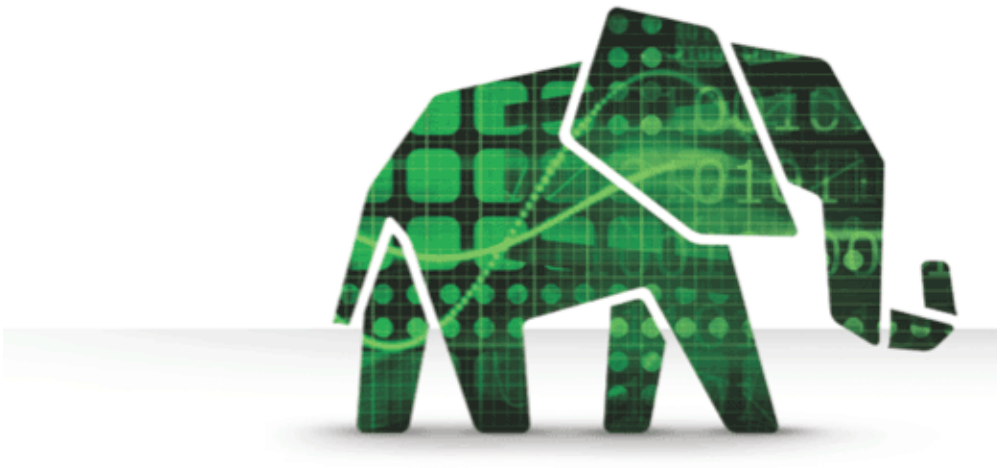
Hortonworks Architecting the Future of Big Data

# Hortonworks Technical Preview for Apache Knox Gateway

Released: 11/22/2013




Architecting the Future of Big Data



Welcome to Hortonworks Inc, technical preview for Apache Knox Gateway. The Technical Preview provides early access to upcoming features in the Hortonworks product, letting you test and review during the development process. These features are considered under development.

Although your feedback is greatly appreciated these features are not intended for use in your production systems and not considered Supported by Hortonworks.

Have fun and please send feedback to us on the Community forums:

http://hortonworks.com/community/forums/

## Introduction

Apache Knox Gateway (Apache Knox) is the Web/REST API Gateway solution for Hadoop. It provides a single access point to access all of Hadoop resources over REST. It also enables the integration of enterprise identity management solutions and numerous perimeter security features for REST/HTTP access to Hadoop. As Hadoop becomes a more critical data processing platform in enterprise there will continue to be increased focus on security.

The typical Hadoop cluster has many ports open – for example e.g. NN, JT, DN, WebHDFS, and HiveServer2 each create a port. The sheer number of open ports presents a great attack vector for Hadoop.

One of the basic security principles for protecting systems is to minimize the attack surface area. In the context of Hadoop this means utilizing network security facilities to minimize the number of ports exposed to the corporate network or the outside world. In such an environment, a firewall may be configured to accept only connections destined for the Apache Knox Gateway over HTTP. These types of measures dramatically reduce the surface area of a Hadoop cluster that is available for attack.

When rewriting the URLs returned to REST clients, Apache Knox uses encryption to protect the sensitive topology information of the Hadoop deployment and network internals. This eliminates the leak of network internals and renders the Hadoop cluster as an opaque black box to attackers.

Traditional Hadoop security uses Kerberos for authentication. This is a very strong and reliable means to provide authentication and SSO for accessing Hadoop resources. However, many enterprises are looking for more flexibility and integration capabilities for their enterprise identity management and SSO solutions.

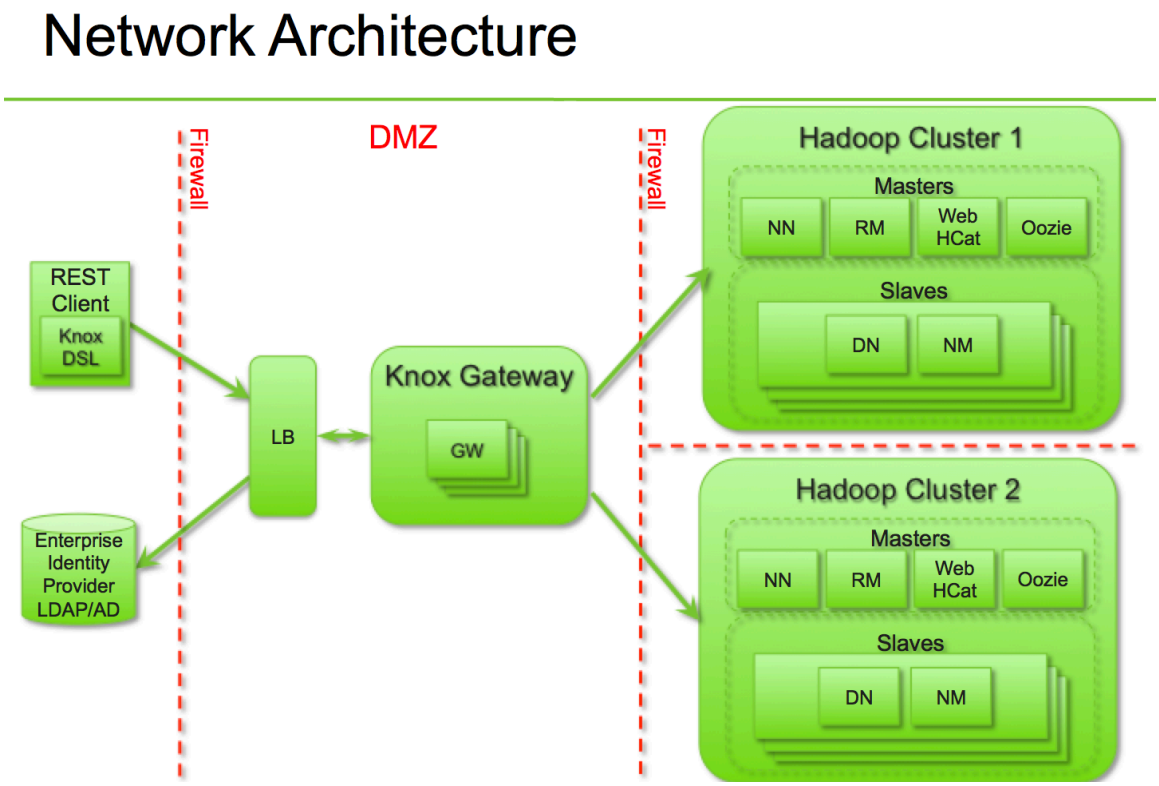
Through its pluggable security provider architecture, the Knox Gateway enables the integration of a myriad of authentication mechanisms and protocols. It facilitates access control mechanisms through the authorization provider pluggability.

Out-of-the-box, Apache Knox provides HTTP BASIC authentication against an LDAP user store. It also includes a simple Service Level Access Control provider for authorization.

Finally, Apache Knox promotes the use of REST/HTTP and the Hadoop Web APIs. This model is a proven, scalable, and flexible mechanism for client interoperability across languages, operating systems, and computing devices. By adopting the use of the Hadoop Web APIs through Knox, clients do not need to have a local Hadoop installation in order for RPC clients to find configuration and Hadoop CLI jars.

Overall, Apache Knox Gateway advances the use of the Hadoop Web APIs while offering extended security integration options for customers. At the same time, Knox provides perimeter security features that are unattainable without a solution like Apache Knox.

The following diagram shows how Apache Knox fits in a Hadoop deployment.




## System Requirements

Apache Knox Technical Preview has the following minimum system requirements:

- Hortonworks Data Platform (HDP)
- Operating Systems
- Software Requirements
- JDK Requirements

### *Hortonworks Data Platform*

Knox requires the HDP 2.0 Sandbox, available here:

http://hortonworks.com/products/hortonworks-sandbox/

### *Operating systems*

- 64-bit Red Hat Enterprise Linux (RHEL) 6
- 64-bit CentOS 6
- Oracle Linux 6

### *Software Requirements*

- yum
- rpm
- curl
- wget
- unzip
- tar

### *JDK Requirements*

Your system must have the correct JDK installed on all the nodes of the cluster. HDP supports the following JDKs.

- Oracle JDK 1.6 update 31 64-bit
- Oracle JDK 7 64-bit
- Open JDK 7 64-bit

**Note:** Set up your environment to define JAVA_HOME and put the Java Virtual Machine and the Java binaries on your path.

```
export JAVA_HOME=/usr/java/default
export PATH=$JAVA_HOME/bin:$PATH
```

After you complete this step, verify that the JRE is available on the PATH. Execute the command below to check:

```
java -version
```

You should see output similar to what is shown below. If not, update your JDK before proceeding.

```
java version "1.6.0_65"
Java(TM) SE Runtime Environment (build 1.6.0_65-b14-462-11M4609)
Java HotSpot(TM) 64-Bit Server VM (build 20.65-b04-462, mixed mode)
```

## Installation

Before you install the Knox gateway, you must configure a Hadoop cluster for it. The following instructions describe how to install the HDP 2.0 Sandbox to create a Hadoop cluster and then install Knox on a local machine to access the Sandbox.

### *Sandbox Installation*

The Knox Technical Preview takes advantage of the HDP 2.0 Sandbox as a Hadoop cluster for which it can act as a gateway. The Sandbox allows the examples to work unmodified in most cases. Other HDP installations can be used but they require changes to the Knox default configuration.

1. Download and Install HDP 2.0 Sandbox, located here:

   http://hortonworks.com/products/hortonworks-sandbox/

2. Start the HDP 2.0 Sandbox on your virtual environment using the instructions provided.

### *Knox Installation*

To install Knox, simply extract the contents of the distribution archive.

1. Download the Knox archive:

   ```
   wget http://public-repo-1.hortonworks.com/HDP\
   -LABS/Projects/Knox/1.3.3.0-59/tars/knox-incubating\
   -0.3.0.1.3.3.0-59.tar.gz
   ```

   **Note:** We've tested these commands on various PDF viewers and found that your experience copying them may vary. If you have any issues copy-pasting commands in this document, try copying first to a text editor and removing any formatting.

2. Extract the Knox archive to a known location. This location will be referred to as your `{GATEWAY_HOME}` in subsequent steps.

   ```
   tar -zxf knox-incubating-0.3.0.1.3.3.0-59.tar.gz
   cd knox-incubating-0.3.0.1.3.3.0-59
   ```

   **Note:** Remember, your current directory is now `{GATEWAY_HOME}`.

## Setup

Complete the following steps to configure Knox:

1. **Verify that the HDP 2.0 Sandbox is running.** Sandbox typically uses port forwarding so that it is accessible via localhost. Use the following command to verify that this is the case in your environment.

```
curl -ik -X GET \
'http://localhost:50070/webhdfs/v1/?op=LISTSTATUS'
```

You should see output similar to the output below where the '...' represents more JSON content.

```
{"FileStatuses":{"FileStatus":[...]}}
```

2. **Start the demonstration LDAP server provided with Knox.** By default, Knox authenticates users against an LDAP directory. The command below starts a simple LDAP directory server in order to simplify the evaluation process. This LDAP directory contains a single user with the name 'guest' and the password 'guest-password'.

   **Note:** The demonstration LDAP server uses the users defined in `{GATEWAY_HOME}/conf/users.ldif`.

   ```
   cd {GATEWAY_HOME}
   java -jar bin/ldap.jar conf &
   ```

   **Note:** You can start an additional console window for the next step to prevent the output from the LDAP server from being mixed with the output of the Knox server. If you do, remember to set the current directory to `{GATEWAY_HOME}`.

3. **Seed the Knox encryption secrets.**
   The gateway performs encryption in certain circumstances and so requires persistent secrets. Knox prompts you for a secret to type, but you do not need to remember the secret and can repeat this command at anytime.
   **Note:** Setting up a cluster of Knox instances requires the same secret to be used on each.

   ```
   java -jar bin/gateway.jar -persist-master -nostart
   ```

### *Running Knox*

To run Knox, complete the following steps:

1. **Start Knox.** This step starts the Knox Gateway server and prints diagnostic output directly to the console for evaluation purposes. To change this behavior, use the `{GATEWAY_HOME}/conf/log4j.properties` or pipe the output to a file.

   ```
   cd {GATEWAY_HOME}
   java -jar bin/gateway.jar
   ```

2. **Open an additional console window and change the current directory in that console to your** `{GATEWAY_HOME}`. Because the previous command was started in the foreground, open an additional console in order to proceed with the following examples. When you, change the current directory in that console to your `{GATEWAY_HOME}`.

   ```
   cd {GATEWAY_HOME}
   ```

## Using Knox for Common Hadoop Use Cases

This section provides an end-to-end example using Knox. First it describes creating a few directories in HDFS, uploading files to the newly created directories, and running a MR job. Then, it provides the steps for querying the status of a job and viewing the job results.

**IMPORTANT:** All of the steps in this section assume that your current directory is your `{GATEWAY_HOME}` directory.

### *Create Directories*

You need to create directories in HDFS to test a guest user.

Use following command to create `/user/guest/test`:

```
curl -iku guest:guest-password -X PUT "https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test\
?op=MKDIRS&permission=777"
```

Create an input directory under `/user/guest/test`:

```
curl -iku guest:guest-password -X PUT "https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test/input\
?op=MKDIRS&permission=777"
```

Create a lib directory under `/user/guest/test`:

```
curl -iku guest:guest-password -X PUT "https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test/lib\
?op=MKDIRS&permission=777"
```

You are now ready to upload files to these directories.

### *Upload Files from local to HDFS*

Upload `hadoop-examples.jar` to `/user/guest/test/lib`.

```
curl -iku guest:guest-password -L -T samples/hadoop-examples.jar \
-X PUT https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test/lib/hadoop-examples.jar\
?op=CREATE
```

Now `hadoop-examples.jar` is uploaded to HDFS directory `/user/guest/test/lib`.

Similarly, upload a README file to `/user/guest/test/input`. Upload the `{GATEWAY_HOME}/README` file into the HDFS directory `/user/guest/test/input`.

```
curl -iku guest:guest-password -X PUT -L -T README \
-X PUT "https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test/input/README\
?op=CREATE"
```

### *Run the MR job*

Next, run the MapReduce job against the text file uploaded to the HDFS directory `/user/guest/test/input` and place the result in `/usr/guest/test/output`:

```
curl -iku guest:guest-password -X POST \
-d arg=/user/guest/test/input \
-d arg=/user/guest/test/output \
-d jar=/user/guest/test/lib/hadoop-examples.jar \
-d class=wordcount \
https://localhost:8443/gateway/sandbox/templeton/v1/mapreduce/jar
```

### *Query the jobs for a user*

Query the user's job queue with the following command:

```
curl -iku guest:guest-password \
https://localhost:8443/gateway/sandbox/templeton/v1/queue
```

The JSON output of the previous command would be similar to:

```
{"job_1385080496897_0001","job_1385080496897_0005","
job_1385080496897_0003"}
```

The output shows the job numbers of the jobs submitted by the user. To query the status of the most recent job, take the highest job number.

Now issue the following example command to query the status of a given job:

```
curl -iku guest:guest-password \
https://localhost:8443/gateway/sandbox/templeton/v1/queue/\
job_1386196988369_0001
```

The output will be JSON content providing details about the job.

```
{"status":{"mapProgress":1.0,"reduceProgress":"NaN","cleanupProgress":0
.0,"setupProgress":0.0,"runState":2,"startTime":0,"queue":"default","pr
iority":"NORMAL","schedulingInfo":"NA","failureInfo":"NA","jobACLs":{},
"jobName":"TempletonControllerJob","jobFile":"https://localhost:8443/ga
teway/sandbox/webhdfs/v1/mr-
history/done/2013/11/22/000000/job_1385080496897_0005_conf.xml","finish
finishTime":0,"historyFile":"","trackingUrl":"http://sandbox.hortonwork
s.com:19888/jobhistory/job/job_1385080496897_0005","numUsedSlots":0,"nu
mReservedSlots":0,"usedMem":0,"reservedMem":0,"neededMem":0,"jobID":{"i
d":5,"jtIdentifier":"1385080496897"},"jobPriority":"NORMAL","username":
"guest","jobId":"job_1385080496897_0005","state":"SUCCEEDED","retired":
false,"uber":false,"jobComplete":true},"profile":{"user":"guest","jobFi
le":":"https://localhost:8443/gateway/sandbox/webhdfs/v1/mr-
```

```
history/done/2013/11/22/000000/job_1385080496897_0005_conf.xml","url":"
http://sandbox.hortonworks.com:19888/jobhistory/job/job_1385080496897_0
005","queueName":"default","jobID":{"id":5,"jtIdentifier":"138508049689
7"},"jobName":"TempletonControllerJob","jobId":"job_1385080496897_0005"
},"id":"job_1385080496897_0005","parentId":null,"percentComplete":null,
"exitValue":0,"user":"guest","callback":null,"completed":"done"}
```


**Note:** The **state":"SUCCEEDED"** and **"jobComplete":true** values provide the desired messages and have been bolded to draw your attention.

### *View the job output*

After the job completes its output is under the output dir. To see the content of this directory, issue the following command:

```
curl -iku guest:guest-password -X GET https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test/output\
?op=LISTSTATUS
```

The output will show two files, the MapReduce result is in file named **part-r-00000.**

To read the contents of the result file, use the following command.

```
curl -iku guest:guest-password -L -X GET https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test/output/part-r-00000\
?op=OPEN
```

### *Remove a directory*

The following example removes the output directory under `/user/guest/test` and its content recursively. This will be important if you want to repeat the steps above again.

```
curl -iku guest:guest-password -X DELETE "https://localhost:8443\
/gateway/sandbox/webhdfs/v1/user/guest/test\
?op=DELETE&recursive=true"
```

## Fixed Issues

The Hortonworks Knox Technical Preview is based on Apache Knox 0.3.0 and includes additional fixes for the following issues:

- KNOX-203 Gateway fails to start when `{GATEWAY_HOME}/bin` not writable
- KNOX-205 Launcher script (`gateway.sh`) not working when gateway installed via RPM
- KNOX-224 Update samples to be consistent with Hive 0.12 configuration parameter names

## Using Hive with Knox

Using Hive on the Knox Technical Preview currently requires additional configuration that should not be required when Knox is generally available.

To run Hive on Knox:

1. **Modify your Sandbox VM configuration to work with Hive on Knox.**
   a) **Stop Sandbox VM for your VM configuration.**
   For example if you are using VirtualBox-based Sandbox:
      1. Select VM "Hortonworks Sandbox 2.0"
      2. **Machine** > **Close** > **ACPI Shutdown**
   **b) Increase Sandbox VM memory to 4Gb to accommodate Ambari.**
      1. Select **Machine** > **Settings.**
      2. From the **System** tab, select the **Motherboard** sub-tab > **Base Memory=4096**
   c) **Create the port mapping for HiveServer2's HTTP port 10001.**
   This makes the HiveServer2 HTTP accessible via localhost:10001.
      1. **Network** tab > **Advanced** section > **Port Forwarding**
      2. Select "+" Button; Name=10001; Host Port=10001; Guest Port=10001
      3. Click **OK**.
   **d) Start your Sandbox VM.**
      1. **Machine** > **Start**
2. **Visit http://localhost:8000/about/ and Enable Ambari.**
3. **Modify the Hive configuration.**
   (The Ambari default user is admin/admin.)
   **a) Visit http://localhost:8080/#/main/services/HIVE/summary**
      1. Stop the Hive service.
      2. Wait for the Hive services to stop.
   **b) Visit http://localhost:8080/#/main/services/HIVE/configs**
      1. In the **Advanced** section, change the property named `hive.server2.enable.doAs` to value false. (The current version of HiveServer2 does not support this in combination with the http transport mode.)
      2. In the **Custom** section, add a property named `hive.server2.transport.mode` with value http. This enables the HTTP transport for HiveServer2.
      3. **Save your configuration.**
   **c) Restart the Sandbox VM.**
4. **Execute Knox's Hive samples.**
   The following commands are to be executed on the host and in the directory where Knox gateway is unzipped. This directory is referred to as the {GATEWAY_HOME} directory.
   **a) Verify that the Knox gateway and LDAP server are running.**
   b) **Copy the Hive client JARs to Knox .**
   **Note:** You are prompted for root password twice. The default password is `hadoop`.

```
scp -P 2222 \
root@localhost:/usr/lib/hadoop/hadoop-common-2.2.0.2.0.6.0-76.jar \
./ext
scp -P 2222 \
root@localhost:/usr/lib/hive/lib/\{hive-jdbc-0.12.0.2.0.6.0-76.jar,\
hive-service-0.12.0.2.0.6.0-76.jar,\
hive-common-0.12.0.2.0.6.0-76.jar,\
libthrift-0.9.0.jar\} \
./ext
```

**c) Copy sample log file from `{GATEWAY_HOME}/samples` directory to the Sandbox**

```
scp -P 2222 samples/hive/sample.log \
root@localhost:/tmp/sample.log
```

d) **Run the Hive Groovy sample.**

This sample demonstrates some basic Hive operations. In particular it demonstrates creating a table, loading data into the table from the sample log file and executing a select query on that table. Review the Groovy source file in the following command for details:

```
java -Djavax.net.ssl.trustStore=\
conf/security/keystores/gateway.jks \
-jar bin/shell.jar \
samples/hive/groovy/jdbc/sandbox/HiveJDBCSample.groovy
```

The resulting output should be many lines of the form:

```
2012-02-03 --- 20:11:56
2012-02-03 --- 20:11:56
2012-02-03 --- 20:11:56
```

These are the values from the first and second column of the sample log table.

**Note:** If you receive an error indicating "`HTTP Response code: 500`", use Ambari to verify that Hive is running. If it is not running, restart the Sandbox VM again. In extreme cases, you may need to delete the Hive PID file by executing:

```
ssh -p 2222 root@localhost\
"rm -f /var/run/hive/hive-server.pid"
```

before restarting the Sandbox VM.

e) **(Optional) Run the Hive Java sample.**

Functionally equivalent to Groovy example above, the Java sample is available if you are not used to Groovy and want to work directly with Java.

```
javac -d samples/hive/java/jdbc/sandbox \
samples/hive/java/jdbc/sandbox/HiveJDBCSample.java
java -Djavax.net.ssl.trustStore=\
conf/security/keystores/gateway.jks \
-cp samples/hive/java/jdbc/sandbox:dep/*:ext/* \
HiveJDBCSample
```

(Optional) If you intend to use the Hive JDBC driver independently from the Knox {GATEWAY_HOME}, you will need the complete list of JARs required to use the Hive JDBC driver:

```
/usr/lib/hadoop/hadoop-common-2.2.0.2.0.6.0-76.jar
/usr/lib/hive/lib/hive-jdbc-0.12.0.2.0.6.0-76.jar
/usr/lib/hive/lib/hive-service-0.12.0.2.0.6.0-76.jar
```

```
/usr/lib/hive/lib/libthrift-0.9.0.jar
/usr/lib/hive/lib/httpcore-4.1.4.jar
/usr/lib/hive/lib/httpclient-4.1.3.jar
/usr/lib/hive/lib/hive-common-0.12.0.2.0.6.0-76.jar
/usr/lib/hive/lib/commons-logging-1.1.1.jar
/usr/lib/hive/lib/slf4j-api-1.7.5.jar
/usr/lib/hive/lib/slf4j-log4j12-1.7.5.jar
/usr/lib/hive/lib/log4j-1.2.17.jar
/usr/lib/hive/lib/commons-codec-1.7.jar
```

## Known Issues and Limitations

At the time of this release, these are the latest known issues and limitations for Knox:

### *KNOX-188: encryptQueryString password is recreated when topology is changed*

**Problem:** URLs with encrypted query strings, which contain sensitive cluster internal info, fail to be decrypted by the gateway after a cluster topology is re-deployed in the gateway instance. On redeployment the password for `encryptQueryString` is regenerated even though the credential store already exists for a given topology file (cluster). This must be changed to only generate if the alias does NOT already exist inside the credential store. This only affects Knox HA cluster deployments. The effect will be that gateway instances cannot decrypt URLs generated by other gateway instances after failover if the password are not kept in sync after each topology deployment.

**Workaround:** Manually synchronize the content of the `{GATEWAY_HOME}/conf/security` folder after topology deployment changes.

### *KNOX-199: ExampleHBase.groovy fails with HBase 0.96 due to empty column qualifier REST API incompatibility*

**Problem:** When running the samples`/ExampleHBase.groovy` Knox Client DSL script, an exception is produced on the client side. This is due to an incompatibility between Knox 0.3.0 Client DSL and a change in HBase 0.96. The result is that the Knox Client DSL cannot be used to access HBase operations that involve empty column qualifiers. This only affects the Knox Client DSL. The HBase REST API can still be used via the Knox Gateway for both HBase 0.95 and 0.96 via other REST clients.

```
// This Client DSL command will fail with Knox 0.3.0 Client DSL and
HBase 0.96.
// It will work with HBase 0.95.

HBase.session(session).table("table1").row("row_id_1").store().column("
family1", null, "fam_value1").now()

// This Client DSL command will work with both HBase 0.95 and HBase
0.96.
```

```
HBase.session(session).table("table1").row("row_id_1").store().column("
family1", "col1", "col_value1").now()
```

### *Secure Oozie POST/PUT Request Payload Size Restriction*

**Problem:** With one exception, there are no known size limits for requests or responses payloads that pass through the gateway. The exception involves POST or PUT request payload sizes for Oozie in a Kerberos-secured Hadoop cluster. In this case, there is a 4Kb payload size limit for the first request made to the Hadoop cluster. This is a result of how the gateway negotiates a trust relationship between itself and the cluster via SPNEGO.

### *LDAP Groups Acquisition*

**Problem:** The LDAP authenticator currently does not support the acquisition of group information out-of-the-box.

**Workaround:** You can implement a custom Shiro Realm extension.

### *Group Membership Propagation*

**Problem:** Groups that are acquired via Identity Assertion Group Principal Mapping are not propagated to the Hadoop services. Therefore groups used for Service Level Authorization policy may not match those acquired within the cluster via `GroupMappingServiceProvider` plugins.

### *Demonstration LDAP Server Exceptions*

The demonstration LDAP server provided with Knox uses ApacheDS 1.5.5. This exception is a known issue and that version of ApacheDS and can be safely ignored.

```
2013-11-23 00:20:05,104 WARN  Unexpected exception forcing session to
close: sending disconnect notice to client.

java.lang.NullPointerException
```

## Forums

Visit the Hortonworks forums for the latest discussions on issues:

http://hortonworks.com/community/forums/

## Troubleshooting

You can find Knox Troubleshooting advice in at the Apache Knox Community, at the following location:
http://knox.incubator.apache.org/books/knox-incubating-0-3-0/knox-incubating-0-3-0.html#Troubleshooting

## Further Reading

The latest Apache Knox documentation is available here:

http://knox.incubator.apache.org/books/knox-incubating-0-3-0/knox-incubating-0-3-0.html

**About Hortonworks**

Hortonworks is a leading commercial vendor of Apache Hadoop, the preeminent open source platform for storing, managing and analyzing big data. Hortonworks Data Platform provides an open and stable foundation for enterprises and a growing ecosystem to build and deploy big data solutions. Hortonworks is the trusted source for information on Hadoop, and together with the Apache community, Hortonworks is making Hadoop easier to install, manage and use. Hortonworks provides technical support, training & certification programs for enterprises, systems integrators & technology vendors.

3460 W. Bayshore Rd.
Palo Alto, CA 94303 USA

**US**: 1.855.846.7866
**International:** 1.408.916.4121
**www.hortonworks.com**