NSO Hands-On session

WORKBOOK

12 November 2024

Presenter:

Customer Success Specialists - Cross-Domain Automation

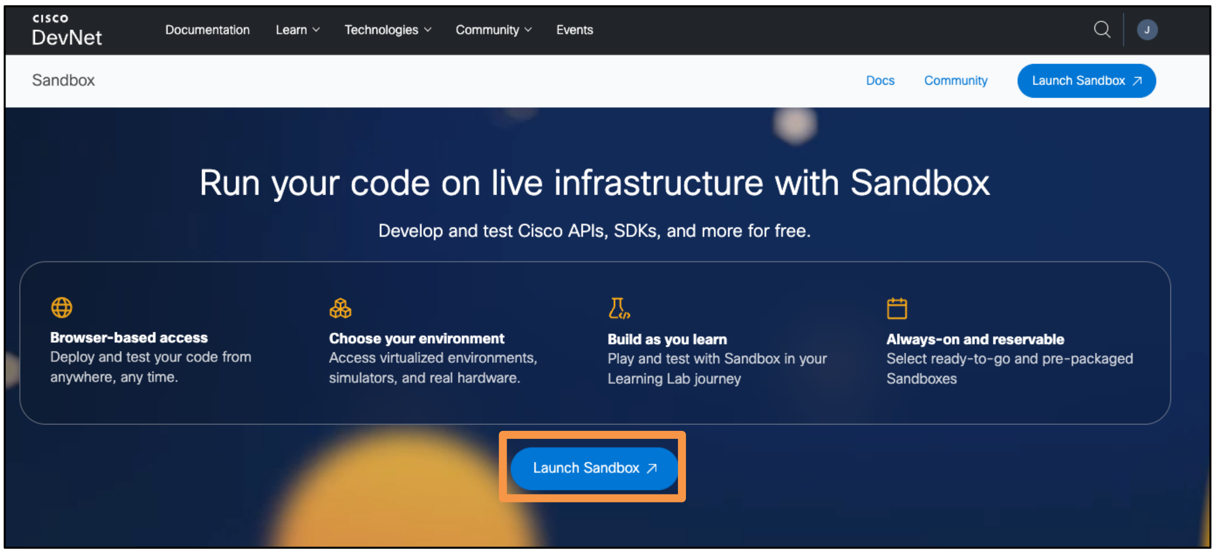
Jorge Mira

Contents

Cisco Network Services Orchestrator

1 - Launch the Devnet Sanbox

Access https://developer.cisco.com/site/sandbox/ and click on the “Launch Sandbox” button.
(Login may be required)



Run your code on live infrastructure with Sandbox

Develop and test Cisco APIs, SDKs, and more for free.

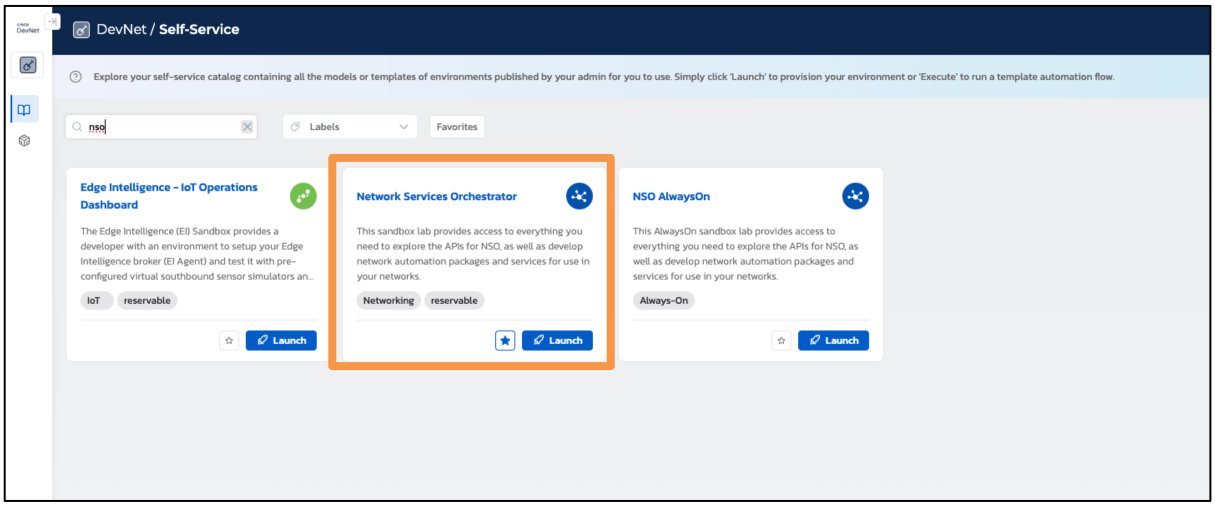
Browser-based access
Deploy and test your code from anywhere, any time.

Choose your environment
Access virtualized environments, simulators, and real hardware.

Build as you learn
Play and test with Sandbox in your Learning Lab journey

Always-on and reservable
Select ready-to-go and pre-packaged Sandboxes

Use the search bar and enter the word “nso”, then click on the “Launch” button for the “Network Services Orchestrator”



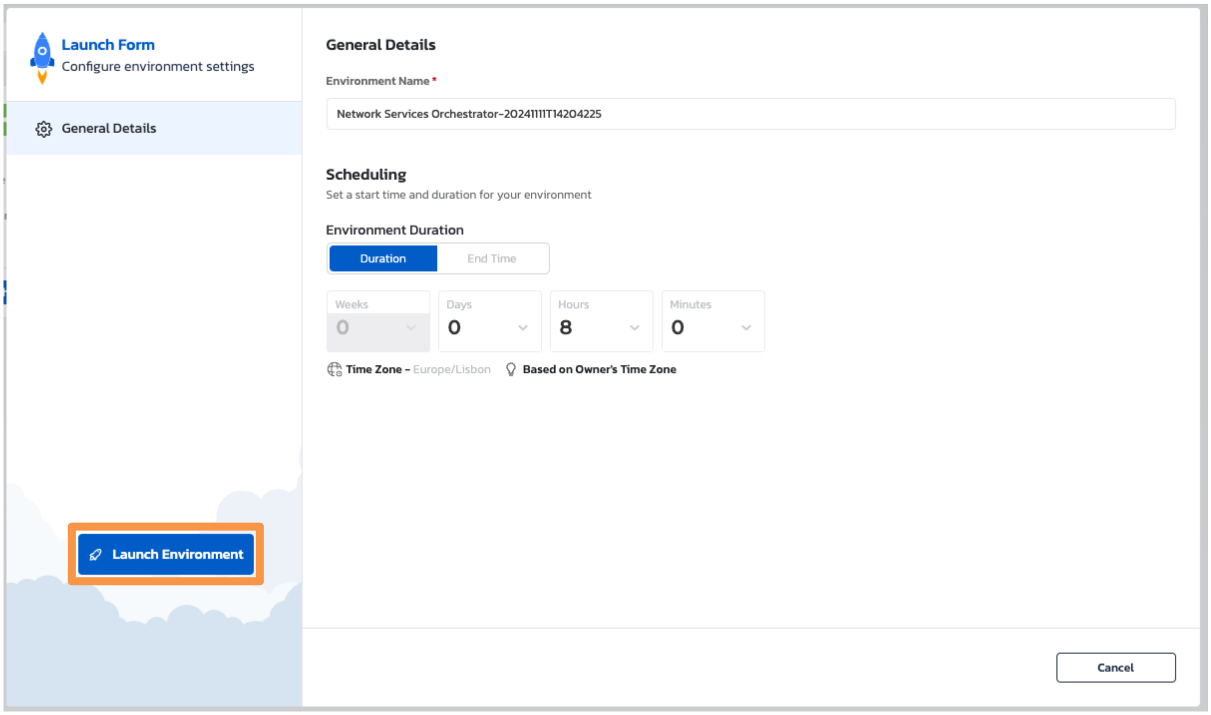
Explore your self-service catalog containing all the models or templates of environments published by your admin for you to use. Simply click 'Launch' to provision your environment or 'Execute' to run a template automation flow.

Edge Intelligence – IoT Operations Dashboard
The Edge Intelligence (EI) Sandbox provides a developer with an environment to setup your Edge Intelligence broker (EI Agent) and test it with pre-configured virtual southbound sensor simulators an...
IoT | reservable

Network Services Orchestrator
This sandbox lab provides access to everything you need to explore the APIs for NSO, as well as develop network automation packages and services for use in your networks.
Networking | reservable

NSO AlwaysOn
This AlwaysOn sandbox lab provides access to everything you need to explore the APIs for NSO, as well as develop network automation packages and services for use in your networks.
Always-On

Choose the duration you wish to book the lab, and click the button “Launch Environment”



General Details

Environment Name *

Network Services Orchestrator-20241111T14204225

Scheduling

Set a start time and duration for your environment

Environment Duration

Duration | End Time

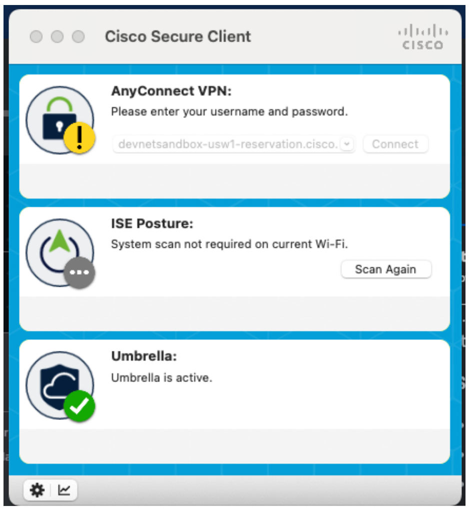
Weeks	Days	Hours	Minutes
0	0	8	0

Time Zone - Europe/Lisbon Based on Owner's Time Zone

Cancel

The lab will now take some time to setup. An e-mail, containing the AnyConnect (VPN) credentials will be sent to you (logged in account) once the lab is ready.

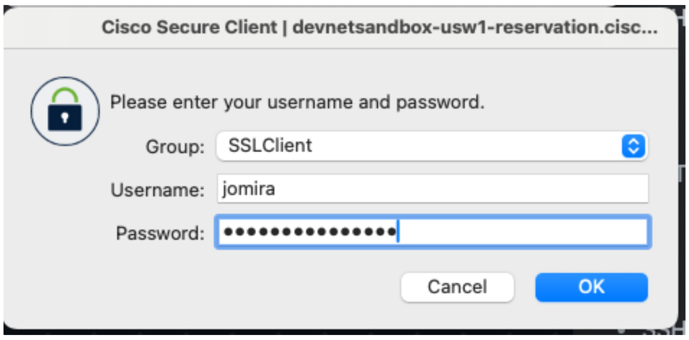
Connect to the Sandbox VPN with the credentials sent by e-mail.


Cisco Secure Client

AnyConnect VPN:
Please enter your username and password.
devnetsandbox-usw1-reservation.cisco. Connect

ISE Posture:
System scan not required on current Wi-Fi.
Scan Again

Umbrella:
Umbrella is active.

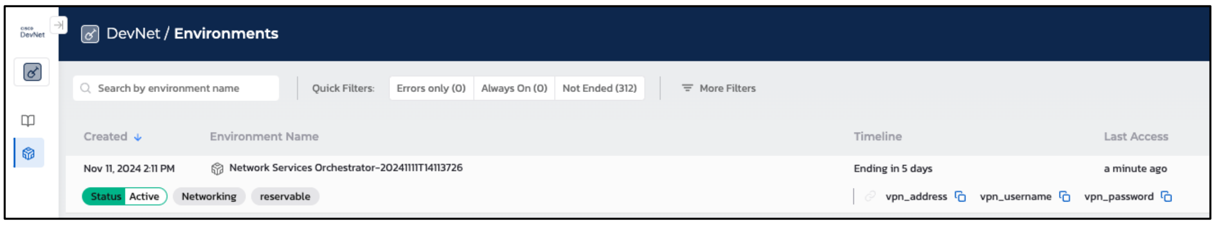

Cisco Secure Client | devnetsandbox-usw1-reservation.cisc...

Please enter your username and password.

Group: SSLClient
Username: jomira
Password: •••••••••••••••

Cancel OK

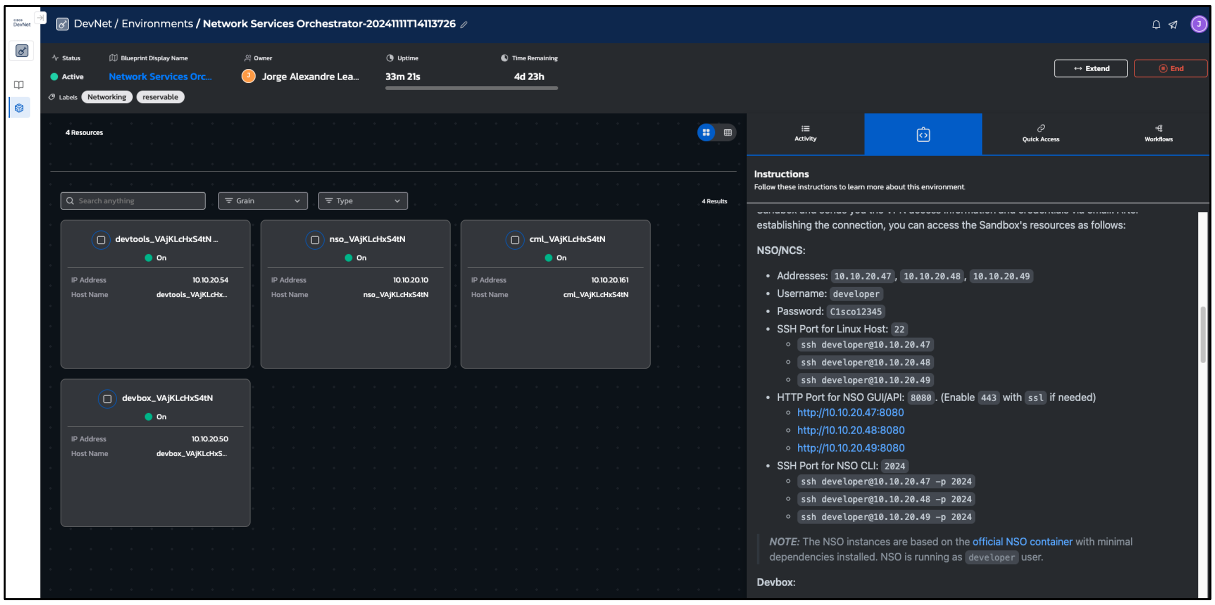
If you wish to check all your booked environments, you can do it by accessing this URL.
https://devnetsandbox.cisco.com/DevNet/environments


DevNet / **Environments**

Search by environment name | Quick Filters: Errors only (0) | Always On (0) | Not Ended (312) | More Filters

Created	Environment Name	Timeline	Last Access
Nov 11, 2024 2:11 PM	Network Services Orchestrator-20241111T14113726	Ending in 5 days	a minute ago
Status Active · Networking · reservable		vpn_address · vpn_username · vpn_password	

Click on the active one. You should see a screen similar to the one bellow.


DevNet / Environments / **Network Services Orchestrator-20241111T14113726**

Status: Active · Blueprint Display Name: Network Services Orc... · Owner: Jorge Alexandre Lea... · Uptime: 33m 21s · Time Remaining: 4d 23h

Labels: Networking, reservable

4 Resources · 4 Results

devtools_VAjKLcHxS4tN ...	On	
IP Address	10.10.20.54	
Host Name	devtools_VAjKLcHx...	
nso_VAjKLcHxS4tN	On	
IP Address	10.10.20.10	
Host Name	nso_VAjKLcHxS4tN	
cml_VAjKLcHxS4tN	On	
IP Address	10.10.20.161	
Host Name	cml_VAjKLcHxS4tN	
devbox_VAjKLcHxS4tN	On	
IP Address	10.10.20.50	
Host Name	devbox_VAjKLcHxS...	

Instructions
Follow these instructions to learn more about this environment.

establishing the connection, you can access the Sandbox's resources as follows:

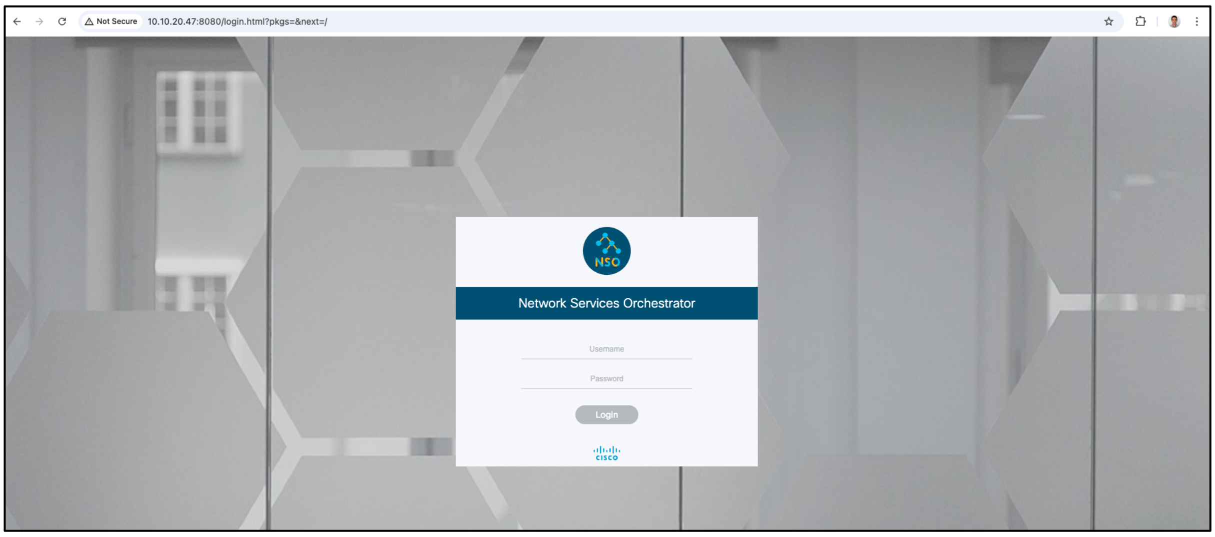
NSO/NCS:

- Addresses: `10.10.20.47`, `10.10.20.48`, `10.10.20.49`
- Username: `developer`
- Password: `C1sco12345`
- SSH Port for Linux Host: `22`
  - `ssh developer@10.10.20.47`
  - `ssh developer@10.10.20.48`
  - `ssh developer@10.10.20.49`
- HTTP Port for NSO GUI/API: `8080`. (Enable `443` with `ssl` if needed)
  - http://10.10.20.47:8080
  - http://10.10.20.48:8080
  - http://10.10.20.49:8080
- SSH Port for NSO CLI: `2024`
  - `ssh developer@10.10.20.47 -p 2024`
  - `ssh developer@10.10.20.48 -p 2024`
  - `ssh developer@10.10.20.49 -p 2024`

> *NOTE:* The NSO instances are based on the official NSO container with minimal dependencies installed. NSO is running as `developer` user.

Devbox:

Test the connectivity by accessing one instance of NSO WebUI (http://10.10.20.47:8080/)


Not Secure
10.10.20.47:8080/login.html?pkgs=&next=/
NSO
Network Services Orchestrator
Username
Password
Login
cisco

Or, using SSH

```
Last login: Mon Nov 11 12:04:07 on ttys001
user@pc ~ % ssh developer@10.10.20.47 -p 2024

The authenticity of host '[10.10.20.47]:2024 ([10.10.20.47]:2024)'
can't be established.
ED25519 key fingerprint is
SHA256:kc0ewQEkFQgnW4CrqfW4oiT3K8DgaXwpKJ85wvctBdk.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
yes
Warning: Permanently added '[10.10.20.47]:2024' (ED25519) to the
list of known hosts.
developer@10.10.20.47's password:
```

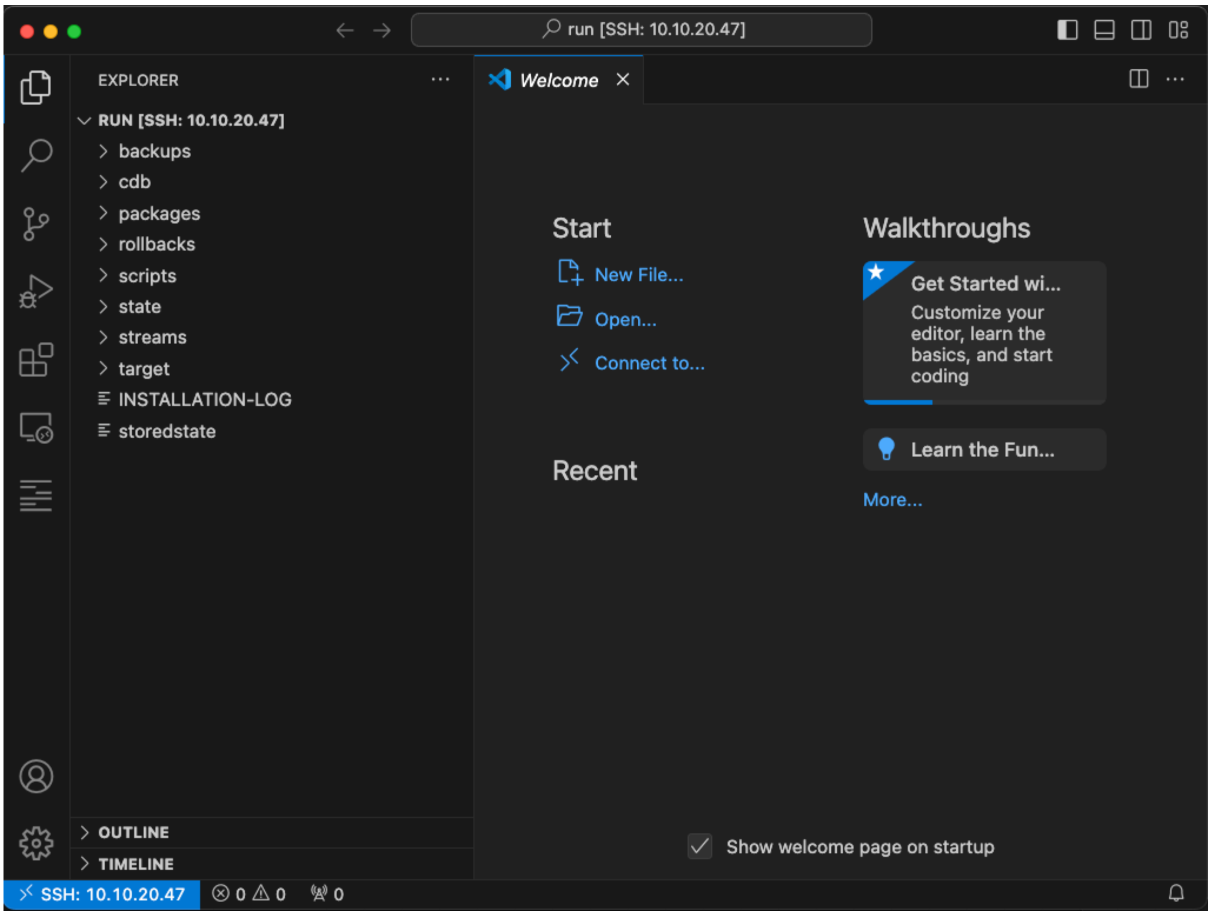
```
developer connected from 192.168.254.11 using ssh on nso-
sandbox1.devnet
developer@ncs> switch cli
developer@ncs#
```

1.1 – Create the Nano Service skeleton package

Navigate to the packages folder under run.

```
developer@nso-sandbox1: > cd /nso/run/packages/
developer@nso-sandbox1:packages > ncs-make-package --nano-service-
skeleton python-and-template NTP
```

Use the Visual Studio Code (in your machine) with the extension "Remote – SSH" to facilitate the editing of the service files.



Login into NSO and run the "packages reload" command.

```
developer@nso-sandbox1:run > ncs_cli -Cu admin

User admin last logged in 2024-11-11T15:15:56.185475+00:00, to nso-
sandbox1, from 192.168.254.11 using cli-ssh
admin connected from 192.168.254.11 using ssh on nso-sandbox1.devnet
admin@ncs# packages reload
reload-result {
    package NTP
    result false
    info NTP-template.xml:4 Unknown namespace: 'NTP:vm-configured'
}
reload-result {
    package cisco-asa-cli-6.18
    result true
}
…
```

We can see the package shows up in the “packages reload” result (ignore the result message).

Its now time to edit our service.

1.2 – Nano Service Definition

We will create a service with 3 stages, to navigate between stages you will a variable (leaf) will need to be affected with the True state.

In each stage, we will apply a different set of device commands.

1st stage we will apply the NTP Server configurations
“ntp server X.X.X.X minpoll X maxpoll X”

2nd stage we will apply the NTP Peer configurations
“ntp peer X.X.X.X”

3rd stage we will apply the NTP MAX Association configurations
“ntp max-associations X”

Each stage will be represented on its own XML Template.

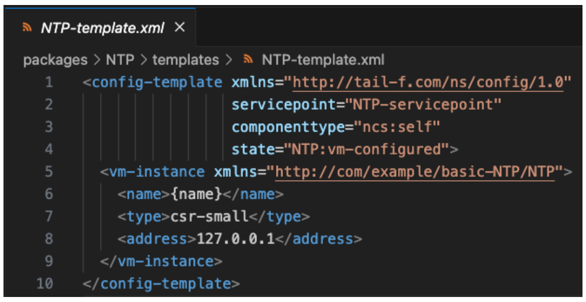
1.3 – Service Templates Definition

1st stage template creation

```
admin@ncs# conf
admin@ncs(config)# devices device core-rtr01
admin@ncs(config-config)# ntp server 1.1.1.1 minpoll 8 maxpoll 12
admin@ncs(config-ntp)# commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
               <device>
                 <name>core-rtr01</name>
                 <config>
                   <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
                     <server>
```

```
                       <server-list>
                         <name>1.1.1.1</name>
                         <minpoll>8</minpoll>
                         <maxpoll>12</maxpoll>
                       </server-list>
                     </server>
                   </ntp>
                 </config>
               </device>
             </devices>
    }
}
```

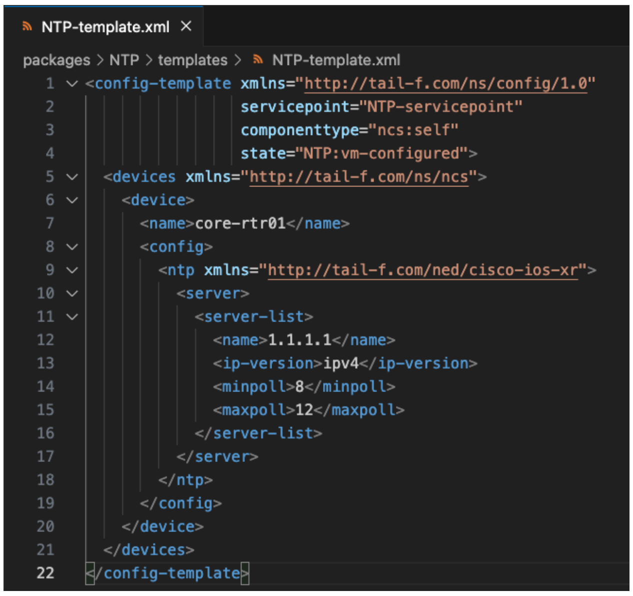
Replace the **<vm-instance>** block present in the skeleton template **(NTP-template.xml**) with the **<devices>** block you got from the CLI.



NTP-template.xml ×

packages > NTP > templates > NTP-template.xml

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="NTP-servicepoint"
                 componenttype="ncs:self"
                 state="NTP:vm-configured">
  <vm-instance xmlns="http://com/example/basic-NTP/NTP">
    <name>{name}</name>
    <type>csr-small</type>
    <address>127.0.0.1</address>
  </vm-instance>
</config-template>
```



NTP-template.xml ×

packages > NTP > templates > NTP-template.xml

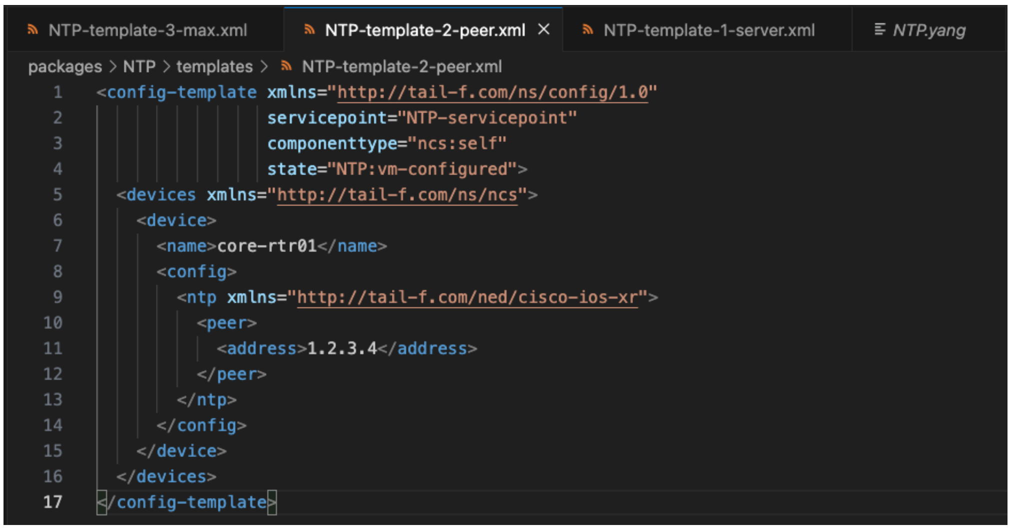
```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="NTP-servicepoint"
                 componenttype="ncs:self"
                 state="NTP:vm-configured">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>core-rtr01</name>
      <config>
        <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <server>
            <server-list>
              <name>1.1.1.1</name>
              <ip-version>ipv4</ip-version>
              <minpoll>8</minpoll>
              <maxpoll>12</maxpoll>
            </server-list>
          </server>
        </ntp>
      </config>
    </device>
  </devices>
</config-template>
```

Let’s create 2 more templates. Copy and paste the NTP-template.xml twice.
Rename the templates to :

NTP-Template-1-server.xml (original)
NTP-template-2-peer.xml (copy)
NTP-template-3-max.xml (copy)

We will now create the second template (**NTP-template-2-peer.xml**) content. Go back to NSO cli and get the XML equivalent of the peer configuration.

```
admin@ncs# conf
Entering configuration mode terminal
admin@ncs(config)# devices device core-rtr01 config
admin@ncs(config-config)# ntp peer 1.2.3.4
admin@ncs(config-ntp)# commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
               <device>
                 <name>core-rtr01</name>
                 <config>
                   <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
                     <peer>
                       <address>1.2.3.4</address>
                     </peer>
                   </ntp>
                 </config>
               </device>
             </devices>
    }
}
```



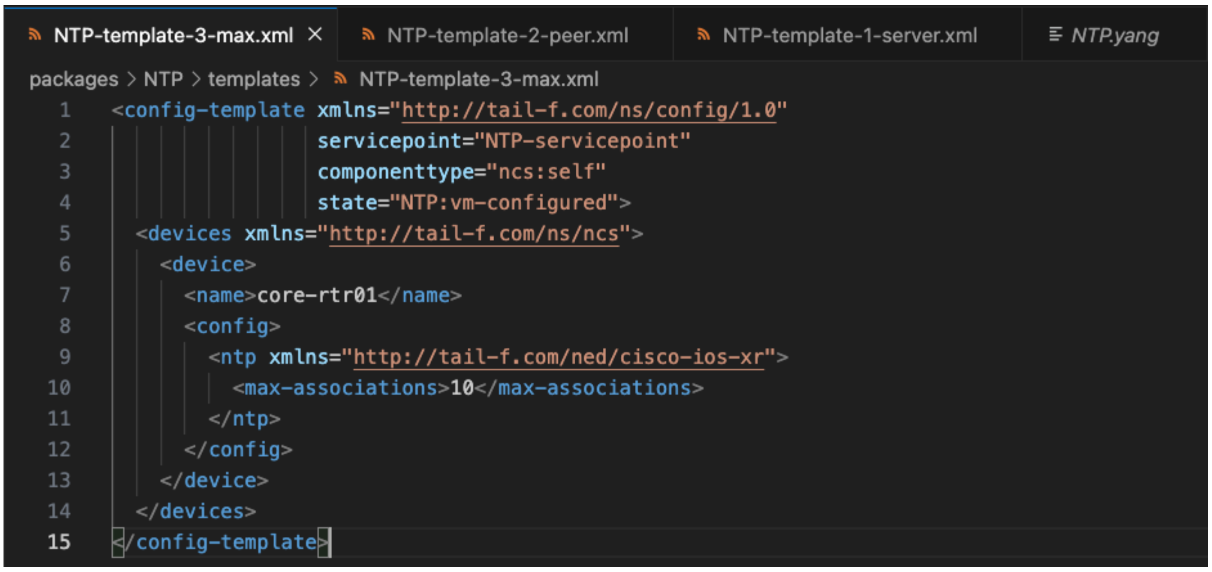
NTP-template-3-max.xml | NTP-template-2-peer.xml | NTP-template-1-server.xml | NTP.yang

packages > NTP > templates > NTP-template-2-peer.xml

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="NTP-servicepoint"
                 componenttype="ncs:self"
                 state="NTP:vm-configured">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>core-rtr01</name>
      <config>
        <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <peer>
            <address>1.2.3.4</address>
          </peer>
        </ntp>
      </config>
    </device>
  </devices>
</config-template>
```

Do the same for the max association’s configuration (**NTP-template-3-max.xml**)

```
admin@ncs(config)# devices device core-rtr01 config
admin@ncs(config-config)# ntp max-associations 10
admin@ncs(config-ntp)# commit dry-run outformat xml
result-xml {
    local-node {
        data <devices xmlns="http://tail-f.com/ns/ncs">
                 <device>
                   <name>core-rtr01</name>
                   <config>
                     <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
                       <max-associations>10</max-associations>
                     </ntp>
                   </config>
                 </device>
               </devices>
    }
}
```


NTP-template-3-max.xml
NTP-template-2-peer.xml
NTP-template-1-server.xml
NTP.yang
packages > NTP > templates > NTP-template-3-max.xml

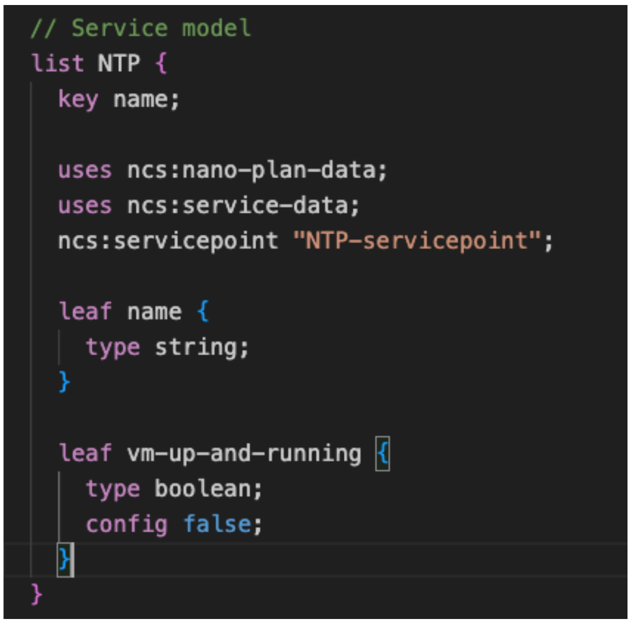
```xml
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="NTP-servicepoint"
                 componenttype="ncs:self"
                 state="NTP:vm-configured">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>core-rtr01</name>
      <config>
        <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <max-associations>10</max-associations>
        </ntp>
      </config>
    </device>
  </devices>
</config-template>
```

The base for our templates is ready.

1.4 – Service Modeling Definition

Let’s start the works in our YANG model.

Look for the “Service model” words in the “NTP.yang” file.

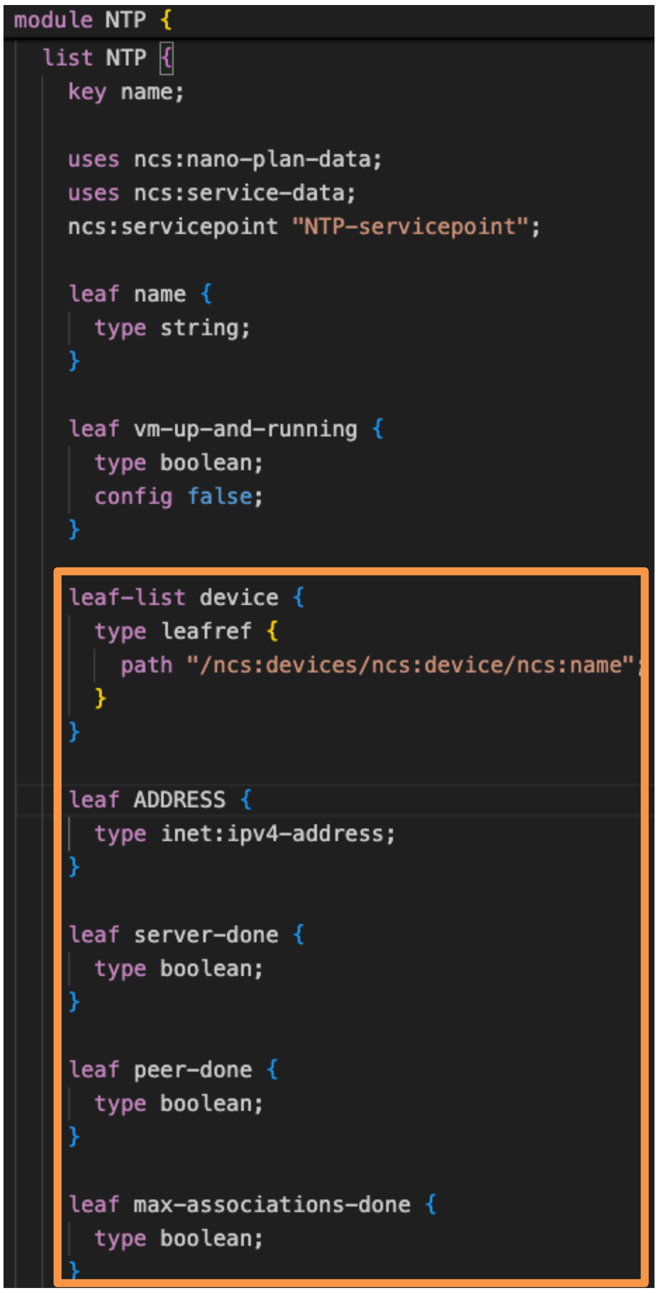


```
// Service model
list NTP {
  key name;

  uses ncs:nano-plan-data;
  uses ncs:service-data;
  ncs:servicepoint "NTP-servicepoint";

  leaf name {
    type string;
  }

  leaf vm-up-and-running {
    type boolean;
    config false;
  }
}
```

Inside this list we will create our service inputs and stage vars.

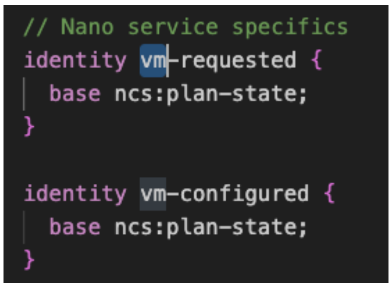
Create the
leaf-list **device** (will pull the device list on NSO)
leaf **ADDRESS** (ipv4 address for the ntp server)
leaf **server-done** (boolean)

leaf **peer-done** (boolean)
leaf **max-associations-done** (boolean)



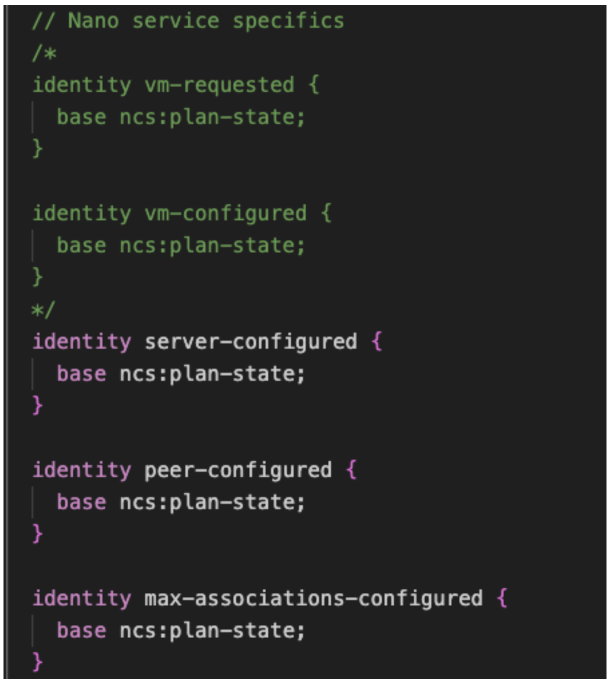
```
module NTP {
  list NTP {
    key name;

    uses ncs:nano-plan-data;
    uses ncs:service-data;
    ncs:servicepoint "NTP-servicepoint";

    leaf name {
      type string;
    }

    leaf vm-up-and-running {
      type boolean;
      config false;
    }

    leaf-list device {
      type leafref {
        path "/ncs:devices/ncs:device/ncs:name"
      }
    }

    leaf ADDRESS {
      type inet:ipv4-address;
    }

    leaf server-done {
      type boolean;
    }

    leaf peer-done {
      type boolean;
    }

    leaf max-associations-done {
      type boolean;
    }
```

Now in the Model, look for the words “Nano service specifics” and let’s define the identities.



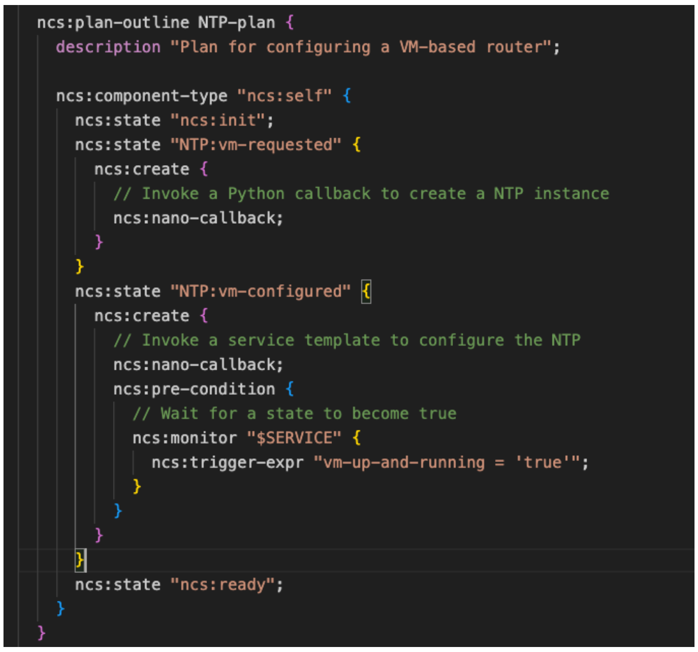
```
// Nano service specifics
identity vm-requested {
  base ncs:plan-state;
}

identity vm-configured {
  base ncs:plan-state;
}
```

Define the identities, "**server-configured**", "**peer-configured**", "**max-associations-configured**".



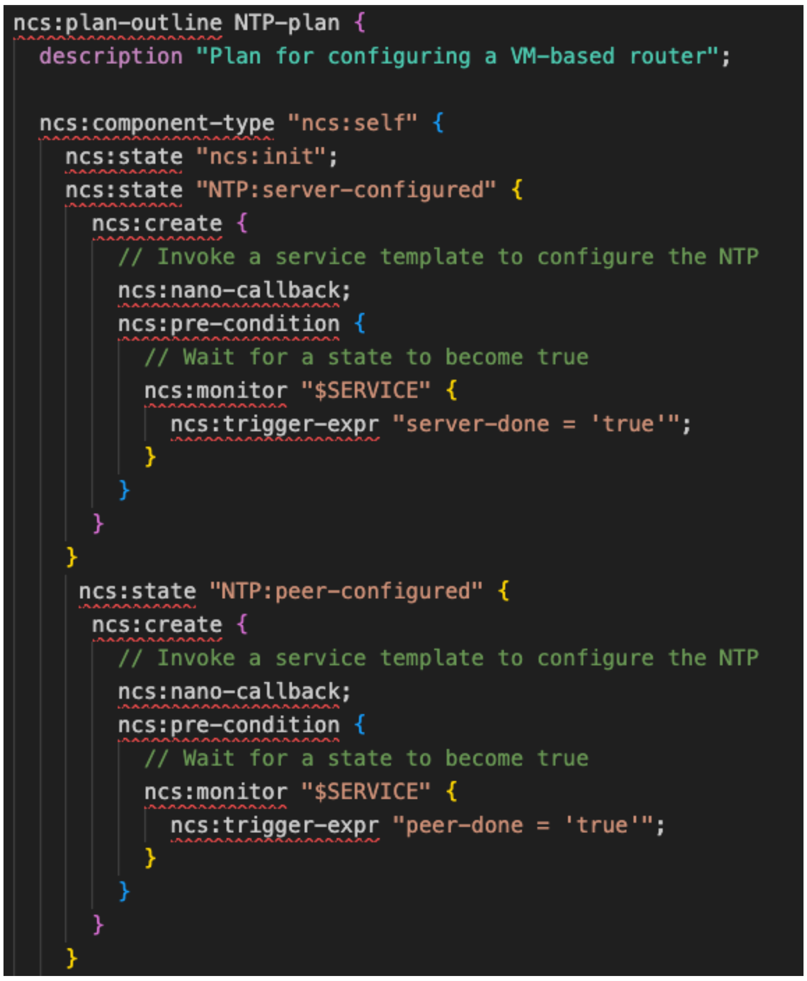
```
// Nano service specifics
/*
identity vm-requested {
  base ncs:plan-state;
}

identity vm-configured {
  base ncs:plan-state;
}
*/
identity server-configured {
  base ncs:plan-state;
}

identity peer-configured {
  base ncs:plan-state;
}

identity max-associations-configured {
  base ncs:plan-state;
}
```

Next step, its to edit the plan, and define what we will do in each stage.

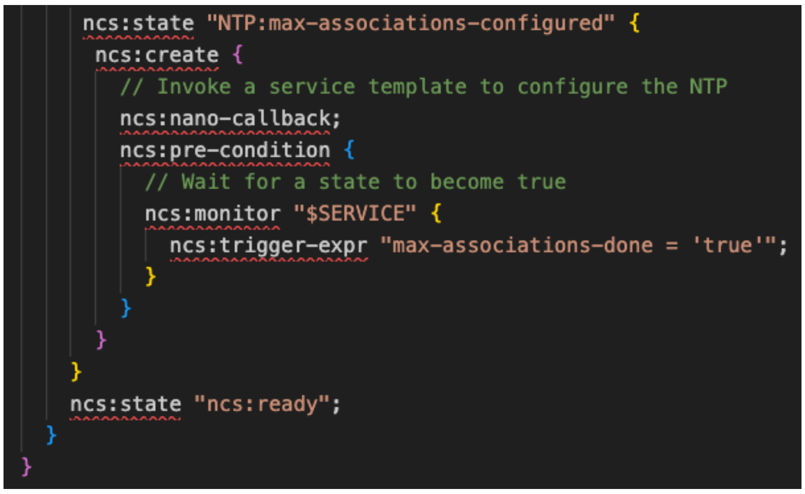


```
ncs:plan-outline NTP-plan {
  description "Plan for configuring a VM-based router";

  ncs:component-type "ncs:self" {
    ncs:state "ncs:init";
    ncs:state "NTP:vm-requested" {
      ncs:create {
        // Invoke a Python callback to create a NTP instance
        ncs:nano-callback;
      }
    }
    ncs:state "NTP:vm-configured" {
      ncs:create {
        // Invoke a service template to configure the NTP
        ncs:nano-callback;
        ncs:pre-condition {
          // Wait for a state to become true
          ncs:monitor "$SERVICE" {
            ncs:trigger-expr "vm-up-and-running = 'true'";
          }
        }
      }
    }
    ncs:state "ncs:ready";
  }
}
```

In each stage we will call the python logic and apply the template associated with each stage.



```
ncs:plan-outline NTP-plan {
  description "Plan for configuring a VM-based router";

  ncs:component-type "ncs:self" {
    ncs:state "ncs:init";
    ncs:state "NTP:server-configured" {
      ncs:create {
        // Invoke a service template to configure the NTP
        ncs:nano-callback;
        ncs:pre-condition {
          // Wait for a state to become true
          ncs:monitor "$SERVICE" {
            ncs:trigger-expr "server-done = 'true'";
          }
        }
      }
    }
     ncs:state "NTP:peer-configured" {
      ncs:create {
        // Invoke a service template to configure the NTP
        ncs:nano-callback;
        ncs:pre-condition {
          // Wait for a state to become true
          ncs:monitor "$SERVICE" {
            ncs:trigger-expr "peer-done = 'true'";
          }
        }
      }
    }
```



```
     ncs:state "NTP:max-associations-configured" {
      ncs:create {
        // Invoke a service template to configure the NTP
        ncs:nano-callback;
        ncs:pre-condition {
          // Wait for a state to become true
          ncs:monitor "$SERVICE" {
            ncs:trigger-expr "max-associations-done = 'true'";
          }
        }
      }
    }
    ncs:state "ncs:ready";
  }
}
```

This is how your “plan-outline” should look like.

The list “vm-instance” can be deleted.

Our service model is ready. Let’s compile it.

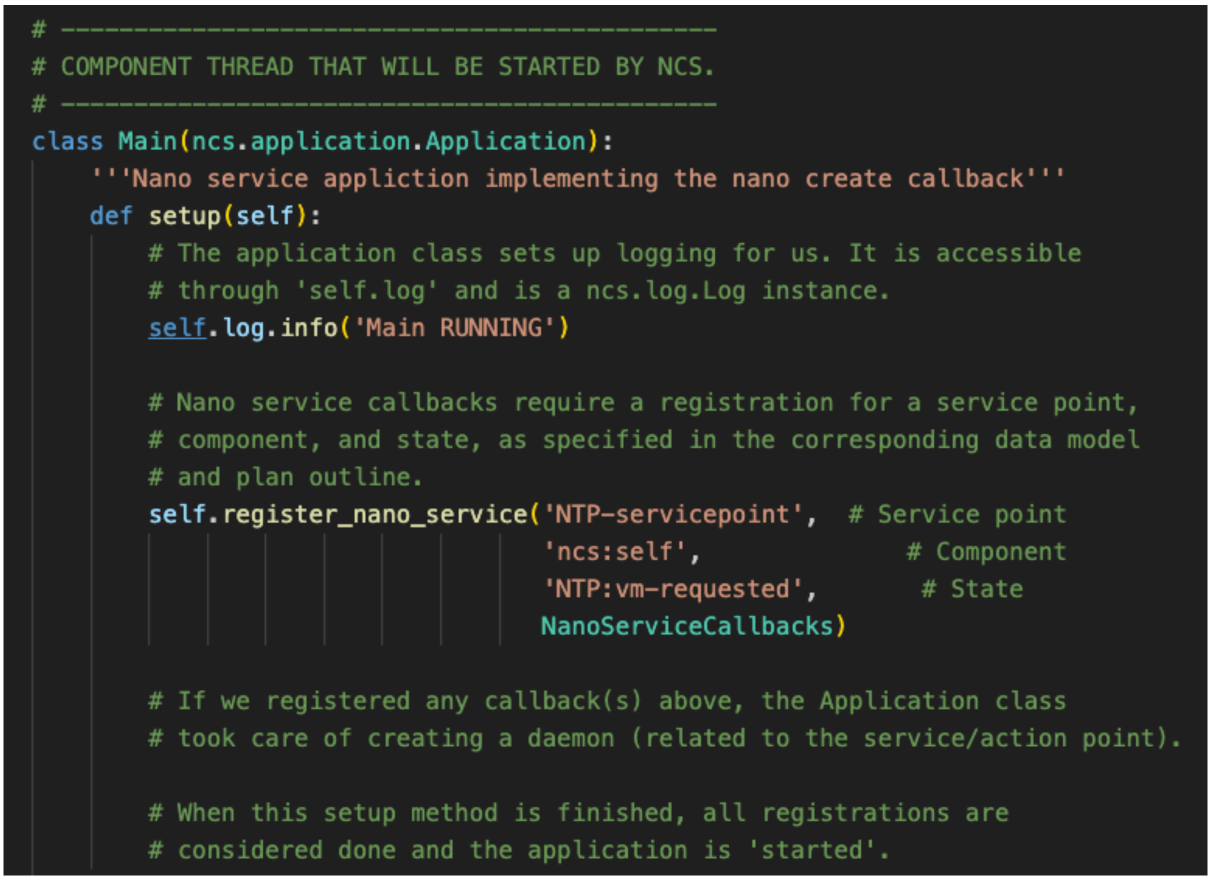
```
developer@nso-sandbox1:run > pwd
/nso/run
developer@nso-sandbox1:run > cd packages/
developer@nso-sandbox1:packages > cd NTP/
```

```
developer@nso-sandbox1:NTP > cd src/
developer@nso-sandbox1:src > make
mkdir -p ../load-dir
mkdir -p java/src//
/opt/ncs/current/bin/ncsc  `ls NTP-ann.yang  > /dev/null 2>&1 &&
echo "-a NTP-ann.yang"` \
              -c -o ../load-dir/NTP.fxs yang/NTP.yang
```

1.5 – Service Logic Definition

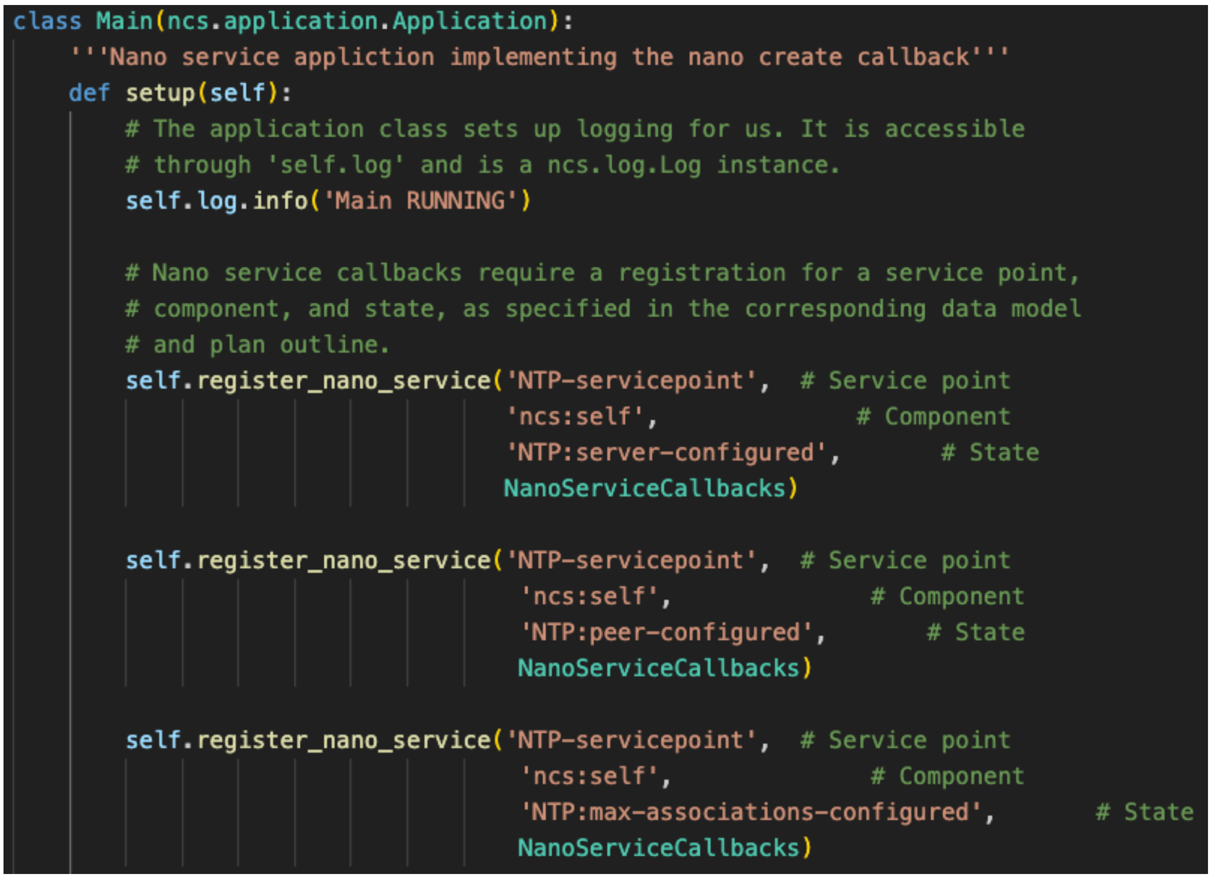
It's time for login definition.

In the "**class Main**" of the "**main.py**" you will find the place for the first changes.
that's where we will register our stages that need callbacks from python.



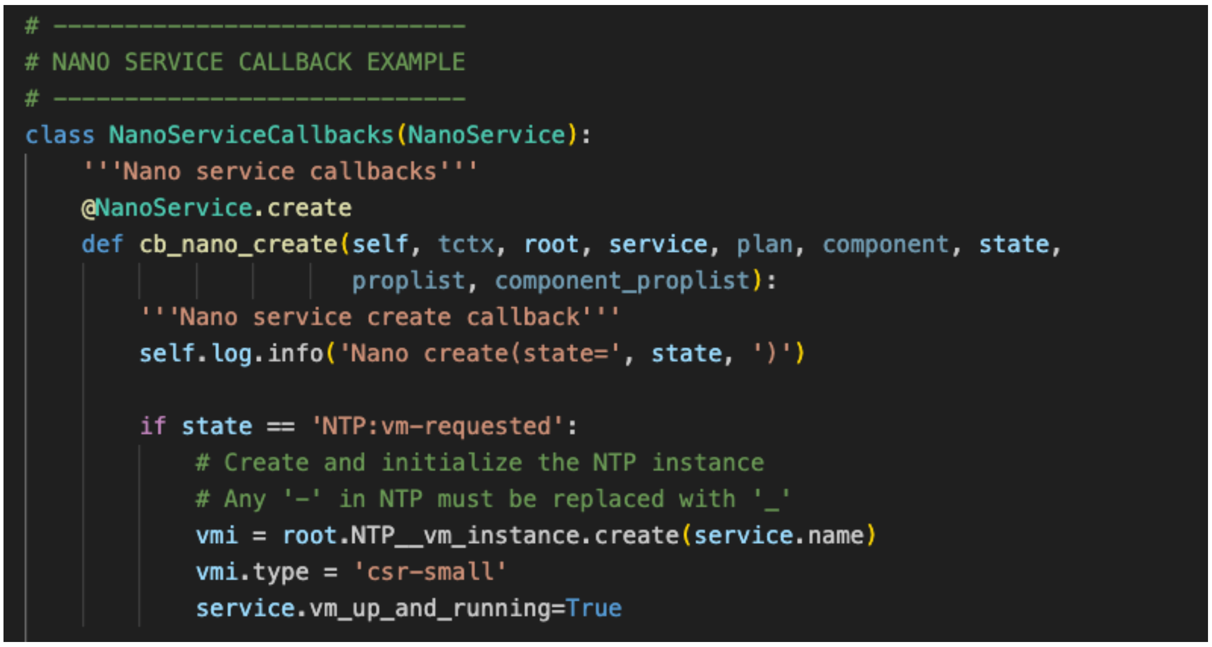
```
# ---------------------------------------------
# COMPONENT THREAD THAT WILL BE STARTED BY NCS.
# ---------------------------------------------
class Main(ncs.application.Application):
    '''Nano service appliction implementing the nano create callback'''
    def setup(self):
        # The application class sets up logging for us. It is accessible
        # through 'self.log' and is a ncs.log.Log instance.
        self.log.info('Main RUNNING')

        # Nano service callbacks require a registration for a service point,
        # component, and state, as specified in the corresponding data model
        # and plan outline.
        self.register_nano_service('NTP-servicepoint',  # Service point
                                   'ncs:self',            # Component
                                   'NTP:vm-requested',     # State
                                   NanoServiceCallbacks)

        # If we registered any callback(s) above, the Application class
        # took care of creating a daemon (related to the service/action point).

        # When this setup method is finished, all registrations are
        # considered done and the application is 'started'.
```

After the changes your "**setup**" function should look like this



```
class Main(ncs.application.Application):
    '''Nano service appliction implementing the nano create callback'''
    def setup(self):
        # The application class sets up logging for us. It is accessible
        # through 'self.log' and is a ncs.log.Log instance.
        self.log.info('Main RUNNING')

        # Nano service callbacks require a registration for a service point,
        # component, and state, as specified in the corresponding data model
        # and plan outline.
        self.register_nano_service('NTP-servicepoint',  # Service point
                                   'ncs:self',           # Component
                                   'NTP:server-configured',      # State
                                  NanoServiceCallbacks)

        self.register_nano_service('NTP-servicepoint',  # Service point
                                    'ncs:self',           # Component
                                    'NTP:peer-configured',       # State
                                    NanoServiceCallbacks)

        self.register_nano_service('NTP-servicepoint',  # Service point
                                    'ncs:self',           # Component
                                    'NTP:max-associations-configured',      # State
                                    NanoServiceCallbacks)
```

Next change is handling the callbacks in python. The current callback looks like this

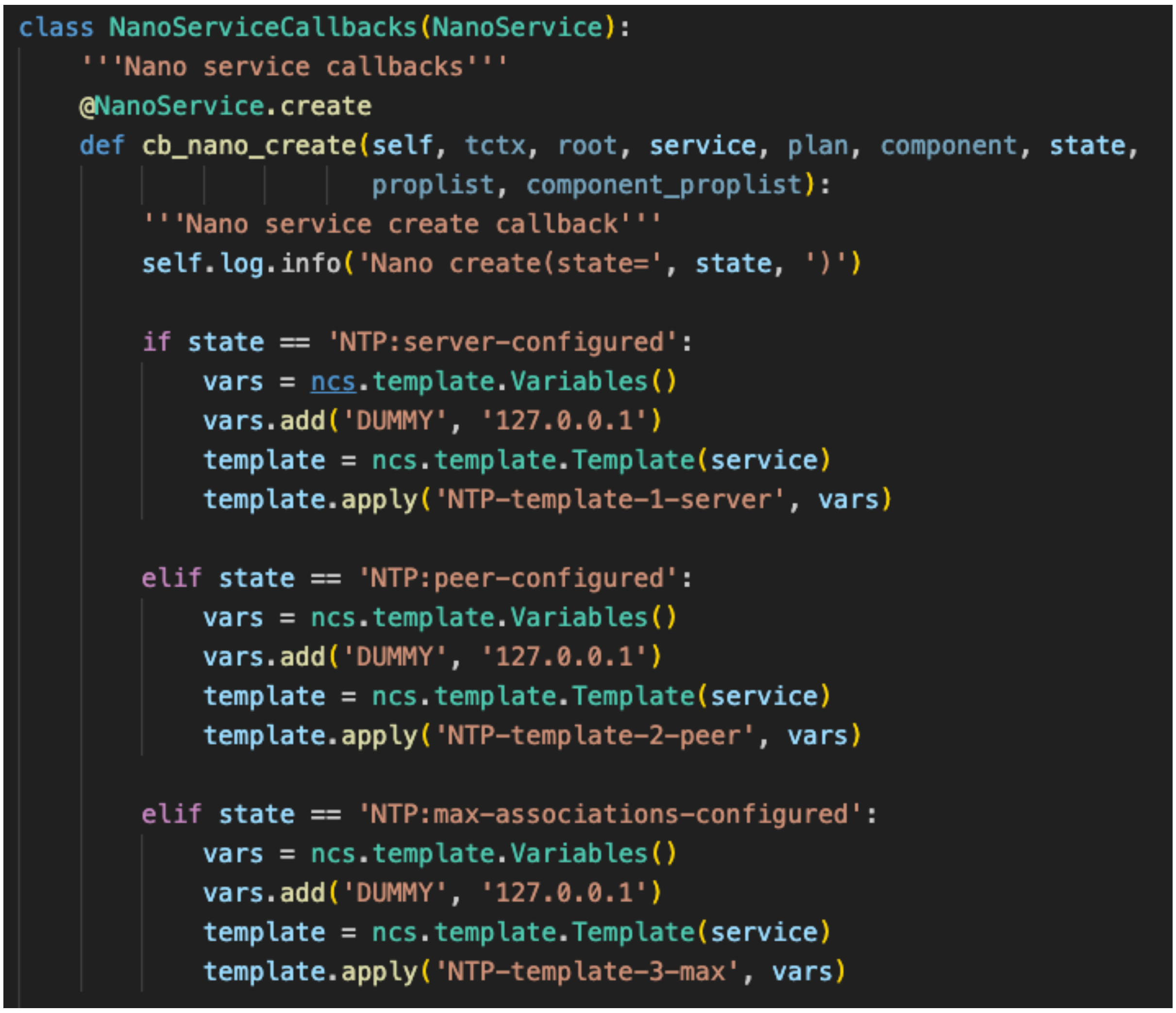


```
# ------------------------------
# NANO SERVICE CALLBACK EXAMPLE
# ------------------------------
class NanoServiceCallbacks(NanoService):
    '''Nano service callbacks'''
    @NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
                       proplist, component_proplist):
        '''Nano service create callback'''
        self.log.info('Nano create(state=', state, ')')

        if state == 'NTP:vm-requested':
            # Create and initialize the NTP instance
            # Any '-' in NTP must be replaced with '_'
            vmi = root.NTP__vm_instance.create(service.name)
            vmi.type = 'csr-small'
            service.vm_up_and_running=True
```

In this case the boolean variable is being affected automatically in the callback, making the nano service to move to the next stage. In our case we will force the Booleans to be changed manually.

In the "**cb_nano_create**" function, the "state" var is where we handle the stage the nano service is present.

Your callback handling should look like this



```python
class NanoServiceCallbacks(NanoService):
    '''Nano service callbacks'''
    @NanoService.create
    def cb_nano_create(self, tctx, root, service, plan, component, state,
                       proplist, component_proplist):
        '''Nano service create callback'''
        self.log.info('Nano create(state=', state, ')')

        if state == 'NTP:server-configured':
            vars = ncs.template.Variables()
            vars.add('DUMMY', '127.0.0.1')
            template = ncs.template.Template(service)
            template.apply('NTP-template-1-server', vars)

        elif state == 'NTP:peer-configured':
            vars = ncs.template.Variables()
            vars.add('DUMMY', '127.0.0.1')
            template = ncs.template.Template(service)
            template.apply('NTP-template-2-peer', vars)

        elif state == 'NTP:max-associations-configured':
            vars = ncs.template.Variables()
            vars.add('DUMMY', '127.0.0.1')
            template = ncs.template.Template(service)
            template.apply('NTP-template-3-max', vars)
```

“vars.add” is where you handle the variables you want to pass to the template via python.

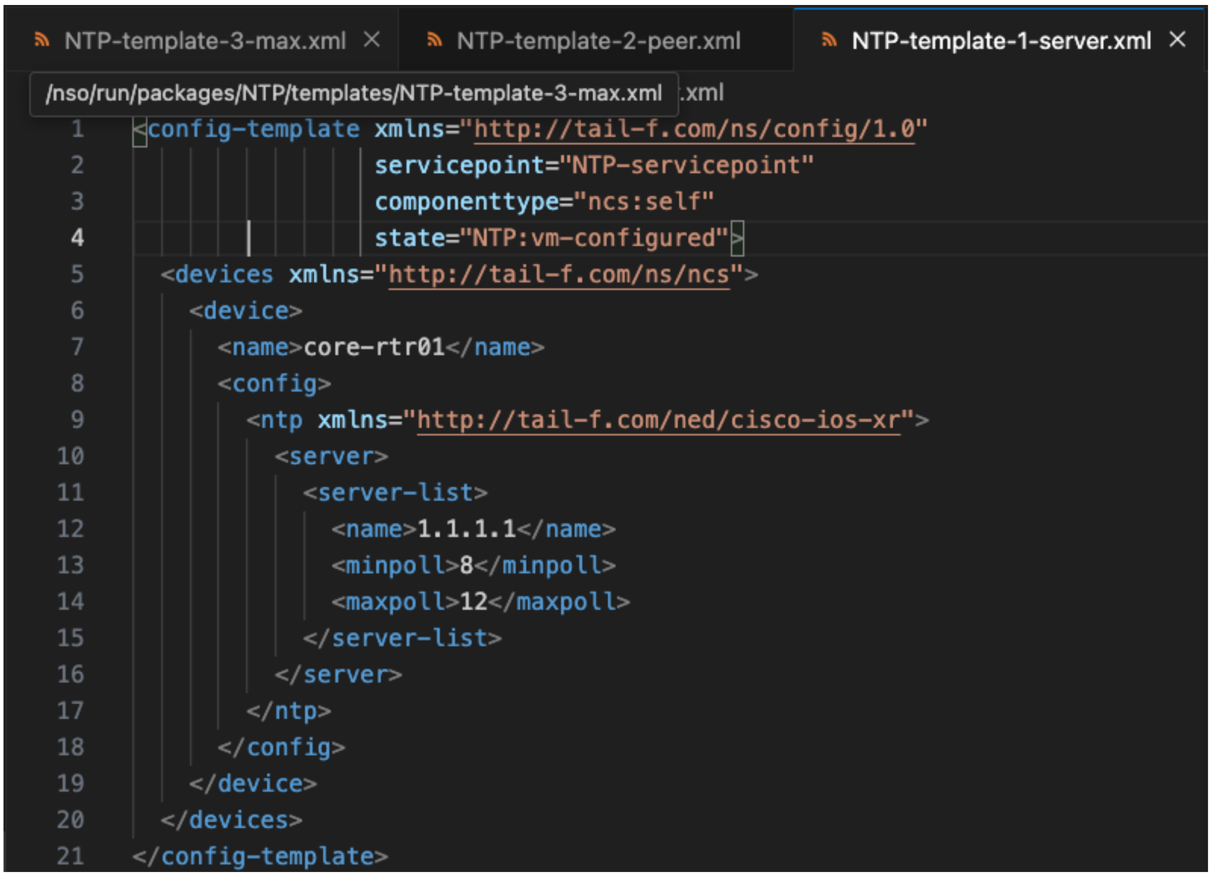
To remember
in the xml template you can insert data from
- python, using **{$python_var}**
or
- directly from the user input , yang **{/yang_var}**

Our python code is ready.

1.6 – Adjustments in the XML templates

Let’s now add reflect our pyhton/yang changes in the XML template.

In the “NTP-template-1-server.xml”

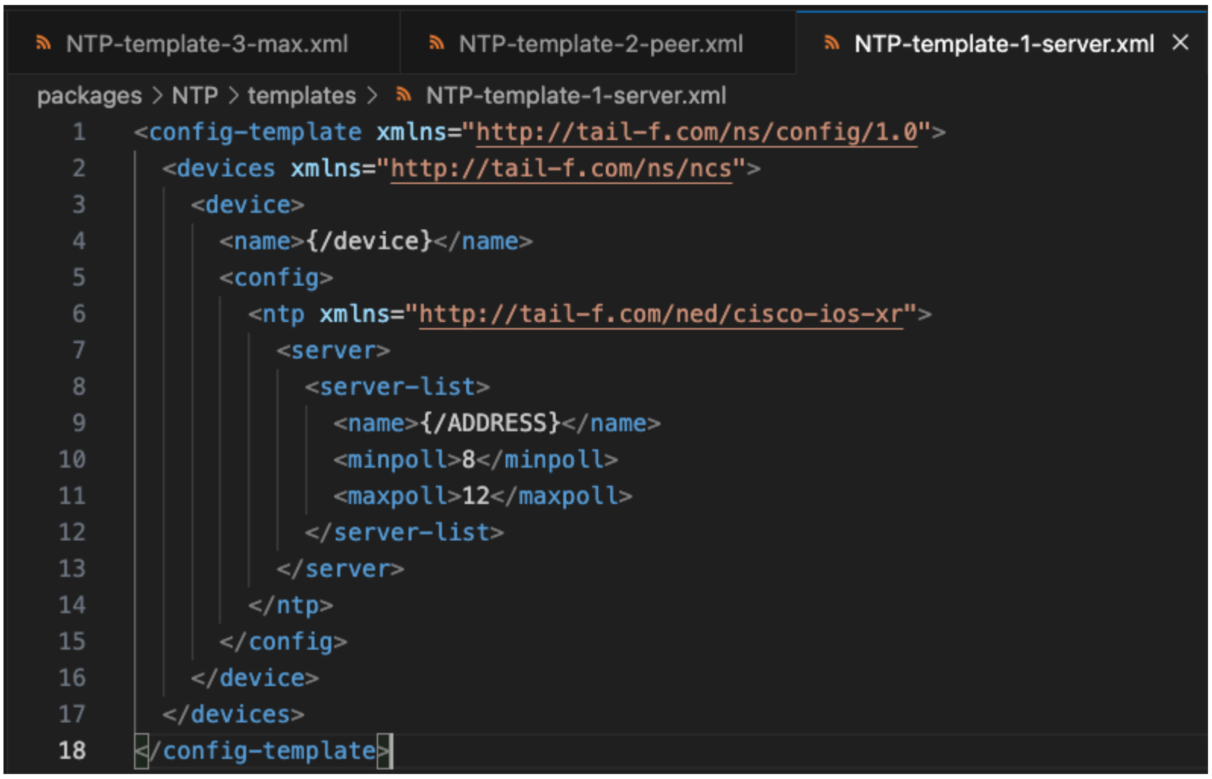

NTP-template-3-max.xml
NTP-template-2-peer.xml
NTP-template-1-server.xml
/nso/run/packages/NTP/templates/NTP-template-3-max.xml

```xml
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="NTP-servicepoint"
                 componenttype="ncs:self"
                 state="NTP:vm-configured">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>core-rtr01</name>
      <config>
        <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <server>
            <server-list>
              <name>1.1.1.1</name>
              <minpoll>8</minpoll>
              <maxpoll>12</maxpoll>
            </server-list>
          </server>
        </ntp>
      </config>
    </device>
  </devices>
</config-template>
```

We will change the device name from static to yang variable, as well, the server-list name.

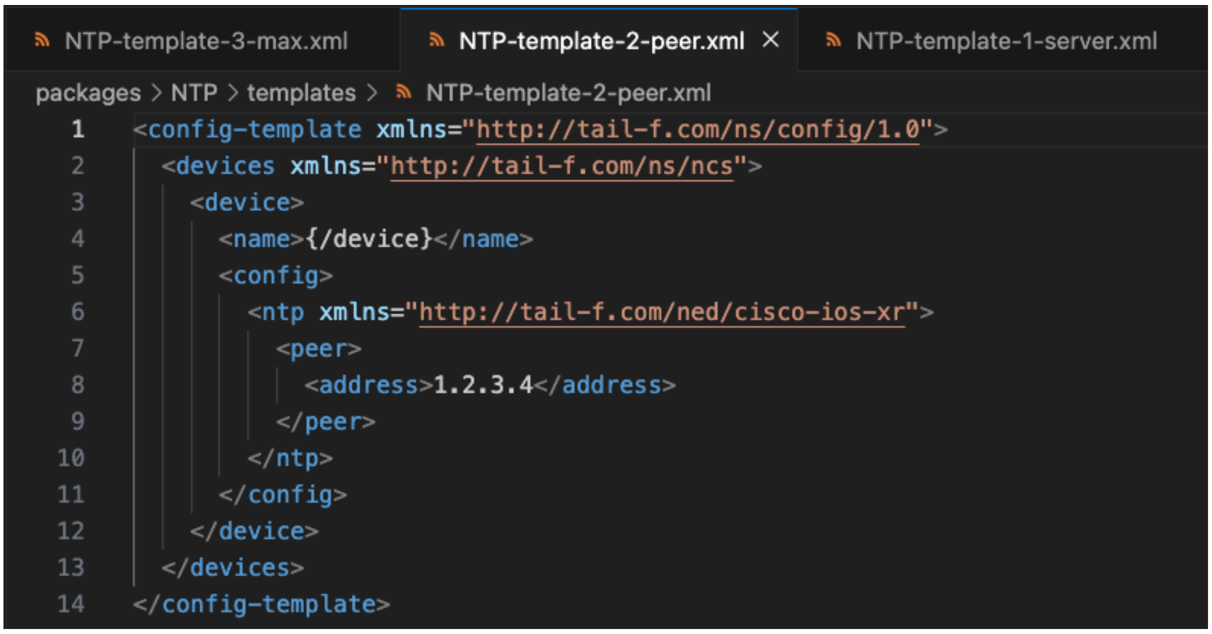
Because we will use the python callback to apply the template, the information related to the state, component and servicepoint can be deleted.

In the end it should look like this.


NTP-template-3-max.xml
NTP-template-2-peer.xml
NTP-template-1-server.xml
packages > NTP > templates > NTP-template-1-server.xml

```xml
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/device}</name>
      <config>
        <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <server>
            <server-list>
              <name>{/ADDRESS}</name>
              <minpoll>8</minpoll>
              <maxpoll>12</maxpoll>
            </server-list>
          </server>
        </ntp>
      </config>
    </device>
  </devices>
</config-template>
```

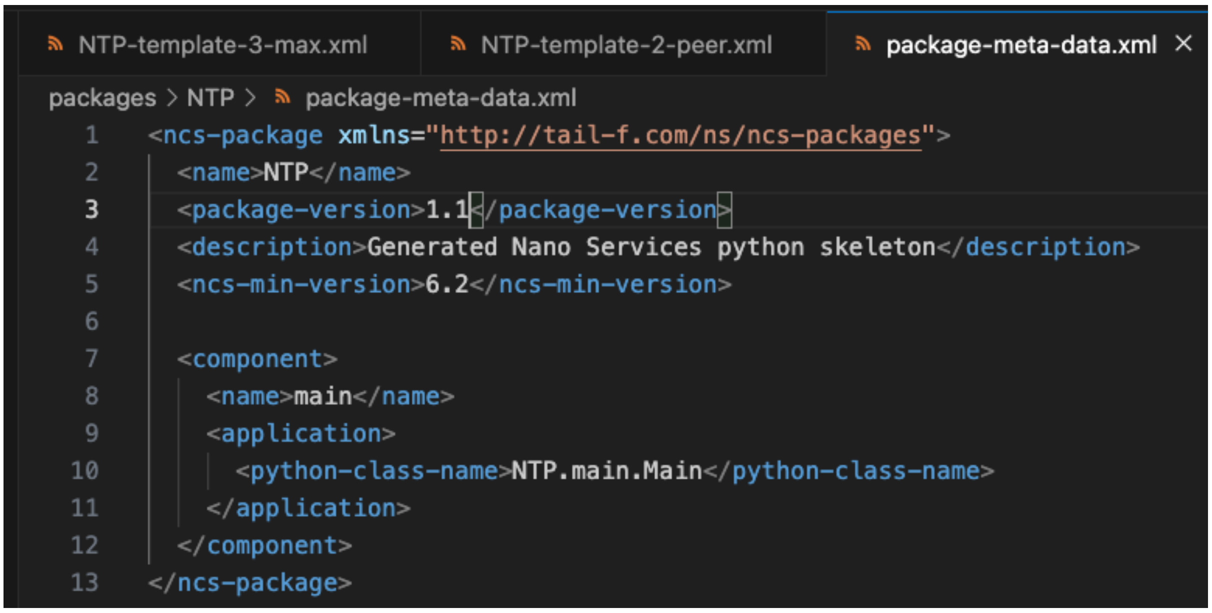
Proceed with the changes on the remaining templates (**NTP-template-2-peer.xml** and **NTP-template-3-max.xml**)


NTP-template-3-max.xml
NTP-template-2-peer.xml
NTP-template-1-server.xml
packages > NTP > templates > NTP-template-2-peer.xml

```xml
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/device}</name>
      <config>
        <ntp xmlns="http://tail-f.com/ned/cisco-ios-xr">
          <peer>
            <address>1.2.3.4</address>
          </peer>
        </ntp>
      </config>
    </device>
  </devices>
</config-template>
```

All the main changes are performed.

Let's now update the service version to 1.1. Do this by opening the **package-meta-data.xml** file in the service root directory **/nso/run/packages/NTP**



```xml
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>NTP</name>
  <package-version>1.1</package-version>
  <description>Generated Nano Services python skeleton</description>
  <ncs-min-version>6.2</ncs-min-version>

  <component>
    <name>main</name>
    <application>
      <python-class-name>NTP.main.Main</python-class-name>
    </application>
  </component>
</ncs-package>
```

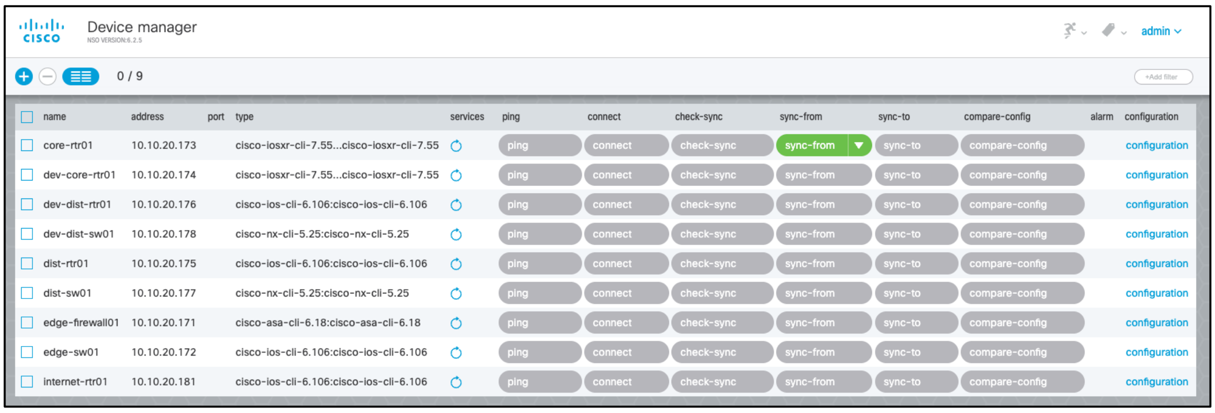
1.7 – Service usage

Reload the package and apply the changes.

```
developer@nso-sandbox1:NTP > ncs_cli -Cu admin

User admin last logged in 2024-11-11T16:18:11.993076+00:00, to nso-
sandbox1, from 192.168.254.11 using cli-ssh
admin connected from 192.168.254.11 using ssh on nso-sandbox1.devnet
admin@ncs# packages reload

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has
completed.
>>> System upgrade has completed successfully.
reload-result {
    package NTP
    result true
}
reload-result {
    package cisco-asa-cli-6.18
    result true
}
```

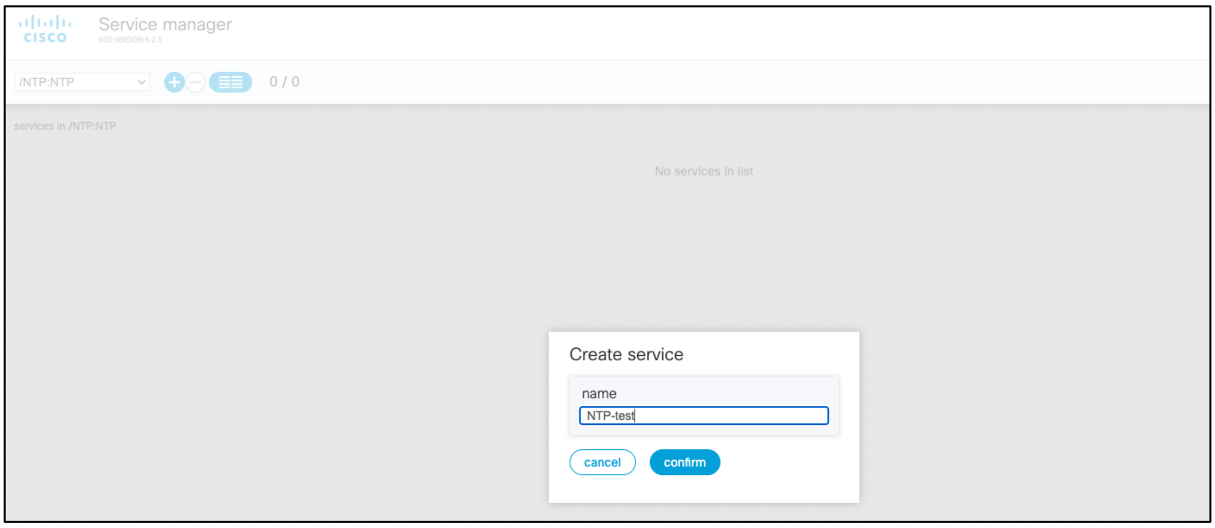
Be sure the devices you want to apply the service are in sync with NSO.

In the WebUI go to the Device Manager and click on the “sync-from” button.

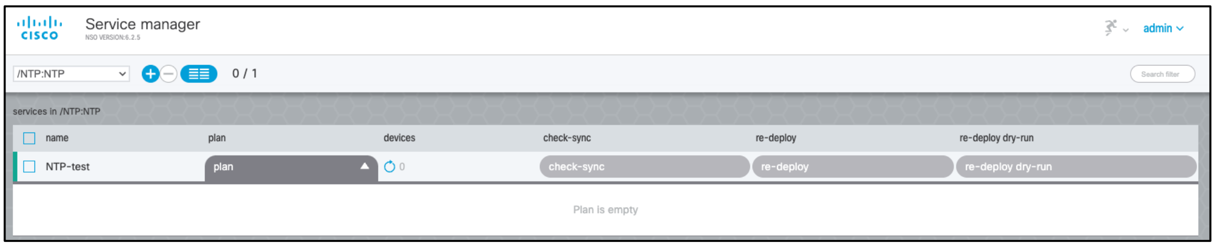


name	address	port	type	services	ping	connect	check-sync	sync-from	sync-to	compare-config	alarm	configuration
core-rtr01	10.10.20.173		cisco-iosxr-cli-7.55...cisco-iosxr-cli-7.55		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
dev-core-rtr01	10.10.20.174		cisco-iosxr-cli-7.55...cisco-iosxr-cli-7.55		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
dev-dist-rtr01	10.10.20.176		cisco-ios-cli-6.106:cisco-ios-cli-6.106		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
dev-dist-sw01	10.10.20.178		cisco-nx-cli-5.25:cisco-nx-cli-5.25		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
dist-rtr01	10.10.20.175		cisco-ios-cli-6.106:cisco-ios-cli-6.106		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
dist-sw01	10.10.20.177		cisco-nx-cli-5.25:cisco-nx-cli-5.25		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
edge-firewall01	10.10.20.171		cisco-asa-cli-6.18:cisco-asa-cli-6.18		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
edge-sw01	10.10.20.172		cisco-ios-cli-6.106:cisco-ios-cli-6.106		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration
internet-rtr01	10.10.20.181		cisco-ios-cli-6.106:cisco-ios-cli-6.106		ping	connect	check-sync	sync-from	sync-to	compare-config		configuration

After the sync, go to the Service Manager and Create the NTP service instance.



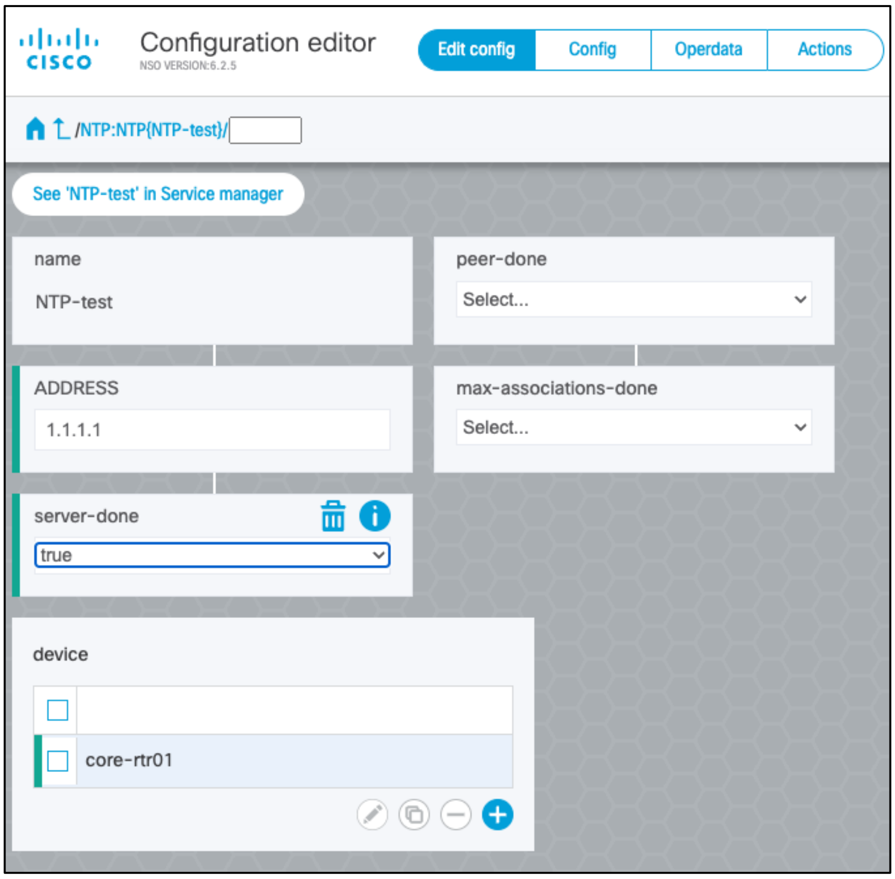
You can notice the service instance already shows a plan. At this point remains empty because no device is selected.



name	plan	devices	check-sync	re-deploy	re-deploy dry-run
NTP-test	plan	0	check-sync	re-deploy	re-deploy dry-run

Plan is empty

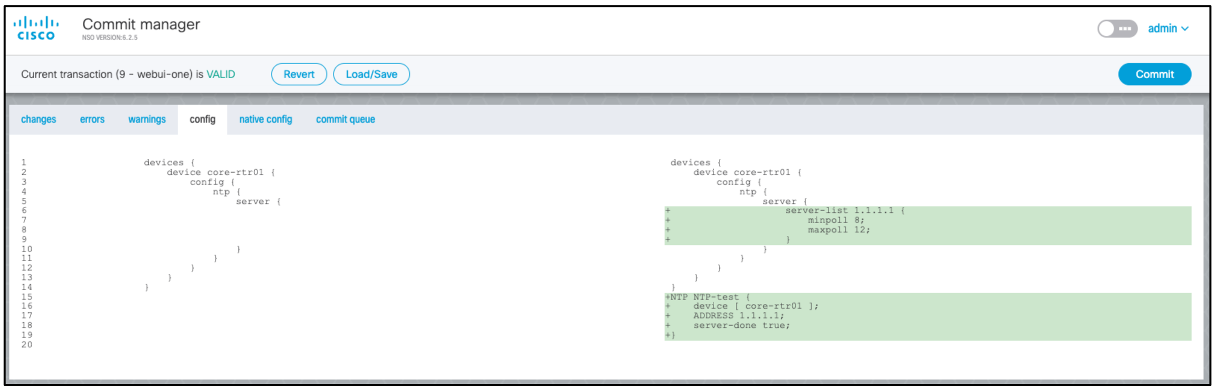
Let’s apply the first service configurations, setting up the “**server-done**” variable as true.


Configuration editor
NSO VERSION:6.2.5

/NTP:NTP{NTP-test}/

name: NTP-test
peer-done: Select...
ADDRESS: 1.1.1.1
max-associations-done: Select...
server-done: true
device: core-rtr01

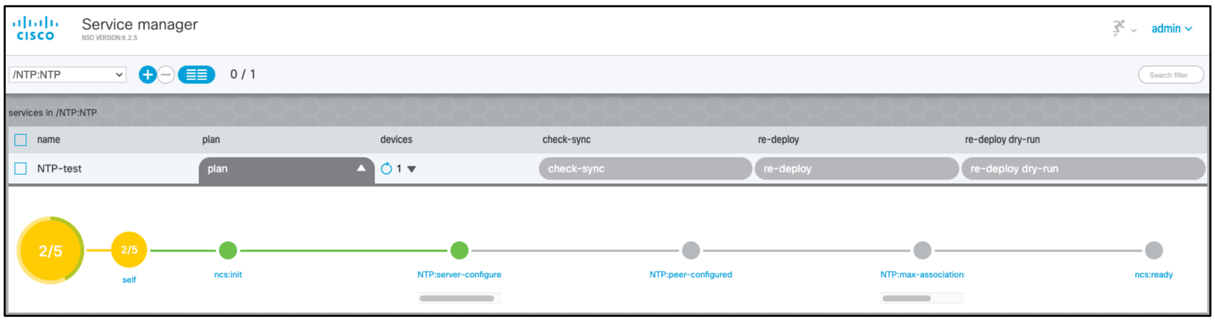
Going to the commit manager, we can notice that only the configurations defined in the first template are applied.


Commit manager
NSO VERSION:6.2.5

Current transaction (9 - webui-one) is VALID

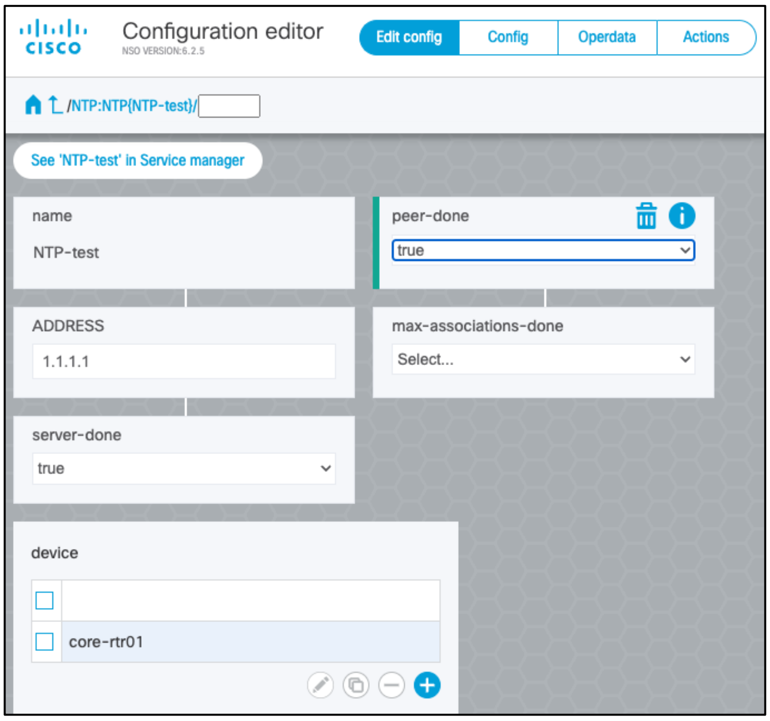
```
devices {
    device core-rtr01 {
        config {
            ntp {
                server {
+                   server-list 1.1.1.1 {
+                       minpoll 8;
+                       maxpoll 12;
+                   }
                }
            }
        }
    }
}
+NTP NTP-test {
+    device [ core-rtr01 ];
+    ADDRESS 1.1.1.1;
+    server-done true;
+}
```

Going back to our plan, we can already see some progress.



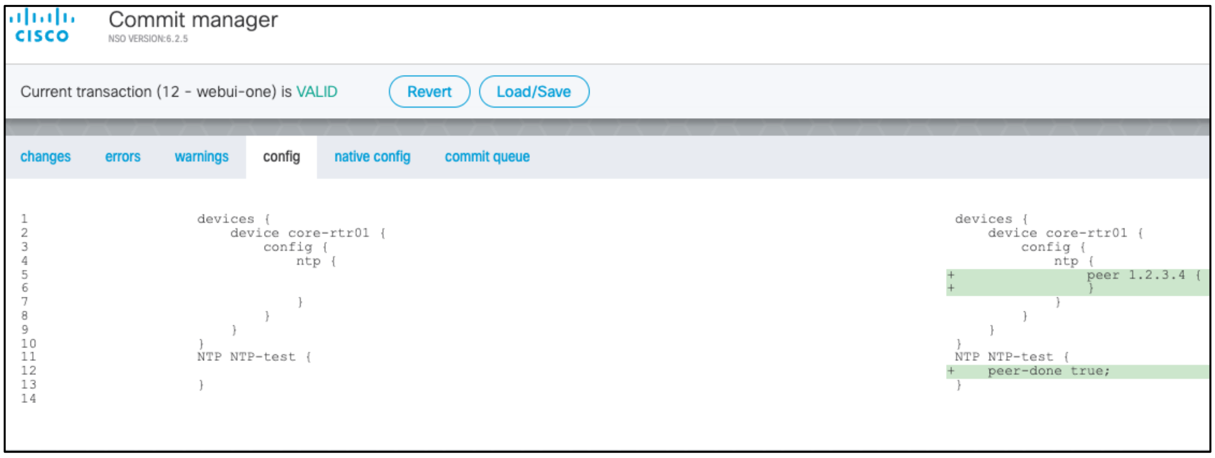
name	plan	devices	check-sync	re-deploy	re-deploy dry-run
NTP-test	plan	1	check-sync	re-deploy	re-deploy dry-run

Let’s now advance to the “peer-configured” stage.



name: NTP-test
peer-done: true
ADDRESS: 1.1.1.1
max-associations-done: Select...
server-done: true
device: core-rtr01

In the commit manager we can see the configuration changes from the second template.

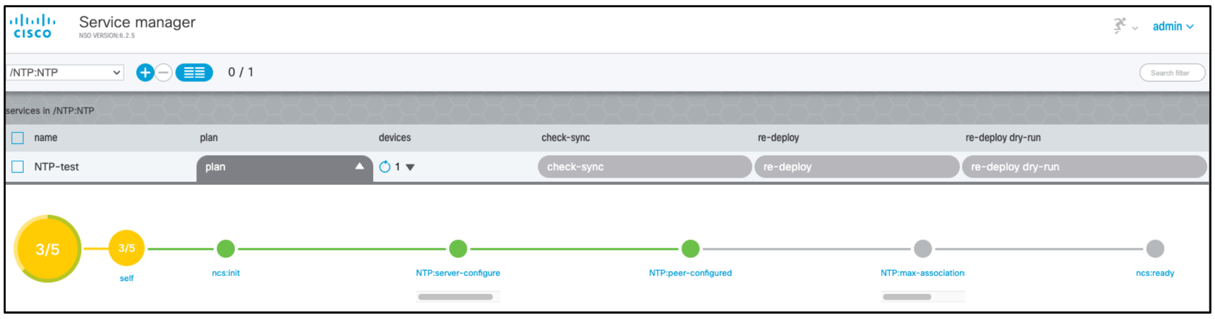


Current transaction (12 - webui-one) is VALID

```
devices {
    device core-rtr01 {
        config {
            ntp {


            }
        }
    }
}
NTP NTP-test {

}
```

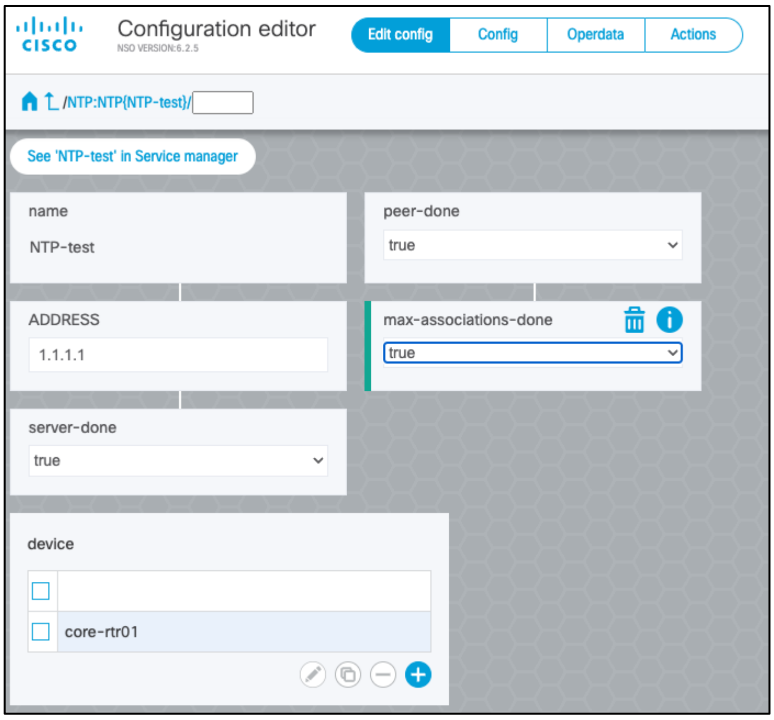
```
 devices {
     device core-rtr01 {
         config {
             ntp {
+                peer 1.2.3.4 {
+                }
             }
         }
     }
 }
 NTP NTP-test {
+    peer-done true;
 }
```

After the commit we can see the plan progress.



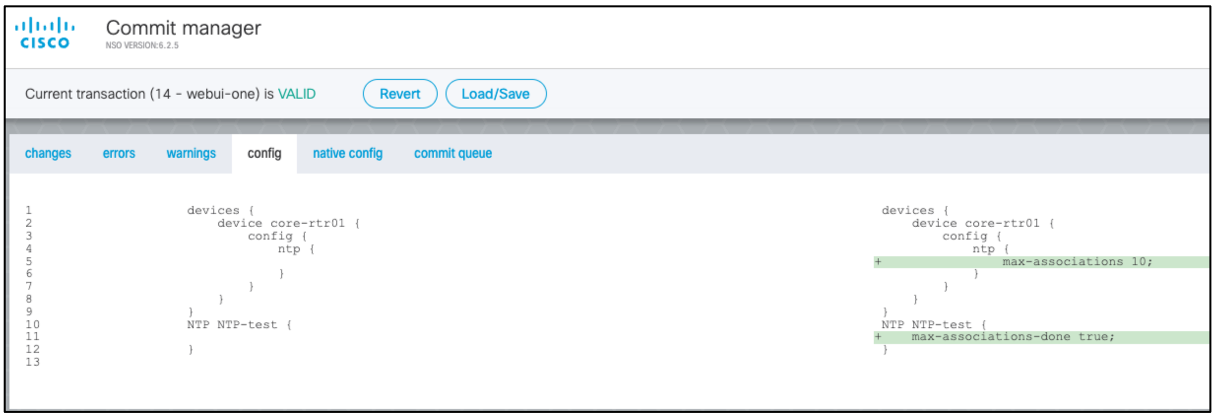
	name	plan	devices	check-sync	re-deploy	re-deploy dry-run
	NTP-test	plan	1	check-sync	re-deploy	re-deploy dry-run

By setting up the last variable “max-associations-done” as true we finish the service application.



name: NTP-test
peer-done: true
ADDRESS: 1.1.1.1
max-associations-done: true
server-done: true
device: core-rtr01

In the commit manager we will find the final configuration.

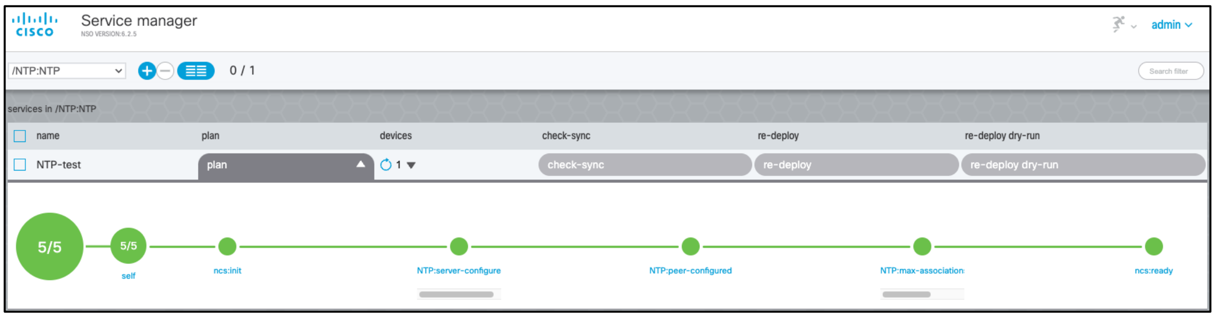


Current transaction (14 - webui-one) is VALID

```
devices {
    device core-rtr01 {
        config {
            ntp {

            }
        }
    }
}
NTP NTP-test {

}
```

```
 devices {
     device core-rtr01 {
         config {
             ntp {
+                max-associations 10;
             }
         }
     }
 }
 NTP NTP-test {
+    max-associations-done true;
 }
```

After we hit the commit button we can see the plan is now finished as the service arrived in the ncs:ready state



Thank you !

For questions related to this guide, jomira@cisco.com