# OpenVox Operator

## Puppet Infrastructure as Kubernetes CRDs

Simon Lauger

VoxConf 2026 · Garching near Munich

# How it started

**The idea in 2018**

Run the Puppet Server in OpenShift 3 - I tried. I failed.

**OSCamp #2 on Puppet · Nuremberg · November 2018**

After Martin Alfke's talk *"Ops Hates Containers. Why?"* I asked him:

- *"Ever run Puppet Server on Kubernetes?"* → **No.**
- *"The server tries to start as root, which does not work on OpenShift. A fix?"* → **No.**

There was no solution at the time. But the idea stayed with me.

# Fast forward to 2026

What was hard back then is finally doable:

- **AI** massively sped up the research and the **Go** development
- It even helped analyze the rootless startup and ship a patch
- OpenVox Server now starts **fully rootless**
- **OpenVox**, the community fork of Puppet, means fixes and features land faster

AI-assisted research and development helped turn the idea into a working implementation.

# Need another compile server?

The usual way:

- New VM, install packages
- Join the CA, sign the cert
- Add it to the load balancer
- Add it as a target in your r10k pipeline
- Patch it. Forever.

> **What we'll do today:** all of that, in Kubernetes, from **a handful of YAML files**.

# About me

- **Simon Lauger**, freelance Platform Engineer (Kubernetes, OpenShift, AWS, Azure)
- Living in **Bayreuth**, ~~Bavaria~~ Franconia, Germany
- Puppet since 2010 · Kubernetes since 2017 · Kubestronaut since 2026
- Contributor to Vox Pupuli and OpenVox
- GitHub **@slauger** · Website: lauger.de
- 🤘 Metal · 🎶 Festivals · 🍺 Beer · 🏃 Running · 🏅 Triathlon · 🚒 Firefighter

LinkedIn

# Agenda

- **Introduction to Containers, Kubernetes and Operators**
- **The OpenVox Operator**: CA, certs, server, code, autosign as CRDs
- **One live demo**: empty namespace to a running server, a code deploy and agent runs
- **How to try it yourself**

# Containers, Kubernetes, Operators

What each one is, and who takes care of what

# Containers

A container is a **self-contained, immutable package**: your app plus everything it needs to run.

- built once, runs the same everywhere
- immutable: you **replace** it, you don't patch it
- on its own: **no scaling, no self-healing**

**You** run it, restart it, scale it, all by hand.

# Kubernetes

More than running containers. A **platform of declarative building blocks**:

- **Run & scale**: Deployments, Pods, self-healing
- **Networking**: Services, Gateway API
- **Config & secrets**: ConfigMaps, Secrets
- **Storage**: Volumes, OCI image volumes

You declare the desired state; Kubernetes controllers keep it true.

# What is a CRD (Custom Resource Definition)?

**A CRD extends the Kubernetes API with your own resource type.**

A Custom Resource Definition lets you:

- define new resources like `CertificateAuthority`, `Server` or `Pool`
- manage them with `kubectl apply`, `kubectl get`
- watch for changes like any other Kubernetes resource
- add **behavior** through a controller (the operator)

# Kubernetes Operators

The idea: **Human ops knowledge → the Kubernetes control loop.**

An Operator extends Kubernetes with application-specific control loops.

- watches **Custom Resources** (defined by CRDs)
- keeps desired and actual state in sync (reconciliation)
- automates ops tasks: **deployments, config, updates, backups, recovery**

# What a CRD looks like

```yaml
apiVersion: openvox.voxpupuli.org/v1alpha1
kind: Server
metadata:
  name: production
spec:
  configRef: production
  certificateRef: production-cert
  image:
    repository: ghcr.io/slauger/openvox-server
    tag: "0.9.6"
  replicas: 3
  poolRefs: [puppet]
```

You declare *what* you want. The operator makes it real, and keeps it that way.

# Why not just run the existing OpenVox image?

**OpenVox Server images exist**, but they're designed for standalone **Podman**, not Kubernetes or OpenShift:

- **~15 entrypoint scripts** turning ENV vars into config files
- extra **agent Ruby** just to run those scripts
- **volume-mount permission** clashes with OpenShift's random UIDs

*Fine for `podman run`. Not how you run things in Kubernetes.*

> The operator replaces those scripts: it renders config into **ConfigMaps and Secrets**, and handles rollouts, restarts and reconciliation.

# The OpenVox Operator

A whole **OpenVox / Puppet infrastructure**, mapped onto Kubernetes as **9 CRDs**:

- **CertificateAuthority**: the Puppet CA (bootstrap, sign, renew, CRL)
- **Certificate**: a single TLS certificate
- **Config**: shared settings for a group of servers
- **Server**: a compile or CA server (a Deployment)
- **Pool**: networking and SNI routing (Service, Gateway API)
- **Database**: managed OpenVox DB (PuppetDB)
- **SigningPolicy**: declarative autosign rules
- **NodeClassifier**: external node classification (ENC)
- **ReportProcessor**: where Puppet run reports go

# Certificates

- the operator manages `Certificate` objects and **auto-renews** them
- multiple Servers can share **one** certificate, so replicas and scaling just work
- need a new **DNS name** (SNI host)? the operator **injects it** into the cert, automatically
- **agent certs** aren't operator-managed: they go through the CA as usual, and the agent renews its own

# Declarative autosign: SigningPolicy

```yaml
apiVersion: openvox.voxpupuli.org/v1alpha1
kind: SigningPolicy
metadata:
  name: trusted-with-psk
spec:
  certificateAuthorityRef: production-ca
  pattern:
    allow: ["*.example.com"]
  csrAttributes:
    - name: pp_preshared_key
      valueFrom:
        secretKeyRef: { name: signing-psk, key: psk }
```

Match by certname pattern, DNS SANs or **CSR attributes** (here: a pre-shared key).

No `autosign.conf`, no Ruby: rendered to a Secret, enforced by the shipped **`openvox-autosign`** Go binary.

# External node classification: NodeClassifier

```yaml
apiVersion: openvox.voxpupuli.org/v1alpha1
kind: NodeClassifier
metadata:
  name: my-classifier
spec:
  url: https://classifier.example.com
  request:
    method: POST
    path: /classified/nodes/{certname}
    body: facts
  response:
    format: yaml
  auth:
    basic:
      secretRef: { name: classifier-creds, usernameKey: username, passwordKey: password }
  cache:
    enabled: true
```

Point at your ENC (Puppet Enterprise, or any HTTP classifier), auth from a Secret.

The shipped `openvox-enc` Go binary **caches to disk**, so a classifier outage doesn't break runs.

# Code deployment, the Kubernetes way

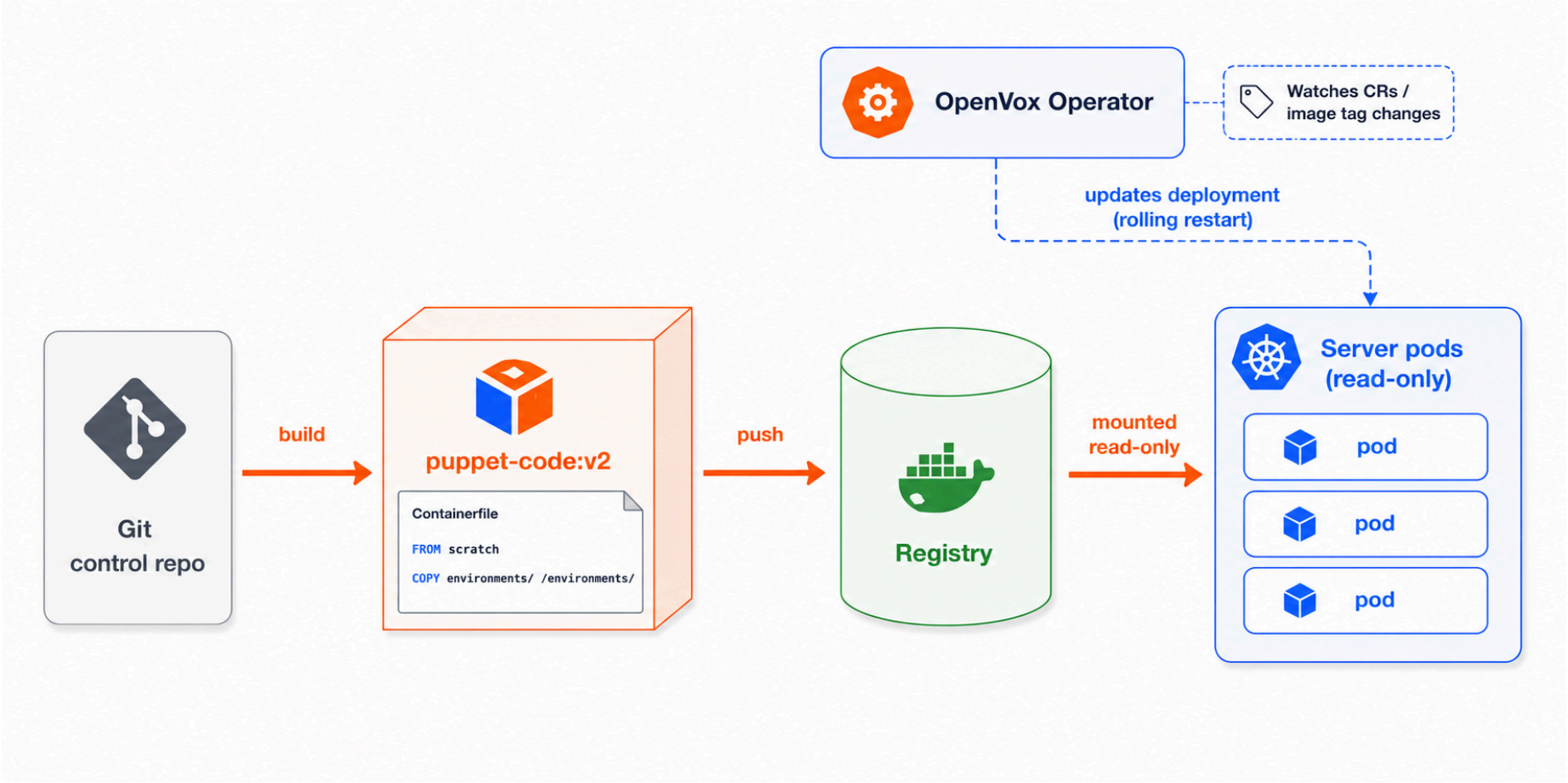
How do you get Puppet code onto the servers? Two declarative options:

- **OCI image volume**: code baked into an immutable, tagged image (recommended)
- **PVC (RWX)**: a shared, mutable volume, if you deploy code another way

The problem with immutable images: a code change means a **restart**. But the operator gives us options to handle that.

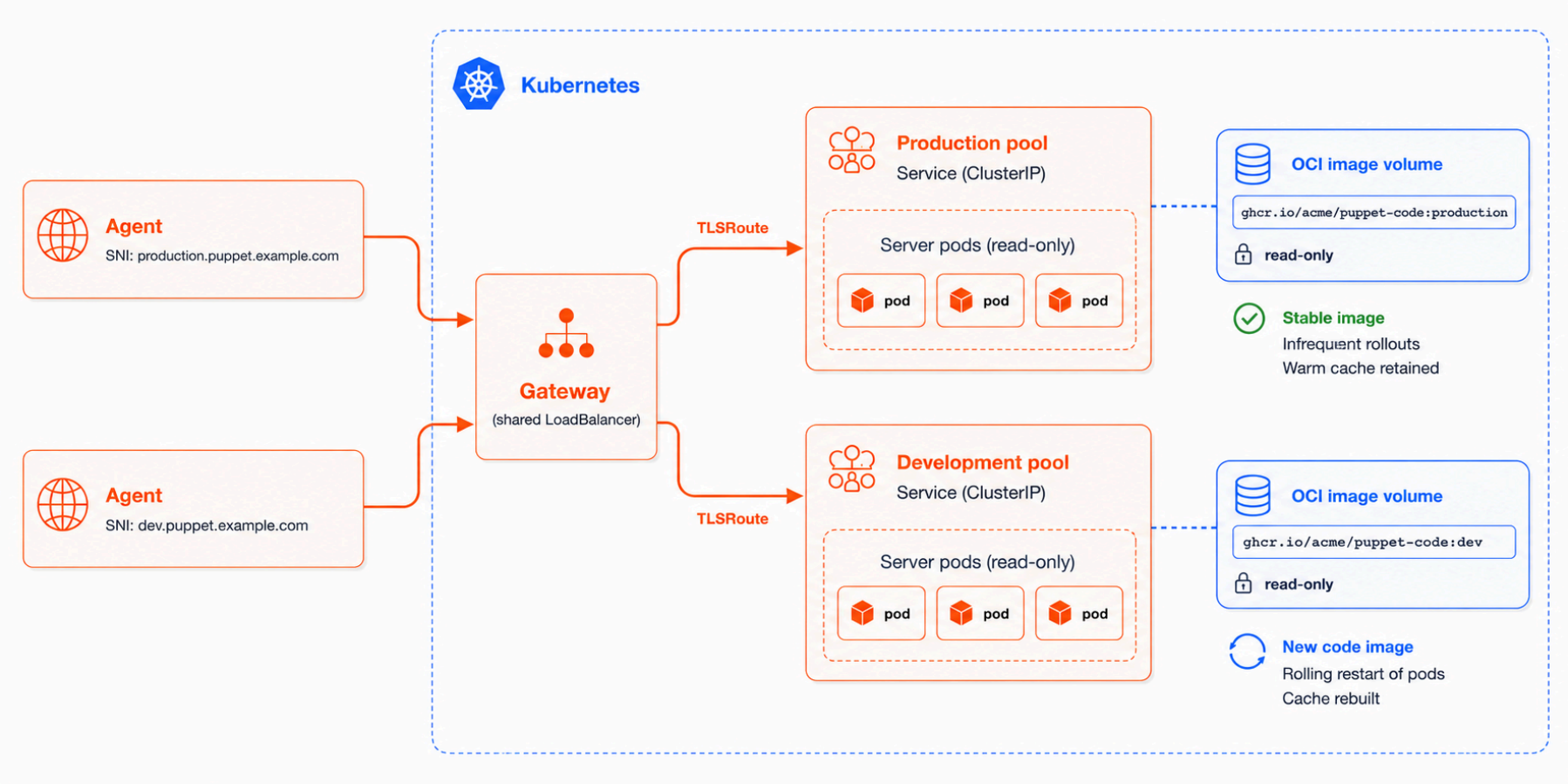
# Code as an OCI image volume

Your Puppet code **IS a container image**: immutable, versioned by tag, no r10k, no cron.



# Multiple Pools to reduce restarts in production

Give `staging` and `production` their own **Pools**, so a dev commit never restarts prod. Feature branches can even get their own **ephemeral environment**.



# A tool that can help: openvox-code

`openvox-code`: a Go based alternative to r10k/g10k. Single static binary, **no Ruby, no Puppetfile** (clean YAML), git-first with bare-clone caching, offline-capable.

```
# Build your control repo into an OCI code image and push it
$ openvox-code build -t ghcr.io/acme/puppet-code:v2 --push
```

Then the operator consumes it by bumping one tag:

```
config:
  code:
    image: ghcr.io/acme/puppet-code:v2   # was :v1
```

*Still in development; feedback and contributions welcome.*

# Demo

Empty namespace → server up → code deploy → agent run

# Try it yourself

```
# 1) install the operator
$ helm install openvox-operator oci://ghcr.io/slauger/charts/openvox-operator \
    -n openvox-system --create-namespace

# 2) deploy a stack
$ helm install openvox oci://ghcr.io/slauger/charts/openvox-stack \
    -n openvox --create-namespace -f values.yaml
```

- **Kubernetes 1.35+** if you want OCI image volumes (or 1.31+ with the `ImageVolume` gate; PVC works on any supported version)
- a **Gateway API** implementation (optional) for SNI/TLS routing
- **cert-manager** (optional) for the admission webhook's TLS

**On GitHub:**

- `github.com/slauger/openvox-operator` (the operator)
- `github.com/slauger/openvox-code` (optional, for building code images)

# What else?

**Managed OpenVox DB**

Provision OpenVoxDB (PuppetDB) with the Database CRD, connecting to an external Postgres (e.g. CloudNativePG).

**Gateway API routing**

Share one LoadBalancer across environments with SNI-based TLSRoute and TLS passthrough.

**Report processing**

Forward Puppet reports declaratively to OpenVoxDB or custom HTTP endpoints.

**Observability**

Monitor server replicas, certificate expiry, and controller metrics with an optional ServiceMonitor.

**Supply-chain security**

Cosign-signed images, SLSA provenance, SBOMs, and Conforma policy validation.

# Questions?

**Simon Lauger** · simon@lauger.de · lauger.de

github.com/slauger/openvox-operator

github.com/slauger/openvox-code

LinkedIn