

...because copy+paste isn't a network configuration strategy

Automation for Amateurs with Netbox and friends

**JAMES
MORGAN**

TECHNICAL MANAGER

**MITCH
BLAIR**

DEVOPS ENGINEER

A bit about VERNet...

- › Est 2004 serves the university, health government and research sectors
- › Owns and operates a fibre network spanning Victoria
- › Ciena optical line systems
- › Juniper layer 2/layer 3 overlay
- › Only ~15 staff company-wide
- › A big network for not many people to look after
- › If we can do this, you absolutely can

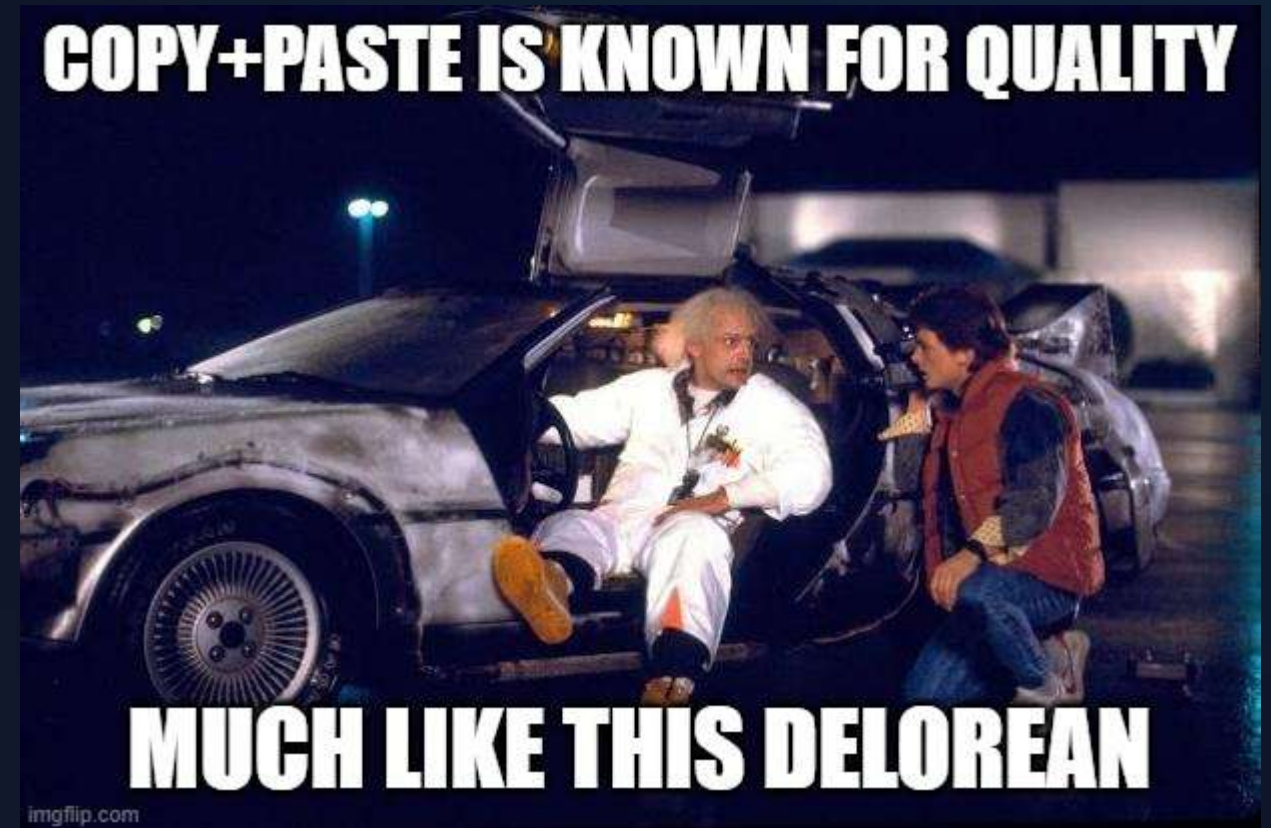
friend - how many people work at your company ?

me - about half of them



But first, a trip back to 2019...

- › Configs lived on devices
- › Sometimes they were backed up, but not always
- › New configuration was derived from copy+pasting other configurations
- › IPAM was a classy spreadsheet
- › Inventory is a less classy spreadsheet
- › Everyone gets read/write

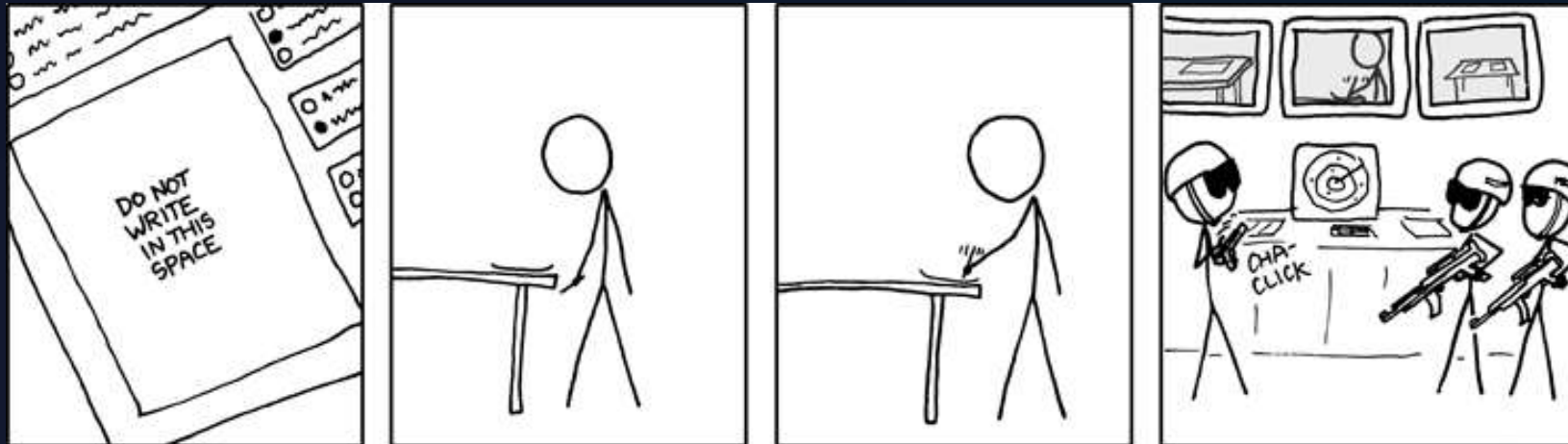


But, does it matter?

- If you lose a node, how will you configure the replacement? Who will do it?
- If you're going to copy+paste, how can you be sure you're not just propagating sub-optimal config, or configuration that doesn't address the problem you're trying to solve?
- If something doesn't work, how often are you asked to "check your configuration"?
- Are you spending productive time of skilled employees reviewing relatively benign changes?
- How can you work at scale if everything is bespoke?

So we tried a few different things...

- > 'Golden' configuration snippets
- > 'Configlets'
- > Automated configuration backups
- > Configuration audits
- > New processes
- > Forms
- > Public floggings



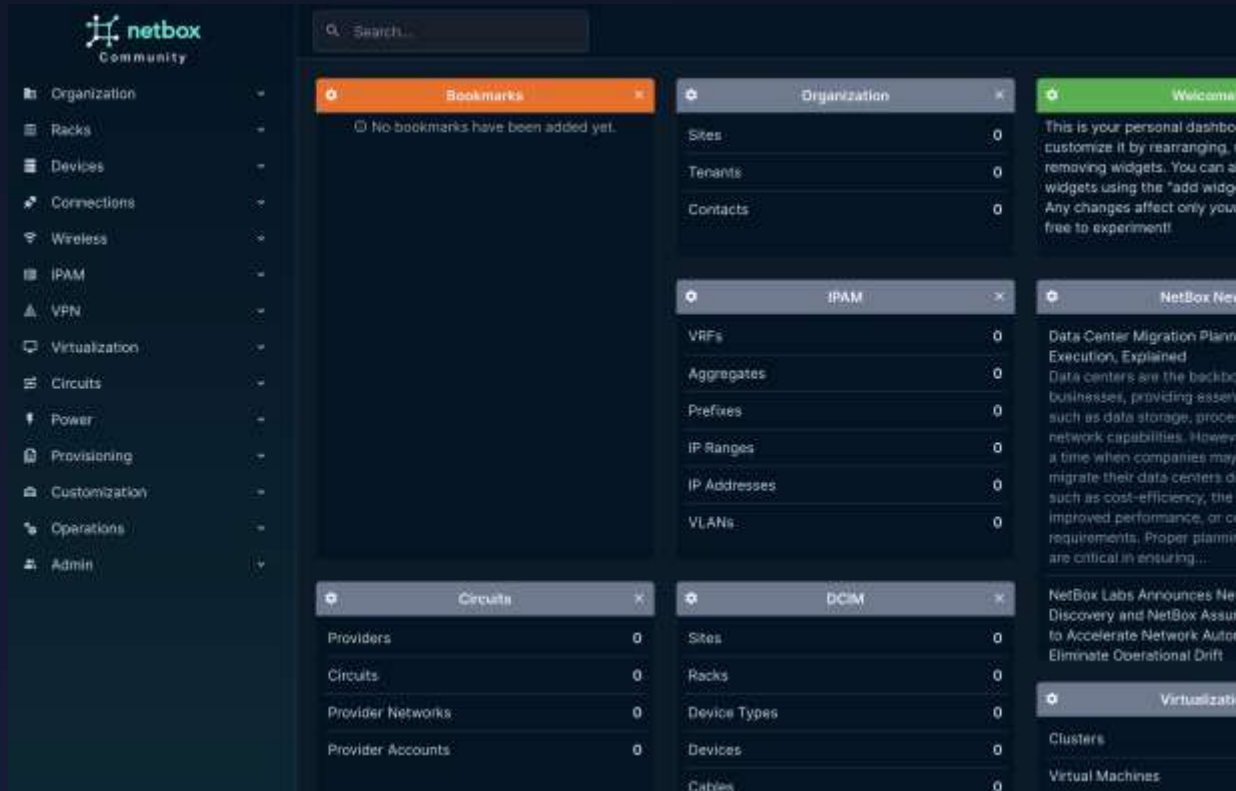
This wasn't working.

- Things took more time to get done
- People became afraid of making mistakes, and therefore avoided situations that put them at risk of it
- We had made more documentation, but the documentation was transactional rather than about state
- People worked out that leaving old/unused config in-place was lower risk than actually removing it
- Not only was it not working, *we had started to lose sight of what 'working' would even look like any more*

So we tried to imagine what 'working' was:

- › Configurations do not live on the hardware, and the hardware config is disposable
- › The configuration is end-to-end templated, and templates filled by automation from records
- › No old or orphaned configuration
- › The records would have a direct relationship to service records in our OSS
- › We'd have a full change-log with audit information
- › Nobody would ever write a config on-box ever again: the automation would push it

So, where should we store state?



Scary stuff: a blank Netbox install

- Introducing; Netbox!
- FOSS, released in 2016
- Modern approach to older IPAM/DCIM solutions
- Mostly Python
- Heaps of APIs, good data model, well used across industry
- All the kool kidz are doing it, yeet!

But wait, what about our existing state?

- > ~~phpIPAM~~
- > ConnectMaster
- > OSS
- > ~~The hardware~~
- > ~~Excel~~
- > ~~Visio~~



So, what goes where?

ConnectMaster

"The Big Stuff"

- › Fibre plant (joints, cables, patches, panels)
- › Optical transmission plant
- › End-to-end pathing

Netbox

"The Tech Stuff"

- › IPAM
- › L2/L3 Hardware
- › OOB
- › Rack Diagrams
- › Configuration Metadata

The OSS

"The Admin Stuff"

- › Customer Records
- › Service Records

Step 1: Migrate, then eradicate

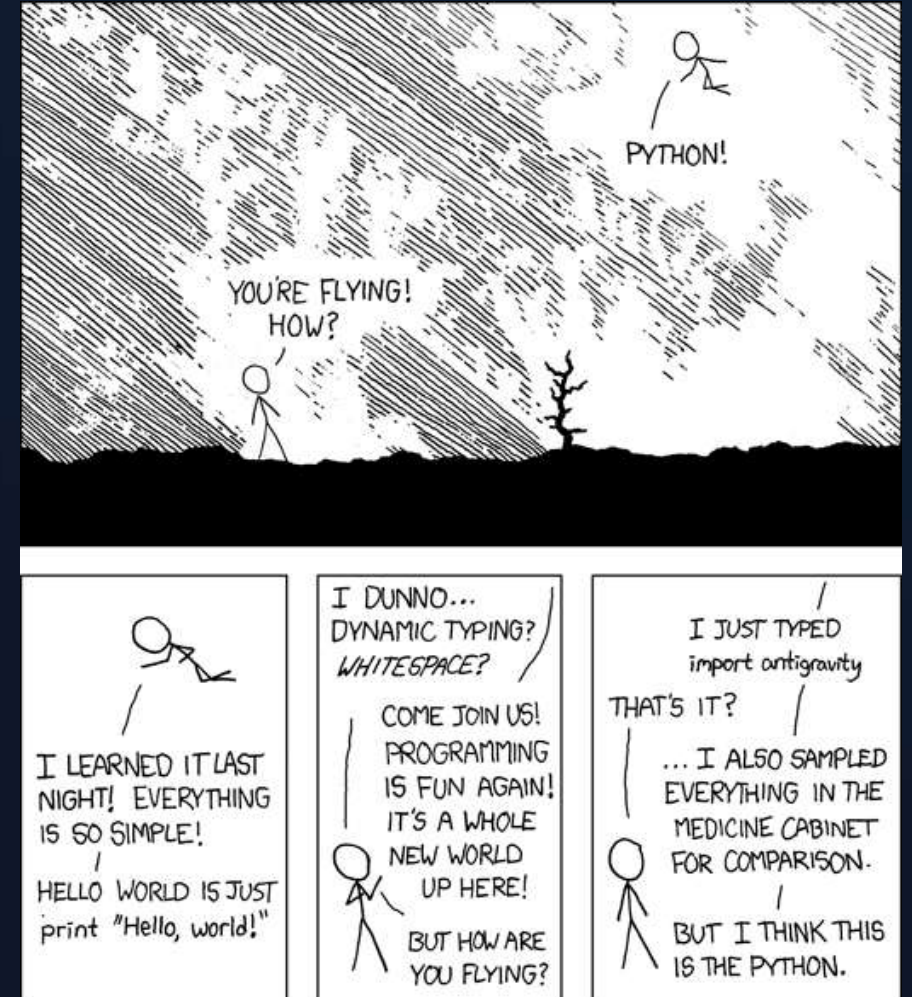
- Import data from phpIPAM and then set it to read-only with a warning banner, followed by disabling it
- Bad because you end up with a lot of orphaned IPs
- Good because it's an easy intro to Netbox for the team
- Import device list from ConnectMaster to bootstrap us



Step 2: Import the state to Netbox?

- The state lives on the boxes
- It's full of old stuff that shouldn't be there
- A lot of it isn't even right
- How would we even get it anyway?

No problem, Mitch knows Python, right?



Step 2A: Import the reliable* bits of state



- The physical layout and inventory of the devices is reliable
 - Except when it's not (misreporting, 'value-add services')
- The way the IGP and core links work is reliable and known
- ...wow, that was less than we thought
- Script this, and keep wiping the db every time you re-run the script until it's right!

Step 3: Build a golden configuration

- There was stuff in our configs that we didn't internally know the purpose or workings of but nobody questioned it
- There were solutions to problems that nobody could remember, or even whether they worked at all
- Start from a perspective of a box with no services, and then consider what operational roles dictate extra sections of config



Step 4: Define your products/services

- We have a lot of services, but they are all instances of various classes
- Lots of them were configured differently; were they different classes, or were they the same class with varying properties?
- If you turn all of the differences into properties, you end up with a LOT
- Consider how many of those can be inferred (eg; could you amend your route targets to something more predictable/automated?)

Step 5: Model services

- Now, we build the bridge between our OSS and Netbox
- Create script modules to create/remove customer and service instances in Netbox
- Your scripts are going to come in handy later when you're running this in production
- By the end of this step Netbox should substantially mirror the operational state of the modelled device, but... better!

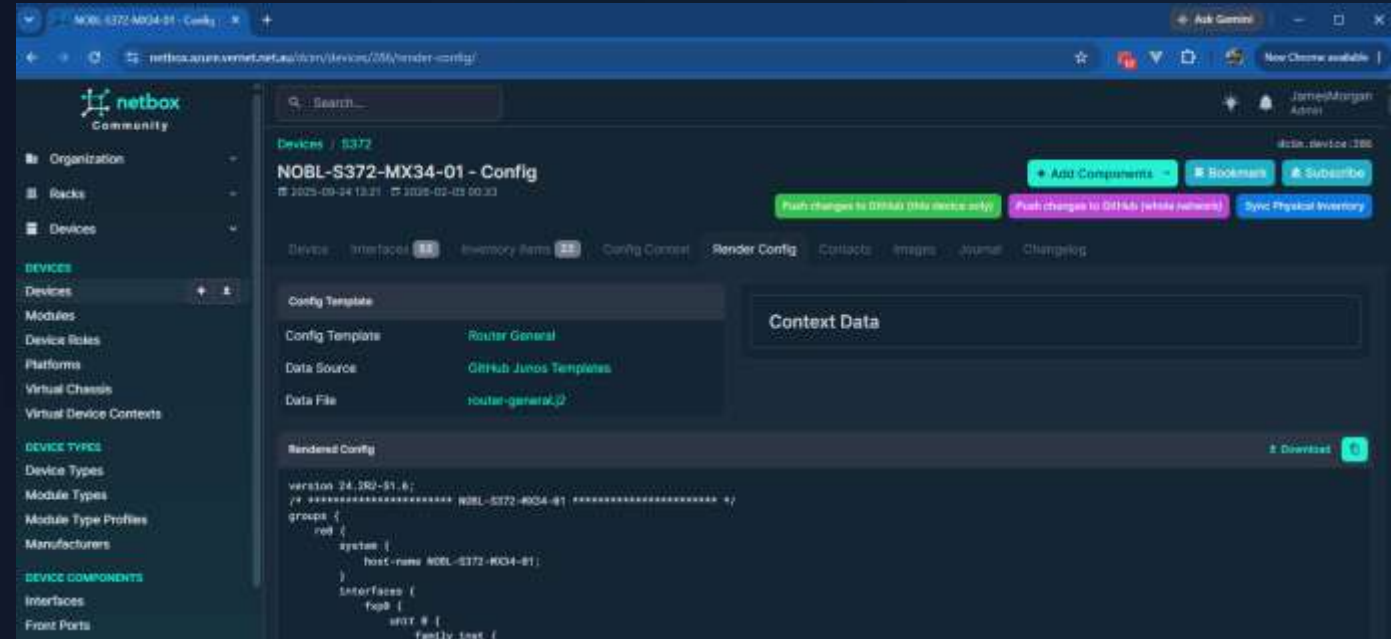
Step 6: Config templating

- This is the fun part where we take our golden configuration and start telling Netbox where to put the variable data
- We also need to take all of our service/role classes and create config templates for them
- Result is going to be a large jumble of mustaches, loops and IFs



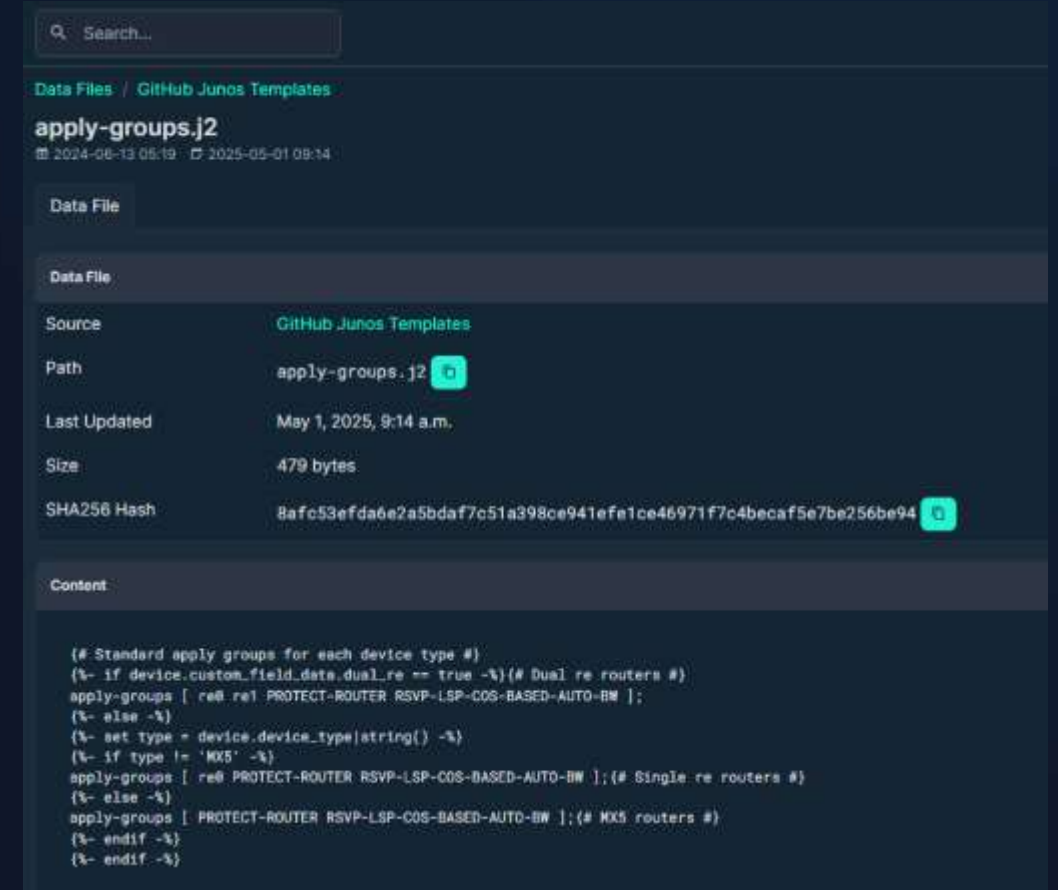
Wait, templates do what now?

- You'll need a git repo to keep your jinja config templates in
- Templates can inherit others
- Netbox can be configured to sync these templates via Git
- Netbox supports rendering the config itself using templates
- Netbox can generate and push configs into a Git repo via PRs



Sounds inconvenient!

- Templates can be very simple, or can be several-hundred line monstrosities
- Moving forward the templates will be where your network engineers spend their config time
- The system is fantastic for making mistakes at scale – so convenient!
- Now that we have templates and data...



The screenshot shows a file management interface for a file named `apply-groups.j2` located in the `GitHub Junos Templates` repository. The file is identified as a `Data File`. The interface displays the following metadata:

Data File	
Source	GitHub Junos Templates
Path	apply-groups.j2
Last Updated	May 1, 2025, 9:14 a.m.
Size	479 bytes
SHA256 Hash	8afc53efda6e2a5bdaf7c51a398ce941efe1ce46971f7c4becaf5e7be256be94

Below the metadata, the file's content is displayed as a Jinja2 template. The content defines standard apply groups for different device types, using conditional logic to handle dual-re routers, single-re routers, and MX5 routers.

```
{% Standard apply groups for each device type %}
{%- if device.custom_field.data.dual_re == true -%}{% Dual re routers %}
apply-groups [ re@ re1 PROTECT-ROUTER RSVP-LSP-COS-BASED-AUTO-BW ];
{%- else -%}
{%- set type = device.device_type|string() -%}
{%- if type != 'MX5' -%}
apply-groups [ re@ PROTECT-ROUTER RSVP-LSP-COS-BASED-AUTO-BW ];{% Single re routers %}
{%- else -%}
apply-groups [ PROTECT-ROUTER RSVP-LSP-COS-BASED-AUTO-BW ];{% MX5 routers %}
{%- endif -%}
{%- endif -%}
```

STEP 7:

LET'S BAKE SOME CONFIG





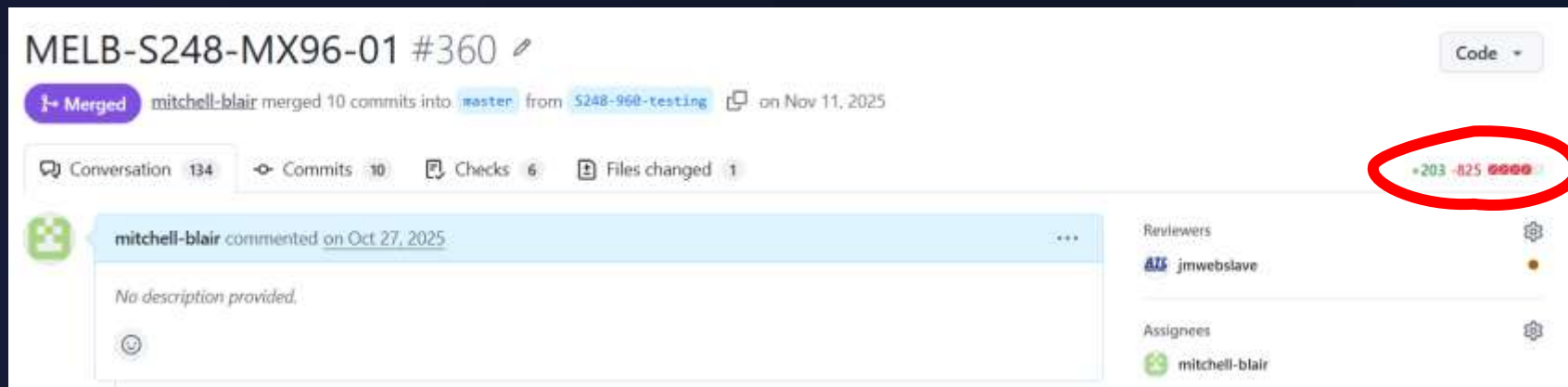
Rules of baking your config:

- The first time you generate a config it'll error out
- The second time you generate it it'll be six lines long

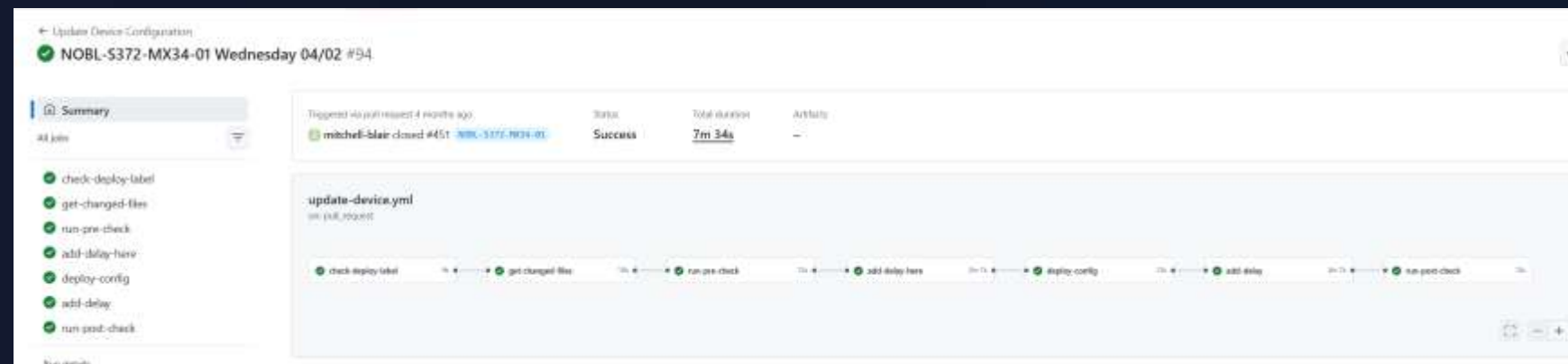
- By about the 12th generation you'll get a full config
- This config will not pass validation on your lab box
- You will suspect this is because you're running a different software version on your lab box, but several up/downgrades will prove this false
- You'll learn more about your config at this stage than all the rest combined

Step 8: Generate!

- > It's now time to generate configs for all of your hardware from Netbox
- > You're going to find hundreds of differences
- > You're going to spend the next few months finding edge cases where your service classes don't have the right properties
- > You're also going to find embarrassing config mistakes in the hardware
- > Nobody will own up to making these
- > You'll do this hundreds of times over, so make scripts to help you early on!



How do these
configs make
it onto the
box?



You'll know it's time when...

- > You can now generate configs out of Netbox that multiple people can look at and not find errors
- > You can pick an important node and replace the config on it with the generated config in a works window... and not have to roll it back
- > You don't have to make any 'manual tweaks' to the generated config

In which case...

Book a series of works windows and spread your new configurations!

Now you need to keep things in-sync

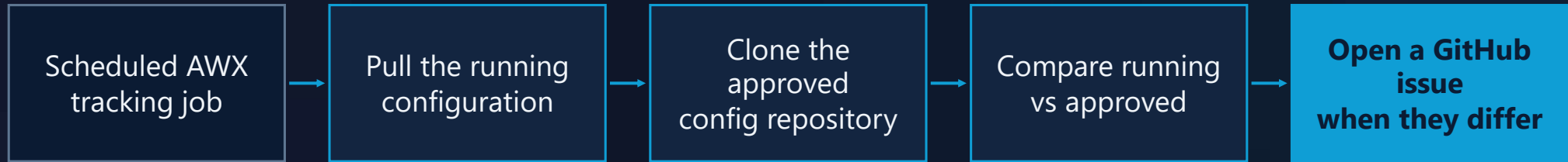


- › This is where all of the scripts you created when you were initially populating data come back to haunt you
- › Decide what parts you want to automate, and what parts you want humans involved in
- › The approach we took won't work for everyone; we haven't decided if it works for us yet

01 INVENTORY SYNC



02 DRIFT TRACKING



03 GENERATE CONFIG



04 TEMPLATE EDITS

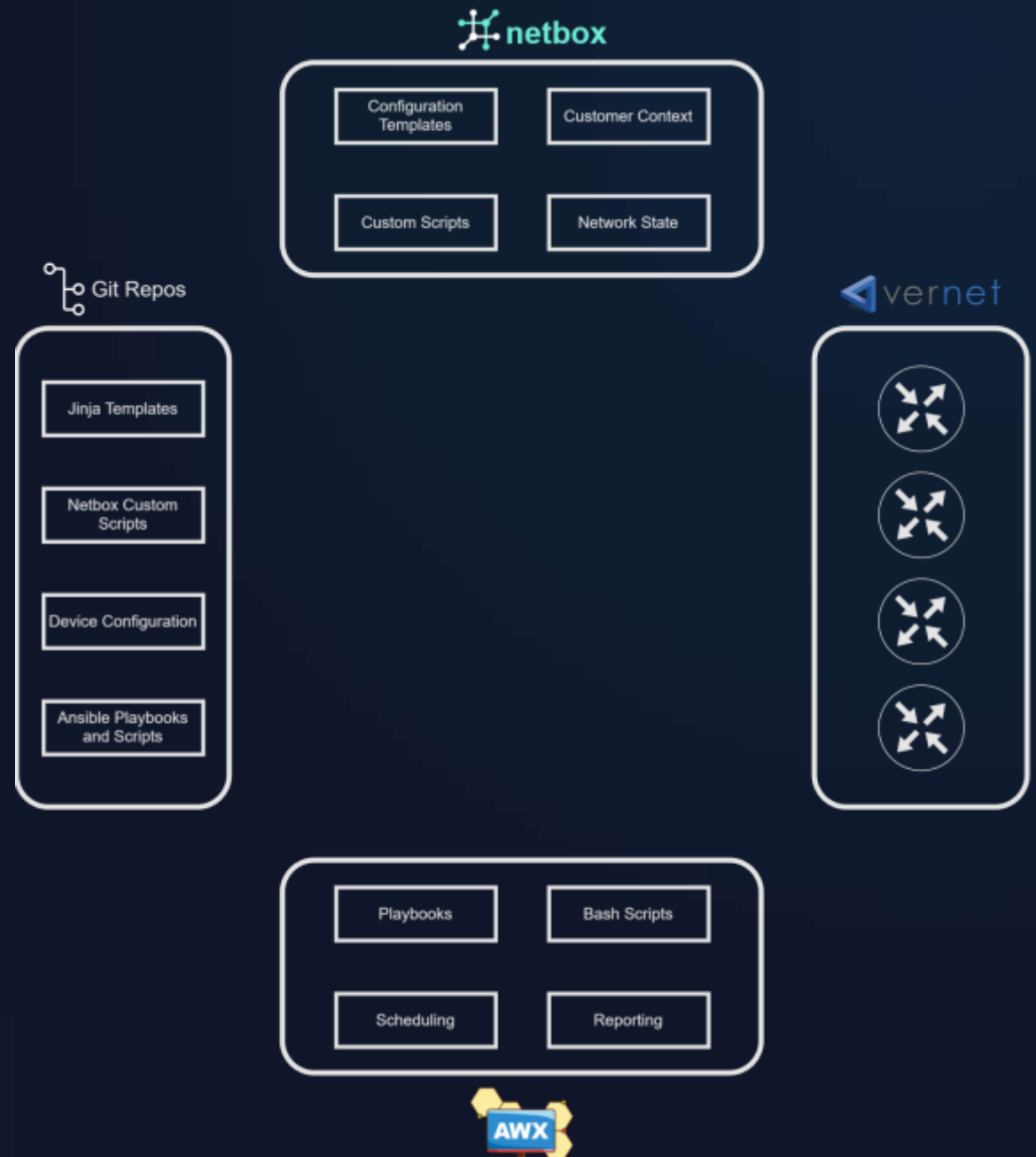


Why is any of this better?

- › Provisioning new boxes takes minutes
- › You move away from reviewing config changes, and instead review template changes
- › With enough validation you eradicate fat-fingers
- › You don't need access to the box to know how it's configured
- › You understand every line of config that's there
- › Config can't persist for no reason
- › You don't waste time with the old "check your configuration"
- › Fewer people need write access

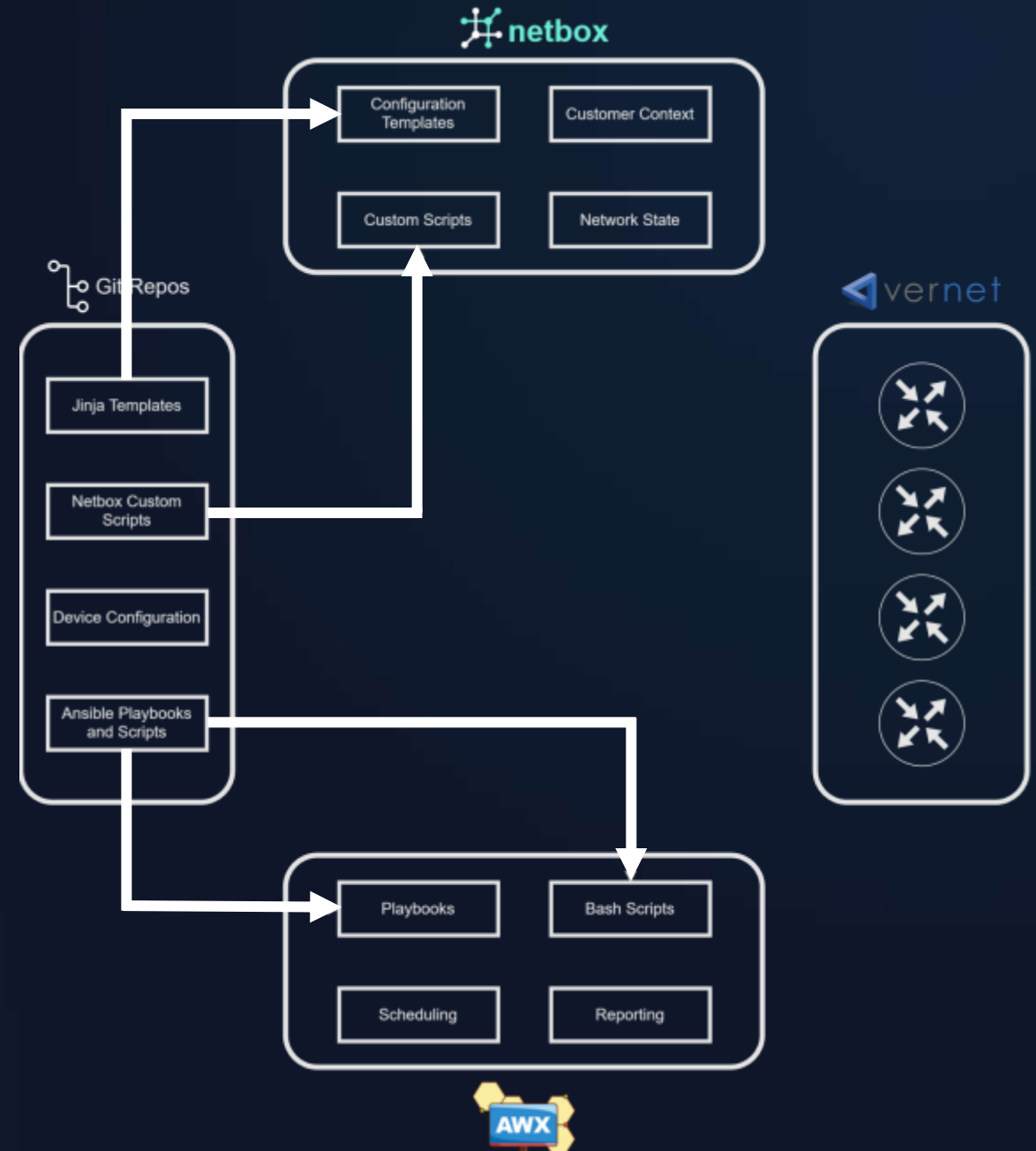
Finished Architecture

- Git repos for templates, custom scripts, Ansible playbooks and approved device configurations
- AWX controls the major workflows and processing for the system
- Netbox contains all network state and customer contexts



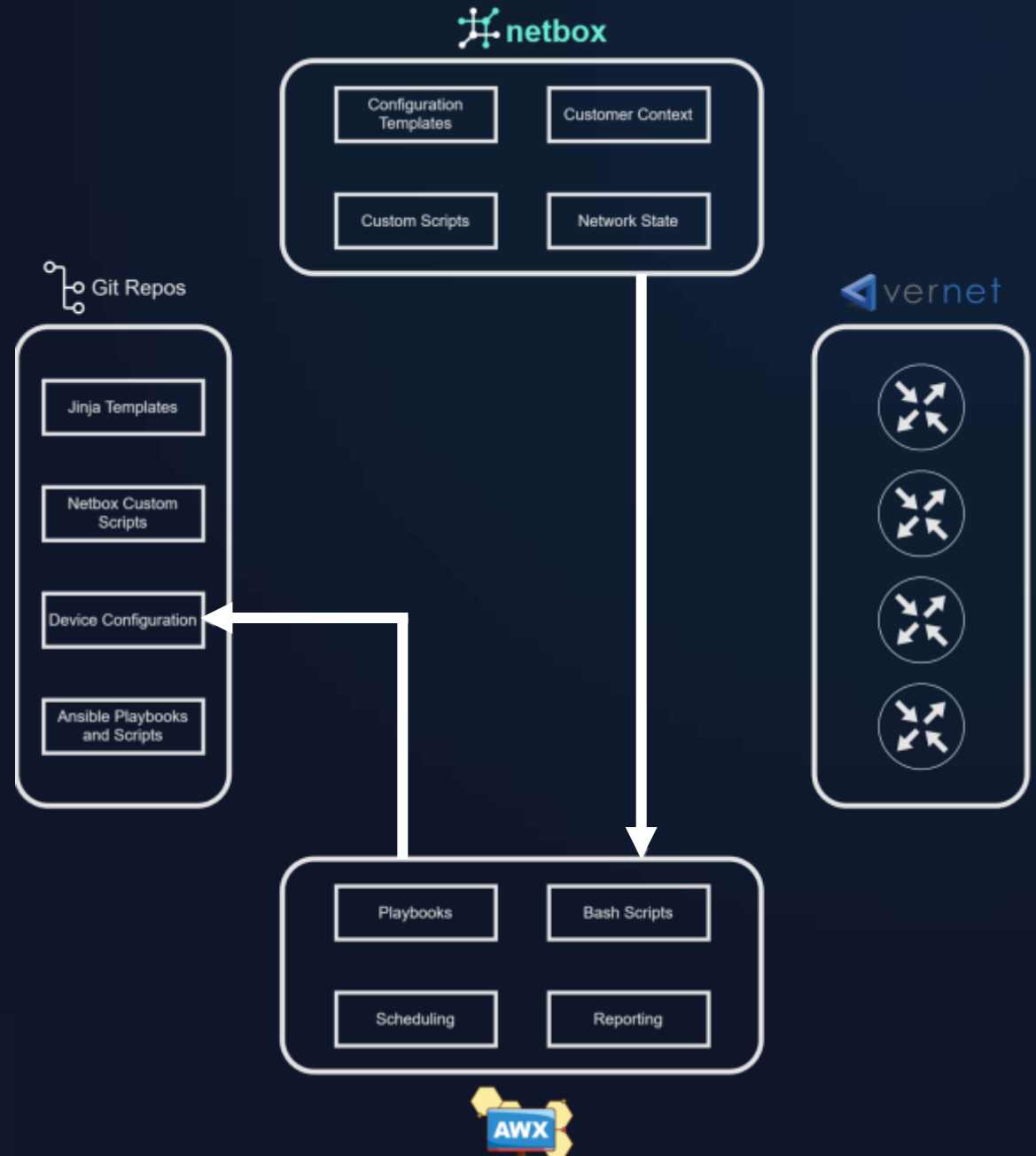
Version Control

- > All templates and scripts are version controlled with approvals in place for changes
- > Any updates to the main branch of each repo will trigger a GitHub action to push changes
- > GitHub actions will update Netbox data sources and AWX projects via API calls



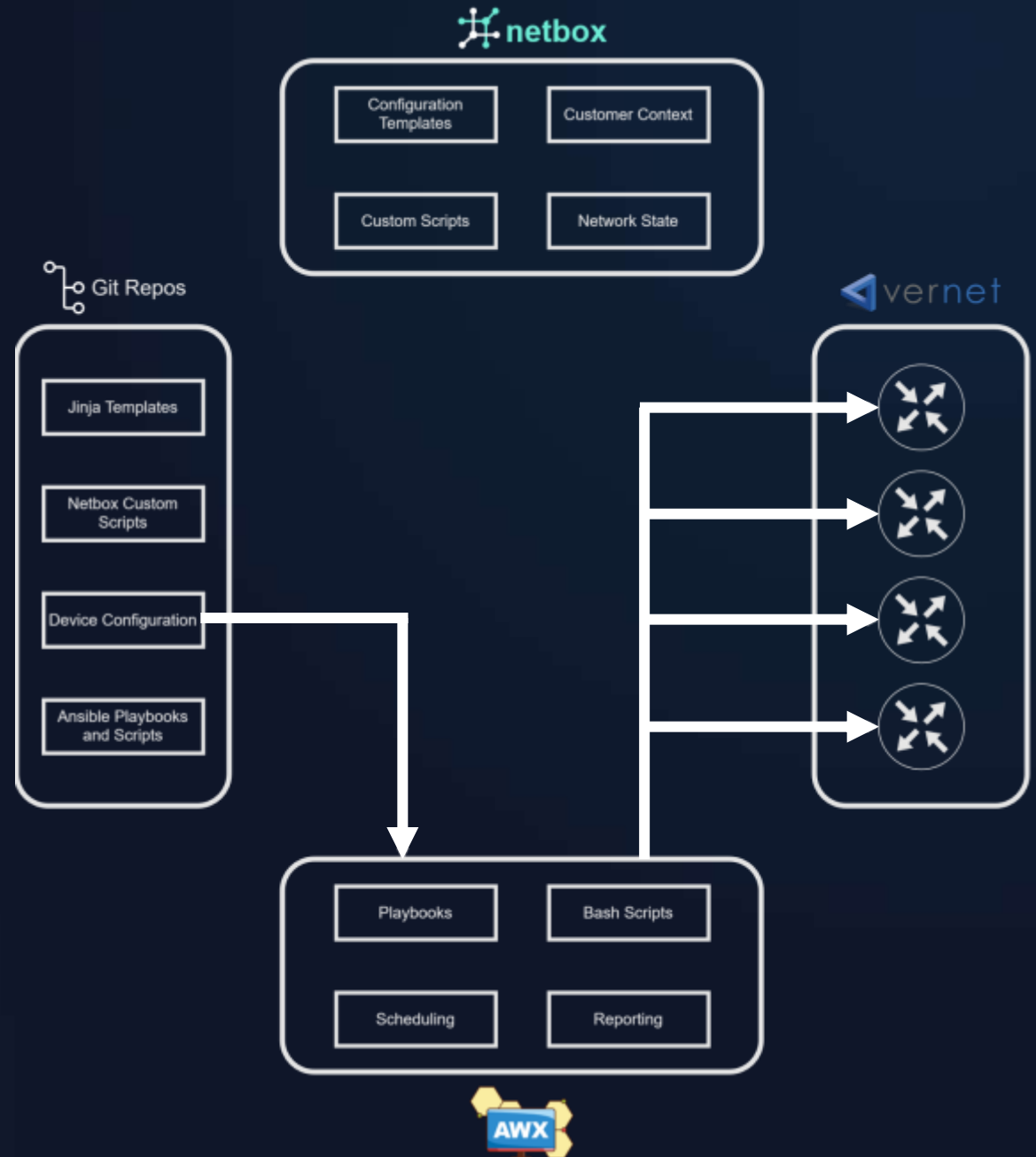
Config Generation

- > Device updates in Netbox can be triggered manually or by automation to generate a new configuration
- > A playbook obtains the full device configuration from Netbox, while also cloning the Device Configuration repo.
- > A bash script will compare the 'approved' configurations with the incoming configurations from Netbox
- > If any changes are detected, a pull request is raised for approval



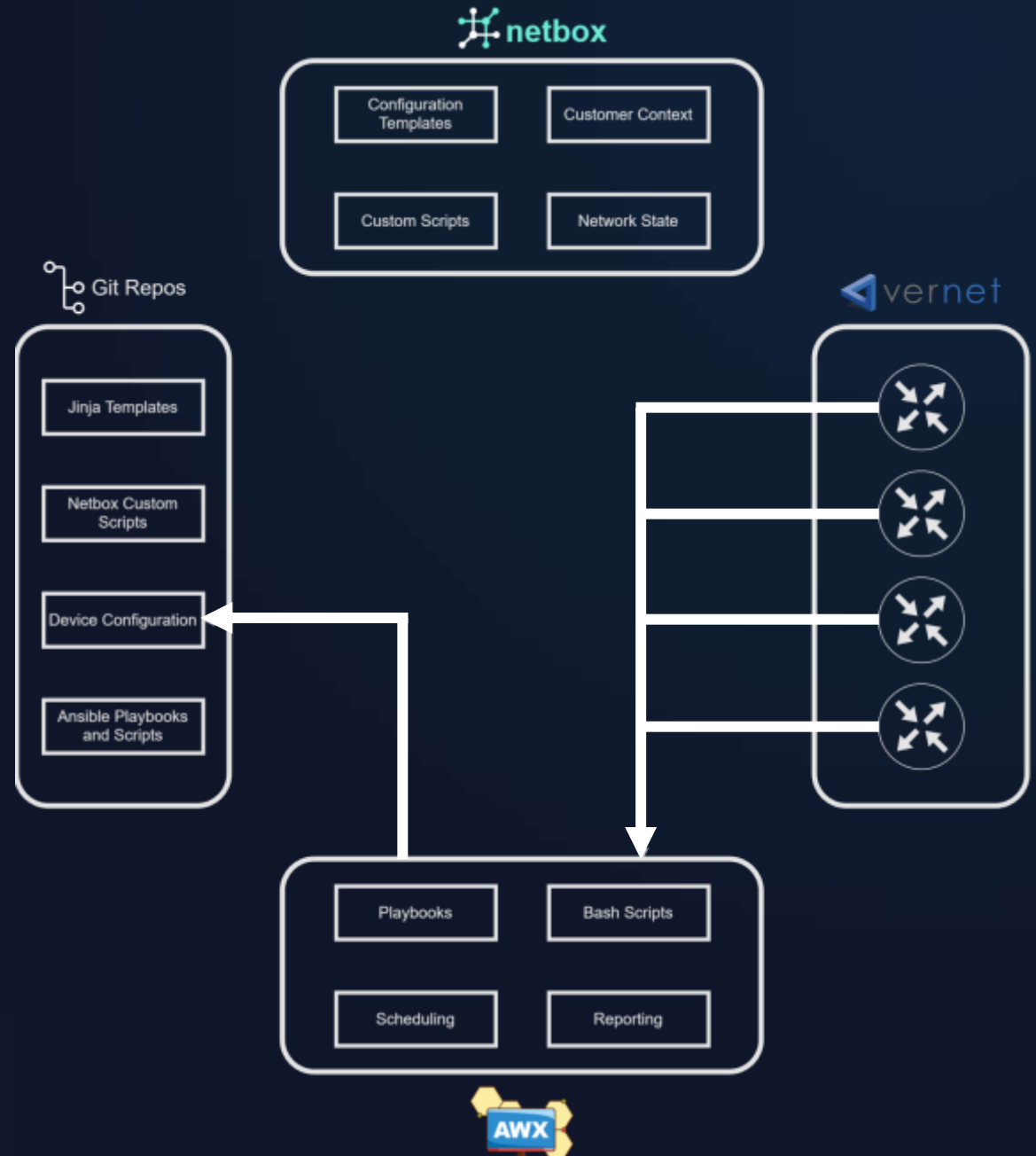
Config Deployment

- > The full device configuration is validated on box before the PR can be merged
- > On PR merge, GitHub actions is used to trigger multiple playbooks:
 - Current state of the device is obtained prior to update
 - Approved configuration is loaded and a commit confirm is performed
 - Commit confirmed after a set period
 - Post state of the device is obtained
 - Any diffs in state is reported in a dedicated group discord channel



Config Tracking

- Running device configurations are regularly pulled by scheduled AWX jobs
- Bash scripts compare the running configs to the approved configs in GitHub
- Any differences found will trigger an issue to be raised in GitHub



Where to next?

- > Ciena provisioning?
- > DNS
- > AI hooks to annotate Github PRs
- > Netbox branching
- > Sync edge-cases
- > Race conditions

Thanks for listening!

- > Scan the QR code here for a Github repo that will (hopefully) give you a lab environment with:
 - Netbox
 - AWX
 - Ansible
 - Sample netbox/config data
- > Please feel free to find Mitch or myself around the conference if you have any questions or want to tell us how we should have done it.

