CloudShe

Ayelet Aharon

Deployment guide  
cloudshell-orch-sandbox script

## Overview

The orchestration package should bring an out of the box solution for our customer’s common use cases:

* Setting up the sandbox upon reservation
* Tearing down sandbox
* Save a snapshot of a sandbox

### Setup

During the setup process the script will iterate over the resources and prepare them for the reservation.

* For each networking device:
  + Perform health check
  + Load firmware
  + Load configuration
  + Re-run health check
* VMs (when represented by Apps Deployment)
  + Deploy apps
  + Power on VM
* Activate all routes and connectors

### Teardown

During the teardown process the script will wipe the configuration from the networking devices and delete VMs bringing them back into “ready for use” state.

* For each networking device:
  + Perform health check
  + Load firmware
  + Load configuration
  + Re-run health check
* VMS
  + Delete/Power off VM
* Disconnect routes and connectors

### Snapshot

The user can save a snapshot of the sandbox. In the background, the script will save the sandbox as a new blueprint, and the current configuration of the devices and VMs will be saved for future use.

* + Save the sandbox as a blueprint
  + Save all the configuration files of the devices on the storage server (e.g. FTP server)

### Prerequisites

* Loading configuration/firmware,
  + Resource must have standard networking shells.
  + An instance of a storage resource (e.g. FTP) should exist in the blueprint
  + If loading image, the resource must have ‘get\_version’ command, returning the resource’s image version
  + Configuration/image files should be saved under a predefined folder structure and should confine to a naming standard. Detailed description can be found in the document

## Data model

The script requires specific resources and attributes. The data model can be imported from the orch-sandboxDataModel.xml file.

### Storage server family

Storage server should have the following rules:

* Admin only
* Searchable

Currently the following models are supported:

* FTP
* TFTP

#### ‘Generic FTP server’ model

Generic FTP server should have the following attributes

* Storage Port – Port being used to connect to the FTP server
* Storage Network configs Path – Root directory on the FTP server, where configuration files reside on
* Storage username – FTP username
* Storage password – FTP password

#### ‘Generic TFTP server’ model

Generic TFTP server should have the following attributes

* Storage Port – Port being used to connect to the TFTP server
* Storage username – TFTP username
* Storage password – TFTP password
* TFTP psexec – Path to psexec.exe file on the execution server
* TFTP Root – Root directory on the TFTP server, where configuration files reside on

### Pool family

Pool family should have the following rules:

* Admin only
* Searchable
* Locked By default

#### ‘Config Set Pool’ model

Config set pool attributes are varying between customers and should be created according to the use case.

It’s recommended to follow the suggested naming convention for pool attributes: *ConfPool\_<name>* e.g. ConfPool\_lab\_network\_subnet. This naming convention will ease the association of the attributes to the pool.

### New Attributes

#### ‘Disable Load Config’

* Type – Boolean
* Default value – False (Unchecked)
* Rules: Configuration, Admin only

This attribute can be associated with a resource. When value set to true (checked), the script will skip the device and will not try to load configuration/image onto it.

#### ‘Config file path’

* Type – String
* Rules: Configuration

This attribute can be associated with a resource, when working with abstract resources. The orchestration script will update this attribute with the path to the temporary configuration file created out of a template. Upon teardown, the script will use this value to delete the temporary configuration file.

## Blueprint template

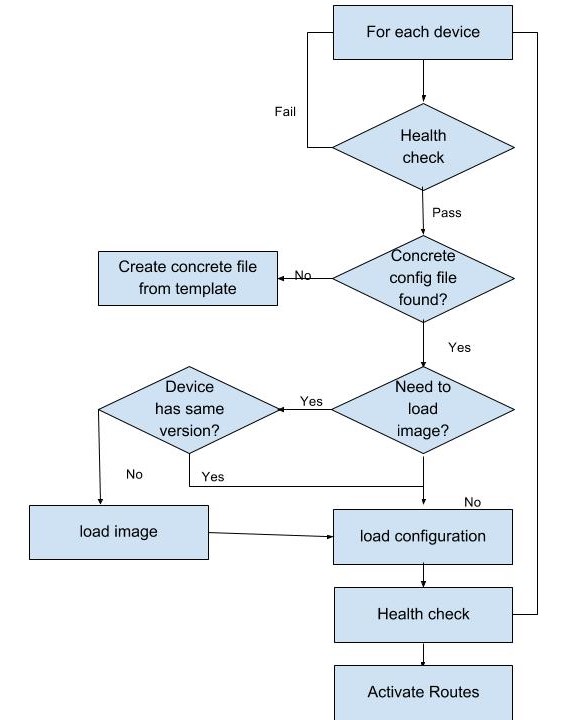
If loading configuration/image, it is recommended to create a blueprint template. The template will include the following:

* Setup orchestration script
* Teardown orchestration script
* Snapshot script
* An abstract FTP/TFTP resource (Visible to admins only), holding the data required to reach it (IP address, username, password, port, root folder)
* An abstract pool resource (Visible to admins only), holding topology data for the configuration file (e.g. lab network subnet, NTP server ip )

## Starting a reservation

When starting a reservation, in case of abstract devices, a resolution process begins according to the requirements described.

Once we have concrete resources, the orchestration driver will launch the following for networking devices:



Apps will be deployed as part of the ‘standard’ setup script which is embedded in the cloudshell-orch-sandbox script.

## Ending a reservation

For each networking device the orchestration driver will launch the following:

## 

For VMS that were deployed - If a snapshot was taken, the VM will power off, otherwise it will be deleted.

## Saving a snapshot

When the user is saving a snapshot, a name for the snapshot needs to be provided by the user. The orchestration script will save the sandbox as a new blueprint and would name it as follows: <original\_Blueprint\_name>\_<username>\_<snapshot name> e.g. LargeOffice\_aaharon\_snapshot1

The orchestration script will create a new folder on the storage server, under the following path:

<storage\_root\_dir>/Snapshots/<original\_Blueprint\_name>\_<username>\_<snapshot name> e.g. home/cloudshell/Snapshots/LargeOffice\_aaharon\_snapshot1

**Networking devices:**

The orchestration script will save the configuration of all networking devices under the created folder. The name for each file will be in the following format: <resource\_name>\_<model>.cfg

**VMs:**

An artifact file for the VM will be created under the snapshot’s directory.

A VM originating from an app, that is being saved as part of a snapshot will be marked and will **not** deleted at the end of reservation. The orchestration script will only power on/off the vm upon setup/teardown.

## Folder structure in storage server

The orchestration script assumes a specific folder structure on the storage server.

Under the root directory (Specified in the attribute ‘Storage Network configs Path’) the following folders should be created:

* **Gold** – Will hold config data used during the setup process
  + **<Blueprint\_name>** - This folder contains the setup data for the blueprint. It should contain the devices’ configuration files and ‘FirmwareData.csv’
* **Base** – Will hold config data used during the teardown process to the devices back to a ‘clean’ state. It should contain the devices’ configuration files and ‘FirmwareData.csv’
* **Snapshots** – Will hold the snapshots’ data
  + **<snapshot\_name>** - This folder contains the VM’s artifact file, devices’ configuration files and ‘FirmwareData.csv’

Under the ‘Gold’ directory, a folder with the **blueprint’s name** should be created..

The ‘Base’ directory should contain configuration files and ‘FirmwareData.csv’ bringing

## Configuration files:

The script has a capability of creating configuration file out of a template. The template file holds parameters in curly brackets. Once the resolution process is done and concrete devices are selected, the orchestration script will create concrete config files and replace the parameters from the template files with values taken from the resources or a config pool resource.

A configuration can be loaded on a device from the options below:

* A concrete config file associated with the device.
* A generic template file associated with a device role and model.
* A generic template file associated with a device model.

The script will first search for a concrete file, if there is no concrete file, it will search for a template associated with the role (alias) of the device and its model, if not found it will search for a template associated with the model.

Naming convention for config file:

* <Resource name>\_<model>.cfg - Used for concrete resource
* <Resource role>\_<model>.tm - Used for abstract resource
* <Model>.tm - A base configuration file for the model

### Config file templates

Template will hold variables in curly brackets within the file. The data for the variables will be taken from attributes that reside, either on a resource or from the configuration pool resource.

The variable naming convention indicates where the data resides on:

{Device.Self.Address} – Management ip

{Device.Self.Name} – Resource name

{Device.Self.Attribute\_name}

{ConfigPool.Attribute\_name}

**Examples of attributes:**

* lab network subnet
* NTP ip address

**Example of a variable in the config file:**

ip route {ConfigPool. ConfPool\_lab\_network\_subnet} {Device.Self.Gateway IP Address}

## Image files

A csv file will be created mapping between a device/model to an image. The name of the file should be **‘FirmwareData.csv’** and it should reside under the Gold/Base folder. The file contains mapping between a device name/model to a firmware image.

If there is a mapping between a specific device to its image, the script will load the image from the given path, otherwise it will look for a mapping between the device’s model to an image file and load that image to the device

Naming convention to be used in the ‘device’ column:

<Resource role>\_<model> - Used for abstract resource

<Resource name>\_<model> - Used for concrete resource

<Model> - A base image for the model

Example:

|  |  |  |
| --- | --- | --- |
| **Device** | **Path** | **Version** |
| Cisco IOS Switch | /project/arts/SMOKE/stableimage/ cat3k\_caa-universalk9.SPA.03.06.06.E.152-2.E6.bin | 3.6.6E |
| GW1\_ Cisco\_IOS\_Switch | /project/arts/SMOKE/stableimage/ cat3k\_caa-universalk9.SPA.03.07.05.E.152-3.E5.bin | 3.7.5E |
| Cat3850-1\_Cisco\_IOS\_Switch | /project/arts/SMOKE/stableimage/ cat3k\_caa-universalk9.SPA.03.07.05.E.152-3.E5.bin | 3.7.5E |