Developer Days
Automation

cisco
The bridge to possible

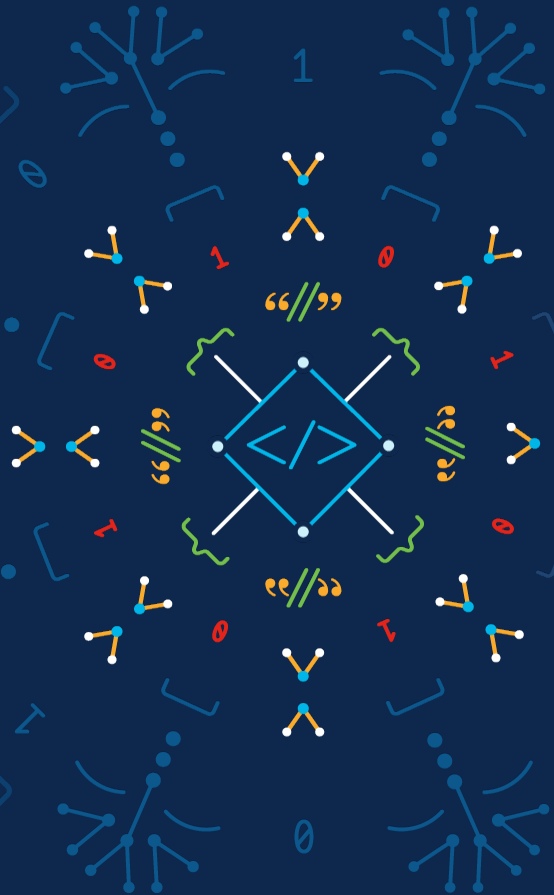
Reaching Network Automation Level 5

LAB

Principles and Practice

Per Andersson
Viktoria Fordos
Jonas Johansson
Ulrik Stridsman
Jan Lindblad
NSO Engineering Architects

May 2024

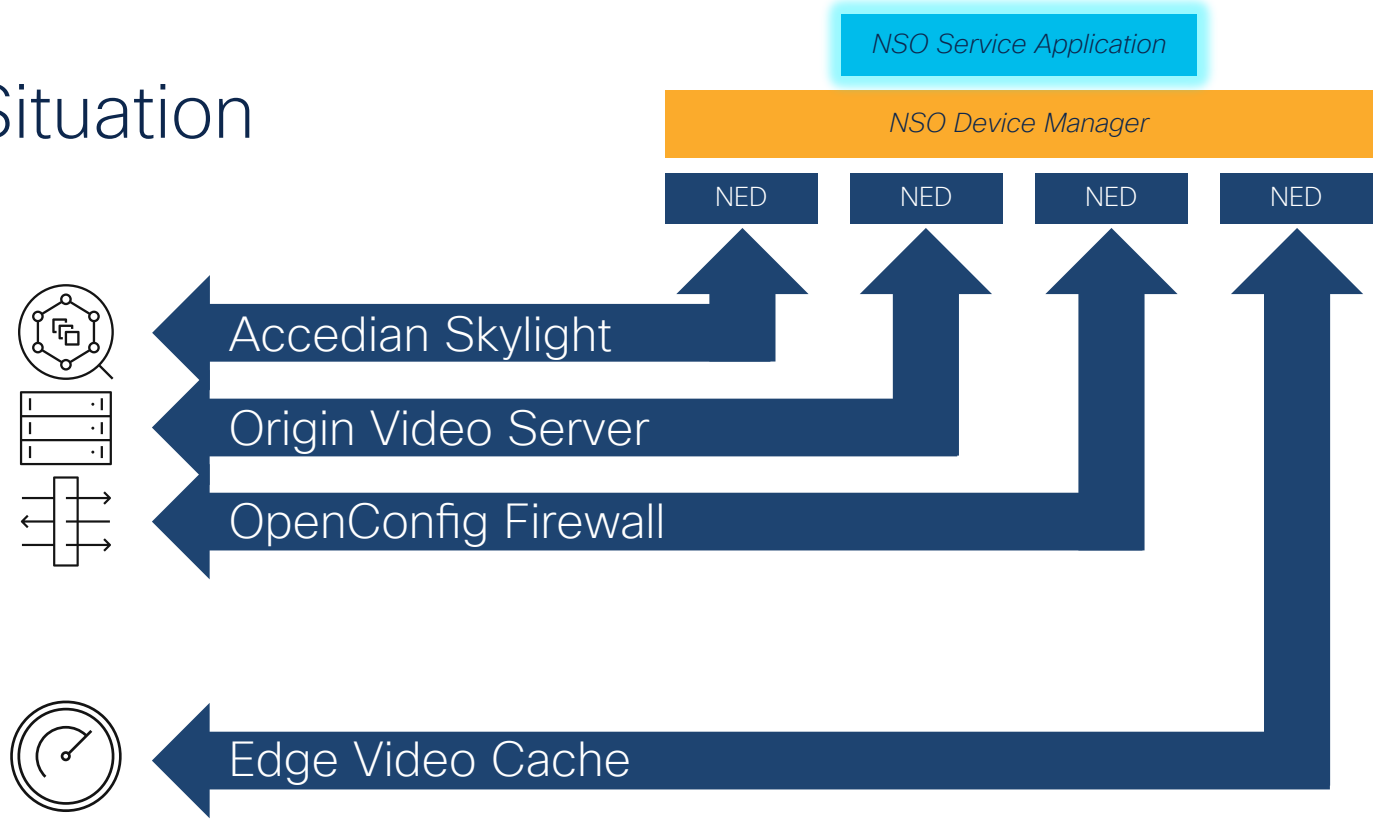


Agenda

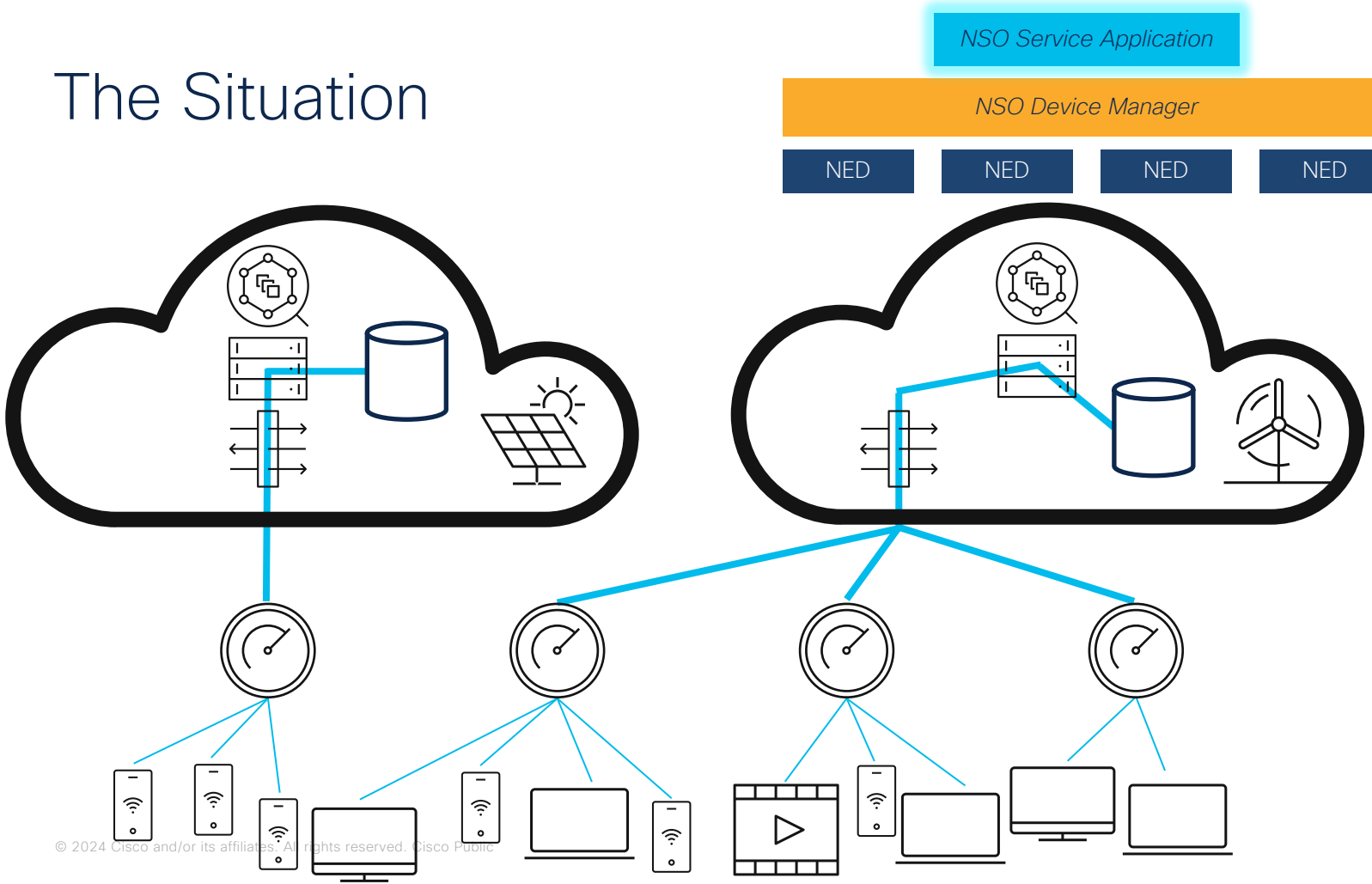
https://github.com/janlindblad/
nso-sustainability-automation-example

- ~~Automation levels~~
- Show the level 3 service
- Task 1: Go to level 4
  - Work
  - Discussion
- Task 2: Go to level 5
  - Work
  - Discussion
- Implications of level 4-5
- Conclusion

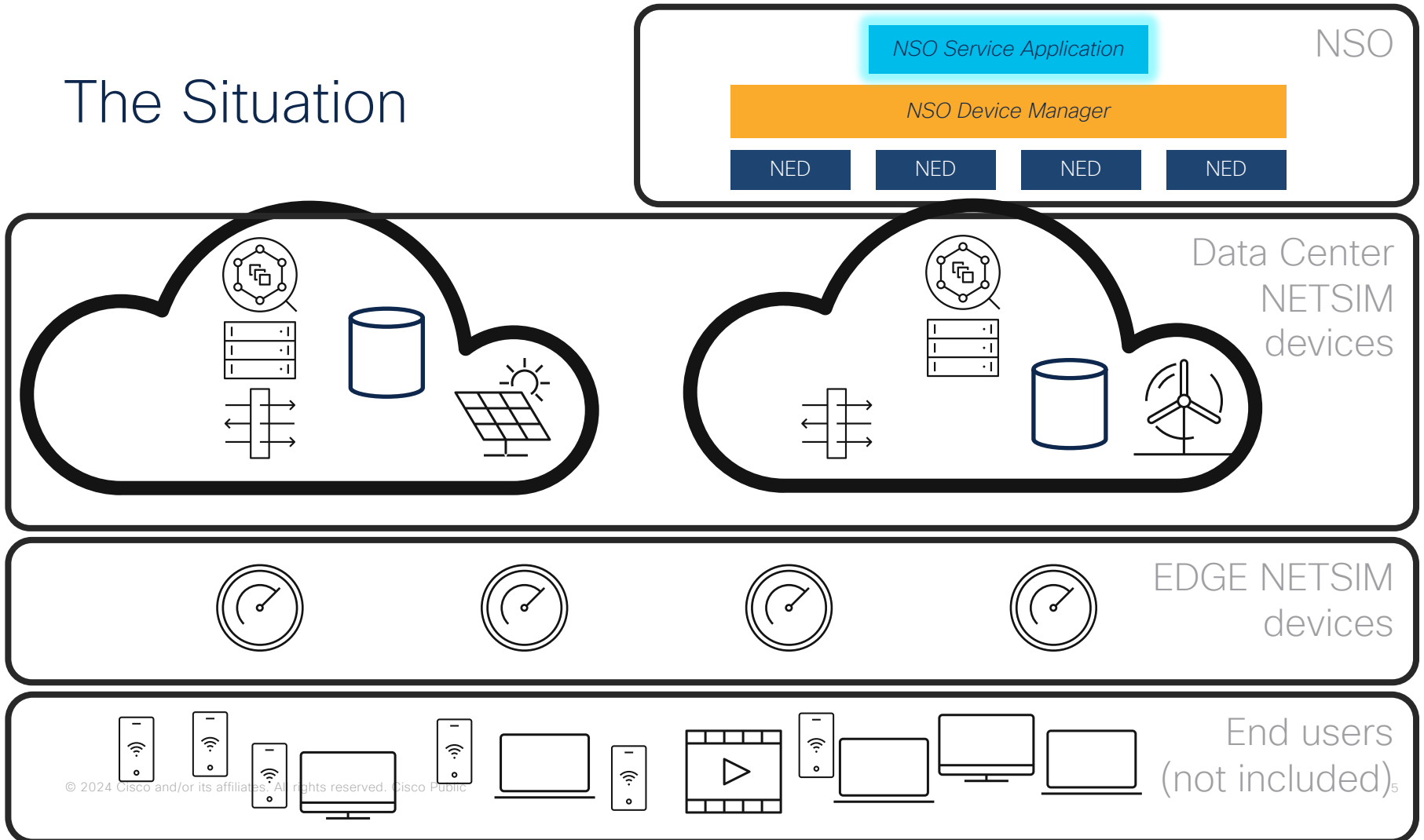
The Situation



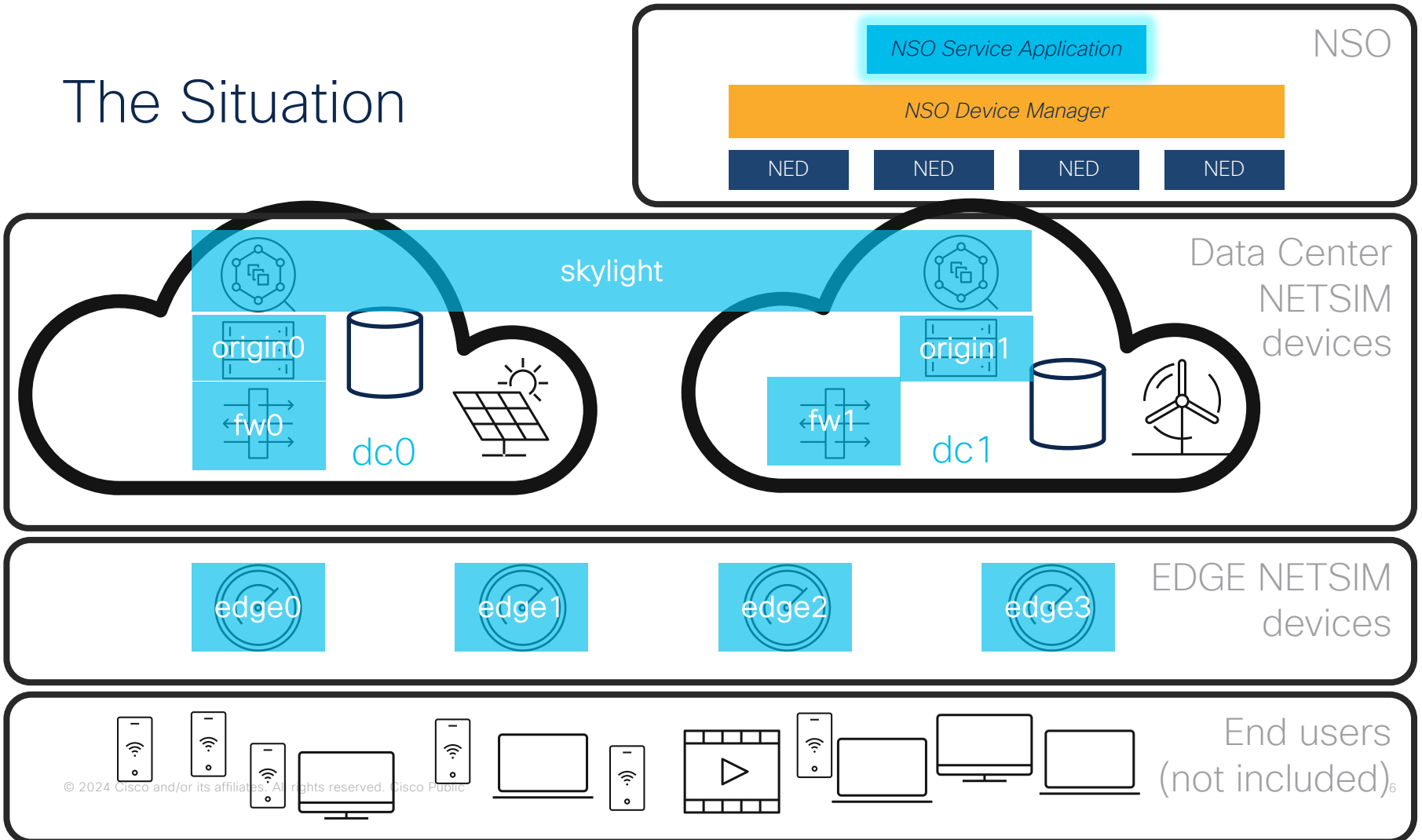
The Situation



The Situation



The Situation



Level 3 DEMO

- Some handy scripts
  + streaming-switch-level.sh,
  + netsim-simulate-jitter.sh
- Reloading packages & checking status
- Looking at the DCs
- Creating and connecting an Edge device
- Looking at the Edge Plan
- Unblocking the DC
- Looking at the Edge Plan again

Code Walk 1 / 4

A look at the Service YANG

Streaming Service YANG
list edge – The Service itself

```
list edge {
  key name;

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

  leaf name {
    type leafref {
      path "/ncs:devices/ncs:device/ncs:name";
    }
    must "derived-from (…/ncs:ned-id, 'edge-nc:edge-nc')" {
      error-message "Only the name of an edge device
                     makes sense here.";
    }
  }
  leaf dc {
    type leafref {
      path "/streaming:dc/streaming:name";
    }
  }
  action load-from-storage {
    tailf:actionpoint load-from-storage;
    output {
      leaf result {
        type string;
      }
    }
  }
}
```

The list of "edge" service instances

This list is a (nano-)service, so it has a servicepoint

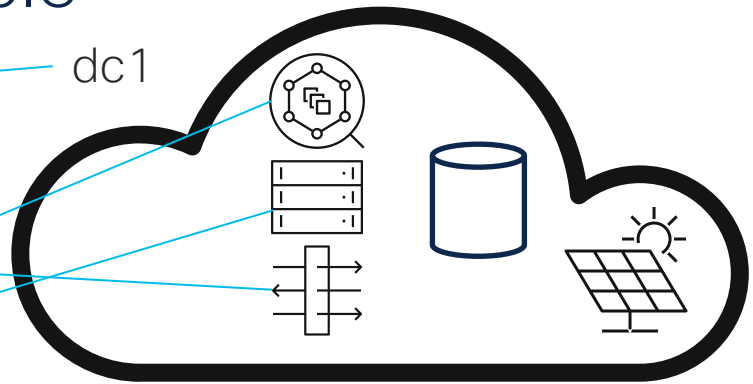
This service is tied hard 1:1 to an "edge" video cache device

Each edge service instance is connected to a DC. Operator configures which one

The load-from-storage action tells the origin video server to ensure it has all the titles currently present in the edge video cache. Useful when an edge device switches DC

Streaming YANG
list dc – Useful Mapping-table

```
list dc {
  key name;
  leaf name {
    type string;
  }
  leaf fw {
    type leafref {
      path "/ncs:devices/ncs:device/ncs:name";
    }
    must "derived-from(…/ncs:ned-id, 'firewall-nc')" {
      error-message "Only the name of a firewall device
                     makes sense here.";
    }
  }
  leaf media-origin { … }
  leaf skylight { … }

  leaf is-active {
    type boolean;
    default false;
  }
}
```

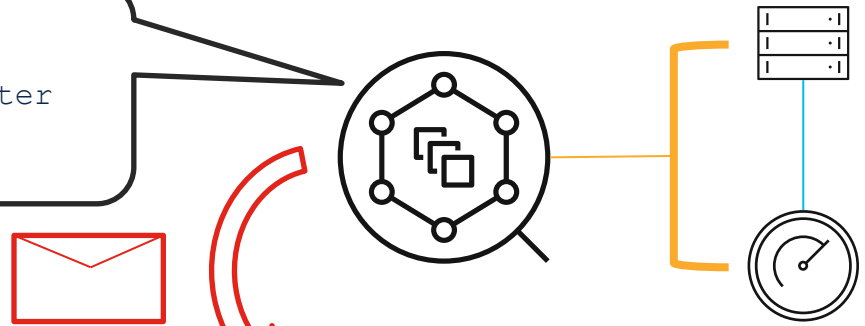


Manual configuration switch to enable DC. Operator checks if DC ready and configures 'true'

Streaming YANG action skylight-notification

You can manually trigger this notification:

```
devices device skylight rpc
  rpc-send-notification-high-jitter
  send-notification-high-jitter
  device dc1
```



Too much jitter

Configured Notification kicker

NSO

```
container actions {
  action skylight-notification {
    tailf:actionpoint skylight-notification;
    input {
      uses kicker:action-input-params;
    }
  }
…
}
```

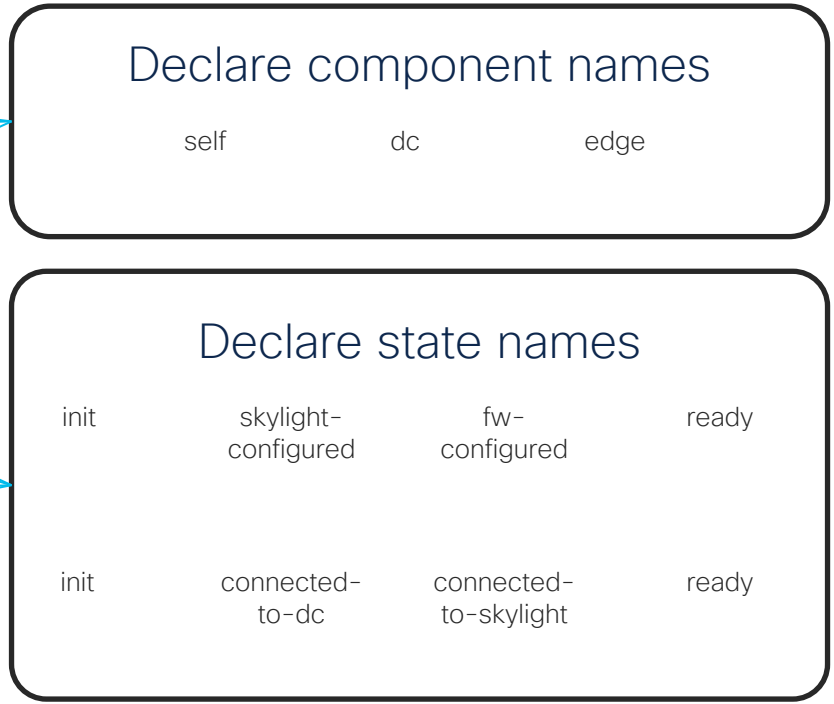
packages/streaming/python/streaming/skylight_notification_action.py

Code Walk 2 / 4

A look at the Nano-Service Plan

Streaming Nano Service Plan YANG Component and State name Declarations

```
…

identity dc {
  base ncs:plan-component-type;
}
identity skylight-configured {
  base ncs:plan-state;
description "Add DC to Skylight";
}
identity fw-configured {
  base ncs:plan-state;
  description "Allow traffic between media origin and edge";
}

identity edge {
  base ncs:plan-component-type;
}
identity connected-to-skylight {
  base ncs:plan-state;
  description "Add edge to Skylight";
}
identity connected-to-dc {
  base ncs:plan-state;
  description "Connect edge to DC, synchronize content list";
}
```

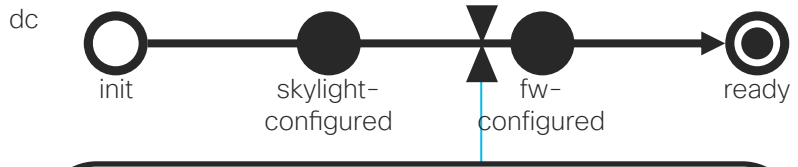


Streaming Nano Service Plan YANG dc component

```
ncs:component-type "dc" {

  ncs:state "ncs:init";

  ncs:state "streaming:skylight-configured" {
    ncs:create {
      ncs:nano-callback;
    }
  }

  ncs:state "streaming:fw-configured" {
    ncs:create {
      ncs:nano-callback;
      ncs:pre-condition {
        ncs:monitor "/streaming:dc[name = $SERVICE/dc]" {
          ncs:trigger-expr "is-active = 'true'";
        }
      }
    }
  }

  ncs:state "ncs:ready";
}
```

nano-callback

Means that this state has a piece of java, python or template to be run/applied



pre-condition

These are useful for waiting for other things to complete before starting processing of this state

Streaming Nano Service Plan YANG edge component

```
ncs:component-type "edge" {

  ncs:state "ncs:init";

  ncs:state "streaming:connected-to-dc" {
    ncs:create {
      ncs:nano-callback;
      ncs:post-action-node "$SERVICE" {
        ncs:action-name "load-from-storage";
        ncs:result-expr "result = 'true'";
      }
    }
  }

  ncs:state "streaming:connected-to-skylight" {
    ncs:create {
      ncs:nano-callback;
    }
  }

  ncs:state "ncs:ready";
}
```

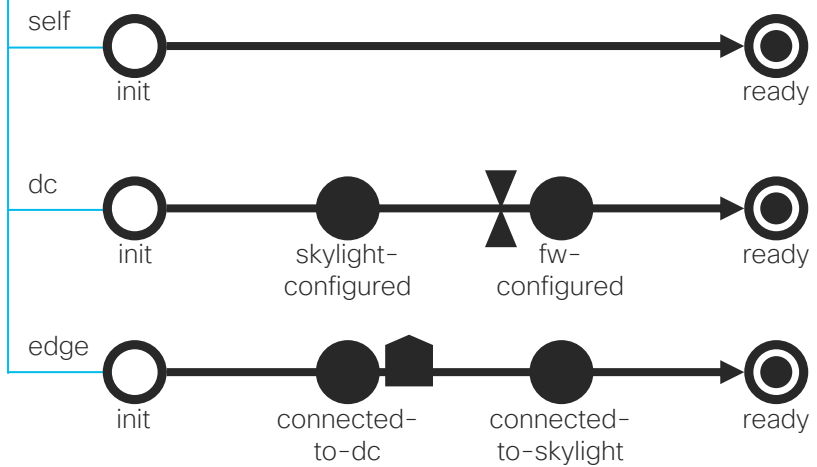
post-action-node

Means this state will call an action after it has been reached/completed. This is useful for side effects *(don't put side effects into ordinary state handling code!)*

Streaming Nano Service Plan YANG attachment to servicepoint

```
ncs:service-behavior-tree edge-servicepoint {
  ncs:plan-outline-ref "streaming:edge-plan";

  ncs:selector {
    ncs:create-component "'self'" {
      ncs:component-type-ref "ncs:self";
    }
    ncs:create-component "'dc'" {
      ncs:component-type-ref "streaming:dc";
    }
    ncs:create-component "'edge'" {
      ncs:pre-condition {
        ncs:monitor
          "$SERVICE/plan/component[type='streaming:dc']
           /state[name='ncs:ready']" {
          ncs:trigger-expr "status = 'reached'";
        }
      }
      ncs:component-type-ref "streaming:edge";
    }
  }
}
```

Edge service behavior tree servicepoint "edge-servicepoint"



pre-condition

Entire components can also have pre-conditions. They are created if/when the condition becomes true

Code Walk 3 / 4

A look at the Service Templates

Streaming Template
edge-servicepoint-edge-connected-to-skylight.xml

```
<config-template xmlns="http://tail-f.com/ns/config/1.0"
                 servicepoint="edge-servicepoint"
                 componenttype="streaming:edge"
                 state="streaming:connected-to-skylight">
  <?set EDGE = {./name}?>
  <?set DC = {./dc}?>
  <?set SESSION_ID = "851d1691-aba9-80a9-cc55-0a0c1219ba16"?>
  <?set-root-node {/}?>
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>skylight</name>
      <config>
        <sessions xmlns="…:accedian-gateway-orchestrator">
          <session>
            <sessionIdentifier>{$SESSION_ID}</sessionIdentifi…
            <orchestratorType>agent</orchestratorType>
            <sessiontype>twamp</sessiontype>
            <ifStateful>true</ifStateful>
            <agent>
              <agentId>{/streaming:dc[streaming:name=$DC]/
                        streaming:skylight-agent-id}</agentId>
              <agentSessionName>{$DC}-agent-to-{$EDGE}-twamp<…
              <enable>true</enable>
              <period>continuous</period>
            </agent>
            <twamp>
              <senderDscp>0</senderDscp>
              <reflectorAddr>{/ncs:devices/ncs:device
                 [ncs:name=$EDGE]/ncs:address}</reflectorAddr>
              <reflectorPort>4000</reflectorPort>
```

Templates with a `servicepoint` are applied directly by the NSO service manager. NSO will not run java/python code for this servicepoint/state.

Hard-coded `SESSION_ID` value?!

This is only here to prevent the template from failing with an error. To make the lab work properly, you need to provide a unique `SESSION_ID` value.

{ expression to be evaluated }

Expressions may use `$VARIABLES` and/or `/xpath` constructs towards the service YANG or other places.

Code Walk 4 / 4

A look at the Python Service Code

Streaming Service Python package-meta-data.xml and setup in main.py

NSO knows about packages.
Each package has a package-meta-data.xml
This file tells NSO what to run:

```
<ncs-package xmlns="http://tail-f.com/ns/ncs-packages">
  <name>streaming</name>
  <package-version>1.0</package-version>
  <description>NSO Automation Levels Example
Service</descrip…
  <ncs-min-version>6.1</ncs-min-version>
  <component>
    <name>main</name>
      <application>
        <python-class-name>streaming.main.Main</python-class…
    </application>
  </component>
</ncs-package>
```

```
class Main(ncs.application.Application):
  '''Nano service appliction implementing the nano create
     callback'''

    def setup(self):
        self.log.info('Main RUNNING')
        self.register_nano_service(
            servicepoint='edge-servicepoint',
            componenttype="streaming:edge",
            state="streaming:connected-to-skylight",
            nano_service_cls=ConnectedToSkylight)

        self.register_action('skylight-notification',
                             SkylightNotificationAction)
        self.register_action('optimize',
                             StreamerOptimizeAction)
        self.register_action('vary-energy-price',
                             StreamerVaryEnergyPriceAction)
        self.register_action('load-from-storage',
                             LoadFromStoragePostAction)

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

Streaming Service YANG
main.py, class ConnectedToSkylight

```
class ConnectedToSkylight(NanoService):
  @NanoService.create
  def cb_nano_create(self, tctx, root, service, plan,
                     component, state, proplist,
compproplist):
    vars = ncs.template.Variables()

    vars.add('SESSION_ID', str(uuid.uuid5(uuid.NAMESPACE_DNS,
             f'{service.name}-edge-connected-to-skylight')))

    # Find the DC with the lowest jitter
    best_dc = None
    self.log.info(f'Selection of DC with the lowest jitter
                    not yet implemented')
    if best_dc is None:
        raise Exception('No DC found')

    vars.add('DC', best_dc)
    service.oper_status.chosen_dc = best_dc.name

    template = ncs.template.Template(service)
    template.apply('edge-servicepoint-edge-
                    connected-to-skylight', vars)
```

ConnectedToSkylight. cb_nano_create()

This python method is called whenever NSO is trying to enter the connected-to-skylight state. Here, this method computes variable values, and applies a template

Agenda

We are here

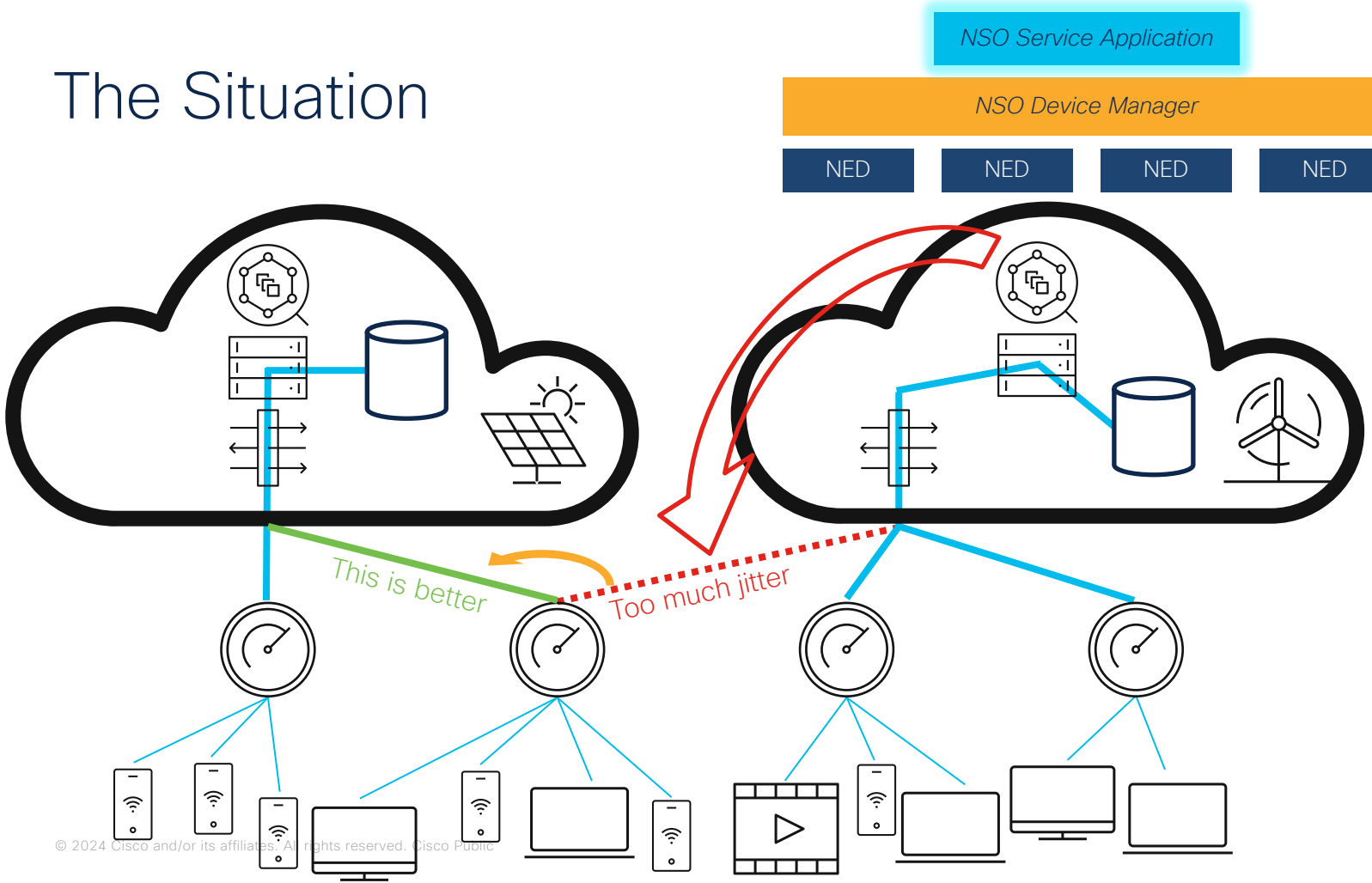
- ~~Automation levels~~
- Show the level 3 service
- Task 1: Go to level 4
  - Work
  - Discussion
- Task 2: Go to level 5
  - Work
  - Discussion
- Implications of level 4-5
- Conclusion

Agenda

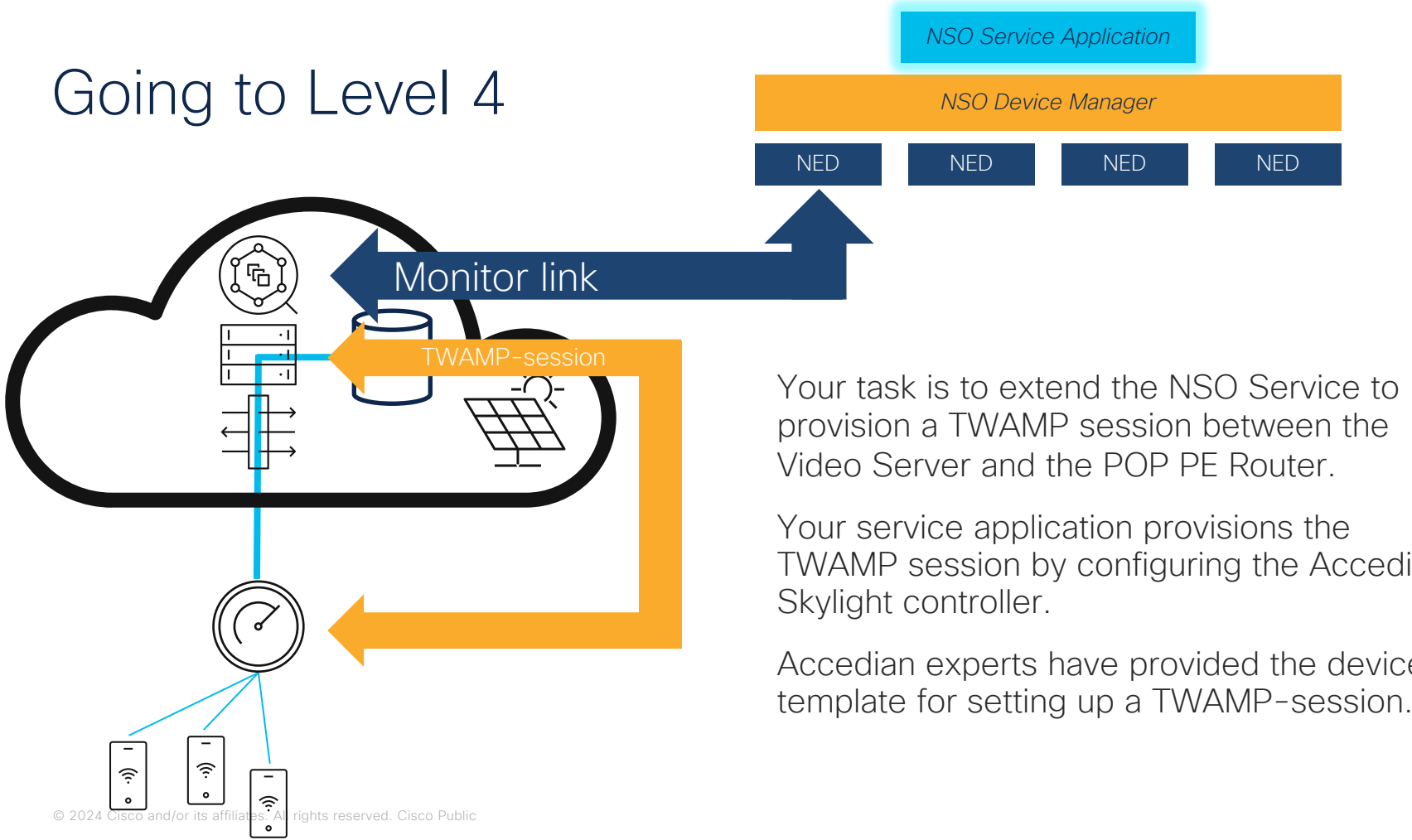
Go to Level 4

- Add device monitoring to the service
  - Ensure Accedian Skylight monitors origin video server and edge video cache
  - Move services that are experiencing trouble

The Situation



Going to Level 4

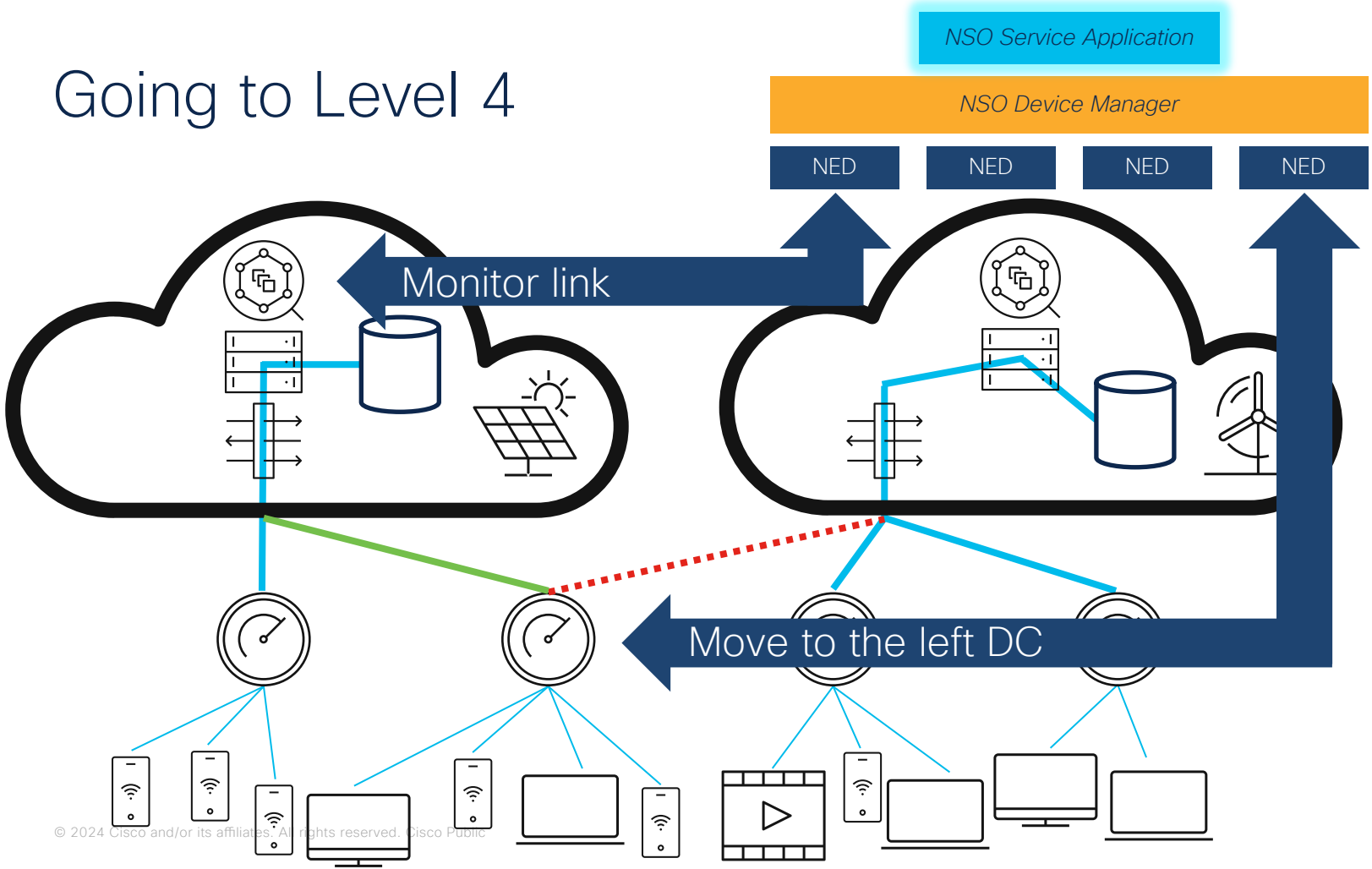


Your task is to extend the NSO Service to provision a TWAMP session between the Video Server and the POP PE Router.

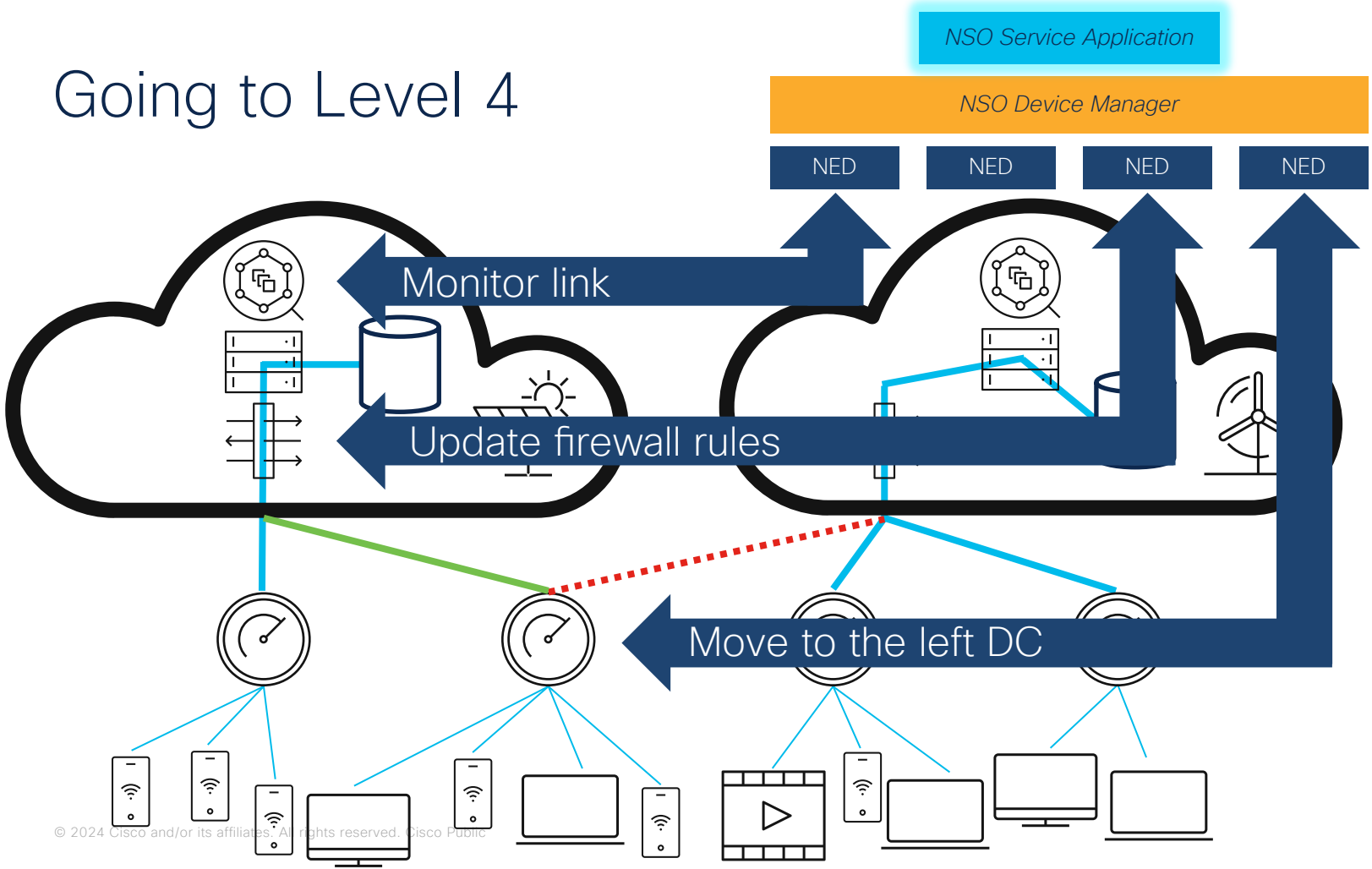
Your service application provisions the TWAMP session by configuring the Accedian Skylight controller.

Accedian experts have provided the device template for setting up a TWAMP-session.

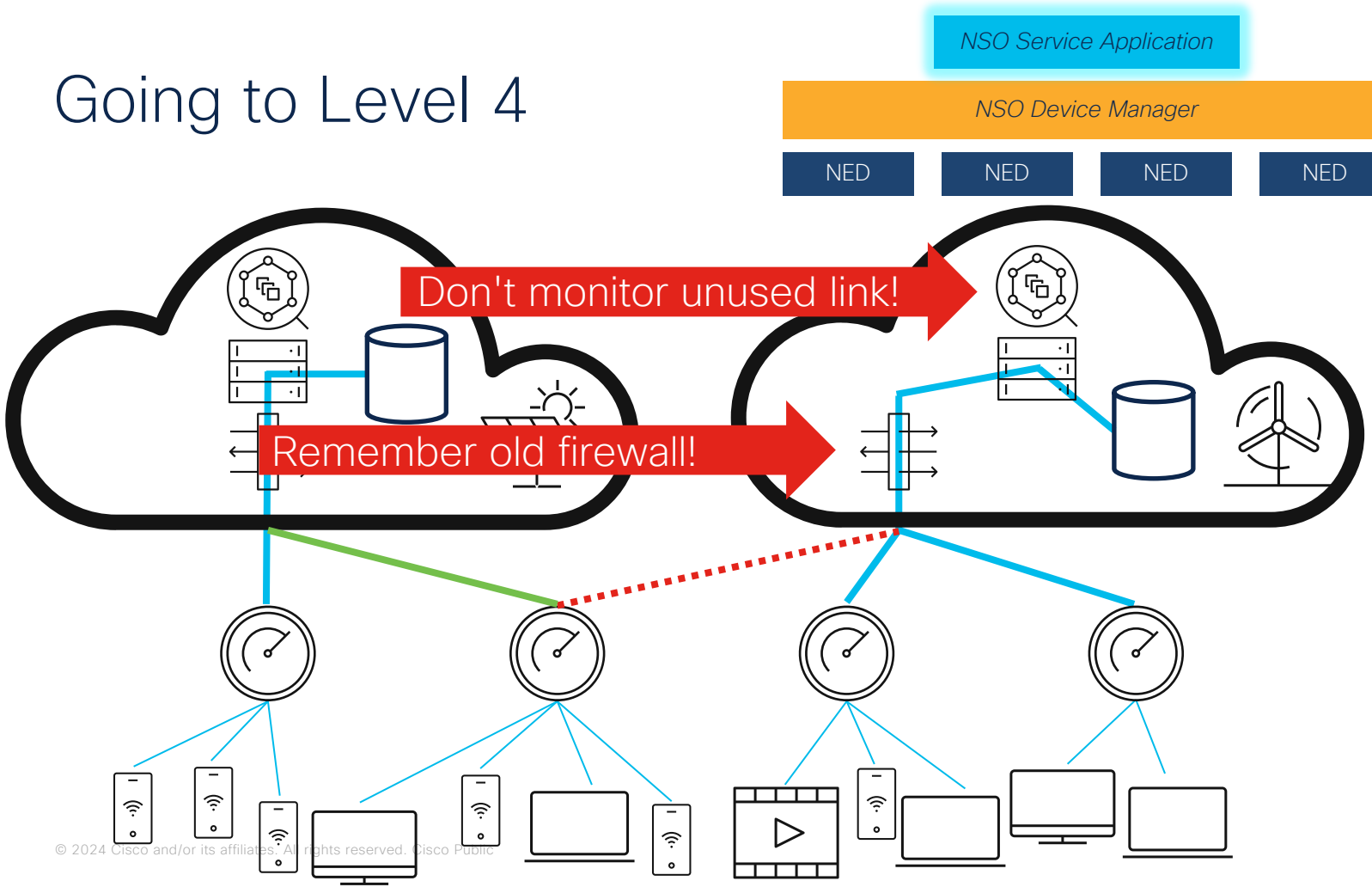
Going to Level 4



Going to Level 4



Going to Level 4



How to

1. Update the YANG

- Add a nano-callback to the plan DC `init` state
- Remove `is-active` from YANG model and plan
- Add leaf `oper-status/jitter` to DC oper model
- Remove `edge/dc`, add `edge/oper-status/chosen-dc`

2. Update the logic

- Add method `DCinit.cb_nano_create()` that writes which DC to use to `edge/oper-status/chosen-dc`
- Update reference to DC in two templates
- Make the `connected-to-skylight` state python + template, and add a notification-kicker

3. Test

- Make, packages reload, check package status
- Create a few edge service instances
- Simulate notifications about jitter variation with `rpc-send-notification-low-jitter` (or `-high-`)
- Observe `edge/oper-status` and log output in `ncs-python-vm-streaming.log`

Level 3 -> 4 changes

• src/yang/streaming-plan.yang	-7+5
• src/yang/streaming.yang	-4+22
• python/streaming/main.py	-10+30
• templates/edge-servicepoint-dc-fw-configured.xml	-1+1
• templates/edge-servicepoint-edge-connected-to-dc.xml	-1+1
• templates/edge-servicepoint-edge-connected-to-skylight.xml	-6+23
IN TOTAL	-29+82

Let's Discuss Level 4

Is this hard? What was the most difficult?

What happens if we put side-effects into service states?

When should service instances be re-deployed?

Which service instances should be re-deployed?

Always re-deploy services immediately?

What about re-deploy workload? Avalanches?
Service Flapping? Load Sharing?

Was this implementation good at network optimization?

Do we want to spread the load or focus the load?

Agenda

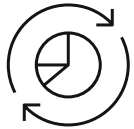
We are here

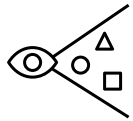
- ~~Automation levels~~
- Show the level 3 service
- Task 1: Go to level 4
  - Work
  - Discussion
- Task 2: Go to level 5
  - Work
  - Discussion
- Implications of level 4-5
- Conclusion

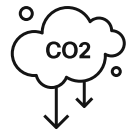
Agenda

Go to Level 5

- Constantly optimize the service expression
  - Create an optimization function that considers multiple factors
  - Keep moving load between data centres to optimize the overall service delivery







How to

1. Update the YANG

- No plan changes
- Add `energy-price` to DC oper model
- Add `edge-capacity` to DC config model

2. Update the logic

- Update optimization function to take jitter, energy price and DC capacity into account
- Complete optimization action
- Update skylight notification action to not optimize services

3. Test

- Create service instances
- Start varying electricity prices
- Start optimizer
- `show dc | repeat` to follow the action and watch optimization in `ncs-python-vm-streaming.log`

Level 4->5 changes

- src/yang/streaming.yang −0+16
- python/streaming/main.py −8+27
- python/streaming/keep_optimizing_action.py −4+5
- python/streaming/skylight_notification_action.py −10+0

IN TOTAL −22+48

Let's Discuss Level 5

Are we working on the right abstraction level?

If we want to tune the optimization function,
how much work is that?

Do you trust the NSO service manager to clean up
between the different service expressions?

Have you done something similar before?
If so, could you share your experience?

Is this useful in the real world?

Are you able to explain the difference between
network automation levels 3, 4 and 5?

Will you tell your colleagues about this?

The network has come alive. ☺ Do you agree?
In what ways is this like a form of life?

Agenda

We are here

- ~~Automation levels~~
- Show the level 3 service
- Task 1: Go to level 4
  - Work
  - Discussion
- Task 2: Go to level 5
  - Work
  - Discussion
- Implications of level 4-5
- Conclusion

Architectural take-aways, Automation Level 3-5

LEVEL 3

- The 8 Corner Stones of Network Automation
  - Service centric
  - Template based
  - Create-only
  - Declarative
  - Transactional
  - Stateful
  - Composable
  - Model driven

LEVEL 4

- Monitor all services
  - Monitoring is part of the service, not an add-on
  - Monitoring starts, stops and changes with the service
  - Monitoring may trigger automatic change, or provide data for holistic decisions

LEVEL 5

- Constantly optimize
  - Service working is one thing
  - Service working optimally is another
  - Services collectively working optimally is yet another

Conclusion

The bridge to possible

Coffee break