SEBA Installation Guide

This section provides instructions to quickly bring up SEBA.

Contents

> *Note: This Installation Guide assumes that prerequisite hardware is met and software (Akraino Stack with CORD Platform) have already been installed.*
>
> *Specifically, wait for the three EtcdCluster CustomResourceDefinitions to appear in Kubernetes:*
>
> ***`kubectl get crd | grep etcd | wc -l`***
>
> *Once the CRDs are present, proceed with the `seba` chart installation.*

Overview

This page walks through the sequence of Helm operations needed to bring up the SEBA profile.

Prerequisites

It assumes the Akraino Stack with CORD Platform has already been installed.

Installation

Install components as a whole

Add the CORD repository and update indexes

```
$ helm repo add cord https://charts.opencord.org
$ helm repo update
```

`Install the CORD platform`

```
$ helm install -n cord-platform --version 6.1.0 cord/cord-platform
```

`Wait until 3 etcd CRDs are present in Kubernetes`

```
$ kubectl get crd | grep -i etcd | wc -l
```

Install the SEBA profile

```
$ helm install -n seba --version 1.0.0 cord/seba
```

Install the AT&T workflow

```
$ helm install -n att-workflow --version 1.0.2 cord/att-workflow
```

Alternatively, install as separate components

Add the official Kubernetes incubator repository (for Kafka) and update the indexes

```
$ helm repo add incubator http://storage.googleapis.com/kubernetes-charts-incubator
$ helm repo update
```

Add the CORD repository and update the indexes

```
$ helm repo add cord https://charts.opencord.org
$ helm repo update
```

Install the CORD platform components

```
$ helm install -n onos cord/onos
$ helm install -n xos-core cord/xos-core
$ helm install --version 0.13.3 \
             --set configurationOverrides."offsets.topic.replication.factor"=1 \
             --set configurationOverrides."log.retention.hours"=4 \
             --set configurationOverrides."log.message.timestamp.type"="LogAppendTime" \
             --set replicas=1 \
             --set persistence.enabled=false \
             --set zookeeper.replicaCount=1 \
             --set zookeeper.persistence.enabled=false \
             -n cord-kafka incubator/kafka
```

Optionally, install the logging and monitoring infrastructure components

```
$ helm install -n nem-monitoring cord/nem-monitoring
$ helm install --set elasticsearch.cluster.env.MINIMUM_MASTER_NODES="1" \
             --set elasticsearch.client.replicas=1 \
             --set elasticsearch.master.replicas=2 \
             --set elasticsearch.master.persistence.enabled=false \
             --set elasticsearch.data.replicas=1 \
             --set elasticsearch.data.persistence.enabled=false \
             -n logging cord/logging
```

Install etcd-operator and wait until 3 etcd CRDs are present in Kubernetes

```
$ helm install -n etcd-operator stable/etcd-operator --version 0.8.3
$ kubectl get crd | grep -i etcd | wc -l
```

Install the rest of the SEBA profile components

```
$ helm install -n voltha cord/voltha
$ helm install -n seba-service cord/seba-services
$ helm install -n base-kubernetes cord/base-kubernetes
```

Install the AT&T workflow

```
$ helm install -n att-workflow --version 1.0.2 cord/att-workflow
```

Verify your installation and next steps

Once the installation completes, monitor your setup using `kubectl get pods`. Wait until all pods are in *Running* state and “tosca-loader” pods are in *Co mpleted* state.

> *Note: The tosca-loader pods may periodically transition into* error *state. This is expected. They will retry and eventually get to the desired state. Note: Depending on the profile you're installing, you may need to check also different namespaces (for example, check the voltha namespace if you're installing SEBA with `kubectl get pods -n voltha`)*

Your POD is now installed and ready for use.

POD Configuration

Once all the components needed for the SEBA profile are up and running on your POD, you will need to configure it. This is typically done using TOSCA.

In this page we are describing the process as a three steps process:

- Fabric Setup
- OLT Provisioning
- Subscriber Provisioning

as that is what logically makes sense, but be aware that all the configurations can be unified in a single TOSCA file.

This configuration is environment specific, so you will need to create your own, but the following can serve as a reference:

Fabric Setup

```
tosca_definitions_version: tosca_simple_yaml_1_0
imports:
 - custom_types/switch.yaml
 - custom_types/switchport.yaml
 - custom_types/portinterface.yaml
 - custom_types/bngportmapping.yaml
 - custom_types/attworkflowdriverwhitelistentry.yaml
 - custom_types/attworkflowdriverservice.yaml
 - custom_types/serviceinstanceattribute.yaml
 - custom_types/onosapp.yaml

description: Configures a full SEBA POD

topology_template:
 node_templates:
    # Fabric configuration
    switch#leaf_1:
 type: tosca.nodes.Switch
 properties:
 driver: ofdpa3
 ipv4Loopback: 192.168.0.201
 ipv4NodeSid: 17
 isEdgeRouter: True
 name: AGG_SWITCH
 ofId: of:0000000000000001
 routerMac: 00:00:02:01:06:01
```

```
   # Setup the OLT switch port
   port#olt_port:
 type: tosca.nodes.SwitchPort
 properties:
 portId: 1
 host_learning: false
 requirements:
 - switch:
 node: switch#leaf_1
 relationship: tosca.relationships.BelongsToOne

   # Port connected to the BNG
   port#bng_port:
 type: tosca.nodes.SwitchPort
 properties:
 portId: 31
 requirements:
 - switch:
 node: switch#leaf_1
 relationship: tosca.relationships.BelongsToOne

   # Setup the fabric switch port where the external
   # router is connected to
 bngmapping:
 type: tosca.nodes.BNGPortMapping
 properties:
 s_tag: any
 switch_port: 31

   # DHCP L2 Relay config
   onos_app#dhcpl2relay:
 type: tosca.nodes.ONOSApp
 properties:
 name: dhcpl2relay
 must-exist: true

 dhcpl2relay-config-attr:
 type: tosca.nodes.ServiceInstanceAttribute
 properties:
 name: /onos/v1/network/configuration/apps/org.opencord.dhcpl2relay
 value: > { "dhcpl2relay" : { "useOltUplinkForServerPktInOut" : false, "dhcpServerConnectPoints" : [ "of:
0000000000000001/31" ] } }  requirements:
 - service_instance:
 node: onos_app#dhcpl2relay
 relationship: tosca.relationships.BelongsToOne
```

OLT Provisioning

```
tosca_definitions_version: tosca_simple_yaml_1_0
imports:
 - custom_types/oltdevice.yaml
 - custom_types/onudevice.yaml
 - custom_types/voltservice.yaml
description: Create an OLT Device in VOLTHA
topology_template:
 node_templates:

    service#volt:
 type: tosca.nodes.VOLTService
 properties:
 name: volt
 must-exist: true

 olt_device:
 type: tosca.nodes.OLTDevice
 properties:
 name: My OLT
 device_type: openolt
 host: 10.90.0.122
 port: 9191
 switch_datapath_id: of:0000000000000002 # the openflow switch to which the OLT is connected
 switch_port: "1" # the port on the switch on which the OLT is connected
 outer_tpid: "0x8100"
 uplink: "65536"
 nas_id: "NAS_ID"
 serial_number: "10.90.0.122:9191"
 requirements:
 - volt_service:
 node: service#volt
 relationship: tosca.relationships.BelongsToOne
```

Subscriber Provisioning

Once the POD has been configured, you can create a subscriber.

To create a subscriber, you'll need to know the serial number of the ONU it is attached to.

Find ONU Serial Number

Once your POD is set up and the OLT has been pushed and activated in VOLTHA, XOS will discover the ONUs available in the system.

You can find them through:

- XOS GUI: on the left side click on `vOLT > ONUDevices`
- XOS Rest API: `http://<pod-id>:<chameleon-port|30006>/xosapi/v1/volt/onudevices`
- VOLTHA CLI: Command Line Interface

If you are connected to the VOLTHA CLI you can use the following command to list all the existing devices:

```
(voltha) devices
Devices:
+------------------+--------------+------+------------------+-------------+-------------+----------------
+----------------+------------------+----------+------------------------+----------------------
+------------------------------+
|               id |         type | root |        parent_id | admin_state | oper_status | connect_status |
parent_port_no |    host_and_port | vendor_id| proxy_address.device_id | proxy_address.onu_id | proxy_address.
onu_session_id |
+------------------+--------------+------+------------------+-------------+-------------+----------------
+----------------+------------------+----------+------------------------+----------------------
+------------------------------+
| 0001941bd45e71d8 |      openolt | True | 000100000a5a0072 |     ENABLED |      ACTIVE |      REACHABLE
|                | 10.90.0.114:9191 |          |                        |
|                                |
| 00015698e67dc060 | broadcom_onu | True | 0001941bd45e71d8 |     ENABLED |      ACTIVE |      REACHABLE |
536870912 |                  |      BRCM|        0001941bd45e71d8 |                   1
|                                1 |
+------------------+--------------+------+------------------+-------------+-------------+----------------
+----------------+------------------+----------+------------------------+----------------------
+------------------------------+
```

Locate the correct ONU, then:

```
(voltha) device 00015698e67dc060
(device 00015698e67dc060) show
Device 00015698e67dc060
+------------------------------+------------------+
|                        field |            value |
+------------------------------+------------------+
|                           id | 00015698e67dc060 |
|                         type |     broadcom_onu |
|                         root |             True |
|                    parent_id | 0001941bd45e71d8 |
|                       vendor |         Broadcom |
|                        model |              n/a |
|             hardware_version |     to be filled |
|             firmware_version |     to be filled |
|                 images.image |        1 item(s) |
|                serial_number |     BRCM22222222 |
+------------------------------+------------------+
|                      adapter |     broadcom_onu |
|                  admin_state |                3 |
|                  oper_status |                4 |
|               connect_status |                2 |
|      proxy_address.device_id | 0001941bd45e71d8 |
|         proxy_address.onu_id |                1 |
| proxy_address.onu_session_id |                1 |
|               parent_port_no |        536870912 |
|                    vendor_id |             BRCM |
|                        ports |        2 item(s) |
+------------------------------+------------------+
|                   flows.items |        5 item(s) |
+------------------------------+------------------+
```

`to find the correct serial number.`

Push a Subscriber into CORD

Once you have this information, you can create the subscriber by customizing the following TOSCA and passing it into the POD:

```
tosca_definitions_version: tosca_simple_yaml_1_0
imports:
 - custom_types/rcordsubscriber.yaml
description: Create a test subscriber
topology_template:
 node_templates:
    # A subscriber
 my_house:
 type: tosca.nodes.RCORDSubscriber
 properties:
 name: My House
 c_tag: 111
 s_tag: 222
 onu_device: BRCM1234 # Serial Number of the ONU Device to which this subscriber is connected
```

Using TOSCA to push to CORD

Once CORD is up and running, a `node` can be added to a POD using the TOSCA interface by uploading the following recipe:

```
tosca_definitions_version: tosca_simple_yaml_1_0

description: Load a compute node in XOS

imports:
 - custom_types/node.yaml

topology_template:
 node_templates:

    # A compute node
 GratefulVest:
 type: tosca.nodes.Node
 properties:
 name: Grateful Vest
```

In TOSCA terminology, the above would be called a `TOSCA node template`.

Where to find the generated specs?

On any running CORD POD, the TOSCA apis are accessible as:

```
$ curl http://<head-node-ip>:<head-node-port>/xos-tosca | python -m json.tool
```

And it will return a list of all the recipes with the related url:

```
{
  "image": "/custom_type/image",
  "site": "/custom_type/site",
  ...
}
```

For examples, to site the TOSCA spec of the Site model, you can use the URL:

```
$ curl http://<head-node-ip>:<head-node-port>/xos-tosca/custom_type/site
```

If you have a running `xos-tosca` container you can also find generated copies of the specs in **`/opt/xos-tosca/src/tosca/custom_types`.**

How to load a TOSCA recipe in the system

The `xos-tosca` container exposes two endpoint:

```
POST http://<cluster-ip>:<tosca-port>/run
POST http://<cluster-ip>:<tosca-port>/delete
```

To load a recipe via `curl` you can use this command:

```
$ curl -H "xos-username: xosadmin@opencord.org" -H "xos-password: <xos-password>" -X POST --data-binary @<path
/to/file> http://<cluster-ip>:<tosca-port>/run
```

If you installed the `xos-core` charts without modifications, the `tosca-port` is **`30007.`**

References

- SEBA installation guide: https://guide.opencord.org/profiles/seba/install.html
- If you have question on the above link, please join this group to ask questions there.
  https://groups.google.com/a/opennetworking.org/forum/#!forum/seba-dev