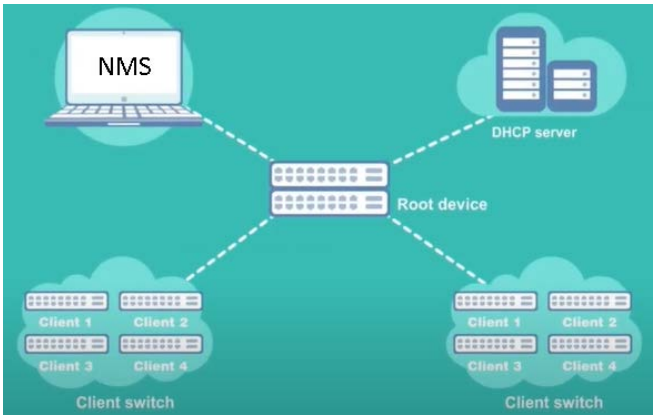
Part 1. Attestation and Provisioning

After devices are installed on a network, administrators need to commission software onsite. If a large number of devices are sparsely distributed on the network, administrators have to manually configure all the devices one by one. This lowers deployment efficiency and increases manpower costs.
Therefore, IT departments of enterprises are facing a big challenge: to automatically manage network devices in batches and one of the best solutions that nowadays exists for this is the Zero Touch Provisioning (ZTP).

ZTP enables devices to be automatically deployed after they are installed on the network and powered on. This facilitates unconfigured device deployment, faulty device replacement, and batch upgrade of software or configurations. It greatly improves efficiency of network management and maintenance, as well as lowering the labor costs.

ZTP has several roles including NMS (network management solution), the DHCP server, the root device, and the client.

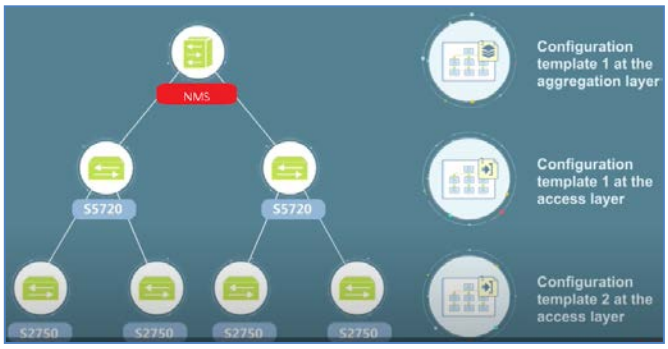


The **NMS** implement network topology planning and generates the configuration files. The **Root Device** is a node that has been deployed and configured connecting to devices to be deployed, which are **client nodes**.

Network engineers can carry out network planning on the NMS and draws the topology of network nodes, including the device type and planning of links and names.

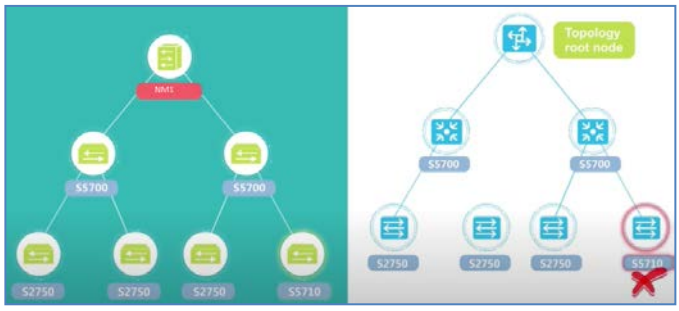
Generally, switches support the template-based configuration: the NMS system provides the default configuration template according to which network engineers generate CLI-based configuration files and specify the device software packages and patch files.

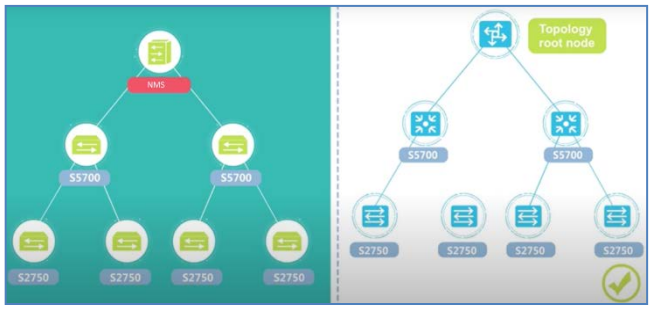
Author: Dlouky Federico Manuel.
Evaluators: Andy Cross, Mitch Bulajic, Annajoy David, Antonio Aliaga Cortés.



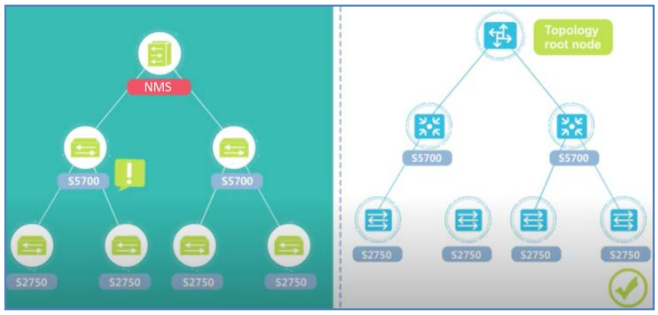
After network devices are deployed, the NMS collects the topologies and compares them with the planned topologies. If there are any differences found in the topologies, the system will prompt a message.

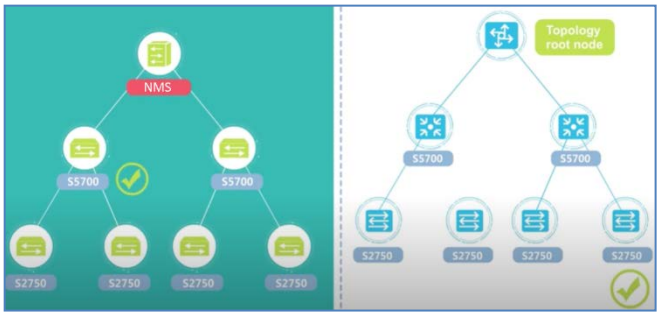
For example, if the network devices are of different types it is required to replace the devices on the live network





If the connection interfaces are inconsistent, modify the network planning and re-connect the devices to the network.



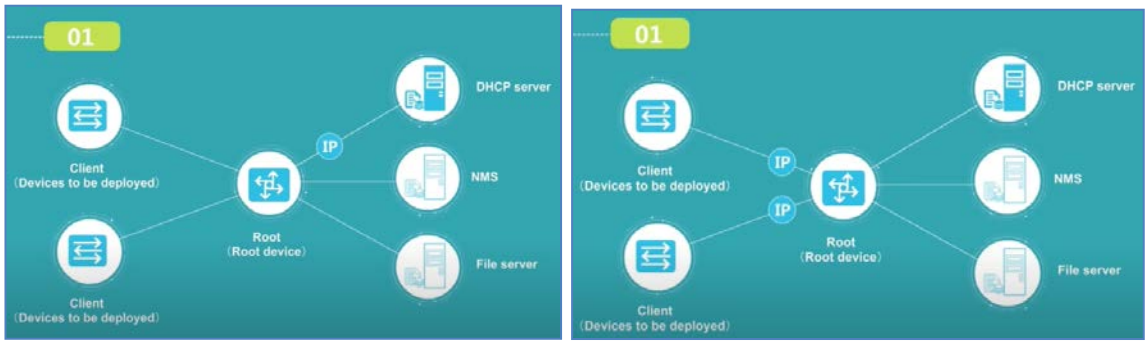


After network administrators verify the device validity, the devices can download the deployment files from the file server.

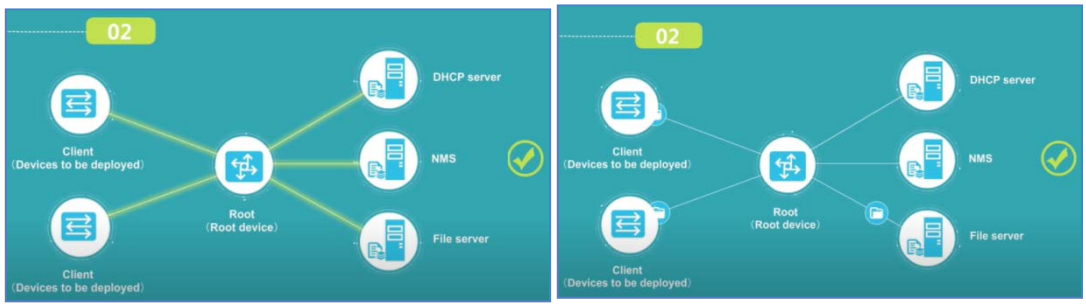
Author: Dlouky Federico Manuel.
Evaluators: Andy Cross, Mitch Bulajic, Annajoy David, Antonio Aliaga Cortés.

In this case, the devices communicate with the **NMS** to complete file downloading and report the deployment progress:

1. After client devices are powered on, the DHCP process is enabled. Client devices obtain IP addresses from the DHCP server and also obtain information about the NMS and the file server.

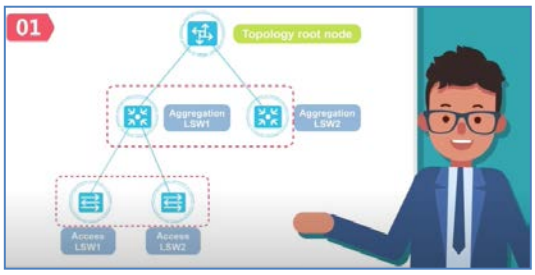


2. After the NMS checks the topologies, unconfigured devices start the ZTP process. Devices to be deployed download relevant files and restart to complete activation.



At the Operation and Maintenance phase, the user can perform visualized operations to upgrade system software or configure network devices in batches:

1. On a topology GUI (graphical user interface) the user can select multiple devices of the same model that need to be upgraded or configured in batches. Specify these devices as a group and customize group-based upgrade policies.



2. The NMS actively sends messages to devices of the group so that devices are informed of the new software package upgrade or the new configuration delivery.

Author: Dlouky Federico Manuel.
Evaluators: Andy Cross, Mitch Bulajic, Annajoy David, Antonio Aliaga Cortés.

Part 2. Device Telemetry

The Message Queue Telemetry Transport (MQTT) protocol is used to transmit messages from the device (publisher) to the IoT Hub (Azure broker). The client (subscriber) will read those messages from IoT Hub.
For this task, I have developed four Python scripts:

1. **AA_device_credentials_generator.py**:
    a. It defines the tasks that the device must execute through the device_function variable.
    b. The function create_credential(device_id) creates a json file with the following content
        i. device_id: it is the serial number provided by the device factory (for this approach it is simulated with a random SHA1 hash).
        ii. device_name: the name is an incremental integer starting from 1000001.
        iii. credential_creation_date: when the credential was created.
        iv. device_function: the tasks that the device must execute.
        v. CONNECTION_STR_PUB: Credentials to publish on IoT HUB
    c. Finally, the device_id, device_name and credential_creation_date are saved in a csv (historical_credentials.csv) to keep a historical record of the generated credentials.
    d. Only the system administrator has the permissions to run create_credential(device_id) to create the json file for the specific device_id, so this script is in the administrator machine.
    e. The device credentials (json files) and the historical_credentials.csv are inside the folder named device_credentials.
2. **AB_device_simulator.py**:
    a. This script will be into each device. It has a device_id provided by the device factory.
    b. If a credential for this device exists into the device_credentials folder, it will download the json file where device_function is present.
    c. It recovers from the json all the tasks it should perform:
        i. Measure the battery level, which will decrease every 10 messages sent to the MQTT broker.
        ii. Generate json message with device_id, device_name, device_latitude, device_longitude, device_battery_level and signal_date (when the signal was emmited).
3. **AC_publisher_IoTHUB.py**:
    a. This script will be in a machine connected to internet and to the devices.
    b. It use the credentials present into azure_mqtt_credentials folder to connect to the IoT HUB.
    c. After a successful connection, it securely sends the messages generated by AB_device_simulator.py to the MQQT broker (endpoint).
4. **AD_suscriber_IoTHUB.py**:
    a. This script will be in client (subscriber) machine.
    b. It securely connects to the IoT Hub through credentials presents in azure_mqtt_credentials.
    c. After a successful connection, it reads the messages send by the device simulator and save them into MongoDB (dabase= db_elastacloud, collection= collection_elastacloud).

Author: Dlouky Federico Manuel.
Evaluators: Andy Cross, Mitch Bulajic, Annajoy David, Antonio Aliaga Cortés.

Part 3. Optimization, Twins and Routing

To achieve the device payload optimization we could edit the device_function variable from the AA_device_credentials_generator.py script. After making all the changes, the administrator should run the create_credential (device_id) function to create a new json file for all the devices. Through ZTP we could create an event so all the devices download the updated json file credential.

Part 4. Downstream

Since AD_suscriber_IoTHUB.py save all the data from IoT HUB to a local MongoDB instance, we can interact with the data using pymongo or processing in a typical Big Data analytical system like Spark. In the AE_DataAnalysis.py script, we can see pymongo and spark in action!

Project

All the project has been uploaded to github: https://github.com/dlouky/IoT_test

Author: Dlouky Federico Manuel.
Evaluators: Andy Cross, Mitch Bulajic, Annajoy David, Antonio Aliaga Cortés.