Deploy Kubernetes on Azure, AWS and GCP with Terraform using Azure DevOps

hometrack

PART OF Zoopla

Agenda

1. About me
2. Hometrack / Zoopla
3. What Is Kubernetes? Why?
4. Kubernetes Components
5. Kubernetes Consepts
6. AKS Virtual Node
7. DEMO
8. Q&A

Osman Sahin

Senior Platform Engineer

Hometrack is the market leader for AVM services to financial institutions in the UK and NLD

Assets: Comprehensive property data and battle tested valuation model

10bn+ data points

Land/plot	Property attributes
Pricing history/ val	Services and amenities
Community and economy	Financial and risk

Benchmark valuation model

- Most accurate AVM on the market
- History over two downturns
- Cross market coverage
- Industry benchmark

Products: Catalogue of valuation related solutions

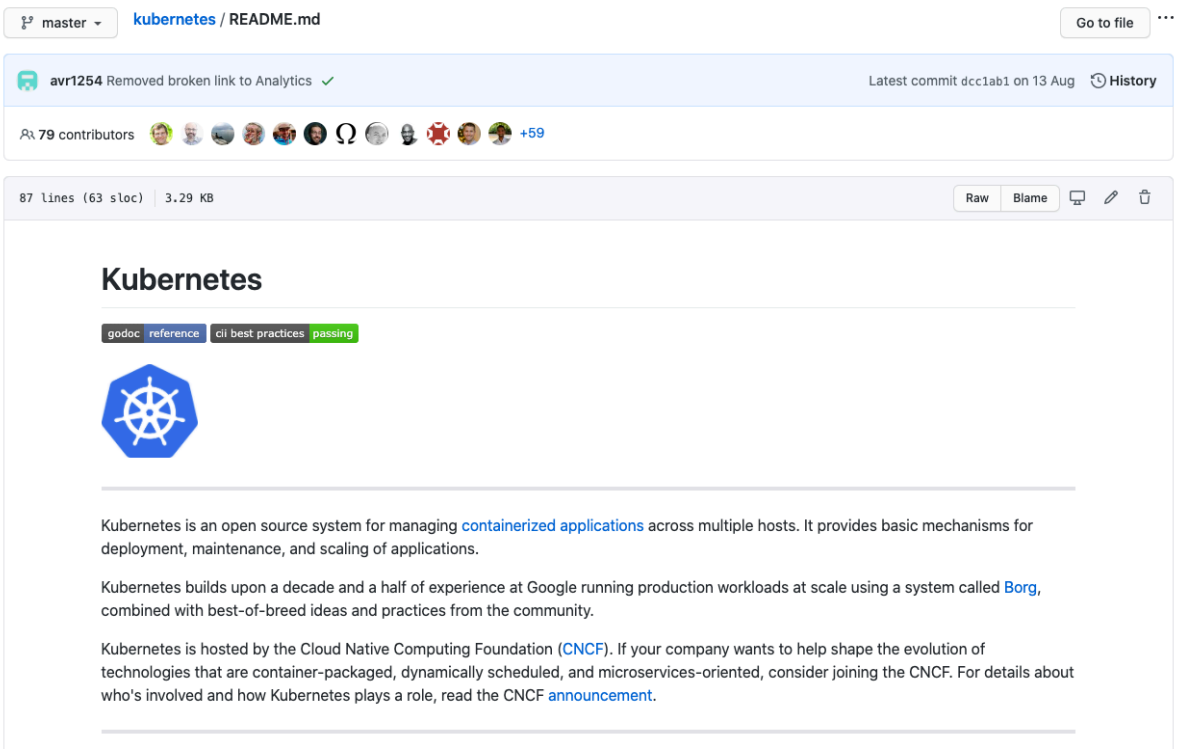
AVM originations	Valuation and confidence level for purchase and remortgage	Financial Services
AVM portfolio	Quarterly review of portfolio returning values and confidence levels	Financial Services
Property Data Solutions	Desktop tool providing comparables and other market insight to property professionals	Public sector housing associations, developers
Data sales	Raw data sold to third parties	Proptech, Zoopla

What is Kubernetes?

hometrack

PART OF Zoopla

Kubernetes

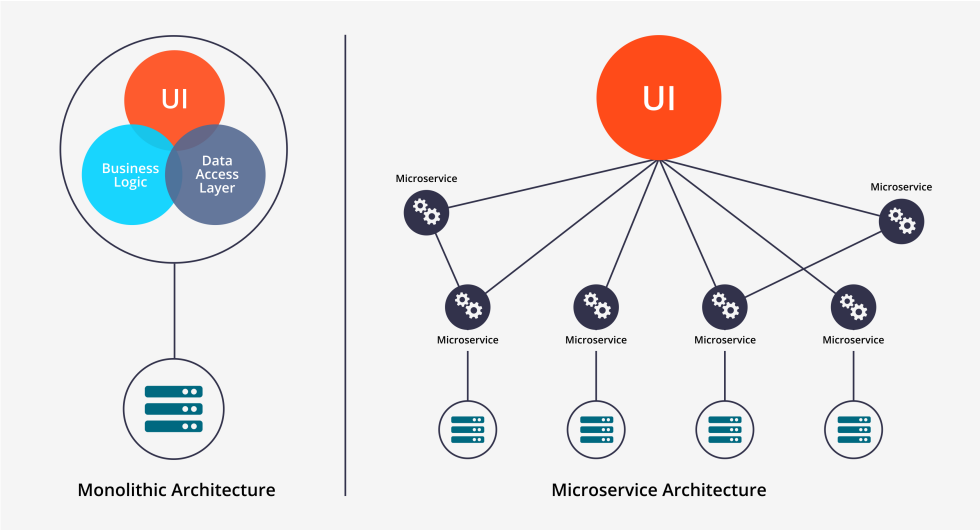


Source: github.com/kubernetes/kubernetes/

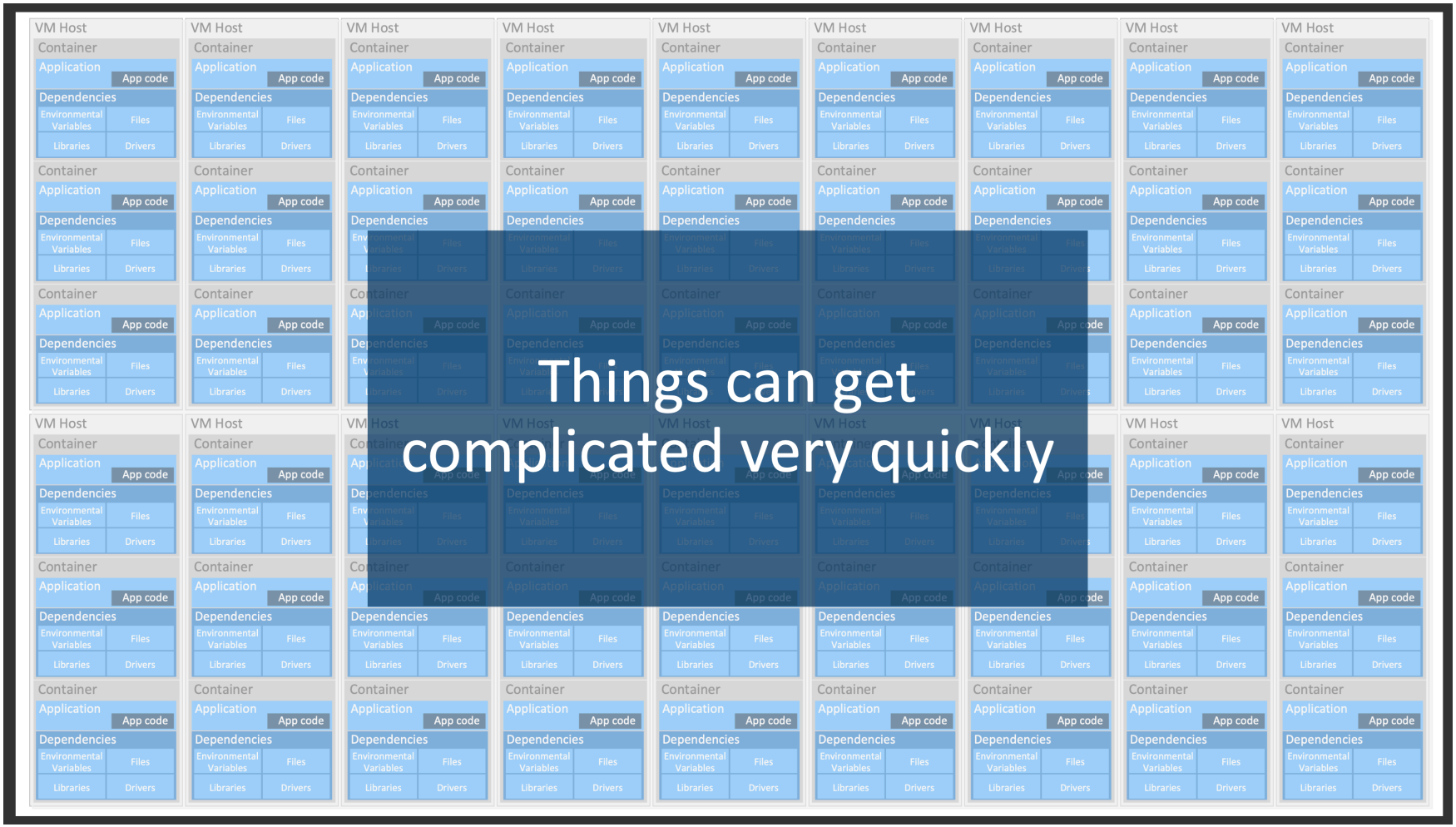
Kubernetes

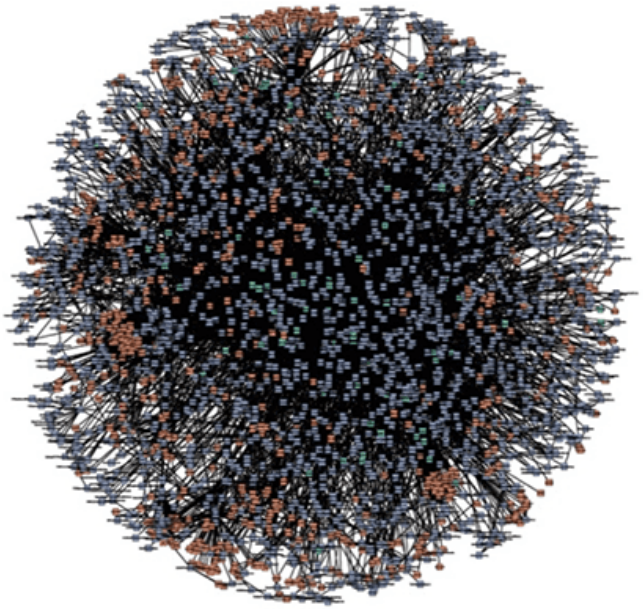
Kubernetes runs Applications in a Cluster.

Applications = Pods.



Source: github.com/kubernetes/kubernetes/


VM Host
Container
Application
App code
Dependencies
Environmental Variables
Files
Libraries
Drivers
Things can get complicated very quickly



amazon.com

NETFLIX

Why Kubernetes?

Business ask us

Why Kubernetes?

Cluster Management: deploy and manage cluster resources

Scheduling: where containers run

Lifecycle and Health: keep containers running despite failure

Naming and Discovery: where are my containers

Load Balancing: evenly distribute traffic

Scalling: make sets of containers elastic in numbers

Image repository: centralized secure Docker container images

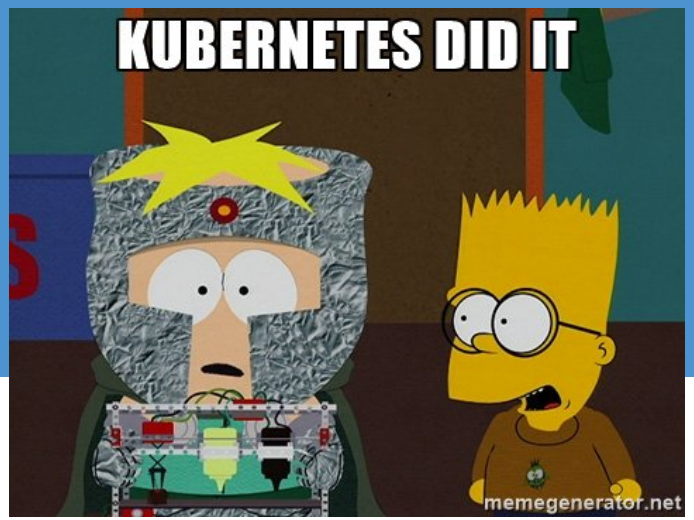
Continous Delivery: CI/CD pipeline and workflow

Logging and Monitoring: track what's happening in containers and cluster

Storage Volumes: persistent data for containers

Kubernetes

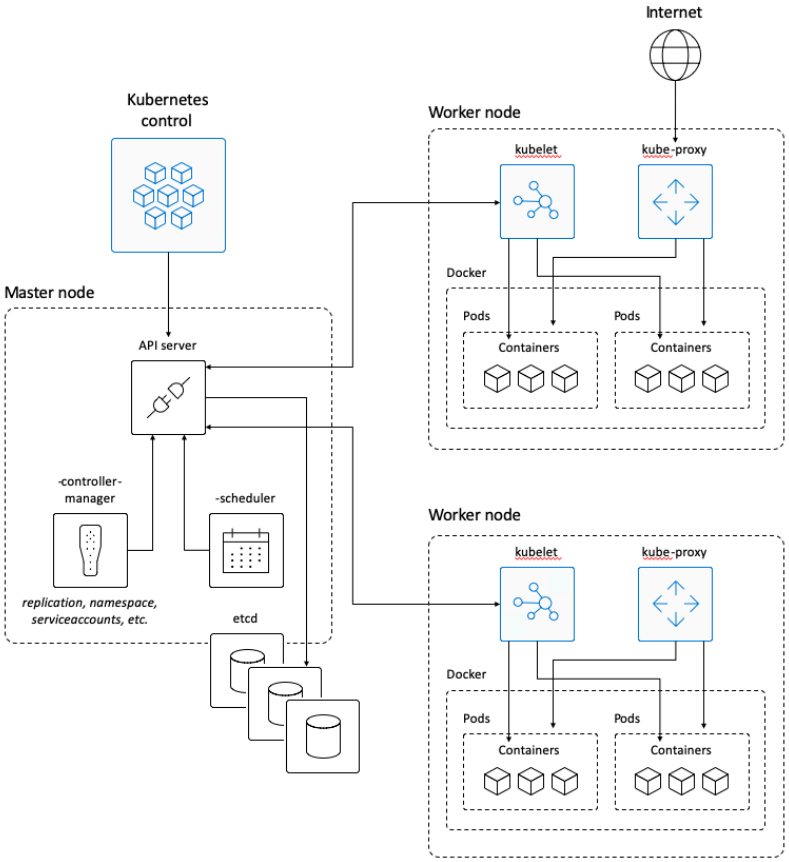
Components / Concepts



hometrack

PART OF Zoopla

Architecture



Kubernetes Master Components

etcd cluster a simple, distributed key value storage which is used to store the Kubernetes cluster data (such as number of pods, their state, namespace, etc), API objects and service discovery details. It is only accessible from the API server for security reasons. etcd enables notifications to the cluster about configuration changes with the help of watchers. Notifications are API requests on each etcd cluster node to trigger the update of information in the node's storage.

Kubernetes Master Components

kube-apiserver Kubernetes API server is the central management entity that receives all REST requests for modifications (to pods, services, replication sets/controllers and others), serving as frontend to the cluster. Also, this is the only component that communicates with the etcd cluster, making sure data is stored in etcd and is in agreement with the service details of the deployed pods.

Kubernetes Master Components

kube-controller-manager runs a number of distinct controller processes in the background (for example, replication controller controls number of replicas in a pod, endpoints controller populates endpoint objects like services and pods, and others) to regulate the shared state of the cluster and perform routine tasks. When a change in a service configuration occurs (for example, replacing the image from which the pods are running, or changing parameters in the configuration yaml file), the controller spots the change and starts working towards the new desired state.

Kubernetes Master Components

cloud-controller-manager is responsible for managing controller processes with dependencies on the underlying cloud provider (if applicable). For example, when a controller needs to check if a node was terminated or set up routes, load balancers or volumes in the cloud infrastructure, all that is handled by the cloud-controller-manager.

Kubernetes Master Components

kube-scheduler helps schedule the pods (a co-located group of containers inside which our application processes are running) on the various nodes based on resource utilization. It reads the service's operational requirements and schedules it on the best fit node. For example, if the application needs 1GB of memory and 2 CPU cores, then the pods for that application will be scheduled on a node with at least those resources. The scheduler runs each time there is a need to schedule pods. The scheduler must know the total resources available as well as resources allocated to existing workloads on each node.

Kubernetes Worker Node Components

kubelet the main service on a node, regularly taking in new or modified pod specifications (primarily through the kube-apiserver) and ensuring that pods and their containers are healthy and running in the desired state. This component also reports to the master on the health of the host where it is running.

Kubernetes Worker Node Components

kube-proxy a proxy service that runs on each worker node to deal with individual host subnetting and expose services to the external world. It performs request forwarding to the correct pods/containers across the various isolated networks in a cluster.

Kubernetes Concepts

pod generally refers to one or more containers that should be controlled as a single application. A pod encapsulates application containers, storage resources, a unique network ID and other configuration on how to run the containers.

pods are single or group of container that share

- localhost
- storage
- ip address
- port range

Kubernetes Concepts

Defining a pod

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! &&
sleep 3600']
```

Kubernetes Concepts

Service pods are volatile, that is Kubernetes does not guarantee a given physical pod will be kept alive (for instance, the replication controller might kill and start a new set of pods). Instead, a service represents a logical set of pods and it can be **defined** as an abstraction on the top of the **pod** which provides a single IP address and DNS name by which pods can be accessed.

A service in Kubernetes exposes a set of pods

- Creates an IP
- Sets up basic routing to the pods
- Talks to the cloud-manager to assign a public IP or load balancer
- ClusterIP, NodePort, Load Balancer

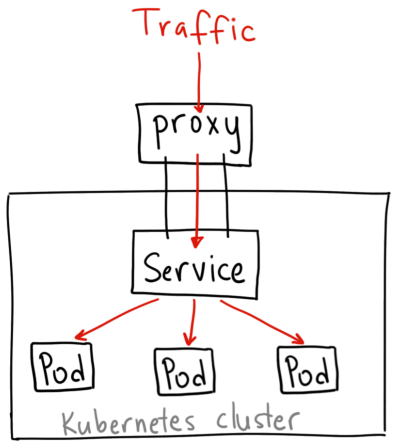
Kubernetes Concepts

Defining a service

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

Kubernetes Concepts

ClusterIP A ClusterIP service is the default Kubernetes service. It gives you a service inside your cluster that other apps inside your cluster can access. There is no external access.



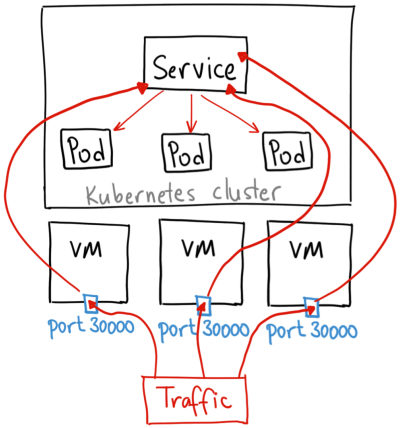
Kubernetes Concepts

Defining a ClusterIP

```
apiVersion: v1
kind: Service
metadata:
   name: my-internal-service
spec:
   selector:
      app: my-app
      type: ClusterIP
   ports:
   - name: http
     port: 80
     targetPort: 80
     protocol: TCP
```

Kubernetes Concepts

NodePort A NodePort service is the most primitive way to get external traffic directly to your service. NodePort, as the name implies, opens a specific port on all the Nodes (the VMs), and any traffic that is sent to this port is forwarded to the service.



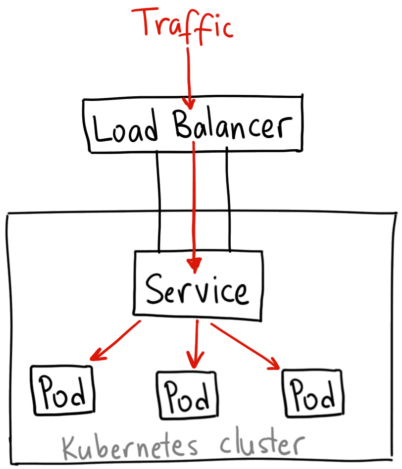
Kubernetes Concepts

Defining a NodePort

```
apiVersion: v1
kind: Service
metadata:
   name: my-nodeport-service
spec:
   selector:
     app: my-app
     type: NodePort
   ports:
   - name: http
     port: 80
     targetPort: 80
     nodePort: 30036
     protocol: TCP
```

Kubernetes Concepts

LoadBalancer A LoadBalancer service is the standard way to expose a service to the internet. On AKS, this will spin up a Network Load Balancer that will give you a single IP address that will forward all traffic to your service.



Kubernetes Concepts

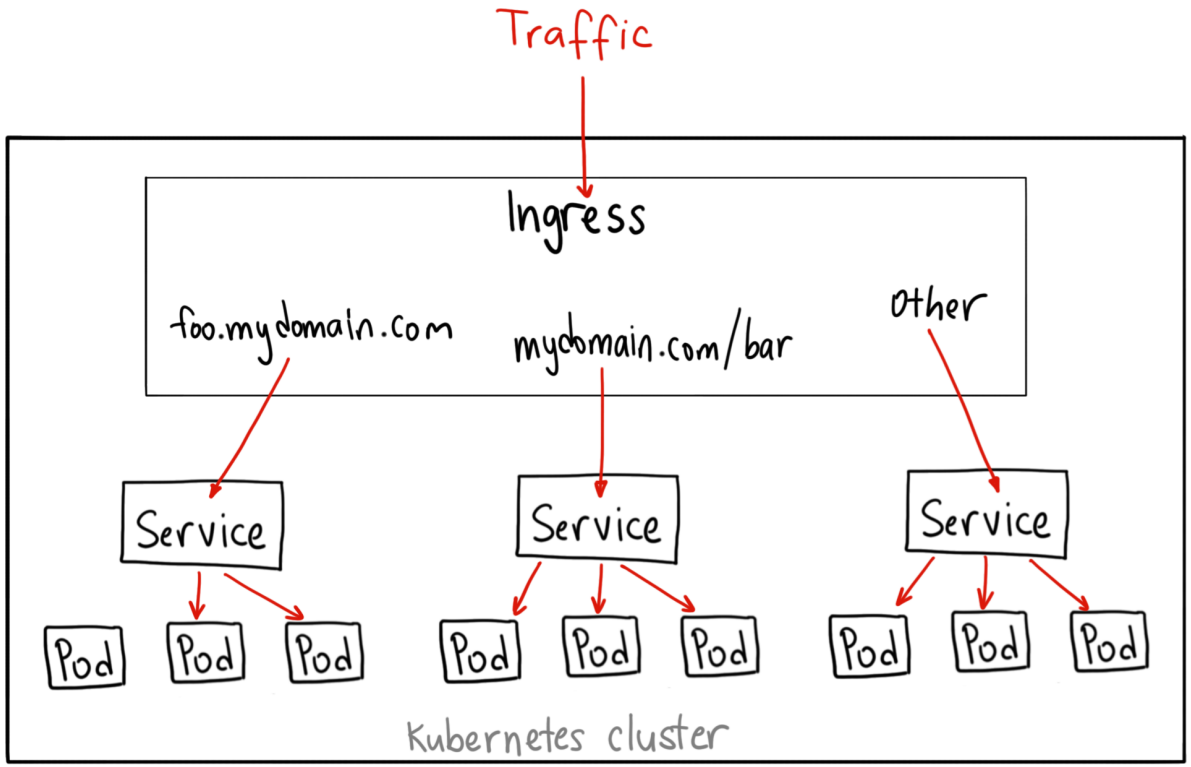
Defining a LoadBalancer

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  clusterIP: 10.0.171.239
  type: LoadBalancer
```

Kubernetes Concepts

Ingrees Unlike all the above examples, Ingress is actually NOT a type of service. Instead, it sits in front of multiple services and act as a “smart router” or entrypoint into your cluster.
You can do a lot of different things with an Ingress, and there are many types of Ingress controllers that have different capabilities.

Kubernetes Concepts



Kubernetes Concepts

Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
spec:
  backend:
    serviceName: other
    servicePort: 8080
  rules:
  - host: foo.mydomain.com
    http:
      paths:
      - backend:
          serviceName: foo
          servicePort: 8080
  - host: mydomain.com
    http:
      paths:
      - path: /bar/*
        backend:
          serviceName: bar
          servicePort: 8080
```

Kubernetes Concepts

Volume similar to a container volume in Docker, but a Kubernetes volume applies to a whole pod and is mounted on all containers in the pod. Kubernetes guarantees data is preserved across container restarts. The volume will be removed only when the pod gets destroyed. Also, a pod can have multiple volumes (possibly of different types) associated.

Kubernetes Concepts

Namespace a virtual cluster (a single physical cluster can run multiple virtual ones) intended for environments with many users spread across multiple teams or projects, for isolation of concerns. Resources inside a namespace must be unique and cannot access resources in a different namespace. Also, a namespace can be allocated a resource quota to avoid consuming more than its share of the physical cluster's overall resources.

Kubernetes Concepts

Deployment describes the desired state of a pod or a replica set, in a yaml file. The deployment controller then gradually updates the environment (for example, creating or deleting replicas) until the current state matches the desired state specified in the deployment file. For example, if the yaml file defines 2 replicas for a pod but only one is currently running, an extra one will get created. Note that replicas managed via a deployment should not be manipulated directly, only via new deployments.

Deployments defines the lifecycle of an application,

- Is made up of pods
- It controls replicasets
- Includes the functionality to update the desired state
- Rolling updates are included

Kubernetes Concepts

Defining a deployment

```
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:1.7.9
        ports:
        - containerPort: 80
```

@scottcoulton

Kubernetes Concepts

Stateful Sets StatefulSet is the workload API object used to manage stateful applications. Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods. Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

Kubernetes Concepts

ReplicaSet A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

A replicaSet defines that state of a running application

- The amount of pods that should be running
- It self-heals the pods to their desired state.

Kubernetes Concepts

DaemonSet A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

Some typical uses of a DaemonSet are:
- running a cluster storage daemon, such as glusterd, ceph on each node.
- running a logs collection daemon on every node, such as fluentd or logstash.
- running a node monitoring daemon on every node, such as Prometheus, Datadog agent, New Relic agent

Kubernetes Concepts

Declarative vs. Imperative

- Commands like kubectl run and kubectl expose are **imperative** commands (do this thing now)

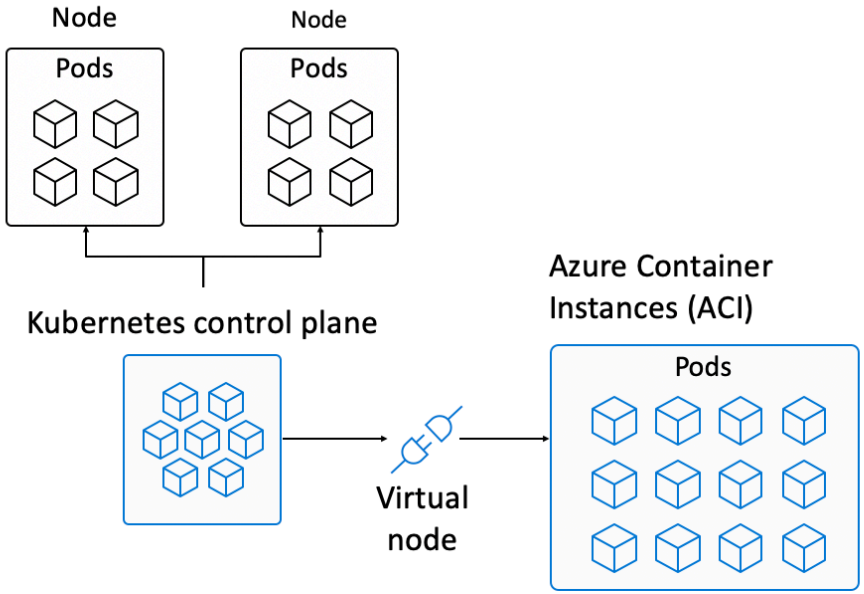
```
$> kubectl run -i --tty busybox --image=busybox --restart=Never -- sh
```

- **Declarative** way – Describe the state of resources in a file (JSON or YAML).

  kubectl apply –f webresource.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
```

AKS Virtual Node

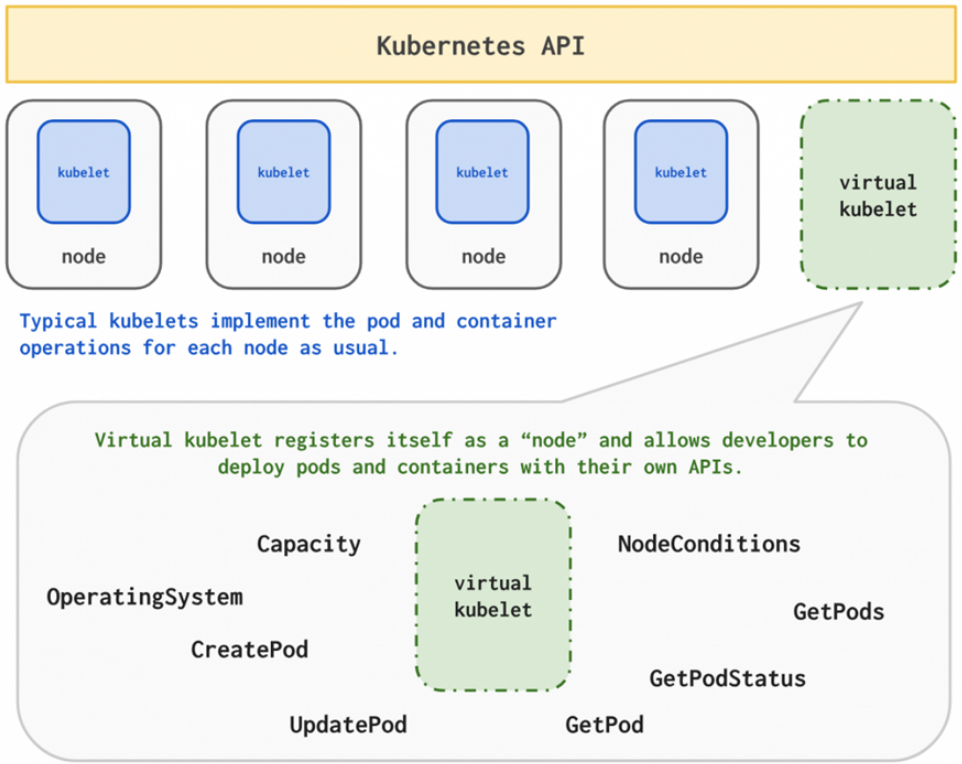


Serverless Kubernetes

No infrastructure to manage

Starts in seconds

AKS Virtual Node is based Off Virtual Kubelet



DEMO

hometrack

PART OF Zoopla

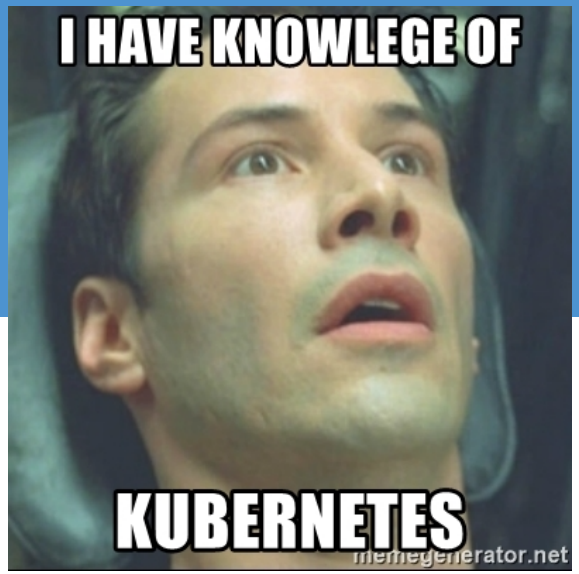
Zoopla

We are hiring!

Over 50+ vacancies in Product & Tech
Come help us re-imagine the property industry!

Apply at careers.zoopla.co.uk

Any Questions?



hometrack

PART OF Zoopla

Thanks

Osman Sahin

Github: https://git.io/JUBUu

hometrack

PART OF Zoopla