Kubernetes Fundamentals

(Greek (Κυβερνήτης) -> kee-ver-NEE-tees)

Kubernetes - K8s

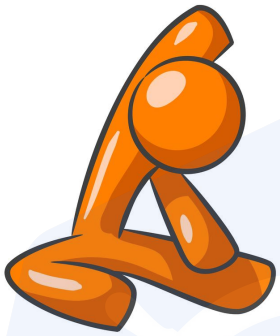
Numeronym, “a number-based word”

Examples:

- Phonetic:
  - K9 = Canine, “Police Dogs”
- Number of letters omitted:
  - l10n = localization
  - i18n = internationalization
  - K8s = Kubernetes

Agenda

- Introduction to Kubernetes
- Namespaces, Pods, and Deployments
- Service Discovery and Load-balancing
- Rolling out updates
- ConfigMaps and Secrets
- Persistent Storage
- Networking and Ingress
- Helm Package Manager
- Kubernetes Internals



Agenda

Why Kubernetes?

eficode
Kubernetes Fundamentals www.eficode.com

Agenda

Docker is awesome

but

Docker isn't enough

Agenda

Docker is awesome
for packaging

Kubernetes is awesome
for orchestration

(Some) Orchestration with Docker Compose!

docker-compose.yaml

```
version: '3'
services:
  nginx:
    image: "nginx"
    ports:
    - "80:80"
  mysql:
    image: "mysql"
    ports:
    - "3306:3306"
```


More Problems

- Multiple application stacks
- Large deployments
- Deployment / Scheduling / Updates / Rollback
- Replication / Flexible scalability
- Resilience / Availability
- Storage
- Ingress / Load balancing
- Security

We need an orchestration tool!

- Availability - scale as defined in your desired state
- Resilience - if a container exits/dies, a new one can be automatically created
- Storage - Local, NFS, iSCSI, “cloud”, etc.
- Deployments - with patterns like Canary, A/B testing, Blue/Green, etc.
- Scheduling - with Resource Limitations
- Updates - with Rolling Updates
- Networking and Cluster DNS
- Service Discovery
- Ingress routing and load balancing across replicas
- Security - with RBAC and network policies

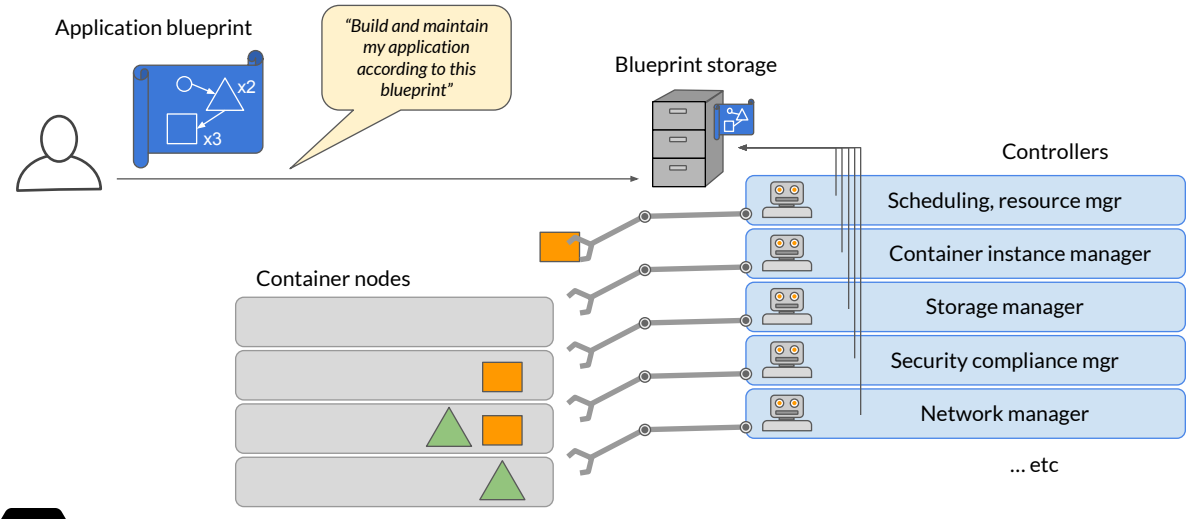
Why Kubernetes?

- Industry Standard for Container Orchestration
  - **Broad industry support** for managed solutions
  - Backed by Cloud Native Computing Foundation (CNCF)
- Strong Innovation in the K8s Area
  - **Commoditizing the container-based application infrastructure**. See CNCF landscape
  - Applications are built to run in K8s (+600 Official Helm charts)
- Based on experience from existing solutions (Google Borg, Docker)
  - Opinionated on how you **model** your container application
  - Not opinionated on the implementation of the K8s platform
    - Almost every component has a number of alternatives..

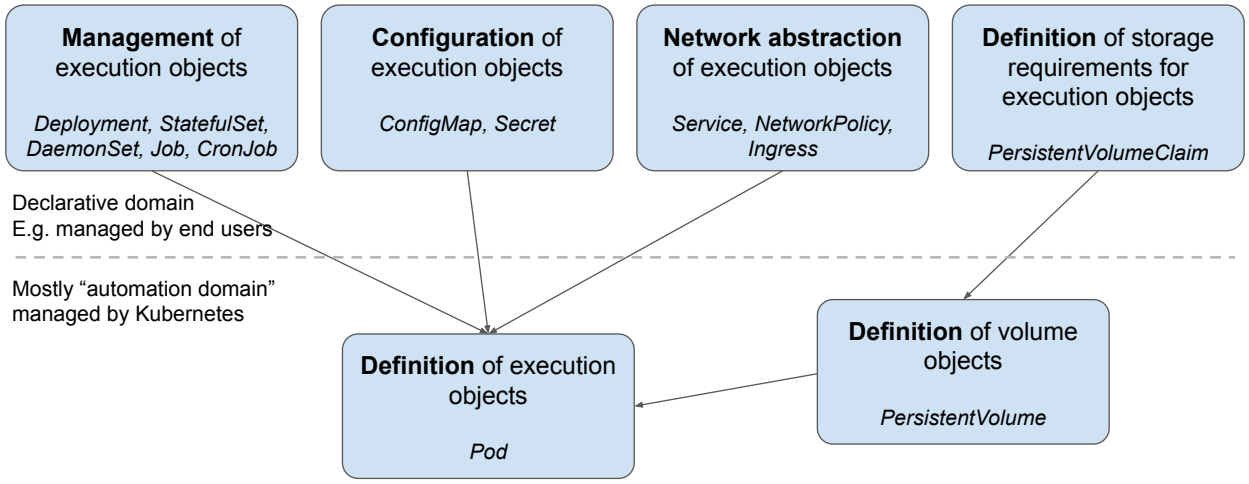
Agenda

How Kubernetes?

The Short Description of Kubernetes

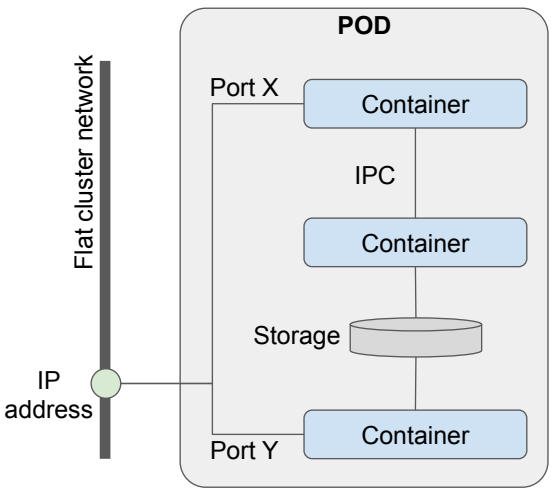


What are the Kubernetes Lego Bricks?



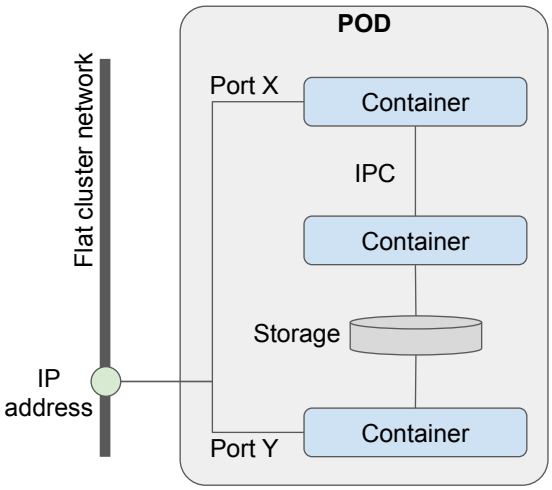
The POD Model

- Smallest object in K8s is a POD
- Consists of one or more containers
- IP addressable with ports (like a VM)



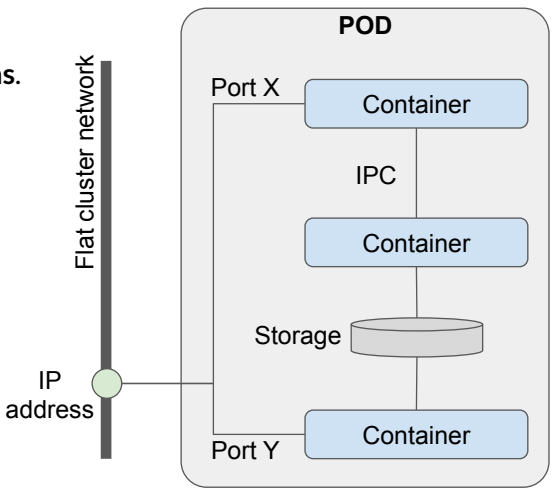
The POD Model

- Smallest object in K8s is a POD
  - Replicated as a unit.
  - Has a life-cycle.
  - Is scheduled to worker nodes.



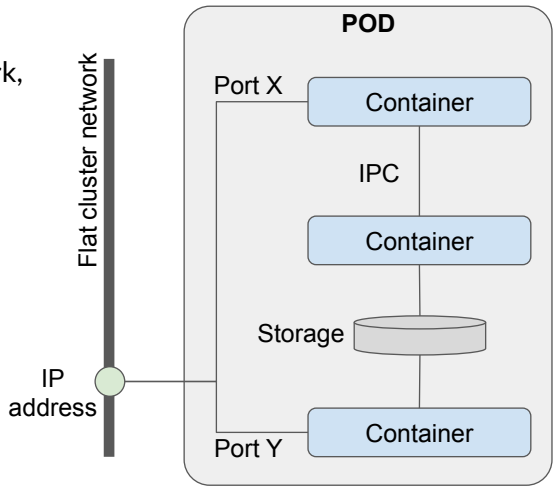
The POD Model

- Consists of one or more containers
  - Grouping enables strong **use-case patterns**. (To be continued.)
  - **NB: Only put highly related containers together in a POD**
    - **Scaled, live, die together.**

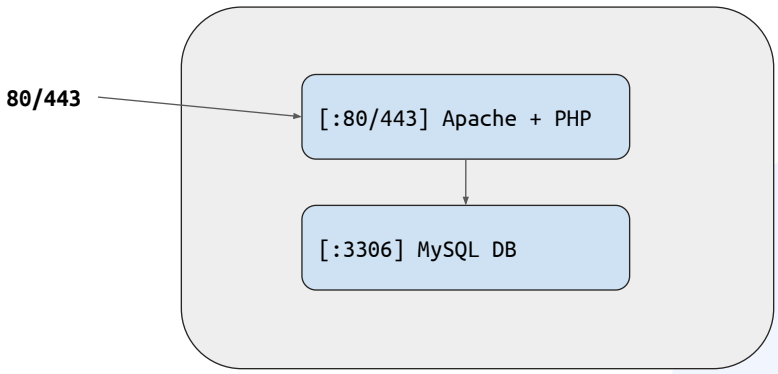


The POD Model

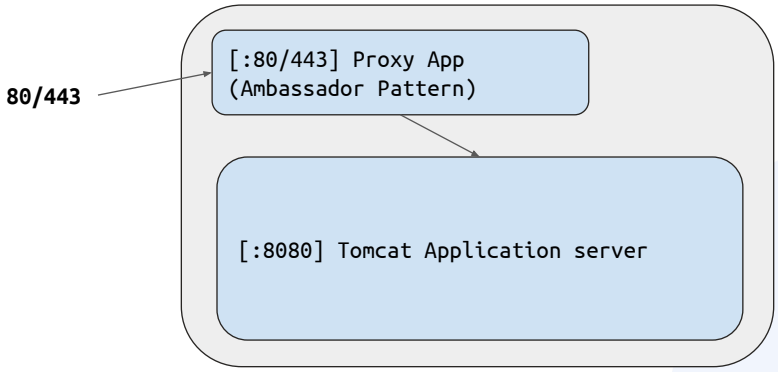
- IP addressable with ports (like a VM)
  - PODs always see network as a flat network, no NAT.



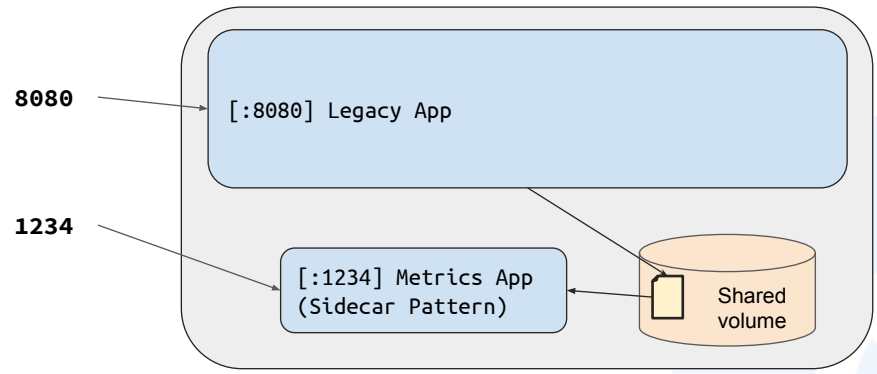
(Bad) Example of multi-container pod



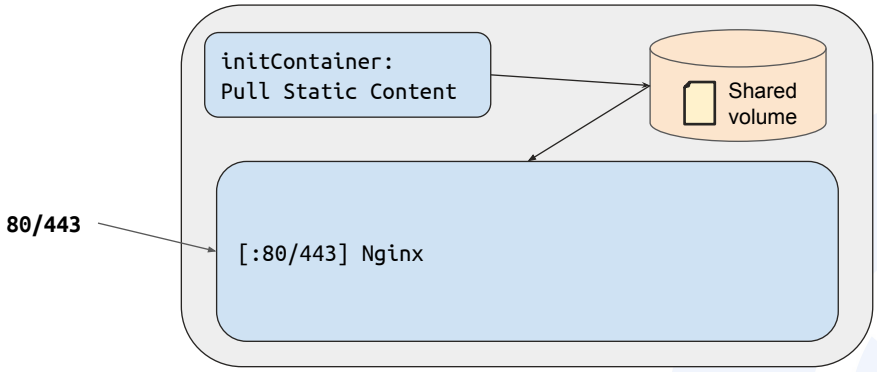
(Good) Example of multi-container pod



(Good) Example of multi-container pod

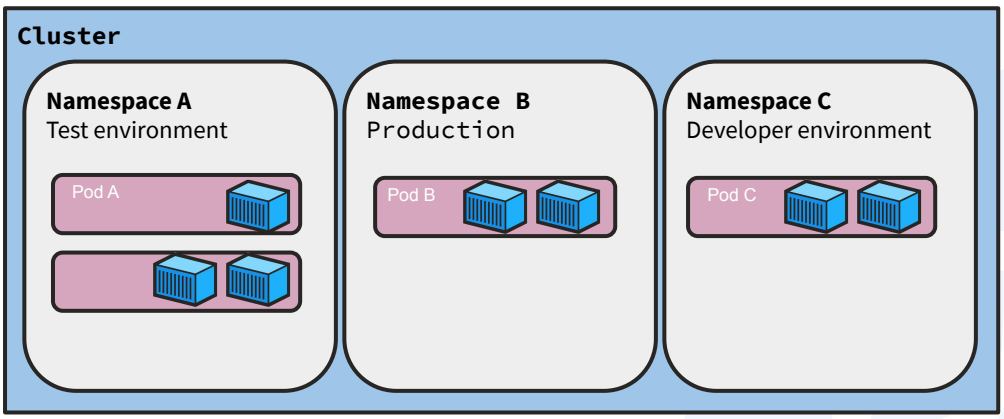


(Good) Example of multi-container pod



Namespaces

"Virtual clusters" in Kubernetes

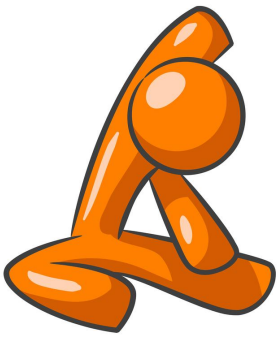


Workshop Setup

- A **shared** Kubernetes cluster on Google Cloud Platform
- Run exercises from your VM
  - `$ kubectl get nodes -o wide`
- Each trainee have a **private** namespace
- A very good tool: **`praqma/network-multitool`**

Exercise 1 - first half

https://github.com/eficode-academy/kubernetes-katas



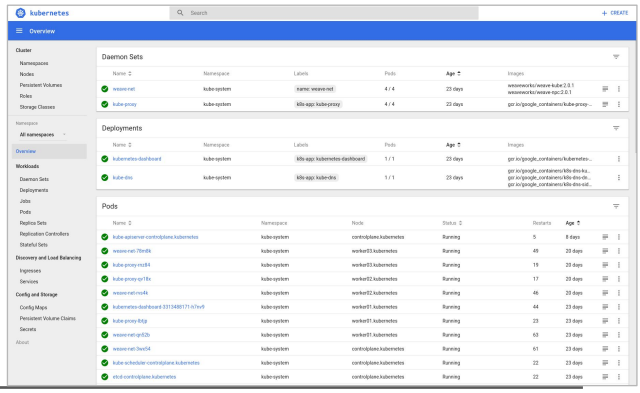
01-pods-deployments.md

*NB: Do the optional 1.1.1, but **STOP** the exercise before section 1.2!*

- Create a pod in your namespace
- Try accessing the pod with your web browser

Interacting with Kubernetes

- Command line: `kubectl`
  - Imperative (create, delete)
  - Declarative (apply -f file.yaml)
- Point and click: K8s Dashboard
  - Very imperative
- Templating
  - Very Declarative
  - Helm, Kustomize
- Helm orchestration
  - Helmfile, Helmsman



Interacting with Kubernetes (2)

Declarative - the “blueprint” approach vs. imperative - the “hacking” approach

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
  labels:
    app: frontend
spec:
  containers:
  - name: webserver
    image: nginx
    ports:
    - containerPort: 80
  - name: myapp
    image: nodeapp
```

Run an nginx Deployment and expose it to the world:

```
docker run -d --restart=always -e DOMAIN=cluster \
       --name nginx-app -p 80:80 nginx
```

On Kubernetes:

```
kubectl run --image=nginx nginx-app --port=80 \
       --env="DOMAIN=cluster"
```

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: my-webserver
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

A Pod spec.

Looks daunting.

Is not.

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion:
kind:
metadata:
  name:


spec:
  ...
```

All resources have

- apiVersion
- kind
- metadata
  - name
- spec

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion: v1
kind:
metadata:
  name:


spec:
  ...
```

apiVersion

"What API is this resource part of?"

- Beta? (might change)
- v1? (very unlikely to change)
- networking.istio.io/v1? (3rd party)

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion: v1
kind: Pod
metadata:
  name:


spec:
  ...
```

kind

"What is this resource?"

- Pod
- Deployment
- Service
- etc.

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: my-webserver
spec:
  ...
```

metadata

nb: always has a name!

"Extra information about this resource?"

- labels (n-dimensional selectors)
- annotations (arbitrary KV data)

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: my-webserver
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

spec

"How's this resource configured?" … very specific

- Pods have containers, volumesMounts
- Deployments have a template.spec
  (= Pod.spec), number of replicas, etc.
- Services have network settings...

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: my-webserver
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

"Where in the API?"

"What is this resource?"

"What's the name of this resource?"

Resources have labels.
Labels connect everything in Kubernetes.

This POD consists of one container

Defining Resources - POD YAML Example

Kubernetes resources are typically defined using YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: my-webserver
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
  - name: myapp
    image: nodeapp
```

"Where in the API?"

"What is this resource?"

"What's the name of this resource?"

Resources have labels.
Labels connect everything in Kubernetes.

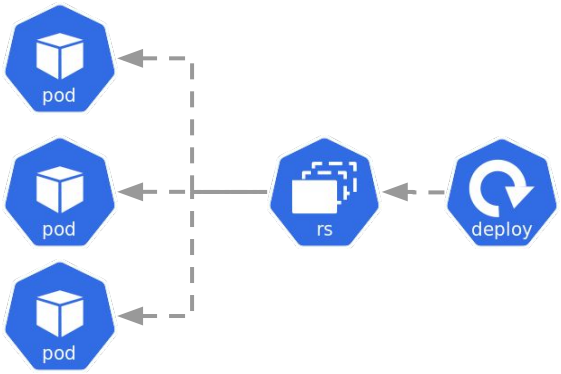
This POD consists of **two** containers

eficode

Kubernetes Fundamentals www.eficode.com

Managing Deployments in Kubernetes

How do we scale our pods?



- Replica set
- Deployment

Managing Deployments in Kubernetes

Deployments manage the release of new versions of applications

ReplicaSets define how many instances should be running

Pods are the k8s scheduling primitive

Containers provide a runtime for a process

Use cases:

- Create
- Update
- Rollback
- Scale
- Pause/Resume

Labels Connects Things in Kubernetes

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: my-webserver
    environment:testing
    partition:customer-a
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

- Everything can have labels.
- Labels are arbitrary key-value pairs
- The glue between most resources

Using multiple labels allows for an N-dimensional grouping of resources

Example 2 - A Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: my-webserver
spec:
 selector:
   matchLabels:
      app: my-webserver
 replicas: 4
 template:
   metadata:
      labels:
      app: my-webserver
   spec:
      containers:
      - name: nginx-container
      image: nginx:1.7.9
      ports:
      - containerPort: 80
```

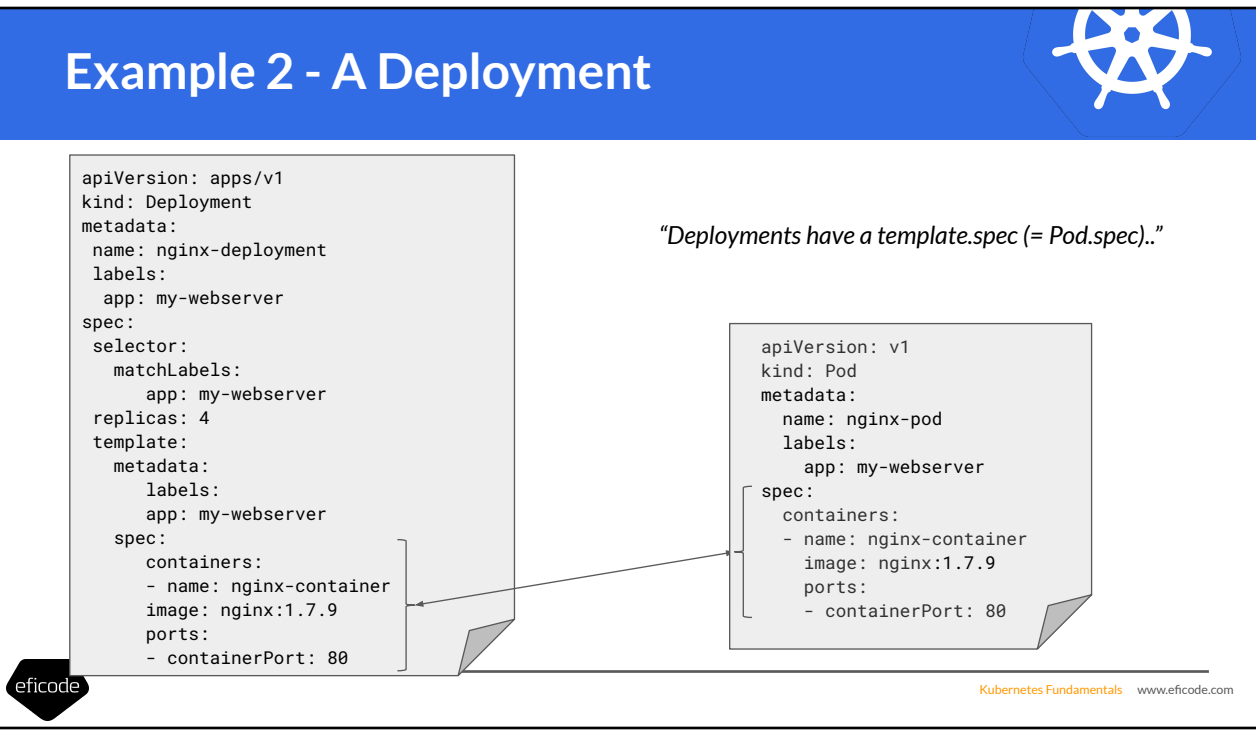
"apps/v1" !? - look in the "Reference"

Higher level abstractions (controller) allowing Kubernetes to do more automation

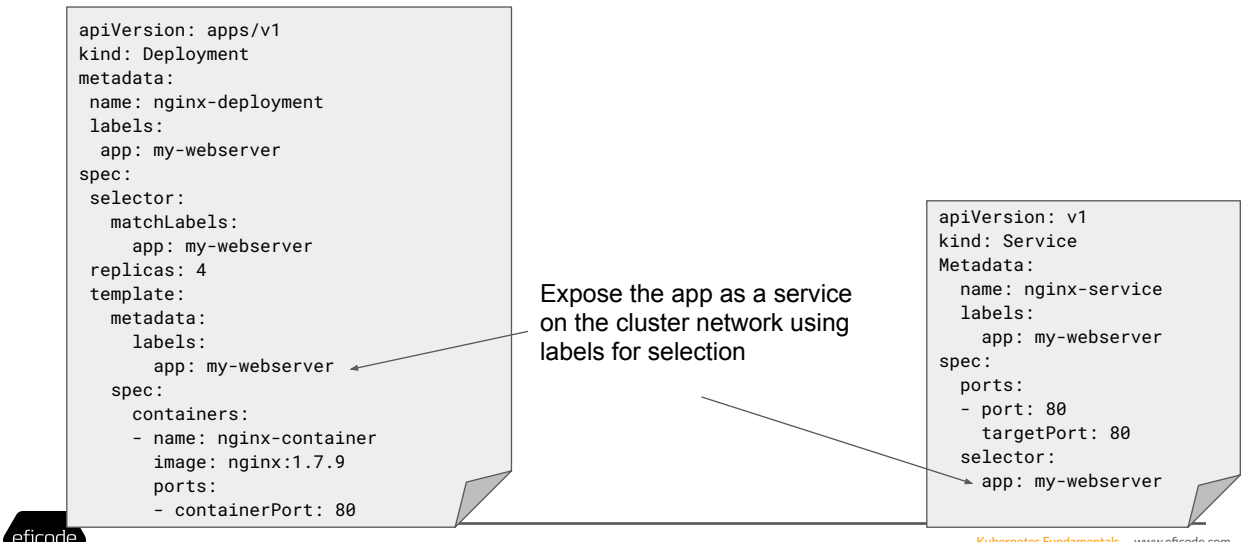
We want four replicas - Kubernetes will deploy and manage these for us

Based on this POD template

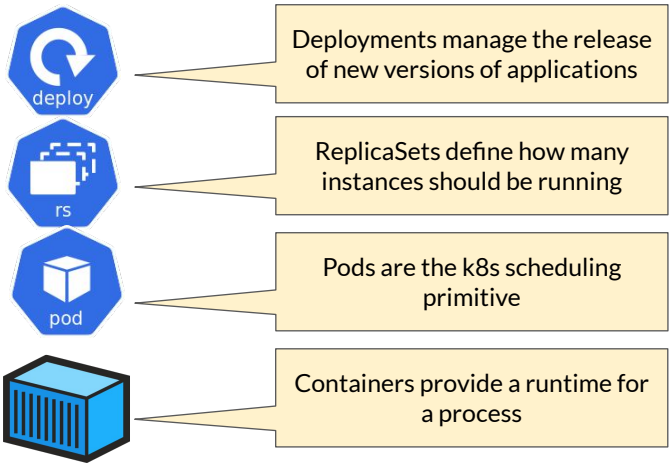
Example 2 - A Deployment



Example 2 - A Deployment



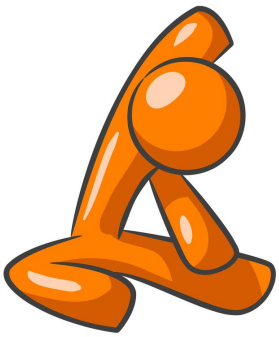
Managing Deployments in Kubernetes



Use cases:

- Create
- Update
- Rollback
- Scale
- Pause/Resume

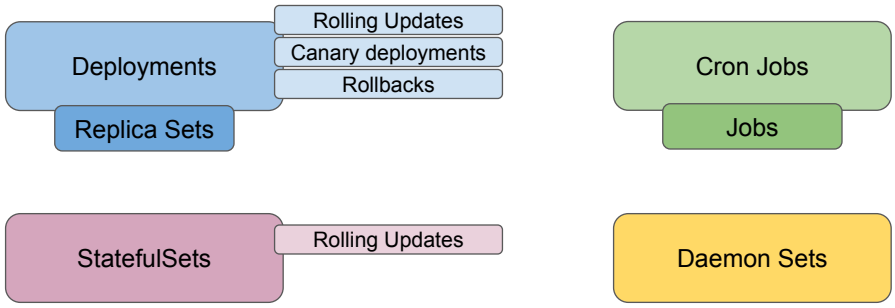
Exercise 1 - Second half



01-pods-deployments.md

- Create deployments/objects in your namespace
- Setup a simple nginx:1.7.9 deployment with single pod
- Try deleting the pod. Does it come up automatically?
- Can you access this pod from outside the cluster?

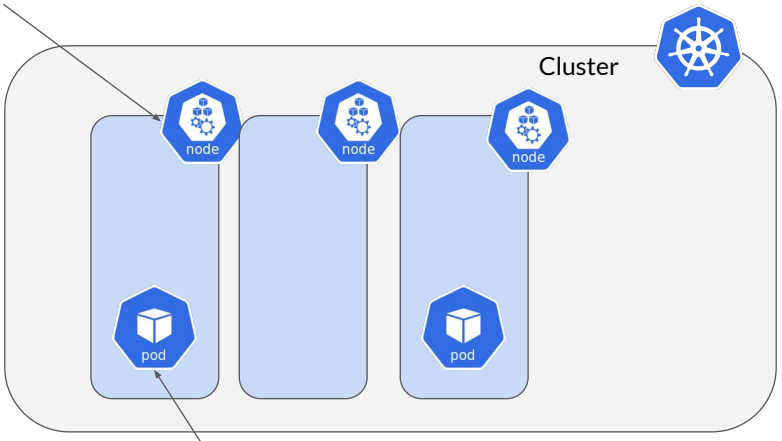
Controllers in Kubernetes



Service Discovery and Load Balancing

IP in Kubernetes

Every node has an IP



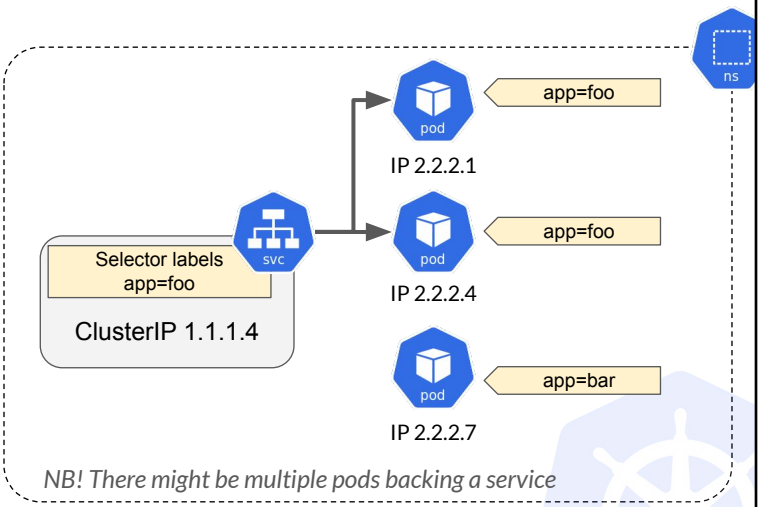
Service can have an IP
- Cluster
- NodePort
- Load Balancer

- Deployments and replica sets do not have an IP

Service Discovery in Kubernetes

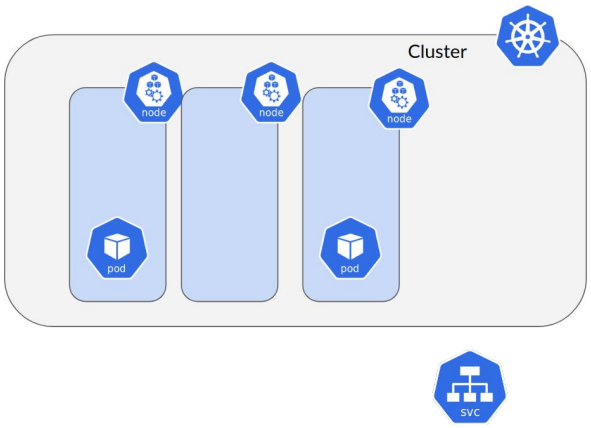
- Pods are ephemeral
  - IP's are dynamic
- Exposed through named **Service**
  - Selected through labels

Service can have an IP
- Cluster
- NodePort
- Load Balancer



NB! There might be multiple pods backing a service

Service Discovery in Kubernetes



DNS names

Pod:

```
<ip>.<namespace>.pod.cluster.local
```

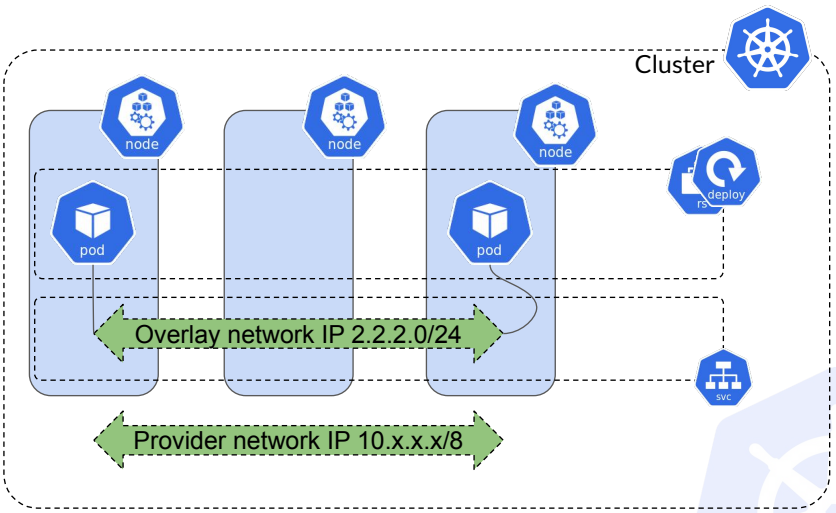
Service:

```
<name>.<namespace>.svc.cluster.local
```


Service Discovery in Kubernetes

Cluster IP

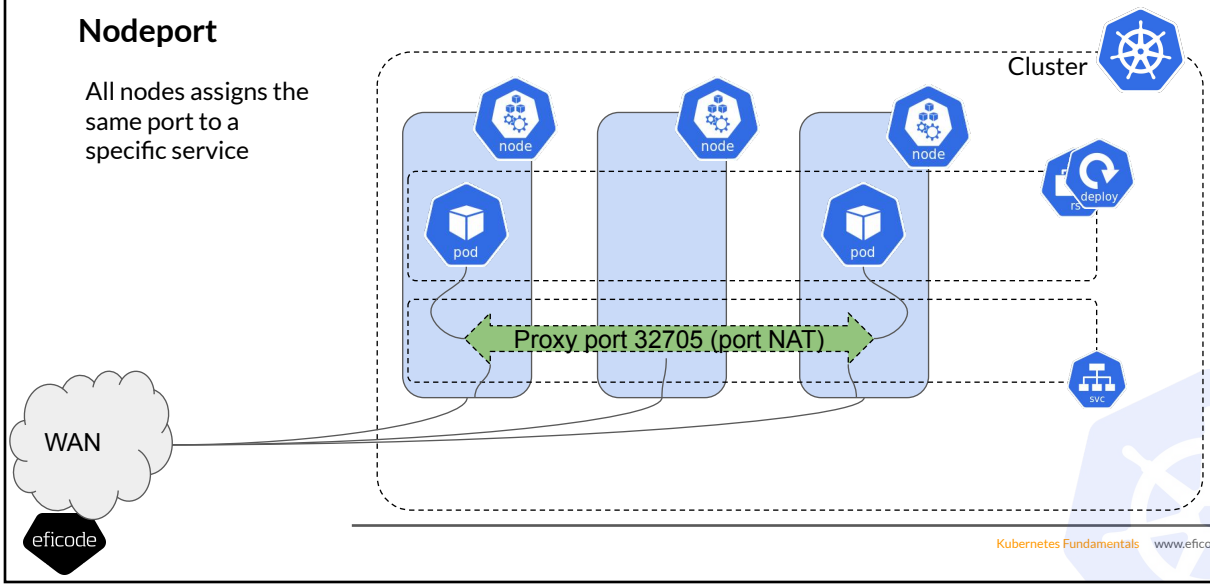
- For internal communication
- The IP is static to the service



Service Discovery in Kubernetes

Nodeport

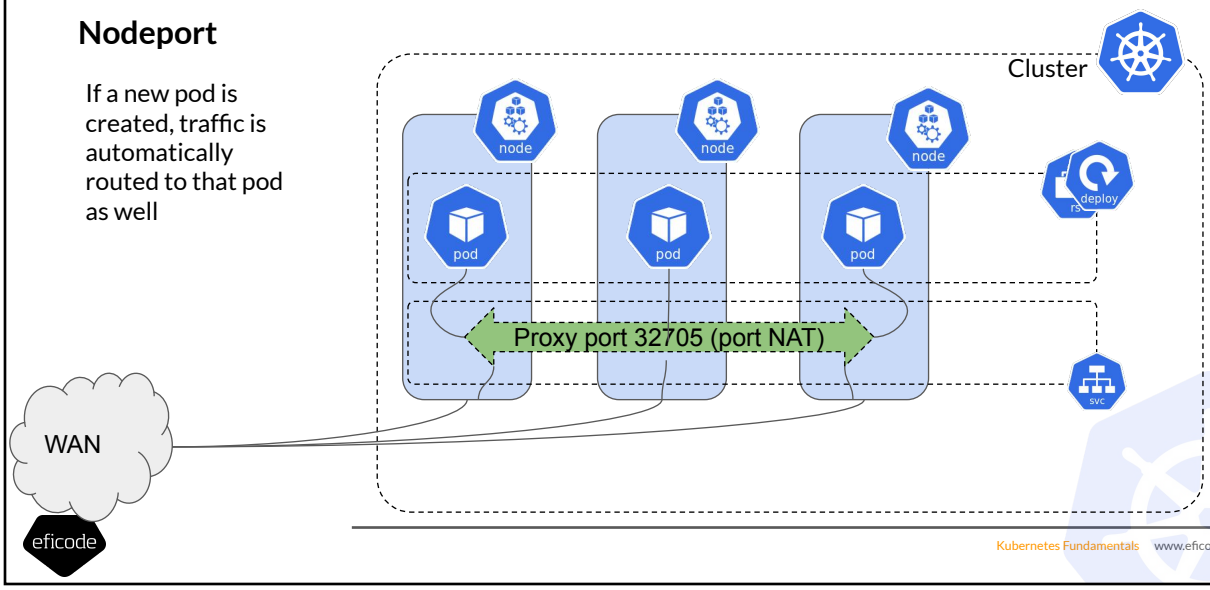
All nodes assigns the same port to a specific service



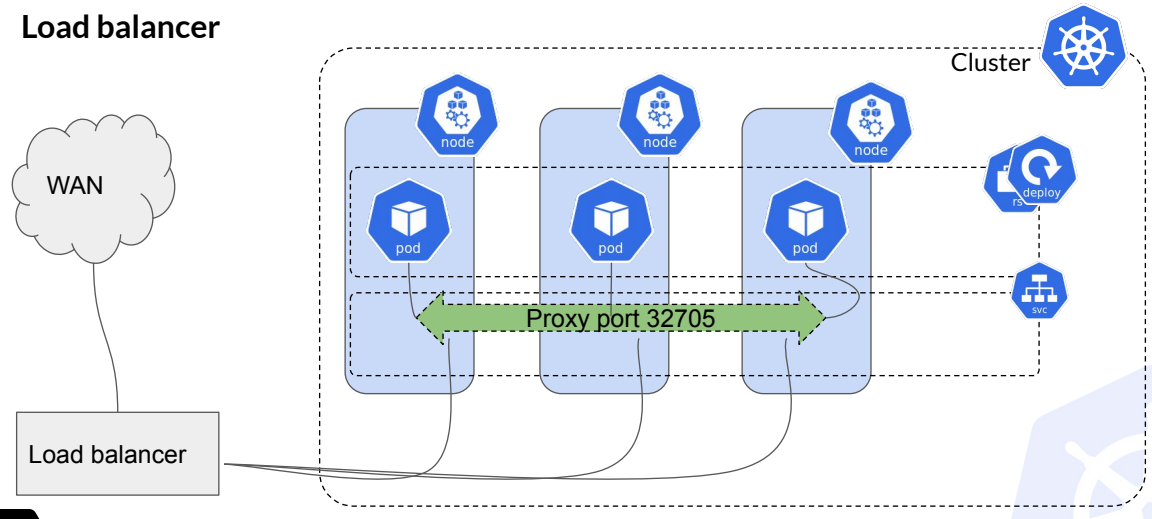
Service Discovery in Kubernetes

Nodeport

If a new pod is created, traffic is automatically routed to that pod as well



Service Discovery in Kubernetes



Network Abstraction - L4/Services

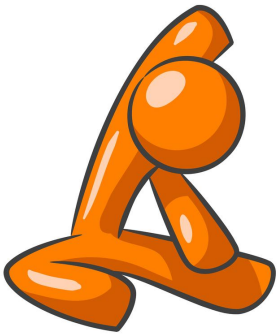
- Services are a networking **L4 'virtual IP' abstraction**
  - **Cluster IP address** is **fixed** while backends can change as PODs change
  - **Traffic is load-balanced** (round-robin, least connections etc.)

- DNS names
  - Don't use IP addresses in your applications, **always use DNS**
  - **DNS search path** reflects service name and namespace:

    `<service-name>.<namespace>.svc.cluster.local`

    In the same namespace we can use DNS short-name "`<service-name>`"

Exercise 2



02-service-discovery-loadbalancing.md

- Expose the deployment - as a service - type ClusterIP
- Can you access this service from outside the cluster?
- Delete / re-create the same service as type NodePort
- Can you access this service from outside the cluster?
- Delete / re-create the same service as type LoadBalancer
- Can you access this pod from outside the cluster?
- Increase number of replicas to four (4)
- What content you see each time you access the service?

Rolling out updates

Updating Resources

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: frontend
spec:
 selector:
   matchLabels:
     app: nginx
 replicas: 4
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: nginx
       image: nginx:1.7.9
       ports:
       - containerPort: 80
```

We can update/patch resources to change our application blueprint, e.g. replicas and image versions

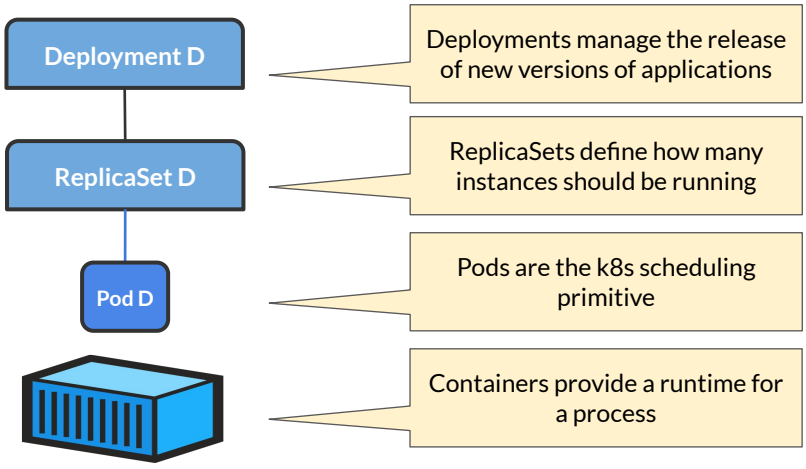
Consider if this is best practice!

```
$ kubectl label ...
$ kubectl set <env | image | resources | ...> …
$ kubectl edit ...
$ kubectl patch ...
$ kubectl scale --replicas 3 …
$ kubectl apply -f manifest.yaml
$ kubectl replace ...
```

Kubernetes has versioned update for the **`template`** part of Deployments, DaemonSets and StatefulSets

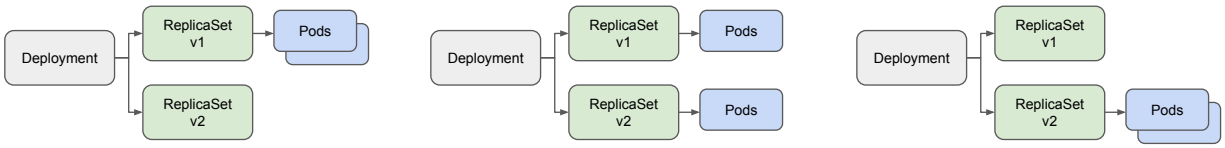
- *All other changes are in-place!*

Managing Deployments in Kubernetes



Rollout Strategies

All controllers have **`RollingUpdate`**

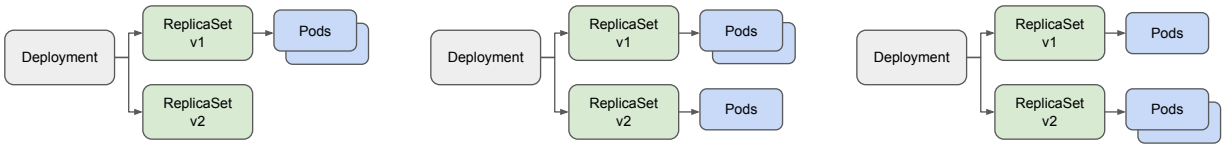


Rollout Strategies

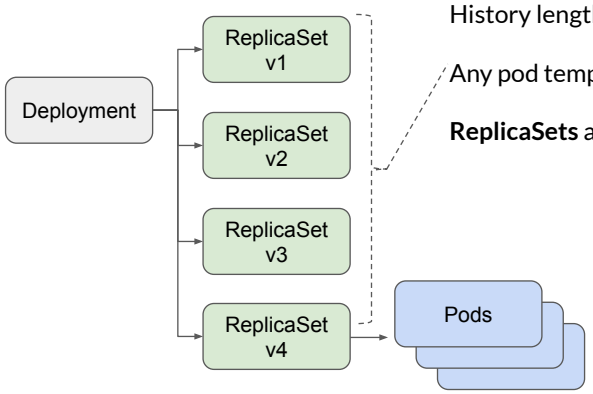
`RollingUpdate` on **Deployments** have parameters:

- **`maxSurge`** (- number of pods above **desired** number)
- **`maxUnavailable`**
  - **`maxSurge`** and **`maxUnavailable`** can't be 0 at the same time

Example: maxSurge = 1, maxUnavailable = 0



Rollout History



Deployments' history is tracked in **ReplicaSets**

History length is defined by `spec.revisionHistoryLimit`

Any pod template change triggers a new **ReplicaSet** / "new version"

ReplicaSets are named using the hash of the pod template

Rollout History Caveats (1)

```
$ kubectl create deployment nginx --image nginx:1.13.6
$ kubectl patch deployment nginx -p '{"spec": {"revisionHistoryLimit": 10}}'
$ kubectl set image deploy nginx nginx=nginx:1.13.7 --record
$ kubectl set image deploy nginx nginx=nginx:1.13.8 --record
$ kubectl set image deploy nginx nginx=nginx:1.13.9 --record
$ kubectl rollout history deploy nginx
REVISION  CHANGE-CAUSE
1         kubectl create deployment nginx --image=nginx:1.13.6 --record=true
2         kubectl set image deploy nginx nginx=nginx:1.13.7 --record=true
3         kubectl set image deploy nginx nginx=nginx:1.13.8 --record=true
4         kubectl set image deploy nginx nginx=nginx:1.13.9 --record=true

$ kubectl set image deploy nginx nginx=nginx:1.13.8 --record
$ kubectl rollout history deploy nginx
REVISION  CHANGE-CAUSE
1         kubectl create deployment nginx --image=nginx:1.13.6 --record=true
2         kubectl set image deploy nginx nginx=nginx:1.13.7 --record=true
4         kubectl set image deploy nginx nginx=nginx:1.13.9 --record=true
5         kubectl set image deploy nginx nginx=nginx:1.13.8 --record=true
```

This is what brings us trouble since we will be reusing an existing ReplicaSet

Ehh…? 1.13.7 is the third step in history, and revision 3 is missing

Expected, 1.13.8 is latest version and 1.13.9 is previous version

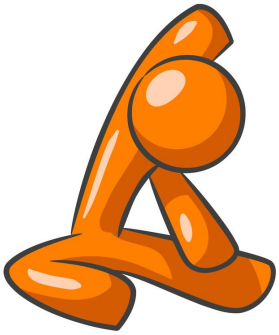
Rollout History Caveats (2)

```
$ kubectl rollout undo deployment nginx
$ kubectl rollout history deploy nginx
REVISION  CHANGE-CAUSE
1         kubectl create deployment nginx --image=nginx:1.13.6 --record=true
2         kubectl set image deploy nginx nginx=nginx:1.13.7 --record=true
5         kubectl set image deploy nginx nginx=nginx:1.13.8 --record=true
6         kubectl set image deploy nginx nginx=nginx:1.13.9 --record=true

$ kubectl rollout undo deployment nginx
$ kubectl rollout history deploy nginx
REVISION  CHANGE-CAUSE
1         kubectl create deployment nginx --image=nginx:1.13.6 --record=true
2         kubectl set image deploy nginx nginx=nginx:1.13.7 --record=true
6         kubectl set image deploy nginx nginx=nginx:1.13.9 --record=true
7         kubectl set image deploy nginx nginx=nginx:1.13.8 --record=true

$ kubectl rollout undo deployment nginx
$ kubectl rollout history deploy nginx
REVISION  CHANGE-CAUSE
1         kubectl create deployment nginx --image=nginx:1.13.6 --record=true
2         kubectl set image deploy nginx nginx=nginx:1.13.7 --record=true
7         kubectl set image deploy nginx nginx=nginx:1.13.8 --record=true
8         kubectl set image deploy nginx nginx=nginx:1.13.9 --record=true
```

Another rollout undo brings us back to 1.13.9

We never come back to 1.13.7

Exercise 3



03-rolling-updates.md

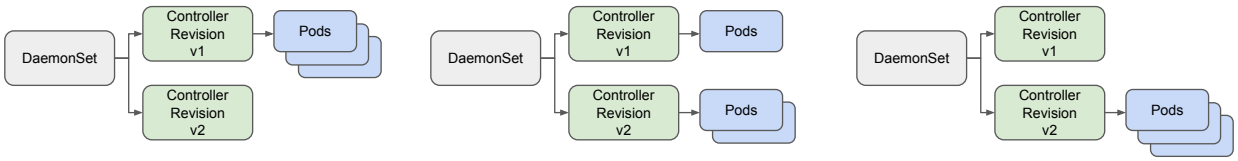
- Learn how to apply changes to running deployments
- Use the previous nginx:1.7.9 deployment with four replicas
- Rollout an update to nginx:1.9.1
- Test with **`curl`** from local computer
- Describe the deployment and replicaSets and notice the difference

Rollout Strategies - RollingUpdate

`RollingUpdate` on **DaemonSets** have parameters:

- **`maxUnavailable`**
  - Can't be 0

Example, maxUnavailable = 2

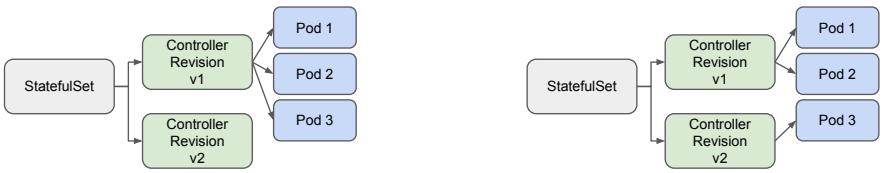


Rollout Strategies - RollingUpdate

`RollingUpdate` on **StatefulSets** have parameters:

- **`Partition`**
  - Only updates pods with "number" higher than **partition**
  - If **partition** is set higher than **replicas**, no pods are updated

Example, partition = 2



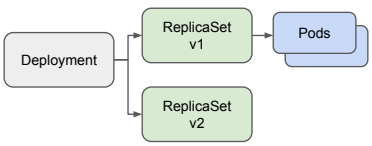
Rollout Strategies - RollingUpdate, more

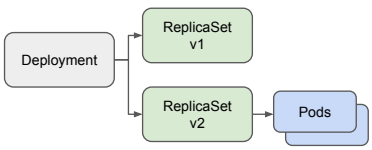
`spec.minReadySeconds`

- Helpful for slowing down rolling updates
- Is set on the Deployment / StatefulSet / DaemonSet spec
  - Not on the **`RollingUpdateStrategy`** (just like the **`revisionHistoryLimit`**)

Other Rollout Strategies

Deployments have **`Recreate`**

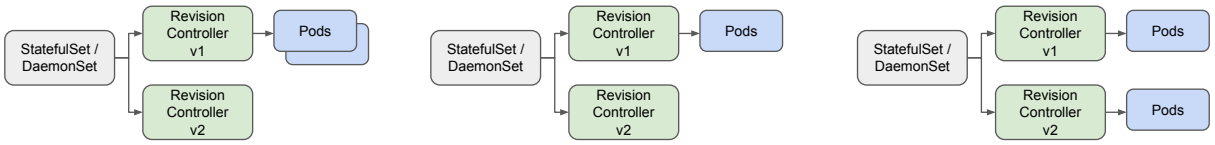




Other Rollout Strategies

StatefulSets and **DaemonSets** have **`OnDelete`**

(Manual deletion)



Break - Recap

- Pods, Deployments
- Redundancy
- Exposing with services (NodePort and LoadBalancer)
- Labels are important
- Updating replicas and images with different strategies
- Rollback

ConfigMaps and Secrets

12 Factor Applications

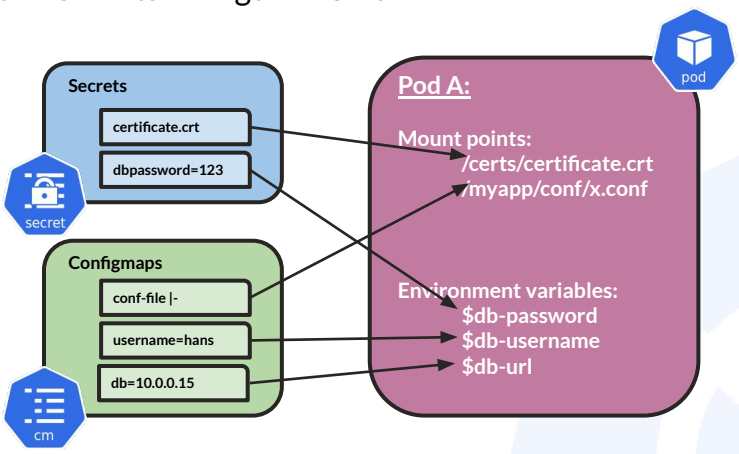
Methodology for application design - resonates well with the thinking behind Kubernetes

- Log to stderr/stdout
- Separate build, release and run stages
- Disposability and statelessness
- Horizontal scale-out
- Backing services are attached services
- Separate configuration and secrets from the executable

- Handle missing dependencies
  - Config files should be optional and have sane defaults
  - Retry access to external services - eliminate the need to deploy services in a specific order

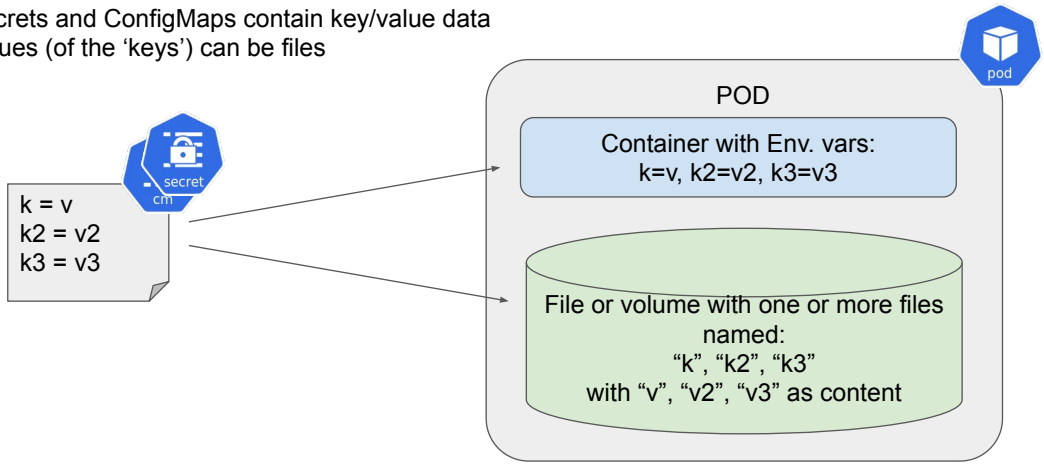
Configuration Objects in Kubernetes

Configmaps and Secrets configure PODs

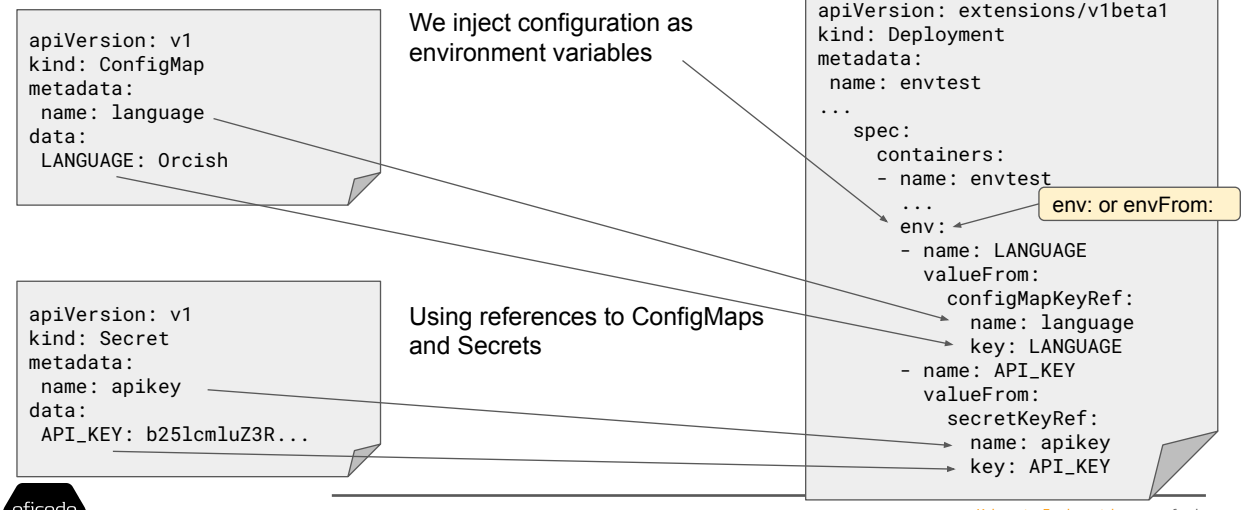


Configuration Objects in Kubernetes

- Secrets and ConfigMaps contain key/value data
- Values (of the 'keys') can be files



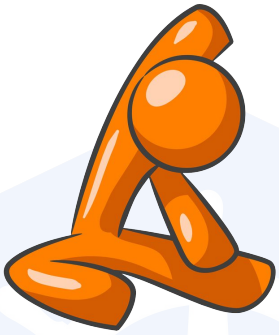
Configuration Objects in Kubernetes



Exercise 4

04-secrets-configmap.md

- Overriding docker image environment variables with deployments



Automatic Pod Recreation (with Helm)

Automatically Roll Deployments When ConfigMaps or Secrets change

Often times configmaps or secrets are injected as configuration files in containers. Depending on the application a restart may be required should those be updated with a subsequent `helm upgrade`, but if the deployment spec itself didn't change the application keeps running with the old configuration resulting in an inconsistent deployment.

The `sha256sum` function can be used to ensure a deployment's annotation section is updated if another file changes:

```
kind: Deployment
spec:
  template:
    metadata:
      annotations:
        checksum/config: {{ include (print $.Template.BasePath "/configmap.yaml") . | sha256sum }}
[...]
```


Persistent Storage

Definition of Storage - Volumes

Why?

- Provide configuration for a container
- Persist data through pod lifecycle
- Share data between containers in a Pod

Mounting Volumes

How

- Volumes are specified in Pod-spec
- Mounted to one or more containers in their Container-spec

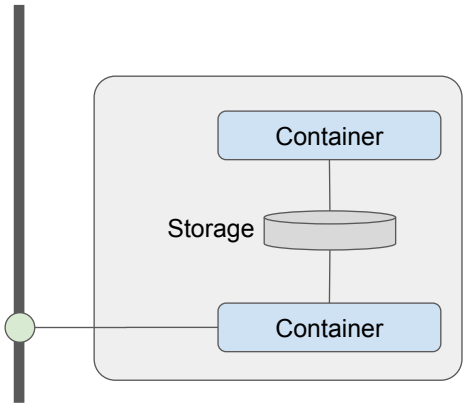
```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  volumes:
  - name: data-volume
    ...
  containers:
  - image: nginx
    name: web-server
    volumeMounts:
    - mountPath: /data
      name: data-volume
```

Sharing Data Between Containers in a Pod

Volume type **emptyDir**

- Only available during Pod lifecycle
- Great for sharing ephemeral data between containers in the same pod

```
...
spec:
  volumes:
  - name: data-volume
    emptyDir: {}
```



Mounting Volumes

Configmap

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  volumes:
  - name: config-volume
    configMap:
      name: special-config
      items:
      - key: special.level
        path: keys
  containers:
  - image: nginx
    name: web-server
    volumeMounts:
    - mountPath: /config
      name: config-volume
```

Volume Types

Generic and Platform Specific Types

- emptyDir, hostPath, local, nfs - generic types
- Secrets, ConfigMap, downwardAPI - accessing Kubernetes resources as volumes
- Cinder, cephfs etc. (many more) - network storage specific
- GCE, AWS, Azure disks - cloud provider integration
- PersistentVolumeClaim - dynamic provisioning

Dynamic Volume Provisioning

PersistentVolume and PersistentVolumeClaim - manual

Admin

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: hostpath
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  reclaimPolicy:
    - Recycle
   mountOptions:
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Dev / User

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-volume
spec:
  resources:
    requests:
      storage: 8Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: “”
```

```
apiVersion: v1
kind: Pod
...
spec:
  volumes:
  - name: data-volume
    persistentVolumeClaim:
      claimName:
        data-volume
```

Dynamic Volume Provisioning

StorageClass and PersistentVolumeClaim - automatic

Admin

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
```

Storage provisioner

Automatically provisioned

```
apiVersion: v1
kind: PersistentVolume
…
spec:
  gcePersistentDisk:
   fsType: ext4
   pdName: gke-cluster-1-3e90xx
```

Dev / User

```
apiVersion: v1
kind: PersistentVolum
metadata:
  name: data-volume
spec:
  resources:
    requests:
      storage: 8Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: “standard”
```

```
apiVersion
kind: Pod
...
spec:
  volumes:
  - name:
    persis
      clai
        da
```

eficode

Storage objects in Kubernetes

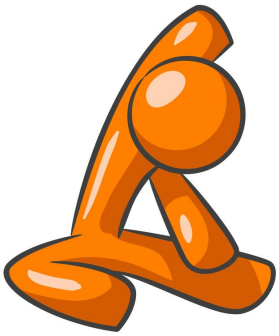
Storage in the cluster with objects:

Pods can reference a Volume of type PersistentVolumeClaim to acquire access to the storage

- Can be mounted to one or more nodes, with different modes
  - ReadWriteOnce - Mounted as **readwrite** on one **Node**
  - ReadOnlyMany - Mounted as **read** on many **Nodes**
  - ReadWriteMany - Mounted as **readwrite** on many **Nodes**
    - RWM is generally not supported by Volume Plugins

eficode

Exercise 5



05-storage.md

- Use the previous nginx:1.9.1 deployment
- Create a PVC using the instructions.
- Update nginx deployment to use this PVC as a mount point for its html directory.
- Deploy this change and test it with curl.

Helpful Task: https://kubernetes.io -> Documentation -> Tasks -> Configure Pods and Containers -> *Configure a Pod to Use a PersistentVolume for Storage*

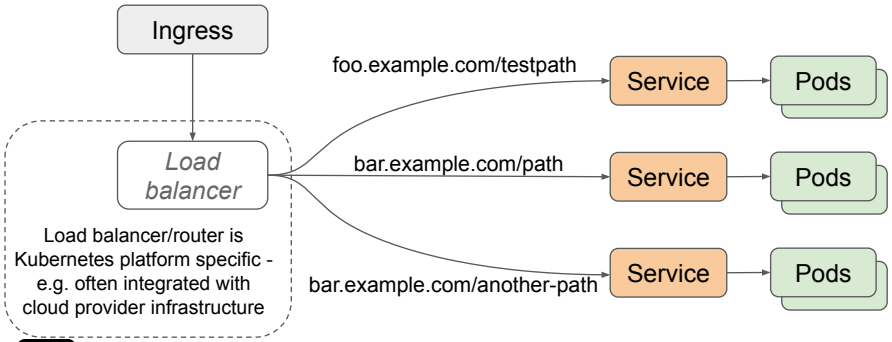
Networking and Ingress

Network Abstraction - L4/Services

- Services are a networking **L4 'virtual IP' abstraction**
  - Cluster **IP address is fixed** while backends can change as number of PODs change
  - **Traffic is load balanced** with several algorithms possible (round-robin, least connections etc.)
- Services are exposed through DNS
  - Don't use IP addresses in your applications, **always use DNS**
  - **DNS search path** reflects service name and namespace
    - <service-name>.<namespace>.svc.cluster.local
    - Access your services using short DNS name <service-name>

Network Abstraction - L7/Ingress

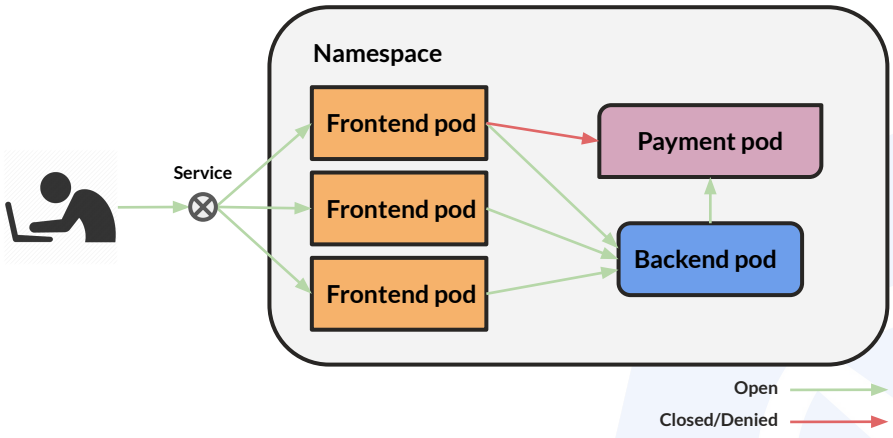
- Ingress provides a networking **L7 HTTP** abstraction
- **Defines** load balancer/routing of L7 traffic




```
apiVersion:
extensions/v1beta1
kind: Ingress
metadata:
  name: web
spec:
  rules:
  - host: foo.example.com
    http:
      paths:
      - path: /testpath
        backend:
          serviceName: web
          servicePort: http
```

Security

Enforcing network policies



NetworkPolicy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: payment-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: payment-gateway
  policyTypes:
  - Ingress
  - Egress
...
```

```
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        project: webshop
    podSelector:
      matchLabels:
        app: backend
  ports:
  - protocol: TCP
    port: 1111
egress:
- to:
  - ipBlock: #some bank ip
      cidr: 1.2.3.4/32
  - ipBlock: #some other bank ip
      cidr: 5.6.7.8/32
  ports:
  - protocol: TCP
    port: 9999
```


POD Security Policies

Enable fine-grained authorization of POD features

- Use of host namespaces (PID, IPC, networking/host-ports)
- Use of volume types (including host path)
- Run container as root vs. non-root
- Read-only filesystem in pod
- Container capabilities (e.g. NET_RAW allows capture packets)
- Priviledge container (e.g. low-level access to host devices through /dev)

POD Security Policies

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
spec:
  allowedCapabilities:
  - SYS_ADMIN
  - NET_ADMIN
  allowedHostPaths:
  - pathPrefix: /tmp
  fsGroup:
    rule: RunAsAny
  hostIPC: false
  hostNetwork: false
  volumes:
  - hostPath
  - configMap
  - secret
```

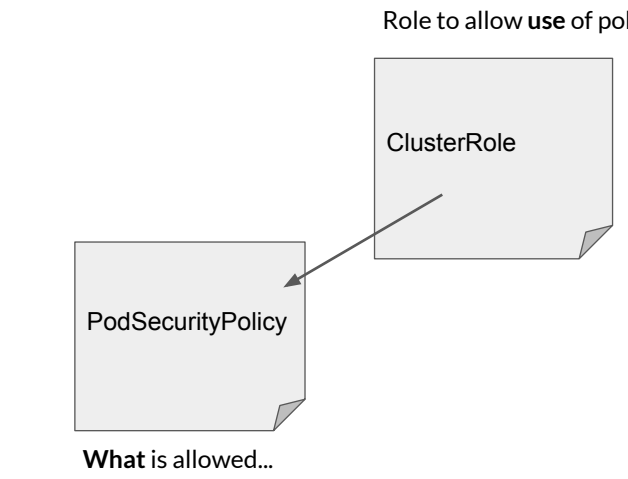
- PSPs are Cluster-scoped - no namespaces
- PSP support needs to be enabled in the cluster
  - One of the many Kubernetes admission-controllers
- Allow-approach: **Default is DENY**
  - There must be a policy that ALLOW use of feature

POD Security Policies

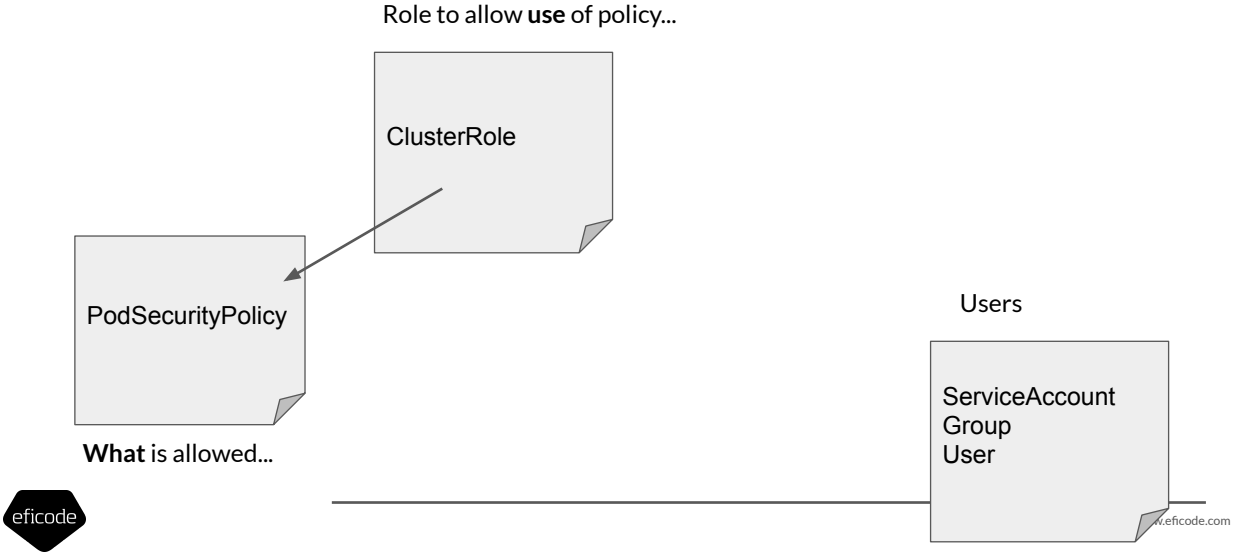
PodSecurityPolicy

What is allowed...

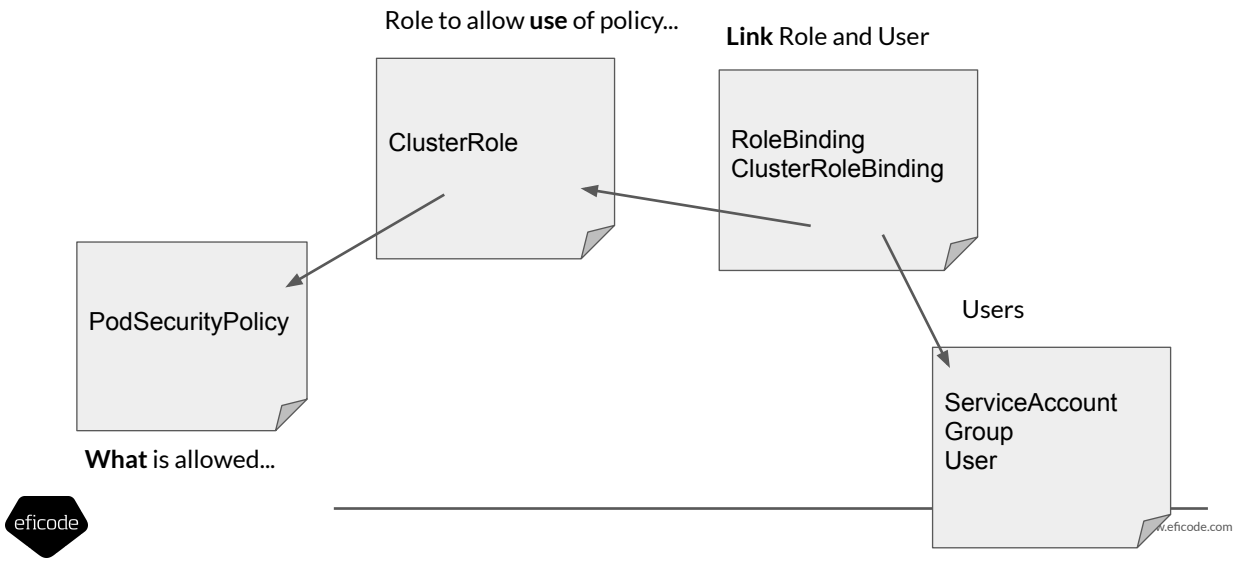
POD Security Policies



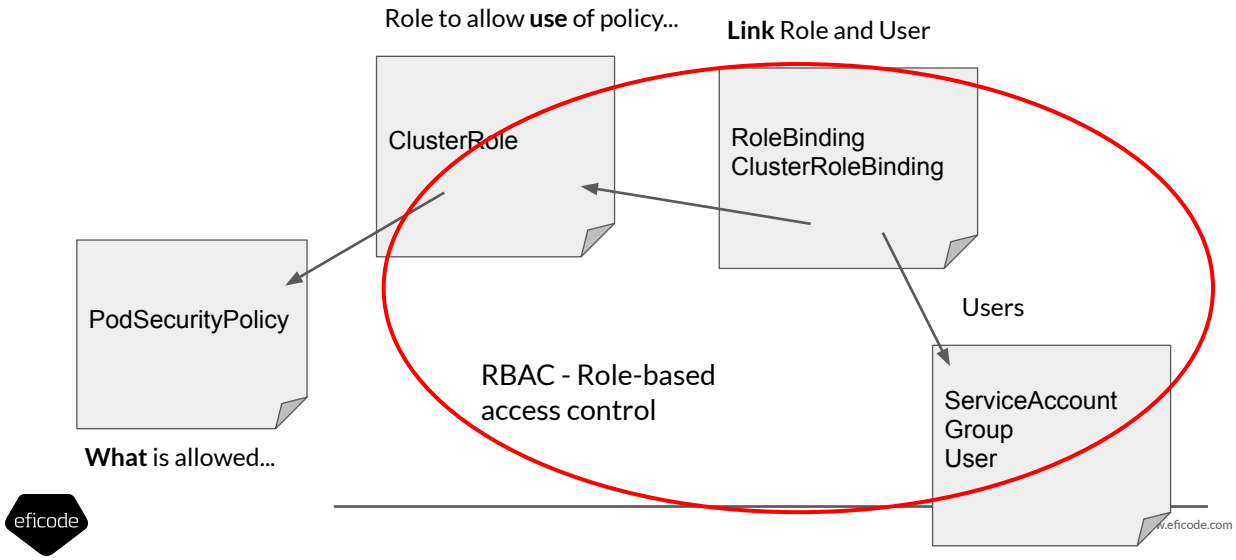
POD Security Policies



POD Security Policies



POD Security Policies



Liveness and Health Checks

Readiness and Health Checks

- Defaults
  - No probe in pod spec imply ready/alive
  - Init-containers imply not ready
- Essential part of the 'declarative approach'

```
...
spec:
  containers:
  - name: my-app
    image: foo
    livenessProbe:
      httpGet:
        path: /
        Port: 1234
      initialDelaySeconds: 5
      periodSeconds: 5
```

```
...
spec:
  containers:
  - name: my-app
    image: foo
    readinessProbe:
      exec:
        command:
        - cat
        - /tmp/ready
      initialDelaySeconds: 5
      periodSeconds: 5
```

Readiness and Health Checks

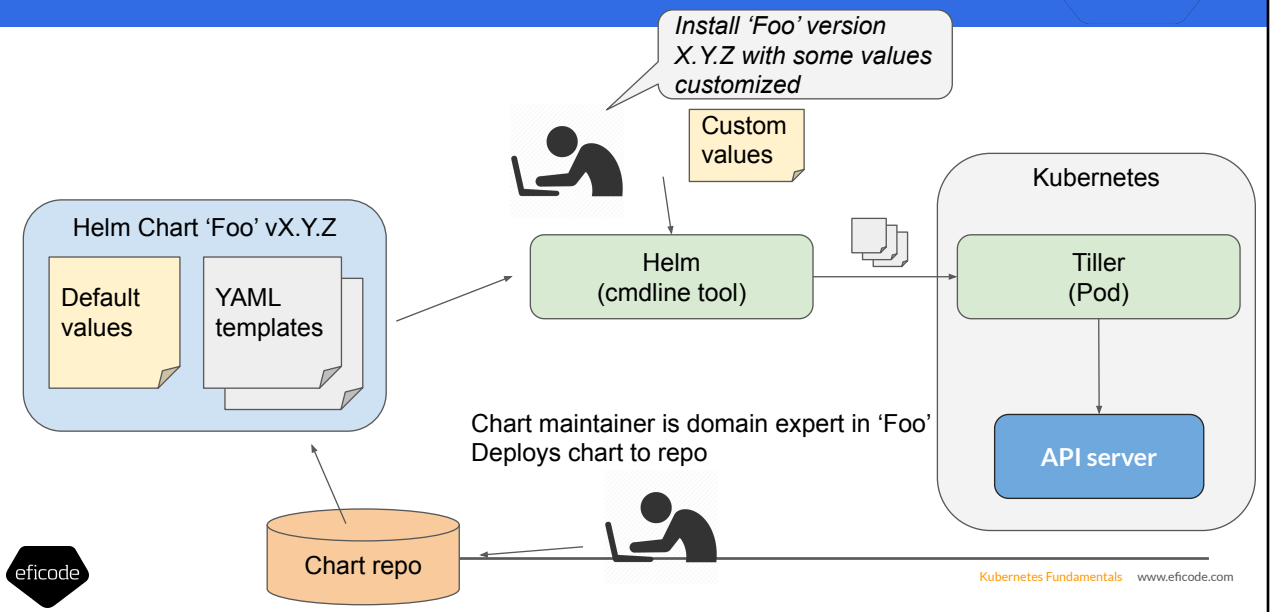
- Liveness probing
  - Container restart depends on health checks
- Readiness probing
  - Pods must be ready before being added as service backends
  - Important in statefulsets scaling
- Probing types
  - HTTP (Return 2xx or 3xx, no authentication)
  - TCP (3-way-handshake indicates success)
  - Exec (return code 0)

```
...
spec:
  containers:
  - name: my-app
    image: foo
    livenessProbe:
      httpGet:
        path: /
        Port: 1234
      initialDelaySeconds: 5
      periodSeconds: 5
```

```
...
spec:
  containers:
  - name: my-app
    image: foo
    readinessProbe:
      exec:
        command:
        - cat
        - /tmp/ready
      initialDelaySeconds: 5
      periodSeconds: 5
```

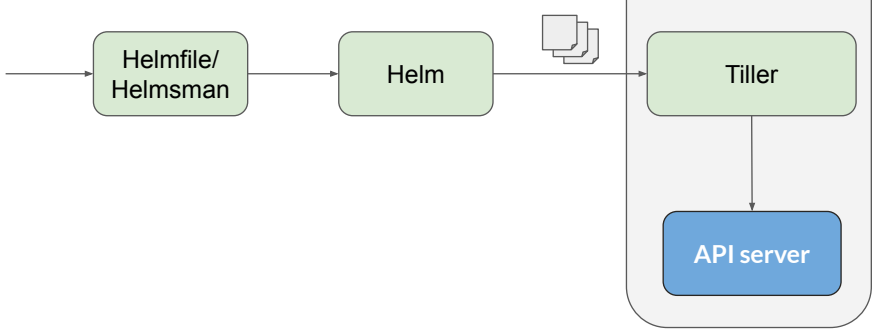

Kubernetes Package Management with Helm

Helm Package Manager

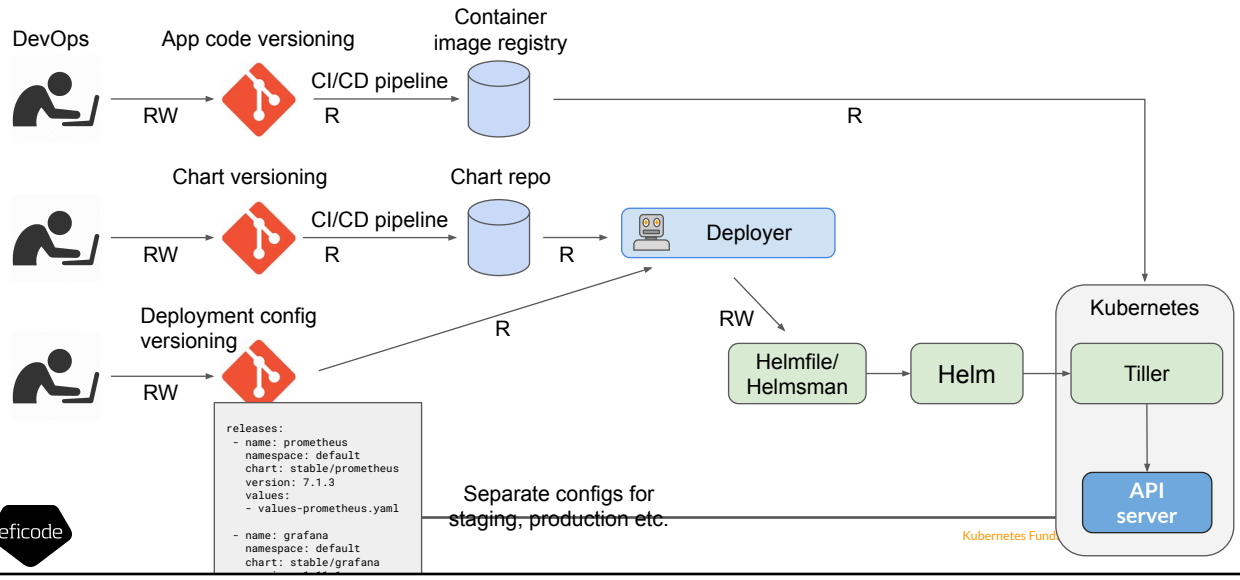


Managing Apps With Helmfile/Helmsman

```
releases:
 - name: prometheus
   namespace: default
   chart: stable/prometheus
   version: 7.1.3
   values:
   - values-prometheus.yaml

 - name: grafana
   namespace: default
   chart: stable/grafana
   version: 1.11.1
   values:
   - values-grafana.yaml
```

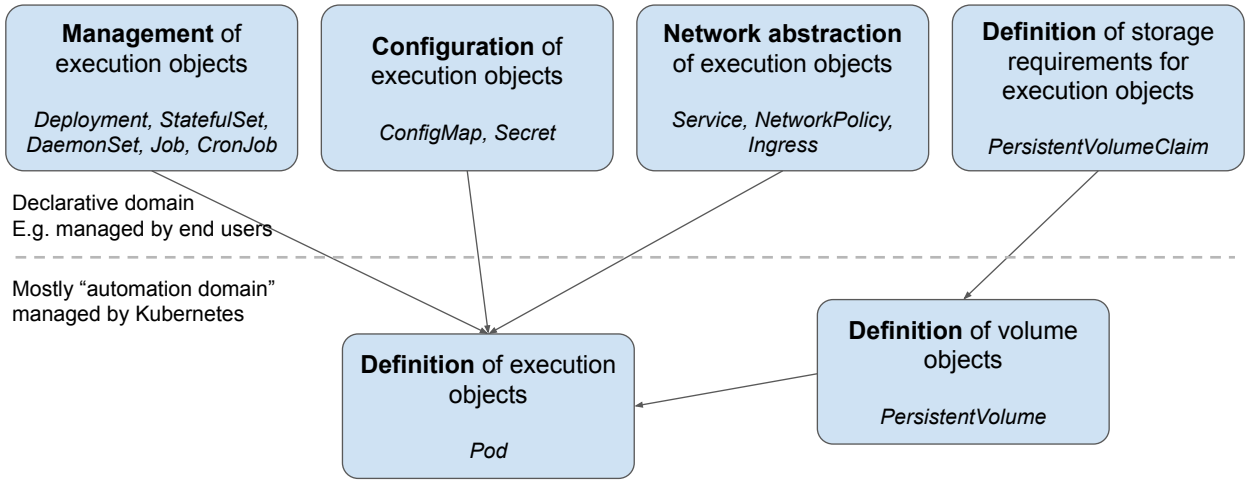


Managing Apps - the Big Picture



Wrap-up / Extras

Remember the Kubernetes Lego Bricks?

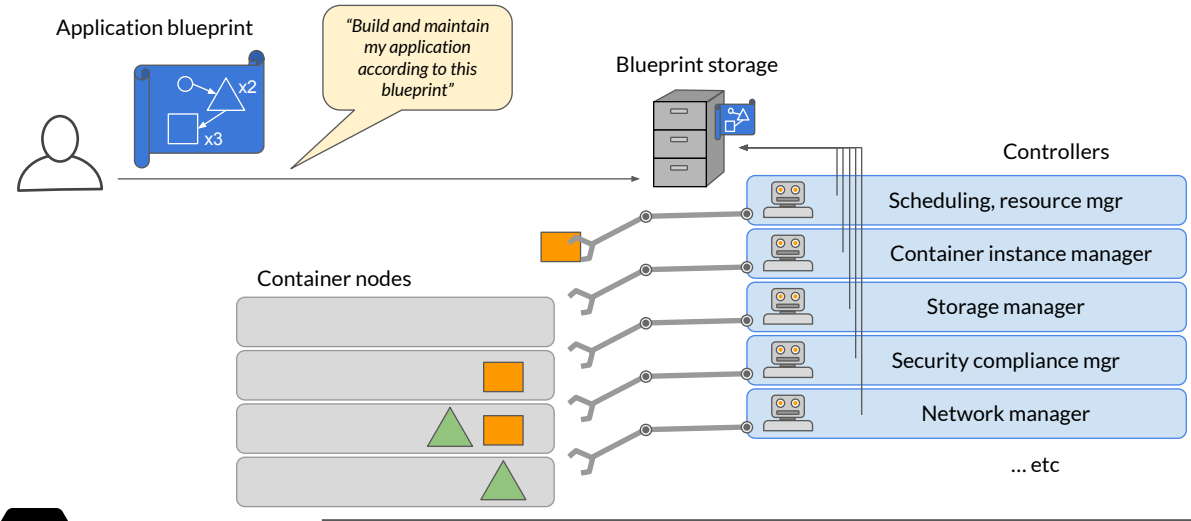


Playing with K8s

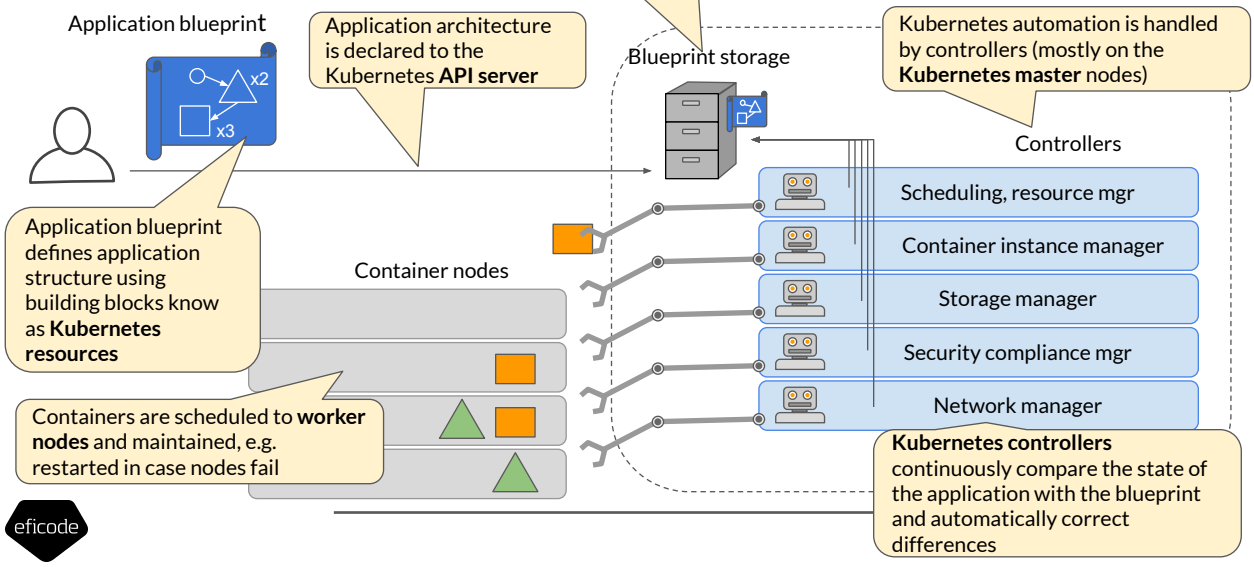
- www.k8s.io - Everything Kubernetes / learning
- www.KataCoda.com - online sandbox / learning
- www.k3s.io - minimal installation

Kubernetes Internals

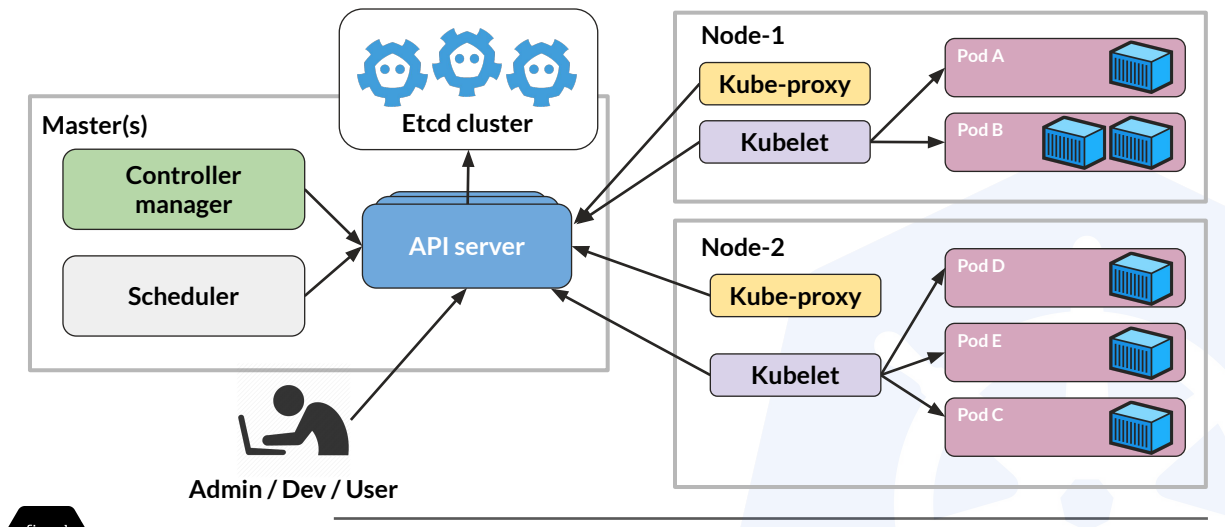
The Short Description of Kubernetes



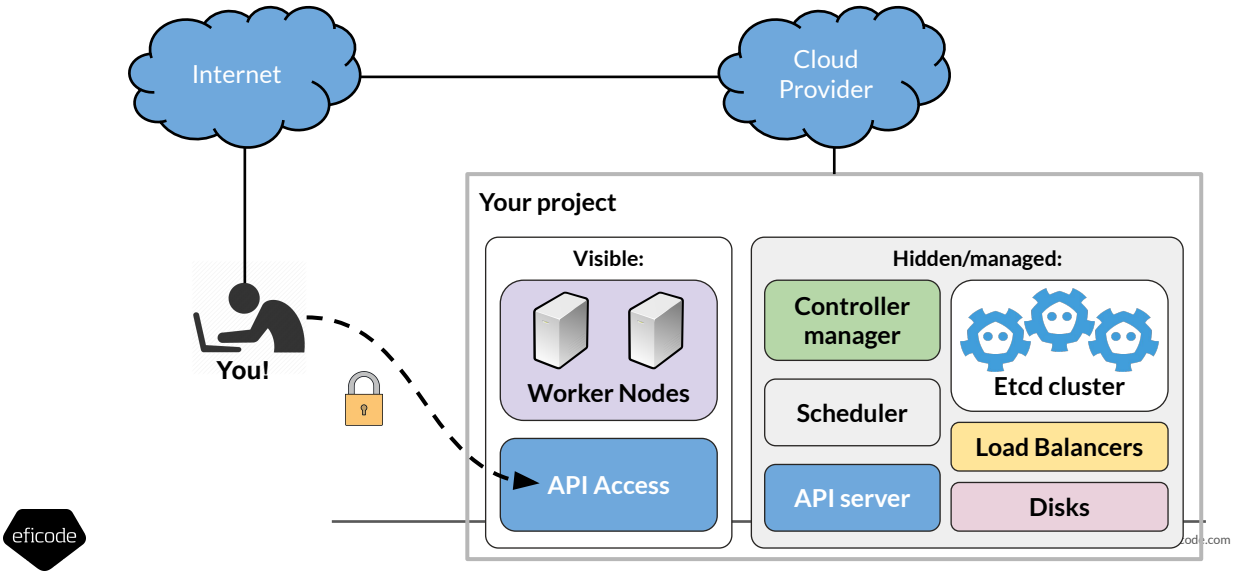
The Short Description o s (2)



Components of Kubernetes

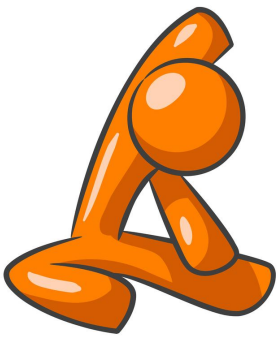


Cluster layout: K8s-aaS (GCP, AWS, ...)



Agenda

- Introduction to Kubernetes
- Namespaces, Pods, and Deployments
- Service Discovery and LoadBalancing
- Rolling out updates
- ConfigMaps and Secrets
- Persistent Storage
- Networking and Ingress
- Helm Package Manager
- Kubernetes Internals




eficode