

Container Technology and GCIS

Container-as-a-Service

Training Workshop
For OGCIO (1-Day)

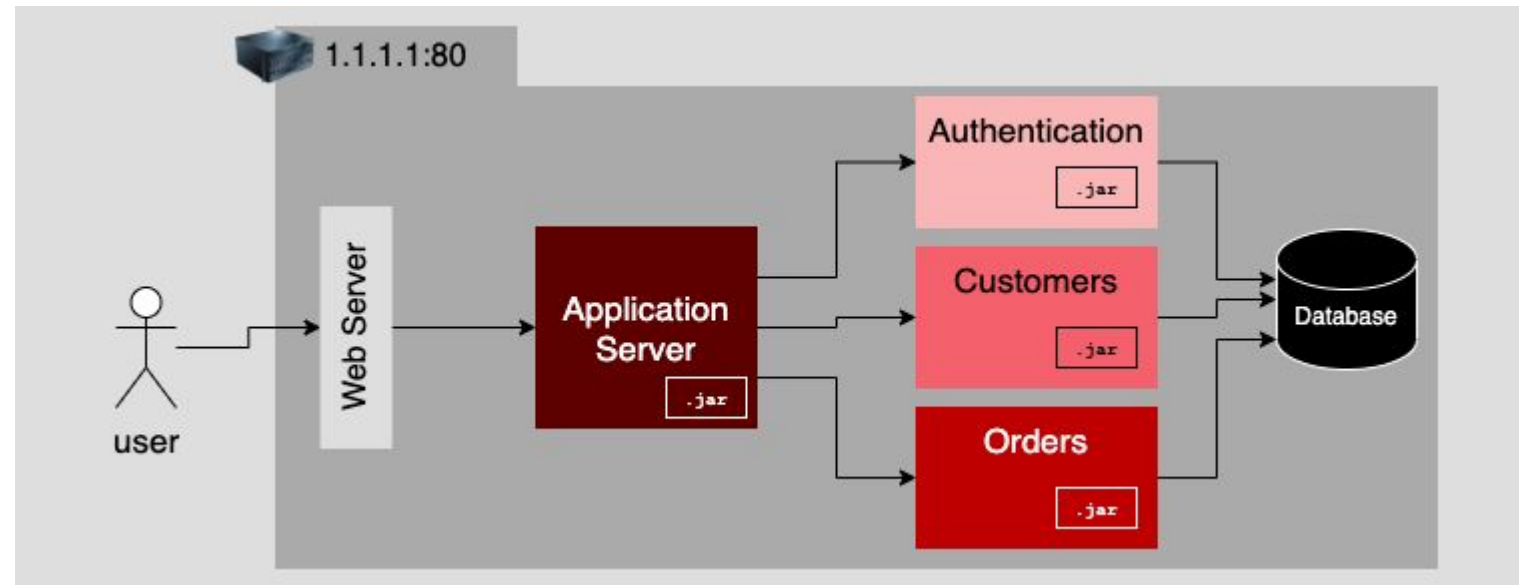
Sunny Chan
Consultant
Red Hat

What we'll discuss today

- ▶ Micro Services
- ▶ Container Technology
- ▶ Container
- ▶ Container Orchestrators
- ▶ Kubernetes Architecture
- ▶ Kubernetes in details
- ▶ Service Mesh
- ▶ CI / CD
- ▶ Case Study
- ▶ GCIS
Container-as-a-service

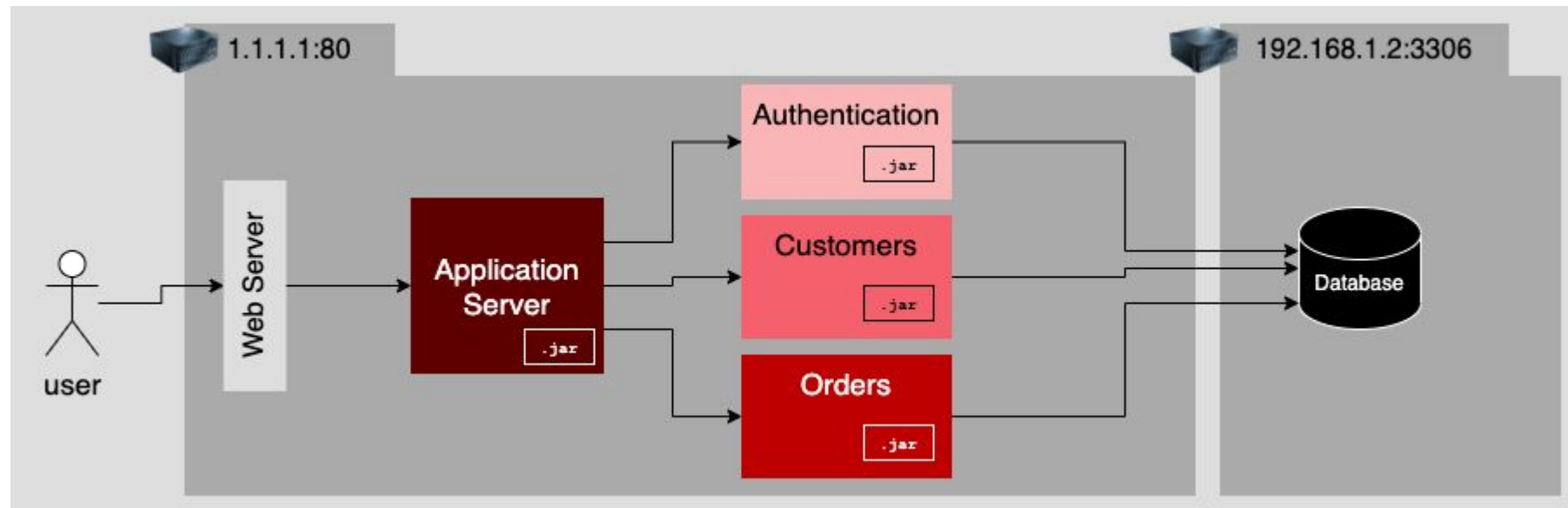
Overview of Microservices

Introduction to monolithic architecture



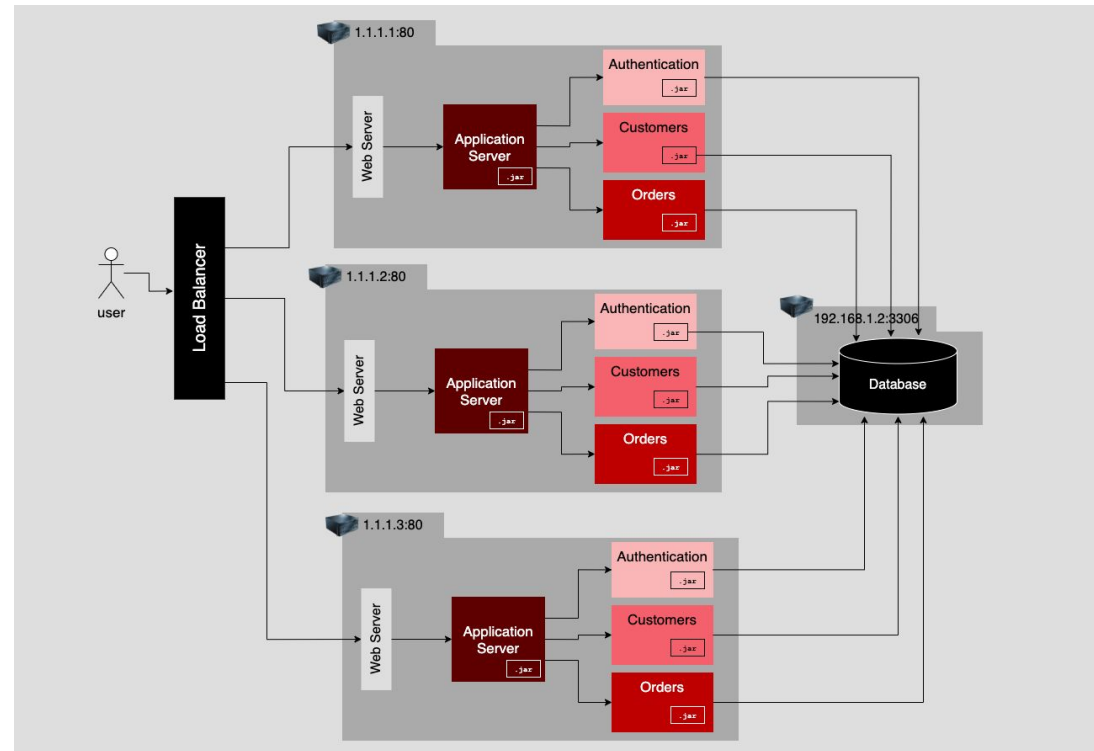
everything the application needs is within the boundary of the application

Introduction to monolithic architecture



moved the monolith's data storage mechanisms to a separate machine private to the network

Introduction to monolithic architecture

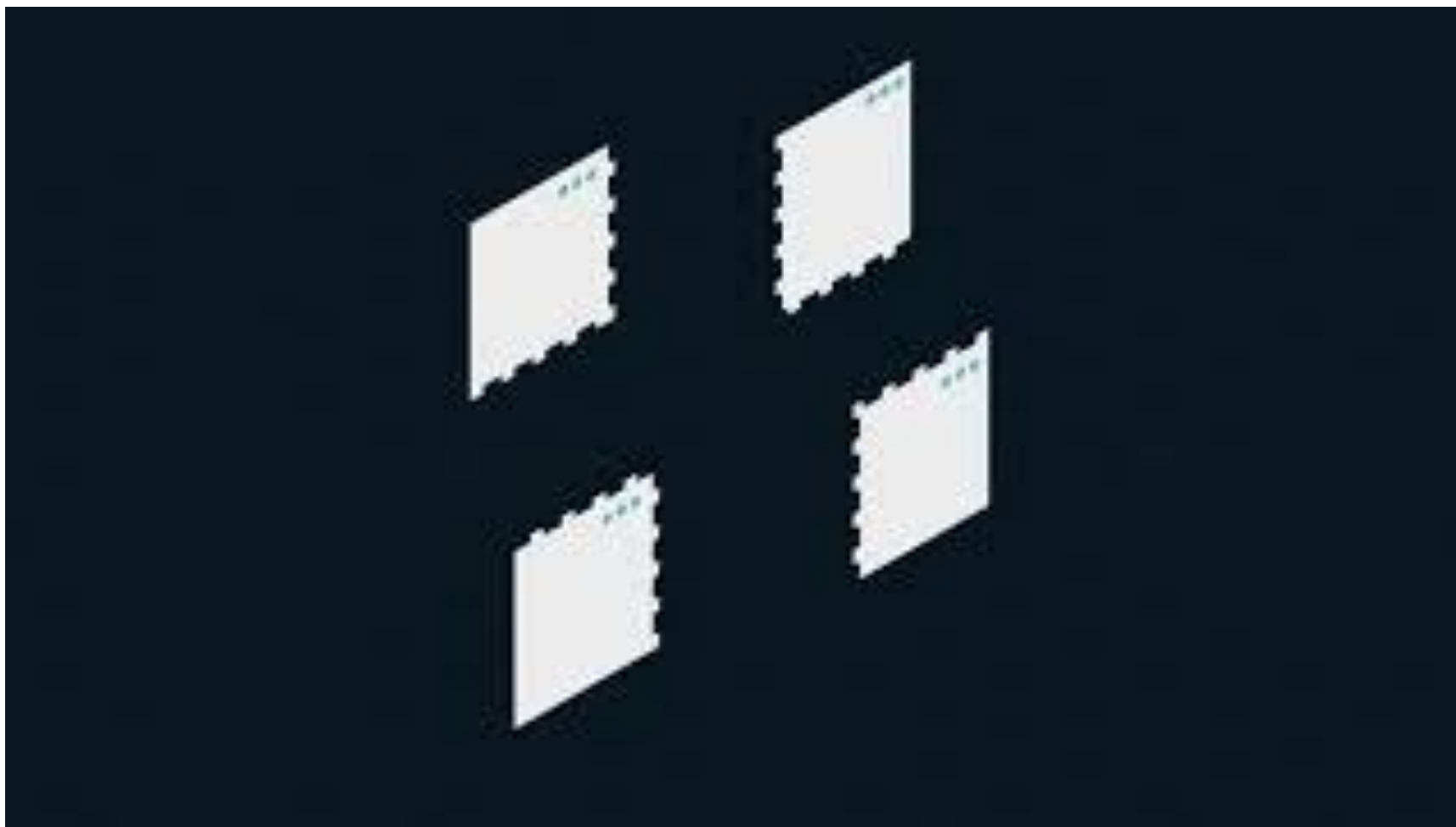


multiple instance design in which many computers hosting identical business logic were put behind a load balancer

Introduction to miniservice architecture

- ▶ Lessly talked in the market (only appear in dzone)
- ▶ Aim to describe SOA but not yet microservices
- ▶ Services that
 - Multiple applications share the same database
 - Services communicate with each other through REST APIs and do not embrace event-based architecture for asynchronous communication
 - Share infrastructure for deployment

What are microservices (43 sec)



What are microservices (0'47 – 1'24)



What is REST APIs?

- ▶ also known as RESTful API
- ▶ an application programming interface (API or web API) that conforms to the constraints of REST architectural style
- ▶ was created by computer scientist Roy Fielding

What is REST APIs?

REST is

- ▶ set of architectural constraints
- ▶ not a protocol or a standard
- ▶ transfers a representation of the state of the resource to the requester or endpoint
- ▶ is delivered in one of several formats via HTTP

What is REST APIs?

Formats:

- ▶ JSON (Javascript Object Notation)
- ▶ HTML
- ▶ XLT
- ▶ or plain text.
- ▶ JSON is the most generally popular file format to use because, despite its name, it's language-agnostic, as well as readable by both humans and machines.

What is REST APIs?

keep in mind:

- ▶ Headers and parameters are also important
- ▶ Contains
 - Request's metadata
 - Authorization
 - Uniform resource identifier (URI)
 - Caching
 - Cookies
 - ...

What is REST APIs?

RESTful has to conform

- ▶ Client-server architecture
- ▶ Stateless client-server communication
- ▶ Cacheable data
- ▶ A uniform interface between components
- ▶ A layered system that organizes each type of server
- ▶ Code-on-demand (optional)

What is REST APIs?

vs SOAP

- ▶ easier to use
- ▶ no specific requirements like XML messaging
- ▶ no built-in security
- ▶ no transaction compliance
- ▶ faster
- ▶ lighter

What is REST APIs?

vs GraphQL

- ▶ GraphQL presents a learning curve for developers familiar with REST APIs.
- ▶ GraphQL shifts much of the work of a data query to the server side, which adds complexity for server developers.
- ▶ GraphQL might require different API management strategies than REST APIs
- ▶ Caching is more complex than with REST.
- ▶ API maintainers have the additional task of writing maintainable GraphQL schema

Microservices vs. Monolithic

	Microservices	Monolithic
Deployment	Requires distinct resources, making orchestrating the deployment complicated	Simple and fast deployment of the entire system
Scalability	Each element can be scaled independently without downtime	It is hard to maintain and handle new changes, the whole system needs to be redeployed
Agility	Integrate with new technologies to solve business purposes	Not flexible and impossible to adopt new tech, languages, or frameworks

Microservices vs. Monolithic

	Microservices	Monolithic
Resiliency	A failure in one microservice does not affect other services	One bug or issue can affect the whole system
Testing	Independent components need to be tested individually	End-to-end testing
Security	Interprocess communication requires API gateways raising security issues	Communication within a single unit makes data processing secure
Development	A team of developers can work independently on each component	Impossible to distribute the team's efforts due to the huge indivisible database

Microservices vs. Miniservices

	Microservices	Miniservices
Scope	Has a Single Responsibility.	Single Domain
Deployment	Is loosely coupled.	Share infrastructure for deployment.
Communication	Synchronous through REST APIs	through Publish-Subscribe Pattern

Benefits of using microservices

- ▶ More flexibility
 - Deployment
 - Scalability
 - Technical
- ▶ Faster release cycle

Disadvantages of using microservices

- ▶ Operational complexity
- ▶ Resource requirements
- ▶ Need to orchestration framework
- ▶ Fit for application run at web scale with large number of users

Overview of Container Technology

Introduction to virtual machine

- ▶ Virtualization functionality, which enables a machine to host multiple virtual machines (VMs), also referred to as guests.
- ▶ VMs use the host's physical hardware and computing resources to run a separate, virtualized operating system (guest OS) as a user-space process on the host's operating system.
- ▶ In other words, virtualization makes it possible to have operating systems within operating systems.

Advantages of Virtual Machine

- ▶ Flexible and fine-grained allocation of resources
- ▶ Software-controlled configurations
- ▶ Separation from the host
- ▶ Space and cost efficiency
- ▶ Software compatibility

Problems of using virtual machine

- ▶ Bulky, VMs are typically measured by the gigabyte
- ▶ Application dependencies managements
- ▶ Large OS footprints

Introduction to container

- ▶ Containers are technologies that allow the packaging and isolation of applications with their entire runtime environment—all of the files necessary to run. This makes it easy to move the contained application between environments (dev, test, production, etc.) while retaining full functionality.

Introduction to container (42 sec)



Docker container and Dockerfile

Basic Vocabulary

- ▶ Container Image
 - is a file which is pulled down from a Registry Server
- ▶ Container Image Format
 - Historically, each Container Engine had its container images format. LXD, RKT, and Docker all had their own image formats
 - Today, almost all major tools and engines have moved to a format defined by the Open Container Initiative (OCI)

Docker container and Dockerfile

Basic Vocabulary

- ▶ Container Engine
 - a piece of software that accepts user requests
 - e.g.: docker, RKT, CRI-O, and LXD
- ▶ Container
 - container is the runtime instantiation of a Container Image
- ▶ Registry Server
 - a fancy file server that is used to store docker repositories
- ▶ Container Runtime

Docker container and Dockerfile

Basic Vocabulary

- ▶ Container Runtime
 - a lower level component typically used in a Container Engine
 - The Open Containers Initiative (OCI) Runtime Standard reference implementation is runc
 - others OCI compliant runtimes, such as crun, railcar, and katacontainers. Docker, CRI-O, and many other Container Engines rely on runc.

Docker container and Dockerfile

Dockerfile / Containerfile

- ▶ You might have seen or used Dockerfiles instead of Containerfiles. These files are largely the same and mainly differ in historical context.
- ▶ A Containerfile lists a set of instructions to construct a container image.
- ▶ Once built, the image is used to create any number of containers.
- ▶ Each instruction causes some change that is captured in a resulting image layer
- ▶ These layers are stacked together to form the resulting container image

Docker container and Dockerfile

Dockerfile / Containerfile Instructions

- ▶ FROM
 - Takes the name of the base image as an argument.
- ▶ WORKDIR
 - Sets the current working directory within the container. Later instructions run within this directory.
- ▶ COPY
 - Copies files from the build host into the file system of the resulting container image.

Docker container and Dockerfile

Dockerfile / Containerfile Instructions

- ▶ ADD
 - Copies files or folders from a local or remote source and adds them to the container's file system.
- ▶ RUN
 - Runs a command in the container and commits the resulting state of the container to a new layer within the image.
- ▶ ENTRYPOINT
 - Sets the executable to run when the container is started.

Docker container and Dockerfile

Dockerfile / Containerfile Instructions

- ▶ CMD
 - Runs a command when the container is started.
- ▶ USER
 - Changes the active user within the container.
- ▶ LABEL
 - Adds a key-value pair to the metadata of the image for organization and image selection.

Docker container and Dockerfile

Dockerfile / Containerfile Instructions

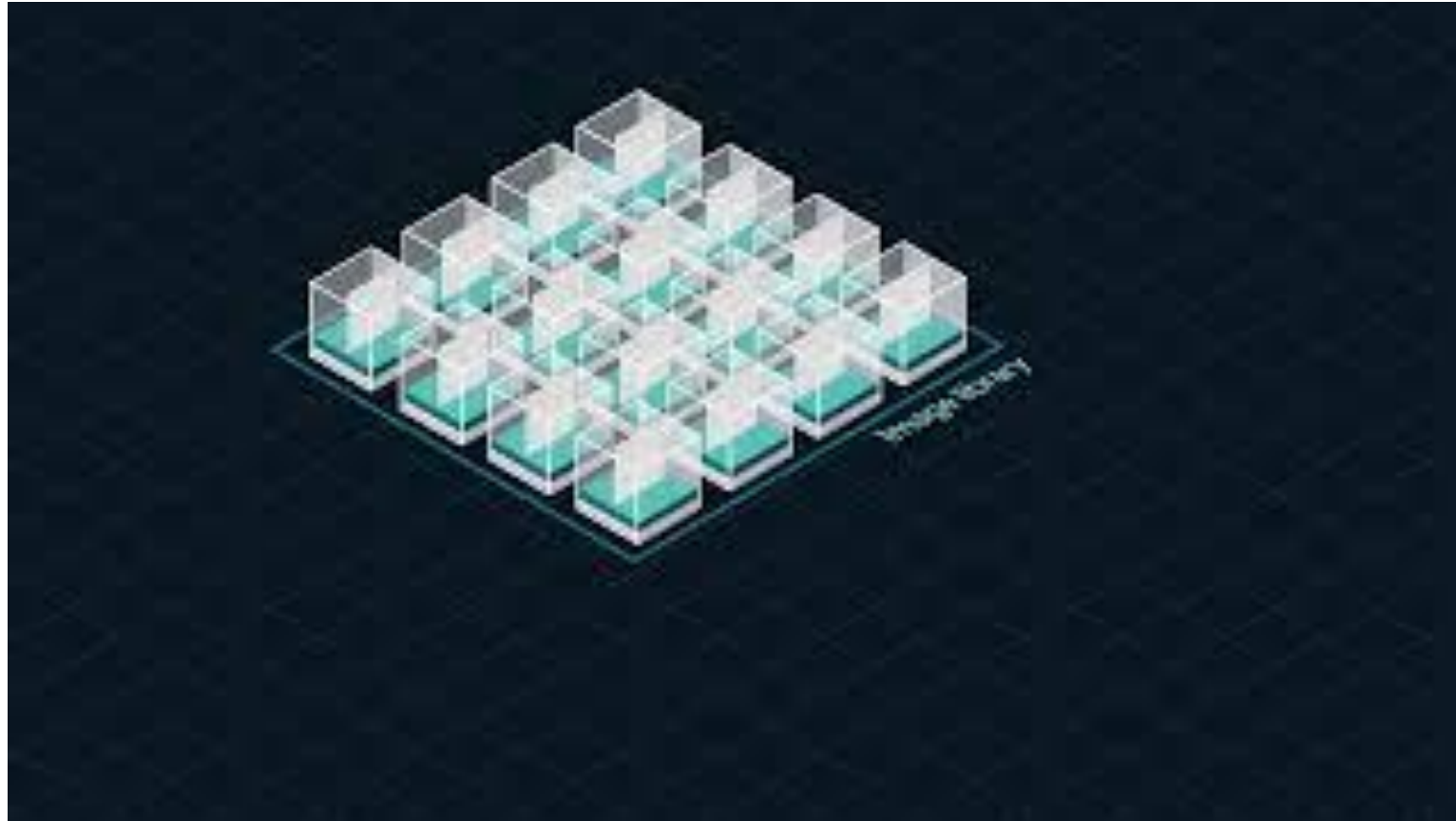
- ▶ EXPOSE
 - Adds a port to the metadata for the image indicating that an application within the container will bind to this port.
- ▶ ENV
 - Defines environment variables that are available in the container.
- ▶ ARG
 - Defines build-time variables, typically to make a customizable container build.

Docker container and Dockerfile

Dockerfile / Containerfile Instructions

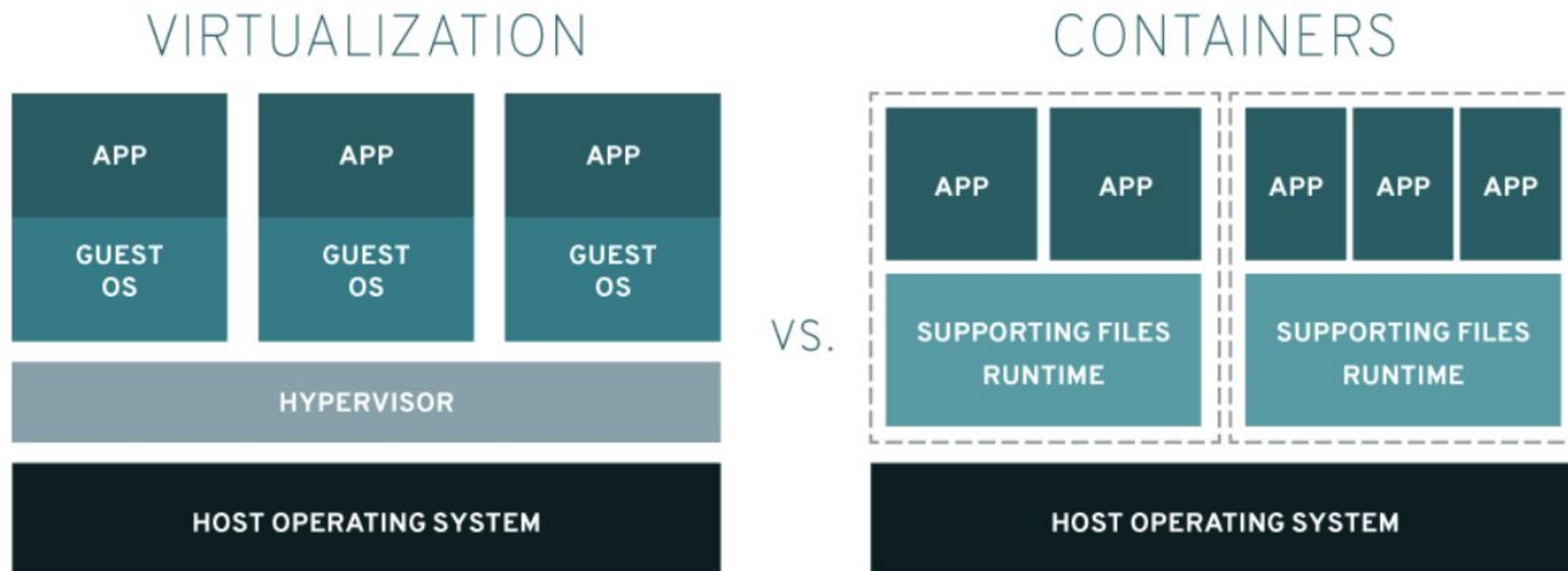
- ▶ VOLUME
 - Defines where to store data outside of the container.

Container vs. Virtual Machine (36 sec)



Container vs. Virtual Machine

How do they work?



Container vs. Virtual Machine


VM	Container
Software called a hypervisor separates resources from their physical machines so they can be partitioned and dedicated to VMs. When a user issues a VM instruction that requires additional resources from the physical environment, the hypervisor relays the request to the physical system and caches the changes. VMs look and act like physical servers, which can multiply the drawbacks of application dependencies and large OS footprints—a footprint that's mostly not needed to run a single app or microservice.	Containers hold a microservice or app and everything it needs to run. Everything within a container is preserved on something called an image—a code-based file that includes all libraries and dependencies. These files can be thought of as a Linux distribution installation because the image comes with RPM packages, and configuration files. Because containers are so small, there are usually hundreds of them loosely coupled together—which is why container orchestration platforms (like Red Hat OpenShift and Kubernetes) are used to provision and manage them.

Container vs. Virtual Machine

	Virtual machines	Containers
Machine-level functionality	Hypervisor	Container engine
Management	VM management interface	Container engine or orchestration software
Virtualization level	Fully virtualized environment	Only relevant parts
Size	Measured in gigabytes	Measured in megabytes
Portability	Generally only same hypervisor	Any OCI-compliant engine

Pros and cons of using container

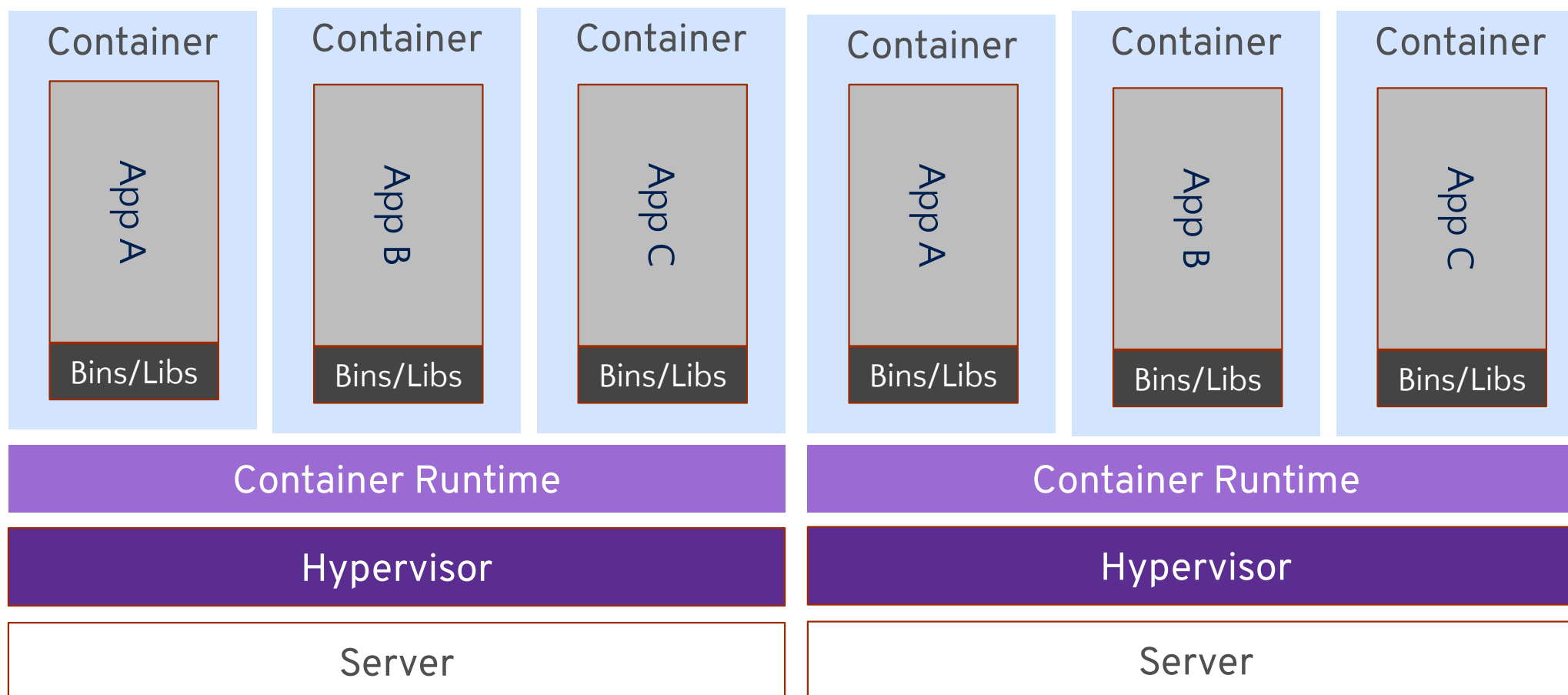
Pros		Cons	
Portability		Security	
Scalability		Complexity	
Isolation		Storage	
Consistency		Networking	
Resource efficiency		Compatibility	



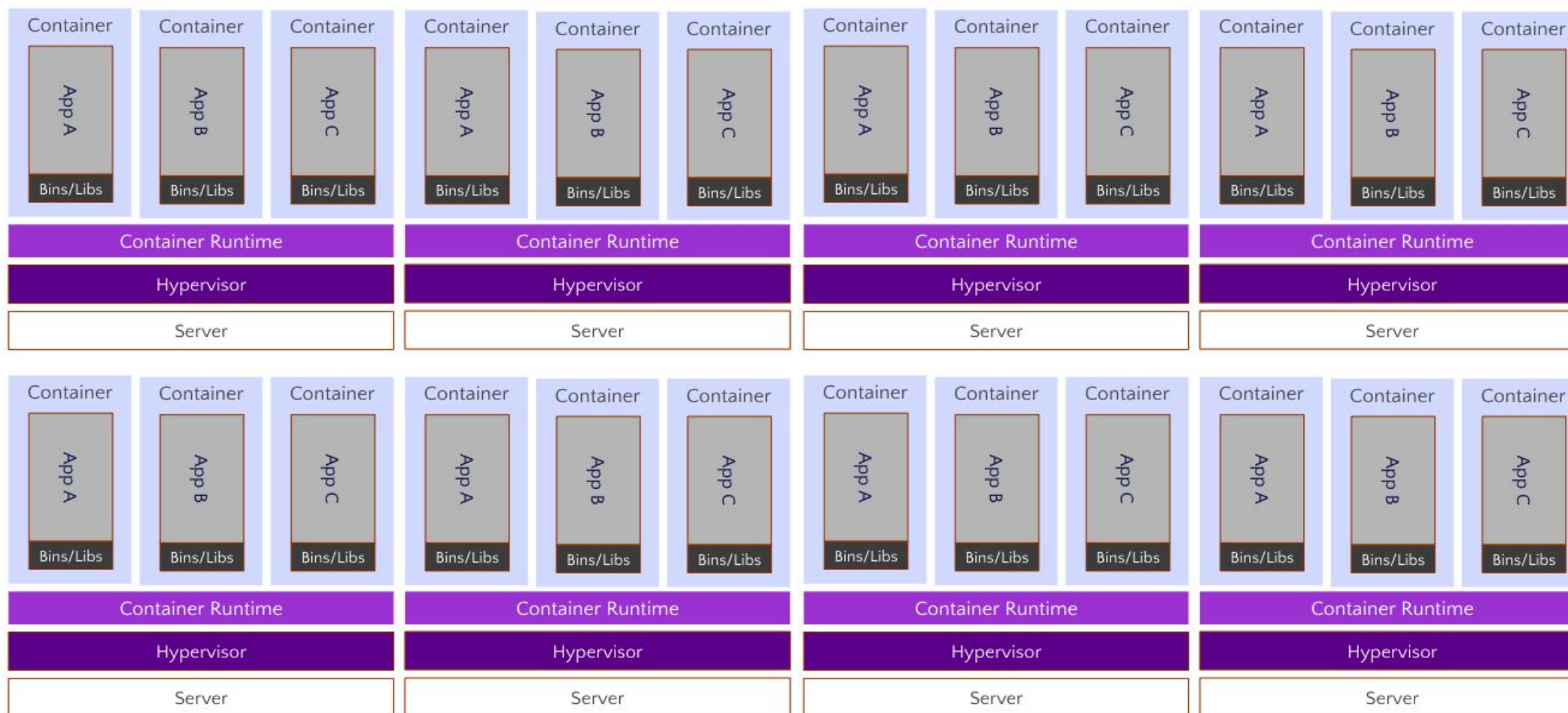
Demonstration:
Build and run
container
images and push
it to registry

Overview of Kubernetes

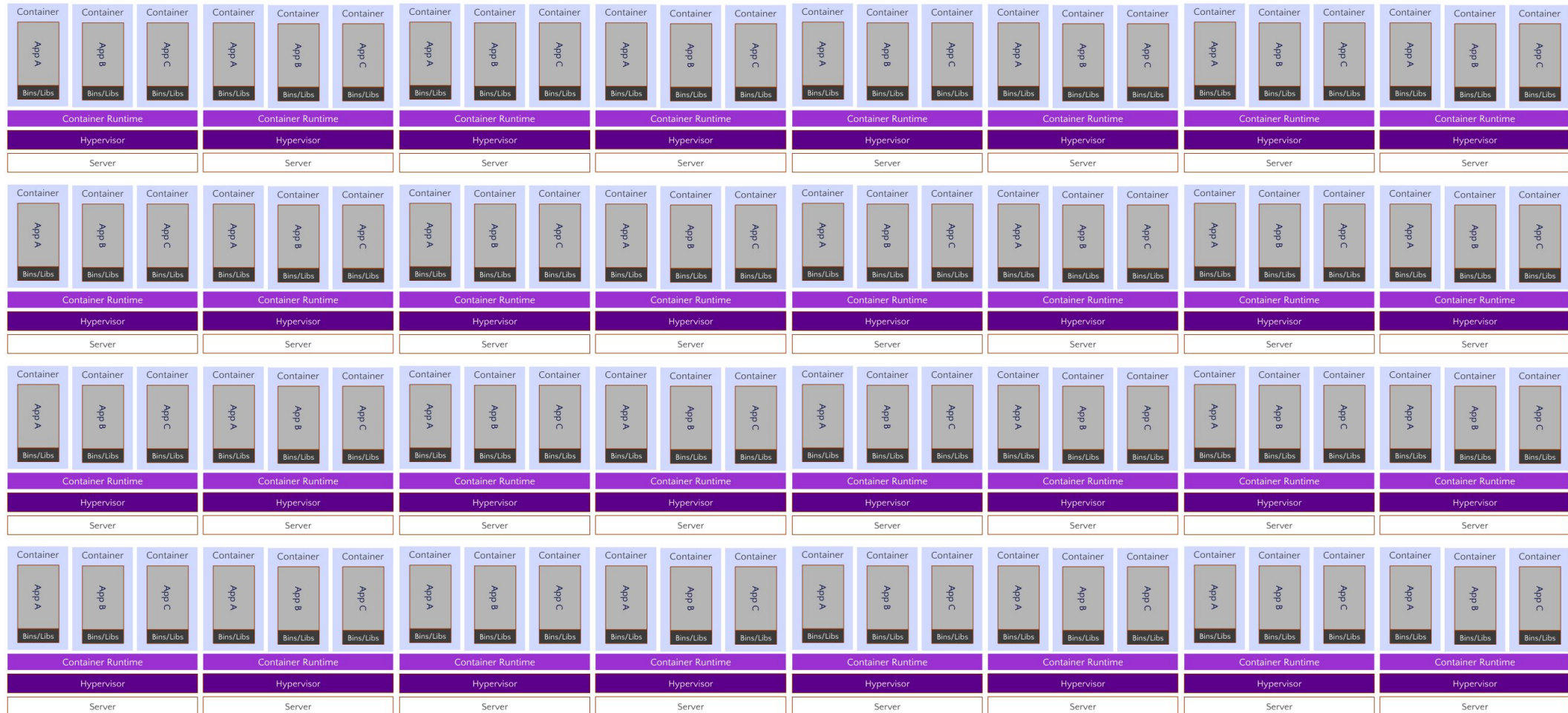
Hosting a few containers are easy....



But when we want more... and more!



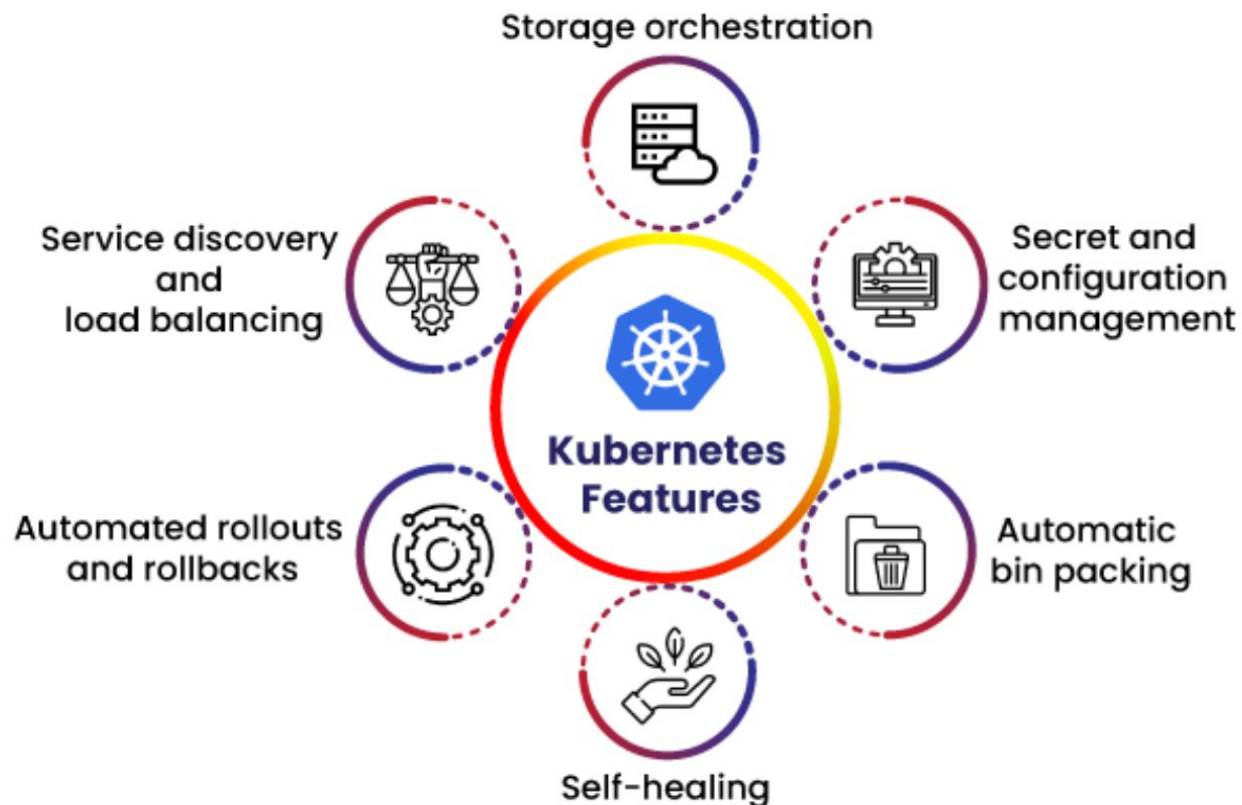
And more... Too many! What a mess!



The needs of a container orchestration engine



The rise of Kubernetes



A way to realize the benefits of containers AT SCALE

Operations

- Promote the use of infrastructure-as-code
- Stable, proven and battle-tested
- Broad ecosystem to integrate

Development

- The foundation of microservices design
- Abstract infra resources, focus on application logic

A brief history of containers and Kubernetes (1)

The Rise of Docker

Docker Inc. launches Docker runtime to make Linux kernel resource isolation features easy to access and usable.

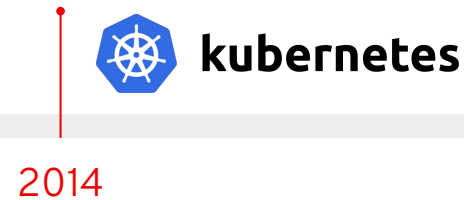


CoreOS

A specialize OS for container was born, by CoreOS Inc.

Kubernetes announced

Google made their internal container system, open-sourced.



2015

Kubernetes 1.0 released

Google gave Kubernetes to Cloud Native Computing Foundation (CNCF) for development.

A brief history of containers and Kubernetes (2)

Red Hat strategically adopted Kubernetes

Red Hat decided to revamp their next-gen application platform, OpenShift 3.0, to Kubernetes v1.0.

**2015**

Open Container Initiative (OCI) formed

As there were non-Docker runtimes being launched, an industry wide standard for container format and runtime was formed.



Docker was defeated by Kubernetes

Docker Inc. adopted Kubernetes in their Docker EE.

2017

**AWS launched EKS
Microsoft Azure launched AKS**

Version 1.0; 3 years after the community and Red Hat.

2018

A brief history of containers and Kubernetes (3)

Red Hat acquired CoreOS Inc.

Red Hat decided to integrate with CoreOS, an immutable and optimized OS for containers.

2019

2018



Red hat launched OpenShift 4.0

Combining the CoreOS features, and with 4+ years of running Kubernetes experience, OpenShift 4.0 has evolved.

Google launched Anthos

Only for GCP and VMWare on-premises.



Anthos

Docker was almost out of the game

Docker sold its flagship container solution, Docker EE to Mirantis.

2020

vmware

VMWare just joined the game to productize Kubernetes

VMware vSphere 7 with proprietary Kubernetes; 5 years after Kubernetes was released and Red Hat adoption of Kubernetes

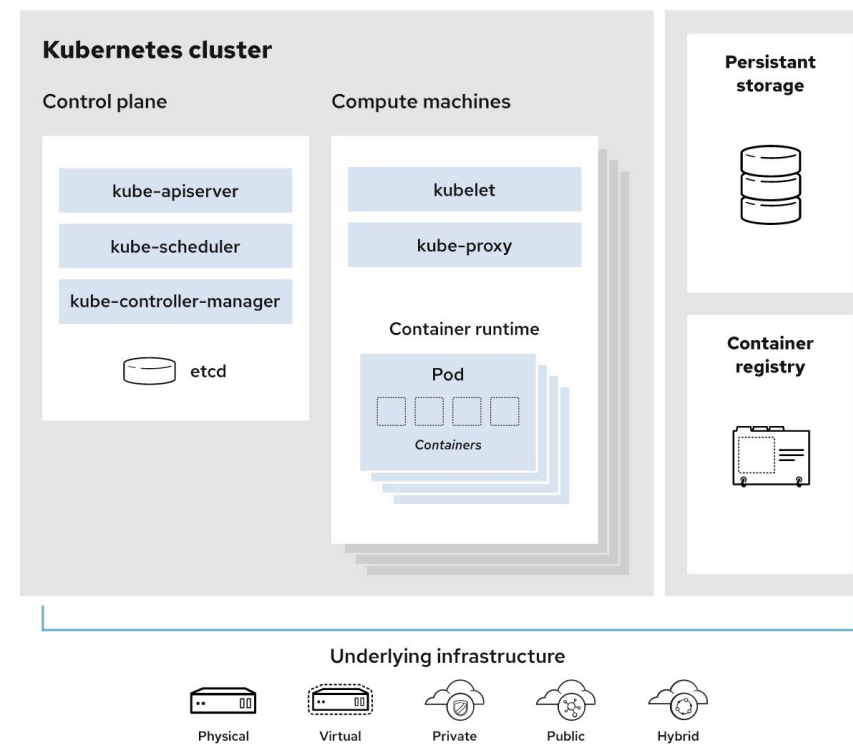
Introduction to Kubernetes architecture

Kubernetes design principles

- ▶ Secure. It should follow the latest security best-practices
- ▶ Easy to use. It should be operable using a few simple commands.
- ▶ Extendable. It shouldn't favor one provider and should be customizable from a configuration file.

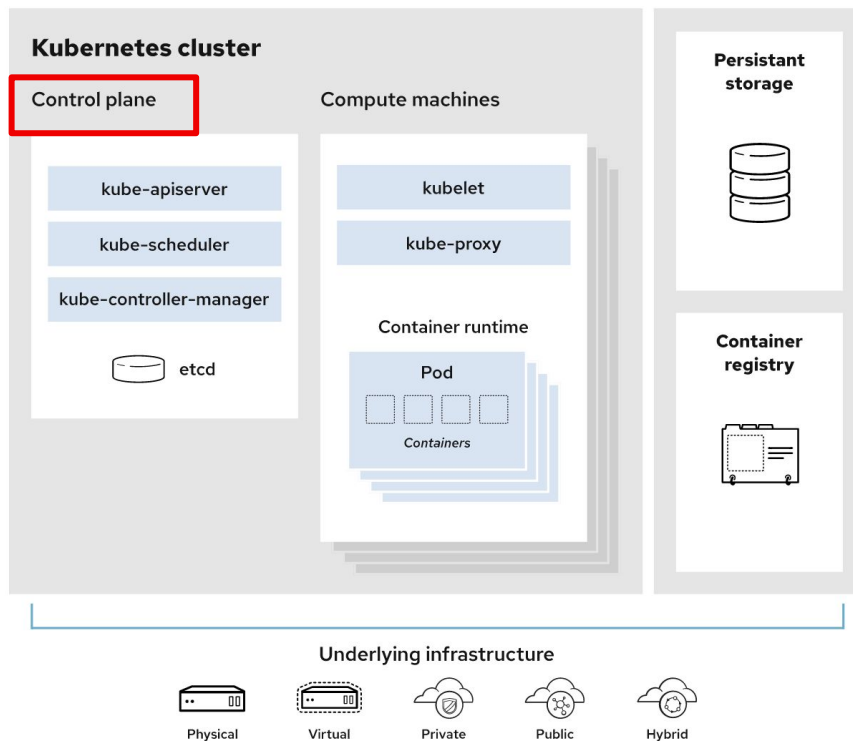
Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?



Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

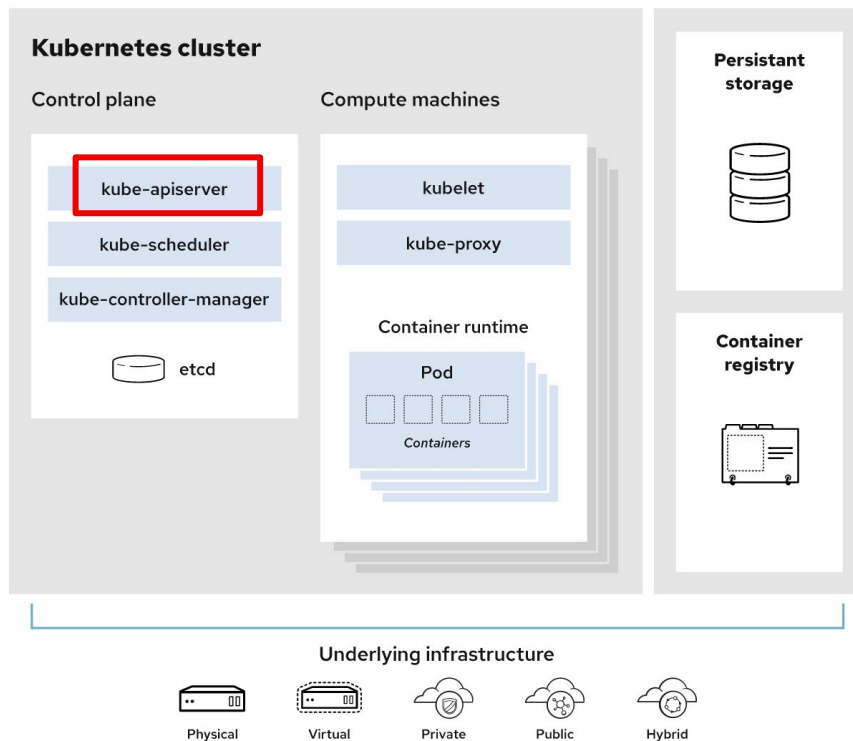


► Control plane

- The control plane. Here we find the Kubernetes components that control the cluster, along with data about the cluster's state and configuration.
- The control plane is in constant contact with your compute machines

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

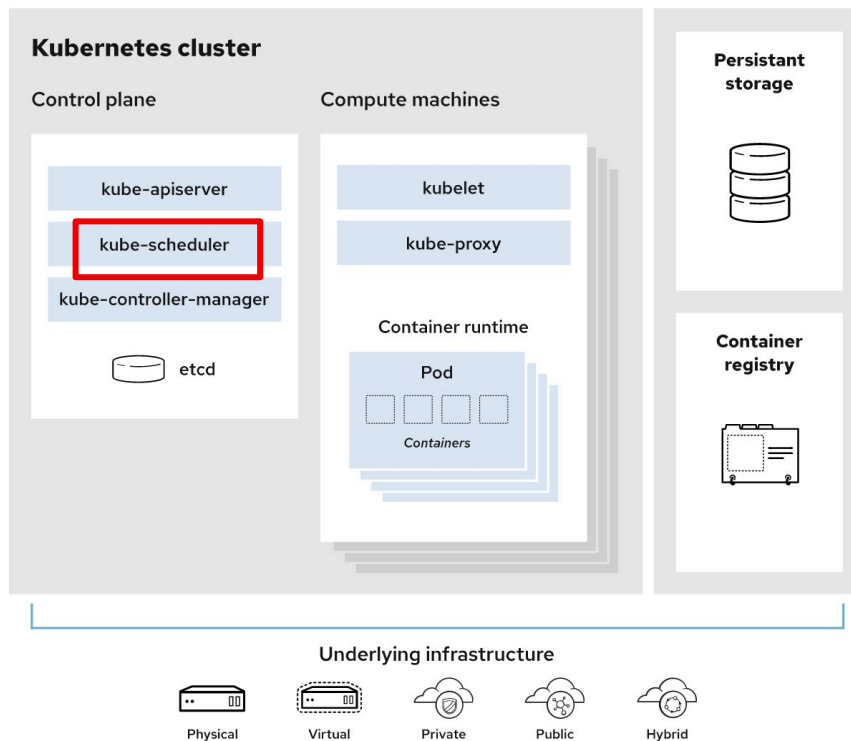


► kube-apiserver

- The Kubernetes API is the front end of the Kubernetes control plane, handling internal and external requests.
- The API server determines if a request is valid and, if it is, processes it
- You can access the API through REST calls, through the `kubectl` command-line interface

Introduction to Kubernetes architecture

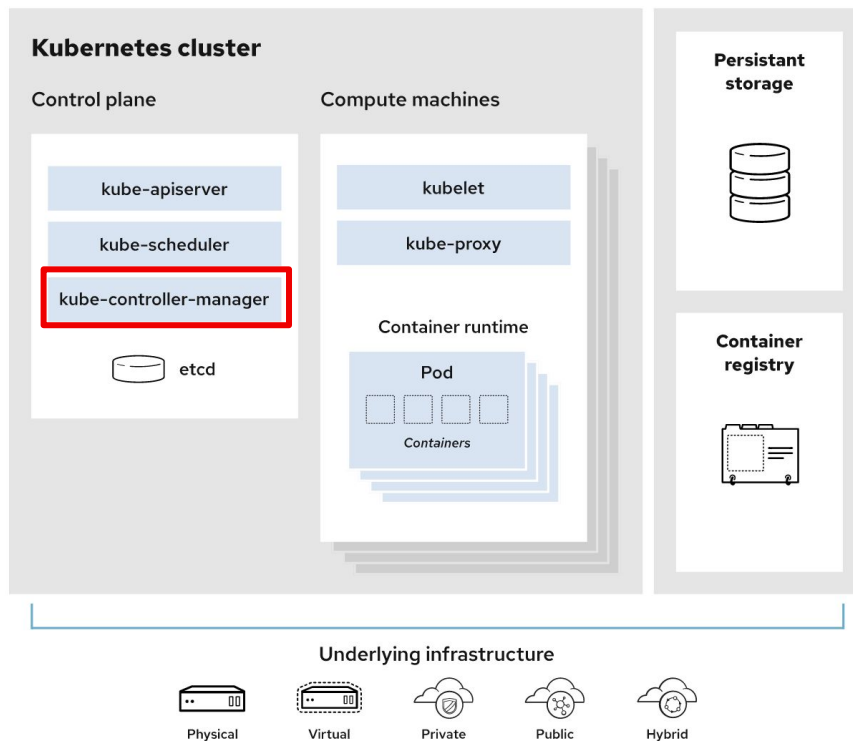
What are the components of a Kubernetes cluster?



- ▶ `kube-scheduler`
 - The scheduler considers the resource needs of a pod, such as CPU or memory, along with the health of the cluster.
 - Then it schedules the pod to an appropriate compute node

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

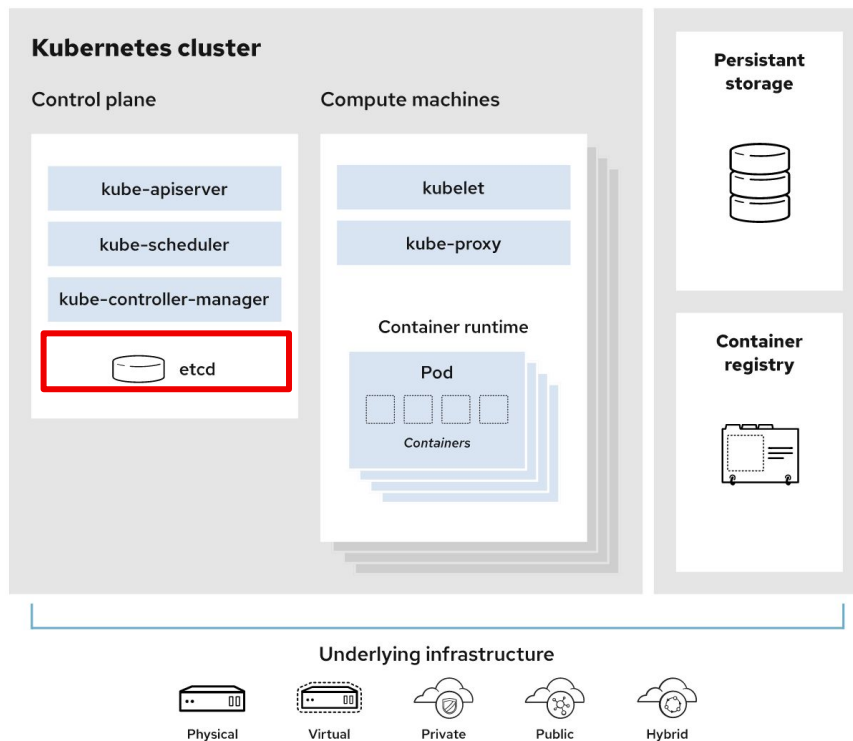


► kube-controller-manager

- Controllers take care of actually running the cluster, and the Kubernetes controller-manager contains several controller functions in one.
- One controller consults the scheduler and makes sure the correct number of pods is running.

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

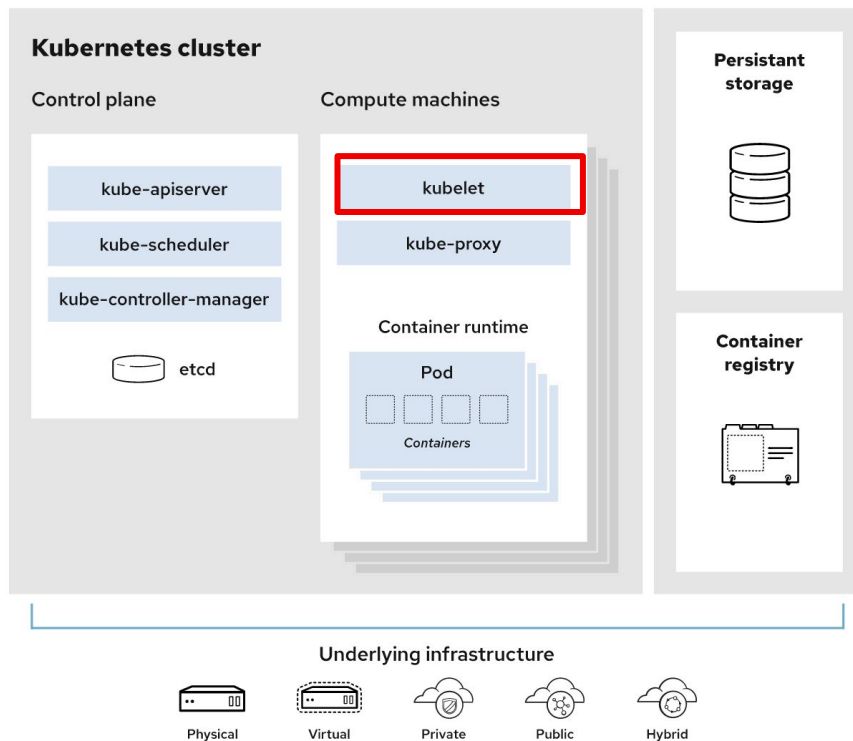


► etcd

- Configuration data and information about the state of the cluster lives in etcd, a key-value store database.
- Fault-tolerant and distributed, etcd is designed to be the ultimate source of truth about your cluster.

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

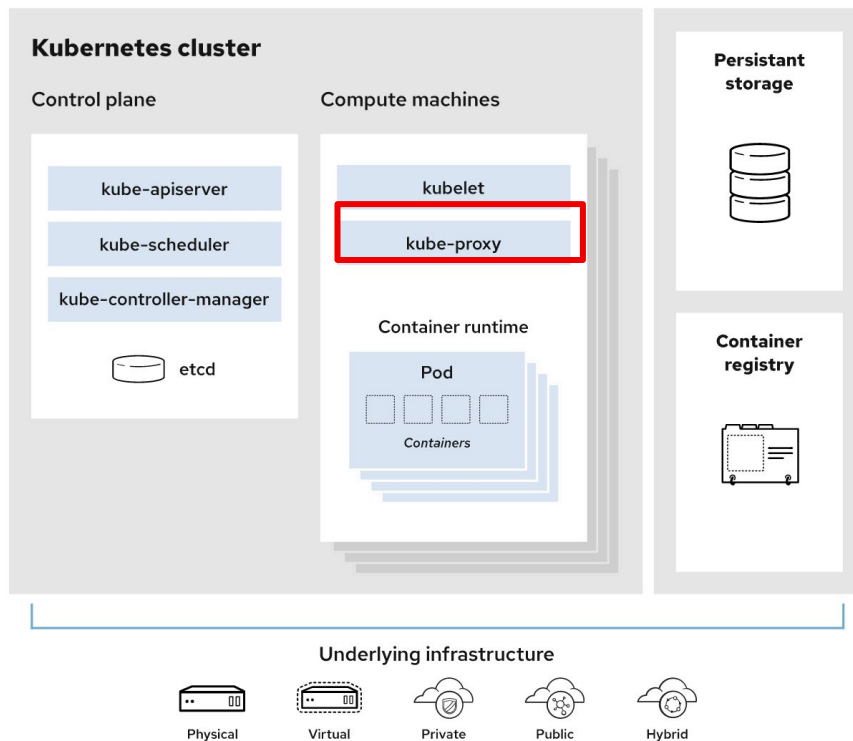


► kubelet

- Each compute node contains a kubelet, a tiny application that communicates with the control plane.
- The kubelet makes sure containers are running in a pod. When the control plane needs something to happen in a node, the kubelet executes the action.

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

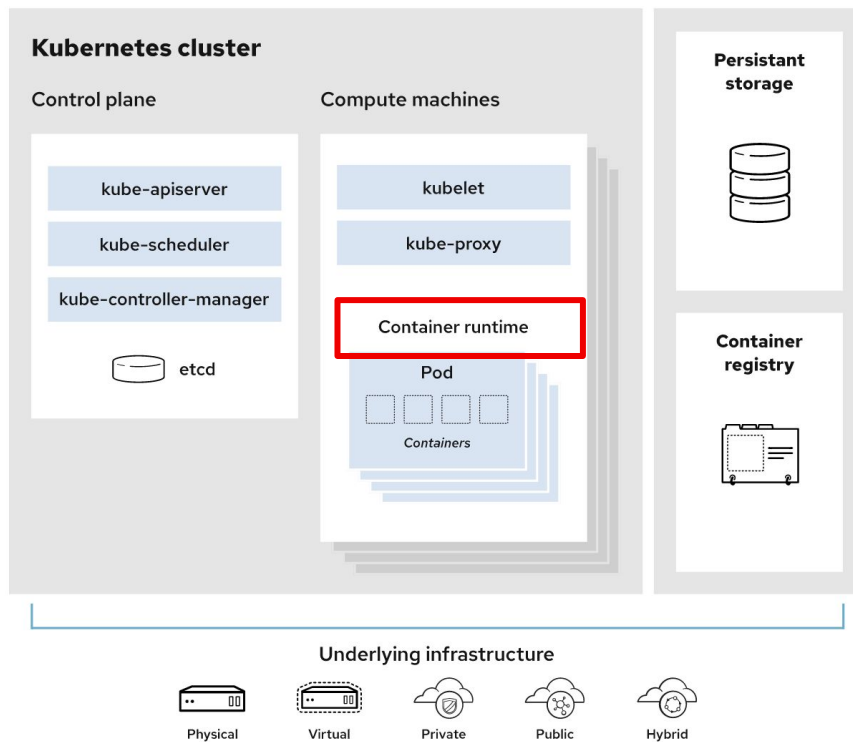


► kube-proxy

- Each compute node also contains kube-proxy, a network proxy for facilitating Kubernetes networking services.
- The kube-proxy handles network communications inside or outside of your cluster—relying either on your operating system’s packet filtering layer, or forwarding the traffic itself.

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

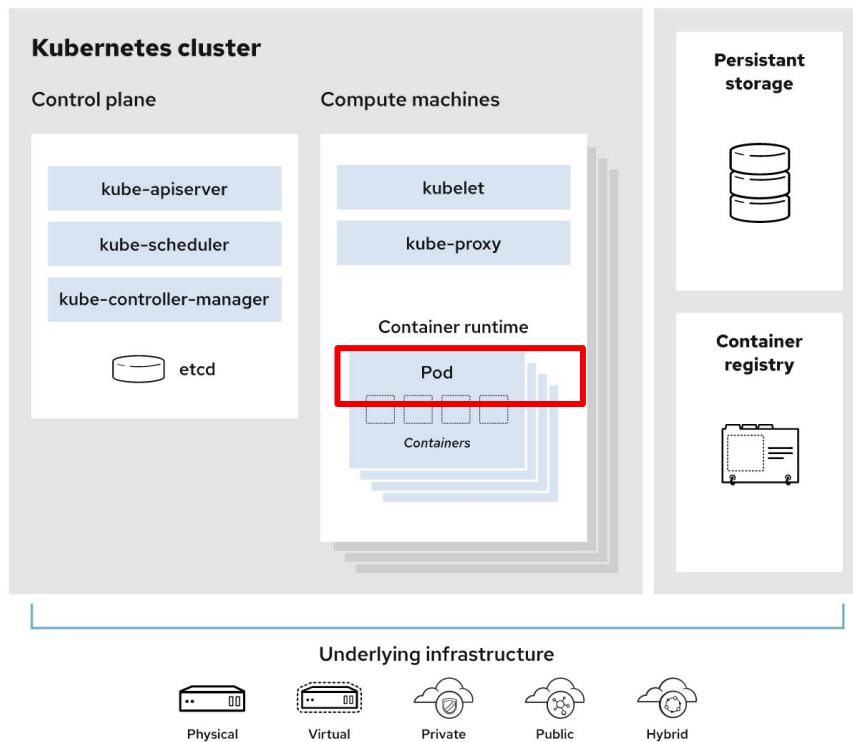


► Container runtime engine

- To run the containers, each compute node has a container runtime engine.
- Docker is one example, but Kubernetes supports other Open Container Initiative-compliant runtimes as well, such as rkt and CRI-O

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

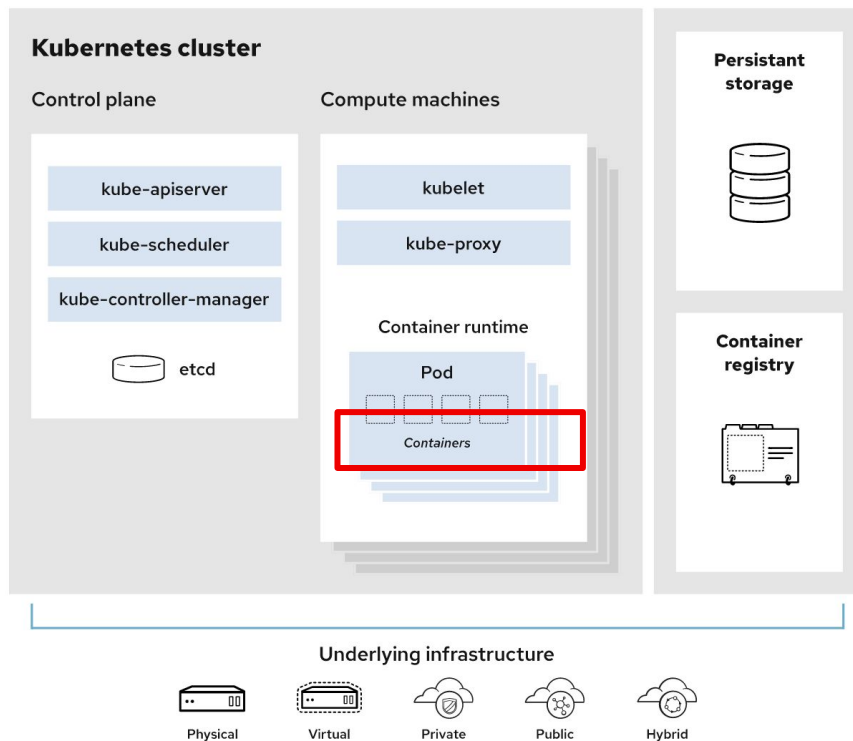


► Pods

- A pod is the smallest and simplest unit in the Kubernetes object model.
- It represents a single instance of an application.

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

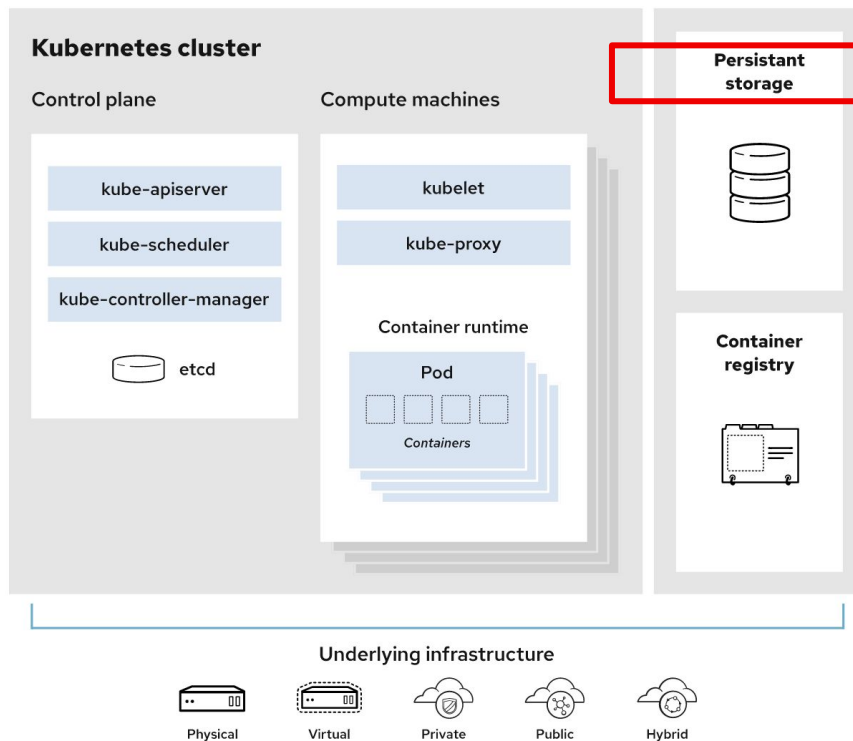


► Containers

- Each pod is made up of a container or a series of tightly coupled containers, along with options that govern how the containers are run.
- Pods can be connected to persistent storage in order to run stateful applications.

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?

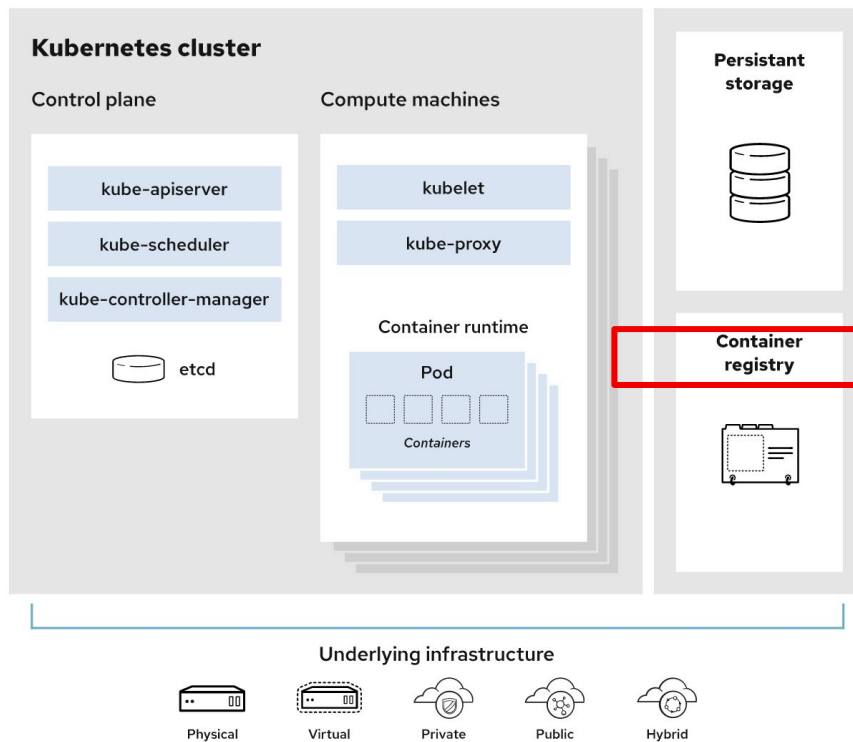


► Persistent storage

- Beyond just managing the containers that run an application, Kubernetes can also manage the application data attached to a cluster.
- Kubernetes allows users to request storage resources without having to know the details of the underlying storage infrastructure.

Introduction to Kubernetes architecture

What are the components of a Kubernetes cluster?



► Container registry

- The container images that Kubernetes relies on are stored in a container registry. This can be a registry you configure, or a third party registry.

Kubernetes objects and basic kubectl commands

Basic Commands

- ▶ `kubectl [command] [TYPE] [NAME] [flags]`
 - `command`
 - Specifies the operation that you want to perform on one or more resources, for example create, get, describe, delete.
 - `TYPE`
 - Specifies the resource type. Resource types are case-insensitive and you can specify the singular, plural, or abbreviated forms.

Kubernetes objects and basic kubectl commands

Basic Commands

- ▶ `kubectl [command] [TYPE] [NAME] [flags]`
 - NAME
 - Specifies the name of the resource.
 - To group resources if they are all the same type: TYPE1 name1 name2 name<#>
 - To specify multiple resource types individually: TYPE1/name1 TYPE1/name2 TYPE2/name3
 - To specify resources with one or more files: -f file1 -f file2 -f file<#>

Kubernetes objects and basic kubectl commands

Basic Commands

get	Display one or many resources.
describe	how details of a specific resource or group of resources.
create	Create a resource from a file or from stdin
apply	Apply a configuration to a resource by file name or stdin. The resource name must be specified. This resource will be created if it doesn't exist yet.
edit	Edit a resource from the default editor.
delete	Delete resources by file names, stdin, resources and names, or by resources and label selector.

Kubernetes objects and basic kubectl commands

Basic Kubernetes Objects

pod	Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.
deployment	A Deployment provides declarative updates for Pods and ReplicaSets.
service	In Kubernetes, a Service is a method for exposing a network application that is running as one or more Pods in your cluster
secret	A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image.
configmap	A ConfigMap is an API object used to store non-confidential data in key-value pairs.
pv	A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.
pvc	A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources.

Kubernetes objects and basic kubectl commands

Examples

- **kubectl get namespace**
 - Obtain all namespace as a list
- **kubectl describe pod/mypod -n myns**
 - Obtain brief information about the corresponding pod in namespace "myns"
- **kubectl create -f myresource.yaml**
 - Create the resources defined in "myresource.yaml"
- **kubectl apply -f myresource.yaml**
 - Create or modify the resources defined in "myresource.yaml"
- **kubectl edit pvc/mypvc**
 - Edit the pvc with name mypvc in current namespace with default editor (notepad in Windows, vi in Linux)
- **kubectl delete pod --all**
 - Delete all the pod in current namespace

Benefits of using Kubernetes



Public cloud agility and simplicity on-premises to reduce friction between developers and IT operations



Cost efficiency by eliminating the need for a separate hypervisor layer to run VMs



Developer flexibility to deploy containers, serverless applications, and VMs from Kubernetes, scaling both applications and infrastructure



Hybrid cloud extensibility with Kubernetes as the common layer across on-premises and public clouds

Demonstration: Creating a Kubernetes deployment using YAML file

Demonstration: Basic functions of Kubernetes

Overview of Service Mesh

Introduction to service mesh



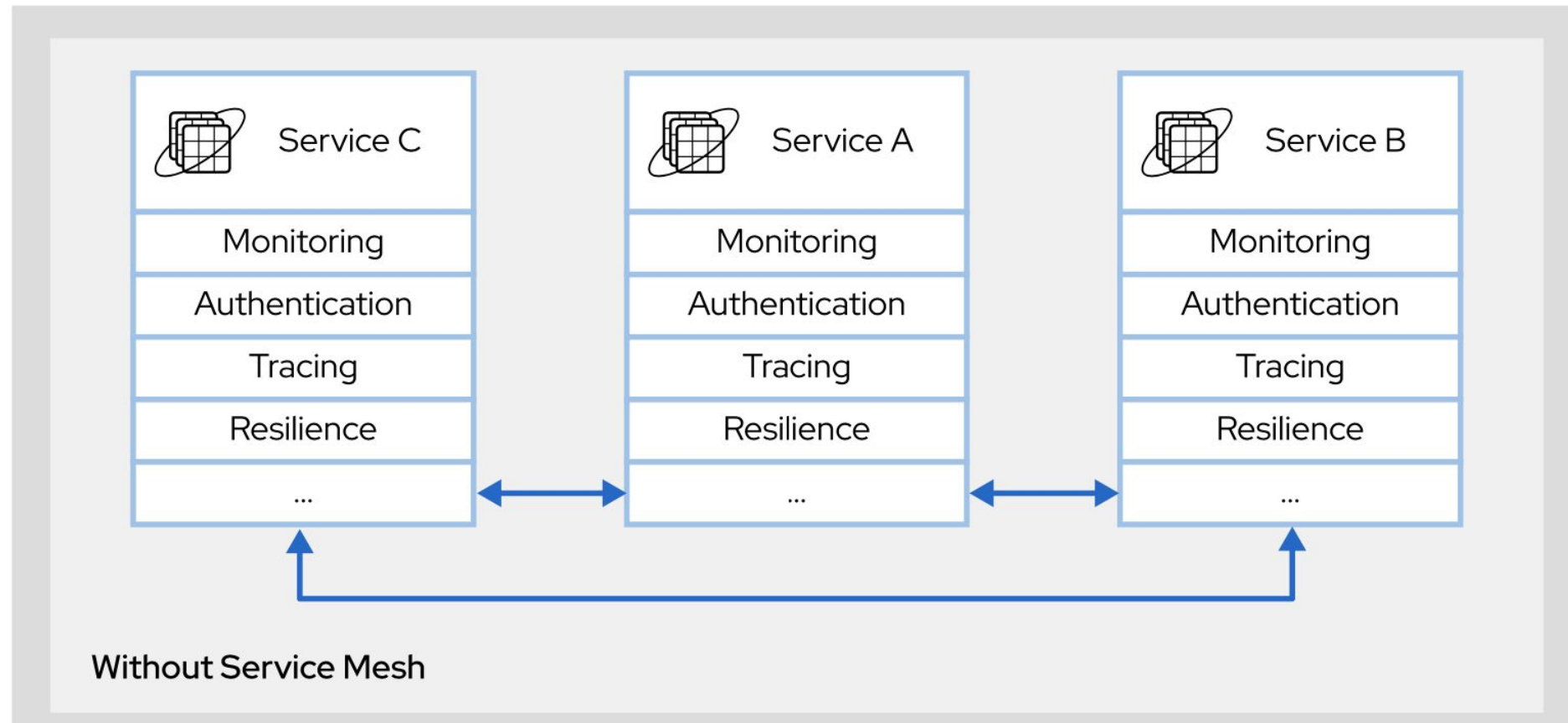
Introduction to service mesh

- ▶ A service mesh is a flexible layer of infrastructure that enables organized, observable communication between services (typically containers and/or microservices).
- ▶ In order to improve and safeguard the system, a service mesh also keeps track of and controls network traffic between services by rerouting it and allowing or denying access as necessary.
- ▶ Improved observability, security, and granular control over network traffic are all advantages of adopting a service mesh.
- ▶ Additionally, it can make it simpler to add or delete services from an application without having an adverse effect on the rest of the system, as well as to manage applications.

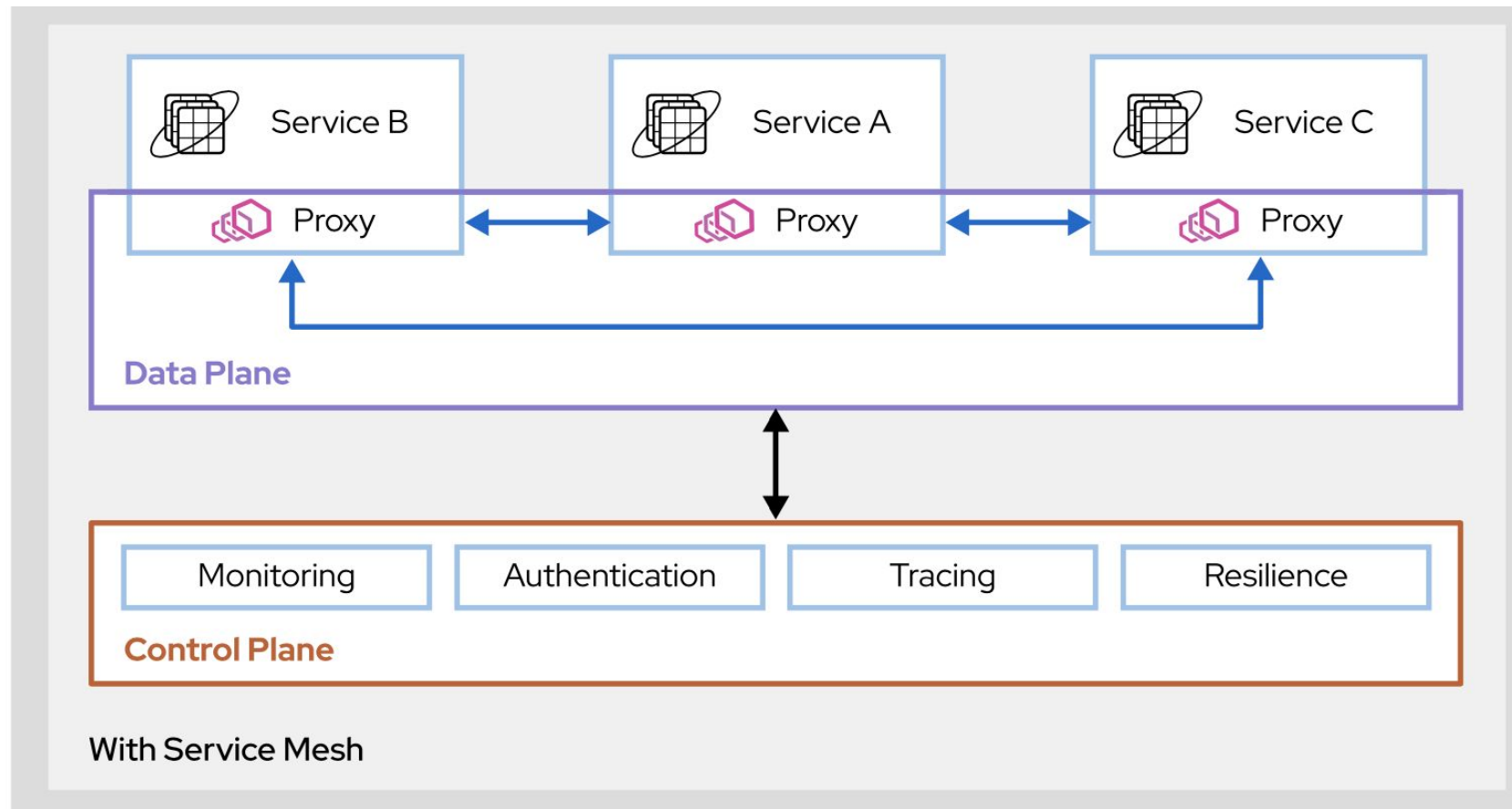
How service mesh works in Kubernetes

- ▶ The Istio service mesh
 - Istio extends Kubernetes to establish a programmable, application-aware network using the powerful Envoy service proxy. Working with both Kubernetes and traditional workloads, Istio brings standard, universal traffic management, telemetry, and security to complex deployments.

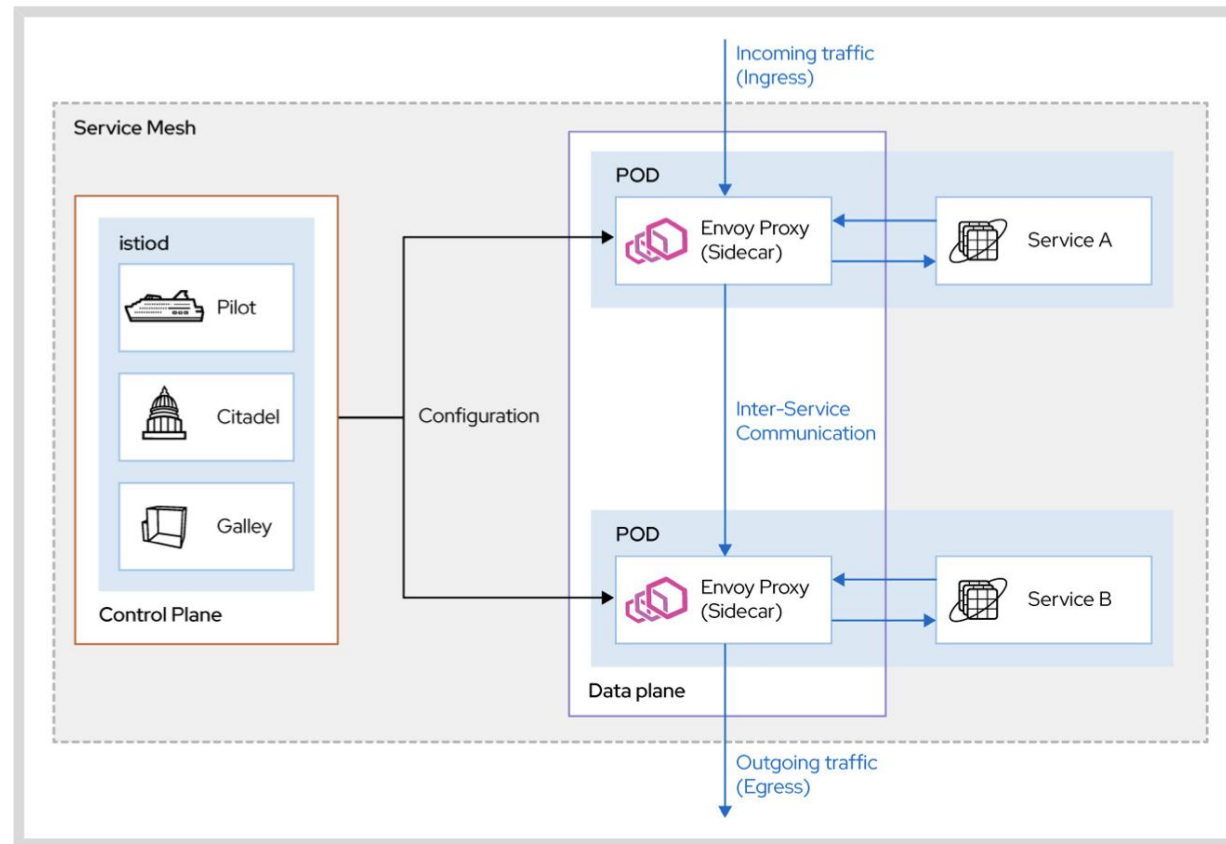
How service mesh works in Kubernetes



How service mesh works in Kubernetes



How service mesh works in Kubernetes



How service mesh works in Kubernetes

- ▶ The **istio-agent** component, known also as the Istio Pilot agent, is part of every Envoy proxy. It helps to bootstrap the Envoy proxy container during startup. Additionally, it maintains other functions, such as:
 - Automating certificate rotation by communicating with the control plane components.
 - Automating routing information.
 - Automating DNS domain configuration.

How service mesh works in Kubernetes

- ▶ The data plane in a service mesh performs the following tasks:
 - Service discovery: Tracks the services deployed in a mesh.
 - Health checks: Track the state (healthy or unhealthy) of the services deployed in a mesh.
 - Traffic shaping and routing: Control the flow of network data between services.
 - Security: Perform authentication and authorization, and secure communication using mutual transport layer security (mTLS) between services in a mesh.
 - Metrics and Telemetry: Gather metrics, logs, and distributed tracing information from services in the mesh.

How service mesh works in Kubernetes

▶ Control Plane Components

- The control plane consists of the istiod deployment. The istiod deployment consists of a single binary that contains a number of APIs used by the OpenShift Service Mesh.
- Pilot
- Citadel
- Galley

How service mesh works in Kubernetes

- ▶ Pilot
 - Maintains the configuration data for the service mesh.
- ▶ Citadel
 - Issues and rotates TLS certificates.
- ▶ Galley
 - Monitors the service mesh configuration and then validates, processes, and distributes the configuration to the proxies.

Benefits of using service mesh

- ▶ Better and easier service discovery
- ▶ More elasticity
- ▶ Easier for Security control such as authentication techniques to validate the identity of the request
- ▶ Better Monitoring, Centralized logging, Tracing

Comparison of service mesh tools

	Istio (OpenShift)	Linkerd	Consul
Current version	1.13	2.11	1.15
License	Apache License 2.0	Apache License 2.0	Mozilla License
Initiated by	Google, IBM, Lyft	Buoyant	HashiCorp
Service Proxy	<u>Envoy, proxyless for gRPC (experimental)</u>	<u>Linkerd2-proxy</u>	<u>defaults to Envoy, exchangeable</u>
Ingress Controller	<u>Envoy / Own Concept, support for Kubernetes Gateway API</u>	any	<u>Envoy. Support for Kubernetes Gateway API with Consul API Gateway</u>
Governance	see Istio Community and Open Usage Commons	see Linkerd Governance and CNCF Charter	<u>see Contributing to Consul</u>
Used in production	<u>yes</u>	<u>yes</u>	<u>yes</u>

Comparison of service mesh tools

	Istio (OpenShift)	Linkerd	Consul
Advantages	Istio can be adapted and extended like no other mesh. Its many features are available for Kubernetes and other platforms.	Linkerd is designed to be non-invasive and is optimized for performance and usability. Therefore, it requires little time to adopt.	Consul service mesh can be used in any Consul environment and therefore does not require a scheduler. The proxy can be changed and extended.
Drawbacks	Istio's flexibility can be overwhelming for teams who don't have the capacity for more complex technology. Also, Istio takes control of the ingress controller.	Linkerd is deeply integrated with Kubernetes and does not currently support non-Kubernetes workloads. It also does not currently support data plane extensions.	Consul uses its own internal storage, and does not rely on Kubernetes for persistent storage.

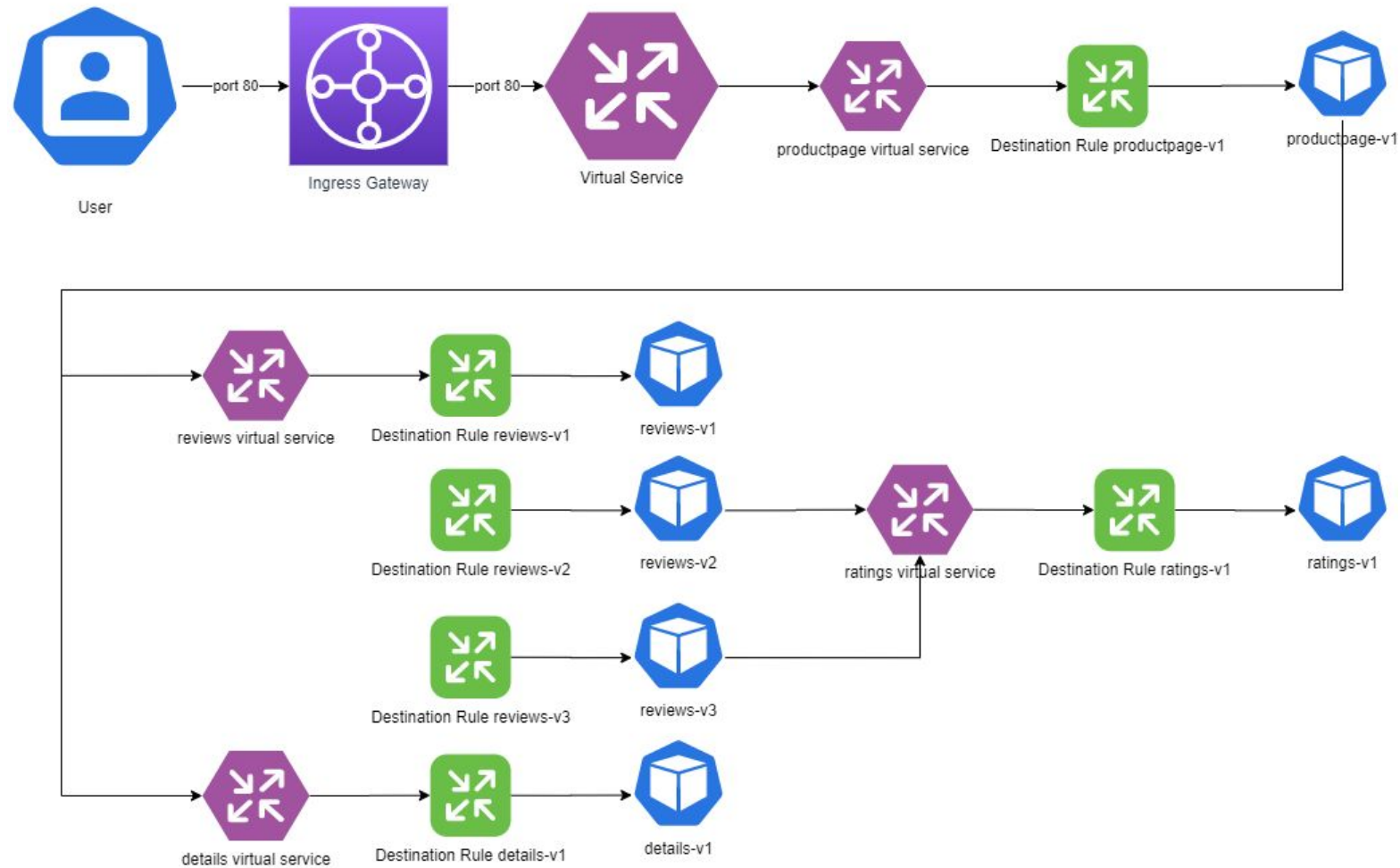
Comparison of service mesh tools

	Istio (OpenShift)	Linkerd	Consul
Sidecar / Data Plane			
Automatic Sidecar Injection	yes	yes	yes
CNI plugin to avoid pod network privileges	<u>yes, in beta</u>	<u>yes</u>	<u>yes</u>
Platform and Extensibility			
Platform	Kubernetes	Kubernetes	Kubernetes, Nomad, VMs, ECS, Lambda
Cloud Integrations	<u>Google Cloud, Alibaba Cloud, IBM Cloud</u>	<u>DigitalOcean</u>	<u>HCP Consul on AWS and Azure</u>
Mesh Expansion/Extension of the Mesh by containers/VMs outside the cluster	<u>yes</u>	no	<u>yes</u>
Multi-Cluster MeshControl and observe multiple clusters	<u>yes</u>	<u>yes</u>	<u>yes</u>

Key Resources for Istio

- ▶ DestinationRule
 - defines policies that apply to traffic intended for a service after routing has occurred.
- ▶ VirtualService
 - defines a set of traffic routing rules to apply when a host is addressed.
- ▶ Gateway
 - describes a load balancer operating at the edge of the mesh receiving incoming or outgoing HTTP/TCP connections

Key Resources for Istio



Demonstration: Using OpenShift Servicemesh (Istio)

Security Considerations

Containers

- ▶ In general, continuous container security for the enterprise is about:
 - Securing the container pipeline and the application
 - Securing the container deployment environment(s) and infrastructure
 - Securing the containerized workloads at runtime

Containers

- ▶ Basic steps for build security into the container pipeline
 - Gather images
 - use trusted images
 - use an image scanner,
 - Anticipate and remediate vulnerabilities
 - The supply chain needs more security policy services
 - Security teams need to balance the networking and governance needs of a containerized environment.

Docker

Host Configuration

Applies To	Configuration Element	Recommendation
All Hosts	Host operating system	Ensure the OS is hardened
	Docker binary	Ensure Docker version is up to date
Linux Hosts	Disk partitions	Create a separate partition for containers
	User accounts	Only grant access to Docker daemon to trusted users
	Docker files and directories, in particular: Docker.service Docker.sock /etc/default/docker /etc/docker/daemon.json /usr/bin/containerd /usr/sbin/runc	Ensure auditing is configured

Docker

Docker Daemon Configuration

Configuration Element	Recommended Setting
Network traffic between containers on default bridge	restricted
Logging level	info
Docker permission to make changes to iptables	allow
Insecure registries	do not use
Aufs storage driver	do not use
TLS authentication	use and configure correctly

Configuration Element	Recommended Setting
Default ulimit	configure as appropriate
User namespace support	enabled
Default cgroup usage	confirm it is used
Base device size	do not change until needed
Docker client command authorization	enabled
Centralized logging	configured

Docker

Docker Daemon Configuration

Configuration Element	Recommended Setting
Remote logging	configured
Live restore	enabled
Userland Proxy	disabled
Custom seccomp profile	applied if appropriate
Experimental features	do not use in production
Container ability to gain new privileges	restricted

Docker

Docker Daemon Configuration Files

File/Directory to Secure	File Permissions	File Ownership
docker.service file	as appropriate	root:root
docker.socket file	644 or stricter	root:root
/etc/docker directory	755 or stricter	root:root
registry certificate file	444 or stricter	root:root
TLS CA certificate file	444 or stricter	root:root
Docker server certificate file	444 or stricter	root:root

Docker

Docker Daemon Configuration Files

File/Directory to Secure	File Permissions	File Ownership
Docker server certificate key file	400 or stricter	root:root
Docker socket file	660 or stricter	root:docker
daemon.json file	644 or stricter	root:root
/etc/default/docker file	644 or stricter	root:root
/etc/sysconfig/docker file	644 or stricter	root:root

Docker

Container Images and Build File Configuration

Configuration Element	Recommendations
Permissions	<ol style="list-style-type: none">1. Create a user for the container2. Remove setuid and setgid permissions
Container content	<ol style="list-style-type: none">1. Avoid unnecessary packages in the container2. Only install verified packages3. Define HEALTHCHECK instructions for the container4. Enable content trust for Docker
Images	<ol style="list-style-type: none">1. Only use trusted base images2. Perform security scans on images3. Rebuild images to include security patches
Dockerfiles	<ol style="list-style-type: none">1. Ensure update instructions are not use alone2. Use COPY instead of ADD3. Do not store secrets in Dockerfiles

Docker

Container Runtime Configuration

Configuration Element	Recommended Setting
AppArmor Profile	enabled (if applicable)
SELinux security options	set (if applicable)
Linux kernel	access restricted within containers
Privileged containers	do not use
Sensitive host directories	never mount on a container
sshd	never run on a container
ports	1. Do not map privileged ports in containers 2. Only open needed ports

Configuration Element	Recommended Setting
Host network namespace, IPC namespace, UTS namespace	do not set to shared
Container resource utilization	1. Limit memory usage 2. Set CPU priority
Container root file system	mount as read only
Incoming container traffic	restrict to specific host interface
'on-failure' restart policy	5
Host devices	do not expose to containers
Default ulimit	overwrite at runtime if needed

Docker

Container Runtime Configuration

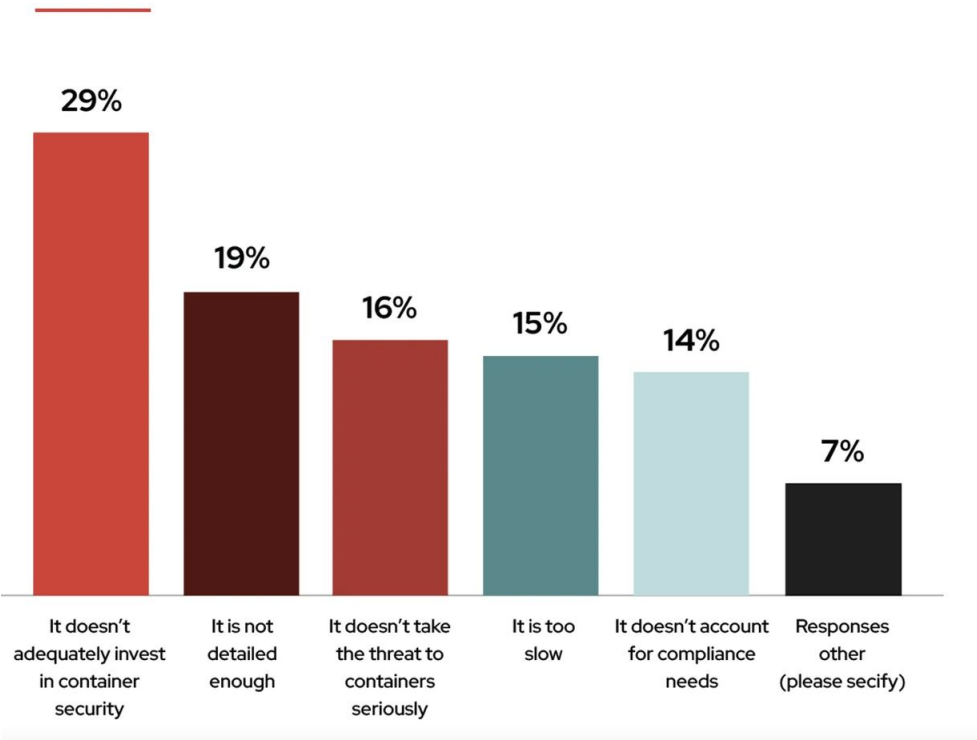
Configuration Element	Recommended Setting
Mount propagation	do not set to shared
Default seccomp profile	do not disable
Docker exec commands	1. Do not use with privileged option 2. Do not use with user=root
cgroup	1. confirm it is used 2. ensure PIDs cgroup limit is used

Configuration Element	Recommended Setting
container additional privileges	restrict
container health check	always perform at runtime
Docker commands	always use latest version of image
Docker default bridge "docker0"	do not use
Docker socket	never mount inside containers

Kubernetes

Most Common Kubernetes Security Issues

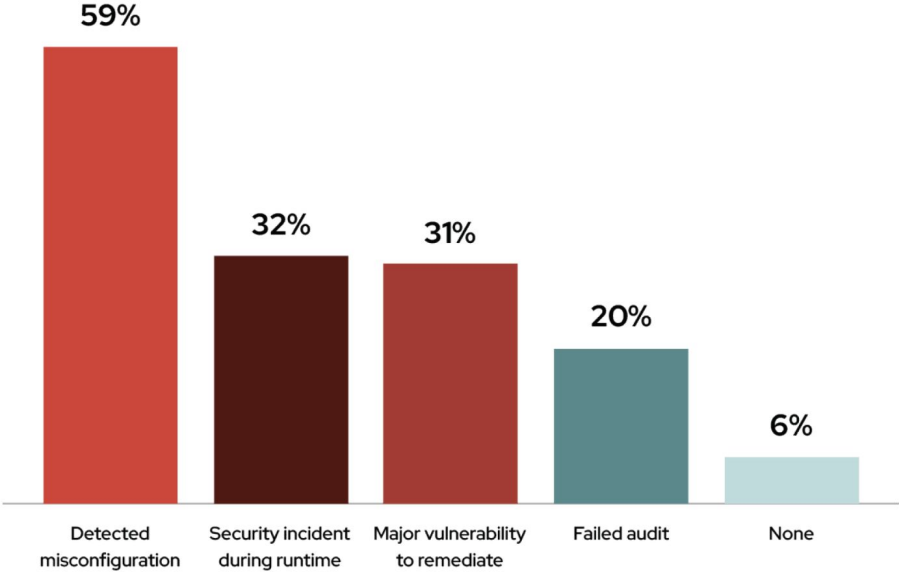
What is your biggest concern about your company's container strategy?



Kubernetes

In the past 12 months, what security incidents or issues related to containers and/or Kubernetes have you experienced?

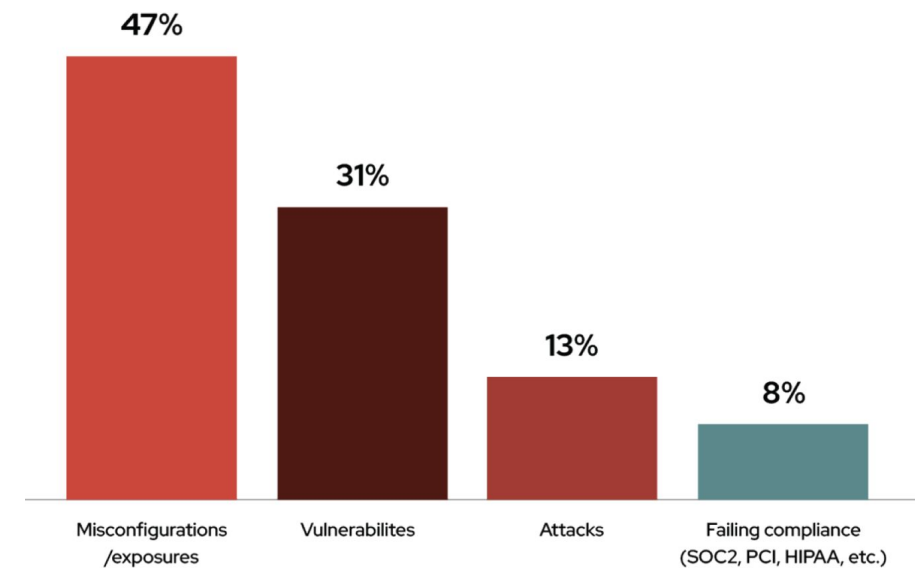
Most Common Kubernetes Security Issues



Kubernetes

Most Common Kubernetes Security Issues

Of the following risks, which one are you most worried about for your container and Kubernetes environments?



Kubernetes – CIS Benchmark in brief

Master Node Configuration Files

Control Plane Component	File/Directory to Secure	File Permissions	File Ownership
API Server	Pod specification file	644 or stricter	root:root
Controller Manager	Pod specification file	644 or stricter	root:root
Controller Manager	controller-manager.conf	644 or stricter	root:root
Scheduler	Pod specification file	644 or stricter	root:root
Scheduler	scheduler.conf	644 or stricter	root:root
etcd	Pod specification file	644 or stricter	root:root
Container Network Interface	CNI file	644 or stricter	root:root

Kubernetes – CIS Benchmark in brief

Master Node Configuration Files

Control Plane Component	File/Directory to Secure	File Permissions	File Ownership
etcd	Data directory	700 or stricter	etcd:etcd
kubeadm	admin.conf	644 or stricter	root:root
Kubernetes PKI	PKI directory		root:root
Kubernetes PKI	PKI certificate	644 or stricter	
Kubernetes PKI	PKI key files	600	

Kubernetes – CIS Benchmark in brief

API Server

Configuration Element	Recommended Setting
–anonymous-auth	false
–basic-auth-file	not set
–anonymous-auth	false
–basic-auth-file	not set
–token-auth-file	not set
–kubelet-https	true
–anonymous-auth	false

Configuration Element	Recommended Setting
–authorization-mode	AlwaysAllow, includes Node and RBAC
admission control plugin	EventRateLimit is set AlwaysAdmit not set AlwaysPullImages is set SecurityContextDeny set (if PodSecurityPolicy not used) ServiceAccount is set NamespaceLifecycle is set PodSecurityPolicy is set NodeRestriction is set
–insecure-bind-address	not set

Kubernetes – CIS Benchmark in brief

API Server

Configuration Element	Recommended Setting
–secure-port	0
–profiling	false
–audit-log-path	false
–audit-log-maxage	30
–service-account-lookup	True
encryption providers	Only use strong cryptographic ciphers
–etcd-certfile	Ensure these parameters / arguments are set to their appropriate values

Configuration Element	Configuration Element
–etcd-keyfile	–kubelet-client-key
–tls-cert-file	–kubelet-certificate-authority
–tls-private-key-file	–audit-log-maxbackup
–client-ca-file	–audit-log-maxsize
–etcd-cafile	–request-timeout argument
–encryption-provider-config	–service-account-key-file
–kubelet-client-certificate	

Kubernetes – CIS Benchmark in brief

Controller Manager

Configuration Element	Recommended Setting
–profiling	false
–use-service-account-credentials	true
RotateKubeletServerCertificate	true
–bind-address	127.0.0.1
–terminated-pod-gc-threshold	Ensure these parameters / arguments are set to their appropriate values
–service-account-private-key-file	
–root-ca-file	

Kubernetes – CIS Benchmark in brief

Scheduler

Configuration Element	Recommended Setting
–profiling	false
–bind-address	127.0.0.1

Kubernetes – CIS Benchmark in brief

ETCD

Configuration Element	Recommended Setting
–client-cert-auth	true
–auto-tls	true
–peer-client-cert-auth	true
–peer-auto-tls	true
Certificate Authority	Use a unique CA
–cert-file	Ensure these parameters / arguments are set to their appropriate values
–key-file	
–peer-cert-file	
–peer-key-file	

Project Management - Methodology and Process Automation

Overview of Agile, DevOps and DevSecOps



Agile

- ▶ Agile is an approach to software development that seeks the continuous delivery of working software created in rapid iterations.
- ▶ In practical terms, agile software development methodologies are all about delivering small pieces of working software quickly to improve customer satisfaction.

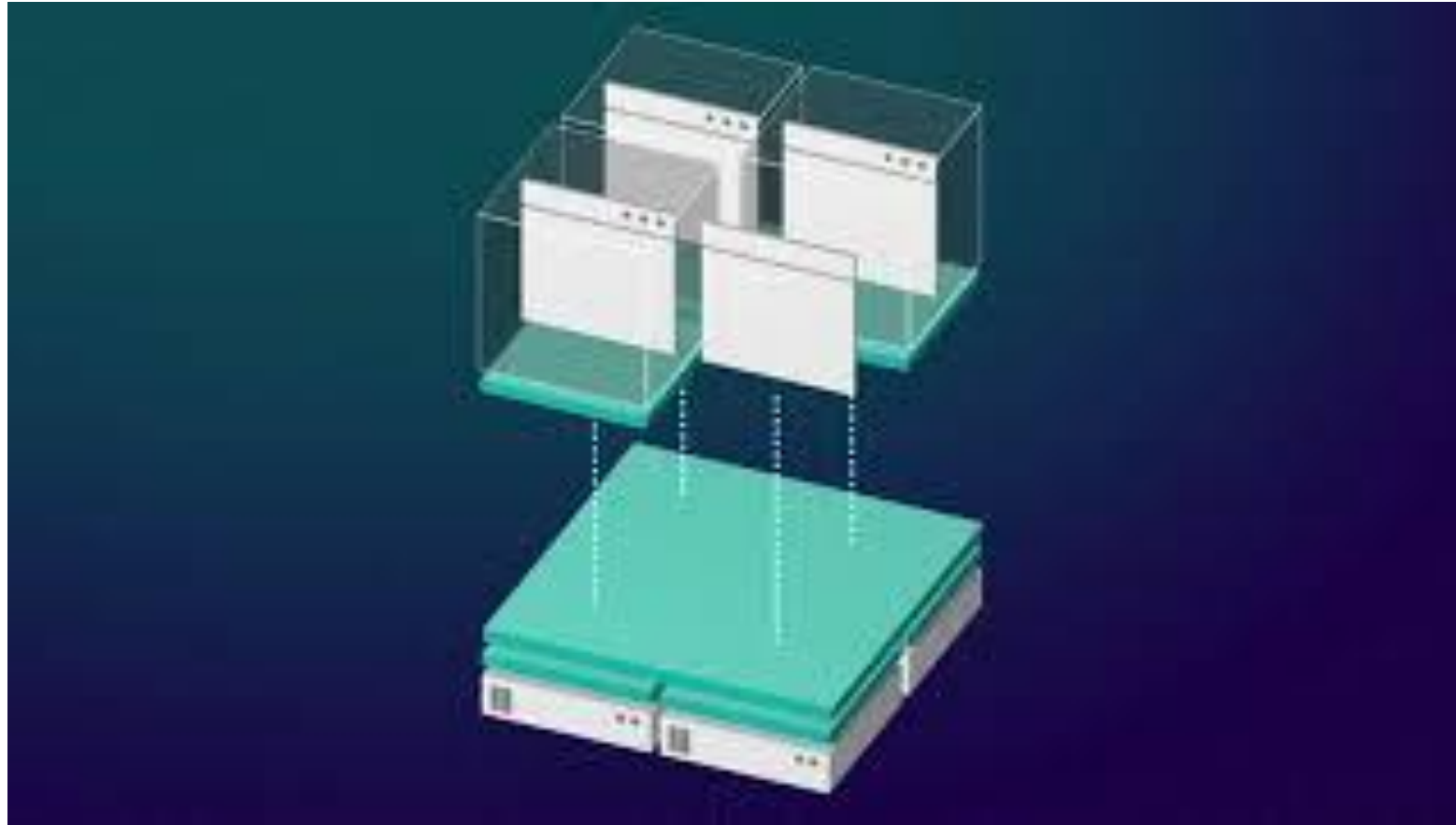
Agile Value

- ▶ Agile as we know it today traces its history to 2001. Reacting to waterfall approaches to project management, which organizes a software project as a series of linear sequences, a group of software developers penned The Manifesto for Agile Software Development.
 - Individuals and interactions over processes and tools
 - Working software over comprehensive documentation
 - Customer collaboration over contract negotiation
 - Responding to change over following a plan

DevOps

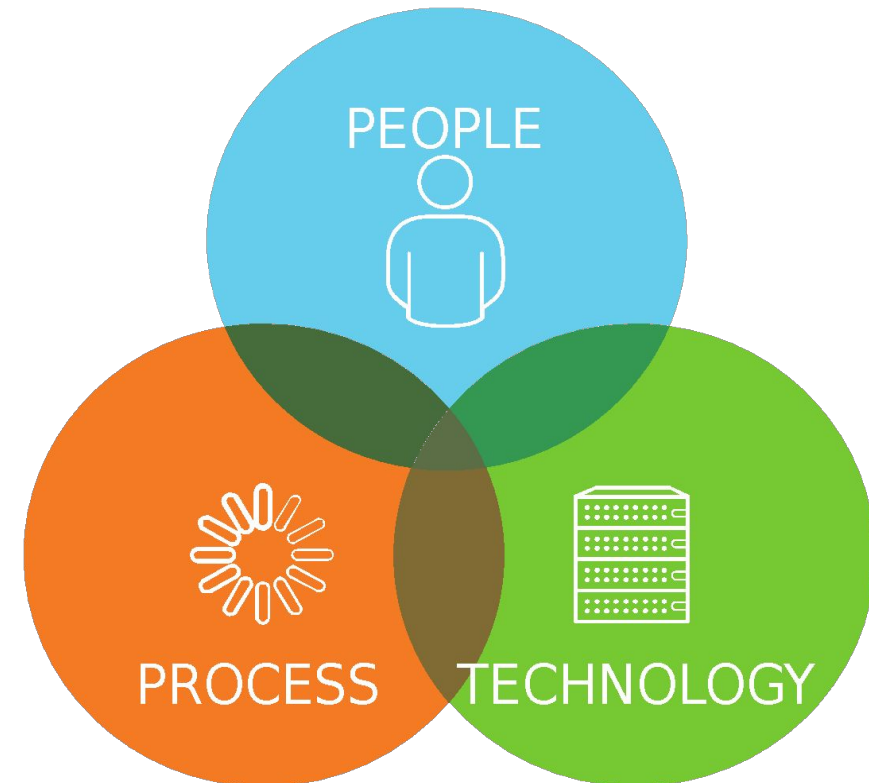
- ▶ The word "DevOps" is a mashup of "development" and "operations" but it represents a set of ideas and practices much larger than those two terms alone, or together. DevOps includes security, collaborative ways of working, data analytics, and many other things.

DevOps



DevOps from Red Hat's point of view

DevOps is the union of **people**, **process**, and **technology** to enable continuous delivery of **value to your end users**.



What DevOps is and is NOT?



What DevOps is NOT

- A single product/solution (nor a bunch of tools)
- A job role
- A process
- Not just an extension of Agile

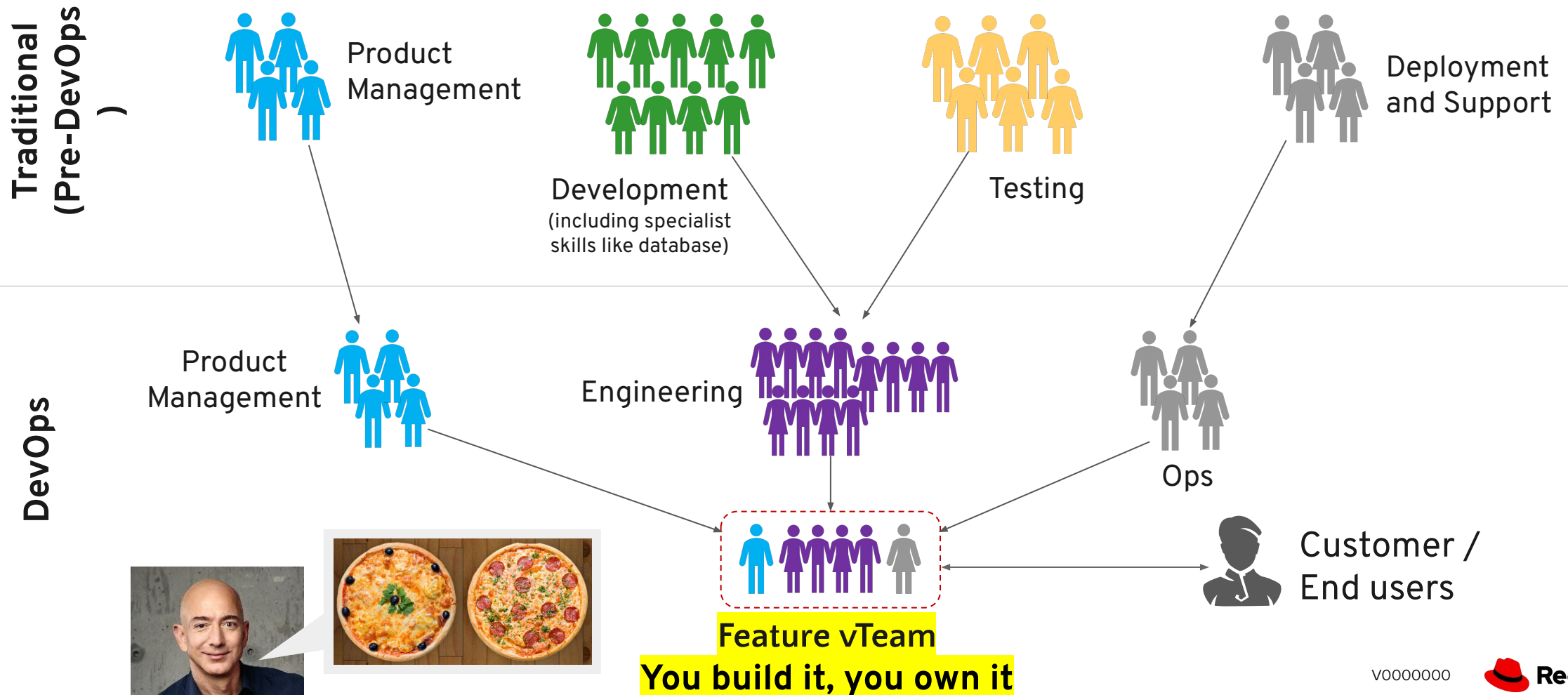


What is DevOps then?

- A set of **principles** for software delivery
- **Cultural change** - collaboration of cross-functional teams (not just Dev and Ops - Security, Test, Business too!)
- Focus on **automation, continuous improvement**

Cultural change: create cross-functional teams

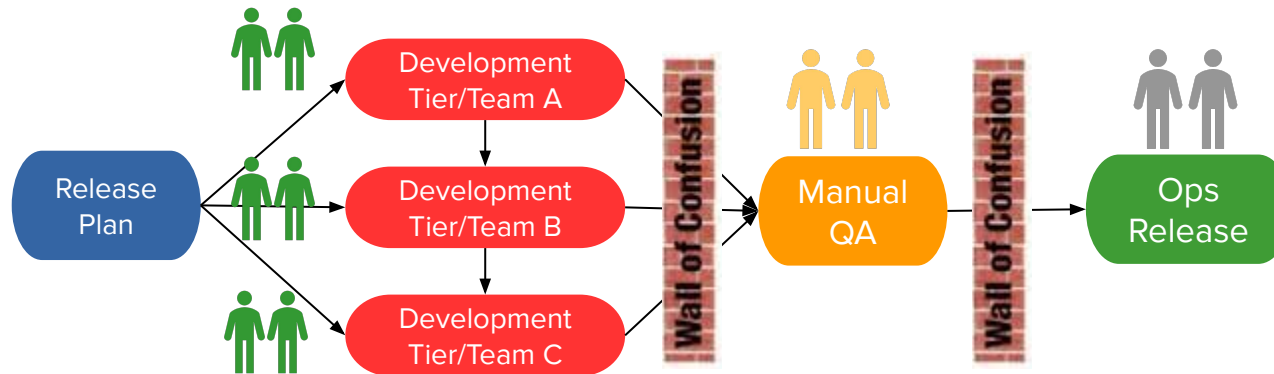
Reduce conflict of interest, same goal with trust



Each feature vteam has certain power of autonomy

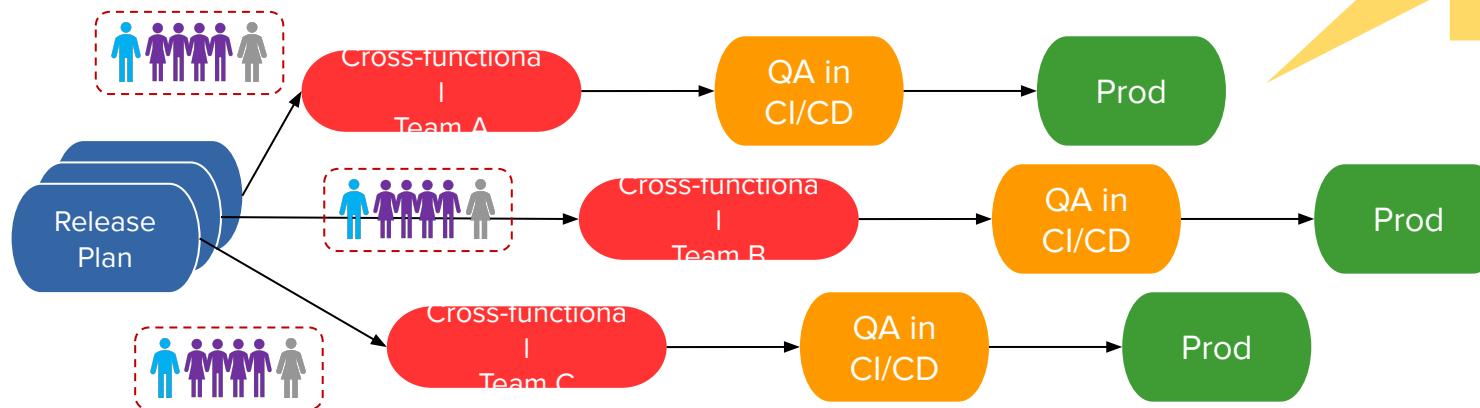
When the whole team works together at the BEGINNING, things will speed up

Traditional
(Pre-DevOps)



Autonomy ≠ no governance. Standard and governance are controlled in CI/CD.

DevOps

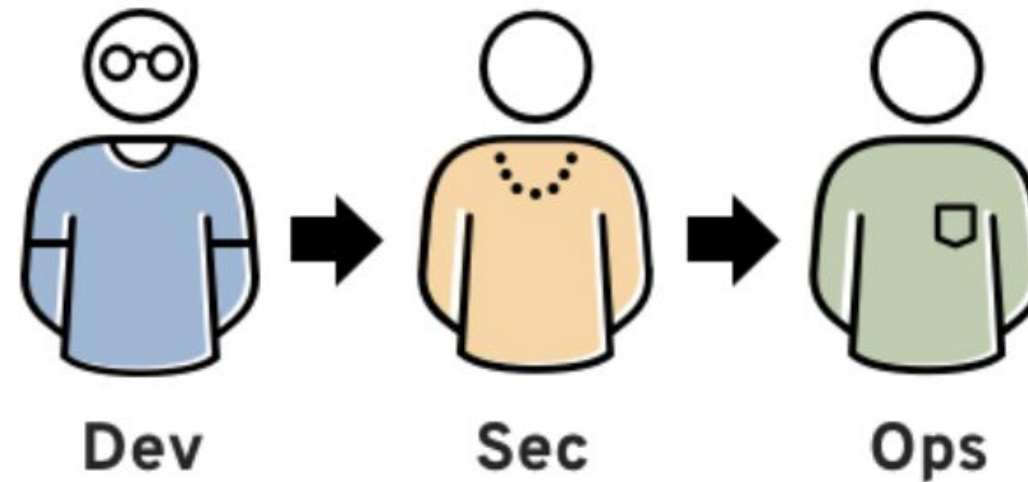


DevSecOps

- ▶ DevSecOps stands for development, security, and operations. It's an approach to culture, automation, and platform design that integrates security as a shared responsibility throughout the entire IT lifecycle.

DevSecOps

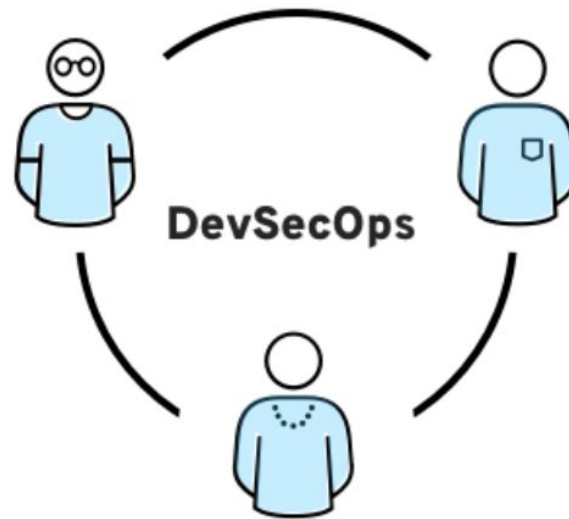
DevSecOps vs. DevOps



In the past, the role of security was isolated to a specific team in the final stage of development

DevSecOps

DevSecOps vs. DevOps



Now, in the collaborative framework of DevOps, security is a shared responsibility integrated from end to end.

DevSecOps

- ▶ DevOps security is built-in
 - Whether you call it “DevOps” or “DevSecOps,” it has always been ideal to include security as an integral part of the entire app life cycle. DevSecOps is about built-in security, not security that functions as a perimeter around apps and data.
- ▶ DevOps security is automated
 - Organizations should step back and consider the entire development and operations environment. This includes source control repositories, container registries, the continuous integration and continuous deployment (CI/CD) pipeline, application programming interface (API) management, orchestration and release automation, and operational management and monitoring.

Overview of CI/CD



Overview of CI/CD

- ▶ CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development.
- ▶ The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment.
- ▶ CI/CD is a solution to the problems integrating new code can cause for development and operations teams (AKA "integration hell").
- ▶ Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment.

Overview of CI/CD

What's the difference between CI and CD (and the other CD)?

- ▶ CI:
 - The "CI" in CI/CD always refers to continuous integration, which is an automation process for developers.
- ▶ CD:
 - The "CD" in CI/CD refers to **continuous delivery** and/or **continuous deployment**, which are related concepts that sometimes get used interchangeably.

Overview of CI/CD



Comparison of CI/CD tools

▶ Tekton

▶ Argo

▶ GitHub Actions

▶ Jenkins X

▶ OpenShift Pipelines

▶ Spinnaker

▶ Circle CI

▶ GitLab

▶ Jenkins

Tekton

Tekton is a Kubernetes-native open source framework for creating continuous integration and delivery (CI/CD) systems.

Pros	Cons
Standardization	Limited to Kubernetes
Portability and flexibility	Less developed feature set than existing solutions
Kubernetes-native	
Built-in best practices	
Open source	

Argo

The Argo Project is a comprehensive group of tools to help you orchestrate all of your container-native workflows.

Pros	Cons
Standardization	Git as a source of truth
Portability and flexibility	Longer implementation time
Kubernetes-native	
Git as a source of truth	
Open source	

GitHub Actions

GitHub Actions was first released in November 2019 and has become a handy tool for open source and enterprise applications.

Pros	Cons
Standardization	Missing more complex CD uses cases
Simpler GitOps	Users are limited to GitHub cloud service, although self-hosted runners are now available
Easy Setup	
Git as a source of truth	
Free tier	
GitHub marketplace for sharing workflows	

Jenkins X

Jenkins X automates the continuous delivery using Git as a source of truth and creates previews on pull requests to help you accelerate delivery.

Pros	Cons
Git as a source of truth	Limited to Kubernetes
Based off Tekton pipelines	It only works with GitHub for now
Tekton with enterprise support and features	
Feedback on issues and pull requests	

OpenShift Pipelines

OpenShift Pipelines is a Kubernetes-native CI/CD solution based on Tekton.

Pros	Cons
Git as a source of truth	Limited to OpenShift platform
Based off Tekton pipelines	
Tekton with enterprise support and features	
Uses OpenShift Operator for application management	

Spinnaker

Spinnaker is an open source, multi-cloud continuous delivery platform that provides application management and deployment, enabling you to release software changes with high velocity and confidence.

Pros	Cons
Fast and flexible deployments	CD specific tool
Out-of-the-box deployment strategies.	Spinnaker requires multiple microservices and management can be time-consuming.
More developed CD feature set than other solutions	
One-click deployment rollbacks	

Circle CI

CircleCI is an open source CI/CD tool built for integration into your version control system. CircleCI can be on-premise or cloud-hosted and you can use it for free for a limited time.

Pros	Cons
Simple integration with the provided YAML template	Existing cloud tooling can cover most of the same use cases
Quick setup with Git Repositories	Complex configurations can be challenging with the Circle.yml file
Simplified hosted service	
Easy configuration via the Circle.yml file	

GitLab

GitLab touts itself as the all-in-one DevOps platform, and for good reason.

Pros	Cons
Pipeline and CI setup using existing industry standards	Open source is lacking all the top features
Source control for any cloud	It can be quite technical and challenging for people to learn
Declarative GitLab runners configured through YAML file	
GitLab Runners can be configured for continuous delivery and continuous deployment	

Jenkins

Jenkins is an automation engine that supports multiple automation patterns, such as pipelines, scripted tasks, and scheduled jobs.

Pros	Cons
written in Java, which makes it available for multiple operating systems	Not container native
extends its capabilities with the use of plug-ins.	Pipeline files has longer learning curve
can install Jenkins in macOS, Windows, and popular Linux distributions	

Demonstration: Using one of the OpenShift Pipeline (Tekkon)

Government Cloud Infrastructure Services (GCIS) and Container- as-a-Service (CaaS)

Use Cases of Containerised Applications

Real example of containerised applications



FAST FACTS

Industry: Transportation

Region: APAC

Location: Hong Kong, China

Company size: 33,000 employees

Real example of containerised applications

- ▶ The Hong Kong airline had to enhance its operational agility to adapt to changes in the aviation industry caused by the COVID-19 pandemic. To stay ahead of the market, it needed an IT model that could handle fluctuating passenger demands. The company leveraged open source solutions like Red Hat OpenShift and Red Hat Ansible Automation Platform to build a multi-cloud hybrid environment and applications using an agile container strategy. With Red Hat solutions, Cathay Pacific has reduced the operational costs of managing on-premises infrastructure and slashed capital expenditures for new servers. With the freeing of IT teams to focus on developing innovative customer-facing applications and solutions, the airline is able to better handle fluctuating demand.



Real example of containerised applications

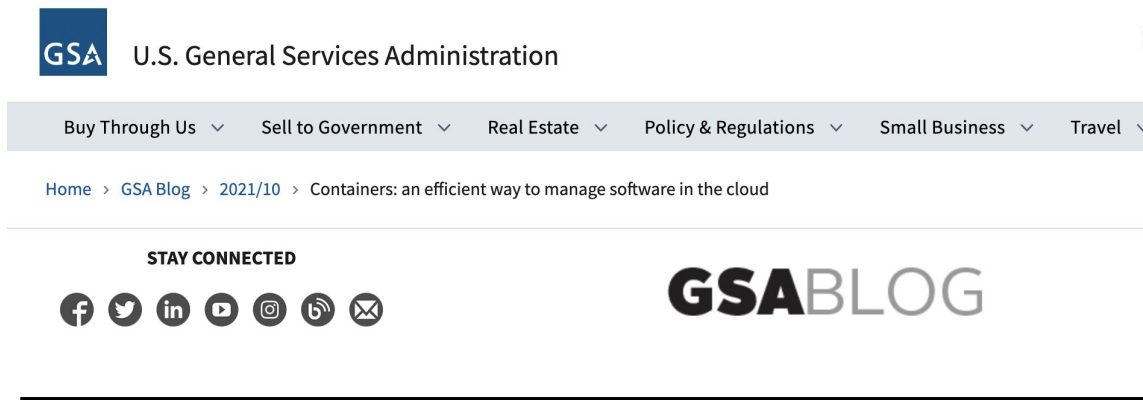


Real example of containerised applications

- ▶ In 2020, the insurance company embarked on a customer experience transformation initiative, which includes the launch of the K-Dollar reward points program. FTLife turned to Red Hat solutions, such as Red Hat OpenShift Container Platform, to build agility into its processes and accelerate the delivery of IT services and applications. Red Hat's open source approach helped FTLife overcome the limitations of traditional application development methods, helping teams build the K-Dollar application faster and minimizing human error through better collaboration. FTLife plans to replicate K-Dollar's success by using its development blueprint for future customer experience initiatives.



How containerised applications help government services



Containers: an efficient way to manage software in the cloud

October 07, 2021 | DCCOI Program Management Office
Post filed in: [Innovation](#)

As the federal government adopts cutting-edge technologies and embraces large changes to existing information technology (IT) infrastructure, containers have become a growing topic of discussion. Some agencies already have budding containerization practices, other agencies are building container capabilities and skills or are just beginning the process.

In collaboration with the Cloud and Infrastructure Community of Practice (C&I CoP), the GSA Data Center and Cloud Optimization Initia



Containerization Readiness Guide

May 12, 2021

Office of Information Integrity and Access

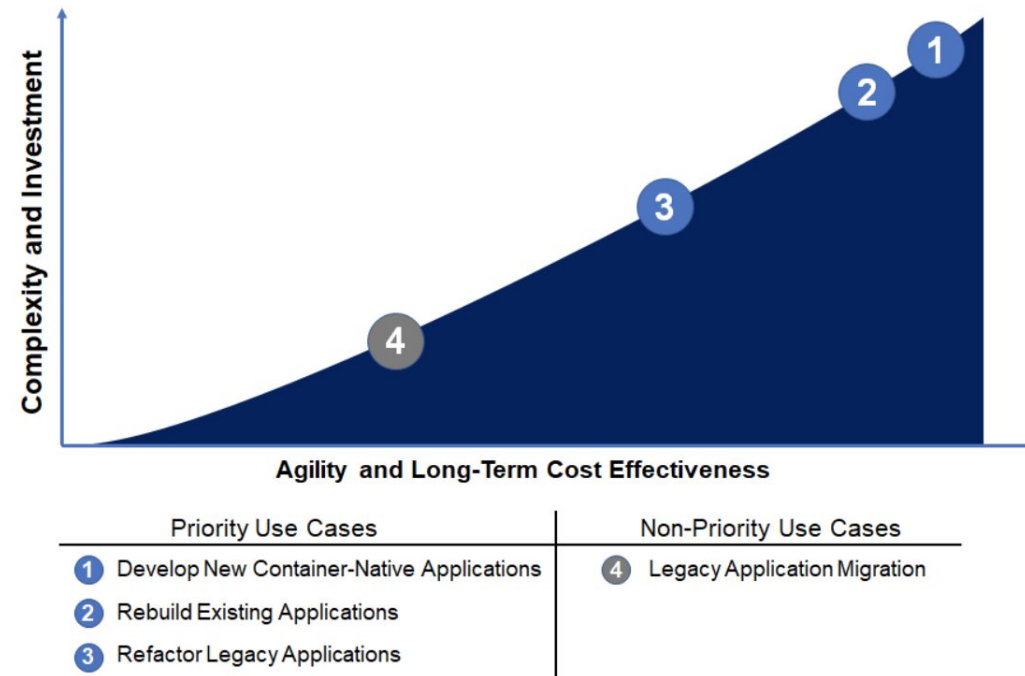
General Services Administration
Office of Government-wide Policy

How containerised applications help government services

- ▶ Develop applications quickly
 - Using containers frees developers from the tedious task of managing multiple configuration environments, supporting libraries, and configurations from testing to production environments. Containers can be created once and used multiple times without additional effort.
- ▶ Scale applications rapidly
 - Federal agencies can use a container management system to cluster multiple containers together, schedule and automate deployments, and manage containers to meet mission needs and priorities.
- ▶ Optimize compute resources
 - Unlike virtual machines, multiple containers can run on a single operating server or a single virtual machine due to their lightweight nature, and their ability to quickly execute and maintain a consistent runtime model.

How containerised applications help government services

Common Container Use Cases



Best practice for designing, building, and managing containerised applications

▶ Best Practice for deploying highly available applications to K8S

- Multiple replicas
- Update strategy
- Handling SIGTERM signal gracefully
- Probes
- External dependencies readiness
- PodDisruptionBudget
- AutoScaling
- Leverage Pod Topology Spread Constraints
- Deploy applications using Blue/Green or Canary strategies

Multiple replicas

- ▶ Running more than one instance of your pods ensures that deleting a single pod will not cause downtime.
- ▶ We recommend setting multiple replicas as part of a deployment

Update strategy

- ▶ There are two deployment strategies, “Rolling” and “Recreate”:
- ▶ The Rolling strategy is the default deployment strategy. It replaces pods, one by one, of the previous version of the application with pods of the new version while ensuring at least one pod is running.
- ▶ In Recreate deployment, we fully scale down the existing application version before we scale up the new version.
- ▶ We recommend using “RollingUpdate” when possible.

Handling SIGTERM signal gracefully

- ▶ When an deployment is restarted or a pod is deleted, Kubernetes sends a SIGTERM signal to all the containers of the pod in an attempt to give the container an opportunity to gracefully shut down.
- ▶ Handle the SIGTERM signal in applications to ensure that applications shut down gracefully.
- ▶ Adjust the terminationGracePeriod setting as required.

Probes

- ▶ Probes (health checks) play a vital role in monitoring application health.
- ▶ Readiness probes determine whether an application is ready to accept traffic. Liveness probes determine if the application should be restarted.
- ▶ Once the startup probe returns successfully, the liveness and readiness probes will begin monitoring the application's health:
- ▶ Our recommendation is to leverage Liveness probes and Readiness probes to help ensure applications are healthy and constantly in a state to serve traffic.

External dependencies readiness

- ▶ You can use `initContainers/startupProbe` to check external dependencies before running your main container.
- ▶ We recommend using an `initContainer` or `startupProbe` to postpone application startup until dependencies are healthy.

External dependencies readiness

- ▶ You can use `initContainers/startupProbe` to check external dependencies before running your main container.
- ▶ We recommend using an `initContainer` or `startupProbe` to postpone application startup until dependencies are healthy.

PodDisruptionBudget

- ▶ A PDB (Pod Disruption Budget) limits the number of pod replicas that the cluster is allowed to take down for maintenance operations
- ▶ PDB would ensure that the number of replicas running is never brought below the number specified.
- ▶ PDB is recommended for critical applications running in production.

AutoScaling

- ▶ HPA is a feature used to scale pods out automatically based on gathered metrics.
- ▶ While an HPA is used to scale additional pods to meet demand, a VPA is used to scale resources vertically for individual pods. A VPA optimizes the CPU and memory request values and can maximize the efficiency of cluster resources.

AutoScaling

Things to consider before using VPA:

- You must have a minimum of two replicas for the VPA to automatically delete pods.
- Pods must be running in the project before VPA can recommend resources and apply the recommendations to new pods.
- VPA reacts to most out-of-memory events, but not in all situations.

Recommendations

- Avoid using HPA and VPA in tandem, unless you configure the HPA to use either custom or external metrics.
- In production, use VPA in Recommendation mode. That would be helpful to understand what the optimal resource request values are and how they vary over time.
- Use HPA for a sudden increase in resource usage over VPA, as VPA provides recommendations over a longer time period.

Leverage Pod Topology Spread Constraints

- ▶ If all pod replicas are scheduled on the same failure domain (such as a node, rack, or availability zone), and that domain becomes unhealthy, downtime will occur until the replicas are redeployed onto another domain.
- ▶ Use Pod Topology Spread Constraints to ensure that pods are distributed across failure domains.


Deploy applications using Blue/Green or Canary strategies

- ▶ Blue/Green deployments involve deploying a new version of your application alongside the old version.
- ▶ Canary deployments allow you to route a subset of traffic to the new version of your application.
- ▶ Use Blue/Green or Canary deployments to prevent disruptions during the rollout of new application versions. Use Canary deployments if greater flexibility of the traffic rollout is required.

Thank you

 linkedin.com/company/red-hat

 facebook.com/redhatinc

 youtube.com/user/RedHatVideos

 twitter.com/RedHat

