AM2CM: Design Document

Overview

Cloudera Manager (CM) Deployment JSON stores the entire state of the Database of the Cloudera Manager. This includes all CM entities, along with their non-default configurations, hosts information and authorization/user information. If used correctly, the deployment JSON can be used as a starting point for the CM to takeover an Ambari governed cluster.
For this to work, the Ambari blueprint of the same cluster will need to be transformed into an acceptable footprint that CM can consume. The tool to carry out this translation has been coined as *AM2CM* (Ambari to Cloudera Manager).

Assumptions:

- Binaries should be upgraded by Ambari to CDP DC 7.1 Runtime.
- The cluster is going to have significant downtime.
- Services considered for migration:
    - Services that do not exist in Cloudera Manager will not be migrated.
    - Base (Storage, Security ergo SDX) services will be migrated at first.
    - Compute and workflow services will be added later as the procedure evolves.

Goals:

- A bring up of an Ambari HDP cluster should be possible from Cloudera Manager. The set of hosts used may or may not be the same (some will be i.e. HDFS etc)
- Workloads involving services that are migrated (list below) should be management plane agnostic. Ex. queries should have same output irregardless if the cluster is managed by Ambari or CM

Non Goals:

- Upgrade to Ambari CDP DC 7.1 from any other previous versions will be done by Ambari, and not by Cloudera Manager

Approach

Ambari blueprint along with a set of mapping files will be used to create a cloudera manager deployment template. This template will help learn the CM about the existing cluster footprint.

1. Install Cloudera Manager on the same host as Ambari.

2. Add the cluster hosts with Cloudera Manager -> Hosts -> Add Hosts. This step will install the CM agent on the same Ambari hosts. This will verify CM is able to communicate with the existing hosts without any port/permission interference.

3. Feed the existing Ambari blueprint to the Ambari-CM converter (AM2CM).
   The Ambari-CM converter currently:
   - Convert Ambari parameter configurations into CM configuration using a static lookup. Parameters that do not have an equivalent configuration will be added to the safety valve of the service.
   - Convert Ambari roles and services into equivalent roles on the CM side. Roles that do not have a 1-1 mapping will be added manually post-migration.
   - Fetch the host-Id and generate the role names/service names for naming compatibility.
   - Specify the parcel version of the cluster (This will be the GA version of the supported release)

4. Import the Ambari-CM output into Cloudera Manager via. the API and verify the cluster entity shows up.

5. Verify the imported deployment. Fix any configuration errors. Verify warnings and errors.

6. Download and distribute the parcels on the hosts added to the cluster.

7. Stop the services from the Ambari UI.

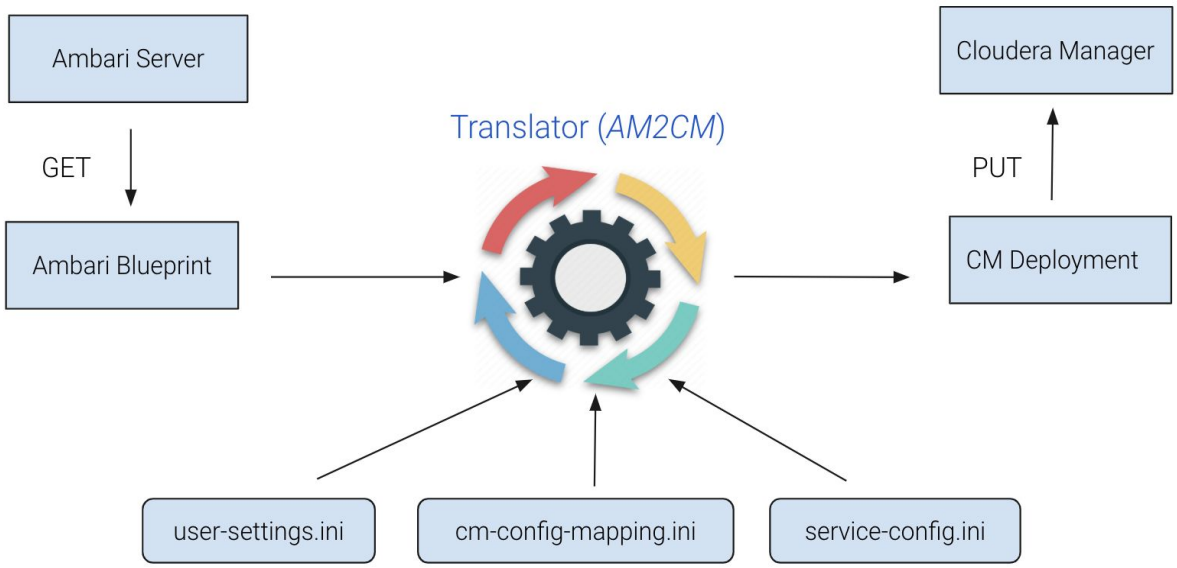
8. Verify the services configuration in Cloudera Manager UI. Add any new services and set dependencies if required.

9. Verify the Kerberos and TLS Configuration. Enable Kerberos/TLS if the cluster needs to be secure.

10. Start the services from the Cloudera Manager UI.

11. Cleanup Ambari and HDP bits.

AM2CM tool



AM2CMl uses Ambari blueprint to generate a CM Deployment template. Blueprints do not have all the information that is needed to generate templates. It relies on a set of mappings defined in the .ini files.

cm-config-mapping.ini: Mapping for configuration name/value from Ambari to CM. This mapping translated an Ambari configuration to an equivalent CM understandable format. For example:

```
dfs_name_dir_list = dfs.namenode.name.dir
```

cm-config-mapping-custom.ini: Blueprints have a few Ambari specific variables. Configurations that do not fit into cm-config-mapping.ini go here. For example:

```
process_username=hdfs_user
```

service-config.ini: Blueprint does not have any service/config definitions, it only has config types and component information.

- Service-Config mappings : Available roles, Example:

  ```
  [CONFIGS-SERVICE-COMPONENT-MAPPINGS]
  HDFS=DATANODE,JOURNALNODE,NAMENODE,SECONDARY_NAMENODE,BALANCER,ZKC
  ,NFSGATEWAY
  ```

- Some Roles have different names in Ambari and CM. Example:
  [CONFIGS-COMPONENT-MAPPINGS]
  SECONDARY_NAMENODE=SECONDARYNAMENODE

- For Service configuration and site-files:
  [CONFIGS-SERVICE-CONFIGTYPE-MAPPINGS]
  HDFS=core-site,hadoop-env,hadoop-policy,hdfs-site,ranger-hdfs-audit,ranger-hdfs-plugin-properties,ranger-hdfs-policymgr-ssl,ranger-hdfs-security,ssl-client,ssl-server,viewfs-mount-table,cluster-env

- user-settings.ini: This file has other inputs like Parcel version, clustername, passwords etc.

Cloudera Manager Deployment template

Cloudera Manager deployment template has the entire state of the Cloudera Manager instance at at the instant it was exported. This includes cluster, service, roles, hosts, global settings, user information.

Clusters/services:

Tool reads service names from service-config.ini and iterates service by service. For every service it does the following.

- Blueprints config-groups configurations would go to roles.
- Role specific configurations go to role config groups.
- Service level configurations goto Service configurations group.

How does the config translation work:
Tool uses cm-config-mapping.ini and cm-config-mapping-custom.ini files for config translations. If the same config is present in both the files then cm-config-mapping-custom.ini takes higher priority.

- For any new configuration mappings , add in cm-config-mapping-custom.ini file
- If the new config belongs to Service level then then add it under
  - Printing param specs for XXX
- If the new config belongs to Role level then then add it under
  - Role param specs for XXX:
- The mappings can be done for Ambari-CM specific variables as well.
  - Ex:namenode_java_heapsize=namenode_heapsize
  -

What if no translation is defined:
If any config does not have any mapping in cm-config-mapping.ini and cm-config-mapping-custom.in files then it uses Safety valves.

- First look into Service level SafetyValve for that config-type - if found, then config goes there. Ex: hdfs_service_config_safety_valve = hdfs-site.xml
-

- If Service level does not have SafetyValve for that config type, look into Role level safety valve for that config type, if found it adds at role level SafetyValve.
  Ex: zookeeper_config_safety_valve = zoo.cfg

Adding new configs:
Ambari blueprints do not have all the configs needed to generate a template. Hence there is a need to have a way to add additional configs to the template. Configs can be added in 2 different ways. Type1 is reading the value from user-settings.ini - Database passwords are not present in the blueprint - hence this method can be used to capture the passwords. Type2 is to add new configs by mapping the value from the blueprint.

- Tyep1: read input from user-settings.ini file
  - Ex:Printing additional param specs for OOZIE_SERVER
  - oozie_database_password=INPUT::oozie_database_password

- Tyep2: By mapping config values from from blueprints
  - Printing additional param specs for ZOOKEEPER
  - enableSecurity = BP::cluster-env::security_enabled

Config value Translations to CM format:
CM does not accept some of the config values in the same way that Ambari exported into Blueprints. Tools code handle some of these config value translations. Each service has its own .java file to do this activity.
Ex: Ambari exports the heap size in MBs where as CM template expects the input in Bytes. Similarly ports, ratios.

Defining default config values:
Tool provides an option to define default config values for every service in service-default-config.ini file - this feature helps to avoid flooding un-mapped configs if it has a default config value.

Config Summary log:

Tool summarize and print all the logs that are mapped, ignored and SafetyValve added configs - this should help Component teams to fix missing config mappings.

Ex:

HDFS-HDFS

SAFETY-VALVE-(hadoop_policy_config_safety_valve)

security.inter.datanode.protocol.acl : *

security.refresh.usertogroups.mappings.protocol.acl : hadoop

security.admin.operations.protocol.acl : hadoop

SAFETY-VALVE-(core_site_safety_valve)

hadoop.proxyuser.hcat.groups : *

hadoop.proxyuser.hbase.groups : *

IGNORED-DEFAULT

dfs.cluster.administrators : hdfs

dfs.blockreport.initialDelay : 120

SAFETY-VALVE-(hdfs_service_config_safety_valve)

dfs.encrypt.data.transfer.cipher.suites : AES/CTR/NoPadding

dfs.client.retry.policy.enabled : false

dfs.block.access.token.enable : true

IGNORED

ipc.server.tcpnodelay : true ( )

MAPPED

ipc.client.idlethreshold (ipc_client_idle_threshold) : 8000

hadoop.security.authorization (hadoop_security_authorization) : true

Security

We need to enhance the process to take care of Kerberos and TLS aspects during migration.

TLS

The Ambari blueprint contains the TLS/SSL related configurations if SSL is enabled in the HDP cluster. These configurations if translated causes issues with enabling AutoTLS in Cloudera Manager. The *importDeployment* API also does not execute AutoTLS enablement workflow as a part of the import mainly because the API was always meant to restore the static configuration state of the cluster from database, and not execute commands in tandem to put the cluster in the original state. The commands were always user motivated after verifying the state of the cluster. Hence, the approach chosen here *is to not include any TLS/SSL configurations* in the cloudera manager deployment template. The user would then input any necessary TLS configuration post import and run the “Enable AutoTLS” command as a part of the Cluster Actions.

Kerberos

AM2CM generates a *deployment template with Kerberos Service configurations* alone - it does not include Kerberos configurations like Kerberos host, credentials and Keytab and Principal names. Before importing the Template admin needs to configure CM with KDC server. Once this template is imported into Cloudera Manager, it flags that keytabs are missing- admin needs to generate keytabs. CM takes care of adding Principal names and keytabs to the Services/hosts..

Services

Basic Core and *Core Extended* are the services planned to be supported for the MVP release of the AM2CM tool. Services other than these will be added as a post-translation task by the administrator.

Basic Core Extended	Core Extended	Basic HA	Basic Integrations	Extended
HDFS YARN MAP-REDUCE2 TEZ HIVE -Metastore -HS2	ZOOKEEPER HBASE SPARK2 -client - history server OOZIE SQOOP KERBEROS	KNOX HDFS - HA YARN - HA HIVE - HA -HS2 -Metastore LIVY2 Server	Ranger Admin Ranger Integrations -HDFS -YARN -HIVE -HBASE RANGER KMS -HDFS integrations	KAFKA ATLAS Ranger Atlas Plugin HBase HA Oozie HA Ranger Kafka Integration HDF

Below is the list of services that will be migrated for MVP along with Kerberos.

- HDFS - HA
- YARN - HA
- ZOOKEEPER
- HBASE
- KAFKA
- OOZIE
- SPARK2
- HIVE2
- TEZ
- Sqoop