Zero-Touch HPC Nodes

NetBox, OpenTofu and Packer for a Self-Configuring SLURM Cluster

Ümit Seren (GH: @timeu) & Leon Schwarzäugl (GH: @swarsel)

Vienna BioCenter - Scientific Computing

FOSDEM 2026 - Feb 01, 2026 - Brussels

The Old Way: Manual HPC Deployment

*Our previous HPC cluster: OpenStack-based with Ansible automation for the **payload (SLURM cluster)**... but the **underlying infrastructure** was still managed entirely by hand (See our FOSDEM 2020 talk "HPC on OpenStack...")*

Technology Stack

✓ Automated (Payload):

- OpenStack for compute resources
- Ansible for SLURM cluster config

✗ Manual (Infrastructure):

- SDN Network configuration
- BMC IP/DHCP management
- Excel-based inventory
- Storage configuration

The Pain Points

⚠️ Long ansible runtime
⚠️ Manual BMC reconfiguration (200+ nodes)
⚠️ Configuration drift between environments
⚠️ No reproducible rebuild process
⚠️ Tribal knowledge and technical debt
⚠️ Risky upgrades with no rollback

Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

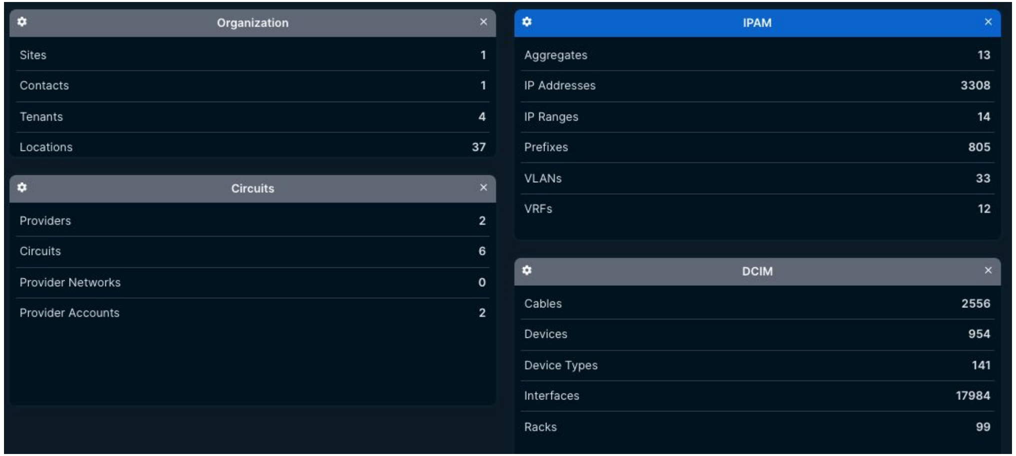
NetBox: DCIM and Source of Truth

NetBox serves as our single source of truth, feeding all automation workflows with accurate, version-controlled infrastructure data.

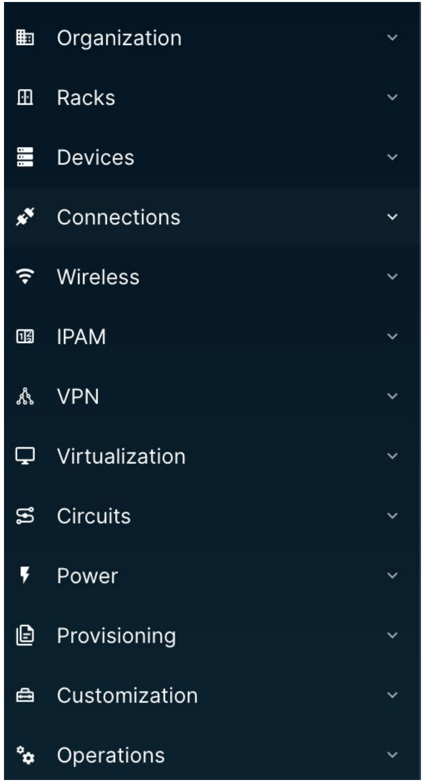
What NetBox Provides:

- 📍 IPAM: Complete IP address management and DHCP reservations
- 🖥️ Device Inventory: Servers, switches and storage devices with network connections
- 🔌 Physical Layer: Rack locations, cabling, patch panels, port assignments
- 🏷️ Metadata: Custom fields for network provisioning
- 🔄 Drift Detection: Continuous validation against external systems and appliances
- 🔗 API-First: REST API & custom export templates for OpenTofu, and custom automation

Scale: 100 racks, 150+ switches, 300+ nodes— all tracked in NetBox



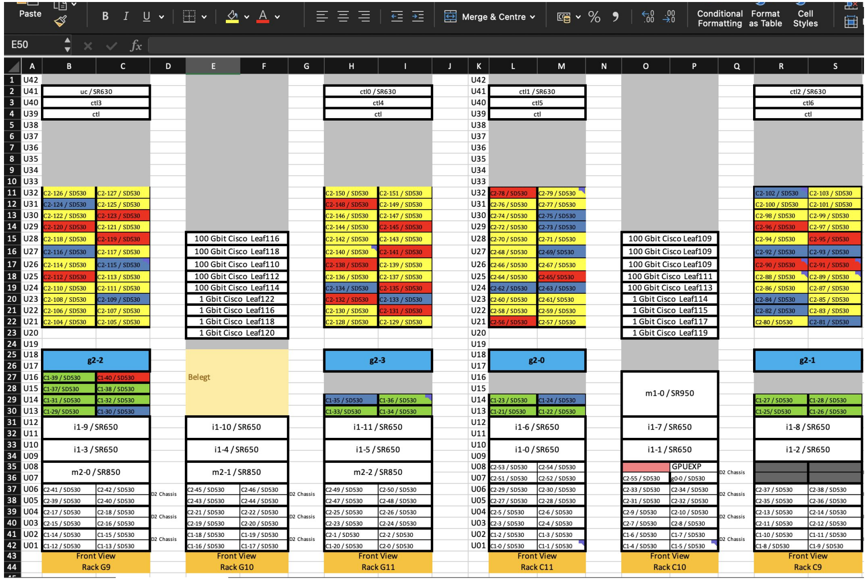
Netbox Feature Overview



NetBox: DCIM and Source of Truth

From Excel to Netbox

Before



After

NetBox Integration: External Systems

NetBox serves as the **central source of truth**, integrating with existing vendor appliances and IPAM systems through **custom drift check scripts** and **custom sync scripts** that ensure data consistency.

Our External Systems

Vendor Management Appliances
Lenovo XClarity, Dell OpenManage Enterprise

IPAM System
Infoblox

Network Infrastructure
Cisco ACI & LLDP Topology Discovery

Drift checks

Custom Drift Check Scripts

Automatically compare NetBox data against vendor appliances

- Verify hardware inventory (serial numbers, models) are correct
- Validate IP address assignments against Infoblox
- Check network topology via LLDP data against SDN
- Internal checks (duplicate IP/MAC, orphaned cables, etc)

Import Automation

Vendor Management:
- MAC addresses
- Serial Numbers

IPAM system:
- IP Addresses
- IP Ranges/Networks

Network SDN:
- Serial numbers
- MAC addresses
- LLDP neighbours

Export Automation

IPAM system (OpenTofu):
- DHCP reservations
- Host entry

Network SDN (Tofu):
- Fabric side switch port configuration

OpenStack (Export Templates) :
- Hypervisor & Controller
- Baremetal machines

Storage system (OpenTofu):
- Network Configuration for SVMs

NetBox: Complete Infrastructure Visibility

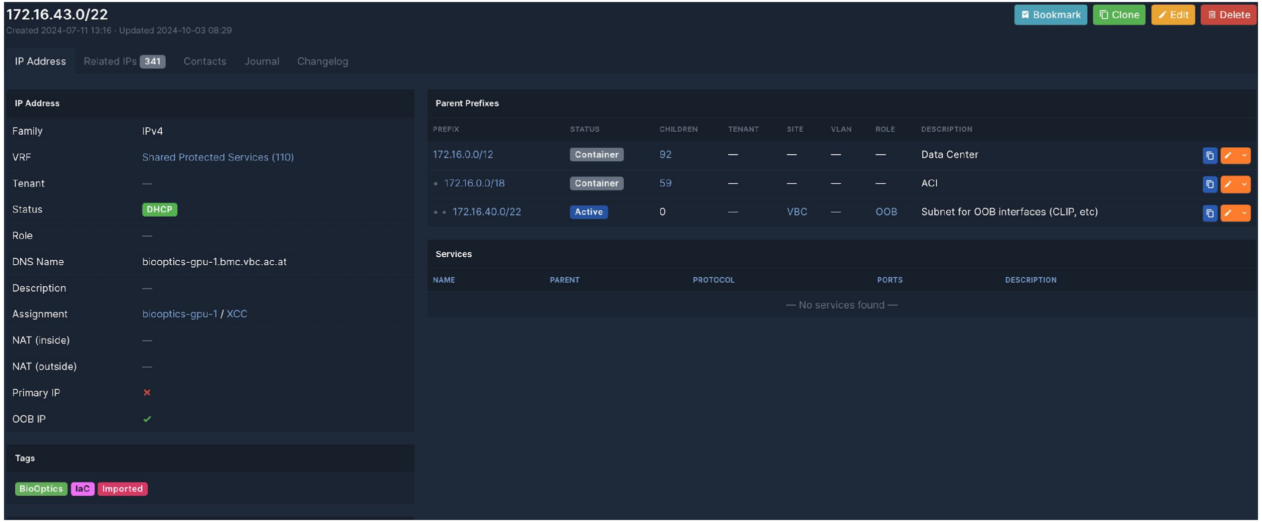
Once all infrastructure data is populated, NetBox provides **comprehensive insights** and **end-to-end traceability** across the entire datacenter stack.

IP Address Traceability

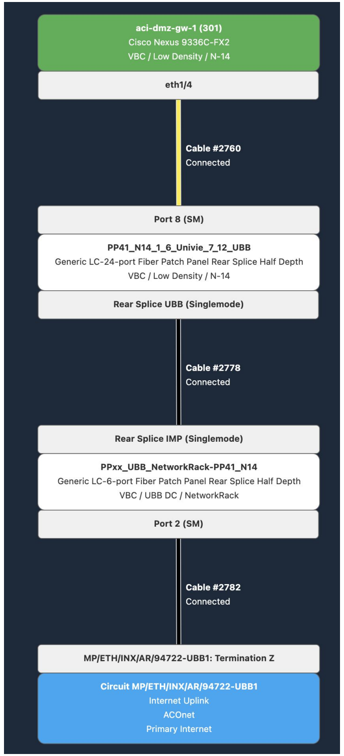
IP to physical port

1. IP: 10.0.5.42 — Compute node interface →
2. compute-node-025 — Rack A-12, U20-U22 →
3. eth0 ➜ Cable #A025 — Physical cable path →
4. PP-A-12 Port 24 — Patch panel →
5. tor-a12 Gi1/0/24 — Physical switchport

NetBox Interface



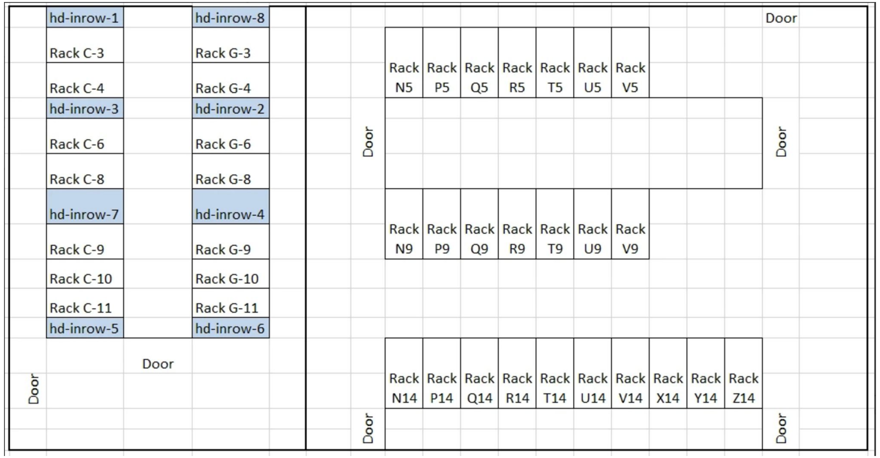
Cabling tracing



NetBox: Floorplan plugin

Netbox as a comprehensive documentation tool of physical space

Before

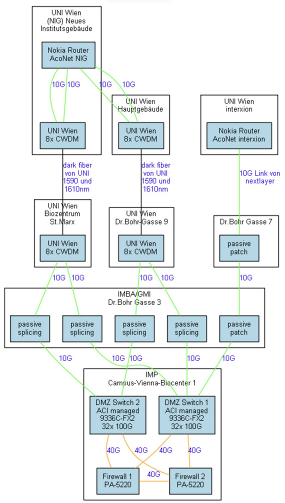


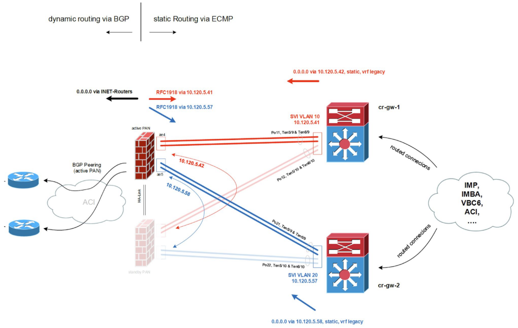
After

NetBox: Topology plugin

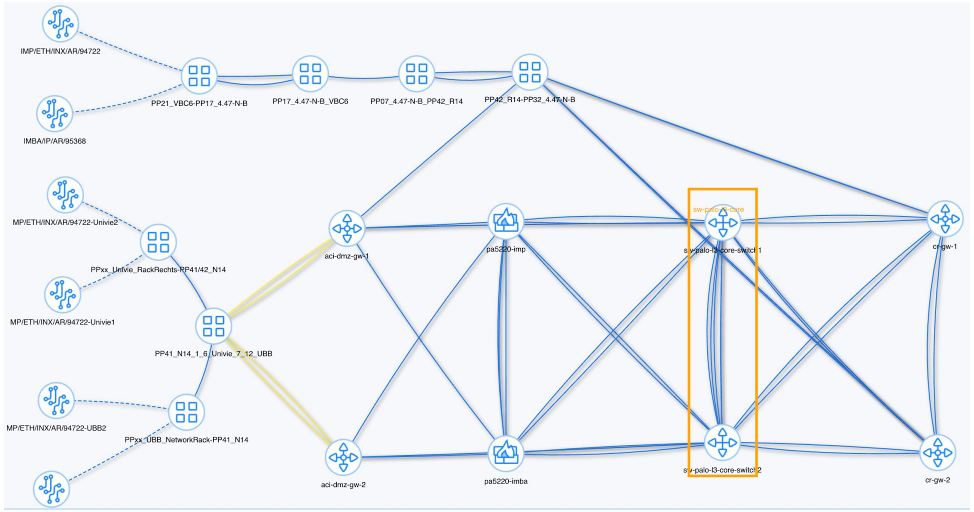
*Netbox as a comprehensive **live** documentation tool of logical network connections*

Before



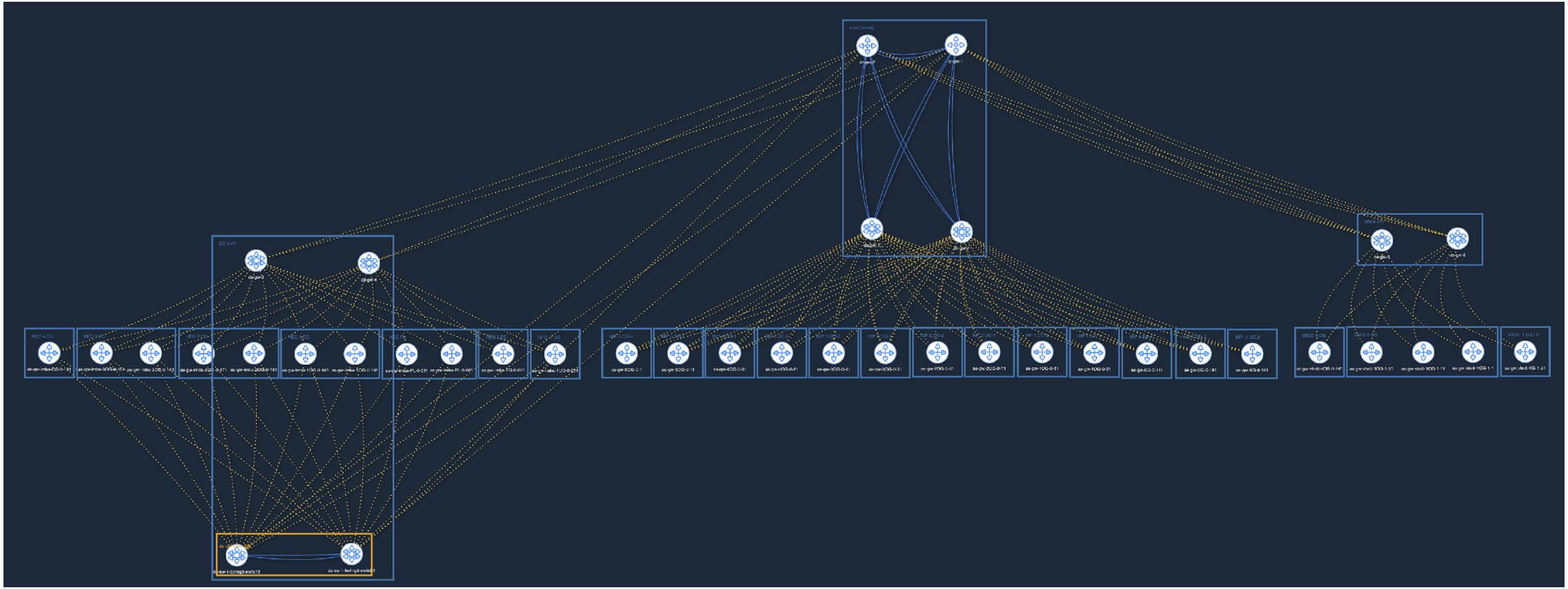


After



NetBox: Topology plugin

*Netbox as a comprehensive **live** documentation tool of logical network connections*



Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

Immutable Images

Why Images?

✓ Identical nodes from the same image
✓ Fast provisioning
✓ Reproducible
✓ Versioned image artifacts
✓ Safe rollbacks to previous versions

Idea

- Take an upstream base OS to customise
- The build system should be agnostic of the image distro/version
- The system should automatically push the image to all endpoints

Immutable Images using Packer

Packer

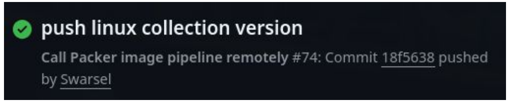
- ✓ Support for many platforms
- ✓ Similar syntax to OpenTofu
- ✓ Support for a wide range of image customization tools

```
packer {
  required_plugins {
    docker = {
      version = ">= 1.0.8"
      source = "github.com/hashicorp/docker"
    }
    ansible = { <…> }
  }
}

source "docker" "ubuntu" {
  image  = "ubuntu:questing"
  commit = true
}
build {
  name    = "packer-example"
  sources = ["source.docker.ubuntu"]
  provisioner "ansible" {
    playbook_file = "${path.root}/playbook.yml"
  }
  post-processor "shell-local" {
    only = ["docker.ubuntu"]
    inline = ["echo 'Done!'"]
  }
}
```

Immutable Images – Our Implementation

1. Call GitHub workflow either from base repo or remotely
2. This sets up packer with the respective vars for the distribution/version
3. Packer boots temporary VM from the base OS image using QEMU
4. Ansible configures the image
   4.1. Additional customizations run if called from a remote repository
5. Cleanup scripts run
6. Output versioned image
7. Upload to various endpoints



Ansible Roles: Install vs Configure

The situation

- Not all customizations can be performed at image build time
- We manage most reusable tasks using Ansible roles
- Most of these roles perform steps that will be used on all roles unconditionally
- Many of these will then perform steps that should not be put in the image

```
- ansible.builtin.include_role:
    name: sssd
    tasks_from: install.yml
```

Our solution

- Split roles into distinct `install` and `configure` tasks

Install Roles (Baked into Image)

When: Run during Packer build (once)

Purpose: Install system-wide software and configs during image build.

What Goes in Install Roles:

- Package installations
- Setup of directories & propagation of files
- Configuration common to all nodes

Configure Roles (Run at Boot)

When: Run by a local service on first boot.

Purpose: Perform tasks that are not common to all hosts.

What Goes in Configure Roles:

- Node-specific configuration
- Enabling of services
- Fetch secrets from Vault

Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

OpenTofu: Reusable Infrastructure Modules

Building blocks for HPC infrastructure as code

What is OpenTofu?

Open-source Terraform fork that enables infrastructure engineers to declaratively define reusable modules that encapsulate infrastructure components as version-controlled, parameterized building blocks.

Modular Design
Encapsulate infrastructure patterns into discrete, reusable declarative modules

Component Library
Build libraries of network, cluster, storage, and compute modules

Parameterization
Configure modules with variables for flexible, context-specific deployment

Version Control
Track and manage infrastructure code changes with standard VCS workflows

Infrastructure modules catalog

```
modules/
└─ cluster/            └─ network/           └─ server/
     ├─ main.tf            ├─ main.tf            ├─ main.tf
     ├─ variables.tf       ├─ variables.tf       ├─ variables.tf
     ├─ outputs.tf         ├─ outputs.tf         ├─ outputs.tf
     └─ terragrunt.hc      └─ terragrunt.hcl     └─
terragrunt.hcl
```

modules/network/main.tf

```
resource "openstack_networking_network_v2" "network" {
  name                  = var.network_name
  admin_state_up        = "true"
  port_security_enabled = "true"
  dns_domain            = var.dns_domain
}

resource "openstack_networking_subnet_v2" "subnet_v" {
  name            = var.subnet_name
  network_id      = openstack_networking_network_v2.network.id
  cidr            = var.nodes_net_cidr
  ip_version      = 4
  dns_nameservers = var.dns_servers
}

resource "openstack_networking_subnet_v2" "subnet_v6" {
  count             = var.enable_ipv6 ? 1 : 0
  name              = var.ipv6_subnet_name
  network_id        = openstack_networking_network_v2.network.id
  ip_version        = 6
  dns_nameservers   = var.dns_ipv6_servers
  ipv6_ra_mode      = var.ipv6_ra_mode
  ipv6_address_mode = var.ipv6_address_mode
  cidr              = var.ipv6_net_cidr
}

resource "openstack_networking_router_v2" "router" {
  count               = var.router_name != null ? 1 : 0
  name                = var.router_name
  admin_state_up      = true
  external_network_id =
data.openstack_networking_network_v2.public_net[0].id
}
```

Terragrunt: Composing OpenTofu Modules

DRY orchestration across environments

What is Terragrunt?

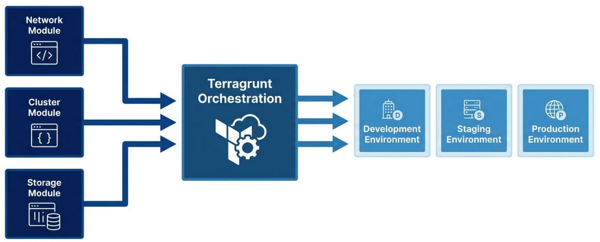
Terragrunt wraps OpenTofu to compose infrastructure modules into reusable, environment-agnostic configurations. It eliminates code duplication by enabling a single module definition to be deployed across multiple environments.

Module Composition
Orchestrate multiple OpenTofu modules as cohesive units

Configuration Management
Centralize shared configuration, minimize duplication

Multi-Environment
Deploy consistent infrastructure across dev, staging, and production

Version Control
Track and manage infrastructure code changes with standard VCS workflows



HPC terragrunt cluster repo

```
├─ root.hcl
└─ dev/                          └─ staging/                      └─ production/
   ├─ env.hcl                       ├─ env.hcl                       ├─ env.hcl
   └─ hpc_virtual/                  └─ hpc_virtual/                  └─ hpc_bm/
      └─ terragrunt.stack.hcl          └─ terragrunt.stack.hcl          └─ terragrunt.stack.hcl
                                    └─ hpc_bm/
                                       └─ terragrunt.stack.hcl
```

terragrunt.stack.hcl

```
env_vars =
read_terragrunt_config(find_in_parent_folders("env.hcl"))

stack "cluster" {
  source = "git::git@github.com:catalog.git//stacks/cluster"
  path = "cluster"

  values {
    network_name      = "hpc_network"
    router_name       = "hpc_router"
    subnet_name       = "hpc_subnet-ipv4"
    ipv6_subnet_name  = "hpc_-subnet-ipv6"
    enable_ipv6       =  true
    cluster_name      = "hpc-vm"
    cluster_image_id  = "62d9542d-85eb-4d85-b4c1-3ebd909d2935"
    node_flavor       =  local.env_vars.flavor
    node_count        =  local.env_vars.srv_count
    key_pair          =  "ssh_key"
    os_cloud          =  local.env_vars.os_cloud
  }
```

env.hcl

```
# dev:
locals {
  srv_count = 2
  flavor = "small"
  os_cloud = "dev"
}

# staging
locals {
  srv_count = 10
  flavor = "medium"
  os_cloud = "stg"
}

# prod
locals {
  srv_count = 100
  flavor = "large"
  os_cloud = "prod"
}
```

Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

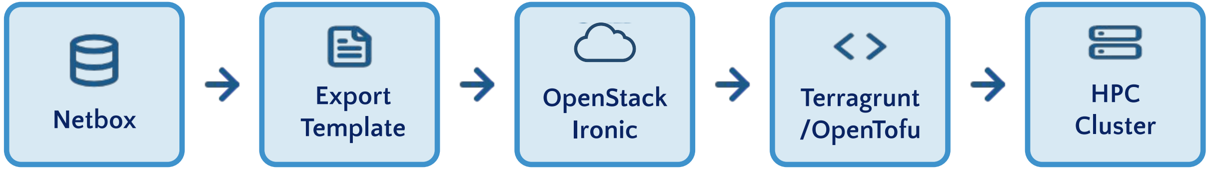
6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

Baremetal Provisioning: OpenStack Ironic

From Netbox via OpenStack Cloud to HPC cluster



Baremetal Provisioning: OpenStack Ironic

Onboarding hardware to the On-Prem OpenStack Cloud

Netbox custom export template

```
chassis:
  - description: c2-47_c2-50
    uuid: e4656520-4344-4e5a-9035-9819e5597cd8
    extra:
      nodes:
        - c2-47
        - c2-48
        - c2-49
        - c2-50
nodes:
  - name: c2-47
    chassis_uuid: e4656520-4344-4e5a-9035-9819e5597cd8
    driver: redfish
    driver_info:
      redfish_address: c2-47.bmc.clip.vbc.ac.at
      redfish_username: USERNAME
      redfish_password: XXXXXX
    boot_interface: redfish-virtual-media
    resource_class: baremetal-C2
    properties:
      capabilities: "boot_mode:uefi,boot_option:local"
      cpu_arch: "x86_64"
      vendor: ACME Computer Corp.
    ports:
      - address: 98:03:9B:64:5A:9E
        pxe_enabled: True
        local_link_connection:
            switch_id: 70:6D:15:41:73:30
            switch_info:
apic_dn:topology/pod-1/paths-114/pathep-[eth1/8],physical_network:physnet1
            port_id: eth1/8
```

Chassis information

Track blade systems and node to chassis assignments

Node information & BMC configuration

BMC address, credentials, type of driver and node configuration

Network information

Network configuration via Openstack-SDN integration

Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

Self-Sufficient Nodes: ansible-init Service

The Problem

How do nodes configure themselves on first boot without manual intervention?

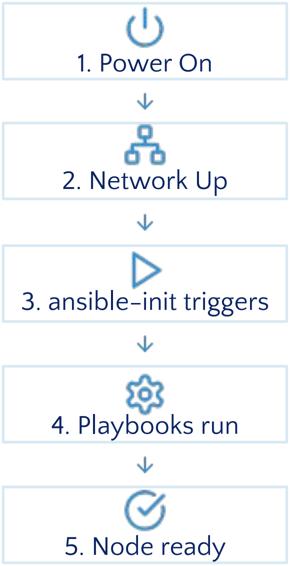
The Solution

- A systemd service runs ansible playbooks on first boot
- This service configures the node using ansible locally
- If the run is successful, we write a sentinel file that will prevent another run on next boot

Our Implementation

- This service specifically runs `configure` tasks
- Which playbooks should run is either controlled through
  - a passed ansible inventory
  - metadata (from curl or config drive)
- The configuration playbooks themselves are baked into the image

Props for this idea go to **StackHPC**



Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

Secret Management: Short-Lived Credentials

The Problem

- Long-lived credentials to vault applications pose security risks
- If leaked, attackers have persistent access
- Difficult to audit who accessed what

The Solution

- Credentials to vault generated on-demand with short TTL
- Each node gets unique credentials
- Automatic expiration minimizes blast radius

Our implementation:

- The vault that nodes access is generated by OpenTofu
- This vault itself fetches secrets from another vault

Terraform

1. Create nodes and related credentials to vault

↓

OpenStack

2. Node boots; secrets access token provided in metadata

↓

ansible-init

3. Secret access in ansible code is scoped to correct vault

↓

Node queries Vault

4. Node uses token to fetch secrets

Architecture Overview: The Zero-Touch Pipeline

From "just racked" to "running SLURM jobs" with zero manual intervention

1. Source of Truth
NetBox DCIM: IPAM, device inventory, rack locations

↓

2. Immutable Images
Packer + Ansible: Pre-built OS with HPC stack baked in

↓

3. Infrastructure Code
OpenTofu + Terragrunt: Declare bare metal nodes, network config

↓

4. Bare Metal Provisioning
OpenStack Ironic: Power on, inspect, deploy OS image

↓

5. Node Bootstrap
ansible-init service: Self-configuration on first boot

↓

6. Secrets & Config
Vault (short-lived creds): Dynamic secrets, no static passwords

↓

7. Cluster Ready
SLURM nodes join cluster: Nodes auto-register, ready for jobs

Acknowledgements

HPC Team

Erich Birngruber
Leon Schwarzäugl
Ümit Seren
Felix Schmitt
Alexander Bindeus

Questions?

We'd love to discuss your HPC automation challenges

Contact Us

Ümit Seren
uemit.seren@vbc.ac.at

Leon Schwarzäugl
leon.schwarzaeugl@imba.oeaw.ac.at

Resources

github.com/clip-hpc

clip.science

Thank you!