Microservices
Containers
Kubernetes
in 20 minutes :)

Seyed M. Afzouni
APKGroup
Sep. 2020

1. Microservices

Monolith

- Running either as a single (small number) process spread across a handful of servers
- Slow release cycles
- Developers package up hand it over to the ops team
- Hardware failures: the ops team manually migrates it to the remaining healthy servers

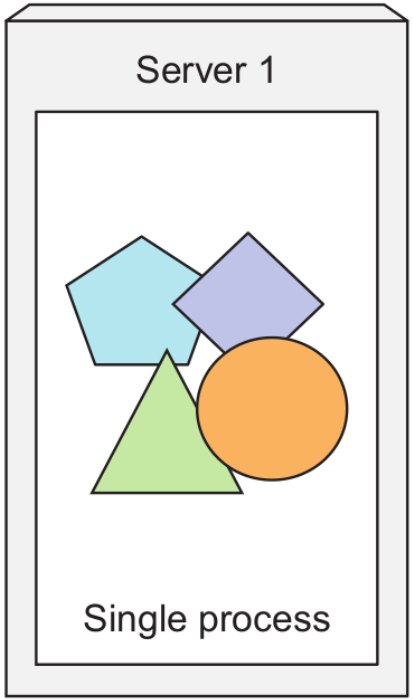
Microservice

Today, big monolithic legacy applications are being broken down into

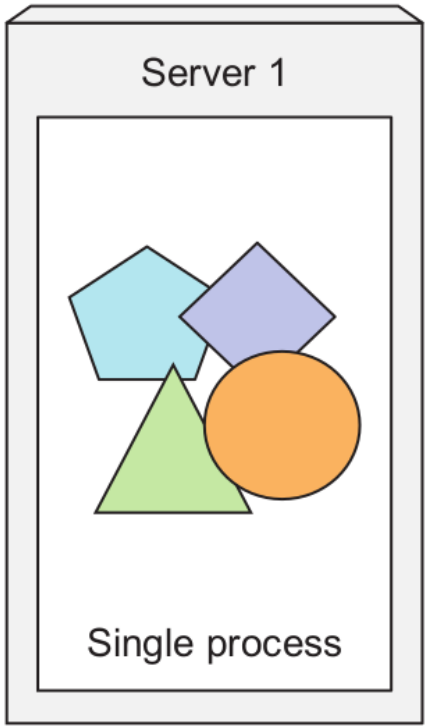
- Smaller
- Independently

running components

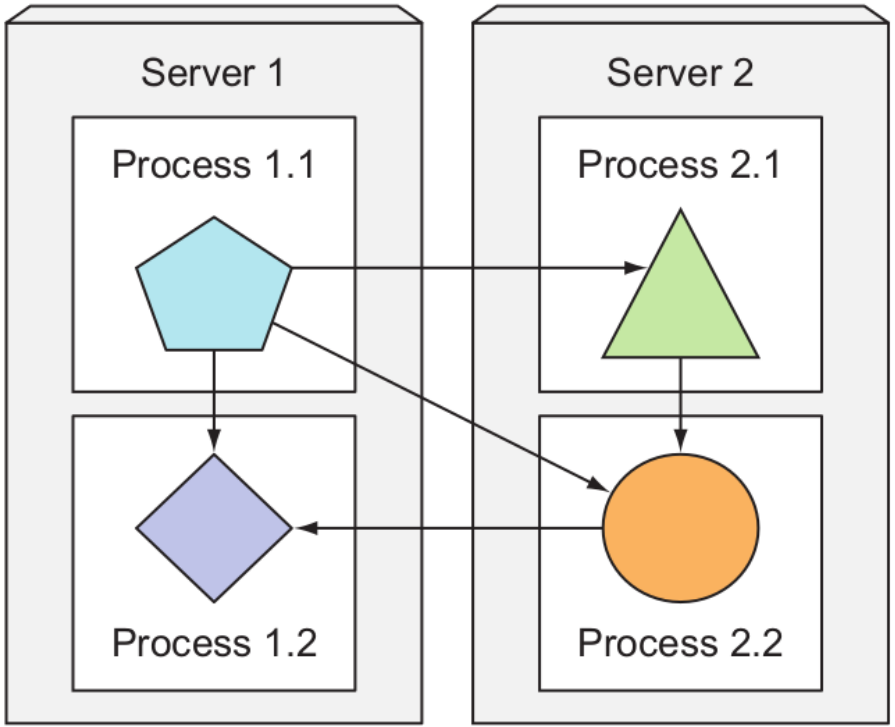
Monolithic application



Monolithic application



Microservices-based application

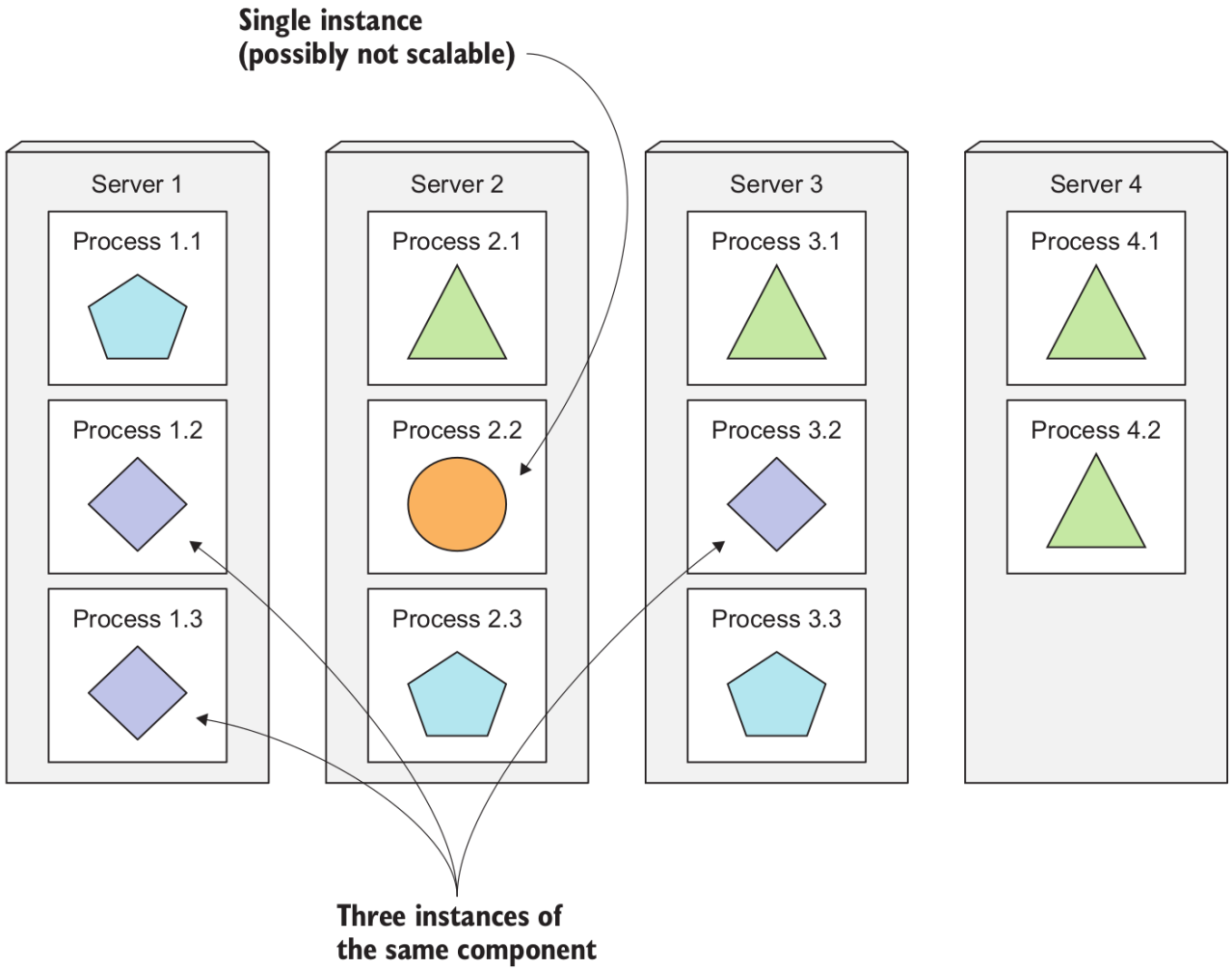


Monolith

- Tightly coupled: Developed, Deployed, and Managed as one entity
- Changes to one part require a redeployment
- Requires powerful servers
- Have to vertically scale (CPU, RAM, ...)
- Horizontal Scale: Setting Up additional servers + running multiple copies (or replicas) of an application

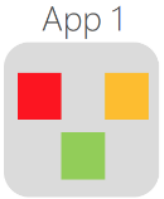
Scaling Microservices

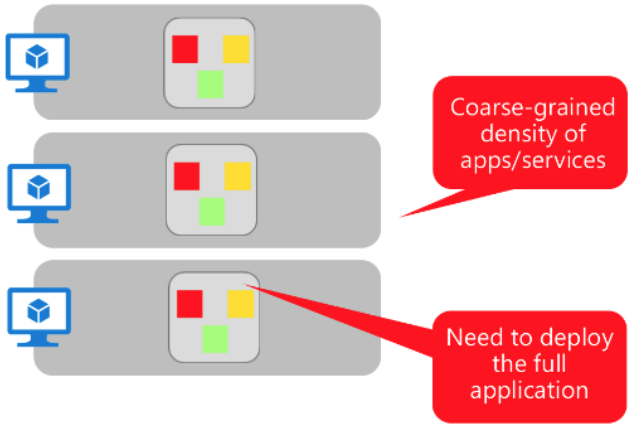
- Scale is done on a per-service basis
- Scaling only those services that require more resources
- Leaving others at their original scale.


Single instance
(possibly not scalable)
Server 1
Process 1.1
Process 1.2
Process 1.3
Server 2
Process 2.1
Process 2.2
Process 2.3
Server 3
Process 3.1
Process 3.2
Process 3.3
Server 4
Process 4.1
Process 4.2
Three instances of
the same component

Monolithic deployment approach

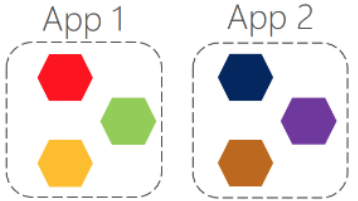
- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs

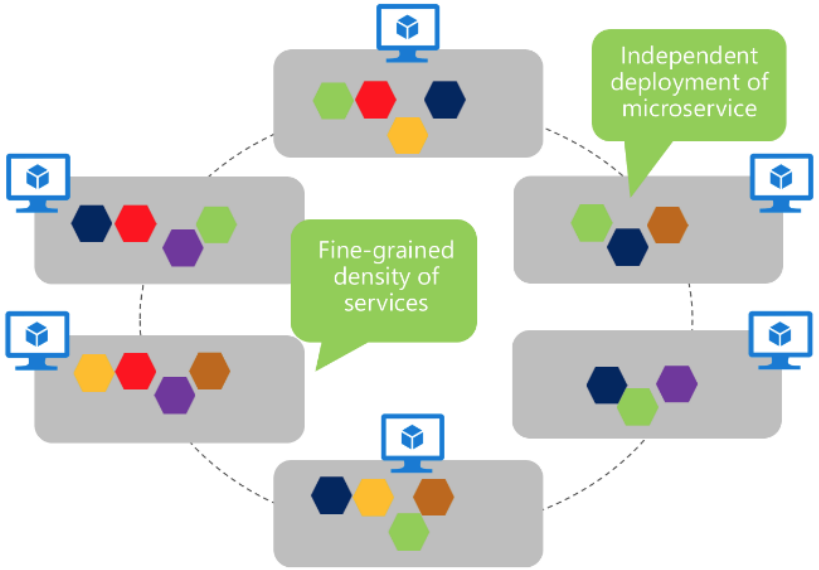




Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs

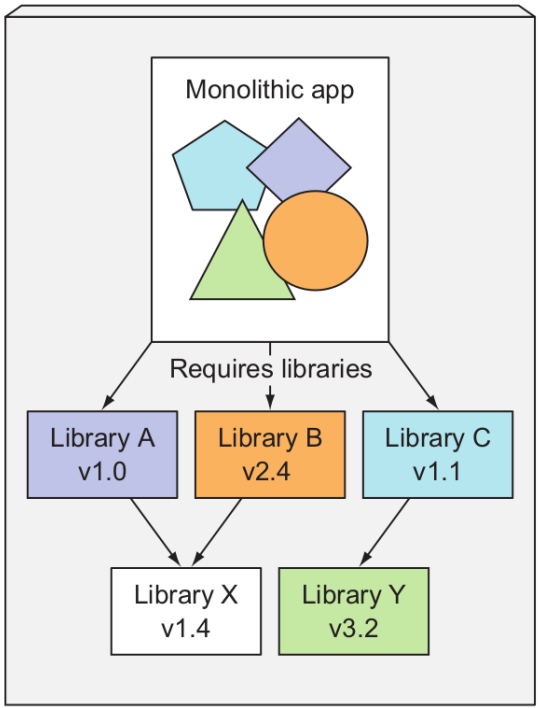




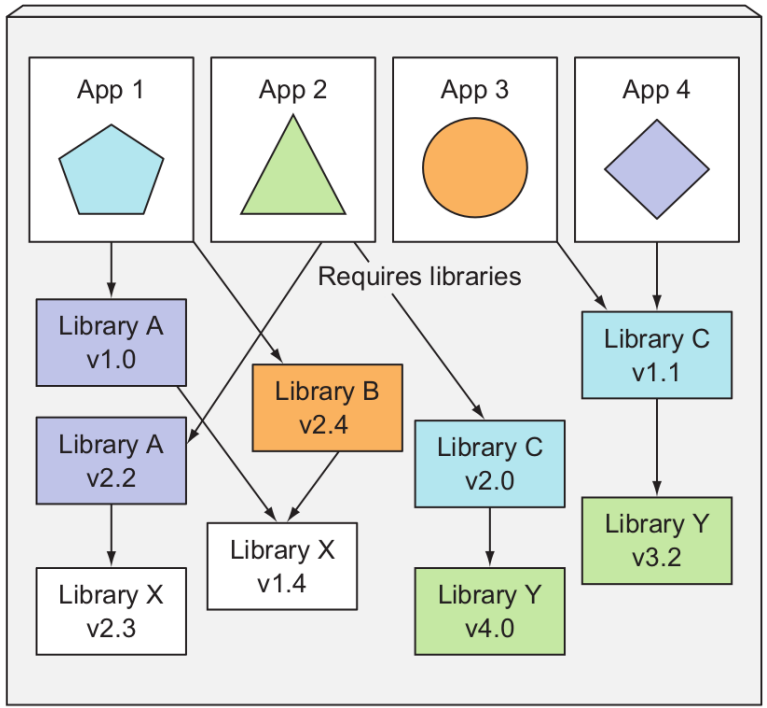
https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture

Divergence of Environment Requirements

Server running a monolithic app

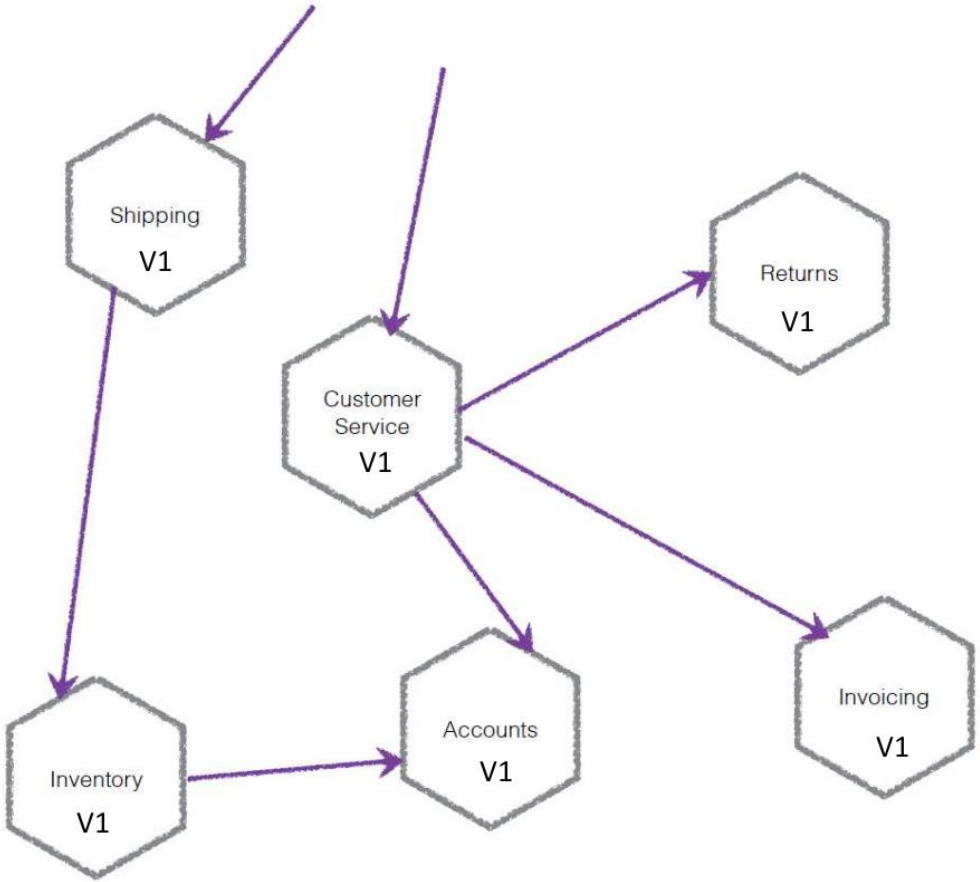


Server running multiple apps

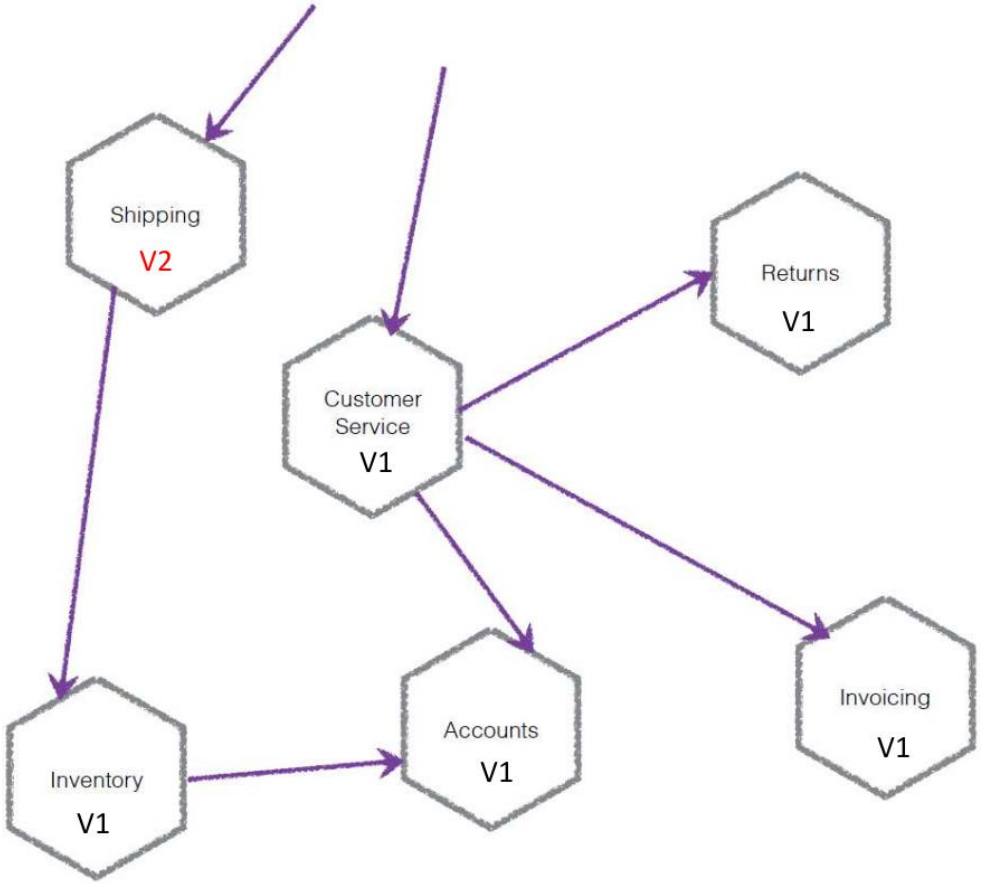


Independently Deployable

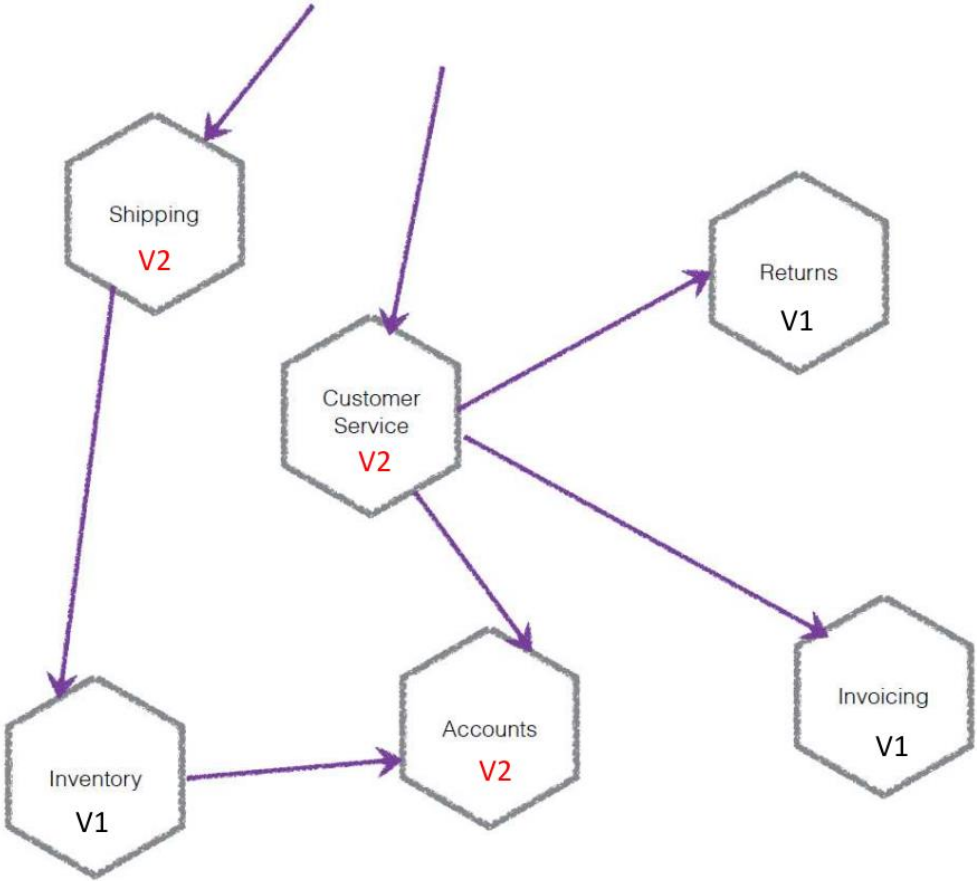
- No lock-step build and deployment
- Not as big as a server app that needs to be built and deployed as a single block



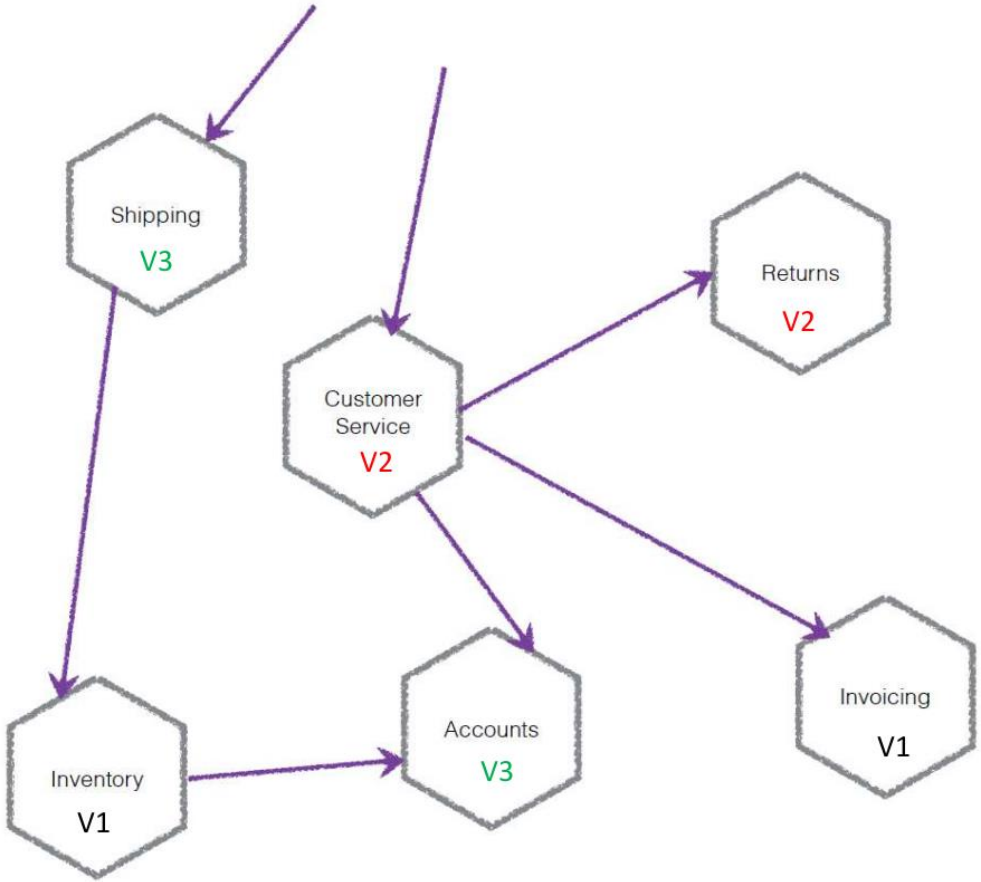
In the begining everything works fine



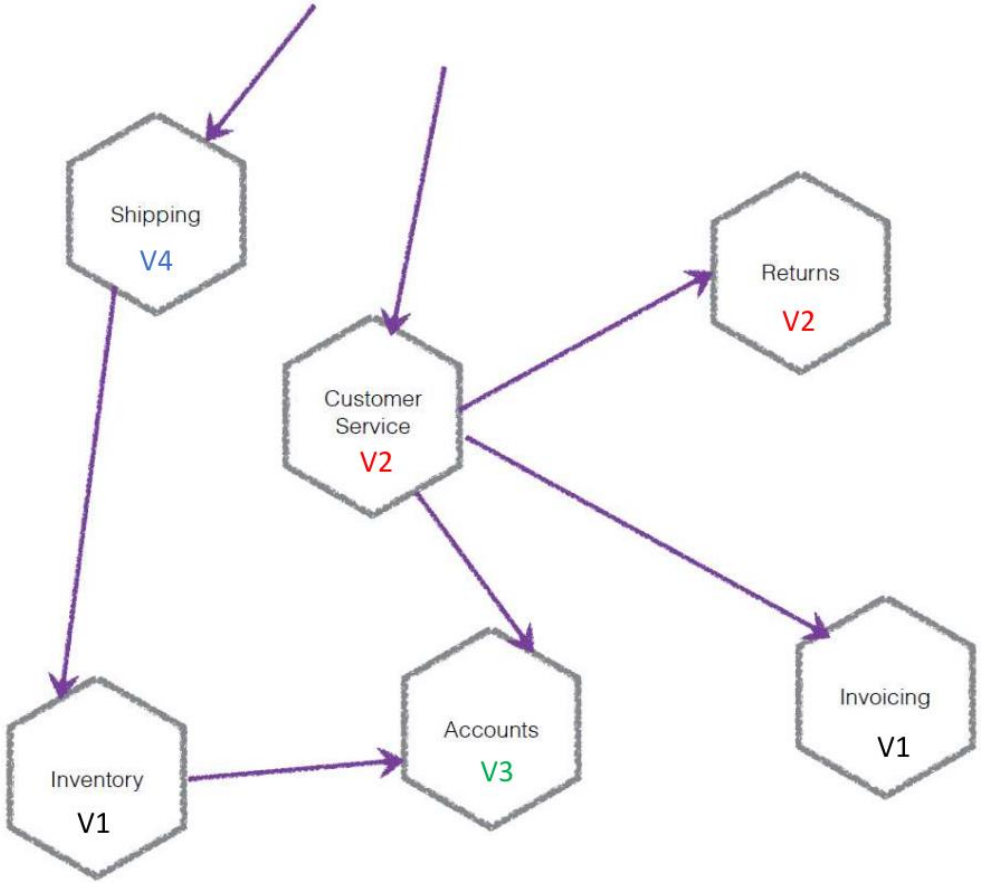
Fortunatelly, no one else needs to know about it.



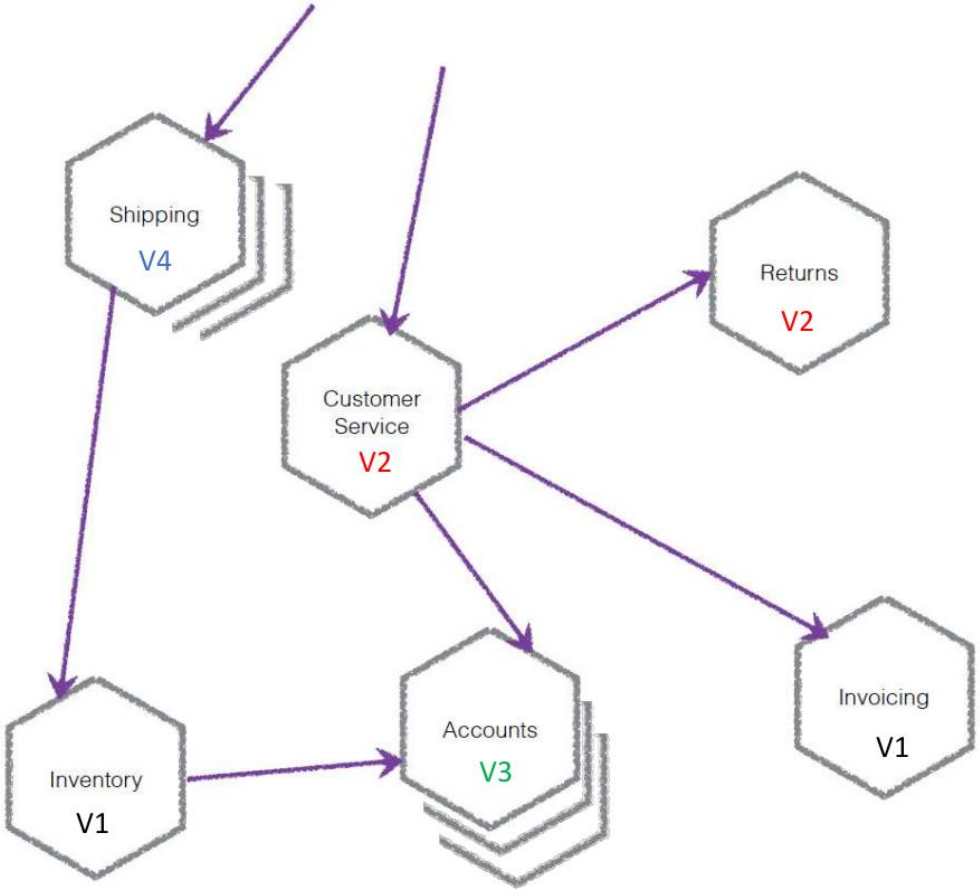
And it happens again



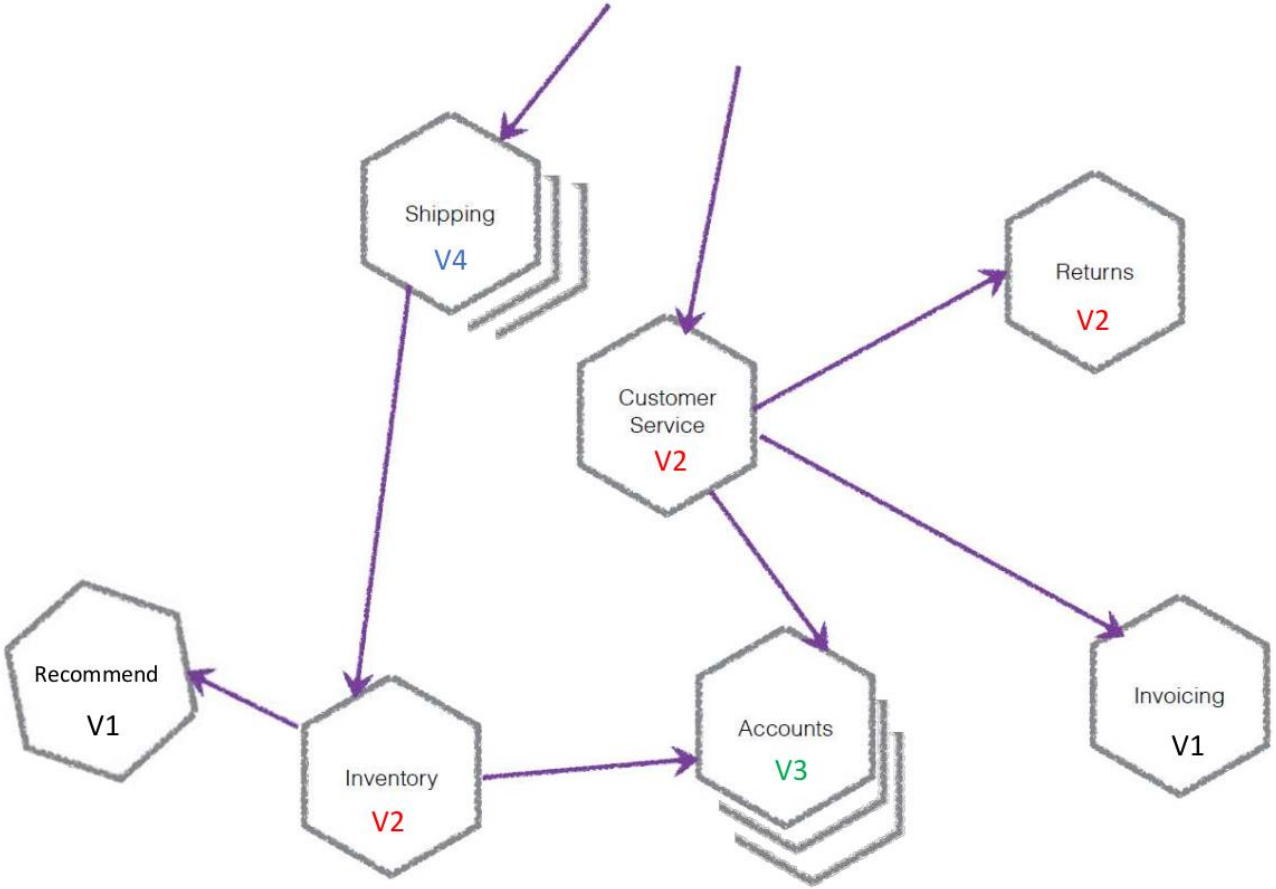
And we also want to refactor and update the technology stack



And enhance the service



And scale the parts of the system that really need extra power



And add new features

Benefits

- Focus on one thing and do it right
- Release functionality faster
- Independent scaling
- Technology diversity; Adopt technologies faster
- Enable resiliency by designing for failure
- Moving to continuous delivery: DevOps and NoOps

Requires

- Automation
- High Cohesion
  - Things that change together, stay together
- Loose coupling
- Stable versioned APIs (Maturity)

Downside

- Cognitive overloading (many tooling options)
- Cognitive overloading (system understanding)
- Testing is more complicated
- Monitoring is more complex
- Operational overhead
- Resiliency isn’t free

2. Container

Introducing container technologies

Introduction

- Different components on the **same machine** will require different libraries/envs (possibly confilicting)
- Large components: dedicated VM to each component and isolate their environments + own OS
- When Number of Components start to grow: can’t give each of them their own VM
  - Waste hardware Resources
  - Must keep your hardware costs down

Isolating Components with Linux Container Technology

- Instead of using VMs to isolate the environments of each microservice
- Rrun multiple services on the same host machine
- Isolating them from each other, similarly to VMs, but with much less overhead

Isolating Components with Linux Container Technology

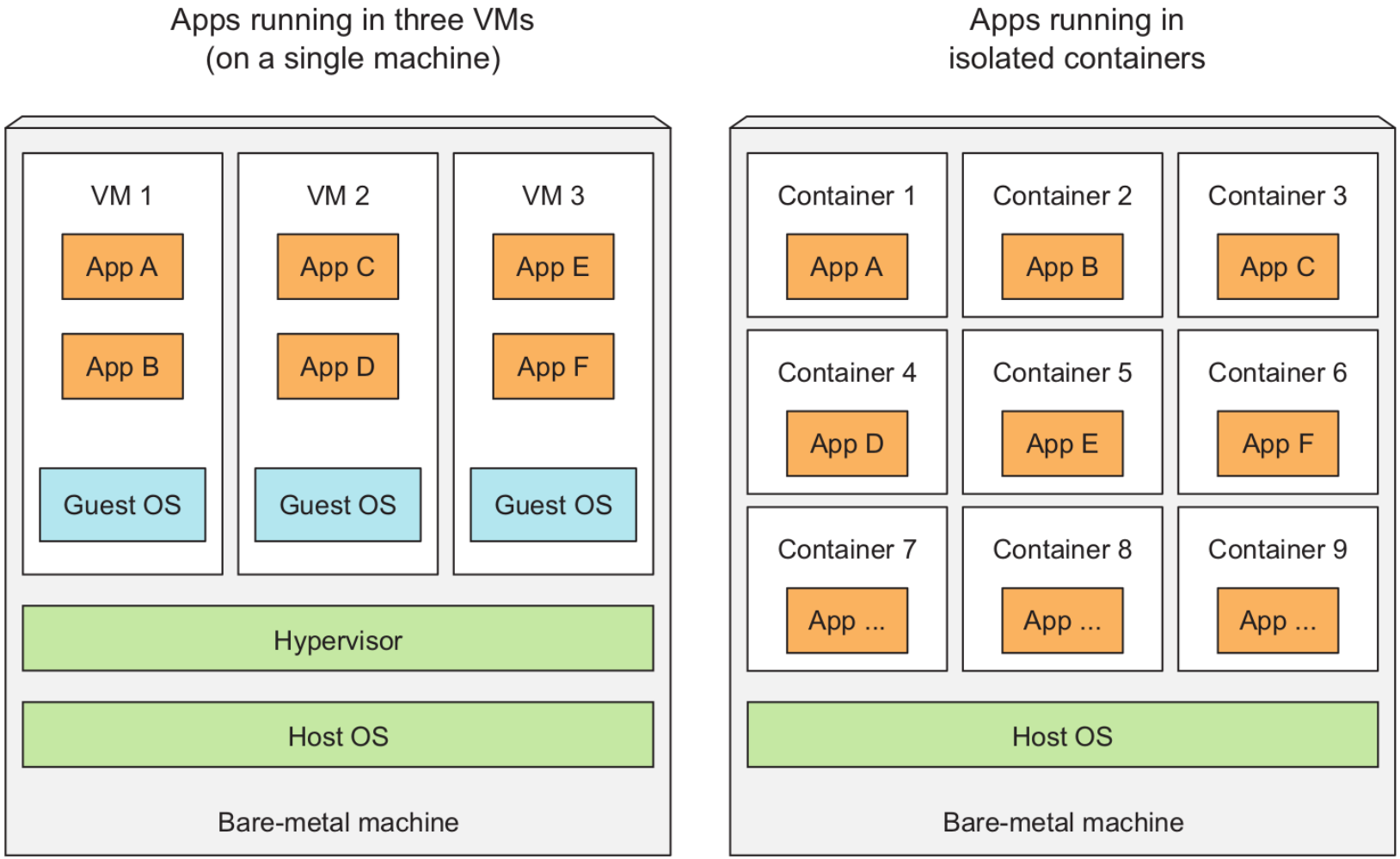
- A process running in a container runs inside the host's operating system, like all the other processes
- Process in the container is still isolated from other processes
- To the process itself, it looks like it's the only one running on the machine and in its OS

Comparing VMs to Container

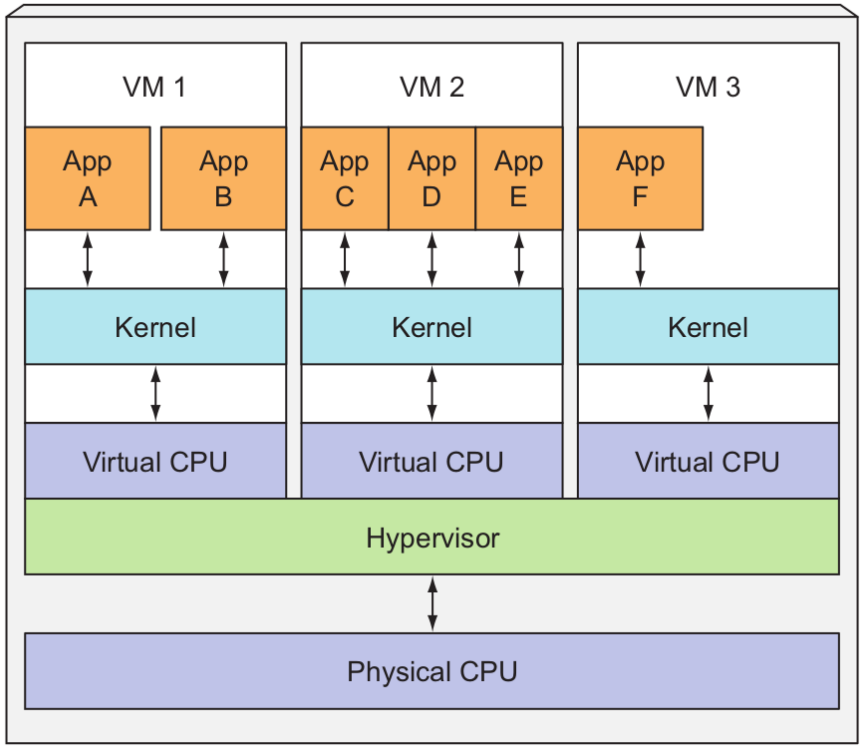
- Containers are much more lightweight
- Allows you to run higher numbers of software components on the same hardware
- VM requires additional compute resources

“

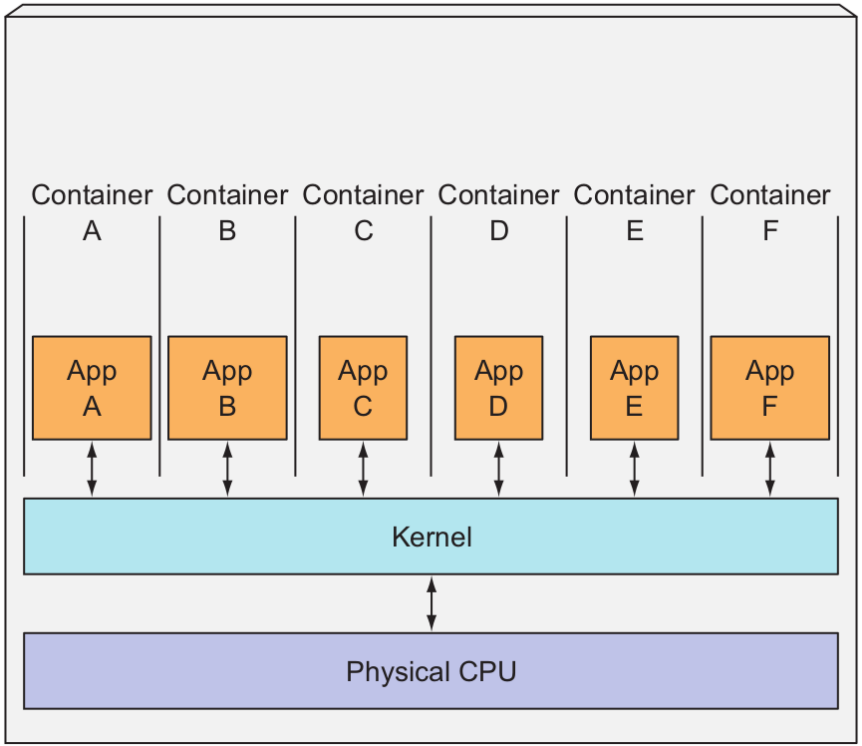
A container, is nothing more than a single isolated process running in the host OS, consuming only the resources that the app consumes and without the overhead of any additional processes.


Apps running in three VMs
(on a single machine)
VM 1
App A
App B
Guest OS
VM 2
App C
App D
Guest OS
VM 3
App E
App F
Guest OS
Hypervisor
Host OS
Bare-metal machine
Apps running in
isolated containers
Container 1
App A
Container 2
App B
Container 3
App C
Container 4
App D
Container 5
App E
Container 6
App F
Container 7
App ...
Container 8
App ...
Container 9
App ...
Host OS
Bare-metal machine

Apps running in multiple VMs



Apps running in isolated containers



Introducing The Mechanisms That Make Container Isolation Possible

- Linux Namespaces
- Linux Control Groups (cgroups)

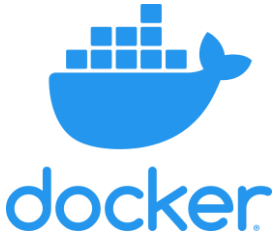
Linux Namespaces

Makes sure each process sees its own personal view ofthe system

(files, processes, network interfaces, hostname, and so on)

Linux Control Groups (cgroups)

Limit the amount of resources the process can consume (CPU, memory, network bandwidth, and so on)



3. Docker

Introducing the Docker container platform

Docker

- The First container system
- Made containers easily portable across different machines
- Simplified the process of packaging up
- Also all its libraries and other dependenciescontainer images are composed of layers
  - Can be shared and reused across multiple images

Docker

Docker is a platform for packaging, distributing, and running applications.

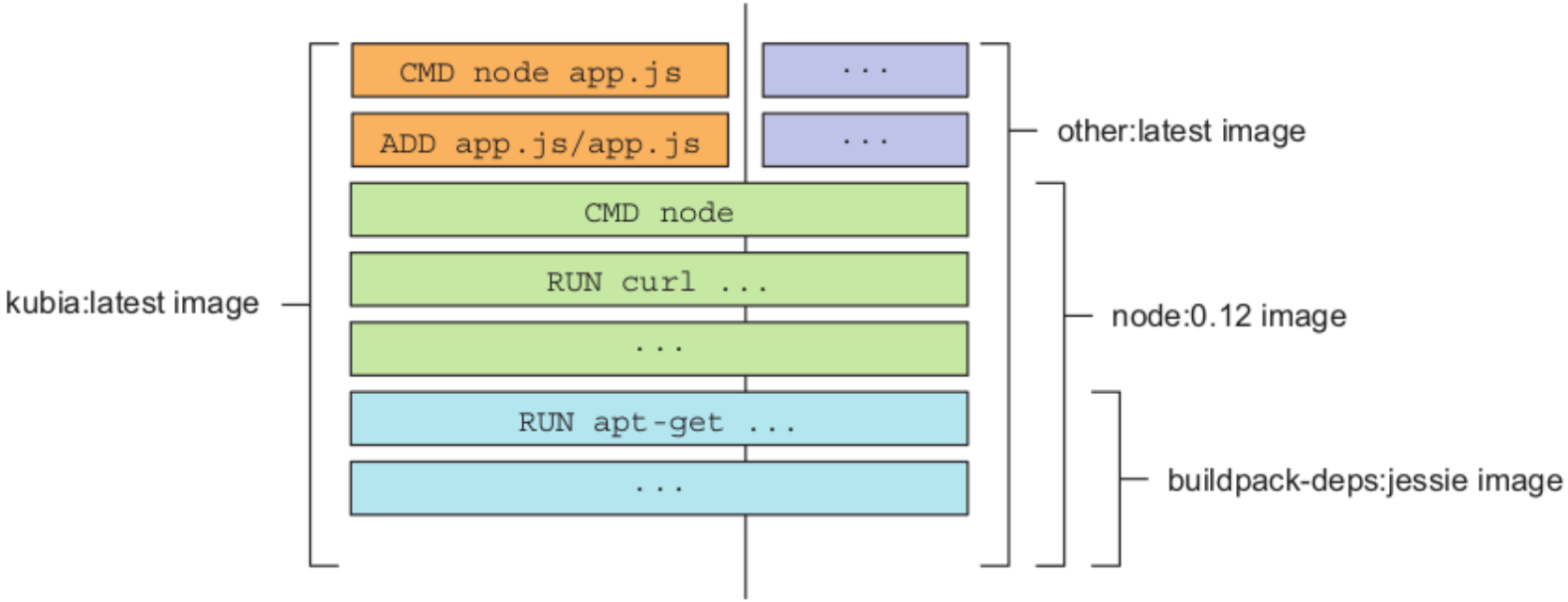
Docker Concepts

- Images
- Registries
- Containers

Images

something you package your application and its environment into. It Contains the filesystem that will be available to the application

Images


CMD node app.js
...
ADD app.js/app.js
...
CMD node
RUN curl ...
...
RUN apt-get ...
...
kubia:latest image
other:latest image
node:0.12 image
buildpack-deps:jessie image

Registries

Stores your Docker images and facilitates easy sharing.

You can push (upload) the image to a registry and then pull (download) it on another computer and run it there.

Container

A Docker-based container is a regular Linux container created from a Docker-based container image.

“

A running container is a process running on the host running Docker.

it’s completely isolated from both the host and all other processes running on it.

“

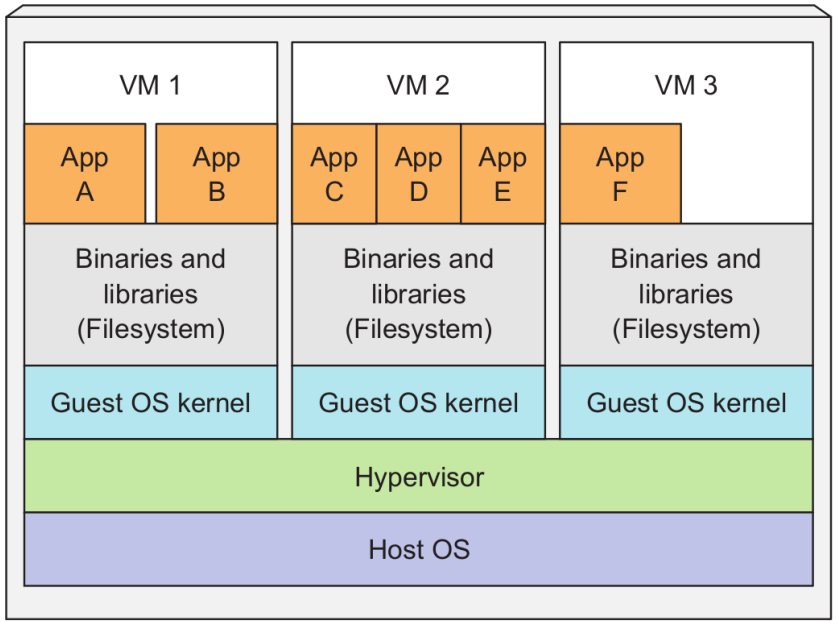
The process is also resource-constrained, meaning it can only access and use the amount of resources

(CPU, RAM, and so on)

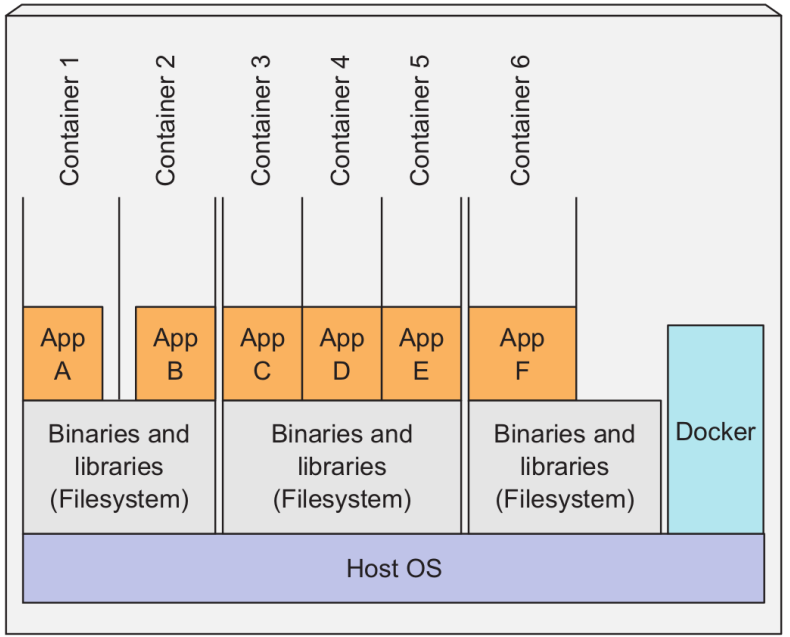
that are allocated to it.

VMs Vs Docker Containers

Host running multiple VMs



Host running multiple Docker containers



4. Kubernetes

Introducing Kubernetes

History

- Google developed an internal system called *Borg* (and later a new system called *Omega*)
- Helped both developers and system administrators **manage** those **thousands** of **applications** and **services**.
- Helped them achieve a much higher utilization of their infrastructure
  - Even tiny improvements in utilization mean savings in the millions of dollars

History

- After having kept Borg and Omega secret for a whole decade, in 2014 Google introduced Kubernetes
- An open-source system based on the experience gained through Borg, Omega, and other internal Google systems.

Introduction

- Allows you to easily deploy and manage containerized Apps
- Relies on the features of Linux containers
- Without having to know any internal details
- Without having to manually deploy these applications on each host
- Apps don’t affect other apps running on the same server

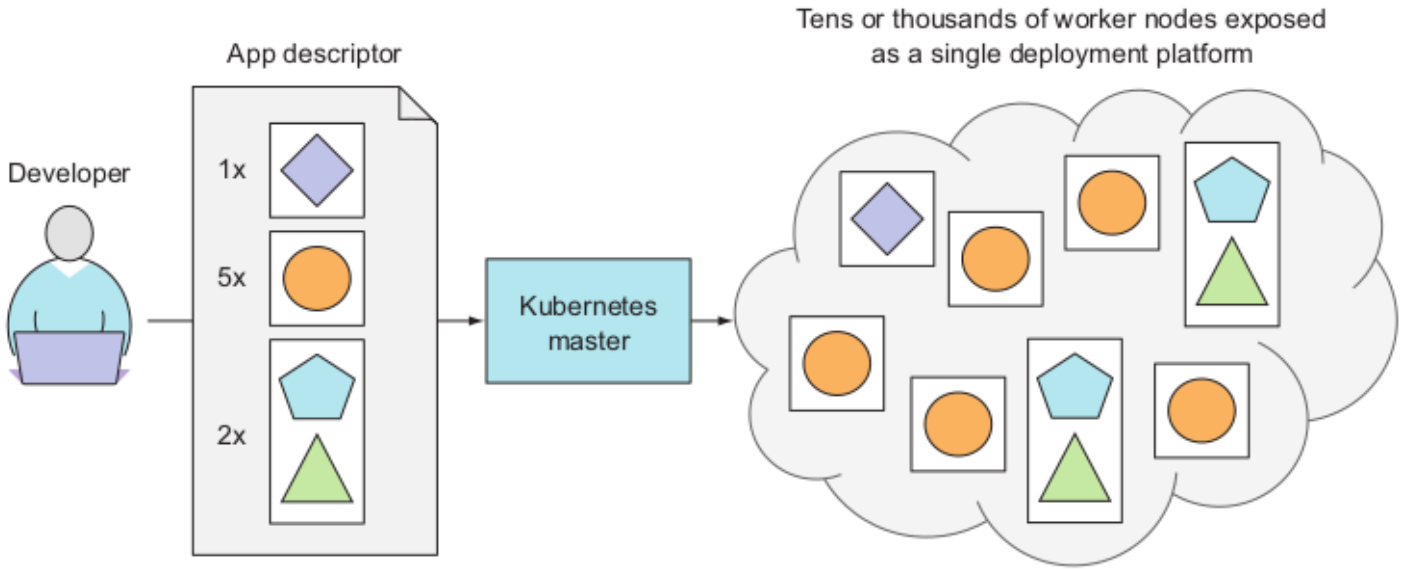
Kubernetes Enables you to run your software applications on thousands of computer nodes

Deploying apps is always the same, whether cluster contains only a couple or thousands of nodes

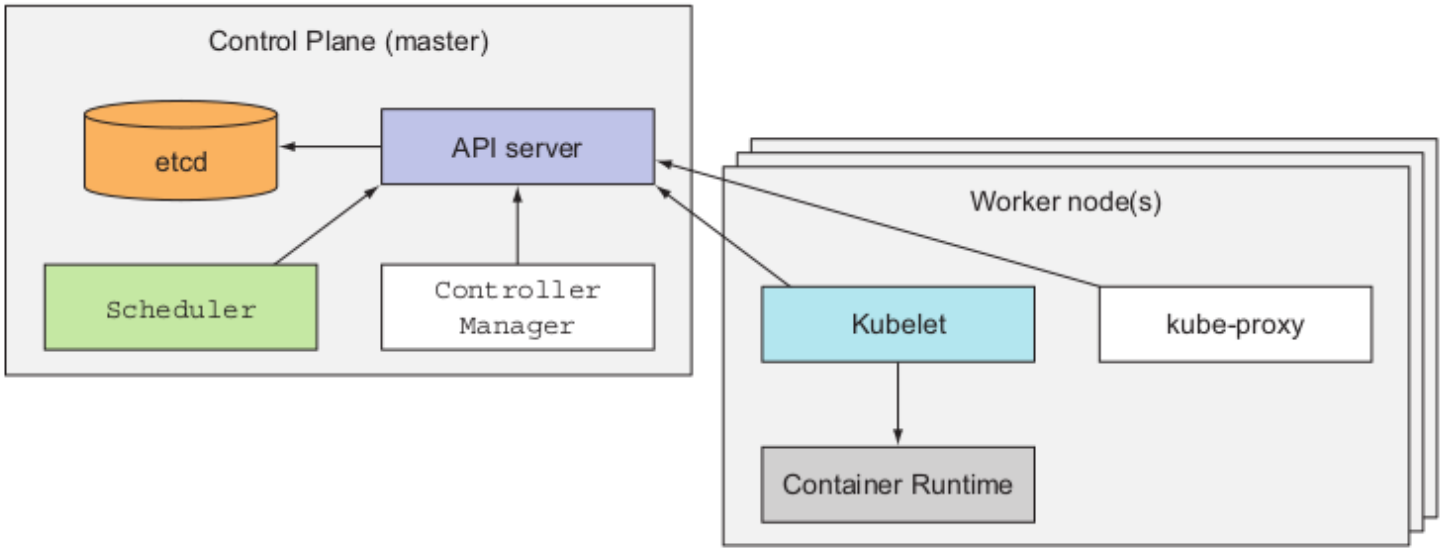
Introduction

- Helping Developers Focus On The Core App Features
  - Service discovery, scaling, load-balancing, self-healing
  - Focus on implementing the actual features
- Helping Ops Teams Achieve Better Resource Utilization
  - Run containerized app somewhere in the cluster
  - Keep all of components running
  - Better resource utilization than is possible with manual scheduling

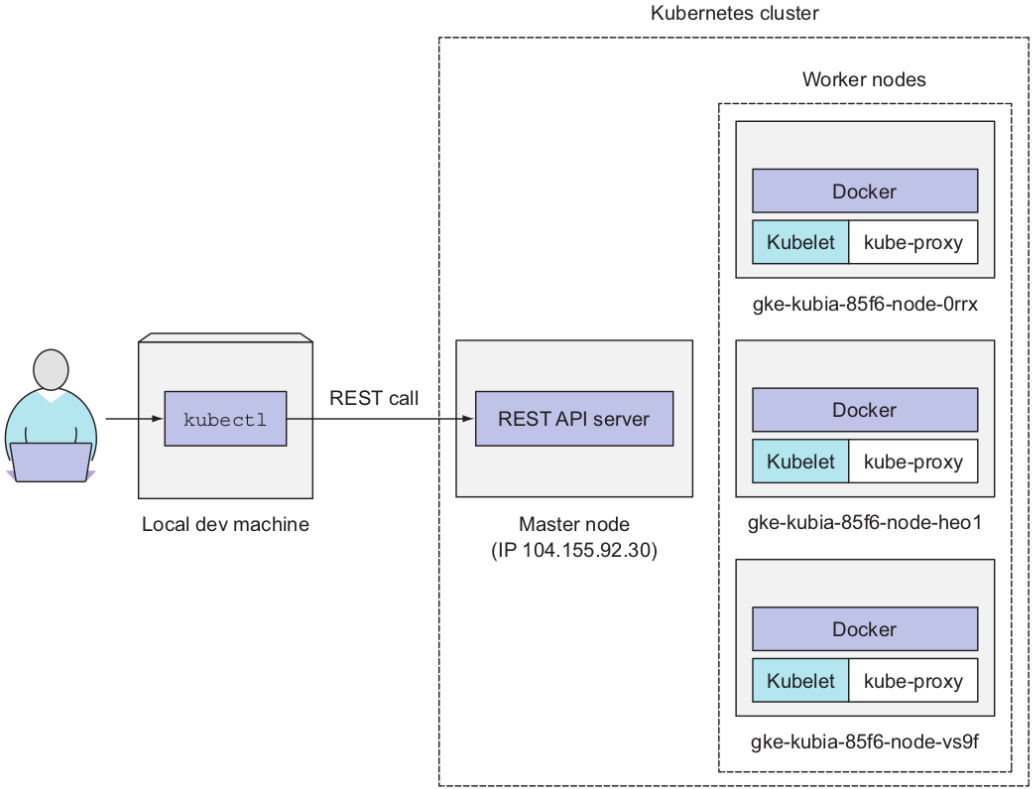
What Kubernetes Does



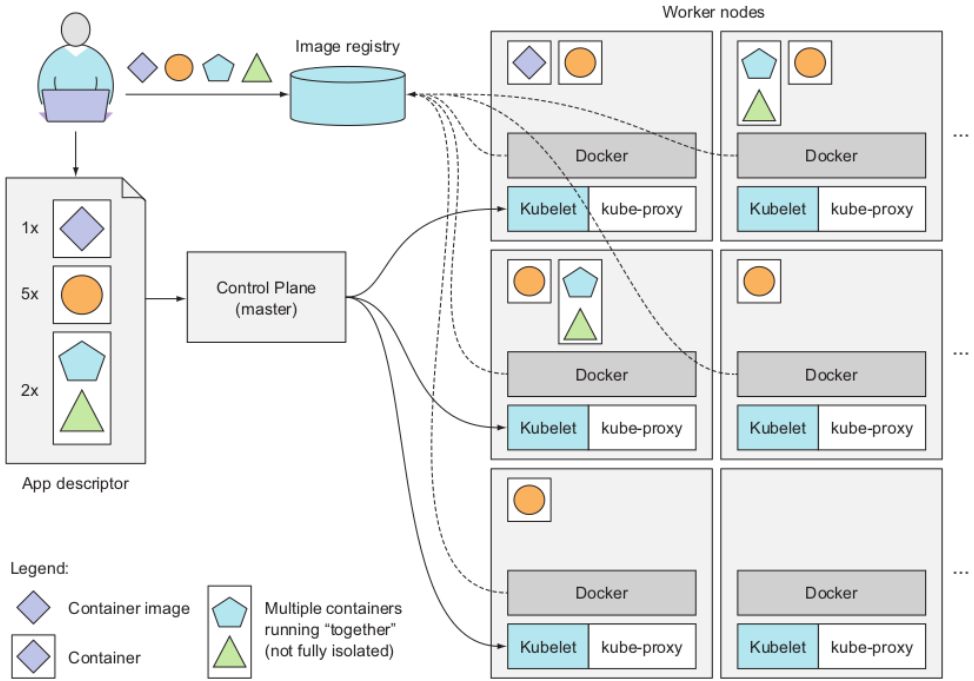
Architecture

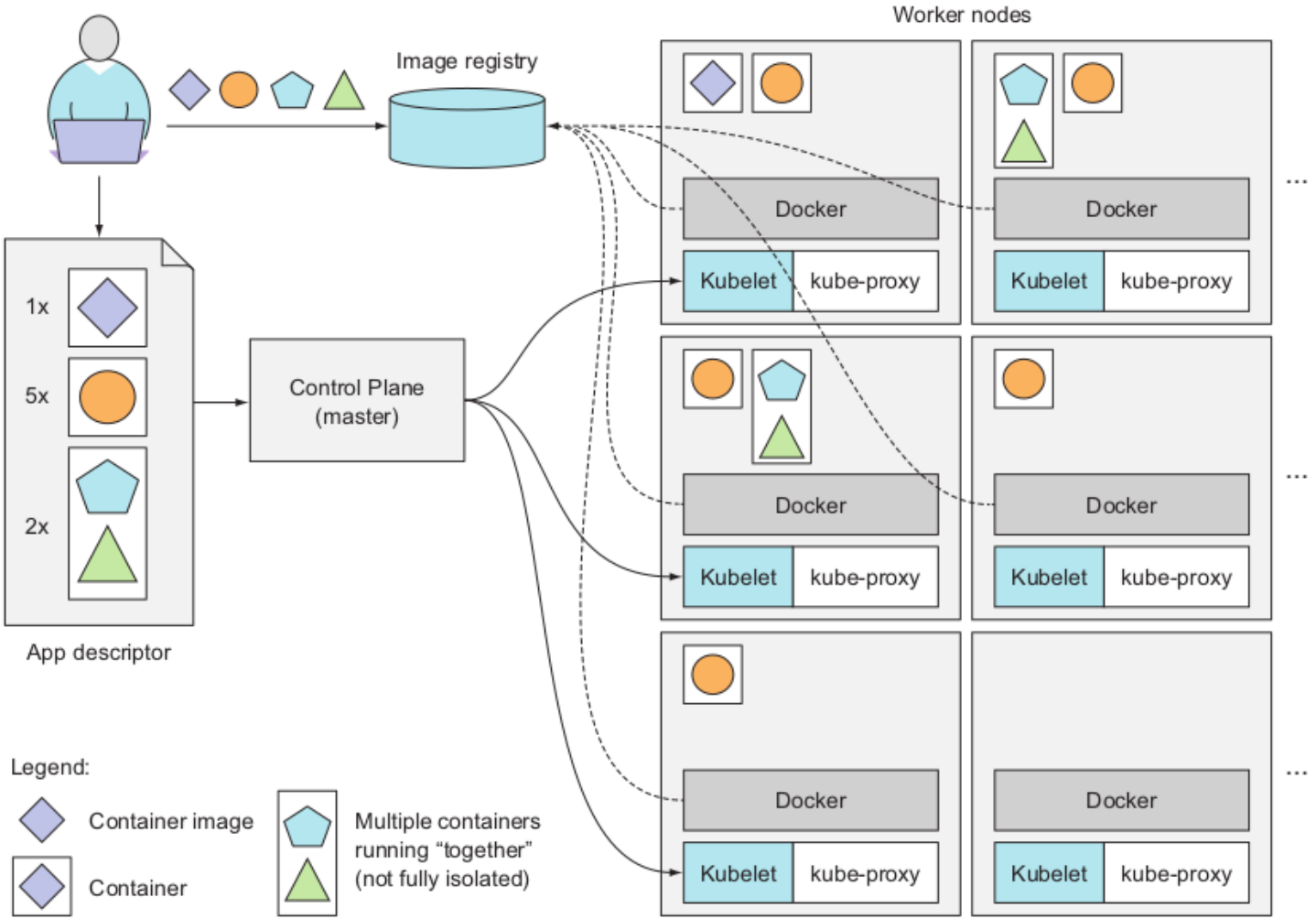


Interacting with Kubernetes cluster

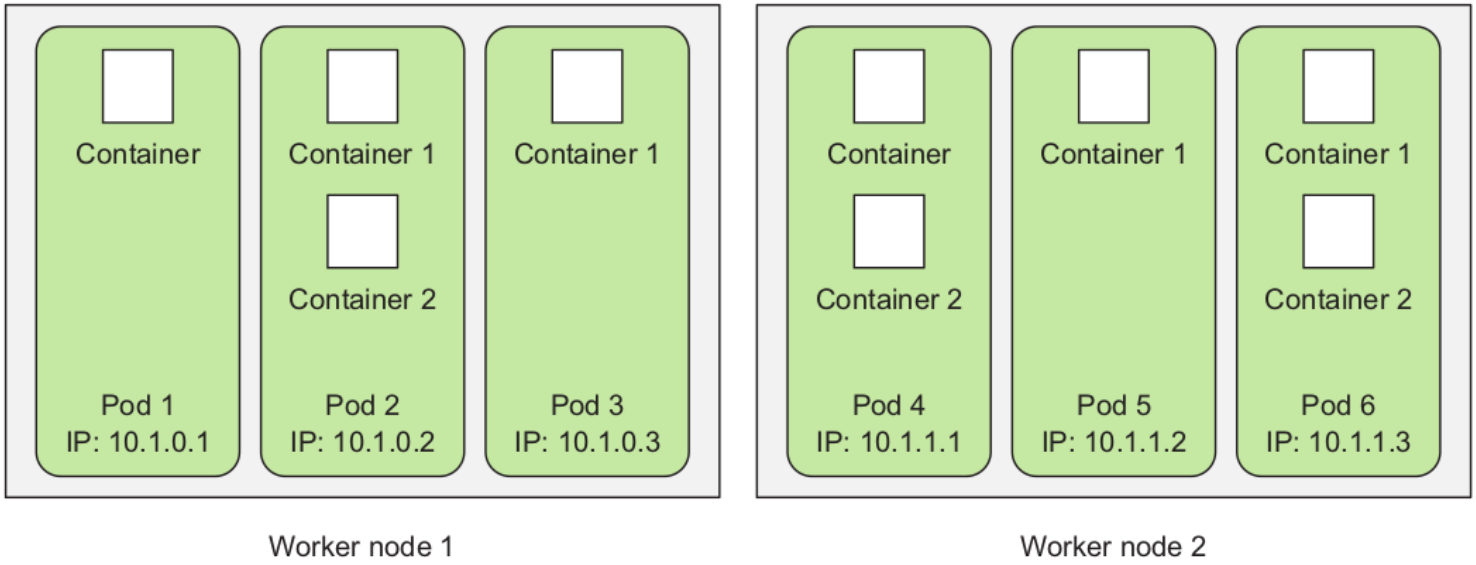


Running an Application in Kubernetes

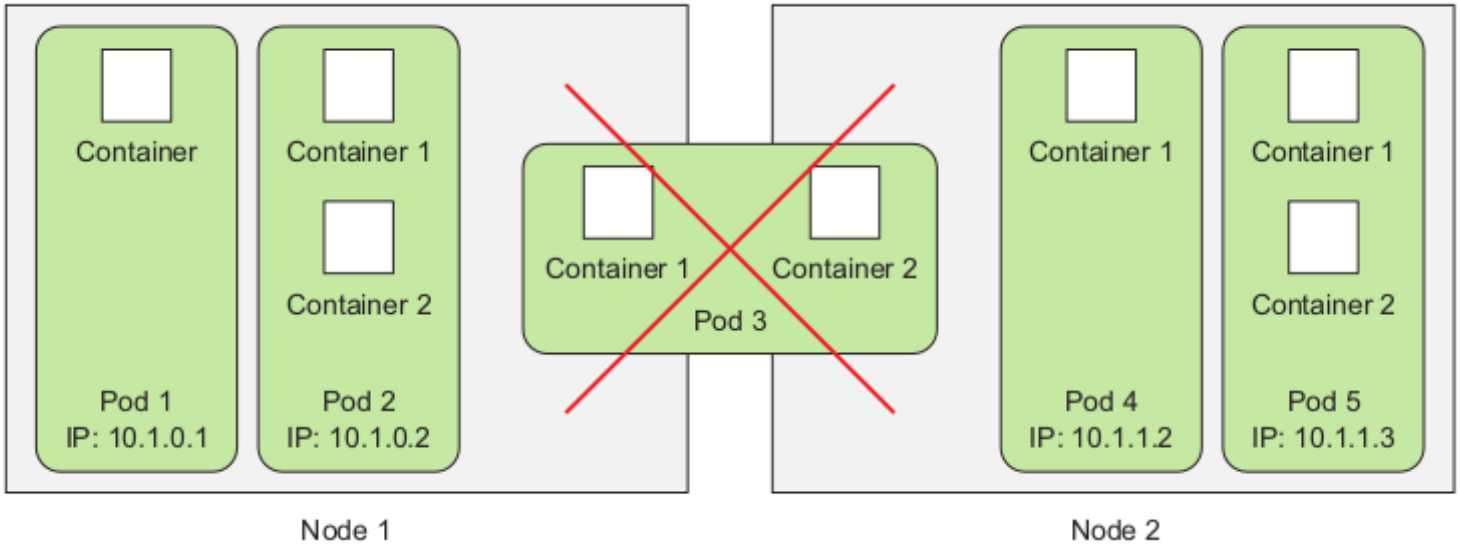



Worker nodes
Image registry
Docker
Kubelet
kube-proxy
1x
5x
2x
Control Plane
(master)
App descriptor
Legend:
Container image
Container
Multiple containers
running "together"
(not fully isolated)

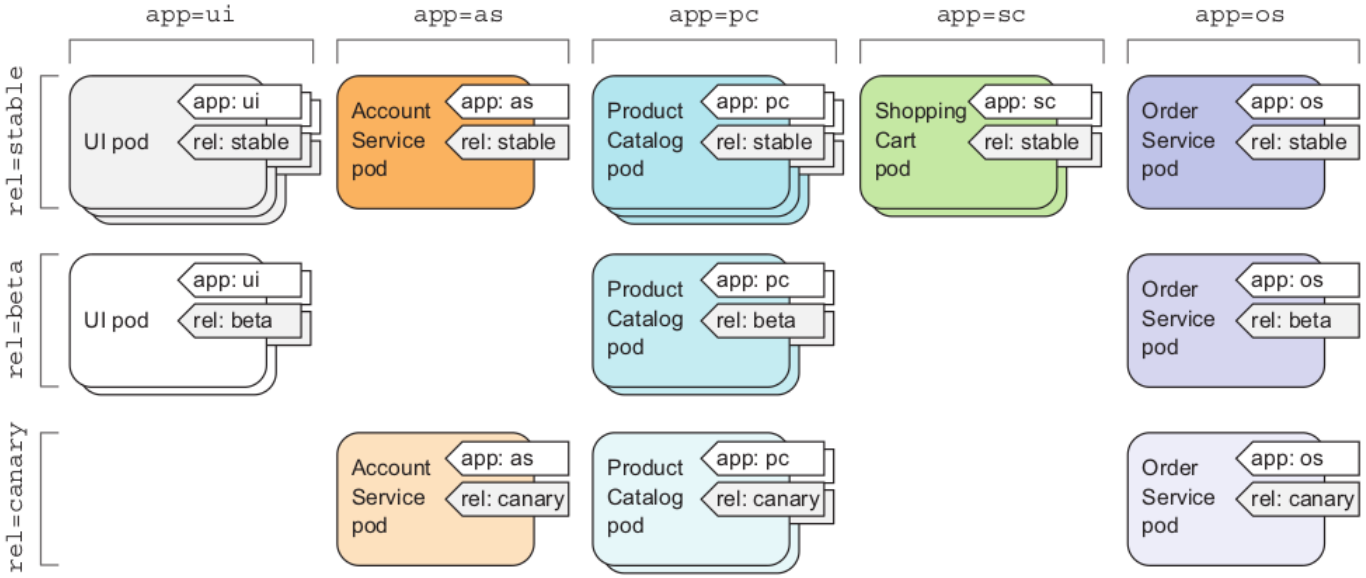
Pods


Container
Pod 1
IP: 10.1.0.1
Container 1
Container 2
Pod 2
IP: 10.1.0.2
Container 1
Pod 3
IP: 10.1.0.3
Worker node 1
Container
Container 2
Pod 4
IP: 10.1.1.1
Container 1
Pod 5
IP: 10.1.1.2
Container 1
Container 2
Pod 6
IP: 10.1.1.3
Worker node 2

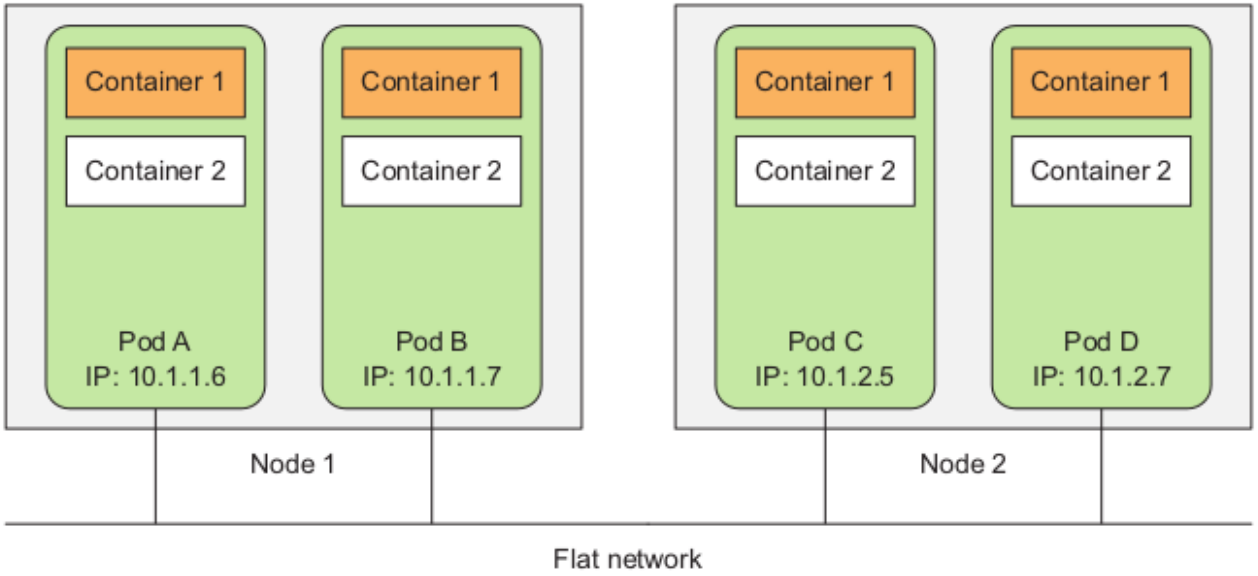
Pods


Container
Pod 1
IP: 10.1.0.1
Container 1
Container 2
Pod 2
IP: 10.1.0.2
Container 1
Container 2
Pod 3
Node 1
Container 1
Container 1
Pod 4
IP: 10.1.1.2
Container 1
Container 2
Pod 5
IP: 10.1.1.3
Node 2

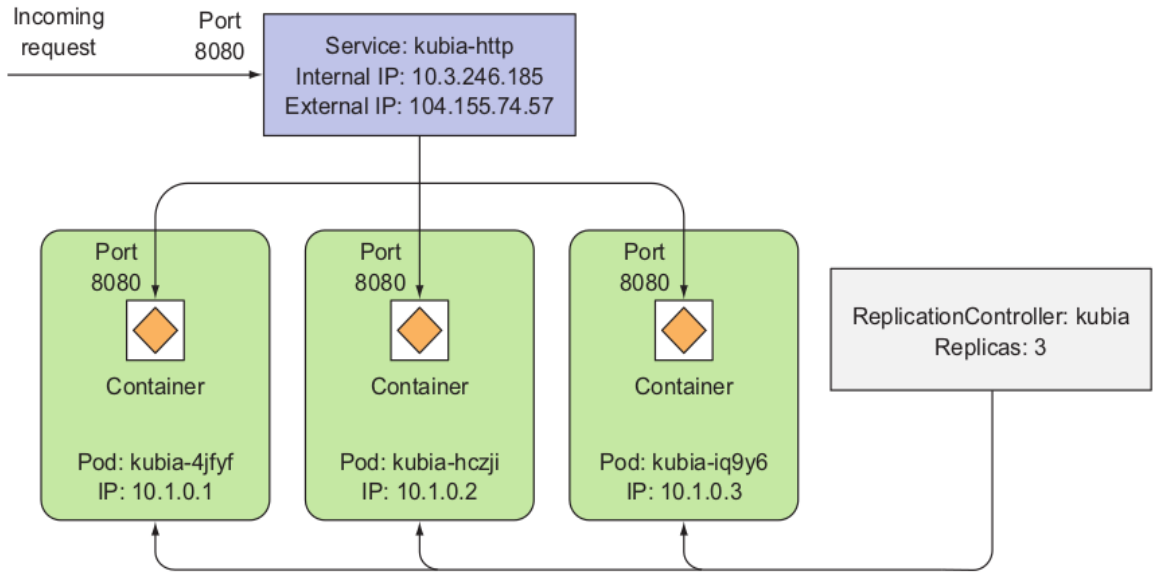
Pods


app=ui
app=as
app=pc
app=sc
app=os
rel=stable
rel=beta
rel=canary
UI pod
app: ui
rel: stable
Account Service pod
app: as
rel: stable
Product Catalog pod
app: pc
rel: stable
Shopping Cart pod
app: sc
rel: stable
Order Service pod
app: os
rel: stable
UI pod
app: ui
rel: beta
Product Catalog pod
app: pc
rel: beta
Order Service pod
app: os
rel: beta
Account Service pod
app: as
rel: canary
Product Catalog pod
app: pc
rel: canary
Order Service pod
app: os
rel: canary

Inter-Pod Network



Replication Controller
Exposed Single service IP and port.



Benefits

- Simplifying Application Deployment
- Achieving Better Utilization Of Hardware
- Health Checking And Self - Healing
- Automatic Scaling
- Simplifying Application Development

Thanks!

Any questions?