



The bridge to possible

Empowering Network Automation

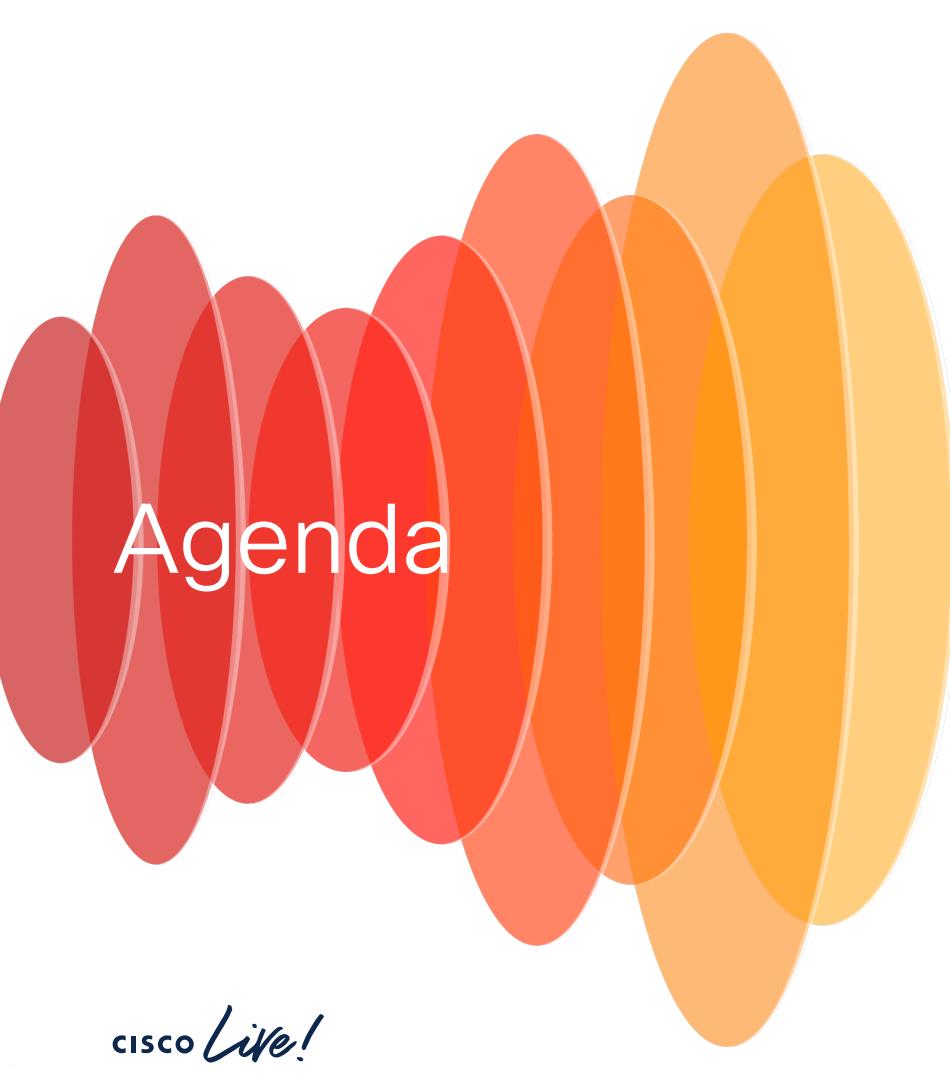
Practice NSO with Python

Jesus Illescas, Developer Advocate
@vincivero
DEVWKS-2551

cisco *Live!*

#CiscoLive





Agenda

- Core Concepts
- Python API
- Playground time
- Conclusion

Hi, I'm Jesus

I'm passionate about
Network Automation,
DevOps &
SW Engineering.

<https://netcode.rocks/>

<https://github.com/jillesca>

<https://twitter.com/vincivero>

<https://www.linkedin.com/in/jesusillesscas>

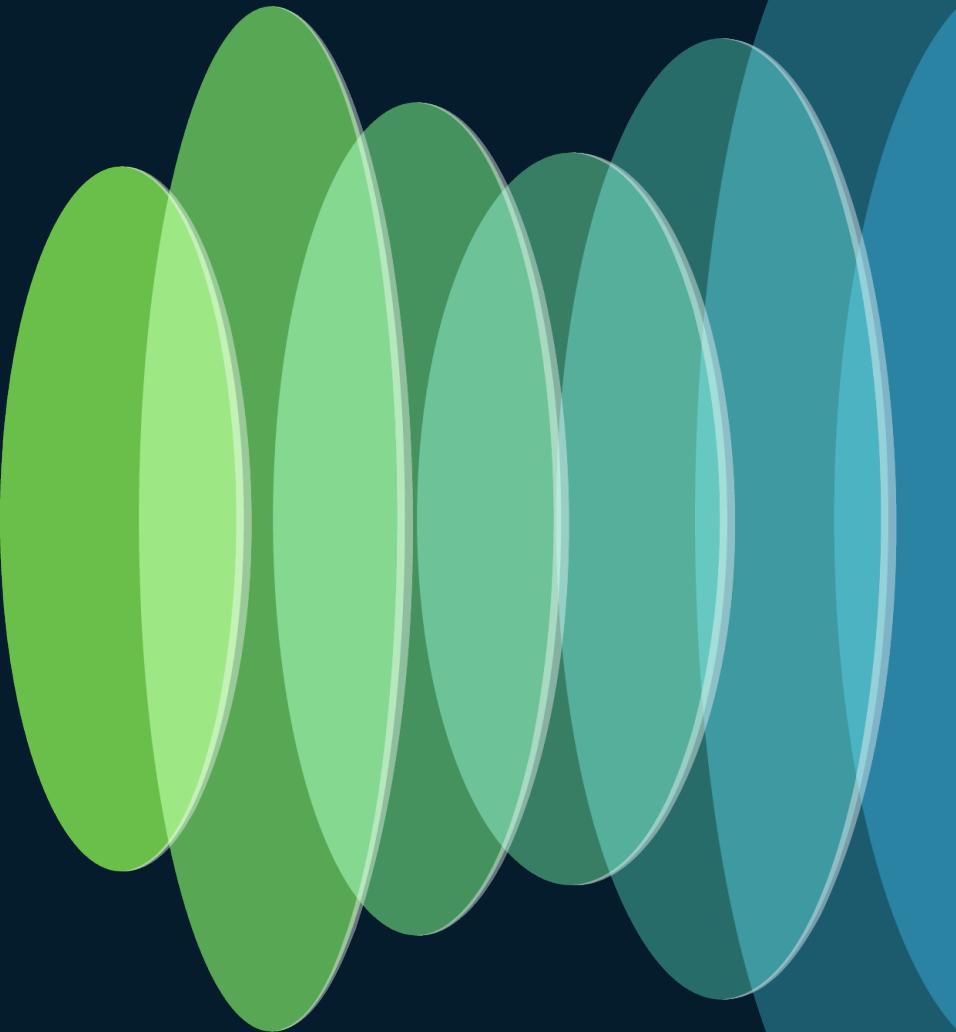


What is not covered

Workshop

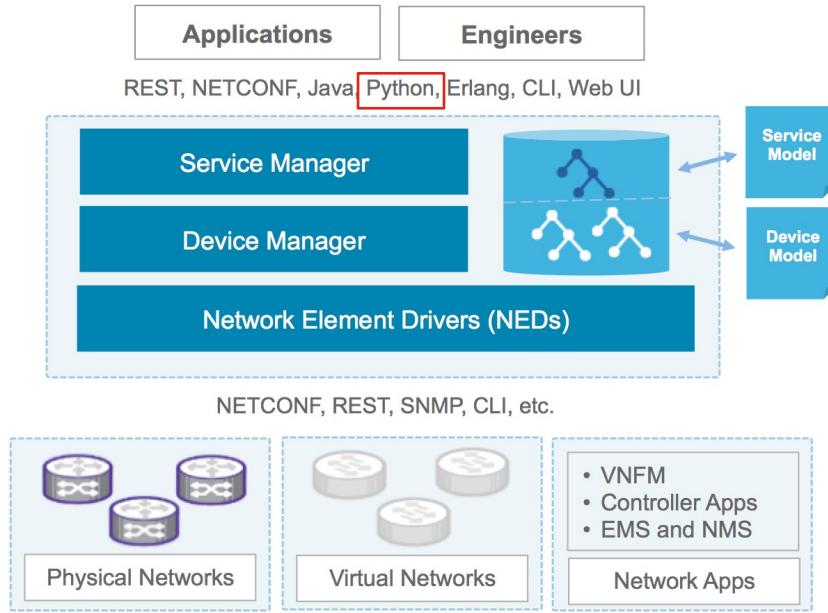
- NSO Basics. Not a 101 session.
- Python Introduction.
- Advanced NSO concepts with Python
(actions, data sharing, nano services,
cache, etc).

Core Concepts



NSO Architecture

Core Concepts



- Multivendor abstraction through Network Element Drivers (NEDs)
- Single datastore for all network elements under management
- Multiple interfaces including CLI, REST, Java Python
- Templates and compliance reporting
- YANG-based configuration schema

<https://developer.cisco.com/docs/ns0/guides/introduction-to-ns0/>

YANG Overview

Core Concepts

```
1  ...list router {  
2  ...  key name;  
3  
4  ...  uses ncs:service-data;  
5  ...  ncs:servicepoint router;  
6  
7  ...leaf name {  
8  ...  type string;  
9  ...}  
10  
11 ...leaf-list device {  
12 ...  type leafref {  
13 ...    path "/ncs:devices/ncs:device/ncs:name";  
14 ...  }  
15 ...}
```

- Data modelling language for network configuration
- Human readable
- Extensible through augmentation
- Standards-based, RFC7950
- CDB schema

<https://developer.cisco.com/docs/nso/guides/the-yang-data-modeling-language/>

NSO services

Core Concepts

- Declarative method to abstract and automate a task you want to do repeatedly
- Customer facing and resource facing services
- Internally it maintains mapping between inputs (user intent) and outputs (infrastructure configurations)



YANG Model

Defines the service in high level terms



Logic

Python/Java collects info, does verification



Template

How the service is rendered on devices



Package meta data

Version requirements, components, dependencies

<https://developer.cisco.com/docs/ns0/guides/developing-a-simple-service/>

Python API

NSO Python API/SDK

Python API

- The NSO Python, Java, Erlang (and C) APIs are **SDKs** used to extend NSO.
- Typically used by applications running on the same processor/system as NSO is running.
- Uses IPC over TCP sockets to communicate with NSO for low latency.
- Not available via `pip install`

The screenshot shows a web page titled "NSO SDK API Reference". At the top left is a dropdown menu set to "Latest". Below it is a search bar with a magnifying glass icon. The main content area has a sidebar on the left with a blue vertical line. The sidebar contains links: "API Reference", "A Note on Names", "Support Contact", and "Legal Notices". To the right of the sidebar, there are three sections with arrows: "NSO Python API", "NSO Java API", and "NSO Erlang API".

Latest

NSO SDK API Reference

API Reference

A Note on Names

Support Contact

Legal Notices

NSO Python API >

NSO Java API >

NSO Erlang API >

<https://developer.cisco.com/docs/ns0/api/ns0-sdk-api-reference/>

Use Cases

Python API

Scripts

- Connect directly to NSO datastore.
- *Onetime* operations.
 - Importing device configurations.
 - Generating reports from the configurations.

Services

- Perform complex calculations & external system integrations.
- Apply XML templates from Python.
- Get FASTMAP algorithm benefits.

High-level & Low-level APIs

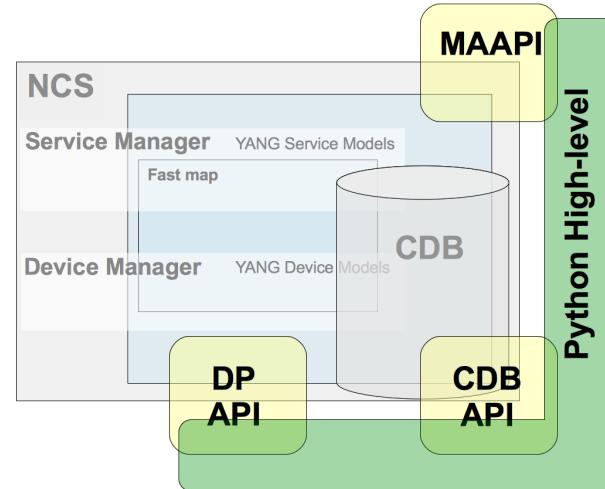
Python API

Low-level APIs are a direct mapping of the NSO C APIs, CDB and MAAPI.

- See `man confd_lib_lib` for more information.

High-level APIs are an abstraction layer on top of the low-level APIs.

- Easier to use.
- Improved code readability and development for common use cases.



<https://developer.cisco.com/docs/ns0/guides/python-api-overview/>

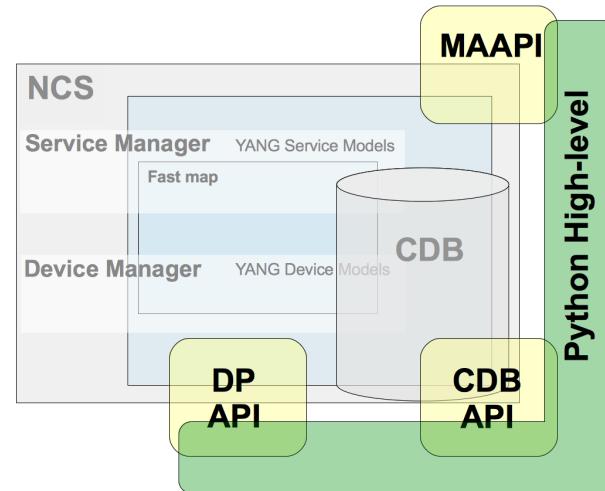
<https://developer.cisco.com/docs/ns0/guides/ncs-man-pages-volume-3/>

MAAPI API

Python API

Management Agent API

- Transactional Northbound interface.
- User session-based interface.
- Configuration & Operational data
 - Read.
 - Written and committed as one transaction.



<https://developer.cisco.com/docs/nso/guides/python-api-overview/>

MAPI API

Python API

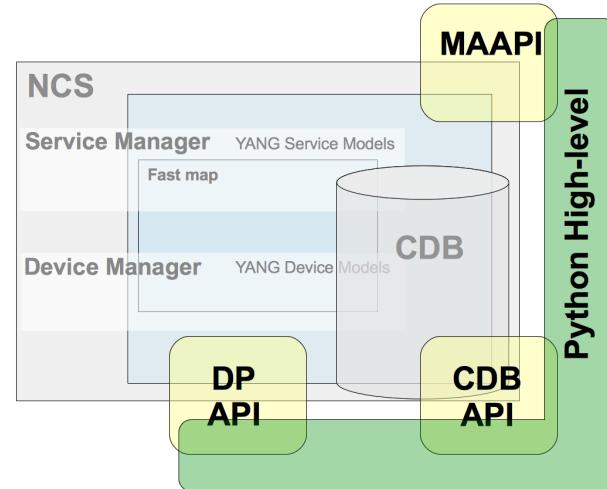
```
1 import ncs
2
3 with ncs.maapi.Maapi() as maapi:
4     with ncs.maapi.Session(maapi=maapi, user="admin", context="system"):
5         with maapi.start_read_trans() as transaction:
6             address = transaction.get_elem("/ncs:devices/device{ex0}/address")
7             print("First read: Address = %s" % address)
8
9 with ncs.maapi.single_write_trans(user="admin", context="system") as transaction:
10    transaction.set_elem2("Vegas was here", "/ncs:devices/device{ex0}/description")
11    transaction.apply()
```

<https://developer.cisco.com/docs/nso/guides/python-api-overview/#high-level-maapi-api>

MAAGIC API

Python API

- Manipulate data according to YANG schema
- Use standard Python object dot notation.
- Special characters are replaced with underscores
 - Element `my-address` becomes `my_address`
 - Crossing Namespaces with *double-underscore*
 - `root.myns__top.val`



MAAGIC = Management Agent API (MAAPI) + magic Python methods

<https://developer.cisco.com/docs/nso/guides/python-api-overview/#maagic-api>

MAAGIC API

Python API

```
# Access list item
router = root.devices.device["internet-rtr0"]

# Create list item
root.services.l3vpn.create("test-l3vpn")

# Check if list item with a key exists
"internet-rtr0" in root.devices.device

# Set leaf value
root.devices.device["internet-rtr0"].address = "10.0.0.1"

# Remove list item
del root.devices.device["internet-rtr0"]
```

<https://developer.cisco.com/docs/nso/guides/python-api-overview/#maagic-api>

MAAGIC Object Navigation

Python API

Action	Object Returned
root.devices	Container
root.devices.device	List
root.devices.device["ce0"]	ListElement
root.devices.device['ce0'].device_type.cli	PresenceContainer
root.devices.device['ce0'].address	str
root.devices.device['ce0'].port	int

Maagic object from a keypath:

```
node = ncs.maagic.get_node(transaction_id, '/ncs:devices/device{ce0}')
```

<https://developer.cisco.com/docs/nso/guides/python-api-overview/#maagic-api>

<https://developer.cisco.com/docs/nso/api/ncs-maagic/>

Transactions and Commits

Python API

- Use Python Context Managers (the “with” key word)
- Transactions are closed by default after they are applied
- Commit options can be specified
- You only need to create a write transactions for actions, not services

```
import ncs

with ncs.maapi.Maapi() as m:
    with ncs.maapi.Session(m, "admin", 'python', groups=['ncsadmin']):
        with m.start_write_trans() as t_rw:

            root = ncs.maagic.get_root(t_rw)
            device_cdb = root.devices.device["eng04-cleaf-02"]
            device_cdb.config.interface.loopback[0].description = "Done from python API"

            # Starting dry-run
            cp = ncs.maapi.CommitParams()
            cp.dry_run_native()
            dry_run_result = t_rw.apply_params(True, cp)
```

Navigate the API

Python API

Python help() function

```
>>> import ncs
>>> with
ncs.maapi.single_write_trans(user="admin",
context="system") as transaction:
... root = ncs.maagic.get_root(transaction)
... devices = root.devices
...
>>> help(devices)
```

Help on Container in module ncs.maagic object:

```
class Container(Node)
| Container(backend, cs_node, parent=None)

| Represents a yang container.

| A (non-presence) container node or a list element,
| contains other nodes.

| Method resolution order:
|     Container
|     Node
|     builtins.object

| Methods defined here:

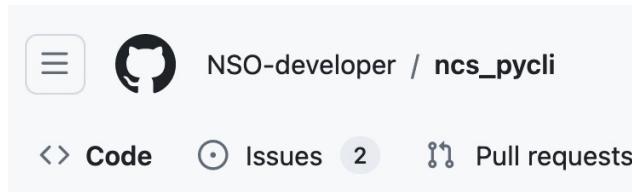
| __init__(self, backend, cs_node, parent=None)
|     Initialize Container node. Should not be called
| explicitly.

| __repr__(self)
|     Get internal representation.

| ...
```

Navigate the API

Python API



NSO-developer / **ncs_pycli**

Code Issues 2 Pull requests

ncs_pycli Public

https://github.com/NSO-developer/ncs_pycli

CISCO Live!

```
In [1]: for device in root.ncs__devices.device:  
....: print(device.name)
```

```
....:  
ce0  
ce1
```

```
In [2]: device = root.ncs__devices.device['ce0']
```

```
In [3]: type(device)  
Out[3]: ncs.maagic.ListElement
```

```
In [4]: device  
Out[4]: ListElement name=device tag=617911018 keys={ce0}
```

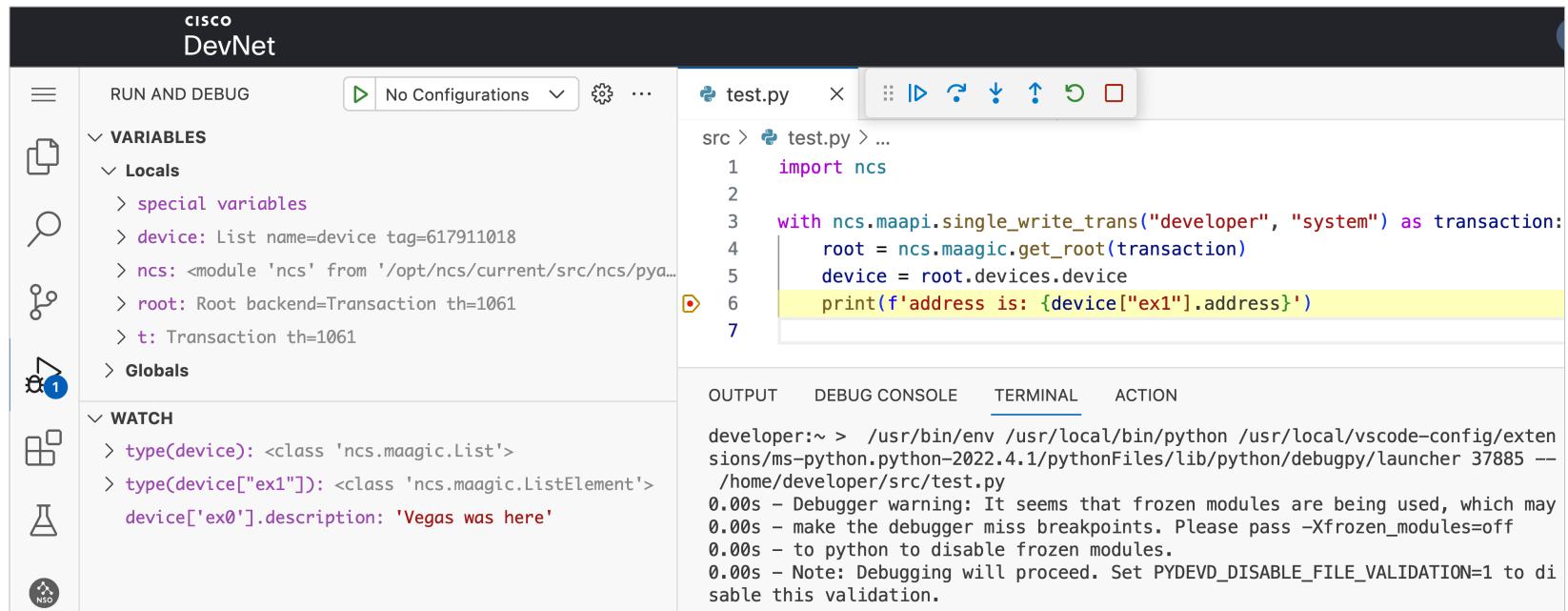
```
In [5]: help(device)
```

```
In [6]: device.  
device.active_settings  
device.address
```

Navigate the API

Python API

Debug



The screenshot shows the Cisco DevNet VS Code extension interface. The top bar displays "cisco DevNet". The left sidebar has icons for RUN AND DEBUG, VARIABLES, WATCH, and NSO. The VARIABLES section shows Locals and Globals. The WATCH section shows type(device) and type(device["ex1"]). The main editor window shows a Python file named "test.py" with the following code:

```
src > test.py > ...
1 import ncs
2
3 with ncs.maapi.single_write_trans("developer", "system") as transaction:
4     root = ncs.maagic.get_root(transaction)
5     device = root.devices.device
6     print(f'address is: {device["ex1"].address}')
7
```

A yellow box highlights the print statement at line 6. Below the editor is a terminal window showing the command run and some initial messages about frozen modules.

OUTPUT	DEBUG CONSOLE	TERMINAL	ACTION
developer:~ > /usr/bin/env /usr/local/bin/python /usr/local/vscode-config/extensions/ms-python.python-2022.4.1/pythonFiles/lib/python/debugpy/launcher 37885 -- /home/developer/src/test.py 0.00s - Debugger warning: It seems that frozen modules are being used, which may 0.00s - make the debugger miss breakpoints. Please pass -Xfrozen_modules=off 0.00s - to python to disable frozen modules. 0.00s - Note: Debugging will proceed. Set PYDEVD_DISABLE_FILE_VALIDATION=1 to di sable this validation.			

Navigate the API

Python API

CISCO DevNet Documentation Learn Technologies Community Events SIGN UP FREE LOG IN

Documentation > All > NSO > Network Services Orchestrator (NSO) v6.3

JAVA API Overview Python API Overview

- Python API Overview
- Introduction
- Python API overview
- Python scripting
- High-level MAAPI API
- Maagic API
- Maagic examples
- PlanComponent
- Python packages
- Low-level APIs
- Advanced topics
- NSO Packages
- Package Development
- Service Development Using Java
- NED Development
- NED Upgrades and Migration
- Service Handling of Ambiguous Device Models
- Scaling and Performance Optimization
- NSO Concurrency Model
- Developing Alarm Applications
- SNMP Notification Receiver
- The web server
- Kicker
- Scheduler

Python API Overview

Introduction

The NSO Python library contains a variety of APIs for different purposes. In this chapter we introduce these and explain their usage. The NSO Python modules deliverables are found in two variants, the low-level APIs and the high-level APIs.

The low-level APIs is a direct mapping of the NSO C APIs, CDB and MAAPI. These will follow the evolution of the C APIs. See [man confd_lib_maapi](#) for further information.

The high-level APIs is an abstraction layer on top of the low-level APIs to make them easier to use, to improve code readability and development rate for common use cases. E.g. services and action callbacks and common scripting towards NSO.

Python API overview

MAAPI - (Management Agent API)
Northbound interface that is transactional and user session based.
Using this interface, both configuration and operational data can be read.
Configuration and operational data can be written and committed as one transaction. The API is complete in the way that it is possible to write a new



CISCO DevNet Documentation Learn Technologies Community Events SIGN UP FREE LOG IN

Documentation > All > NSO > Network Services Orchestrator (NSO) API v6.3

Latest NSO SDK API Reference NSO Python API

- ncs
- ncs.alarm
- ncs.application
- ncs.cdb
- ncs.childlist
- ncs.dp
- ncs.error
- ncs.events
- ncs.experimental
- ncs.fsm
- ncs.ha
- ncs.keypath
- ncs.log
- ncs.maagic
- ncs.maapi

Module ncs.maapi

MAAPI high level module.

This module defines a high level interface to the low-level maapi functions.

The 'Maapi' class encapsulates a MAAPI connection which upon constructing, sets up a connection towards ConfD/NCS. An example of setting up a transaction and manipulating data:

CODE SNIPPET

```
import ncs

m = ncs.maapi.Maapi()
m.start_user_session('admin', 'test_context')
t = m.start_write_trans()
t.get_elem('/model/data{one}/str')
t.set_elem('testing', '/model/data{one}/str')
t.apply()
```

Another way is to use context managers, which will handle all cleanup related to transactions, user sessions and socket connections:

CODE SNIPPET

<https://developer.cisco.com/docs/ns0/guides/python-api-overview/#python-api-overview>

<https://developer.cisco.com/docs/ns0/api/ncs/#package-ncs>

https://developer.cisco.com/docs/ns0/guides/ncs-man-pages-volume-3/#man.3.conf_d_maapi



Navigate the API

Python API

Maagic with Devtools

```
admin@ncs# devtools true
admin@ncs# show running-config devices device ex0 | display maagic
root.ncs_devices.device['ex0'].address 127.0.0.1
root.ncs_devices.device['ex0'].port 12022
root.ncs_devices.device['ex0'].authgroup default
root.ncs_devices.device['ex0'].device-type.netconf.ned-id ne-nc-1.0
root.ncs_devices.device['ex0'].config.aaa_aaa.authentication.users.user['admin']
root.ncs_devices.device['ex0'].config.aaa_aaa.authentication.users.user['oper']

admin@ncs# config
Entering configuration mode terminal
admin@ncs(config)# devices device ex0 port 10233
admin@ncs(config-device-ex0)# top
admin@ncs(config)# show configuration | display maagic
root = ncs.maagic.get_root(t)
root.ncs_devices.device['ex0'].port = '10233'
```

<https://developer.cisco.com/docs/nso/guides/nso-cli/#displaying-the-configuration>

Navigate the API

Python API

Logs

- `tailf-ncs-python-vm.yang` & `ncs.conf` define the python-vm container.
 - See example 26: “*The Python VM YANG model*” for full explanation.
- Some of `python-vm` nodes are by default invisible.
 - See documentation for xml and cli command needed.
- Filename: `logs/ncs-python-vm-<package-name>.log` by default.

<https://developer.cisco.com/docs/nso/guides/the-nso-python-vm/#the-nso-python-vm>

Navigate the API

Python API

Debug

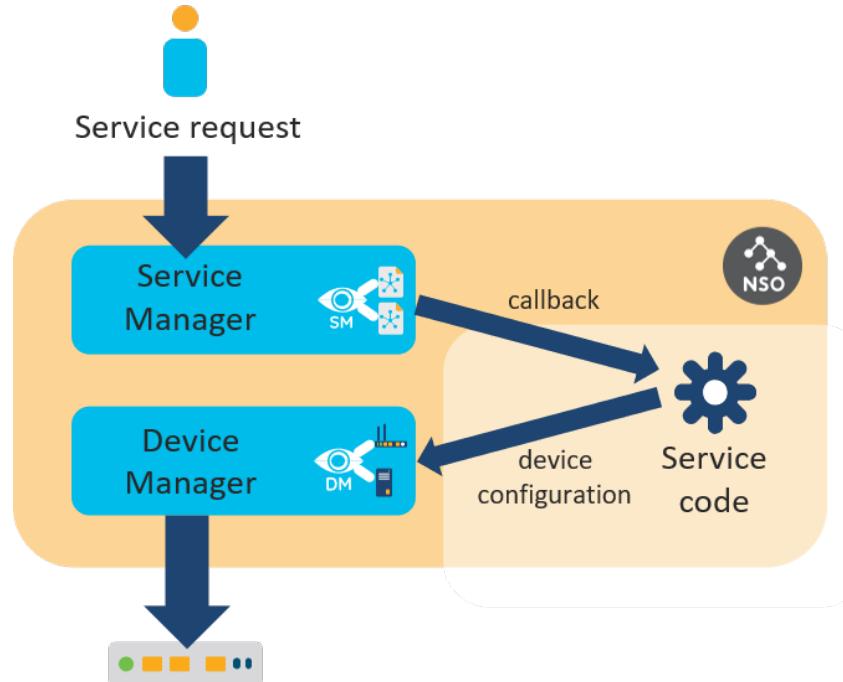
- Python packages run without an attached console.
 - Standard output collected in `ncs-python-vm.log`
- Python APIs provide logging objects based on the standard Python logging module.
- Default logging level set to info.

```
$ ncs_cli -Cu admin
admin@ncs# config
admin@ncs(config)# python-vm logging level level-debug
admin@ncs(config)# commit
```

<https://developer.cisco.com/docs/nso/guides/the-nso-python-vm/#debugging-of-python-packages>

Services with Python

Python API



<https://developer.cisco.com/docs/nso/guides/python-api-overview/#python-packages>

Services with Python

Python API

```
from ncs.application import Application
from ncs.application import Service
import ncs.template

class ServiceCallbacks(Service):
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')
        # Add the service logic

class Service(Application):
    def setup(self):
        self.log.info('Worker RUNNING')
        self.register_service('service-servicepoint', ServiceCallbacks)

    def teardown(self):
        self.log.info('Worker FINISHED')
```

```
list router {
    key name;

    uses ncs:service-data;
    ncs:servicepoint "service-servicepoint";

    leaf name {
        type string;
    }

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

<https://developer.cisco.com/docs/nso/guides/python-api-overview/#python-packages>

Services with Python

Python API

```
class ServiceCallbacks(Service):
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')

        # Add the service logic >>>>>
        vars = ncs.template.Variables()
        vars.add('MAGIC', '42')
        vars.add('CE', service.device)
        vars.add('INTERFACE', service.unit)
        template = ncs.template.Template(service)
        template.apply('pyservice-template', vars)

        self.log.info('Template is applied')

        dev = root.devices.device[service.device]
        dev.description = "This device was modified by %s" % service._path
```

<https://developer.cisco.com/docs/nso/guides/python-api-overview/#python-packages>

Services with Python

Python API

```
class ServiceCallbacks(Service):

    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')

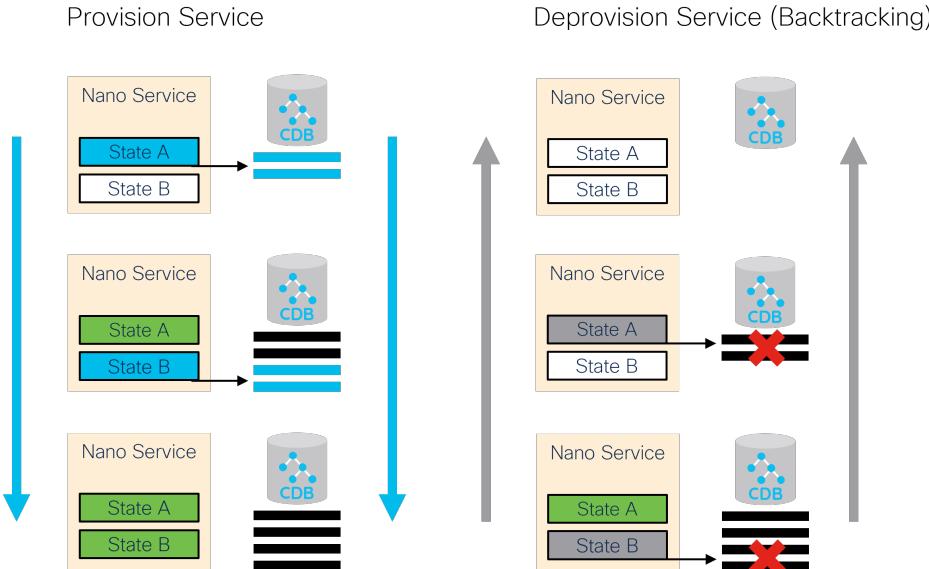
    @Service.pre_modification
    def cb_pre_modification(self, tctx, op, kp, root, proplist):
        self.log.info('Service premod(service=', kp, ')')

    @Service.post_modification
    def cb_post_modification(self, tctx, op, kp, root, proplist):
        self.log.info('Service premod(service=', kp, ')')
```

<https://developer.cisco.com/docs/nso/guides/python-api-overview/#python-packages>

Nano Services

Python API



- Reactive FASTMAP (RFM)
- Provide provisioning steps.
- Handle side-effects, useful for working with external systems.

<https://developer.cisco.com/docs/nso/guides/nano-services-for-staged-provisioning>

Playground time!

Playground

Python API

Go to

[Github.com/jillesca/DEVWKS-2551](https://github.com/jillesca/DEVWKS-2551)



Conclusion

NSO Python API

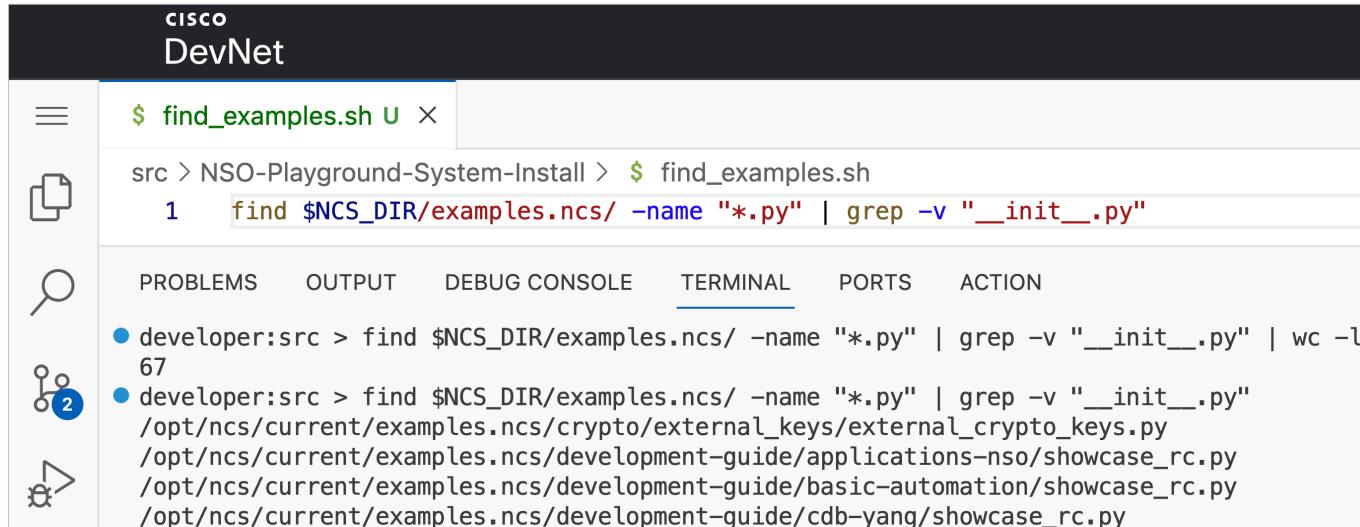
Conclusion

- NSO Python API/SDK can be used for Scripting or Services.
- Low-level & High-level Python API.
- MAAPI & MAAGIC APIs for High-level API.
- Documentation & Examples are key, read them.
- If external systems are involved, use nano services.

More Resources

NSO Built-in Python Examples

```
find $NCS_DIR/examples.ncs/ -name "*.py" | grep -v "__init__.py"
```



The screenshot shows a terminal window titled "cisco DevNet". The terminal window has a sidebar with icons for file, search, and notifications (with 2 notifications). The main area shows the command being run:

```
$ find_examples.sh U X
src > NSO-Playground-System-Install > $ find_examples.sh
1   find $NCS_DIR/examples.ncs/ -name "*.py" | grep -v "__init__.py"
```

Below the command, the terminal output shows two entries:

- developer:src > find \$NCS_DIR/examples.ncs/ -name "*.py" | grep -v "__init__.py" | wc -l
67
- developer:src > find \$NCS_DIR/examples.ncs/ -name "*.py" | grep -v "__init__.py"
/opt/ncs/current/examples.ncs/crypto/external_keys/external_crypto_keys.py
/opt/ncs/current/examples.ncs/development-guide/applications-nso/showcase_rc.py
/opt/ncs/current/examples.ncs/development-guide/basic-automation/showcase_rc.py
/opt/ncs/current/examples.ncs/development-guide/cdb-yang/showcase_rc.py

*rc in a filename stands for "RESTCONF"

Call to Action

Continue your learning journey

- Expand your knowledge
 - <https://developer.cisco.com/docs/nso/guides/python-api-overview>
- Practice the NSO Python API
 - <https://developer.cisco.com/learning/labs/service-dev-201/introduction>
- Get started with NSO
 - https://developer.cisco.com/learning/tracks/get_started_with_nso
- Review best practices for service development
 - <https://github.com/NSO-developer/nso-service-dev-practices>

Continue your learning journey after the event with Cisco DevNet!

- Personalized learning modules
- Living interactive tooling
- Other resources



Scan to get started



Continue your education

- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

Contact me at:
<https://www.linkedin.com/in/jesusillescas>



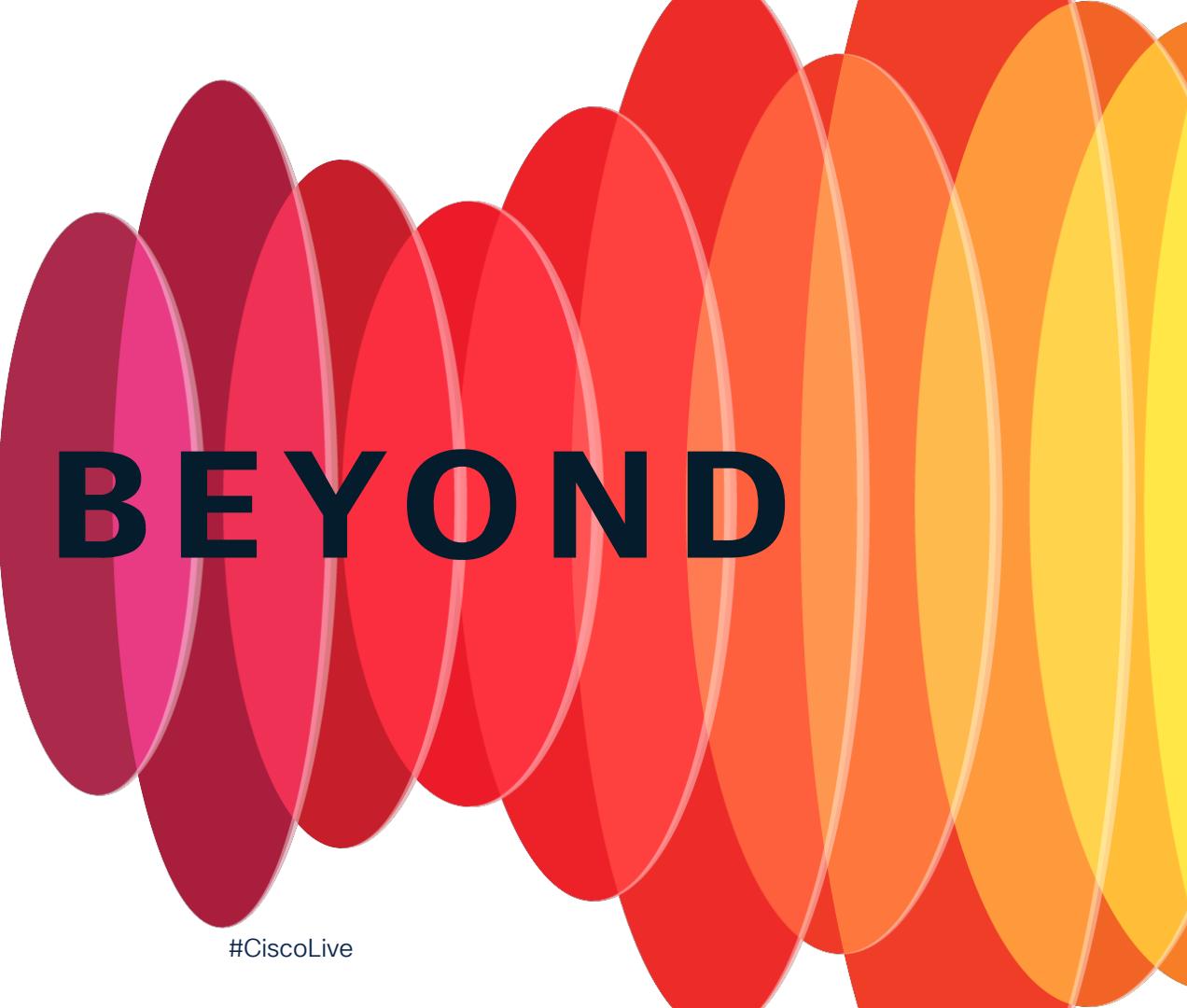
The bridge to possible

Thank you

cisco *Live!*

#CiscoLive

cisco *Live!*



GO BEYOND

#CiscoLive

Backup Slides

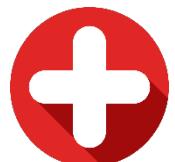
Define a service and NSO will figure out the rest

Core Concepts



Create

Convert service request into valid config
Easy: need to check resource availability



Repair

Recover from arbitrary resource changes
Hard: identify/fix or migrate to new infra



Update

Config change from change in service
Medium: need to add/release resources



Delete

Remove service instance, release resources
Medium: need reference counting of resources

FastMap automatically infers lifecycle operations from the “create” definition

Execute cli/rpc on devices

Python API

- NEDs abstract the device connection
- Actions can be used to execute arbitrary CLI commands or NETCONF RPCs against devices
- Great for complex audit use cases

```
>>> device_cdb = root.devices.device["eng04-cleaf-02"]
>>> cli_any_command = device_cdb.live_status.nx_stats__exec.any
>>> show_ip_int_br_input = cli_any_command.get_input()
>>> show_ip_int_br_input.args = ["show ip interface brief"]
>>> show_ip_int_br_result = cli_any_command.request(show_ip_int_br_input)

>>> print(show_ip_int_br_result.result)

IP Interface Status for VRF "default"(1)
Interface          IP Address      Interface Status
Lo0                123.30.164.1    protocol-up/link-up/admin-up
```

MAAGIC API

Python API

Anything modelled in YANG is available in the python API

Same concept as an SDK, but driven by YANG models

Common types

- List
- List Element
- Container

Common attributes

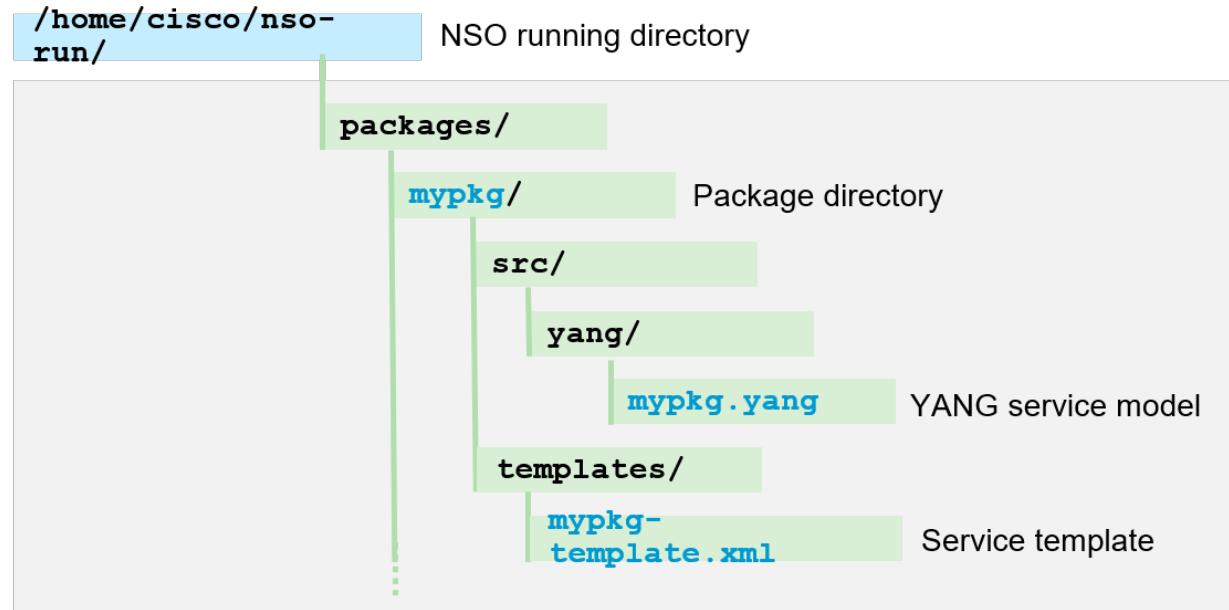
- `_parent`
- `_path`

```
>>> root = ncs.maagic.get_root(t_rw)
>>> type(root)
<class 'ncs.maagic.Root'>
>>> type(root.devices)
<class 'ncs.maagic.Container'>
>>> type(root.devices.device)
<class 'ncs.maagic.List'>
>>> type(root.devices.device["eng04-cleaf-02"])
<class 'ncs.maagic.ListElement'>
```

<https://developer.cisco.com/docs/nso/api/ncs-maagic/>

Service Package Architecture

Core Concepts



Service Challenges

Create



Modify



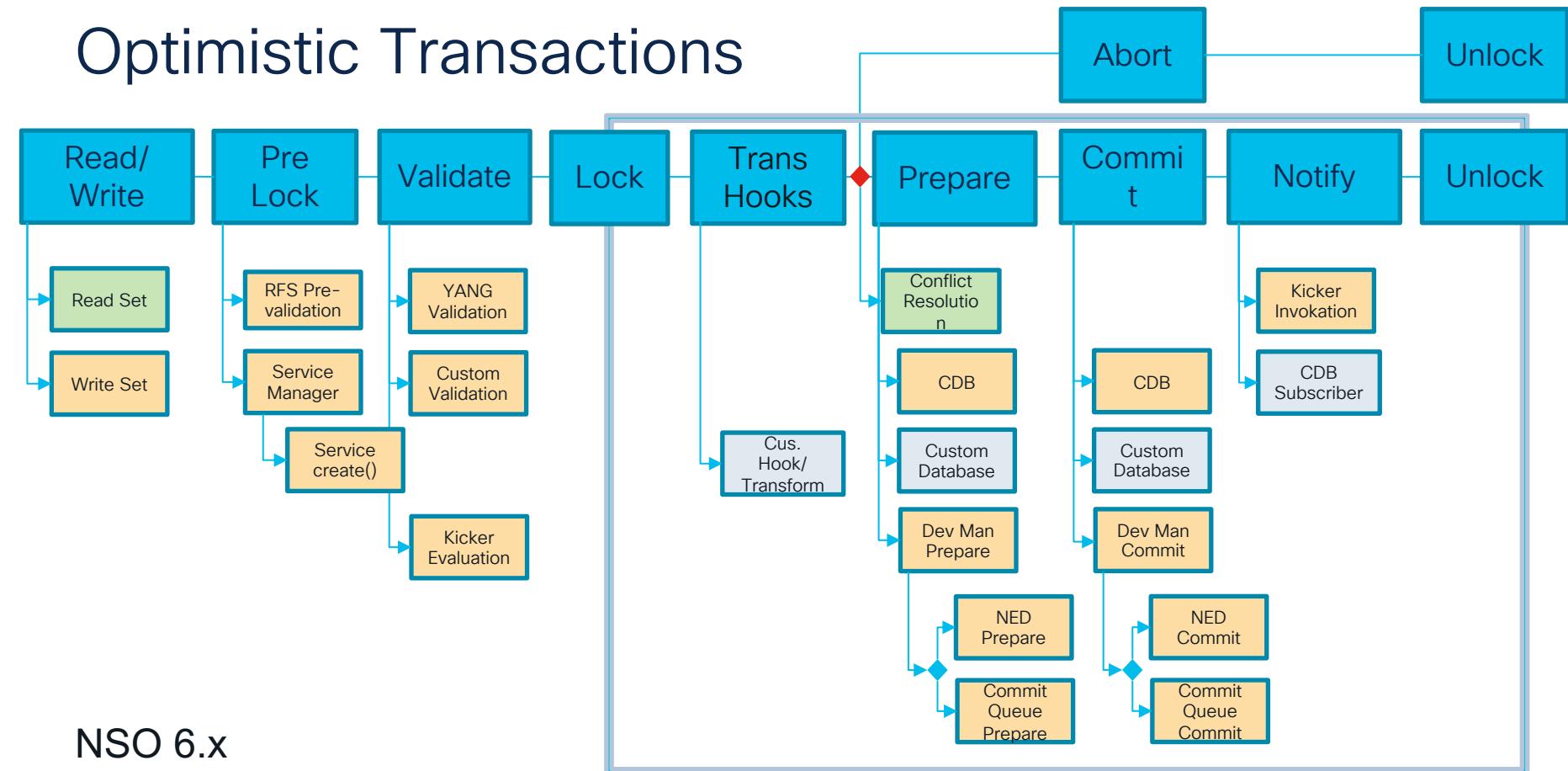
Delete



Repair

- | | | | |
|--|--|--|--|
| <ul style="list-style-type: none">• Easy• Given a set of service-level inputs, provide a known and valid output to network• May require some additional resource collection to fulfill the configuration set | <ul style="list-style-type: none">• Challenging• Allow arbitrary changes to the network service• May require collecting or handing back resources to fulfill the configuration set | <ul style="list-style-type: none">• Hard• Delete any given instance of a service and clean up the resources• May require reference counting for shared resources | <ul style="list-style-type: none">• Almost impossible• Recover from any arbitrary resource changes• Identify and fix any anomalies• Migrate to new infrastructure |
|--|--|--|--|

Optimistic Transactions



NSO 6.x

CISCO Live!