "roles": ["role.authuser", \
  "role.fulladmin"], "last_modified": 1721850600.3619504, "last_modified_on": \
  "head23.cluster.local", "last_modified_by": "admin1", "uid": \
  "d763705495c3423083ae35f0850da018"}}
```

Update the record to include a description (switching to “admin2” in the URL):

```
curl -X PATCH https://head1.cluster.local/api/v1/admin/admin2 --data \
  '{"description":"John Doe, HPC admin"}' -H "Authorization: Bearer \
  eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

Update the list of roles to be just “role.authuser”:

```
curl -X PATCH https://head1.cluster.local/api/v1/admin/admin2 --data \
  '{"roles":["role.authuser"]}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

While the PATCH request only updates that one field, leaving other fields as-is, the update to that field will overwrite the old data. There is no way to append or add to the list of roles.

Verify the new data:

```
curl -X GET https://head1.cluster.local/api/v1/admin/admin2 -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"name": "admin2", "roles": ["role.authuser"], \
  "last_modified": 1721851033.3245502, "last_modified_on": "head23.cluster.local", \
  "last_modified_by": "admin1", "description": "John Doe, HPC admin", "uid": \
  "d763705495c3423083ae35f0850da018"}}
```

Finally, delete the account:

```
curl -X DELETE https://head1.cluster.local/api/v1/admin/admin2 -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

6.4 Node Objects

ICE ClusterWare™ node objects represent the computational nodes and other controllable entities inside the cluster (switches, PDUs, etc.).

When issuing requests, the UID field in the URL can be either the actual UID of the object, the name of the object as given in the "name" field, the MAC address, or the IP address. Thus, once a user, user1, has been created, it can be referenced through /admin/user1 or /admin/<UID>.

6.4.1 Data Fields

Node objects can have several fields:

name	Assigned by ClusterWare; the name is computed from the naming pool pattern and the node's index (which is also computed based on that pool's membership)
description	Optional: A text string with descriptive information
type	Optional: Defaults to 'compute', can be one of ('unknown', 'compute', 'head', 'simulated')
mac	Required: Every node must have a MAC address
attributes	Optional: A set of key-value pairs that are assigned to the node and may be used when configuring the node at boot-time
groups	Optional: An ordered list of Attribute Groups that this node belongs to; attributes from each group will be applied (overwritten) based on the order of the groups
naming_pool	Optional: A naming pool may be assigned and used to calculate the name, index, and IP address of the node
index	Assigned by ClusterWare; the index (integer) is computed by ClusterWare based on the naming pool
ip	Assigned by ClusterWare; the IP address is computed by ClusterWare based on the naming pool

(continues on next page)

(continued from previous page)

`power_uri`

Optional: The URI of the node's BMC or IPMI controller; the `power_uri` can be a template with bracketed fields that are substituted with the node's attribute, status, or hardware values (see examples below)

`redfish_uri`

Optional

6.4.2 Additional Endpoints

Several endpoints provide finer-grained access to the list of Attribute Groups that the node belongs to. For example, `POST /node/<UID>/groups` allows appending one or more groups to the current list. To completely replace the list of groups, use the `PATCH /node/<UID>` action with the `groups` key.

`GET /node/<UID>/groups`

Returns a list of Attribute Groups (UIDs) that the node belongs to

`POST /node/<UID>/groups`

Accepts one key or a list of keys representing Attribute Groups that will be appended to the current list

`DELETE /node/<UID>/groups`

Accepts one key or a list of keys representing Attribute Groups that will be removed from the current list of Attribute Groups

Several endpoints provide finer-grained access to the list of attributes assigned to this specific node. Issuing a `POST /node/<UID>/attributes` action will append one or more groups to the current list. To completely replace the attributes, use the `PATCH /node/<UID>` action with the `attributes` key. Note that nodes will inherit attributes from all groups that they belong to, and will then overwrite them with any node-specific attributes.

`GET /node/<UID>/attributes`

Returns a JSON object for all of the attributes (key-value pairs) for this node; these may have assigned specifically to this node or may have been inherited from any of the Attribute Groups that it belongs to

`PUT /node/<UID>/attributes`

Replaces the current set of node-specific attributes with the sent data; note that if a key is removed from the current set, it may still be inherited from a joined Attribute Group

`PATCH /node/<UID>/attributes`

Updates the current set of node-specific attributes with the sent data; these values will take precedence over any joined Attribute Groups

`DELETE /node/<UID>/attributes`

Accepts one key or a list of keys to be removed from the node-specific attributes; note that if an attribute is set by an Attribute Group, it will still be present in the node's overall list of attributes (until it is removed from that Attribute Group or the node leaves that group)

Several endpoints are designed for power control functionality. The `power_uri` field is needed for deeper power control functions. For example, a power-off action will attempt to execute a graceful shutdown, but if that fails it will fallback

to an IPMI or BMC power-off.

GET /node/<UID>/power_uri

Returns a JSON object with “parsed” and “unparsed” keys; the “unparsed” key contains a single URI that can be used to connect to the power control unit; the “parsed” key contains an object with fields representing the server, path, etc. that have been parsed from the URI

GET /node/<UID>/power_state

Returns the power state of the node: on, off, unknown

PUT /node/<UID>/power_state

Accepts an object with “state” and “force” fields (all optional); “state” is one of: power on, power off; “force” is True or False. There is an advanced option, “steps”, which allows for more control over the sequence of power-control attempts (for example, hard versus soft power-off) - contact Penguin Solutions for more information.

Several endpoints are used for the node status and hardware reporting system. Note that this is separate from the ClusterWare Monitoring and Alerting system; this data is kept inside the ClusterWare database and can be used in various templates and node-selectors. The specific values to be found in status and hardware can be configured using a plugin system. More information on monitoring and the plugin system can be found in the ClusterWare Admin and Reference Guides.

GET /node/<UID>/status

Returns a JSON object with the node’s status information - this can be a sizeable set of data (several KB); with default plugins, status will include things like load average (load_avg) and memory usage (ram_free) as well as a list of the loaded packages and modules and many more.

POST /node/putstatus

Accepts a JSON object with a “uid” field (required) and “status” and/or “hardware” keys; the “status” or “hardware” keys are JSON objects containing the actual data. This endpoint is intended to be used by ClusterWare tools to update node status information.

POST /node/putattrs

Accepts a JSON object with a “uid” field (required) and a set of key-value pairs; the key-value pairs will be added to the node’s current attributes. This endpoint is intended to be used by ClusterWare tools to update node attribute information.

Related to the above, the following two endpoints are used by the compute nodes to update information on the ClusterWare head nodes. These endpoints do not use token-based authentication, but instead look at the IP-address of the incoming connection to determine which node is being updated.

Caution

Improper use of these endpoints could corrupt the ClusterWare node information, which could lead to improper functioning of Dynamic Groups, node selectors, and the Publish-Subscribe system. Contact Penguin Solutions for more information.

PUT /node/putstatus

(continues on next page)

(continued from previous page)

Accepts a JSON object with “status” and/or “hardware” keys, each of which is a JSON object containing the actual data; unlike the POST version of this endpoint, there is no “uid” key needed with the PUT operation

PUT /node/putattribs

Accepts a JSON object with key-value pairs; these key-value pairs will be added to the node’s attributes; unlike the POST version of this endpoint, there is no “uid” key needed with the PUT operation

To interact with the nodes, including running commands on them, several endpoints can be used:

PUT /node/<UID>/exec

Accepts a JSON object with “cmd” (required) and an optional “stdin” key; if “stdin” is sent, it will be used as the standard input for an interactive command; the output will be an octet-stream with the output from the command. Note that this output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the raw data from the command.

6.4.3 Example

Create a new node:

```
curl -X POST https://head1.cluster.local/api/v1/nodes --data \
  '{"mac":"11:22:33:44:55:66"}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "341b9947f6e644b78f6d88f5d7f898f4"}
```

Verify the node’s data using the returned UID:

```
curl -X GET https://head1.cluster.local/api/v1/node/341b9947f6e644b78f6d88f5d7f898f4 \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"mac": "11:22:33:44:55:66", "attributes": \
  {"_boot_config": "DefaultBoot"}, "index": 3, "ip": "192.168.122.103", \
  "type": "compute", "last_modified": 1721997178.5100713, "last_modified_on": \
  "head23.cluster.local", "last_modified_by": "admin1", "uid": \
  "341b9947f6e644b78f6d88f5d7f898f4", "groups": [], "hardware": {}, \
  "power_uri": null, "name": "n3", "hostname": "n3", "domain": "cluster.local"}}
```

Since Nodes can be referenced by name or MAC address, the same information can be found using:

```
curl -X GET https://head1.cluster.local/api/v1/node/11:22:33:44:55:66 -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
```

And now that the node name is known, the same information can be found with:

```
curl -X GET https://head1.cluster.local/api/v1/node/n3 -H "Authorization: Bearer \
  eyJhbGciOiJIUzI1Ni..."
```

Set attributes on the node itself (not through an Attribute Group):

```
curl -X PATCH https://head1.cluster.local/api/v1/node/n3/attributes --data \
  '{"ipmi_user":"service", "ipmi_pass":"servicepass"}' -H "Authorization: \
  Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

Verify the new data, grabbing just the attributes (not the whole node record):

```
curl -X GET https://head1.cluster.local/api/v1/node/n3/attributes -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"_boot_config": "DefaultBoot", "ipmi_user": "service", \
  "ipmi_pass": "servicepass"}}
```

Setting the power_uri using the attributes in the template

```
curl -X PATCH https://head1.cluster.local/api/v1/node/n3 --data \
  '{"power_uri": "ipmi://<ipmi_user>:<ipmi_pass>@10.10.1.3/ipmi"}' \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

Verify the power_uri and its substitutions

```
curl -X GET https://head1.cluster.local/api/v1/node/n3/power_uri -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"unparsed": "ipmi://service:servicepass@10.10.1.3/ipmi", \
  "parsed": {"scheme": "ipmi", "schemes": "", "server": "10.10.1.3", "path": \
  "ipmi", "server_username": "service:servicepass", "username": null, \
  "password": null, "host": "ipmi"}}
```

Notice that the text strings “<ipmi_user>” and “<ipmi_password>” have been substituted for the values from the node’s attributes. To get the un-substituted value for the power_uri, simply use the GET /node/<UID> endpoint.

To determine the power state of the node:

```
curl -X GET https://head1.cluster.local/api/v1/node/n3/power_state -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "on"}
```

Note that if the node is unreachable via its primary or management network, it may take a while for the system to determine that the node is down just due to standard network timeouts.

To power off the node:

```
curl -X PUT https://head1.cluster.local/api/v1/node/n3/power_state --data \
  '{"state": "power off"}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

6.5 Attribute-Group Objects

An Attribute Group object represents a set of attributes that will be attached to a node based on its membership in the group. A node can be a member of several groups, arranged in a hierarchical list. For example, a node that is a member of group1 and group2 will inherit group1’s attributes, which may then be overwritten by group2’s attributes.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the “name” field.

6.5.1 Data Fields

The Attribute Group fields are:

```

name
    Required: The name for the group

description
    Optional: A text string with descriptive information

attributes
    Optional: A JSON-object containing key-value pairs of attributes

```

While the `attributes` field accepts a generic JSON object, it can only reference one level into that key-value store.

6.5.2 Additional Endpoints

Several endpoints can be used to directly manipulate the attribute key-value store instead of going through the Attribute object first. The `PATCH /attrs/<UID>` command allows for overwriting the entire `attributes` field, but cannot be used to add or append attributes one at a time. The `PATCH /attrs/<UID>/attributes` endpoint allows for that kind of direct update.

```

GET /attrib/<UID>/attributes
    Returns a JSON object containing the key-value pairs

PUT /attrib/<UID>/attributes
    Replaces the current set of key-value pairs with the sent data; any keys not in
    the sent data will be removed from the Attribute Group

PATCH /attrib/<UID>/attributes
    Updates the current set of key-value pairs with the sent data; only the keys that
    are sent will be modified in the Attribute Group's data; any other keys will be
    left as-is

DELETE /attrib/<UID>/attributes
    Accepts one key or a list of keys to be removed from the Attribute Group's data;
    note that a node can still show that attribute key as present if a node-specific
    attribute has been set, or if the node is joined to another Attribute Group where
    it is set

```

6.5.3 Example

Create a new Attribute Group:

```

curl -X POST https://head1.cluster.local/api/v1/attrs --data '{"name":"MyAttrs"}' \
-H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "5997e993e0bb49568bef18536614b733"}

```

Read the basic information:

```

curl -X GET https://head1.cluster.local/api/v1/attrib/MyAttrs -H \
"Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"name": "MyAttrs", "last_modified": 1721852251.1698143, \
"last_modified_on": "head23.cluster.local", "last_modified_by": "admin1", \
"uid": "5997e993e0bb49568bef18536614b733", "attributes": {}}}

```

Update the attributes key-value store through the main record:

```
curl -X PATCH https://head1.cluster.local/api/v1/attrib/MyAttribs --data \
  '{"attributes":{"foo":"bar"}}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

Verify that the update worked:

```
curl -X GET https://head1.cluster.local/api/v1/attrib/MyAttribs -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"name": "MyAttribs", "last_modified": 1721852433.4113355, \
  "last_modified_on": "head23.cluster.local", "last_modified_by": "admin1", \
  "attributes": {"foo": "bar"}, "uid": "5997e993e0bb49568bef18536614b733"}}
```

To get just the attributes and not the whole record:

```
curl -X GET https://head1.cluster.local/api/v1/attrib/MyAttribs/attributes -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"foo": "bar"}}
```

Directly manipulate the attributes key-value store to add new data:

```
curl -X PATCH https://head1.cluster.local/api/v1/attrib/MyAttribs/attributes --data \
  '{"abc":"def"}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

Verify the new data:

```
curl -X GET https://head1.cluster.local/api/v1/attrib/MyAttribs/attributes -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"foo": "bar", "abc": "def", "last_modified": \
  1721852648.6145608, "last_modified_on": "533a8c21fd5642c38138214d7ad783ae", \
  "last_modified_by": "admin1"}}
```

Directly delete one key out of the key-value store (send a list of keys to delete):

```
curl -X DELETE https://head1.cluster.local/api/v1/attrib/MyAttribs/attributes --data \
  '["abc"]' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

And verify that the key is gone:

```
curl -X GET https://head1.cluster.local/api/v1/attrib/MyAttribs/attributes -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"foo": "bar", "last_modified": 1721852648.6145608, \
  last_modified_on": "533a8c21fd5642c38138214d7ad783ae", "last_modified_by": \
  "admin1"}}
```

To delete the object entirely:

```
curl -X DELETE https://head1.cluster.local/api/v1/attrib/MyAttribs -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

6.6 Boot Config Objects

A Boot Config object represents a set of information used to boot up a node from power-on to a running OS image on the node. It may include specific command-line parameters for the booting node, a “boot style” (for example, ram-based image or disked image), kickstart files, etc.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the “name” field.

6.6.1 Data Fields

The Attribute Group fields are:

name	Required: The name for the Boot Config
kernel	Required: This is the kernel (file) being used by the Boot Config; note that since this requires a large file to be uploaded, the standard POST/PATCH endpoints will not work to update the kernel - additional endpoints are provided for handling these files
initramfs	Required: This is the initramfs (file) being used during the early boot phases; note that since this requires a large file to be uploaded, the standard POST/PATCH endpoints will not work to update the initramfs - additional endpoints are provided for handling these files Optional: A text string with descriptive information
frozen	Optional: A True/False indicator that allows admins to block changes to the boot config; note that is intended as a protection against accidents, not against malicious attacks, since one can simply set frozen=False and then make changes.
cmdline	Optional: Command-line to be used when booting the operating system; this may include options for SELinux enforcement or power-control
image	Optional: A UID or name for a ClusterWare Image object (see below); note that ClusterWare stores this as a UID, but will accept a UID or name when creating or updating the Boot Config
boot_style	Optional: A string representation of the type of boot-up process being used, one of: ('rwram', 'roram', 'iscsi', 'next', 'disked', 'sanboot', 'live'); the default is “rwram” and indicates a node that will be booted into a ram-based image that readable and writable
repo	Optional

(continues on next page)

(continued from previous page)

kickstart

Optional: Indicates the kickstart file to be used for disk-based installations that are using Kickstart; this should be a file that has been copied or created in the `/opt/scyld/clusterware/kickstarts` directory.

6.6.2 Additional Endpoints

Additional object-class endpoints are available:

GET /bootconfigs/errors

Will check all Boot Config entries that they have valid data, that any referenced images or files are present, etc.; returns a JSON object with a “success” field; if an error occurs, a “reason” field will give more information on why it failed

GET /bootconfigs/mkiso

Accepts a JSON object with “uid” and “image” keys, and will create and download a bootable ISO image. This is an advanced feature, please contact Penguin Solutions for more information.

As mentioned earlier, the kernel and initramfs may be large files and thus must be uploaded separately – they cannot be “updated” with the standard POST/PATCH actions. To interact with these files, use the following endpoints:

GET /bootconfig/<UID>/kernel

Returns a file containing the kernel being used by this Boot Config. Note that the output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the binary data of the file being requested.

PUT /bootconfig/<UID>/kernel

Replaces the current kernel file with the uploaded file

DELETE /bootconfig/<UID>/kernel

Deletes the kernel file associated with this Boot Config; in multi-head configurations, the file will be deleted from all heads automatically

GET /bootconfig/<UID>/initramfs

Returns a file containing the initramfs being used by this Boot Config. Note that the output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the binary data of the file being requested

PUT /bootconfig/<UID>/initramfs

Replaces the current initramfs file with the uploaded file

DELETE /bootconfig/<UID>/initramfs

Deletes the initramfs file associated with this Boot Config; in multi-head configurations, the file will be deleted from all head nodes automatically

For kickstart-based deployments, the kickstart file can be a template that uses “<variable>” syntax to substitute values from the node. To retrieve the parsed and substituted kickstart file, use the following endpoint. Unlike kernel and initramfs files, kickstart files are NOT automatically synchronized across head nodes – admins must manually copy the file from one head node to the others.

GET /bootconfig/<UID>/kickstart

(continues on next page)

(continued from previous page)

Returns the parsed and substituted kickstart file; the output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the (text) data of the file being requested.

Since Boot Configs are critical to the operation of the cluster, an endpoint can be used to check that the required data is available and valid. For example, to check that the image file being used is actually present:

```
GET /bootconfig/<UID>/errors
Checks that entries in the boot config are valid and return a JSON object
with a “success” field; if an error occurs, a “reason” field will give more
information on why it failed
```

An endpoint is provided to export a Boot Config for backup or sharing:

```
GET /bootconfig/<UID>/export
This is an advanced feature, please contact Penguin Solutions for more
information
```

6.6.3 Example

Create a new Boot Config, using an existing DefaultImage as the base image:

```
curl -X POST https://head1.cluster.local/api/v1/bootconfigs --data \
'{"name":"NewBoot","image":"DefaultImage"}' -H \
"Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "ed31013d670745139681c46cf657ed40"}
```

Note that the image can be referenced by UID or, as shown here, by name. Verify the data:

```
curl -X GET https://head1.cluster.local/api/v1/bootconfig/NewBoot -H \
"Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"name": "NewBoot", "image": "DefaultImage", \
"last_modified": \1721921837.5399132, "last_modified_on": "head23.cluster.local", \
"last_modified_by": "admin1", "uid": "ed31013d670745139681c46cf657ed40"}}
```

Set it to be “frozen” so that future changes cannot be made:

```
curl -X PATCH https://head1.cluster.local/api/v1/bootconfig/NewBoot --data \
'{"frozen":true}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

Now that it is frozen, attempt to modify the “description” field:

```
curl -X PATCH https://head1.cluster.local/api/v1/bootconfig/NewBoot --data
'{"description":"new description"}' -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": false, "reason": "Cannot modify frozen boot configuration."}
```

This Boot Config does not yet have a kernel or initramfs and thus would not be able to actually boot any nodes yet. Using the existing DefaultBoot (created during installation), download its kernel and initramfs:

```
curl -X GET https://head1.cluster.local/api/v1/bootconfig/DefaultBoot/kernel -o \
defaultboot_kernel -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
```

(continues on next page)

(continued from previous page)

```
curl -X GET https://head1.cluster.local/api/v1/bootconfig/DefaultBoot/initramfs -o \
defaultboot_initramfs -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
```

6.7 Image Objects

An Image object represents a file system that will be deployed onto a compute node. It is not intended to have absolutely all software that an end-user might want, but rather, a somewhat minimal set of drivers, configuration files, tools, and middleware to bring up the node. Network-mounted file systems may be used to provide additional software and libraries for end-users.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field.

6.7.1 Data Fields

The Attribute Group fields are:

name	Required: The name for the image
content	Required: This is the content of the file system that will be used in this Image; since this requires a large file to be uploaded, the standard POST/PATCH endpoints will not work to update the content - additional endpoints are provided for handling it below
description	Optional: A text string with descriptive information
frozen	Optional: A True/False indicator that allows admins to block changes to the boot config; this is intended as a protection against accidents, not against malicious attacks, since one can simply set frozen=False and then make changes.
parent	Optional: A UID or name of a "parent" Image object that the current object was derived from; this field is for informational purposes only and is not used during any actions (for example, changes to a parent Image are not propagated into its children); ClusterWare stores this as a UID, but will accept a UID or name when creating or updating the Image
distro	Optional: A UID of a distro (a Software Distribution object)

6.7.2 Additional Endpoints

As mentioned earlier, the content of an Image is a large file and thus must be uploaded separately – it cannot be “updated” with the standard POST/PATCH actions. To interact with the content file, use the following endpoints:

```
GET /image/<UID>/content
Returns a file containing the file system image being used by this Image object.
```

(continues on next page)

(continued from previous page)

The output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the binary data of the file being requested

PUT /image/<UID>/content

Replaces the current content file with the uploaded file

DELETE /image/<UID>/content

Deletes the content file associated with this Image; in multi-head configurations, the file will be deleted from all heads automatically

GET /image/<UID>/content/stat

Returns a JSON object containing statistics on the image: the file’s size (“size”) and last modification time (“mtime”)

GET /image/<UID>/content/info

Returns a JSON object containing more detailed statistics on the image: the file’s size (“size”), last modification time (“mtime”), checksum (“chksum”), a reference count (“refcnt”), and the file format (“fmt”, currently “cwsquash”)

There is one additional endpoint that is used when capturing an image from a running compute node. This is an advanced feature; please contact Penguin Solutions for more information.

GET /image/{uid}/capture

Accepts a JSON object with “uid” key (required), and one or more optional keys: “exclude”, “credentials”, “compress”.

6.7.3 Example

Create an empty image:

```
curl -X POST https://head1.cluster.local/api/v1/images --data '{"name":"NewImage"}' \
-H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "d16d9eb8786c408ba32780ee6b722a50"}
```

Look at an existing Image (DefaultImage will be created at installation):

```
curl -X GET https://head1.cluster.local/api/v1/image/DefaultImage -H \
"Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"name": "DefaultImage", "distro": "CentOS", "parent": null, \
"description": "Default image generated by scyld-add-boot-config", "exports": \
{"head23.cluster.local": {"targetcli": {"category": "iscsi", "location": \
"iqn.1998-04.com.penguincomputing:cb2eeba6fe3748febbb15c7cc1cfb165"}, \
"last_modified": 1721928167.775344, "last_modified_on": "head23.cluster.local"}}, \
"content": {"cwsquash": {"size": 1222410240, "mtime": 1721919588.6732733, \
"chksum": "sha1:7e24ac56d109394755302377c9226fa11aee45d", "filename": \
"486c1857bd214d3fad56dea1399b6440", "uid": "486c1857bd214d3fad56dea1399b6440"}, \
"last_modified": 1721919589.1442792, "last_modified_on": "head23.cluster.local", \
"last_modified_by": "admin1"}, "last_modified": 1721919581.66072, \
"last_modified_on": "head23.cluster.local", "last_modified_by": "admin1", \
"uid": "cb2eeba6fe3748febbb15c7cc1cfb165"}}
```

Look into the details statistics for the image:

```
curl -X GET https://head1.cluster.local/api/v1/image/DefaultImage/content/info \
-H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"size": 1222410240, "mtime": 1721919588.6732733, "chksum": \
"sha1:7e24ac56d109394755302377c9226fa111aae45d", "refcnt": 1, "fmt": "cwsquash"}}
```

6.8 Dynamic Group Objects

Where Attribute Groups require intentional effort to join or leave, Dynamic Groups (DynGroups) are a way to group nodes based on shared or common attributes, status, or hardware information. DynGroups also do not alter a node's attributes, so they are most useful in targeting an action at a group of similar nodes.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field.

6.8.1 Data Fields

The DynGroups fields are:

name	Required: The name for the Dynamic Group
selector	Required: A string representing a node-selector function (see below)
description	Optional: A text string with descriptive information

A node-selector function is a logical function that is computed based on a given node's information. Since a node's boot configuration is set in the `_boot_config` attribute, the following will select all nodes that using `DefaultBoot`:

```
a[_boot_config]=="DefaultBoot"
```

the `a[...]` means to look up a nodes attribute, similarly `h[...]` can be used for hardware and `s[...]` for status. The amount of total memory on a node is given in the hardware information, under the `ram_total` key:

```
h[ram_total] > 1000000
```

will give the set of nodes with more than 1000000 bytes of memory.

See *Attribute Groups and Dynamic Groups* for more information on the node-selector language.

6.8.2 Additional Endpoints

To find the set of nodes that are currently in a given DynGroup, use the following endpoint:

```
GET /dyngroup/<UID>/nodes
Returns a JSON object with a list of node UIDs; if no nodes are currently in the
dynamic group, an empty list will be returned
```

6.8.3 Example

Create a dynamic group:

```
curl -X POST https://head1.cluster.local/api/v1/dyngroups --data \
  '{"name":"BigMemory","selector":"h[ram_total] > 100GB"}'
-H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "b30886d9c8ca40d99d66b668074811b1"}
```

Verify the group's information:

```
curl -X GET https://head1.cluster.local/api/v1/dyngroup/BigMemory -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"name": "BigMemory", "selector": "h[ram_total] > 100GB", \
  "parsed": "(hardware[\"ram_total\"] > 100GB)", "last_modified": 1721930148.843629, \
  "last_modified_on": "head23.cluster.local", "last_modified_by": "admin1", \
  "uid": "b30886d9c8ca40d99d66b668074811b1"}}
```

Get the list of nodes currently in this DynGroup:

```
curl -X GET https://head1.cluster.local/api/v1/dyngroup/BigMemory/nodes -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": ["199eadbea2084b80a4efcf445538efc5", \
  "d3b59618750e4957bc3139a0a29cb1a8", "4343fc9a431d49e89639b8cf3644db9a", \
  "9e831349cbcf484f8db075e12ef4dd9d", "35d6faece6a7411cb26a7c7b02fda708"]}
```

Delete the DynGroup:

```
curl -X DELETE https://head1.cluster.local/api/v1/dyngroup/BigMemory -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

6.9 Naming Pool Objects

Naming Pools provide a mechanism to tie groups of nodes together through a common naming scheme, like n001 through n100. There is a default naming scheme that nodes will be assigned into; alternatively, when creating a node, it can be directly inserted into an existing Naming Pool.

In addition to naming the nodes according to a pattern, the Naming Pool can also provide offsets to IP addresses or include nodes into specified network segments.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field.

6.9.1 Data Fields

The Naming Pool fields are:

```
name
  Required: The name for the Dynamic Group

pattern
  Required: A string representing a naming pattern; a pattern may have one or more
  letters followed by a set of curly-braces followed by another one or
  more letters (see below)

description
```

(continues on next page)

(continued from previous page)

Optional: A text string with descriptive information

first_index

Optional: Sets the first index for nodes added to this pool; default is 0

ip_base

Optional: Sets a base IP address for the first node in the pool; default is empty (use the next available address in the default network)

ip_offset

Optional: Sets an offset for the IP address of the first node in the pool; default is empty (use the next available address in the default network)

network

Optional: Sets the Network to use when setting IP addresses

group

Optional: Sets an Attribute Group object that all nodes in this pool will inherit from; any node added to the pool will automatically be assigned the attributes of this group (unless other Attribute Groups are assigned)

parent

Optional: Sets a parent Naming Pool for nested pools

There are several options for how naming pool patterns may be constructed:

pattern="n{"

This is the default, and indicates that nodes will be named "n0" and up; note that in such a pattern, the name length is not constant which may lead to output formatting errors (i.e. "n0" versus "n100" will lead to a 2 character offset)

pattern="gpu{:03d}"

Will give names like gpu000, gpu001, gpu002, etc.; the ":03d" is formatted, zero-padded, 3 digits

6.9.2 Additional Endpoints

Several additional endpoints are available:

GET /namingpools/info/<NODE>

For a given node, NODE, the system will reply with the naming pool information used when naming that node

PUT /namingpool/<UID>/push

Used to push node names to be updated; this is an advanced feature, please contact Penguin Solutions for more information.

6.9.3 Example

Create a naming pool:

```
curl -X POST https://head1.cluster.local/api/v1/namingpools --data \  
'{"name":"gpu","pattern":"gpu{}}" -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..." \  
{"success": true, "data": "1049f3b2f2d744f088a4ee6cc3ecfc91"}
```

Check on a node's naming pool:

```
curl -X GET https://head1.cluster.local/api/v1/namingpools/info/gpu0 \  
-H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..." \  
{"success": true, "data": {"index": 0, "pattern": "gpu{}}}
```

6.10 Software Repository Objects

Software Repository objects (repos) are used to collect information about an upstream or local repository for software packages.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field.

6.10.1 Data Fields

Admin objects have several fields:

```
name  
    Required: The name of the user on the underlying system  
  
description  
    Optional: A text string with descriptive information  
  
full_name  
    Optional: A text string for identifying the repos  
  
urls  
    Optional: A list of URLs for the repos  
  
mirrors  
    Optional: A list of mirror URLs for the repos  
  
disable  
    Optional: A list of True/False flags for the URLs  
  
keys  
    Optional: A list of keys for the URLs  
  
check  
    Optional  
  
rhel_entitlement  
    Optional: RedHat license entitlement key
```

6.10.2 Additional Endpoints

For locally hosted repos, several endpoints can be used to investigate the ISO:

```
GET /repo/<UID>/iso
    Downloads the ISO image for this repo. Note that the output format is different
    from other ClusterWare commands - there are no "success" or "data" keys, just the
    binary data of the ISO file being requested

PUT /repo/<UID>/iso
    Uploads an ISO for this repo

DELETE /repo/<UID>/iso
    Deletes the ISO for this repo; note that the repo's object itself will remain,
    just the ISO will be deleted

GET /repo/<UID>/iso/stat
    Returns basic statistics about the ISO file
```

6.11 Software Distribution Objects

Software Distribution objects (distros) are used to collect information about a software distribution. For example, which repos hold packages and what packaging scheme is used.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field.

6.11.1 Data Fields

Admin objects have several fields:

```
name
    Required: the name of the user on the underlying system

description
    Optional: A text string with descriptive information

repos
    Optional: A list of one or more Software Repository objects

packaging
    Optional: A string representing the packaging scheme: rpm, deb, apk; if it cannot
    be inferred from other data, "rpm" will be assumed

release
    Optional: A string representing the name of the release
```

6.11.2 Additional Endpoints

There are no additional endpoints for distros; the basic operations all function as expected. For example, GET /distro/<UID> will return information about the distro.

6.12 State Set Objects

A State Set is a set of criteria that are used to identify groups of nodes in a given state. The states sets primitive is accessed through the `/nodes/waitfor` URL, but is otherwise analogous to the other endpoints. Several endpoints provide for the creation and modification of state sets.

6.12.1 Data Fields

The state set fields are:

```
name
  Required: The name for the state set

api_state_set
  Required: A dictionary of state-names and their corresponding node-selector
```

For example, `POST /nodes/waitfor` allows creation of a new state set. To completely replace the list of states, use the `PATCH /nodes/waitfor` action with the “states” key.

6.12.2 Additional Endpoints

Several additional endpoints are available:

```
GET /nodes/waitfor
  Returns a list of state sets (UIDs) that are currently defined

POST /nodes/waitfor
  Create a new state set with the posted parameters

GET /nodes/waitfor/<NAME>
  Returns detailed information for a given state set (by Name or UID)

PUT /nodes/waitfor/<NAME>
  Update the information for a given state set (by Name or UID)

DELETE /nodes/waitfor/<NAME>
  Deletes a given state set

GET /nodes/waitfor/<NAME>/nodes
  Returns a list of nodes currently in the given state set
```

6.12.3 Example

Create a new state set:

```
curl -X POST https://head1.cluster.local/api/v1/nodes/waitfor \
  --data '{"name":"my_states", "states": { "red_state": "index > 10" } }' \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "my_states"}
```

Get the detailed information on a state set:

```
curl -X GET https://head1.cluster.local/api/v1/nodes/waitfor/my_states \
-H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"red_state": "index>10"}}
```

Delete the state set:

```
curl -X DELETE https://head1.cluster.local/api/v1/nodes/waitfor/my_states \
-H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true}
```

6.13 Network Objects

Network objects are used to collect information about network subnets that can be used for deploying compute nodes.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field. For Networks the name field is optional, so using the UID may be the most reliable approach.

6.13.1 Data Fields

Network objects have several fields:

```
first_ip
    Required: The starting IP address for this network

ip_count
    Required: (Integer) number of IP addresses that may be assigned on this subnet

mask_bits
    Required: (Integer) number of bits for the network mask

name
    Optional: The name of the user on the underlying system

description
    Optional: A text string with descriptive information

first_index
    Optional: (Integer) number for the first index of the nodes in this network

router_ip
    Optional: The IP address for the router on this network

gateway_ip
    Optional: The IP address for the gateway on this network

domain
    Optional: The domain name to be set for this network

node_iface
    Optional: Network interface used during booting
```


6.13.2 Additional Endpoints

There are several additional endpoints for networks:

```
GET /nets/byiface/<PART>
```

This endpoint is deprecated and will be removed in a future release; returns a JSON object with a list of networks that reference the interface name, NAME; full or partial matches are returned

```
GET /nets/byname/<NAME>
```

Returns a JSON object with a list of networks that reference the network name, NAME; full or partial matches are returned

```
GET /nets/byip/<IPADDR>
```

Returns a JSON object with a list of networks that reference the partial or full IP address, IPADDR

```
GET /nets/bykey/<PART>
```

Returns a JSON object with a list of networks that reference the partial or full UID, PART

6.14 Git Repository Objects

Git Repository objects (gitrepos) are used to create and manage locally hosted Git repositories, which may be useful for automated configuration systems.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field.

6.14.1 Data Fields

Admin objects have several fields:

name

Required: The name of the user on the underlying system

description

Optional: A text string with descriptive information

public

Optional

url

Optional: A URL for the git repos that will be used for synchronization; note that either "url" or "upstream" can be defined, but not both

upstream

Optional: A URL for an upstream git repos that will be used for synchronization; note that either "url" or "upstream" can be defined, but not both

branch_map

Optional: A list of local-to-remote branch mappings

sync_period

(continues on next page)

(continued from previous page)

Optional: Time between synchronization events for this gitrepos; can be a number (seconds) or a string of the form “10m”, “2h”, “1d” for minutes, hours, or days, respectively

6.14.2 Additional Endpoints

For gitrepos, several endpoints are available:

```
GET /gitrepo/{uid}/url
  Returns a URL string that can be used to clone the gitrepo. The output
  format is different from other ClusterWare commands - there are no “success” or
  “data” keys, just the URL string. This endpoint is public and does not require an
  authentication token

GET /gitrepo/{uid}/repo
  Returns the top-level file in the repo. The output format is different
  from other ClusterWare commands - there are no “success” or “data” keys, just the
  (text) file. This endpoint is public and does not require an authentication token

GET /gitrepo/{uid}/repo/<PATH>
  Returns the file in the repo at location PATH. The output format is
  different from other ClusterWare commands - there are no “success” or “data” keys,
  just the (text) file. This endpoint is public and does not require an
  authentication token

GET /gitrepo/{uid}/content
  Returns the top-level file in the repo. The output format is different
  from other ClusterWare commands - there are no “success” or “data” keys, just the
  (text) file. This endpoint is public and does not require an authentication token

GET /gitrepo/{uid}/content/<PATH>
  Returns the file in the repo at location PATH. The output format is
  different from other ClusterWare commands - there are no “success” or “data” keys,
  just the (text) file. This endpoint is public and does not require an
  authentication token

GET /gitrepo/{uid}/sync
  While this is a GET operation, it accepts a JSON object payload with an “action”
  key (required) and “branchmap” key (optional); action should be one of: 'status',
  'checkout', 'reset', 'pull', 'rebase'

POST /gitrepo/{uid}/sync
  Accepts a JSON object with an “action” key (required) and a “branchmap” key
  (optional); action should be one of: 'status', 'checkout', 'reset', 'pull',
  'rebase'

GET /gitrepo/{uid}/branches
  While this is a GET operation, it accepts a JSON object with optional keys:
  branchmap, all, filter; the branchmap is a list of branches to get information
  about, the all parameter is True/False, and filter allows for more arbitrary
  filtering of the gitrepos list
```

(continues on next page)

(continued from previous page)

DELETE /gitrepo/{uid}/branches

Accepts a JSON object with one or more optional keys: “branches” (a list of branches to delete) or “all” (True/False, delete all branches in the gitrepos)

6.15 Hostname Objects

The ICE ClusterWare™ platform runs an DNS service to handle the name records needed for compute nodes and that service can be extended to provide other hostname records as well. If a hostname record includes a MAC address, then the ClusterWare system will include it in its DHCP service.

When issuing requests, the UID field in the URL can be either the actual UID of the object or the name of the object as given in the "name" field.

6.15.1 Data Fields

name

Required: the name for the host

type

Required: a string representing the type of DNS record this will be: arec (IPv4), aaaarec (IPv6), or srvrec (service)

description

Optional: a text string with descriptive information

ip

Required for arec and aaaarec: the IP address associated with this name

mac

Optional for arec and aaaarec: the MAC address associated with this name; if no mac is given, then DHCP will not be configured for this host

target

Required for srvrec: the target server that is running this service

port

Required for srvrec: the port that the service is running on

service

Optional for srvrec: the service name for this SRV record, defaults to the name of the record

proto

Optional for srvrec: the protocol that the service uses: tcp or udp

weight

Optional for srvrec: the “weight” given to this SRV record; if multiple SRV records exist for the same target, the higher weight will be preferred

priority

Optional for srvrec: the “priority” given to this SRV record; if multiple SRV

(continues on next page)

(continued from previous page)

records exist for the same target, the higher priority will be preferred

domain

Optional for srvrec: the domain portion of the FQDN; if not specified, it will default to whatever the ClusterWare head node is configured with

6.15.2 Additional Endpoints

An additional endpoint is provided to search the hostnames by UID:

```
GET /hostnames/bypart/<PART>
```

A partial UID can be sent in <PART> and the system will return a list of all hostname UIDs that match that partial UID

6.15.3 Example

Create a hostname record:

```
curl -X POST https://head1.cluster.local/api/v1/hostnames --data \
  '{"name":"nfserver","ip":"10.1.1.10"}' \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": "e921725181c741fba6f4ec8a027a432f"}
```

Verify the record's contents:

```
curl -X GET https://head1.cluster.local/api/v1/hostname/nfserver -H \
  "Authorization: Bearer eyJhbGciOiJIUzI1Ni..."
{"success": true, "data": {"type": "arec", "name": "nfserver", "ip": "10.1.1.10", \
  "last_modified": 1721931565.353962, "last_modified_on": "head23.cluster.local", \
  "last_modified_by": "admin1", "uid": "e921725181c741fba6f4ec8a027a432f"}}
```

6.16 Cluster-wide Endpoints

There are a number of endpoints dedicated to cluster-wide information, or ICE ClusterWare™ configuration information:

```
GET /cluster
```

Returns a JSON object with detailed information about the cluster, including the current time on the cluster, the monitoring and authentication configuration, image formats, plugins, etc.

```
PATCH /cluster
```

Accepts a JSON object with one or more key-value pairs; for each key, the corresponding entry in the cluster database is overwritten; keys may be:

- default_group (change the default group that nodes are assigned to)
- naming (change the default naming scheme)
- accept_new_nodes (whether the cluster will attempt to boot nodes that have not be previously defined)
- default_distro (change the default distro)
- influx_token (change the token used by InfluxDB and Grafana, parts of the ClusterWare monitoring and alerting system)

(continues on next page)

(continued from previous page)

- cluster_name (change the name of the cluster)
- cluster_id (change the Penguin Solutions cluster ID number)
- auth_config (change settings needed by the authentication system)

GET /cluster/database
Returns a JSON object with information about the underlying ClusterWare database. This endpoint is public and does not require an authentication token.

GET /cluster/summary
Returns a JSON object with a list of all object UIDs for all ClusterWare object types.

GET /cluster/usage
Returns a JSON object with cluster-wide usage information; this requires that a cluster accounting system be configured

GET /cluster/time
Returns a JSON object with the current time on the head node. This endpoint is public and does not require an authentication token.

GET /cluster/accountant
Returns a JSON object with the current cluster accounting configuration

PUT /cluster/accountant
Used to set the cluster accounting configuration; this is an advanced topic, please contact Penguin Solutions for more information.

GET /cluster/head/{uid}
Returns a JSON object with information about the specified head node.

GET /cluster/head/{uid}/log
Returns a JSON object with information about logs from the specified head node.

GET /cluster/hosts
Returns a JSON object with hostname-ipaddress pairs for all known nodes in the cluster. This endpoint is public and does not require an authentication token.

GET /cluster/sshkeys
Returns a JSON object with information about the authorized keys for the cluster. This endpoint is public and does not require an authentication token.

GET /cluster/cabundle
Returns a JSON object with information about the CA certificates for the cluster. This endpoint is public and does not require an authentication token.

GET /cluster/license
Returns a JSON object with information about the ClusterWare license for this cluster.

GET /whoami
Returns a JSON object identifying the current user

(continues on next page)

(continued from previous page)

GET /whereami

Returns a JSON object identifying where (what node) the current user is attaching from; this assumes that ClusterWare knows the server (that is, it's a compute node registered in the system)

6.17 Head Node Endpoints

The ICE ClusterWare™ platform stores head nodes objects in the database to assist with management and maintenance.

6.17.1 Data Fields

Head nodes have several fields:

uid

Required: A unique ID usually configured via a file on disk (base.ini)

ssh_public

Required: The SSH public key for this head node

description

Optional: A text string with descriptive information

ca_pem

Optional: A CA certificate

6.17.2 Additional Endpoints

Several endpoints can be used to interact with the head nodes:

GET /heads/down

Returns a JSON object with a list of all currently down head nodes

DELETE /heads/down

Deletes any currently down head nodes, removing them from the cluster; returns a JSON object with a list of head nodes that were removed

GET /head/services

Returns a list of all ClusterWare services and their current status (up or down). This endpoint does not require an authentication token, but requires that the connection come from another known head node

POST /head/services

accepts a JSON object with “status” and “services” keys (both required); status must be a comma-separated list of: 'start', 'stop', 'restart', 'enable', 'disable', 'reload'; and services must be a list of one or more head node services (e.g. from a call to GET /head/services). This endpoint does not require an authentication token, but requires that the connection come from another known head node

GET /head/peerurl

Returns a JSON object with the peer URL for attaching a new head to the cluster.

(continues on next page)

(continued from previous page)

This endpoint is public and does not require an authentication token

GET /head/files/unused

Returns a list of all ClusterWare-managed files (images, ISOs, etc.) that are currently unused. This endpoint does not require an authentication token, but requires that the connection come from another known head node

DELETE /head/files/unused

Deletes any currently unused files, removing them from the cluster; returns a JSON object with a list of files that were removed. This endpoint does not require an authentication token, but requires that the connection come from another known head node

GET /database/clean

Returns a JSON object with information about what a database “cleaning” would do (but does not perform the cleaning)

POST /database/clean

Performs database cleaning on the head node and returns a JSON object with information about what was performed

POST /database/leave

Requests that this head node leave the cluster. This endpoint does not require an authentication token, but requires that the connection come from another known head node

Caution

This action may cause significant disruption to a running cluster; contact Penguin Solutions for additional information or assistance.

6.18 Boot-time Support Endpoints

Several ICE ClusterWare™ endpoints are designed to support low-level boot-time processes like kickstart, ztp, and ignition:

GET /kickstart/<NAME>

Returns the kickstart file with the given name; the file must already exist and be located in the /opt/scyld/clusterware/kickstarts directory. The output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the (text) file. This endpoint is public and does not require an authentication token

GET /ztp/script

Returns a ZTP script or config file for a given node; the node is determined by the incoming connection’s IP address and the node must have its “_boot_config” attribute set to “ztp:<ZTP_SCRIPTNAME>”. The script must already exist and be located in the /opt/scyld/clusterware/kickstarts directory. The output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the (text) file. This endpoint is public and does not require

(continues on next page)

(continued from previous page)

an authentication token

GET /ignition/bin

Returns the ignition binary for use by the booting node. The output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the (text) file. This endpoint is public and does not require an authentication token

GET /ignition/<UID>

Returns the ignition configuration file for the node identified by UID; the node must have its “_ignition” attribute set to the configuration to use. That configuration may be a URL, a git repo URL, or a locally hosted config file. The file must already exist and be located in the /opt/scyld/clusterware/kickstarts directory. The output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the (text) file. This endpoint is public and does not require an authentication token

GET /install/head/scyld-install

Returns a `scyld-install` script that can be used to join a server to the current head node. This may be useful when automating the creation of multi-head clusters. The output format is different from other ClusterWare commands - there are no “success” or “data” keys, just the (text) file. This endpoint is public and does not require an authentication token

POST /install/password

Accepts a JSON object with an “admin_pass” key and value and verifies that the password is correct for access to the ClusterWare database. This may be useful when automating the creation of multi-head clusters. This endpoint is public and does not require an authentication token

GET /install/repo

Returns a JSON object with a data field containing a repository configuration suitable for another head node. This may be useful when automating the creation of multi-head clusters. This endpoint is public and does not require an authentication token

GET /install/client/installer

Returns a shell script that can be used to install a system with the ClusterWare node package and related monitoring packages. If requested with no parameters, a “pre-installation” script will be returned that will attempt to find out the CPU architecture, OS name, and software packaging system. Alternatively, send the following HTTP parameters and get a more customized installer script: ::

```
cpu=x86_64
  specify the CPU architecture; one of x86_64 or aarm64
os=rhel8
  specify the OS; one of rhel, debian, cumulus, or sonic
pkg=tar
  specify the software packaging system; one of tar, rpm, or deb
```

If one or more parameters are omitted, then the system will select the installer that best matches the set of parameters that were sent. Clients must be aware

(continues on next page)

(continued from previous page)

that a different packaging system may be returned; for example, requesting the rpm installer may return a tar-based installer instead.

```
GET /install/client/download/<PKGSYS>/<NAME>
```

Downloads the package named NAME from the head node, where the packaging system is determined by PKGSYS. PKGSYS is one of tar, rpm, or deb. Note that this simply downloads the package and does not install it.

6.18.1 Client Download Endpoints

ClusterWare head nodes host a set of user-provided software packages that compute nodes can download and install. Several packages are pre-populated in the system, and you can copy your own packages to `/opt/scyld/clusterware/clientpkgs` directories to allow clients to download them as well. For a given package, the following endpoints allow the client to define the desired architecture and package type:

```
/install/client/download/{pkgname}
/install/client/download/{pkgtype}/{pkgname}
/install/client/download/{pkgtype}/{pkgarch}/{pkgname}
```

Where:

- `pkgname` is a string containing the name of the package to download
- `pkgtype` is one of (rpm, deb, tar, tgz, tar.gz). tar, tgz, and tar.gz are equivalent. Default=rpm
- `pkgarch` is one of (x86_64, amd64, aarch64, arm64). X86_64 and amd64 are equivalent; aarch64 and arm64 are equivalent. x86_64/aarch64 are the RHEL/rpm names and amd64/arm64 are the Debian names. Target architecture must be defined or the default will be used. Default = x86_64

Note

`pkgtype` and `pkgarch` can also be defined in HTTP parameters at the end of the URL. For example, `http://head1.cluster.local/api/v1/download/mypackage?pkg=rpm&arch=x86_64`.

If you request a tar file, the system first looks for a tar file in the `/opt/scyld/clusterware/clientpkgs/tar` directory. If nothing is found there, the system looks in `/opt/scyld/clusterware/clientpkgs/rpm` and, if the package is found, the rpm is converted to tar and stored in `/opt/scyld/clusterware/clientpkgs/tar` for future access. This same conversion rule is in place for all three endpoints.

Important

In a multi-head cluster, the main directory is not automatically replicated to all head nodes. You must manually add packages to all head nodes.

RELEASE NOTES, CHANGELOG, AND KNOWN ISSUES

ICE ClusterWare™ Release v12.4.2 is the latest update to the ClusterWare platform.

Release Notes provides high-level summary information about the latest ClusterWare release.

Changelog includes a history of the most recent ClusterWare releases.

Known Issues And Workarounds has a summary of notable known current issues.

7.1 Release Notes

ICE ClusterWare™ Release v12.4.2 is the latest update to the ClusterWare platform.

For the most up-to-date product documentation, visit <https://docs.ice.penguinsolutions.com/>. The most recent version will accurately reflect the current state of the ClusterWare yum repository of RPMs that you are about to install. For additional helpful information about ClusterWare, visit the Penguin Computing Support Portal at <https://www.penguinsolutions.com/computing/support/technical-support/>.

7.1.1 12.4.2 Release Notes

- Version 12.4.2 reverts a bare metal boot regression and a SELinux telegraf upgrade issue.

7.1.2 12.4.1 Release Notes

- Version 12.4.1 includes improvements to the ClusterWare graphical user interface (GUI), including a new GPU dashboard available from Grafana.
- Various bug fixes and performance improvements.

7.1.3 12.4.0 Release Notes

- Version 12.4.0 introduces the concept of "providers", i.e. external sources for hardware resources. The initial provider uses *virsh* to allow ClusterWare administrators to connect to an existing hypervisor running *libvirt* and allocate virtual machines in a streamlined way using the new *scylld-clusterctl providers* command. A cluster administrator can use these virtual machines as ephemeral test instances while developing images or deploy a persistent image to the virtual disks to make a login node.

Although this first *virsh* provider can only be bound to individual hypervisors, another provider currently under development will be able to allocate machines on KubeVirt compatible Kubernetes deployments. Other providers could also be implement an interface between ClusterWare and other virtualization platforms or provisioning systems.

- This version also includes a reworked graphical user interface (GUI) and many bug fixes and performance improvements.

- Starting with version 12.4.0, the product has been rebranded from Scyld ClusterWare to ICE ClusterWare in the GUI and documentation.

See *Changelog* for a full history of ClusterWare releases, and *Known Issues And Workarounds* for a summary of notable known current issues.

7.2 Changelog

See *Release Notes* for summary information about the latest ICE ClusterWare™ release. This section contains a more detailed ChangeLog history of all recent releases.

7.2.1 12.4.2-g0000 - March 17, 2025

- Revert a bare metal boot issue due to a conflict with upcoming Sonic ZTP functionality.
- Fix an SELinux issue that interferes with Telegraf on upgrade.

7.2.2 12.4.1-g0000 - March 7, 2025

- Continuing improvements to the ClusterWare graphical user interface (GUI), including more consistent display of labeled fields and components.
- Updates to dashboards, including a new GPU dashboard.
- SELinux/MLS improvements.
- Rebuild the initramfs when deploying grub to a local drive for a persistent installation.
- Implement a new `_wipe_all` attribute that will destroy all local data on a booting system.
- Move to Git version 2.48.1 to patch known vulnerabilities.
- Bug fixes and documentation updates.

7.2.3 12.4.0-g0000 - February 3, 2025

- The product is rebranded from Scyld ClusterWare to ICE ClusterWare. Initial changes are reflected in the product GUI and in the documentation. Future releases will introduce additional branding updates, including updates to the command line tools.
- Implement the first *providers* plugin, specifically supporting hypervisors running *libvirt* using *virsh* and *virt-install* commands.
- Include a couple of example deploy scripts in the `/opt/scyld/clusterware-tools/examples/deploy` directory.
- Reduce repetitive logging.
- Implement a new `_altmacs` reserved attribute that passes alternative MAC addresses for a node to the DHCP server. This attribute may be replaced by a more robust solution in future releases.
- Significant simplifications and improvements to the `scyld-kube` tool used for deploying Kubernetes.
- Mark nodes as "busy" if *virsh list* shows running virtual machines on the node.
- A new `scyld-modimg --deploy` argument allows administrators to execute an *Ansible* playbook against an image or combine the copy and execute steps for running a shell script inside the image.
- The `scyld-modimg` command now accepts a `--progress` argument to either not print remaining time or to print dots instead of detailed progress.
- Propagate errors from the *debootstrap* tool out to the user to simplify Ubuntu image creation debugging.
- Prevent users with the NoAccess role from even logging in and prevent *tmpadmins* from minting tokens.

- Improve parsing of `scyld-modimg --run` scripts and document the functionality.
- Add `--discard-on-error` option to `scyld-modimg` to facilitate scripting and automation.
- The `scyld-clusterctl nets` tool allows admins to define additional networks where ClusterWare nodes may be connected.
- Improved ClusterWare graphical user interface (GUI) information architecture to help new users navigate the product.
- Each primitive now presents a set of labeled fields and components within the ClusterWare GUI that are customized to that primitive.
- Updated ClusterWare GUI colors and logos to match the new product branding.
- Make `ipmitool` and `rasdaemon` weak dependencies of `clusterware-node`.
- Implement a new `_aim_status` reserved attribute and add support in `scyld-nodectl status` to show status based on that attribute. Contact Penguin Computing to learn more.
- Rearrange the build system to better isolate Pyramid code.
- Move image exports from the image to the head that does the export.
- Replace `libvirt` power plugin with a version that calls `virsh`.
- Remove the deprecated socket-based waitfor code.
- Add stricter versioned dependencies between some packages.
- Ensure `scyld-clusterctl hosts` entries are pushed to `scyld-nss`.
- Remove more references to e17 and remove development packages required by e17 builds.
- Keep the `dnsmasq` service up during clusterware service restarts.
- Allow the `mosquitto` service to start even with missing certificates.
- Add image locking during modification to prevent administrators from accidentally overwriting each other's changes.
- Improve `scyld-modimg` to make conflicts between different instances less likely.
- Implement shared-key encryption for communications between head nodes using `stunnel`.
- Improve our parsing of `ip` output.
- Add documentation about communication encryption.
- Switch `telegraf` from UDP to HTTP(S) with a new relay service, significantly reducing telemetry gaps.
- Improved method for deploying client packages to switches.
- Document how to change the `etcd` password and create a script to recover if the `etcd` passwords is lost. Contact Penguin Computing for assistance with the script.
- Improve the `slurm` and `kubernetes` installation scripts.
- Include the API Reference as a part of our standard documentation.
- Add a missing dependency required to build newer Ubuntu images.
- Update the supported distros table to include e18.10 and e19.5.
- Update documentation information architecture and HTML site design to improve user experience.
- Assorted other bug fixes and documentation updates.

7.2.4 12.3.0-g0000 - October 4, 2024

- Reduce polling in `scylid-nodectl status --refresh`, but leverage the waitfor framework and MQTT.
- Switch to a Unix socket to communicate between the ClusterWare backend and etcd to enable updating gRPC.
- Add a new `_bootnet` attribute for customizing the name of the `bootnet` interface.
- Support `--selector` to select nodes in `slurm-scylid.setup`.
- Introduce an improved `clusterware-node` deployment mechanism for SONiC switches.
- Make compute node code scripting less likely to produce a bad parent-head-node line in `/etc/hosts`.
- Support creating tmpfs subdirectories in ignition for diskless STIG'd systems.
- Cleaner handling of the `client.sslverify` setting.
- Reduce the head node minimum memory check after removal of Couchbase.
- Restrict access to the GUI to only accept secure remote connections.
- Bump the version numbers for most Python dependencies.
- Correct "frozen" image handling during import and refuse to delete frozen images.
- Remove deprecated code, including code specific to e17 head nodes.
- Add functionality for Telegraf to collect ClusterWare node attributes.
- Change technique for converting node lists into ranges when reporting status.
- Tighten some directory permissions.
- Correct the `_ipxe_sanboot` creation during bootload installation.
- Fix a `scylid-bootctl export` failure that previously required a patch.
- Provide a mechanism for setting a realtime IO priority on etcd.
- Make it more difficult to modify a cached version of an image unintentionally.
- Improve gitrepo backend handling to avoid common failures.
- Stop creating `.old.XX` files when modifying objects in multi-head clusters.
- Avoid the MOTD interfering with `scylid-nodectl scp`.
- Small fixes to boot chaining failure handling.
- Wider use of the cluster certificate authority to securing communications.
- Fixes for netplan configurations in Ubuntu images.
- Restart Telegraf when moving between head nodes.
- KeyCloak integration improvements.
- Assorted other bug fixes and documentation updates.

7.2.5 12.2.0-g0000 - July 26, 2024

- Improve Grafana column scaling.
- Quiet a warning about TripleDES by removing it as an option from paramiko.
- Support `_boot_style=iscsi` on e18 and e19 systems.
- Update CentOS 7 and CentOS Stream 8 URLs to use `vault.centos.org` since e17 is now also EOL.

- Improve DNS resolution of head nodes with multiple IPs using *localise-queries* in the *dnsmasq.conf.template* but also include a *leases.register_heads* boolean to disable entire feature.
- Write NetworkManager connection files on el9 systems and improve netplan configuration file writing on Ubuntu.
- Initial Redfish support including an aggregation daemon with more changes and documentation coming later.
- Provide a mechanism to create a bootable ISO from one or more boot configs.
- Improve handling of slurm uid and gid syncing when installing packages.
- Add arguments to `scyld-nodect1 kexec` to allow for one-time-booting using a specific image or boot configuration.
- Improve the `scyld-modimg --capture` error handling.
- Downgrade `ansible-core` to 2.15.10 to match Python 3.9.
- Small improvements and cleanups across the GUI.
- Introduce a new RBAC system for administrators, current scoped cluster-wide. All existing admins will now have the FullAdmin role.
- Support substitution within the *power_uri* field.
- Initial support for deploying Harvester nodes from an ISO.
- Unhide the existing `scyld-clusterctl nets` functionality.
- Include the mosquito MQTT server to publish system events.
- Confirm keys added through `scyld-adminctl` can be loaded with paramiko.
- Improved Ubuntu image handling in `scyld-modimg`.
- Expose the limited but existing `scyld-nodect1 scp` functionality.
- Improve ZTP handling but still only supporting Cumulus.
- Improve the unknown nodes tab for unrecognized dhcp clients.
- Include a mechanism to mask attribute values in normal output. Default to masking `_remote_pass`, `_tpm_owner_pass`, and `_bmc_pass`.
- Make more of an effort to mask the SOL password in output.
- Prevent the creation of unrecognized reserved attributes and update reserved attributes documentation.
- Include a `sched_watcher` agent for collecting node status from slurm.
- Rework compute node client certificate handling.
- Clean up dhcp6 error messages.
- Fix kernel version sorting in `sclyd-mkramfs`.
- Update numerous python and npm dependencies.
- Assorted other bug fixes and documentation updates.

7.2.6 12.1.1-g0000 - January 23, 2024

- Assorted fixes for `initramfs` ignition use when booting el9 nodes.
- Rework how `scyld-nodect1 ssh` gets node keys allowing for ssh into el9 nodes with FIPS enabled.
- Print names in place of some UIDs returned by `scyld-*ctl` tools.
- Note and handle that `ram_total` / `ram_free` are stored in KiB.

- Check all uses of `urlparse().netloc` and replace several with `urlparse().hostname`.
- Assorted test script and other bug fixes.

7.2.7 12.1.0-g0000 - December 28, 2023

- Head node hosted gitrepos can mirror upstream repositories.
- Several bug fixes around the `scyld-nodectl waitfor` functionality.
- Hide the exports section in `scyld-imgctl` output unless `-L` is used.
- Fix a long standing bug during file upload where "Finishing up..." still be displayed after upload was complete.
- Fix a long standing bug during file upload that caused an additional file checksum computation.
- Deprecate the `nodes.boot_timeout` global in favor of a per-node `_boot_timeout` attribute.
- Fix head node eject / leave functionality to make it less likely a removed head node will automatically rejoin or try to provide services to compute nodes.
- Fix `PREFER_KMOD` handling in `/opt/scyld/clusterware-tools/conf/mkramfs.conf`
- Technology preview of a scheduler-watcher that can be used to feed scheduler status into the ClusterWare database. Attribute names and other details may change.
- Enable the slider to show and hide scheduler status within the GUI if any node has status information.
- Avoid address-in-use socket errors with multiple backend daemon threads.
- Fix typos that broke `sync-uids` and `take-snapshot` in ClusterWare 12.
- Make systems for node status, hardware, health, and monitoring use plugins for easier management.
- Authenticate with a user's SSH agent if they have already uploaded their public keys into the system.
- New support for partitioning during boot using ignition. See the documentation for the `_ignition` reserved attribute for details.
- Support for installing the GRUB 2 bootloader during boot. See the documentation for the `_bootloader` reserved attribute for details.
- Improved image capture capabilities with better error handling and using optional credentials and `sudo`.
- Implement a local signing authority for node client certificates stored in node TPMs.
- Support searching for a node by hostname even when it differs from the ClusterWare node name.
- Allow matching of naming pools in node selection using the same syntax that already matched dynamic groups.
- Add support for attaching an attribute group to a naming pool.
- Add `_domain` to specify the domain without using `_hostname`.
- Confirmed ClusterWare works on Rocky 9.3 and similar distros.
- Add a mechanism (`chroot.env_paths`) to define specific environment variables during image creation.
- Fix several bugs around node renaming that could have permitted multiple nodes with the same MAC or similar issues.
- Assorted GUI improvements, bug fixes, and performance improvements.

7.2.8 12.0.1-g0000 - July 24, 2023

- Reimplement and expose the `scylld-nodectl scp` functionality.
- Push `scylld-pack-node` to systems when running `scylld-modimg capture`. This also allows us to remove the `clusterware-common` package.
- Improve proxy handling during the installation process.
- Improve the handling of the `_hosts` attribute.
- Initial support for scripting `scylld-modimg` through `--run`.
- Provide a mechanism for changing the default hash from `sha1` to `sha256` or `sha512`.
- Deprecate `scylld-install --clear` in favor of `--clear-all`.
- Fix output labelling in `scylld-nodectl exec` results.
- Mark node status and the current head node in `managed --heads` output.
- Expand image capture to use `_remote_user / _remote_pass`.
- Improved Debian / Ubuntu image creation.
- Use the latest squashfs tools for packing and unpacking images.
- Assorted bug fixes and performance improvements.

7.2.9 12.0.0-g0000 - April 21, 2023

- The first release of ClusterWare version 12. Please see *Updating ClusterWare 11 to ClusterWare 12* for more details.
- Support RHEL / Rocky 9 as a head node and compute node platform.
- Upgrade to use Python 3.9 on all head node platforms.
- Entirely rewritten GUI with much more functionality.
- Switch to Telegraf, InfluxDB version 2, and Grafana instead of TICK. See *Grafana Telemetry Dashboard* for details about Grafana.
- Initial support for GRUB 2 as an alternative for iPXE.
- Configure `chrony` at install time for time sync within the cluster.
- Update `manageddb save` to default to saving ONLY the database.
- Fix selection language matching for `attributes[_boot_config]`.
- Include a newer (4.6) version of squashfs tools for more recent SELinux-related features.
- Allow command line clients to authenticate by signing messages with their SSH keys.
- Remove `banner.txt` support and use `SSH LogLevel` to control banner display when executing remote commands.
- Avoid a crash when two attributes only differ in capitalization.
- Fix "accept unknown nodes" behavior.
- Fix behavior of `scylld-nodectl exec --label`.
- Implement a new JWT-based authentication system with refresh tokens.
- New in-memory caching and indexing mechanism to improve document store lookup times.
- Provide a mechanism to record additional DNS mappings in the ClusterWare database.

- Default to installing config-less Slurm.
- Provide a tool to create a scyld-kube.iso for installation on clusters without internet access.
- Support booting nodes using UEFI in HTTP mode.
- Implement a restricted `status-updater` for "busy" nodes in C code, and provide attribute `_status_cpuset` to restrict `status-updater` service subprocesses to a specific set of CPU cores.
- Remove all references to Couchbase and some remaining NFS references.
- Enable `scyld-nss` by default on head nodes for name resolution.
- Use the dracut version native to the image instead of a custom ClusterWare version.
- Multi-head clusters now automatically rebalance nodes between heads.
- Many other bug fixes and optimizations.

7.3 Known Issues And Workarounds

The following are known issues of significance with the latest ICE ClusterWare™ version and suggested workarounds.

- Scyld OpenMPI versions 4.0 and 4.1 for RHEL/CentOS 8 require `ucx` version 1.9 or greater, which is available from CentOS 8 Stream and RHEL 8.4.
- When using a TORQUE or Slurm job scheduler (see *Job Schedulers*), if a node reboots whose image was **not** created using `/opt/scyld/clusterware-tools/bin/sched-helper`, then the cluster administrator must manually restart the job scheduler.

For example, if needed for a single node `n0`: `NODE=n0 torque-scyld-node` or `NODE=n0 slurm-scyld-node`.
Or to restart on all nodes: `torque-scyld.setup cluster-restart` or `slurm-scyld.setup cluster-restart`.

Ideally, compute node images are updated using `torque-scyld.setup update-image` or `slurm-scyld.setup update-image`, which installs the TORQUE/Slurm config file in the image and enables the appropriate service at node startup.

- If administrators are using `scyld-modimg` to concurrently modify two different images, then one administrator will see a message of the form:

```
WARNING: Local cache contains inconsistencies.  
Use --clean-local to delete temporary files, untracked files,  
and remove missing files from the local manifest.
```

then use `scyld-modimg --clean-local`.

However, only execute `--clean-local` after **all** `scyld-modimg` image manipulations have completed.

- Ensure that `/etc/sudoers` does not contain the line `Defaults requiretty`; otherwise, DHCP misbehaves.
- The `NetworkManger-config-server` package includes a `NetworkManager.conf` config file with an enabled "no-auto-default" setting. That is incompatible with ClusterWare compute node images and will cause nodes to lose network connectivity after their boot-time DHCP lease expires. Either disable that setting or remove the `NetworkManger-config-server` package from compute node images.
- The `scyld-clusterctl repos create` command has a `urls=` argument that specifies where the new repo's contents can be found. The most common use is `urls=http://<URL>`. The alternative `urls=file://<pathname>` does **not** currently work. Instead, you must first manually create an http-accessible repo from that `pathname`. See *Creating Local Repositories without Internet*.

- When moving a head node from one etcd-based cluster to another using the `managedb join` command, please reboot the joining head once the join is complete.
- If a new head node is failing to join an existing etcd-based cluster, check `/var/log/clusterware/etcd.log` and look for repeated lines of the form:

```
<DATE> <SERVER> etcd: added member <HEX> [<URL>:52380] to cluster <HEX>
```

If the log file contains multiple of these line per join attempt, then please try running `managedb recover` on an existing head node and joining all head nodes back into the cluster one-at-a-time. Re-joining heads that were previously in the cluster may require a `--purge` argument, i.e. `managedb join --purge <IP>`

- `scylld-install` performs its early check to determine if a newer *clusterware-installer* RPM is available by parsing the appropriate clusterware repo file (typically `/etc/yum.repos.d/clusterware.repo`) to find the first `base_url=` line. If there are multiple such lines, i.e., specifying multiple ClusterWare repos, then the cluster administrator should order the repos so that the repo containing the newest RPMs is the first repo in the file.
- A compute node using a version of *clusterware-node* older than 11.2.2 and booting from a head node that has upgraded to 11.7.0 or newer may not successfully send its status to the head node. Please upgrade the `clusterware-node` package inside the image to resolve this problem.
- Joining a ClusterWare 11 head node to a ClusterWare 12 head node will perform the join, but will not update the joining head node to ClusterWare 12. We recommend updating the ClusterWare 11 node to 12 prior to performing the join. See *Updating ClusterWare 11 to ClusterWare 12* for guidance about performing this update.
- Updating from 12.0.1 and earlier to 12.1.0 requires reconfiguration of the Influx/Telegraf monitoring stack. The following command can be used to update the necessary config files: `/opt/scylld/clusterware/bin/influx_grafana_setup --tele-env`, followed by `systemctl restart telegraf`. All data will persist through the upgrade.
- When using ignition to make partitions the partition specified with `_disk_root` will be formatted with the ext4 file system even if the ignition file specifies another file system such as XFS.

FREQUENTLY ASKED QUESTIONS (FAQ)

The following is a set of common questions and cross-reference pointers to the answers in the ICE ClusterWare™ documentation.

8.1 Software Install/Update

How do I install or update the product RPMs?

Always use `scyld-install` to install or update the basic ClusterWare packages. See *Install ICE ClusterWare* and *Updating Base Distribution Software*.

For optional ClusterWare packages that are not managed by `scyld-install`, see *Additional Software*.

Use a simple `yum install` or `yum update` to install or update non-ClusterWare base distribution packages.

How do I install or update software without head node Internet access?

See *Creating Local Repositories without Internet*.

8.2 Cluster Management

What if all ``scyld-*`` commands fail?

One reason may be the root filesystem is full. See *Head Node Filesystem Is 100% Full*.

Another reason may be the etcd database exceeds its size limit. See *etcd Database Exceeds Size Limit*.

What are hardware requirements for the latest release?

See *Required and Recommended Components*.

How do I add a compute node?

See *Node Creation with Known MAC address(es)* or *Node Creation with Unknown MAC address(es)*.

How do I replace a compute node?

See *Replacing Failed Nodes*.

How do I configure multiple head nodes?

See *Managing Multiple Head Nodes*.

How do I configure a job scheduler, like Slurm, TORQUE, or OpenPBS?

See *Job Schedulers*.

How do I install and configure OpenMPI?

See *OpenMPI, MPICH, and/or MVAPICH*.

How do I keep the host keys consistent across all compute nodes?

See *Compute Node Host Keys*.

How do I change a node name?

See *Node Names and Pools*.

How do I change IP addresses?

See *Changing IP Addresses*.

8.3 Manipulating Compute Node Images

How do I create an image containing a non-default kernel?

See *Modifying Images*.

How do I recreate the default image, boot config, and attributes?

See *Recreating the Default Image*.

How do I create an image containing a non-default base distribution?

See *Creating Arbitrary Rocky Images* or *Creating Arbitrary RHEL Images*.

How do I delete unused images or boot configurations to free storage space?

See *Deleting Unused Images*.

8.4 Issues with Interacting with Compute Nodes

What if all ``scylid-*`` commands fail?

One reason may be the etcd database exceeding its size limit. See *etcd Database Exceeds Size Limit*.

Why does ``scylid-nodectl -i <NODE_NAME> ssh`` fail?

Why does ``scylid-nodectl -i <NODE_NAME> shutdown`` or ``reboot`` fail?

LICENSE AGREEMENTS

9.1 End-User License Agreement

Penguin Computing Software End User License Agreement

Last revised: 1/30/2025

LEGAL NOTICE - READ CAREFULLY BEFORE INSTALLING OR OTHERWISE USING THIS SOFTWARE.

This License Agreement (the "Agreement") is a legal agreement between you, a single legal entity ("End User"), and Penguin Computing ("Penguin"). This Agreement governs your use of the ICE ClusterWare™ or ICE RemoteWare™ software defined below (the "Software") and any accompanying written materials (the "Documentation"). You must accept the terms of this Agreement before installing, downloading, accessing or otherwise using such Software and documentation.

By "ACCEPTING" at the end of this Agreement, you are indicating that you have read and understood, and assent to be bound by, the terms of this Agreement. If you are an individual working for a company, then you represent and warrant you have all necessary authority to bind your company to the terms and conditions of this Agreement.

If you do not agree to the terms of the Agreement, you are not granted any rights whatsoever in the Software or Documentation. If you are not willing to be bound by these terms and conditions, do not "ACCEPT" the EULA and remove the software from the system immediately.

END USER LICENSE AGREEMENT FOR SOFTWARE

1. Definitions

- "Clustered System" means a collection of computer systems managed by the Software and for which the total number of computers in the system is specified in the End User purchase order.
- "Master Node" means the computer or computers designated as the Master Node(s) in the applicable End User purchase order, where the Software is initially installed and from which the total number of computers comprising the Clustered System are managed.
- "Software" means the software provided under this Agreement by Penguin or its authorized distributor or reseller and for which the applicable End User purchase order specifies: (i) the Software to be licensed by End User; (ii) the Master Node(s); (iii) the license fees; and (iv) the total number of computers in the Clustered System for which End User has paid applicable license fees and the term of the Software usage.

The Software is comprised of a collection of software components that fall into three (3) categories: (a) "Unpublished Software" which is owned by Penguin and/or its licensors and licensed under the terms of this Agreement; (b) "Published Software" which is owned by Penguin and licensed under the GPL version 2 open source license or such other open source license as Penguin may elect in its sole discretion; and (c) "Open Source Software" which is owned by various entities other than Penguin and is subject to the "open source" or "free software" licenses, including but not limited to General Public Licenses (GPL), Lesser General Public License (LGPL), Apache, Artistic, BSD, IBM Public, Mozilla, Omron, Open Group Public License, and Python licenses.

- "Client Connections" means the simultaneous connections between any software client and Software, where a connection creates a persistent and unique Software session per software client.

2. License

- **License Grant.** Subject to the terms and conditions of this Agreement, Penguin grants to End User a non-exclusive, non-transferable, non-sub licensable right and license to (a) reproduce (solely to download and install), perform, and execute the Unpublished Software on the specified Master Node(s), solely for End User's internal purposes, and (i) solely for use on the number of computers in the Clustered System and (ii) not to exceed the maximum number of Client Connections for which End User has paid the required license fees for the authorized term; and (b) make one (1) copy of the Unpublished Software and Documentation for backup and/or archival purposes only.
 - **Restrictions.** The End User shall not, and shall not permit any third party to: (a) sell, lease, license, rent, loan, or otherwise transfer the Unpublished Software or Documentation, with or without consideration; (b) permit any third party to access or use the Unpublished Software or Documentation; (c) permit any third party to benefit from the use or functionality of the Unpublished Software via a timesharing, service bureau, or other arrangement; (d) transfer any of the rights granted to End User under this Agreement; (e) reverse engineer, decompile, or disassemble the Unpublished Software; (f) modify or create derivative works based upon the Unpublished Software or Documentation, in whole or in part; (g) reproduce the Unpublished Software or Documentation, except as expressly permitted in Section 2.1 above; (h) remove, alter, or obscure any proprietary notices or labels on the Unpublished Software or Documentation; (i) use the Unpublished Software for any purpose other than expressly permitted in Section 2.1 above; or (j) use the Unpublished Software for more than the total number of computers, or longer than the authorized term the End User is licensed for pursuant to Section 2.1 above.
 - **Open Source Software.** The Open Source Software and Published Software are not subject to the terms and conditions of Sections 2.1, 2.2, or 6. Instead, each item of Open Source Software and Published Software is licensed under the terms of the end-user license that accompanies such Open Source Software and Published Software, as may be located in the product packaging or available on-line. End User agrees to abide by the applicable license terms for any such Open Source Software and Published Software. Nothing in this Agreement limits End User's rights under, or grants End User rights that supersede, the terms and conditions of any applicable end user license for the Open Source Software or Published Software. In particular, nothing in this Agreement restricts End User's right to copy, modify, and distribute any of the Open Source Software and Published Software that is subject to the terms of the GPL and LGPL. For the Open Source Software and Published Software subject to the GPL and LGPL, for a period of three (3) years following End User's receipt of the Software, End User may contact Penguin at the address below in writing and request a copy of the source code for such Open Source Software or Published Software at Penguin's then-current fees.
3. **Ownership.** The Software is licensed, not sold. Penguin and its licensors retain exclusive ownership of all applicable worldwide copyrights, trade secrets, patents, and all other intellectual property rights throughout the world, and all applications and registrations relating thereto, in and to the Unpublished Software, Published Software, and Documentation, and any full or partial copies thereof, including any additions or modifications to the Unpublished Software and Documentation. End User acknowledges that, except for the limited license rights expressly provided in this Agreement or the Open Source Licenses, as applicable, no right, title, or interest to the intellectual property in the Software or Documentation is provided to End User, and that End User does not obtain any rights, express or implied, in the Software or Documentation. All rights in and to the Software not expressly granted to End User in this Agreement or the Open Source Licenses, as applicable, are expressly reserved to Penguin and its licensors. The "ICE ClusterWare™", "ICE RemoteWare™" and "Penguin Computing" trademarks and associated logos are the trademarks of Penguin and its affiliates. This Agreement does not permit End User to use the Penguin trademarks.
4. **Limited Warranty.** TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, THE SOFTWARE IS PROVIDED AND LICENSED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESSED OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, TITLE OR FITNESS FOR A PARTICULAR PURPOSE. PENGUIN DOES NOT WARRANT THAT

THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET THE END USER'S REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE ERROR FREE OR APPEAR PRECISELY AS DESCRIBED IN THE ACCOMPANYING DOCUMENTATION.

5. **Limitation of Liability.** TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, PENGUIN NOR ANY OF ITS AUTHORIZED DISTRIBUTORS, RESELLERS AND LICENSORS WILL BE LIABLE TO END USER FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, LOST PROFITS, LOST OPPORTUNITIES, LOST SAVINGS, OR LOST DATA OR COST OF COVER ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE OR DOCUMENTATION OR ANY SERVICES HEREUNDER, HOWEVER CAUSED ON ANY THEORY OF LIABILITY (INCLUDING CONTRACT, STRICT LIABILITY, OR NEGLIGENCE), EVEN IF PENGUIN, ITS AUTHORIZED DISTRIBUTORS, RESELLERS OR LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL PENGUIN'S AGGREGATE LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT ACTUALLY PAID BY END USER TO PENGUIN FOR THE SOFTWARE GIVING RISE TO THE CLAIM.

END USER ACKNOWLEDGES THAT THE AGREEMENT REFLECTS AN ADEQUATE AND ACCEPTABLE ALLOCATION OF RISK.

6. **Confidential Information.** Unpublished Software and the structure, organization, and code of the Unpublished Software, including but not limited to the shell scripts of the Unpublished Software, are confidential and proprietary information ("Confidential Information") of Penguin and/or its licensors. End User agrees to safeguard such Confidential Information with a degree of care commensurate with reasonable standards of industrial security for the protection of trade secrets and proprietary information such that no unauthorized use is made of such information and no disclosure of any part of its contents is made to anyone other than End User's employees whose duties reasonably require such disclosure in order to effectuate the purposes of this Agreement.
7. **Term and Termination.** This Agreement will remain in effect until terminated or for the authorized term of license usage. End User may terminate this Agreement by removing the Unpublished Software from End User's computers, ceasing all use thereof, and destroying all copies of the Unpublished Software and Documentation and certifying to Penguin that it has done so. Any breach of this Agreement by End User will result in the immediate and automatic termination of this Agreement and licenses granted by Penguin herein, and End User shall cease all use of and destroy all copies of the Unpublished Software and Documentation and certify to Penguin that it has done so. In addition to termination, Penguin will have the right to pursue any other remedies available to it under law or in equity.
8. **Export Controls.** End User acknowledges and agrees that the Software and Documentation which is the subject of this Agreement may be controlled for export purposes. End User agrees to comply with all United States export laws and regulations including, but not limited to, the United States Export Administration Regulations, International Traffic in Arms Regulations, directives and regulations of the Office of Foreign Asset Control, treaties, Executive Orders, laws, statutes, amendments, and supplement thereto. End User assumes sole responsibility for any required export approval and/or licenses and all related costs and for the violation of any United States export law or regulation.
9. **U.S. Government End Users.** The Software is a "commercial item" as that term is defined at 48 C.F.R. 2.101 (OCT 1995), consisting of "commercial computer software" and "commercial computer software documentation" as such terms are used in 48 C.F.R. 12.212 (SEPT 1995). Consistent with 48 C.F.R. 212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire the Software with only those rights set forth herein.
10. **Miscellaneous.** This Agreement is the final, complete and exclusive agreement between the parties relating to the Software and Documentation, and supersedes all prior or contemporaneous proposals, representations, understandings, or agreements relating thereto, whether oral or written. Software shall be deemed irrevocably accepted by End User upon installation. No waiver or modification of the Agreement will be valid unless signed by each party. The waiver of a breach of any term hereof will in no way be construed as a waiver of any other term or breach hereof. The headings in this Agreement do not affect its interpretation. End User may not assign or transfer any of its rights or obligations under this Agreement to a third party without the prior written consent of Penguin. Any attempted assignment or transfer in violation of the foregoing will be null and void. If any

provision of this Agreement is held by a court of competent jurisdiction to be unenforceable, the remaining provisions of this Agreement will remain in full force and effect. This Agreement is governed by the laws of the State of California without reference to conflict of laws principles that would require the application of the laws of any other state. The United Nations Convention on Contracts for the International Sale of Goods shall not apply to this Agreement. All disputes arising out of this Agreement will be subject to the exclusive jurisdiction of the state and federal courts located in San Francisco County, California, and the parties agree and submit to the personal and exclusive jurisdiction and venue of these courts. Should you have any questions about this Agreement, or if you desire to contact Penguin, please contact us by mail at Penguin Computing, Inc., 45800 Northport Loop West, Fremont, CA 94538.

9.2 Third-Party License Agreements

GNU General Public License (GPL)

GNU GPL 1: <http://www.gnu.org/copyleft/gpl.html>

GNU GPL 2: <https://opensource.org/licenses/gpl-2.0.php>

GNU GPL 3: <https://opensource.org/licenses/gpl-3.0.html>

Red Hat RHEL License

Visit <https://www.redhat.com/en/about/agreements> to view the license for each specific location.

etcd

<https://github.com/etcd-io/etcd/blob/main/LICENSE>

Apache 2.0 license.

Telegraf, InfluxDB

<https://github.com/influxdata/influxdb/blob/master/LICENSE>

MIT License:

Copyright (c) 2-15-2018 InfluxData, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Grafana

<https://grafana.com/legal/grafana-labs-license/>

GNU Affero General Public License (AGPL) Version 3:

Compatible with GNU GPL v3, plus an AGPL requirement to make source code available if distributing any works based upon the licensed software.

jemalloc

<https://github.com/jemalloc/jemalloc/blob/dev/COPYING>

Unless otherwise specified, files in the jemalloc source distribution are subject to the following license:

Copyright (C) 2002-2018 Jason Evans <jasone@canonware.com>. All rights reserved.

Copyright (C) 2007-2012 Mozilla Foundation. All rights reserved.

Copyright (C) 2009-2018 Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

MPICH (formerly MPICH2)

<https://github.com/pmodels/mpich/blob/master/COPYRIGHT>

The following is a notice of limited availability of the code, and disclaimer which must be included in the prologue of the code and in all source listings of the code.

Copyright Notice

- 2002 University of Chicago

Permission is hereby granted to use, reproduce, prepare derivative works, and to redistribute to others. This software was authored by:

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne IL 60439

(and)

Department of Computer Science, University of Illinois at Urbana-Champaign

GOVERNMENT LICENSE

Portions of this material resulted from work developed under a U.S. Government Contract and are subject to the following license: the Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this computer software to reproduce, prepare derivative works, and perform publicly and display publicly.

DISCLAIMER

This computer code material was prepared, in part, as an account of work sponsored by an agency of the United States Government. Neither the United States, nor the University of Chicago, nor any of their employees, makes any warranty express or implied, or assumes any legal liability or responsibility for

the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

MUNGE

GNU GPL 3, from <https://slurm.schedmd.com/disclaimer.html>

Copyright (C) 2007-2018 Lawrence Livermore National Security, LLC.

Copyright (C) 2002-2007 The Regents of the University of California. UCRL-CODE-155910.

MUNGE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Additionally for the MUNGE library (libmunge), you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MUNGE is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License and GNU Lesser General Public License for more details.

MVAPICH

<http://mvapich.cse.ohio-state.edu/static/media/mvapich/LICENSE-MV2.TXT>

Copyright 2003-2018 The Ohio State University.

Portions Copyright 1999-2002 The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy). Portions copyright 1993 University of Chicago.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- (1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- (3) Neither the name of The Ohio State University, the University of California, Lawrence Berkeley National Laboratory, The University of Chicago, Argonne National Laboratory, U.S. Dept. of Energy nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OpenMPI

<https://www.open-mpi.org/community/license.php>

Most files in this release are marked with the copyrights of the organizations who have edited them. The copyrights below are in no particular order and generally reflect members of the Open MPI core team who have contributed code to this release. The copyrights for code used under license from other parties are included in the corresponding files.

Copyright (c) 2004-2010 The Trustees of Indiana University and Indiana University Research and Technology Corporation. All rights reserved.

Copyright (c) 2004-2017 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.

Copyright (c) 2004-2010 High Performance Computing Center Stuttgart, University of Stuttgart. All rights reserved.

Copyright (c) 2004-2008 The Regents of the University of California. All rights reserved.

Copyright (c) 2006-2017 Los Alamos National Security, LLC. All rights reserved.

Copyright (c) 2006-2017 Cisco Systems, Inc. All rights reserved.

Copyright (c) 2006-2010 Voltaire, Inc. All rights reserved.

Copyright (c) 2006-2017 Sandia National Laboratories. All rights reserved.

Copyright (c) 2006-2010 Sun Microsystems, Inc. All rights reserved. Use is subject to license terms.

Copyright (c) 2006-2017 The University of Houston. All rights reserved.

Copyright (c) 2006-2009 Myricom, Inc. All rights reserved.

Copyright (c) 2007-2017 UT-Battelle, LLC. All rights reserved.

Copyright (c) 2007-2017 IBM Corporation. All rights reserved.

Copyright (c) 1998-2005 Forschungszentrum Juelich, Juelich Supercomputing Centre, Federal Republic of Germany

Copyright (c) 2005-2008 ZIH, TU Dresden, Federal Republic of Germany

Copyright (c) 2007 Evergrid, Inc. All rights reserved.

Copyright (c) 2008 Chelsio, Inc. All rights reserved.

Copyright (c) 2008-2009 Institut National de Recherche en Informatique. All rights reserved.

Copyright (c) 2007 Lawrence Livermore National Security, LLC. All rights reserved.

Copyright (c) 2007-2017 Mellanox Technologies. All rights reserved.

Copyright (c) 2006-2010 QLogic Corporation. All rights reserved.

Copyright (c) 2008-2017 Oak Ridge National Labs. All rights reserved.

Copyright (c) 2006-2012 Oracle and/or its affiliates. All rights reserved.

Copyright (c) 2009-2015 Bull SAS. All rights reserved.

Copyright (c) 2010 ARM ltd. All rights reserved.

Copyright (c) 2016 ARM, Inc. All rights reserved.

Copyright (c) 2010-2011 Alex Brick . All rights reserved.

Copyright (c) 2012 The University of Wisconsin-La Crosse. All rights reserved.

Copyright (c) 2013-2016 Intel, Inc. All rights reserved.

Copyright (c) 2011-2017 NVIDIA Corporation. All rights reserved.

Copyright (c) 2016 Broadcom Limited. All rights reserved.

Copyright (c) 2011-2017 Fujitsu Limited. All rights reserved.

Copyright (c) 2014-2015 Hewlett-Packard Development Company, LP. All rights reserved.

Copyright (c) 2013-2017 Research Organization for Information Science (RIST). All rights reserved.

Copyright (c) 2017-2018 Amazon.com, Inc. or its affiliates. All Rights reserved.

Copyright (c) 2018 DataDirect Networks. All rights reserved.

Additional copyrights may follow

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

The copyright holders provide no reassurances that the source code provided does not infringe any patent, copyright, or any other intellectual property rights of third parties. The copyright holders disclaim any liability to any recipient for claims brought against recipient by any third party for infringement of that parties intellectual property rights.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

pdsh

"GPL", from <https://github.com/chaos/pdsh/blob/master/pdsh.spec>

Python 2.2 and beyond

<https://docs.python.org/3/license.html>

Python versions 2.2 and beyond are distributed under the Python Software Foundation (PSF) license, which PSF deems "GPL-compatible". This license is not the GNU GPL license because PSF allows for distributing a modified version of Python without making those changes open source. The license makes it possible to combine Python with other software that is released under the GPL; the others don't allow that.

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.
4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.
6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

Slurm Workload Manager - SchedMD

<https://slurm.schedmd.com/disclaimer.html>

Slurm is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Slurm is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

TORQUE Resource Manager

OpenPBS (Portable Batch System) v2.3 Software License

Copyright (c) 1999-2000 Veridian Information Solutions, Inc. All rights reserved.

For a license to use or redistribute the OpenPBS software under conditions other than those described below, or to purchase support for this software, please contact Veridian Systems, PBS Products Department ("Licensor") at:

www.OpenPBS.org	+1 650 967-4675	sales@OpenPBS.org
	877 902-4PBS (US toll-free)	

This license covers use of the OpenPBS v2.3 software (the "Software") at your site or location, and, for certain users, redistribution of the Software to other sites and locations. Use and redistribution of OpenPBS

v2.3 in source and binary forms, with or without modification, are permitted provided that all of the following conditions are met. After December 31, 2001, only conditions 3-6 must be met:

1. Commercial and/or non-commercial use of the Software is permitted provided a current software registration is on file at www.OpenPBS.org. If use of this software contributes to a publication, product, or service, proper attribution must be given; see www.OpenPBS.org/credit.html
2. Redistribution in any form is only permitted for non-commercial, non-profit purposes. There can be no charge for the Software or any software incorporating the Software. Further, there can be no expectation of revenue generated as a consequence of redistributing the Software.
3. Any Redistribution of source code must retain the above copyright notice and the acknowledgment contained in paragraph 6, this list of conditions and the disclaimer contained in paragraph 7.
4. Any Redistribution in binary form must reproduce the above copyright notice and the acknowledgment contained in paragraph 6, this list of conditions and the disclaimer contained in paragraph 7 in the documentation and/or other materials provided with the distribution.
5. Redistributions in any form must be accompanied by information on how to obtain complete source code for the OpenPBS software and any modifications and/or additions to the OpenPBS software. The source code must either be included in the distribution or be available for no more than the cost of distribution plus a nominal fee, and all modifications and additions to the Software must be freely redistributable by any party (including Licensor) without restriction.
6. All advertising materials mentioning features or use of the Software must display the following acknowledgment:

"This product includes software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and Veridian Information Solutions, Inc. Visit www.OpenPBS.org for OpenPBS software support, products, and information."

7. DISCLAIMER OF WARRANTY

THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED.

IN NO EVENT SHALL VERIDIAN CORPORATION, ITS AFFILIATED COMPANIES, OR THE U.S. GOVERNMENT OR ANY OF ITS AGENCIES BE LIABLE FOR ANY DIRECT OR INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This license will be governed by the laws of the Commonwealth of Virginia, without reference to its choice of law rules.

Addendum

To obtain complete source code for OpenPBS and modifications/additions provided in torque visit www.openpbs.org and/or www.supercluster.org/downloads.

FEEDBACK

We welcome any reports on errors or difficulties that you may find. We also would like your suggestions on improving this document. Please direct all comments and problems to support@penguincomputing.com.

When writing your email, please be as specific as possible, especially with errors in the text. Please include the chapter and section information. Also, please mention in which version of the manual you found the error. This version is ICE ClusterWare™ Release v12.4.2.

10.1 Finding Further Information

If you encounter a problem installing your cluster and find that the *Install* cannot help you, the following are sources for more information:

- The *Changelog* contains per-release specifics, and a *Known Issues And Workarounds* section.
- The *Administration* contains references to ClusterWare commands.

10.2 Contacting Penguin Computing Support

If you choose to contact Penguin Computing Support, you may be asked to submit a system information snapshot. Execute `scyld-sysinfo --no-tar` to view this snapshot locally, otherwise execute `scyld-sysinfo` to produce the compressed tarball that can be emailed or otherwise communicated to Penguin Computing.