
Apache Ranger
HORTONWORKS
UNIVERSITY

Objectives

- Describe Apache Ranger
- Understand the need for Apache Ranger
- Discuss Apache Ranger Architecture
- Configure Apache Solr
- Install and Configure Apache Ranger
- Optional Apache Ranger Configurations


What is Apache Ranger
Objectives
277
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

What is Apache Ranger

- Framework to Enable, Monitor, & Manage Comprehensive Data Security Across the Apache Hadoop Platform
- Centralized Security Administration to Manage All Security Related Tasks in a Central UI or using REST APIs.
- Fine Grained Authorization for a Specific Action and/or Operation with Hadoop Component/ Tool
- Enhanced Support for Different Authorization Methods
  - Role Based Access Control,
  - Attribute Based Access Control
- Centralize Auditing of User Access and Administrative Actions Within All the Hadoop Components

http://ranger.apache.org

Ranger is a framework to enable, monitor and manage comprehensive data security across the Hadoop platform.

The vision with Ranger is to provide comprehensive security across the Apache Hadoop ecosystem. With the advent of Apache YARN, the Hadoop platform can now support a true data lake architecture. Enterprises can potentially run multiple workloads, in a multi tenant environment. Data security within Hadoop needs to evolve to support multiple use cases for data access, while also providing a framework for central administration of security policies and monitoring of user access.

Please read the FAQs if you need to understand how it works over Apache Hadoop components.

What is Apache Ranger

Centralized Security Administration has Four Aspects

- Authentication – Previously Discussed
  - Effected by Kerberos in native Apache Hadoop
  - Secured by the Apache Knox Gateway Via the HTTP/REST API
- Authorization
  - Fine-Grained Access Control Provides Flexibility in Defining Policies on
    - The Directory and File Level, Via HDFS
    - The Database, Table and Column Level, Via Hive
    - The Table, Column Family and Column Level, Via Hbase

Centralized security administration in a Hadoop environment has four aspects:

Authentication

Effected by Kerberos in native Apache Hadoop, and secured by the Apache Knox Gateway via the HTTP/REST API. (For further information, see the Apache Knox Gateway Manager Guide.)

Authorization

Fine-grained access control provides flexibility in defining policies...

on the folder and file level, via HDFS

on the database, table and column level, via Hive

What is Apache Ranger

Centralized Security Administration has Four Aspects

- Audit
  - Controls Access into the System Via Extensive User Access Auditing in HDFS, Hive, HBase at...
    - IP Address
    - Resource/Resource Type
    - Timestamp
    - Access Granted or Denied
- Data Protection Provided By
  - Wire Encryption
  - File/Directory Encryption Via HDFS Encryption
  - File/Table Encryption Via HDFS Encryption and Hortonworks Partners
  - File/Column Encryption Via Hortonworks partners

Centralized security administration in a Hadoop environment has four aspects:

Authentication

Effected by Kerberos in native Apache Hadoop, and secured by the Apache Knox Gateway via the HTTP/REST API. (For further information, see the Apache Knox Gateway Manager Guide.)

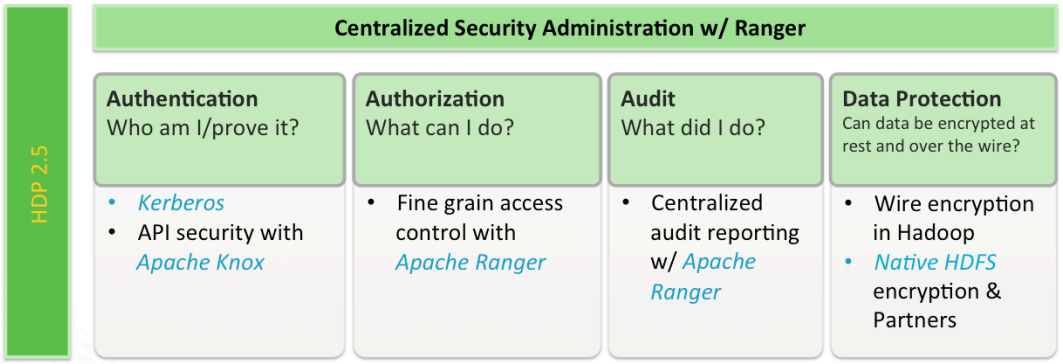
Authorization

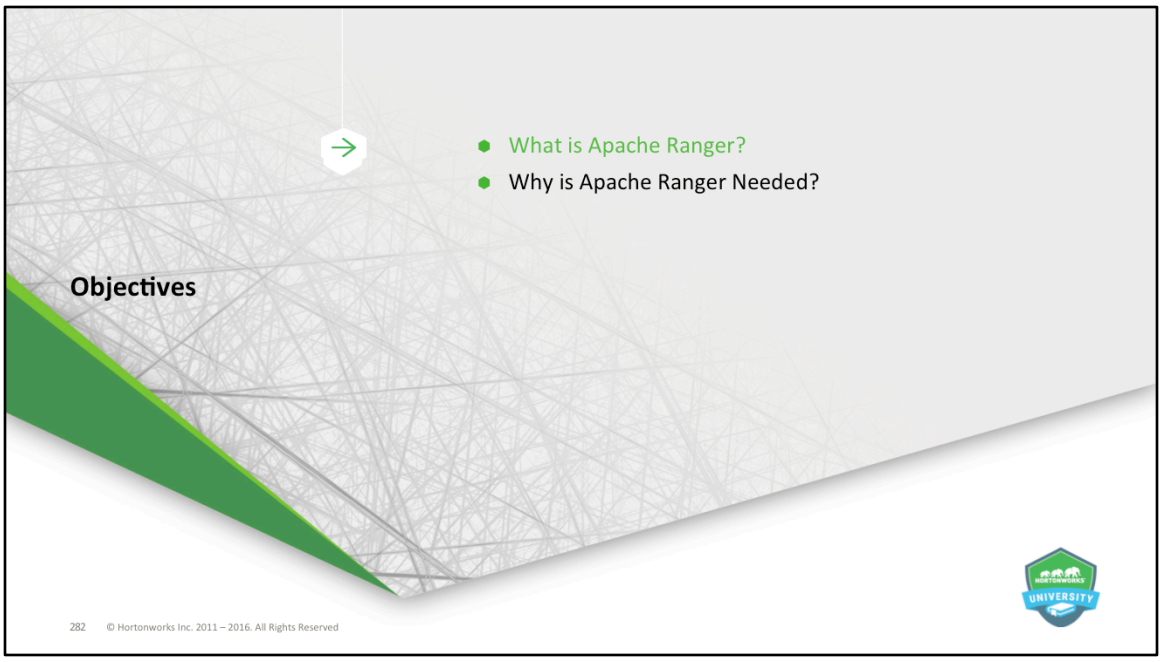
Fine-grained access control provides flexibility in defining policies...

on the folder and file level, via HDFS

on the database, table and column level, via Hive

What is Apache Ranger




Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
282
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Why is Apache Ranger Needed

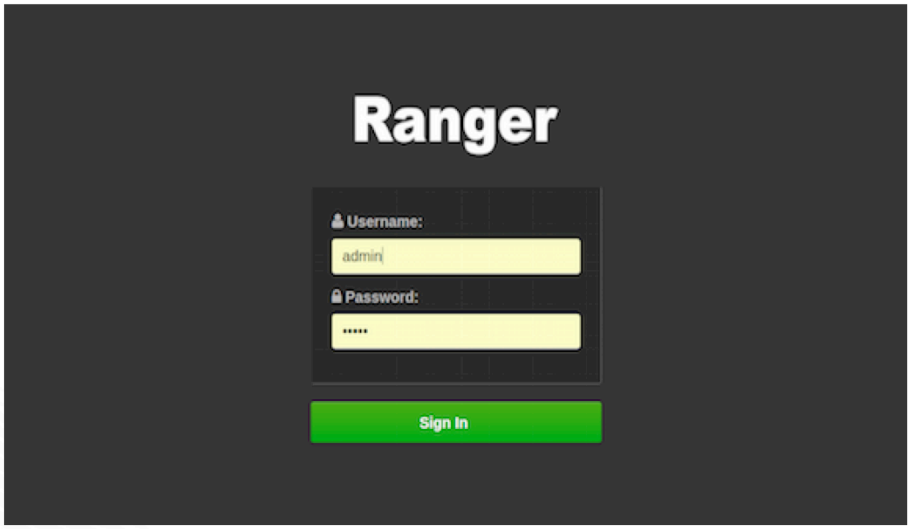
- Offers Centralized Security Framework to Manage Fine Grained Access Control Over Hadoop and Ecosystem Components
- Standardize Authorization Method Across All Hadoop Components.
- Using Administration Console, users can easily manage policies around accessing a resource (file, folder, database, table, column etc) for a particular set of users and/or groups, and enforce the policies within Hadoop.
- Can Enable Audit Tracking and Policy Analytics for Deeper Control of the Hadoop environment
- Provides Ability to Delegate Administration of Certain Data to Other Group Owners, with the Aim of Decentralizing Data Ownership

http://ranger.apache.org

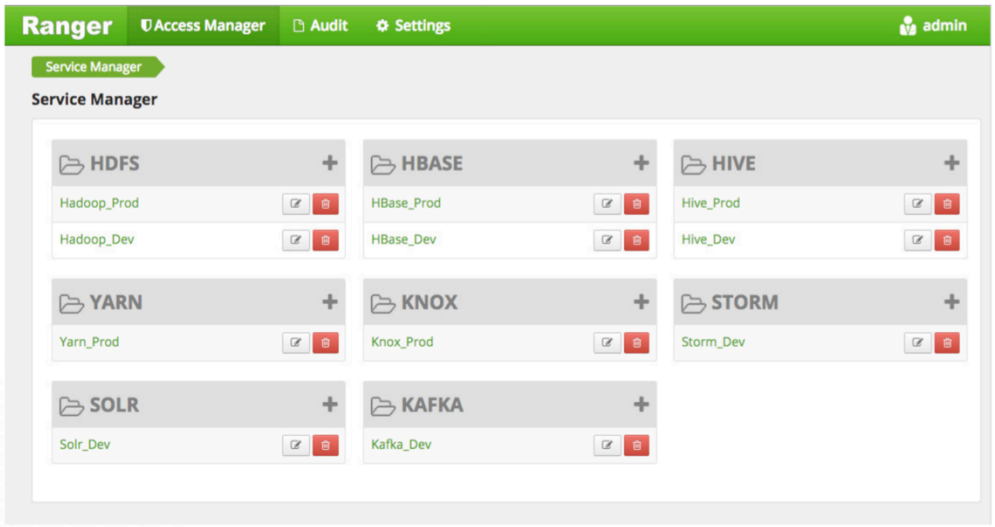
What does Apache Ranger offer for Apache Hadoop and related components?

Apache Ranger offers a centralized security framework to manage fine grained access control over Hadoop and related components (Apache Hive, HBase etc.). Using Ranger administration console, users can easily manage policies around accessing a resource (file, folder, database, table, column etc) for a particular set of users and/or groups, and enforce the policies within Hadoop. They also can enable audit tracking and policy analytics for deeper control of the environment. Ranger' solution also provides ability to delegate administration of certain data to other group owners, with an aim of decentralizing data ownership

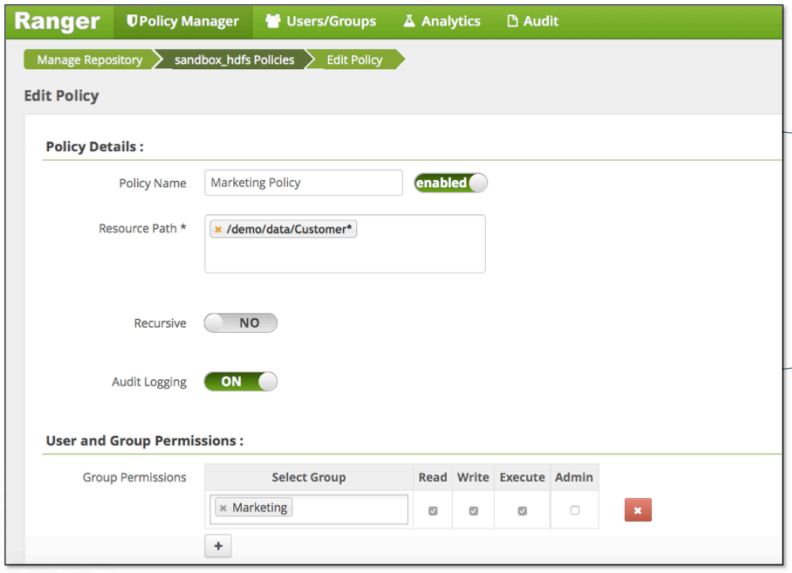
Why is Apache Ranger Needed



Why is Apache Ranger Needed



Why is Apache Ranger Needed

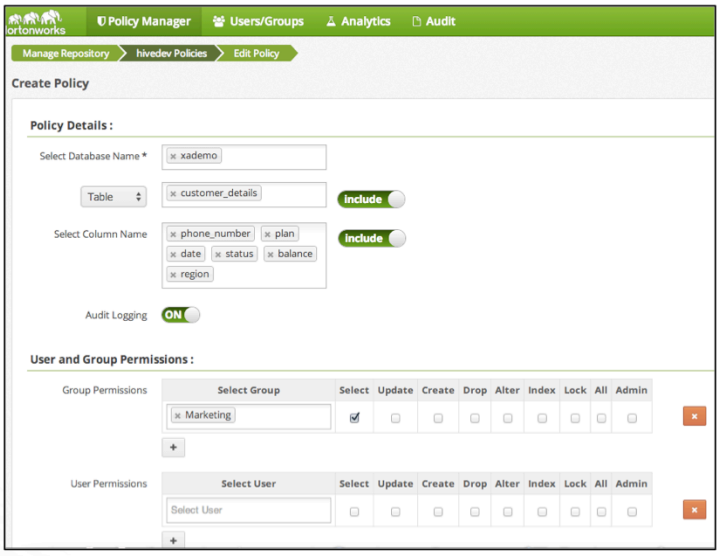


file level access control, flexible definition

Control permissions

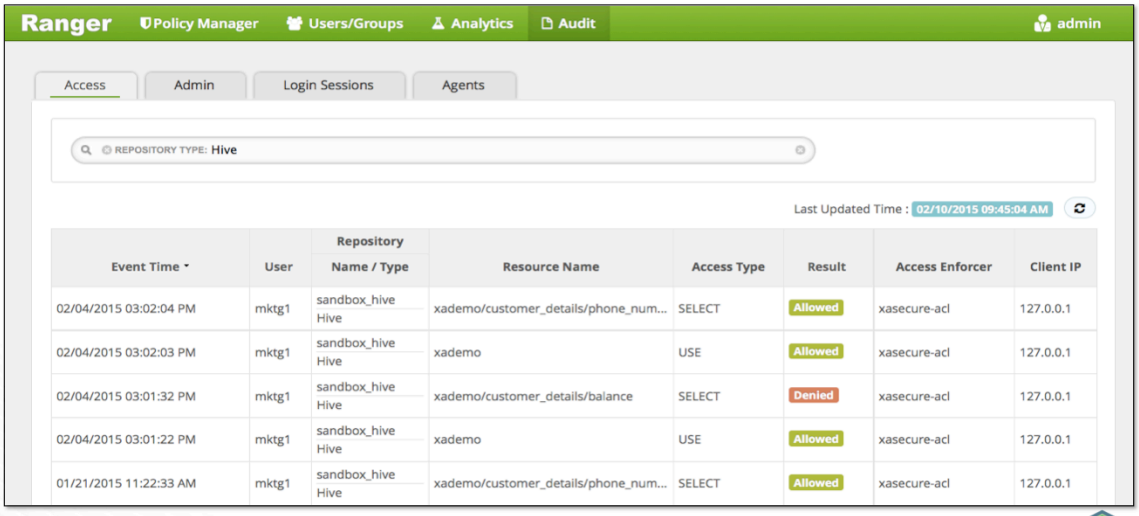
Setup Authorization Policies for HDFS

Why is Apache Ranger Needed

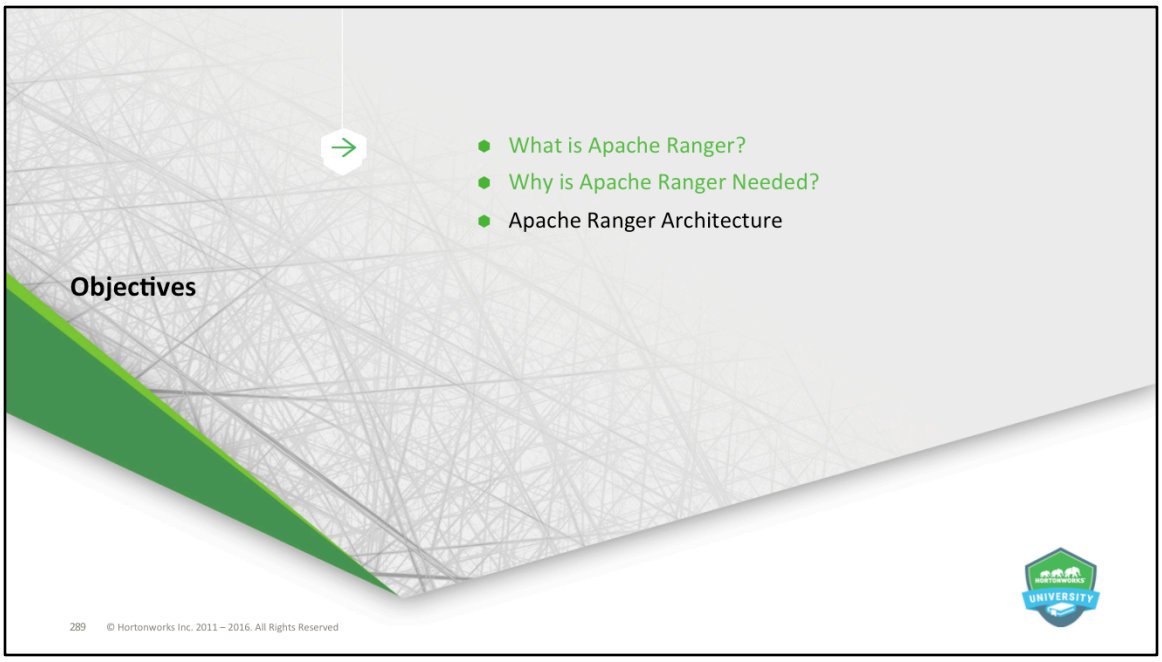


Setup Authorization Policies for Hive

Why is Apache Ranger Needed



Monitor through Auditing


Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
Apache Ranger Architecture
289
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Apache Ranger Architecture

- At the Core Has Centralized Web Application
- Modules Consisting of
  - Policy Administration
  - Audit and Reporting
- Authorized User Are Able to Manage Security Policies using Web Tools or REST API's
- Security Policies Are Enforced within Hadoop using Lightweight Plugins
- Plugins Run as Part of the Service – No Additional OS Level Process to Manage
  - Namenode – HDFS
  - HiveServer2 – Hive
  - Hbase Server – Hbase
  - Nimbus Server – Storm
  - Knox Server - Knox

How does it work over Hadoop and related components

Apache Ranger' solution at the core has a centralized web application, which consists of the policy administration, audit and reporting modules. Authorized users will be able to manage their security policies using the web tool or using REST APIs. These security policies are enforced within Hadoop ecosystem using lightweight Ranger Java plugins, which run as part of the same process as the namenode (HDFS), Hive2Server(Hive), HBase server (Hbase), Nimbus server (Storm) and Knox server (Knox) respectively. Thus there is no additional OS level process to manage.

Is there a single point of failure?

No, Apache Ranger is not a Single Point of Failure. Ranger' plugins run within the same process as the component, e.g. NameNode for HDFS. These agents pull the policy-changes using REST API at a configured regular interval (e.g.: 30 second). The plugin is able to function even if the policy server is temporarily down and will provide the authorization enforcement. Also, the policy manager web application can be hosted on a HA infrastructure. (with multiple apache server, multiple tomcat servers and a standby database server w/o replication setup).

Apache Ranger Architecture

- No Single Point of Failure
- Plugins Run Within the Same Process as the Component Service
- Agent Pull Policy-Changes using REST API at Configured Regular Interval
- Plugin Able to Function
  - When Policy Server is Temporarily Down
  - Will Still Provide Authorization Enforcement
- Policy Manager Web Application can be Host on High Available Infrastructure Utilizing
  - Multiple Apache Servers
  - Multiple Tomcat Servers
  - Standby Database Server Without Replication Setup

http://ranger.apache.org

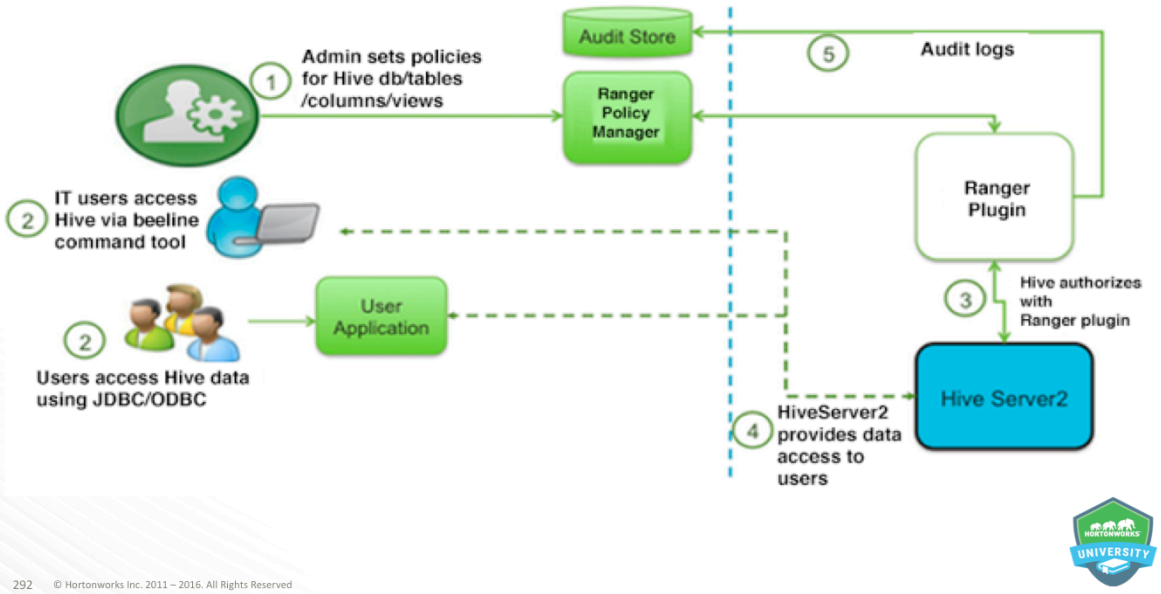
How does it work over Hadoop and related components

Apache Ranger' solution at the core has a centralized web application, which consists of the policy administration, audit and reporting modules. Authorized users will be able to manage their security policies using the web tool or using REST APIs. These security policies are enforced within Hadoop ecosystem using lightweight Ranger Java plugins, which run as part of the same process as the namenode (HDFS), Hive2Server(Hive), HBase server (Hbase), Nimbus server (Storm) and Knox server (Knox) respectively. Thus there is no additional OS level process to manage.

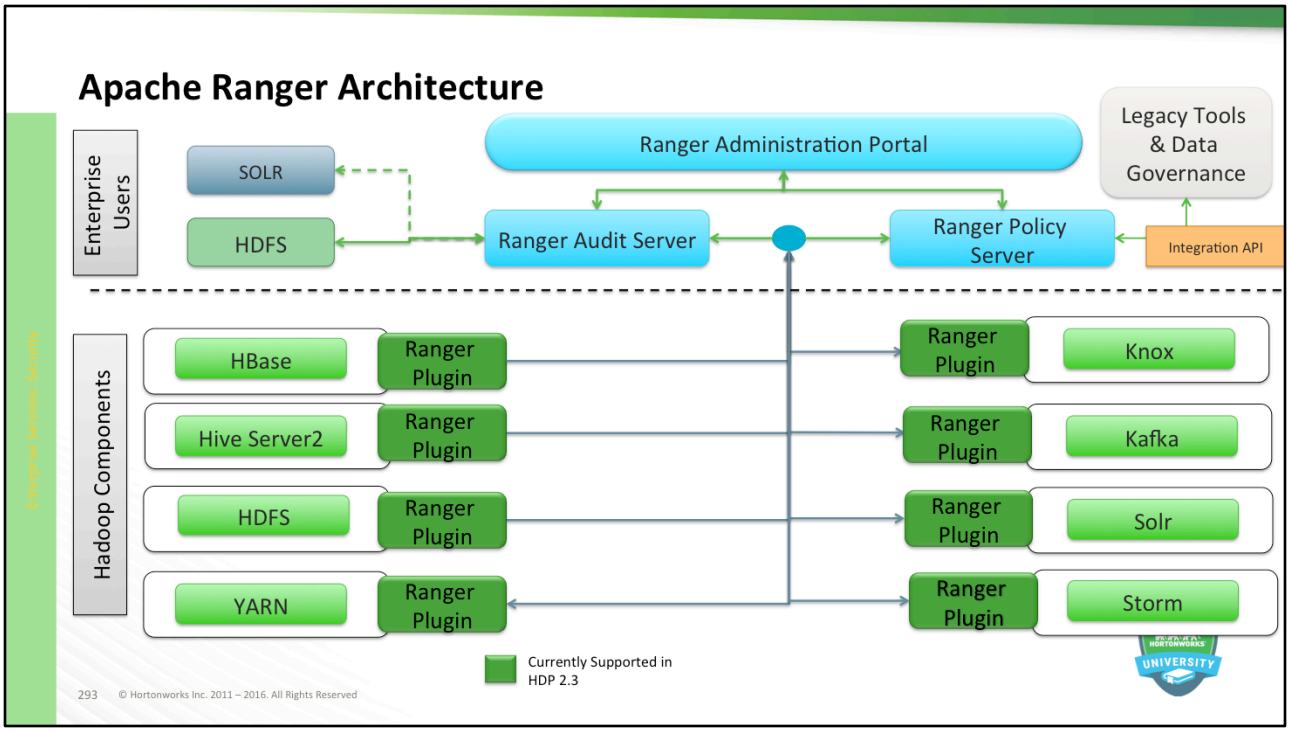
Is there a single point of failure?

No, Apache Ranger is not a Single Point of Failure. Ranger' plugins run within the same process as the component, e.g. NameNode for HDFS. These agents pull the policy-changes using REST API at a configured regular interval (e.g.: 30 second). The plugin is able to function even if the policy server is temporarily down and will provide the authorization enforcement. Also, the policy manager web application can

Apache Ranger Architecture



http://ranger.apache.org
Lets take a min to step through how Hive plugin would work from end user perspective
(then talk through each step)

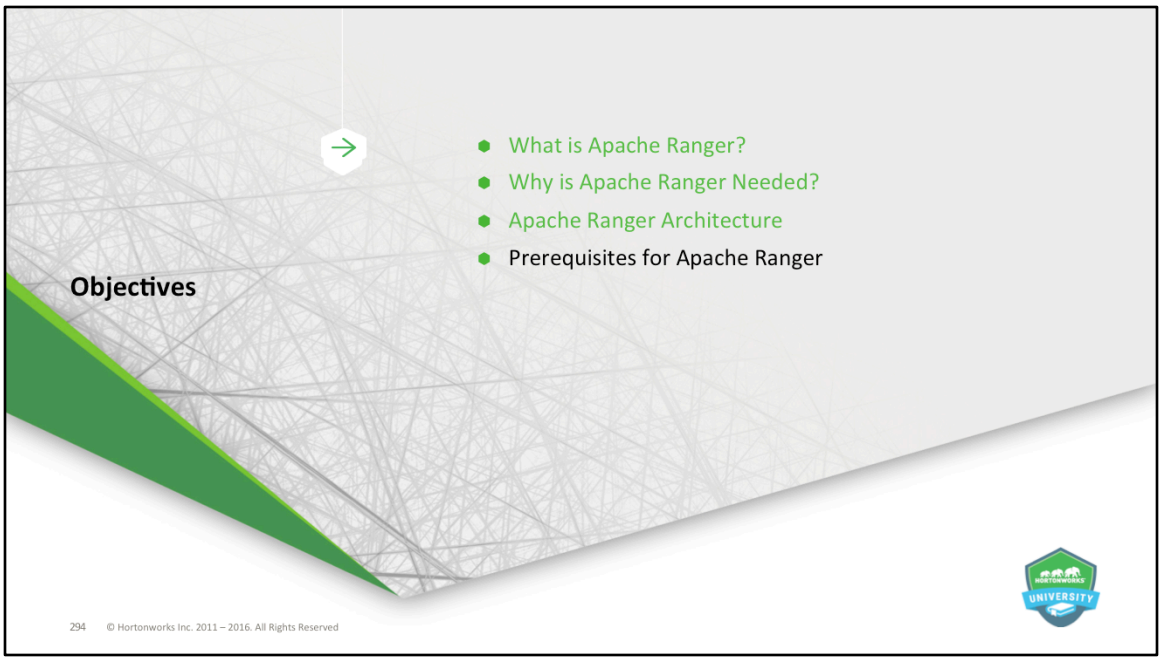


Architectuall overview of how it works

Similar to Hive plugin in earlier slide, plugins for other Hadoop components would work the same way

Plugins are embedded with the component so if Admin goes down, policies are still in effect (may not have latest policies downloaded that's all)

No OS administrative overhead (i.e. no new daemons spawned) for each plugin


Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
Apache Ranger Architecture
Prerequisites for Apache Ranger
294
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Prerequisites for Apache Ranger

- Recommended that Audits Stored in Both HDFS and Solr
- Install Apache Solr
- Setup either OS/AD sync (e.g. SSSD) or Hadoop Group Mapping for AD
  - To Ensure LDAP/AD Group Level Authorization is Enforced in Hadoop
- Ranger Installation Will Create Two New Users
  - rangeradmin
  - rangerlogger
- Ranger Installation Will Create Two New Databases
  - ranger
  - ranger_audit
- Select and Configure Database Instance – MySQL, PostgreSQL, Oracle

Before you install Ranger, make sure your cluster meets the following requirements:

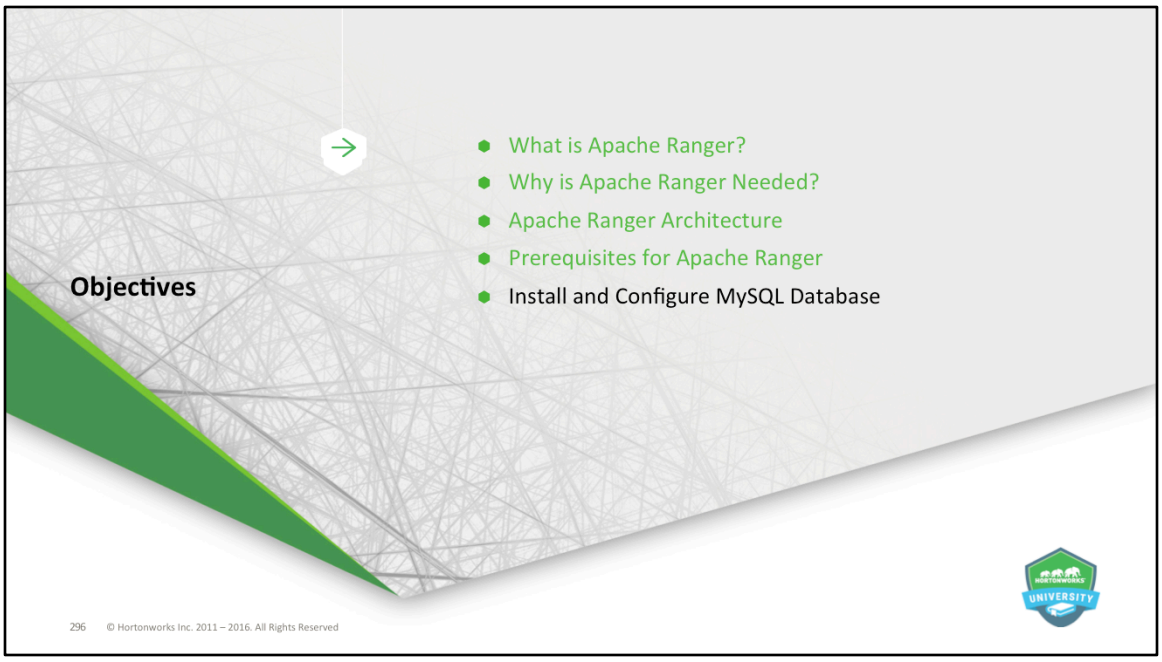
It is recommended that you store audits in both HDFS and Solr, so you should install Apache Solr.

To ensure that LDAP/AD group level authorization is enforced in Hadoop, you should set up Hadoop group mapping for LDAP.

A MySQL, Oracle, PostgreSQL, MS SQL, or SQL Anywhere database instance must be running and available to be used by Ranger.

The Ranger installation will create two new users (default names: rangeradmin and rangerlogger) and two new databases (default names: ranger and ranger_audit).

Configuration of the database instance for Ranger is described in the following sections for some of the databases supported by Ranger.


Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
Apache Ranger Architecture
Prerequisites for Apache Ranger
Install and Configure MySQL Database
296
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Prerequisites for Apache Ranger

- Install MySQL Database Instance

```
# yum install mysql-server
# chkconfig mysqld on
# service mysqld start
# mysql-secure-installation
```

- Add Grant to Root User or Database Administrator Account

```
CREATE USER 'root'@'%';
GRANT ALL PRIVILEGES ON *.* to 'root'@'%' WITH GRANT OPTION;
SET PASSWORD FOR 'root'@'%' = PASSWORD('BadPass#1');
SET PASSWORD = PASSWORD('BadPass#1');
FLUSH PRIVILEGES;
exit;
```

The MySQL database administrator should be used to create the Ranger databases.

The following series of commands could be used to create the rangerdba user with password rangerdba.

Log in as the root user, then use the following commands to create the rangerdba user and grant it adequate privileges.

Use the exit command to exit MySQL.

You should now be able to reconnect to the database as rangerdba using the following command:

mysql –u root –p <password>

Prerequisites for Apache Ranger

- Install MySQL Database Instance - Continued
- Test Connection - Prompts for Password

  # mysql -u root -p
- Install mysql-connector-java if needed

  yum install mysql-connector-java
- Setup Ambari for MySQL

  # ambari-server setup --jdbc-db=mysql --jdbc-driver=/usr/share/java/mysql-connector-java.jar
- Add Service – Ranger Install will Create the Databases “ranger” & “ranger_audit”

The MySQL database administrator should be used to create the Ranger databases.

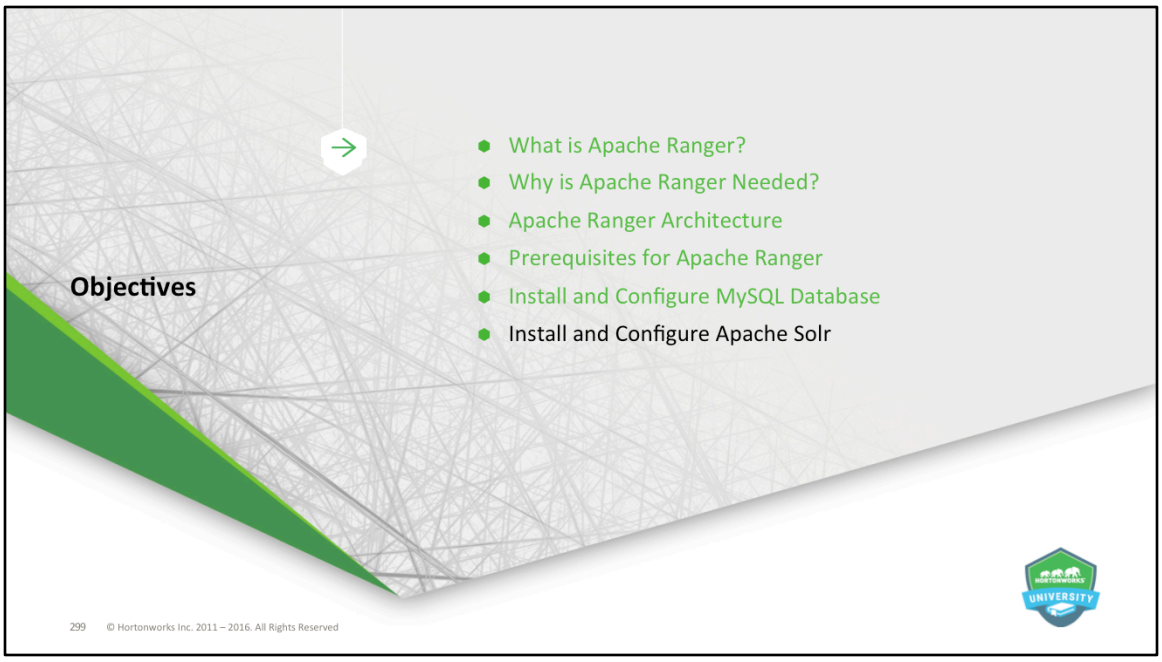
The following series of commands could be used to create the rangerdba user with password rangerdba.

Log in as the root user, then use the following commands to create the rangerdba user and grant it adequate privileges.

Use the exit command to exit MySQL.

You should now be able to reconnect to the database as rangerdba using the following command:

mysql –u root –p <password>


Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
Apache Ranger Architecture
Prerequisites for Apache Ranger
Install and Configure MySQL Database
Install and Configure Apache Solr
299
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Install Apache Solr with Ambari

- Apache Solr a Open-Source Enterprise Search Platform Utilized by Ranger
- Stores Audit Logs
- Provides Search Capability Through Ranger Admin UI
- Solr Must be Installed/Configured Prior to Installing Ranger
- Recommended Ranger Audits Written Both Solr and HDFS
- HDFS is a long-term destination for Audit Logs
- SolrCloud is Recommended Configuration
- Scalable Architecture – Can Run as Single Node or Multi-Node Cluster
- Includes Features Such as Replication and Sharding - Useful for High Availability (HA) and Scalability

Apache Solr is an open-source enterprise search platform. Apache Ranger can use Apache Solr to store audit logs, and Solr can also to provide a search capability of the audit logs through the Ranger Admin UI.

Important

Solr must be installed and configured before installing RangerAdmin or any of the Ranger component plugins.

It is recommended that Ranger audits be written to both Solr and HDFS. Audits to Solr are primarily used to enable search queries from the Ranger Admin UI. HDFS is a long-term destination for audits -- audits stored in HDFS can be exported to any SIEM system, or to another audit store.

Configuration Options

Solr Standalone -- Solr Standalone is only recommended for testing and evaluation.

Install Apache Solr with Ambari

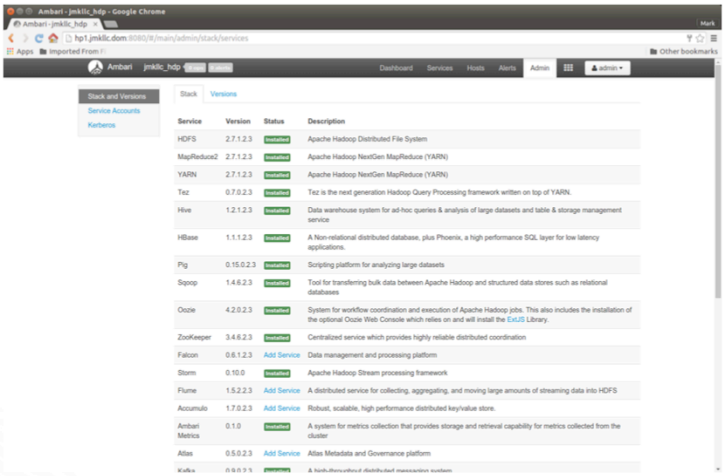
- Ambari service for Apache Solr Currently Not Included in Ambari 2.2.x
- Demo Service Available via GitHub At https://github.com/abajwa-hw/solr-stack:
  - git clone https://github.com/abajwa-hw/solr-stack.git /var/lib/ambari-server/resources/stacks/HDP/2.4/services/SOLR
- Restart Ambari Server

To deploy the Solr stack, run below

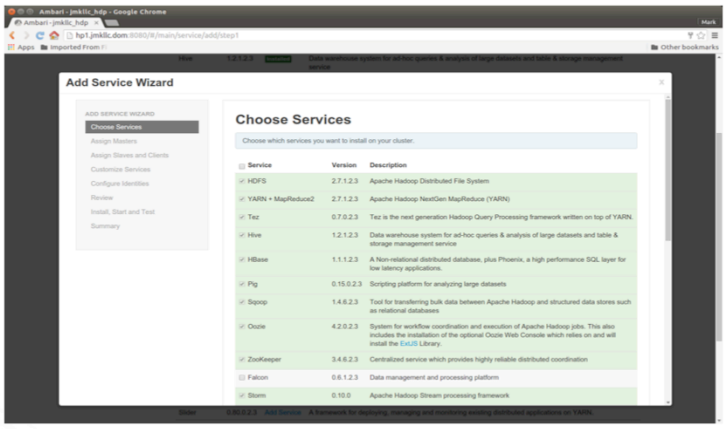
VERSION=`hdp-select status hadoop-client | sed 's/hadoop-client - \([0-9]\.[0-9]\).*/\1/'`

sudo git clone https://github.com/abajwa-hw/solr-stack.git /var/lib/ambari-server/resources/stacks/HDP/$VERSION/services/SOLR

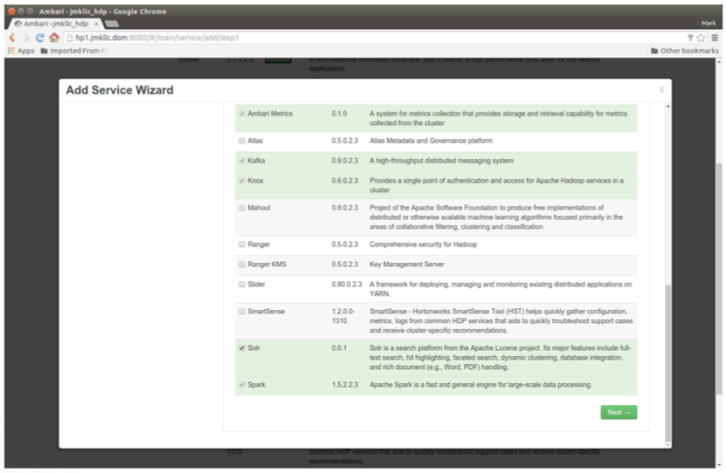
Install Apache Solr with Ambari



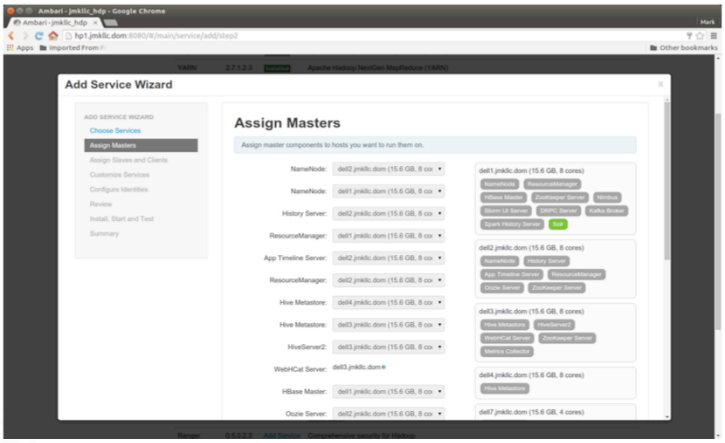
Install Apache Solr with Ambari



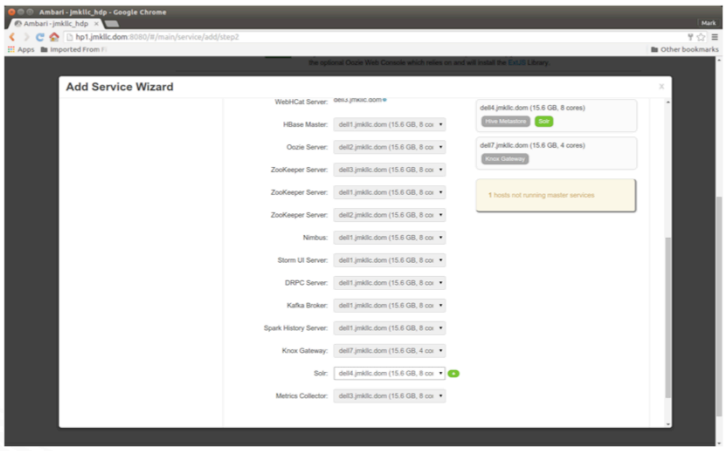
Install Apache Solr with Ambari



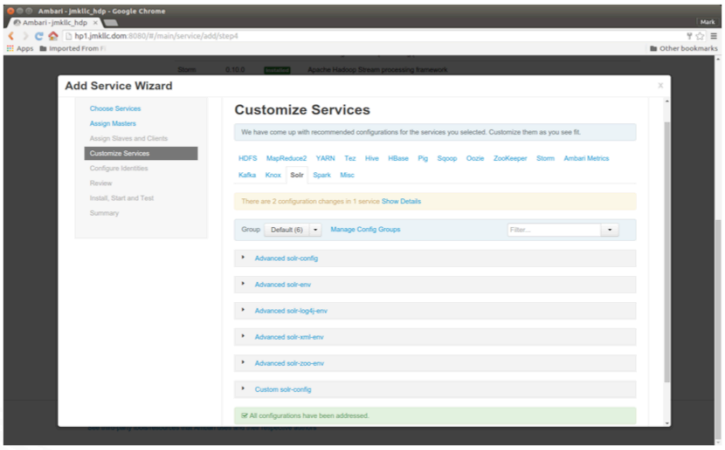
Install Apache Solr with Ambari



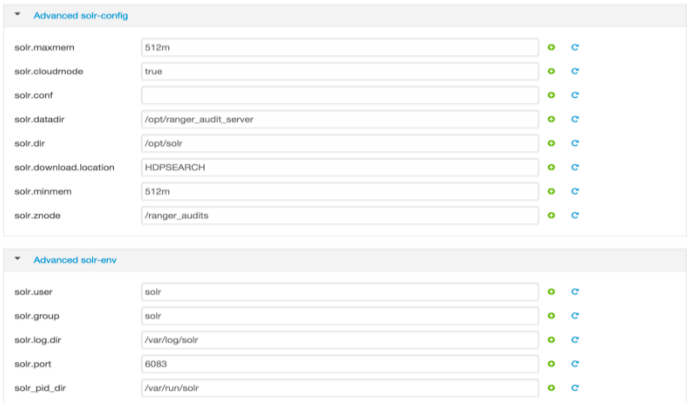
Install Apache Solr with Ambari



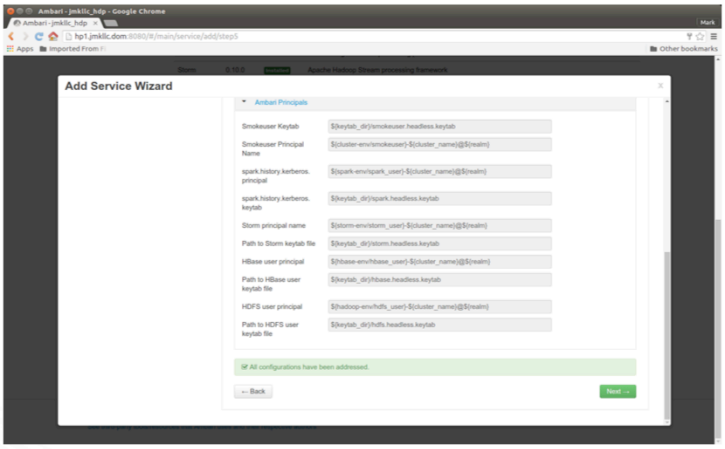
Install Apache Solr with Ambari



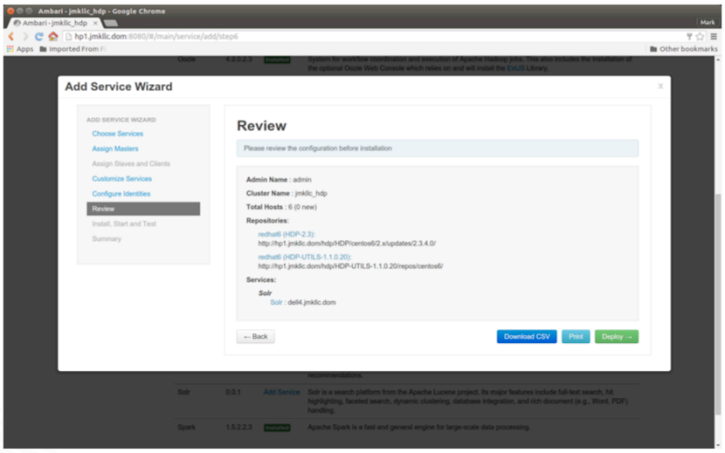
Install Apache Solr with Ambari



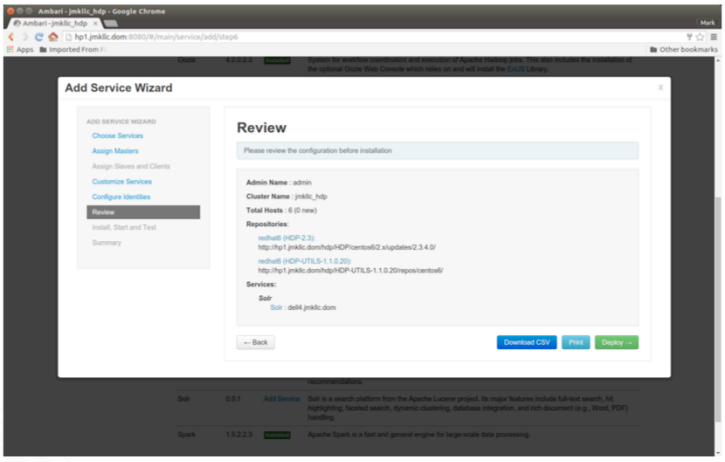
Install Apache Solr with Ambari



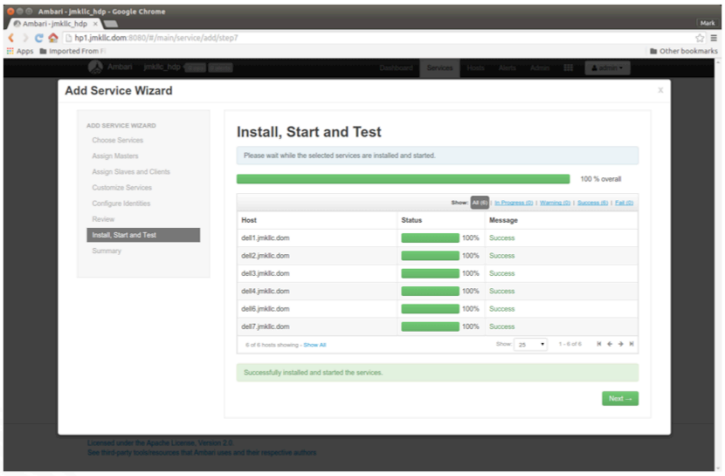
Install Apache Solr with Ambari



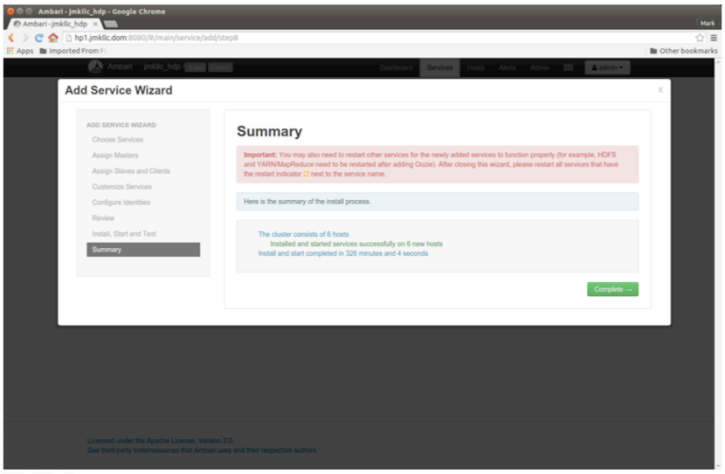
Install Apache Solr with Ambari



Install Apache Solr with Ambari



Install Apache Solr with Ambari



Setup Apache Solr for Apache Ranger Audits

- Download the solr_for_audit_setup_v3 file to the /usr/local/ directory:

  wget https://issues.apache.org/jira/secure/attachment/12761323/solr_for_audit_setup_v3.tgz -O /usr/local/solr_for_audit_setup_v3.tgz
- Switch to the /usr/local/ directory and extract the solr_for_audit_setup_v3 file

  cd /usr/local

  tar xvf solr_for_audit_setup_v3.tgz
- Extract into a /usr/local/solr_for_audit_setup_v3 directory
- Switch to the /usr/local/solr_for_audit_setup_v3 directory

  cd /usr/local/solr_for_audit_setup
- Open the install.properties file in the vi text editor.

  vi install.properties
- Set property values, save the changes to the install.properties file.

Use the following procedure to configure SolrCloud.

Download the solr_for_audit_setup_v3 file to the /usr/local/ directory:

wget https://issues.apache.org/jira/secure/attachment/12761323/solr_for_audit_setup_v3.tgz -O /usr/local/solr_for_audit_setup_v3.tgz

Use the following commands to switch to the /usr/local/ directory and extract the solr_for_audit_setup_v3 file.

cd /usr/local

tar xvf solr_for_audit_setup_v3.tgz

The contents of the .tgz file will be extracted into a /usr/local/solr_for_audit_setup_v3 directory.

Use the following command to switch to the /usr/local/solr_for_audit_setup_v3

Setup Apache Solr for Apache Ranger Audits

- JAVA_HOME=$JAVA_HOME
- SOLR_USER=solr
- SOLR_INSTALL=false
- SOLR_INSTALL_FOLDER=/opt/lucidworks-hdpsearch/solr
- **SOLR_RANGER_HOME=/opt/ranger_audit_server**
- **SOLR_RANGER_PORT=6083**
- **SOLR_DEPLOYMENT=solrcloud**
- **SOLR_ZK=localhost:2181/ranger_audits**
- SOLR_HOST_URL=http://$host:\${SOLR_RANGER_PORT}
- SOLR_SHARDS=1
- **SOLR_REPLICATION=2**
- SOLR_LOG_FOLDER=/var/log/solr/ranger_audits
- SOLR_MAX_MEM=1g

Use the following procedure to configure SolrCloud.

Download the solr_for_audit_setup_v3 file to the /usr/local/ directory:

wget https://issues.apache.org/jira/secure/attachment/12761323/solr_for_audit_setup_v3.tgz -O /usr/local/solr_for_audit_setup_v3.tgz

Use the following commands to switch to the /usr/local/ directory and extract the solr_for_audit_setup_v3 file.

cd /usr/local

tar xvf solr_for_audit_setup_v3.tgz

The contents of the .tgz file will be extracted into a /usr/local/solr_for_audit_setup_v3 directory.

Use the following command to switch to the /usr/local/solr_for_audit_setup_v3

Setup Apache Solr for Apache Ranger Audits

- Use the following command to run the set up script.

  ./setup.sh

- Run the following command only once from any node to add the Ranger Audit configuration (including schema.xml) to ZooKeeper:

  /opt/lucidworks-hdpsearch/solr/ranger_audit_server/scripts/add_ranger_audits_conf_to_zk.sh

- Run the following command only once from any node to create the Ranger Audit collection:

  /opt/lucidworks-hdpsearch/solr/ranger_audit_server/scripts/create_ranger_audits_collection.sh

- You can use a web browser to open the Solr Admin Console at the following address:

  http:<solr_host>:6083/solr

Use the following procedure to configure SolrCloud.

Download the solr_for_audit_setup_v3 file to the /usr/local/ directory:

wget https://issues.apache.org/jira/secure/attachment/12761323/solr_for_audit_setup_v3.tgz -O /usr/local/solr_for_audit_setup_v3.tgz

Use the following commands to switch to the /usr/local/ directory and extract the solr_for_audit_setup_v3 file.

cd /usr/local

tar xvf solr_for_audit_setup_v3.tgz

The contents of the .tgz file will be extracted into a /usr/local/solr_for_audit_setup_v3 directory.

Use the following command to switch to the /usr/local/solr_for_audit_setup_v3

Apache SolrCloud

- SolrCloud provides support for
  - Distributing Index Process and Queries Automatically
  - ZooKeeper provides failover and load balancing
  - Every shard can have multiple replicas for robustness
- In Cloud Mode, there are Leaders and Replicas
  - Leaders are automatically elected
  - Initially on a first-come-first-served basis
- Leader goes down,
  - one of its replicas is automatically elected as the new leader.
- As each node is started, it's assigned to the shard with the fewest replicas

SolrCloud provides support for distributing both the index process and the queries automatically, and ZooKeeper provides failover and load balancing. Additionally, every shard can also have multiple replicas for additional robustness.

In Solr Cloud Mode, there are leaders and replicas. Leaders are automatically elected, initially on a first-come-first-served basis, and then based on the Zookeeper process described at http://zookeeper.apache.org/doc/trunk/recipes.html#sc_leaderElection..

If a leader goes down, one of its replicas is automatically elected as the new leader. As each node is started, it's assigned to the shard with the fewest replicas. When there's a tie, it's assigned to the shard with the lowest shard ID.

When a document is sent to a machine for indexing, the system first determines if the machine is a replica or a leader.

If the machine is a replica, the document is forwarded to the leader for processing.

If the machine is a leader, SolrCloud determines which shard the document should go to, forwards the document the leader for that shard, indexes the document for this

Apache SolrCloud

- When document is sent to node for indexing…
  - System determines if its a Replica or a Leader
- If Replica:
  - Document forwarded to Leader for processing
- If Leader:
  - Determines which shard to forward document to
  - Forwards document to Leader for that shard
  - Indexes document for this shard
  - Forwards index notation to itself and any Replicas

SolrCloud provides support for distributing both the index process and the queries automatically, and ZooKeeper provides failover and load balancing. Additionally, every shard can also have multiple replicas for additional robustness.

In Solr Cloud Mode, there are leaders and replicas. Leaders are automatically elected, initially on a first-come-first-served basis, and then based on the Zookeeper process described at http://zookeeper.apache.org/doc/trunk/recipes.html#sc_leaderElection..

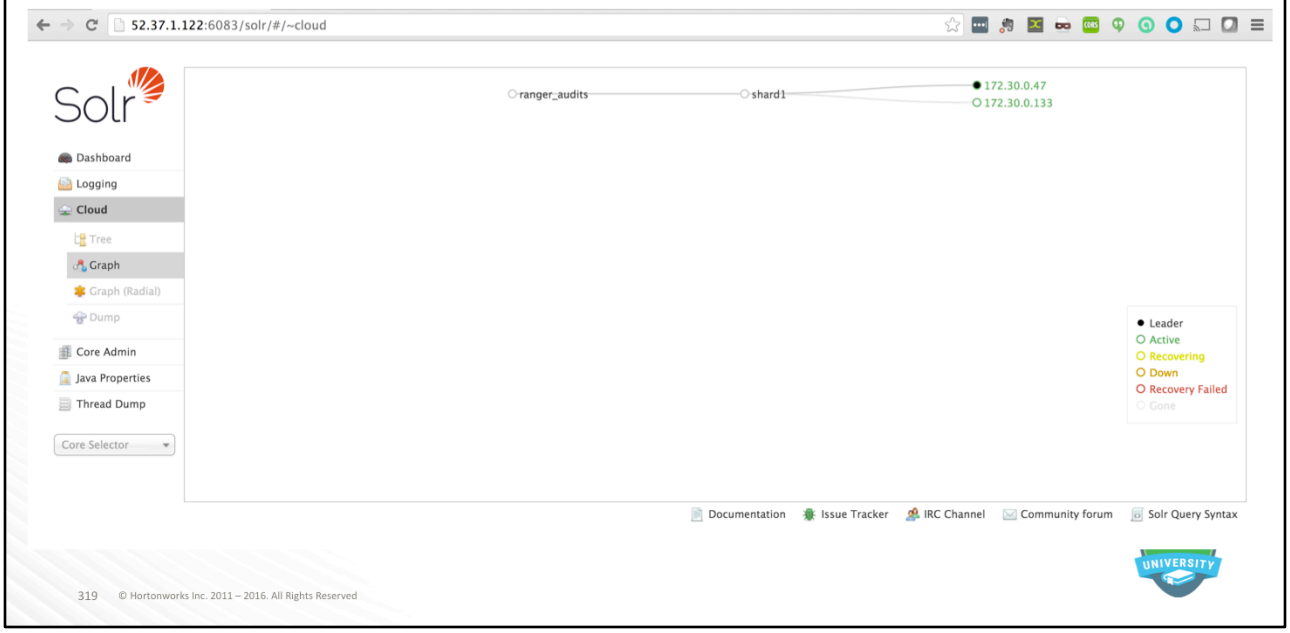
If a leader goes down, one of its replicas is automatically elected as the new leader. As each node is started, it's assigned to the shard with the fewest replicas. When there's a tie, it's assigned to the shard with the lowest shard ID.

When a document is sent to a machine for indexing, the system first determines if the machine is a replica or a leader.

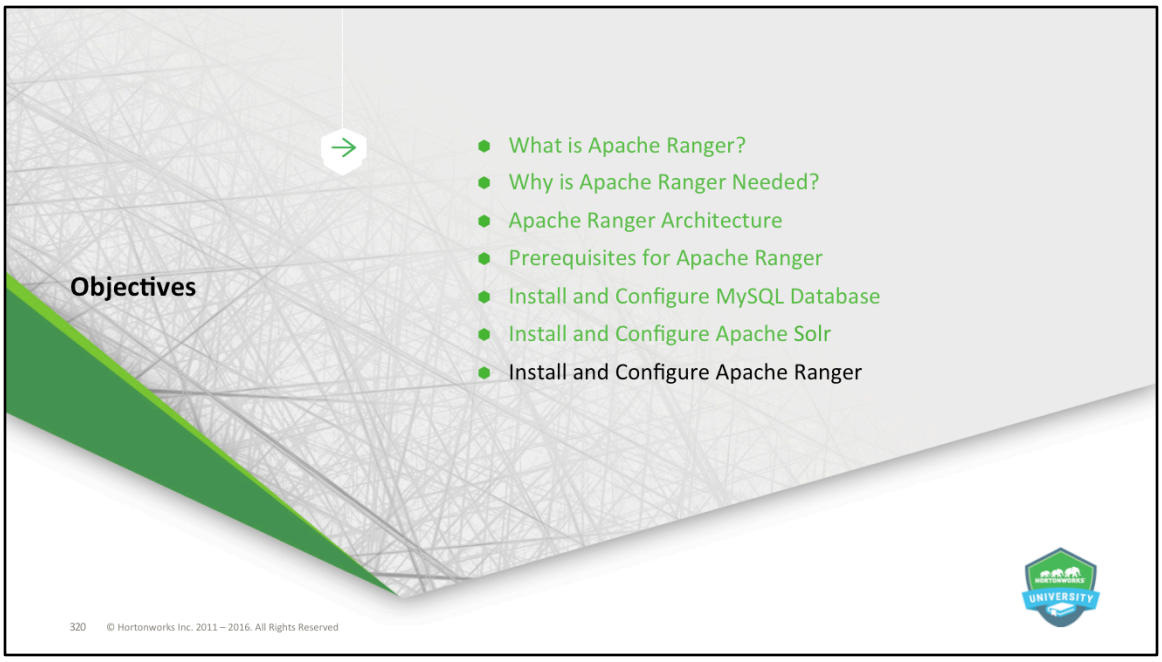
If the machine is a replica, the document is forwarded to the leader for processing.

If the machine is a leader, SolrCloud determines which shard the document should go to, forwards the document the leader for that shard, indexes the document for this

Apache Solr Cloud Leader



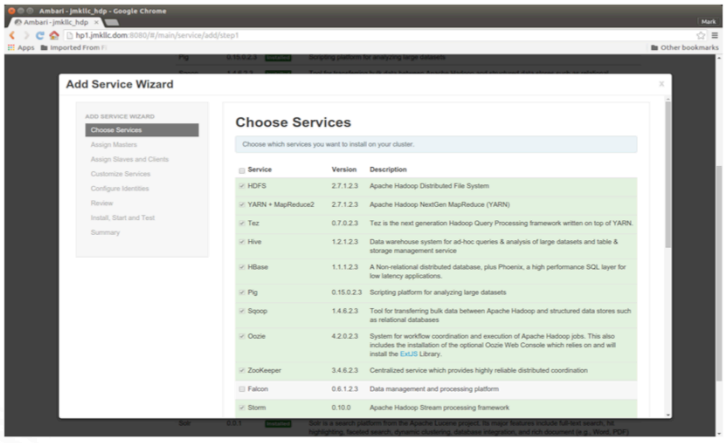
Click the Cloud > Graph tab to find the leader host (172.30.0.242 in below example)


Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
Apache Ranger Architecture
Prerequisites for Apache Ranger
Install and Configure MySQL Database
Install and Configure Apache Solr
Install and Configure Apache Ranger
320
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

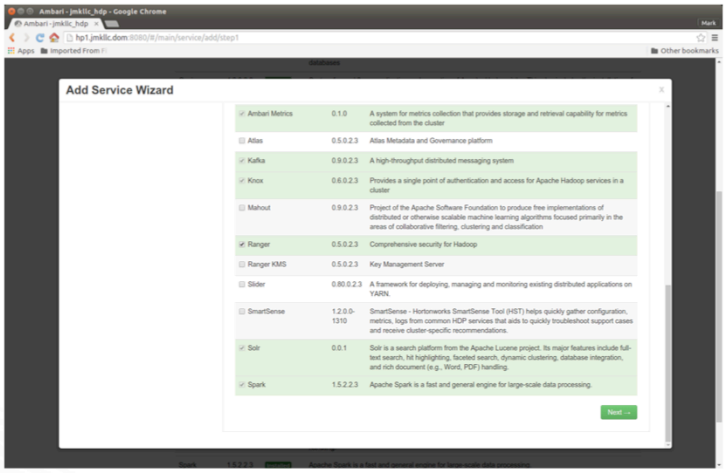
Install Apache Ranger with Ambari

- Install Ranger Via Add Service Wizard Under “Admin > Stack and Versions”
- Add Service Wizard – Select Ranger
- Choose Services – Confirm Ranger is Selected
- Assign Master – Choose Hosts to Run Ranger
- Customize Services – See Following Slides
- Review Configuration – Deploy
- Monitor Deployment – Install, Start and Test
- Review Summary
- Installation Complete

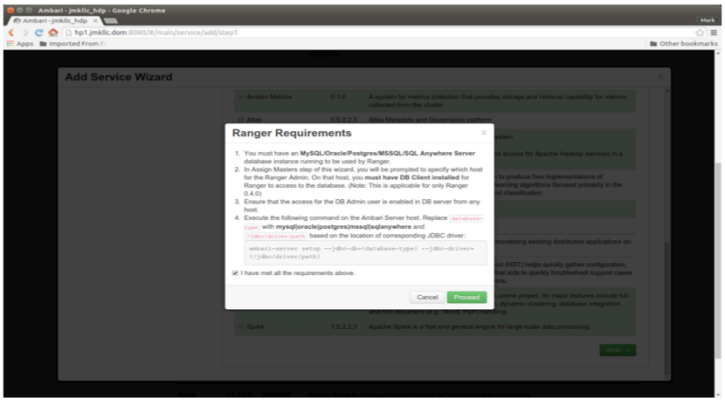
Install Apache Ranger with Ambari



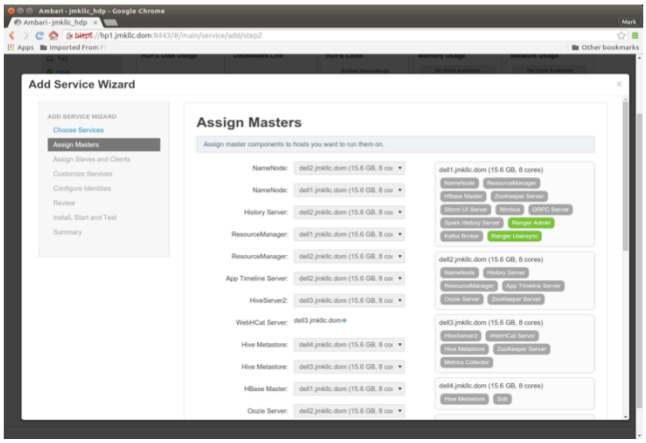
Install Apache Ranger with Ambari



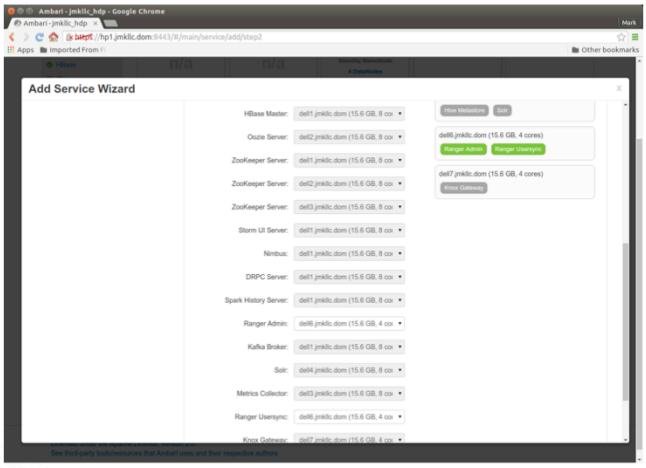
Install Apache Ranger with Ambari



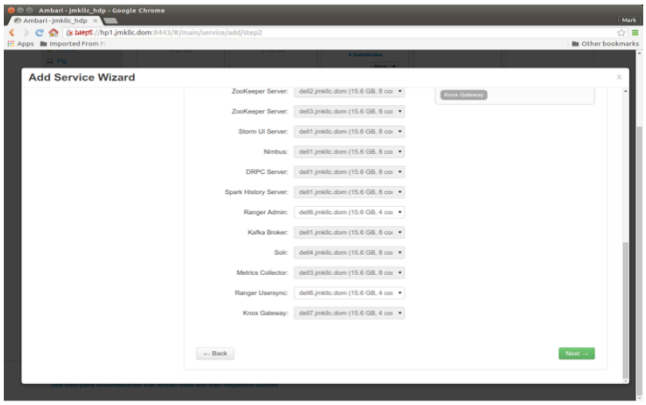
Install Apache Ranger with Ambari



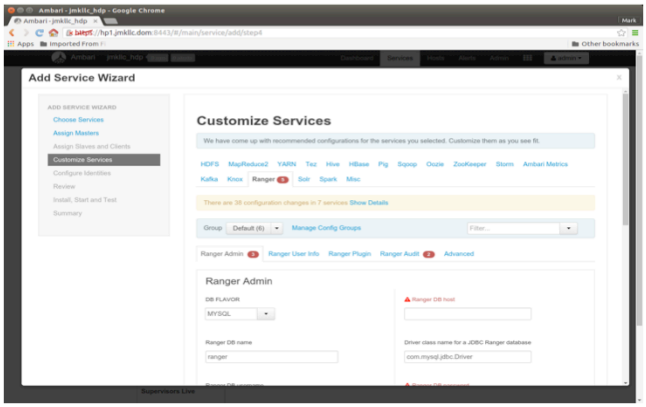
Install Apache Ranger with Ambari



Install Apache Ranger with Ambari



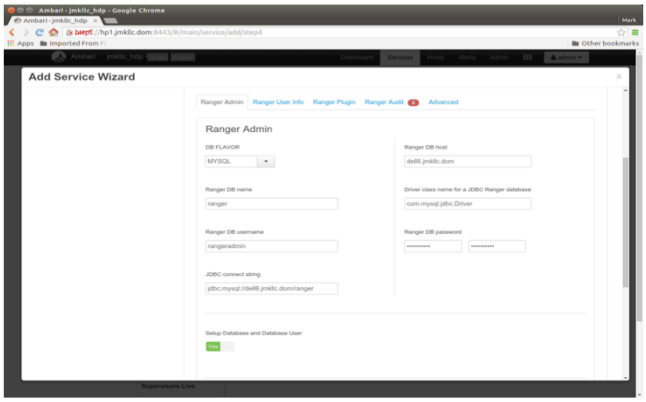
Install Apache Ranger with Ambari



Ranger DB Host = FQDN of host where Mysql is running (e.g. ip-172-30-0-242.us-west-2.compute.internal)

Enter passwords

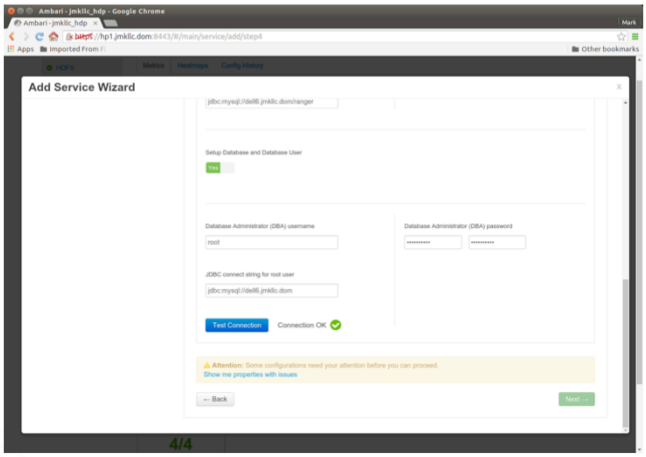
Install Apache Ranger with Ambari



Ranger DB Host = FQDN of host where Mysql is running (e.g. ip-172-30-0-242.us-west-2.compute.internal)

Enter passwords

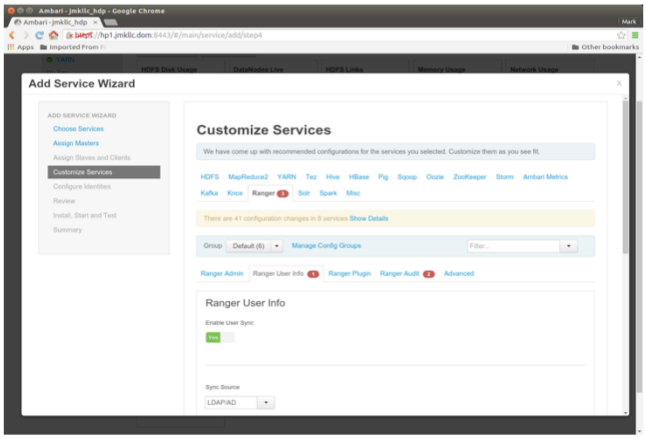
Install Apache Ranger with Ambari



Enter Passwords - BadPass#1

Test Connection

Install Apache Ranger with Ambari

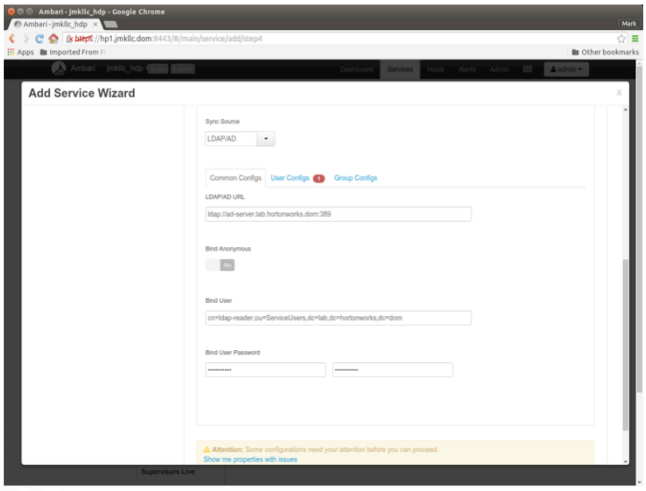


'Sync Source' = AD/LDAP

Common configs subtab

Enter password

Install Apache Ranger with Ambari

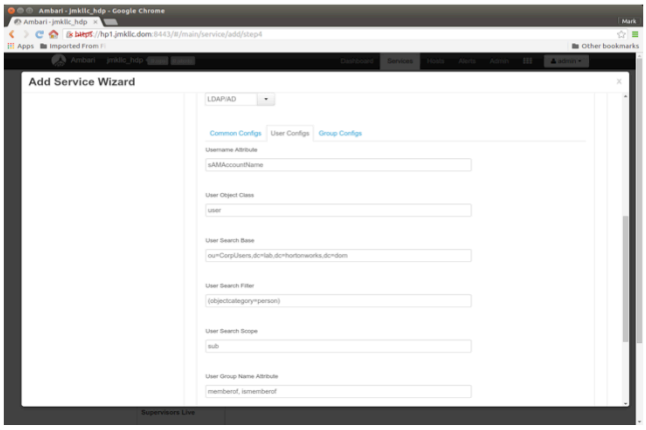


'Sync Source' = AD/LDAP

Common configs subtab

Enter password

Install Apache Ranger with Ambari

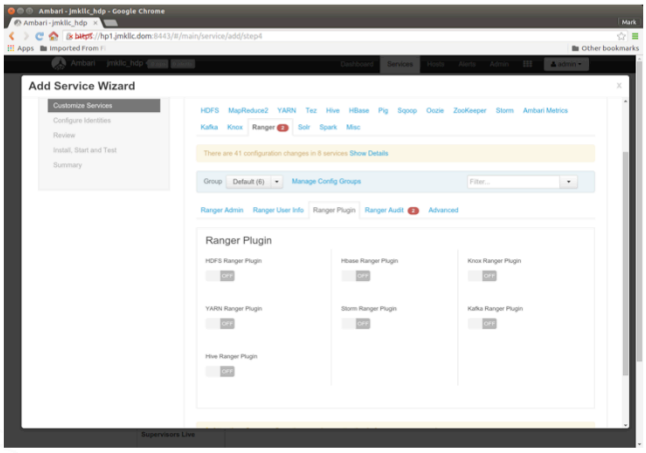


User configs subtab

User Search Base = ou=CorpUsers,dc=lab,dc=hortonworks,dc=net

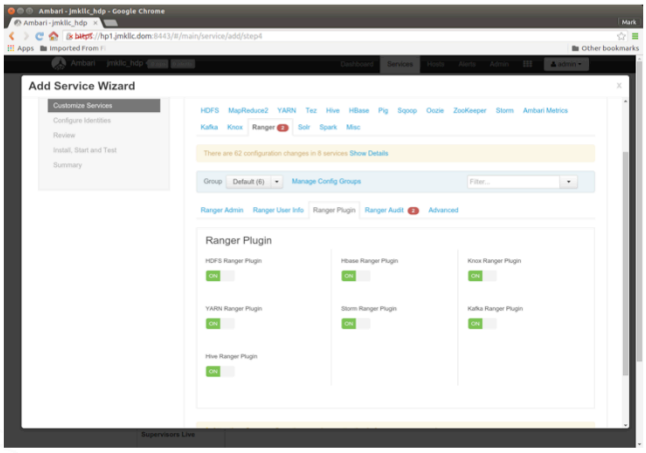
User Search Filter = (objectcategory=person)

Install Apache Ranger with Ambari



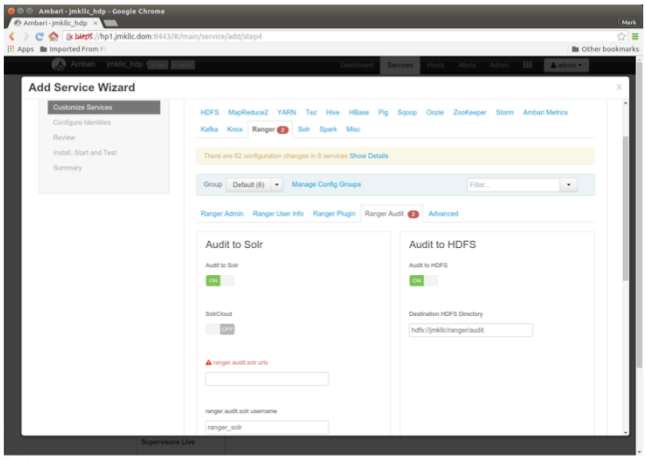
Plugins disabled

Install Apache Ranger with Ambari



Enable all plugins

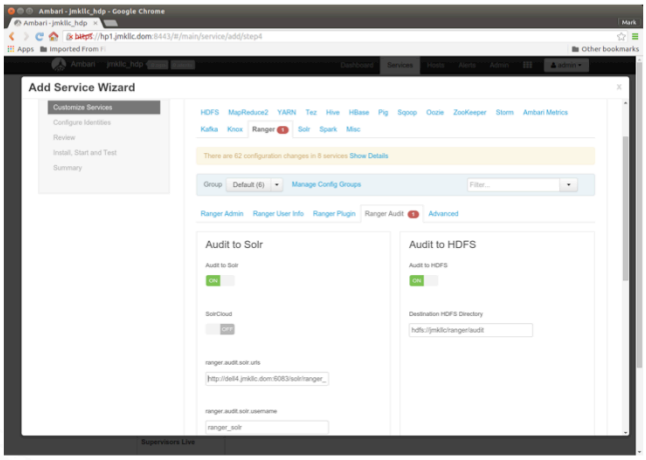
Install Apache Ranger with Ambari



SolrCloud = ON

enter password

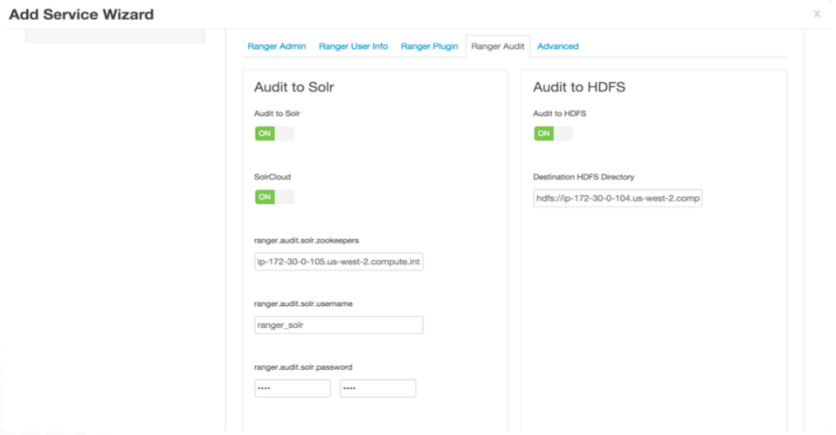
Install Apache Ranger with Ambari



SolrCloud = ON

enter password

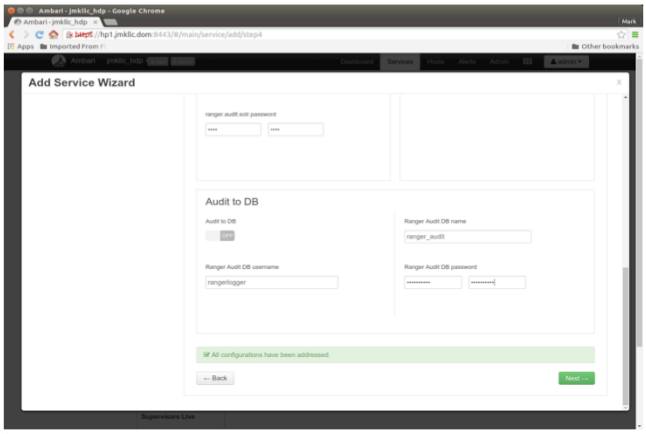
Install Apache Ranger with Ambari



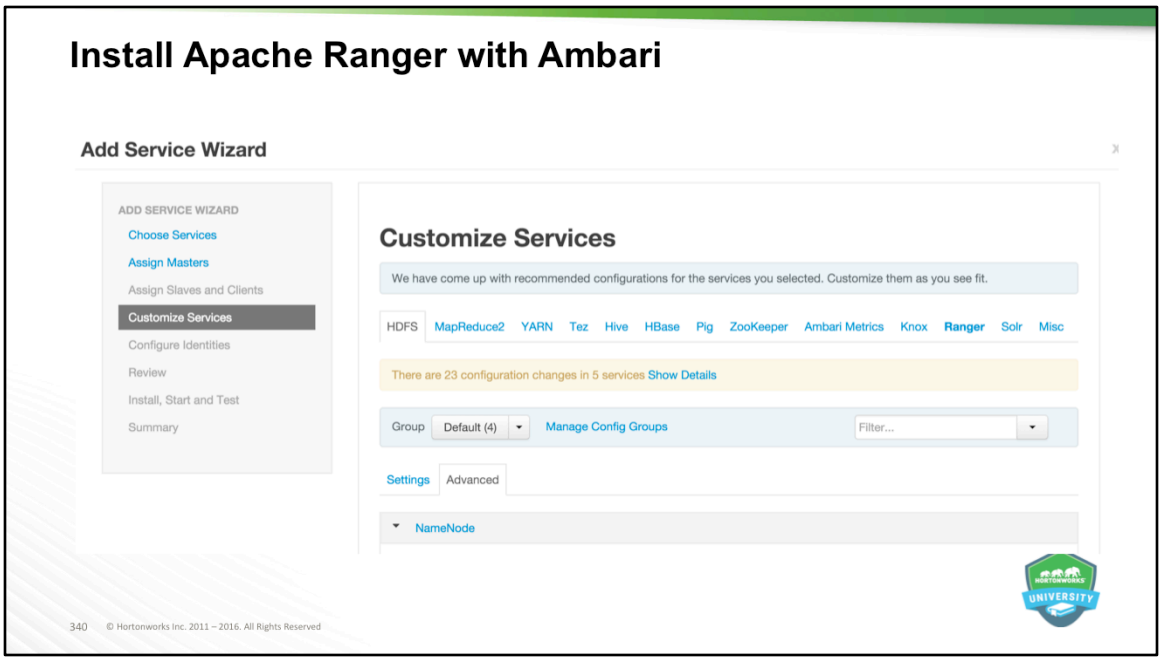
SolrCloud = ON

enter password

Install Apache Ranger with Ambari

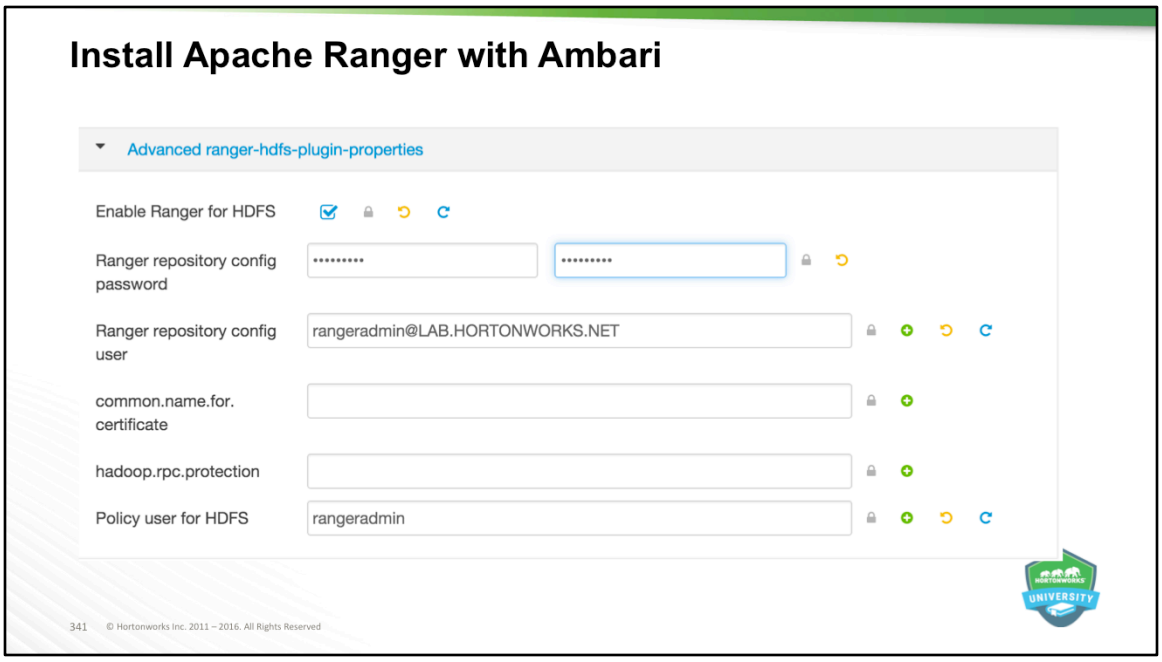


Install Apache Ranger with Ambari

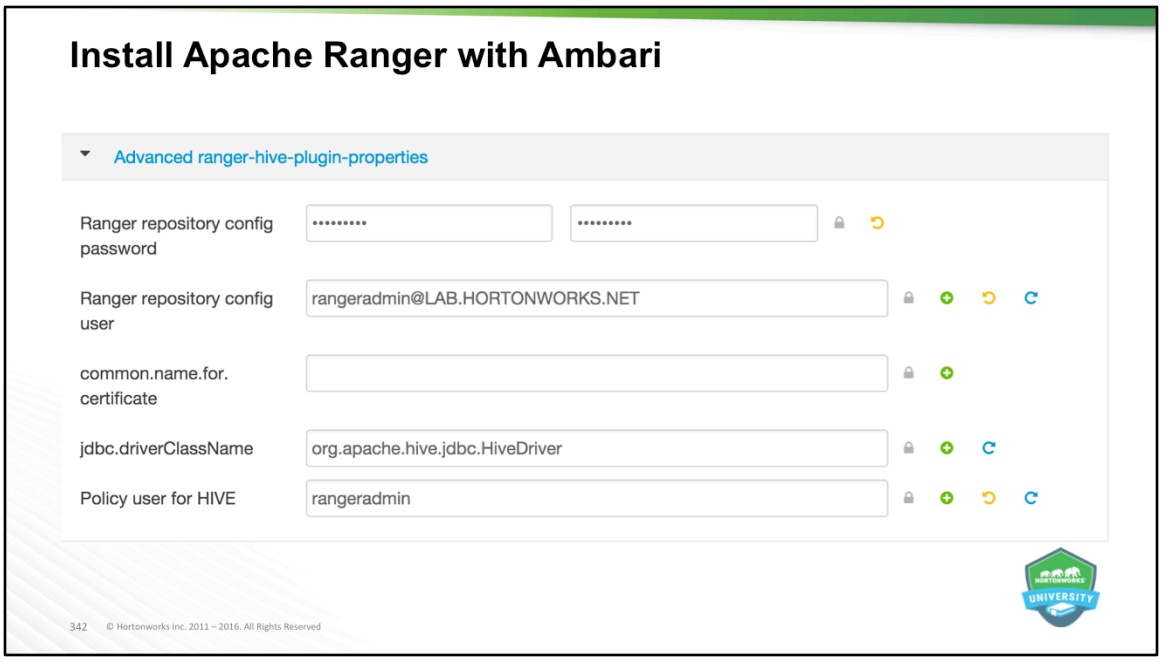


Now configure Hadoop components so Ranger can use rangeradmin@LAB.HORTONWORKS.NET principal to query HDFS, YARN, Hive, Hbase, Knox.

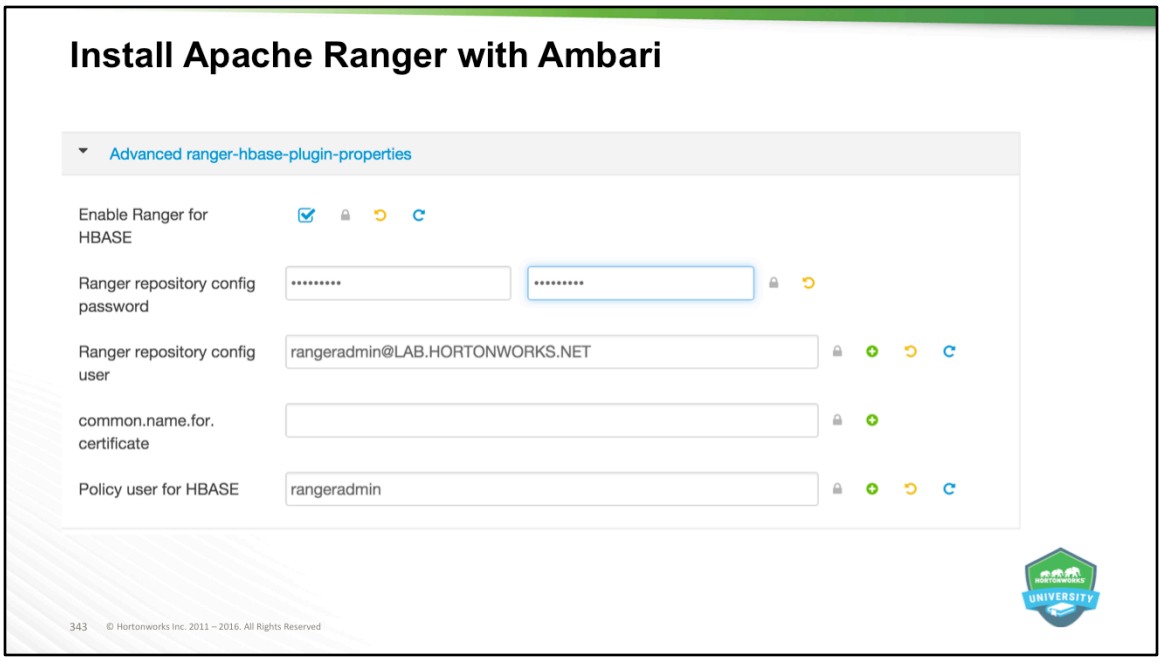
We will do this by clicking the tabs for each of these services and modifying Ranger specific properties.



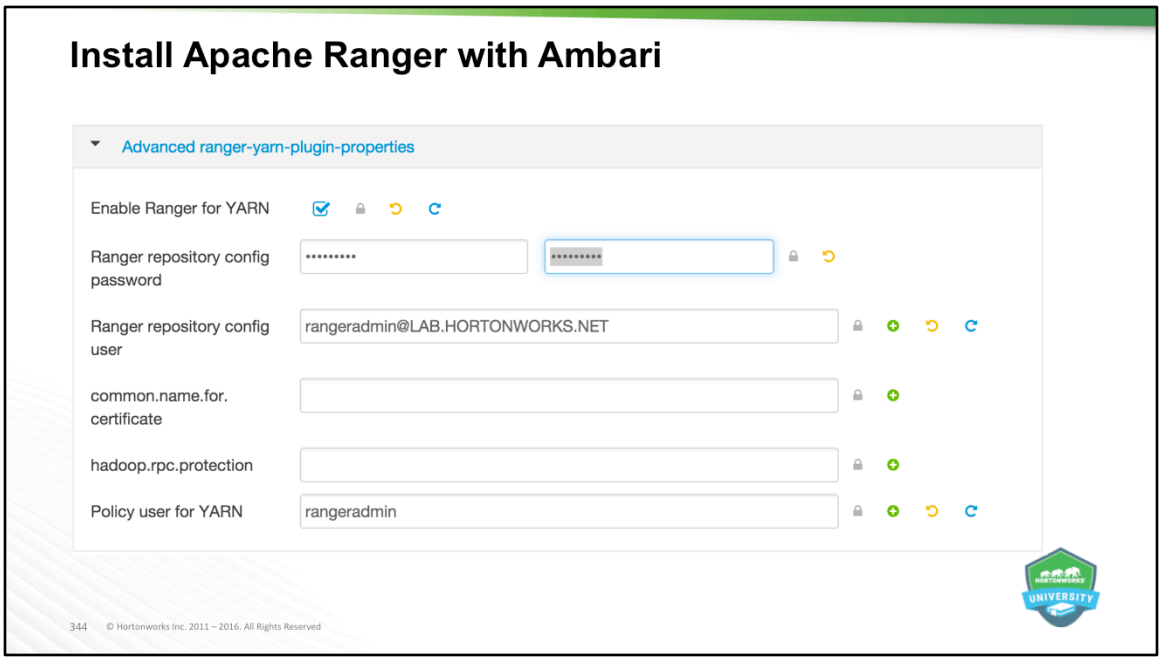
Now configure HDFS so Ranger can use rangeradmin@LAB.HORTONWORKS.NET principal to query HDFS



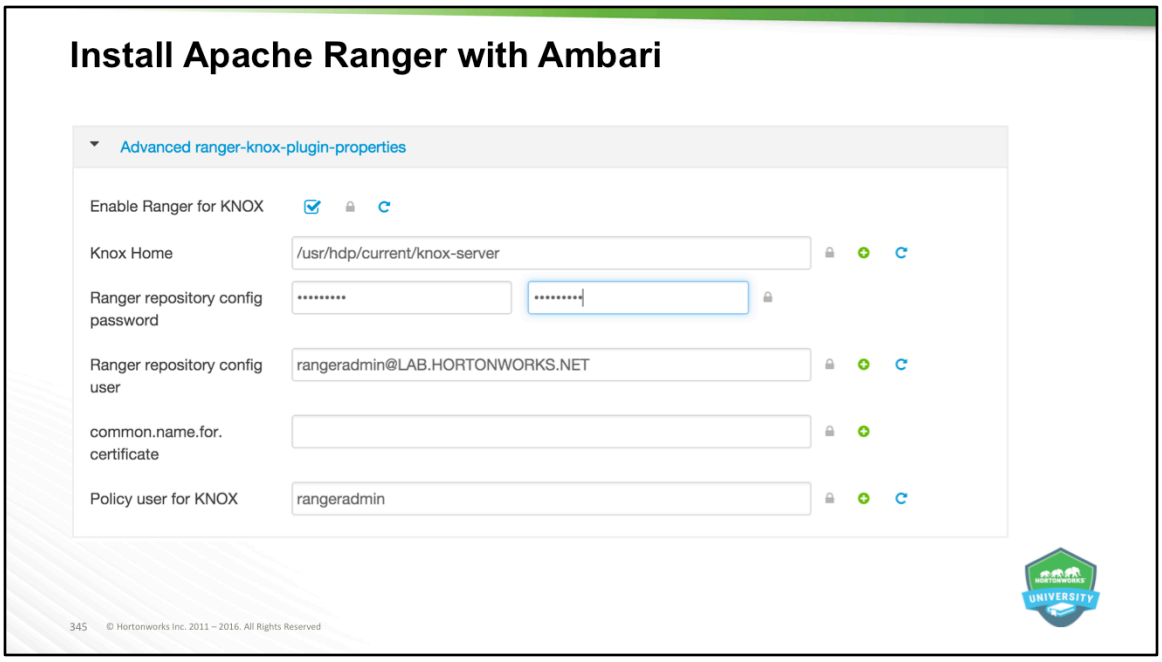
Now configure Hive so Ranger can use rangeradmin@LAB.HORTONWORKS.NET principal to query Hive



Now configure Hbase so Ranger can use rangeradmin@LAB.HORTONWORKS.NET principal to query Hbase

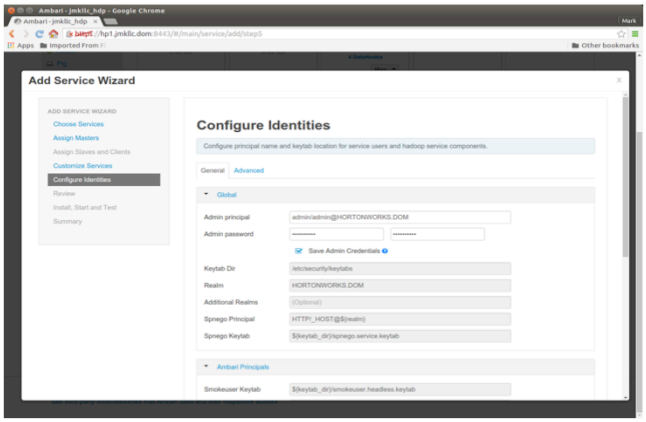


Now configure YARN so Ranger can use rangeradmin@LAB.HORTONWORKS.NET principal to query YARN



Now configure KNOX so Ranger can use rangeradmin@LAB.HORTONWORKS.NET principal to query KNOX

Install Apache Ranger with Ambari

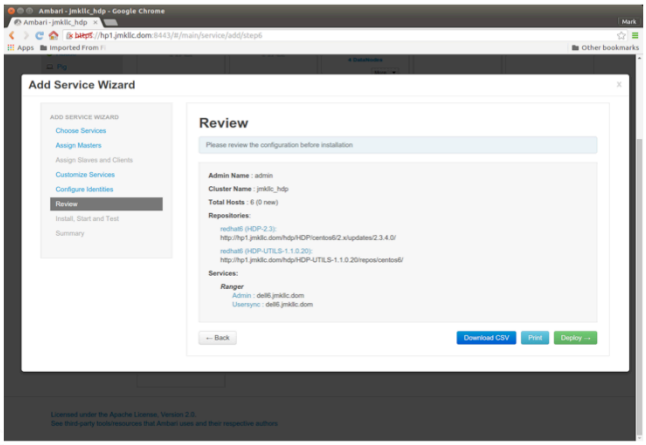


After clicking Next, Ambari will prompt for:

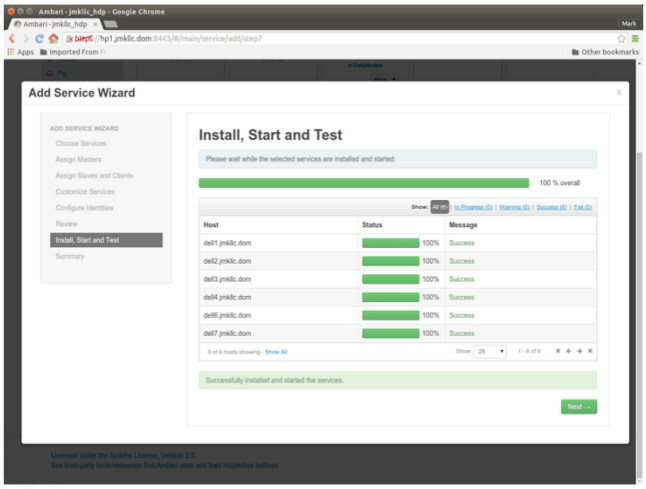
Admin principal: hadoopadmin@LAB.HORTONWORKS.NET

Admin password: BadPass#1

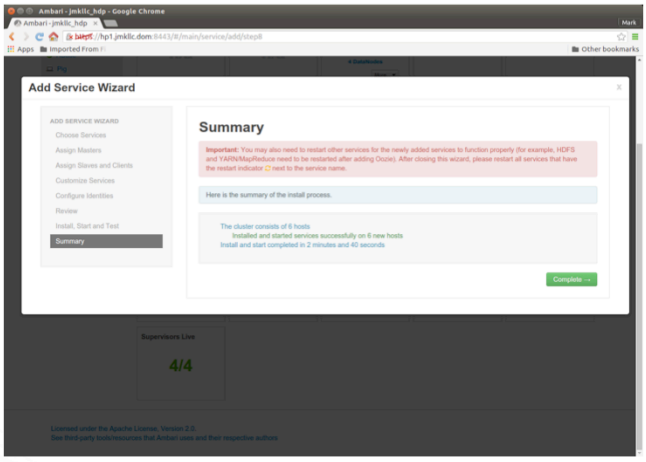
Install Apache Ranger with Ambari



Install Apache Ranger with Ambari

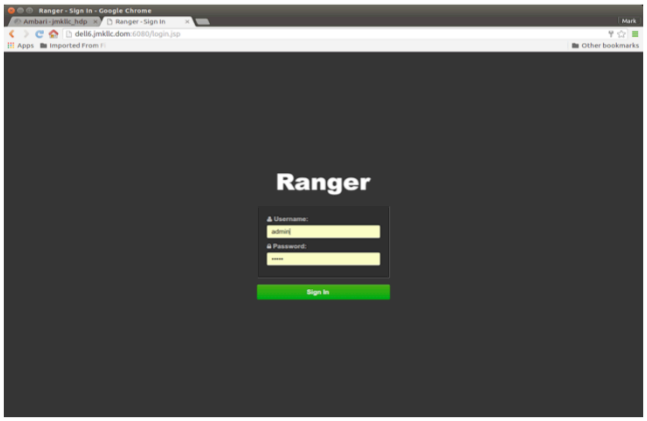


Install Apache Ranger with Ambari

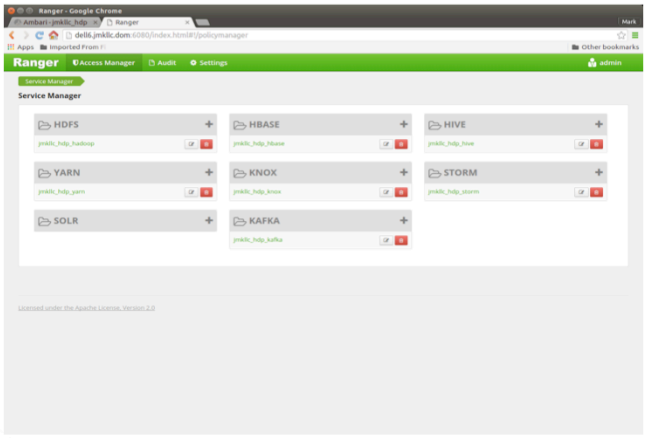


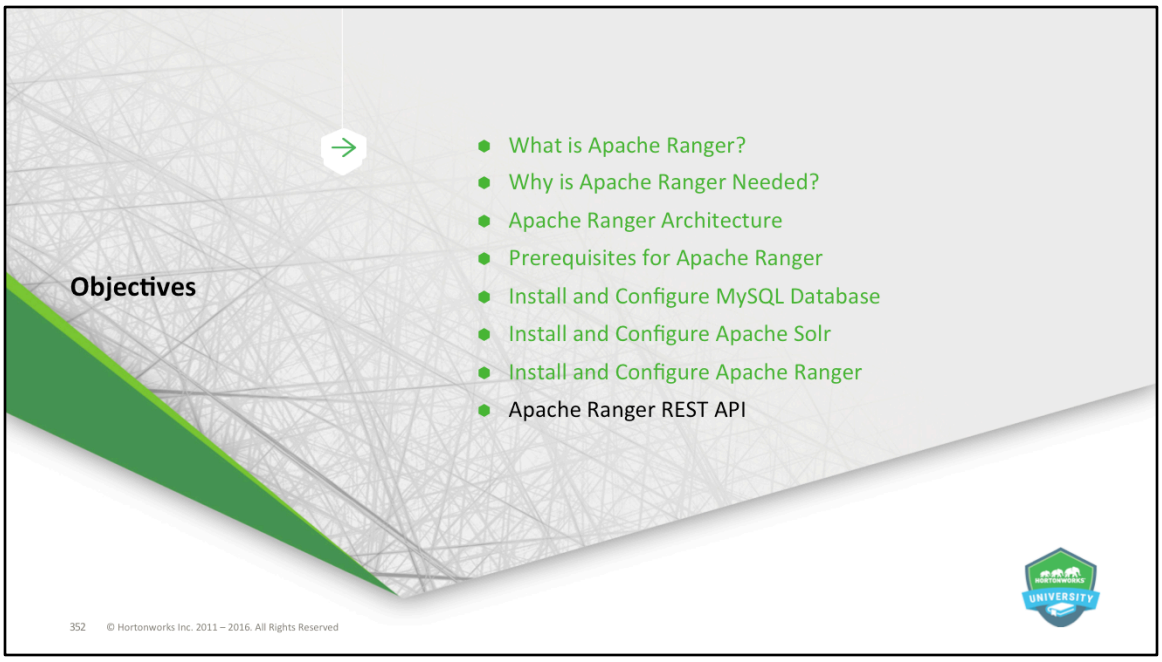
Point out that Services will need to be restarted for the Ranger Plugins show up in the Ranger UI

Install Apache Ranger with Ambari



Install Apache Ranger with Ambari




Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
Apache Ranger Architecture
Prerequisites for Apache Ranger
Install and Configure MySQL Database
Install and Configure Apache Solr
Install and Configure Apache Ranger
Apache Ranger REST API
352
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Apache Ranger REST API

- Sample REST API calls - more details available on Apache Ranger wiki
- GET request to *fetch details for policy* with ID = 2:

curl -iv -u <user>:<password> -H "Content-type:application/json" -X GET http://<RANGER_HOST>:6080/service/public/api/policy/2

```
{
  "id":2,  "createDate":"2015-11-21T07:03:21Z",  "updateDate":"2015-12-08T05:54:24Z",
  "owner":"Admin", "updatedBy":"Admin",  "policyName":"Ranger_audits", "description":"",
  "repositoryName":"bigdata_hadoop", "repositoryType":"hdfs", "resourceName":"/apps/solr/ranger_audits",
  "permMapList":[
    {
      "userList":["solr"],
      "groupList":[ ],
      "permList":[ "Read","Write","Execute"]
    }
  ],
  "isEnabled":true,  "isRecursive":true,  "isAuditEnabled":false, "version":"5",
  "replacePerm":false
}
```


http://ranger.apache.org

How does it work over Hadoop and related components

Apache Ranger' solution at the core has a centralized web application, which consists of the policy administration, audit and reporting modules. Authorized users will be able to manage their security policies using the web tool or using REST APIs. These security policies are enforced within Hadoop ecosystem using lightweight Ranger Java plugins, which run as part of the same process as the namenode (HDFS), Hive2Server(Hive), HBase server (Hbase), Nimbus server (Storm) and Knox server (Knox) respectively. Thus there is no additional OS level process to manage.

Is there a single point of failure?

No, Apache Ranger is not a Single Point of Failure. Ranger' plugins run within the same process as the component, e.g. NameNode for HDFS. These agents pull the policy-changes using REST API at a configured regular interval (e.g.: 30 second). The plugin is able to function even if the policy server is temporarily down and will provide the authorization enforcement. Also, the policy manager web application can

Apache Ranger REST API

- POST request to create *HDFS* policy:

curl -iv -u <user>:<password> -d @<policy payload> -H "Content-Type: application/json" -X POST http://<RANGER-Host>:6080/service/public/api/policy

- Payload to be passed in:

```
{
    "policyName": "name_of_policy",
    "resourceName": "/path1,/path2/blub",
    "description": "", "repositoryName": "",
    "repositoryType": "hdfs",
    "isEnabled": "true", "isRecursive": "true", "isAuditEnabled": "true",
    "permMapList": [{
        "groupList": ["somegroup"],
        "permList": ["Read","Execute", "Write", "Admin"]
    }]
}
```


http://ranger.apache.org

How does it work over Hadoop and related components

Apache Ranger' solution at the core has a centralized web application, which consists of the policy administration, audit and reporting modules. Authorized users will be able to manage their security policies using the web tool or using REST APIs. These security policies are enforced within Hadoop ecosystem using lightweight Ranger Java plugins, which run as part of the same process as the namenode (HDFS), Hive2Server(Hive), HBase server (Hbase), Nimbus server (Storm) and Knox server (Knox) respectively. Thus there is no additional OS level process to manage.

Is there a single point of failure?

No, Apache Ranger is not a Single Point of Failure. Ranger' plugins run within the same process as the component, e.g. NameNode for HDFS. These agents pull the policy-changes using REST API at a configured regular interval (e.g.: 30 second). The plugin is able to function even if the policy server is temporarily down and will provide the authorization enforcement. Also, the policy manager web application can

Apache Ranger REST API

- POST request to create *Hive* policy:

curl -iv -u <user>:<password> -d @<policy payload> -H "Content-Type: application/json" -X POST http://<RANGER-Host>:6080/service/public/api/policy

- Payload to be passed in:

```
{
"policyName":"name_of_policy",
  "databases":"db1,db2",
  "tables":"mytable,yourtable",  "columns":"",
  "udfs":"",  "description":"",  "repositoryName":"",
  "repositoryType":"hive",  "tableType":"Inclusion",  "columnType":"Inclusion",
  "isEnabled":"true",  "isAuditEnabled":"true",
  "permMapList": [{
            "groupList": ["somegroup"],
            "permList": ["Select"]
      }]
}
```

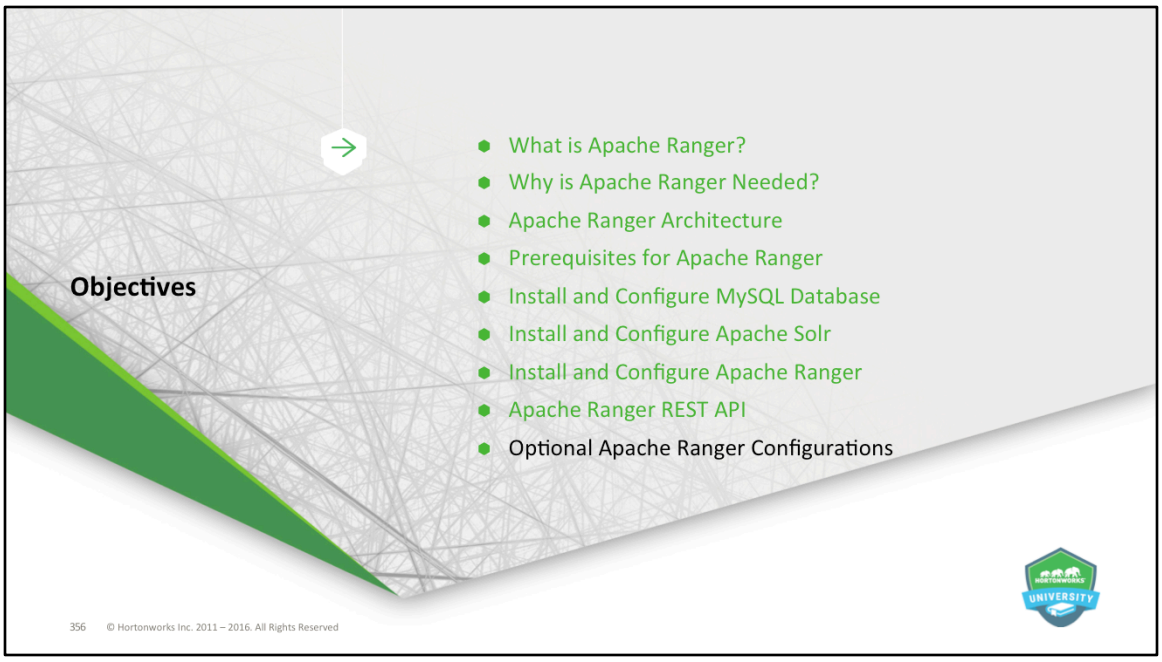

http://ranger.apache.org

How does it work over Hadoop and related components

Apache Ranger' solution at the core has a centralized web application, which consists of the policy administration, audit and reporting modules. Authorized users will be able to manage their security policies using the web tool or using REST APIs. These security policies are enforced within Hadoop ecosystem using lightweight Ranger Java plugins, which run as part of the same process as the namenode (HDFS), Hive2Server(Hive), HBase server (Hbase), Nimbus server (Storm) and Knox server (Knox) respectively. Thus there is no additional OS level process to manage.

Is there a single point of failure?

No, Apache Ranger is not a Single Point of Failure. Ranger' plugins run within the same process as the component, e.g. NameNode for HDFS. These agents pull the policy-changes using REST API at a configured regular interval (e.g.: 30 second). The plugin is able to function even if the policy server is temporarily down and will provide the authorization enforcement. Also, the policy manager web application can


Objectives
What is Apache Ranger?
Why is Apache Ranger Needed?
Apache Ranger Architecture
Prerequisites for Apache Ranger
Install and Configure MySQL Database
Install and Configure Apache Solr
Install and Configure Apache Ranger
Apache Ranger REST API
Optional Apache Ranger Configurations
356
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Optional Apache Ranger Configurations

- Supported Databases
  - MySQL
  - PostgreSQL
  - Oracle
  - MS SQL Server
- Ranger Authentication - Access to Console
  - UNIX
  - Active Directory
  - LDAP/LDAPS
- User Synchronization
  - UNIX Usersync
  - LDAP Usersync
  - AD Usersync

Knowledge Check

Knowledge Check

1. Which of the 5 Security Pillars does Apache Ranger address?
2. Apache Ranger provides a framework to do what?
3. Apache Ranger provide enhanced support for Authorization Methods, name the two methods?
4. True/False - Ranger provides centralized auditing of all user access and administrative action with in all the Hadoop components?

1. Administration, Authorization, Audit and Data Protection (Ranger KMS)

2. Enable, Monitor and Manage comprehensive data security across the Hadoop platform

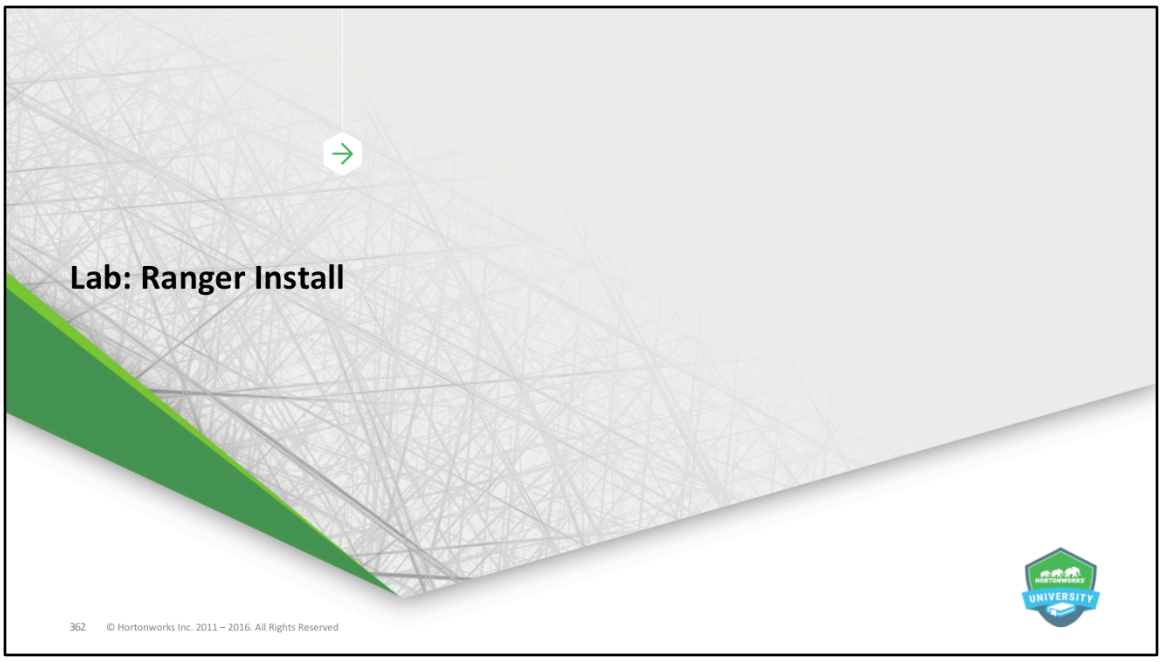
3. Role Based Access Control and Attribute Bases Access Control

4. TRUE

Summary

Summary

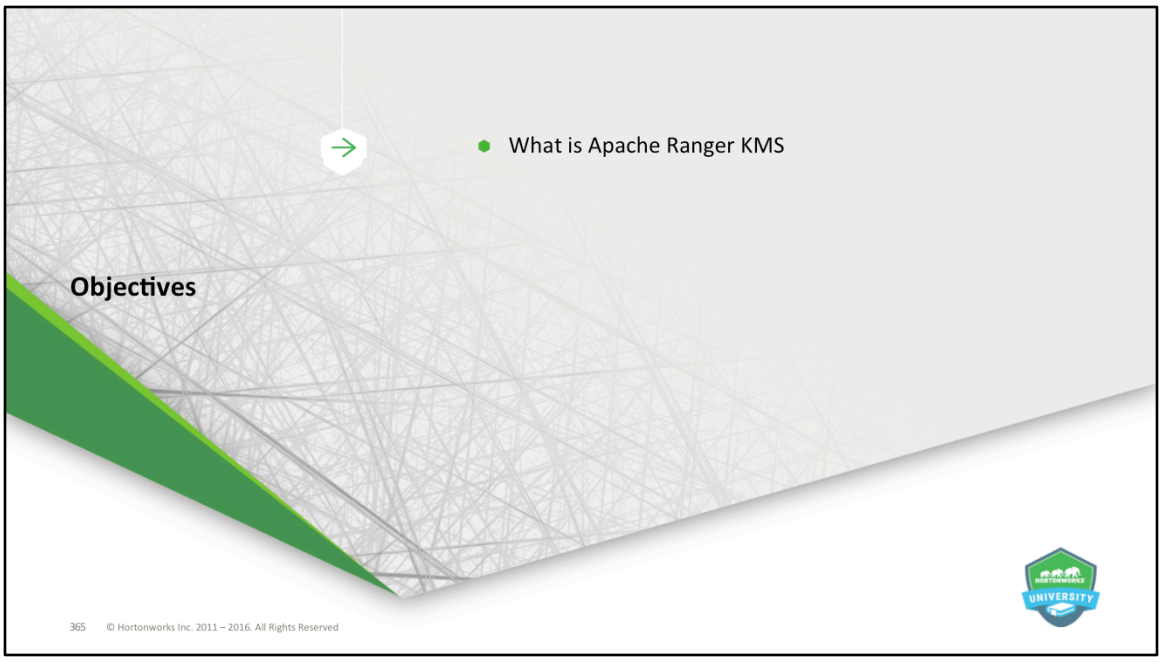
- Apache Ranger Provides a Framework to Enable, Monitor, & Manage Comprehensive Data Security Across the Apache Hadoop Platform
- Apache Ranger has Centralized Security Administration to Manage All Security Related Tasks in a Central UI or using REST APIs.
- Centralized Security Administration has Four Aspects: Authentication, Authorization, Audit and Data Protection
- Can Enable Audit Tracking and Policy Analytics for Deeper Control of the Hadoop environment
- Provides Ability to Delegate Administration of Certain Data to Other Group Owners, with the Aim of Decentralizing Data Ownership


Lab: Ranger Install
362
© Hortonworks Inc. 2011 – 2016. All Rights Reserved


Apache Ranger KMS
and HDFS Encryption
HORTONWORKS
UNIVERSITY

Objectives

- Describe Apache Ranger KMS
- Describe HDFS Encryption
- Install and Configure Apache Ranger KMS
- Configure HDFS to Use Encryption
- Describe HDFS Encryption Zones


What is Apache Ranger KMS
Objectives
365
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

What is Apache Ranger KMS

- Open Source, Scalable Cryptographic Key Management Service
- Supports HDFS ”Data at Rest” Encryption
- Ranger KMS Based on Hadoop KMS Originally Developed by the Apache Community
- Hadoop KMS Stores Keys by Default in a File-Based Java Keystore
- Ranger Extends Native Hadoop KMS Functionality by Storing Keys in a Secure Database

Ranger Key Management Service (Ranger KMS) is a open source, scalable cryptographic key management service supporting HDFS "data at rest" encryption*.

Ranger KMS is based on the Hadoop KMS originally developed by the Apache community. The Hadoop KMS stores keys in a file-based Java keystore by default. Ranger extends the native Hadoop KMS functionality by allowing you to store keys in a secure database.

Ranger provides centralized administration of the key management server through the Ranger admin portal.

There are three main functions within the Ranger KMS:

Key management. Ranger admin provides the ability to create, update or delete keys using the Web UI or REST APIs.

What is Apache Ranger KMS

- Ranger Provides Centralized Administration of Key Management
- Three Main Functions Within Ranger KMS:
  - Key Management
    - Ranger Admin Provides the Ability to Create, Update or Delete Keys using Web UI or REST APIs
  - Access Control Policies
    - Ranger Admin Provides the Ability to Manage Access Control Policies Within Ranger KMS.
    - Access Policies Control Permissions to Generate or Manage Keys
    - Adds Another Layer of Security for Data Encrypted in Hadoop
  - Audit
    - Ranger Provides Full Audit Trace of All Actions Performed by Ranger KMS

Ranger Key Management Service (Ranger KMS) is a open source, scalable cryptographic key management service supporting HDFS "data at rest" encryption*.

Ranger KMS is based on the Hadoop KMS originally developed by the Apache community. The Hadoop KMS stores keys in a file-based Java keystore by default. Ranger extends the native Hadoop KMS functionality by allowing you to store keys in a secure database.

Ranger provides centralized administration of the key management server through the Ranger admin portal.

There are three main functions within the Ranger KMS:

Key management. Ranger admin provides the ability to create, update or delete keys using the Web UI or REST APIs.

What is Apache Ranger KMS

- Ranger KMS with HDFS Encryption are Recommended For Use in All Clusters
- Ranger KMS Service is Scalable
- Multiple Instances of Can Be Configured Behind a Load Balancer.

Ranger Key Management Service (Ranger KMS) is a open source, scalable cryptographic key management service supporting HDFS "data at rest" encryption*.

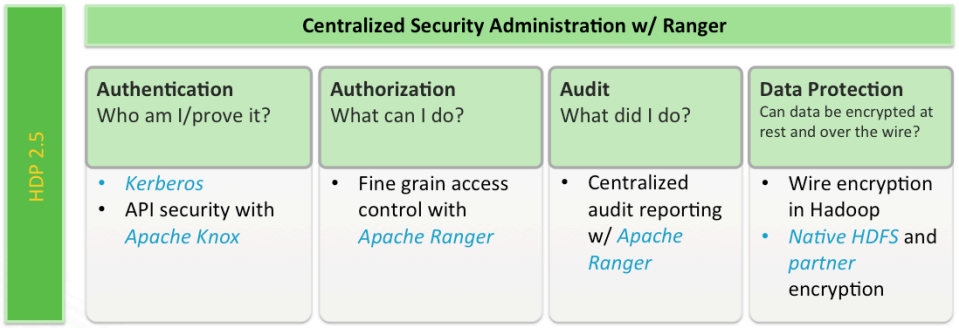
Ranger KMS is based on the Hadoop KMS originally developed by the Apache community. The Hadoop KMS stores keys in a file-based Java keystore by default. Ranger extends the native Hadoop KMS functionality by allowing you to store keys in a secure database.

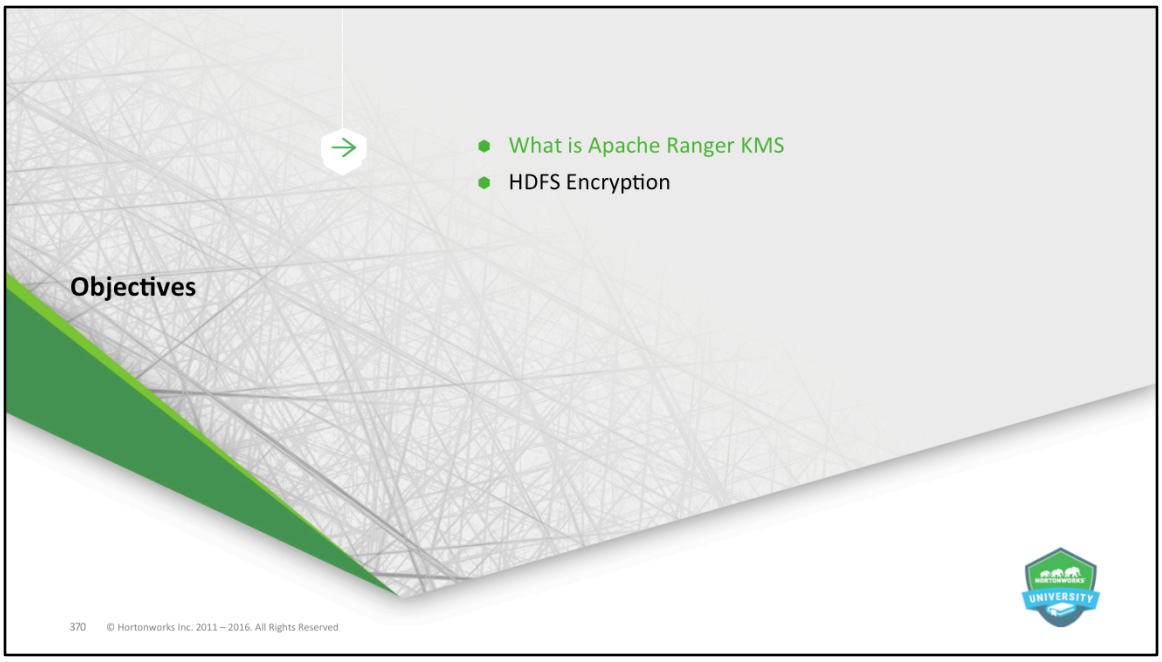
Ranger provides centralized administration of the key management server through the Ranger admin portal.

There are three main functions within the Ranger KMS:

Key management. Ranger admin provides the ability to create, update or delete keys using the Web UI or REST APIs.

What is Apache Ranger KMS




What is Apache Ranger KMS
HDFS Encryption
Objectives
370
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

HDFS Encryption

- Encryption is a Form of Data Security
- Hadoop Provides Several Ways to Encrypt Stored Data
- Lowest Level is Volume Encryption – Entire Disk Drive
  - Protects Data After Physical Theft
  - Accidental Loss of Disk Volume
  - Does Not Support Finer-Grained Encryption of Specific Files/Directories
  - Will Not Protect Against Viruses or Other Attacks Occurring While System is Running
- Application Level Encryption
  - Encryption Within Application Running on Hadoop
  - Support Higher Level of Granularity
  - Prevents “Rogue Admin” Access
  - But Adds Layer of Complexity to Application Architecture

Encryption is a form of data security that is required in industries such as healthcare and the payment card industry. Hadoop provides several ways to encrypt stored data.

The lowest level of encryption is volume encryption, which protects data after physical theft or accidental loss of a disk volume. The entire volume is encrypted; this approach does not support finer-grained encryption of specific files or directories. In addition, volume encryption does not protect against viruses or other attacks that occur while a system is running.

Application level encryption (encryption within an application running on top of Hadoop) supports a higher level of granularity and prevents "rogue admin" access, but adds a layer of complexity to the application architecture.

A third approach, HDFS data at rest encryption, encrypts selected files and directories stored ("at rest") in HDFS. This approach uses specially designated HDFS directories known as "encryption zones."

HDFS Encryption

- HDFS Data at Rest Encryption
  - Encrypts Selected/Specific Files and Directories
  - Uses Specially Designated HDFS Directories known as “Encryption Zones”

Encryption is a form of data security that is required in industries such as healthcare and the payment card industry. Hadoop provides several ways to encrypt stored data.

The lowest level of encryption is volume encryption, which protects data after physical theft or accidental loss of a disk volume. The entire volume is encrypted; this approach does not support finer-grained encryption of specific files or directories. In addition, volume encryption does not protect against viruses or other attacks that occur while a system is running.

Application level encryption (encryption within an application running on top of Hadoop) supports a higher level of granularity and prevents "rogue admin" access, but adds a layer of complexity to the application architecture.

A third approach, HDFS data at rest encryption, encrypts selected files and directories stored ("at rest") in HDFS. This approach uses specially designated HDFS directories known as "encryption zones."

HDFS Encryption

- Data At Rest Encryption Implements End-to-End Encryption of Data Read From/Written To HDFS
- End-to-End Encryption – Data is Encrypted/Decrypted by Client
- HDFS Does Not Have Access to Unencrypted Data/Keys
- Encryption Involves Several Elements
  - HDFS Encryption Zone – Special HDFS Directory where Data
    - Encrypted on Write
    - Decrypted on Read
  - EZ Key – Master Encryption Key associated with all files in an EZ
  - Each Zone Associated with Encryption Key When Zone is Created
  - Each File within Zone has Unique Encryption Key – "Data Encryption Key" (DEK)

HDFS data at rest encryption implements end-to-end encryption of data read from and written to HDFS. End-to-end encryption means that data is encrypted and decrypted only by the client. HDFS does not have access to unencrypted data or keys.

HDFS encryption involves several elements:

Encryption key: A new level of permission-based access protection, in addition to standard HDFS permissions.

HDFS encryption zone: A special HDFS directory within which all data is encrypted upon write, and decrypted upon read.

Each encryption zone is associated with an encryption key that is specified when the zone is created.

Each file within an encryption zone has a unique encryption key, called the "data

HDFS Encryption

- HDFS Does Not Have Access to DEKs (only to EDEKs)
- DataNodes See Stream of Encrypted Bytes.
- HDFS Stores “Encrypted Data Encryption Keys" (EDEKs) as Part of the File's Metadata on the NameNode.
- Clients Decrypt an EDEK and Use the Associated DEK to Encrypt/Decrypt Data During Write/ Read Operations
- Ranger KMS has these Basic Responsibilities
  - Provide Access to Stored Encryption Zone Keys
  - Generate and Manage Encryption Zone Keys and Create Encrypted Data Keys to be Stored in Hadoop
  - Decrypt EDEKs to DEKs for HDFS clients
  - Audit All Access Events in Ranger KMS

HDFS data at rest encryption implements end-to-end encryption of data read from and written to HDFS. End-to-end encryption means that data is encrypted and decrypted only by the client. HDFS does not have access to unencrypted data or keys.

HDFS encryption involves several elements:

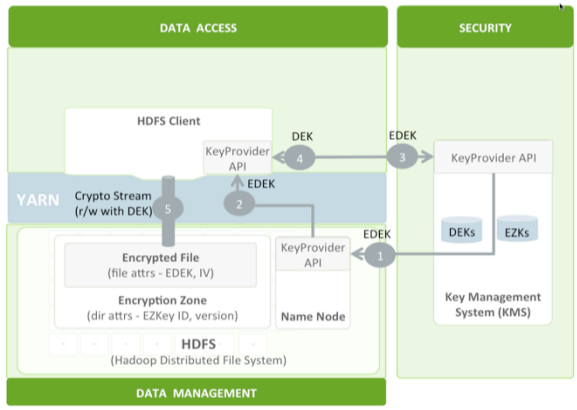
Encryption key: A new level of permission-based access protection, in addition to standard HDFS permissions.

HDFS encryption zone: A special HDFS directory within which all data is encrypted upon write, and decrypted upon read.

Each encryption zone is associated with an encryption key that is specified when the zone is created.

Each file within an encryption zone has a unique encryption key, called the "data

Apache Ranger KMS



Acronym	Description
EZ	Encryption Zone (an HDFS directory)
EZK	Encryption Zone Key; master key associated with all files in an EZ
DEK	Data Encryption Key, unique key associated with each file. EZ Key used to generate DEK
EDEK	Encrypted DEK, Name Node only has access to encrypted DEK.
IV	Initialization Vector

Process for writing a file when HDFS encryption enabled is shown here

0. Pre-req: EZ must be created
- KMS admin creates a Encryption Zone (EZ) Key (hadoop key create mykey)
- HDFS admin creates an Encryption Zone using the EZKey (hdfs crypto -createZone -keyName myKey -path /home/me/zone1). Assuming that /home/me/zone1 already exists and empty
- EZ created with EZK id/version stored as dir attributes (as shown in EZ box here)

When user creates a file in the EZ with appropriate access permissions,

1. Name Node uses EZ Key ID and version to get a EDEK (encrypted data encryption key) from KMS

2. Name Node returns EDEK to HDFS Client

3. HDFS client requests KMS to decrypt EDEK

4. KMS checks that the client has permission to access the encryption zone key version. If successful, passes client DEK

5. HDFS client uses DEK and Hadoop Cryptographic File System (CryptoOutputStream) to write an encrypted file in the HDFS EZ.

The EDEK is then stored persistently as part of the file xattr (as shown in Encrypted File box above)

When user reads a file in the EZ, check for appropriate permissions and KMS authorization
- Name Node passes EDEK to KMS
- KMS provides HDFS Client with DEK
- HDFS client uses DEK and HDFS Cryptographic File System (CryptoInputStream) to read the un-encrypted contents of the file

HDFS Encryption

- Role Separation
  - Access to the Key Encryption/Decryption Process Typically Restricted to End Users
    - Encrypted Keys Safely Stored and Handled by HDFS
    - HDFS Admin User does not have access to the Encrypted Keys
    - So even if HDFS compromised, malicious user only gains access to ciphertext and encrypted keys
  - Recommended: create a separate HDFS admin user for Key Management.
  - Results in Two Types of HDFS Administrator Accounts:
    - HDFS service user: the system-level account associated with HDFS (hdfs by default)
    - HDFS admin user: an account in hdfs supergroup, used by HDFS administrators to configure/manage HDFS
  - For example, to create HDFS admin user called : **encrypter**
    - In Advanced hdfs-site: dfs.cluster.administrators = hdfs, **encrypter**
    - In Advanced dbks-site: hadoop.kms.blacklist.DECRYPT_EEK = hdfs, **encrypter**

Ranger Key Management Service (Ranger KMS): An open source key management service based on Hadoop's KeyProvider API.

For HDFS encryption, the Ranger KMS has three basic responsibilities:

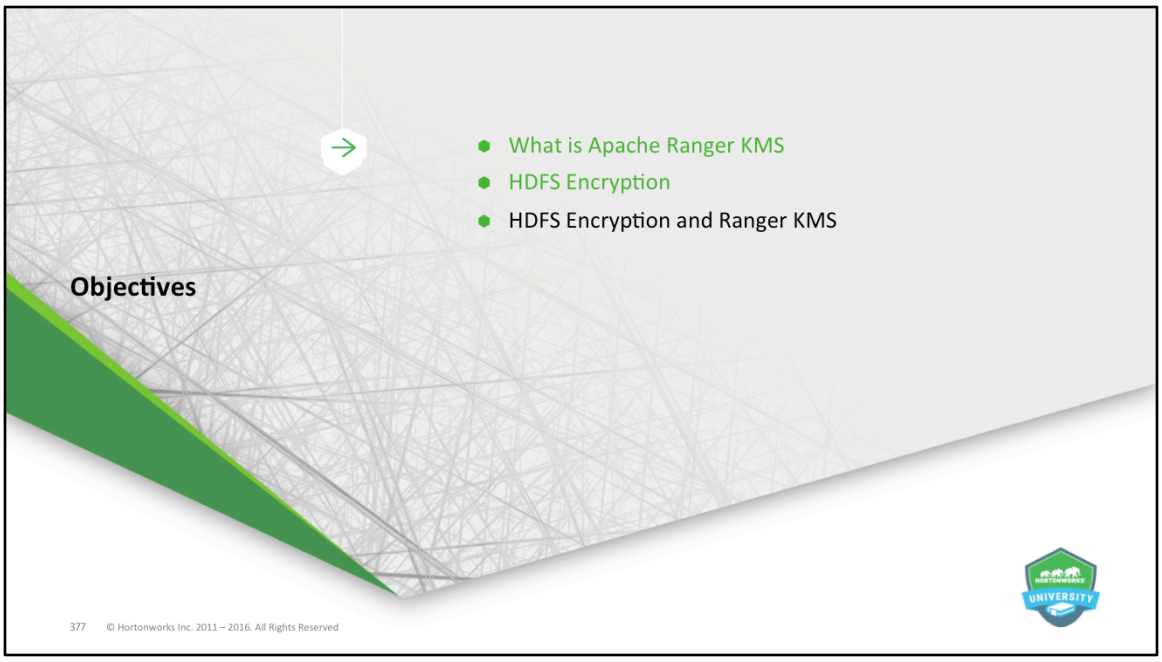
Provide access to stored encryption zone keys.

Generate and manage encryption zone keys, and create encrypted data keys to be stored in Hadoop.

Audit all access events in Ranger KMS.

Role Separation

Access to the key encryption/decryption process is typically restricted to end users.


Objectives
What is Apache Ranger KMS
HDFS Encryption
HDFS Encryption and Ranger KMS
377
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Overview HDFS Encryption and Ranger KMS

- Once Ranger KMS is Set Up – NameNode and HDFS Client Configured
- HDFS Administrator can Use Command Line Tools
  - Create Encryption Keys – "hadoop key"
  - Set Up New Encryption Zones – "hdfs crypto"
- Overall WorkFlow
  - Create HDFS encryption zone key that will be
    - used to encrypt the file-level data in the encryption zone
    - key is stored and managed by Ranger KMS
  - Create a new HDFS folder
    - Specify required permissions, owner, and group for the folder
  - Using the new encryption zone key, designate the folder as an encryption zone.

After the Ranger KMS has been set up and the NameNode and HDFS clients have been configured, an HDFS administrator can use the hadoop key and hdfs crypto command-line tools to create encryption keys and set up new encryption zones.

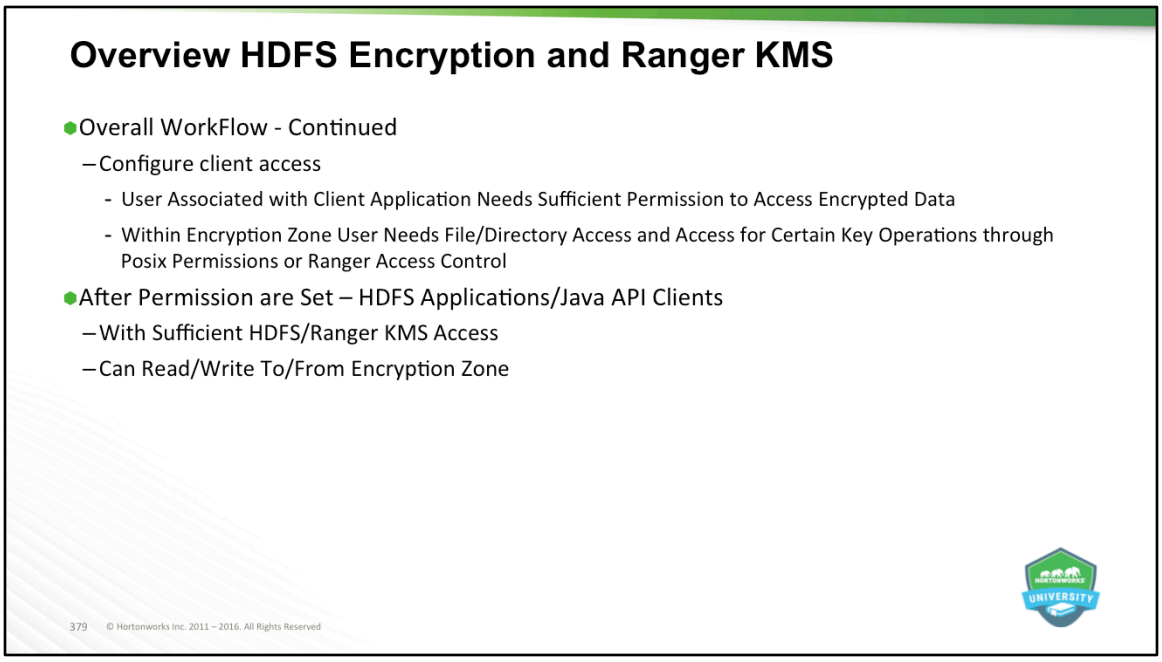
The overall workflow is as follows:

Create an HDFS encryption zone key that will be used to encrypt the file-level data encryption key for every file in the encryption zone. This key is stored and managed by Ranger KMS.

Create a new HDFS folder. Specify required permissions, owner, and group for the folder.

Using the new encryption zone key, designate the folder as an encryption zone.

Configure client access. The user associated with the client application needs



After the Ranger KMS has been set up and the NameNode and HDFS clients have been configured, an HDFS administrator can use the hadoop key and hdfs crypto command-line tools to create encryption keys and set up new encryption zones.

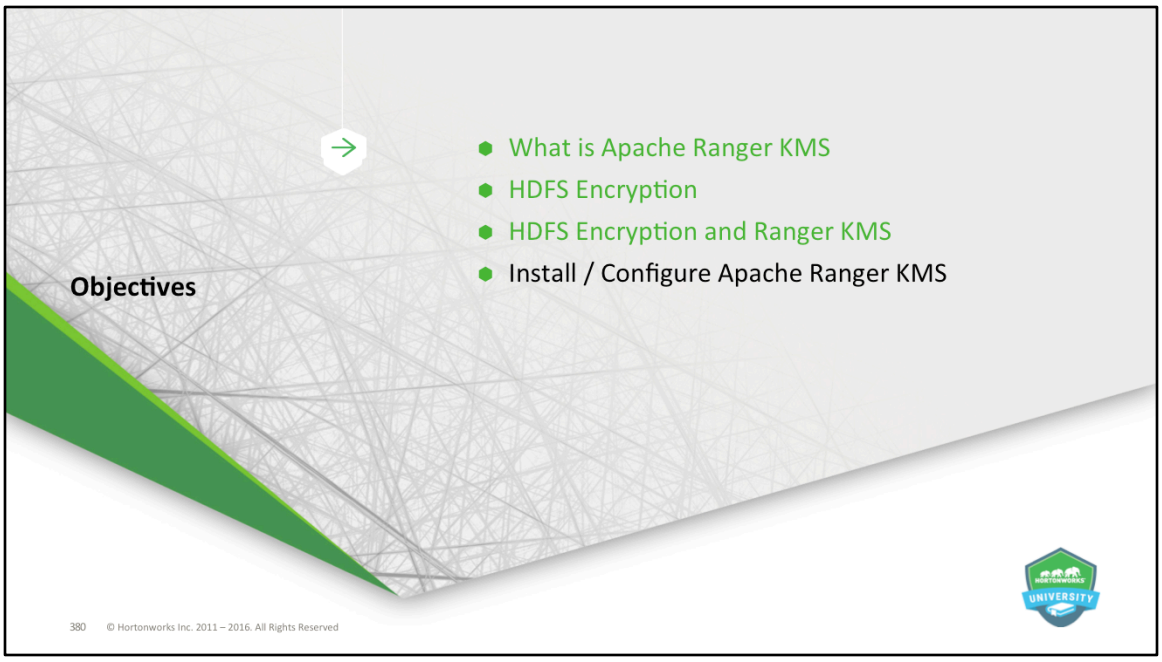
The overall workflow is as follows:

Create an HDFS encryption zone key that will be used to encrypt the file-level data encryption key for every file in the encryption zone. This key is stored and managed by Ranger KMS.

Create a new HDFS folder. Specify required permissions, owner, and group for the folder.

Using the new encryption zone key, designate the folder as an encryption zone.

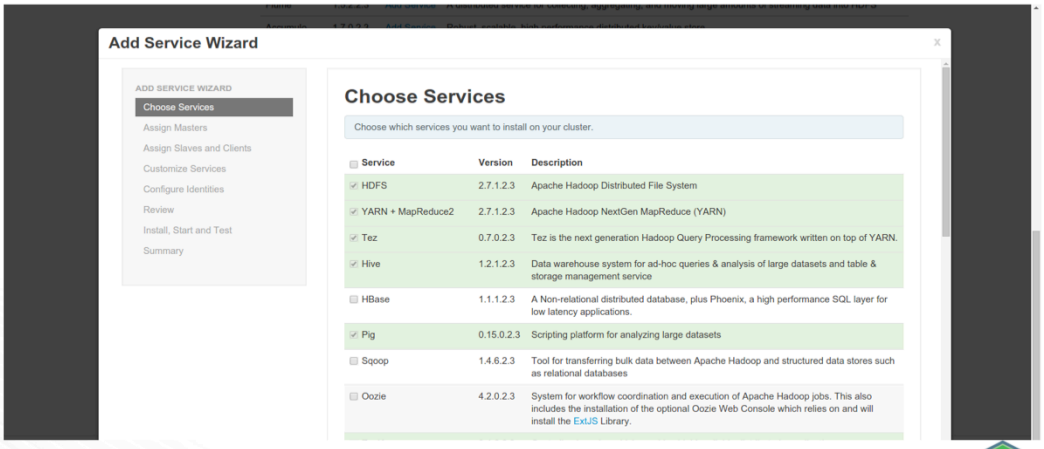
Configure client access. The user associated with the client application needs


Objectives
What is Apache Ranger KMS
HDFS Encryption
HDFS Encryption and Ranger KMS
Install / Configure Apache Ranger KMS
380
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

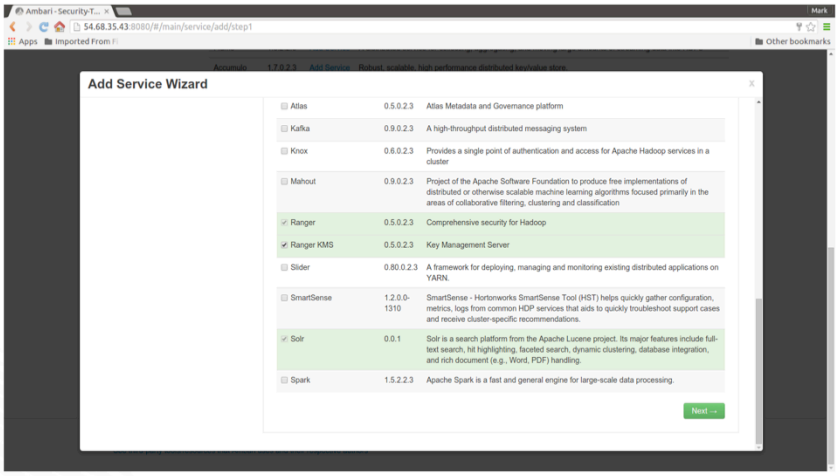
Install Apache Ranger KMS with Ambari

- Install Ranger KMS Via Add Service Wizard Under “Admin > Stack and Versions”
- Add Service Wizard – Select Ranger KMS
- Choose Services – Confirm Ranger KMS is Selected
- Assign Master – Choose Hosts to Run Ranger KMS
- Customize Services – Details in next slides
- Review Configuration – Deploy
- Monitor Deployment – Install, Start and Test
- Review Summary
- Installation Complete

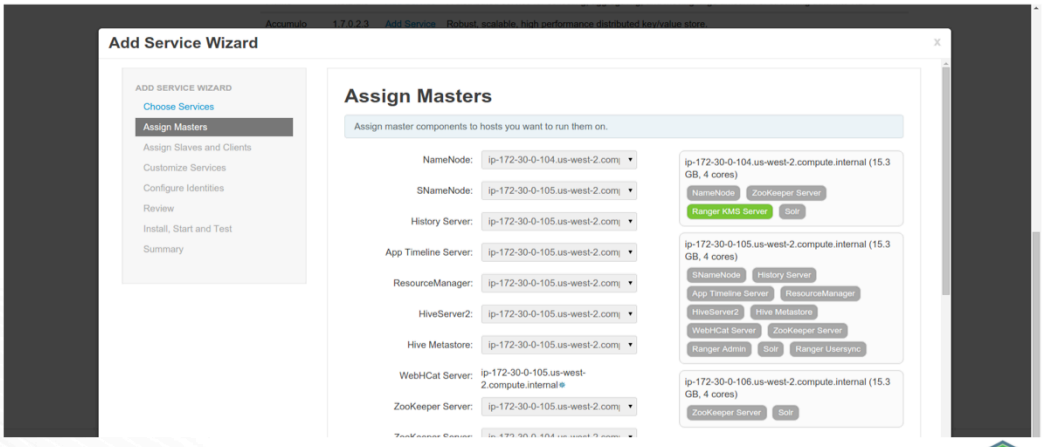
Install Apache Ranger KMS with Ambari



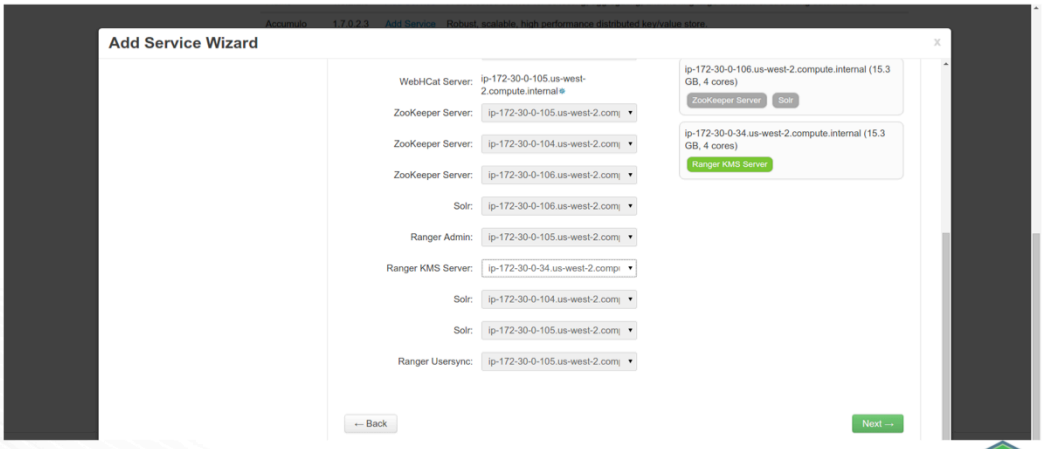
Install Apache Ranger KMS with Ambari



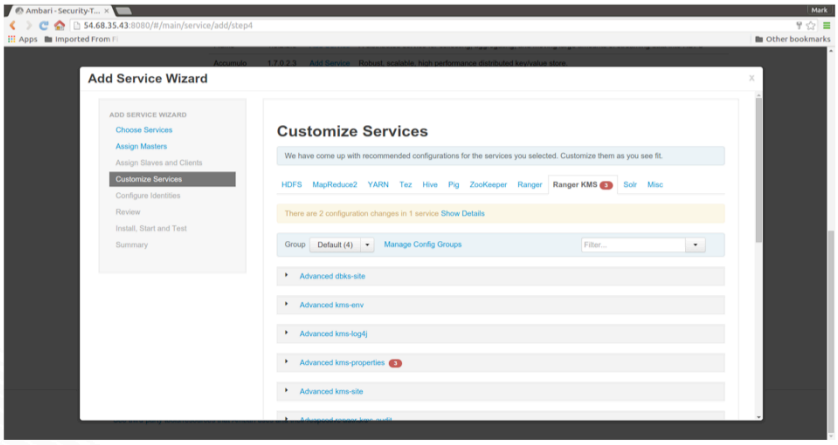
Install Apache Ranger KMS with Ambari



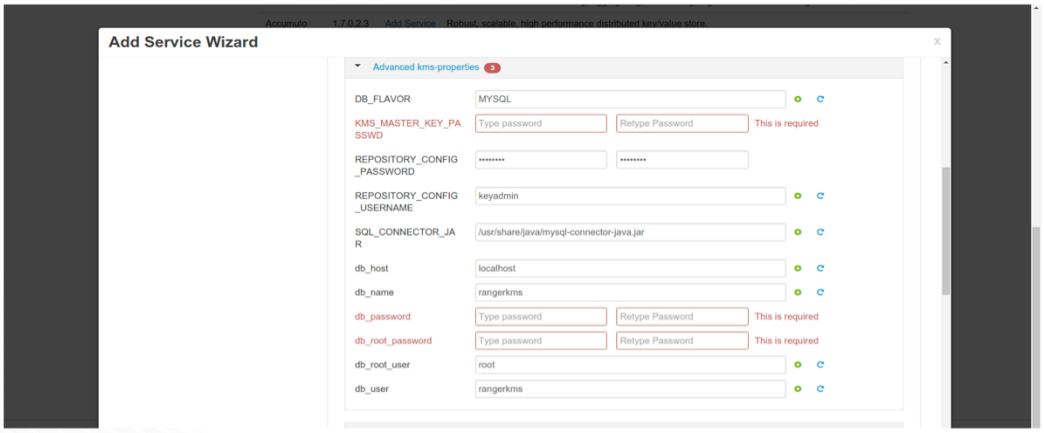
Install Apache Ranger KMS with Ambari



Install Apache Ranger KMS with Ambari



Install Apache Ranger KMS with Ambari



387 © Hortonworks Inc. 2011 – 2016. All Rights Reserved

Advanced kms-properties:

KMS_MASTER_KEY_PASSWORD = BadPass#1

REPOSIORY_CONFIG_USERNAME = keyadmin@LAB.HORTONWORKS.NET

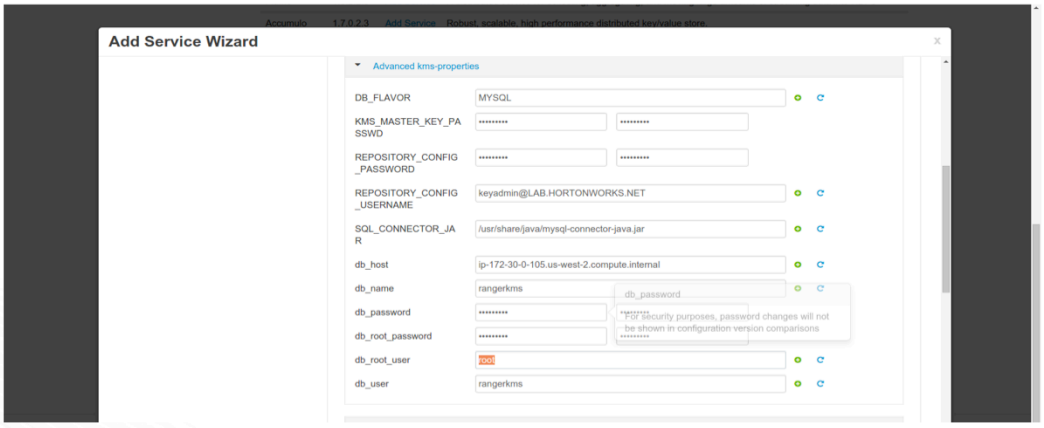
REPOSIORY_CONFIG_PASSWORD = BadPass#1

db_host = Internal FQDN of MySQL node e.g. ip-172-30-0-181.us-west-2.compute.internal

db_password = BadPass#1

db_root_password = BadPass#1

Install Apache Ranger KMS with Ambari



Advanced kms-properties:

KMS_MASTER_KEY_PASSWORD = BadPass#1

REPOSIORY_CONFIG_USERNAME = keyadmin@LAB.HORTONWORKS.NET

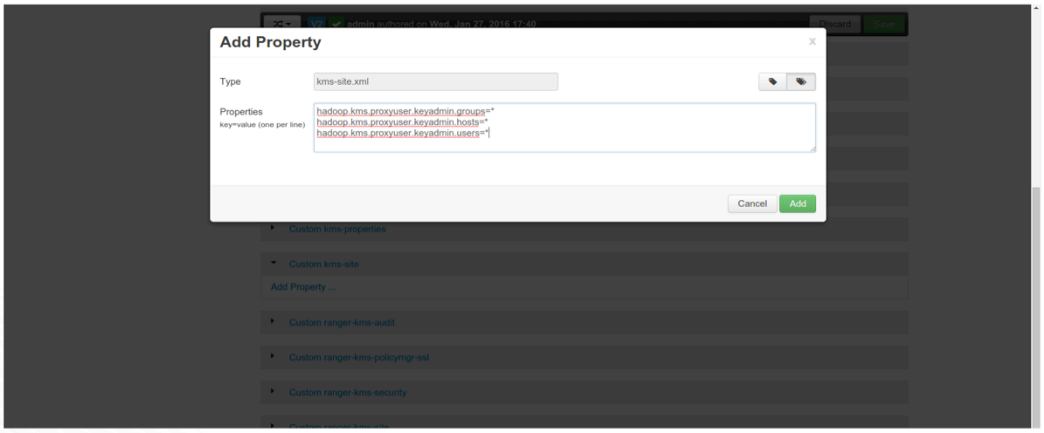
REPOSIORY_CONFIG_PASSWORD = BadPass#1

db_host = Internal FQDN of MySQL node e.g. ip-172-30-0-181.us-west-2.compute.internal

db_password = BadPass#1

db_root_password = BadPass#1

Install Apache Ranger KMS with Ambari



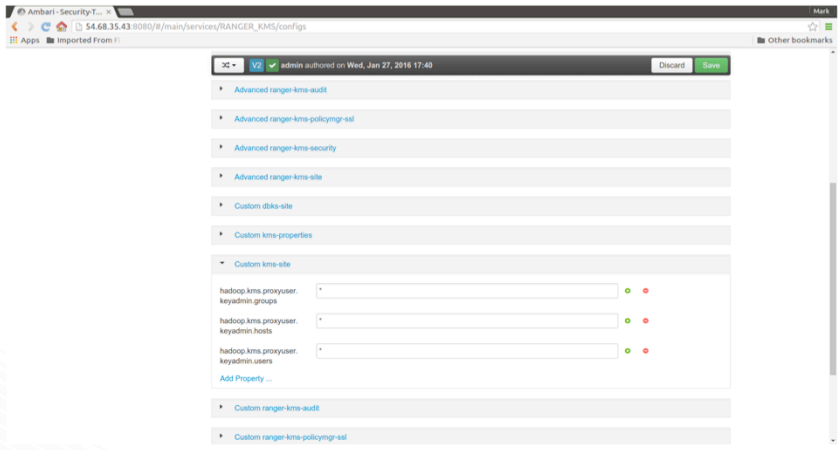
Kms-site:

hadoop.kms.proxyuser.keyadmin.groups=*

hadoop.kms.proxyuser.keyadmin.hosts=*

hadoop.kms.proxyuser.keyadmin.users=*

Install Apache Ranger KMS with Ambari

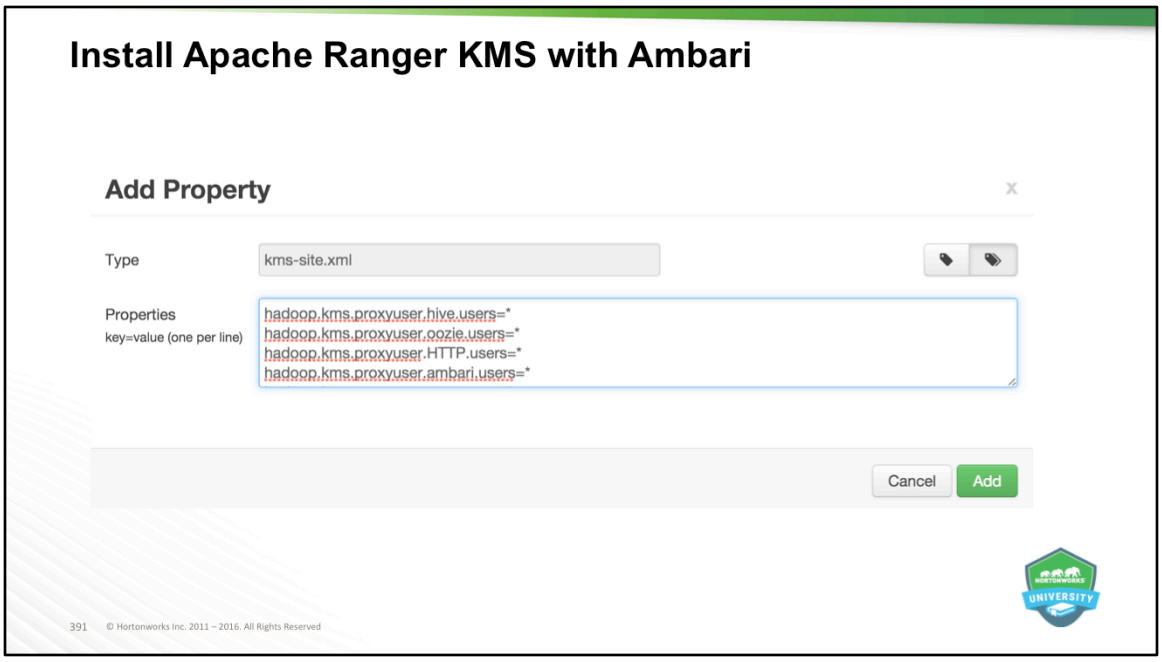


Kms-site:

hadoop.kms.proxyuser.keyadmin.groups=*

hadoop.kms.proxyuser.keyadmin.hosts=*

hadoop.kms.proxyuser.keyadmin.users=*



Kms-site

hadoop.kms.authentication.type=kerberos

hadoop.kms.authentication.kerberos.keytab=/etc/security/keytabs/spnego.service.keytab

hadoop.kms.authentication.kerberos.principal=*

hadoop.kms.proxyuser.hive.users=*

hadoop.kms.proxyuser.oozie.users=*

hadoop.kms.proxyuser.HTTP.users=*

hadoop.kms.proxyuser.ambari.users=*

hadoop.kms.proxyuser.yarn.users=*

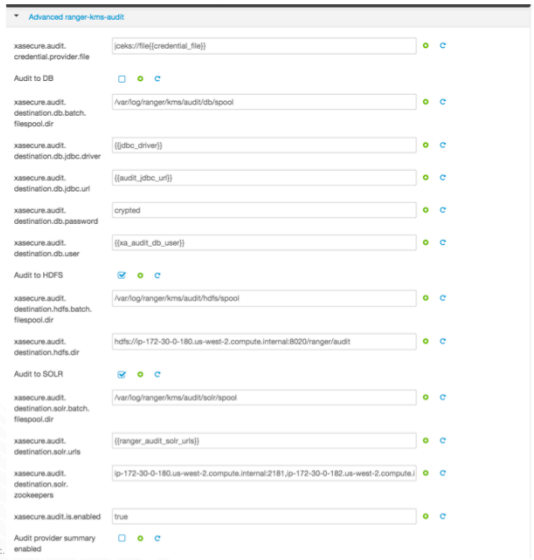
hadoop.kms.proxyuser.hive.hosts=*

hadoop.kms.proxyuser.oozie.hosts=*

hadoop.kms.proxyuser.HTTP.hosts=*

hadoop.kms.proxyuser.ambari.hosts=*

Install Apache Ranger KMS with Ambari

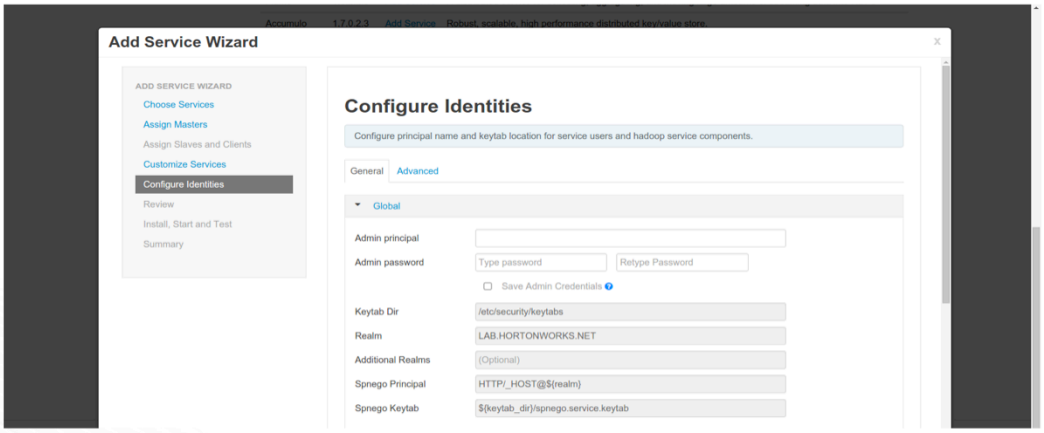


Advanced ranger-kms-audit:

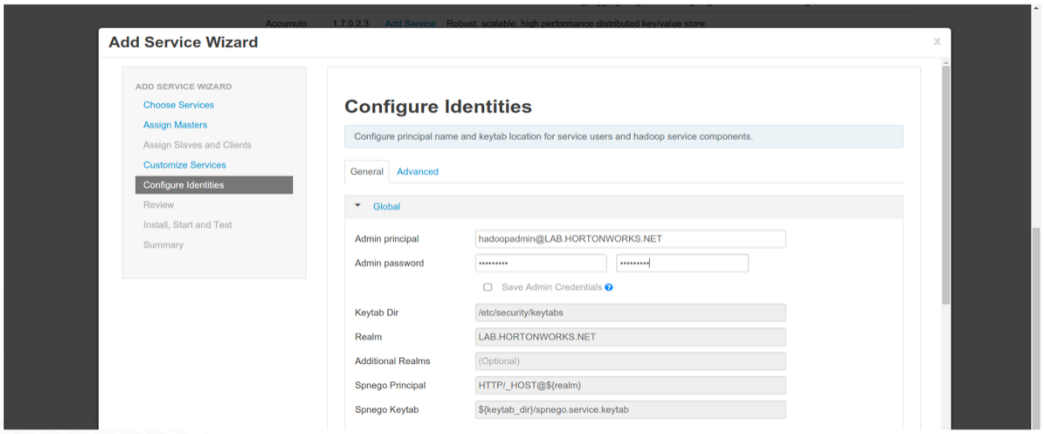
xasecure.audit.destination.hdfs.dir = hdfs://YOUR_NN_INTERNAL_HOSTNAME:8020/ranger/audit

xasecure.audit.destination.solr.zookeepers=YOUR_ZK_QUORUM/ranger_audits

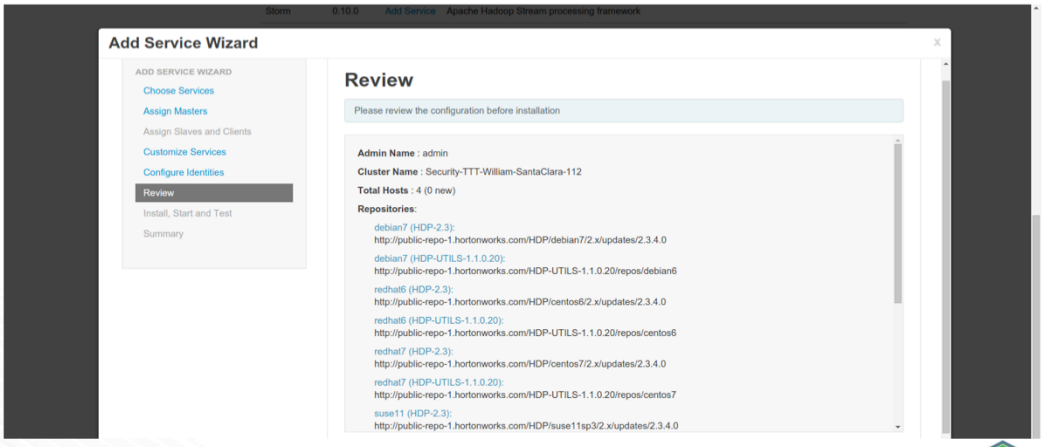
Install Apache Ranger KMS with Ambari



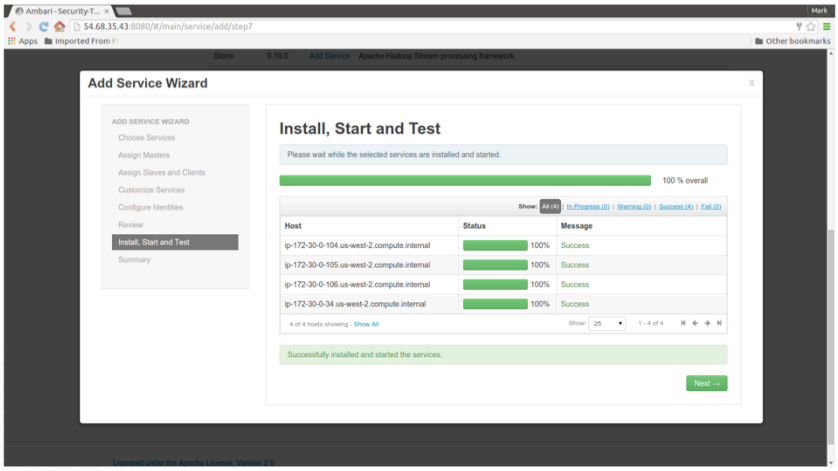
Install Apache Ranger KMS with Ambari



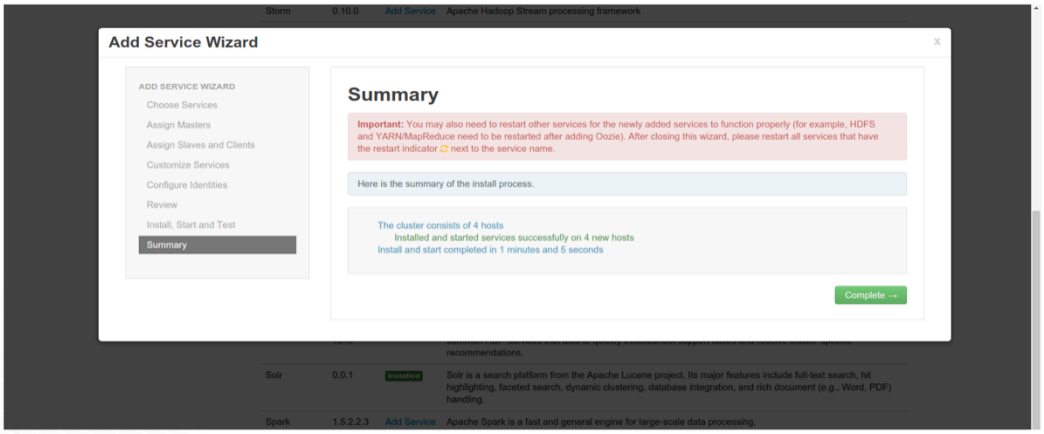
Install Apache Ranger KMS with Ambari



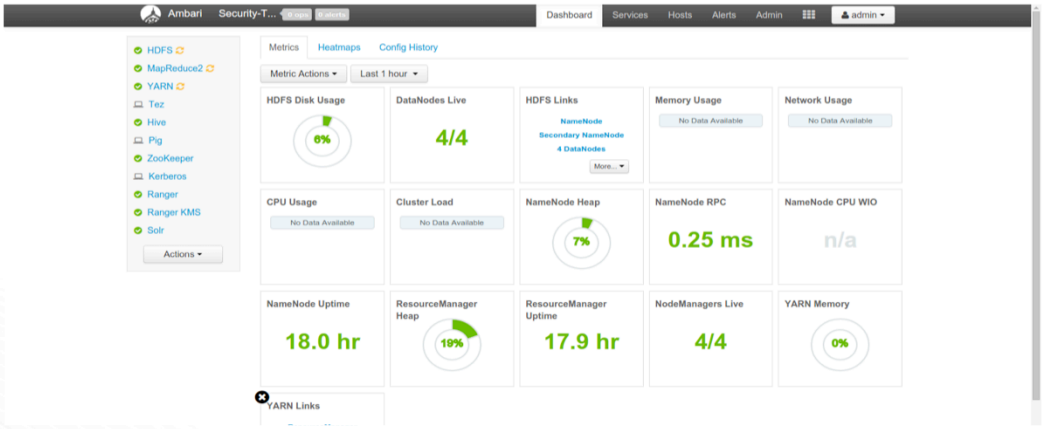
Install Apache Ranger KMS with Ambari



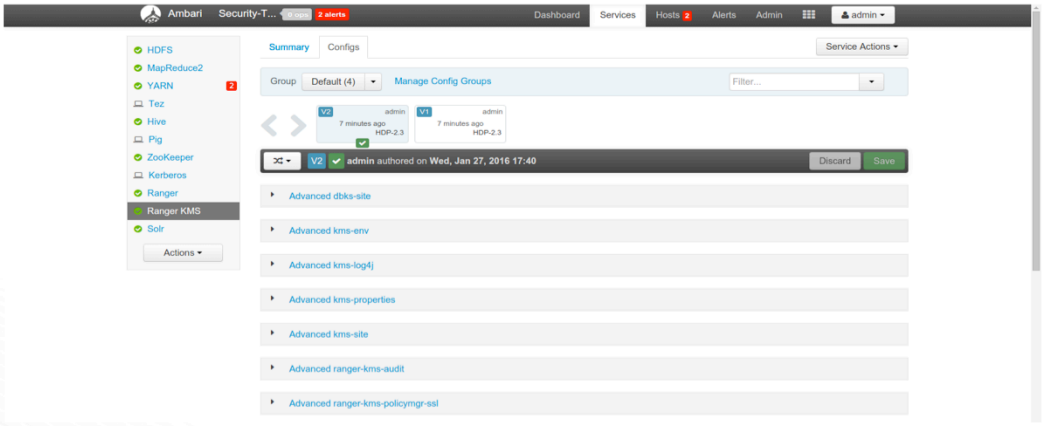
Install Apache Ranger KMS with Ambari



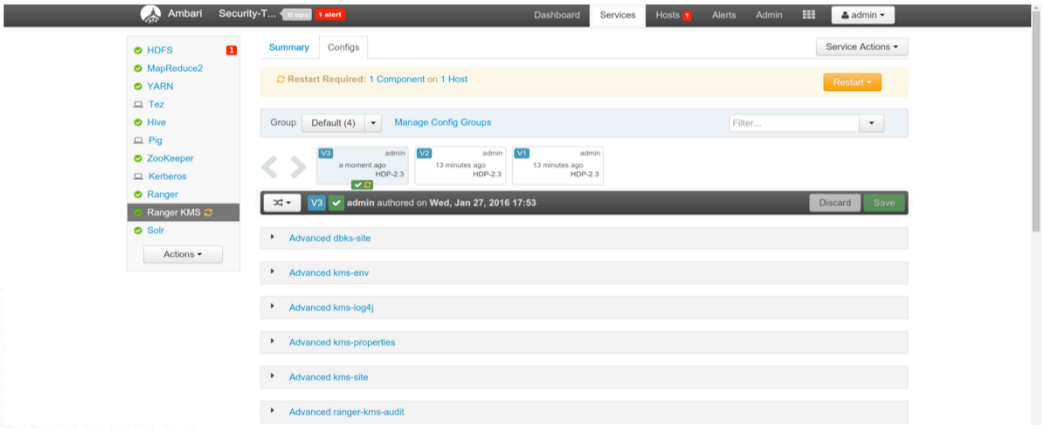
Install Apache Ranger KMS with Ambari



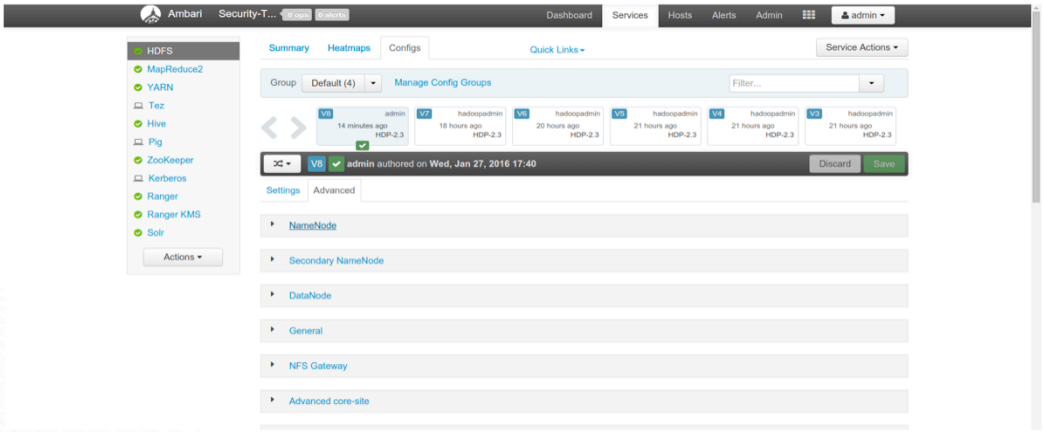
Install Apache Ranger KMS with Ambari



Install Apache Ranger KMS with Ambari



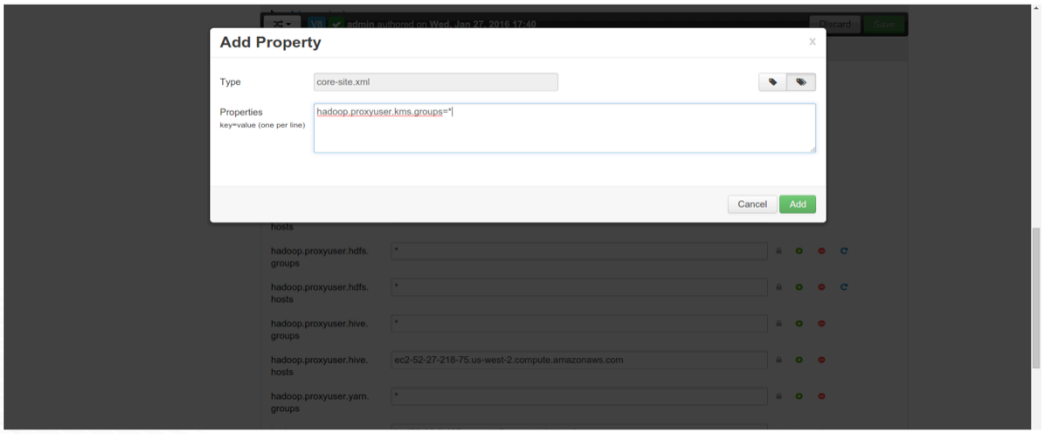
Install Apache Ranger KMS with Ambari



Install Apache Ranger KMS with Ambari

After KMS install, add proxyuser for KMS

Install Apache Ranger KMS with Ambari

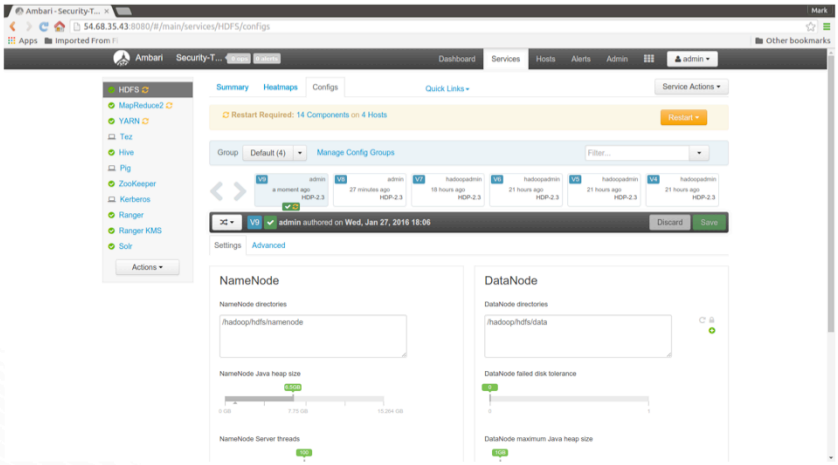


HDFS > Configs > Custom core-site:

hadoop.proxyuser.kms.groups = *

Install Apache Ranger KMS with Ambari

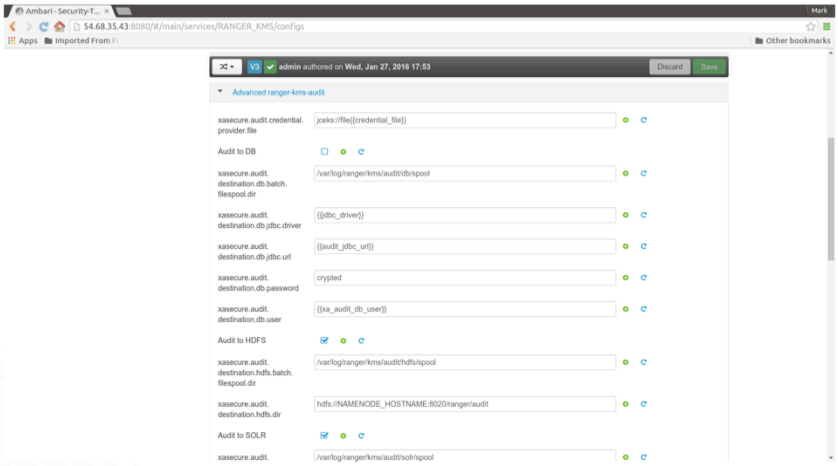
Install Apache Ranger KMS with Ambari



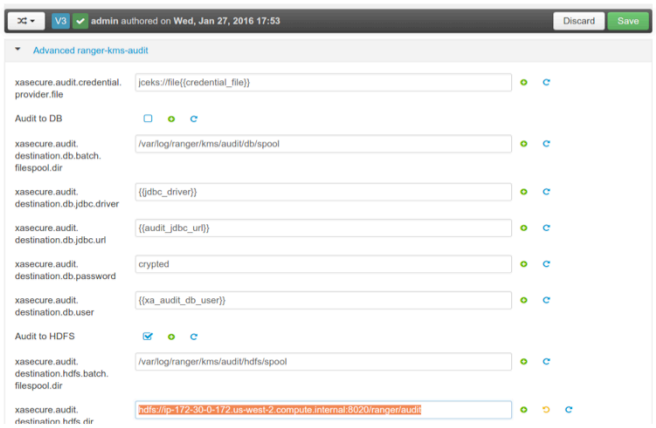
405 © Hortonworks Inc. 2011 – 2016. All Rights Reserved

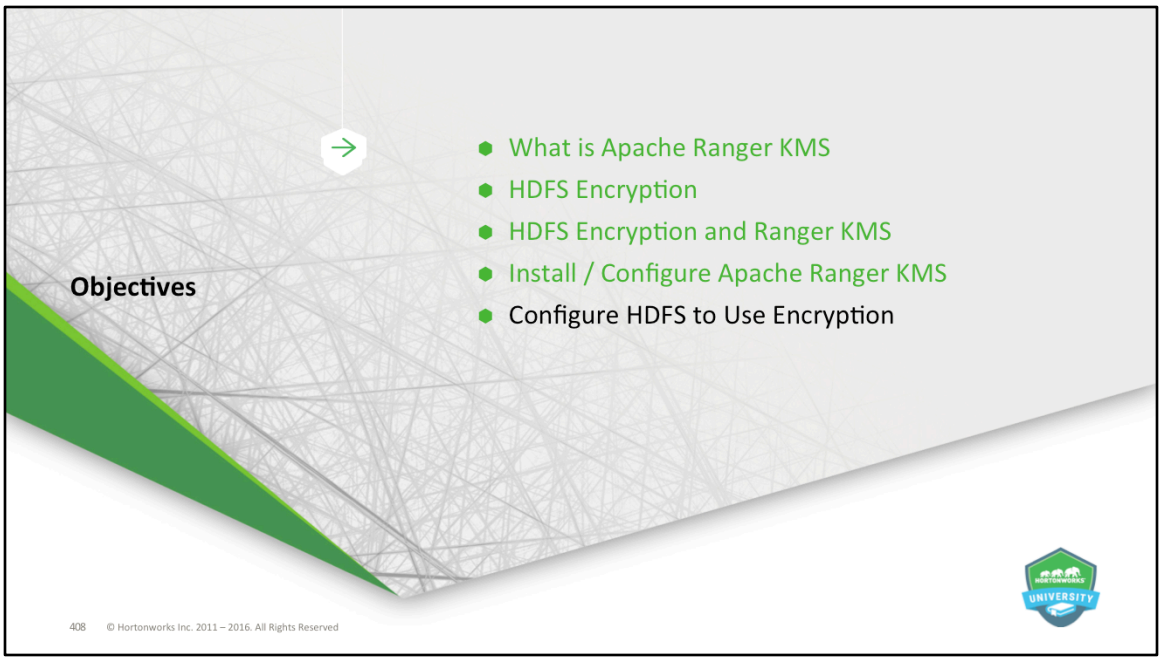
Restart all affected services

Install Apache Ranger KMS with Ambari



Install Apache Ranger KMS with Ambari




Objectives
What is Apache Ranger KMS
HDFS Encryption
HDFS Encryption and Ranger KMS
Install / Configure Apache Ranger KMS
Configure HDFS to Use Encryption
408
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Configure HDFS to Use Encryption

- HDFS administrator can use the hadoop key and hdfs crypto tools to create encryption keys and set up new encryption zones
- Overall workflow follows:
  - Create an HDFS encryption zone key that will be used to encrypt the file-level data encryption key for every file in the encryption zone.
  - This key is stored and managed by Ranger KMS.
  - Create a new HDFS folder.
  - Specify required permissions, owner, and group for the folder.
  - Using the new encryption zone key, designate the folder as an encryption zone.
  - Configure client access. The user associated with the client application needs sufficient permission to access encrypted data.
  - In an encryption zone, the user needs file/directory access (through Posix permissions or Ranger access control), as well as access for certain key operations.

After the Ranger KMS has been set up and the NameNode and HDFS clients have been configured, an HDFS administrator can use the hadoop key and hdfs crypto command-line tools to create encryption keys and set up new encryption zones.

The overall workflow is as follows:

Create an HDFS encryption zone key that will be used to encrypt the file-level data encryption key for every file in the encryption zone. This key is stored and managed by Ranger KMS.

Create a new HDFS folder. Specify required permissions, owner, and group for the folder.

Using the new encryption zone key, designate the folder as an encryption zone.

Configure client access. The user associated with the client application needs

Configure HDFS to Use Encryption

- Overall workflow follows: - Continued
- Set permissions on which users/groups have access to key via Ranger policy
- After permissions are set, Java API clients and HDFS applications with sufficient HDFS and Ranger KMS access privileges can write and read to/from files in the encryption zone.

After the Ranger KMS has been set up and the NameNode and HDFS clients have been configured, an HDFS administrator can use the hadoop key and hdfs crypto command-line tools to create encryption keys and set up new encryption zones.

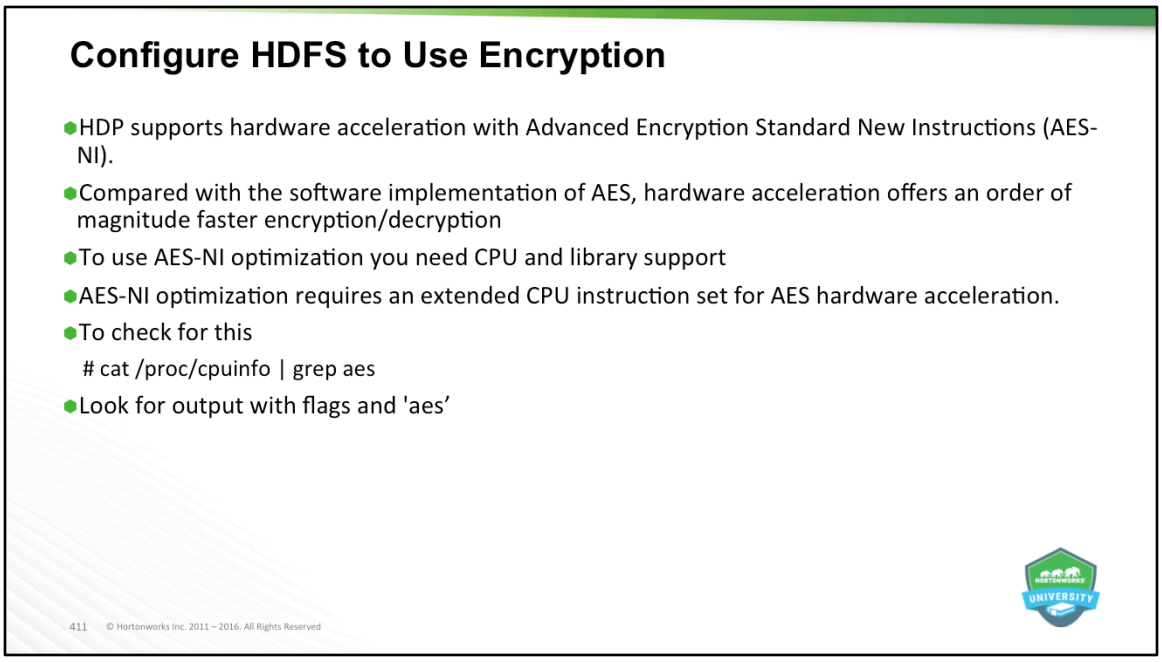
The overall workflow is as follows:

Create an HDFS encryption zone key that will be used to encrypt the file-level data encryption key for every file in the encryption zone. This key is stored and managed by Ranger KMS.

Create a new HDFS folder. Specify required permissions, owner, and group for the folder.

Using the new encryption zone key, designate the folder as an encryption zone.

Configure client access. The user associated with the client application needs



HDP supports hardware acceleration with Advanced Encryption Standard New Instructions (AES-NI). Compared with the software implementation of AES, hardware acceleration offers an order of magnitude faster encryption/decryption.

To use AES-NI optimization you need CPU and library support,

CPU Support for AES-NI optimization

AES-NI optimization requires an extended CPU instruction set for AES hardware acceleration.

There are several ways to check for this; for example:

$ cat /proc/cpuinfo | grep aes

Configure HDFS to Use Encryption

- OpenSSL 1.0.1e version of the libcrypto.so library supports hardware acceleration
- A version of the libcrypto.so libary with AES-NI support must be installed on HDFS cluster nodes and MapReduce client hosts
- The following instructions describe how to install and configure the libcrypto.so library.
- RHEL/CentOS 6.5 or later
- On HDP cluster nodes, the installed version of libcrypto.so supports AES-NI, but you will need to make sure that the symbolic link exists:

  $ sudo ln -s /usr/lib64/libcrypto.so.1.0.1e /usr/lib64/libcrypto.so
- On MapReduce client hosts, install the openssl-devel package:

  $ sudo yum install openssl-devel

HDP supports hardware acceleration with Advanced Encryption Standard New Instructions (AES-NI). Compared with the software implementation of AES, hardware acceleration offers an order of magnitude faster encryption/decryption.

To use AES-NI optimization you need CPU and library support,

CPU Support for AES-NI optimization

AES-NI optimization requires an extended CPU instruction set for AES hardware acceleration.

There are several ways to check for this; for example:

$ cat /proc/cpuinfo | grep aes

Configure HDFS to Use Encryption

- Verify client host is ready to use the AES-NI instruction set optimization for HDFS encryption

  # hadoop checknative
- Output similar to the following:

```
15/08/12 13:48:39 INFO bzip2.Bzip2Factory: Successfully loaded & initialized native-bzip2 library system-native
14/12/12 13:48:39 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
Native library checking:
hadoop:  true /usr/lib/hadoop/lib/native/libhadoop.so.1.0.0
zlib:    true /lib64/libz.so.1
snappy:  true /usr/lib64/libsnappy.so.1
lz4:     true revision:99
bzip2:   true /lib64/libbz2.so.1
openssl: true /usr/lib64/libcrypto.so
```

HDP supports hardware acceleration with Advanced Encryption Standard New Instructions (AES-NI). Compared with the software implementation of AES, hardware acceleration offers an order of magnitude faster encryption/decryption.

To use AES-NI optimization you need CPU and library support,

CPU Support for AES-NI optimization

AES-NI optimization requires an extended CPU instruction set for AES hardware acceleration.

There are several ways to check for this; for example:

$ cat /proc/cpuinfo | grep aes

Configure HDFS to Use Encryption

- Verify client host is ready to use the AES-NI instruction set optimization for HDFS encryption
- True in the openssl row - Hadoop has detected the right version of libcrypto.so and optimization will work.
- False in this row - do not have the correct version.

HDP supports hardware acceleration with Advanced Encryption Standard New Instructions (AES-NI). Compared with the software implementation of AES, hardware acceleration offers an order of magnitude faster encryption/decryption.

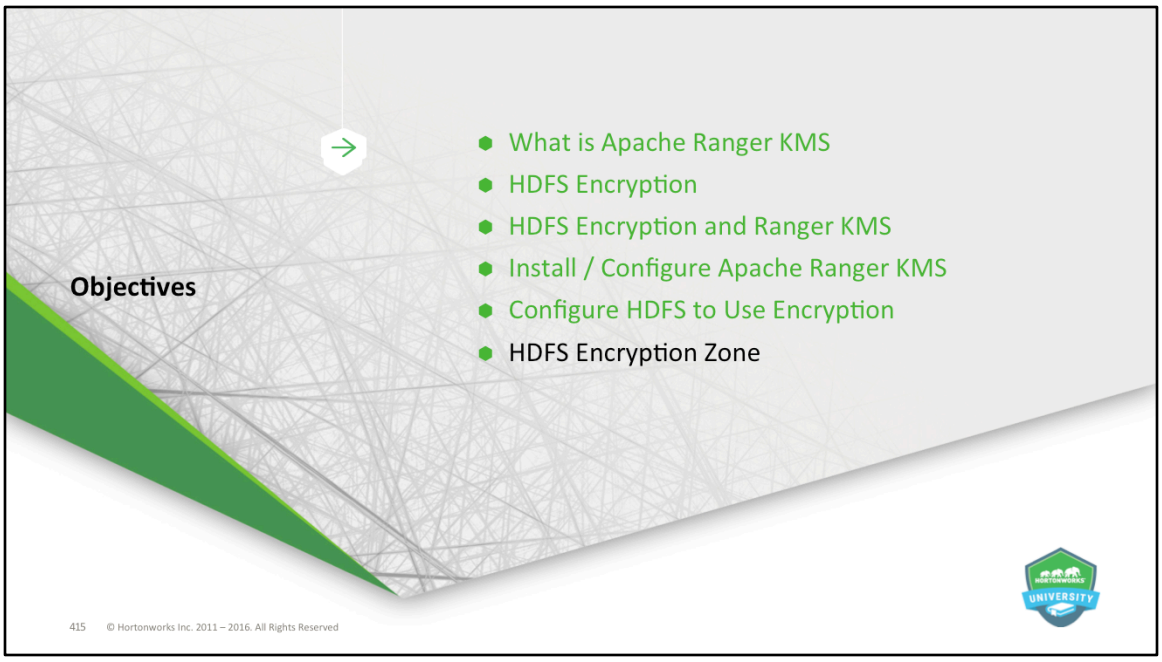
To use AES-NI optimization you need CPU and library support,

CPU Support for AES-NI optimization

AES-NI optimization requires an extended CPU instruction set for AES hardware acceleration.

There are several ways to check for this; for example:

$ cat /proc/cpuinfo | grep aes


Objectives
What is Apache Ranger KMS
HDFS Encryption
HDFS Encryption and Ranger KMS
Install / Configure Apache Ranger KMS
Configure HDFS to Use Encryption
HDFS Encryption Zone
415
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

HDFS Encryption Zone – Create Encryption Key

- Create a "master" encryption key for the new encryption zone.
- Each key will be specific to an encryption zone.
- Ranger supports AES/CTR/NoPadding as the cipher suite.
- Key size can be 128 or 256 bits.
- Recommendation: create a new superuser for key management.
- In the following examples**, superuser encrypter** creates the key.
- This separates the data access role from the encryption role, strengthening security.
- Create an Encryption Key using Ranger KMS (Recommended)
- In the Ranger Web UI screen:
  - Choose the Encryption tab at the top of the screen.
  - Select the KMS service from the drop-down list.

Create an Encryption Key

Create a "master" encryption key for the new encryption zone. Each key will be specific to an encryption zone.

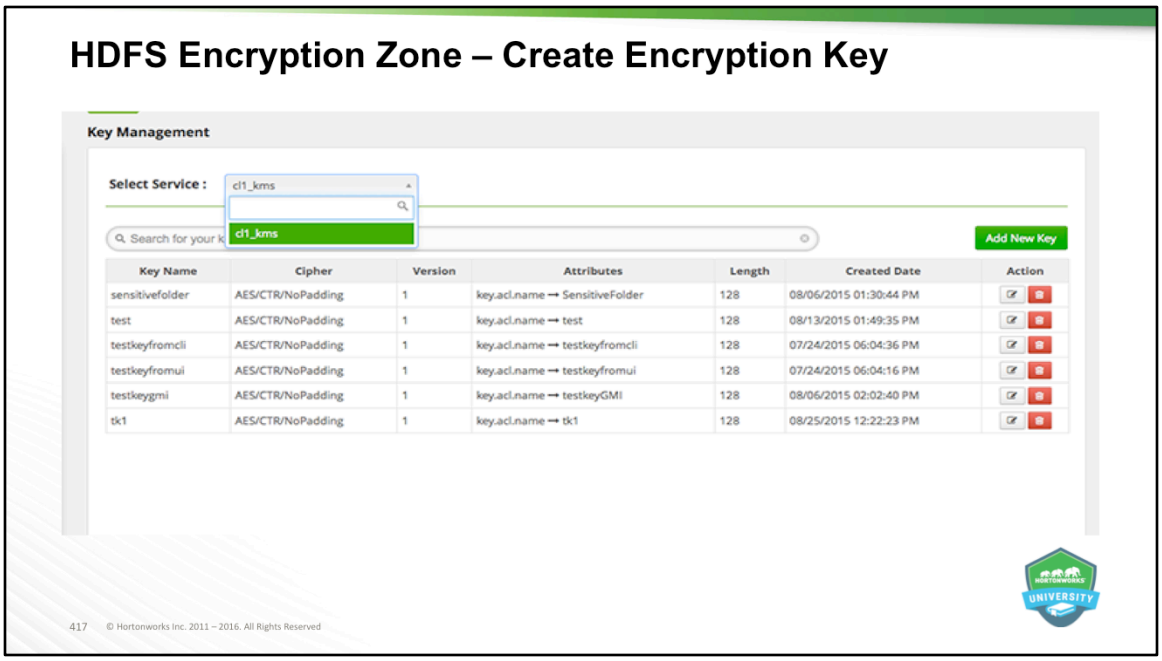
Ranger supports AES/CTR/NoPadding as the cipher suite. (The associated property is listed under HDFS -> Configs in the Advanced hdfs-site list.)

Key size can be 128 or 256 bits.

Recommendation: create a new superuser for key management. In the following examples, superuser encr creates the key. This separates the data access role from the encryption role, strengthening security.

Create an Encryption Key using Ranger KMS (Recommended)

HDFS Encryption Zone – Create Encryption Key



Create an Encryption Key

Create a "master" encryption key for the new encryption zone. Each key will be specific to an encryption zone.

Ranger supports AES/CTR/NoPadding as the cipher suite. (The associated property is listed under HDFS -> Configs in the Advanced hdfs-site list.)

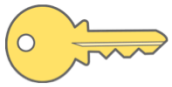
Key size can be 128 or 256 bits.

Recommendation: create a new superuser for key management. In the following examples, superuser encr creates the key. This separates the data access role from the encryption role, strengthening security.

Create an Encryption Key using Ranger KMS (Recommended)

HDFS Encryption Zone – Create Encryption Key

- To create a new key:
  - Click on "Add New Key":
  - Add a valid key name.
  - Select the cipher name.
    - Ranger supports AES/CTR/NoPadding as the cipher suite.
  - Specify the key length, 128 or 256 bits.
  - Add other attributes as needed, and then save the key.



EZ Key

To create a new key:

Click on "Add New Key":

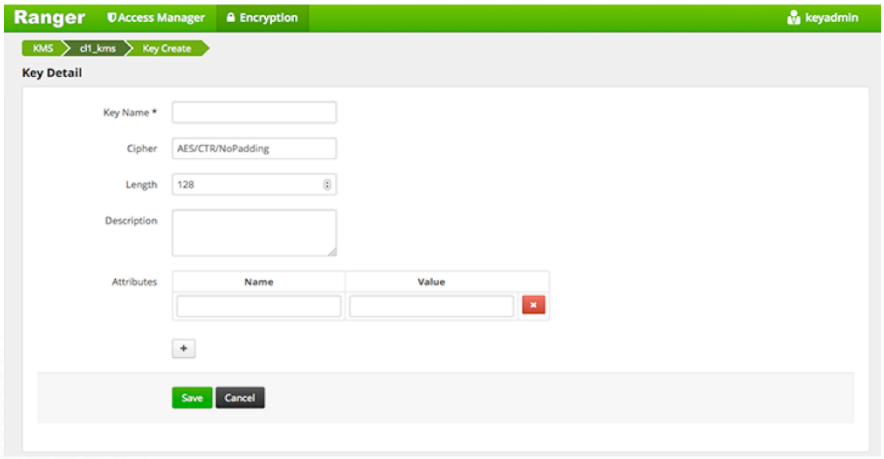
Add a valid key name.

Select the cipher name. Ranger supports AES/CTR/NoPadding as the cipher suite.

Specify the key length, 128 or 256 bits.

Add other attributes as needed, and then save the key.

HDFS Encryption Zone – Create Encryption Key



To create a new key:

Click on "Add New Key":

Add a valid key name.

Select the cipher name. Ranger supports AES/CTR/NoPadding as the cipher suite.

Specify the key length, 128 or 256 bits.

Add other attributes as needed, and then save the key.

HDFS Encryption Zone – Create Encryption Zone

- Each encryption zone must be defined using an empty directory and an existing encryption key.
- An encryption zone cannot be created on top of a directory that already contains data.
- Recommendation: use one unique key for each encryption zone
- Use the crypto "createZone" command to create a new encryption zone:

  # crypto -createZone -keyName <keyName> -path <path>
- where:

  -keyName: specifies the name of the key to use for the encryption zone

  -path: specifies the path of the encryption zone to be created, must be empty directory

Each encryption zone must be defined using an empty directory and an existing encryption key. An encryption zone cannot be created on top of a directory that already contains data.

Recommendation: use one unique key for each encryption zone.

Use the crypto createZone command to create a new encryption zone. The syntax is:

-createZone -keyName <keyName> -path <path>

where:

-keyName: specifies the name of the key to use for the encryption zone.

-path specifies the path of the encryption zone to be created. It must be an empty

HDFS Encryption Zone – Create Encryption Zone

- Service account "hdfs" can create zones, but cannot write data unless the account has sufficient permission
- Recommendation:
  - Define a separate user account for the HDFS administrator
  - Do not provide access to keys for this user in Ranger KMS
  - For example, to create HDFS admin user called **encrytper**:
    - In Advanced hdfs-site: dfs.cluster.administrators = hdfs,encrypter
    - In Advanced dbks-site: hadoop.kms.blacklist.DECRYPT_EEK = hdfs,encrypter

Each encryption zone must be defined using an empty directory and an existing encryption key. An encryption zone cannot be created on top of a directory that already contains data.

Recommendation: use one unique key for each encryption zone.

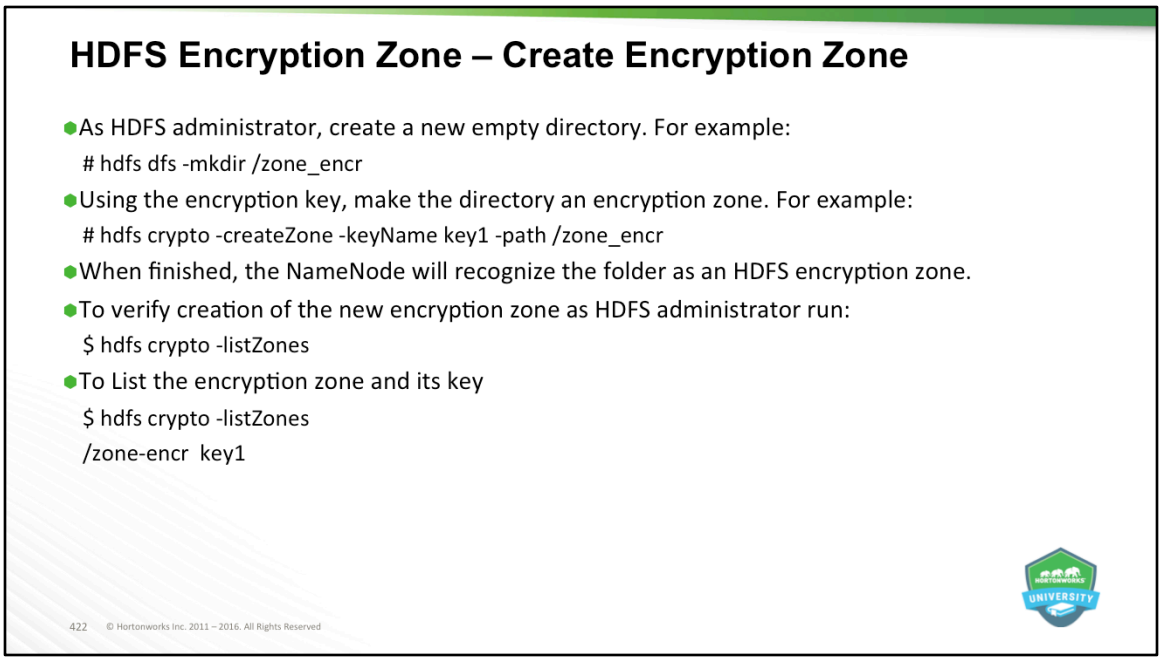
Use the crypto createZone command to create a new encryption zone. The syntax is:

-createZone -keyName <keyName> -path <path>

where:

-keyName: specifies the name of the key to use for the encryption zone.

-path specifies the path of the encryption zone to be created. It must be an empty



As HDFS administrator, create a new empty directory. For example:

hdfs dfs -mkdir /zone_encr

Using the encryption key, make the directory an encryption zone. For example:

hdfs crypto -createZone -keyName key1 -path /zone_encr

When finished, the NameNode will recognize the folder as an HDFS encryption zone.

To verify creation of the new encryption zone, run the crypto -listZones command as an HDFS administrator:

-listZones

HDFS Encryption Zone – Create Encryption Zone

- Property in hdfs-default.xml file causes listZone requests to be batched:
  - dfs.namenode.list.encryption.zones.num.responses
- Improves NameNode performance
- Specifies the maximum number of zones that will be returned in a batch
- Default is 100

- To remove an encryption zone, delete the root directory of the zone.
  $hdfs dfs -rm -R /zone_encr

As HDFS administrator, create a new empty directory. For example:

hdfs dfs -mkdir /zone_encr

Using the encryption key, make the directory an encryption zone. For example:

hdfs crypto -createZone -keyName key1 -path /zone_encr

When finished, the NameNode will recognize the folder as an HDFS encryption zone.

To verify creation of the new encryption zone, run the crypto -listZones command as an HDFS administrator:

-listZones

HDFS Encryption Zone – Read/Write Encryption Zone

- HDFS Clients/Application with Sufficient HDFS/Ranger KMS Permission can:
  - Read Files From EZ
  - Write File To EZ
- Client Write Process:
  - Client Writes to Encryption Zone
  - NameNode Checks Clients Write Access Permissions
  - Asks Ranger KMS to Create File-Level Key, Encrypted with Encryption Zone Master Key
  - NameNode Store File-Level Encrypted Data Encryption Key (EDEK) as File's Metadata
  - Returns EDEK to Client
  - Client Asks Ranger KMS to Decode EDEK to DEK
  - Client Uses DEK to Write Encrypted Data
  - Ranger Check Permission for User Before Decrypting EDEK and Producing DEK for Client

Clients and HDFS applications with sufficient HDFS and Ranger KMS permissions can read and write files from/to an encryption zone.

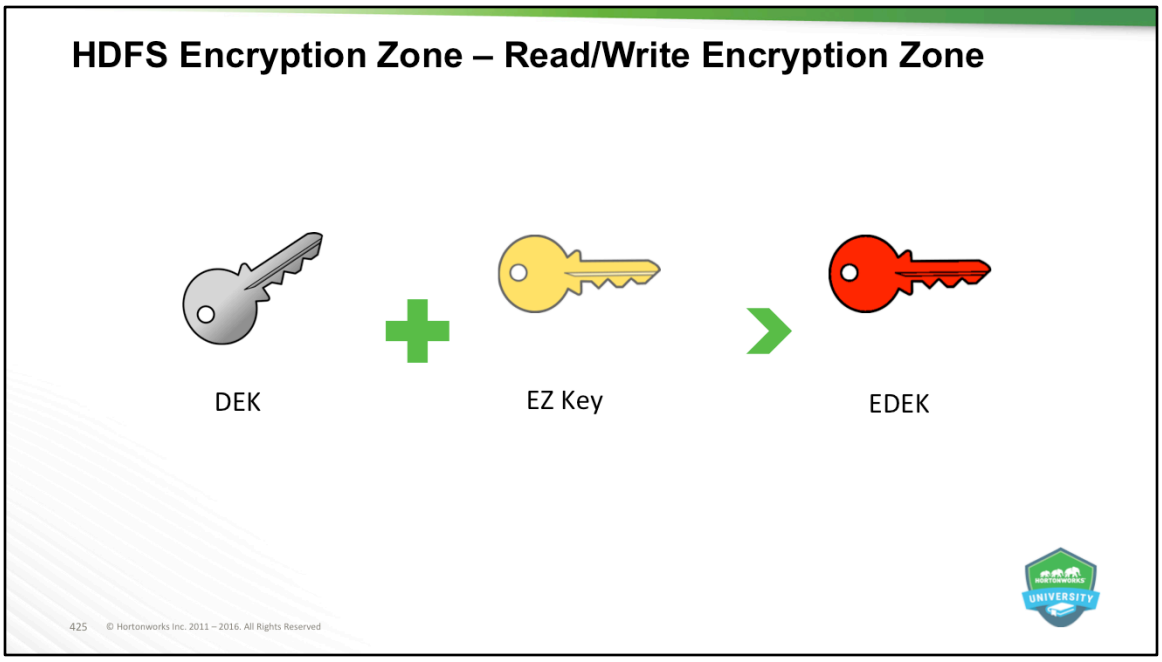
Overview of the client write process:

The client writes to the encryption zone.

The NameNode checks to make sure that the client has sufficient write access permissions. If so, the NameNode asks Ranger KMS to create a file-level key, encrypted with the encryption zone master key.

The Namenode stores the file-level encrypted data encryption key (EDEK) generated by Ranger KMS as part of the file's metadata, and returns the EDEK to the client.

The client asks Ranger KMS to decode the EDEK (to DEK), and uses the DEK to write encrypted data. Ranger KMS checks for permissions for the user before decrypting



Clients and HDFS applications with sufficient HDFS and Ranger KMS permissions can read and write files from/to an encryption zone.

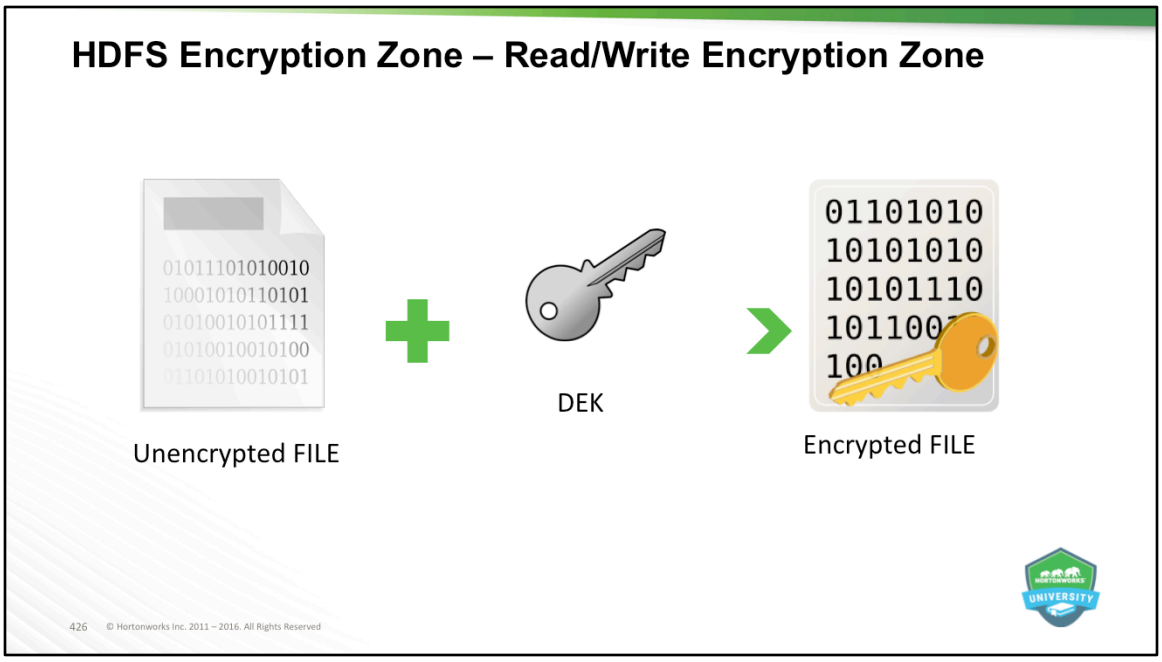
Overview of the client write process:

The client writes to the encryption zone.

The NameNode checks to make sure that the client has sufficient write access permissions. If so, the NameNode asks Ranger KMS to create a file-level key, encrypted with the encryption zone master key.

The Namenode stores the file-level encrypted data encryption key (EDEK) generated by Ranger KMS as part of the file's metadata, and returns the EDEK to the client.

The client asks Ranger KMS to decode the EDEK (to DEK), and uses the DEK to write encrypted data. Ranger KMS checks for permissions for the user before decrypting



Clients and HDFS applications with sufficient HDFS and Ranger KMS permissions can read and write files from/to an encryption zone.

Overview of the client write process:

The client writes to the encryption zone.

The NameNode checks to make sure that the client has sufficient write access permissions. If so, the NameNode asks Ranger KMS to create a file-level key, encrypted with the encryption zone master key.

The Namenode stores the file-level encrypted data encryption key (EDEK) generated by Ranger KMS as part of the file's metadata, and returns the EDEK to the client.

The client asks Ranger KMS to decode the EDEK (to DEK), and uses the DEK to write encrypted data. Ranger KMS checks for permissions for the user before decrypting

HDFS Encryption Zone – Read/Write Encryption Zone

- Client Read Process:
  - Client Issues Read Request for File in Encryption Zone
  - NameNode Checks Clients Read Access Permissions
  - Returns File's EDEK and Encryption Zone Key Version Used to Encrypt EDEK
  - Client Asks Ranger KMS to Decrypt EDEK
  - Ranger KMS Checks for Permission to Decrypt EDEK for End User
  - Ranger KMS Decrypts and Returns Unencrypted Data Encryption Key (DEK)
  - Client Uses DEK to Decrypt and Read the Files

Clients and HDFS applications with sufficient HDFS and Ranger KMS permissions can read and write files from/to an encryption zone.

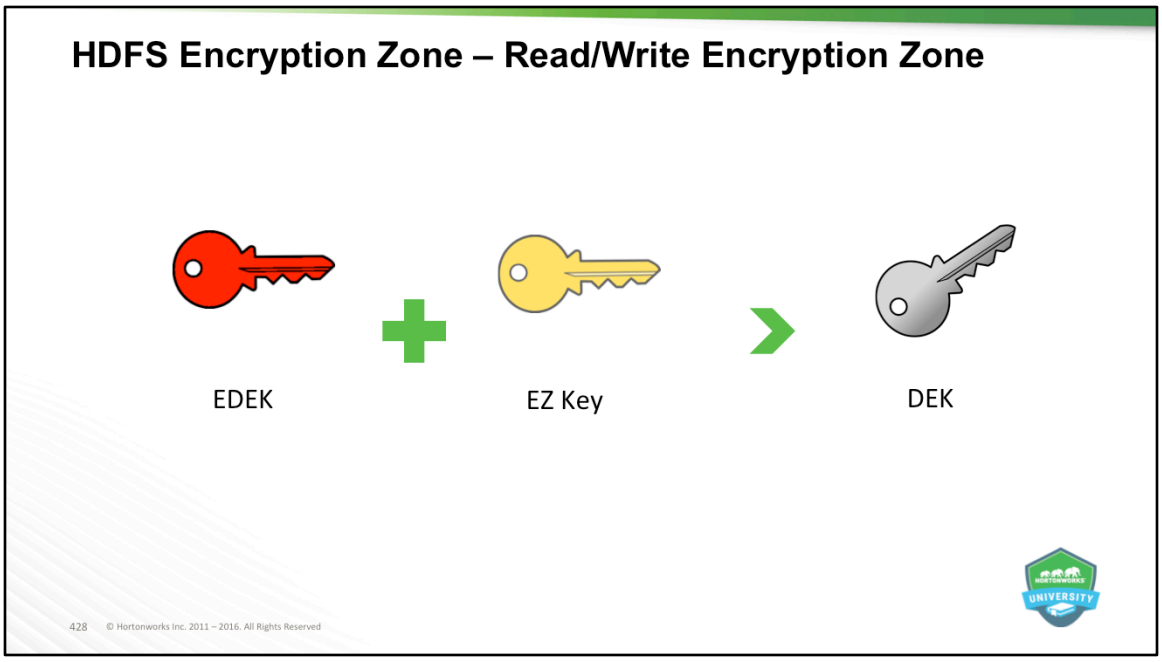
Overview of the client write process:

The client writes to the encryption zone.

The NameNode checks to make sure that the client has sufficient write access permissions. If so, the NameNode asks Ranger KMS to create a file-level key, encrypted with the encryption zone master key.

The Namenode stores the file-level encrypted data encryption key (EDEK) generated by Ranger KMS as part of the file's metadata, and returns the EDEK to the client.

The client asks Ranger KMS to decode the EDEK (to DEK), and uses the DEK to write encrypted data. Ranger KMS checks for permissions for the user before decrypting



Clients and HDFS applications with sufficient HDFS and Ranger KMS permissions can read and write files from/to an encryption zone.

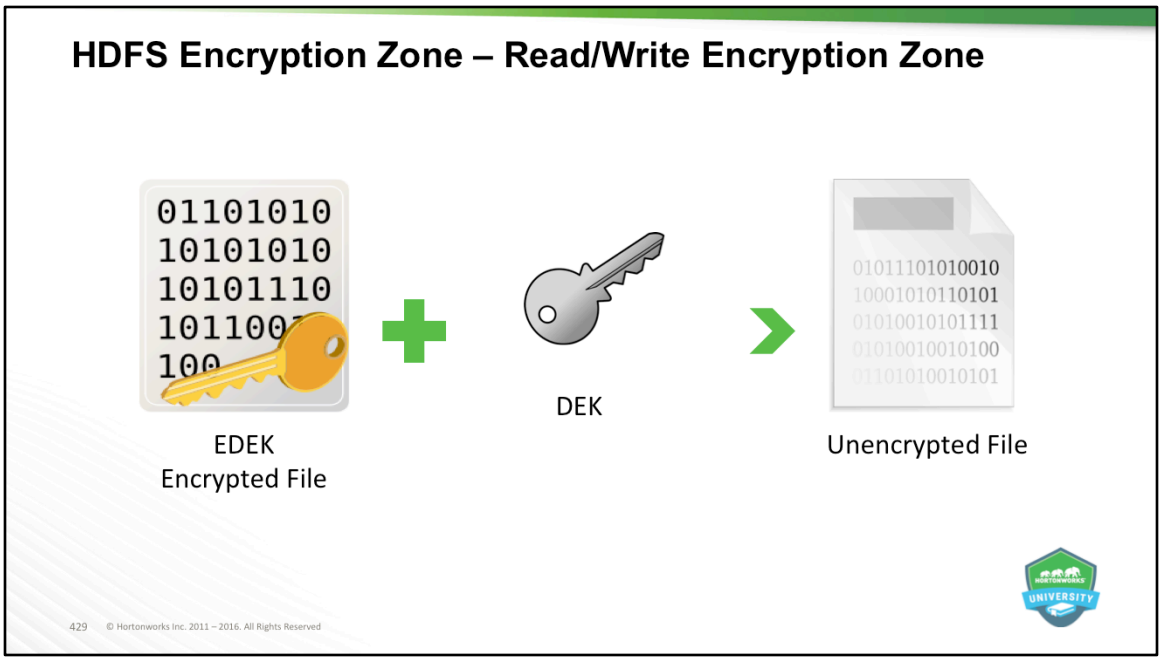
Overview of the client write process:

The client writes to the encryption zone.

The NameNode checks to make sure that the client has sufficient write access permissions. If so, the NameNode asks Ranger KMS to create a file-level key, encrypted with the encryption zone master key.

The Namenode stores the file-level encrypted data encryption key (EDEK) generated by Ranger KMS as part of the file's metadata, and returns the EDEK to the client.

The client asks Ranger KMS to decode the EDEK (to DEK), and uses the DEK to write encrypted data. Ranger KMS checks for permissions for the user before decrypting



Clients and HDFS applications with sufficient HDFS and Ranger KMS permissions can read and write files from/to an encryption zone.

Overview of the client write process:

The client writes to the encryption zone.

The NameNode checks to make sure that the client has sufficient write access permissions. If so, the NameNode asks Ranger KMS to create a file-level key, encrypted with the encryption zone master key.

The Namenode stores the file-level encrypted data encryption key (EDEK) generated by Ranger KMS as part of the file's metadata, and returns the EDEK to the client.

The client asks Ranger KMS to decode the EDEK (to DEK), and uses the DEK to write encrypted data. Ranger KMS checks for permissions for the user before decrypting

Data Protection – HDFS Encryption

In summary:

- To convert DEK to EDEK (or vice versa): Need EZ Key
- To encrypt or decrypt a file: Need DEK
- EZ key and DEK stored in KMS
- EDEK stored in NameNode
- Notice that with HDFS encryption enabled:
  - Client needs to interact with *both* HDFS (for data/EDEK) as well as KMS (to decrypt EDEK) to get access to the DEK (required to read/write files)
  - There are two levels of checks - whether the client has permission to access:
    - the HDFS dir/file – check performed by NameNode
    - the encryption zone key version – check performed by KMS
- Recommended: create a separate HDFS admin user for Key Management.

Knowledge Check

Knowledge Check

1. True/False - Apache Ranger KMS is not based on Hadoop KMS?
2. True/False - Apache Ranger KMS supports encryption with “Data in Motion”?
3. Data at Rest implements _____ Encryption?
4. True/False - HDFS has direct access to unencrypted data and keys?
5. What two items must exists to create an encryption zone?
6. True/False - It is recommended that one unique key be used for each encryption zone?

1. False
2. False
3. End-to-End
4. False
5. Empty Directory and Encryption Key
6. True

Summary

Summary

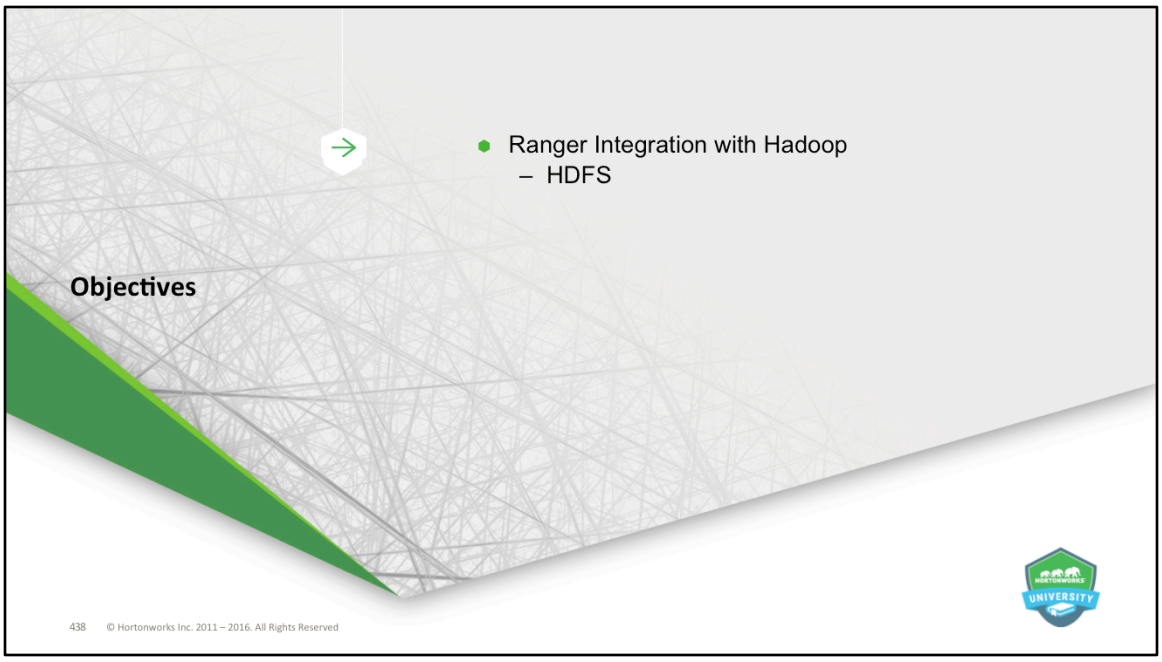
- Apache Range KMS is a Open Source, Scalable Cryptographic Key Management Services
- Based in Hadoop KMS Originally Developed by the Apache Community
- Supports HDFS “Data at Rest” Encryption
- HDFS Encryption is a Form of Data Security
- Data At Rest Encryption Implements End-to-End Encryption of Data Read From/Written To HDFS
- HDFS Does Not Have Access to Unencrypted Data/Keys
- Each encryption zone must be defined using an empty directory and an existing encryption key.
- An encryption zone cannot be created on top of a directory that already contains data.
- Use one unique key for each encryption zone

Lab: Ranger KMS/Data Encryption Setup


Secure Access With Ranger
HORTONWORKS
UNIVERSITY

Lesson Objectives

- Describe Apache Ranger Plugin Integration with
  - HDFS
  - Hive
  - HBase
  - Knox
  - Storm


Ranger Integration with Hadoop
– HDFS
Objectives
438
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Ranger Plugin - HDFS Integration

- HDFS Permissions
- POSIX like permission model (owner/group for files and folders)
- ACL's for fine-grained permissions (for specific set of users/groups)
  - hdfs dfs -getfacl [-R] <path>
  - hdfs dfs -setfacl [-R] [options] <path>

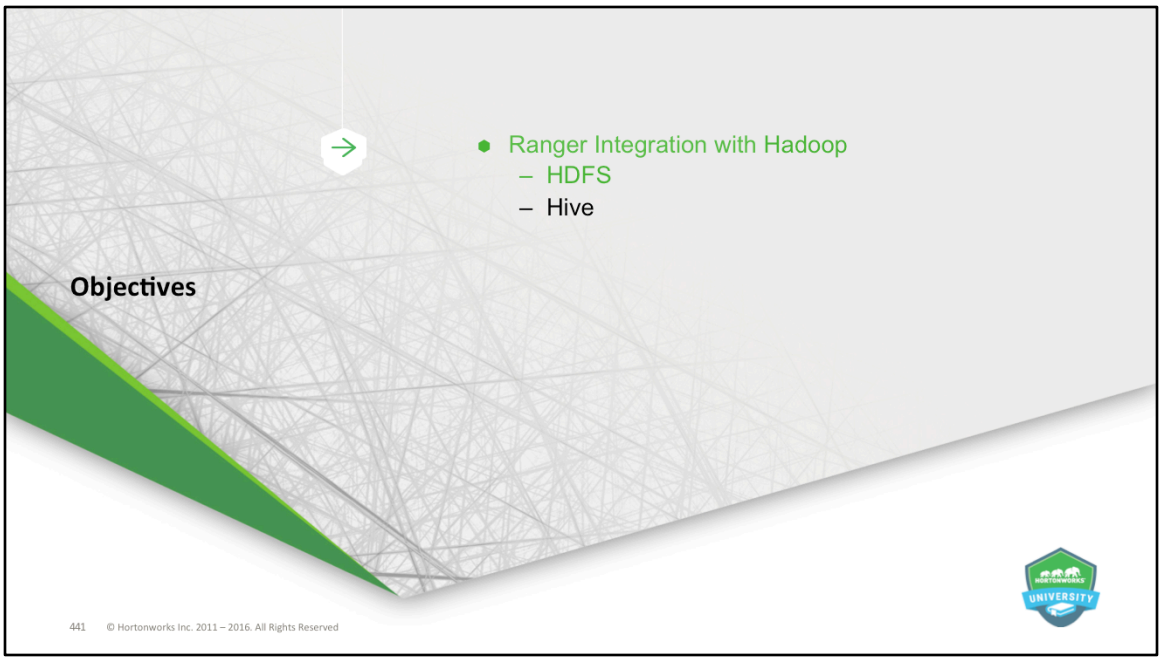
- dfs.permissions.enabled must be set to true

Ranger Plugin - HDFS Integration

- Ranger Plugin acts as an authorizer within Namenode.
- Need to install on all namenodes (in HA environment)
- User can define policies on files and folders
  - Use of wildcard to define policies (/finance/audit_*)
  - Read, Write, Execute permissions are allowed
- Plugin evaluates HDFS requests and provide access
- No specific ranger policy exists - HDFS ACLs are used as fallback
- Kerborized HDFS Access how-to:
  - Just kinit and access as usual (assuming Ranger policy/ACL already in place)

  $ kinit someuser

  $ hdfs fs –ls /tmp


Ranger Integration with Hadoop
– HDFS
– Hive
Objectives
441
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Ranger Plugin - Hive Integration

- Hive facilitates querying and managing large datasets in distributed storage
- HiveServer2 (HS2) is a server interface to Hive
- HiveServer2 supports Access Control similar to relational database model
  - SELECT/UPDATE/DELETE permissions on tables/columns
  - Permission defined for USERS/ROLES
  - Also provides a pluggable authorizer model

Ranger Plugin - Hive Integration

- Ranger Plugin acts as authorization provider for HiveServer2
- User can define policies on databases, tables/view, columns and UDFs
- Select, Update, Create, Drop, Alter, Index, Lock Permissions allowed
- Plugin evaluates Hive requests and grants/denies access based on the policies
- Creates necessary audit logs based on audit
- Specific Ranger policy must exist for gaining access

Ranger Plugin - Hive Integration

- When the user executes GRANT/REVOKE statements
  - Hive plugin will creates/deletes necessary Ranger Policies
  - Provided the user has permission to create/delete policies
- GRANT SELECT, UPDATE, ALTER ON TABLE test_data TO USER guest;
  - will create a new Ranger policy
- REVOKE UPDATE, ALTER ON TABLE test_data FROM USER guest;
  - will update/delete existing Ranger policy
- ROLE specified in the GRANT/REVOKE statements will be mapped to corresponding GROUPS from your corporate directories
- Optionally, can disable GRANT/REVOKE commands
  - Force authorization policies management solely via Ranger Policy Admin

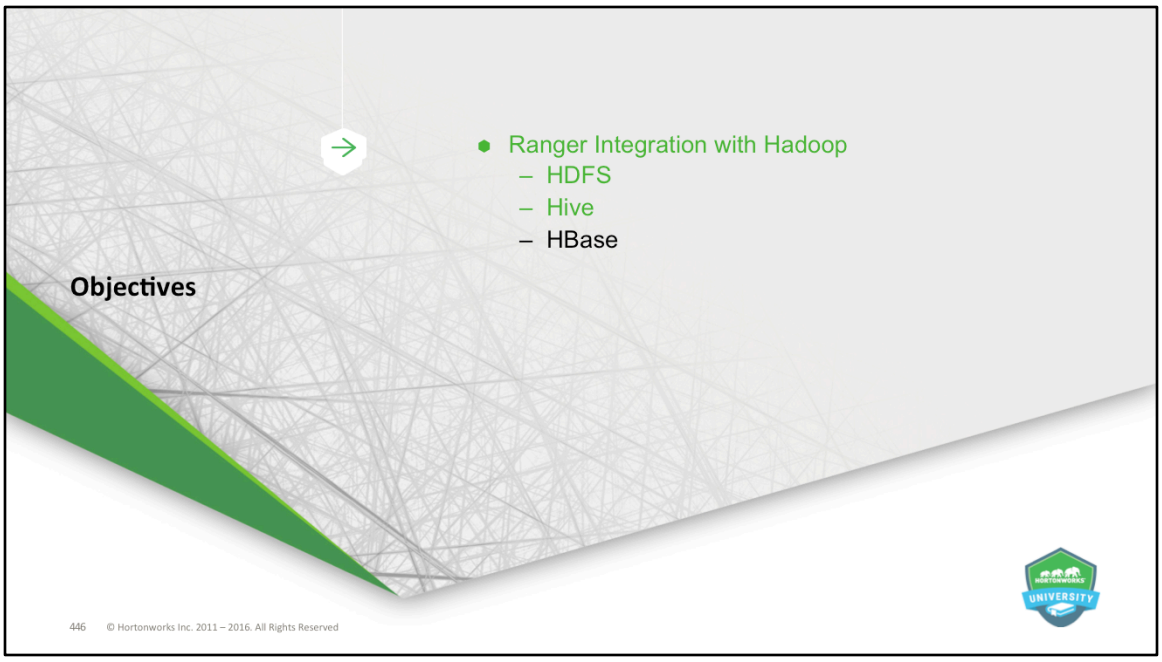
Ranger Plugin - Hive Integration

- Hive Access How-to (using Beeline or another Hive JDBC client)
- Assuming Ranger policy/ACL already in place:
  - Access unsecure Hive

    beeline -u "jdbc:hive2://localhost:10000/default"
  - Access secure Hive - Binary transport mode (the default)
    - Note the update to include hive principal

    beeline -u "jdbc:hive2://localhost:10000/default;principal=hive/$(hostname -f)@HORTONWORKS.COM"
  - Access secure Hive – HTTP transport mode
    - Note the update to use HTTP and the need to provide the kerberos principal.

    beeline -u "jdbc:hive2://localhost:10001/default;transportMode=http;httpPath=cliservice;principal=HTTP/$(hostname -f)@HORTONWORKS.COM


Ranger Integration with Hadoop
– HDFS
– Hive
– HBase
Objectives
446
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Ranger Plugin - HBase Integration

- Hbase is a non-relational DB on top of Hadoop/HDFS
- HBase Provides Role Based Access Control/ACLs
- ACLs are implemented as a coprocessor called AccessController
- Ranger implements a similar coprocessor for enforcing access control based on Ranger Policies

Ranger Plugin - HBase Integration

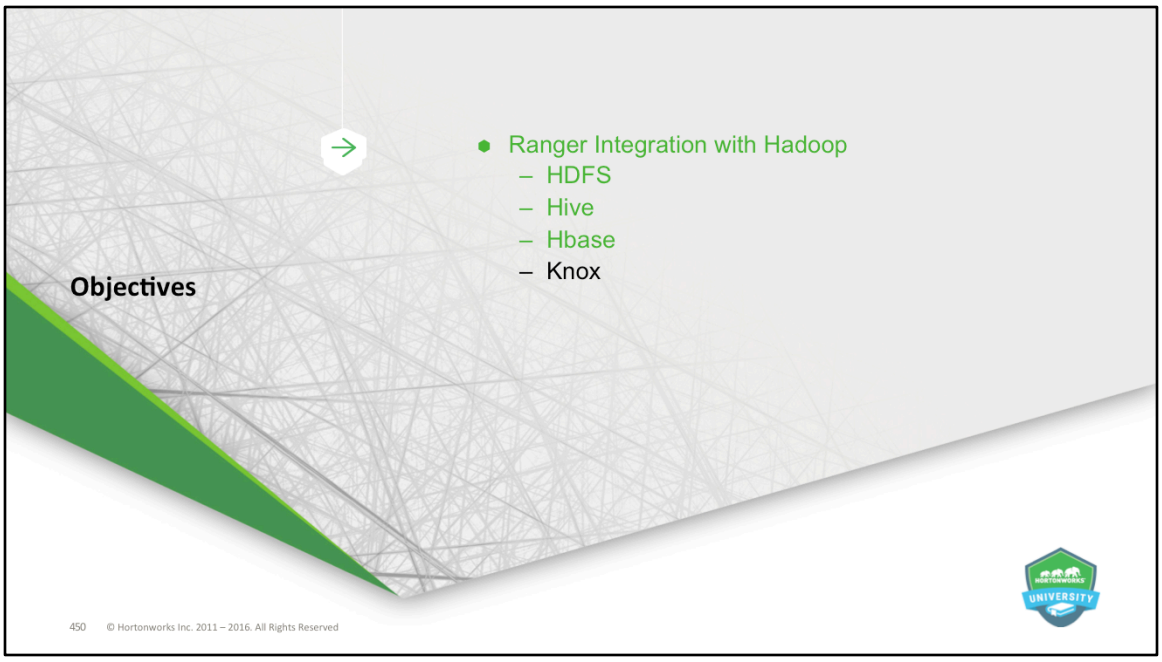
- Ranger Plugin is implemented as a coprocessor of HBase Master/Region Servers to enforce Ranger Policies
- User can define policies on tables, column families and qualifiers
- Supports wildcard in defining policies (table = fin_*, col_fam = audit*)
- Read, Write, Create, Admin permissions allowed - RWCA
- Plugin evaluates Hbase requests grants/denies access based on policies
- Creates necessary audit logs based on audit
- Specific ranger policy must exist for gaining access
- Kerborized Hbase Access how-to (assuming Ranger policy/ACL in place):
  - Just kinit and access as usual

  $ kinit someuser

  $ hbase shell

Ranger Plugin - HBase Integration

- Similar to Hive, Hbase also supports Grant/Revoke commands to manage access control from within Hbase shell.
- Ranger HBase plug-in creates/updates Ranger policies to reflect permissions set via Grant/ Revoke (from within Hbase)
- grant 'bob', 'RWCA', 'test_data'
  - will create a new Ranger policy
- grant 'bob', 'R', 'test_data'
  - will update the Ranger policy created by earlier grant;
- Option to disable Grant/revoke commands
  - Force authorization policies management solely via Ranger Policy Admin tool


Objectives
Ranger Integration with Hadoop
– HDFS
– Hive
– Hbase
– Knox
450
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Ranger Plugin - Knox Integration

- Knox provides perimeter security for Hadoop REST API
- Authentication and token verification at the perimeter
- Authentication integration with enterprise and cloud identity management systems
- Service level authorization at the perimeter
- Single URL hierarchy that aggregates REST APIs of a Hadoop cluster
- Hadoop services with built-in support
  - WebHDFS
  - Hive
  - WebHCat
  - Oozie
  - Hbase (Stargate)
  - Yarn Resource Manager
  - Storm

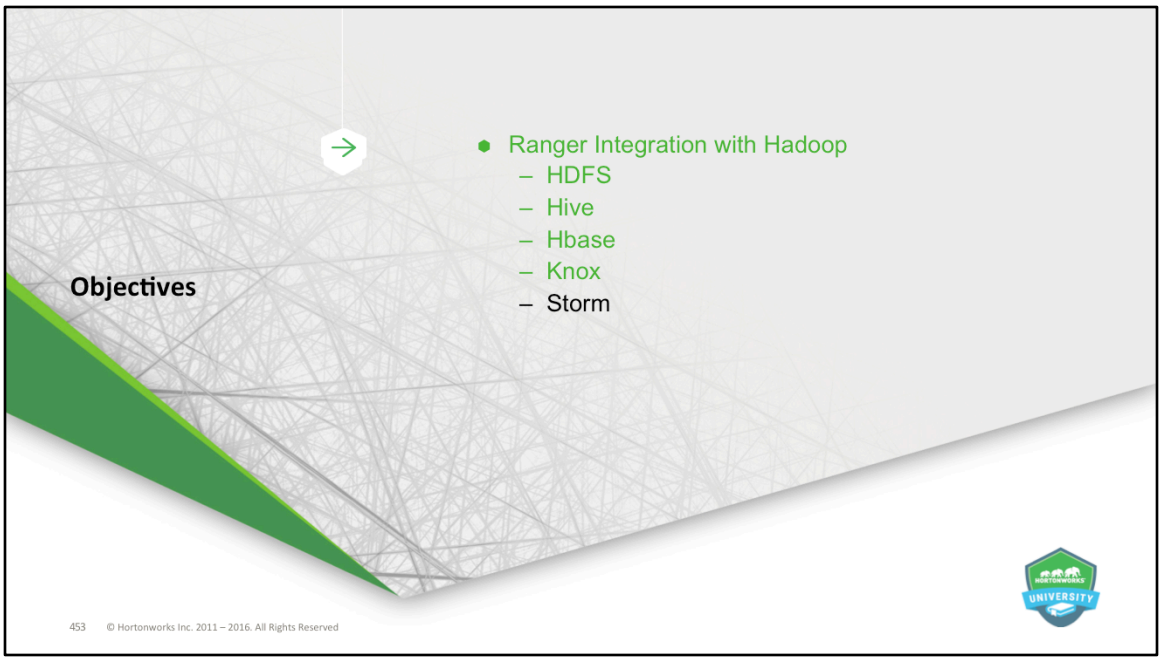
Ranger Plugin - Knox Integration

- Knox provides service level authorization based on xml configuration
- Ranger Plugin will allow service level authorization enforcement via Ranger Policies by acting as Authorization Provider within Knox Gateway
- User can define policies on topologies and services
  - Provide access to services based on user/group/ip-address
  - E.g. Finance group will have access to WebHDFS from 10.1.1.*
- Plugin evaluates Knox requests and grants/denies access based on the policies
- Creates necessary audit logs based on audit policies
- WebHDFS access over Knox how-to (assuming Ranger policy/ACL in place):

$ curl -ik -u sales1:BadPass#1 **https**://$(hostname -f):**8443**/gateway/default/webhdfs/v1/?op=LISTSTATUS

  - Unsecure WebHDFS access sample:

$ curl -sk –L **http**://$(hostname -f):**50070**/webhdfs/v1/?op=LISTSTATUS


Objectives
Ranger Integration with Hadoop
– HDFS
– Hive
– Hbase
– Knox
– Storm
453
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Ranger Plugin - Storm Integration

- Storm is a distributed real-time computation system
- Storm provides general primitives for real-time computation similar to how Hadoop provides general primitives for batch processing
- Storm Topologies :: Hadoop MR Jobs
- Nimbus server (runs on master node)
- Supervisor (runs on each worker node)
- Communication via ZooKeeper
- Storm security is implemented based on Kerberos authentication

Ranger Plugin - Storm Integration

- Ranger Plugin acts as an authorizer within Nimbus server
- Ranger Plugin can authorize all incoming requests based on the Ranger Policies
- User can define policies on topologies
- Permissions can be set
  - getClusterInfo, Submit/Get/Kill/Activate/Deactivate Topology
- Plugin evaluates Storm requests and grants/denies access based on the policies
- Creates necessary audit logs based on audit policies
- Kerborized Storm Access (assuming Ranger policy/ACL in place):
  - Just kinit and access as usual

  $ kinit someuser

  $ storm jar /usr/hdp/current/storm-client/contrib/storm-starter/storm-starter-topologies-*.jar storm.starter.WordCountTopology WordCountTopology -c localhost

Knowledge Check

Knowledge Check

1. Ranger Plugin's act as a authorizer for which services?
2. If a Ranger policy does not exists for HDFS what is used as a fallback?
3. True/False - Ranger policy is not needed to gain access to Hive?
4. What type of policies can be defined for Hive?
5. True/False – Ranger Plugin is implemented as a coprocessor for Hbase?
6. What type of policies can be defined for Hbase?

1. HDFS Namenode, HiveServer2, Knox and Storm
2. HDFS ACL's
3. FALSE – Policy must exist
4. Databases, tables/views columns and UDF's
5. TRUE
6. Tables, Column Families and Qualifiers

Summary

Summary

- Ranger Plugin acts as an authorizer within Namenode
  - Plugin evaluates HDFS requests and provide access
  - No specific ranger policy exists, HDFS ACLs are used as fallback
- Ranger Plugin acts as authorization provider for HiveServer2
  - User can define policies on databases, tables/view, columns and UDFs
  - Plugin evaluates Hive requests and grants/denies access based on the policies
  - Specific Ranger policy must exist for gaining access
- Ranger Plugin is implemented as a coprocessor of HBase Master/Region Servers to enforce Ranger Policies
  - User can define policies on tables, column families and qualifiers
  - Plugin evaluates Hbase requests grants/denies access based on policies
  - Specific ranger policy must exist for gaining access

SSO = Single Sign On

SAML = Security Assertion Markup Language - XML Standard that allow secure web domains to exchange user authentication and authorization data

Summary

- Ranger Plugin will allow service level authorization enforcement via Ranger Policies by acting as Authorization Provider within Knox Gateway
  - User can define policies on topologies and services
  - Plugin evaluates Knox requests and grants/denies access based on the policies
- Ranger Plugin acts as an authorizer within Storm - Nimbus server
  - Ranger Plugin can authorize all incoming requests based on the Ranger Policies
  - User can define policies on topologies
  - Plugin evaluates Storm requests and grants/denies access based on the policies

SSO = Single Sign On

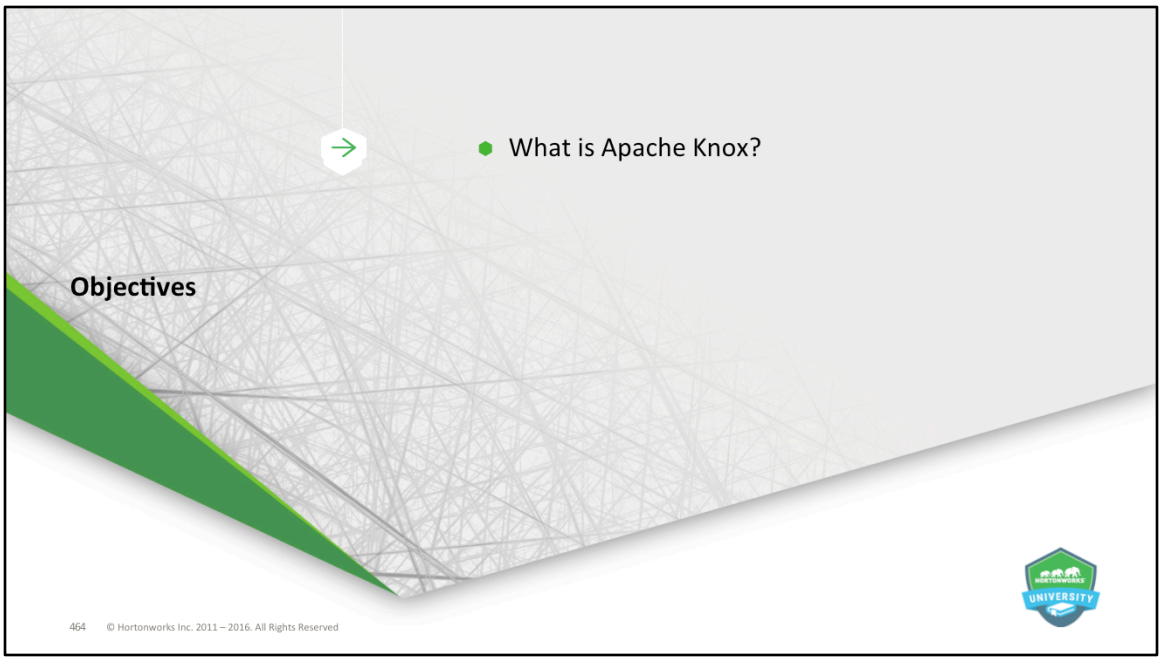
SAML = Security Assertion Markup Language - XML Standard that allow secure web domains to exchange user authentication and authorization data

Lab: Secured Hadoop Exercises


Knox Overview
HORTONWORKS
UNIVERSITY

Lesson Objectives

- Describe Apache Knox
- Understand the need for Apache Knox


What is Apache Knox?
Objectives
464
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

What is Apache Knox

- System to Extent Reach of Apache Hadoop Services to Users Outside of the Cluster
- Provides Single Point of Authentication and Access
- Simplifies Hadoop Security for
  - Users Who Access the Cluster Data/Execute Job
  - Operators Who Control Access/Manage the Cluster
- Provide perimeter security for Hadoop REST API'
  - Provide authentication and token verification at the perimeter
  - Enable authentication integration with enterprise and cloud identity management systems
  - Provide service level authorization at the perimeter
- Expose a single URL hierarchy that aggregates REST APIs of the Cluster
  - Limit the network endpoints/firewall holes required to access the Cluster
  - Hide the internal Hadoop cluster topology from potential attackers

The Apache Knox Gateway is a system that provides a single point of authentication and access for Apache Hadoop services in a cluster. The goal is to simplify Hadoop security for both users (i.e. who access the cluster data and execute jobs) and operators (i.e. who control access and manage the cluster). The gateway runs as a server (or cluster of servers) that provide centralized access to one or more Hadoop clusters. In general the goals of the gateway are as follows:

Provide perimeter security for Hadoop REST APIs to make Hadoop security easier to setup and use

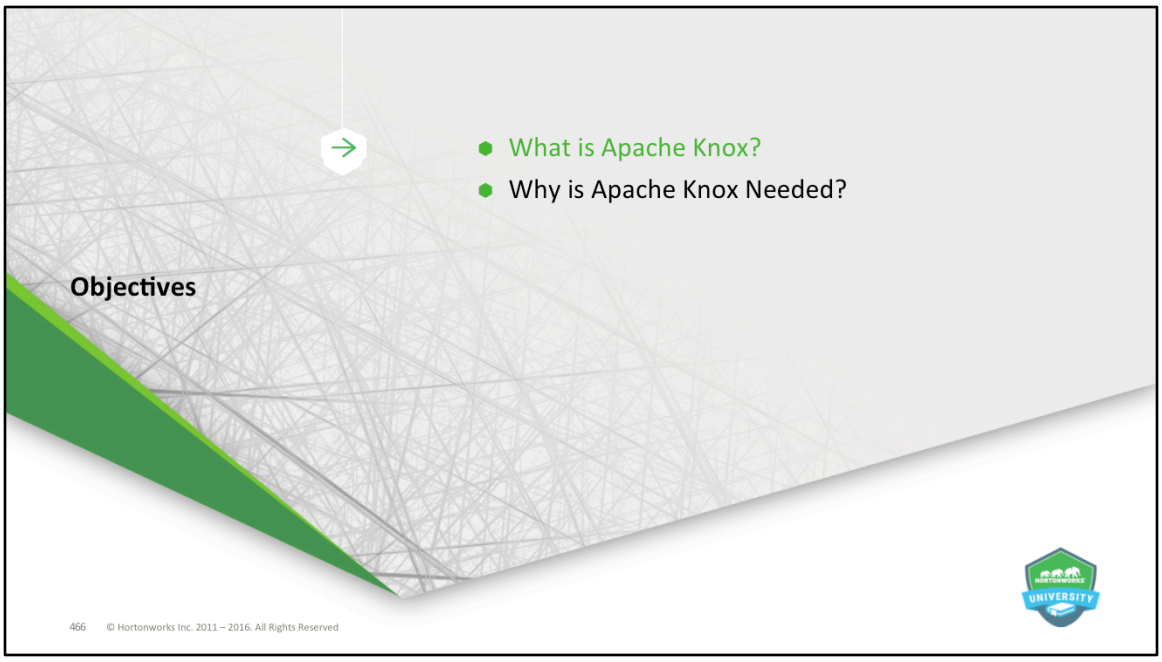
Provide authentication and token verification at the perimeter

Enable authentication integration with enterprise and cloud identity management systems

Provide service level authorization at the perimeter

Expose a single URL hierarchy that aggregates REST APIs of a Hadoop cluster

Limit the network endpoints (and therefore firewall holes) required to access a Hadoop cluster


Objectives
What is Apache Knox?
Why is Apache Knox Needed?
466
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Why is Apache Knox Needed

- Integrates with Identity Management and Single Sign-On (SSO) Systems
- Allows Identity from these Systems to be Used for Access to the Cluster
- Provides Security to Multiple Hadoop Clusters with the Following Advantages:
  - Simplifies access: Extends Hadoop's REST/HTTP services by encapsulating Kerberos to within the Cluster
  - Enhances security: Exposes Hadoop's REST/HTTP services without revealing network details, providing SSL out of the box.
  - Centralized control: Enforces REST API security centrally, routing requests to multiple Hadoop clusters.
  - Enterprise integration: Supports LDAP, Active Directory, SSO, SAML and other authentication systems.

The Apache Knox Gateway ("Knox") is a system to extend the reach of Apache™Hadoop® services to users outside of a Hadoop cluster without reducing Hadoop Security. Knox also simplifies Hadoop security for users who access the cluster data and execute jobs.

Knox integrates with Identity Management and SSO systems used in enterprises and allows identity from these systems be used for access to Hadoop clusters.

Knox Gateways provides security for multiple Hadoop clusters, with these advantages:

Simplifies access: Extends Hadoop's REST/HTTP services by encapsulating Kerberos to within the Cluster.

Enhances security: Exposes Hadoop's REST/HTTP services without revealing network details, providing SSL out of the box.

Why Apache Knox Needed

Enhanced Security

- Protect network details
- SSL for non-SSL services
- WebApp vulnerability filter

Centralized Control

- Central REST API auditing
- Service-level authorization
- Alternative to SSH “edge node”

Simplified Access

- Kerberos encapsulation
- Extends API reach
- Single access point
- Multi-cluster support
- Single SSL certificate

Enterprise Integration

- LDAP integration
- Active Directory integration
- SSO integration
- Apache Shiro extensibility
- Custom extensibility

Hadoop REST API with Knox

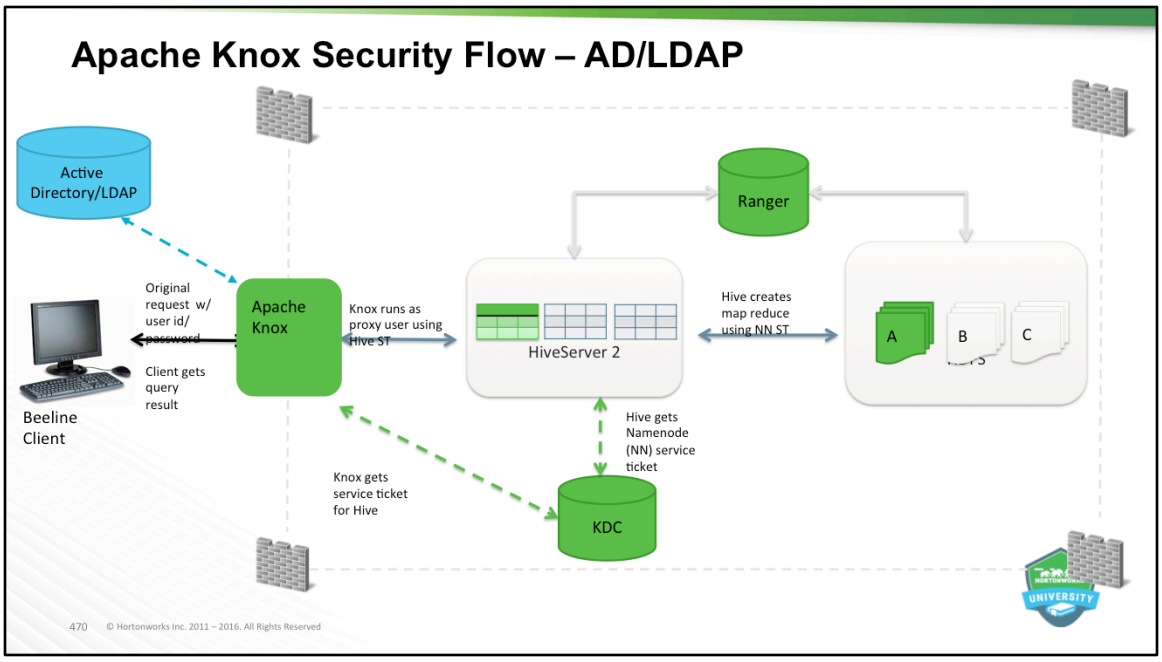
Service	Direct URL	Knox URL
WebHDFS	http://namenode-host:50070/webhdfs	https://knox-host:8443/webhdfs
WebHCat	http://webhcat-host:50111/templeton	https://knox-host:8443/templeton
Oozie	http://oozie-host:11000/oozie	https://knox-host:8443/oozie
HBase	http://hbase-host:60080	https://knox-host:8443/hbase
Hive	http://hive-host:10001/cliservice	https://knox-host:8443/hive
YARN	http://yarn-host:8088/ws	https://knox-host:8443/resourcemanager

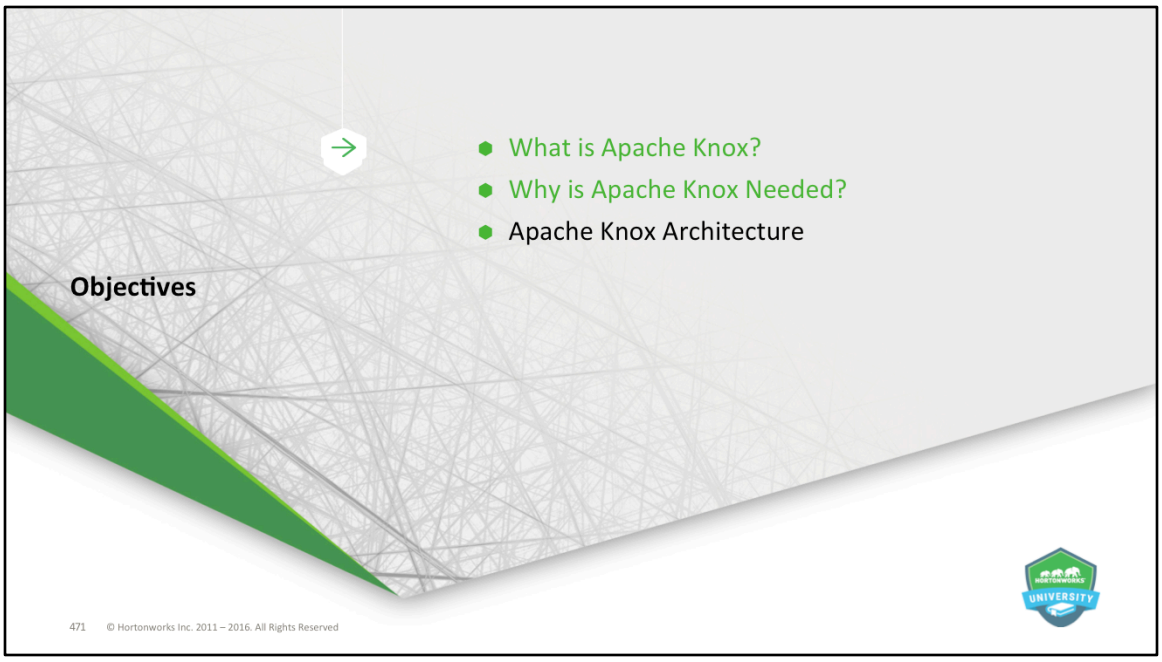
Masters could be on many different hosts

SSL config on one host

One host, one port

Consistent paths

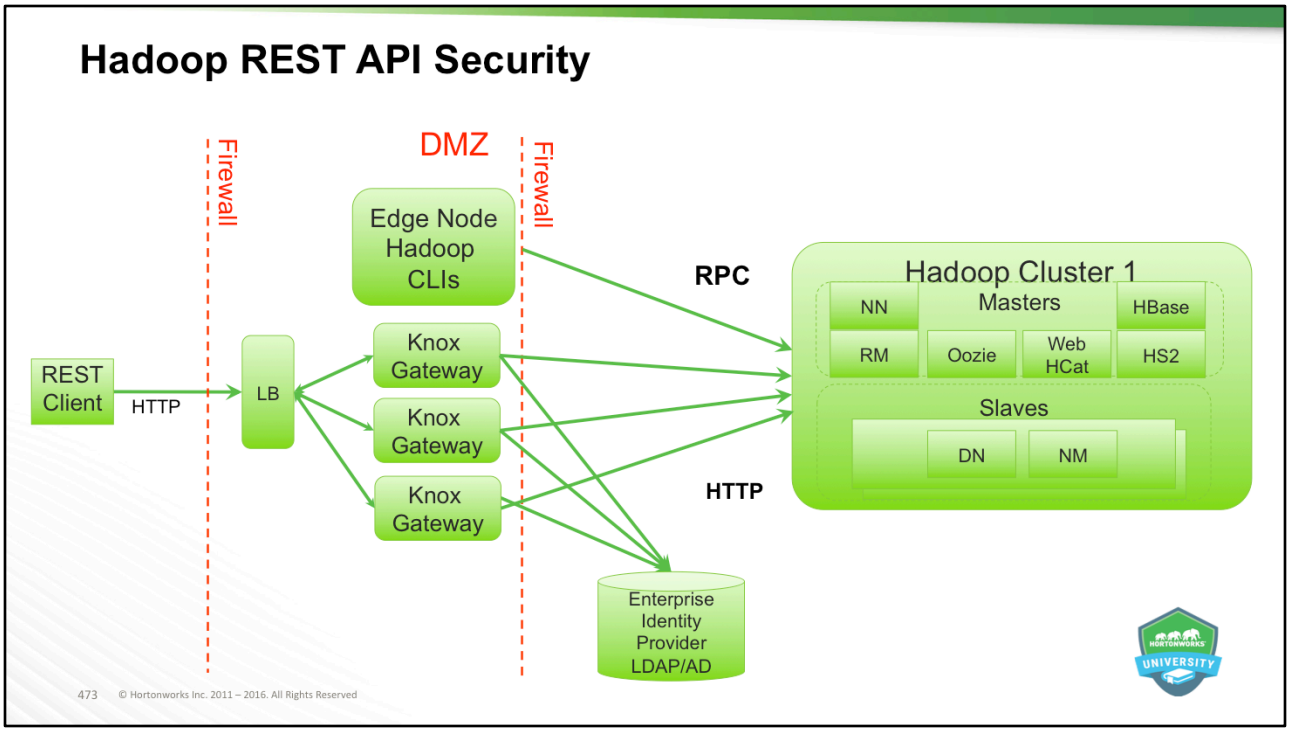

Apache Knox Security Flow – AD/LDAP
Active Directory/LDAP
Ranger
Original request w/ user id/ password
Apache Knox
Knox runs as proxy user using Hive ST
HiveServer 2
Hive creates map reduce using NN ST
A
B
C
Client gets query result
Beeline Client
Hive gets Namenode (NN) service ticket
Knox gets service ticket for Hive
KDC
470
© Hortonworks Inc. 2011 – 2016. All Rights Reserved


Objectives
What is Apache Knox?
Why is Apache Knox Needed?
Apache Knox Architecture
471
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Apache Knox Architecture

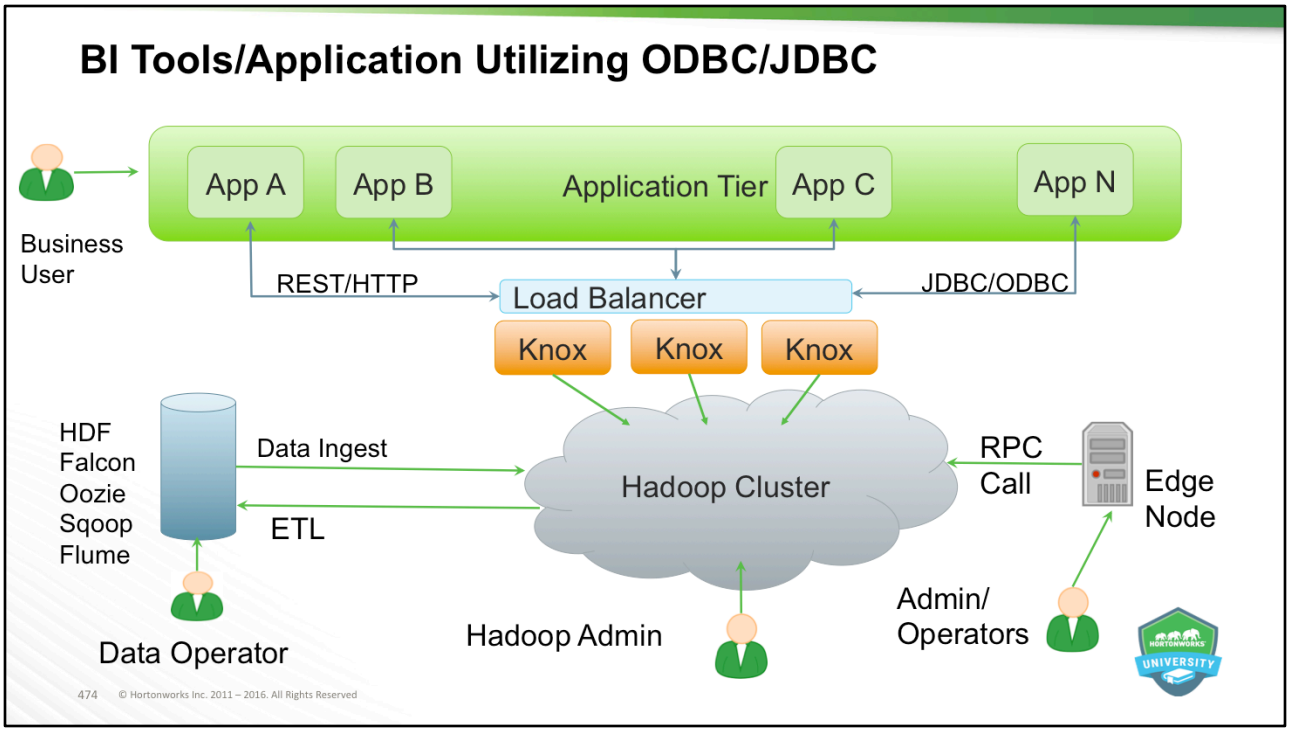
- Users Access Hadoop Externally Do So Through Knox
  - Via Apache REST API
  - Through Hadoop CLI tools
- Supported Hadoop Services Both Unsecure and Kerberized

Service	Version
YARN	2.6.0
WebHDFS	2.6.0
WebHCat/Templeton	0.13.0
Oozie	4.1.0
Hbase/Stargate	0.98.4
Hive Via WebHCat	0.14.0
Hive Via JDBC	0.14.0



Typical topology would look like this where you have 3 layers:
1. Presentation
2. Application
3. Data

Knox can run in DMZ in between firewalls so your cluster and AD remains hidden from outside world
With Knox you can optionally expose data from multiple clusters (without end users knowing where data came from)



Applications can interact with Knox via REST/HTTP or JDBC/ODBC

Can run multiple Knox gateways via LoadBalancers

Knox is not meant to bulk import/export of data (would be like drinking water from pool using a straw)
Bulk operations should happen within cluster via HDF, Sqoop etc

Knowledge Check

Knowledge Check

1. What does Apache Knox provide for Hadoop REST API's?
2. What are the Four Advantages of Apache Knox Security?
3. How does Apache Knox Simplifies Access and Enhance Security?
4. True/False - Apache Knox Gateway provides SSL out of the box?
5. What Enterprise Directory Services does Knox support?

1. Perimeter Security

2. Simplifies Access; Enhances Security; Centralized Control; Enterprise Integration

3. Extends Hadoop's REST/HTTP Services by Encapsulating Kerberos to within the Hadoop cluster and Exposes Single URL Hierarchy without revealing Hadoop Cluster details,

4. TRUE

5. LDAP, Active Directory, Single Sign-On and SAML (Security Assertion Markup Language)

Summary

Summary

- Apache Knox Gateway is a System to Extend the Reach of Apache Hadoop Services to Users Outside of the Cluster
- Knox Gateway Provides a Single Point of Authentication and Access
- Provide Perimeter Security for Hadoop REST API's
- Simplifies Hadoop Security by Encapsulating Kerberos to within the Hadoop Cluster
- Knox Exposes a Single URL Hierarchy that Aggregates REST API's of the Hadoop Cluster
- Integrates with Enterprise Authentication: LDAP, Active Directory, SSO and SAML

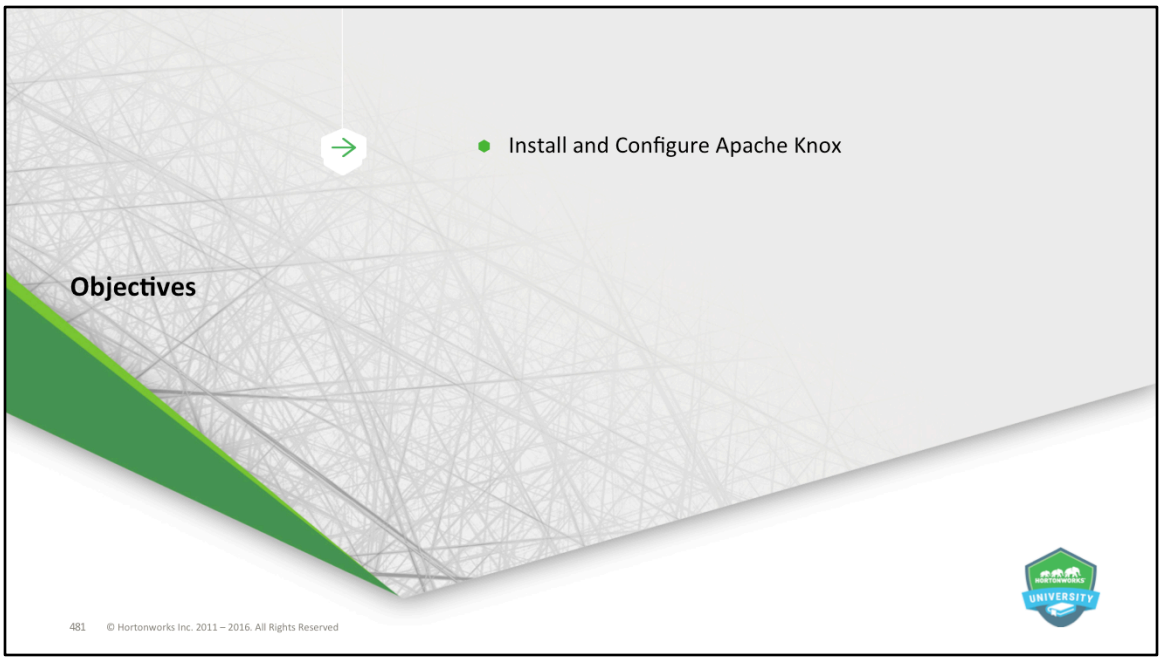
SSO = Single Sign On

SAML = Security Assertion Markup Language - XML Standard that allow secure web domains to exchange user authentication and authorization data


Knox Installation
HORTONWORKS
UNIVERSITY

Objectives

- Install and Configure Apache Knox
- Configure Apache Knox Authentication - LDAP/AD

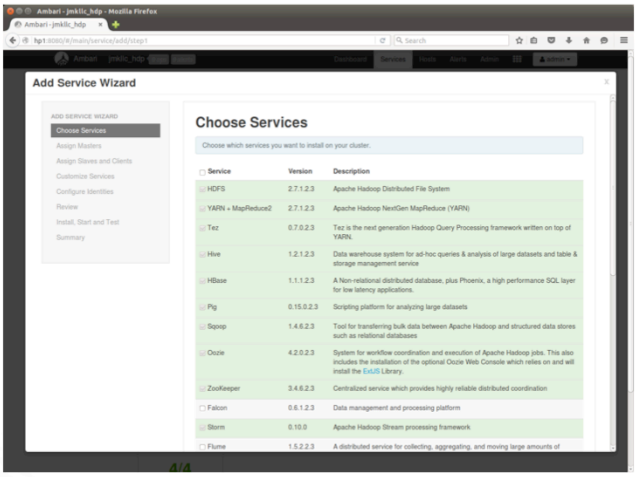

Install and Configure Apache Knox
Objectives
481
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Install Apache Knox with Ambari

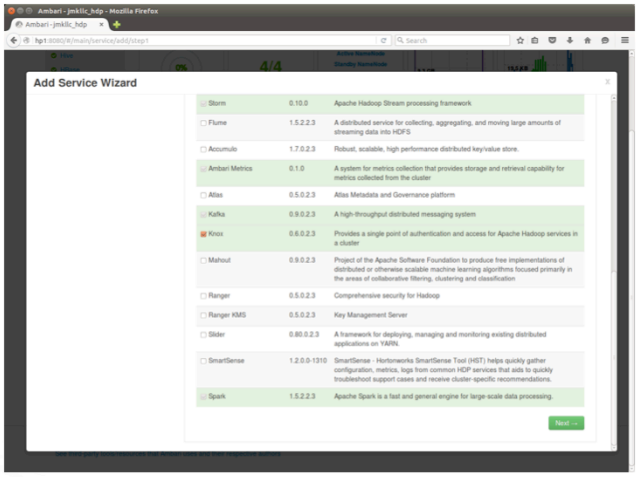
- Install Knox Via Add Service Wizard Under “Admin > Stack and Versions”
- Add Service Wizard – Select Knox
- Choose Services – Confirm Knox is Selected
- Assign Master – Choose Hosts to Run Knox
- Customize Services – Create Knox Master Secret Password
- Review Configuration – Deploy
- Monitor Deployment – Install, Start and Test
- Review Summary
- Installation Complete

Install Apache Knox with Ambari

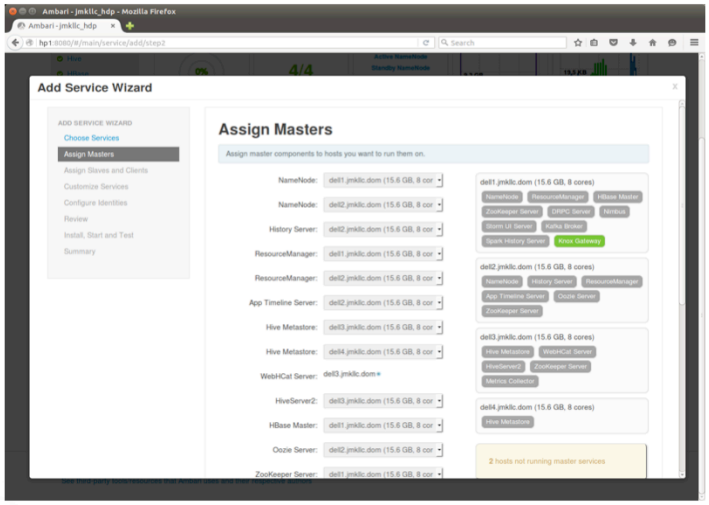
Install Apache Knox with Ambari



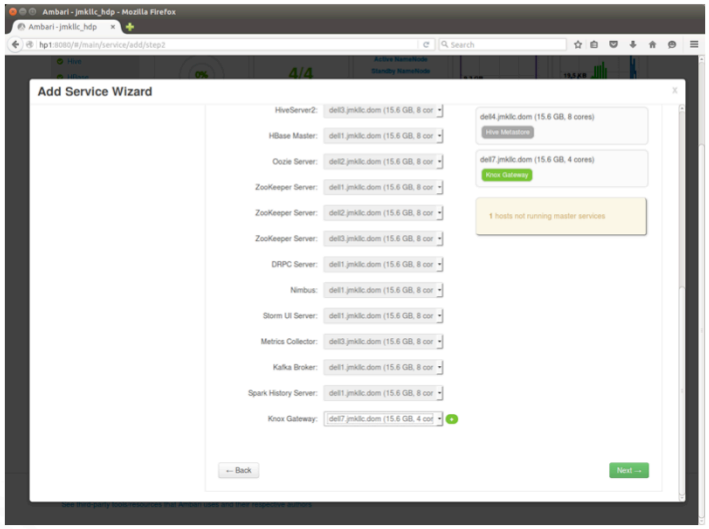
Install Apache Knox with Ambari



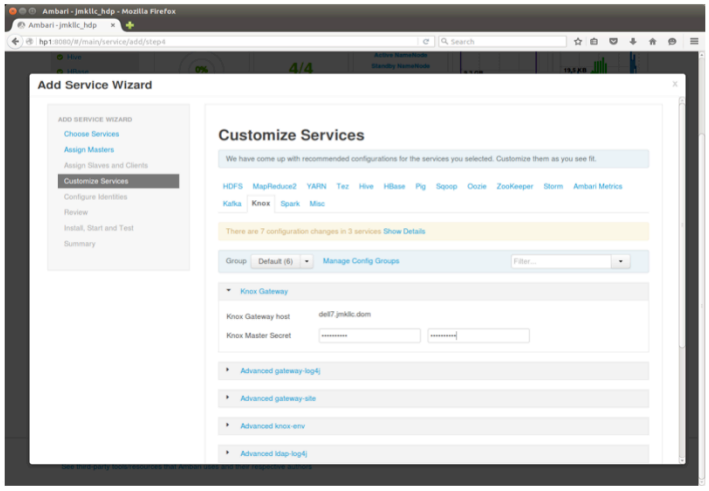
Install Apache Knox with Ambari



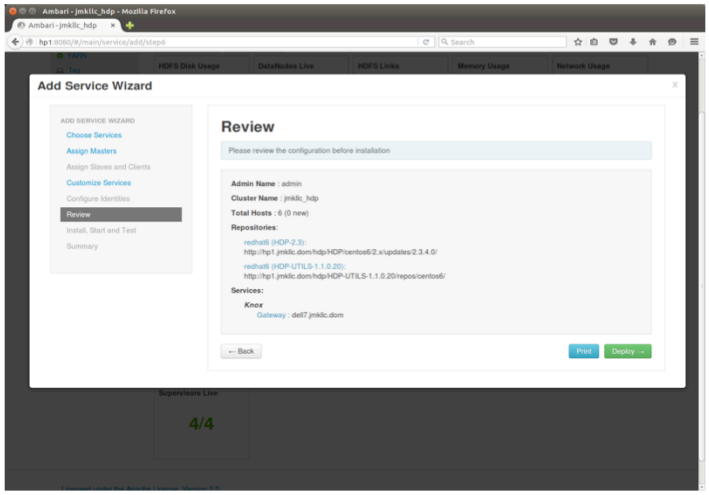
Install Apache Knox with Ambari



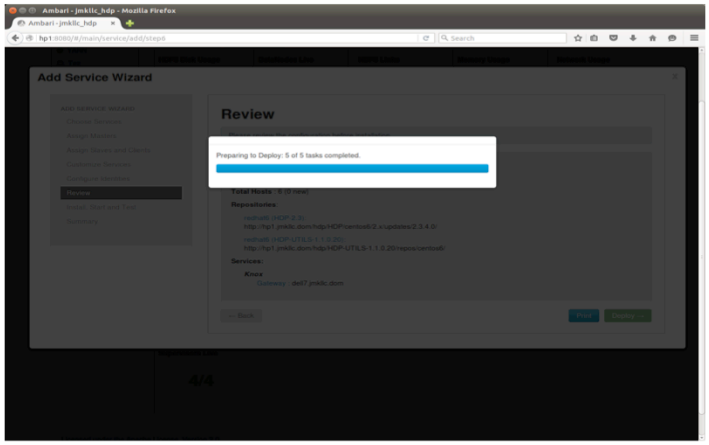
Install Apache Knox with Ambari



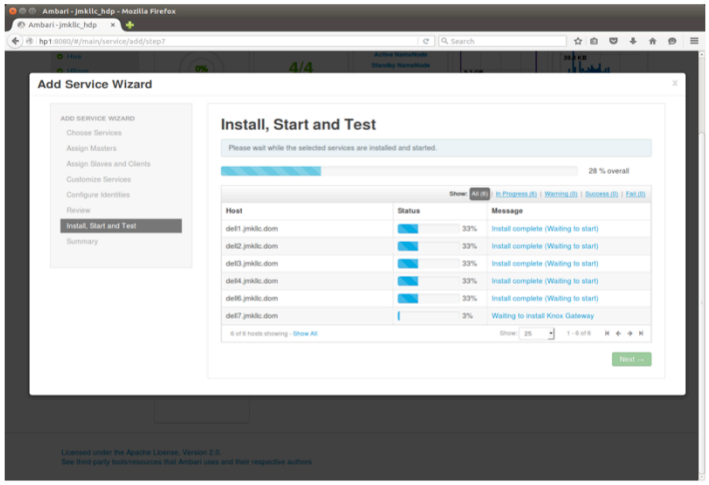
Install Apache Knox with Ambari



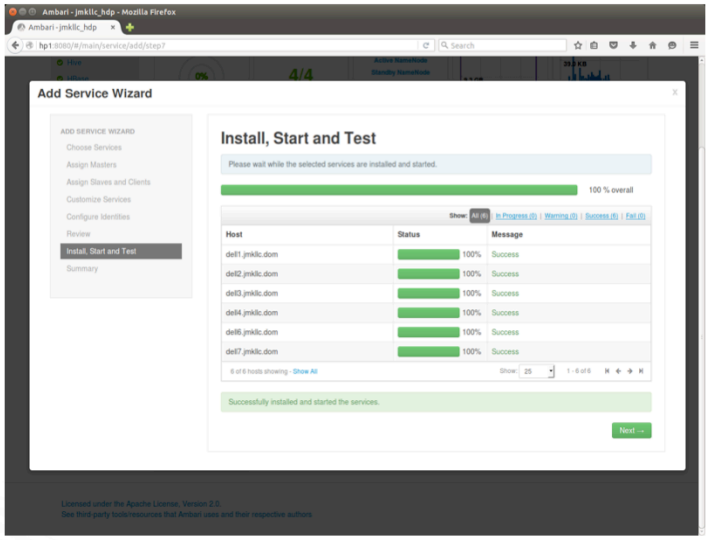
Install Apache Knox with Ambari



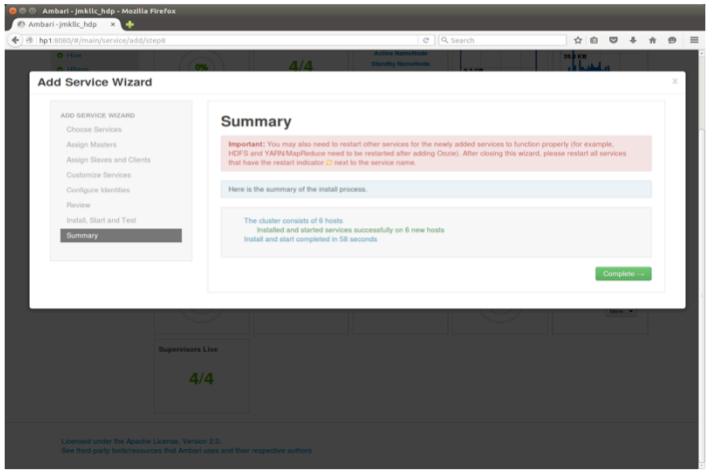
Install Apache Knox with Ambari



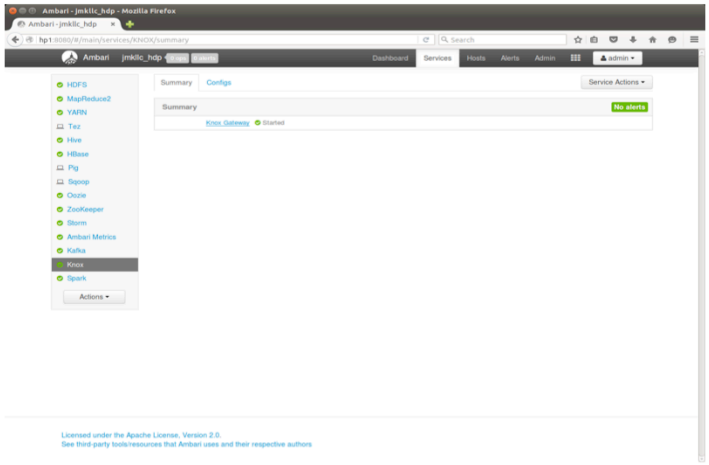
Install Apache Knox with Ambari

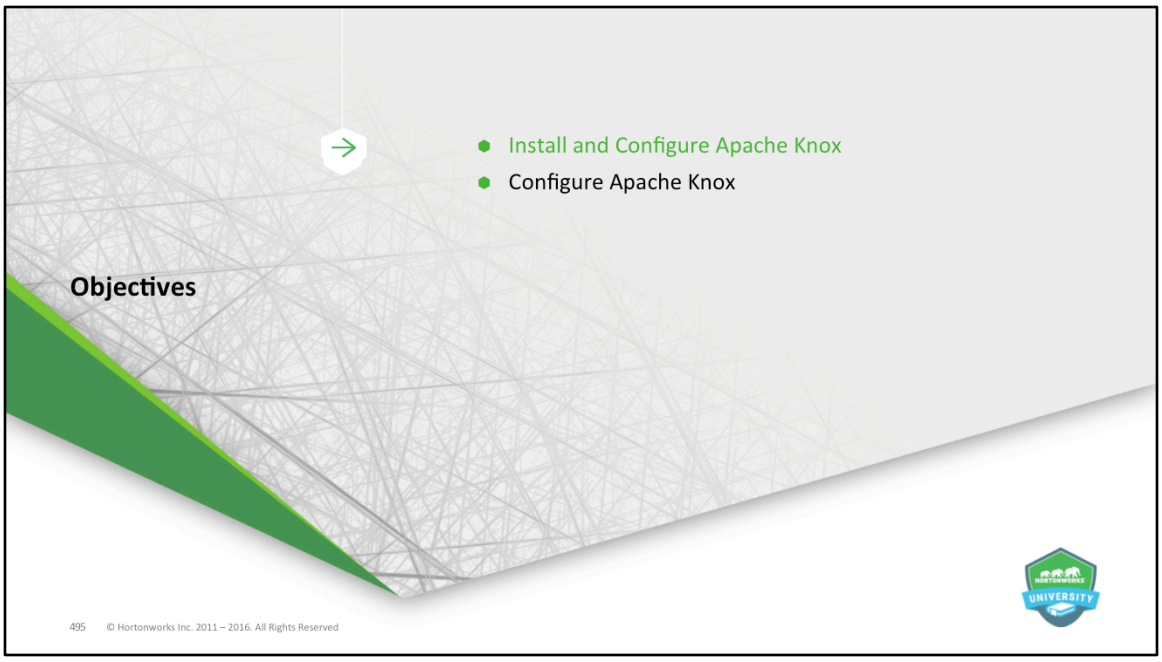


Install Apache Knox with Ambari



Install Apache Knox with Ambari




Install and Configure Apache Knox
Configure Apache Knox
Objectives
495
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Configure Apache Knox

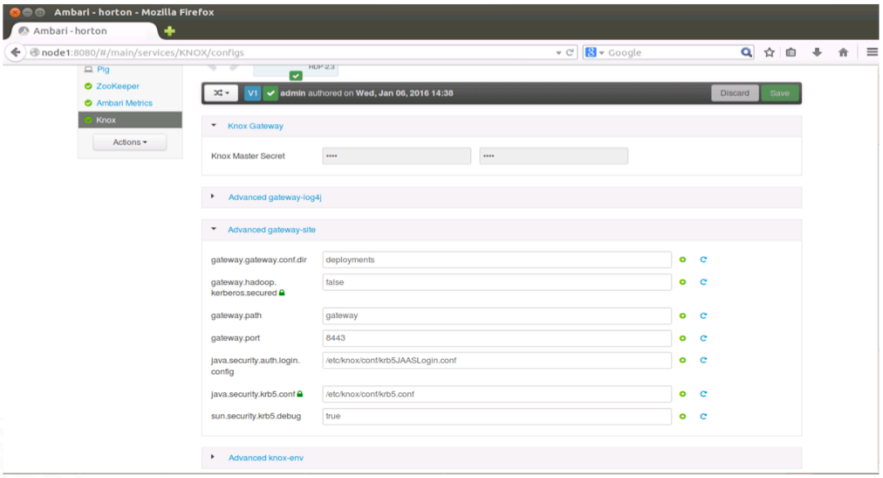
- Customize Gateway Port and Path
- Properties Effect the URL Used by External Clients
- Default Port is Set to 8443
- Default Context Path is “gateway”

  https://knox.hortonworks.com:8443/gateway/default/…
- To Change Port or Context Path – Edit gateway-site.xml File

Property	Value
gateway.port	8443
gateway.path	gateway

- Restart Service

Configure Apache Knox



Configure Apache Knox

- Master Secret is Required to Start Gateway
- Protects Artifacts Used by Gateway Instance
  - Keystore
  - Trust Stores
  - Credential Stores
- Ambari Persist the Master Secret in "master" File
  /usr/hdp/<HDP VERSION>/knox/data/security/master
- Secret File Encrypted with AES 128 Bit Encryptions
- File is Owned by and Readable/Writable by Knox User

Configure Apache Knox

- Defining Cluster Topologies – Ambari Install Provides Default Topology
- Gateway Supports One or More Hadoop Clusters
- Each Cluster Configuration is Defined in a Topology Deployment Descriptor File
  /usr/hdp/<HDP VERSION>/knox/conf/topologies
- Deployed to Corresponding WAR File
  /usr/hdp/<HDP VERSION>/knox/data/deployments
- These Files Define How Gateway Communication with Cluster
- Descriptor File is XML File with Following Sections
  - gateway/provider – Configuration Settings Enforced by Gateway While Providing Access
  - service – Defines Hadoop Service URL's Used to Proxy Communication From External Clients
- Automatically Redeploys when Detection of New/Change Topology Descriptor File

Configure Apache Knox

- Cluster Topology Provider/Service Roles

Type	Role	Description
gateway/ provider	hostmap	Maps external to internal node hostnames, replacing the internal hostname with the mapped external name when the hostname is embedded in a response from the cluster.
	authentication	Integrates an LDAP store to authenticate external requests accessing the cluster via the Knox Gateway. Refer to Set Up LDAP Authentication for more information.
	federation	Defines HTTP header authentication fields for an SSO or federation solution provider. Refer to Set up HTTP Header Authentication for Federation/SSO.
	identity-assertion	Responsible for the way that the authenticated user's identity is asserted to the service that the request is intended for. Also maps external authenticated users to an internal cluster that the gateway asserts as the current session user or group. Refer to Configure Identity Assertion for more information.
	authorization	Service level authorization that restricts cluster access to specified users, groups, and/or IP addresses. Refer to Configure Service Level Authorization for more information.
	webappspec	Configures a web application security plugin that provides protection filtering against Cross Site Request Forgery Attacks. Refer to Configure Web Application Security for more information.
HA provider	high avilability	Syncs all Knox instances to use the same topologies credentials keystores.
service	$service_name	Binds a Hadoop service with an internal URL that the gateway uses to proxy requests from external clients to the internal cluster services. Refer to Configure Hadoop Service URLs for more information.

gateway/provider hostmap Maps external to internal node hostnames, replacing the internal hostname with the mapped external name when the hostname is embedded in a repsonse from

the cluster.

authenticationIntegrates an LDAP store to authenticate external requests accessing the cluster via the Knox Gateway. Refer to Set Up LDAP Authentication for more

information.

federation Defines HTTP header authentication fields for an SSO or federation solution provider. Refer to Set up HTTP Header Authentication for Federation/SSO

identity-assertion

Responsible for the way that the authenticated user's identity is asserted to the service that the request is intended for. Also maps external authenticated users

Configure Apache Knox

- Setting Up Hadoop Service URL's
- To Configure Access to Internal Hadoop Service – Edit
  /usr/hdp/<HDP VERSION>/knox/conf/topologies/<CLUSTER NAME>.xml
- Add/Modify Entry for each Hadoop Service

```
<topology>
  <gateway>
  ...
  </gateway>
  <service>
    <role>NAMENODE</role>
    <url>hdfs://node1.hortonworks.com:8020</url>
  </service>
</topology>
```

- Gateway Creates New WAR File with Modified Timestamp in
  /usr/hdp/<HDP VERSION>/knox/data/deployments

To configure access to an internal Hadoop service through the Knox Gateway:

Edit $gateway/conf/topologies$cluster-name.xml to add an entry similar to the following, for each Hadoop service:

```
<topology>
  <gateway>
  ...
  </gateway>
  <service>
    <role> $service_name </role>
    <url> $schema://$hostname:$port</url>
  </service>
</topology>
```

where:


Objectives
Install and Configure Apache Knox
Configure Apache Knox
Configure Apache Knox Authentication
502
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Configure Apache Knox Authentication

- Setting Up LDAP/AD Authentication
- Knox Gateway uses Apache Shiro to Authenticate Users Against LDAP Store
- Provides HTTP BASIC Authentication against LDAP User Directory
- Currently Support Only a Single Organizational Unit (OU)
- Does Not Support Nested OUs
- To Enable Add “ShiroProvider” authentication provider to Cluster’s Topology File

LDAP authentication is configured by adding a "ShiroProvider" authentication provider to the cluster's topology file. When enabled, the Knox Gateway uses Apache Shiro (org.apache.shiro.realm.ldap.JndiLdapRealm) to authenticate users against the configured LDAP store.

Knox Gateway provides HTTP BASIC authentication against an LDAP user directory. It currently supports only a single Organizational Unit (OU) and does not support nested OUs.

Configure Apache Knox Authentication

Enable LDAP Authentication

- Create Keystore Alias for the LDAP Manager User
  - Read Password for Use In knoxcli.sh Script
  - Handy Way to Set Environment Variable without Storing in Shell Command History
    $ read -s -p "Password: " knoxpass
  - Create Password Alias for Knox called knoxLdapSystemPassword
    $ sudo sudo -u knox /usr/hdp/current/knox-server/bin/knoxcli.sh create-alias knoxLdapSystemPassword --cluster default --value ${knoxpass}
    $ unset knoxpass
- Add Following LDAP Entries in Topology

LDAP authentication is configured by adding a "ShiroProvider" authentication provider to the cluster's topology file. When enabled, the Knox Gateway uses Apache Shiro (org.apache.shiro.realm.ldap.JndiLdapRealm) to authenticate users against the configured LDAP store.

Knox Gateway provides HTTP BASIC authentication against an LDAP user directory. It currently supports only a single Organizational Unit (OU) and does not support nested OUs.

To enable LDAP authentication:

Open the cluster topology descriptor file, $cluster-name.xml, in a text editor.

Add the ShiroProvider authentication provider to /topology/gateway as follows:

<provider>

Configure Apache Knox Authentication

Locate the Following Section

```
        <param>
         <name>main.ldapRealm</name>
         <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</value>
        </param>
<!-- Add these changes for AD/user sync -->
<param>
  <name>main.ldapContextFactory</name>
  <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
</param>
<!-- main.ldapRealm.contextFactory needs to be placed before other main.ldapRealm.contextFactory* entries  -->
<param>
  <name>main.ldapRealm.contextFactory</name>
  <value>$ldapContextFactory</value>
</param>
```


```
                <param>
            <name>main.ldapRealm</name>
            <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</
value>
          </param>

<!-- changes for AD/user sync -->

<param>
  <name>main.ldapContextFactory</name>
  <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
</param>

<!-- main.ldapRealm.contextFactory needs to be placed before other
main.ldapRealm.contextFactory* entries  -->
```

Configure Apache Knox Authentication

```
<!-- Add these change for AD url -->
<param>
  <name>main.ldapRealm.contextFactory.url</name>
  <value>ldap://ad01.lab.hortonworks.net:389</value>
</param>
<!-- system user -->
<param>
  <name>main.ldapRealm.contextFactory.systemUsername</name>
  <value>cn=ldap-reader,ou=ServiceUsers,dc=lab,dc=hortonworks,dc=net</value>
</param>
<!-- pass in the password using the alias created earlier -->
<param>
  <name>main.ldapRealm.contextFactory.systemPassword</name>
  <value>${ALIAS=knoxLdapSystemPassword}</value>
</param>
```


```
                <param>
            <name>main.ldapRealm</name>
            <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</
value>
          </param>

<!-- changes for AD/user sync -->

<param>
  <name>main.ldapContextFactory</name>
  <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
</param>

<!-- main.ldapRealm.contextFactory needs to be placed before other
main.ldapRealm.contextFactory* entries  -->
```

Configure Apache Knox Authentication

Locate the Following Section

```
        <param>
          <name>main.ldapRealm.contextFactory.authenticationMechanism</name>
          <value>simple</value>
        </param>
        <param>
          <name>urls./**</name>
          <value>authcBasic</value>
        </param>
<!--  Add these changes for  AD groups of users to allow -->
<param>
  <name>main.ldapRealm.searchBase</name>
  <value>ou=CorpUsers,dc=lab,dc=hortonworks,dc=net</value>
</param>
```


```
                <param>
            <name>main.ldapRealm</name>
            <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</
value>
          </param>

<!-- changes for AD/user sync -->

<param>
  <name>main.ldapContextFactory</name>
  <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
</param>

<!-- main.ldapRealm.contextFactory needs to be placed before other
main.ldapRealm.contextFactory* entries  -->
```

Configure Apache Knox Authentication

```
<param>
  <name>main.ldapRealm.userObjectClass</name>
  <value>person</value>
</param>
<param>
  <name>main.ldapRealm.userSearchAttributeName</name>
  <value>sAMAccountName</value>
</param>
<!-- changes needed for group sync-->
<param>
  <name>main.ldapRealm.authorizationEnabled</name>
  <value>true</value>
</param>
<param>
  <name>main.ldapRealm.groupSearchBase</name>
  <value>ou=CorpUsers,dc=lab,dc=hortonworks,dc=net</value>
</param>
```


```
                <param>
            <name>main.ldapRealm</name>
            <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapRealm</
value>
          </param>

<!-- changes for AD/user sync -->

<param>
   <name>main.ldapContextFactory</name>
   <value>org.apache.hadoop.gateway.shirorealm.KnoxLdapContextFactory</value>
</param>

<!-- main.ldapRealm.contextFactory needs to be placed before other
main.ldapRealm.contextFactory* entries  -->
```

Configure Apache Knox Authentication

```
<param>
  <name>main.ldapRealm.groupObjectClass</name>
  <value>group</value>
</param>
<param>
  <name>main.ldapRealm.groupIdAttribute</name>
  <value>cn</value>
</param>
```

- Save the Advanced Topology
- Restart Knox Service

Configure Apache Knox Authentication

- HDFS Configuration for Knox
- Allow Knox to Access Cluster on User Behalf
- Edit HDFS Custom Core-Site Add:

  hadoop.proxyuser.knox.groups=users,hadoop-admins,sales,hr,legal

  hadoop.proxyuser.knox.hosts=*
- Restart HDFS
  - The Impersonation is needed otherwise the Following Error is Displayed:

  org.apache.hadoop.security.authorize.AuthorizationException: User: knox is not allowed to impersonate sales1"

Tell Hadoop to allow our users to access Knox from any node of the cluster. Make the below change in Ambari > HDFS > Config > Custom core-site

hadoop.proxyuser.knox.groups=users,hadoop-admins,sales,hr,legal

hadoop.proxyuser.knox.hosts=*

(better would be to put a comma separated list of the FQDNs of the hosts)

Now restart HDFS

Without this step you will see an error like below when you run the WebHDFS request later on:

org.apache.hadoop.security.authorize.AuthorizationException: User: knox is not allowed to impersonate sales1"

Configure Apache Knox Authentication

- Ranger Configuration for Knox
- Setup Knox Policy for "sales" Group Access Via WEBHDFS
- Login to Ranger - Access Manager > Knox
- Click on Add New Policy
  - Policy name: webhdfs
  - Topology name: default
  - Service name: WEBHDFS
  - Group permissions: sales
  - Permission: check Allow
  - Add

Ranger Configuration for Knox

Setup a Knox policy for sales group for WEBHDFS by:

Login to Ranger > Access Manager > KNOX > click the cluster name link > Add new policy

Policy name: webhdfs

Topology name: default

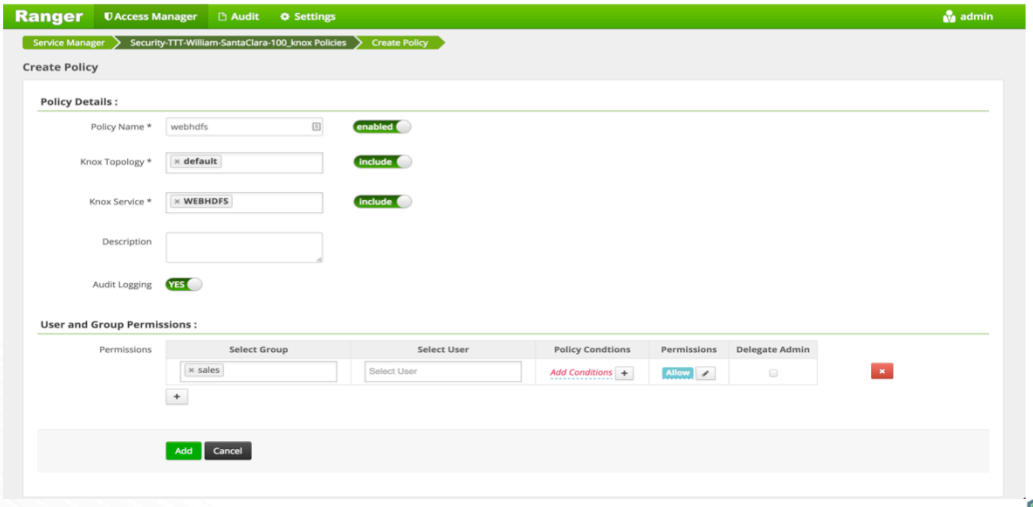
Service name: WEBHDFS

Group permissions: sales

Permission: check Allow

Add

Configure Apache Knox Authentication



Ranger Configuration for Knox

Setup a Knox policy for sales group for WEBHDFS by:

Login to Ranger > Access Manager > KNOX > click the cluster name link > Add new policy

Policy name: webhdfs

Topology name: default

Service name: WEBHDFS

Group permissions: sales

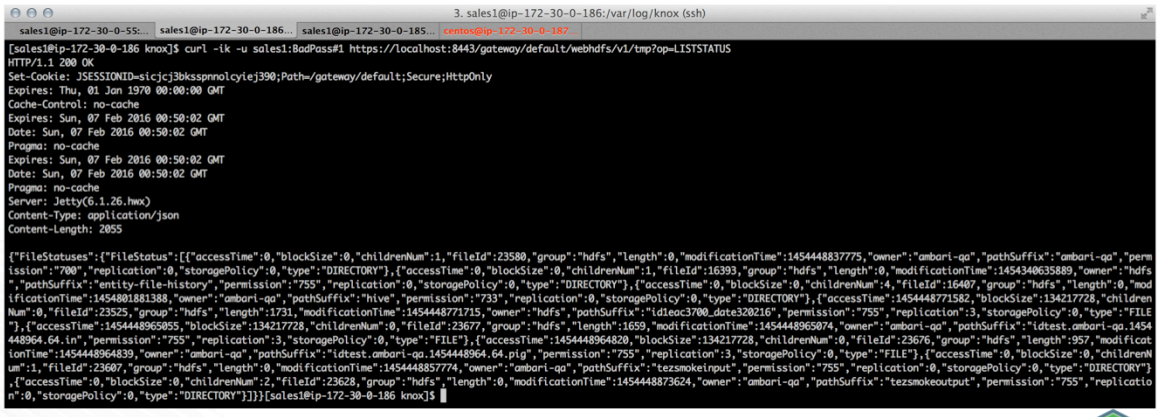
Permission: check Allow

Add

Configure Apache Knox Authentication

Testing Provider

$ curl -ik -u sales1:BadPass#1 https://localhost:8443/gateway/default/webhdfs/v1/tmp?op=LISTSTATUS



Using cURL, you can test your LDAP configuration as follows:

Open the command line on an external client.

Enter the following command to list the contents of the directory tmp/test:

curl -i -k -u ldap_user : password -X GET / 'https:// gateway_host :8443/ gateway_path / cluster_name /webhdfs/api/v1/tmp/test?op=LISTSTATUS

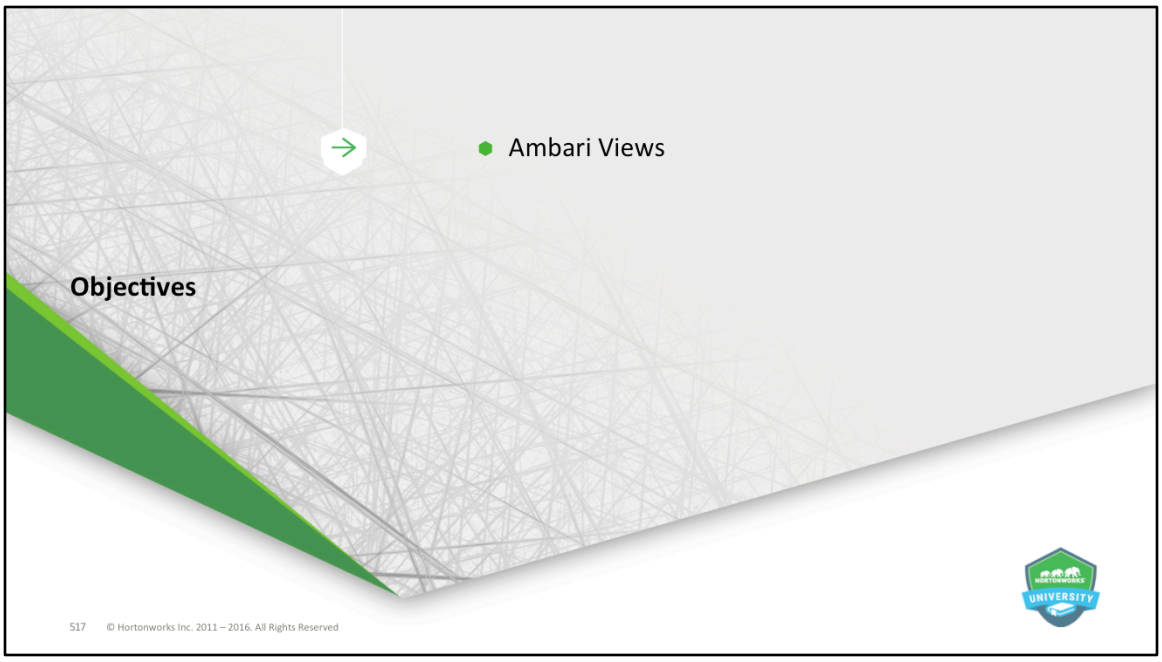
If the directory exists, a content list displays; if the user cannot be authenticated, the request is rejected with an HTTP status of 401 unauthorized.

Lab: Knox


Ambari Views for
Controlled Access
HORTONWORKS
UNIVERSITY

Lesson Objectives

- Summarize the purpose of Ambari Views
- Configure a Standalone Ambari Views Server
- Configure Ambari Views Server for Kerberos
- Enable Kerberos for Views


Ambari Views
Objectives
517
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Ambari Views

- Framework to Allow Developers to Create UI Components that "Plug-Into" Ambari Web UI
- Included are a Built-In Set of Views that are Pre-Deployed for Use
- Views Can Be Deployed/Managed in "Operational" Ambari Server
- View Can Also Be Deployed/Manager in One or More Separate "Standalone" Servers
- Standalone Instances
  - Useful When Users Will Access Cluster Via Views
  - Useful For User Who Should Not Have Access to "Operational" Ambari Server
  - Allow A Scale-Out Approach for Handling Large Number of Users


Ambari Views
Setup Standalone Ambari Views Server
Objectives
519
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Setup Standalone Ambari View Server

Prerequisites

- Instances Should Be Same Version
- Point to the Same Underlying Database - Not DB Being Used by Operational Server
- Database Should Be Scaled and Highly-Available
- Authentication LDAP/AD Should Be Configured the Same for All Instances
- Kerberos-Enabled Clusters Require Instances Configured for Kerberos
- Run Multiple Instances Behind Reverse Proxy

Setup Standalone Ambari View Server

- Setup Similar to Operational Ambari Server
- Does Not Install/Manage a Cluster
- Does Not Deploy/Communicate with Ambari Agents
- Run as Web Server Instance Serving View for Users
- Must Be Configured Same as Operational Ambari Server
  - Configure Server for Non-Root
  - Encrypt Database and Passwords
  - Enable LDAP/AD Authentication
  - Enable HTTPS/SSL Server
  - Enable SPNEGO Authentication for Hadoop

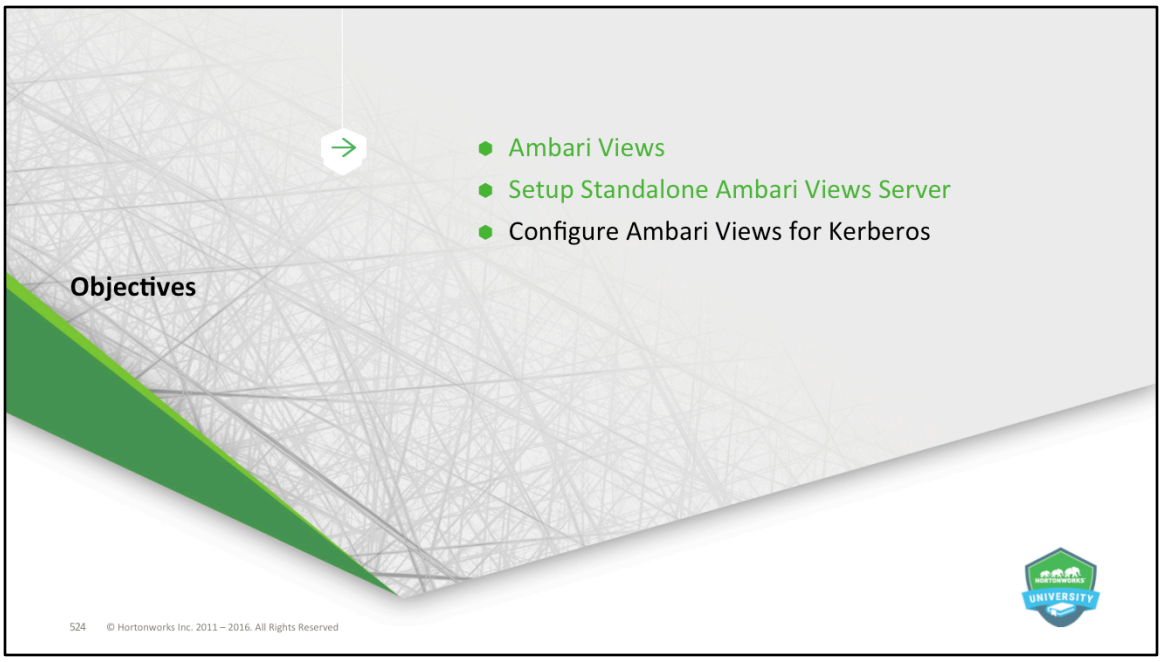
Setup Standalone Ambari View Server

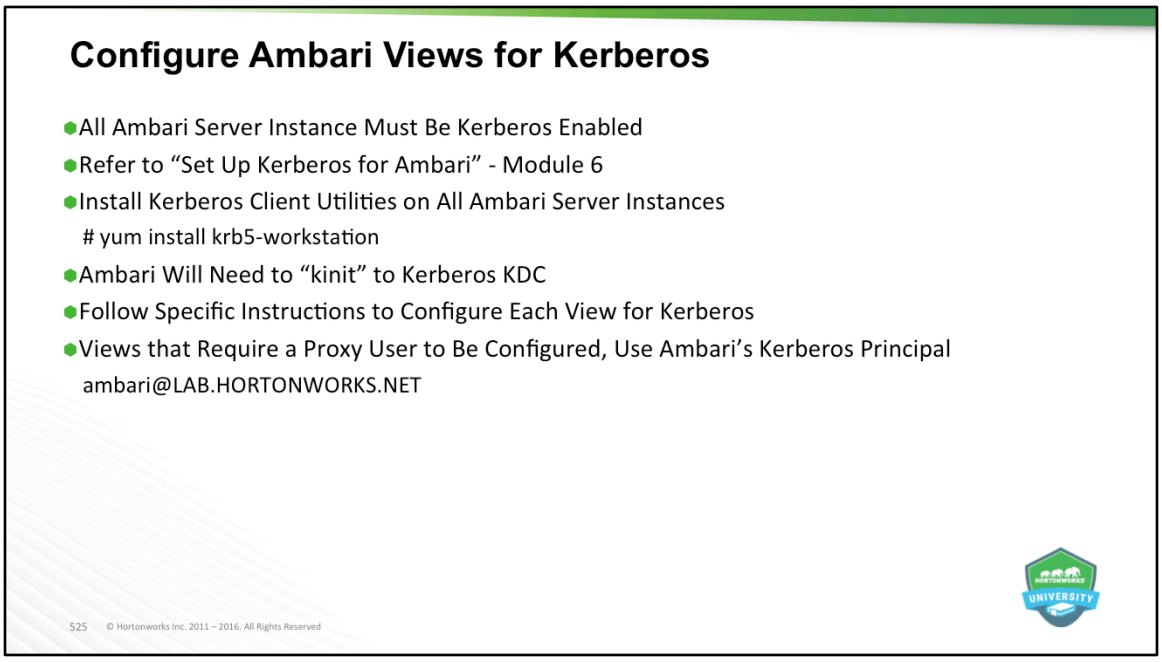
- Installation Overview
- Install Ambari-Server Package
- Execute Ambari Setup
- Configure LDAP/AD Authentication
- Configure Server for Non-Root
- Encrypt Database/Passwords
- Configure HTTPS/SSL
- Configure SPNEGO Authentication for Hadoop
- Deploy Views
- Create and Configure View Instances
- Repeat Process for Additional View Instances

Setup Standalone Ambari View Server

- Strongly Recommend to Increase Amount of Memory Available When Host Multiple Views
- Each View Requires It's Own Memory Footprint
- Increasing Maximum Allocable Memory Helps Support Multiple Views and Concurrent Use
- Recommended Values in /var/lib/ambari-server/ambari-env.sh

  AMBARI_JVM_ARGS=-Xmx4096m -XX:PermSize=128m -XX:MaxPermSize=128m
- Restart Ambari Server


Objectives
Ambari Views
Setup Standalone Ambari Views Server
Configure Ambari Views for Kerberos
524
© Hortonworks Inc. 2011 – 2016. All Rights Reserved



If the cluster your views will communicate with is Kerberos-enabled, you need to configure the Ambari Server instance(s) for Kerberos and be sure to configure the views to work with Kerberos.

Refer to the Set Up Kerberos for Ambari for the instructions on how to configure Ambari Server for Kerberos. Be sure to configure all standalone Ambari Server instances for Kerberos.

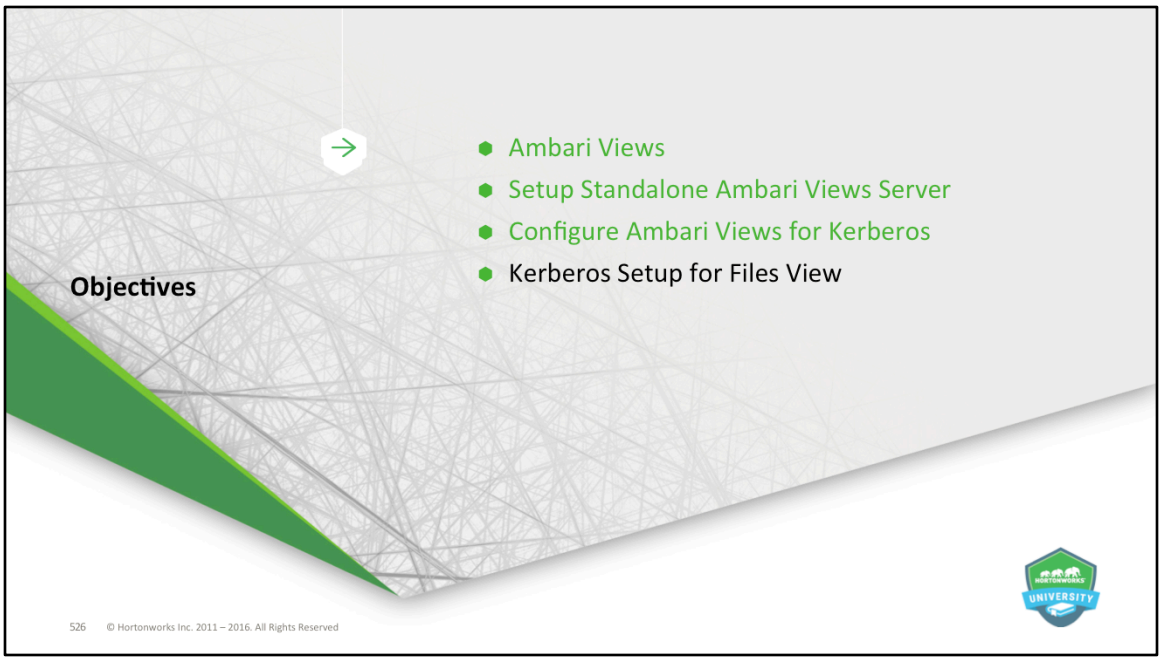
Important

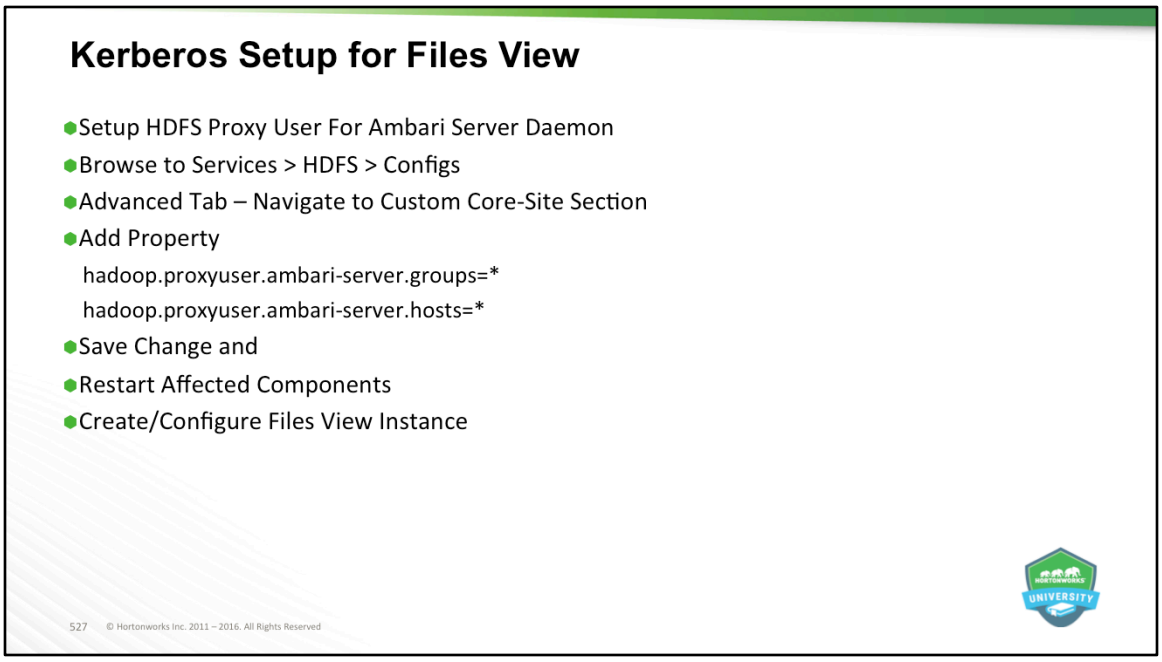
Be sure to install the Kerberos client utilities on the Ambari Server so that Ambari can kinit.

RHEL/CentOS/Oracle Linux

yum install krb5-workstation

SLES


Objectives
Ambari Views
Setup Standalone Ambari Views Server
Configure Ambari Views for Kerberos
Kerberos Setup for Files View
526
© Hortonworks Inc. 2011 – 2016. All Rights Reserved



To set up an HDFS proxy user for the Ambari Server daemon account, you need to configure the proxy user in the HDFS configuration. This configuration is determined by the account name the ambari-server daemon is running as. For example, if your ambari-server is running as root, you set up an HDFS proxy user for root with the following:

In Ambari Web, browse to Services > HDFS > Configs.

Under the Advanced tab, navigate to the Custom core-site section.

Click Add Property… to add the following custom properties:

if you have configured Ambari Server for Kerberos, be sure to modify this property name for the primary Kerberos principal user. For example, if ambari-server is setup for Kerberos using principal ambari-server@EXAMPLE.COM, you would use the following properties instead:

Kerberos Setup for Files View

- Configure File View Instance
- Browse to Ambari Administration Interface
- Click on Views – Expand File Views
- Setting and Cluster Configuration Enter Following Properties

Property	Description	Value
WebHDFS Username	This is the username the view will access HDFS as. Leave this default value intact to represent the authenticated view user.	${username}
WebHDFS Authorization	This is the semicolon-separated authentication configuration for WebHDFS access.	auto=KERBEROS; proxyuser=ambari-server

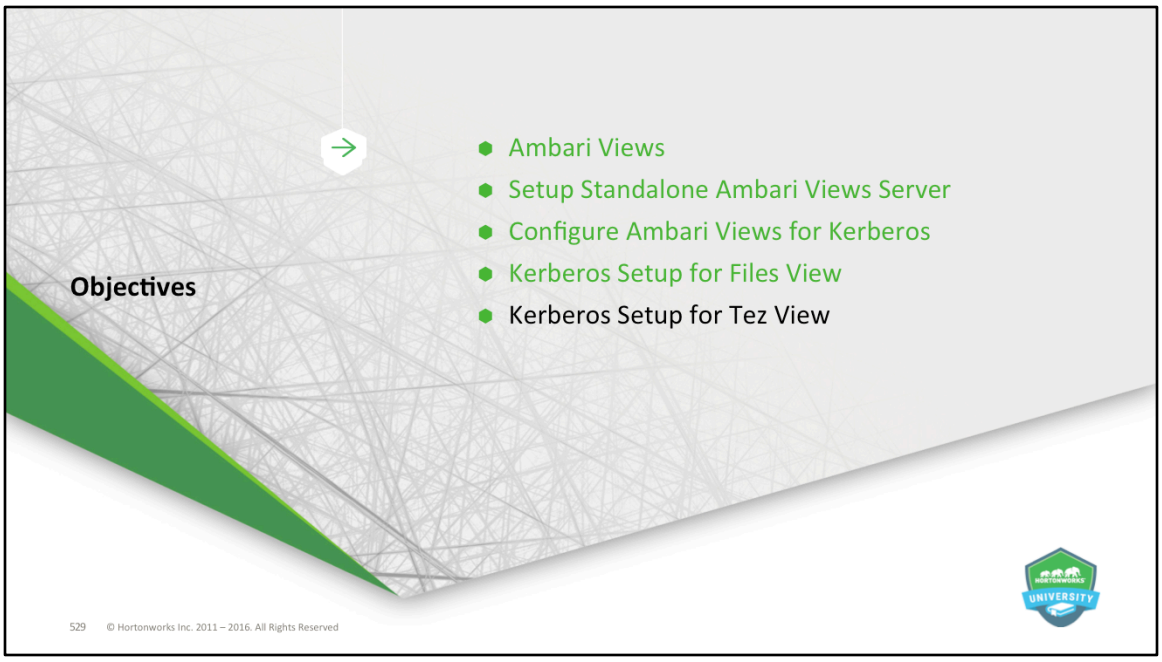
You must first set up Kerberos for Ambari by configuring the Ambari Server daemon with a Kerberos principal and keytab. Refer to Configuring Views for Kerberos for instructions. After you have set up Kerberos for Ambari, in the Settings section of the Files View, enter the following:

Property Description

Example Value

WebHDFS Username This is the username the view will access HDFS as. Leave this default value intact to represent the authenticated view user. ${username}

WebHDFS Authorization This is the semicolon-separated authentication configuration for WebHDFS access.


Objectives
Ambari Views
Setup Standalone Ambari Views Server
Configure Ambari Views for Kerberos
Kerberos Setup for Files View
Kerberos Setup for Tez View
529
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Kerberos Setup for Tez View

- Browse to Services > YARN > Configs
- Advanced Tab – Navigate to Custom Yarn-Site Section
- Add Property

Property	Value
yarn.timeline-server.http-authentication.proxyuser.**ambari-server**.hosts	*
yarn.timeline-server.http-authentication.proxyuser.**ambari-server**.user	*
yarn.timeline-server.http-authentication.proxyuser.**ambari-server**.groups	*
yarn.resourcemanager.webapp.delegation-token-auth-filter.enabled	true
yarn.resourcemanager.proxyuser.**ambari-server**.hosts	*
yarn.resourcemanager.proxyuser.**ambari-server**.users	*
yarn.resourcemanager.proxyuser.**ambari-server**.groups	*

- Restart Affected Components

yarn.resourcemanager.webapp.delegation-token-auth-filter.enabled

yarn.timeline-service.http-authentication.proxyuser.${ambari principal name}.hosts

yarn.timeline-service.http-authentication.proxyuser.${ambari principal name}.groups

yarn.timeline-service.http-authentication.proxyuser.${ambari principal name}.users

The following settings are optional, if proxyusers settings are added in core-site.xml:

yarn.resourcemanager.proxyuser.${ambari principal name}.hosts

yarn.resourcemanager.proxyuser.${ambari principal name}.users

yarn.resourcemanager.proxyuser.${ambari principal name}.groups

Kerberos Setup for Tez View

- Browse to Services > HDFS > Configs
- Advanced Tab – Navigate to Core-Site Section
- Verify Following Properties – Add As Necessary

Property	Value
hadoop.http.authentication.type	kerberos
hadoop.http.filter.initializers	org.apache.hadoop.security.AuthenticationFilterInitializer
hadoop.http.authentication.kerberos.keytab	/etc/security/keytabs/spnego.service.keytab
hadoop.http.authentication.kerberos.principal	HTTP/_HOST@HORTONWORKS.COM
hadoop.http.authentication.signature.secret.file	/etc/hadoop/conf/secret_http_file
hadoop.http.authentication.cookie.domain	hortonworks.com
hadoop.proxyuser.**ambari-server**.hosts	*
hadoop.proxyuser.**ambari-server**.groups	*
hadoop.proxyuser.**ambari-server**.users	*

Property Value

hadoop.http.authentication.type kerberos

hadoop.http.filter.initializers org.apache.hadoop.security.AuthenticationFilterInitializer

hadoop.http.authentication.kerberos.keytab <Path to keytab container HTTP principal>

for example: <etc/security/keytabs/spnego.service.keytab>

hadoop.http.authentication.kerberos.principal HTTP/_HOST@REALM

for example: HTTP/_HOST@EXAMPLE.COM

Kerberos Setup for Tez View

- Restart Affected Components
- Create/Configure Tez View Instance

Property
Value

hadoop.http.authentication.type kerberos

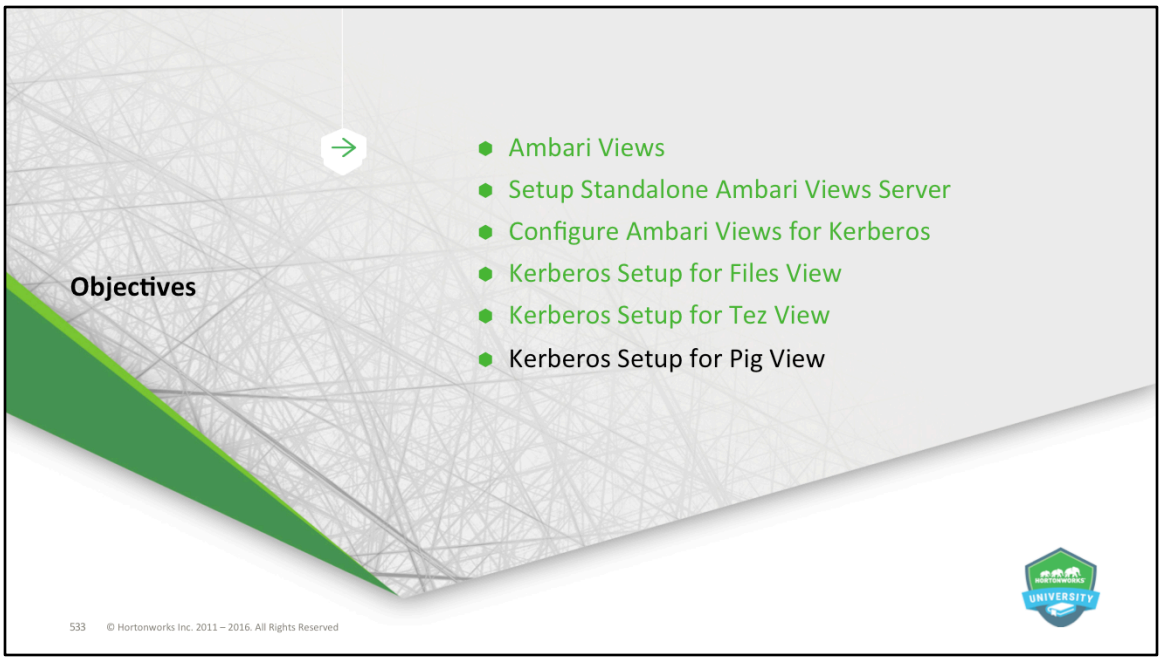
hadoop.http.filter.initializers
org.apache.hadoop.security.AuthenticationFilterInitializer

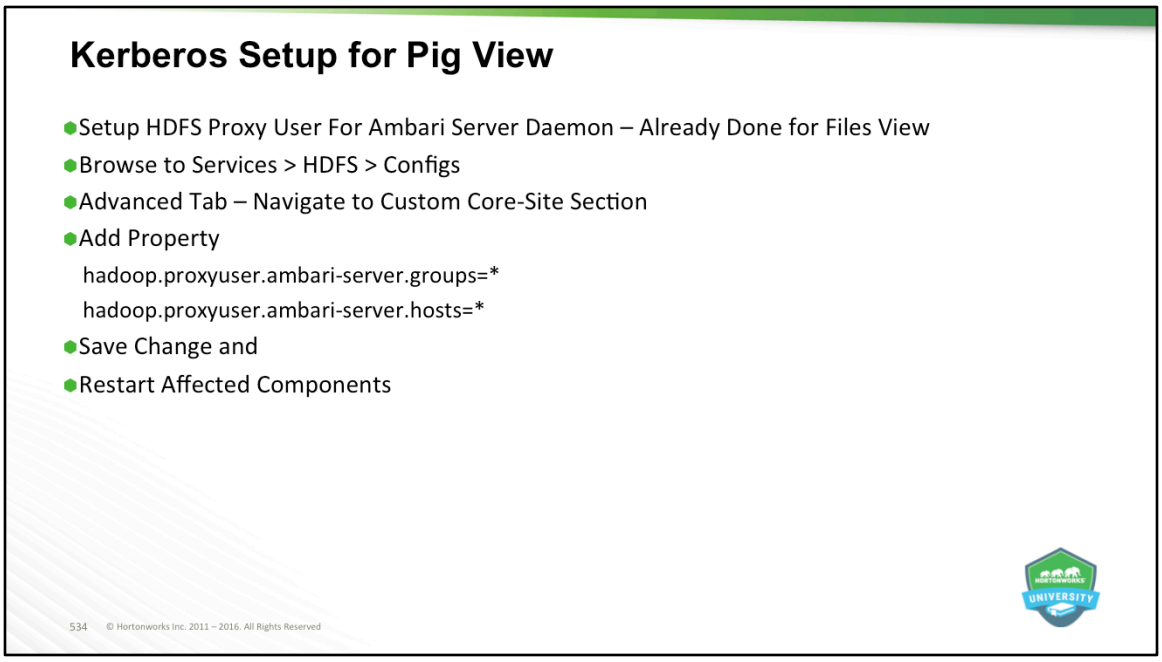
hadoop.http.authentication.kerberos.keytab <Path to keytab container HTTP principal>

for example: <etc/security/keytabs/spnego.service.keytab>

hadoop.http.authentication.kerberos.principal HTTP/_HOST@REALM

for example: HTTP/_HOST@EXAMPLE.COM


Objectives
Ambari Views
Setup Standalone Ambari Views Server
Configure Ambari Views for Kerberos
Kerberos Setup for Files View
Kerberos Setup for Tez View
Kerberos Setup for Pig View
533
© Hortonworks Inc. 2011 – 2016. All Rights Reserved



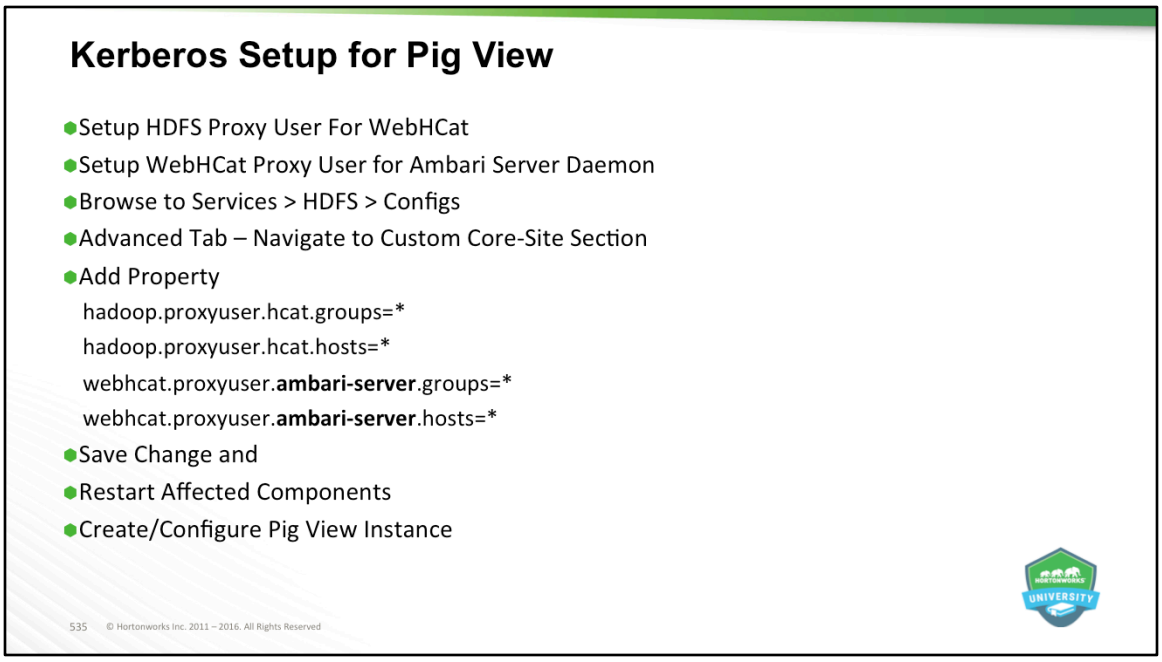
To set up an HDFS proxy user for the Ambari Server daemon account, you need to configure the proxy user in the HDFS configuration. This configuration is determined by the account name the ambari-server daemon is running as. For example, if your ambari-server is running as root, you set up an HDFS proxy user for root with the following:

In Ambari Web, browse to Services > HDFS > Configs.

Under the Advanced tab, navigate to the Custom core-site section.

Click Add Property… to add the following custom properties:

if you have configured Ambari Server for Kerberos, be sure to modify this property name for the primary Kerberos principal user. For example, if ambari-server is setup for Kerberos using principal ambari-server@EXAMPLE.COM, you would use the following properties instead:



You must set up an HDFS proxy user for WebHCat and a WebHCat proxy user for the Ambari Server daemon account.

To setup the HDFS proxy user for WebHCat :

In Ambari Web, browse to Services > HDFS > Configs.

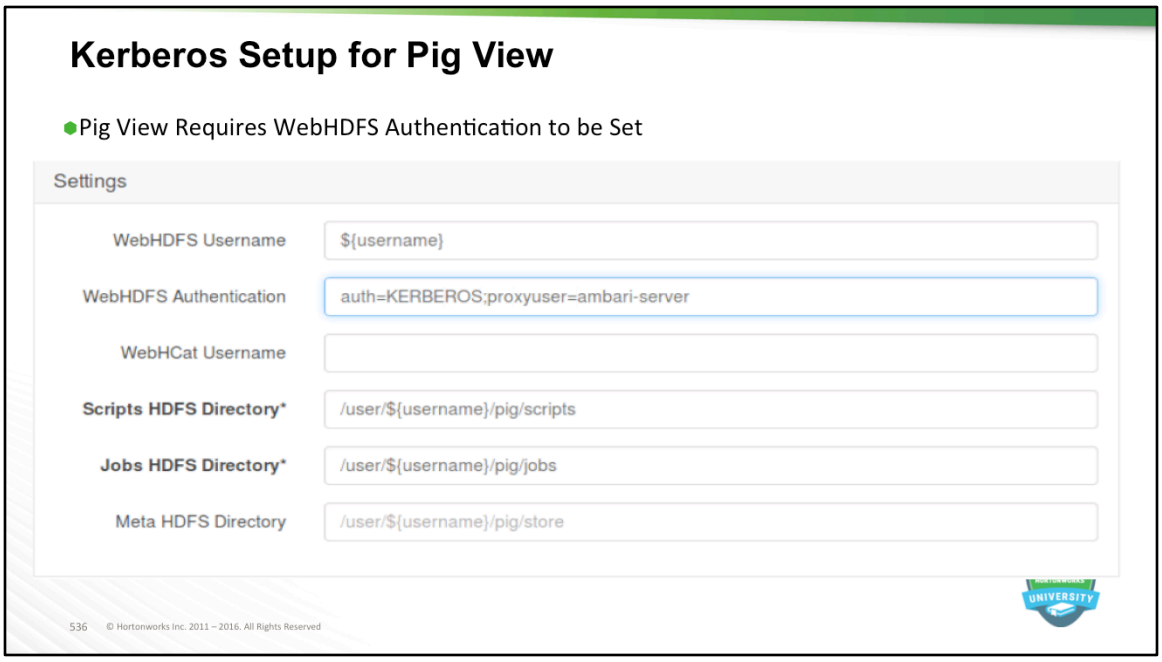
Under the Advanced tab, navigate to the Custom core-site section.

Click Add Property… to add the following custom properties:

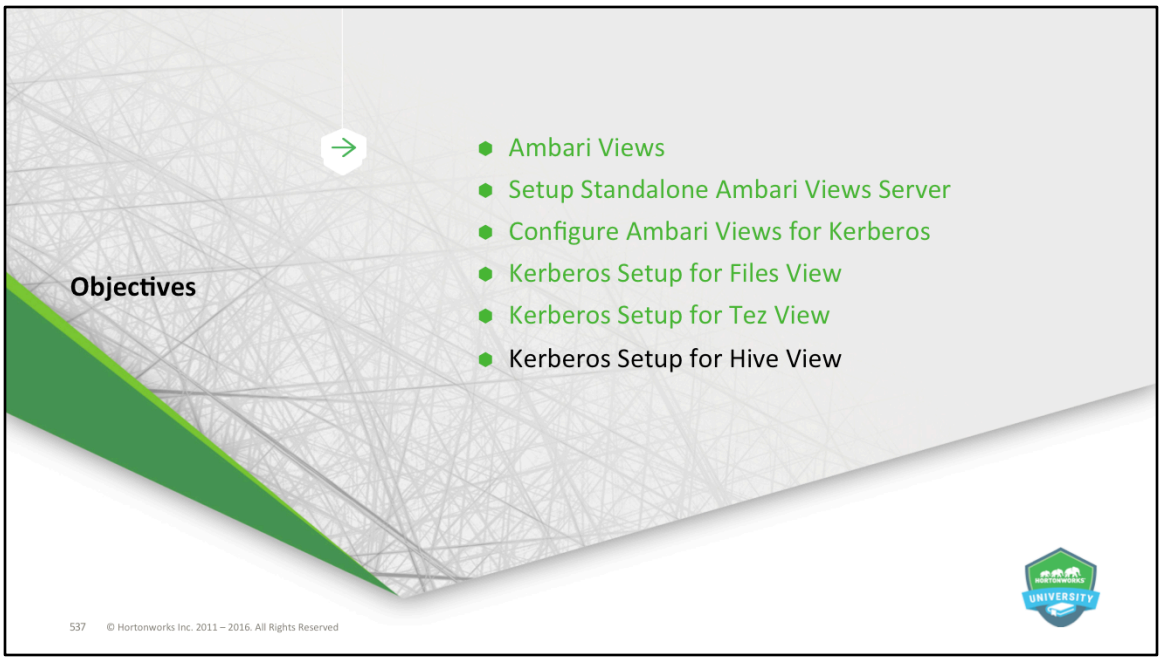
hadoop.proxyuser.hcat.groups=*

hadoop.proxyuser.hcat.hosts=*

Save the configuration change and restart the required components as indicated by Ambari.



After you have set up basic Kerberos for the Pig View, Pig requires that WebHDFS Authentication be set to auth=KERBEROS;proxyuser=<ambari-user-principal>

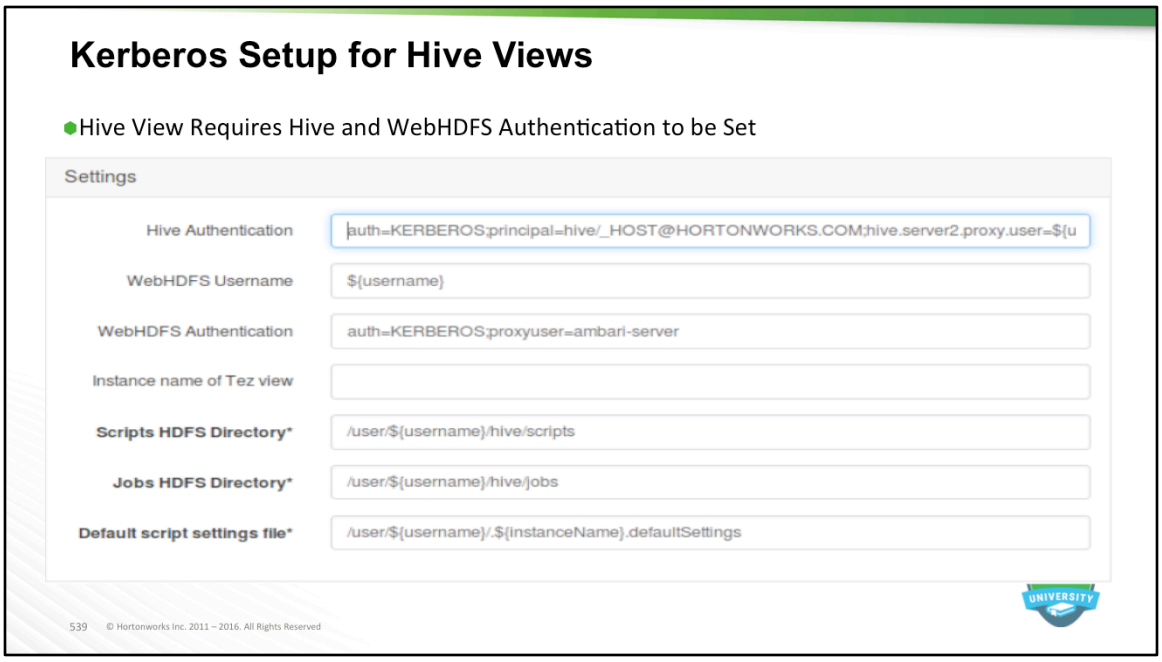

Objectives
Ambari Views
Setup Standalone Ambari Views Server
Configure Ambari Views for Kerberos
Kerberos Setup for Files View
Kerberos Setup for Tez View
Kerberos Setup for Hive View
537
© Hortonworks Inc. 2011 – 2016. All Rights Reserved

Kerberos Setup for Hive View

- Setup HDFS Proxy User For Ambari Server Daemon – Already Done for Files View
- Browse to Services > HDFS > Configs
- Advanced Tab – Navigate to Custom Core-Site Section
- Add Property

  hadoop.proxyuser.ambari-server.groups=*

  hadoop.proxyuser.ambari-server.hosts=*
- Save Change and
- Restart Affected Components
- Create/Configure Hive View Instance



After you have set up basic Kerberos for the Hive View, Hive requires the following additional settings:

Table 8.4. Kerberos Settings for Hive Views

Property	Value
WebHDFS Authentication	auth=KERBEROS;proxyuser=<ambari-principal>
Hive Authentication	KERBEROS and the principal is set to the same principal that is specified in hive-site.xml for hive.server2.authentication.kerberos.principal. For example:

Knowledge Check

Knowledge Check

1. True/False - Ambari Servers does not include a set of built-in views?
2. What is the purpose of a standalone Ambari Views server?
3. True/False - Ambari Views Server can not communicate with a Kerberos enabled Hadoop cluster?
4. True/False - Multiple Ambari Views Server can be setup behind a reverse proxy?
5. True/False - The Ambari Views Server are used to manage the Hadoop cluster?

1. FALSE

2. Used to provide access to user who will access views but will not and should not have access to the operational Ambari Server.

3. FALSE - The Ambari Views Server requires Kerberos configuration as well, in order to communicate with Kerberized Hadoop Cluster.

4. TRUE

5. FALSE - Used to provide access to user.

Summary

Summary

- Ambari Views allow developers to create UI components that plug into the Ambari Web interface.
- Ambari Server includes a set of built-in views that are pre-deployed with your cluster.
- Views can be deployed and managed in the "operational" Ambari server that is managing your cluster however, a standalone Ambari Views server is useful when users who will access views will not and should not have access to the operational Ambari server.
- Ambari Views Server requires Kerberos configuration if communicating with a Kerberos-enabled cluster.

Lab: Other Security Features for Ambari