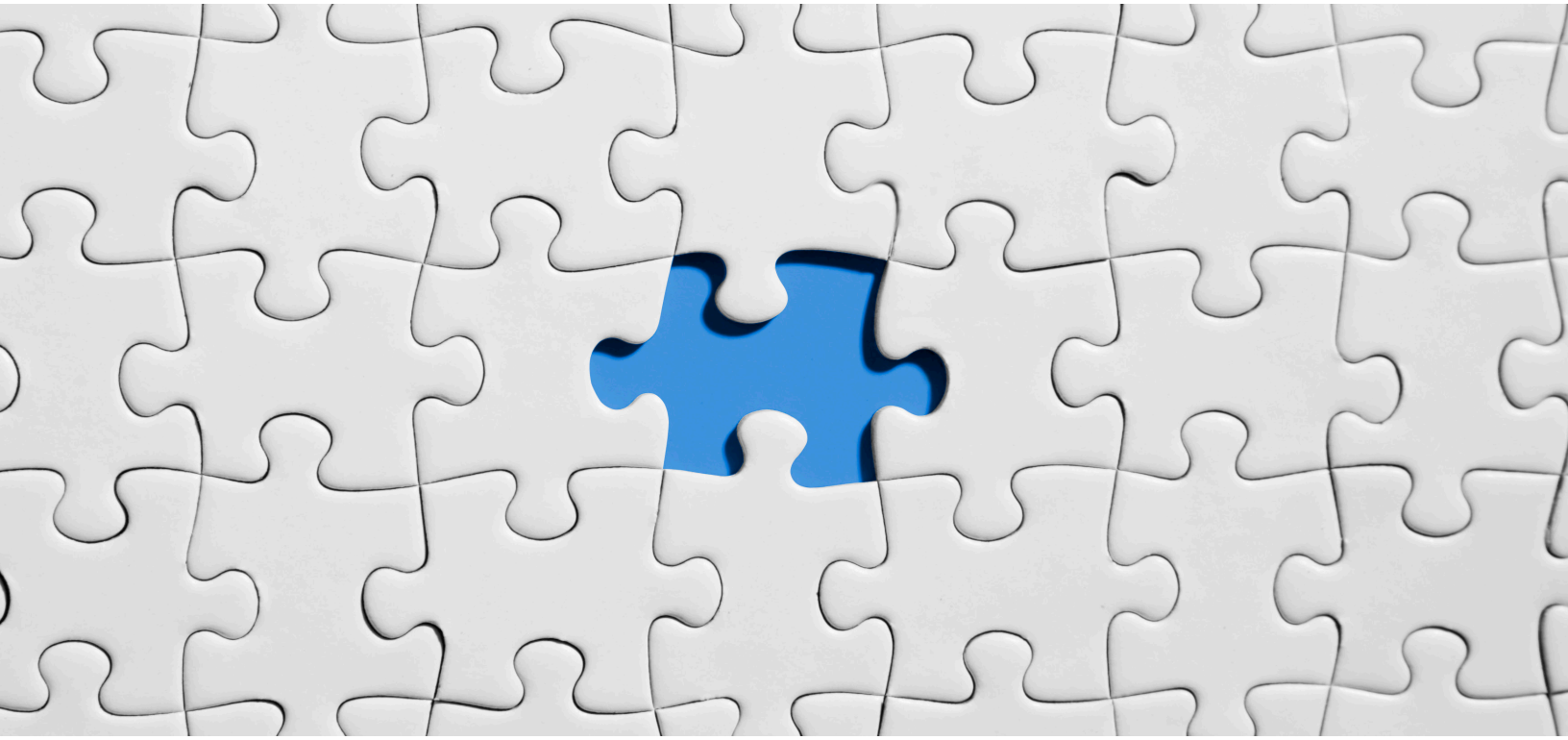
ECS, HORTONWORKS & KERBEROS: A BIG DATA SOLUTION



Cristina Alvarez

Denis Jannot

Knowledge Sharing Article

Table of Contents

Disclaimer: The views, processes or methodologies published in this article are those of the authors. They do not necessarily reflect Dell EMC's views, processes or methodologies.

1. Introduction

We are moving toward the fourth industrial revolution, in which mobile communications, social media and sensors are blurring the boundaries between people, the internet and the physical world. We live in the age of the data. In a broad range of application areas, data is being collected at an unprecedented scale. Decisions that previously were based on guesswork, or on painstakingly handcrafted models of reality, can now be made using data-driven mathematical models. While such Big Data analysis is clearly a new trend effective management and analysis of large-scale data poses an interesting but critical challenge.

Many IT companies have invested in Big Data products. Dell EMC is one such company that has put a lot of effort on this growing area, proposing alternatives to drive this Big Data challenge from the storage perspective.

Apache Hadoop has emerged as the preferred tool for performing powerful data analysis. A distributed computing ecosystem designed to process large amounts of data very efficiently, Hadoop includes a file-system and job engine, and is supported by an array of client interfaces. Traditional Hadoop analytics involves a series of complex workflows and data movement. Legacy Hadoop implementations consist on one or several ingestion clusters (usually NAS systems) and one or several compute clusters (Hadoop cluster). And, usually, customers have to select and copy the data between these two clusters to analyze it and extract results. This results in multiple copies of the data, extra network and storage resource consumption, complex data protection, delays, complex management, and scalability issues. But, what if the customer could analyze the data in-place in a system that already takes care of data protection and that is easily scalable? DellEMC Elastic Cloud Storage provides a solution for that, being Hadoop Distributed File System (HDFS) compatible and allowing in-place analytics, offering native replication mechanisms, geo data access, multiprotocol data access, high availability, and simplifying the Hadoop architecture making the solution easily scalable.

To offer an end-to end solution, Dell EMC went through the certification process of Elastic Cloud Storage (ECS) on Hortonworks (HWX), one of the Hadoop distribution market leaders, validating the integration of both platforms, thereby saving customers time to implement the solution, while providing them with an assurance of interoperability.

Additionally, most of the Hadoop implementations need a security mechanism to protect data access. Kerberos is one of the most commonly-used authentication protocols. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. But it is extremely difficult to configure for those who are not experts on this matter.

In conclusion, the integration of Hortonworks-ECS-Kerberos offers the end-to-end Hadoop solution that customers are seeking. However, it is still an emerging solution in the Dell EMC portfolio. This paper will augment existing documentation about how to configure, integrate, and test these three components.

This paper describes how to perform the complex configuration and integration among Kerberos, Hortonworks and ECS, how to run functional tests (including Distcp, Hive, Spark and HBase and frameworks, using ECS as internal or external file system in the Hadoop cluster), and how to run benchmarks that demonstrate that ECS is a good fit for Hadoop frameworks where latency is not critical, like Hive or Spark.

2. Context & Background information

The base information provided in this section helps to understand the different components described in this article and the naming convention used in it.

2.1. Software pre-requisites

- Hadoop Distributed Platform (HDP) 2.4.2 / 2.4.3 / 2.2 & Kerberized
  - Check Hortonworks references to find the HWX version associated with each Hadoop version.
  - In this document, Hortonwors (HWX) or HDP are used equally, referring to the Hadoop node / cluster.
- ECS version 3.x, with basic configuration (namespace, user and bucket)
  - Check the ECS references to configure the system properly for these tests.
- Active Directory (AD) and Kerberos Key Distribution Center (KDC)
  - These tests have been done using a KDC in AD instance.

2.2. Test environment - Naming Convention

To make examples more understable, this is the naming convention used in this article :

- ECS nodes:
  - ecsparis1.paris.lab [10.10.10.11]
  - ecsparis2.paris.lab [10.10.10.12]
  - ecsparis3.paris.lab [10.10.10.13]
  - ecsparis4.paris.lab [10.10.10.14]

- HDP nodes:
  - hdp2.paris.lab [10.10.10.122]

- AD - KDC:
  - paris.lab [10.10.10.99]

- Users:
  - hdp2admin – Admin user for hdp OU in AD
  - vipr-ecsX users - AD users to create ECS keytabs in AD
  - hdpuser1@PARIS.LAB - ECS / Hadoop user
  - hdpuser2@PARIS.LAB - ECS / Hadoop user
  - hdfs-hdp2@PARIS.LAB - ECS user - hdfs Service principal in Hadoop cluster

- Buckets:
  - hdpuser1bucket1 bucket - Owned by hdpuser1@PARIS.LAB user
  - hdpuser1bucket2 bucket - Owned by hdpuser1@PARIS.LAB user
  - hdpuser2bucket1 bucket - Owned by hdpuser2@PARIS.LAB user
  - hdp22 bucket - Owned by hdfs-hdp2@PARIS.LAB user
    - Used for ECS as default File System (FS) configuration in HDP

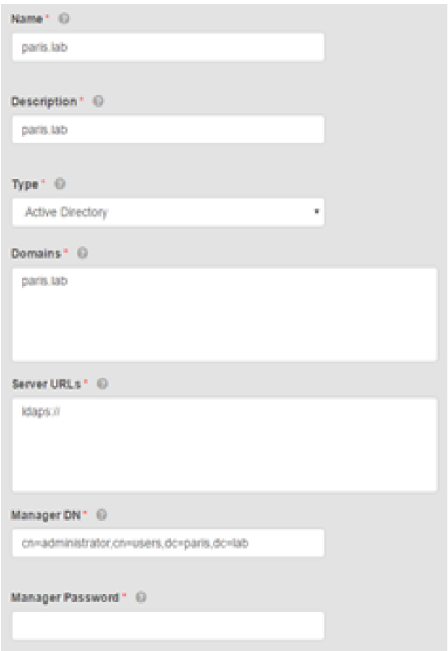
3. Kerberos Configuration

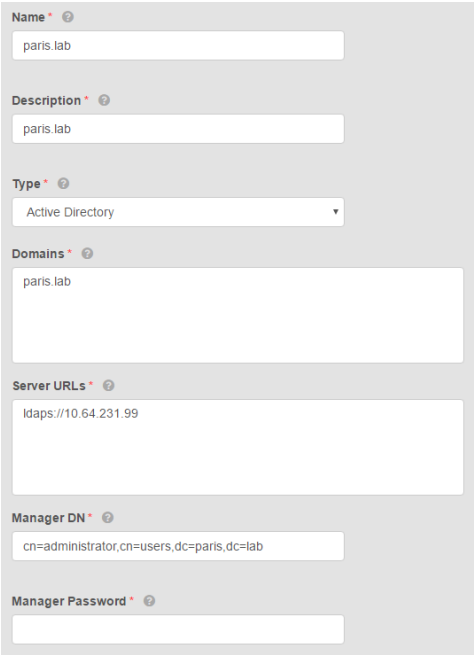
This section describes how to secure the communication between ECS, Hortonworks and the Active Directory using Kerberos.

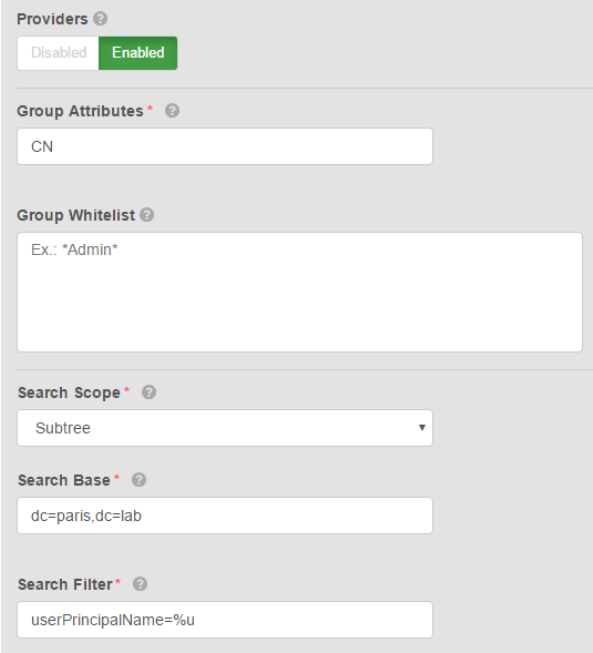
3.1. Kerberos configuration in Active Directory

- *In ECS:*

Configure the Authentication Provider in ECS:

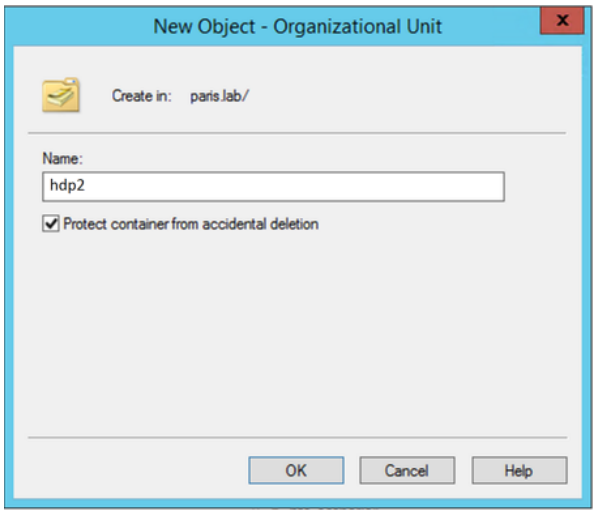




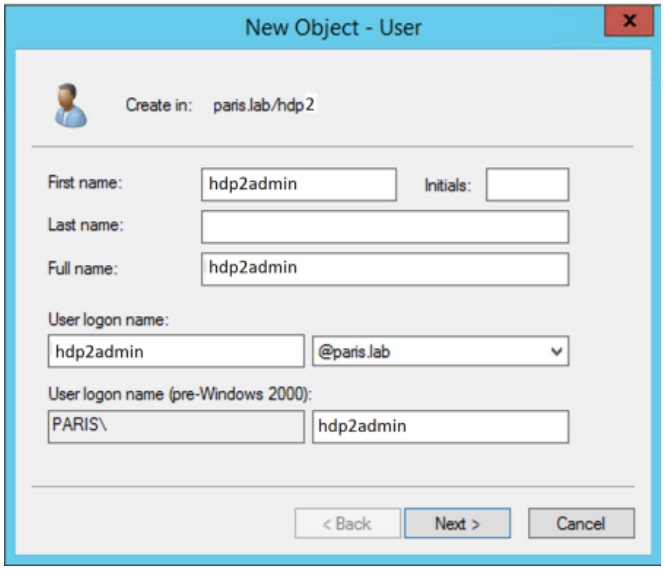


- *From the AD*

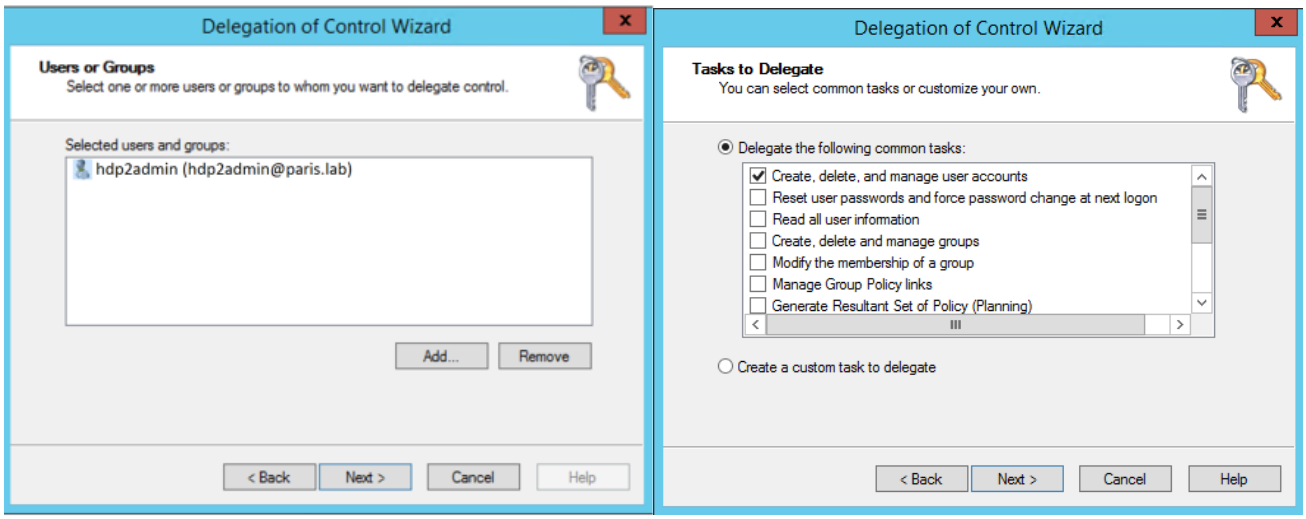
Create a new Organizational Unit (OU):



Create a new admin User for that OU:

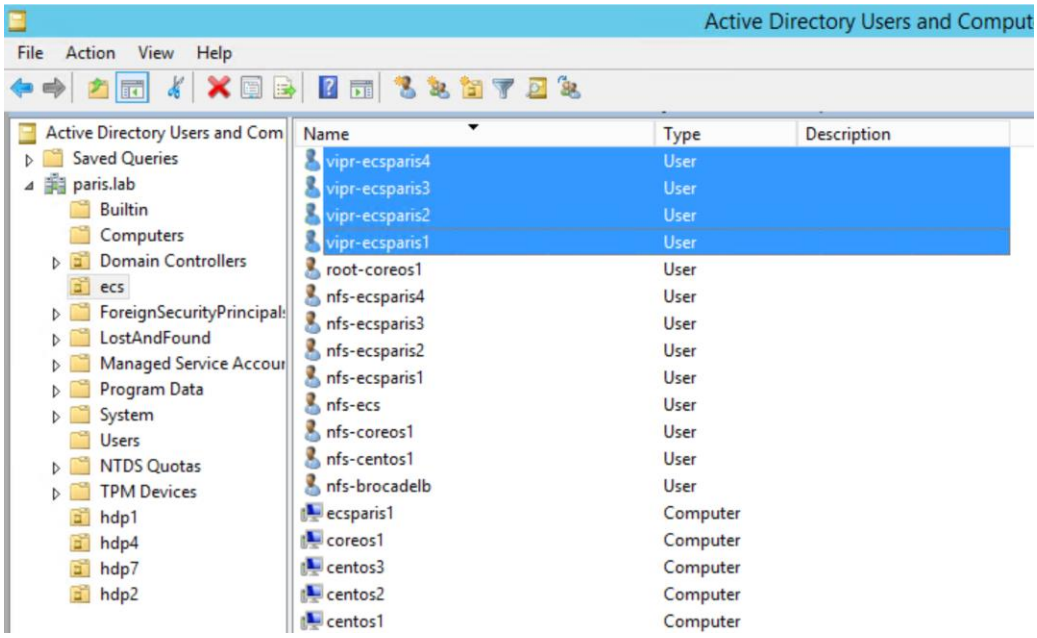


Delegate the control of that OU to that recently created user:



Using an AD for Kerberos, and not an external KDC, it is necessary to manually create:

- *vipr-ecsX* users in the AD
- Keytabs for the ECS in the AD




```
ktpass -princ vipr/ecsparis1.paris.lab@PARIS.LAB +rndPass -mapUser vipr-ecs1@PARIS.LAB -
mapOp set -crypto All -ptype KRB5_NT_PRINCIPAL -out ecsparis1.paris.lab@PARIS.LAB.keytab
ktpass -princ vipr/ecsparis2.paris.lab@PARIS.LAB +rndPass -mapUser vipr-ecs2@PARIS.LAB -
mapOp set -crypto All -ptype KRB5_NT_PRINCIPAL -out ecsparis2.paris.lab@PARIS.LAB.keytab
ktpass -princ vipr/ecsparis3.paris.lab@PARIS.LAB +rndPass -mapUser vipr-ecs3@PARIS.LAB -
mapOp set -crypto All -ptype KRB5_NT_PRINCIPAL -out ecsparis3.paris.lab@PARIS.LAB.keytab
ktpass -princ vipr/ecsparis4.paris.lab@PARIS.LAB +rndPass -mapUser vipr-ecs4@PARIS.LAB -
mapOp set -crypto All -ptype KRB5_NT_PRINCIPAL -out ecsparis4.paris.lab@PARIS.LAB.keytab
```

Save the Keytabs. They will have to be copied to the ECS node [section 3.3.1].

3.1.1. System Security Services Daemon (SSSD) Configuration [Optional]

This section describes how to configure SSSD for your AD users, if desired. SSSD will allow AD users to directly *ssh* into the hdp cluster, without the need of using *kinit* commands.

- *On your hdp node:*

```
# yum -y -q install epel-release
# yum -y -q install sssd oddjob-mkhomedir authconfig sssd-krb5 sssd-ad sssd-tools
# yum -y -q install adcli
```

```
# ad_user="hdp2admin"
# ad_domain="paris.lab"
# ad_dc="10.10.10.99"
# ad_root="dc=paris,dc=lab"
# ad_ou="ou=hdp2,${ad_root}"
# ad_realm=${ad_domain^^}
```

Kinit as your AD administrator:

```
# kinit administrator
Password for administrator@PARIS.LAB:
```

```
# echo adcli join -v \ --domain-controller=${ad_dc} \ --domain-ou="${ad_ou}" \ --login-ccache="/tmp/krb5cc_0" \ --login-user="${ad_user}" \ -v \ --show-details

# adcli join -v --domain-controller=${ad_dc} --domain-ou="${ad_ou}" --login-ccache="/tmp/krb5cc_0" --login-user="${ad_user}" -v --show-details
```

```
#vi /etc/sssd/sssd.conf

[sssd]
## master & data nodes only require nss. Edge nodes require pam.
services = nss, pam, ssh, autofs, pac
config_file_version = 2
domains = PARIS.LAB
override_space = _

[domain/PARIS.LAB]
id_provider = ad
ad_server = 10.10.10.99
auth_provider = ad
chpass_provider = ad
access_provider = ad
enumerate = False
krb5_realm = PARIS.LAB
ldap_schema = ad
ldap_id_mapping = True
cache_credentials = True
ldap_access_order = expire
ldap_account_expire_policy = ad
ldap_force_upper_case_realm = true
```

```
fallback_homedir = /home/%d/%u
default_shell = /bin/false
ldap_referrals = false

[nss]
memcache_timeout = 3600
override_shell = /bin/bash
```

```
# chmod 0600 /etc/sssd/sssd.conf
# systemctl restart sssd.service
# systemctl status sssd.service
# sudo authconfig --enablesssd --enablesssdauth --enablemkhomedir --enablelocauthorize --update
# sudo chkconfig oddjobd on
# sudo service oddjobd restart
# sudo chkconfig sssd on
# sudo service sssd restart
# sysctl status sssd.service
```

```
# ssh hdpuser1@PARIS.LAB@hdp2.PARIS.LAB
hdpuser1@PARIS.LAB@hdp2.paris.lab's password:
Creating home directory for hdpuser1@PARIS.LAB.
```

3.2. Kerberos configuration in HortonWorks

- *From the AD*

Extract the certificate from the AD sever.

- *From the Hdp Linux host:*

Import the AD certificate:

- Create '*/etc/pki/ca-trust/source/anchors/activedirectory.pem*' and paste the certificate contents

- Trust CA cert:

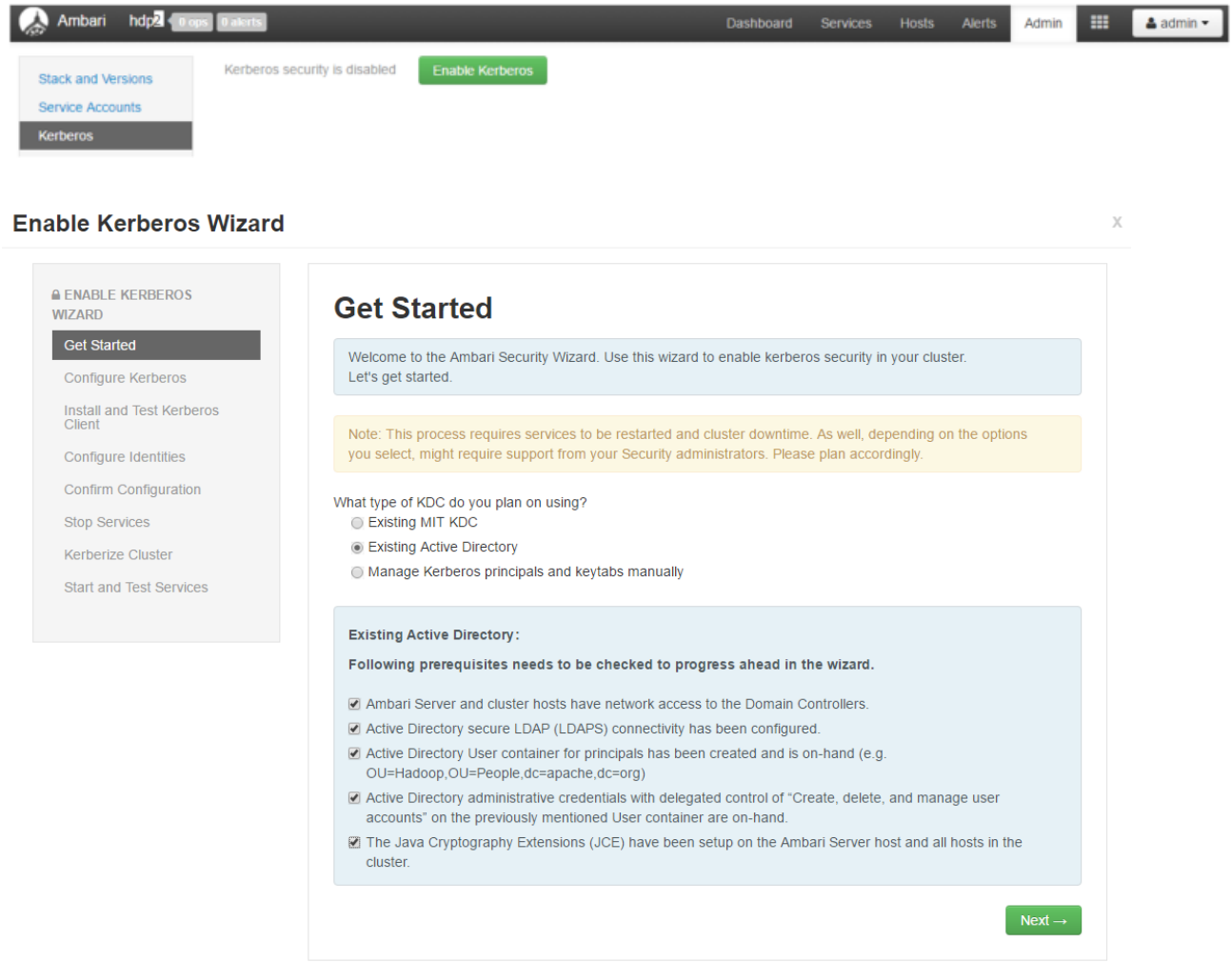
```
# sudo update-ca-trust enable
# sudo update-ca-trust extract
# sudo update-ca-trust check
```

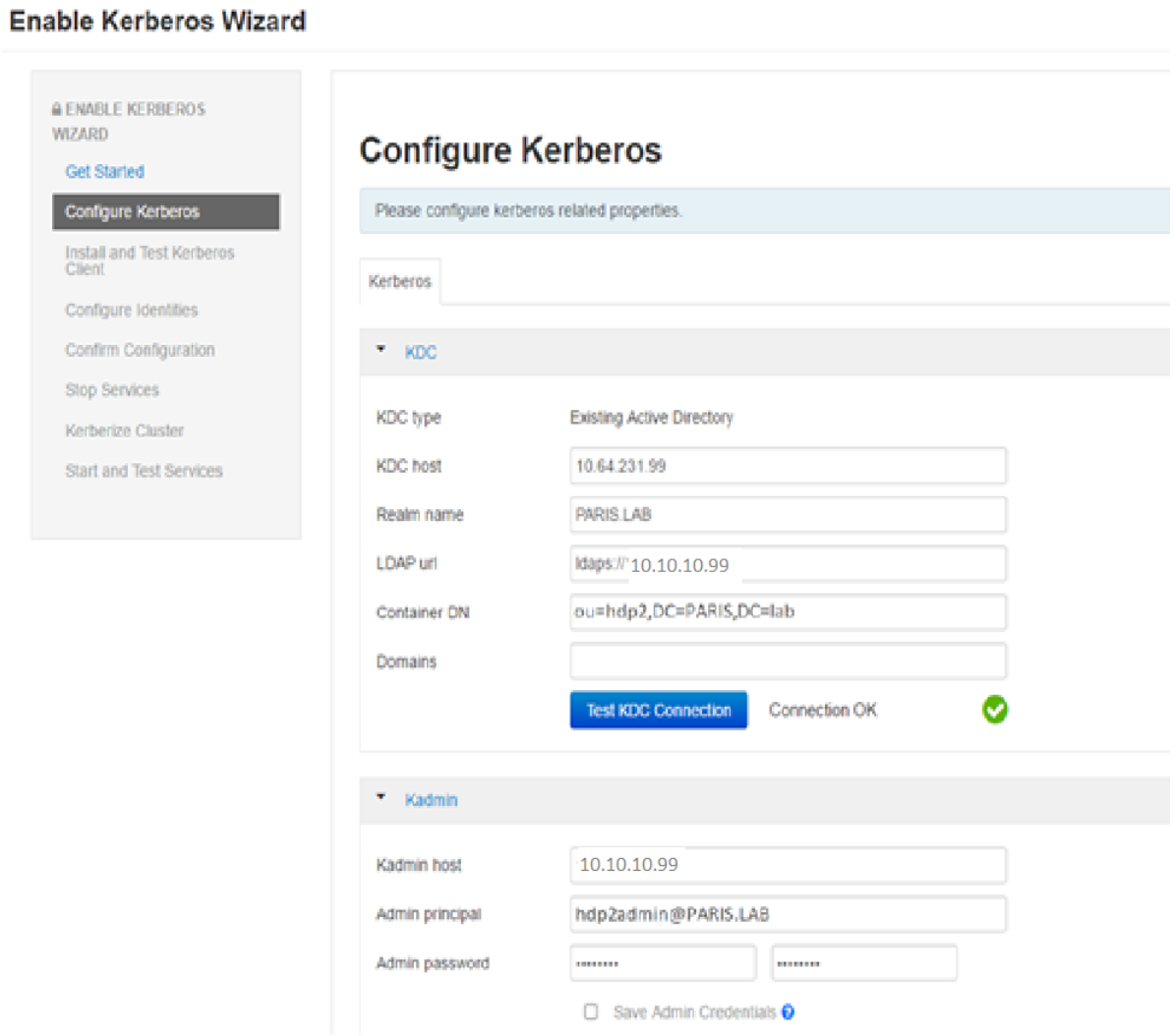
- Trust CA cert in Java:

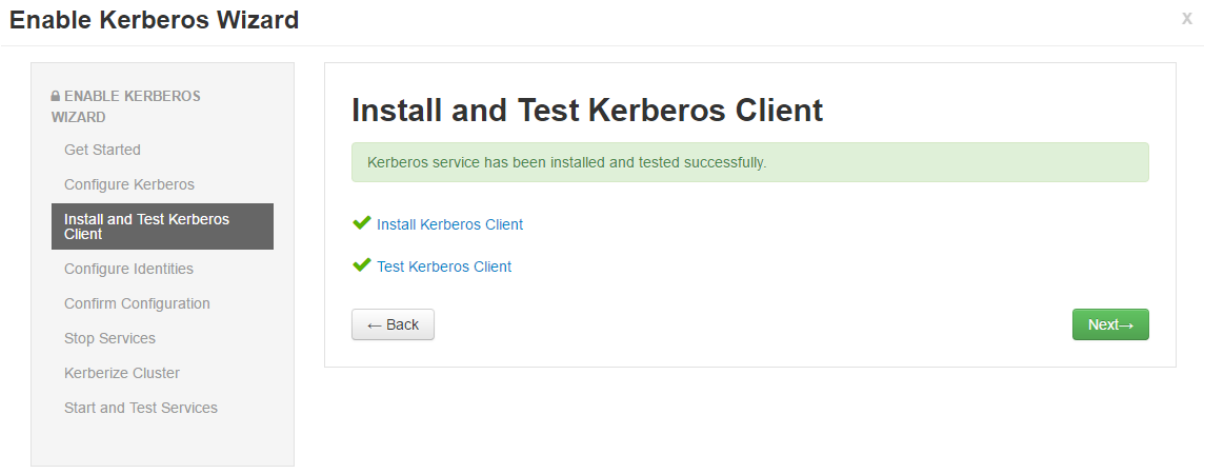
```
# mycert=/etc/pki/ca-trust/source/anchors/activedirectory.pem sudo keytool -importcert -noprompt -storepass changeit -file ${mycert} -alias ad -keystore /etc/pki/java/cacerts
```

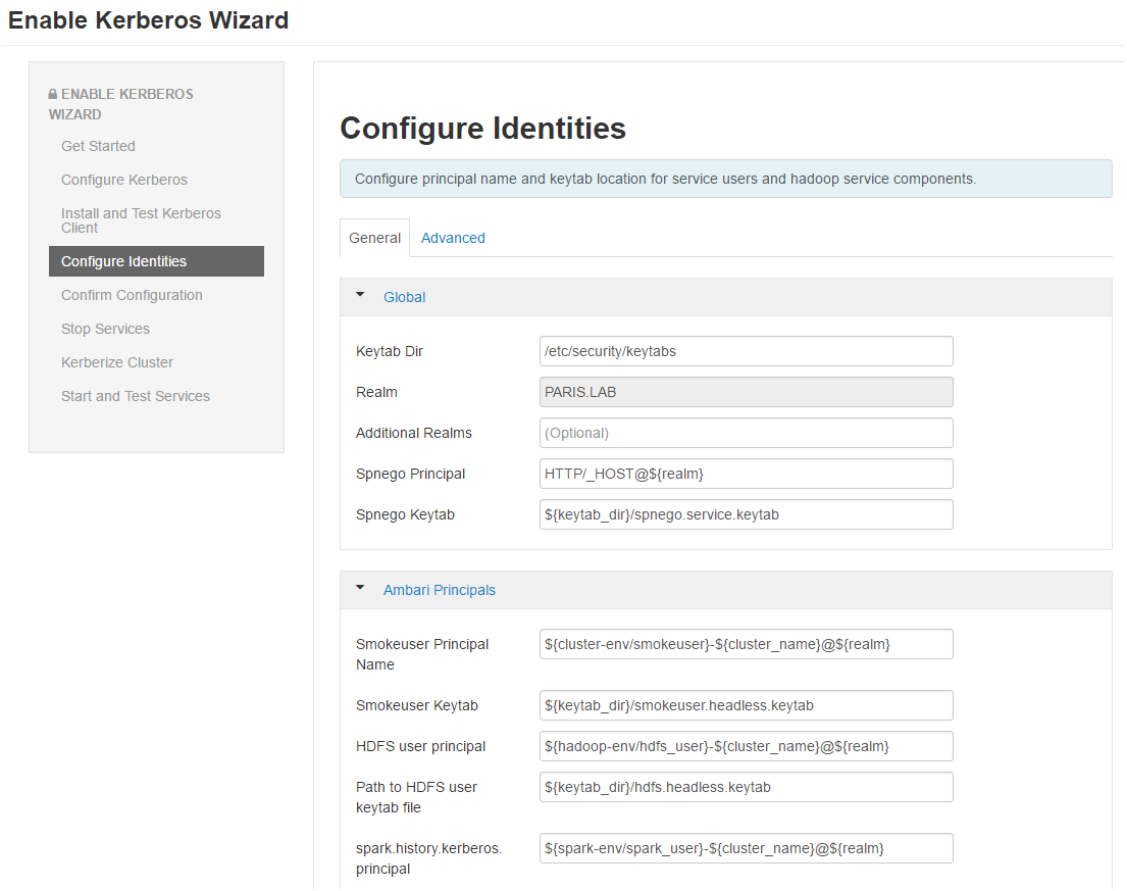
- *From the Ambari Graphical User Interface (GUI):*

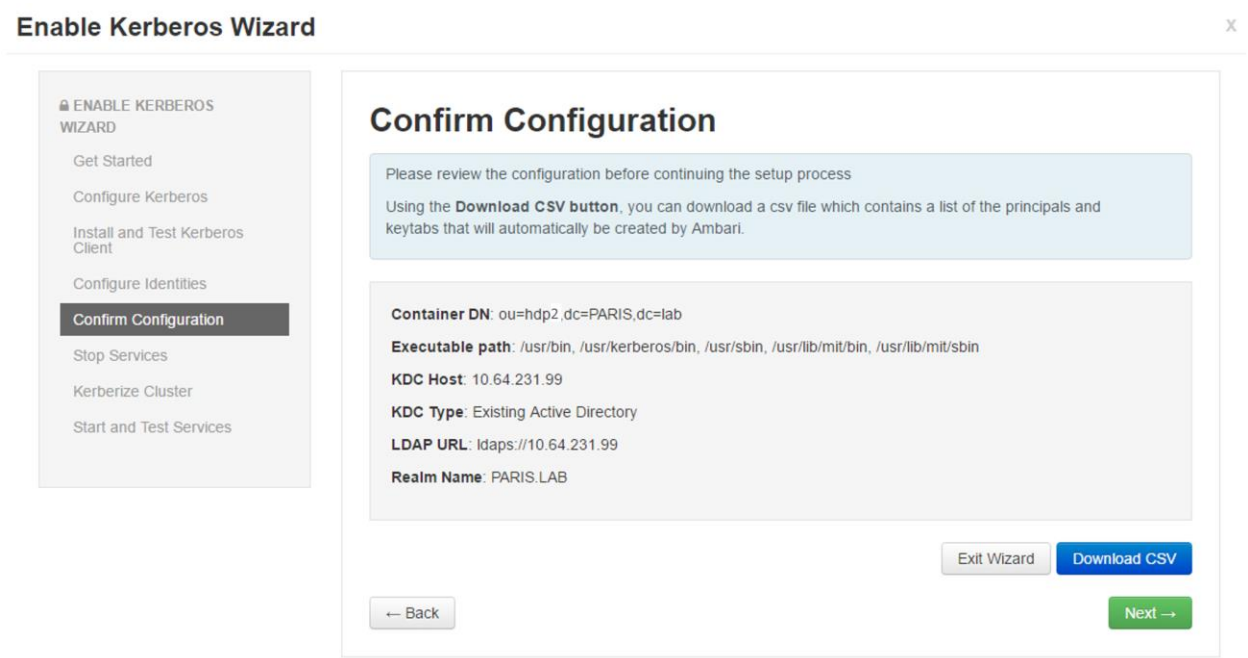
Click "*Enable Kerberos*" tab and follow the wizard according to the following options:


Ambari
hdp2
0 ops
0 alerts
Dashboard
Services
Hosts
Alerts
Admin
admin
Stack and Versions
Service Accounts
Kerberos
Kerberos security is disabled
Enable Kerberos
Enable Kerberos Wizard
ENABLE KERBEROS WIZARD
Get Started
Configure Kerberos
Install and Test Kerberos Client
Configure Identities
Confirm Configuration
Stop Services
Kerberize Cluster
Start and Test Services
Get Started
Welcome to the Ambari Security Wizard. Use this wizard to enable kerberos security in your cluster.
Let's get started.
Note: This process requires services to be restarted and cluster downtime. As well, depending on the options you select, might require support from your Security administrators. Please plan accordingly.
What type of KDC do you plan on using?
Existing MIT KDC
Existing Active Directory
Manage Kerberos principals and keytabs manually
Existing Active Directory:
Following prerequisites needs to be checked to progress ahead in the wizard.
Ambari Server and cluster hosts have network access to the Domain Controllers.
Active Directory secure LDAP (LDAPS) connectivity has been configured.
Active Directory User container for principals has been created and is on-hand (e.g. OU=Hadoop,OU=People,dc=apache,dc=org)
Active Directory administrative credentials with delegated control of "Create, delete, and manage user accounts" on the previously mentioned User container are on-hand.
The Java Cryptography Extensions (JCE) have been setup on the Ambari Server host and all hosts in the cluster.
Next →

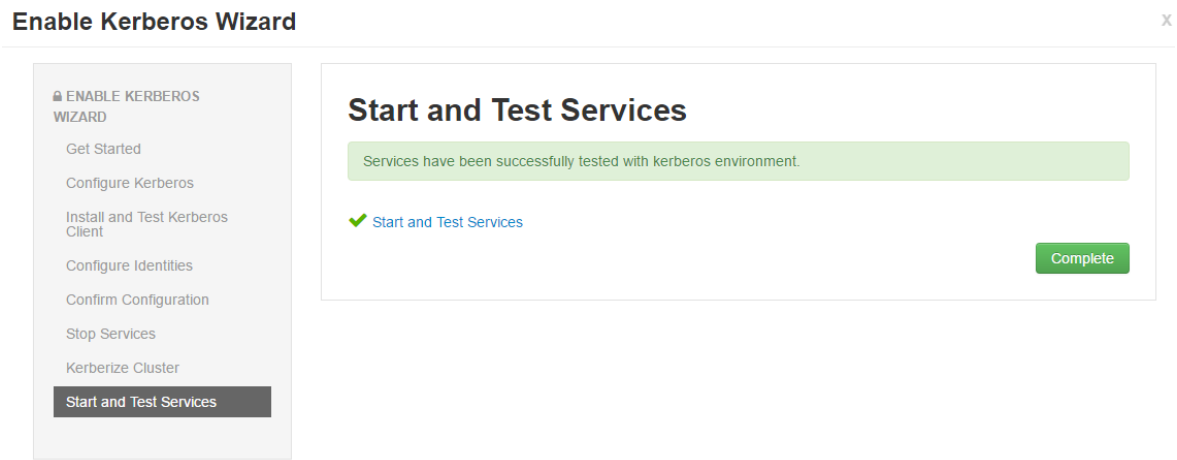

Enable Kerberos Wizard
ENABLE KERBEROS WIZARD
Get Started
Configure Kerberos
Install and Test Kerberos Client
Configure Identities
Confirm Configuration
Stop Services
Kerberize Cluster
Start and Test Services
Configure Kerberos
Please configure kerberos related properties.
Kerberos
KDC
KDC type
Existing Active Directory
KDC host
10.64.231.99
Realm name
PARIS.LAB
LDAP url
ldaps://10.10.10.99
Container DN
ou=hdp2,DC=PARIS,DC=lab
Domains
Test KDC Connection
Connection OK
Kadmin
Kadmin host
10.10.10.99
Admin principal
hdp2admin@PARIS.LAB
Admin password
Save Admin Credentials


Enable Kerberos Wizard
ENABLE KERBEROS WIZARD
Get Started
Configure Kerberos
Install and Test Kerberos Client
Configure Identities
Confirm Configuration
Stop Services
Kerberize Cluster
Start and Test Services
Install and Test Kerberos Client
Kerberos service has been installed and tested successfully.
Install Kerberos Client
Test Kerberos Client
← Back
Next→


Enable Kerberos Wizard
ENABLE KERBEROS WIZARD
Get Started
Configure Kerberos
Install and Test Kerberos Client
Configure Identities
Confirm Configuration
Stop Services
Kerberize Cluster
Start and Test Services
Configure Identities
Configure principal name and keytab location for service users and hadoop service components.
General
Advanced
Global
Keytab Dir
/etc/security/keytabs
Realm
PARIS.LAB
Additional Realms
(Optional)
Spnego Principal
HTTP/_HOST@${realm}
Spnego Keytab
${keytab_dir}/spnego.service.keytab
Ambari Principals
Smokeuser Principal Name
${cluster-env/smokeuser}-${cluster_name}@${realm}
Smokeuser Keytab
${keytab_dir}/smokeuser.headless.keytab
HDFS user principal
${hadoop-env/hdfs_user}-${cluster_name}@${realm}
Path to HDFS user keytab file
${keytab_dir}/hdfs.headless.keytab
spark.history.kerberos. principal
${spark-env/spark_user}-${cluster_name}@${realm}


Enable Kerberos Wizard
ENABLE KERBEROS WIZARD
Get Started
Configure Kerberos
Install and Test Kerberos Client
Configure Identities
Confirm Configuration
Stop Services
Kerberize Cluster
Start and Test Services
Confirm Configuration
Please review the configuration before continuing the setup process
Using the Download CSV button, you can download a csv file which contains a list of the principals and keytabs that will automatically be created by Ambari.
Container DN: ou=hdp2,dc=PARIS,dc=lab
Executable path: /usr/bin, /usr/kerberos/bin, /usr/sbin, /usr/lib/mit/bin, /usr/lib/mit/sbin
KDC Host: 10.64.231.99
KDC Type: Existing Active Directory
LDAP URL: ldaps://10.64.231.99
Realm Name: PARIS.LAB
Exit Wizard
Download CSV
← Back
Next →

[…]



3.3. Kerberos configuration in ECS

3.3.1. Keytab configuration

Copy the **keytabs**, the **ECS HDFS package** and the **UnlimitedJCEPolicy** to the first ECS node.

Note that HDFS support tools are provided in a HDFS Client ZIP file, *hdfsclient-<ECS version>-<version>.zip*, that you can download from the ECS support pages on support.emc.com. The unlimited JCE policy archive can be downloaded from oracle.com.

- *On the first ECS node:*

```
# cd /home/admin
# unzip hdfsclient-3.0.0.0.85807.98632a9.zip
# unzip UnlimitedJCEPolicyJDK7.zip
```

Edit *inventory.txt* in the *playbooks/samples* directory to refer to the ECS data nodes and KDC server:

```
# vi viprfs-client-3.0.0.0.85807.98632a9/playbooks/samples/inventory.txt
[data_nodes]
10.10.10.[11:14]

[kdc]
10.10.10.99
```

```
# cp -r UnlimitedJCEPolicy viprfs-client-3.0.0.0.85807.98632a9/playbooks/samples/
```

```
# mkdir viprfs-client-3.0.0.0.85807.98632a9/playbooks/samples/keytabs
# cp *keytab viprfs-client-3.0.0.0.85807.98632a9/playbooks/samples/keytabs/
```

Start the *utility container* on ECS Node 1 and make the Ansible playbooks available to the container.

```
# sudo docker load -i /opt/emc/caspian/checker/docker/images/utilities.txz

# sudo docker images
REPOSITORY          TAG                    IMAGE ID        CREATED         VIRTUAL SIZE
emcvipr/object      3.0.0.0-86239.1c9e5ec  6b24682a1ecb    6 weeks ago     1.561 GB
```

```
fabric/syslog            1.3.0.0-3024.09f2704    0d85e0e38369        10 weeks ago        396.8 MB
caspian/fabric           1.3.0.0-3024.09f2704    0603856d6974        10 weeks ago        411.9 MB
utilities                1.5.0.0-432.2ae21ed     be5eee0f7494        5 months ago        740.7 MB
caspian/fabric-registry  1.2.0.0-41.e66a0a4      b5f54f94a108        7 months ago        355.9 MB
caspian/fabric-zookeeper 1.2.0.0-67.b4aa9c2      70bfa83af17e        7 months ago        386.9 MB

# sudo docker run -v /opt/emc/caspian/fabric/agent/services/object/main/log:/opt/storageos/logs -v
/home/admin/viprfs-client-3.0.0.0.85807.98632a9/playbooks:/ansible --name=ecs-tools -i -t --
privileged --net=host be5eee0f7494 /bin/bash
```

- In the *Utility container*:

```
# cd /ansible
# vi samples/krb5.conf

[libdefaults]
  renew_lifetime = 7d
  forwardable = true
  default_realm = PARIS.LAB
  ticket_lifetime = 24h
  dns_lookup_realm = false
  dns_lookup_kdc = false
  #default_tgs_enctypes = aes des3-cbc-sha1 rc4 des-cbc-md5
  #default_tkt_enctypes = aes des3-cbc-sha1 rc4 des-cbc-md5


[domain_realm]
  paris.lab = PARIS.LAB

[logging]
  default = FILE:/var/log/krb5kdc.log
  admin_server = FILE:/var/log/kadmind.log
  kdc = FILE:/var/log/krb5kdc.log

[realms]
  PARIS.LAB = {
    admin_server = 10.10.10.99
    kdc = 10.10.10.99
  }
```

Use ansible to set up the *vipr-kerberos*. Note that the *generate-vipr-keytabs.yml* doesn't work when using a KDC in AD, and not an external KDC.

Since Keytabs have already been created and uploaded to the ECS, just use the *setup-vipr-kerberos.yml* script to configure ECS for Kerberos authentication. This will configure the data nodes with the ECS service principal.

```
# ansible-galaxy install -r requirements.txt -f
```

```
# vi samples/generate-vipr-keytabs.yml
---
###
```

```
# Generates keytabs for ViPR/ECS data nodes.
###

- hosts: data_nodes
  serial: 1

  roles:
    - role: vipr_kerberos_principal
      kdc: "{{ groups.kdc | first }}"
      principals:
        - name:  vipr/_HOST@PARIS.LAB
          keytab: keytabs/_HOST@PARIS.LAB.keytab
```

```
# samples/setup-vipr-kerberos.yml
---
###
# Configures ViPR/ECS for Kerberos authentication.
# - Configures krb5 client
# - Installs keytabs
# - Installs JCE policy
###

 - hosts: data_nodes

   roles:
     - role: vipr_kerberos_config
       krb5:
         config_file: krb5.conf
       service_principal:
         name: vipr/_HOST@PARIS.LAB
         keytab: keytabs/_HOST@PARIS.LAB.keytab

     - role: vipr_jce_config
       jce_policy:
         name: unlimited
         src: UnlimitedJCEPolicy/
```

```
# export ANSIBLE_HOST_KEY_CHECKING=False
```

```
# cd samples

# ansible-playbook -v -k -i inventory.txt --user admin -b --become-user=root setup-vipr-kerberos.yml

...
PLAY RECAP *********************************************************************
10.10.10.11              : ok=21   changed=8    unreachable=0    failed=0
10.10.10.12              : ok=21   changed=8    unreachable=0    failed=0
10.10.10.13              : ok=21   changed=8    unreachable=0    failed=0
10.10.10.14              : ok=21   changed=8    unreachable=0    failed=0
```

Verify that the correct ECS service principal, one per data node, has been created (from the KDC):

```
# kadmin.local -q "list_principals" | grep vipr
```

3.3.2. Bucket metadata – Securing ECS bucket

Additional to this ECS configuration, it will be necessary to secure the ECS buckets.

The metadata values required to secure an ECS bucket for use with a secure Hadoop cluster can be supplied by running ECS Management REST API commands.

Once metadata is loaded into a bucket, it is referred to as a "secure bucket" and you must have Kerberos principals to access it. A request from a non-secure Hadoop node will be rejected. If metadata has not been loaded, the bucket is not secure and a request from a secure Hadoop node will be rejected.

This procedure will be described in this section.

- *From the ECS GUI:*

Create an ECS object user called hdpuser1@PARIS.LAB (PARIS.LAB must be in uppercase).

Create a bucket with FS enabled, hadoop group.

```
User owner: hdpuser1@PARIS.LAB
Bucket name: hdpuser1bucket1
FS enabled, Group = hadoop
```

Configure S3Browser

- *On one HDP node:*

```
# export JAVA_HOME=/usr/jdk64/jdk1.8.0_60
```

Create a working directory for your bucket metadata, and copy all the scripts to that directory.

```
# mkdir /root/hdpuser1bucket1
#vi gatherDetails.sh
#vi bundleDetails.sh
#vi upload.sh
```

Update *gatherDetails.sh* to add all the master nodes, separated by space. In this case, we only have one *masternode= hdp2.paris.lab*.

```
# for host in `cat /etc/hadoop/conf/slaves` `hostname -f`
# for host in `cat /etc/hadoop/conf/slaves` masternode1.paris.lab masternode2.paris.lab
for host in `cat /etc/hadoop/conf/slaves` hdp2.paris.lab
```

```
# ./gatherDetails.sh
```

```
# ./bundleDetails.sh > details.json
```

Modify the shortname for the *hbase-hdp2* user in the *details.json* file

```
# vi details.json

        {
                "name": "internal.kerberos.user.hbase-hdp2.name",
                "value":        "hbase-hdp2@PARIS.LAB"
        },
        {
                "name": "internal.kerberos.user.hbase-hdp2.shortname",
                "value":        "hbase@PARIS.LAB"
        },
        {
                "name": "internal.kerberos.user.hbase-hdp2.groups",
                "value":        "hadoop"
        },
```

This allows ECS to consider *hbase@PARIS.LAB* and *hbase-hdp2@PARIS.LAB* as being the same users.

Add the AD user to the *details.json* file.

```
# vi details.json
{
        "head_type":    "hdfs",
        "metadata":     [
                {
                        "name": "internal.kerberos.user.hdpuser1.name",
                        "value":        "hdpuser1@PARIS.LAB"
                },
                {
                        "name": "internal.kerberos.user.hdpuser1.shortname",
                        "value":        "hdpuser1"
                },
                {
                        "name": "internal.kerberos.user.hdpuser1.groups",
                        "value":        "hadoop,users"
                },
...
```

Find and modify the *upload.sh* script to match your bucket information and upload the configuration to the bucket metadata.

```
# vi upload.sh
…

#!/bin/sh

ECSNODE="10.10.10.13"
BUCKET="hdpuser1bucket1"
NAMESPACE="paris"
JSONFILE="details.json"
```

```
echo "Upload bucket cache info to $ECSNODE for $BUCKET and namespace $NAMESPACE"

echo "Get Token"
TOKEN=$(curl -s -k -u root:ChangeMe -D - -o /dev/null https://$ECSNODE:4443/login | grep X-SDS-
AUTH-TOKEN | tr -cd '\40-\176')

echo "Push metadata to ECS"
curl -s -k -X PUT -H "$TOKEN" -H "Accept: application/json" -H "Content-Type: application/json" -T
$JSONFILE
https://$ECSNODE:4443/object/bucket/$BUCKET/metadata?namespace=$NAMESPACE
sleep 2

echo "Get metadata for verification"
curl -s -k -X GET -H "$TOKEN"
"https://$ECSNODE:4443/object/bucket/$BUCKET/metadata?headType=HDFS&namespace=$NAM
ESPACE" | xmllint -format -
```

```
# ./upload.sh
```

4. Hortonworks (HWX) integration with ECS

This section describes how to integrate Hortonworks with ECS, explaining what files are needed and how they should be configured.

4.1. Configuring ECS HDFS Client Library JAR file on HDP nodes:

Obtain the **ECS HDFS Client Library** for your Hadoop distribution from the EMC Support site on *support.emc.com* and include it in the classpath of each client node in the Hadoop cluster:

- *On one HDP node:*

```
# hadoop classpath
/usr/hdp/2.4.2.0-258/hadoop/conf:/usr/hdp/2.4.2.0-258/hadoop/lib/*:/usr/hdp/2.4.2.0-258/hadoop/.//*:/usr/hdp/2.4.2.0-258/hadoop-hdfs/./:/usr/hdp/2.4.2.0-258/hadoop-hdfs/lib/*:/usr/hdp/2.4.2.0-258/hadoop-hdfs/.//*:/usr/hdp/2.4.2.0-258/hadoop-yarn/lib/*:/usr/hdp/2.4.2.0-258/hadoop-yarn/.//*:/usr/hdp/2.4.2.0-258/hadoop-mapreduce/lib/*:/usr/hdp/2.4.2.0-258/hadoop-mapreduce/.//*:/usr/hdp/2.4.2.0-258/tez/*:/usr/hdp/2.4.2.0-258/tez/lib/*:/usr/hdp/2.4.2.0-258/tez/conf
```

Copy the *viprfs-client-3.0.0.0-hadoop-2.7.jar* file to */usr/hdp/2.4.2.0-258/hadoop/lib/* directory in each HDP node of your HDP cluster:

```
Copy viprfs-client-3.0.0.0-hadoop-2.7.jar to all
scp viprfs-client-3.0.0.0.85807.98632a9/client/viprfs-client-3.0.0.0-hadoop-2.7.jar
root@10.10.10.122:/usr/hdp/2.4.2.0-258/hadoop/lib/
```

Update the classpath configuration setting for MapReduce, yarn and also explicitly specify path to the JAR for Tez.

4.2. Config parameters in Ambari

Update the *core-site.xml* and other service properties in order to integrate ECS with a Hadoop cluster that uses Kerberos authentication mode. The default FS is still provided by HWX.

- *On Ambari GUI:*

Log in to the Ambari interface:

```
http://10.10.10.122:8080
admin / admin
```

Add the following parameters to the *custom core-site.xml*:

- HDFS – Configs – Advanced - *custom core-site.xml:*

```
fs.AbstractFileSystem.viprfs.impl     com.emc.hadoop.fs.vipr.ViPRAbstractFileSystem
fs.vipr.installation.Site1.hosts     10.10.10.13, 10.10.10.12
fs.vipr.installation.Site1.resolution     dynamic
fs.vipr.installation.Site1.resolution.dynamic.time_to_live_ms     900000
fs.vipr.installations     Site1
fs.viprfs.auth.anonymous_translation     CURRENT_USER
fs.viprfs.auth.identity_translation     CURRENT_USER_REALM
fs.viprfs.impl     com.emc.hadoop.fs.vipr.ViPRFileSystem
```

viprfs.security.principal	vipr/_HOST@PARIS.LAB

Add the following configuration [*/usr/hdp/2.4.2.0-258/hadoop/lib/viprfs-client-3.0.0.0-hadoop-2.7*.jar] to these services:

- Yarn:
  yarn.application.classpath /usr/hdp/2.4.2.0-258/hadoop/lib/viprfs-client-3.0.0.0-hadoop-2.7.jar
- Tez:
  tez.cluster.additional.classpath.prefix /usr/hdp/2.4.2.0-258/hadoop/lib/viprfs-client-3.0.0.0-hadoop-2.7.jar
- MapReduce2:
  mapreduce.application.classpath /usr/hdp/2.4.2.0-258/hadoop/lib/viprfs-client-3.0.0.0-hadoop-2.7.jar

- Hive:
  Add the following entry under *custom hive-site.xml*:
  hive.aux.jars.path /usr/hdp/2.4.2.0-258/hadoop/lib/viprfs-client-3.0.0.0-hadoop-2.7.jar

  On advanced spark-env, add the following to *spark-env template:*

```
export SPARK_CLASSPATH=/usr/hdp/2.4.2.0-258/hadoop/lib/viprfs-client-3.0.0.0-hadoop-2.7.jar:/usr/hdp/2.4.2.0-258/hadoop/lib/guava-11.0.2.jar
export SPARK_DIST_CLASSPATH=/usr/hdp/2.4.2.0-258/hadoop/lib/viprfs-client-3.0.0.0-hadoop-2.7.jar:/usr/hdp/2.4.2.0-258/hadoop/lib/guava-11.0.2.jar
```

Restart all Hadoop services.

5. ECS as a secondary FS in HDP cluster

The HWX cluster is already configured to use ECS. This section describes how to use ECS as a secondary File System in the Hadoop cluster.

5.1. Bucket configuration

In order to R/W data from an external bucket, you will need to upload the bucket metadata to that bucket, as explained in section 3.3.2.

```
User owner: hdpuser1@PARIS.LAB
Bucket name: hdpuser1bucket1
FS enabled, Group = hadoop
```

In the following sections we'll explain how to test it and how to configure other advanced options.

5.2. Bucket access verification

5.2.1. AD user hdpuser1 / bucket hdpuser1bucket1

- *On your HDP linux host*

Kinit as the AD user *hdpuser1* and verify that you can list the bucket just configured (*hdpuser1bucket1*).

```
# kinit hdpuser1
Password for hdpuser1@PARIS.LAB:
```

Upload a file using S3Browser (*workload-config_cluster_60K_900Million_1.txt* in the example below), and verify that the permissions are correct and that you can *cat* that file from you hdp cluster as *hdpuser1* user:

```
# hdfs dfs -ls viprfs://hdpuser1bucket1.paris.Site1/
Found 1 items
-rwxrwx---   1 hdpuser1 hadoop        620 2017-02-21 10:54
viprfs://hdpuser1bucket1.paris.Site1/workload-config_cluster_60K_900Million_1.txt

# hdfs dfs -cat viprfs://hdpuser1bucket1.paris.Site1/workload-config_cluster_60K_900Million_1.txt
```

Verify that you can copy a file from your Hadoop cluster as *hdpuser1* AD user and that you can see it from S3 Browser with the *hdpuser1* object user credentials.

```
# hdfs dfs -copyFromLocal details.json viprfs://hdpuser1bucket1.paris.Site1/
```

5.2.2. AD user hdpuser1 / Bucket hdpuser1bucket2

- *From the ECS GUI*

Create a second bucket called *hdpuser1bucket2* owned by *hdpuser1@PARIS.LAB*

```
User owner: hdpuser1@PARIS.LAB
Bucket name: hdpuser1bucket2
FS enabled, Group = hadoop
```

Upload the bucket metadata for this new bucket *hdpuser1bucket2* following the same steps described before.

Verify that you can upload data from your hdp cluster / S3 Browser and access it from S3 Browser / hdp cluster, as *hdpuser1* user.

```
# hdfs dfs -copyFromLocal details.json viprfs://hdpuser1bucket2.paris.Site1/
```

5.2.3. Multi user Access to a bucket - AD user hdpuser1 & hdpuser2 / Bucket hdpuser2bucket1

- *From the ECS GUI*

Create a second user called *hdpuser2@PARIS.LAB* and a new bucket *hdpuser2bucket1* owned by this new user.

```
User owner: hdpuser2@PARIS.LAB
Bucket name: hdpuser2bucket1
FS enabled, Group = hadoop
```

Kinit as *hdpuser2* an verify that the bucket access is correct.

```
# kinit hdpuser2
Password for hdpuser2@PARIS.LAB:

# hdfs dfs -ls viprfs://hdpuser2bucket1.paris.Site1/

# hdfs dfs -copyFromLocal details.json viprfs://hdpuser2bucket1.paris.Site1/

# hdfs dfs -ls viprfs://hdpuser2bucket1.paris.Site1/
Found 1 items
-rw-r--r--   1 hdpuser2 hadoop       4668 2017-02-21 11:34
viprfs://hdpuser2bucket1.paris.Site1/details.json
```

Upload the bucket metadata to allow access to both *hdpuser1* and *hdpuser2* to that bucket *hdpuser2bucket1*.

```
#vi details.json

{
        "head_type":    "hdfs",
        "metadata":     [
                {
                        "name": "internal.kerberos.user.hdpuser1.name",
                        "value":        "hdpuser1@PARIS.LAB"
                },
                {
                        "name": "internal.kerberos.user.hdpuser1.shortname",
                        "value":        "hdpuser1"
                },
                {
                        "name": "internal.kerberos.user.hdpuser1.groups",
```

```
                "value":        "hadoop"
        },
        {
                "name": "internal.kerberos.user.hdpuser2.name",
                "value":        "hdpuser2@PARIS.LAB"
        },
        {
                "name": "internal.kerberos.user.hdpuser2.shortname",
                "value":        "hdpuser2"
        },
        {
                "name": "internal.kerberos.user.hdpuser2.groups",
                "value":        "hadoop"
        },
…
```

Verify that you can access the data in that bucket as *hdpuser1* user.

```
# kinit hdpuser1
Password for hdpuser1@PARIS.LAB:

# hdfs dfs -cat viprfs://hdpuser2bucket1.paris.Site1/details.json
```

5.2.4. Multi-cluster access

In order to access the same bucket from several HWX clusters, *hdp2.paris.lab* and *hdpsecondcluster.paris.lab* for example, you will have to update the proxy entries in the bucket metadata.

```
        {
                "name": "hadoop.proxyuser.yarn.hosts",
                "value":        "hdp2.paris.lab, hdpsecondcluster.paris.lab "
        },
```

6. ECS as the default HWX FS

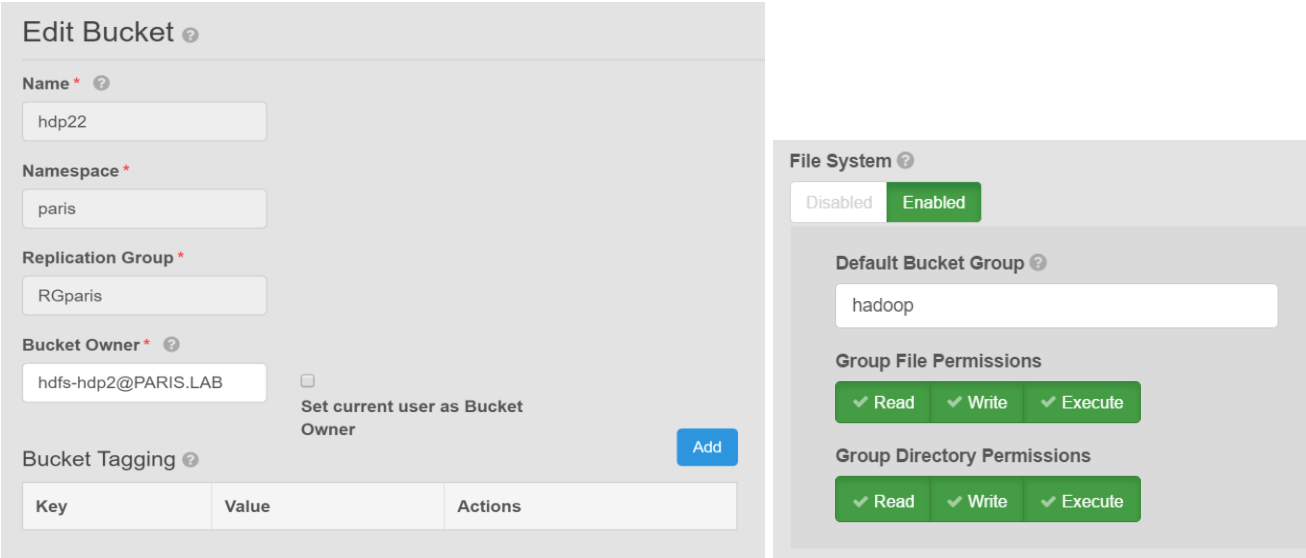
ECS can be also used as the primary File System in the Hadoop cluster. This section describes how to do it.

6.1. Bucket configuration

- *From the ECS GUI*

Create a new ECS Object user corresponding to your Service Principal *hdfs* in your HDP cluster.

Create a bucket, owned by that user, with FS enabled and belonging to *hadoop* group.



Upload the bucket metadata for that object user. Follow the instructions in the *Bucket metadata configuration* section.

6.2. HWX Configuration

Update the *core-site.xml* with the properties needed to integrate ECS HDFS with a Hadoop cluster that uses Kerberos authentication mode. In this section, HDP will use the ECS as the default FS.

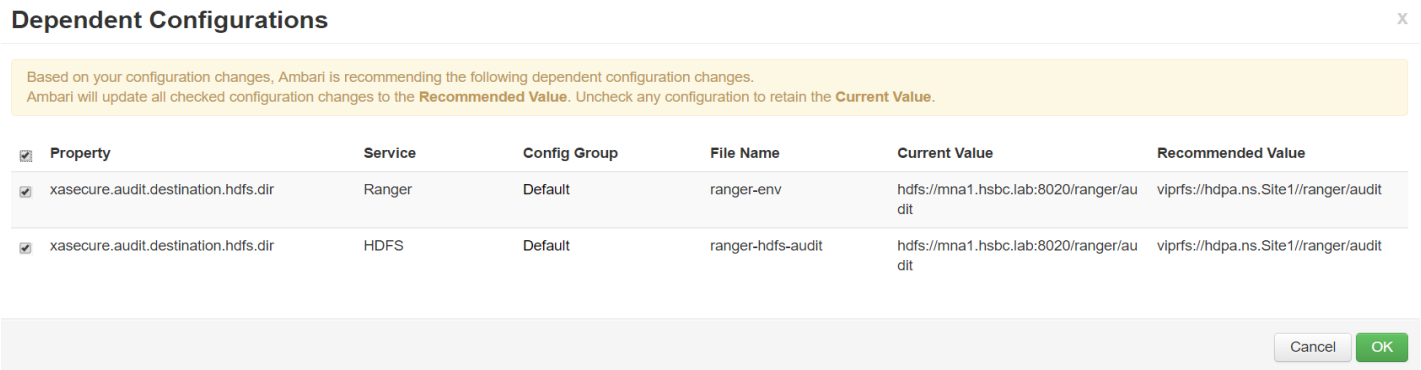
➢ *From the Ambari GUI:*

http://10.10.10.122:8080 admin / admin

Modify the Ambari configuration of the *custom core-site.xml* file performed in *section 4.2*:

- HDFS – Configs – Advanced
  Modify the default File System property in the custom *core-site.xml*:

Replace:	fs.defaultFS	hdfs://hdp2.paris.lab:8020
By:	fs.defaultFS	viprfs://hdpa.paris.Site1/



Click on OK

- Ranger
  Replace *hdfs* by *viprfs* in all the Ranger properties

- Hive
  Replace *hdfs* by *viprfs* in the hive templeton

- HBase
  On *Advanced hbase-site*, add:
  hbase.rootdir viprfs://hdp22.paris.Site1/apps/hbase/data

  Note: If HBase doesn't work properly, as the *hdfs* user, modify the directory permissions:
  # hdfs dfs -chmod 775 /apps/hbase

- Spark:
  On *Advanced spark-defaults*:
  spark.eventLog.dir viprfs://hdp22.paris.Site1/spark-history
  spark.history.fs.logDirectory viprfs://hdp22.paris.Site1/spark-history

6.3. Bucket access verification

➢ *From your HDP Linux host:*

Kinit as *hdfs*, and upload a file using either the *viprfs://* path or the local / one.

```
#  kinit -kt /etc/security/keytabs/hdfs.headless.keytab hdfs-hdp2@PARIS.LAB

# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: hdfs-hdp2@PARIS.LAB

Valid starting       Expires              Service principal
03/22/2017 15:42:22  03/23/2017 01:42:22  krbtgt/PARIS.LAB@PARIS.LAB
        renew until 03/29/2017 16:42:22

# hdfs dfs -ls  /tmp
Found 7 items
drwxr-xr-x   - hdfs hadoop          0 2017-03-20 17:18 viprfs://hdp22.paris.Site1/tmp/cris
drwxr-xr-x   - hdfs hadoop          0 2017-03-15 19:44 viprfs://hdp22.paris.Site1/tmp/entity-file-history

# hdfs dfs -copyFromLocal test1.txt /tmp/

# hdfs dfs -ls  /tmp
Found 8 items
drwxr-xr-x   - hdfs hadoop          0 2017-03-20 17:18 viprfs://hdp22.paris.Site1/tmp/cris
drwxr-xr-x   - hdfs hadoop          0 2017-03-15 19:44 viprfs://hdp22.paris.Site1/tmp/entity-file-history

# hdfs dfs -ls  viprfs://hdp22.paris.Site1/tmp
Found 8 items
drwxr-xr-x   - hdfs hadoop          0 2017-03-20 17:18 /tmp/cris
drwxr-xr-x   - hdfs hadoop          0 2017-03-15 19:44 /tmp/entity-file-history
```

7. S3a Configuration

This section will explain how to configure S3a (in ECS) as a secondary FS in the HDP cluster.

- *From the Ambari GUI:*

Add the following parameters to the custom *core-site.xml*

```
fs.s3a.impl     org.apache.hadoop.fs.s3a.S3AFileSystem
fs.s3a.access.key     hdpuser1@PARIS.LAB
fs.s3a.secret.key     zgvd7NkrChtPpFI58R9fjXw8qHrTaMWcDzavR5aN
fs.s3a.endpoint     http://10.10.10.13:9020
fs.s3a.connection.ssl.enabled     disabled
```

- *From your HDP Linux host:*

Verify that you can list the content of the bucket:

```
#hdfs dfs -ls s3a://hdpuser1bucket1/
Found 4 items
-rw-rw-rw-   1       4420 2017-02-21 13:55 s3a://hdpuser1bucket1/details.json
-rw-rw-rw-   1       4528 2017-02-21 13:49 s3a://hdpuser1bucket1/install
-rw-rw-rw-   1    1863951 2017-02-21 13:49 s3a://hdpuser1bucket1/virtualenv-15.1.0.tar.gz
-rw-rw-rw-   1        620 2017-02-21 10:54 s3a://hdpuser1bucket1/workload-
config_cluster_60K_900Million_1.txt
```

Verify that you can copy files to that bucket:

```
# hdfs dfs -copyFromLocal details.json /user/hdpuser1/
```

```
# hadoop distcp /user/hdpuser1/details.json s3a://hdpuser1bucket1/
```

8. Functional testing

Once the Hortonworks cluster and ECS are configured and secured with Kerberos, different frameworks can be used depending on the use case.

This section describes how to configure and test Distcp, Hive, Spark and HBase, with ECS as the default and ECS as the secondary File System for the Hadoop cluster.

8.1. Distcp operations

DistCp (distributed copy) is a tool used for large inter/intra-cluster copying. It uses MapReduce to effect its distribution, error handling and recovery, and reporting.

We will use *Distcp* to copy from *hdfs* to *hdfs*.

***Note:** If SSSD is not configure, it is necessary to locally add *hdpuser1* user to the hdp VM.

```
# useradd hdpuser1
# cat /etc/passwd
```

***Note:** If you are using a VM with limited resources for the *hdp* host, take a look at the recommended values for *mapreduce* and *yarn* in the Ambari GUI.

8.1.1. ECS as the default FS

As *hdfs-hdp2@PARIS.LAB* user with ECS *hdp22* bucket as the default FS:

```
# kinit hdfs
# hdfs dfs -mkdir /user/hdpuser1
# hdfs dfs -chown hdpuser1:hadoop /user/hdpuser1
```

8.1.1.1. Copying to an ECS bucket

As *hdpuser1@PARIS.LAB* user, initiate a *distcp* of a local file (ECS default FS) to an external ECS bucket:

```
# hadoop distcp /tmp/test1.txt viprfs://hdpuser1bucket1.paris.Site1/
```

Clean the environment for new tests:

```
# hdfs dfs -rm -skipTrash /tmp/test1.txt
```

8.1.1.2. Copying from an ECS bucket

```
# hadoop distcp viprfs://hdpuser1bucket1.paris.Site1/test1.txt /tmp/
```

8.1.2. ECS as a secondary FS

Default HWX FS. ECS as a secondary FS.

```
# kinit hdfs
# hdfs dfs -mkdir /user/hdpuser1
# hdfs dfs -chown hdpuser1:hadoop /user/hdpuser1
```

8.1.2.1. Copying to an ECS bucket

```
# hadoop distcp /user/hdfs/details.json viprfs://hdp22.paris.Site1/
```

Clean the environment for new tests:

```
# hdfs dfs -rm -skipTrash /user/hdfs/details.json
```

8.1.2.2. Copying from an ECS bucket

```
# hadoop distcp viprfs://hdp22.paris.Site1/details.json /user/hdfs/
```

8.2. Hive tests

Hive is an open source project run by Apache that facilitates reading, writing, and managing large datasets residing in distributed storage using SQL.

There are two types of tables in Hive (internal and external).

An **internal table** is also called a managed table, meaning it's "managed" by Hive. When you drop an internal table, Hive will delete both the schema/table definition, and will also physically delete the data/rows (truncation) associated with that table from HDFS.

An **external table** is not "managed" by Hive. When you drop an external table, the schema/table definition is deleted and gone, but the data/rows associated with it are left alone. The table's rows are not deleted.

By default, when you create an internal table, its location will be in /apps/hive/warehouse. An external table requires you to specify a location in HDFS where the data for the table you're creating will live.

In the following sections we'll perform tests using both types.

We will always use the following file (*table.csv*) to load data in Hive:

```
#vi table.csv
11223344,Vivek,9898054422,5039492020013110,100
12345678,Denis,9898054421,5039492020013110,2000
10001000,Ganesh,9898054423,5039492020013110,100000
11112222,Eric,9898054424,5039492020013110,500
33334444,Aengus,9898054425,5039492020013120,1000
44443333,Richard,9898054426,5039493320013110,201
44445555,Cristina,9898054427,5049493320013110,100000000
```

8.2.1. ECS as the default FS

8.2.1.1. Internal tables

Copy the table to the default FS (ECS), as *hive* user.

```
# su hive
# cd /
# vi table.csv
```

```
# kinit -kt /etc/security/keytabs/hive.service.keytab hive/hdp2.paris.lab@PARIS.LAB
```

```
# hdfs dfs -copyFromLocal table.csv /
```

Enter the *hive* shell:

```
# hive
WARNING: Use "yarn jar" to launch YARN applications.
Logging initialized using configuration in file:/etc/hive/2.4.3.0-227/0/hive-log4j.properties
```

Create an internal table called *employee.*

```
#hive> CREATE TABLE IF NOT EXISTS employee (
   UserID Int, Name String ,PhoneNo String,CardNo String,Amount Int)
   COMMENT 'Employee details'
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY ","
   LINES TERMINATED BY '\n'
   STORED AS TEXTFILE;
OK
Time taken: 13.363 seconds
```

And load the data into it:

```
# hive> LOAD DATA INPATH '/table.csv' OVERWRITE INTO TABLE employee ;
Loading data to table default.employee
Table default.employee stats: [numFiles=1, totalSize=346]
OK
Time taken: 20.359 seconds
```

****Note:** It should be fixed in future ECS releases and it shouldn't be neccessary, but if you have permissions issues add an acl to the default ECS bucket [*hdp22*] with the *hive* user [*hive @PARIS.LAB*].

Verify you can select info from the table:

```
# hive> SELECT * FROM employee;
OK
11223344        Vivek   9898054422      5039492020013110        100
12345678        Denis   9898054421      5039492020013110        2000
10001000        Ganesh  9898054423      5039492020013110        100000
11112222        Eric    9898054424      5039492020013110        500
33334444        Aengus  9898054425      5039492020013120        1000
44443333        Richard 9898054426      5039493320013110        201
44445555        Cristina        9898054427      5049493320013110        100000000
Time taken: 13.837 seconds, Fetched: 7 row(s)
```

8.2.1.2. External tables

Copy the table, as *hive* user.

```
# su hive
# cd /
# vi table.csv
```

```
# kinit -kt /etc/security/keytabs/hive.service.keytab hive/hdp2.paris.lab@PARIS.LAB
```

```
# hdfs dfs -mkdir /user/hive/ext
```

```
# hdfs dfs -copyFromLocal table.csv /user/hive/ext
```

Enter the *hive* shell:

```
# hive
WARNING: Use "yarn jar" to launch YARN applications.
Logging initialized using configuration in file:/etc/hive/2.4.3.0-227/0/hive-log4j.properties
```

```
# hive> CREATE EXTERNAL TABLE IF NOT EXISTS employeeext (
    > UserID Int, Name String ,PhoneNo String,CardNo String,Amount Int)
    > COMMENT 'Employee details'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ","
    > LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE
    > LOCATION '/user/hive/ext/';
OK
Time taken: 7.782 seconds
```

```
# hive> SELECT * FROM employeeext;
OK
11223344        Vivek   9898054422      5039492020013110        100
12345678        Denis   9898054421      5039492020013110        2000
10001000        Ganesh  9898054423      5039492020013110        100000
11112222        Eric    9898054424      5039492020013110        500
33334444        Aengus  9898054425      5039492020013120        1000
44443333        Richard 9898054426      5039493320013110        201
44445555        Cristina        9898054427      5049493320013110        100000000
Time taken: 11.542 seconds, Fetched: 7 row(s)
```

8.2.2. ECS as a secondary FS

****Note:** When ECS is not the default FS, you will have to tell Hive to acquire the delegation token for the ECS bucket.

```
# hive> set mapreduce.job.hdfs-
servers=viprfs://hdpuser1bucket1.paris.Site1,hdfs://hdp2.paris.lab:8020;
```

For now, it works when Hive uses MR.

```
hive> set hive.execution.engine=mr;
```

The *mapreduce.job.hdfs-servers* parameter can also be set at the Hadoop cluster level, by adding it under *Custum mapred-site*.

8.2.2.1. Internal tables

Copy the table to the external FS (ECS, *hdpuser1bucket1*), as *hive* user.

```
# su hive
# cd /
# vi table.csv
```

```
# kinit -kt /etc/security/keytabs/hive.service.keytab hive/hdp2.paris.lab@PARIS.LAB
```

```
# hdfs dfs -copyFromLocal table.csv viprfs://hdpuser1bucket1.paris.Site1/
```

Enter the *hive* shell:

```
# hive
WARNING: Use "yarn jar" to launch YARN applications.
Logging initialized using configuration in file:/etc/hive/2.4.3.0-227/0/hive-log4j.properties
```

Create an internal table called *employee2.*

```
# hive> CREATE TABLE IF NOT EXISTS employee1 (
    > UserID Int, Name String ,PhoneNo String,CardNo String,Amount Int)
    > COMMENT 'Employee details'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ","
    > LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE
    > ;
OK
Time taken: 8.925 seconds
```

And load the data into it:

```
# hive> LOAD DATA INPATH 'viprfs://hdpuser1bucket1.paris.Site1/table.csv' OVERWRITE INTO
TABLE employee1 ;
Loading data to table default.employee1
Table default.employee1 stats: [numFiles=1, totalSize=346]
OK
Time taken: 16.363 seconds
```

Verify you can select info from the table:

```
# hive> SELECT * FROM employee1;
OK
11223344        Vivek   9898054422      5039492020013110        100
12345678        Denis   9898054421      5039492020013110        2000
10001000        Ganesh  9898054423      5039492020013110        100000
```

```
11112222        Eric    9898054424      5039492020013110        500
33334444        Aengus  9898054425      5039492020013120        1000
44443333        Richard 9898054426      5039493320013110        201
44445555        Cristina        9898054427      5049493320013110        100000000
Time taken: 11.758 seconds, Fetched: 7 row(s)
```

8.2.2.2. External tables

Copy the table to the external FS (ECS, hdpuser1bucket1), as *hive* user.

```
# su hive
# cd /
# vi table.csv
```

```
# kinit -kt /etc/security/keytabs/hive.service.keytab hive/hdp2.paris.lab@PARIS.LAB
```

```
# hdfs dfs -mkdir viprfs://hdpuser1bucket1.paris.Site1/user/hive/ext1
```

```
# hdfs dfs -copyFromLocal table.csv viprfs://hdpuser1bucket1.paris.Site1/ext1
```

Enter the *hive* shell:

```
# hive
WARNING: Use "yarn jar" to launch YARN applications.
Logging initialized using configuration in file:/etc/hive/2.4.3.0-227/0/hive-log4j.properties
```

```
# hive> CREATE EXTERNAL TABLE IF NOT EXISTS employeeext1 (
    > UserID Int, Name String ,PhoneNo String,CardNo String,Amount Int)
    > COMMENT 'Employee details'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ","
    > LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE
    > LOCATION 'viprfs://hdpuser1bucket1.paris.Site1/ext1/';
OK
Time taken: 1.391 seconds
```

```
# hive> SELECT * FROM employeeext1;
OK
11223344        Vivek   9898054422      5039492020013110        100
12345678        Denis   9898054421      5039492020013110        2000
10001000        Ganesh  9898054423      5039492020013110        100000
11112222        Eric    9898054424      5039492020013110        500
33334444        Aengus  9898054425      5039492020013120        1000
44443333        Richard 9898054426      5039493320013110        201
44445555        Cristina        9898054427      5049493320013110        100000000
Time taken: 11.738 seconds, Fetched: 7 row(s)
```

8.3. Spark tests

Apache Spark is a fast and general engine for large-scale data processing.

8.3.1. ECS as the default FS

- In your Spark client:

```
# cd /usr/hdp/current/spark-client/
# ./bin/spark-shell --master yarn-client --driver-memory 512m --executor-memory 512m
```

```
scala> val tweets = sqlContext.read.json("/user/hdpuser1/tweets.json")
scala> tweets.printSchema()
scala> tweets.registerTempTable("tweets")
scala> val users = sqlContext.sql("SELECT user.name FROM tweets")
scala> users.count()
scala>
sqlContext.read.json("/user/hdpuser1/tweets.json").write.format("parquet").save("/user/hdpuser1/par
quet")
```

8.3.2. ECS as a secondary FS

As the *hdfs* user:

```
# kinit -kt /etc/security/keytabs/hdfs.headless.keytab hdfs-hdp2@PARIS.LAB
# hdfs dfs -mkdir /user/hdpuser2
# hdfs dfs -chown hdpuser2:hadoop /user/hdpuser2
```

In your Spark client:

```
# cd /usr/hdp/current/spark-client/
#./bin/spark-shell --master yarn-client --driver-memory 512m --executor-memory 512m
```

****Note:** When ECS is not the default FS, you will have to tell Spark to acquire the delegation token for the ECS bucket.

```
spark.yarn.access.namenodes=viprfs://hdpuser2bucket1.paris.Site1,hdfs://hdp2.paris.lab:8020
scala> val tweets = sqlContext.read.json("viprfs://hdpuser2bucket1.paris.Site1/tweets.json")
scala> tweets.printSchema()
scala> tweets.registerTempTable("tweets")
scala> val users = sqlContext.sql("SELECT user.name FROM tweets")
scala> users.count()
scala>
sqlContext.read.json("viprfs://hdpuser2bucket1.paris.Site1/tweets.json").write.format("parquet").save
("viprfs://hdpuser2bucket1.paris.Site1/parquet")
```

8.4. HBase tests

Apache HBase is an open-source, distributed, versioned, non-relational database used when you need random, realtime read/write access to your Big Data.

8.4.1. ECS as the default FS

```
#vi table.csv
11223344,Vivek,9898054422,5039492020013110,100
12345678,Denis,9898054421,5039492020013110,2000
10001000,Ganesh,9898054423,5039492020013110,100000
11112222,Eric,9898054424,5039492020013110,500
```

```
33334444,Aengus,9898054425,5039492020013120,1000
44443333,Richard,9898054426,5039493320013110,201
44445555,Cristina,9898054427,5049493320013110,100000000
```

```
# su hbase
# cd /
# vi table.csv
```

```
# kinit -kt /etc/security/keytabs/hbase.service.keytab hbase/hdp2.paris.lab@PARIS.LAB
# hdfs dfs -copyFromLocal table.csv /
# hbase shell
```

hbase(main):001:0> **create 'employee_hbase','UserId','Name','PhoneNo', 'CardNo', 'Amount'**

****Note:** Workaround if you get a permission error.

```
# hdfs dfs -mkdir /user/hbase/
# hdfs dfs -chown hbase:hadoop /user/hbase/
```

```
# hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=,  -
Dimporttsv.columns=HBASE_ROW_KEY,Name,PhoneNo,CardNo,Amount employee_hbase
/table.csv
```

Other HBASE tests:

```
# hbase shell
```

```
# hbase(main): 001:0> scan 'employee_hbase'
# hbase(main):004:0> count  'employee_hbase'
# hbase(main):005:0> deleteall 'employee_hbase', '44445555'
# hbase(main):006:0> scan 'employee_hbase'
# hbase(main):007:0> get 'employee_hbase', '44443333'
# hbase(main):008:0> put 'employee_hbase' , '120000', 'Name', 'Rocky'
# hbase(main):009:0> scan 'employee_hbase'
# hbase(main):010:0> get 'employee_hbase' , '120000', 'Name'
# hbase(main):011:0> get 'employee_hbase' , '120000'
# hbase(main):012:0> put 'employee_hbase', '10000001', 'Name', 'schwazeneger'
# hbase(main):013:0> put 'employee_hbase', '10000001', 'Name','Shivajinagar'
# hbase(main):014:0> get 'employee_hbase', '10000001'
# hbase(main):015:0> scan 'employee_hbase'
```

8.4.2. ECS as a secondary FS

```
# su hbase
# cd /
# vi table.csv
```

```
kinit -kt /etc/security/keytabs/hbase.service.keytab hbase/hdp2.paris.lab@PARIS.LAB
```

```
# hbase shell
```

```
# hbase(main):023:0* create 'employee','UserId','Name','PhoneNo', 'CardNo', 'Amount'
```

```
# hdfs dfs -copyFromLocal 'table.csv' 'viprfs://hdpuser1bucket1.paris.Site1/'
```

```
# hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=,  -
Dimporttsv.columns=HBASE_ROW_KEY,Name,PhoneNo,CardNo,Amount employee
viprfs://hdpuser1bucket1.paris.Site1/table.csv

# hbase shell
```

```
# hbase(main):001:0> scan 'employee'
```

Other HBASE tests:

```
# hbase>deleteall 'employee', '44445555'
# hbase> scan 'employee'
# hbase> get 'employee', '44443333'
# hbase(main):001:0> put 'employee' , '120000', 'Name', 'Rocky'
# hbase(main):002:0> get 'employee' , '120000', 'Name'
# hbase(main):003:0> get 'employee' , '120000'
# hbase(main):008:0> get 'employee', '10000001'
# hbase(main):013:0* put 'employee', '10000001', 'Name','Shivajinagar'
# hbase(main):014:0> get 'employee', '10000001'
```

9. Benchmarks

Several type of benchmarks can be used to determine what is the best framework in the best configuration to meet customer expectations.

This section introduces benchmarking and testing tools that are included in the Apache Hadoop distribution, such as teskDFSIO, TeraSort, Hive-testbench and Spark-perf. These tools stress the Hadop cluster to measure its performance. It's useful to compare results among native FS, ECS as the default FS and ECS as a secondary FS, in order to find the best cost-effective solution for the customer.

9.1. TestDFSIO

The TestDFSIO benchmark is a read and write test for HDFS. It is helpful for tasks such as stress testing HDFS, to discover performance bottlenecks in your network, to shake out the hardware, OS and Hadoop setup of your cluster machines (particularly the NameNode and the DataNodes) and to give you a first impression of how fast your cluster is in terms of I/O.

TestDFSIO can only be used to perform benchmarks on the default FS, so it's important to configure ECS HDFS as the default FS on HDP.

9.1.1. Preparation

The *mapreduce.tar.gz* file must be uploaded to the ECS bucket that is used as default FS:

```
# hdfs dfs -mkdir -p viprfs://hdp22.paris.Site1/hdp/apps/2.4.3.0-227/mapreduce/
# hdfs dfs -copyFromLocal -f /usr/hdp/2.4.3.0-227/hadoop/mapreduce.tar.gz
viprfs://hdp22.paris.Site1/hdp/apps/2.4.3.0-227/mapreduce/mapreduce.tar.gz
```

9.1.2. Write

The example below shows how to write 100 x 128 MB files:

```
# /usr/hdp/current/hadoop-client/bin/hadoop jar /usr/hdp/2.4.3.0-227/hadoop-mapreduce/hadoop-
mapreduce-client-jobclient-2.7.1.2.4.3.0-227-tests.jar TestDFSIO - -write -nrFiles 100 -size 128MB
```

9.1.3. Read

The example below shows how to read 100 x 128 MB files:

```
# /usr/hdp/current/hadoop-client/bin/hadoop jar /usr/hdp/2.4.3.0-227/hadoop-mapreduce/hadoop-
mapreduce-client-jobclient-2.7.1.2.4.3.0-227-tests.jar TestDFSIO - -read -nrFiles 100 -size 128MB
```

9.2. Teragen/Terasort/Teravalidate

The TeraSort benchmark is probably the most well-known Hadoop benchmark. It sorts 1TB of data (or any other amount of data you want) as fast as possible. It is a benchmark that combines testing the HDFS and MapReduce layers of an Hadoop cluster.

A full TeraSort benchmark run consists of the following three steps:

- Generating the input data via TeraGen.
- Running the actual TeraSort on the input data.
- Validating the sorted output data via TeraValidate.

Teragen/Terasort/Teravalidate can be used to perform benchmarks on the default FS or on an external FS.

9.2.1. Preparation

The directory used for the different steps must be created on ECS HDFS:

```
# hdfs dfs -mkdir viprfs://hdp22.paris.Site1/tmp/teragen10GB
```

9.2.2. Write

The example below shows how to write 10 GB of data:

```
/usr/hdp/current/hadoop-client/bin/hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples-*.jar teragen -Dmapreduce.map.log.level=INFO -Dmapreduce.reduce.log.level=INFO -Dyarn.app.mapreduce.am.log.level=INFO -Dio.file.buffer.size=131072 -Dmapreduce.map.cpu.vcores=1 -Dmapreduce.map.java.opts=-Xmx4307m -Dmapreduce.map.maxattempts=1 -Dmapreduce.map.memory.mb=5743 -Dmapreduce.map.output.compress=true -Dmapreduce.map.output.compress.codec=org.apache.hadoop.io.compress.Lz4Codec -Dmapreduce.reduce.cpu.vcores=1 -Dmapreduce.reduce.java.opts=-Xmx1536m -Dmapreduce.reduce.maxattempts=1 -Dmapreduce.reduce.memory.mb=2048 -Dmapreduce.task.io.sort.factor=100 -Dmapreduce.task.io.sort.mb=256 -Dyarn.app.mapreduce.am.command-opts=-Xmx768m -Dyarn.app.mapreduce.am.resource.mb=1024 -Dmapreduce.job.maps=64 104857600 viprfs://hdp22.paris.Site1/tmp/teragen10GBtunning/out
```

Note: 10 GB = 104857600 x 100 bytes

All the parameters can be tweaked to improve performance, but the example above is showing those that we recommend using as a starting point.

9.2.3. Sort

The example below shows how to sort 10 GB of data:

```
# /usr/hdp/current/hadoop-client/bin/hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples-*.jar terasort -Dmapreduce.map.log.level=INFO -Dmapreduce.reduce.log.level=INFO -Dyarn.app.mapreduce.am.log.level=INFO -Dio.file.buffer.size=131072 -Dmapreduce.map.cpu.vcores=1 -Dmapreduce.map.java.opts=-Xmx1536m -Dmapreduce.map.maxattempts=1 -Dmapreduce.map.memory.mb=2048 -Dmapreduce.map.output.compress=true -Dmapreduce.map.output.compress.codec=org.apache.hadoop.io.compress.Lz4Codec -Dmapreduce.reduce.cpu.vcores=1 -Dmapreduce.reduce.java.opts=-Xmx1536m -Dmapreduce.reduce.maxattempts=1 -Dmapreduce.reduce.memory.mb=2048 -Dmapreduce.task.io.sort.factor=3000 -Dmapreduce.task.io.sort.mb=768 -Dyarn.app.mapreduce.am.command-opts=-Xmx768m -Dyarn.app.mapreduce.am.resource.mb=1024 -Dmapreduce.job.reduces=128 -Dmapreduce.terasort.output.replication=1 viprfs://hdp22.paris.Site1/tmp/teragen10GBtunning/out viprfs://hdp22.paris.Site1/tmp/teragen10GBtunning/sorted
```

9.2.4. Read

The example below shows how to read 10 GB of data:

```
# /usr/hdp/current/hadoop-client/bin/hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples-*.jar teravalidate -Dmapreduce.map.log.level=INFO -Dmapreduce.reduce.log.level=INFO -Dyarn.app.mapreduce.am.log.level=INFO -Dio.file.buffer.size=131072 -Dmapreduce.map.cpu.vcores=1 -Dmapreduce.map.java.opts=-Xmx1536m -Dmapreduce.map.maxattempts=1 -Dmapreduce.map.memory.mb=2048 -Dmapreduce.map.output.compress=true -Dmapreduce.map.output.compress.codec=org.apache.hadoop.io.compress.Lz4Codec -Dmapreduce.reduce.cpu.vcores=1 -Dmapreduce.reduce.java.opts=-Xmx1536m -Dmapreduce.reduce.maxattempts=1 -Dmapreduce.reduce.memory.mb=2048 -Dmapreduce.task.io.sort.factor=100 -Dmapreduce.task.io.sort.mb=256 -Dyarn.app.mapreduce.am.command-opts=-Xmx768m -Dyarn.app.mapreduce.am.resource.mb=1024 -Dmapreduce.job.maps=64 -Dmapreduce.terasort.output.replication=1 viprfs://hdp22.paris.Site1/tmp/teragen10GBtunning/sorted viprfs://hdp22.paris.Site1/tmp/teragen10GBtunning/validated
```

9.3. Hive-testbench

The hive-testbench is a data generator and set of queries that lets you experiment with Apache Hive at scale. The testbench allows you to experience base Hive performance on large datasets, and provides an easy way to see the impact of Hive tuning parameters and advanced settings.

Hive-testbench can only be used to perform benchmarks on the default FS, so it's important to configure ECS HDFS as the default FS on HDP.

9.3.1. Preparation

Download and build Hive-testbench:

```
# yum install java-1.8.0-openjdk-devel.i686
# git clone https://github.com/hortonworks/hive-testbench.git
# cd hive-testbench
#./tpcds-build.sh
```

Generate the data with the scale factor of your choice (3 in our example):

```
# export JAVA_HOME=/usr/jdk64/jdk1.8.0_60/
# export PATH=$JAVA_HOME/bin:$PATH
#./tpcds-setup.sh 3
```

A scale factor of 3 generate 3 GB of user data, but is using less than 2 GB on disks as the data is stored in a compressed format.

Rename 2 SQL queries that are known to be far too long to execute:

```
# cd sample-queries-tpcds
# mv query13.sql query13.sql.skip
# mv query48.sql query48.sql.skip
```

If you have manually the warehouse data from standard HDFS (using *hadoop distcp*), you need to update the Hive Metastore using the commands below:

```
# export HIVE_CONF_DIR=/etc/hive/conf/conf.server
# hive --service metatool -updateLocation viprfs://hdp22.paris.Site1 hdfs://hdp2.paris.lab:8020
```

9.3.2. Run

First of all, you should run a single query to validate that everything is correctly configured:

```
# hive -i testbench.settings
# hive> use tpcds_bin_partitioned_orc_3;
# hive> source query12.sql;
```

Then, you can run the full test suite:

```
# cd ..
#./runSuite.pl tpcds 3
```

9.3.3. Analyze

You can use the script below to consolidate the results from the logs:

```
# ls -1 *log | while read log;do
  grep "Time take" $log | grep row | awk -v file=$log '{
    split(file,test,".")
    print test[1]","$3
  }'
done
```

9.1. Spark-perf tests

Spark-perf is a suite of performance test for Spark. It allows to parametrize different test configurations to evaluate Spark performance in different sceranios.

9.1.1. Preparation

- Install the EPEL package

```
# wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
# rpm -ivh epel-release-latest-7.noarch.rpm
```

- Download *spark-perf*

```
# git clone https://github.com/databricks/spark-perf.git
```

- Install *python2-pip*

```
# yum install python2-pip
```

If *python2-pip* can't be installed this way, it can be done using the following commands

```
# curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
# python2 get-pip.py
```

- Install *argparse*

```
# easy_install argparse
```

- Configure *spark-perf*

```
# cd spark-perf
# cp config/config.py.template config/config.py
# vi config/config.py
```

Here are the parameters to use for HDP 2.4.3 with a scale of 0.001:

```
SPARK_HOME_DIR = "/usr/hdp/2.4.3.0-227/spark"
SPARK_CLUSTER_URL = "yarn"
SCALE_FACTOR = 0.001
SPARK_DRIVER_MEMORY = "1g"
2 x JavaOptionSet("spark.executor.memory", ["1g"]),
```

9.1.2. Run

```
#./bin/run
Detected project directory: /root/spark-perf
Loading configuration from /root/spark-perf/config/config.py
WARNING: No slaves file found at path: /usr/hdp/2.4.3.0-227/spark/conf/slaves
      ...We will assume no slaves exist.

About to remove all files and directories under /usr/hdp/2.4.3.0-227/spark/work on ['localhost'], is this
ok? [y, n] y
ssh -o StrictHostKeyChecking=no -o ConnectTimeout=5 localhost 'rm -r /usr/hdp/2.4.3.0-
227/spark/work/*'
…
Result: scala-count-w-fltr, count-with-filter --num-trials=10 --inter-trial-wait=3 --num-partitions=1 --
reduce-tasks=1 --random-seed=5 --persistent-type=memory  --num-records=200000 --unique-
keys=20 --key-length=10 --unique-values=1000 --value-length=10  --storage-
location=hdfs://hdp2.paris.lab:9000/test//spark-perf-kv-data, 0.1505, 0.021, 0.128, 0.199, 0.148

-----------------------------------------------------------------
Finished running 8 tests in Spark-Tests.
See summary in results/spark_perf_output__2017-03-19_15-49-23

Number of failed tests: 0, failed tests:
-----------------------------------------------------------------
Finished running all tests.
```

9.1.3. Analyze

The results are available under the *results* directory.

```
# cat results/spark_perf_output__2017-03-31_12-42-35
# Test name, test options, median, std dev, min, first, last
scheduling-throughput, scheduling-throughput --num-trials=10 --inter-trial-wait=3 --num-tasks=10000
--num-jobs=1 --closure-size=0 --random-seed=5, 23.709, 5.777, 20.815, 23.121, 34.9

scala-agg-by-key, aggregate-by-key --num-trials=10 --inter-trial-wait=3 --num-partitions=1 --reduce-
tasks=1 --random-seed=5 --persistent-type=memory  --num-records=200000 --unique-keys=20 --
key-length=10 --unique-values=1000 --value-length=10  --storage-
location=hdfs://hdp2.paris.lab:9000/test//spark-perf-kv-data, 0.2635, 0.051, 0.248, 0.321, 0.331
```

[…]

You can see in bold what the numbers mean. The lower the better.

10. Conclusion

This document describes how to configure Hortonwoks, ECS and Kerberos to offer a Big Data solution for customer interested in running analytics in data stored in a data lake, or for customers that just want to use ECS as the primary storage, running analytics in a platform that support multiprotocol access. The ECS configurations and customer alternatives are huge.

Big Data customers who are dealing with massive growth of large unstructured data sources need a storage architecture alternative that provides scalability, flexibility and ease of management. ECS is a perfect fit for that. The objective of this paper is to reinforce the idea of using Dell EMC ECS as the storage platform option for Big Data environments, taking advantage of what object storage offers, and showing all the implementations alternatives for this new solution.

11. References

[1] – DellEMC Elastic Cloud Storage - http://www.dellemc.com/en-us/storage/ecs/index.htm#collapse
[2] – ECS 3.1 Product Documentation - https://community.emc.com/docs/DOC-56978
[3] – ECS Data Access Guide - https://www.emc.com/collateral/TechnicalDocument/docu86295.pdf
[4] – ECS Administration Guide - https://www.emc.com/collateral/TechnicalDocument/docu86293.pdf
[5] – Hortonworks Website - https://es.hortonworks.com/
[6] – Kerberos - https://web.mit.edu/kerberos/
[7] – Apache Distcp - https://hadoop.apache.org/docs/current/hadoop-distcp/DistCp.html
[8] – Apache Hive - https://hive.apache.org/
[9] – Apache Spark - https://spark.apache.org/
[9] – Apache HBase - https://hbase.apache.org/
[10] – Test DFSIO - https://github.com/facebookarchive/hadoop-20/blob/master/src/test/org/apache/hadoop/fs/TestDFSIO.java
[11] – Hive-testbench - https://github.com/hortonworks/hive-testbench
[12] – Spark-perf - https://github.com/databricks/spark-perf
[13] – ECS Support Website - https://support.emc.com