



DeveloperDays  
Network Services Orchestrator

# NSO Developer Days

A Low Code Approach to Brownfield Service Discovery

Dan Sullivan  
Cisco Technical Solutions Architect  
6/19/2019

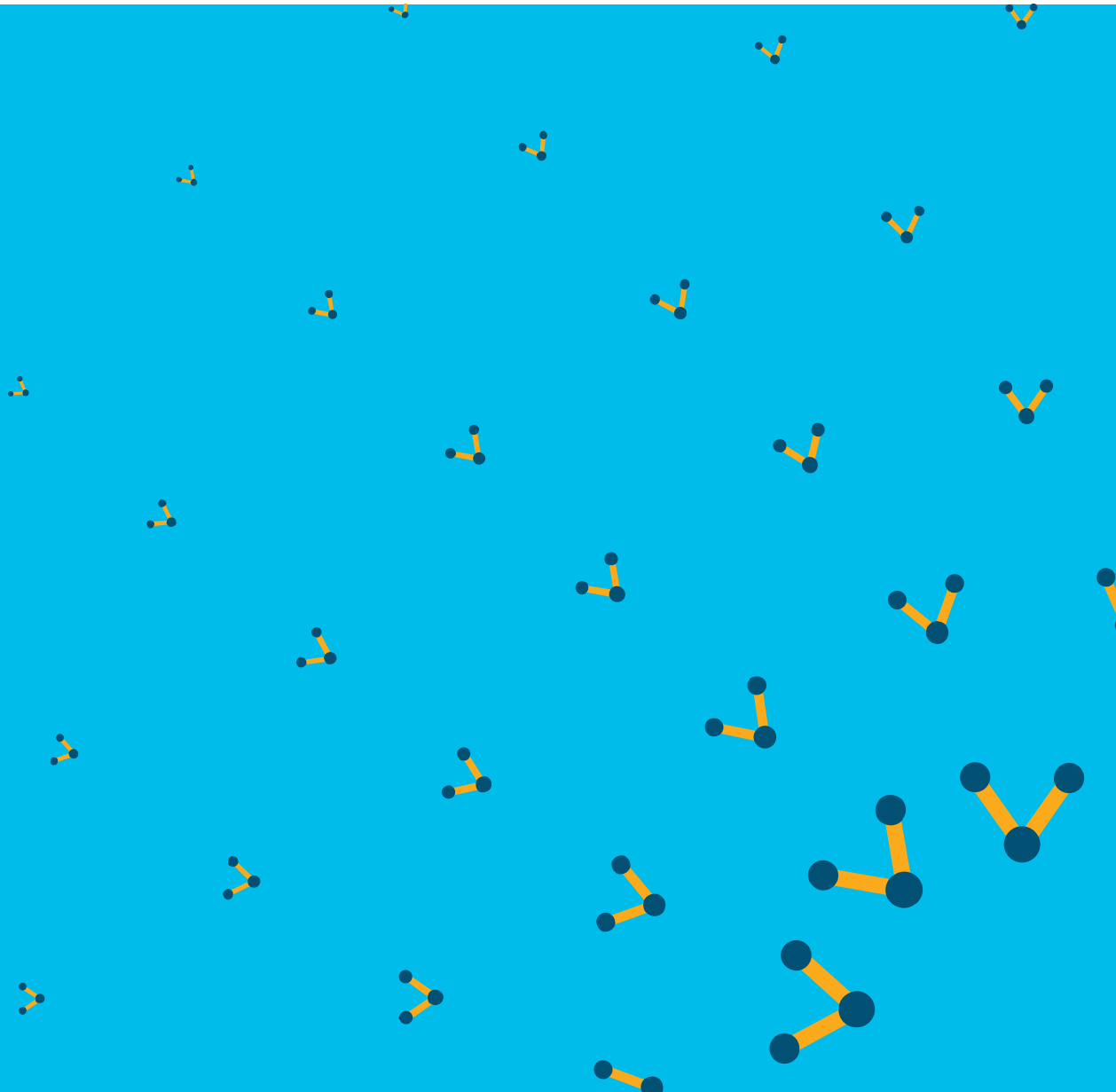
# Abstract

Developing service models and implementing those models in NSO is fairly straight forward process.

Brownfield service discovery or adding services provisioned outside of NSO and into NSO has its challenges.

The purpose of this presentation is to examine that process and provide a brief demonstration

# NSO Service Discovery



# NSO Brownfield Service Discovery

© 2019 Cisco and/or its affiliates. All rights reserved. Cisco Public

## High Level Steps:

- Define an NSO service model which can support the existing services
- Extract services from inventory system
- Compare services deployed in the network w/inventory
- Import new services into NSO
- Update inventory with actual service configurations

# NSO Brownfield Service Discovery

© 2019 Cisco and/or its affiliates. All rights reserved. Cisco Public

## Challenges:

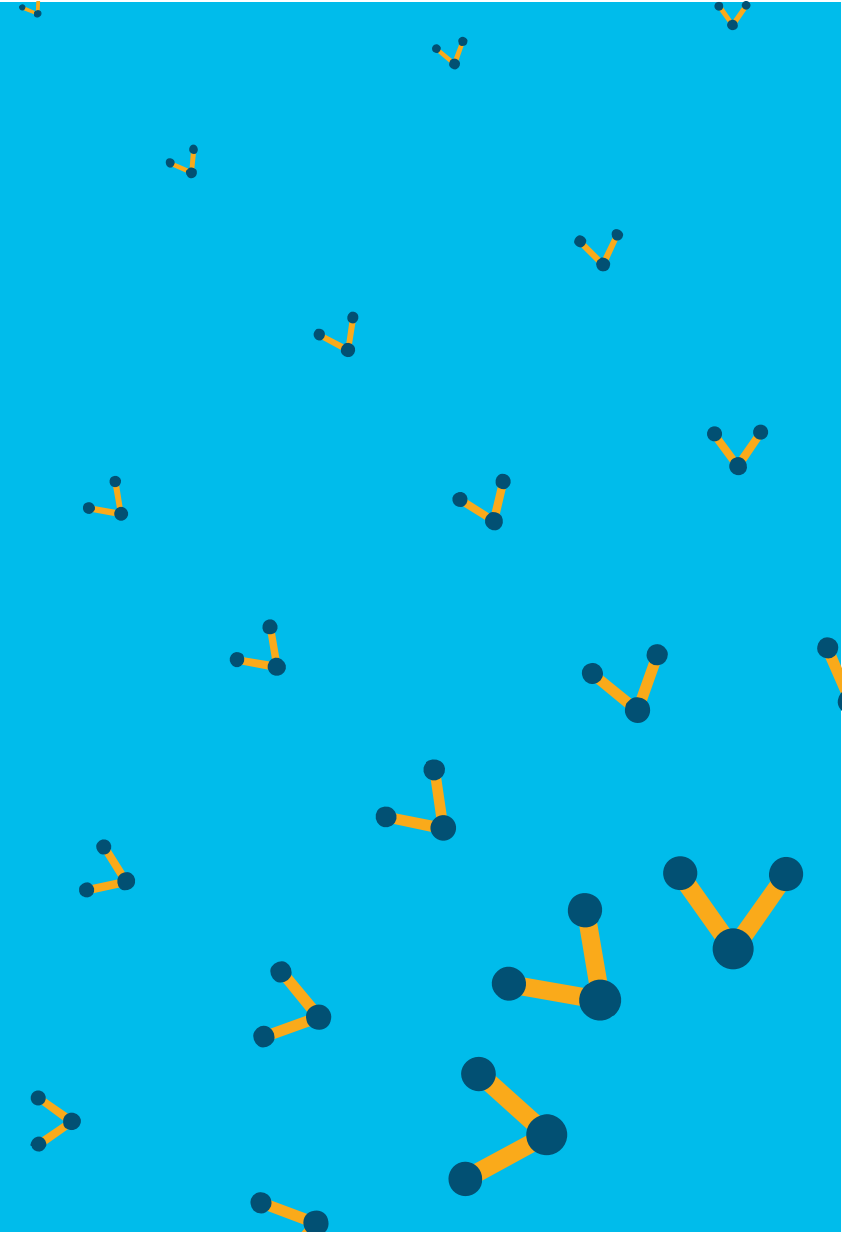
- Inventory doesn't always match what's deployed in the network
  - ❖ Inventory service intent differs from the network
  - ❖ Services missing from inventory
  - ❖ Services missing from the network
  - ❖ Service parameters differ from actual network configuration
- Reconciling differences is not always easy to do
- Process can be time consuming/iterative

# NSO Brownfield Service Discovery

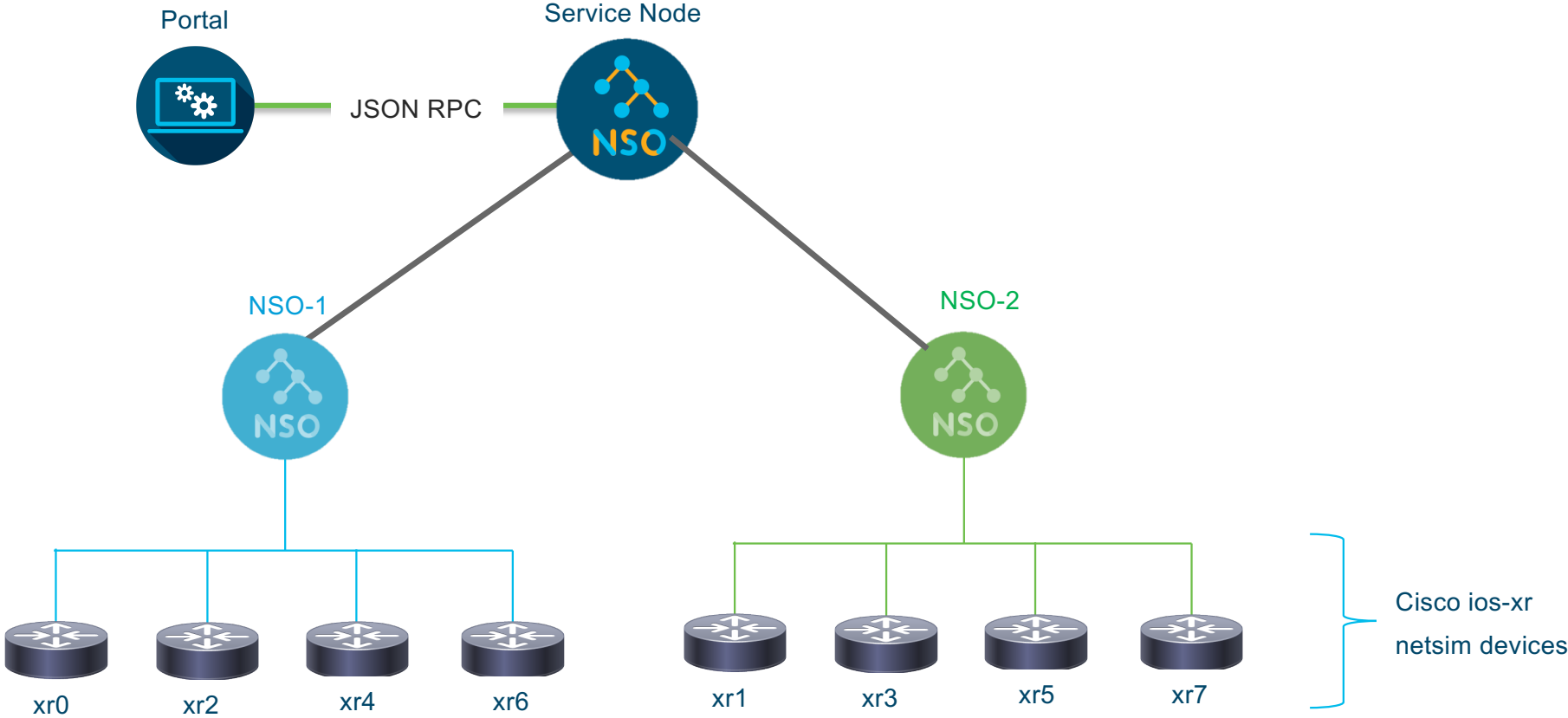
## Solution:

- Low code approach
- Need a solution that can be developed quickly
- Utilize NSO service template(s) for the "heavy lifting"
- Provide a service discovery process that onboards service(s)
- True up inventory w/Network and NSO if possible

# Demonstration



# Service Discovery Demonstration Architecture





## What does the demo actually accomplish?

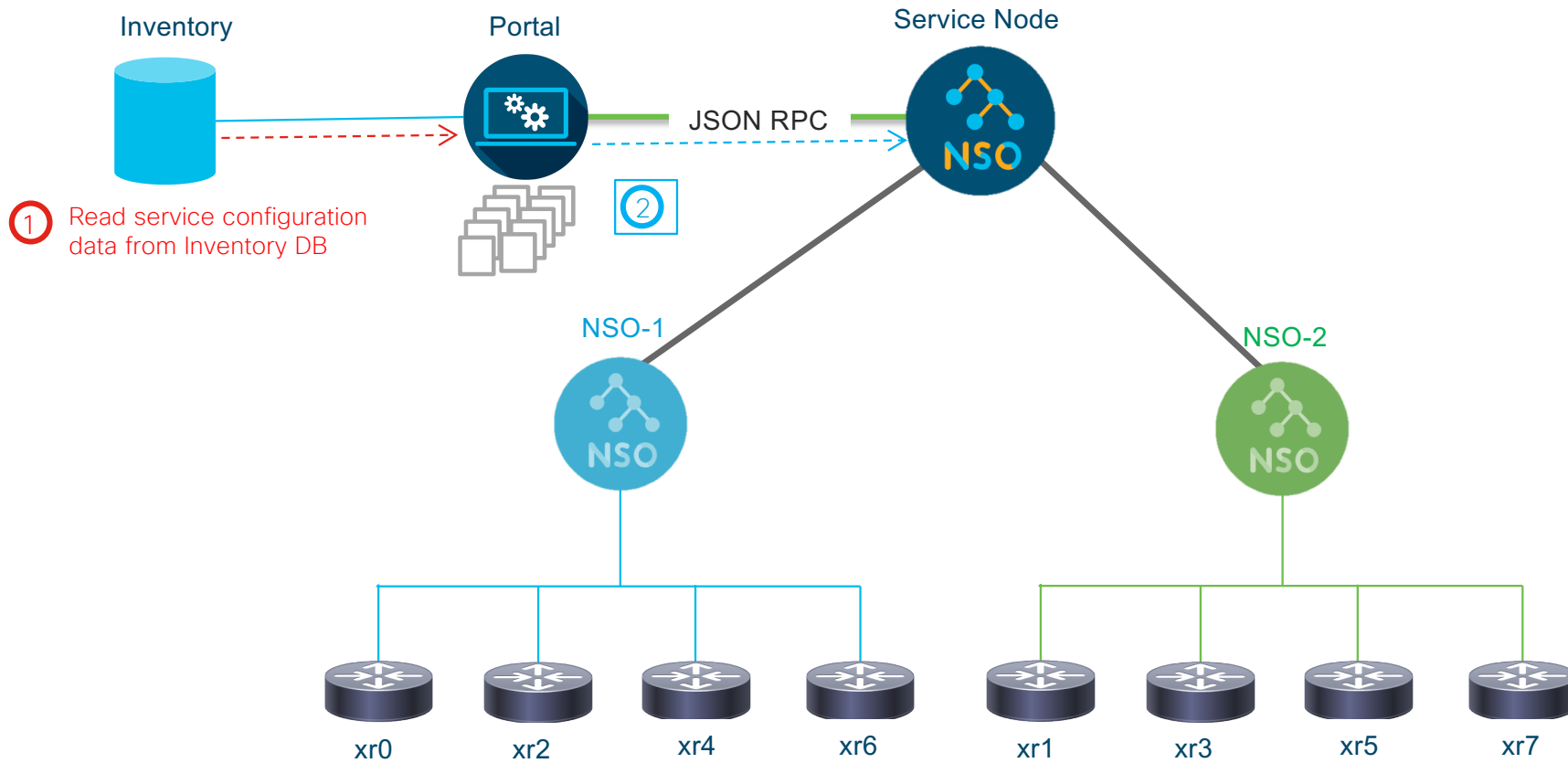
- Network devices pre-configured with a of VRF service data for a ~150 instances
- Demo will then read service instance data from an Excel spreadsheet and attempt to reconcile the inventory service data with the actual network configuration(s)
- Goal of the reconcile process is to handle various challenges:
  - ✓ Service(s) exist in the inventory but not the network
  - ✓ Service(s) exist in the network but not the inventory
  - ✓ Service parameter(s) don't match between inventory and network
- Reconcile all of these changes into NSO (and possibly the network)

# How does the demo actually work?

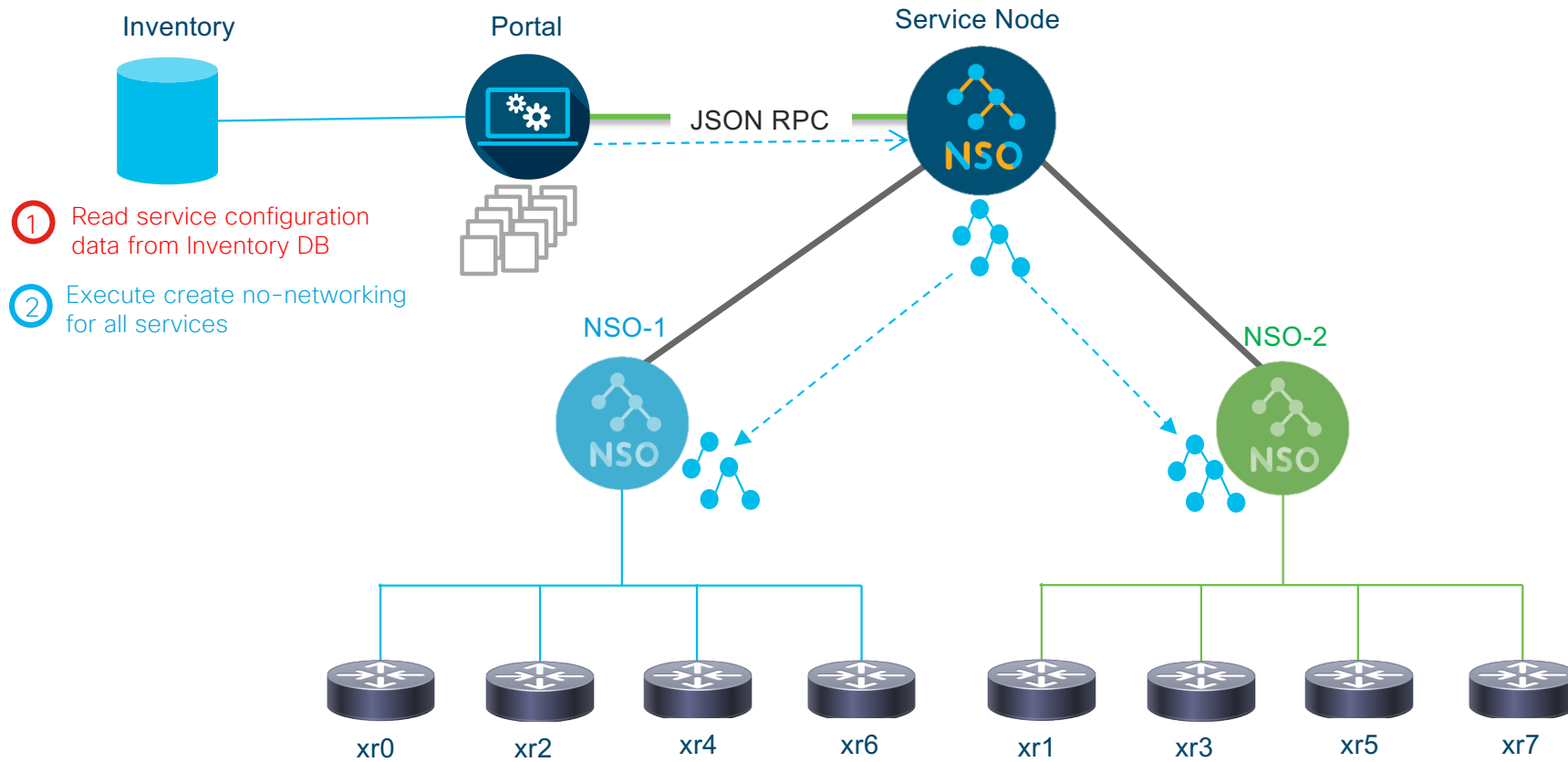
Using Portal GUI Application via JSON RPC accomplish the following:

- Read inventory data from inventory database (Excel)
- Create NSO service instances based on the inventory data
- Attempt to onboard the service(s) into NSO re-deploy reconcile
- For failing service instances attempt to reconcile service intent with NSO using oob-reconcile action
- Discover any service instances which exist in the network but not the inventory
- Write updated service data back to inventory database (Excel)

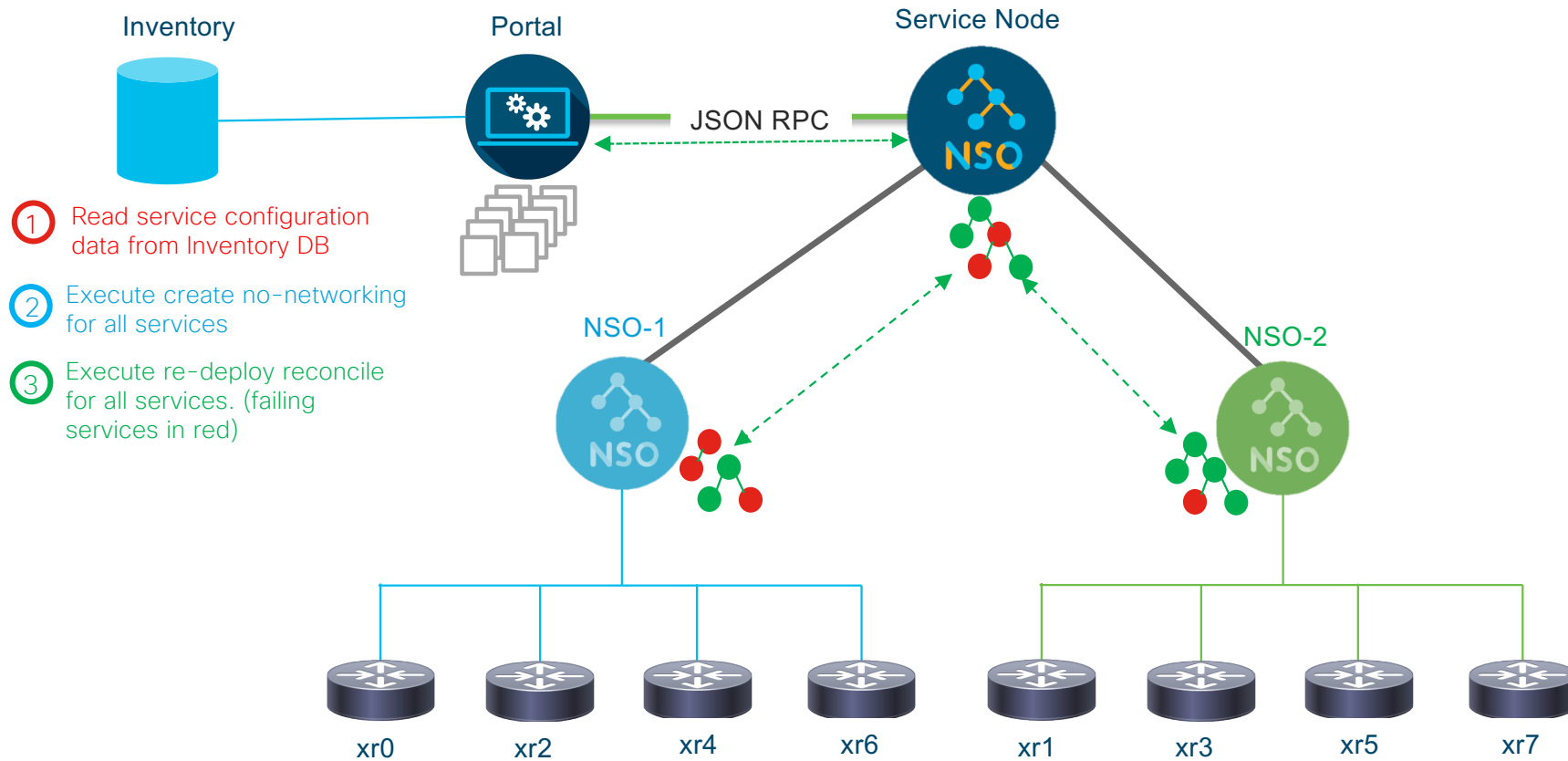
# Demonstration Architecture



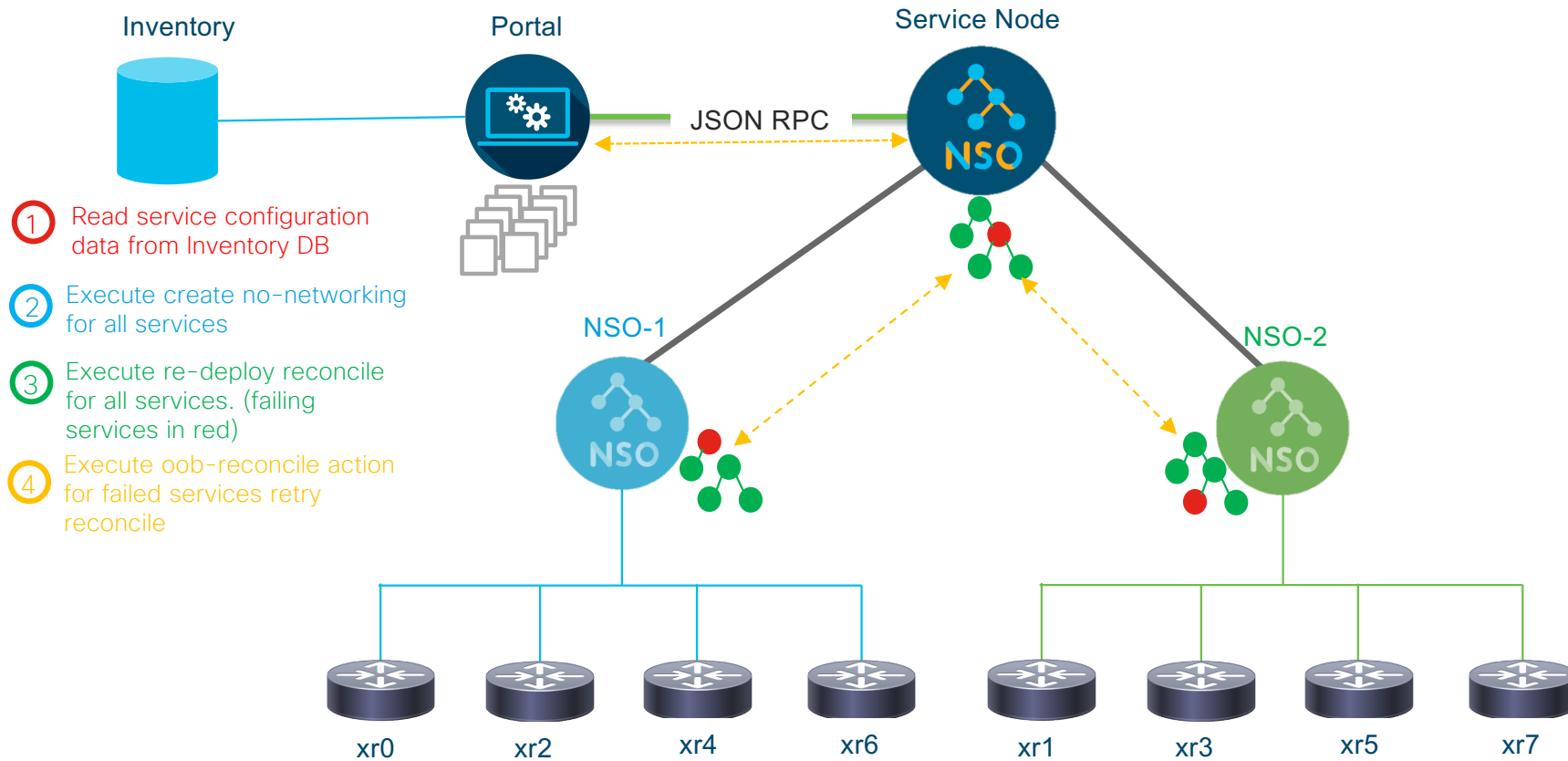
# Demonstration Architecture



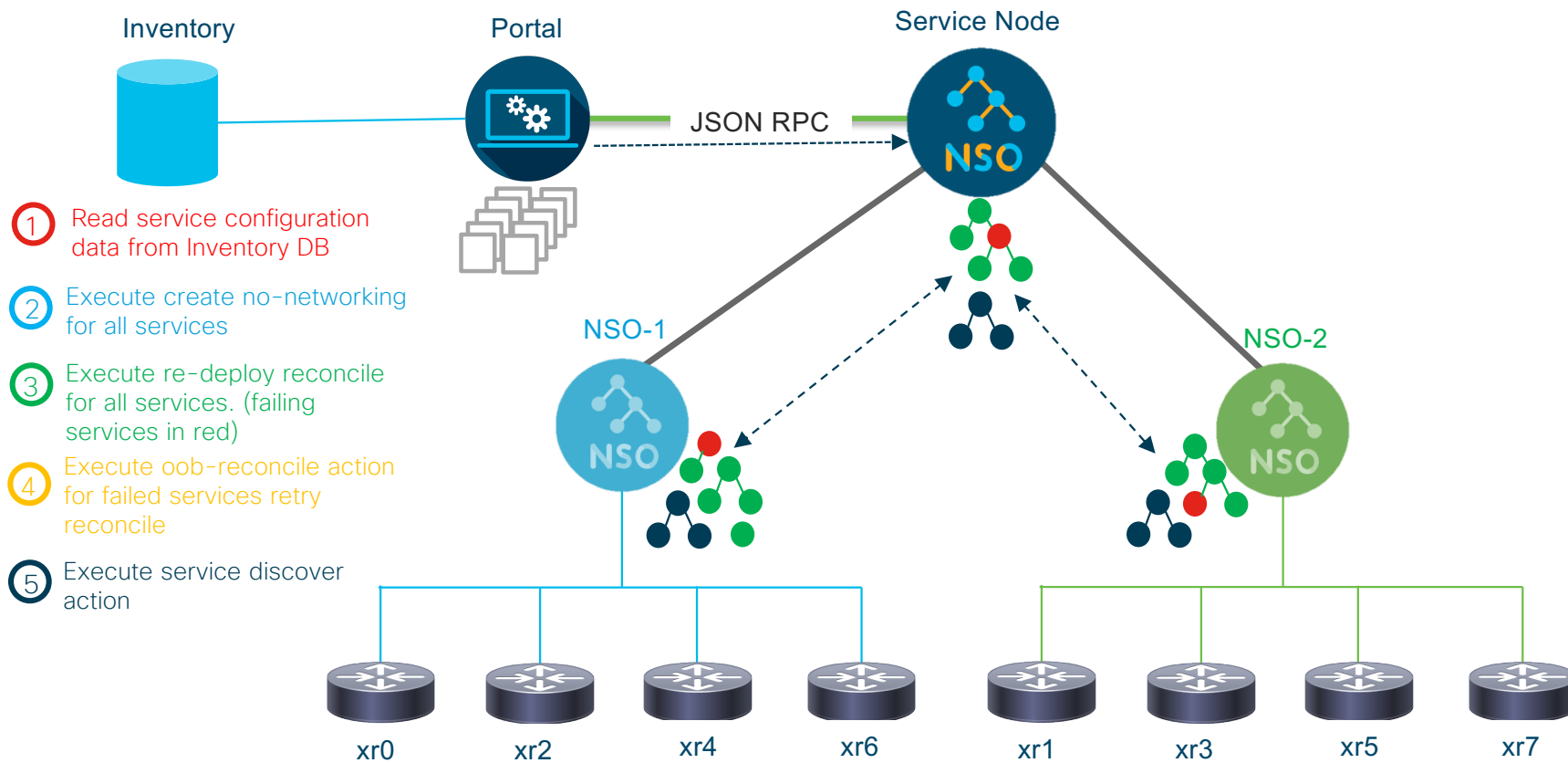
# Demonstration Architecture



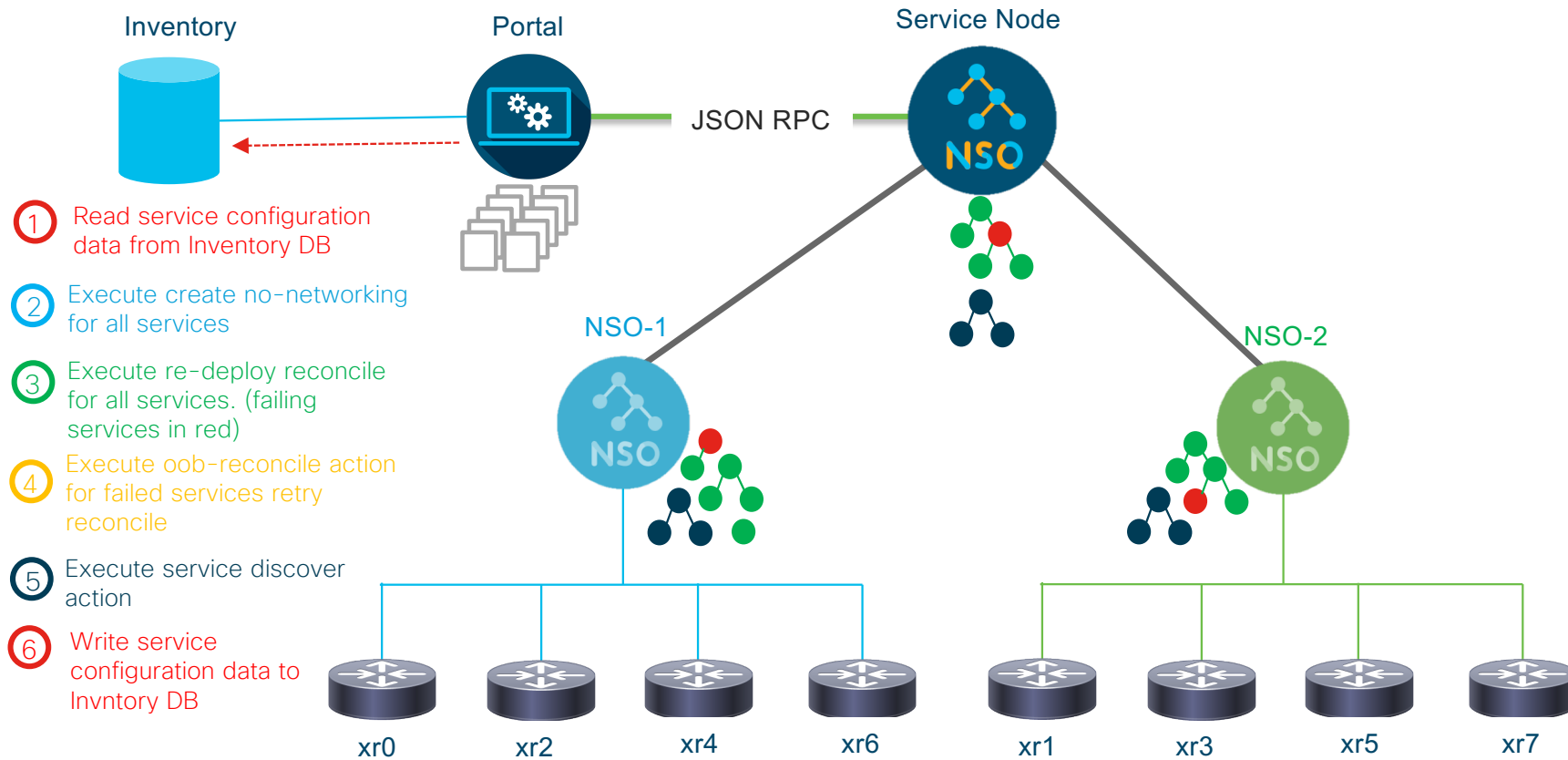
# Demonstration Architecture



# Demonstration Architecture

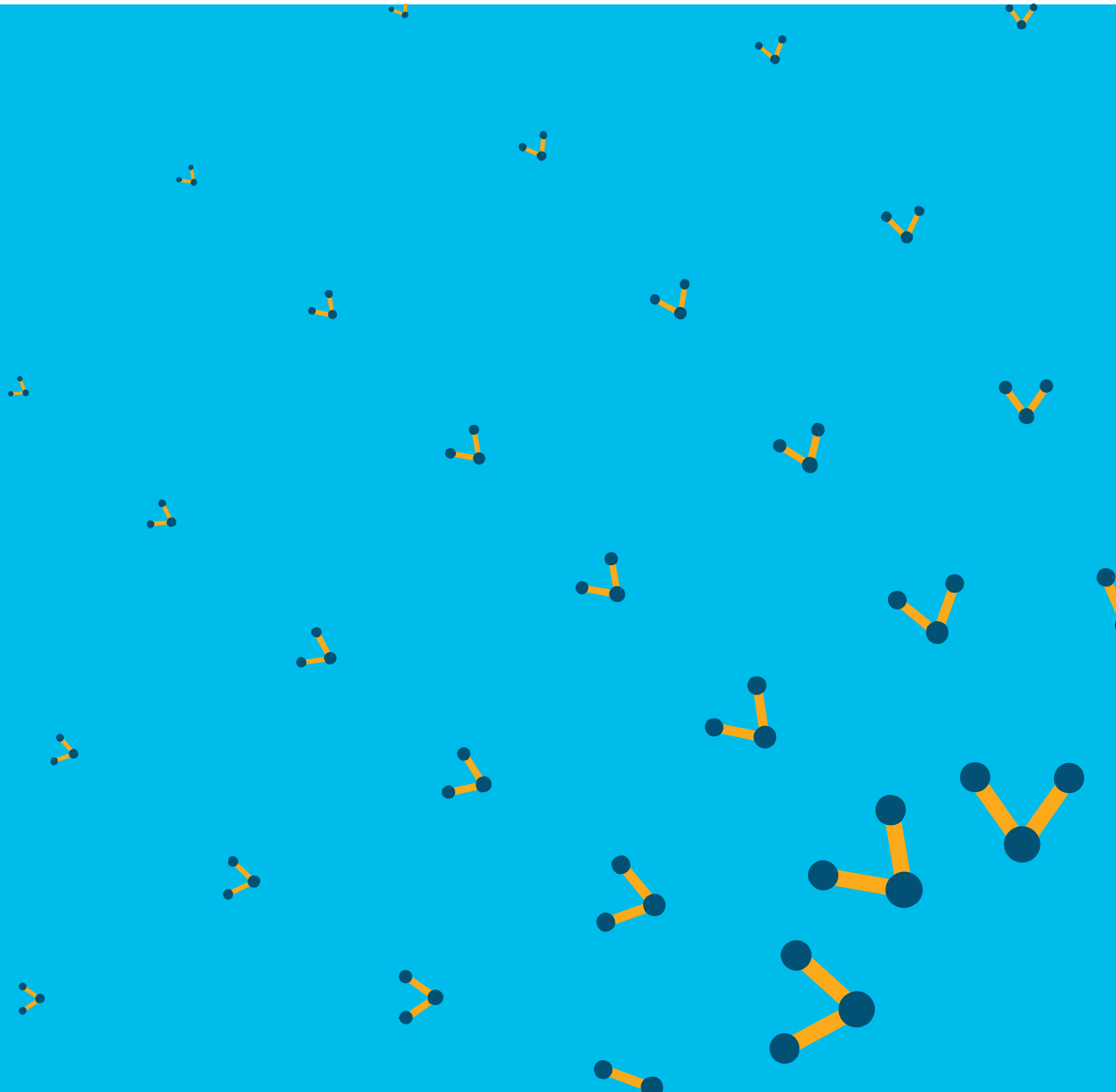


# Demonstration Architecture





Demo Time



# Cisco VRF Service

## Service Description:

- LSA Service
- Configures VRF instances
- Service can create VRF on multiple devices across LSA device nodes
- Action created for reconciling out-of-band changes
- Service discovery action created for discovering “unknown” service instances

# VRF Service Model

```
list vrf {
  description "VRF Service";
  key name;
  leaf name {
    tailf:info "Unique service id";
    tailf:cli-allow-range;
    type string;
  }
  uses ncs:service-data;
  ncs:servicepoint vrf-servicepoint;

  leaf description {
    type string;
  }
  list devices {
    key name;

    leaf name { type string;}
    leaf route-distinguisher { type string; }
    list import-route-target {
      key name;
      leaf name {
        type string;
      }
    }
    list export-route-target {
      key name;
      leaf name {
        type string;
      }
    }
  }
}
```

© 2019 Cisco and/or its affiliates. All rights reserved. Cisco Public

```
leaf import-route-policy {
  type string;
}
leaf export-route-policy {
  type string;
}
leaf max-prefix-limit {
  type uint16;
}
leaf max-prefix-threshold {
  type uint16;
}
}
////////////////////////////////////
// Service oob-reconcile action
////////////////////////////////////
tailf:action oob-reconcile {
  tailf:actionpoint vrf-reconcile-point;
  input {
    leaf attach {
      type empty;
    }
  }
  output {
    leaf status {
      type string;
    }
    leaf error-message {
      type string;
    }
  }
}
}
```

# VRF Service oob-reconcile Action (device node)

- Reconcile network configuration to service configuration
- Template does the opposite of a “normal” service template
- Template goes from device configuration => service configuration
- Template itself is 24 lines or so
- Service node initiates the reconcile action to the device nodes

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <? set VRF = {name}?>
  <vrf xmlns="http://example.com/vrf">
    <name>{$VRF}</name>
    <?foreach {devices}?>
      <?set DEVICE = {name}?>
      <devices>
        <name>{name}</name>
        <?save-context vrf-service?>
        <import-route-target tags="delete">
          <name>{import-route-target}</name>
        </import-route-target>
        <export-route-target tags="delete">
          <name>{export-route-target}</name>
        </export-route-target>
        <?switch-context vrf-service?>
        <?set-context-node {../../ncs:devices/ncs:device[name=$DEVICE]/config/vrf/vrf-list[name=$VRF]}?>
        <route-distinguisher>rd</route-distinguisher>
        <import-route-target tags="replace">
          <name>{address-family/ipv4/unicast/import/route-target/address-list}</name>
        </import-route-target>
        <export-route-target tags="replace">
          <name>{address-family/ipv4/unicast/export/route-target/address-list}</name>
        </export-route-target>
        <import-route-policy>{address-family/ipv4/unicast/import/route-policy}</import-route-policy>
        <export-route-policy>{address-family/ipv4/unicast/export/route-policy}</export-route-policy>
        <max-prefix-limit>{address-family/ipv4/unicast/maximum/prefix/limit}</max-prefix-limit>
        <max-prefix-threshold>{address-family/ipv4/unicast/maximum/prefix/mid-thresh}</max-prefix-threshold>
        <?switch-context vrf-service?>
      </devices>
    <?end?>
  </vrf>
</config-template>
```

## VRF Service oob-reconcile Action (service node)

- Service node reconcile template copies device node service information to service node configuration
- Template does the opposite of the “normal” service template
- Template goes from device node configuration => service node configuration
- Template itself is 24 lines or so

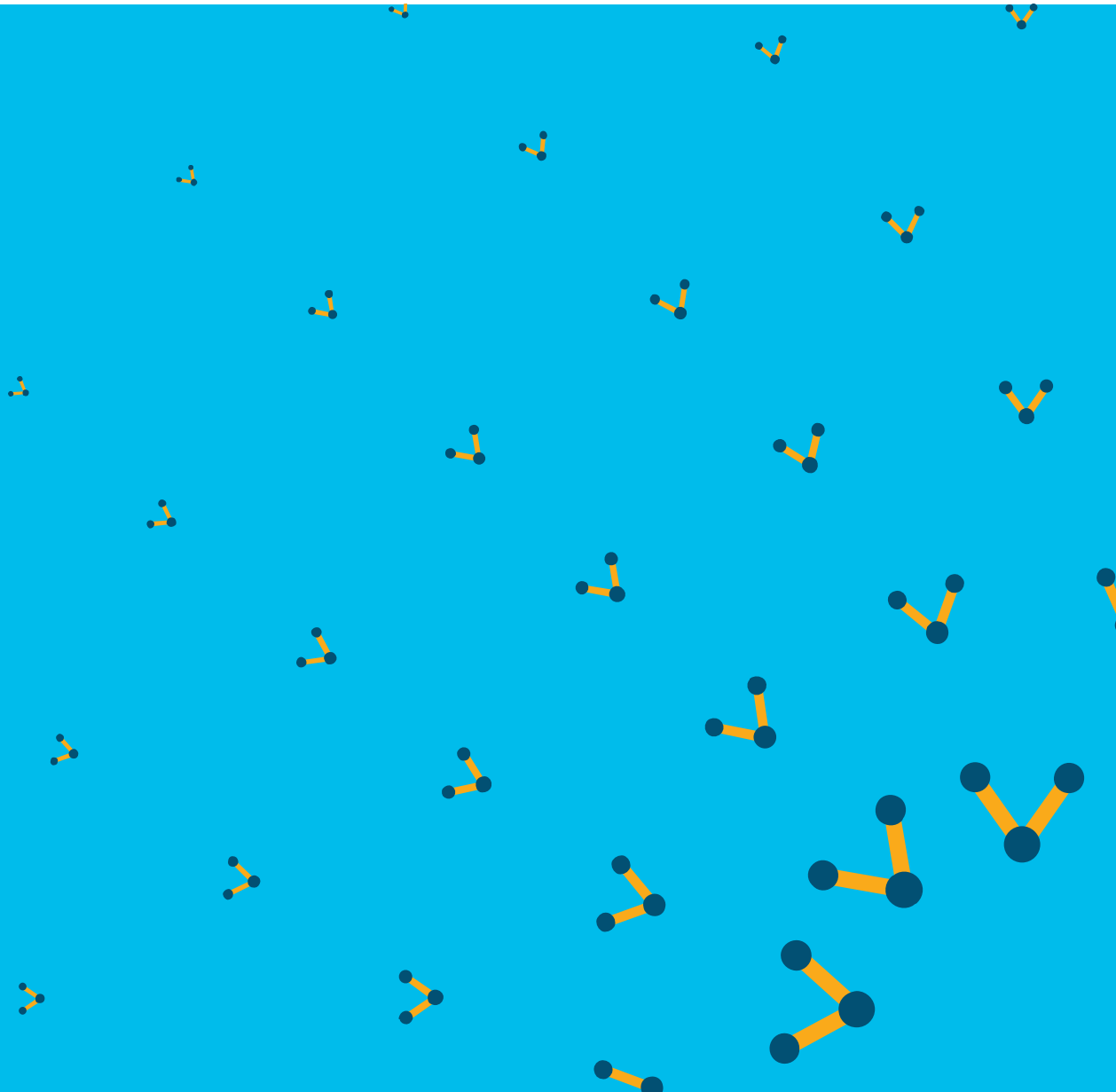
```
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <services xmlns="http://tail-f.com/ns/ncs">
    <vrf xmlns="http://example.com/svrf">
      <name>{$VRF}</name>
      <?foreach {devices}?>
        <devices>
          <name>{name}</name>
          <import-route-target tags="replace">
            <name>{import-route-target}</name>
          </import-route-target>
          <export-route-target tags="replace">
            <name>{export-route-target}</name>
          </export-route-target>
          <import-route-policy>{import-route-policy}</import-route-policy>
          <export-route-policy>{export-route-policy}</export-route-policy>
          <max-prefix-limit>{max-prefix-limit}</max-prefix-limit>
          <max-prefix-threshold>{max-prefix-threshold}</max-prefix-threshold>
          <route-distinguisher>{route-distinguisher}</route-distinguisher>
        </devices>
      <?end?>
    </vrf>
  </services>
</config-template>
```

# VRF Service Discover Action

```
////////////////////////////////////  
// Service vrf-discover action  
////////////////////////////////////  
container cmd {  
  tailf:action vrf-discover {  
    tailf:actionpoint vrf-discover-point;  
    input {  
      leaf-list devices {  
        type string;  
      }  
    }  
    output {  
      leaf status {  
        type string;  
      }  
      leaf error_message {  
        type string;  
      }  
      leaf-list vrfs {  
        type string;  
      }  
    }  
  }  
}
```

```
<config-template xmlns="http://tail-f.com/ns/config/1.0">  
<? set VRF = {$DISCVRF}?>  
  <vrf xmlns="http://example.com/vrf">  
    <name>{$DISCVRF}</name>  
    <?save-context vrf-service?>  
    <?set-context-node {/ncs:device[name=$DEVICE]/config/vrf/vrf-  
list[name=$VRF]}?>  
    <description>{description}</description>  
    <devices>  
      <name>{$DEVICE}</name>  
      <?save-context vrf-service?>  
      <route-distinguisher>{rd}</route-distinguisher>  
      <import-route-target>  
        <name>{address-family/ipv4/unicast/import/route-target/address-  
list}</name>  
      </import-route-target>  
      <export-route-target>  
        <name>{address-family/ipv4/unicast/export/route-target/address-  
list}</name>  
      </export-route-target>  
      <import-route-policy>{address-family/ipv4/unicast/import/route-  
policy}</import-route-policy>  
      <export-route-policy>{address-family/ipv4/unicast/export/route-  
policy}</export-route-policy>  
      <max-prefix-limit>{address-  
family/ipv4/unicast/maximum/prefix/limit}</max-prefix-limit>  
      <max-prefix-threshold>{address-  
family/ipv4/unicast/maximum/prefix/mid-thresh}</max-prefix-threshold>  
      <?switch-context vrf-service?>  
    </devices>  
  </vrf>  
</config-template>
```

Questions?



Thanks for listening

