Winter Ready 2019 Technical Lab
Level: 300
Contact: omidm

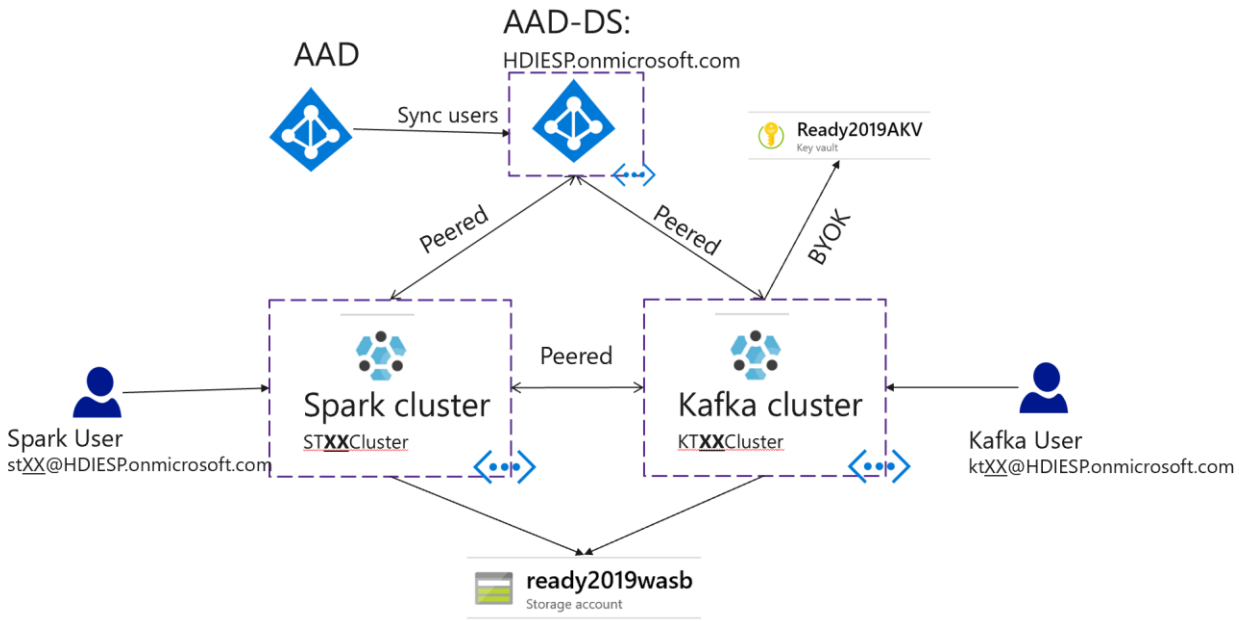
Team XX ==

Spark User Instructions

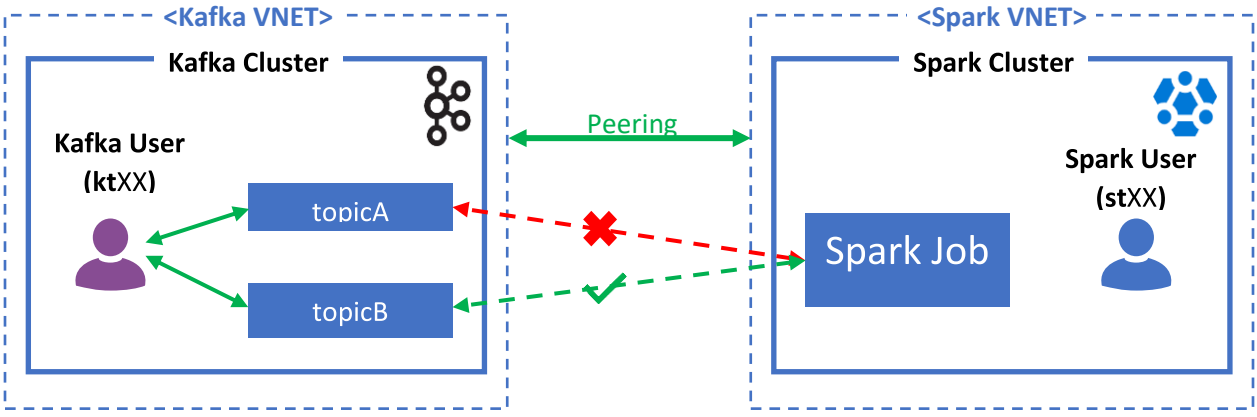
Overview:

This Lab walks through the setup of a typical streaming data pattern in Azure. Data is ingested in Apache Kafka on HDInsight into specific topics. This data in Kafka is then read by a Spark streaming job. What makes this pattern more relevant to enterprises is that both clusters have Enterprise Security Package enabled: hence access to Kafka topics can be gated by Apache Ranger policies.

This is the architecture that you'll be creating:



Inside the Kafka cluster there will be two topics (topicA and topicB) to be created by Kafka User, and inside Spark cluster there will be a Spark streaming job that consume the data from Kafka using Kerberos authentication. All of this traffic will go through the two VNETs that are peered using private IPs.



Winter Ready 2019 Technical Lab
Level: 300
Contact: omidm

Team XX ==

Spark User Instructions

You are the **spark user** in this diagram. Your teammate is the Kafka user. The rest of this document covers the necessary steps for creating a secure Spark cluster, executing a spark job that tries to read from both topics in the secure Kafka cluster using domain credentials.

You will go through the following scenarios:

1. Consume data from "topicA" with your domain credentials.
    a. Expected behavior: The spark job should fail with "Not authorized to access topics: ……." error message. The Ranger audit records in Kafka cluster shows that the access is denied.
2. Consume data from "topicB" with your domain credentials.
    a. Expected behavior: The spark job would run successfully. The Ranger audit records in Kafka cluster would show that the access is allowed.

Pre-requisites:

- AAD-DS is already setup. The lab instructor is the AAD admin and can provide access when/if needed. Think of the instructor as the AD team which is typically different from the big data team in enterprises. The AD team has already created your domain credentials which you will use later in this lab.
- An ESP Kafka cluster deployed in a different VNET: Your teammate in this lab will be creating and preparing this cluster. When needed you can coordinate with him to get related information.

Common info you will need:

- Domain User Credentials: st**XX**@HDIESP.onmicrosoft.com , Pass: Ask Instructor.
- Replace **XX** in the rest of this doc with you team number at the top of this page. If you don't know your team number, please ask the instructor.

Instructions:

Login with your domain username in the common info above to https://portal.azure.com

We will use ARM templates to deploy resources needed in this Lab.

You can find the required templates on GitHub, use the "Deploy to Azure" button when needed:

https://github.com/tylerfox/HDInsight-ESP-Kafka-Spark

1- Create a VNET & Peer

- Go to GitHub link above and click on "template_vnet.json" Deploy to Azure button OR simply click here: 

- Resource Group: Team**XX**RG

Spark User Instructions

- VNET name: ST**XX**VNET
- Your IP range depends on your team number (XX in top of this doc). Let **Y = [(2 *XX) – 1]**, then your VNET IP range is 10.Y.0.0/16 and Subnet IP range is **10.Y.0.0/24**
- Set Custom DNS server to AAD-DS DNS servers: **10.0.0.5,10.0.0.4**
- Set AAD-DS Resource Group: **AADDS-RG**
- AAD-DS VNET name: **AD-VNET**
- Hit "Purchase" to start the deployment.

This deployment creates a VNET/Subnet with the specified IP range, peer it to the AAD-DS VNET and set custom DNS server settings.

2- Create an HDInsight ESP Spark Cluster

- After the step1 is successfully finished, Go to GitHub link above and click on "template_spark.json" Deploy to Azure button. 

- Resource Group: Team**XX**RG
- Name: ST**XX**Cluster
- Admin_SSH_username: **sshuser**
- sshuser Password: Pick a pass and don't forget it. You will need this pass later.
- WASB account name: **ready2019wasb**
- WASB RG: **Ready2019RG**
- Spark VNET name: from previous section (ST**XX**VNET)
- AAD-DS resource group: **AADDS-RG**
- AAD-DS domain name: **HDIESP.onmicrosoft.com**
- Cluster Admin AAD account: st**XX**@HDIESP.onmicrosoft.com
- Cluster access group name: Team**XX**AADGroup
- Managed Identity RG: **Ready2019RG**
- Managed Identity name: **Ready2019MI**

This deployment creates an ESP Spark cluster in your VNET and will take about 20-30 minutes. You can proceed to the next step while the deployment is in progress.

3- **Peer to Kafka VNET:**

- Before you proceed, check with your teammate or look in to your resource group to make sure that Kafka VNET with the name KT**XX**VNET is already created.
- Go to GitHub link above and click on "template_peer.json" Deploy to Azure button.

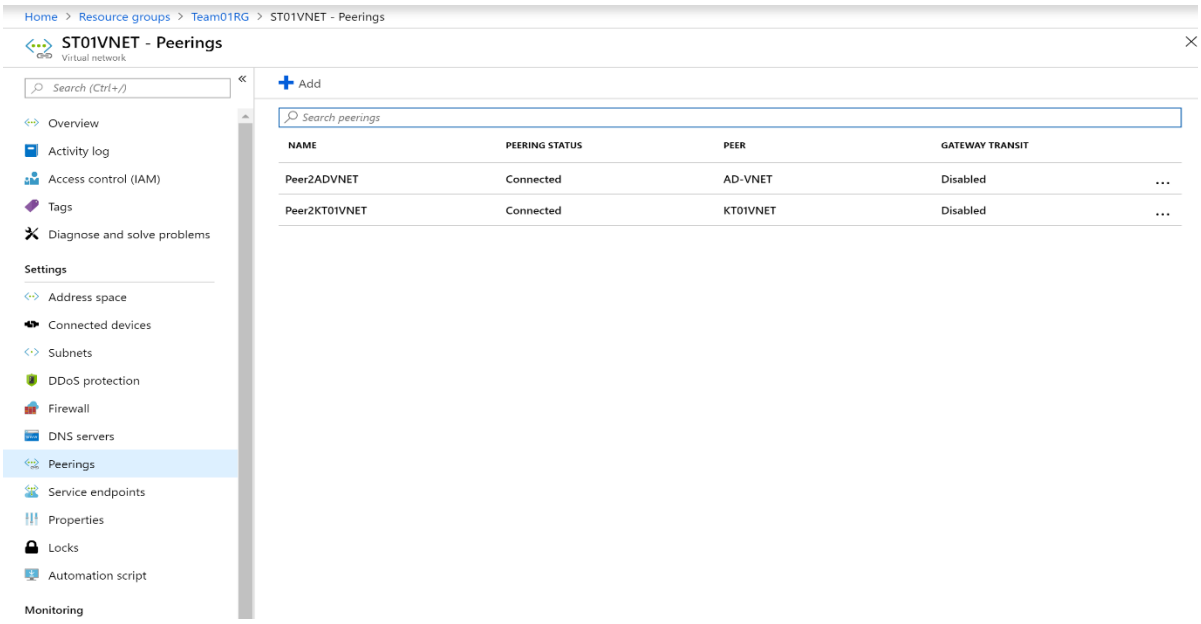

- My VNET: ST**XX**VNET
- Remote VNET: KT**XX**VNET

Winter Ready 2019 Technical Lab
Level: 300
Contact: omidm

Team XX ==

Spark User Instructions

This deployment peer your VNET to the Kafka cluster VNET. Your teammate should do the same and then the peering status will change from initiated to “Connected”:



4- Prepare the cluster environment:

While your cluster deployment is going on (takes 20-30 min), you can use a cluster that we have pre-created for you to complete the rest of this lab.

Your pre-created cluster name is: ST**XX**ACluster (replace XX with your team number) and the resources are created in the Team**XX**RG resource group.

- SSH to the cluster using the “sshuser” credentials you picked when creating the cluster. For pre-created clusters ask the instructor for sshuser password.
  - ssh sshuser@ST**XX**ACluster-ssh.azurehdinsight.net
    - Ask your instructor for the sshuser password on the pre-created cluster.
- Create a Kerberos keytab for your domain username. You can later use this keytab to authenticate to remote domain joined clusters without entering a password. Note the **UPPERCASE** in the domain name:
  - ktutil
  - ktutil: addent -password -p stXX@HDIESP.ONMICROSOFT.COM -k 1 -e RC4-HMAC
  - Password for st**XX**@HDIESP.ONMICROSOFT.COM: provided by lab instructors
  - ktutil: wkt st**XX**.keytab
  - ktutil: q

Winter Ready 2019 Technical Lab
Level: 300
Contact: omidm

Team XX ==

Spark User Instructions

- Create a st**XX**_jaas.conf file with the following content which will be used for authenticating to the Kafka:

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="./stXX.keytab"
  useTicketCache=false
  serviceName="kafka"
  principal="stXX@HDIESP.ONMICROSOFT.COM";
};
```

- Typically, data scientists will have their own logic for consuming the steaming data. In this lab we will follow the example instructions outlined here for reading the Kafka topic.
    - Get the spark-streaming jar ready by building you own. The sample jar used in this lab is located on Github. Please following these steps.
        a. sudo apt install maven
        b. git clone https://github.com/markgrover/spark-secure-kafka-app.git
        c. cd `spark-secure-kafka-app`
        d. mvn clean package
            a. the jar will be created in the "target" folder
- Test Keytab and Kerberos Authentications:
    i. cd
    ii. kinit st**XX**@HDIESP.ONMICROSOFT.COM -t st**XX**.keytab
    iii. klist - to make sure login was successful and domain user is available in the Cache. You should see an output like below:

```
sshuser@hn0-st01cl:~$ klist
Ticket cache: FILE:/tmp/krb5cc_2019
Default principal: st01@HDIESP.ONMICROSOFT.COM

Valid starting       Expires              Service principal
02/12/2019 02:38:44  02/12/2019 12:38:44  krbtgt/HDIESP.ONMICROSOFT.COM@HDIESP.ONMICROSOFT.COM
        renew until 02/19/2019 02:38:44
sshuser@hn0-st01cl:~$
```

Now your Spark cluster is ready to consume data from Kafka topics.

5- Consume data from Kafka topics:

Issue a spark-submit command to read from kafka topic "**topicA**" with st**XX** domain user keytab:

Example:

spark-submit --num-executors 1 --master yarn --deploy-mode cluster --packages org.apache.spark:spark-streaming-kafka-0-10_2.11:2.3.2.3.1.0.4-1 --repositories http://repo.hortonworks.com/content/repositories/releases/ --files st**XX**_jaas.conf#st**XX**_jaas.conf,st**XX**.keytab#st**XX**.keytab --driver-java-options "-

Winter Ready 2019 Technical Lab
Level: 300
Contact: omidm

Team XX ==

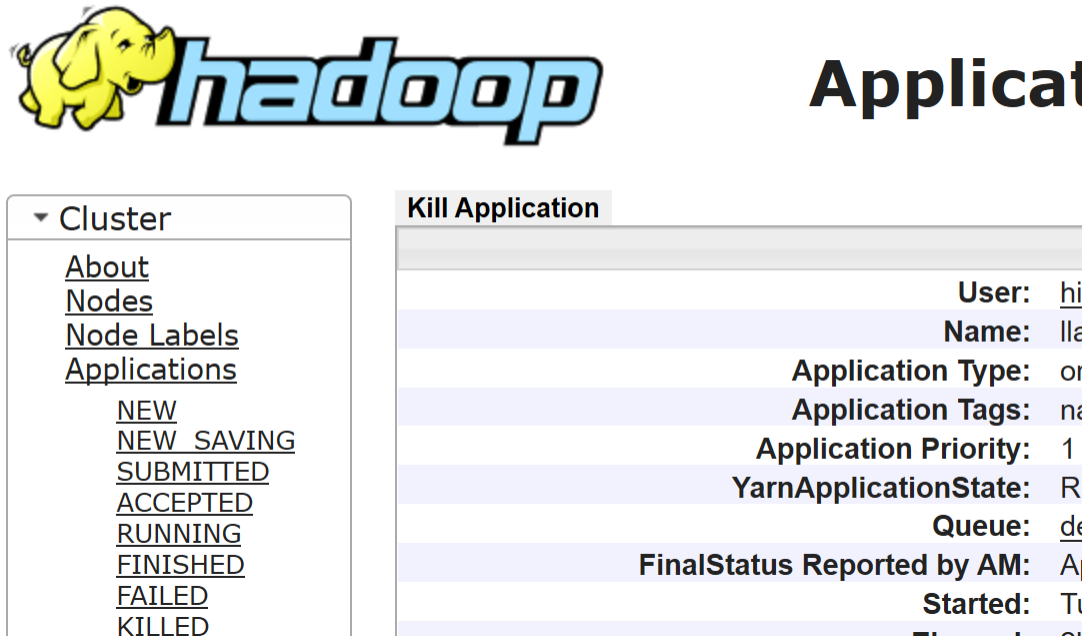
Spark User Instructions

Djava.security.auth.login.config=./st**XX**_jaas.conf" --class com.cloudera.spark.examples.DirectKafkaWordCount --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./st**XX**_jaas.conf" /home/sshuser/spark-secure-kafka-app/target/spark-secure-kafka-app-1.0-SNAPSHOT.jar **9._.0._**:9092 **topicA** false

9._.0._ is the IP address of one of the brokers in the Kafka cluster. You can ask your teammate to provide you with this IP address from /etc/hosts file on the Kafka cluster, look for worker nodes starting with wn0-*. The pre-created Kafka cluster name is KT**XX**ACluster.

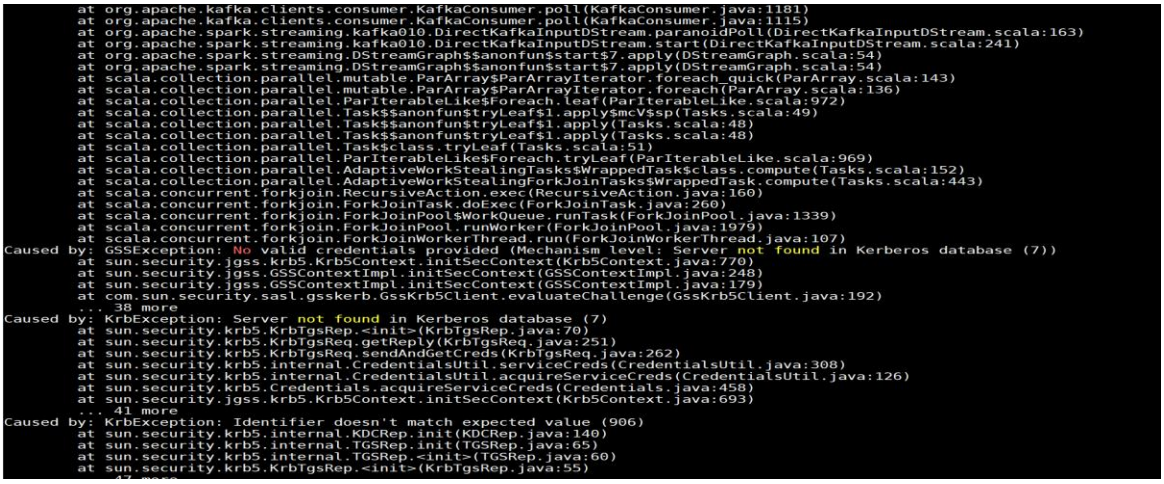
Troubleshooting:

If you make a mistake you can kill your running application: Go to Yarn UI to see your application and if needed "Kill Application": https://st**XX**cluster.azurehdinsight.net/yarnui/



If you see the following error message in the output of the spark submit job:
Caused by: GSSException: No valid credentials provided (**Mechanism level: Server not found in Kerberos database (7)**)

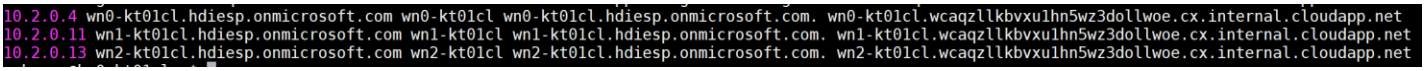
Spark User Instructions



This is due to a known DNS bug that is preventing the Kafka brokers FQDNs to be resolved to the right IP.

The workaround for this issue is to do the following:

- Ask your teammate to send you the list of worker nodes entries from Kafka cluster in "/etc/hosts". This list should look like this:



- Add this list to your "/etc/hosts" on both the head node and the worker node(s) of spark cluster:
  i. Edit /etc/hosts →sudo vim /etc/hosts → add the list at the end and save
  ii. ssh wn0-st**XX**ac (In this example the cluster has only one worker node)
    - The password is the same one you used for sshuser
  iii. sudo vim /etc/hosts → add the list at the end and save

Then try the spark-submit command again. This time you should see the "**TopicAuthorizationException**" which is expected and it means the Ranger policies on the Kafka cluster is enforced properly, since this user(st**XX**) is not allowed to read from "**topicA**".

> **spark-submit** --num-executors 1 --master yarn --deploy-mode cluster --packages org.apache.spark:spark-streaming-kafka-0-10_2.11:2.3.2.3.1.0.4-1 --repositories http://repo.hortonworks.com/content/repositories/releases/ --files st**XX**_jaas.conf#st**XX**_jaas.conf,st**XX**.keytab#st**XX**.keytab --driver-java-options "-Djava.security.auth.login.config=./st**XX**_jaas.conf" --class com.cloudera.spark.examples.DirectKafkaWordCount --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./st**XX**_jaas.conf" /home/sshuser/spark-secure-kafka-app/target/spark-secure-kafka-app-1.0-SNAPSHOT.jar **9._.0._**:9092 **topicA** false

Spark User Instructions

```
         tracking URL: http://hn0-st01cl.hdiesp.onmicrosoft.com:8088/proxy/application_1549484352974_0013/
         user: st01
19/02/07 02:28:01 INFO Client: Application report for application_1549484352974_0013 (state: RUNNING)
19/02/07 02:28:02 INFO Client: Application report for application_1549484352974_0013 (state: RUNNING)
19/02/07 02:28:03 INFO Client: Application report for application_1549484352974_0013 (state: RUNNING)
19/02/07 02:28:04 INFO Client: Application report for application_1549484352974_0013 (state: FINISHED)
19/02/07 02:28:04 INFO Client:
         client token: Token { kind: YARN_CLIENT_TOKEN, service:  }
         diagnostics: User class threw exception: org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [topicA]
        at ... run in separate thread using org.apache.spark.util.ThreadUtils ... ()
        at org.apache.spark.streaming.StreamingContext.liftedTree1$1(StreamingContext.scala:578)
        at org.apache.spark.streaming.StreamingContext.start(StreamingContext.scala:572)
        at com.cloudera.spark.examples.DirectKafkaWordCount$.main(DirectKafkaWordCount.scala:85)
        at com.cloudera.spark.examples.DirectKafkaWordCount.main(DirectKafkaWordCount.scala)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.spark.deploy.yarn.ApplicationMaster$$anon$4.run(ApplicationMaster.scala:721)

         ApplicationMaster host: 10.1.0.7
         ApplicationMaster RPC port: 0
         queue: default
         start time: 1549506432051
         final status: FAILED
         tracking URL: http://hn0-st01cl.hdiesp.onmicrosoft.com:8088/proxy/application_1549484352974_0013/
```

Now if you issue the same spark-submit command to read from "**topicB**":

> **spark-submit** --num-executors 1 --master yarn --deploy-mode cluster --packages org.apache.spark:spark-streaming-kafka-0-10_2.11:2.3.2.3.1.0.4-1 --repositories http://repo.hortonworks.com/content/repositories/releases/ --files st**XX**_jaas.conf#st**XX**_jaas.conf,st**XX**.keytab#st**XX**.keytab --driver-java-options "-Djava.security.auth.login.config=./st**XX**_jaas.conf" --class com.cloudera.spark.examples.DirectKafkaWordCount --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config=./st**XX**_jaas.conf" /home/sshuser/spark-secure-kafka-app/target/spark-secure-kafka-app-1.0-SNAPSHOT.jar **9._.0._**:9092 **topicB** false

NOTE: This is a streaming job that will continue running until you kill it from the YARN UI. Continue with the steps below while the job runs.

This operation is allowed based on the Ranger policies and should result in consuming the data from topicB and showing the word count as per our example logic.

While the spark streaming app is running, go to Yarn UI (https://stXXacluster.azurehdinsight.net/yarnui/)

→ Your application

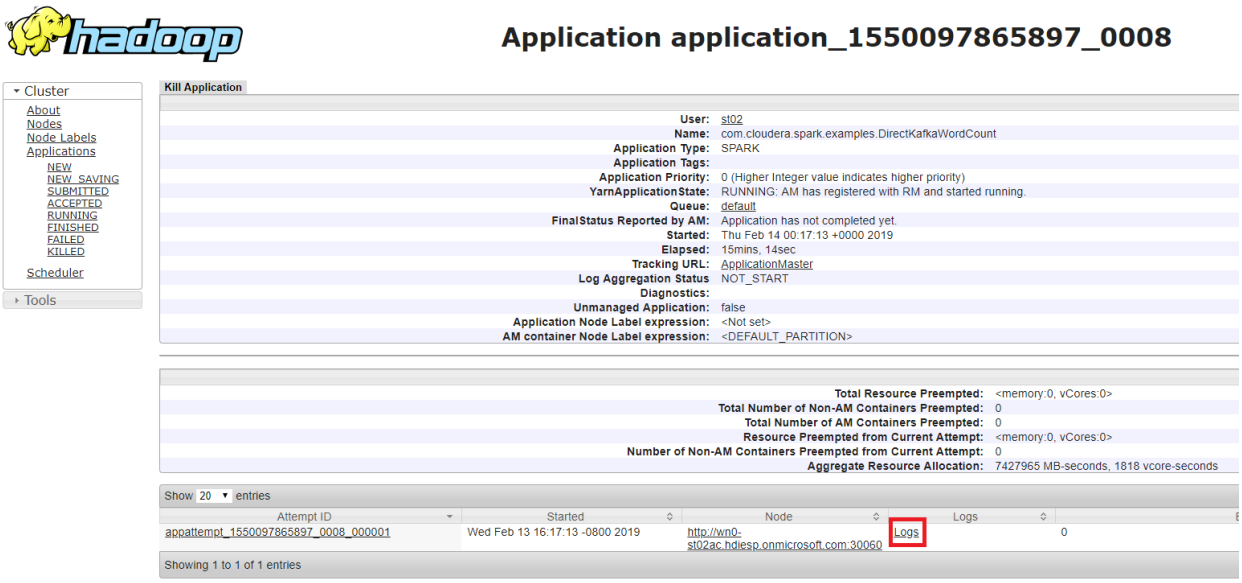
ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State
application_1550097865897_0008	st02	com.cloudera.spark.examples.DirectKafkaWordCount	SPARK	default	0	Wed Feb 13 16:17:13 -0800 2019	N/A	RUNNING

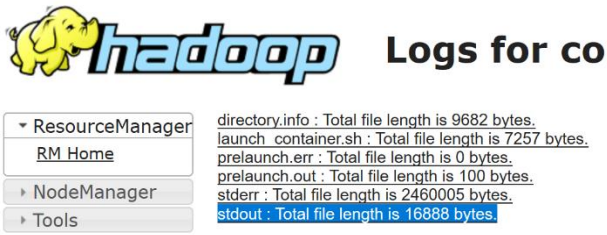
→ Logs and check the stdout

Winter Ready 2019 Technical Lab
Level: 300
Contact: omidm

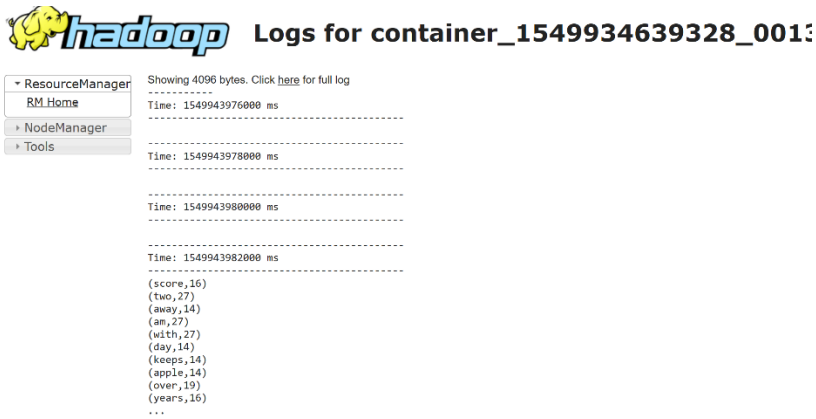
Team XX ==

Spark User Instructions



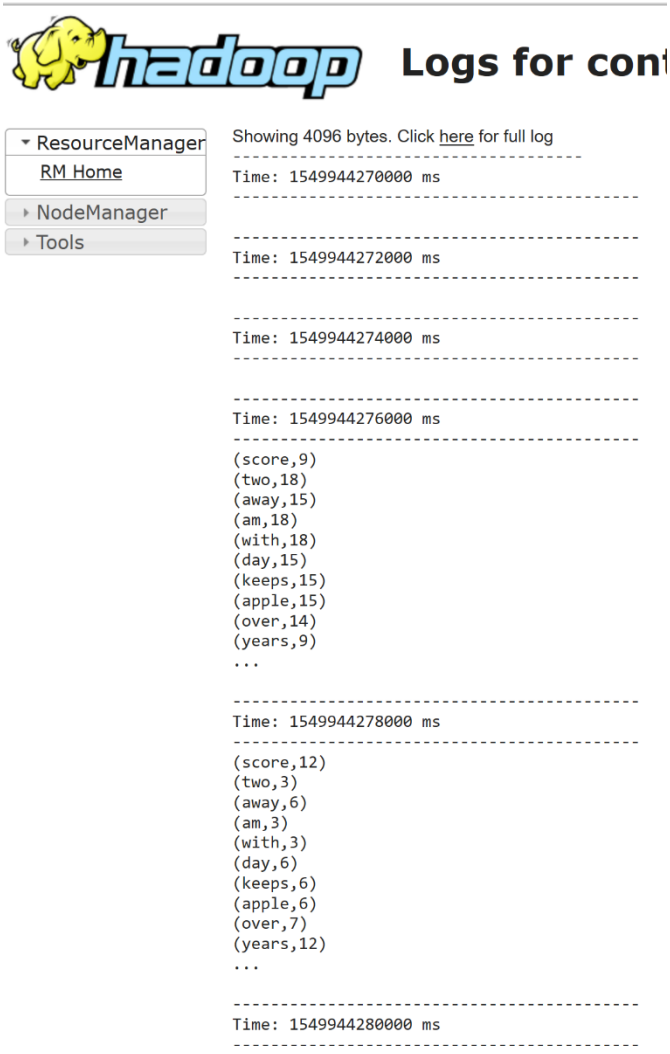


The word count result should look like below:



While the app is running, you can ask your teammate to submit more data to topicB using the java command and as the data coming in, refresh the browser to see the results in your logs output.

Spark User Instructions

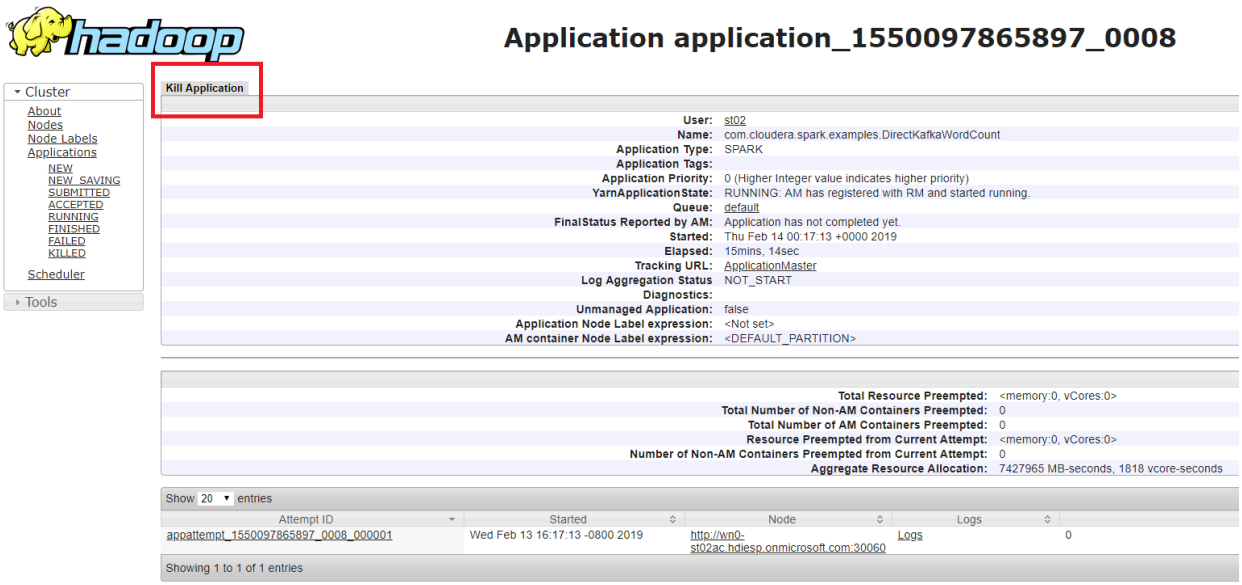


Finally, kill the spark streaming application from the YARN UI

Winter Ready 2019 Technical Lab
Level: 300
Contact: omidm

Team XX ==

Spark User Instructions



Congratulations! You have just created a secure streaming pipeline consisting of two Spark and Kafka clusters, deployed in a virtual network with Ranger policy enforcement for fine grained authorization and AAD-DS integration for authentication. This is the most secure Hadoop you can create in the cloud!