Introduction to Kubernetes

Some facts about Kubernetes

- Is an open source container orchestration system

- Originally developed by Google - announced mid 2014, first release in 2015

- The project is maintained by the Cloud Native Computing Foundation

- Design was influenced by Google's internal cluster manager - Borg

- First name pitch was "Seven of Nine", an ex-Borg character from Star Trek
  - This is where the seven-spoke wheel comes from

- The name Kubernetes originates from Greek, meaning helmsman or pilot
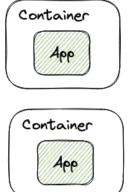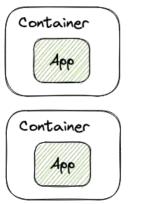
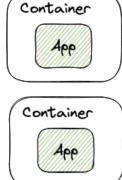kubernetes

What problems does Kubernetes solve?

- Increased use of Microservices over Monoliths

- Lead to increased usage of containers

- Demand for formalized way of managing multiple containers

Kubernetes offers

- A rich extendable API configured through YAML, JSON or CLI

- High availability through safe rollout/rollback and self-healing

- Capabilities for scaling in and out

- Disaster recover

kubernetes

Architectural Overview

Basic Architecture

At least one master node
The control plane of the cluster
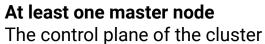
Multiple worker nodes
Usually referred to as *nodes*
Running the applications workload

Node = virtual or physical machine

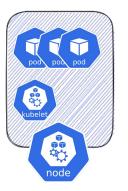# Basic Architecture

A kubelet process running on each node
The primary node agent

Containers running inside pods
Containerized applications running inside pods are deployed throughout worker nodes
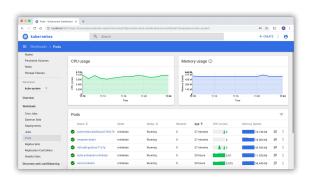
Node = virtual or physical machine

Basic Architecture - Control Plane

API Server
The API Server services REST operations and provides the frontend to the cluster's shared state

kubernetes

Basic Architecture - Control Plane

API Server
The API Server services REST operations and provides the frontend to the cluster's shared state

Controller Manager
A control loop that watching shared state and makes changes attempting to move the current state towards the desired state
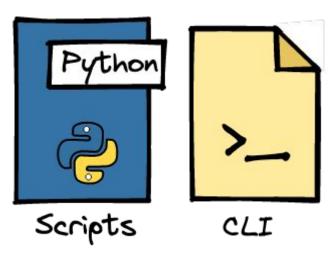
kubernetes

Basic Architecture - Control Plane

API Server
The API Server services REST operations and provides the frontend to the cluster's shared state

Controller Manager
A control loop that watching shared state and makes changes attempting to move the current state towards the desired state

Scheduler
Ensures Pod placement by watching for newly created Pods with no assigned node, and selects a node for them to run on

kubernetes

Basic Architecture - Scheduling Pods

40% Resources Used

60% Resources Used
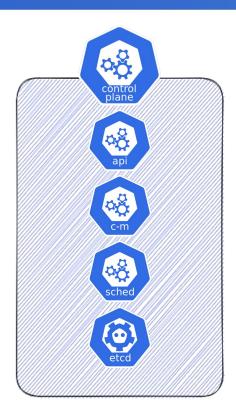
Resource Requests
Determine how many resources a container in a Pod needs e.g. CPU and Memory

In this case the pod will be scheduled on the node to the left

Basic Architecture - Control Plane

API Server
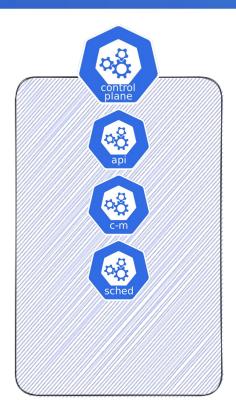The API Server services REST operations and provides the frontend to the cluster's shared state

Controlle Manager
A control loop that watching shared state and makes changes attempting to move the current state towards the desired state
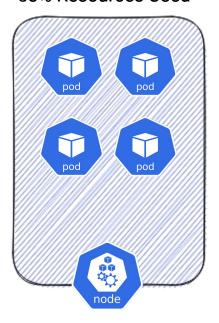
Scheduler
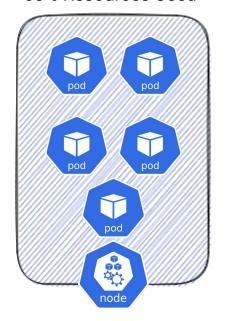Ensures Pod placement by watching for newly created Pods with no assigned node, and selects a node for them to run on
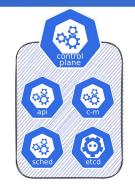
etcd
Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data

kubernetes

Basic Architecture - Virtual Network
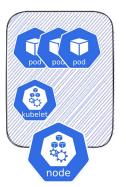
Enable communication between master and worker nodes

Spanning across all nodes in the cluster

Enabling the cluster to function as a single powerful machine that has the sum of all resources across all nodes

Basic Kubernetes Concepts

Basic Kubernetes Concepts

Workloads

Pod

ReplicaSet

Deployment

Job

StatefulSet

CronJob

DaemonSet

Networking and Load Balancing

Ingress

Service

Configuration

ConfigMap

Secret

Storage

Volume

Persistent Volume

Persistent Volume Claim

Storage Class

https://kubernetes.io/docs/concepts/

kubernetes

Nodes and Pods

Web App

15.211.29.1

PostgreSQL

15.211.29.3

node

167.116.235.45

pod
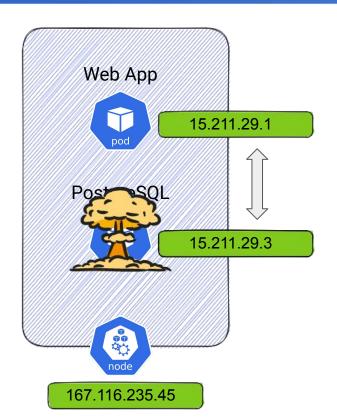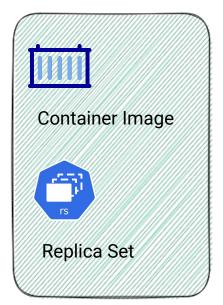
- Pods are the smallest unit in Kubernetes
- A pod can contain one or more containers
- Serves as an abstraction over one or more containers
- Each pod gets its own internal IP address
  - All containers in a pod gets that IP
- **Pods are ephemeral!**
- And gets a new IP address when restarted

kubernetes

Deployments and ReplicaSet

Container Image

Replica Set
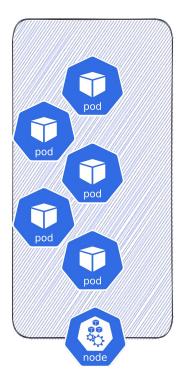
- While you can deploy pods separately, you should use Deployment manifests to deploy workloads
- Deployment serves as a blueprint for pods and replica sets by describing a desired state in the Deployment
- Because of the desired state, Kubernetes will make sure to move the deployment towards the desired state
- **Deployments are meant for StateLESS applications**

kubernetes

Deployments and ReplicaSet - Example

We have a 2 node cluster

Deployment specifies:

Container Image: nginx:1.14.2
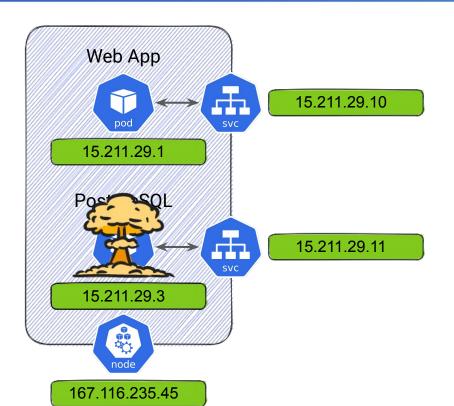
ReplicaSet: 5

This results in 5 replicas of the pod running the Nginx container distributed between the 2 nodes depending on their resources available.

Service and Ingress

Web App

15.211.29.10

15.211.29.1

PostgreSQL

15.211.29.11

15.211.29.3

node

167.116.235.45
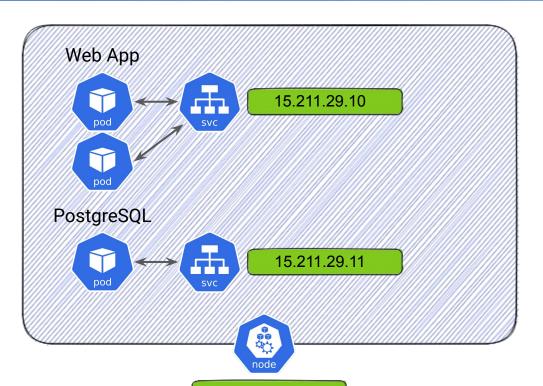
- Offers a permanent IP address
- The lifecycle of the pod and service are disjoined
- When a pod is recycled, the IP of the service stays the same
- Meaning that the endpoint stays the same even though the pod has a new IP

kubernetes

Service and Ingress - one way of exposing workloads on the Internet

Web App

15.211.29.10

PostgreSQL

15.211.29.11

node

167.116.235.45
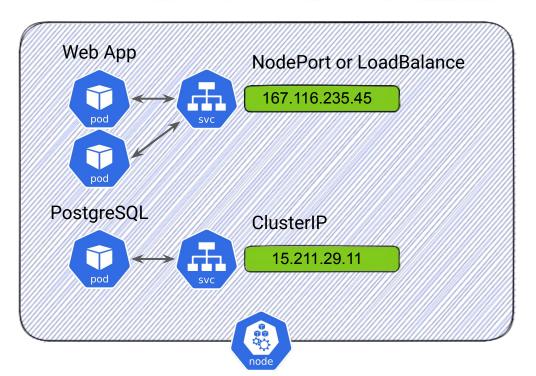
- One Deployment for our stateless web app with a ReplicaSet of 2
- One StatefulSet for our stateful database
- Both running on a single node with a Service in front of them

kubernetes

Service and Ingress - one way of exposing workloads on the Internet

Web App

pod ⟷ svc

NodePort or LoadBalance

167.116.235.45

pod

PostgreSQL

pod ⟷ svc

ClusterIP

15.211.29.11

node

167.116.235.45

There are multiple types of Service
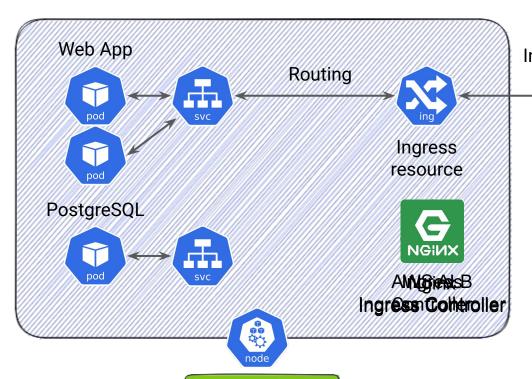
- **ClusterIP**: Exposes the Service on a cluster-internal IP
- **NodePort**: Exposes the Service on each Node's IP at a static port
- **LoadBalancer**: Exposes the Service externally usually to a Cloud provider load balancer

We would like http://web-app.com
Instead of http://167.116.235.45:32647

kubernetes

Service and Ingress - one way of exposing workloads on the Internet

Web App

pod

pod

svc

Routing

ing

Ingress resource

Ingress Managed
Load Balancer

NGINX

Nginx Load
Balancer
Application
Load Balancer

PostgreSQL

pod

svc

NGINX

AWS ALB
Nginx
Ingress Controller
Oss Controller

node

167.116.235.45

To expose our web app on a URL

- We need an Ingress resource

- Ingress resource does not do
  anything on its own

- We also need an Ingress Controller

- Many different controllers exist

kubernetes

Summary of what we have covered so far

Workloads

Pod

ReplicaSet

Deployment

Job

StatefulSet

CronJob

DaemonSet

Networking and Load Balancing

Ingress

Service

Configuration

ConfigMap

Secret

Storage

Volume

Persistent Volume

Persistent Volume Claim

Storage Class

https://kubernetes.io/docs/concepts/

kubernetes

StatefulSet and Volumes
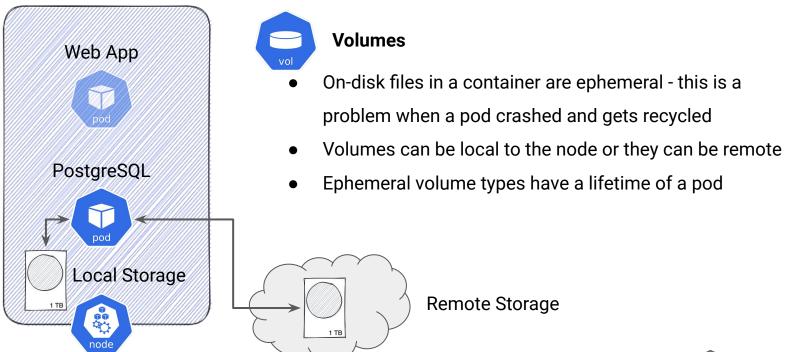
StatefulSet

- Are similar to Deployments but provides guarantees about the ordering, sequence and uniqueness of these Pods
- Valuable for applications that one or more of the following:
  - Stable, unique network identifiers
  - Stable, persistent storage
  - Ordered, graceful deployment and scaling
  - Ordered, automated rolling updates

Web App

pod

PostgreSQL

pod

node

kubernetes

StatefulSet and Volumes

Volumes

- On-disk files in a container are ephemeral - this is a problem when a pod crashed and gets recycled
- Volumes can be local to the node or they can be remote
- Ephemeral volume types have a lifetime of a pod

Web App

PostgreSQL

Local Storage

1 TB

node

Remote Storage

1 TB

kubernetes
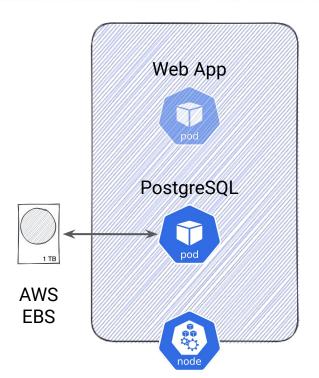
StatefulSet and Volumes - PV & PVC

Persistent Volumes

- Persistent volumes exist beyond the lifetime of a pod
- Retain policy allows the volume to exist even when no pods are claiming it (using it)
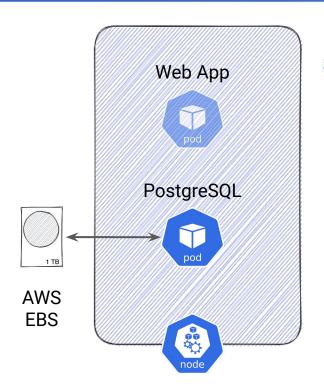
Persistent Volume Claims

- Allow a pod to consume abstract storage
- Claims can request specific size and access modes (ReadWriteOnce, ReadOnlyMany or ReadWriteMany)
- Pods can use PVC as its volume

Web App

PostgreSQL

1 TB

AWS EBS

pv

pvc

pod

pod

node

kubernetes

StatefulSet and Volumes - Storage Class
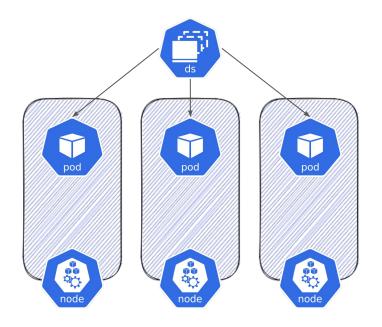
Storage Class

- Provides a way to describe the "classes" of storage that is offered

- Different classes might map to quality-of-service levels, or to backup policies, performance, durability, etc.

- A Persistent Volume (PV) is provisioned using a Storage Class (SC)

Web App

pod

PostgreSQL

pod

1 TB

AWS EBS

node

kubernetes

DaemonSets

DaemonSets

- Ensures that all (or some) nodes run a copy of a pod
- As nodes are added to the cluster, pods are added to them
- Typical use cases are:
  - running a cluster storage daemon on every node
  - running a logs collection daemon on every node
  - running a monitoring daemon on every node
  - running an agent to collect telemetry from apps

kubernetes

Job & CronJob

Job

- A job creates one or more pods to carry out a job
- Continues to retry execution of the pods until a specified number of them successfully terminate
- Jobs can be used to run multiple Pods in parallel
- Pods are not deleted after a job completes
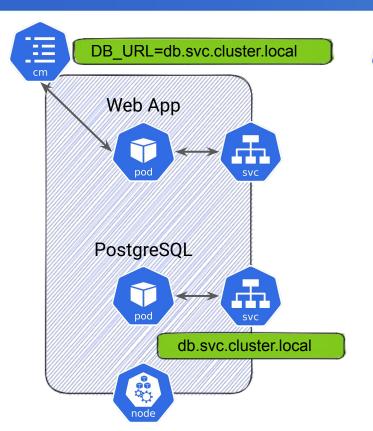
CronJob

- Creates Jobs on a repeating schedule
- Is meant for performing regular scheduled actions
- Follows the Cron format

kubernetes

ConfigMaps & Secrets

DB_URL=db.svc.cluster.local

Web App

pod ↔ svc

PostgreSQL

pod ↔ svc

db.svc.cluster.local

node

ConfigMap

- The pods can communicate with each other using a service
- But, the Web App needs a way to store the hostname of the database
  - If we build the database hostname into the image, we would have to rebuild and redeploy in order to change it

- ConfigMap offers a way to store external configuration for your application

kubernetes

ConfigMaps & Secrets

DB_USER=db-user
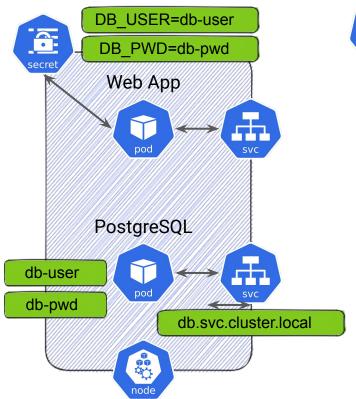
DB_PWD=db-pwd

Web App

PostgreSQL

db-user

db-pwd

db.svc.cluster.local

Secret

- Secrets are meant for storing secret data, e.g. credentials

- Is stored in base64 encoded *and unencrypted by default*

- To safely use Secrets, take at least the following steps

  - Enable Encryption at Rest for Secrets

  - Enable or configure RBAC rules with least-privilege access to Secrets

  - Restrict Secret access to specific containers

  - Consider using external Secret store providers

kubernetes

Onto the demo!

Setup

For the demo, we will use:

- Docker desktop, or some other container runtime
  - For this demo I am using **Colima** (https://github.com/abiosoft/colima)
- A local Kubernetes platform
  - I am using **Kind** (https://kind.sigs.k8s.io/docs/user/quick-start/)
  - To create a Kind cluster, run the following command

    ```
    kind create cluster --kubeconfig ~/.kube/config
    ```

  - Another popular local Kubernetes platform is Minikube, but multiple others exist
- We will need **kubectl**, a CLI tool for communicating with Kubernetes API server
  - Installation instructions can be found here: https://kubernetes.io/docs/tasks/tools/
- (Optional) I am using k9s to get nice and easy navigation between resources.

kubernetes

Applying a single pod

```
> kubectl apply -f pod.yaml
> kubectl get pods -o wide
```
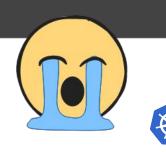
Looking at the details of the pod, we will see that it uses port 80

```
> kubectl describe pod nginx
```

But we cannot access it from a browser? http://localhost:80
Remember that Kubernetes Services are used for exposing workloads!
We can however punch a hole to the pod using `port-forward`

```
> kubectl port-forward nginx 8080:80
```

Let's delete the pod and see what happens!
It doesn't come up again….
Remember Deployments and ReplicaSet tells Kubernetes what the desired state is.

Applying a Deployment

```
> kubectl apply -f deployment.yaml
```

Looking at the details of the Deployment, we can see the desired state is 3 replicase

```
> kubectl describe deployment nginx-deployment
Replicas: 3 desired | 3 updated | 3 total | 3 available | 0 unavailable
```

We can also see the Deployments rolling update strategy

```
RollingUpdateStrategy:   25% max unavailable, 25% max surge
```

This means that max 25% of pods are unavailable during update and max update rate is 25%
Only `PodSpec` trickers an update, others such as scaling does not trigger an update!

Applying a Deployment

Let's scale the Deployment to 10 replicas by editing the deployment.yaml file and applying it. We will see the Deployment scales up the number of pods to 10 but does not follow the `RollingUpdateStrategy`

```
open deployment.yaml
Change replicas: 3 to 10 apply the change
> kubectl apply -f deployment.yaml
```

Now, let's change the image to use nginx:1.23.3. We will see that the 25% max and surge now applies to the rolling update. This is to ensure availability when deploying changes to our applications.

```
open deployment.yaml
Change image: nginx:1.14.2 to nginx:1.23.3 apply the change
> kubectl apply -f deployment.yaml
```

Let's try to be destructive by randomly deleting pods in the Deployment.
We will see that because the Deployment tells Kubernetes how the desired state
looks, it will always try to move the deployment towards that state and simply starts a new pod.
But notice how the pod gets assigned a new IP address!

Applying a Service and Deployment - ClusterIP

Next, we will add a Service to our Deployment.

```
> kubectl apply -f service_and_deployment.yaml
```

We now have a Service and a selector `app: nginx` that connects the Service and Deployment. Let's take a look at the Service.

```
> kubectl describe service svc-nginx
```

We see that the Service is of type. Remember this means the Service is only visible within the cluster network and not exposed internally.

```
Type: ClusterIP
```

However, we can port-forward to the Service. We will notice we hit the same pod each time we refresh as this is not a load balancing Service.

```
> kubectl port-forward service/svc-nginx 8080:80
```

Applying a Service and Deployment - NodePort

Let's change the Service to use NodePort to expose the Deployment on the Node's port.

```
> open service_and_deployment_nodeport.yaml
```

We can see that we have specified `type: NodePort` which will expose the port on the Nodes IP address.

```
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30000
```

Let's apply the manifest and see the NodePort service come up.

```
> kubectl apply -f service_and_deployment_nodeport.yaml
```

The Service is now reachable on http://localhost:30000/ though still not load balancing.

Applying an Ingress & Nginx Ingress Controller

Let's see if we can get proper exposure with load balancing ingress.
Remember that an Ingress resources doesn't do anything by itself! We need an ingress controller so we will install one from Nginx - but many others exist.

```
> kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
```

This installs the nginx ingress controller along with a bunch of other resources.
Next, we will apply our Deployment, Service, and Ingress resource.

```
> kubectl apply -f service_and_deployment_ingress.yaml
```

Now, when we open http://localhost/ we will see that we get a response from our deployed pods and that it is changing (load balancing) per request.

Interactive Tutorial

If you don't want to install the tooling and a local Kubernetes cluster, you can find an interactive tutorial on the link below. The tutorial is using Minikube instead of Kind, but the kubectl commands are all the same.

[Interactive Kubernetes Tutorial](#)

kubernetes

Get in touch

Jimmi Kristensen

Principal Engineer
Github: https://github.com/jimmikristensen
LinkedIn: https://www.linkedin.com/in/jimmikristensen/
Twitter: @jimmibk