Hortonworks Data Platform

Knox Gateway Administration Guide

(October 13, 2015)

docs.cloudera.com

Hortonworks Data Platform: Knox Gateway Administration Guide

Copyright © 2012-2014 Hortonworks, Inc. Some rights reserved.

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain, free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to contact us directly to discuss your specific needs.

Except where otherwise noted, this document is licensed under
Creative Commons Attribution ShareAlike 4.0 License.
http://creativecommons.org/licenses/by-sa/4.0/legalcode

Table of Contents

List of Tables

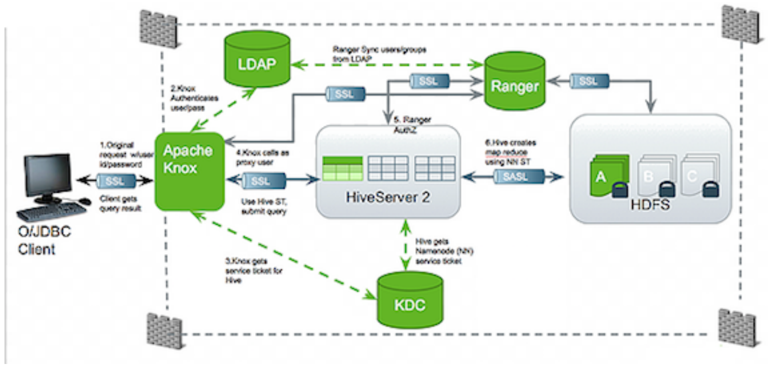
1. Apache Knox Gateway Overview

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

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

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

- **Simplifies access:** Extends Hadoop's REST/HTTP services by encapsulating Kerberos to within the Cluster.
- **Enhances security:** Exposes Hadoop's REST/HTTP services without revealing network details, providing SSL out of the box.
- **Centralized control:** Enforces REST API security centrally, routing requests to multiple Hadoop clusters.
- **Enterprise integration:** Supports LDAP, Active Directory, SSO, SAML and other authentication systems.


LDAP
Ranger Sync users/groups from LDAP
Ranger
SSL
SSL
SSL
2.Knox Authenticates user/pass
5. Ranger AuthZ
1.Original request w/user id/password
Apache Knox
4.Knox calls as proxy user
6.Hive creates map reduce using NN ST
SSL
SSL
SASL
Client gets query result
Use Hive ST, submit query
HiveServer 2
A
B
C
HDFS
O/JDBC Client
Hive gets Namenode (NN) service ticket
3.Knox gets service ticket for Hive
KDC

Typical Security Flow: Firewall, Routed Through Knox Gateway

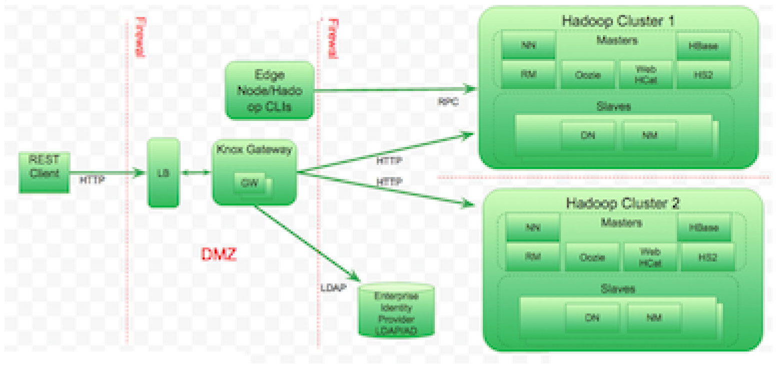
Knox can be used with both unsecured Hadoop clusters, and Kerberos secured clusters. In an enterprise solution that employs Kerberos secured clusters, the Apache Knox Gateway provides an enterprise security solution that:

- Integrates well with enterprise identity management solutions
- Protects the details of the Hadoop cluster deployment (hosts and ports are hidden from end users)
- Simplifies the number of services with which a client needs to interact

1.1. Knox Gateway Deployment Architecture

Users who access Hadoop externally do so either through Knox, via the Apache REST API, or through the Hadoop CLI tools.

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



NN=NameNode, RM=Resource Manager, DN=DataNote, NM=NodeManager

1.2. Supported Hadoop Services

Apache Knox Gateway supports the following Hadoop services versions in both Kerberized and Non-Kerberized clusters:

Table 1.1. Supported Hadoop Services

Service	Version
YARN	2.6.0
WebHDFS	2.6.0
WebHCat/Templeton	0.13.0
Oozie	4.1.0
HBase/Stargate	0.98.4
Hive (via WebHCat)	0.14.0
Hive (via JDBC)	0.14.0

2. Configuring the Knox Gateway

This section describes how to configure the Knox Gateway. This section describes how you can:

2.1. Create and Secure the Gateway Directories

Installing Knox Gateway with the platform-specific installers creates the following directories:

- HADOOP_NODE_INSTALL_ROOT
- knox-X.X.X.X.X.X.X-XXXX – the $gateway directory. For example, *D:/ hdp/knox-0.4.0.2.1.1.0-1557* The directory contains the following files:

Table 2.1. Apache Service Gateway Directores

Directory/Filename	Description
conf/topologies	Contains global gateway configuration files.
bin	Contains the executable shell scripts, batch files, and JARs for clients and servers.
deployments	Contains cluster topology descriptor files that define Hadoop clusers.
lib	Contains the JARs for all the components that make up the gateway.
dep	Contains the JARs for all of the component upon which the gateway depends.
ext	A directory where user-supplied extensions JARs can be placed to extends the gateway functionality.
samples	Contains a number of samples that can be used to explore the functionality of the gateway.
templates	Contains default configuration files that can be copied and customized.
README	Provides basic informaiton about the Apache Knox Gateway.
ISSUES	Describes significant known issues.
CHANGES	Enumerates the changes between releases.
LICENSE	Documents the license under which this software is provided.
NOTICE	Documents required attribution notices for included dependencies.
DISCLAIMER	Documents that this release is from a project undergoing incubaiton at Apache.

- *SystemDrive/hadoop/logs knox* –contains the output files from the Knox Gateway.

2.2. Customize the Gateway Port and Path

The Knox Gateway properties effect the URL used by the external clients to access the internal cluster. By default the port is set to 8443 and the context path is gateway.

To change the context path or port:

1. Edit `gateway-site.xml` and modify the following properties:

   - `property`name
   - `gateway.port name`value
   - `gateway.port`value

   where:

   - `$gateway_port` is the HTTP port for the gateway (default port=8443)
   - `$gateway` is the context path in the external URL (preconfigured value=gateway). For example, https://knox.hortonworks.com:8443/hadoop/,where hadoop is the context path.

2. Restart the gateway:

   ```
   cd $gateway.bin/gateway.sh stop bin/gateway.sh start
   ```

   The gateway loads the new configuration on startup.

2.3. Manage the Master Secret

The master secret is required to start the gateway. The secret protects artifacts used by the gateway instance, such as the keystore, trust stores and credential stores.

You configure the gateway to persist the master secret, which is saved in the `$gateway / data/security/master` file. Ensure that this directory has the appropriate permissions set for your environment. To set the master secret, enter:

```
cd $gateway bin/knoxcli.cmd create-master
```

A warning displays indicating that persisting the secret is less secure than providing it at startup. Knox protects the password by encrypting it with AES 128 bit encryption; where possible, the file permissions are set to be accessible only by the knox user.

Warning

Ensure that the security directory, `$gateway/data/security`, and its contents are readable and writable only by the knox user. This is the most important layer of defense for master secret. **Do not assume that the encryption is sufficient protection.**

Changing the Master Secret

The Master Secret can be changed under dire situations where the Administrator has to redo all the configurations for every dateway instance in a deployment, and no longer knows the Master Secret. Recreating the Master Secret requires not only recreating the master, but also removing all existing keystores and reprovisioning the certificates and credentials.

1. To change the Master Secret:

   ```
   cd $gateway bin/knoxcli.cmd create-master--force
   ```

2. If there is an existing keystore, update the keystore.

2.4. Manually Redeploy Cluster Topologies

You are not required to manually redeploy clusters after updating cluster properties. The gateway monitors the topology descriptor files in the `$gateway/conf/topologies` directory and automatically redepolys the cluster if any descriptor changes or a new one is added. (The corresponding deployment is in `$gateway/data/deployments`.)

However, you must redeploy the clusters after changing any of the following gateway properties or gateway-wide settings:

- Time settings on the gateway host
- Implementing or updating Kerberos
- Implementing or updating SSL certificates
- Changing a cluster alias

Redeploying all clusters at the same time

When making gateway-wide changes (such as implementing Kerberos or SSL), or if you change the system clock, you must redeploy all the Cluster Topologies. Do the following:

1. To verify the timestamp on the currently deployed clusters enter: `cd $gatewaydir data/deployments`. The system displays something similar to:

   ```
   Directory of C:/hdp/knox-0.4.0.2.1.1.0-1557/data/deployments

   04/17/2014 05:30 PM <DIR> .
   04/17/2014 05:30 PM <DIR> ..
   04/17/2014 05:30 PM <DIR> cluster.war.145514f4dc8
   04/17/2014 05:30 PM <DIR> myCluster.war.145514f4dc8
   04/11/2014 08:35 AM <DIR> sandbox.war.145514f4dc8
    0 File(s) 0 bytes
    5 Dir(s) 9,730,977,792 bytes free
   ```

2. To redeploy all clusters, enter `/bin/knoxcli.cmd redeploy`.

3. To verify that a new cluster WAR was created, enter `cd $gatewaydir data/ deployments`.The system displays something similar to:

```
Directory of C:/hdp/knox-0.4.0.2.1.1.0-1557/data/deployments

04/17/2014 05:34 PM <DIR> .
04/17/2014 05:34 PM <DIR> ..
04/17/2014 05:30 PM <DIR> cluster.war.145514f4dc8
04/17/2014 05:34 PM <DIR> cluster.war.1457241b5dc
04/17/2014 05:30 PM <DIR> myCluster.war.145514f4dc8
04/17/2014 05:34 PM <DIR> myCluster.war.1457241b5dc
04/11/2014 08:35 AM <DIR> sandbox.war.145514f4dc8
04/17/2014 05:34 PM <DIR> sandbox.war.1457241b5dc
 0 File(s) 0 bytes
 8 Dir(s) 9,730,850,816 bytes free
```

A new file is created for each cluster, with the current timestamp.

Redeploy only specific clusters

When making changes that impact a single cluster, such as changing an alias or restoring from an earlier cluster topology descriptor file, you only redeploy the effected cluster. Do the following:

1. To verify the timestamp on the currently deployed Cluster Topology WAR files, enter: `cd $gatewaydir data/deployments.`The system displays something similar to:

```
Directory of C:/hdp/knox-0.4.0.2.1.1.0-1557/data/deployments

04/17/2014 05:30 PM <DIR> .
04/17/2014 05:30 PM <DIR> ..
04/17/2014 05:30 PM <DIR> cluster.war.145514f4dc8
04/17/2014 05:30 PM <DIR> myCluster.war.145514f4dc8
04/11/2014 08:35 AM <DIR> sandbox.war.145514f4dc8
 0 File(s) 0 bytes
 5 Dir(s) 9,730,977,792 bytes free
```

2. To redeploy a specific cluster, enter:

```
cd $gateway bin/knoxcli.cmd redeploy --cluster $cluster_name
```

where `$cluster_name` is the name of the cluster topology descriptor (without the `.xml` extension). For example, myCluster.

3. To verify that the cluster was deployed, enter: `cd $gatewaydir data/deployments`. The system displays something similar to:

```
Directory of C:/hdp/knox-0.4.0.2.1.1.0-1557/data/deployments

04/17/2014 05:30 PM <DIR> .
04/17/2014 05:30 PM <DIR> ..
04/17/2014 05:30 PM <DIR> cluster.war.145514f4dc8
04/17/2014 05:30 PM <DIR> myCluster.war.145514f4dc8
04/17/2014 05:34 PM <DIR> myCluster.war.1457241b5dc
04/11/2014 08:35 AM <DIR> sandbox.war.145514f4dc8
 0 File(s) 0 bytes
 5 Dir(s) 9,730,977,792 bytes free
```

You should see that existing cluster war files are unchanged, but the war file for myCluster was updated (has a current timestamp).

2.5. Manually Start and Stop Apache Knox

Except for changes to ../ ../conf/topology/*.xml, changes to the Knox Gateway global settings in `$gateway /conf/gateway-site.xml` cannot be loaded before the Gateway is restarted.

To manually stop Knox:

```
cd $gateway/bin/gateway.sh stop
```

This is known as a clean shutdown, as the gateway script cleans out all `.out`and `.err` files in the logs directory.

To manually start Knox for the first time, or re-start Knox after a clean shutdown:

```
cd $gateway /bin/gateway.sh start
```

To manually re-start Knox after an unclean shutdown:

```
cd $gateway/bin/gateway.sh clean /bin/gateway.sh start
```

This command eliminates old `.out`and `.err` files in the logs directory.

3. Defining Cluster Topologies

The Knox Gateway supports one or more Hadoop clusters. Each Hadoop cluster configuration is defined in a topology deployment descriptor file in the `$gateway/conf/topologies` directory and is deployed to a corresponding WAR file in the `$gateway/data/deployments` directory. These files define how the gateway communicates with each Hadoop cluster.

The descriptor is an XML file contains the following sections:

- `gateway/provider` – configuration settings enforced by the Knox Gateway while providing access to the Hadoop cluster.
- `service` – defines the Hadoop service URLs used by the gateway to proxy communications from external clients.

The gateway automatically redeploys the cluster whenever it detects a new topology descriptor file, or detects a change in an existing topology descriptor file.

The following table provides an overview of the providers and services:

Table 3.1. Cluster Topology Provider and Service Roles

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

Type	Role	Description
		Web Application Security for more information.
HA provider	high availability	Syncs all Knox instances to use the same topologies credentials keystores.
service	$service_name	Binds a Hadoop service with an internal URL that the gateway uses to proxy requests from external clients to the internal cluster services. Refer to Configure Hadoop Service URLs for more information.

Cluster topology descriptors have the following XML format:

```
<topology>
 <gateway>
 <provider>
 <role> </role>
 <name> </name>
 <enabled> </enabled>
 <param>
 <name> </name>
 <value> </value>
 </param>
 </provider>
 </gateway>
 <service> </service>
</topology>
```

4. Configuring the Knox Topology to Connect to Hadoop Cluster Services

The Apache Knox Gateway redirects external requests to an internal Hadoop service using service name and URL of the service definition.

This chapter describes:

4.1. Setting up Hadoop Service URLs

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

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

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

   where:

   - `$service_name` is either WEBHDFS, WEBHCAT, WEBHBASE, OOZIE, HIVE, NAMENODE, or JOBTRACKER.
   - `<url>` is the complete internal cluster URL required to access the service, including:
     - `$schema` – the service protocol
     - `$hostname` – the resolvable internal host name
     - `$port` – the service listening port

2. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway/data/deployments`.

   Note

   It is not necessary to restart the Knox server after making changes to the topology/Hadoop Cluster services.

4.2. Example Service Definitions

Configure each service that you want to expose externally, being careful to define the internal hostname and service ports of your cluster.

The following example uses the defaults ports and supported service names.

```
<service>
 <role>NAMENODE</role>
 <url>hdfs:// namenode-host :8020</url>
 </service>

 <service>
 <role>JOBTRACKER</role>
 <url>rpc:// jobtracker-host :8050</url>
 </service>

 <service>
 <role>RESOURCEMANAGER</role>
 <url>http://red3:8088/ws</url>
 </service>

 <service>
 <role>WEBHDFS</role>
 <url>http://localhost:50070/webhdfs</url>
 </service>

 <service>
 <role>WEBHCAT</role>
 <url>http://webcat-host :50111/templeton</url>
 </service>

 <service>
 <role>OOZIE</role>
 <url>http://oozie-host :11000/oozie</url>
 </service>

 <service>
 <role>WEBHBASE</role>
 <url>http://webhbase-host :60080</url>
 </service>

 <service>
 <role>HIVE</role>
 <url>http://hive-host :10001/cliservice</url>
 </service>
```

4.3. Validating Service Connectivity

Use the commands in this section to test connectivity between the gateway host and the Hadoop service, and then test connectivity between an external client to the Hadoop service through the gateway.

Tip

If the communication between the gateway host and an internal Hadoop service fails, telnet to the service port to verify that the gateway is able to access the cluster node. Use the hostname and ports you specified in the service definition.

Testing WebHDFS by getting the home directory

- At the gateway host, enter `curl http:// $webhdfs-host :50070/webhdfs/v1?op=GETHOMEDIRECTORY` . The host displays:

```
{"Path":"/user/gopher"}
```

- At an external client, enter `curl https:// $gateway-host : $gateway_port / $gateway / $cluster_name / $webhdfs_service_name /v1?op=GETHOMEDIRECTORY`. The external client displays:

```
{"Path":"/user/gopher"}
```

Testing WebHCat/Templeton by getting the version

- At the gateway host, enter `curl http:// $webhdfs-host :50070/templeton/v1/version`. The host displays:

```
{"supportedVersions":["v1"],"version":"v1"}
```

- At an external client, enter `curl https:// $gateway-host : $gateway_port / $gateway / $cluster_name / $webhcat_service_name /v1/version` . The external client displays:

```
{"supportedVersions":["v1"],"version":"v1"}
```

Testing Oozie by getting the version

- At the gateway host, enter `curl http://` $oozie-host `:11000/oozie/v1/admin/build-version`. The host displays:

```
{"buildVersion":"4.0.0.2.1.1.0-302"}
```

- At an external client, enter `curl https://` $gateway-host `:` $gateway_port `/` $gateway `/` $cluster_name `/` $oozie_service_name `/v1/admin/build-version`. The external client displays:

```
{"buildVersion":"4.0.0.2.1.1.0-302"}
```

Testing HBase/Stargate by getting the version

- At the gateway host, enter `curl http:// $hbase-host :17000/version`. The host displays:

```
rest 0.0.2 JVM: Oracle Corporation 1.7.0_51-24.45-b08 OS: Linux 3.8.0-29-
generic amd64 Server: jetty/6.1.26 Jersey: 1.8:
```

- At an external client, enter `curl http:// $hbase-host :17000/version`. The external client displays:

```
rest 0.0.2 JVM: Oracle Corporation 1.7.0_51-24.45-b08 OS: Linux 3.8.0-29-
generic amd64 Server: jetty/6.1.26 Jersey: 1.8
```

Testing HiveServer2

Both of the following URLs return an authentication error, which users can safely ignore.

1. At the gateway host, enter:

   ```
   curl http:// $hive-host :10001/cliservice
   ```

2. At an external client, enter:

   ```
   curl https:// $gateway-host : $gateway_port / $gateway /
   $cluster_name / $hive_service_name /cliservice
   ```

5. Mapping the Internal Nodes to External URLs

Hostmapping is an advanced configuration topic. Generally, it is only required in deployments in virtualized environments, such as those used in cloud and some development and testing environments.

The isolation of the Hadoop cluster is accomplished through virtualization that will hide the internal networking details (such as IP addresses and/or hostnames) from the outside world, while exposing other IP addresses and/or hostnames for use by clients accessing the cluster from outside of the virtualized environment. The exposed IP addresses and hostnames are available for use in the topology descriptor service definitions. This configuration works great for requests that are initiated from the external clients themselves which only ever use the Knox Gateway exposed endpoints.

Difficulties from these virtualized environments arise when the Hadoop cluster redirects client requests to other nodes within the cluster and indicates the internal hostname locations, rather than those designated to be exposed externally. Since the Hadoop services don't know or care whether a request is coming from an external or internal client, it uses its only view of the cluster, which is the internal details of the virtualized environment.

The Knox Gateway needs to know how to route a request that has been redirected by the Hadoop service to an address that is not actually accessible by the gateway. Hostmapping acts as an adapter that intercepts the redirects from the Hadoop service and converts the indicated internal address to a known external address that Knox will be able to route to once the client resends the request through the client facing gateway endpoint. The gateway uses the hostmap to replace the internal hostname within the routing policy for the particular request with the externally exposed hostname. This enables the dispatching from the Knox Gateway to successfully connect to the Hadoop service within the virtualized environment. Otherwise, attempting to route to an internal-only address will result in connection failures.

A number of the REST API operations require multi-step interactions that facilitate the client's interaction with multiple nodes within a distributed system such as Hadoop. External clients performing multi-step operations use the URL provided by the gateway in the responses to form the next request. Since the routing policy is hidden by the gateway from the external clients, the fact that the subsequent requests in the multi-stepped interaction are mapped to the appropriate externally exposed endpoints is not exposed to the client.

For example, when uploading a file with WebHDFS service:

1. The external client sends a request to the gateway WebHDFS service.
2. The gateway proxies the request to WebHDFS using the service URL.
3. WebHDFS determines which DataNodes to create the file on and returns the path for the upload as a Location header in a HTTP redirect, which contains the datanode host information.

4. The gateway augments the routing policy based on the datanode hostname in the redirect by mapping it to the externally resolvable hostname.

5. The external client continues to upload the file through the gateway.

6. The gateway proxies the request to the datanode by using the augmented routing policy.

7. The datanode returns the status of the upload and the gateway again translates the information without exposing any internal cluster details.

5.1. Setting Up a Hostmap Provider

Add the `hostmap` provider to the cluster topology descriptor and a parameter for each DataNode in the cluster, as follows:

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

2. Add the Hostmap provider to `topology/gateway` using the following format:

   ```
   <provider> <role>hostmap</role> <name>static</name>
   <enabled>true</enabled> <param> <name> $external-name </name>
   <value> $internal-dn-host </value> </param> </provider>
   ```

   where:

   * `$cluster-name.xml` is the name of the topology descriptor file, located in $gateway /conf/topologies.

   * `$external-name` is the value that the gateway uses to replace $internal_host host names in responses.

   * `$internal-dn-host` is a comma-separated list of host names that the gateway will replace when rewriting responses.

3. To the `hostmap` provider, add a `param` for each additional DataNode in your cluster:

   ```
   <param> <name> $external-name2 </name> <value> $internal-dn2-host </
   value> </param>
   ```

4. Save the file.

   Saving the results automatically deploys the topology with the change. The result is the creation of a new WAR file with modified timestamp in *$gateway/data/deployments*.

5.2. Example of an EC2 Hostmap Provider

In this EC2 example two VMs have been allocated. Each VM has an external hostname by which it can be accessed via the internet. However the EC2 VM is unaware of this external host name, and instead is configured with the internal hostname.

* **External hostnames** - ec2-23-22-31-165.compute-1.amazonaws.com, ec2-23-23-25-10.compute-1.amazonaws.com

- **Internal hostnames** - ip-10-118-99-172.ec2.internal, ip-10-39-107-209.ec2.internal

The following shows the Hostmap definition required to allow access external to the Hadoop cluster via the Apache Knox Gateway.

```
<topology> <gateway> ... <provider> <role>hostmap</
role> <name>static</name> <enabled>true</enabled> <!--
For each host enter a set of parameters --> <param>
<name>ec2-23-22-31-165.compute-1.amazonaws.com</name>
<value>ip-10-118-99-172.ec2.internal</value> </param>
<param> <name>ec2-23-23-25-10.compute-1.amazonaws.com</name>
<value>ip-10-39-107-209.ec2.internal</value> </param> </
provider> ... </gateway> <service> ... </service> ... </topology>
```

5.3. Example of Sandbox Hostmap Provider

Hortonwork's Sandbox 2.x poses a different challenge for hostname mapping. This Sandbox version uses port mapping to make Sandbox appear as though it is accessible via localhost. However, Sandbox is internally configured to consider sandbox.hortonworks.com as the hostname. So from the perspective of a client accessing Sandbox the external host name is localhost.

The following shows the `hostmap` definition required to allow access to Sandbox from the local machine:

```
<topology> <gateway> ... <provider> <role>hostmap</
role> <name>static</name> <enabled>true</enabled> <param>
<name>localhost</name> <value>sandbox,sandbox.hortonworks.com</
value></param> </provider> ... </gateway> ... </topology>
```

5.4. Enabling Hostmap Debugging

Warning

Changing the rootLogger value from ERROR to DEBUG generates a large amount of debug logging.

Enable additional logging by editing the `gateway-log4j.properties` file in the directory.

1. Edit the `$gateway /conf/gateway-log4j.propertiesgateway-log4j.properties` file to enable additional logging.

2. Change ERROR to DEBUG on the following line:

   ```
   log4j.rootLogger=ERROR, drfa
   ```

   ### Warning

   Changing the rootLogger value from ERROR to DEBUG generates a large amount of debug logging.

3. Stop and then restart the gateway:

   ```
   cd $gateway bin/gateway.sh stop bin/gateway.sh start
   ```

6. Configuring Authentication

Apache Knox Gateway supports authentication using either an LDAP or federation provider for each configured cluster. This section explains how to configure authentication:

- Setting Up LDAP Authentication [18]
- Setting Up HTTP Header Authentication for Federation_SSO [20]

Note

For information on how to configuretion an identity assertion provider, see Configuring Identity Assertion.

6.1. Setting Up LDAP Authentication

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

Note

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

To enable LDAP authentication:

1. Open the cluster topology descriptor file, `$cluster-name.xml`, in a text editor.
2. Add the `ShiroProvider` authentication provider to `/topology/gateway` as follows:

   ```
   <provider> <role>authentication</role> <name>ShiroProvider</
   name> <enabled> true </enabled> <param> <name>main.ldapRealm</
   name> <value>org.apache.shiro.realm.ldap.JndiLdapRealm</
   value> </param> <name>main.ldapRealm.userDnTemplate</
   name> <value> $USER_DN </value> </param>
   <name>main.ldapRealm.contextFactory.url</name>
   <value> $protocol :// $ldaphost : $port </value> </param>
   <name>main.ldapRealm.contextFactory.authenticationMechanism</
   name> <value>simple</value> </param> <name>urls./**</name>
   <value> $auth_type </value> </param> <name>sessionTimeout</name>
   <value> $minutes </value> </param> </provider>
   ```

   where:

   - `$USER_DN`

     is a comma-separated list of attribute and value pairs that define the User Distinguished Name (DN). The first pair must be set to " $attribute_name ={0}" indicating that the $attribute_name is equal to the user token parsed from the request. For example, the first attribute in an OpenLdap definition is UID={0}. The main.ldapRealm.userDnTemplate parameter is only required when authenticating against an LDAP store that requires a full User DN.

- `$protocol :// $ldaphost : $port`

  is the URL of the LDAP service, Knox Gateway supports LDAP or LDAPS protocols.

- `$auth_type`

  is either authcBasic, which provides basic authentication for both secured and non-secured requests, or ssl authcBasic, which rejects non-secured requests and provides basic authentication of secured requests.

- `$minutes`

  is the session idle time in minutes, the default timeout is 30 minutes.

3. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway/data/deployments`.

6.2. Example of an Active Directory Configuration

Typically the AD main.ldapRealm.userDnTemplate value looks slightly different than OpenLDAP. The value for `main.ldapRealm.userDnTemplate` is only required if AD authentication requires the full `User DN`.

Note

If AD can allows authentication based on the CN (common name) and password only, no value is required for `main.ldapRealm.userDnTemplate`.

```
<provider> <role>authentication</role> <name>ShiroProvider</
name> <enabled>true</enabled> <param> <name>main.ldapRealm</
name> <value>org.apache.shiro.realm.ldap.JndiLdapRealm</value>
</param> <param> <name>main.ldapRealm.userDnTemplate</name>
<value>cn={0},ou=people,dc=hadoop,dc=apache,dc=org</value>
</param> <param> <name>main.ldapRealm.contextFactory.url</
name> <value>ldap://localhost:389</value> </param> <param>
<name>main.ldapRealm.contextFactory.authenticationMechanism</
name> <value>simple</value> </param> <param> <name>urls./**</name>
<value>authcBasic</value> </param> </provider>
```

6.3. Example of an OpenLDAP Configuration

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

```
</value> </param> <param> <name>main.ldapRealm.contextFactory</
name> <value>$ldapContextFactory</value> </param>
```

6.4. Testing an LDAP Provider

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

1. Open the command line on an external client.

   Note

   cURL is not a built-in command line utility in Windows.

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

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

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

6.5. Setting Up HTTP Header Authentication for Federation_SSO

The Knox Gateway supports federation solution providers by accepting HTTP header tokens. This section explains how to configure HTTP header fields for SSO or Federation solutions that have simple HTTP header-type tokens. For further information, see the Authentication chapter of the Apache Knox 0.5.0 User's Guide.

The gateway extracts the user identifier from the HTTP header field. The gateway can also extract the group information and propagate it to the Identity-Assertion provider.

Important

The Knox Gateway federation plug-in, `HeaderPreAuth`, trusts that the content provided in the authenticated header is valid. Using this provider requires proper network security.

Only use the HeaderPreAuth federation provider in environments where the identity system does not allow direct access to the Knox Gateway. Allowing direct access exposes the gateway to identity spoofing. Hortonworks recommends defining the `preauth.ip.addresses` parameter to ensure requests come from a specific IP addresses only.

To configure the HTTP header tokens:

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

2. Add a `HeaderPreAuth` federation provider to `topology/gateway` as follows:

```
<provider> <role>federation</role> <name>HeaderPreAuth</
name> <enabled>true</enabled> <param>
<name>preauth.validation.method</name> <value> $validation_type
</value> </param> <param> <name>preauth.ip.addresses</
name> <value> $trusted_ip </value> </param> <param>
<name>preauth.custom.header</name> <value> $user_field </value> </
param> <param> <name>preauth.custom.group.header</name> <value>
$group_field </value> </param> </provider>
```

where the values of the parameters are specific to your environment:

- `$validation_type (Optional, recommended)`

  Indicates the type of trust, use either preauth.ip.validation indicating to trust only connections from the address defined in preauth.ip.addresses OR null (omitted) indicating to trust all IP addresses.

- `$trusted_ip` (Required when the pre-authentication method is set to `preauth.ip.validation`)

  A comma-separated list of IP addresses, addresses may contain a wild card to indicate a subnet, such as 10.0.0.*.

- `$user_field`

  The name of the field in the header that contains the user name that the gateway extracts. Any incoming request that is missing the field is refused with **HTTP status 401, unauthorized**. If not otherwise specified, the default value is SM_USER.

- `$group_field (Optional)`

  The name of the field in the header that contains the group name that the gateway extracts. Any incoming request that is missing the field results in no group name being extracted and the connection is allowed.

3. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway/data/deployments`.

6.6. Example of SiteMinder Configuration

The following example is the bare minimum configuration for SiteMinder (with no IP address validation):

```
<provider> <role>federation</role> <name>HeaderPreAuth</name>
<enabled>true</enabled> <param> <name>preauth.custom.header</
name> <value>SM_USER</value> </param> <param>
<name>preauth.ip.addresses</name> <value>10.10.0.*</value> </
param> </provider>
```

6.7. Testing HTTP Header Tokens

Use following cURL command to request a directory listing from HDFS while passing in the expected header SM_USER, note that the example is specific to sandbox:

```
curl -k -i --header "SM_USER: guest" -v 'https://localhost:8443/
gateway/sandbox/webhdfs/v1/tmp?op=LISTSTATUS'
```

Omitting the –header “SM_USER: guest” above results in a **HTTP status 401 unauthorized**

7. Configuring Identity Assertion

The Knox Gatewayidentity-assertion provider maps an authenticated user to an internal cluster user and/or group. This allows the Knox Gateway accept requests from external users without requiring internal cluster user names to be exposed.

The gateway evaluates the authenticated user against the identity-assertion provider to determine the following:

1. Does the user match any user mapping rules:
   - **True**:The first matching $cluster_user is asserted, that is it becomes the authenticated user.
   - **False**:The authenticated user is asserted.
2. Does the authenticated user match any group mapping rules:
   - **True**:The authenticated user is a member of all matching groups (for the purpose of authorization).
   - **False**:The authenticated user is not a member of any mapped groups.

Note

When authenticated by an SSO provider, the authenticated user is a member of all groups defined in the request as well as any that match the group.principal.mapping.

7.1. Structure of the Identity-Assertion Provider

All cluster topology descriptors must contain anidentity-assertion provider in thetopology/gateway definition.

The following is the complete structure of theidentity-assertion provider. The parameters are optional.

```
<provider>
 <role>identity-assertion</role>
 <name>Pseudo</name>
 <enabled>true</enabled>
 <param>
 <name>principal.mapping</name>
 <value> $user_ids = $cluster_user [; $user_ids = $cluster_user1 ;...]</value>
 </param>
 <param>
 <name>group.principal.mapping</name>
 <value> $cluster_users = $group1 ; $cluster_users = $group2 </value>
 </param>
 </provider>
```

where:

- $user_idsis a comma-separated list of external users or the wildcard (*) indicates all users.

- `$cluster_user` the Hadoop cluster user name the gateway asserts, that is the authenticated user name.

Note

Note that identity-assertion rules are not required; however, whenever an authentication provider is configured an `identity-assertion` provider is also required.

7.2. Define Pseudo Identity Assertion

When you define the `Pseudo identity-assertion` provider without parameters, the authenticated user is asserted as the authenticated user. For example, using simple assertion if a user authenticates as "guest", the user's identity for grouping, authorization, and running the request is "guest".

To define a basic identify-assertion provider:

1. Open the cluster topology descriptor file, `$cluster-name.xml`, in a text editor.
2. Add a `Pseudoidentity-assertion` provider to`topology/gateway` as follows:

   ```
   <provider> <role>identity-assertion</role> <name>Pseudo</name>
   <enabled>true</enabled> </provider>
   ```

3. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway/data/deployments`.

7.3. Mapping Authenticated User to Cluster

The `principal.mapping` parameter of an `identity-assertion` provider determines the user name that the gateway asserts (uses as the authenticated user) for grouping, authorization, and to run the request on the cluster.

Note

If a user does not match a principal mapping definition, the authenticated user becomes the effective user.

To add user mapping rule to an identity-assertion provider:

1. Open the cluster topology descriptor file, `$cluster-name.xml`, in a text editor.
2. Add a `Pseudo` identity-assertion provider to `topology/gateway` with the `principal.mapping` parameter as follows:

   ```
   <provider> <role>identity-assertion</role> <name>Pseudo</
   name> <enabled>true</enabled> <param> <name>principal.mapping</
   name> <value> $user_ids = $cluster_user ; $user_ids =
   $cluster_user1 ;...</value> </param> </provider>
   ```

where the value contains a semi-colon-separated list of external to internal user mappings, and the following variables match the names in your environment:

- `$user_ids`

  is a comma-separated list of external users or the wildcard (*) indicates all users.

- `$cluster_user`

  is the Hadoop cluster user name the gateway asserts, that is the authenticated user name.

3. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway/data/deployments`.

7.4. Example User Mapping

The gateway evaluates the list in order, from left to right; therefore a user matching multiple entries, resolves to the first matching instance.

In the following example, when a user authenticates as, the gateway asserts the user as, and all other users as.

```
<provider> <role>identity-assertion</role> <name>Pseudo</name>
<enabled>true</enabled> <param> <name>principal.mapping</name>
<value>guest=sam;*=dwayne</value> </param> </provider>
```

The following example shows how to map multiple users to different cluster accounts:

```
<provider> <role>identity-assertion</role> <name>Pseudo</name>
<enabled>true</enabled> <param> <name>principal.mapping</name>
<value>guest,joe,brenda,administrator=sam;janet,adam,sue=dwayne</
value> </param> </provider>
```

7.5. Mapping Authenticated Users to Groups

The Knox Gateway uses group membership for Service Level Authorization only. The gateway does not propagate the user's group when communicating with the Hadoop cluster.

The `group.principal.mapping` parameter of the identity-assertion provider determines the user's group membership. The gateway evaluates this parameter after the `principal.mapping` parameter using the authenticated user. Unlike `principal.mapping`, the group mapping applies all the matching values. A user is a member of all matching groups.

Note

Although user and group mappings are often used together, the instructions in this section only explain how to add group mappings.

7.6. Configuring Group Mapping

To map authenticated users to groups:

1. Open the cluster topology descriptor file, `$cluster-name.xml`, in a text editor.
2. Add a `Pseudo identity-assertion` provider to `topology/gateway` with the `group.principal.mapping` parameter as follows:

   ```
   <provider> <role>identity-assertion</role> <name>Pseudo</name>
   <enabled>true</enabled> <param> <name>group.principal.mapping</
   name> <value> $cluster_users = $group ; $cluster_users = $group
   </value> </param> </provider>
   ```

   where:

   - the value is a semi-colon-separated list of definitions and the variables are specific to your environment.
   - `$cluster_users` is a comma-separated list of authenticated user or the wildcard (*) indicating all users.
   - `$group` is the name of the group that the user is in for Service Level Authorization.
3. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway/data/deployments`.

7.7. Examples of Group Mapping

```
<provider> <role>identity-assertion</role> <name>Pseudo</
name> <enabled>true</enabled> <param> <name>principal.mapping</
name> <value>guest,alice=hdfs;mary=hive</value> </
param> <param> <name>group.principal.mapping</name>
<value>*=users;sam,dwayne,brenda=admins;joe=analysts</value> </
param> </provider>
```

8. Configuring Service Level Authorization

Note

Group membership is determined by the `identity-assertion parametergroup.principal.mapping`.

Group membership is determined by the `identity-assertion parametergroup.principal.mapping`.

8.1. Setting Up an Authorization Provider

The `ACLAuthz` provider determines who is able to access a service through the Knox Gateway by comparing the authenticated user, group, and originating IP address of the request to the rules defined in the authorization provider.

Configure the AclsAuthz provider as follows:

1. Open the cluster topology descriptor file, `$cluster-name .xml`, in a text editor.
2. Add a `AclsAuthz` authorization provider to `topology/gateway` with a parameter for each service as follows:

   ```
   <provider> <role>authorization</role> <name>AclsAuthz</name>
   <enabled>true</enabled> <param> <name> $service_name .acl.mode</
   name> <value> $mode </value> </param> <param> <name>
   $service_Name .acl</name> <value> $cluster_users ;
   $groups_field ; $IP_field </value> </param> ... </provider>
   ```

   where:

   - `$service_name` matches the name of a service element. For example,`webhdfs`.
   - `$mode` determines how the identity context (the effective user, their associated groups, and the original IP address) is evaluated against the fields as follows:
     - AND specifies that the request must match an entry in all three fields of the corresponding `$service_name .acl` parameter.
     - OR specifies that the request only needs to match an entry in any field, `$users_field` OR `$groups_field`, OR `$IP_field`.

     Note

     The `$service_name .acl.mode` parameter is optional. When it is not defined, the default mode is AND ; therefore requests to that service must match all three fields.
   - `$cluster_users` is a comma-separated list of authenticated users. Use a wildcard (*) to match all users.

- `$groups_field` is a comma-separated list of groups. Use a wildcard (*) to match all groups.
- `$IP_field` is a comma-separated list of IPv4 or IPv6 addresses. An IP address in the list can contain wildcard at the end to indicate a subnet (for example: 192.168.*). Use a wildcard (*) to match all addresses.

3. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway/data/deployments`.

8.2. Examples of Authorization

The following examples illustrate how to define authorization rule types to restrict access to requests matching:

- **Only users in a specific group and from specific IP addresses**

  The following rule is restrictive. It only allows the guest user in the admin group to access WebHDFS from a system with the IP address of either 127.0.0.2 or 127.0.0.3:

  ```
  <provider> <role>authorization</role> <name>AclsAuthz</name>
  <enabled>true</enabled> <param> <name>webhdfs.acl</name>
  <value>guest;admin;127.0.0.2,127.0.0.3</value> </param> </
  provider>
  ```

  When the parameter `acl.mode` is not defined the default behavior is ALL, therefore following rule is the same as the one above:

  ```
  <provider> <role>authorization</role> <name>AclsAuthz</name>
  <enabled>true</enabled> <param> <name>webhdfs.acl.mode</name>
  <value>AND</value> </param> <param> <name>webhdfs.acl</name>
  <value>guest;admin;127.0.0.2,127.0.0.3</value> </param> </
  provider>
  ```

  Note

  If Guest is not in the admin group, the request is denied.

- **Two of the three conditions**

  The following rule demonstrates how to require two conditions, user and group but not IP address, using the Wildcard. The rule allows the guest user that belongs to the admin group to send requests from anywhere because the IP field contains an asterisk which matches all IP addresses:

  ```
  <provider> <role>authorization</role> <name>AclsAuthz</name>
  <enabled>true</enabled> <param> <name>webhdfs.acl</name>
  <value>guest;admin;*</value> </param> </provider>
  ```

- **One of the three conditions**

When the `$service .acl.mode` parameter is set to OR, the request only needs to match one entry in any of the fields. The request fails with HTTP Status 403 unauthorized, if no conditions are met.

The following example allows:

- `guest` to send requests to WebHDFS from anywhere.
- Any user in the admin group to send requests to WebHDFS from anywhere.
- Any user, in any group, to send a request to WebHDFS from 127.0.0.2 or 127.0.0.3.

```
<provider> <role>authorization</role> <name>AclsAuthz</name>
<enabled>true</enabled> <param> <name>webhdfs.acl.mode</name>
<value>OR</value> </param> <param> <name>webhdfs.acl</name>
<value>guest;admin;127.0.0.2,127.0.0.3</value> </param> </
provider>
```

- **Allow all requests**

  The following rule grants all users, in any group, and from any IP addresses to access WebHDFS:

  Note

  When a wildcard is used in a field it matches any value. Therefore the Allow all requests example is the same as not defining an ACL.

```
<provider> <role>authorization</role> <name>AclsAuthz</name>
<enabled>true</enabled> <param> <name>webhdfs.acl</name>
<value>*;*;*</value> </param> </provider>
```

9. Audit Gateway Activity

The Knox Gateway Audit Facility tracks actions that are executed by Knox Gateway per user request or that are produced by Knox Gateway internal events, such as topology deployments.

Tip

The Knox Audit module is based on the Apache log4j. You can customize the logger by changing the log4j.appender.auditfile.Layout property in `$gateway /conf/gateway-log4j.properties` to another class that extends Log4j. For detailed information see Apache's log4j.

9.1. Audit Log Fields

Auditing events on the gateway are informational, the default auditing level is informational (INFO) and it cannot be changed.

The Audit logs located at `C:/hadoop/logs/knox/gateway-audit.log` have the following structure:

EVENT_PUBLISHING_TIMEROOT_REQUEST_ID | PARENT_REQUEST_ID | REQUEST_ID | LOGGER_NAME | TARGET_SERVICE_NAME | USER_NAME | PROXY_USER_NAME | SYSTEM_USER_NAME | ACTION | RESOURCE_TYPE | RESOURCE_NAME | OUTCOME | LOGGING_MESSAGE

where:

- EVENT_PUBLISHING_TIME : contains the timestamp when record was written.
- ROOT_REQUEST_ID : Reserved, the field is empty.
- PARENT_REQUEST_ID : Reserved, the field is empty.
- REQUEST_ID : contains a unique value representing the request.
- LOGGER_NAME : contains the logger name. For example `audit`.
- TARGET_SERVICE_NAME : contains the name of Hadoop service. Empty indicates that the audit record is not linked to a Hadoop service. For example, an audit record for topology deployment.
- USER_NAME : contains the ID of the user who initiated session with Knox Gateway.
- PROXY_USER_NAME : contains the authenticated user name.
- SYSTEM_USER_NAME : Reserved, field is empty.
- ACTION : contains the executed action type. The value is either authentication, authorization, redeploy, deploy, undeploy, identity-mapping, dispatch, or access.
- RESOURCE_TYPE contains the resource type of the action. The value is either `uri`, `topology`, or `principal`.

- RESOURCE_NAME : contains the process name of the resource. For example, `topology` shows the inbound or dispatch request path and `principal` shows the name of mapped user.
- OUTCOME contains the action results, `success`, `failure`, or `unavailable`.
- LOGGING_MESSAGE contains additional tracking information, such as the HTTP status code.

9.2. Change Roll Frequency of the Audit Log

Audit records are written to the log file */var/log/knox/gateway-audit.log* and by default roll monthly. When the log rolls, the date that it rolled is appended to the end of the current log file and a new one is created.

To change the frequency:

1. Open the `$gateway /conf/gateway-log4j.properties` file in a text editor.
2. Change the `log4j.appender.auditfile.DatePattern` as follows:

   ```
   log4j.appender.auditfile.DatePattern = $interval
   ```

   where `$interval` is one of the following:

Setting	Description
yyyy-MM	Rollover at the beginning of each month
yyyy-ww	Rollover at the first day of each week. The first day of the week depends on the locale.
yyyy-MM-dd	Rollover at midnight each day.
yyyy-MM-dd-a	Rollover at midnight and midday of each day.
yyyy-MM-dd-HH	Rollover at the top of every hour.
yyyy-MM-dd-HH-mm	Rollover at the beginning of every minute.

   Tip

   For more examples, see Apache log4j: Class DailyRollingFileAppender.
3. Save the file.
4. Restart the gateway:

   ```
   cd $gateway bin/gateway.sh stop bin/gateway.sh start
   ```

10. Gateway Security

The Knox Gateway offers the following security features:

10.1. Implementing Web Application Security

The Knox Gateway is a Web API (REST) Gateway for Hadoop clusters. REST interactions are HTTP based, and therefore the interactions are vulnerable to a number of web application security vulnerabilities. The web application security provider allows you to configure protection filter plugins.

Note

The initial vulnerability protection filter is for Cross Site Request Forgery (CSRF). Others will be added in future releases.

10.2. Configuring Protection Filter Against Cross Site Request Forgery Attacks

A Cross Site Request Forgery (CSRF) attack attempts to force a user to execute functionality without their knowledge. Typically the attack is initiated by presenting the user with a link or image that when clicked invokes a request to another site with which the user already has an established an active session. CSRF is typically a browser based attack.

The only way to create a HTTP request from a browser is with a custom HTTP header is to use Javascript XMLHttpRequest or Flash, etc. Browsers have built-in security that prevent web sites from sending requests to each other unless specifically allowed by policy. This means that a website www.bad.com cannot send a request to http://bank.example.com with the custom header X-XSRF-Header unless they use a technology such as a XMLHttpRequest. That technology would prevent such a request from being made unless the bank.example.com domain specifically allowed it. This then results in a REST endpoint that can only be called via XMLHttpRequest (or similar technology).

Note

After enabling CSRF protection within the gateway, a custom header is required for all clients that interact with it, not just browsers.

To add a CSRF protection filter:

1. Open the cluster topology descriptor file, `$cluster-name .xml`, in a text editor.
2. Add a `WebAppSec` webappsec provider to `topology/gateway` with a parameter for each service as follows:

```
<provider> <role>webappsec</role> <name>WebAppSec</name>
<enabled>true</enabled> <param> <name>csrf.enabled</name>
<value> $csrf_enabled </value> </param> <param> <!-- Optional --
> <name>csrf.customHeader</name> <value> $header_name </value>
</param> <param> <!-- Optional --> <name>csrf.methodsToIgnore</
name> <value> $HTTP_methods </value> </param> </provider>
```

where:

- `$csrf_enabled` is either true or false.
- `$header_name` when the optional parameter csrf.customHeader is present the value contains the name of the header that determines if the request is from a trusted source. The default, X-XSRF-Header, is described by the NSA in its guidelines for dealing with CSRF in REST.

`$http_methods` when the optional parameter `csrf.methodsToIgnore` is present the value enumerates the HTTP methods to allow without the custom HTTP header. The possible values are GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, or PATCH. For example, specifying GET allows GET requests from the address bar of a browser. Only specify methods that adhere to REST principals in terms of being idempotent.

3. Save the file.

   The gateway creates a new WAR file with modified timestamp in `$gateway /data/deployments`.

10.3. Validate CSRF Filtering

The following curl command can be used to request a directory listing from HDFS while passing in the expected header X-XSRF-Header.

```
curl -k -i --header "X-XSRF-Header: valid" -v -u guest:guest-
password https://localhost:8443/gateway/sandbox/webhdfs/v1/tmp?
op=LISTSTATUS
```

Omitting the –header “X-XSRF-Header: valid” above results in an **HTTP 400 bad_request**. Disabling the provider, by setting csrf.enabled to false allows a request that is missing the header.

10.4. Configuring Knox With a Secured Hadoop Cluster

Once you have a Hadoop cluster that uses Kerberos for authentication, you must configure Knox to work with that cluster.

To enable the Knox Gateway to interact with a Kerberos-protected Hadoop cluster, add a knox user and Knox Gateway properties to the cluster.

Do the following:

1. Find the fully-qualified domain name of the host running the gateway:

   ```
   hostname -f
   ```

   If the Knox host does not have a static IP address, you can define the knox host as * for local developer testing.

2. At every Hadoop Master:

   - Create a UNIX account for Knox:

     ```
     useradd -g hadoop knox
     ```

   - Edit `core-site.xml` to include the following lines (near the end of the file):

     ```
     <property>
      <name>hadoop.proxyuser.knox.groups</name>
      <value>users</value>
     </property>

     <property>
      <name>hadoop.proxyuser.knox.hosts</name>
      <value>$knox-host</value>
     </property>
     ```

     where `$knox-host` is the fully-qualified domain name of the host running the gateway.

   - Edit `webhcat-site.xml` to include the following lines (near the end of the file):

     ```
     <property>
      <name>hadoop.proxyuser.knox.groups</name>
      <value>users</value>
     </property>

     <property>
      <name>hadoop.proxyuser.knox.hosts</name>
      <value>$knox-host</value>
     </property>
     ```

     where `$knox_host` is the fully-qualified domain name of the host running the gateway.

3. At the Oozie host, edit `oozie-site.xml` to include the following lines (near the end of the file):

   ```
   <property>
    <name>oozie.service.ProxyUserService.proxyuser.knox.groups</name>
    <value>users</value>
   </property>

   <property>
    <name>oozie.service.ProxyUserService.proxyuser.knox.hosts</name>
    <value>$knox-host</value>
   </property>
   ```

where `$knox-host` is the fully-qualified domain name of the host running the gateway.

4. At each node running HiveServer2, edit `hive-site.xml` to include the following properties and values:

```
<property>
 <name>hive.server2.enable.doAs</name>
 <value>true</value>
</property>

<property>
 <name>hive.server2.allow.user.substitution</name>
 <value>true</value>
</property>

<property>
 <name>hive.server2.transport.mode</name>
 <value>http</value>
 <description>Server transport mode. "binary" or "http".</description>
</property>

<property>
 <name>hive.server2.thrift.http.port</name>
 <value>10001</value>
 <description>Port number when in HTTP mode.</description>
</property>

<property>
 <name>hive.server2.thrift.http.path</name>
 <value>cliservice</value>
 s<description>Path component of URL endpoint when in HTTP mode.</
description>
</property>
```

10.5. Configure Wire Encryption

For the simplest of evaluation deployments, the initial startup of the Knox Gateway will generate a self-signed cert for use on the same machine as the gateway instance. These certificates are issued for "localhost" and will require specifically disabling hostname verification on client machines other than where the gateway is running.

10.6. Self-Signed Certificate with Specific Hostname for Evaluations

In order to continue to use self-signed certificates for larger evaluation deployments, a certificate can be generated for a specific hostname. This will allow clients to properly verify the hostname presented in the certificate as the host that they requested in the request URL.

To create a self-signed certificate:

1. Create a certificate: where `$gateway-hostname` is the FQDN of the Knox Gateway.

```
cd $gateway bin/knoxcli.cmd create-cert --hostname $gateway-
hostname
```

2. Export the certificate in PEM format:

```
keytool -export -alias gateway-identity -rfc -file
$certificate_path -keystore $gateway /data/security/keystores/
gateway.jks
```

Note

cURL option accepts certificates in PEM format only.

3. Restart the gateway:

```
cd $gateway bin/gateway.sh stop bin/gateway.sh start
```

4. After copying the certificate to a client, use the following command to verify:

```
curl --cacert $certificate_path -u $username : $password https://
 $gateway-hostname : $gateway_port /gateway/ $cluster_name /webhdfs/v1?op=
GETHOMEDIRECTORY
```

10.7. CA-Signed Certificates for Production

For production deployments or any deployment in which a certificate authority issued certificate is needed, the following steps are required.

1. Import the desired certificate/key pair into a java keystore using keytool and ensure the following:
   - The certificate alias is gateway-identity.
   - The store password matches the master secret created earlier.
   - Note the key password used - as we need to create an alias for this password.
2. Add a password alias for the key password:

```
cd $gateway bin/knoxcli.cmd create-cert create-alias gateway-
identity-passphrase --value $actualpassphrase
```

Note

The password alias must be "gateway-identity-passphrase".

10.8. Setting Up Trust for the Knox Gateway Clients

In order for clients to trust the certificates presented to them by the gateway, they will need to be present in the client's truststore as follows:

1. Export the gateway-identity cert from the `$gateway /data/security/keystores/gateway.jks` using java keytool or another key management tool.

2. Add the exported certificate to the cacerts or other client specific truststore or the `gateway.jks` file can be copied to the clients to be used as the truststore.

Note

If taking this approach be sure to change the password of the copy so that it no longer matches the master secret used to protect server side artifacts.

11. Setting Up Knox for WebHDFS HA

This chapter describes how to set up the Knox Gateway for WebHDFS HA (high availability).

To set up Knox for WebHDFS HA:

11.1. Configure WebHDFS for Knox

REST API access to HDFS in a Hadoop cluster is provided by WebHDFS. The WebHDFS REST API documentation is available online. The following properties for Knox WebHDFS must be enabled in the `/etc/hadoop/conf/hdfs-site.xml` configuration file. The example values shown in these properties are from an installed instance of the Hortonworks Sandbox.

```
<property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
</property>
<property>
    <name>dfs.namenode.rpc-address</name>
    <value>sandbox.hortonworks.com:8020</value>
</property>
<property>
    <name>dfs.namenode.http-address</name>
    <value>sandbox.hortonworks.com:50070</value>
</property>
<property>
    <name>dfs.https.namenode.https-address</name>
    <value>sandbox.hortonworks.com:50470</value>
</property>
```

The values above must be reflected in each topology descriptor file deployed to the gateway. The gateway by default includes a sample topology descriptor file located at `{GATEWAY_HOME}/deployments/sandbox.xml`. The values in the following sample are also configured to work with an installed Hortonworks Sandbox VM.

```
<service>
    <role>NAMENODE</role>
    <url>hdfs://localhost:8020</url>
</service>
<service>
    <role>WEBHDFS</role>
    <url>http://localhost:50070/webhdfs</url>
</service>
```

The URL provided for the NAMENODE role does not result in an endpoint being exposed by the gateway. This information is only required so that other URLs can be rewritten that reference the Name Node’s RPC address. This prevents clients from needing to be aware of the internal cluster details.

11.2. Configure Knox for WebHDFS HA

Important

Before you can configure the Knox Gateway for WebHDFS HA (high availability), you must first configure WebHDFS for Knox.

Knox provides basic failover and retry functionality for REST API calls made to WebHDFS when HDFS HA has been configured and enabled.

To enable HA functionality for WebHDFS in Knox the following configuration must be added to the topology file.

```
<provider>
   <role>ha</role>
   <name>HaProvider</name>
   <enabled>true</enabled>
   <param>
       <name>WEBHDFS</name>
       <value>maxFailoverAttempts=3;failoverSleep=1000;maxRetryAttempts=
300;retrySleep=1000;enabled=true</value>
   </param>
</provider>
```

The <role> and <name> of the provider must be as shown above. The <name> in the <param> section must match that of the service role name that is being configured for HA, and the <value> in the <param> section is the configuration for that particular service in HA mode. In this case the <name> is `WEBHDFS`.

The various configuration parameters are described below:

- `maxFailoverAttempts` – The maximum number of times a failover will be attempted. The current failover strategy is very simplistic in that the next URL in the list of URLs provided for the service is used, and the one that failed is put at the bottom of the list. If the list is exhausted and the maximum number of attempts has not been reached, the first URL that failed will be tried again (the list will start again from the original top entry).
- `failoverSleep` – The amount of time in milliseconds that the process will wait or sleep before attempting to failover.
- `maxRetryAttempts` – The maximum number of times that a retry request will be attempted. Unlike failover, the retry is done on the same URL that failed. This is a special case in HDFS when the node is in safe mode. The expectation is that the node will come out of safe mode, so a retry is desirable here as opposed to a failover.
- `retrySleep` – The amount of time in milliseconds that the process will wait or sleep before a retry is issued.
- `enabled` - Flag to turn the particular service on or off for HA.

For the service configuration itself, the additional URLs for standby nodes should be added to the list. The active URL (at the time of configuration) should ideally be added at the top of the list. For example:

```
<service>
    <role>WEBHDFS</role>
    <url>http://{host1}:50070/webhdfs</url>
    <url>http://{host2}:50070/webhdfs</url>
</service>
```

12. About Hortonworks Data Platform

Copyright

© Copyright © 2012 - 2015 Hortonworks, Inc. Some rights reserved.

This work by Hortonworks, Inc. is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner enablement services. **All of our technology is, and will remain, free and open source.**

For more information on Hortonworks technology, Please visit the Hortonworks Data Platform page. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.