KnoxToken Sessions with KnoxShell in Apache Knox 0.12.0

The 0.12.0 release of Apache Knox had a focus on the KnoxShell module of the product. This module as been getting some uptake recently and a number of improvements were made in its security, API classes, credential collectors and even structure and packaging.

The KnoxShell release artifact provides a small footprint client environment that removes all unnecessary server dependencies, configuration, binary scripts, etc. It is comprised of a couple different things that empower different sorts of users.

- A set of SDK type classes for providing access to Hadoop resources over HTTP
- A Groovy based DSL for scripting access to Hadoop resources based on the underlying SDK classes
- Token based Sessions for KnoxShell to provide a CLI SSO session for executing multiple scripts

This article will go over the use of the KnoxToken service and KnoxShell from download to actively scripting in a few minutes.

This particular article will introduce the use of KnoxToken Sessions and Service and therefore requires the 0.12.0 release of Apache Knox.

KnoxToken Sessions Overview

A few words on KnoxToken Sessions with the KnoxShell before we get into installation and configuration.

The overall goal of KnoxToken Sessions is to provide an SSO experience for commandline access to KnoxShell scripts and shell without requiring a challenge for credentials with every interaction or script execution.

Much the same way that kerberos provides SSO for interacting with kerberized services, KnoxToken Sessions require an 'init' to login, provides a 'list' to show details of the current session and 'destroy' to logout and invalidate the session.

At the heart of the session is a token that must be acquired from a configured KnoxToken service within a given Knox instance topology.

The KnoxToken service is a simple REST API for acquiring tokens that can be used to represent the same authentication event until it expires or is explicitly invalidated.

The token response is stored in a file permissions protected file in the user's home directory. The file name is '.knoxtokencache'.

Download

In the 0.12.0 release, you may get to the knoxshell download through the Apache Knox site.

From this above page click the Gateway client binary archive link or just use the one here.

Unzip this file into your preferred location which will result in a knoxshell-0.12.0 directory and we will refer to that location as the {GATEWAY_HOME}.

CD {GATEWAY_HOME}

You should see something similar to the following:

```
home:knoxshell-0.12.0 larry$ ls -l

total 296

-rw-r--r--@  1 larry  staff  71714 Mar 14 14:06 LICENSE
-rw-r--r--@  1 larry  staff    164 Mar 14 14:06 NOTICE
-rw-r--r--@  1 larry  staff  71714 Mar 15 20:04 README
drwxr-xr-x@ 12 larry  staff    408 Mar 15 21:24 bin
drwxr--r--@  3 larry  staff    102 Mar 14 14:06 conf
drwxr-xr-x+  3 larry  staff    102 Mar 15 12:41 logs
drwxr-xr-x@ 18 larry  staff    612 Mar 14 14:18 samples
```

Directory	Description
bin	contains the main knoxshell jar and related shell scripts
conf	only contains log4j config
logs	contains the knoxshell.log file
samples	has numerous examples to help you get started

Setup Truststore for Client

Get/setup truststore for the target Knox instance or fronting load balancer

- if you have access to the server you may use the command
    - `knoxcli.sh export-cert –type JKS`
- copy the resulting gateway-client-identity.jks to your user home directory
- you may also ask your Knox administrator to provide you with the public cert for the gateway and create your own truststore within your user home directory

NOTE: if you see errors related to SSL and PKIX your truststore is not properly setup

Configure the KnoxToken Service in a Gateway Topology

```
<service>
 <role>KNOXTOKEN</role>
 <param>
   <name>knox.token.ttl</name>
   <value>300000000</value>
 </param>
 <param>
   <name>knox.token.audiences</name>
   <value>tokenbased</value>
 </param>
 <param>
   <name>knox.token.target.url</name>
   <value>https://localhost:8443/gateway/tokenbased</value>
 </param>

</service>
```

Parameter	Description	Default
knox.token.ttl	This indicates the lifespan of the token. Once it expires a new token must be acquired from KnoxToken service. T his is in milliseconds. The 36000000 in the topology above gives you 10 hrs.	30000 That is 30 seconds.
knox.token. audiences	This is a comma separated list of audiences to add to the JWT token. This is used to ensure that a token received by a participating application knows that the token was intended for use with that application. It is optional. In the event that an endpoint has expected audiences and they are not present the token must be rejected. In the event where the token has audiences and the endpoint has none expected then the token is accepted.	empty
knox.token.target. url	This is an optional configuration parameter to indicate the intended endpoint for which the token may be used. The KnoxShell token credential collector can pull this URL from a knoxtokencache file to be used in scripts. This eliminates the need to prompt for or hardcode endpoints in your scripts.	n/a

The above service descriptor element should be put into a topology such as sandbox.xml that is protected by the ShiroProvider or some other provider that will authenticate

a user based on HTTP Basic Auth.

Establishing a KnoxToken Session for KnoxShell

```
bash-3.2$ bin/knoxshell.sh init https://localhost:8443/gateway/sandbox
Enter username: guest
Enter password:
log4j:WARN No appenders could be found for logger (org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
knoxinit successful!
Token Type: Bearer
Expires On: 04/18/2017 03:12:19
Target URL: https://localhost:8443/gateway/tokenbased
```

Let's take a look at the resulting .knoxtokencache file:

```
bash-3.2$ cat ~/.knoxtokencache
{"access_token":"eyJhbGciOiJSUzI1NiJ9.
eyJzdWIiOiJndWVzdCIsImF1ZCI6InRva2VuYmFzZWQiLCJpc3MiOiJLTk9YU1NPIiwiZXhwIjoxNDkyNDgyODU3fQ.

kt7EhtK_Xm7Nmuohvceu4Uxjw36_CI6rW7UMaQ4iyBOpuMqm8V9SUVO5TEOmuCXIHr50Z_EC-CsvN9Ghk_LAacZnSP5-
c3oVJKgojIf7WBTC2GkR7K5Gd6OHNXzyvH0M2

TGxmEaeR9P8yuw4c3gX8dD6RtWG_f07Nco-_GxC8n4",

"target_url":"https://localhost:8443/gateway/tokenbased",

"token_type":"Bearer ",

"expires_in":1492482857955}
```

As you can see above, the .knoxtokencache file contains the full JSON response from the KnoxToken service.

The following JSON elements are always present:

Element	Description
access_token	The actual JWT token to be used to access resources through the Knox Gateway.
token_type	The intended use for the token. At this type, it is always 'Bearer'.
expires_in	The expiration date of the token.

In addition, the following optional elements may be configured on the KnoxToken side:

Element	Description
`knox.token.target.url`	This is an optional configuration parameter to indicate the intended endpoint for which the token may be used. The KnoxShell token credential collector can pull this URL from a knoxtokencache file to be used in scripts. This eliminates the need to prompt for or hardcode endpoints in your scripts.

Execute a Sample Script

Here is a look at a simple ls script:

```
import groovy.json.JsonSlurper
import java.util.HashMap
import java.util.Map
import org.apache.knox.gateway.shell.Credentials
import org.apache.knox.gateway.shell.Hadoop
import org.apache.knox.gateway.shell.hdfs.Hdfs

credentials = new Credentials()
credentials.add("KnoxToken", "none: ", "token")
credentials.collect()

token = credentials.get("token").string()

gateway = credentials.get("token").getTargetUrl()

println ""
println "********************************GATEWAY INSTANCE*************************************"
println gateway
println "***************************************************************************************"
println ""

headers = new HashMap()
headers.put("Authorization", "Bearer " + token)

session = Hadoop.login( gateway, headers )

if (args.length > 0) {
  dir = args[0]
} else {
  dir = "/"
}

text = Hdfs.ls( session ).dir( dir ).now().string
json = (new JsonSlurper()).parseText( text )
statuses = json.get("FileStatuses");

println statuses

session.shutdown()
```

Some things to note about this sample:

1. the gateway URL is not hardcoded it is acquired from the .knoxtokencache file along with the token to use
    - alternatives would be to hardcode it, pass it as an argument to the script, use an environment variable or prompt for it with a ClearInput credential collector
2. credential collectors are used to gather credentials or other input from various sources. In this sample the KnoxToken collector is used to collect the token from the file permission protected file.
3. The Hadoop.login method establishes a login session of sorts which will need to be provided to the various API classes as an argument.
4. the response text is easily retrieved as a string and can be parsed by the JsonSlurper or whatever you like

Execute the ls example script from the {GATEWAY_CLIENT_HOME} directory - for instance:

bin/knoxshell.sh ls

```
bash-3.2$ bin/knoxshell.sh ls /
```

```
****************************GATEWAY INSTANCE**********************************
https://localhost:8443/gateway/tokenbased
*******************************************************************************

[FileStatus:[[accessTime:0, blockSize:0, childrenNum:0, fileId:16392, group:hadoop, length:0, modificationTime:1490985341750, owner:yarn, pathSuffix:app-logs, permission:777, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:2, fileId:16389, group:hadoop, length:0, modificationTime:1490985337320, owner:yarn, pathSuffix:ats, permission:755, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:1, fileId:16463, group:goodloans, length:0, modificationTime:1490996647104, owner:loanscore, pathSuffix:goodloans, permission:777, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:1, fileId:16399, group:hdfs, length:0, modificationTime:1490985346256, owner:hdfs, pathSuffix:hdp, permission:755, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:1, fileId:16465, group:loanscore, length:0, modificationTime:1490996650722, owner:loanscore, pathSuffix:loanscore, permission:755, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:1, fileId:16395, group:hdfs, length:0, modificationTime:1490985343509, owner:mapred, pathSuffix:mapred, permission:755, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:1, fileId:16397, group:hadoop, length:0, modificationTime:1490985343886, owner:mapred, pathSuffix:mr-history, permission:777, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:2, fileId:16386, group:hdfs, length:0, modificationTime:1490991689460, owner:hdfs, pathSuffix:tmp, permission:777, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:1, fileId:16461, group:unwise, length:0, modificationTime:1490996632817, owner:loanscore, pathSuffix:unwise, permission:777, replication:0, storagePolicy:0, type:DIRECTORY], [accessTime:0, blockSize:0, childrenNum:3, fileId:16387, group:hdfs, length:0, modificationTime:1492000099227, owner:hdfs, pathSuffix:user, permission:755, replication:0, storagePolicy:0, type:DIRECTORY]]]
```

Invalidate the KnoxToken Session

In order to invalidate the KnoxToken session so that no one can leverage your previous login, execute the KnoxShell destroy command.

```
bash-3.2$ bin/knoxshell.sh destroy
bash-3.2$
```

Try and rerun the ls script and note that you must re-init since the session is no longer valid:

```
bash-3.2$ bin/knoxshell.sh bin/ls /
Cached knox token cannot be found. Please login through knoxinit.
```