Kubernetes as CfgMgmt-Tool

Andy Wirtz

3th February 2020

Andy Wirtz

Andy:

- IT Consultant at ATIX AG, Germany
- Automation of data centers
- Deployment of cloud native services
- Expertise in Docker, Kubernetes, Istio

Contact:

- Phone: +49 (0)89 452 35 38-248
- Mail: wirtz@atix.de
- www.xing.com/profile/Andy_Wirtz2
- www.linkedin.com/in/andy-wirtz

Agenda

1. Kubernetes
2. CfgMgmt
3. GitOps

Agenda

1 Kubernetes

2 CfgMgmt

3 GitOps

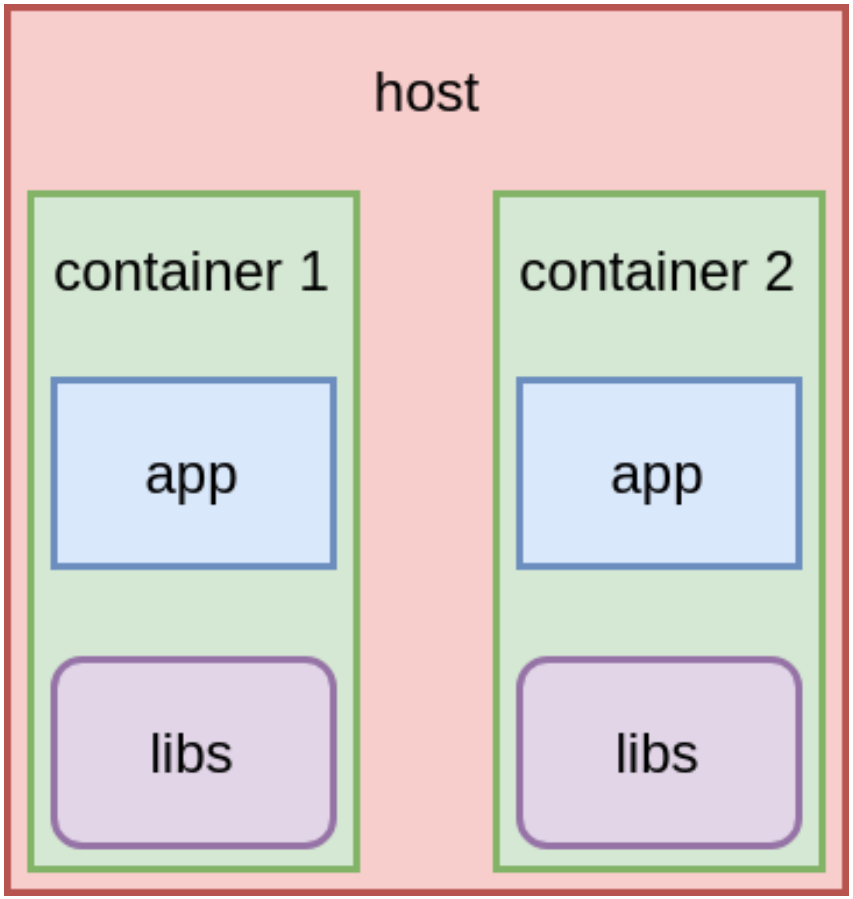
Microservices & Containers

Microservices:

- one process per application
- communication via APIs
- new features faster/easier

Containers:

- application plus libraries
- isolation
- independence



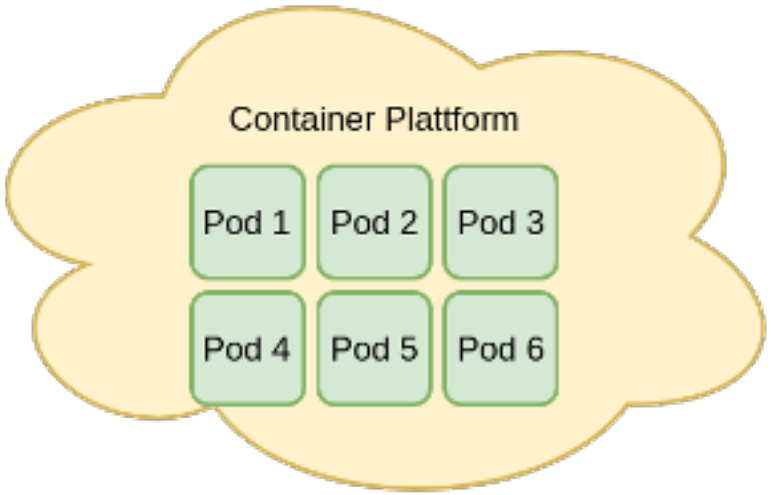
Kubernetes

Container platform:

- hardware abstraction
- cloud for containers
- efficient resource management

Automation:

- scaling based on requests
- self-healing
- updates without downtime



Cloud Native Ecosystem

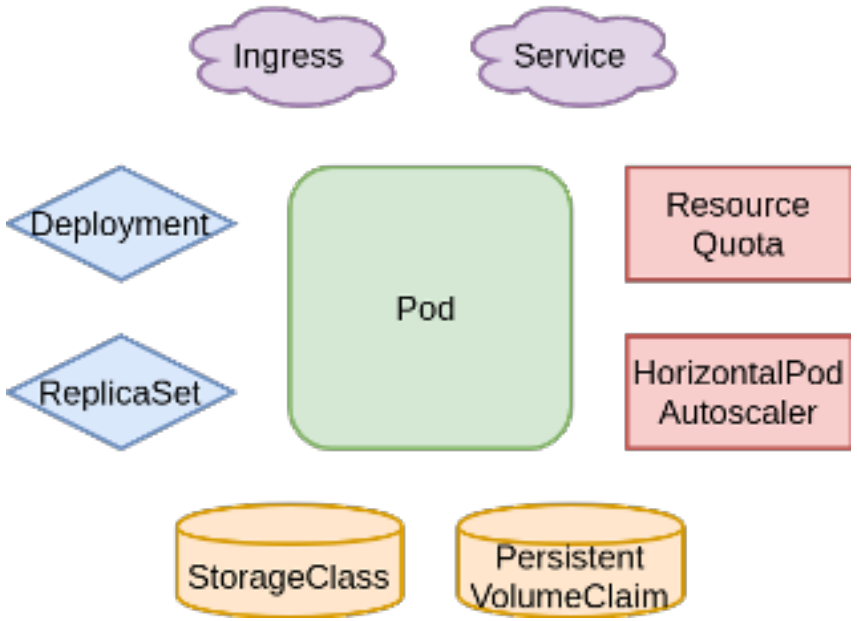
Cluster configurations:

- service mesh
- persistent storage
- central logging
- central monitoring
- CI/CD
- backup/recovery

Complexity

Kubernetes feature set:

- deployent of workloads
- service discovery & LB
- configs & storage
- resource limits
- security
- custom extensions



Agenda

1 Kubernetes

2 CfgMgmt

3 GitOps

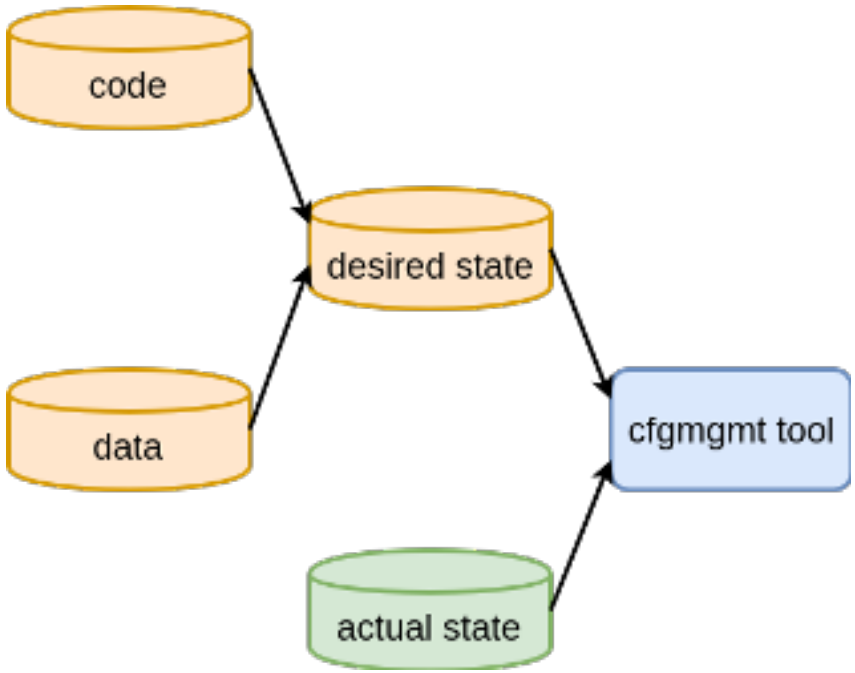
CfgMgmt

Declarative model:

- define the desired state
- make use of idempotence
- tool does necessary steps

Automation:

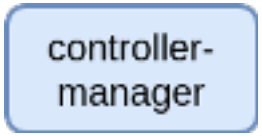
- standardization
- reproducibility
- essential in highly dynamic IT



Controller-Manager

Active component:

- single binary/process
- logically separate processes
- runs controller processes



CfgMgmt-Tool:

- declarative model
- compares current/desired state
- acts if need be

Other components

Scheduler:

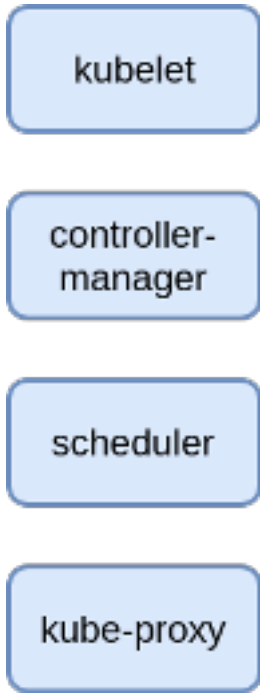
- determines node to run pod

Kubelet:

- starts/stops pods

Kube-proxy:

- manipulates ip-tables



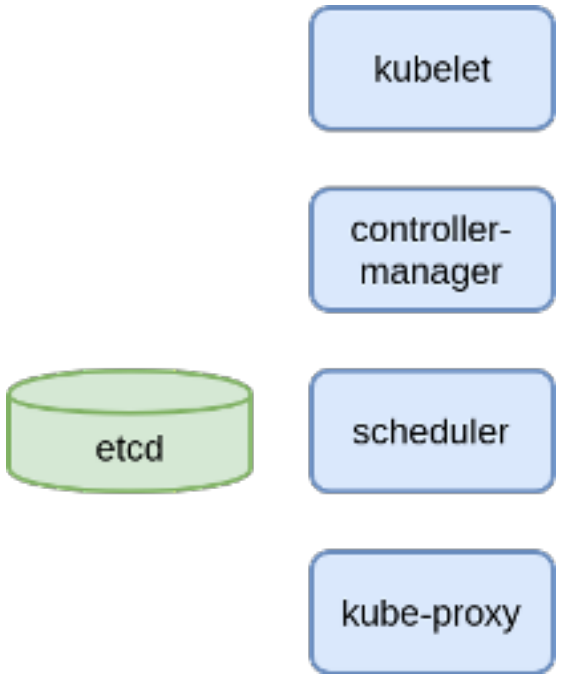
Etcd

Database:

- consistent and highly-available
- key-value store
- for all cluster data

Source:

- single source of truth
- for the current state
- for the desired state



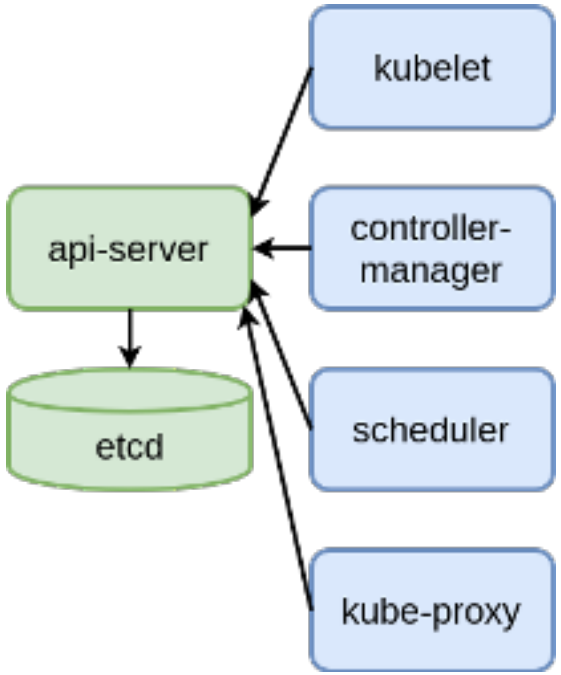
API Server

API resources:

- foundation for declarative configuration
- over 50 API resources
- easy and well-defined

API server:

- exposes API
- front end of control plane
- for all communication



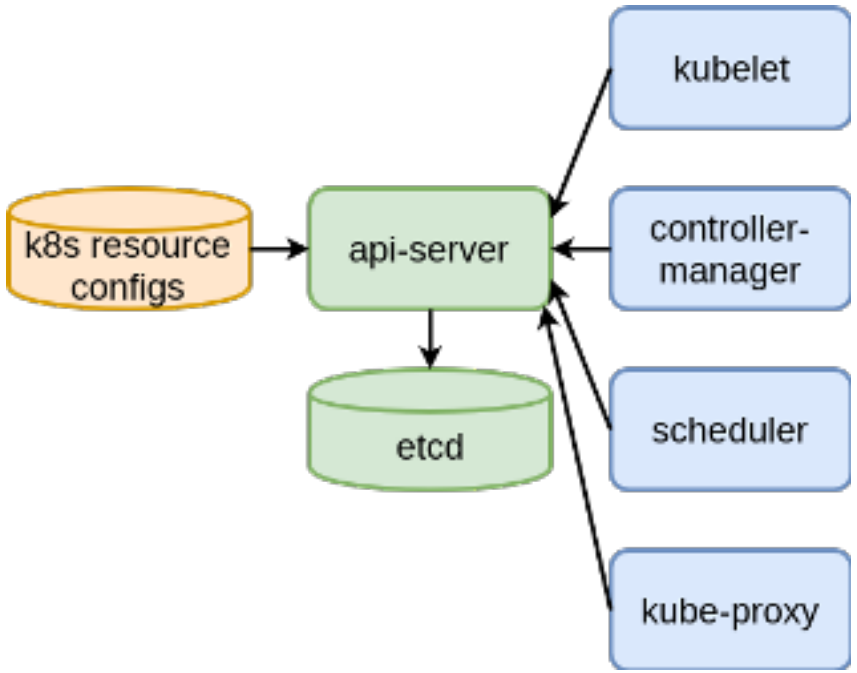
Kubectl client

Command-line interface:

- our client tool
- for controlling the cluster
- imperative/declarative commands

Work on API objects:

- create/update
- delete
- get/list/watch



Example

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: atixapp-deploy
spec:
  replicas: 3
  minReadySeconds: 10
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    ...
    spec:
      containers:
      - image: atixreg/atixapp:v1
        name: atixapp
        readinessProbe:
          periodSeconds: 1
          httpGet:
            path: /
            port: 8080
```

Bad practices I

Fully declarative command:

```
kubectl apply -f atixapp-deploy.yaml
```

Other commands:

```
kubectl create -f atixapp-deploy.yaml

kubectl replace -f atixapp-deploy.yaml

kubectl edit deployment atixapp-deploy

kubectl patch deployment atixapp-deploy \
  -p '{"spec":{"minReadySeconds":10}}'

kubectl set image deployment atixapp-deploy \
  atixapp=atixreg/atixapp:v2
```

Bad practices II

Combining imparative configuration schemes:

```
---
- name: Patch atixapp deployment
  command: kubectl patch deployment atixapp-deploy \
    -p '{"spec":{"minReadySeconds":10}}'
```

Bad practices III

Version control:

```
.
|-- installation_191203
|   |-- atixapp-deploy.yaml
|-- installation_200103
|   |-- atixapp-deploy.yaml
|-- installation_200203
    |-- atixapp-deploy.yaml
```

Agenda

1 Kubernetes

2 CfgMgmt

3 GitOps

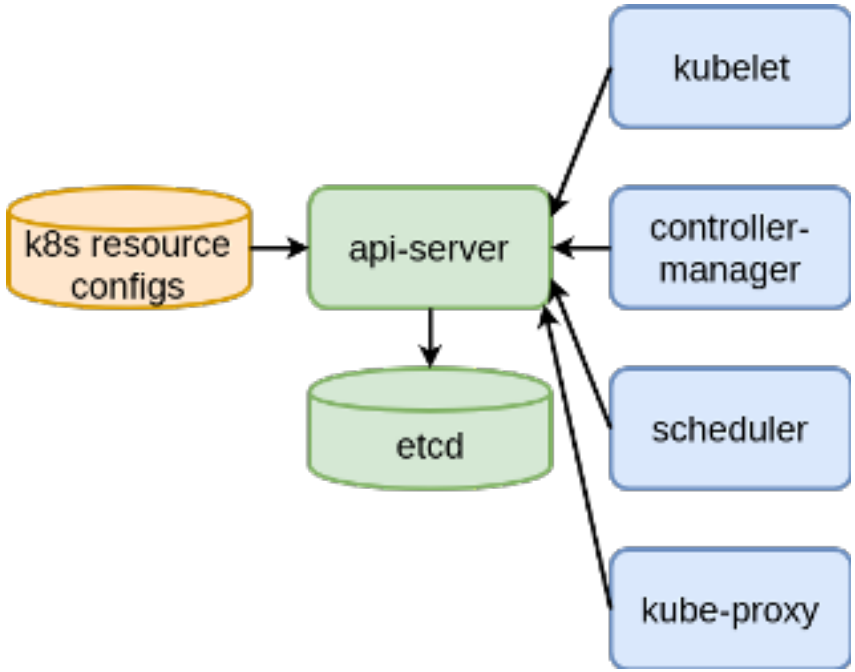
Git

Git:

- distr. version control system
- for kubernetes resource configs
- git flow for efficient branching

Objectives:

- handle multiple versions
- handle multiple environments
- work together



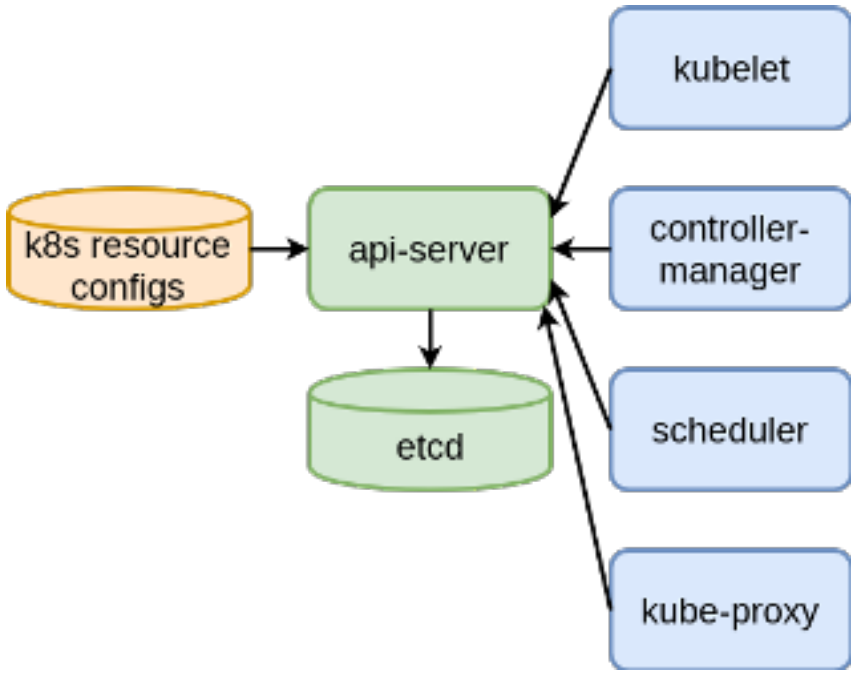
GitOps

Principles:

- desc. entire system declaratively
- version desired state in Git
- automatic apply approved changes

Monitoring:

- ensure correctness
- alert on divergence
- use software agents



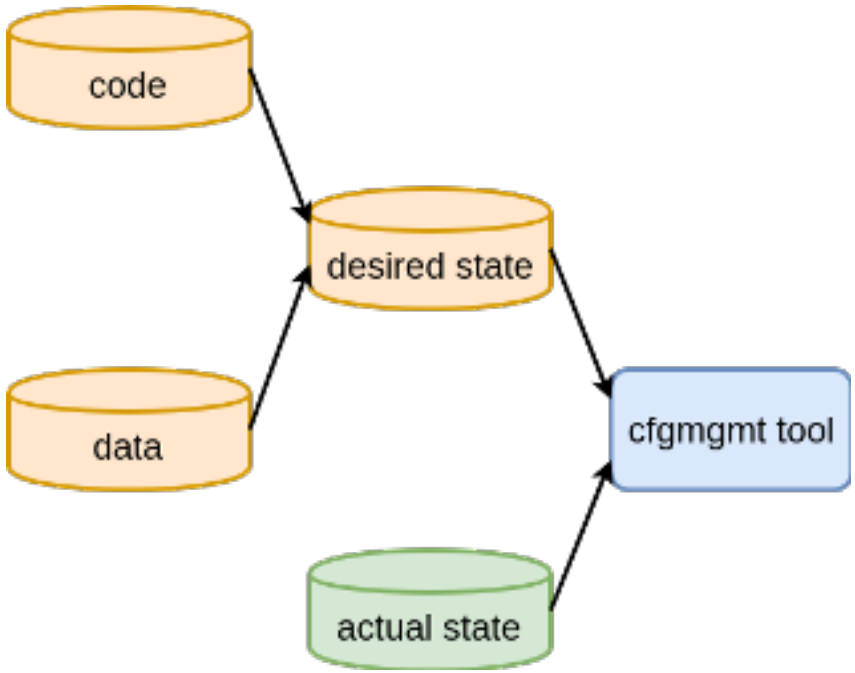
Infrastructure as Code

Infrastructure as Code:

- declarative model
- define desired state
- tool acts automatically

Administration via code:

- use version control repositories
- make use of idempotence
- seperate code and data



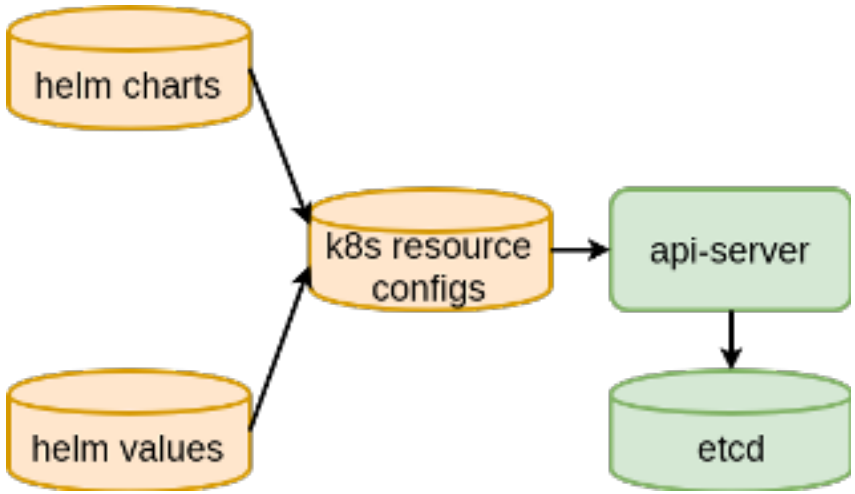
Helm

Package manager:

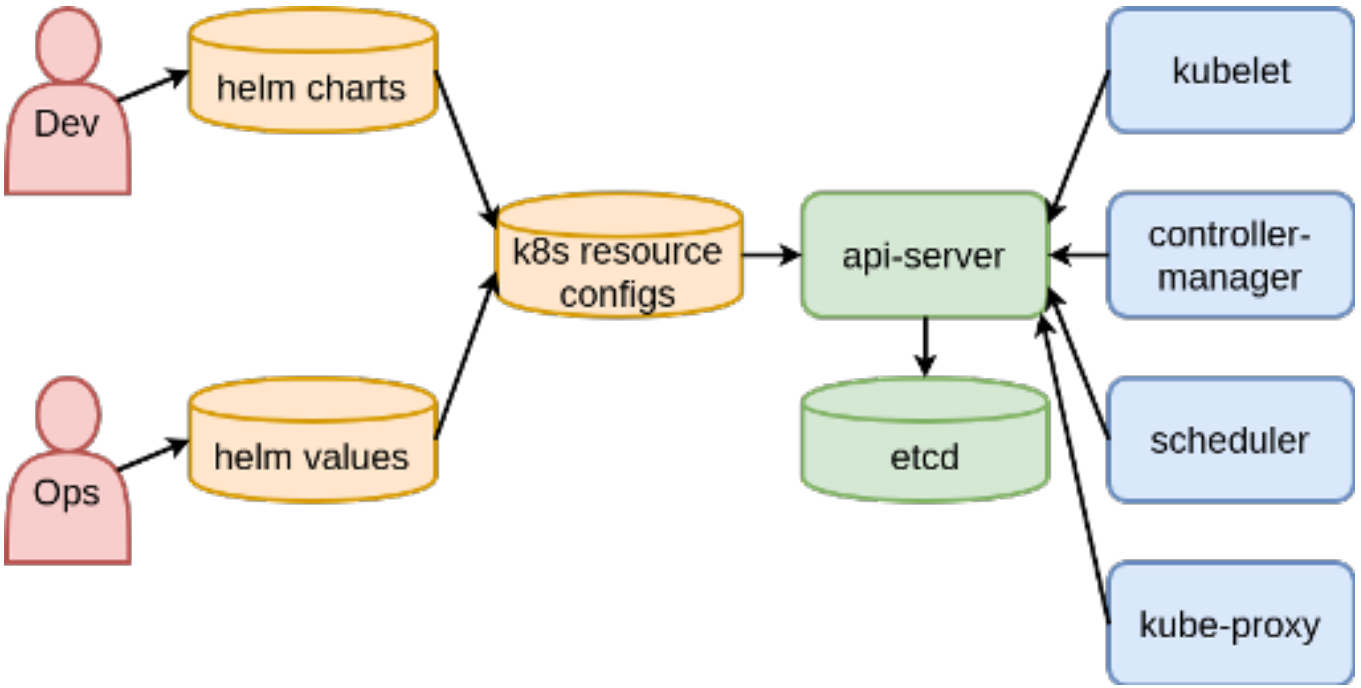
- templates for resource configs
- default values for paramater
- environment specific values

Separation:

- Helm chart repo
- Helm values repo
- render k8s resource configs



Workflow


Dev
helm charts
Ops
helm values
k8s resource configs
api-server
etcd
kubelet
controller-manager
scheduler
kube-proxy

Summary

Configure:

- your kubernetes platform
- the namespaces for tenants
- security policies

Deploy:

- your application stack
- infrastructure components
- cluster plugins

Outlook

Easy steps:

- render k8s resource configs
- commit/push configs to git
- use git hook to apply configs

Automation:

- push based: ansible, gitlab
- pull based: argo, flux