
cisco Live!
ALL IN

The bridge to possible

Advanced Automation with Cisco NSO

Viktor Leijon, Principal Engineer

BRKATO-3000

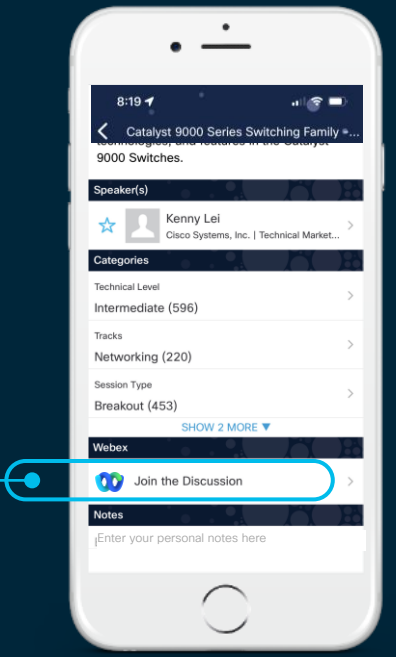
Cisco Webex App

Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space



https://ciscolive.ciscoevents.com/ciscolivebot/#BRKATO-3000

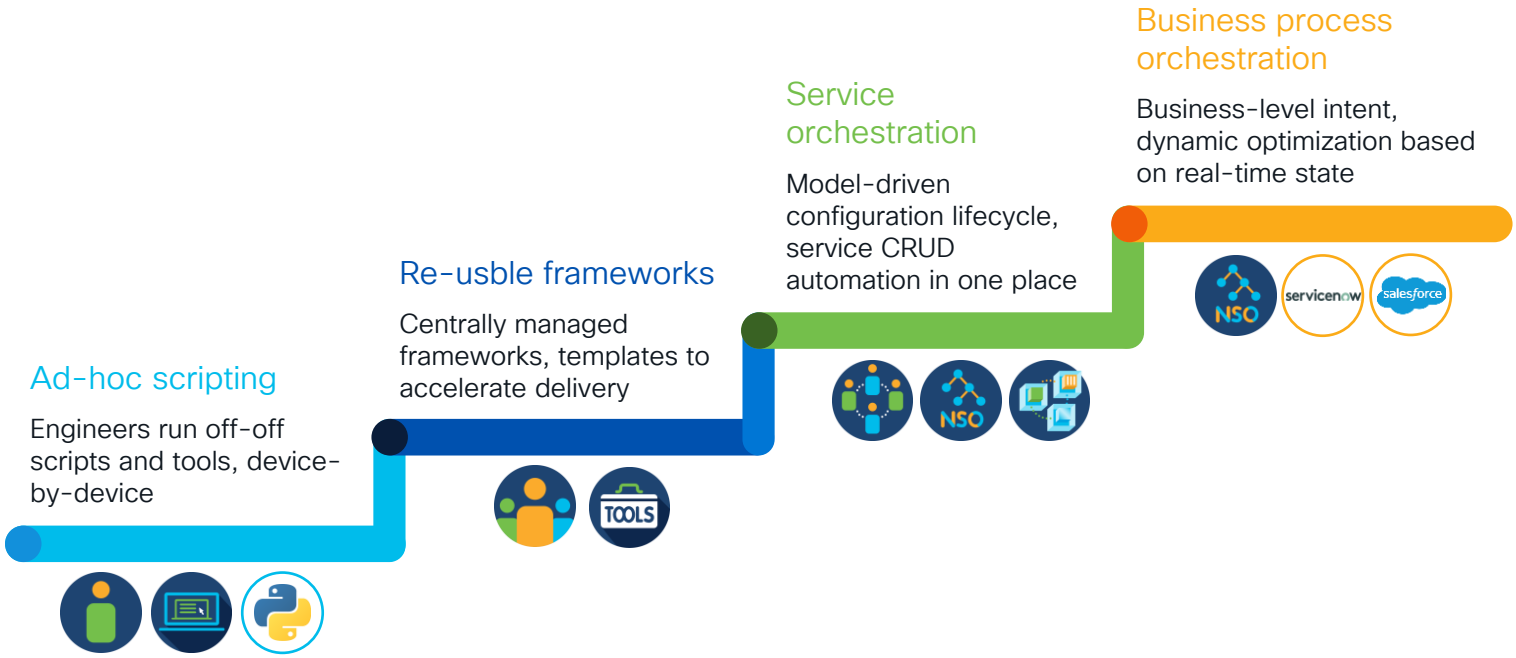
Webex spaces will be moderated by the speaker until June 17, 2022.

Agenda

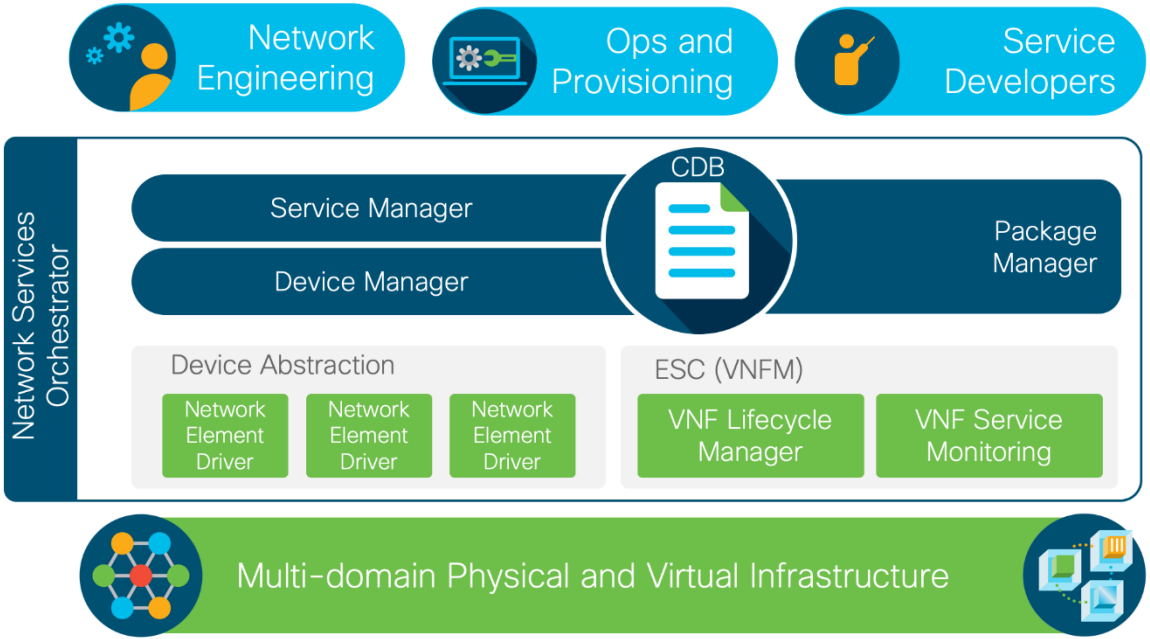
- NSO Basics
- Changing the Network State
- Building Blocks
- Designing an Advanced Service
- Conclusion and Questions

NSO Basics

Automation: From science project to indespensible

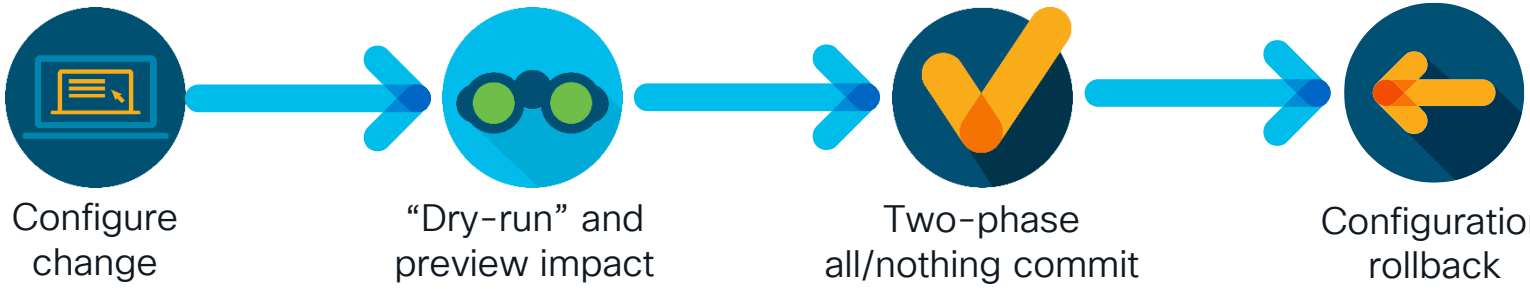


NSO Architecture



- Model-driven, end-to-end service lifecycle
- Loosely-coupled and modular architecture leveraging open APIs and standard protocols
- Orchestration across multi-domain and multi-layer for network-wide, centralized policy and services
- Seamless integration with northbound tooling

Transactions and models = no more oops



- Provides a two-phase commit protocol to address distributed network atomicity
- Dry-run and rollback capabilities for changes
- Implements full ACID properties
- Uses YANG as native schema language

Define a service and NSO will figure out the rest

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

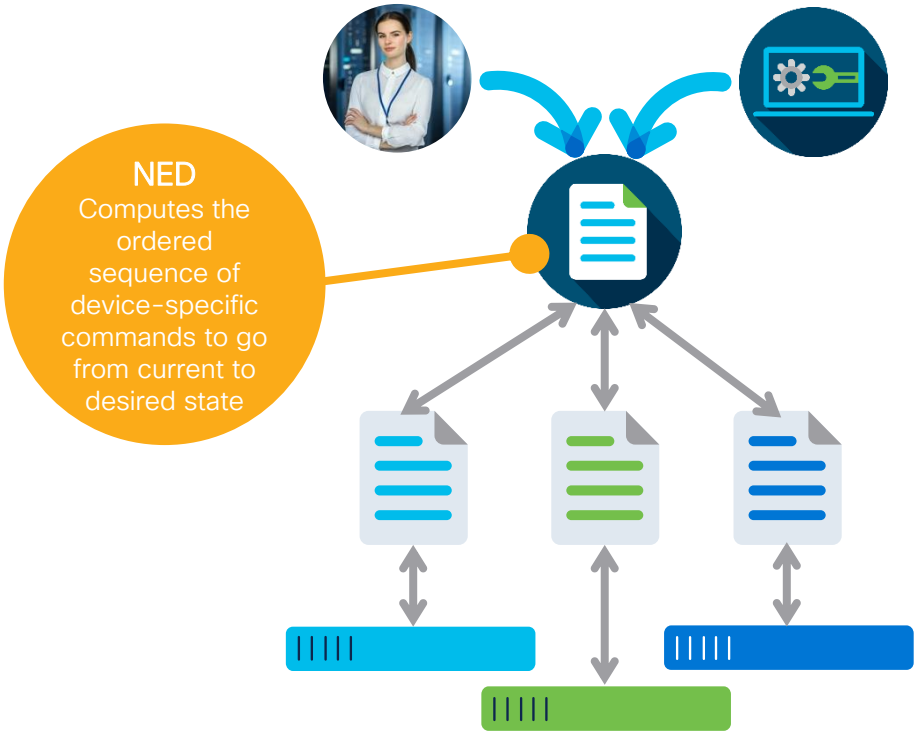
Remove service instance, release resources

Medium: need reference counting of resources

FASTMAP automatically infers lifecycle operations from the “create” definition

Individual operations don’t need to be manually coded

NEDs tame multi-vendor complexity



- Imposes the NETCONF structure on the world
- Abstracts underlying protocol and data-models
- Normalizes error-handling across vendors
- Eliminates the device adapter problem
- Removes complex device logic from the service logic

Changing the Network State

Changing the Network State

Task

The mission to be accomplished towards the network

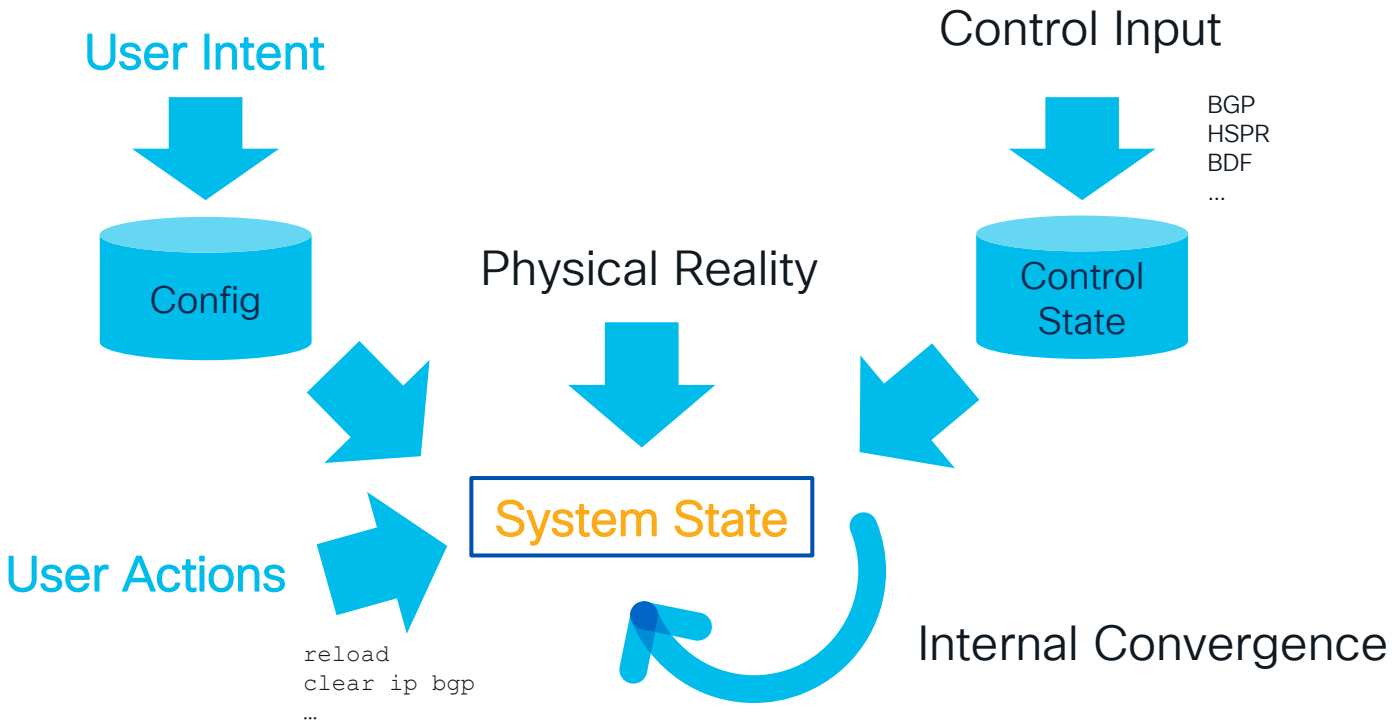
Action

An individual request towards the network

Intent

An atomic configuration change

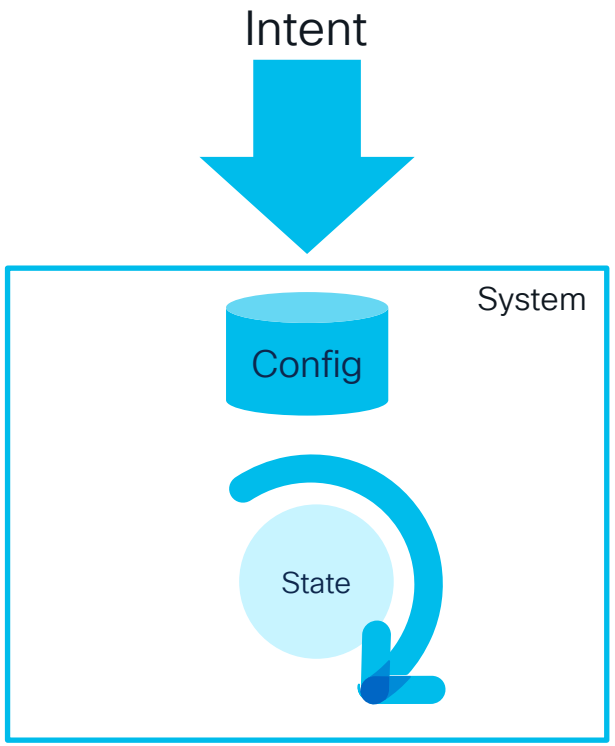
Causes of State



Intent based interface principles

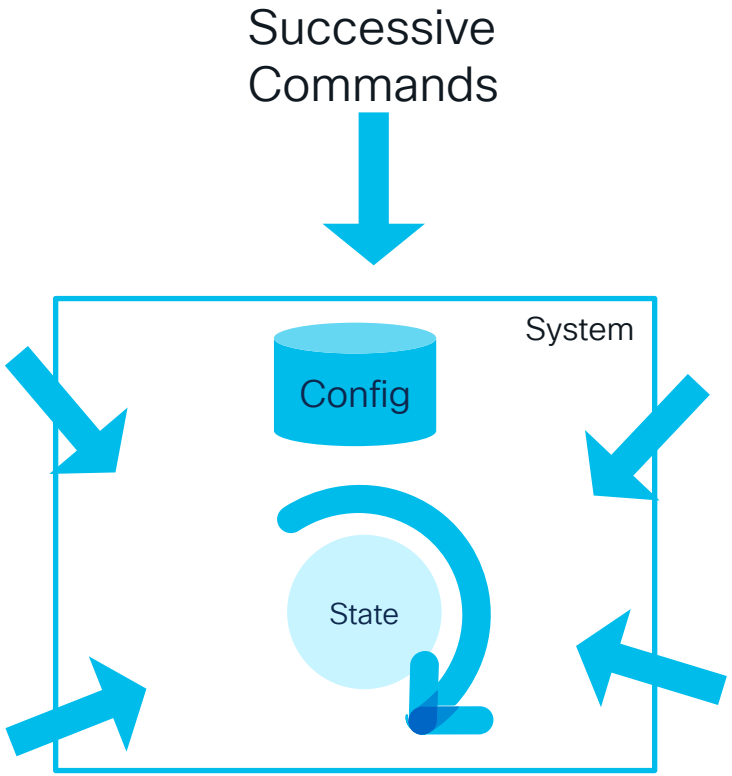
1. Writing your intent is enough
2. The system strives to execute on the intent
3. Intents are idempotent – multiple requests with the same intent has no additional effect
4. You can always write intent regardless of current state
5. The system never modifies received intent

Intent is configuration done right!



Action driven interfaces

1. Explicit commands to move between states
2. The correct command depends on the current state
3. The state is exposed to the user
4. Requires timed sequences of commands to achieve complex effects
5. Traditionally managed through workflows/runbooks

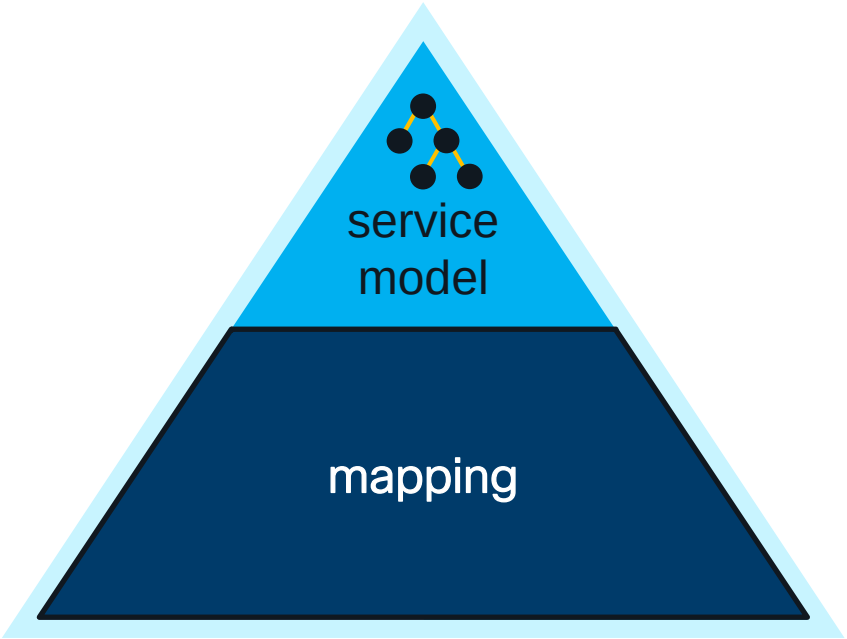


Limitations to intent based automation?

- Other systems are not all intent based?
- May want to sequence our changes
- May want human involvement
- Sometimes we need to depend on operational state
  - What if operational state changes?
- What if something goes wrong?

Building Blocks

Anatomy of a service



FASTMAP is a core NSO technology

The developer provides a callback or a template

It is invoked as part of a transaction

A two-fold process:

1. Generate the minimum config required to send northbound to the device
2. Create a service-undo configuration

Actions

- A callback from NSO to code
- Gives a CLI and REST API to your python function
- You control the transactions
  - Sequencing of events
  - Control over performance
  - Error handling possibilities
- No FASTMAP

Python

```python
class ReportAction(Action):
    @Action.action
    def cb_action(self, uinfo, name, kp, input, output, trans):
        with single_read_trans('admin', 'system') as t:
            root = get_root(t)
            result = ''
            for dev in root.devices.device:
                for iface in dev.live_status.interfaces_state.interface:
                    result += f'Device: {dev.name}, Interface: {iface.name}, Speed: {iface.speed}\n'
            self.log.info('Result: {}', result)
            output.result = result
```

RESTCONF

```
curl -u admin:admin -X POST -H 'Accept: application/yang-data+json'
http://localhost:8080/restconf/data/vpn:action/report
```

NSO CLI

```
admin@ncs# action report
Result
Device: xereal, Interface: GigabitEthernet0/0/0/0, Speed: 1000000000
Device: xereal, Interface: Loopback10, Speed: 0
Device: xereal, Interface: MgmtEth0/RP0/CPU0/0, Speed: 0
Device: xereal, Interface: Null0, Speed: 0
```

What is XPath?

- XML Path Language
- Find Paths in XML documents
- Standardized by w3c
- Used in XSLT
- YANG uses XPath 1.0

```
<world xmlns="http://example.com/xpath">
  <people>
    <name>Charlie</name>
    <age>11</age>
    <hair>long</hair>
   </people>
</world>
```

```
/world/people[name='Charlie']/name -> Charlie
/world/people[name='Charlie']/age  -> 11
/world/people[name='Charlie']/hair -> long
/world/people[name='Charlie']/* -> Charlie, 11, long
boolean(/world/people[name='Charlie']/age < 10)
  -> false
count(/world/people[name='Charlie']/*) -> 3
```

XPath and NSO

- CDB is an XML document*
- XPath in YANG
  - `must`
  - `when`
  - `leafref`
  - `identityref`
- NSO extensions
  - Templates
  - Query API
  - Kickers
  - Nano services

```
container world {
  list people {
    key name;
    leaf name {
      type string;
    }
    leaf age {
      type uint16;
    }
    leaf hair {
      type enumeration {
        enum long;
        enum short;
      }
    }
  }
}
```

cisco Live!

*Terms and conditions apply

The Basics of XPath

- The basic XPath describes a path, similar to a filesystem path
  - `/world/people/name`
  - Sometimes with prefixes `/xpath:world/xpath:people/xpath:name`
- Can have predicates
  - `/world/people[name='Charlie']`
- The results are either nodes or node-sets
- Learn more: https://gitlab.com/nso-developer/xpath-example

Kickers

- Database triggers for CDB
- Monitors part of the tree
- Based on XPath expressions
- Executes an action on activation
- An asynchronous alternative to subscribers

```
kickers data-kicker mykicker
 monitor     /alias
 kick-node   /action
 action-name kick-action
!

kickers data-kicker advanced-kicker
 monitor      /devices/device
 trigger-expr "starts-with(address, '127')"
 trigger-type enter-and-leave
 kick-node    /action
 action-name  kick-action
!
```

A simple debug kicker

```
class KickerAction(Action):
  def iterator(self, kp, op, oldv, newv):
    self.log.info(f'kp={kp}, op={OPER[op]}, newv={newv}')
    return ncs.ITER_RECURSE

  @Action.action
  def cb_action(self, uinfo, name, kp, input, output, trans):
    self.log.info(f'Triggering kicker {input.kicker_id}, ' +
                  f'for path {input.path}, tid: {input.tid}')
    with ncs.maapi.Maapi() as m:
      trans = m.attach(input.tid)
      trans.diff_iterate(self.iterator, ncs.ITER_WANT_ATTR)
      m.detach(input.tid)
```

Kicker Example

NSO

```
admin@ncs(config)# show conf
devices device ios0
 address 127.0.0.1
!
admin@ncs(config)# commit | debug kicker
 2022-05-01T20:51:29.204 kicker: advanced-kicker at /ncs:devices/ncs:device[ncs:name='ios0']
changed; invoking 'kick-action' trigger-expr false -> true
Commit complete.
```

LOG

```
<INFO> [...]:0-1-usid-421-kick-action: - Triggering kicker advanced-kicker, for path
/ncs:devices/device{ios0}, tid: 5056
<INFO> [...]: - kp=/ncs:devices/device{ios0}, op=MOP_MODIFIED, newv=None
<INFO> [...]: - kp=/ncs:devices/device{ios0}/address, op=MOP_VALUE_SET, newv=127.0.0.1
```

Designing an Advanced Service

Three Rules of Design

Carefuly, early, design is the key to advanced network automation

1

Understanding the Problem

Document the problem as completely as possible, including non-functional requirements and the operational environment

2

Lifecycle Thinking

Consider the entire CRUD cycle, including service repair and fault handling

3

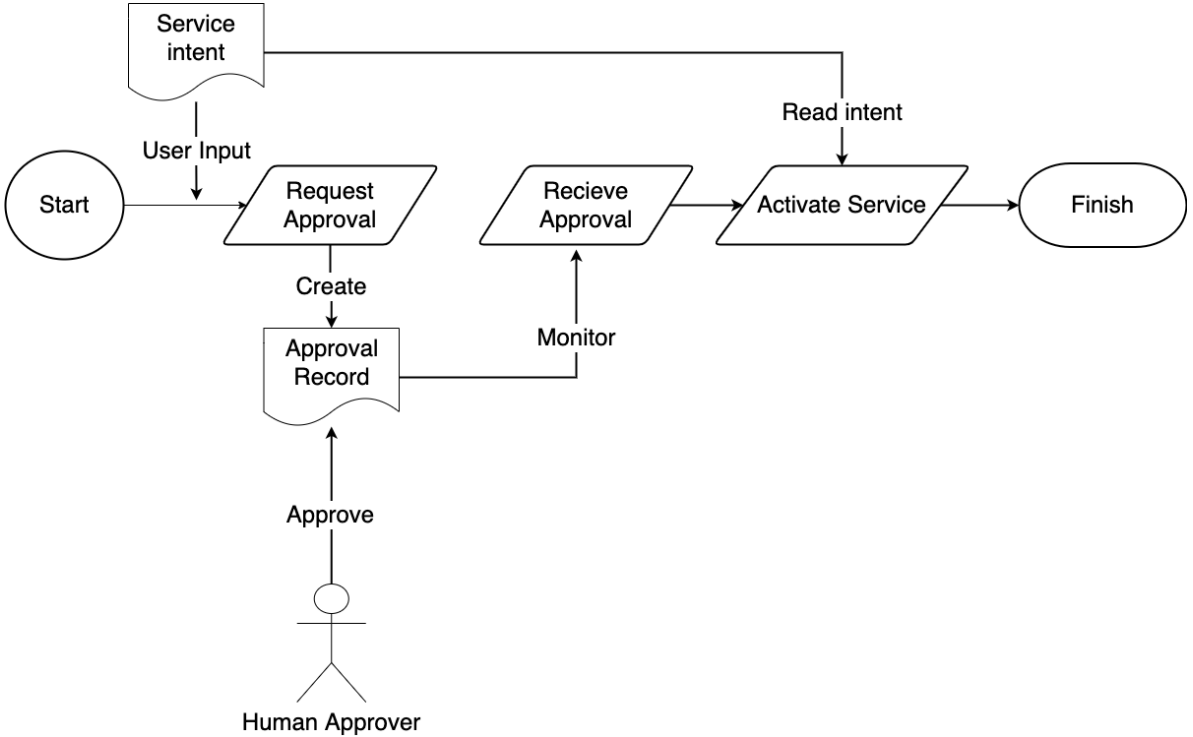
Separation of Concerns

Divide the problem into distinct pieces that can be executed independently

Example: Simple Service with Manual Approval

- Desired flow:
  1. Accept service input
  2. Generate a native dry-run
  3. Present the output to the user
  4. Wait for approval
  5. Configure the network after approval
- Additional requirements:
  - Entirely API-driven
  - CRUD-support
  - Progress report

Service Logic



Part A: Approval Mechanism

- A simple model for approvals
- Descriptive text in config
- Approval via an action
- Approval status is kickable

```
list approval {
  key id;
  leaf id {
    type string;
  }

  leaf text {
    type string;
  }

  action approve {
    tailf:actionpoint approve;
    input {
      leaf comment {
        type string;
      }
    }
  }

  container approval {
    config false;
    tailf:cdb-oper {
      tailf:persistent true;
    }
    leaf approved {
      type boolean;
      default false;
    }
    leaf comment {
      type string;
    }
  }
}
```

Approval API – RESTCONF

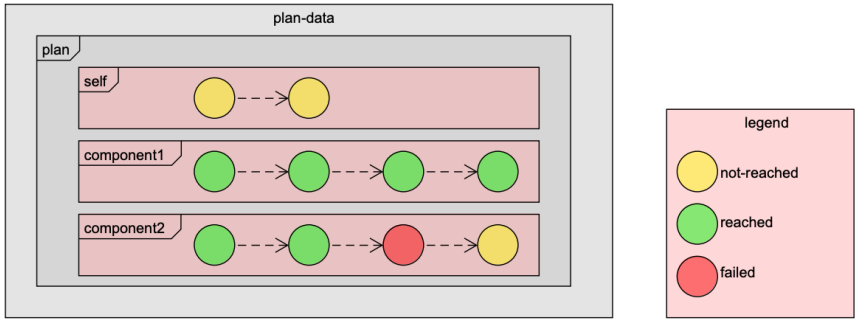
```
% curl -u admin:admin -H "Accept: application/yang-data+json" http://localhost:8080/restconf/data/vpn:approval=v1
{
  "vpn:approval": [
    {
      "id": "v1",
      "text": "Please approve: \nDevice ios0:\n router bgp 65001\n bgp log neighbor changes detail\n address-family
ipv4 unicast\n  advertise best-external\n exit\nexit\n”
    }
  ]
}

% curl -u admin:admin -X POST -H "Content-Type:  application/yang-data+json” \
http://localhost:8080/restconf/data/vpn:approval=v1/approve \
      -d '{"comment": "Looks ok"}’

% curl -u admin:admin -H "Accept: application/yang-data+json" http://localhost:8080/restconf/data/vpn:approval=v1
{
  "vpn:approval": [
    {
      "id": "v1",
      "text": "Please approve: \nDevice ios0:\n router bgp 65001\n bgp log neighbor changes detail\n address-family
ipv4 unicast\n  advertise best-external\n exit\nexit\n",
      "approval": {
        "approved": true,
        "comment": "Looks ok”
      }
    }
  ]
}
```

Part B: A Nano Service

- With FASTMAP there is a single diffset
  - Delete and update happens for all parts at once
- In some cases update/delete has to be gradual
- Nano-services allow for efficient life cycle implementation
- Declarative execution control



Service Model

- Standard service model
- Added nano-plan-data

```
list vpn {

  action request-approval {
    tailf:actionpoint requestapproval;
  }

  key name;
  leaf name {
    type string;
  }

  uses ncs:nano-plan-data;
  uses ncs:service-data;
  ncs:servicepoint vpn-servicepoint;

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

  leaf override-approval {
    type boolean;
    default "false";
    tailf:hidden full;
  }
}
```

Nano service component

- A single component
- Four states
  - Init
  - Waiting-approval
  - Approved
  - Ready

```
ncs:component-type "ncs:self" {
  ncs:state "ncs:init";
  ncs:state "vpn:waiting-approval" {
    ncs:create {
      ncs:nano-callback;
      ncs:post-action-node "$SERVICE" {
        ncs:action-name "request-approval";
      }
    }
  }
  ncs:state "vpn:approved" {
    ncs:create {
      ncs:pre-condition {
        ncs:any {
          ncs:monitor "/approval[id=$SERVICE/name]"{
             ncs:trigger-expr
               "approval/approved = 'true'";
          }
          ncs:monitor "$SERVICE" {
              ncs:trigger-expr
                "override-approval = 'true'";
          }
        }
      }
      ncs:nano-callback;
    }
  }
  ncs:state "ncs:ready";
}
```

Behavior tree

- Simples possible tree
- A single instance of the component we defined

```
ncs:service-behavior-tree vpn-servicepoint {
  ncs:plan-outline-ref "vpn:vpn-plan";
    ncs:selector {
      ncs:create-component "'self'" {
      ncs:component-type-ref "ncs:self";
    }
  }
}
```

Request Approval

- Creates approval records
- Runs after initial service creation
- Does not run on re-deploys

```
class RequestApproval(Action):
  @Action.action
  def cb_action(self, uinfo, name, kp,
                input, output, trans):
    self.log.info(f'Action request approval {kp}')
    # Do a dry run and collect output
    msg = "Please approve: \n"
    with single_write_trans('admin', 'system') as trans:
      svc = get_node(trans, kp)
      svcname = svc.name
      # Use override-approval to enable a dry-run
      svc.override_approval = True
      cp = ncs.maapi.CommitParams()
      cp.dry_run_native()
      rv = trans.apply_params(True, cp)
      if 'device' in rv:
        devices = rv['device']
        for dev in devices:
          msg += f"Device {dev}:\n {devices[dev]}"
      self.log.info("Message: ", msg)
    # Create the approval record
    with single_write_trans('admin', 'system') as trans:
      root = get_root(trans)
      approval = root.approval.create(svcname)
      approval.text = msg
      trans.apply()
```

Service Creation

```
admin@ncs# conf
Entering configuration mode terminal
admin@ncs(config)# vpn example device ios1
admin@ncs(config-vpn-example)# commit
Commit complete.
admin@ncs(config-vpn-example)# end
admin@ncs# show vpn example plan
            BACK                                                                POST ACTION
TYPE  NAME  TRACK  GOAL  STATE              STATUS       WHEN                 ref  STATUS
-------------------------------------------------------------------------------------------
self  self  false  -     init               reached      2022-05-02T02:17:09  -    -
                          waiting-approval   reached      2022-05-02T02:17:09  -    create-reached
                          approved           not-reached  -                    -    -
                          ready              not-reached  -                    -    -
admin@ncs# show running-config approval
approval example
 text Please approve:
      Device ios1:
       router bgp 65001
       bgp log neighbor changes detail
       address-family ipv4 unicast
        advertise best-external
       exit
      exit

!
```

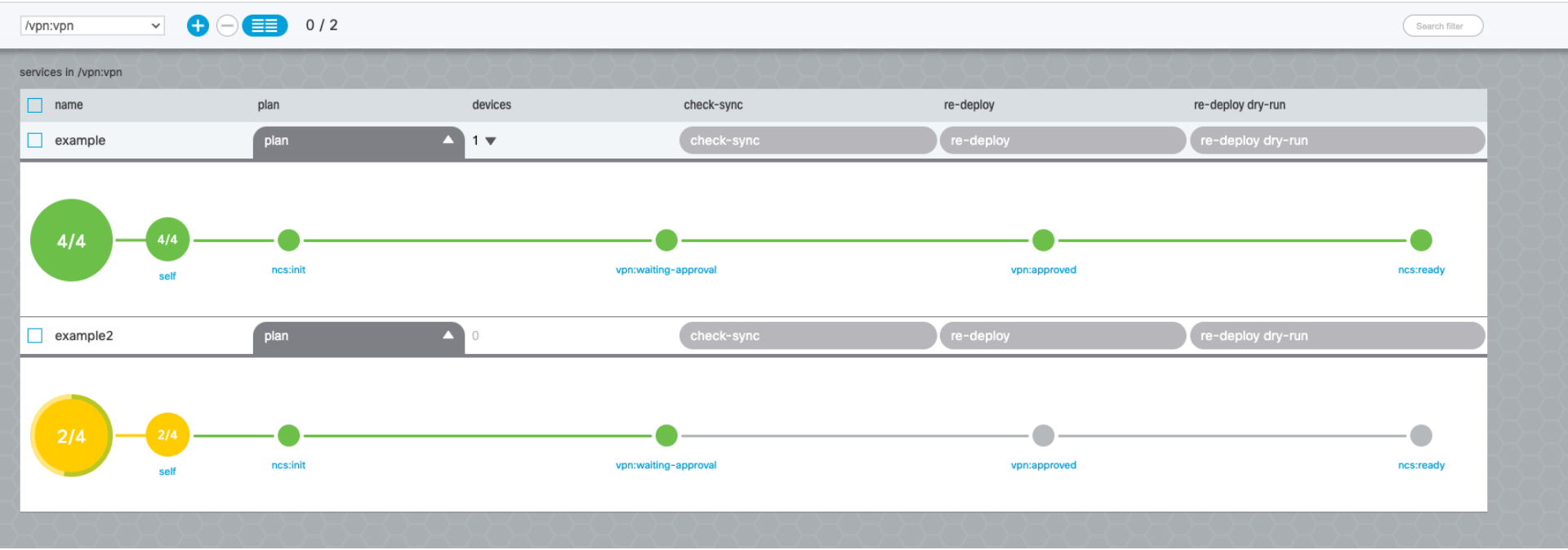
Automatic Kicker Creation

```
kickers data-kicker "pre-condition: /vpn:vpn{example}/plan/component{ncs:self
self}/state{vpn:approved}/pre-conditions/create/pre-condition{0}"
 monitor      /approval[id=$SERVICE/name]
 trigger-expr "approval/approved  = 'true'"
 variable PLAN
  value /vpn[name='example']/plan
 !
 variable SERVICE
  value /vpn[name='example']
 !
 variable ZOMBIE
  value "/ncs:zombies/ncs:service[ncs:service-
path=\"/vpn[name='example']\"]"
 !
 kick-node    /vpn:vpn[vpn:name='example']
 action-name  reactive-re-deploy
!
```

Approval

```
admin@ncs# show approval
ID       APPROVED  COMMENT
-----------------------------
example  false     -
admin@ncs# approval example approve comment "Looks good"
admin@ncs# show approval
ID       APPROVED  COMMENT
-------------------------------
example  true      Looks good
admin@ncs# show vpn plan
                        LOG               BACK                                                                  POST ACTION
NAME     FAILED  MESSAGE  ENTRY  TYPE  NAME  TRACK  GOAL  STATE             STATUS   WHEN                 ref  STATUS          ID
-------------------------------------------------------------------------------------------------------------------------------
example  -       -        -      self  self  false  -     init              reached  2022-05-02T02:17:09  -    -
                                                          waiting-approval  reached  2022-05-02T02:17:09  -    create-reached
                                                          approved          reached  2022-05-02T02:18:54  -    -
                                                          ready             reached  2022-05-02T02:18:54  -    -
```

NSO WebUI



Conclusion

- Principles are important
- Nano services can take your automation one step further
- This is just the start
- We are here to help you
- Scale up in complexity to meet your needs

Start simple

Separation of concerns

Nano-services

https://github.com/vleijon/cl2022

Technical Session Surveys

- Attendees who fill out a minimum of four session surveys and the overall event survey will get Cisco Live branded socks!
- Attendees will also earn 100 points in the Cisco Live Game for every survey completed.
- These points help you get on the leaderboard and increase your chances of winning daily and grand prizes.

Cisco Learning and Certifications

From technology training and team development to Cisco certifications and learning plans, let us help you empower your business and career. www.cisco.com/go/certs

Pay for Learning with Cisco Learning Credits

(CLCs) are prepaid training vouchers redeemed directly with Cisco.

Learn

Cisco U.
IT learning hub that guides teams and learners toward their goals

Cisco Digital Learning
Subscription-based product, technology, and certification training

Cisco Modeling Labs
Network simulation platform for design, testing, and troubleshooting

Cisco Learning Network
Resource community portal for certifications and learning

Train

Cisco Training Bootcamps
Intensive team & individual automation and technology training programs

Cisco Learning Partner Program
Authorized training partners supporting Cisco technology and career certifications

Cisco Instructor-led and Virtual Instructor-led training
Accelerated curriculum of product, technology, and certification courses

Certify

Cisco Certifications and Specialist Certifications
Award-winning certification program empowers students and IT Professionals to advance their technical careers

Cisco Guided Study Groups
180-day certification prep program with learning and support

Cisco Continuing Education Program
Recertification training options for Cisco certified individuals

Here at the event? Visit us at **The Learning and Certifications lounge at the World of Solutions**

Continue your education

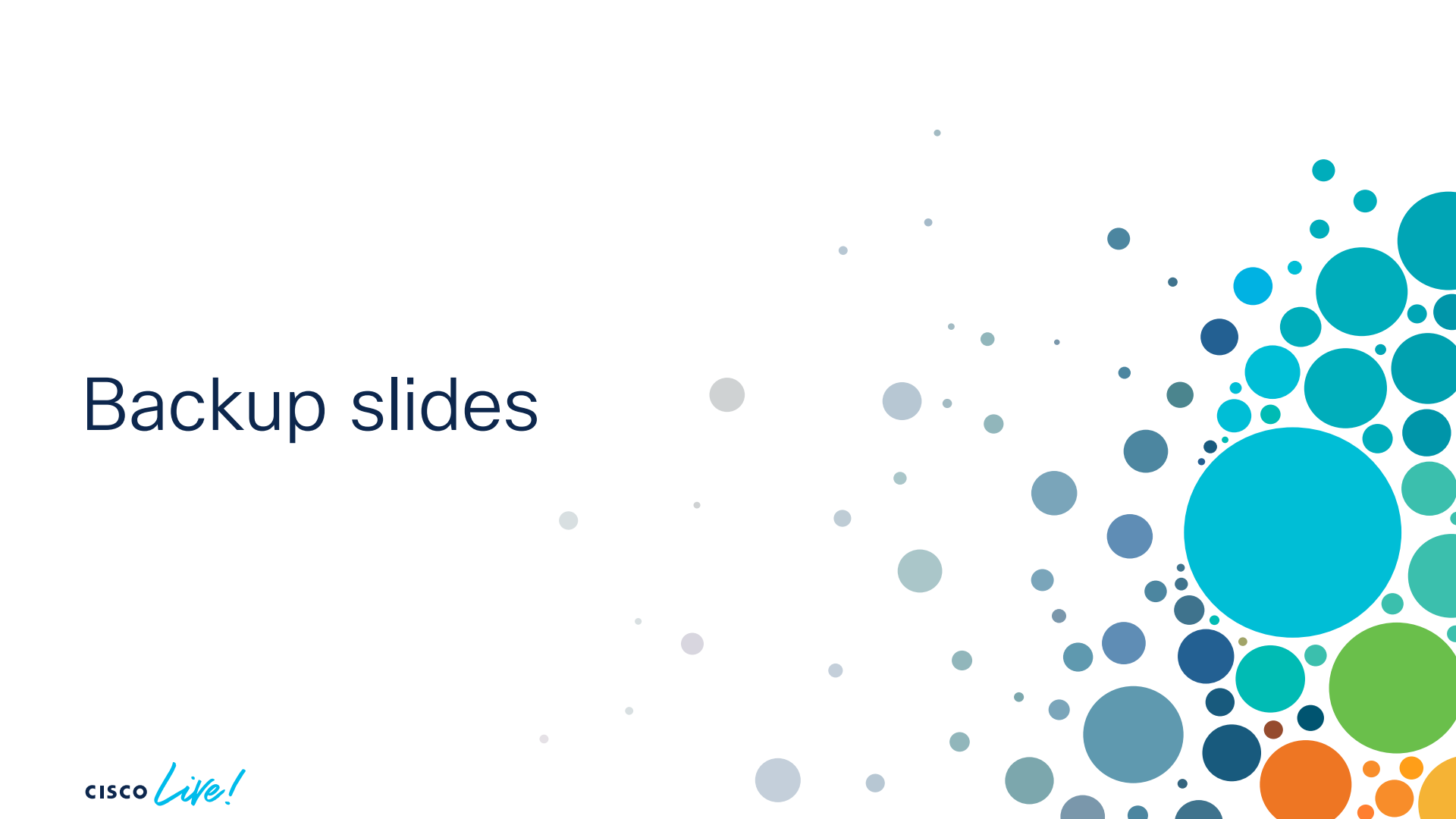
- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

cisco Live!

Thank you


cisco Live!
ALL IN

Backup slides

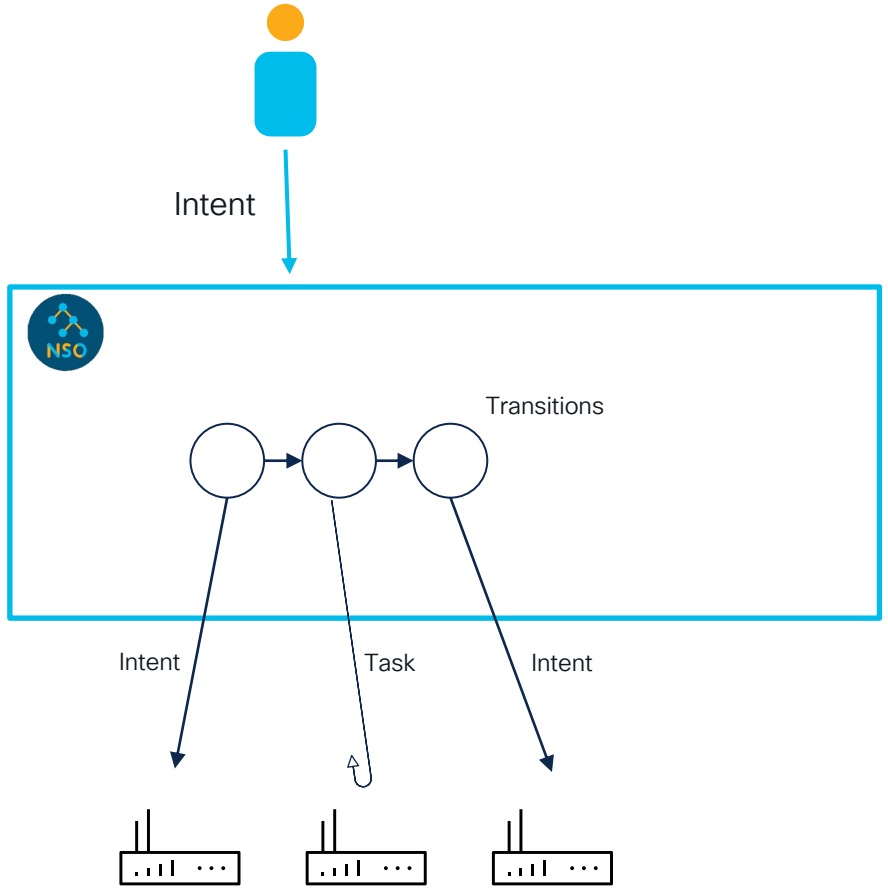


Tasks

- An operation that can be easily defined
- Has a beginning and an end
- Executes a distinct command

Combining Transactions and Tasks: Transitions

- Intent is divided into transitions
- Each transition is meaningful
- We aim for stepwise convergence
- Tasks can be used between the pieces of intent



Rich software interfaces = easier integration

IETF RESTCONF

python

> CLI

170+ 3rd-party devices

IETF NETCONF

ERLANG

Web UI

Industry’s broadest multivendor support