Deploying Containerized Workloads Using Google Cloud Kubernetes Engine

INTRODUCING GOOGLE KUBERNETES ENGINE (GKE)

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Containers for lightweight compute

Ideal for hybrid, multi-cloud

Kubernetes container orchestration technology

Industry standard with Google origins

GKE for Kubernetes on GCP

Prerequisites and Course Outline

Software and Skills

Basic understanding of cloud computing

Basic understanding of how virtual machines work

Basic understanding of how containers work

Prerequisites: Basic Cloud Computing

Choosing and Implementing Google Cloud Compute Engine Solutions

- Basics of using the Google Cloud Platform

Docker and Containers: The Big Picture

- Introduction to containers

Course Outline

Introducing GKE

- VMs vs. containers
- Terminology: Pods, kubelets, node pools clusters

Creating and administering clusters

- Clusters using the web console and command line
- Autoscaling, autohealing and autorepairing nodes

Deploying containerized workloads

- Deploying containers and exposing services
- Sharing state with persistent volumes
- User requests with ingress objects
- Binary authorization for attestations

Monitoring clusters with Stackdriver

- Stackdriver Kubernetes monitoring and Prometheus

Scenarios: SpikeySales.com

Hypothetical Online Retailer

- Flash sales of trending products
- Spikes in user traffic

SpikeySales on the GCP

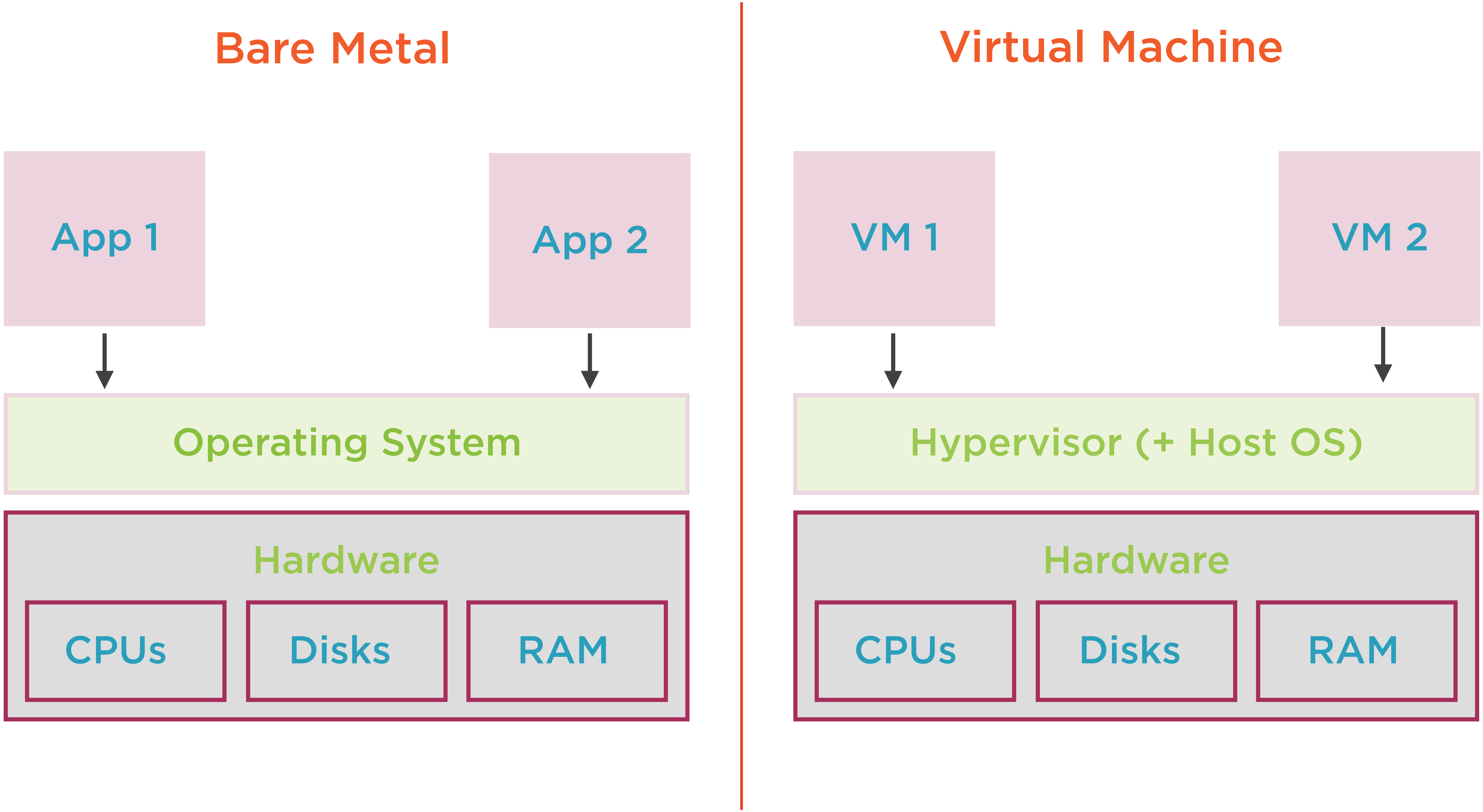
- Cloud computing fits perfectly
- Pay-as-you-go
- No idle capacity during off-sale periods

SpikeySales and Cloud Storage Buckets

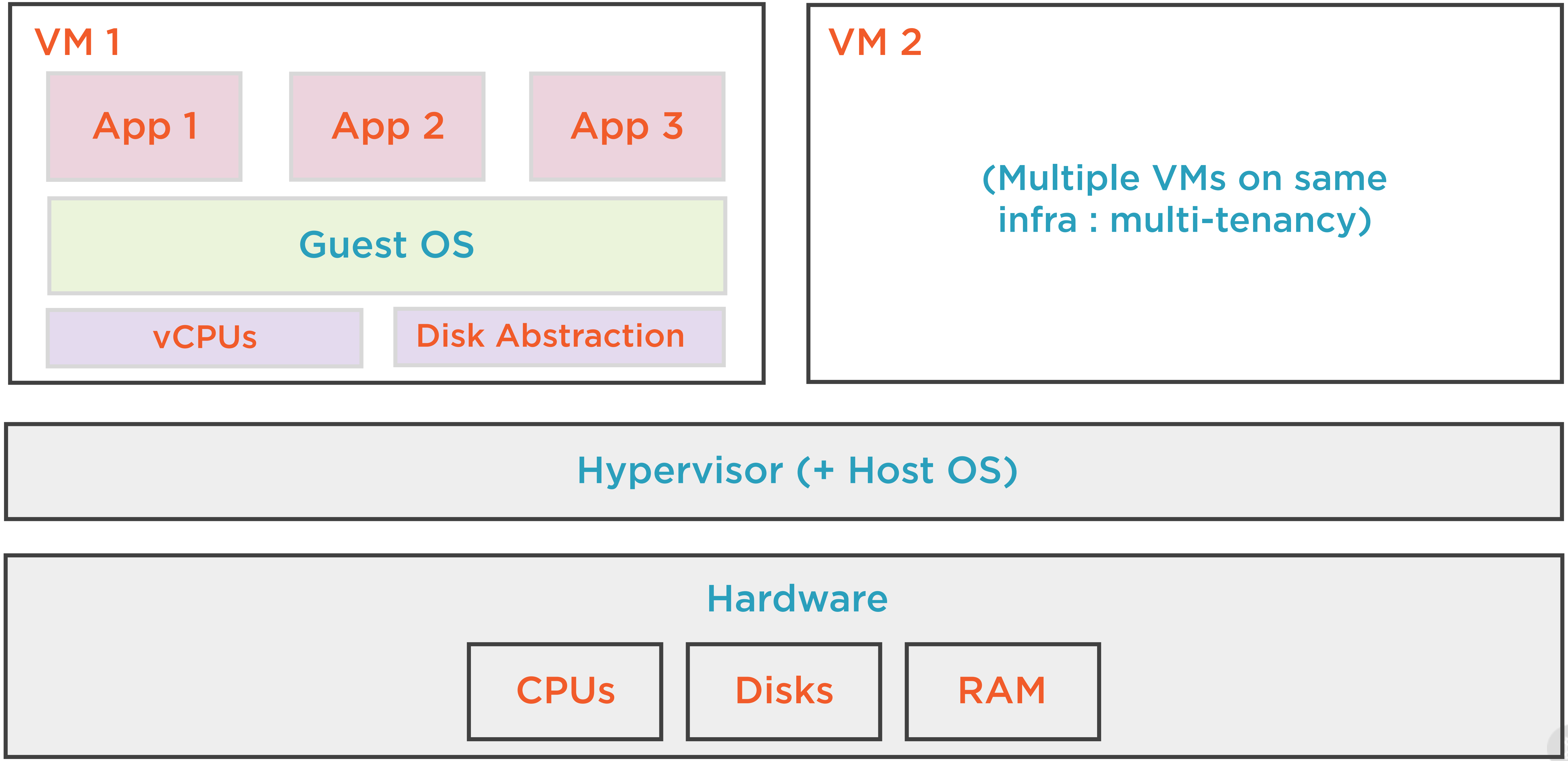
- Move data to the cloud
- Elastic, pay-as-you-go, global access

Introducing Containers

Bare Metal and Virtual Machines



Apps on Virtual Machines



VM 1

App 1

App 2

App 3

Guest OS

vCPUs

Disk Abstraction

VM 2

(Multiple VMs on same infra : multi-tenancy)

Hypervisor (+ Host OS)

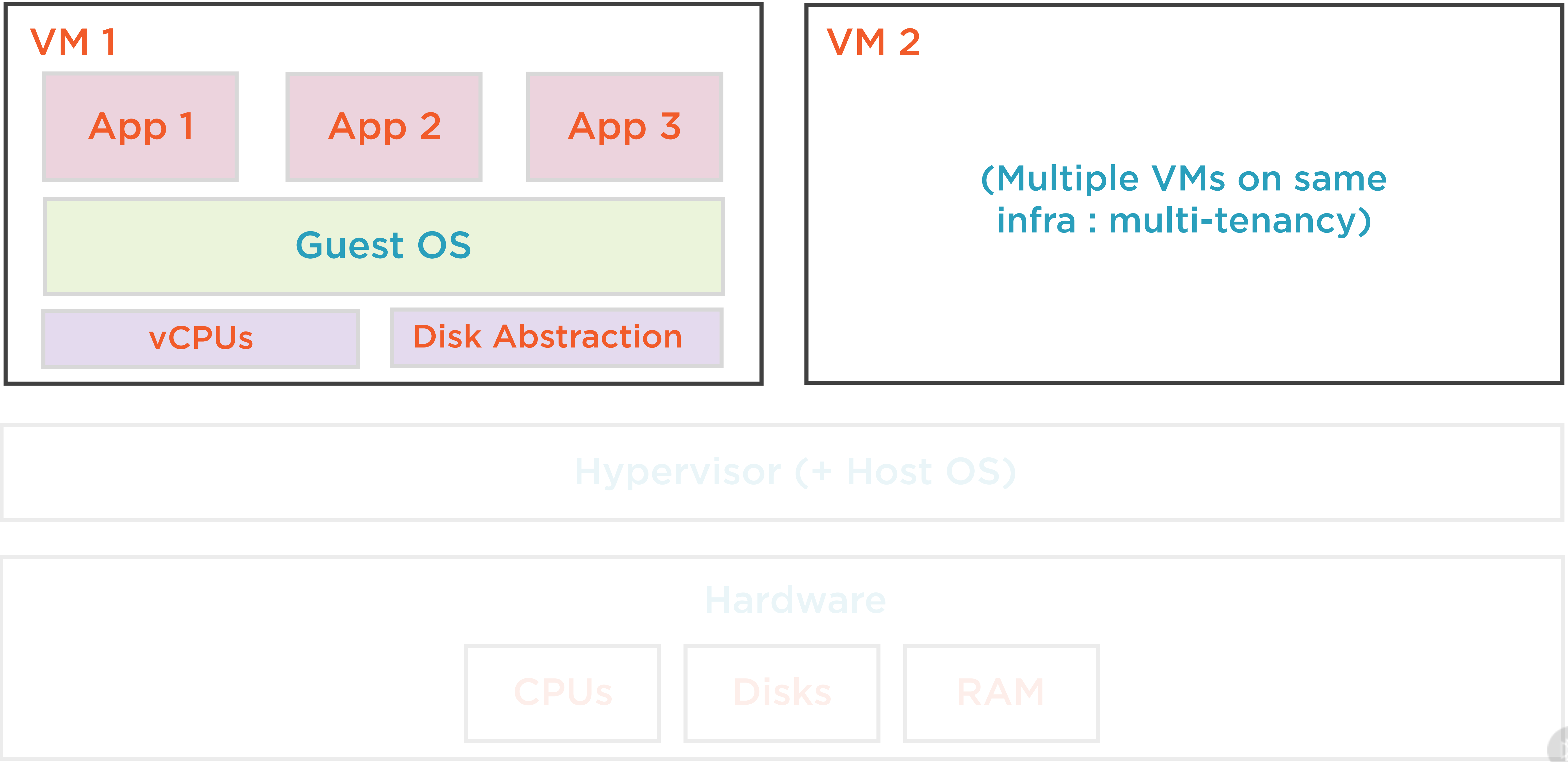
Hardware

CPUs

Disks

RAM

Apps on Virtual Machines



VMs are far more lightweight than bare metal environments and are often used to run applications

Drawbacks of VMs

Contain guest OS

- Introduces platform dependency
- Bloats image size to GB (apps far smaller)

Heavyweight

- Slow to boot up

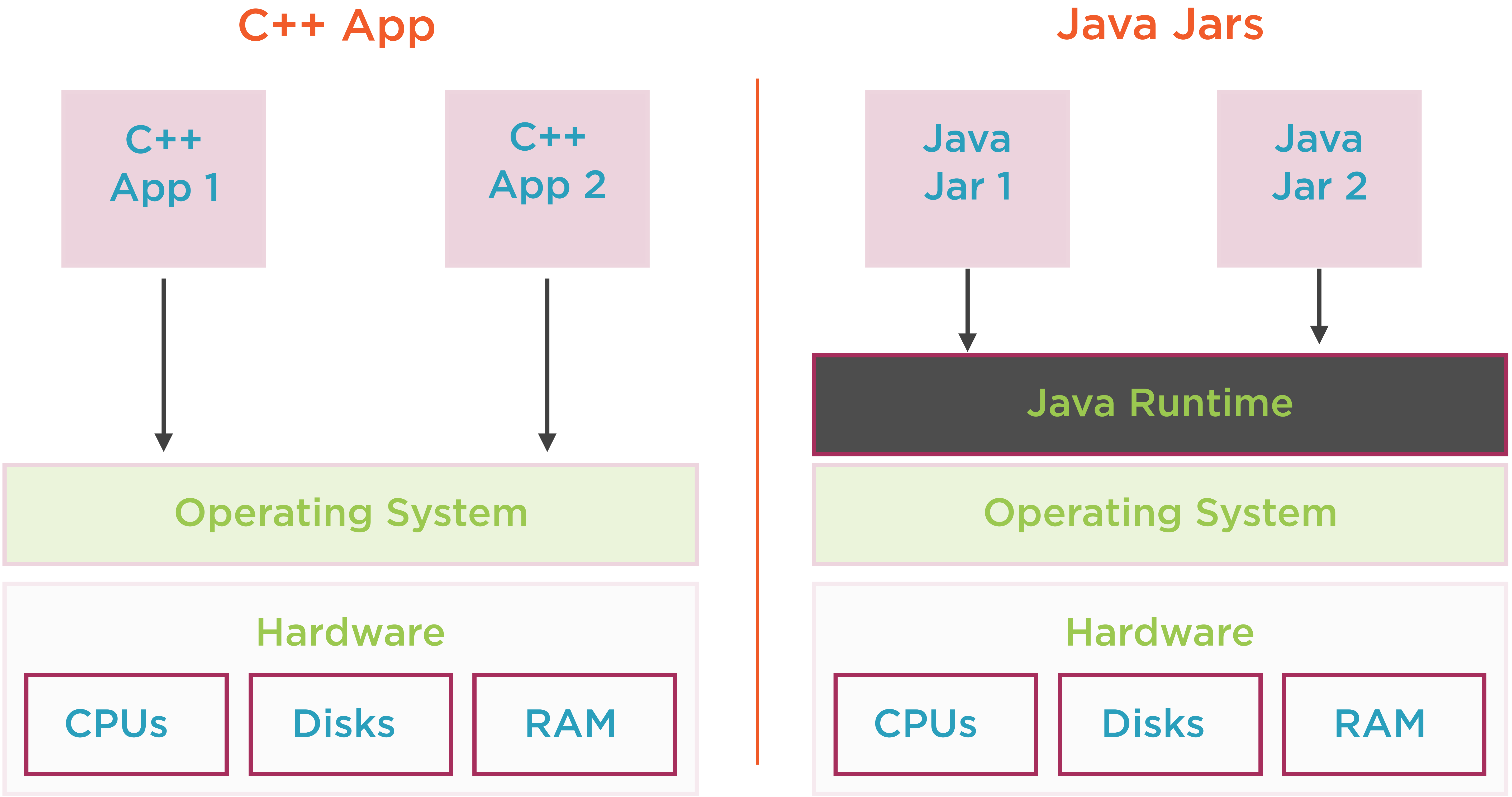
Not trivial to migrate

- VM migration tools needed

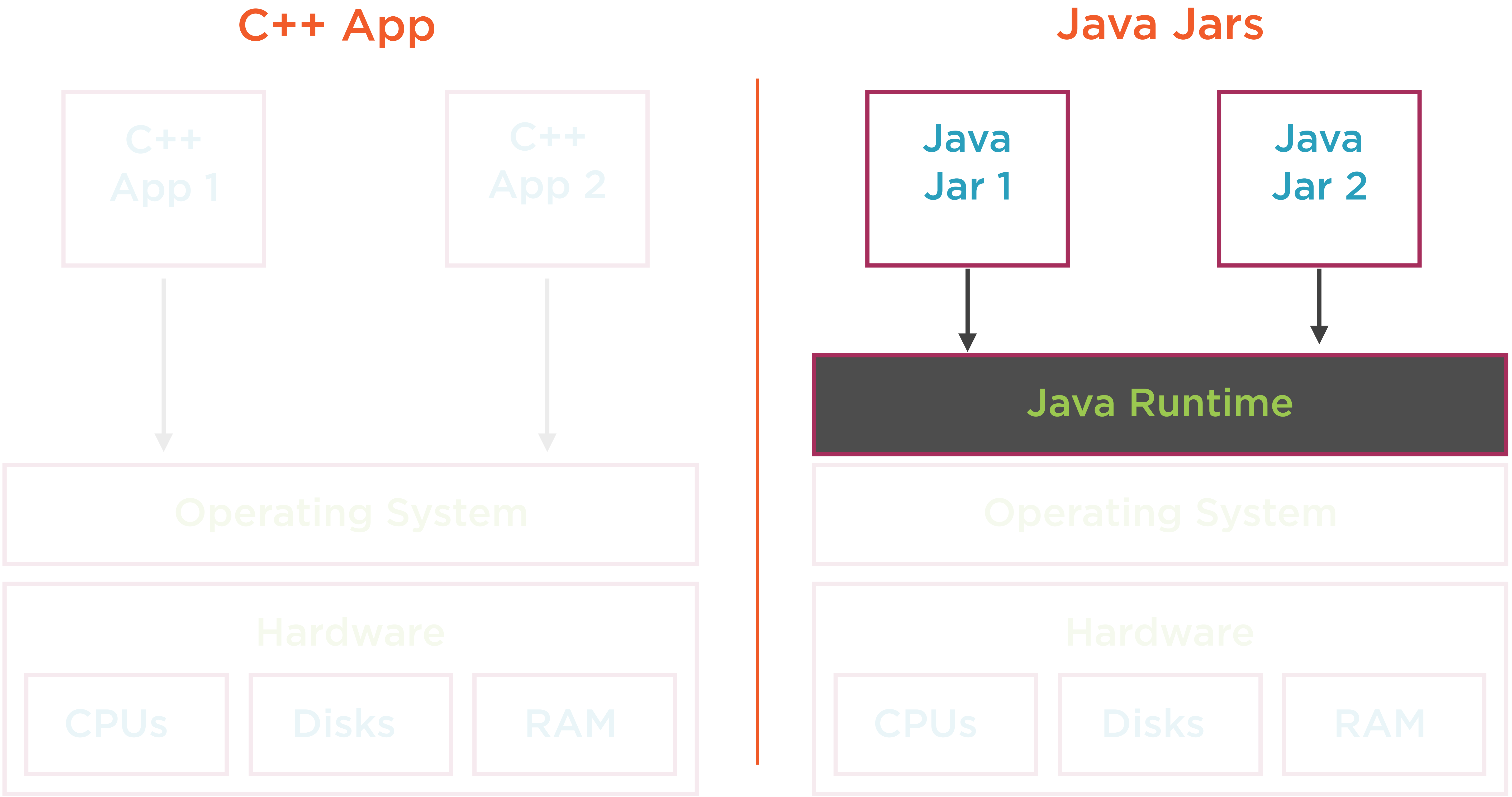
Container

A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

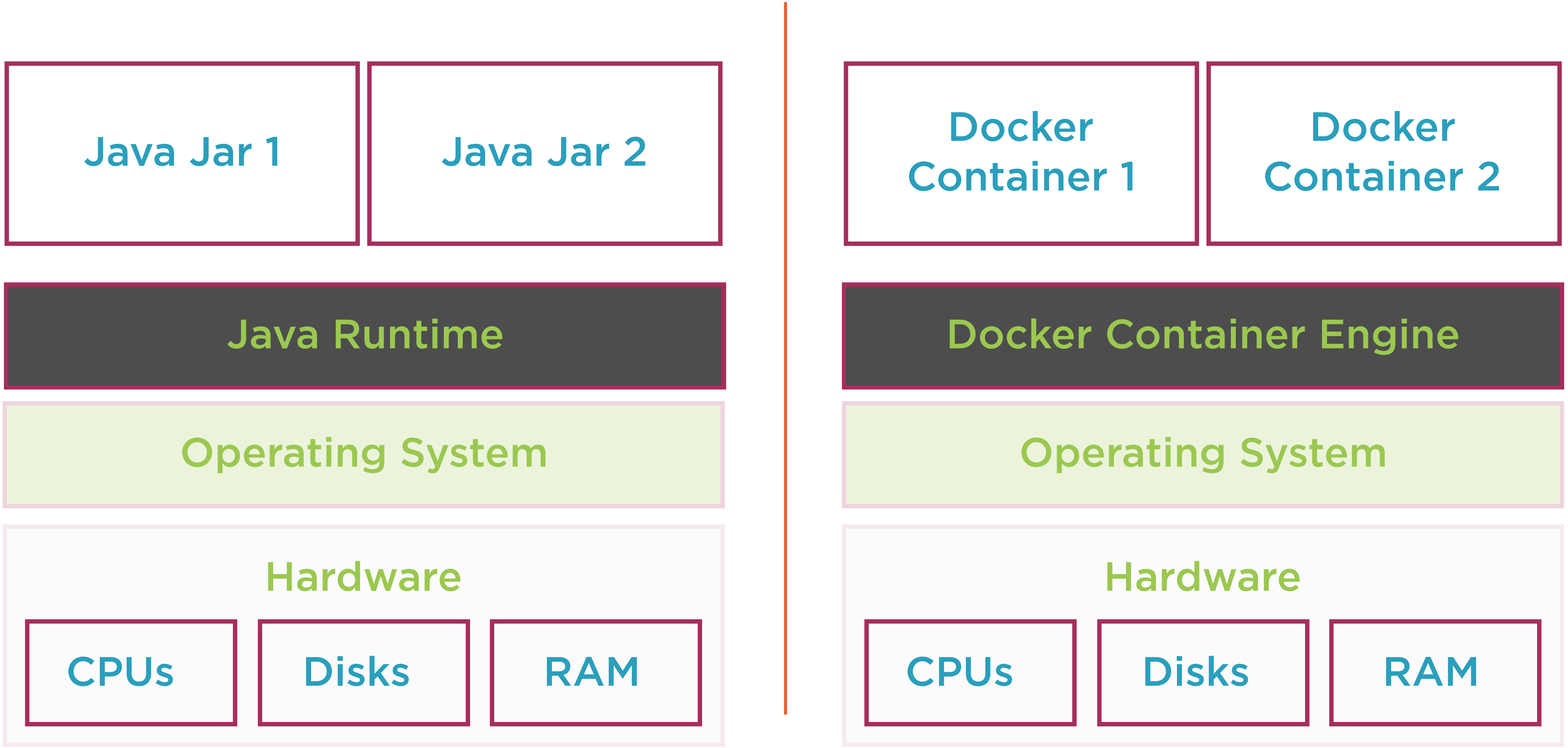
Containers Are ‘Like’ Jars



Containers Are ‘Like’ Jars



Containers Are ‘Like’ Jars



Java Jars

Java Jar 1

Java Jar 2

Java Runtime

Operating System

Hardware

CPUs

Disks

RAM

Docker Containers

Docker Container 1

Docker Container 2

Docker Container Engine

Operating System

Hardware

CPUs

Disks

RAM

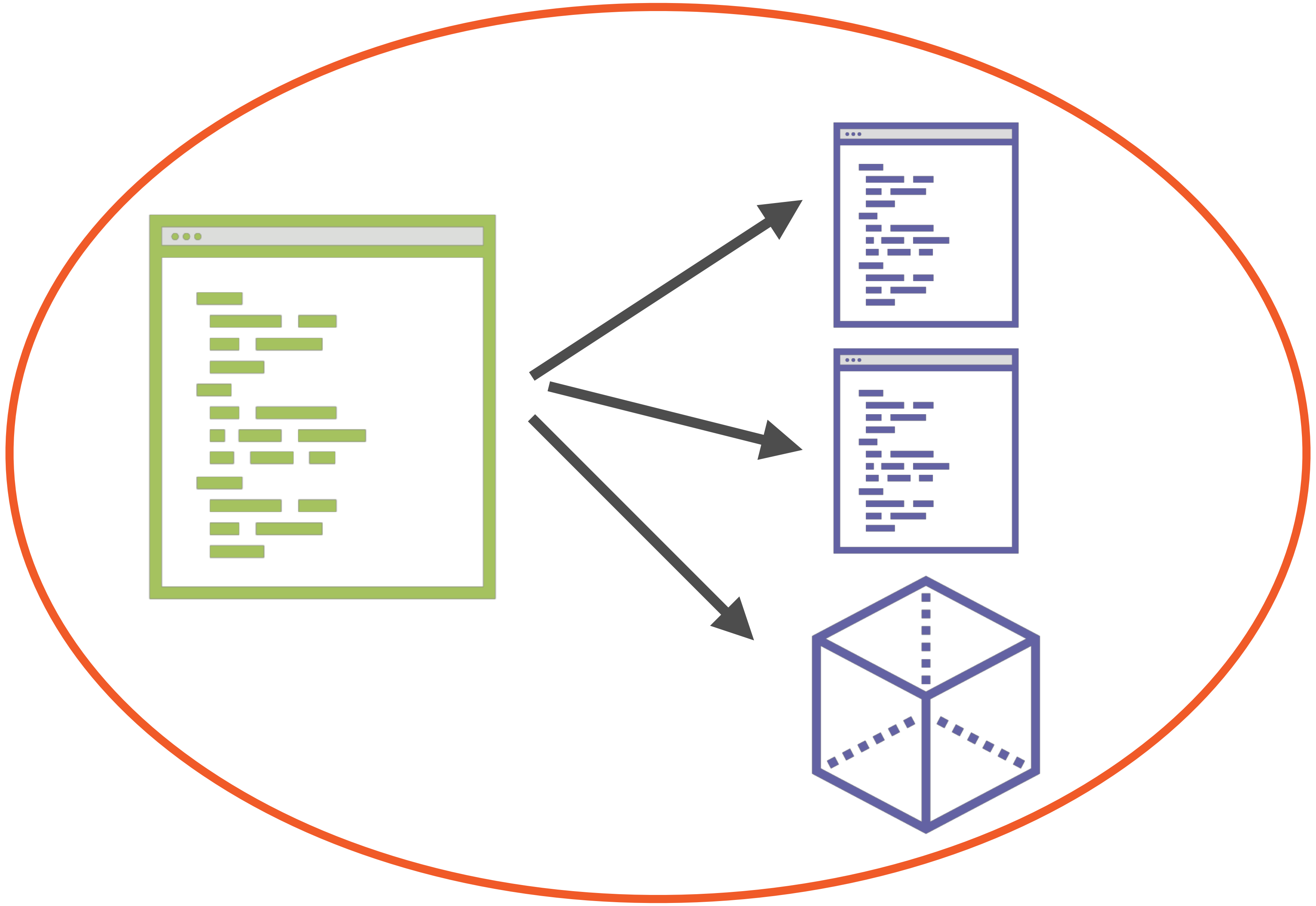
Java Jars and Containers

Java Jar Files	Containers
Files containing apps	Files containing apps
Platform independent	Platform independent
Run on layer of abstraction	Run on layer of abstraction
Java Runtime	Docker Runtime

Container

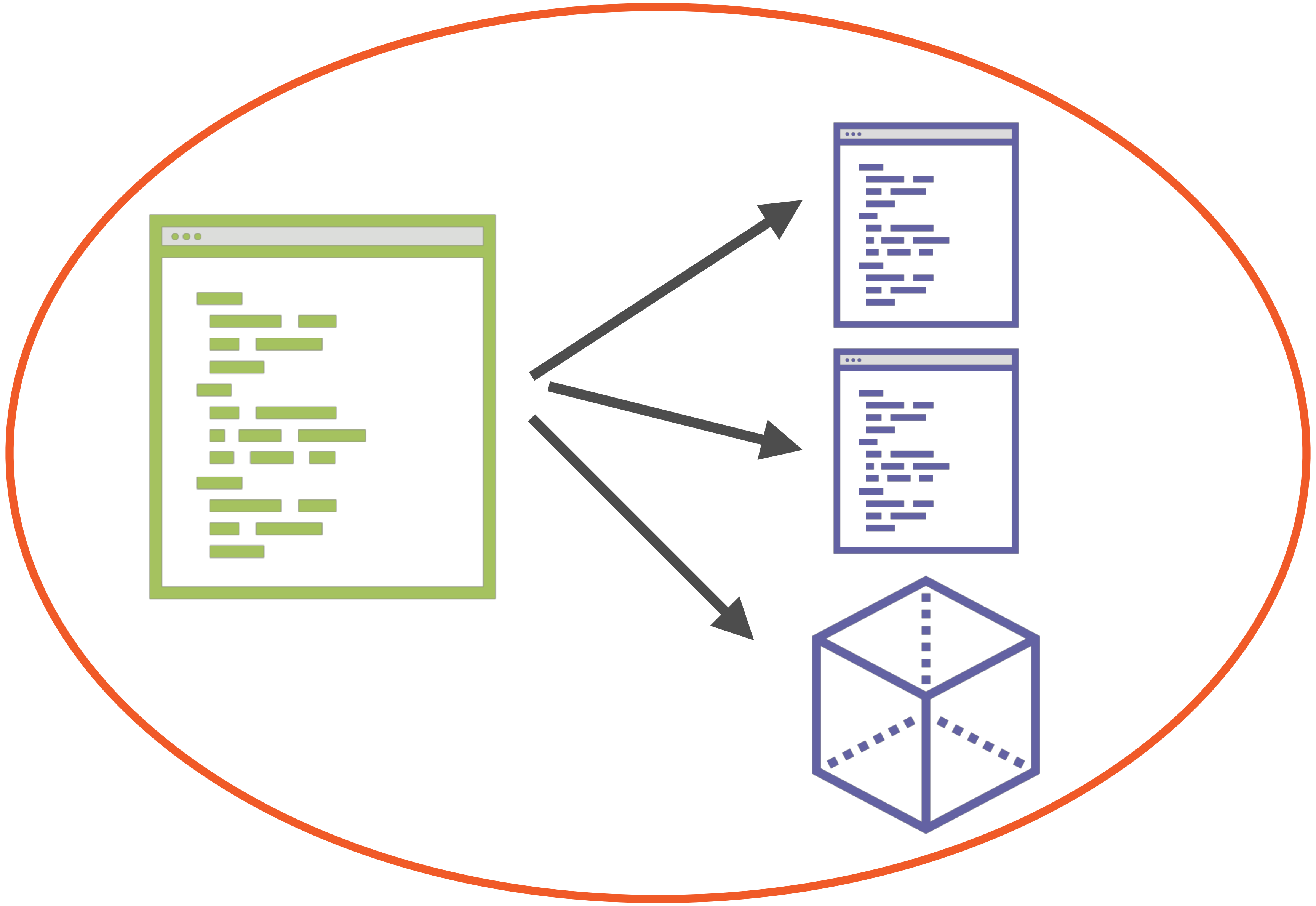
A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings

Docker Containers



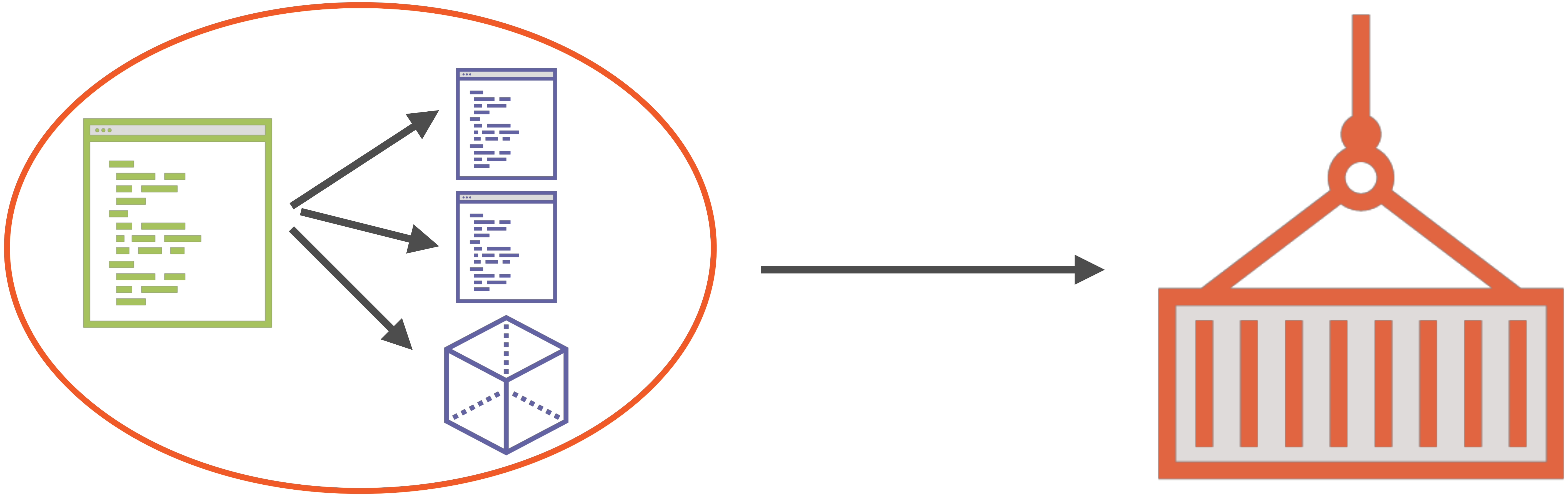
A container packages code and its dependencies into an image

Docker Containers



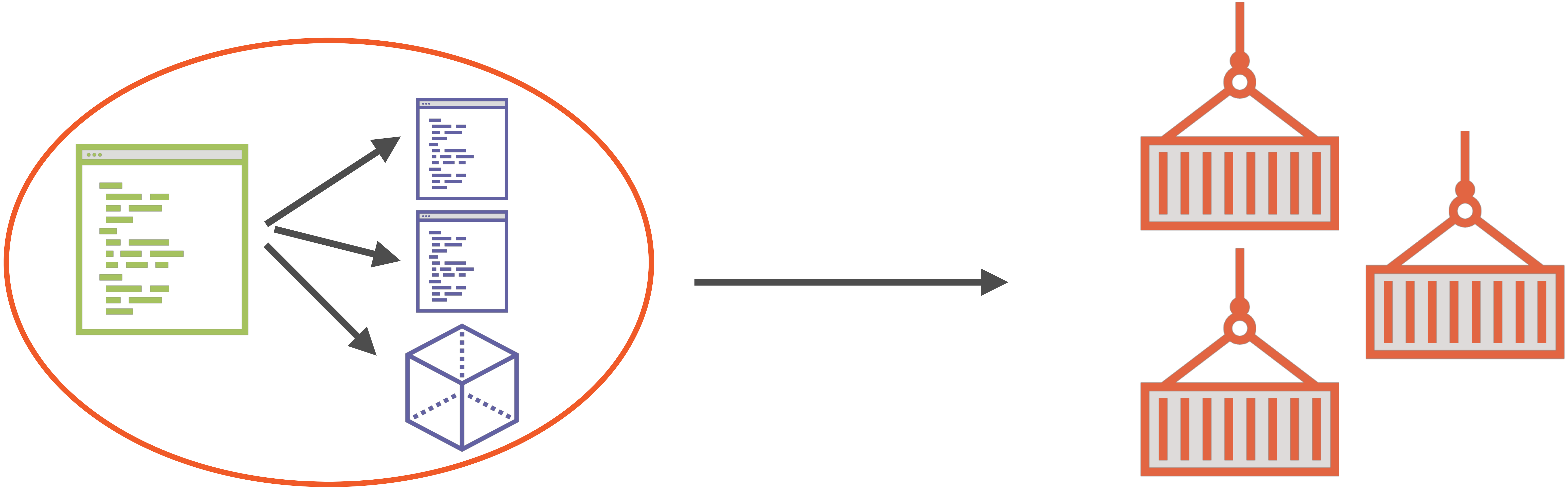
A fully self-contained environment

Docker Containers



Code is abstracted away from the machine

Docker Containers



Create as many containers as you want from the same image

Introducing Kubernetes

Attractions of Containers

No guest OS

- Platform independent
- Considerably smaller than VM images

Lightweight

- Small and fast
- Quick to start
- Speeds up autoscaling

Hybrid, multi-cloud

- Hybrid: Work on-premise and on cloud
- Multi-cloud: Not tied to any specific cloud platform

Standalone Container Limitations

No autohealing

- Crashed containers won’t restart automatically
- Need higher level orchestration

No scaling or autoscaling

- Overloaded containers don’t spawn more automatically
- Need higher level orchestration

No load balancing

- Containers can’t share load automatically
- Need higher level orchestration

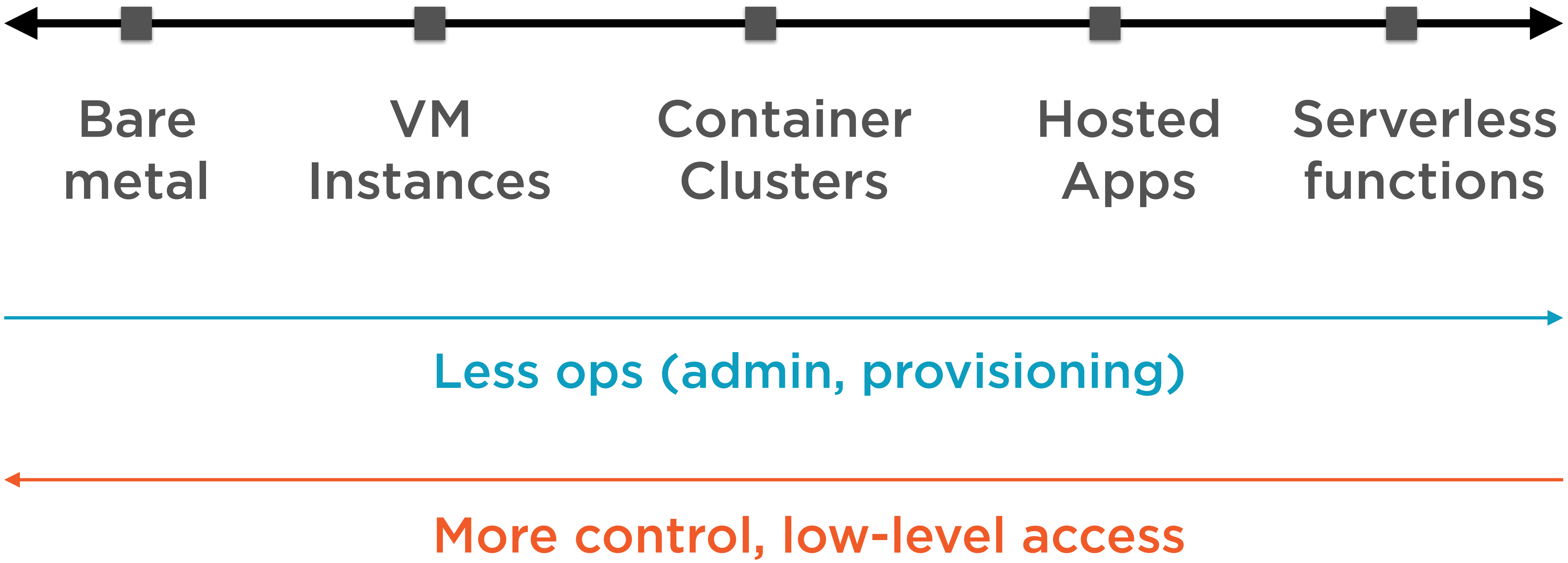
No isolation

- Crashing containers can take each other down
- Need sandbox to separate them

Kubernetes

Orchestration technology for containers - convert isolated containers running on different hardware into a cluster

Compute Choices



Bare metal

VM Instances

Container Clusters

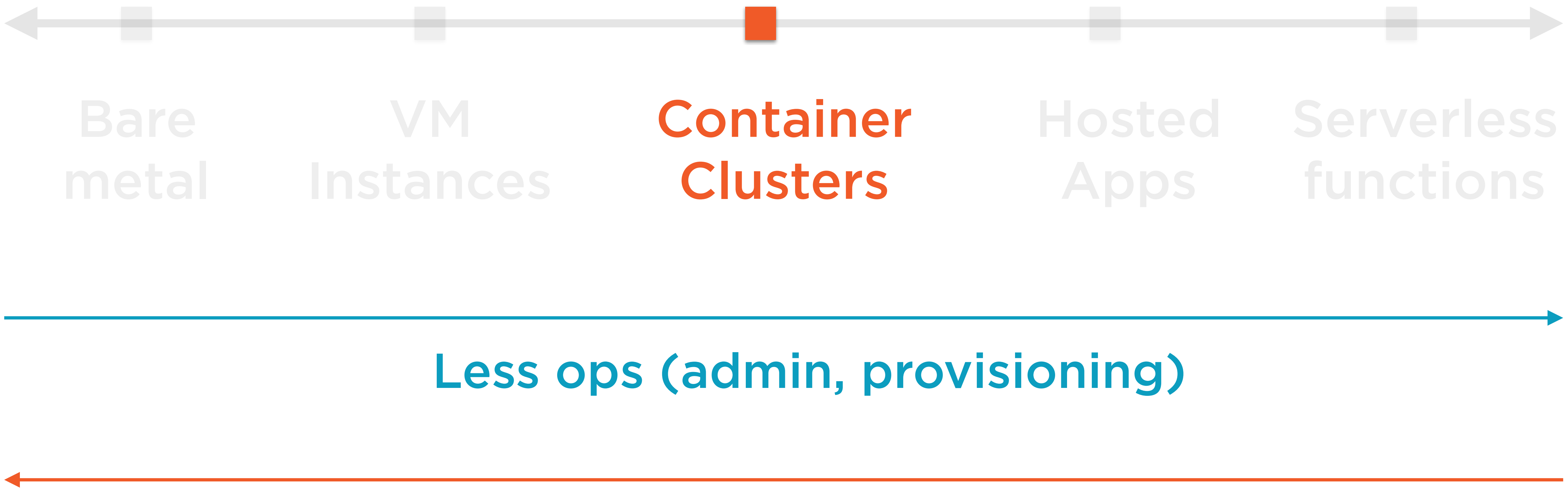
Hosted Apps

Serverless functions

Less ops (admin, provisioning)

More control, low-level access

Compute Choices



Bare metal

VM Instances

Container Clusters

Hosted Apps

Serverless functions

Less ops (admin, provisioning)

More control, low-level access

Kubernetes is fast emerging as middle-ground between IaaS and PaaS in a hybrid, multi-cloud world

Hybrid, Multi-cloud

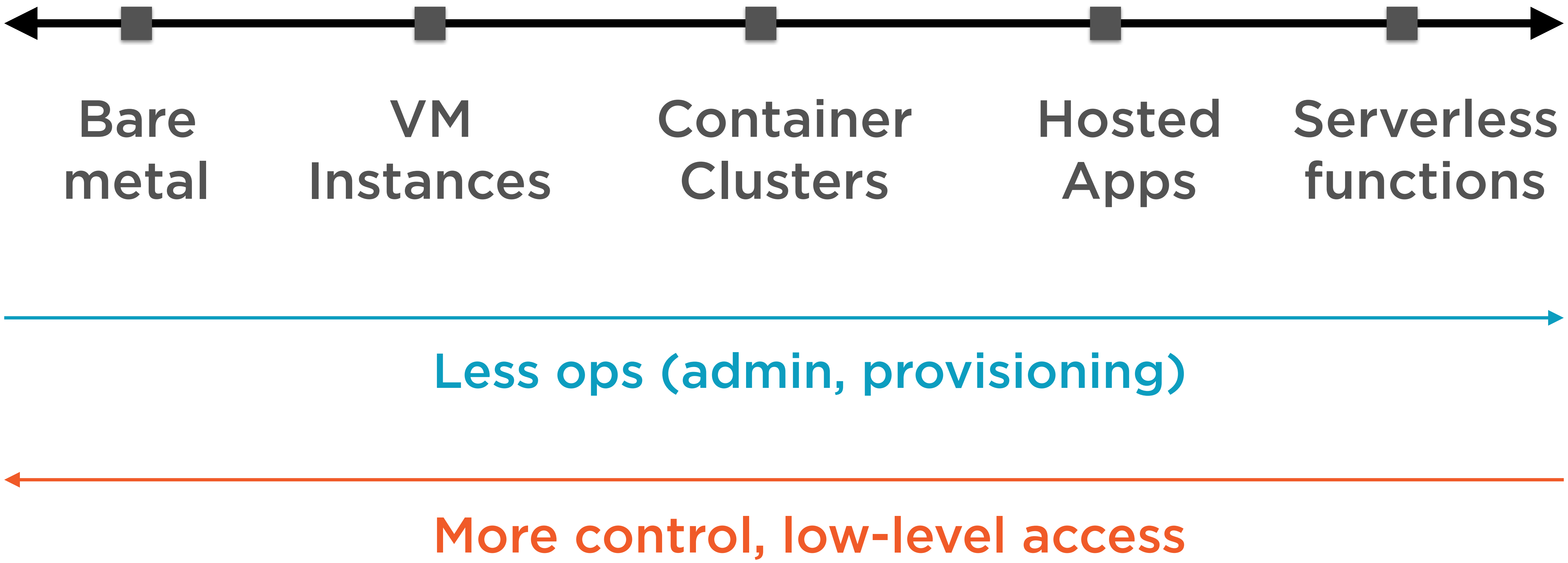
Hybrid: Runs on-premise and on cloud

- Provides smooth migration path

Multi-cloud: Supported by all big cloud platforms

- Important for strategic independence
- Amazon-Whole Foods merger

Compute Choices



Bare metal

VM Instances

Container Clusters

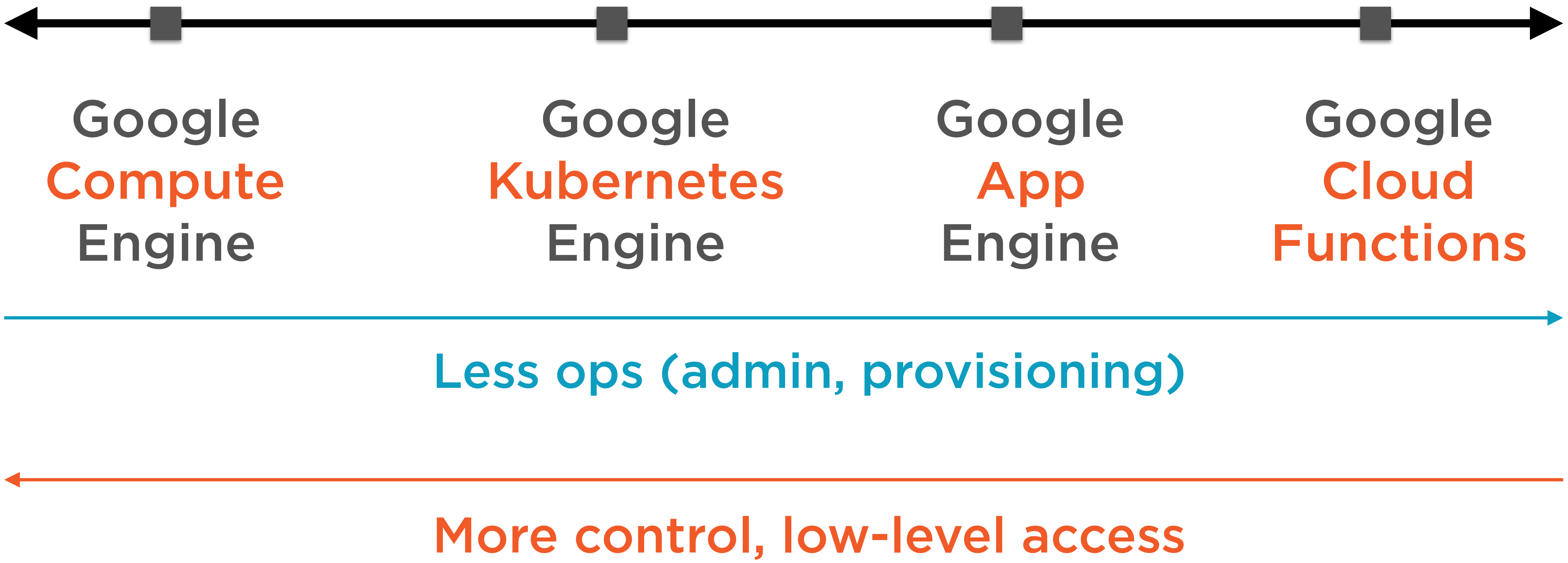
Hosted Apps

Serverless functions

Less ops (admin, provisioning)

More control, low-level access

GCP Compute Choices



Google **Compute** Engine

Google **Kubernetes** Engine

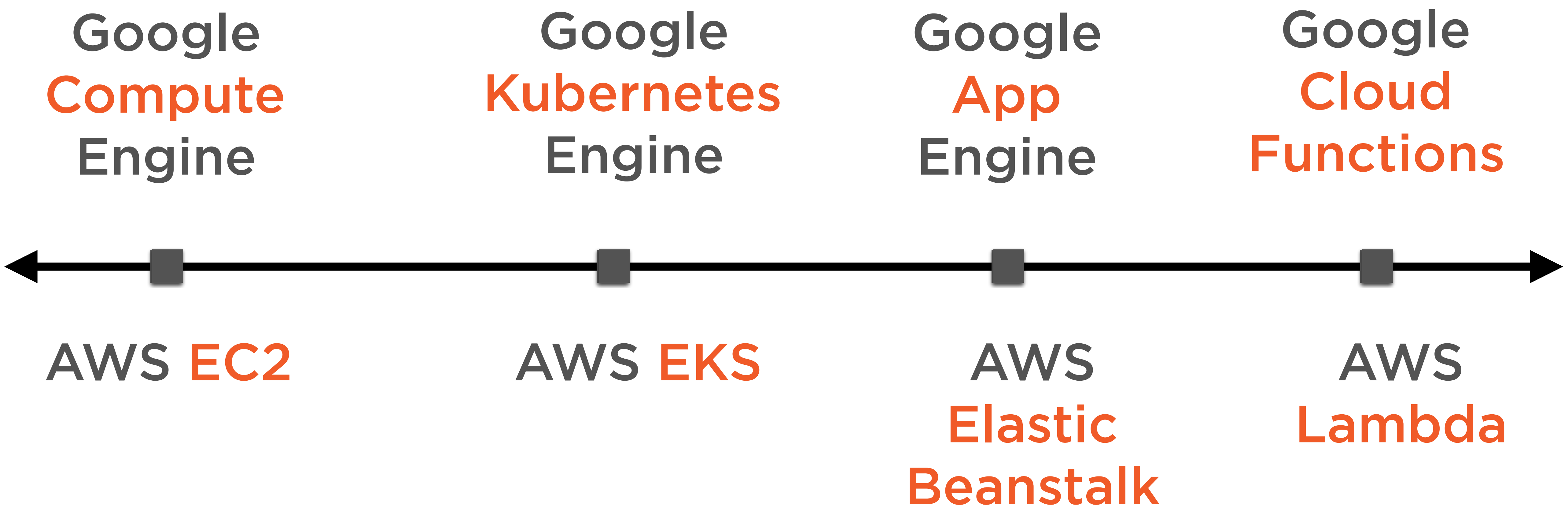
Google **App** Engine

Google **Cloud Functions**

Less ops (admin, provisioning)

More control, low-level access

GCP Compute Choices



Google Compute Engine

Google Kubernetes Engine

Google App Engine

Google Cloud Functions

AWS EC2

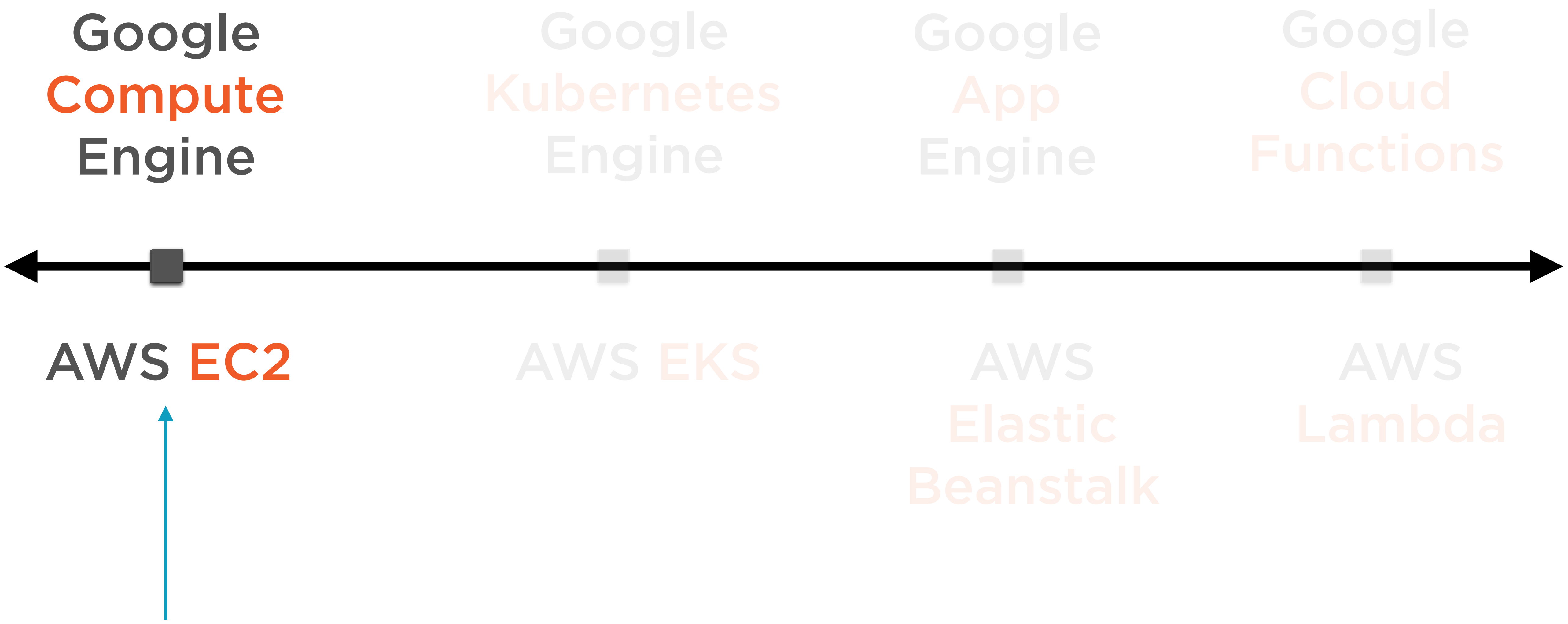
AWS EKS

AWS Elastic Beanstalk

AWS Lambda

Every major cloud platform supports the same range of compute choices

GCP Compute Choices



Google Compute Engine

Google Kubernetes Engine

Google App Engine

Google Cloud Functions

AWS EC2

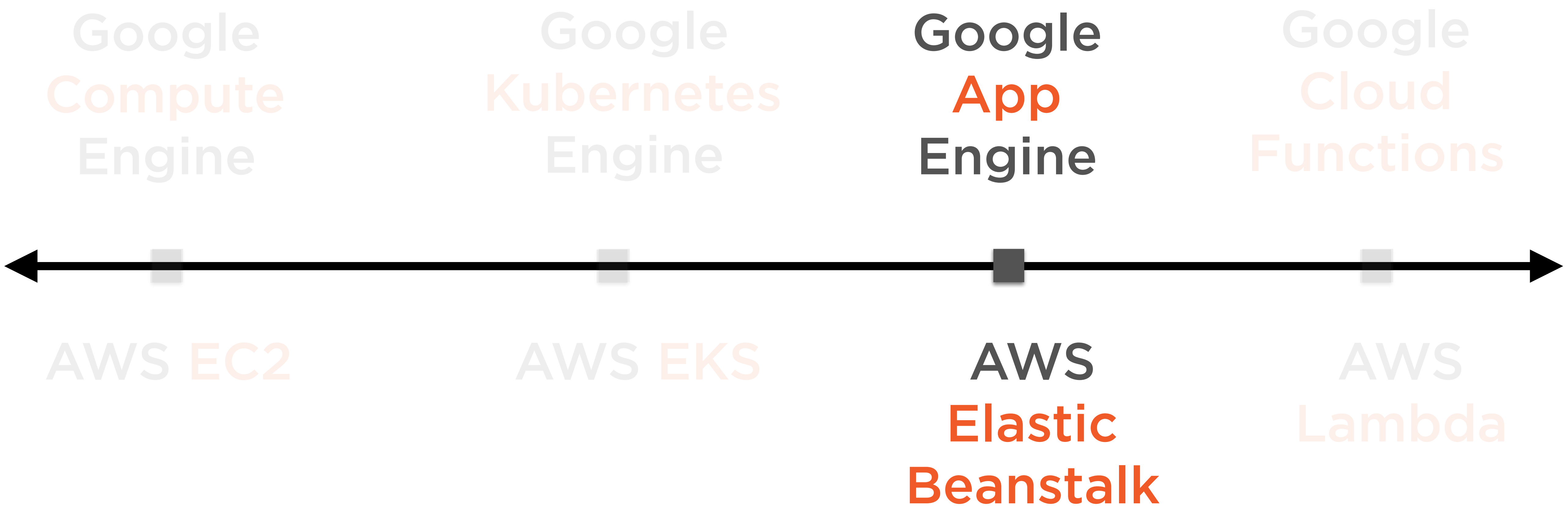
AWS EKS

AWS Elastic Beanstalk

AWS Lambda

“**I**nfrastructure-**a**s-**a**-**S**ervice” (IaaS)

GCP Compute Choices



Google Compute Engine

Google Kubernetes Engine

Google App Engine

Google Cloud Functions

AWS EC2

AWS EKS

AWS Elastic Beanstalk

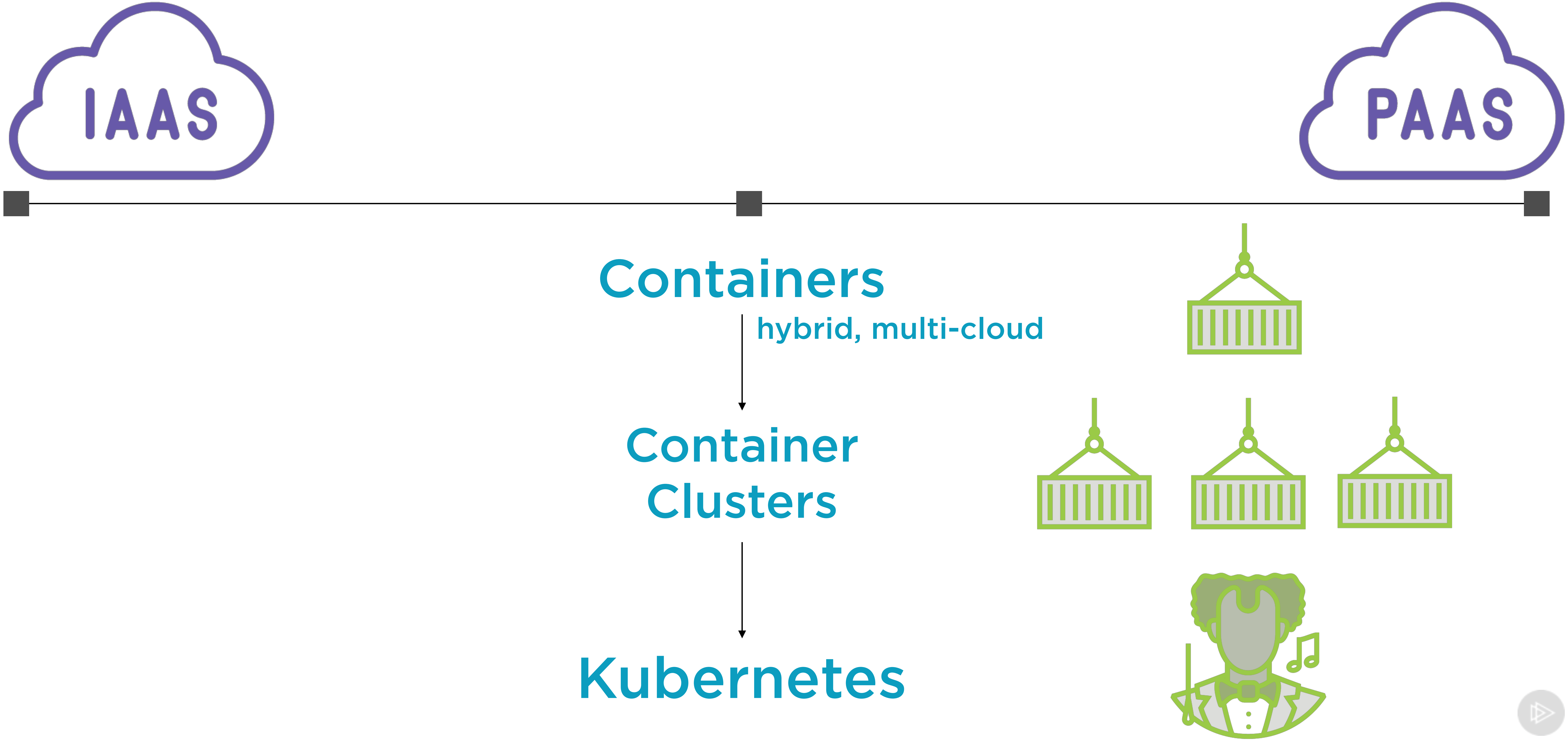
AWS Lambda

“**P**latform-**a**s-**a**-**S**ervice” (PaaS)

IaaS vs. PaaS

Infrastructure-as-a-Service	Platform-as-a-Service
Heavy operational burden	Provider lock-in
Migration is hard	Migration is very hard

Compute Choices



IAAS

PAAS

Containers

hybrid, multi-cloud

Container Clusters

Kubernetes

Kubernetes as Orchestrator

Fault-tolerance

Autohealing

Isolation

Scaling

Autoscaling

Load balancing

All of these are possible in a Kubernetes cluster using higher level abstractions

Hybrid, Multi-cloud

Kubernetes is supported by each of the Big Three

Special relationship with GCP

Kubernetes originated at Google

Google Kubernetes Engine (GKE)

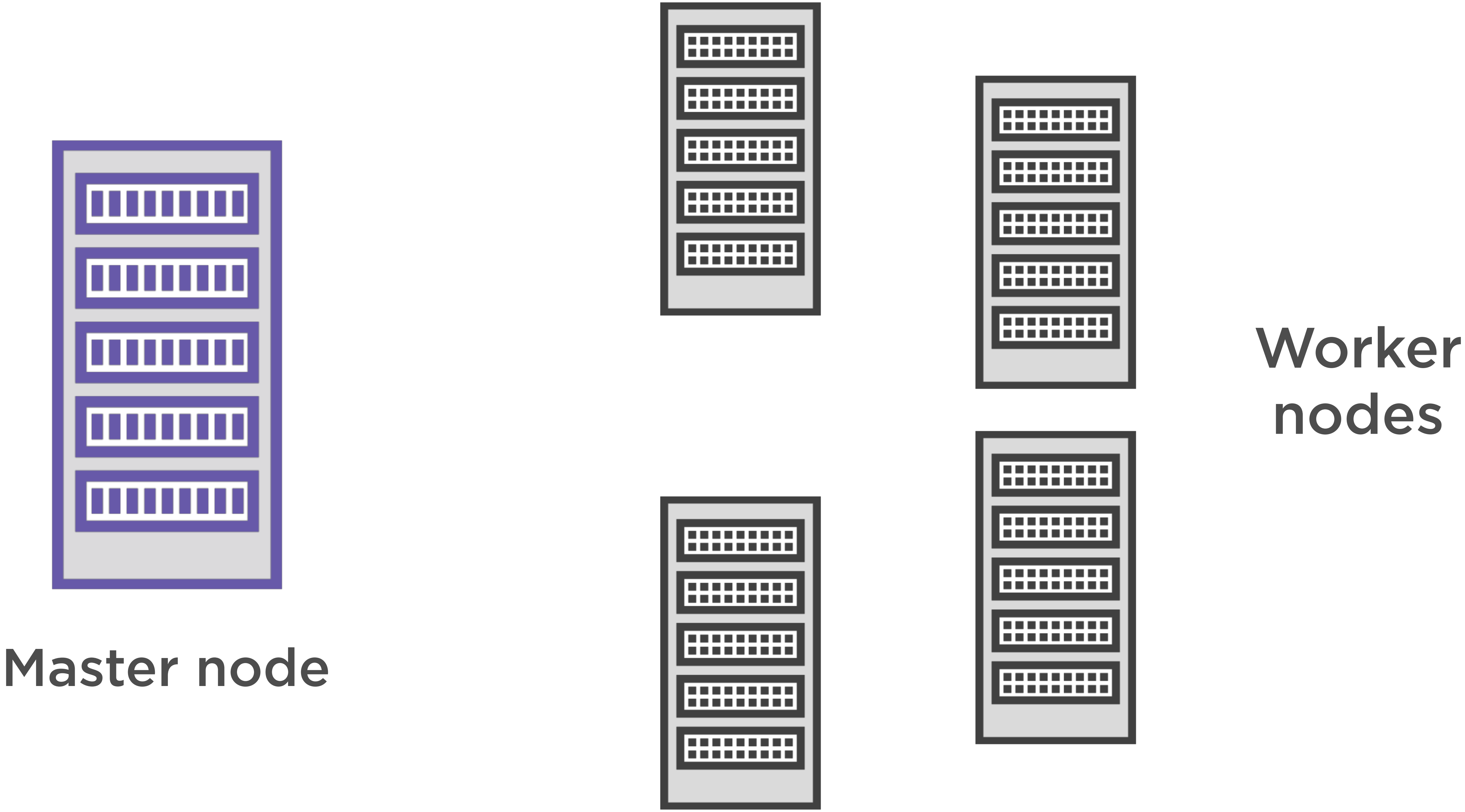
Google Kubernetes Engine

Service for working with Kubernetes clusters on GCP

Runs Kubernetes on GCE VM instances

Clusters, Nodes, Node Pools, Node Images

Kubernetes Clusters



Master node

Worker nodes

Master

One or more nodes designated master nodes

Managed by GKE

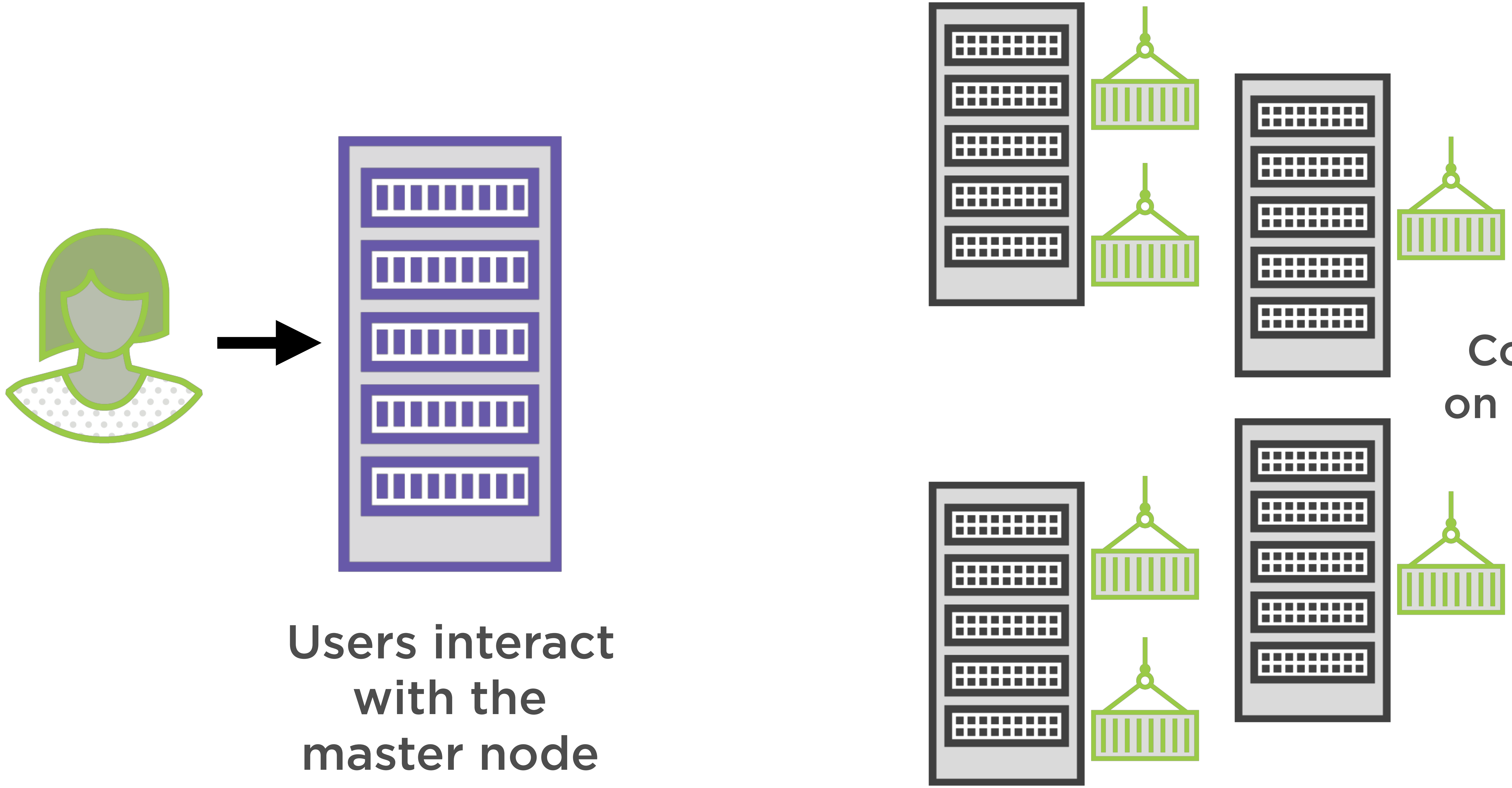
Not visible directly to user

Multi-master for high-availability

Kubernetes Control Plane directed from here

All user interactions with Kubernetes clusters are via the **kube-apiserver** running on the master node

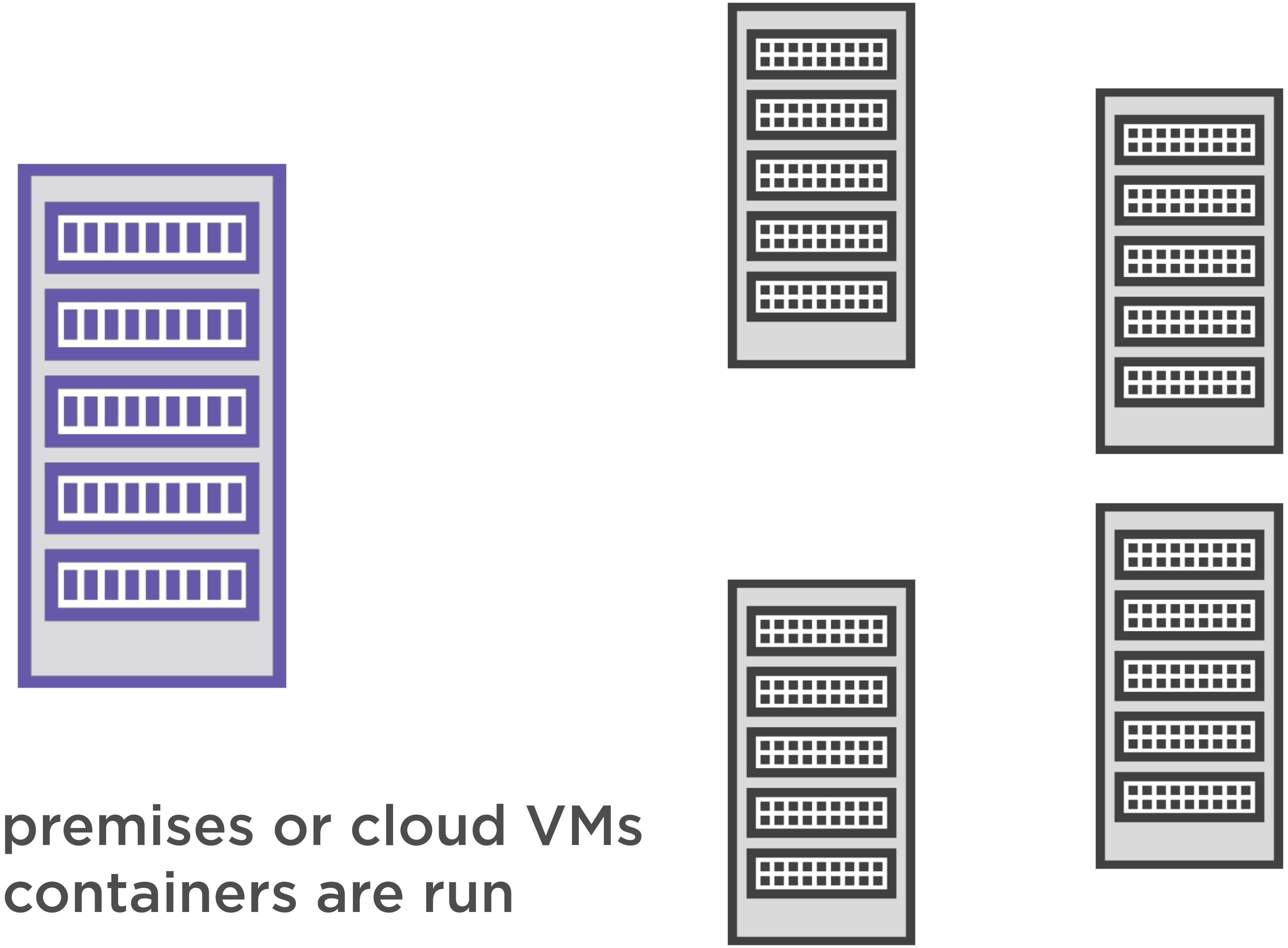
Kubernetes Clusters



Users interact with the master node

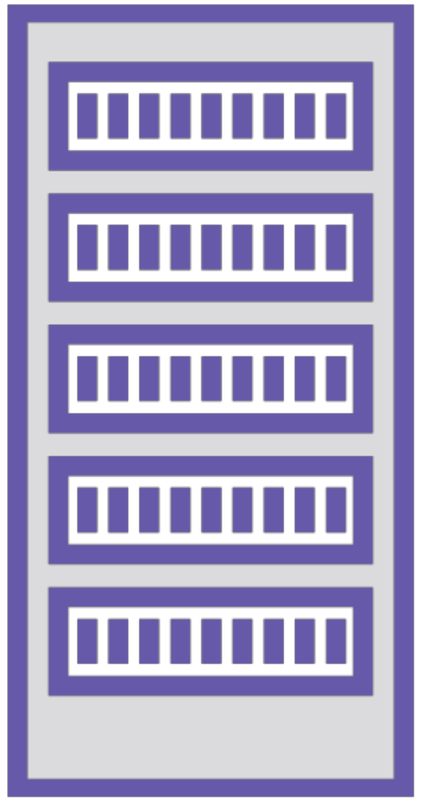
Containers run on worker nodes

Nodes

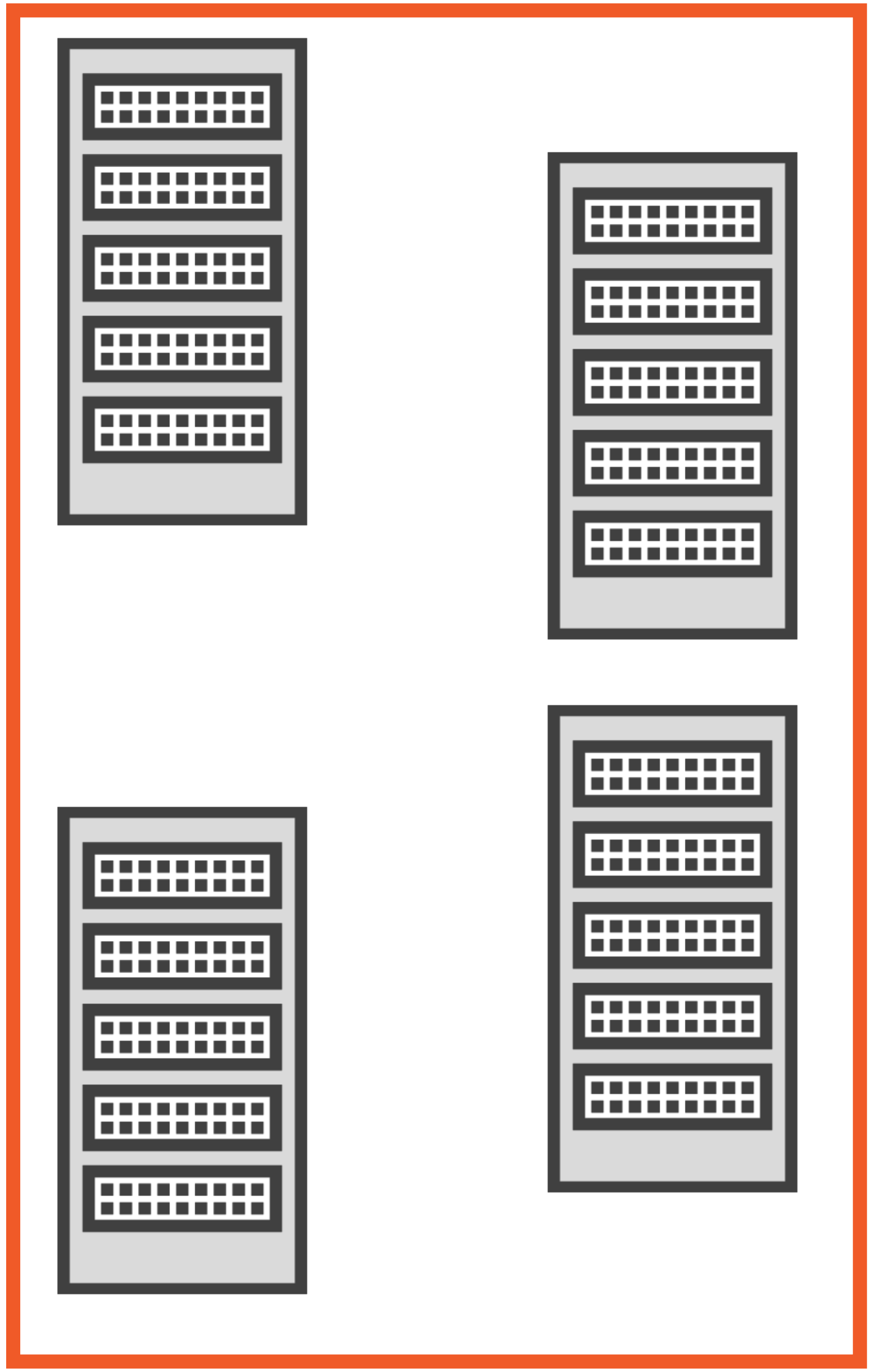


Nodes are on-premises or cloud VMs on which containers are run

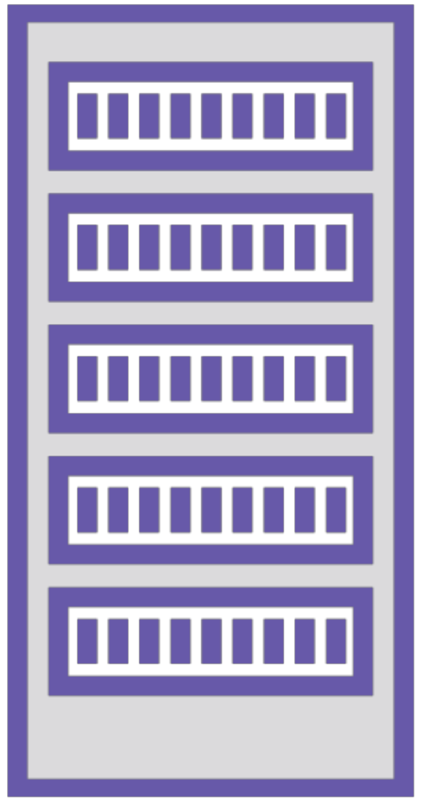
Node Pools

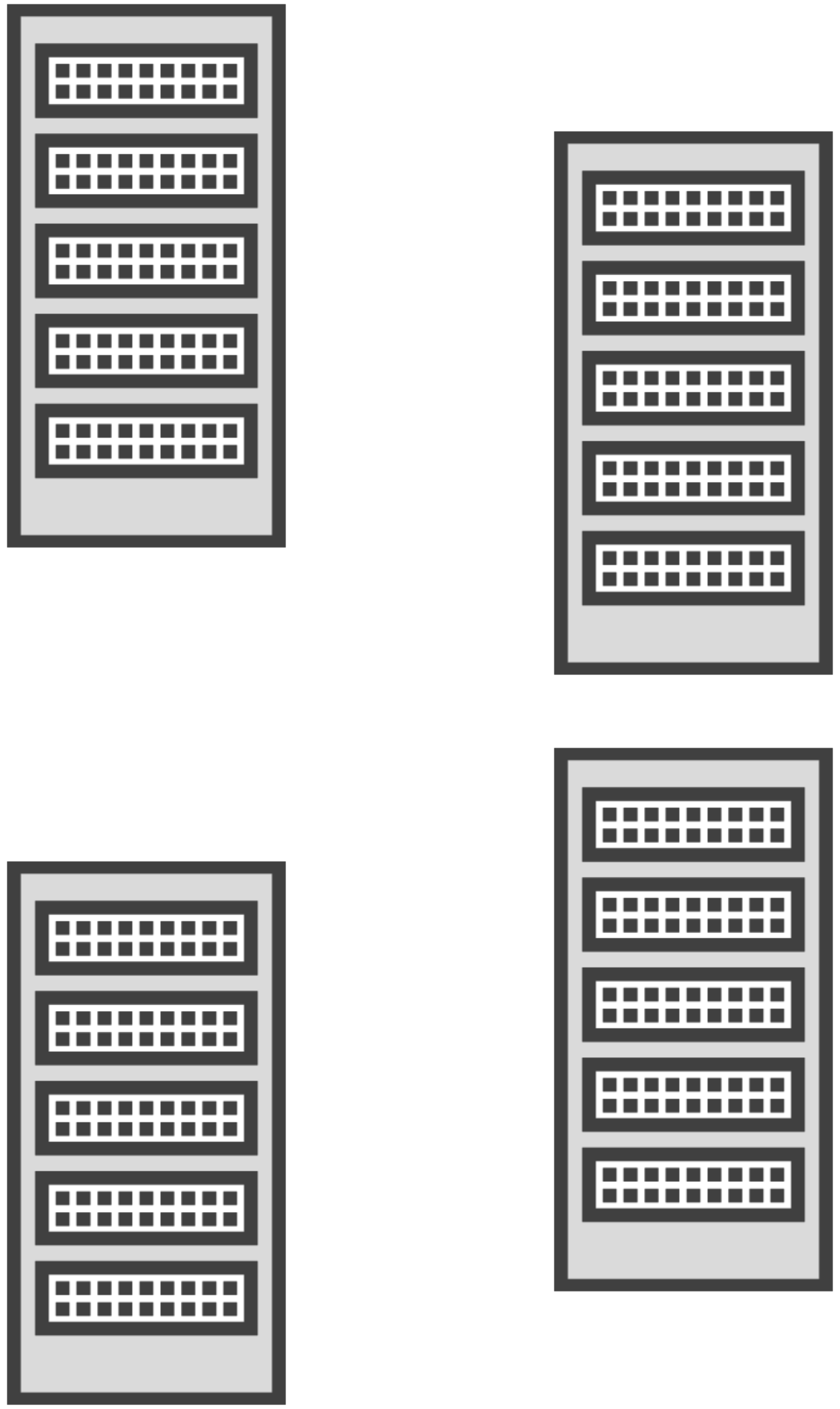


A subset of node instances which have the same configuration are called node pools



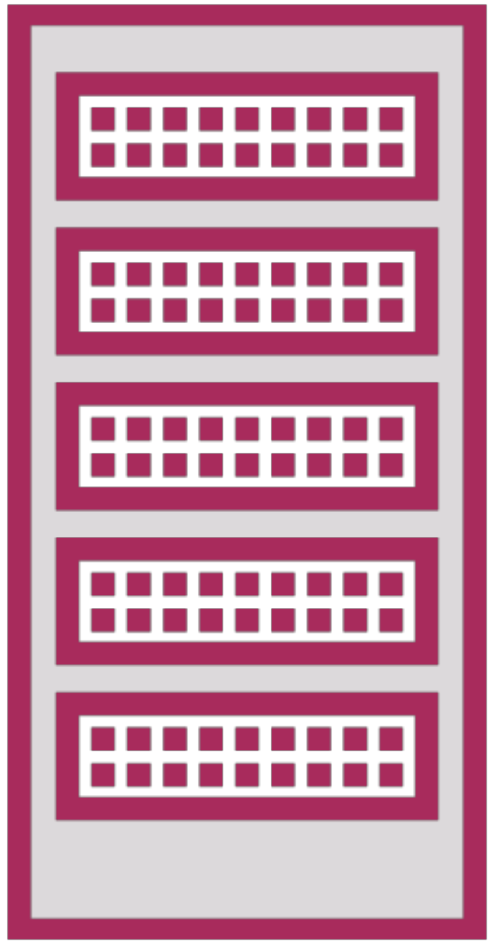
Node Images





Special operating system images are available on the Google Cloud to run on Kubernetes nodes

Node Images



Container-optimized OS

- Enhances node security
- Supported by teams at Google

Container-optimized OS with `containerd`

- `containerd` as the main container runtime

Ubuntu

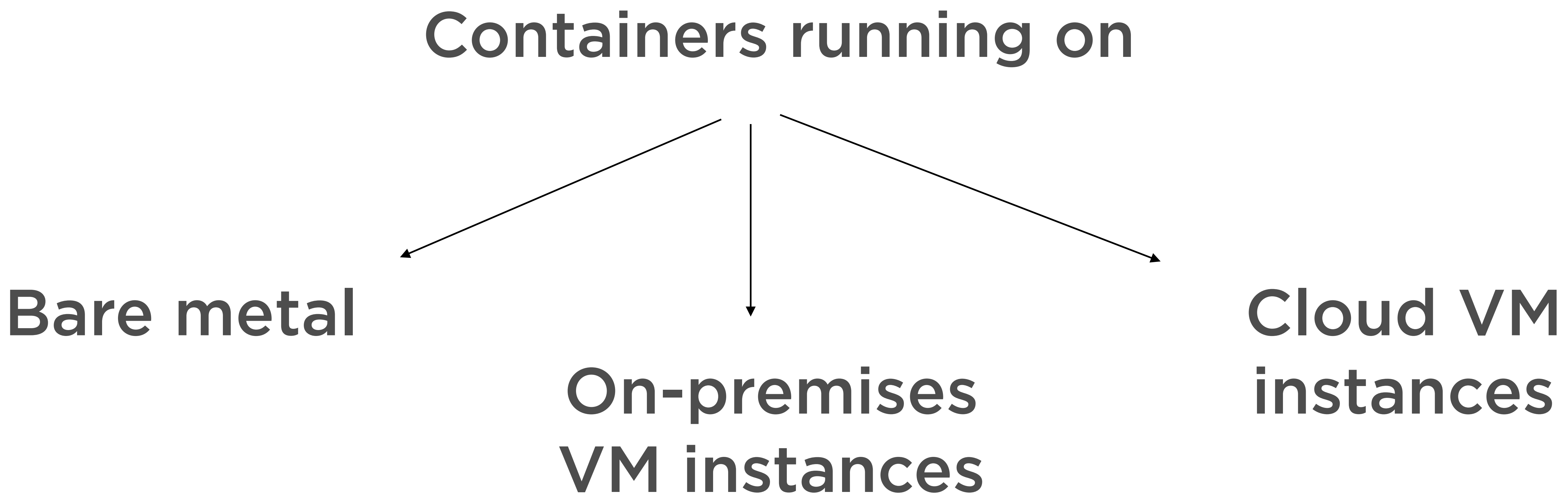
- Optimized for GKE
- Additional support for XFS, CephFS, Sysdig or Debian packages

Pods

Kubernetes

Orchestration technology for containers - convert isolated containers running on different hardware into a cluster

Isolated Container Deployments



Containers running on

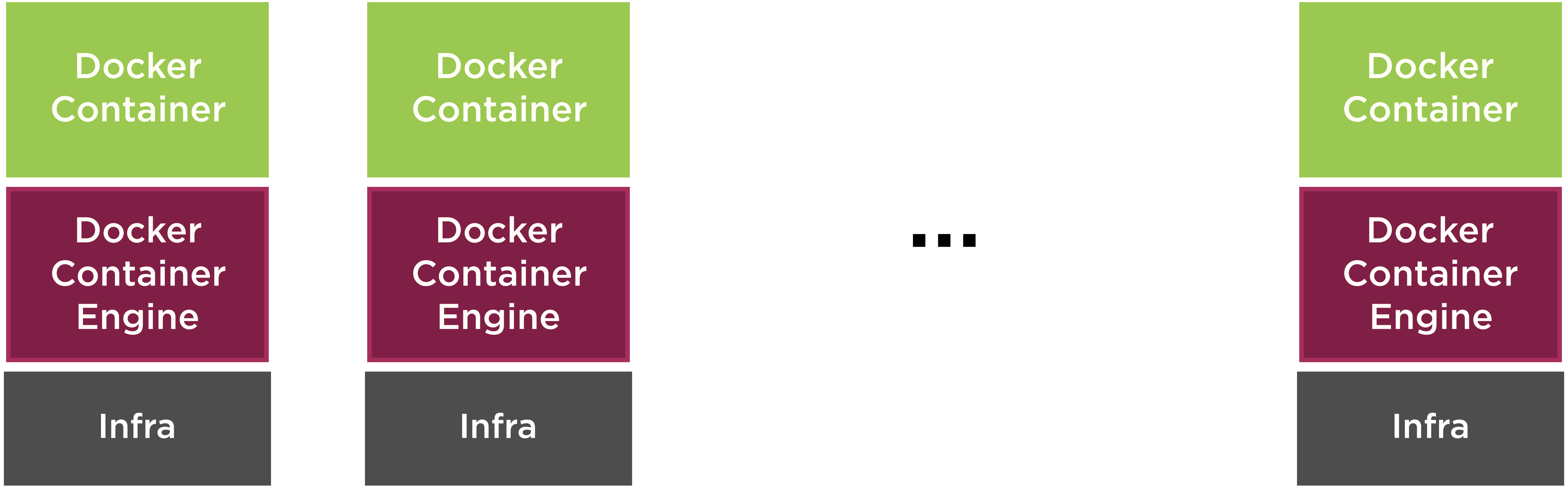
Bare metal

On-premises VM instances

Cloud VM instances

Potentially thousands of containers on hundreds of VMs

Isolated Container Deployments



Docker Container

Docker Container Engine

Infra

Docker Container

Docker Container Engine

Infra

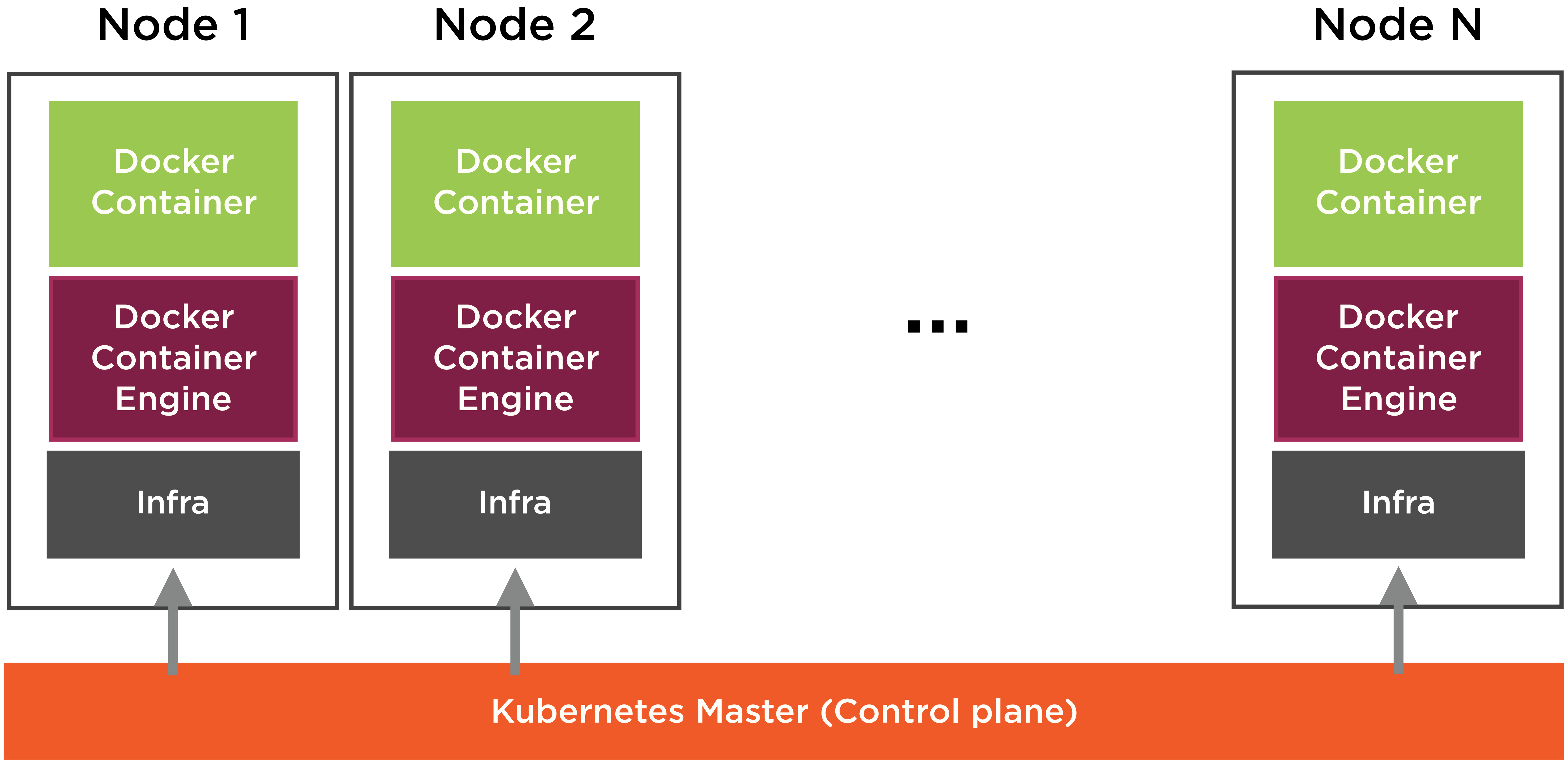
...

Docker Container

Docker Container Engine

Infra

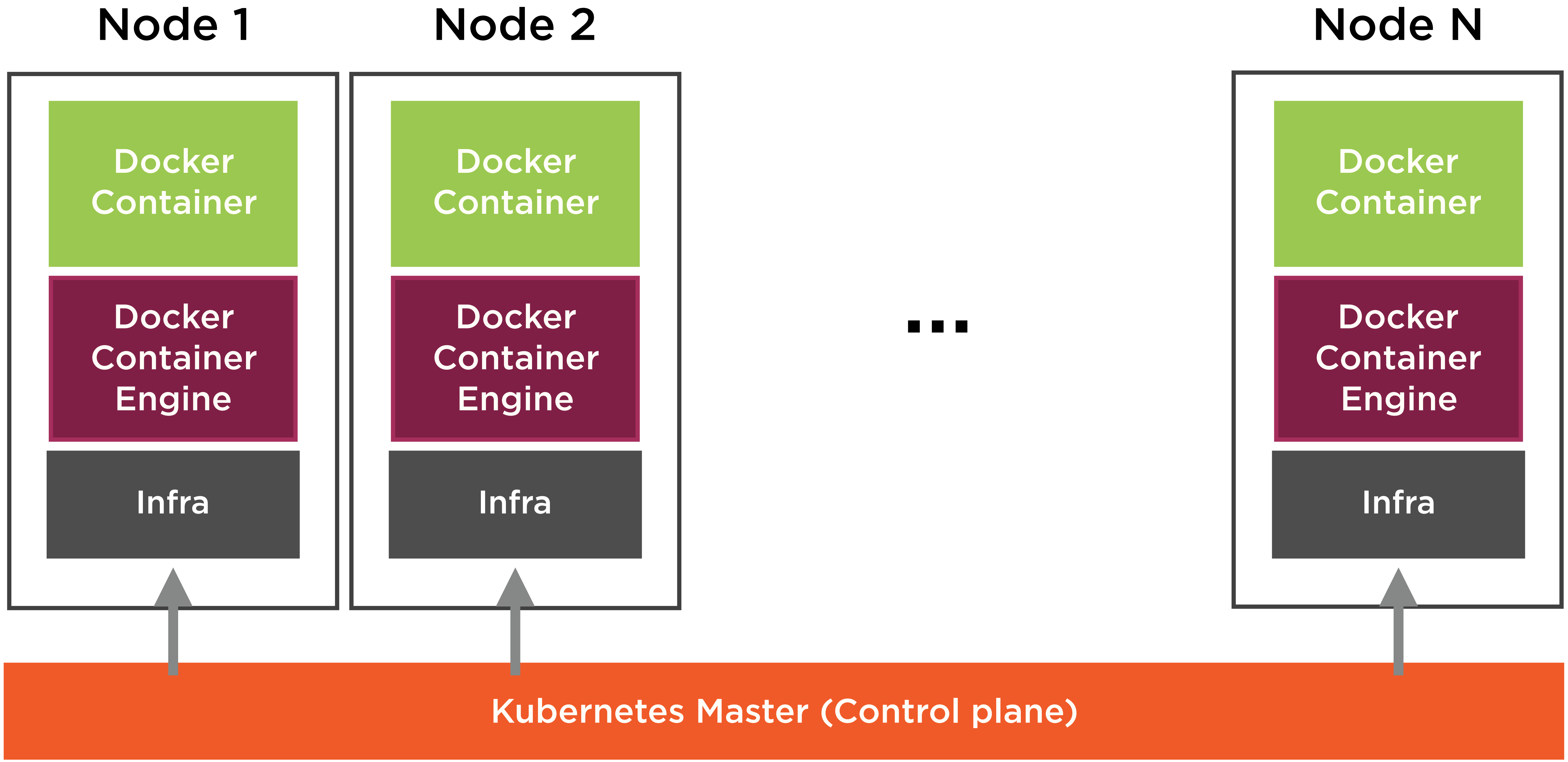
Kubernetes: Cluster Orchestration



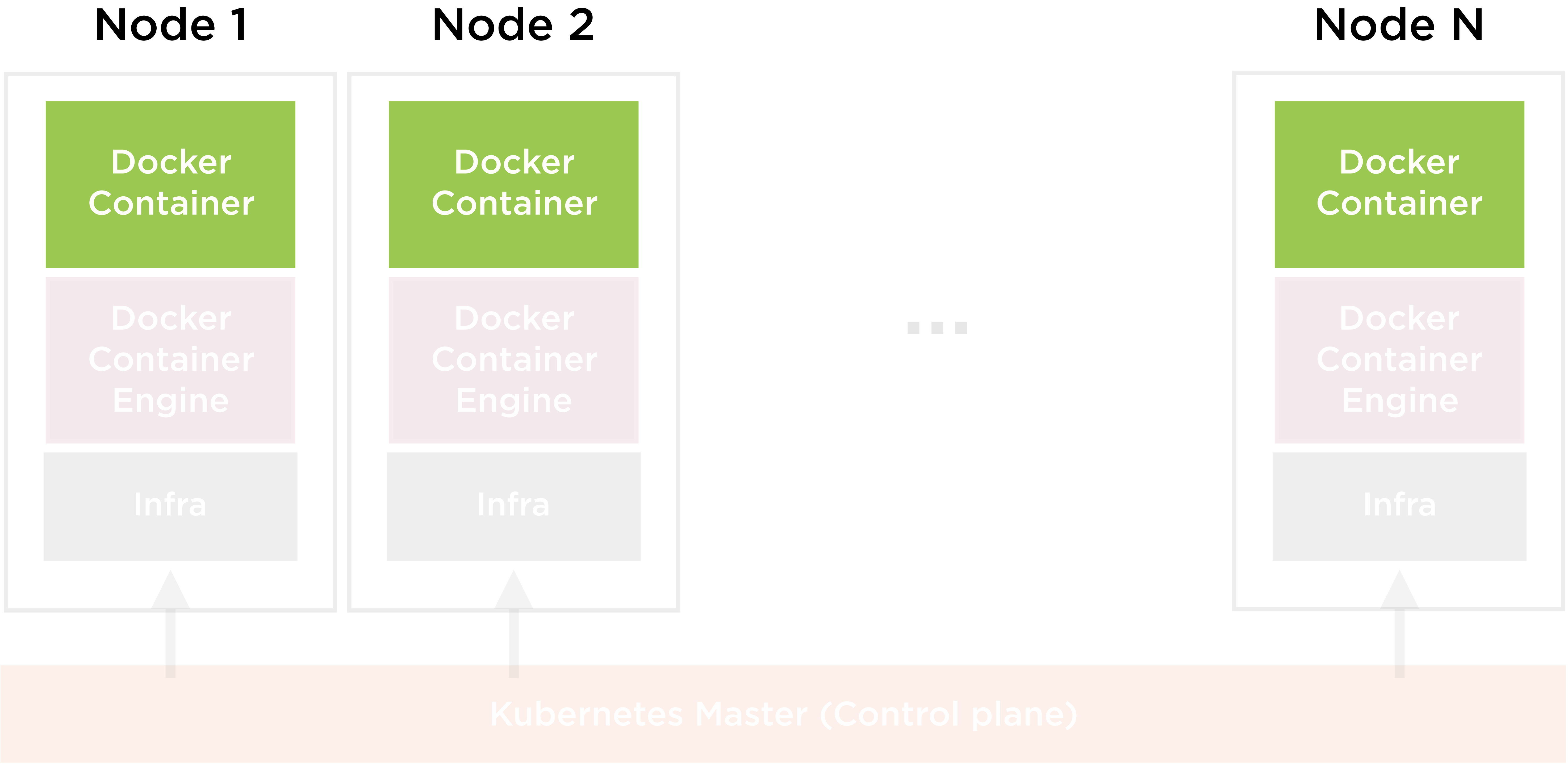
Kubernetes does not interact directly with containers

Instead it uses a number of higher-level entities referred to as **objects**

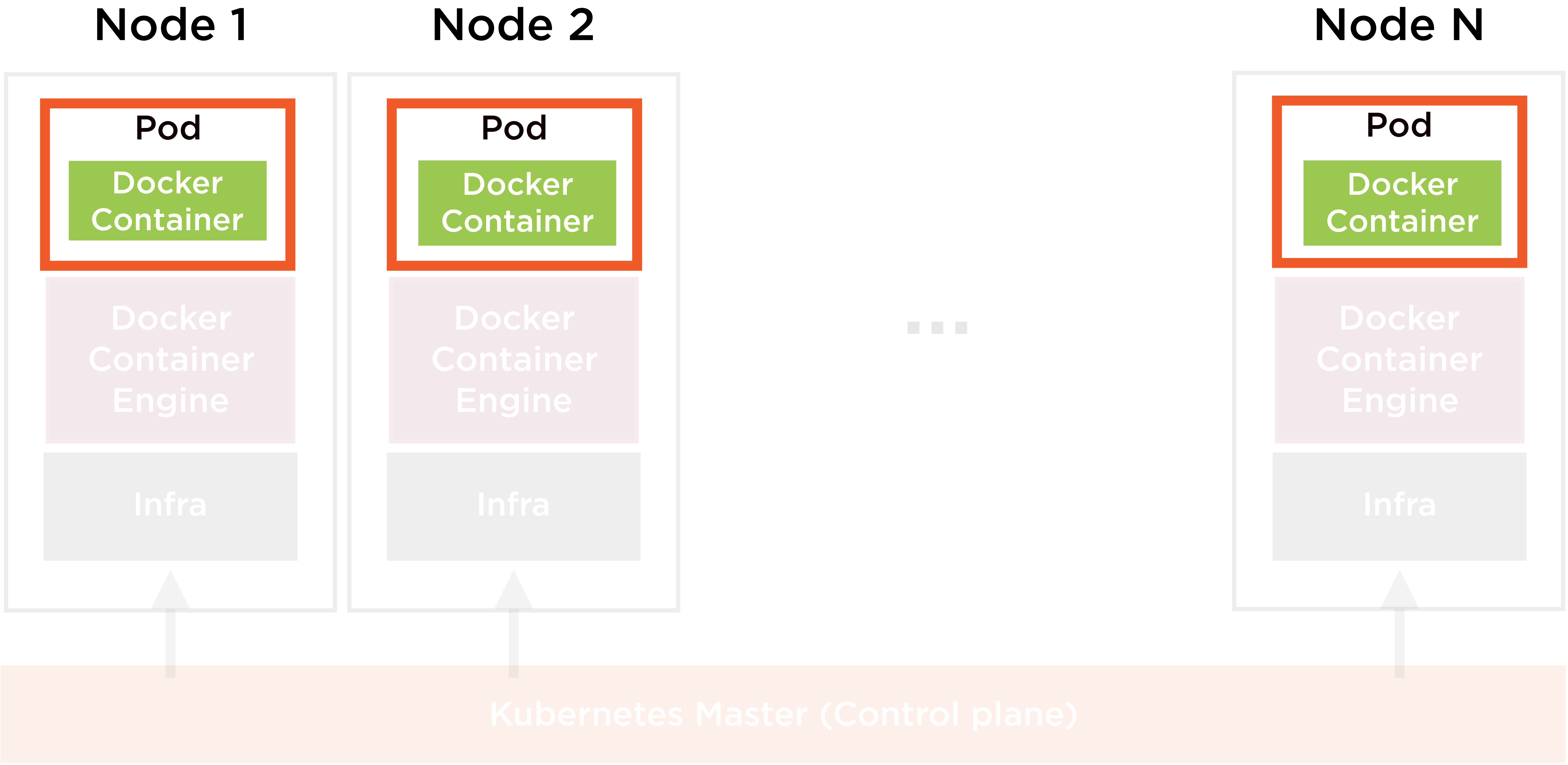
Kubernetes: Cluster Orchestration



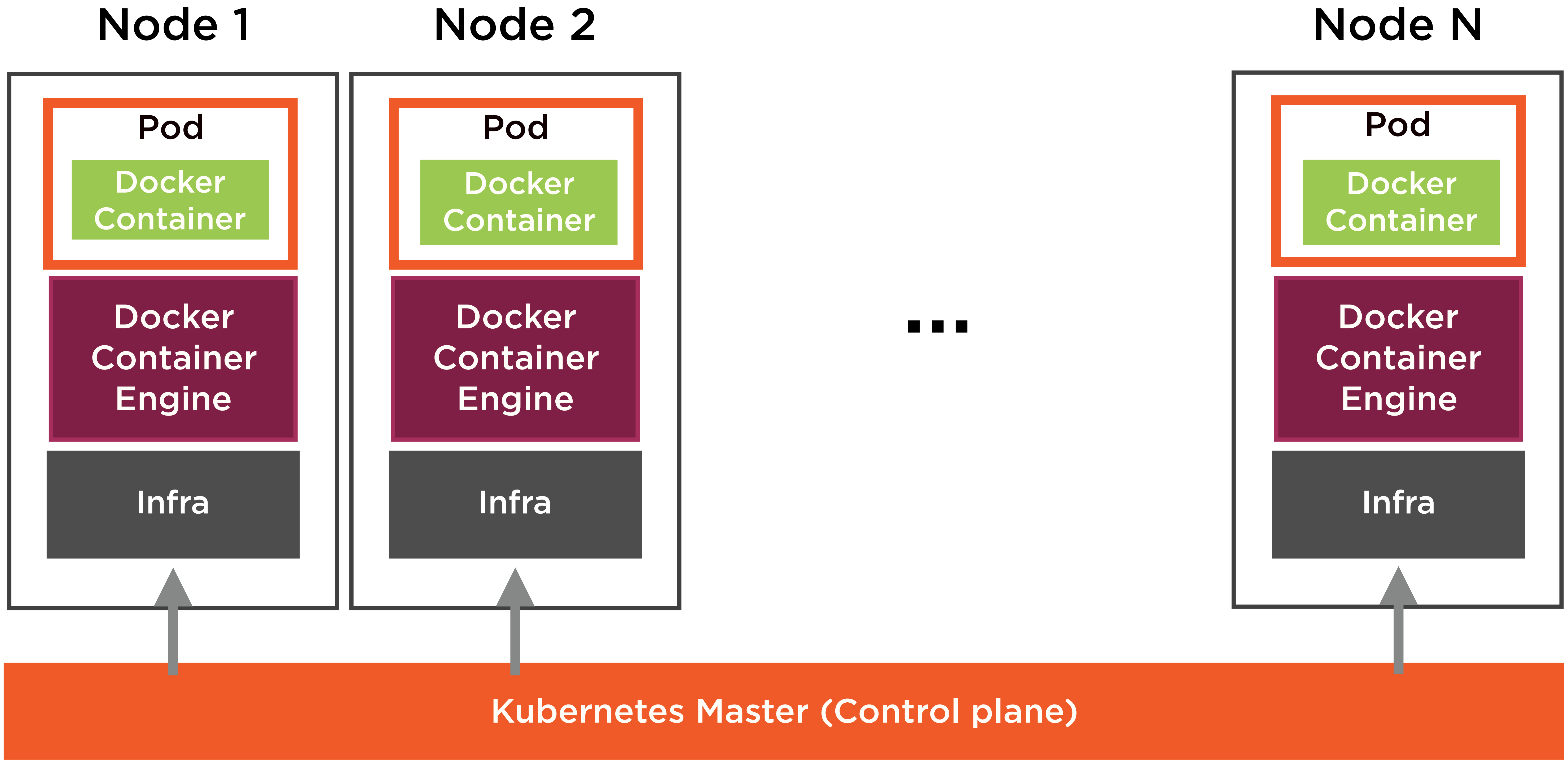
Kubernetes: Cluster Orchestration



Kubernetes: Containers Run Within Pods



Kubernetes: Cluster Orchestration



Pods as Atomic Units

Container deployment

All containers in pod are deployed, or none are

Node association

Entire pod is hosted on the same node

Pod is atomic unit of deployment in Kubernetes

Pods on Kubernetes Nodes

Encapsulates one or more containers

Pods run on nodes

Nodes are controlled by master

In GKE

- Nodes are GCE VM instances
- Master is managed by GKE service

Pods on Kubernetes Nodes

Can not run a container without enclosing pod

Pods provide isolation between containers

Pod acts as sandbox for enclosed containers

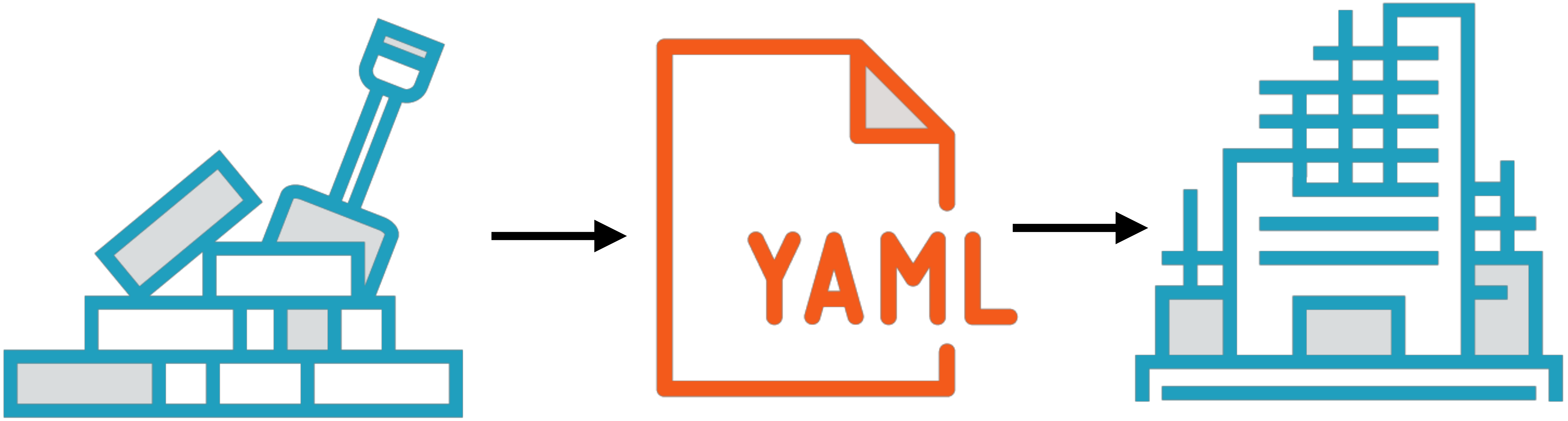
Multi-container pods are possible

- tightly-coupled
- not usually recommended

The **Pod object** is the lowest level of abstraction around a container

Every object is associated with a specification file which represents the **desired end state** of the object

YAML Specification Files



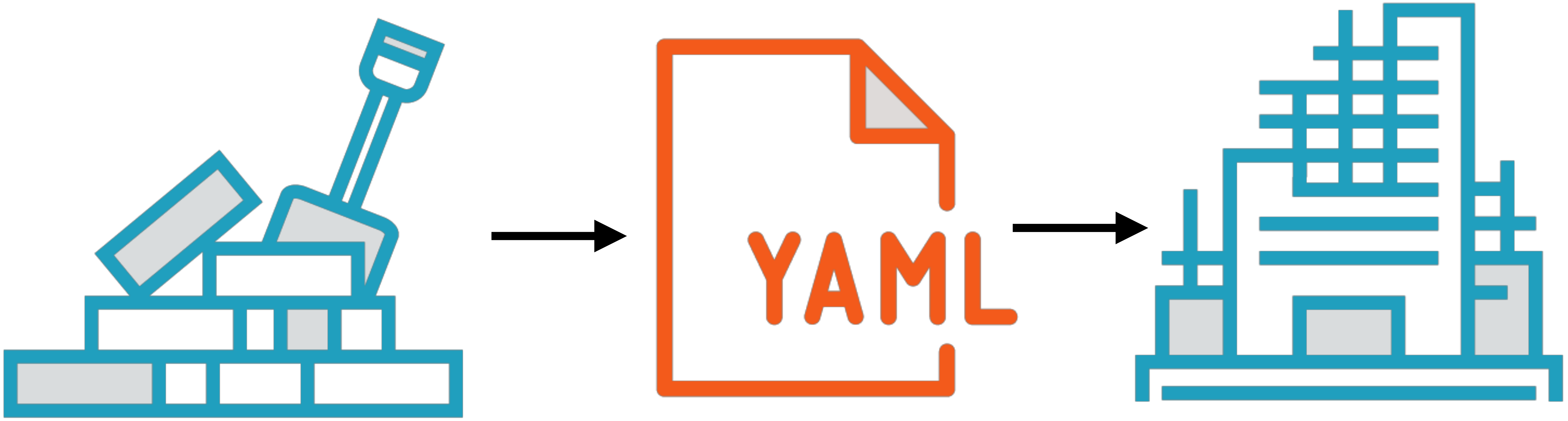
Current State

The current state of the object

Desired State

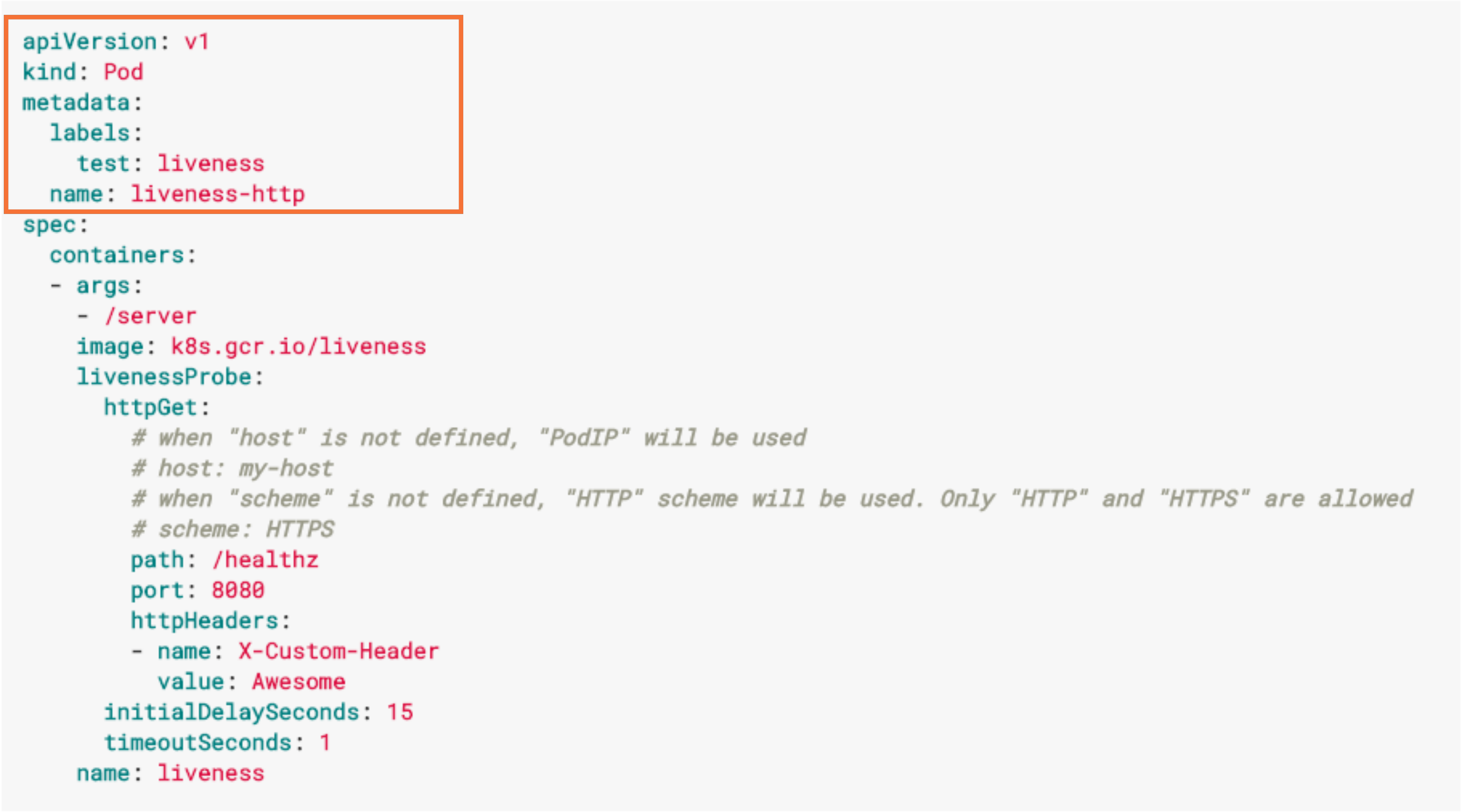
The end state of the object

YAML Specification Files



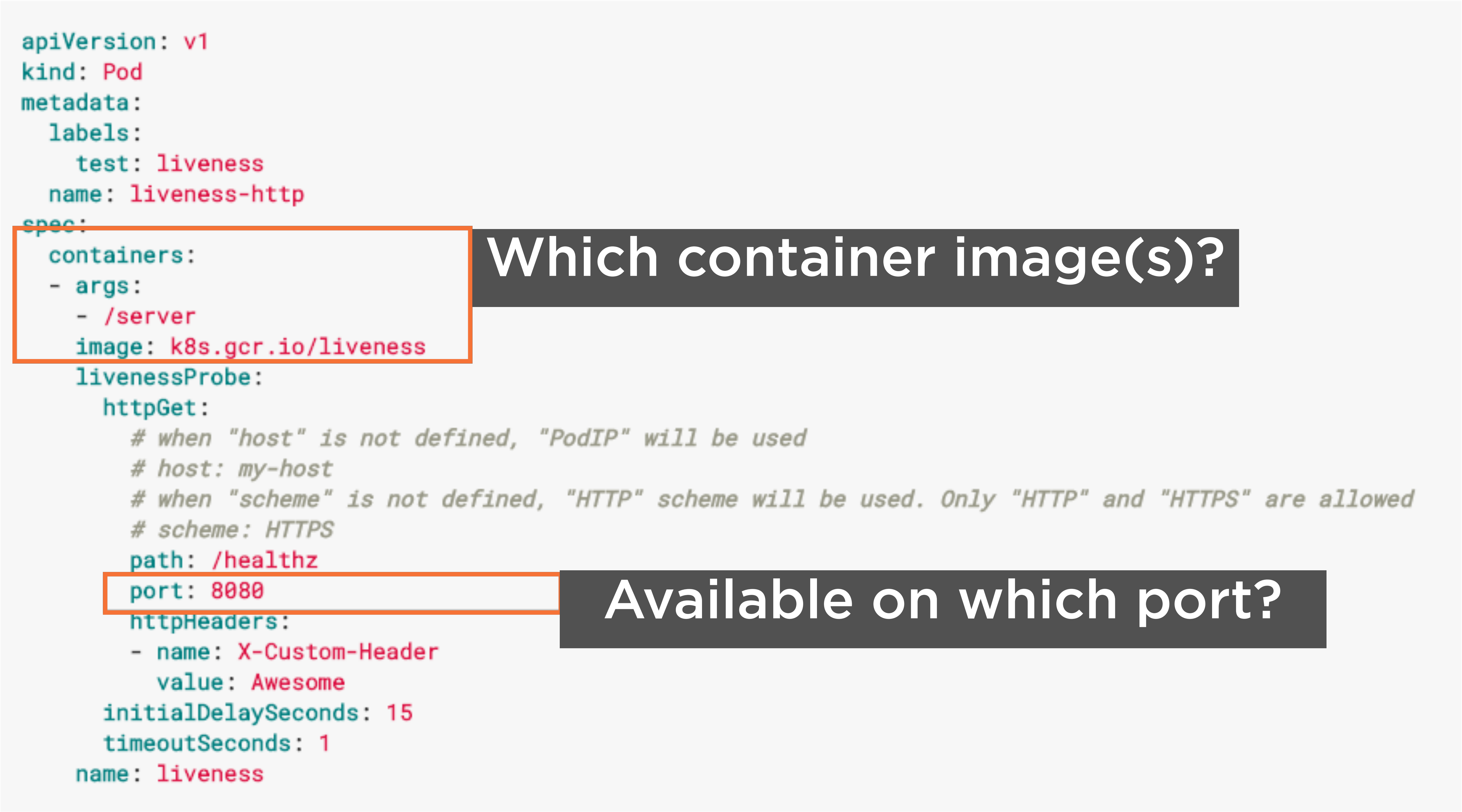
Controllers in the Kubernetes cluster run reconciliation loops to get the actual state to match the desired state

Pod Specification File



```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
      initialDelaySeconds: 15
      timeoutSeconds: 1
    name: liveness
```

Pod Specification File



```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        # when "host" is not defined, "PodIP" will be used
        # host: my-host
        # when "scheme" is not defined, "HTTP" scheme will be used. Only "HTTP" and "HTTPS" are allowed
        # scheme: HTTPS
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
      initialDelaySeconds: 15
      timeoutSeconds: 1
    name: liveness
```

Which container image(s)?

Available on which port?

Using the Google Kubernetes Engine almost completely eliminates the need to explicitly configure YAML files

Simply use the web console or the gcloud command line utility

Pod Spec

Pods are automatically assigned unique IP addresses

For each container

- Container image with source repository
- Port on which container will be available

Pods can specify shared storage volumes

Pod can contain multiple containers

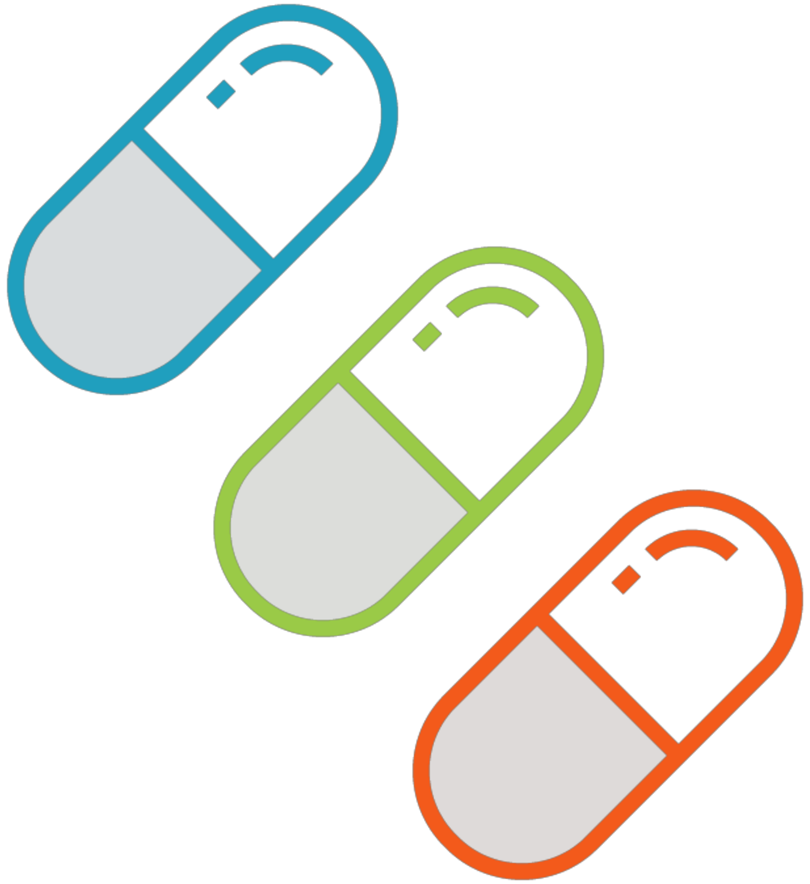
Multi-container Pods

Avoid unless necessary

Tight coupling, no isolation

No independent scaling possible

Prefer service-oriented architecture to monolithic design



Pod Status

Pending:

- Request accepted, but not yet fully created

Running:

- Pod bound to node, all containers started

Succeeded:

- All containers terminated successfully (will not be restarted)

Failed:

- All containers have terminated, and at least one failed

Unknown:

- Pod status could not be queried

Kubernetes for Container Orchestration

Limitation of Standalone Pods

No autohealing

- Crashed pods won’t restart automatically
- Need higher level orchestration

No scaling or autoscaling

- Overloaded pods don’t spawn more automatically
- Need higher level orchestration

No load balancing

- Pod IP addresses are ephemeral
- Pods can’t share load automatically
- Need higher level orchestration

Higher-level Abstractions

ReplicaSet, ReplicationController

- Scaling and healing

Deployment

- Versioning and rollback

Service

- Static (non-ephemeral) IP addresses
- Stable networking

Persistent volumes

- Non-ephemeral storage

Kubernetes as Orchestrator

Fault tolerance

- Recover from pod or node failures

Autohealing

- Crashed containers restart

Scaling

- ReplicaSets for multiple copies of the pod

Kubernetes as Orchestrator

Autoscaling

- Scale up and scale down based on load
- Horizontal pod autoscalers

Isolation

- Containers run inside pods

Load balancing

- Distribute traffic to containers

Kubernetes as Orchestrator

Master

Worker nodes

Job scheduling

Resource allocation

Comparing actual and desired state

Kubernetes as Orchestrator

Docker container engine : Java Runtime

Docker containers : Jar files

Kubernetes : Hadoop

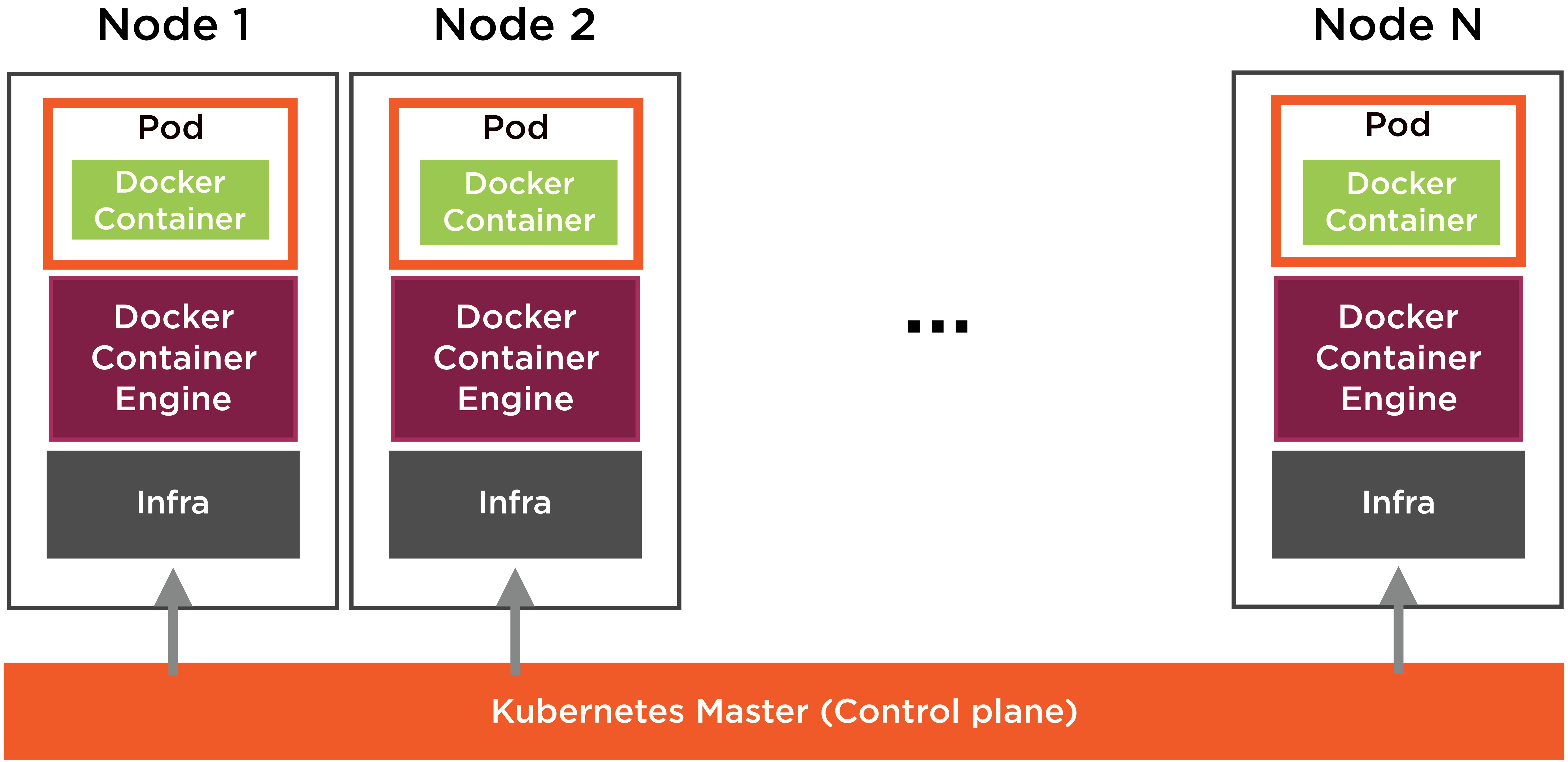
Kubernetes as Orchestrator

User interacts with Kubernetes

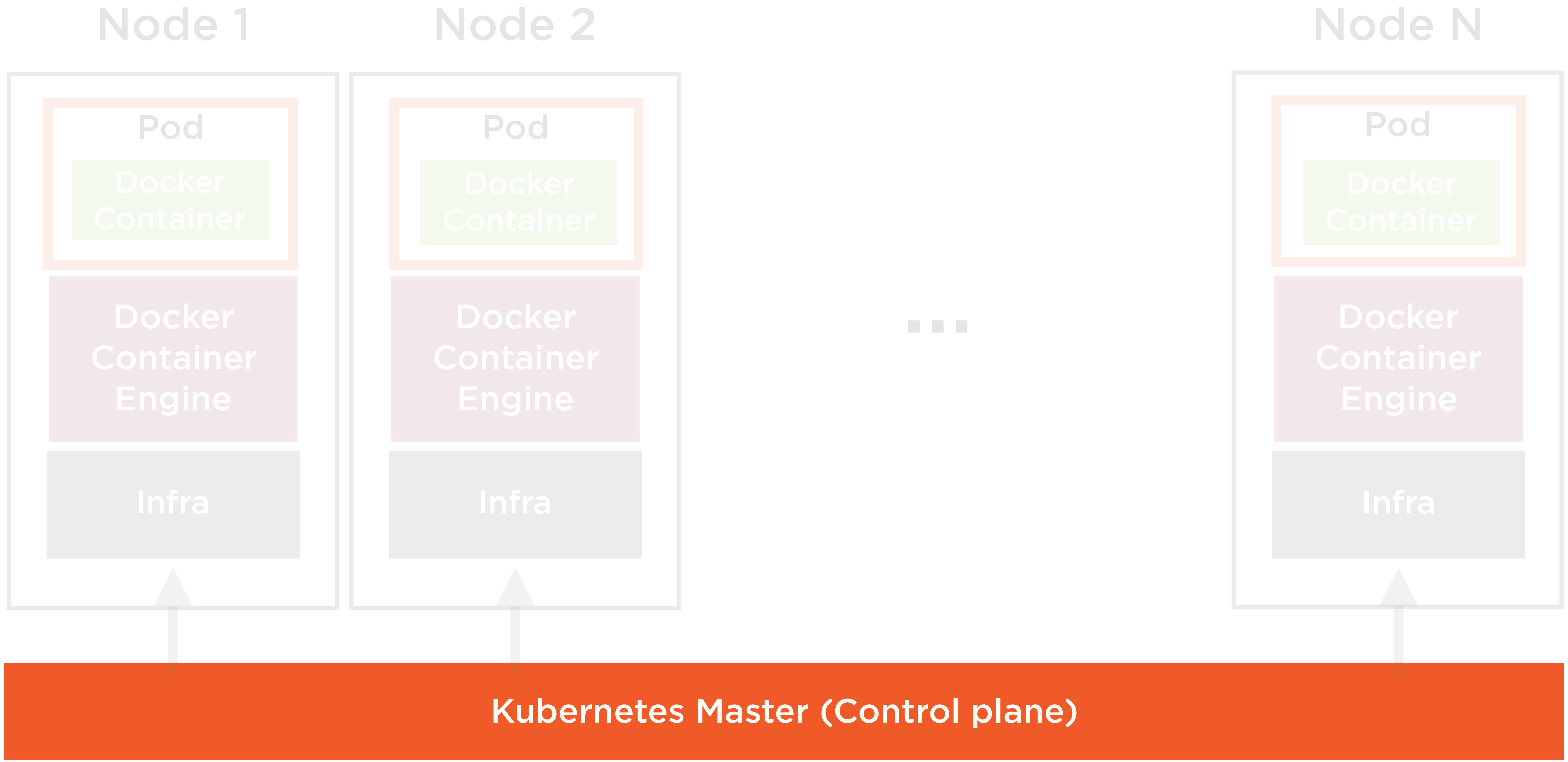
- kubectl and other command line tools
- YAML config files

Kubernetes relays and orchestrates pods

Kubernetes: Cluster Orchestration



Kubernetes: Cluster Orchestration



Replication and Deployment

Higher-level Abstractions

ReplicaSet, ReplicationController

- Scaling and healing

Deployment

- Versioning and rollback

Service

- Static (non-ephemeral) IP addresses
- Stable networking

Persistent volumes

- Non-ephemeral storage

Higher-level Abstractions

ReplicaSet, ReplicationController

- Scaling and healing

Deployment

- Versioning and rollback

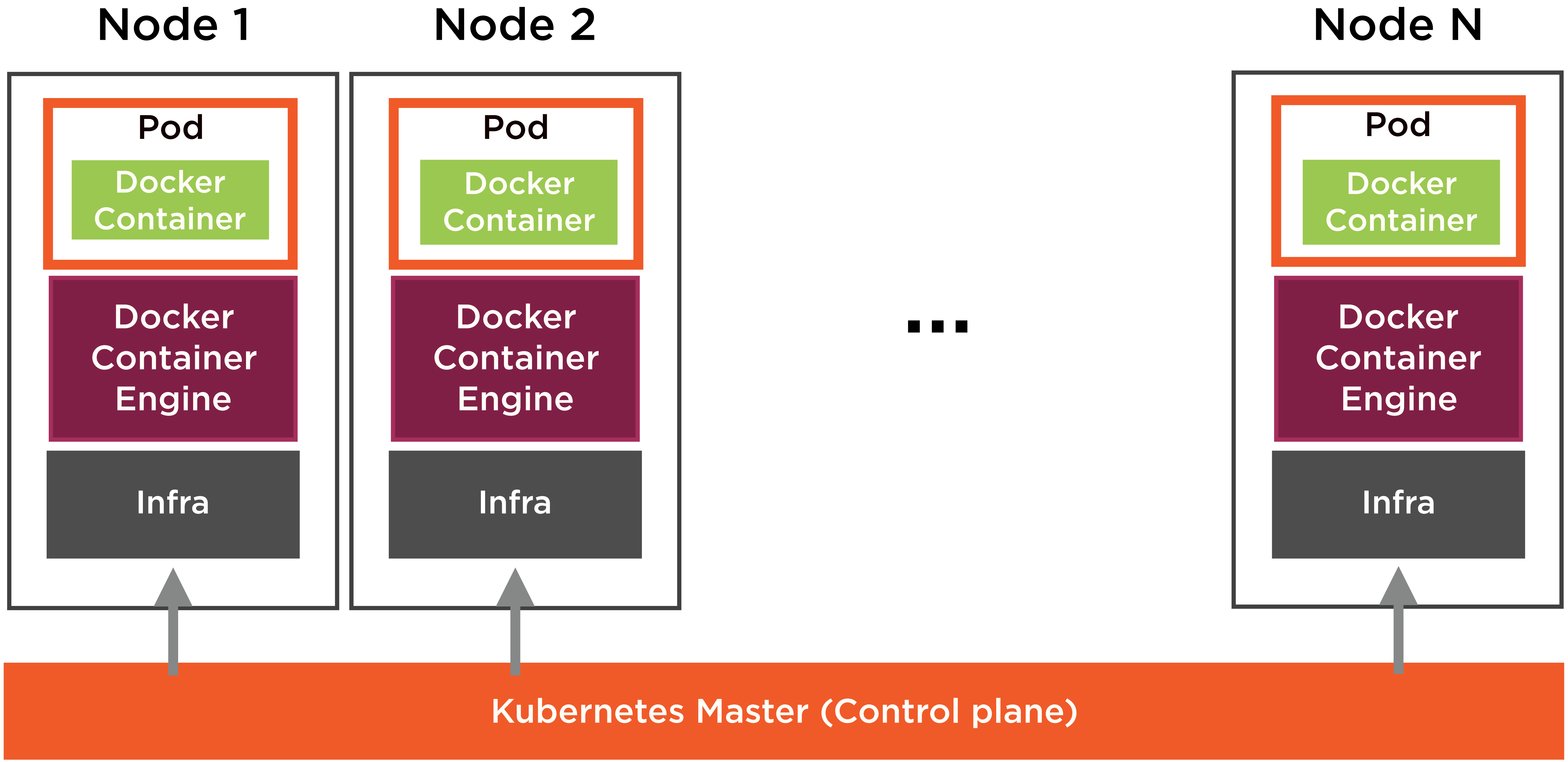
Service

- Static (non-ephemeral) IP addresses
- Stable networking

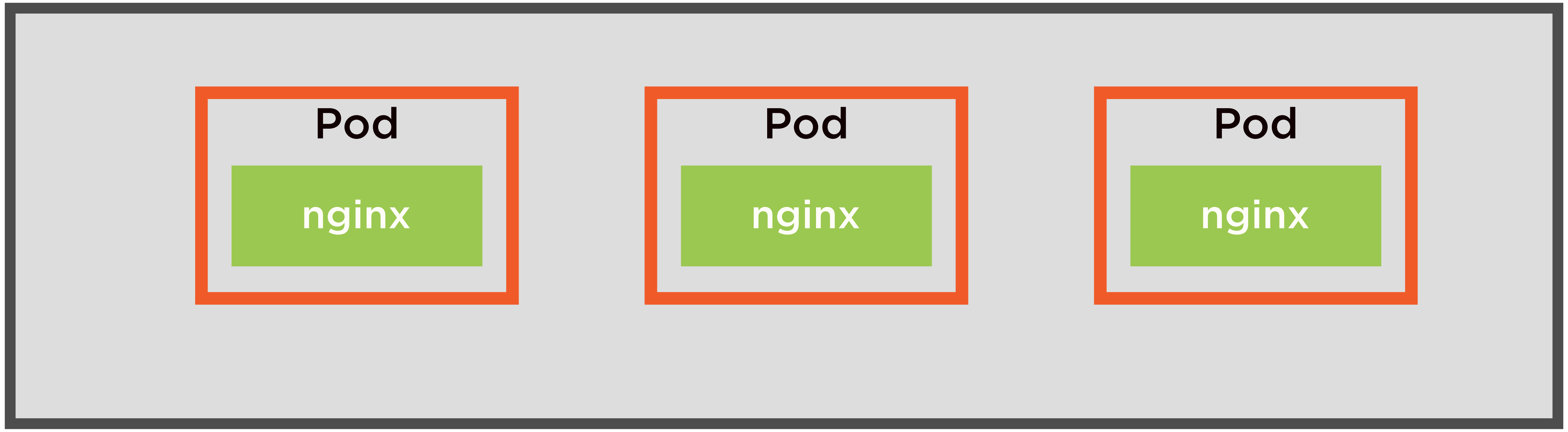
Persistent volumes

- Non-ephemeral storage

Kubernetes: Cluster Orchestration



ReplicaSet



Multiple copies of the pod are managed together using a ReplicaSet

Self-healing and autoscaling for our pods

Multiple instances of the pod are created and deployed to clusters

Replicas are represented by a ReplicaSet object

ReplicaSet

If pod crashes, ReplicaSet will start a new one

Key to scaling and healing

All pods are replicas of each other

ReplicaSet

Loosely coupled with pods

A ReplicaSet object will govern all pods that match its label selector

Users must ensure no conflicts, orphans

The ReplicaSet Object

Multiple identical pods which are replicas of each other

Higher-level Abstractions

ReplicaSet, ReplicationController

- Scaling and healing

Deployment

- Versioning and rollback

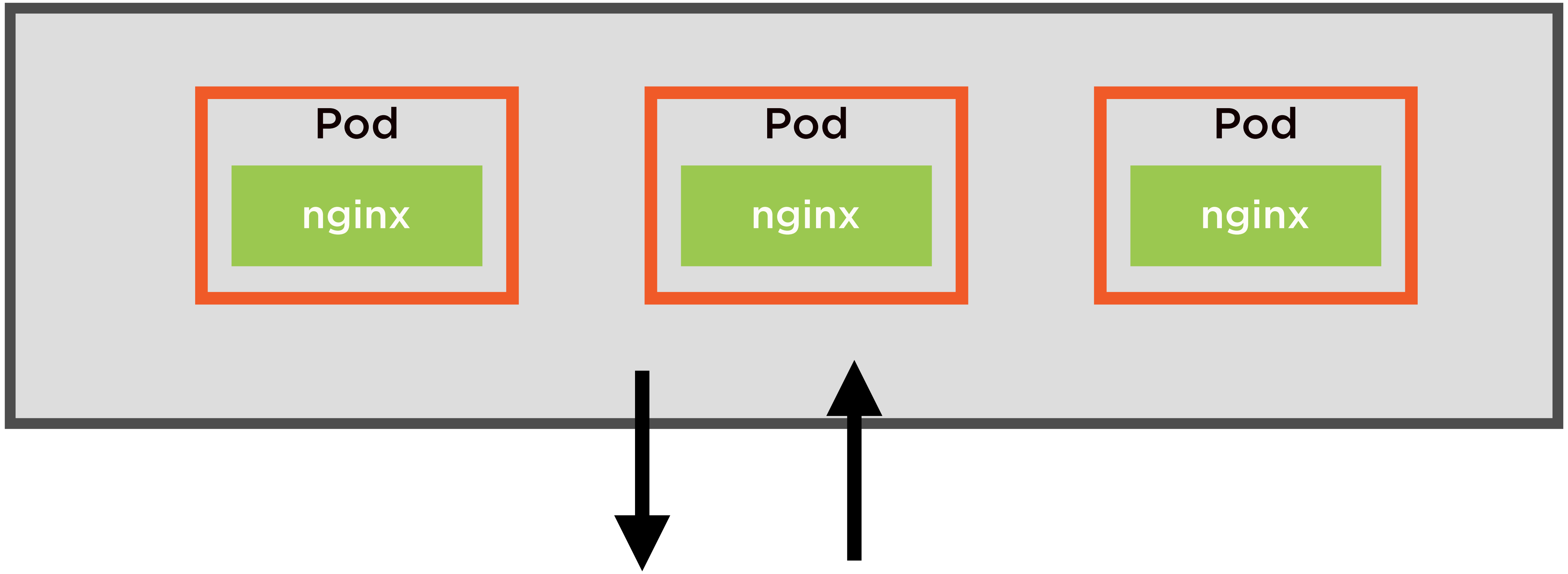
Service

- Static (non-ephemeral) IP addresses
- Stable networking

Persistent volumes

- Non-ephemeral storage

Deployment



Works with ReplicaSets to add versioning, rollback and other advanced deployment functionality

The Deployment Object

Adds on deployment and rollback functionality

Deployment Objects

Easy to push out new version of container

Triggers creation of new ReplicaSet and new containers

Pods in old ReplicaSet gradually reduced to zero

Deployment Rollbacks

Every change to a Deployment object triggers creation of a new revision

Trivial to rollback to previous revision

Offers versioning support

Use Cases of Deployments

Manual scaling: Edit number of replicas

Autoscaling: Use HPA with deployment as target

Can pause/resume deployments midway

Can monitor deployment status

Services

Limitation of Standalone Pods

No autohealing

- Crashed pods won’t restart automatically
- Need higher level orchestration

No scaling or autoscaling

- Overloaded pods don’t spawn more automatically
- Need higher level orchestration

No load balancing

- Pod IP addresses are ephemeral
- Pods can’t share load automatically
- Need higher level orchestration

Higher-level Abstractions

ReplicaSet, ReplicationController

- Scaling and healing

Deployment

- Versioning and rollback

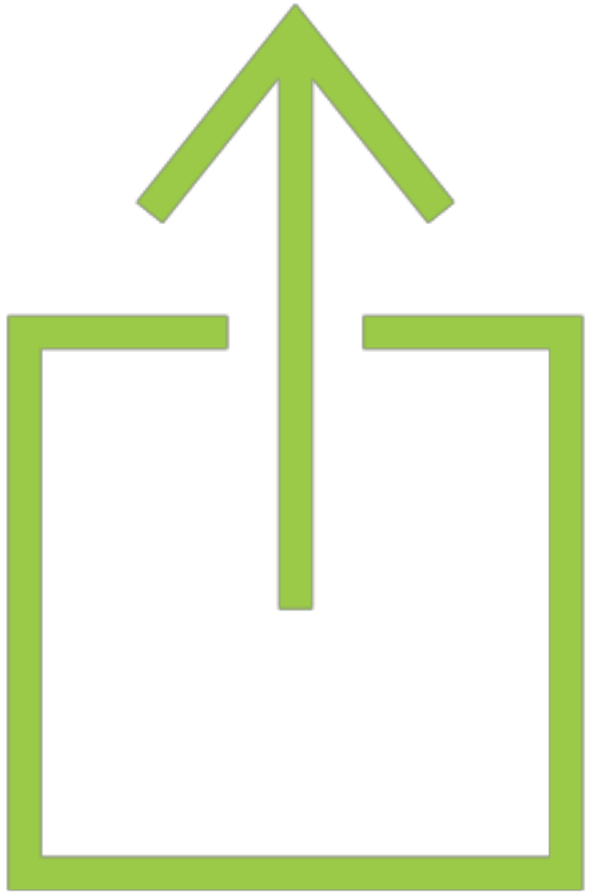
Service

- Static (non-ephemeral) IP addresses
- Stable networking

Persistent volumes

- Non-ephemeral storage

Higher-level Abstractions



ReplicaSet, ReplicationController

- Scaling and healing

Deployment

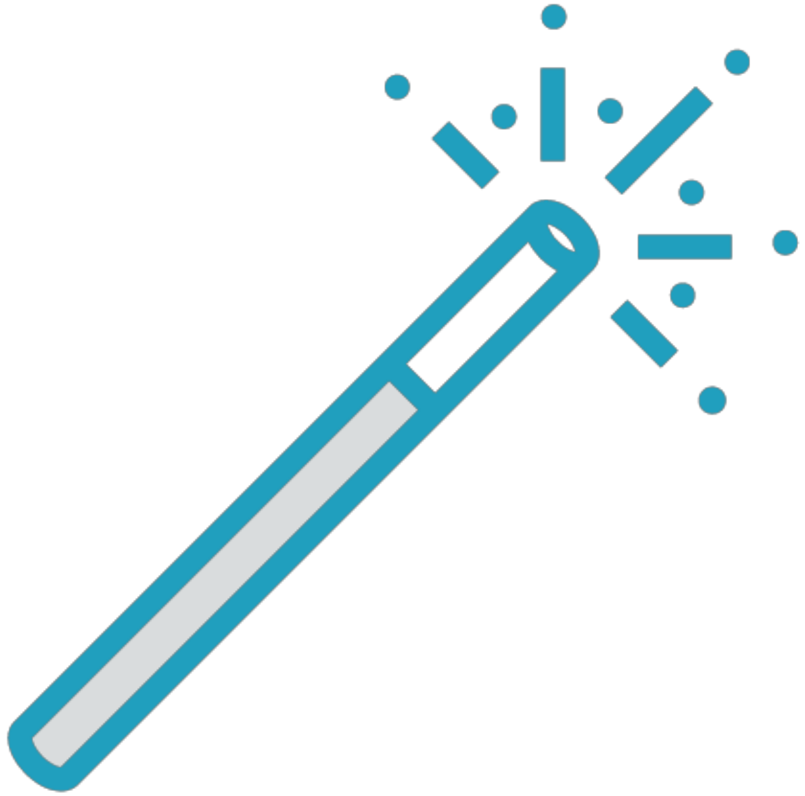
- Versioning and rollback

Service

- Static (non-ephemeral) IP addresses
- Stable networking

Persistent volumes

- Non-ephemeral storage



Ephemeral IP Addresses

Containers expose ports in pod spec

Pod IP addresses are ephemeral

Poses a problem for clients

How are they to know where to send request?

Services provide stable IP addresses for external connections and load balancing

Service Objects

Provides stable (non-ephemeral) IP address

Connects to set of back-end pods

Set of pods changes dynamically

- Logically selected via label selector

Front-end IP remains unchanged

Basic load balancing too

Endpoint Object

Each service has associated endpoint object

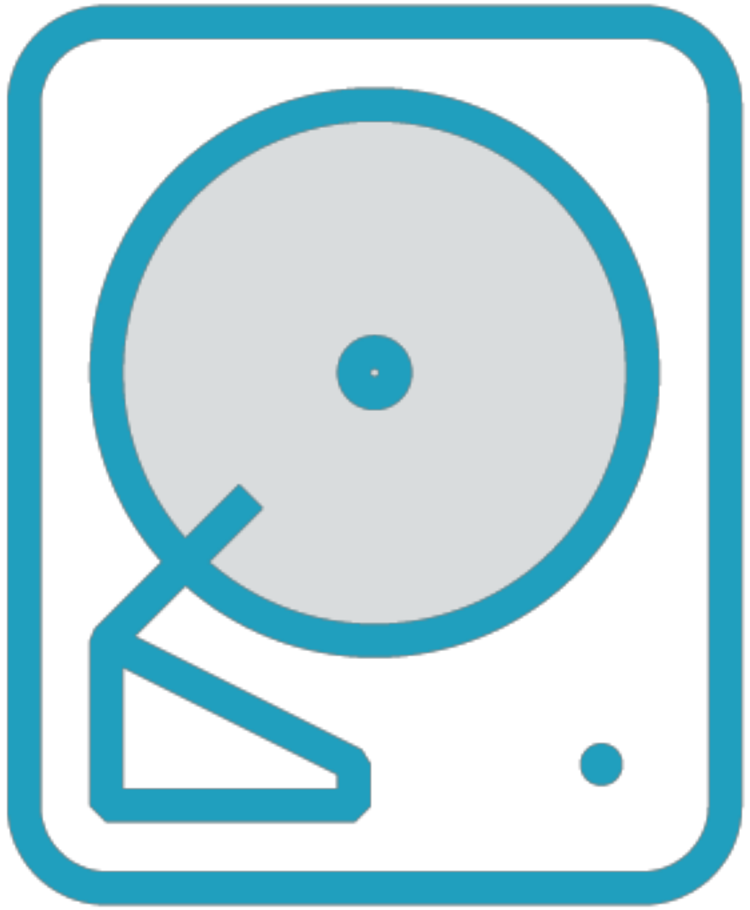
Dynamic list of pods selected by service

Populated by Kubernetes

Dynamically updates as pods are created/ deleted

Volume Abstractions

Storage with Containers



On disk files within a container

- Only accessible to the container itself
- Ephemeral: is lost when the container stops or crashes

Volume abstractions

- A directory accessible to all containers in a pod
- Specified in the PodSpec
- Have the same lifetime as the enclosing pod

For durable storage use persistent volumes

The volume is preserved even when the pod is removed and can be handed off to another pod

Higher-level Abstractions

ReplicaSet, ReplicationController

- Scaling and healing

Deployment

- Versioning and rollback

Service

- Static (non-ephemeral) IP addresses
- Stable networking

Persistent volumes

- Non-ephemeral storage

Higher-level Abstractions

ReplicaSet, ReplicationController

- Scaling and healing

Deployment

- Versioning and rollback

Service

- Static (non-ephemeral) IP addresses
- Stable networking

Persistent volumes

- Non-ephemeral storage

Volumes

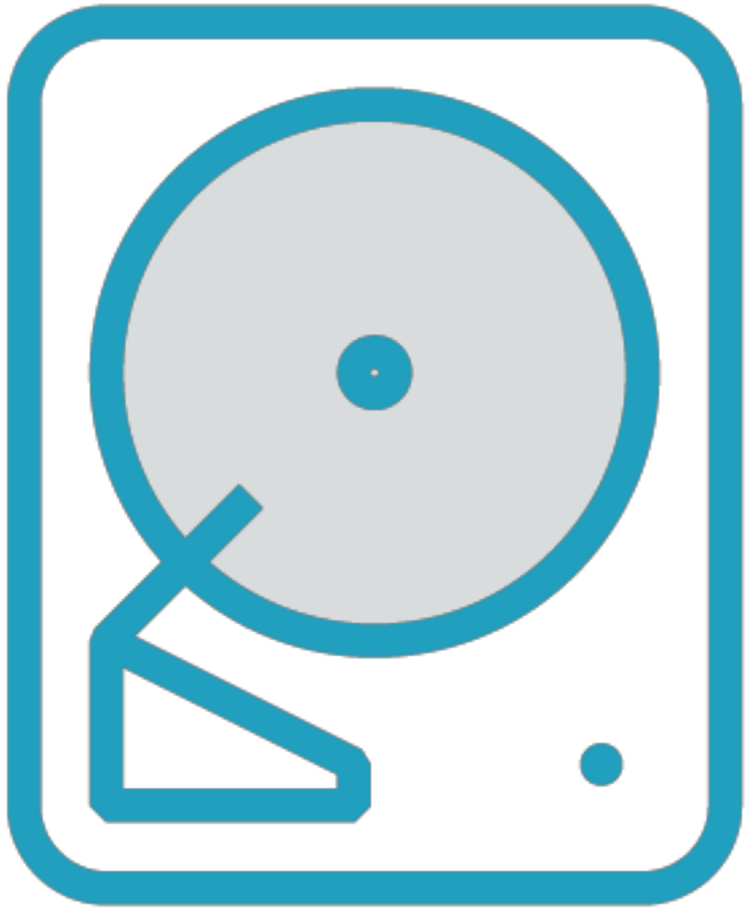
Permanence

Storage that outlives an individual container or individual pod

Shared state

Safe mechanism for containers in pod to share state

Persistent volumes are volumes whose life is not tied to an individual pod



Types of Volumes

configMap

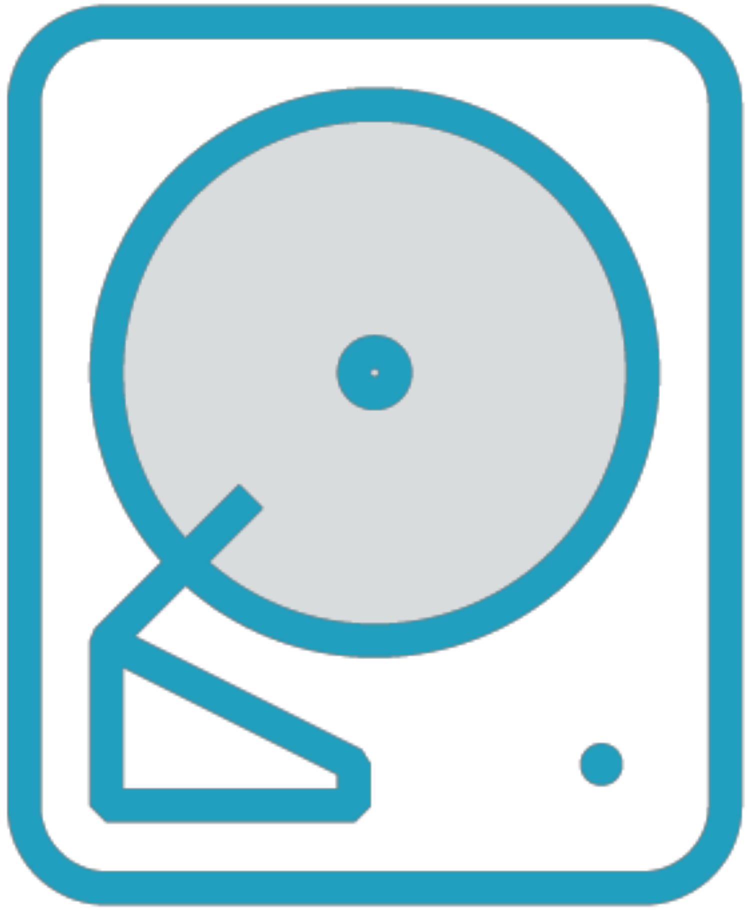
emptyDir

gitRepo

secret

hostPath

Using Volumes



Define volume in pod spec

Have each container mount volume

Each container mounts independently

At different path

Volumes and Persistent Volumes

Volumes

Storage abstraction with life longer than individual container inside pod

Life tied to life of pod

Persistent Volumes

Storage abstraction with life longer than individual pod

Life not tied to life of pod

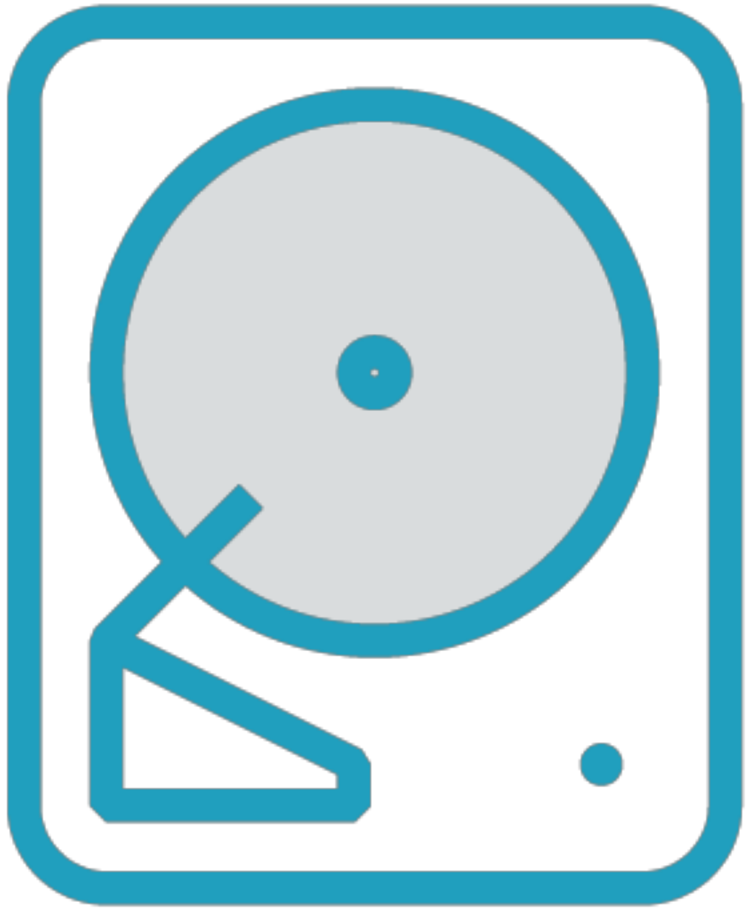
Cloud-specific Persistent Volumes

awsElasticBlockStore

azureDisk

azureFile

gcePersistentDisk



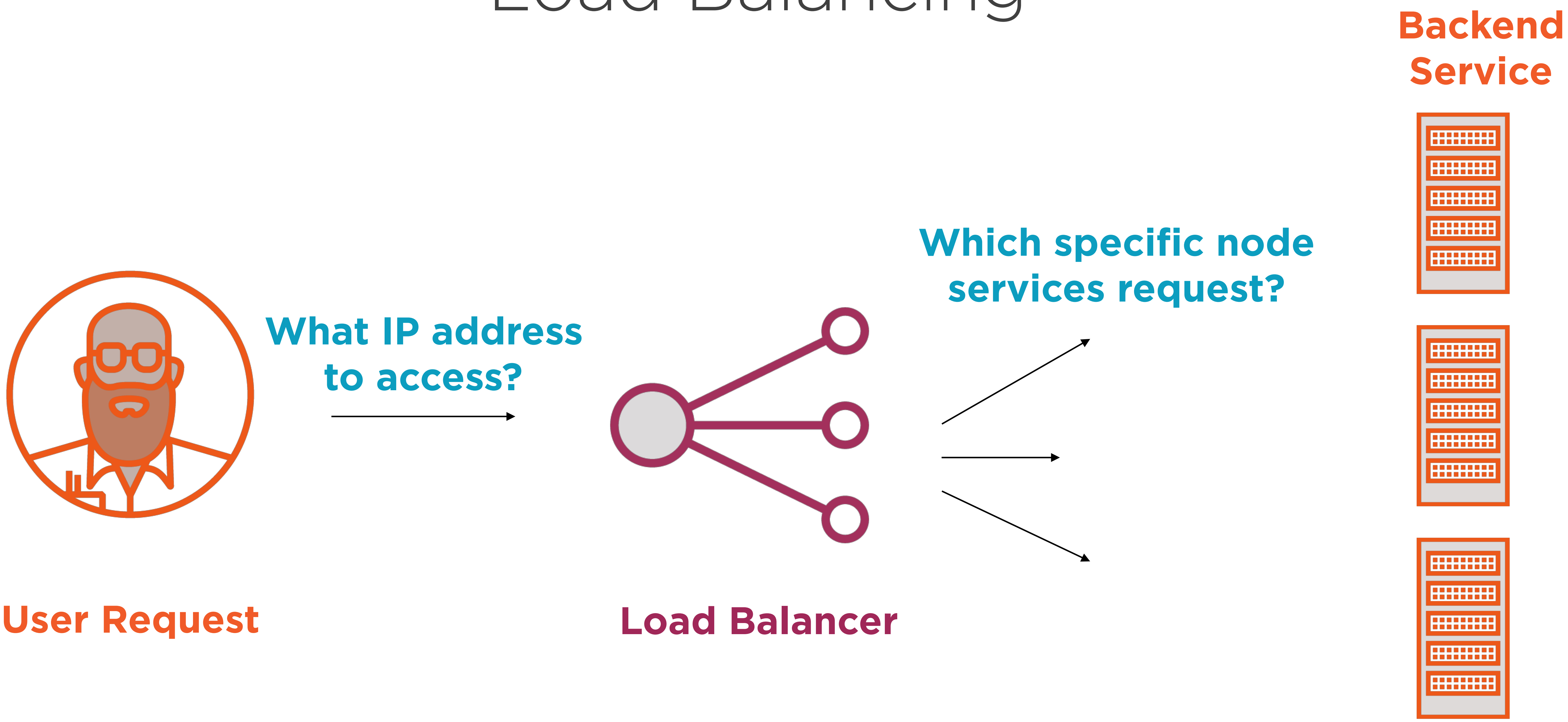
Persistent Volumes

Two types of provisioning

- Static: Administrator pre-creates volume
- Dynamic: Containers gain access by filing PersistentVolumeClaim

Load Balancing

Load Balancing


Backend Service
What IP address to access?
Which specific node services request?
User Request
Load Balancer

Load Balancer to the rescue

Load Balancers

Stable front-end IP

Forwarding rules to funnel traffic

Connect to backend service

Distribute load intelligently

Health checks to avoid unhealthy instances

Exposing a service on the GKE allows the option of configuring a load balancer to distribute traffic

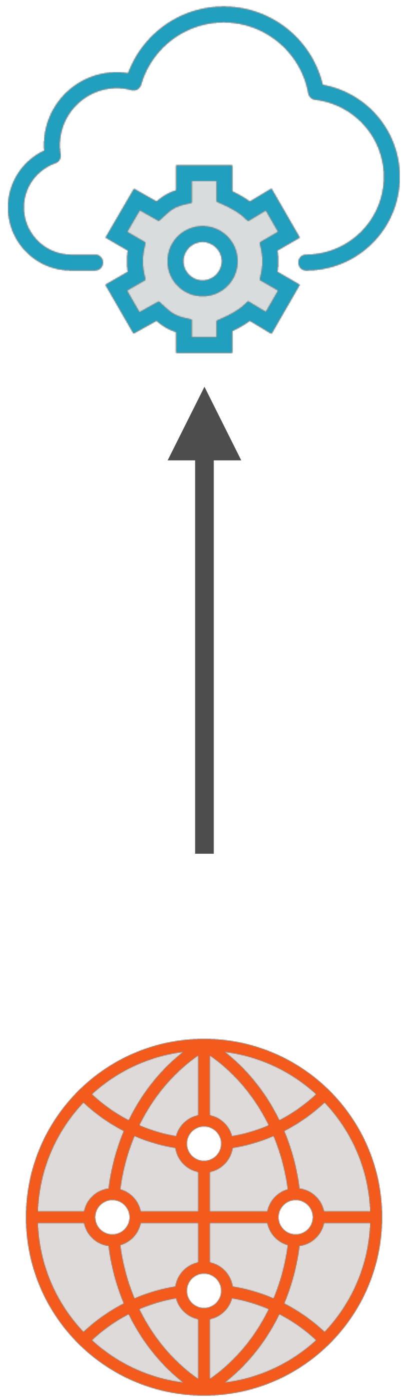
Ingress

Ingress Object

Kubernetes object defining a collection of rules that allow inbound connections to reach cluster services. On GKE, a single ingress object can control access to multiple services

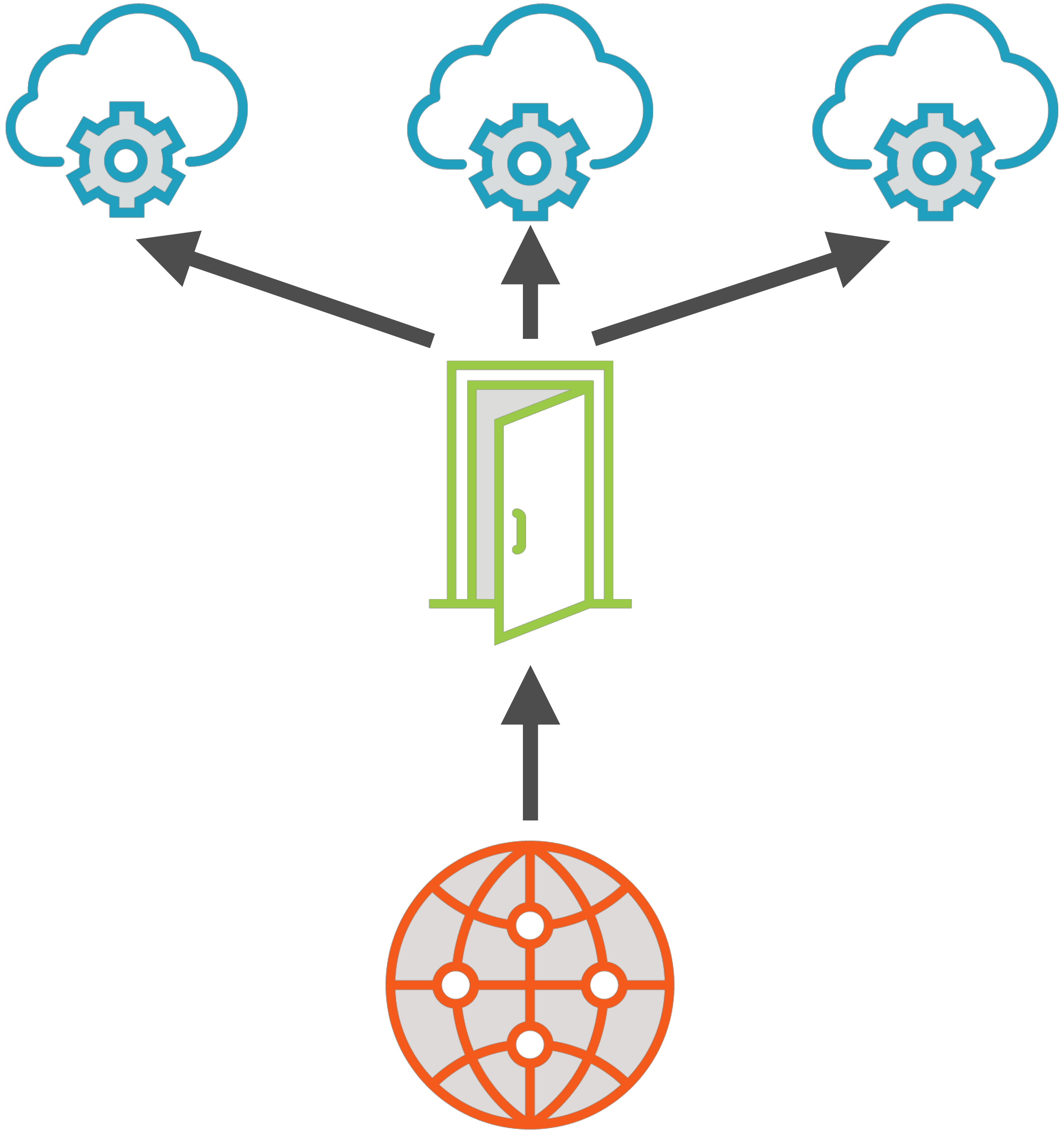
A single service can expose an IP address for access

Ingress



Ingress

With multiple services it makes sense to have rules defined using an ingress object



Ingress Objects

Can be configured to support

- Externally-reachable URLs
- Load balancing
- SSL termination
- Name-based virtual hosting

Ingress Objects

GKE fully supports both ingress and load balancer objects

On other platforms need to do some additional work

(Need to configure ingress object as pod)

GKE Ingress Objects

GKE clusters have HttpLoadBalancing add-on enabled

Causes additional controller to run on master

This controller supports HTTP(S) load balancing for ingress objects

StatefulSets and DaemonSets

StatefulSets

A set of pods, similar to ReplicaSet

Important difference from ReplicaSet

- Pods created unique
- Identified by name
- Not interchangeable
- Always associated with persistent volume

DaemonSet

Manages groups of replicated pods

Attempts to keep one pod per node

Across all nodes or a subset

As nodes added, pods created too

As nodes removed, pods are garbage collected

DaemonSet

Cluster storage daemons

Log collection daemons

Node monitoring daemons

Horizontal Pod Autoscaler

Autoscaling with Kubernetes

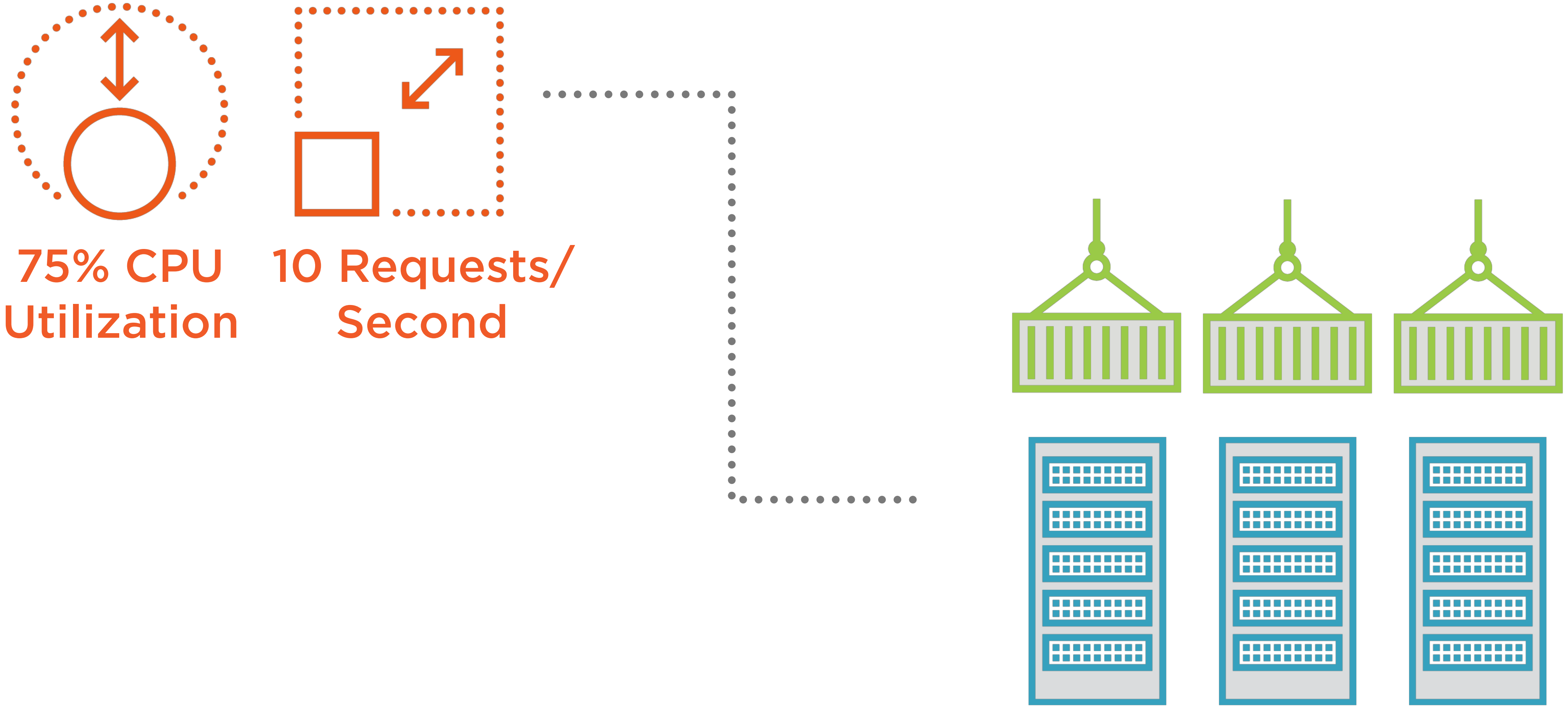
Autoscaling

Compute capacity automatically changes with changing need

Autohealing

Platform ensures health of compute resources

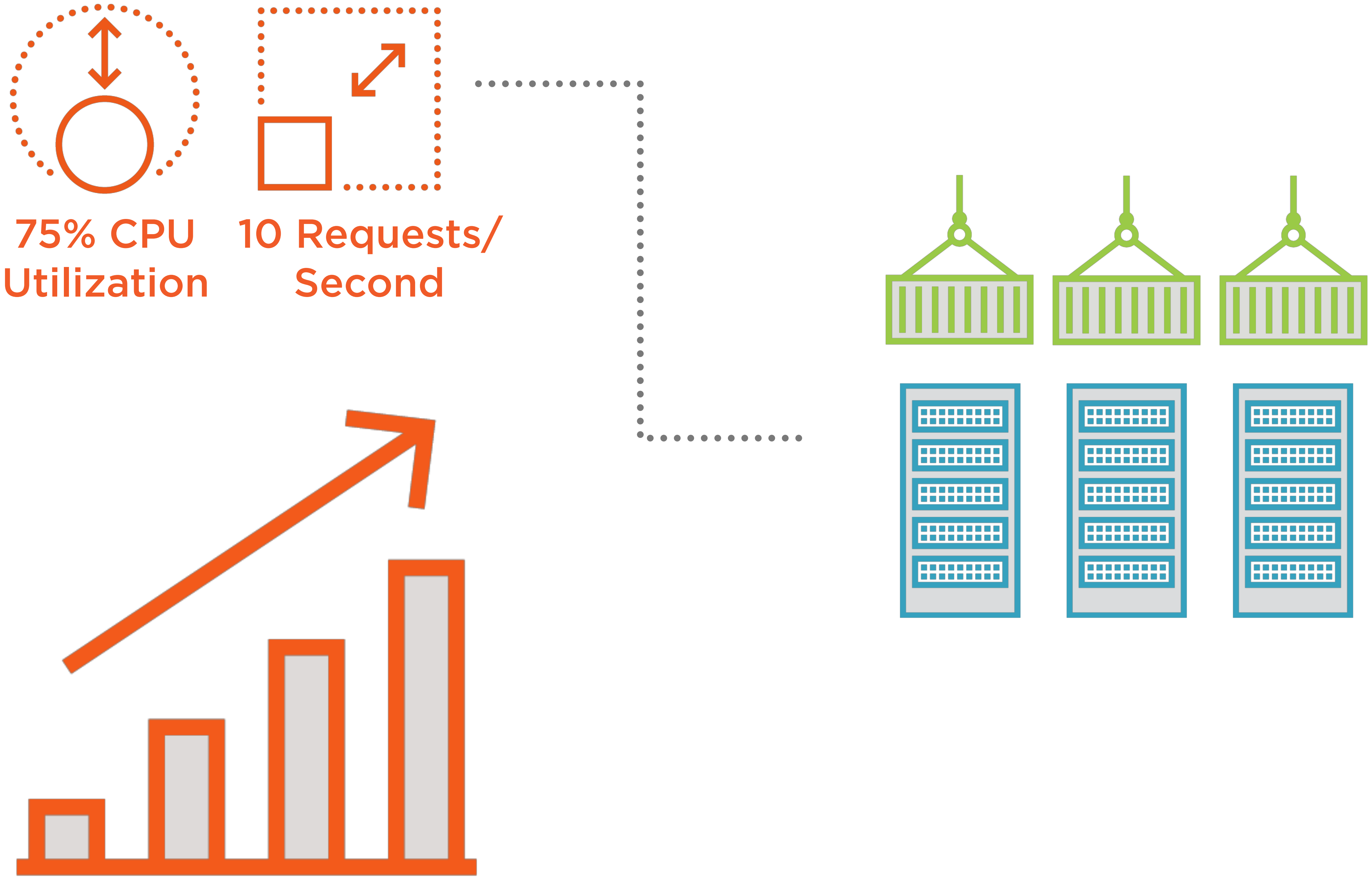
Autoscaling with Kubernetes



75% CPU Utilization

10 Requests/ Second

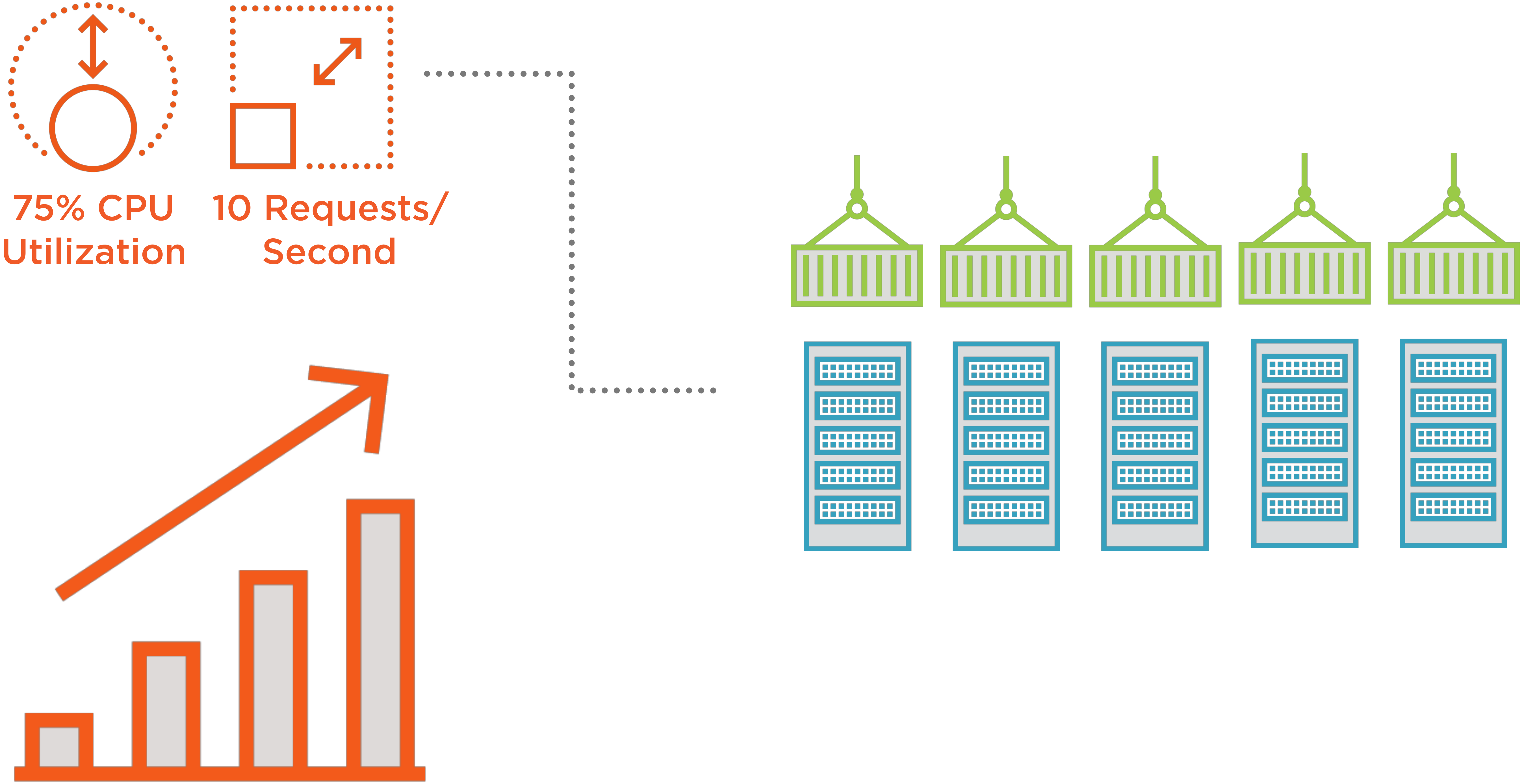
Autoscaling with Kubernetes



75% CPU Utilization

10 Requests/ Second

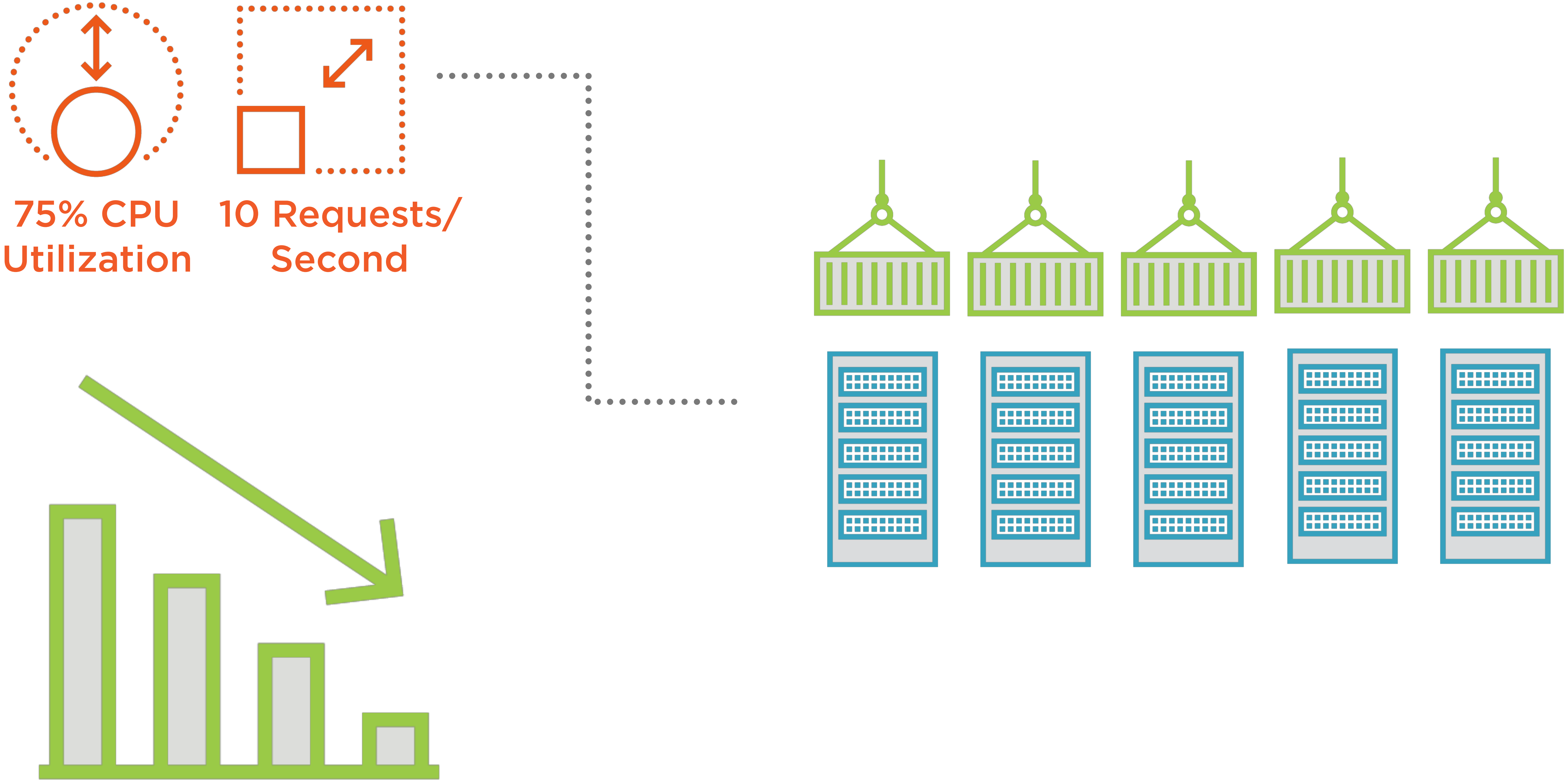
Autoscaling with Kubernetes



75% CPU Utilization

10 Requests/ Second

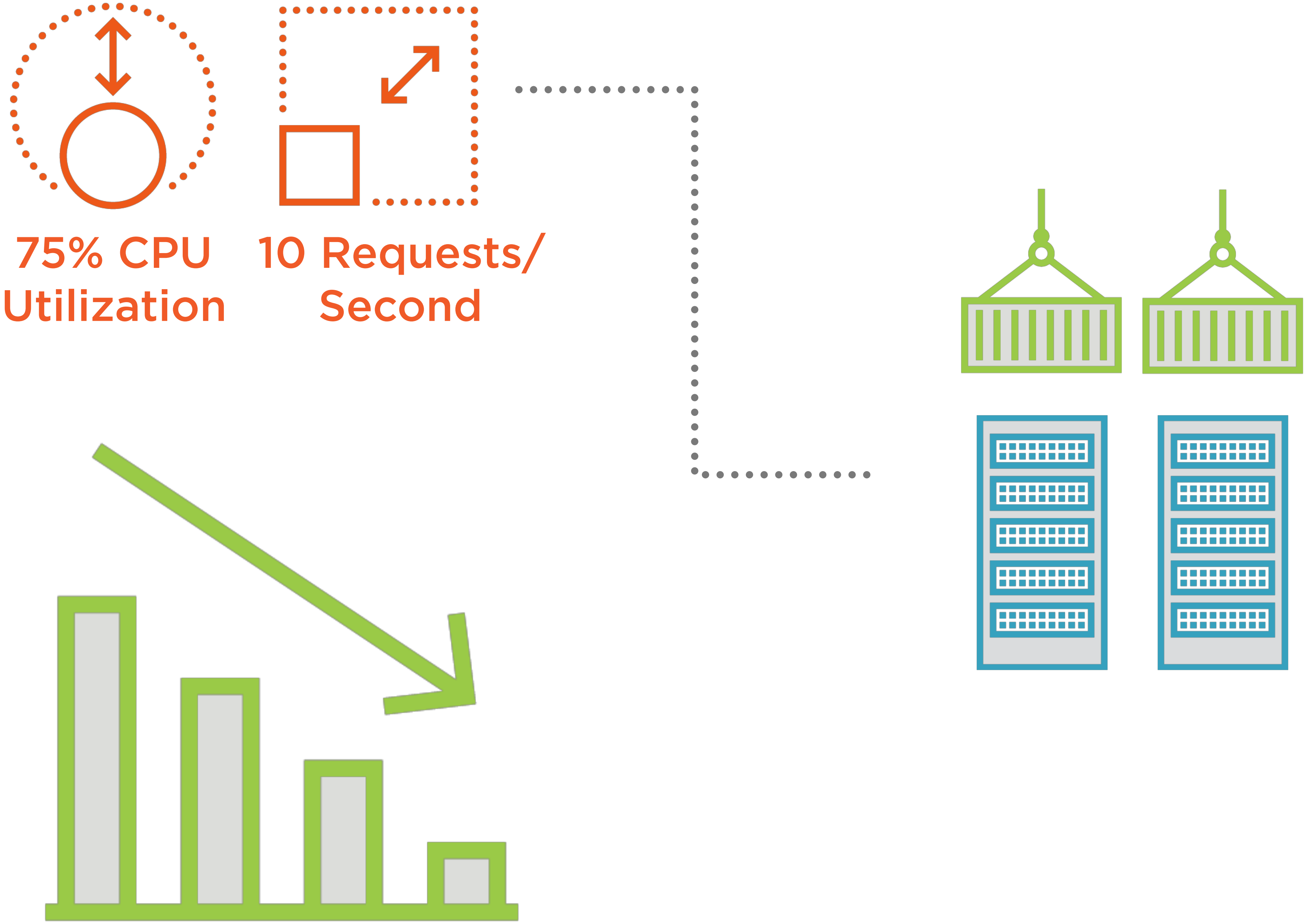
Autoscaling with Kubernetes



75% CPU Utilization

10 Requests/ Second

Autoscaling with Kubernetes



75% CPU Utilization

10 Requests/ Second

Horizontal Pod Autoscaler

Even higher level abstraction

Specify any scalable object as target

Along with autoscaling policy

HPA Targets

Scalable objects

- ReplicaSet
- Deployment

Can’t target non-scalable objects

- DaemonSets

HPA Metric

CPU utilization

Custom metrics

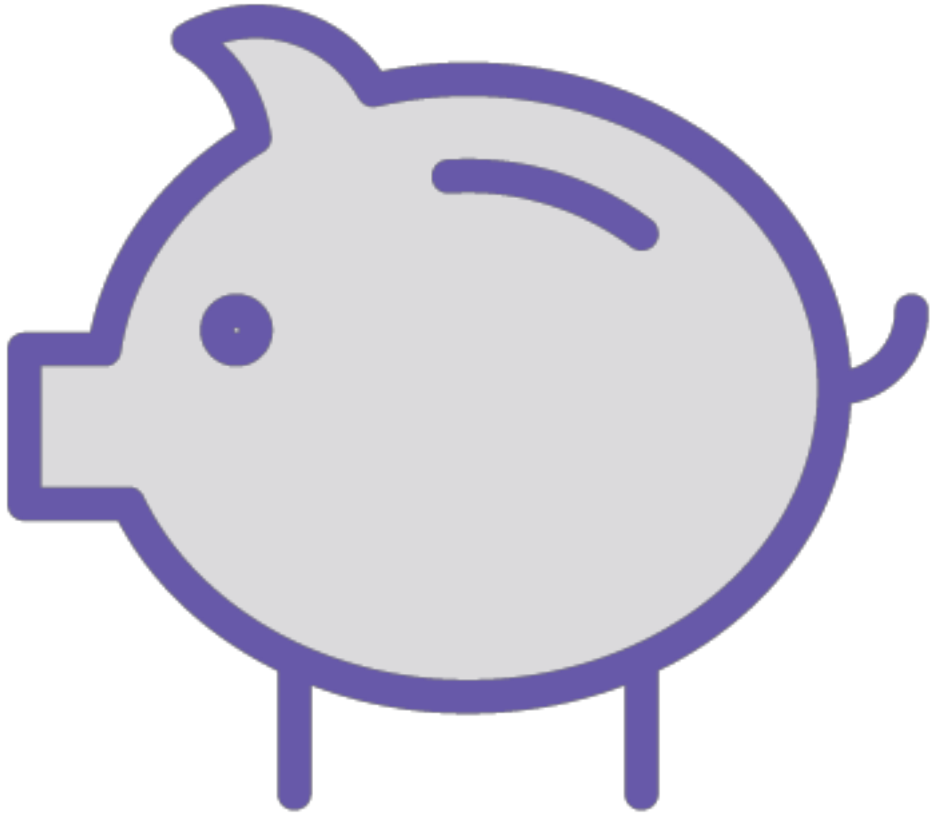
Preventing Thrashing

Thrashing occurs when HPA can’t find stable size

Use cool-down periods to avoid this

Set intervals between successive operations of same kind

GKE Pricing



Compute and Storage

GCE instances are used for nodes in the cluster

Billed for instances based on GCE usage

Per-second basis with 1 minute minimum cost

Costs vary based on machine type, region, type of disk etc.

Summary

Containers for lightweight compute

Ideal for hybrid, multi-cloud

Kubernetes is a container orchestration technology

Industry standard with Google origins

GKE for Kubernetes on GCP