TOPF

Open-Sourcing PostFinance's Tool to Migrate and Manage Talos Linux Clusters

Clément Nussbaumer — PostFinance

PostFinance

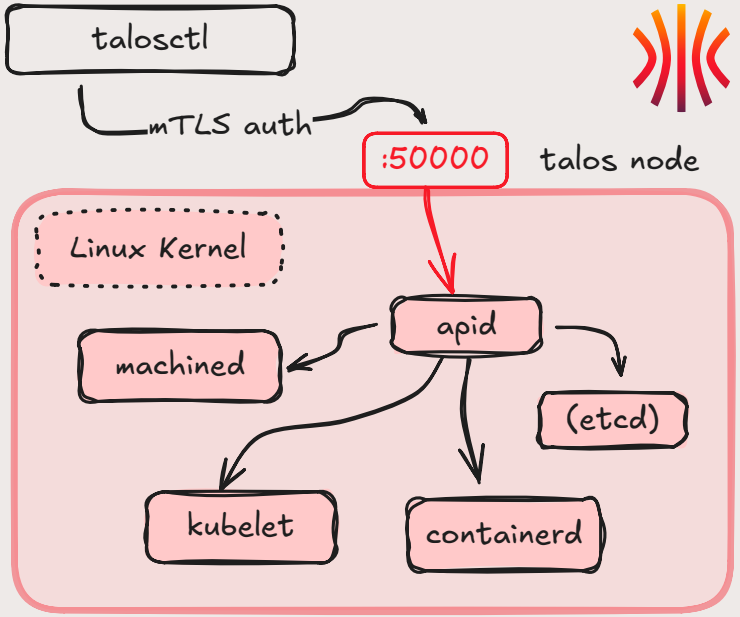
clement.n8r.ch

Talos Linux

A minimal O.S. — fewer than 50 binaries

> immutable, minimal, secure declarative configuration file and gRPC API [1]

```
$ talosctl services
SERVICE                 STATE     HEALTH
apid                    Running   OK
containerd              Running   OK
cri                     Running   OK
etcd                    Running   OK
kubelet                 Running   OK
machined                Running   OK
syslogd                 Running   OK
trustd                  Running   OK
udevd                   Running   OK
```



1. https://www.talos.dev/v1.12/learn-more/philosophy/

Starting Point

kubeadm + Ansible — battle-tested, but showing its age

```
$ tree tasks
tasks
├── cri/
├── etcd-restore.yml
├── kernel/
├── kubeadm_addons/
├── kubeadm_master/
├── kubeadm_prepare/
├── kubeadm_worker/
├── preflight_check_master/
├── preflight_check_worker/
├── sysctl/
├── master.yml
├── worker.yml
├── reset-master.yml
├── reset-worker.yml
└── …

10 directories, 32 files
```

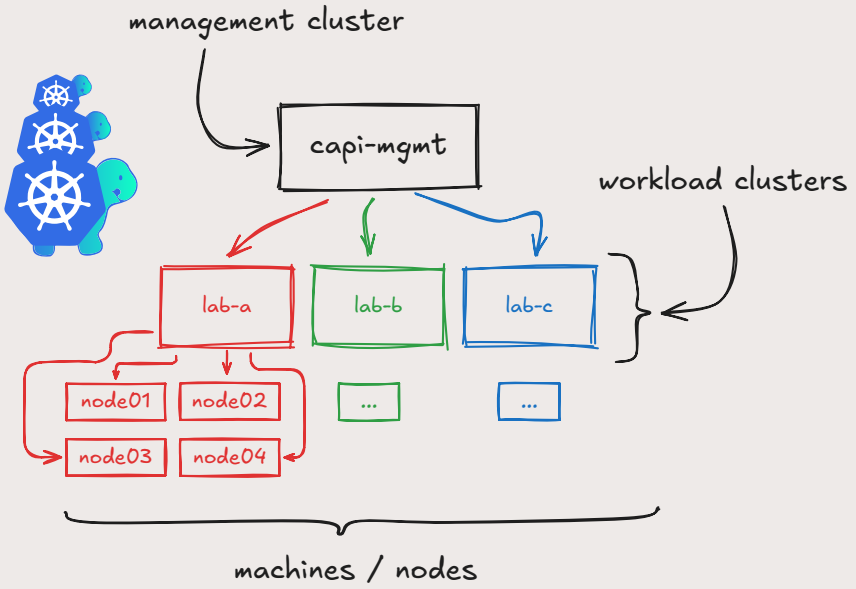
Infrastructure: Ansible + kubeadm

- **32** Ansible task files · **1,878** lines of playbooks
- **20** Jinja2 templates — kubeadm, containerd, sysctl, …

> **mostly idempotent — but not always.** Actions gated on a template's `changed` flag could be silently skipped after a mid-run failure.

Our initial plan: ClusterAPI + Talos

- **declarative** cluster lifecycle
- MachineDeployments — Deployments, but for nodes
- Talos provider for machine config
- GitOps-friendly: ArgoCD watches the CRDs



Change of Plans

why we're not doing ClusterAPI (for now)

SideroLabs pivoted

- Talos CAPI providers → **low priority** [1]
- focus moved to **Omni** [2]

Our concerns

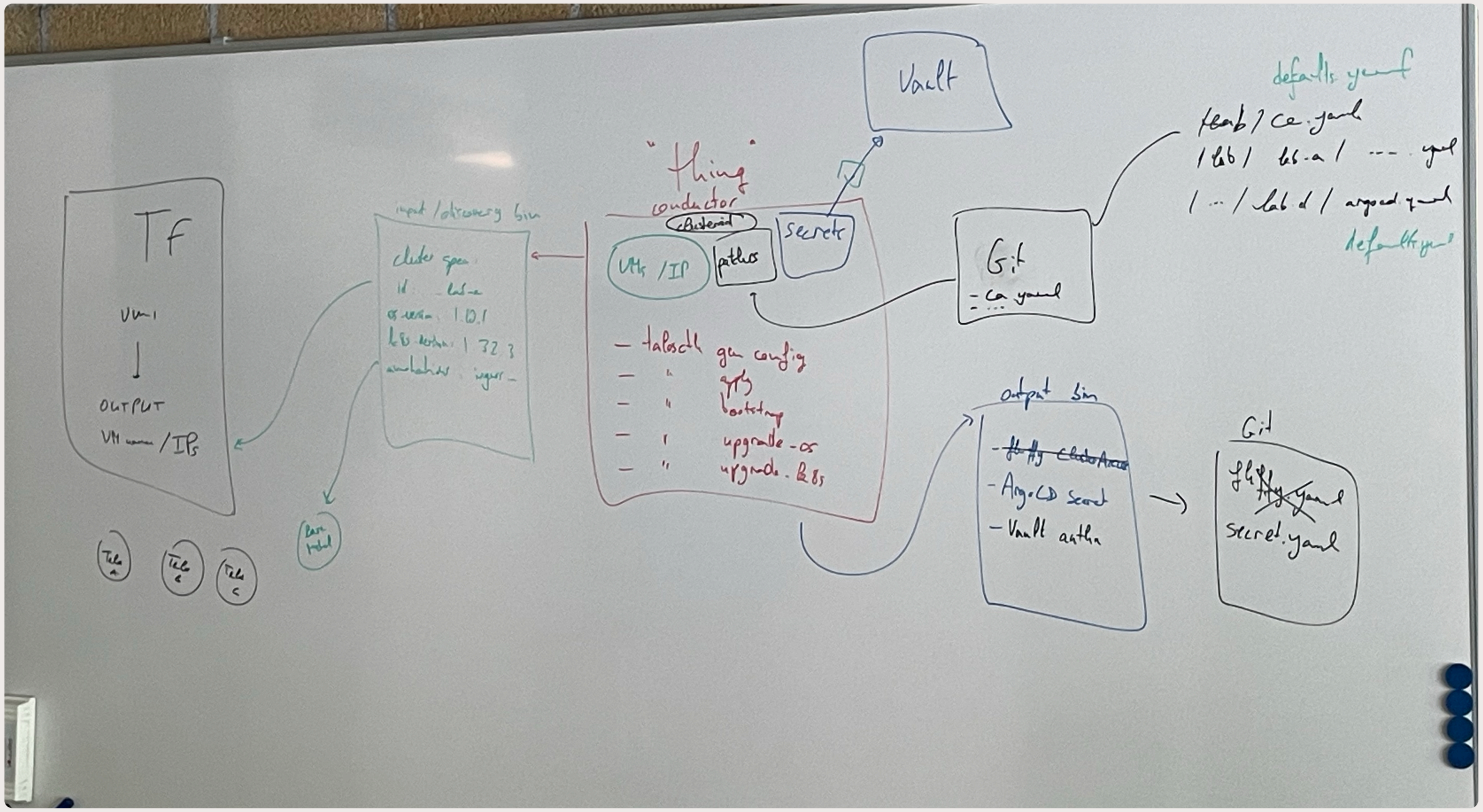
- Kubernetes to manage Kubernetes
- in-place upgrades vs. machine replacement
- CAPI complexity vs. our simplicity goal

So, our options

- **Omni** — becomes an authn proxy in front of the API server (`client → Omni → apiserver`)
- **Ansible** wrapping `talosctl` ? 😬
- a **purpose-built tool** for Talos 🪴

1. https://github.com/siderolabs/cluster-api-bootstrap-provider-talos/issues/193#issuecomment-2449472526
2. https://www.siderolabs.com/blog/kubernetes-cluster-full-lifecycle-management-without-cluster-api/

Thing



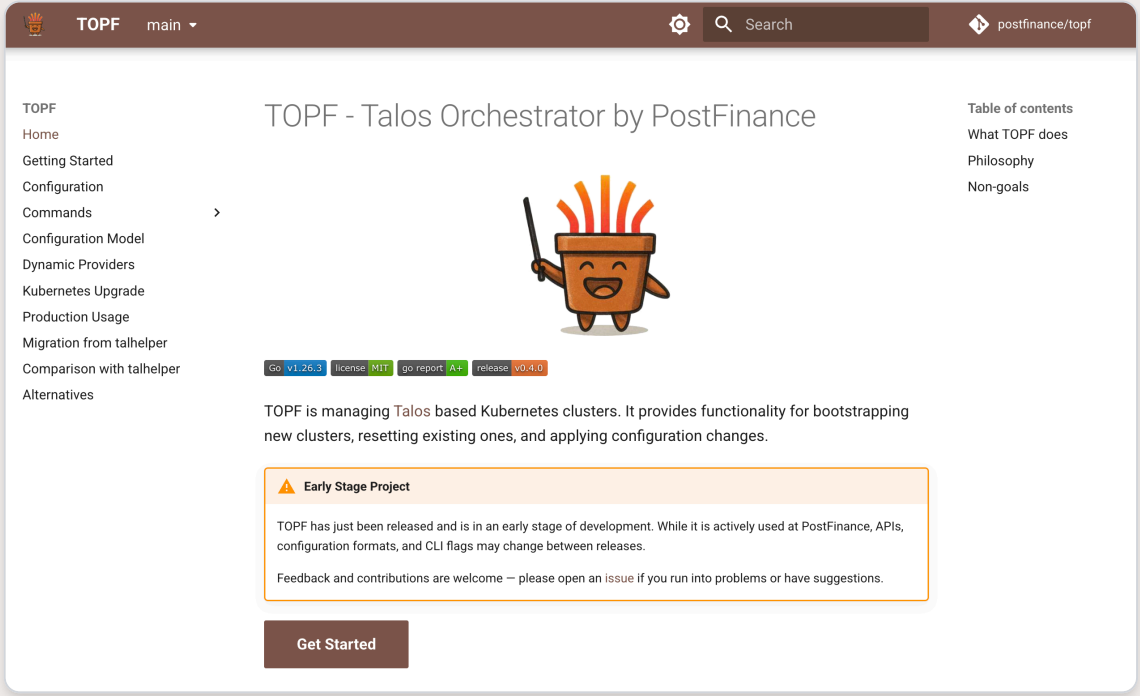
The original whiteboard sketch of what would become TOPF

TOPF

Talos Orchestrator by PostFinance

Not an operator.
Not a controller.
A CLI tool.

🪴 *Topf is also German for a **(plant) pot** — fitting, since it's where our clusters grow.*



What TOPF adds to Talos

the verbs are familiar (`apply` , `upgrade` , `reset` …) — the workflow is the point

GitOps patch management

- layered YAML patches, kept DRY
- Go templating + sprig
- every change is a reviewable PR

inventory & secrets as data

- nodes from external providers
- secrets resolved at render time
- nothing hardcoded in the repo

safe rollouts

- pre-flight checks before touching nodes
- dry-run diffs of every change
- controlled, parallel patch + OS upgrades

Talos already gives you `apply` / `upgrade` / `reset` . TOPF wraps them in a config model, an inventory, and guardrails.

Layered YAML Patches

```
.
├── all/
│   └── 01-install-disk.yaml
└── topf.yaml
```

Start simple: one patch applied to **every** node.

- `topf.yaml` — the entry point

```yaml
clusterName: my-cluster
clusterEndpoint: https://192.168.1.80:6443
talosVersion: 1.13.3
kubernetesVersion: 1.35.5
nodes:
  - host: node1
    ip: 192.168.1.11
    role: control-plane
```

- `all/01-install-disk.yaml`

```yaml
machine:
  install:
    disk: /dev/sda
```

Layered YAML Patches

```
.
├── all/
│   └── ● 01-install-disk.yaml
├── control-plane/
│   └── ● 01-allow-scheduling.yaml
├── worker/
│   └── ● 01-gpu.yaml
└── ● topf.yaml
```

● control-plane/01-allow-scheduling.yaml

```yaml
cluster:
  allowSchedulingOnControlPlanes: true
```

● worker/01-gpu.yaml

```yaml
machine:
  kernel:
    modules:
      - name: nvidia
      - name: nvidia_uvm
```

Add **role** layers: control-plane and worker get their own patches.

Layered YAML Patches

```
.
├── all/
│   ├── 01-install-disk.yaml
│   └── 02-provider-id.yaml.tpl
├── control-plane/
│   └── 01-allow-scheduling.yaml
├── worker/
│   └── 01-gpu.yaml
├── node/
│   └── node1/
│       └── 01-install-disk.yaml
└── topf.yaml
```

- all/02-provider-id.yaml.tpl

```yaml
machine:
  kubelet:
    extraArgs:
      provider-id: {{ .Node.Data.uuid }}
```

- node/node1/01-install-disk.yaml

```yaml
# node1 has different hardware → override
machine:
  install:
    disk: /dev/nvme0n1
```

Live Demo 🪴

github.com/postfinance/topf

What's New

recent additions since the first release

vals secret resolution

resolve secrets at render time [1]

```
network:
  interfaces:
    - interface: wg0
      wireguard:
        privateKey: ref+awsssm://secrets/wg.key
```

sprig templating

full sprig function library in `.yaml.tpl` patches [2]

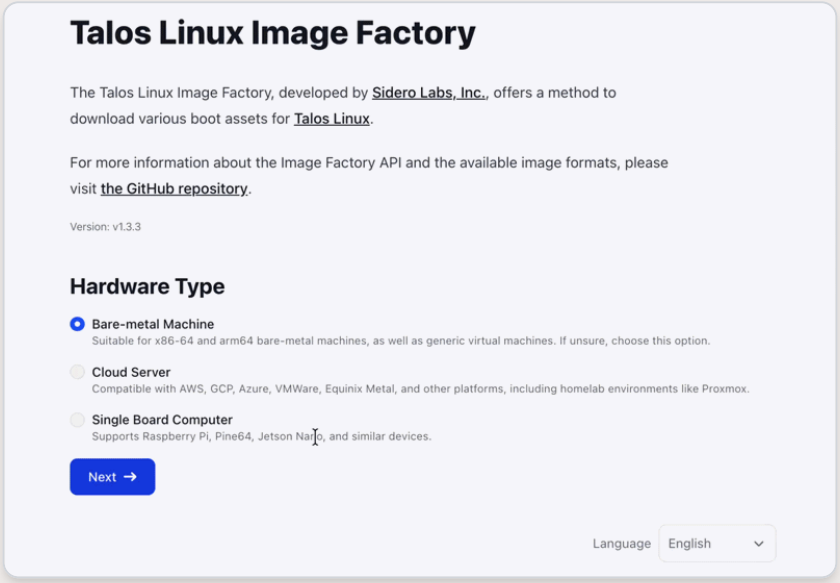
```
machine:
  nodeLabels:
    region: {{ .Data.region | default "eu" }}
    host: {{ .Node.Host | lower }}
```

1. https://github.com/helmfile/vals
2. https://github.com/masterminds/sprig

Schematic Resolution

stop juggling Talos Factory [1] image hashes

Before — the Factory UI



After — a manifest in Git

- `manifest.yaml.tpl`

```yaml
customization:
  extraKernelArgs:
    - node-arg-{{ .Node.Host }}
  systemExtensions:
    officialExtensions:
      - siderolabs/vmtoolsd-guest-agent
```

- `topf.yaml`

```yaml
schematicId: "@manifest.yaml.tpl"
```

1. https://factory.talos.dev/

Dynamic Providers

resolve node lists and secrets at runtime from external sources

Nodes Provider

instead of a static list…

```
# topf.yaml
nodes:
  - host: node1
    ip: 192.168.1.11
    role: control-plane
```

…point at a binary:

```
# topf.yaml
nodesProvider: ./nodes-from-terraform
```

e.g. read nodes from a Terraform output, cloud API, or inventory

Secrets Provider

default: a local SOPS-encrypted bundle…

```
# secrets.yaml (SOPS-encrypted)
secrets:
  bootstrapToken: ENC[AES256_GCM,data:…]
```

…or fetch it from an external store:

```
# topf.yaml
secretsProvider: ./secrets-from-vault
```

e.g. Vault, AWS, any external secret store

TOPF in GitLab CI

one pipeline per cluster — GitOps for day-2 ops

```yaml
# .gitlab-ci.yml
default:
  before_script:
    - git clone --branch "$CONFIG_REF" "https://$CONFIG_REPO" .

dry-run:
  script: topf apply --dry-run

apply:
  script: topf apply --confirm=false
  when: manual

upgrade:
  script: topf upgrade --confirm=false
  when: manual
```

```
$ topf apply --dry-run

  node2  (Mode: NO_REBOOT)
  machine:
    install:
-     disk: /dev/sda
+     disk: /dev/nvme0n1
    kernel:
      modules:
+       - name: nvidia

  1 node changed — exit code 2
```

TOPF vs Alternatives

scripts / Ansible

Imperative `talosctl` wrapping

- no safety net, hard to audit

talhelper

Declarative config generation

- but you still apply & manage lifecycle

Omni (SideroLabs)

Turnkey managed platform

- authn proxy in front of the API server

Takeaways

three lessons that outlast the tool

1. The hard part was never the OS — it was lifecycle
2. You don't always need a control plane to manage one
3. Layered, structured config is easy to work with and simple to review

PostFinance

Thank you!

postfinance/topf

talos.dev

clement.n8r.ch

Questions? 🪴