DevOps and kubernetes at LifeWatch ERIC VLIC

From agile development to operational services with a small team

Spiros Koulouzis, Gabriel Pelouze

{gabriel.pelouze, spiros.koulouzis}@lifewatch.eu

LifeWatch ERIC VLIC
Multiscale Networked Systems, University of Amsterdam

Master Security and Network Engineering: Large Systems — 2 December 2024

Introduction

Who we are

- LifeWatch is a European Research Infrastructure Consortium (ERIC) providing **e-Science research** facilities to scientists investigating biodiversity and ecosystem functions
- The LifeWatch Virtual Laboratory & Innovations Centre (VLIC) is implementing **virtual research environments**, for open-source cooperation

Koen Greuel: Developer for Cloud-based Virtual Research Environments

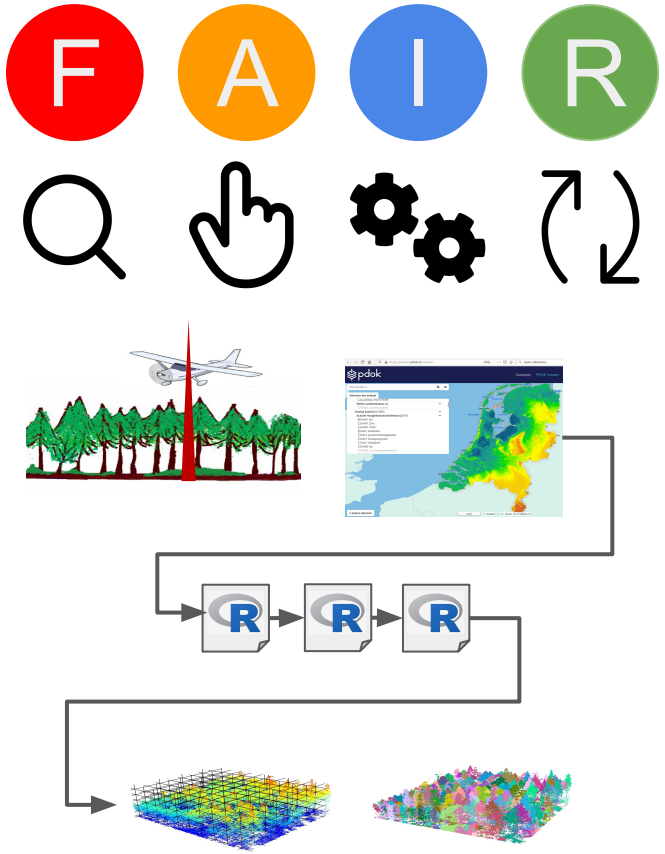
Gabriel Pelouze: Developer for Cloud-based Virtual Research Environments

Spiros Koulouzis: Virtual Research Environment Developer

Zhiming Zhao: VLIC Technical Manager

Identifying habitat types for ecology

- In 2019 we collaborated with the Biogeography & Macroecology (BIOMAC) group to develop a **habitat classification workflow** from a set of **R scripts** that use LiDAR data
- Use the code to **extend the study area** to more parts of the Netherlands and potentially europe
- Make the workflow Findable Accessible Interoperable Reusable (**FAIR**)
- Reach a **bigger community** beyond UvA
- Actions:
  - **Dockerize** the R code
  - Build a **generic workflow** that can run on the cloud



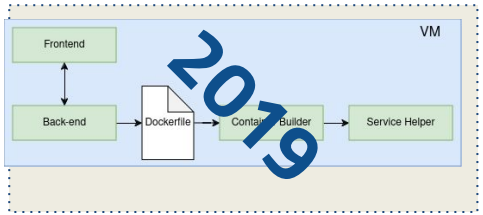
Learning from Failures

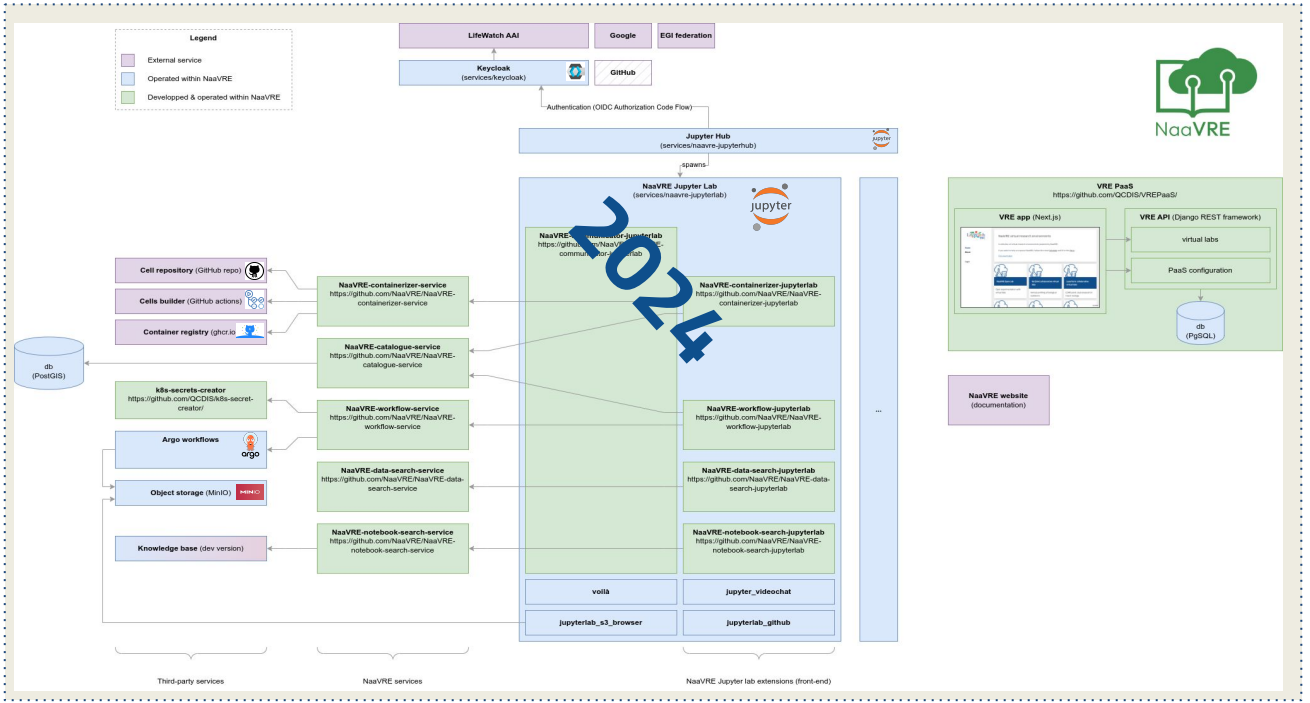
- The exercise failed:
  - Code was developed on a **large HPC** cluster with several **GPUs** and over **26 TB** of RAM (2017 specs)
  - Many **hardcoded** parameters
  - Unfamiliar with R: **Too much time spent** understanding the code and its dependencies
- Lessons learned:
  - We need a framework that enforces **good coding practices** even if the developer has no coding training
  - Easy **code sharing and documentation**
  - **Automate** the **dockerization** of code snippets
  - Enable flexible **input parameters**

```
library("lidR")
library("rgdal")
source("D:/USER/eEcoLiDAR/PhDPaper1_Classifying_
  wetland_habitats/Function_LiDARMetricsCalc.R")

# Set working directory
workingdirectory="D:/USER/for_lifawatch/" ## set this
  directory where your input las files are located
setwd(workingdirectory)

cores=4
chunksize=2000
buffer=1
resolution=10

rasterOptions(maxmemory = 200000000000) # 200 GB
```

From FAIR-cells to NaaVRE

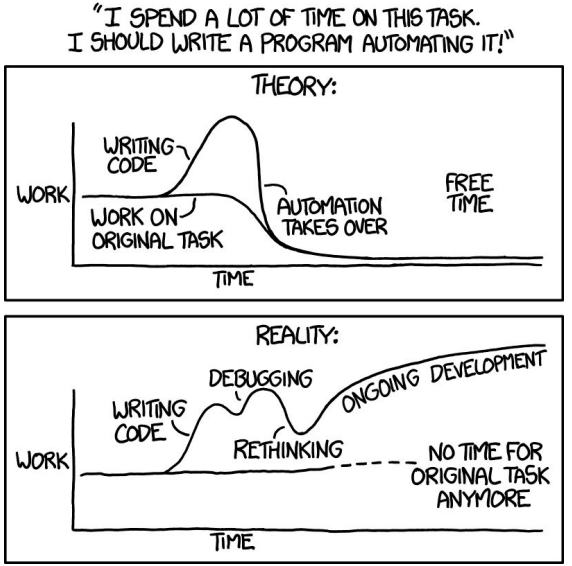
- From 2058 to 39042 lines of code & from one developer to three
- From one JupyterLab addon to sixteen microservices





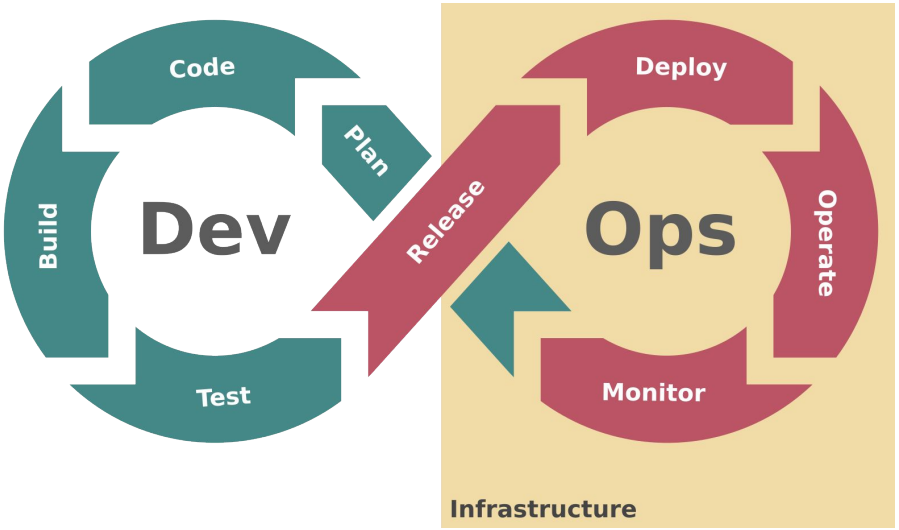
Large infrastructure, small team

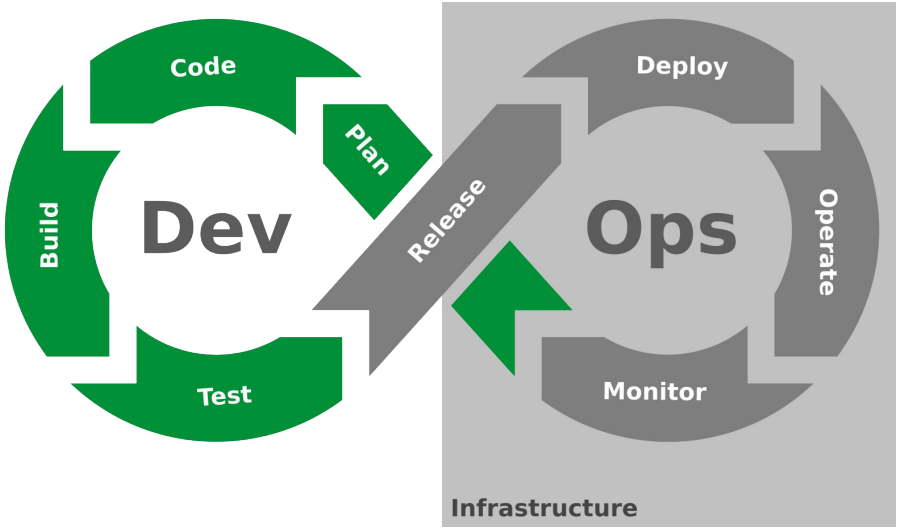
- Objective: deliver stable NaaVRE PaaS to users
- We need to support our own development (DevOps) and infrastructure needs
- We're a small team, we need to be smart with our time and amplify our efforts
- How to choose which features to implement?
- How do we make the code portable?
- How can we have consistent development and deployment environment?
- How do we efficiently test the code and the integrated system?
- When should we release a new version?
- How do we install the PaaS stack (16 microservices)?
- What should we monitor?
- What is our incident management process?



Outline

1. Plan, build, code and test
2. Release
3. Infrastructure
4. Deploy
5. Operate
6. Monitor
7. Integration with external services
8. Summary and takeaways

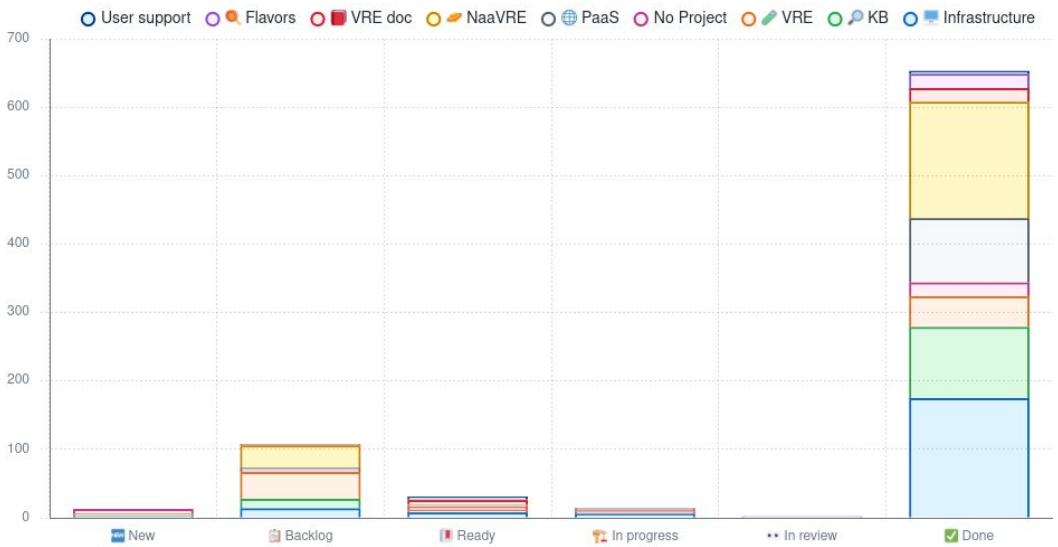




Plan, code, build and test

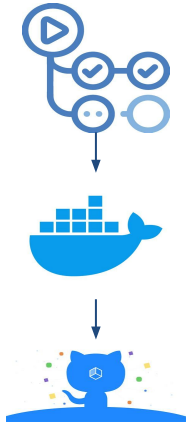
Plan & Code

- **How to choose which features to implement?**
- Talk with end-users
- Increase impact (publication, stability, productivity)
- Minimize coding
- Minimize the amount of services we deploy and maintain
- Use sustainable open-source solutions and technologies



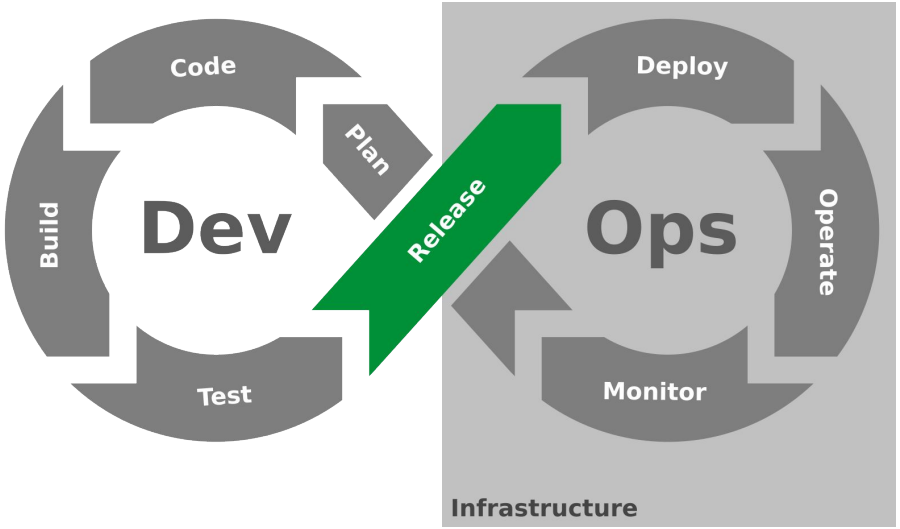
Build & Test

- **How do we make the code portable?**
- Use Github actions to build each services as a container
- We can have isolation and built-in versioning
- **How can we have consistent development and deployment environment?**
- In some cases we need to have dependencies outside python packages
- We use conda to have consistent environment i.e. same version and build of GDAL
- Conda adds a significant **overhead** in the image both in build **time** and **size**
- **How do we efficiently test the code?**
- Bugs must get their own test
- GitHub actions can take a lot of time



CONDA

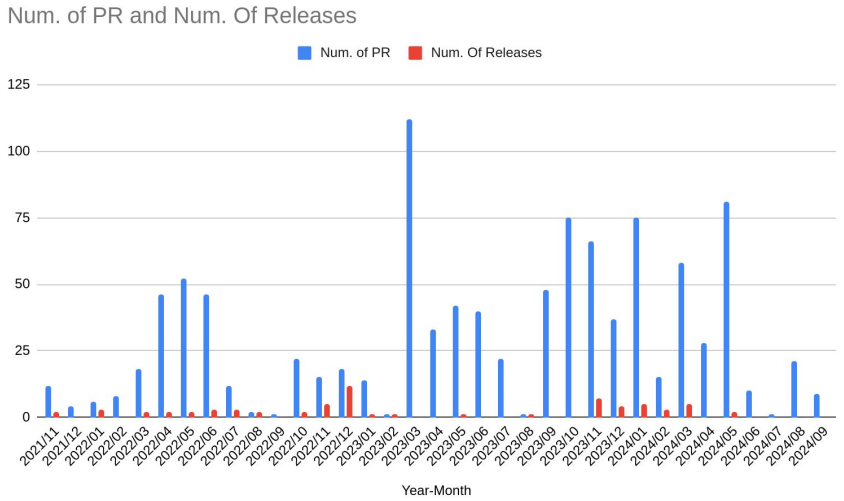
cy

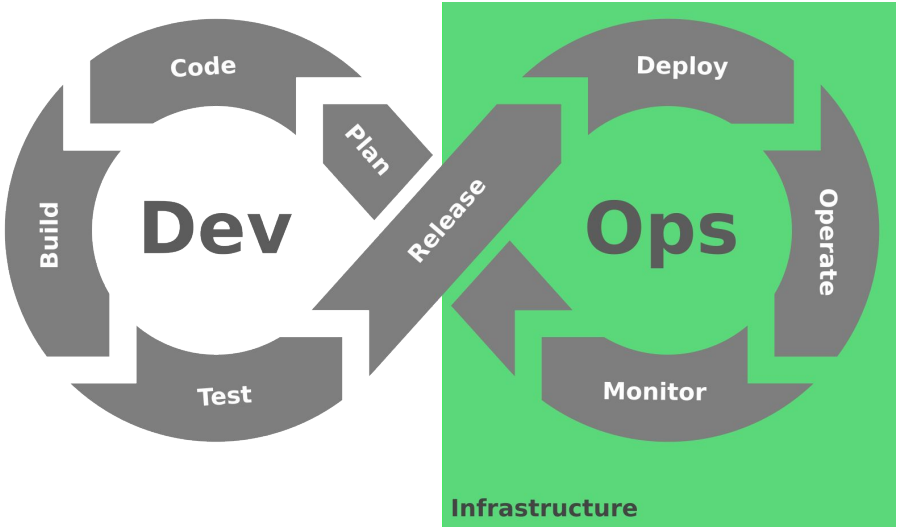

Code
Plan
Build
Dev
Release
Ops
Deploy
Operate
Test
Monitor
Infrastructure

Release

Release

- **When should we release a new version?**
- When we have critical bug fixes
- When a new feature will bring the most impact
- When we have enough new features
- When the time is right
- **How do we install all the PaaS stack?**
- One docker one microservice
- 16 microservices (DB, SSO, etc.)
- Build helm charts packages
- Publish in GitHub helm repository





Infrastructure

Infrastructure: what are our options

Public cloud
High monthly running cost

Research cloud

On-premise
SNE open Lab, LifeWatch ERIC datacenters

User provided
Hardware managed by users or their institute

- Choice driven by funding, often different for each project
- Most infrastructure is best effort
- Wide variety

Is it really the cloud?

Public cloud

SURF

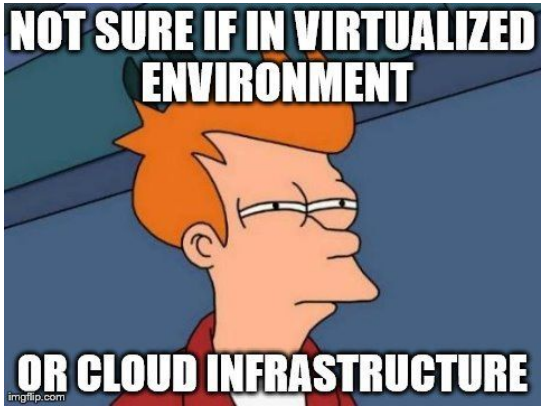
EGI Advanced Computing for Research

Research cloud

On-premise

User provided

VMs	VMs	VMs	VMs	Bare metal
APIs		OpenStack	Emails!	
		Hardware		



- **Cloud == automation**
- Sometimes it's just a bunch of VMs
- Otherwise, it's a mix of multi-cloud and hybrid-cloud
- Sometimes all the way down to hardware provisioning

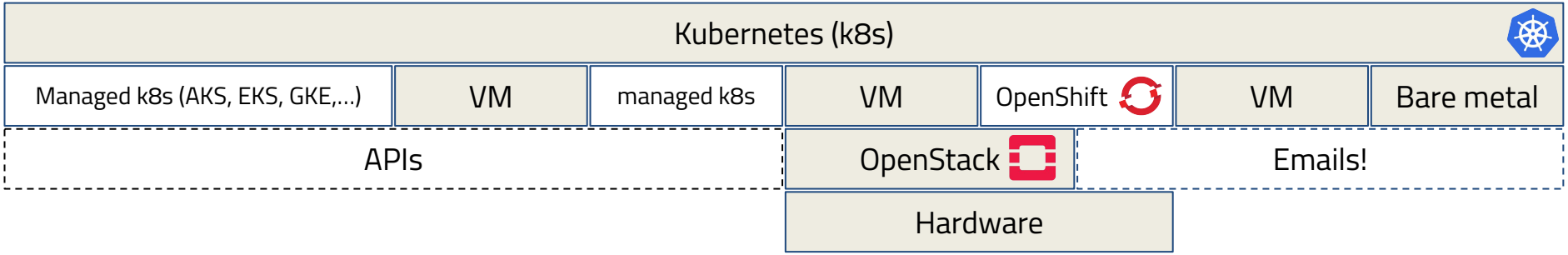
Kubernetes: a unified platform for deploying apps

Public cloud

Research cloud

On-premise

User provided



- Container orchestration across multiple nodes
- Abstraction layer: developers can specify resources for their app using yaml (containers, networking, config, etc.)
- Easy deployment and rollback using Helm package manager

Kubernetes deployment: options

- **Managed k8s**
  - Easy to set up
  - Not always available
  - Expensive in public cloud
- **Self-managed k8s**
  - Hard to set up
  - Sometimes the only choice available
  - Many options to choose from; some are system-dependent
    - Container runtime interface (CRI) → containerd, cri-io
    - Container networking interface (CNI) → calico, flannel, weave
    - Persistent volume provisionner → MinIO, NFS
    - Load balancer and reverse proxy → HAProxy, Nginx, Apache
    - etcd settings
    - Certificates management

Kubernetes deployment: our approach

Starting point: provisioned VMs
Configuration with ansible playbooks

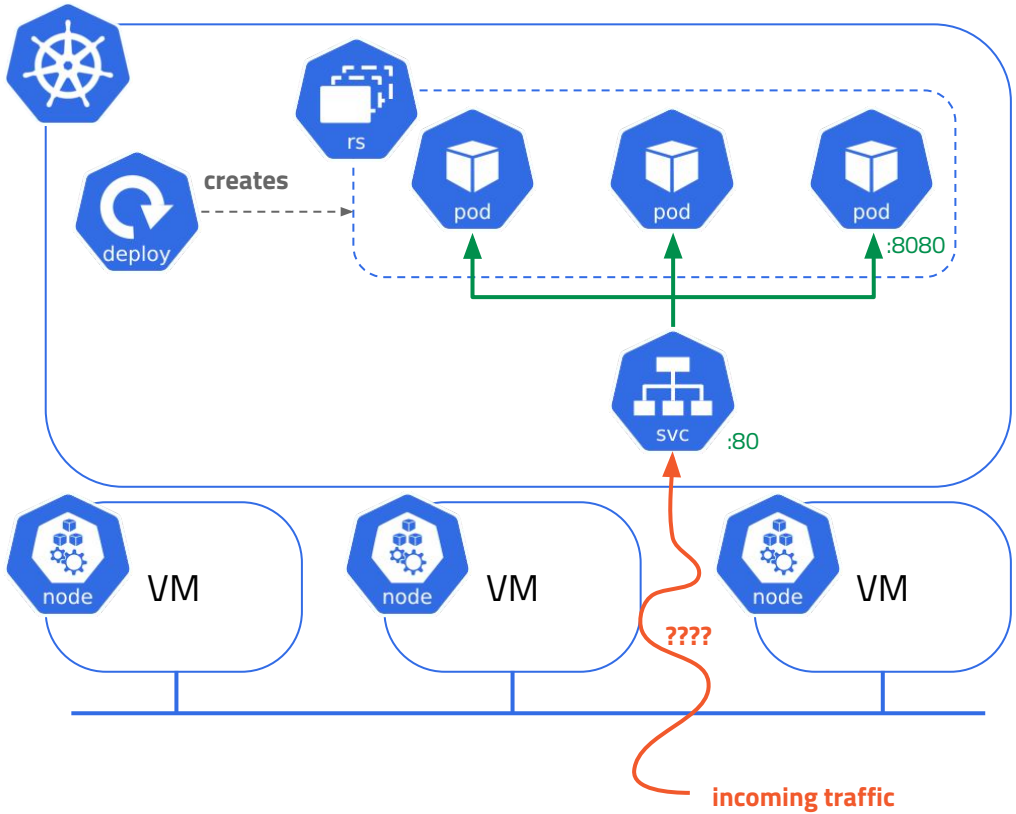
Steps:

- System preparation
  - configure system (kernel modules, sysctl net params, disable swap, unattended upgrades)
  - install software (containerd, kubeadm, kubelet, kubectl, helm)
  - configure firewall rules
- High availability (optional):
    - highly-available IP with VRRP (keepalived)
    - load balancer (haproxy)
    - all deployed as static pods on k8s
- Bootstrap cluster
  - `kubeadm init` and `kubeadm join`
  - install CNI with helm
- Persistent volumes
  - Provision NFS or S3
  - Install volume provisionner with helm
- Ingress controller: installed with helm
- Cluster users

https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

Incoming traffic: load balancing and ingress

Basic workload deployment



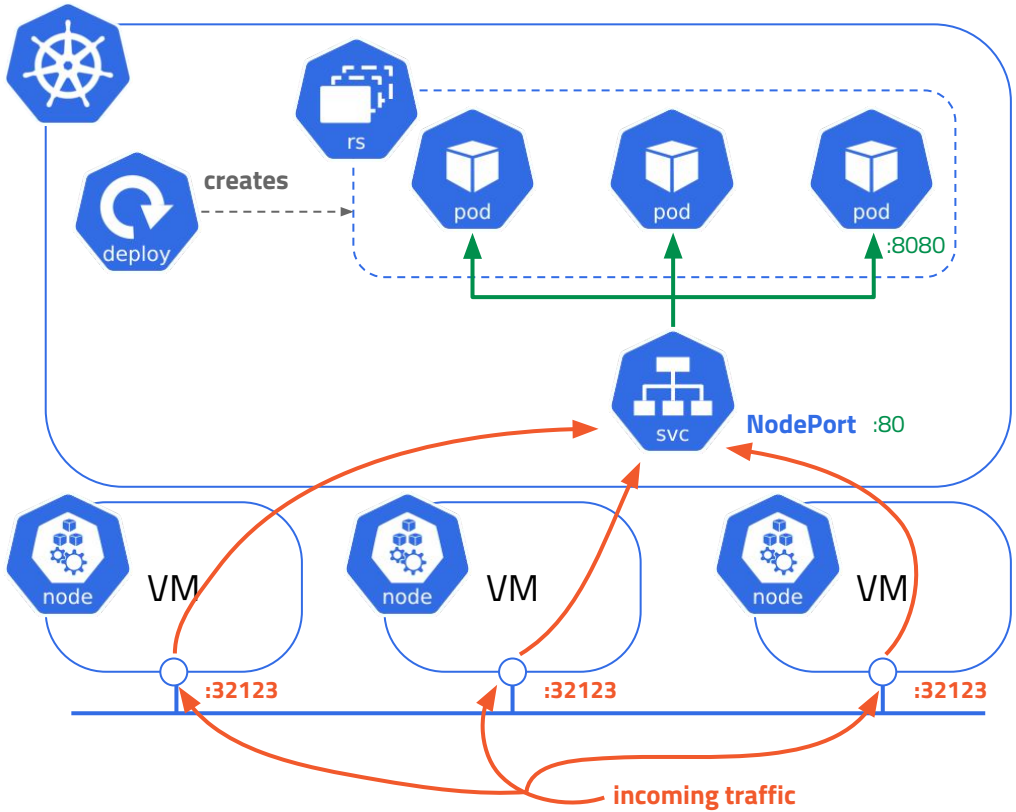
Pods run one or more containers. They are part of a **replica set**, created by a **deployment**.

A **service** provides a unified entry point to a group of pods.

```yaml
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
   app.kubernetes.io/name: MyApp
 ports:
   - protocol: TCP
     port: 80
     targetPort: 8080
```

Incoming traffic: load balancing and ingress

Option 1: NodePort

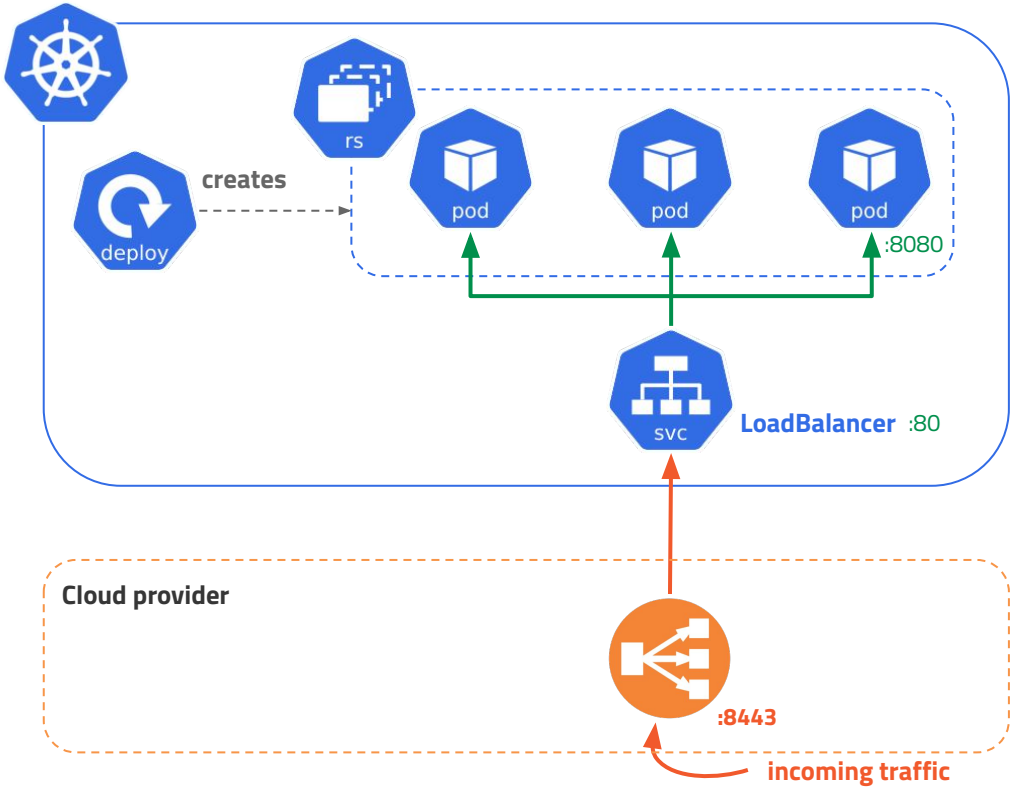


- Only one port per service
- Port range 30000-32767
- Not suitable to receive public traffic
- → good for externally-managed reverse proxy or load-balancer

```yaml
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 type: NodePort
 selector:
   app.kubernetes.io/name: MyApp
 ports:
   - port: 80
     targetPort: 8080
     nodePort: 32123
```

Incoming traffic: load balancing and ingress

Option 2: LoadBalancer

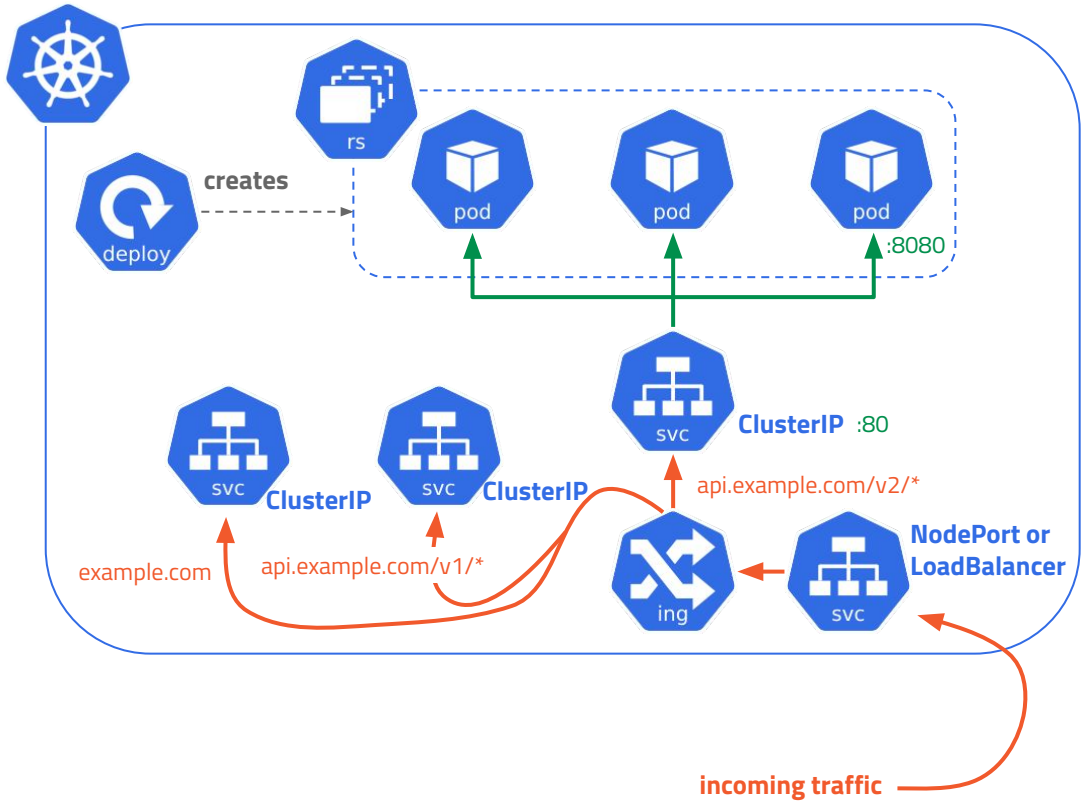


- Provisions load balancer from cloud provider through cloud controller manager
- Only one service per LB
- Suitable for all protocols

```yaml
apiVersion: v1
kind: Service
metadata:
 name: my-service
 annotations:
  service.beta.kubernetes.io/
     aws-load-balancer-type: external
  service.beta.kubernetes.io/
     aws-load-balancer-nlb-target-type: ip
  service.beta.kubernetes.io/
     aws-load-balancer-scheme: internet-facing
spec:
 type: LoadBalancer
 selector:
   app.kubernetes.io/name: MyApp
 ports:
   - protocol: TCP
     port: 80
     targetPort: 8080
```

Incoming traffic: load balancing and ingress

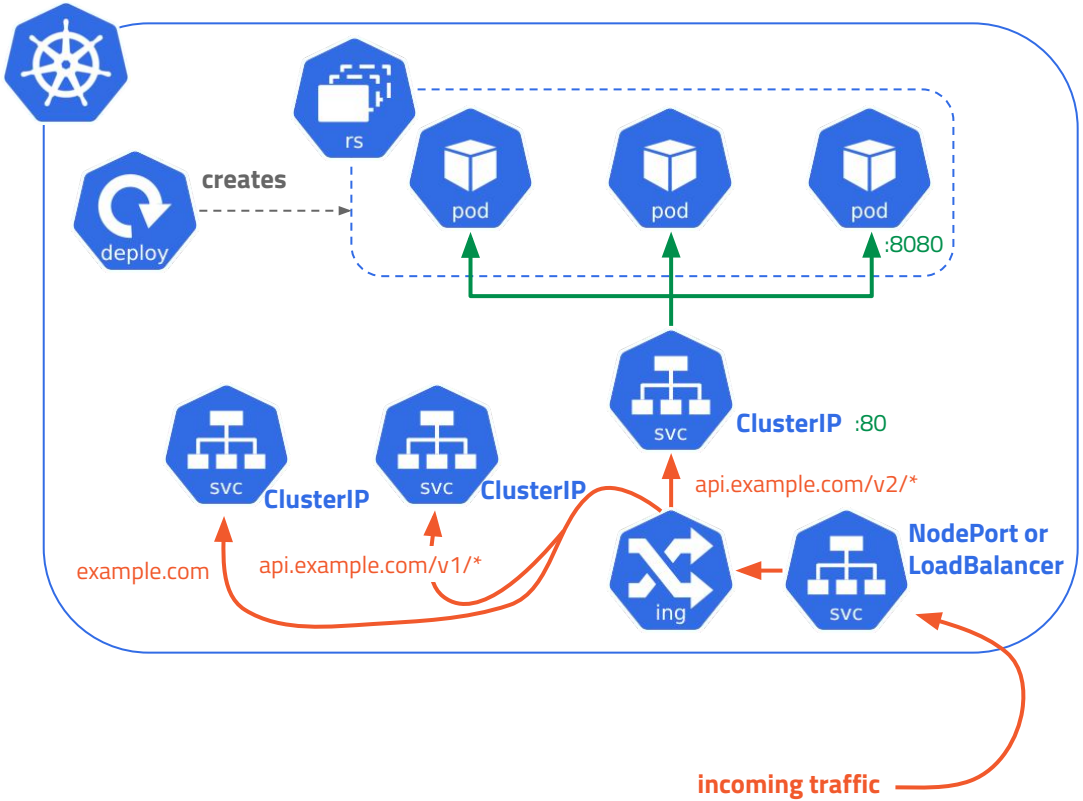
Option 3: Ingress



- Rule-based http routing
- Points to multiple internal services configured as ClusterIP
- Many ingress controller options (caddy, nginx, træfik, cloud provider specific, etc.)
- Ingress controller needs to be exposed as NodePort or LoadBalancer service (but only one!)

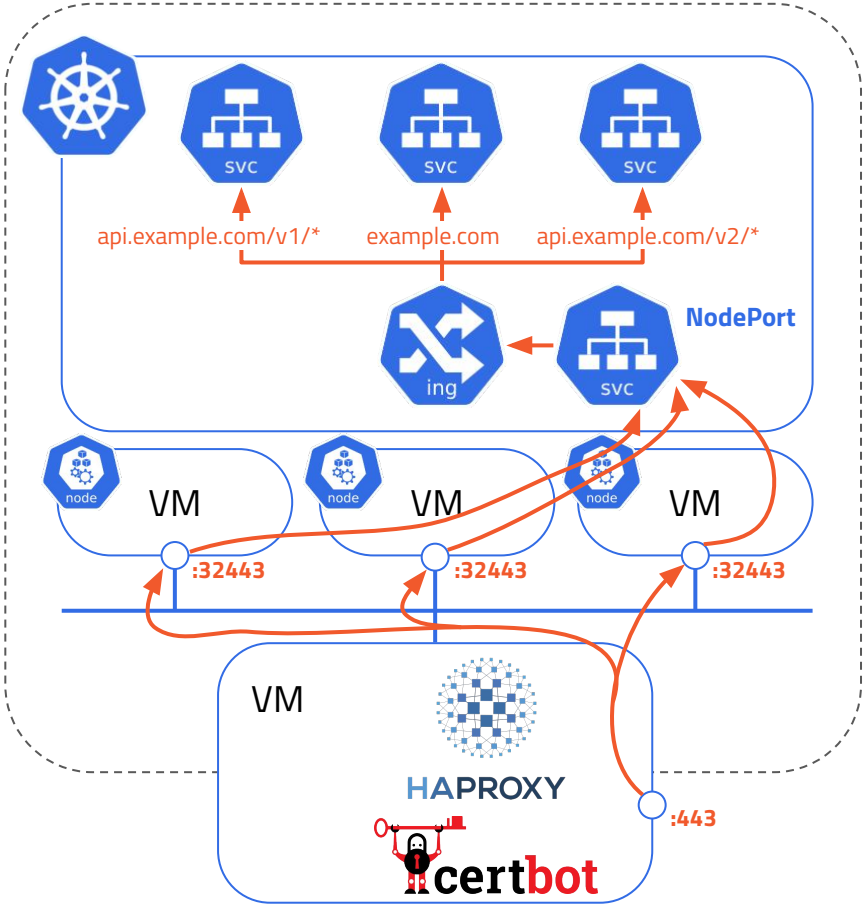
Incoming traffic: load balancing and ingress

Option 3: Ingress




```yaml
apiVersion: v1
kind: Service
metadata: {name: my-service}
spec:
 type: ClusterIP
 selector: {app.kubernetes.io/name: MyApp}
 ports: [{protocol: TCP, port: 80, targetPort: 8080}]
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: minimal-ingress
spec:
 ingressClassName: nginx
 rules:
 - host: api.example.com
   http:
     paths:
     - path: /v2/
       pathType: Prefix
       backend:
         service:
           name: my-service
           port:
             number: 80
```

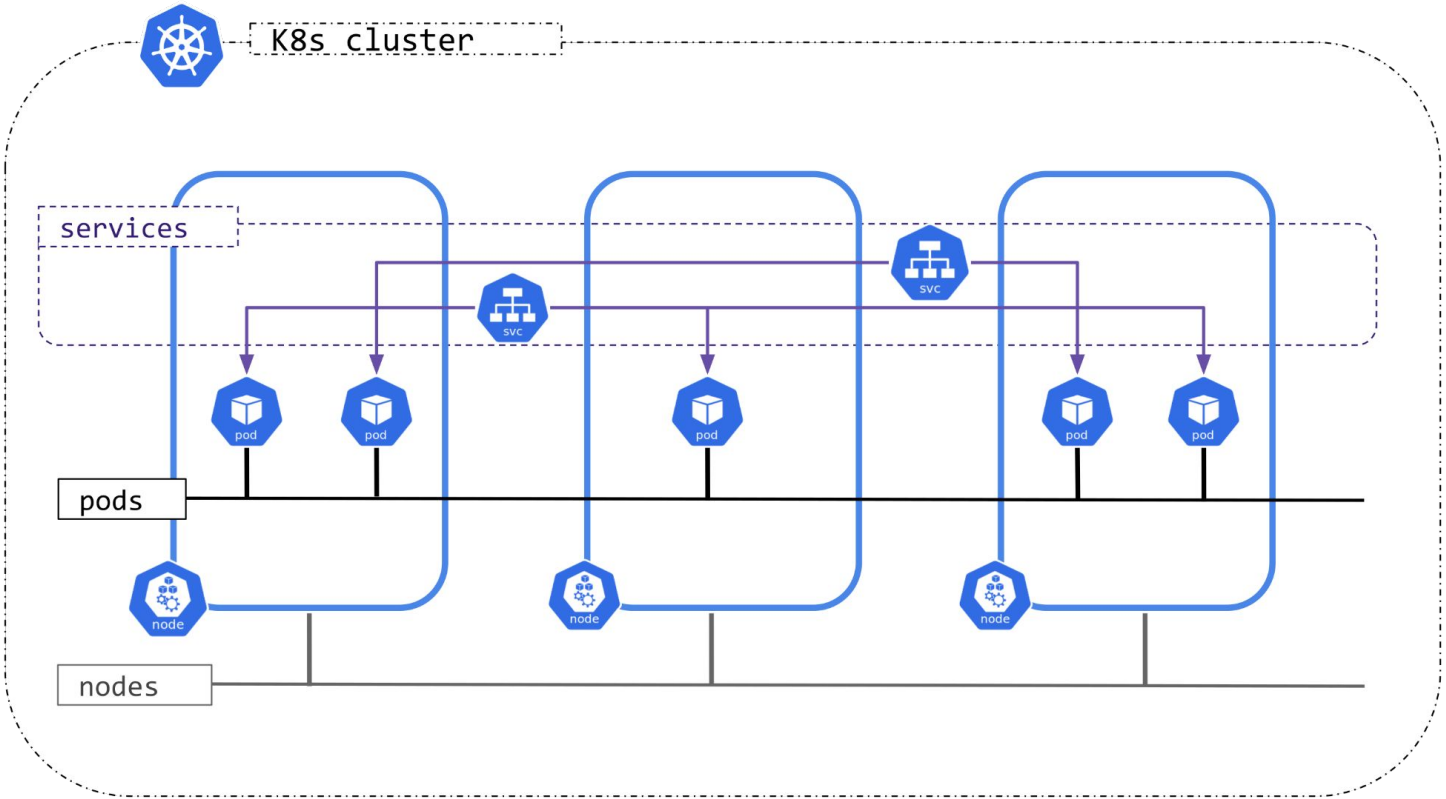
Incoming traffic: our self-managed load-balancer approach



- Standalone VM(s) running HAProxy
- Let's Encrypt certificates provisioned by certbot through HTTP01 challenge
- Round-robin load-balancing
- Trust self-signed SSL certs from k8s cluster

```
frontend https-in
  mode http
  bind :443 ssl crt /etc/ssl/private/
  option forwardfor
  use_backend k8s if { hdr(Host) -i example.com api.example.com }
  default_backend error

frontend http-in
  mode http
  bind :80
  acl acme_challenge path_beg /.well-known/acme-challenge/
  redirect scheme https if !{ ssl_fc } !acme_challenge
  use_backend certbot if acme_challenge

backend certbot
  server certbot certbot_letsencrypt:80 resolvers docker

backend k8s
  balance roundrobin
  server worker01 10.0.1.46:32443 check fall 3 rise 2 ssl ca-file /etc/ssl/backend-certs/k8s/
  server worker02 10.0.1.55:32443 check fall 3 rise 2 ssl ca-file /etc/ssl/backend-certs/k8s/
  server worker03 10.0.1.80:32443 check fall 3 rise 2 ssl ca-file /etc/ssl/backend-certs/k8s/
  ...
```

More about internal cluster networking

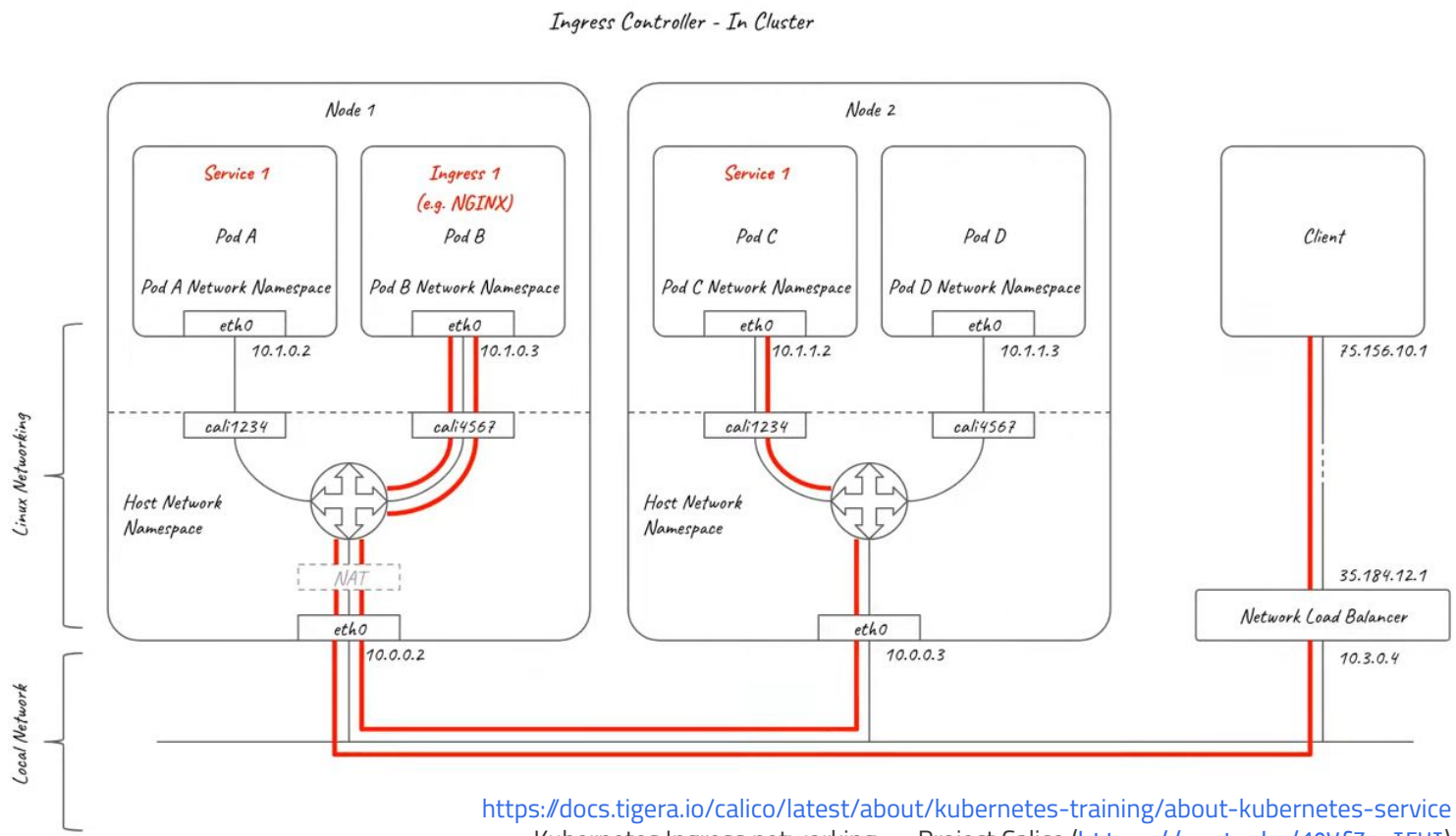


Virtualized overlay network

Implementation left to the container network interface (CNI)

https://kubernetes.io/docs/concepts/cluster-administration/networking/

More about internal cluster networking



Virtualized overlay network

Implementation left to the container network interface (CNI)

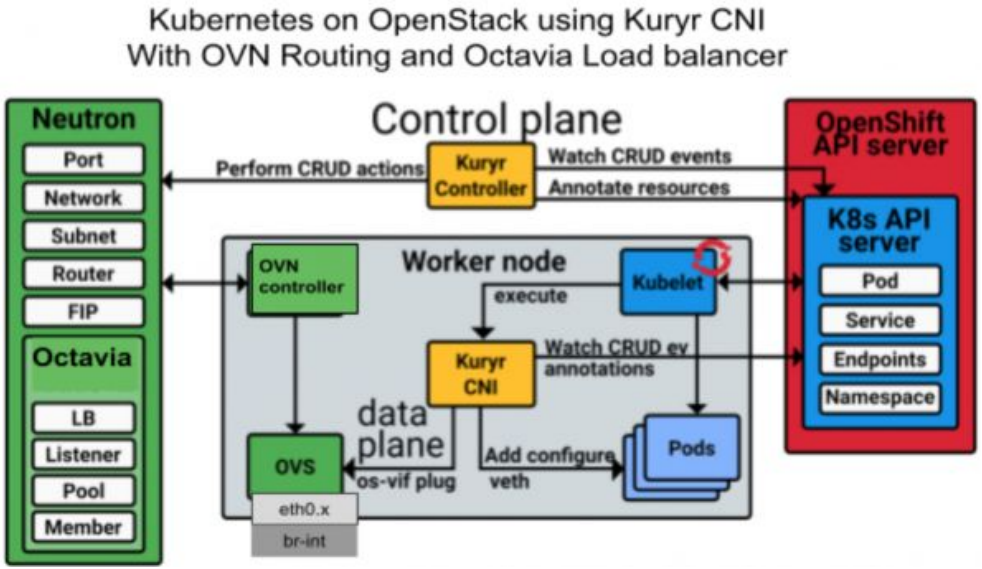
Examples:

- Calico on top of any VMs
- Kuryr on OpenStack avoids double overlay

https://docs.tigera.io/calico/latest/about/kubernetes-training/about-kubernetes-service

Kubernetes Ingress networking — Project Calico (https://youtu.be/40VfZ_nIFWI)

More about internal cluster networking



Virtualized overlay network

Implementation left to the container network interface (CNI)

Examples:

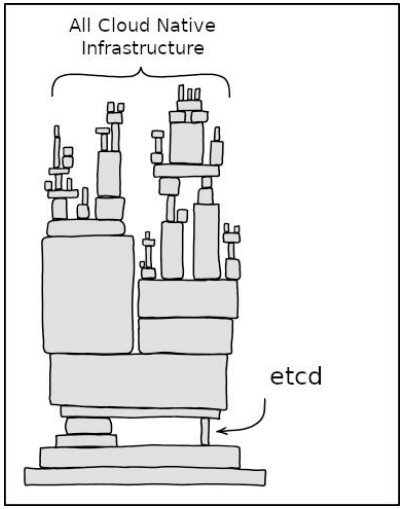
- Calico on top of any VMs
- Kuryr on OpenStack avoids double overlay

https://superuser.openinfra.dev/articles/run-your-kubernetes-cluster-on-openstack-in-production/

High Availability: in-cluster

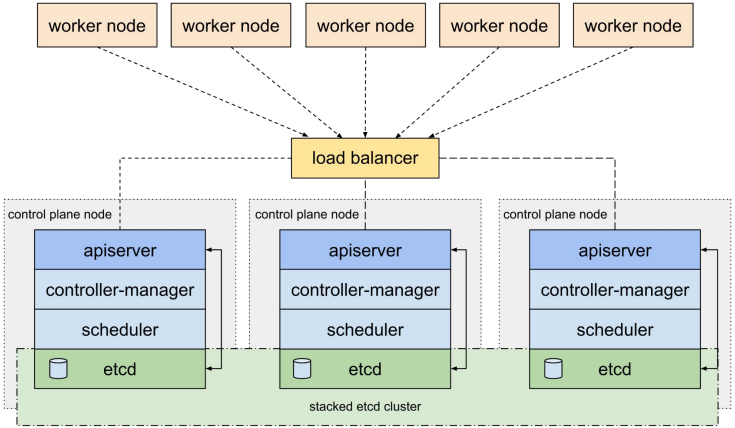
etcd: distributed key-value store (raft consensus)

- odd number of members
- tuning of heartbeat interval and election timeout depending on network and node characteristics (https://etcd.io/docs/latest/tuning)

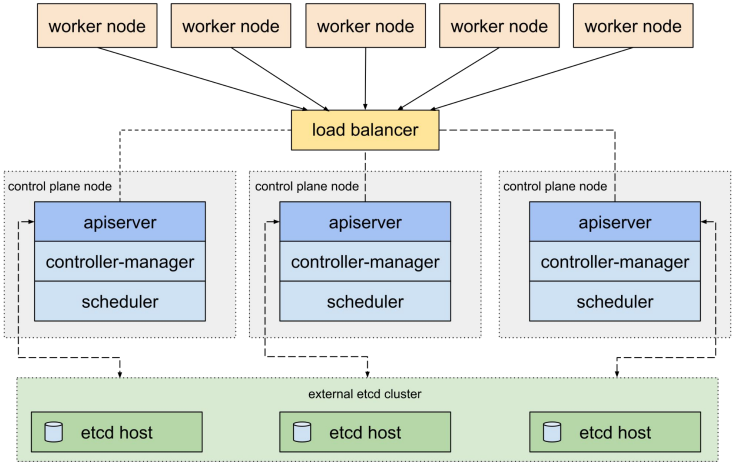


Cluster Size	Majority	Failure Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2

kubeadm HA topology - stacked etcd



kubeadm HA topology - external etcd

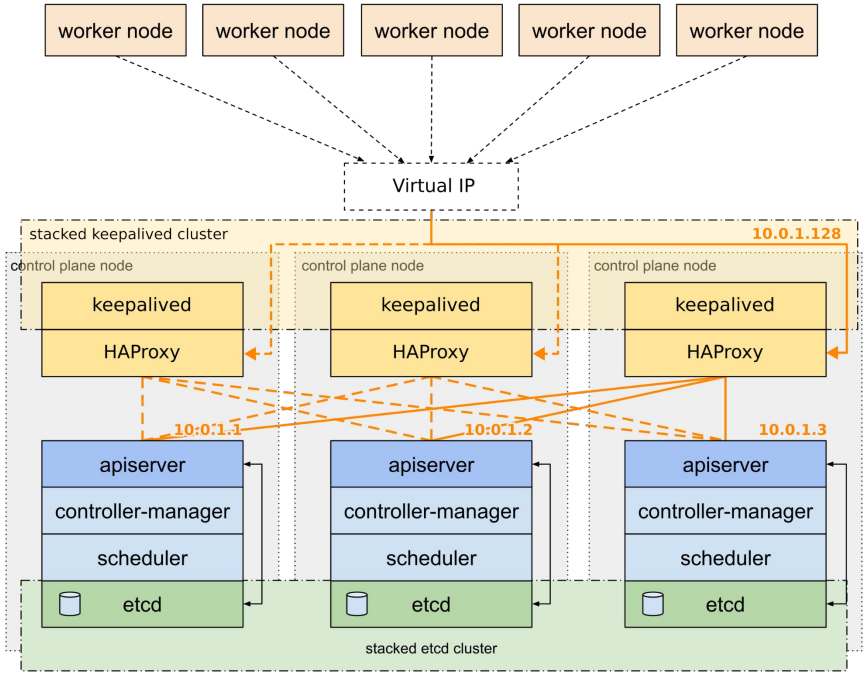


https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/

High Availability: in-cluster

Load balancer

- highly-available virtual IP (VRRP with keepalived)
- load-balanced kube API (HAProxy)




```
# /etc/haproxy/haproxy.cfg
# ...
frontend apiserver
  bind 10.0.1.128:7443
  mode tcp
  default_backend apiserver
backend apiserver
  option httpchk GET /healthz
  http-check expect status 200
  mode tcp
  option ssl-hello-chk
  balance roundrobin
  server master01 10.0.1.92:7443 check
  server master02 10.0.1.27:7443 check
  server master03 10.0.1.35:7443 check
```

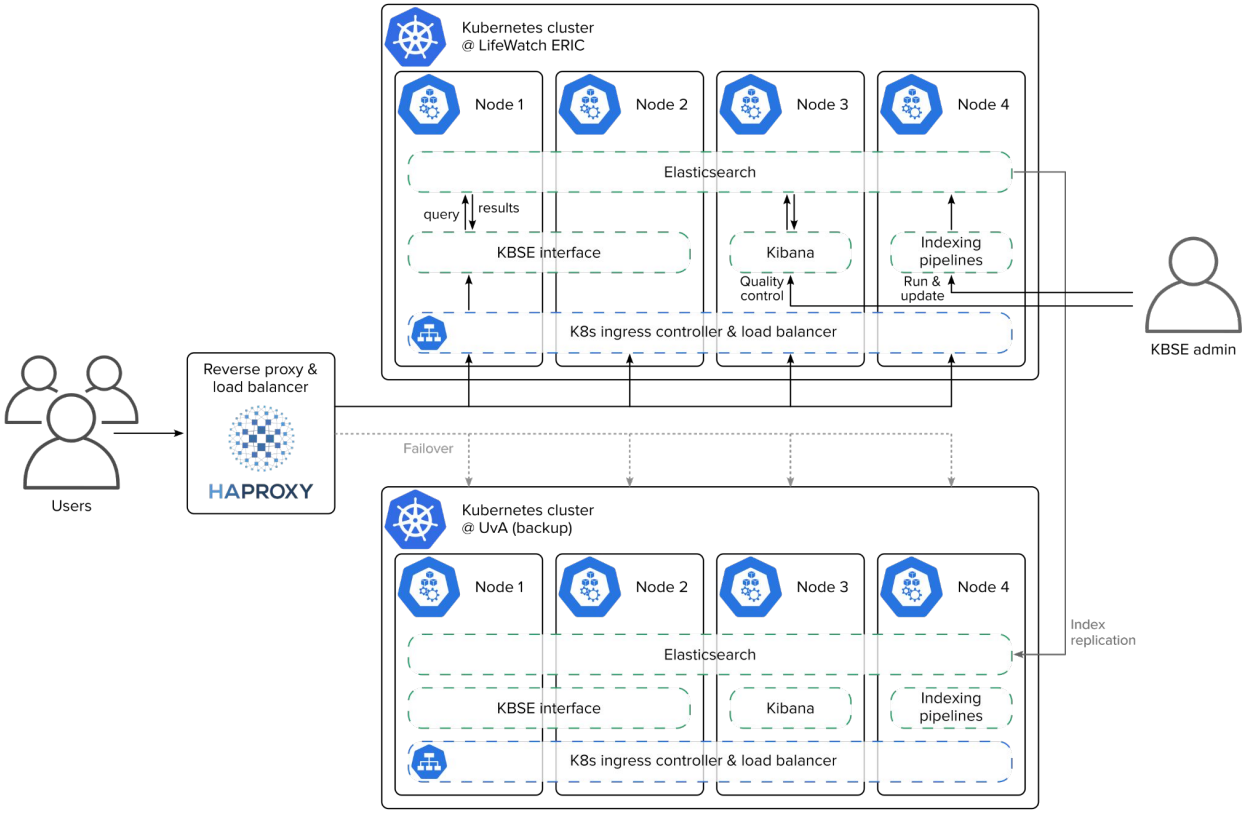
```
! /etc/keepalived/keepalived.conf
global_defs {
  router_id LVS_DEVEL
}
vrrp_script check_apiserver {
  script "/etc/keepalived/check_apiserver.sh"
  interval 3
  weight -2
  fall 10
  rise 2
}
vrrp_instance VI_1 {
  state MASTER
  interface eth0
  virtual_router_id 51
  priority 101
  authentication {
    auth_type PASS
    auth_pass QXJlIHlvdSBhIGhhY2tlcj8gOk8KCg==
  }
  virtual_ipaddress {
    10.0.1.128
  }
  track_script {
    check_apiserver
  }
}
```

```
#!/bin
# /etc/keepalived/check_apiserver.sh
errorExit() {
  echo "*** $*" 1>&2
  exit 1
}
checkLink() {
  curl --silent --max-time 2 --insecure "$*" -o /dev/null || errorExit "Error GET $*"
}
checkLink "https://localhost:7443/"
if ip addr | grep -q "10.0.1.128"; then
  checkLink "https://10.0.1.128:7443/"
fi
```

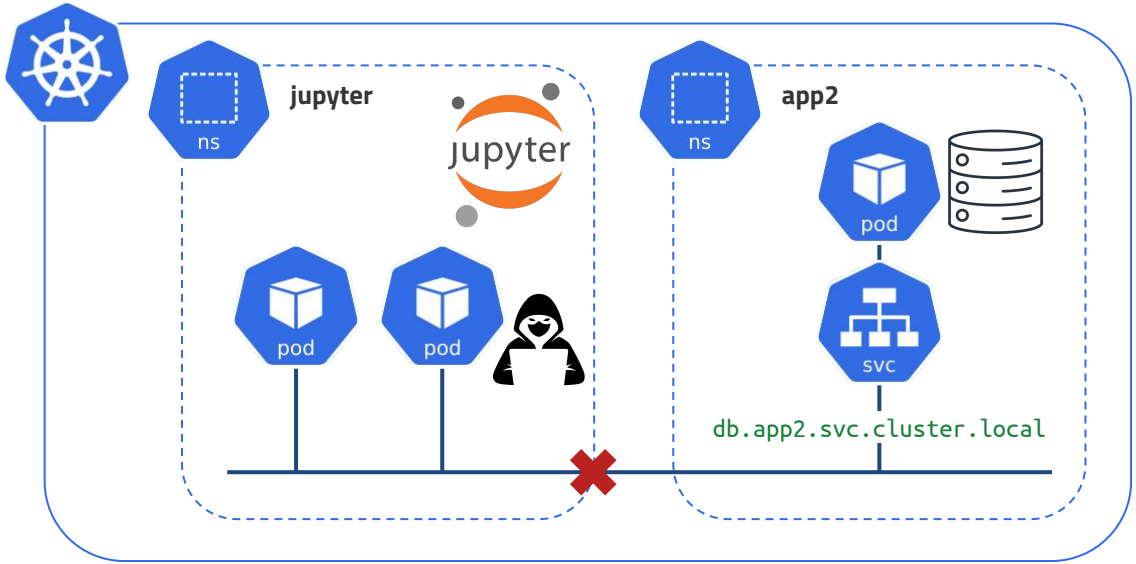
High Availability: multi-cluster

- Load balancer configured to fail over to another k8s cluster
- Load balancer can be highly-available within same LAN
- Failover to a different site
  - Floating IP (in multi-zone cloud)
  - DNS failover (value is debated)
  - Anycast

```
# /etc/haproxy/haproxy.cfg
frontend ft_app
 bind 10.0.0.1:80
 default_backend bk_app_main

backend bk_app_main
 option allbackups
 server s1 10.0.0.101:80 check
 server s2 10.0.0.102:80 check
 server s3 10.0.0.103:80 check backup
 server s4 10.0.0.104:80 check backup
```



Network isolation: how to prevent devs to do dumb things




```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: namespace-isolation
  namespace: jupyter
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: jupyter
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
```

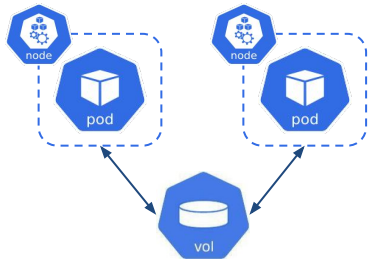
K8s network policies

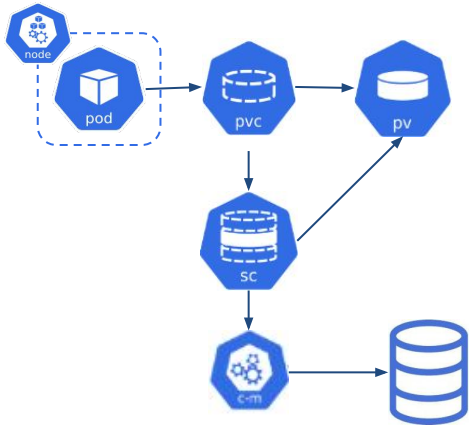
- enforced by the CNI
- not all CNIs enforce them…
  → *migrated from Flannel to Calico recently*

https://kubernetes.io/docs/concepts/services-networking/network-policies/

Storage

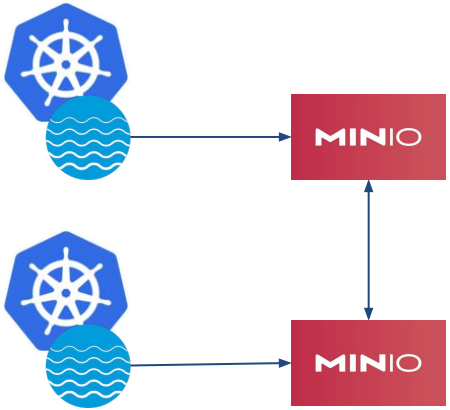
- Pods are the smallest deployable units of computing that you can create and manage in Kubernetes and can contain one or more containers
- Pods are ephemeral (rather than durable): When restarting a pod all changed are gone
- **Volumes**: Independent from pod lifecycle
- On restart which node will the pod land on? Volumes must be available on all nodes
- What if cluster crashes?
- Where is the physical storage?
- How does the storage class affect the application performance, fault tolerance?





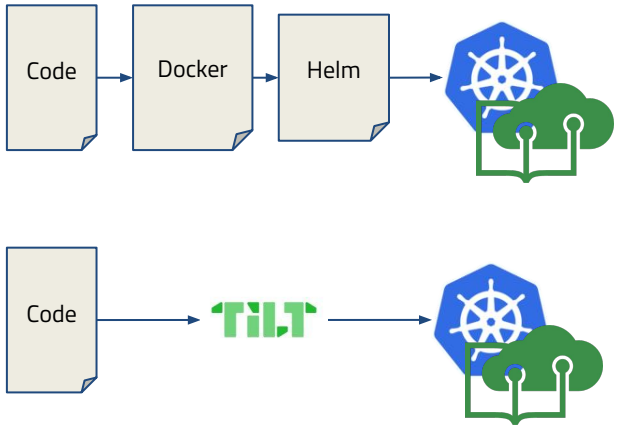
Backup

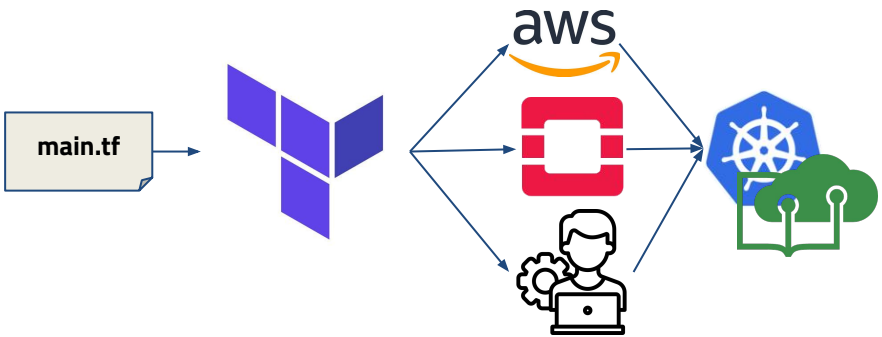
- How do we recover from a disaster ?
- We run over 64 services in a k8s cluster
- Each service has multiple ConfigMaps, Secrets, PVs, etc.
- Setup an automated backup and recovery with velero
- Physical data needs to be outside the cluster
- Deployed 2 MinIO services on separate VMs
- Each MinIO service replicates the other
- Open issues:
    a. New deployment need to be manually configured for backups
    b. Velero backs up PVCs attached to pods. In jupyterHub pods are removed after user logs in. How do we backup home dir?
    c. What happens if we lose the physical disks?
    d. Users can use enormous amounts of storage. How do we control increasing size in home directories?

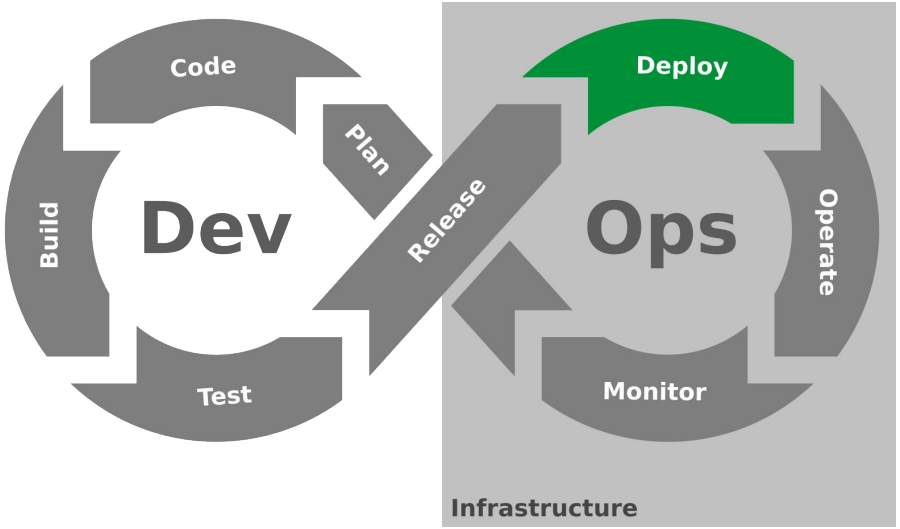


Infrastructure as Code (IaC)

- Provisioning and managing infrastructure (compute, data, network resources) through machine-readable definition files
- How can we run integration tests in an identical environment as the production?
- Imperative vs declarative: How to deploy the infrastructure vs what the infrastructure should look like

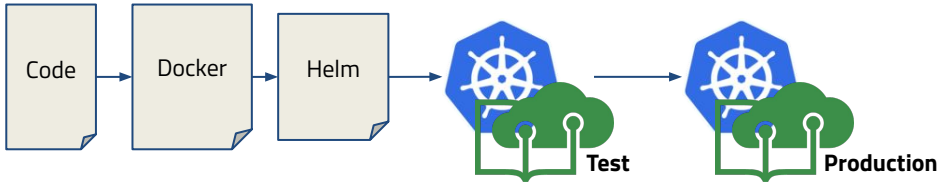




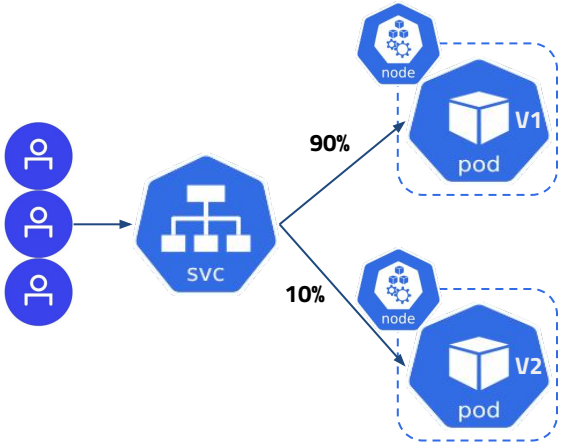

Code
Plan
Build
Dev
Release
Test
Deploy
Ops
Operate
Monitor
Infrastructure

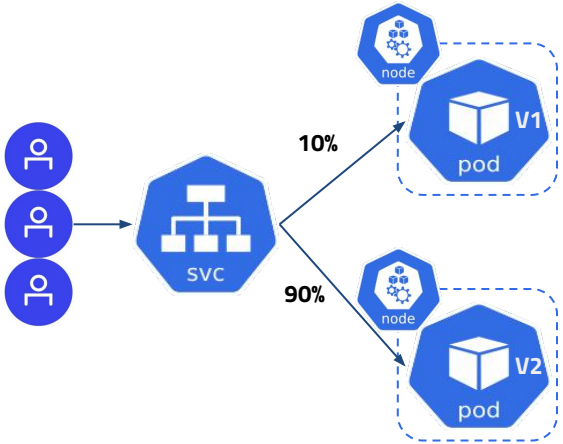
Deploy

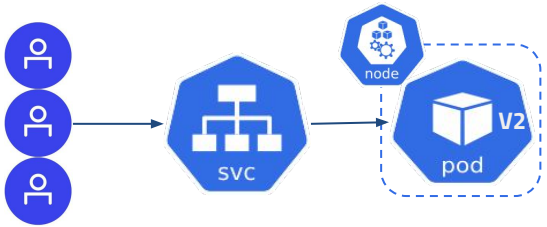
Deploy

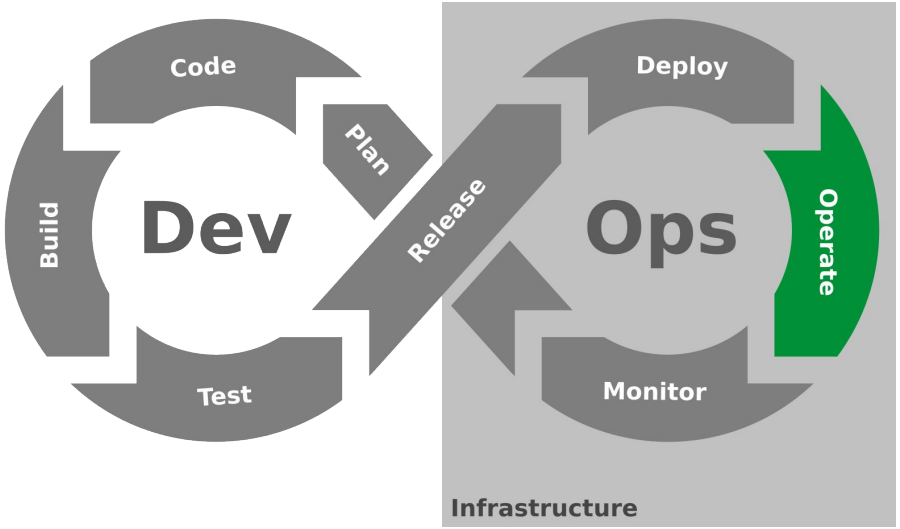



```
helm -n vl-laserfarm upgrade --install --create-namespace vl-laserfarm jupyterhub/jupyterhub --version v3.3.7 -f k8s/n-a-a-vre/vl-laserfarm.yaml
helm -n vl-openlab upgrade --install --create-namespace vl-openlab jupyterhub/jupyterhub --version v3.3.7 -f k8s/n-a-a-vre/vl-openlab.yaml
helm -n vl-phytoplankton upgrade --install --create-namespace vl-phytoplankton jupyterhub/jupyterhub --version v3.3.7 -f k8s/n-a-a-vre/vl-phytoplankton.yaml
helm -n vl-veluwe-proto-dt upgrade --install --create-namespace vl-veluwe-proto-dt jupyterhub/jupyterhub --version v3.3.7 -f k8s/n-a-a-vre/vl-veluwe-proto-dt.yaml
helm -n vl-vol2bird upgrade --install --create-namespace vl-vol2bird jupyterhub/jupyterhub --version v3.3.7 -f k8s/n-a-a-vre/vl-vol2bird.yaml
helm -n vl-waddenzee-proto-dt upgrade --install --create-namespace vl-waddenzee-proto-dt jupyterhub/jupyterhub --version v3.3.7 -f\
k8s/n-a-a-vre/vl-waddenzee-proto-dt.yaml
```





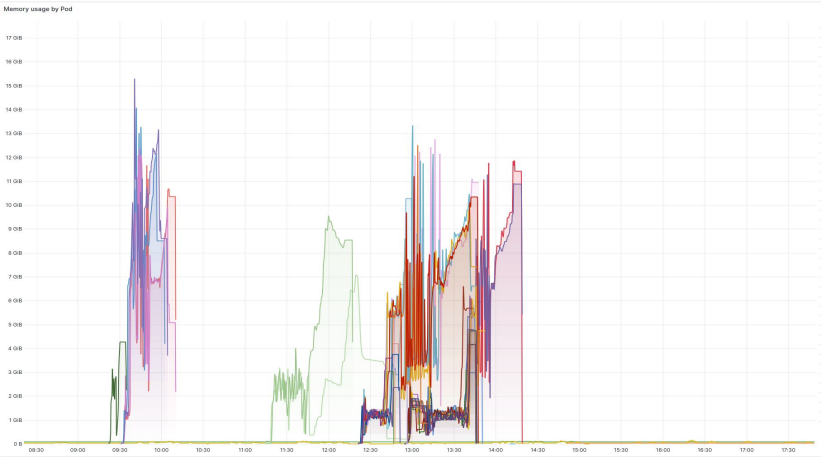


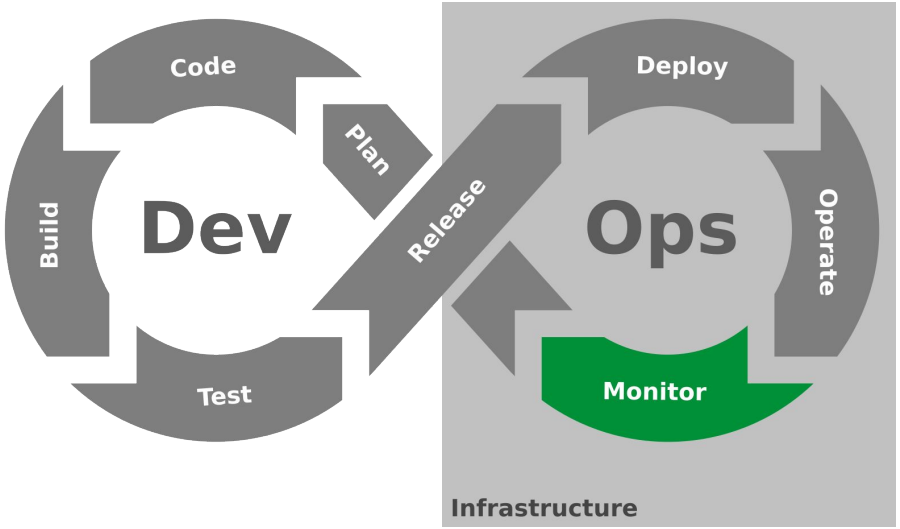

Code
Plan
Build
Dev
Release
Test
Deploy
Operate
Ops
Monitor
Infrastructure

Operate

Operate

- We co-develop virtual labs
- Small initial user base (4-5 users)
- We expect to attract more
- Workflows use large datasets (~ 1TB)
- Spike demand
- Bi-weekly meetings: Users report bugs, request features
- How can we transparently scale VMs resources?
- To speed up workflow execution we need fast storage mounts: Plan to get 9TB of Non-Volatile Memory Express (NVMe) storage
- Tag VMs for the k8s scheduler
- Pod (anti-)affinity
- Resources utilization vs performance

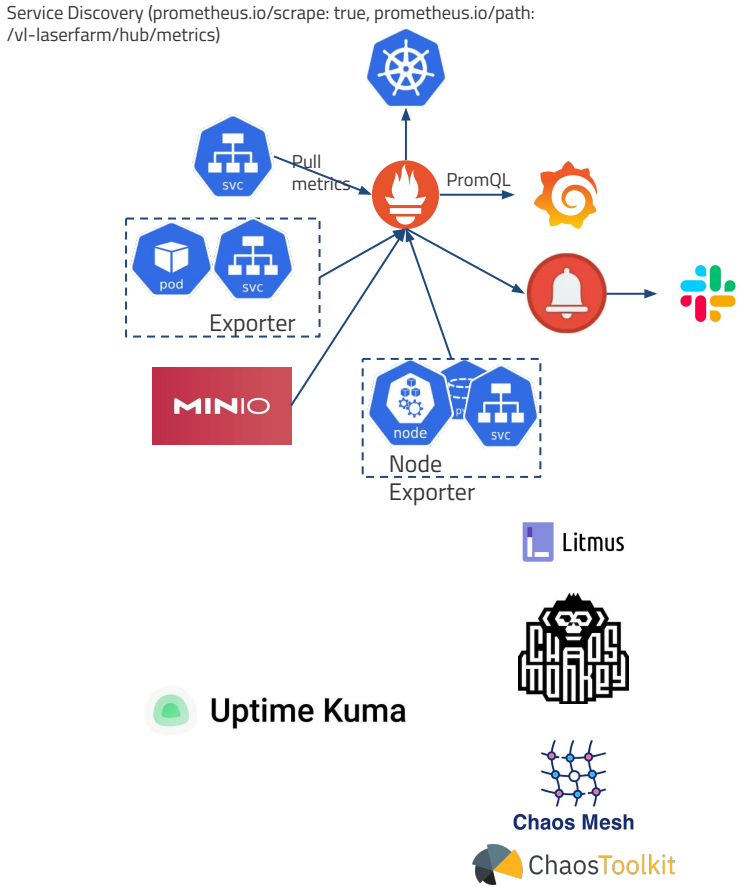



Code
Plan
Build
Dev
Release
Test
Deploy
Ops
Operate
Monitor
Infrastructure

Monitor

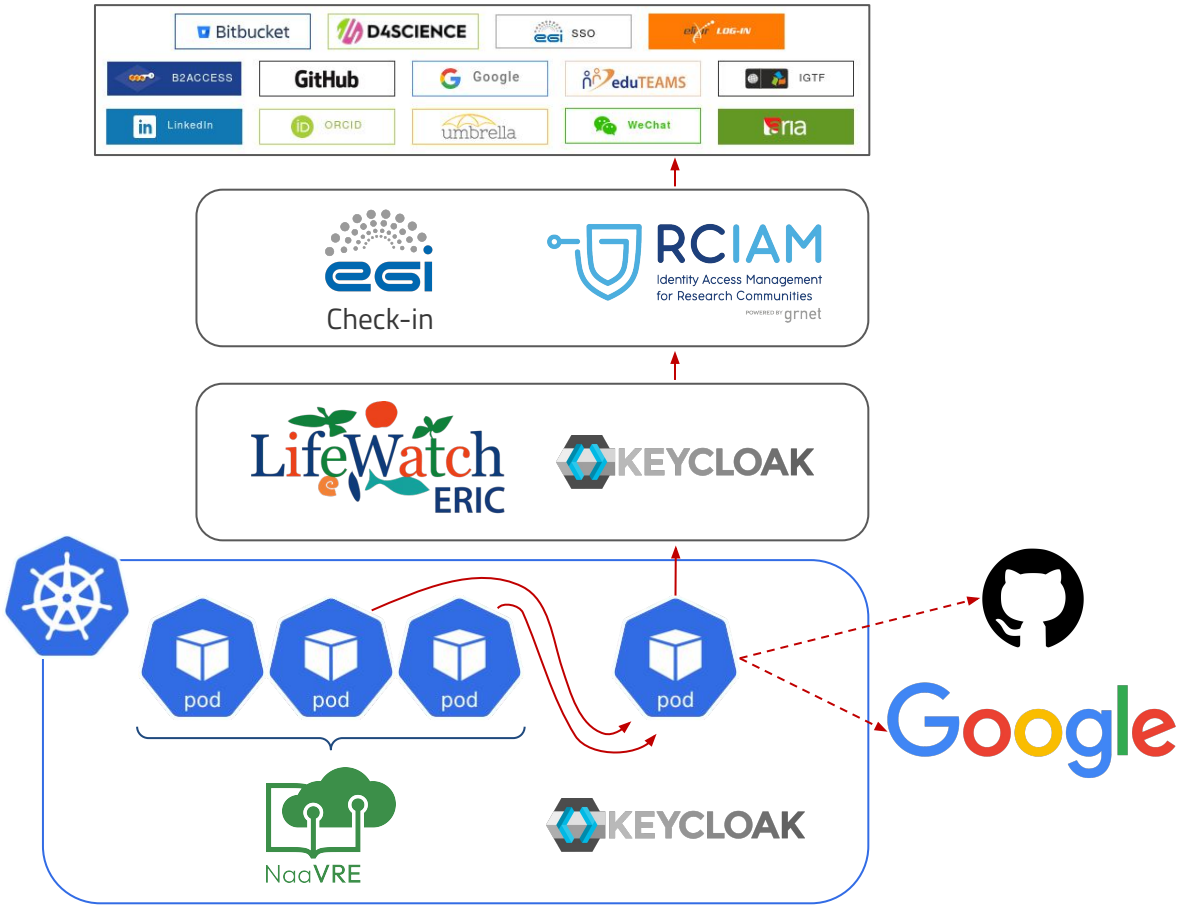
Monitor

- Are the services running ?
- Deploy and configure Kube-Prometheus-stack
- Challenges: Few users with irregular usage patterns
- Failures go unnoticed for a long time, but need to be fixed
- Run tests on online services and set up alerts
- Chaos engineering:
  - Definition of the steady state: How should behave?
  - Hypothesis: Will steady state persist?
  - Real-life variables: Variables that represent real-live events, i.e. traffic surge
  - Disprove the hypothesis: Try to break the system



Integration with external services

SSO and identity federation

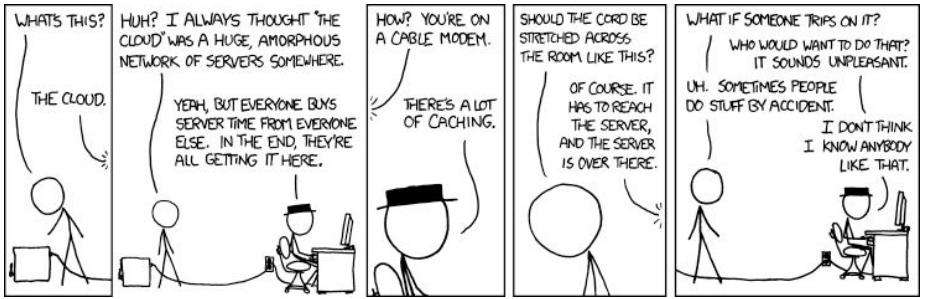


- **Keycloak:** Local SSO for all components of the NaaVRE app
- Federated with the LifeWatch Research Infrastructure (other teams + national research communities) → **OpenID Connect**
- Federated with EGI Check-in (academic and social accounts)

Summary and takeaways

Summary and takeaways

- We use DevOps to deliver a Virtual Research Environment PaaS to our users
- We run on heterogeneous cloud & non-cloud infrastructure
- Kubernetes allows us to have a uniform platform
- Many things need to be considered when running your own k8s clusters
- DevOps pipelines are incrementally built as our system evolves to meet new challenges



https://xkcd.com/908/