Design and experiences

# Automation for brown field networks

— Allan Eising

— DKNOG12

# Agenda

— Designing scalable automation

— Integrating scalable automation into existing networks

# About me

— 15+ years of operations and development experience in business service provider networks
— 15+ years of dreaming of a ”Provision” button
— Working as an architect in the network automation realm at Telia Company
— Living and working in Norway for 5 years
— Started this thing called DKNOG
— https://automate.network
— @allaneising on twitter

Green field

Brown field

# Goal

— Replacing the full system stack after a merger

— Writing new network automation but import old services

— Ensure migrated services are produced as close as possible to their old counterparts

— Platform: Cisco NSO

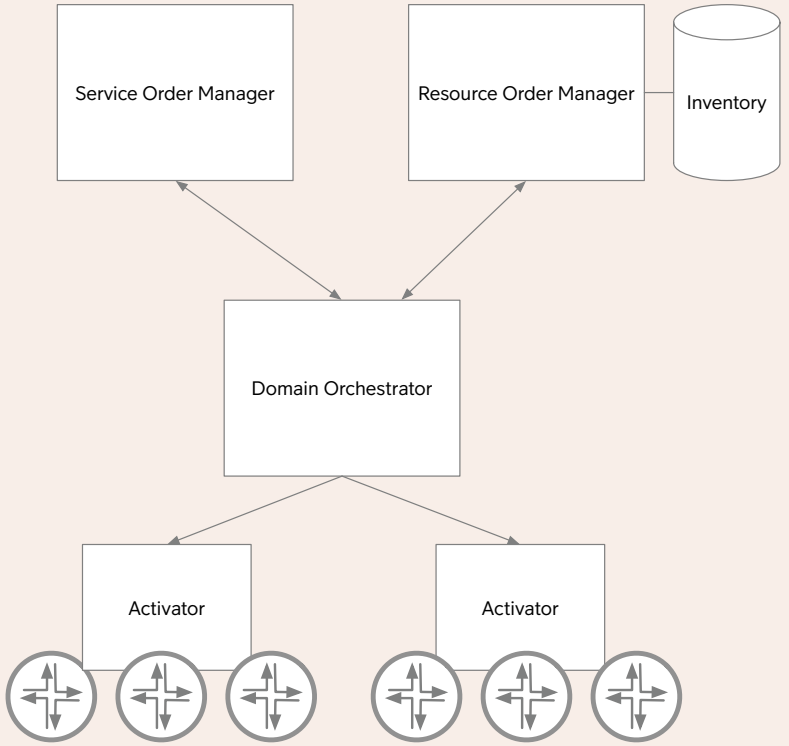
# Service target

## B2B services are not trivial to automate

- High degree of bespoke
- Complex solutions
- Interworking between different technology generation and platforms
- Blurred lines between transport parameters and service properties
- Documentation often lacking

# Desired architecture

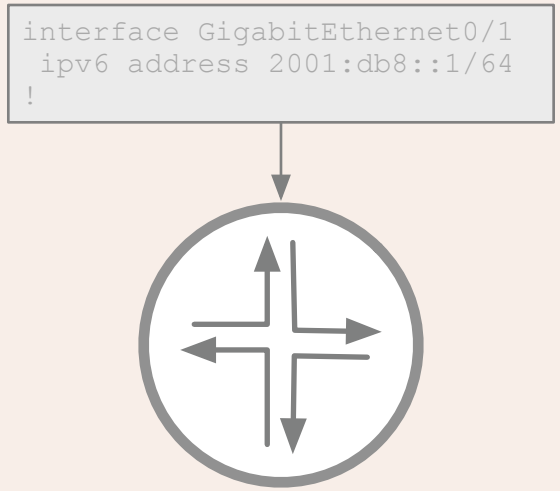
— Ideal architecture
— Less can do fine
— Not in picture: BSS, sales, assurance, etc.



# *Intent-based* Orchestration

# Bottoms up!

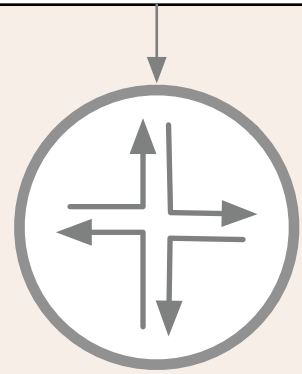
— Let’s imagine we want to configure an internet service



# Bottoms up!

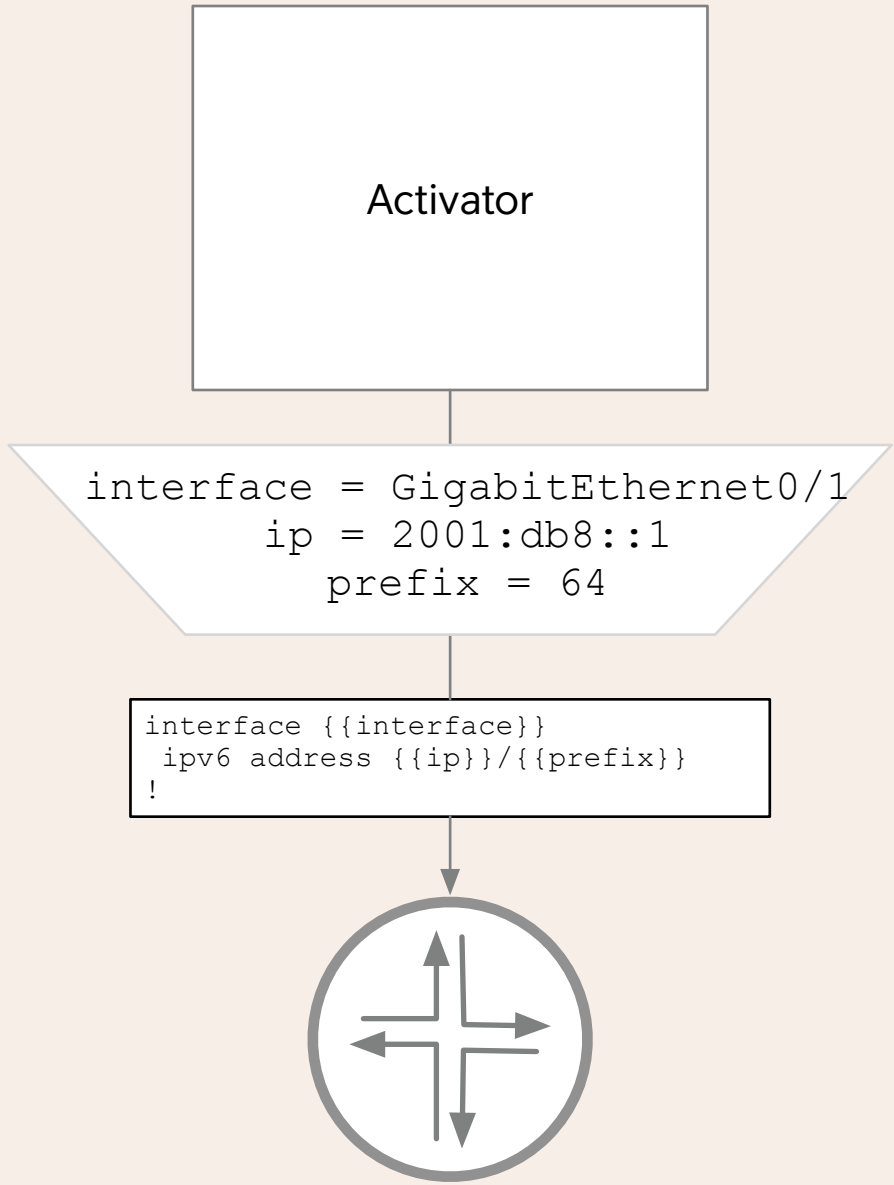
— Let’s imagine we want to configure an internet service
— Usually we do this by some sort of template

```
interface {{interface}}
 ipv6 address {{ip}}/{{prefix}}
!
```



# Bottoms up!

— Let’s imagine we want to configure an internet service
— Usually we do this by some sort of template
— An activator is responsible for filling out the template



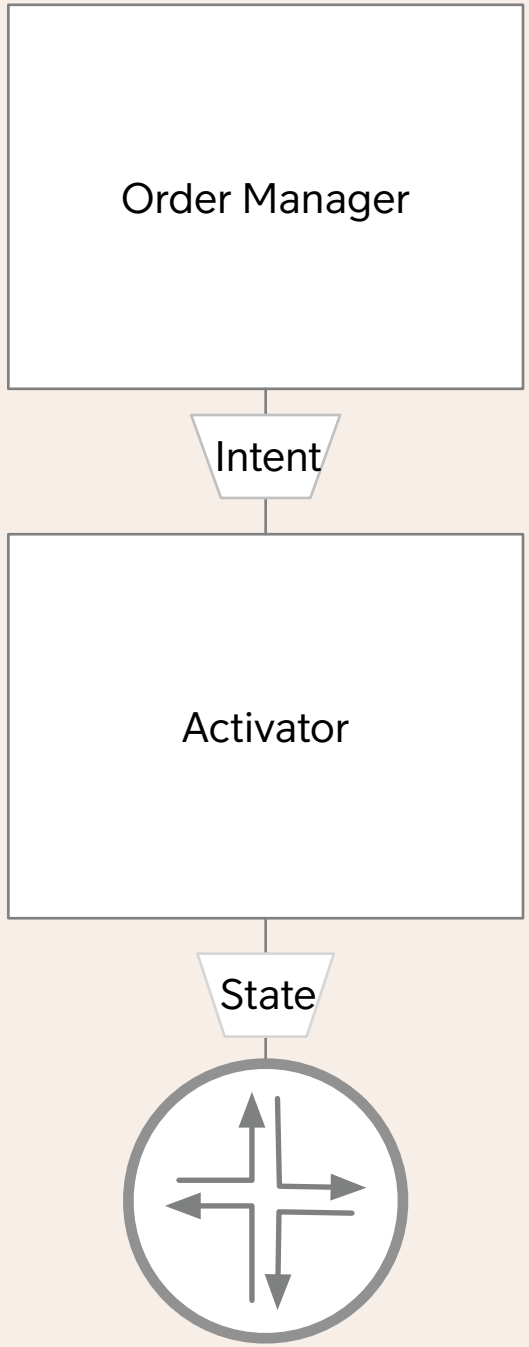
# Public API

— What happens if the customer has to move?
— Is it still the same service?

```
{
    "service_id": 1,
    "parameters": {
        "device": "PE01",
        "interface": "GigabitEthernet0/1",
        "ip": "2001:db8::1",
        "prefix": 64
    }
}
```

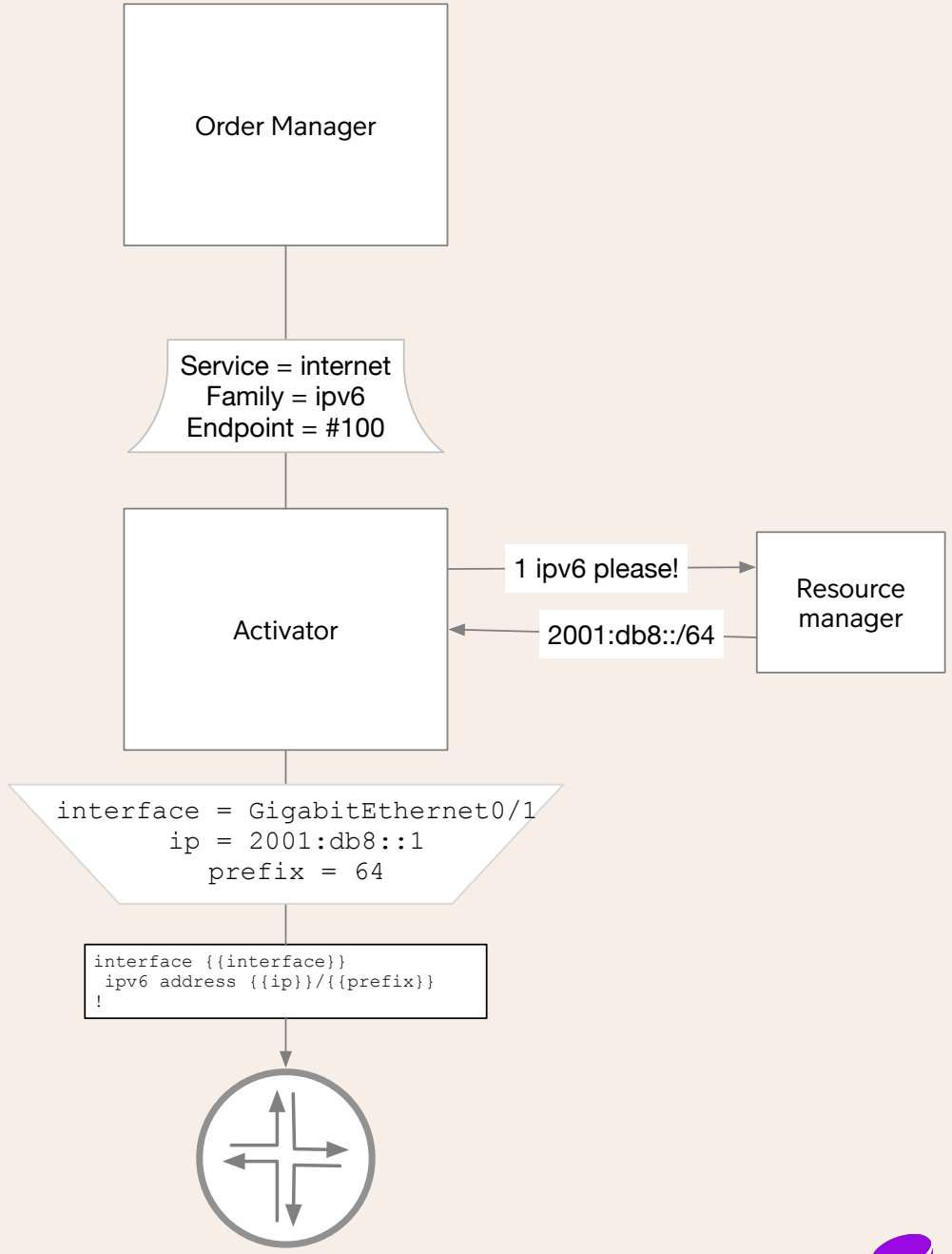
# Intent vs state

- The northbound systems communicate *intent* to the activator
  - "I want an internet service with IPv6"
- The southbound systems manage *state* in the network
  - "IPv6 address 2001:db8::1/64 should be present on interface GigabitEthernet 0/1 on node X"



# Resolving state

- The order manager is oblivious to technical implementation
- The Activator implements an *endpoint* oriented API
- The *endpoint* is a pointer to a logical interface
- The Activator decomposes the service and resolves the needed resources
- The activator requests the needed resources from the Resource Manager (ROM)



# Service modeling

- We have modeled three different services:
  - Business internet/transit
  - L3VPN
  - L2VPN point-to-point
- To the right is a simplified example of a L3VPN service
- We use YANG to model, hence the hierarchy
- This model is further enhanced with overrides for various default/inherited parameters

```
l3vpn VRF12345
   access 10000
     bandwidth
       up 100m
       down 100m
     subnet 10
       ipv4
         192.0.2.0/31
       ipv6
         2001:db8::/64
     endpoint 642a0e17-749c-45de-a010-5039f558f62e
       subnet 10
     routing
       static
         198.18.0.0/24
            next-hop 192.0.2.1
```

# Truth

It cannot be understated how impossibly hard it is to perform clean data migration

# Goal

We need to be able to recreate all our services just from our documentation

# Pick an approach

— Exact output

*Or*

— Functional equivalence

# Exact output

— Our starting point
— Replicating existing service configuration 1:1
— Success rate can be measured
— Requires a consistent foundation
— May not be automation friendly
— Takes longer time

# Analytics

— Using dry-runs of the service logic, we loaded the entire service fleet into a pre-production environment.

— We scored the line of output for each service into three categories:

1. In Sync
2. Safe changes
3. Unsafe changes

# Safe and unsafe changes

- Safe changes are changes that have no negative impact on the service intent
- This includes things like *interface descriptions* not matching
- Also includes examples where the network was incorrectly provisioned

- Unsafe changes are everything that requires human verification
- The could be due to incorrect documentation
  - Incorrect IP addresses
  - VLAN mismatches
  - Shaping rates not matching
  - Etc.
- This also uncovers bugs in the automation.

# Functional equivalence

— Best done after go-live
— At some point you are forced to simplify
— This breaks your ability to rely on the dry-run reports as a KPI

# Human or automation friendly?

— Avoid things like apply-groups and free-form comments in service configuration!
— Keep service configuration as self-contained as possible
— Unsafe inheritance between services
— Concepts that made manual configuration easier are often counter-productive when automating.

# Realization challenges

— Various differences between cisco platforms are extremely frustrating
— Table on the right shows various levels of flow-control support on some Cisco Switches
— Solved with various capability flags to signal special behaviors

| Platform | RX | TX |
|---|---|---|
| WS-X4516-10GE | supported | supported |
| WS-X4248-FE-SFP | Not supported | Not supported |
| WS-X4306-GB | Supported | Supported, default off |
| ME-3400G-12CS-D | Support, default off | Unsupported |
| WS-C3550-24-SMI | Supported, default off | FastEthernet: unsupported GigabitEthernet: supported |
| ME-4924-10GE | Supported | Supported, default off |

# Go live hypercare

## Be prepared to deliver extra support for up to half a year after go-live

# Wrapping up

I have many warstories best taken over a cold beer

# Let’s take questions instead!

# Thank you!