O-RAN OAM simulation

Network Topology Simulator (NTS)

The Network Topology Simulator is a framework that allows simulating devices that expose a management interface through a NETCONF/YANG interface and VES interface.

Description

Overview

The NETCONF/YANG management interface is simulated, and any valid YANG models can be loaded by the framework to be exposed. The term 'valid yang' refers to the O-RAN YANG Guidelines, which basically is a reference to the RFC8047 - Guidelines for Authors and Reviewers of Documents Containing YANG Data Models .

Random data is generated based on the specific models, such that each simulated device presents different data on its management interface.

The NTS Manager can be used to specify the simulation details and to manage the simulation environment at runtime.

The NTS framework is based on several open-source projects:

- Netopeer2
- libnetconf2
- libyang
- sysrepo - all of these are used for the implementation of the NETCONF Server, both in the NTS Manager and in each simulated device
- cJSON - used to create the JSON payloads for talking with the simulation framework
- pyang - used to create random data from the YANG models that are exposed

Each simulated device is represented as a docker container, where the NETCONF Server is running. The creation and deletion of docker containers associated with simulated devices is handled by the NTS Manager. The NTS Manager is also running as a docker container and exposes a NETCONF/YANG interface to control the simulation.

NTS Manager

The purpose of the NTS Manager is to ease the utilization of the NTS framework. It enables the user to interact with the simulation framework through a NETCONF/YANG interface. The user has the ability to modify the simulation parameters at runtime and to see the status of the current state of the NTS. The NETCONF/YANG interface will be detailed below.

```
module: network-topology-simulator
+--rw simulator-config!
| +--rw simulated-devices? uint32
| +--rw mounted-devices? uint32
| +--rw netconf-call-home? boolean
| +--rw notification-config
| | +--rw fault-notification-delay-period* uint32
| | +--rw ves-heartbeat-period? uint32
| | +--rw is-netconf-available? boolean
| | +--rw is-ves-available? boolean
| +--rw controller-details
| | +--rw controller-ip? inet:ip-address
| | +--rw controller-port? inet:port-number
| | +--rw netconf-call-home-port? inet:port-number
| | +--rw controller-username? string
| | +--rw controller-password? string
| +--rw ves-endpoint-details
| +--rw ves-endpoint-ip? inet:ip-address
| +--rw ves-endpoint-port? inet:port-number
| +--rw ves-endpoint-auth-method? authentication-method-type
| +--rw ves-endpoint-username? string
| +--rw ves-endpoint-password? string
| +--rw ves-endpoint-certificate? string
| +--rw ves-registration? boolean
+--ro simulator-status
+--ro simulation-usage-details
| +--ro running-simulated-devices? uint32
```

```
| +--ro running-mounted-devices? uint32
| +--ro ssh-connections? uint32
| +--ro tls-connections? uint32
| +--ro base-netconf-port? uint32
| +--ro cpu-usage? percent
| +--ro mem-usage? uint32
+--ro notification-count
| +--ro total-ves-notifications
| | +--ro normal? uint32
| | +--ro warning? uint32
| | +--ro minor? uint32
| | +--ro major? uint32
| | +--ro critical? uint32
| +--ro total-netconf-notifications
| +--ro normal? uint32
| +--ro warning? uint32
| +--ro minor? uint32
| +--ro major? uint32
| +--ro critical? uint32
+--ro simulated-devices-list* [uuid]
+--ro uuid string
+--ro device-ip? string
+--ro device-port* uint32
+--ro is-mounted? boolean
+--ro operational-state? operational-state-type
+--ro notification-count
+--ro ves-notifications
| +--ro normal? uint32
| +--ro warning? uint32
| +--ro minor? uint32
| +--ro major? uint32
| +--ro critical? uint32
+--ro netconf-notifications
+--ro normal? uint32
+--ro warning? uint32
+--ro minor? uint32
+--ro major? uint32
+--ro critical? uint32

rpcs:
+---x restart-simulation
+---x add-key-pair-to-odl
+---x invoke-notification
+---w input
| +---w device-id string
| +---w yang-module-name string
| +---w notification-object string
+--ro output
+--ro status enumeration
```

Detailed information about the YANG attributes

Configuration

- **simulated-devices** - represents the number of simulated devices. The default value is 0, meaning that when the NTS is started, there are no simulated devices. When this value is increased to **n**, the NTS Manager starts docker containers in order to reach **n** simulated devices. If the value is decreased to **k**, the NTS Manager will remove docker containers, until the number of simulated devices reaches **k**;
- **mounted-devices** - represents the number of devices to be mounted to an ODL based SDN Controller. The same phylosophy as in the case of the previous leaf applies. If this number is increased, the number of ODL mountpoints increases. Else, the simulated devices are being unmounted from ODL. The number of mounted devices cannot exceed the number of simulated devices. The details about the ODL controller where to mount/unmount are given by the **controller-details** container; **Please note that this cannot be set to a value > 0 if the *ves-registration* leaf is set to 'True'**; For each simulated device, a number of NETCONF endpoints will be mounted, according to the **ssh-connections** and **tls-connections** leafs. See **NETCONF Endpoints** section for more details;

- **netconf-call-home** - if set to `true`, each simulated device will try to use NETCONF Call Home feature and try to reach the ODL Controller. The default value is `false`.
- **notification-config** - this container groups the configuration about fault notification generation of each simulated device;
- **fault-notification-delay-period** - the amount of seconds between two generated fault notifications. For example, if this has a value of *10*, each simulated device will generate a **random** fault notification every *10* seconds; **when this is set to 0, it will reset the notification counters for the VES and NETCONF notifications, which are exposed in the simulator-status**; The type is a leaf-list, such that the user could define a pattern for sending the notifications. E.g.: [10, 3, 5] means that a notification will be sent after 10 seconds, then after another 3 seconds, then after 5 seconds, and then again after 10, 3, 5 etc.
- **ves-heartbeat-period** - the amount of seconds between VES heartbeat messages that can be generated by each simulated device. The details about the VES connection endpoint are given in the **ves-endpoint-details** container;
- **is-netconf-available** - if set to 'True', NETCONF notifications will be sent when a random fault notification is generated, The NETCONF notification that is being sent is currently *o-ran-fm: alarm-notif*; if set to 'False', NETCONF notifications are not being sent out;
- **is-ves-available** - if set to 'True', VES *faultNotification* messages will be sent when a random fault notification is generated; if set to 'False', VES *faultNotification* messages are not generated;
- **controller-details** - this container groups the configuration related to the ODL based SDN controller that the simulated devices can connect to;
- **controller-ip** - the IP address of the ODL based SDN controller where the simulated devices can be mounted. Both IPv4 and IPv6 are supported;
- **controller-port** - the port of the ODL based SDN controller;
- **netconf-call-home-port** - the NETCONF Call Home port of the ODL based SDN controller;
- **controller-username** - the username to be used when connecting to the ODL based SDN controller;
- **controller-password** - the password to be used when connecting to the ODL based SDN controller;
- **ves-endpoint-details** - this container groups the configuration related to the VES endpoint where the VES messages are targeted;
- **ves-endpoint-ip** - the IP address of the VES endpoint where VES messages are targeted;
- **ves-endpoint-port** - the port address of the VES endpoint where VES messages are targeted;
- **ves-endpoint-auth-method** - the authentication method to be used when sending the VES message to the VES endpoint. Possible values are:
    - *no-auth* - no authentication;
    - *cert-only* - certificate only authentication; in this case the certificate to be used for the communication must be configured;
    - *basic-auth* - classic username/password authentication; in this case both the username and password need to be configured;
    - *cert-basic-auth* - authentication that uses both username/password and a certificate; all three values need to be configured in this case;
- **ves-endpoint-username** - the username to be used when authenticating to the VES endpoint;
- **ves-endpoint-password** - the password to be used when authenticating to the VES endpoint;
- **ves-endpoint-certificate** - the certificate to be used when authenticating to the VES endpoint;
- **ves-registration** - if this is set to 'True' **when simulated devices are starting**, each simulated device will send out *pnfRegistration* VES messages to the configured VES endpoint; if this is set to 'False', *pnfRegistration* VES messages will not be sent out. **Please note that this cannot be set to 'True' is simulated devices are already mounted to ODL based SDN controller (mounted-devices leaf > 0)**; For each simulated device, **ssh-connections + tls-connections** p nfRegistration VES messages will be sent out. See **NETCONF Endpoints** section for more details.

Status

- **simulation-usage-details** - this container groups the information about the current simulator status;
- **running-simulated-devices** - the current number of running simulated devices;
- **running-mounted-devices** - the current number of running simulated devices that have been mounted to the ODL based SDN controller; For each simulated device, 10 NETCONF endpoints will be mounted (7 SSH + 3 TLS). See **NETCONF Endpoints** section for more details.
- **ssh-connections** - represents the number of SSH endpoints that are exposed by each of the simulated devices. **Please note that the total number of SSH and TLS connections cannot exceed 100.** The default value is 1. **The value can only be changed when the NTS Manager is started, through the SshConnections environment variable**.
- **tls-connections** - represents the number of TLS endpoints that are exposed by each of the simulated devices. **Please note that the total number of SSH and TLS connections cannot exceed 100.** The default value is 0. **The value can only be changed when the NTS Manager is started, through the SshConnections environment variable**.
- **base-netconf-port** - the port that was used as a base when craeting simulated devices;
- **cpu-usage** - the percentage of the CPU used currently by the simulation framework;
- **mem-usage** - the amount of RAM used (in MB) currently by the simulation framework;
- **notification-count** - this container groups the details about the total number of notifications that were generated by the simulated devices;
- **total-ves-notifications** - this container groups the details about the total number of VES notifications that were generated by the simulated devices, grouped by severity;
- **total-netcnof-notifications** - this container groups the details about the total number of NETCONF notifications that were generated by the simulated devices - grouped by severity;

- **simulated-devices-list** - this list contains the details about each simulated devices that is currently running;
- **uuid** - the Universally Unique ID of the simulated device;
- **device-ip** - the IP address of the simulated device;
- **device-port** - the port of the simulated device, where the NETCONF connection is exposed;
- **is-mounted** - boolean to show whether the device is currently mounted to an ODL based SDN controller;
- **operational-state** - the operational state of the current simulated device; it can be either *not-specified*, *created*, *running* or *exited*;
- **notification-count** - this container groups the details about the number of notifications that were generated by this particular simulated device;
- **ves-notifications** - this container groups the details about the number of VES notifications that were generated by this simulated device, grouped by severity;
- **netconf-notifications** - this container groups the details about the number of NETCONF notifications that were generated by this simulated device - grouped by severity.

RPCs

- **add-key-pair-to-odl** - this RPC can be used to trigger the loading of a *keystore* entry in an ODL based SDN controller such that the controller can connect to the simulated devices via **TLS**. A private key, an associated certificate and a trusted certificate are loaded in the *keystore* entry in ODL. The certificate associated with the private key to be used by ODL in the TLS communication is signed by the same CA as the certificates used by the simulated devices, easing the TLS configuration in both the NETCONF Server and the ODL.
- **restart-simulation** - this RPC is not yet implemented.
- **invoke-notification** - this RPC is used for forcing a simulated device to send a NETCONF notification, as defined by the user.
    - The **input** needed by the RPC:
        - **device-id** - this is a string containing the name of the simulated device that we want to send the notification. The user is responsible to give a correct name which really exists, otherwise the RPC will fail.
        - **yang-module-name** - this is a string containing the name of the YANG module which implements the notification that we want to send. E.g.: ***org-openroadm-device*** module defines several notifications.
        - **notification-object** - this is a string containing the notification object that we are trying to send from the simulated device, in JSON format. **Please note that the user has the responsibility to ensure that the JSON object is valid, according to the definition of the notification in the YANG module.** There is no possibility to see what was wrong when trying to send an incorrect notification. The RPC will only respond with an "ERROR" status in that case, without further information. E.g. of a JSON containing a notification object of type ***otdr-scan-result*** defined in the ***org-openroadm-device*** YANG module: ***{ "org-openroadm-device:otdr-scan-result":{"status":"Successful"," status-message":"Scan result was successful","result-file":"/home /result-file.txt"}}***. **Please note that the notification object contains also the name of the YANG model defning it, as a namespace, as seen in the example.**
    - The **output** returned by the RPC:
        - **status** - if the notification was send successfully by the simulated device, the RPC will return a **SUCCESS** value. Else, the RPC will return a **ERROR** value.

Simulated Device

Each simulated device is represented as a docker container, inside which the NETCONF Server runs. The simulated device exposes the YANG models which are found inside the **yang** folder. A custom version of the *pyang* utility is used to generate random data for each of the YANG modules found inside the **yang** folder.

NETCONF Endpoints

Each simulated device exposes a number of NETCONF endpoints which represented by the sum of the **SshConnections** and **TlsConnections** environment variables, on consecutive ports. The first simulated device uses the ports starting from the **NETCONF_BASE** environment variable used when starting the NTS Manager, while the next one uses the next ports and so on and so forth. E.g. if the **NETCONF_BASE=50000** and **SshConnections=5** and **TlsConnections=3**, the first simulated device will expose ports from *50000* to *50007*, the second simulated device will expose ports from *5008* to *50015* etc.

The first **SshConnections** ports exposed by a simulated device are **SSH** based. A NETCONF client can connect to the exposed endpoint using one of the SSH ports (e.g. 50000 to 50007, considering the previous example) and the **username/password**: *netconf/netconf*.

The last **TlsConnections** ports exposed by a simulated device are **TLS** based. A NETCONF client can connect to the exposed endpoint using one of the TLS ports (e.g. 50006 to 50008, considering the previous example), using a valid certificate and the **username**: *netconf*.

Usage

Building the images

The `docker-build-nts-manager.sh` script can be used to built the docker image associated with the NTS Manager. This will create a docker image named *ntsim_manager_light*, which will be used to start the simulation framework. Inside the docker image, port 830 will wait for connections for the NETCONF/YANG management interface.

The `docker-build-onf-core-model-1-2.sh` script can be used to build the docker image associated with a simulated device, exposing the ONF CoreModel version 1.2.

The `docker-build-onf-core-model-1-4.sh` script can be used to build the docker image associated with a simulated device, exposing the ONF CoreModel version 1.4.

The `docker-build-openroadm.sh` script can be used to build the docker image associated with a simulated device, exposing the OpenROADM models.

The `docker-build-o-ran-device.sh` script can be used to build the docker image associated with a simulated device, exposing the O-RAN models.

The `docker-build-o-ran-sc-o-ran-ru.sh` script can be used to build the docker image associated with a simulated device, exposing the O-RAN-SC models.

The `docker-build-x-ran-device.sh*` script can be used to build the docker image associated with a simulated device, exposing the X-RAN models.

Starting the NTS Manager

The NTS Manager can be started using the `docker-compose.yml` file that is provided inside tthe **scripts** folder. Further, the parameters present in this file are explained.

```
version: '3'
services:
  ntsimulator:
    image: "ntsim_manager:latest"
    container_name: NTS_Manager
    ports:
     - "8300:830"
    volumes:
     - "/var/run/docker.sock:/var/run/docker.sock"
     - "/var/tmp/NTS_Manager:/opt/dev/scripts"
     - "/usr/bin/docker:/usr/bin/docker"
    labels:
      "NTS-manager": ""
    environment:
      NTS_IP: "172.17.0.1"
      NETCONF_BASE: 50000
      DOCKER_ENGINE_VERSION: "1.40"
      MODELS_IMAGE: "ntsim_oran"
      VesHeartbeatPeriod: 0
      IsVesAvailable: "true"
      IsNetconfAvailable: "true"
      VesRegistration: "false"
      VesEndpointPort: 8080
      VesEndpointIp: "172.17.0.1"
      SshConnections: 1
      TlsConnections: 0
      K8S_DEPLOYMENT: "false"
      CONTAINER_NAME: "ntsimulator"
      NetconfCallHome: "true"
      NetconfCallHomePort: 6666
      ControllerIp: "10.20.11.121"
      ControllerPort: 8181
      ControllerUsername: "admin"
      ControllerPassword: "admin"
      IPv6Enabled: "true"
```

- Port mapping:
  - `"8300:830"` - this maps the *830* port from inside the docker container of the NTS Manager to the port *8300* from the host, and binds it to any IP address on the host:
- Volumes - these map 3 important things:
  - the docker socket from the host is mapped inside the docker container: `/var/run/docker.sock:/var/run/docker.sock` - **please do not modify the path inside the container!**;
  - any folder from the host can be mapped to inside the docker container into othe **/opt/dev/scripts** folder: `/var/tmp/NTS_Manager:/opt/dev/scripts` - **please do not modify the path inside the container!**;

        - the path to the docker executable needs to be mapped inside the container: `/usr /bin/docker:/usr/bin/docker` - **please do not modify the path inside the container!**;
    - Labels - this associates the *NTS-manager* label to the docker container where the NTS runs;
    - Environment variables:
        - **NTS_IP** - this should point to an IP address **from the host**, through which the simulated devices will be accessed;
        - **NETCONF_BASE** - this is the starting port used to expose NETCONF endpoints. Starting from this, each device will use 10 consecutive ports for its endpoints; **Please note that if multiple managers are deployed on the same machine, this environment variable needs to be different for each of the managers!**
        - **DOCKER_ENGINE_VERSION** - this is the version of the *docker engine* installed currently on the host. This can be verified using `docker version` command in the host, and looking to the `API version: #.##` variable from the Server details.
        - **MODELS_IMAGE** - this represents the name of the docker image that represents the simulated device. The NTS Manager will start containers using this image, when starting simulated devices.
        - **VesHeartbeatPeriod** - this can change the default value of the **ves-heartbeat-period** l eaf used by the NTS Manager.
        - **IsVesAvailable** - this can change the default value of the **is-ves-available** leaf used by the NTS Manager.
        - **IsNetconfAvailable** - this can change the default value of the **is-netconf-available** leaf used by the NTS Manager.
        - **VesRegistration** - this can change the default value of the **ves-registration** leaf used by the NTS Manager.
        - **VesEndpointPort** - this can change the default value of the **ves-endpoint-port** leaf used by the NTS Manager.
        - **VesEndpointIp** - this can change the default value of the **ves-endpoint-ip** leaf used by the NTS Manager.
        - **SshConnections** - **this can change the number of SSH connection endpoints which are exposed by each simulated device.**
        - **TlsConnections** - **this can change the number of TLS connection endpoints which are exposed by each simulated device.**
        - **K8S_DEPLOYMENT** - this value can be set to `true` when the user wants to the NTS Framework in a Kubernetes deployment. The default is `false`.
        - **CONTAINER_NAME** - this represents the name to be used by the NTS Manager for asigning to each simulated device, suffixed by a number. The default is `ntsim`. E.g.: the first simulated device will be *ntsim-0*, the second onoe *ntsim-1* and so on. **Please note that if multiple managers are deployed on the same machine, this environment variable needs to be different for each of the managers!**
        - **ControllerIp** - this can change the default value of the **controller-ip** leaf used by the NTS Manager. The default is `172.17.0.1`.
        - **ControllerPort** - this can change the default value of the **controller-port** leaf used by the NTS Manager. The default is `8181`.
        - **NetconfCallHomePort** - this can change the default value of the **netconf-call-home-port** leaf used by the NTS Manager. The default is `6666`.
        - **ControllerUsername** - this can change the default value of the **controller-username** l eaf used by the NTS Manager. The default is `admin`.
        - **ControllerPassword** - this can change the default value of the **controller-password** le af used by the NTS Manager. The default is `admin`.
        - **NetconfCallHome** - this can change the default value of the **netconf-call-home** leaf used by the NTS Manager. The default is `false`.
        - **IPv6Enabled**: if this is set to `true`, IPv6 is enabled. **Please note that for a working configuration, it is the user responsibility to correctly configure the Docker daemon to work with IPv6, prior to starting the NTS Manager!**

After modifying the `docker-compose.yml` file with values specific to your host, the NTS Manager can be started by running the command `docker-compose up` from the **scripts** folder.

After the NTS Manager is started, it will wait for connections on its NETCONF/YANG management interface. One can connect to this using a NETCONF Client. The **username/password** for connecting are: *netconf/netconf*.

Example of `docker ps` command result, after the NTS Manager was started:

```
7ff723b7f794        ntsim_manager:latest   "sh -c '/usr/bin/sup…"   5 days
ago          Up 5 days
```

Using the NTS Manager

When the NTS Manager is started, its default configuration looks like this:

```
<simulator-config xmlns="urn:onf:params:xml:ns:yang:network-topology-simulator"
>
    <simulated-devices>0</simulated-devices>
    <mounted-devices>0</mounted-devices>
```

```
    <netconf-call-home>false</netconf-call-home>
    <notification-config>
        <fault-notification-delay-period>0</fault-notification-delay-period>
        <ves-heartbeat-period>0</ves-heartbeat-period>
        <is-netconf-available>true</is-netconf-available>
        <is-ves-available>true</is-ves-available>
    </notification-config>
    <controller-details>
        <controller-ip>172.17.0.1</controller-ip>
        <controller-port>8181</controller-port>
        <netconf-call-home-port>6666</netconf-call-home-port>
        <controller-username>admin</controller-username>
        <controller-password>admin</controller-password>
    </controller-details>
    <ves-endpoint-details>
        <ves-endpoint-ip>172.17.0.1</ves-endpoint-ip>
        <ves-endpoint-port>30007</ves-endpoint-port>
        <ves-endpoint-auth-method>no-auth</ves-endpoint-auth-method>
        <ves-registration>false</ves-registration>
    </ves-endpoint-details>
</simulator-config>
```

This configuration can be altered by connecting to the NTS Manager with a NETCONF Client.

Starting a simulated device

Example RPC for starting **one** simulated device:

```
<?xml version="1.0" encoding="utf-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <simulator-config xmlns="urn:onf:params:xml:ns:yang:network-topology-
simulator">
        <simulated-devices>1</simulated-devices>
      </simulator-config>
    </config>
  </edit-config>
</rpc>
```

If the leaf `<simulated-devices>1</simulated-devices>` will be set to a value of **1**, the NTS Manager will start a new docker container. We can verify that this was successfull by running the `docker ps` command. The results will look like this:

```
c18eb7a362f5        ntsim_oran             "sh -c '/usr/bin/sup…"   4 days
ago          Up 4 days           172.17.0.1:50000->830/tcp, 172.17.0.1:50001-
>831/tcp, 172.17.0.1:50002->832/tcp, 172.17.0.1:50003->833/tcp, 172.17.0.1:
50004->834/tcp, 172.17.0.1:50005->835/tcp, 172.17.0.1:50006->836/tcp,
172.17.0.1:50007->837/tcp, 172.17.0.1:50008->838/tcp, 172.17.0.1:50009->839
/tcp   reverent_bhabha
```

Troubleshooting

No simulated devices are starting

If, after setting the leaf `<simulated-devices>1</simulated-devices>` to a value greater that 0, no new containers are created, please make sure that the image name specified in the **MODELS_IMAGE** environment variable when starting the NTS Manager is present in the host. You can verify that using the `docker images` command.

Example of a result of such a command:

```
ntsim_oran_light       latest           57b065de4458     4 days ago     186MB
```

This means that `MODELS_IMAGE: "ntsim_oran_light:latest"` can be used as an environment variable when starting the NTS Manager.

Release notes

version 0.6.5

Added features:

- **basic-auth** is now available for pnfRegistration messages - if the VES Endpoint supports basic-auth (with username and password), the user can configure these endpoint details and the pnfRegistration message will use them.

version 0.6.4

Bug fixes and improvements:

- Fixed bug where pnfRegistration messages for TLS connections were not sent.
- Fixed bug for manual notification generation failed when notification object was too long.

version 0.6.1

Added features:

- **Manual notification generation** - this enables the user to send a NETCONF notification from a simulated device, using the *invoke-notification* RPC defined in the NTS Manager.
- **automatic pulling of the simulated device image** - the NTS Manager will automatically try to pull the image of the simulated device (as defined in the **MODELS_IMAGE** environment variable) before starting a simulated device.
- **custom naming of simulated devices** - the user can now define its own name prefix for simulated devices, through the **CONTAINER_NAME** environment variable defined in the docker-compose.yml. E.g.: if CONTAINER_NAME: "ntsim", the simulated devices will be: *ntsim-0*, *ntsim -1* etc.

Bug fixes and improvements:

- **ssh-connections** and **tls-connections** are now removed from the simulator-config, and can be set only when the NTS Manager is started, through the **SshConnections** and **TlsConnections** environment variables defined in the docker-compose.yml. The leafs are not moved in the simulator-status, such that the user can check at runting what are the values defined.
- **fault-notification-delay-period** has now the attribute *ordered-by user*, meaning that the order defined by the user when adding multiple values to the leaf-list is preserved.

version 0.5.1

Added features:

- **NETCONF Call Home** - this enables each simulated device to use the Call Home feature to connect to the ODL instance, once it boots up. **Please note that when a device is using Call Home, it will no longed expose all the number of *ssh-connections* and *tls-connections*, but a single Call Home endpoint. **
- **controller-details** configuration leafs are now exposed can now be set at startup time in the docker-compose YAML file, as environment variables.