

Hi! :)

# Who am I you might ask

- David Hondl
- **Cloud Engineer @ XXXLdigital**
- Things I **care** about
  - Coding / DevOps practises / CNCF
  - Music and mountains
  - All the standard stuff



# Kubernetes as a universal **controlplane**

An Introduction into Crossplane

What and why is

# Kubernetes

kubect!

containers!

YAML!



cloud hosted

CNCF

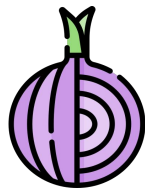
deploy apps

service mesh

so scalable

# The main use case

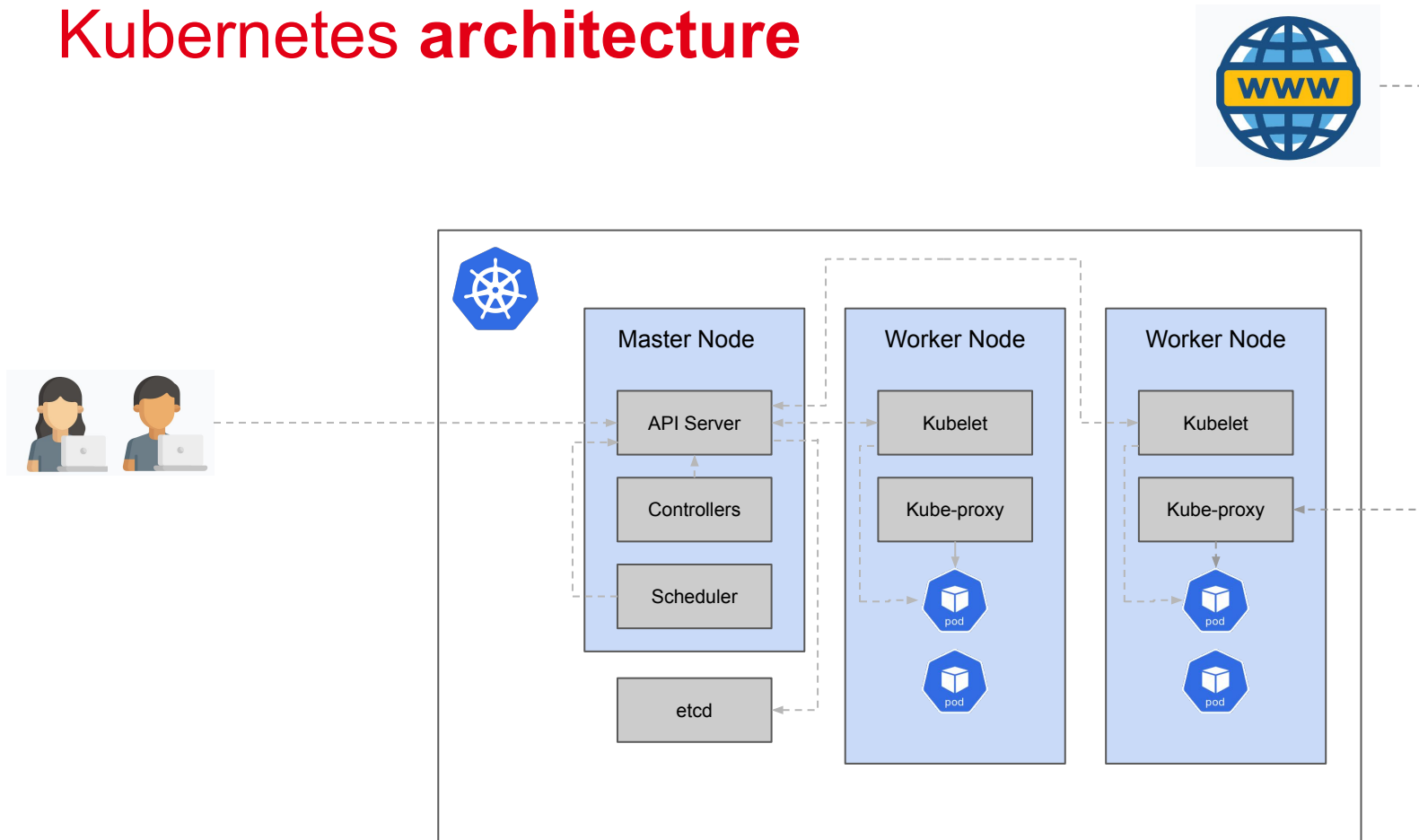
- Schedules and manages **containers** at scale
- Auto-**heal** and **scale**
- **Expose** deployment
- **Desired state** is applied
- `kubectl`



But it is **MORE** than that

Interesting K8s patterns I want to talk about

# Kubernetes architecture







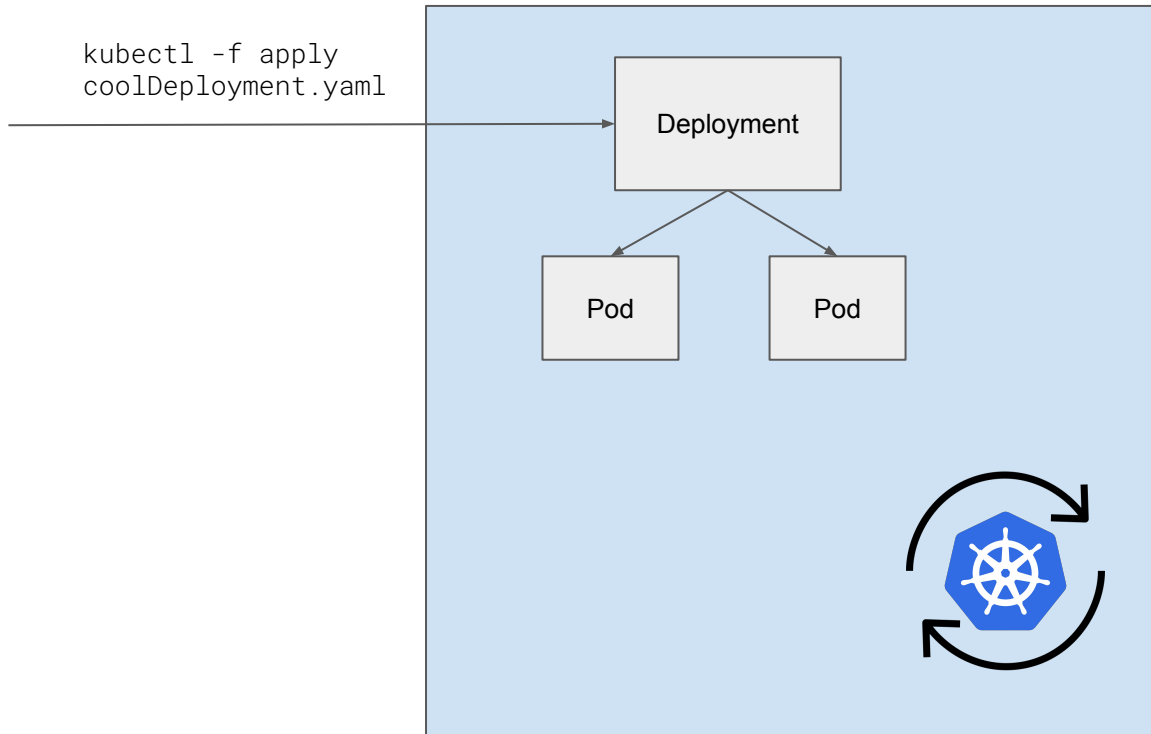
# Resources and Reconciliation

- Can be **modified** by user (**CRUD**)
- Stored in **etcd**
- Desired state is **enforced** by Kubernetes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
  ...
```



```
kubectl -f apply  
coolDeployment.yaml
```



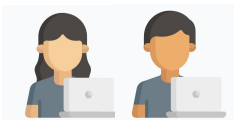
1. Observe
2. Analyze
3. React



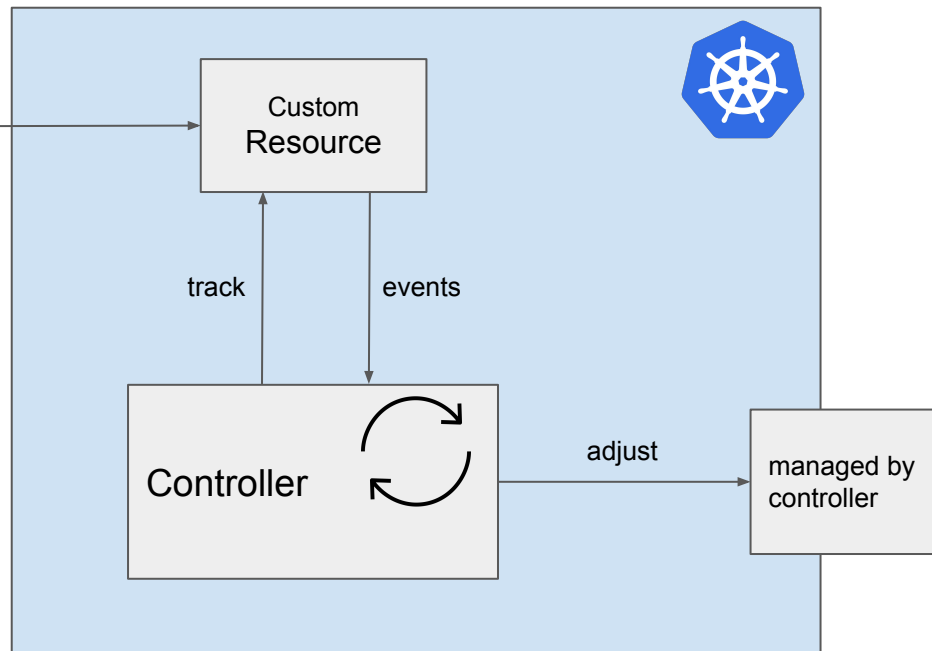
# Custom Resources and Operator pattern

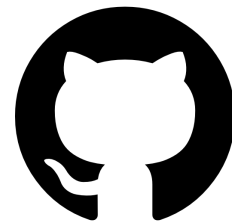
- **Custom Resources** extend Kubernetes
- Defined by **CustomResourceDefinition**
- Watched by controller
- CRDs + Controller = **Operator**
- Istio, MongoDB, Dynatrace

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: reviews-route
spec:
  hosts:
    - example.prod.svc.cluster.local
    ...
```



```
kubectl -f apply  
myResource.yaml
```

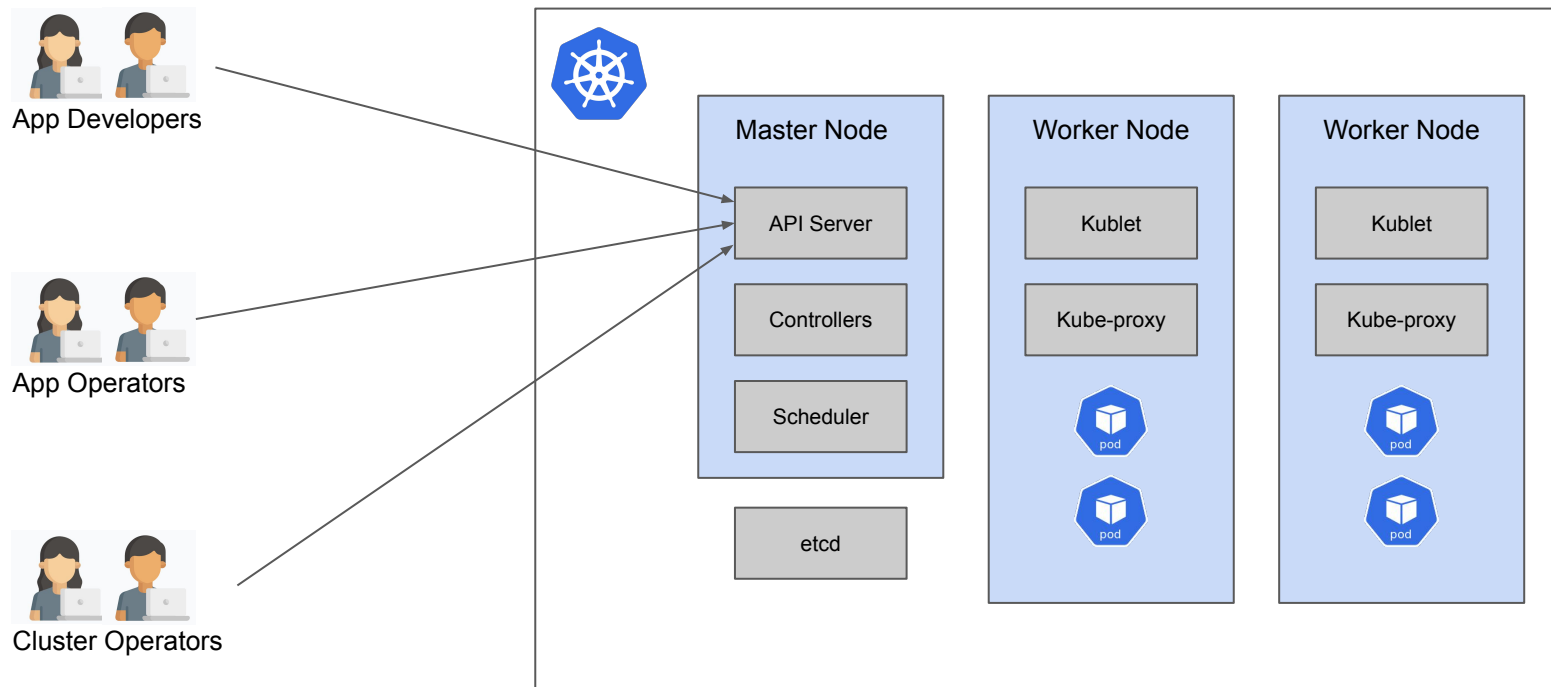




# Focus on collaboration

- **Declarative**, optionally revisioned in Git
  - enables **GitOps**
- **RESTful** CRUD operations
- **Multi-persona collaboration** using API Groups in Kubernetes

# Object / API access depends on user group





**apiVersion:** apps/v1

**kind:** Deployment

**metadata:**

name: nginx-deployment

labels:

app: nginx

**spec:**

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.14.2

ports:

- containerPort: 80

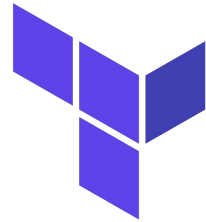
Ingredient #2

# infrastructure as code

# What is it?

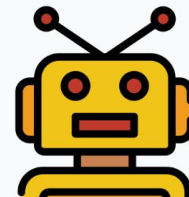
- Automate third party provisioning via applicable configurations
- It goes as follows:
  - Define your **needs** as config
  - Put them in a **repository**
  - **Apply** them via IaC tool
  - ???
  - **Profit**
- IaC tools talk to **all platforms** for you in a uniform way

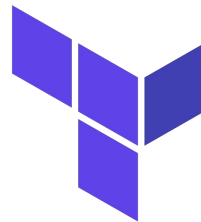
```
terraform {  
  required_providers {  
    aws = {  
      source  = "hashicorp/aws"  
      version = "~> 4.16"  
    }  
  }  
  required_version = ">= 1.2.0"  
}  
grouping  
provider "aws" {  
  region = "us-west-2"  
}  
part of provider  
resource "aws_instance" "app_server" {  
  instance_type = "t2.micro"  
  
  tags = {  
    Name = "ExampleAppServerInstance"  
  }  
}
```



# What makes it superior to us puny humans?

- Repeatability
- Consistency
- Transparency





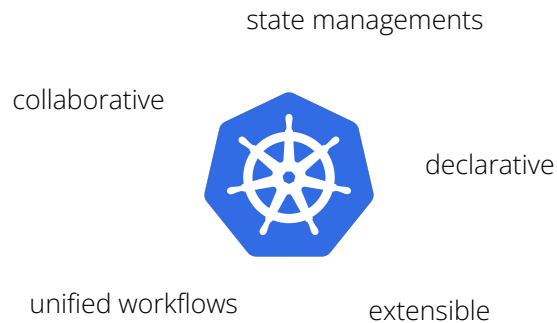
# Observations on current IaC tools

- **Template** approach
  - **Sequential / Synchronous** provisioning
  - **Access** depends on who executes it
  - Modeling **team boundaries** - who owns what
- **Monolithic** state
  - **Drift management** - keeping the state in sync with your config
  - State lock
- Yet **another** language / tool
- Thus → Infrastructure-silo **OpsPerson**

# With this in mind...

**Infrastructure as code**  
limitations and requirements

Write operators!





modify

(Custom)  
Resource

track

events

Controller

adjust

managed by  
controller

We can create this  
for infra concerns!





# Surprise announcement!

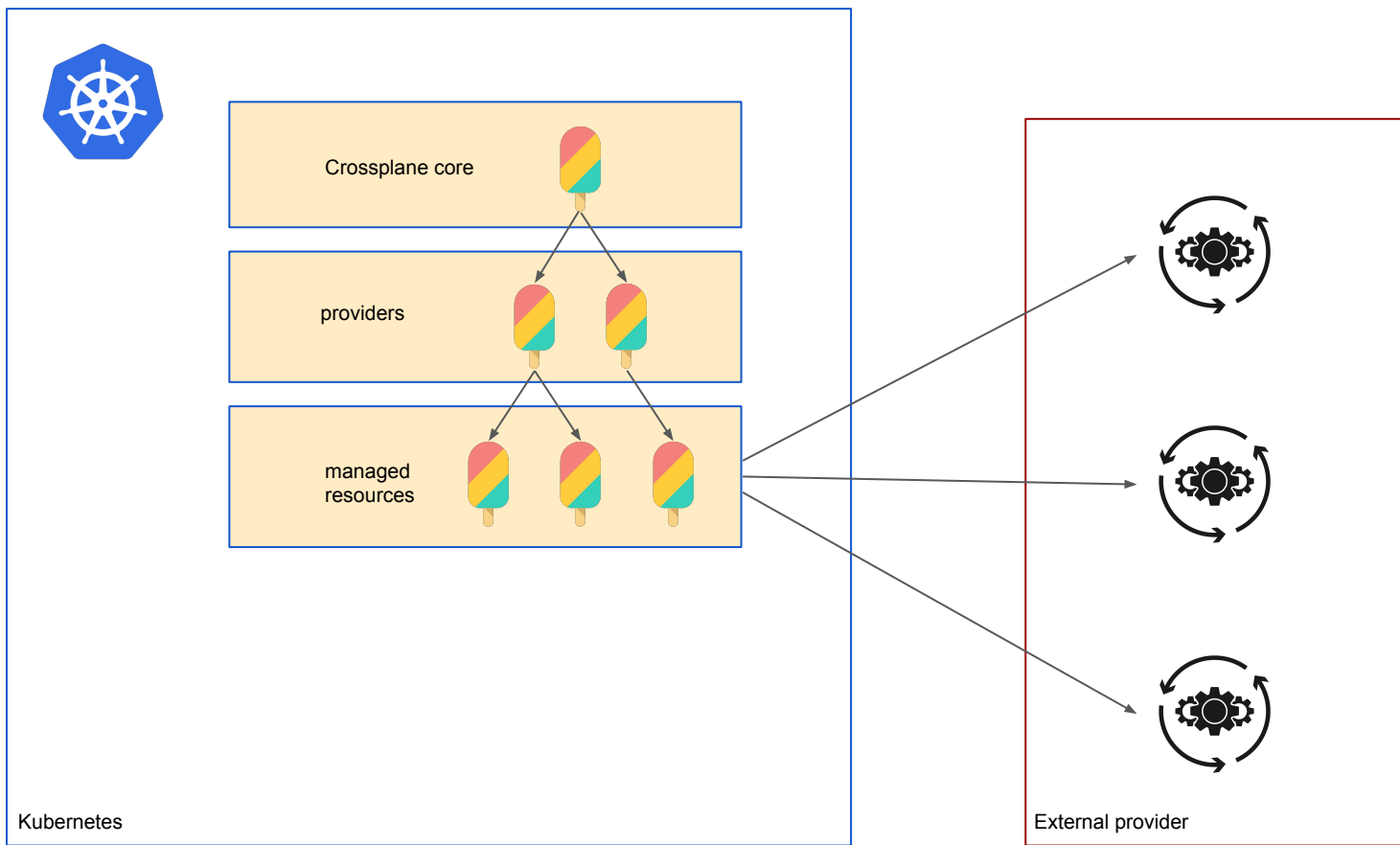


# Crossplane



# Crossplane

- Open source
- Control plane **framework**
- Built on top of **Kubernetes (principles)**
- It actually doesn't start with a **K**





# Crossplane provider strategy

- Providers are **separately maintained**
- Open source
- Can be shared via **market place**
- Can be generated via **Upjet**
- Still a **lot** of work to do

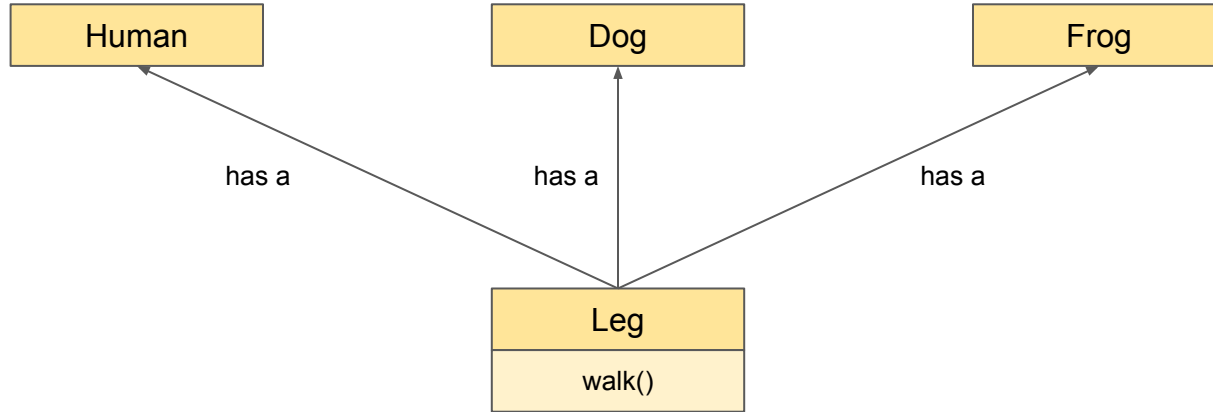
# SHOWTIME

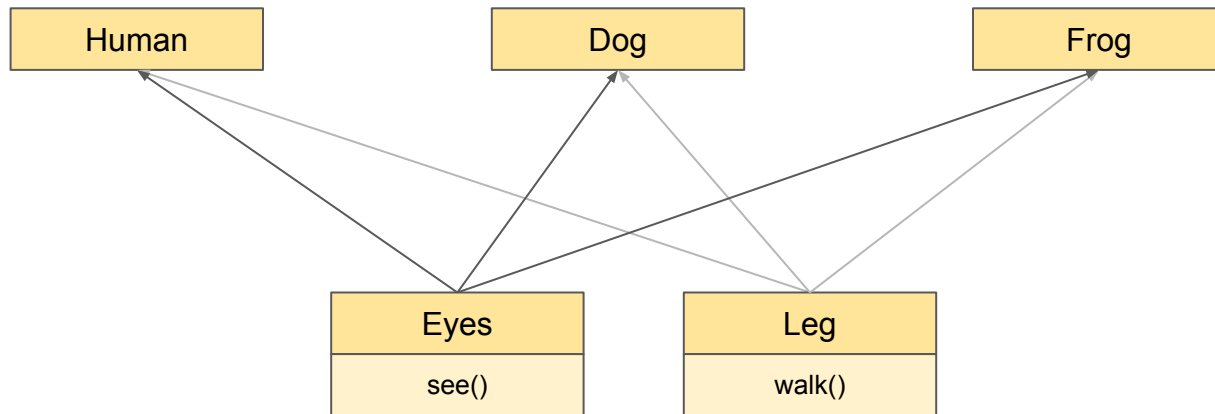
Core & providers



# Crossplane resources

- Install new resources via **providers**
- Create **recipes** from managed resources
- **XRDs** build on the concept of **CRDs**
- **Composition** pattern





Not even trying  
with an image  
this time





App Developers

namespaced resource



AppSpace



Platform Engineer

global resource



AppSpace

KubernetesCluster

Project

GCPCluster

Nodepool

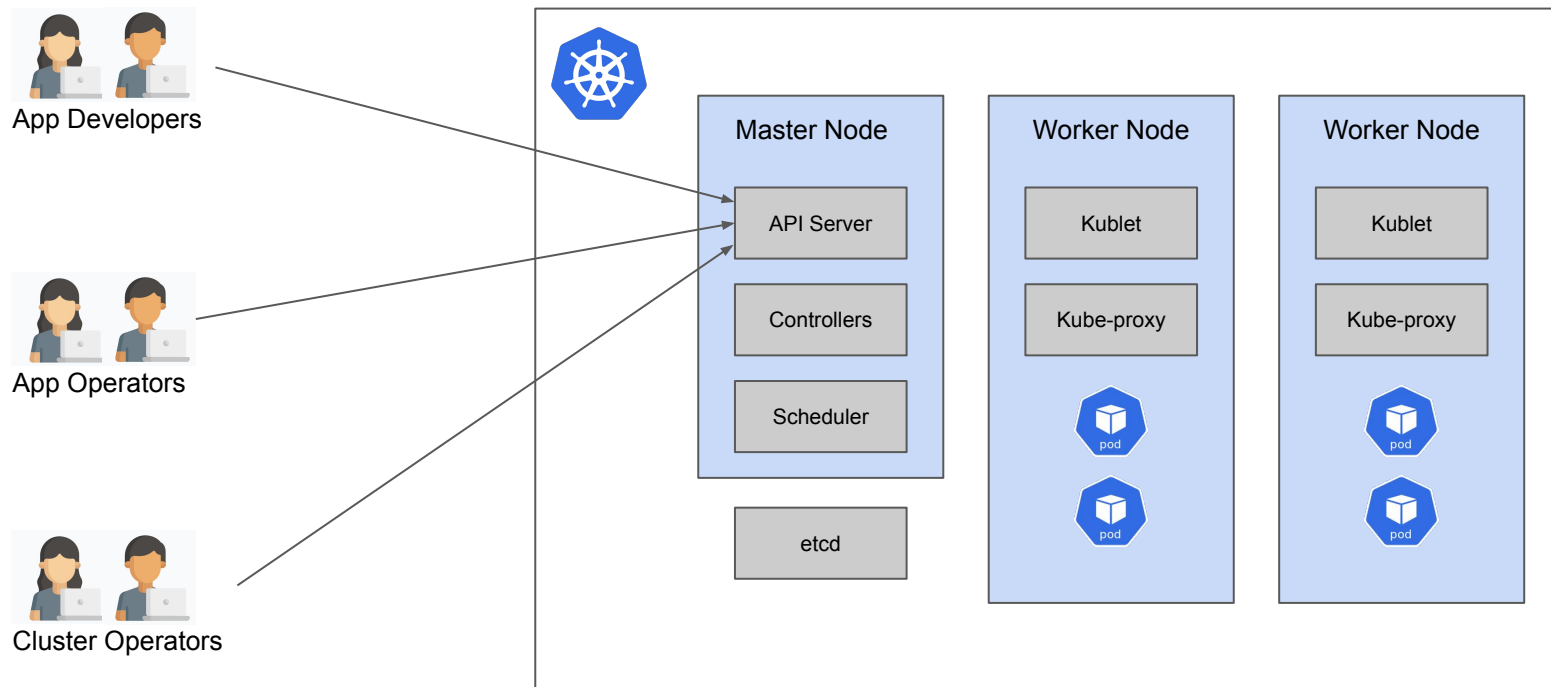
NetworkingRule

GCPProject

ProjectService

Actually you are an  
**API designer**

# Object / API access depends on user group



# SHOWTIME

XRDs

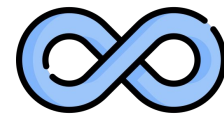
How can this help us with

# Overcoming **limitations**



# A clear API

- Who loves YAML?? **\*crowd cheers\***
- Talk to a well defined (Kubernetes) **API**
- **Semantic versioning** and a rollout strategy



# Continuous reconciliation

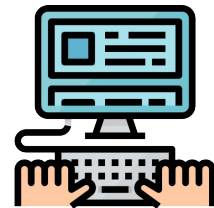
- Instead of **synchronous provisioning**
  - The same as other K8s resources
- It replaces **YOU** (as reconciliation loop)
- **Decouple** and break monolithic representation
  - Avoid configuration drift
- There always will be **dependencies**, model them correctly
  - E.g. K8s **labels**



# Clear access control

- **Third party access**
  - Providers / controllers handle access secret
  - Multiple ProviderConfigs possible
- **User access**
  - Via **Kubernetes API**
  - **RBAC** all the way





# Concept of self-service

- **Expose** specific claims to groups of people
- Underlying resources with separate state are **abstracted**
- Provisioning **without** supervision / manual interference

# SHOWTIME

Apply ALL the things

**Thank you** for listening!