Container Orchestration with Kubernetes

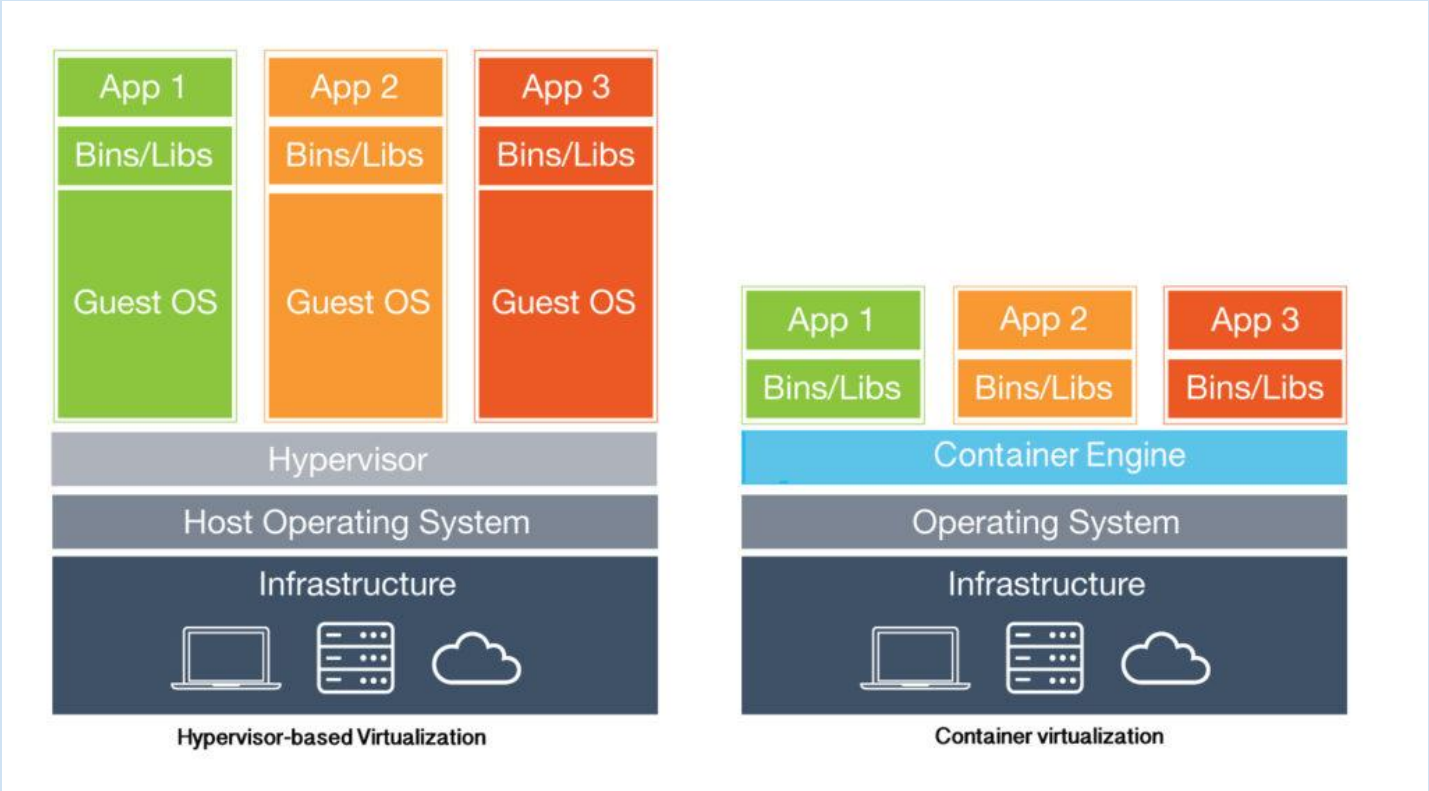
Giulia Frascaria

Table of Contents

Why containers?

- Infrastructure as a Service (IaaS) is difficult to maintain
- VMs are big, a lot of unnecessary resources need to be allocated to run them
- Containers (Docker) are light VMs
- Dependencies are bundled automatically

See the difference?



Running Applications with Docker

- Write code, pack it with dependencies, run
- Easy to run applications in containers
- Good step towards Platform as a Service (PaaS)

But is it really PaaS?

From Containers to Services

We have an application in Docker containers, now what?

- Expose the service to the user
- Link pieces together (e.g. microservices)
- Scale to the demand of the users
- Recover from failures
- Manage updates

And a lot more...

What is a Cloud Orchestrator? - Part 1

In a sense, the operating system of distributed systems

- Allocates resources
- Schedules “processes”
- Abstracts the (distributed) hardware
- Guarantees isolation (together with Docker)

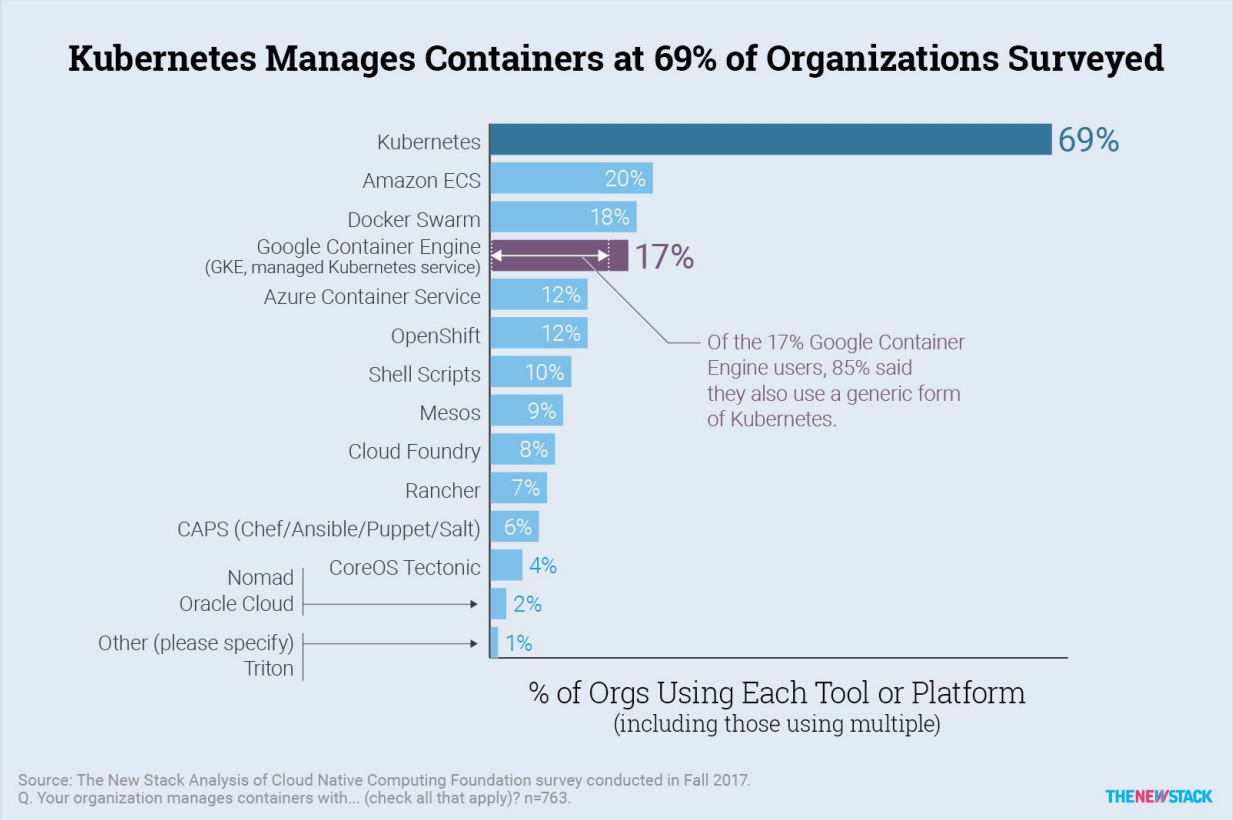
But that’s not all

What is a Cloud Orchestrator? - Part 2

New cloud-specific features

- Performs updates
- Hides failures
- Scales services to meet user demand
- Manages life cycle of applications
- Load balances between replicas

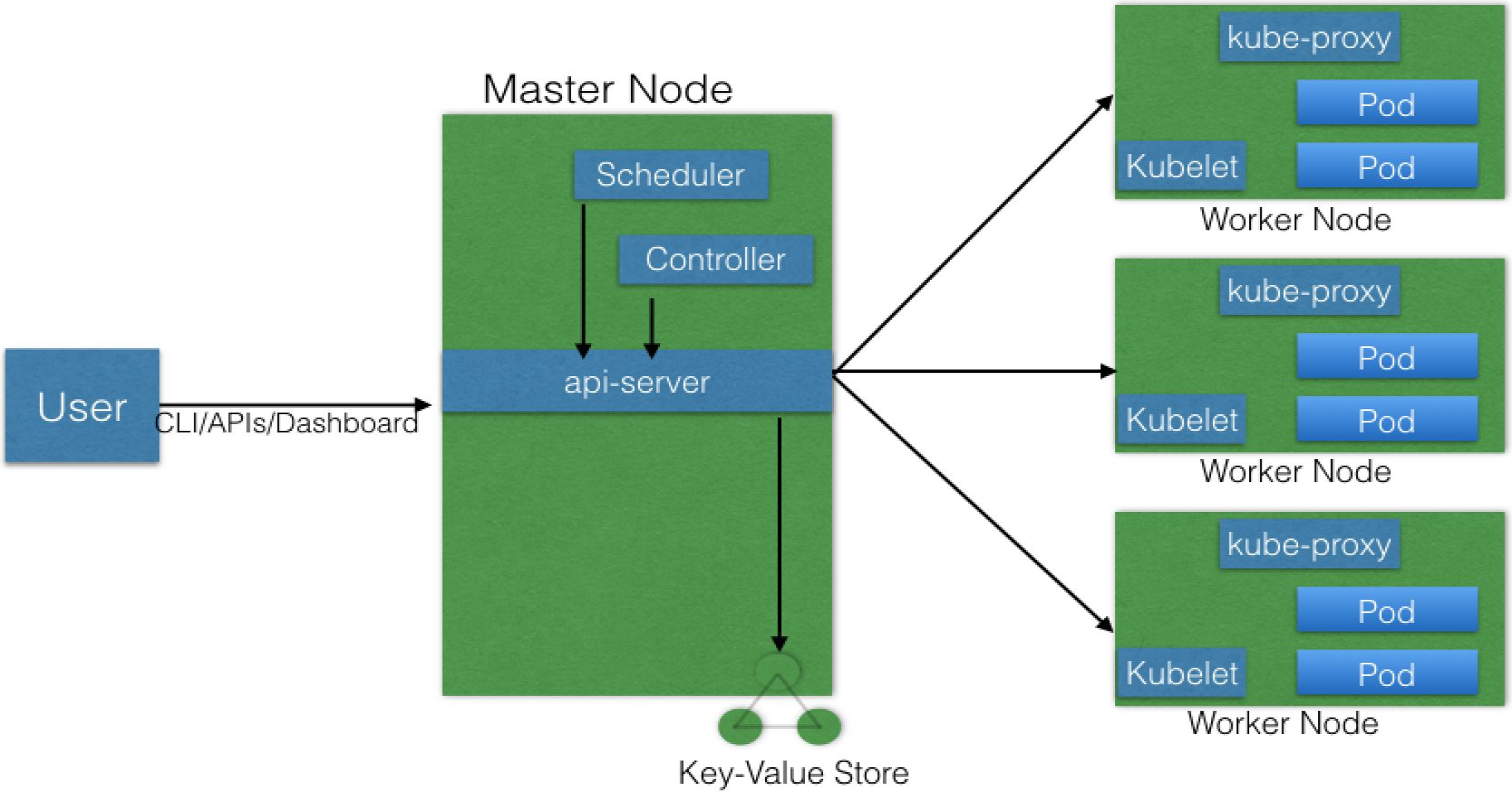
Why Kubernetes



High-level Architecture

- Cluster or physical or virtual machines: **nodes**
- One or more **master** nodes
- One or more **worker** nodes
- A **key-value storage** (etcd)

Magic happens inside the nodes

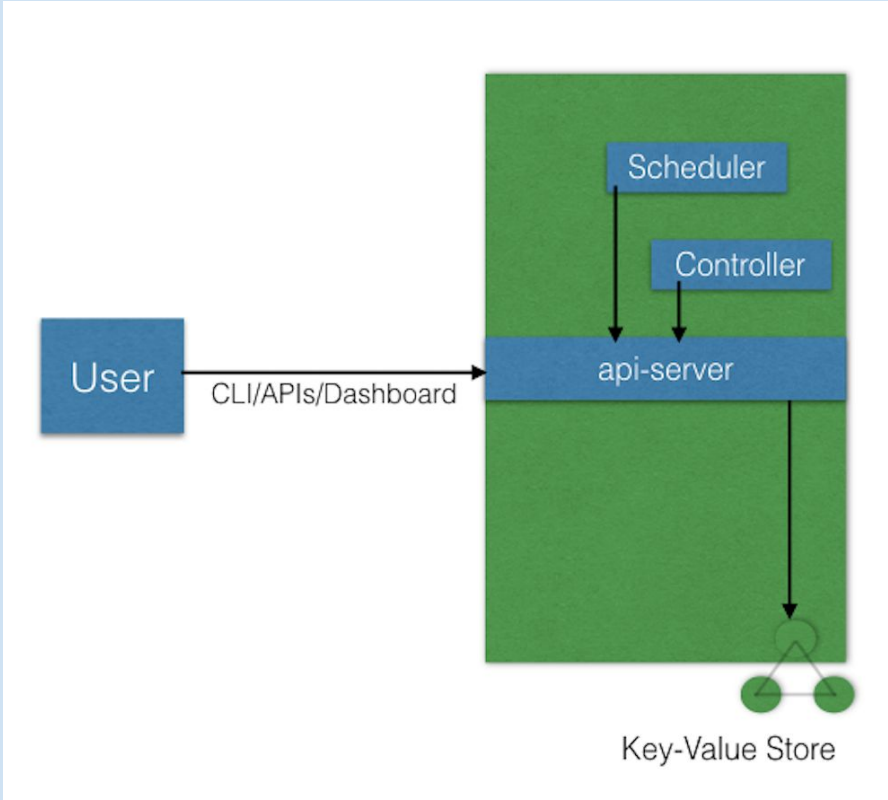

Master Node
Scheduler
Controller
api-server
User
CLI/APIs/Dashboard
Key-Value Store
kube-proxy
Pod
Kubelet
Pod
Worker Node
kube-proxy
Pod
Kubelet
Pod
Worker Node
kube-proxy
Pod
Kubelet
Pod
Worker Node

Master Node

- Manages the cluster
- Entry point for applications

Main components:

- API server
- Controller
- Scheduler
- etcd



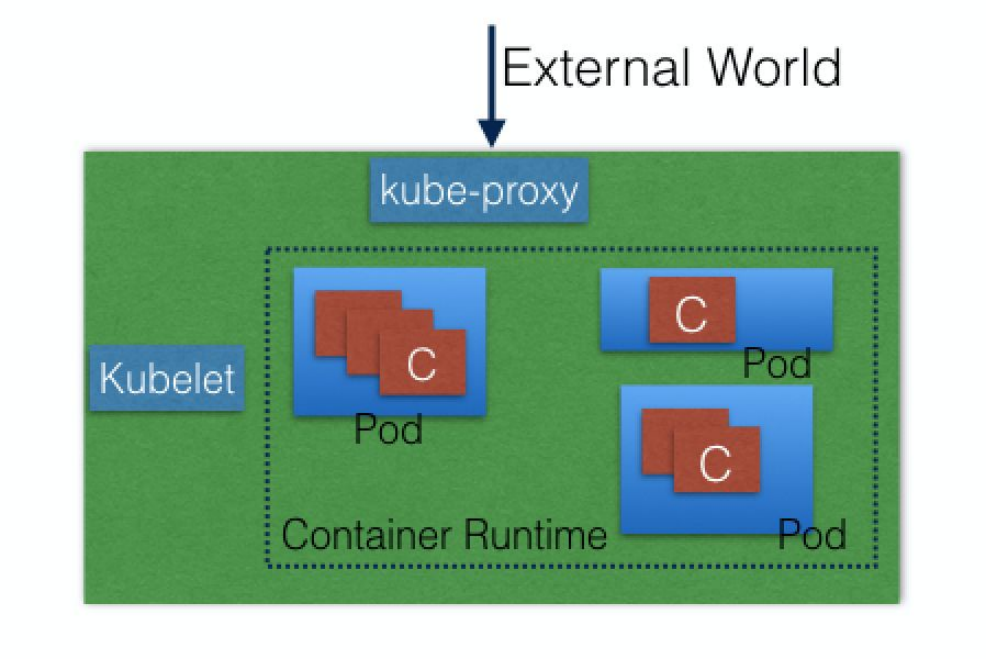
Worker Node

Runs applications in a Container Runtime (Docker)

Main components:

- Kubelet
- Kube-proxy
- Pods

We will talk about pods



Kubelet

The engine in every worker node

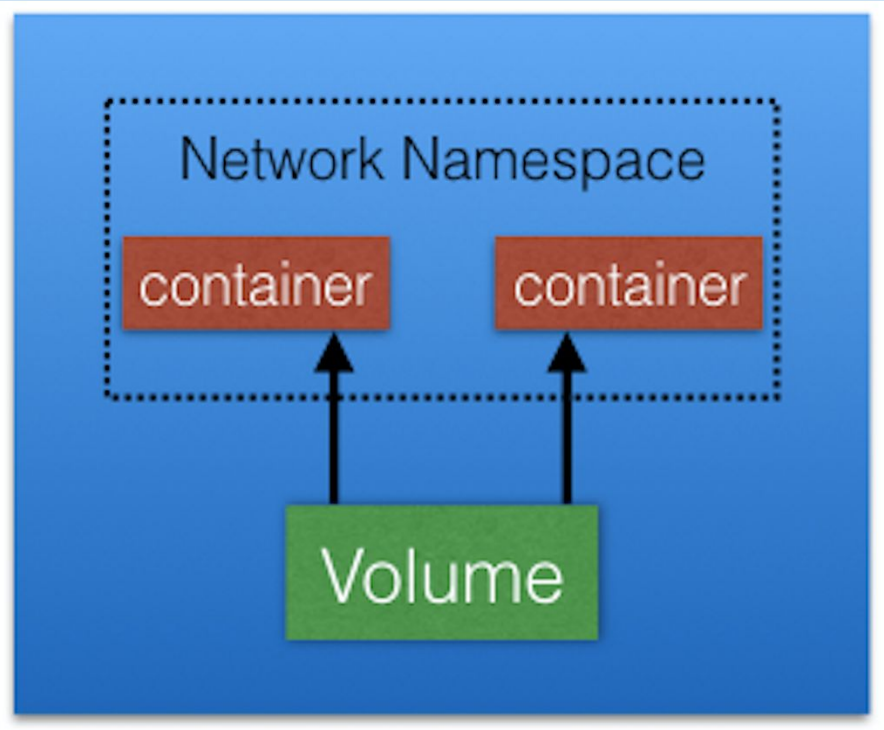
- Communicates with master node
- Runs containers through the Container Runtime Interface
- Only runs containers that were created by Kubernetes

Pod

Smallest object in Kubernetes. Unit of deployment

- Set of containers that are co-located and co-scheduled
- Exposes single networking interface (i.e. each pod has **one IP**)
- Containers inside a pod see each other as localhost
- Containers share storage
- Can’t self-heal (remember the controller?)

Inside a Pod



Networking in Kubernetes

Nodes are connected through a network, but there's more!
Cluster networking requirements in kubernetes:

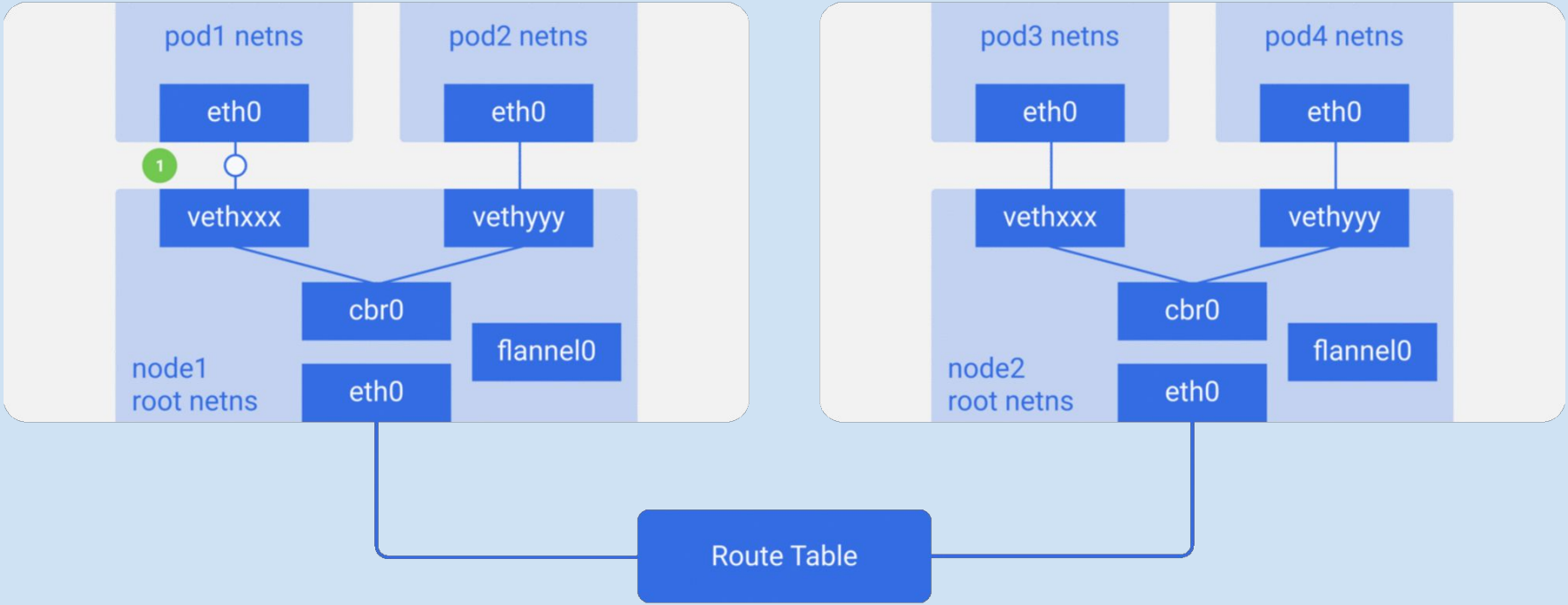
- All pods can communicate without NAT
- All nodes can communicate with all pods (and vice-versa) without NAT
- The IP that every pod sees for itself is the same IP that others see

What does that all mean?

Overlay SDN!

- Container Network Interface
- Different CIDR ranges for nodes and pods
- Tunnel through physical network

Under the hood



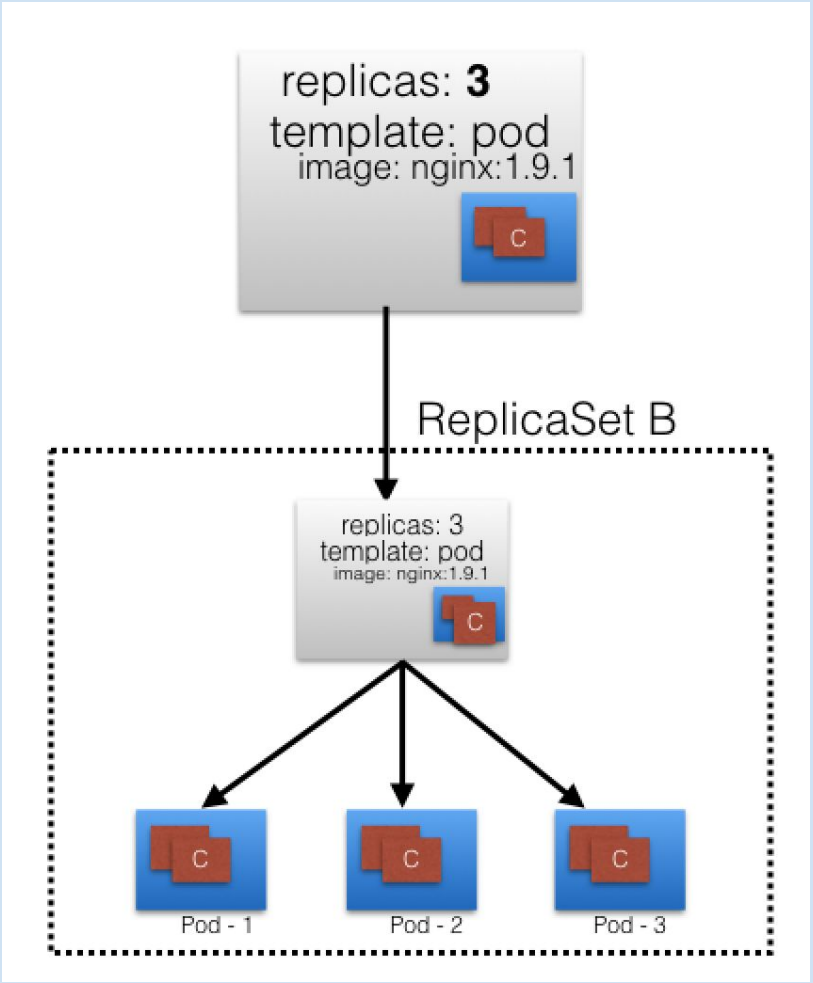
From Pods to Deployments

A pod deploys a single instance of an app, but we saw Kubernetes manages replicas, how?

Deployment: controller of updates to Pods and ReplicaSets (sets of replicated pods)

- Ensures that the right number of replicas is deployed
- Roll-out/roll-back of updates for Pods and ReplicaSets

Wait, what?


replicas: 3
template: pod
image: nginx:1.9.1
C
ReplicaSet B
replicas: 3
template: pod
image: nginx:1.9.1
C
C
C
C
Pod - 1
Pod - 2
Pod - 3

Service

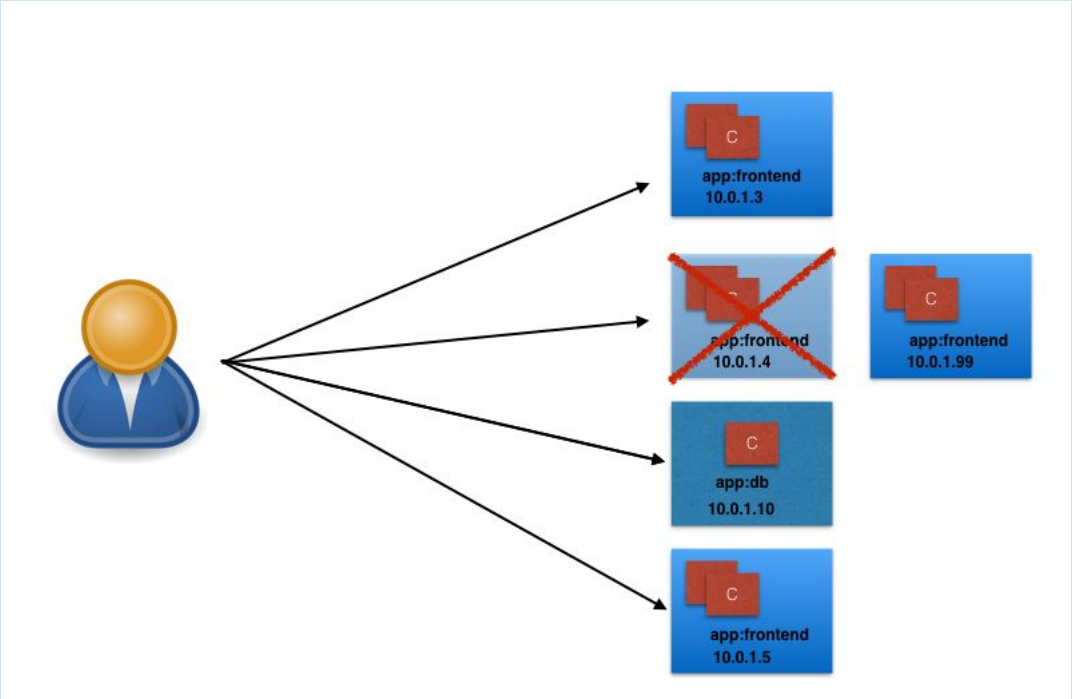
Pods are ephemeral structures.

Services provide a stable access point to Pods

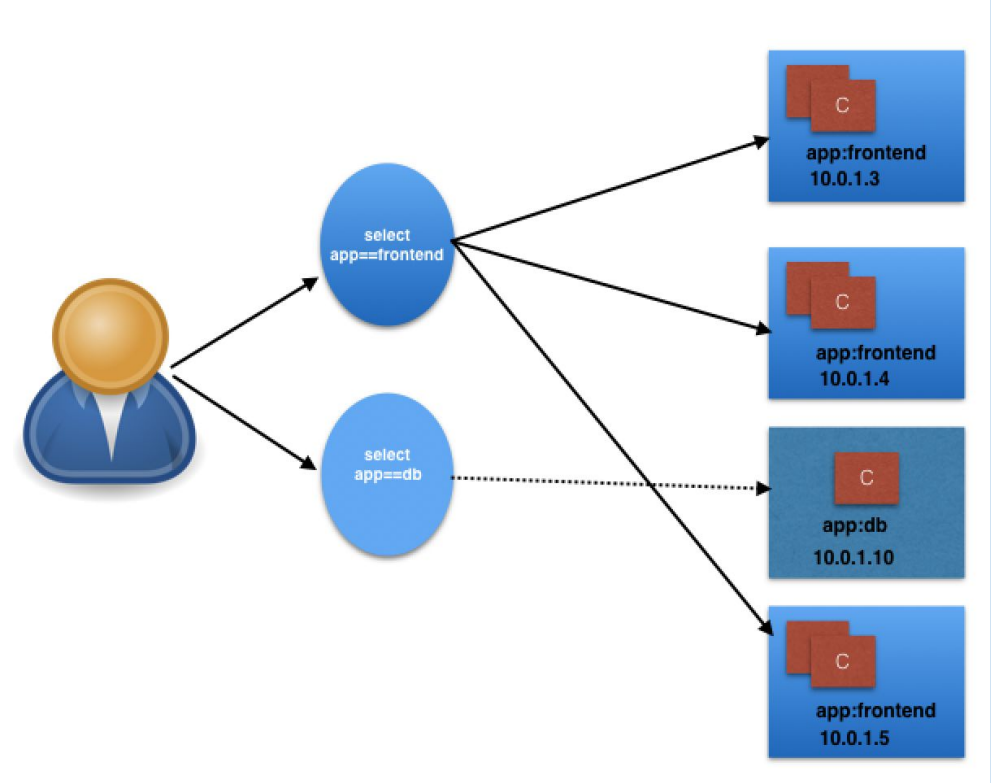
- Pods could die, be replicated
- Pod resources (in particular IP address) are not static

A service exposes stable, cluster-wide IP for a set of pods

Life without services



Life with services

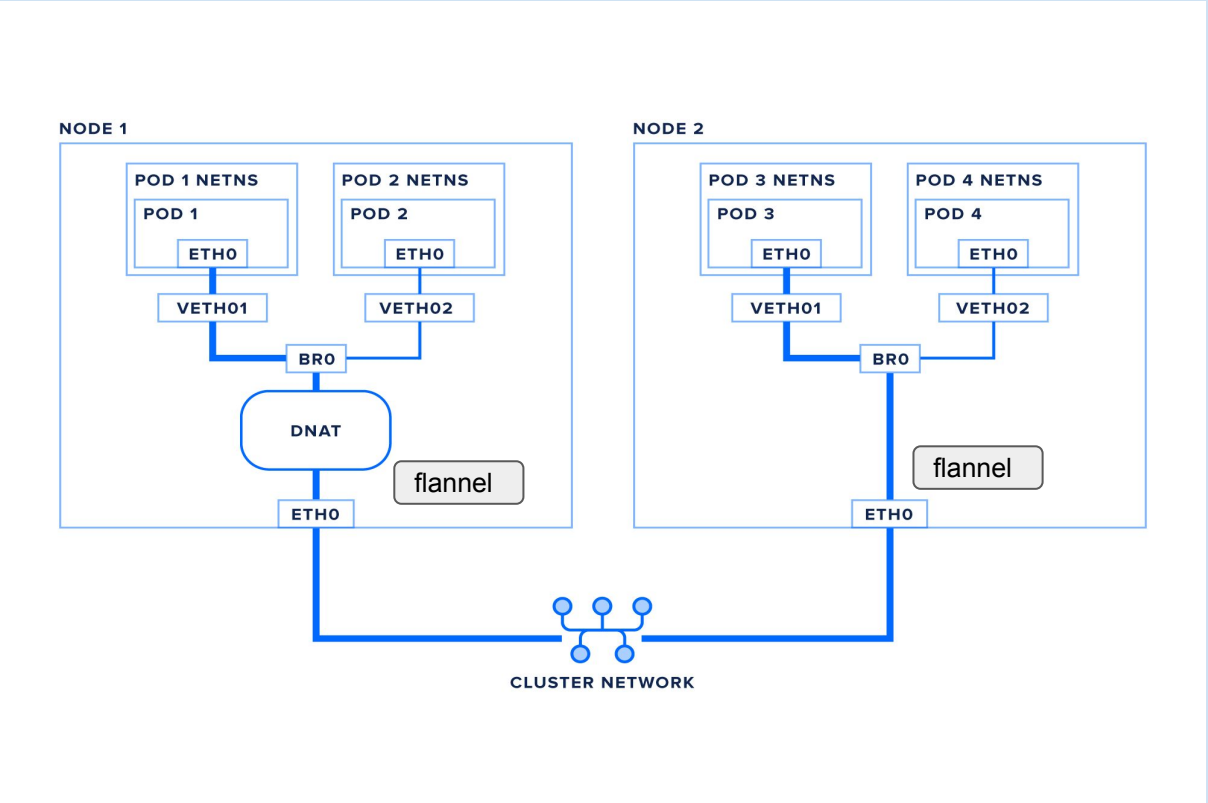


Remember kube-proxy?

- Allows to route traffic to endpoints
- Configures iptables of the node (updated on creation, deletion and replication)
- Can route traffic to other nodes and pods to load-balance

```
$ sudo iptables -S -t nat | grep KUBE-SVC-URRHIARQWDHNXJTW
-N KUBE-SVC-URRHIARQWDHNXJTW
-A KUBE-NODEPORTS -p tcp -m comment --comment "default/nginxservice:" -m tcp --dport 30235 -j KUBE-SVC-URRHIARQWDHNXJTW
-A KUBE-SERVICES -d 10.0.0.179/32 -p tcp -m comment --comment "default/nginxservice: cluster IP" -m tcp --dport 80 -j KUBE-SVC-URRHIARQWDHNXJTW
-A KUBE-SVC-URRHIARQWDHNXJTW -m comment --comment "default/nginxservice:" -m statistic --mode random --probability 0.20000000019 -j KUBE-SEP-RYFM2HXHC6IPPMAX
-A KUBE-SVC-URRHIARQWDHNXJTW -m comment --comment "default/nginxservice:" -m statistic --mode random --probability 0.25000000000 -j KUBE-SEP-FDLRP2EJRFUXWEBK
-A KUBE-SVC-URRHIARQWDHNXJTW -m comment --comment "default/nginxservice:" -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-7MGJBTMYPTJLX6PC
-A KUBE-SVC-URRHIARQWDHNXJTW -m comment --comment "default/nginxservice:" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-OYWFXI3ITHAWD7NY
-A KUBE-SVC-URRHIARQWDHNXJTW -m comment --comment "default/nginxservice:" -j KUBE-SEP-CXXDIQBZNSZTULZQ
```

A small update



From services to microservices

- Monolith applications are hard to maintain
- Want to separate lifecycles of components
- Ecosystems are complex, a lot of interactions

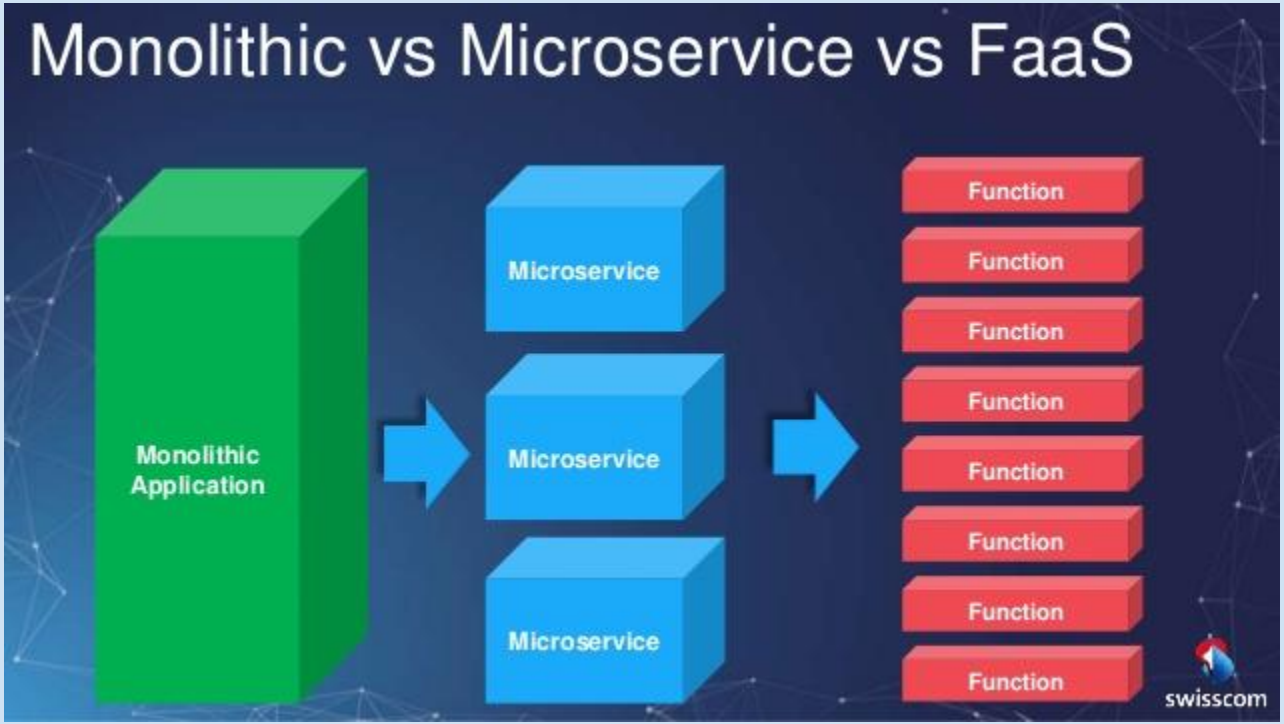
Microservices are just a design paradigm, isolate independent functionalities

Traditional microservices

- Potentially high number of microservices cooperating (Netflix has 500+)
- For us, every microservice is a kubernetes service
- Unfortunately, kubernetes helps to deploy ONE service, not a service mesh
- Huge DevOps work, but engineers just want to do the Dev

Wouldn’t it be great if...

This is how little we like DevOps



(Server)less is more!

- Only care about Dev, leave the Ops to someone else
- (by the way, the server is still there but we don’t have to maintain it)
- Function as a Service (FaaS) gives a better idea
- Developers write the code, server execution happens “automatically”

But what does automatically mean?

Serverless 101

- Code is loaded and executed on demand
- Runtime (e.g. Python) is ready, and serves multiple functions
- Developers just write the code and define trigger events
- Billing is based on execution time, not resources

Kubernetes services are persistent though, now what?

Serverless in Kubernetes

- Need to add an additional layer
- Kubeless, Fission, Apache OpenWisk…
- Keep the runtime ready, link it to the code on demand
- Under the hood, runtimes are kubernetes services, but they are deployed by the serverless framework

Some references

- https://kubernetes.io/
- https://www.youtube.com/watch?v=y2bhV81MfKQ
- https://www.digitalocean.com/community/tutorials/kubernetes-networking-under-the-hood
- https://www.katacoda.com/courses/kubernetes
- https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS158x+1T2018/course/
- https://www.computer.org/csdl/mags/ic/2018/05/mic20180050008.html