Security Administration

Lab Guide

Rev 1.1.0

HORTONWORKS UNIVERSITY

Table of Contents

Contents

Lab: Setting up the Environment 3
Lab: Configure AD Resolution and Certificate 13
Lab: Security Options for Ambari 14
Lab: Kerberize the Cluster 22
Lab: Ranger Install 35
Lab: Ranger KMS/Data Encryption Setup 46
Lab: Secured Hadoop Exercises 64
Lab: Knox 85
Lab: Other Security Features for Ambari 97
Appendix – Install SolrCloud 102

Lab: Setting up the Environment

About This Lab

Objective:	Setup the environment and login to the cluster
File locations:	N/A
Successful outcome:	You will have installed the Ambari components and connected to the HDFS cluster
Before you begin	Get the SSH credentials

Lab Steps

Lab Environment Architecture:

There are a total of 4 nodes/Virtual machines that we will be using for the lab section of this course.

- The HDP cluster will consist of 3 nodes
  - 1 x Node/Virtual Machine pre - installed with the Ambari Server and various master/client services
  - 2 x Nodes/Virtual Machines contain various master/client services
- 1 x Node/Virtual Machine pre-installed with the Active Directory Server

Perform the following steps:

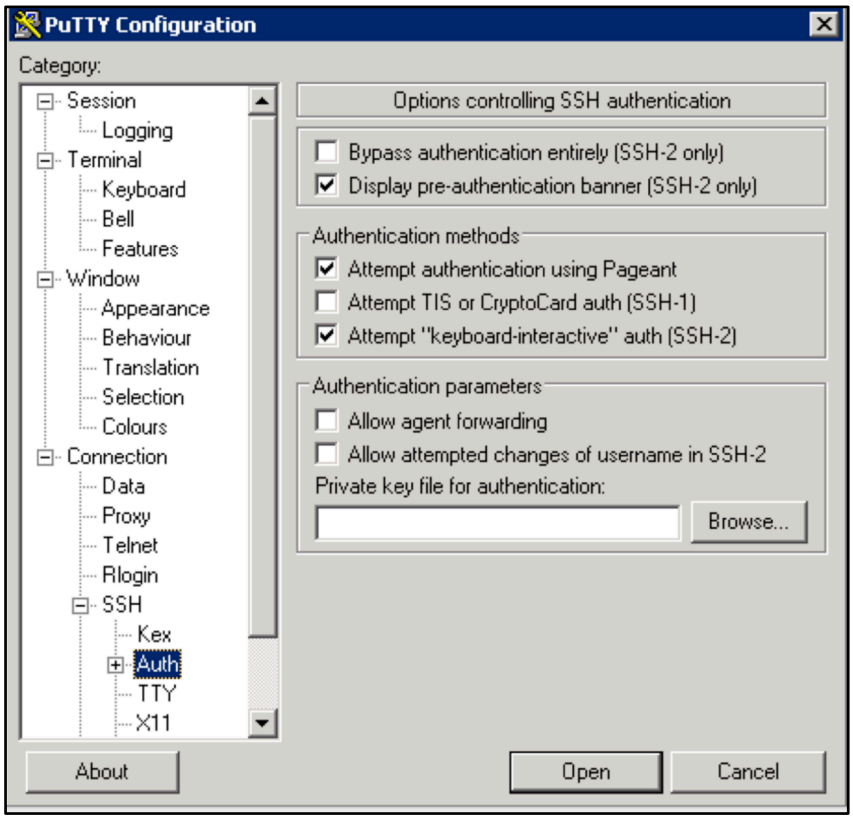
1. To connect using Putty from Windows laptop:
   a. Right click to download this ppk key from this URL:
   https://raw.githubusercontent.com/HortonworksUniversity/Ops_Labs/1.1.0/training-keypair.ppk

      Save link as > save to Downloads folder

   b. Use putty to connect to your node using the ppk key

      ```
      Connection > SSH > Auth > Private key for authentication >
      Browse... > Select training-keypair.ppk Image
      ```

      ```
      Connection > Data > Auto-login username > Enter "centos"
      ```



Make sure to click `Save` on the session page before logging in

2. To connect from Linux/MacOSX laptop/ SSH into Ambari node of your cluster by following the below steps:

   a. **Right click to down load this pem key from this URL**

      https://raw.githubusercontent.com/HortonworksUniversity/Ops_Labs/1.1.0/training-keypair.pem

      Save link as > save to Downloads folder

   b. **On a new terminal copy pem key to `~/.ssh dir` and correct permissions:**

```
cp ~/Downloads/training-keypair.pem ~/.ssh/
chmod 400 ~/.ssh/training-keypair.pem
```

   c. **Login to the Ambari node of the cluster you have been assigned by replacing `IP_ADDRESS_OF_AMBARI_NODE` below with Ambari node IP Address:**

```
ssh -i  ~/.ssh/training-keypair.pem centos@IP_ADDRESS_OF_AMBARI_NODE
```

   d. **Change user to `root`:**

```
sudo su -
```

Note: Similarly open two new terminal windows, login via SSH to each of the other nodes in your cluster, as you will need to run commands on each node in future labs

TIP: Since in the other labs you may be required to run the same set of commands on each of the cluster hosts, now would be a good time to setup your favorite tool to do so.
See:
https://www.reddit.com/r/sysadmin/comments/3d8aou/running_linux_commands_on_multiple_servers/

On OSX, an easy way to do this is to use iTerm (https://www.iterm2.com/): open multiple tabs/splits and then use `Broadcast input` feature (under `Shell -> Broadcast input to all Panes in All Tabs -> click ok)`

If you are not already familiar with such a tool, you can also just run the commands on the cluster, one host at a time.

3. Login to Ambari

   a. **Login to Ambari web UI by opening** `http://AMBARI_PUBLIC_IP:8080` **and log in with** `admin/BadPass#1` on the ambari server node.

      Note: You will see a list of Hadoop components running on your cluster on the left side of the page

   b. **They should all show green (i.e. started) status. If not, start them by Ambari via** `Service Actions` **menu for that service**

4. Finding internal/external hosts

   Following are useful techniques you can use in future labs to find your cluster specific details:

   a. **From SSH terminal, how can I find the cluster name?**

      - **Run the below commands on the ambari node to fetch cluster name via Ambari API**

        Password: BadPass#1

```
 output=`curl -u admin:$PASSWORD -i -H 'X-Requested-By: ambari'
http://localhost:8080/api/v1/clusters`
 cluster=`echo $output | sed -n 's/.*"cluster_name" : "\([^\"]*\)".*/\1/p'`

 echo $cluster
```

   b. **From SSH terminal, how can I find internal hostname (aka FQDN) of the node I'm logged into?**

```
hostname -f
ip-172-30-0-186.us-west-2.compute.internal
```

c. From SSH terminal, how can I to find external hostname of the node I'm logged into?

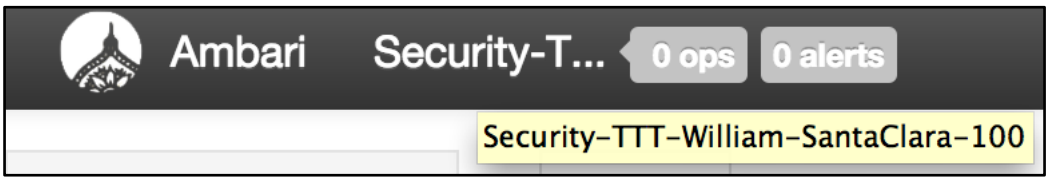
```
curl icanhazptr.com
ec2-52-33-248-70.us-west-2.compute.amazonaws.com
```

d. From SSH terminal, how can I to find external (public) IP of the node I'm logged into?

```
curl icanhazip.com
54.68.246.157
```

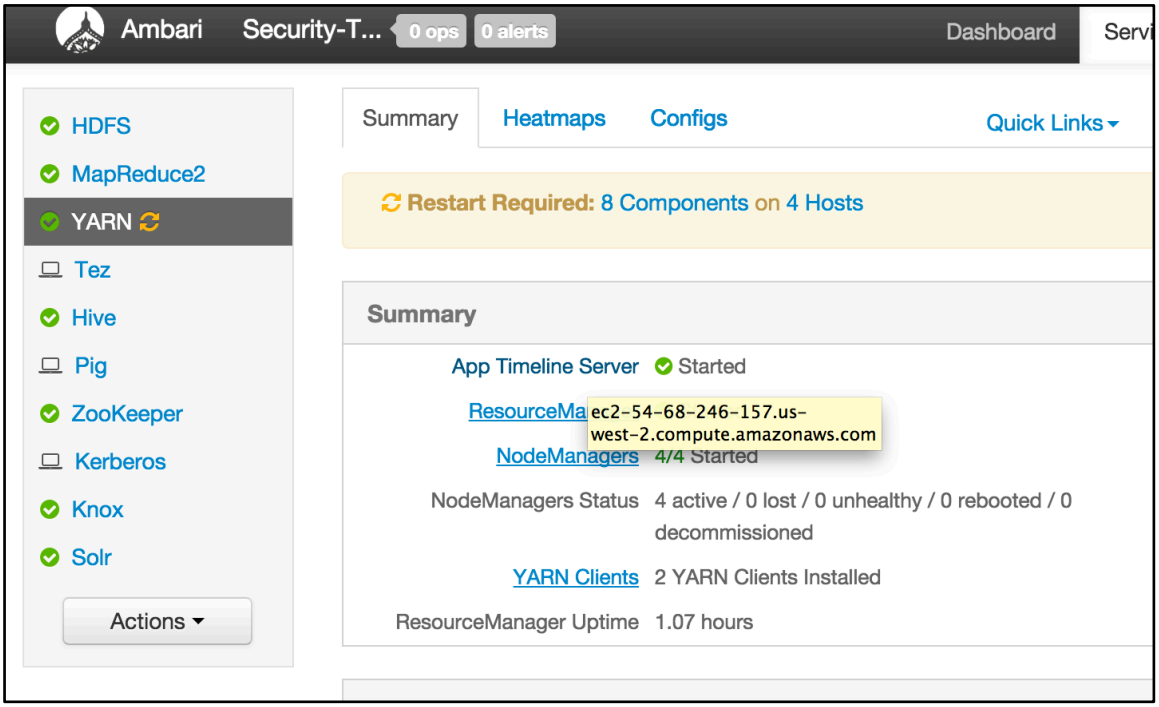
e. From Ambari web UI how do I check the cluster name?

- It is displayed on the top left of the Ambari dashboard, next to the Ambari logo. If the name appears truncated, you can hover over it to produce a helptext dialog with the full name



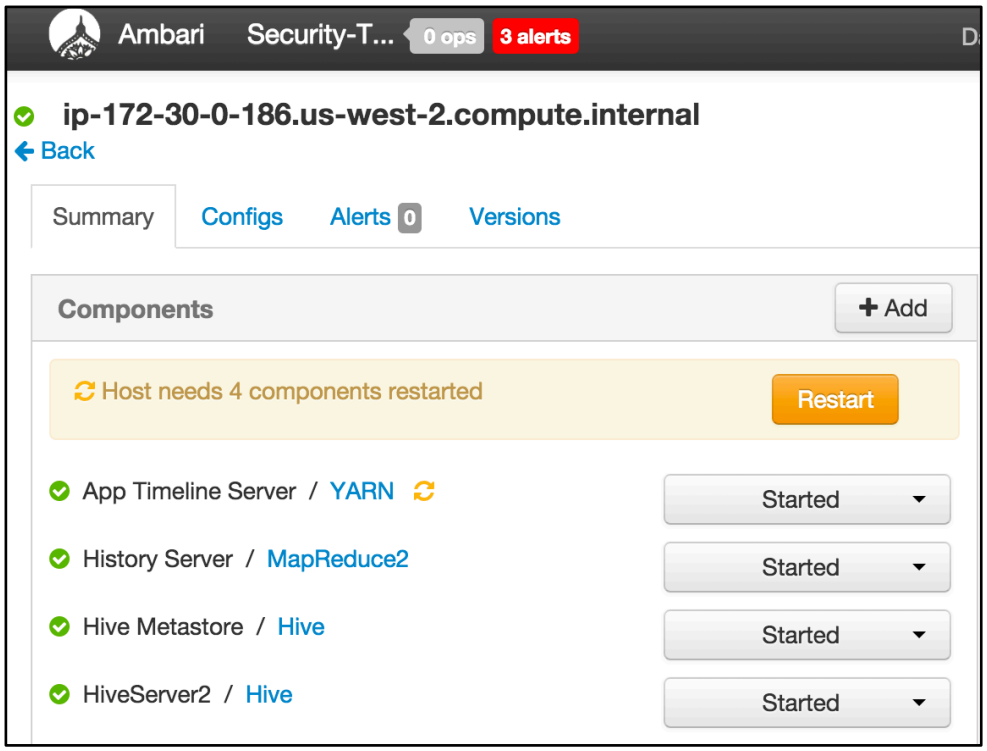
f. From Ambari how can I find external hostname of node where a component (e.g. Resource Manager or Hive) is installed?

- Click the parent service (e.g. YARN) and hover over the name of the component. The external hostname will appear.



g. From Ambari how can I find internal hostname of node where a component (e.g. Resource Manager or Hive) is installed?

- Click the parent service (e.g. YARN) and click on the name of the component. It will take you to hosts page of that node and display the internal hostname on the top.



Note: In future labs you may need to provide private or public hostname of nodes running a particular component (e.g. YARN RM or Mysql or HiveServer).

5. Import sample data into Hive by running the below commands on the node where HiveServer2 is installed to download data and import it into a Hive table for later labs

   Hint: You can find the node using Ambari

   a. **Download all raw data required for all the labs by running the below command on all 3 nodes.**

```
cd /tmp
sudo wget https://github.com/HortonworksUniversity/Ops_Labs/archive/1.1.0.zip
sudo unzip 1.1.0.zip
sudo mv /tmp/Ops_Labs-1.1.0 /tmp/Ops_Labs
sudo mv /tmp/Ops_Labs/labfiles /
```

   b. **Create user** `dir` **for** `admin`, `sales1` **and** `hr1`

```
sudo -u hdfs hadoop fs  -mkdir /user/admin
sudo -u hdfs hadoop fs  -chown admin:hadoop /user/admin

sudo -u hdfs hadoop fs  -mkdir /user/sales1
sudo -u hdfs hadoop fs  -chown sales1:hadoop /user/sales1

sudo -u hdfs hadoop fs  -mkdir /user/hr1
sudo -u hdfs hadoop fs  -chown hr1:hadoop /user/hr1
```

c. **Now create Hive table in default database**

- **Start beeline shell from the node where Hive is installed:**

```
beeline -u "jdbc:hive2://localhost:10000/default"
```

- **At beeline prompt, run below:**

```
CREATE TABLE `sample_07` (
`code` string ,
`description` string ,
`total_emp` int ,
`salary` int )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TextFile;

load data local inpath '/labfiles/security/sample_07.csv' into table sample_07;

CREATE TABLE `sample_08` (
`code` string ,
`description` string ,
`total_emp` int ,
`salary` int )
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TextFile;

load data local inpath '/labfiles/security/sample_08.csv' into table sample_08;
```

Note: Notice that in the JDBC connect string for connecting to an unsecured Hive while it's running in default (i.e. binary) transport mode:

- **Port is 10000**
- **No Kerberos principal was needed**

This will change after we:

- **Enable Kerberos**
- **Configure Hive for HTTP transport mode (to go through Knox)**

Note: Exit beeline shell by entering: `exit;`

6. Why is security needed?

a. **HDFS Access On Unsecured Cluster**

- **On your unsecured cluster try to access a restricted `dir` in HDFS:**

```
hdfs dfs -ls /tmp/hive

## this should fail with Permission Denied
```

- **Now try again after setting `HADOOP_USER_NAME env var`:**

```
export HADOOP_USER_NAME=hdfs

hdfs dfs -ls /tmp/hive

## this shows the file listing!
```

Note: If you unset the `env var` and it will fail again:

```
unset HADOOP_USER_NAME
hdfs dfs -ls /tmp/hive
```

b. **WebHDFS Access On Unsecured Cluster**

- **From node running NameNode, make a WebHDFS request using below command:**

```
curl -sk -L "http://$(hostname -f):50070/webhdfs/v1/user/?op=LISTSTATUS"
```

Note: In the absence of Knox, notice it goes over HTTP (not HTTPS) on port 50070 and no credentials were needed

c. **Web UI Access On Unsecured Cluster**

- **From Ambari, notice you can open the WebUIs without any authentication:**

i. `HDFS > Quicklinks > NameNode UI`

ii. `Mapreduce > Quicklinks > JobHistory UI`

iii. `YARN > Quicklinks > ResourceManager UI`

Note: This should tell you why Kerberos (and other security) is needed on Hadoop

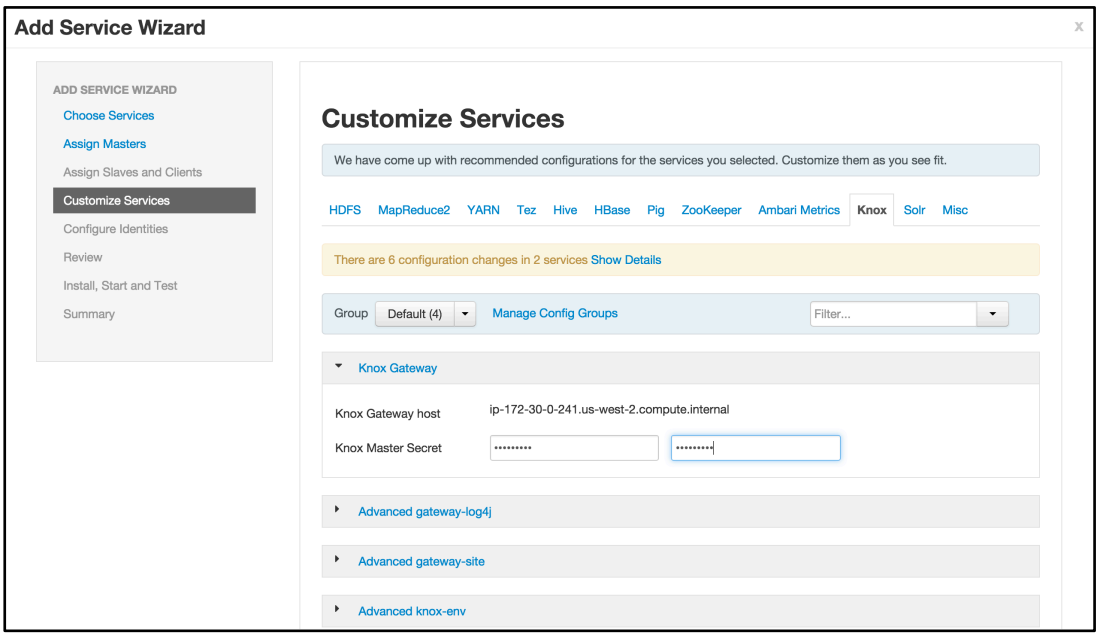
7. Install Additional Components

a. **Install Knox Via Ambari by logging into Ambari web UI by opening** `http://AMBARI_PUBLIC_IP:8080` **and log in with** `admin/BadPass#1`

b. **Use the** `Add Service` **Wizard to install Knox (if not already installed) on any node in the cluster**

c. **When prompted for the** `Knox Master Secret`**, set it to** `knox`

Note: Do not use password with special characters (like `#`**,** `$` **etc) here as it seems beeline has problems with it**



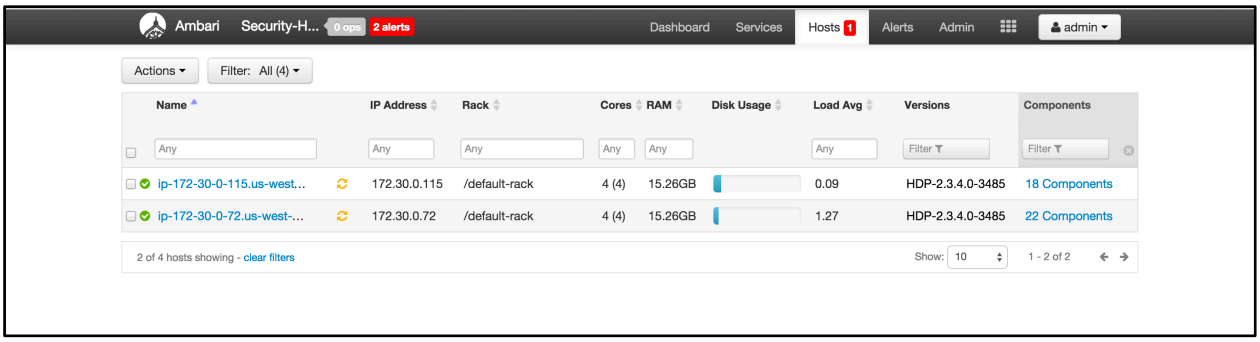
d. Click `Next > Proceed Anyway > Deploy` to accept all defaults

Note: After the install is completed, Ambari will show that a number of services need to be restarted. Ignore this for now; we will restart them at a later stage.

e. Ensure Tez is installed on all nodes where Pig clients are installed. This is done to ensure Pig service checks do not fail later on.

`Ambari > Pig >` click the `Pig clients` link

Note: This tell us which nodes have Pig clients installed:

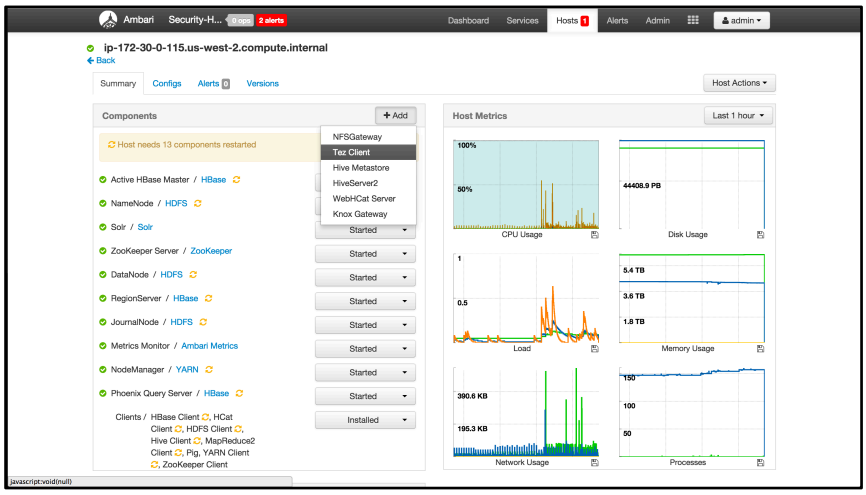


f. For each node that has Pig installed:

- Click on the hyperlink of the node name to view that shows all the services running on that particular node

g. Click `+Add` and select `Tez client > Confirm add`

Note: If `Tez client` does not appear in the list, it is already installed on this host, so you can skip this host Image



RESULT

The environment is now setup and connected to HDFS.

Lab: Configure AD Resolution and Certificate

About This Lab

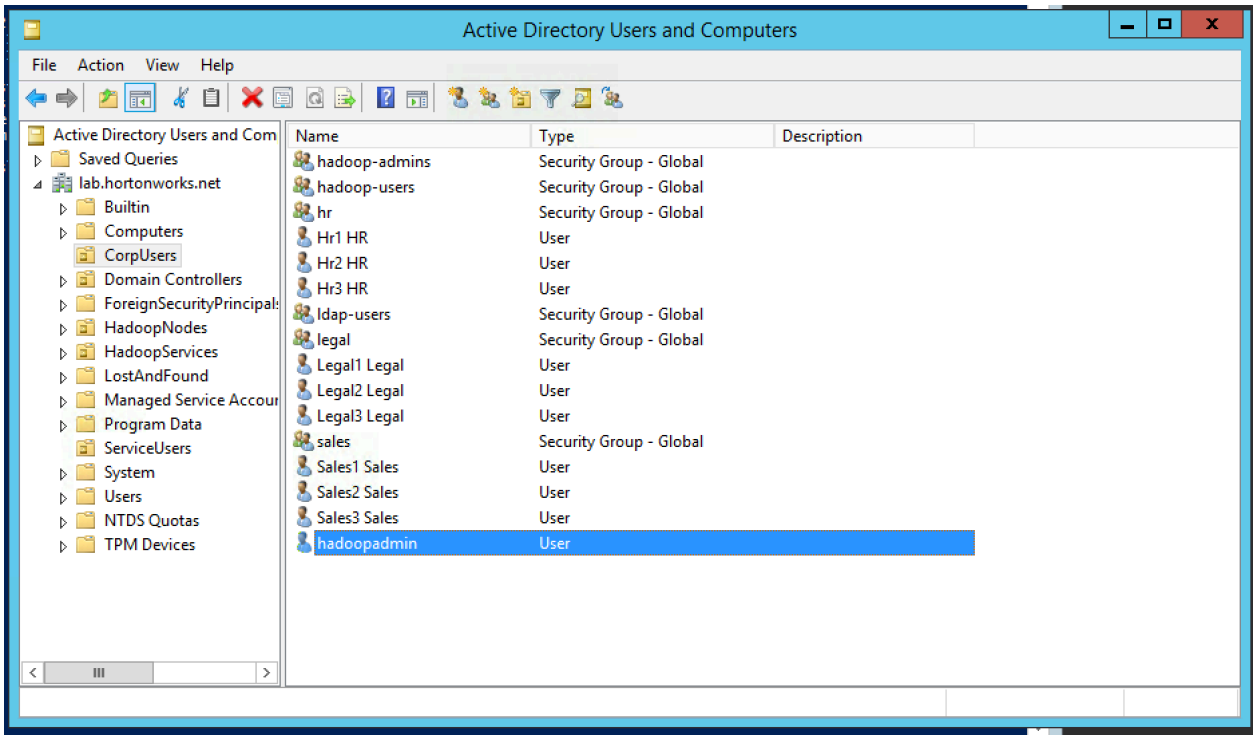
Objective:	Configure name resolution & certificate to AD
File locations:	N/A
Successful outcome:	Successfully configured name resolution and certificate to AD
Before you begin	N/A

Overview

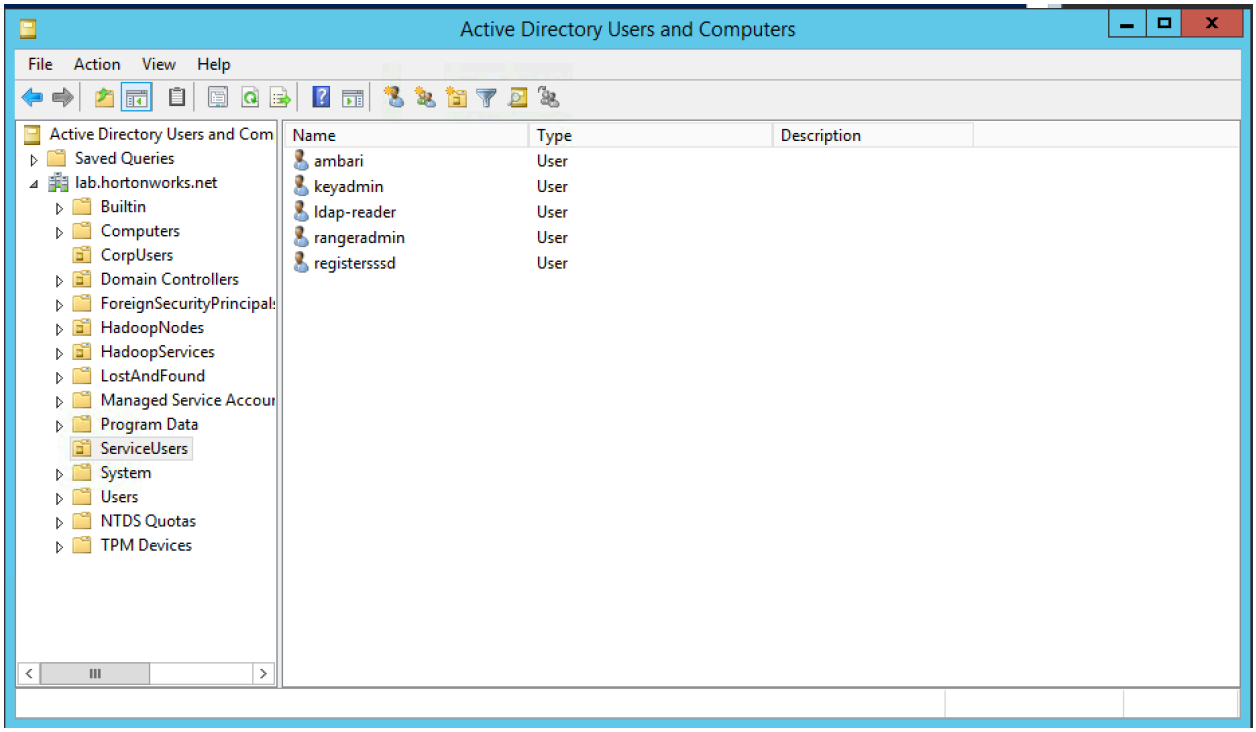
Active Directory will already be setup by the instructor. A basic structure of `OrganizationalUnits` will have been pre-created to look something like the below:

- `CorpUsers` OU, which contains:

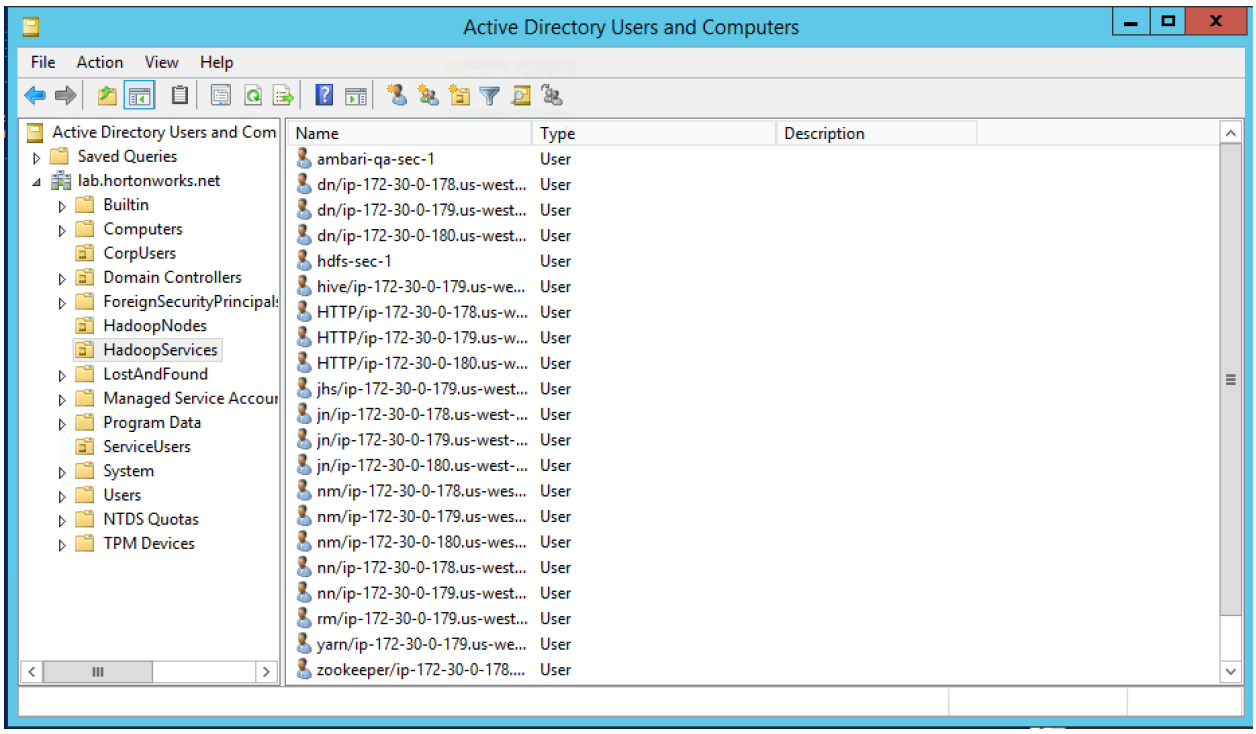
  business users and groups (e.g. `it1`, `hr1`, `legal1`) and `hadoopadmin`: Admin user (for AD, Ambari, ...)



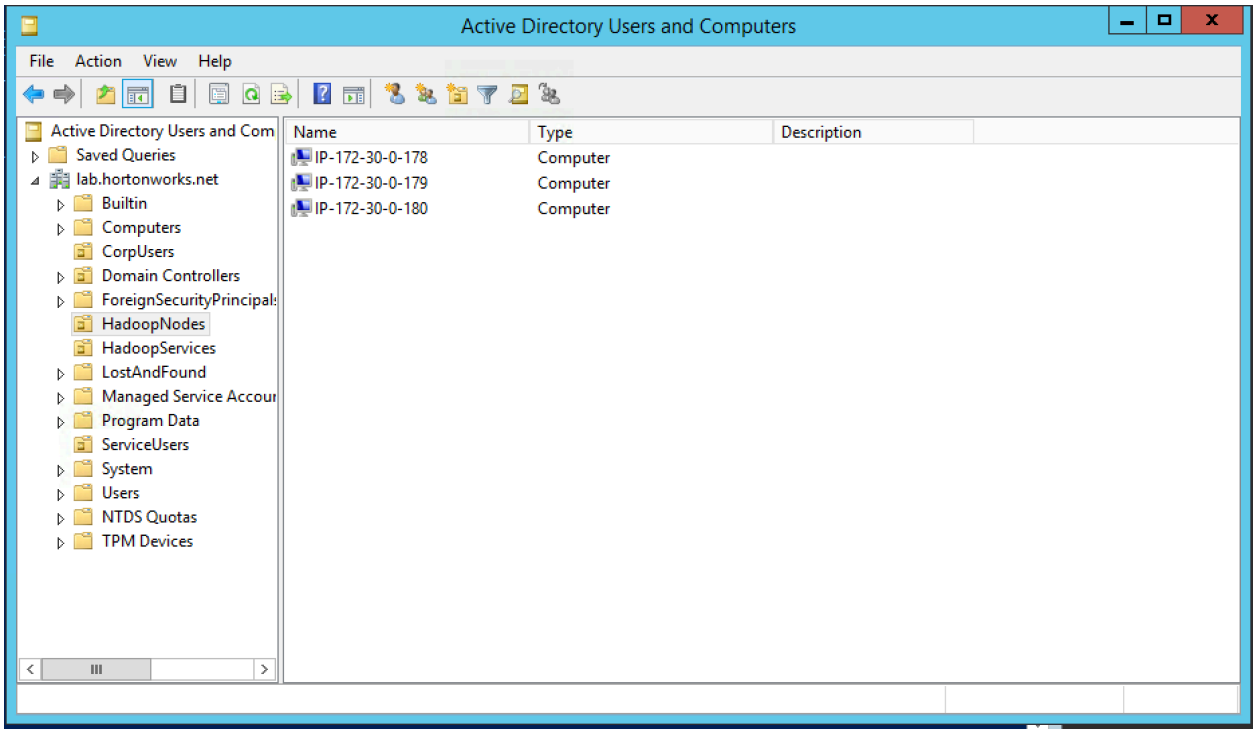
- `ServiceUsers` OU: service users - that would not be created by Ambari (e.g. `rangeradmin`, `ambari` etc)



- `HadoopServices` **OU: Hadoop service principals (will be created by Ambari)**



- `HadoopNodes` **OU: list of nodes registered with AD**



In addition, the below steps would have been completed in advance per this doc:
`http://docs.hortonworks.com/HDPDocuments/Ambari-2.2.0.0/bk_Ambari_Security_Guide/content/_use_an_existing_active_directory_domain.html`

`Notes:`

- Ambari Server and cluster hosts have network access to, and be able to resolve the DNS names of, the Domain Controllers.
- Active Directory secure LDAP (LDAPS) connectivity has been configured.
- Active Directory User container for principals has been created and is on-hand. For example: `ou=HadoopServices,dc=lab,dc=hortonworks,dc=net`
- Active Directory administrative credentials with delegated control of `Create, delete, and manage user accounts` on the previously mentioned User container are on-hand. e.g. `hadoopadmin`

For general info on Active Directory refer to Microsoft website:
`https://technet.microsoft.com/en-us/library/cc780336(v=ws.10).aspx`

Lab Steps

Perform the following steps:

Note: Run below steps on all nodes

1. **Configure name resolution & certificate to Active Directory**

   a. Add your Active Directory's internal IP to `/etc/hosts` (if not in DNS). Make sure you replace the IP address of your Active Directory for these labs.

      - Change the IP to match your ADs internal IP

```
ad_ip=GET_THE_AD_IP_FROM_YOUR_INSTRUCTOR

echo "${ad_ip} ad01.lab.hortonworks.net ad01" | sudo tee -a /etc/hosts
```

   b. Add your CA certificate (if using self-signed & not already configured)

      - In this case we have pre-exported the CA cert from our AD and made available for download.

```
sudo yum -y install openldap-clients ca-certificates

sudo cp /labfiles/security/extras/ca.crt /etc/pki/ca-trust/source/anchors/hortonworks-net.crt

sudo update-ca-trust force-enable

sudo update-ca-trust extract
sudo update-ca-trust check
```

   c. Test certificate & name resolution with `ldapsearch`

```
## Update ldap.conf with our defaults

 sudo tee -a /etc/openldap/ldap.conf > /dev/null << EOF
TLS_CACERT /etc/pki/tls/cert.pem
URI ldaps://ad01.lab.hortonworks.net ldap://ad01.lab.hortonworks.net
BASE dc=lab,dc=hortonworks,dc=net
EOF

##test connection to AD using openssl client

 openssl s_client -connect ad01:636 </dev/null

## test connection to AD using ldapsearch (when prompted for password, enter:
BadPass#1)

 ldapsearch -W -D ldap-reader@lab.hortonworks.net
```

   d. Make sure to repeat the above steps on all nodes

RESULT

You have successfully configured name resolution and certificate to AD

Lab: Security Options for Ambari

About This Lab

Objective: Setup Kerberos for Ambari

File locations: N/A

Successful outcome: Successfully sync Ambari and AD servers

Before you begin Reference document found at:
`http://docs.hortonworks.com/HDPDocuments/Ambari-2.2.0.0/bk_Ambari_Security_Guide/content/ch_amb_sec_guide.html`

Related lesson: N/A

Lab Steps

Perform the following steps:

1. Ambari server as non-root

a. Create a user for the Ambari Server if it does not exists

```
useradd -d /var/lib/ambari-server -G hadoop -M -r -s /sbin/nologin ambari
```

b. Otherwise – Update the Ambari Server user with the following

```
usermod -d /var/lib/ambari-server -G hadoop -s /sbin/nologin ambari
```

c. Grant the user 'sudoers' rights. This is required for Ambari Server to create it's Kerberos keytabs. You can remove this after kerberizing the cluster

```
echo 'ambari ALL=(ALL) NOPASSWD:SETENV: /bin/mkdir, /bin/cp, /bin/chmod, /bin/rm' > /etc/sudoers.d/ambari-server
```

d. To setup Ambari server as non-root run below on Ambari-server node:

```
sudo ambari-server setup
```

e. Then enter the below at the prompts:

i. `OK to continue? y`

ii. `Customize user account for ambari-server daemon? y`

iii. `Enter user account for ambari-server daemon (root):ambari`

iv. `Do you want to change Oracle JDK y/n? n`

v. `Enter advanced database configuration y/n? n`

Sample output:

```
sudo ambari-server setup
Using python  /usr/bin/python2
Setup ambari-server
Checking SELinux...
SELinux status is 'enabled'
SELinux mode is 'permissive'
WARNING: SELinux is set to 'permissive' mode and temporarily
disabled.
OK to continue [y/n] (y)? y
Customize user account for ambari-server daemon [y/n] (n)? y
Enter user account for ambari-server daemon (root):ambari
Adjusting ambari-server permissions and ownership...
Checking firewall status...
Redirecting to /bin/systemctl status  iptables.service

Checking JDK...
Do you want to change Oracle JDK [y/n] (n)? n
Completing setup...
Configuring database...
Enter advanced database configuration [y/n] (n)? n
Configuring database...
Default properties detected. Using built-in database.
Configuring ambari database...
Checking PostgreSQL...
Configuring local database...
Connecting to local database...done.
Configuring PostgreSQL...
Backup for pg_hba found, reconfiguration not required
Extracting system views...
.......
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
```

f. For now we will skip configuring Ambari Agents for Non-Root

2. Ambari Encrypt Database and LDAP Passwords

a. Needed to allow Ambari to cache the admin password. Run below on Ambari-server node:

b. To encrypt password, run below:

```
sudo ambari-server stop
sudo ambari-server setup-security
```

c. Then enter the below at the prompts:

   i. Enter choice: `2`

   ii. Provide master key: `BadPass#1`

   iii. Re-enter master key: `BadPass#1`

   iv. Do you want to persist? `y`

d. Then start Ambari:

```
sudo ambari-server start
```

Sample output

```
sudo ambari-server setup-security
```

```
Using python  /usr/bin/python2
Security setup options...
===================================================================
==========
Choose one of the following options:
  [1] Enable HTTPS for Ambari server.
  [2] Encrypt passwords stored in ambari.properties file.
  [3] Setup Ambari kerberos JAAS configuration.
  [4] Setup truststore.
  [5] Import certificate to truststore.
===================================================================
==========
Enter choice, (1-5): 2
Please provide master key for locking the credential store:
Re-enter master key:
Do you want to persist master key. If you choose not to persist,
you need to provide the Master Key while starting the ambari
server as an env variable named AMBARI_SECURITY_MASTER_KEY or the
start will prompt for the master key. Persist [y/n] (y)? y
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup-security' completed successfully.
```

SSL For Ambari server

Enables Ambari WebUI to run on HTTPS instead of HTTP

3. Create self-signed certificate

Note: For this lab we will be generating a self-signed certificate. In production environments you would want to use a signed certificate (either from a public authority or your own CA).

a. Generate the certificate & key by running the below command

```
 openssl req -x509 -newkey rsa:4096 -keyout ambari.key -out
ambari.crt -days 1000 -nodes -subj "/CN=$(curl icanhazptr.com)"
```

b. Move and secure the certificate and key

```
 chown ambari-server ambari.crt ambari.key
 chmod 0400 ambari.crt ambari.key
 mv ambari.crt /etc/pki/tls/certs/
 mv ambari.key /etc/pki/tls/private/
```

4. Setup SSL for Ambari server

a. Stop Ambari server

```
 sudo ambari-server stop
```

b. Setup HTTPS for Ambari

```
 sudo ambari-server setup-security
```

```
Using python  /usr/bin/python2
Security setup options...
===================================================================
==========
Choose one of the following options:
  [1] Enable HTTPS for Ambari server.
  [2] Encrypt passwords stored in ambari.properties file.
  [3] Setup Ambari kerberos JAAS configuration.
  [4] Setup truststore.
  [5] Import certificate to truststore.
===================================================================
==========
Enter choice, (1-5): 1
Do you want to configure HTTPS [y/n] (y)? y
SSL port [8443] ? 8443
Enter path to Certificate: /etc/security/ssl/ambari.crt
Enter path to Private Key: /etc/security/ssl/ambari.key
Please enter password for Private Key: BadPass#1
Importing and saving Certificate...done.
Adjusting ambari-server permissions and ownership...
```

c. Start Ambari

```
sudo ambari-server start
```

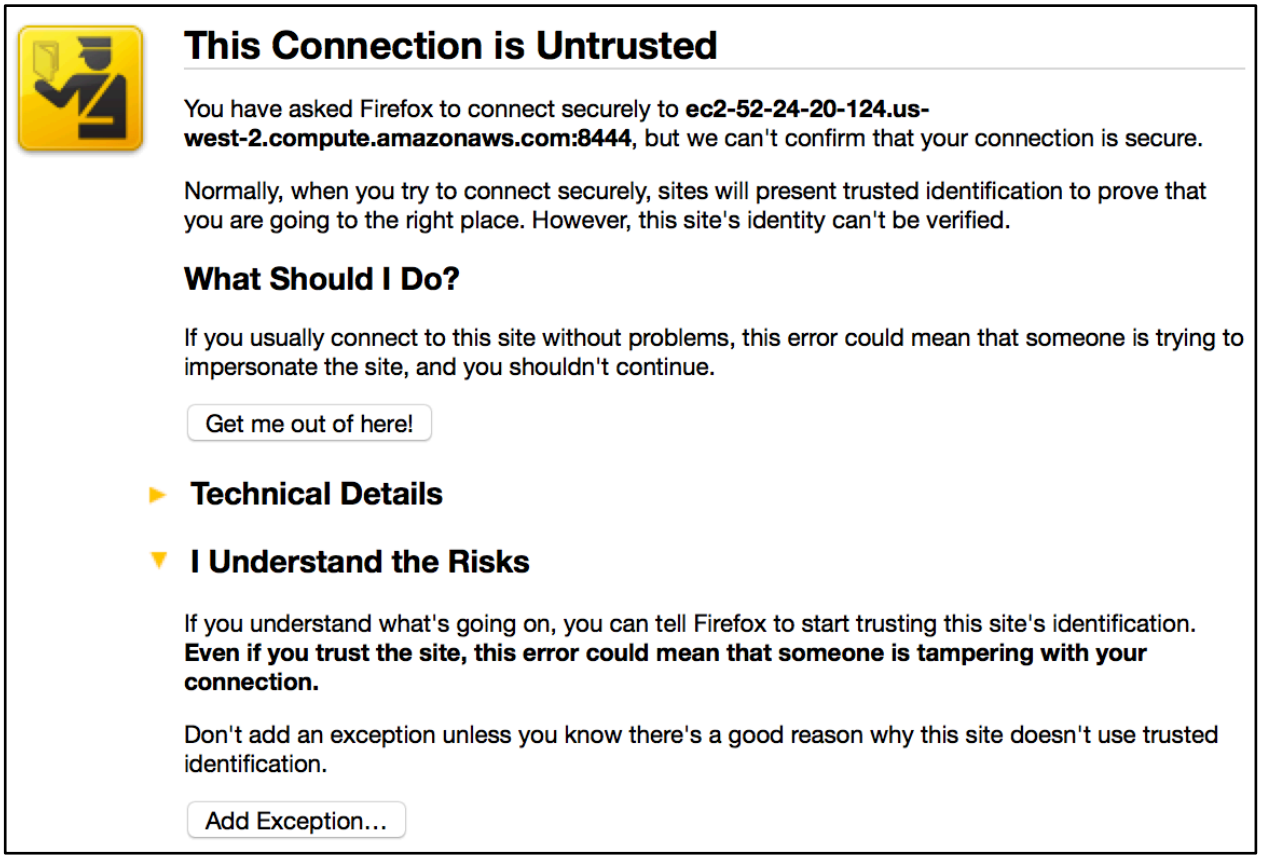
Now you can access Ambari on HTTPS on port `8443`, e.g .

https://ec2-52-32-113-77.us-west-2.compute.amazonaws.com:8443

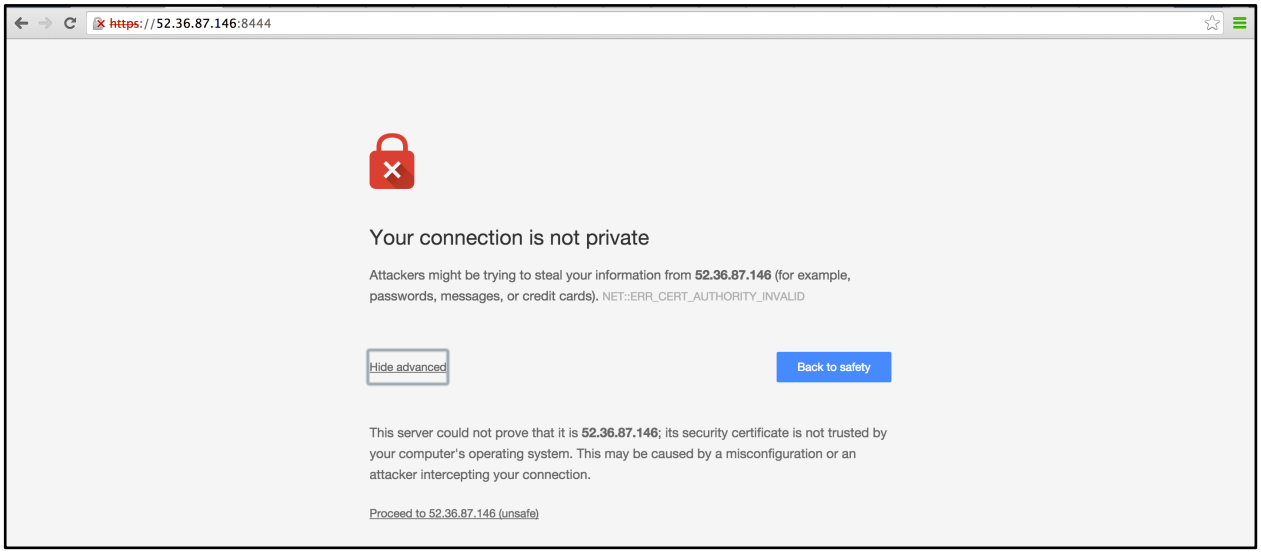
Note: Add your ambari server name above to access ambari on https port

Note: The browser will not trust the new self signed ambari certificate. You will need to trust that cert first.

If using Firefox, you can do this by clicking on `'I understand the risk' > 'Add Exception...'`



If using Chrome, you can do this by clicking on `'Advanced' > 'Proceed to xxxxxx'`



5. Setup Ambari/AD sync

Note: Run below only on Ambari node

a. This puts our AD-specific settings into variables for use in the following command

```
ad_host="ad01.lab.hortonworks.net"
ad_root="ou=CorpUsers,dc=lab,dc=hortonworks,dc=net"
ad_user="cn=ldap-
reader,ou=ServiceUsers,dc=lab,dc=hortonworks,dc=net"
```

b. Execute the following to configure Ambari to sync with LDAP.

Note: Use the default password used from without this course.

```
ambari-server setup-ldap \
--ldap-url=${ad_host}:389 \
--ldap-secondary-url= \
--ldap-ssl=false \
--ldap-base-dn=${ad_root} \
--ldap-manager-dn=${ad_user} \
--ldap-bind-anonym=false \
--ldap-dn=distinguishedName \
--ldap-member-attr=member \
--ldap-group-attr=cn \
--ldap-group-class=group \
--ldap-user-class=user \
--ldap-user-attr=sAMAccountName \
--ldap-save-settings \
--ldap-bind-anonym=false \
--ldap-referral=
```

```
[root@ip-172-30-0-218 ~]# ambari-server setup-ldap \
>   --ldap-url=${ad_host}:389 \
>   --ldap-secondary-url= \
>   --ldap-ssl=false \
>   --ldap-base-dn=${ad_root} \
>   --ldap-manager-dn=${ad_user} \
>   --ldap-bind-anonym=false \
>   --ldap-dn=distinguishedName \
>   --ldap-member-attr=member \
>   --ldap-group-attr=cn \
>   --ldap-group-class=group \
>   --ldap-user-class=user \
>   --ldap-user-attr=sAMAccountName \
>   --ldap-save-settings \
>   --ldap-bind-anonym=false \
>   --ldap-referral=
Using python  /usr/bin/python
Setting up LDAP properties...
Primary URL* {host:port} (:389):
Secondary URL {host:port} :
Use SSL* [true/false] (false):
User object class* (user):
User name attribute* (sAMAccountName):
Group object class* (group):
Group name attribute* (cn):
Group member attribute* (member):
Distinguished name attribute* (distinguishedName):
Base DN* (ou=CorpUsers,dc=lab,dc=hortonworks,dc=net):
Referral method [follow/ignore] :
Bind anonymously* [true/false] (false):
Manager DN* (cn=ldap-reader,ou=ServiceUsers,dc=lab,dc=hortonworks,dc=net):
Enter Manager Password* :
Re-enter password:
====================
Review Settings
====================
authentication.ldap.managerDn: cn=ldap-reader,ou=ServiceUsers,dc=lab,dc=hortonworks,dc=net
authentication.ldap.managerPassword: *****
Saving...done
Ambari Server 'setup-ldap' completed successfully.
[root@ip-172-30-0-218 ~]#
```

c. Restart Ambari server

```
sudo ambari-server restart
```

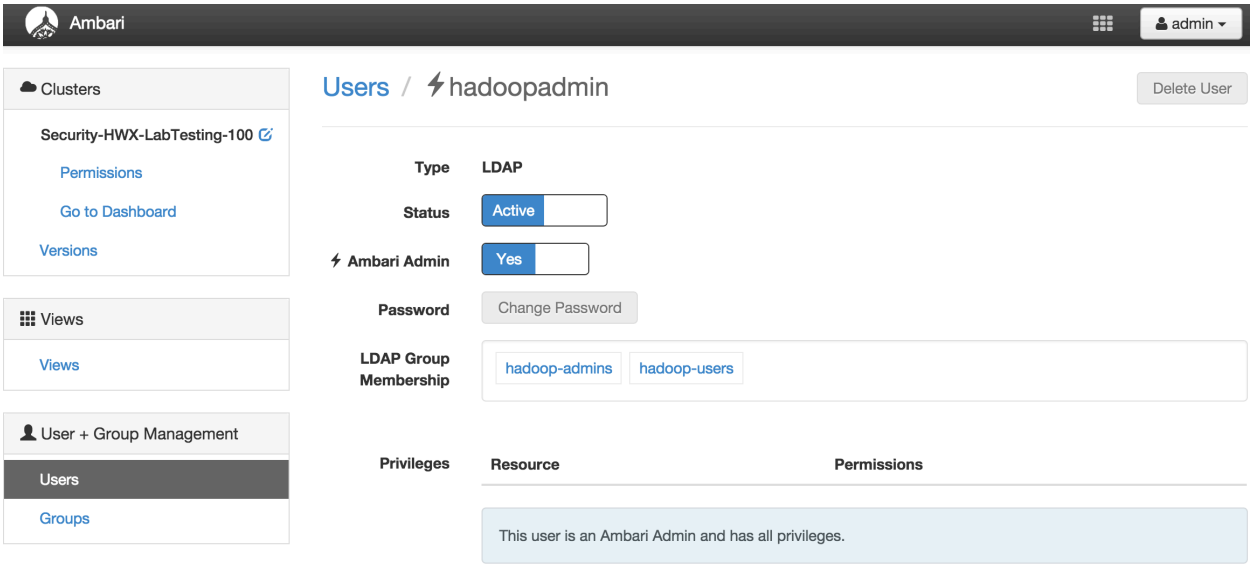
d. Run LDAP sync to sync only the groups we want

Note: When prompted for user/password, use the *local* Ambari admin credentials (i.e. admin/BadPass#1)

```
echo hadoop-users,hr,sales,legal,hadoop-admins > groups.txt
sudo ambari-server sync-ldap --groups groups.txt
```

```
[centos@ip-172-30-0-241 ~]$ sudo ambari-server sync-ldap --groups groups.txt
Using python  /usr/bin/python2
Syncing with LDAP...
Enter Ambari Admin login: admin
Enter Ambari Admin password:
Syncing specified users and groups...

Completed LDAP Sync.
Summary:
  memberships:
    removed = 0
    created = 25
  users:
    updated = 0
    removed = 0
    created = 15
  groups:
    updated = 0
    removed = 0
    created = 5

Ambari Server 'sync-ldap' completed successfully.
```

e. Give 'hadoop-admin' admin permissions in Ambari to allow the user to manage the cluster
   i. Login to Ambari as your local 'admin' user (i.e. admin/BadPass#1)
   ii. Grant 'hadoopadmin' user permissions to manage the cluster:
      1. Click the dropdown on top right of Ambari UI
      2. Click 'Manage Ambari'
      3. Under 'Users', select 'hadoopadmin'
      4. Change 'Ambari Admin' to Yes



f. Sign out and then log back into Ambari, this time as 'hadoopadmin' and verify the user has rights to monitor/manage the cluster

g. (optional) Disable local 'admin' user using the same 'Manage Ambari' menu

6. Ambari views

a. **Ambari views setup on secure cluster will be covered in another lab.**

RESULT:

Successfully sync'd Ambari and AD servers

Lab: Kerberize the Cluster

About This Lab

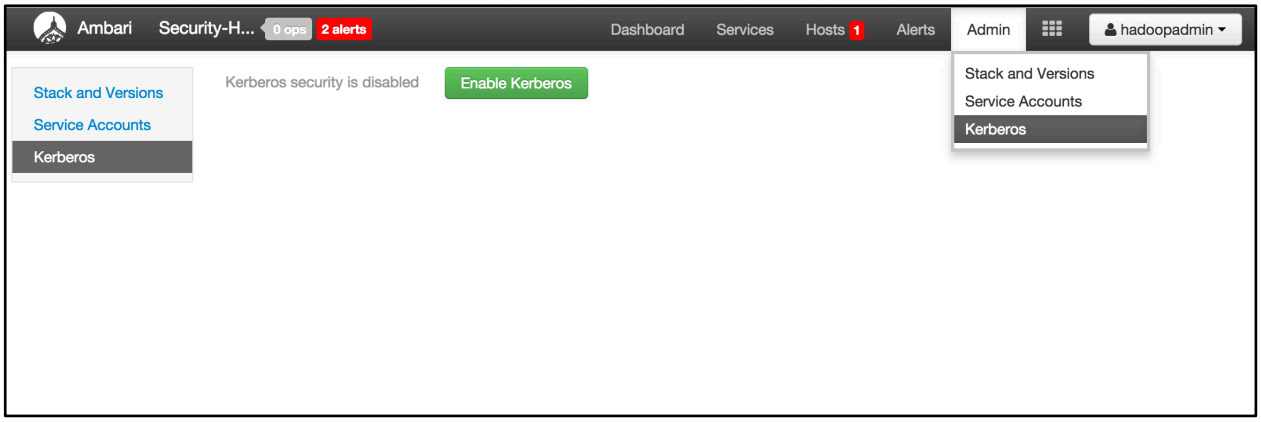
Objective:	Kerberize the Cluster
File locations:	N/A
Successful outcome:	Successfully integrated the cluster with Kerberos
Before you begin	N/A
Related lesson:	N/A

Lab Steps

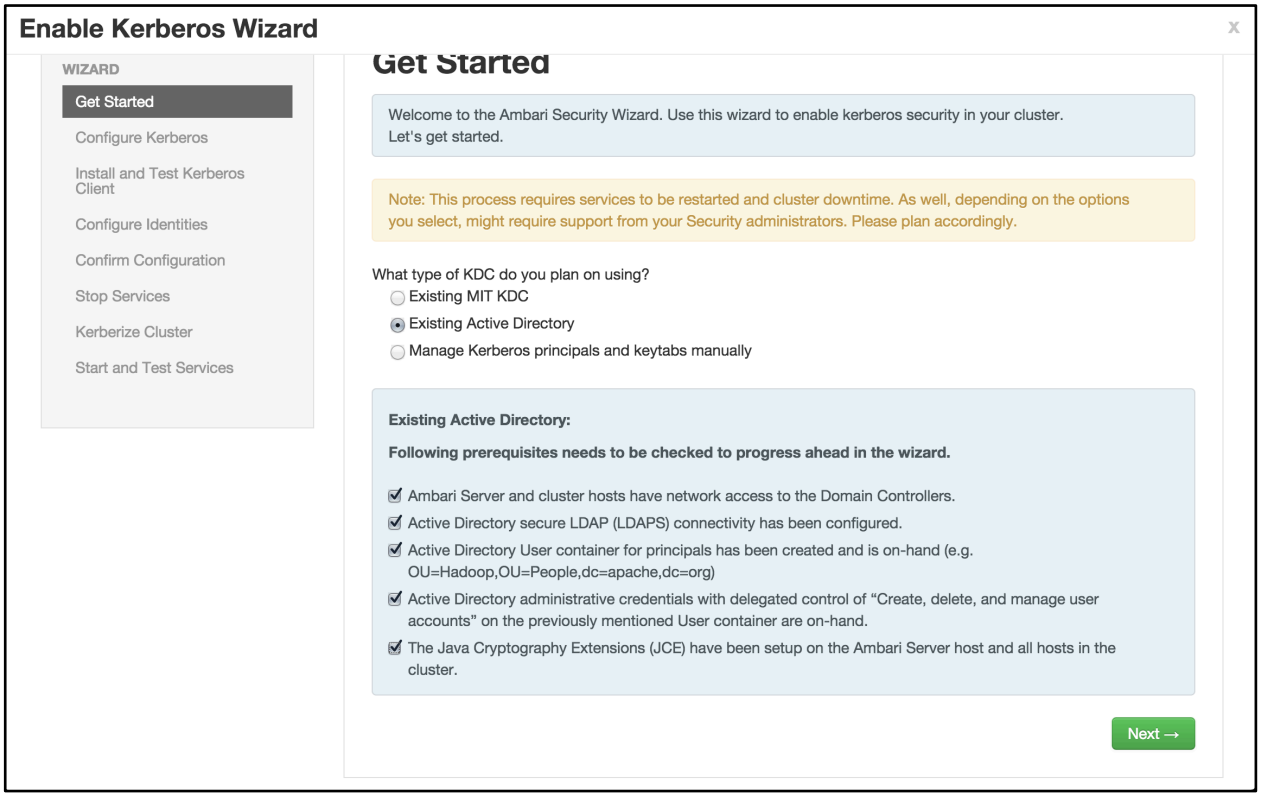
Perform the following steps:

1. Run Ambari Kerberos Wizard against Active Directory Environment

a. Enable kerberos using Ambari security wizard (under `Admin tab > Kerberos > Enable kerberos > proceed`).



b. Select "Existing Active Directory" and check all the boxes



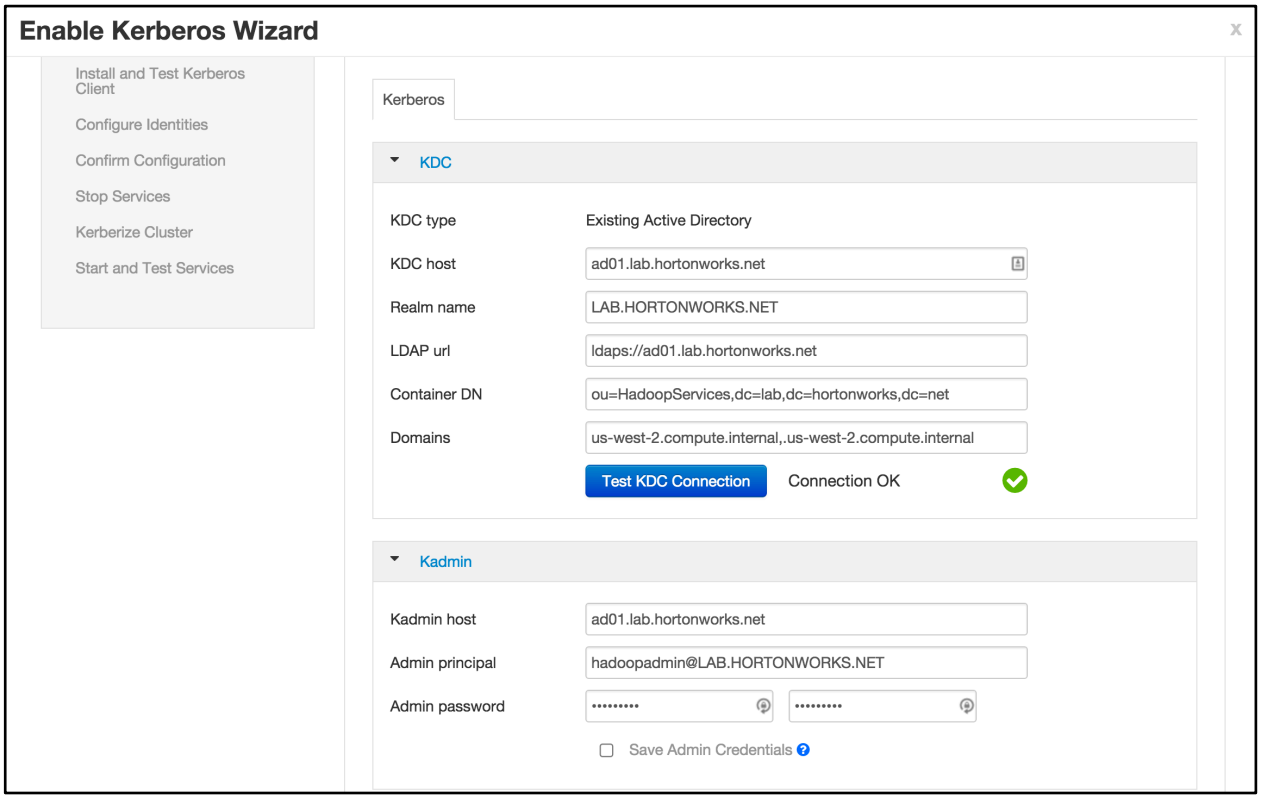
c. Enter the below details:

   i. KDC:

      1. KDC host: `ad01.lab.hortonworks.net`
      2. Realm name: `LAB.HORTONWORKS.NET`
      3. LDAP url: `ldaps://ad01.lab.hortonworks.net`
      4. Container DN: `ou=HadoopServices,dc=lab,dc=hortonworks,dc=net`
      5. Domains: `us-west-2.compute.internal,.us-west-2.compute.internal`

   ii. Kadmin:

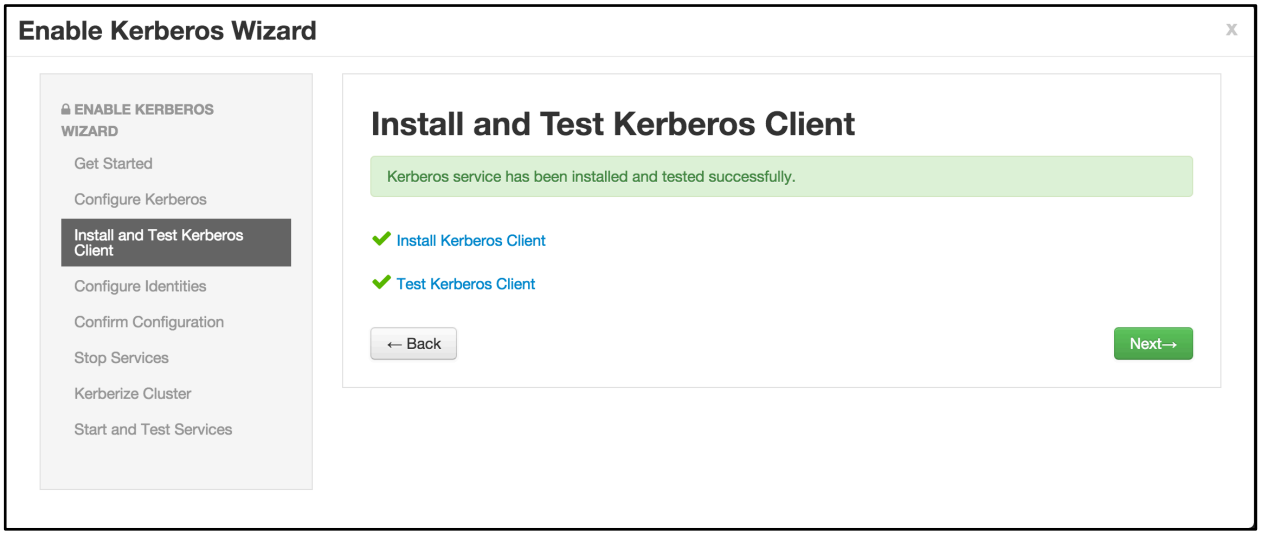
      1. Kadmin host: `ad01.lab.hortonworks.net`
      2. Admin principal: `hadoopadmin@LAB.HORTONWORKS.NET`
      3. Admin password: `BadPass#1`

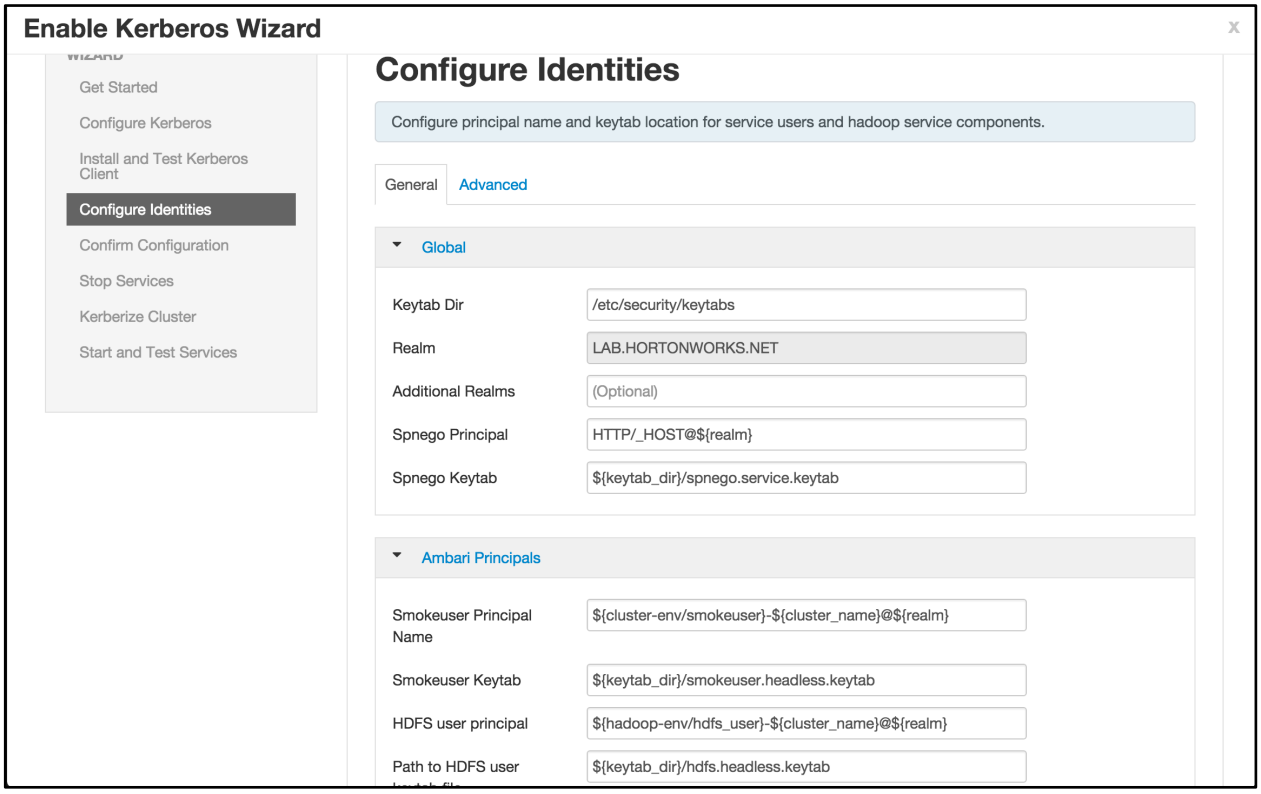


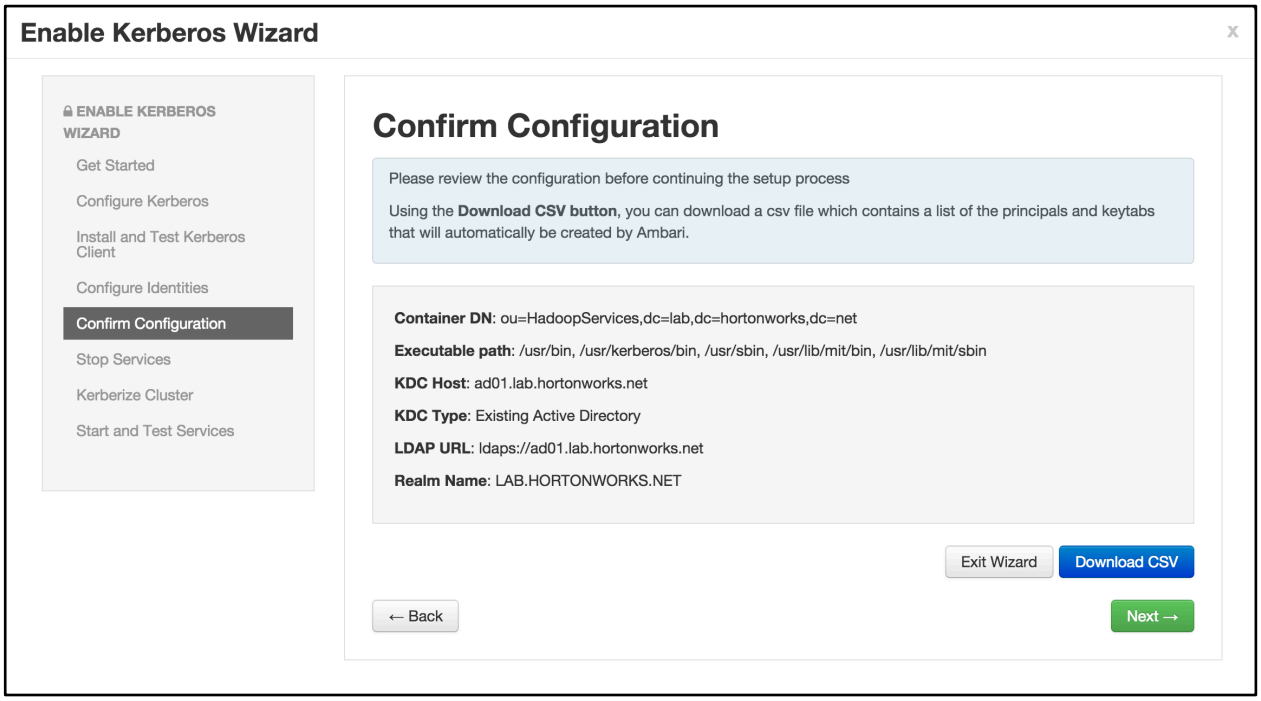
iii. Notice that the `Save admin credentials` checkbox is grayed out. We will enable that later.

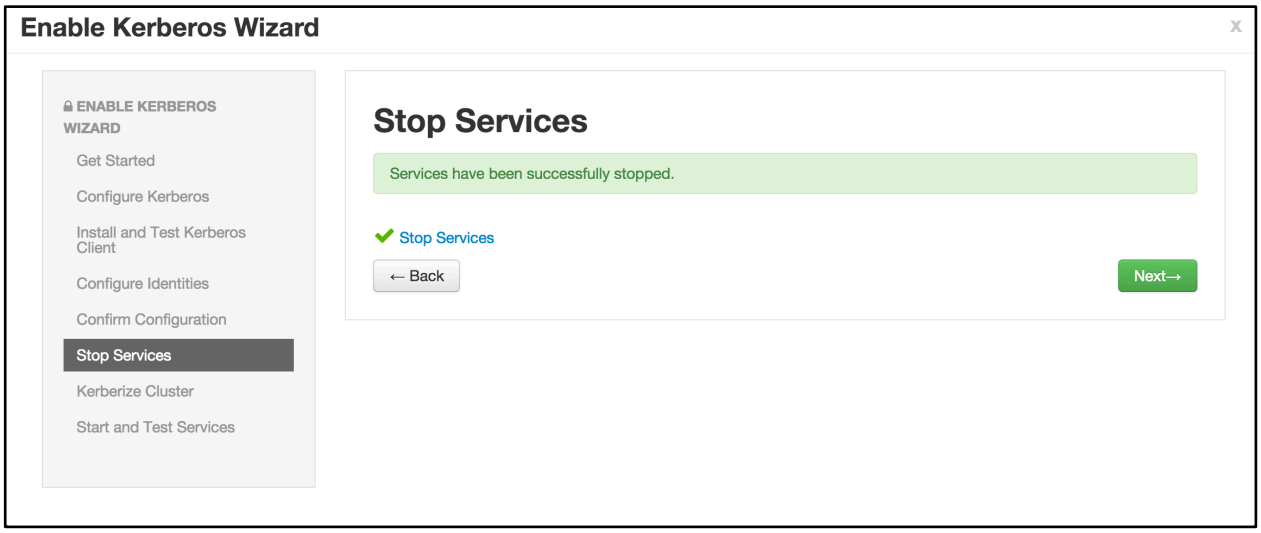
iv. Sometimes the `Test Connection` button may fail (usually related to AWS issues), but if you previously ran the "Configure name resolution & certificate to Active Directory" steps on all nodes, you can proceed.

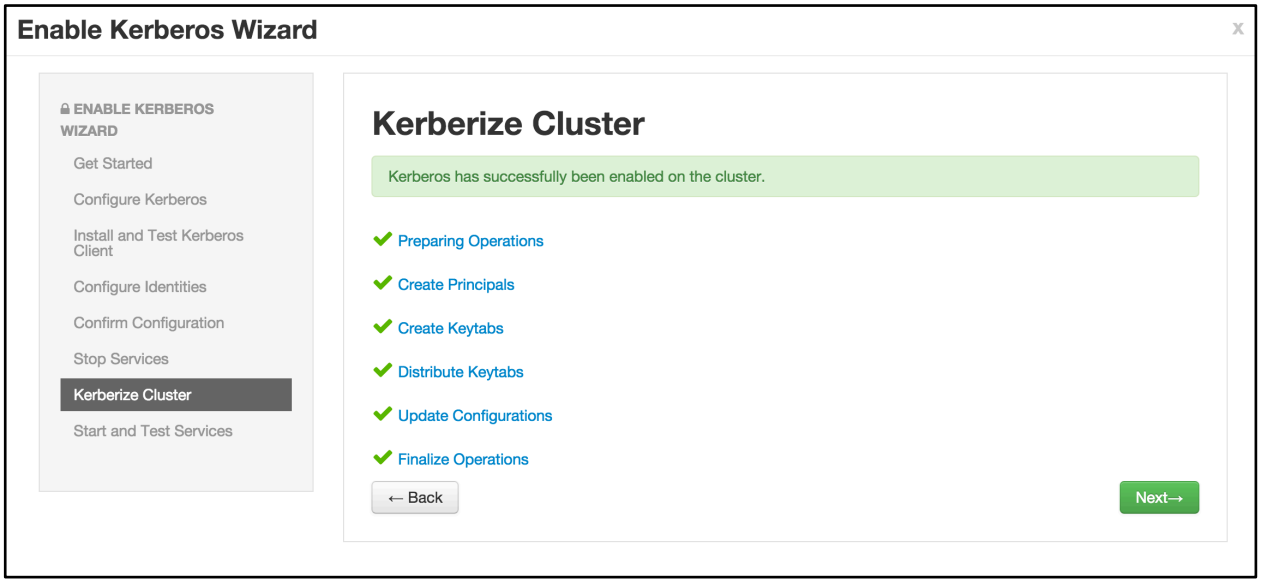
d. Now click Next on all the following screens to proceed with all the default values

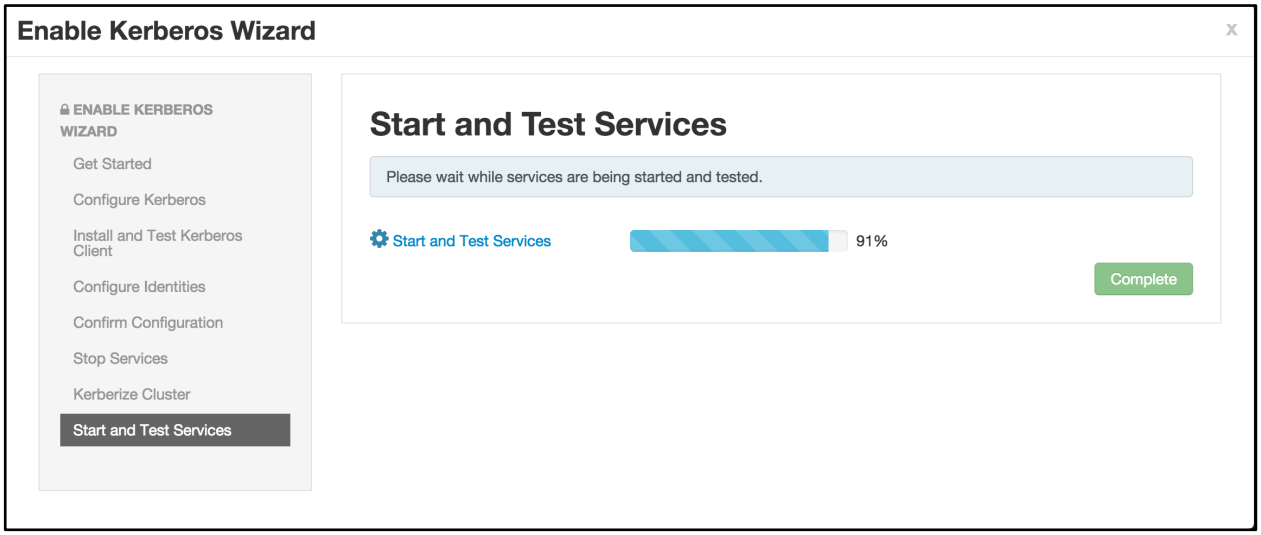












e. **NOTE:** If the wizard fails after completing more than 90% of "Start and test services" phase, you can just click `Complete` and manually start any unstarted services (e.g. WebHCat or HBase master)

f. Run the next commands on the node containing the NameNode

g. Check the `keytabs` directory and notice that `keytabs` have been generated here:

```
# ls -la /etc/security/keytabs/
```

h. Run a `klist -kt` on one of the service `keytab` files to see the principal name it is for. Sample output below (executed on host running Namenode):

```
$ sudo klist -kt /etc/security/keytabs/nn.service.keytab

Keytab name: FILE:/etc/security/keytabs/nn.service.keytab
KVNO Timestamp           Principal
---- ------------------- ------------------------------------------
--------------
```

```
   0 02/09/2016 18:04:44 nn/ip-172-30-0-181.us-west-
2.compute.internal@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 nn/ip-172-30-0-181.us-west-
2.compute.internal@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 nn/ip-172-30-0-181.us-west-
2.compute.internal@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 nn/ip-172-30-0-181.us-west-
2.compute.internal@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 nn/ip-172-30-0-181.us-west-
2.compute.internal@LAB.HORTONWORKS.NET
```

i. Notice how the service `keytabs` are divided into the below 3 parts. The instance here is the FQDN of the node so these `keytabs` are host specific.

```
{name of entity}/{instance}@{REALM}.
```

j. Run a `klist -kt` on one of the headless `keytab` files to see the principal name it is for. Sample output below (executed on host running Namenode):

```
$ sudo klist -kt /etc/security/keytabs/hdfs.headless.keytab

Keytab name: FILE:/etc/security/keytabs/hdfs.headless.keytab
KVNO Timestamp           Principal
---- ------------------- ----------------------------------------
--------------
   0 02/09/2016 18:04:44 hdfs-Security-HWX-LabTesting-
100@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 hdfs-Security-HWX-LabTesting-
100@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 hdfs-Security-HWX-LabTesting-
100@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 hdfs-Security-HWX-LabTesting-
100@LAB.HORTONWORKS.NET
   0 02/09/2016 18:04:44 hdfs-Security-HWX-LabTesting-
100@LAB.HORTONWORKS.NET
```

k. Notice how the headless `keytabs` are divided into the below 3 parts. These `keytabs` are cluster specific (i.e one per cluster)

```
{name of entity}-{cluster}@{REALM}.
```

2. Setup AD/OS integration via SSSD

Currently your hadoop nodes do not recognize users/groups defined in AD. You can check this by running below:

```
# id it1

# groups it1

# hdfs groups it1

## groups: it1: no such user
```

Before proceeding with these steps, ensure the AD admin has given `registersssd` user permissions to add the workstation to `OU=HadoopNodes` (needed to run `adcli join` successfully).

NOTE: The following is just a sample way of using SSSD. It will vary completely by environment, and needs tuning and testing for your environment.

a. Run the steps in this section on each node

```
# ad_user="registersssd"
# ad_domain="lab.hortonworks.net"
# ad_dc="ad01.lab.hortonworks.net"
# ad_root="dc=lab,dc=hortonworks,dc=net"
# ad_ou="ou=HadoopNodes,${ad_root}"
# ad_realm=${ad_domain^^}

# sudo kinit ${ad_user}

## enter BadPass#1 for password
# sudo yum makecache fast

# sudo yum -y -q install epel-release ## epel is required for
adcli

# sudo yum -y -q install sssd oddjob-mkhomedir authconfig sssd-
krb5 sssd-ad sssd-tools adcli

## paste all the lines in this block together, in one shot
# sudo adcli join -v \
  --domain-controller=${ad_dc} \
  --domain-ou="${ad_ou}" \
  --login-ccache="/tmp/krb5cc_0" \
  --login-user="${ad_user}" \
  -v \
  --show-details

## paste all the lines in this block together, in one shot
# sudo tee /etc/sssd/sssd.conf > /dev/null <<EOF
[sssd]
## master & data nodes only require nss. Edge nodes require pam.

services = nss, pam, ssh, autofs, pac
config_file_version = 2
domains = ${ad_realm}
override_space = _

[domain/${ad_realm}]
id_provider = ad
ad_server = ${ad_dc}
#ad_server = ad01, ad02, ad03
#ad_backup_server = ad-backup01, 02, 03
auth_provider = ad
chpass_provider = ad
access_provider = ad
enumerate = False
krb5_realm = ${ad_realm}
ldap_schema = ad
ldap_id_mapping = True
cache_credentials = True
ldap_access_order = expire
ldap_account_expire_policy = ad
ldap_force_upper_case_realm = true
```

```
fallback_homedir = /home/%d/%u
default_shell = /bin/false
ldap_referrals = false

[nss]
memcache_timeout = 3600
override_shell = /bin/bash
EOF
# sudo chmod 0600 /etc/sssd/sssd.conf
# sudo service sssd restart

# sudo authconfig --enablesssd --enablesssdauth --enablemkhomedir
--enablelocauthorize --update

# sudo chkconfig oddjobd on

# sudo service oddjobd restart

# sudo chkconfig sssd on

# sudo service sssd restart

# sudo kdestroy
```

b. Confirm that your nodes OS can now recognize AD users

```
id sales1
groups sales1
```

3. Refresh HDFS User-Group mappings

a. Once the above is completed on all nodes you need to refresh the user group mappings in HDFS & YARN by running the below commands

b. Restart HDFS service via Ambari. This is needed for Hadoop to recognize the group mappings (else the `hdfs groups` command will not work)

c. Execute the following on the Ambari node:

```
#export PASSWORD=BadPass#1

## detect name of cluster
output=`curl -u hadoopadmin:$PASSWORD -i -H 'X-Requested-By:
ambari'  http://localhost:8080/api/v1/clusters`

cluster=`echo $output | sed -n 's/.*"cluster_name" :
"\([^\"]*\)".*/\1/p'`

## refresh user and group mappings
# sudo sudo -u hdfs kinit -kt
/etc/security/keytabs/hdfs.headless.keytab hdfs-${cluster}

# sudo sudo -u hdfs hdfs dfsadmin -refreshUserToGroupsMappings
```

d. `Execute the following on the node where the YARN ResourceManager is installed:`

```
# sudo sudo -u yarn kinit -kt
/etc/security/keytabs/yarn.service.keytab yarn/$(hostname -
f)@LAB.HORTONWORKS.NET

# sudo sudo -u yarn yarn rmadmin -refreshUserToGroupsMappings
```

e. `kinit` as a normal Hadoop user:

```
# kinit hr1
```

f. Check the group mappings:

```
# hdfs groups

# yarn rmadmin -getGroups hr1
```

g. The output should look like below, indicating both OS-level and Hadoop-level group mappings:

```
$hdfs groups
hr1@LAB.HORTONWORKS.NET : domain_users hr hadoop-users

$yarn rmadmin -getGroups hr1
hr1 : domain_users hr hadoop-users
```

h. Remove the Kerberos ticket:

```
# kdestroy
```

4. Test OS/AD integration and Kerberos security

a. Login as `sales1` user and try to access the same `/tmp/hive HDFS dir`:

```
# sudo su - sales1

# hdfs dfs -ls /tmp/hive
## since we did not authenticate, this fails with GSSException:
No valid credentials provided

#authenticate
kinit
##enter BadPass#1

klist
## shows the principal for sales1

hdfs dfs -ls /tmp/hive
## fails with Permission denied

## Now try to get around security by setting the same env
variable

#export HADOOP_USER_NAME=hdfs

# hdfs dfs -ls /tmp/hive

##  log out as sales1
# logout
```

b. Notice that now that the cluster is Kerberized, we were not able to circumvent security by setting the `env var`

5. Kerberos for Ambari Views

a. For Ambari Views to access the cluster, Ambari must be configured to use Kerberos to access the cluster. The Kerberos wizard handles this configuration for you (as of Ambari 2.4). For those configurations to take affect, execute the following on the Ambari Server:

```
# sudo ambari-server restart
```

6. Enabling SPNEGO Authentication for Hadoop

a. **Needed to secure the Hadoop components webUIs (e.g. Namenode UI, JobHistory UI, Yarn ResourceManager UI etc...)**

b. **Run steps on ambari server node**

c. **Create Secret Key Used for Signing Authentication Tokens**

```
# sudo dd if=/dev/urandom of=/etc/security/http_secret bs=1024
count=1
# sudo chown hdfs:hadoop /etc/security/http_secret
# sudo chmod 440 /etc/security/http_secret
```

d. **Place the file in Ambari resources dir so it gets pushed to all nodes**

```
# sudo cp /etc/security/http_secret /var/lib/ambari-
server/resources/host_scripts/
# sudo ambari-server restart
```

e. Wait 30 seconds for the http_secret file to get pushed to all nodes under /var/lib/ambari-agent/cache/host_scripts

f. On non-Ambari nodes, once the above file is available, run below to put it in right dir and correct its permissions

```
# sudo cp /var/lib/ambari-agent/cache/host_scripts/http_secret
/etc/security/
# sudo chown hdfs:hadoop /etc/security/http_secret
# sudo chmod 440 /etc/security/http_secret
```

g. **In Ambari > HDFS > Configs, set the below**

h. **Under Advanced core-site:**

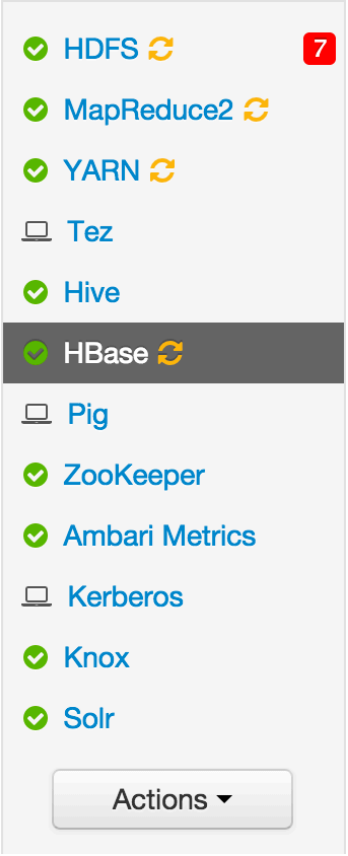
hadoop.http.authentication.simple.anonymous.allowed=false

i. Under Custom core-site, add the below properties (using bulk add tab):

```
hadoop.http.authentication.signature.secret.file=/etc/security/http_sec
ret hadoop.http.authentication.type=kerberos
hadoop.http.authentication.kerberos.keytab=/etc/security/keytabs/spnego
.service.keytab
hadoop.http.authentication.kerberos.principal=HTTP/_HOST@LAB.HORTONWORK
S.NET hadoop.http.authentication.cookie.domain=lab.hortonworks.net
hadoop.http.filter.initializers=org.apache.hadoop.security.Authenticati
onFilterInitializer
```

j. Save configs

k. Restart all services that require restart (HDFS, Mapreduce, YARN, HBase). You can use the 'Actions' > 'Restart All Required' button to restart all the services in one shot



l. Now when you try to open any of the web UIs like below you will get 401: Authentication required
   i. HDFS: Namenode UI
   ii. Mapreduce: Job history UI
   iii. YARN: Resource Manager UI

RESULT:

Have successfully Kerberized the cluster

Lab: Ranger Install

About This Lab

Objective:	In this lab we will install Apache Ranger via Ambari and setup Ranger plugins for Hadoop components: HDFS, Hive, Hbase, YARN, Knox. We will also enable Ranger audits to Solr and HDFS.
File locations:	N/A
Successful outcome:	Apache Ranger and the plugins will be successfully installed.
Before you begin	N/A
Related lesson:	N/A

Lab Steps:

Ranger Prerequisites

1. **Create and confirm MySQL user `'root'`**

   a. Prepare MySQL DB for Ranger use.

   b. Run these steps on the node where MySQL/Hive is located. To find this, you can either:

      iv. Use Ambari UI, or

      v. Just run `mysql` on each node: if it returns `mysql: command not found`, move onto next node

```
sudo mysql
```

   c. Execute following in the MySQL shell. Change the password to your preference.

```
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
```

   d. Confirm MySQL user: `mysql -u root -h $(hostname -f) -p -e "select count(user) from mysql.user;"`

   e. Output should be a simple count. Check the last step if there are errors.

2. Prepare Ambari for MySQL

   a. Run this on Ambari node

   b. Add MySQL JAR to Ambari:

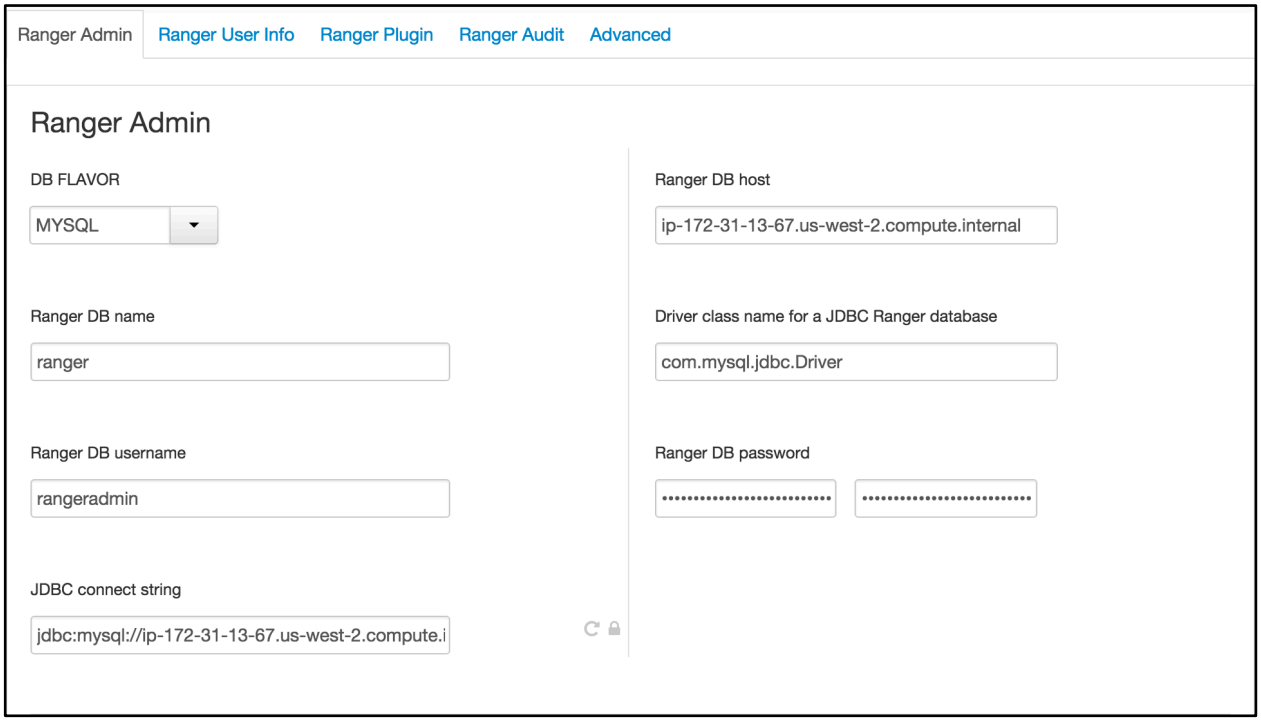
```
sudo ambari-server setup --jdbc-db=mysql --jdbc-driver=/usr/share/java/mysql-connector-java.jar
```

   c. If the file is not present, it is available on RHEL/CentOS with:

```
sudo yum -y install mysql-connector-java
```

Ranger Install

1. **Install Ranger**
    a. Start the Ambari `Add Service` wizard and select `Ranger`
    b. When prompted for where to install it, choose any node you like
    c. On the `Ranger Requirements` popup windows, you can check the box and continue as we have already completed the pre-requisite steps
    d. On the `Customize Services` page of the wizard there are a number of tabs that need to be configured as below
    e. Go through each Ranger Config tab, making below changes:
    f. Ranger Admin tab:
        i. Ranger DB Host = FQDN of host where Mysql is running (e.g. `ip-172-30-0-242.us-west-2.compute.internal`)
        ii. Enter passwords: `BadPass#1`
        iii. Click `Test Connection` button

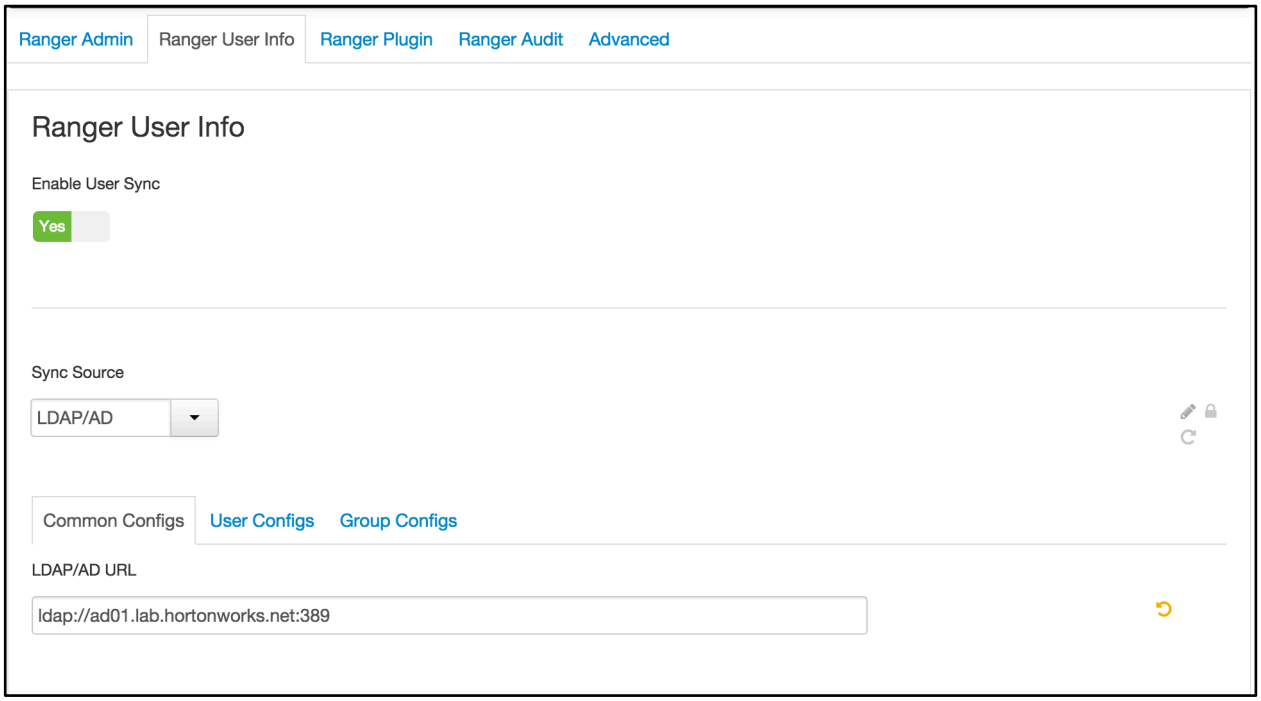


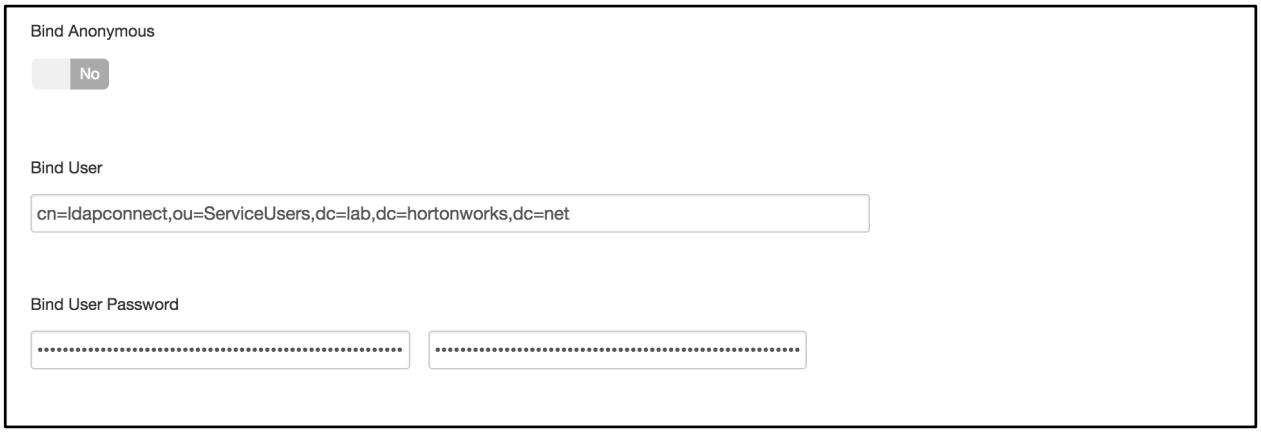
g. Ranger User Info tab

   i. Sync Source = LDAP/AD

   ii. Common Configs subtab

   iii. Enter password: `BadPass#1`



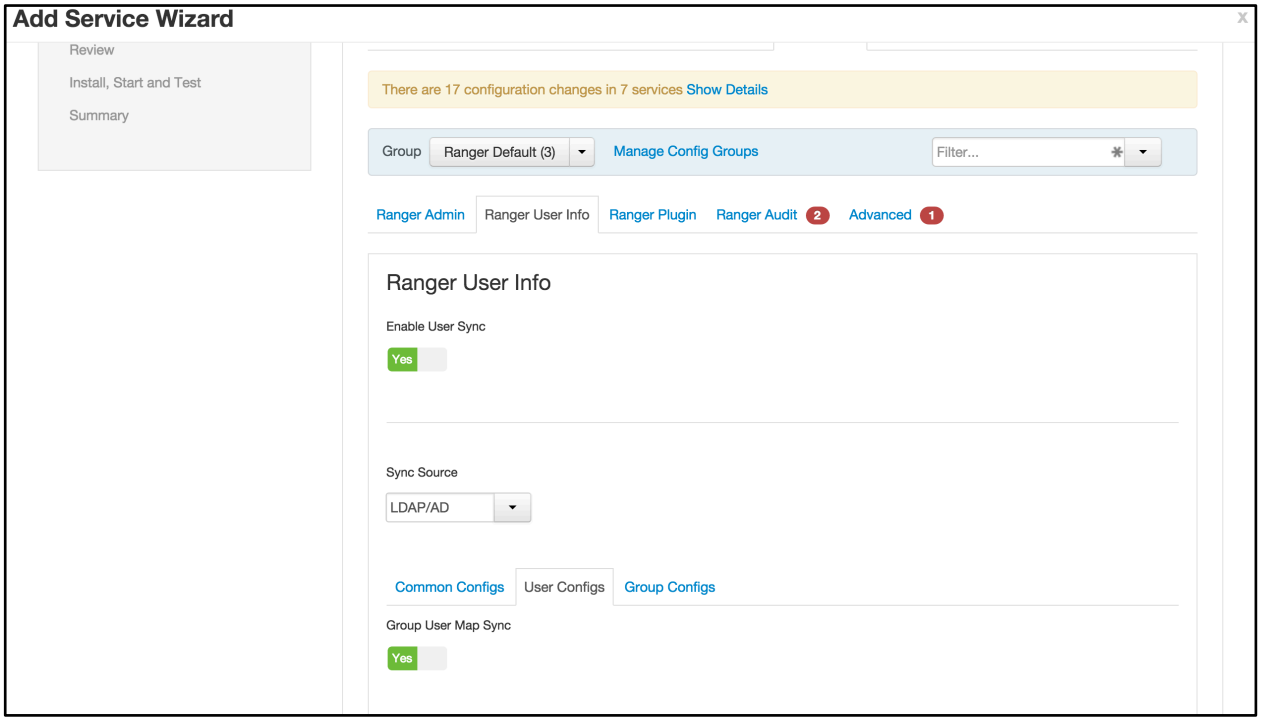


h. Ranger User Info tab

   i. User Configs subtab

   ii. User Search Base = `ou=CorpUsers,dc=lab,dc=hortonworks,dc=net`

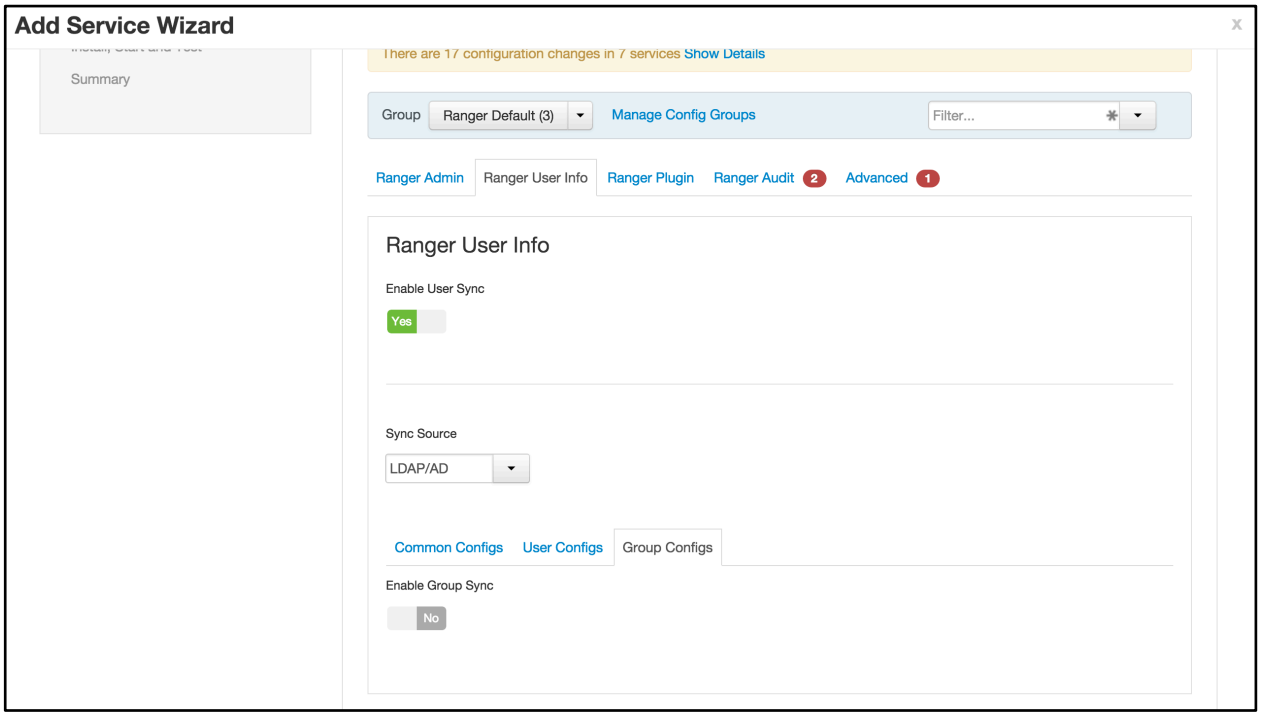
   iii. User Search Filter = `(objectcategory=person)`



i. Ranger User Info tab

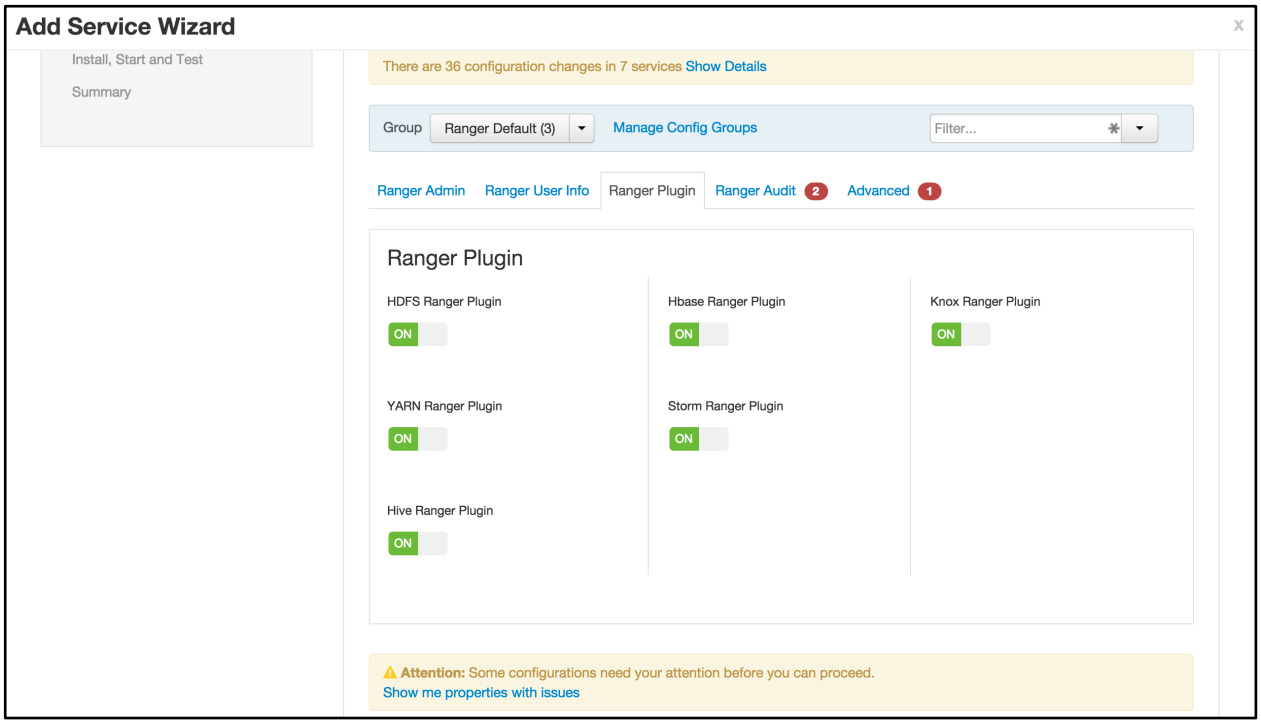
   i. Group Configs subtab

   ii. No changes needed



j. Ranger Plugins tab

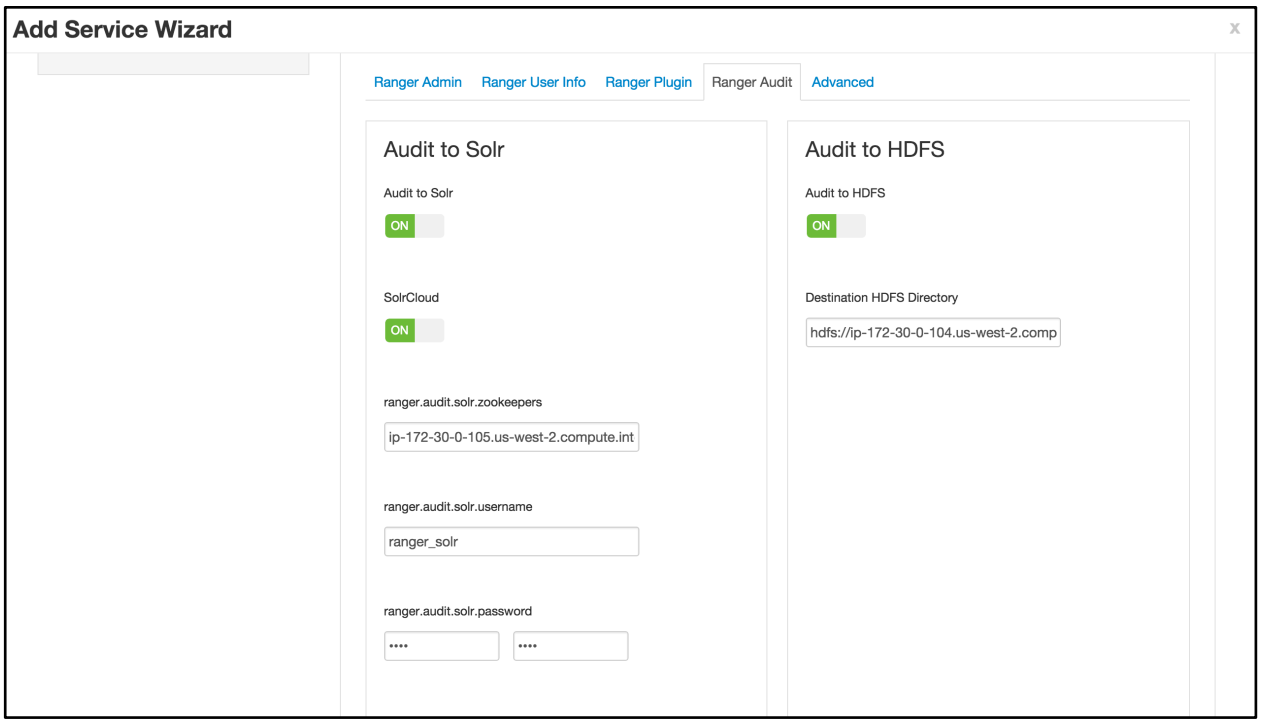
   i. Enable all plugins

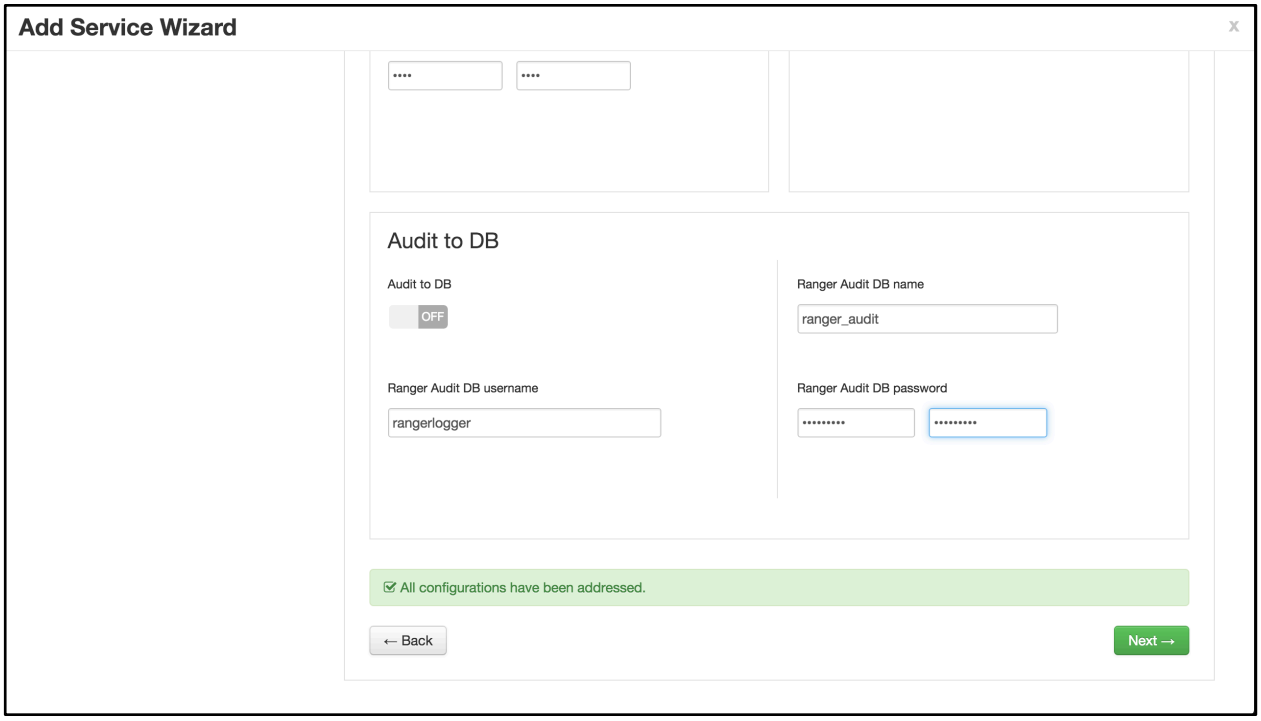


k. Ranger Audits tab

   i. SolrCloud = `ON`

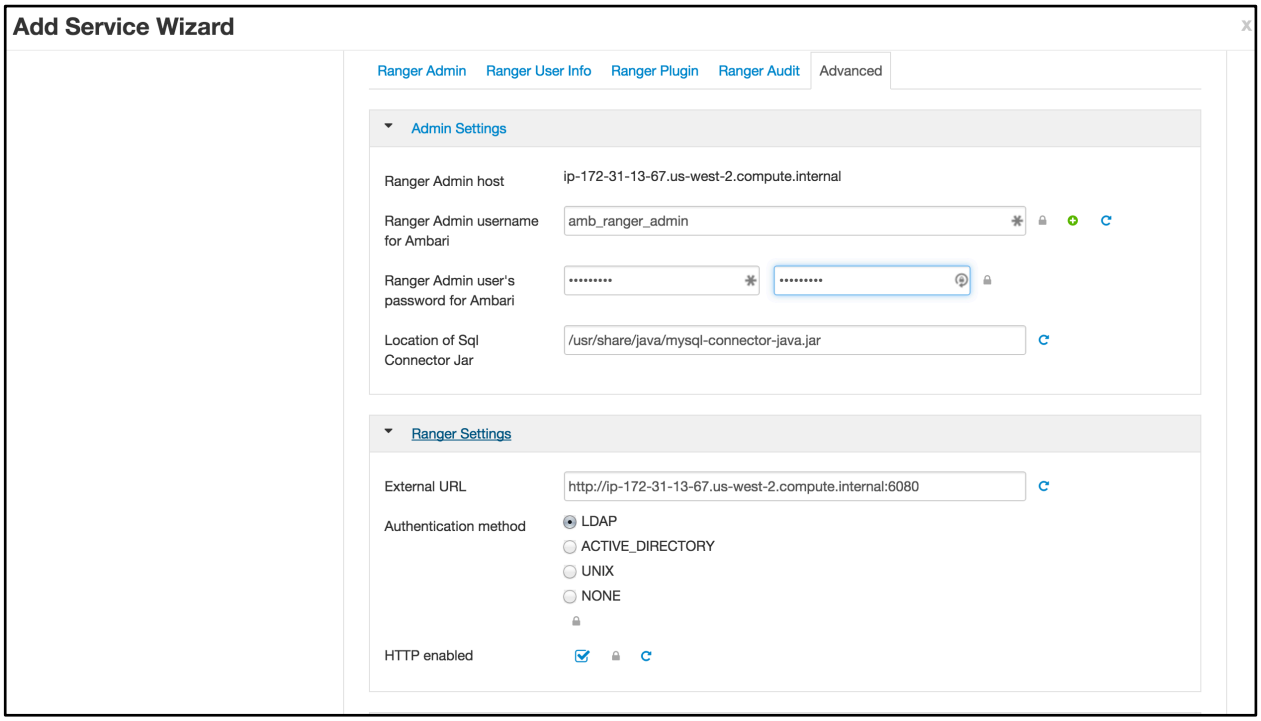
ii. **Enter password:** `BadPass#1`





l. Advanced tab

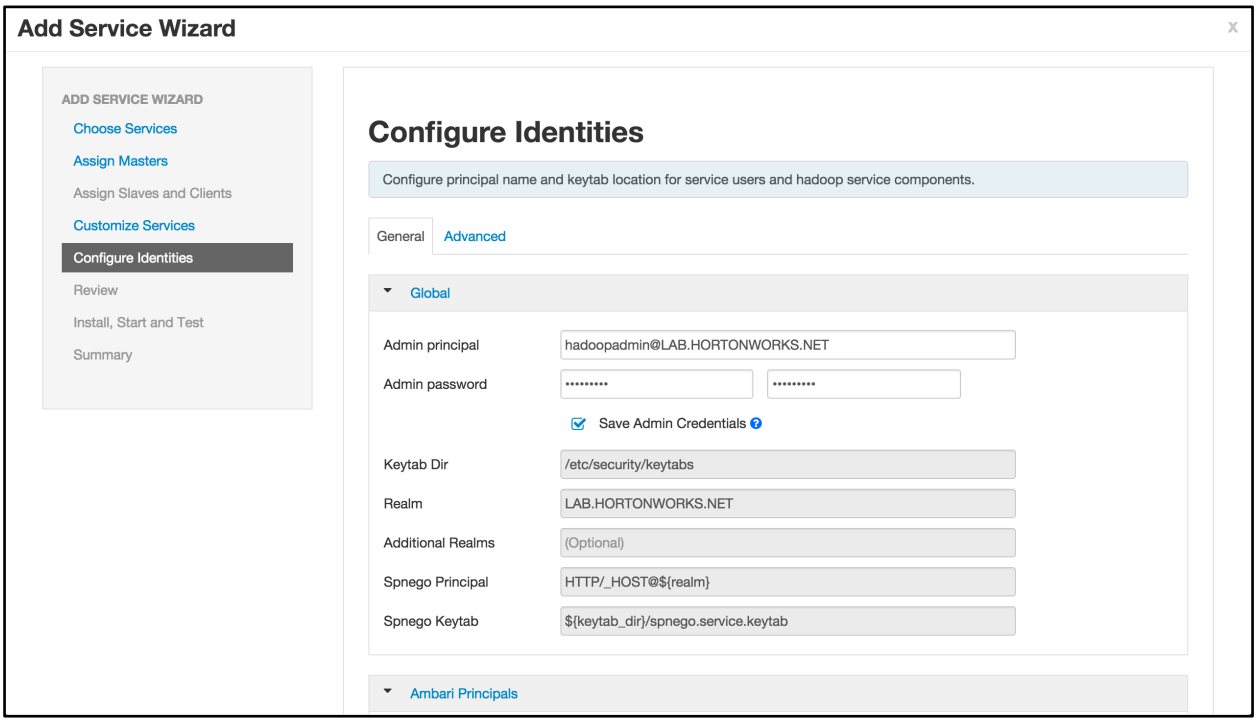
i. No changes needed (skipping configuring Ranger authentication against AD for now)



m. Click Next > Proceed Anyway to proceed

n. If prompted, on Configure Identities page, you may have to enter your AD admin credentials:

   i. Admin principal: hadoopadmin@LAB.HORTONWORKS.NET
   ii. Admin password: BadPass#1
   iii. Notice that you can now save the admin credentials. Check this box too



o. Click Next > Deploy to install Ranger

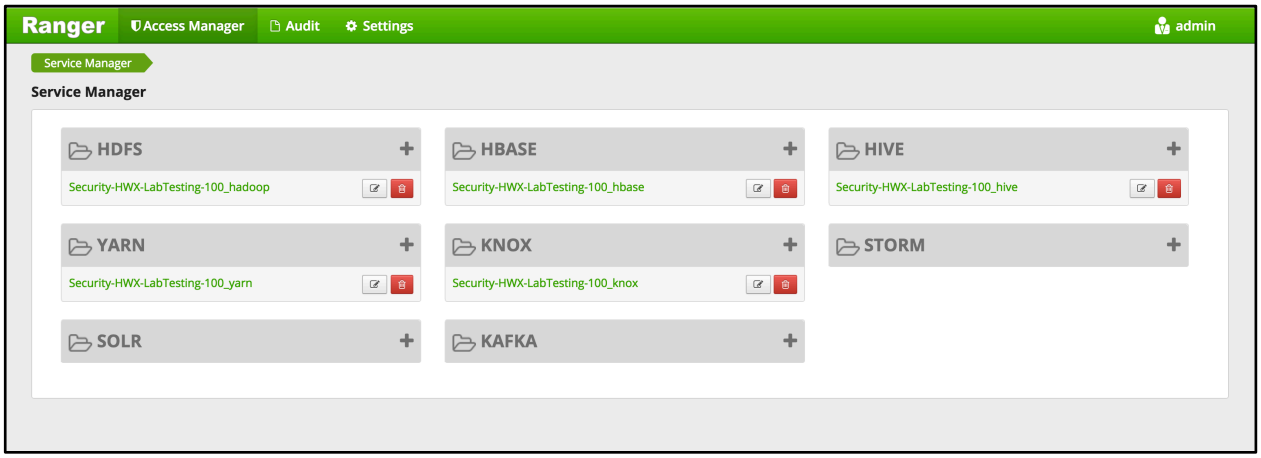
p. Once installed, restart components that require restart (e.g. HDFS, YARN, Hive etc)

- (Optional) In case of failure (usually caused by incorrectly entering the Mysql nodes FQDN in the config above), delete Ranger service from Ambari and retry.

2. Check Ranger

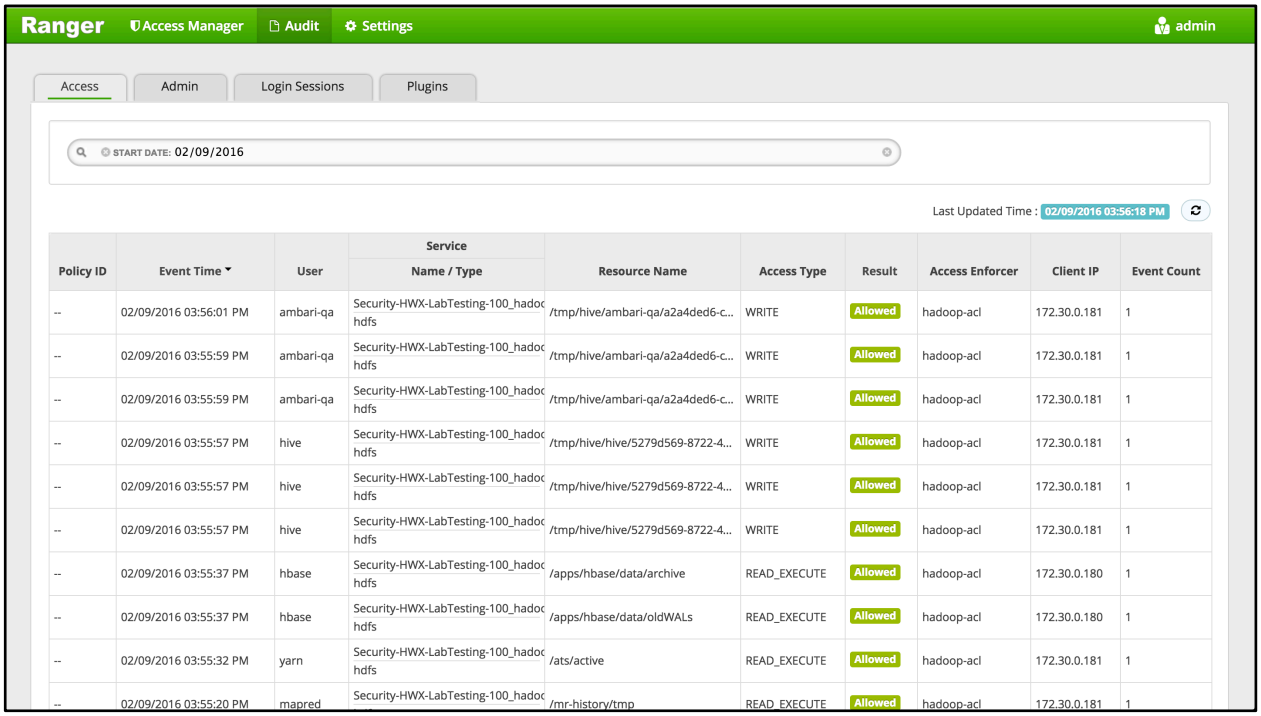
a. **Open Ranger UI at** `http://RANGERHOST_PUBLIC_IP:6080` **using** `admin/admin`

b. **Confirm that repos for HDFS, YARN, Hive, HBase, Knox appear under** `Access Manager` **tab**



Security-HWX-LabTesting-100_hadoop
Security-HWX-LabTesting-100_hbase
Security-HWX-LabTesting-100_hive
Security-HWX-LabTesting-100_yarn
Security-HWX-LabTesting-100_knox

c. Confirm that audits appear under `Audit > Access` tab



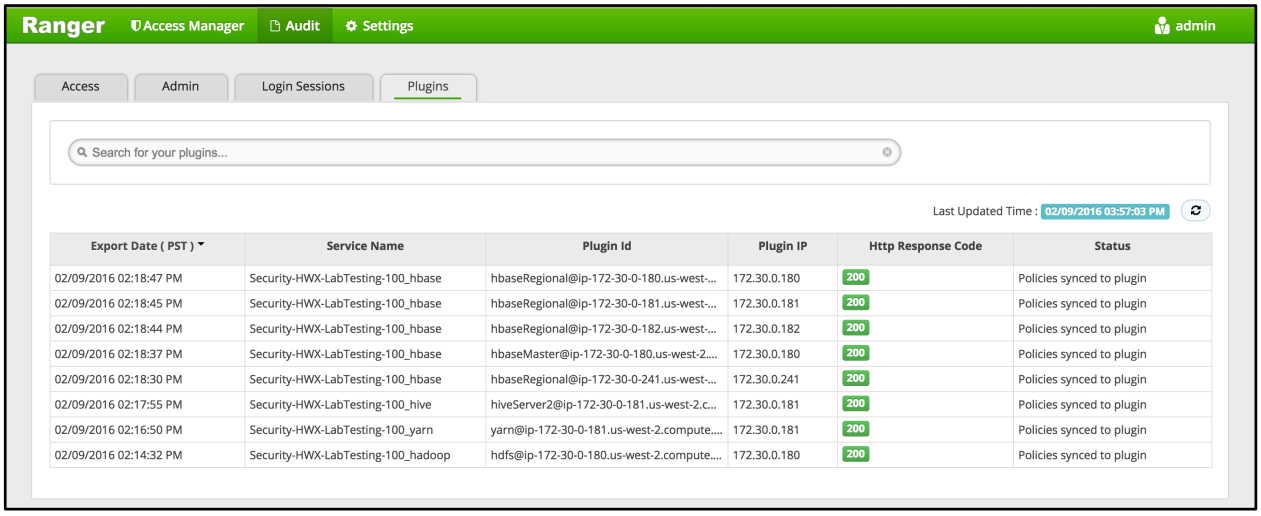
START DATE: 02/09/2016

Last Updated Time : 02/09/2016 03:56:18 PM

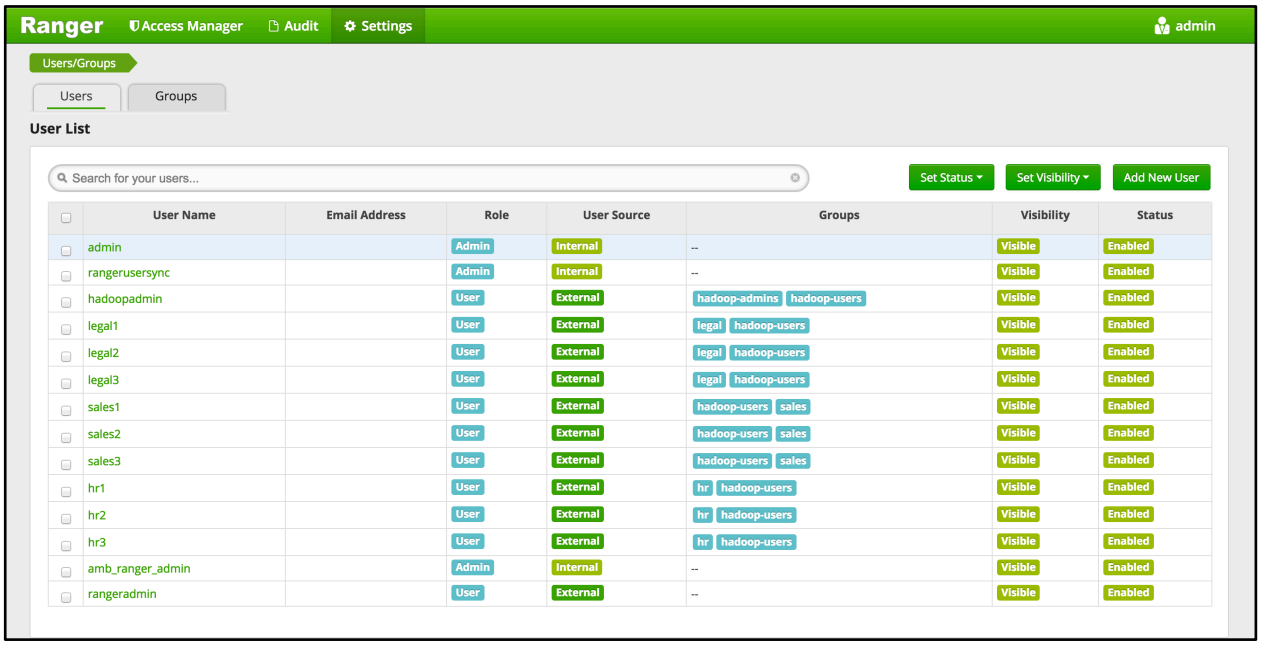
Policy ID	Event Time	User	Service	Resource Name	Access Type	Result	Access Enforcer	Client IP	Event Count
			Name / Type						
--	02/09/2016 03:56:01 PM	ambari-qa	Security-HWX-LabTesting-100_hadoc hdfs	/tmp/hive/ambari-qa/a2a4ded6-c...	WRITE	Allowed	hadoop-acl	172.30.0.181	1
--	02/09/2016 03:55:59 PM	ambari-qa	Security-HWX-LabTesting-100_hadoc hdfs	/tmp/hive/ambari-qa/a2a4ded6-c...	WRITE	Allowed	hadoop-acl	172.30.0.181	1
--	02/09/2016 03:55:59 PM	ambari-qa	Security-HWX-LabTesting-100_hadoc hdfs	/tmp/hive/ambari-qa/a2a4ded6-c...	WRITE	Allowed	hadoop-acl	172.30.0.181	1
--	02/09/2016 03:55:57 PM	hive	Security-HWX-LabTesting-100_hadoc hdfs	/tmp/hive/hive/5279d569-8722-4...	WRITE	Allowed	hadoop-acl	172.30.0.181	1
--	02/09/2016 03:55:57 PM	hive	Security-HWX-LabTesting-100_hadoc hdfs	/tmp/hive/hive/5279d569-8722-4...	WRITE	Allowed	hadoop-acl	172.30.0.181	1
--	02/09/2016 03:55:57 PM	hive	Security-HWX-LabTesting-100_hadoc hdfs	/tmp/hive/hive/5279d569-8722-4...	WRITE	Allowed	hadoop-acl	172.30.0.181	1
--	02/09/2016 03:55:37 PM	hbase	Security-HWX-LabTesting-100_hadoc hdfs	/apps/hbase/data/archive	READ_EXECUTE	Allowed	hadoop-acl	172.30.0.180	1
--	02/09/2016 03:55:37 PM	hbase	Security-HWX-LabTesting-100_hadoc hdfs	/apps/hbase/data/oldWALs	READ_EXECUTE	Allowed	hadoop-acl	172.30.0.180	1
--	02/09/2016 03:55:32 PM	yarn	Security-HWX-LabTesting-100_hadoc hdfs	/ats/active	READ_EXECUTE	Allowed	hadoop-acl	172.30.0.181	1
--	02/09/2016 03:55:20 PM	mapred	Security-HWX-LabTesting-100_hadoc	/mr-history/tmp	READ_EXECUTE	Allowed	hadoop-acl	172.30.0.181	1

NOTE: If audits do not show up here, you may need to restart Solr from Ambari

d. Confirm that plugins for HDFS, YARN, Hive, etc. appear under `Audit > Plugins` tab



e. Confirm users and group synchronization from Active Directory into Ranger is working by clicking `Settings > Users/Groups` tab in Ranger UI and noticing AD users/groups are present



f. Confirm HDFS audits working by querying the `audits dir` in HDFS:

```
sudo -u hdfs hdfs dfs -cat /ranger/audit/hdfs/*/*
```

RESULT:

Have successfully Installed Apache Ranger and it's plugins

Lab: Ranger KMS/Data Encryption Setup

About This Lab

Objective:	In this lab we will install Ranger KMS via Ambari. Next we will create some encryption keys and use them to create encryption zones (EZs) and copy files into them.
File locations:	N/A
Successful outcome:	Successfully install Ranger KMS. Copy files into encryption zones successfully.
Before you begin	N/A
Related lesson:	N/A

Lab Steps

Perform the following steps:

1. **Setup Proxy Users**
   a. In this section we will have to setup proxyusers. This is done to enable impersonation whereby a superuser can submit jobs or access HDFS on behalf of another user (e.g. because superuser has Kerberos credentials, but user `joe` doesn't have any). For more details on this, refer to `https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/Superusers.html`

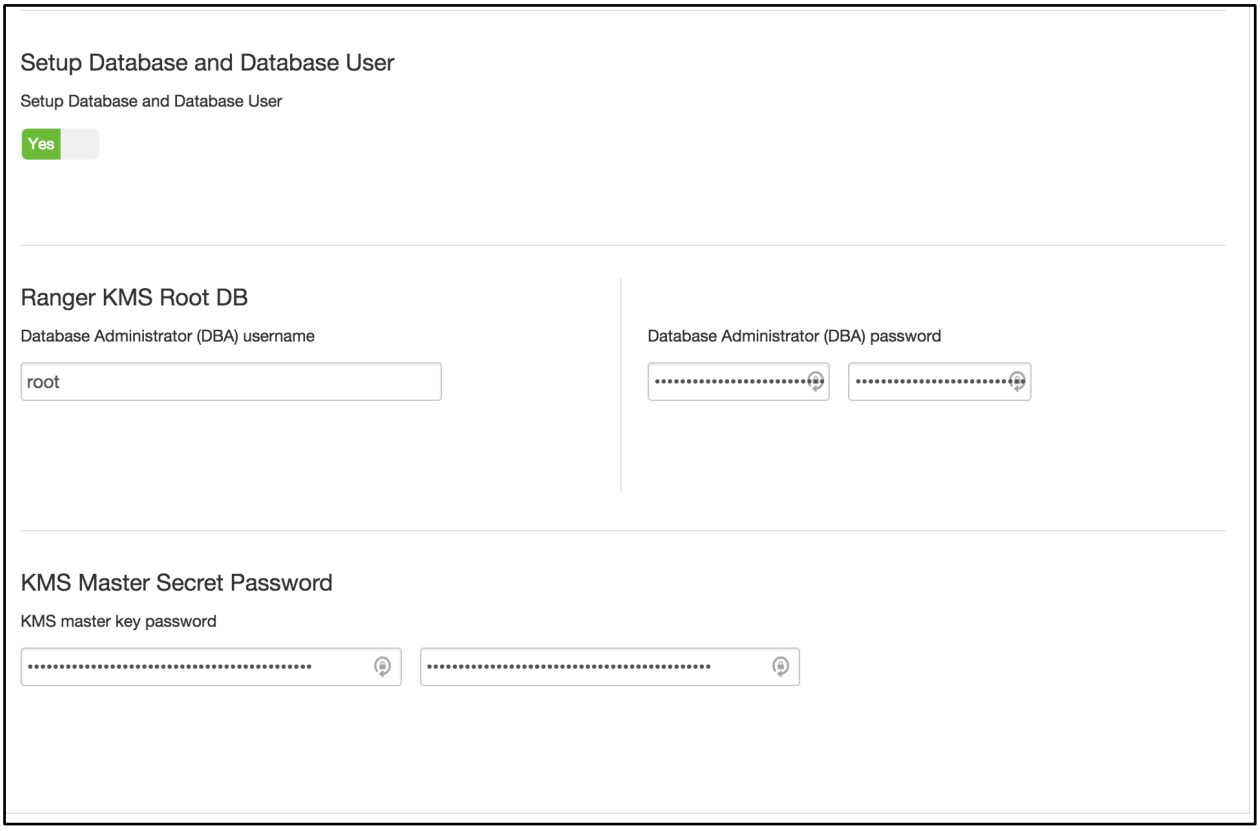
   b. Before starting KMS install, find and note down the below 3 pieces of information. These will be used during KMS install.
      i. Open `Ambari > Ranger > Config > Filter` for `ranger.audit.solr.zookeepers` and note down its value
         1. It will be something like
            `ip-172-30-0-180.us-west-2.compute.internal:2181,ip-172-30-0-182.us-west-2.compute.internal:2181,ip-172-30-0-181.us-west-2.compute.internal:2181/ranger_audits`
      ii. Find the internal hostname of host running namenode and note it down
         1. From `Ambari > HDFS >` click the `NameNode` hyperlink. The internal hostname should appear in upper left of the page.
      iii. Find the internal hostname of host running Mysql and note it down
         1. From `Ambari > Mysql >` click the `Mysql Server` hyperlink. The internal hostname should appear in upper left of the page.

   c. Open Ambari > start 'Add service' wizard > select 'Ranger KMS'.

   d. Pick any node to install on

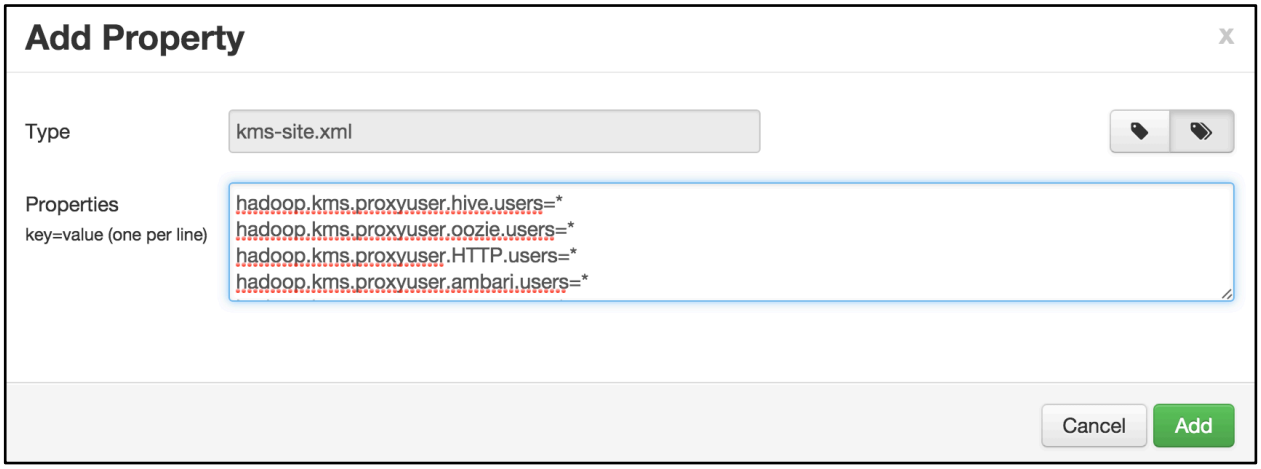
   e. Under Ambari > Ranger KMS > Settings tab :

- Ranger KMS DB host:
- Ranger KMS DB password: BadPass#1
- DBA password: BadPass#1
- KMS master secret password: BadPass#1



f. **Custom `kms-site` (to avoid adding one at a time, you can use 'bulk add' mode):**

```
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=*
hadoop.kms.proxyuser.yarn.hosts=*
hadoop.kms.proxyuser.keyadmin.groups=*
hadoop.kms.proxyuser.keyadmin.hosts=*
hadoop.kms.proxyuser.keyadmin.users=*
```



g. Click `Next > Proceed Anyway` to proceed with the wizard

h. If prompted on `Configure Identities` page, you will have to enter your AD admin credentials:

   i. Admin principal: `hadoopadmin@LAB.HORTONWORKS.NET`

   ii. Admin password: `BadPass#1`

   iii. Check the `Save admin credentials` checkbox

i. Click `Next > Deploy` to install RangerKMS

j. Restart Ranger and RangerKMS via Ambari (hold off on restarting HDFS and other components for now)

k. Confirm these properties got populated to `kms://http@(kmshostname):9292/kms`

   i. `HDFS > Configs > Advanced core-site`:

      - `hadoop.security.key.provider.path`

   ii. `HDFS > Configs > Advanced hdfs-site`:

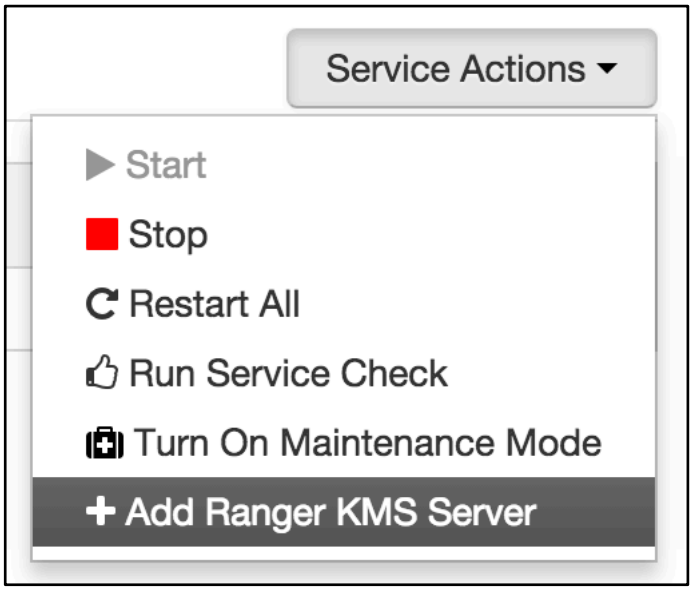
      - `dfs.encryption.key.provider.uri`

l. Restart the services that require it e.g. HDFS, MapReduce, YARN via Actions > Restart All Required

m. Restart Ranger and RangerKMS services.

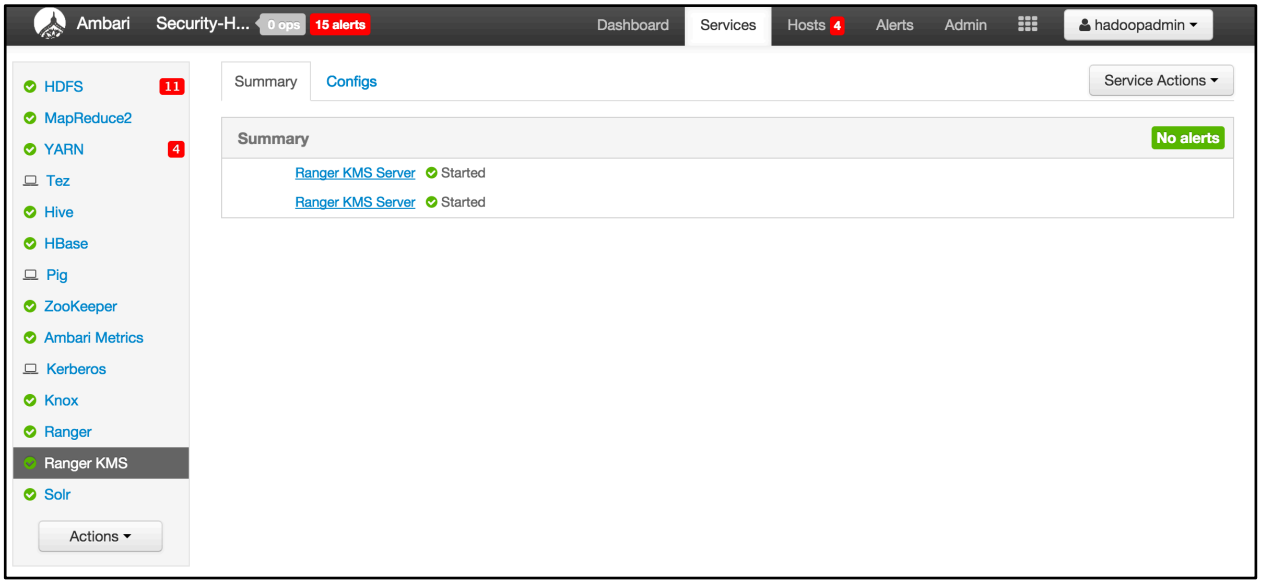
n. **OPTIONAL:** Add another KMS:

   i. `Ambari > Ranger KMS > Service Actions > Add Ranger KMS Server` > Pick any host



ii. **After it is installed, you can start it by going to:** `Ambari > Ranger KMS > Service Actions > Start`

iii. **Once started you will see multiple KMS Servers running in Ambari:**



2. Ranger KMS/Data Encryption Exercise

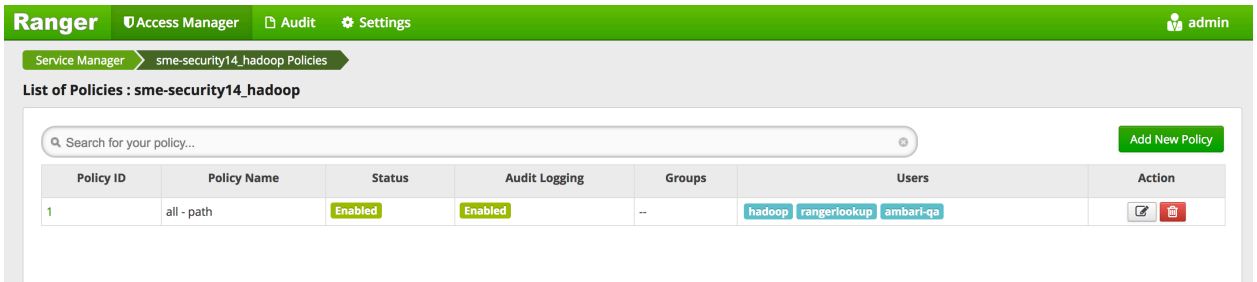
Before we can start exercising HDFS encryption, we will need to set:

- policy for hadoopadmin access to HDFS
- policy for hadoopadmin access to Hive
- policy for hadoopadmin access to the KMS keys we created

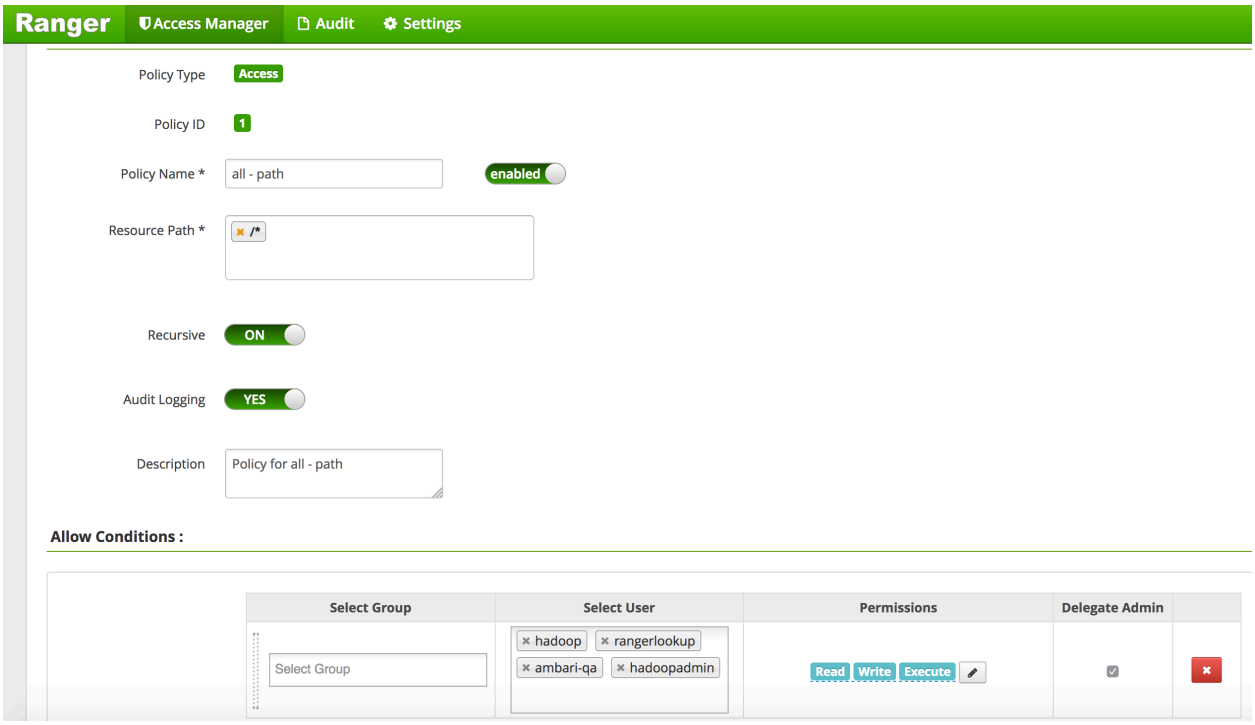
a. Add the user hadoopadmin to the Ranger HDFS global policies

i. Access Manager > HDFS > (clustername)_hdfs

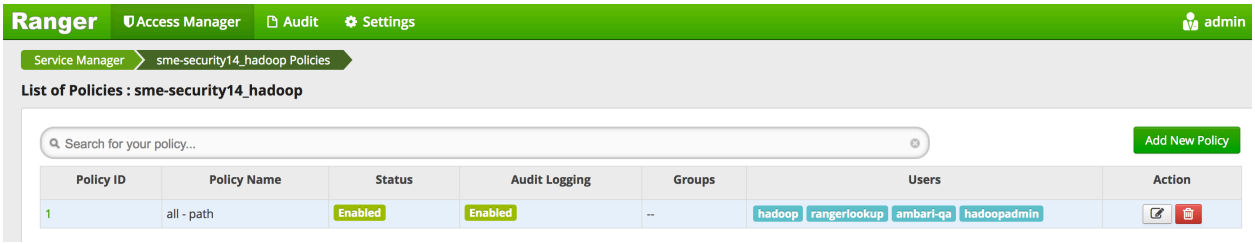
ii. This will open the list of HDFS policies



b. Edit the 'all - path' global policy (the first one) and add hadoopadmin to global HDFS policy and Save



c. Your policy now includes hadoopadmin

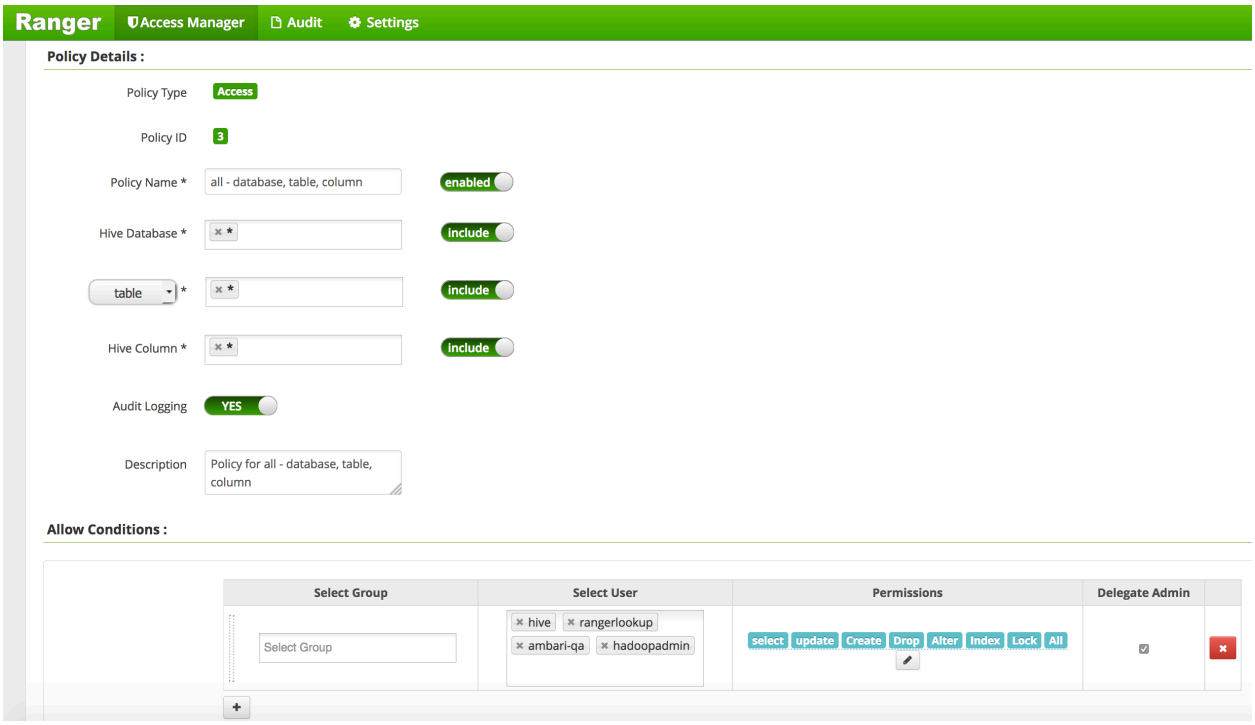


d. Add the user hadoopadmin to the Ranger Hive global policies. (Hive has two global policies: one on Hive tables, and one on Hive UDFs)

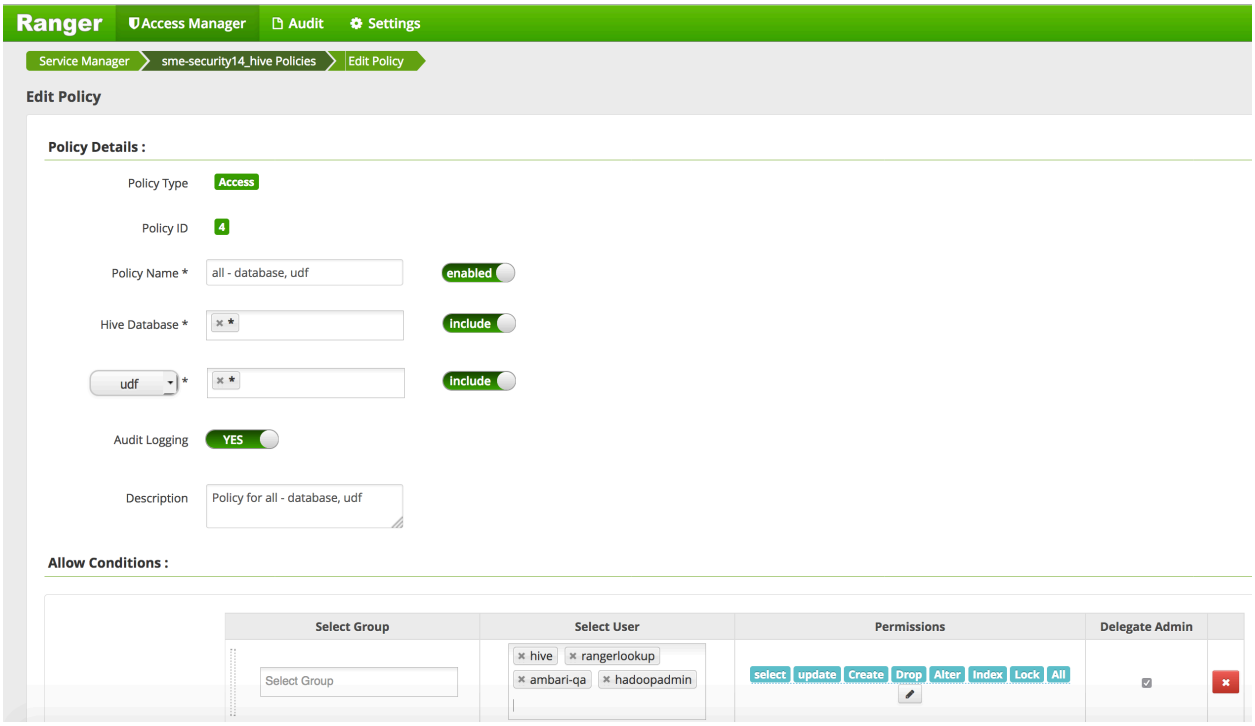
i. Access Manager > HIVE > (clustername)_hive

ii. This will open the list of HIVE policies Image

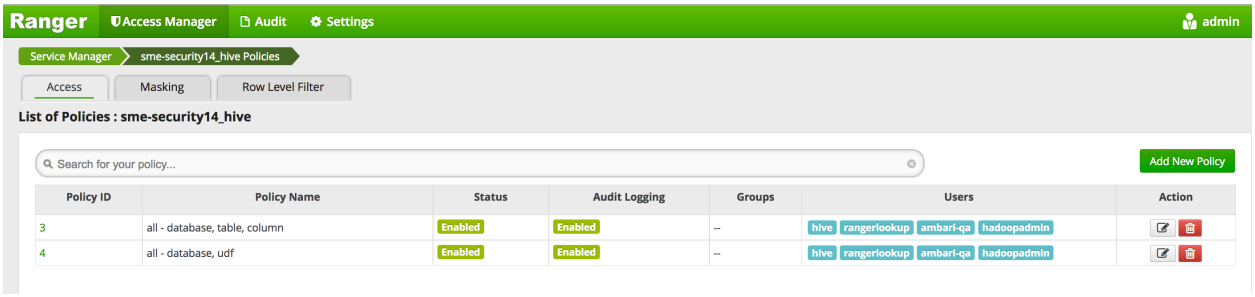
iii. Edit the 'all - database, table, column' global policy (the first one) and add hadoopadmin to global HIVE policy and Save



e. Edit the 'all - database, udf' global policy (the second one) and add hadoopadmin to global HIVE policy and Save



f. Your policies now includes hadoopadmin



g. Add policy for keyadmin to be able to access /ranger/audit/kms

i. First Create the hdfs directory for Ranger KMS Audit

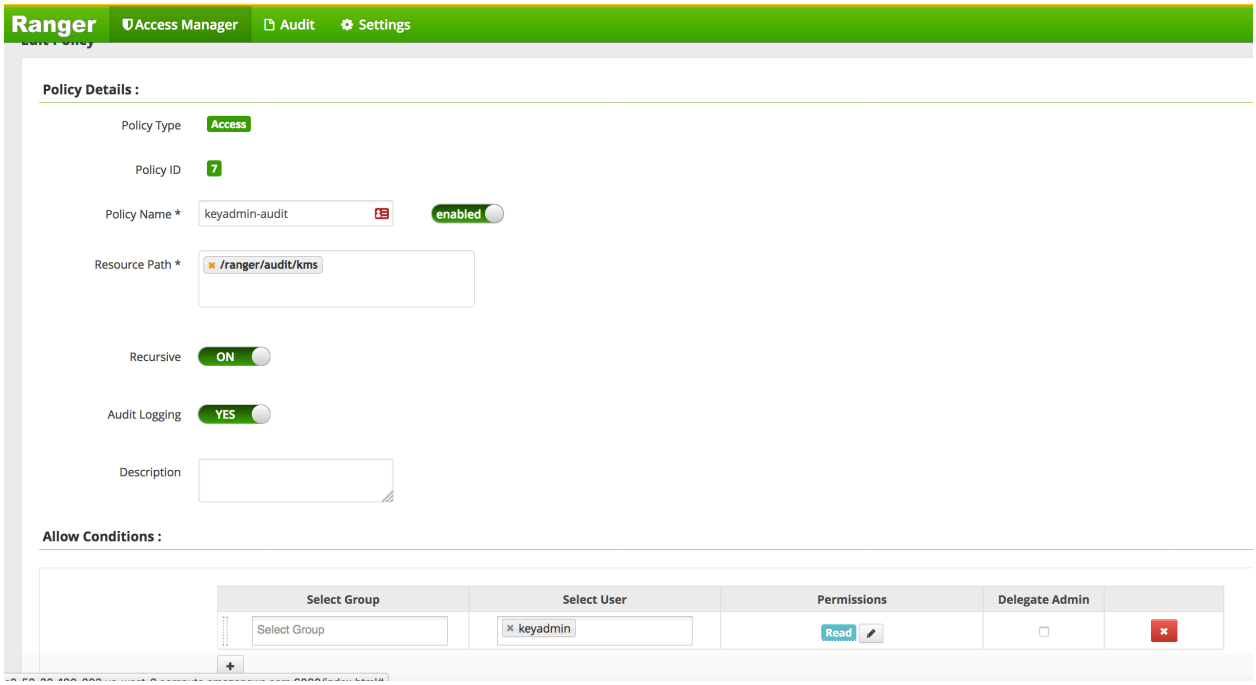
```
#run below on Ambari node

 export PASSWORD=BadPass#1

#detect name of cluster
 output=`curl -u hadoopadmin:$PASSWORD -k -i -H 'X-Requested-By: ambari'
https://localhost:8443/api/v1/clusters`
 cluster=`echo $output | sed -n 's/.*"cluster_name" : "\([^\"]*\)".*/\1/p'`

 echo $cluster
## this should show the name of your cluster

## if not you can manully set this as below
## cluster=Security-HWX-LabTesting-XXXX
```

```
#then kinit as hdfs using the headless keytab and the principal name
 sudo -u hdfs kinit -kt /etc/security/keytabs/hdfs.headless.keytab "hdfs-${cluster,,}"

#Create the Ranger KMS Audit Directory
 sudo -u hdfs hdfs dfs -mkdir -p /ranger/audit/kms
 sudo -u hdfs hdfs dfs -chown -R kms:hdfs /ranger/audit/kms
 sudo -u hdfs hdfs dfs -chmod 700 /ranger/audit/kms
 sudo -u hdfs hdfs dfs -ls /ranger/audit/kms
```

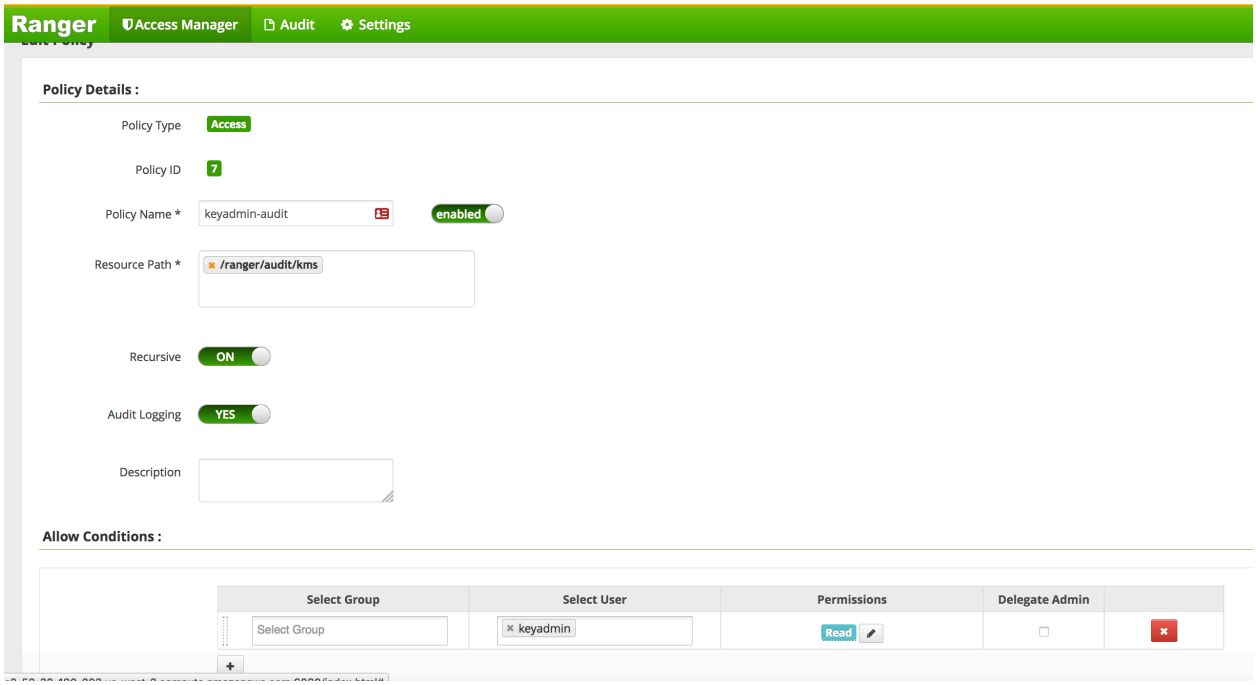
ii. Access Manager > HDFS > (clustername)_hdfs

iii. This will open the list of HDFS policies

iv. Create a new policy for keyadmin to be able to access /ranger/audit/kms and Save

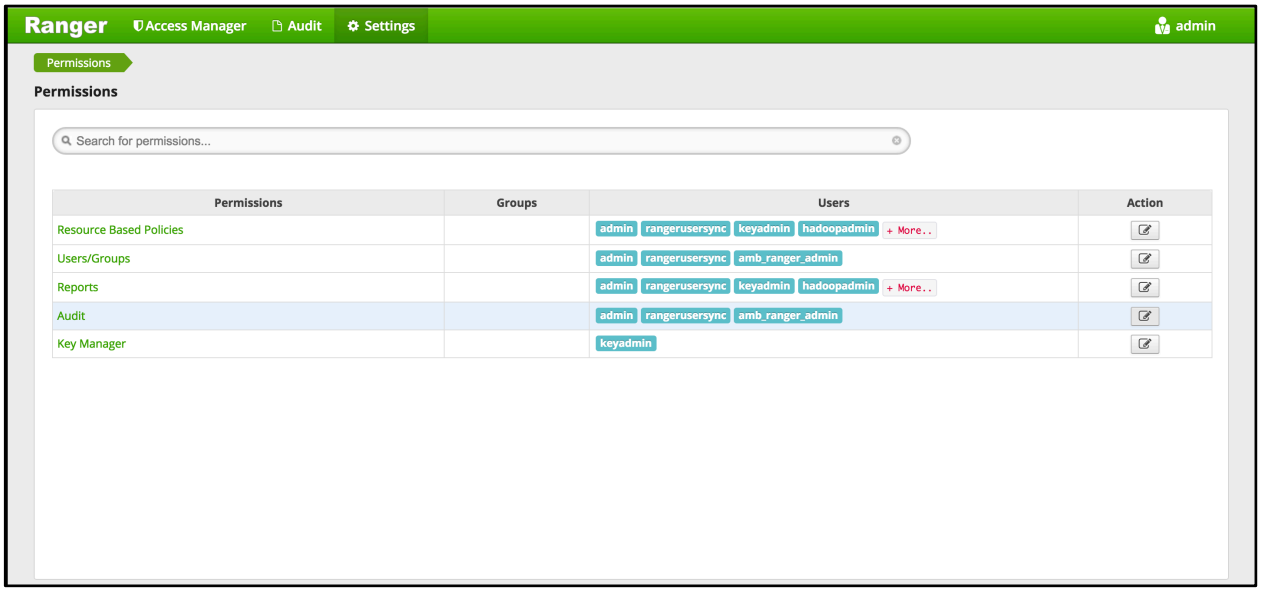


v. Your policy has been added



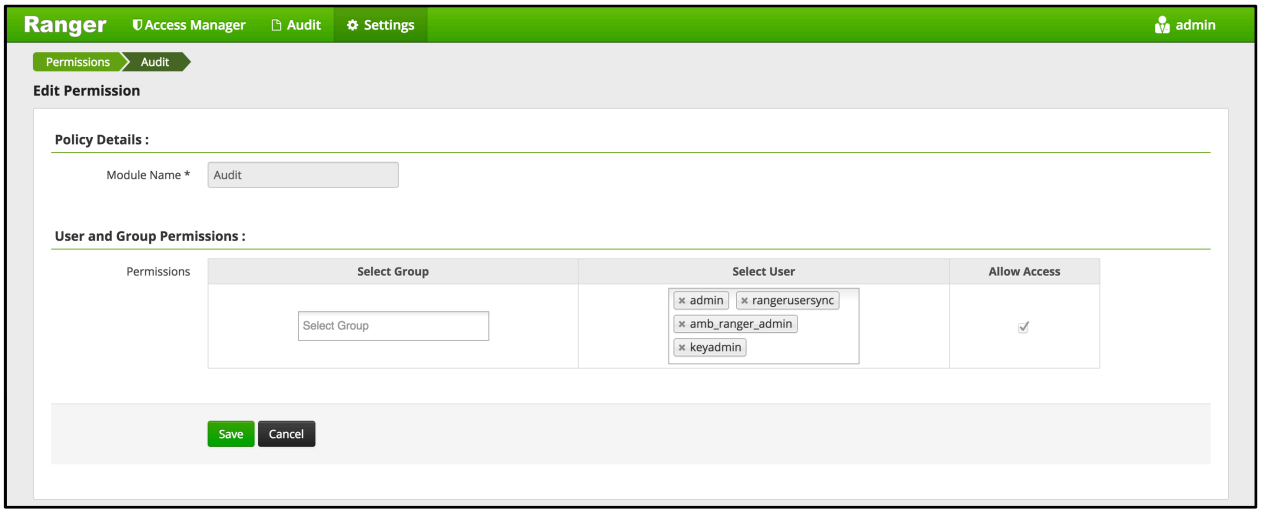
h. Give `keyadmin` permission to view Audits screen in Ranger:

i. `Settings tab > Permissions`



ii. Click `Audit` (second row from bottom) to change users who have access to Audit screen

iii. Under `Select User`, add `keyadmin` user



iv. `Save`

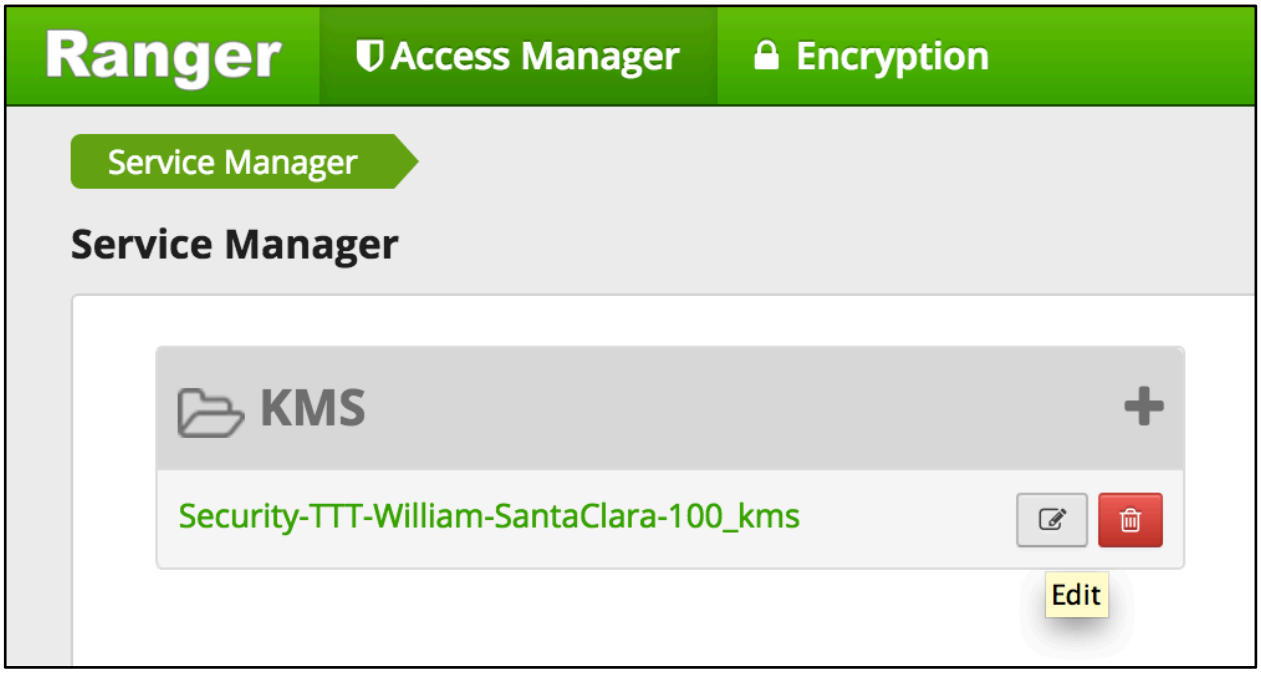
i. Logout of Ranger

i. `Top right > admin > Logout`

j. Login to Ranger as `keyadmin`/`keyadmin`

k. Confirm the KMS repo was setup correctly

i. Under `Service Manager > KMS >` Click the `Edit` icon (next to the `Trash` icon) to edit the KMS repo



ii. Click `Test connection`

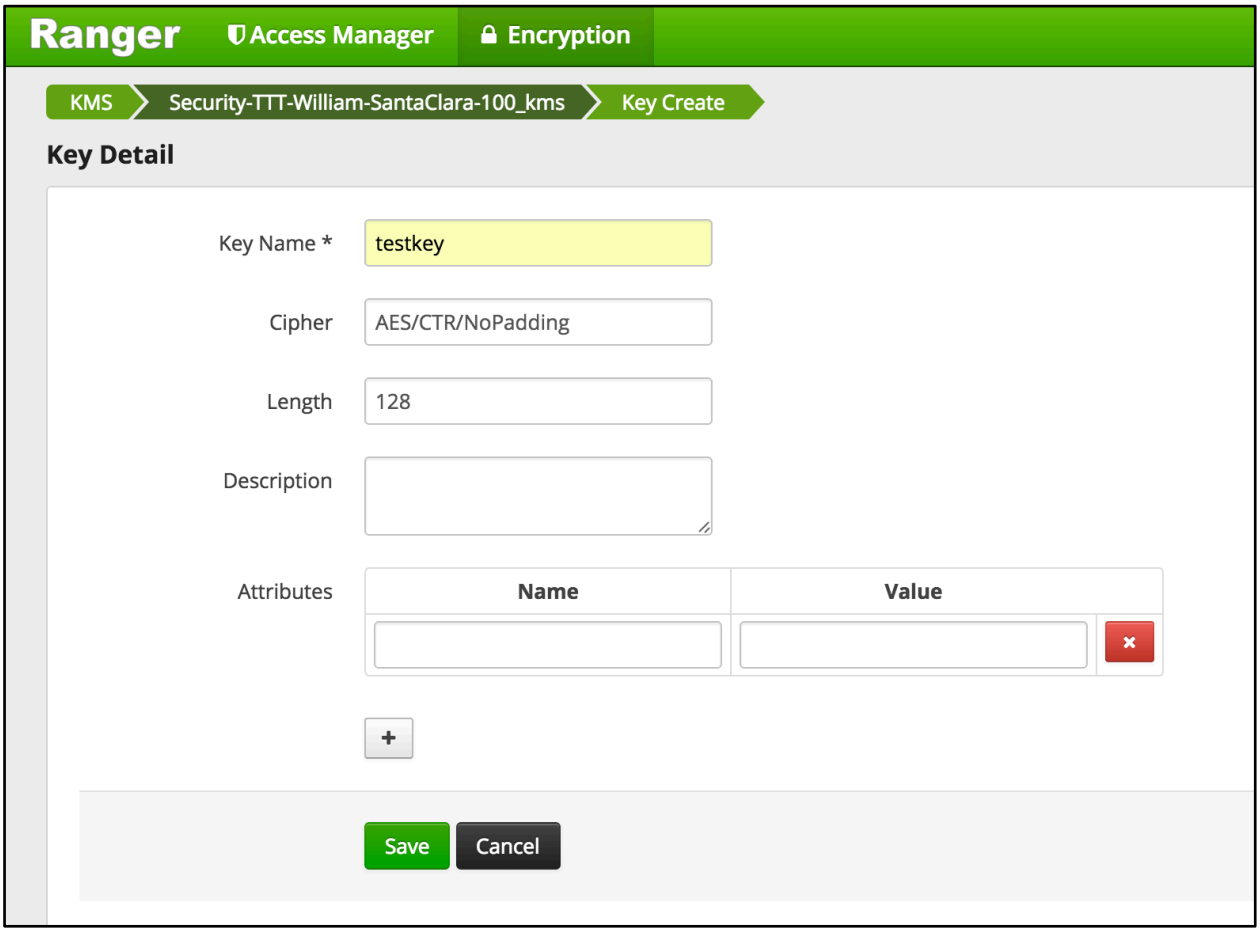
l. **Create a key called** `testkey`**; for reference, see** `http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.0/bk_security/content/use_ranger_kms.html`

   i. **Select** `Encryption > Key Manager`

   ii. **Select** `KMS service > pick your kms > Add new Key`

   If an error is thrown, go back and test connection as described in previous step

   iii. **Create a key called** `testkey > Save`



m. **Similarly, create another key called** `testkey2`

   i. **Select** `Encryption > Key Manager`

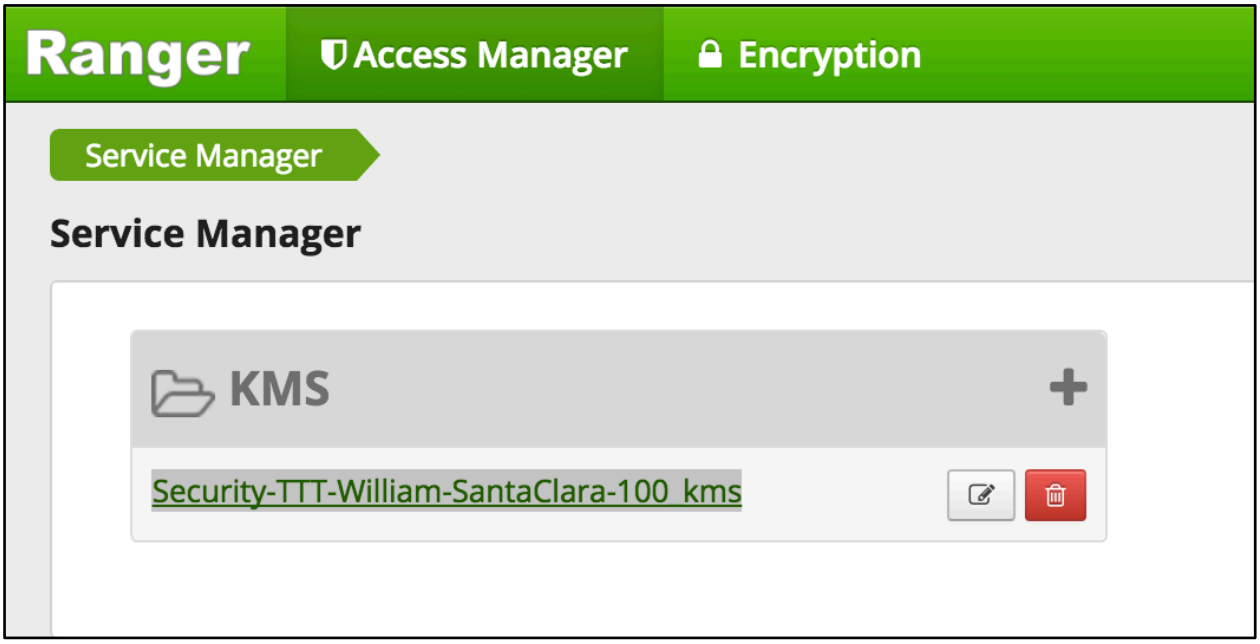
   ii. **Select** `KMS service > pick your kms > Add new Key`

   iii. **Create a key called** `testkey2 > Save`

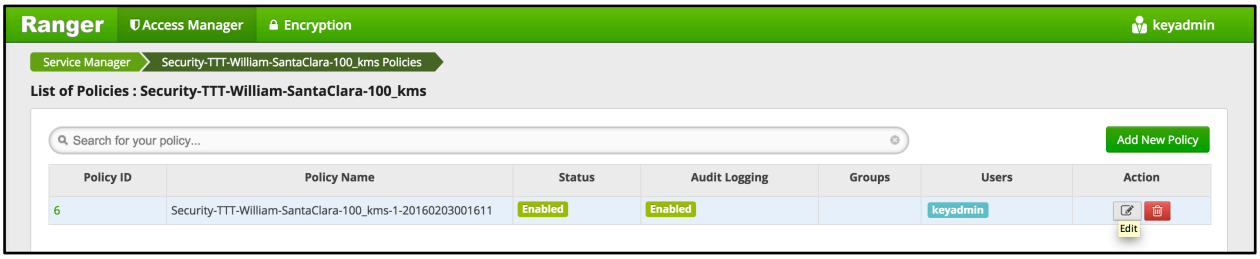
n. **Add users** `hadoopadmin`, `nn`, **and** `hive` **to default KMS key policy**

   i. **Click** `Access Manager` **tab**

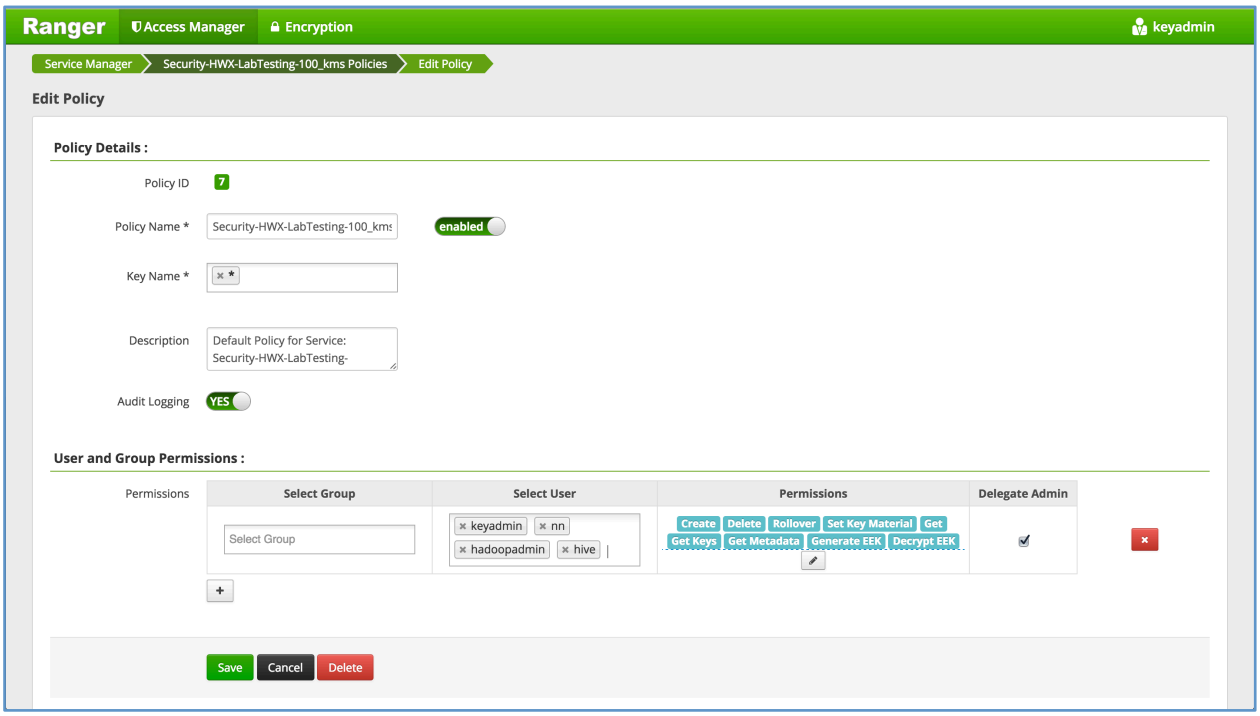
   ii. **Click** `Service Manager > KMS > (clustername)_kms link`



iii. Edit the default policy



iv. Under 'Select User', Add hadoopadmin users and click Save



v. **NOTE That:**

1. `nn` user needs `GetMetaData` and `GenerateEEK` privilege
2. `hive` user needs `GetMetaData` and `DecryptEEK` privilege

o. Run the commands below to create a zone using the key, and perform basic key and encryption zone (EZ) exercises:

i. Create EZs using keys

ii. Copy file to EZs

iii. Delete file from EZ

iv. View contents for raw file

v. Prevent access to raw file

vi. Copy file across EZs

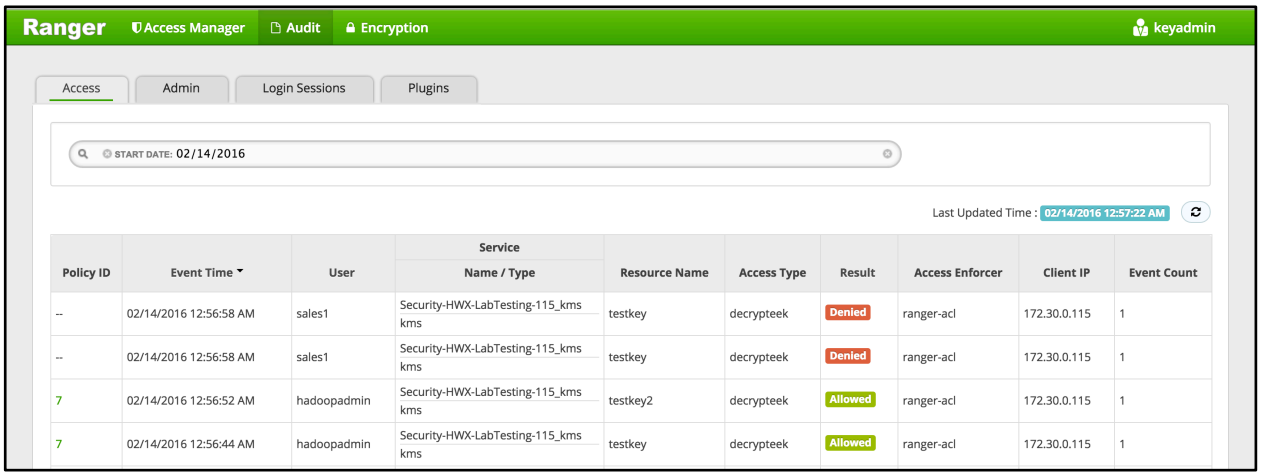
vii. move Hive warehouse `dir` to EZ

```
## run below on Ambari node

 export PASSWORD=BadPass#1

## detect name of cluster

 output=`curl -u hadoopadmin:$PASSWORD -k -i -H 'X-Requested-By: ambari'
https://localhost:8444/api/v1/clusters`

 cluster=`echo $output | sed -n 's/.*"cluster_name" : "\([^\"]*\)".*/\1/p'`

# echo $cluster

## this should show the name of your cluster
```

```
## if not you can manully set this as below
## cluster=Security-HWX-LabTesting-XXXX

## first we will run login 3 different users: hdfs, hadoopadmin, sales1

## kinit as hadoopadmin and sales using BadPass#1

 sudo -u hadoopadmin kinit
## enter BadPass#1

 sudo -u sales1 kinit
## enter BadPass#1

## then kinit as hdfs using the headless keytab and the principal name

 sudo -u hdfs kinit -kt /etc/security/keytabs/hdfs.headless.keytab hdfs-
${cluster}

## as hadoopadmin list the keys and their metadata
 sudo -u hadoopadmin hadoop key list -metadata

## as hadoopadmin create dirs for EZs

 sudo -u hadoopadmin hdfs dfs -mkdir /zone_encr

 sudo -u hadoopadmin hdfs dfs -mkdir /zone_encr2

## as hdfs create 2 EZs using the 2 keys

 sudo -u hdfs hdfs crypto -createZone -keyName testkey -path /zone_encr

 sudo -u hdfs hdfs crypto -createZone -keyName testkey2 -path /zone_encr2

## if you get 'RemoteException' error it means you have not given namenode
user permissions on testkey by creating a policy for KMS in Ranger

## check EZs got created

 sudo -u hdfs hdfs crypto -listZones

## create test files

 sudo -u hadoopadmin echo "My test file1" > /tmp/test1.log

 sudo -u hadoopadmin echo "My test file2" > /tmp/test2.log

## copy files to EZs

 sudo -u hadoopadmin hdfs dfs -copyFromLocal /tmp/test1.log /zone_encr

 sudo -u hadoopadmin hdfs dfs -copyFromLocal /tmp/test2.log /zone_encr

 sudo -u hadoopadmin hdfs dfs -copyFromLocal /tmp/test2.log /zone_encr2
```

```
## Notice that hadoopadmin allowed to decrypt EEK but not sales user (since
there is no Ranger policy allowing this)

 sudo -u hadoopadmin hdfs dfs -cat /zone_encr/test1.log

 sudo -u hadoopadmin hdfs dfs -cat /zone_encr2/test2.log

## this should work

 sudo -u sales1      hdfs dfs -cat /zone_encr/test1.log

## this should give you below error
## cat: User:sales1 not allowed to do 'DECRYPT_EEK' on 'testkey'
```

p. Check the `Ranger > Audit` page and notice that the request from `hadoopadmin` was allowed, but the request from `sales1` was denied



Policy ID	Event Time	User	Service Name / Type	Resource Name	Access Type	Result	Access Enforcer	Client IP	Event Count
--	02/14/2016 12:56:58 AM	sales1	Security-HWX-LabTesting-115_kms kms	testkey	decrypteek	Denied	ranger-acl	172.30.0.115	1
--	02/14/2016 12:56:58 AM	sales1	Security-HWX-LabTesting-115_kms kms	testkey	decrypteek	Denied	ranger-acl	172.30.0.115	1
7	02/14/2016 12:56:52 AM	hadoopadmin	Security-HWX-LabTesting-115_kms kms	testkey2	decrypteek	Allowed	ranger-acl	172.30.0.115	1
7	02/14/2016 12:56:44 AM	hadoopadmin	Security-HWX-LabTesting-115_kms kms	testkey	decrypteek	Allowed	ranger-acl	172.30.0.115	1

q. Now lets test deleting and copying files between EZs. For reference, see `http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.0/bk_security/content/copy-to-from-encr-zone.html`

```
## try to remove file from EZ using usual -rm command

 sudo -u hadoopadmin hdfs dfs -rm /zone_encr/test2.log

## rm: Failed to move to trash.... /zone_encr/test2.log can't be moved from
an encryption zone.

## recall that to delete a file from EZ you need to specify the skipTrash
option

 sudo -u hadoopadmin hdfs dfs -rm -skipTrash /zone_encr/test2.log

## confirm that test2.log was deleted and that zone_encr only contains
test1.log

 sudo -u hadoopadmin hdfs dfs -ls  /zone_encr/

## copy a file between EZs using distcp with -skipcrccheck option
 sudo -u hadoopadmin hadoop distcp -skipcrccheck -update
/zone_encr2/test2.log /zone_encr/
```

r. Let's now look at the contents of the raw file:

```
## View contents of raw file in encrypted zone as hdfs super user. This
should show some encrypted characters

 sudo -u hdfs hdfs dfs -cat /.reserved/raw/zone_encr/test1.log

## Prevent user hdfs from reading the file by setting
security.hdfs.unreadable.by.superuser attribute. Note that this attribute
can only be set on files and can never be removed.

 sudo -u hdfs hdfs dfs -setfattr -n security.hdfs.unreadable.by.superuser
/.reserved/raw/zone_encr/test1.log

## Now as hdfs super user, try to read the files or the contents of the raw
file

 sudo -u hdfs hdfs dfs -cat /.reserved/raw/zone_encr/test1.log

## You should get below error
## cat: Access is denied for hdfs since the superuser is not allowed to
perform this operation.
```

s. Configure Hive for HDFS Encryption using `testkey`. See `http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.0/bk_security/content/hive-access-encr.html`

```
 sudo -u hadoopadmin hdfs dfs -mv /apps/hive /apps/hive-old
 sudo -u hadoopadmin hdfs dfs -mkdir /apps/hive
 sudo -u hdfs hdfs crypto -createZone -keyName testkey -path /apps/hive
 sudo -u hadoopadmin hadoop distcp -skipcrccheck -update /apps/hive-
old/warehouse /apps/hive/warehouse
```

t. To configure the Hive scratch directory (`hive.exec.scratchdir`) so that it resides inside the encryption zone:

   i. `Ambari > Hive > Configs > Advanced`

      - `hive.exec.scratchdir = /apps/hive/tmp`

   ii. Restart Hive

u. Configure Tez for EZ (this is a bug that should be fixed in next release)

   i. `Ambari > Tez > Configs > Custom tez-site`

      - `tez.dag.recovery.enabled = false`

   ii. Restart Tez

v. Make sure that the permissions for `/apps/hive/tmp` are set to `1777`

```
 sudo -u hdfs hdfs dfs -chmod -R 1777 /apps/hive/tmp
```

w. Confirm permissions by accessing the `scratch dir` as `sales1`

```
 sudo -u sales1 hdfs dfs -ls /apps/hive/tmp
## this should provide listing
```

x. Destroy ticket for `sales1`

```
sudo -u sales1 kdestroy
```

y. Logout of Ranger as `keyadmin` user

RESULT:

You successfully installed Ranger KMS using Ambari. You also copied files into encryption zones successfully.

Lab: Secured Hadoop Exercises

About This Lab

Objective:	In this lab we will see how to interact with Hadoop components (HDFS, Hive, Hbase, Sqoop) running on a kerborized cluster and create Ranger appropriate authorization policies for access.
File locations:	N/A
Successful outcome:	Successfully access interacted with Hadoop components in secured mode and used Ranger to manage authorization policies and audits.
Before you begin	N/A
Related lesson:	N/A

Lab Steps

Perform the following steps:

We will Configure Ranger policies to:

- Protect /sales HDFS dir - so only sales group has access to it
- Protect sales hive table - so only sales group has access to it
- Protect sales HBase table - so only sales group has access to it

1. **Access Secured HDFS**
   a. Create a `/sales` directory in HDFS and ensure only users belonging to `sales` group (and admins) have access.
   b. Login to Ranger (using `admin`/`admin`), and confirm the HDFS repo was setup correctly in Ranger
      i. In `Ranger > Under Service Manager > HDFS >` Click the `Edit` icon (next to the `Trash` icon) to edit the HDFS repo
      ii. Click `Test connection`
      iii. If it fails, enter below fields and retry:
         1. Username: `rangeradmin@LAB.HORTONWORKS.NET`
         2. Password: `BadPass#1`
   c. Once the test passes, click `Save`
   d. Create `/sales` dir in HDFS as `hadoopadmin`

```
## authenticate
 sudo -u hadoopadmin kinit
## enter password: BadPass#1
```

```
##create dir and set permissions to 000
 sudo -u hadoopadmin hadoop fs -mkdir /sales
 sudo -u hadoopadmin hadoop fs -chmod 000 /sales
```

e. Now login as sales1 and attempt to access it before adding any Ranger HDFS policy:

```
 sudo su - sales1
 hdfs dfs -ls /sales
```

f. This fails with `GSSException: No valid credentials provided` because the cluster is Kerberized, and we have not authenticated yet

g. Authenticate as `sales1` user and check the ticket:

```
# kinit
##enter password: BadPass#1

# klist
## Default principal: sales1@LAB.HORTONWORKS.NET
```

h. Now try accessing the directory again as `sales1`:

```
# hdfs dfs -ls /sales
```

i. This time it fails with authorization error:

```
Permission denied: user=sales1, access=READ_EXECUTE,
inode="/sales":hadoopadmin:hdfs:d---------
```

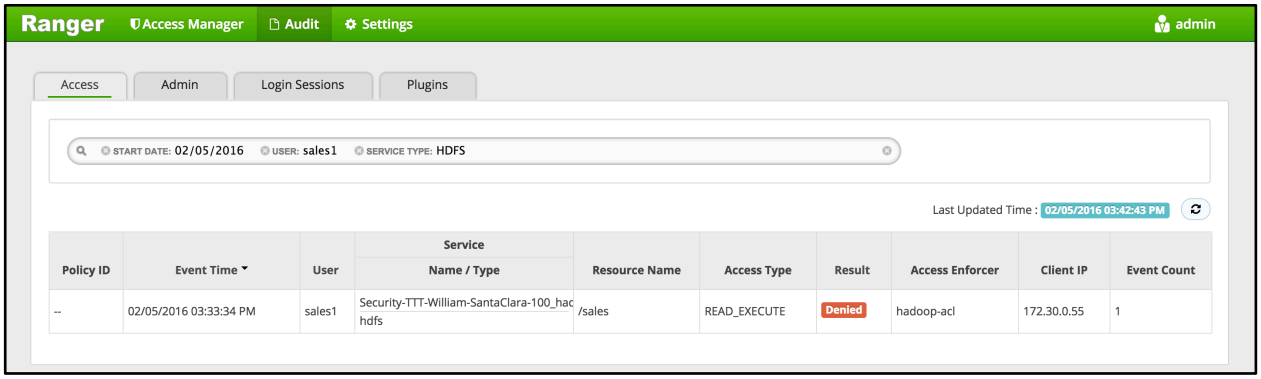
j. Login into Ranger UI, at `http://RANGER_HOST_PUBLIC_IP:6080/index.html` as `admin/admin`

k. In Ranger, click on `Audit` to open the `Audits` page and filter by below:

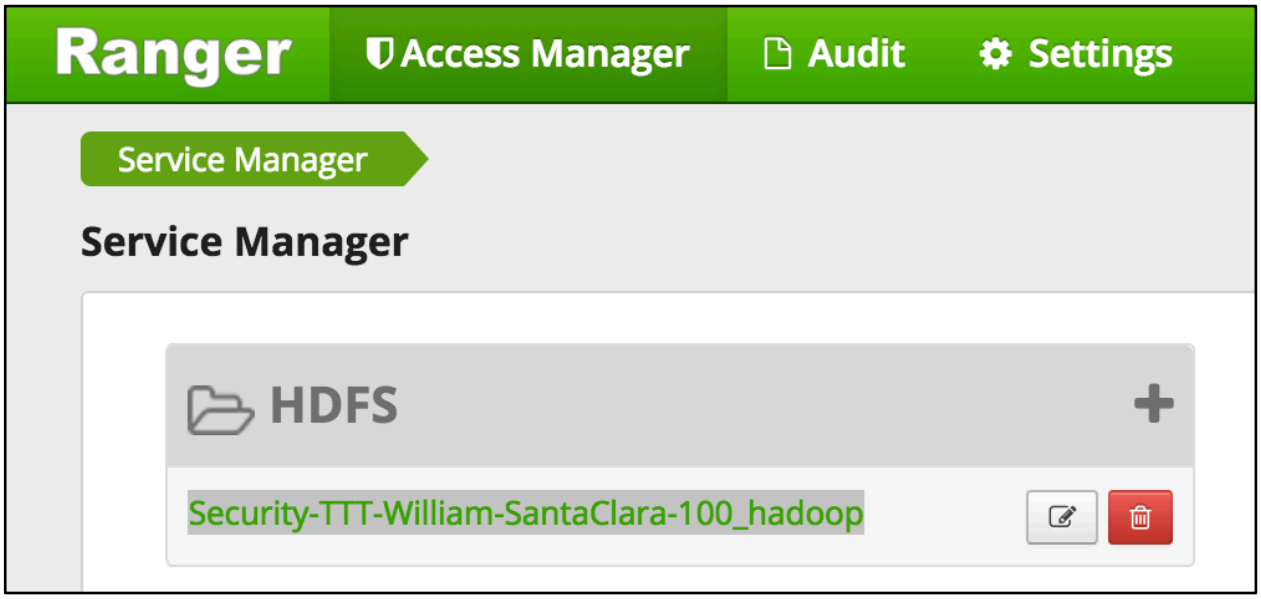
   i. Service Type: HDFS

   ii. User: `sales1`

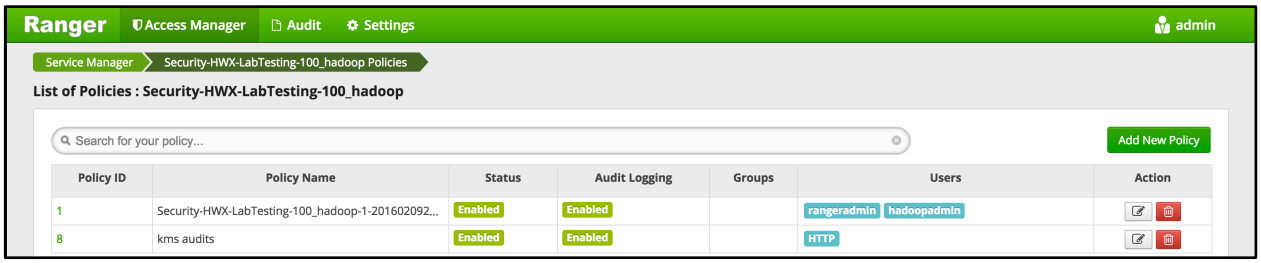
l. Notice that Ranger captured the access attempt and since there is currently no policy to allow the access, it was `Denied`



m. To create an HDFS Policy in Ranger, on the 'Access Manager' tab click `HDFS > (clustername)_hadoop`



i. This will open the list of HDFS policies:



n. Click `Add New Policy` button to create a new one, allowing `sales` group users access to `/sales` dir:

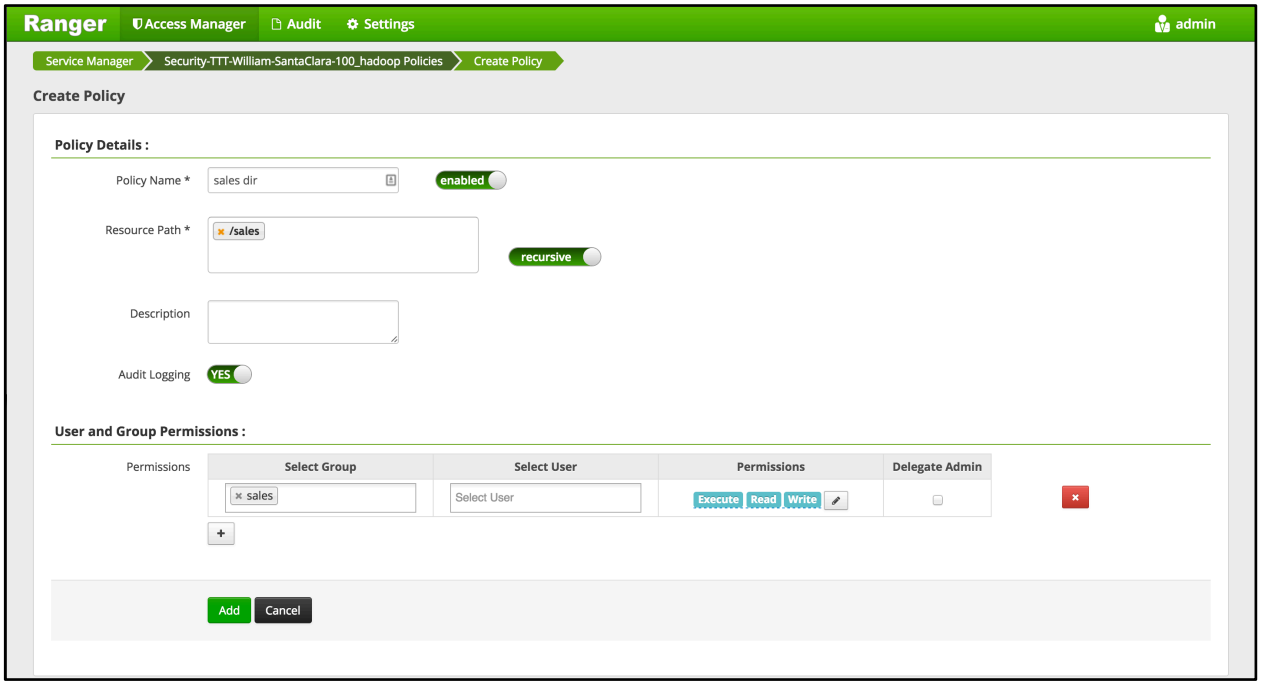
   i. Policy Name: `sales dir`

   ii. Resource Path: `/sales`

   iii. Group: `sales`

   iv. Permissions: `Execute Read Write`

   v. `Add`



o. Wait 30 seconds for policy to take effect

p. Now try accessing the `dir` again as `sales1` and now there is no error seen

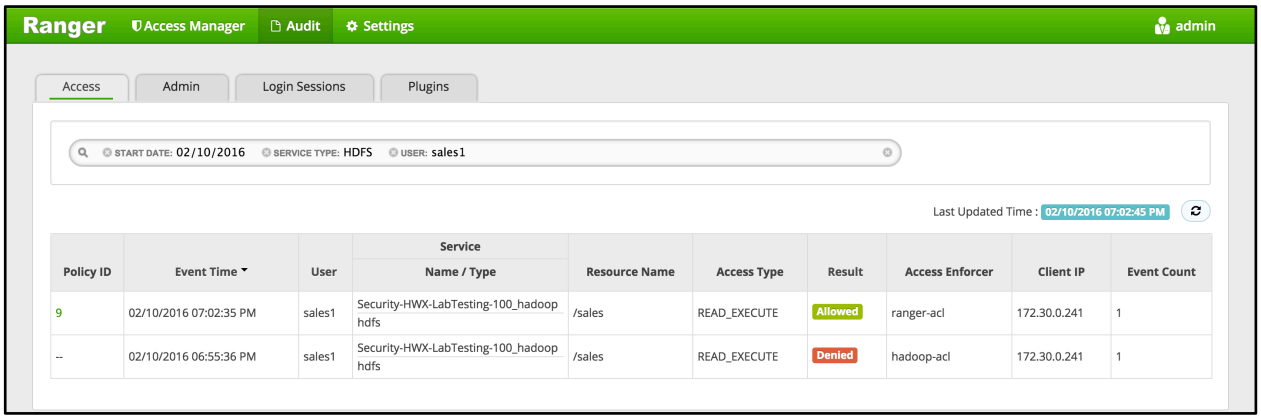
```
hdfs dfs -ls /sales
```

q. In Ranger, click on `Audit` to open the `Audits` page and filter by below:

   i. Service Type: HDFS

   ii. User: `sales1`

r. Notice that Ranger captured the access attempt, and since this time there is a policy to allow the access, it was `Allowed`



   i. You can also see the details that were captured for each request:

      1. Policy that allowed the access

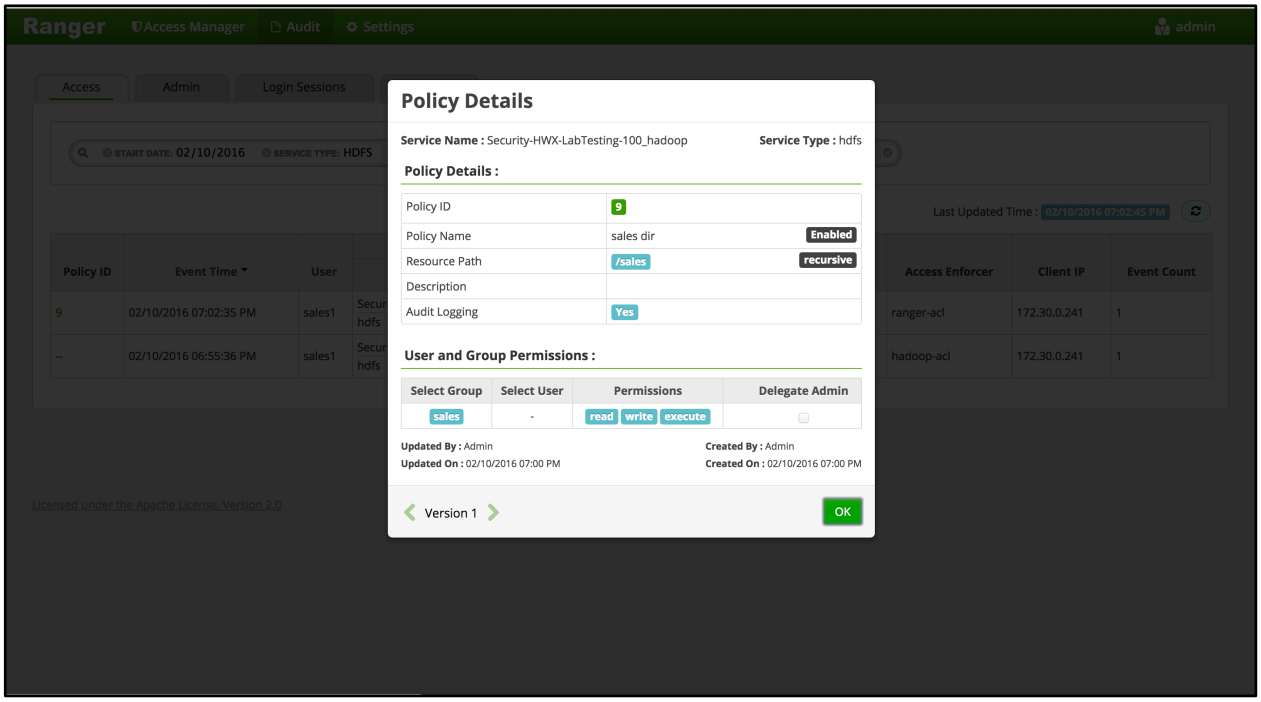
      2. Time

      3. Requesting user

      4. Service type (e.g. HDFS, Hive, HBase etc)

5. Resource name
6. Access type (e.g. read, write, execute)
7. Result (e.g. allowed or denied)
8. Access enforcer (i.e. whether native acl or Ranger acls were used)
9. Client IP
10. Event count

s. For any allowed requests, notice that you can quickly check the details of the policy that allowed the access by clicking on the policy number in the `Policy ID` column



3. Now let's check whether non-`sales` users can access the directory
4. Logout as `sales1` and log back in as `hr1`

```
 Kdestroy

## logout as sales1
 logout

## login as hr1 and authenticate
 sudo su - hr1

 kinit
## enter password: BadPass#1

 klist
## Default principal: hr1@LAB.HORTONWORKS.NET
```

a. Try to access the same directory as `hr1` and notice it fails

```
hdfs dfs -ls /sales
## ls: Permission denied: user=hr1, access=READ_EXECUTE,
inode="/sales":hadoopadmin:hdfs:d---------
```

b. In Ranger, click on `Audit` to open the `Audits` page and this time filter by `Resource Name`

   i. Service Type: HDFS

   ii. Resource Name: `/sales`

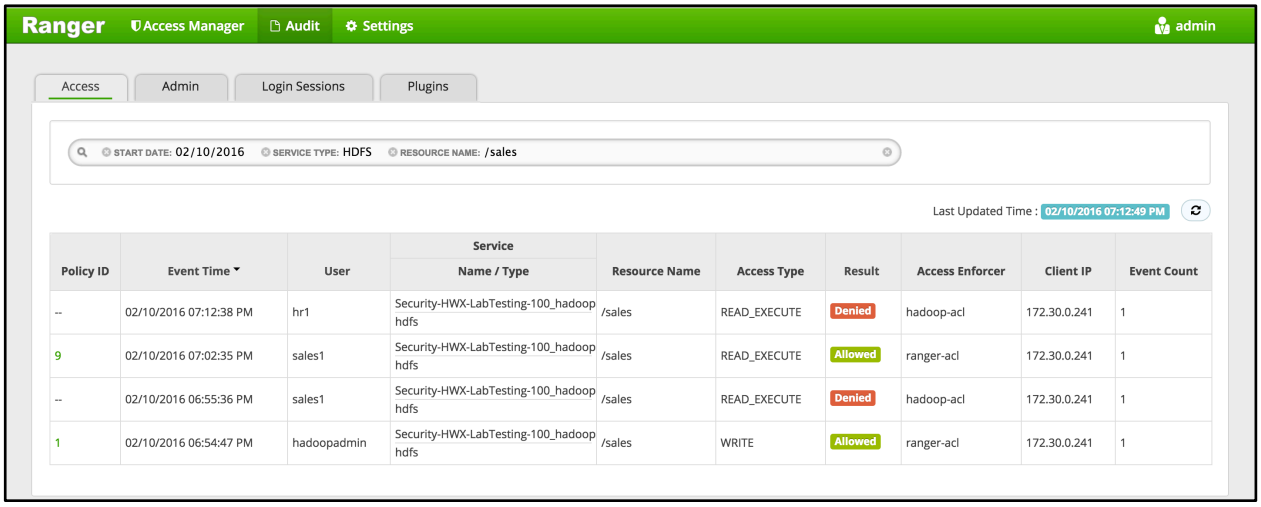
c. Notice you can see the history/details of all the requests made for /sales directory:

   i. Created by `hadoopadmin`

   ii. Initial request by `sales1` user was denied

   iii. Subsequent request by `sales1` user was allowed (once the policy was created)

   iv. Request by `hr1` user was denied



Policy ID	Event Time	User	Service Name / Type	Resource Name	Access Type	Result	Access Enforcer	Client IP	Event Count
--	02/10/2016 07:12:38 PM	hr1	Security-HWX-LabTesting-100_hadoop hdfs	/sales	READ_EXECUTE	Denied	hadoop-acl	172.30.0.241	1
9	02/10/2016 07:02:35 PM	sales1	Security-HWX-LabTesting-100_hadoop hdfs	/sales	READ_EXECUTE	Allowed	ranger-acl	172.30.0.241	1
--	02/10/2016 06:55:36 PM	sales1	Security-HWX-LabTesting-100_hadoop hdfs	/sales	READ_EXECUTE	Denied	hadoop-acl	172.30.0.241	1
1	02/10/2016 06:54:47 PM	hadoopadmin	Security-HWX-LabTesting-100_hadoop hdfs	/sales	WRITE	Allowed	ranger-acl	172.30.0.241	1

d. Logout as `hr1`:

```
kdestroy
logout
```

e. We have successfully setup an HDFS directory which is only accessible by the `sales` group (and admins)

2. Access Secured Hive

a. Setup Hive authorization policies to ensure `sales` users only have access to code, description columns in `default.sample_07`.

b. Enable Hive on Tez by setting below and restarting Hive

   i. `Ambari > Hive > Configs`

      1. Execution Engine = Tez

c. Confirm the HIVE repo was setup correctly in Ranger

d. In `Ranger > Service Manager > HIVE >` Click the `Edit` icon (next to the `Trash` icon) to edit the HIVE repo

i. Click `Test connection`

ii. If it fails, enter below fields and retry:

1. Username: `rangeradmin@LAB.HORTONWORKS.NET`

2. Password: `BadPass#1`

iii. Once the test passes, click `Save`

e. Now, run these steps from node where Hive (or client) is installed

f. Login as `sales1` and attempt to connect to default database in Hive via beeline, and access `sample_07` table

g. Notice that in the JDBC connect string for connecting to an secured Hive while its running in default (i.e. binary) transport mode:

i. Port remains `10000`

ii. Now a Kerberos principal needs to be passed in

h. Login as `sales1` without a Kerberos ticket, and try to open beeline connection:

```
 sudo su - sales1
 kdestroy
 beeline -u
"jdbc:hive2://localhost:10000/default;principal=hive/$(hostname -
f)@LAB.HORTONWORKS.NET"
```

i. This fails with `GSS initiate failed` because the cluster is Kerberized and we have not authenticated yet

j. To exit beeline:

```
 !q
```

k. Authenticate as `sales1` user and check the ticket:

```
 kinit
## enter password: BadPass#1

 klist
## Default principal: sales1@LAB.HORTONWORKS.NET
```

l. Now try connect to Hive via beeline as `sales1`:

```
 beeline -u
"jdbc:hive2://localhost:10000/default;principal=hive/$(hostname -
f)@LAB.HORTONWORKS.NET"
```

m. **NOTE:** If you get the below error, it is because you did not add Hive to the global KMS policy in an earlier step (along with `nn` and `hadoopadmin`). Go back and add it in:

```
 org.apache.hadoop.security.authorize.AuthorizationException:
User:hive not allowed to do 'GET_METADATA' on 'testkey'
```

n. This time it connects. Now try to run a query:

```
   # beeline> select code, description from sample_07;
```

o. Now it fails with authorization error:

```
HiveAccessControlException Permission denied: user [sales1] does
not have [SELECT] privilege on [default/sample_07]
```

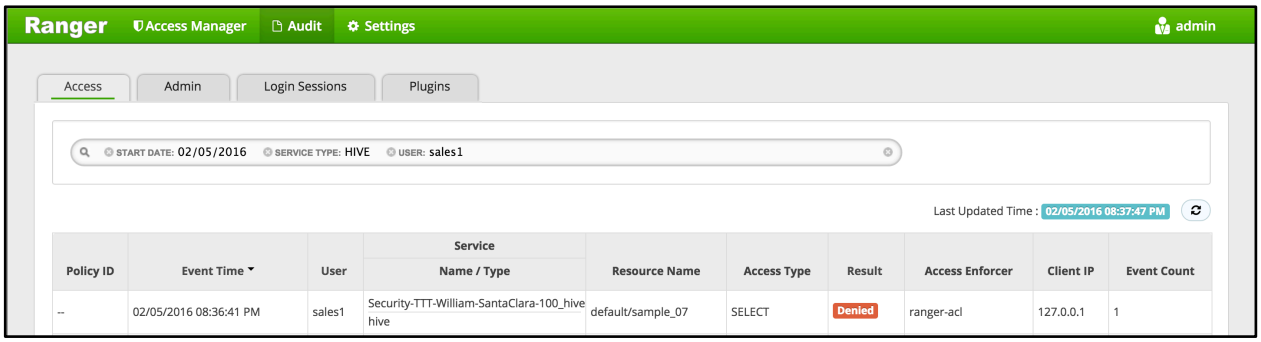
p. Login into Ranger UI at `http://RANGER_HOST_PUBLIC_IP:6080/index.html` as `admin/admin`

q. In Ranger, click on 'Audit' to open the Audits page and filter by below.

   i. Service Type: Hive

   ii. User: `sales1`

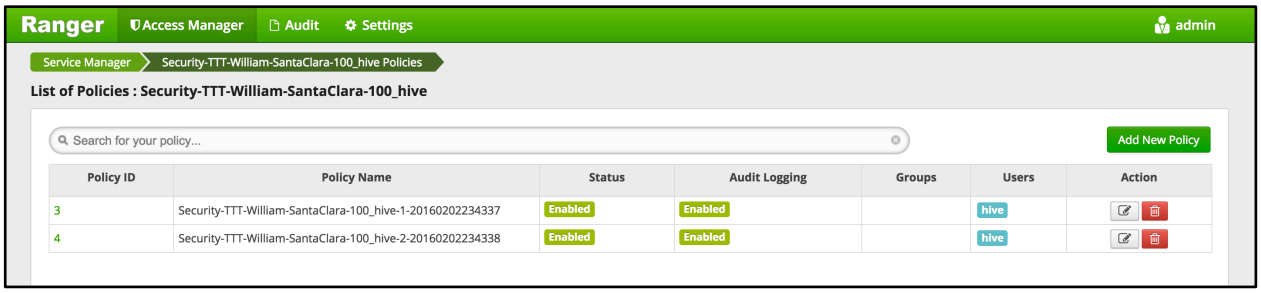
r. Notice that Ranger captured the access attempt and since there is currently no policy to allow the access, it was `Denied`



s. To create an HIVE Policy in Ranger, follow below steps:

t. On the `Access Manager` tab, click `HIVE > (clustername)_hive`

u. This will open the list of HIVE policies



v. Click `Add New Policy` button to create a new one allowing `sales` group users access to `code` and `description` columns in `sample_07` table:

   i. Policy Name: `sample_07`

   ii. Hive Database: default

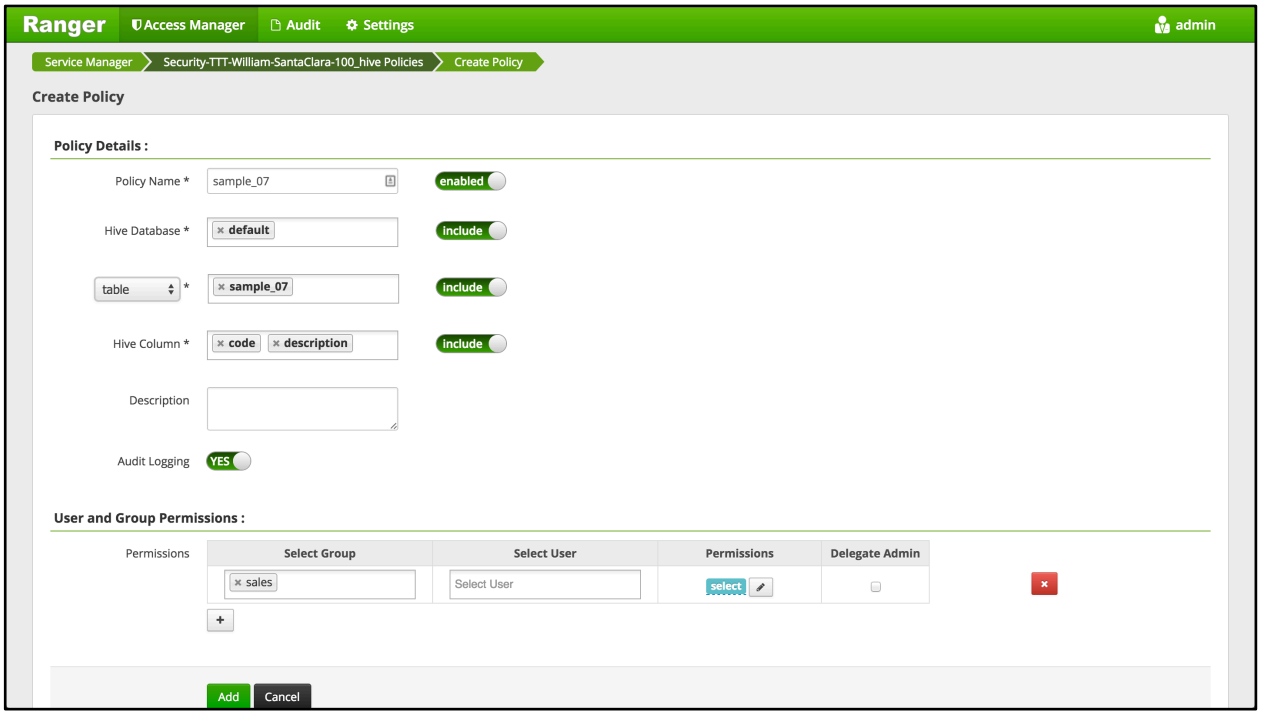
   iii. Table: `sample_07`

iv. Hive Column: `code description`

v. Group: `sales`

vi. Permissions : `select`

vii. Add



viii. Notice that as you typed the name of the DB and table, Ranger was able to look these up and autocomplete them; this was done using the `rangeradmin` principal we provided during Ranger install.

w. Wait 30 second for the new policy to be picked up

x. Now try accessing the columns again, and now the query works

```
beeline> select code, description from sample_07;
```

y. Note though, that if instead you try to `describe` the table or query `all` columns, it will be denied - because we only gave `sales` users access to two columns in the table

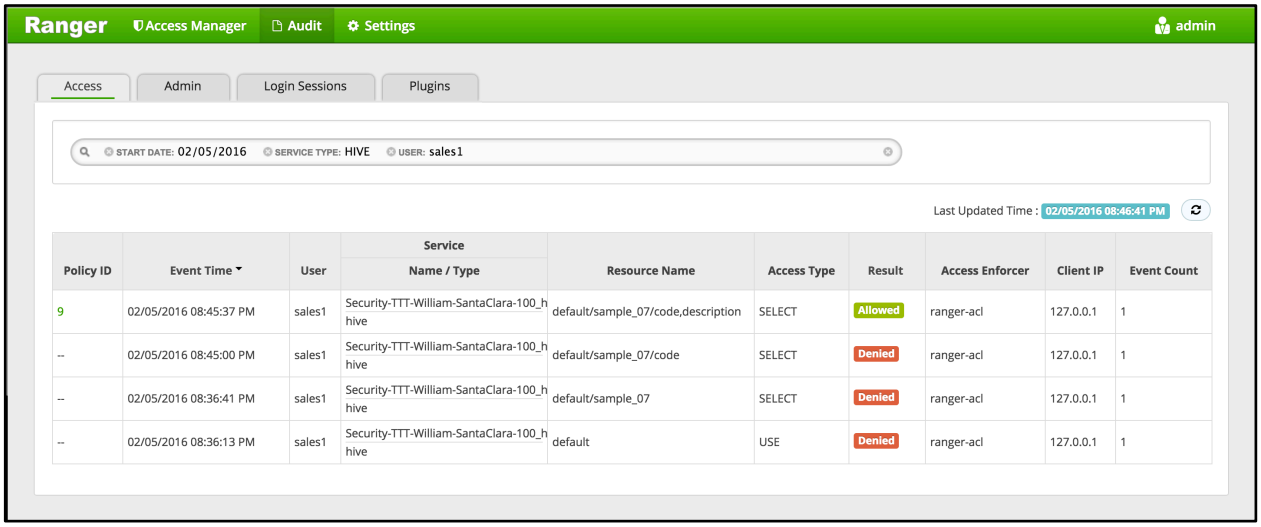
```
beeline> desc sample_07;
beeline> select * from sample_07;
```

z. In Ranger, click on `Audit` to open the `Audits` page and filter by below:

i. Service Type: HIVE

ii. User: `sales1`

aa. Notice that Ranger captured the access attempt and since this time there is a policy to allow the access, it was `Allowed`



Policy ID	Event Time	User	Service Name / Type	Resource Name	Access Type	Result	Access Enforcer	Client IP	Event Count
9	02/05/2016 08:45:37 PM	sales1	Security-TTT-William-SantaClara-100_hive	default/sample_07/code,description	SELECT	Allowed	ranger-acl	127.0.0.1	1
--	02/05/2016 08:45:00 PM	sales1	Security-TTT-William-SantaClara-100_hive	default/sample_07/code	SELECT	Denied	ranger-acl	127.0.0.1	1
--	02/05/2016 08:36:41 PM	sales1	Security-TTT-William-SantaClara-100_hive	default/sample_07	SELECT	Denied	ranger-acl	127.0.0.1	1
--	02/05/2016 08:36:13 PM	sales1	Security-TTT-William-SantaClara-100_hive	default	USE	Denied	ranger-acl	127.0.0.1	1

bb. You can also see the details that were captured for each request:

i. Policy that allowed the access

ii. Time

iii. Requesting user

iv. Service type (e.g. HDFS, Hive, HBase etc)

v. Resource name

vi. Access type (e.g. read, write, execute)

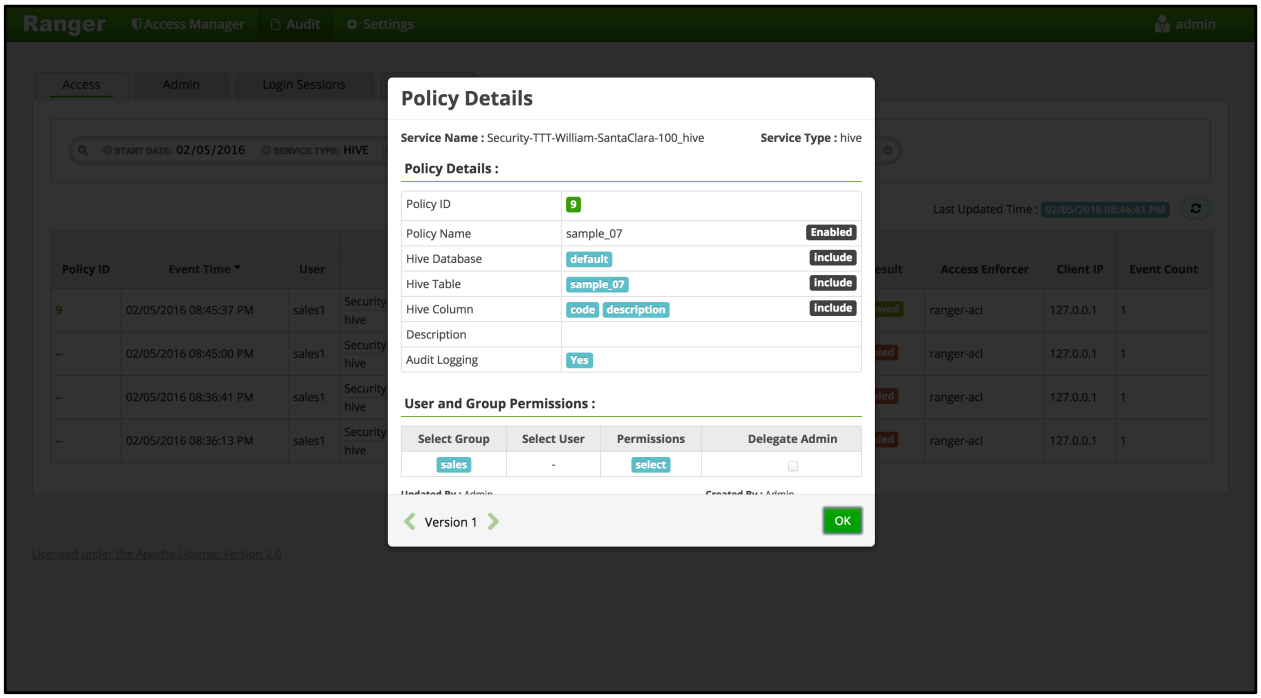
vii. Result (e.g. allowed or denied)

viii. Access enforcer (i.e. whether native acl or Ranger acls were used)

ix. Client IP

x. Event count

cc. For any allowed requests, notice that you can quickly check the details of the policy that allowed the access by clicking on the policy number in the `Policy ID` column



dd. Exit beeline

```
!q
```

ee. Now let's check whether non-sales users can access the table

ff. Logout as `sales1` and log back in as `hr1`:

```
kdestroy

## logout as sales1
logout

## login as hr1 and authenticate
sudo su - hr1

kinit
## enter password: BadPass#1

klist
## Default principal: hr1@LAB.HORTONWORKS.NET
```

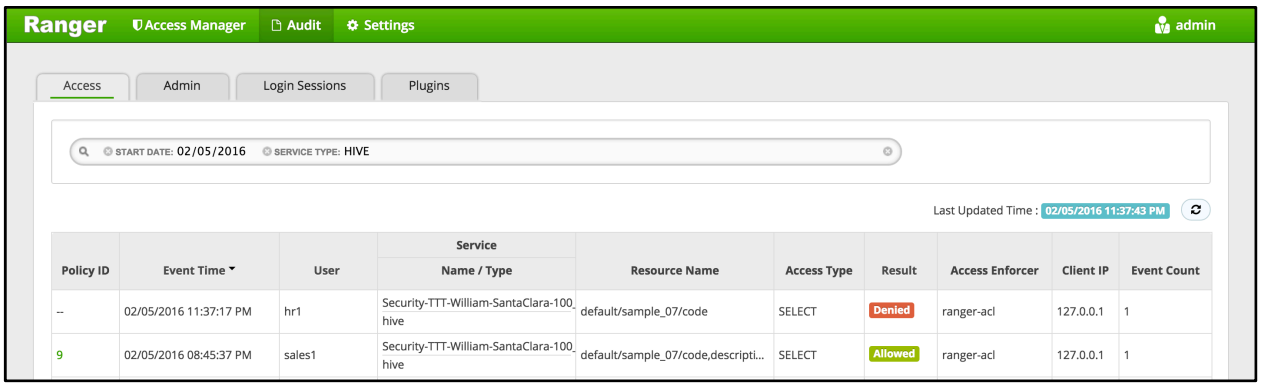
gg. Try to access the same table as `hr1` and notice it fails:

```
beeline -u
"jdbc:hive2://localhost:10000/default;principal=hive/$(hostname -
f)@LAB.HORTONWORKS.NET"

beeline> select code, description from sample_07;
```

hh. In Ranger, click on `Audit` to open the `Audits` page and filter by `Service Type = Hive`

   i. Service Type: HIVE

ii. Here you can see the request by `sales1` was allowed but `hr1` was denied:



e

jj. Exit beeline:

```
!q
```

kk. Logoff as `hr1`:

```
logout
```

ll. We have setup Hive authorization policies to ensure only `sales` users have access to `code`, `description` columns in `default.sample_07`

3. Access Secured HBase

a. Create a table called 'sales' in HBase and setup authorization policies to ensure only sales users have access to the table

b. Run these steps from any node where HBase Master or RegionServer services are installed

c. Login as `sales1`

```
sudo su - sales1
```

d. Start the HBase shell:

```
hbase shell
```

e. List tables in `default` database

```
hbase> list 'default'
```

f. This fails with `GSSException: No valid credentials provided` because the cluster is Kerberized and we have not authenticated yet

g. To exit HBase shell:

```
exit
```

h. Authenticate as `sales1` user and check the ticket:

```
kinit
#enter password: BadPass#1

klist
## Default principal: sales1@LAB.HORTONWORKS.NET
```

i. Now try connect to Hbase shell and list tables as `sales1`:

```
 hbase shell
hbase> list 'default'
```

j. This time it works. Now try to create a table called `sales` with column family called `cf`:

```
 hbase> create 'sales', 'cf'
```

k. Now it fails with authorization error:

```
org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient
permissions for user 'sales1@LAB.HORTONWORKS.NET' (action=create)
```

l. **NOTE:** There will be a lot of output from above. The error will be on the line right after your create command

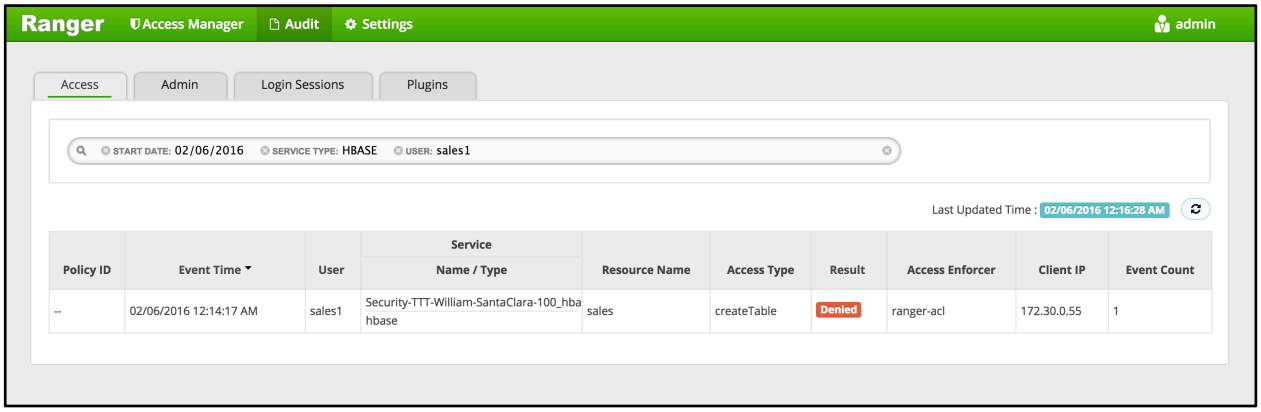
m. Login into Ranger UI at `http://RANGER_HOST_PUBLIC_IP:6080/index.html` as `admin`/`admin`

n. In Ranger, click on `Audit` to open the `Audits` page and filter by below.

   i. Service Type: Hbase

   ii. User: `sales1`

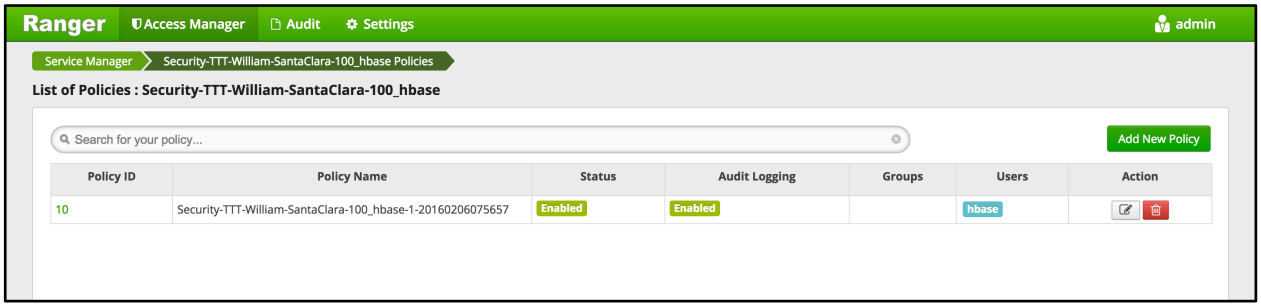
o. Notice that Ranger captured the access attempt and since there is currently no policy to allow the access, it was `Denied`



p. To create an HBASE Policy in Ranger, follow below steps:

q. On the `Access Manager` tab click `HBASE > (clustername)_hbase`

r. This will open the list of HBASE policies



s. Click `Add New Policy` button to create a new one allowing `sales` group users access to `sales` table in HBase:

   i. Policy Name: `sales`

   ii. Hbase Table: `sales`

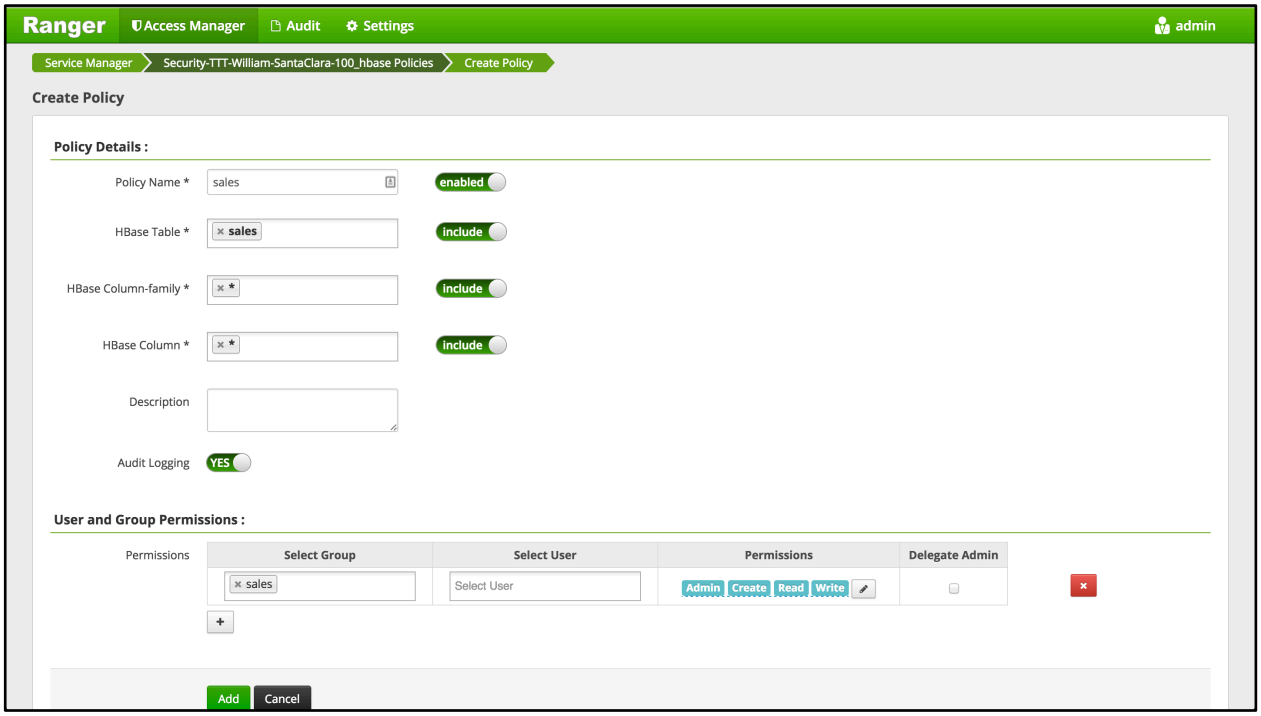
   iii. Hbase Column Family: `*`

   iv. Hbase Column: `*`

   v. Group : `sales`

   vi. Permissions : `Admin Create Read Write`

   vii. `Add`



t. Wait 30 seconds for the policy to take effect

u. Now try creating the table and now it works:

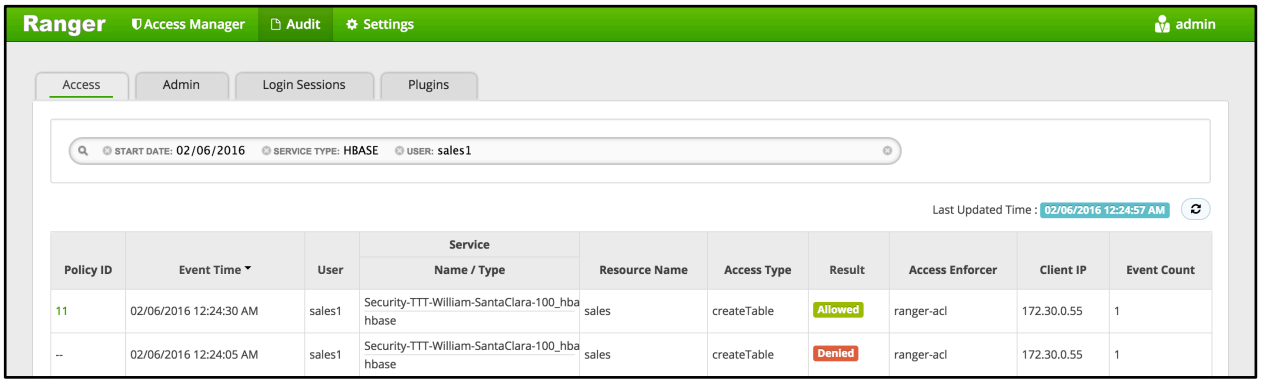
```
hbase> create 'sales', 'cf'
```

v. In Ranger, click on `Audit` to open the `Audits` page and filter by below:

   i. Service Type: HBASE

ii. User: `sales1`

w. Notice that Ranger captured the access attempt and since this time there is a policy to allow the access, it was `Allowed`



x. You can also see the details that were captured for each request:

i. Policy that allowed the access

ii. Time

iii. Requesting user

iv. Service type (e.g. hdfs, hive, hbase etc)

v. Resource name

vi. Access type (e.g. read, write, execute)

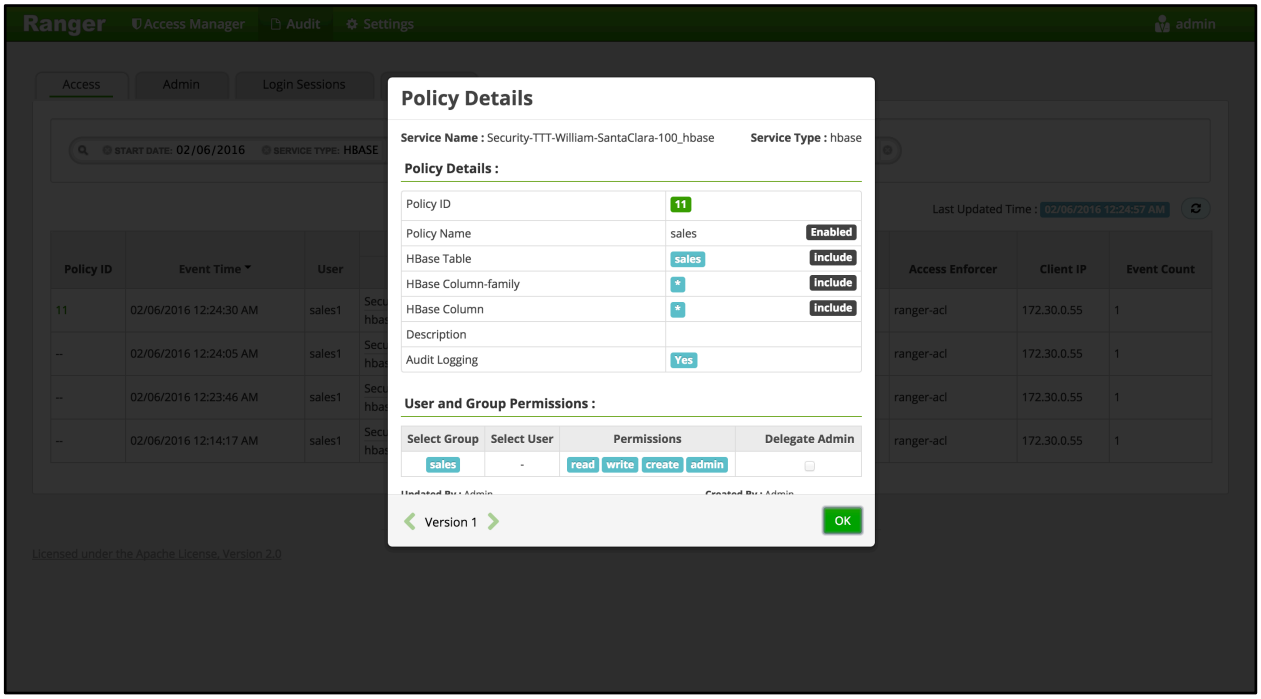
vii. Result (e.g. allowed or denied)

viii. Access enforcer (i.e. whether native acl or ranger acls were used)

ix. Client IP

x. Event count

y. For any allowed requests, notice that you can quickly check the details of the policy that allowed the access by clicking on the policy number in the `Policy ID` column



z. Exit HBase shell

```
hbase> exit
```

aa. Now let's check whether non-`sales` users can access the table

bb. Logout as `sales1` and log back in as `hr1`:

```
kdestroy
## logout as sales1
logout

## login as hr1 and authenticate
sudo su - hr1

kinit
## enter password: BadPass#1

klist
## Default principal: hr1@LAB.HORTONWORKS.NET
```

cc. Try to access the same dir as `hr1` and notice this user does not even see the table:

```
hbase> shell
hbase> describe 'sales'
hbase> list 'default'
```

```
hbase(main):001:0> describe 'sales'

ERROR: Unknown table sales!

Here is some help for this command:
Describe the named table. For example:
  hbase> describe 't1'
  hbase> describe 'ns1:t1'

Alternatively, you can use the abbreviated 'desc' for the same thing.
  hbase> desc 't1'
  hbase> desc 'ns1:t1'
```

dd. Try to create a table as `hr1`, and it fails with `org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient permissions`

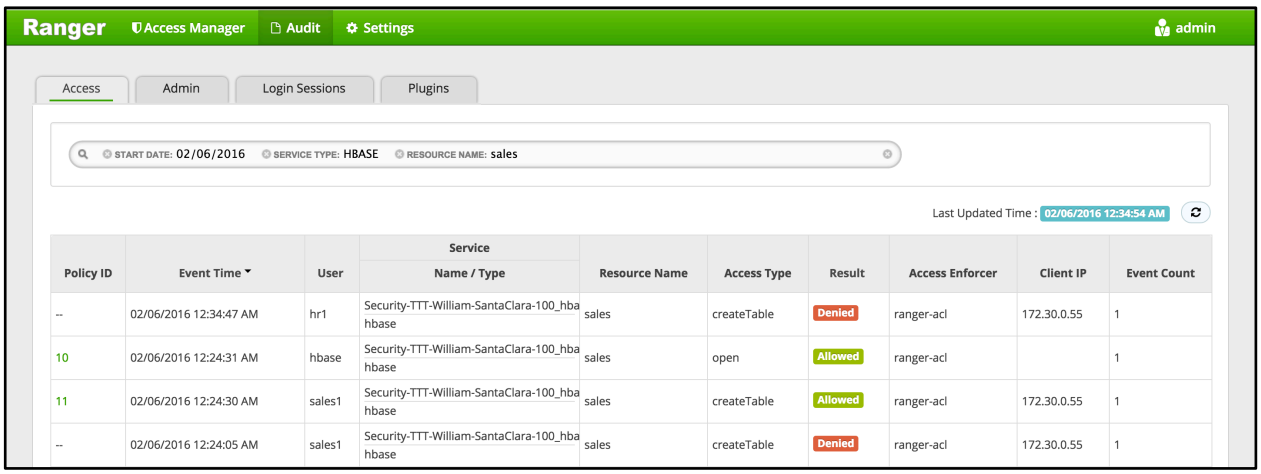
```
hbase> create 'sales', 'cf'
```

ee. In Ranger, click on `Audit` to open the `Audits` page and filter by:

i. Service Type: HBASE

ii. Resource Name: `sales`

ff. Here you can see the request by `sales1` was allowed but `hr1` was denied



gg. Exit HBase shell

```
hbase> exit
```

hh. Logout as `hr1`:

```
kdestroy
```

```
logout
```

ii. We have successfully created a table called `sales` in HBase, and setup authorization policies to ensure only `sales` users have access to the table. This shows how you can interact with Hadoop components on a Kerberized cluster and use Ranger to manage authorization policies and audits.

4. OPTIONAL: Use Sqoop to Import

a. If Sqoop is not already installed, install it via Ambari on same node where Mysql/Hive are installed:

i. `Admin > Stacks and Versions > Sqoop > Add service >` select node where Mysql/Hive are installed and accept all defaults

ii. You will be asked to enter admin principal/password:

```
hadoopadmin@LAB.HORTONWORKS.NET
BadPass#1
```

b. On the host running Mysql, change user to `root` and use a sample `.csv` and login to Mysql:

```
sudo su -
mv /labfiles/security/PII_data_small.csv /tmp
mysql -u root -pBadPass#1
```

c. At the `mysql>` prompt, run the commands below to:

i. Create a table in Mysql

ii. Give access to `sales1`

iii. Import the data from `.csv`

iv. Test that table was created

v. Create database `people`

vi. Use `people`

```
 create database people;
 use people;
 create table persons (people_id INT PRIMARY KEY, sex text, bdate DATE,
firstname text, lastname text, addresslineone text, addresslinetwo text, city
text, postalcode text, ssn text, id2 text, email text, id3 text);

 GRANT ALL PRIVILEGES ON people.* to 'sales1'@'%' IDENTIFIED BY 'BadPass#1';

 LOAD DATA LOCAL INfile '/tmp/PII_data_small.csv' REPLACE INTO TABLE persons
FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';

 select people_id, firstname, lastname, city from persons where
lastname='SMITH';
exit
```

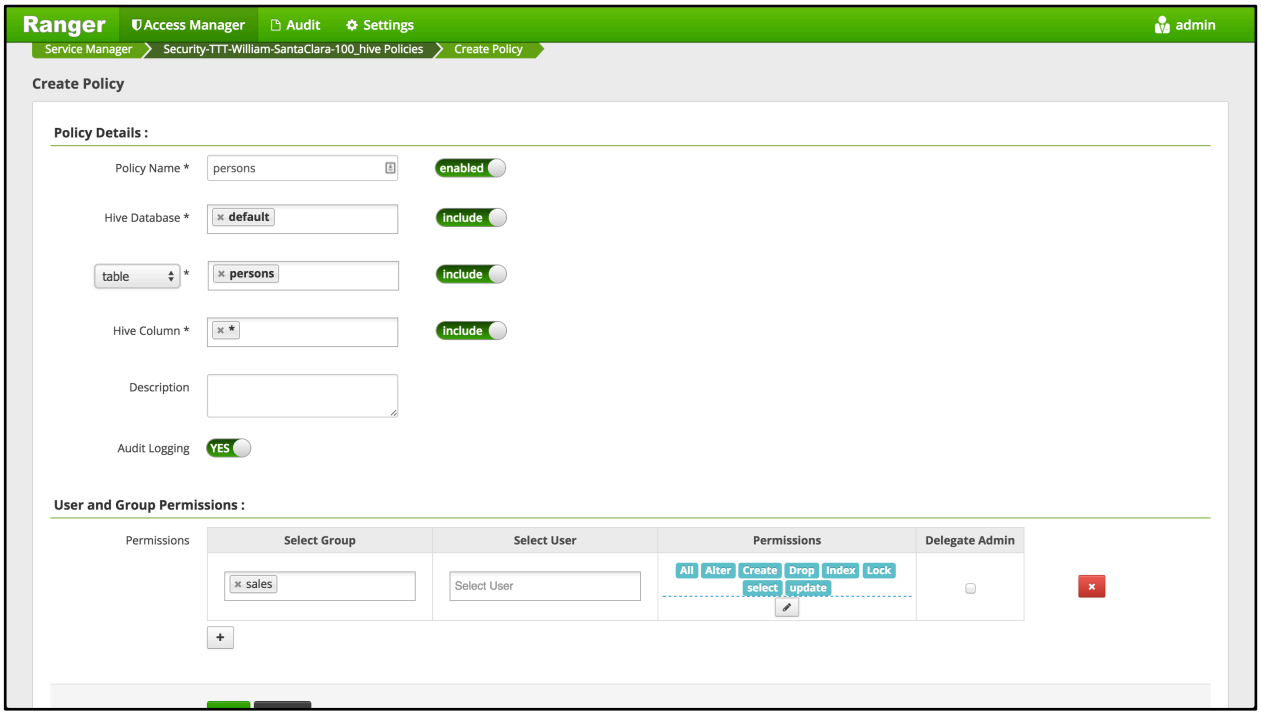
d. Log off as `root`:

```
logout
```

e. Create Ranger policy to allow sales group all permissions on persons table in Hive

i. `Access Manager > Hive > (cluster)_hive > Add new policy`

ii. Create new policy as below and click `Add`:



f. Create Ranger policy to allow sales group all permissions on /ranger/audit/kms dir in HDFS

   i. Access Manager > HDFS > (cluster)_hdfs > Add new policy

   ii. Create new policy as below and click Add: **TODO: add screenshot**

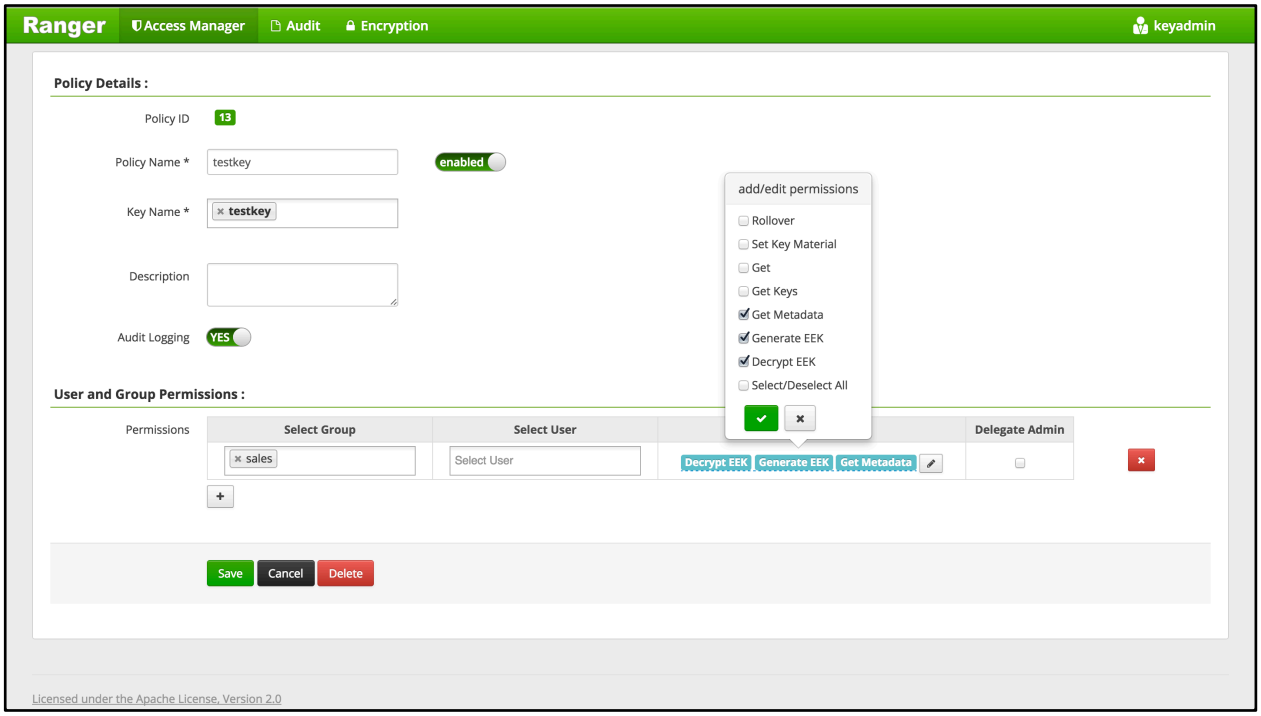
   iii. Log out of Ranger

g. Create a Ranger policy to allow `sales` group `Get Metadata GenerateEEK DecryptEEK` permissions on `testkey` (i.e. the key used to encrypt Hive warehouse directories)

   i. Login to Ranger http://RANGER_PUBLIC_IP:6080 with keyadmin/keyadmin

   ii. Access Manager > KMS > (cluster)_KMS > Add new policy

   iii. Create new policy as below and click Add:



iv. Log out of Ranger and re-login as `admin`/`admin`

h. Login as `sales1`

```
sudo su - sales1
```

i. As `sales1` user, `kinit` and run a Sqoop job to create `persons` table in Hive (in ORC format) and import data from MySQL. Below are the details of the arguments passed in:

   i. Table: MySQL table name
   ii. username: Mysql username
   iii. password: Mysql password
   iv. hcatalog-table: Hive table name
   v. create-hcatalog-table: hive table should be created first
   vi. driver: classname for Mysql driver
   vii. m: number of mappers

```
 kinit
## enter BadPass#1 as password

sqoop import --verbose --connect "jdbc:mysql://$(hostname -f)/people" --table
persons --username sales1 --password BadPass#1 --hcatalog-table persons --
hcatalog-storage-stanza "stored as orc" -m 1 --create-hcatalog-table  --
driver com.mysql.jdbc.Driver
```

j. This will start a MapReduce job to import the data from Mysql to Hive in ORC format

k. **NOTE:** if the MapReduce job fails with below, most likely you have not given sales group all the permissions needed on the EK used to encrypt Hive directories

```
java.lang.RuntimeException:
com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link
failure
```

l. Login to beeline

```
beeline -u "jdbc:hive2://localhost:10000/default;principal=hive/$(hostname -f)@LAB.HORTONWORKS.NET"
```

m. Query persons table in beeline

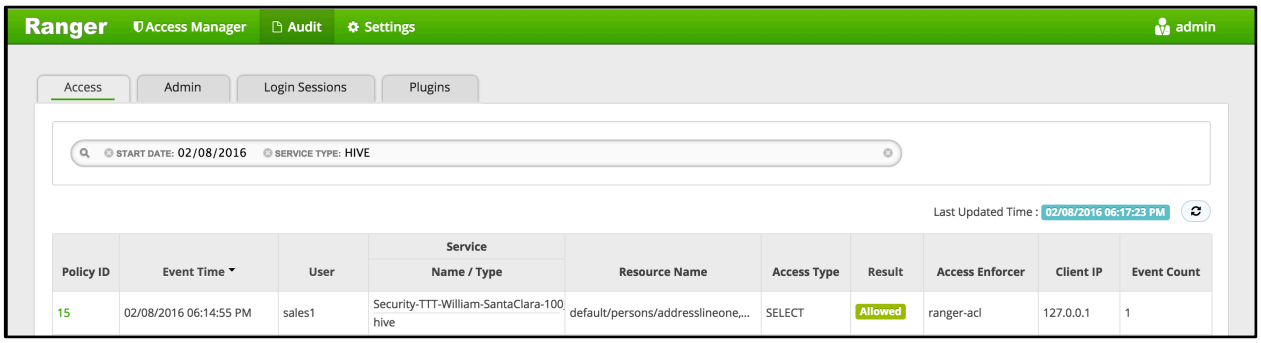
```
beeline> select * from persons;
```

n. Since the authorization policy is in place, the query should work

o. Ranger audit should show the request was allowed:

   i. Under Ranger > Audit > query for

   ii. Service type: HIVE



Drop Encrypted Hive Tables

p. From beeline, try to drop the `persons` table:

```
beeline> drop table persons;
```

q. You will get error similar to below:

```
message:Unable to drop default.persons because it is in an encryption zone and trash is enabled.  Use PURGE option to skip trash.
```

r. To drop a Hive table when Hives directories are located in `EncryptionZone`, you need to include `purge` as below:

```
beeline> drop table persons purge;
```

s. Destroy the ticket and logout as sales1

```
kdestroy
logout
```

RESULT:

This completes the lab. You have now interacted with Hadoop components in secured mode and used Ranger to manage authorization policies and audits.

Lab: Knox

About This Lab

Objective:	In this lab we will configure Apache Knox for AD authentication and make WebHDFS, Hive requests over Knox (after setting the appropriate Ranger authorization polices for access).
File locations:	N/A
Successful outcome:	Successfully access Knox with AD authentication and run Hive/WebHDFS requests over Knox
Before you begin	N/A
Related lesson:	N/A

Lab Steps

Perform the following steps:

1. Knox Configuration: Knox Configuration for AD Authentication

1. Run these steps on the node where Knox was installed earlier
2. To configure Knox for AD authentication, we need to enter AD related properties in the topology XML via Ambari. The problem is it requires us to enter LDAP bind password, but we do not want it exposed as plain text in the Ambari configs. We solve this by creating a keystore alias for the LDAP manager user (which you will later pass in to the topology via the `systemUsername` property)
3. Read in the password with the following command (this will prompt you for a password and save it in `knoxpass` environment variable); enter `BadPass#1`:

```
read -s -p "Password: " knoxpass
```

   i. This is a handy way to set an `env var` without storing the command in your history

   ii. Create password alias for Knox called `knoxLdapSystemPassword`

```
sudo -u knox /usr/hdp/current/knox-server/bin/knoxcli.sh create-alias
knoxLdapSystemPassword --cluster default --value ${knoxpass} -unset
knoxpass
```

4. Now lets configure Knox to use our AD for authentication. Replace below content in `Ambari > Knox > Config > Advanced topology`:

   i. How to tell what configs were changed from defaults:

      1. Default configs remain indented below
      2. Configurations that were added/modified are not indented

```
<topology>

    <gateway>

        <provider>
            <role>authentication</role>
            <name>ShiroProvider</name>
```

```
                    <enabled>true</enabled>
                    <param>
                        <name>sessionTimeout</name>
                        <value>30</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  -->
<param>
    <name>main.ldapRealm.contextFactory</name>
    <value>$ldapContextFactory</value>
</param>

<!-- 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.contextFactory.authenticationMechanism</name>
                        <value>simple</value>
                    </param>
                    <param>
                        <name>urls./**</name>
                      <value>authcBasic</value>
                    </param>
```

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


                </provider>

                <provider>
                    <role>identity-assertion</role>
                    <name>Default</name>
                    <enabled>true</enabled>
                </provider>

                <provider>
                    <role>authorization</role>
                    <name>XASecurePDPKnox</name>
                    <enabled>true</enabled>
                </provider>

            </gateway>

            <service>
                <role>NAMENODE</role>
                <url>hdfs://{{namenode_host}}:{{namenode_rpc_port}}</url>
            </service>

           <service>
                <role>JOBTRACKER</role>
                <url>rpc://{{rm_host}}:{{jt_rpc_port}}</url>
            </service>
```

```
            <service>
                <role>WEBHDFS</role>

<url>http://{{namenode_host}}:{{namenode_http_port}}/webhdfs</url>
            </service>

            <service>
                <role>WEBHCAT</role>

<url>http://{{webhcat_server_host}}:{{templeton_port}}/templeton</url>
            </service>

            <service>
                <role>OOZIE</role>

<url>http://{{oozie_server_host}}:{{oozie_server_port}}/oozie</url>
            </service>

            <service>
                <role>WEBHBASE</role>
                <url>http://{{hbase_master_host}}:{{hbase_master_port}}</url>
            </service>

            <service>
                <role>HIVE</role>

<url>http://{{hive_server_host}}:{{hive_http_port}}/{{hive_http_path}}</url>
            </service>

            <service>
                <role>RESOURCEMANAGER</role>
                <url>http://{{rm_host}}:{{rm_port}}/ws</url>
            </service>
        </topology>
```

5. Then restart Knox via Ambari

2. HDFS Configuration for Knox

a. 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)

b. Now restart HDFS

c. 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"
```

3. Ranger Configuration for WebHDFS over Knox

a. Setup a Knox policy for sales group for WEBHDFS by:

b. Login to `Ranger > Access Manager > KNOX > click the cluster name link > Add new policy`:

   i. Policy name: webhdfs

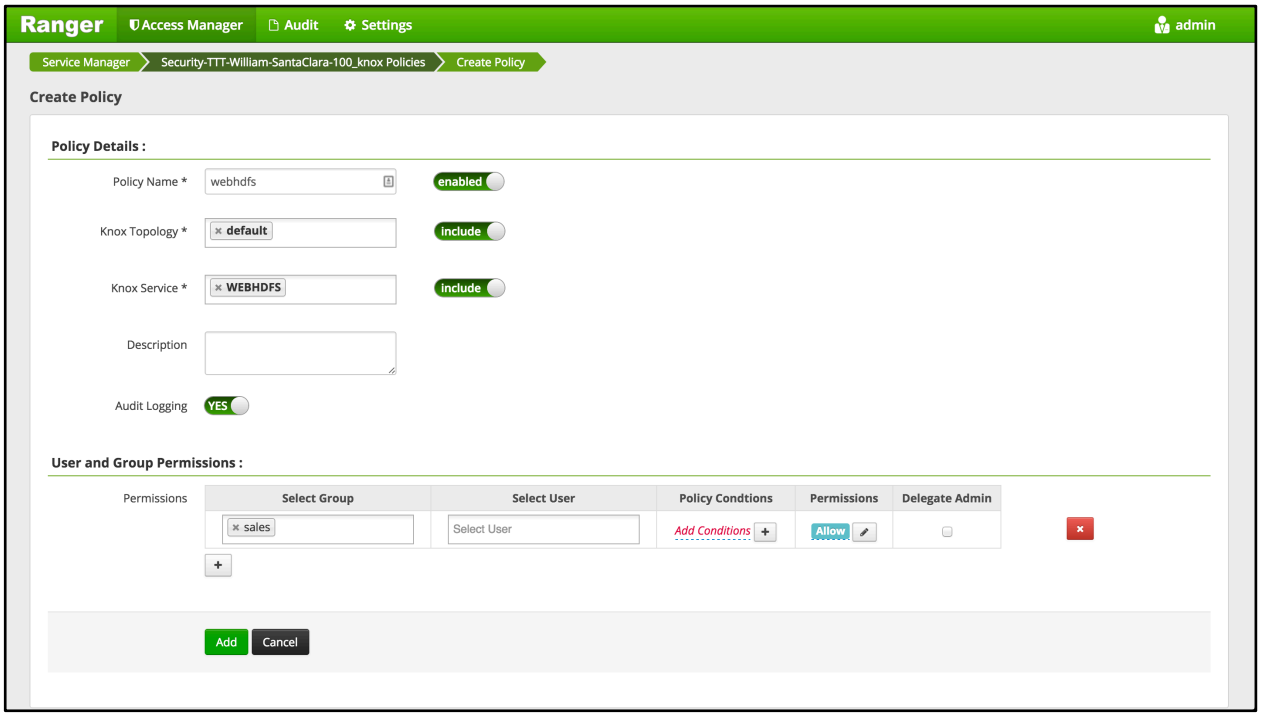
   ii. Topology name: default

   iii. Service name: WEBHDFS

   iv. Group permissions: sales

   v. Permission: check Allow

   vi. Add



4. WebHDFS over Knox exercises

a. Now we can post some requests to WebHDFS over Knox to check it's working. We will use `curl` with following arguments:

- `-i` (aka `-include`): used to output HTTP response header information. This will be important when the content of the HTTP Location header is required for subsequent requests.
- `-k` (aka `-insecure`) is used to avoid any issues resulting from the use of demonstration SSL certificates.
- `-u` (aka `-user`) is used to provide the credentials to be used when the client is challenged by the gateway.

NOTE: Most of the samples do not use the cookie features of `cURL` for the sake of simplicity. Therefore, we will pass in user credentials with each `curl` request to authenticate.

b. From the host where Knox is running, send the below curl request to 8443 port where Knox is running to run `ls` command on the `/` directory in HDFS:

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

c. This should return a JSON object containing list of directories/files located in the root directory and their attributes

d. To avoid passing password on command prompt you can pass in just the username (to avoid having the password captured in the shell history). In this case, you will be prompted for the password

```
curl -ik -u sales1
https://localhost:8443/gateway/default/webhdfs/v1/?op=LISTSTATUS

## enter BadPass#1
```

e. For the remaining examples below, for simplicity, we are passing in the password on the command line, but feel free to remove the password and enter it in manually when prompted

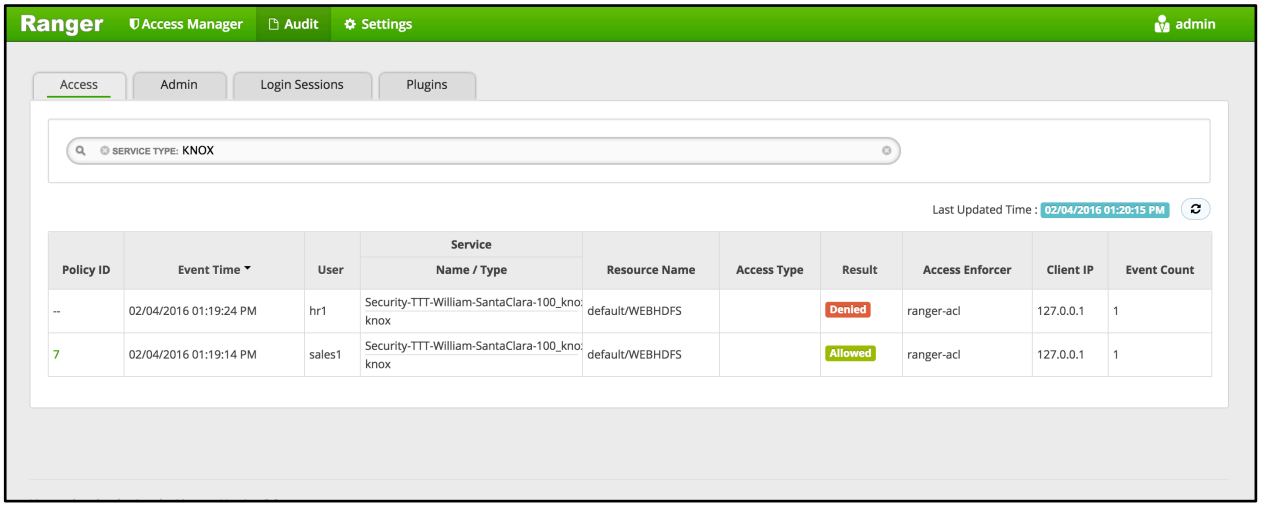
f. Try the same request as `hr1` and notice it fails with `Error 403 Forbidden`:

```
 curl -ik -u hr1:BadPass#1
https://localhost:8443/gateway/default/webhdfs/v1/?op=LISTSTATUS
```

- This is expected since in the policy above, we only allowed `sales` group to access WebHDFS over Knox.
- Notice that to make the requests over Knox, a Kerberos ticket is not needed; the user authenticates by passing in AD/LDAP credentials

g. Check in Ranger Audits to confirm the requests were audited:

   i. `Ranger > Audit > Service type: KNOX`



h. Other things to access WebHDFS with Knox:

i. Use a cookie to make a request without passing in credentials

1. When you ran the previous curl request it would have listed HTTP headers as part of output. One of the headers will be `Set Cookie`
2. e.g. `Set-Cookie: JSESSIONID=xxxxxxxxxxxxxxx;Path=/gateway/default;Secure;HttpOnly`

ii. You can pass in the value from your setup and make the request without passing in credentials:

```
curl -ik --cookie "JSESSIONID=xxxxxxxxxxxxxxx;Path=/gateway/default;Secure;HttpOnly" -X GET https://localhost:8443/gateway/default/webhdfs/v1/?op=LISTSTATUS
```

i. Open a file via WebHDFS

i. List files under /tmp and pick a text file to open:

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

ii. You can run below command to create a test file into /tmp

```
echo "Test file" > /tmp/testfile.txt
sudo -u sales1 kinit
## enter BadPass#1
sudo -u sales1 hdfs dfs -put /tmp/testfile.txt /tmp
sudo -u sales1 kdestroy
```

iii. Open this file via WebHDFS

```
curl -ik -u sales1:BadPass#1 -X GET https://localhost:8443/gateway/default/webhdfs/v1/tmp/testfile.txt?op=OPEN
```

iv. Look at value of Location header. This will contain a long URL

```
[centos@ip-172-30-0-241 ~]$ curl -ik -u sales1:BadPass#1 -X GET https://localhost:8443/gateway/default/webhdfs/v1/tmp/testfile.txt?op=OPEN
HTTP/1.1 307 Temporary Redirect
Set-Cookie: JSESSIONID=oe60r33jwtr11qgpcqa6un58x;Path=/gateway/default;Secure;HttpOnly
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Cache-Control: no-cache
Expires: Thu, 11 Feb 2016 08:57:09 GMT
Date: Thu, 11 Feb 2016 08:57:09 GMT
Pragma: no-cache
Expires: Thu, 11 Feb 2016 08:57:09 GMT
Date: Thu, 11 Feb 2016 08:57:09 GMT
Pragma: no-cache
Location: https://localhost:8443/gateway/default/webhdfs/data/v1/webhdfs/v1/tmp/testfile.txt?_=AAAACAAAABAAAAEwfRbtpNWhoIj27sixGIUJxGYNLZvdT8ZE1tmMQdSukmwwr
7tV-XOK_iqHj98r0JGfvB6i3SCm_keL86FZ0c-h8YdssoUvHYtJfZYt94FtWH3z8-7A_eBO_zMZcGFZItzZg0_44_YC0SnqGhx0QHy7BQy3bD7SIyxOq0wPY1X4XZUGNWjAzpD8Ha3Ara4hVD4X-ceFCQxSp
EvtigjDt71pFsLLjfFh4tTiVkVslmwYTxXzB4JL51mPWzIfyPKojiOBbUE-RidWHU5vh0WTzVJpKjFKQjTJTYlBClHZ1Q0yN8z7uR56tLgehWmlXBYeEZcoepF1BQbMBwMvBqv_X80gim_2mf8CKMNrSbkEH
Xb3pNdO_cSNCButtxqXNhdYLs_cCRsMFYk8ahUEUBitQkT3PuOsB6_sDCtfDL2ZdiwXCzLrOVKdut-C2A
Server: Jetty(6.1.26.hwx)
Content-Type: application/octet-stream
Content-Length: 0
```

v. Access contents of file /tmp/testfile.txt by passing the value from the above Location header

```
curl -ik -u sales1:BadPass#1 -X GET
'{https://localhost:8443/gateway/default/webhdfs/data/v1/webhdfs/v1/tmp/testf
ile.txt?_=AAAACAAAABAAAAEwvyZNDLGGNwahMYZKvaHHaxymBy1YEoe4UCQOqLC7o8fg0z6845k
TvMQN_uULGUYGoINYhH5qafY_HjozUseNfkxyrEo313-Fwq8ISt6MKEvLqas1VEwC07-
ihmK65Uac8wT-
Cmj2BDab5b7EZx9QXv29BONUuzStCGzBYCqD_OIgesHLkhAM6VNOlkgpumr6EBTuTnPTt2mYN6YqB
STX6cc6OhX73WWE6atHy-lv7aSCJ2I98z2btp8XLWWHQDmwKWSmEvtQW6Aj-
JGInJQzoDAMnU2eNosdcXaiYH856zC16IfEucdb7SA_mqAymZuhm8lUCvL25hd-
bd8p6mn1AZlOn92VySGp2TaaVYGwX-
6L9by73bC6sIdi9iKPl3Iv13GEQZEKsTm1a96Bh6ilScmrctk3zmY4vBYp2SjHG9JRJvQgr2XzgA}
'
```

j. Use groovy scripts to access WebHDFS

i. Edit the groovy script to set:

- gateway = `https://localhost:8443/gateway/default`
- username = `sales1`
- password = `BadPass#1`

```
sudo vi /usr/hdp/current/knox-server/samples/ExampleWebHdfsLs.groovy
```

ii. Run the script

```
sudo java -jar /usr/hdp/current/knox-server/bin/shell.jar
/usr/hdp/current/knox-server/samples/ExampleWebHdfsLs.groovy
```

iii. Notice output show list of directories in HDFS

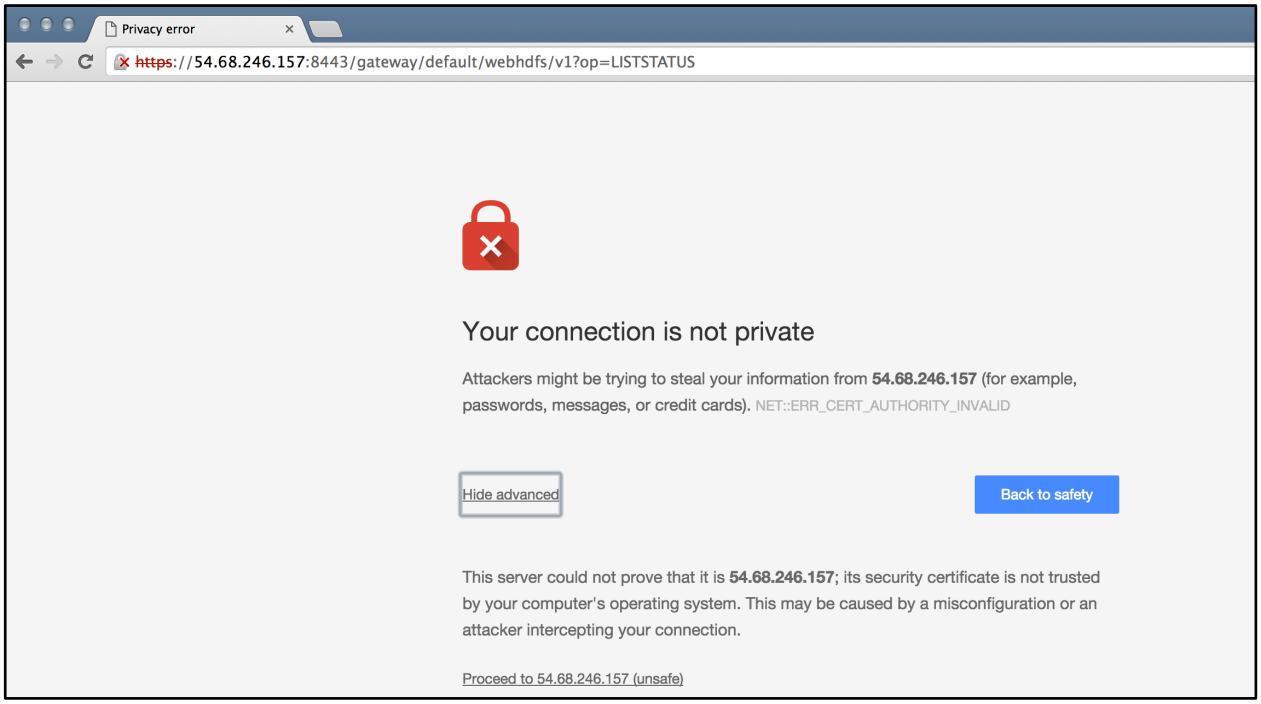
```
[app-logs, apps, ats, hdp, mapred, mr-history, ranger, tmp, user,
zone_encr]
```

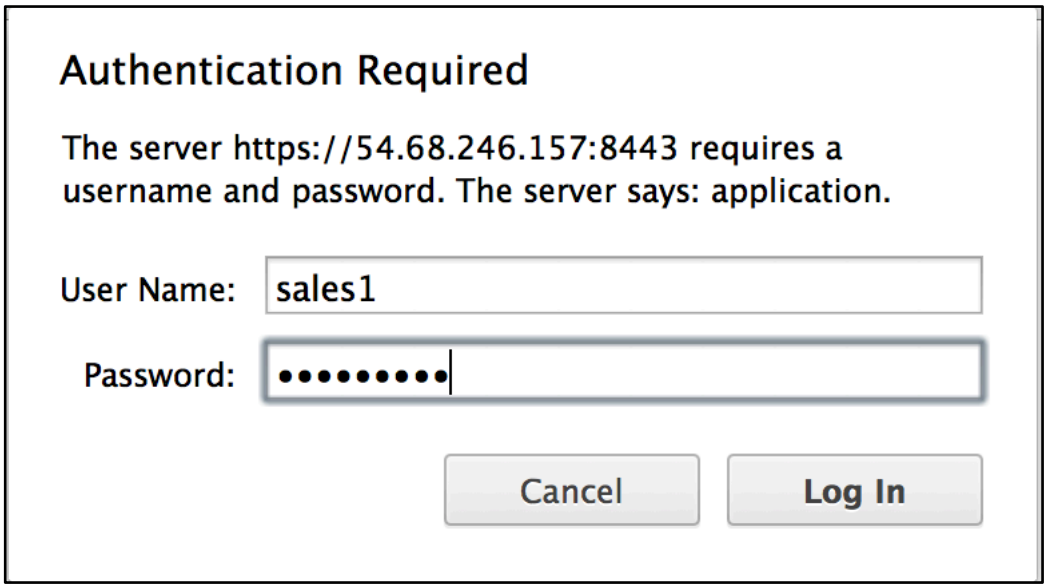
k. Access via browser

i. Take the same URL we have been hitting via `curl` and replace `localhost` with public IP of Knox node (remember to use https!) e.g. `https://PUBLIC_IP_OF_KNOX_HOST:8443/gateway/default/webhdfs/v1?op=LISTSTATUS`

ii. Open the URL via browser

iii. Login as `sales1`/`BadPass`#1





l. We have shown how you can use Knox to avoid the end user from having to know about internal details of cluster…

- Whether its Kerberized or not, and
- Regardless of the cluster topology (e.g. what node WebHDFS was running)

5. Hive over Knox

a. Configure Hive for Knox

1. In Ambari, under `Hive > Configs >` set the below and restart Hive component

```
hive.server2.transport.mode = http
```

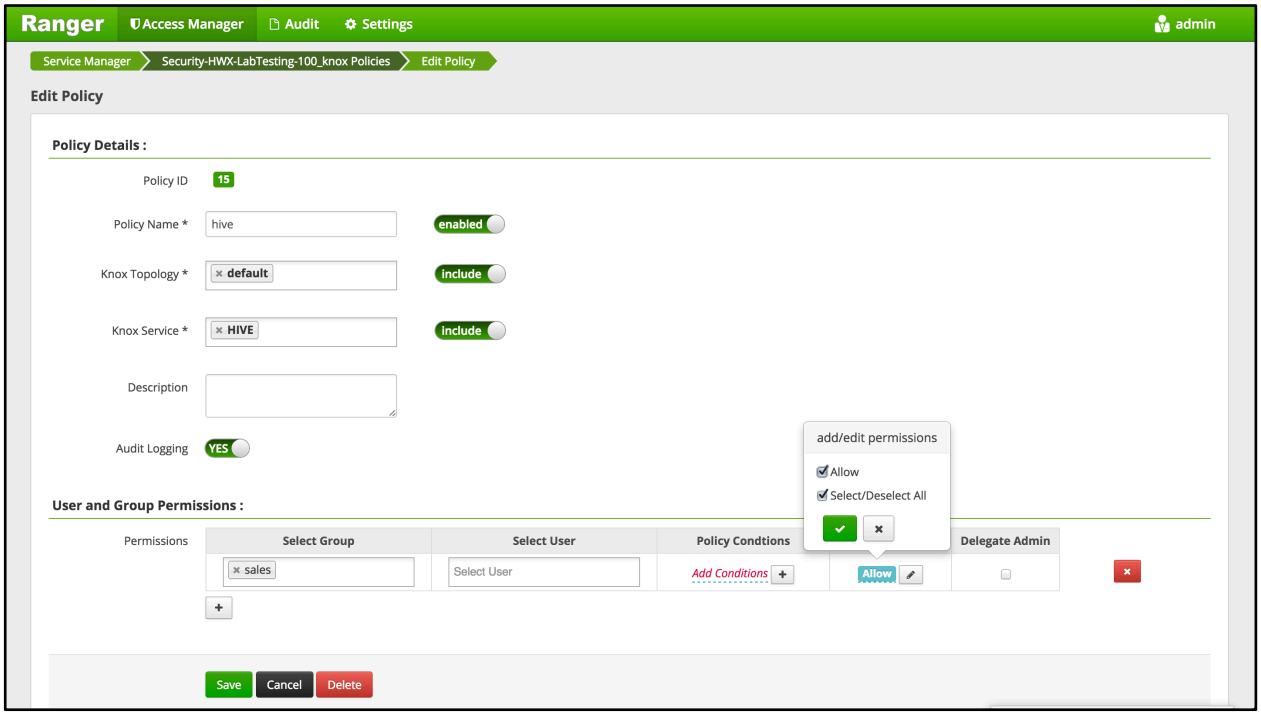
2. Give users access to `jks` file
   a. This is only for testing since we are using a self-signed cert
   b. This only exposes the truststore, not the keys

```
 sudo chmod o+x /usr/hdp/current/knox-server /usr/hdp/current/knox-
server/data /usr/hdp/current/knox-server/data/security /usr/hdp/current/knox-
server/data/security/keystores

 sudo chmod o+r /usr/hdp/current/knox-
server/data/security/keystores/gateway.jks
```

6. Ranger Configuration for Hive over Knox:

Note: Setup a Knox policy for sales group for HIVE by:

a. Login to `Ranger > Access Manager > KNOX > click the cluster name link > Add new policy`
   i. Policy name: `hive`
   ii. Topology name: `default`
   iii. Service name: `HIVE`
   iv. Group permissions: `sales`
   v. Permission: check `Allow`
   vi. `Add`



7. Use Hive for Knox

a. By default Knox will use a self-signed (untrusted) certificate. To trust the certificate:

b. First on Knox node, create the `/tmp/knox.crt` certificate:

```
knoxserver=$(hostname -f)

openssl s_client -connect ${knoxserver}:8443 <<<'' | openssl x509 -out
/tmp/knox.crt
```

c. On node where beeline will be run from (e.g. Hive node):

   i. Copy over the `/tmp/knox.crt`

   ii. Trust the certificate

```
sudo keytool -import -trustcacerts -keystore /etc/pki/java/cacerts -
storepass changeit -noprompt -alias knox -file /tmp/knox.crt
```

d. Now connect via beeline:

```
beeline -u
"jdbc:hive2://KnoxserverInternalHostName:8443/;ssl=true;transportMode=http;ht
tpPath=gateway/default/hive" -n sales1 -p BadPass#1
```

e. Notice that in the JDBC connect string for connecting to an secured Hive running in http transport mode:

   i. Port changes to Knox's port 8443

   ii. Traffic between client and Knox is over HTTPS

   iii. A Kerberos principal not longer needs to be passed in

f. Test these users:

   i. `sales1/BadPass#1` should work

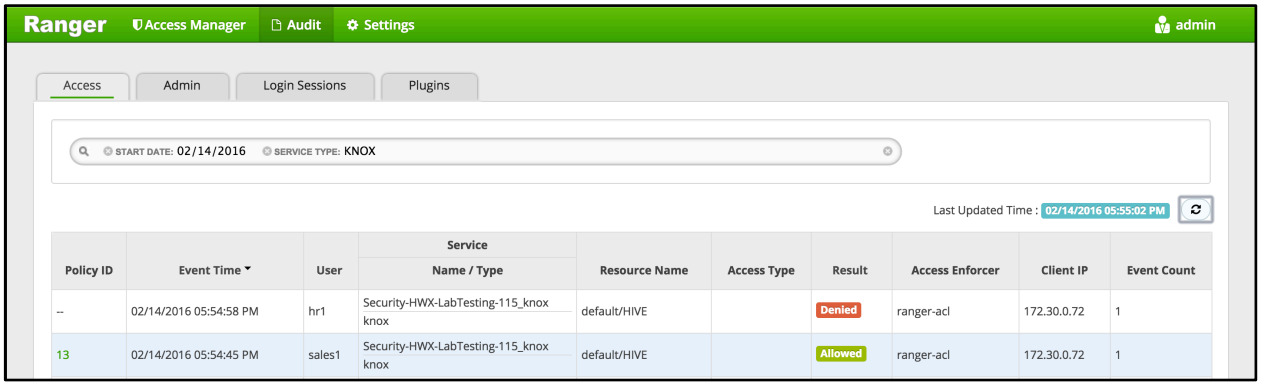
   ii. `hr1/BadPass#1` should not work

iii. Will fail with:

```
 Could not create http connection to
jdbc:hive2://hostname:8443/;ssl=true;transportMode=http;httpPath=gateway/defa
ult/hive. HTTP Response code: 403 (state=08S01,code=0)
```

g. Check in Ranger Audits to confirm the requests were audited:

i. `Ranger > Audit > Service type: KNOX`



h. This shows how Knox helps end users access Hive securely over HTTPS using Ranger to set authorization policies and for audits

RESULT:

Should be able to successfully access Knox with AD authentication and run Hive/WebHDFS requests over Knox

Lab: Other Security Features for Ambari

About This Lab

Objective:	In this lab we will setup Ambari views on Kerberized cluster.
File locations:	N/A
Successful outcome:	Successfully setup Ambari views on a Kerborized cluster
Before you begin	N/A
Related lesson:	N/A

Overview

For reference documentation, see

`http://docs.hortonworks.com/HDPDocuments/Ambari-2.2.0.0/bk_ambari_views_guide/content/ch_using_ambari_views.html`

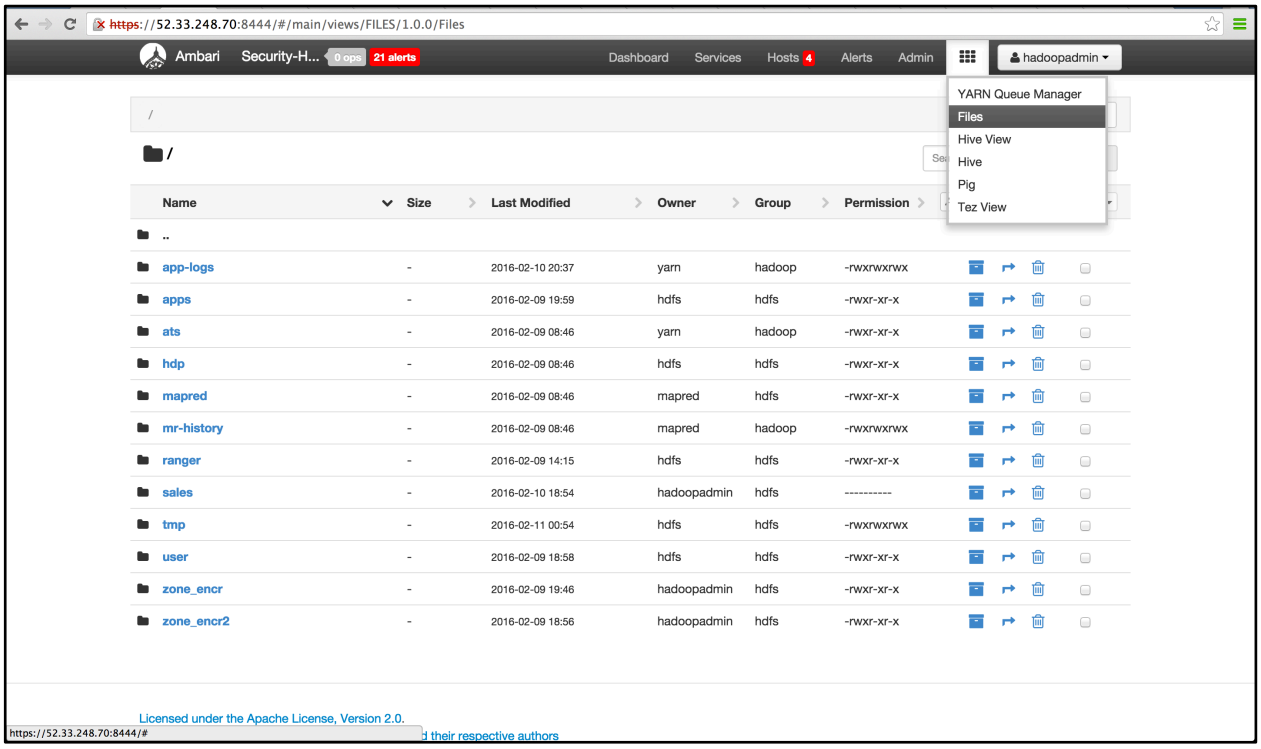
Lab Steps

Perform the following steps:

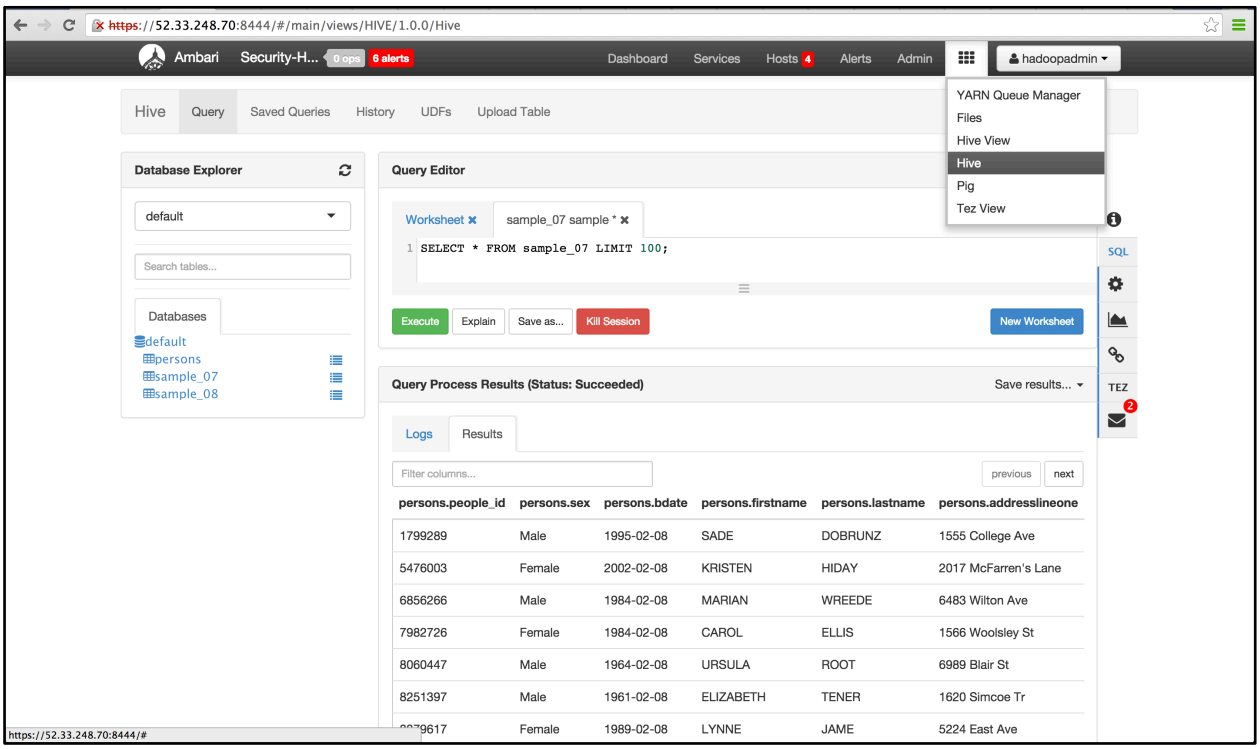
1. Change transport mode back to binary in Hive settings:
   a. In Ambari, under `Hive > Configs >` set the below and restart Hive component.
      - hive.server2.transport.mode = binary
2. You may also need to change proxy user settings to be less restrictive
3. Automation to install views (does not support Ambari running on HTTPS)

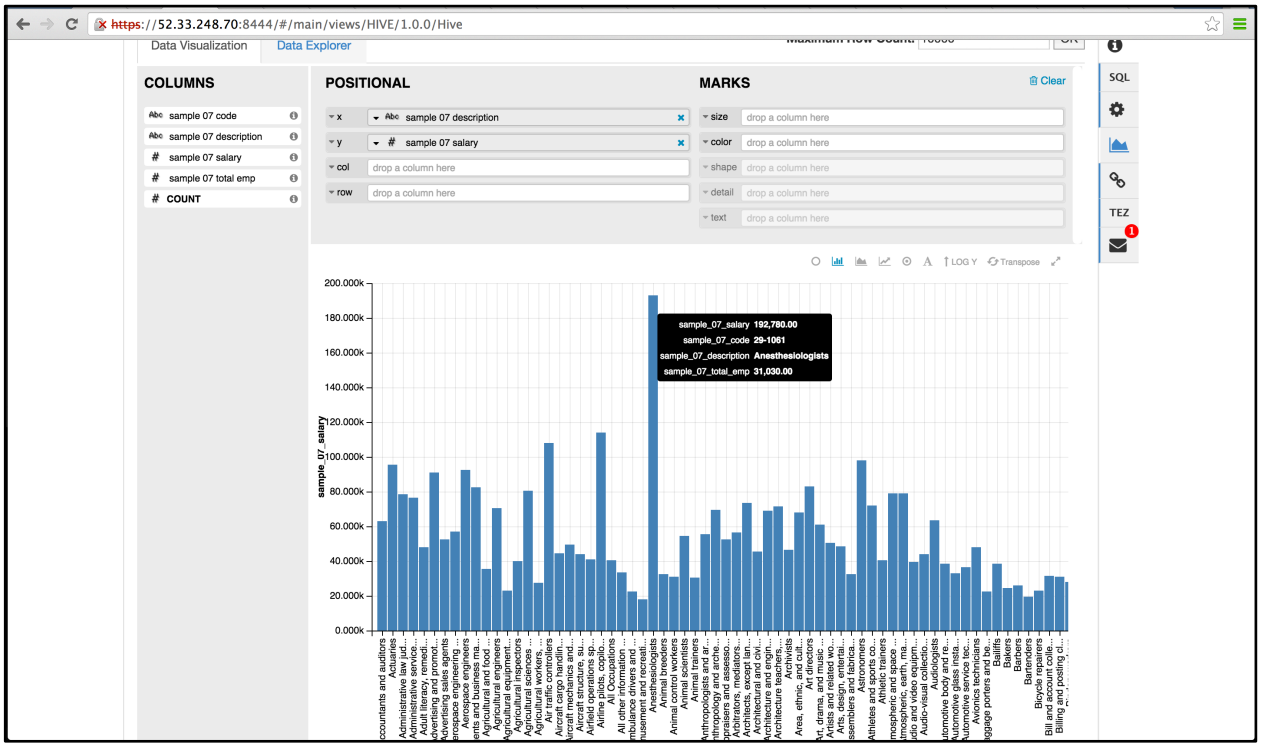
```
sudo su

cd /tmp/Ops_Labs/build/security/ambari-bootstrap-master/extras/

export ambari_user=hadoopadmin
export ambari_pass=BadPass#1
export ambari_port=8444
export ambari_protocol=https

source ambari_functions.sh
./ambari-views/create-views.sh
```

4. Restart HDFS and YARN via Ambari
5. Access the views:
   a. Files view

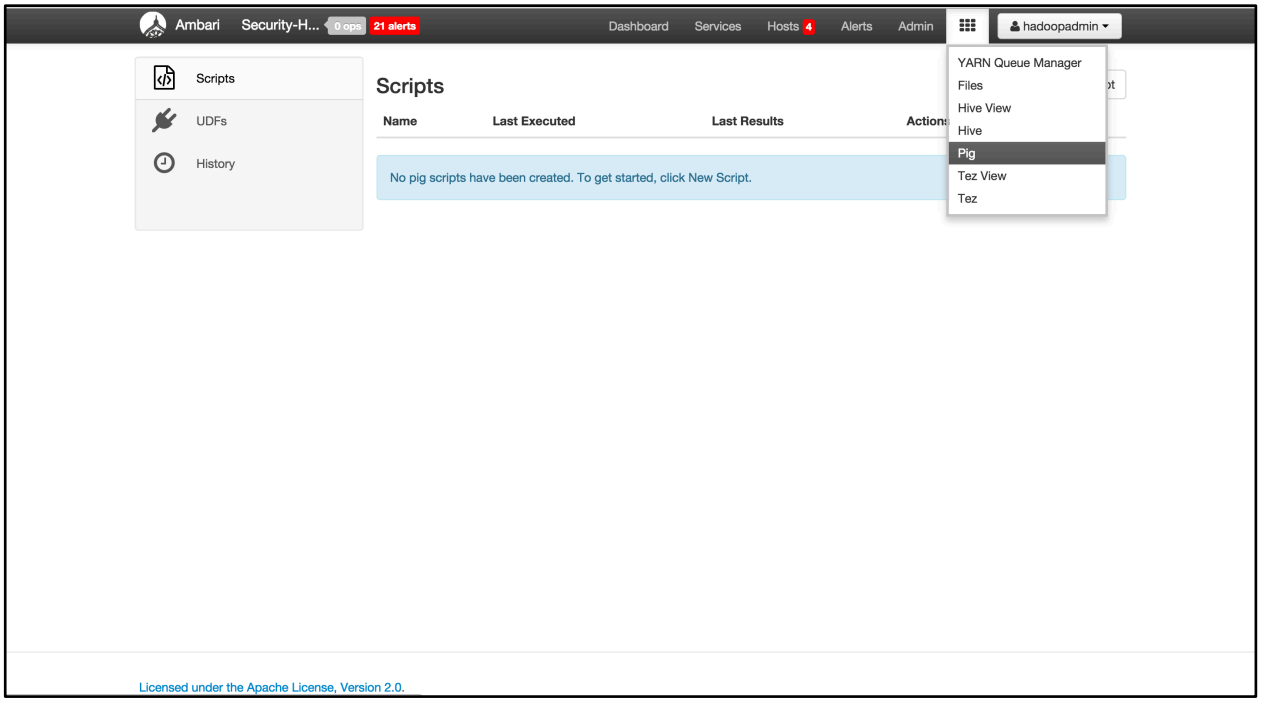


b. Hive view

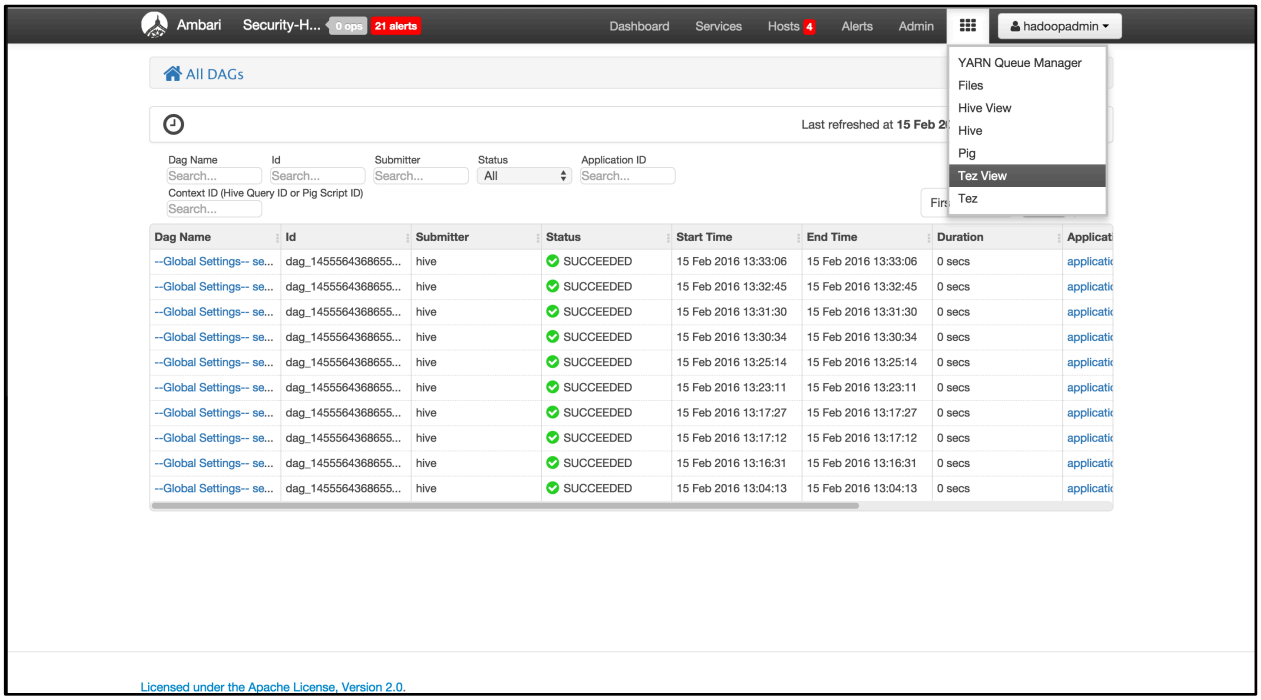


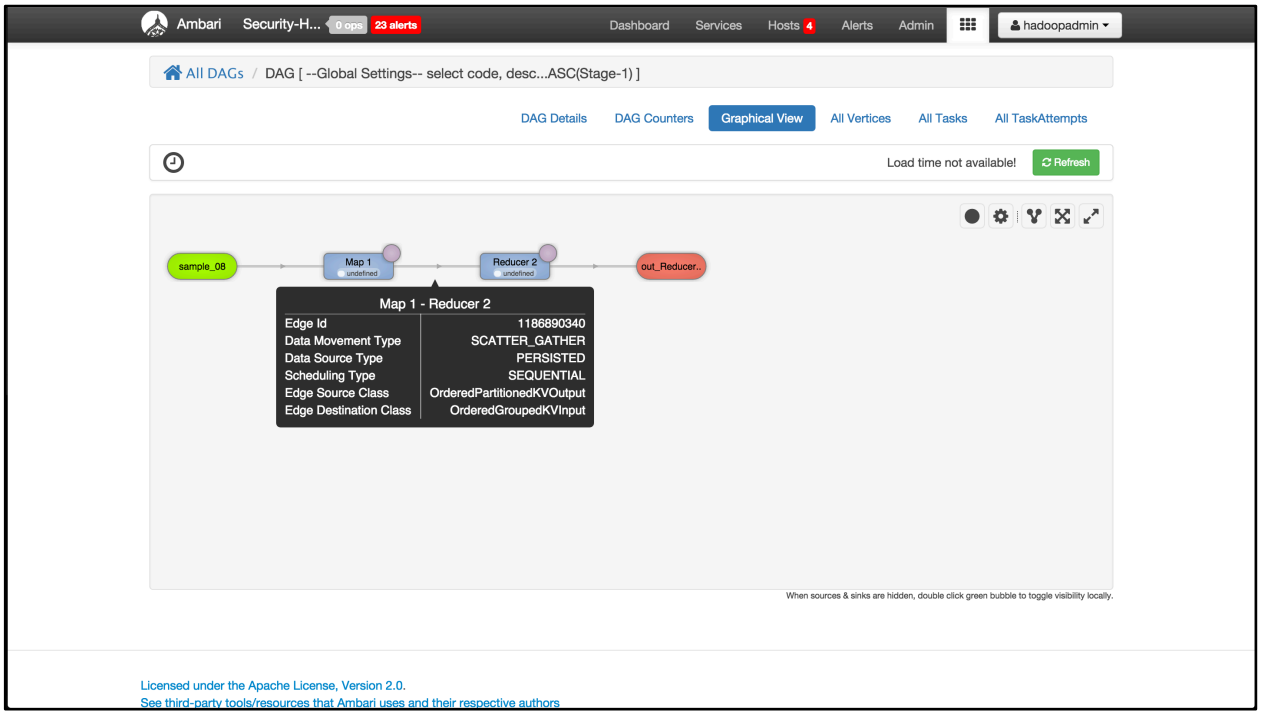


c. Pig view



d. Tez view





Enable Users to Log Into Ambari Views

In Ambari follow steps below:

1. On top right of page, click `Manage Ambari`

a. Under Views, Navigate to `Hive > Hive >` Under `Permissions` grant `sales1` access to Hive view

b. Similarly, you can give sales access to Files view

Result

At this point, you should be able to login to Ambari as `sales1` user and navigate to the views. Ambari views is now setup on a kerborized cluster.

Appendix – Install SolrCloud

About This Lab

Objective: Install SolrCLoud

File locations: N/A

Successful outcome: Successfully installed Solr

Before you begin

Related lesson: Lab 5

Lab Steps

Perform the following steps:

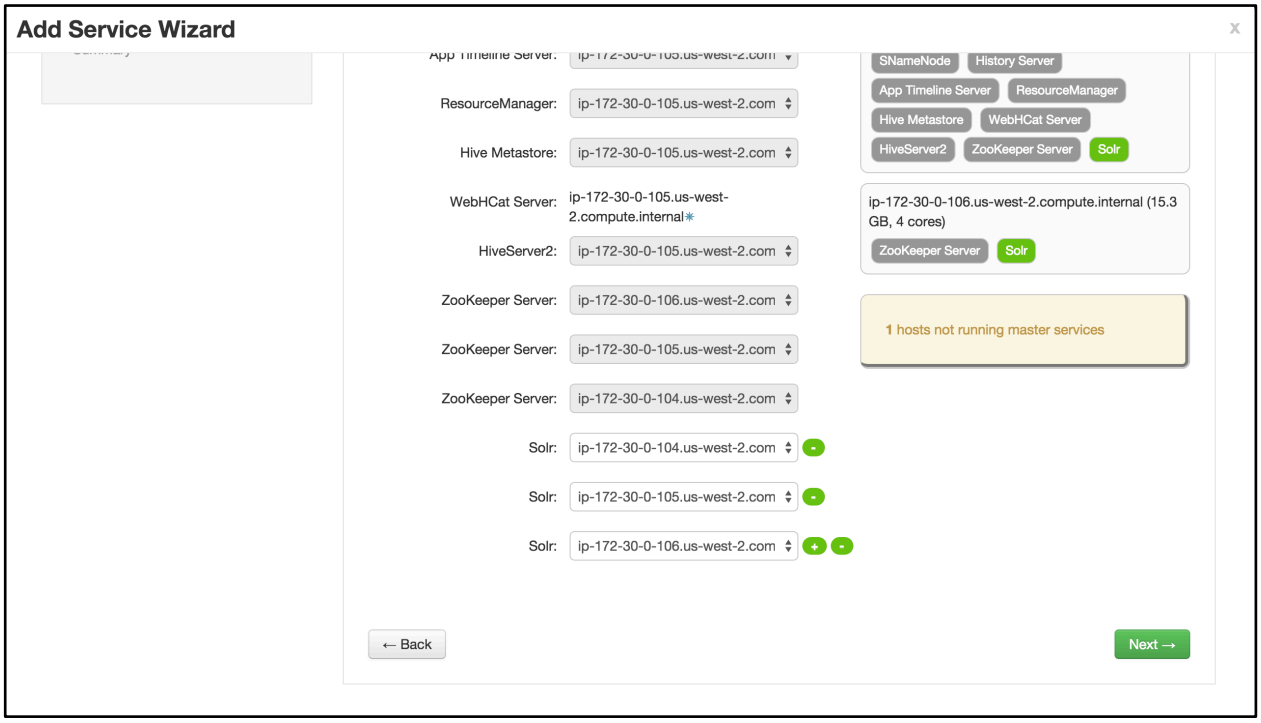
1. **Option 1: Install Solr Manually**
   a. Manually install Solr on each node where Zookeeper is running:

```
export JAVA_HOME=/usr/java/default

sudo yum -y install lucidworks-hdpsearch
```

2. **Option 2: Use Ambari Service for Solr**
   a. Install Ambari service for Solr:

```
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

sudo ambari-server restart
```

   b. Login to Ambari as `hadoopadmin` and wait for all the services to turn green
   c. Install Solr by starting the `Add service` wizard (using `Actions` dropdown) and choosing Solr. Pick the defaults in the wizard except:
      i. On the screen where you choose where to put Solr, use the + button next to Solr to add Solr to each host that runs a Zookeeper Server



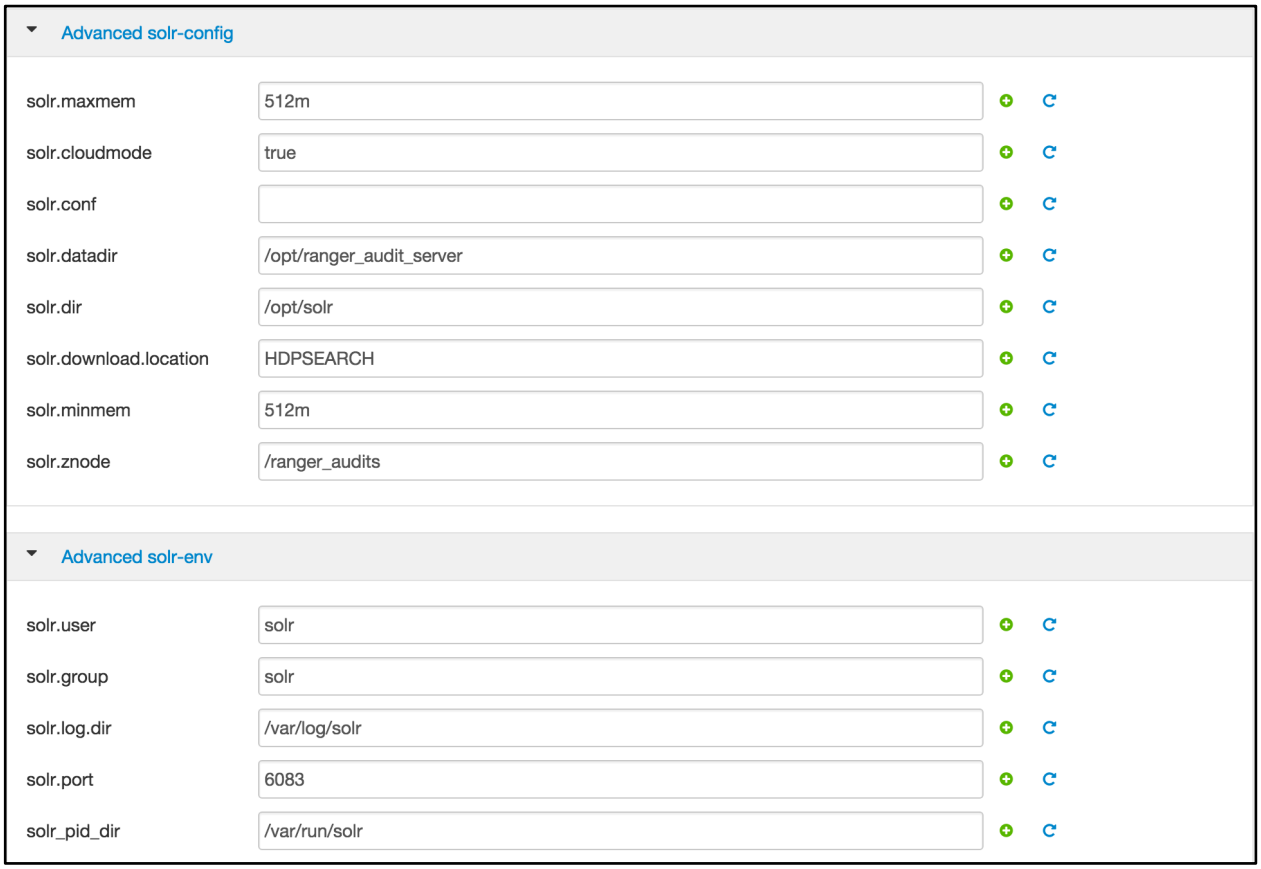
ii. On the screen to Customize the Solr service:

1. Under `Advanced solr-config`:

   Set `solr.datadir` to `/opt/ranger_audit_server`
   Set `solr.download.location` to `HDPSEARCH`
   Set `solr.znode` to `/ranger_audits`

2. Under `Advanced solr-env`:

   Set `solr.port` to `6083`



d. Under Configure Identities page, you will have to enter your AD admin credentials:

   i. Admin principal: `hadoopadmin@LAB.HORTONWORKS.NET`

   ii. Admin password: `BadPass#1`

e. Then go through the rest of the install wizard by clicking `Next` to complete installation of Solr

f. **OPTIONAL:** In case of failure, run below from Ambari node to delete the service so you can try again:

```
 export SERVICE=SOLR
 export AMBARI_HOST=localhost
 export PASSWORD=BadPass#1

 output=`curl -u hadoopadmin:$PASSWORD -i -H 'X-Requested-By:
ambari'  http://localhost:8080/api/v1/clusters`

 CLUSTER=`echo $output | sed -n 's/.*"cluster_name" :
"\([^\"]*\)".*/\1/p'`

## attempt to unregister the service
curl -u admin:$PASSWORD -i -H 'X-Requested-By: ambari' -X DELETE
http://$AMBARI_HOST:8080/api/v1/clusters/$CLUSTER/services/$SERVI
CE
```

```
## in case the unregister service resulted in 500 error, run the
below first and then retry the unregister API

 curl -u admin:$PASSWORD -i -H 'X-Requested-By: ambari' -X PUT -d
'{"RequestInfo": {"context" :"Stop $SERVICE via REST"}, "Body":
{"ServiceInfo": {"state": "INSTALLED"}}}'
http://$AMBARI_HOST:8080/api/v1/clusters/$CLUSTER/services/$SERVI
CE

 sudo service ambari-server restart

## restart agents on all nodes

 sudo service ambari-server restart
```

Classes Available Worldwide Through Our Partners

Study Options Worldwide

In combination with our partner providers, classes are often available in numerous locations across the world.

Private On-site Training

Hortonworks training in-house covers all of our basic coursework, and provides a more intimate setting for 6 or more students.
Contact us for more details

Learn from the company focused solely on Hadoop.

What Makes Us Different?

1. Our courses are designed by the **leaders and committers** of Hadoop
2. We provide an **immersive** experience in **real-world** scenarios
3. We prepare you to **be an expert** with highly valued, **fresh skills**
4. Our courses are available **near you**, or accessible **online**

Hortonworks University courses are designed by the leaders and committers of Apache Hadoop. We provide immersive, real-world experience in scenario-based training. Courses offer unmatched depth and expertise available in both the classroom or online from anywhere in the world. We prepare you to be an expert with highly valued skills and for Certification.