An Introduction to Kubernetes

Cemal Ünal

- Software Engineer @ Havelsan Inc.

 cemalunal

 cemalunal

 cunal@havelsan.com.tr

Container Orchestration

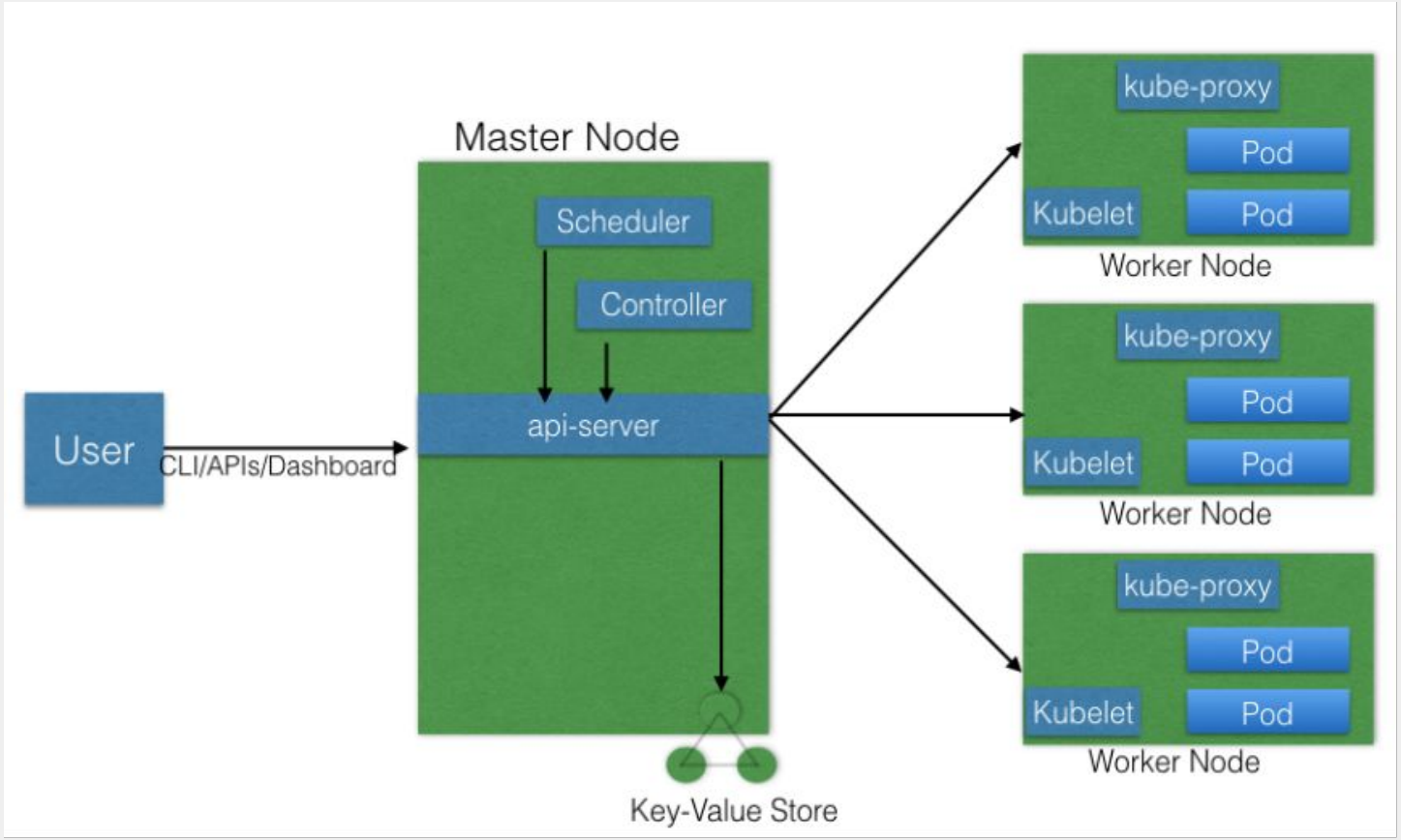
Requirements:

- Fault-tolerance
- On-demand scalability
- Optimal resource usage
- Self healing
- Highly available in case of any failures
- Auto-discovery to automatically discover and communicate with each other
- Seamless updates/rollbacks without any downtime

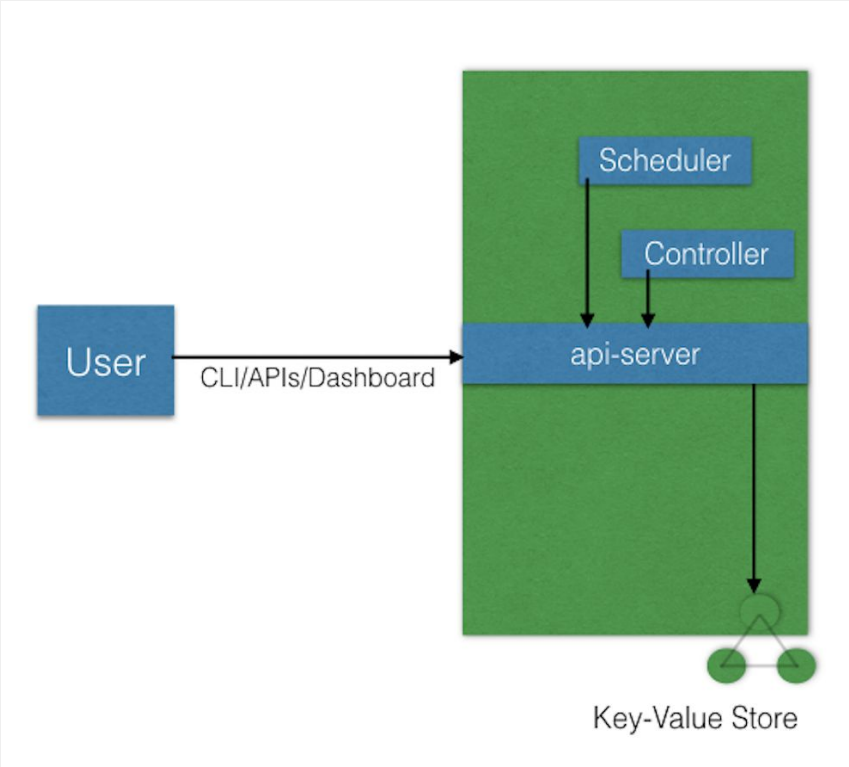
Kubernetes Features

- Self-healing
  - Kills and restarts the containers which do not respond to health checks
- Horizontal scaling
  - Can automatically scale applications based on resource usage like CPU and memory
- Service discovery and Load balancing
  - Groups sets of containers and refers to them via a Domain Name System (DNS). And load-balances requests between these group of containers
- Automated rollouts and rollbacks
  - Can roll out and roll back new versions/configurations of an application, without introducing any downtime

Kubernetes


Master Node
Scheduler
Controller
api-server
User
CLI/APIs/Dashboard
Key-Value Store
kube-proxy
Pod
Kubelet
Pod
Worker Node
kube-proxy
Pod
Kubelet
Pod
Worker Node
kube-proxy
Pod
Kubelet
Pod
Worker Node

Kubernetes - Master Node



- Responsible to manage the cluster
- Components
  - API Server
  - Scheduler
  - Component Manager
  - etcd: store the cluster state

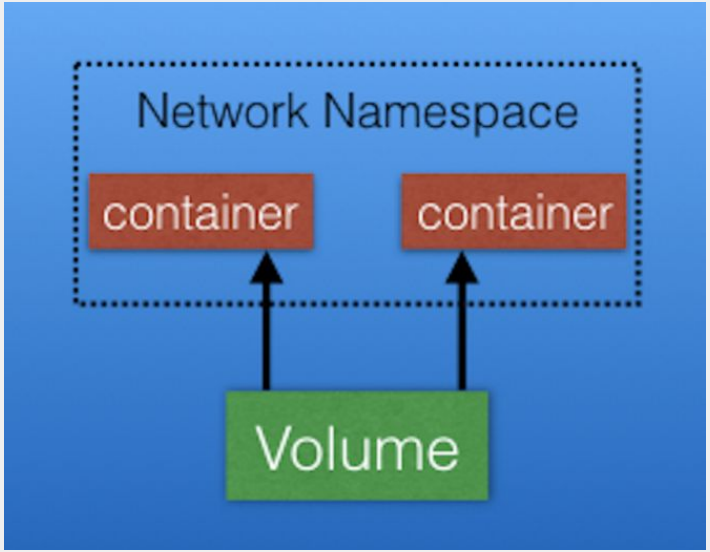
Kubernetes Concepts - Pods

- Smallest and simplest Kubernetes object.
- Represents a single instance of the application.
- They do not have the capability to self-heal by themselves. (Use deployment instead)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.17.0
    ports:
    - containerPort: 80
```

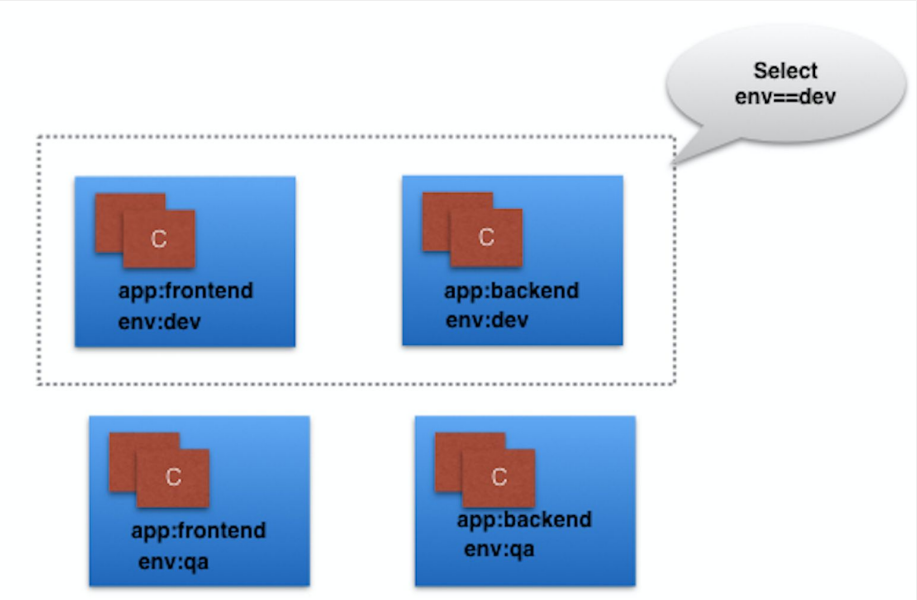
Kubernetes Concepts - Pods

- Are scheduled together on the same host
- Share the same network namespace
- Mount the same external storage (volumes)



Kubernetes Concepts - Labels

- Key-value pairs that can be attached to any Kubernetes object.
- Labels are used to organize and select a subset of objects, based on the requirements in place

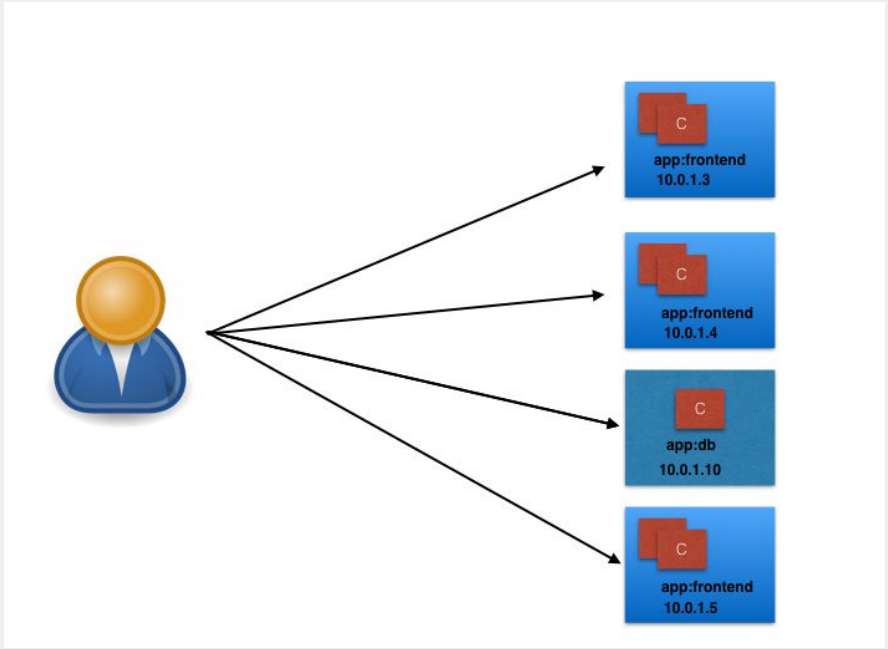


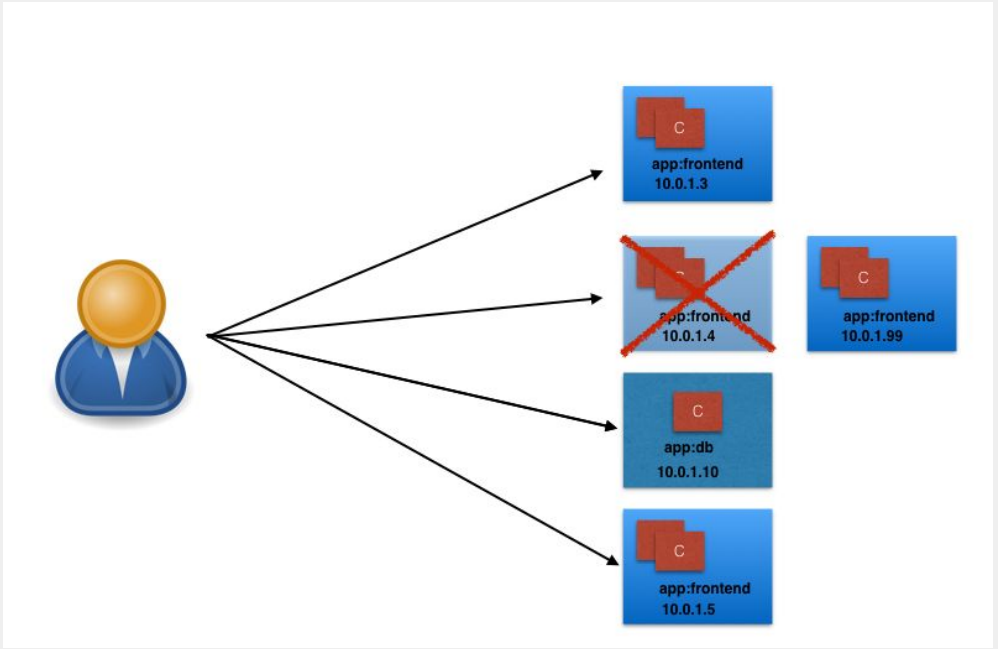
Kubernetes Concepts - Deployment

- Makes sure that the current running number of pods always matches the desired state.
- Provide declarative updates to Pods.

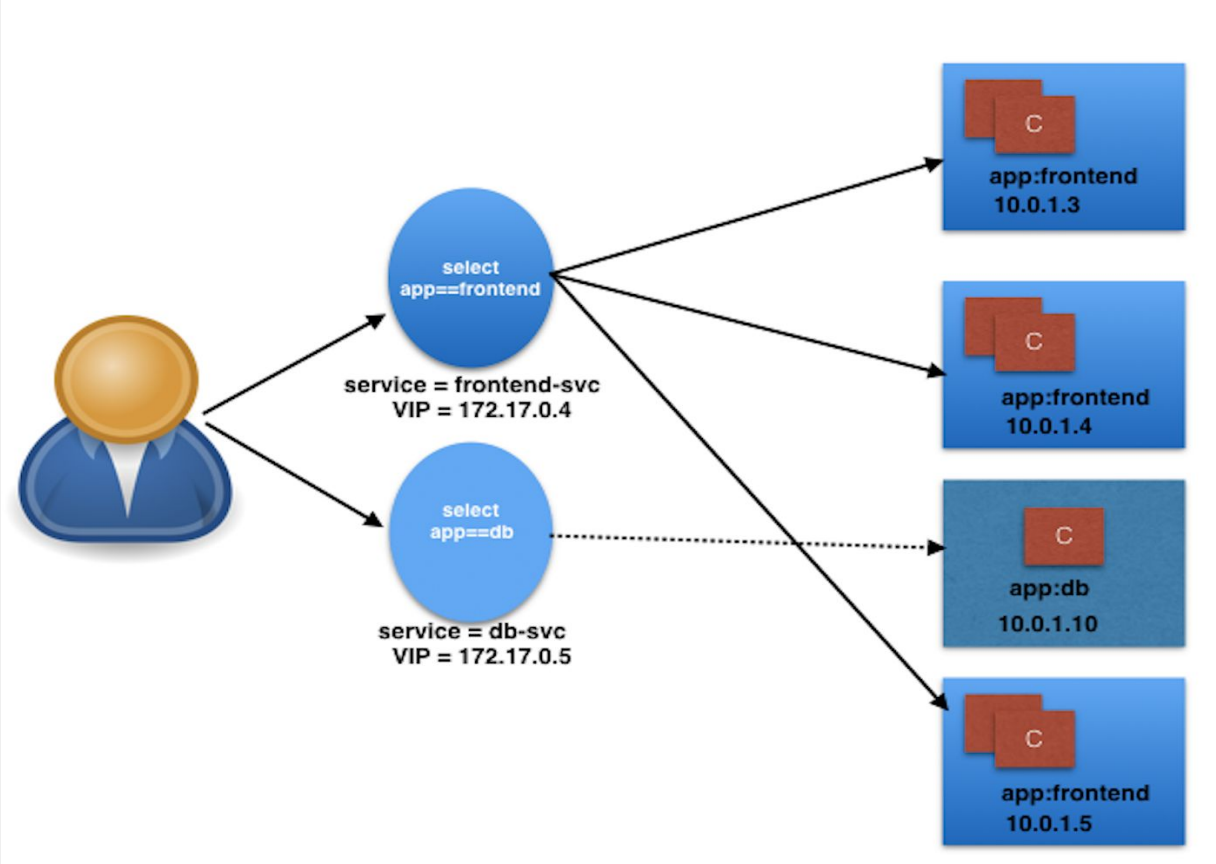
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:1.16.0
        name: nginx
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
        terminationGracePeriodSeconds: 10
```

Kubernetes Concepts - Services





Kubernetes Concepts - Services



Kubernetes Concepts - Services

- Using Services, Pods can be grouped to provide common access points from the external world.

- Service types:
  - Headless
  - ClusterIP
  - NodePort
  - LoadBalancer

```yaml
apiVersion: v1
kind: Service
metadata:
  name: demo-backend-deployment-service
  namespace: kubernetes-demo
  labels:
    app: demo-backend-deployment
spec:
  type: ClusterIP
  ports:
    - port: 80
  selector:
    app: demo-backend-deployment
```

Kubernetes Concepts - Services

- Service types:
  - Headless
  - ClusterIP
    - Default service type
    - Its IP address only accessible within the cluster.
  - NodePort
    - ClusterIP + A port from the range 30000-32767 is mapped to the respective Service, from all the worker nodes
  - LoadBalancer
    - NodePort + ClusterIP + an external load bolancer will route to them
    - Service is exposed externally using the underlying infrastructure's load balancer feature.

Kubernetes Concepts - ConfigMap

- Allow us to decouple the configuration details from the container image
- Using ConfigMaps, we can pass configuration details as key-value pairs, which can be later consumed by Pods

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: frontend-cm
  namespace: k8s-demo
data:
  REACT_APP_BACKEND_URI: "http://10.0.0.0"
```

```yaml
spec:
  containers:
    - image: cunal/demo-frontend:v0.0.1
      imagePullPolicy: IfNotPresent
      envFrom:
      - configMapRef:
          name: simple-frontend-cm
```

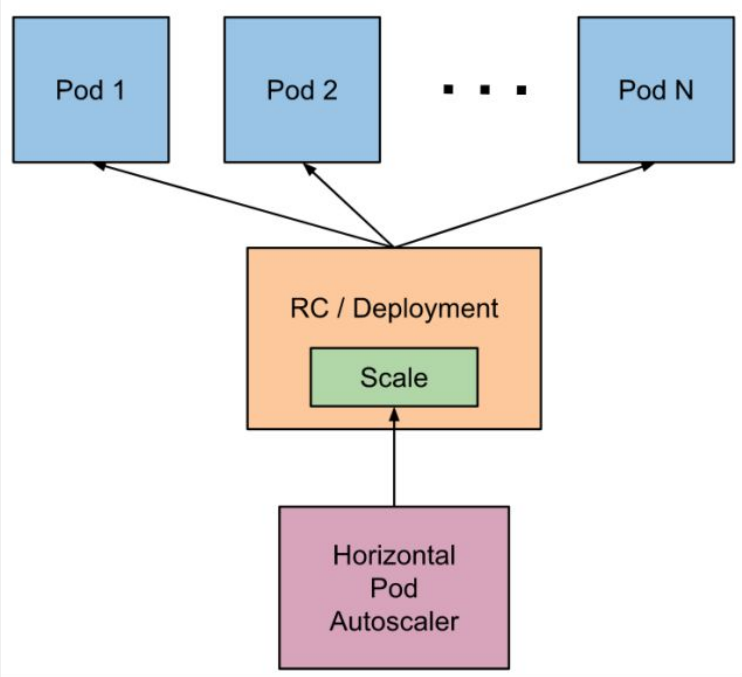
Kubernetes Concepts - Namespaces

- Use it to segment the services into manageable chunks.
- Namespaces can make Kubernetes a lot more manageable and gives us increased control, security and flexibility.

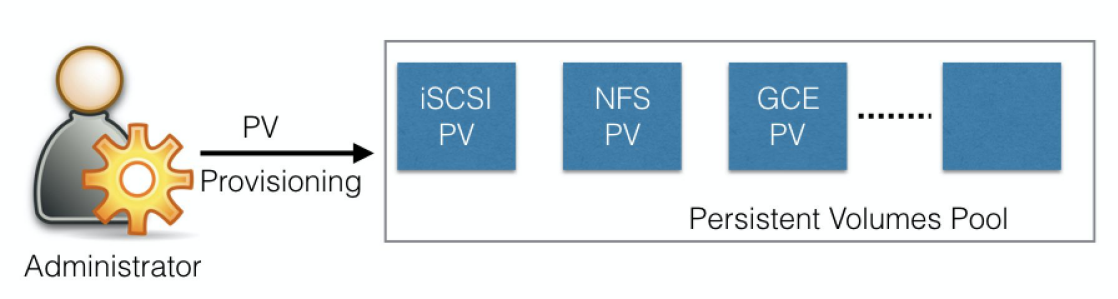
```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: demo
```

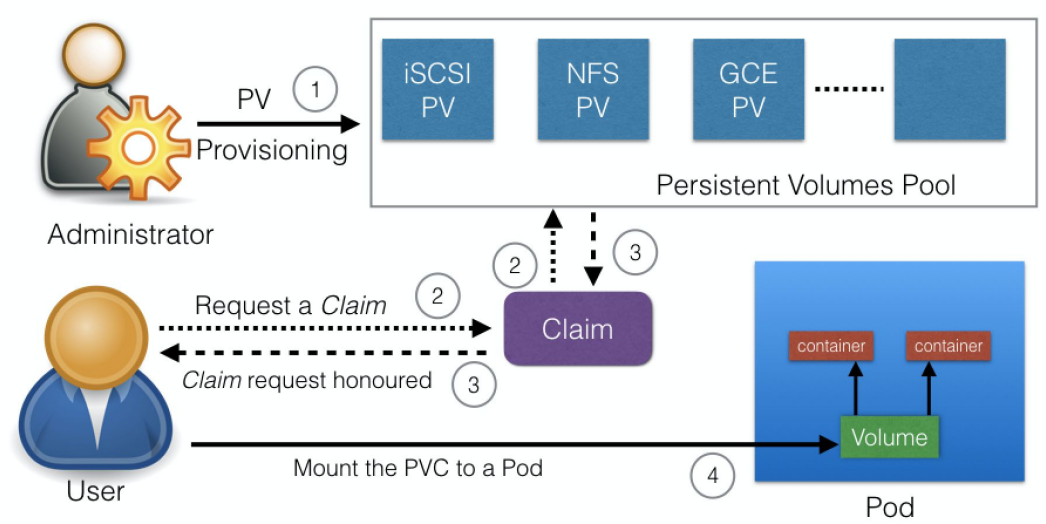
Horizontal Pod Autoscaler

- Automatically scales the number of pods in a deployment.
- Autoscale is based on observed CPU utilization or some custom metrics.



Volume Management





DEMO

Role Based Access Control (RBAC)

- Mostly useful for
  - Giving access to pods calling Kubernetes API (with Kubernetes Service Account)
  - Giving fine-grained access to people/groups calling Kubernetes API

ClusterRole	A preset of capabilities
Role	ClusterRole, but namespace-scoped
ClusterRoleBinding	Give permission of a ClusterRole to users
RoleBinding	ClusterRoleBinding, but namespace-scoped

Role Based Access Control (RBAC)

Example **Role + Binding** only allows **view of deployments** in the **development** namespace:

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: development
  name: deployment-viewer
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "replicasets", "pods"]
  verbs: ["get", "list"]
```

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: deployment-viewer-binding
  namespace: development
subjects:
- kind: User
  name: cemal
  apiGroup: ""
roleRef:
  kind: Role
  name: deployment-viewer
  apiGroup: ""
```

Resource Quotas

Limits **memory/cpu/storage** that pods can use, and how many objects of each type **(pods, load balancers, ConfigMaps)** on a **per-namespace** basis

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: my-namespace
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
  namespace: my-namespace
spec:
  hard:
    configmaps: "10"
    pods: "4"
    deployments: "20"
    services: "10"
    services.loadbalancers: "2"
```

Assigning Pods to Nodes

```
$ kubectl label nodes node01 disktype=ssd
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

THANKS!