CNaaS NMS Installation

Prerequisites

1. Linux management server connected to two distribution switches
2. Multiple free /24 networks for IPv4 management addressing
3. Somewhere to host git repositories (you need to create three new repositories)

Allocate IP-addresses

Allocate management addresses in your IPAM for:

1. A /24 for management loopbacks, used on dist/core devices for management
2. A block of many /24s for management of access switches, one /24 per pair of dist-switches
3. A block of many networks (each /24 or smaller) for ZTP of access switches
4. 2 /31 linknets from management server to dist switches

Assign internal IPs for fabric infrastructure, these does not have to be routable outside of CNaaS and can overlap other VRFs unless you need to extend your VXLAN fabric outside of the campus

1. A /16 for infra linknets (linknets between VXLAN devices)
2. A /16 for infra loopbacks (loopbacks used for routing protocols and VTEPs)

Git repositories

You need to set up three new git repositories:

1. cnaas-example-templates
2. cnaas-example-settings
3. cnaas-example-etc

You then need to initialize the repositories and add some data.

Templates repo

An example for the templates repository can be found at https://github.com/SUNET/cnaas-nms-templates , you can try cloning this repository to your local PC and have a look.

To initialize a new bare repository you have setup on a server:

```
git clone git@gitserver.yourdomain.com:cnaas-example-templates.git
cd cnaas-example-templates
git init
```

To add some data to your own repository we suggest starting meld (https://meldmerge.org/) and do a directory comparison from the cnaas-nms-templates repo from github and the cnaas-example-templates directory. Copy the files/directories from cnaas-nms-templates to cnaas-example-templates.

For Arista you will need to modify the template eos/dist.j2 and change "ip helper-address" on vlan1 to point to your VM running the CNaaS-NMS docker containers.

Then add/commit/push:

```
git add .
git commit -a -m "init"
git push
```

Settings repo

For the settings repository you can initiate it with some empty directories and files for now:

```
mkdir {access,core,devices,dist,fabric,global,groups}
touch global/{base_system.yml,groups.yml,routing.yml,vxlans.yml}
touch {access,core,dist,fabric}/base_system.yml
touch devices/README
touch groups/README
```

Edit global/routing.yml and configure the IP blocks you allocated before:

```
underlay:
  infra_link_net: 10.198.0.0/16
  infra_lo_net: 10.199.0.0/16
  mgmt_lo_net: 10.100.3.0/24
```

Edit global/groups.yml and add the group ALL:

```
groups:
  - group:
      name: 'ALL'
      regex: '.*'
```

Commit and push

etc repo

For etc repository add a directory called dhcpd and then add a file called dhcpd.conf in there based on https://github.com/SUNET/cnaas-nms/blob/develop/docker/dhcpd/dhcpd.conf to dhcpd/dhcpd.conf

```
mkdir -p dhcpd && touch dhcpd/dhcpd.conf
```

In dhcd.conf replace the last subnet definition with the ZTP network for access switches you want to use for this distribution switch pair. If you have many pairs of distribution switches and ZTP networks you need to copy the subnet definition for each ZTP network.

Commit and push

Management server

Install a Linux server with docker and docker-compose as a management server. Connect it redundantly to two distribution switches.

Create a docker-compose.yml file on the server with the following contents:

```
---
version: '3.7'
services:
  cnaas_api:
    image: docker.sunet.se/cnaas/api:stable
    ports:
      - 443:1443
    networks:
      - cnaas
    environment:
      - GITREPO_TEMPLATES
      - GITREPO_SETTINGS
      - GITREPO_ETC
      - USERNAME_DHCP_BOOT
      - PASSWORD_DHCP_BOOT
      - USERNAME_DISCOVERED
      - PASSWORD_DISCOVERED
      - USERNAME_INIT
      - PASSWORD_INIT
      - USERNAME_MANAGED
      - PASSWORD_MANAGED
      - FIRMWARE_URL
      - TEMPLATE_SECRET_ADMIN_HASH
      - TEMPLATE_SECRET_TRANSCEIVER
      - TEMPLATE_SECRET_RADIUS
    depends_on:
      - "cnaas_postgres"
      - "cnaas_redis"
      - "cnaas_httpd"
    volumes:
      - type: volume
        source: cnaas-templates
        target: /opt/cnaas/templates
      - type: volume
        source: cnaas-settings
        target: /opt/cnaas/settings
      - type: volume
```

```
        source: cnaas-jwtcert
        target: /opt/cnaas/jwtcert
      - type: volume
        source: cnaas-cacert
        target: /opt/cnaas/cacert

  cnaas_httpd:
    image: docker.sunet.se/cnaas/httpd:latest
    ports:
      - 80:1180
    networks:
      - cnaas
    environment:
      - GITREPO_TEMPLATES
    volumes:
      - type: volume
        source: cnaas-www-data
        target: /opt/cnaas/www/

  cnaas_dhcpd:
    image: docker.sunet.se/cnaas/dhcpd:latest
    ports:
      - 67:67/udp
    environment:
      - GITREPO_TEMPLATES
      - GITREPO_SETTINGS
      - GITREPO_ETC
      - DB_PASSWORD
      - DB_HOSTNAME=cnaas_postgres
      - JWT_AUTH_TOKEN="eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiJ9.
eyJpYXQiOjE1NzEwNTk2MTgsIm5iZiI6MTU3MTA1OTYxOCwianRpIjoiNTQ2MDk2YTUtZTNmOS00NzFlLWE2NTctZWFlYTZkNzA4NmVhIiwic3ViIj
oiYWRtaW4iLCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.
Sfffg9oZg_Kmoq7Oe8IoTcbuagpP6nuUXOQzqJpgDfqDq_GM_4zGzt7XxByD4G0q8g4gZGHQnV14TpDer2hJXw"
    networks:
      - cnaas

  cnaas_postgres:
    image: docker.sunet.se/cnaas/postgres:latest
    volumes:
      - cnaas-postgres-data:/var/lib/postgresql/data
    environment:
      - POSTGRES_USER
      - POSTGRES_PASSWORD
      - POSTGRES_DB=cnaas
    networks:
      - cnaas

  cnaas_redis:
    image: redis:latest
    networks:
      - cnaas

networks:
  cnaas:
    driver: bridge
    name: cnaas
    ipam:
      config:
      - subnet: 172.30.0.0/24
    driver_opts:
      com.docker.network.bridge.name: br-cnaas

volumes:
  cnaas-templates:
    external: true
  cnaas-settings:
    external: true
  cnaas-postgres-data:
    external: true
  cnaas-jwtcert:
    external: true
  cnaas-cacert:
    external: true
  cnaas-www-data:
    external: true
```

:latest here should be replaced with specific version or stable once we have a stable release.

The environment variable defininitions controls much of how CNaaS NMS is run. You can ether configure the variables right in this file by typing GITREPO_TEMPLATES="https://github.com/..." etc or set the environment variables before running the docker commands to start the docker containers.

Set USERNAME_*/PASSWORD_* to your desired passwords for the switches, if using default dhcp-init.j2 user/pass for dhcp/discovered/init is admin /abc123abc123, for MANAGED you should probably use something more secure. The TEPLATE_SECRET_ADMIN_HASH should be a password hash for the password you set under PASSWORD_MANAGED.

"FIRMWARE_URL" should point to the HTTP container and the path where firmwares are stored. This will most likely be the IP address of the NMS server with the prefix "/firmware". For example "<IP address of your NMS>/firmware/"

Create the three needed persistent docker volumes:

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

Start docker-compose and check the logs to try and fix any broken environment variables such as git repositories:

```
docker-compose up -f ./docker-compose.yml
```

The API container will need the public key for a JWT certificate for authenticating API/JWT tokens to start. You can generate a certificate using the commands described under the "Authentication tokens" section below or just copy in a dummy cert in the meantime:

```
docker cp /etc/cnaas-nms/public.pem docker_cnaas_api_1:/opt/cnaas/jwtcert/public.pem
```

The API container will also need a rootCA if you want to generate valid certificates for API services on devices, you can create a rootCA via:

```
docker exec -u root -it docker_cnaas_api_1 /opt/cnaas/createca.sh
```

Check everything is up and running:

```
docker ps
```

```
docker logs -f docker_cnaas_dhcpd_1
```

SSL certificates

Valid SSL certificates can be copied into the container using docker cp and placed at /etc/nginx/conf.d/cnaas_nakeoil.crt and .key , after you have copied the files reload nginx: docker exec -u root -it cnaas_front_cnaas_front_1 nginx -s reload

Authentication tokens

JSON Web Tokens (JWT https://jwt.io/) are used to authenticate users. Setup howto for CNaaS auth poc server .

Initialize database

Once the databases and API are up you can create your first initial dist switches. Start by checking if you can access the API (the JWT token has to be signed by the cert(auth-server) you copied in to the api container earlier. the CNAASURL can very depending on setup):

```
export JWT_AUTH_TOKEN="eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiJ9.
eyJpYXQiOjE1NzEwNTk2MTgsIm5iZiI6MTU3MTA1OTYxOCwianRpIjoiNTQ2MDk2YTUtZTNmOS00NzFlLWE2NTctZWFlYTZkNzA4NmVhIiwic3ViIj
oiYWRtaW4iLCJmcmVzaCI6ZmFsc2UsInR5cGUiOiJhY2Nlc3MifQ.
Sfffg9oZg_Kmoq7Oe8IoTcbuagpP6nuUXOQzqJpgDfqDq_GM_4zGzt7XxByD4G0q8g4gZGHQnV14TpDer2hJXw"
export CNAASURL="https://localhost"
curl -s -H "Authorization: Bearer ${JWT_AUTH_TOKEN}" ${CNAASURL}/api/v1.0/system/version
```

If you don't have installed valid certificates you will need to add -k to curl to make it accept self signed certificates. This curl command should return a result with status success but no data.

Before adding the dist devices you must make the API pull from the templates and settings repositories:

```
curl -s -H "Authorization: Bearer $JWT_AUTH_TOKEN" ${CNAASURL}/api/v1.0/repository/templates -d '{"action":
"refresh"}' -X PUT -H "Content-Type: application/json"
curl -s -H "Authorization: Bearer $JWT_AUTH_TOKEN" ${CNAASURL}/api/v1.0/repository/settings -d '{"action":
"refresh"}' -X PUT -H "Content-Type: application/json"
```

If there's any syntax files in settings you need to fix them and then run the refresh command again.

You then need to add your distribution switches with data similar to this:

Dist1

```
{
  "hostname": "eosdist1",
  "management_ip": "10.100.3.0",
  "platform": "eos",
  "state": "MANAGED",
  "device_type": "DIST"
}
```

Dist2

```
{
  "hostname": "eosdist2",
  "management_ip": "10.100.3.1",
  "platform": "eos",
  "state": "MANAGED",
  "device_type": "DIST"
}
```

Manually assign the first IPs from your mgmt_lo_net pool you allocated before.

You can POST the data to the API with a command like this:

```
curl -s -H "Authorization: Bearer $JWT_AUTH_TOKEN" ${CNAASURL}/api/v1.0/device -X POST -d '{"hostname":
"eosdist1", "platform": "eos", "state": "MANAGED", "device_type": "DIST"}' -H "Content-Type: application/json"
```

And again for dist2.

Create a linknet between dist1 and dist2 if they are directly connected:

```
curl -s -H "Authorization: Bearer $JWT_AUTH_TOKEN" ${CNAASURL}/api/v1.0/linknets -X POST -d '{"device_a":
"eosdist1", "device_a_port": "Ethernet3", "device_b": "eosdist2", "device_b_port": "Ethernet3"}' -H "Content-
Type: application/json"
```

Create a management domain for the access switches connected to this pair of distribution switches:

```
curl -s -H "Authorization: Bearer $JWT_AUTH_TOKEN" ${CNAASURL}/api/v1.0/mgmtdomains -X POST -d '{  "ipv4_gw":
"10.0.6.1/24",  "device_a": "eosdist1",  "device_b": "eosdist2",  "vlan": 600}' -H "Content-Type: application
/json"
```

Manually configure the interfaces from dist to your management server by editing the yaml files in the settings repository. Create a new directory under settings/devices/ with the name of your distribution switch. In that directory create three empty files called base_system.yml, interfaces.yml and routing.yml . Edit interfaces.yml and add something like:

```
---
interfaces:
  - name: Ethernet1
    ifclass: custom
    config: |-
      no switchport
      vrf forwarding MGMT
      ip address 10.100.2.101/24
```

Edit routing.yml

```
---
extroute_static:
  vrfs:
    - name: MGMT
      ipv4:
        - destination: 10.100.2.0/16
          nexthop: 10.100.2.1
          name: management servers
          interface: Ethernet1
```

Commit and push.

Make the API pull from the templates and settings repositories:

```
curl -s -H "Authorization: Bearer $JWT_AUTH_TOKEN" ${CNAASURL}/api/v1.0/repository/templates -d '{"action":
"refresh"}' -X PUT -H "Content-Type: application/json"
curl -s -H "Authorization: Bearer $JWT_AUTH_TOKEN" ${CNAASURL}/api/v1.0/repository/settings -d '{"action":
"refresh"}' -X PUT -H "Content-Type: application/json"
```

If there's any syntax files in settings you need to fix them and then run the refresh command again.

Initialize distribution switches

You should then be able to generate the config for the distribution devices:

```
curl -s  -H "Authorization: Bearer $JWT_AUTH_TOKEN" "${CNAASURL}/api/v1.0/device/eosdist/generate_config" | sed 's
/\\n/\n/g'
```

Copy and paste this config onto the switches (or copy it as a file and apply). CNaaS-NMS should then be able to access the devices over the network and make future changes via the normal device_syncto API calls.

Access switches connected to the distribution switches should be able to boot via DHCP and then be initialized via the API.

Troubleshooting tips

If ZTP/DHCP is not working for access switches: Make sure the IP helper in the templates are correct and point to the management server running the dhcpd container.

API complains about invalid settings in settings repo: Use the settings model API to figure out what variables the API expects: https://cnaas-nms.readthedocs.io/en/latest/apiref/settings.html , you can also try POSTing specific settings to the settings API to make sure they pass the checks before committing/pushing/refreshing.

DHCP container does not start correctly: Make sure you have kept the block 172.30.0.0 in dhcpd.conf since this is the network docker uses internally and isc-dhcpd requires that a local interface is defined in the config.

curl complains: Make sure a valid SSL certificate is installed or use -k.

curl output is hard to read: Install the command "jq" and pipe output like so: curl https://... | jq

Web UI

Installing CNaaS Front WebUI