

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	The Pain Points
<ul style="list-style-type: none">✓ Automated (Payload):<ul style="list-style-type: none">• OpenStack for compute resources• Ansible for SLURM cluster config✗ Manual (Infrastructure):<ul style="list-style-type: none">• SDN Network configuration• BMC IP/DHCP management• Excel-based inventory• Storage configuration	<ul style="list-style-type: none">⚠ 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

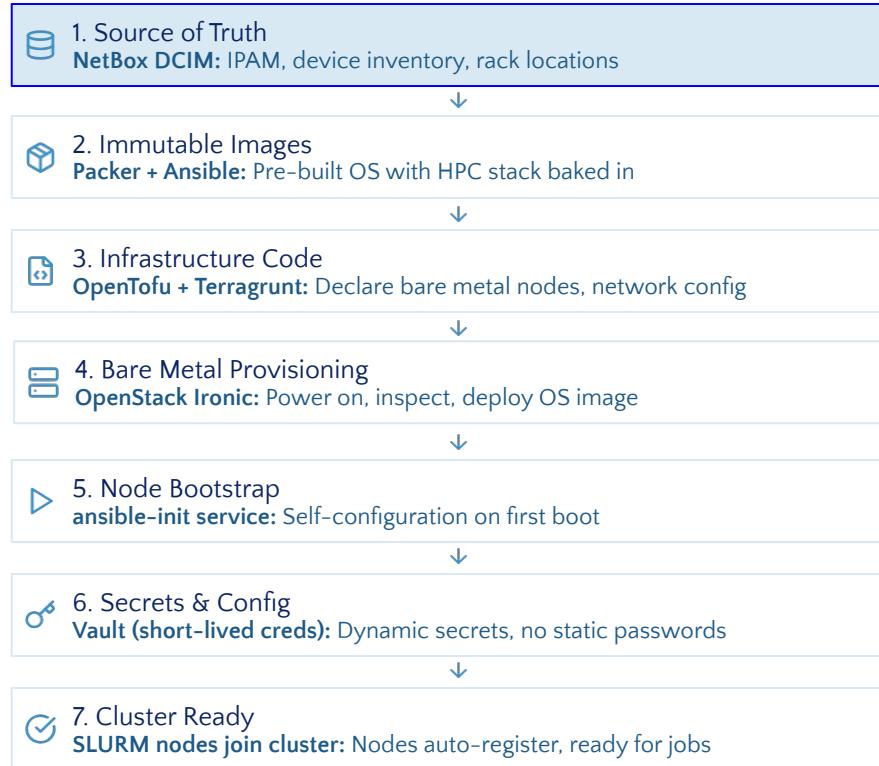
Architecture Overview: The Zero-Touch Pipeline

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



Architecture Overview: The Zero-Touch Pipeline

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



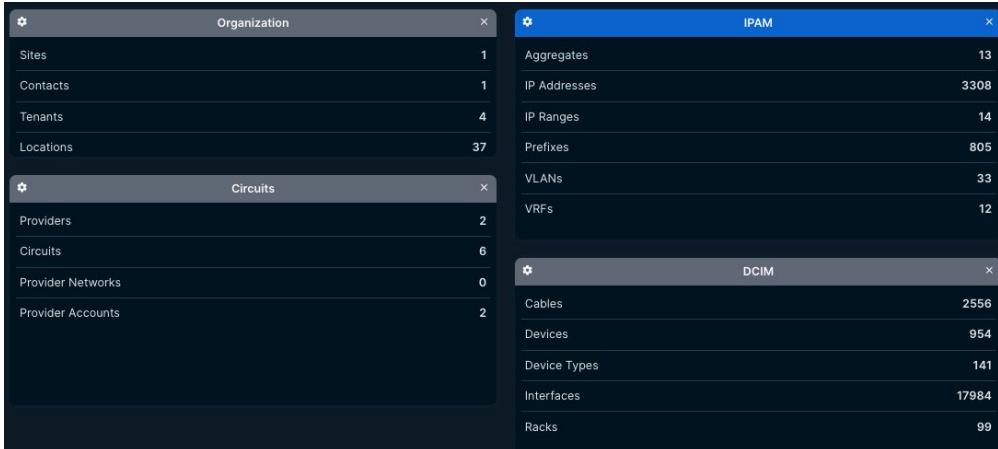
NetBox: DCIM and Source of Truth

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

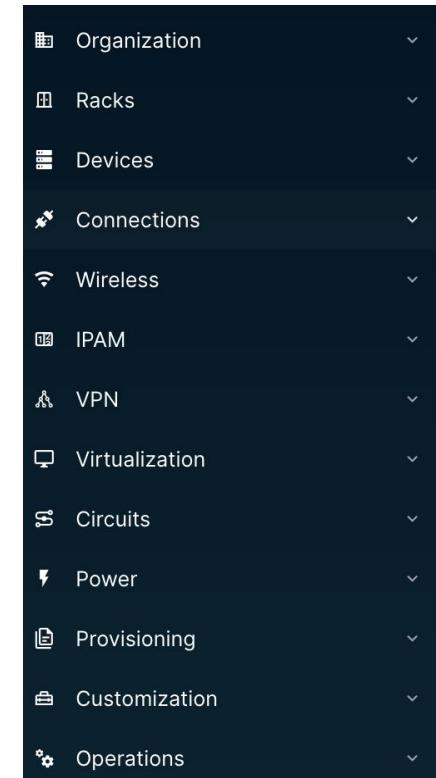
Scale: 100 racks, 150+ switches, 300+ nodes—all tracked in NetBox



The image shows a screenshot of the NetBox web interface. It features three main dashboard cards:

- Organization:** Shows 1 Site, 1 Contact, 4 Tenants, and 37 Locations.
- IPAM:** Shows 13 Aggregates, 3308 IP Addresses, 14 IP Ranges, 805 Prefixes, 33 VLANs, and 12 VRPs.
- DCIM:** Shows 2556 Cables, 954 Devices, 141 Device Types, 17984 Interfaces, and 99 Racks.

Netbox Feature Overview



The image shows a sidebar titled "Netbox Feature Overview" with a list of features, each with a collapse/expand icon:

- Organization
- Racks
- Devices
- Connections
- Wireless
- IPAM
- VPN
- Virtualization
- Circuits
- Power
- Provisioning
- Customization
- Operations

NetBox: DCIM and Source of Truth

From Excel to Netbox

Before

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	U42																		
2	U43																		
3	U40																		
4	U39																		
5	U38																		
6	U37																		
7	U36																		
8	U35																		
9	U34																		
10	U33																		
11	U32																		
12	U31																		
13	U30																		
14	U29																		
15	U28																		
16	U27																		
17	U26																		
18	U25																		
19	U24																		
20	U23																		
21	U22																		
22	U21																		
23	U20																		
24	U19																		
25	U18																		
26	U17																		
27	U16																		
28	U15																		
29	U14																		
30	U13																		
31	U12																		
32	U11																		
33	U10																		
34	U9																		
35	U8																		
36	U7																		
37	U6																		
38	U5																		
39	U4																		
40	U3																		
41	U2																		
42	U1																		
43	Front View																		
	Rack G9																		

After

	C-9	CLIP	C-10	CLIP	C-11	CLIP	G-9	CLIP	G-10	CLIP	G-11	CLIP
42	c11-2 prod		c11-2 staging		c11-2 dev		c11-1 prod		c11-1 staging		c11-1 dev	
43	c12-1 prod		c12-1 staging		c12-1 dev		c12-2 prod		c12-2 staging		c12-2 dev	
44	c13-1 prod		c13-1 staging		c13-1 dev		c13-2 prod		c13-2 staging		c13-2 dev	
45	c14-1 prod		c14-1 staging		c14-1 dev		c14-2 prod		c14-2 staging		c14-2 dev	
46	c15-1 prod		c15-1 staging		c15-1 dev		c15-2 prod		c15-2 staging		c15-2 dev	
47	c16-1 prod		c16-1 staging		c16-1 dev		c16-2 prod		c16-2 staging		c16-2 dev	
48	c17-1 prod		c17-1 staging		c17-1 dev		c17-2 prod		c17-2 staging		c17-2 dev	
49	c18-1 prod		c18-1 staging		c18-1 dev		c18-2 prod		c18-2 staging		c18-2 dev	
50	c19-1 prod		c19-1 staging		c19-1 dev		c19-2 prod		c19-2 staging		c19-2 dev	
51	c20-1 prod		c20-1 staging		c20-1 dev		c20-2 prod		c20-2 staging		c20-2 dev	
52	c21-1 prod		c21-1 staging		c21-1 dev		c21-2 prod		c21-2 staging		c21-2 dev	
53	c22-1 prod		c22-1 staging		c22-1 dev		c22-2 prod		c22-2 staging		c22-2 dev	
54	c23-1 prod		c23-1 staging		c23-1 dev		c23-2 prod		c23-2 staging		c23-2 dev	
55	c24-1 prod		c24-1 staging		c24-1 dev		c24-2 prod		c24-2 staging		c24-2 dev	
56	c25-1 prod		c25-1 staging		c25-1 dev		c25-2 prod		c25-2 staging		c25-2 dev	
57	c26-1 prod		c26-1 staging		c26-1 dev		c26-2 prod		c26-2 staging		c26-2 dev	
58	c27-1 prod		c27-1 staging		c27-1 dev		c27-2 prod		c27-2 staging		c27-2 dev	
59	c28-1 prod		c28-1 staging		c28-1 dev		c28-2 prod		c28-2 staging		c28-2 dev	
60	c29-1 prod		c29-1 staging		c29-1 dev		c29-2 prod		c29-2 staging		c29-2 dev	
61	c30-1 prod		c30-1 staging		c30-1 dev		c30-2 prod		c30-2 staging		c30-2 dev	
62	c31-1 prod		c31-1 staging		c31-1 dev		c31-2 prod		c31-2 staging		c31-2 dev	
63	c32-1 prod		c32-1 staging		c32-1 dev		c32-2 prod		c32-2 staging		c32-2 dev	
64	c33-1 prod		c33-1 staging		c33-1 dev		c33-2 prod		c33-2 staging		c33-2 dev	
65	c34-1 prod		c34-1 staging		c34-1 dev		c34-2 prod		c34-2 staging		c34-2 dev	
66	c35-1 prod		c35-1 staging		c35-1 dev		c35-2 prod		c35-2 staging		c35-2 dev	
67	c36-1 prod		c36-1 staging		c36-1 dev		c36-2 prod		c36-2 staging		c36-2 dev	
68	c37-1 prod		c37-1 staging		c37-1 dev		c37-2 prod		c37-2 staging		c37-2 dev	
69	c38-1 prod		c38-1 staging		c38-1 dev		c38-2 prod		c38-2 staging		c38-2 dev	
70	c39-1 prod		c39-1 staging		c39-1 dev		c39-2 prod		c39-2 staging		c39-2 dev	
71	c40-1 prod		c40-1 staging		c40-1 dev		c40-2 prod		c40-2 staging		c40-2 dev	
72	c41-1 prod		c41-1 staging		c41-1 dev		c41-2 prod		c41-2 staging		c41-2 dev	
73	c42-1 prod		c42-1 staging		c42-1 dev		c42-2 prod		c42-2 staging		c42-2 dev	
74	c43-1 prod		c43-1 staging		c43-1 dev		c43-2 prod		c43-2 staging		c43-2 dev	
75	c44-1 prod		c44-1 staging		c44-1 dev		c44-2 prod		c44-2 staging		c44-2 dev	
76	c45-1 prod		c45-1 staging		c45-1 dev		c45-2 prod		c45-2 staging		c45-2 dev	
77	c46-1 prod		c46-1 staging		c46-1 dev		c46-2 prod		c46-2 staging		c46-2 dev	
78	c47-1 prod		c47-1 staging		c47-1 dev		c47-2 prod		c47-2 staging		c47-2 dev	
79	c48-1 prod		c48-1 staging		c48-1 dev		c48-2 prod		c48-2 staging		c48-2 dev	
80	c49-1 prod		c49-1 staging		c49-1 dev		c49-2 prod		c49-2 staging		c49-2 dev	
81	c50-1 prod		c50-1 staging		c50-1 dev		c50-2 prod		c50-2 staging		c50-2 dev	
82	c51-1 prod		c51-1 staging		c51-1 dev		c51-2 prod		c51-2 staging		c51-2 dev	
83	c52-1 prod		c52-1 staging		c52-1 dev		c52-2 prod		c52-2 staging		c52-2 dev	
84	c53-1 prod		c53-1 staging		c53-1 dev		c53-2 prod		c53-2 staging		c53-2 dev	
85	c54-1 prod		c54-1 staging		c54-1 dev		c54-2 prod		c54-2 staging		c54-2 dev	
86	c55-1 prod		c55-1 staging		c55-1 dev		c55-2 prod		c55-2 staging		c55-2 dev	
87	c56-1 prod		c56-1 staging		c56-1 dev		c56-2 prod		c56-2 staging		c56-2 dev	
88	c57-1 prod		c57-1 staging		c57-1 dev		c57-2 prod		c57-2 staging		c57-2 dev	
89	c58-1 prod		c58-1 staging		c58-1 dev		c58-2 prod		c58-2 staging		c58-2 dev	
90	c59-1 prod		c59-1 staging		c59-1 dev		c59-2 prod		c59-2 staging		c59-2 dev	
91	c60-1 prod		c60-1 staging		c60-1 dev		c60-2 prod		c60-2 staging		c60-2 dev	
92	c61-1 prod		c61-1 staging		c61-1 dev		c61-2 prod		c61-2 staging		c61-2 dev	
93	c62-1 prod		c62-1 staging		c62-1 dev		c62-2 prod		c62-2 staging		c62-2 dev	
94	c63-1 prod		c63-1 staging		c63-1 dev		c63-2 prod		c63-2 staging		c63-2 dev	
95	c64-1 prod		c64-1 staging		c64-1 dev		c64-2 prod		c64-2 staging		c64-2 dev	
96	c65-1 prod		c65-1 staging		c65-1 dev		c65-2 prod		c65-2 staging		c65-2 dev	
97	c66-1 prod		c66-1 staging		c66-1 dev		c66-2 prod		c66-2 staging		c66-2 dev	
98	c67-1 prod		c67-1 staging		c67-1 dev		c67-2 prod		c67-2 staging		c67-2 dev	
99	c68-1 prod		c68-1 staging		c68-1 dev		c68-2 prod		c68-2 staging		c68-2 dev	
100	c69-1 prod		c69-1 staging		c69-1 dev		c69-2 prod		c69-2 staging		c69-2 dev	
101	c70-1 prod		c70-1 staging		c70-1 dev		c70-2 prod		c70-2 staging		c70-2 dev	
102	c71-1 prod		c71-1 staging		c71-1 dev		c71-2 prod		c71-2 staging		c71-2 dev	
103	c72-1 prod		c72-1 staging		c72-1 dev		c72-2 prod		c72-2 staging		c72-2 dev	
104	c73-1 prod		c73-1 staging		c73-1 dev		c73-2 prod		c73-2 staging		c73-2 dev	
105	c74-1 prod		c74-1 staging		c74-1 dev		c74-2 prod		c74-2 staging		c74-2 dev	
106	c75-1 prod		c75-1 staging		c75-1 dev		c75-2 prod		c75-2 staging		c75-2 dev	
107	c76-1 prod		c76-1 staging		c76-1 dev		c76-2 prod		c76-2 staging		c76-2 dev	
108	c77-1 prod		c77-1 staging		c77-1 dev		c77-2 prod		c77-2 staging		c77-2 dev	
109	c78-1 prod		c78-1 staging		c78-1 dev		c78-2 prod		c78-2 staging		c78-2 dev	
110	c79-1 prod		c79-1 staging		c79-1 dev		c79-2 prod		c79-2 staging		c79-2 dev	
111	c80-1 prod		c80-1 staging		c80-1 dev		c80-2 prod		c80-2 staging		c80-2 dev	
112	c81-1 prod		c81-1 staging		c81-1 dev		c81-2 prod		c81-2 staging		c81-2 dev	
113	c82-1 prod		c82-1 staging		c82-1 dev		c82-2 prod		c82-2 staging		c82-2 dev	
114	c83-1 prod		c83-1 staging		c83-1 dev		c83-2 prod		c83-2 staging		c83-2 dev	
115	c84-1 prod		c84-1 staging		c84-1 dev		c84-2 prod		c84-2 staging		c84-2 dev	
116	c85-1 prod		c85-1 staging		c85-1 dev		c85-2 prod		c85-2 staging		c85-2 dev	
117	c86-1 prod		c86-1 staging		c86-1 dev		c86-2 prod		c86-2 staging		c86-2 dev	
118	c87-1 prod		c87-1 staging		c87-1 dev		c87-2 prod		c87-2 staging		c87-2 dev	
119	c88-1 prod		c88-1 staging		c88-1 dev		c88-2 prod		c88-2 staging		c88-2 dev	

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

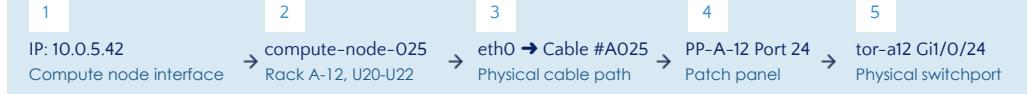
Storage system (OpenTofu):

- 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



NetBox Interface

172.16.43.0/22
Created 2024-07-11 13:16 · Updated 2024-10-03 08:29

IP Address 341 Related IPs 341 Contacts Journal Changelog

IP Address

Family	IPV4
VRF	Shared Protected Services (110)
Tenant	—
Status	DHCP
Role	—
DNS Name	biooptics-gpu-1.bmc.vbc.ac.at
Description	—
Assignment	biooptics-gpu-1/XCC
NAT (inside)	—
NAT (outside)	—
Primary IP	✗
OOB IP	✓
Tags	BioOptics IaC Imported

Parent Prefixes

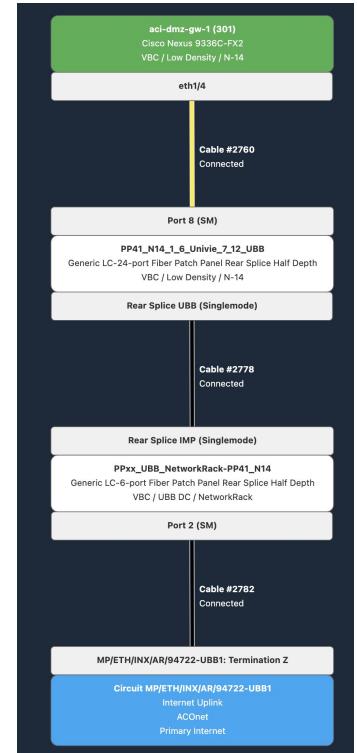
PREFIX	STATUS	CHILDREN	TENANT	SITE	VLAN	ROLE	DESCRIPTION
172.16.0.0/12	Container	92	—	—	—	—	Data Center
+ 172.16.0.0/18	Container	59	—	—	—	—	ACI
+ + 172.16.40.0/22	Active	0	—	VBC	—	OOB	Subnet for OOB interfaces (CLIP, etc)

Services

NAME	PARENT	PROTOCOL	PORTS	DESCRIPTION
— No services found —				

Bookmarks Clone Edit Delete

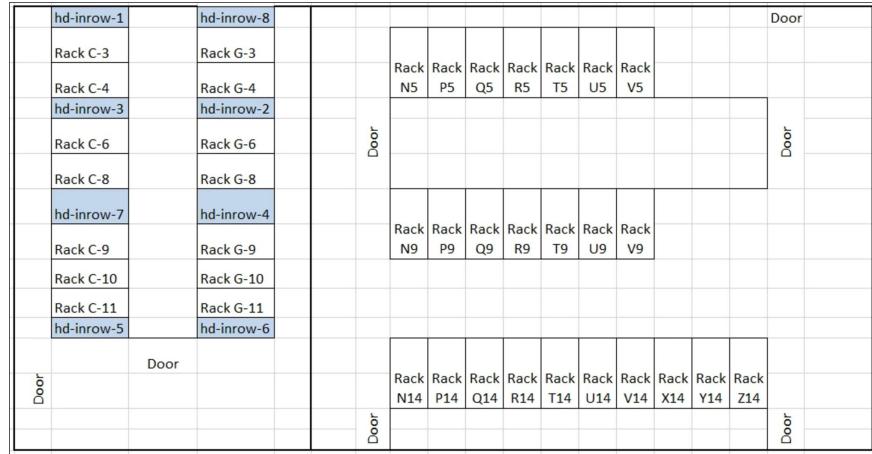
Cabling tracing



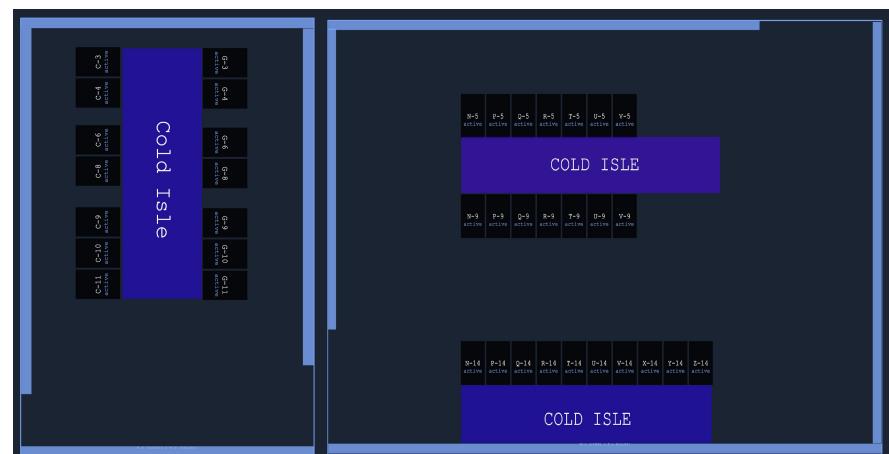
NetBox: Floorplan plugin

Netbox as a comprehensive documentation tool of physical space

Before



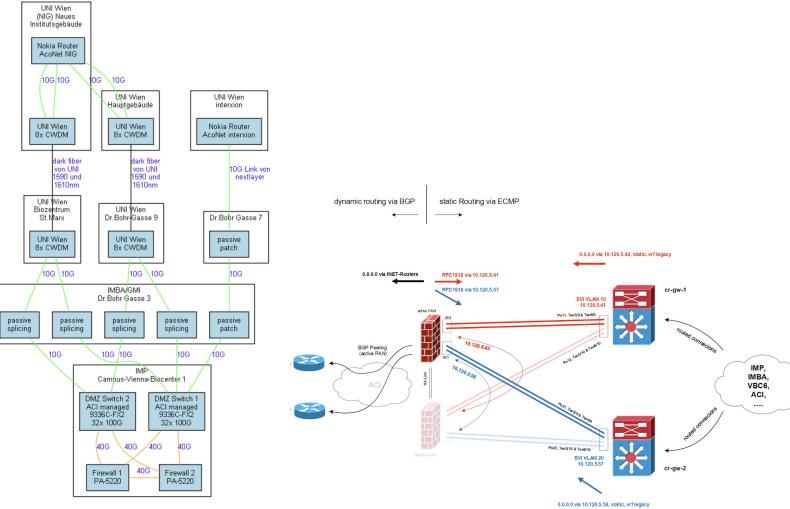
After



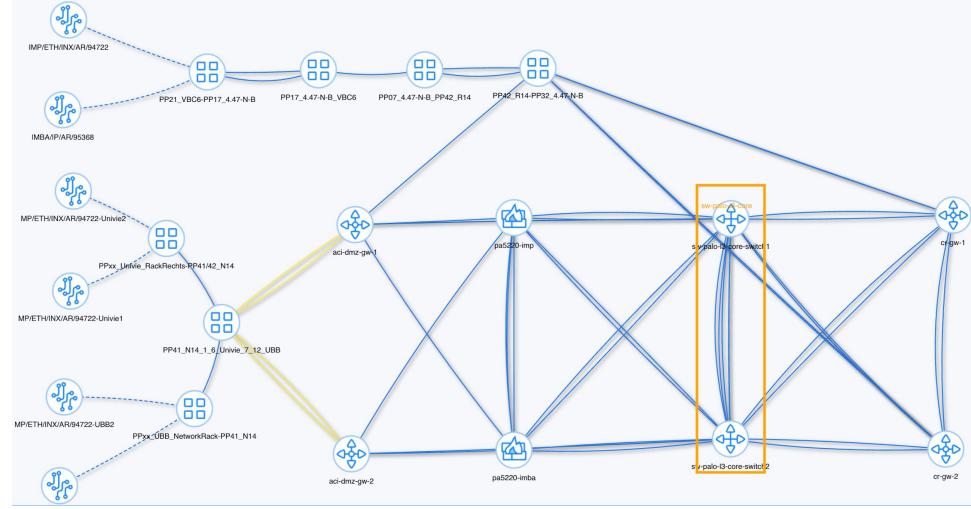
NetBox: Topology plugin

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

Before

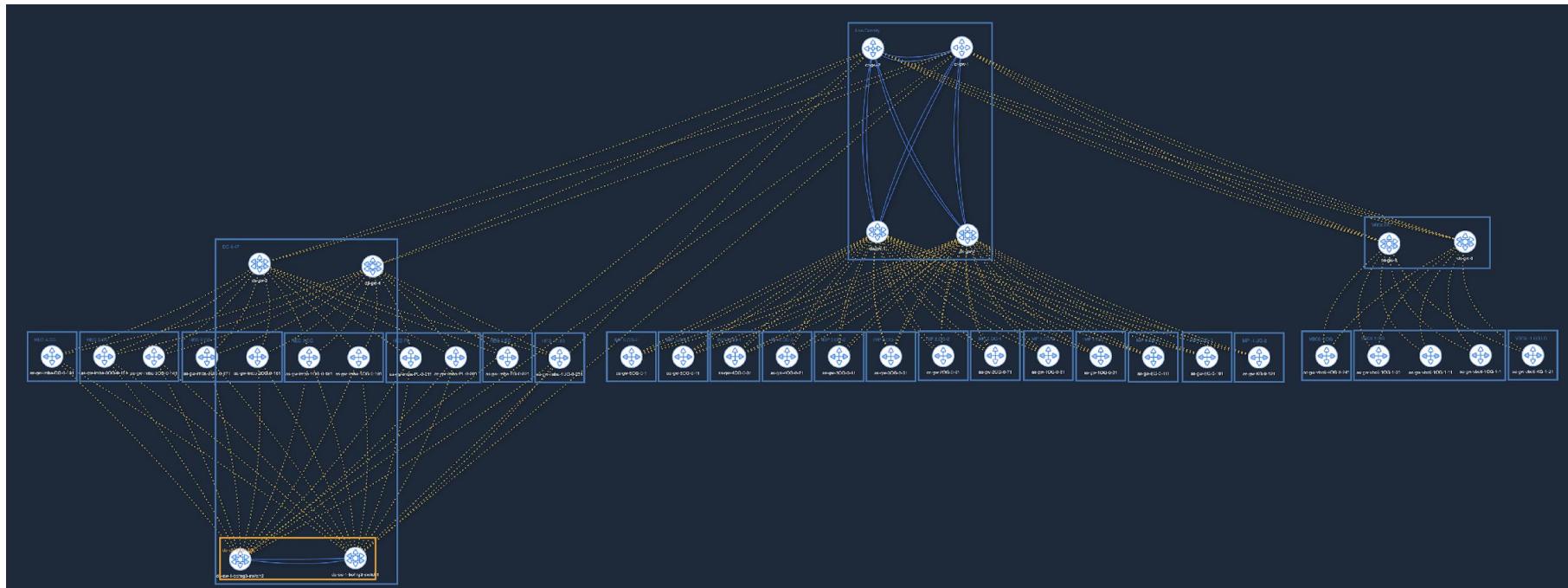


Afte



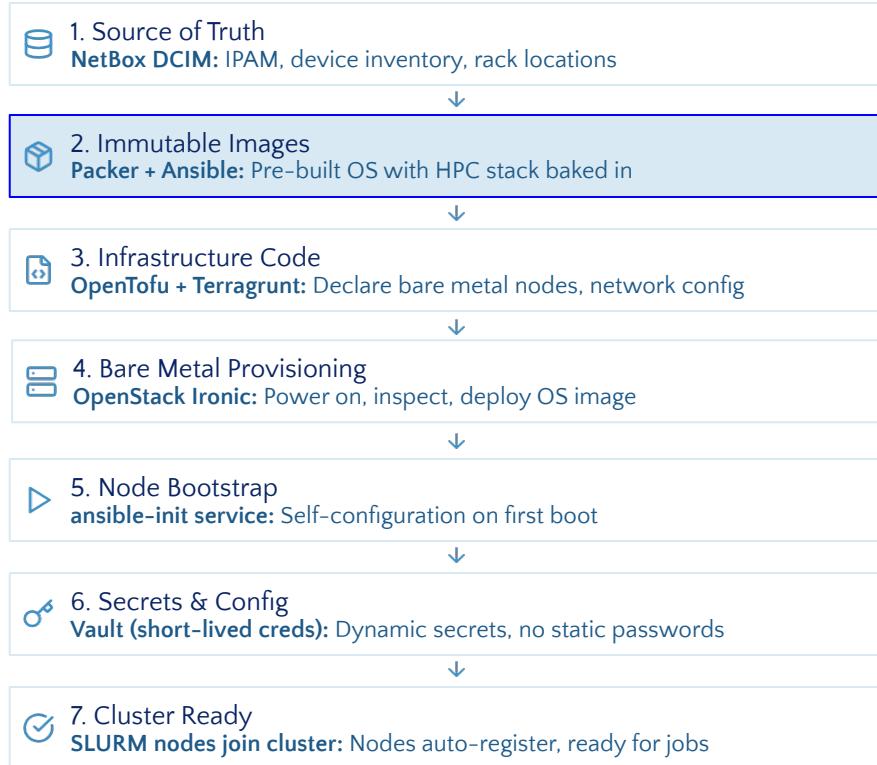
NetBox: Topology plugin

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



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

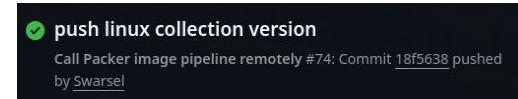
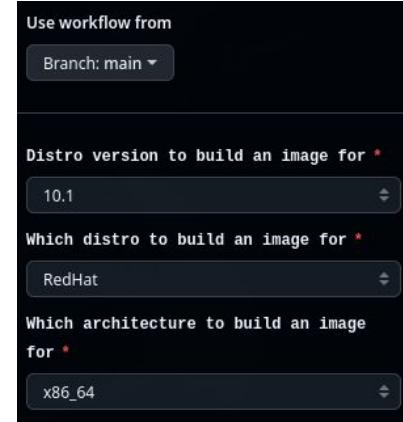
Packer

- ✓ 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

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.

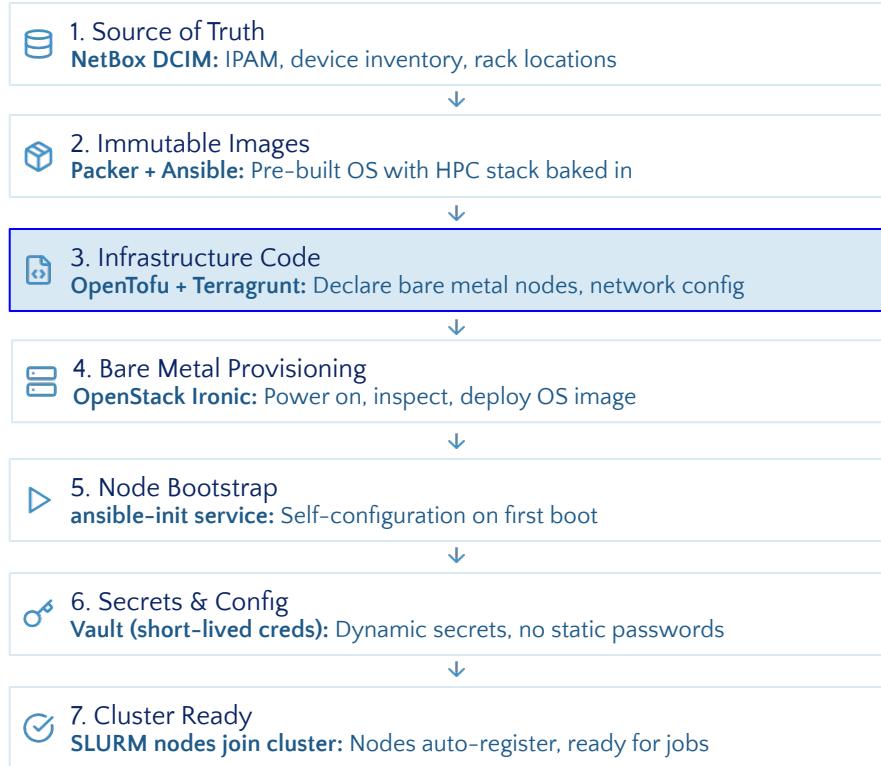
Purpose: Perform tasks that are not common to all hosts.

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



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/
    ├── main.tf
    ├── variables.tf
    ├── outputs.tf
    └── terragrunt.hcl
  └── network/
    ├── main.tf
    ├── variables.tf
    ├── outputs.tf
    └── terragrunt.hcl
  └── server/
    ├── main.tf
    ├── variables.tf
    └── outputs.tf
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/
    │   └── env.hcl
    │       └── hpc_virtual/
    │           └── terragrunt.stack.hcl
    └── staging/
        └── env.hcl
            └── hpc_virtual/
                └── terragrunt.stack.hcl
                    └── hpc_bm/
                        └── terragrunt.stack.hcl
    └── production/
        └── env.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

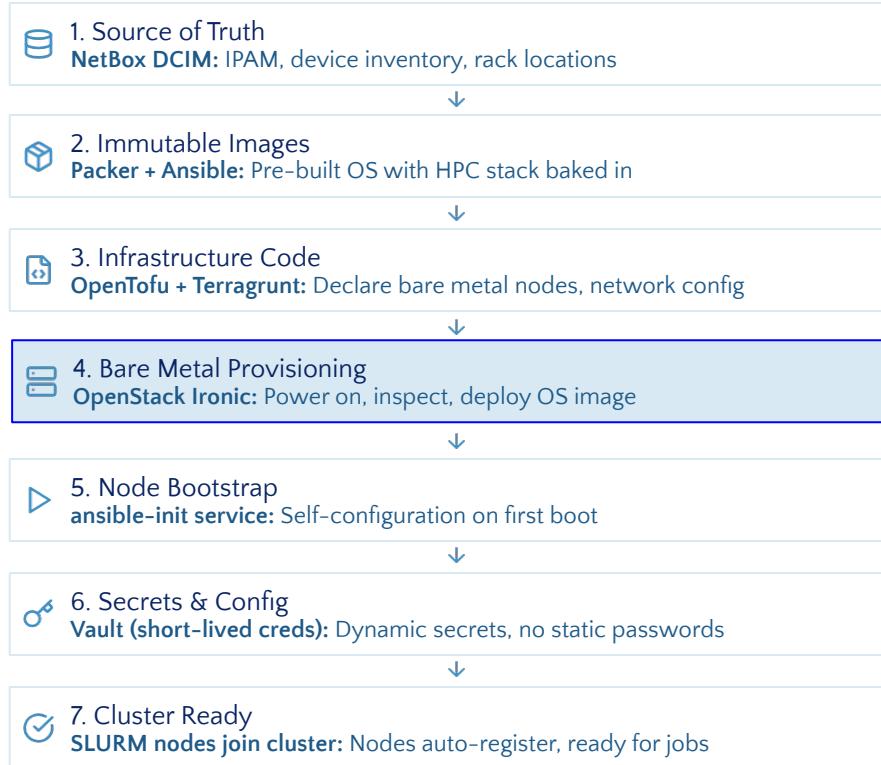
```
# dev:
locals {
  srv_count = 2
  flavor = "small"
  os_cloud = "dev"
}

# 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



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/pathp-[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

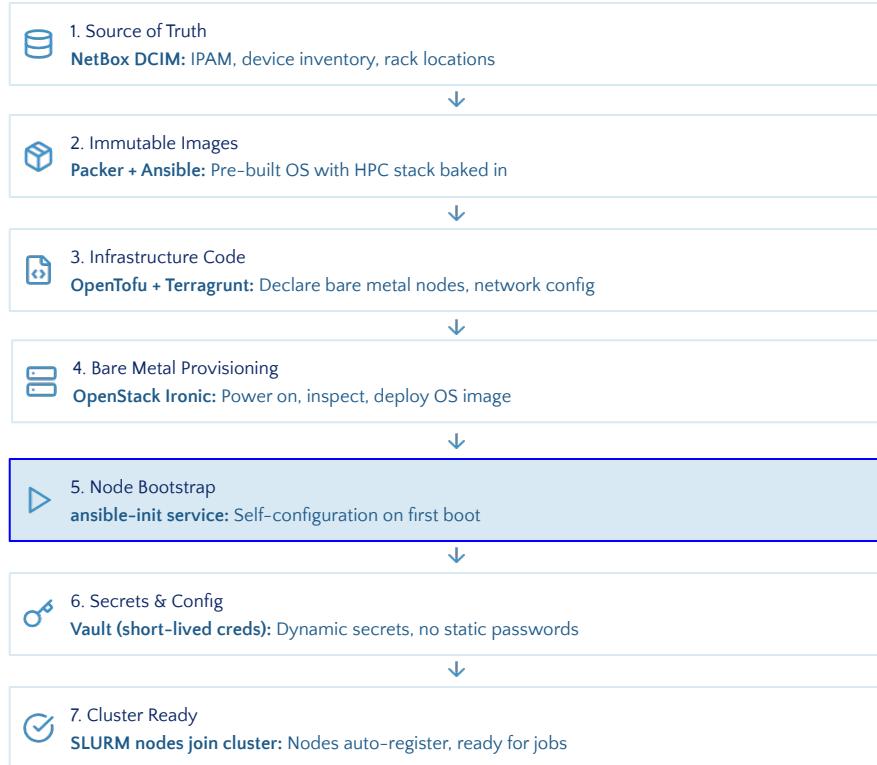


Network information

Network configuration via Openstack-SDN integration

Architecture Overview: The Zero-Touch Pipeline

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



Self-Sufficient Nodes: ansible-init Service

The Problem

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

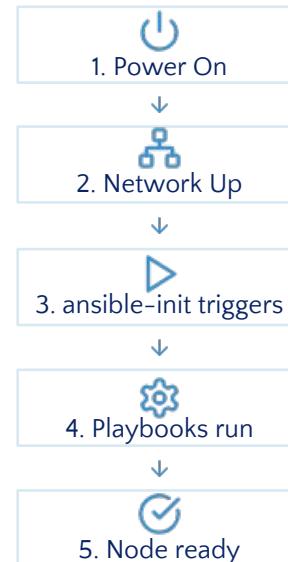
The Solution

- 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

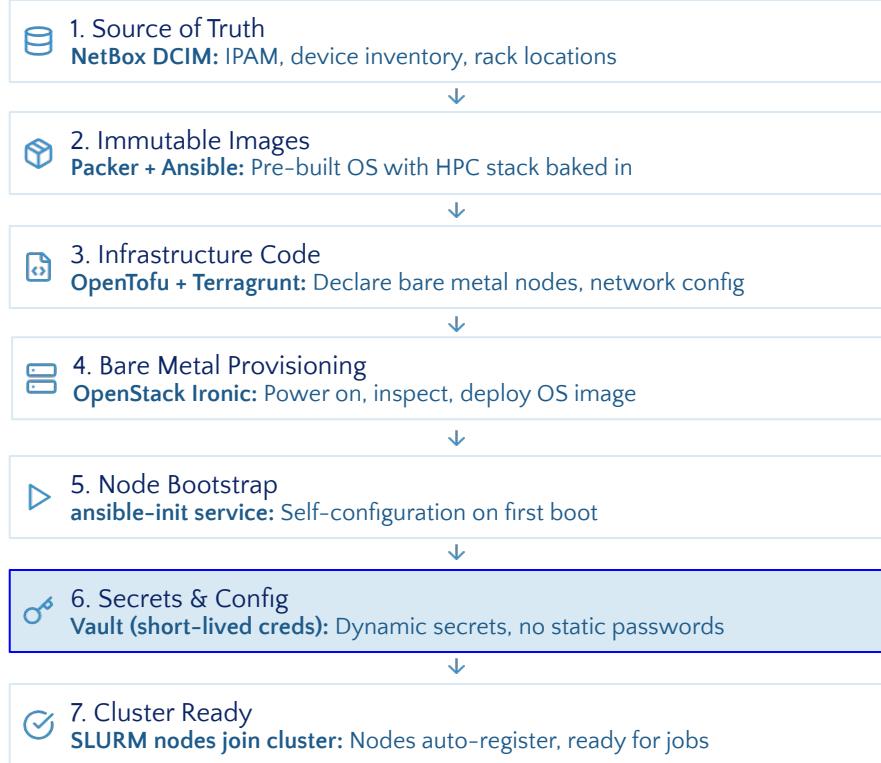
- This service specifically runs `configure` tasks
- Which playbooks should run is either controlled through
 - a passed ansible inventory
 - metadata (from curl or config drive)
- The configuration playbooks themselves are baked into the image

Props for this idea go to **StackHPC**



Architecture Overview: The Zero-Touch Pipeline

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



Secret Management: Short-Lived Credentials

The Problem

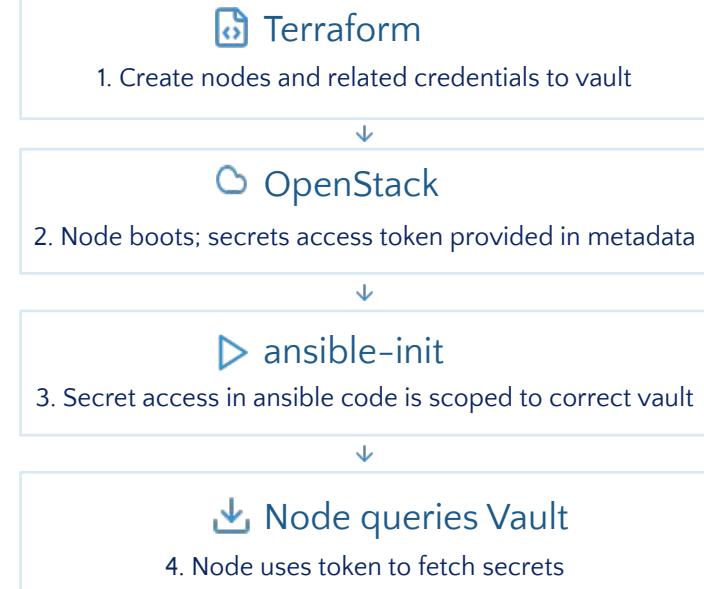
- Long-lived credentials to vault applications pose security risks
- If leaked, attackers have persistent access
- Difficult to audit who accessed what

The Solution

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

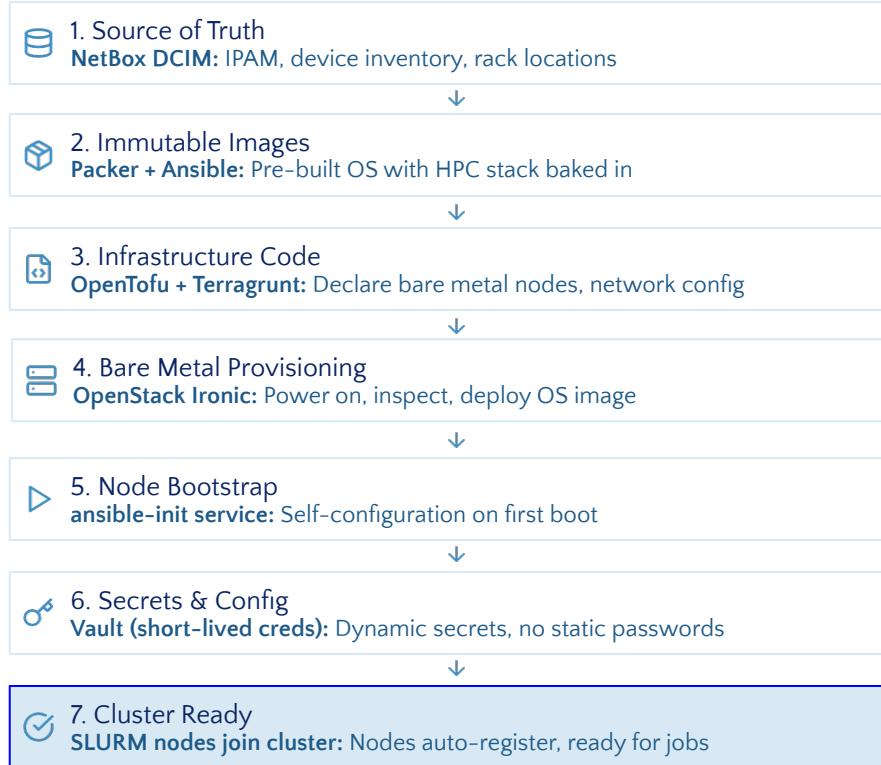
Our implementation:

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



Architecture Overview: The Zero-Touch Pipeline

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



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.oewa.ac.at

Resources

 github.com/clip-hpc
 clip.science

Thank you!