Exercise 4 – Encrypting a cluster

Last Updated: May 28, 2014; Copyright©2011-2014 by Cloudera, Inc.

Overview

Planning

When encrypting an entire Hadoop cluster, it is important to consider three things:

- The number of data nodes in the cluster – This will be the number of zNcrypt clients that you will need to install.
- The number of HDFS drives per data node – This will be the number of encrypted mount-points per zNcrypt client.
- The amount of raw data that is on each HDFS drive – This will be the determining factor in how long the encryption process will take.

By taking into account the three factors listed above, we can start to plan:

- How we should install the zNcrypt clients, and whether or not to use a deployment tool (Parcels, Chef, Ansible, Puppet, etc).
    - If this is a small test/demo cluster, then it might not be necessary that we spend the time and bandwidth configuring Chef/Puppet, for the install.
    - If this is a large prod/QA cluster, then we will definitely want to spend the extra time setting up a repeatable process to configure each data node.
- What form of encryption we should use (file-level or block-level).
    - For small test clusters, file-level encryption should be sufficient.
    - For large production clusters (or anything that will be used to provide benchmarks), block-level encryption should be used over file-level encryption.
        - ***IMPORTANT!*** When using block-level encryption, you will need to consider the amount of data *currently* on the HDFS drives that will need to be encrypted. In order to use block-level encryption, you will need to reformat each device that will be used to store encrypted data. This means that you will either need to:
            - Migrate the data off of the drives you will want to encrypt (which will incur its own time and maintenance overhead for copying the data off then back on).
            - Reformat the drive without migrating the data, and then let HDFS replicate the data back through the cluster (can incur a time overhead to wait for replication).
- How long the entire process (installation through encryption) will take, and whether or not we will need to allocate downtime to a specific set of nodes (or the entire cluster).

- For small clusters (or clusters that are not being used), this might not be an issue. You can take each node down one at a time to encrypt, or you can take the entire cluster down.
- For large production clusters (or clusters that are in active use), you will need to plan how/when to take each node down for encryption. Depending on how in-use the cluster is, it might be more efficient to request downtime for a subset of nodes, and work your way through the cluster.

Now that we have a "plan of attack" on encrypting the cluster, it's time to execute our plan.

Bash scripting

If you would like to use Bash scripts to deploy and configure zNcrypt, we have a Github repository filled with scripts to help here. With the important ones being:

- zncrypt-install.sh– This script can be used to install zNcrypt on almost every platform zNcrypt is supported (requires Internet access).
- zncrypt-autoconfigure.sh– This script can be used to configure zNcrypt for every step of the process cited above. You specify your environmental attribute at the top of the script, and then execute after stopping the data node process. A sample configuration to accomplish the example steps above (using file-level encryption) might look similar to

  WARNING! When using the 'zncrypt-autoconfigure.sh' script, you will want to stop the 'hadoop-hdfs-datanode' process on all target machines **before** executing the script. The datanode directories might still be in use, and could potentially corrupt the file system if left active during encryption.

  Note: zncrypt-autoconfigure.sh creates an ACL with just the java process which would allow any Java process to access the data. Once the cluster is encrypted, you will want to create a process based ACL and then delete the ACL created by the script.

  Test the script on a single cluster. The following should be the minimum things to address:

  1. Is previous data still present?
  2. Do you see any issues in CM?
  3. Does dmesg report any issues?
  4. Can you add data to the cluster with block written to the DN?

Step 1 – Configure shell scripts

For this exercise, the **zncrypt-autconfigure.sh** script was modified to implement a step by step process for the encryption. The configuration is in **zncrypt_config.sh** and the register and prepare were separated from the encryption step.

To prepare the scripts, configure the following shell scripts to run the Encrypt installation:

1. **admin_setSSH_key.sh** – this script is sourced as part of each script and sets the SSH user and the key file to use for logins to each node in the cluster.

2. hostnames.txt – this file is a list of the hostnames that zncrypt will be installed and configured on.

3. **Zncrypt_config.sh** – this script set variables used for the encryption on each node.

The parameters include the credentials for the Key Trustee server, the encrypted mount points and storage locations and the category for the hadoop ACLs. Note that "storage", "mount" and "to_encrypt" can be lists to handle multiple directories.

Note: This cluster should be running for steps 1 through 3.

Step 2 – Verify encryption directories

Run the script **1_verify_dn_dirs.sh.** This will verify that you have the host names correctly entered, SSH is set up properly. It performs an "ls" on each of the "to_encrypt" directories.

Step 3 – Install zncrypt

Run the script **2_zncrypt_install.sh.** This script will install Encrypt on each of the target hosts. It does not configure zncrypt.

Step 4 – Register and prepare encryption drives

Run the script **3_prepare.sh.** This script will install register zncrypt with the Key Trustee server and then prepare the encryptions storage and mount points and create the process based ACLs with profiles for the datanode, namenode and secondary namenode.

Verify the mount points are created with the script **admin_df.sh**. You should see the mount point(s) specified in the config file.

Verify the ACLs are created with the script **admin_showACLs.sh**. You should see the ACLs for the

Step 5 – Stop the cluster

Using Cloudera Manager, stop the cluster. Run the script **4_verifydn_down.sh** to verify that all processes are down before proceed**.**

Step 6 – Encrypt the cluster

WARNING: Performing this step with the cluster running can lead to corruption.

Run the script **5_encypt.sh.** This script will install register zncrypt with the Key Trustee server and then prepare the encryptions storage and mount points and create the process based ACLs with profiles for the datanode, namenode and secondary namenode.

Note: if you did this step with the cluster up, the **admin_fsck.sh script will perform an hadoop** fsck on each node to attempt to fix any corruption.

Step 7 – Encrypt the Metadata DB

WARNING: Performing this step with the cluster running can lead to corruption.

Examine the script **6_encrypt_postgres.sh** and ensure the directories are appropriate for your installation**.** The default host is the first entry in the hostnames.txt file; but, you can enter the host that the DB server is running on.

There is also a 6_encrypt_mysql.sh if the cluster uses mysql for the metadata store.

Step 8 – Start the cluster

Clear the dmesg on each host with the **admin_cleardmesg.sh** script.

Start the cluster with Cloudera Managerr. While the cluster is booting, monitor dmesg output with **admin_showdmesg.sh**.

Step 9 - Post Installation steps

Test haddop functions such as ls, mkdir, put and get to ensure the ACLs are properly done and working.

Continue to monitor with **admin_showdmesg.sh** to identify processes that need ACLs. You can set the entire cluster to permissive mode with **admin_setpermissive.sh** for debugging and to allow the process to start and gather ACLs. **Do not run large jobs while in permissive mode** because this would impact performance and generate large log files in dmesg. When you're done debugging, use the script **admin_setenforcing.sh** to set the encryption back to enforcing ACLs.

And, lastly, you need to rotate the master passphrase on every machine since it was stored in the file system on each machine during the install using the **zncrypt key –set** command

Support

Issues? Comments? Questions? Just want to say hi?

Feel free to contact our awesome support team at support@gazzang.com.