

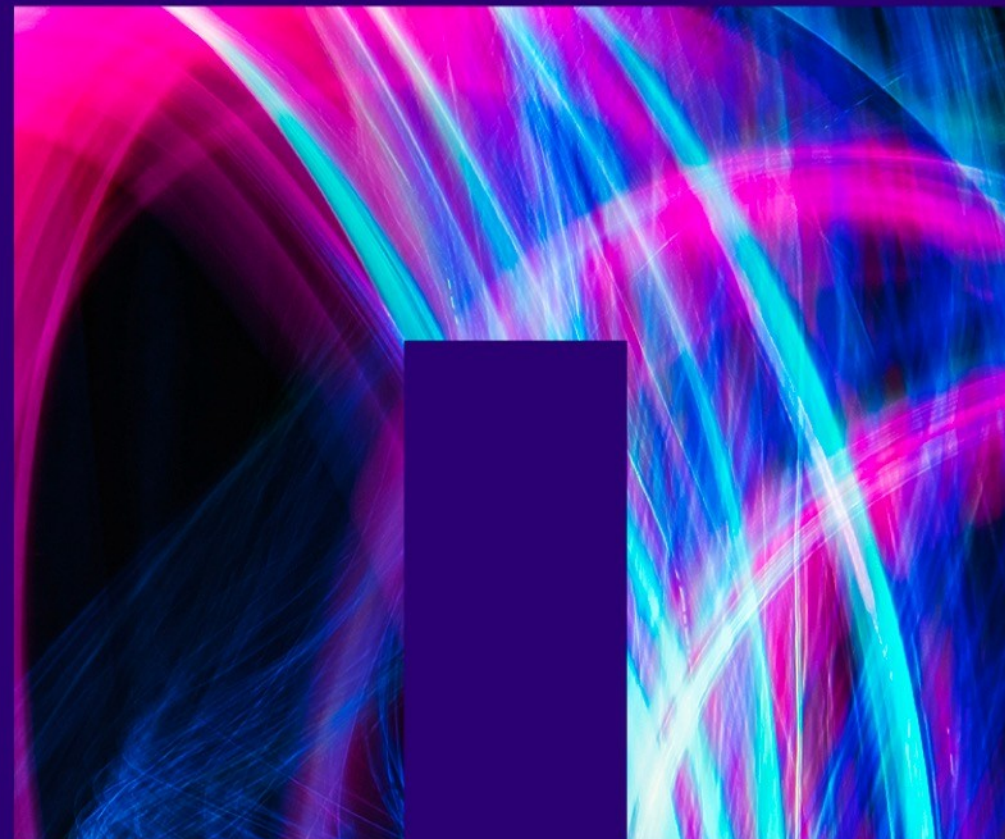
MAKING LUSTRE BORING

**An Open-Source Ansible Collection for
Deploying and Operating Lustre**

Gerhard Wolf

Consulting Solutions Architect

Thursday, October 1, 2026



The challenge: Lustre is powerful — but deployment is complex

The difficulty is not any single component. It is coordinating the full stack consistently.

BLOCK STORAGE

[E-Series] pools • volumes • host mappings

HIGH AVAILABILITY

Pacemaker • Corosync • cluster resources

LUSTRE SERVICES

MGS / MDS / OSS • targets • mounts

CLIENTS

Packages • network • mount configuration

MULTI-TENANCY

Nodemaps • filesets • quotas • isolation • QoS

OBSERVABILITY

Metrics • dashboards • performance

Manual coordination creates friction

- Many tools, configuration surfaces, and tuning knobs
- Ordering and dependencies matter
- Small inconsistencies become hard-to-debug failures
- Repeating the same deployment takes time

The automation opportunity — describe the desired state once and reproduce it consistently across every building block.

What Is Ansible?

A brief primer before we look at how automation deploys this solution

- An open-source IT automation engine used to configure systems, deploy software, and orchestrate infrastructure
- Agentless – connects over standard SSH; no software needs to be pre-installed on target hosts
- Declarative – you describe the desired end state, and Ansible determines the steps to get there
- Idempotent – running the same automation multiple times produces the same result, with no unwanted side effects
- Uses simple, human-readable YAML to define automation logic
- Widely used across the industry for repeatable, consistent infrastructure deployment

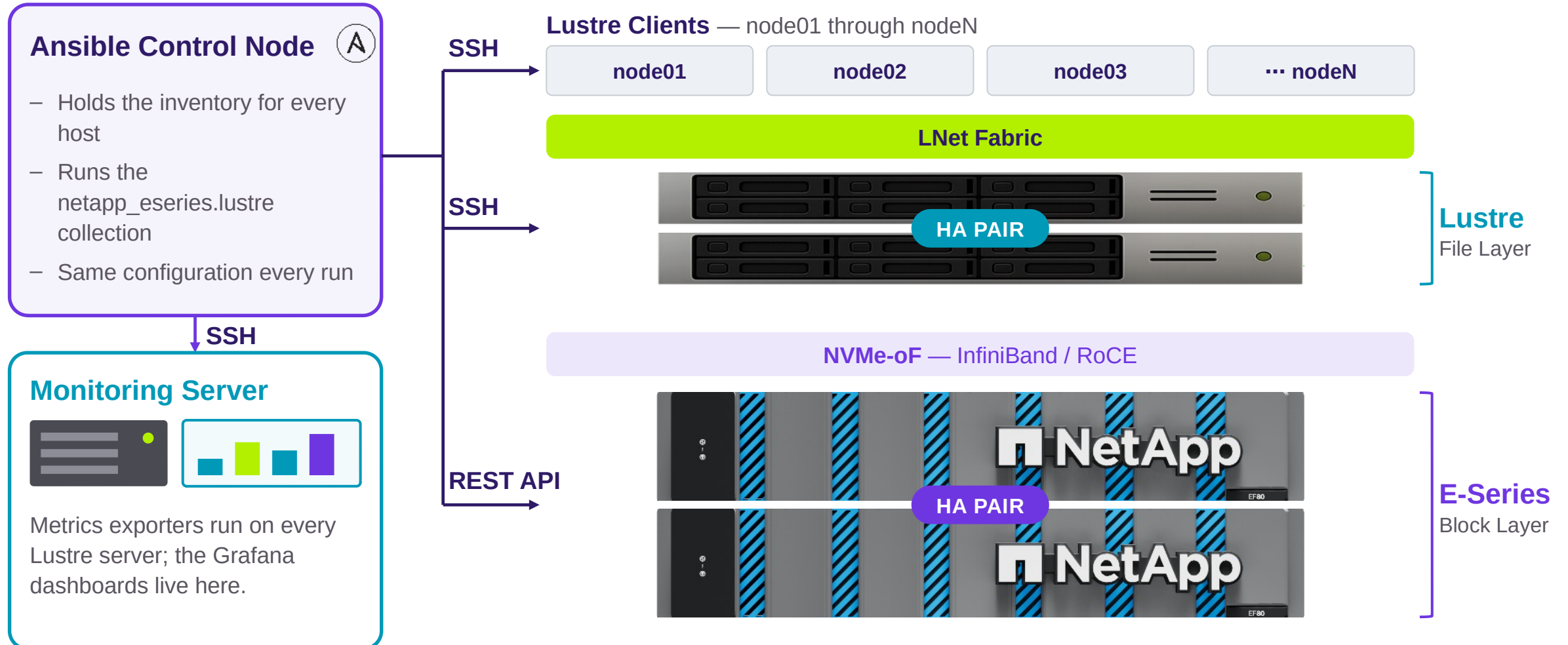
Ansible core concepts

The building blocks that make automation repeatable

Concept	Description
Control Node	The machine you run Ansible from; it connects out over SSH, so nothing must be installed on the managed systems
Inventory	The list of hosts and groups Ansible manages (e.g., metadata servers, object storage servers, clients)
Playbook	A YAML file mapping hosts from the inventory to the work applied to them
Task	One step — a single action on a host (e.g., install a package, mount a filesystem)
Role	A reusable bundle of tasks, variables, and templates for a specific function (e.g., configure a metadata server)
Collection	A distributable package of roles, modules, and plugins (e.g., netapp_eseries.lustre)
Variables	Site-specific values (e.g., names, IPs) that tailor playbooks and roles for an environment

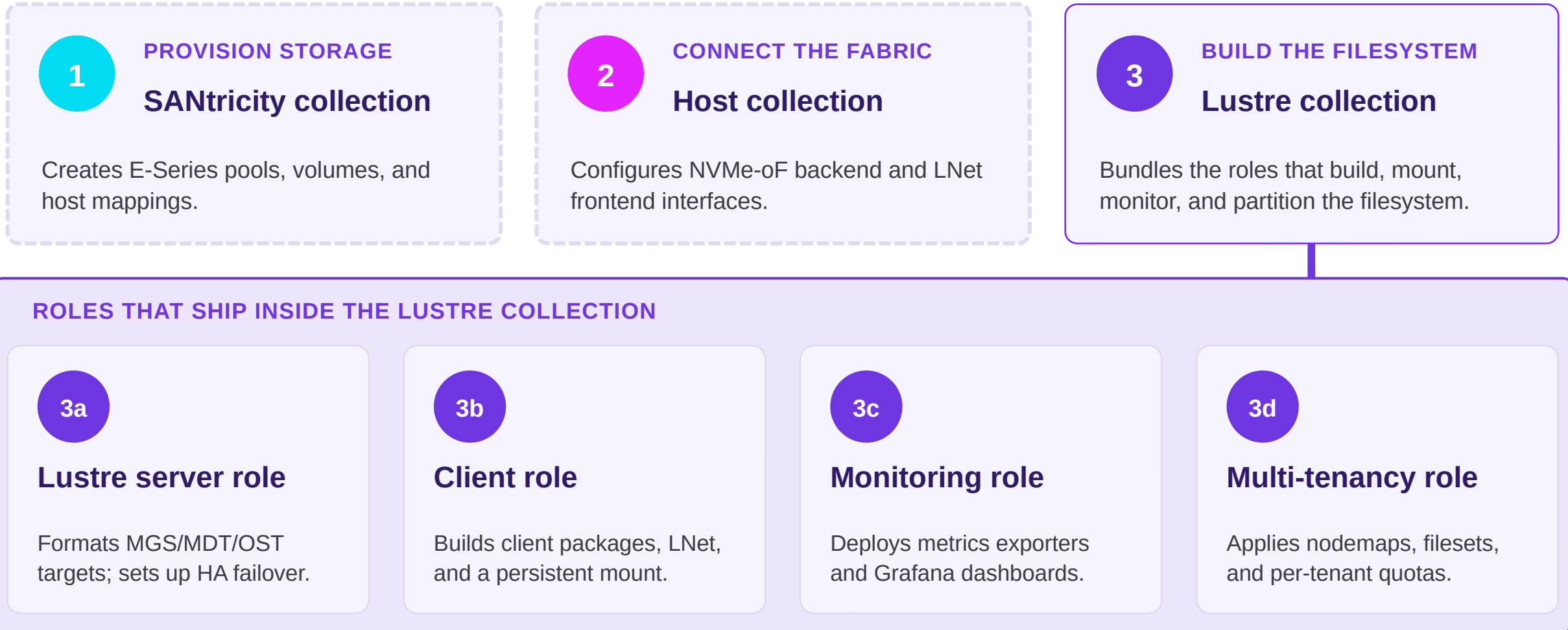
The environment — where the playbooks run

One control node reaches every client, server, and array — and stands up monitoring the same way



What Ansible deploys

Every building block, client, tenant, and dashboard — deployed end to end.



Built on what we already had

The Lustre collection grew out of our existing E-Series Ansible — so everything storage-specific comes in through a single call.

IT GREW ORGANICALLY

- Existing E-Series Ansible already provisioned pools, volumes, and host mappings
- The Lustre roles were built on top of what already worked

IT GOES THROUGH ONE CALL

- Every storage-specific fact enters through one call:
na_santricity_facts
- It populates one data structure — device name and path

THE REST IS JUST LUSTRE

- Pacemaker, fencing, and LNet never call a storage API
- Client mounts, tenancy, and monitoring are pure Lustre

One call to replace — that is what makes the E-Series anchoring a clean layer rather than a limitation.

Example of the folders

Four folders, one deployment tree

```
lustre_deployment_template/  
  building_blocks/    # Lustre HA server  
                      clusters  
    IB_H2_IB_DAC_A2_EF80/  
    ROCE_H2_ROCE_DAC_A2_EF80/  
  playbooks/  
  clients/    # IB and RoCE client  
              inventories  
    IB_lustre_cluster_clients/  
    ROCE_lustre_cluster_clients/  
  lustre_monitoring/    # Prometheus and  
                        Grafana stack  
  lustre_tenant/    # tenant nodemaps,  
                    filesets, quotas
```

Generated / hand-authored cluster

```
lustre_building_block_cluster_1/  
  lustre_inventory.yml  
  group_vars/  
    all.yml  
    ha_cluster.yml  
    eseries_storage_systems.yml  
    mgs.yml  
    mds_01.yml  
    mds_02.yml  
    oss_01.yml ... oss_04.yml  
  host_vars/  
    lustre_01.yml ... lustre_04.yml  
    netapp_01.yml ... netapp_04.yml
```

Deploying the file system, clients, and monitoring

Three playbook runs from the control node — multi-tenancy comes later, on its own.



-i + inventory — which hosts



Playbook — what to run



--tags — a subset of roles

CONTROL NODE — DEPLOY THE FILE SYSTEM

```
$ ansible-playbook -i lustre_building_block_cluster_2/lustre_inventory.yml playbooks/lustre_playbook.yml
```

CONTROL NODE — DEPLOY THE CLIENT

```
$ ansible-playbook -i lustre_client_inventory.yml lustre_client_playbook.yml
```

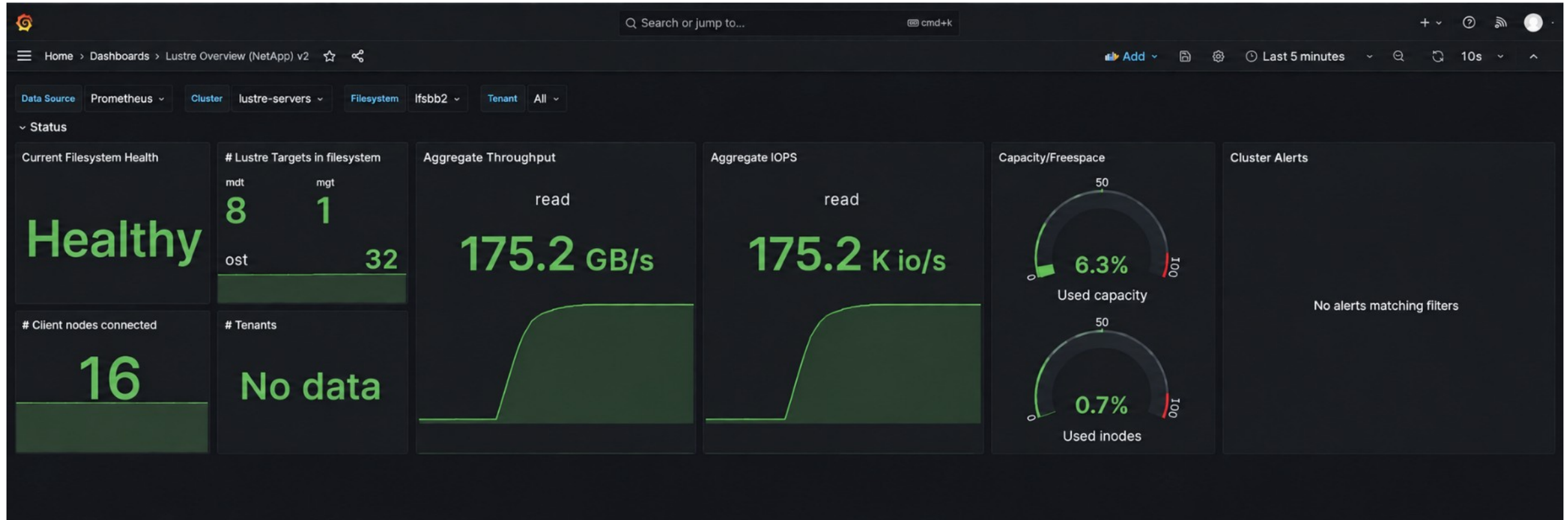
CONTROL NODE — DEPLOY MONITORING / GRAFANA

```
$ ansible-playbook -i hosts.yml lustre_monitoring_playbook.yml --tags exporters
```

One command, one desired state — every server, pool, and Lustre service configured the same way each run.

Live visibility with Grafana

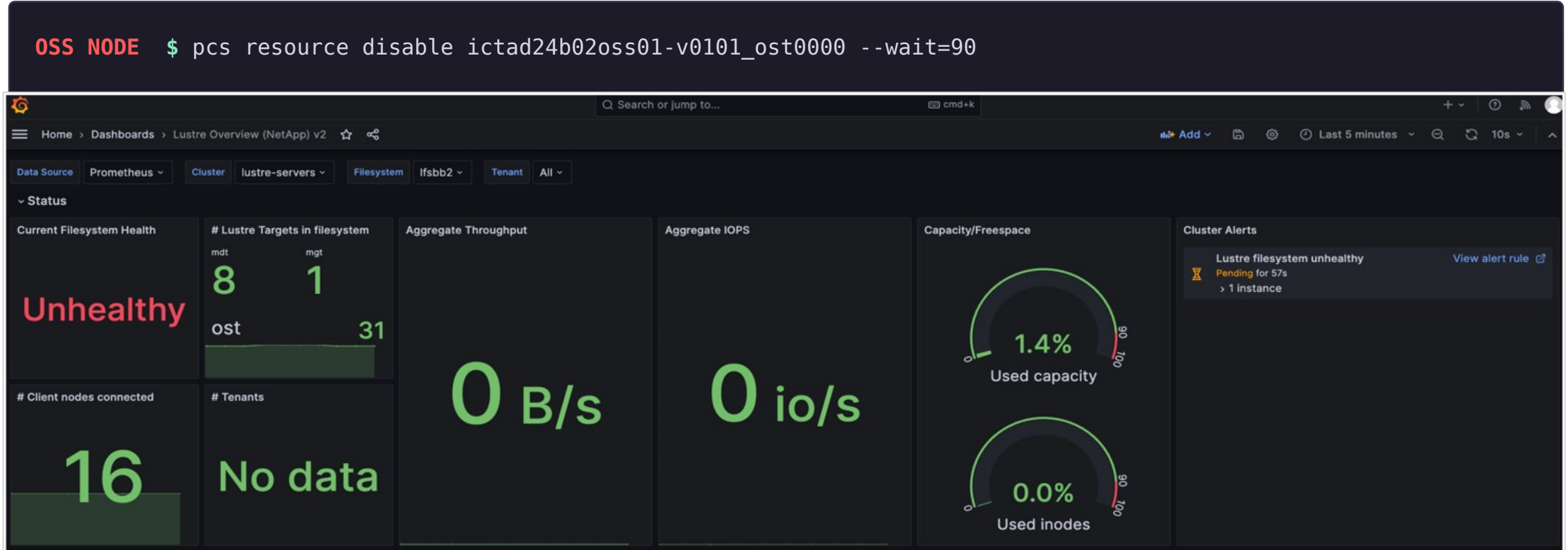
Ansible deploys the monitoring stack too - metrics exporters and Grafana dashboards



Metrics included — read/write bandwidth, IOPs, OST/MDT capacity, HA cluster status, and connected clients.

Break it, on purpose

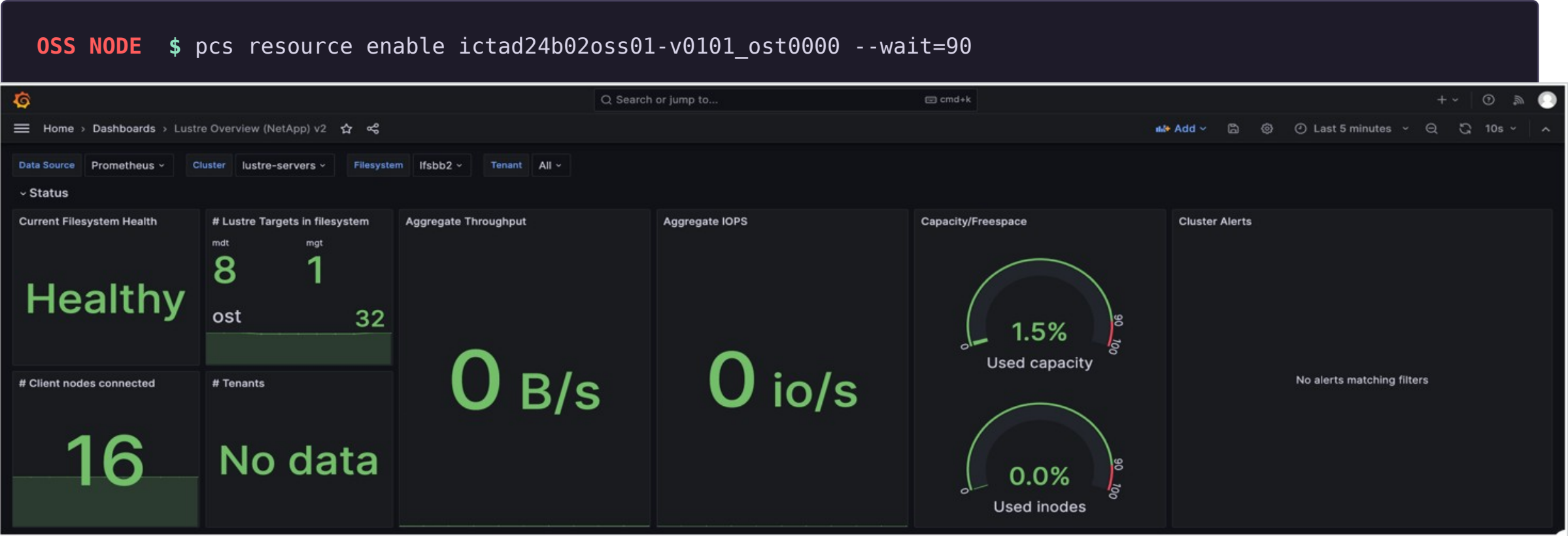
We deliberately take a node down — cluster is unhealthy.



The dashboard reflects the failure immediately — a cluster alert is generated too.

Heal it

We bring the resource back — the cluster is healthy again.



Configuring tenants

Three equally-sized tenants, each mapped to its own group of Lustre clients.

 -i + inventory — which hosts

 Playbook — what to run

 --tags — a subset of roles

 -e — per-run variables

CONTROL NODE — CREATE TENANT 1

```
$ ansible-playbook -i hosts.yml lustre_tenant_playbook.yml --tags multitenancy -e "tenant=tenanta  
clients=node01 quota=1TB"
```

CONTROL NODE — CREATE TENANT 2

```
$ ansible-playbook -i hosts.yml lustre_tenant_playbook.yml --tags multitenancy -e "tenant=tenantb  
clients=node02:node03 quota=2TB"
```

CONTROL NODE — CREATE TENANT 3

```
$ ansible-playbook -i hosts.yml lustre_tenant_playbook.yml --tags multitenancy -e "tenant=tenantc  
clients=node04:node05:node06 quota=5TB"
```

Tenancy, added live — three tenants, isolated by client group, deployed using the same playbook pattern.

All three tenants running together

Visualizing an IOR benchmark, launched against all three tenants at the same time from the control node.

```
COMPUTE NODES $ mpirun -H {node01|node02,node03|node04,node05,node06} ... -o /mnt/tenant{a,b,c}/ior & wait
```

WRITE PHASE



READ PHASE



Shared, not starved — by default. What happens when you need a harder guarantee than that comes next.

What tenancy does today — and what's next

Identity and permission isolation work today; rate limits and richer quotas arrive in Lustre 2.18.

REAL TODAY

Identity, permission & data isolation

- Nodemaps set admin/trusted flags, UID/GID squash, and RBAC per tenant
- Filesets (sub-directory mounts) isolate namespaces

POSSIBLE, NOT AUTOMATED

Network isolation & rate limits

- Clients identified by NID, not a cryptographic key
- Network isolation is manual:
 - VLAN segmentation
 - Encrypted traffic (GSS/SSK)
- Rate limits set by hand with `lctl`

ROADMAP — COMING SOON

Rate limits & quota aggregation

- Nodemap-native NRS TBF rate limits ([LU-17902](#))
- Quota aggregation across project IDs ([LU-18222](#))
- Both targeted for Lustre 2.18

Lustre 2.18 brings the capability — the collection still has to automate it.

How the community can use ansible-lustre

It is open source under BSD-3 — use it as-is, extend it, or contribute back.

1

USE IT AS-IS

Deploy an E-Series building block

Every role in this demo (HA, LNet, clients, tenancy, monitoring) is tested end to end and available today.

2

ADAPT IT

Point it at your storage

One provisioning call to replace. Give your own the same inputs and return the same facts, and every Lustre role runs unchanged.

3

CONTRIBUTE BACK

Send fixes and reports

Issues, patches, and test reports help from any environment — E-Series included, not just new backends.

Source github.com/netapp/ansible-lustre

Install `ansible-galaxy collection install netapp_eseries.lustre`

Issues & PRs github.com/netapp/ansible-lustre/issues

Depends on github.com/netapp/host • github.com/netapp/santricity

01

Lustre deployment becomes repeatable: one command, the same result every run

02

NetApp has released it as an open-source, BSD-3-licensed Ansible collection

03

Take it, adapt it, or contribute: github.com/netapp/ansible-lustre

IT'S
EASY
TO
SEE

QUESTIONS ?

MERCI