Deploy OOM and SDC (or ONAP) on a single VM with microk8s - Honolulu Setup

Introduction

This tutorial is going to explain how to setup a local Kubernetes cluster and minimal helm setup to run and deploy SDC (but can be extended to several/all ONAP components) on a single host.

The rationale for this tutorial is to spin up a development environment quickly and efficiently without the hassle of setting up a multi node cluster/Network file share that are needed in a full deployment.

This was a trial in G release but is now extended to the requirements of H release.

WARNING :

There are limitations in using this setup, the purpose is to allow for developers to quickly get access to a working environment '*the way their component is supposed to be deployed*'

This is not meant for production obviously, and the tweaks that are done to the OOM/K8s setup are most likely going to evolve with further releases.

Minimum Requirements

- One VM running Ubuntu 20.04 LTS (should also work on 18.04), with internet access to download charts/containers and the oom repo
- Root/sudo privileges
- Sufficient RAM depending on how many components you want to deploy
    - Around 20G of RAM allows for a few components, the minimal setup for SDC is enabling :
        - Shared Cassandra
        - AAF
        - Portal (if you need UI access)
        - SDC
    - This was tested with a huge VM - 128G of RAM and 12 VCPU, running most of the components in Honolulu development.
    - This was tested with a small VM to run components on a local Laptop (need enough ram to create a 20G Ram VM using VBox, VMWare...)
- above 160G available storage should be sufficient, mostly depends on how many components you want to enable in OOM charts.
    - Storage is required mostly to store container images

Overall Procedure

- Install/remove Microk8s with appropriate version
- Install/remove Helm with appropriate version
- Tweak Microk8s
- Download oom repo
- Install the needed Helm plugins
- Install ChartMuseum as a local helm repo
- Install docker (now needed to build oom charts)
- Build all oom charts and store them in the chart repo
- Tweak oom override file to fine tune deployment based on your VM capacity and component needs
- Deploy/Undeploy charts
- Enable local UI access
- Full Deployment Override on a single VM
- Run testsuites

1) Install/Upgrade Microk8s with appropriate version

Why Microk8s ?

Microk8s is a bundled lightweight version of kubernetes maintained by Canonical, it has the advantage to be well integrated with snap on Ubuntu, which makes it super easy to manage/upgrade/work with

More info on : https://microk8s.io/docs

There are 2 things to know with microk8s :

1) it is wrapped by snap, which is nice but you need to understand that it's not exactly the same as having a proper k8s installation (more info below on some specific commands)

2) it is not using docker as the container runtime, it's using containerd, it's not an issue, just be aware of that as you won't see containers using classic docker commands

How to remove/install microk8s ?

If you have a previous version of microk8s, you first need to uninstall it (upgrade is possible but it is not recommended between major versions so I recommend to uninstall as it's fast and safe)

```
sudo snap remove microk8s
```

You need to select the appropriate version to install, to see all possible version do :

```
sudo snap info microk8s
```

this tutorial is focused on Honolulu release so we will use k8s version 1.19, to do so, you just need to select the appropriate channel

```
sudo snap install microk8s --classic --channel=1.18/stable
sudo snap refresh microk8s --classic --channel=1.19/stable

Or (when the master node bug will be fixed on 1.19 kubernetes snap install)

sudo snap install microk8s --classic --channel=1.19/stable
```

You may need to change your firewall configuration to allow pod to pod communication and pod to internet communication :

```
sudo ufw allow in on cni0 && sudo ufw allow out on cni0
sudo ufw default allow routed
```

Addons ?

MicroK8s is lightweight but comes with several possible addons, OOM and ONAP requires just a few to be enabled, you can choose to enable more if you want to

DNS addon : we need the DNS addon so that pods can 'see' each other by host name.

Storage addon : we will enable the default Host storage class, this allows local volume storage that are used by some pods to exchange folders between containers.

```
sudo microk8s enable dns storage
```

that's it, you should have a running k8s cluster, ready to host ONAP pods

I recommend to get familiar with microk8s, here are a few useful commands but you can read more on the microk8s website :

- microk8s status: Provides an overview of the MicroK8s state (running / not running) as well as the set of enabled addons
- microk8s enable: Enables an addon
- microk8s disable: Disables an addon
- microk8s kubectl: Interact with kubernetes
- microk8s config: Shows the kubernetes config file
- microk8s inspect: Performs a quick inspection of the MicroK8s intallation
- microk8s reset: Resets the infrastructure to a clean state very useful for a dev lab

- microk8s stop: Stops all kubernetes services
- microk8s start: Starts MicroK8s after it is being stopped

2) Install/remove Helm with appropriate version

Helm is the package manager for k8s, we require a specific version for each ONAP release, the best is to look at the OOM guides to see which one is required (link to add)

For the Honolulu release we need Helm 3 - A significant improvement with Helm3 is that it does not require a specific pod running in the kubernetes cluster (no more Tiller pod)

As Helm is self contained, it's pretty straightforward to install/upgrade, we can also use snap to install the right version

```
sudo snap install helm --classic --channel=3.5/stable
```

Note: You may encounter some log issues when installing helm with snap

Normally the helm logs are available in "~/.local/share/helm/plugins/deploy/cache/onap/logs", if you notice that the log files are all equal to 0, you can uninstall the helm with snap and reinstall it manually

```
wget https://get.helm.sh/helm-v3.5.4-linux-amd64.tar.gz
```

```
tar xvfz helm-v3.5.4-linux-amd64.tar.gz
```

```
sudo mv linux-amd64/helm /usr/local/bin/helm
```

3) Tweak Microk8s

The below tweaks are not strictly necessary, but they help in making the setup more simple and flexible.

A) Increase the max number of pods & Add priviledged config

As ONAP may deploy a significant amount of pods, we need to inform kubelet to allow more than the basic configuration (as we plan an all in box setup), if you only plan to run a limited number of components, this is not needed

to change the max number of pods, we need to add a parameter to the startup line of kubelet.

1. **Edit the file located at :**

```
sudo nano /var/snap/microk8s/current/args/kubelet
```

add the following line at the end :

```
--max-pods=250
```

save the file and restart kubelet to apply the change :

```
sudo service snap.microk8s.daemon-kubelet restart
```

2. **Edit the file located at :**

```
sudo nano /var/snap/microk8s/current/args/kube-apiserver
```

add the following line at the end :

```
--allow-privileged=true
```

save the file and restart kubelet to apply the change :

```
sudo service snap.microk8s.daemon-apiserver restart
```

B) run a local copy of kubectl

Microk8s comes bundled with kubectl, you can interact with it by doing:

```
sudo microk8s kubectl describe node
```

to make things simpler as we will most likely interact a lot with kubectl, let's install a local copy of kubectl so we can use it to interact with the kubernetes cluster in a more straightforward way

We need kubectl 1.19 to match the cluster we have installed, let's again use snap to quickly choose and install the one we need

```
sudo snap install kubectl --classic --channel=1.19/stable
```

Now we need to provide our local kubectl client with a proper config file so that it can access the cluster, microk8s allows to retrieve the cluster config very easily

Simply create a .kube folder in your home directory and dump the config there

```
cd
mkdir .kube
cd .kube
sudo microk8s.config > config
chmod 700 config
```

the last line is there to avoid helm complaining about too open permission

you should now have helm and kubectl ready to interact with each other, you can verify this by trying :

```
kubectl version
```

this should output both the local client and server version

```
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.7", GitCommit:"
1dd5338295409edcfff11505e7bb246f0d325d15", GitTreeState:"clean", BuildDate:"2021-01-13T13:23:52Z", GoVersion:"go1.
15.5", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"19+", GitVersion:"v1.19.7-34+02d22c9f4fb254", GitCommit:"
02d22c9f4fb2545422b2b28e2152b1788fc27c2f", GitTreeState:"clean", BuildDate:"2021-02-11T20:13:16Z", GoVersion:"go1.
15.8", Compiler:"gc", Platform:"linux/amd64"}
```

4) Download OOM repo

OOM provides a couple of plugins that we need to install for helm, the easiest is to get them straight from the repository.

Since we will also build the charts locally, it's the right time to download a local copy, you'll need to install git if it's not available yet on your machine

the example below pulls the latest version from master, it's probably wiser to select the right version (honolulu branch or a specific review you want to test)

```
cd
git clone "https://gerrit.onap.org/r/oom"
```

5) Install the needed Helm plugins

Onap deployments are using the deploy and undeploy plugins for helm to install them just run :

```
helm plugin install ./oom/kubernetes/helm/plugins/undeploy/
helm plugin install ./oom/kubernetes/helm/plugins/deploy/
```

this will copy the plugins into your home directory .helm folder and make them available as helm commands

Another plugin we need is the push plugin, with helm3 there is no more an embedded repo to use.

```
helm plugin install --version v0.9.0 https://github.com/chartmuseum/helm-push.git
```

Once all plugins are installed, you should see them as available helm commands when doing :

```
helm --help
```

6) Install the chartmuseum repository

To align with how the previous release were deployed, we will setup a local chart repository.

To do, download the chartmuseum script and run it as a background task

```
curl https://raw.githubusercontent.com/helm/chartmuseum/main/scripts/get-chartmuseum | bash
mkdir ~/chartmuseum
chartmuseum --port=8080   --storage="local"   --storage-local-rootdir="~/chartstorage" &
```

you should see the chartmuseum repo starting locally, you can press enter to come back to your terminal

you can now inform helm that a local repo is available for use :

```
helm repo add local http://localhost:8080
```

Note this is a very basic setup, meaning you'll need to startup your local repo if you shut down your VM or if you exit this terminal, there are other ways to do it, like using a docker compose to ensure 'it's always on'

Also, you can setup a local database for Helm to store it's deployment (more info on Helm website), this is useful if you want to upgrade and undeploy charts, now on this setup which is aimed for developpers, the approach is a clean install and wipe if we need to change something so we won't describe this part here, see last point on how to deploy/undeploy with this setup

7) Install docker

During the processing of the oom helm charts (oom-cert-service component of platform), docker is needed as it is used to create certificates.

Follow the docker install instructions for Ubuntu here : https://docs.docker.com/engine/install/ubuntu/

unless you already have docker, in which case you can skip this part altogether.

Or use snap:

```
sudo snap install docker
```

8) Build all oom charts and store them in the chart repo

You should be ready to build all helm charts, go into the oom/kubernetes folder and run a full make

Ensure you have "make" installed:

```
sudo apt install make
```

Then build OOM

```
cd ~/oom/kubernetes
make all
```

You can speed up the make skipping the linting of the charts

```
make all -e SKIP_LINT=TRUE
```

You'll notice quite a few message popping into your terminal running the chartmuseum, showing that it accepts and store the generated charts, that's normal, if you want, just open another terminal to run the helm commands

Once the build completes, you should be ready to deploy ONAP

9) Tweak oom override file to fine tune deployment based on your VM capacity and component needs

Before deploying, it's good to decide which ONAP components you need and create an override file. The override file will tune your deployment to your specific needs.

We have a few considerations to take on our stand alone VM :

1) CPU/Memory might be limited, so be careful on what you enable / disable. K8S 1.19 now properly enforce CPU and memory requests/limits so it is important to define/override them correctly

2) Timeouts : the more component you enable, the more the liveness probes and readiness probes timeouts may have to be increased, knowing that we will most likely over commit the VM CPU.

First, here is a very basic override file to enable a very small setup with SDC, we will only enable aaf, portal, SDC and cassandra shared, this should work on a laptop with enough ram :

copy the content below to a file called **onap-honolulu.yaml**

```
#################################################################
# Global configuration overrides.
#
# These overrides will affect all helm charts (ie. applications)
# that are listed below and are 'enabled'.
#################################################################
global:
  # Change to an unused port prefix range to prevent port conflicts
  # with other instances running within the same k8s cluster
  nodePortPrefix: 302
  nodePortPrefixExt: 304
  masterPassword: secretpassword
  # ONAP Repository
  # Uncomment the following to enable the use of a single docker
  # repository but ONLY if your repository mirrors all ONAP
  # docker images. This includes all images from dockerhub and
  # any other repository that hosts images for ONAP components.
  #repository: nexus3.onap.org:10001

  # readiness check - temporary repo until images migrated to nexus3
  readinessRepository: oomk8s
  # logging agent - temporary repo until images migrated to nexus3
  loggingRepository: docker.elastic.co

  # image pull policy
  pullPolicy: IfNotPresent

  # override default mount path root directory
  # referenced by persistent volumes and log files
  persistence:
    mountPath: /dockerdata-nfs

  # flag to enable debugging - application support required
  debugEnabled: false

#################################################################
# Enable/disable and configure helm charts (ie. applications)
# to customize the ONAP deployment.
#################################################################
aaf:
  enabled: true
  aaf-service:
    readiness:
      initialDelaySeconds: 150
cassandra:
  enabled: true
  replicaCount: 3
  config:
    cluster_domain: cluster.local
    heap:
      max: 1G
      min: 256M
  liveness:
    initialDelaySeconds: 60
    periodSeconds: 20
    timeoutSeconds: 10
    successThreshold: 1
    failureThreshold: 3
    # necessary to disable liveness probe when setting breakpoints
    # in debugger so K8s doesn't restart unresponsive container
    enabled: true

  readiness:
    initialDelaySeconds: 120
    periodSeconds: 20
    timeoutSeconds: 10
    successThreshold: 1
    failureThreshold: 3
portal:
  enabled: true
sdc:
  enabled: true
  config:
    environment:
      vnfRepoPort: 8703
  sdc-be:
    config:
      javaOptions: "-Xmx1g -Xms512m"
    liveness:
      periodSeconds: 300
      timeoutSeconds: 180
    readiness:
      periodSeconds: 300
      timeoutSeconds: 240
  sdc-fe:
    resources:
      small:
        limits:
          cpu: 1
          memory: 2Gi
        requests:
          cpu: 100m
          memory: 500Mi
```

You'll notice the tweaks to the timeouts and memory settings and limits, this is mostly what you have to adapt depending on your needs

Once the file is ready you can deploy :

1) First you need to ensure that the onap namespace exists (it now must be created prior deployment)

```
kubectl create namespace onap
```

2) Launch the chart deployment, pay attention to the last parameter, it must point to your override file create above

```
helm deploy onap local/onap --namespace onap -f ~/onap-honolulu.yaml
```

The deployment can take a while depending on how many charts you've enabled.

You should see all pods starting up and you should be able to see logs using kubectl, dive into containers etc...

How to undeploy and start fresh

The easiest is to use kubectl, you can clean up the cluster in 3 commands :

```
kubectl delete namespace onap
kubectl delete pv --all
sudo rm -rf /dockerdata-nfs
```

don't forget to create the namespace again before deploying again (helm won't complain if it is not there, but you'll end up with an empty cluster after if finishes)

Note : you could also reset the K8S cluster by using the microk8s feature : microk8s reset

but that's a more drastic approach, you would need to reinstall the addons, dump again the config for kubectl etc.... this can be useful if you get weird behavior though.

10) Enable UI access

Once all pods are running and all jobs are completed,

You may want to connect to the ONAP UIs locally, to do so you first need to add the following to your /etc/host (adapt the IP address if you use a remote VM)

```
# for myVM ONAP
127.0.0.1 aaf-gui
127.0.0.1 aai.ui.simpledemo.onap.org
127.0.0.1 appc.api.simpledemo.onap.org
127.0.0.1 cds.api.simpledemo.onap.org
127.0.0.1 cdt.api.simpledemo.onap.org
127.0.0.1 clamp.api.simpledemo.onap.org
127.0.0.1 nbi.api.simpledemo.onap.org
127.0.0.1 policy.api.simpledemo.onap.org
127.0.0.1 portal.api.simpledemo.onap.org
127.0.0.1 robot-onap.onap.org
127.0.0.1 sdc.api.fe.simpledemo.onap.org
127.0.0.1 sdc.api.simpledemo.onap.org
127.0.0.1 sdc.workflow.plugin.simpledemo.onap.org
127.0.0.1 so-monitoring
127.0.0.1 vid.api.simpledemo.onap.org
```

You can then access the portal UI by opening your browser to :
https://portal.api.simpledemo.onap.org:30225/ONAPPORTAL/login.htm
user/pass is cs0008/demo123456!

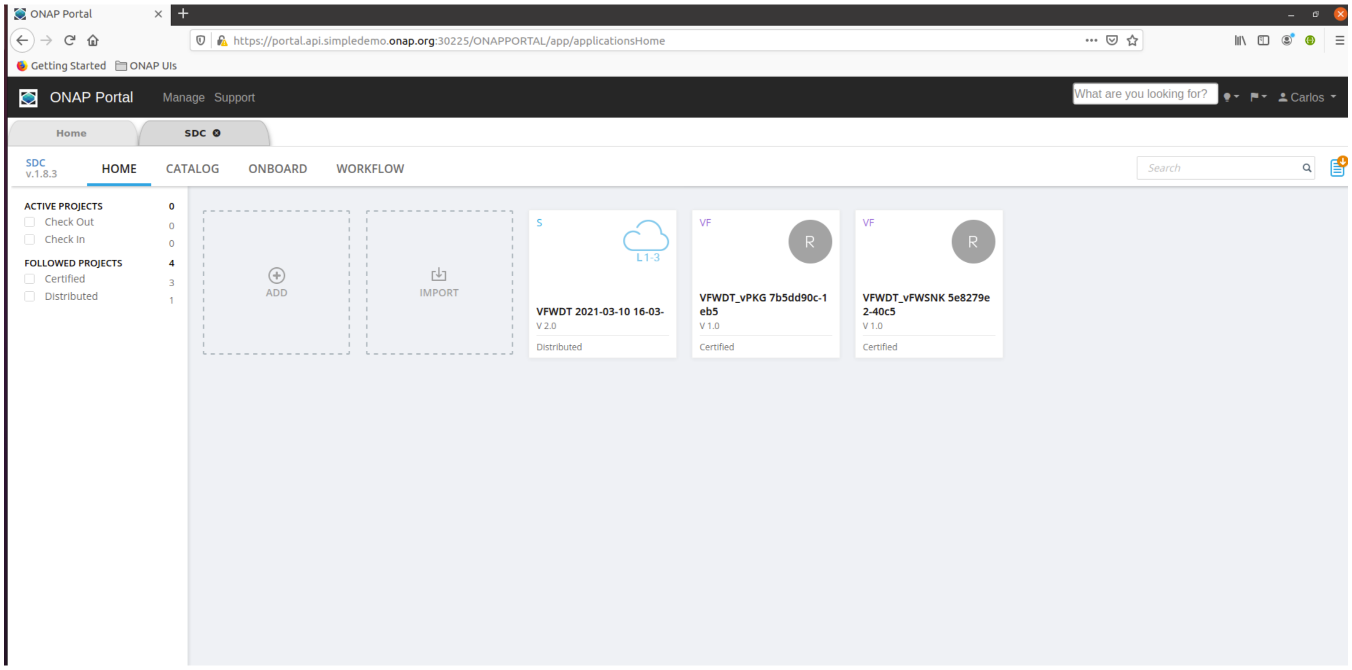
Note : Since SDC (and other UIs) are embedded into the portal UI frame, you browser may complain about trusting its certificate, to work around this issue, open the SDC UI directly first (to get your browser to accept the certificate), once done you should be able to open the SDC UI through the portal. To do so, open another tab of your browser to :

https://sdc.api.fe.simpledemo.onap.org:30207/

https://sdc.workflow.plugin.simpledemo.onap.org:30256/

Note : you need to repeat the above for each deployment, as certificates are generated by AAF during startup, certificates will change each time you deploy

You should now be able to use SDC :



11) Full Deployment Override on a single VM

Below is a full blown override file used to run mostly all Honolulu components, this was tested working on Windriver Lab on a VM with 12VCpu and 128G Ram

```
# Copyright © 2017 Amdocs, Bell Canada
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

#################################################################
# Global configuration overrides.
#
# These overrides will affect all helm charts (ie. applications)
# that are listed below and are 'enabled'.
#################################################################
global:
  # Change to an unused port prefix range to prevent port conflicts
  # with other instances running within the same k8s cluster
  nodePortPrefix: 302
  nodePortPrefixExt: 304
  masterPassword: secretpassword
  addTestingComponents: true
  cmpv2Enabled: true
  flavor: unlimited
  # ONAP Repository
  # Uncomment the following to enable the use of a single docker
  # repository but ONLY if your repository mirrors all ONAP
  # docker images. This includes all images from dockerhub and
  # any other repository that hosts images for ONAP components.
  #repository: nexus3.onap.org:10001

  # readiness check - temporary repo until images migrated to nexus3
  readinessRepository: oomk8s
  # logging agent - temporary repo until images migrated to nexus3
  loggingRepository: docker.elastic.co

  # image pull policy
  pullPolicy: IfNotPresent

  # override default mount path root directory
  # referenced by persistent volumes and log files
  persistence:
    mountPath: /dockerdata-nfs

  # flag to enable debugging - application support required
  debugEnabled: false

#################################################################
# Enable/disable and configure helm charts (ie. applications)
# to customize the ONAP deployment.
#################################################################
aaf:
  enabled: true
  aaf-service:
    readiness:
      initialDelaySeconds: 150
aai:
  enabled: true
  flavorOverride: unlimited
  global:
    flavorOverride: unlimited
    cassandra:
      replicas: 3
  aai-cassandra:
    flavorOverride: unlimited
    replicaCount: 3
  aai-babel:
    flavorOverride: unlimited
  aai-data-router:
    flavorOverride: unlimited
  aai-elasticsearch:
    flavorOverride: unlimited
  aai-graphadmin:
    flavorOverride: unlimited
  aai-modelloader:
    flavorOverride: unlimited
  aai-resources:
    flavorOverride: unlimited
  aai-schema-service:
    flavorOverride: unlimited
  aai-search-data:
    flavorOverride: unlimited
  aai-sparky-be:
    flavorOverride: unlimited
    readiness:
      initialDelaySeconds: 150
      periodSeconds: 20
      timeoutSeconds: 10
    liveness:
      timeoutSeconds: 10
  aai-traversal:
    flavorOverride: unlimited
appc:
  enabled: false
cassandra:
  enabled: true
  replicaCount: 3
  config:
    cluster_domain: cluster.local
    heap:
      max: 1G
      min: 256M
  liveness:
    initialDelaySeconds: 60
    periodSeconds: 20
    timeoutSeconds: 10
    successThreshold: 1
    failureThreshold: 3
    # necessary to disable liveness probe when setting breakpoints
    # in debugger so K8s doesn't restart unresponsive container
    enabled: true

  readiness:
    initialDelaySeconds: 120
    periodSeconds: 20
    timeoutSeconds: 10
    successThreshold: 1
    failureThreshold: 3
cds:
  enabled: true
clamp:
  enabled: true
cli:
  enabled: true
consul:
  enabled: true
contrib:
  enabled: true
  awx:
    enabled: true
  netbox:
    enabled: false
dcaegen2:
  enabled: true
  dcae-cloudify-manager:
    readiness:
      periodSeconds: 120
      timeoutSeconds: 20
    liveness:
      timeoutSeconds: 10
dmaap:
  enabled: true
esr:
  enabled: false
log:
  enabled: false
  log-logstash:
    replicaCount: 1
sniro-emulator:
  enabled: false
oof:
  enabled: false
mariadb-galera:
  enabled: true
msb:
  enabled: true
multicloud:
  enabled: true
nbi:
  enabled: true
platform:
  enabled: true
policy:
  enabled: true
pomba:
  enabled: false
portal:
  enabled: true
robot:
  enabled: true
  flavor: small
  appcUsername: "appc@appc.onap.org"
  appcPassword: "TBD"
  openStackKeyStoneUrl: "http://10.12.25.2:5000"
  openStackKeystoneAPIVersion: "v3"
  openStackPublicNetId: "TBD"
  openStackTenantId: "TBD"
  openStackUserName: "TBD"
  openStackUserDomain: "TBD"
  openStackProjectName: "TBD"
  ubuntu14Image: "trusty-server-cloudimg-amd64-disk1"
  ubuntu16Image: "xenial-server-cloudimg-amd64-disk1"
  openStackPrivateNetCidr: "10.0.0.0/16"
  openStackPrivateNetId: "TBD"
  openStackPrivateSubnetId: "TBD"
  openStackSecurityGroup: "TBD"
  openStackOamNetworkCidrPrefix: "10.0"
  dcaeCollectorIp: "yourhostip"
  vnfPubKey: "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDKXDgoo3+WOqcUG8
/5uUbk81+yczgwC4Y8ywTmuQqbNxlY1oQ0YxdMUqUnhitSXs5S
/yRuAVOYHwGg2mCs20oAINrP+mxBI544AMIb9itPjCtgqtE2EWo6MmnFGbHB4Sx3XioE7F4VPsh7japsIwzOjbrQe+Mua1TGQ5d4nfEOQaaglXLL
PFfuc7WbhbJbK6Q7rHqZfRcOwAMXgDoBqlyqKeiKwnumddo2RyNT8ljYmvB6buz7KnMinzo7qB0uktVT05FH9Rg0CTWH5norlG5qXgP2aukL0gk1
ph8iAt7uYLf1ktp+LJI2gaF6L0/qli9EmVCSLr1uJ38Q8CBflhkh"
  demoArtifactsVersion: "1.4.0"
  demoArtifactsRepoUrl: "https://nexus.onap.org/content/repositories/releases"
  scriptVersion: "1.4.0"
  nfsIpAddress: "yourhostip"
  config:
    openStackEncryptedPasswordHere: "TBD"
    openStackSoEncryptedPassword: "TBD"
sdc:
  enabled: true
  config:
    environment:
      vnfRepoPort: 8703
  sdc-be:
    config:
      javaOptions: "-Xmx1g -Xms512m"
    liveness:
      periodSeconds: 300
      timeoutSeconds: 180
    readiness:
      periodSeconds: 300
      timeoutSeconds: 240
  sdc-fe:
    resources:
      small:
        limits:
          cpu: 1
          memory: 2Gi
        requests:
          cpu: 100m
          memory: 500Mi
sdnc:
  enabled: true

  replicaCount: 1

  mysql:
    replicaCount: 1
so:
  enabled: true
  so-catalog-db-adapter:
    config:
      openStackUserName: "TBD"
      openStackKeyStoneUrl: "http://10.12.25.2:5000/v3"
      openStackEncryptedPasswordHere: "TBD"
      openStackKeystoneVersion: "KEYSTONE_V3"
uui:
  enabled: false
vfc:
  enabled: false
vid:
  enabled: true
vnfsdk:
  enabled: true
```

If you want to use SO / Robot - you need to fill in proper properties in the override, this depends on your openstack instance / tenant

12) Run testsuites

If you have deployed the robot pod, you can perform some tests using the provided scripts in the oom repo

go to the oom/kubernetes/robot folder and run ete-k8s.sh to see the available test suites

```
cd ~/oom/kubernetes/robot
./ete-k8s.sh
```

to run a specific test suite, just provide the namespace and the testsuite tag, below is testing distribution with SDC (beware that some tests require the availability of several components to succeed)

```
cd ~/oom/kubernetes/robot
./ete-k8s.sh onap healthdist
```

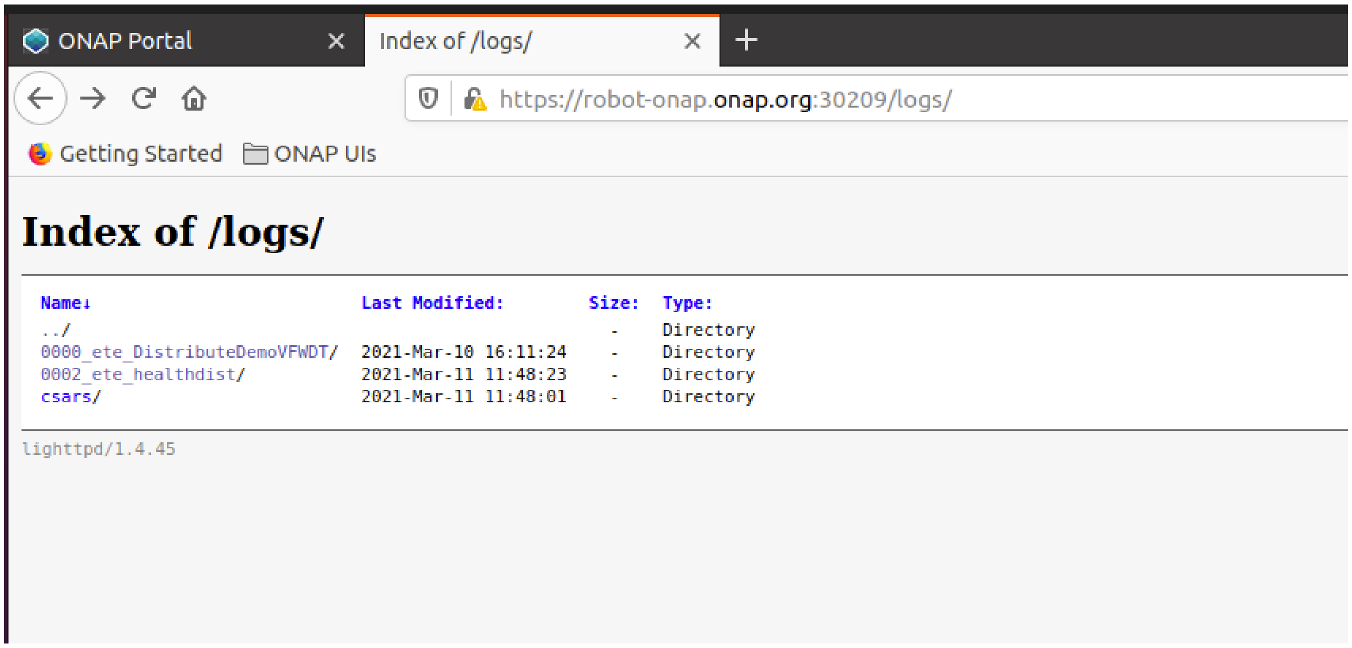
the script will contact the robot pod within the cluster and execute the tests (may take a while)

```
++ kubectl --namespace onap exec onap-robot-8569b648b9-5x49x -- /var/opt/ONAP/runTags.sh -V /share/config
/robot_properties.py -v GLOBAL_BUILD_NUMBER:10433 -d /share/logs/0002_ete_healthdist -i healthdist --display 92
Starting Xvfb on display :92 with res 1280x1024x24
Executing robot tests at log level TRACE
==============================================================================
Testsuites
==============================================================================
/var/opt/ONAP/robot/library/paramiko/transport.py:33: CryptographyDeprecationWarning: Python 2 is no longer
supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in a
future release.
  from cryptography.hazmat.backends import default_backend
Testsuites.Health-Check :: Test that ONAP components are available via basi...
==============================================================================
Health Distribution Test                                              | PASS |
------------------------------------------------------------------------------
Testsuites.Health-Check :: Test that ONAP components are available... | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Testsuites                                                            | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /share/logs/0002_ete_healthdist/output.xml
Log:     /share/logs/0002_ete_healthdist/log.html
Report:  /share/logs/0002_ete_healthdist/report.html
```

if you have enabled UI access you can see the test logs in a browser by connecting to

https://robot-onap.onap.org:30209

(user/pass is test/test)


ONAP Portal | Index of /logs/

https://robot-onap.onap.org:30209/logs/

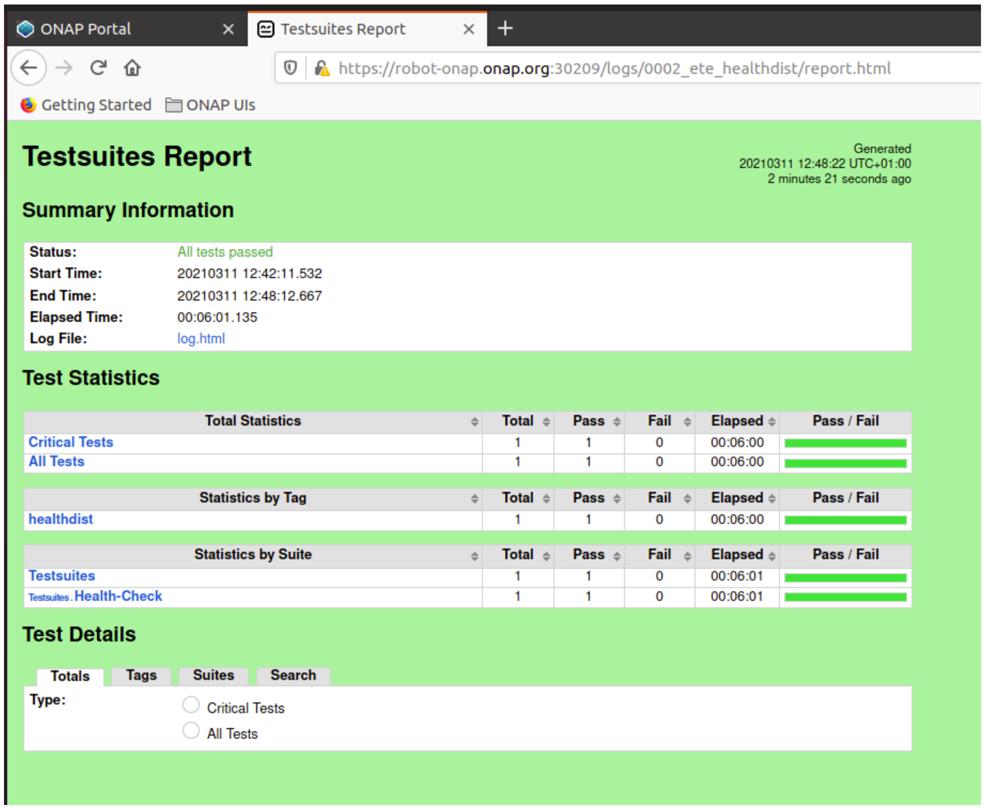
Getting Started | ONAP UIs

Index of /logs/

```
Name↓                          Last Modified:        Size:  Type:
../                                                  -      Directory
0000_ete_DistributeDemoVFWDT/  2021-Mar-10 16:11:24  -      Directory
0002_ete_healthdist/           2021-Mar-11 11:48:23  -      Directory
csars/                         2021-Mar-11 11:48:01  -      Directory
```

lighttpd/1.4.45

browse to the test suite you have started and open the folder, click the report.html to see robot test results


ONAP Portal | Testsuites Report

https://robot-onap.onap.org:30209/logs/0002_ete_healthdist/report.html

Getting Started | ONAP UIs

Testsuites Report

Generated
20210311 12:48:22 UTC+01:00
2 minutes 21 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20210311 12:42:11.532
End Time:	20210311 12:48:12.667
Elapsed Time:	00:06:01.135
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:06:00	🟩
All Tests	1	1	0	00:06:00	🟩

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
healthdist	1	1	0	00:06:00	🟩

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Testsuites	1	1	0	00:06:01	🟩
Testsuites.Health-Check	1	1	0	00:06:01	🟩

Test Details

Type: ○ Critical Tests ○ All Tests