OpsMill

AUTOCON 6 | WS:D2

Building a Network Source of Intent

From architecture to practice | NAF Slack #ac6-ws-d2

Alex Gittings, Damien Garros, Mikhail Yohman,
and Adam Byczkowski | 17 November 2026

Agenda

01. Welcome, first look, and the framework (15 min)

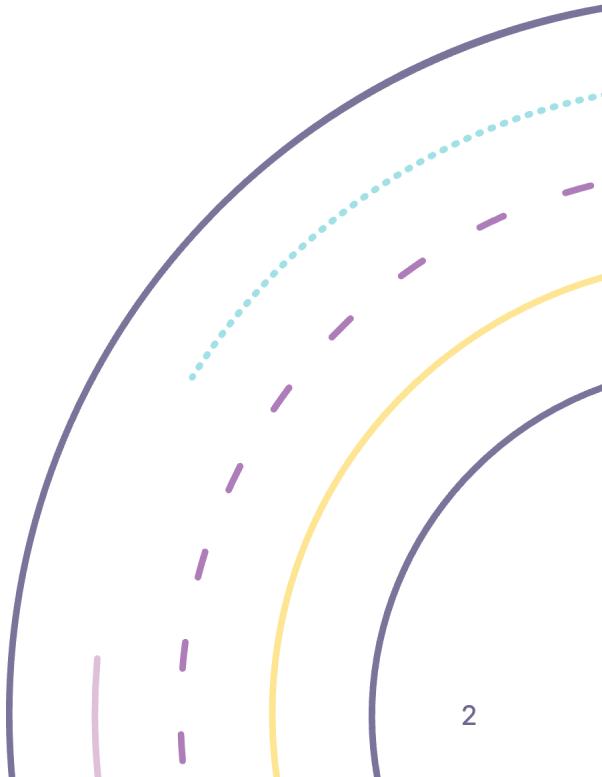
02. Schema Modeling (40 min)

03. Design-Driven Generator (45 min)

04. Transforms and Config Rendering (40 min)

05. AI-Assisted Development (60 min)

06. From the lab to your network (20 min)



Welcome to OtterNet

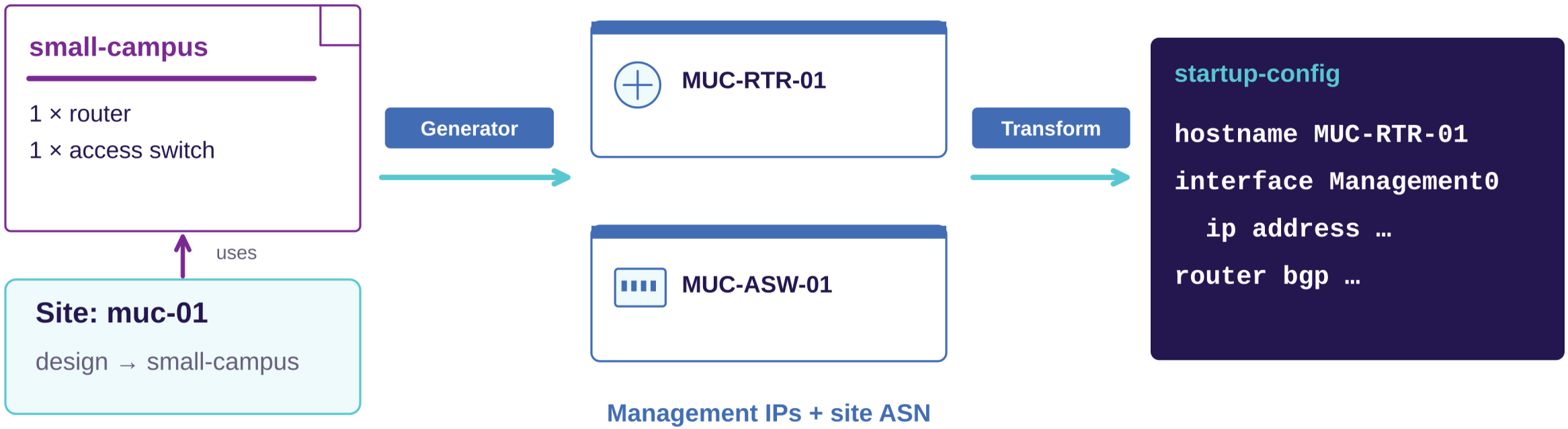
OtterNet is a fictional ISP that sells dedicated internet access to business customers.

You are on its network automation team. The company is expanding fast, and office deployments must become repeatable.

TODAY'S GOAL

Deploy the new Munich site from its design to a reviewed, rendered configuration.

What you'll build today



Review the schema, data, code, and rendered output before merging.

Find your way around the lab

Log in

Open your Instruqt invite. In Infrahub, log in as admin with password infrahub.

Editor and Terminal

Edit files in Code Editor. Run infrahubctl in Terminal.

Schema and data

In /playground, schema files are in schemas/ and object files in objects/.

Branch selector

Every view and query uses the branch you select.

Proposed changes

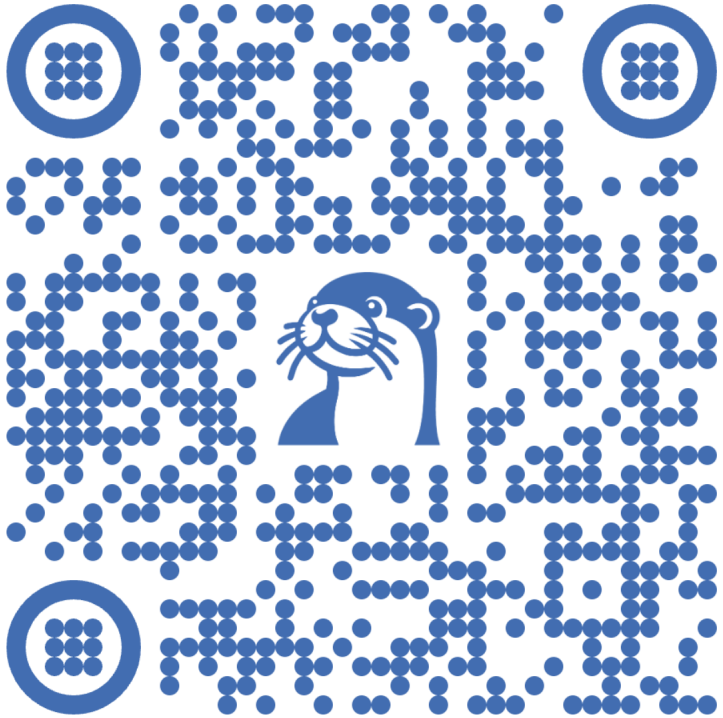
View a branch's diff and checks, then merge it.

GraphQL results

Each value is nested as edges, node, attribute, value.

> Each track opens a fresh, prepared environment. Your previous VM changes do not carry over; the next lab starts with the state it needs.

Join the lab



opsmill.com/labs2

TRACK

Schema Modeling

1. Scan the code, or open the invite link
2. Start the **Schema Modeling** track
3. Log in to Infrahub as admin, password infrahub

> You use this environment for the first task and for Track 1.

Your first task: explore London

ESSENTIAL

Log in to Infrahub in the Schema Modeling environment

Under Location, find the London site, lon-01

Open the device LON-RTR-01 and follow its relationships: location, interfaces, and groups

CHALLENGE

Switch the branch selector, then switch back to main

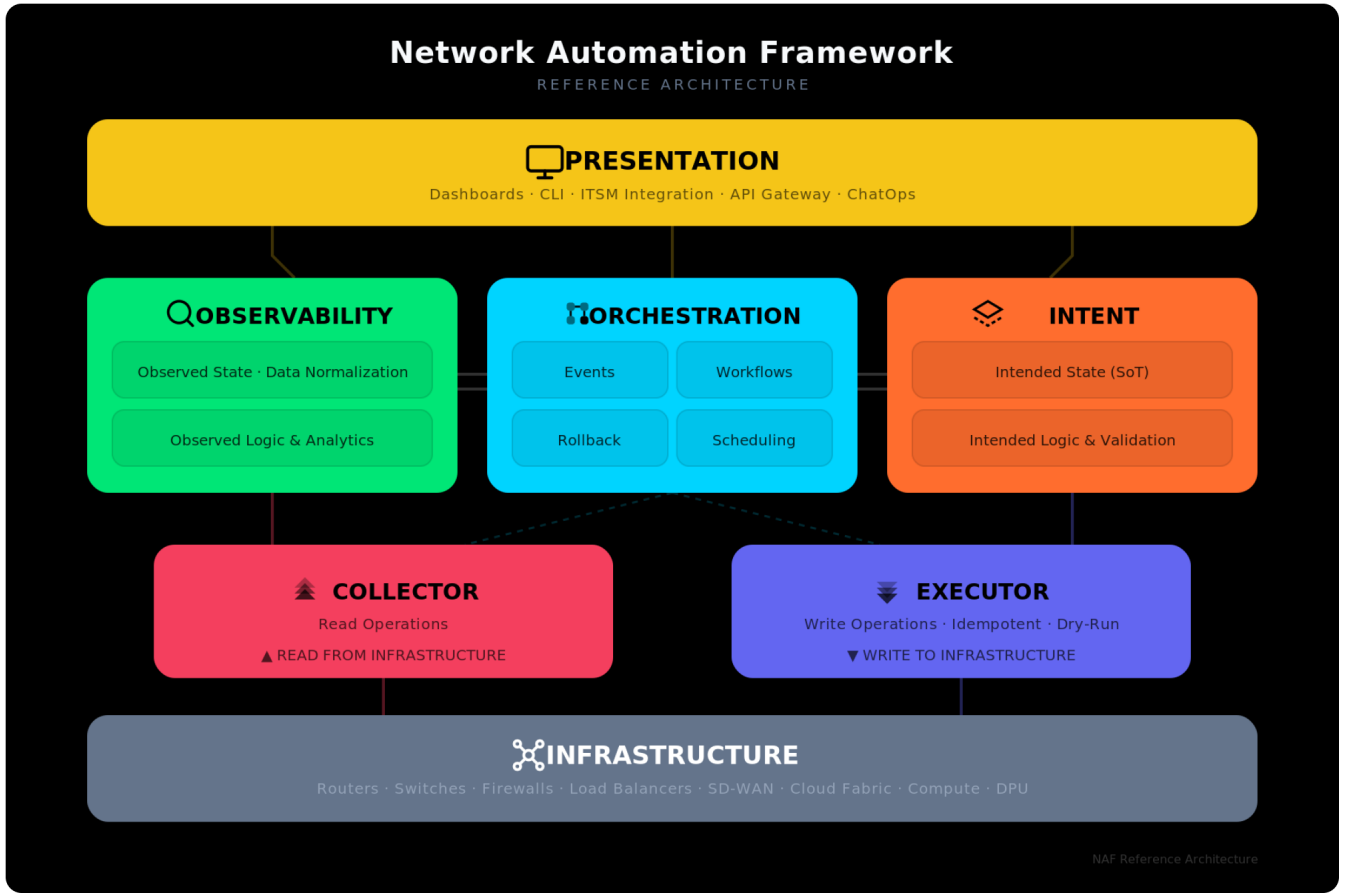
CHECKPOINT

You can say which site, interfaces, and groups LON-RTR-01 is linked to

Lab time: 5 minutes

Functionality: Data Consumption

Today's work sits in the NAF Intent block



Intent holds the desired state of the network.

> *"The (ideally) vendor-agnostic data and design logic needed to rebuild your network's desired state from scratch."*
>
> NAF Intent block, working draft

Service intent + design policy → technical intent

In the labs: muc-01 + the small-campus design → the generator creates the devices.

We treat the rendered configuration as intent too. The draft puts rendering in the Executor and calls the question open.

Network intent has three layers

Service also called the service layer	What the service needs	**OtterNet:** muc-01 uses small-campus, which calls for three VLANs
↓ Generate with a generator		
Technical	Networks, addresses, routes, and access rules	**OtterNet:** MUC-RTR-01, MUC-ASW-01, management IPs, and an ASN
↓ Translate with a transform		
Component	Device settings and configurations	**OtterNet:** the startup-config artifact for each device

Rendering is part of intent. Each configuration is versioned and reviewed with the data that produced it.

OpsMill's three intent layers. Designing Network Automation at Scale and the NAF Fremaework describes two: service and technical.

Four things a source of truth does

Data Structure

Schema constraints validate writes. Operational checks validate proposed changes.

TODAY

Schema Modeling

Data Persistence

Connected queries, transactions, versions and branches, and lineage

TODAY

Every track: branches and proposed changes

Data Consumption

APIs and views for people and systems, including AI agents through MCP

TODAY

Transforms and AI

Data Federation

Joins other systems. Exactly one system decides each value.

TODAY

Closing discussion

Each track opens by showing which of these it exercises.

Source: Designing Network Automation at Scale, chapters 4 and 5 | designingnetworkautomation.com

Where your own logic goes

Schema — The foundation: your nodes, attributes, and relationships. Everything below reads it. Track 1

TRACK 2

Generator

Creates, updates, or deletes data. Rerunning it changes nothing.

IN: data · LOGIC: Python

OUT: data in the graph

TRACK 3

Transform

Renders data into a file, such as a device configuration. No side effects.

IN: data · LOGIC: Jinja2 or Python

OUT: a stored artifact

SUPPLIED IN TRACK 2

Check

Validates a business rule before a proposed change can merge.

IN: data · LOGIC: Python

OUT: pass or fail

Generators derive the technical objects. Transforms render them into artifacts. Checks decide whether the change can merge.

Track 1: Schema Modeling

MODEL **Site design**	**GENERATE** Technical objects	**RENDER** Configuration	**REVIEW** Accepted change

Development stages, not track names. Every track works on a branch, and review happens throughout.

SOURCE OF TRUTH FUNCTIONALITIES

- **Data Structure**
- Data Persistence
- Data Consumption
- Data Federation

INTENT LAYERS

Business: the site designs. Technical: the sites, devices, and addresses.

NAF INTENT REQUIREMENTS

MUST represent any network aspect in a structured form

SHOULD give a unified view of the desired state

SHOULD include metadata for data governance

A well-designed schema makes intent reusable, consistent, and enforceable.

One site, three layers, one schema

LAYER	WHAT OTTERNET STORES FOR MUC-01	THE SCHEMA THAT HOLDS IT

Service
you write this schema
	Munich is a small campus: one router and one access switch	`OtnSiteDesign → OtnCampusSite`
		`OtnDesignDeviceEntry`
		`LocationSite.design`

↓ **Generate** with a generator		

Technical
base schema, extended
	MUC-RTR-01 with management IP 172.16.2.1/32, in a site with prefix 172.16.2.0/24 and ASN 65000	`DcimDevice`
		`LocationSite.mgmt_prefix`
		`LocationSite.bgp_asn`

↓ **Translate** with a transform		

Component
built into Infrahub
	`hostname MUC-RTR-01`
	`ip address 172.16.2.1/32`
	`router bgp 65000`
		`CoreArtifact`
		the stored startup-config, one per device

Relationships in the schema connect the layers, so the request, the devices, and the configuration stay linked.

Anatomy of a schema node

LocationSite

NODE · namespace Location + name Site

ATTRIBUTES: values

`name`	Text
`bgp_asn`	Number
`mgmt_domain`	Text, computed from name, read-only

RELATIONSHIPS: links to other nodes

`design`	cardinality one, optional
`mgmt_prefix`	cardinality one

`design` → `OtnSiteDesign` generic: campus or data center

`mgmt_prefix` → `IpamPrefix` the site's management /24

THE BUILDING BLOCKS

Node: a type of object, such as a site or a device

Attribute: one value with a type, checked on every write

Relationship: a link to another node, with a cardinality

Generic: a shared contract that several nodes inherit

From this one YAML definition, Infrahub builds the UI pages and forms, the GraphQL API, and the validation.

Why two fields are enough for Munich

Field on muc-01	Value	Comes from	How
`name`	muc-01	**You**	Required input
`design`	small-campus	**You**	Pick a design
`mgmt_domain`	muc-01.mgmt.otternet.net	**Schema**	Computed from the name
`mgmt_prefix`	172.16.2.0/24	**Pool**	Next free /24
`bgp_asn`	65000	**Pool**	Next free ASN
Devices	MUC-RTR-01, MUC-ASW-01	**Generator**	Built from the design

You supply two fields. The schema, the pools, and the generator supply the rest, the same way every time.

Start from a curated schema

INFRAHUB MARKETPLACE

`marketplace.infrahub.app`

A public catalog of pre-built schemas that you fetch and load into any Infrahub instance.

```
# fetch a schema, or a whole collection
infrahubctl marketplace get infrahub/dcim
infrahubctl marketplace get \
  infrahub/base-schemas --collection
# then load it like any schema
infrahubctl schema load schemas/
```

Schemas

Versioned YAML for one domain, such as infrahub/dcim

Collections

Curated bundles of related schemas, such as infrahub/base-schemas

IN THE LAB

LocationSite and DcimDevice are already loaded from the base schemas. You add OtterNet's designs on top and extend LocationSite, without editing the base files.

Reuse the common models. Write schema only for what makes your network different.

A site follows its design

SITE

Munich

The location to deploy, linked to one design

DESIGN

Small, medium, or large

A reusable campus pattern

ENTRIES

Device entries

The devices the design needs, by role

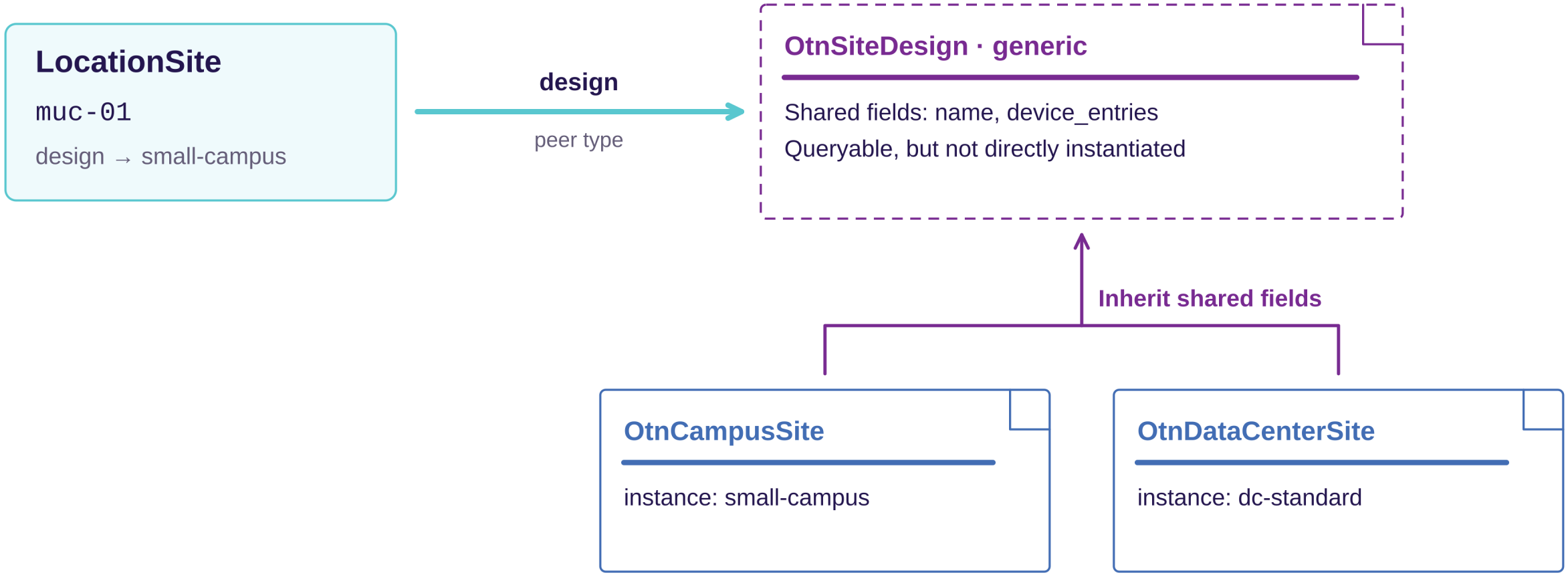
TEMPLATES

Object templates

The blueprint each device is created from

You describe the pattern once. The generator in the next track builds every site from it.

Why generics: one link, many types



Munich can select a campus or data center design through the same relationship.

dc-standard is an illustrative design name, not a lab object.

Challenge 01: one generic, two site types

schemas/extensions/designs/design.yml, shortened

```yaml
generics:
  - name: SiteDesign
    namespace: Otn
    relationships:
      - name: device_entries
        peer: OtnDesignDeviceEntry
        kind: Component
        identifier: design__device_entries
nodes:
  - name: CampusSite
    namespace: Otn
    inherit_from:
      - OtnSiteDesign
      - CoreArtifactTarget
  # DataCenterSite: same inherit_from
```

generics

Shared fields, defined once. One query on OtnSiteDesign returns both site types.

inherit_from

Each design type gets the generic's fields, plus artifact support

kind: Component

Device entries belong to the design and are deleted with it

identifier

Links the two sides of one relationship

Challenge 02: extend a site you didn't write

Appended to design.yml, shortened

```yaml
extensions:
  nodes:
    - kind: LocationSite
      attributes:
        - name: mgmt_domain
          computed_attribute:
            kind: Jinja2
            jinja2_template: "{{ name__value }}.mgmt.otternet.net"
      relationships:
        - name: design
          peer: OtnSiteDesign
```

extensions

Adds fields to LocationSite without editing the base schema files

peer: OtnSiteDesign

The peer is the generic, so a site can follow a campus or a data center design

computed_attribute

Infrahub derives mgmt_domain from the site name. Nobody types it.

Resource managers: ask, don't track

A pool allocates resources without conflicting assignments. Asking again for an existing allocation returns the same value.

OtterNet pool	Draws from	Hands out	Keyed by
`otn-mgmt-prefix-pool` CoreIPPrefixPool	172.16.0.0/16	A /24 for each site	The site code
`otn-asn-pool` CoreNumberPool	65000–65534	One BGP ASN for each site	The site's bgp_asn
`muc-01-mgmt-pool` CoreIPAddressPool, created by the generator	The site's /24	A management IP for each device	The hostname

Same key, same value

Ask again for muc-01 and you get the same prefix and ASN. That makes a generator rerun safe.

Allocating is not merging

Allocating lon-01.bgp_asn on two branches returns the same ASN: the allocation is keyed to the node and attribute. Merging brings the site's value onto main.

Object templates: define a device once

A template holds the values every device of one kind shares. Each new device starts from it.

DESIGN ENTRY

Router entry

`OtnDesignDeviceEntry`

In the small-campus design

count: 1

template: the router template

uses →

OBJECT TEMPLATE

Router template

Role

Device type

Platform

Interfaces, as component templates

creates →

DEVICE

MUC-RTR-01

Name: set by the generator

Management IP: from the site pool

Everything else: filled from the template

Turn on in the schema

generate_template: true on the node

Override per device

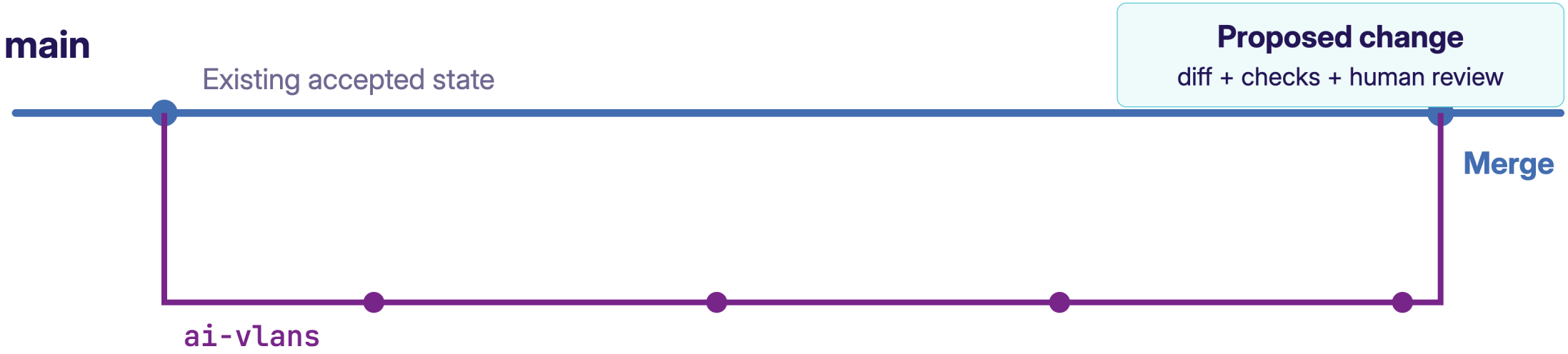
Any value can change after the device is created

No retroactive changes

Editing a template doesn't change devices already created

Every change: branch, then review

Preview of the AI track's ai-vlans branch. In Schema Modeling, your branches are schema-design and schema-extensions.



Schema
Two VLAN nodes

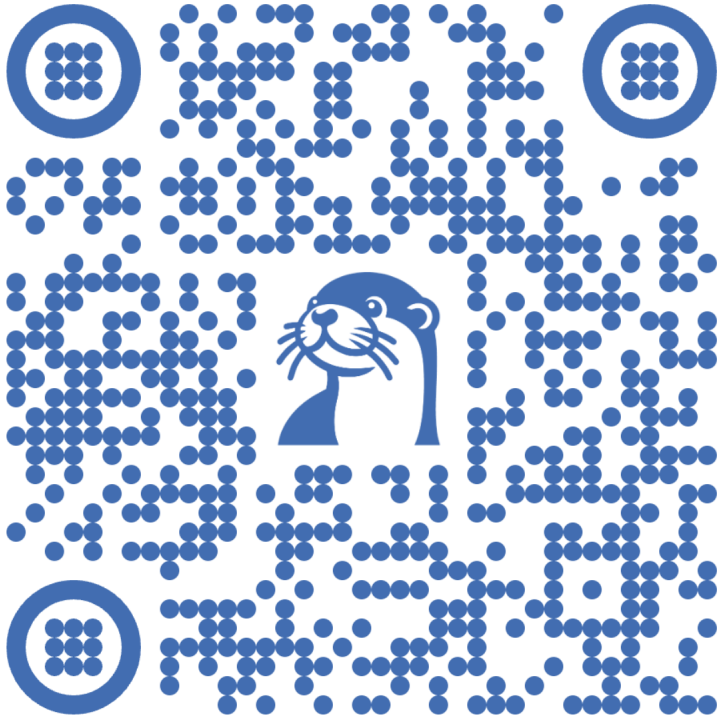
Data
Design VLAN entries and muc-01 VLANs

Code
Generator and transform

Artifact
Rendered vlan blocks

Inspect the branch's full result before you merge it into main.

Lab: Schema Modeling



opsmill.com/labs2

TRACK 1

Schema Modeling

1. Go back to the Schema Modeling track you started for the first task
2. Not started yet? Scan the code and start it now
3. Begin with challenge 01

> Session slot: 40 minutes. The next slide lists your tasks.

Your task: model a repeatable campus

ESSENTIAL

01 Define the OtnSiteDesign generic and its site types

02 Extend LocationSite with an extensions block

CHALLENGE

03 Build the IP prefix and BGP ASN pools

04 Load the device templates and create a device from one

STRETCH

05 Create the designs and bring Munich online

CHECKPOINT

Success looks like this:

The new design types appear in the UI after you merge

LocationSite has design and mgmt_prefix relationships

Each challenge's Check passes

Session slot: 40 minutes

Functionality: Data Structure

Track 2: Design-Driven Generator

MODEL	GENERATE	RENDER	REVIEW
Site design	**Technical objects**	Configuration	Accepted change

Development stages, not track names. Every track works on a branch, and review happens throughout.

SOURCE OF TRUTH FUNCTIONALITIES

- Data Structure
- **Data Persistence**
- Data Consumption
- Data Federation

INTENT LAYERS

Generate: the business layer expands into the technical layer.

NAF INTENT REQUIREMENTS

MAY include the logic for managing intended state

MUST support create, read, update, and delete

SHOULD be transactional, with versioned access

A reusable design generates consistent infrastructure, device after device.

A generator turns design knowledge into code

Script using the SDK	Registered generator
Can read the design, use pools, and upsert	Declared with a query and a target group
Runs when someone starts it	Runs once for each member of the group
Not declared, so Infrahub can’t run it	Run by Infrahub on demand or from an event

The SDK calls are the same. Registering the logic lets Infrahub run it for every site.

From design to devices

BEFORE

muc-01

design → small-campus

1 router + 1 access switch

No devices generated yet

No site resources allocated yet

Run generator



AFTER

MUC-RTR-01

`172.16.2.1/32`

MUC-ASW-01

`172.16.2.2/32`

Site resources

`Prefix: 172.16.2.0/24 ASN: 65000`

Illustrative allocations. Inspect your own branch for the actual values.

The query feeds the generator

```
query CampusSite($name: String!) {
  LocationSite(name__value: $name) {
    edges { node {
      design { node {
        ... on OtnCampusSite {
          device_entries { edges { node {
            count { value }
            device_hostname_prefix { value }
            template { node { id } }
          } } } }
      } } } }
  } }
```

$name

Filled from each target site's name, such as muc-01

... on OtnCampusSite

design points at the generic, so a fragment selects the campus fields

device_entries

How many of which device, and the template for each

How the generator asks the pools

generators/campus_site_generator.py, shortened

```
# 1. Site prefix, keyed by the site code
mgmt_prefix = await self.client.allocate_next_ip_prefix(
    resource_pool=prefix_pool, identifier=site_code)

# 2. Site ASN: give the field the pool
site_node.bgp_asn = asn_pool
await site_node.save(allow_upsert=True)

# 3. Device IP from the site pool, keyed by hostname
mgmt_ip = await self.client.allocate_next_ip_address(
    resource_pool=mgmt_pool, identifier=hostname)
```

Ask, don't track

The code never checks what is in use. The pool does.

Two ways to ask

Call allocate_next_... directly, or assign the pool to a field and save

A stable key

Site code and hostname don't change between runs, so neither do the values

Run it twice, get the same result

RUN 1

MUC-RTR-01
172.16.2.1/32

MUC-ASW-01
172.16.2.2/32

Site prefix: 172.16.2.0/24
Site ASN: 65000

RUN 2 · SAME INPUT

MUC-RTR-01
172.16.2.1/32

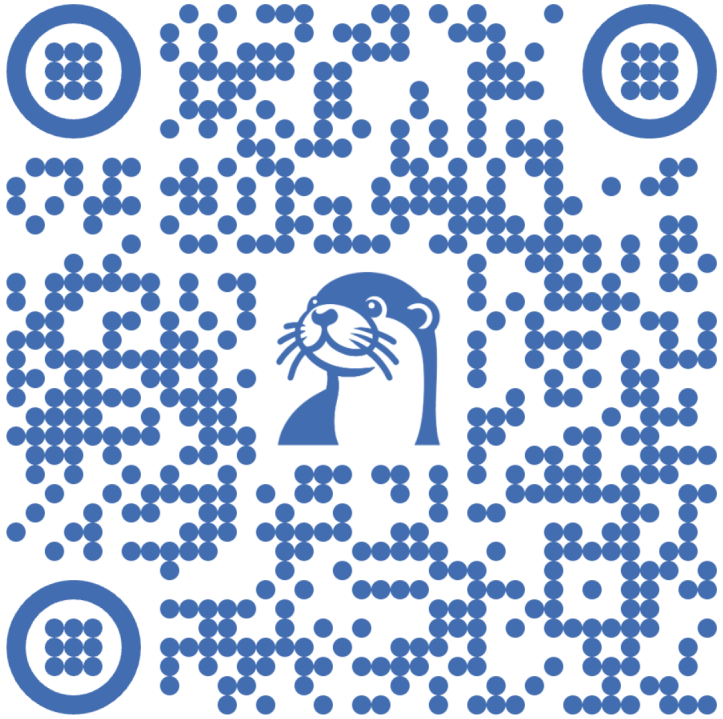
MUC-ASW-01
172.16.2.2/32

Site prefix: 172.16.2.0/24
Site ASN: 65000

Still 2 devices · 0 additional devices · same IPs, prefix, and ASN

Example values. Inspect your own branch for the actual allocations.

Lab: Design-Driven Generator



`opsmill.com/labs2`

TRACK 2

Design-Driven Generator

1. Scan the code, or open the invite link
2. Start the **Design-Driven Generator** track
3. Begin with challenge 01

> A fresh, prepared environment: muc-01 already has its design. Session slot: 45 minutes.

Your task: turn the design into devices

ESSENTIAL

01 Create the campus_sites group and add muc-01

02 Write the generator's query and test it

03a Implement the generator and run it on build-generator

03b Rerun and verify: compare with the first run

CHALLENGE

04 Review the proposed change, let the checks run, and merge

CHECKPOINT

Success looks like this:

MUC-RTR-01 and MUC-ASW-01 exist, each with a management IP

After the second run: still two devices, no duplicates

Same management IPs, prefix, and ASN as after the first run

Each challenge's Check passes

Session slot: 45 minutes

Functionality: Data Persistence

Break

10 minutes. Next: Transforms and Config Rendering.

Track 3: Transforms and Config Rendering

MODEL	GENERATE	RENDER	REVIEW
Site design	Technical objects	**Configuration**	Accepted change

Development stages, not track names. Every track works on a branch, and review happens throughout.

SOURCE OF TRUTH FUNCTIONALITIES

- Data Structure
- Data Persistence
- **Data Consumption**
- Data Federation

INTENT LAYERS

Translate: the technical layer becomes the component layer. Rendering stays in intent.

NAF INTENT REQUIREMENTS

SHOULD use a neutral representation, derived into vendor configuration

MUST be exposed through a standardized, documented API

Device configuration is part of intent: rendered from the data and reviewed with it.

Where this track ends: the MUC-RTR-01 config

Line in the rendered config	Where the value comes from
`! Managed by OtterNet SoT — muc-01.mgmt.otternet.net`	The site's computed mgmt_domain
`hostname MUC-RTR-01`	The device name the generator set
`ip domain-name MUC-RTR-01.muc-01.otternet.net`	The device's computed fqdn
`ip address <IP from 172.16.2.0/24>`	Allocated by the generator from the site's IP pool
`router bgp <ASN from 65000-65534>`	Allocated by the generator from otn-asn-pool

Angle brackets mark output patterns: the IP and ASN differ in each lab. Nobody types these values.

From data to device config

STEP 1

GraphQL query

Gathers one device's data

STEP 2

Jinja2 template

Formats that data as plain text

STEP 3

Configuration

The text a device or pipeline consumes

The query uses the same edges, node, and value structure as the generator's query.

Anatomy of a transform

SOURCE DATA	JINJA2 EXPRESSION	RENDERED LINE
Device name `MUC-RTR-01`	`{{ device.name.value }}`	`hostname MUC-RTR-01`
Management IP `172.16.2.1/32`	`{{ mgmt_ip }}`	`ip address 172.16.2.1/32`
Site ASN `65000`	`{{ site.bgp_asn.value }}`	`router bgp 65000`

The query supplies the values. The template supplies the configuration syntax.

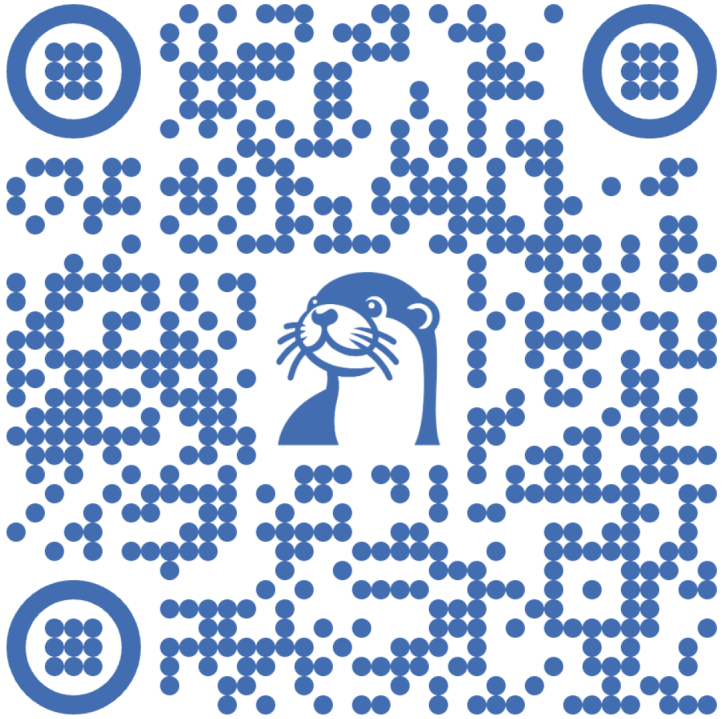
Example values. Your lab's IP and ASN differ. The full query and template are in Instruqt, challenge 01.

A transform renders, an artifact keeps the result

Transform	Stored artifact
A query plus a template	Rendered for each member of a group
Renders on demand	Stored and versioned
Runs locally while you develop	Shown as a diff in the proposed change

Reviewers approve a config diff a network engineer can read, not only a data diff.

Lab: Transforms and Config Rendering



`opsmill.com/labs2`

TRACK 3

Transforms and Config Rendering

1. Scan the code, or open the invite link
2. Start the **Transforms and Config Rendering** track
3. Begin with challenge 01

> A fresh, prepared environment: the Munich devices already exist.
> Session slot: 40 minutes.

Your task: render the Munich config

ESSENTIAL

01 Write the query and template, register them, and render MUC-RTR-01 locally

CHALLENGE

02 Store the config as an artifact, review it in a proposed change, and merge

STRETCH

03 Run the generator from an event rule

04 Deploy a new Berlin site end to end

CHECKPOINT

Success looks like this:

infrahubctl render prints hostname MUC-RTR-01

The IP is from 172.16.2.0/24 and the ASN from 65000–65534

You can name the source of each line

Session slot: 40 minutes

Functionality: Data Consumption

Break

10 minutes. Next: AI-Assisted Development.

Track 4: AI-Assisted Development

MODEL	GENERATE	RENDER	REVIEW
Site design	Technical objects	Configuration	Accepted change

AI works across all four stages. Every track works on a branch, and review happens throughout.

SOURCE OF TRUTH FUNCTIONALITIES

- Data Structure
- **Data Persistence**
- **Data Consumption**
- Data Federation

NAF INTENT REQUIREMENTS

MUST be exposed through a standardized, documented API

SHOULD be transactional, with versioned access

INTENT LAYERS

One change across all three layers, from schema to config.

AI speeds up the work without skipping validation, versioning, or review.

One requirement touches every layer

Generator: one VLAN per entry

1 · DESIGN

small-campus

`vlan_entries`

```
10 users
20 voice
30 printers
```

2 · SITE

muc-01

`vlans`

```
10 users
20 voice
30 printers
```

3 · TRANSFORM

Fetch the site's VLANs

Print one vlan block per VLAN, sorted by VLAN ID

4 · CONFIG

MUC-ASW-01

```
vlan 10
 name users
!
vlan 20
 name voice
! …
```

A rerun creates no duplicate VLANs.

A site with no VLANs, such as lon-01, renders no vlan lines.

Review the schema, code, site data, and rendered configuration together.

Define done before you prompt

#	Acceptance criterion
1	Sites have VLANs and designs have VLAN entries, IDs 1 to 4094
2	Only small-campus has entries: 10 users, 20 voice, 30 printers
3	After the generator runs, muc-01 has those VLANs, and a rerun adds none
4	MUC-ASW-01 renders three vlan blocks, and LON-RTR-01 renders none
5	Everything is on ai-vlans, and you review and merge one proposed change

Write the criteria first. They are how you judge what the assistant produces.

Skills teach the assistant Infrahub's conventions

A skill per kind of change

Schemas, objects, checks, generators, and transforms, plus data analysis

CLAUDE.md picks the skill

The project file names the skill for each change. You can also name one yourself.

One install command

npx skills add works for Claude Code, GitHub Copilot, Cursor, and other assistants

Describe the change in plain language, then review what comes back.

Why a skill, not just a prompt

PROMPT ONLY

The assistant guesses Infrahub's conventions

Changes can be incomplete or wrong

You review a half-finished draft

PROMPT + SKILL

The skill carries Infrahub best practice

The output follows platform conventions

You review a finished change

WHAT A SKILL DOES NOT DO

It does not make the design decisions for you, and it does not replace your review.

An example prompt from challenge 03

> Change campus_site_generator so that for each VLAN entry in the site's design, it creates an OtnVLAN at the site with the same vlan_id and name. Update the generator's GraphQL query to fetch the design's vlan_entries. Keep the generator idempotent: running it again must not create duplicate VLANs.

CLAUDE.md tells the assistant to load the infrahub-managing-generators skill for this change. You review the diff it shows.

Name the code

campus_site_generator and its query

State the rule

One VLAN at the site for each VLAN entry in its design

Name the data

Fetch the design's vlan_entries in the query

State the constraint

A second run creates no duplicate VLANs

From requirement to approved change

STEP 1

Schema and data

Two VLAN nodes and the small-campus entries, on ai-vlans

STEP 2

Generator

Creates the design's VLANs at muc-01, run on the branch

STEP 3

Transform

Adds vlan blocks, rendered for MUC-ASW-01

STEP 4

Proposed change

Schema, data, files, and artifact diffs

STEP 5

Human review

You merge it into main

The assistant commits and pushes after each step. The change reaches main only when you merge it.

The Infrahub MCP server

Any MCP client can read the schema, query the graph, and propose changes on a branch.

CONTEXT

Learns the schema before it asks

`infrahub://schema`

READ

Finds objects and relationships

`get_nodes`
`search_nodes`
`query_graphql`

WRITE

Changes land on a session branch

`node_upsert`
`mutate_graphql`
`propose_changes`

Every write: `mcp/session-…` **→ proposed change → a person merges**

Challenge 06: work on live data through MCP

STEP 1

Connect

Add .mcp.json, restart Claude Code, and confirm with /mcp

STEP 2

Query live data

List the muc-01 devices, then each site's design, VLANs, and device count

STEP 3

Request a change

Add VLAN 40 cameras to muc-01 and open a proposed change

STEP 4

Review

Only muc-01 changes, and the Munich configs gain vlan 40

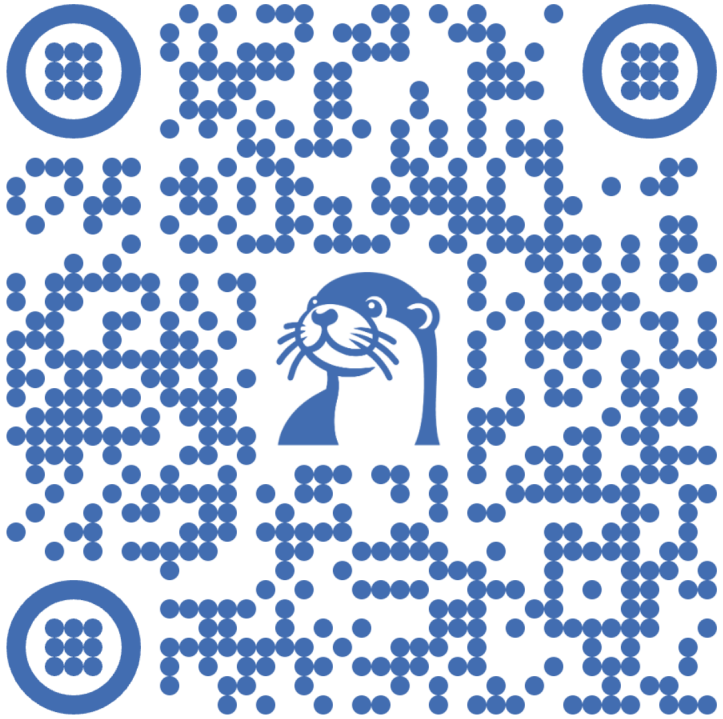
STEP 5

Merge

You merge it, and muc-01 shows four VLANs on main

The MCP server writes to an mcp/session branch, never to main. A data change made through MCP reaches the router configs through the transform you changed in challenge 04.

Lab: AI-Assisted Development



opsmill.com/labs2

TRACK 4

AI-Assisted Development

1. Scan the code, or open the invite link
2. Start the **AI-Assisted Development** track
3. Begin with challenge 01: install the skills

> A fresh, prepared environment with its own starting state. All six challenges are essential.

Your task: one change, schema to config

ESSENTIAL: EVERY CHALLENGE

01 Install the Infrahub skills and start Claude Code

02 Add the VLAN nodes and the design's VLAN entries on ai-vlans

03 Change the generator and run it on the branch

04 Change the transform and render MUC-ASW-01

05 Review and merge the proposed change

06 Connect MCP, query live data, request a change, review its proposed change, and merge it

CHECKPOINT

Success looks like this:

On main, muc-01 has VLANs 10, 20 and 30

MUC-ASW-01's startup-config artifact lists the VLANs

On main, muc-01 also has VLAN 40 cameras

Each challenge's Check passes

Functionalities: Data Consumption, Data Persistence

What we built: Munich, end to end

A site design

Munich follows a reusable campus design

Generated devices

MUC-RTR-01 and MUC-ASW-01, built from templates

Allocated resources

Management IPs and an ASN, from pools

Rendered configs

Stored artifacts derived from the data

Reviewed changes

Every step merged through a proposed change

An AI-assisted change

Site VLANs from schema to config, then a change through MCP

Model → Generate → Render → Check → Review → Merge

Back in the architecture

Concept	Where you used it today	NAF Intent requirement
Three intent layers	Generator: business to technical. Transform: technical to component.	MAY: intent logic. SHOULD: neutral data.
Data Structure	The schema, designs, templates, and pools	MUST: structured form. SHOULD: unified view.
Data Persistence	Branches and proposed changes in every track	SHOULD: transactional, versioned access
Data Consumption	GraphQL, artifacts, skills, and MCP	MUST: documented API, and CRUD
Data Federation	Not in the labs: see the appendix	SHOULD: governance metadata

"Intent is the data; the Source of Truth is the system that stores and serves it."

Sources: OpsMill intent layers | Designing Network Automation at Scale, chapters 4 and 5 | NAF Reference Framework

What would this look like in your network?

- Which network data do you still manage by hand?
- Which system owns each data domain?
- Which configs could you generate from intent?
- Where should validation and review happen?
- Where could AI assistance help, and where not?

INFRAHUB PROVIDES

The data platform, APIs, branches, allocation, validation, and artifacts

YOUR TEAM DEFINES

Service models, business rules, templates, and acceptance criteria

Questions and next steps

Keep going

Run the orientation track

Finish the remaining challenges

npx skills add opsmill/infrahub-skills

Read more

docs.infrahub.app

reference.networkautomation.forum

designingnetworkautomation.com

Questions after today: #ac6-ws-d2 on NAF Slack, or autocon-workshop@opsmill.com

Join us tonight



opsmill.com/ac6-social

HOSTED BY OPSMILL AT AUTOCON 6

Otto's Social Club

Tuesday, November 17, 5–8 pm

Cottonwood Room, Westin La Paloma, Tucson. Mezzanine Level, directly above the front desk.

Meet the people behind Infrahub. Drinks, hors d'oeuvres, and OpsMill swag you won't find at the booth.

> **Scan the code to RSVP and save your spot**

Appendix: reference material



Each track starts from a prepared environment

TRACK 1

Schema Modeling

Starts with the base schema and seed data loaded

TRACK 2

Generator

Starts with the modeled schema and a connected repository

TRACK 3

Transforms

Starts with the completed generator and the Munich devices

TRACK 4

AI-Assisted Development

Starts with its own prepared state

Your work does not carry over between tracks. Each new environment continues the story from where the previous track ends.

Five goals of a source of truth

Capture all intent

From service requests down to IP addresses, including planned and retiring items

Maintain integrity

Validate types and ranges, track changes, approve major ones, and roll back

Support parallel work

Propose changes at once, bundle them all-or-nothing, and branch

Give secure access

For people and machines, through APIs, web UIs, and CLIs

Consolidate other systems

Reference or sync their data, and name the owner of each item

Out of scope

Observed state, device interaction, and orchestration belong to other blocks

Source: Designing Network Automation at Scale, chapter 4, Network Intent | designingnetworkautomation.com

Relationships: cardinality and kind

Setting	What it means	Example
cardinality: one	At most one peer. optional sets whether it is required	Device to platform
cardinality: many	A set of peers	Site to devices
min_count, max_count	Limits how many peers are allowed	At least one uplink
kind: Attribute	The peer's key fields show inline in the UI	Status, role
kind: Component	The peer is part of the parent and can be deleted with it	Design to device entries
kind: Parent	A required link upward that builds the hierarchy	Interface to device
kind: Generic	The default: a link with no special behavior	Device to tags

Working with object templates

Optional at creation

Pick a template or start from scratch

Override per object

Change any attribute or relationship after creation

Linked to pools

Each new object gets a unique IP or number

Components included

Interfaces and other components get templates too

Editing a template doesn’t change objects already created from it.

Anatomy of a generator

```python
from infrahub_sdk.generator import InfrahubGenerator

class CampusSiteGenerator(InfrahubGenerator):
    async def generate(self, data: dict):
        site = data["LocationSite"]["edges"][0]["node"]
        design = site["design"]["node"]
        for entry in design["device_entries"]["edges"]:
            # allocate IP and ASN from a pool
            # create the device from its template
            device = await self.client.create(...)
            await device.save(allow_upsert=True)
```

InfrahubGenerator

The SDK base class

generate(data)

Your logic, run on the query result

save(allow_upsert)

Creates the object once, then updates it on every rerun

Devices from object templates

```
hostname = f"{site_code}-{prefix}-{index:02d}"
# MUC-RTR-01
mgmt_ip = await self.client.allocate_next_ip_address(
    resource_pool=mgmt_pool, identifier=hostname)
device = await self.client.create(
    kind="DcimDevice", data={
        "name": hostname,
        "object_template": {"id": template_id},
        "primary_address": mgmt_ip,
        "member_of_groups": ["device_config"],
    })
await device.save(allow_upsert=True)
```

The template fills the rest

Role, type, platform, and interfaces

From the design

Count and prefix come from the device entry

IP keyed by hostname

A rerun gets the same IP

Register the generator in .infrahub.yml

```yaml
generator_definitions:
  - name: campus_site_generator
    file_path: generators/campus_site_generator.py
    query: campus_site
    targets: campus_sites
    class_name: CampusSiteGenerator
    parameters:
      name: name__value
```

file_path

Where the code lives

query

The GraphQL input

targets

The group it runs for, once per member

parameters

Query variables for each target

Run it locally with infrahubctl generator campus_site_generator on your branch, then push. Infrahub imports the definition from Git.

Federation: one system decides each value

Systems of record

IPAM owns addresses, the asset system owns hardware, and the CMDB owns services

Infrahub Sync

Adapters for NetBox, Nautobot, IP Fabric, and more, plus generic REST. It shows the diff, then applies it.

Governance per domain

Choose direction and frequency for each domain. Each attribute has one authoritative source.

A source of truth that ignores existing sources becomes one more silo.

Every attribute records where it came from

Source

The account or repository that wrote the value. Set automatically by sync or Git.

Owner

The group or account that decides when systems disagree

Protected

Only the owner can edit the value. Not yet fully implemented.

This metadata exists for each attribute and relationship, not only for each object.

Render versus build

Render (transforms)

No side effects

Produces an artifact from data

Safe to run at any time

Build (generators)

Creates or changes data

Today, we run it on a branch

And merge it through a proposed change

Reads render freely. In this workshop, writes go behind a proposed change.

Event rules close the loop

TRIGGER

Site added

A member is added to campus_sites

ACTION

Run the generator

The rule starts the generator definition

RESULT

One proposed change

Devices, checks, and artifacts, reviewed together

Group membership becomes the deploy button. Transforms challenges 03 and 04 cover this.

OpsMill



opsmill.com



github.com/opsmill/infrahub