

HDP Operations: HDP Administration 2

Student Guide

Rev 1.1





Copyright © 2012 - 2016 Hortonworks, Inc. All rights reserved.

The contents of this course and all its lessons and related materials, including handouts to audience members, are Copyright © 2012 - 2015 Hortonworks, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Hortonworks, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Hortonworks, Inc. Hortonworks, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

All Apache Projects are registered trademarks of the Apache Software Foundation.

All other trademarks are the property of their respective owners.



Become a **Hortonworks Certified Professional** and establish your credentials:

- HDP Certified Developer: for Hadoop developers using frameworks like Pig, Hive, Sqoop and Flume.
- HDP Certified Administrator: for Hadoop administrators who deploy and manage Hadoop clusters.
- HDP Certified Developer: Java: for Hadoop developers who design, develop and architect Hadoop-based solutions written in the Java programming language.
- HDP Certified Developer: Spark: for Hadoop developers who write and deploy applications for the Spark framework.
- HDF Certified Professional: for DataFlow Operators responsible for building and deploying HDF workflows.

How to Register: Visit www.examslocal.com and search for “Hortonworks” to register for an exam. The cost of each exam is \$250 USD, and you can take the exam anytime, anywhere using your own computer. For more details, including a list of exam objectives and instructions on how to attempt our practice exams, visit <http://hortonworks.com/training/certification/>

Earn Digital Badges: Hortonworks Certified Professionals receive a digital badge for each certification earned. Display your badges proudly on your résumé, LinkedIn profile, email signature, etc.





Self Paced Learning Library

On Demand Learning

Hortonworks University Self-Paced Learning Library is an on-demand dynamic repository of content that is accessed using a Hortonworks University account. Learners can view lessons anywhere, at any time, and complete lessons at their own pace. Lessons can be stopped and started, as needed, and completion is tracked via the Hortonworks University Learning Management System.

Hortonworks University courses are designed and developed by Hadoop experts and provide an immersive and valuable real world experience. In our scenario-based training courses, we offer unmatched depth and expertise. We prepare you to be an expert with highly valued, practical skills and prepare you to successfully complete Hortonworks Technical Certifications.

Target Audience: Hortonworks University Self-Paced Learning Library is designed for those new to Hadoop, as well as architects, developers, analysts, data scientists, and IT decision makers. It is essentially for anyone who desires to learn more about Apache Hadoop and the Hortonworks Data Platform.

Duration: Access to the Hortonworks University Self-Paced Learning Library is provided for a 12-month period per individual named user. The subscription includes access to over 400 hours of learning lessons.

The online library accelerates time to Hadoop competency. In addition, the content is constantly being expanded with new material, on an ongoing basis.

Visit: <http://hortonworks.com/training/class/hortonworks-university-self-paced-learning-library/>

Table of Contents

Performing an HDP Rolling Upgrade	1
Lesson Objectives.....	1
HDP Stacks	1
Ambari and Stacks	2
HDP Stack Version Numbers	2
Viewing HDP Stacks	3
Ambari and Compatible Stack Versions.....	3
HDP Upgrade Method.....	4
Upgrade Methods.....	4
HDP Rolling Upgrades	6
Rolling Upgrades Support Downgrades.....	6
Support for Rolling Upgrades	7
Upgrade Path Restrictions.....	9
Pre-Rolling Upgrade Checklist	9
Performing a Rolling Upgrade	10
General Prerequisites.....	10
Service-Specific Prerequisites.....	11
Database Preparation	11
HDFS Preparation.....	11
Registering a New HDP Version	12
Installing a New HDP Version	16
Initiating the Rolling Upgrade	17
Older Version Removed	27
Knowledge Check.....	29
Questions.....	29
Answers	30
Summary	31
Configuring Heterogeneous HDFS Storage	33
Lesson Objectives.....	33
Heterogeneous HDFS Storage	33
Prior HDFS Storage Model.....	33
Current HDFS Storage Model	34
Storage Preferences.....	34

Archive Storage	35
HDFS Storage Types	36
HDFS Storage Operations	36
Storage Policies.....	36
Data Block Replica Placement Rules.....	37
Storage Policy Resolution.....	37
Labeling DataNode Storage Devices.....	38
Configuring Storage Types and Policies.....	38
Configuring a Storage Policy	38
HDFS Mover.....	39
Viewing Block Placement Information.....	40
Knowledge Check	41
Questions.....	41
Answers	42
Summary	43
Managing the HDFS NFS Gateway	45
Lesson Objectives.....	45
HDFS NFS Gateway Use Cases	45
HDFS NFS Gateway Architecture and Operations	46
NFS Gateway Scalability	47
User Authentication	47
Proxy User	48
Access Controls	49
Reordering Sequential Writes	50
Installing and Configuring HDFS NFS Gateway	50
Installing an NFS Gateway	51
Selecting Nodes for Ambari Agents	51
Assigning Slaves and Clients.....	53
Choosing Configuration Groups	53
Reviewing and Deploying NFS Gateway.....	54
Update Access Time Property	55
Verifying NFS Gateway Startup	55
Configuring HDFS NFS Gateway Client	56
Starting and Stopping HDFS NFS Gateway Service	57
Knowledge Check	59

Questions	59
Answers	60
Summary	61
Configuring HDFS Centralized Cache	63
Lesson Objectives.....	63
HDFS Centralized Cache	63
Use Cases	64
Cache Operation.....	64
Determining Cache Locality.....	64
Cache Control.....	67
Cache Pools and Directives.....	67
Configuring Centralized Cache	68
Optional Properties	69
Managing Cache Pools and Directives.....	69
Cache Pool Command Options	70
Cache Directive Command Options	71
Removing a Cache Directive and Cache Pool	73
Knowledge Check	75
Questions.....	75
Answers	76
Summary	77
Managing Data Compression	79
Lesson Objectives.....	79
File Formats and Compression	79
Roles for Managing Compression.....	79
Benefits and Considerations	80
Ratio Pros and Cons	80
Serdess and Codecs	81
Record vs. Block Compression	81
Splittable Compression Formats.....	82
Common Hadoop File Types	82
Hadoop Compression Algorithms	83
Best Compression Algorithm	83
Common Hadoop Codecs	83
Configure New Cluster Algorithms.....	84

Defining Default MapReduce and Tez Algorithms	85
MapReduce Compression	85
Tez Compression.....	85
Knowledge Check	87
Questions.....	87
Answers	88
Summary	89
YARN Node Labels	91
Lesson Objectives.....	91
Scenario.....	91
Organizations, Queues, and SLAs	91
The HDP Cluster – Node View	92
Requirements	92
YARN Node Labels	93
Creating Node Labels	93
Node Assignment Without Labels	93
Node Assignment with Labels	94
Adding a Label to a Node	95
Creating Node Labels.....	96
Modifying or Removing Node Labels.....	96
Removing Node Labels	97
Assigning and Removing a Node Label	97
Node and Label Relationships.....	98
Configuring Queues to Access Node Label Resources	99
Enabling Node Labels in the Root Queue	99
Assigning Labels to Individual Queues	100
Undoing Node Labels.....	101
Node Labels and Data Locality.....	103
Knowledge Check	105
Questions.....	105
Answers	106
Summary	107
Deploying Applications using Apache Slider	109
Lesson Objectives.....	109
About Apache Slider	109

Slider Logical View	110
Slider Components.....	110
Key Slider Benefits	111
Cluster-Level Administration	111
Cluster-Level Slider Deployment	112
Slider View Installation	113
Slider Packages.....	115
Deploying a Slider Package for the Slider View.....	116
Deploying and Managing Slider Applications.....	117
Flex Slider Applications.....	118
Slider Applications in ResourceManager UI	120
Stopping and Destroying a Slider Application	120
Knowledge Check	123
Questions.....	123
Answers	124
Summary	125
Integrating Ambari with LDAP.....	127
Lesson Objectives.....	127
Ambari User Sources and Permissions	127
Ambari Users and Groups.....	129
Ambari User and Group Permissions	131
Admin Privileges for Local vs. LDAP/AD Users	131
Integrating Ambari Server with LDAP	132
Setting LDAP Properties on Ambari Server	133
LDAP Synchronization.....	136
Knowledge Check	139
Questions.....	139
Answers	140
Summary	141
Hive Tuning	143
Lesson Objectives.....	143
Tune Hive for Interactive Queries	143
Batch Versus Interactive Processing.....	143
Why Tune Hive?.....	144
Tuning for Interactive Queries.....	144

Summary	149
Apache Hive High Availability	151
Lesson Objectives.....	151
Apache Hive	151
Structured and Unstructured Data	151
Hive Architecture Components.....	152
Submitting Hive Queries.....	153
Hive HA Requirements and Benefits.....	154
Hive HA Requirements	154
Hive HA Benefits.....	155
Hive HA Architecture and Operation.....	156
Metastore Service HA Architecture	157
WebHCat Server HA Architecture.....	158
Installing Hive and Configuring Hive HA	159
Installing Hive	159
Configuring Hive HA	165
Adding Redundant HiveServer2 Instances	167
Adding Redundant WebHCat Servers	169
Verify Hive HA Access	171
Knowledge Check	173
Questions.....	173
Answers	174
Summary	175
Managing Workflows Using Apache Oozie	177
Lesson Objectives.....	177
Apache Oozie	177
Oozie Workflow	178
Oozie Coordinator	179
Oozie Bundle	182
Oozie Benefits	182
Oozie Architecture.....	183
Installing and Configuring Oozie Using Ambari.....	184
Choosing Services.....	184
Assigning Masters	185
Assigning Slaves and Clients.....	186

Customizing Services.....	187
Reviewing and Deploying.....	189
Installing, Starting and Testing	189
Completing the Install	190
Restarting Services	190
Managing Oozie with Ambari	190
Reading Oozie Log Files	192
Deploying and Managing Oozie Jobs	192
Deploying and Executing a Workflow.....	192
Job Properties File Configuration.....	193
Oozie Web UI	193
Command Line Administration	195
Knowledge Check	197
Questions.....	197
Answers	198
Summary	199
Apache Oozie High Availability	201
Lesson Objectives.....	201
Apache Oozie	201
Oozie Component Architecture	202
Apache Oozie HA	203
Oozie HA Component Architecture	203
Accessing Oozie Job Logs	204
Configuring Oozie HA	205
Configuring the Virtual IP Address	205
Adding Another Oozie Server	205
Confirm the Installation	205
Monitoring the Oozie Server Installation	206
Starting the New Oozie Server.....	206
Verifying Redundant Oozie Servers.....	207
Configuring Oozie HA-related Properties.....	207
Restart the Oozie Service	209
Updating HDFS Oozie Proxy User.....	209
Restarting Services Indicated by Ambari.....	209
Listing Oozie Servers	210

Knowledge Check	211
Questions	211
Answers	212
Summary	213
Introduction to Falcon	215
Lesson Objectives.....	215
Data Governance Challenges.....	215
Getting Data into the Cluster	215
Data Decisions in a Multi-Cluster Environment	217
Data Decisions in a Fully Utilized HDP Environment.....	218
Data Lifecycle Considerations	218
Complex Data Pipelines.....	219
Late Data and Reprocessing	221
Lineage.....	222
Metadata Governance.....	223
Falcon Overview.....	223
Data Lifecycle Management	223
Pipeline Development and Management	223
Falcon Architecture	225
Deploying Falcon	226
Choosing Services.....	226
Assigning Masters	227
Assigning Slaves and Clients.....	227
Customizing Services.....	228
Reviewing and Deploying.....	229
Installing, Starting, and Testing Falcon	229
Completing the Installation	230
Verifying the Service.....	231
Falcon Entities and Mirroring	231
Pipeline Abstraction	232
Creating Entities	235
Falcon Search UI	243
Viewing Log Files.....	246
Managing Dependencies	246
Lineage.....	247

Knowledge Check	249
Questions	249
Answers	250
Summary	251
Automate Cluster Provisioning Using Ambari Blueprints	253
Lesson Objectives.....	253
Blueprint Scenarios.....	253
Scenario 1: Test/Dev Cluster Deployment	254
Scenario 2: Cluster Scale-Out.....	255
Benefits	256
Blueprint Usage Overview	257
View Existing Cluster Configuration	257
Step 1: Capture Blueprint.....	258
Step 2: Install Ambari Server.....	259
Step 3: Register Blueprint File with Ambari	260
Step 4 (Optional): Local Repo Configuration Files.....	261
Step 5 (Optional): Set Local Repo Locations	262
Step 6: Create Cluster Creation Template	263
Step 7: Deploy Cluster Creation Template.....	263
Step 8: Install Ambari Agents.....	264
Step 9: Register Ambari Agents with Ambari Server	264
Step 10: Confirm Successful Cluster Deployment.....	265
Add Hosts via Blueprints.....	265
Logical Cluster Definition File.....	266
“Blueprints” Section	266
“host_groups” Section	267
“configurations” Section	271
Cluster Creation Template File	273
General Cluster Deployment Settings	273
Simple Cluster Creation Template Example	277
Host Creation Template File	278
Specify Nodes Host Creation Template Example.....	278
Configuration Property Best Practices	279
Precedence Diagram.....	279
Sensitive Configuration Information in Files	280

Knowledge Check	281
Questions.....	281
Answers	282
Summary	283

Performing an HDP Rolling Upgrade

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Recall the definition of an HDP stack and interpret its version number
- ✓ View the current stack and identify compatible Ambari software versions
- ✓ Recall the types and methods of upgrade available in HDP
- ✓ Describe the rolling upgrade process, restrictions, and pre-upgrade checklist
- ✓ Perform a rolling upgrade using the Ambari Web UI

HDP Stacks



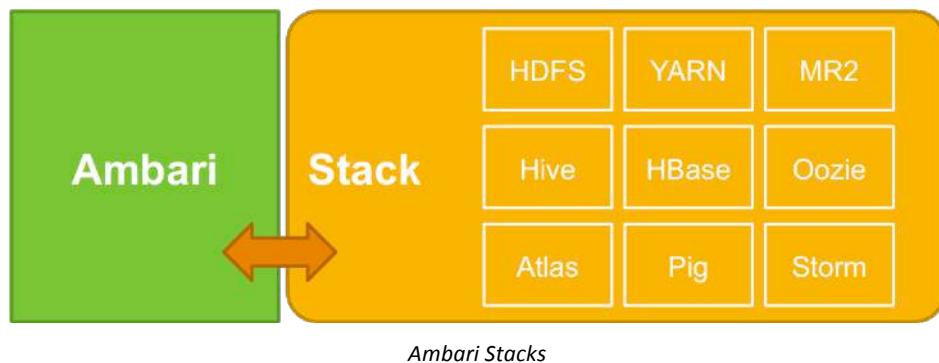
HDP Stack

A Stack and its associated services are defined in a Stack definition. A Stack defines a set of services and where to obtain the software packages for those services. A Stack can have one or more versions, and each version can be active or inactive. For example, the HDP Stack used by Ambari can include versions 2.2 and 2.3, but only one of those versions is active on a cluster node at any one time. The concept of active and inactive versions is tied to the upgrade process.

A service defines a set of components that make up the service. For example, the NameNode and DataNode are components of the HDFS service. The ResourceManager and NodeManager are components of the YARN service.

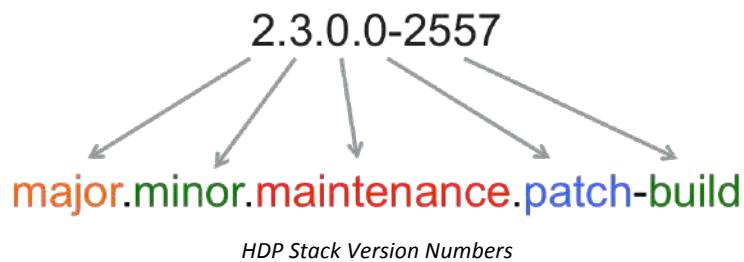
Components have a defined lifecycle. For example, they can be installed, started, stopped, and restarted.

Ambari and Stacks



Ambari uses the concept of a Stack to install software. By leveraging the Stack definition, Ambari has a consistent and defined interface to install, manage, and monitor a set of services. Ambari is also easily extensible by adding new Stack versions with new services.

HDP Stack Version Numbers



The full HDP stack version number consists of five fields. The fields are illustrated and labeled here:

- The first field represents the major version number.
The current major version is version 2.
- The second field represents the minor version number.
The current minor version number is version 3.
- The next two fields represent the maintenance number and the patch number.
- The final field is the software build number.

Viewing HDP Stacks

Stack Versions			
Service	Version	Status	Description
HDFS	2.7.1.2.3	Installed	Apache Hadoop Distributed File System
MapReduce2	2.7.1.2.3	Installed	Apache Hadoop NextGen MapReduce (YARN)
YARN	2.7.1.2.3	Installed	Apache Hadoop NextGen MapReduce (YARN)
Tez	0.7.0.2.3	Installed	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
Hive	1.2.1.2.3	Installed	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
HBase	1.1.1.2.3	Add Service	A Non-relational distributed database, plus Phoenix, a high performance SQL layer for low latency applications.
Pig	0.15.0.2.3	Installed	Scripting platform for analyzing large datasets
Sqoop	1.4.6.2.3	Installed	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
Oozie	4.2.0.2.3	Installed	System for workflow coordination and execution of Apache Hadoop jobs. This also includes the installation of the Oozie UI.

The Admin > Stack and Versions > Stack Window

The Ambari Web UI can display the current HDP stack. To view the HDP stack, select **Stack and Versions** from the **Admin** menu, and then click the **Stack** tab.

The browser pane lists each installed component, along with its version number, whether it is installed, and a brief description. An **Add Service** link is provided to begin the installation process for any uninstalled component.

Ambari and Compatible Stack Versions

		HDP version			
		2.3	2.2	2.1	2.0
Ambari version	2.1	✓	✓	✓	✓
	2.0		✓	✓	✓
	1.7		✓	✓	✓

Ambari and HDP Stack Compatibility

Hortonworks recommends upgrading to the latest version of Ambari prior to upgrading HDP.

If planning to upgrade HDP, Ambari needs to be able to recognize, install, and manage the target HDP stack version. Use this table to determine compatible Ambari and HDP versions.

HDP Upgrade Method

HDP supports three types of upgrades:

- **Major**

A major upgrade results in a change of the major number.

For example, an upgrade from version 1.x to 2.x would be a major upgrade.

- **Minor**

A minor upgrade results in a change of the minor number.

For example, an upgrade from version 2.2 to 2.3 would be a minor upgrade.

- **Maintenance**

A maintenance upgrade results in a change of the maintenance number.

For example, an upgrade from version 2.3.0 to 2.3.1 would be a maintenance upgrade.

Upgrade Methods

HDP also supports two upgrade methods; an in-place upgrade and a rolling upgrade.

In-Place Upgrade

In-Place Upgrade	<ul style="list-style-type: none">• Installs a new version over the old version• Is a manual process• Requires one or more service disruptions<ul style="list-style-type: none">• Steps include stopping <i>all</i> services in cluster• Is not reversible (downgradeable)• Is possible in all HDP versions• Is possible for major, minor, and maintenance upgrades
------------------	--

HDP In-Place Upgrade Method

An in-place upgrade installs a new version of the software over the older version. It is a manual and labor-intensive process. To perform an in-place upgrade, each service must be stopped, upgraded, and restarted. Because most clusters include multiple services, there will be multiple service disruptions. Typically an in-place upgrade results in one or more services being upgraded during the upgrade procedure.

An in-place upgrade is not reversible through a downgrade operation. One advantage of an in-place upgrade is that it is possible for all HDP versions and all types of upgrades including major, minor, and maintenance upgrades.

To perform an in-place upgrade, refer to the detailed instructions in the Hortonworks documentation at <http://docs.hortonworks.com>.

Rolling Upgrade

Rolling Upgrade	<ul style="list-style-type: none">• Installs a new version <i>next to</i> the old version• Is an automated process orchestrated by Ambari• Minimizes service interruption<ul style="list-style-type: none">• Varies with the service being upgraded• Preserves cluster customizations, data, and metadata• Is reversible until specifically finalized by administrator• Is supported starting with HDP 2.2• Is supported for minor and maintenance upgrades
------------------------	---

HDP Rolling Upgrade Method

The rolling upgrade feature fundamentally involves software upgrade of an active Hadoop cluster from one version of the Hadoop software stack to another. The advantage of rolling upgrade is that the cluster can be upgraded in stages without requiring a complete shut down of all its applications and jobs. Some applications can continue to actively use the cluster while the upgrade is in progress.

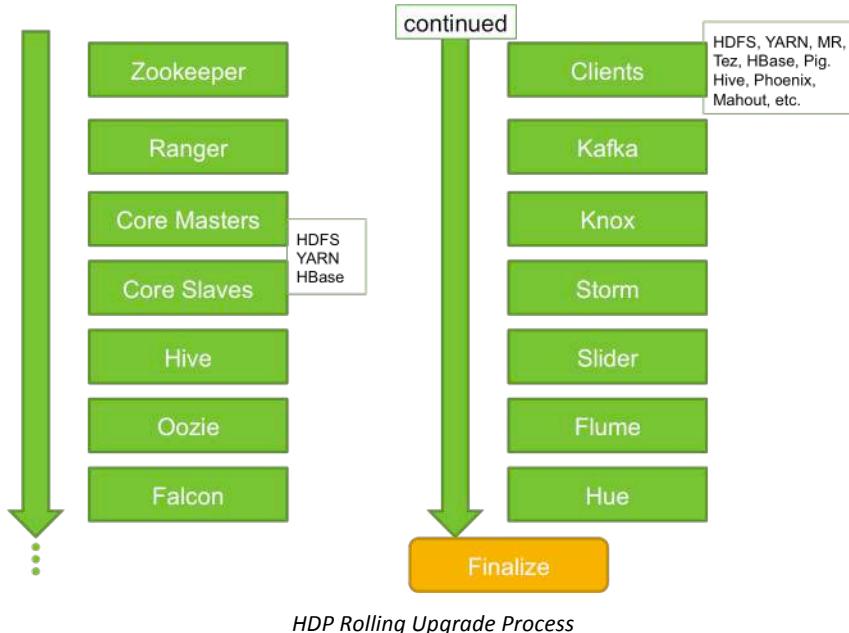
A rolling upgrade installs a new version of the software next to the older version. It is an automated process orchestrated by Ambari. To perform a rolling upgrade, each service must be stopped, upgraded, and restarted. However, with HDFS, YARN, HBase, Hive, and Oozie HA configured, service disruption is minimized. Not all services have HA configurations so some services will be disrupted more than others. Service disruption is described in more detail later in this lesson.

A rolling upgrade preserves cluster configurations, metadata, and data and is reversible through a downgrade operation. Downgrades are described later in this lesson. Once a rolling upgrade is finalized by an administrator, it cannot be downgraded or rolled back.

Minor and maintenance rolling upgrades are supported starting with the HDP 2.2 release. It is assumed, but not guaranteed, that rolling upgrade will be supported for a major upgrade from 2.x to 3.x.

This lesson focuses on rolling HDP upgrades.

HDP Rolling Upgrades

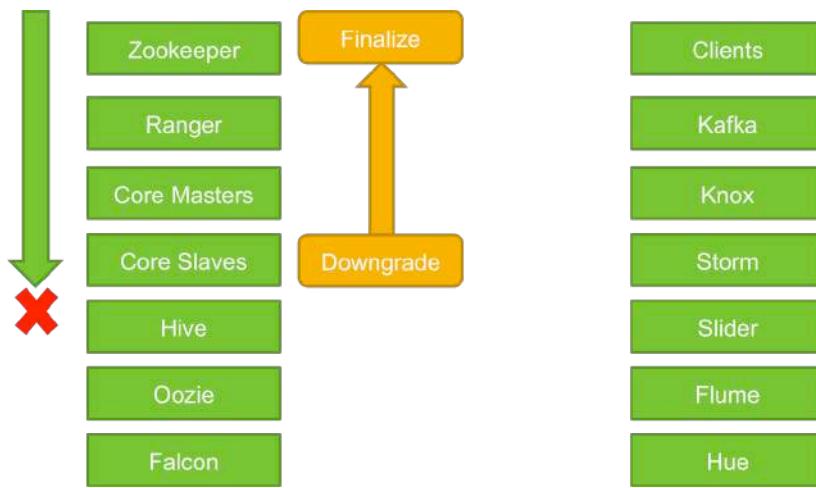


HDP has a safe, certified process for performing a rolling upgrade. The cluster upgrade is accomplished using a set of steps orchestrated by Ambari. The cluster services are switched over to the new version of software in a rolling fashion. Any components that are not installed on the cluster are skipped. A successful upgrade is finalized at the end and the new version of the software becomes the current version of the cluster.

Rolling Upgrades Support Downgrades

A rolling upgrade can be reversed and downgraded at many points during a rolling upgrade operation.

Downgrade Operations



HDP Rolling Upgrade Downgrade Operation

Rolling upgrade supports downgrade operations. A downgrade operation enables server, worker, and client components to be reverted to their prior software version, however, any new data or metadata that has been created is maintained.

Typically, the cluster administrator will do a partial upgrade, restricting it to a subset of the cluster, observe the behavior of the newer stack, and finally continue with the upgrade of the remaining cluster if the partial upgrade went well. After the partial upgrade, and before committing to a cluster-wide upgrade, administrators can also choose to downgrade the cluster's subset back to the previous version as needed. This might be the case, for example, if a serious upgrade related problem is encountered, performance has degraded, or some applications are not running well. To support this type of behavior, downgrade is offered at key points throughout the rolling upgrade process.

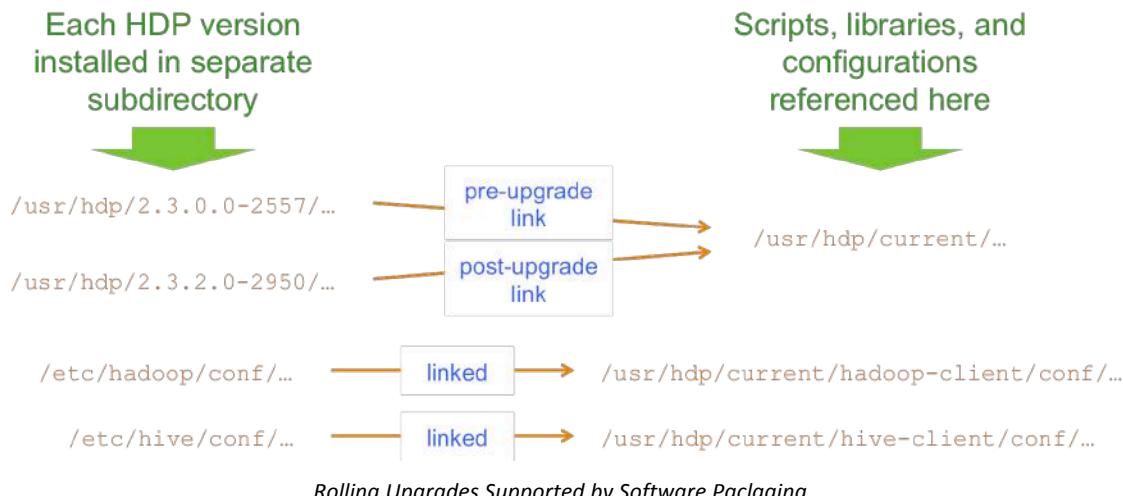
In addition, if Ambari detects an upgrade problem then downgrade is offered at that point too. In the illustration, a downgrade operation can be started after the core slaves and core masters have been upgraded if Ambari or an administrator detects a problem at that point. The downgrade can be finalized once all upgraded components have been reverted to their prior version.

Support for Rolling Upgrades

Key areas that required significant engineering to implement for support of rolling upgrades included:

- Software Packaging
- MapReduce and Tez

Supported by Software Packaging



The way in which early versions of the HDP software were packaged and installed only supported in-place upgrades. Starting with version 2.2, HDP software is now packaged and installed to support rolling upgrades.

In order to support rolling upgrades, a key requirement is to have multiple versions of the Hadoop stack installed side-by-side on each machine as the cluster is rolled from one version to the next. HDP has implemented a solution where it leverages standard package management approaches (RPMs, Debians, etc.) and adds the version number to the package name. This has two advantages. System administrators can continue to use and rely on their existing tooling and best practices – an advantage against proprietary packaging formats. Further, in addition to allowing automated install and upgrade via Ambari, the solution enables administrators who prefer their own installation scripts and mechanisms to perform rolling upgrades themselves.

Each version of HDP is installed into its own subdirectory on the cluster nodes. In the illustration, the HDP 2.3.0.0 software is installed into the `/usr/hdp/2.3.0.0-2557` subdirectory. If this subdirectory is linked to the `/usr/hdp/current/` subdirectory, then it is the current version of the software for the cluster. Any scripts used to manage the cluster, along with any libraries or configuration files used to run the cluster are referenced by their path through the `/usr/hdp/current/` directory.

A newer version of the HDP software can be installed next to the older version, and without the need to remove the older version. In the illustration, the HDP 2.3.2.0 version of the software is installed into the `/usr/hdp/2.3.2.0-2950` subdirectory. When a rolling upgrade is finalized, this subdirectory is linked to the `current` directory and the 2.3.2.0 version becomes the current version of the software on the cluster. Finalizing an upgrade also causes the removal of the subdirectory containing the older version of the software. Prior to finalizing an upgrade, it is possible to downgrade to the earlier version of the software by re-linking the older subdirectory back to the `current` directory.

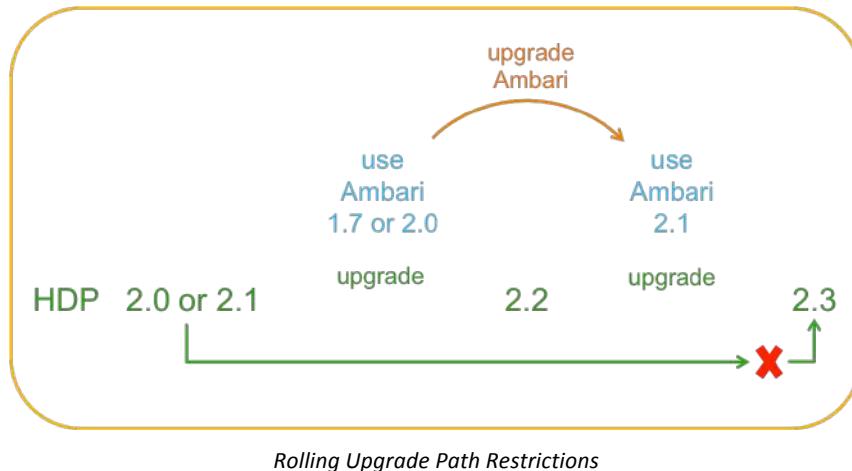
Even configuration files are linked to a specific version. As examples in the illustration, the `/etc/hadoop/conf` directory is linked to the `/usr/hdp/current/hadoop-client/conf` directory while the `/etc/hive/conf` directory is linked to the `/usr/hdp/current/hive-client/conf` directory. This type of linking is performed for all Hadoop software projects.

MapReduce/Tez Support

A challenging problem during an upgrade is that running applications must maintain the context based on the version from which they were submitted rather than the current running version of YARN. Before HDP 2.2, where the upgrade model was a shut-down-and-upgrade, applications like Hadoop MapReduce and Tez depended upon a consistent local installation of libraries on all the cluster machines. This includes both client and server machines. This meant that the context of the MapReduce or Tez client and the context of the YARN cluster had to be identical. When rolling upgrades or downgrades start in a cluster, the versions of local installation of the libraries will definitely change while an application is in progress; resulting in inconsistent behavior, and possibly runtime errors.

To solve this problem, in HDP 2.2, HDP no longer depends on local installation but instead uses HDFS to store multiple versions of the libraries that are under use. Based on the YARN distributed cache feature, all application containers can now consistently execute using the version of libraries that matches the context of the application at the time of submission, regardless of what version the platform corresponds to on any node. Once the platform upgrade succeeds and stabilizes, administrators can, at their own pace, explicitly upgrade all the frameworks in the cluster along with their clients. An added benefit of this approach is that different users can run their applications using different versions of libraries and upgrade at will.

Upgrade Path Restrictions



Ambari 2.1 does not support directly upgrading from HDP 2.0 or 2.1 to HDP 2.3. In order to upgrade HDP 2.0 or 2.1 to HDP 2.3, you must:

- 1 . First use Ambari 1.7 or 2.0 to upgrade HDP to 2.2.
- 2 . Once completed, upgrade Ambari 1.7 or 2.0 to Ambari 2.1.
- 3 . Then use Ambari 2.1 to complete the HDP upgrade from 2.2 to 2.3.

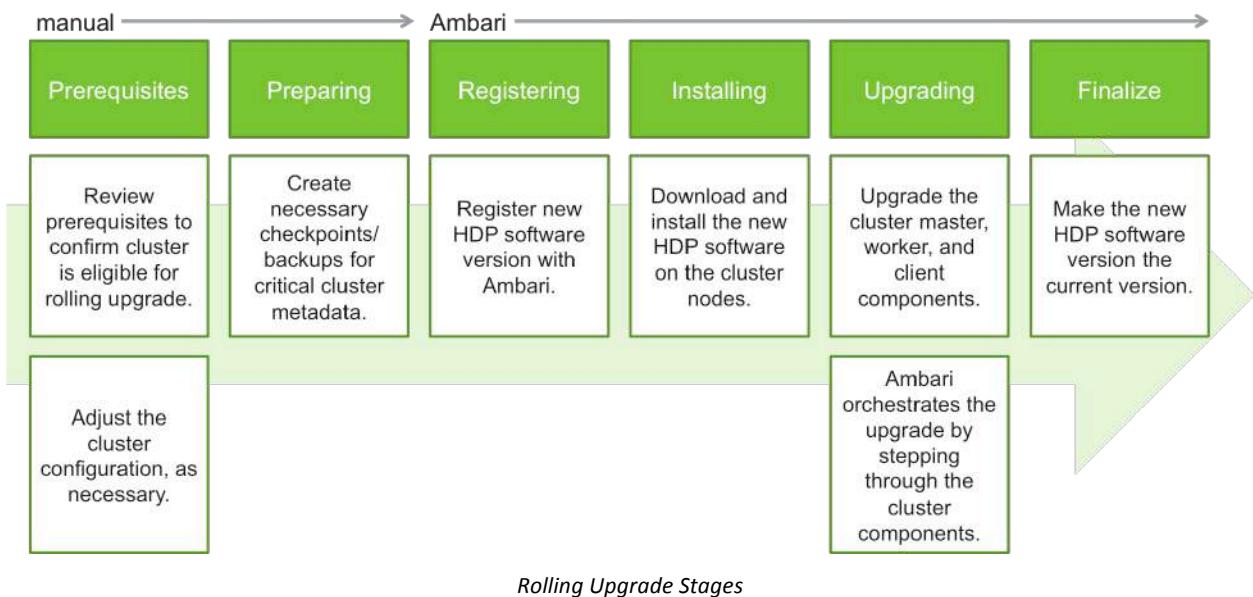
Pre-Rolling Upgrade Checklist

Before upgrading HDP, Hortonworks strongly recommends reviewing this checklist. The purpose is to confirm that the cluster is healthy and will experience minimal service disruption before attempting an upgrade.

- Use the Ambari Web UI to ensure that all services in the cluster are running.
- For each service in the cluster, run the **Service Check** on the service's **Service Actions** menu to confirm that the service is operational. Service checks are used extensively during rolling upgrade so if they fail when run manually, they will likely fail during upgrade too.
- For each service use the **Stop** and **Start** buttons in the Ambari Web UI to verify that the service can be stopped and started. Services are repeatedly stopped and started during upgrade. If they fail when initiated manually, they will likely fail during upgrade too.
- Understand and, as necessary, remediate any Ambari alerts.
- Ensure that you have an up-to-date backup of any supporting databases, including Hive, Ranger, Oozie, and any others.
- Enable HDFS, YARN, HBase, and Hive HA to minimize service disruption.
- Ensure that each cluster node has at least 2.5 GB of disk space available for each HDP version. The target installation directory is `/usr/hdp/<version>`.
- Operationally, be aware that new service user accounts might be created to support new software projects that were not installed as part of the earlier HDP release. For example, these new user accounts might need to be added to an LDAP server or created as Kerberos principals.

Performing a Rolling Upgrade

Performing a rolling upgrade is a multi-stage, multi-step process.



Rolling Upgrade Stages

The prerequisites and preparing stages are sets of manual steps performed by an administrator while the remaining stages are orchestrated by Ambari using information supplied by an administrator using the Ambari Web UI. This illustration provides an overview of each stage of the process. Each stage is described in more detail later in this lesson.

General Prerequisites

Prerequisite	Description
Current HDP version	Must be running HDP 2.2 or later.
Ambari agent heartbeats	Ambari agents must be running and heartbeating to the Ambari server.
Host maintenance mode	Any hosts in maintenance mode cannot be hosting master service components.
Service maintenance mode	No services can be in maintenance mode.
Services running	All services must be running and able to pass Ambari Service Checks.

General Prerequisites

General prerequisites apply to every cluster.

Service-Specific Prerequisites

Service-specific prerequisites apply only to a cluster running specific types of services. Several services have service-specific prerequisites. These include:

- HDFS
- YARN
- MapReduce2
- Tez
- Hive
- Oozie

Service prerequisites can change per HDP version. Most prerequisites require minimal effort to meet.

For specific information about a specific service, refer to the Ambari Upgrade Guide found at <http://docs.hortonworks.com>.

Database Preparation

As is true with many upgrade procedures in a data center, it is best to take precautions that protect against the possible loss or corruption of data. Hortonworks recommends performing a backup of any databases that support cluster operation. This includes the Ambari, Hive Metastore, Oozie, Ranger admin, and Ranger audit databases. Because these databases are supported on several database software platforms, refer to the vendor-specific documentation when performing these database backups.

HDFS Preparation

HDFS must be prepared prior to performing an upgrade. The following steps should be completed as the HDFS superuser on the Active NameNode.

1) First check for a previous HDFS upgrade that was never finalized.

- a) Any prior HDFS upgrade must be finalized before proceeding with the new upgrade.
- b) Determine the directory path used by the NameNode to store the `edits` and `fsimage` files. This directory path is listed in the `dfs.namenode.name.dir` property in the `hdfs-site.xml` file. If the cluster was installed by the Ambari then the directory path is likely `/hadoop/hdfs/namenode`.

The easiest way to determine if there is an incomplete HDFS upgrade is to look for a subdirectory named `previous`.

For example, look for the directory path `/hadoop/hdfs/namenode/previous`. If such a directory is found, finalize the HDFS upgrade by running the command `hdfs dfsadmin -finalizeUpgrade`.

2) Next create reference files that can be used after the rolling upgrade of HDFS to help verify that the rolling upgrade was successful and has not damaged or removed any data or metadata from HDFS.

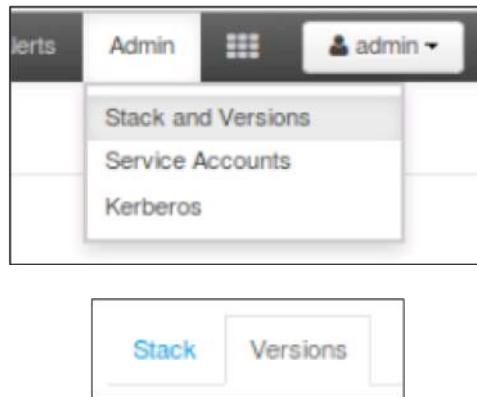
- a) Run the command `hdfs fsck / -files -blocks -locations > hdfs-preupgrade-fsck.log`.
- b) After an upgrade, the `fsck` command can be run again with its output stored to another log file. The two log files can be compared if there is any concern about the state of HDFS.

- 3) The command `hdfs dfsadmin -report > preupgrade-list-of-datanodes.log` can be used to capture a list of DataNodes before the upgrade.
 - a) After the upgrade, the `hdfs dfsadmin` command can be run again with its output stored to another log file. The two log files can be compared if there is any concern about the number of DataNodes recognized by the cluster.
- 4) Next create a current HDFS checkpoint using the following three commands.
 - a) First, enter safemode by running the command `hdfs dfsadmin -safemode enter`.
 - b) Next create the checkpoint by running the command `hdfs dfsadmin -saveNamespace`.
 - c) Then exit safemode by running the command `hdfs dfsadmin -safemode leave`.
- 5) To ensure that HDFS is recoverable in the event that something would go wrong during the upgrade, back up the current checkpoint files to another directory.
 - a) For example, run a command similar to `cp /hadoop/hdfs/namenode/current/* /backup-location`.

Registering a New HDP Version

To perform a rolling upgrade you must register the new target HDP version with Ambari. Registering an HDP version makes it available for use and management by Ambari.

Selecting Stack and Versions



Registering a New HDP Version

To register a new HDP version, log in to the Ambari Web UI as an administrator. Select **Stack and Versions** from the **Admin** menu. Then click **Versions** to open the Versions pane.

Manage Versions



The Ambari Manage Versions Pane

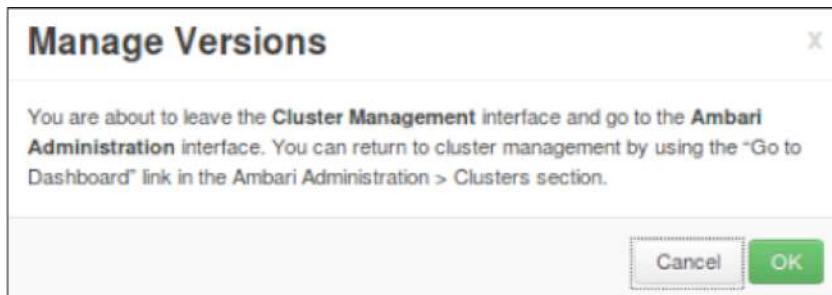
The current HDP version is displayed in the Versions pane. The Versions pane displays the current HDP version installed on the cluster. Associated with the version are three numbers labeled Not Installed, Installed, and Current.

Not Installed refers to the number of cluster nodes for which a new HDP version is registered but not installed. In the example the number is zero because no new HDP software has been registered.

Installed refers to the number of cluster nodes for which a new HDP version is registered and installed, but is not the current version. In the example the number is zero because no new HDP software has been registered.

Current refers to the number of cluster nodes for which the HDP version is installed and is the current version. In the example the number is three because there are three nodes in the cluster and all of them have HDP version 2.3.0.0 as the current version.

To register a new HDP version with Ambari click the **Manage Versions** button.

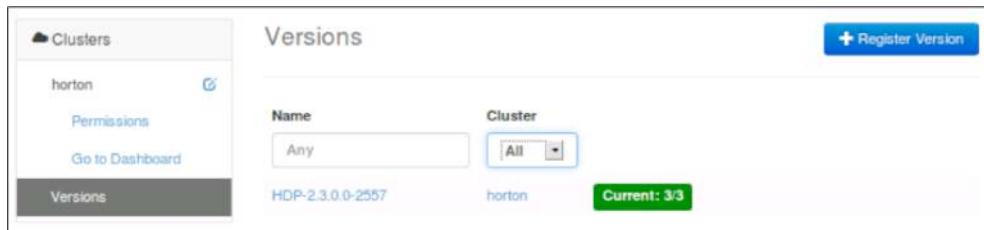


Manage Versions Dialog Window

Read the message in the Manage Versions dialog window and then click **OK** to continue registering a new HDP version with Ambari.

Performing an HDP Rolling Upgrade

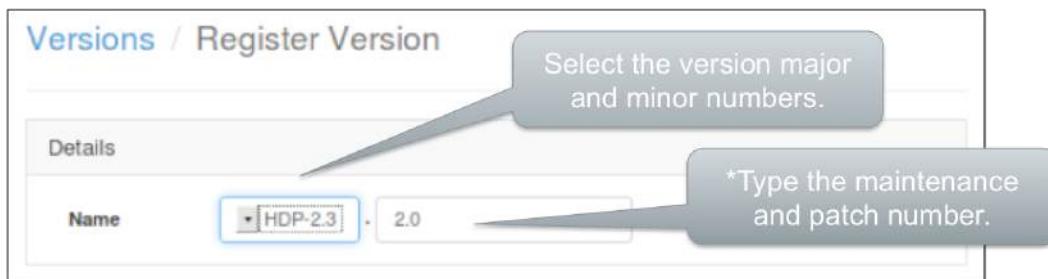
Register Version



The Register Version Button

Click the **Register Version** button.

Desired Major and Minor Numbers



Selecting Major and Minor Numbers

Use the drop-down menu to select the desired major and minor numbers. Then type the desired maintenance and patch numbers. In the example, 2.3 was selected from the menu and 2.0 was typed in the text box.

It is not necessary to know ahead of time the build number for the software version. Ambari will automatically detect and display the build number later once the software has been downloaded and installed.

Base URL

The screenshot shows a 'Repositories' configuration window. At the top, a note says: 'Provide Base URLs for the Operating Systems you are configuring. Uncheck all other Operating Systems.' Below is a table with columns: OS, Name, and Base URL.

OS	Name	Base URL
redhat6	HDP	<input type="text" value="http://public-repo-1.hortonworks.com/HDP/centos6/2.x/upgrade"/>
	HDP-UTILS	<input type="text" value="http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/"/>
redhat7	HDP	<input type="text"/>
	HDP-UTILS	<input type="text"/>
suse11	HDP	<input type="text"/>
	HDP-UTILS	<input type="text"/>

At the bottom, there is a checkbox for 'Skip Repository Base URL validation (Advanced)' and two buttons: 'Cancel' and 'Save'.

Selecting the Base URL

Once the HDP version has been selected, scroll down in the window to add the base URL information for the HDP and HDP-UTILS software. There are different base URLs for each of the supported base operating systems.

First, click the check box for the base operating system that is used in the cluster. Then type the base URLs for the HDP and HDP-UTILS software.

The base URLs used will commonly vary from cluster to cluster. For example, some companies and clusters might use a local repository. If this is the case then the base URLs should reference the local repository. You would have to know these URLs ahead of time in order to enter them here.

Other companies or clusters might use the public Hortonworks repositories available on the Internet. This is only practical if network bandwidth availability or security concerns do not limit the capability of using the public repositories. In this case the URLs for the Hortonworks public repositories can be found by clicking the **HDP Repositories** link found in the Registering a New Version section of the online *Ambari Upgrade Guide* at <http://docs.hortonworks.com>.

Installing a New HDP Version

The screenshot shows the Ambari 'Versions' interface. At the top, there are filters for 'Name' (set to 'Any') and 'Cluster' (set to 'All'). Below the table, a message says '2 of 2 versions showing - clear filters'. Navigation buttons at the bottom include a page size dropdown set to 10, and buttons for 'Previous', '1', and 'Next'.

Selecting Cluster to Start Installation Process

After the new HDP version has been registered with Ambari it must be downloaded and installed on the cluster nodes. Click **Install on** and then select the cluster name from the menu. In the illustration the cluster name is **horton**.

Installing the Package

The screenshot shows the Ambari 'Stack Versions' interface. It lists two versions: HDP-2.3.0.0-2557 (Current) and HDP-2.3.2.0. The HDP-2.3.2.0 card includes a 'Manage Versions' button and an 'Install Packages' button. Callouts provide instructions: one points to the 'Install Packages' button with the text 'Registered, but not installed on any hosts.', and another points to the 'Manage Versions' button with the text 'Edit the repositories, if necessary.'

The Install Packages Button

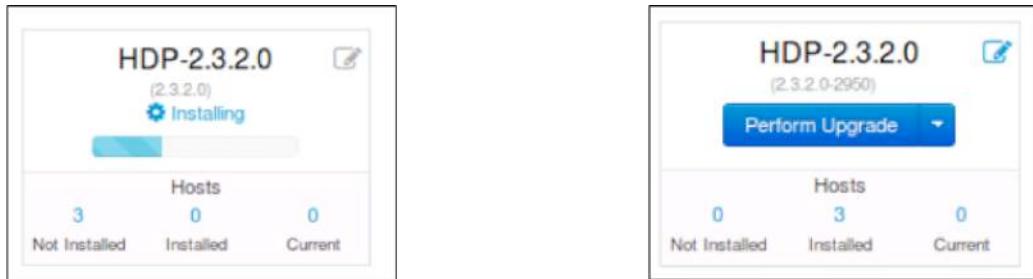
Notice that there is a new HDP version registered with Ambari but it is not installed on any cluster hosts.

To install the new HDP version on the nodes click the **Install Packages** button. When the confirmation window opens (not shown), click **OK** to continue.

If package installation fails it could be caused by a Base URL misconfiguration. To resolve this click the edit icon to update the base URLs, as necessary.

Performing an HDP Rolling Upgrade

Monitoring the Installation



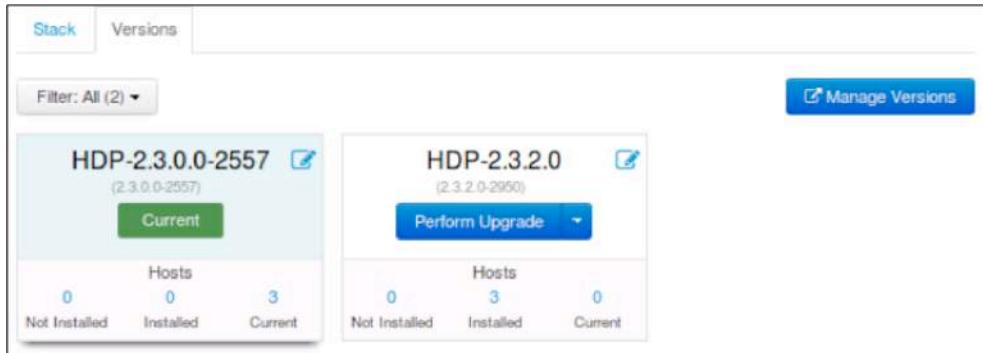
Monitoring the Installation

During software installation a status bar displays installation progress. After installation is complete the number of hosts Not Installed should be zero while the number of Installed hosts should equal the number of nodes in the cluster.

The **Perform Upgrade** button includes a drop-down menu with the choice to **Reinstall Packages**. This is provided in the case that package installation failed on one or more nodes and a retry is necessary. It is also useful in the case where additional nodes are added to the cluster after new software has been installed but before an upgrade has been run.

Initiating the Rolling Upgrade

Now that the software has been installed, the next step is to initiate the rolling upgrade of the cluster components.

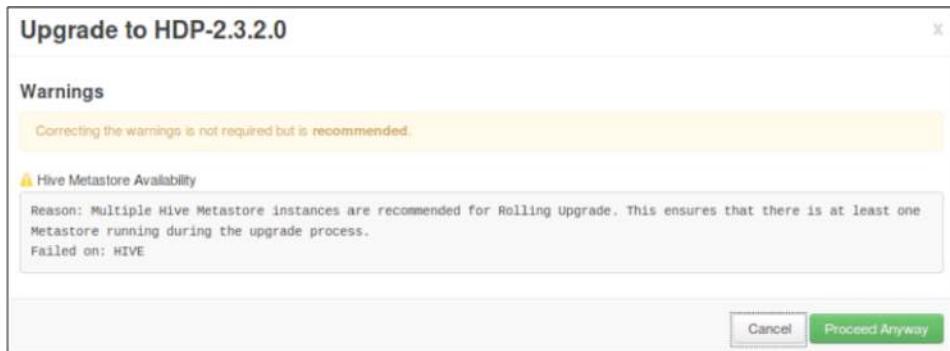


Selecting Perform Upgrade

Click **Perform Upgrade** to start a rolling upgrade.

Performing an HDP Rolling Upgrade

Warning Windows

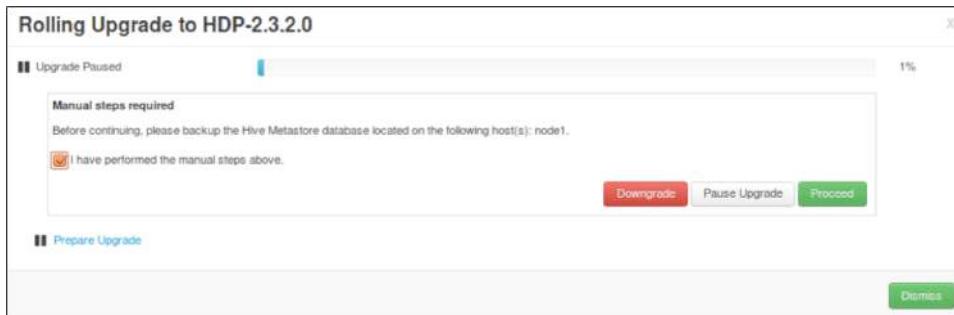


Warning Windows Related to Current Configurations

Depending on the installed services and their configurations, one or more warning windows might open. For example, a warning window will open if Hive is installed but is not configured with two instances of the Hive Metastore. The window recommends configuring a second Hive Metastore database prior to performing the rolling upgrade. Having two instances of the Hive Metastore will limit Hive downtime during the upgrade because Hive clients can use one instance of the Metastore while the other instance is being upgraded. You are offered the choice to **Cancel** the upgrade or **Proceed Anyway**.

If Ambari determines there are no conditions that require warnings, then it presents only a confirmation window. In this case click **OK** in the confirmation window (not shown).

Confirmation



Confirming that Manual Steps have been Completed

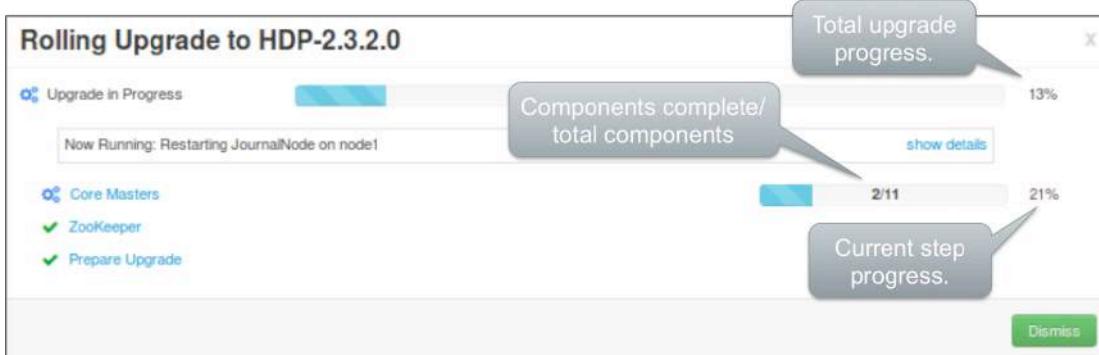
Ambari starts the rolling upgrade process by prompting for confirmation that the manual preparation steps have been completed by the administrator. In this screen capture, Ambari has detected that Hive is installed and prompts the administrator to confirm that the Hive Metastore database has been backed up. If Oozie and Ranger are installed then Ambari will prompt the administrator to confirm that their databases have been backed up as well.

If the administrator has not backed up a database, they can click **Pause Upgrade** and perform the backup. The upgrade can be continued once the backup is complete. The upgrade can be continued from the **Admin > Stack and Versions > Versions** tab in the Ambari Web UI.

Assuming the backup has already been completed, click the confirmation check box and then click **Proceed** to continue the rolling upgrade.

If Ambari determines there are no conditions that require warnings, then it presents only a confirmation window. In this case click **OK** in the confirmation window (not shown).

Upgrading ZooKeeper, Ranger, and Core Masters



Upgrading ZooKeeper

Ambari starts the rolling upgrade by upgrading the ZooKeeper servers. While upgrading the ZooKeeper instances one by one, Ambari must make sure that the quorum is maintained. Ambari orchestrates the upgrades in such a way that the next ZooKeeper server is only marked for upgrade after the previous one has successfully finished and has returned to service. This guarantees that the ZooKeeper quorum is always up, running, and consistent during a rolling upgrade.

Then the Ranger Admin and Ranger UserSync servers are upgraded and restarted. There is minimal service disruption as existing services continue to run. In the example, Ranger was not installed so it was not upgraded.

Then Ambari upgrades the core master service components for HDFS, YARN, and HBase. These services remain available assuming that the NameNode, ResourceManager, and HBase Master are configured for high availability. The standby server components are stopped, upgraded, and restarted first. These newly upgraded servers then become the active servers, enabling the formerly active servers to be stopped, upgraded, and restarted. Clients continue to operate normally during this time.

Running Available Service Checks



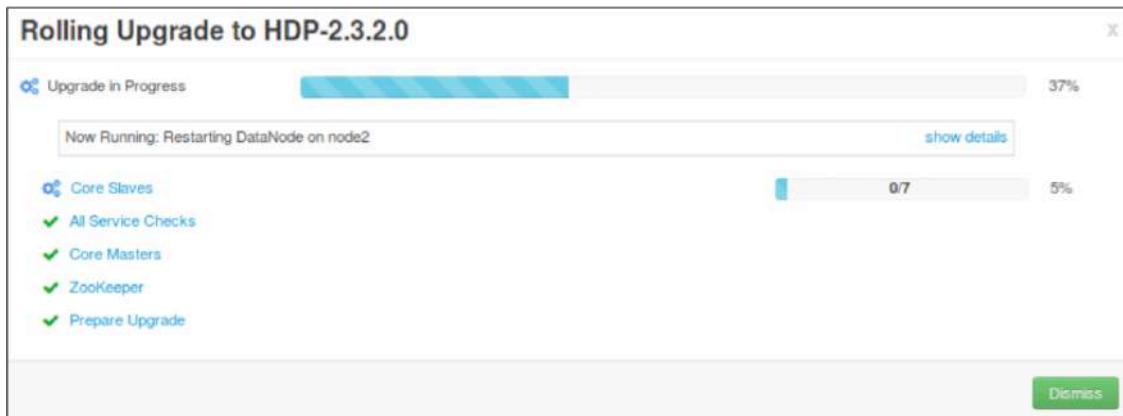
Running Available Service Checks

Ambari tests the functionality of the newly upgraded ZooKeeper, Ranger, and core master servers by running any available service checks for any installed services. The more services that are installed, the longer this step takes during the rolling upgrade. If any tests fail then Ambari presents the choice to perform a downgrade.

Performing an HDP Rolling Upgrade

Clicking the **show details** link expands the display window to show more detail about the current upgrade task. Each task upgrades a component on one of the cluster machines. The window commonly displays the contents of a `/var/lib/ambari-agent/output-<nnn>.txt` file on the node currently being upgraded. In the event of an error, the display would also show the contents of a `/var/lib/ambari-agent/errors-<nnn>.txt` file.

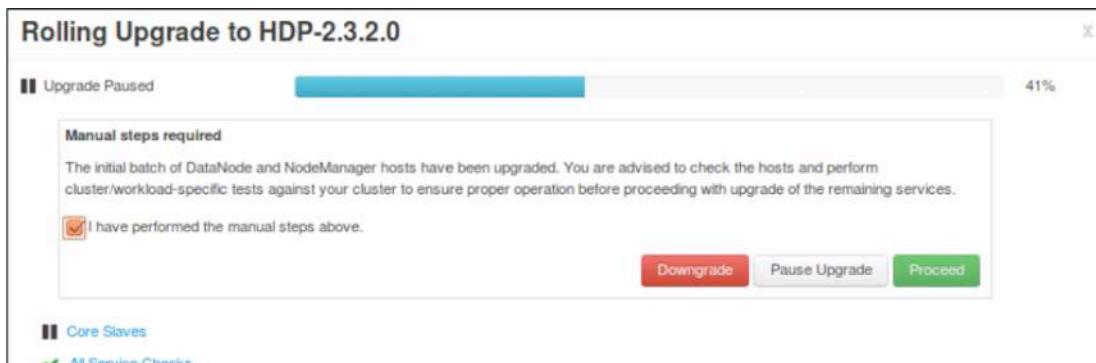
Upgrading Core Slaves



Upgrading Core Slaves

Ambari then begins to upgrade 20% of the core slave machines. The core slave machines include any machines running a NodeManager, DataNode, or RegionServer. Upgrading only the first 20% and then pausing enables an administrator to test the upgrade before rolling upgrade upgrades the remaining bulk of the core slaves. If testing does not yield satisfactory results, the administrator can abort the upgrade and downgrade the cluster.

Optional Functional Tests



Optional Functional Tests

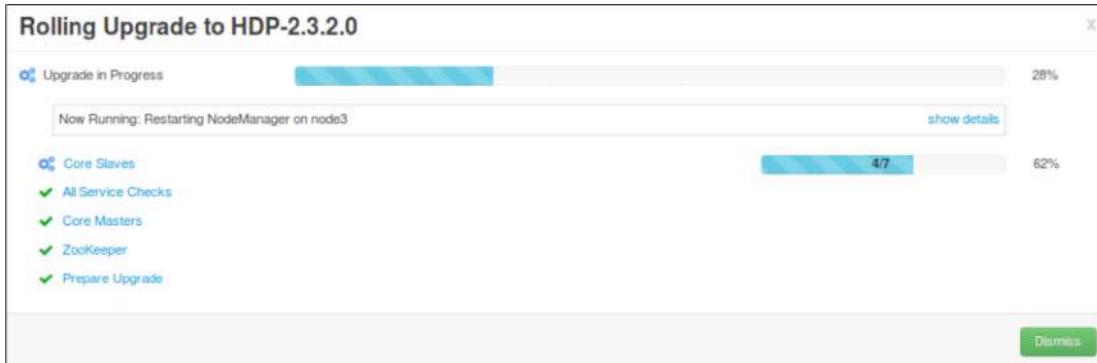
To save time in the event that a functional or performance problem might be found after the upgrade, Ambari initially only upgrades 20 percent of the core slave machines. Once these machines are upgraded, Ambari pauses the upgrade to allow an administrator to perform any optional functional or performance tests.

Once the administrator is satisfied that the new HDP version is functional across the first 20 percent of the core slave machines, the administrator can allow the rolling upgrade to be continued. To continue the rolling upgrade click the confirmation check box and then click **Proceed**.

Performing an HDP Rolling Upgrade

Be aware that both the upgraded and non-upgraded clients can use the upgraded master servers during this time so there is minimal interruption to cluster operation.

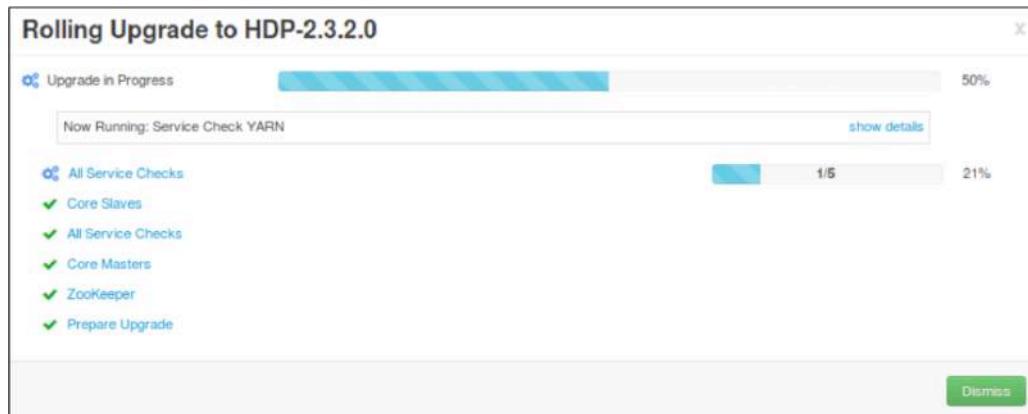
Completing Slave Upgrades



Upgrading Remaining 80% of Core Slave Machines

Ambari then upgrades the remaining 80 percent of the core slave machines. Be aware that a large clusters with dozens or hundreds of nodes could takes hours or even days to complete the upgrade process.

Running Available Service Checks Again



Installed Services Checks

Ambari tests the functionality of the newly upgraded core services by running any available service checks for any installed services. The more services that are installed, the longer this step takes during the rolling upgrade. It can take anywhere from a few minutes to nearly a half an hour.

Performing an HDP Rolling Upgrade

Hive Master Servers



Updating Hive Master Servers

Next Ambari updates the Hive master server components. These include the Hive Metastore, HiveServer2, and WebHCat servers. These services remain available assuming that the Hive Metastore and HiveServer2 have been configured for high availability. Only one component at a time in the high availability configuration is stopped, upgraded, and restarted.

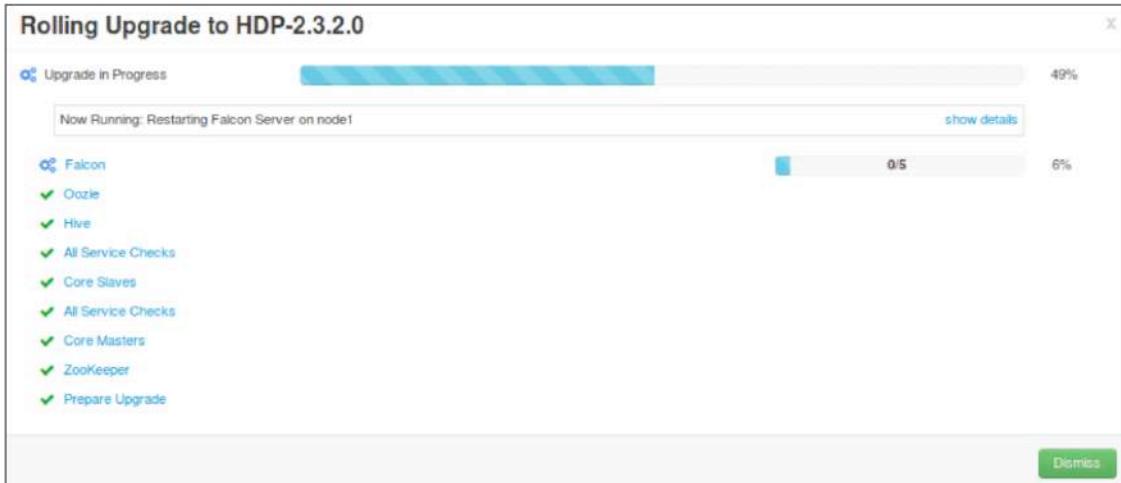
Switching the HiveServer2 Port Number



Switching HiveServer2 Port Numbers

Ambari warns the administrator that it will switch the HiveServer2 port from 10000 to 10010 during the upgrade. Click the check box to acknowledge the notification and then click **Proceed** to continue upgrading Hive.

Upgrading Other Installed Services



Upgrading Other Installed Services

At this point other services' server components are upgraded. Which ones are upgraded depends on which ones are installed. The server components that could be upgraded at this point include Spark, Oozie, and Falcon.

Spark

Spark must be stopped to perform the upgrade, which results in service disruption.

Oozie

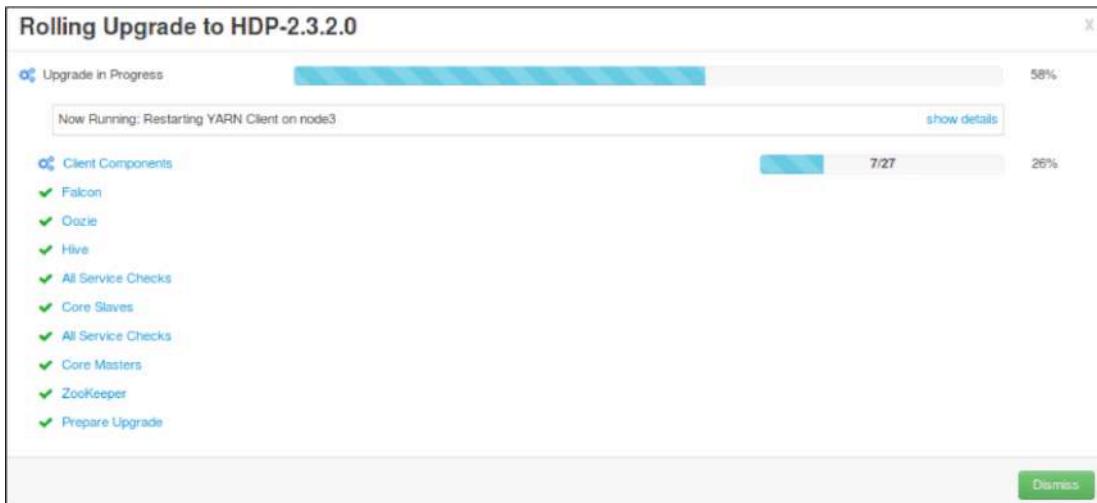
Oozie is built using a fast start architecture. In a fast start architecture, servers are expected to be restartable after failures. Oozie stores state information in its database and that state is read at server start time. Rolling upgrade simply restarts all the servers at once after they are upgraded. Each server reads state from the database and the service resumes from the point where it stopped. The result is very minimal downtime during the service restart.

Falcon

Falcon must be stopped to perform the upgrade, which results in service disruption.

Performing an HDP Rolling Upgrade

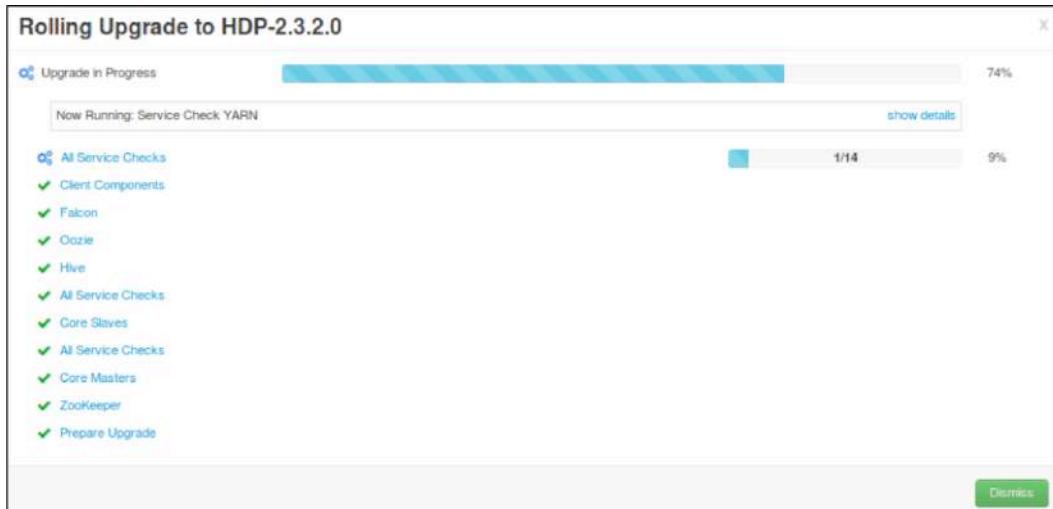
Upgrading the Hadoop Ecosystem



Upgrading the Hadoop Ecosystem

The Hadoop ecosystem clients are upgraded at this time. Depending on what is installed, this can include HDFS, YARN, MapReduce2, ZooKeeper, Tez, Hive, Pig, Sqoop, Hcat, and other clients. The length of time to performing this step will depend on the types of clients installed and the number of nodes in the cluster.

Running Available Service Checks Again

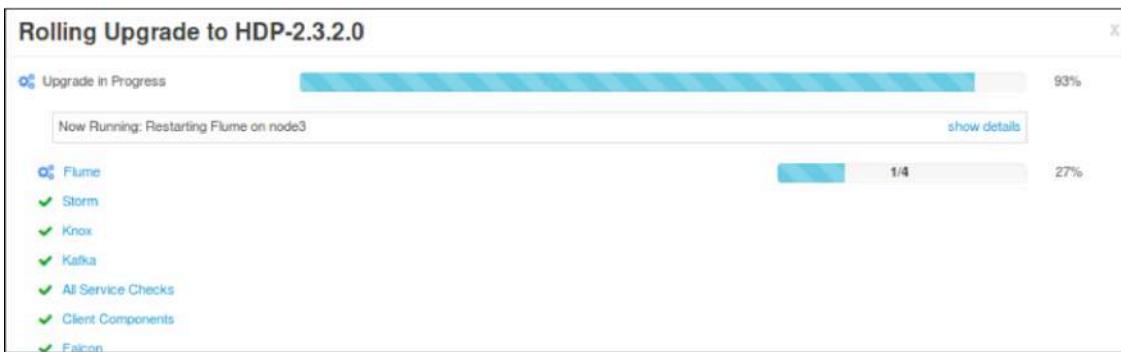


Service Checks for Cluster Services

Ambari tests the functionality of the upgraded cluster services by running any available service checks for any installed services. The more services that are installed, the longer this step takes during the rolling upgrade. It can take anywhere from a few minutes to nearly a half an hour.

Performing an HDP Rolling Upgrade

Remaining Installed Services



Checking Remaining Installed Services

Any remaining installed services are upgraded at this point. Which services are upgraded depends on which services are installed. Services that could be upgraded at this point include Kafka Brokers, Knox Gateways, Slider, Storm components, and Flume components.

Kafka Brokers, Knox gateways, and Storm components must be stopped for the upgrade process, which results in service disruption.

A Slider client can continue to run an existing application during Slider upgrade. However, when a Slider application is updated it must be stopped, which results in service disruption.

Flume agents are stopped and upgraded one at a time, which minimizes service disruption.

Final Upgrade Activities

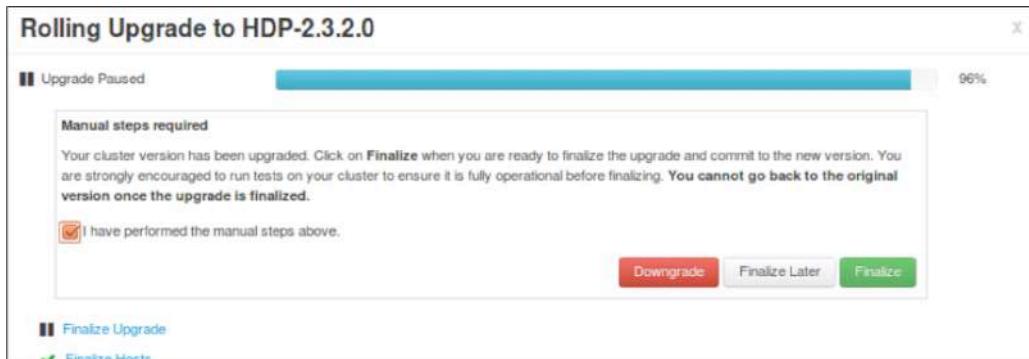


Final Upgrade Activities

Ambari performs any final upgrade activities.

Performing an HDP Rolling Upgrade

Finalizing the Upgrade

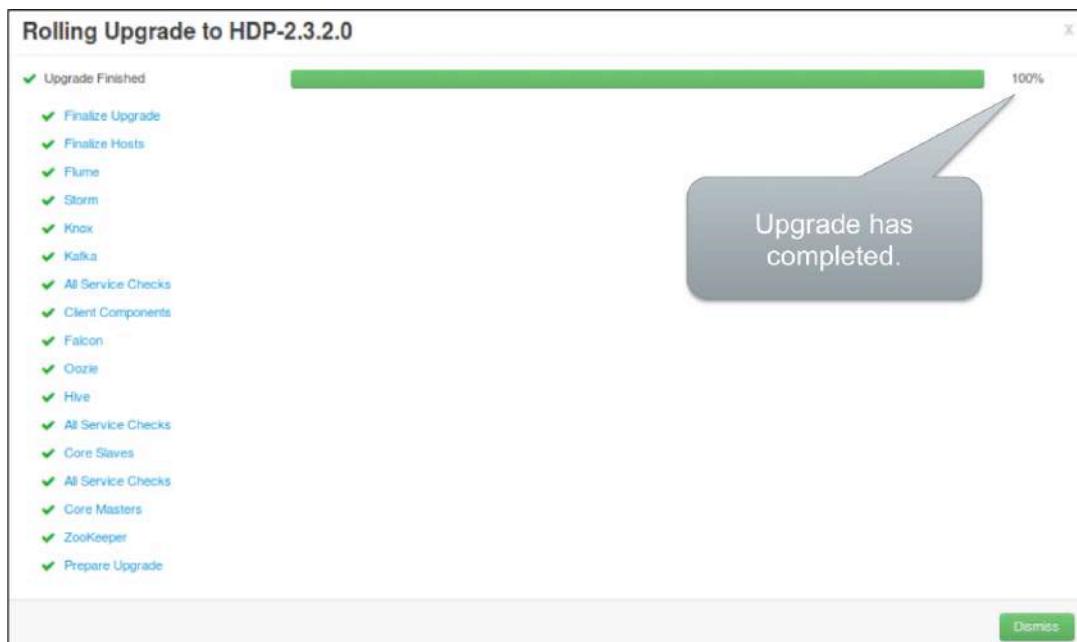


Manual Finalization of the Rolling Upgrade

The administrator must manually choose to finalize the upgrade. Once the upgrade has been finalized, a downgrade is no longer possible.

The administrator could choose **Finalize Later** and perform functional or performance tests for a period of time. Once the administrator is satisfied the upgrade software is working successfully, they can browse in the Ambari Web UI to **Admin > Stack and Versions > Versions** and choose to finalize the upgrade.

Closing the Upgrade



Closing the Rolling Upgrade

The final window shows the upgrade is 100 percent complete. Click **Dismiss** to close the window.

Older Version Removed



The screenshot shows the Cloudera Manager interface with the 'Stack' tab selected. Under the 'Versions' section, there is a single entry for 'HDP-2.3.2.0 (2.3.2.0-2950)'. A green button labeled 'Current' is highlighted. Below this, a summary of hosts is shown: 0 Not Installed, 0 Installed, and 3 Current. A blue 'Manage Versions' button is located in the top right corner. A tooltip at the bottom of the screen reads 'Final Version is No Longer Available'.

The final result is that the new HDP version is listed as the current version. The previous version of the HDP software has been removed from the window.

This page left blank intentionally.

Knowledge Check

Use the following questions and answers to assess your understanding of the concepts presented in this lesson.

Questions

- 1) The three types of upgrade include major, minor, and _____ upgrades.
- 2) Rolling upgrade is supported starting from HDP version _____ .
- 3) True or false? A downgrade operation removes any new software version, data, or metadata generated by a rolling upgrade.
- 4) True or false? Downgrade is still possible after finalizing a rolling upgrade.
- 5) True or false? You cannot perform a rolling upgrade without enabling cluster high availability features.
- 6) Registering a new HDP version makes it available for use by _____ .
- 7) Why does rolling upgrade only initially upgrade 20 percent of the core slaves?

Answers

1) The three types of upgrade include major, minor, and _____ upgrades.

Answer: Maintenance

2) Rolling upgrade is supported starting from HDP version _____ .

Answer: 2.2

3) True or false? A downgrade operation removes any new software version, data, or metadata generated by a rolling upgrade.

Answer: False. New data and metadata is retained.

4) True or false? Downgrade is still possible after finalizing a rolling upgrade.

Answer: False

5) True or false? You cannot perform a rolling upgrade without enabling cluster high availability features.

Answer: False. You can perform the upgrade but will experience more service disruption.

6) Registering a new HDP version makes it available for use by _____ .

Answer: Ambari

7) Why does rolling upgrade only initially upgrade 20 percent of the core slaves?

Answer: To potentially save time. There are typically a large number of worker nodes in a cluster. Enabling an administrator to test functionality on a small number of worker nodes before upgrading them all could save time in the event the cluster must be downgraded.

Summary

- Ambari uses the concept of a Stack to install software.
- Stacks have versions: HDP 2.0, HDP 2.1, HDP 2.2, HDP 2.3, and so on.
- Hortonworks recommends upgrading to the latest version of Ambari prior to upgrading HDP.
- HDP supports three types of upgrades: major, minor, and maintenance.
- HDP supports in-place and rolling upgrades.
- A rolling upgrade is a certified process where services are switched over to new version in rolling fashion.
- Rolling upgrade includes the capability to downgrade the server and worker components to the prior version.

Configuring Heterogeneous HDFS Storage

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Identify the purpose and operation of heterogeneous HDFS storage
- ✓ Identify and describe the HDFS storage types
- ✓ Identify and summarize the operation of HDFS storage policies
- ✓ Configure storage types and policies

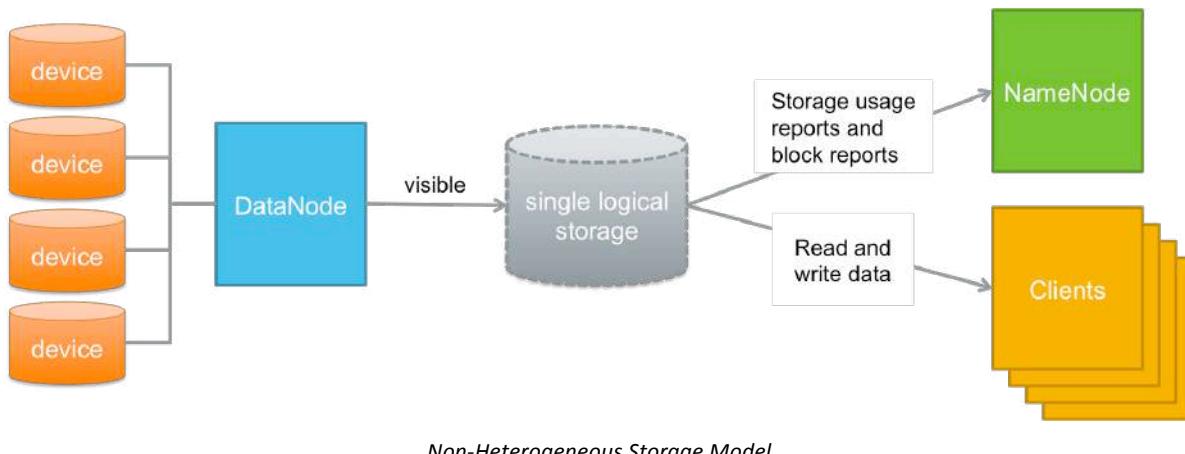
Heterogeneous HDFS Storage

Storage Type	Sequential Throughput	Random IOPS	Cost	Example Usage
HDD	High	Low	Low	Batch processing (MapReduce/Tez), data archiving
SSD	High	High	Medium	Interactive query (Hive), random I/O (HBase)
RAM disk	Very High	Very High	High	Interactive query (Hive), random I/O (HBase)

Heterogeneous HDFS storage is designed to satisfy different data processing and data storage requirements. As such, it has support for hard disk drives (HDD), solid-state drives (SSD), and RAM disks. These storage types vary by performance characteristics, cost per gigabyte, and application suitability. The table lists and compares the performance characteristics and costs for HDD, SSD, and RAM disks.

A RAM disk is volatile and data stored on one has no durability. For example, a sudden power loss or system failure would result in the loss of data in a RAM disk. For this reason, only data that is temporary or re-generatable should be stored in a RAM disk.

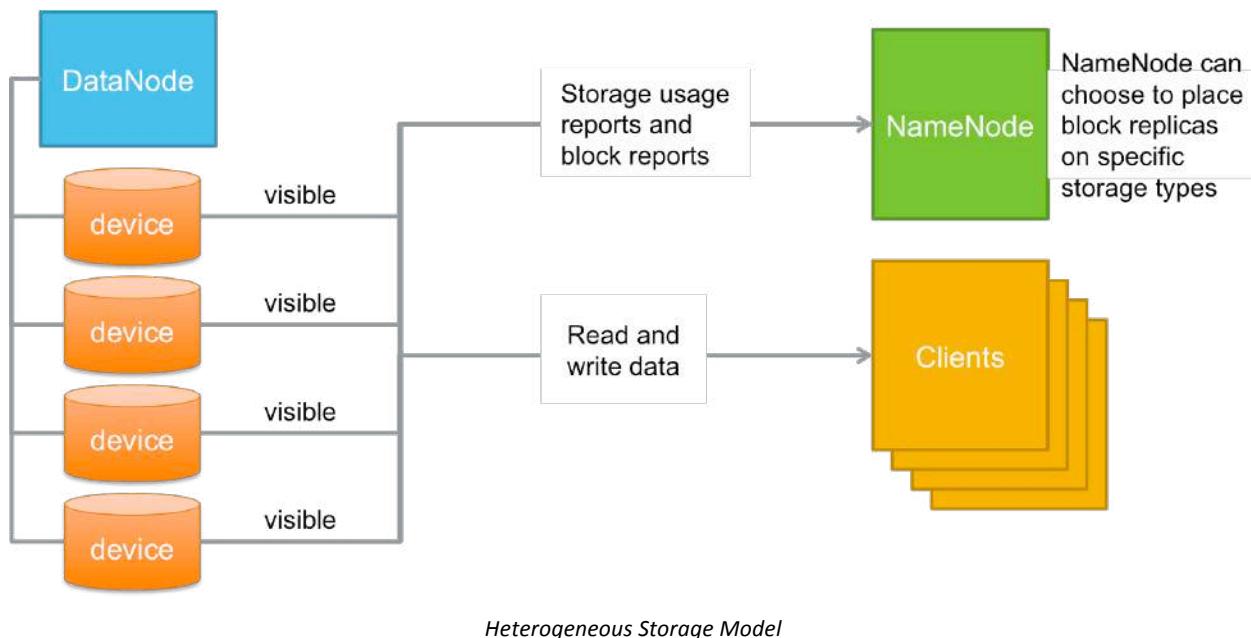
Prior HDFS Storage Model



The HDFS NameNode and HDFS clients have historically viewed each DataNode as a single, logical storage unit. The NameNode was not aware of the number or types of storage devices attached to a DataNode.

Each DataNode communicated its storage state to the NameNode using storage and block reports. A storage report contained summary capacity and usage information that was aggregated from all attached storage devices. Block reports contained lists of data blocks currently stored on the DataNode.

Current HDFS Storage Model



With heterogeneous storage, the HDFS storage model has been updated so that the DataNode exposes the types and usage information for each individual storage device to the NameNode. This enables the NameNode to choose not just a target DataNode when placing data block replicas, but also the specific storage type on each target DataNode.

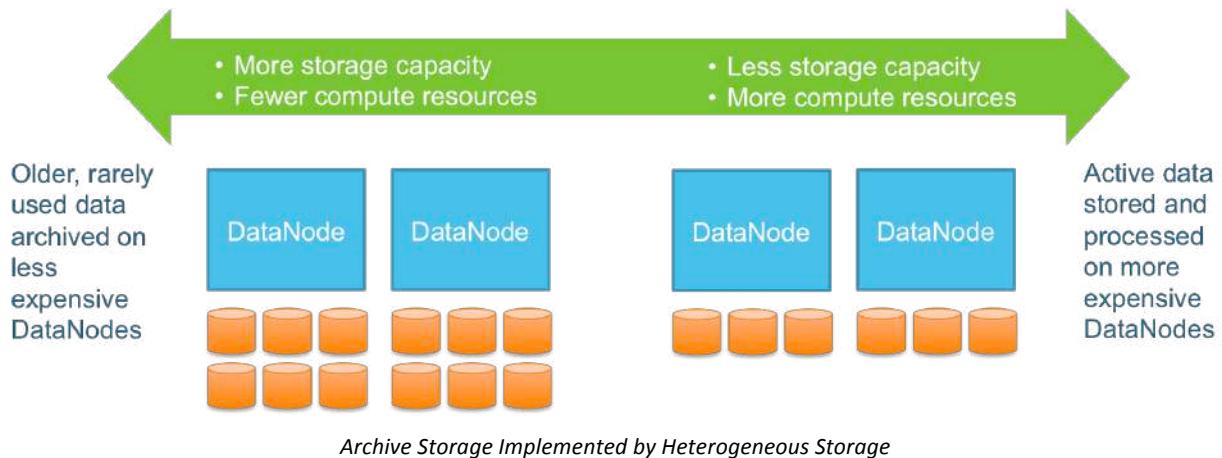
Storage Preferences

Heterogeneous storage uses storage preferences. A storage preference is a hint to HDFS specifying where data block replicas should be placed. It includes the number of replicas to write and the preferred storage type for each replica.

HDFS attempts to satisfy storage preferences based on the availability of storage space and the availability of storage quota. The HDFS quota feature has been updated so that HDFS administrators can limit not only the total amount of storage space consumed by a user or application, but can also now limit the amount of space a user or application can consume on a specific storage type. Quotas and quota administration is described in another course. If the preferred storage type is not available, a fallback type can be used. Fallback rules are described later in this lesson.

For example, a storage preference could be configured that requests all replicas to be placed on SSD storage. However, if there is no more SSD storage space, or the user's storage type quota would be exceeded, the replicas could be written to hard disk drives instead.

Archive Storage



Heterogeneous storage can also be used to implement HDFS archive storage. HDFS commonly stores both active and inactive data. Active data, sometimes called hot data, is regularly processed to produce business value. Inactive data, sometimes called cold data, is maintained in HDFS but rarely processed. Cold data might be maintained for months or years depending on such factors as regulatory requirements or business need. As the amount of cold data under storage grows, there is a benefit to optimizing HDFS for cost. Creating an archive storage tier consisting of lower-cost DataNodes that feature slower spinning, higher density disks, and less compute power increases cost efficiency.

Normal vs. Archive DataNode Configuration

	Normal DataNode	Archive DataNode
Storage capacity	40 TB	210 TB
Number of drives	12	60
Number of CPUs	32	4
Amount of memory	128 GB	64 GB
Cost per GB	X (will vary)	x/5 (will vary)
Runs YARN	yes	no
NodeManager		

DataNodes configured to regularly process data are configured differently from DataNodes configured to only store inactive data. The biggest difference is that the archive DataNodes do not run the YARN NodeManager and cannot process the data.

Note

Because an archive DataNode does not run the YARN NodeManager and cannot process data, any data that must be processed must be copied over the network to a Tez or MapReduce mapper running on a normal DataNode.

HDFS Storage Types

Storage Device Label	Description
DISK	Identifies a spinning hard disk drive on a DataNode with relatively more compute resources.
ARCHIVE	Identifies a spinning hard disk drive on a DataNode with relatively fewer compute resources but relatively more storage capacity.
SSD	Identifies a solid-state disk drive on a DataNode.
RAM_DISK	Identifies a DataNode with a portion of memory configured as a RAM disk.

An HDFS administrator uses storage device labels to identify each storage device on a DataNode. If the storage device is not specifically assigned a label, it is assumed to be the type DISK. The NameNode uses the label information to attempt to satisfy storage preferences when writing data block replicas to HDFS storage. How labels are assigned to a storage device is described later in this lesson.

The table lists the four possible HDFS storage device labels and provides a description of each one.

HDFS Storage Operations

HDFS heterogeneous storage introduces a new concept of storage policies. Storage policies direct file data block replicas to be stored on different storage types.

Storage Policies

Storage Policy Name and ID Number	Description
Hot (7)	Used for file data block replicas that must be both stored and frequently processed on a DataNode. All replicas are stored on type DISK.
Cold (2)	Used for file data block replicas that should be archived on a DataNode. All replicas are stored on type ARCHIVE.
Warm (5)	Used for file data block replicas that are partially hot and cold. One replica is stored in DISK, all others are stored on type ARCHIVE.
All_SSD (12)	Used for file data blocks where all replicas must be stored on type SSD.
One_SSD (10)	Stores one replica on type SSD, and all other replicas on type DISK.
Lazy_Persist (15)	Store a single data block replica on type RAM_DISK.

A storage policy is assigned by an administrator to a directory or a specific file. How to assign storage policies is described later in this lesson.

The table lists the six storage policies and provides a description of each.

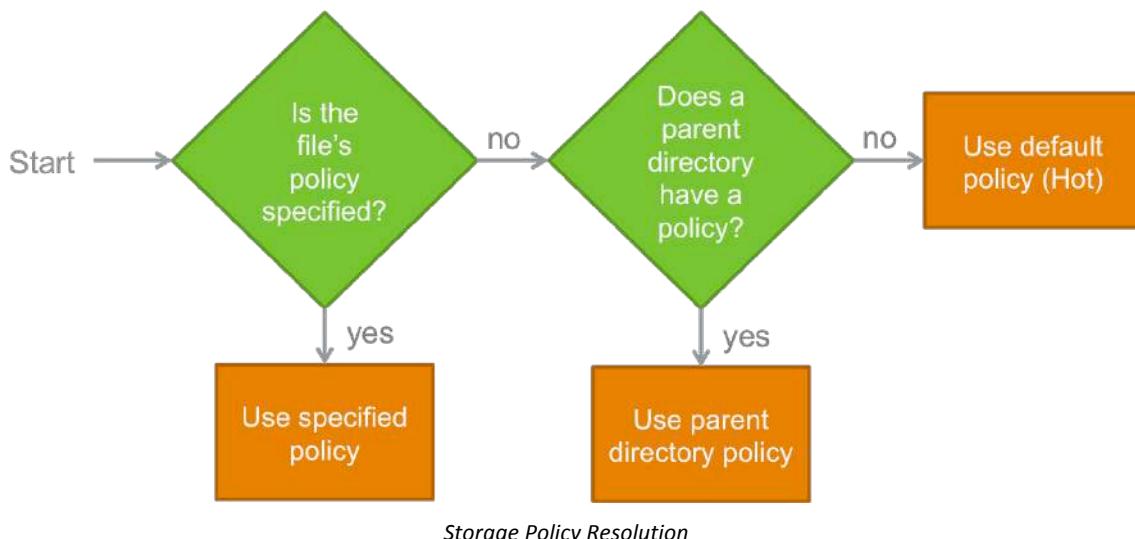
Data Block Replica Placement Rules

Policy Name	First Choice Block Placement (with n replicas)	Fallback for file creation (first replica)	Fallback for replication (remaining replicas)
HOT	DISK: all n	n/a	ARCHIVE
COLD	ARCHIVE: all n	n/a	n/a
WARM	DISK: first n , ARCHIVE: $n-1$	ARCHIVE, then DISK	ARCHIVE, then DISK
ALL_SSD	SSD: all n	DISK	DISK
ONE_SSD	SSD: first n , DISK: $n-1$	SSD, then DISK	SSD, then DISK
LAZY_PERSIST	RAM_DISK: first n , DISK: $n-1$	DISK	DISK

When a storage type has sufficient space or a storage type quota is not violated, data block replicas are placed according to the first choice block placement column in the table. If a storage type has insufficient space or a storage type quota would be violated, the fallback locations specified in the last two columns of the table are used.

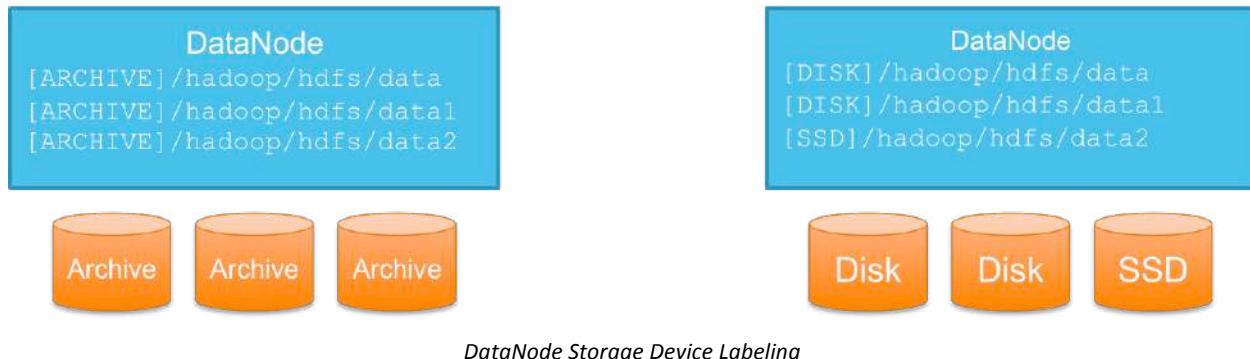
The Lazy_Persist policy is only useful for file data blocks that have only a single replica. Multiple replicas do not improve performance because all of the remaining replicas would be *lazily* written to DISK.

Storage Policy Resolution



An administrator can assign a storage policy to a directory or a specific file. Files and directories can inherit a policy from a parent directory. If a file's policy is not specified, the default policy is Hot. The flowchart describes how HDFS resolves the storage policy for a given file.

Labeling DataNode Storage Devices



By default, any unlabeled DataNode storage device is interpreted as DISK. Storage devices are explicitly labeled by updating the `dfs.datanode.data.dir` property in the `hdfs-site.xml` file. This property contains a comma-separated list of directory paths used to access storage devices. Preface each path with a storage type.

The illustration provides examples of labeled storage devices.

Assuming that the cluster is managed using Ambari, use the Ambari Web UI or Ambari API to update the `dfs.datanode.data.dir` property on each DataNode. If different groups of DataNodes will have different storage device labels, you will need to use the Ambari Configuration Groups feature to accommodate these configuration differences.

Configuring Storage Types and Policies

Enabling HDFS Storage Policies

Support for HDFS storage policies is enabled by default in the `hdfs-site.xml` file. By default, the value of the `dfs.datanode.policy.enabled` is set to true.

Existing policies cannot be edited. New policies cannot be created.

Configuring a Storage Policy

Storage policies are configured either per directory or per file.

To list storage policy types, type the command `hdfs storagepolicies -listPolicies`. Any user can use this command.

The remaining commands must be entered by the HDFS superuser. For example, the `hdfs` user is commonly the HDFS superuser.

The syntax to assign a storage policy to a file or directory is:

```
hdfs storagepolicies -setStoragePolicy -path <path> -policy <policy_name>
```

For example, to configure the directory `/user/root/web` with a WARM policy, type the command:

```
hdfs storagepolicies -setStoragePolicy -path /user/root/web -policy WARM
```

The syntax to view an assigned policy for a file or directory is:

```
hdfs storagepolicies -getStoragePolicy -path <path>
```

For example, to view the policy assigned to the directory /user/root/web, type the command:

```
hdfs storagepolicies -getStoragePolicy -path /user/root/web
```

It is important to note that existing file data block replicas are not automatically moved just because they are assigned to a new storage policy that requires a different storage type. The file data block replicas must be moved. How to accomplish this is described later in this lesson.

HDFS Mover

Options	Description
-p <file dir>	Specify a space separated list of HDFS files or directories to migrate.
-f <local_file>	Specify a local file containing a list of HDFS files or directories to scan.

The `HDFS mover` is a newer utility. It scans data block replicas for compliance with storage policy placement rules. As necessary, it moves data block replicas to a new type a storage. The mover should be run after assigning a new policy to an existing file or an existing directory with existing files.

The `mover` not only scans and moves replicas appropriately; it also maintains HDFS replication and rack awareness requirements.

The `mover` has two options that are listed and described in the table. If a local file is created with a list of HDFS files and directories to scan, the file should be formatted with a single file or directory on each line.

For example, to have the `mover` scan the `/apps/weblog` directory and the `/hive/04_2015` file, type the command:

```
hdfs mover -p /apps/weblog /hive/04_2015
```

To have the `mover` scan the list of files and directories contained in the file `/home/steve/filelist`, type the command:

```
hdfs mover -f /home/steve/filelist
```

Viewing Block Placement Information

```
[hdfs@node1 ~]$ hdfs fsck /user/root/testpolicy -files -blocks -locations
Connecting to namenode via http://node1:50070/fsck?ugi=hdfs&files=1&blocks=1&locations=1&path=%2Fuser%2Froot%2Ftestpolicy
FSCK started by hdfs (auth:SIMPLE) from /172.17.0.2 for path /user/root/testpolicy at Wed Oct 28 12:48:09 EDT 2015
/user/root/testpolicy <dir>
/user/root/testpolicy/hosts 173 bytes, 1 block(s):  OK
0. BP-1492469471-172.17.0.2-1445390772022:blk_1073742412_1588 len=173 repl=3 [DatanodeInfoWithStorage[172.17.0.2:50010,DS-425a3a6a-675d-4f3b-a089-a54f3d2534d3,ARCHIVE], DatanodeInfoWithStorage[172.17.0.3:50010,DS-a07710ef-d2af-47ca-9fc6-beec812b55fc,ARCHIVE], DatanodeInfoWithStorage[172.17.0.4:50010,DS-d2bdc5f6-7fa3-4db0-94e1-e85efadff96b,ARCHIVE]]]

/user/root/testpolicy/passwd 1560 bytes, 1 block(s):  OK
0. BP-1492469471-172.17.0.2-1445390772022:blk_1073742411_1587 len=1560 repl=3 [DatanodeInfoWithStorage[172.17.0.3:50010,DS-a07710ef-d2af-47ca-9fc6-beec812b55fc,ARCHIVE], DatanodeInfoWithStorage[172.17.0.4:50010,DS-d2bdc5f6-7fa3-4db0-94e1-e85efadff96b,ARCHIVE], DatanodeInfoWithStorage[172.17.0.2:50010,DS-425a3a6a-675d-4f3b-a089-a54f3d2534d3,ARCHIVE]]]

Status: HEALTHY
Total size: 1733 B
```

The HDFS `fsck` command displays the storage type for each block replica if the `-files -blocks -locations` command options are used. The screen capture shows two small, single block files, each with three data block replicas. The replicas are stored on disk drives with a storage type of ARCHIVE.

Knowledge Check

Knowledge checks can be used by students as a self-assessment tool.

Questions

- 1) Heterogeneous HDFS storage is designed to enhance performance, increase cost efficiency, and support diverse _____ .
- 2) HDFS attempts to satisfy storage preferences based on the availability of storage space and _____ .
- 3) True or false? If the preferred storage type is unavailable, a file write always fails.
- 4) Name one of the primary differences between a normal DataNode and an archive DataNode.
- 5) Describe the first choice data block replica placement guidelines for the WARM policy.
- 6) True or false? Files can inherit their policy from a parent directory.
- 7) Which `hdfs-site.xml` property is used to label DataNode storage devices?
- 8) Which command scans data block replicas for compliance with current storage policy placement rules.

Answers

- 1) Heterogeneous HDFS storage is designed to enhance performance, increase cost efficiency, and support diverse _____ .

Answer: Workloads

- 2) HDFS attempts to satisfy storage preferences based on the availability of storage space and _____ .

Answer: Storage quota

- 3) True or false? If the preferred storage type is unavailable, a file write always fails.

Answer: False

- 4) Name one of the primary differences between a normal DataNode and an archive DataNode.

Answer: An active DataNode commonly has more storage capacity/density or fewer compute resources

- 5) Describe the first choice data block replica placement guidelines for the WARM policy.

Answer: Place the first replica on DISK, and the remaining replicas on ARCHIVE

- 6) True or false? Files can inherit their policy from a parent directory.

Answer: True

- 7) Which `hdfs-site.xml` property is used to label DataNode storage devices?

Answer: `dfs.datanode.data.dir`

- 8) Which command scans data block replicas for compliance with current storage policy placement rules.

Answer: `hdfs mover` **command**

Summary

- Heterogeneous HDFS storage is designed to enhance performance, increase cost efficiency, and support diverse workloads.
- Heterogeneous storage uses *storage preferences*, which provide a hint to HDFS specifying the type of storage where data block replicas should be placed.
- An HDFS administrator uses storage type labels to identify DataNode storage devices.
- Storage policies direct file data block replicas to be stored in different HDFS storage types.
- An administrator can assign a storage policy to a directory or a specific file.
- The `hdfs storagepolicies` command assigns policies to a directory or file.
- The `hdfs mover` command scans and moves data block replicas for compliance with current storage policy placement rules.

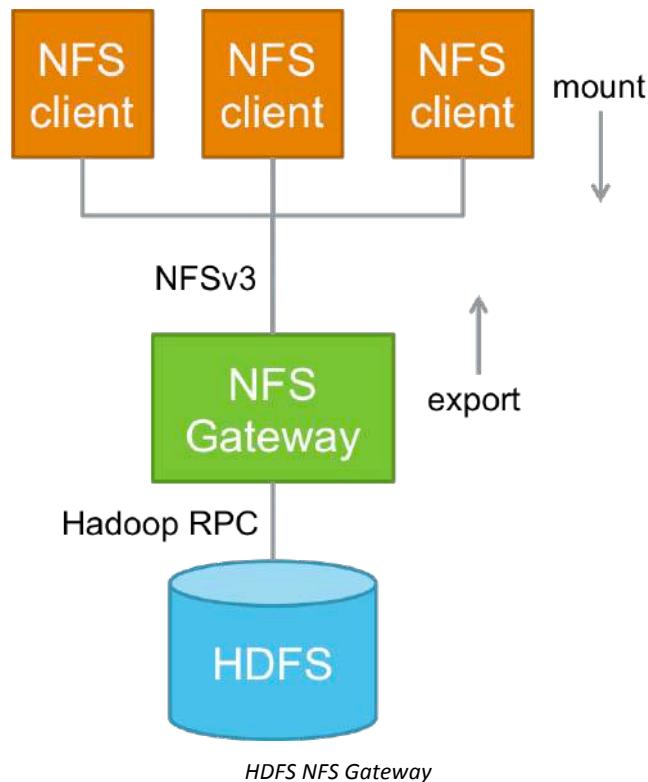
Managing the HDFS NFS Gateway

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Identify HDFS NFS Gateway use cases
- ✓ Recall HDFS NFS Gateway architecture and operation
- ✓ Install and configure an HDFS NFS Gateway
- ✓ Configure an HDFS NFS Gateway client
- ✓ Start and Stop the HDFS NFS Gateway Service

HDFS NFS Gateway Use Cases



The HDFS NFS Gateway provides another method to access HDFS. NFSv3 clients mount the exported HDFS root directory from the NFS Gateway. NFS clients use the NFS protocol version 3 over TCP to communicate with NFS Gateway. The NFS Gateway uses Hadoop RPC to communicate with HDFS.

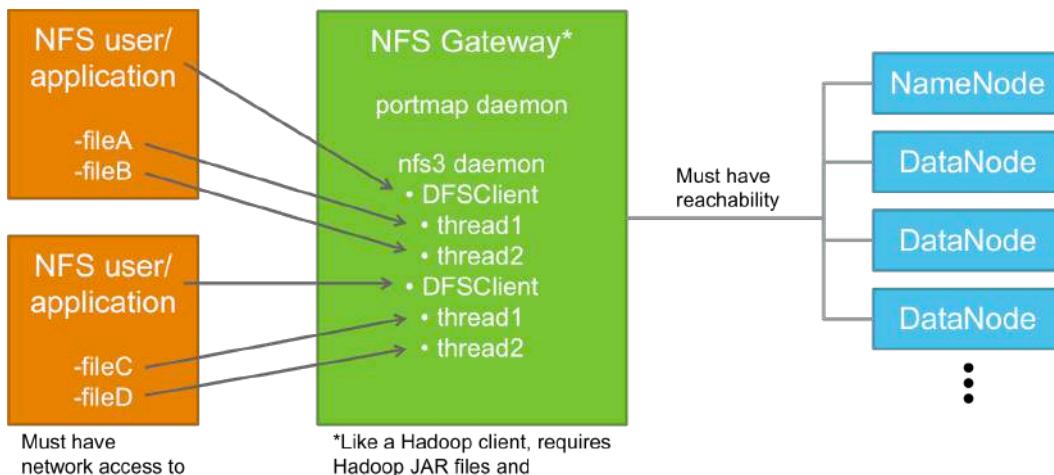
The NFS Gateway supports and enables the following use cases:

- Browse the HDFS file system through the local file system on NFSv3 client compatible operating systems. Linux users, for example, can use commands like `ls`, `cd`, `cat`, or `more` to move through HDFS directories and view files.
- Download files from the HDFS file system onto their local file system. Linux users, for example, can use commands like `cp` or `mv` to download files.
- Upload files from the local file system directly to the HDFS file system. Linux users, for example, can use commands like `cp` or `mv` to upload files.
- Stream data directly to HDFS through the mount point. File append is supported but random write is not supported. Linux users, for example, can use a shell redirection operator like `>>` to append new data to existing files.

The NFS Gateway is a very easy way to browse, download, upload, or stream data to HDFS. However, the NFS Gateway is not the fastest method to work with HDFS. For higher performance, the use of Java RPC or WebHDFS is recommended. As with any data transfer system, benchmarking is recommended to reveal actual performance data.

All read, write, and list operations are controlled by UNIX-style user privileges. The user performing the operations on the files or directories, as seen through the NFS mount point, must have privileges to do so, or the operation will report an error.

HDFS NFS Gateway Architecture and Operations

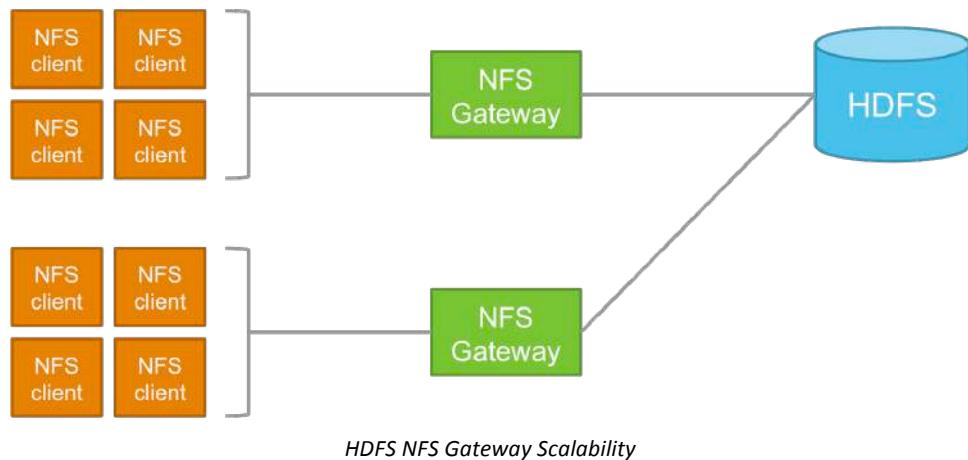


The NFS Gateway uses a portmap and nfs3 daemon. The nfs3 daemon performs the functions of the normal NFS mountd and nfsd daemons. The nfs3 daemon also implements the DFSClient. The DFSClient implements the Hadoop RPC connection to the HDFS NameNode and DataNodes. The NFS Gateway requires that the NameNode and DataNodes be reachable.

A DFSClient is created for each NFS client user/application. A DFSClient uses multiple threads to access multiple files on behalf of an NFS client user/application.

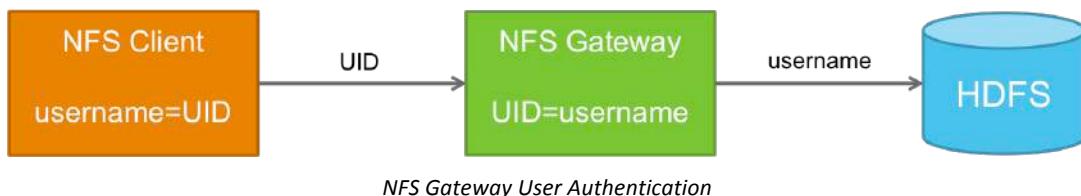
Like a normal Hadoop client machine, the NFS Gateway requires the Hadoop JAR files and the client configuration files. The Hadoop JAR files install software like the `yarn` and `hdfs` commands and their required class libraries. These will be installed by Ambari when configuring an NFS Gateway. The NFS Gateway can optionally run on a NameNode, DataNode, or Hadoop client machine.

NFS Gateway Scalability



Each NFS Gateway has a finite amount of network, CPU, and memory resources. As the number of active NFS clients increases, an NFS Gateway could become a performance bottleneck. To increase scalability for NFS clients, add multiple NFS Gateways on additional hosts. However, NFS client mounts are static do not failover, and cannot be configured to failover, between NFS Gateways during failures. Clients configured to use a failed NFS Gateway would have to be manually reconfigured to use another NFS Gateway.

User Authentication



When a user logs in to a host, their username is passed to NFS. If a user switches to another user after logging in, the new username is passed to NFS. In either case, the NFS client resolves the username to a UID using either a local `/etc/passwd` file or an LDAP database.

The user's UID is sent to the NFS Gateway when the user, or user's application, attempts to access HDFS. The NFS Gateway must resolve the UID to a username. The NFS Gateway can use either the local `/etc/passwd` file or an LDAP database.

The NFS Gateway sends the resolved username to the HDFS NameNode. The NameNode performs an HDFS permissions/ACLs check to determine if the user is granted read or write access, or no access, to the requested file or directory.

It is important that the UID on the NFS Client and NFS Gateway and the username on the NFS client, NFS Gateway, and in HDFS, are all consistent. To ensure the easy creation and management of consistent username-to-UID mappings across the enterprise, Hortonworks recommends the use of LDAP or Microsoft Active Directory. If usernames and UID are not consistent across all nodes, then the NFS Gateway can be configured with a `/etc/nfs.map` file. This file is used on the NFS Gateway to map NFS client UIDs to NFS Gateway UIDs. For more information about this file, refer to the NFS Gateway documentation at <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsNfsGateway.html>.

Proxy User



NFS Gateway Proxy User

The HDFS NameNode uses two properties defined in the `/etc/hadoop/conf/core-site.xml` file to determine which hosts and users may run an NFS Gateway, and which users may access HDFS through an NFS Gateway.

If Ambari is used to install the NFS Gateway, then the NFS service runs as the user `hdfs`. The `hadoop.proxyuser.hdfs.hosts` property determines which hosts may run an NFS Gateway as the user `hdfs`. By default, Ambari configures this property with an asterisk, which means that any host can run an NFS Gateway as the user `hdfs`. To restrict which hosts may run an NFS Gateway as the user `hdfs`, modify the property to have a comma-separated list of hostnames.

The `hadoop.proxyuser.hdfs.groups` property determines which users may be proxied by the user `hdfs`. These users may submit NFS access requests to an NFS Gateway. HDFS permissions and ACLS and other access controls (described in this lesson) still apply. By default, Ambari configures this property with an asterisk, which means that requests from any user may be proxied by the `hdfs` user. To restrict which users may be proxied by the user `hdfs`, modify this property to have a comma-separated list of group names. If a list of group names is used, rather than an asterisk, then it is required to include the group root along with any other desired groups.

The groups that a user belongs to are resolved by the Java class listed in the `hadoop.security.group.mapping` property in either the `core-default.xml` or `core-site.xml` file. By default the class is

`org.apache.hadoop.security.JniBasedUnixGroupsMappingWithFallback`. This class automatically determines if the Java Native Interface (JNI) is available and if so, will use the API within Hadoop to determine user-group memberships. The JNI should be available by default. If for some reason the JNI is not available, then the

`org.apache.hadoop.security.ShellBasedUnixGroupsMapping` class is used to invoke a Linux shell to determine user-group memberships.

By default user-group memberships are determined by looking at the NameNode's Linux configuration. The `/etc/group` file is examined to see to which groups a user belongs. If all group memberships are solely defined in LDAP, then the `hadoop.security.group.mapping` property could be updated to specify the use of the `org.apache.hadoop.security.LdapGroupsMapping` class. With this configuration only LDAP would be consulted to resolve user-group memberships.

Use the Ambari Web UI **Services > HDFS > Configs > Advanced > Custom core-site** to change these properties.

Reference

The NFS Gateway can be configured to use Kerberos authentication. Kerberos is not described in this course.

For more information, refer to the **NFS Gateway documentation at:**

<https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsNfsGateway.html>.

Access Controls

By default, the HDFS root directory export can be mounted by any NFS client. For more security, an optional access control list can be created that controls which hosts can mount the HDFS export. The access control list not only controls which hosts may mount the HDFS export, but whether they can mount it read-write or read-only. This access control list is examined before the HDFS permissions and ACLs, and might limit a user's write access to an HDFS file. For example, if the access control list only grants read-only mounts, then a user with HDFS write permissions on a file would not be able to write to the file.

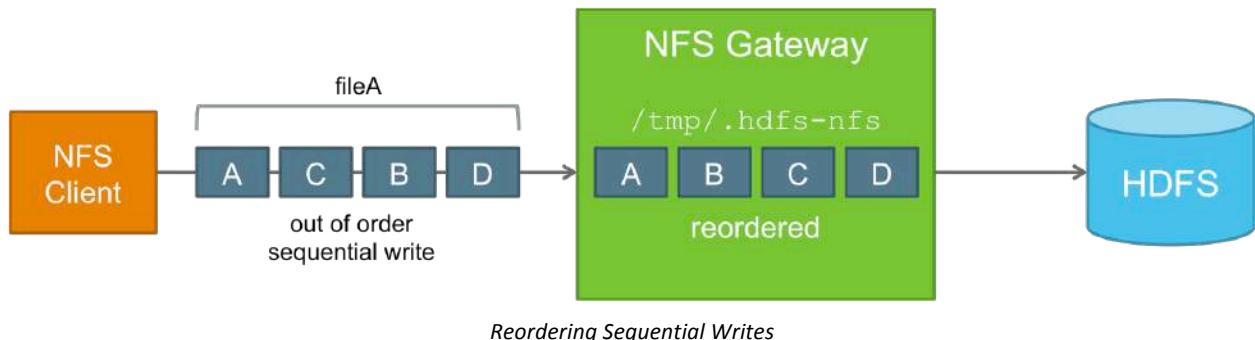
The access control list can be modified in the Ambari Web UI by browsing to **Services > HDFS > Configs > Advanced > NFS Gateway** and modifying the **Allowed hosts** property. This property is found in the `/etc/hadoop/conf/hdfs-site.xml` file and is actually named `nfs.exports.allowed.hosts`.



The property may consist of a semi-colon separated list of hostnames and NFS permissions. Possible permission values are `rw` or `ro`. If a permission is not specified, the default is `ro`.

Allowable hostnames include alpha-numeric host names, IP addresses, and Java regular expressions. An example is `myhost.org ro;192.168.5.13 rw; \w*\.\example\.\com`. Any host matching the last expression would get read-only permissions.

Reordering Sequential Writes



An NFS client's writes often can arrive out of order, especially when the export is not mounted using the `sync` option. This means that sequential writes can arrive at the NFS Gateway in random order.

The `nfs.dump.dir` property in the `/etc/hadoop/conf/hdfs-site.xml` file on the NFS Gateway specifies the directory where file data is temporarily stored and reordered prior to sending the file data to HDFS. Data in this dump directory is automatically removed once it has been sent to HDFS.

The administrator should ensure that the dump directory is on a storage volume with sufficient storage space for the expected workload. For example, if the NFS client might concurrently upload ten 100 MB files, then the directory should have at least 1 GB of space to cope with a worst-case write order scenario. On some Linux systems, for example, the `/tmp` directory might have only a few GB of disk space. Use of the `sync` option by clients reduces the burden on the dump directory by ensuring more ordered writes.

By default, the `/tmp/.hdfs-nfs` directory is specified. This directory could be changed using the Ambari Web UI by browsing to the **Services > HDFS > Configs > Advanced > NFS Gateway** section. Modify the **NFSGateway** dump directory setting. Only the NFS Gateway requires restarting after changing this property.

Installing and Configuring HDFS NFS Gateway

Installing NFS Software Packages

Linux provides a variety of NFS utilities for administering and monitoring NFS exports and mounts. At the least these utilities should be installed on the NFS Gateway and NFS client machines. For example, the `showmount` command is not available in CentOS until the `nfs-utils` package is installed.

To ensure that all NFS utilities are available, install the NFS software packages appropriate for the OS type. For example, to install the NFS packages on a CentOS host, run the command `yum -y install nfs*`.

Installing an NFS Gateway

The screenshot shows the Ambari Web UI interface. At the top, there's a navigation bar with tabs for 'Dashboard', 'Services', 'Hosts', and 'Alerts'. Below the navigation is a search bar labeled 'Filter: All (1)'. Underneath the search bar is a table header with columns: IP Address, Rack, Cores, RAM, Disk Usage, and Load Avg. In the 'Selected Hosts (0)' section, there's a dropdown menu with options: 'Selected Hosts (0)', 'Filtered Hosts (1)', and 'All Hosts (1)'. The 'Filtered Hosts (1)' option is currently selected, showing one host entry: 'node4' with IP '172.17.0.2', Rack '/default-rack', Cores '4 (4)', RAM '14.44GB', Disk Usage '0.08', and Load Avg '0.08'.

Installing an NFS Gateway with the Ambari Web UI

Installing the NFS Gateway was a completely manual process before Ambari 2.1. Starting with Ambari 2.1, the NFS Gateway can be installed using Ambari either during initial cluster deployment or after deployment using the Ambari Web UI. The process to add an NFS Gateway after a cluster has been deployed requires adding a new host to the cluster. To add a new host using the Ambari Web UI, select **Hosts > Add New Hosts**.

Even though Ambari automates the NFS Gateway installation, you must still manually mount the exported HDFS root directory on an NFS client host. Mounting is described later in this lesson.

Selecting Nodes for Ambari Agents

The screenshot shows the 'Add Host Wizard' interface. On the left, a sidebar lists steps: 'ADD HOST WIZARD', 'Install Options' (which is selected), 'Confirm Hosts', 'Assign Slaves and Clients', 'Configurations', 'Review', 'Install, Start and Test', and 'Summary'. The main panel is titled 'Install Options' and contains a sub-section 'Target Hosts' with a text input field containing 'node4'. A callout bubble points to this field with the text 'Enter the hostname of the NFS Gateway'. Below this, there's a section 'Host Registration Information' with a radio button selected for 'Provide your SSH Private Key to automatically register hosts'. A file browser window is open, showing a file named 'training-keypair.pem'. A tooltip over this section says 'Assumes password-less SSH access is configured'. At the bottom right is a green 'Register and Confirm' button.

Selecting Nodes for Ambari Agents

An Ambari agent must be installed on a host and then registered with the Ambari Server before the Ambari Server can install an NFS Gateway on the host. Type the fully-qualified domain name (FQDN) of a host in the **Target Hosts** text box. If you choose to use a short hostname instead, the installer will open a warning window when you click **Register and Confirm**. Use of FQDNs is highly recommended to ensure proper cluster operation in all situations.

The Host Registration Information section includes two radio buttons. Which radio button is selected depends on whether or not you have pre-configured the Ambari Server machine with password-less SSH access to the cluster nodes. If password-less SSH has been pre-configured, select the first radio button and click **Browse**. Use the Browse window to locate the pre-configured SSH RSA private key file that enables the Ambari Server to log in to each node. The Hortonworks online manuals at <http://docs.hortonworks.com> include instructions for configuring password-less SSH log in.

If password-less SSH log in has not been pre-configured, select the second radio button. This opens a window warning you that you must manually install and configure Ambari agent software on each node. Once installed and properly configured, an agent will automatically register with the Ambari Server. Manually installing and registering Ambari agents is described in the Hortonworks online documentation at <http://docs.hortonworks.com>.

After the choices have been made, click **Register and Confirm**.

Installing and Registering Ambari Agents

Host	Progress	Status	Action
node4	<div style="width: 100%;">Success</div>	Success	<input type="button" value="Remove"/>

Monitoring the Ambari Agent Installation and Registration Process in the Confirm Hosts Window

The Confirm Hosts window enables you to monitor the progress of the Ambari agent installation and registration process. A node with a successful agent registration displays Success and a green progress bar.

Ambari performs configuration checks on the node once an agent has been installed. If Ambari detects any potential configuration problems they can be viewed by clicking the link **Click here** to see the warnings. A window opens with detailed information about potential issues. Some warnings are serious and must be resolved before proceeding while some warnings can be potentially ignored. In each case, a warning must be evaluated for its impact on cluster operation.

When ready to proceed with the installation process, click **Next**.

Assigning Slaves and Clients

The screenshot shows the 'Assign Slaves and Clients' step of the 'Add Host Wizard'. On the left, a sidebar lists steps: 'Install Options', 'Confirm Hosts', 'Assign Slaves and Clients' (which is selected), 'Configurations', 'Review', 'Install, Start and Test', and 'Summary'. The main area has a title 'Assign Slaves and Clients' with a sub-instruction: 'Assign slave and client components to hosts you want to run them on. Hosts that are assigned master components are shown with *.' Below is a table:

Host	all none	all none	all none	all none
node4	<input type="checkbox"/> DataNode	<input checked="" type="checkbox"/> NFSGateway	<input type="checkbox"/> NodeManager	<input type="checkbox"/> Client

At the bottom are 'Back' and 'Next >' buttons.

Assigning Slaves and Clients

Use the Assign Slaves and Clients window to choose where to run service worker components, an NFS Gateway, and Hadoop client software.

At a minimum you must select the **NFSGateway** check box. You may optionally select the **DataNode**, **NodeManager**, and **Client** check boxes. Selecting the **NFSGateway** check box installs the necessary NFS software, the Hadoop JAR files, and the Hadoop configuration files.

Make the required selections and click **Next**.

Choosing Configuration Groups

The screenshot shows the 'Configurations' step of the 'Add Host Wizard'. The sidebar is identical to the previous one. The main area has a title 'Configurations' with a sub-instruction: 'Select the configuration groups to which the added hosts will belong to.' Below is a table:

Service	Configuration Group
HDFS	HDFS Default

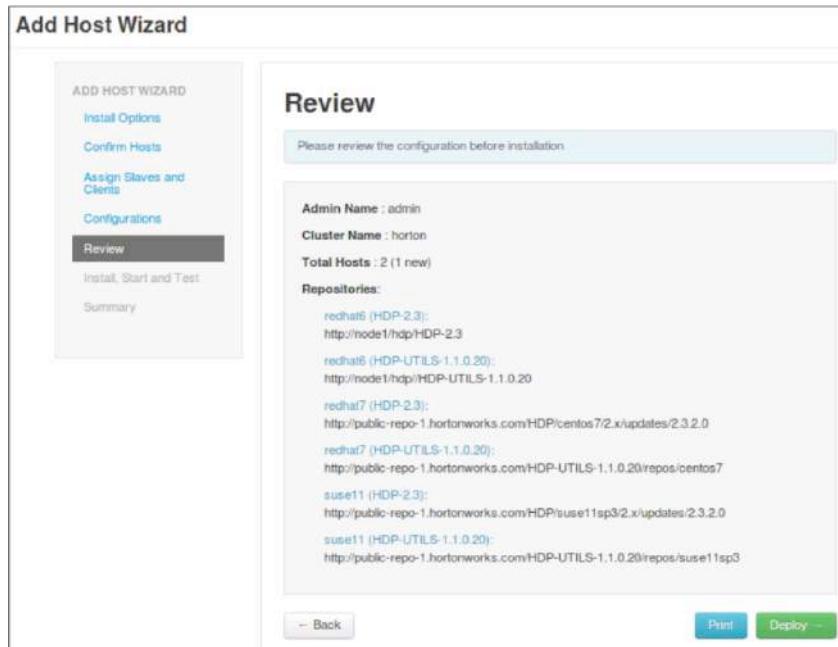
At the bottom are 'Back' and 'Next >' buttons.

Choosing Configuration Groups

Ambari configuration groups accommodate differences between sets of cluster nodes. Different sets of nodes can be configured with different configuration settings. Groups are primarily based on differences in hardware such as number of CPUs, amount of memory, etc.

The configuration settings that should be applied to a new NFS Gateway might vary based on the size and amount of the NFS Gateway's compute, storage, and network resources. For this reason, you are offered the choice of choosing a specific configuration group for each service.

Reviewing and Deploying NFS Gateway

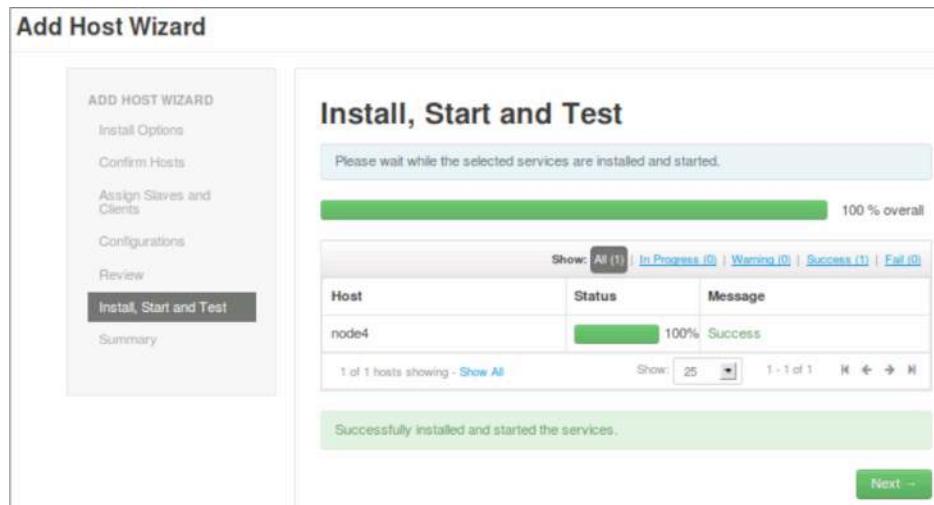


Reviewing and Confirming Installation Choices

The Review window enables you to print the configuration and deploy the NFS Gateway.

Click **Deploy** when ready to install the software.

Monitoring the Installation Process



Monitoring the Installation Process from the Install, Start and Test Window

The Install, Start, and Test window displays the installation progress of the NFS Gateway. If the NFS Gateway fails to install properly, the error displayed in the Message column is a hypertext link. Click the link to get more information about the error. If you can resolve the issue, you can use Ambari to install the NFS Gateway again.

Update Access Time Property

NFSGateway maximum Java heap size: 1024 MB
Access time precision: 3600000 ms
Allowed hosts: *:rw
NFSGateway dump directory: /tmp/.hdfs-nfs

Updating the Access Time Property

The `dfs.namenode.accesstime.precision` property in the `hdfs-site.xml` file enables the NFS Gateway to work with the HDFS NameNode to update file access times. By default, an NFS client mounts an export with access time updates enabled, but the default property value of 0 disables access time updates in HDFS. Updating the property to 3,600,000 milliseconds, or one hour, ensures that NFS clients can write to HDFS through the NFS Gateway.

To change the property value in the Ambari Web UI, browse to **Services > HDFS > Configs > Advanced > NFS Gateway** and change **Access time precision** to 3600000. Save the change and then restart any services indicated in the Ambari Web UI.

Verifying NFS Gateway Startup

Ambari

node4

Summary Configs Alerts 0 Versions

Components	Host Metrics
Metrics Monitor / Ambari Metrics Started	100%
NFSGateway / HDFS Started	

Verifying Gateway Startup with Ambari

Use the Ambari Web UI to verify that the NFS Gateway has been started. Browse to the Hosts page and click the NFS Gateway host. The NFSGateway service should be started.

Command Line

```
[root@node4 ~]# rpcinfo -p node4
    program vers proto port service
    100000    4   tcp   111  portmapper
    100000    3   tcp   111  portmapper
    100000    2   tcp   111  portmapper
    100000    4   udp   111  portmapper
    100000    3   udp   111  portmapper
    100000    2   udp   111  portmapper
    100005    1   udp   4242  mountd
    100005    2   udp   4242  mountd
    100005    3   udp   4242  mountd
    100005    1   tcp   4242  mountd
    100005    2   tcp   4242  mountd
    100005    3   tcp   4242  mountd
    100003    3   tcp   2049  nfs
[root@node4 ~]# showmount -e node4
Export list for node4:
/ *
[root@node4 ~]#
```

Verifying Gateway Startup from the Command Line

The command line may also be used to verify NFS Gateway startup.

The `rpcinfo` command verifies that multiple versions of the portmapper, mountd, and nfs services have all been registered to TCP and UDP port numbers. The NFS Gateway only uses TCP. In NFS operation the mountd and nfs services do not run at well-known ports. Instead, at startup they bind to ephemeral ports and then register their port and program numbers with the portmapper, which is sometimes called `rpcbind`.

The portmapper runs at well-known port 111. An NFS client does not know ahead of time the port numbers of the mountd or nfs services but they do know the program numbers of these services along with the well-known port number of the portmapper. The NFS client connects to the portmapper and requests the current port numbers of the mountd and nfs services.

The `showmount` command verifies that the HDFS / directory has been exported by the NFS Gateway. In the `showmount` example, the asterisk is the access control list. The asterisk denotes that any client machine may mount the exported HDFS root directory from the NFS Gateway.

Configuring HDFS NFS Gateway Client

Each NFS client will require an NFS mount point. An NFS mount point is typically an empty directory beneath the Linux / directory. For example, the command: `mkdir /hdfs_nfs` would create an NFS mount point.

To immediately mount the HDFS export use the `mount` command. For example, the command

```
mount -t nfs -o vers=3,proto=tcp,nolock,noacl,sync node4:/ /hdfs_nfs
```

mounts the HDFS / directory exported from the NFS Gateway on node4 to the `/hdfs_nfs` directory on the local NFS client.

The `NFS mount` command features many options.

- The NFS Gateway only uses NFS version 3 and TCP as the transport method. For these reasons include the `vers=3,proto=tcp` options.
- The Network Lock Manager (NLM) is not supported so include the `nolock` option.
- Linux and HDFS ACLs are implemented differently so the `noacls` must be included.
- The `sync` option is highly recommended because it can minimize or avoid reordered sequential writes and result in more predictable throughput.
- Hard mounts are the default and should not be changed because even after a client sends all data to the NFS Gateway, it might take some additional time for the data to transfer to HDFS. Hard mounts mitigate against data loss.
- If very large files are being uploaded or downloaded then the `wsize` or `rsize` options can be included to increase transfer size. The NFS Gateway supports a maximum of 1 MB transfers.

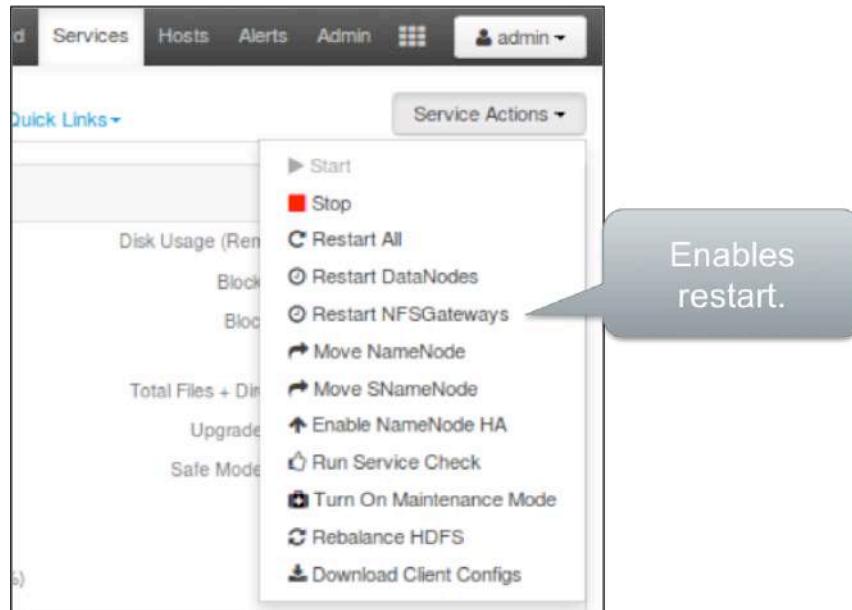
To verify the mount, type the `mount` command without any options. The list of current mounts is displayed.

To ensure that NFS clients mount HDFS even following a reboot, add the NFS mount to the client's mount table file. For example, the mount table file in CentOS is the `/etc/fstab` file. For example, an entry might be: `node4:/ /hdfs_nfs nfs vers=3,proto=tcp,nolock,noacl,sync.`

Starting and Stopping HDFS NFS Gateway Service

The Ambari Web UI can start, stop, and restart the NFS Gateway service. For example, you would have to restart the service if you modified an NFS Gateway configuration property.

From the Services page



Starting and Stopping Services from the Services Page

From the Hosts page

The screenshot shows the Ambari interface for managing services on a host named 'node4'. At the top, there are tabs for Dashboard, Services, Hosts, and Alerts. Below the tabs, the host name 'node4' is displayed with a status indicator (orange circle with minus sign). A 'Back' link is present. The main area has tabs for Summary, Configs, Alerts (0), and Versions. Under Components, two services are listed: 'Metrics Monitor / Ambari Metrics' (status: Started) and 'NFSGateway / HDFS' (status: Stopped). For the stopped service, a dropdown menu is open with options: Start, Turn On Maintenance Mode, and Delete. To the right, a 'Host Metrics' section displays a graph of CPU Usage. A large gray speech bubble points to the 'Start' button in the dropdown, containing the text: 'Toggles between Stop and Start.'

Starting and Stopping Services from the Services Page

Knowledge Check

Knowledge checks can be used by students as a self-assessment tool.

Questions

- 1) True or false? The NFS Gateway supports new file creation or appends to an existing file.
- 2) How do you increase NFS Gateway scalability?
- 3) Does HDFS use the user's name or the user's UID when checking permissions?
- 4) For the NFS Gateway, what is the purpose of the `hadoop.proxyuser.hdfs.hosts` property?
- 5) If the `nfs.exports.allowed.hosts` property allows only read-only access, but a user's HDFS permissions allow read-write access, which type of access does the user get?
- 6) What is the purpose of the NFS Gateway dump directory?
- 7) Which NFS client mount option reduces the dependence on the NFS Gateway dump directory?

Answers

- 1) True or false? The NFS Gateway supports new file creation or appends to an existing file.

Answer: True

- 2) How do you increase NFS Gateway scalability?

Answer: Add more NFS Gateways and divide the NFS clients between them

- 3) Does HDFS use the user's name or the user's UID when checking permissions?

Answer: The user's name

- 4) For the NFS Gateway, what is the purpose of the `hadoop.proxyuser.hdfs.hosts` property?

Answer: The property determines which hosts may run an NFS Gateway as user "hdfs"

- 5) If the `nfs.exports.allowed.hosts` property allows only read-only access, but a user's HDFS permissions allow read-write access, which type of access does the user get?

Answer: The property takes precedence and the user gets read-only access

- 6) What is the purpose of the NFS Gateway dump directory?

Answer: The NFS Gateway dump directory is used to reorder writes for larger files

- 7) Which NFS client mount option reduces the dependence on the NFS Gateway dump directory?

Answer: The sync option

Summary

- The NFS Gateway is a very easy way to browse, download, upload, or stream data to HDFS.
- The NFS Gateway is not the fastest, most scalable solution.
- Multiple NFS Gateways increase scalability but client NFS mounts do not failover between gateways.
- By default, the HDFS root directory export can be mounted by any NFS client but it is possible to add an optional access control list.
- Install Linux NFS software packages on the NFS Gateway and NFS client hosts.
- Use the Ambari Web UI to install the NFS Gateway.
- Edit the NFS client `/etc/fstab` file to make the mount automatic at reboot.

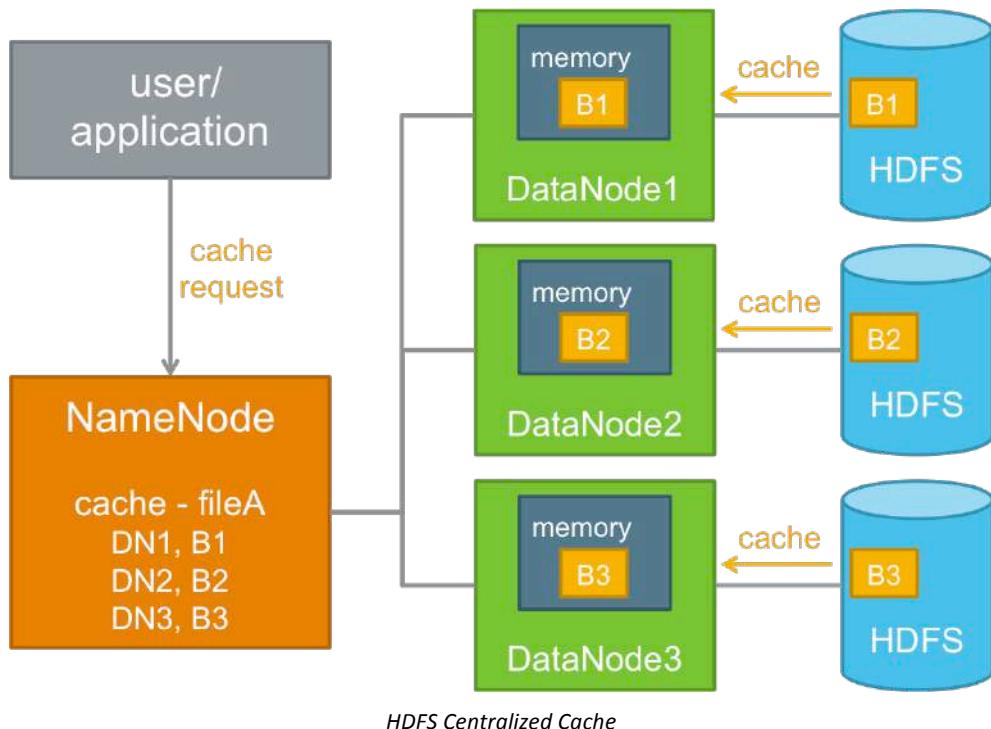
Configuring HDFS Centralized Cache

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Summarize the purpose and operation of HDFS centralized caching
- ✓ Configure HDFS centralized cache
- ✓ Define and manage Cache Pools and Cache Directives

HDFS Centralized Cache



Centralized HDFS cache enables a user or application to specify paths to directories or files that will be cached in DataNode memory by HDFS, thereby improving performance for applications that repeatedly access the same data. The file blocks are cached on the DataNodes in Java off-heap memory. HDFS centralized cache is possible because of the increasing memory capacities available on modern system hardware. This makes it possible for some working sets, or at least parts of working sets, to fit in aggregate cluster memory.

Users or applications send cache requests to the NameNode. The NameNode communicates with the DataNodes that have the file data blocks available on disk, and directs those DataNodes to cache the file data blocks in memory.

Use Cases

HDFS centralized cache is useful for two primary use cases.

- First, it increases performance when files are read repeatedly.

For example, if a small fact table is repeatedly used for joins with larger tables, the small fact table is a candidate for caching. As long as a file or files remain cached in memory it eliminates the repeated disk I/O operations that would have been necessary to keep rereading the file. The file must be small enough to fit in combined memory space of the DataNodes. How to control how much DataNode memory can be made available for caching is described later in this lesson.

- Second, caching one or more of the files that comprise the working set of a higher priority workload ensures that that workload does not have to compete for disk I/O resources with a lower priority workload.

Once the file or files has been cached in memory, no more disk I/O is necessary to read the data blocks. This can benefit mixed workload environments where there are performance service-level agreements.

Cache Operation

There were three design goals for HDFS centralized cache.

- The first goal was to provide a mechanism to lock data files into DataNode memory.

This enables faster read performance. This and other performance enhancements are described in more detail later in this lesson.

- The second goal was to provide global knowledge of cluster cache state.

This enables tasks to be scheduled for cache locality on the DataNodes. This is described in more detail later in this lesson.

- The third goal was to provide centralized management of the cluster cache.

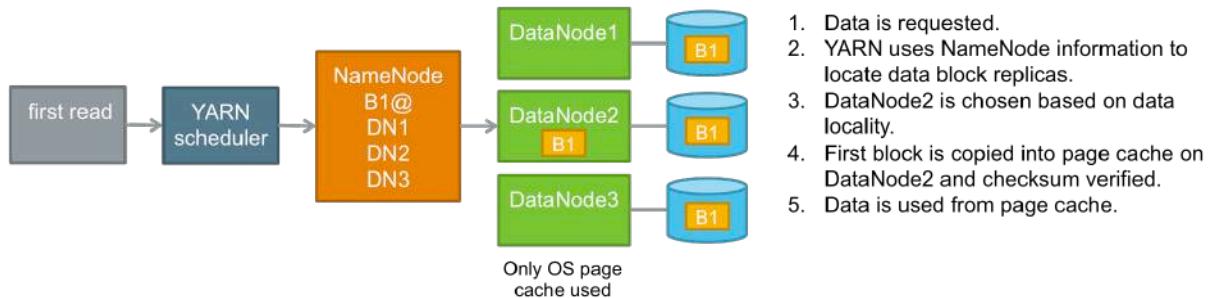
This enables administrative control over DataNode memory resources. These controls are described in more detail later in this lesson.

Determining Cache Locality

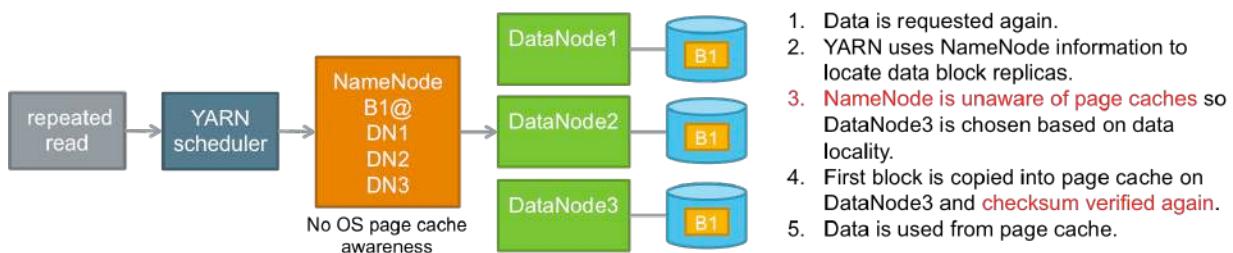
Even without HDFS centralized caching, a form of DataNode memory caching is still used. The Linux operating system initially copies any data read from disk to a page cache in the kernel's memory space. This includes any data blocks that are read from HDFS. The problem is that the NameNode is not aware of what is cached in the page caches on the DataNodes. This means that the information on the NameNode cannot be used by YARN to schedule tasks for cache locality. Tasks are only scheduled based on data locality—data is on the DataNode's disk.

No Cache Locality without Centralized Cache

These diagrams illustrate what happens when a replicated data block is repeatedly accessed and the NameNode is not aware of the state of the DataNode page caches.

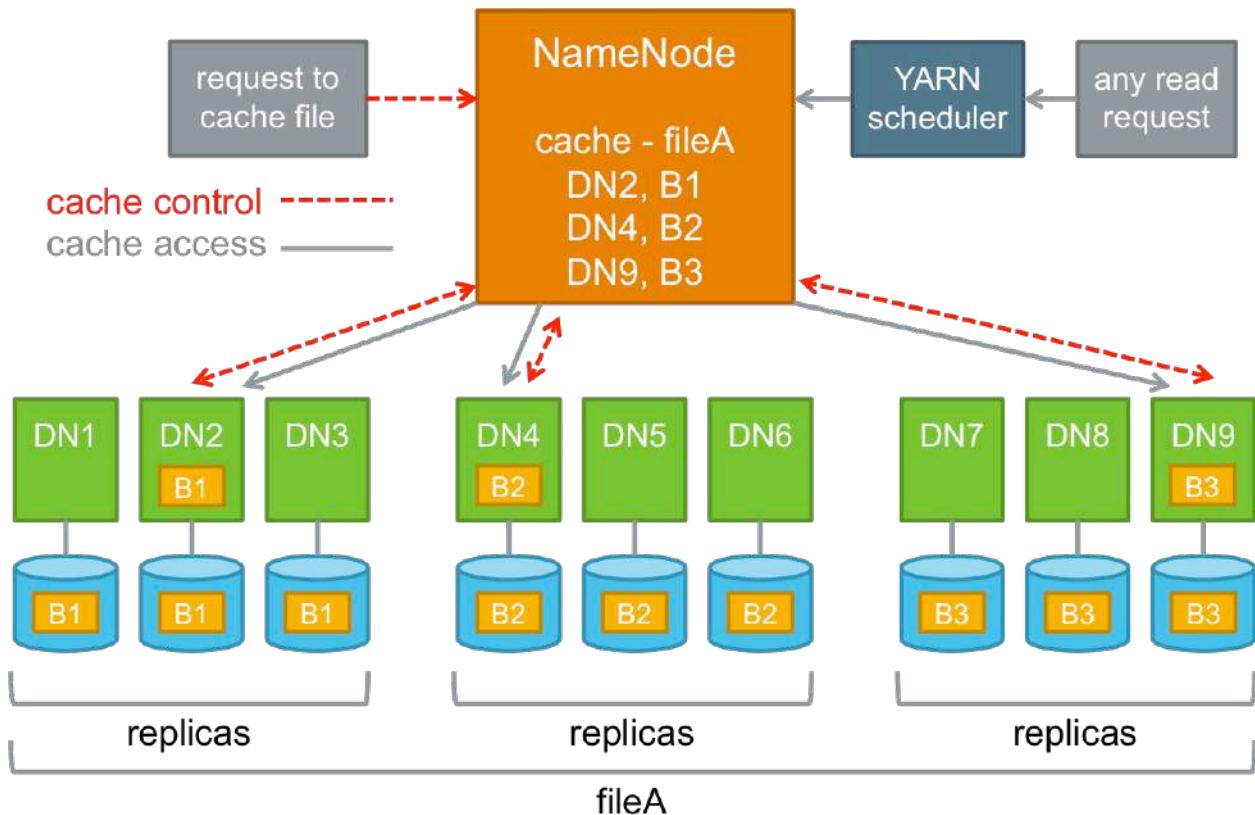


The first time the data block is requested the YARN job scheduler uses NameNode information to select a DataNode with local data. In the first diagram, the data block was read from the disk on DataNode2 and cached in the page cache. When the data was read its checksum was also verified. Using this same DataNode again would result in faster operation because there would be no disk I/O nor would the checksum have to be verified again.



In the second diagram the same data block is requested again. Once again the YARN job scheduler uses the NameNode information to select a DataNode with local data. However, because the NameNode is not aware of the state of the DataNode page caches, it cannot select a DataNode based on cache locality. The only basis for selection is data locality. This means that DataNode2 might not be used again, and in the diagram DataNode3 is used. The data block was read from the disk again, cached in the page cache again, and the checksum was verified again. All of these actions would have been unnecessary if the page cache on DataNode2 could have been used again.

There is yet another potential problem with the DataNode page caches. There is no way to lock, or pin, data into a page cache. The page cache is typically implemented using a least recently used (LRU) algorithm. This means that when there is contention for the page cache memory space, more recently accessed data overwrites older data. Because most clusters run multiple jobs at the same time, the jobs compete for page cache space. If second large job runs after a first job then the data blocks from the second job can overwrite the data blocks from the first job. In this situation, even if the NameNode was aware of the page caches, the first job's data is not locked into memory and might not be there when the first job runs again.

Cache Locality with Centralized Cache

With HDFS centralized cache, the NameNode manages and maintains information about any cached data blocks across all the DataNodes. The diagram illustrates the operation of HDFS centralized caching.

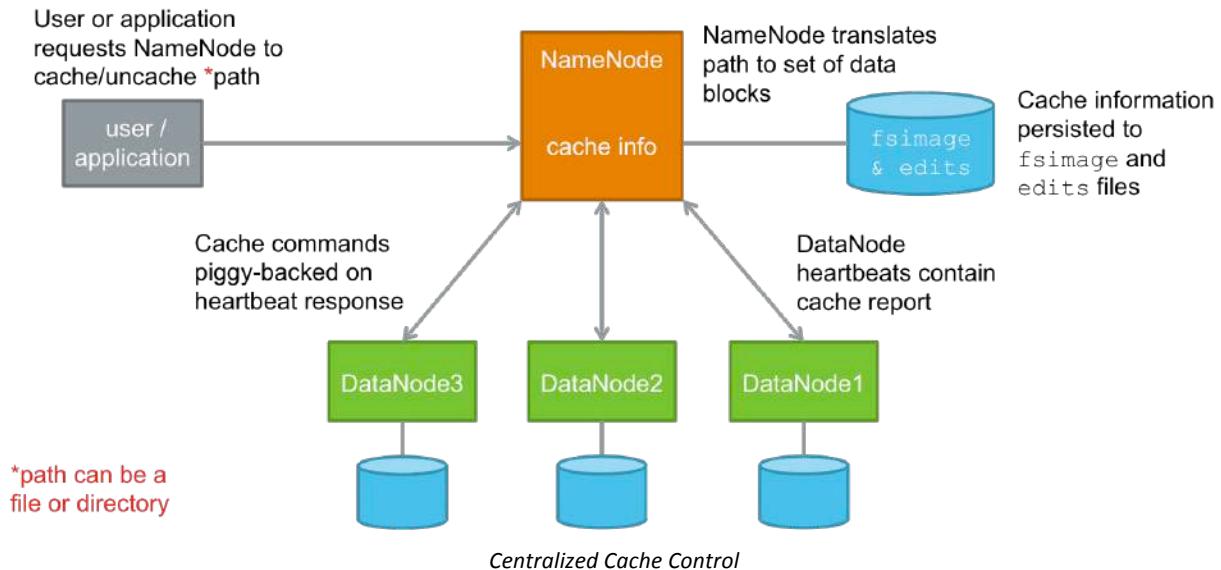
A user explicitly requests the NameNode to cache a file's data blocks in the HDFS centralized cache. Those blocks are read from disk and their checksums are verified. The blocks are then stored in DataNode memory until the user explicitly requests that they be uncached, or until an optional expiration time is reached.

The default replication factor for caching is one, which means that only one replica from one disk is stored in the cache on one DataNode. To use the cached data block, an application must be scheduled on the DataNode where the cached block is located. Once a block is cached in memory, its checksum does not have to be checked again for subsequent reads.

When an application needs to read the file's data blocks, the NameNode is aware that the data blocks are cached and on which DataNodes they are cached. The YARN scheduler uses the NameNode information to schedule tasks for cache locality.

If caching only a single replica on a single node could create a performance bottleneck, then a user could request that multiple replicas be cached across multiple DataNodes. How to cache one or more replicas is described later in this lesson.

Cache Control



The diagram illustrates how the HDFS centralized cache is controlled.

The NameNode is responsible for coordinating all of the DataNode off-heap caches in the cluster. The NameNode receives cache requests from users. HDFS centralized caching includes both a command-line interface for users and a Java API for applications. A user can request the NameNode to either cache or uncache a path. A path can be a file or directory. Directories are not recursively cached. Symlink targets are not cached. The NameNode manages DataNode caches by piggy-backing cache and uncache commands on the DataNode heartbeat responses.

The NameNode periodically receives a cache report from each DataNode. The cache report is piggy-backed on the DataNode heartbeat. The cache report describes all of the cached data blocks on the DataNode. How often a report is sent is controlled by a property described later in this lesson.

The aggregated data from the cache reports is maintained in NameNode memory. It is also persisted to disk in the `fsimage` and `edits` files.

Cache Pools and Directives

Cache Pool – name : owner : group : permissions : pool
memory limit : maximum expiration time

Cache Directive – path : expiration time : replication factor

Cache Directive – path : expiration time : replication factor

Cache Directive – path : expiration time : replication factor

Cache Pools and Directives

A Cache Pool is an administrative entity used to manage groups of Cache Directives. Cache Pools have Linux-like permissions that restrict which users and groups have access to the pool. Write permissions allow users to add and remove Cache Directives to the pool. Read permissions allow users to list the Cache Directives in a pool, as well as additional expiration, replication, and usage information. Execute permissions are unused.

Cache Pools are also used for resource management. Cache Pools can enforce a maximum memory limit, which restricts the aggregate number of bytes that can be cached by directives in the pool. The sum of the pool limits cannot exceed the amount of aggregate memory reserved for HDFS caching on the cluster. This limit is described later in this lesson. Cache Pools can also enforce a maximum expiration time, which no Cache Directive in the pool is allowed to exceed.

A Cache Directive defines the path that will be cached. Paths can be either directories or files. Directories are cached non-recursively; meaning only files in the first-level listing of the directory will be cached.

Cache Directives also specify additional parameters, such as the cache expiration time and replication factor. The expiration time is specified on the command line as a time-to-live (TTL), which represents a relative expiration time in the future. A Cache Directive that has expired is no longer taken into consideration by the NameNode when making caching decisions. The default expiration time is never.

The replication factor specifies the number of block replicas to cache. If multiple Cache Directives refer to the same file, the maximum cache replication factor is applied. The default replication factor is one.

Configuring Centralized Cache

Centralized Cache Requirements

There are a few requirements when configuring HDFS centralized cache. HDP 2.1 or later must be running on the cluster. The centralized cache depends on the Java Native Interface (JNI) code so HDP must be configured to use the `libhadoop.so` shared library. By default, HDP is configured to use JNI. The only required property is `dfs.datanode.max.locked.memory` in the `hdfs-site.xml` file. This property does not exist in `hdfs-site.xml` by default. To add the property and a value using the Ambari Web UI, browse to **Services > HDFS > Configs > Advanced > Custom hdfs-site > Add Property**.

The `dfs.datanode.max.locked.memory` property determines the maximum amount of per-DataNode memory that can be used by centralized caching. It is configured in bytes. The amount configured must be less than the amount of memory the operating system reserves for locked memory. The amount of memory reserved by the operating system can be displayed by running the `ulimit -l` command. While it varies by operating system, a common way to set this value is to modify the `/etc/security/limits.conf` file on every DataNode machine. The Windows operating system does not have an equivalent of the `limits.conf` file. If the `ulimit` setting is less than the `dfs.datanode.max.locked.memory` property setting, then the DataNodes will fail to restart after changing the property.

For example, assume that an administrator configures `dfs.datanode.max.locked.memory` equal to 268435456 bytes. The `limits.conf` file should be configured for a higher value by adding the entry `hdfs - memlock 269000`. The `hdfs` field represents the user that is locking the memory. It assumes that HDFS is running as the user `hdfs`. The dash (-) field informs the operating system to enforce a limit as a hard limit for the `hdfs` user. The `memlock` field represents the resource that is being limited, which is the amount of memory where data can be locked in and not removed. The last field represents the hard limit of 269000 kilobytes.

Optional Properties

Property	Description
<code>dfs.namenode.path.based.cache.refresh.interval.ms</code>	Determines how often, in ms, the NameNode rescans Cache Directives. (default is 30 seconds)
<code>dfs.cachereport.intevalMsec</code>	Determines how often, in ms, DataNodes send cache reports. (default is 10 seconds)
<code>dfs.datanode.cache.revocation.timeout.ms</code>	Determines how long, in ms, the DataNode waits before evicting a data block the client is reading without re-verifying the checksum. (default is 15 minutes)
<code>dfs.datanode.cache.revocation.polling.ms</code>	Determines how often, in ms, the DataNode should poll to determine if clients are using a data block the DataNode wants to uncache. (default is half second)

There are several optional properties that can be used to control HDFS centralized cache operation. The table lists some of the more interesting ones. None of these properties exists by default in the `hdfs-site.xml` file so the default values from the `hdfs-default.xml` file are used.

The Ambari Web UI can be used to add any of these properties to the `hdfs-site.xml` file with the purpose of changing the default values. To add and change a property in the Ambari Web UI, browse to **Services > HDFS > Configs > Advanced > Custom hdfs-site** and click **Add Property**.

Managing Cache Pools and Directives

HDP includes a command-line interface to manage HDFS centralized cache. The command is `hdfs cacheadmin`. You must be the HDFS superuser or a user with the required permissions in order to use this command.

The HDFS superuser may create, modify, list, and remove a Cache Pool. Any user with permissions may add, modify, list, and remove a Cache Directive.

To get help and syntax information about all of the `cacheadmin` sub-commands, type `hdfs cacheadmin -help`. To get help and syntax information about a specific `cacheadmin` sub-command, use `hdfs cacheadmin -help <sub-command>`.

Cache Pool Command Options

Option	Description
<code>-addPool name</code>	The name of the pool.
<code>[-owner <username>]</code>	The user name of the owner of the pool. (default is current user)
<code>[-group <groupname>]</code>	The group to which the pool is assigned. (default is primary group of the current user)
<code>[-mode <octalvalue>]</code>	The Linux-style permissions (owner, group, others) assigned to the pool specified as an octal value. (default is 755)
<code>[-limit <bytes>]</code>	The maximum number of bytes that can be cached by the directives in the pool. (default is no limit)
<code>[-maxTtl <value>]</code>	The maximum allowed time-to-live for directives being added to the pool in seconds (120s), minutes (10m), hours (12h), and days (7d). (default is never expires)

addPool Command Options

The `hdfs cacheadmin -addPool` sub-command includes several options. Understanding the options for this sub-command enables an administrator to understand the options available when using the other Cache Pool sub-commands. Review the options and their descriptions in the table.

Creating and Listing a Cache Pool

```
[root@node1 ~]# su - hdfs
[hdfs@node1 ~]$ hdfs cacheadmin -addPool stevePool -owner steve -mode 744 -limit 10485760 -maxTtl 7d
Successfully added cache pool stevePool.
[hdfs@node1 ~]$ hdfs cacheadmin -listPools stevePool
Found 1 result.
NAME      OWNER     GROUP      MODE          LIMIT        MAXTTL
stevePool  steve     hadoop     rwxr--r--  10485760  007:00:00:00.000
[hdfs@node1 ~]$ hdfs cacheadmin -listPools -stats stevePool
Found 1 result.
NAME      OWNER     GROUP      MODE          LIMIT        MAXTTL  BYTES_NEEDED  BYTES_C
ACHED  BYTES_OVERLIMIT FILES_NEEDED FILES_CACHED
stevePool  steve     hadoop     rwxr--r--  10485760  007:00:00:00.000           0
          0           0           0           0
[hdfs@node1 ~]$ ■
```

Creating and Listing a Cache Pool

Use the `hdfs -cacheadmin -addPool` and `-listPools` sub-commands to create a Cache Pool and then list its attributes.

HDFS superuser privileges are required to create a Cache Pool. In the example, `su - hdfs` was used to assume HDFS superuser privileges.

In the example, the `-addPool` sub-command was used to create a Cache Pool named `stevePool` that is owned by the user `steve`. The `744` permissions provide read and write access to the pool by the owner `steve`. The user `steve` will be able to add, modify, and remove Cache Directives. Everyone else on the cluster will be able to only view Cache Directives in the pool. Execute permissions are ignored.

Because the pool's group was not specified by adding the `-group <groupname>` option, the primary group of the `hdfs` user was used. The primary group in the example is the group `hadoop`.

The pool was created with a 10 megabyte limit, which means that all of the directives added to the pool cannot collectively cache more than 10 megabyte of data in memory across the cluster. If the Cache Directive being added would cause the pool's limit to be exceeded, the command fails with an appropriate error message.

The `-maxTtl` is configured for seven days. This means that no Cache Directive added to the pool can keep its data cached for longer than seven days. If the Cache Directive being added to the pool exceeds the maximum pool time-to-live, the command to add the Cache Directive fails with an appropriate error message. If the Cache Directive being added does not specify a time-to-live, it inherits the pool's time-to-live value. The default expiration date is never.

In the examples, notice that the `-listPools` sub-command can be used with or without the `-stats` option. The `-stats` option causes the command output to display current pool usage information.

Modifying and Listing a Cache Pool

```
[hdfs@node1 ~]$ hdfs cacheadmin -modifyPool stevePool -maxTtl 1d
Successfully modified cache pool stevePool to have max time-to-live 1d
[hdfs@node1 ~]$ hdfs cacheadmin -listPools
Found 1 result.
NAME      OWNER    GROUP    MODE          LIMIT        MAXTTL
stevePool  steve    hadoop   rwxr--r--  10485760  001:00:00:00.000
[hdfs@node1 ~]$
```

Modifying and Listing a Cache Pool

The `hdfs cacheadmin` includes a `-modifyPool` sub-command. It can be used to modify a pool's attributes except for the pool name. In the example the time-to-live value was changed from its previous setting to one day. Any Cache Directives in the pool whose time-to-live values exceed the new pool expiration limit will have their time-to-live values automatically truncated to match the pool's value.

Cache Directive Command Options

Option	Description
<code>-path <pathname></code>	The path to cache. The path can be a file or directory.
<code>-pool <poolname></code>	The Cache Pool to which the Cache Directive is added. You must have write permission on the Cache Pool.
<code>[-force]</code>	Skips checking of the Cache Pool limits.
<code>[-replication <number>]</code>	The cache replication factor determines how many DataNodes cache each data block. (default is 1)
<code>[-ttl <value>]</code>	The maximum allowed time-to-live for the directive pool in seconds (120s), minutes (10m), hours (12h), and days (7d). Cannot exceed the pool time-to-live maximum. (default is never expires)

addDirectives Command Options

The `hdfs cacheadmin -addDirectives` sub-command includes several options. Understanding the options for this sub-command enables an administrator or user to understand the options available when using the other Cache Directive sub-commands. Review the options and their descriptions in the table.

Adding and Listing a Cache Directive

```
[root@node1 ~]# su - steve
[steve@node1 ~]$ hdfs cacheadmin -addDirective -path /user/steve/lookuptable -pool stevePool
Added cache directive 3
[steve@node1 ~]$ hdfs cacheadmin -listDirectives -pool stevePool
Found 1 entry
  ID POOL      REPL EXPIRY PATH
  3 stevePool    1 never   /user/steve/lookuptable
[steve@node1 ~]$ hdfs cacheadmin -listDirectives -pool stevePool -stats
Found 1 entry
  ID POOL      REPL EXPIRY PATH          BYTES_NEEDED  BYTES_CACHE
D FILES_NEEDED FILES_CACHED
  3 stevePool    1 never   /user/steve/lookuptable           173
0          1          0
[steve@node1 ~]$ hdfs cacheadmin -listDirectives -pool stevePool -stats
Found 1 entry
  ID POOL      REPL EXPIRY PATH          BYTES_NEEDED  BYTES_CACHE
D FILES_NEEDED FILES_CACHED
  3 stevePool    1 never   /user/steve/lookuptable           173           17
3          1          1
[steve@node1 ~]$ ■
```

Adding and Listing a Cache Directive

Use the `hdfs -cacheadmin -addDirective` and `-listDirectives` sub-commands to create a Cache Directive and then list its attributes.

A normal user must have write permission on a pool to create a Cache Directive. In the example, `su - steve` was used to assume the user steve's permissions.

The `-addDirective` sub-command was used to cache the file `/user/steve/lookuptable` into the pool named `stevePool`. Then the `-listDirectives` sub-command was used to display the directives in the pool named `stevePool`. The Cache Directory was automatically assigned an ID of 4. The first ID assigned is one and is incremented by one each time a new directive is added. Cache Directive ID numbers are never reused.

The `-listDirectives` sub-command includes the `-pool <poolname>` option. This option causes all Cache Directives in the specified pool to be listed. The `-listDirectives` sub-command also includes a `-path <pathname>` option. This option lists any Cache Directive with the specified path, regardless of its pool location.

Notice that the `-listDirectives` sub-command can be used with and without the `-stats` option. The `-stats` option causes the command output to include current directive usage information. Also notice that the first time the `-stats` option was used, the output for `BYTES_CACHED` and `FILES_CACHED` were zero. That is because it typically takes a minute or more to cache a file, depending on the file's size. When the `-stats` options was used a few minutes later, the `BYTES_CACHED` and `FILES_CACHED` fields were non-zero.

Removing a Cache Directive and Cache Pool

```
[steve@node1 ~]$ hdfs cacheadmin -listDirectives -path /user/steve/lookuptable
Found 1 entry
ID POOL          REPL EXPIRY PATH
3 stevePool      1 never   /user/steve/lookuptable
[steve@node1 ~]$ hdfs cacheadmin -removeDirectives -path /user/steve/lookuptable
Removed cache directive 3
Removed every cache directive with path /user/steve/lookuptable
[steve@node1 ~]$ exit
logout
[root@node1 ~]# su - hdfs
[hdfs@node1 ~]$ hdfs cacheadmin -listPools
Found 1 result.
NAME      OWNER GROUP MODE           LIMIT MAXTTL
stevePool  steve  hadoop rwxr--r--  10485760 never
[hdfs@node1 ~]$ hdfs cacheadmin -removePool stevePool
Successfully removed cache pool stevePool.
[hdfs@node1 ~]$ █
```

Removing a Cache Directive or Cache Pool

Use the `hdfs -cacheadmin -removePool` and `-removeDirectives` sub-commands to remove a Cache Pool and Cache Directive.

HDFS superuser privileges are required to remove a Cache Pool. A normal user must have write permissions on a pool to remove a Cache Directive.

In the example the `-listDirectives` sub-command was used to display the path associated with the Cache Directive. Then using the path, the `-removeDirectives` sub-command was used to remove the Cache Directive. If the specified path was used in multiple Cache Pools, all of the Cache Directives that share that path would be removed.

The `-listPools` sub-command was used to display the names of the Cache Pools. Then, using the pool name, the `-removePools` sub-command was used to remove the Cache Pool.

This page left blank intentionally.

Knowledge Check

Use the following questions and answers to assess your understanding of the concepts presented in this lesson.

Questions

- 1) HDFS centralized cache is useful for increasing performance when smaller files are read _____.
- 2) True or false? Because HDFS centralized cache is implemented on the NameNode, the NameNode machine's memory must be large enough to hold all the cached file data blocks.
- 3) Which centralized cache object is used to control user access to the cache?
- 4) Which permissions are necessary to create a Cache Pool?
- 5) Which permissions are necessary for a user to add or remove a Cache Directive?
- 6) Which centralized cache property must be configured in concert with the operating system's `ulimit -l` setting?
- 7) Which `hdfs` command is used to manage Cache Pools and Directives?

Answers

- 1) HDFS centralized cache is useful for increasing performance when smaller files are read _____.

Answer: Repeatedly

- 2) True or false? Because HDFS centralized cache is implemented on the NameNode, the NameNode machine's memory must be large enough to hold the cached data blocks.

Answer: False, the NameNode manages the cache, the cache is distributed across DataNode memory.

- 3) Which centralized cache object is used to control user access to the cache?

Answer: The Cache Pool

- 4) Which permissions are necessary to create a Cache Pool?

Answer: HDFS superuser permissions

- 5) Which permissions are necessary for a user to add or remove a Cache Directive?

Answer: Write permissions on the containing Cache Pool

- 6) Which centralized cache property must be configured in concert with the operating system's ulimit -l setting?

Answer: dfs.datanode.max.locked.memory

- 7) Which hdfs command is used to manage Cache Pools and Directives?

Answer: hdfs cacheadmin

Summary

- HDFS centralized cache enables users and applications to specify files or directories to cache in DataNode memory.
- HDFS centralized cache is useful for increasing performance when files are read repeatedly.
- Cache requests are sent to the NameNode, which directs DataNodes to cache file blocks.
- The NameNode information can be used by the YARN scheduler to schedule tasks for cache locality.
- A Cache Pool is an administrative entity used to manage groups of Cache Directives.
- A Cache Directive defines the path that will be cached.
- The `hdfs cacheadmin` command is used to manage the cache.

Managing Data Compression

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Summarize the importance of file format and compression algorithm selection
- ✓ Describe the administrator and non-administrator roles in managing compression
- ✓ Summarize the benefits and considerations when using compression in Hadoop
- ✓ List and describe splittable compression formats
- ✓ List common compression algorithms available in Hadoop
- ✓ Define the compression algorithms available for use in a cluster
- ✓ Configure default MapReduce and Tez compression algorithms

File Formats and Compression

Hadoop applications use an input and output file format, and optional compression. Hadoop applications, for example, would include programs like Hive or Pig.

Choosing an optimal file format is perhaps the largest driver of application performance. Examples of relevant questions include: Does it compress well? Does it have a flexible schema so that new fields of information can be easily added without reprocessing large amounts of data? Common file formats are described later in this lesson.

In addition to file format, compression greatly affects application performance. Relevant questions include: Does the compression algorithm support record or block compression? How fast is compression and decompression? How much compression can be achieved? Is the compressed data splittable? Compression and compression concepts are described later in this lesson.

Roles for Managing Compression

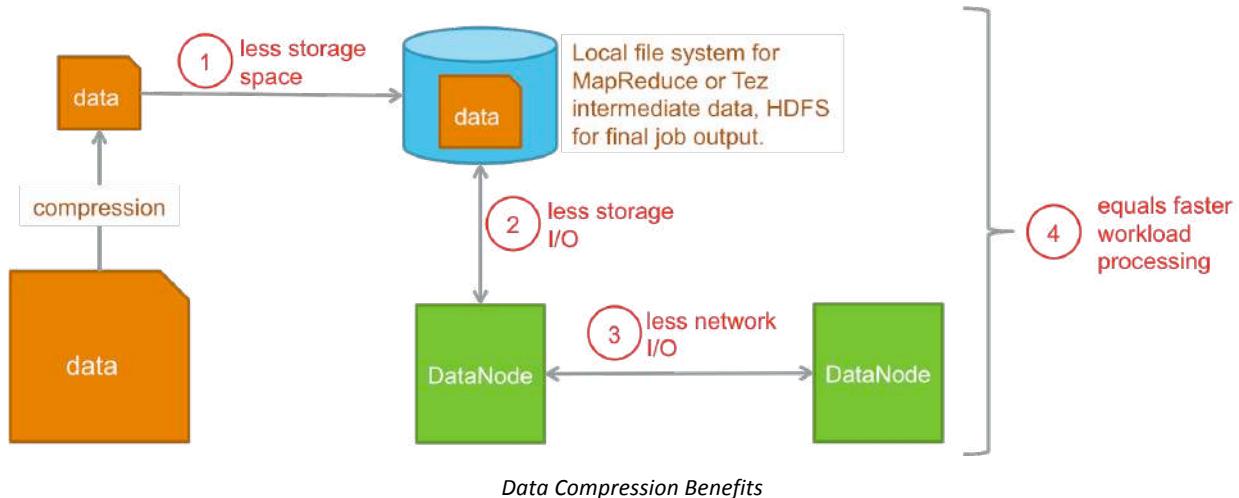
There is a division of responsibilities between a cluster administrator and application developers or users.

A cluster administrator adds new compression software to the cluster and configures the cluster to recognize the new software. This process is described later in this lesson. The administrator also optionally configures a default compression method for MapReduce and Tez. This is also described later in this lesson.

An application developer or user chooses which application file input and output formats to use. Common file formats are listed later in this lesson. An application developer or user may also choose to override any default compression methods defined by a cluster administrator.

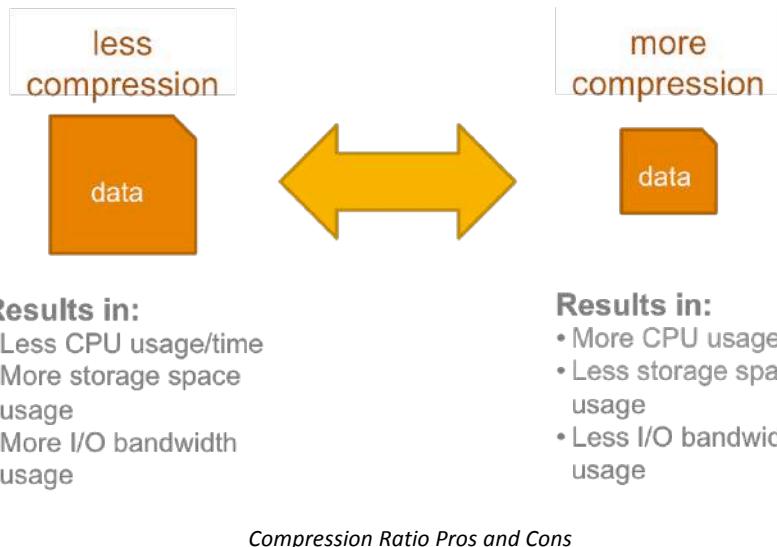
As previously mentioned, choosing an optimal file format and compression algorithm is critical for application performance. These developer issues and concerns are outside the scope of this administration lesson.

Benefits and Considerations



Data is commonly compressed in Hadoop. Data compression offers many benefits. Compressed data uses less storage space on the local DataNode file system or in HDFS. The local file system is used to temporarily store MapReduce and Tez intermediate data while HDFS is used to permanently store final job output. Compressed data requires less storage and network I/O bandwidth. All these benefits work together to enable faster processing of data in the Hadoop cluster.

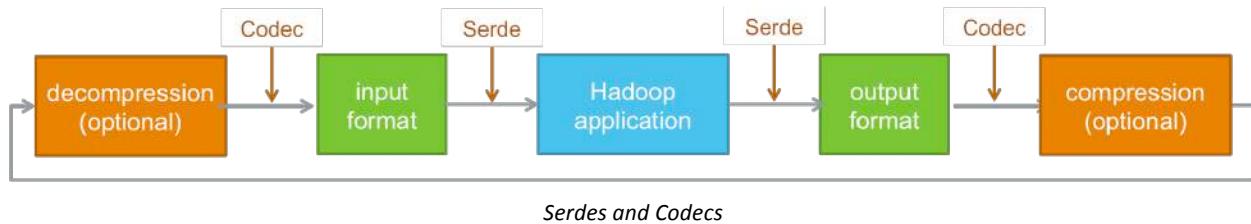
Ratio Pros and Cons



Using more or less compression is a “trade-off” between cluster resources. It is primarily a space versus time trade-off. Compressing data less requires fewer CPU resources and time but also requires more storage space and greater storage and network I/O resources. Compressing data more requires more CPU resources and time but less storage space and fewer storage and network I/O resources.

Hadoop clusters tend to be more I/O bound than they are CPU bound so using more compression in Hadoop typically yields better overall performance and more efficient cluster operation.

Serdes and Codecs

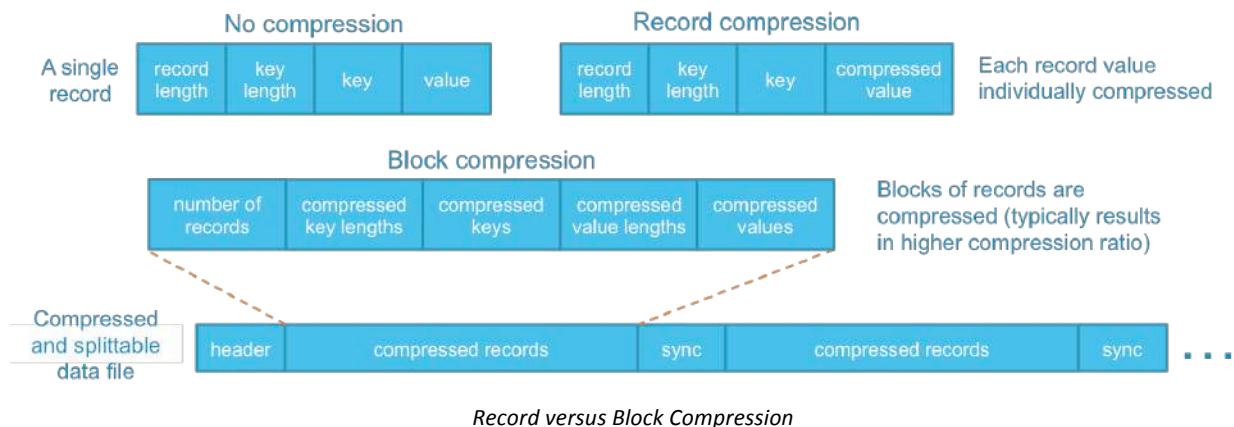


In order for an application to use a specific file format, it must have a Serde for that format. A Serde is a **serializer / deserializer** that is specific to a particular file format. Serialization is the process of converting structured objects into a byte stream for transmission over a network or for writing to disk. Deserialization is the reverse process. For example, Hive and Pig default to a Serde that reads and writes text files but Serdes are available for other file formats. Common Hadoop file formats are listed later in this lesson.

Applications can also use optional file compression. The Hadoop administrator can define a default compression codec—**compression/decompression**—for MapReduce or Tez jobs. For example, if Hive or Pig is configured to use Tez, then Hive or Pig would use the administrator-defined default compression codec unless the user or application specifies an alternate compression codec.

When MapReduce or Tez reads a compressed file, it automatically detects the compression type and selects the correct codec to decompress the file.

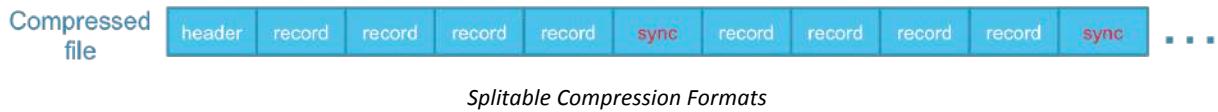
Record vs. Block Compression



Files contain individual records that must be processed. In this example, each record contains a key/value pair that could be processed by Tez or MapReduce. (The Sequence file format is used in the illustration.) The sync fields make the compressed data splittable. Splittable formats are described later in this lesson.

Hadoop includes codecs with the capability of performing record or block compression. Record compression individually compresses the value stored in each record. Block compression compresses groups of records. Block compression typically is more efficient and provides better compression ratios because it can take advantage of any similarities between the information in the records. Block compression should be preferred over record compression.

Splittable Compression Formats



Splittable compression formats insert synchronization markers into a file. The value of the sync marker is stored in the file's header. The number of inserted sync markers is typically minimized in order to minimize file size overhead. Sync markers are necessary in order for MapReduce or Tez maps to process file data locally on each DataNode.

Remember that files that are larger than the HDFS block size are split across multiple DataNodes. A sync marker enables a file reader on a DataNode, like Tez or MapReduce, to locate the start of a record in its portion of the file. This results in parallel read operation across DataNodes and faster data processing.

Without sync markers, a reader on a single DataNode must process an entire file. The DataNode with the first portion of the file must read the remaining portions of the file from other DataNodes. This loss of data locality results in increased network traffic between DataNodes and much slower data processing.

The use of splittable compression formats is highly recommended for MapReduce or Tez jobs, or any other data file that is processed across multiple DataNodes.

Splittability is not a concern for files smaller than the HDFS block size.

Common Hadoop File Types

Hadoop supports several file types. The file formats used are chosen by developers and end users. Choosing the best file format requires per-data, per-application performance testing.

Some of the more common ones are listed here.

Text/CSV files

These files are not splittable and do not support block compression. Text/CSV files do not have a schema, but new fields could be added to the end of a record. One advantage of text/CSV files is that they can be processed by many programming languages. They also make data portable between Hadoop and other database programs.

Sequence files

These files are splittable and support block compression. Sequence files are processed only by the Java programming language.

Avro files

These files are splittable and support block compression. The advantages of Avro file format are that they provide a good, multi-purpose storage format and include good support for schema evolution.

Record Columnar files (RCFiles)

These files are splittable and support block compression. RCFile tabular data does not support schema evolution. RCFile format has been superseded by the ORC and Parquet file formats.

Optimized Row Columnar (ORC) files

These files are splittable and support block compression. ORC file tabular data does not support schema evolution.

Parquet files

These files are splittable and support block compression. Parquet file tabular data has limited support schema evolution. New columns can be added to the end of a structure.

Hadoop Compression Algorithms

Format	Algorithm	Splittable	Distributed with Hadoop
zlib	Uses DEFLATE (LZ77 and Huffman coding)	no	yes ¹
gzip	Wrapper around zlib	no	yes
bzip2	Burrows-Wheeler transform	yes	yes
Snappy	LZ77	no	yes
LZO	Variant of LZ77	no ²	no ³
LZOP	LZOIX	no	No ³

The table lists the common compression options in Hadoop. The LZO and LZOP software is protected by the GNU General Public License (GPL) and cannot be distributed as part of Hadoop. It is freely available but must be downloaded and installed separately. For example, to download the LZO software type the command `yum install lzo lzo-devel hadoop-lzo hadoop-lzo-native`.

The gzip, bzip2, Snappy, and LZO algorithms are capable of block compression.

Best Compression Algorithm

There are several possible ways to measure a compression algorithm's performance. For example, if application performance is the highest concern then measure the compression speed or decompression speed. If reduced storage size is the highest concern then measure the compression factor. Another variable is that compression performance will vary by file size and file format.

This all means that the best compression format is going to be determined by testing the application and its data set. This testing is commonly going to be done by the application developer or end user on a per-application basis.

Common Hadoop Codecs

Format	Codec	Distributed with Hadoop
zlib	org.apache.hadoop.io.compress.DefaultCodec	yes
gzip	org.apache.hadoop.io.compress.GzipCodec	yes
bzip2	org.apache.hadoop.io.compress.BZip2Codec	yes
Snappy	org.apache.hadoop.io.compress.SnappyCodec	yes
LZO	com.hadoop.compression.lzo.LzoCodec	no
LZOP	com.hadoop.compression.lzo.LzopCodec	no

Codecs used in Hadoop are implemented as Java classes. Several codecs are part of the standard Hadoop distribution. These are listed in the table above. The codecs for LZO and LZOP must be downloaded and installed by a system administrator before they are available for use by application developers or end users.

Configure New Cluster Algorithms

It only takes a few steps to make a new codec available to applications on a Hadoop cluster.

Defining Available Cluster Compression Algorithms

- 1) Download the software for the new codec. Some codec software includes both a JAR file and native libraries.

Depending on the software codec, you might have to compile the software in order to generate the required codec JAR file and any native libraries. Native libraries are sometimes required by the compression utilities. These steps will vary by codec.

- 2) Once the JAR file and any optional native libraries exist, they should be copied to all the worker nodes in the cluster. Codec JAR files can be copied to `/usr/hdp/2.3.0.0/hadoop/lib/`. This places them in the Java classpath. If the codec includes native libraries, they should be copied to `/usr/hdp/2.3.0.0/hadoop/lib/native/`. For example, the Snappy codec JAR file and native libraries are located in these directories. (The directory path component `2.3.0.0-2557` will vary based on the installed Hadoop version.)
- 3) Once the codec software has been installed on all the worker nodes, the cluster must be configured so that the codec is available to applications. The class name of the codec must be added to the comma-separated list of codec class names listed in the `io.compression.codecs` property in the `/etc/hadoop/conf/core-site.xml` file. To update this file using the Ambari Web UI, browse to **Services > HDFS > Configs > Advanced > Advanced core-site**. Find the property and add the new codec to the list.

- 4) Then restart any services as indicated in the Ambari Web UI.

Defining Default MapReduce and Tez Algorithms

Default compression algorithms can be defined for both MapReduce and Tez.

MapReduce Compression

A cluster administrator can configure default compression settings for MapReduce using the Ambari Web UI. The Ambari Web UI updates the `/etc/hadoop/conf/mapred-site.xml` file. Go to **Services > MapReduce2 > Configs > Advanced > Advanced mapred-site** and either update or add the following properties.

To ensure that all MapReduce intermediate data that is written to the local file system, or sent over the network, is compressed, configure the `mapreduce.map.output.compress` property to `true`. By default it is `false`. You must then configure default compression algorithm to use for this compression. By default the `mapred.map.output.compress.codec` property is not defined in the `mapred-site.xml` file. To use Snappy, for example, add and configure the `mapred.map.output.compress.codec` property to `org.apache.hadoop.io.compress.SnappyCodec`. Snappy is very fast although it does not compress as much as other compression algorithms. With temporary data this might be appropriate.

The administrator can also configure default compression settings for final MapReduce job output to HDFS storage. This is optional in that this type of compression is typically configured on a per-job basis either by the application developer or by an end user when a job is submitted to a cluster. Even with a default configuration, the default can be overridden on a per-job basis.

To configure a default compression for final job output, configure the `mapreduce.output.fileoutputformat.compress` property to `true`. By default it is `false`. You must then configure default compression algorithm to use for this compression. To use gzip, for example, add and configure the `mapreduce.output.fileoutputformat.compress.codec` property to `org.apache.hadoop.io.compress.GzipCodec`. Gzip is a little slower although it provides better compression than Snappy. With permanent data this might be appropriate.

Finally configure the compression algorithm to use record or block compression. Depending on the compression algorithm, the choices are `NONE`, `RECORD`, or `BLOCK`. Block compression is typically the most efficient as there are commonly more opportunities for compression across many records than there are for a single record.

After making these configuration changes, restart any services as indicated in the Ambari Web UI.

Tez Compression

Apache Tez is a newer, more efficient processing framework than MapReduce. Like MapReduce, Tez can produce intermediate data and final data. Both can be compressed. Unlike MapReduce, Tez only has property values to compress intermediate data. Final job output compression is controlled by the application developer or end user when they submit a job.

Tez intermediate compression is enabled by default in HDP 2.3. The compression properties are located in the `/etc/tez/conf/tez-site.xml` file. To view or modify the property settings, in the Ambari Web UI browse to **Services > Tez > Configs > General**.

The `tez.runtime.compress` property is set to `true` and enables intermediate data compression. The `tez.runtime.compress.codec` property is set to `org.apache.hadoop.io.compress.SnappyCodec` and configures the compression method to use to compress the intermediate data. Another method could be chosen by choosing another codec.

This page left blank intentionally.

Knowledge Check

Knowledge checks can be used by students as a self-assessment tool.

Questions

- 1) True or false? The administrator has final control of the output file format and compression algorithm used by an application.
- 2) True or false? In Hadoop, the CPU overhead associated with compression typically outweighs the benefits gained by compression.
- 3) True or false? Applications require a specific codec to work with different file formats.
- 4) Splittable compression formats are enabled by inserting _____ .
- 5) What is the purpose of the `io.compression.codecs` property in the `core-site.xml` file?

Answers

- 1) True or false? The administrator has final control of the output file format and compression algorithm used by an application.

Answer: False

- 2) True or false? In Hadoop, the CPU overhead associated with compression typically outweighs the benefits gained by compression.

Answer: False

- 3) True or false? Applications require a specific codec to work with different file formats.

Answer: False – they require a specific serde

- 4) Splittable compression formats are enabled by inserting _____ _____ .

Answer: Sync markers

- 5) What is the purpose of the `io.compression.codecs` property in the `core-site.xml` file?

Answer: It defines the compression algorithms available to applications running on the cluster.

Summary

- Hadoop applications use an input and output file format and optional compression.
- Cluster administrators add new compression software to the cluster and optionally configure a default compression method for MapReduce and Tez.
- Using more or less compression is a “trade-off” between cluster resources.
- Specific Serdes are used to work with different file formats.
- Specific Codecs are used to work with different compression algorithms.
- Block compression is typically preferred over record compression.
- Splittable compression formats enable parallel read operation and faster data processing.
- Administrators add newly installed codec names to the `io.compression.codecs` property in the `core-site.xml` file.
- The “best” compression algorithm is determined by testing the application and its data set.

YARN Node Labels

Lesson Objectives

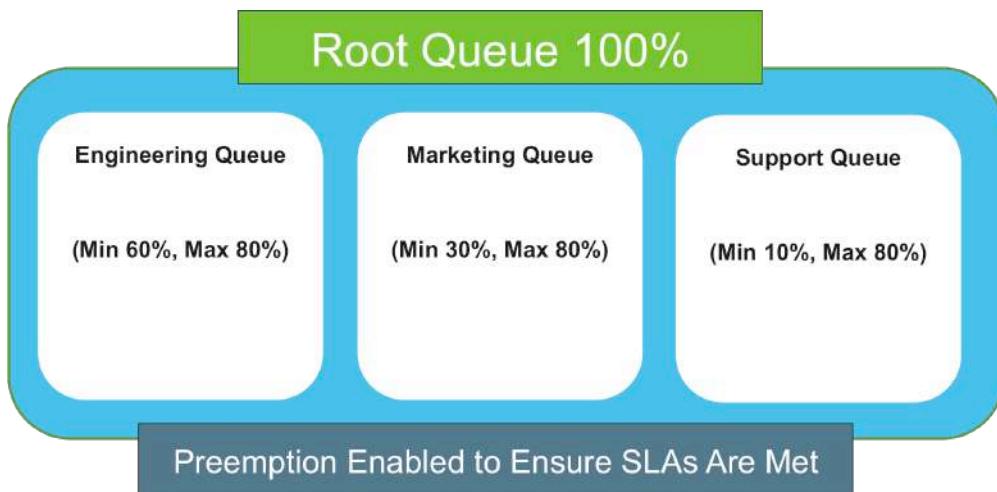
After completing this lesson, students should be able to:

- ✓ Summarize the purpose and operation of YARN node labels
- ✓ Create node labels
- ✓ Add a label to a node
- ✓ Modify or remove node labels
- ✓ Configure queues to access node label resources

Scenario

We will begin by creating a scenario in which node labels can be useful.

Organizations, Queues, and SLAs



Organizations, Queues, and SLAs

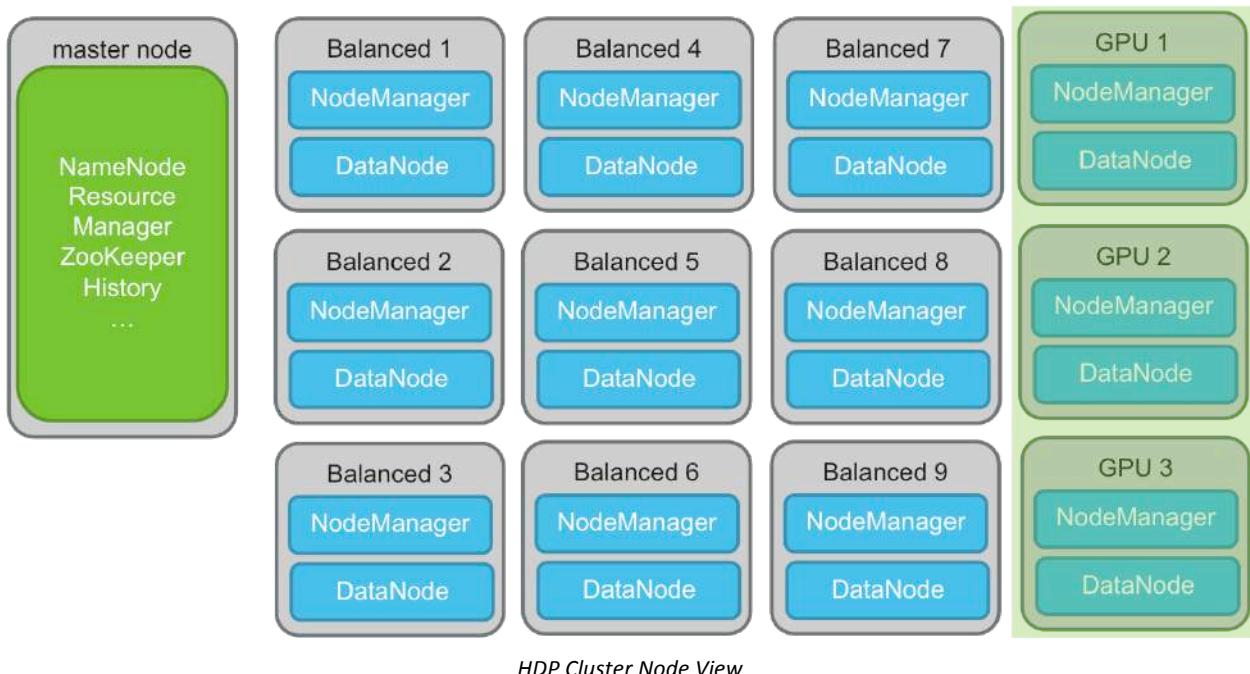
Assume three organizations currently access the HDP cluster: Engineering, Marketing, and Support.

The queues have been configured to meet SLAs so that:

- Engineering is guaranteed access to 60% of cluster resources upon demand
- Marketing is guaranteed access to 30% of cluster resources on demand
- Support is guaranteed access to 10% of resources upon demand

All queue maximums have been set so that any organization can use up to 80% of cluster resources if they are currently available.

The HDP Cluster – Node View



HDP Cluster Node View

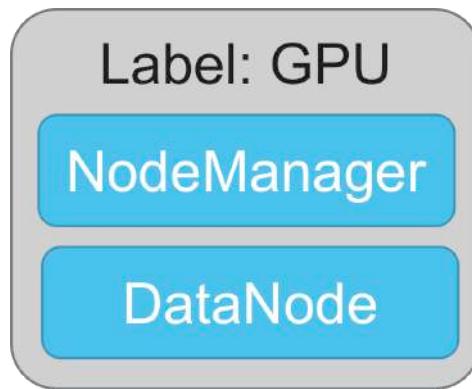
The HDP cluster has 12 worker nodes. Nine of them are balanced nodes with no special characteristics and three of them have been configured with GPUs (paid for by Marketing). All nodes have the same number of CPU cores and the same amount of memory installed.

Requirements

The requirements for managing the cluster are as follows:

- Creating separate Hortonworks clusters for the GPU-enabled machines is not an acceptable way to manage access to the machines. Requiring multiple logins to separate clusters as a way to separate resources requires extra management servers and is sub-optimal from a user experience perspective.
- The Marketing group has developed specialized YARN and MapReduce jobs that require exclusive access to nodes with GPUs. All other jobs should run on standard balanced nodes.

YARN Node Labels



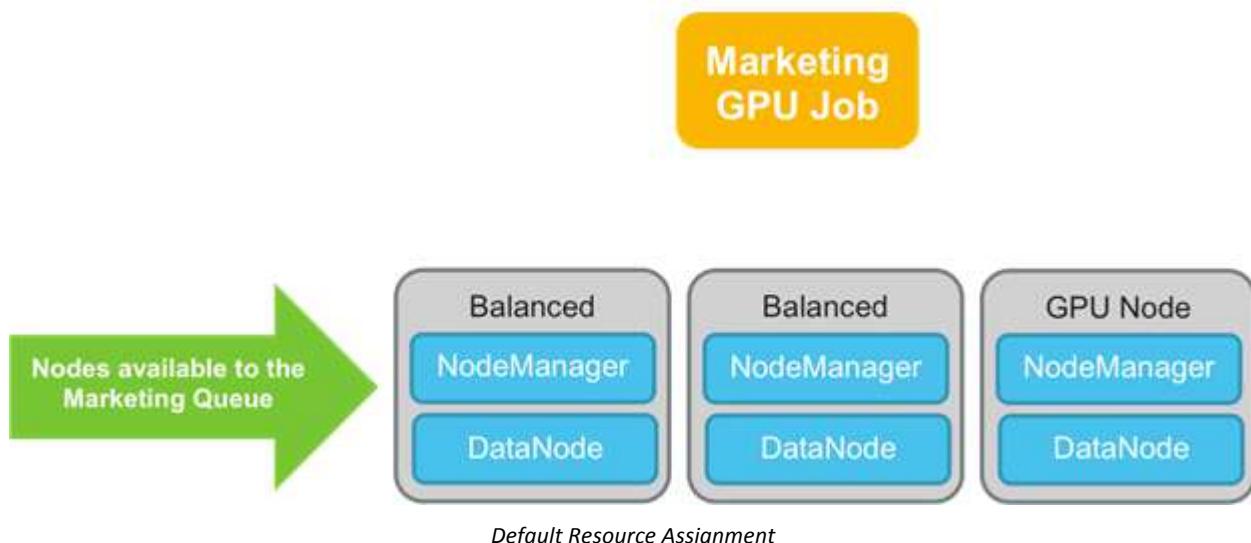
YARN node labels allow an HDP administrator to separate out, or partition, cluster resources into a sub-cluster, or sub-queue, inside a single HDP cluster. They allow YARN jobs to run on a certain subset of physical nodes.

Node labels are enabled in the Ambari Web UI and created and attached to nodes via a `yarn rmadmin` CLI commands. Once configured, labeled node resources are assigned to and divided between queues via the YARN Queue Manager View.

Creating Node Labels

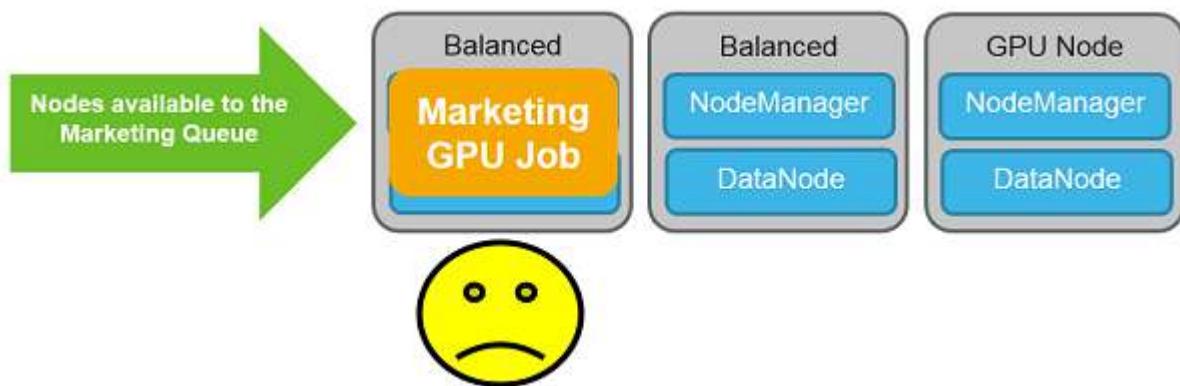
Node labels can affect the way certain types of jobs are assigned in an HDP cluster.

Node Assignment Without Labels



YARN Node Labels

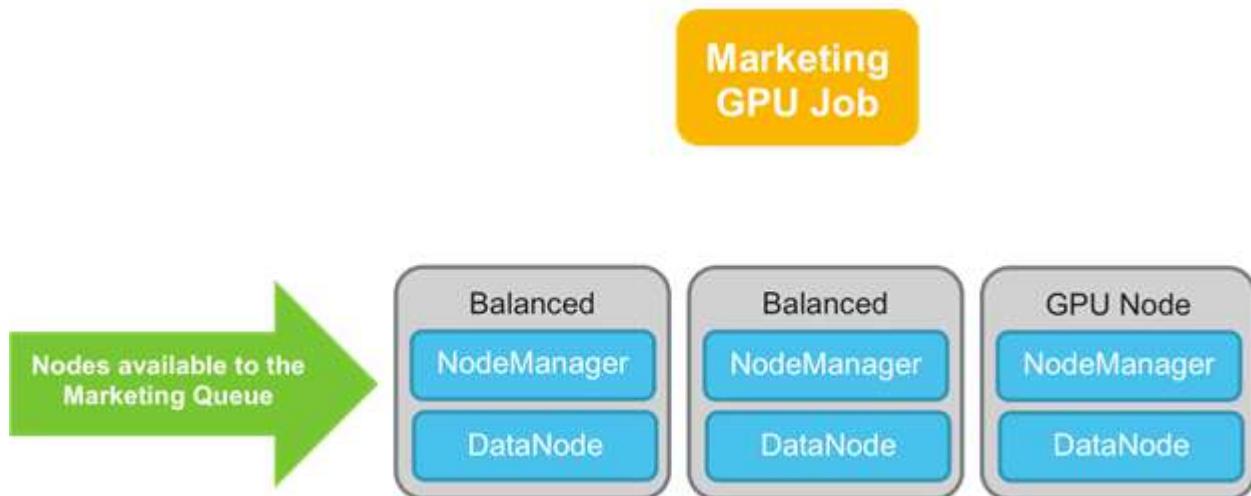
By default, job resources are allocated based on amount of memory and potentially number of CPU cores, but underlying hardware is not a consideration. In the scenario pictured, a user in the Marketing group is about to run a job that would benefit from running on a node with GPUs configured. However, only one out of three available nodes are configured with GPUs. The YARN resource manager only tracks amount of memory available, and potentially number of CPU cores, but has no awareness of other hardware considerations. Assuming the amount of memory and number of CPU cores are equivalent across the nodes, the job has an equal chance of being assigned to any available node.



Default Allocation Provides no Guarantee of Appropriate Assignment of a Job

Thus, the probability that the GPU-specific job will be assigned to a node with GPU capabilities is 33 1/3%. More often than not, this will result in this job being assigned to the wrong node type.

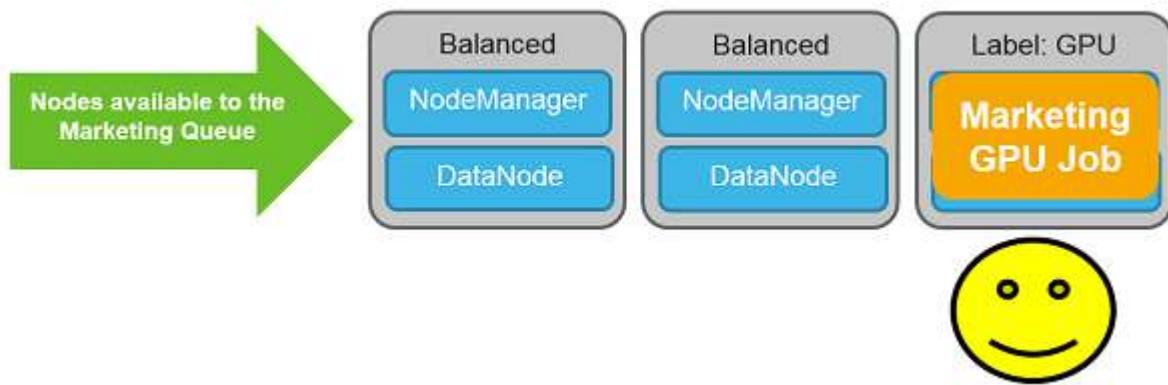
Node Assignment with Labels



Node Labels Enable Jobs Being Made to Run on Specific Nodes

YARN Node Labels

When node labels are enabled, configured, and assigned to queues, jobs in a given queue can be made to run only on nodes with a specific label. For example, a node with GPU capabilities might be given a label, in our example “GPU.” Assuming this label’s resources have been assigned to the queue the user will use (and if this is a MapReduce job, the node label has been set as default), the job will run on the labeled nodes.



With a Node Label, the YARN Job is Assigned to an Appropriate Queue

Adding a Label to a Node

The screenshot shows the Ambari Web UI under the Services > YARN > Configs tab. The left sidebar lists services like HDFS, MapReduce2, YARN, Tez, Hive, Pig, Sqoop, ZooKeeper, Flume, and Ambari Metrics. The main area shows a list of configurations for the YARN Default group, with V19 selected. Below this, the "Advanced" tab is selected. On the right, the "YARN Features" section is visible, containing "Node Labels" which is currently set to "Enabled". A red oval highlights the "Node Labels" section.

Enabling Node Labels

YARN node labels are enabled in the Ambari Web UI under **Services > YARN > Configs > Settings > YARN Features**. Simply click the Enabled button, save your configuration change, make notes, and then restart services as prompted.

Behind the scenes, Ambari is performing the following steps:

- Creates the /system/yarn/node-labels directory in HDFS
- Sets ownership of that directory to yarn:hadoop
- Sets 700 (drwx-----) permissions on the /system/yarn directory so that only the yarn user can access the .../node-labels directory.
- Sets the following properties and values in /etc/hadoop/conf/yarn-site.xml:
 - yarn.node-labels.enabled = true
 - yarn.node-labels.fs-store.root-dir = /system/yarn/node-labels

Creating Node Labels

Node labels are created at the command line using the following command:

```
yarn rmadmin -addToClusterNodeLabels <label>
```

This is an administrative command, therefore an appropriate user must be logged in to run it. Typically, this means you must log in as the built-in yarn user prior to running the command.

Multiple labels can be specified at one time via a comma-separated list surrounded by quotation marks:

```
yarn rmadmin -addToClusterNodeLabels "<label1>,<label2>,<label3>"
```

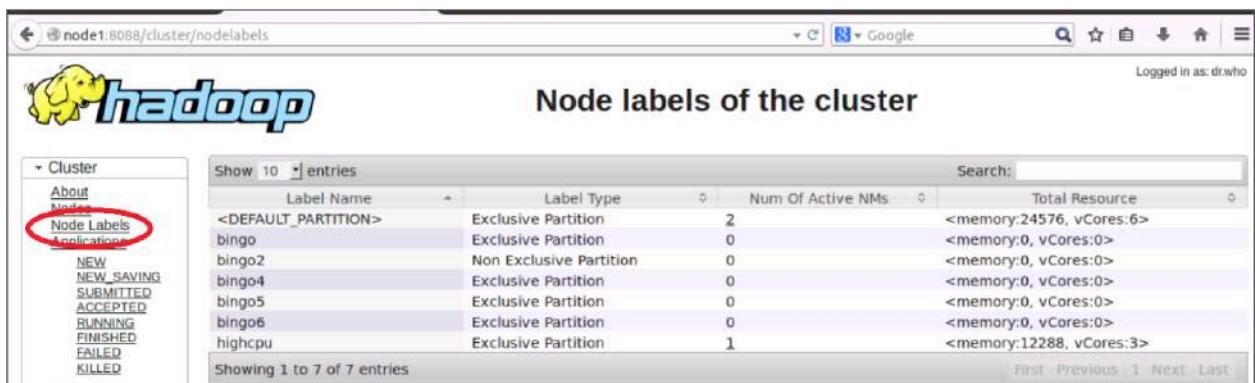
Modifying or Removing Node Labels

To view node labels and their exclusivity settings, use the yarn cluster --list-node-labels command from the CLI.

```
[hdfs@node1 hadoop-yarn-client]# yarn cluster --list-node-labels
15/11/03 17:11:26 INFO impl.TimelineClientImpl: Timeline service address: http://node1:8188/ws/v1/timeline/
15/11/03 17:11:27 INFO client.RMProxy: Connecting to ResourceManager at node1/172.17.0.2:8080
Node Labels: <bingo5:exclusivity=true>,<bingo6:exclusivity=true>,<bingo4:exclusivity=true>,<bingo2:exclusivity=false>,<highcpu:exclusivity=true>,<bingo:exclusivity=true>
```

This is a user command, so any user can execute it.

Alternatively, node labels and settings can be viewed via the ResourceManager UI by clicking on the **Node Labels** link.



Label Name	Label Type	Num Of Active NMs	Total Resource
<DEFAULT_PARTITION>	Exclusive Partition	2	<memory:24576, vCores:6>
bingo	Exclusive Partition	0	<memory:0, vCores:0>
bingo2	Non Exclusive Partition	0	<memory:0, vCores:0>
bingo4	Exclusive Partition	0	<memory:0, vCores:0>
bingo5	Exclusive Partition	0	<memory:0, vCores:0>
bingo6	Exclusive Partition	0	<memory:0, vCores:0>
highcpu	Exclusive Partition	1	<memory:12288, vCores:3>

Node Labels link in the ResourceManager UI

Note

It is possible to set a node label's exclusivity setting, which in theory would allow labeled nodes to be used by non-labeled applications if exclusivity was set to false. However this node label feature is experimental and as of the time of this writing, there was no behavioral difference between exclusive and non-exclusive node label partitions.

Removing Node Labels

Node labels that have not yet been associated with a queue can be removed at the command line using the following command:

```
yarn rmadmin -removeFromClusterNodeLabels <comma-separated list of labels>
```

In the example above, the command was used to remove labels bingo, bingo2, bingo4, bingo5, and bingo6. Quotation marks around the comma-separated list are unnecessary.

Confirming Successful Operation

```
[yarn@node1 root]# yarn rmadmin -removeFromClusterNodeLabels bingo,bingo2,bingo4,bingo5,bingo6
15/11/03 17:49:22 INFO client.RMProxy: Connecting to ResourceManager at node1/172.17.0.2:8141
[yarn@node1 root]# yarn cluster --list-node-labels
15/11/03 17:49:43 INFO impl.TimelineClientImpl: Timeline service address: http://node1:8188/ws/v1/timeline/
15/11/03 17:49:43 INFO client.RMProxy: Connecting to ResourceManager at node1/172.17.0.2:8050
Node Labels: <highcpu:exclusivity=true>
[yarn@node1 root]#
```

Cluster					
	Label Name	Label Type	Num Of Active NMs	Total Resource	Search:
<DEFAULT_PARTITION>	Exclusive Partition	2		<memory:24576, vCores:6>	
highcpu	Exclusive Partition	1		<memory:12288, vCores:3>	
Showing 1 to 2 of 2 entries					
First Previous 1 Next Last					

Confirming a Successful Operation with CLI and ResourceManager UI

Successful operation can be confirmed from the CLI using the `yarn cluster --list-node-labels` command or by clicking or refreshing the Node Labels link in the ResourceManager UI.

Assigning and Removing a Node Label

Use the following command to assign a label to a node:

```
yarn rmadmin -replaceLabelsOnNode <node FQDN>=<preconfigured label>
```

For example, in our lab environment since FQDNs are not used, we could attach the `highcpu` label to `node3` with the following command:

```
yarn rmadmin -replaceLabelsOnNode node3=highcpu
```

Multiple node label assignments can be made by creating a space-separated (*not* comma-separated) list enclosed by quotation marks

```
yarn rmadmin -replaceLabelsOnNode "<node1>=<label> <node2>=<label>"
```

For example, to attach the `highcpu` label to both `node2` and `node3` in our lab environment, execute the following command:

```
yarn rmadmin -replaceLabelsOnNode "node2=highcpu node3=highcpu"
```

To remove a label from a node, execute the same command but do not specify a label

```
yarn rmadmin -replaceLabelsOnNode <node FQDN>
```

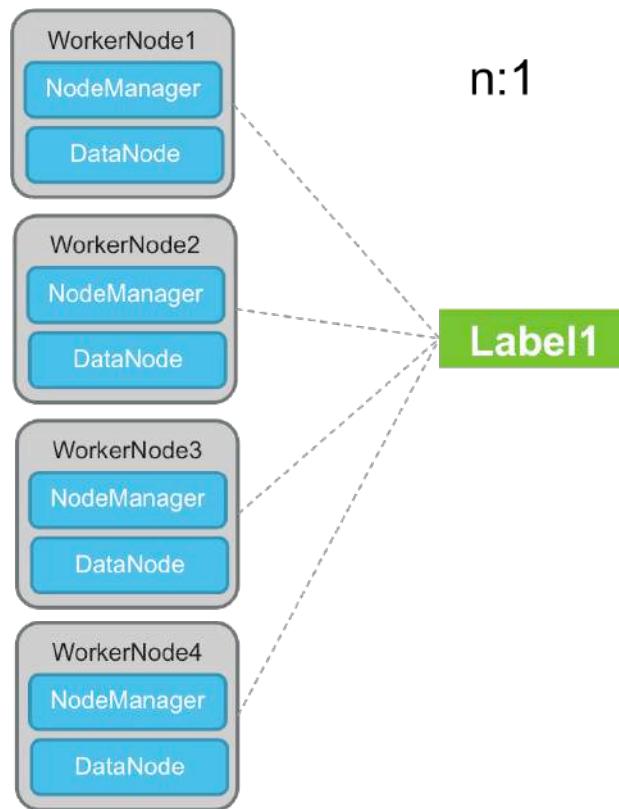
For example, to remove the highcpu label from node2, execute the following command:

```
yarn rmadmin -replaceLabelsOnNode node2
```

Again, multiple labels can be removed by creating a space-separated list of nodes enclosed by quotation marks. For example, to remove the highcpu label from both node2 and node3, execute the following command:

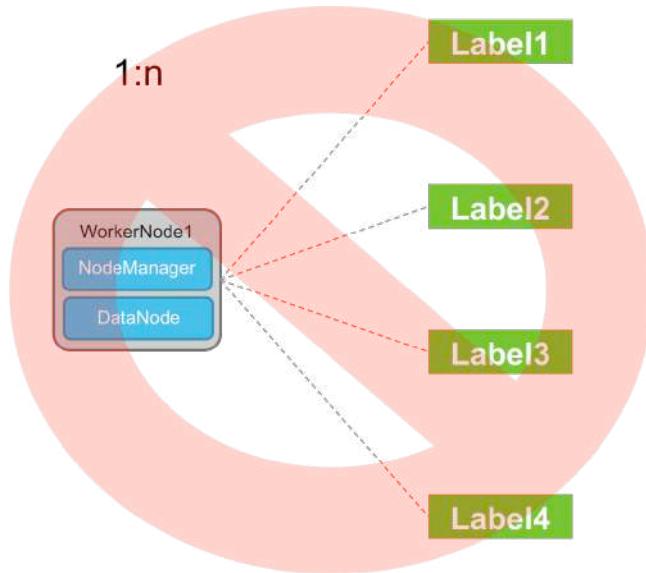
```
yarn rmadmin -replaceLabelsOnNode "node2 node3"
```

Node and Label Relationships



n:1 Relationship Between Worker Node and Labels

There is an n:1 relationship between worker nodes and labels, meaning that multiple worker nodes can be assigned to a single label. For example, in the scenario where there were four nodes that had GPU capabilities, a label named GPU could be created and all four of those nodes could be attached to that label.



Unsupported: Multiple Labels to a Single Node

Assigning multiple labels to a single node is currently unsupported.

Configuring Queues to Access Node Label Resources

Node labels must first be configured and enabled in the Root Queue. Once that is done, node labels can be assigned to individual queues.

Enabling Node Labels in the Root Queue

root (100%)

default (0%)

Engineering (60%)

Marketing (30%)

Support (10%)

root

Capacity: 100 %

Max Capacity: 100 %

Enable node labels

Enabling Node Labels in the Root Queue

Node labels must first be enabled in the root queue. Open the YARN Queue Manager View in Ambari UI, select the root queue, and click the Enable node labels button on the right.

Configuring Node Label Capacity

The screenshot shows the YARN Node Labels configuration interface. On the left, there's a sidebar with 'Add Queue' and 'Actions' buttons. Below them is a list of queues: 'root (100%)' (red), 'default (0%)' (orange), 'Engineering (60%)' (green), 'Marketing (30%)' (light green), and 'Support (10%)' (light green). The 'root' queue is selected. On the right, the 'root' configuration panel shows 'Capacity' and 'Level Total' at 100%. Under 'root', there's a 'GPU' section with 'Capacity: 0 %' and 'Max Capacity: 100 %'. A red circle highlights the 'Enable node labels' checkbox and the green 'Node Labels Access' button.

Configuring Node Label Capacity in the Root Queue

After clicking the Enable node labels button, next click the asterisk button labeled Node Labels Access. When it turns green, a new window will appear which will allow you to set the node label capacity available to the cluster.

Setting Queue Capacity

The screenshot shows the 'root' configuration panel again. The 'GPU' section has 'Capacity: 100 %' and 'Max Capacity: 100 %'. A red circle highlights the 'Capacity' slider. To the right, there's a 'Use by default' button.

Queue Capacity Set to 100%

Set Capacity at the root queue to 100%. Now the resources with this node label can be assigned to your queues.

Assigning Labels to Individual Queues

The screenshot shows the 'Marketing' queue configuration. The 'Marketing' queue is highlighted in blue. On the right, the 'Marketing' configuration panel shows 'Capacity' and 'Level Total' at 100%. Under 'Marketing', there's a 'GPU' section with 'Capacity: 30 %' and 'Max Capacity: 81 %'. A red circle highlights the 'Enable node labels' checkbox.

Enabling Node Labels in Queues

Click on the queue that you want to assign this node label to – in our case, the Marketing queue – and click the Enable node labels button.

Configuring Labels

The screenshot shows the YARN Queue Manager interface. On the left, there's a tree view of queues: root (100%), default (0%), Engineering (60%), Marketing (30%, selected), and Support (10%). Below this is a Scheduler section with a maximum value of 10000. On the right, the Marketing queue is expanded. It shows a capacity of 100% for the Marketing label. There's also a GPU label configuration with a capacity of 100% and a radio button for 'Use by default'. A 'Node Labels Access' asterisk is present, which when clicked, opens a configuration window titled 'Marketing'. This window shows the Marketing queue details with a checked 'Enable node labels' checkbox. It also displays the GPU label configuration with a capacity of 100% and a radio button for 'Use by default'.

Configuring Labels in an Individual Queue

Click the Node Labels Access asterisk to make the node label configuration window appear. Then set GPU capacity to appropriate levels (in our case, 100%) and select Use by default – especially if you want MapReduce jobs that are configured to run on this queue to run on labeled nodes. While YARN jobs can specify both queue and node label, MapReduce jobs can only specify queue. Therefore, if a default node label is not configured, MapReduce jobs will not run on labeled nodes. .

Once you save your changes and refresh the queue settings, jobs that are configured to run in the Marketing queue will be automatically directed to nodes with the GPU label.

Undoing Node Labels

A couple of caveats regarding node labels:

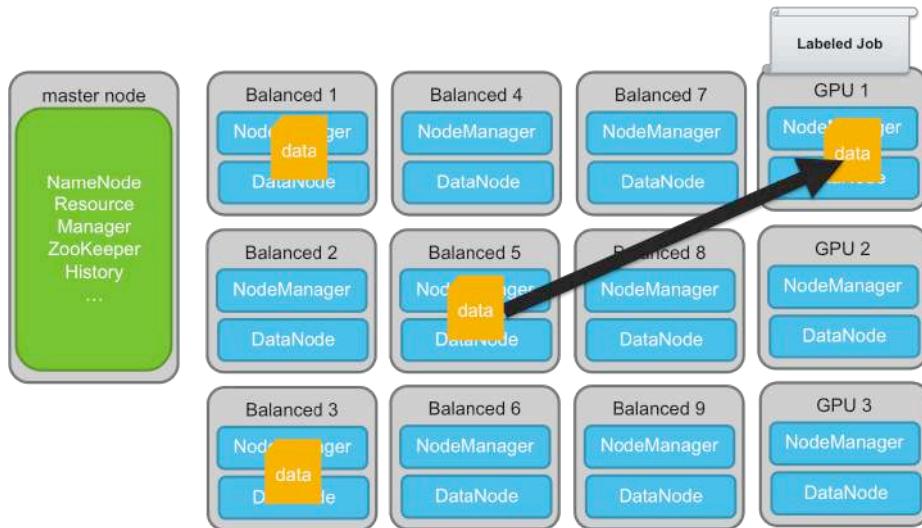
If you disable use of node labels, remember to disable them in the root queue before saving your queue configuration and restarting YARN. If you disable all of the leaf queues but not the root queue, YARN ResourceManager will immediately shut down after restart.

Also, always check YARN service after changing node label settings. Sometimes it needs a restart and may not tell you in the YARN Queue Manager interface. You may find it helpful to have two browser windows open – one set to the YARN Queue Manager View, and the other set to the YARN service in the Ambari Web UI.

YARN Node Labels

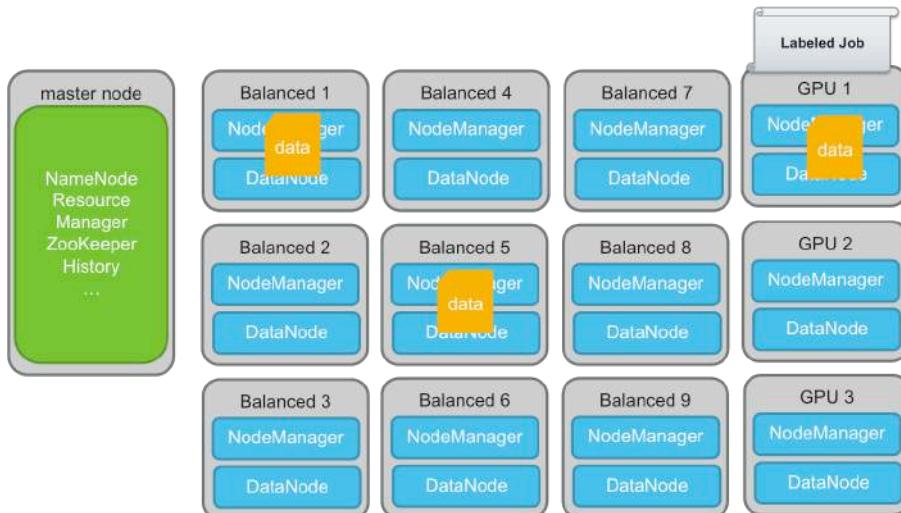
Node Labels and Data Locality

One significant caveat to using node labels is that they only affect the location of processing of data. They have no bearing on where data is written in the first place.



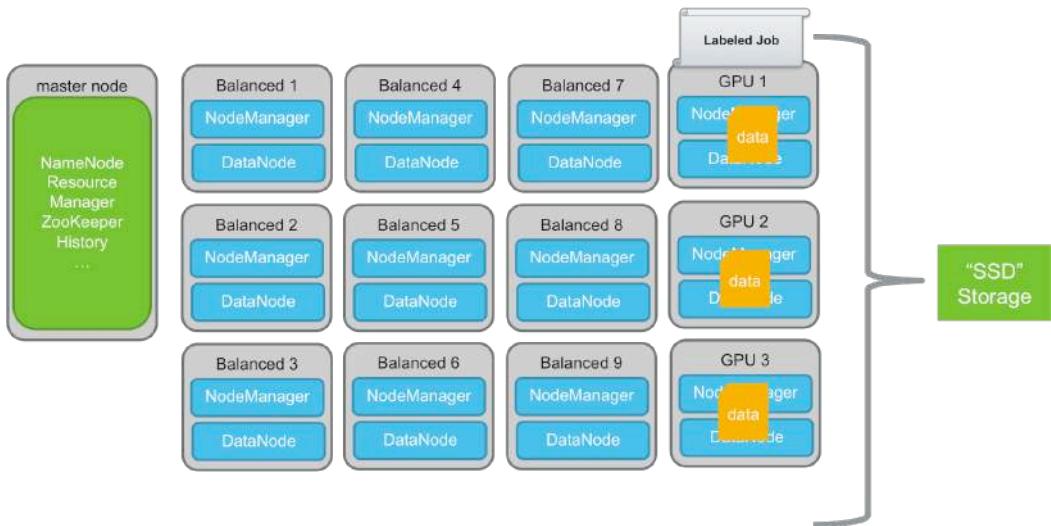
For example, assume someone has loaded a large amount of data into HDFS, but not specifically to GPU-labeled nodes. Later, a user decides to run a job that uses that data and forces the job to run on a labeled node. When that job runs, the first thing that has to happen is the data has to be transferred across the network to the labeled node, which might have a significant impact on performance.

There are a number of variables that can affect performance in this scenario, including the speed of the network connection.



How the data was loaded can matter as well – for example, if it was initially copied to a GPU-labeled node as part of a job, at least one copy will exist on *that* node, which will then be the preferred node when a GPU-labeled job is executed again using that data.

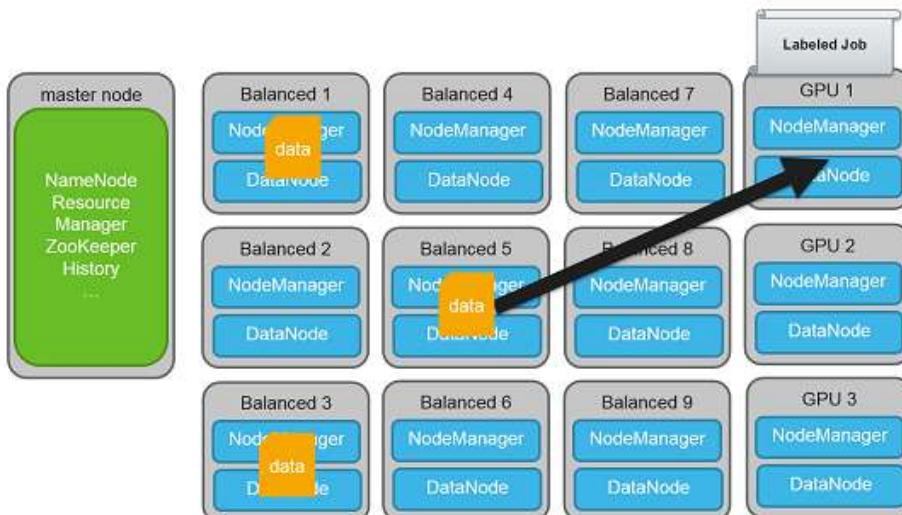
YARN Node Labels



An administrator might choose to use some combination of heterogeneous storage settings and node labels to force data locality (for example: setting policies so that certain user's data all get written to "SSD" nodes – which may actually be running regular spinning disks, but also happen to be the GPU-labeled nodes.) This workaround would be limited to just one or two node labels, so would not be appropriate for a complex labeled environment.

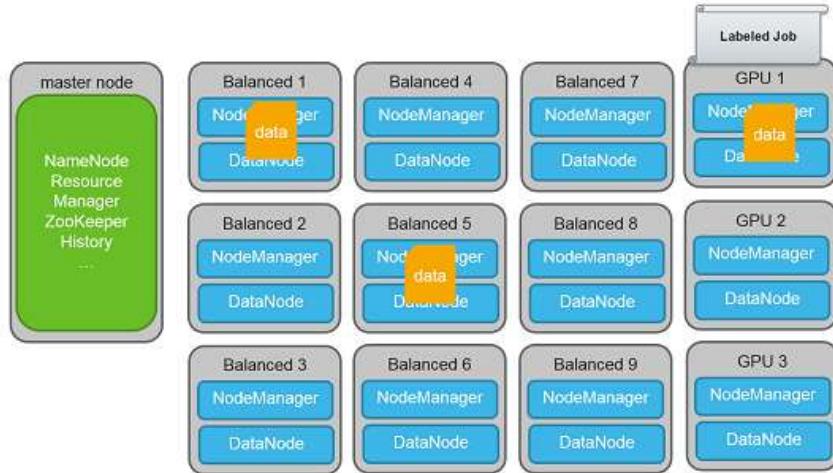
Node Labels and Data Locality

One significant caveat to using node labels is that they only affect the location of processing of data. They have no bearing on where data is written in the first place. For example, assume someone has loaded a large amount of data into HDFS, but not specifically to GPU-labeled nodes. Later, a user decides to run a job that uses that data and forces the job to run on a labeled node. When that job runs, the first thing that has to happen is the data has to be transferred across the network to the labeled node, which might have a significant impact on performance.

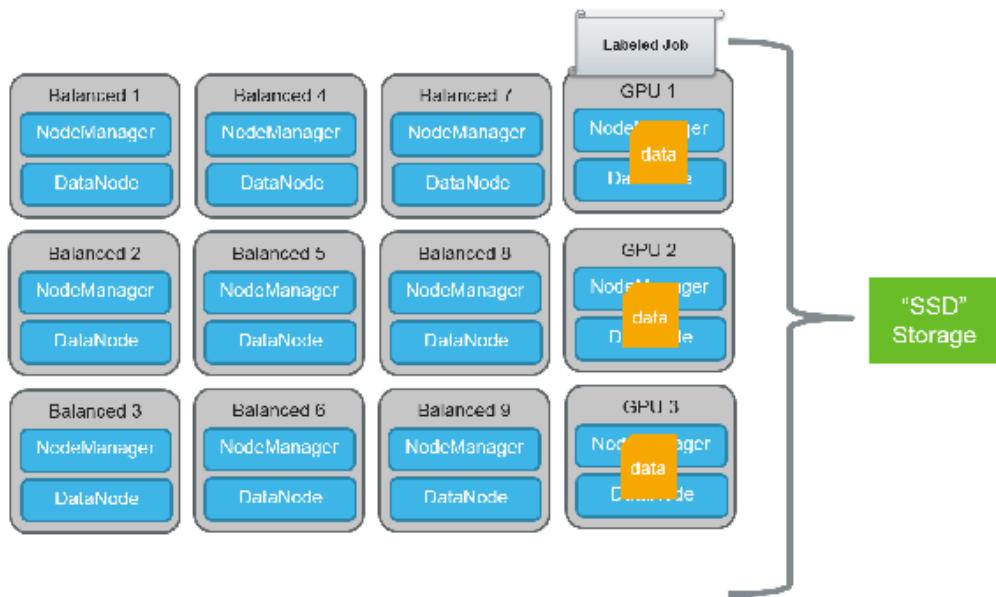


YARN Node Labels

There are a number of variables that can affect performance in this scenario, including the speed of the network connection. How the data was loaded can matter as well – for example, if it was initially copied to a GPU-labeled node as part of a job, at least one copy will exist on *that* node, which will then be the preferred node when a GPU-labeled job is executed again using that data.



An administrator might choose to use some combination of heterogeneous storage settings and node labels to force data locality (for example: setting policies so that certain user's data all get written to "SSD" nodes – which may actually be running regular spinning disks, but also happen to be the GPU-labeled nodes.)



This workaround would be limited to just one or two node labels, so would not be appropriate for a complex labeled environment.

Knowledge Check

Questions

- 1) True or False: A YARN job assigned to a queue with a default node label will always be assigned to a node with that label.
- 2) True or False: A MapReduce job assigned to a queue with a default node label will always be assigned to a node with that label.
- 3) True or False: A single label can be applied to multiple nodes.
- 4) True or False: A single node can have multiple labels applied to it.

Answers

- 1) True or False: A YARN job assigned to a queue with a default node label will always be assigned to a node with that label.

Answer: True

- 2) True or False: A MapReduce job assigned to a queue with a default node label will always be assigned to a node with that label.

Answer: True

- 3) True or False: A single label can be applied to multiple nodes.

Answer: True

- 4) True or False: A single node can have multiple labels applied to it.

Answer: False

Summary

- Node labels allow the creation of an isolated set of nodes within a larger HDP cluster
- Multiple nodes can be associated with a single label
- A single node can only have one label
- Jobs assigned to a queue with a default node label configured will always run on nodes with that label

Deploying Applications using Apache Slider

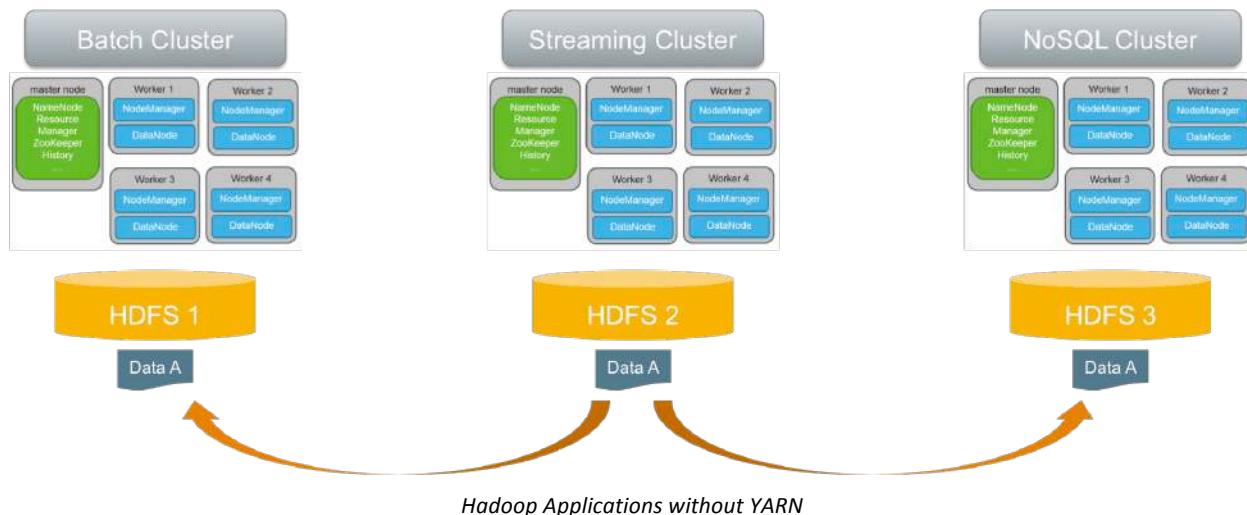
Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Recall the purpose, benefits, and components of Slider
- ✓ Install and manage Slider as a cluster-level service in Ambari
- ✓ Deploy and manage a Slider application package using the Slider View

About Apache Slider

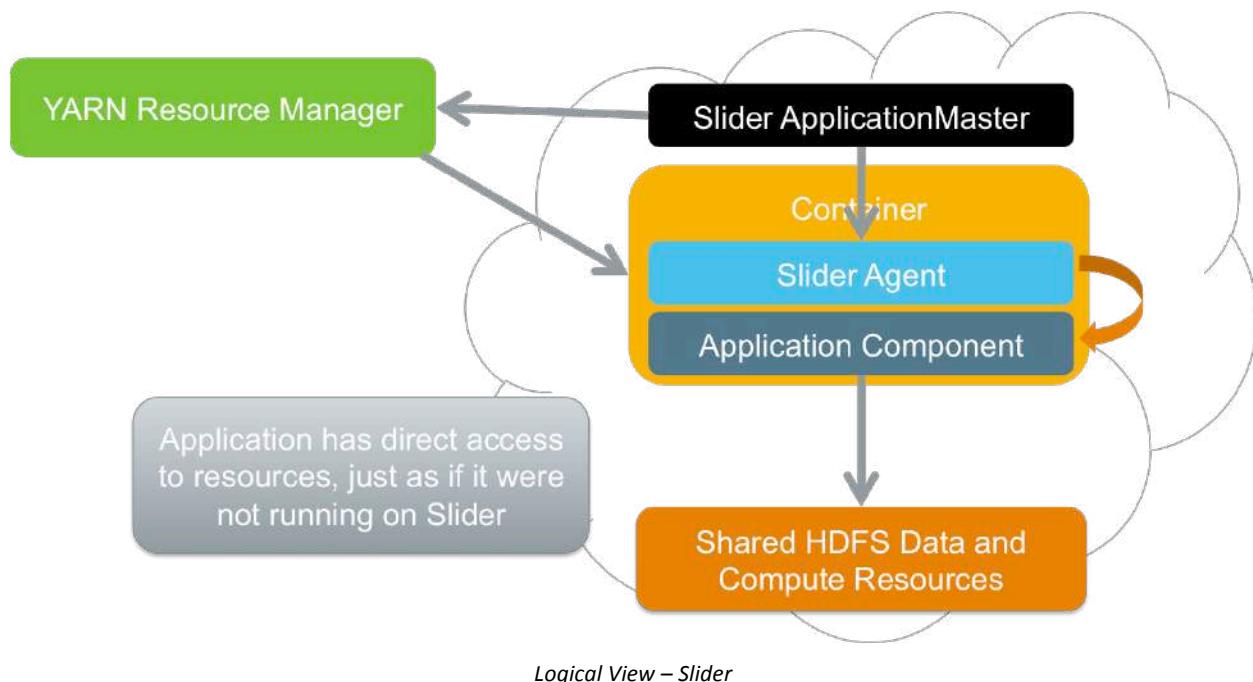
Up until now, this course and the previous course have simply assumed that HDP applications run natively on YARN. Many Hadoop applications, however, were not built to integrate with the YARN API, and instead are designed to access HDP resources directly when running. Without a central resource manager to ensure good behavior between applications, it is often necessary to create specialized, separate clusters for a given application or project. This, in turn, means that when you want to do something different with the data that application was using, it is necessary to copy that data between clusters.



YARN as a resource manager mitigates this issue by allowing different types of applications to access the same underlying resources pooled into a single data lake. However, for an application to safely run in a shared resources environment, it must be written to conform to the YARN API. In many cases, the application may need to be rewritten completely, or perhaps heavily modified, in addition to the work required to build out additional components.

Lacking the resources to convert an application to run natively on YARN, a company might still be forced to create multiple HDP clusters to support non-YARN applications, and manage the complexity around that. Apache Slider can make this unnecessary.

Slider Logical View



Apache Slider is a collection of client libraries that allows an application designed to run directly on Hadoop to be managed via YARN. It acts as a proxy between an application and the YARN resource manager. The application components still interact directly with Hadoop, so Slider is not performing any virtualization or emulation functions. Thus, there are no direct performance impacts to running an application using Slider compared to running it without.

Slider Components

ApplicationMaster

The Slider ApplicationMaster handles integration with external services, specifically YARN ResourceManager and NodeManager components. This can be used when no special placement or scheduling mechanisms are required by the application. If additional requirements are necessary, the application must be implemented as a native YARN application.

Agent

The Slider agent is a lightweight service that executes commands to instantiate application components. It also monitors and reports on application component status to the Slider ApplicationMaster.

Application Components and Data

An application component is one of one or more parts that make up a service provided by an application. For example, the HBase application is comprised of a master server, one or more region servers, and other components. Each of these would be launched in a separate container, and each would be monitored by the Slider agent.

Key Slider Benefits

The fact that YARN manages resources and monitors performance allows a non-YARN application to run alongside other applications of different types. This means they can share both data and compute resources while being protected by the YARN framework. This also provides non-YARN applications with the application and component recovery capabilities that YARN applications enjoy.

Packaging an application to run on Slider is relatively simple compared to rewriting it to run natively on YARN. It is also easier to develop against than YARN when writing new applications. Thus, it can be a powerful tool to speed up development and improve ROI across various HDP-based initiatives.

Cluster-Level Administration

This section will discuss how to deploy and manage Slider from an administrative perspective.

User Roles and Administrative Scenario

- Application Developer
 - Can package Slider with an application, no administrator assistance required
 - Needs to understand how to install Slider on the desktop and manage configuration files
 - Can build applications designed to leverage Slider on HDP

Slider Application Developer Roles

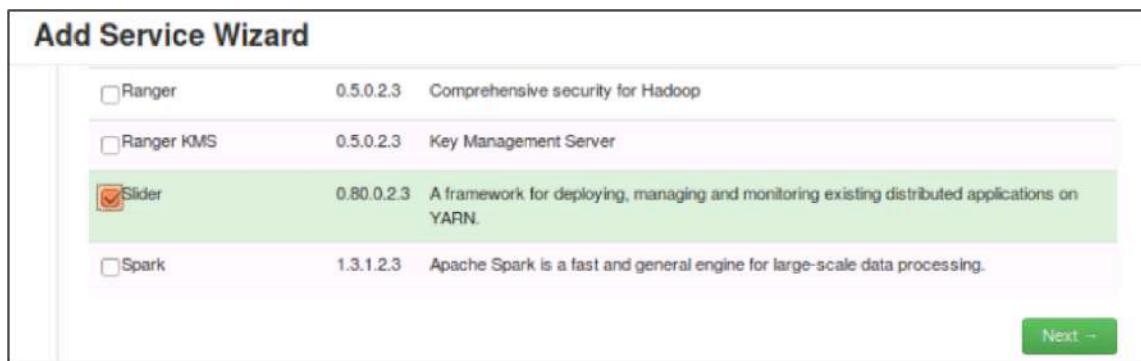
From a developer's perspective, Slider is a tool to aid in application development and deployment. Nothing is required to be installed on the HDP cluster itself in order to run a Slider application. An application developer can simply package the Slider client libraries with the application they have developed, and they will be installed and executed by YARN just like any other application. From YARN's perspective, there is no difference between running an application managed by Slider and an application written to communicate directly with the YARN APIs. This allows the deployment of applications that might be running on different versions of Slider, as well as the ability to run multiple versions of the same application, all on the same shared resources accessing the same data.

*Slider Administrator Roles*

From an administrator's perspective, then, there is nothing that must be done to enable Slider and Slider-based applications to run on the HDP cluster. However, in some cases it can be useful to provide an interface for non-developers to be able to deploy Slider applications. For example, you might want to enable a database administrator (DBA) to deploy HBase on your HDP cluster, without requiring them to do any application packaging on their own. With Slider deployed as a cluster-level service, you can provide the ability to deploy and manage Slider-enabled applications to any HDP user.

That scenario will be the focus of this lesson.

Cluster-Level Slider Deployment

*The Add Service Wizard*

When choosing services to add to a cluster, Slider is one of the installable options via the Add Service Wizard. This installs the Slider client libraries in the standard HDP service library at the following directory: `/usr/hdp/current/slider-client/`.

Once installed, a user has the ability to interact with Slider at the cluster level from the command line and use it to run and manage Slider applications.

Slider View in Ambari

The screenshot shows the 'Slider View' page in Ambari. At the top, there are filters for Name, Status, Type, User, Start Time, and End Time. The results table shows one application: 'hbase-test-1' with status 'ACCEPTED', type 'HBASE (1.1.2.2.3.2.0-2950)', user 'yarn', start time 'Fri, 20 Nov 2015, 16:13:49 -05:00 GMT', and end time '-'. Below the table, it says '1 of 1 slider apps showing - clear filters'. At the bottom right, there are buttons for 'Show' (set to 10), '1 - 1 of 1', and navigation arrows.

Slider View in Ambari

The second way to enable Slider at the cluster level is by installing and configuring the Slider View in Ambari. This installs the same client libraries, but in a different location: /var/lib/ambari-server/resources/views/work/SLIDER{2.0.0}/WEB-INF/lib/.

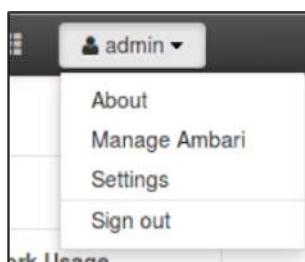
Installing the Slider View, and storing Slider-enabled applications in the appropriate directory, will allow an administrator to give HDP users access to run and manage those applications in an easy-to-use and easy-to-manage graphical user interface.

This will be the method described in this lesson.

Slider View Installation

Installing and configuring the Slider View is much like any other Ambari View installation.

- 1) Log in as a user with appropriate permissions and on that user's action button, select Manage Ambari.



- 2) Then select Views > Slider > Create Instance.

The screenshot shows the 'Views' section of the Ambari interface. On the left, there is a list of views, with 'Slider' being the selected item. On the right, the 'Slider' view is displayed, showing the version '2.0.0 (0)' and a large blue 'Create Instance' button.

- 3) Configure the Instance Name, Display Name, and Description fields, as well as the Ambari server URL and a username/password that has appropriate permissions on the Ambari instance this View should manage.

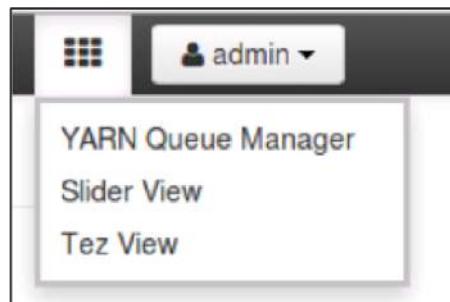
The screenshot shows the 'Slider' view configuration screen. On the left, there's a 'Details' section with fields for 'Instance Name', 'Display Name', and 'Description'. Below these is a 'Visible' checkbox. On the right, there's a 'Permission' section titled 'Grant permission to these users' with a 'Use' dropdown set to 'admin' and a checked checkbox next to it.

- 4) In addition, a Slider user account can be specified, but by default the yarn user will be used to launch and manage Slider applications. In order for this to work, the yarn user must have a home folder configured in HDFS with permissions set to make it accessible.

```
[root@node1 scripts]# su hdfs
[hdfs@node1 scripts]# hdfs dfs -mkdir /user/yarn
```

```
[hdfs@node1 scripts]# hdfs dfs -chown yarn:yarn /user/yarn
[hdfs@node1 scripts]# hdfs dfs -ls /user/
Found 4 items
drwxrwx---  - ambari-qa hdfs      0 2015-11-20 15:27 /user/ambari-qa
drwxr-xr-x  - hcat    hdfs      0 2015-11-20 15:28 /user/hcat
drwx-----  - hive    hdfs      0 2015-11-20 15:28 /user/hive
drwxr-xr-x  - yarn    yarn      0 2015-11-20 15:47 /user/yarn
```

- 5) Once the View is saved, choose the groups or users that should have access to it and make sure that the correct HDFS home directory has been created and permissions set. Appropriate users should now be able to log into the Ambari Web UI and access the Slider View from the Views menu.



Slider Packages

Slider packages are zip files that contain the application plus two additional configuration files Slider needs in order to manage the application. They are:

- resources.json
- app_config.json

resources.json

```
{
  "schema": "http://example.org/specification/v2.0.0",
  "global": {
    "yarn.memory": "512"
  },
  "components": {
    "HBASE_MASTER": {
      "yarn.role.priority": "1",
      "yarn.component.instances": "1",
      "yarn.vcores": "1"
    },
    "HBASE_REGIONSERVER": {
      "yarn.role.priority": "2",
      "yarn.component.instances": "1"
    }
  }
}
```

Slider Package – resources.json

The first package file is the `resources.json` file, which identifies such things as the schema definition, global YARN memory and log aggregation settings, and a list of the application components along with configuration information such as their relative priority, number of instances to provision at launch, node labels, and number of vcores to request.

app_config.json

```
{
  "application.def": "/slider/hbase_v096.zip",
  "site.global.app_log_dir": "${AGENT_LOG_ROOT}/app/log",
  "site.global.app_pid_dir": "${AGENT_WORK_ROOT}/app/run",
  "site.global.hbase_master_heapsize": "1024m",
  "site.global.ganglia_server_host": "${NN_HOST}",
  "site.global.ganglia_server_port": "8667",
  "site.global.ganglia_server_id": "Application1",
  "site.hbase-site.hbase.tmp.dir": "${AGENT_WORK_ROOT}/work",
  "site.hbase-site.hbase.master.info.port": "${HBASE_MASTER_PORT}",
  "site.hbase-site.hbase.regionserver.port": "0",
  "site.hbase-site.hbase.zookeeper.quorum": "${ZK_HOST}",
  "site.core-site.fs.defaultFS": "${NN_URI}",
}
```

Slider Package – app_config.json

The second package file is the `app_config.json` file, which contains application-specific settings that are configured at application launch. This primarily includes application variables that need to be set before the application runs.

Deploying a Slider Package for the Slider View

Assuming a Slider package has been properly configured, deploying it for use with the Ambari Slider View is a straightforward process:

- 1) First, copy the zip file to the `/var/lib/ambari-server/resources/apps/` directory of the Ambari server.

```
root@ubuntu:~/sys/slider# scp slider-hbase-app-package-1.1.2.2.3.2.0-2950.zip root@node1:/var/lib/ambari-server/resources/apps/
slider-hbase-app-package-1.1.2.2.3.2.0-2950.z 100%   96MB  48.0MB/s   00:02
root@ubuntu:~/sys/slider#
```

- 2) Next, restart the Ambari server.

```
[root@node1 scripts]# ambari-server restart
Using python /usr/bin/python2.6
Restarting ambari-server
Using python /usr/bin/python2.6
Stopping ambari-server
Ambari Server stopped
Using python /usr/bin/python2.6
Starting ambari-server
Ambari Server running with administrator privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start.....
```

- 3) Then log into the Slider View, and the application will be available for users to deploy.

The screenshot shows a modal dialog titled "Select Application". It contains the following fields:

- Application Types:** A dropdown menu showing "HBASE (1.1.2.2.3.2.0-2950)".
- Description:** Text stating "Deploys HBASE (1.1.2.2.3.2.0-2950) cluster on YARN."
- Name:** An input field with placeholder text "Enter Name (required)".
- Enable Two-Way SSL:** A checkbox that is unchecked.

Deploying and Managing Slider Applications

To launch a Slider application:

- 1) Log into the Slider View and select the Create App button.



- 2) Choose the application to deploy from the Applications Types drop-down menu, and then provide a name for this instance.

Select Application	
Application Types	HBASE (1.1.2.2.3.2.0-2950)
Description	Deploys HBASE (1.1.2.2.3.2.0-2950) cluster on YARN.
Name	hbase-test-1

- 3) Additional configuration options will be loaded based on the JSON configuration files provided with the application.

HBASE (1.1.2.2.3.2.0-2950) application requires resources to be allocated on the cluster. Provide resource allocation requests for each component of the application below.				
Instances	Memory (MB)	CPU Cores	YARN Labels	
HBASE_MASTER	1	1500	1	<input type="checkbox"/>
HBASE_REGIONSERVER	1	1500	1	<input type="checkbox"/>
HBASE_REST	1	556	1	<input type="checkbox"/>
HBASE_THRIFT	1	556	1	<input type="checkbox"/>
HBASE_THRIFT2	1	556	1	<input type="checkbox"/>

← Back Next →

Provide configuration details for HBASE (1.1.2.2.3.2.0-2950) application				
<ul style="list-style-type: none"> General Global Hbase Env Hbase Site Custom 				
← Back Next →				

- 4) Verify all settings and click Finish to launch the application.

Summary	
App Name: hbase-test-1	
App Type: HBASE (1.1.2.2.3.2.0-2950)	
Two-Way SSL Enabled: false	
Components	
HBASE_MASTER: 1	
HBASE_REGIONSERVER: 1	
HBASE_REST: 1	
HBASE_THRIFT: 1	
HBASE_THRIFT2: 1	
Configuration	
<pre>"application.def":"./slider/package/HBASE/slider-hbase-app-package-1.1.2.2.3.2.0-2950.z "create.default.zookeeper.node":"true", "java_home":"/usr/jdk64/jdk1.8.0_40", "site_global.app.root": "%(AGENT_WORK_ROOT)/app/install/hbase-1.1.2.2.3.2.0-2950".</pre>	

- 5) Then return to the Slider View home page and the newly launched application instance will be listed.

The screenshot shows the 'Slider View' interface. At the top, there are filters for Name (Any), Status (All Status), Type (Any), User (Any), Start Time (All Dates), and End Time (All Dates). Below the filters, a table lists one application instance: 'hbase-test-1' with status 'ACCEPTED', type 'HBASE (1.1.2.2.3.2.0-2950)', user 'yarn', start time 'Fri, 20 Nov 2015, 16:13:49 -05:00 GMT', and end time '-'. At the bottom of the table, it says '1 of 1 slider apps showing - clear filters'. On the right side, there are buttons for 'yarn' and '+ Create App', and navigation links for 'Show 10' and '1 - 1 of 1'.

Flex Slider Applications

- Clicking on the name of the application instance will bring up the application Summary tab. Information about the application, including component status and node location, can be found here.

The screenshot shows the 'Slider View' interface with the 'Summary' tab selected for the application 'hbase-test-1'. The left panel displays the 'Summary' section with details such as Status (RUNNING), Type (HBASE (1.1.2.2.3.2.0-2950)), YARN Application ID (application_1448053690006_0001), Started (Fri, 20 Nov 2015, 16:13:49 -05:00 GMT), Finished (-), Diagnostics (-), Cluster ID (54b63ac5-a460-41a6-a403-06bab6bc7bf6), Cluster Requests (29), Dead Region 0 Servers, Is Active Master (true), Master Active Time (19h:1m), Master Start Time (Fri, 20 Nov 2015 21:14:39 GMT), Metric Average (2.0), Load, Region Servers (1), Server Name (node2,46345,1448054079334), and Zookeeper Quorum (node1:2181). The right panel displays the 'Status' section, which lists components: HBASE_MASTER, HBASE_REGIONSERVER, HBASE_REST, HBASE_THRIFT, HBASE_THRIFT2, and slider-appmaster, all marked as 1 out of 1 active. The 'Components' section shows the same components with their respective node locations: node2, node1, node3, node2, node1, and node2.

- In addition, clicking on the blue Actions button brings up a menu that allows an application to Flex or Stop.



- Flexing an application means configuring additional components. For example, if it became necessary to add another region server to the HBase instance in the example pictured, that could be easily accomplished by simply changing the number of instances and clicking the Save button.

Flex

Update the number of desired instances for each component of this application

Components	Instances
HBASE_MASTER	1
HBASE_REGIONSERVER	1
HBASE_REST	1
HBASE_THRIFT	1
HBASE_THRIFT2	1

Cancel **Save**

- The number of region servers would be increased on the HDP cluster accordingly.

<input checked="" type="checkbox"/>	HBASE_REGIONSERVER component	2 out of 2 active
<input checked="" type="checkbox"/>	HBASE_REGIONSERVER	node1
<input checked="" type="checkbox"/>	HBASE_REGIONSERVER	node2

Applications can also be flexed down if not all instances of a component are required to manage the current workload using the same method.

Slider Applications in ResourceManager UI

ID	User	Name	Application Type	Queue	Start Time	Finish Time	State	Final Status	Running Containers	Progress	Tracking UI	ApplicationMaster
application_1448053690006_0001	yarn	hbase-test-1	org.apache.slider	default	Fri Nov 20 16:13:49 -0500 2015	N/A	RUNNING	UNDEFINED	6			0

Slider Applications in the ResourceManager UI

Slider applications are just like any other YARN application from a management perspective. Therefore, information about Slider applications and components can also be viewed using the ResourceManager UI.

Stopping and Destroying a Slider Application

A Slider application can be stopped, in which case all components are shut down to free up cluster resources. However, the application's data remains, and is available to the application should it be restarted.

Stopping a Slider Application

If there is no need to store that data for future use, after stopping the application it can be destroyed.

Destroying a Slider Application

Deploying Applications using Apache Slider

Once the application has been destroyed, it will no longer be visible in the Slider View.

The screenshot shows the Apache Slider 'Slider View' interface. At the top, there are six filter fields: 'Name' (set to 'Any'), 'Status' (set to 'All Status'), 'Type' (set to 'Any'), 'User' (set to 'Any'), 'Start Time' (set to 'All Dates'), and 'End Time' (set to 'All Dates'). A blue button labeled '+ Create App' is located in the top right corner. Below the filters, a message says 'No slider apps to display'. At the bottom, a footer bar indicates '0 of 0 slider apps showing - [clear filters](#)' and includes a 'Show' dropdown set to '10', a page number '0 - 0 of 0', and navigation arrows.

This page left blank intentionally.

Knowledge Check

Use the following questions and answers to assess your understanding of the concepts presented in this lesson.

Questions

- 1) Slider allows an HDP application to be managed by _____ without the need to change application code.
- 2) In addition to the application, name the two components that make up a Slider application.
- 3) Name the two JSON files that an application developer must include in a Slider package.
- 4) Name the two ways to make Slider available as a cluster-level service.
- 5) What does it mean to flex a Slider application?
- 6) What is the difference between stopping and destroying a Slider application?

Answers

- 1) Slider allows an HDP application to be managed by _____ without the need to change application code.

Answer: YARN

- 2) In addition to the application, name the two components that make up a Slider application.

Answer: ApplicationMaster and Agent

- 3) Name the two JSON files that an application developer must include in a Slider package.

Answer: resources.json and app_config.json

- 4) Name the two ways to make Slider available as a cluster-level service.

Answer: Add it as a Service or install and configure the Slider View in Ambari

- 5) What does it mean to flex a Slider application?

Answer: Dynamically change the number of components

- 6) What is the difference between stopping and destroying a Slider application?

Answer: Stopping frees up resources, but retains application data which can be available if the application restarts. Destroying means all application-specific data is deleted.

Summary

- Apache Slider allows non-YARN applications to be managed by YARN, which allows them to safely run on HDP clusters that share resources and data with multiple types of applications
- Slider provides a default ApplicationMaster and one Slider agent per deployed application component
- While Slider can be deployed as a cluster-level service, a developer can also package Slider with an application, making administrative action unnecessary
- Slider can be deployed as a service on the cluster, enabling CLI interaction, or via the Ambari Slider View
- Slider application instances can be created, stopped, started, and destroyed

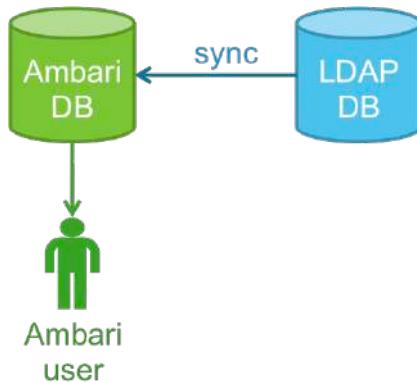
Integrating Ambari with LDAP

Lesson Objectives

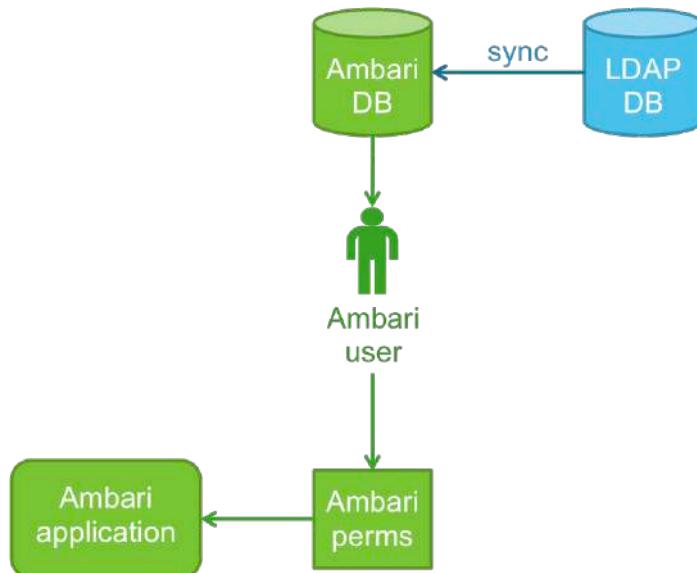
After completing this lesson, students should be able to:

- ✓ Identify characteristics of Ambari local versus LDAP users and groups
- ✓ Integrate Ambari Server with LDAP

Ambari User Sources and Permissions

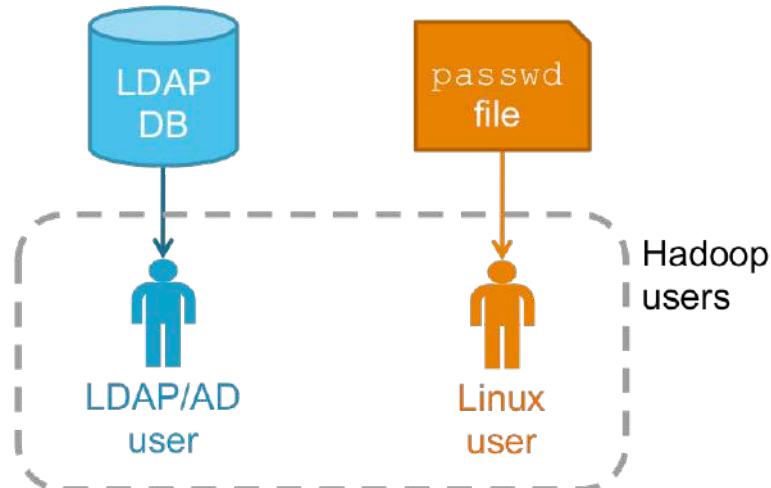


Ambari users are defined in the Ambari database. Ambari users may also be imported from an LDAP/AD database. Ambari users may use Ambari.

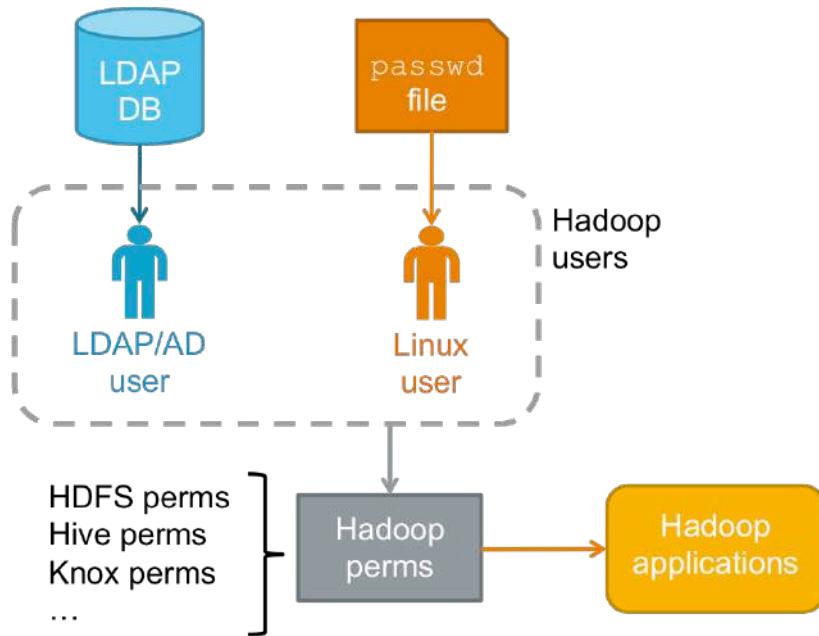


What any specific Ambari user can do in Ambari is determined by their Ambari privileges.

Hadoop users can be defined in an LDAP/AD database or in the Linux /etc/passwd file. Hadoop users may use Hadoop cluster applications.

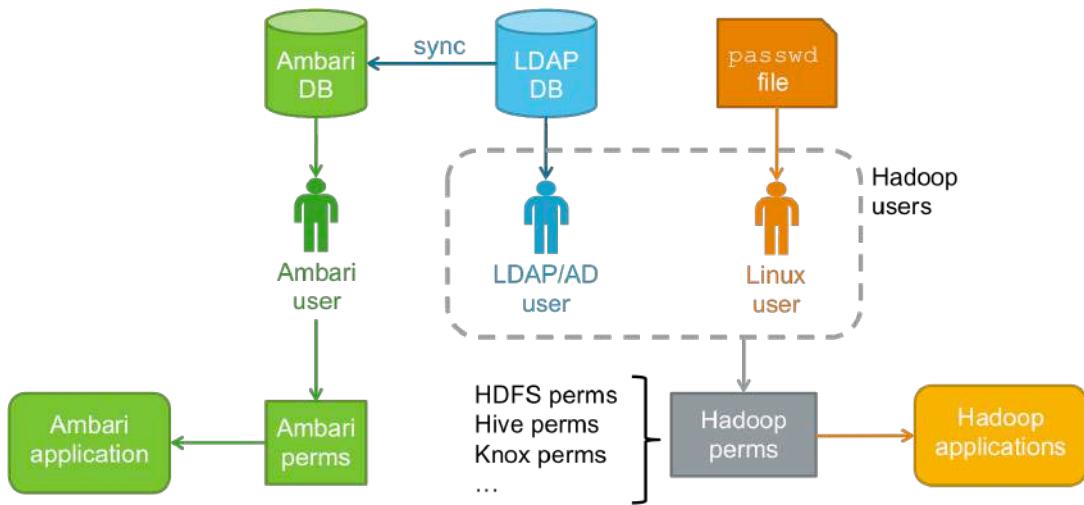


What any specific Hadoop user can do in a cluster depends on their Hadoop permissions. Hadoop permissions are defined in a variety of places.



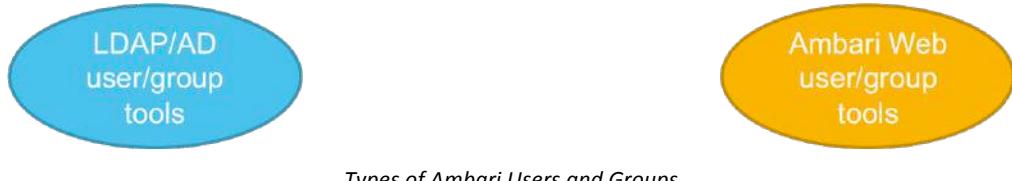
For example, the Hadoop Distributed File System has permissions, Apache Hive defines permissions, Apache Knox defines permissions, and so on. Some permissions are described in other lessons in this course while other permissions are described in other courses.

LDAP/AD users can be imported into Ambari and also used by Linux. For this reason, an LDAP/AD user might be able to use both Ambari and various Hadoop cluster applications.

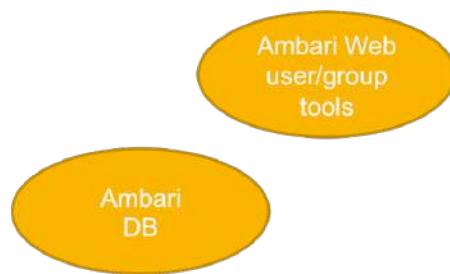


Which actions a user can perform depends on their Ambari and Hadoop permissions.

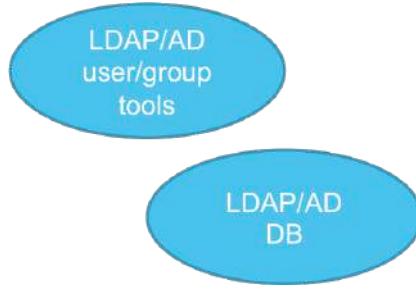
Ambari Users and Groups



Ambari supports two types of users and groups: Local and LDAP/AD.

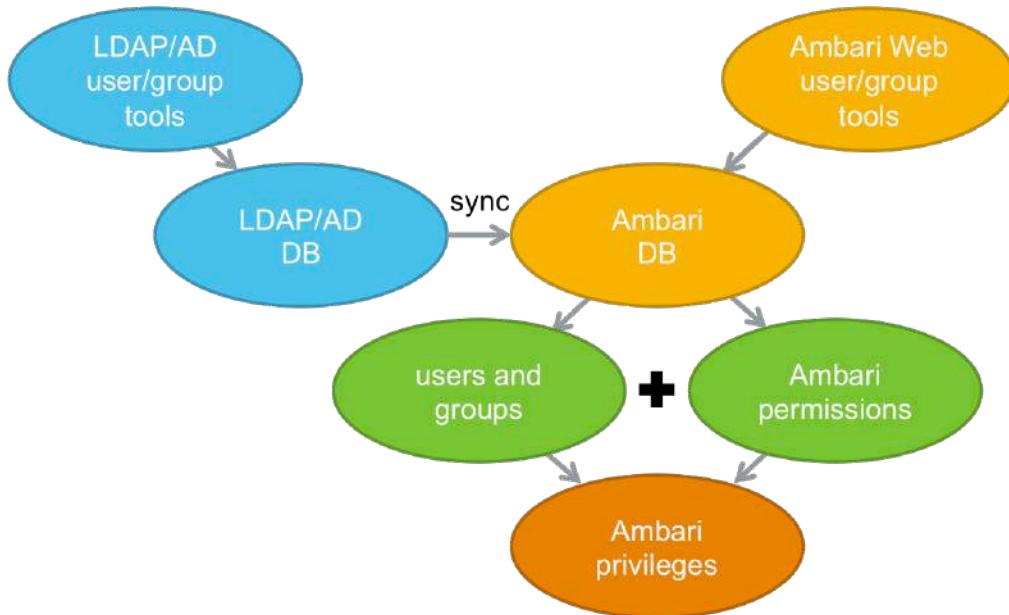


Local users and groups are created, managed, and deleted by Ambari and maintained in the Ambari database.



LDAP/AD Users and Groups

LDAP/AD users and groups are created, managed, and deleted by LDAP or Active Directory tools and maintained in an LDAP/AD directory database. LDAP/AD users and groups have only basic account and group membership information stored in the Ambari database. This information is transferred from LDAP/AD when an administrator runs an import and synchronization command.



User and Group Authentication and Permissions

To access LDAP/AD users and groups, the Ambari Server must be configured to communicate with an LDAP server or an Active Directory domain.

Local users authenticate to Ambari during log in. LDAP/AD users authenticate to an LDAP or Active Directory Server during log in.

An Ambari administrator assigns Ambari permissions to users and groups. These Ambari permissions determine user and group Ambari privileges. Ambari privileges determine what a user or group is allowed to see or do when using Ambari.

Ambari User and Group Permissions

Ambari Permission Type*	Privileges
No permission (default)	Ambari log in permitted, but no access to information
Read-only	Ambari log in permitted, may view but not modify services and configurations
Operator	Ambari log in permitted, may start, restart, stop, and add new services, may modify or revert configurations
Admin	Ambari log in permitted, has full permissions for services and configurations, may manage user and group accounts and grant permissions

*In addition to these, users and groups may be assigned access to one or more Ambari Views.

Ambari features four levels of user and group permissions. They are no permission, Read-Only permission, Operator permission, and Admin permission. These permissions determine how a user can use Ambari to interact with Hadoop services and configurations. Services are such things as HDFS, YARN, Hive, and so on. Services, for example, can be stopped and started. Configurations affect running services and the cluster topology.

The default for newly created Ambari users and groups is no permissions. These users may log in to Ambari but cannot see any cluster or service information or perform any actions.

Users or groups with Read-Only permission may log in to Ambari and view, but not modify, service and configuration information.

Those users and groups with Operator permission may log in to Ambari and view service and configuration information. They may also start, restart, and stop existing services as well as add new services. They may also modify current configurations or revert to previous configurations.

Ambari users with Admin permission may do anything. They may log in, view information, and manage services and configurations. In addition, they may also create new users and groups, manage group membership, and assign permissions to users and groups. They may even create other Ambari users with Admin permission. By default during installation, an Ambari admin user account is created and assigned Admin permission.

In addition to the four permission types, users and groups may be assigned access to one or more Ambari Views.

Admin Privileges for Local vs. LDAP/AD Users

Ambari User with Admin Permission	Local User/Group	LDAP/AD User/Group
Change passwords	Available	Not Available*
Assign Ambari Admin permission	Available	Available
Change group membership	Available	Not Available*
Delete users	Available	Not Available*
Mark users as active or inactive	Available	Available

*Must perform the action using LDAP/AD tools or utilities.

As an Ambari user with Admin permission, you can create new local users, delete local users, change local user passwords, and edit user settings. You can also control certain privileges for local and LDAP/AD users. However, many user and group management functions are not available to an Ambari administrator for LDAP/AD users and groups. This is because you need an LDAP/AD user or group management tool to manage LDAP/AD users and groups. Ambari can import these users and groups and can assign them Ambari permissions, but Ambari cannot administer the users and groups themselves.

The table shown here lists the privileges available as well as those not available to an Ambari administrator user.

Integrating Ambari Server with LDAP

The steps to integrate Ambari with LDAP are as follows:

- 1) Stop the Ambari server using the ambari-server stop command.

```
ambari-server stop
```

- 2) Run the ambari-server setup-ldap command and answer the resulting questions in order to configure Ambari for LDAP integration.

```
ambari-server setup-ldap
```

- 3) Restart the Ambari server using the ambari-server start command.

```
ambari-server start
```

- 4) Synchronize Ambari with LDAP using the ambari-server sync-ldap command, with the appropriate option(s).

```
ambari-server sync-ldap --[option]
```

Setting LDAP Properties on Ambari Server

```
[root@node1 ~]# ambari-server setup-ldap
Using python /usr/bin/python2.6
Setting up LDAP properties...
Primary URL* {host:port} (node4:389):
Secondary URL {host:port} : node4:389
Use SSL* [true/false] (false):
User object class* (person):
User name attribute* (uid):
Group object class* (groupofnames):
Group name attribute* (cn):
Group member attribute* (member):
Distinguished name attribute* (dn):
Base DN* (dc=TestAdmin,dc=com):
Referral method [follow/ignore] :
Bind anonymously* [true/false] (false): true
=====
Review Settings
=====
Save settings [y/n] (y)? y
Saving...done
Ambari Server 'setup-ldap' completed successfully.
```

Configuring LDAP on Ambari Server

The `ambari-server setup-ldap` command initiates an interactive configuration that asks for the information needed to connect to the LDAP server and modifies the `ambari-properties` configuration file accordingly. The answers to these questions will vary based on your particular LDAP server and its configurations, so you will want to make sure you have the following information available prior to initiating the LDAP setup:

- Primary LDAP server URL and port used (default is port 389)
- Secondary LDAP server URL and port, if applicable
- SSL required? (true/false) – determined by LDAP settings configured by the LDAP administrator
- User and group object classes – defined by the LDAP schema
- User and group name attributes – defined by the LDAP schema
- Group member attribute – defined by the LDAP schema
- Distinguished name attribute – defined by the LDAP API (default = dn)
- Base DN – comma-separated list of domain components (dc) that, from left to right, provides the query that gets to the LDAP server. Read from right to left by the system, so the effective query would be to find the “com” domain component, and then inside it find the TestAdmin domain component. More recognizable from a human-readable standpoint, the screenshot directs all base DN queries to TestAdmin.com.
- Referral method [follow/ignore] – Java Naming and Directory Interface (JNDI) property setting for how to handle LDAP referrals, which can allow directory administrators to set up back-end search paths for large / complex and evolving LDAP infrastructures.
- Bind anonymously? (true/false) – determined by LDAP settings configured by the LDAP administrator

Note

It is beyond the scope of this class to discuss LDAP configuration and schema definition, however your LDAP administrator should be able to provide you with the appropriate answers to the LDAP setup questions. If you answer a question incorrectly, you can repeat the LDAP setup procedure, which will update any changed responses in the properties file

Ambari LDAP Authentication Properties

Property	Values	Description
primaryUrl	Server:port	The hostname and port for the LDAP server Example: my.ldap.server:389
secondaryUrl	Server:port	The hostname and port for the secondary LDAP server Example: my.backupldap.server:389
useSSL	true or false	If true, use SSL when connecting to the LDAP server.
usernameAttribute	[LDAP attribute]	The attribute for username. Example: uid
baseDn	[Distinguished Name]	The root Distinguished Name to search in the directory for users. Example: ou=people,dc=hadoop,dc=apache,dc=org
referral	[Referral method]	Determines if LDAP referrals should be followed or ignored.
primaryUrl	Server:port	The hostname and port for the LDAP server Example: my.ldap.server:389
secondaryUrl	Server:port	The hostname and port for the secondary LDAP server Example: my.backupldap.server:389
useSSL	true or false	If true, use SSL when connecting to the LDAP server.
usernameAttribute	[LDAP attribute]	The attribute for username. Example: uid
baseDn	[Distinguished Name]	The root Distinguished Name to search in the directory for users. Example: ou=people,dc=hadoop,dc=apache,dc=org
referral	[Referral method]	Determines if LDAP referrals should be followed or ignored.

authentication.ldap. Properties*

The `ambari-server setup-ldap` command responses are written to the `/etc/ambari-server/conf/ambari.properties` file. The properties are all prepended with “`authentication.ldap.`” followed by the name of the property, an equals sign, and the value configured during the LDAP setup interactive configuration process. The properties and their values and descriptions are as follows:

- **primaryURL
(server:port)**
The hostname and port for the LDAP server.
Example: my.ldap.server:389
- **secondaryURL
(server:port)**
The hostname and port for the secondary LDAP server.
Example: my.backupldap.server:389

- **useSSL**
(true or false)
If true, use SSL when connecting to the LDAP server
- **usernameAttribute**
(LDAP attribute, defined by schema)
The attribute for username.
Example: uid
- **baseDN**
(LDAP Distinguished Name)
The root Distinguished Name to search in the directory for users.
Example: ou=people,dc=hadoop,dc=apache,dc=org
- **referral**
(follow or ignore)
JNDI setting that determines if LDAP referrals should be followed or ignored
- **bindAnonymously**
(true or false)
If true, bind to the LDAP server anonymously
- **managerDN**
(Full Distinguished Name)
If bindAnonymously=false, the DN for the manager.
Example: uid=hdfs,ou=people,dc=hadoop,dc=apache,dc=org
- **managerPassword**
(password)
If bindAnonymously=false, the password for the manager
- **userObjectClass**
(LDAP Object Class, defined by schema)
The object class that is used for user accounts.
Example: organizationalPerson
- **groupObjectClass**
(LDAP Object Class, defined by schema)
The object class that is used for groups.
Example: groupOfUniqueNames
- **groupMembershipAttr**
(LDAP attribute, defined by schema)
The attribute for group membership.
Example: uniqueMember
- **groupNamingAttr**
(LDAP attribute, defined by schema)
The attribute for group name.

Verifying Ambari LDAP Properties

```
api.authenticate=true
authentication.ldap.baseDn=dc=TestAdmin,dc=com
authentication.ldap.bindAnonymously=true
authentication.ldap.dnAttribute=dn
authentication.ldap.groupMembershipAttr=member
authentication.ldap.groupNamingAttr=cn
authentication.ldap.groupObjectClass=groupofnames
authentication.ldap.managerDn=uid=guest,ou=people,dc=TestAdmin,dc=com
authentication.ldap.managerPassword=/etc/ambari-server/conf/ldap-password.dat
authentication.ldap.pagination.enabled=false
authentication.ldap.primaryUrl=node4:389
authentication.ldap.secondaryUrl=node4:389
authentication.ldap.useSSL=false
authentication.ldap.userObjectClass=person
authentication.ldap.usernameAttribute=uid
bootstrap.dir=/var/run/ambari-server/bootstrap
```

Verifying Ambari LDAP Properties

To verify your settings have been configured, use a text editor or viewer to open the /etc/ambari-server/conf/ambari.properties file. You should see a number of authentication.ldap.* entries with the proper configurations.

Restarting Ambari

Assuming everything has been configured properly, you can restart Ambari and begin synchronization with LDAP.

LDAP Synchronization

There are three basic functions of the ambari-server sync-ldap command:

- Synchronize all users and groups in LDAP with the --all option
- Synchronize specific users and groups provided via a text file of comma-separated users and groups with the --users <user list file> and --groups <group list file> options
- Update existing users and groups (resynchronize) with the --existing option.

You will be asked to provide credentials for an Ambari Admin. When syncing with LDAP, local user accounts with matching username will switch to LDAP type, which means their authentication will be against the external LDAP and not against the local Ambari user store.

If using the --users and --groups options, the comma separated entries in each of these files should be based on the values in LDAP of the attributes chosen during setup:

- The "User name attribute" should be used for the user list file
- The "Group name attribute" should be used for the groups list file

When using the --existing option, users will be removed from Ambari if they no longer exist in LDAP, and group membership in Ambari will be updated to match LDAP. Group membership is determined using the Group Membership Attribute specified during LDAP setup.

Synchronizing All Users and Groups

```
[root@node1 ~]# ambari-server sync-ldap --all
Using python /usr/bin/python2.6
Syncing with LDAP...
Enter Ambari Admin login: admin
Enter Ambari Admin password:
Syncing all...

Completed LDAP Sync.
Summary:
memberships:
  removed = 0
  created = 0
users:
  updated = 0
  removed = 0
  created = 2
groups:
  updated = 0
  removed = 0
  created = 1

Ambari Server 'sync-ldap' completed successfully.
[root@node1 ~]#
```

Output of the ambari-server sync-ldap --all Command

This screenshot shows a sample output using the `ambari-server sync-ldap --all` command. This will import all entities with matching LDAP user and group object classes into Ambari. This should be used only if you are sure you want to synchronize all users and groups from LDAP into Ambari. If you only want to synchronize a subset of users and groups, use a specific set of users and groups option.

Additionally, there is a limit of 1,000 users that can be synchronized at a time. If your LDAP setup contains more than 1,000 users, you are required to use the `--users` and `--groups` options when synchronizing with LDAP and perform the synchronization batches in groups of 1,000 or fewer at a time.

Note

You can review log files for failed synchronization attempts, at `/var/log/ambari-server/ambari-server.log` on the Ambari Server host.

Integrating Ambari with LDAP

Verifying LDAP Users and Groups

The screenshots show the Ambari User + Group Management interface. The top screenshot displays the 'Groups' page, which lists a single group named 'engineering' of type 'LDAP' with 0 members. The bottom screenshot displays the 'Users' page, which lists three users: 'admin' (Local, Active), 'jeden' (LDAP, Active), and 'sbradshaw' (LDAP, Active). Both screenshots have the 'Groups' tab selected in the sidebar.

Group Name	Type	Members
engineering	LDAP	0 members

Username	Type	Status
admin	Local	Active
jeden	LDAP	Active
sbradshaw	LDAP	Active

Users and Groups Ambari Management Interface

Once synchronization with LDAP is complete, you can confirm success by logging into Ambari and going to the management interface, then click on Groups and Users under User + Group Management. You should see named groups and users, and under the Type section you will see their source – in particular, Local vs. LDAP.

Knowledge Check

Questions

- 1) Where are the three locations a Hadoop user can be defined?
- 2) True or False: Ambari Server must be running in order to synchronize with the LDAP server.
- 3) True or False: Updates to LDAP users and groups are automatically synchronized to an integrated Ambari server by default.
- 4) What two functions can an Ambari administrator perform on integrated LDAP users?
- 5) Mary is integrating an Ambari server with an LDAP environment composed of 1,500 users and 150 groups. What is the quickest way to get all of these users and groups added to Ambari?

Answers

- 1) Where are the three locations a Hadoop user can be defined?

Answer: Ambari server, LDAP server, and local system (Linux user)

- 2) True or False: Ambari Server must be running in order to synchronize with the LDAP server.

Answer: True. Ambari server only needs to be stopped when setting up LDAP.

- 3) True or False: Updates to LDAP users and groups are automatically synchronized to an integrated Ambari server by default.

Answer: False. The `ambari-server sync-ldap --existing` command must be run to update previously imported users and groups.

- 4) What two functions can an Ambari administrator perform on integrated LDAP users?

Answer: Assign Ambari permissions and mark active or inactive. (Password changes, group membership, and user deletion are not available)

- 5) Mary is integrating an Ambari server with an LDAP environment composed of 1,500 users and 150 groups. What is the quickest way to get all of these users and groups added to Ambari?

Answer: Mary must use the `ambari-server sync-ldap` command with the `--users` and `--groups` options, and provide two comma-separated lists of the users and groups to integrate. The `--all` option will not work because the user list exceeds 1,000 users.

Summary

- Ambari can be synchronized with an existing LDAP infrastructure to ease user and group management
- LDAP environments of less than 1,000 users can be integrated all at once, otherwise multiple integration actions are required
- LDAP updates are not automatically pushed to Ambari, however a single command can be used to update all existing users and groups

Hive Tuning

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Configure YARN queues, Tez, and Hive properties to support performance goals

Tune Hive for Interactive Queries

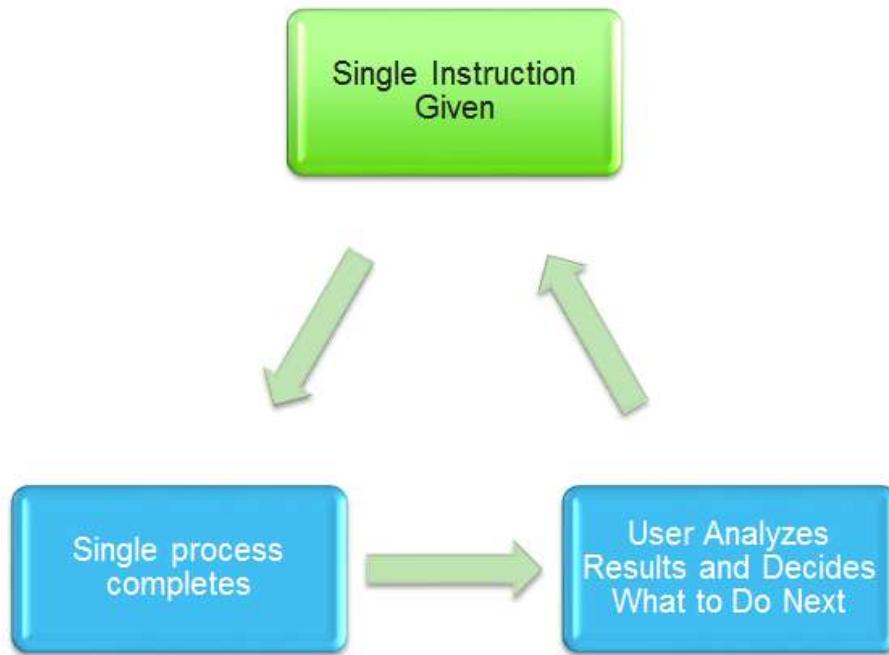
Hive settings can be tweaked to support high-performance applications and interactive queries.

Batch Versus Interactive Processing



Batch processing is the execution of a series of one or more possibly complex tasks in an automated fashion. Batch operations are normally repeated on a consistent basis, perhaps hourly, daily, weekly, monthly, etc. The key thing about batch operations is that they run without manual intervention – once they are initiated, they run autonomously until they are complete and produce a predictable output each time. They have a clearly defined point when the job is complete.

Interactive processing is the execution of a series of tasks, but each step is decided manually by a human user and generally based on the results of the previous task.



Interactive processing is generally executed a single instruction at a time, and has an undetermined number of steps – the user will continue executing commands until the desired goal has been achieved.

Why Tune Hive?

Hive was originally created to run batch operations. In the early days of Hadoop, the speed limitations of MapReduce made it difficult to use Hive for interactive queries, as even a simple query might take several minutes to complete. Today, however, with the introduction of Tez as the default data processing framework and general performance improvement options in Hive, Hive can be a viable solution for interactive as well as batch processing.

When using Hive for interactive processing, there are a number of things that can be done to improve performance. Some of those things are procedural in nature – for example, the use of the ORC file format and column-level statistics. (Please refer to docs.hortonworks.com for additional information on these and other procedural practices for improving Hive performance.) A number of system-level settings can also be made to improve performance.

Tuning for Interactive Queries

System-level settings that can improve performance of interactive queries include:

- Increase HDFS replication factor for frequently accessed data
- Create multiple HiveServer2 instances
- Configure queue management strategies based on application characteristics
- Configure Tez container management settings based on application characteristics

Hive Tuning

Increase HDFS Replication Factor

The screenshot shows the Ambari configuration interface for the HDFS service. The 'General' tab is selected. The configuration parameters shown are:

- WebHDFS enabled: Enabled (checkbox checked)
- Hadoop maximum Java heap size: 1024 MB (input field, unit MB)
- Reserved space for HDFS: 1073741824 bytes (input field, unit bytes)
- HDFS Maximum Checkpoint Delay: 21600 seconds (input field, unit seconds)
- Block replication: 3 (input field)

Increasing the replication factor for HDFS has the effect of allowing workloads running on the same data to be spread across a greater number of physical nodes, which improves overall processing performance. The trade-off is that each data block is replicated more times, thus requiring more storage. This can be configured under **HDFS Services > Configs > Advanced > General**.

Create Multiple HiveServer2 Instances

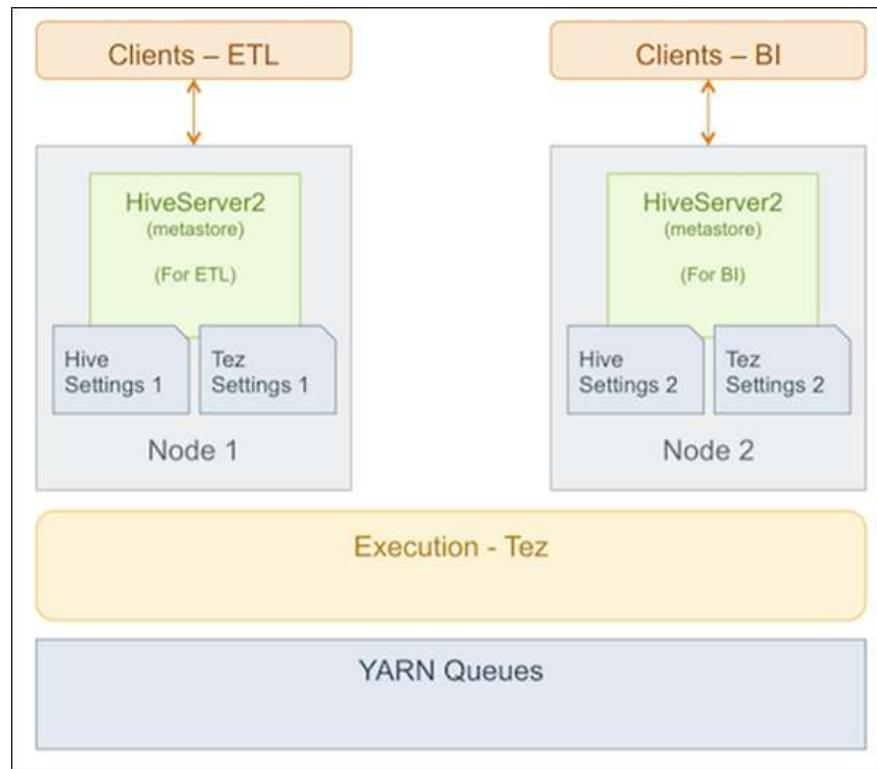
The screenshot shows the Ambari service status interface. The sidebar on the left shows the services: HDFS, MapReduce2, YARN, Tez, **Hive**, Pig, ZooKeeper, and Ambari Metrics. The 'Hive' service is selected. The main panel shows the 'Summary' tab for the Hive service, listing the following components and their status:

Component	Status
Hive Metastore	Started
HiveServer2	Started
MySQL Server	Started
WebHCat Server	Started
HCat Clients	2 HCat Clients Installed
Hive Clients	2 Hive Clients Installed

To the right of the summary, a 'Service Actions' dropdown menu is open, listing various actions for the Hive service:

- Start
- Stop
- Restart All
- Move Hive Metastore
- Move HiveServer2
- Move MySQL Server
- Move WebHCat Server
- Run Service Check
- Turn On Maintenance Mode
- Add Hive Metastore
- Add HiveServer2
- Download Client Configs

For multiple workloads or applications, using multiple HiveServer2 instances is recommended. Each HiveServer2 instance can have its own settings for Hive and Tez. For example, one group or tool can be configured to use the instance that stays tuned for batch operations, while a second group or tool can use an instance installed on a different node with settings tuned for interactive queries.



Queue Strategies

Interactive Query

Default query queues

default queue ▾

Start Tez session at Initialization

False

Session per queue

1

In general, as of HDP 2.3 and beyond, it is recommended that a single interactive queue be configured and that all applications that perform interactive processing be assigned to that queue. To control the number of applications that can run in the queue at a time, configure the number of Tez sessions per queue in the **Interactive Query** section of **Hive Services > Configs**.

However, assume you have two applications with two different types of commonly used queries. One type of query takes approximately 5 seconds to run, and the other type takes approximately 45 seconds to run. If both of these types of queries were assigned to the same queue, the shorter-running queries must wait for the longer-running queries. In this case, it is recommended that the two queries with different execution times be assigned to separate queues.

Configure Tez Idle Container Settings

The screenshot shows the 'Advanced tez-site' configuration section with the following settings:

- tez.am.view-acls**: A dropdown menu with an asterisk (*) selected.
- tez.am.am-rm.heartbeat.interval-ms.max**: Set to 250.
- tez.am.container.idle.release-timeout-max.millis**: Set to 20000.
- tez.am.container.idle.release-timeout-min.millis**: Set to 10000.
- tez.am.container.reuse.enabled**: Set to true.

By default, Tez waits a minimum of 10 seconds and a maximum of 20 seconds before releasing idle containers. Interactive queries in which users may take more time than that between operations have to reallocate containers more often, which can affect query performance. Increasing these timeout settings can improve performance for those types of queries, at the expense of making idle resources available to other users and jobs.

Configure at Tez Services > Configs > Advanced tez-site.

Configure Tez Held Containers

The screenshot shows the 'Optimization' configuration page for Tez Held Containers:

- Execution Engine**: Set to TEZ.
- Hold Containers to Reduce Latency**: A toggle switch set to False.
- Tez Container Size**: Set to 512 MB.
- Number of Containers Held**: A slider set to 3, with a scale from 1 to 20.

Tez can be configured to “prewarm” containers and keep a minimum number of them available regardless of actual current usage. This reduces lag time for new Tez operations. It can be configured under **Hive Services > Configs > Optimization**.

Increase Tez DAG Submission Timeout



By default, an finished Tez job waits 10 minutes for a new DAG to be submitted before shutting down. This setting can be modified at **Tez Services > Configs > Advanced tez-site**.

Additional Considerations

There are a number of additional settings that can be monitored and changed in order to improve overall performance of Hive interactive queries:

- Set YARN, Tez, and Hive memory settings for optimum performance
- Monitor parallelism, garbage collection times, and join behavior, then modify query performance via Hive and Tez configuration changes
- Use the ORC file format
- Use column statistics and the cost-based optimizer (CBO)
- Design data for optimum use of partitions and buckets

REFERENCE

Refer to Hive Performance Tuning Guide at <http://docs.hortonworks.com> for additional information on these options.

Summary

- For optimum performance with interactive Hive queries:
 - Modify Hive, Tez, and YARN settings based on application characteristics
 - Modify queues and queue settings based on application characteristics

Apache Hive High Availability

Lesson Objectives

After completing this lesson, students should be able to:

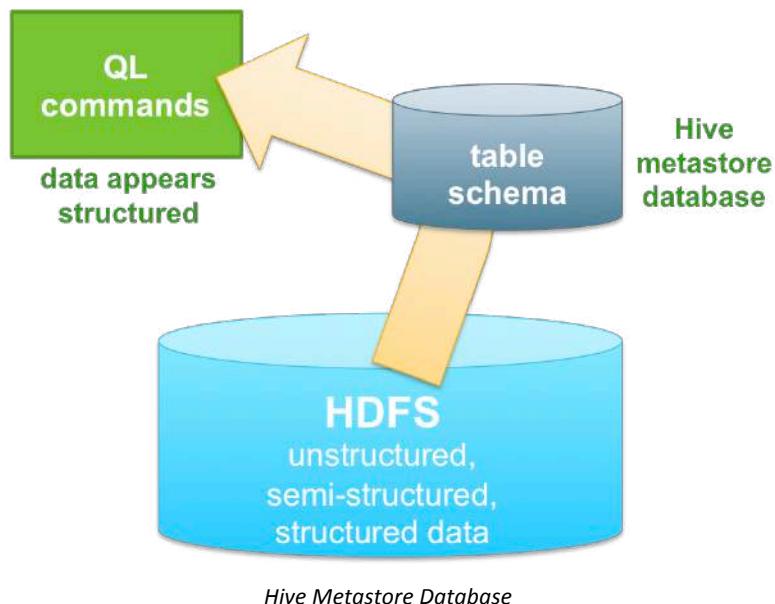
- ✓ Recall basic facts about Hive and the Hive architecture
- ✓ Recall the purpose and benefits of Hive HA
- ✓ Summarize the Hive HA architecture and operation
- ✓ Configure and test Hive HA

Apache Hive

Hive is a data warehouse infrastructure built on top of Hadoop.

Hive includes a SQL-like language called Query Language, or QL. Hive QL statements are automatically converted to Tez or MapReduce jobs. The default in HDP 2.3 is conversion to Tez jobs. Hive and QL enable an enterprise to utilize existing SQL skillsets to quickly derive value from a Hadoop deployment, without users having to learn the Tez or MapReduce programming frameworks.

Structured and Unstructured Data



Hive is not a relational database although, on the surface, it can appear like one.

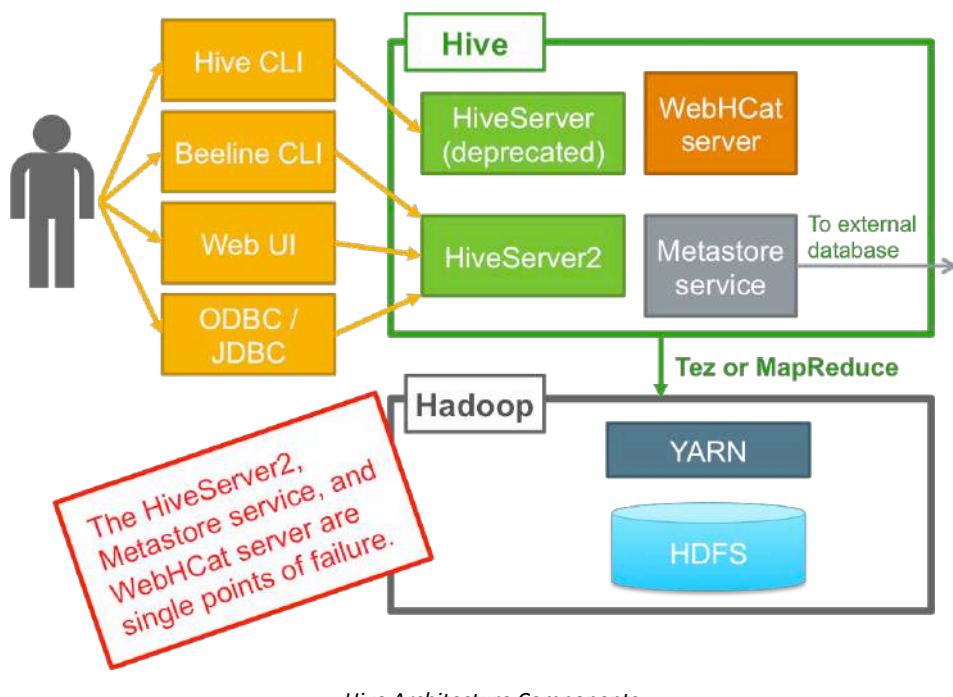
Hadoop was built to collect, store, and analyze massive amounts of data. As such, the Hadoop distributed file system is a reservoir of data from multiple sources. The data is often a mix of unstructured, semi-structured, and structured data. Hive provides a mechanism to project structure onto HDFS data and then query it using QL. However, there is a limit to what Hive can do. Sometimes it is necessary to use another tool, like Apache Pig, to pre-format the unstructured data before processing it using Hive.

If you are familiar with databases, then you understand that unstructured data has no schema associated with it. If you are not familiar with database schemas, they define the columns of a database table along with the type of data in each column. Data types include such things as a string, an integer, a floating point number, or a date.

A Hive installation includes a metastore database. Several database types are supported by Hive including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. To project structure on HDFS data, QL includes statements to create a table with user-defined schema information. The table schema is stored in the metastore database.

The user-defined schema is associated with the data stored in one or more HDFS files when you use QL statements to load the files into a table. The format of the data on HDFS remains unchanged but it appears as structured data when using QL commands to submit queries.

Hive Architecture Components



Hive Architecture Components

The original Hive was implemented as a heavy command-line tool that accepted queries and then executed those queries by running them as a MapReduce job. This original architecture was replaced by a client-server architecture that included the Hive CLI and the HiveServer daemon. The HiveServer daemon was responsible for compiling and monitoring MapReduce jobs. The Hive CLI was a Thrift-based client used to send SQL commands to the HiveServer for execution.

In the current architecture, the HiveServer has been replaced by the HiveServer2 daemon. The HiveServer2 provides multi-client concurrency, improved authentication, and supports clients connecting through JDBC or ODBC. The Hive CLI has been replaced by the new Beeline command-line interface. Beeline is a JDBC client although the JDBC driver that is used communicates with HiveServer2 using HiveServer2's Thrift APIs.

The WebHCat server handles REST API requests to the HCatalog, which enables HTTP-based requests to access the Hive metastore information.

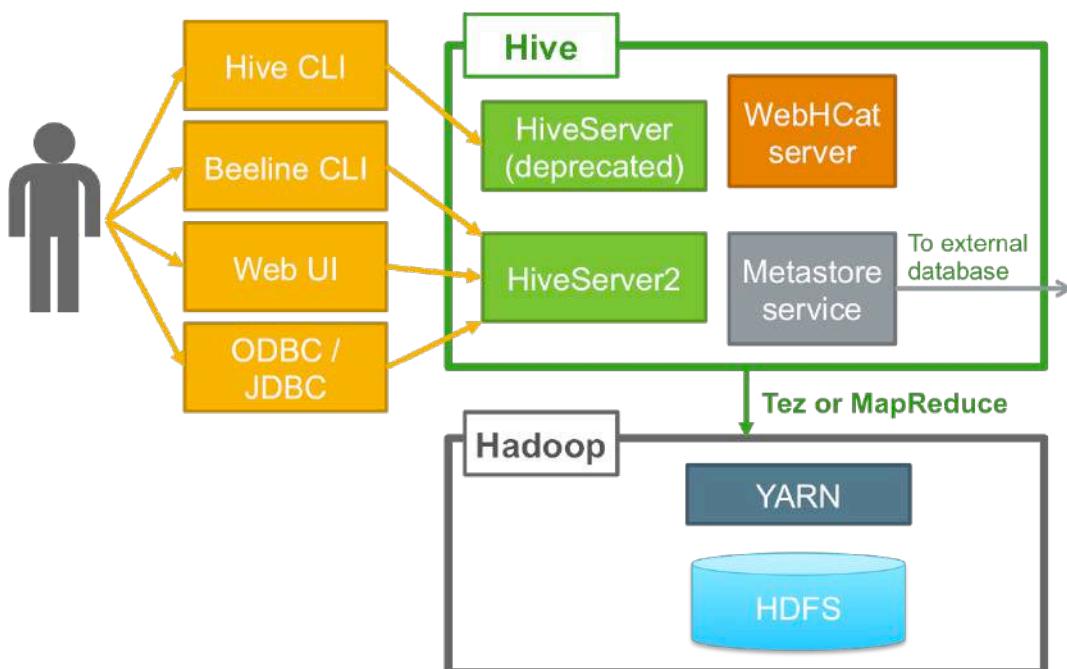
Hive also includes a metastore service. The metastore service manages and provides client access to the underlying relational database that stores Hive table schema and partition information. The relational database can be implemented using MySQL, Oracle, and PostgreSQL. Windows-based HDP environments can also use SQL Server.

Reference

Refer to the HDP documentation at <http://docs.hortonworks.com> for information about supported database versions.

When Hive is first installed, the HiveServer2, metastore service, and WebHCat server are single points of failure. However, the HiveServer2, metastore service, and WebHCat server can be configured with redundant components for high availability.

Submitting Hive Queries



Interactively Submitting Hive Queries Through the Hive CLI

Hive includes many methods to submit queries. Queries submitted to either the HiveServer or newer HiveServer2 result in a Tez or MapReduce job being submitted to YARN. YARN, the Hadoop resource scheduler, works in concert with HDFS to run the job in parallel across the machines in the cluster.

Hive CLI

The Hive CLI was used to interactively or non-interactively submit QL commands to the original HiveServer daemon. The Hive CLI has been deprecated and will not work with the newer HiveServer2 daemon. The illustration shows the Hive CLI being used interactively. Users enter QL commands at the `hive>` prompt. QL commands can also be placed into a file and run using `hive -f <file_name>`.

The remaining methods all submit QL queries to the newer HiveServer2.

Beeline CLI

The Beeline CLI is a new JDBC client that connects to a local or remote HiveServer2. When connecting locally, Beeline works just like the Hive CLI. Beeline can connect to a remote HiveServer2 using a variety of methods that include TCP and HTTP.

For example, to connect to a remote host using TCP, type beeline at the shell prompt. At the resulting beeline> prompt, type !connect jdbc:hive2://<HS2_host>:<port> to connect to the remote HiveServer2. The <HS2_host> would be the hostname of the machine running the HiveServer2 while the port is typically 10000. HTTP access is useful for submitting queries to a firewall-protected cluster, assuming the firewall will allow HTTP traffic.

For more information about Beeline syntax, refer to the online Beeline documentation.

ODBC and JDBC

ODBC and JDBC drivers enable you to connect to popular business intelligence tools to query, analyze, and visualize Hive data.

Other Options (not shown)

While not illustrated above, you may also use the Ambari Hive View or the Hadoop User Experience, called HUE, to submit Hive queries. The Hive View is an Ambari View plugin that enables a user to query Hive using a browser-based interface. HUE is a graphical interface that also enables a user to interactively enter Hive queries. Like most graphical tools, both provide many features to make using Hive easier. For example, you can write and test a series of Hive queries and then easily save them to a script for future use. You can use a single interface to execute queries and view the results and execution logs.

Hive HA Requirements and Benefits

There are requirements to configure Hive HA but Hive HA provides several benefits.

Hive HA Requirements

There are a few requirements when implementing Hive HA:

- Hive HA is supported only on HDP 2.2 or later.
- Hive HA must have access to a ZooKeeper cluster.
- The HiveServer2, metastore service, and WebHCat server must be configured with redundant components.
- The external relational database supporting the Hive metastore service should be configured for high availability. Hive supports different database platforms so use vendor-specific configuration instructions when configuring database high availability.

Hive HA Benefits

Hive HA provides the benefits of high availability, load balancing, and support for rolling upgrade.

High Availability

If multiple HiveServer2 instances are configured, and all instances fail except one, a Hive client can still successfully connect and submit Hive queries. Failed HiveServer2 instances must be restarted manually. Existing client connections must also be manually reestablished to a working HiveServer2 instance.

If multiple metastore service instances are configured, and all instances fail except one, a Hive client can still successfully connect and submit Hive queries. Failed metastore service instances must be restarted manually. Existing client connections are automatically reestablished to a working metastore service instance.

If multiple WebHCat servers are configured, and all instances fail except one, a WebHCat client can still successfully connect and access the HCatalog. Failed WebHCat servers must be restarted manually. Existing client connections are automatically reestablished to a working WebHCat server.

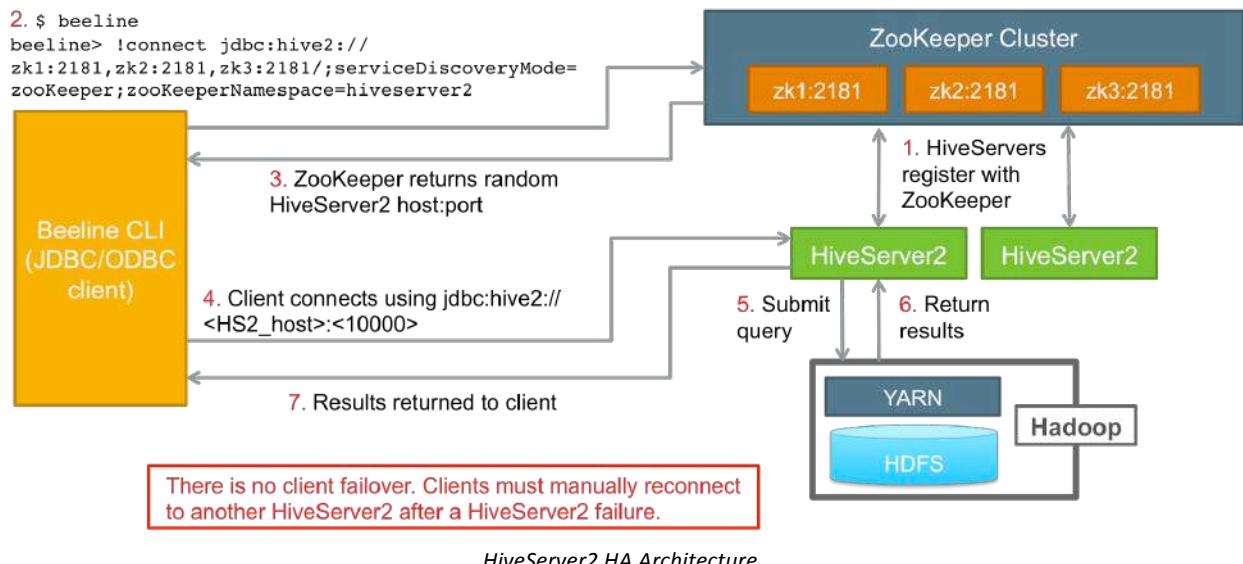
Load Balancing

If multiple HiveServer2 instances are configured, Hive clients randomly connect to one of the instances. This ensures that all HiveServer2 instances receive roughly the same workload. Requests to the metastore service are not load balanced. If it is available, Hive clients connect to the first one listed in the `hive-site.xml` file. If the first one listed is not available then another one from the list is randomly selected.

Rolling Upgrade

It is possible to register HiveServer2 instances based on their version. During rolling upgrade it is possible to configure new-version HiveServer2 instances as active and old-version instances as passive. Hive HA only opens new client connections to an active HiveServer2 instance, which means that over time the old-version instances become unused and can be removed. Rolling upgrade is described in another lesson.

Hive HA Architecture and Operation



HiveServer2 HA Architecture

Starting with HDP 2.2, Ambari can deploy multiple HiveServer2 instances to enable high availability. Each HiveServer2 registers with the ZooKeeper cluster. The client JDBC and ODBC driver has been updated with the capability to use ZooKeeper to dynamically discover—called dynamic discovery—an available HiveServer2.

Client Query Execution

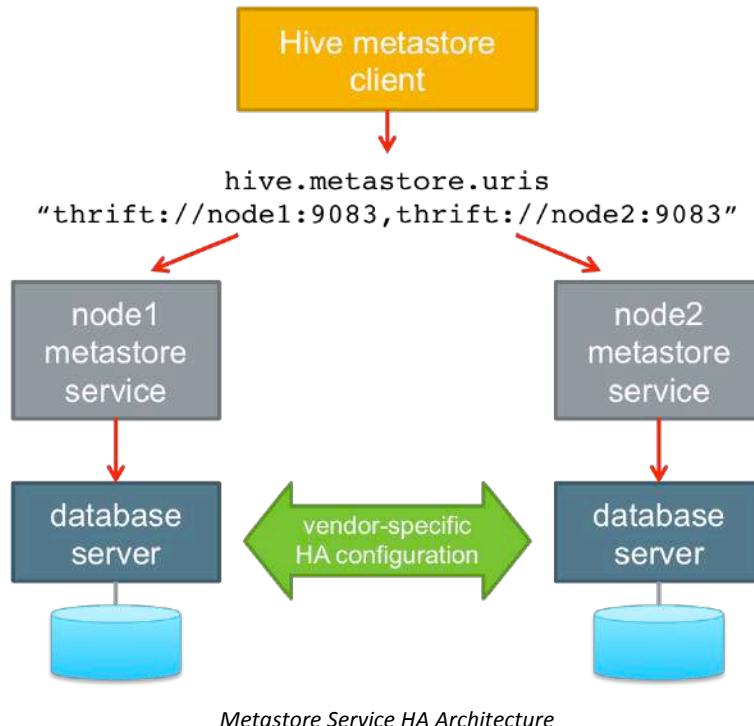
Client query execution works as follows:

- 1) Each HiveServer2 instance registers itself with the ZooKeeper cluster. As long as the HiveServer2 is active, it remains registered with ZooKeeper.
- 2) The client driver connects to the ZooKeeper cluster.
- 3) The ZooKeeper cluster randomly returns the hostname and port number of one of the active HiveServer2 machines. This facilitates high availability and load balancing.
- 4) The client driver connects to the HiveServer2 machine and sends its query.
- 5) The HiveServer2 machine submits the query to Hadoop as a Tez or MapReduce job.
- 6) Hadoop returns the information to the HiveServer2 machine.
- 7) The HiveServer2 machine returns the query result to the client driver.

If a HiveServer2 instance fails while a client is connected, the session is lost. Because the client must handle failure, there is no automatic failover. The client must reconnect to a remaining HiveServer2 using a new request to ZooKeeper.

HA vs. Fault Tolerance: There may be confusion about the difference between a highly available service – in which service failure recovery is automatic but job continuity is not guaranteed – and a fault tolerant service. Fault tolerant services allow for the failure of a service component without interrupting in-place jobs. This is accomplished by running every job simultaneously on two different nodes, and tracking progress of the job at memory and cpu-execution code level. Fault tolerance is both extremely difficult and costly in terms of resource usage and application performance. HDP services are highly available but not fault tolerant, which is why it is necessary to re-run a job if the HiverServer2 node it is assigned to fails.

Metastore Service HA Architecture

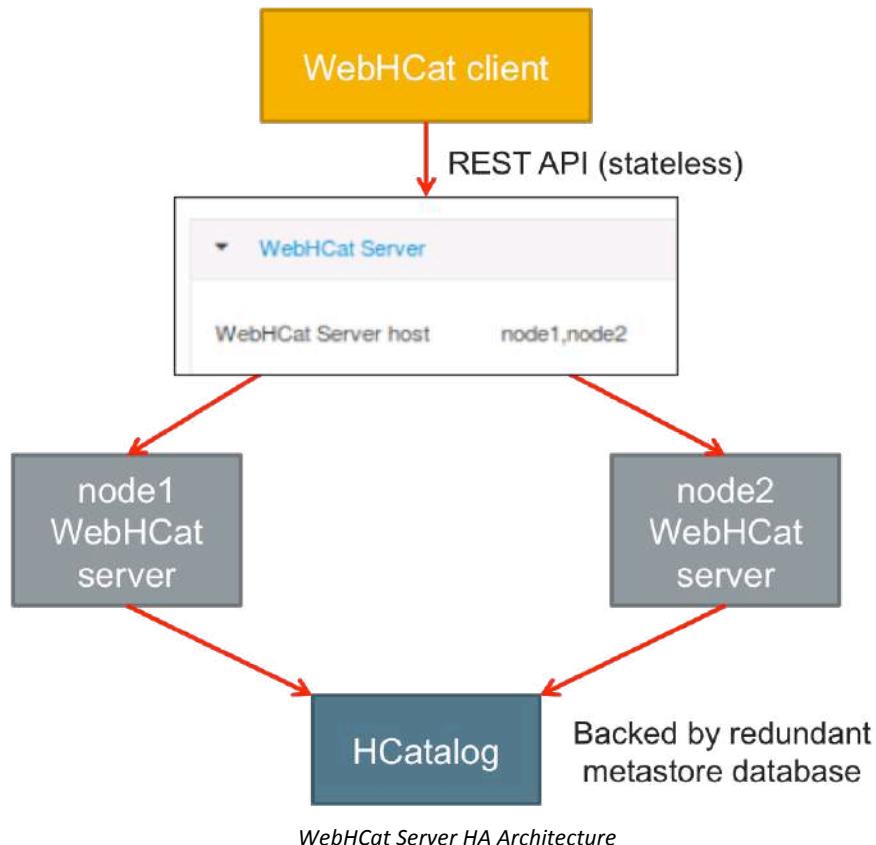


Ambari can deploy the metastore service on multiple machines for high availability. The metastore service HA solution is designed to handle service failures.

The Hive metastore client uses the value of the `hive.metastore.uris` property in the `/etc/hive/conf/hive-site.xml` file to locate a metastore service. Before HA is configured the value will reference a single instance, similar in format to `thrift://node1:9083`. Following HA configuration the value will reference multiple instances, similar in format to `thrift://node1:9083,thrift://node2:9083`.

A Hive metastore client always uses the first URI to connect to a metastore service. If the first metastore service is unavailable, the client will randomly try a URI from the list and attempt to connect to it. Failover is automatic.

WebHCat Server HA Architecture



Ambari can deploy the WebHCat server on multiple machines for high availability. The WebHCat server HA solution is designed to handle service failures.

The WebHCat client uses Ambari configuration information to locate a WebHCat server. Before HA is configured the configuration will reference a single instance. Following HA configuration the configuration will reference multiple instances.

A WebHCat client always uses the first entry to connect to a WebHCat server. If the first entry is unavailable, the client will try the next entry from the list and attempt to connect to it. Because the REST API is stateless, client failover is automatic.

Installing Hive and Configuring Hive HA

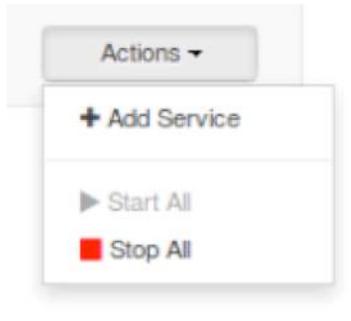
Installing and configuring Hive HA is a multi-phase process, where each phase consists of multiple steps. While this might initially appear to be time consuming or error prone, it is actually made simple by using the Ambari Web UI.

The four phases are:

- 1 . Install Hive (first phase)
- 2 . Configure Hive HA
 - a. Add redundant metastore services (second phase)
 - b. Add redundant HiveServer2 instances (third phase)
 - c. Add redundant WebHCat servers (fourth phase)

The steps in each phase are illustrated next.

Installing Hive



Installing Hive Using Ambari Web UI

To install Hive, log in to the Ambari Web UI as a user with at least Operator permissions. Click the **Services** tab to reveal the service **Actions** menu. Click **Actions** and select **Add Service**. The Add Service wizard opens.

Choose Services

Choose Services

Choose which services you want to install on your cluster.

Select Hive and Pig.

Service	Version	Description
<input checked="" type="checkbox"/> HDFS	2.7.1.2.3	Apache Hadoop Distributed File System
<input checked="" type="checkbox"/> YARN + MapReduce2	2.7.1.2.3	Apache Hadoop NextGen MapReduce (YARN)
<input checked="" type="checkbox"/> Tez	0.7.0.2.3	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
<input checked="" type="checkbox"/> Hive	1.2.1.2.3	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
<input type="checkbox"/> HBase	1.1.1.2.3	A Non-relational distributed database, plus Phoenix, a high performance SQL layer for low latency applications.
<input checked="" type="checkbox"/> Pig	0.15.0.2.3	Scripting platform for analyzing large datasets

Choosing Services in Ambari Web UI

Select the **Hive** and **Pig** services to install Hive. If you do not select Pig, Ambari will warn you that Pig is a requirement when installing Hive and ask you to select Pig too. Scroll down and click **Next** (not shown) when ready to continue.

Assign Masters

Component	Host Assigned
SNameNode	node1 (14.4 GB, 4 cores)
NameNode	node1 (14.4 GB, 4 cores)
History Server	node1 (14.4 GB, 4 cores)
App Timeline Server	node1 (14.4 GB, 4 cores)
ResourceManager	node1 (14.4 GB, 4 cores)
Hive Metastore	node1 (14.4 GB, 4 cores)
WebHCat Server	node1*
HiveServer2	node1 (14.4 GB, 4 cores)

node1 (14.4 GB, 4 cores)

 SNameNode NameNode History Server

 App Timeline Server ResourceManager

 Hive Metastore WebHCat Server

 HiveServer2 ZooKeeper Server

 Metrics Collector

node2 (14.4 GB, 4 cores)

 ZooKeeper Server

node3 (14.4 GB, 4 cores)

 ZooKeeper Server

Assigning Masters in the Ambari Web UI

Ambari provides an initial layout of service master components. Use the **Assign Masters** window either to confirm or move the master components. This is when information about workloads and resource capacity gained from pre-planning, or pre-testing with a pilot cluster, is very helpful. The resource capacity of each host must be considered along with service redundancy.

Use the drop-down menu next to each component to move the master component to another cluster node, as necessary. Click **Next** (not shown) to proceed.

Assign Slaves and Clients

Host	all none	all none	all none	all none
node1*	<input type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node2*	<input type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node3*	<input type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client

Show: 25 ▾ 1 - 3 of 3 ⏪ ⏩ ⏴

← Back
Next →

Assigning Slaves and Clients in the Ambari Web UI

Use the Assign Slaves and Clients window to choose where to run service worker components and Hadoop client software.

The DataNode is the HDFS service worker component and is already installed on all nodes in the screen capture.

An HDFS NFS Gateway machine can be used as a method to access HDFS. The HDFS NFS Gateway is described in another lesson.

The NodeManager is the YARN service worker component and is already installed on all nodes in the screen capture.

Client represents the Hadoop client software. Client software is used by users and applications to access cluster services and resources. For example, client software is used to access HDFS storage, run YARN jobs, or in this specific case, submit Hive queries. Client software is commonly installed on cluster nodes. Client software is also commonly installed on utility machines used as gateway machines to access cluster resources.

Select all nodes from which Hive queries will be submitted and click **Next**.

Customize Services

Customize Services

We have come up with recommended configurations for the services you selected. Customize them as you see fit.

HDFS MapReduce2 YARN Tez **Hive 1** Pig ZooKeeper Ambari Metrics Misc

Group Hive Default (3) Manage Config Groups

Settings Advanced 1

Hive Metastore 1

Hive Metastore hosts node1

Database Host node4

Database Type MySQL

Hive Database

- New MySQL Database
- Existing MySQL Database
- Existing PostgreSQL Database
- Existing Oracle Database

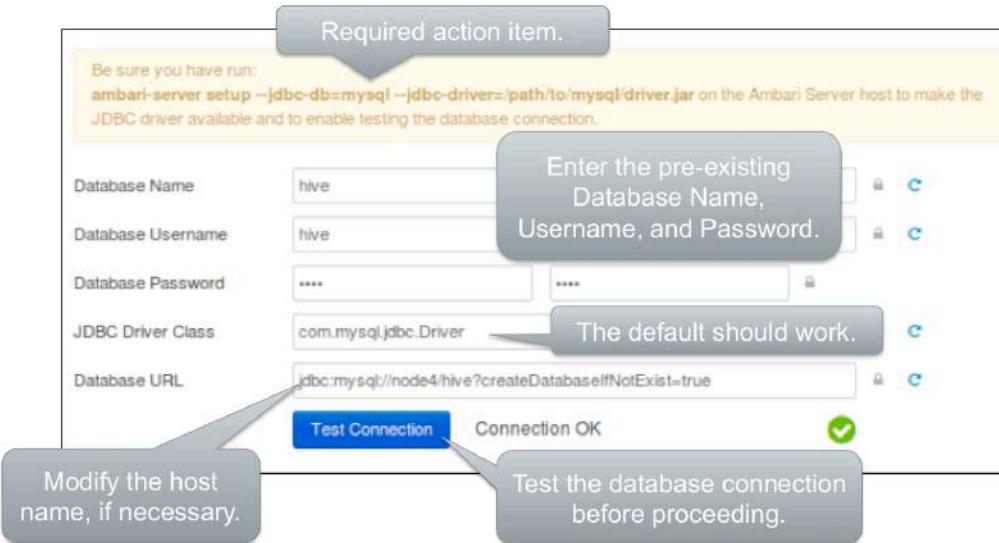
Customizing Services in Ambari Web UI

Hadoop services must typically be customized for each specific cluster installation. Ambari will do some limited customization based on the choices made during installation. For example, configuration properties that include the hostname of the Hive Metastore host will be updated by Ambari based on your choice of a Hive Metastore host in the earlier Assign Masters window.

In addition to the customizations made by Ambari, the Customize Services window enables a user to customize a myriad of configuration settings. In the screen capture, the **Existing MySQL Database** radio button was selected in order to configure Hive to use an existing MySQL database. Additional settings related to this choice are shown next.

Depending on the services selected in the earlier Choose Services window, Ambari might display red alert icons. In the example here, Ambari is alerting the user that the Hive service is missing one or more required configuration settings. In this case, scrolling down in the window reveals that the Hive service requires a database password for the metastore database selected during the installation.

Configuring a MySQL Database



Configuring a MySQL Databases in Ambari Web UI

In the previous screen capture, a choice was made to use a pre-existing MySQL database. To access and use this database requires the configuration of more configuration properties.

In order for Ambari to work with the pre-existing MySQL database, Ambari must be configured with a MySQL driver using the `ambari-server setup` command shown in the screen capture. The required `mysql-connector-java-<version>-bin.jar` file can be downloaded from <http://dev.mysql.com>.

The administrator must know the name of the pre-existing database and enter it here. It is `hive` in the example. The administrator must also know the username and password for the user that has administrative privileges on the database. In the example, the username and password are both `hive`.

The default JDBC Driver Class property for MySQL should work so there is no need to modify it.

If the Database Host hostname was updated then the Database URL will likely already have its hostname updated to reflect the actual hostname of the machine running the MySQL database. You can also manually change the URL if necessary.

Scroll down and click **Next** (not shown) to proceed.

Review and Deploy

The screenshot shows the 'Review' window in the Ambari Web UI. At the top, a message says 'Please review the configuration before installation'. Below this, the configuration details are listed:

- Admin Name :** admin
- Cluster Name :** horton
- Total Hosts :** 3 (0 new)
- Repositories:**
 - redhat6 (HDP-2.3):
http://node1/hdp/HDP-2.3
 - redhat7 (HDP-UTILS-1.1.0.20):
http://node1/hdp/HDP-UTILS-1.1.0.20
 - redhat7 (HDP-2.3):
http://public-repo-1.hortonworks.com/HDP/centos7/x/updates/2.3.2.0
 - redhat7 (HDP-UTILS-1.1.0.20):
http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/centos7
 - suse11 (HDP-2.3):
http://public-repo-1.hortonworks.com/HDP/suse11sp3/x/updates/2.3.2.0
 - suse11 (HDP-UTILS-1.1.0.20):
http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/suse11sp3
- Services:** (List of services is partially visible)

At the bottom right are 'Print' and 'Deploy' buttons.

Reviewing Hive Install Information in Ambari Web UI

The Review window enables you to print the configuration and deploy the new software.

Click **Deploy** when ready to install Hive.

Install, Start, and Test

The screenshot shows the 'Install, Start and Test' window in the Ambari Web UI. It displays the progress of installing services across three hosts: node1, node2, and node3. The progress bar at the top indicates 100 % overall completion.

Host	Status	Message
node1	100%	Success
node2	100%	Success
node3	100%	Success

Below the table, a message says 'Successfully installed and started the services.' and there is a 'Next' button.

Watching Installation Progress in Ambari Web UI

The Install, Start, and Test window displays the progress on each cluster node. If any node fails to install properly, the error displayed in the Message column is a hypertext link. Click the link to get more detailed error information. If you can resolve the issue, you can use Ambari to attempt to install the node again.

Click **Next** when ready to proceed.

Summary Window

The screenshot shows the Ambari Summary window. At the top, it says "Important: You may also need to restart other services for the newly added services to function properly (for example, HDFS and YARN/MapReduce need to be restarted after adding Oozie). After closing this wizard, please restart all services that have the restart indicator next to the service name." Below this, a message box states: "Here is the summary of the install process." Inside the message box, it says: "The cluster consists of 3 hosts. Installed and started services successfully on 3 new hosts. Install and start completed in 3 minutes and 36 seconds." At the bottom right of the window is a green "Complete" button.

Install Summary Window in Ambari Web UI

Read the information in the **Summary** window and then click **Complete**.

Restart Services

The screenshot shows the Ambari Services page. On the left, there is a sidebar with service icons: HDFS (green), MapReduce2 (green), YARN (green), Tez (blue), Hive (green), Pig (blue), ZooKeeper (green), and Ambari Metrics (green). A red box highlights the HDFS, MapReduce2, and YARN services. The main area has tabs for "Summary", "Heatmaps", "Configs", and "Quick Links". A yellow banner at the top right says "Restart Required: 5 Components on 3 Hosts" with a "Restart" button. Below the banner is a "Summary" table with the following data:

Name	Status
NameNode	Started
SecondaryNameNode	Started
DataNodes	3/3 Started

On the right side of the summary table, there are status indicators: "Disk Usage (Remaining) 183.0 GB / 279.4 GB (65.50%)", "Blocks (total) 30", and "Block Errors 0 corrupt / 0 missing / 0 under replicated". A green "No alerts" button is also present.

Restarting Services in the Ambari Web UI

Restart any services indicated in the Ambari Web UI.

At this point Hive has been installed but it is not configured for high availability.

Configuring Hive HA

After Hive has been installed the next step is to configure Hive HA. Use the Ambari Web UI to add one or more Hive metastore services, HiveServer2 instances, and WebHCat servers.

Adding Redundant Metastore Services

The screenshot shows the Ambari Web UI interface. On the left, there's a sidebar with icons for HDFS, MapReduce2, YARN, Tez, Pig, ZooKeeper, and Ambari Metrics. The 'Hive' icon is selected. The main area has tabs for 'Summary' and 'Configs'. Under 'Summary', it lists 'Hive Metastore' (Started), 'HiveServer2' (Started), 'WebHCat Server' (Started), 'HCat Clients' (3 clients installed), and 'Hive Clients' (3 clients installed). To the right is a 'Service Actions' dropdown menu with options like Start, Stop, Restart All, Move Hive Metastore, Move HiveServer2, Move WebHCat Server, Run Service Check, Turn On Maintenance Mode, Add Hive Metastore, Add HiveServer2, and Download Client Configs. A callout bubble points to the 'Add Hive Metastore' option in the menu.

Configure Hive Metastore Service HA in Ambari Web UI

Browse to **Services > Hive** to configure Hive metastore service HA. Click the **Service Actions** button and select **Add Hive Metastore**.

Select a Metastore Service Host

The screenshot shows a 'Confirmation' dialog box. It asks 'Select the host to add Hive Metastore component' and has a dropdown menu with 'node2' selected. Below it is a question 'Are you sure you want to add Hive Metastore?'. At the bottom are 'Cancel' and 'Confirm Add' buttons.

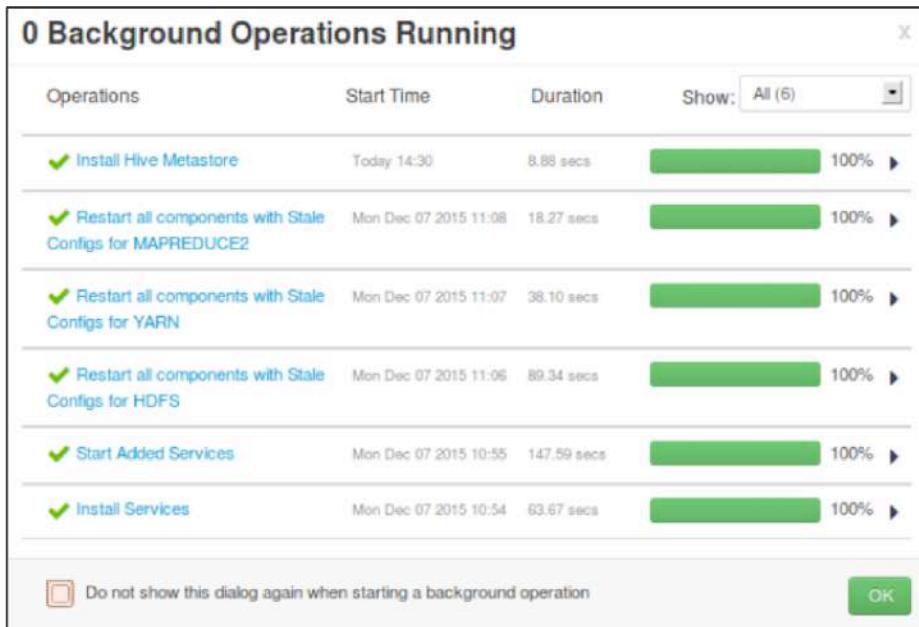
The screenshot shows a second 'Confirmation' dialog box. It states 'Adding Hive Metastore will reconfigure such properties:' followed by a list: '• hive.metastore.uris' and '• templeton.hive.properties'. At the bottom are 'Cancel' and 'OK' buttons.

Selecting a Metastore Service Host in Ambari Web UI

In the first window use the drop-down menu to select a host for the new metastore service. Consider workload and available system resources. Then click **Confirm Add**.

In the next window click **OK** to confirm the configuration.

Monitor Metastore Configuration Progress

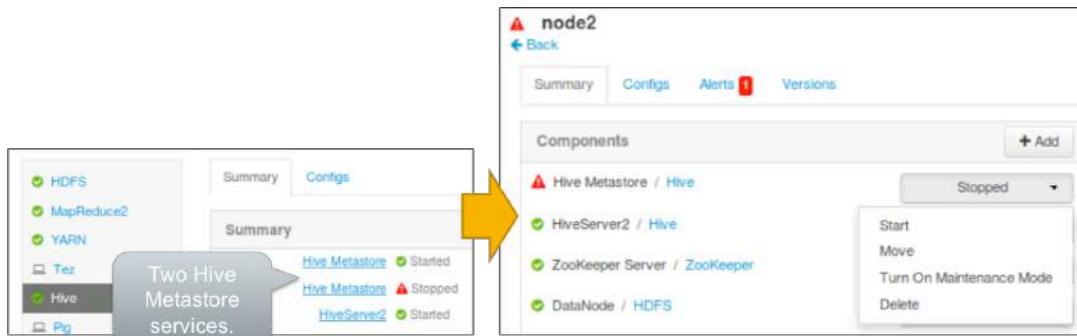


Monitoring Metastore Configuration Progress in Ambari Web UI

Use Background Operations Running window to monitor the configuration progress. If any problems are encountered, the displayed error message is a hypertext link. Click the link to display detailed error information.

When the configuration has finished, click **OK** to continue.

Start the New Metastore Service



Starting a New Metastore Service in Ambari Web UI

Verify that there are two Hive Metastore services. If the new Hive Metastore has not started, click the **Hive Metastore** link. Then find the stopped Hive Metastore in the list, click the **Stopped** menu, and select **Start**.

Restart Services

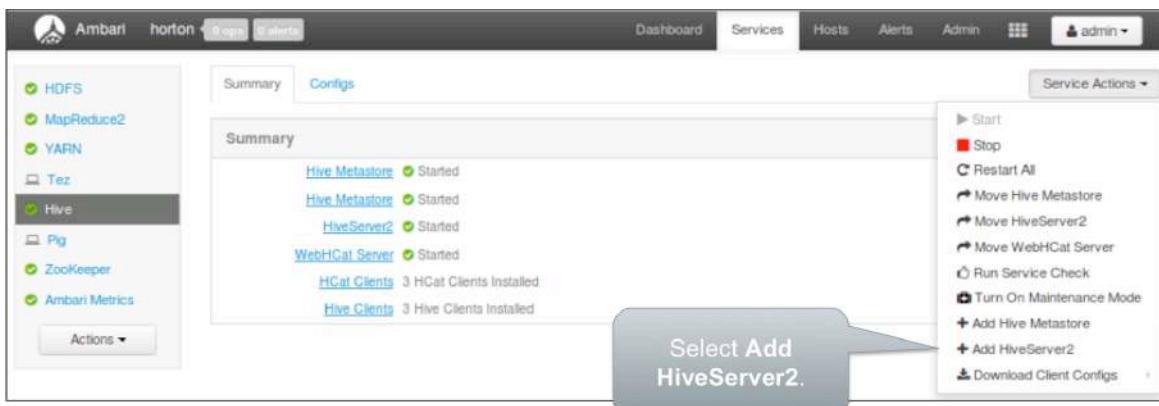


Restarting Service in the Ambari Web UI

Restart any services indicated in the Ambari Web UI.

At this point another Hive metastore service has been configured.

Adding Redundant HiveServer2 Instances



Adding the HiveServer2 Service in Ambari Web UI

Browse to **Services > Hive** to configure HiveServer2 HA. Click the **Service Actions** button and select **Add HiveServer2**.

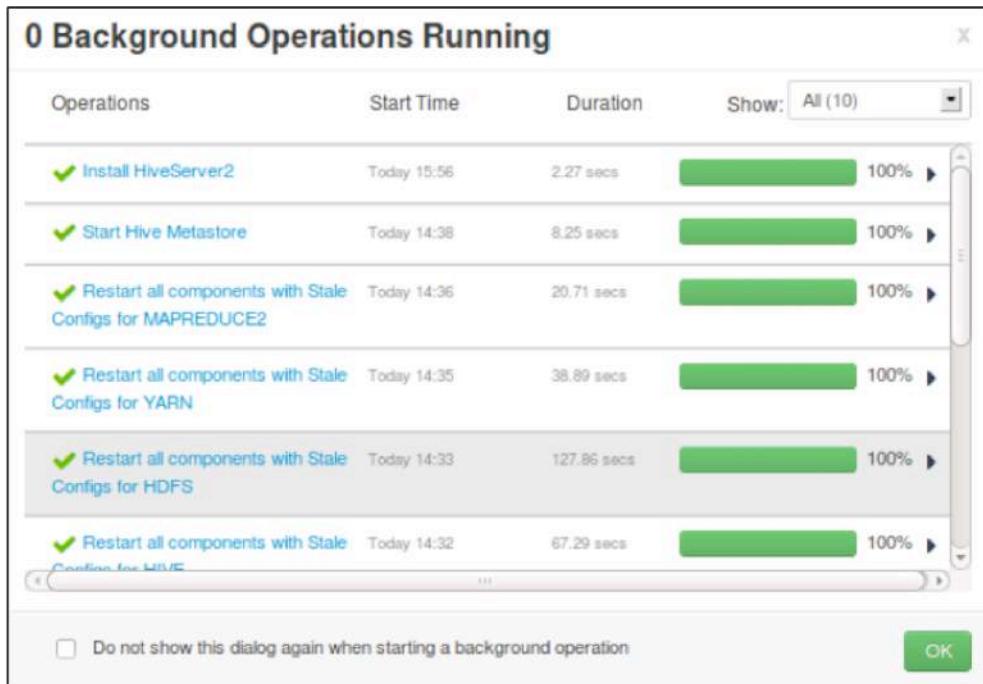
Selecting a HiveServer2 Host



Selecting a HiveServer2 Host in Ambari Web UI

In this window use the drop-down menu to select a host for the new HiveServer2. Consider workload and available system resources. Then click **Confirm Add**.

Monitoring HiveServer2 Configuration Progress



Monitoring HiveServer2 Configuration Progress in Ambari Web UI

Use **Background Operations Running** window to monitor the configuration progress. If any problems are encountered, the displayed error message is a hypertext link. Click the link to display detailed error information.

When the configuration has finished, click **OK** to continue.

Starting the New HiveServer2

The screenshot shows the Ambari Web UI interface. On the left, there is a sidebar with various service icons. A callout bubble points to the "Hive" icon with the text "Two HiveServer2 instances.". The main content area shows the "node2" cluster summary. In the "Components" section, the "HiveServer2 / Hive" component is listed with a red warning icon and the status "Stopped". To the right of the component list, there is a dropdown menu with options: Start, Move, Turn On Maintenance Mode, and Delete.

Starting the New HiveServer2 in Ambari Web UI

Verify that there are two HiveServer2 instances. If the new HiveServer2 has not started, click the **HiveServer2** link. Then find the stopped HiveServer2 in the list, click the **Stopped** menu, and select **Start**.

Adding Redundant WebHCat Servers

Name *	IP Address	Rack	Cores	RAM	Disk Usage	Load Avg	Versions	Components
<input type="checkbox"/> Any	Any	Any	Any	Any	Any	Filter T	Filter T	
<input checked="" type="checkbox"/> node1	172.17.0.2	/default-rack	4 (4)	14.44GB	<div style="width: 100%;"> </div>	1.39	HDP-2.3.0.0-2557	21 Components
<input checked="" type="checkbox"/> node2	172.17.0.3	/default-rack	4 (4)	14.44GB	<div style="width: 100%;"> </div>	1.35	HDP-2.3.0.0-2557	14 Components
<input type="checkbox"/> node3	172.17.0.4	/default-rack	4 (4)	14.44GB	<div style="width: 100%;"> </div>	1.24	HDP-2.3.0.0-2557	12 Components

Configuring WebHCat Servers in Ambari Web UI

The procedure to configure a redundant WebHCat server is different than configuring a redundant HiveServer2 or Hive metastore service. Rather than browse to **Services > Hive**, browse to **Hosts** and click the host where the new WebHCat server should be installed.

Selecting a WebHCat Server

Components
<input checked="" type="checkbox"/> Hive Metastore / Hive
<input checked="" type="checkbox"/> HiveServer2 / Hive

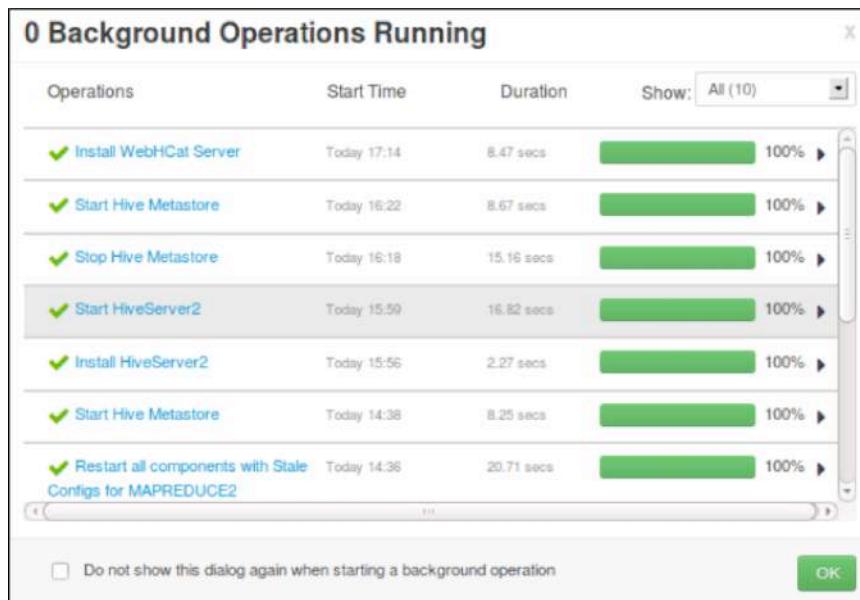
Confirmation

Are you sure you want to add WebHCat Server?

Selecting the WebHCat Server in Ambari Web UI

Click the **Add** menu button and select **WebHCat Server**. Then click **Confirm Add** in the Confirmation window.

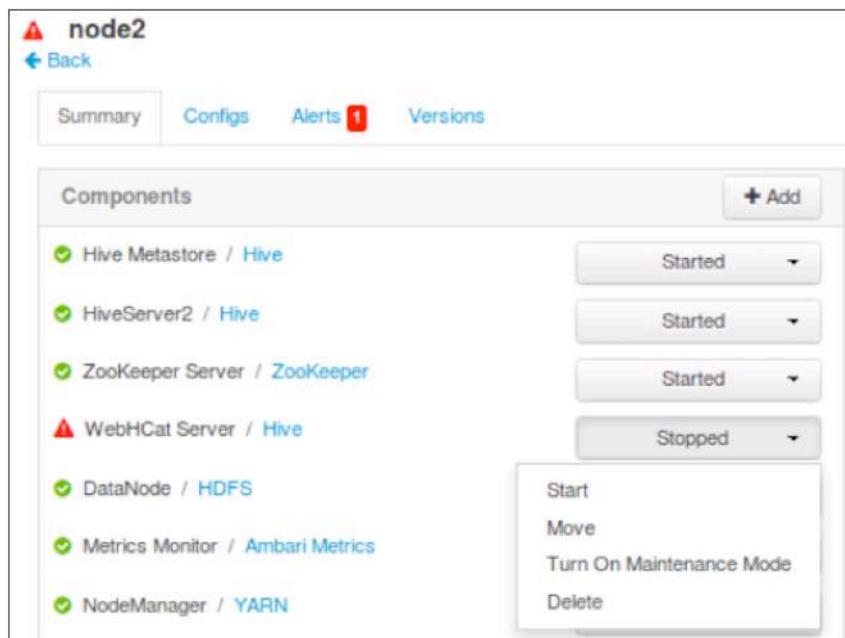
Monitoring WebHCat Configuration Progress



Monitoring WebHCat Configuration Progress in Ambari Web UI

Use Background Operations Running window to monitor the configuration progress. If any problems are encountered, the displayed error message is a hypertext link. Click the link to display detailed error information. When the configuration has finished, click **OK** to continue.

Starting the New WebHCat Server



Starting the New WebHCat Server in Ambari Web UI

If the new WebHCat Server has not started, then find the stopped WebHCat Server in the list, click the **Stopped** menu, and select **Start**.

Verify the Hive HA Configuration

The screenshot shows the Ambari Web UI interface. On the left, there's a sidebar with various service icons. The 'Hive' icon is selected and highlighted in dark grey. In the main content area, the 'Configs' tab is active under the 'Summary' section. A red box highlights the list of services: 'Hive Metastore' (2 instances), 'HiveServer2' (2 instances), and 'WebHCat Server' (2 instances), all of which are listed as 'Started'. Below this, it says 'HCat Clients: 3 HCat Clients Installed' and 'Hive Clients: 3 Hive Clients Installed'. At the top right, there's a 'Service Actions' button and a 'No alerts' indicator.

Verifying Redundant Hive HA Components in Ambari Web UI

The final step in the Ambari Web UI is to verify that there are redundant Hive Metastore, HiveServer2, and WebHCat instances. Browse to **Services > Hive** and check the list of configured services.

Verify Hive HA Access

```
[root@node1 ~]# beeline
WARNING: Use "yarn jar" to launch YARN applications.
Beeline version 1.2.1.2.3.0.0-2557 by Apache Hive
beeline> !connect jdbc:hive2://node1:2181,node2:2181,node3:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2 root hadoop
Connecting to jdbc:hive2://node1:2181,node2:2181,node3:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2
Connected to: Apache Hive (version 1.2.1.2.3.0.0-2557)
Driver: Hive JDBC (version 1.2.1.2.3.0.0-2557)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://node1:2181,node2:2181,node3:2181> show databases;
+-----+-----+
| database_name |
+-----+-----+
| default      |
+-----+-----+
1 row selected (3.699 seconds)
0: jdbc:hive2://node1:2181,node2:2181,node3:2181> !q
Closing: 0: jdbc:hive2://node1:2181,node2:2181,node3:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2
[root@node1 ~]#
```

Using Beeline to Verify Hive HA Access

Use Beeline to confirm Hive HA access. The Beeline connection syntax from the beeline prompt is:

```
!connect jdbc:hive2://<ZooKeeper_quorum>/;serviceDiscoveryMode=zooKeeper;zooKeeperNameSpace=hiveserver2 <username> <password>
```

The illustration shows an actual example of testing Hive HA access using Beeline.

This page left blank intentionally.

Knowledge Check

Use the following questions and answers to assess your understanding of the concepts presented in this lesson.

Questions

- 1) The current Hive client-server components are implemented by Beeline and _____ .
- 2) The _____ service is backed by an underlying relational database.
- 3) The _____ server handles REST API requests to Hive.
- 4) True or false? HiveServer2 support automatic client failover.
- 5) True or false? The Metastore service supports automatic client failover.
- 6) To achieve Hive HA, the HiveServer2, metastore service, and WebHCat must be configured with _____ components.
- 7) In the following Beeline command, what service is running on nodes 1, 2, and 3?

```
!connect jdbc:hive2://  
node1:2181,node2:2181,node3:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNameS  
pace=hiveserver2 <username> <password>
```

Answers

1) The current Hive client-server components are implemented by Beeline and _____.

Answer: HiveServer2

2) The _____ service is backed by an underlying relational database.

Answer: Metastore

3) The _____ server handles REST API requests to Hive.

Answer: WebHCat

4) True or false? HiveServer2 support automatic client failover.

Answer: False

5) True or false? The Metastore service supports automatic client failover.

Answer: True

6) To achieve Hive HA, the HiveServer2, metastore service, and WebHCat must be configured with _____ components.

Answer: Redundant

7) In the following Beeline command, what service is running on nodes 1, 2, and 3?

```
!connect jdbc:hive2://  
node1:2181,node2:2181,node3:2181;/serviceDiscoveryMode=zooKeeper;zooKeeperNameS  
pace=hiveserver2 <username> <password>
```

Answer: ZooKeeper service

Summary

- Hive is a data warehouse infrastructure built on top of Hadoop.
- By default, the HiveServer2, Metastore service, and WebHCat server are single points of failure.
- Hive HA is implemented by configuring redundant HiveServer2, Metastore service, and WebHCat server instances.
- Hive HA provides the benefits of high availability, load balancing, and support for rolling upgrade.
- Installing and configuring Hive HA is a multi-phase process, where each phase consists of multiple steps performed using the Ambari Web UI.
- Hive HA requires HDP 2.2 or later.

Managing Workflows Using Apache Oozie

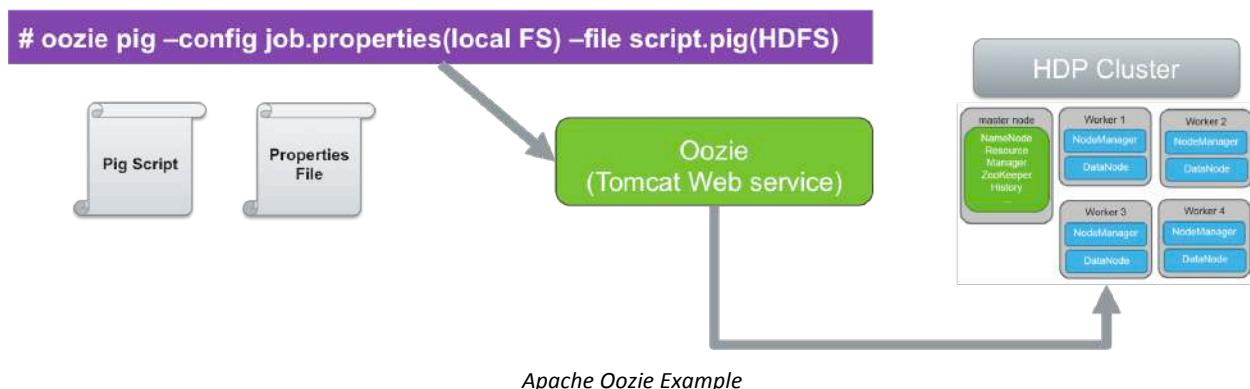
Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Recall the purpose, job types, structure, and benefits of Oozie
- ✓ Install and configure Oozie using Ambari
- ✓ Deploy and manage a sample Oozie workflow

Apache Oozie

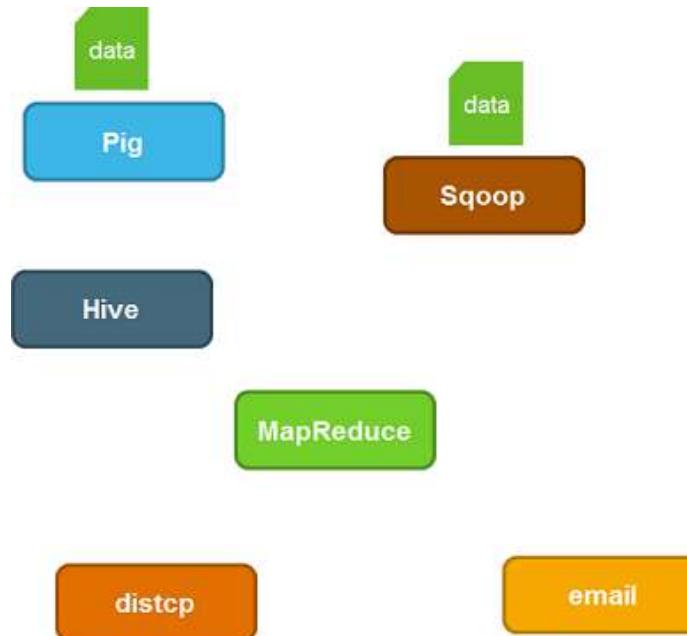
At a basic level, Apache Oozie is a Java Web service, running in Tomcat, that can be used to submit certain types of jobs to an HDP cluster. Hadoop job types that are currently supported include Pig, Hive, Sqoop, and MapReduce (including Streaming). Oozie can also be used to schedule Java applications and various actions, such as distcp, SSH, email, and shell script execution on specific systems. Jobs can be submitted via the CLI, and Oozie also supports both a Java and a web service API. (The CLI actually interfaces with the web service API.)



A basic Oozie example starts with a job to be performed, such as a Pig script, and an available HDP Cluster with Oozie installed, as well as Pig, with the Pig script and all necessary supporting components and libraries having already been copied to HDFS. In order to run this job, a number of variables need to be passed to Oozie – such as the URL of the HDP NameNode, the location (in HDFS) of the script to be executed as well as the supporting libraries, and possibly other information. In addition, the Pig script may contain variables, as defined by the person who developed the script, that need to be supplied at runtime as well. These variables can be passed as part of the CLI command, however it is often more useful to create a properties file (usually named `job.properties`) that contains all necessary variable information for the script to run. This properties file is generally located on the local filesystem, rather than HDFS.

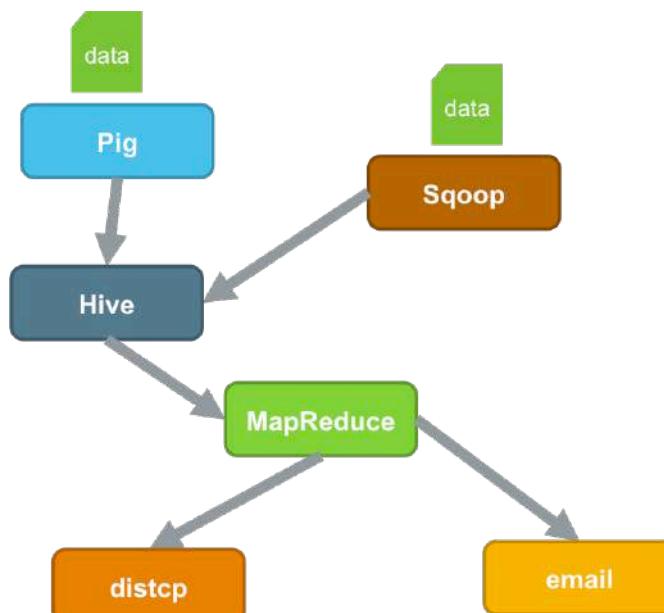
In the example provided, the CLI is used to run the Pig script using Oozie. The location of the properties file on the local filesystem is provided, as well as the location of the Pig script in HDFS. At that point, Oozie then schedules the job on the HDP Cluster by running a MapReduce map-only job to execute the script, and then tracks it to completion.

Oozie Workflow



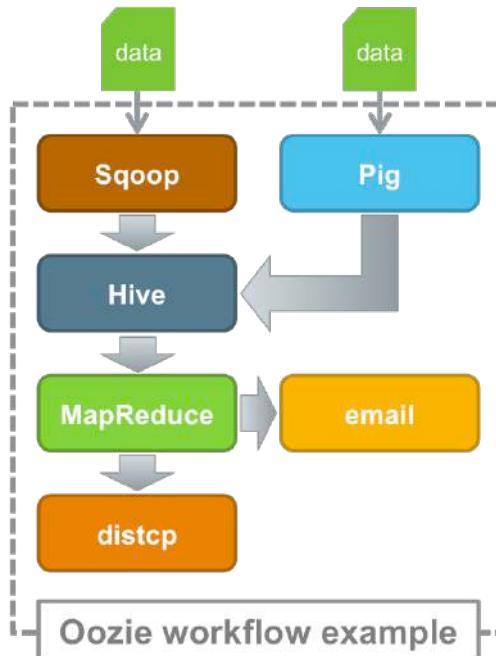
Distinct Oozie Jobs and Actions

The power of Oozie lies in its capability to combine multiple different types of jobs into a sequential, logical unit of work. For example, these six blocks represent six distinct Oozie jobs or actions. The Pig job performs an action on data already in HDFS, and the Sqoop job imports data from a MySQL server. The results of these two jobs need to be sent to Hive, where further manipulation occurs. The results of the Hive job are then used by a MapReduce program which performs calculations, and those results should be sent other places via email and distcp.



Joined to Form a Directed Acyclic Graph

This type of logical structure is known as a **Directed Acyclic Graph**, or **DAG**. Without the ability to combine these jobs via Oozie, the execution of this process would be manual, tedious, potentially error-prone, and possibly require the development of custom applications for tracking and management. However, Oozie is specifically designed as a ready-to-go, server-based workflow engine. This means that the logical flow of the various jobs and actions can be combined into a single set of instructions, via the creation of an XML file (usually named workflow.xml), and the entire DAG will be executed as described as though it was a single, logical unit of work.



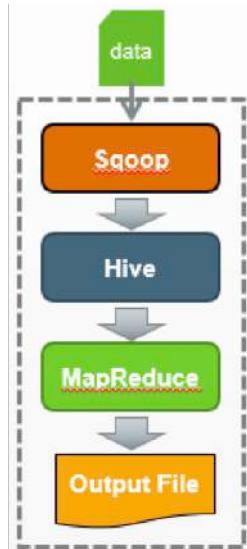
An Oozie Workflow

This is known as a workflow, and represents a core building block in most Oozie projects. An Oozie workflow can be submitted as a single job via Oozie.

Oozie Coordinator

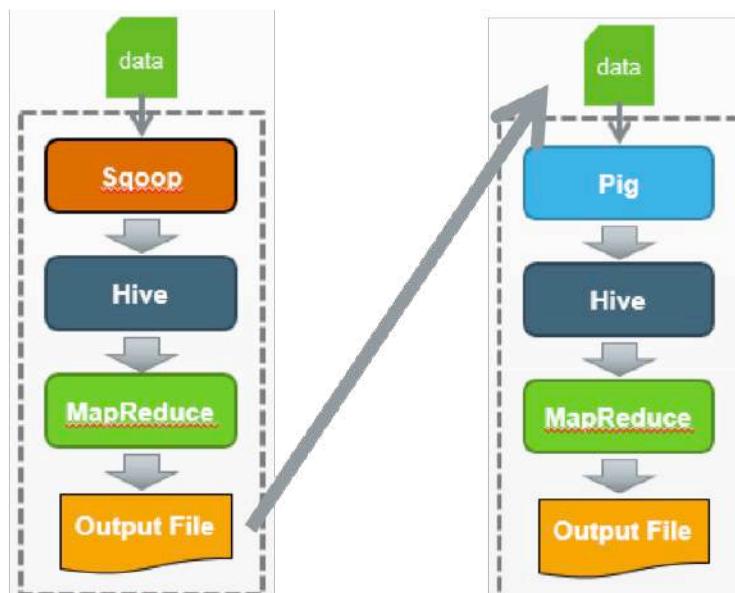
The power of Oozie is further expanded with the inclusion of a coordinator function. An Oozie coordinator, which is launched as an Oozie job via an XML file just like a workflow would be, allows a user to automate the execution of a workflow based on time, some external event, or the availability of specific data files. This can be used to create a chain of workflows in which the output of one workflow that runs at one frequency can be used as the initial input for another workflow that runs at a different frequency.

This logical chaining together of multiple, interdependent workflows is referred to as a data application pipeline.



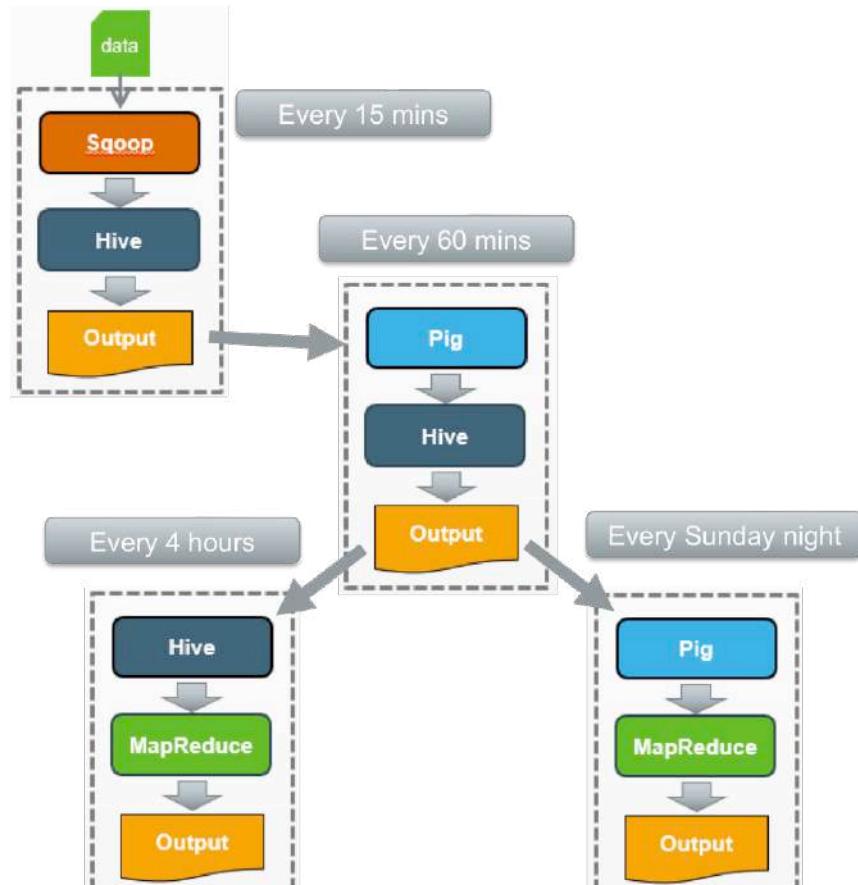
First Oozie Workflow

In the example provided, the first workflow is scheduled to pull data from a MySQL database every 15 minutes and process that data into an output file. Then, a second workflow, which runs every 60 minutes, is configured to examine the last four files generated by the first outflow and perform additional operations. The Oozie coordinator XML file (usually named `coordinator.xml`) can specify that a certain file must be available before the second workflow is executed. Additional checks to ensure data integrity can be performed, and should be configured by the data application pipeline developer.



Two Workflows that Form a Data Application Pipeline

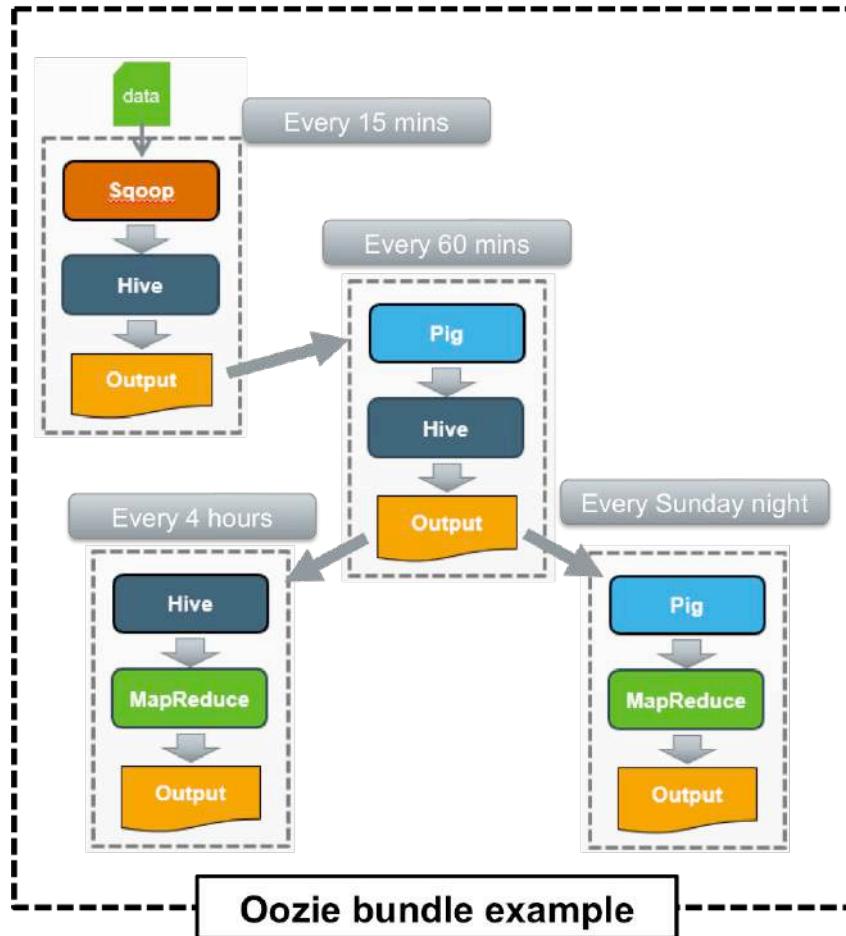
Furthermore, just like a workflow allows multiple jobs to be arranged and executed as a logical DAG structure, so too can multiple workflows be arranged as a logical DAG via multiple coordinators with the proper timing and data checks in place.



Multiple Oozie Workflows Executed as a Logical DAG Structure

It is important to note, however, that each coordinator (defined by an XML file) can only contain a single workflow.

Oozie Bundle



Bundling Multiple Oozie Coordinators

Oozie bundles further abstract the workflow concept by allowing for multiple coordinators (and thus their workflows) to be managed as a single logical unit. A bundle, which – like other Oozie job types - is defined via an XML file, allows for the execution of a set of coordinators at a specified time in the future, as well as suspending/resume/stop/rerun functions via a single command.

Oozie Benefits

In summary, Oozie enables HDP users to build complex data transformations and operations by combining various types of HDP jobs into a single, logical workflow. It furthermore enables automation of workflow execution, which can be coordinated into a logical data application pipeline. This data application pipeline can itself be bundled into a single manageable unit.

Oozie thus results in lower costs via:

- Better control over complex, multi-stage jobs,
- Automation and scheduling of repetitive tasks, and
- Reduced manual labor and human error.

Oozie Architecture

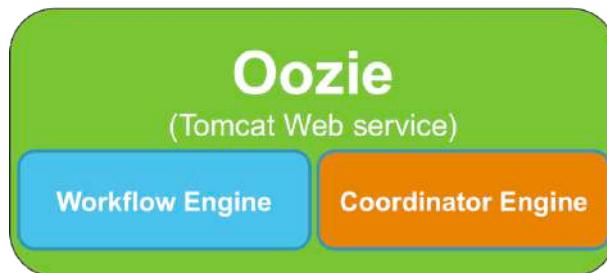
Oozie is comprised of three basic components.

The first component is a workflow engine, which runs and manages workflows. It is worth noting that every job Oozie runs is managed as a workflow, even if it is executed as a single job type without a workflow definition file.



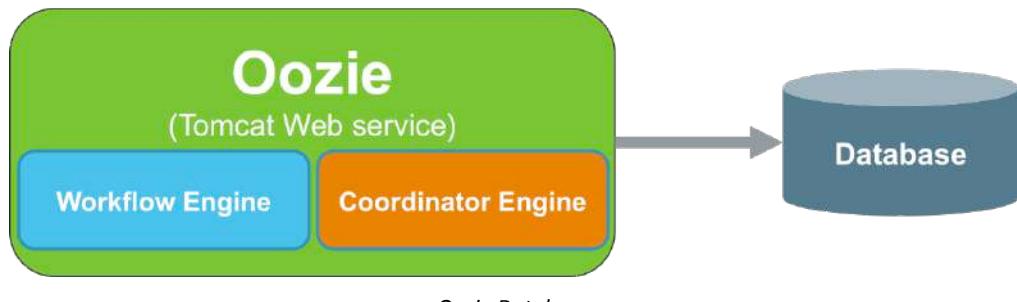
Oozie Workflow Engine

The second component is the coordinating engine scheduler, which runs workflow jobs based on predefined schedules (fixed or cron intervals), data availability, or external event triggers.



Oozie Coordinator Engine

The third component is the Oozie database, which stores workflow definition, variables, state information, and other data as may be required based on the configuration of the tasks Oozie is performing.

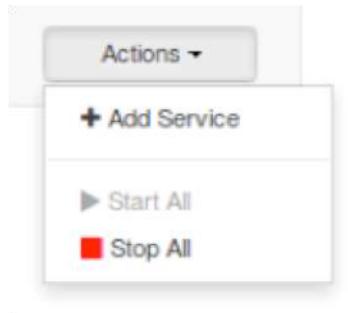


Oozie Database

By default Oozie will install and use Derby as its database, however this is not recommended for use in production systems and will not allow for Oozie HA configuration. Oozie can also be configured to use an existing MySQL, PostgreSQL, or Oracle database. Any of these are supported for production purposes, and will allow for Oozie HA configuration.

It is the job of an HDP developer to define and configure Oozie workflows, coordinators, and bundles. As an HDP administrator, it is important to understand how Oozie works, how to install and configure Oozie, how to test Oozie functionality, and finally how to monitor and manage resources used by Oozie jobs.

Installing and Configuring Oozie Using Ambari



Add Service Window in Ambari Web UI

To install Oozie, log in to the Ambari Web UI as a user with at least Operator permissions. Click the Services tab to reveal the service Actions menu. Click the Actions menu and select Add Service. The Add Service wizard opens.

Choosing Services

Choose Services

Choose which services you want to install on your cluster.

Service	Version	Description
<input type="checkbox"/> HDFS	2.7.1.2.3	Apache Hadoop Distributed File System
<input checked="" type="checkbox"/> YARN + MapReduce2	2.7.1.2.3	Apache Hadoop NextGen MapReduce (YARN)
<input type="checkbox"/> Tez	0.7.0.2.3	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
<input type="checkbox"/> Hive	1.2.1.2.3	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
<input type="checkbox"/> HBase	1.1.1.2.3	A Non-relational distributed database, plus Phoenix, a high performance SQL layer for low latency applications.
<input type="checkbox"/> Pig	0.15.0.2.3	Scripting platform for analyzing large datasets
<input type="checkbox"/> Sqoop	1.4.6.2.3	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
<input checked="" type="checkbox"/> Oozie	4.2.0.2.3	System for workflow coordination and execution of Apache Hadoop jobs. This also includes the installation of the optional Oozie Web Console which relies on and will install the ExtJS Library .

Choosing Services in Ambari Web UI

Select the Oozie service. Scroll down and click **Next** (not shown) when ready to continue.

Assigning Masters

Assign Masters

Assign master components to hosts you want to run them on.

SNameNode:	node1 (14.4 GB, 4 cores)	node1 (14.4 GB, 4 cores)
NameNode:	node1 (14.4 GB, 4 cores)	node2 (14.4 GB, 4 cores)
History Server:	node1 (14.4 GB, 4 cores)	node3 (14.4 GB, 4 cores)
App Timeline Server:	node1 (14.4 GB, 4 cores)	node2 (14.4 GB, 4 cores)
ResourceManager:	node1 (14.4 GB, 4 cores)	node3 (14.4 GB, 4 cores)
Hive Metastore:	node1 (14.4 GB, 4 cores)	node2 (14.4 GB, 4 cores)
Hive Metastore:	node2 (14.4 GB, 4 cores)	ZooKeeper Server
WebHCat Server:	node1*	Oozie Server
WebHCat Server:	node2*	ZooKeeper Server
Oozie Server:	node1 (14.4 GB, 4 cores)	ZooKeeper Server
ZooKeeper Server:	node1 (14.4 GB, 4 cores)	ZooKeeper Server

Assigning Masters in Ambari Web UI

Ambari provides an initial layout of service master components. Use the **Assign Masters** window either to confirm or move the master components. This is when information about workloads and resource capacity gained from pre-planning, or pre-testing with a pilot cluster, is very helpful. The resource capacity of each host must be considered along with service redundancy.

Use the drop-down menu next to each component to move the master component to another cluster node, as necessary. Click **Next** (not shown) to proceed.

Assigning Slaves and Clients

Assign Slaves and Clients

Assign slave and client components to hosts you want to run them on.
Hosts that are assigned master components are shown with *.
"Client" will install Oozie Client

Host	all none	all none	all none	all none
node1*	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node2*	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node3*	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client

Show: 25 1 - 3 of 3 ⌂ ⌃ ⌁ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌊ ⌋

[← Back](#) [Next →](#)

Assigning Slaves and Clients in Ambari Web UI

Use the **Assign Slaves and Clients** window to choose where to run service worker components and Hadoop client software.

The DataNode is the HDFS service worker component and is already installed on all nodes in the screen capture.

An HDFS NFS Gateway machine can be used as a method to access HDFS. The HDFS NFS Gateway is described in another lesson.

The NodeManager is the YARN service worker component and is already installed on all nodes in the screen capture.

Client represents the Hadoop client software. Client software is used by user and application clients to access cluster services and resources. For example, client software is used to access HDFS storage, run YARN jobs, or in this specific case, submit Oozie jobs. Client software is commonly installed on cluster nodes. Client software is also commonly installed on utility machines used as gateway machines to access cluster resources.

Select all nodes from which you expect to submit Oozie jobs and click **Next**.

Customizing Services

Assign Slaves and Clients

Assign slave and client components to hosts you want to run them on.
Hosts that are assigned master components are shown with *.
"Client" will install Oozie Client

Host	all none	all none	all none	all none
node1*	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node2*	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node3*	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client

Show: 25 1 - 3 of 3 Back Next

Customizing Service in Ambari Web UI

Hadoop services must typically be customized for each specific cluster installation. Ambari will do some limited customization based on the choices made during installation. For example, configuration properties that include the hostname of the Oozie Server will be updated by Ambari based on your choice of a Oozie Server host in the earlier Assign Masters window.

In addition to the customizations made by Ambari, the Customize Services window enables a user to customize a myriad of configuration settings. In the screen capture, the Existing MySQL Database radio button was selected in order to configure Oozie to use an existing MySQL database. Additional settings related to this choice are shown next.

Depending on the services selected in the earlier Choose Services window, Ambari might display red alert icons. In the example here, Ambari is alerting the user that the Oozie service is missing one or more required configuration settings. In this case, scrolling down in the window reveals that the Oozie service requires a database password for the pre-existing database selected during the installation.

Configuring a MySQL Driver

Be sure you have run:
ambari-server setup --jdbc-db=mysql --jdbc-driver=/path/to/mysql driver.jar on the Ambari Server host to make the JDBC driver available and to enable testing the database connection.

Database Name	oozie		
Database Username	oozie		
Database Password	*****	*****	
JDBC Driver Class	com.mysql.jdbc.Driver		
Database URL	jdbc:mysql://node4/oozie		
<input type="button" value="Test Connection"/> Connection OK			
Oozie Data Dir	/hadoop/oozie/data		

Configuring the MySQL Driver in Ambari Web UI

In the previous screen capture, a choice was made to use a pre-existing MySQL database. To access and use this database requires the configuration of more configuration properties.

In order for Ambari to work with the pre-existing MySQL database, Ambari must be configured with a MySQL driver using the `ambari-server setup` command shown in the screen capture. The required `mysql-connector-java-<version>-bin.jar` file can be downloaded from <http://dev.mysql.com>.

The administrator must know the name of the pre-existing database and enter it here. It is oozie in the example. The administrator must also know the user name and password for the user that has administrative privileges on the database. In the example, the user name and password are both oozie.

The default JDBC Driver Class property for MySQL should work so there is no need to modify it.

The Database URL will likely need to have its hostname updated to reflect the actual hostname of the machine running the MySQL database.

Scroll down and click **Next** (not shown) to proceed.

Reviewing and Deploying

Review

Please review the configuration before installation.

Admin Name : admin
Cluster Name : horton
Total Hosts : 3 (0 new)

Repositories:

- redhat6 (HDP-2.3):
http://node1/hdp/HDP-2.3
- redhat7 (HDP-UTILS-1.1.0.20):
http://node1/hdp/HDP-UTILS-1.1.0.20
- redhat7 (HDP-2.3):
http://public-repo-1.hortonworks.com/HDP/centos7/2.x/updates/2.3.2.0
- redhat7 (HDP-UTILS-1.1.0.20):
http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/centos7
- suse11 (HDP-2.3):
http://public-repo-1.hortonworks.com/HDP/suse11sp3/2.x/updates/2.3.2.0
- suse11 (HDP-UTILS-1.1.0.20):
http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/suse11sp3

Services:

None

[Back](#) [Print](#) [Deploy](#)

Warning

HDFS and YARN services will be restarted during installation.

[Cancel](#) [Proceed Anyway](#)

Reviewing and Deploying in Ambari Web UI

The **Review** window enables you to print the configuration and deploy the new software. Click **Deploy** when ready to install Oozie.

Click **Proceed Anyway** after you have read the message in the Warning window.

Installing, Starting and Testing

Install, Start and Test

Please wait while the selected services are installed and started.

100 % overall

Host	Status	Message
node1	100%	Success
node2	100%	Success
node3	100%	Success

3 of 3 hosts showing - [Show All](#)

Show: [All \(3\)](#) [In Progress \(0\)](#) [Warning \(0\)](#) [Success \(3\)](#) [Fail \(0\)](#)

Successfully installed and started the services.

[Next →](#)

Installing, Starting , and Testing in Ambari Web UI

The **Install, Start, and Test** window displays the progress on each cluster node. If any node fails to install properly, the error displayed in the Message column is a hypertext link. Click the link to get more detailed error information. If you can resolve the issue, you can use Ambari to attempt to install the node again.

Click **Next** when ready to proceed.

Completing the Install

Summary

Important: You may also need to restart other services for the newly added services to function properly (for example, HDFS and YARN/MapReduce need to be restarted after adding Oozie). After closing this wizard, please restart all services that have the restart indicator  next to the service name.

Here is the summary of the install process.

The cluster consists of 3 hosts
Installed and started services successfully on 3 new hosts
Install and start completed in 9 minutes and 40 seconds

Complete →

Summary Window in Ambari Web UI

Read the information in the **Summary** window and then click **Complete**.

Restarting Services

Service Actions ▾

Summary **Heatmaps** **Configs** **Quick Links ▾**

HDFS  **MapReduce2**  **YARN**  **Tez** **Hive** **Pig** **Oozie**

Restart Required: 3 Components on 3 Hosts **Restart ↗...**

Summary **No alerts**

NameNode  Started Disk Usage (Remaining) 168.8 GB / 279.4 GB (60.41%)
cNameNode  Started Blocks (total) 538

Restarting Services in Ambari Web UI

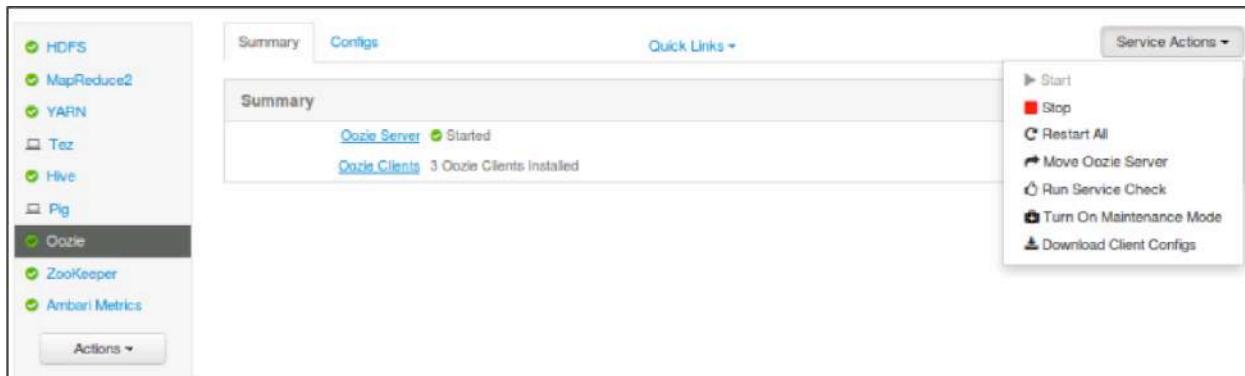
Restart any services indicated in the Ambari Web UI.

At this point Oozie has been installed.

Managing Oozie with Ambari

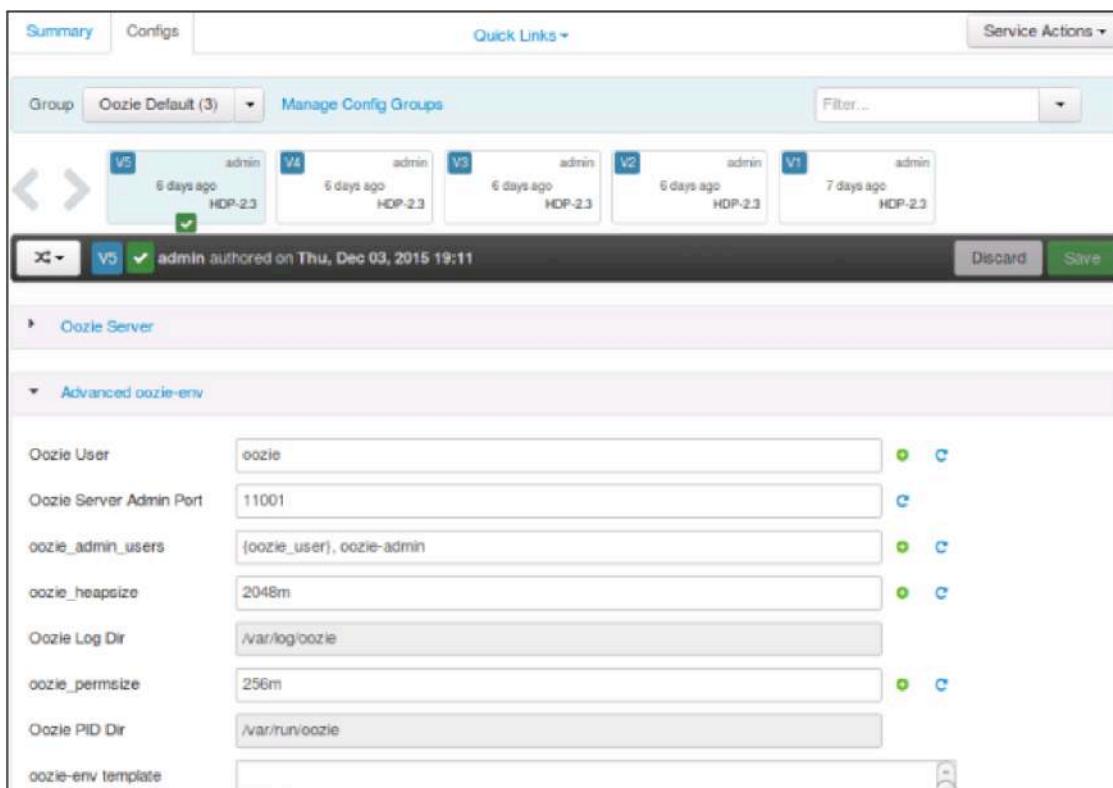
After installation, Oozie can be managed in Ambari by going to **Services > Oozie**. Two tabs are available: **Summary** and **Configs**. In addition, a **Service Actions** button allows for management of the Oozie service, including moving it from one HDP node to another.

Managing Workflows Using Apache Oozie



Summary Tab for Oozie in Ambari Web UI

The **Summary** tab provides information on the current Oozie status and number of Oozie clients in the cluster.



Configs Tab for Oozie in Ambari Web UI

The **Configs** tab contains settings including server database and service configuration properties.

Reading Oozie Log Files

```
[root@node2 oozie]# pwd
/var/log/oozie
[root@node2 oozie]# ls
catalina.2015-12-02.log          oozie.log.2015-12-03-05  oozie.log.2015-12-04-20  oozie.log.2015-12-06-11  oozie.log.2015-12-08-02
oozie.log.2015-12-03-06          oozie.log.2015-12-04-21  oozie.log.2015-12-06-12  oozie.log.2015-12-08-03
oozie.log.2015-12-03-07          oozie.log.2015-12-04-22  oozie.log.2015-12-06-13  oozie.log.2015-12-08-04
oozie.log.2015-12-03-08          oozie.log.2015-12-04-23  oozie.log.2015-12-06-14  oozie.log.2015-12-08-05
oozie.log.2015-12-03-09          oozie.log.2015-12-05-00  oozie.log.2015-12-06-15  oozie.log.2015-12-08-06
oozie.log.2015-12-03-10          oozie.log.2015-12-05-01  oozie.log.2015-12-06-16  oozie.log.2015-12-08-07
oozie.log.2015-12-03-11          oozie.log.2015-12-05-02  oozie.log.2015-12-06-17  oozie.log.2015-12-08-08
oozie.log.2015-12-03-12          oozie.log.2015-12-05-03  oozie.log.2015-12-06-18  oozie.log.2015-12-08-09
oozie.log.2015-12-03-13          oozie.log.2015-12-05-04  oozie.log.2015-12-06-19  oozie.log.2015-12-08-10
oozie.log.2015-12-03-14          oozie.log.2015-12-05-05  oozie.log.2015-12-06-20  oozie.log.2015-12-08-11
oozie.log.2015-12-03-15          oozie.log.2015-12-05-06  oozie.log.2015-12-06-21  oozie.log.2015-12-08-12
oozie.log.2015-12-03-16          oozie.log.2015-12-05-07  oozie.log.2015-12-06-22  oozie.log.2015-12-08-13
oozie.log.2015-12-03-17          oozie.log.2015-12-05-08  oozie.log.2015-12-06-23  oozie.log.2015-12-08-14
oozie.log.2015-12-03-18          oozie.log.2015-12-05-09  oozie.log.2015-12-07-00  oozie.log.2015-12-08-15
oozie.log.2015-12-03-19          oozie.log.2015-12-05-10  oozie.log.2015-12-07-01  oozie.log.2015-12-08-16
oozie.log.2015-12-03-20          oozie.log.2015-12-05-11  oozie.log.2015-12-07-02  oozie.log.2015-12-08-17
oozie.log.2015-12-03-21          oozie.log.2015-12-05-12  oozie.log.2015-12-07-03  oozie.log.2015-12-08-18
oozie.log.2015-12-03-22          oozie.log.2015-12-05-13  oozie.log.2015-12-07-04  oozie.log.2015-12-08-19
oozie.log.2015-12-03-23          oozie.log.2015-12-05-14  oozie.log.2015-12-07-05  oozie.log.2015-12-08-20
oozie.log.2015-12-04-00          oozie.log.2015-12-05-15  oozie.log.2015-12-07-06  oozie.log.2015-12-08-21
oozie.log.2015-12-04-01          oozie.log.2015-12-05-16  oozie.log.2015-12-07-07  oozie.log.2015-12-08-22
oozie.log.2015-12-04-02          oozie.log.2015-12-05-17  oozie.log.2015-12-07-08  oozie.log.2015-12-08-23
oozie.log.2015-12-04-03          oozie.log.2015-12-05-18  oozie.log.2015-12-07-09  oozie.log.2015-12-09-00
oozie.log.2015-12-04-04          oozie.log.2015-12-05-19  oozie.log.2015-12-07-10  oozie.log.2015-12-09-01
oozie.log.2015-12-04-05          oozie.log.2015-12-05-20  oozie.log.2015-12-07-11  oozie.log.2015-12-09-02
```

Oozie Log File

If an administrator needs to collect log file information, the files can be located on the Oozie server machine under the `/var/log/oozie` directory.

Deploying and Managing Oozie Jobs

An HDP administrator should know how to deploy, manage, and monitor Oozie jobs.

Deploying and Executing a Workflow

Before a workflow job can be executed, as a first step the workflow XML file and all other files necessary to run components of the workflow must be deployed to HDFS. This might include JAR files for Map/Reduce jobs, shells for streaming Map/Reduce jobs, native libraries, Pig scripts, and other resource files. These files are often packaged together as a ZIP file and referred to as a workflow application.

The second component that must be in place is the properties file (typically named `job.properties`) which contains variables and values needed by Oozie and by the workflow application. As a second step, this file is copied to the local file system from which the job will be executed.

The job is then ready to be executed. The third step, then is to launch the job. As an administrator, jobs will usually be launched via command line, however developers can also use API calls to run and manage Oozie jobs. The command to execute an Oozie job is:

```
# oozie job -oozie http://<FQDN of Oozie server>:11000/oozie -config <local path to properties file> -run
```

For example, in our lab environment the following command might be run, depending on your configuration:

```
# oozie job -oozie http://node2:11000/oozie -config /usr/user1/oozie-examples/job.properties -run
```

```
[yarn@node1 map-reduce]# pwd  
/usr/yarn/examples/apps/map-reduce  
[yarn@node1 map-reduce]# oozie job -oozie http://node2:11000/oozie -config job.properties -run  
job: 0000054-151203191211755-oozie-oozi-W  
[yarn@node1 map-reduce]#
```

Successful Oozie Job Submission Returns a Job ID

Job Properties File Configuration

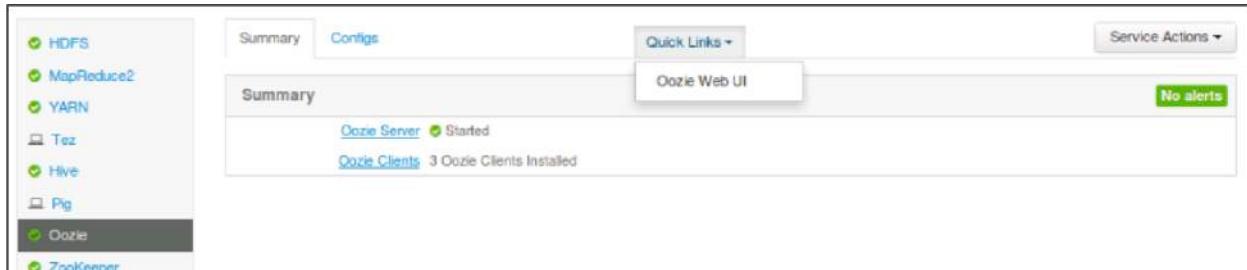
```
nameNode=hdfs://node1:8020  
jobTracker=node1:8021  
queueName=default
```

Job Properties File Values

While most values in a job properties file should be set by an HDP developer, occasionally an administrator may need to edit a couple of properties in a job properties file to account for actual cluster settings. The most common edits to make are to specify the HDFS NameNode and the YARN queue in which the workflow job should execute.

Note: the Job Tracker server is no longer a component of HDP.

Oozie Web UI



Oozie Quick Links Menu

Oozie Web UI (also known as the Oozie Web Console) can be reached by clicking on the Oozie service in **Ambari > Quick Links > Oozie Web UI**.

The screenshot shows the Oozie Web UI with the 'Active Jobs' tab selected. It displays a table with one row of data:

Job Id	Name	Status	Run	User	Group	Created	Started	Last Modified	Ended
1 0000054-151203191211755-oozie-oozi-W	map-reduce-wf	RUNNING	0	yarn		Wed, 09 Dec 2015 18:25:31 G...	Wed, 09 Dec 2015 18:25:31 G...	Wed, 09 Dec 2015 18:25:31 G...	

The top navigation bar includes tabs for Workflow Jobs, Coordinator Jobs, Bundle Jobs, System Info, Instrumentation, and Settings. The status bar at the bottom right shows 'Server version [4.2.0.2.3.0.0-2557]'.

Oozie Web UI

This opens a web page that contains tabs for viewing information on workflow jobs, coordinator jobs, bundle jobs, and additional information on the system.

Managing Workflows Using Apache Oozie

The screenshot shows the 'Job Info' tab of the Oozie Web UI. The job details are as follows:

Job Id:	0000054-151203191211755-oozie-oozi-W
Name:	map-reduce-wf
App Path:	hdfs://node1.8020/user/yam/examples/examples/apps/map-reduce/work
Run:	0
Status:	RUNNING
User:	yarn
Group:	
Parent Coord:	
Create Time:	Wed, 09 Dec 2015 18:25:31 GMT
Start Time:	Wed, 09 Dec 2015 18:25:31 GMT
Last Modified:	Wed, 09 Dec 2015 18:25:31 GMT
End Time:	

Below the details, there is a table titled 'Actions' showing two actions:

Action Id	Name	Type	Status	Transition	StartTime	EndTime
1 0000054-151203191211755-oozie-oozi-W@start	start	START	OK	mr-node	Wed, 09 Dec 2015 18:25:31 G...	Wed, 09 Dec 2015 18:25:31 G...
2 0000054-151203191211755-oozie-oozi-W@mr-node	mr-node	map-reduce	PREP			

The Job ID Tab in the Oozie Web UI

Double-clicking on a **job ID** will open a window inside the page that contains additional information on the workflow job, including individual actions that are running or have been run. Additional tabs include views into the job definition, configuration, log, error log, audit log, and a visual representation of the DAG for this workflow.

The screenshot shows the 'Action Info' tab of the Oozie Web UI. The action details are as follows:

Name:	mr-node
Type:	map-reduce
Transition:	
Start Time:	
End Time:	
Status:	PREP
Error Code:	
Error Message:	
External ID:	
External Status:	
Console URL:	View
Tracker URI:	

Double-clicking on an **action ID** will open an additional window that contains information about that action, its configuration, and URLs of any child jobs.

Command Line Administration

While the Oozie Web UI can be used to monitor and view information on Oozie jobs, management of those jobs requires use of the command line. Common administration tasks might include:

- Job management, including run, dry run, rerun, suspend, resume, and kill
- Change values for coordinator and bundle jobs
- Gather information and check status (similar to Oozie Web UI information)
- Check status and display properties of Oozie system environment

Reference

A complete list of command line options for Oozie 4.2.0 (current as of the time of this writing) can be found here:

http://oozie.apache.org/docs/4.2.0/DG_CommandLineTool.html

Example: Oozie Job Info

```
[yarn@node1 map-reduce]# oozie job -oozie http://node2:11000/oozie -info 0000054-151203191211755-oozie-oozi-W
Job ID : 0000054-151203191211755-oozie-oozi-W
-----
Workflow Name : map-reduce-wf
App Path      : hdfs://node1:8020/user/yarn/examples/examples/apps/map-reduce/workflow.xml
Status        : RUNNING
Run          : 0
User          : yarn
Group         : -
Created       : 2015-12-09 18:25 GMT
Started       : 2015-12-09 18:25 GMT
Last Modified : 2015-12-09 19:06 GMT
Ended         : -
CoordAction ID: -

Actions
-----
ID           Status   Ext ID      Ext Status Err Code
0000054-151203191211755-oozie-oozi-W:@start:    OK      -          OK      -
0000054-151203191211755-oozie-oozi-W@mr-node     START_RETRY- -          -          JA006
```

Oozie CLI Job Info Example

For example, a user can use the `-info` option to display information about an oozie job. The syntax is as follows:

`oozie job -oozie http://<serverFQDN>:11000/oozie -info <job ID>`

Note that this output matches what is displayed in the Oozie Web UI.

The syntax to kill an Oozie job, using the `-kill` option, is as follows:

This page left blank intentionally.

Knowledge Check

Use the following questions and answers to assess your understanding of the concepts presented in this lesson.

Questions

- 1) What four types of Hadoop jobs does Oozie currently support?
- 2) What three job properties file components are sometimes configured by an administrator prior to running an Oozie job?
- 3) A collection of HDP jobs that can be executed as a single logical unit is called a _____?
- 4) True or False: The mechanism that assembles multiple workflows into a single logical unit is called the coordinator.
- 5) Name the three basic components of an Oozie server.
- 6) Name the four options for the Oozie server database.
- 7) What gets returned when an Oozie job is successfully run?

Answers

- 1) What four types of Hadoop jobs does Oozie currently support?

Answer: MapReduce, Sqoop, Pig, and Hive

- 2) What two job properties file components are sometimes configured by an administrator prior to running an Oozie job?

Answer: nameNode and queueName

- 3) A collection of HDP jobs that can be executed as a single logical unit is called a _____?

Answer: Workflow

- 4) True or False: The mechanism that assembles multiple workflows into a single logical unit is called the coordinator.

Answer: False. The coordinator can only define parameters around a single workflow. Assembling multiple workflows requires managing them via coordinator definitions and configuring them as part of an Oozie bundle.

- 5) Name the three basic components of an Oozie server.

Answer: Workflow engine, coordinator engine, and database.

- 6) Name the four options for the Oozie server database.

Answer: Derby, MySQL, PostgreSQL, and Oracle.

- 7) What gets returned when an Oozie job is successfully run?

Answer: Job ID

Summary

- Oozie allows for the combination of multiple types of Hadoop jobs and cluster actions to be configured into a single, logical unit of work known as a workflow.
- Workflows can be scheduled and chained into data application pipelines using Oozie's coordinator engine.
- Multiple Oozie coordinated workflows can be combined into a single, logical unit of work known as a bundle.
- Oozie is comprised of a workflow engine, coordinator engine, and a database.
- Oozie and its jobs can be managed via CLI or API calls.
- Information on Oozie jobs can be viewed via the Oozie Web UI.

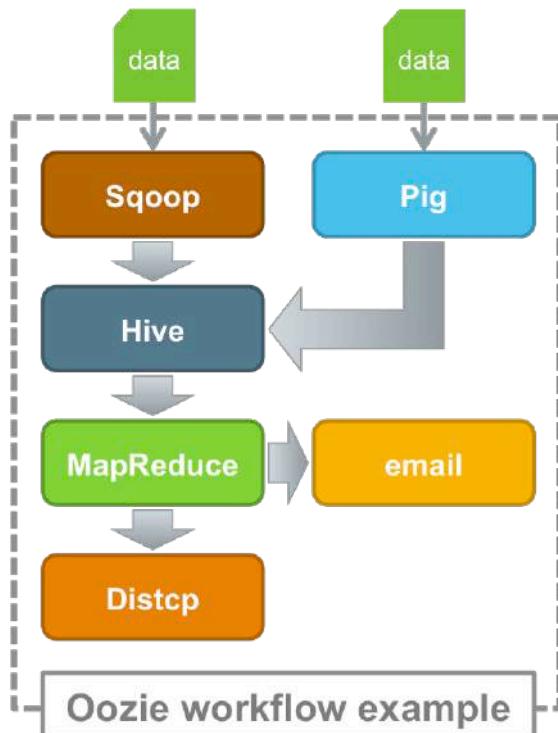
Apache Oozie High Availability

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Recall the purpose and architecture of Oozie
- ✓ Summarize the Oozie HA benefits, architecture, and operation
- ✓ Configure Oozie HA

Apache Oozie



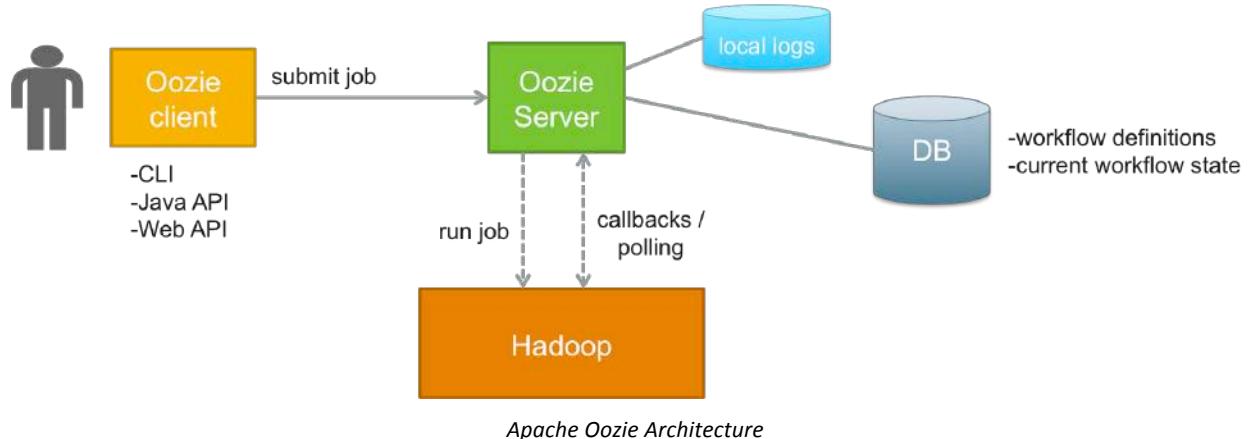
Apache Oozie

Apache Oozie is a server-based workflow engine used to execute Hadoop jobs.

Oozie enables Hadoop users to build and schedule complex data transformations by combining MapReduce, Apache Hive, Apache Pig, and Apache Sqoop jobs into a single, logical unit of work. Oozie can also perform Java, Linux shell, Distcp, SSH, email, and other operations.

It is implemented as a Java Web application running in Tomcat.

Oozie Component Architecture



To properly understand why an Oozie HA configuration is beneficial, it helps to understand the Oozie architecture components and their operation.

Oozie jobs are submitted from an Oozie client. Jobs can be submitted using a command-line interface, a Java API, or a Web API. A job is submitted to an Oozie Server.

The Oozie Server is implemented as a Java Web service running in Tomcat. The Oozie Server receives job submissions from one or more Oozie clients. The Oozie Server runs the job on the Hadoop cluster and can either actively poll or be notified by callbacks when job actions have completed or failed. The Oozie Server writes job log information to the local file system and this log information can be streamed from the Oozie Server to the Oozie Web Console. The Oozie Server also maintains a set of in-memory locks for each job to prevent multiple threads on the server from attempting to process the same job at the same time.

All Oozie state information is maintained in the Oozie database. The Oozie database is a relational database that holds workflow definitions and the current state of each job. The Oozie database may be implemented using Derby, MySQL, PostgreSQL, or Oracle.

The Oozie Server is designed to be stateless. Already running jobs continue to run even if the Oozie Server fails. New or pending jobs can be started only after the Oozie Server is brought back online.

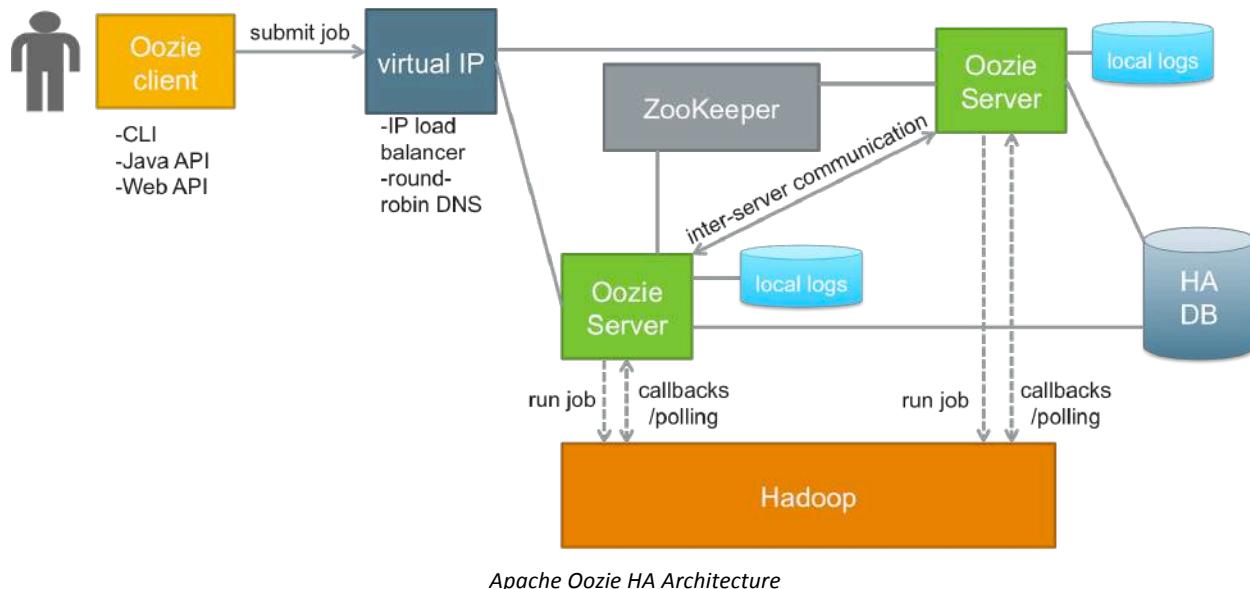
Note that in this architecture the Oozie Server and the Oozie database are single points of failure. If the Oozie Server fails then no new jobs can be submitted or run. If the Oozie database fails then the Oozie service fails and all jobs stop.

Apache Oozie HA

Oozie HA provides several benefits.

- The first benefit is that it is transparent to the end users. The steps to submit a job do not change whether Oozie HA is configured or not. The only difference is that users use a virtual IP address to connect to the Oozie service rather than use the actual IP address an Oozie Server.
- Oozie HA increases service availability through the configuration of redundant Oozie Servers. An Oozie HA environment consists of two or more Oozie Servers configured in an active-active relationship.
- The active-active server relationship enables load balancing and horizontal scalability. Multiple clients can submit jobs to multiple Oozie Servers, effectively using the resources provided by multiple Oozie Servers.
- The final benefit is that Oozie HA supports rolling upgrades. Not all Oozie Servers have to be updated at the same time, which means that some existing servers may continue to run jobs while other servers are being upgraded.

Oozie HA Component Architecture



An Oozie HA configuration is achieved through redundancy that removes the Oozie Server and the Oozie database as single points of failure. If a server fails, clients can use a surviving server. The Oozie Servers should be identical because a client cannot predetermine which server will process a job, so all servers should have the same amount of system resources.

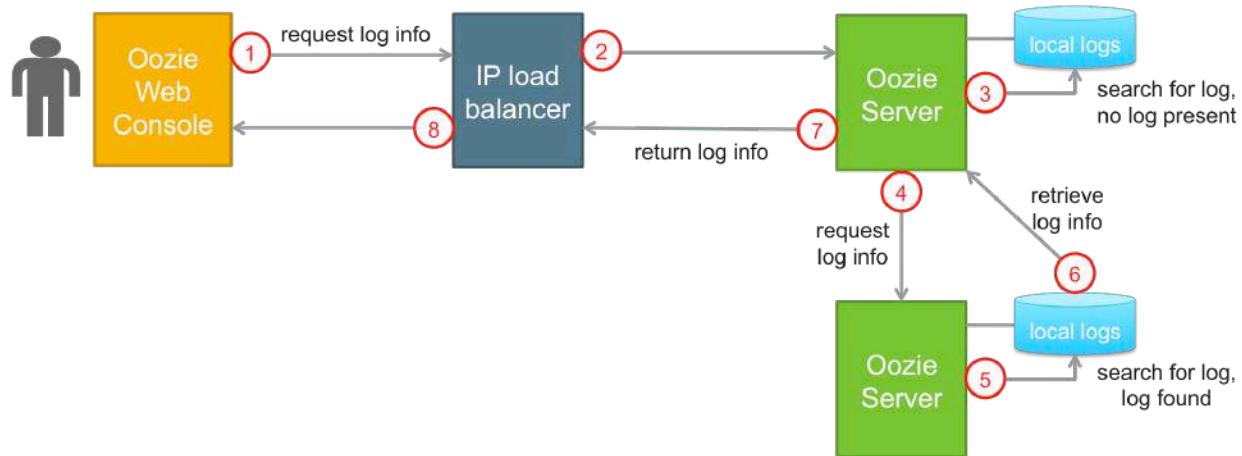
All Oozie Servers should also have concurrent access to a shared relational database. While not strictly required, the relational database that backs the Oozie Servers should also be made highly available using the best practices defined for the database system in use. For this reason, using the default embedded Derby database is not suitable for Oozie HA. The Derby database cannot be made highly available.

In Oozie HA, clients no longer submit jobs to the IP address or hostname of a specific Oozie Server but instead submit jobs to a single, virtual IP address that represents the Oozie service. This IP address can be provided by a dedicated IP load balancer or by round-robin DNS. The IP load balancer or round-robin DNS forwards the client request to a specific Oozie Server.

In a single Oozie Server configuration, in-memory locks are used to prevent different threads from trying to process the same job. In Oozie HA, these in-memory locks have been converted to distributed locks available to all of the Oozie Servers. ZooKeeper maintains these distributed locks.

The Oozie Servers also use ZooKeeper to locate one another. This is important because the individual Oozie Servers must also be able to communicate with each other to coordinate their activities.

Accessing Oozie Job Logs



Oozie Servers Communicate in Order to “Find” Job Logs

Each Oozie Server maintains its own log information about the workflow jobs and actions that it has processed. These logs are maintained on each Oozie Server's local file system rather than on the shared relational database. Oozie clients must have a way to retrieve log information, even if they are not connected to the Oozie Server that actually has the log information.

Consider the situation where there are two Oozie Servers; servers A and B. If server B processed a job then the logs for that job would be located on server B. But a request to view those logs might be sent by the IP load balancer to server A. Server A will not find the required logs when it searches its local file system. Rather than fail, Oozie HA ensures that server A sends a request to server B for the log information. Server B searches, finds, and returns the log information to server A. Server A returns the information to the requesting client. This process is illustrated in the diagram.

Configuring Oozie HA

The main steps to configure Oozie HA are:

- 1) Configure the virtual IP address on an IP load balancer or round-robin DNS
- 2) Use the Ambari Web UI to install an additional Oozie Server
- 3) Configure Oozie virtual IP address settings in `oozie-env.sh`
- 4) Configure Oozie HA-related properties in `oozie-site.xml`
- 5) Update the HDFS Oozie proxy user in `core-site.xml`
- 6) Restart all services indicated by Ambari

Configuring the Virtual IP Address

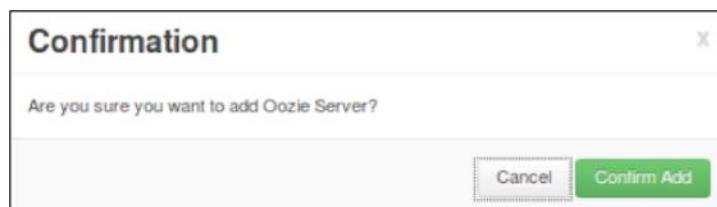
The first step is to configure the virtual IP address. The steps to configure the virtual IP address vary depending on whether an IP load balancer or round-robin DNS will be used. Use the vendor-specific instructions to complete this task.

Adding Another Oozie Server

Adding An Additional Oozie Service in Ambari Web UI

Use the Ambari Web UI to add additional Oozie Servers. To add another Oozie Server browse to the **Hosts** tab. From the list of cluster hosts, select a host for the Oozie Server and click it. Then on the **Summary** tab click the **Add** button and select **Oozie Server**.

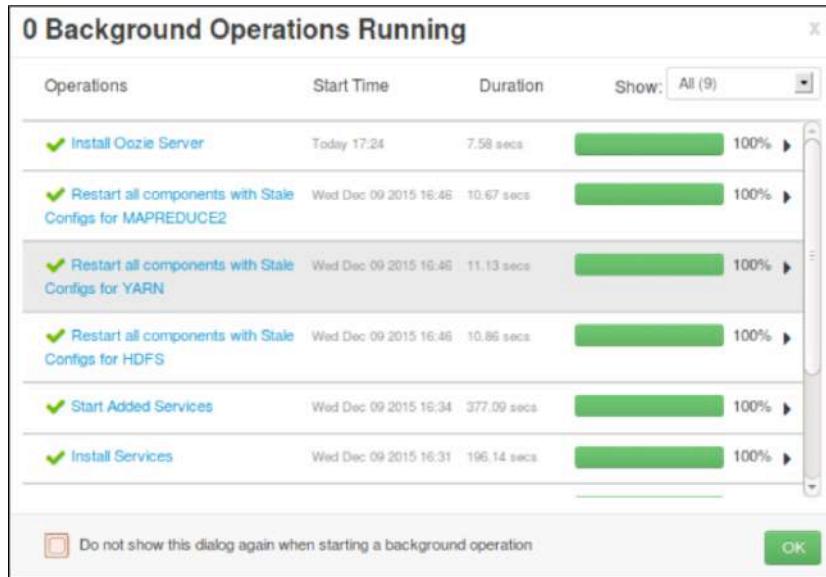
Confirm the Installation



Amabri Web UI Confirmation Window

When the confirmation window opens, click **Confirm Add** to start the installation of the Oozie Server.

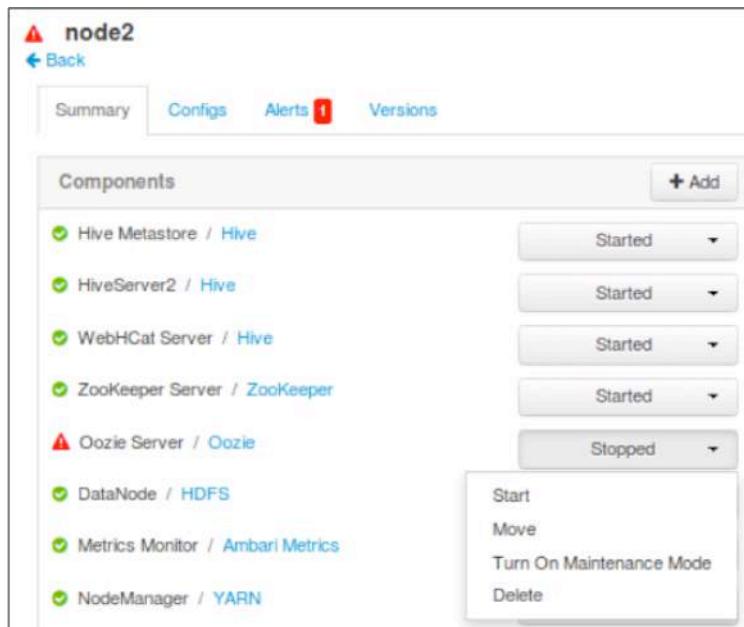
Monitoring the Oozie Server Installation



Monitoring Oozie Server Installation Progress in Ambari Web UI

Use Background Operations Running window to monitor the configuration progress. If any problems are encountered, the displayed error message is a hypertext link. Click the link to display detailed error information. When the configuration has finished, click **OK** to continue.

Starting the New Oozie Server



Starting the New Oozie Server

If the new Oozie Server has not started, then find the stopped Oozie Server in the list, click the **Stopped** menu, and select **Start**.

Verifying Redundant Oozie Servers

The screenshot shows the Ambari Web UI with the 'Services' tab selected. On the left, a sidebar lists services: HDFS, MapReduce2, YARN, Tez, Hive, Pig, **Oozie**, and ZooKeeper. The 'Oozie' service is currently selected. In the main content area, there's a 'Summary' card. Inside the card, two 'Oozie Server' entries are listed, each with a green checkmark and the status 'Started'. A red rectangular box highlights these two entries. Below the servers, the text 'Oozie Clients 3 Oozie Clients Installed' is visible.

Verifying Redundant Oozie Servers

The final step in the Ambari Web UI is to verify that there are redundant Oozie Server instances. Browse to **Services > Oozie** and check the list of configured servers.

Configuring Oozie HA-related Properties

The Oozie HA configuration requires a few manual modifications. These include:

- `oozie.base.url`
- `oozie.services.ext`
- `oozie.zookeeper.connection.string`
- `OOZIE_BASE_URL variable`

`oozie.base.url`

The screenshot shows the Ambari Web UI with the 'oozie.base.url' configuration property. The URL field contains the value 'http://virtual_IP:11000/oozie'. This entire URL field is highlighted with a blue rectangular box.

oozie.base.url

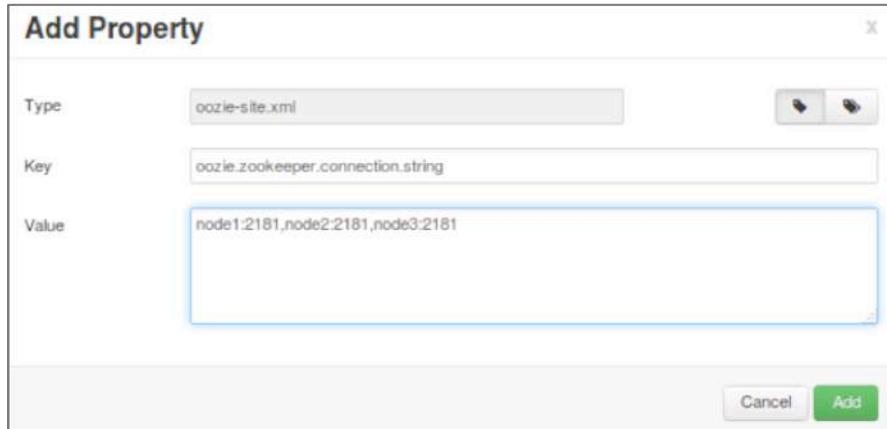
The `oozie.base.url` property must reference the virtual IP address of the Oozie service. Replace the current value that references a single Oozie Server with the IP address or hostname provided by the IP load balancer or round-robin DNS.

`oozie.services.ext`

The screenshot shows the Ambari Web UI with the 'oozie.services.ext' configuration property. The field contains the value 'ie.service.ZKJobsConcurrencyService,org.apache.oozie.service.ZKLogStreamingService'. This entire field is highlighted with a blue rectangular box.

oozie.services.net

Update the `oozie.services.ext` property value to configure Oozie to use ZooKeeper for distributed locking and distributed logging. Replace the current value with `org.apache.oozie.service.ZKLocksService,org.apache.oozie.service.ZKLogStreamingService,org.apache.oozie.service.ZKJobsConcurrencyService`.

oozie.zookeeper.connection.string*oozie.zookeeper.connection.string*

Browse to **Services > Hosts > Configs > Custom oozie-site** and click **Add Property** to add the property `oozie.zookeeper.connection.string`. The `oozie.zookeeper.connection.string` property must contain a comma-separated list of the hosts and ports in the ZooKeeper cluster.

For example, `node1:2181,node2:2181,node3:2181`.

OOZIE_BASE_URL

```
oozie-env template
# The port Oozie server runs
#
export OOZIE_HTTP_PORT={{oozie_server_port}}

# The admin port Oozie server runs
#
export OOZIE_ADMIN_PORT={{oozie_server_admin_port}}

# The host name Oozie server runs on
#
# export OOZIE_HTTP_HOSTNAME=`hostname -f`

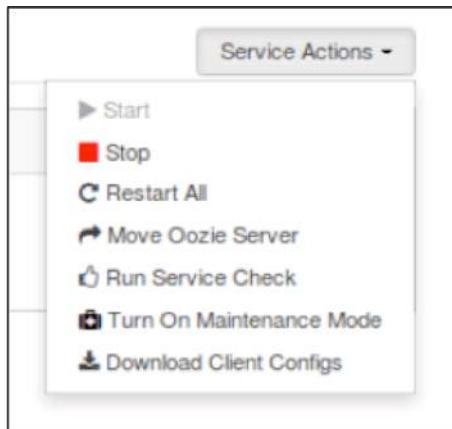
# The base URL for callback URLs to Oozie
#
export OOZIE_BASE_URL="http://virtual_IP:11000/oozie"
export JAVA_LIBRARY_PATH={{hadoop_lib_home}}/native/Linux-amd64-64

# At least 1 minute of retry time to account for server downtime during
# upgrade/downgrade
export OOZIE_CLIENT_OPTS="${OOZIE_CLIENT_OPTS} -Doozie.connection.retry.count=5"
```

OOZIE_BASE_URL Variable

Browse to **Services > Oozie > Configs > Advanced oozie-env** to make the final Oozie configuration change. Uncomment and update the environment variable `OOZIE_BASE_URL`. It must reference the virtual IP address or hostname provided by the IP load balancer or round-robin DNS.

Restart the Oozie Service



Restarting the Oozie Service

Complete the configuration by restarting the Oozie service. In the Ambari Web UI, browse to **Services > Oozie**, then click **Service Actions** and select **Restart All**.

Updating HDFS Oozie Proxy User



Updating HDFS Oozie Proxy User

Update the HDFS `core-site.xml` configuration by browsing to **Services > HDFS > Configs > Advanced > Custom core-site**. Modify the `hadoop.proxyuser.oozie.hosts` property to include a comma-separated list of the Oozie Servers.

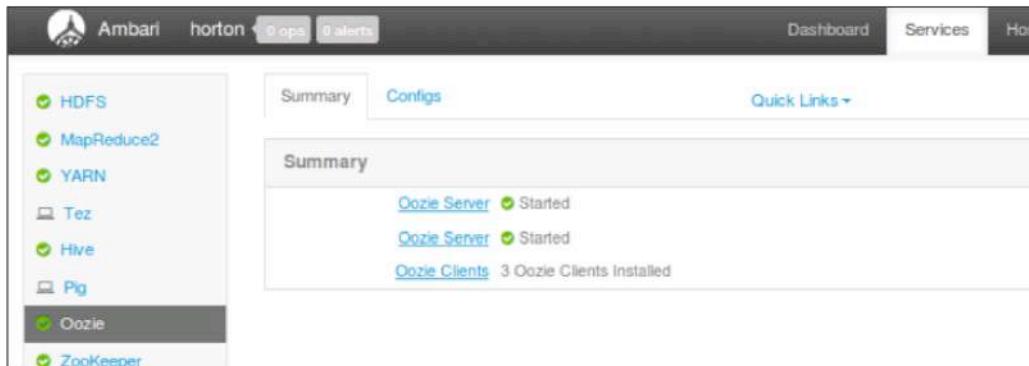
Restarting Services Indicated by Ambari



Restarting Services in Ambari Web UI

The final step is to restart any services as indicated in the Ambari Web UI.

Listing Oozie Servers



The screenshot shows the Ambari Web UI interface. At the top, there's a navigation bar with the Ambari logo, the cluster name 'horton', and status indicators for '0 ops' and '0 alerts'. Below the navigation bar, there are tabs for 'Dashboard', 'Services', and 'Hosts'. The 'Services' tab is selected. On the left, a sidebar lists various services: HDFS, MapReduce2, YARN, Tez, Hive, Pig, Oozie (which is highlighted with a dark grey background), and ZooKeeper. The main content area has tabs for 'Summary' and 'Configs', with 'Summary' being active. Under 'Summary', it says 'Oozie Server' is 'Started'. There are also links for 'Oozie Client' and '3 Oozie Clients Installed'. A 'Quick Links' dropdown menu is visible at the top right.

Listing Current Oozie Servers in Ambari Web UI

Both the command line and the Ambari Web UI can be used to list the current Oozie Servers in an Oozie HA configuration.

From the command line type `oozie admin -oozie http://<virtual_IP>:11000/oozie -servers` to list the current Oozie Servers.

From the Ambari Web UI browse to **Services > Oozie** and click each **Oozie Server** to determine its host.

Knowledge Check

Use the following questions and answers to assess your understanding of the concepts presented in this lesson.

Questions

- 1) The Oozie Server and the Oozie _____ are the two primary points of failure in a non-Oozie HA configuration.
- 2) True or false? Oozie HA is implemented by an active-passive server configuration.
- 3) True or false? Oozie HA must have a highly available database configuration.
- 4) True or false? Oozie HA must have a highly available ZooKeeper configuration.
- 5) Oozie HA Servers are accessed through a _____ IP address.
- 6) Oozie HA _____ locking is implemented using ZooKeeper.

Answers

- 1) The Oozie Server and the Oozie _____ are the two primary points of failure in a non-Oozie HA configuration.

Answer: Database

- 2) True or false? Oozie HA is implemented by an active-passive server configuration.

Answer: False. It is an active-active configuration.

- 3) True or false? Oozie HA must have a highly available database configuration.

Answer: False. It is recommended but not strictly required.

- 4) True or false? Oozie HA must have a highly available ZooKeeper configuration.

Answer: False. It is recommended but not strictly required.

- 5) Oozie HA Servers are accessed through a _____ IP address.

Answer: virtual

- 6) Oozie HA _____ locking is implemented using ZooKeeper.

Answer: distributed

Summary

- Apache Oozie is a server-based workflow engine used to execute Hadoop jobs.
- Oozie HA provides several benefits:
 - It is transparent to users
 - It increases availability through an active-active Oozie Server configuration
 - It enables load balancing and horizontal scalability
 - It supports rolling upgrade
- An Oozie HA configuration is achieved through redundancy that removes the Oozie Server and the Oozie database as single points of failure.
- Oozie HA requires HDP 2.2 or later, ZooKeeper, a virtual IP address, and a highly available relational database that supports multiple, concurrent connections.

Introduction to Falcon

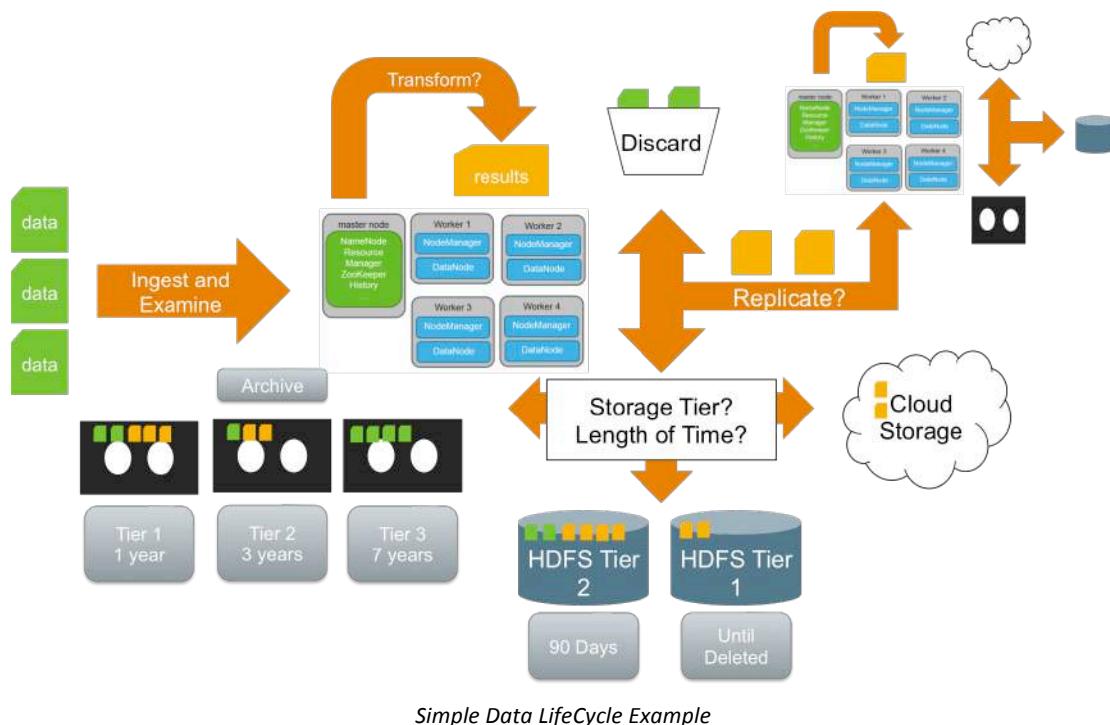
Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Describe the challenges of data governance in large, complex environments
- ✓ Recall the purpose and capabilities of Falcon
- ✓ Deploy Falcon
- ✓ Describe the purpose and configuration of Cluster, Feed and Process Entities
- ✓ Create a Cluster Entity and set up mirroring using the Falcon UI

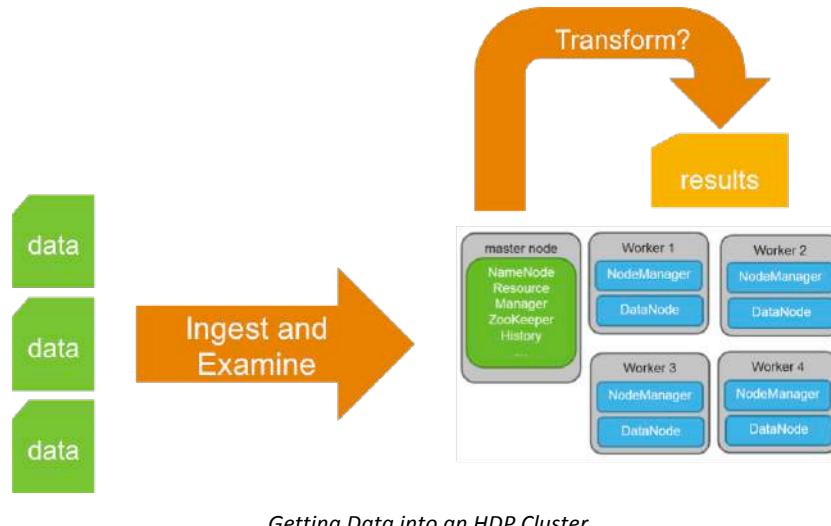
Data Governance Challenges

Data lifecycle management is a complex issue. Let's take a simple example in order to make this point.



Getting Data into the Cluster

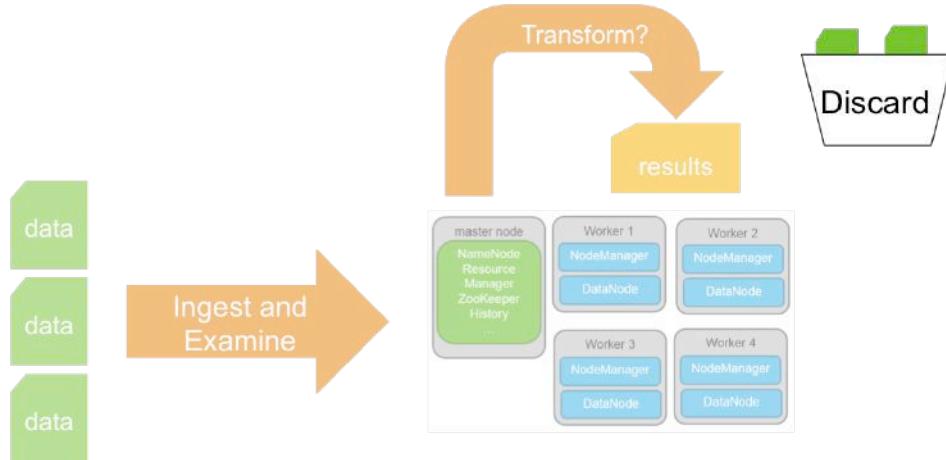
We start with some raw data and an HDP cluster. The first step in managing the data is to get it into the HDP cluster. We must have some mechanism to ingest that data – perhaps Sqoop, Flume, or Storm – and then another mechanism to analyze and decide what to do with it next. Does this data require some kind of transformation in order to be used? If so, ETL processes must be run, and those results generated into another file. Quite often, this is not a single step, but multiple steps configured into a data application pipeline.



Getting Data into an HDP Cluster

Keeping or Discarding Data

The next decision comes with regards to whether to keep or discard the data. Not all data must be kept, either because it has no value (empty files, for example), or it is not necessary to keep once it has been processed and transformed. Thus some raw data can simply be deleted.



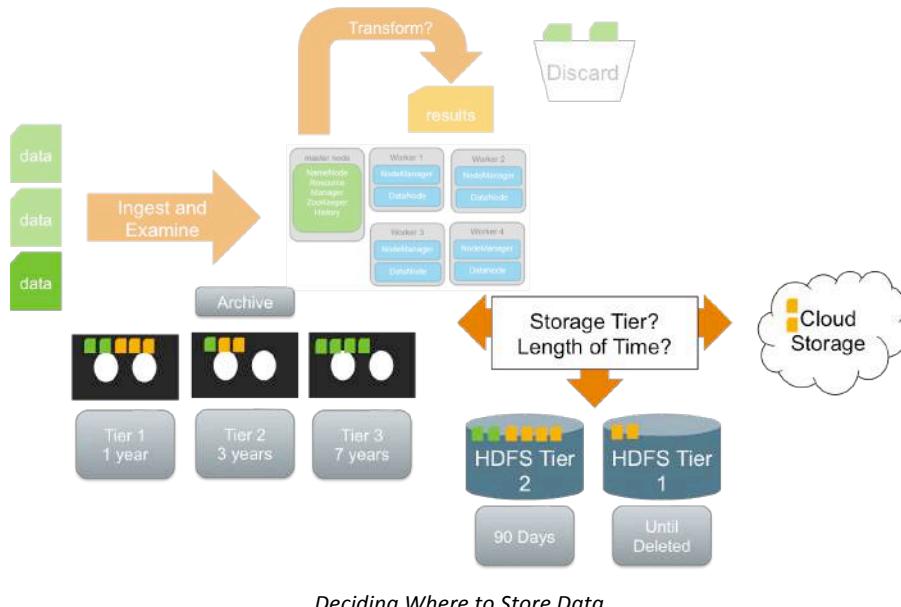
Deciding which Data can be Discarded

Storage Locations

Data that must be kept requires additional decisions to be made. For example, where will the data be stored, and for how long? Your Hadoop cluster might have multiple tiers of HDFS storage available, perhaps separated via some kind of node label mechanism. In the example, we have two HDFS storage tiers. Any data that is copied to tier 2 should be stored for 90 days. We have another, higher tier of HDFS Storage, and any data stored here should be kept until it is manually deleted.

You may decide that some data should be archived rather than made immediately available via HDFS, and you can have multiple tiers of archives as well. In our example we have three tiers of archival storage, and data is kept for one, three, and seven years depending on where it is stored.

A third location where data might end up is on some kind of cloud storage, such as AWS or Microsoft Azure.

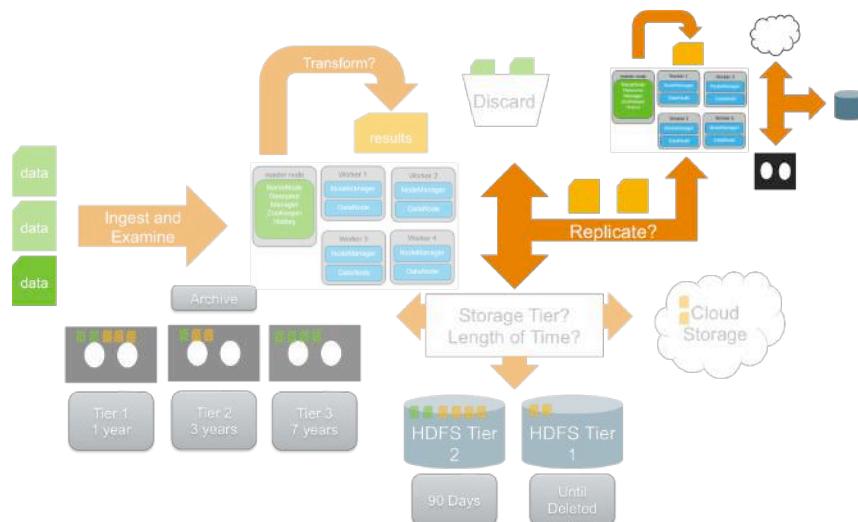


Deciding Where to Store Data

Both raw data and transformed data might be kept anywhere in this storage infrastructure as result of having been input and processed by this HDP cluster.

Data Decisions in a Multi-Cluster Environment

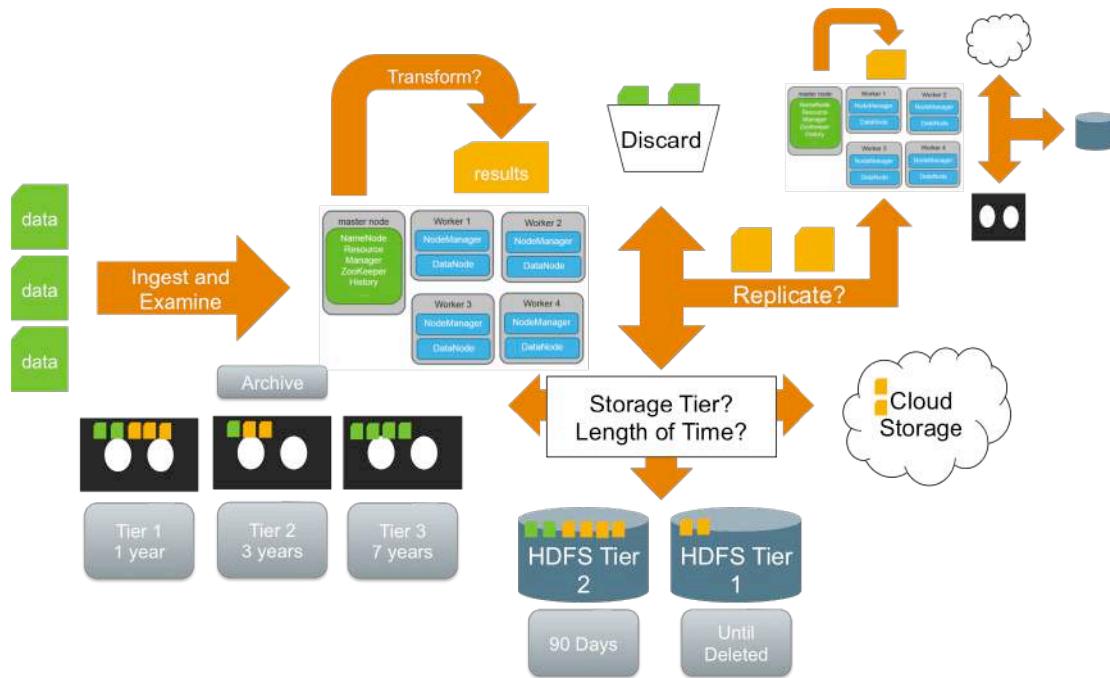
In addition, you may be working in a multi-cluster environment, in which case an additional decision is required. What data needs to be replicated between the clusters? If files need to be replicated to another HDP cluster, then once that cluster ingests and examines that data, this same kind of processes and decision mechanisms need to be employed. Perhaps additional transformation is required. Perhaps some files can be examined and deleted. For files that are to be kept, their location and length of time to retain must be decided, just as on the first cluster.



Decisions about Data in a Multi-Cluster Environment

Data Decisions in a Fully Utilized HDP Environment

This is a relatively simple example of the kind of data lifecycle decisions that need to be made in an environment where the capabilities of HDP are being fully utilized. This can get significantly more complex with the addition of additional storage tiers, retention requirements, and geographically dispersed HDP clusters which must replicate data between each other, and perhaps the central global cluster designed to do all final processing.



Data Lifecycle in a Simple, Fully Utilized HDP Environment

Data Lifecycle Considerations

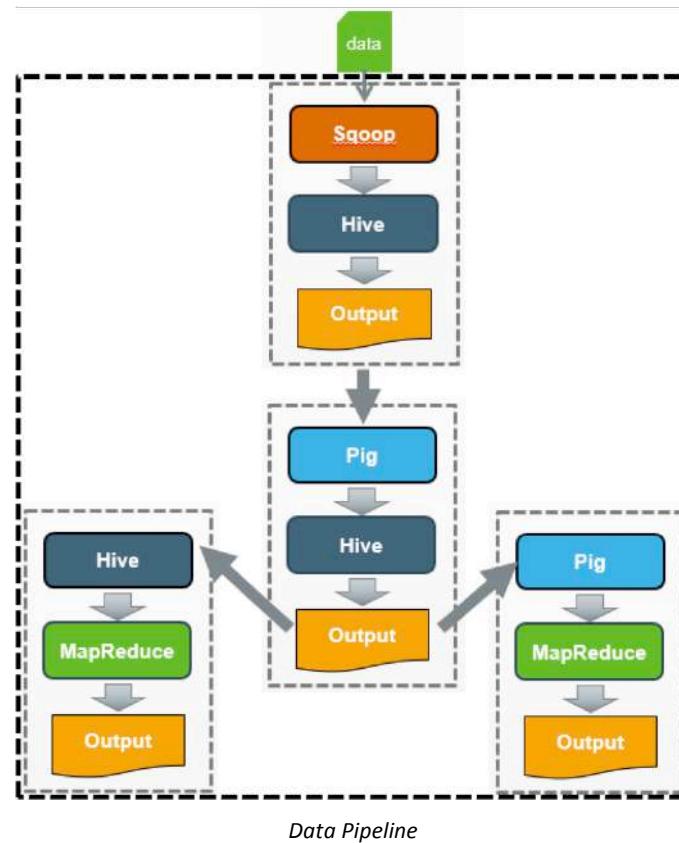
When thinking about data lifecycle management, there are a significant number of considerations that need to be made. The mechanism to acquire data must be determined – whether it is sourced in a database, log files, streaming data, or some customized feed. In addition, the mechanisms that will be used to export that data must be determined – whether it be email, file share, distcp, or other - and to what locations.

Sometimes it makes sense to have multiple clusters, into replicate data between them. For example, data may be generated across multiple geographically dispersed sites. It may make sense to ingest and initially process this data in a cluster that is located close to the source, and then only send the processed data files to a central global cluster in order to save time and bandwidth. This is especially true when the raw data files are extremely large and would be time consuming and expensive to ship as-is to a centralized cluster. In addition, BC/DR considerations may inform replication decisions. It may make sense to replicate mission-critical files across multiple geographically dispersed HDP clusters in order to quickly recover from failure at a given site.

Data and bandwidth-tiering decisions also need to be made. Replication may be more critical for some data, and thus the ability to assign more bandwidth to the critical data can be useful, in addition to determining how frequently data is replicated. In addition, for some data the raw files may need to be replicated across clusters, whereas for other data, aggregate summaries may suffice.

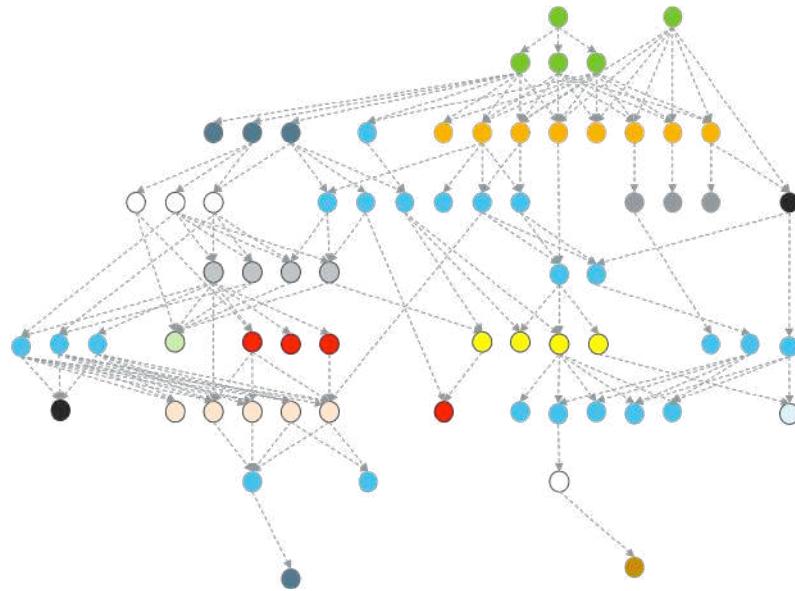
A number of these decisions may be driven by service level agreements that exist. For example, you may need to store data on special hardware in order to meet performance requirements for some users, whereas for other, more cost-sensitive users, a lower level of performance is acceptable. In addition, proper ACLs need to be set to ensure that the wronged user is not able to move or delete data without authorization.

Complex Data Pipelines



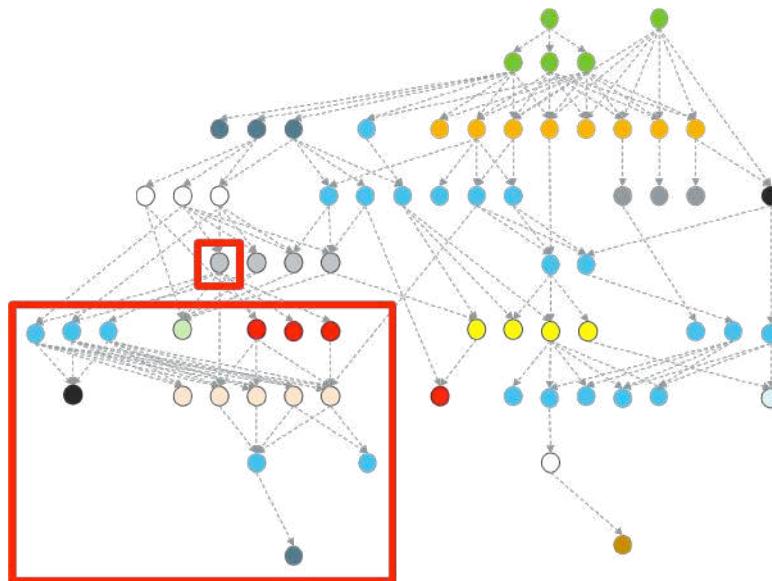
Data application pipelines, often simply referred to as data pipelines, are a collection of coordinated workflows in which the output of one workflow serves as the input for one or more additional workflows. Groups of these workflows can be managed as a single unit by using Oozie bundles. However, what if instead of just a few coordinated workflows, you are managing a data application pipeline of hundreds, thousands, or tens of thousands of them?

Pipeline Management



Data Pipeline of 68 Individual Workflows

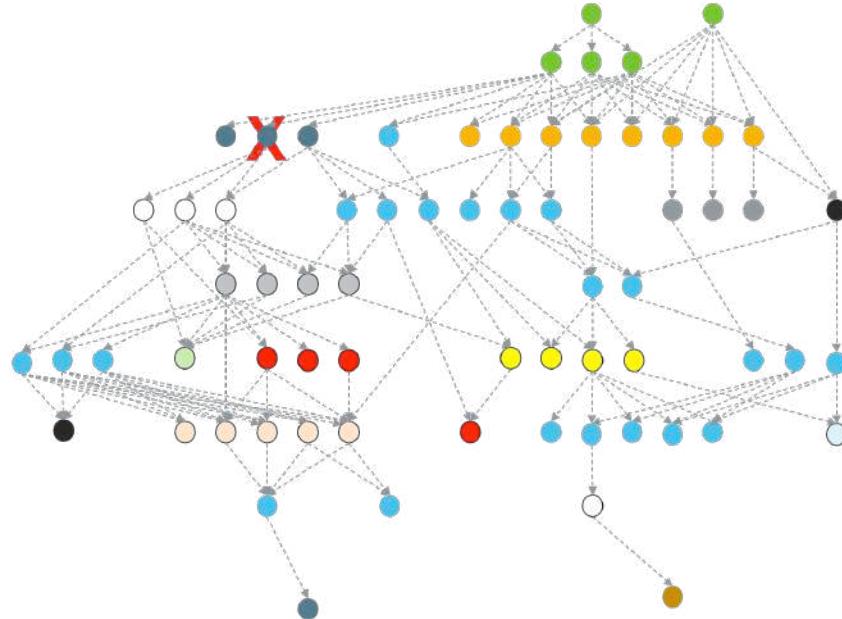
The diagram above represents just 68 individual workflows., each as a separate dot. Each dot's color represents a different development group that is responsible for that workflow, with the dotted arrows representing the data pipeline dependencies.



A Change Upstream May Require Changes in any Dependent Downstream Coordinator

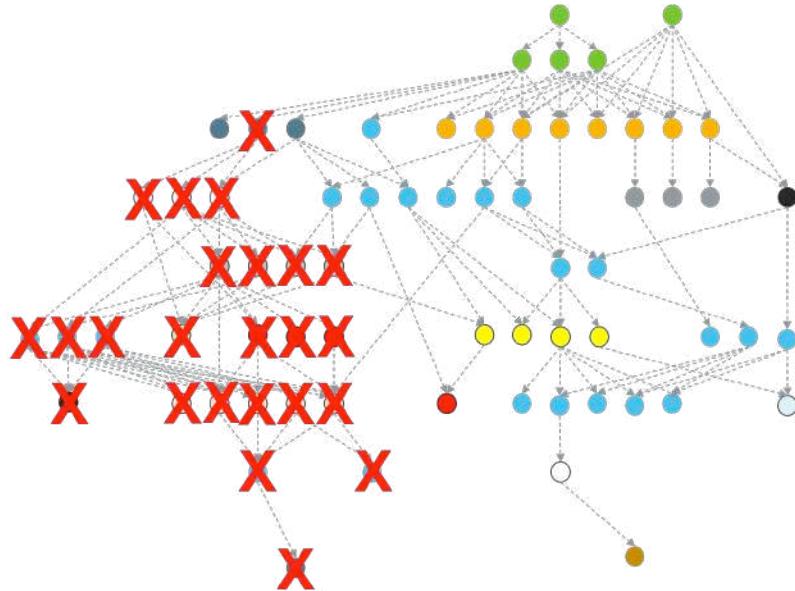
Oozie coordinators defines by time / frequency, etc. and outputs from one may be used by multiple others. If something changes upstream, then it is necessary to remember to adjust *every* dependent coordinator that may be affected by that change. Overall view of pipelines, retention, and so forth, exists as independent workflows / coordinators that may span multiple systems. Even with bundling, this becomes highly prone to error when number of workflows moves into the hundreds, thousands, or even tens-of-thousands. If all information is available and tracked in a single place, it becomes easier to manage these pipelines.

Late Data and Reprocessing



When a Process Runs Later than Expected

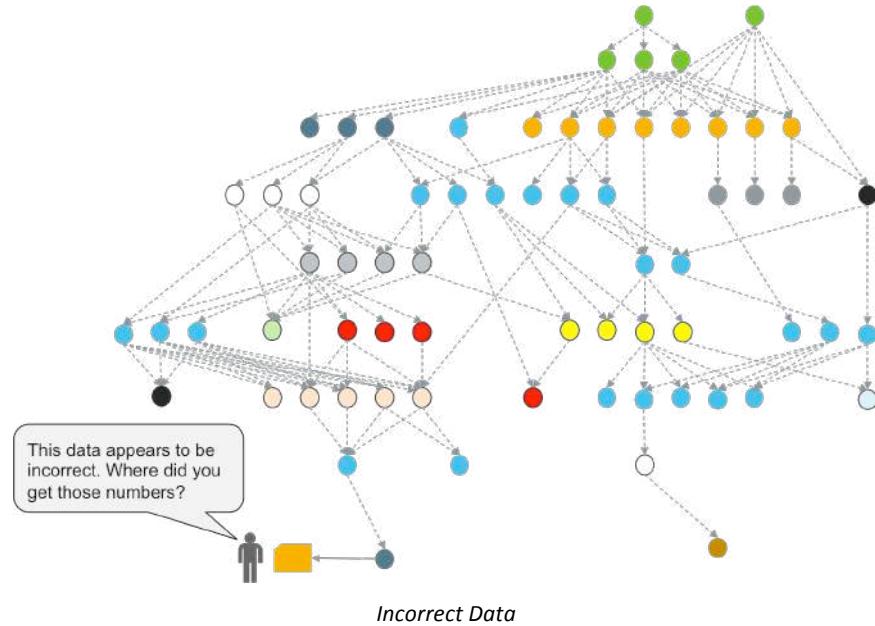
What happens in a complex system when the scheduling mechanism fails?



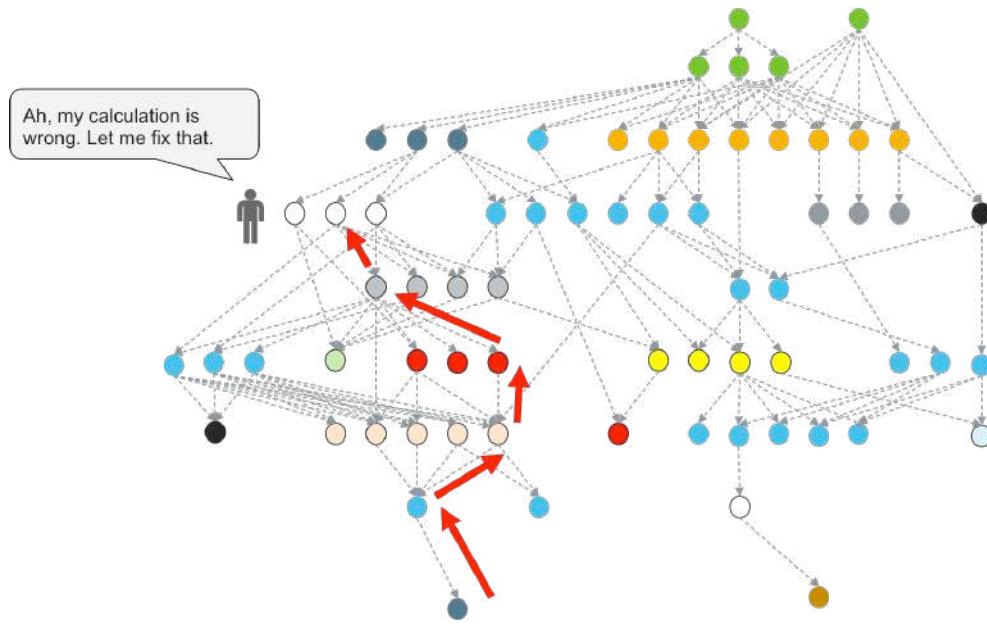
You May Need to Re-run All Dependent Workflows

For example, what if an upstream process runs late for some reason – for example, the data arrives later than expected? You need to have a way to recognize this situation, and then re-run workflow and all dependent workflow once the data becomes available.

Lineage



Lineage refers to the need to keep track of how data was transformed, and perhaps figure out where and error is being generated via audit logs at each step of the data pipeline.



Tracing Incorrect Data Back to the Source Error

This can allow for the correction of whatever error was made, after which the data pipeline can be re-executed.

Metadata Governance

Another level of data governance centers on the ability to tag elements within data pipelines and then use those tags as manageable objects. This enables data classification, centralized auditing, the collection of search and lineage history, and the ability to finely tune security and policy engines across the cluster.

Note

While Falcon does provide some metadata tagging capabilities, Apache Atlas is the tool that is designed to manage metadata at the cluster level. Metadata governance will not be discussed in depth in this lesson.

Falcon Overview

Apache Falcon is a framework which meets data governance needs in three specific areas: centralized data lifecycle management, compliance and audit, and data replication in archival.

Data Lifecycle Management

Falcon leverages Oozie to schedule and coordinate data processing workflows. Falcon also leverages Oozie capabilities to transform and analyze data. For example, Oozie can run MapReduce, Hive, Pig, Sqoop, Java, Linux shell, distcp, SSH, email, along with a number of other programs. Falcon simplifies management by providing a framework to define, deploy, and manage complex data pipelines that would be difficult to manage using Oozie alone.

Falcon enables a user to define how, when, and where to ingest data into a cluster. Falcon's built-in data ingestion capabilities include the complex logic of how to determine and handle late data arrival. It also includes logic that enables a user to define how often, and how many times, data ingestion should be retried following a failure. Falcon also enables users to define different data retention policies for different types of data. When the retention policies are met, the data is automatically deleted from a cluster. This automation reduces user error, saves administration time, and reduces storage requirements.

Falcon can be used to ensure disaster readiness in business continuity be a quick and easy setup of complex data replication policies via a relatively easy to use web interface. It also provides significant capabilities around end-to-end monitoring of data pipelines.

Pipeline Development and Management



Data Pipeline Requirements

Data pipelines require in some cases raw data, transformed data, the code required in order to implement actions as part of a workflow, and of course one or more HDP clusters on which all of this data is stored and processed.



Falcon Framework Supports Data Pipeline Development and Management

Falcon provides a framework in which many of the tasks involved encoding complex data pipelines are implemented as generic operations. Examples of genericized operations include: replication, data acquisition, data relay, lifecycle management, late data management, metadata tagging, lineage tracing, and pipeline audit.

The Falcon framework allows individual workflows to be chained together, managed by these default options, which means that workflows can act as part of a data pipeline platform rather than requiring developers to re-code much of their workflow logic with each new project. Another way to think of this is that it allows for the creation of data pipelines in which nearly everything about them is controlled via a standardized template. This allows developers to focus on business logic within the data manipulation workflow rather than low-level logistics.

Falcon Architecture

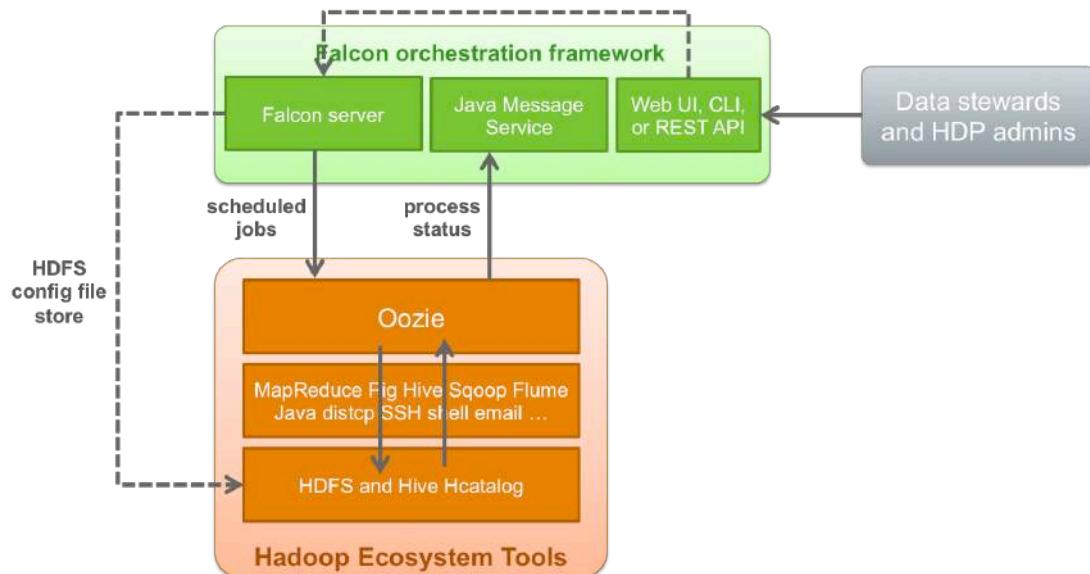


Falcon Architecture

Falcon is comprised of the Falcon orchestration framework and several components of Hadoop that interact with this framework.

The Falcon orchestration framework is comprised of three major components: the Falcon server, a Java message service, and a web UI, CLI, or REST API that is used by data stewards and HDP admins use to interact with it. The primary Hadoop tool leveraged by Falcon is Apache Oozie, which further leverages a number of additional tools such as Pig, Hive, Sqoop, Flume, MapReduce, Java programs, distcp, and so on. Falcon also interacts with HDFS and the Hive HCatalog for data storage and replication purposes.

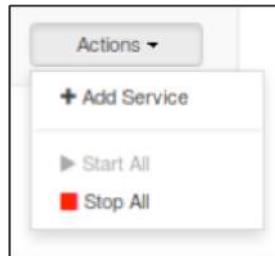
Typical Instance



A Typical Falcon Instance

In a typical instance, a Falcon user will log into Falcon via the Falcon Web UI or CLI, which then connects them to the Falcon server. The user will create configuration files to manage Falcon data pipelines, which are typically stored on HDFS, and then schedule those jobs to run. Falcon communicates this information to Oozie, and Oozie then coordinates and runs whatever underlying workflows are required to perform the required steps. Oozie tracks progress of these operations, and then reports success or failure back to Falcon via the Java message service. All of this information is made available via the Falcon Web UI, and can also be accessed via CLI commands.

Deploying Falcon



Ambari > Services > Actions > Add Service

To install Falcon, log in to the Ambari Web UI as a user with at least Operator permissions. Click the Services tab to reveal the service Actions menu. Click the Actions menu and select Add Service. The Add Service wizard opens.

Choosing Services

Service	Version	Description
HDFS	2.7.1.2.3	Apache Hadoop Distributed File System
Falcon	0.6.1.2.3	Data management and processing platform
Storm	0.10.0	Apache Hadoop Stream processing framework

Ambari – Choose Services

In the **Choose Services** window, select the Falcon service. Scroll down and click **Next** (not shown) when ready to continue.

Assigning Masters

Assign Masters

Assign master components to hosts you want to run them on.

Falcon Server: node1 (14.4 GB, 4 cores)

Ambari – Assign Masters

Ambari provides an initial layout of service master components. Use the **Assign Masters** window either to confirm or move the master components. This is when information about workloads and resource capacity gained from pre-planning, or pre-testing with a pilot cluster, is very helpful. The resource capacity of each host must be considered along with service redundancy.

Use the drop-down menu next to each component to move the master component to another cluster node, as necessary. Click **Next** (not shown) to proceed.

Assigning Slaves and Clients

Assign Slaves and Clients

Assign slave and client components to hosts you want to run them on.
Hosts that are assigned master components are shown with *.
"Client" will install Falcon Client

Host	all none	all none	all none	all none
node1*	<input type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node2*	<input type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
node3*	<input type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client

Show: 25 1 - 3 of 3 < > Next →

← Back

Ambari – Assign Slaves and Clients

Use the **Assign Slaves and Clients** window to choose where to run service worker components and Hadoop client software.

- The DataNode is the HDFS service worker component and is already installed on all nodes in the screen capture.
- An HDFS NFS Gateway machine can be used as a method to access HDFS. The HDFS NFS Gateway is described in another lesson.

- The NodeManager is the YARN service worker component and is already installed on all nodes in the screen capture.
- Client represents the Hadoop client software. Client software is used by user and application clients to access cluster services and resources. For example, client software is used to access HDFS storage, run YARN jobs, or in this specific case, submit Falcon jobs. Client software is commonly installed on cluster nodes. Client software is also commonly installed on utility machines used as gateway machines to access cluster resources.

Select all nodes from which Falcon jobs will be submitted and click **Next**.

Customizing Services

Customize Services

We have come up with recommended configurations for the services you selected. Customize them as you see fit.

Hadoop services must typically be customized for each specific cluster installation. Ambari will do some limited customization based on the choices made during installation. For example, configuration properties that include the hostname of the Falcon Server will be updated by Ambari based on your choice of a Falcon Server host in the earlier **Assign Masters** window.

In addition to the customizations made by Ambari, the **Customize Services** window enables a user to customize a myriad of configuration settings.

Depending on the services selected in the earlier **Choose Services** window, Ambari might display red alert icons. In the example here, Ambari has no alerts for the user which means that Ambari can install Falcon with the current configuration settings.

Click **Next** when ready to proceed.

Reviewing and Deploying

Review

Please review the configuration before installation.

Admin Name : admin
Cluster Name : horton
Total Hosts : 3 (0 new)

Repositories:

- redhat6 (HDP-2.3):
http://node1/hdp/HDP-2.3
- redhat6 (HDP-UTILS-1.1.0.20):
http://node1/hdp/HDP-UTILS-1.1.0.20
- redhat7 (HDP-2.3):
http://public-repo-1.hortonworks.com/HDP/centos7/2.x/updates/2.3.2.0
- redhat7 (HDP-UTILS-1.1.0.20):
http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/centos7
- suse11 (HDP-2.3):
http://public-repo-1.hortonworks.com/HDP/suse11sp3/2.x/updates/2.3.2.0
- suse11 (HDP-UTILS-1.1.0.20):
http://public-repo-1.hortonworks.com/HDP-UTILS-1.1.0.20/repos/suse11sp3

Services:

Falcon

[← Back](#) [Print](#) [Deploy →](#)

Ambari – Review

The **Review** window enables you to print the configuration and deploy the new software.

Click **Deploy** when ready to install Falcon.

Installing, Starting, and Testing Falcon

Install, Start and Test

Please wait while the selected services are installed and started.

100 % overall

			Show: All (3) In Progress (0) Warning (0) Success (3) Fail (0)
Host	Status	Message	
node1	<div style="width: 100%;">100%</div>	Success	
node2	<div style="width: 100%;">100%</div>	Success	
node3	<div style="width: 100%;">100%</div>	Success	

3 of 3 hosts showing - [Show All](#) Show: 25 1 - 3 of 3 ⌂ ⌃ ⌂ ⌃ ⌂ ⌃

Successfully installed and started the services.

[Next →](#)

Ambari Install, Start, and Test

The **Install, Start, and Test** window displays the progress on each cluster node. If any node fails to install properly, the error displayed in the Message column is a hypertext link. Click the link to get more detailed error information. If you can resolve the issue, you can use Ambari to attempt to install the node again.

Click **Next** when ready to proceed.

Completing the Installation

Summary

Important: You may also need to restart other services for the newly added services to function properly (for example, HDFS and YARN/MapReduce need to be restarted after adding Oozie). After closing this wizard, please restart all services that have the restart indicator  next to the service name.

Here is the summary of the install process.

The cluster consists of 3 hosts
Installed and started services successfully on 3 new hosts
Install and start completed in 34 seconds

Complete 

Ambari – Summary

Read the information in the Summary window and then click **Complete**.



The screenshot shows the Ambari Summary page. On the left, there's a sidebar with service icons: HDFS (green checkmark), MapReduce2 (green checkmark), YARN (green checkmark), and Tez (grey square). The main area has tabs: Summary (selected), Heatmaps, Configs, Quick Links, and Service Actions. A yellow banner at the bottom states: "Restart Required: 8 Components on 3 Hosts". A green "Restart" button is visible on the right.

Ambari – Restart Services

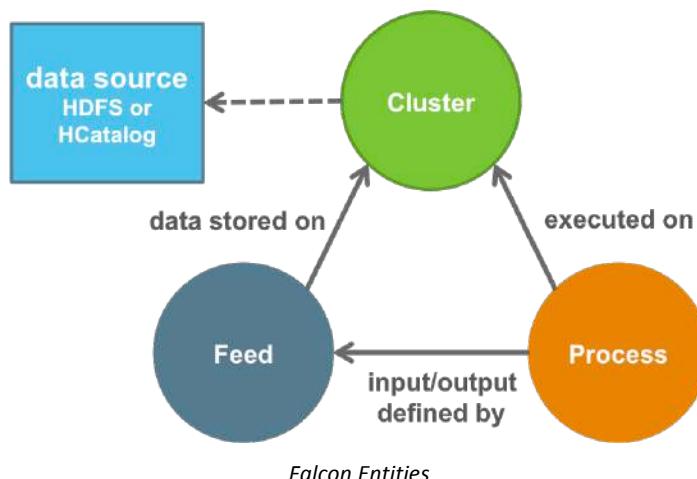
Then restart any services indicated in the Ambari Web UI.

Verifying the Service

Ambari – Services > Falcon

The final step in the Ambari Web UI is to verify that there is a running Falcon Server. Browse to **Services > Falcon** and check the list of configured servers and clients.

Falcon Entities and Mirroring



Falcon uses higher-level abstractions called **Cluster**, **Feed**, and **Process** entities to define and orchestrate workflows. Entities are created using user-configured XML files and specify such things as data management policies, clusters and cluster services, data inputs and outputs, data processing programs, HDFS resources, and retry behavior.

- The **Cluster** entity defines the infrastructure for an HDP cluster. It defines the location of various components and directories required to successfully run workflows and pipelines.
- The **Feed** entity defines the location of data on the cluster, be it on HDFS or HCatalog. It also defines other characteristics of that data, such as frequency, time-based policy specifications, on which clusters the data is available, and retention policies.

- The **Process** entity defines configuration and processing logic for components in a data pipeline. This includes: which Feed entities should be used as input data, which output feeds results should be sent to, the frequency with which an operation should be executed, which clusters to run these processes on, and how – and how often – to retry processing after a failure.

The diagram further illustrates configuration dependencies between the Falcon entities.

- Before any other entities can be fully defined, one or more Cluster entities must be configured and available.
- A Feed entity must know where its data is stored on the cluster whether it be an HDFS or Hcatalog source.
- Process entities rely on both Feed and the Cluster entities. They must know which data feeds to use when running the workflows, as well as the clusters on which these processes should be executed.

These entities are defined by user-configured XML files, which can be generated by using the Falcon Web UI.

Pipeline Abstraction



The concept of entities allows for the abstraction of nearly every component in a data pipeline.

Specifying Clusters as Abstract Cluster Entities



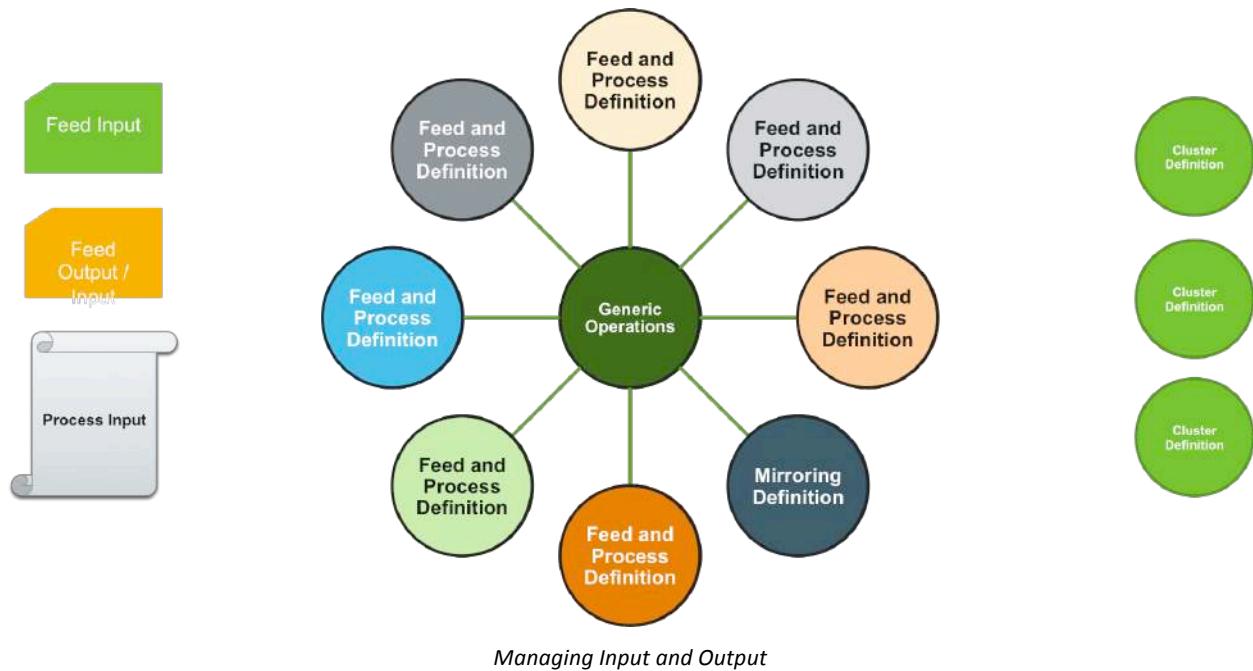
HDP clusters are defined and can be specified as abstract Cluster entities.

Defining and Managing Generic Business Logic with Entity Definitions



Nearly all of the generic business logic that surrounds data pipelines can be easily defined and managed in various Feed and Process entity definitions.

Managing Input and Output



Managing Input and Output

Both raw and transformed data become inputs and outputs of various data feeds and processes, and if workflows are chunked appropriately and defined generically enough, even they can be reused multiple times across various data pipelines.

Data Mirroring



Data Mirroring

It is worth noting here that there is a special case when it comes to data replication. The Falcon Web UI provides a separate interface for configuring data mirroring. Technically data mirroring also requires feed and process definitions, however these are generic enough that they have been built into the interface and can be configured in a single step assuming the Cluster entities or cloud storage locations have already been defined.

Falcon Entity Benefits

The Cluster, Feed, and Process entities enable the separation of business logic from application logic. Falcon enables easy data management through a declarative mechanism. Users use the entities to define infrastructure endpoints, datasets, and processing rules declaratively and Falcon automatically leverages existing Hadoop components to build, schedule, and monitor data processing pipelines.

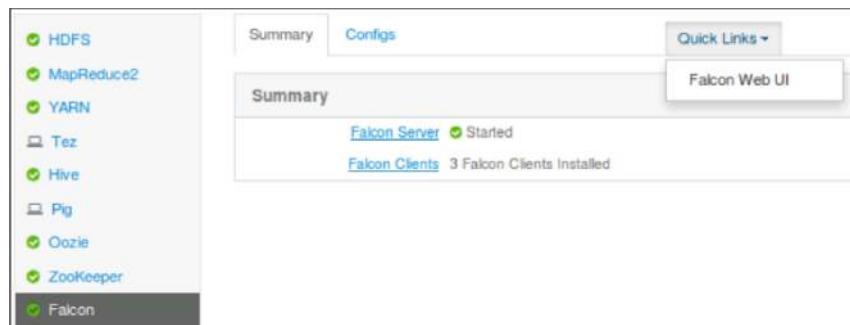
Actions defined in entities can be sent parameterized values so you can use them to work on many different Falcon-defined processes. This helps to reduce code complexity and makes it easier to manage multiple workflows consistently across one or more clusters.

While you can build complicated Oozie workflows using a single set of Falcon entities, Hortonworks recommends that you split up complex Oozie workflows into modular steps. This helps Falcon to better manage retention of any intermediary data sets. It also allows for greater reuse of both Process and Feed entities.

A user creates entity specifications and submits them to Falcon using the Falcon Web UI. Falcon transparently transforms the entity specifications into scheduled Oozie jobs. Oozie returns job status to Falcon using the Java Message Service.

Creating Entities

We will create entities using the Falcon Web UI.



Ambari > Services > Falcon

It can be accessed via the Ambari Services > Falcon page by selecting Quick Links, and then Falcon Web UI from the resulting drop-down menu.



Falcon UI Login

Introduction to Falcon

You will be presented with a log-in screen, where you would supply the username under which you want to run the data pipelines.



Falcon UI Interface

Once checked in, you are presented with an interface that contains a search feature as well as buttons that allow you to create the various entities.

The Falcon Web UI generates entity XML files based on input provided by the user, thus greatly simplifying the creation of entities and reducing the potential for human error during configuration.

Cluster Entity

New Cluster

General

Name:

Color: Description:

Tags: value:
+ add tag

Access Control List

Owner	Group	Permissions
ambari-qa	users	0x755

Interfaces

Type	Endpoint	Version
readonly	http://sandbox.hortonworks.com:50070	2.2.0
write	hdfs://sandbox.hortonworks.com:8020	2.2.0
execute	sandbox.hortonworks.com:8050	2.2.0
workflow	http://sandbox.hortonworks.com:11000/oozie/	4.0.0
messaging	tcp://sandbox.hortonworks.com:61616?daemon=true	5.1.6
registry	<input type="text"/>	<input type="text"/>

Properties

name	value
<input type="text"/>	<input type="text"/>

+ add property

Location

name	path
staging	<input type="text"/>
temp	<input type="text"/>
working	<input type="text"/>

+ add location

Cancel Next

XML Preview

```
<cluster xmlns="uri:falcon:cluster:0.1"> name="undefined" description="undefined" color="undefined" tags=""></cluster>
<interfaces>
  <interface type="readonly" endpoint="http://sandbox.hortonworks.com:50070" version="2.2.0">
    </interface>
  <interface type="write" endpoint="hdfs://sandbox.hortonworks.com:8020" version="2.2.0">
    </interface>
  <interface type="execute" endpoint="sandbox.hortonworks.com:8050" version="2.2.0">
    </interface>
  <interface type="workflow" endpoint="http://sandbox.hortonworks.com:11000/oozie/" version="4.0.0">
    </interface>
  <interface type="messaging" endpoint="tcp://sandbox.hortonworks.com:61616?daemon=true" version="5.1.6">
    </interface>
  <interface type="registry" endpoint="" version="">
    </interface>
</interfaces>
<locations>
  <location name="staging" path="">
    </location>
  <location name="temp" path="">
    </location>
  <location name="working" path="">
    </location>
  <location name="path" path="">
    </location>
</locations>
<ACL owner="ambari-qa" group="users" permission="0x755"/>
<properties>
  <property name="value"></property>
</properties>
</cluster>
```

Falcon UI – New Cluster Interface

Clicking on the Cluster button in the Falcon Web UI will bring up the New Cluster interface, which is a single-page that contains all of the settings necessary to create a cluster entity. The values that are defined are as follows:

The screenshot shows a form titled "New Cluster Information Form". The fields include:

- Name:** A text input field.
- Colo:** A text input field.
- Description:** A text input field.
- Tags:** A section with two input fields for "key" and "value", and a button "+ add tag".
- Access Control List:**
 - Owner:** A text input field.
 - Group:** A text input field containing "users".
 - Permissions:** A text input field containing "0x755".
- Interfaces:** A section with a list of interface types: readonly, write, execute, workflow, messaging, registry.

New Cluster Information Form

Name - Name of the cluster entity. Not necessarily the actual cluster name. Must be unique.

Colo and Description - Name and description of the data center.

Tags - Metadata tagging. Can be used to locate a specific Cluster entity while searching.

Access Control List - Access control list for this Cluster entity. Owner is the user that owns the entity. Group is one which has access to read. (The schema definition indicates that these permissions are not currently enforced, however they should be set appropriately so that Cluster entities do not have to be recreated for future upgrades. However, the entity definition page, drawn from this schema, contains no such qualification.

There are six different interface types that can be configured: readonly, write, execute, workflow, messaging, registry.

- **readonly** -- Required. Specifies the HDP cluster's hftp address, which is required for distcp (distributed copy) used in replication. It is the value of dfs.http.address in the cluster configuration. For example, hftp://node1.namenode:50070/
- **write** -- Required. Specifies the interface required to write to HDFS. It is the value of fs.default.name in the cluster configuration. For example, hdfs://node1.namenode:8020
- **execute** -- Required. Specifies the interface for the YARN ResourceManager, and is required to execute MapReduce jobs. It is the value of yarn.resourcemanager.address in the cluster configuration. For example: node1:8050
- **workflow** -- Required. Specifies the interface for the Oozie workflow engine. This is the Oozie URL – example: http://node2:11000/oozie
- **messaging** -- Required. Specifies the interface for sending feed availability and other alerts. Example: tcp://node1:61616?daemon=true

Note: Falcon comes installed with a preconfigured messaging service that can be used.

- **registry** -- Optional. Specifies the interface for HCatalog, and is used to register or deregister partitions in the Hive Metastore and to fetch events on partition availability.

Two additional sections of settings exist: Properties and Location.

Interfaces		
Type	Endpoint	Version
readonly	http://sandbox.hortonworks.com:50070	2.2.0
write	hdfs://sandbox.hortonworks.com:8020	2.2.0
execute	sandbox.hortonworks.com:8050	2.2.0
workflow	http://sandbox.hortonworks.com:11000/oozie/	4.0.0
messaging	tcp://sandbox.hortonworks.com:61616?daemon=true	5.1.6
<input type="checkbox"/> registry		
Properties		
name	value	
<input type="text"/>	<input type="text"/>	
+ add property		

Location	
name	path
staging	<input type="text"/>
temp	<input type="text"/>
working	<input type="text"/>
<input type="text"/>	<input type="text"/>
+ add location	

- **Properties**

A optional list of cluster properties as key-value pairs. Currently, primarily exists for specification of JMS implementation class, which for Falcon is default brokerimplclass = org.apache.activemq.ActiveMQConnectionFactory.

- **Location**

Specify HDFS locations for the staging, temp, and working directories of Falcon jobs on this cluster. The directories must exist prior to jobs being run, and must be owned by the falcon user.

Cluster Entity Prerequisites

Before you create a cluster entity, the following conditions must be met:

1) All necessary components must be installed and running

- HDP
- Oozie client and server
- Falcon

2) The directory structure on HDFS for the staging, temp, and working folders where the cluster entity stores the dataset must exist and be owned by the falcon user.

To create these directories in an HDFS directory named cluster01 run the following commands:

```
# su falcon
$ hdfs dfs -mkdir -p /apps/falcon/cluster01/staging
$ hdfs dfs -mkdir -p /apps/falcon/cluster01/working
$ hdfs dfs -mkdir -p /apps/falcon/cluster01/tmp
```

Important

Permissions on the cluster staging directory must be set to 777 (read/write/execute for owner/group/others). Only Oozie job definitions are written to the staging directory so setting permissions to 777 does not create any vulnerability.

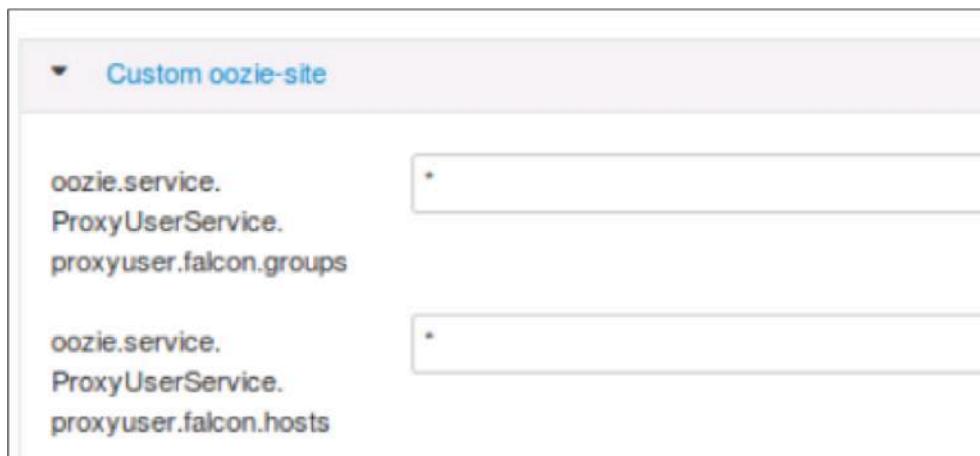
Continuing our example, you would run the following command:

```
$ hdfs dfs -chmod -R 777 /apps/falcon/cluster01/staging
```

Permissions Required for Oozie Registration:

Add the following properties to the Custom oozie-site section of **Oozie > Configs**. Then restart oozie and you should be able to submit the cluster entity.

```
oozie.service.ProxyUserService.proxyuser.falcon.hosts=*
oozie.service.ProxyUserService.proxyuser.falcon.groups=*
```



Oozie > Configs

Introduction to Falcon

Feed Entity

The screenshot shows the Falcon UI for creating a new feed entity. The interface is divided into five tabs: General, Properties, Location, Clusters, and Summary. The General tab is currently selected. It contains fields for Name, Description, Tags (with a key-value pair example), Groups (comma-separated), Access Control List (with Owner, Group, and Permissions), and Schema (with Location and Provider). To the right, there is an XML Preview window showing the generated XML code:

```
<feed xmlns='uri:falcon:feed:0.1'>
<clusters>
<cluster type='source'
<validity start='2015-12-15T19:02Z' end='2015-12-15T19:02Z'/>
<retention action='delete' />
<locations>
<location type='data'>
</location>
<location type='stats'>
</location>
<location type='meta'>
</location>
</locations>
</cluster>
</clusters>
<locations>
<location type='data' path='/'>
</location>
<location type='stats' path='/'>
</location>
<location type='meta' path='/'>
</location>
</locations>
</locations>
```

Falcon UI – New Feed Interface

Clicking on the Feed button in the Falcon Web UI will bring up the New Feed interface, which consists of multiple pages that contain all of the settings necessary to create a feed entity. Policies in a process entity include such things as:

- Which cluster the data will be fed into
- The HDFS path for the data files
- How often the data will be fed into the cluster
- How long to retain the data in the cluster
- The time period to expect data, which determines when data is late and must be processed in a different manner
- Where, and how often, to replicate the data

Feed definition would almost always be the responsibility of an HDP developer rather than an administrator.

Process Entity

The screenshot shows the Falcon UI's "New Process" interface. It's a five-step wizard:

- Step 1: General**: Fields include Name, Tags (key: value pairs), Workflow Name, Engine (radio buttons for Oozie, Pig, or Hive), Path (/), and Access Control List (Owner: ambari-qa, Group: users, Permissions: 0x755).
- Step 2: Properties**
- Step 3: Clusters**
- Step 4: Inputs & Outputs**
- Step 5: Summary**

XML Preview (shown on the right):

```
<process xmlns="uri:falcon:process:0.1">
<clusters>
<cluster>
<validity start='2015-12-15T19:11Z' end='2015-12-15T19:11Z'/>
</cluster>
</clusters>
<parallel></parallel>
<workflow path='/'>
<ACL owner='ambari-qa' group='users' permission='0x755'>
</ACL>
</process>
```

Falcon UI – New Process Interface

Clicking on the Process button in the Falcon Web UI will bring up the New Process interface, which consists of multiple pages that contain all of the settings necessary to create a process entity. The Process entity defines the cluster to process the data, the program to process the data, the Feed entity to use, and how often to retry failed data processing. Policy specifications in a Process entity include such things as:

- The name of the cluster where the process is executed
- The name of the input Feed entity, which defines the dataset to read
- The name of the output Feed entity, which defines the dataset to output
- The Hive, Pig, or Oozie processing task to use in the pipeline
- How frequently the process should be executed
- The number of retry attempts before reporting a failure
- The delay period between retry attempts
- A late data policy that specifies an alternate workflow to execute for data that arrives late based on definitions supplied in the Feed entity

Process definition would almost always be the responsibility of an HDP developer rather than an administrator.

Once a process entity has been created, it is considered the completion of a data pipeline.

Mirroring

The screenshot shows the 'Falcon Mirrors' section with a 'New Mirror' dialog. The 'General' tab is active. The 'Mirror Name' field is empty. Under 'Tags', there is a 'key' field containing 'falcon_mirroring_type' and a 'value' field containing 'HDFS'. The 'Mirror type' section shows 'File System' selected. The 'Source' section has 'Location: HDFS' selected, a dropdown for 'Path' with '-Select cluster-' and an empty 'Path' input, and a 'Run job here' checkbox. The 'Target' section has 'Location: HDFS' selected, a dropdown for 'Path' with '-Select cluster-' and an empty 'Path' input, and a 'Run job here' checkbox. The 'Validity' section shows a start date of '12/14/2015', an end date of 'mm/dd/yyyy', and times of '05:23 PM' for both start and end. The 'Send alerts to' section has an 'Email' input and a '+add alert' button. The 'Advanced options' section has a dropdown menu. At the bottom are 'Cancel' and 'Next' buttons.

Falcon UI – Mirror Entity Creation Screen

The Mirror button is a special-case Feed and Process entity creation interface where, on a single screen, data mirroring can be quickly and easily set up to replicate information between HDFS locations (which would usually span multiple clusters) or cloud-based data sources (Microsoft Azure or Amazon S3). Several settings are available to create a mirror job:

Mirror Name – The name of the mirror entity. Must be unique.

Tags – Used for metadata tagging. An example is provided in the UI, but this can be expanded.

Mirror Type -- Either **File System** (HDFS) or **Hive catalog** mirror type.

Source -- The location, name, and path of the cluster or Hive table that is to be mirrored, and whether the mirroring job runs on the source cluster.

Target -- The location, name, and path where the mirrored data is to be stored, and - if applicable - whether the mirroring job runs on the target cluster.

Validity – Once launched, when the mirroring process should be allowed to begin, and when it should be ended.

Send alerts to – Email address where administrative alerts should be sent.

The screenshot shows the 'Advanced options' configuration screen. It includes sections for Frequency (set to 'Every 5 minutes'), Allocation (Max Maps for Distcp: 5, Max bandwidth (MB): 100), Retry (Policy: PERIODIC, Delay: 30 minutes, Attempts: 3), and Access Control List (Owner: root, Group: users, Permissions: 0x755). At the bottom are 'Cancel' and 'Next' buttons.

Falcon UI – Advanced Mirror Options

Advanced Options -- How often the target cluster is updated, how much bandwidth to allocate to the mirroring job, retry policy, and the ACL for the mirror entity.

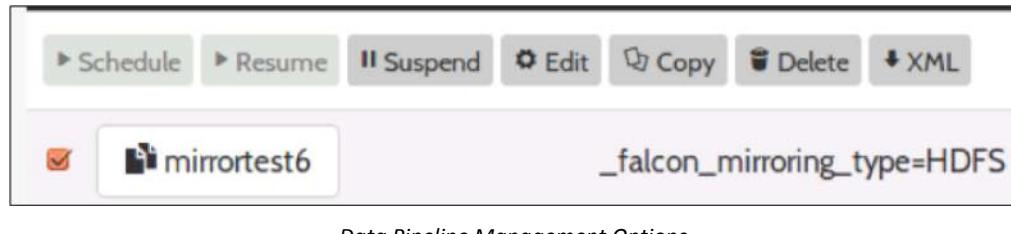
Falcon Search UI

The screenshot shows the Falcon Search UI interface. At the top is a search bar with 'Name: test'. Below it is a table with columns: Name, Tags, Cluster, Type, and Status. The table lists four entries, each with a checkbox and a small icon:

Name	Tags	Cluster	Type	Status
<input type="checkbox"/> mirortest6	_falcon_mirroring_type=HDFS	test	HDFS	SUSPENDED
<input type="checkbox"/> mirortest5	_falcon_mirroring_type=HDFS	test	HDFS	SUSPENDED
<input type="checkbox"/> mirortest4	_falcon_mirroring_type=HDFS	test	HDFS	SUSPENDED
<input type="checkbox"/> mirortest3	_falcon_mirroring_type=HDFS	test	HDFS	SUSPENDED

Falcon Search UI

The Falcon Web UI comes with a search functionality that allows you to search for and manage data pipelines via Process entity definitions by name.



Once the process appears in the search results, you can select it and perform management operations such as Schedule (initiate the data pipeline), Suspend and Resume, Edit the process entity definition, Copy a process entity, Delete a process entity, or download the process entity XML definition.

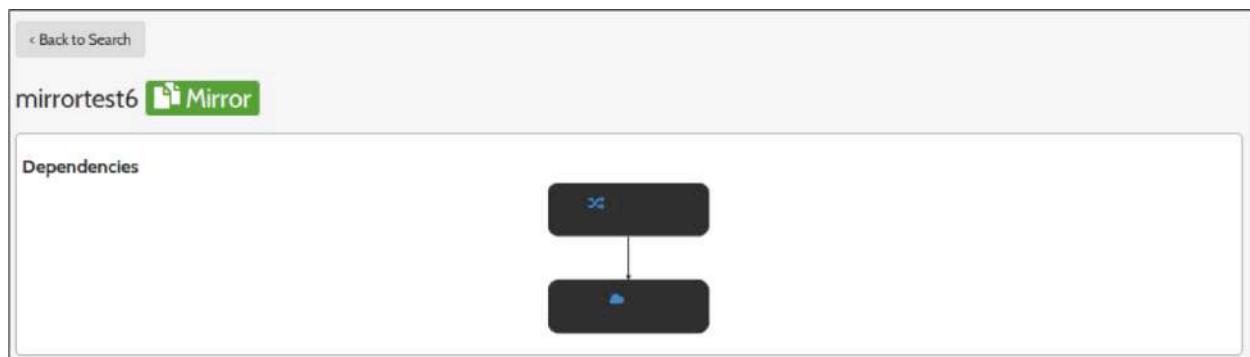
Notes

A data pipeline may be edited at any time, in real time, with the exception of the validation dates. If those need to be changed, a new data pipeline must be created.

To completely remove a data pipeline from Falcon, you must first suspend it (which stops it from running) and then delete it (which removes it from the system.)

You can also click on the data pipeline name for additional management options.

Process and Pipeline Management



Dependencies Pane

Click on a process / data pipeline to manage it. The resulting page gives you the ability to view dependencies, properties, and instances. The Dependencies pane will graphically illustrate any dependent processes, feeds, and cluster relationships for the data pipeline. For a mirror pipeline, this should only be the availability of the clusters (the screenshot was taken from a mirror job replicating data from one location in HDFS to another on the same cluster.)

Properties Pane

The screenshot shows the 'Properties' pane for a data pipeline. It includes sections for 'Mirror', 'Access Control List', 'Workflow', and 'Timing'. The 'Mirror' section shows the name 'mirrortest6' and tags '_falcon_mirroring_type = HDFS' and 'No tags selected'. The 'Access Control List' section shows owner 'root' and group 'root' with permissions '0x755'. The 'Workflow' section shows the name 'mirrortest6-WF', engine 'oozie', and path '/apps/data-mirroring/workflows/hdfs-replication-workflow.xml'. The 'Timing' section is partially visible.

Properties Pane

The Properties pane displays the configured settings for this data pipeline.

Instances Pane

The screenshot shows the 'Instances' pane displaying five instances of a pipeline. Each instance is listed with its name, start time, end time, and status. All instances are marked as 'SUCCEEDED'.

Instance Name	Start Time	End Time	Status
2015-12-21T17:40Z	12/21/2015 12:40	12/21/2015 12:41	SUCCEEDED
2015-12-21T17:35Z	12/21/2015 12:35	12/21/2015 12:35	SUCCEEDED
2015-12-21T17:30Z	12/21/2015 12:30	12/21/2015 12:31	SUCCEEDED
2015-12-21T17:25Z	12/21/2015 12:25	12/21/2015 12:25	SUCCEEDED
2015-12-21T17:20Z	12/21/2015 12:20	12/21/2015 12:20	SUCCEEDED

Instances Pane

The Instances pane displays data on individual instances of the pipeline, and also allows you to manage individual instances. Management functions include downloading log files, suspend / resume functions, manually re-running an instance, and killing an instance.

You can also click on the instance name for further management options.

Viewing Log Files

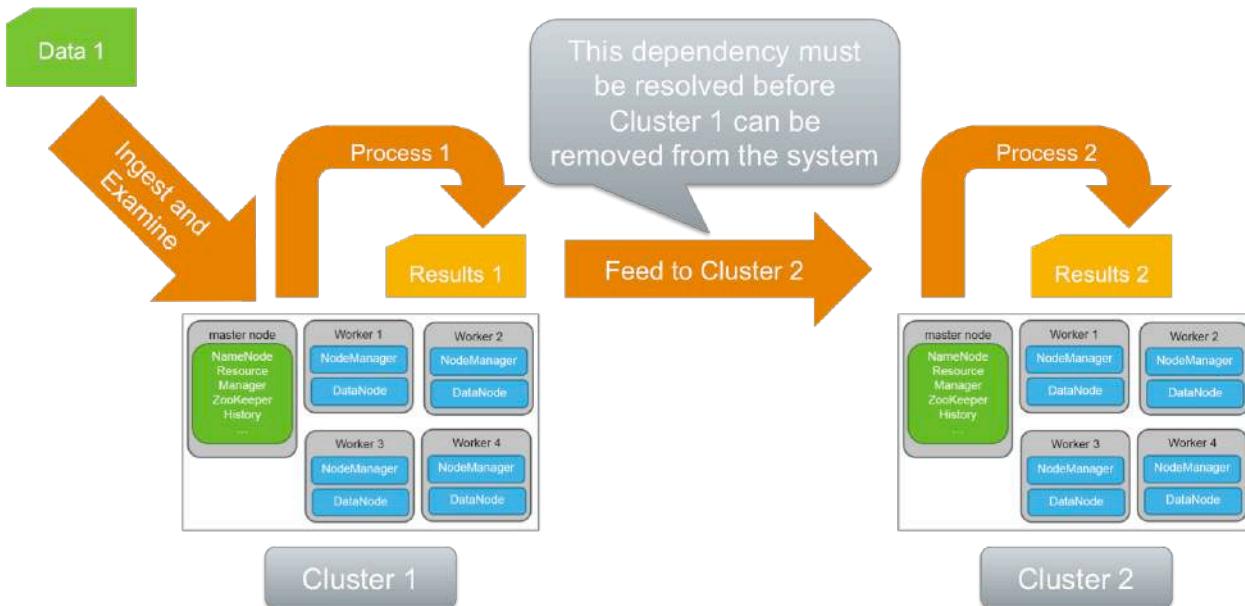
```
[root@node2 logs]# ls
falcon.application.log.2015-12-09  falcon.application.log.2015-12-26  falcon.audit.log.2015-12-09
falcon.application.log.2015-12-10  falcon.application.log.2015-12-27  falcon.audit.log.2015-12-15
falcon.application.log.2015-12-11  falcon.application.log.2015-12-28  falcon.metric.log
falcon.application.log.2015-12-12  falcon.application.log.2015-12-29  falcon.metric.log.2015-12-09
falcon.application.log.2015-12-13  falcon.application.log.2015-12-30  falcon.metric.log.2015-12-11
falcon.application.log.2015-12-14  falcon.application.log.2015-12-31  falcon.metric.log.2015-12-14
falcon.application.log.2015-12-15  falcon.application.log.2016-01-01  falcon.metric.log.2015-12-15
falcon.application.log.2015-12-16  falcon.application.log.2016-01-02  falcon.metric.log.2015-12-16
falcon.application.log.2015-12-17  falcon.application.log.2016-01-03  falcon.metric.log.2015-12-21
falcon.application.log.2015-12-18  falcon.application.log.2016-01-04  falcon.metric.log.2015-12-24
falcon.application.log.2015-12-19  falcon.application.log.2016-01-05  falcon.out.2015120921571449716271
falcon.application.log.2015-12-20  falcon.application.log.2016-01-06  falcon.out.2015121610361450280210
falcon.application.log.2015-12-21  falcon.application.log.2016-01-07  falcon.out.2015121611041450281844
falcon.application.log.2015-12-22  falcon.application.log.2016-01-08  falcon.security.audit.log
falcon.application.log.2015-12-23  falcon.application.log.2016-01-09  falcon.security.audit.log.2015-12-09
falcon.application.log.2015-12-24  falcon.application.log.2016-01-10  retry
falcon.application.log.2015-12-25  falcon.application.log.2016-01-11  testfile
falcon.application.log.2015-12-25  falcon.audit.log                testfile2
```

Pipeline Log Output

While pipeline logs can be gathered from the Falcon Web UI, additional logging information can be located on the Falcon Server itself at `/usr/hdp/current/falcon-server/logs`. For recent errors, the `falcon.application.log` file can be helpful for troubleshooting.

Managing Dependencies

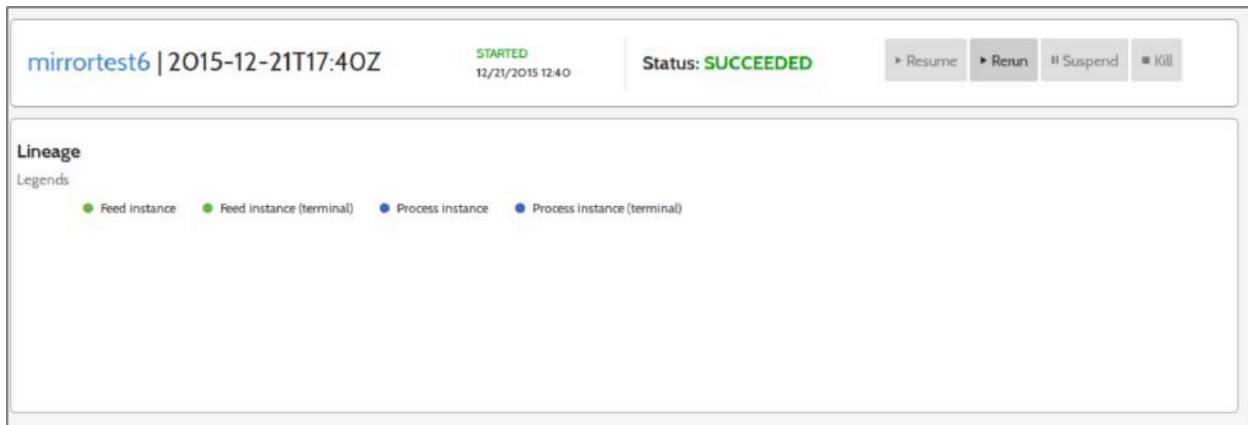
Cross-entity dependencies in Falcon are important because a dependency cannot be removed from a pipeline until all the dependents are first removed.



Cluster 2 Processing Depends on Cluster 1 Processing

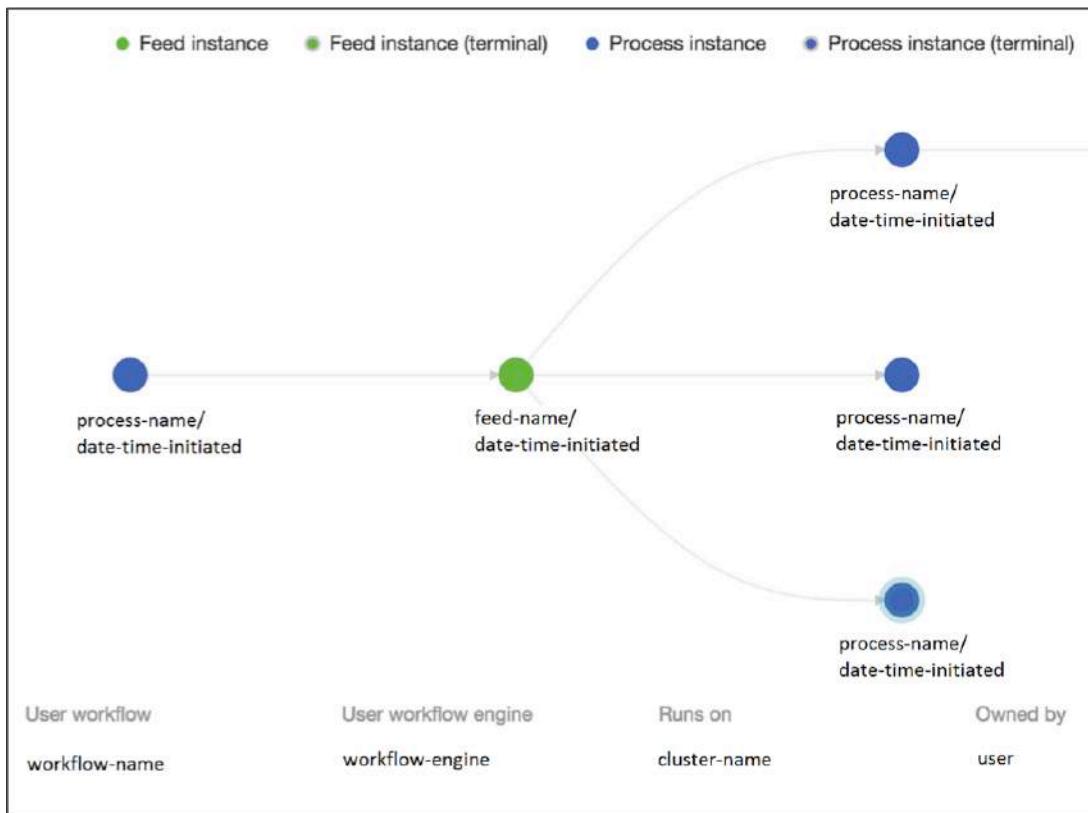
For example, if Falcon manages two clusters (Cluster 1 and Cluster 2) and Cluster 1 is going to be taken down, you must first resolve the Cluster 2 dependencies as processes on Cluster 2 currently depend on processing that takes place on Cluster 1. As Falcon manages more clusters, the ability to view these dependencies in a graphical format becomes crucial.

Lineage



Lineage Pane

When you click on an instance, the most significant additional feature available is the Lineage pane. In this, you can view the processes, data feeds, and clusters being utilized by the instance.



Lineage of a Mirroring Job

For a mirroring job, this feature may not be extremely valuable, but for more complex data pipelines it can be critical in trying to track down issues or for historical reference.

This page left blank intentionally.

Knowledge Check

Questions

- 1) The tool for governance of enterprise metadata is _____.
- 2) True or False: Falcon allows a developer to determine what happens when expected data does not arrive on time.
- 3) True or False: Falcon is an independent tool, however Oozie or some other workflow engine can enhance its capabilities.
- 4) Troubleshooting a data pipeline instance by tracking the processes and feeds back in time is an example of using _____.
- 5) List the three types of Falcon entities.
- 6) List the two types of data storage formats that Falcon works with.

Answers

- 1) The tool for governance of enterprise metadata is _____.

Answer: Atlas

- 2) True or False: Falcon allows a developer to determine what happens when expected data does not arrive on time.

Answer: True

- 3) True or False: Falcon is an independent tool, however Oozie or some other workflow engine can enhance its capabilities.

Answer: False. Falcon requires Oozie (or theoretically, some other workflow engine) in order to operate.

- 4) Troubleshooting a data pipeline instance by tracking the processes and feeds back in time is an example of using _____.

Answer: Lineage

- 5) List the three types of Falcon entities.

Answer: Cluster, Feed, and Process

- 6) List the two types of data storage formats that Falcon works with.

Answer: HDFS and HCatalog

Summary

- Falcon is a governance tool for managing data pipelines and lifecycle management
- Falcon utilizes a separate workflow engine (Oozie)
- Falcon manages data pipelines through three entity types
 - Cluster
 - Feed
 - Process
- Data pipeline management is handled via the Falcon Web UI, although management via CLI and a REST API are also available
- The Falcon Web UI provides significant visualization capabilities that enable historical tracking and troubleshooting of data pipelines

Automate Cluster Provisioning Using Ambari Blueprints

Lesson Objectives

After completing this lesson, students should be able to:

- ✓ Summarize the purpose and benefits of Ambari Blueprints
- ✓ Recall the processes used to deploy a cluster using Ambari Blueprints
- ✓ Configure Ambari Blueprints logical cluster configuration files
- ✓ Configure Ambari Blueprints cluster creation configuration files
- ✓ Configure Ambari Blueprints host creation configuration files
- ✓ Identify Ambari Blueprints configuration property precedence and best practices

Blueprint Scenarios

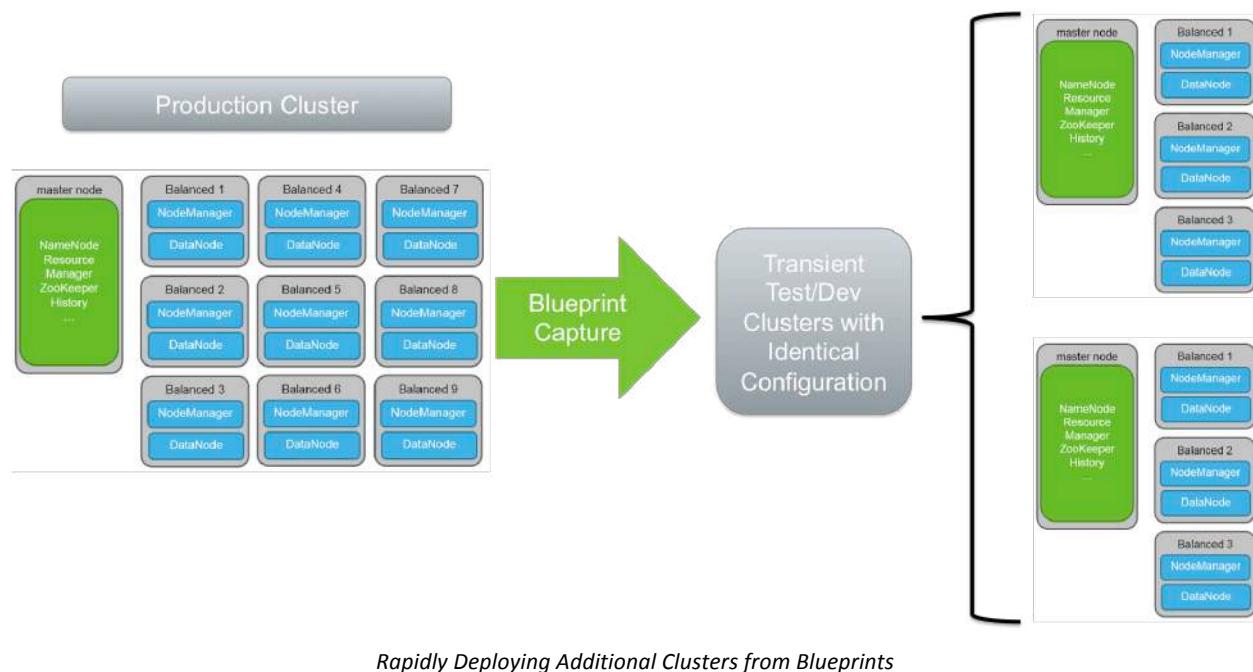
Ambari Blueprints are a declarative definition of a cluster. Declarative means Ambari Blueprints provides a high-level structure that allows an automated installation of an HDP cluster without requiring that the exact procedure be defined. They allow the creation of an HDP cluster via REST API calls, without the need to use the Ambari UI.

Ambari Blueprints have a multitude of uses. In this course, we will cover a couple of common ones, which will introduce the capabilities of this tool.

Scenario 1: Test/Dev Cluster Deployment

One scenario in which Ambari Blueprints can be used to great effect is in the deployment of logical copies of a production cluster for test and development purposes.

It is easy to imagine scenarios in which an HDP administrator might be called on to deploy test and development clusters that are identically configured to an existing production environment. For example, an administrator may want to test the effects of installing a new software stack prior to deploying it in an active production environment. Perhaps a development team is testing a new application, but needs to know how it will perform prior to deploying in a production environment. A third example might be a company deciding to use an outside group for application development, and they want to provide them with an accurate environment to test in without ever granting someone outside the company access to production resources.



Rapidly Deploying Additional Clusters from Blueprints

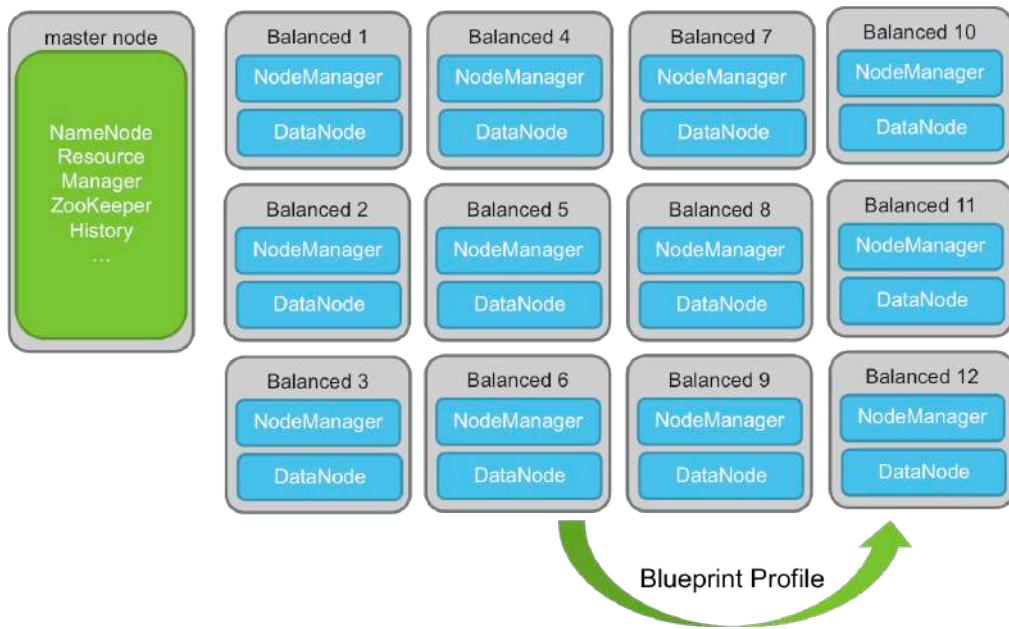
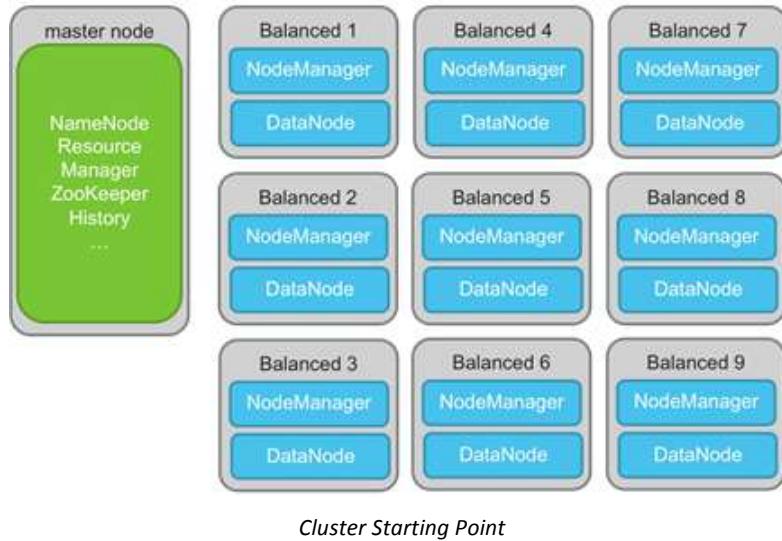
Whatever the case may be, the ability to capture the existing state of a cluster – including the configuration of all HDP components installed – in a blueprint, and then use that blueprint to rapidly deploy additional clusters that have identically configured master and worker nodes can save a significant amount of time, as well as reduce potential for human error during test/dev cluster deployment.

Note

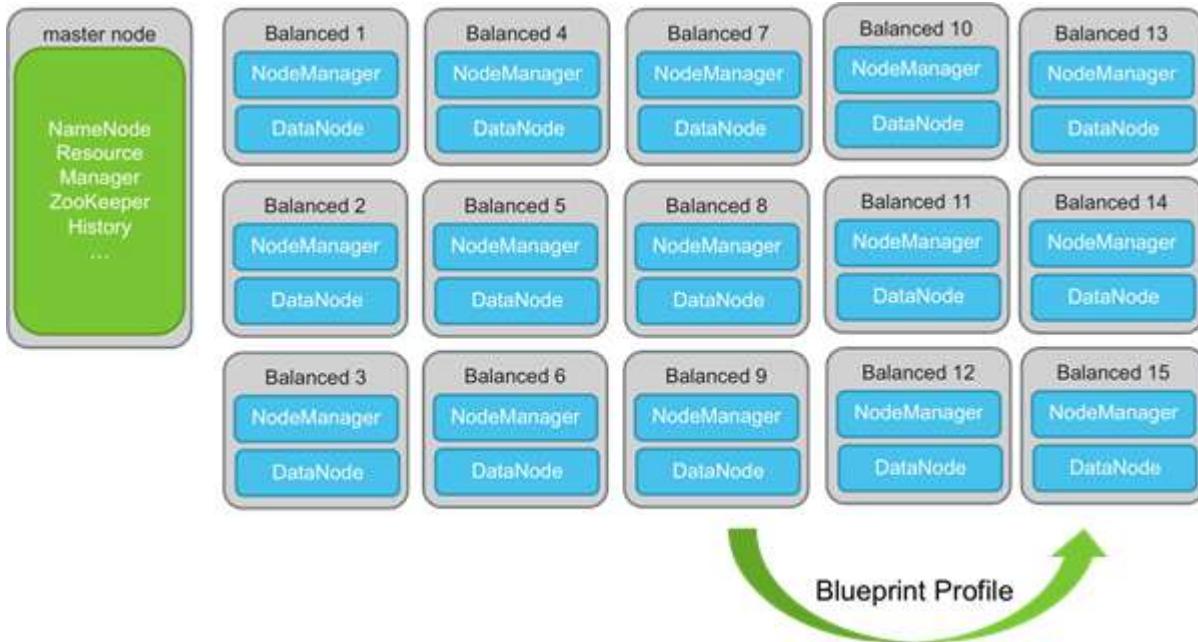
The test/dev clusters are *logical* mirrors of the production cluster. There is no requirement that the test/dev clusters have the exact same hardware or number of nodes. The production cluster might well be deployed in a pure physical environment, and the two test/dev clusters could be deployed in an on-premise virtual environment or a remotely-managed cloud environment.

Scenario 2: Cluster Scale-Out

A second scenario in which Ambari Blueprints can be extremely helpful is in the automated scaling out of an existing cluster.



In the scenario above, the HDP administrator uses a blueprint profile of a host group to deploy additional nodes with the same configuration in an automated fashion.

*Repetitive Scaling Out with Blueprints*

This operation can be repeated as often as needed whenever additional nodes need to be added to an existing cluster, without ever having to log into the Ambari Web UI.

While these two scenarios do not represent everything that can be done with Ambari Blueprints, they serve to illustrate the power of using Ambari Blueprints to deploy and scale out HDP clusters.

Benefits

The benefits of using Ambari Blueprints for cluster deployment and scale-out include:

- Logical template allows for quick, repeatable deployments of test and dev clusters that mirror production clusters except in size
- Enable the same cluster configuration to be provisioned across various environments, including physical or virtual machines
- Enable a template host configuration to be provisioned on new cluster nodes
- Reduce human error impact on cluster deployment or scale-out

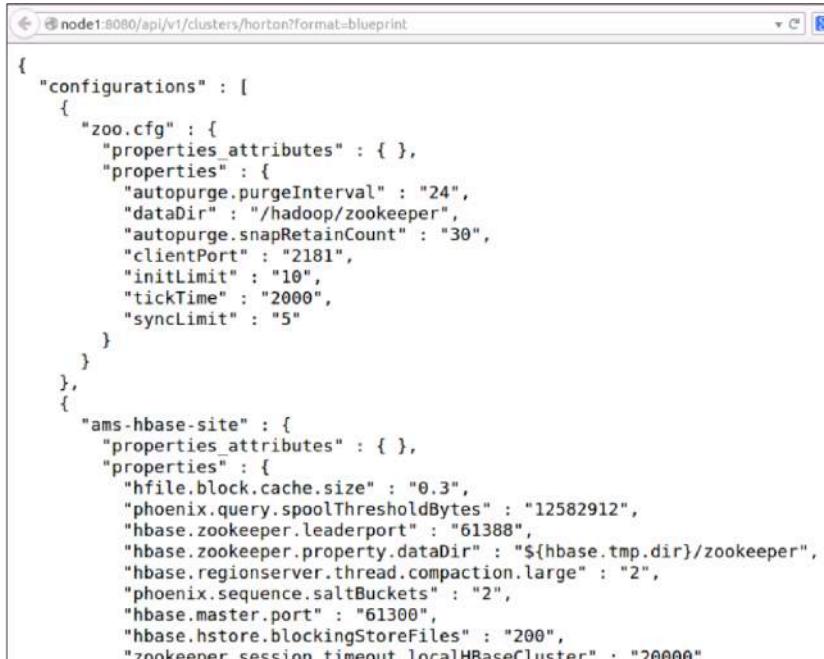
Blueprint Usage Overview

Deploying a new cluster or additional nodes on an existing cluster requires several steps. These include:

- 1) Create or capture a blueprint JSON file as a cluster template
- 2) Install an Ambari server for the new cluster
- 3) Register the blueprint to the new Ambari server using an API call
- 4) Optional (but recommended): Create local repo configuration JSON files
- 5) Optional (but recommended): Use API calls to set local repo locations
- 6) Create a cluster creation template JSON file
- 7) Deploy the cluster creation template using an API call
- 8) Install the Ambari agent software on nodes to be deployed
- 9) Register the Ambari agents with the Ambari server
- 10) Confirm successful automated cluster deployment

In this section, we will walk through the process of downloading a logical blueprint from an existing cluster and using it as part of the process to deploy a new cluster using Ambari Blueprints.

View Existing Cluster Configuration



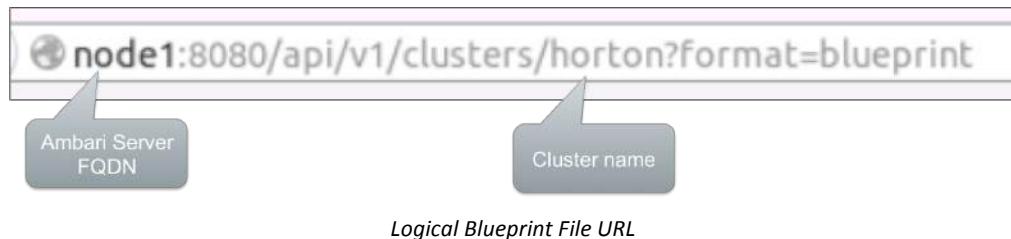
The screenshot shows a browser window with the URL `node1:8080/api/v1/clusters/horton?format=blueprint`. The page displays a large JSON object representing the cluster configuration. The JSON structure includes sections for configurations, properties, and specific service settings like ZooKeeper and HBase.

```
{
  "configurations": [
    {
      "zoo.cfg": {
        "properties_attributes": {},
        "properties": {
          "autopurge.purgeInterval": "24",
          "dataDir": "/hadoop/zookeeper",
          "autopurge.snapRetainCount": "30",
          "clientPort": "2181",
          "initLimit": "10",
          "tickTime": "2000",
          "syncLimit": "5"
        }
      }
    },
    {
      "ams-hbase-site": {
        "properties_attributes": {},
        "properties": {
          "hfile.block.cache.size": "0.3",
          "phoenix.query.spoolThresholdBytes": "12582912",
          "hbase.zookeeper.leaderport": "61388",
          "hbase.zookeeper.property.dataDir": "${hbase.tmp.dir}/zookeeper",
          "hbase.regionserver.thread.compaction.large": "2",
          "phoenix.sequence.saltBuckets": "2",
          "hbase.master.port": "61300",
          "hbase.hstore.blockingStoreFiles": "200",
          "zookeeper.session.timeout.localHBaseCluster": "20000"
        }
      }
    }
  ]
}
```

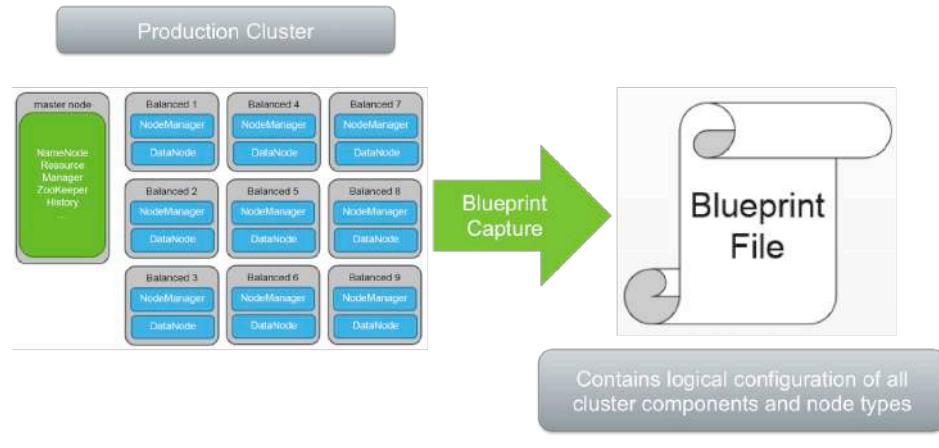
File used as Logical Blueprint for Deploying other Clusters

Every HDP cluster contains a file that can be retrieved via an API call and used as the logical blueprint for deploying other clusters. To quickly view this file, visit the following URL in a web browser:

`http://<AmbariServer>:8080/api/v1/clusters/<cluster name>?format=blueprint`



Step 1: Capture Blueprint



Capturing the Blueprint File

In order to deploy a cluster using Ambari Blueprints, you first have to create the JSON file that contains the logical configuration of your cluster, its services, and the various ways in which the nodes in the cluster can be deployed. One of the easiest ways to do this is to start from an existing cluster and pull the blueprint template using an API call. This file can then be used as the template for a new cluster deployment.

To do this using cURL, use the following command to capture the blueprint and write it to a file:

```
# curl -H "X-Requested-By: user" -X GET -u user:pass http://<node>:8080/api/v1/clusters/<cluster>?format=blueprint > <blueprint-name>
```

Verify Blueprint Creation

To verify that the blueprint was created, use a text editor such as vi or nano to view the file you retrieved and wrote with the previous API call.

Nano is a lightweight, easy-to-use text editor for Linux that can be used if an administrator is not already familiar with vi. If nano is not installed on your system, you can install it with the following command:

```
# yum -y install nano
```

```
# yum -y install nano  (or use vi if familiar with it)
# nano <blueprint-name>
```

Installing nano

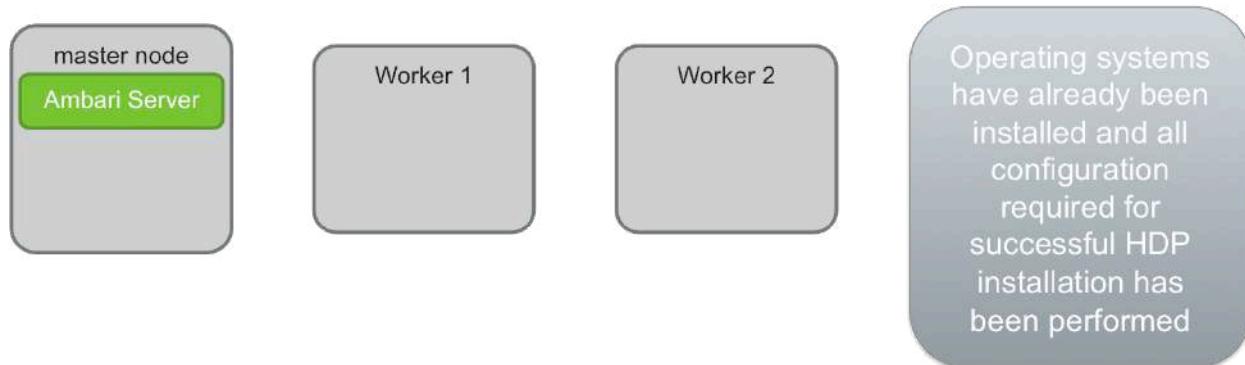
You can then view the blueprint file with the following command:

```
# nano <blueprint name>
```

```
{
  "configurations" : [
    {
      "zoo.cfg" : {
        "properties_attributes" : { },
        "properties" : {
          "autopurge.purgeInterval" : "24",
          "dataDir" : "/hadoop/zookeeper",
          "autopurge.snapRetainCount" : "30",
          "clientPort" : "2181",
          "initLimit" : "10",
          "tickTime" : "2000",
          "syncLimit" : "5"
        }
      }
    },
    {
      "ams-hbase-site" : {
        "properties_attributes" : { }
      }
    }
  ]
}
```

Viewing the Blueprint File with nano

Step 2: Install Ambari Server



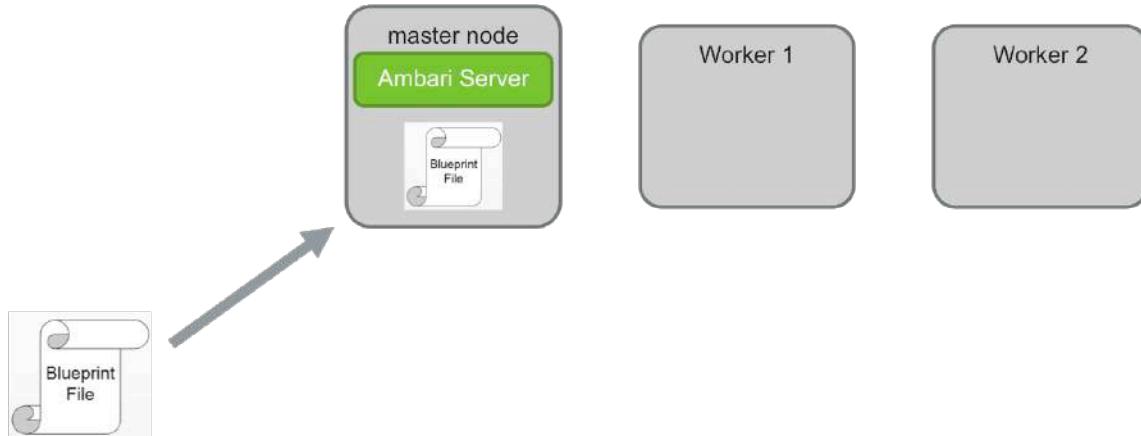
Configured Servers on an HDP Cluster

Prior to installing HDP on any set of hardware, via whatever mechanism, all operating systems need to have been installed and configured per HDP documentation. This is not part of what Ambari Blueprints does, but can be managed through third-party tools – for example, Puppet, Chef, Ansible, or a host of other automated deployment tools on the market today.

Once all servers have been configured, the first step in using Ambari Blueprints is to install an Ambari Server on your cluster. The commands to do this are:

```
# yum -y install ambari-server
# ambari-server setup -s
# ambari-server start
```

Step 3: Register Blueprint File with Ambari



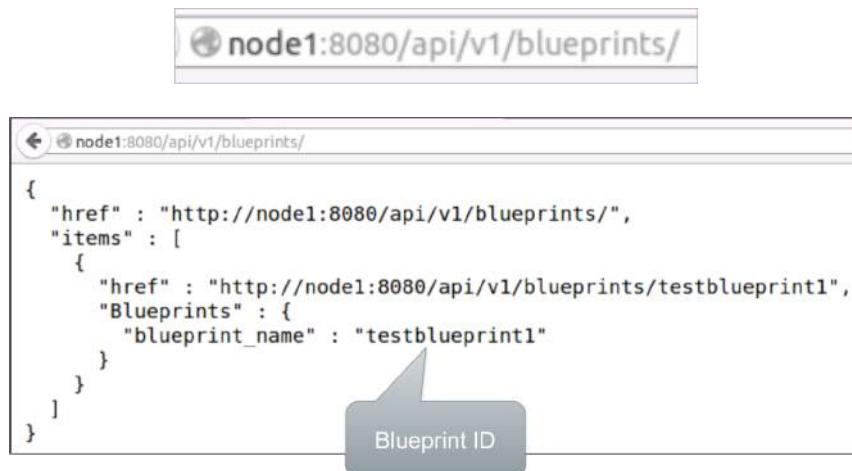
Registering the Logical Blueprint File

Once Ambari Server has been installed, the administrator must register the logical blueprint file created earlier. This can be accomplished using curl via the following API call:

```
# curl -u user:pass -i -H "X-Requested-By: user" -X POST -d @<blueprint-name>
http://<node>:8080/api/v1/blueprints/<blueprint-identification>
```

Verify Blueprint Registration

To verify the logical blueprint was successfully registered, visit the following URL using a web browser and confirm that the blueprint name exists: <http://<AmbariServer>:8080/api/v1/blueprints>

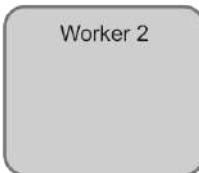
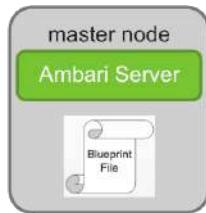


Verifying Blueprint Registration

Step 4 (Optional): Local Repo Configuration Files

Using local repo files is strongly recommended when using Ambari Blueprints, especially for large cluster deployments. Two files needs to be created for every base operating system present in the cluster – one file for the HDP software, and another for the HDP-UTILS software.

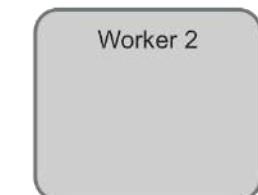
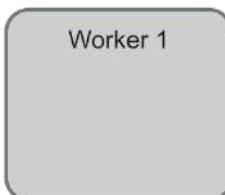
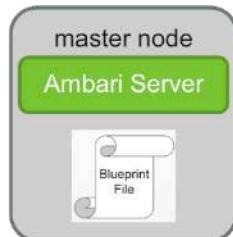
The format for the repo files to be created is as follows:



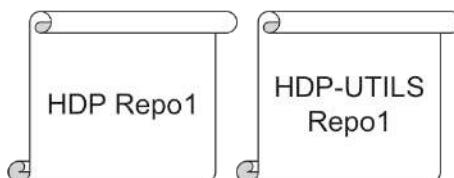
```
{
  "Repositories" :
    {
      "base_url" : "http://<local repo
location>",
      "verify_base_url" : true
    }
}
```

Creating Local Repo Configuration File

If the cluster contains a mixture of operating systems, this may require a number of files be created.



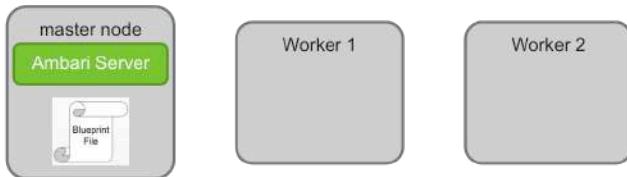
```
{
  "Repositories" :
    {
      "base_url" : "http://<local repo
location>",
      "verify_base_url" : true
    }
}
```



Creating Multiple Local Repo Configuration Files

Step 5 (Optional): Set Local Repo Locations

Once the local repo files have been created, they need to be registered with the Ambari Server via an API call.



```
# curl -u user:pass -i -H "X-Requested-By: user" -X PUT -d @<Repo Configuration File>
http://<node>:8080/api/v1/stacks/HDP/versions/<V>/operating_systems/<OS>/repositories/
<Repo Configuration Being Edited>
```

Note

All other API calls discussed will be `GET` or `POST`, but this operation requires a `PUT` call to be executed.

This step would need to be repeated for each HDP and HDP-UTILS local repo that would need to be configured prior to launching the installation.

Verify Local Repo Configurations

```
{
  "href" : "http://node1:8080/api/v1/stacks/HDP/versions/2.3/operating_systems/redhat6/repositories/HDP-2.3/",
  "Repositories" : [
    {
      "base_url" : "http://node1/hdp/HDP-2.3",
      "default_base_url" : "http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.3.0.0",
      "latest_base_url" : "http://public-repo-1.hortonworks.com/HDP/centos6/2.x/updates/2.3.2.0",
      "mirrors_list" : null,
      "os_type" : "redhat6",
      "repo_id" : "HDP-2.3",
      "repo_name" : "HDP",
      "stack_name" : "HDP",
      "stack_version" : "2.3"
    }
  ]
}
```

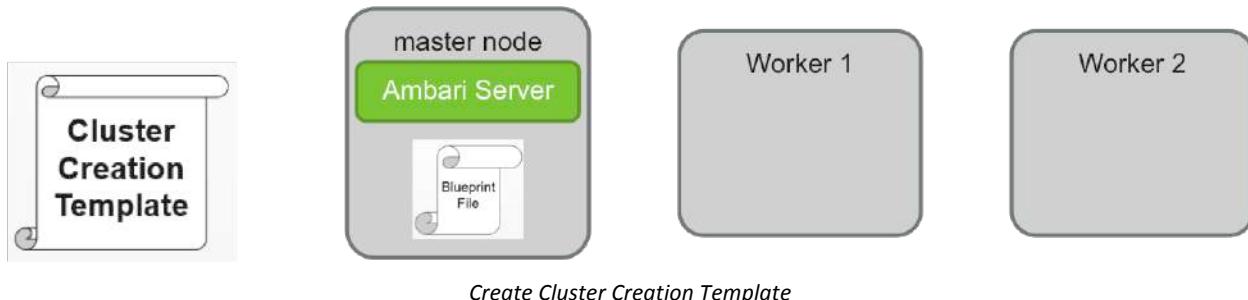
Verifying the Local Repo Configurations

To verify that the local repos have been configured properly, open a web browser to the following pages:

```
http://<AmbariServer>:8080/api/v1/stacks/HDP/versions/<version>/operating_systems/<OS>/repositories/<LocalRepoSet>
```

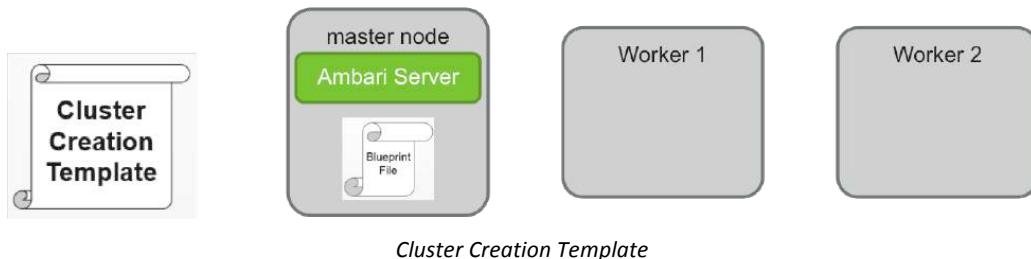
Make sure that the `base_url` value is set appropriately.

Step 6: Create Cluster Creation Template



The next step is to create the cluster creation template file. This file contains the physical cluster layout (via specific FQDN specification) or programmatic cluster layout (as in number of nodes to install, and optionally what operating system they should have, how many CPUs, etc.) and assigns hosts to the configuration groups specified in the logical blueprint.

Step 7: Deploy Cluster Creation Template



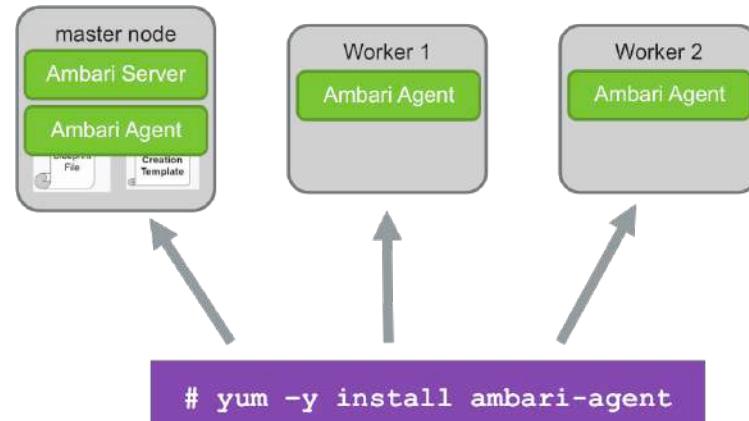
Once the cluster creation template has been created, initiate the cluster installation process. This can be done using cURL to execute the following API call:

```
# curl -u user:pass -i -H "X-Requested-By: user" -X POST -d @<Cluster Creation File>
http://<node>:8080/api/v1/clusters/<cluster name>
```

Note

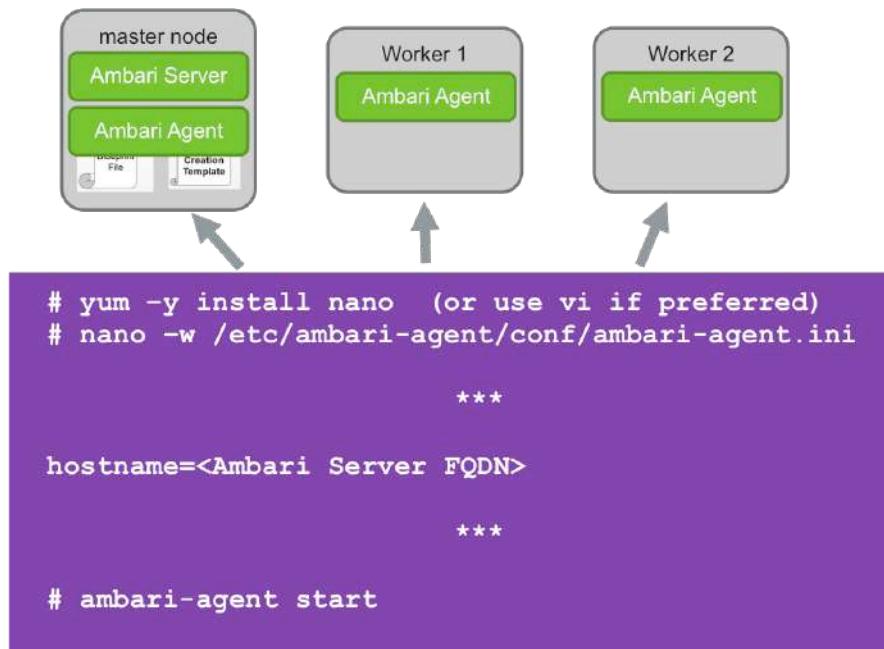
This step can be performed before the Ambari agent software has been installed and configured – as we have outlined in the process for this lesson – or the admin has the option to first install all Ambari agents and register them with the Ambari Server prior to initiating the cluster installation. If this step is done beforehand, the Ambari Server will simply initiate the appropriate actions as new nodes become available on an ongoing basis.

Step 8: Install Ambari Agents

*Installing Ambari Agents*

Before HDP will install via Ambari Blueprints, the Ambari agent software must be installed, configured, and started on each node. To install, use the command pictured above.

Step 9: Register Ambari Agents with Ambari Server

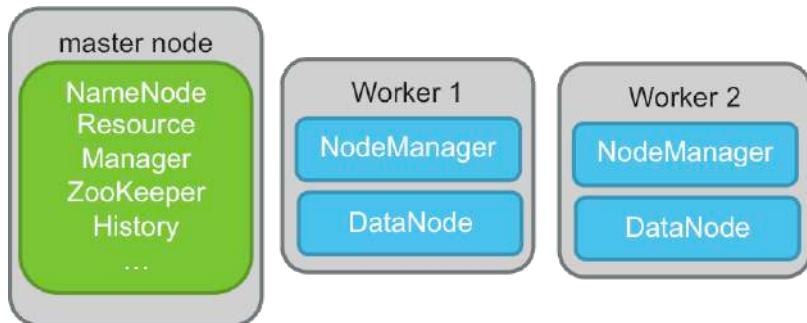
*Registering Ambari Agents with Ambari Server*

Once the Ambari agent software has been installed, use a text editor such as nano or vi to edit the `/etc/ambari-agent/conf/ambari-agent.ini` file. The hostname value should be modified to the FQDN of Ambari Server.

Once the `ambari-agent.ini` file has been configured, run the following command to start the Ambari agent and register it with Ambari Server:

```
# ambari-agent start
```

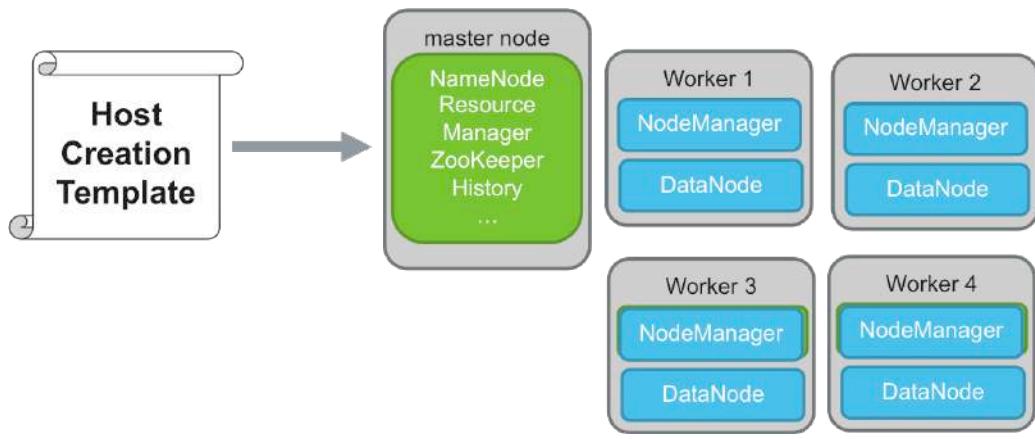
Step 10: Confirm Successful Cluster Deployment



Successful Cluster Deployment

Assuming the cluster creation template file has already been activated, HDP installation should begin as soon as the Ambari agent registers with Ambari Server. This can be monitored and confirmed via the Ambari UI.

Add Hosts via Blueprints



Adding Hosts via Blueprints

Once a cluster has been created using Ambari Blueprints, it is possible to scale that cluster out using additional API calls. This requires the creation of a host creation template file, which can then be posted to Ambari Server using cURL via the following command:

```
# curl -u user:pass -i -H "X-Requested-By: user" -X POST -d @<Host Creation File>
http://<node>:8080/api/v1/clusters/<cluster name>/hosts
```

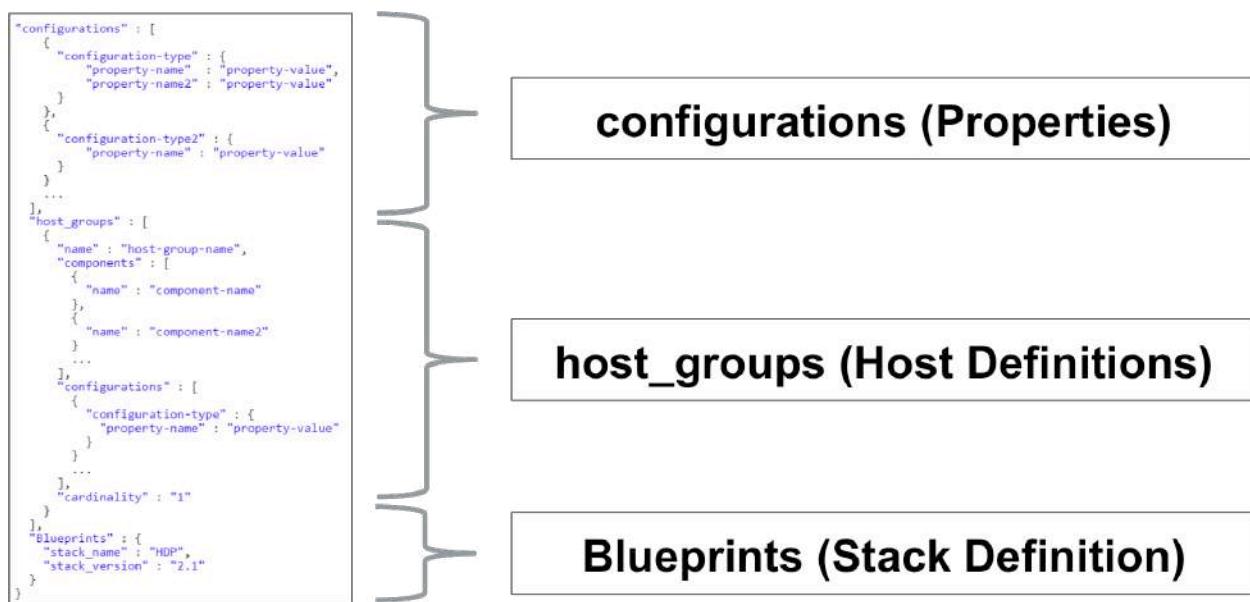
Note

The cluster must have been installed from blueprints, otherwise this will return an error.

Logical Cluster Definition File

The Ambari Blueprints deployment process requires the creation of several configuration files that are read via API calls. The first one we will examine is the logical cluster creation file, often simply referred to as the blueprint or logical blueprint.

Blueprints Logical Cluster Definition



A logical cluster definition is generally composed of three main sections: configurations, host_groups, and Blueprints.

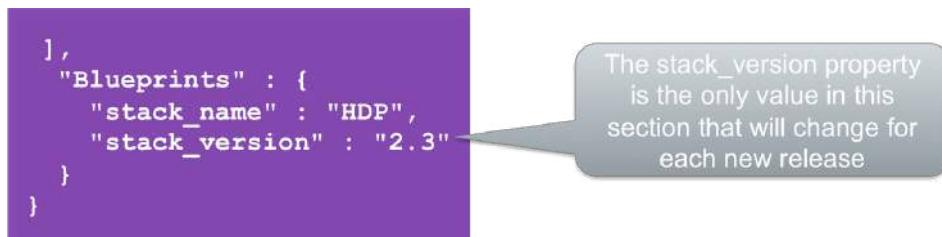
The file is formatted in JSON, which is beyond the scope of this class to discuss. Fortunately, an administrator can use an exported logical blueprint from an existing server, or one of many pre-canned and online templates available to help build or modify existing files. For more information, visit the Apache documentation on Ambari Blueprints:

<https://cwiki.apache.org/confluence/display/AMBARI/Blueprints>

“Blueprints” Section

The Blueprints section of the logical cluster configuration file contains the stack name (which refers to the distribution of Hadoop being installed – in our case, HDP) and the version of that stack being deployed. Generally speaking, it is only the stack_version property that will change as the software updates over time.

The general form of the Blueprints section is as follows:



Blueprints Section of Cluster Definition File

“host_groups” Section

The host_groups section of the logical blueprint serves as a template to define the various configurations in which hosts can be deployed separated out by group name. The section contains four subsections: name, components, configurations, and cardinality.

```
],
  "host_groups" : [
    {
      "name" : "<host group name>",
      "components" : [
        {
          "name" : "<component name>"
        },
        {
          "name" : "<component2 name>"
        }
        ...
      ],
      "configurations" : [
        {
          "<configuration name>" : {
            "<property>" : "<value>"
          }
        }
        ...
      ],
      "cardinality" : "1"
    }
  ],
}
```

“host_groups” Section of Cluster Definition File

host_groups “name” Section

Each host group definition must have a name, which is an arbitrary label applied to the group. Human-created logical blueprints will usually provide a descriptive name for each host group, for example master1, master2, gateway, worker1, worker2, etc. Ambari-generated logical blueprints will simply name hosts groups host_group_#, with the # starting at 1 and incrementing with each new host group configuration in the cluster.

A single-node cluster definition created for proof-of-concept purposes might have a only single host group defined. A small, simple cluster might have two – one for the master node, and one for all worker nodes. Larger, more complex clusters with services spread across various nodes and multiple config groups might potentially have dozens of host groups defined. The primary concern is that for every different host configuration in the cluster, a host group must be defined.

The general form of the host_groups name section is as follows:

```
],
  "host_groups" : [
    {
      "name" : "<host group name>",

```

host_groups “name” Section

Note: The order of these sections is irrelevant. Thus, human-generated host group definitions usually place the name value near the top of the host group definition, whereas Ambari-generated logical blueprints will place the name value near the bottom of the definition.

Host_groups “components” Section

Host groups must be configured to install the components that they should run. At least one named component must exist per host group. The general form of the components section is as follows:

```
"components" : [
    {
        "name" : "<component name>"
    },
    {
        "name" : "<component2 name>"
    }
    ...
]
```

host_groups “compondents” Section

Component Names for Blueprints

The component names represent the parts of the various HDP services that should be installed on a given node. Each service has at least one component, but most of them have two or more that can be installed.

A list of all services available in the current HDP stack can be viewed in a web browser at the following address:

<http://<AmbariServer>:8080/api/v1/stacks/HDP/versions/<version>/services>

To see the components that can be configured for a host configuration group add the following to the end of the address above:

[.../<SERVICE>/components](http://<AmbariServer>:8080/api/v1/stacks/HDP/versions/<version>/services/<SERVICE>/components)

As a reference guide the complete list of components can be downloaded as a PDF, sorted and arranged for your convenience:

<http://tinyurl.com/HDP23Blueprint>

HOST GROUP CONFIGURATION OPTIONS	
SERVICES	HOST COMPONENTS
ACCUMULO	ACCUMULO_CLIENT ACCUMULO_GC ACCUMULO_MONITOR ACCUMULO_TRACER ACCUMULO_TSERVER
AMARI_METRICS	METRICS_COLLECTOR METRICS_MONITOR
ATLAS	ATLAS_SERVER
FALCON	FALCON_CLIENT FALCON_SERVER
FLUME	FLUME_HANDLER
CANCLIA	CANCLIA_MONITOR

Some components are client components while others are server components. Some components require additional configuration information be supplied in order to install correctly, while most can simply be named and installed using default settings. Remember, the easiest way to ensure that your logical blueprint is configured correctly is to retrieve it from a cluster that has been configured properly with the various components.

Host_groups “configurations” Section

The configurations portion of host_groups is completely optional. If used, the values contained therein will be set for all hosts in that host group, overriding the default settings for the cluster. This will generally manifest as a separate config group for the hosts in that host group when viewed in Ambari UI. For example, an administrator can create a separate YARN config group by overriding yarn-site values in a given host group. An example of how those settings might be configured might be the following:

```
"yarn-site" : {
  "yarn.acl.enable" : "true",
  "yarn.node-labels.enabled" : "true",
}
```

The general form of the configurations settings section is as follows:

```
"configurations" : [
  {
    "<configuration name>" : {
      "<property>" : "<value>"
    }
  }
]
```

When exporting a logical blueprint from Ambari Server, if no config groups have been defined on the cluster, this section will usually be empty.

host_groups “cardinality” section

The cardinality portion of the host_groups section in the logical blueprint file is an optional value that is ignored during cluster deployment. The primary purpose of this value is to provide a hint for the HDP administrator trying to sort out the purpose of various host groups exported from Ambari Server. The cardinality value will be the number of hosts in the cluster that matched the host group configuration. Thus, for example, a value of 1 in a large cluster would likely indicate that a particular host group definition is that of a master server or some other utility server, whereas significantly larger numbers would indicate the host group defines a worker node configuration.

The general form of the cardinality section is as follows:

```
],
  "cardinality" : "1"
}
```

host-groups Section Summary

```
],
  "host_groups" : [
    {
      "name" : "<host group name>",
      "components" : [
        {
          "name" : "<component name>"
        },
        {
          "name" : "<component2 name>"
        }
        ...
      ],
      "configurations" : [
        {
          "<configuration name>" : {
            "<property>" : "<value>"
          }
        }
        ...
      ],
      "cardinality" : "1"
    }
  ],
}
```

host-groups Section

In summary, there will be one host group for every different host configuration in your cluster. Large, complex clusters with scattered services across lots of nodes and various config groups based on hardware, etc. will end up with a significant number of host groups. On the other hand, if most master services are installed across a few nodes and all hosts are identically configured, a cluster with thousands of worker nodes might just have one host group definition for all of them.

“configurations” Section

The configurations section of the logical blueprint file usually resides at the top of the file. Its structure and format are identical to the configurations sub-section of the host_groups section. The difference is that settings present here will be applied to all hosts in a cluster deployed from this logical blueprint unless overridden by configurations set in host_groups configurations in the same file or in the same section of the cluster creation template.

As with the host_groups configurations section, cluster-wide configurations are completely optional. The general form of the configurations section (and any configurations section in any Ambari Blueprints file) is as follows.

```
{  
  "configurations" : [  
    {  
      "<configuration type>" : {  
        "<property>" : "<value>",  
        "<property2>" : "<value>"  
      }  
    },  
    {  
      "<configuration type2>" : {  
        "<property>" : "<value>"  
      }  
    }  
    ...  
  ],
```

“configurations” Section

Simple Logical Blueprint Example

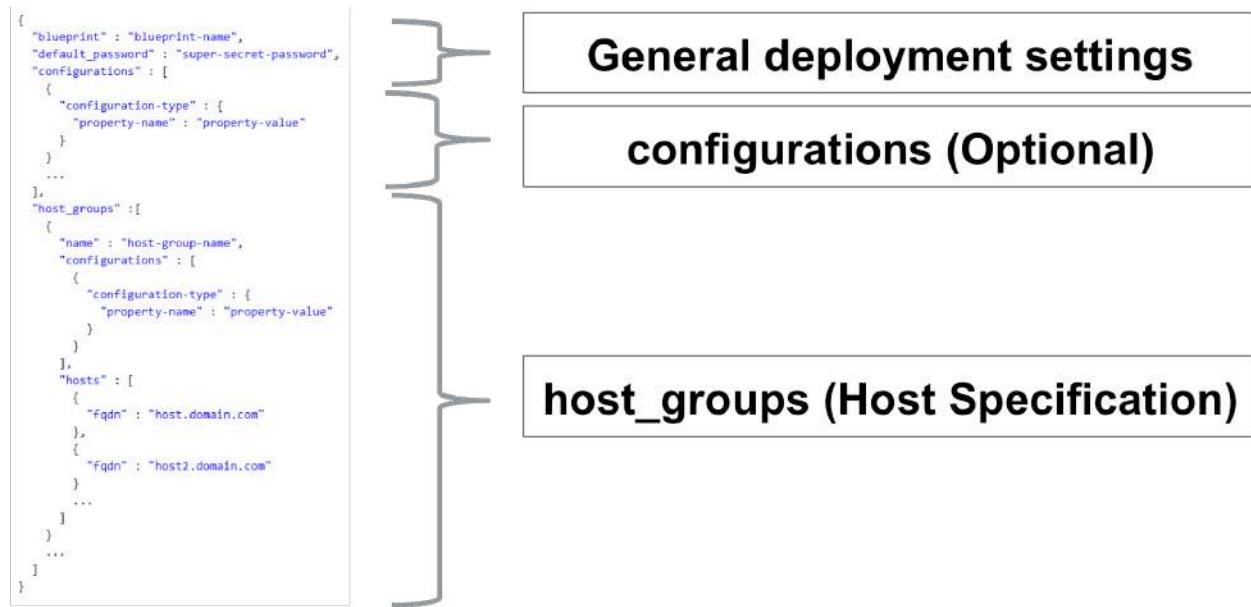
While logical blueprints can be very large and complex, they can also be very simple depending on the need. For example, consider the following logical blueprint, which could be used to deploy a single-node cluster for POC purposes:

```
{  
    "configurations" : [],  
    "host_groups" : [  
        {  
            "name" : "host_group_1",  
            "components" : [  
                { "name" : "NAMENODE" },  
                { "name" : "SECONDARY_NAMENODE" },  
                { "name" : "DATANODE" },  
                { "name" : "HDFS_CLIENT" },  
                { "name" : "RESOURCEMANAGER" },  
                { "name" : "NODEMANAGER" },  
                { "name" : "YARN_CLIENT" },  
                { "name" : "HISTORYSERVER" },  
                { "name" : "MAPREDUCE2_CLIENT" },  
                { "name" : "ZOOKEEPER_SERVER" },  
                { "name" : "ZOOKEEPER_CLIENT" },  
                { "name" : "METRICS_COLLECTOR" },  
                { "name" : "METRICS_MONITOR" }  
            ],  
            "cardinality" : "1"  
        }  
    ],  
    "Blueprints" : {  
        "stack_name" : "HDP",  
        "stack_version" : "2.3"  
    }  
}
```

Logical Blueprint for a Single-Node Cluster

Cluster Creation Template File

The cluster creation template file, also sometimes referred to as the host mapping file, is similar to the logical blueprint in many ways, but has some distinct features that allow it to be used to deploy, rather than define, HDP clusters.



Cluster Creation Template File

A cluster creation template is generally composed of three main sections: general deployment settings, configurations, and host_groups.

General Cluster Deployment Settings

There are three general deployment settings which are usually configured at the top of a cluster creation template: blueprint, config_recommendation_strategy, and default_password.

The blueprint setting identifies the name of the logical blueprint against which this cluster creation template should be applied. This blueprint must have already been registered with Ambari Server prior to initiating a cluster deployment.

The config_recommendation_strategy setting is an optional setting which specifies how Ambari stack advisor recommendations – which, in an Ambari UI-based install, would appear during the installation wizard process – will be applied. The default setting is NEVER_APPLY, which means stack advisor recommendations are ignored. If the setting is not configured in the cluster creation template, this will be the result. The other two options are ONLY_STACK_DEFAULTS_APPLY, which applies stack advisor settings for included services, and ALWAYS_APPLY, which applies all recommended configuration properties on all services.

The default_password setting is another optional setting which specifies a password to use for HDP services if they are not specifically specified in a configurations section, such as the blueprint (not advised) or the cluster creation template configurations section.

The general form of these settings is as follows:

```
{  
  "blueprint" : "<blueprint name>",  
  "config_recommendation_strategy" : "ONLY_STACK_DEFAULTS_APPLY",  
  "default_password" : "<password>",
```

"configurations" Section

The configurations section of the cluster creation file is configured identically to the other configurations sections we have discussed. If it contains any content, the settings will override any conflicting settings in the logical blueprint configurations section. However, if the blueprint contains conflicting settings in its host_groups configurations section, those settings will take precedence over the general cluster settings specified here.

The general form is the same as in the logical blueprint file:

```
{  
  "configurations" : [  
    {  
      "<configuration type>" : {  
        "<property>" : "<value>",  
        "<property2>" : "<value>"  
      }  
    },  
    {  
      "<configuration type2>" : {  
        "<property>" : "<value>"  
      }  
    }  
    ...  
  ],
```

"host_groups" Section

```
[  
  "host_groups" : [  
    {  
      "name" : "host-group-name",  
      "configurations" : [  
        {  
          "configuration-type" : {  
            "property-name" : "property-value"  
          }  
        },  
        "fqdn" : "host.domain.com"  
      ],  
      "hosts" : [  
        {  
          "fqdn" : "host2.domain.com"  
        },  
        ...  
      ]  
    ]  
  ]  
]
```

"host-groups" Section

The host_groups section of the cluster creation template defines the topology of the HDP cluster. The section has five possible subsections: name, host_count, host_predicate, configurations, and hosts.

host_groups "name" Section

The name subsection specifies the name of the host group template, as configured in the logical blueprint, to be used for deploying hosts in the cluster. The name must exactly match a host_group section name in the logical blueprint. The general format of this value is as follows:

```
[  
  "host_groups" : [  
    {  
      "name" : "<host group name>",  
    }  
  ]  
]
```

host_groups "host_count" Section

The host_count subsection specifies the number of hosts to be installed with the configuration in the named host group. By default, it will simply choose from any available host with a registered Ambari agent installed. There is no requirement that anything about the hosts – such as underlying operating system, number of CPUs, etc. – match from one host to another. However, this can be paired with a host_predicate value, which provides more control over what hosts will be configured with this host group.

The general form of the host_count section is as follows:

```
"host_count" : "2",
```

Where "2" represents the number of hosts to be installed.

host_groups "host_predicate" Section

The host_predicate section of host_groups is an optional setting that can be used in conjunction with host_count to control how a cluster is deployed. It specifies criteria that must be met in order for a host to be part of a given host group, which may be useful when a cluster contains hosts with different operating systems or hardware characteristics.

The general form of the host_predicate setting is as follows:

```
"host_predicate" : "Hosts/os_type=centos6&Hosts/cpu_count=4",
```

Multiple host predicates can be assigned by separating a configured list with an & symbol. For example:

```
"host_predicate" : "Hosts/os_type=centos6&Hosts/cpu_count=4",
```

This would specify that the host group would only be applied to hosts running Centos 6 which also had four CPUs.

A complete list of configuration options and values can be viewed in a web browser at the following address:

<http://<AmbariServer>:8080/api/v1/hosts/<node>>

host_groups "configurations" Section

The configurations section of the host_groups configuration section of the cluster creation template file is configured identically to the other configurations sections we have discussed. If it contains any content, the settings will override any conflicting settings anywhere else in any file.

The general form is the same as in the logical blueprint file:

```
{
  "configurations" : [
    {
      "<configuration type>" : {
        "<property>" : "<value>",
        "<property2>" : "<value>"
      }
    },
    {
      "<configuration type2>" : {
        "<property>" : "<value>"
      }
    }
    ...
  ],
}
```

host_groups “hosts” Section

An HDP administrator has the option to specify specific hosts to be deployed rather than a number and optional predicates. This is accomplished by specifying a hosts value instead of the host_count setting. The hosts are manually identified by their fully qualified domain name (FQDN), one by one. Each host must be specifically named and with a correctly formatted FQDN in order for it to be installed as part of the host group. This might be useful in a multi-host environment where an administrator wanted to configure a specific host or hosts as master nodes, or other special-purpose uses.

The general form of the hosts section is as follows:

```
"hosts" : [
    {
        "fqdn" : "<host1 FQDN>"
    },
    {
        "fqdn" : "<host2 FQDN>"
    }
    ...
]
```

Simple Cluster Creation Template Example

While cluster creation templates can be long and complex, they can also be quite short and simple based on the cluster being installed. For example, examine the following cluster creation template file, which would be used for the installation of a single-node cluster for POC purposes and specifies a particular host on which the single-node cluster should be installed:

```
{
    "blueprint" : "testblueprint1",
    "default_password" : "changeme",
    "host_groups" : [
        {
            "name" : "host_group_1",
            "hosts" : [
                {
                    "fqdn" : "node1"
                }
            ]
        }
    ]
}
```

Cluster Creation Template Example: Single-node Cluster, Specific Host Installation

Host Creation Template File

The host creation template file is similar in structure to the cluster creation template file, only with fewer configurations (no cluster-wide configurations can be specified, for example). The API call also posts to a different URI.

Host creation template files can be configured to use host_count, and optionally host_predicate, or alternatively can specify individual nodes to scale out to.

Specify Nodes Host Creation Template Example

If an HDP administrator wants to scale out the HDP cluster to specific hosts, each node must be individually specified in the host creation template file along with the logical blueprint and the host_group name. Each host must have the information, even if all hosts will be using the same logical blueprint and the same host group. The general structure of such a file is as follows:

```
[  
  {  
    "blueprint" : "<logical blueprint>",  
    "host_group" : "<named host group>",  
    "host_name" : "<host1 FQDN>"  
  },  
  {  
    "blueprint" : "<logical blueprint>",  
    "host_group" : "<named host group>",  
    "host_name" : "<host2 FQDN>"  
  }  
]
```

Specify Nodes Host Creation Template Example

If the specific hosts do not matter, or if there are only a set number of hosts available, then the host_count / host_predicate mechanism can be employed in the host creation template file. The general form of this type of host creation template file would be as follows:

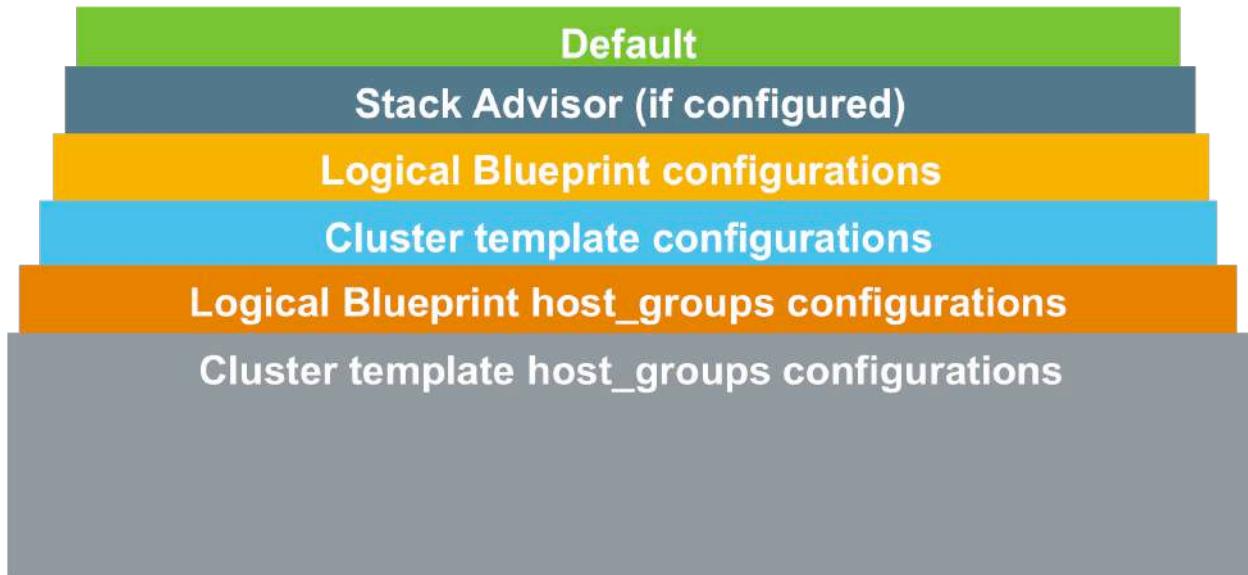
```
[  
  {  
    "blueprint" : "<logical blueprint>",  
    "host_group" : "<named host group>",  
    "host_count" : 30,  
    "host_predicate" : "Hosts/os_type=centos6&Hosts/cpu_count=4"  
  }  
]
```

Configuration Property Best Practices

Cluster configuration settings can come from the following locations:

- Default installation values
- Stack advisor suggested values
- Blueprint (logical configuration) file
 - configuration section
 - host_groups configuration section
- Cluster or host creation template file
 - configuration section
 - host_groups configuration section

Precedence Diagram



The precedence order of host-relevant host configuration properties and settings, when conflicting values exist, is as follows:

- 1) Default values can be overwritten by stack advisor values, if configured to do so in the cluster creation template file.
- 2) These values will be overwritten by cluster-wide configurations specified in the logical blueprint file, which will be overwritten by cluster-wide configurations specified in the cluster creation template file.
- 3) Configurations specified under host_groups in the logical blueprint file will override values in all other files and locations, with one exception. They can be overwritten themselves if host_groups configurations settings have been specified in the cluster creation template file.

Sensitive Configuration Information in Files

Sometimes sensitive information must be placed in Ambari Blueprints files in order for a cluster to be deployed correctly. When this is necessary, an administrator should **always** place such information in the cluster creation template rather than the logical blueprint. The logical blueprint is stored on the Ambari server and is easily accessible via an API call, whereas cluster creation templates are not stored nor accessible once uploaded.

One example where this might apply: Oozie is managed via oozie-site configurations, and requires a JDBC password and database URL in order to function properly. While these items could be included in the logical blueprint, that could pose a significant security risk. Rather than the blueprint, an administrator would want to put these configurations in the cluster creation template, as follows:

```
"configurations": [
  { "oozie-site": {
      "oozie.service.JPAService.jdbc.password" : "<password>",
      "oozie.service.JPAService.jdbc.url" : "<JDBC URL>",
      ...
    } },
]
```

Non-sensitive Oozie configuration settings could go in the logical blueprint, or could also be listed in the cluster configuration template to ensure continuity.

Knowledge Check

Questions

- 1) Name the four types of files one might use when deploying or scaling a cluster using Ambari Blueprints.
- 2) True or False: Ambari Server must be installed in order to use Ambari Blueprints for cluster installation.
- 3) True or False: Ambari agents can be installed as part of a blueprint-based installation.
- 4) True or False: A configuration specified in the cluster creation template will override the same configuration specified anywhere in the logical blueprint.
- 5) Where should sensitive information such as passwords and internal URLs be defined – the logical blueprint or the cluster creation template? Why?
- 6) What are the minimum number of configurations that must be specified across blueprints and cluster creation template files?

Answers

- 1) Name the four types of files one might use when deploying or scaling a cluster using Ambari Blueprints.

Answer: Logical Configuration Blueprint, Repo Configuration File, Cluster Creation Template, Host Creation Template

- 2) True or False: Ambari Server must be installed in order to use Ambari Blueprints for cluster installation.

Answer: True

- 3) True or False: Ambari agents can be installed as part of a blueprint-based installation.

Answer: False

- 4) True or False: A configuration specified in the cluster creation template will override the same configuration specified anywhere in the logical blueprint.

Answer: False – host_groups configurations in the logical blueprint will override general cluster configuration settings in the cluster creation template

- 5) Where should sensitive information such as passwords and internal URLs be defined – the logical blueprint or the cluster creation template? Why?

Answer: Cluster creation template – because information in a blueprint is stored and retrievable via an API call.

- 6) What are the minimum number of configurations that must be specified across blueprints and cluster creation template files?

Answer: Zero

Summary

- Ambari Blueprints are a declarative cluster definition
- Can be used to easily replicate the logical structure of an existing cluster across multiple new clusters (example: test/dev) via REST API calls
- Can be used to easily scale existing clusters that were installed via Ambari Blueprints
- Requires creation of a logical blueprint file and a cluster creation template
- Can also utilize local repo configuration files and host creation templates
- Requires preconfigured operating systems, pre-installed Ambari Server, and Ambari agents installed and manually registered across all nodes

Classes Available Worldwide Through Our Partners



Study Options Worldwide

In combination with our partner providers, classes are often available in numerous locations across the world.



Private On-site Training

Hortonworks training in-house covers all of our basic coursework, and provides a more intimate setting for 6 or more students.

[Contact us for more details](#)



Learn from the company focused solely on Hadoop.



What Makes Us Different?

1. Our courses are designed by the **leaders** and **committers** of Hadoop
2. We provide an **immersive** experience in **real-world** scenarios
3. We prepare you to **be an expert** with highly valued, **fresh skills**
4. Our courses are available **near you**, or accessible **online**

Hortonworks University courses are designed by the leaders and committers of Apache Hadoop. We provide immersive, real-world experience in scenario-based training. Courses offer unmatched depth and expertise available in both the classroom or online from anywhere in the world. We prepare you to be an expert with highly valued skills and for Certification.