Hortonworks Data Platform

Apache Knox Gateway Administrator Guide

(May 22, 2014)

docs.hortonworks.com

Hortonworks Data Platform : Apache Knox Gateway Administrator Guide

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.

Table of Contents

List of Tables

1. Knox Gateway Overview

The Apache Knox Gateway is a REST API gateway for interacting with Apache Hadoop clusters. The gateway provides a single access point for all REST interactions with Hadoop clusters.

In this capacity, the Apache Knox Gateway is able to provide valuable functionality to aid in the control, integration, monitoring and automation of critical administrative and analytical needs of the enterprise.

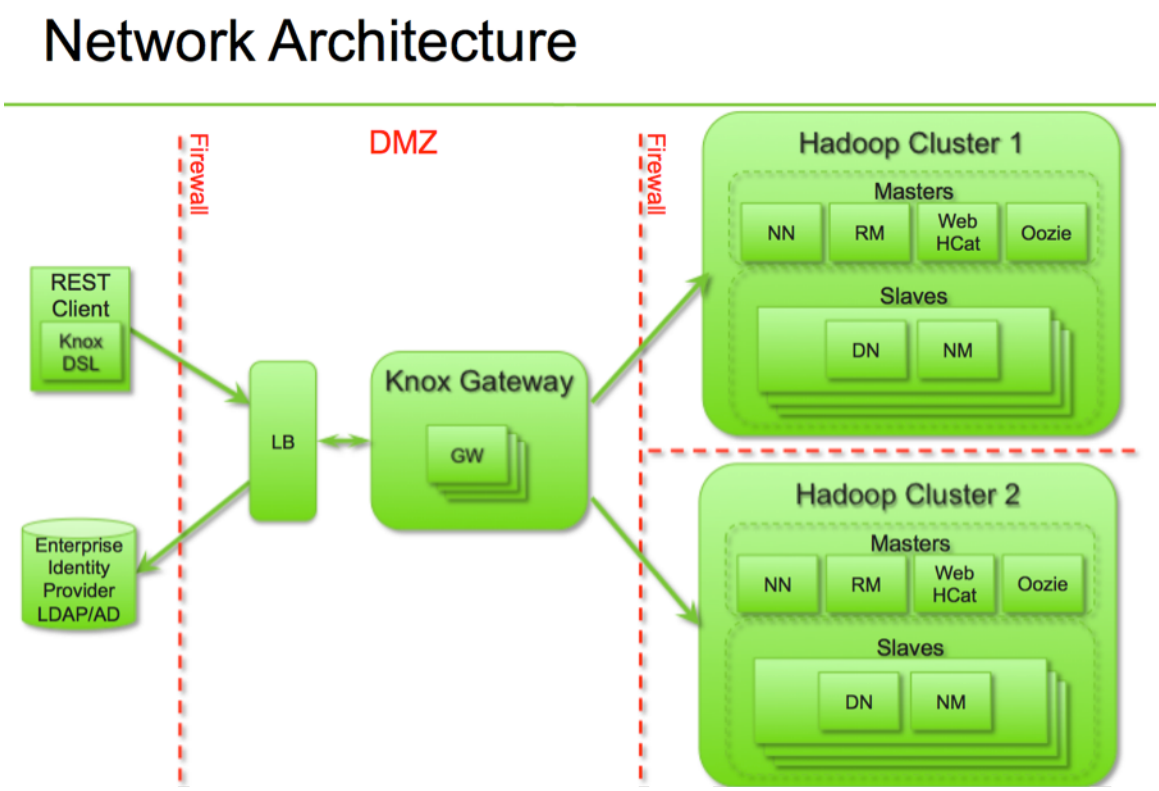
- Authentication (LDAP and Active Directory Authentication Provider)
- Federation/SSO (HTTP Header Based Identity Federation)
- Authorization (Service Level Authorization) Auditing

While there are a number of benefits for unsecured Hadoop clusters, the Apache Knox Gateway also complements the Kerberos secured cluster quite nicely. Coupled with proper network isolation of a Kerberos secured Hadoop cluster, the Apache Knox Gateway provides the enterprise with a solution that:

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

1.1. Knox Gateway Network Architecture

All external users and applications access the cluster through the Apache Knox Gateway URL. The following diagram shows how Apache Knox fits in a Hadoop deployment.



1.1.1. 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
WebHDFS	2.4.0
WebHCat/Templeton	0.13.0
	0.12.0
Oozie[a]	4.0.0
HBase/Stargate	0.98
Hive (via WebHCat)	0.13
Hive (via JDBC)	0.13

[a]Multinode Linux Clusters only.

2. Configure the Knox Gateway

This section explains the following gateway global settings:

- Secure the Gateway Directories
- Customize the Gateway Port and Path
- Manage the Master Secret
- Manually Redeploy Cluster Topologies
- Starting and Stopping Knox

2.1. Secure the Gateway Directories

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

- `/usr/lib/knox` is the *`$gateway_home`* directory, see table below.
- `/var/log/knox` contains the output files from the Knox Gateway.
- `/var/run/knox` contains the Process ID (PID) for the currently running Knox Gateway.

The table below describes the files and directories in *`$gateway_home`*:

Table 2.1. Gateway Home Directory Contents

Directory/file name	Description
`conf`	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 clusters, see Configure Cluster Topologies.
`lib`	Contains the JARs for all the components that make up the gateway.
`dep`	Contains the JARs for all of the components upon which the gateway depends.
`ext`	A directory where user supplied extension JARs can be placed to extends the gateways 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 information 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 incubation at Apache.

Hortonworks recommends creating a specific user, such as `knox`, to run and secure the gateway. The following outlines the recommended ownership when running the Knox Gateway:

- `/usr/lib/knox` (*`$gateway_home`*): Contains the installed application files (i.e. binaries). Owned by root and read-only for others.
- `/etc/knox`: Contains the gateway configuration files. Owned by root and read-only for others.
- `/var/lib/knox/data`: Contains gateway security, cluster deployments, and auditing information; the gateway writes to this directory at runtime. Owned, writable and readable by the `knox` user.
- `/var/log/knox`: Contains gateway log files. Owned and writableby the `knox` user and read-only for others.
- `/var/run/knox`: Contains the gateway PID files. Owned and writableby the `knox` user and read-only for others.

Note

For instructions on setting up security, such as Kerberos and SSL configuration, see Configuring Knox Gateway Security.

2.2. Customize the Gateway Port and Path

This section explains how to configure the global Knox Gateway properties. The gateway properties effects 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 the `gateway-site.xml` and modify the following properties:

```
<property>
  <name>gateway.port</name>
  <value>$gateway_port</value>
  <description>The HTTP port for the Gateway.</description>
</property>

<property>
   <name>gateway.path</name>
   <value>$gateway_path</value>
   <description>The default context path for the gateway.</description>
</property>
```

   where:

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

2. Restart the gateway:

```
su -l knox -c '$gateway_home/bin/gateway.sh stop'
su -l knox -c '$gateway_home/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.

2.3.1. Setting the Master Secret

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

To set the master secret, run the following command:

```
su -l knox -c '$gateway_home/bin/knoxcli.sh 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 and where possible the file permissions are set to be accessible by the knox user only.

Warning

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

2.3.2. Change the Master Secret

To change the master secret, run the following command: -

```
su -l knox -c '$gateway_home/bin/knoxcli.sh create-master  --force'
```

Warning

When changing the master secret and there is an existing keystore, you must also update the keystore.

2.4. Manually Redeploy Cluster Topologies

When the gateway properties or gateway-wide settings are changed, you must manually redeploy the clusters. Redeploy cluster topologies after changing the following:

- Time settings on the gateway host
- Implementing or updating Kerberos
- Implementing or updating SSL certificates

- Changing a cluster alias

Tip

Updating cluster properties does not require you to manually redeploy clusters. Cluster topology descriptor files are in the `/etc/knox/conf/topologies` directory and the corresponding deployment is in `/var/lib/knox/data/deployments` . The gateway monitors the topology directory for changes. If the descriptor changes or a new one is added, the gateway automatically redeploys the cluster.

2.4.1. Redeploy all Clusters

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

To redeploy all Cluster Topologies:

1. Verify the timestamp on the currently deployed clusters.

```
ls -lh /var/lib/knox/data/deployments
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 ad.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 dynamicgroup.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 myCluster.war.14505eeb448
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 myOtherCluster.war.14505ea9980
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth127sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth254sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 sandbox.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 staticgroup.war.14505e7e618
```

2. Redeploy all clusters.

```
$gateway_home/bin/knoxcli.sh redeploy
```

3. Verify that a new cluster WAR was created.

```
ls -lh /var/lib/knox/data/deployments
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 ad.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 ad.war.145062ab3a8
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 dynamicgroup.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 dynamicgroup.war.145062ab3a8
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 myCluster.war.14505eeb448
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 myCluster.war.145062ab3a8
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 myOtherCluster.war.14505ea9980
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:11 myOtherCluster.war.1450606f350
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth127sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 preauth127sla.war.145062ab3a8
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth254sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 preauth254sla.war.145062ab3a8
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 sandbox.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 sandbox.war.145062ab3a8
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 staticgroup.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 staticgroup.war.145062ab3a8
```

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

2.4.2. Redeploy Specific Clusters

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

To redeploy a specific cluster:

1. Verify the timestamp on the currently deployed Cluster Topology WAR files.

```
ls -lh /var/lib/knox/data/deployments
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 ad.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 dynamicgroup.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 myCluster.war.14505eeb448
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 myOtherCluster.war.14505ea9980
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth127sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth254sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 sandbox.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 staticgroup.war.14505e7e618
```

2. Redeploy a specific cluster. -

```
$gateway_home/bin/knoxcli.sh redeploy --cluster $cluster_name
```

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

3. Verify that the cluster was deployed.

```
ls -lh /var/lib/knox/data/deployments
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 ad.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 dynamicgroup.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 myCluster.war.14505eeb448
drwxrwxr-x 3 knox knox 4.0K Mar 28 00:50 myCluster.war.145062ab3a8
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth127sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 preauth254sla.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 sandbox.war.14505e7e618
drwxrwxr-x 3 knox knox 4.0K Mar 27 23:54 staticgroup.war.14505e7e618
```

A new file with the current timestamp is created.

2.5. Manually start and stop Knox

Any configuration change to the Knox Gateway global settings in `/etc/knox/conf/gateway-site.xml` requires a gateway service restart before the change is loaded.

2.5.1. Manually start Knox

To start the gateway that was stopped with the gateway script, use the following command:

```
su -l knox -c '$gateway_home/bin/gateway.sh start'
```

2.5.2. Manually start the after an Unclean Shutdown

To start the gateway that was not stopped using the gateway script, use the clean switch as follows:

```
su -l knox -c '$gateway_home/bin/gateway.sh clean'
su -l knox -c '$gateway_home/bin/gateway.sh start'
```

Warning

The clean switch clears all `.out` and `.err` files in the logs directory as well as the PID in `/var/run/knox` directory .

2.5.3. Manually stop Knox

To stop the gateway, use the following command:

```
su -l knox -c '$gateway_home/bin/gateway.sh stop'
```

3. Define Cluster Topology

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

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 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 mapped external name when hostname is embedded in a response from the cluster, see Map Hadoop Cluster Host Names.
	authentication	Integrates an LDAP store to authenticate external requests accessing the cluster via the Knox Gateway, see Set up LDAP Authentication.
	federation	Defines HTTP header authentication fields for an SSO or federation solution provider, see Set up HTTP Header Authentication for Federation/SSO.
	identity-assertion	Maps external authenticated users to an internal cluster user that the gateway asserts as the current session user or group, see Configure Identity Assertion.
	authorization	Service level authorization that restricts cluster access to specified users, groups and/or IP addresses, see Configure Service Level Authorization.
	webappsec	Configures a web application security plug-in that provides protection filtering against Cross Site Request Forgery attacks, see Configure Web Application Security.
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, see Configure Hadoop Service URLs.

Note

The gateway automatically redeploys the cluster when a new topology descriptor file or change is detected.

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. Configure the 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 covers the following topics:

- Set up Hadoop Service URLs
- Example of Service Definitions
- Validate Service Connectivity

4.1. Set up Hadoop Service URLs

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

1. Add the following `service` for each Hadoop service to the `/etc/knox/conf/topologies` *`$cluster-name`*`.xml`:

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

   where:

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

2. Save the file.

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

4.2. Example of Service Definitions

Configure each service that you want to expose externally. 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>WEBHDFS</role>
        <url>http://webhdfs-host: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>
```

Note

Define the internal hostname and service ports of your cluster.

4.3. Validate Service Connectivity

Use following commands test connectivity between the gateway host and the Hadoop service, 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.

- To test WebHDFS by getting the home directory:
  - Enter the following command on the gateway host:

```
curl http://$webhdfs-host:50070/webhdfs/v1?op=GETHOMEDIRECTORY
{"Path":"/user/gopher"}
```

- Enter the following command on an external client:

```
curl https://$gateway-
host:$gateway_port/$gateway_path/$cluster_name/$webhdfs_service_name/v1?
op=GETHOMEDIRECTORY
{"Path":"/user/gopher"}
```

- To test WebHCat/Templeton by getting the version:
  - Enter the following command on the gateway host:

```
curl http://$webhdfs-host:50070/templeton/v1/version
{"supportedVersions":["v1"],"version":"v1"}
```

  - Enter the following command on an external client:

```
curl https://$gateway-
host:$gateway_port/$gateway_path/$cluster_name/$webhcat_service_name/v1/
version
{"supportedVersions":["v1"],"version":"v1"}
```

- To test Oozie by getting the version:
  - Enter the following command on the gateway host:

```
curl http://$oozie-host:11000/oozie/v1/admin/build-version
{"buildVersion":"4.0.0.2.1.1.0-302"}
```

  - Enter the following command on an external client:

```
curl https://$gateway-
host:$gateway_port/$gateway_path/$cluster_name/$oozie_service_name/v1/
admin/build-version
{"buildVersion":"4.0.0.2.1.1.0-302"}
```

- To test HBase/Stargate by getting the version:
  - Enter the following command on the gateway host:

```
curl http://$hbase-host:17000/version
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
```

  - Enter the following command on an external client:

```
curl https://$gateway-
host:$gateway_port/$gateway_path/$cluster_name/$hbase_service_name/version
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
```

- To test HiveServer2:

Note

Both of the following URL return an authentication error.

  - Enter the following command on the gateway host:

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

- Enter the following command on an external client:

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

5. Map the Internal Nodes to External URLs

Hostmapping is an advanced configuration topic and is only required in very specific types of deployments. These deployment scenarios generally involve the use of virtualized environments as are 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 from the outside world. These details will include ip addresses and/or hostnames. The virtualized environment will expose additional ip addresses and/or hostnames for use by clients accessing the cluster from outside of the virtualized environment. In these cases, 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.

Where the difficulties from these virtualized environments arise are cases where the Hadoop cluster redirects client requests to other nodes within the cluster and indicates the internal hostname locations rather than those exposed for external access. 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 is 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. Set up a Hostmap Provider

Add tthe `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:

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

   The gateway creates a new WAR file with modified timestamp in `/var/lib/knox/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. Enable 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 `/etc/knox/conf/ gateway-log4j.properties` file.
2. Change ERROR to DEBUG on the following line:

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

3. Restart the gateway:

```
su -l knox -c '$gateway_home/bin/gateway.sh stop'
su -l knox -c '$gateway_home/bin/gateway.sh start'
```

6. Configure Authentication

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

- Set up LDAP Authentication
- Set up Federation/SSO Authentication

Note

An identity assertion provider is required when authentication is configured, see Configuring Service Users and Groups.

6.1. Set 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 the following variables are specific to your environment:

- *$USER_DN* is a comma separated list of attribute and value pairs that define the User Distinguished Name (DN). The first pair must be set to "*$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 `/var/lib/knox/data/deployments` .

6.1.1. 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`.

The following is an example provider configuration for Active Directory (AD):

```
<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>
```

6.1.2. Example of an OpenLDAP Configuration

The following is an example provider configuration for OpenLDAP:

```
<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>uid={0},ou=people,dc=hadoop,dc=apache,dc=org</value>
    </param>
    <param>
        <name>main.ldapRealm.contextFactory.url</name>
        <value>ldap://localhost:33389</value>
    </param>
    <param>
        <name>main.ldapRealm.contextFactory.authenticationMechanism</name>
        <value>simple</value>
    </param>
    <param>
        <name>urls./**</name>
        <value>authcBasic</value>
    </param>
</provider>
```

6.1.3. Testing an LDAP Provider

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

1. Open the command line on an external client, note that 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/namenode/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.

6.2. Set 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 suggestions on product specific solutions, see the Apache Knox Gateway User 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.

Warning

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* 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.
- *$group_field* (Optional) 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 `/var/lib/knox/data/deployments` .

6.2.1. 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>
```

6.2.2. Testing an 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. Configure Identity Assertion

The Knox Gateway `identity-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 effective user.
   - **False**: The authenticated user is asserted, that is the effective user is the same as the authenticated user.
2. Does the effective user match any group mapping rules:
   - **True**: The effective user is a member of all matching groups (for the purpose of authorization).
   - **False**: The effective user is not a member of any mapped groups.

Note

When authenticated by an SSO provider, the effective 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 an `identity-assertion` provider in the `topology/gateway` definition. For the minimal requirements, see Set up Basic Identity-Assertion.

The following is the complete structure of the `identity-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:

- *$user_ids* is a comma separated list of external users or the wildcard (*) indicates all users.
- *$cluster_user* the Hadoop cluster user name the gateway asserts, that is the effective user name.

Note

Note that identity-assertion rules are not required, see Set up Basic Identity Assertion for details. However, whenever an authentication provider is configured an `identity-assertion` provider is also required.

7.2. Set up Basic Identity Assertion

When you define the `Pseudo identity-assertion` provider without parameters, the authenticated user is asserted as the effective 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 `Pseudo identity-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 `/var/lib/knox/data/deployments` .

7.3. Map Effective User to Cluster User

The `principal.mapping` parameter of an `identity-assertion` provider determines the user name that the gateway asserts (uses as the effective 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`* the Hadoop cluster user name the gateway asserts, that is the effective user name.

3. Save the file.

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

7.3.1. Example of 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 *guest*, the gateway asserts the user as *sam* and all other users as *dwayne*.

```
<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>
```

7.4. Map Effective 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 *effective 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.

7.4.1. Configure Group Mappings

To map effective 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:

   - *`$cluster_users`* is a comma separated list of effective user or the wildcard (*) indicating all users.
   - *`$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 `/var/lib/knox/data/deployments` .

7.4.2. Examples of Group Mapping

The following example indicates that all (*) users are members of the “users” group and that users sam, dwayne, and brenda are all members of the admins group and only joe is in the analysts group.

```
<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>
```

Important

For additional examples of user and group mappings, see the Apache Knox User Guide.

8. Configure Service Level Authorization

Use Knox Gateway ACLs to restrict access to Hadoop services.

Note

Group membership is determined by the `identity-assertion` parameter `group.principal.mapping`, see Mapping Users to Groups and/or by the group field of an SSO authenticated request, see Setting up a Federation Provider for SSO.

8.1. Set up an Authorization Provider

The ACLAuthz provider determines who is able to access a service through the Knox Gateway by comparing the effective 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 the how the request 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.

- *$cluster_users* is a comma separated list of effective users. Use a wildcard (*) to match all users.
- *$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 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 /var/lib/knox/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 then 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:

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

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.

9. Audit Gateway Actitivity

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 `/etc/knox/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 `/var/log/knox/gateway-audit.log` have the following structure:

```
EVENT_PUBLISHING_TIME ROOT_REQUEST_ID|PARENT_REQUEST_ID|REQUEST_ID|LOGGER_NAME|TARGET_SERVIC
```

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 effective 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 `/etc/knox/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:

Table 9.1. Audit Log Settings: Roll Frequency

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:

```
su -l knox -c '$gateway_home/bin/gateway.sh stop'
su -l knox -c '$gateway_home/bin/gateway.sh start'
```

10. Gateway Security

The Knox Gateway offers the following security features:

- Implement Web Application Security
- Configure Knox with a Secured Hadoop Cluster
- Configure Wire Encryption (SSL)

10.1. Implement 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.

10.1.1. Configure 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 builtin 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 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 `/var/lib/knox/data/deployments` .

10.1.2. 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.

10.2. Configure Knox with a Secured Hadoop Cluster

Once you have a Hadoop cluster that is using Kerberos for authentication, you have to do the following to configure Knox to work with that cluster.

10.2.1. Configure Knox Gateway on the Hadoop Cluster

To allow the Knox Gateway to interact with a Keberos protected Hadoop cluster, add a knox user and Knox Gateway properties to the cluster.

1. On *every* Hadoop Master perform the following commands:

   a. Create Unix account for Knox:

   ```
   useradd -g hadoop knox
   ```

   b. Add the following lines to the `core-site.xml` on each master node 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.

   Note

   You can usually find this by running hostname -f. You can define the knox host as * for local developer testing if Knox host does not have static IP.

   c. Add the following lines to the `webhcat-site.xml` on each master node towards 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.

Note

You can usually find this by running hostname -f. You can define the knox host as * for local developer testing if Knox host does not have static IP.

2. On the Oozie host, add the following lines to the `oozie-site.xml` 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.

Note

You can usually find this by running hostname -f on that host. You could use * for local developer testing if Knox host does not have static IP.

3. On the nodes runnin HiveServer2, add the following properties to the `hive-site.xml`:

```
<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>
 <description>Path component of URL endpoint when in HTTP mode.</
description>
</property>
```

Note

Some of the properties may already be in the `hive-site.xml`. Ensure that the values match the ones above.

10.2.2. Add Knox Principal to KDC

On the KDC, create a Kerberos principal keytab for Knox as follows:

1. SSH to the KDC host.
2. Execute `kadmin.local` to open an interactive session:

```
kadmin.local
```

3. Add a key for knox with the following commands:

```
add_principal -randkey knox/knox@EXAMPLE.COM
ktadd -k /etc/security/keytabs/knox.service.keytab -norandkey knox/$knox-host@EXAMPLE.COM
```

where: *$knox-host* is the fully qualified domain name of the Knox Gateway and *EXAMPLE.COM* is the name of your KDC Realm.

4. Close the interactive session:

```
exit
```

10.2.3. Configure Knox Gateway for Keberos

After preparing the cluster and creating a keytab for Knox, perform the following procedure to complete the configuration.

1. Copy the Knox keytab to Knox host.
2. Add unix account for the knox user on Knox host as follows:

```
useradd -g hadoop knox
```

Copy knox.service.keytab created on KDC host on to the Knox host `/etc/knox/conf/knox.service.keytab`.

3. Change the owner of the file to the knox user and set the premissions as follows:

```
chown knox knox.service.keytab
chmod 400 knox.service.keytab
```

4. Update krb5.conf at `/etc/knox/conf/krb5.conf` on Knox host.

Tip

You can also copy the *$gateway_home*`/templates/krb5.conf` file provided in the Knox binary download and customize it to suit your cluster.

5. Update the `/etc/knox/conf/krb5JAASLogin.conf` on Knox host.

Tip

You can also copy the *`$gateway_home`*`/templates/krb5JAASLogin.conf` file provided in the Knox binary download and customize it to suit your cluster. Replace the *`$knox-host`* with the Knox Gateway FQDN and *`EXAMPLE.COM`* with the your KDC Realm.

6. Update *`$gateway_home`*`/conf/gateway-site.xml` on Knox host by changing the following value:

```
<property>
  <name>gateway.hadoop.kerberos.secured</name>
  <value>true</value>
  <description>Boolean flag indicating whether the Hadoop cluster protected
 by gateway is secured with Kerberos</description>
</property>
```

7. Restart Knox as follows:

```
su -l knox -c '$gateway_home/bin/gateway.sh stop'
su -l knox -c '$gateway_home/bin/gateway.sh start'
```

8. Redeploy the Cluster Topology as follows:

   a. Redeploy all Clusters using the following command:

   ```
   $gateway_home/bin/knoxcli.sh redeploy
   ```

   b. Verify that a new Cluster Topology WAR was created with the following command:

   ```
   ls -lh /var/lib/knox/data/deployments
   ```

   A new file for each with the same timestamp is created.

Note

After you do the above configurations and restart Knox, Knox uses SPNego to authenticate with Hadoop services and Oozie. There is no change in the way you make calls to gateway whether you use cURL or Knox DSL.

10.3. Configure Wire Encryption (SSL)

10.3.1. Using Self-Signed Certificate for Evaluations

10.3.1.1. Self-signed Localhost Certificate for Evaluations

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.

10.3.1.2. 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:

```
$GATEWAY_HOME/bin/knoxcli.sh create-cert --hostname $gateway-hostname
```

   where *$gateway-hostname* is the FQDN of the Knox Gateway.

2. Export the certificate in PEM format:

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

Note

cURL option accepts certificates in PEM format only.

3. Restart the gateway:

```
su -l knox -c "$GATEWAY_HOME/bin/gateway.sh stop"
su -l knox -c "$GATEWAY_HOME/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
```

10.3.2. 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:

```
$GATEWAY_HOME/bin/knoxcli.sh create-alias gateway-identity-passphrase --
value $actualpassphrase
```

Note

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

10.3.3. Set 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_HOME*/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.