

Intent guide for network engineers

How to define, read and edit your network intent in Oktavia, in any domain: from the Services consoles or the Configurator, understanding scopes and without skipping any control.

Built by Oktanet

Mexico · 2026

Public document

CONTENTS

1. Who this guide is for
2. The idea: declare, don't type
3. Where you work
4. The domains
5. Scopes: where each value goes
6. Editing from the domain console
7. The JSON you are editing
8. When to use the Configurator
9. State intent
10. Taking the change to the network
11. Recipes by domain
12. Good practices

1. Who this guide is for

For network and automation engineers, at customers or partners, who design and maintain a network's intent in Oktavia. It applies to any domain: Campus, WAN, Security, Management and, at service providers, Transport and Customer Services.

By the end you will know which scope each value belongs in, how to read and edit intent from your domain console, when to use the Configurator, and how to take a change to the network without skipping any control.

You don't need to program. Being able to read JSON helps, and this guide covers what you need.

2. The idea: declare, don't type

In Oktavia you don't write commands per device: you declare how the network should be. That declaration is the **intent**, and it is the source of truth. It has two sides:

	CONFIG INTENT	STATE INTENT
Answers	How it should be configured.	How it should be running.
Example	These NTP servers, these VLANs, these BGP neighbors.	Neighbors up, tunnels encrypted, VLANs present.

	CONFIG INTENT	STATE INTENT
Compared with	The device configuration.	What the device reports.
In the console	Intent tab.	Rules tab.

Editing intent **does not touch the network**. Saving only changes what the network should be, so after saving, an audit will show as failures whatever hasn't reached the devices yet. The change travels through a cycle with human approval ([section 10](#)).

3. Where you work

It is the same intent seen from four places. What you save in one appears in the others.

PLACE	USE IT FOR
Domain console Services → Campus, WAN...	Daily work: see your domain's intent in one table, filter it by site and edit values. This is the recommended starting point.
Configurator Automation → Configuration generator	Declaring a new service or a new scope, guided field-by-field forms, previewing a device's configuration, import and export.
Argos The assistant	Proposing intent in natural language, bringing it in from a device backup, and closing the cycle.
JSON proposal	Large changes prepared outside the platform: upload to the Configurator, review as a delta and apply.

4. The domains

Every catalog service belongs to a domain, and each domain has a console with the same tabs: Summary, Intent, Rules, Compliance, Changes, Workflows and Telemetry. A WAN engineer works in WAN without seeing the rest, and the Scope at the top narrows everything to one site.

DOMAIN	WHAT IT COVERS	TYPICAL SERVICES
Campus	The site network.	VLANs, ports, spanning tree, gateways, DHCP, internal routing, wireless.
WAN	What connects sites.	BGP and its policies, FlexVPN and GRE tunnels, SD-WAN, exit routes, NAT.
Security	Who gets in and what passes.	Firewall, ACLs, zones, administrators and access hardening.
Management	The device baseline.	NTP, SNMP, syslog, DNS, SSH, version and model.

DOMAIN	WHAT IT COVERS	TYPICAL SERVICES
Transport Service providers	The core.	IS-IS, Segment Routing, TI-LFA, BFD.
Services Service providers	What each customer buys.	VPNv4, L3VPN, pseudowires, QoS.

Data center and cloud will join as domains with the same way of working: what you learn here applies just the same.

5. Scopes: where each value goes

Intent is declared once, at the most general scope possible, and devices inherit it. This is the most important design decision: a good scope avoids hundreds of exceptions.

#	SCOPE	APPLIES TO	USE IT FOR
1	defaults	The whole organization.	The baseline: domain, NTP, syslog, banner.
2	by_site	One site.	What is specific to a site.
3	by_site_type	A type of site.	The common design of offices or branches.
4	by_role	A function.	The standard for wan_router, lan_switch, pe...
5	by_platform	A vendor or OS.	What changes by platform.
6	by_model	A model.	Approved software versions.
7	by_tag	Devices with a tag.	Groups that don't follow site or role.
8	by_filter	A combination.	For example, a role within a site.
9	by_device	One device.	IPs, interfaces, neighbors and real exceptions.

Layers apply in that order and **the last one wins**. This is how merging works:

- **Objects merge field by field.** What a scope doesn't mention keeps arriving from above.
- **Lists are replaced whole.** They never add up.
- **"enabled": false** turns the service off for the devices in that scope.

```
defaults          ntp.servers = ["192.0.2.1", "192.0.2.2"]
by_role / wan_router ntp.servers = ["198.51.100.9"]
```

```
wan_router → ["198.51.100.9"]
lan_switch → ["192.0.2.1", "192.0.2.2"]
```

The router receives one server, not three: its list replaced the one above.

Two classic mistakes. First, lists: if routers need the two common servers plus their own, their list carries all three. Second, order: `by_site_type` comes after `by_site`, so if both declare a field the type wins. For a one-site exception use `by_filter` or `by_device`.

6. Editing from the domain console

This is the most direct way to see and change your intent. Open **Services**, pick the domain and go to **Intent**.

Reading the table

There is one row per value, grouped by declaration: **Service**, **Scope**, **Parameter** and **Value**. The scope label says how many devices it wins on:

- **Wins on N devices:** this is the declaration that applies to them.
- **No effect:** a more specific scope beats it on all its devices. Worth reviewing.
- **Disabled:** it has `enabled: false`.

With the **Scope** at the top you pick a site and the table shows only what applies to its devices, each value from the declaration that wins for them, whichever layer it is in. It is the best way to answer "what does this site really have?". The search box finds services, parameters and values.

Changing a value

- 1 Find the service and click **Edit values** on the declaration you want to change. Pick the wrong scope and you will change more or fewer devices than you think.
- 2 Edit the JSON that opens.
- 3 Click **Review the change**. Oktavia validates it and shows each value before and after, and how many devices are affected.
- 4 If it is right, click **Save to the intent**. Compliance rules and the expected configuration are updated.
- 5 Use the button that appears to take it to the network with Argos.

What you need to know. Editing requires being an organization administrator. The console changes declarations that already exist: to declare a new service or use another scope, click **Declare in another scope**, which opens the Configurator. If someone saved while you were editing, Oktavia rejects the change and asks you to reload, so you don't overwrite their work.

7. The JSON you are editing

Each declaration is an object with the service's fields. This is an `ntp`:

```
{
  "servers": ["192.0.2.1", "192.0.2.2"],
  "source_interface": "Loopback0",
  "vrf": "Mgmt"
}
```

A list of objects, such as `vlan_database`, has one element per VLAN. The table shows each in its own row:

```
{
  "entries": [
    { "id": 100, "name": "data" },
    { "id": 200, "name": "voice" }
  ]
}
```

Rules when editing:

- **Write only what this scope defines.** The rest is inherited. Copying inherited values freezes them here, and the standard stops arriving.
- **Adding to a list means rewriting it:** include the existing elements and the new one.
- **To go back to inheriting**, remove the declaration from the Configurator. If a device must not have something inherited, turn the service off with `"enabled": false` or declare the value it should have.

Secrets

Keys are never shown or written in clear text. On screen they appear as `••••`; leave them that way and Oktavia keeps the saved value. For a new key, store it in the credential vault and use its reference:

```
{ "address": "203.0.113.2", "remote_as": 64501, "password": "<<vault:bgp>>" }
```

If you type a secret in clear text, Oktavia rejects the save and asks for the reference.

8. When to use the Configurator

Open the Configurator when you are going to:

- **Declare a service for the first time** or take it to another scope. Catalog forms guide each field.
- **Review where something is declared** across all scopes, with its effective or shadowed status.
- **Preview** the lines a device would receive.
- **Import or export:** JSON proposals, CSV and the full intent.
- **Tidy up:** Standardization lifts what is repeated across many devices to a common scope, and separates what only reorganizes from what changes devices.

When saving, it writes only the fields you touched. And if you are about to override inherited lists, it warns you which ones first: read it.

Bringing in an existing network

You don't have to start from scratch. Argos compares what is declared with what devices have and, reading their backups, brings into intent what is configured but undeclared. It arrives as a proposal to review. Careful: importing is not approving. If a device has a mistake and you import it, it becomes intent.

9. State intent

The **Rules** tab of each console shows how the network should be running, also in one table: rule, result, device, parameter and expected value, with failures first. Pick a site to see its devices and results.

- **Automatic from config:** Oktavia derives it from your intent. If you declare a tunnel, the rule requires it to be up. It isn't edited: it changes when you edit the config.
- **Manual:** you define it in Compliance rules, for example a threshold.
- **Set by hand:** an automatic rule someone set manually; it no longer follows the config.

Configuration rules, one per declared service, are in the fold below. They compare the device configuration with your intent.

10. Taking the change to the network

After saving, the console offers the Argos phrase to close the cycle. Every change follows the same path:

- 1 **Audit.** Oktavia compares intent with the devices and finds what is missing.
- 2 **Propose.** Each difference becomes a ticketed change in the Digital Twin, with its diff and rollback.
- 3 **Review and approve.** A person approves it; it can be tested first without applying.
- 4 **Apply and verify.** A new audit confirms the device matches the intent.

On an IOS XR core the change is a single network transaction with *commit confirmed*: if you don't confirm, the devices roll back on their own.

11. Recipes by domain

Management · Change the NTP servers for the whole network

- 1 In Management → Intent, find NTP. With no site selected you will see all its declarations.
- 2 Edit the defaults one with the complete server list.

- 3 If declarations appear in other scopes, decide with the right people whether they are real exceptions. Those that aren't, remove in the Configurator so they inherit.

Campus · Add a VLAN to a site

- 1 In Campus → Intent, pick the site and find VLAN.
- 2 Edit the declaration that wins on its switches and add the element to `entries`, keeping the existing ones.
- 3 If the VLAN must cross links between switches, also review their trunks.
- 4 Then check in Layer 2 services which switches it lives on and which gateway it ends at.

WAN · Add a BGP neighbor

- 1 In WAN → Intent, pick the site and find BGP. Neighbors almost always live in `by_device`.
- 2 Add the neighbor to the list, with its key as a vault reference.
- 3 Check whether prefix-lists or route-maps are needed.
- 4 After closing the cycle, check the adjacency in Layer 3 services and the path in Routing analysis.

Security · Adjust an ACL

- 1 In Security → Intent, find the ACL. If it is in a common scope, changing it affects all the devices the label shows.
- 2 For a single device, leave the common one alone: declare it in its `by_device` with the complete list.

Services (provider) · Give a customer bandwidth

- 1 In Services → Intent, edit `l3vpn` or `l2vpn` and add `bandwidth_mbps` to the customer's site or circuit.
- 2 With the `xr_qos` service declared on the `pe` role, Oktavia derives the policy.
- 3 Take the change to the core as a network transaction and review the service in Customer services.

12. Good practices

- **Start with the most general.** Go down a scope only when there is a reason.

- **by_device** is for what belongs to the device: IPs, interfaces, neighbors and agreed exceptions.
- **Design by site type** so new sites are born the same.
- **Read "how many devices"** before saving. If the number surprises you, don't save.
- **Review anything marked "No effect"**. A declaration that wins on no device confuses the next engineer.
- **No secrets in clear text**. Everything through the vault.
- **Standardize before hardening**. If you harden with everything in `by_device`, the role standard never exists.
- **Verify with compliance, not memory**. A change ends when the audit confirms it.

To go deeper (in Spanish): [when to use each scope](#), the [Service Vars](#) reference and [domain consoles](#).

About Oktanet

Oktanet is a Mexican network automation company focused on controlled, traceable, and scalable operations for enterprise environments in Latin America.

For a demonstration on your own environment, reach us at oktanet.io.

[LinkedIn](#) · [YouTube](#)

© 2026 Oktanet · Oktavia · Public document, with no confidential customer information.