SUSE

Introduction to Kubernetes

Instructor

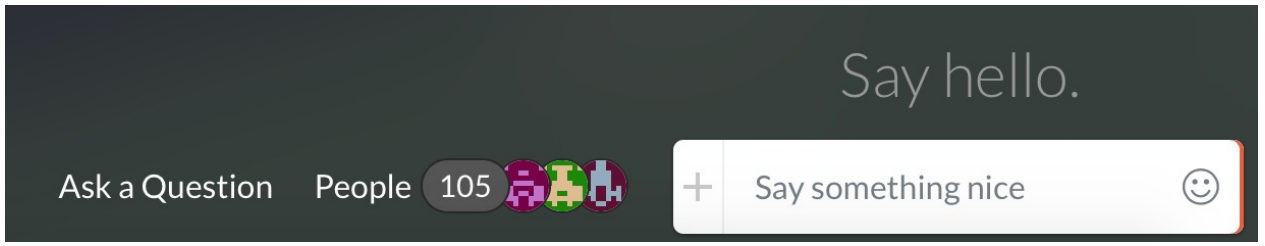
Robert Sirchia

I'm a Senior Technical Evangelist at SUSE. I specialize in cloud-native development and cloud operations.

I am all about learning and sharing this knowledge with others.

Crowdcast

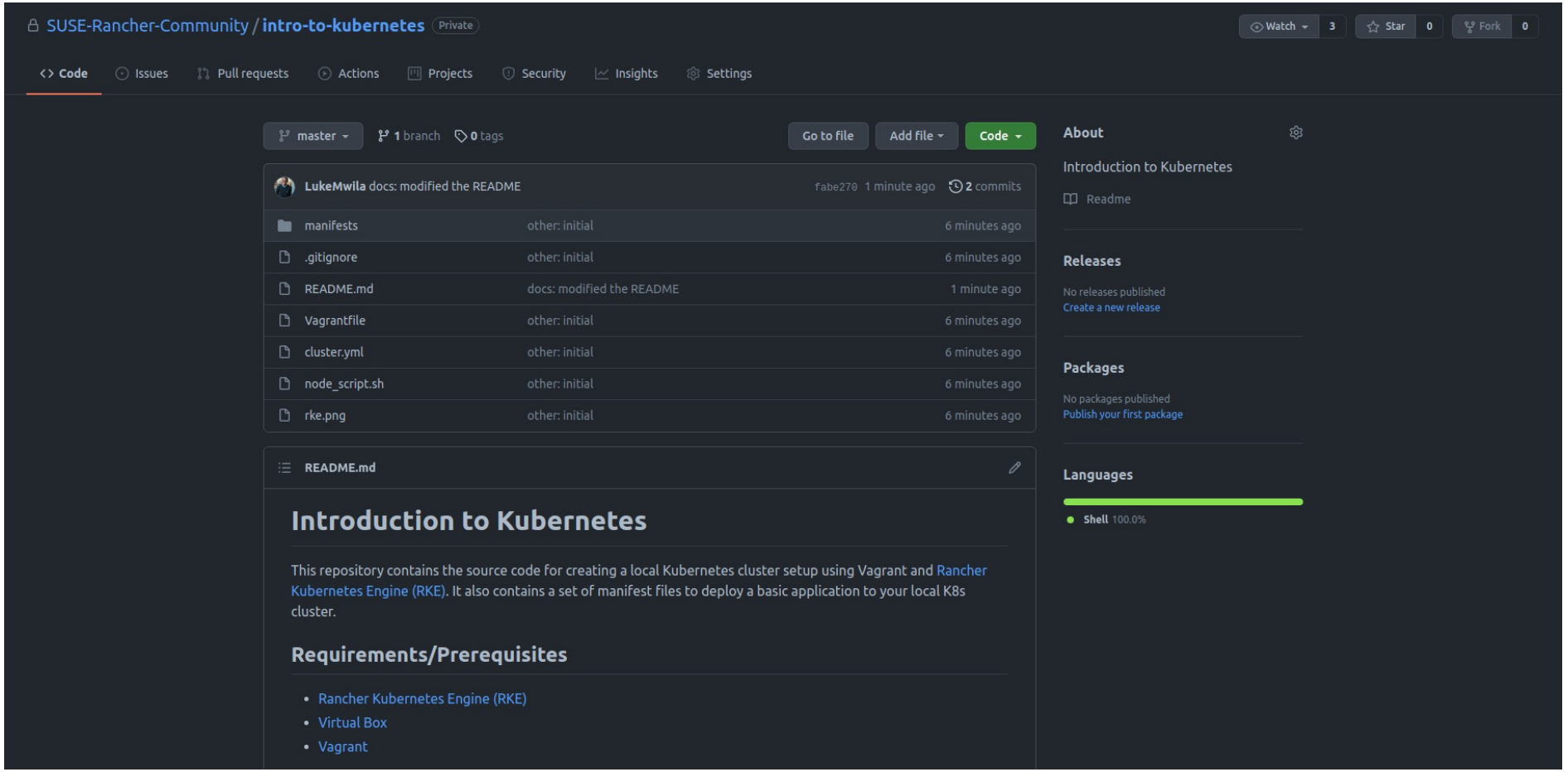
- Ask questions and upvote
- Look out for polls
- Say something nice in the chat
- Session is recorded



Agenda

1. Why Kubernetes?
2. What is Kubernetes?
3. The K8s Architecture
4. Creating a Kubernetes Cluster with K3d
5. Kubernetes Resource Management with kubectl
6. Q & A
7. The Main Kubernetes Objects
8. Q & A

Supporting Material

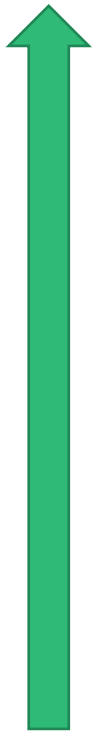


Why Kubernetes?

Benefits of Containers

- Container images are smaller and have a faster start up
- New container image snapshots are much faster
- Container allow you to isolate workloads
- Containers are lighter
- Containers can run anywhere once built
- Containers have better resource utilization

Managing Containers



Production
1000s of containers

Staging
100s of containers

Development
50 containers

- Deploying images and starting up containers
- Integrate and orchestrate modular parts
- Manage scaling of containers and clusters based on demand
- Resource balancing in container and clusters
- Provide network communication across cluster
- Load balancing across containers

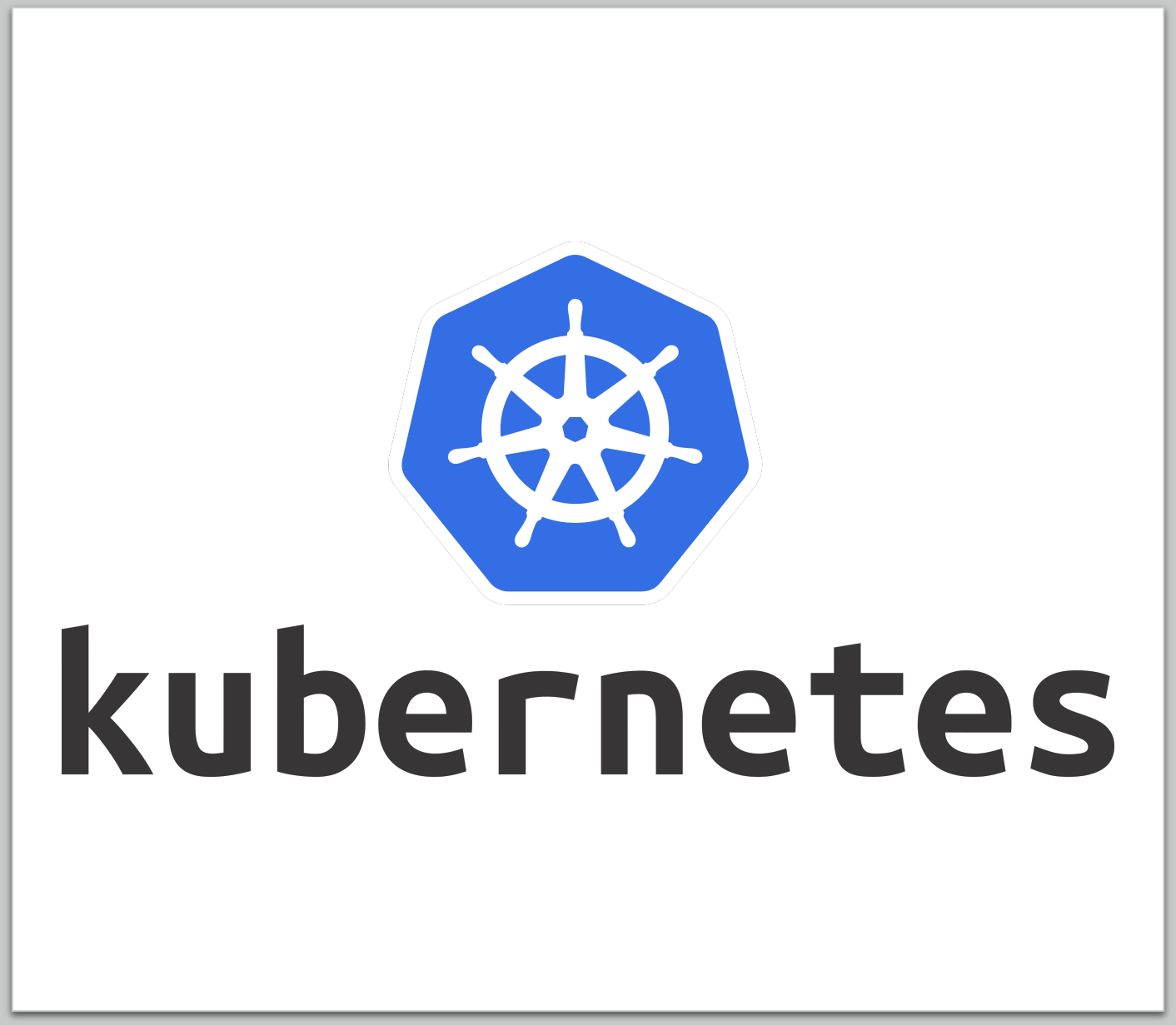
What is Kubernetes?

Container Orchestration with Kubernetes

Kubernetes is an open-source container orchestration tool or platform.

It is built to manage workloads that consist of 100s or 1000s of containers.

Niantic's gaming app, Pokémon Go, is powered by Kubernetes and scaled to manage traffic from millions of users across the globe.

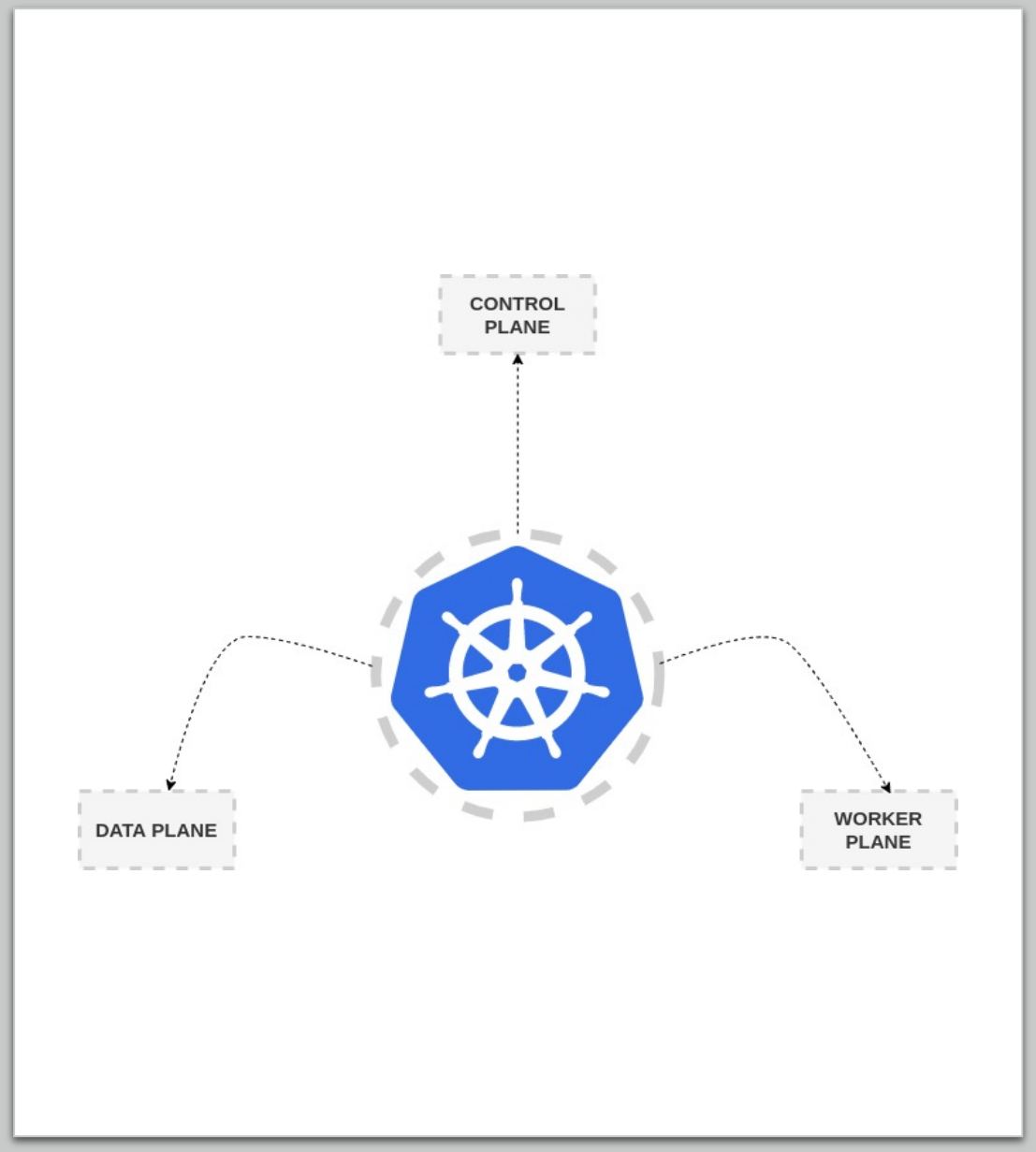


The K8s Architecture

The 3 Planes of K8s

- The Control Plane – This is the brain.
- The Data Plane – This is the memory or storage.
- The Worker Plane – This is the body responsible for running the application.

These 3 planes makeup what is known as a Kubernetes cluster. A cluster can be on one device or distributed across devices.



Creating a K8s Cluster

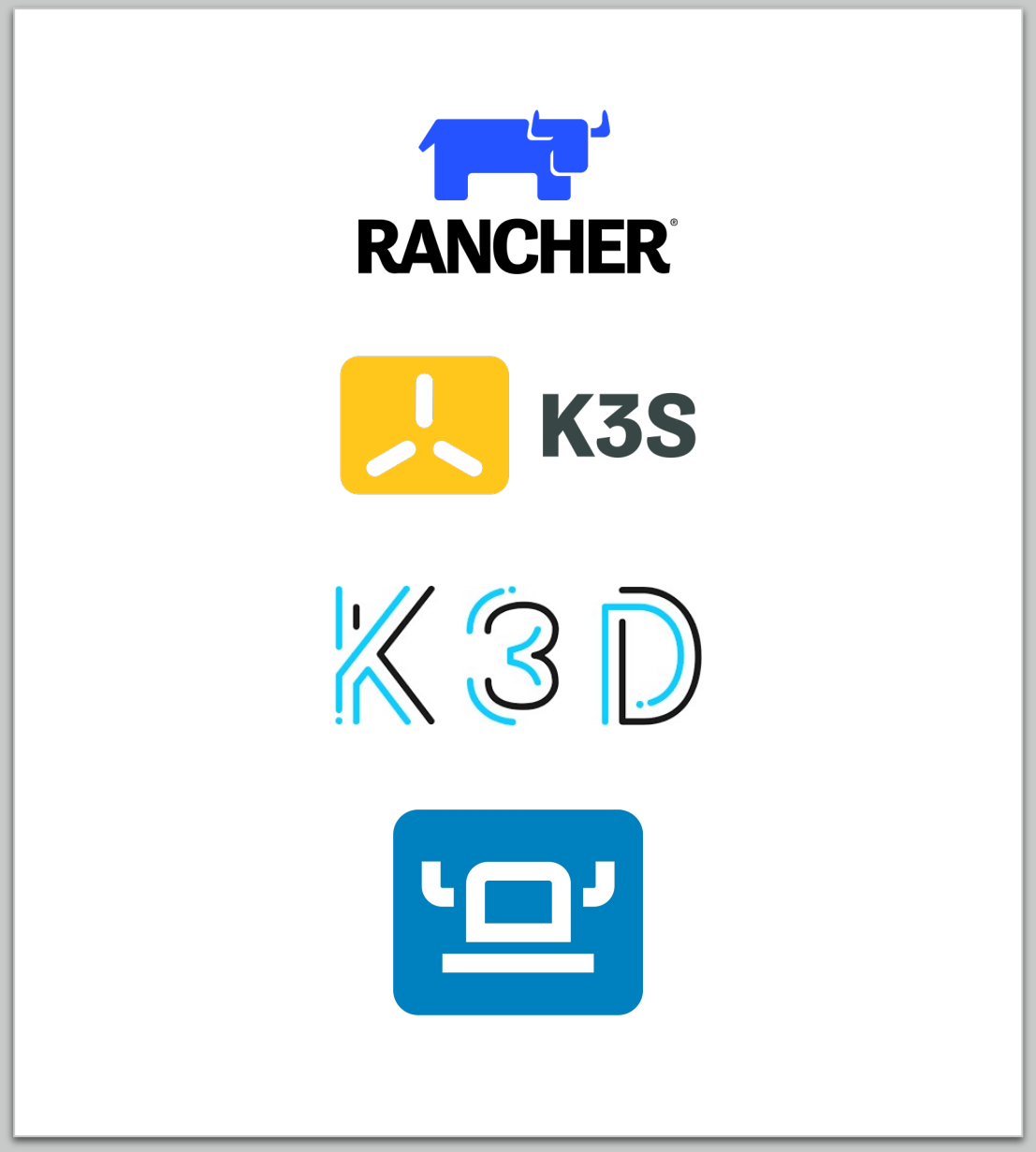
Setting up a Kubernetes cluster is complex and time consuming i.e. KH Kubernetes the hard way.

In most cases, Kubernetes administrators work with CNCF-certified Kubernetes distributions, hosted clusters or installers.

K8s distros: Rancher Kubernetes, RKE Government, K3s, K3d

K8s hosted clusters: Amazon EKS, Google Cloud GKE, Microsoft Azure AKS

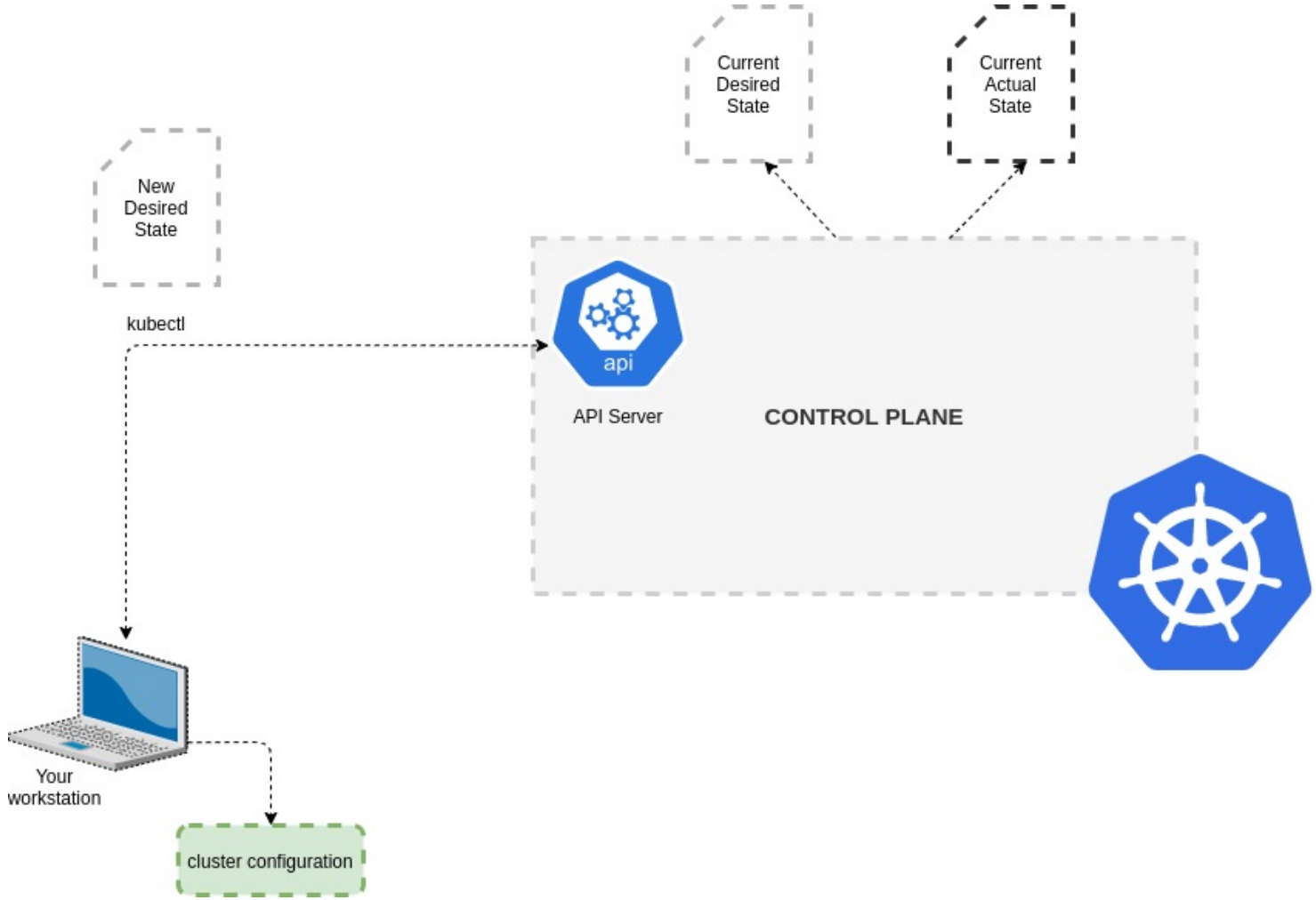
K8s installers: RKE



Kubernetes Resource Management with kubectl

Communication with K8s Cluster

- Using kubectl CLI tool
- Client to server model
- All CRUD operations via API
- Kubernetes declarative model



Declaring State in K8s with Manifests

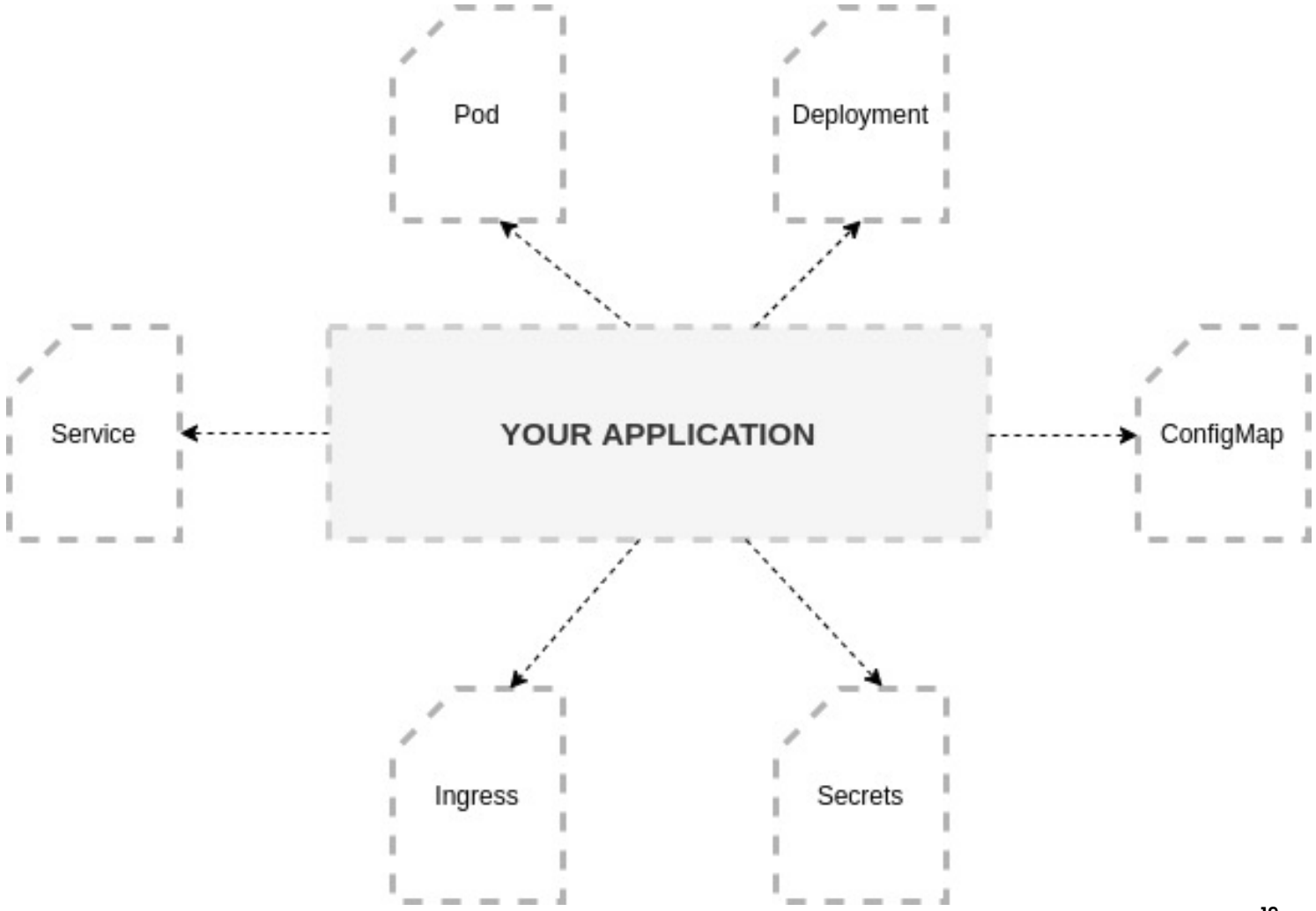
These are YAML configuration files. Most K8s objects consist of 4 top-level required fields:

- apiVersion – This field defines the API version number that the Kubernetes object belongs to.
- kind – This field specifies the type of Kubernetes object to be created.
- metadata – This field contains data that describes the object being created (i.e. name).
- spec – This field details the container configuration.

The Main Kubernetes Objects

Deploying a Kubernetes Application

- Pod
- Deployments
- Services
- Ingresses
- ConfigMaps
- Secrets



Pods

Pods are the smallest deployable artifacts in a Kubernetes cluster.

Where one or more containers can run.

Sharing networking, storage and resources.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
```

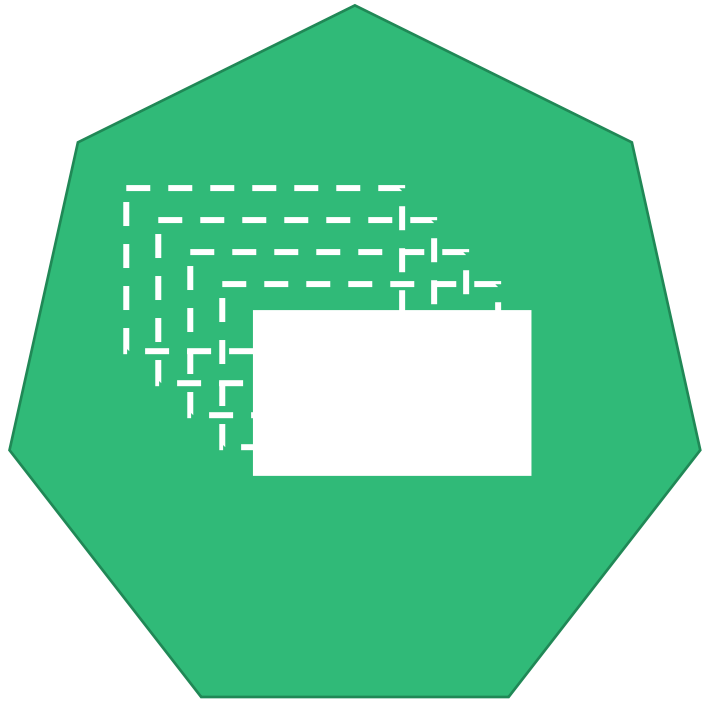
Pod yaml

ReplicaSets

ReplicaSets are actually distinct Kubernetes objects but are typically not created with manifest files. Instead, they are created as part of Deployments.

In practice, you will want multiple replicas of a container running at a particular time, as opposed to just one. The main reasons for this are:

- Fault tolerance - the more instances you have of your container, the more fault-tolerant your application will be.
- Application scaling - the more instances you have of your container, the more requests that can be handled.



```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

ReplicaSet yaml

Deployments

Deployments are a special type of Kubernetes object categorically referred to as a controller.

Kubernetes uses controllers to ensure that the cluster's existing state is always updated to match the desired state.

Deployments are used to manage Pod releases. They manage Pods in the following ways:

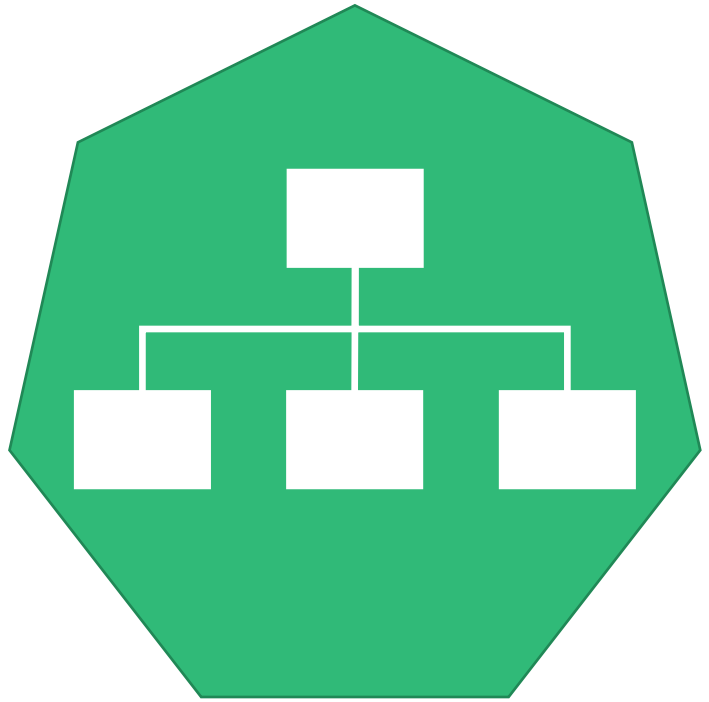
- Scaling of Pods – Are the correct number of Pods running?
- Monitoring the state of Pods – Are the Pods running or are they failing?
- Updates to Pods – Is there a new version that needs to be rolled out?

```yaml
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
```

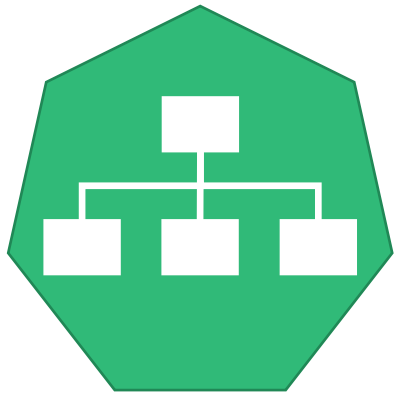
Deployment yaml

Services

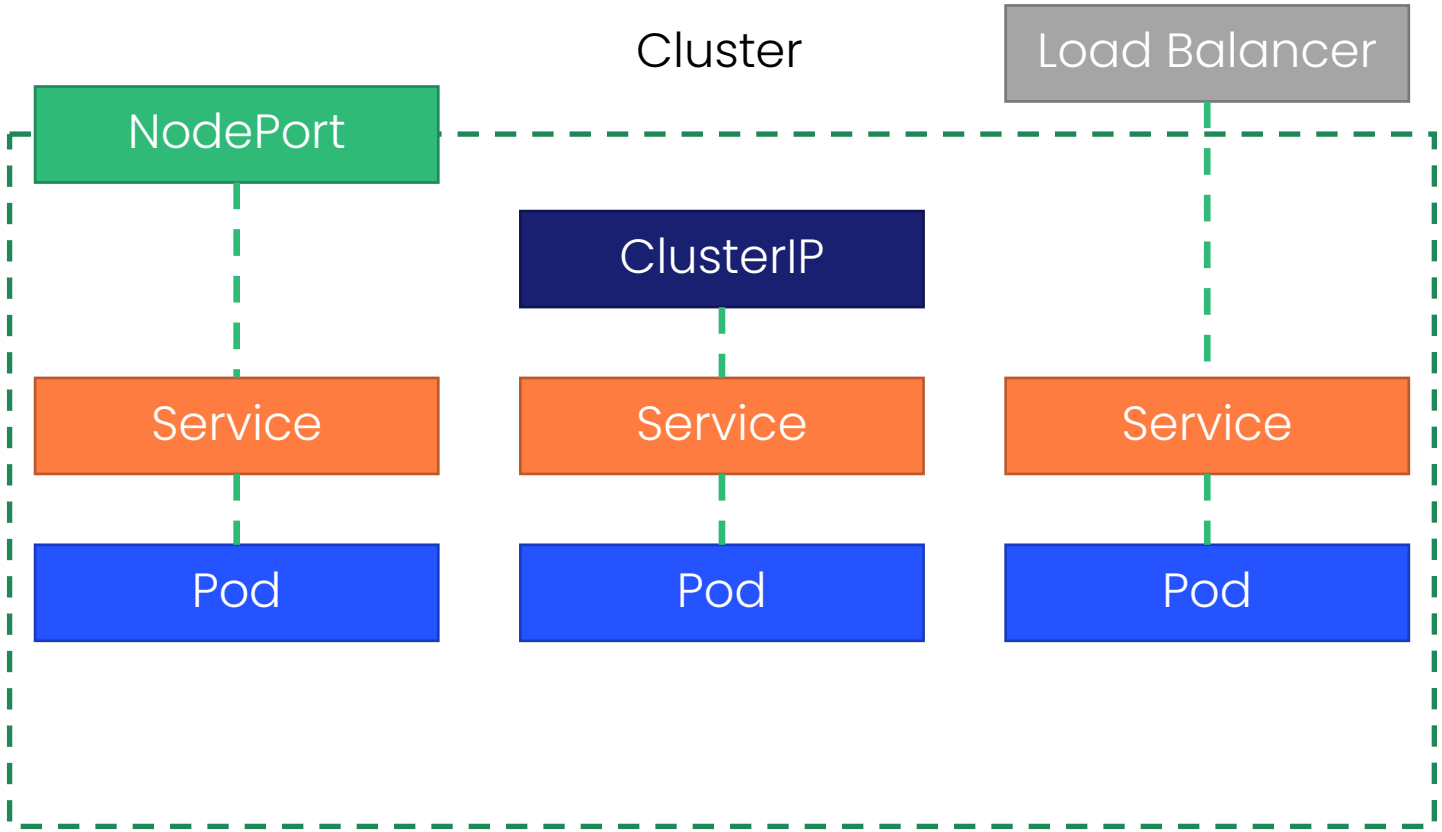
We need a solution that will discover Pods and keep track of their continuously changing IPs. That solution is the Service object. Services balance traffic across Pods in the cluster based on a label attached to the Pods.

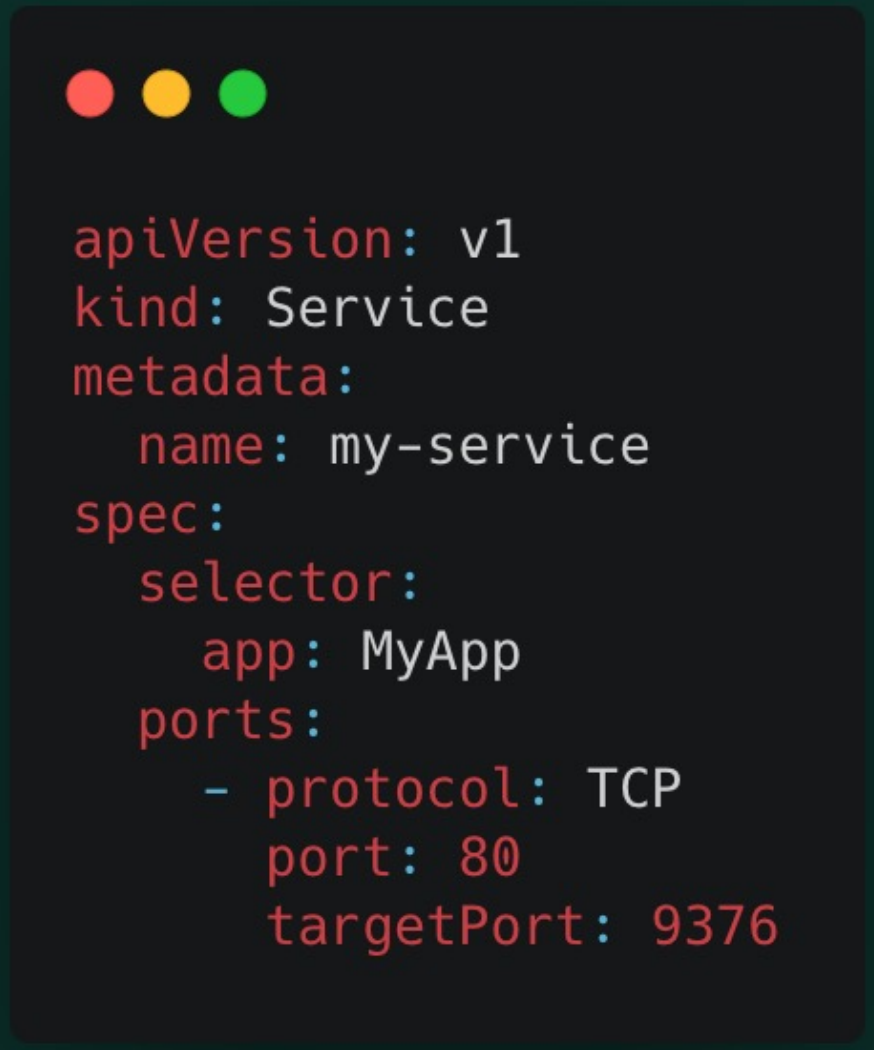


Services



- ClusterIP – Only accessible from within the cluster.
- NodePort – Gets a cluster-wide port and is also accessible.
- LoadBalancer – Integrate with the public cloud to create a load balancer in a cloud environment.





```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Service yaml

Ingresses

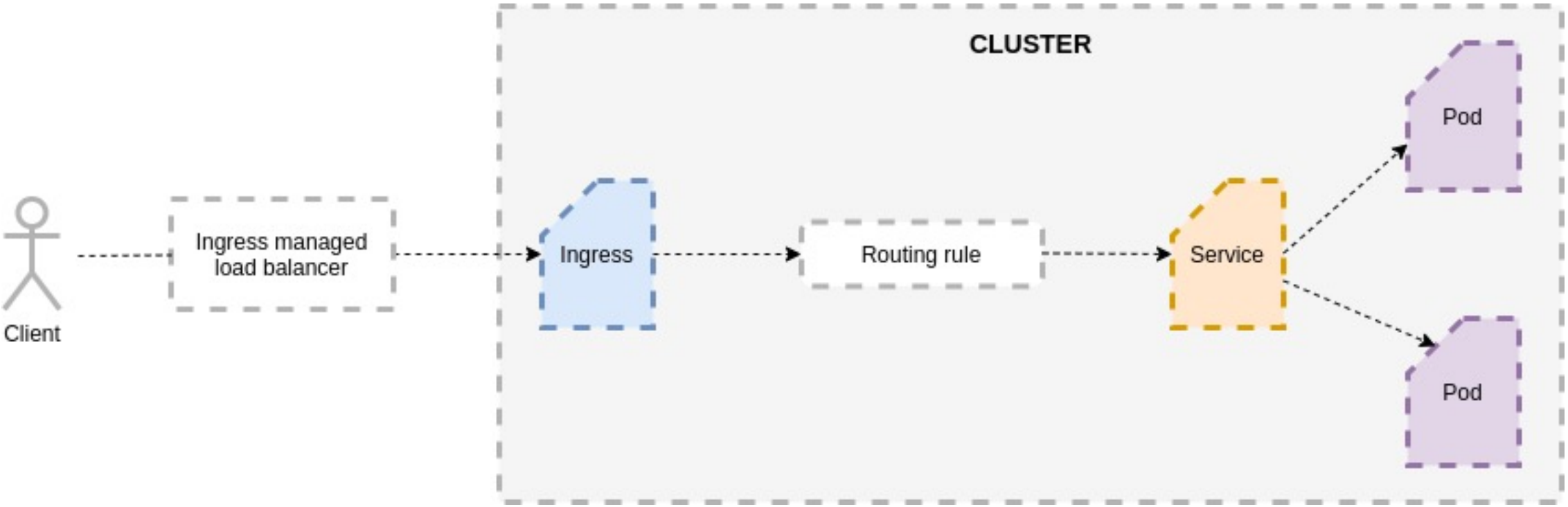
When it comes to getting external network traffic to and from your application, Services are only sufficient to a certain degree. They have issues that Ingress solves.

- HTTP Traffic
- Routing
- Security
- Cost

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Ingress yaml

Ingresses

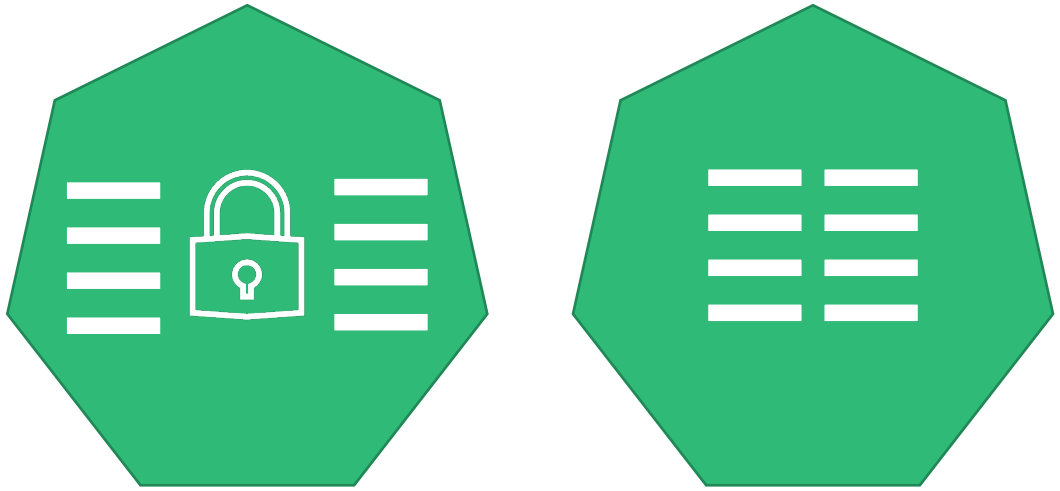


ConfigMaps & Secrets

In K8s, you may want to pass dynamic values to your applications at runtime to control how they behave. This is known as application configuration.

There are two primary ways to store configuration data in Kubernetes: ConfigMaps and Secrets

You can pass ConfigMap and Secret data to your containers as environment variables. These variables will be visible to your container process at runtime.



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

ConfigMap yaml

SUSE

Q&A

Next Steps

- Community
  - Events
- Slack

Thank You