CNaaS NMS Development Environment

In addition to installing the requirements for running the Python code for CNaaS-NMS you might also want a test environment so you can run integration tests for your newly developed code. For this you will need docker to run all the components of CNaaS, and a virtualization software to run some virtual switches. In this example we will use VirtualBox and Arista vEOS switches.

Create VMs

Download the VMDK for Arista vEOS (tested with vEOS-lab-4.22.3M.vmdk , possibly issues with early 4.23 releases?). Copy this file into three new VMDK files that we will use to create the actual VMs: eosdist1.vmdk , eosdist2.vmdk , and eosaccess.vmdk

Then start VirtualBox and create three new VMs, using 2 GB memory each and choose to point them to an existing harddrive file (you will need to "add" the VMDK files you copied earlier to the "Virtual media manager" in VirtualBox). After creating the new VMs, we need to configure the network adapters. Before we can do this go into the menu FileHost network manager... in VirtualBox and create a new network called vboxnet1 for example, and enter IP 10.100.2.2 / 255.255.255.0 (no DHCP server). On Linux/MAC you need to allow this IP-range in by creating `/etc/vbox/networks.conf` and specifying allowed ranges there. For example, to allow 10.0.0.0/8 and 192.168.0.0/16 IPv4 ranges as well as 2001::/64 range put the following lines into `/etc/vbox/networks.conf`:

```
* 10.0.0.0/8 192.168.0.0/16
* 2001::/64
```

Then configure the VMs with the following network adapters:

eosdist1:

Make sure all NICs are "Intel PRO/1000 T Server" or Desktop, but **not** "MT", and NIC 2-4 have Promiscous mode: Allow all

NIC1: NAT (Management1)

NIC2: Host-only adapter: cnaas (Ethernet1)

NIC3: Internal network: link_d1a1 (Ethernet2)

NIC4: Internal network: link_d1d2 (Ethernet3)

eosdist2:

Make sure all NICs are "Intel PRO/1000 T Server" or Desktop, but **not** "MT", and NIC 2-4 have Promiscous mode: Allow all

NIC1: NAT (Management1)

NIC2: Host-only adapter: cnaas (Ethernet1)

NIC3: Internal network: link_d2a1 (Ethernet2)

NIC4: Internal network: link_d1d2 (Ethernet3)

eosaccess:

Make sure all NICs are "Intel PRO/1000 T Server" or Desktop, but **not** "MT", and NIC 2-4 have Promiscous mode: Allow all

NIC1: NAT (Management1)

NIC2: Host-only adapter: cnaas (Ethernet1)

NIC3: Internal network: link_d1a1 (Ethernet2)

NIC4: Internal network: link_d2a1 (Ethernet3)

You will also need to add some static routing on your host so return traffic will find it's way through eosdist1 to the correct VM:

```
sudo ip route add 10.0.6.0/24 via 10.100.2.101 dev vboxnet1
sudo ip route add 192.168.0.0/24 via 10.100.2.101 dev vboxnet1
sudo ip route add 10.100.3.101/32 via 10.100.2.101 dev vboxnet1
sudo ip route add 10.100.3.102/32 via 10.100.2.102 dev vboxnet1
```

These commands will not persist through a reboot of the host, and they must be added after Virtualbox/vboxnet1 adapter is started.

Configure VMs

Start up eosdist1 and eosdist2. Login with admin/<enter> when the have booted up, and then enter the command "zerotouch cancel" when the have booted up. Enter a config like this using console/SSH on eosdist1:

```
hostname eosdist1
username admin privilege 15 role network-admin secret abc123abc123
vrf instance MGMT
interface Ethernet1
   no switchport
   vrf MGMT
   ip address 10.100.2.101/24
   no lldp transmit
   no lldp receive
interface Ethernet2
   description DOWNLINK
   switchport mode trunk
   channel-group 3 mode active
!
interface Port-Channel3
   description DOWNLINK
   switchport mode trunk
   port-channel lacp fallback individual
   port-channel lacp fallback timeout 3
!
interface Loopback1
   vrf MGMT
   ip address 10.100.3.101/32
!
interface Vlan1
   description ZTP DHCP
   vrf MGMT
   ip address 192.168.0.1/24
   ip helper-address 10.100.2.2
ip routing vrf MGMT
management api http-commands
   no shutdown
   !
   vrf MGMT
      no shutdown
```

The Adapter1/NIC1 in Virtualbox should correspond to the interface called Management1 inside the VM. Adapter2 should be Ethernet1 in the VM, Adapter3 should be Ethernet2 in the VM and so on.

Eosdist2 only needs a hostname of "eosdist2" for the current tests to run, but you could also configure it in a similar way to eosdist1.

To test that routing through eosdist1 works correctly you can run some ping commands from inside the eosdist1 VM:

```
ping vrf MGMT 10.100.2.2
```

```
ping vrf MGMT 10.100.2.2 source 192.168.0.1
```

If the first command doesn't work something with the interface configuration might be wrong. If the second command doesn't work, it might be "ip route add" commands in the previos section is missing.

Run integrationtests.sh

Git clone cnaas-nms and go to the directory test/ , there you will find a script called integrationtest.sh . This script will start the necessary docker containers and then begin running some tests for ZTP and so on. Before starting the docker containers we need to create a few volumes:

```
docker volume create cnaas-templates
docker volume create cnaas-settings
docker volume create cnaas-postgres-data
docker volume create cnaas-jwtcert
```

To get authentication working you need a JWT certificate. You can either download this dummy public.pem cert for example and place it inside the API container at /opt/cnaas/jwtcert/public.pem (or setup some external JWT server like SUNET auth poc).

You are now ready to start the integration tests. When running integrationtests.sh it will wait for a device to enter the DISCOVERED state for 10 minutes, so you can start by booting up eosaccess now, and then start integrationtests.sh. eosaccess should start ZTP boot via DHCP from the DHCPd container started by integrationtests.sh , and then reboot once again. The second time it starts up a job should be scheduled to discover the device, you can check the progress here by tailing logs from the dhcp and api containers like this: "docker logs -f docker_cnaas_dhcpd_1" (or docker_cnaas_api_1).

After a device in state DISCOVERED has been found ZTP will automatically start and then the rest of integration tests will run. After the integration tests has completed you will get a prompt to continue, if you want to log in to VMs or docker containers to check some results or errors now is a good time, otherwise press Enter to continue. Next the script will wait for jobs to finish and then run some unit tests. Once all this is completed the code coverage results will be gathered and optionally uploaded to codecov.io. Code coverage reports will be mapped to your currently checked out branch of git.

Troubleshooting

After running integrationtests.sh 5 containers should be started. You can check their status with the command "docker ps":

```
[johanmarcusson@indy-x1 docker]$ docker ps
CONTAINER ID        IMAGE                                    COMMAND                  CREATED
STATUS              PORTS                               NAMES
9d1c5ecd239e        docker_cnaas_dhcpd                       "/bin/sh -c 'supervi…"   3 minutes ago       Up 3
minutes        0.0.0.0:67->67/udp                docker_cnaas_dhcpd_1
0b71d9f984b1        docker_cnaas_api                         "/bin/sh -c 'supervi…"   3 minutes ago       Up 3
minutes        0.0.0.0:443->1443/tcp             docker_cnaas_api_1
d21cfc296bc0        docker.sunet.se/auth-server-poc:latest   "/bin/sh -c 'supervi…"   3 minutes ago       Up 3
minutes        0.0.0.0:2443->1443/tcp            docker_cnaas_auth_1
d933a4324e1c        docker.sunet.se/cnaas/httpd              "/bin/sh -c 'supervi…"   3 minutes ago       Up 3
minutes        1443/tcp, 0.0.0.0:80->1180/tcp   docker_cnaas_httpd_1
6b3bee0d55ce        docker_cnaas_postgres                    "docker-entrypoint.s…"   9 days ago          Up 9
days           0.0.0.0:5432->5432/tcp            docker_cnaas_postgres_1
53c953d2db8f        docker_cnaas_redis                       "docker-entrypoint.s…"   9 days ago          Up 9
days           0.0.0.0:6379->6379/tcp            docker_cnaas_redis_1
```

If you have some other service running on port tcp 443 or udp 67 for example that container will not be able to start. Use "netstat -anp | grep <port>" to find out what program on your computer might be conflicting with the container and stop it.

If integrationtests.sh crashes and it will not stop the containers after exiting, so you might want to manually bring down the containers. You can do this with docker-compose:

```
docker-compose -f docker/docker-compose.yaml down
```

To rebuild just the API container you can use the command:

```
docker-compose build --no-cache cnaas_api
```

Or if you want to try build a specific branch:

```
docker-compose build --no-cache --build-arg BUILDBRANCH=feature.myfeature cnaas_api
```

Resetting / rerun of tests

You might need to run the tests many times to get everything to pass cleanly with a good code coverage. To make things go a bit faster you can skip the DHCP boot / ZTP of the eosaccess switch, then log in via SSH and paste something like this:

```
conf t
hostname mac-0800275C091F
interface Ethernet2
 no channel-group 1 mode active
 switchport mode access
interface Ethernet3
 no channel-group 1 mode active
switchport mode access
int vlan 600
 shutdown
inte vlan1
 ip address dhcp
 dhcp client accept default-route
end
```

Usually eosdist1 does not come back to "fallback" mode on it's portchannel automatically, so you will need to log in and run: `conf t  int eth2 shutdown  no shutdown` end to reset it.

You also need to reset the postgres database, the easiest way to do this while the docker container is run running is simply: `docker volume rm cnaas-postgres-data && docker volume create cnaas-postgres-data`

Then start integrationtests.sh again and to trigger the DHCP hook to start ZTP log in to eosaccess and run: `conf t  int vlan1  shutdown  no shutdown  end` , check the logs of the DHCP container and wait for the DHCP lease to be given out and make sure it added a new device to the database. When that is done do shut/no shut on vlan1 once more to trigger discovery of the device, this will take a few minutes but then the integration tests should run through.

If you don't have enough memory or don't want to start eosdist2 you can skip that VM entirely, but then you have to manually "fake" the neighbor in the database by inserting this row into PostgreSQL exactly at the point where eosaccess got it's first DHCP lease:

```
docker exec -i -t docker_cnaas_postgres_1 psql -U cnaas
INSERT INTO linknet (device_a_id, device_b_id, device_a_port, device_b_port) VALUES (3, 2, 'Ethernet3',
'Ethernet2');
```

Running the API outside of docker

For faster development and debugging you might want to run just the python API part on your local system instead of in a docker container. This is described in https://github.com/SUNET/cnaas-nms README

The docker image runs debian 10 which uses Python3.7.3. If your system python is not using this version you might want to use pyenv:

```
apt-get/dnf install pyenv
pyenv virtualenv 3.7.3 cnaas373
cd cnaas-nms/
cp etc/db_config.yml.sample /etc/cnaas-nms/db_config.yml
pyenv local cnaas373
python3 -m pip install -r requirements.txt
cd src/
python3 -m cnaas_nms.run
```

You will need to have postgres and redis running to be able to start the API. You can start just those containers with:

```
cd docker/
docker-compose up --no-deps -d cnaas_postgres
docker-compose up --no-deps -d cnaas_redis
```