

# Kubernetes

Let's talk about container orchestration!

Then let's turn into peiratéés!

# This Section ToC

What does Kubernetes do?

Attacking Kubernetes clusters

Defense: RBAC and Authorization Modules

Exercise: Own the Nodes

YAML Review

Defense: Network Policies

Defense: CIS Benchmark

Defense: Image Safety

# Cloud Native's Birth: the API (Service) Moment

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology you use.
- All service interfaces, without exception, must be designed from the ground up to be externalize-able. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- The mandate closed with: Anyone who doesn't do this will be fired.  
Thank you; have a nice day!

**Jeff Bezos' 2002 API  
Mandate Memo**

# Amazon Web Services

- The memo forced every single connectable software project at Amazon to function as a product.
- In 2002, the same year as the memo, Amazon went from an online retailer to the cloud service provider that also operated a retail business.
- Amazon's market share in cloud services is 33%, which is larger than the next three players put together (as of 2022).

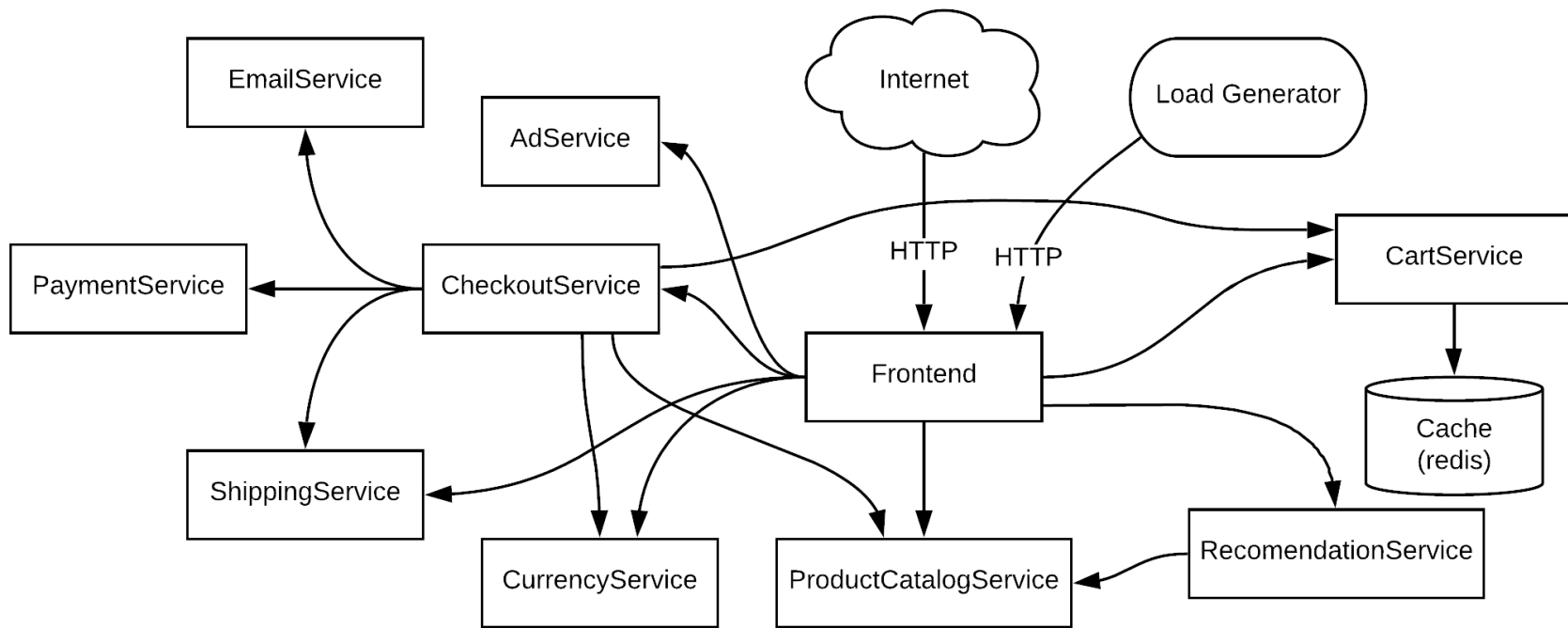
## Market Share

AWS: 33%

Azure: 22%

GCP: 9%

# Microservice Architecture



Ref: <https://github.com/GoogleCloudPlatform/microservices-demo>  
(Copyright 2020 Google LLC, distributed under Apache license)

**Google launched 2 billion  
containers per week in  
2014**

**(approx. 3,300/second)**



**They did this with roughly 2.5  
million servers in 2016.**

**Hard drives had an annualized  
failure rate of 1.95% in 2016**

**At one drive per server, that's  
133 drive failures per day, or  
every 9 minutes.**



**What features would you  
need to manage that?**

**Reference and Fascinating Presentation:  
Joe Beda, GlueCon 2014 Presentation**

**<https://bit.ly/3fmYzu0>**

# Kubernetes Features

- Bin Packing (Assigning workloads to machines)
- Self Healing
- Horizontal Scaling
- Service Discovery and Load Balancing
- Secret and Configuration Management
- Storage Orchestration
- Automated Rollouts and Rollbacks
- A/B Testing

**Software-defined  
Datacenter via  
Container  
Orchestration**

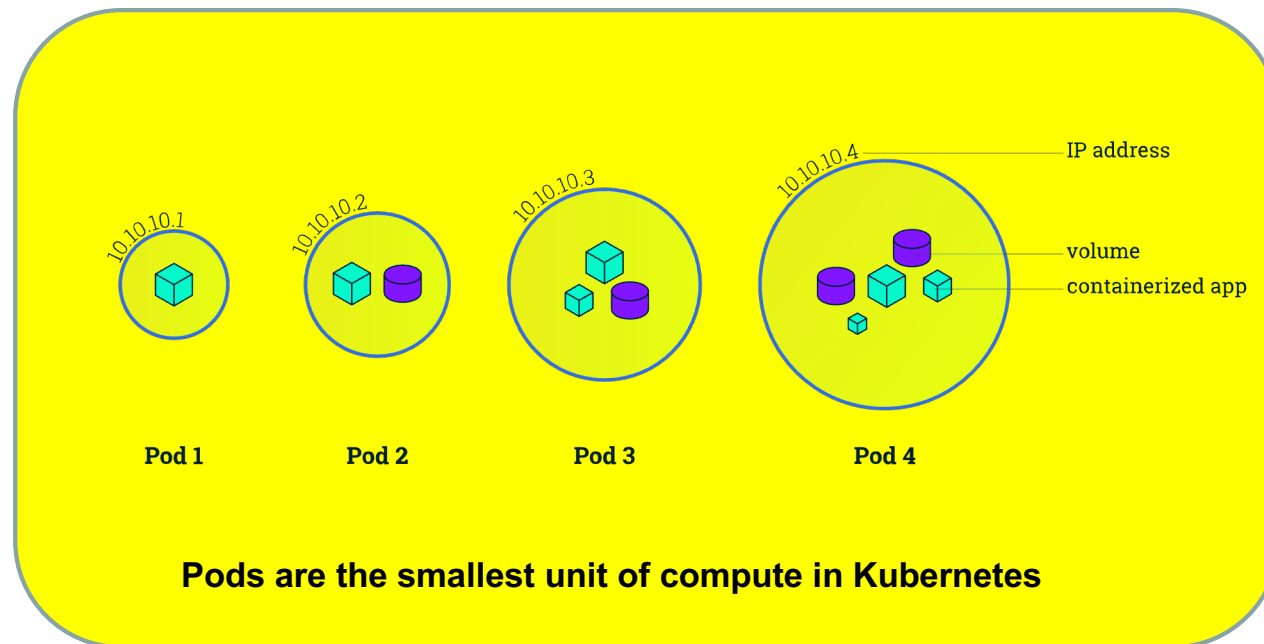
Control Loops and the Declarative Model make this possible

# Kubernetes Concepts and Terms

- Pods and Volumes
- Nodes
- Services
- Deployments
- Namespaces



# Pods: Containers and Volumes



**All containers in a pod share an IP address and may share the volumes defined in that pod.**

## Reference: Pods

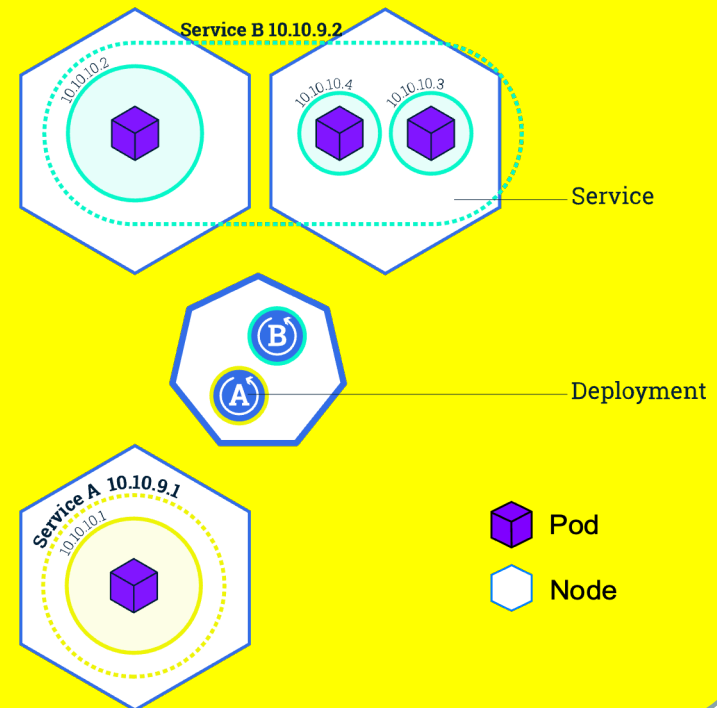
- a collection of one or more containers
- the smallest unit of work in Kubernetes
  - Expresses shares-a-host dependency between containers
  - If two programs absolutely must be placed onto the same node, use separate containers sharing a pod
- always includes a “pause” container
- shares a single network kernel namespace between containers
  - All containers in a pod have the same IP address
  - Programs across a pod must avoid binding to the same port numbers
- may define a volume for storage, which can be mounted into one or more of the containers' filesystems.

# Deployment: Creating and Maintaining Pods

## Deployment:

**A deployment creates pods from the image you specify.**

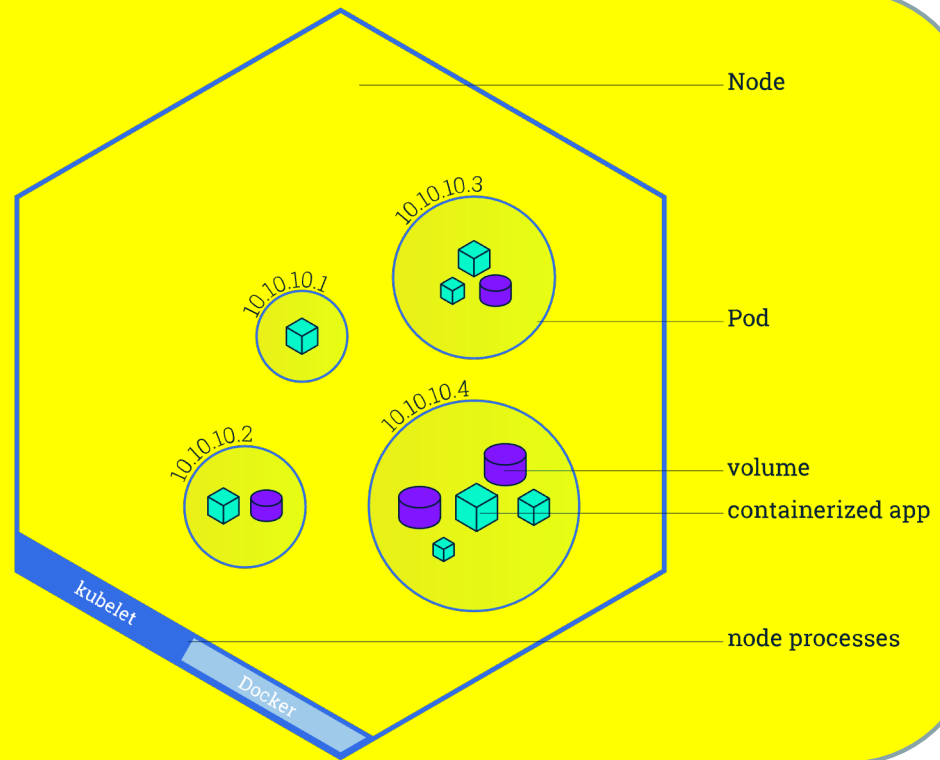
**It maintains and scales the right number of pods, through both crashes and load increases/decreases.**



# Nodes: Hosts in the Cluster

## Nodes run:

- Kubelet
- Container Runtime (Docker, containerd, ...)
- Kube-Proxy



## Reference: Nodes

- A node is a Kubernetes host (virtual or physical machine) where containers are staged.
- A node has these components:
  - A container runtime (Docker, containerd, CRI-O,...)
  - `kubelet`
  - `kube-proxy`
- **Container runtime:** instructs the kernel to create containers
- **kubelet:** tells the container runtime what to create, destroy or configure.
- **kube-proxy:** configures `iptables`, IPVS, and otherwise proxies traffic.

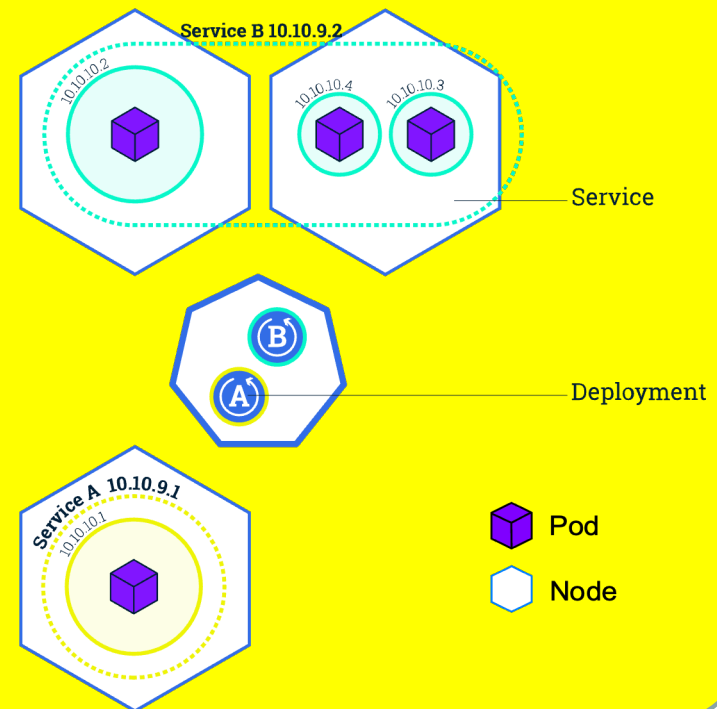
# Services: Load Balancers

**Service: a load balancer**

**A service creates:**

- a DNS name
- a virtual IP address
- an incoming/outgoing port pair

**These redirect traffic to pods whose labels match those specified in the service's manifest.**



# Services: DNS Names

Services create a DNS name (A record) in DNS:

**app.default.svc.cluster.local**

Services also create SVC records for the named port:

**\_80-80.\_tcp.app.default.svc.cluster.local**

```
apiVersion: v1
kind: Service
metadata:
  name: app
  namespace: default
  labels:
    app: app
spec:
  type: ClusterIP
  selector:
    app: app
  ports:
    - name: 80-80
      port: 80
      protocol: TCP
      targetPort: 80
  status:
    loadBalancer: {}
```

# Namespaces Organize Objects

- A namespace is a logical grouping for Kubernetes objects (pods, roles, ...)
- Namespaces might separate projects, users, or departments – it's up to the admins.
- Every cluster starts with a default namespace and at least three kube- namespaces.
- The primary two universal namespaces you'll interact with are:

`default:` Resources are deployed here when namespace isn't specified

`kube-system:` Kubernetes' default control plane components are here.

Any namespace that begins with `kube-` is considered a control-plane namespace.



# Kubernetes Glossary

- Containers: Linux namespace and control group-based "lightweight VMs"
- Pods: collections of containers, the smallest unit of work in Kubernetes
- Nodes: hosts on which the containers/pods run
- Services: load balancers, allowing pods to scale and fail
- Deployments: method for creating pods and handling scaling and failing
- Namespaces: logical groupings of resources, possibly by tenant, department or application

# Control Loops

- Kubernetes is a "declarative" system, rather than an "imperative" one.
- You tell Kubernetes to keep (5) copies of a container running, by creating a deployment.
- Kubernetes takes responsibility for keeping five containers staged, spread out to as many as five machines (nodes), watching for container or node failures.
- It does this by running control loops, which continually check the reality of the cluster against the desired state you've specified.
- Whenever the reality doesn't match the desired state, a controller takes action to correct that, without waiting for a human to notice.

# Control Plane Node-Only Components (1/2)

The following Kubernetes control plane components are run only on control plane nodes:

- **Kubernetes API Server**
  - Accepts declarative object configurations, generated by kubectl and API requests.
  - Serves as the first point of contact for the cluster.
- **etcd Server**
  - Retains the state of every object in the cluster.
  - Allows "is the answer different from the last time I asked" queries.
- **Controller Manager**
  - Runs control loops to bring the cluster's state to parity with etcd's contents
  - Contains multiple controllers, all compiled into one binary.

## Control Plane Node-Only Components (2/2)

- Scheduler
  - Chooses a node for each new pod, subject to constraints. (i.e., "bin packs workloads")
- CoreDNS (replacing Kube-DNS)
  - Gives every endpoint a DNS name, like postgres.mktg.svc.cluster.local

# Vital Kubernetes Target Components: All Nodes

- Kubelet
  - Bridges the Kubernetes infrastructure to the container runtime (e.g., containerd, CRI-O, Docker,...)
- Container Runtime
  - Pulls container images and instructs the kernel to create/destroy containers, as well as other functionality.
- Kube-Proxy
  - configures `iptables`, IPVS, and otherwise proxies traffic.
- Pods
  - Control plane components
  - Workloads.