Consortium GARR

From Clicks to Code

Optical Network Automation Journey at GARR

by Matteo Colantonio

ITNOG

01

What didn't work and why

02

The Workflow Orchestrator framework

03

Our implementation without config managers

01

What didn't work and why

Exp.#1: Ansible

We did upgrade 92 transponders on schedule with AWX but…

	3526	Status	Successful	Started	3/20/2024, 6:45:43 AM
Finished	3/20/2024, 8:12:20 AM	Job Template	G30 Upgrade Template	Job Type	Playbook Run
Launched By	G30 Upgrade Scheduler	Inventory	G30 Inventory	Project	G30 Upgrade Project
Revision	cf4lblbaea7eef7ab420d04ce9b6f49dabc5d2f8	Playbook	playbook.yml	Verbosity	0 (Normal)
Execution Environment	GARR EE 3.0.1	Controller Node	awx-ba1-task-d7658bdcb-d72m		
Job Slice	0/1	Forks	0		
Created	3/20/2024, 6:45:27 AM	Last Modified	3/20/2024, 6:45:42 AM		

Ansible Shortcomings

Calling one function in a loop:

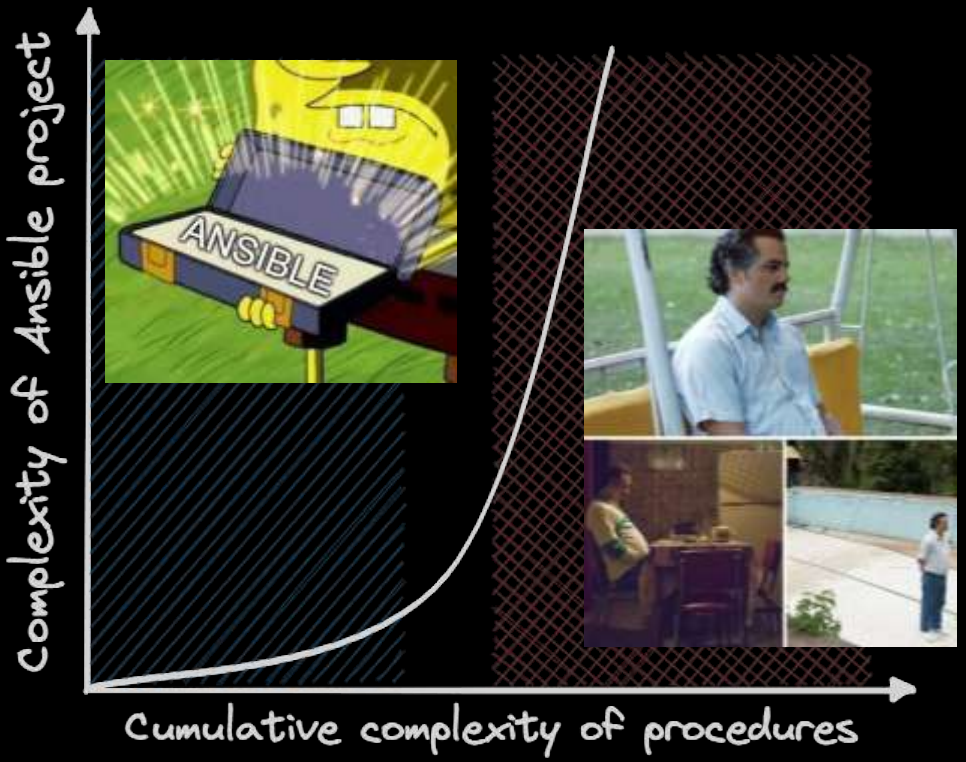
```
.
├── bax_check
│   ├── bax_card_checks.yml
│   └── bax_card_restart.yml
└── playbook.yml
```

Devices with TL1 interface? DIY modules!

```
ENT-OEL::prova:trnbi:::LABEL=test,SRCNODENAME=ols-a,DSTNODENAME=ols-
b,MODULATION=PM-NONE, RATE=NA,CARDTYPE=UNKNOWN,FREQSLOTPLANTYPE=FREQ-SLOT-
PLAN-NONE,VALIDFRANGELIST=191325000&196125000,EXPLICITROUTE=ols-a&1-A-2-
L1&ols-b&1-A-1-L1,OELSOURCE=MANUAL,GAURDBAND=0,NUMFECITRNS=4:IS;
```

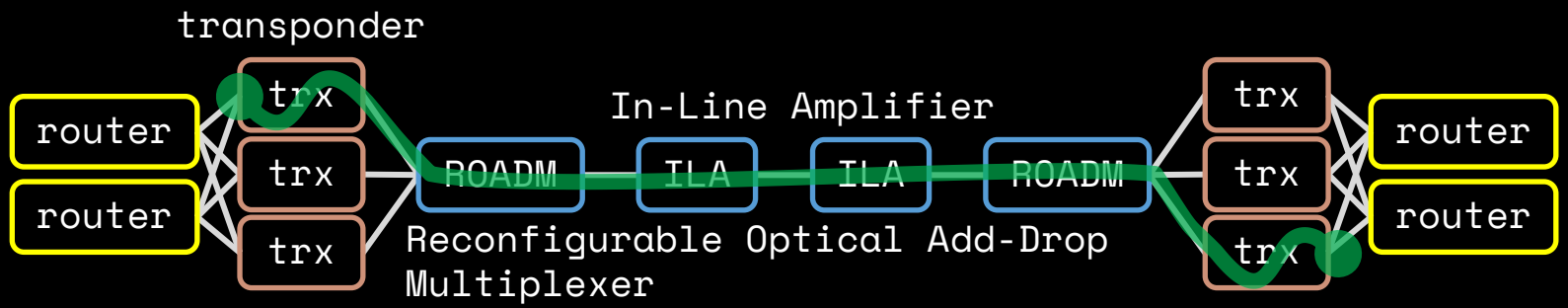
Lesson Learned

- Great for very simple procedures
- Better to use programming languages for complex workflows
- Config-centric, not service-centric



Exp.#2: Vendor Controller's API/NBI

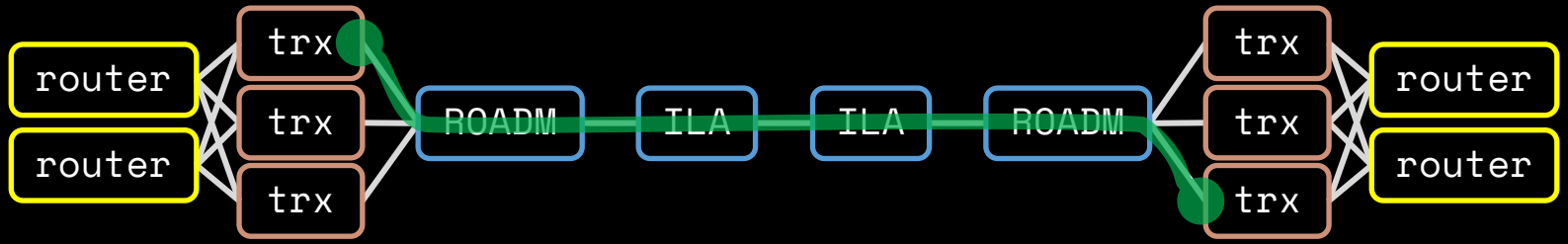
Goal: automate optical circuits provisioning



Manual procedure: ~40-50 clicks across 4 GUIs

NBI Shortcomings

Only Line-to-Line

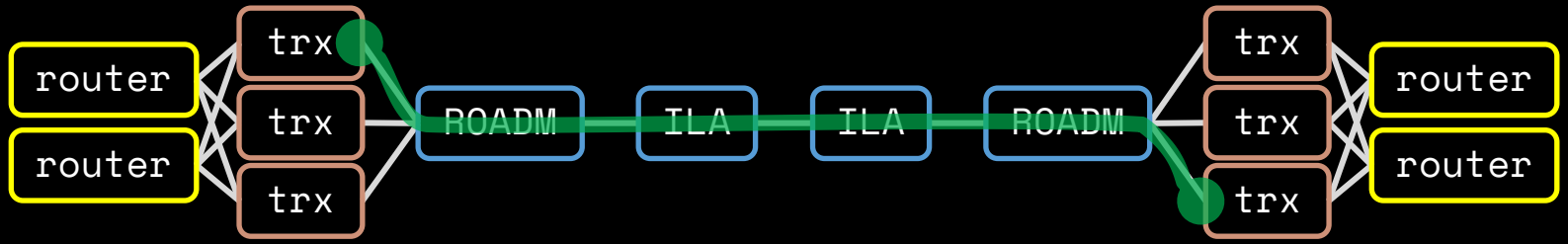


Didn't replace clicks:

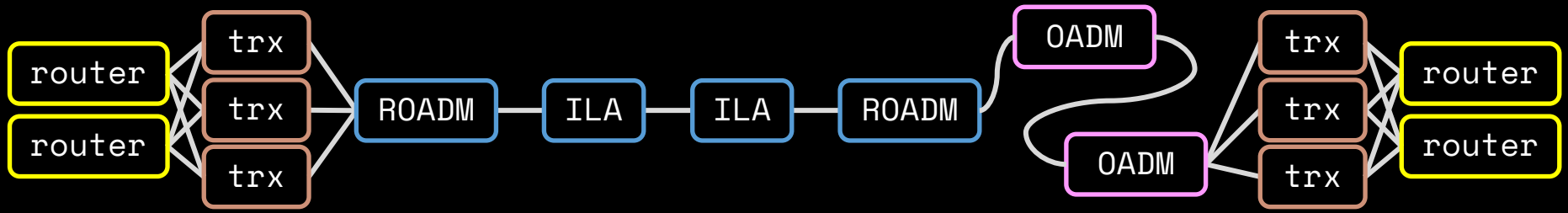
- pre-provisionig configurations
- creation of cross-connections on some cards
- fix non-meaningful names, add descriptions

NBI Shortcomings

Only Line-to-Line

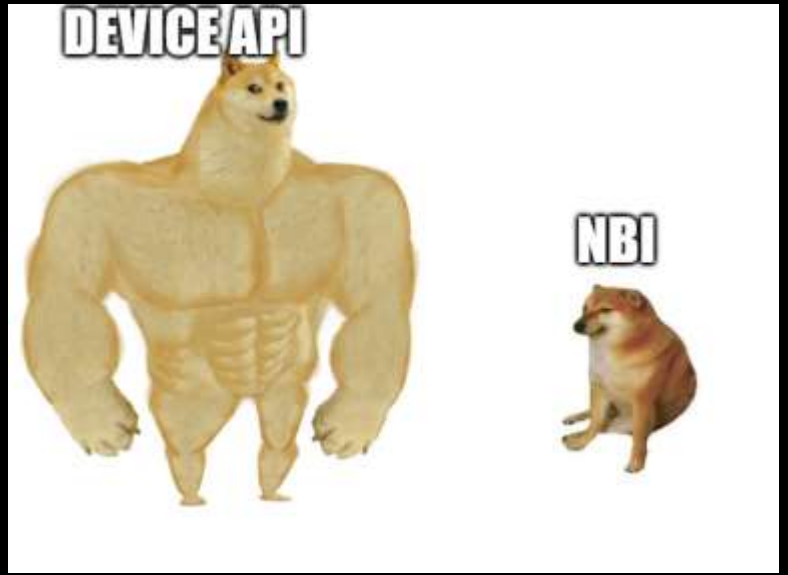


Cannot do anything if there's metro OLS



Lesson Learned

- Can be a bottleneck
- You’re not in the driving seat
- Use device interfaces (TL1, NETCONF, RESTCONF)



01

What didn't work and why

02

The Workflow Orchestrator framework

The Workflow Orchestrator

Developed by SURF

With ESnet ENERGY SCIENCES NETWORK, open-sourced and registered in the Commons Conservancy

workfloworchestrator.org

Other organizations using it

GÉANT

HEAnet Ireland's National Education & Research Network

Consortium GARR

What you get out of the box

An open-source framework where you

Define your network services/ entities (Products)

Track individual customer instances of those Products (Subscriptions)

Define clear procedures (Workflows) for creating, modifying, validating, and terminating Subscriptions

- FastAPI core engine + NextJS UI
- Everything is stored and tracked in the database

Products: modeling your reality

Define and link your building blocks:

```
class OpticalFiberBlock(ProductBlockModel,
product_block_name="OpticalFiber"):
    fiber_name: str | None = None
    oss_id: int
    terminations: ListOfPorts[OpticalDevicePortBlock]
    # … other fields
```

Turn them into a Product:

```
class OpticalFiber(SubscriptionModel):
    optical_fiber: OpticalFiberBlock
```

Workflows: making things happen

A list of Python functions (steps) - do whatever you want

```
>> reserve_resources_on_netbox              # Regular step
>> confirm_patching                         # User input step
>> check_reachability                       # Retry step
>> callback_config_manager                  # Callback step
>> conditional(should_enable_monitoring)(   # Conditional step
       enable_monitoring                    # Group of steps
   )
```

Write code that does not collapse under real-world pressure to coordinate actions across devices, systems, and people

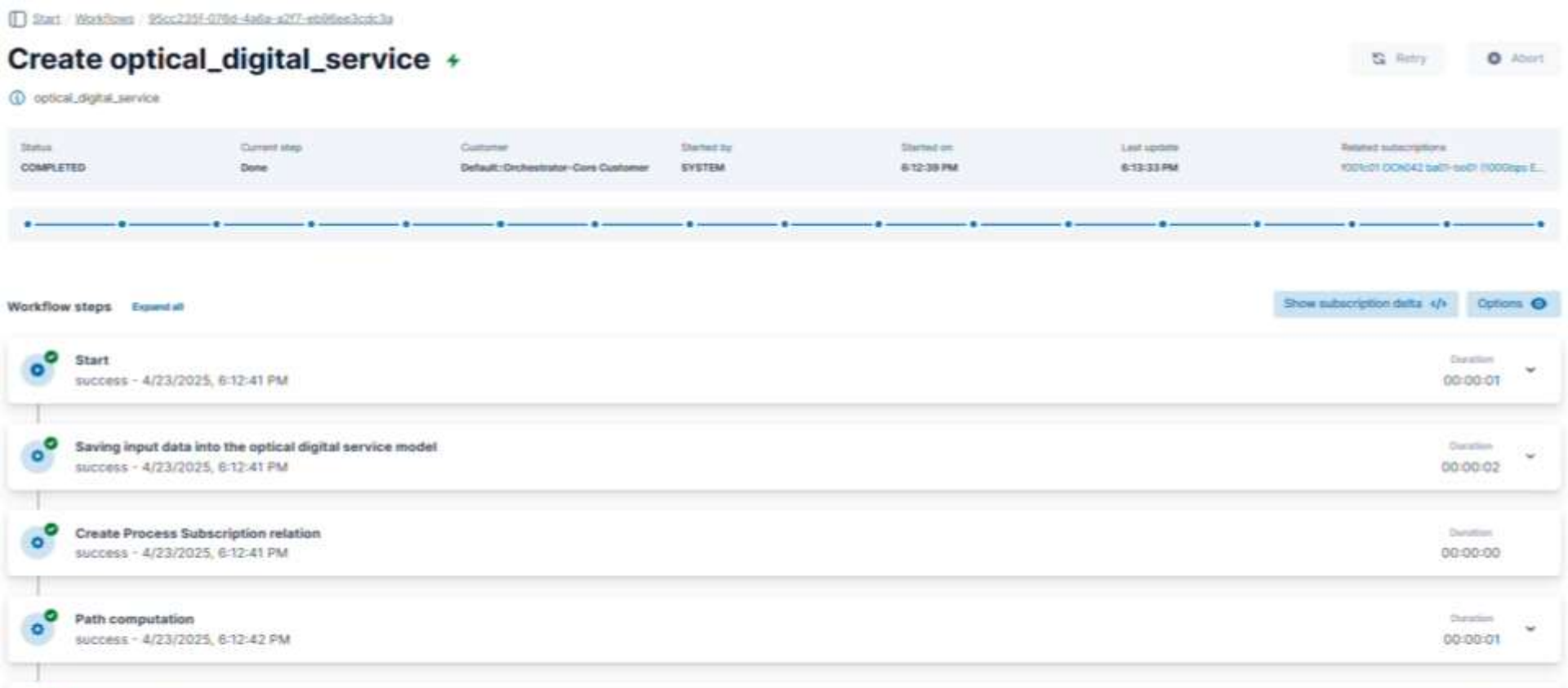
02

The Workflow Orchestrator framework

03

Our implementation without config managers

From 50 clicks…



Easy? No. Worth it?

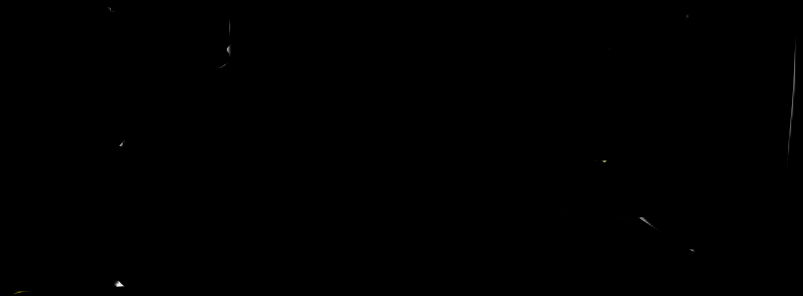
The real value is more than just automating tasks. It's managing the entire lifecycle of each service in a structured, repeatable, and auditable way.

- central definition of your services
- consistent execution of service management
- visibility into the state and history of each service
- a reusable toolkit for future services

Hardware changes. Products and workflows stay.

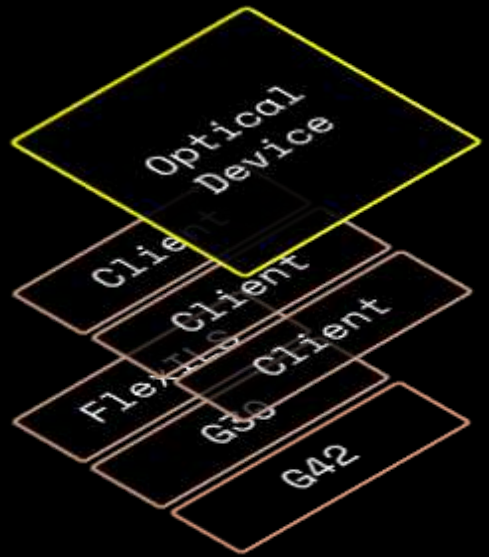
Abstract Composable Models and Device Clients

- One coherent architecture of models that aligns with our mental models



Abstract Composable Models and Device Clients

- One coherent architecture of models that aligns with our mental models
- Clients to decouple from hardware



One function to rule them all

Problem: same task, different platform
For example, in a workflow step:

```
for port in subscription.optical_fiber.terminations:
    device = port.device
    port_name = port.port_name
    set_port_admin_state(device, port_name, "up")
```

Idea: one function with multiple implementations for each platform

```python
@set_port_admin_state.register(Platform.FlexILS)
def _(
    optical_device: OpticalDeviceBlock,
    port_name: str,
    admin_state=Literal["up", "down", "maintenance"],
) -> Dict[str, Any]:
    # FlexILS implementation

@set_port_admin_state.register(Platform.G30)
def _(…same args…) -> Dict[str, Any]:
    # G30 implementation
```

Benefits: keeps code organized, easy to add new platforms, simplifies workflow logic

Keep it simple, Talk Direct

Problem:
- NBI = loss of control/functionalities
- Ansible = just adds complexity

Idea: communicate directly with devices like any other API

Benefits: simplicity and maintainability

```python
@set_port_admin_state.register(Platform.Groove_G30)
def _(
    optical_device: OpticalDeviceBlock,
    port_name: str,
    admin_state=Literal["up", "down", "maintenance"],
) -> Dict[str, Any]:
    ids = port_name.split("-")[-1]  # port-1/2/3 -> 1/2/3
    shelf_id, slot_id, port_id = ids.split("/")  # 1/2/3 -> 1, 2, 3

    g30 = g30_client(optical_device.mngmt_ip) # RESTCONF client
    port = g30.data.ne.shelf(shelf_id).slot(slot_id).card.port(port_id)
    # dynamic Path → https://{{host}}:{{port}}{{+restconf}}/data/ne:ne/shelf={{
shelf_id}}/slot={{slot_id}}/card/port={{port_id}}

    port.modify(admin_status=admin_state) # PATCH method with data validation

    return port.retrieve(depth=2) # GET method
```

```python
@set_port_admin_state.register(Platform.GX_G42)
def _(
    optical_device: OpticalDeviceBlock,
    port_name: str,
    admin_state=Literal["up", "down", "maintenance"],
) -> Dict[str, Any]:
    shelf_id, slot_id, port_id = port_name.split("-")  # 1-4-L1 -> 1, 4, L1

    g42 = g42_client(optical_device.mngmt_ip)
    port = g42.data.ne.equipment.card(f"{shelf_id}-{slot_id}").port(port_id)

    port.modify(admin_state=admin_state)

    return port.retrieve(depth=2)
```

03

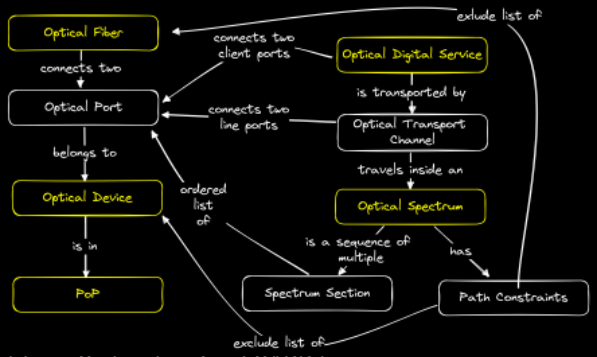
Our implementation without config managers

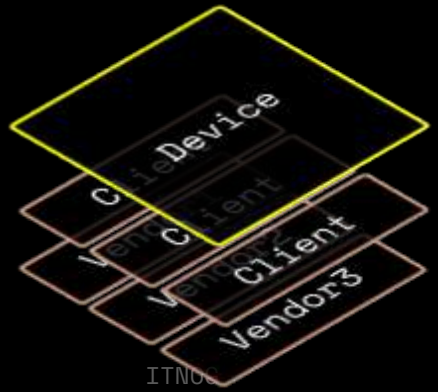
Key Take-Aways

Scalable and maintainable: composable models, stateful instances and procedures

Programmable: Use devices' programmable interfaces and YANG models, not just config lines

Sustainable transformation: automate one service at a time, nudge people out of inertia





Questions?