Cisco COS Rolling Update Service User Guide

June 16, 2017

Cisco Systems, Inc.
www.cisco.com

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco website at www.cisco.com/go/offices.

Preface

This preface describes who should read the *Cisco COS Rolling Update Service User Guide* and explains its overall organization and document conventions. It contains the following sections:

Audience

This guide is for the networking professional managing the COS Rolling Update Service (RUS) product. Before using this guide, you should have experience working with Linux platforms and be familiar with the concepts and terminology of Ethernet, local area networking, clustering and high availability, and network services such as DNS and NTP.

Document Organization

This document contains the following chapters and appendices:

Chapters or Appendices	Descriptions
Overview	Describes RUS, its components, features, key concepts, and considerations for deployment.
Deploying RUS	Gives procedures for installing and configuring RUS and for debugging the update process if needed.
Using the RUS Text Client	Provides instructions for using the Rolling Update SHell (RUSH) text client for RUS.
Troubleshooting	Provides help for solving common problems associated with using RUS.

Document Conventions

This document uses the following conventions:

Convention	Indication
bold font	Commands and keywords and user-entered text appear in **bold** font.
italic font	Document titles, new or emphasized terms, and arguments for which you supply values are in *italic* font.
[ ]	Elements in square brackets are optional.
{x | y | z }	Required alternative keywords are grouped in braces and separated by vertical bars.
[x | y | z]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
`courier` font	Terminal sessions and information the system displays appear in `courier` font.
< >	Nonprinting characters such as passwords are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Tip Means *the following information will help you solve a problem*. The tips information might not be troubleshooting or even an action, but could be useful information, similar to a Timesaver.

Caution Means *reader be careful*. In this situation, you might perform an action that could result in equipment damage or loss of data.

Timesaver Means *the described action saves time*. You can save time by performing the action described in the paragraph.

Warning **IMPORTANT SAFETY INSTRUCTIONS**

This warning symbol means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents. Use the statement number provided at the end of

each warning to locate its translation in the translated safety warnings that accompanied this device.

SAVE THESE INSTRUCTIONS

Warning **Statements using this symbol are provided for additional information and to comply with regulatory and customer requirements.**

Related Publications

Refer to the following documents for additional information about RUS :

- *Release Notes for Cisco COS Rolling Update Service*
- *Cisco Cloud Object Storage Release 3.14.1 User Guide*

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation*.

To receive new and revised Cisco technical content directly to your desktop, you can subscribe to the *What's New in Cisco Product Documentation RSS feed*. The RSS feeds are a free service.

Overview

Introduction

When using distributed erasure coding (DEC), Cisco Cloud Object Storage (COS) requires a way to update COS on each node in a DEC cluster while maintaining access to user data stored on the cluster.

Resiliency provides part of the solution. If the cluster is properly designed for resiliency, updating COS on one node makes the data on that node temporary unavailable, but does not affect access to the data on the cluster as a whole. In principle, resiliency allows COS updates to execute sequentially or *roll* through the nodes in the cluster until the entire cluster is updated, without ever losing user data availability.

The COS Rolling Update Service (RUS) completes the solution by managing the rolling update process. Using RUS to update the nodes in a cluster provides the following benefits:

- Application data stored on the updating cluster remains available throughout the update.
- Any impact to cluster resiliency caused by the update is made predictable and manageable.
- Update policies can be defined to modify the update behavior.
- Update sources (product repository ISOs) are maintained at a single remote location available to all nodes, as opposed to requiring the update source on each node.
- Updates are defined and managed with minimal administrative input.
- Once initiated, cluster update requires no further administrative action other than monitoring the progress.
- Cluster updates can be paused, restarted, canceled, reset, and monitored.

RUS Resources

RUS combines and manages a variety of resources to achieve automated cluster-wide product updates. These include Clients, Cluster Managers, Clusters, Policies, and Update Sources such as Latent Feature List, Feature Disparity List, and Update Candidates.

The administrator registers these resources to RUS through a RUS client. Multiple instances of each resource type can be registered with RUS at the same time. Once registered, these resources can be properly associated by the administrator for a given cluster update event.

RUS Clients

RUS is an update server installed on a compute node outside the COS cluster. RUS provides a RESTful API for use by one or more clients executing elsewhere. The administrator currently uses a Python text client named Remote Update SHell (RUSH) to initialize and execute a cluster updates. RUSH can execute on the same node as RUS if desired. See Using the RUS Text Client, page 3-1 for details.

The RUS client must know the URL and port needed to access RUS. The default RUS port address is 2112, but this can be set to another value if required. For added security, Basic Authentication can also be used, but is not required.

Cluster Managers

One or more **cluster managers** interface with the actual cluster manager service such as Cisco Virtualized Video Processing Controller (V2PC). Multiple cluster managers can be registered with RUS at the same time, representing separate cluster manager services managing different sets of clusters.

Cluster managers provide RUS with the cluster definitions maintained by each manager. By default, registering a cluster manager populates RUS with the cluster context provided by the manager.

Cluster Context

Cluster context includes all nodes within the cluster, the given name of the node cluster manager, and the hostname. This context also includes subsets or groups of associated nodes within the cluster. In the case of COS, these subsets are resiliency groups.

RUS Policies

A unique set of **policies** is defined for each registered cluster to constrain the update process. The default value of each policy ensures the least risky update process. Mutable policies can only be changed prior to starting the cluster update. After the update starts, policies become read-only until the cluster is reset or until RUS is restarted. RUS ensures that these policies are honored throughout the update process.

Table 1-1 describes the currently defined policies and lists their default values.

Table 1-1 RUS Policies

Policy	Default Value	Description
concurrent_updates	none	'optimal': performs as many concurrent updates as possible within the constraints of other policies. 'limited': if the cluster is divided into groups then concurrency applies only to the nodes within each group. Each group is updated sequentially. 'none': each node is updated sequentially throughout the entire cluster.
continue_on_fail	no	'yes': continues the cluster update when a node fails to update. 'no': update process waits for the administrator to either continue or cancel the update.
maintain_availability	yes	'yes': update does not impact user data availability. This also means that the update will not start if it means that availability will be lost due to insufficient data resiliency. 'no': allow updates even if user data availability is lost. If concurrent_updates is 'optimal,' all nodes will go offline and update simultaneously.
pause	no	'yes': pause before starting the next node update. The update process is paused when all currently updating nodes are complete. The update process remains paused until the administrator continues the update. 'no': do not pause the update process at the end of each node update.

Table 1-1 RUS Policies

Policy	Default Value	Description
minimum_resiliency	<x>	This value defines the minimum level of resiliency to be maintained for the cluster during the cluster update. A positive value for x explicitly defines the desired minimum resiliency level. For example, a value of 1 for a COS cluster with an operational resiliency of 3 allows 2 nodes to update concurrently per resiliency group within the cluster, provided concurrent_updates is set to 'optimal'. A negative value for x represents the value relative to the cluster's operational resiliency. If operational resiliency is 3 and minimum_resiliency is set to -1, only one node per group is updated, maintaining a minimum resiliency of 2 potential node failures during the update. A negative value cannot reduce minimum resiliency below 0. This value is ignored if maintain_availability is set to ‘no’.
force_update	no	‘yes’: force an update regardless of the COS version currently installed on the cluster nodes, disregarding the recommendation in the update source. This policy can also be used to force an update on an inconsistent or incomplete cluster.
dry_run	no	'yes': reports which nodes will be updated and what will be updated on each node, but does not actually perform the update.

Update Sources

The **update source** resource represents a COS repository ISO (**cos_repo-<version>-x86_64.iso**) intended to be the source for a cluster update. The update source must be remotely accessible to all nodes in the cluster via HTTP or FTP.

The registered source contains a human-readable manifesto containing context relative to performing an update using the source and allows an administrator to make informed decisions and accommodations.

The manifesto contains the following update sources:

- Product Name and Version
- Update Qualifiers
- Latent Features List
- Feature Disparities
- Update Candidates
- Update Qualifiers

After the source is registered, the RUSH client ‘SHOW SOURCES’ command makes this manifesto context available to help the administrator make informed decisions and accommodations.

The manifesto may contain a list of qualifiers that could have an effect on the update. Table 1-2 provides a list of currently defined qualifiers.

Table 1-2 Update Source Qualifiers

Qualifier	Description
installation_only	The update source is an installation-only update. The product must be fully reinstalled as opposed to an in-place update. A scenario for this case would be a move to a new Linux distribution. It will be assumed that an installation will occur by virtue of a mass-deployment service when the node is rebooted by RUS during the update process.
no_revert	The update cannot be reverted by RUS.
node_interruption	Each node is placed out of service during the update of the node. User data accessibility is maintained if sufficient resiliency policies are in force.
cluster_interruption	The cluster user data is inaccessible during the update. This may be due to changes to the file system, database, protocols, or other changes that prevent an updated node from interacting with non-updated nodes. The entire cluster must be updated before data availability can be restored. Simultaneous updates can be enabled to expedite the update process.
feature_disparity	The update possesses at least one feature disparity.
latent_feature	The update possesses at least one latent feature.

Latent Features List

A **latent feature** is a feature that cannot be enabled before all nodes have been updated; for example, a change to the database schema or the utilization of cluster resiliency groups. If the 'latent_feature' qualifier is present, the manifesto also contains a list of the latent features.

Feature Disparity List

A **feature disparity** exists when an update alone will not enable a feature available in the update. Although an update may be performed successfully, a feature of this class is made available only after a full reinstallation of the product; for example, an update to the partition scheme to accommodate scale. In such cases, an administrator can choose to perform either an update or a full reinstallation. If the 'feature_disparity' qualifier is present, the manifesto also contains a list of the feature disparities.

Update Candidates

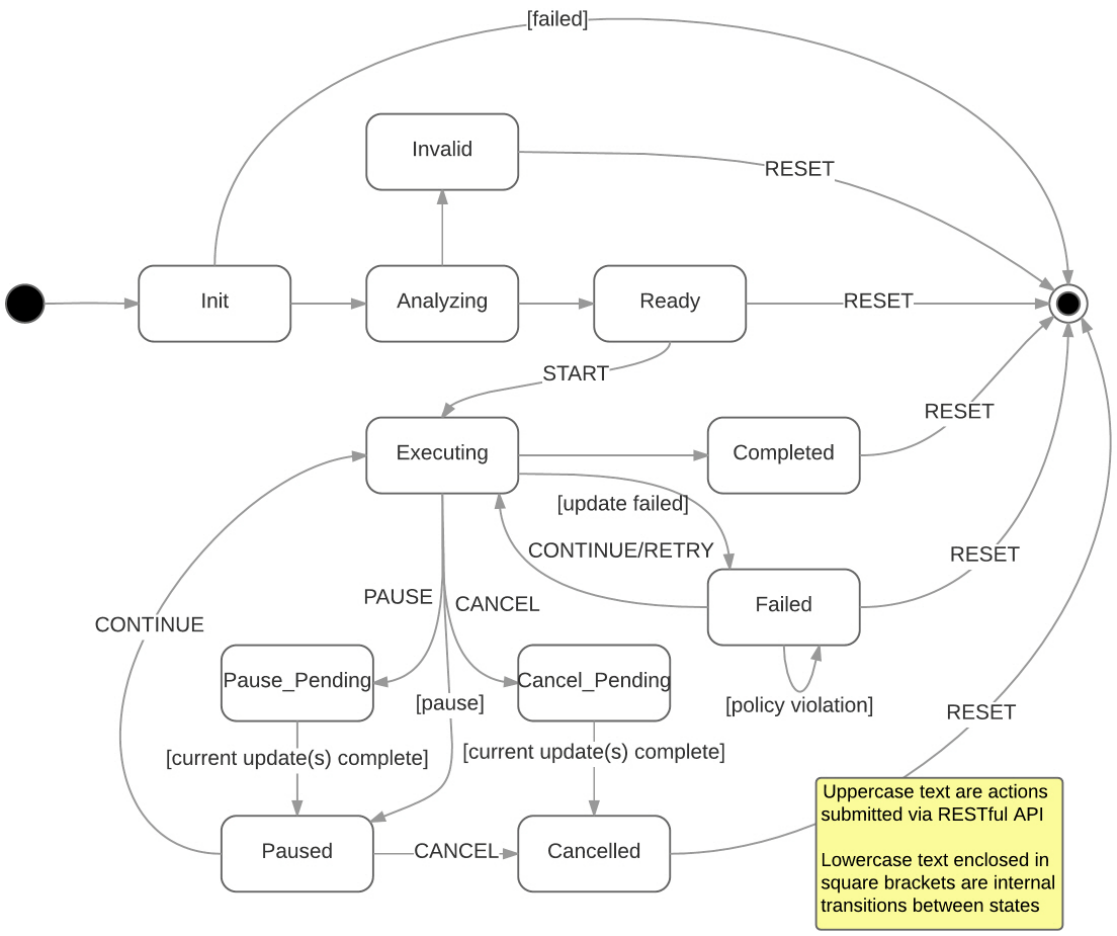
Update candidates are the qualified set of previous product versions supported by RUS and the update source. The update source defines one or more update candidates. By default, RUS will not allow an update from an unsupported version. This default behavior can be overridden by setting the 'force_update' policy to 'yes'.

Caution Updating an unsupported version can produce unexpected and often undesirable results.

Update Process

After a cluster manager and an update source is registered with RUS by the client, RUS can be instructed by the client to begin the update process. The update process will progress through several different states throughout the update process as illustrated below.

Figure 1-1 RUS Update Process



The Init, Analyzing, and Ready states occur before the update begins. Once the update process begins it remains in the Executing state while the cluster nodes are being updated. Without any intervention, the update process will either complete or fail.

An administrator can pause or cancel an executing update. A paused update can be continued or canceled. A canceled update will stop the update process after all nodes currently being updated are completed. A canceled update process can only be reset. A failed update can be retried, continued, or canceled. When an update fails, RUS stops the update after all nodes currently being updated are completed, unless overridden by the 'continue_on_fail' policy. Stopping the update gives the administrator an opportunity to remedy the update failure. Retrying a cluster update will try again to update all failed nodes, and then proceed to update all other pending nodes if the failure no longer occurs.

Continuing a failed cluster update skips the failed nodes and continues with the nodes pending an update. Continuing a failed update leaves the cluster update in a failed state after all remaining nodes have been updated. At this point, the failed update can still be retried to re-attempt an update on the failed node(s), or it can be reset.

Resetting a cluster will cause the cluster and node context to be reacquired and evaluated and also resets any update status, placing the cluster into a pre-update initial condition.

Update Phases

The update occurs in a defined set of phases. The update source provides a release-specific update RPM that contains the code for accomplishing the required tasks, if any, of each phase. Table 1-3 describes each of these phases.

Table 1-3 Update Phases

Phase	Description
INIT	Cluster-centric tasks performed prior to doing an update on any node. This phase executes on the first node only.
PREP	Tasks to be performed prior to the node being deactivated.
PRE	Tasks to be performed after the node is deactivated and prior to installing or updating the product.
INSTALL	Tasks specific to the actual installation or update of the product components.
REBOOT	This is an informal and conditional update phase that occurs as necessary in the update process. If a reboot is required during the update process, RUS will initiate, monitor, and report the reboot event. If the node fails to reestablish a connection with RUS after the reboot is initiated, the node is declared "UNREACHABLE" and the update fails on that node.
POST	Tasks that occur after update and prior to service restart.
RECOVERY	Recovery tasks that occur after the service has started. Typically, these tasks include checking system stability and preparedness for handling the deactivation of the next node.
FINI	Cluster-centric tasks performed after all nodes have been updated. This phase executes on the last node only.

Operational Resiliency

Operational resiliency refers to the number of object store nodes in a cluster that can be offline at the same time, for any reason, and still maintain access to all objects in the cluster.

The amount of resiliency built into the cluster has an impact on the cluster update. Greater resiliency allows more nodes to be updated concurrently, yielding a shorter the cluster update time, but at greater risk for potential loss of accessibility.

Operational resiliency is based on three factors:

- Resiliency of the cluster – Object store cluster resiliency is defined by the number of parity stripes created for each object written into the data store. The number of parity stripes is defined when the COS cluster is first defined (in V2PC, if COS is installed as a managed service of V2PC).
- Number resiliency groups in the cluster – Resiliency groups, introduced in COS Release 3.14.1, are static subsets of cluster nodes across which any given object is striped. Object resiliency is based solely on this set of nodes. A cluster can be divided into multiple groups according to the number of nodes in the cluster.
- Resiliency of the Content Database – The content database stores the object metadata required for accessing the objects in the object store. Prior to COS 3.14.1, this database was distributed across the object store's cluster nodes.

Based on the number of database nodes, the Cassandra database is configured with a replication factor of 1, 3, or 5, achieving resilience to 0, 1, or 2 node failures, respectively.

With COS 3.14.1, the content database can be placed on a dedicated content metadata cluster (CMC) external to the object store cluster to separate the database resiliency from the object resiliency.

Note This document refers to an external content metadata cluster as a CMC. All other references to a cluster refer to the object store cluster.

The cluster's operational resiliency is the lesser resiliency of the object store cluster and the content database, if the content database is distributed across the object store cluster. If the database is on the external CMC then the operational resiliency is the object store resiliency.

The object store's operational resiliency is the maximum number of nodes that can be offline before objects may become inaccessible. For example:

- If the cluster is not using a CMC, the database resiliency is 1, and the object store resiliency is 3, the operational resiliency is 1. This means that no more than 1 node in the entire cluster can be offline for the cluster to still provide access to all objects stored in the cluster.
- If this same cluster were using a CMC, the operational resiliency will be 3, indicating that no more than 3 arbitrary nodes can be taken offline before access to all objects may be lost.

The reason all objects *may* be lost in the case of using a CMC has to do with the introduction of resiliency groups. With resiliency groups, operational resiliency (that is, the number of parity stripes) applies to each group, not the cluster as a whole. Therefore, if operational resiliency were 3, three nodes from each resiliency group can be taken offline while maintaining access to all objects stored in the group.

If a large cluster had five resiliency groups with an operational resiliency of 3, then potentially 15 nodes could be updated simultaneously and the cluster still provide access to all objects stored within it. However, it must be precisely the correct set of 15 nodes. For example, given the very same large cluster, one could have only four nodes offline and lose access to a great number of objects stored in the cluster if all four nodes were in the same group.

Concurrent Updates

The default RUS update policy values establish the most conservative means of performing a cluster update: one node at a time provided that data availability can be maintained. If even one node update causes data availability to be compromised, the update will not start.

Setting the 'maintain_availability' policy to 'no' allows an update to start even with the possibility of losing access to stored objects throughout the cluster update process. Choosing different policy values enables RUS to allow multiple nodes to be updated concurrently.

Table 1-4 indicates how related policies work together to regulate concurrent updates with the CMC.

Table 1-4 Update Behavior for Different Policy Values with CMC

concurrent_updates =	If maintain_availability = yes	If maintain_availability = no
none	One node at a time across the entire cluster, provided availability is not lost.	One node at a time across the entire cluster.
limited	minimum_resiliency policy defines the number of concurrent nodes per group, each group updates serially.	All nodes in the group update concurrently. Each group updates serially.
optimal	minimum_resiliency policy defines the number of concurrent nodes per group, all groups update concurrently	All nodes in the cluster update concurrently.

Table 1-5 indicates how related policies work together to regulate concurrent updates without the CMC.

Note When the content database is distributed across the object store cluster nodes, resiliency groups are ignored.

Table 1-5 Update Behavior for Different Policy Values without CMC

concurrent_updates =	If maintain_availability = yes	If maintain_availability = no
none	One node at a time across the entire cluster, provided that availability is not lost.	One node at a time across the entire cluster.
limited or optimal	The minimum_resiliency policy defines the number of concurrent nodes.	All nodes in the cluster update concurrently.

Clusters Without DEC

For clusters that do not implement DEC, either update one node at a time or place the cluster offline and update all nodes concurrently.

Deploying RUS

A full installation of the Rolling Update Service (RUS) involves deploying the RUS service and SALT master on a computer or VM operating outside the COS cluster. RUS installation also involves deploying a SALT minion on each node of each COS cluster to be updated by RUS.

RUS requires that the update source (a COS repository ISO) be copied to and mounted on an FTP or HTTP server. This server can be the same as or different from the server on which RUS executes.

Note These instructions do not include the deployment and setup of an FTP or HTTP server.

The Cisco Virtualized Video Processing Controller (V2PC) used for COS management runs several virtual machines when fully deployed. V2PC also supports the creation of additional VM worker nodes for running ancillary services.

RUS can be deployed on a worker node and take advantage of conveniences provided by V2PC. Refer to **Installing and Provisioning the Cisco-COS Application on V2PC** in the "Deploying COS" chapter of the *Cisco Cloud Object Storage User Guide* for your COS release.

This section describes how to create a zone and worker. Create a new zone to which the RUS worker will be added; don't use the same zone associated with the COS application.

Installing the RUS Node

Although RUS can be installed anywhere JAVA 1.8 is installed, the following instructions are specific to installing RUS on a V2PC worker node.

Prerequisites

Before starting the installation, confirm that you have the following:

- V2PC VM worker node running CentOS 7 (described above)
- RUS installation ISO

The RUS installation ISO can be downloaded from the Cisco software downloads site at:

https://software.cisco.com/portal/pub/download/portal/select.html?&mdfid=286284263&softwareid=286306271

Installation Procedure

Step 1 Log in to the V2PC management node (a Linux node) as described in the *Cisco Virtualized Video Processing Controller Deployment Guide* for your V2PC release.

Step 2 Use the ssh identity file **v2pcssh.key** (copied to the management node during V2PC deployment) to log in to the worker node, and then switch to user root, as follows:

```
ssh -i v2pcssh.key v2pc@<worker IP> sudo su
```

Step 3 Copy or download the RUS installation ISO to the management node, mount the ISO, and then execute the installation script as follows:

```
mount -o loop <rus_repo iso> /mnt
/mnt/install.sh
```

Note To update an existing RUS installation, copy the latest RUS ISO to the worker node, mount the ISO, and then run the installation script as just described.

Installing COS Nodes

The COS cluster node only requires the SALT minion to be installed, configured, and running. For COS nodes with COS 3.14.1 or later installed, the Salt minion has already been installed, configured, running, and is waiting for the Salt master to come online.

For COS nodes running a COS version prior to 3.14.1, perform the following steps:

Step 1 Locate or create the file **/etc/salt/minion** and insert the following as the only line in the file:

master: master.cos-saltmaster.service.<datacenter>.<domain>

where the values for <datacenter> and <domain> are found in the file /etc/consul/consul.json. For example:

```
master: master.cos-saltmaster.service.region-0.v2pc.com
```

Step 2 Start the Salt minion as follows:

```
service salt-minion start
```

Step 3 Ensure that the Salt minion starts after subsequent reboots as follows:

```
chkconfig salt-minion on
```

RUS Startup

Choose one of the following methods to start RUS:

- Using Linux service manager, by executing:

  ```
  systemctl start rus
  ```

- Manually, by executing:

```
java -Dlog4j.configurationFile=/etc/opt/cisco/rus/log4j2.xml -cp
/opt/cisco/rus/bin/rus.jar:/opt/cisco/rus/lib/* Agent <options>
```

  where **<options>** are one or more options described in Command Line Options, page 2-3.

Note In addition to allowing for the use of command line options, starting RUS manually also allows RUS to display its messages to the console.

Command Line Options

Use any of the following command line options when starting RUS manually to override the definitions found in the RUS configuration file.

- **-p <port-number>**

  Sets the port value of the service. The default value is 2112.

- **-l <log-level>**

  Sets the level for messages recorded in the RUS log file or posted to the console. The value of <log-level> can be any of the following strings: 'fatal ', 'error', 'warn', 'info', or 'debug'. The default level is 'info' as defined in the /etc/opt/cisco/rus/log4j2.xml file.

- **-A <username:password>**

  Allows the administrator to define username and password credentials for using RUS.

Configuring RUS

RUS Configuration File

An optional JSON formatted configuration file created at **/etc/opt/cisco/rus/rus.conf** can contain static and default values for use by RUS. These values are placed in the **config** section, and can include the service port number, logging level control, and service credentials.

The configuration file can also include a **cluster_managers** array of comma-separated definitions that identify the cluster managers to be automatically registered with RUS each time RUS is started.

The following JSON syntax template shows the allowed settings for a **rus.conf** file:

```
{
    "config":
    {

        "port": <port number>,
        "log_level": "<'fatal', 'error', 'warn', 'info', or 'debug'>",
        "password": "<username>:<password>"

    },
    "cluster_managers": [
    {
        "name": "<user-defined cluster manager name>",
        "url": "<http or ftp url to the cluster manager>",
        "nature": "v2pc_v1",
```

```
        "auto_populate": "yes"
    } ]
}
```

Logging

While in operation, RUS maintains a log file at /var/log/rus/rus.log. RUS can be configured to record log entries that meet or exceed a defined log level. These log levels, from most urgent to least, are:

- **fatal**
- **error**
- **warn**
- **info**
- **debug**

By default, RUS logs all messages at level **info**. The log level can be set in various ways described elsewhere in this document. In addition, the **debug** level can be toggled at any time by sending the USR2 signal to the RUS process ID. For example:

```
kill -USR2 <pid>
```

Logging output is also controlled by the file **/opt/cisco/rus/etc/log4j2.xml**, which is read by RUS at each startup. Details on the format and use of the log4j2.xml file can be found at:

https://logging.apache.org/log4j/2.x/manual/configuration.html#XML

For example, to change the default log level used by RUS, change the <Root level=...> string from **info** to of the other defined log levels.

Default log4j2.xml File

The following is the default XML file used when RUS is installed. It defines the log level, the format of the log entries, and a rolling log file policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="RUS" packages="">
<Appenders>
<RollingFile name="File" fileName="/var/log/rus/rus.log"
filePattern="/var/log/rus/rus.log.%i">
<PatternLayout>
<Pattern>%d{yyyy-MM-dd HH:mm:ss} ${hostName} %m%n</Pattern>
</PatternLayout>
<SizeBasedTriggeringPolicy size="20 MB"/>
<DefaultRolloverStrategy max="5" fileIndex="min"/>
</RollingFile>
</Appenders>
<Loggers>
<Root level="info">
<AppenderRef ref="RollingFile"/>
</Root>
</Loggers>
</Configuration>
```

Debugging a Failed Update Process

If a cluster is configured and operating properly and RUS is set up correctly, an update, once started, will execute to completion without any further administrative intervention.

The client occasionally checks the status of the update, report progress, and offer an estimated time to completion. Depending on the client, this update status may be reported continually or on demand.

At the end of the update, RUS reports an update status of SUCCESS or FAILED. Accompanying this status is a collection of messages that have accumulated throughout the update process.

In the case of a failed update, these messages should include a cause for the failure. The client should also provide a means of reporting the individual node status. This information reports which nodes have successfully completed and which have not. There may also be node-specific messages relative to the update. This information is intended to be sufficient to lead the administrator to the cause of the failure.

A review of the RUS log entries may be helpful in diagnosing a failure. The RUS log file is located on the node running RUS at **/var/log/rus/rus.log**. An update log is also kept on each node in the file **/var/log/cos_update.log**. This node-side log file is recreated at each node update event. When viewing this file, ensure that its contents pertain to the most recent update attempt.

If an update fails and there is insufficient information to identify the failure, change the RUS log level to **debug**, reset, and reattempt the update. The RUS log file will now fill additional entries which, when interpreted correctly, should lead to the cause of an update failure.

Note The **debug** log level produces a large amount of technical information and is not intended for casual viewing.

For additional information on dealing with more common failure modes, see Troubleshooting, page 4-1.

Using the RUS Text Client

Introduction

The Rolling Update SHell (RUSH) is a command-line text client for managing the COS Rolling Update Service (RUS). RUSH is usually run as an interactive command shell using a **rush>** prompt, in which it supports command history and auto-tab completion of commands. RUSH also supports running individual commands via the command line by means of the **-e <command>** option.

In addition, RUSH supports execution of a list of commands from an input command file using the **-f <command-filename>** option. When using an input command file, all commands in the file are run sequentially unless an error or failure occurs, at which point execution stops.

Installing RUSH

The RUSH RPM is found on **rus_repo ISO**. RUSH is a Python application and can be installed and run on any system that:

- Has Python 2.6 installed
- Has access to RUS

By default, RUSH is installed on the same system where RUS is installed. This is done so that, when RUSH executes, it conveniently discovers RUS running on the same system, and does not have to be told where RUS is running.

Usage

RUSH can be run in any of the following ways:

- **rush [options]**
- **rush [host]**
- **rush [host] [port]**
- **rush [options] [host]**
- **rush [options] [host] [port]**

By default, RUSH connects to host **127.0.0.1** and port **2112**, which is enabled by default. If a host and (optional) port number are given on the command line, they take precedence over any defaults.

Options

The following options are available when running RUSH.

Note The options **--color** and **--connect-timeout=60 (seconds)** are the defaults when no options are specified.

- **--version**

  Displays the program version number and then exits.

- **-h, --help**

  Displays usage and help information related to how to run the program and then exits. Also outside of rush there is a man page available that may be viewed by typing **man rush** at the Linux command prompt.

- **-C, --color**

  Used to turn on color output for RUSH commands. This is the default setting.

- **--no-color**

  Used to turn off color output for RUSH commands.

- **--connect-timeout=<connection-timeout>**

  Specifies the connection timeout in seconds (default: 60 seconds).

- **--encoding=<encoding>**

  Specifies a non-default encoding for output. If you are experiencing problems with unicode characters, using utf8 may fix the problem. (Default from system preferences: ANSI_X3.4-1968)

- **-e <command>, --execute=<command>**

  Executes the specified command then exits.

- **-f <file>, --file=<file>**

  Executes the commands contained in the specified file until completion or until an error is encountered, then exits.

- **-u <username>, --username=<username>**

  Authenticate as user.

- **-p <password>, --password=<password>**

  Authenticate using password. If not provided on the command line, RUSH prompts for the password without echoing any characters to the screen.

- **--rushrc=<rushrc-path>**

  Specifies an alternative rushrc file location. By default, RUSH looks for the rushrc file in the hidden .rush directory in your home directory. You configure the rushrc file by setting the following options in the [ui] section of the file:

  [ui] options are:

  - **color = <true|false>**

    Always use color output.

  - **completekey = <key>**

    Use this key for auto-completion of a RUSH shell entry. Default is the tab key.

RUSH Commands

A number of commands are available to administer and manage the COS cluster rolling update process. Help for any command is available by typing **HELP [command]** at the RUSH prompt.

Installing RUSH also installs a **man** page. Execute **man rush** to display this document.

Registering Cluster Managers and Update Sources

Use the following commands to register a cluster manager or update a source ISO URL.

- **REGISTER CLUSTER_MANAGER <name> '<url>' [<nature> [no_auto_populate]]**

  Registers a cluster manager and assigns it the specified **<name>**. **'<url>'** is the URL for accessing the cluster manager, enclosed in single quotes. **<nature>** (optional) identifies the nature of the cluster manager. Valid values are **sim** or **v2pc_v1** (default).

  The optional **no_auto_populate** (or just **no** for short) can be specified to prevent RUS from automatically querying the cluster manager for cluster and node information.

  To view the current list of all registered cluster managers, use the command **SHOW CLUSTER_MANAGERS**.

Note When running on a V2PC worker node, RUSH attempts to automatically register a V2PC cluster manager named **v2pc** by using the information in the **/etc/consul/consul.json** file to auto-generate the appropriate V2PC URL.

- **REGISTER SOURCE <name> '<url>'**

  Registers an update source and assigns it the specified **<name>**. **'<url>'** is the URL for accessing a mounted COS repo ISO, enclosed in single quotes.

  To view the current list of all registered update sources, use the command **SHOW SOURCES**. This command also shows the source manifesto context including the product name, version, qualifiers, update candidates, and so on.

Viewing Cluster and Node Information

Use the following commands to view cluster and node information for any clusters managed by a registered cluster manager.

- **SHOW CLUSTERS [<cluster-name>]**

  Displays details for all clusters registered with RUS, or for the specified cluster if an optional **<cluster-name>** is provided.

- **SHOW NODES <cluster-name>**

  Displays details for all nodes belonging to the specified cluster.

Reviewing and Updating Policies

A number of policies control the cluster update process. Each policy is defined as a name-value pair, and can be of type **mutable** or **immutable**. The following commands are used to review the current policy settings and to update any mutable policies.

- **SHOW POLICIES <cluster-name>**

  Displays details for all policies assigned to the specified cluster.

- **UPDATE POLICY '<policy-name1>=<value1>[,<policy-name2>=<value2>,...]' <cluster-name>**

  Modifies one or more policies for the specified **<cluster-name>**. Each policy to be updated is specified as **<policy-name>=<value>**, enclosed in single quotes and delimited by commas (no spaces). For example:

  ```
  UPDATE POLICY
  'concurrent_updates=optimal,maintain_availability=yes,minimum_resiliency=1' my-cluster
  ```

  Only policies of type **mutable** can be modified.

Starting, Controlling, and Monitoring a Cluster Update

After a cluster manager and source have been registered and any desired policy changes have been made, the cluster update can be initiated. The following commands are used to start, control, and monitor the cluster update process.

- **START UPDATE <cluster-name> <source-name>**

  Starts an update operation for the specified **<cluster-name>** using the update source ISO image associated with **<source-name>**.

- **SHOW STATUS <cluster-name> [<watch-seconds>]**

  Displays the current update process status for the specified cluster and each node in the cluster. An optional **<watch-seconds>** value causes the status to be updated every **<watch-seconds>** seconds until the operator enters **<Ctrl-C>** or the update is completed.

- **CANCEL <cluster-name>**

  Cancels any active update operation for the specified cluster. A canceled update process cannot be continued.

- **CONTINUE <cluster-name>**

  Resumes a suspended or paused update operation for the specified cluster, or continues the update past a failed node.

- **PAUSE <cluster-name>**

  Pauses any active update operation for the specified cluster.

- **RESET <cluster-name>**

  Resets any active update operation for the specified cluster. This action is rejected when the update is in the **executing**, **pause_pending**, or **paused** state, but may be used for an update in the **completed** phase. After performing a reset, RUS returns to a suitable state for a new update.

- **RETRY <cluster-name>**

  Retries a failed update operation on the specified cluster after the cause of the failure has been corrected.

Viewing Logs

Logging entries are maintained on the server running RUS in the file **/var/log/rus/rus.log**. Use the following command to view log file entries:

SHOW LOG ['<file-name>']

This command displays up to 200 most recent RUS log entries on the console. Optionally, you can provide a filename (enclosed in single quotes) to which the same log output will also be written.

Cleanup

Use the following commands to unregister a cluster manager and update source, or to exit RUSH.

- **UNREGISTER CLUSTER_MANAGER <cluster-manager-name>**

  Unregisters the specified cluster manager.

- **UNREGISTER SOURCE <source-name>**

  Unregisters the specified update source.

- **EXIT (or QUIT)**

  Exits RUSH.

Note Exiting RUSH has no impact on RUS. If RUSH is restarted, it will find all RUS context in place as it was before exiting.

Troubleshooting

This section provides helps for solving common problems that may be encountered while using RUS and the RUSH client to update a cluster.

Configuration Issues

Cluster Definition Changes

The RUSH text client does not receive notifications from V2PC when changes to the cluster definition are made. If changes are made after the V2PC cluster manager is registered with RUS, unregister and then re-register the cluster manager to refresh the cluster context reported by the cluster manager.

Note Do not change the cluster definition while the cluster is being updated.

Cluster Contains Nodes Outside of Resiliency Groups

This condition is most likely due to disabled but accessible nodes in the cluster. The cluster group topology consists of active nodes only. The cserver service must be operational on a node for that node to be associated with a group. Executing **show log** from the RUSH text client will post the most recent log entries, one of which will be a list of hostnames identifying the unassigned nodes.

RUSH Example

```
rush> show clusters
Managed Clusters for Cluster Manager v2pc:
[v2pc-cluster]
Cluster State: INCONSISTENT
Node count: 49
Group count: 3
DEC data stripes: 10
DEC parity stripes: 3
Operational resiliency: 3
Cluster contains nodes outside of resiliency groups
Cluster 'v2pc-cluster': failed to start update due to cluster state 'INCONSISTENT'
rush> show log
...
... [Cluster] Cluster v2pc-cluster: unassigned nodes: [vcos-41, vcos-36, vcos-39]
...
```

An update on an inconsistent cluster can be performed by setting the **force_update** policy to **yes**.

Cluster Has Inconsistent Object Store Resiliency

The cluster is misconfigured. All nodes in a cluster must have the same object store resiliency. Object store resiliency is defined by the Distributed Erasure Coding (DEC) data and parity stripes values found in the **/arroyo/test/setupfile** for the node maintained by V2PC. Do not perform the update on this misconfigured cluster until the inconsistency is resolved.

After resolving the inconsistency, execute the RUSH command **RESET <cluster name>**. If the cluster is now reported as WHOLE, the update may be started. If DEC is not defined on any cluster node, the cluster is considered to be in a consistent state.

Nodes Outside of Resiliency Groups

Beginning with Release 3.14.1, COS supports the ability to subdivide a cluster into resiliency groups based on the DEC data and parity stripe numbers. Normally, all nodes in a DEC-enabled cluster are assigned to a group. RUS will not update unassigned nodes by default.

The list of unassigned nodes can be seen in a recent log entry. Use the RUSH command **show log <cluster name>** and look for the entry:

... Cluster <name>: unassigned nodes: <hostname>,<hostname>,...

If there are nodes outside of groups and none are expected, the following are the most likely reasons:

- For a node to be properly reported in their assigned group, the cserver service must be running on the node. Check all unassigned nodes and confirm that cserver is running, and if not, start it using the command **service cserver start**.
- Each node in the cluster must be able to communicate with other nodes across the data network. If a node is reachable from the management network but not from the data network, the node will be reported as unassigned to a group.
- The cluster is improperly configured. From the V2PC GUI, confirm that all cluster nodes are properly assigned to a group and that cosinit was executed with the proper profile reference.

Note A cluster that does not use DEC will have all its node implicitly assigned to the same group.

Could Not Obtain DB Resiliency

All nodes in a cluster must have the same database configuration. If RUS is not able to discover the node database configuration, this error occurs while the cluster is being initialized. The failure is most likely due to an improperly configured node. The Cassandra database is enabled but the cqlsh commands may not be reporting the expected information. Check the **rus.log** file for a stderr message from the relevant node indicating the problem.

Inactive Nodes

If all nodes are inactive, proper cluster state may not be obtainable. Normally, RUS is unable to start an update in this condition. Set the **force_update** policy to **yes** to have RUS update the cluster despite the node state. If only some nodes are inactive but reachable, RUS attempts to update the inactive nodes.

Partial Cluster

A partial cluster occurs when the cluster manager indicates that there are more cluster nodes than RUS is able to access. The fundamental means of access from RUS to each node in the cluster is by virtue of Salt. Therefore, a partial cluster can occur when either the cluster manager reports nodes that are not in the actual cluster or Salt is not active or properly configured on some cluster nodes.

Execute **SHOW NODES <cluster name>** from RUSH to see the nodes that are unreachable. Check the V2PC cluster configuration and correct it as needed, or correct the accessibility problem on these nodes. See Deploying RUS, page 2-1 for instructions on setting up the Salt minion on the cluster node.

Incomplete Resiliency Groups

A defined group has nodes that are not accessible. This can be due to the following:

- A node is accessible from the data network but not from the management network.
- A misconfiguration between V2PC and the cluster that results in cserver being able to access nodes it believes are in a group, but these nodes are not included in the cluster definition provided by the cluster manager.

Update Related Issues

'PRE' Phase Timeout

During the PRE phase, COS services are disabled on the node. Rarely, this disabling of services gets hung, and more rarely, csrever may crash. This failure is most often due to uncommon cases involving timed out communications or an unexpected system state. The timeout causes the update to fail. Evaluate the node condition, remedy as needed, restart the node, and then retry the update or reset and restart it.

'INSTALL' Phase Failure

During the INSTALL phase, a number tasks are performed that are capable of some type of failure. An error message often accompanies the failure notice, and should be reported by RUSH. Examining the node's **/var/log/cos_update.log** file should reveal the exact point of failure. If the failure occurs while yum is updating the RPMs, also check **/var/log/yum.log** for error messages.

Reboot Timeout

Either the reboot did not complete, the network interfaces did not come up after reboot, or the Salt minion was not successfully started. If possible, log in to the node and start the Salt minion, or if started, use the **chkconfig** utility to enable the Salt minion service.

Unreachable Nodes

During the INSTALL phase for some product releases, the Salt minion running on the cluster node may be updated. During the Salt minion update, the Salt master will be unable to reach the node. This should be a momentary condition, lasting approximately 30 seconds. If the node remains unreachable for a much longer period during the INSTALL phase timeout, there may be a system crash or network failure requiring administrative attention.

Nodes Stuck on TRANSITION State after a Successful Update

Execute the RUSH command **RESET <cluster name>** to force the refresh of cluster status, followed by **SHOW NODES <cluster name>** to see the latest node status. If problem persists, ensure that all services are started at the node in question: **cosd**, **cserver**, and **cassandra** if using the on-cluster DB. If any of these services are stopped, use **service <servicename> start** to start them.

RUSH Client Timeout

By default, the RUSH client waits for up to 1 minute for the RUS server to respond to RESTful API calls. In some cases, typically only seen when simultaneously updating all nodes of a large cluster, RUS may be unable to respond within this period, causing RUSH to report a timeout error.

If this occurs, simply retry the timed-out RUSH command. If the timeout error continues, extend the default timeout period using **--conn ect-timeout=<seconds>** option when starting RUSH. For example, to set the RUSH timeout value to 5 minutes, start RUSH using **rush --connect-timeout=300 <rus-hostname> <rus-port>**.

RUS Terminates Unexpectedly

If an update was in progress when RUS was unexpectedly terminated, an **Update Cluster** semaphore will remain in force in the V2PC doc server. While this semaphore is present in the doc server, the cluster will be in Maintenance mode and will not return to full operational state until the semaphore is removed. Use the RUSH command **RESET <cluster name>** to reset the cluster and remove this semaphore.

SALT Related Issues

Removing Antiquated Salt Minion Keys

This action is sometimes required when a node is reinstalled. First remove the node from the cluster in V2PC, and then remove the node's Salt key from the Salt master.

Execute the following commands at the Salt master to identify the node's key and delete it:

```
salt-key
salt-key -y -d <the-antiquated-key>
```

After the key is removed, reinstall the COS node. When COS is installed and started, a new Salt key is regenerated automatically, provided the COS node has access to the Salt master. The new Salt master must accept the new Salt key as described in Regenerating Minion Keys, page 4-5.

Moving to a New Salt Master

When moving to a new Salt master:

Step 1 Disable the Salt master at its current location and start it at its new location.

Step 2 Confirm that each cluster node has network access to the new Salt master.

Step 3 On each cluster node, remove **/etc/salt/pki/minion/minion_master.pub** and restart the minion as follows:

```
service salt-minion restart
```

Note New Salt keys may need to be accepted by the new Salt master. See Regenerating Minion Keys, page 4-5 for details.

Regenerating Minion Keys

To regenerate the COS node Salt key, simply start or restarting the Salt minion. After allowing 30-60 seconds for the Salt minion to generate the new key, use the **salt-key** command to confirm that the new key appears in the Salt master key list.

If the new Salt key appears in the Unaccepted Keys list, execute **salt-key -a <new key>** to have the Salt master accept the new key.