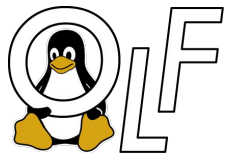
Introduction to Kubernetes and OpenShift for System Administrators

Ohio LinuxFest (OLF) ‘23

Jay Ryan
Senior Solutions
Architect

Brad Krumme
Senior Platform Solutions
Architect



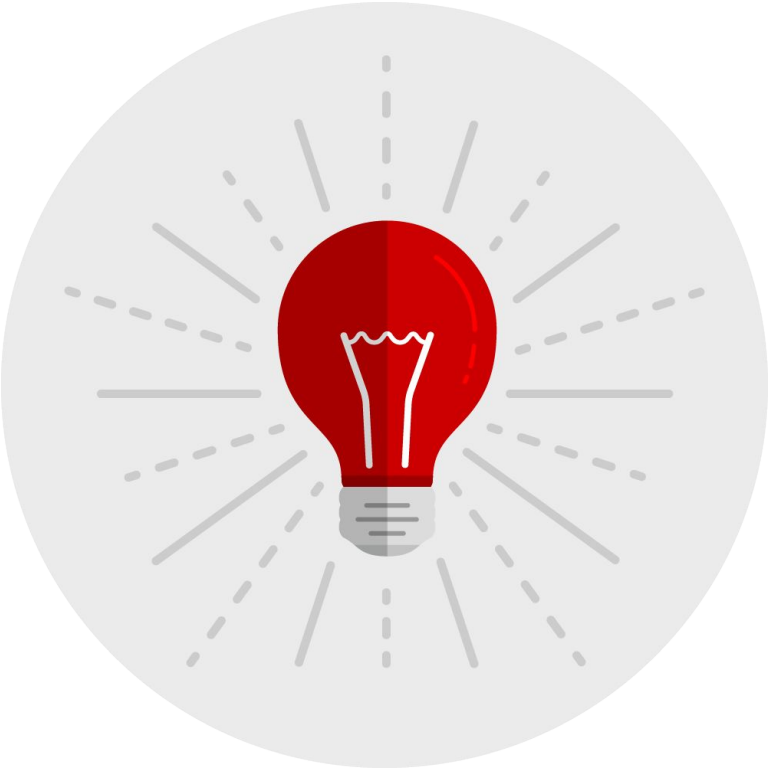
Red Hat

AGENDA

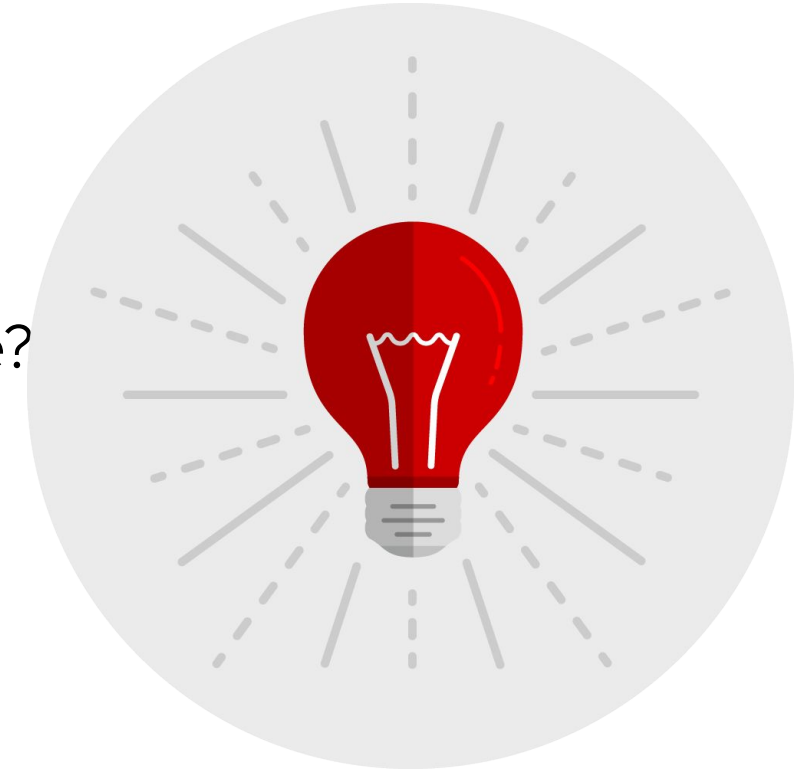
Morning Session 09:00→ 12:00 EST

Lunch Break 12:00→ 1:00 EST

Afternoon Session 1:00→ 5:00 EST



AGENDA



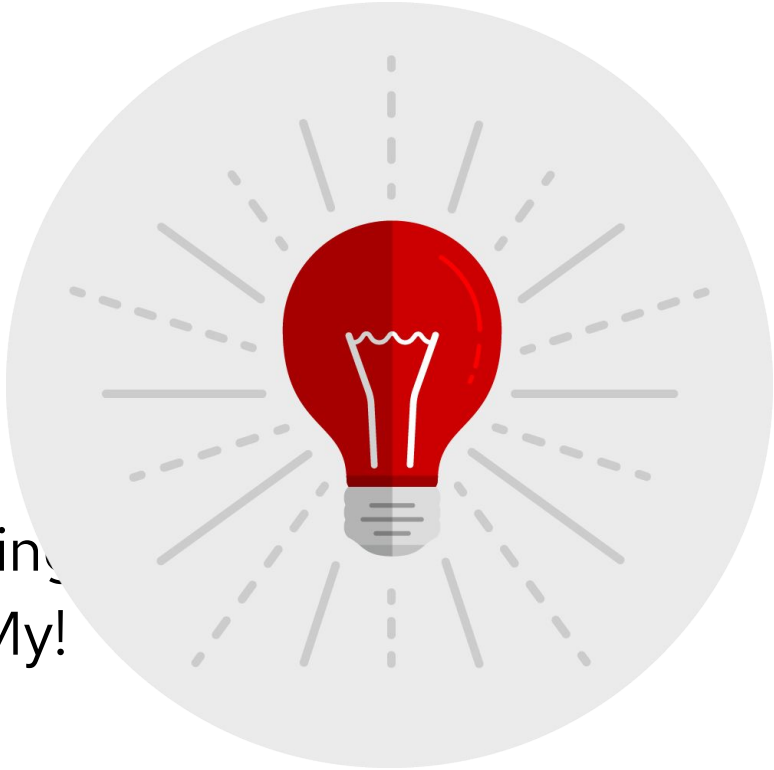
Morning Session

- Kubernetes and Containers - Why are we here?
- Container Technology Introduction
- What is Kubernetes
- Kubernetes Cluster Fundamentals
- 10 minute break (~10:00-10:30?)
- Workshop Introduction
- LAB: Module 1 & 2
- LAB: Installing / Managing Applications

AGENDA

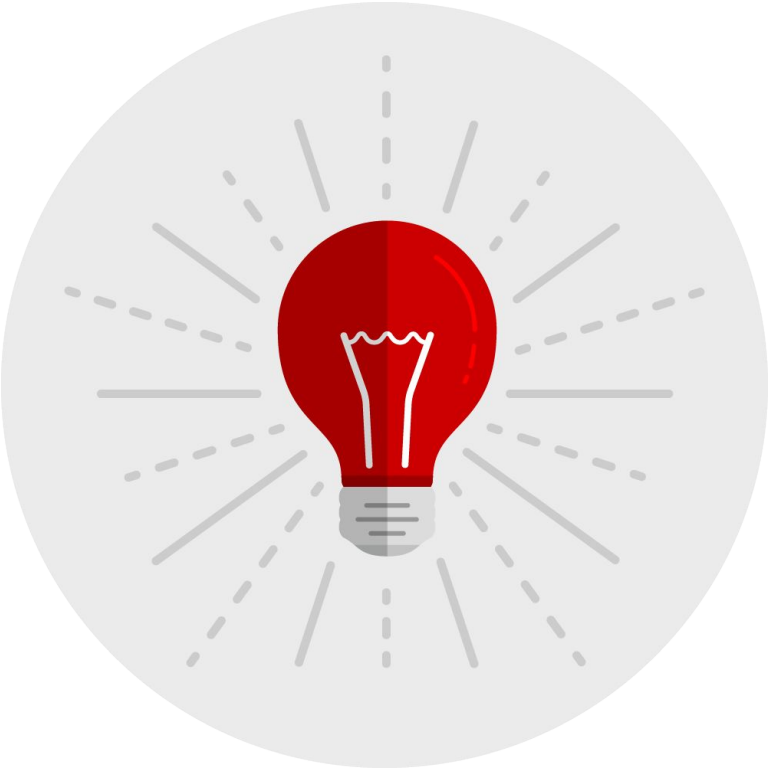
Afternoon Session

- Security fundamentals
- LAB: Managing and Scaling Nodes
- Day 2 Operations:
  - LAB: Logging / Authentication /Monitorin
  - LAB: Templates / Quotas / Limits - Oh My!
  - LAB: Networking / NetworkPolicy
- LAB: GitOps
- Bonus LAB: Windows Containers!!!



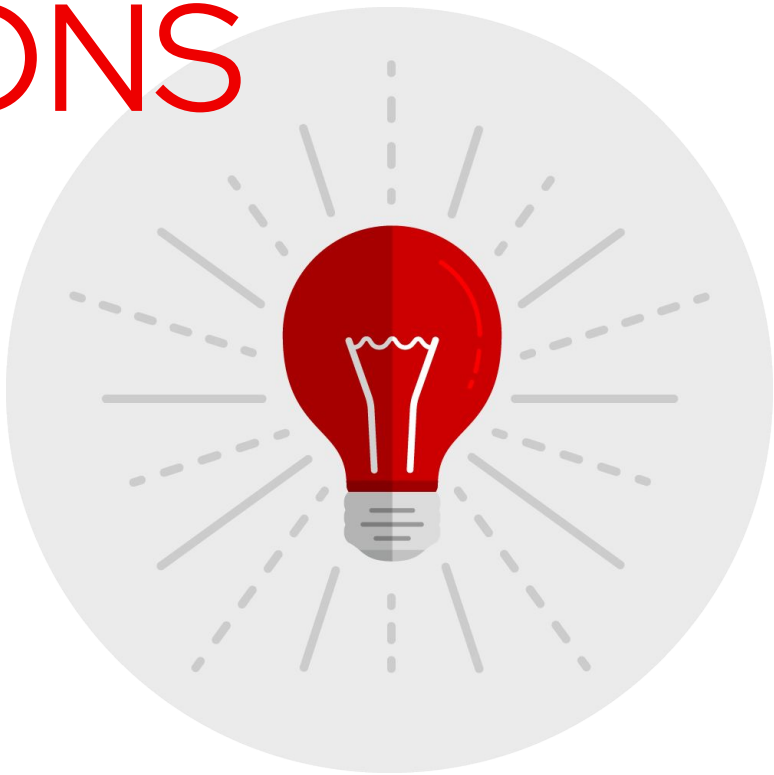
LOGISTICS

- Requirements - A laptop with internet access ~~and ssh access to a public address~~
- No food or drinks on the tables
- There are no stupid questions
- I don't know / I'll get back to you
- Small group / Let's make it Interactive
- You will get out what you put in
- Building Logistics / Questions?



INTRODUCTIONS

- What Company or Organization are you from?
- What's your role (not your title!)
- What are you looking to get out of this Lab today?

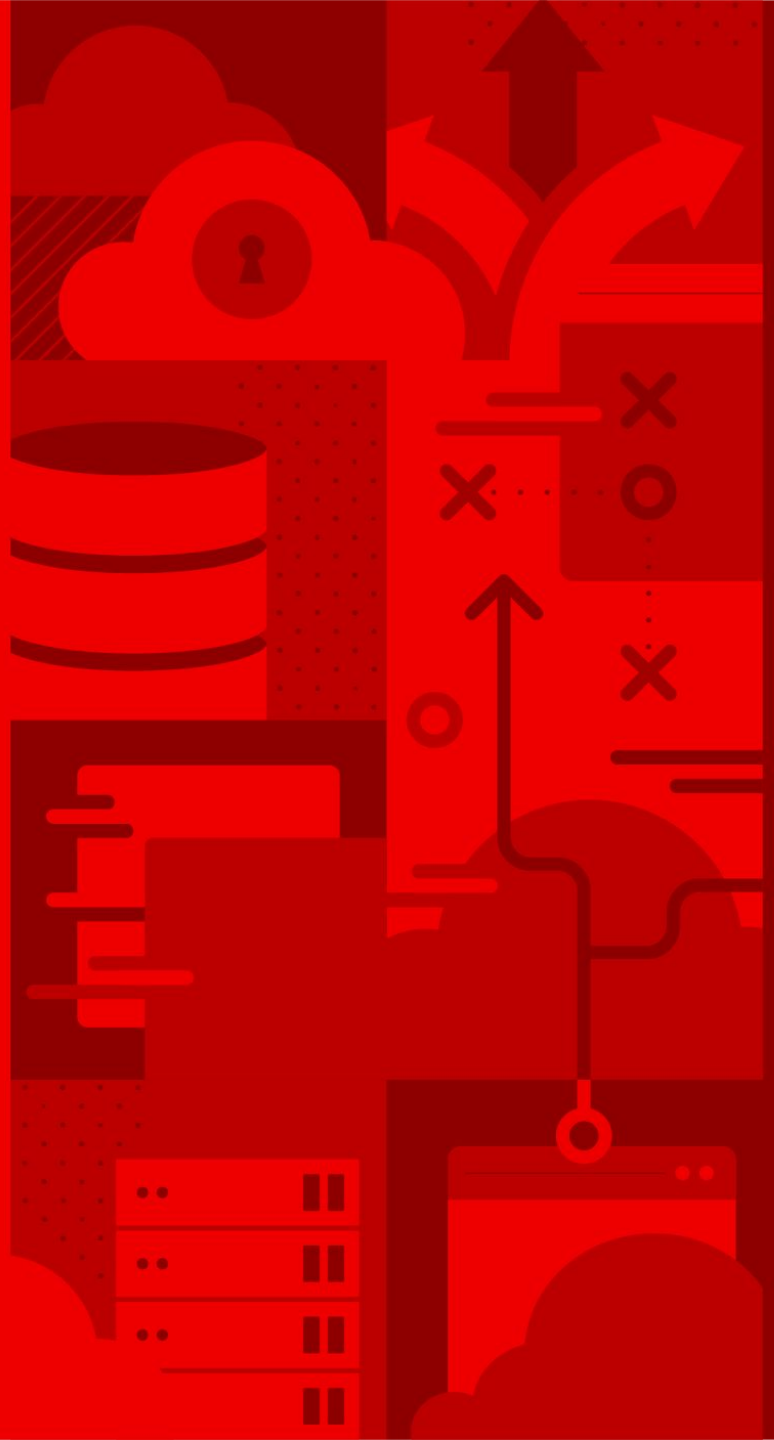


About Me!

Jay Ryan

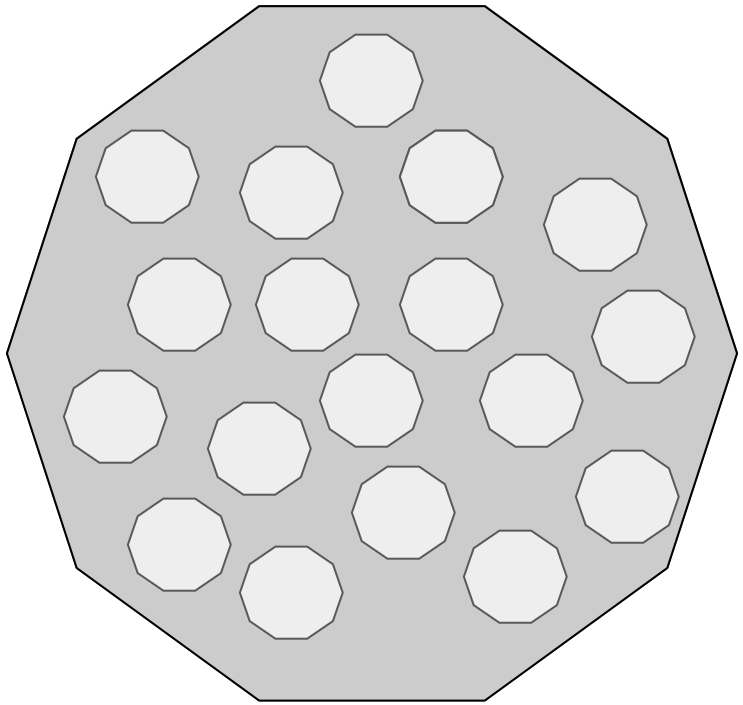
Senior Account Solutions Architect
@Red Hat

@jaywryan

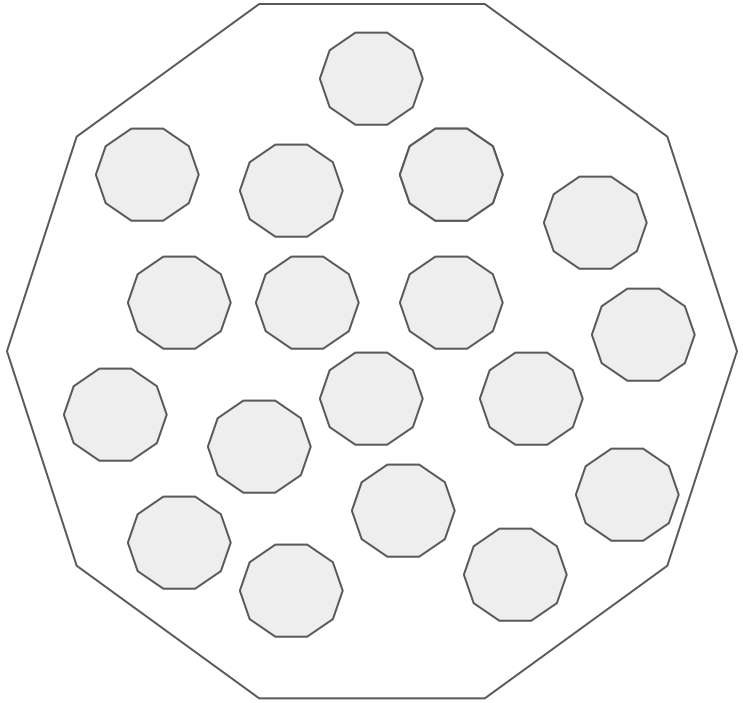


Kubernetes and Containers: Why are we here?

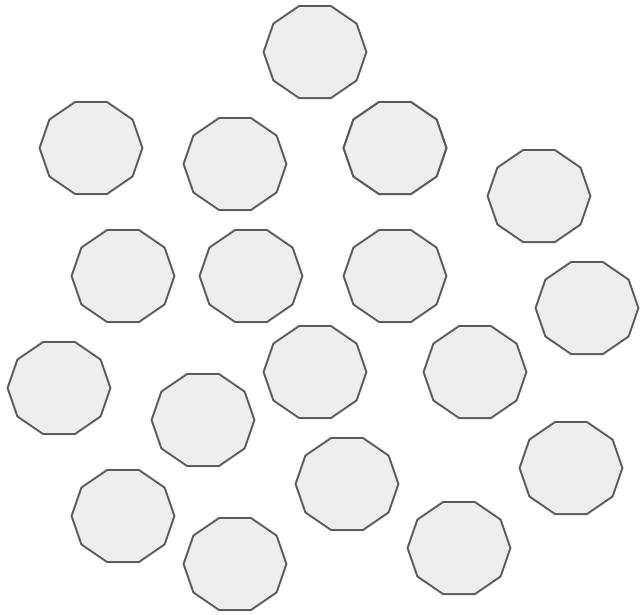
The Application



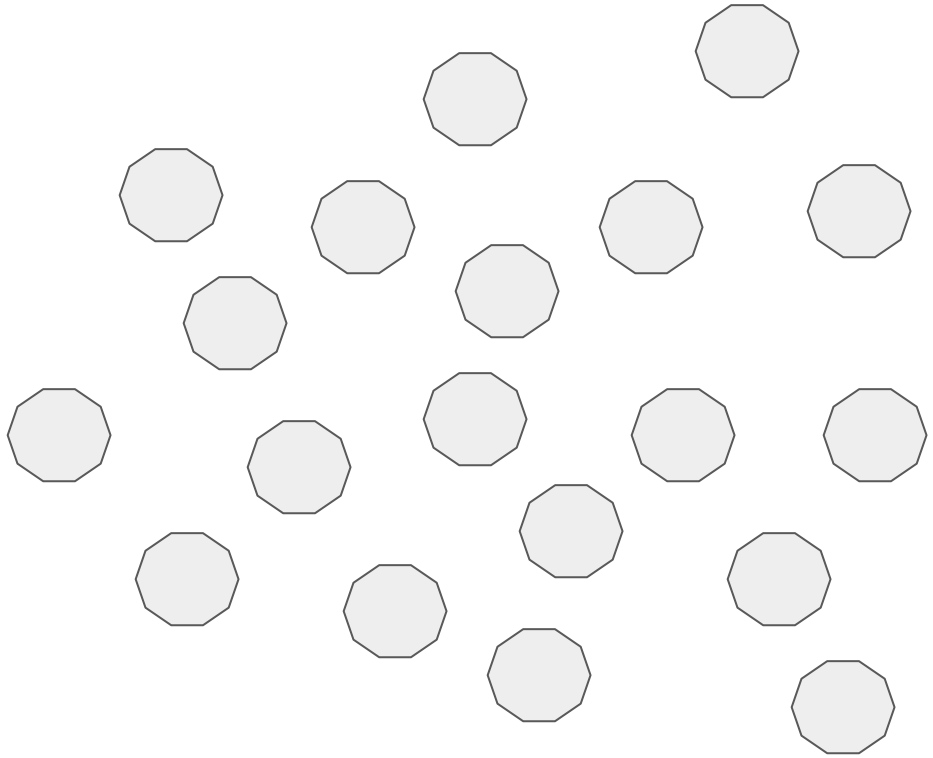
Modules



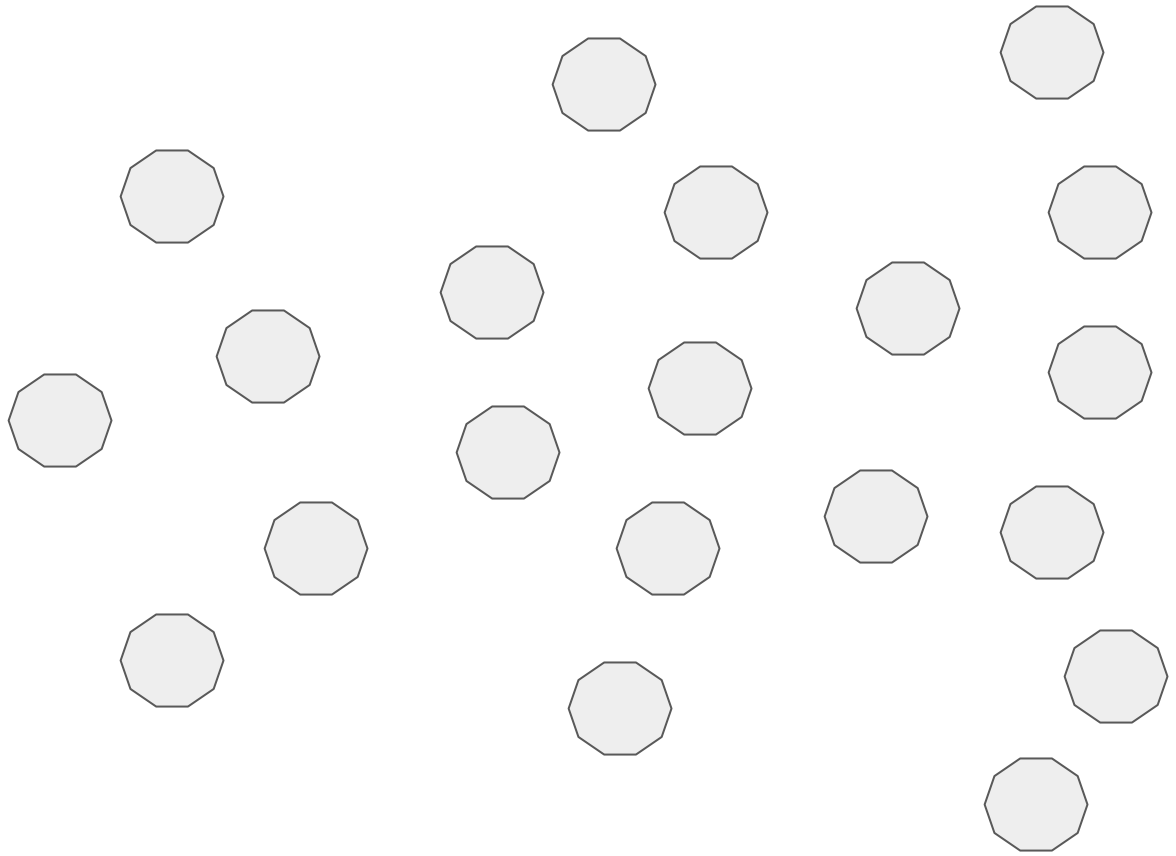
Microservices



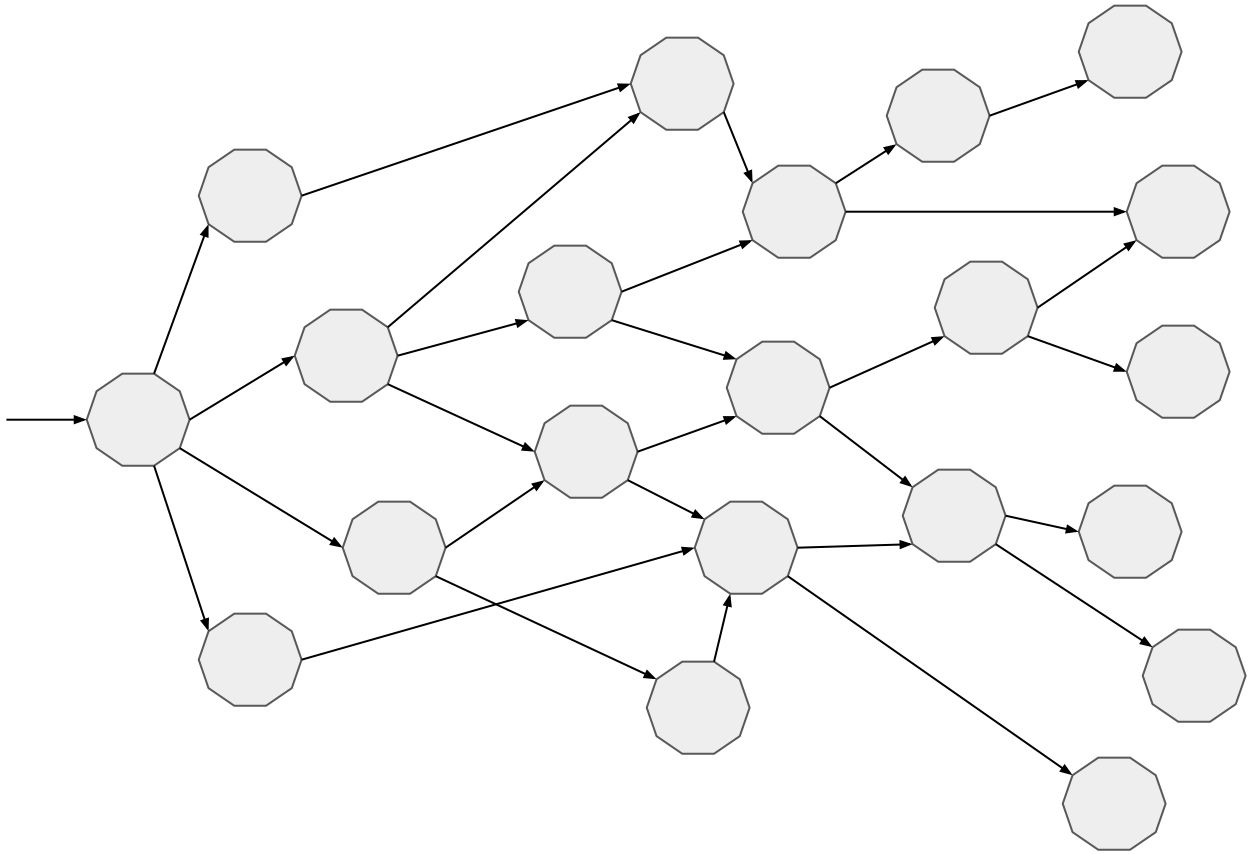
Microservices



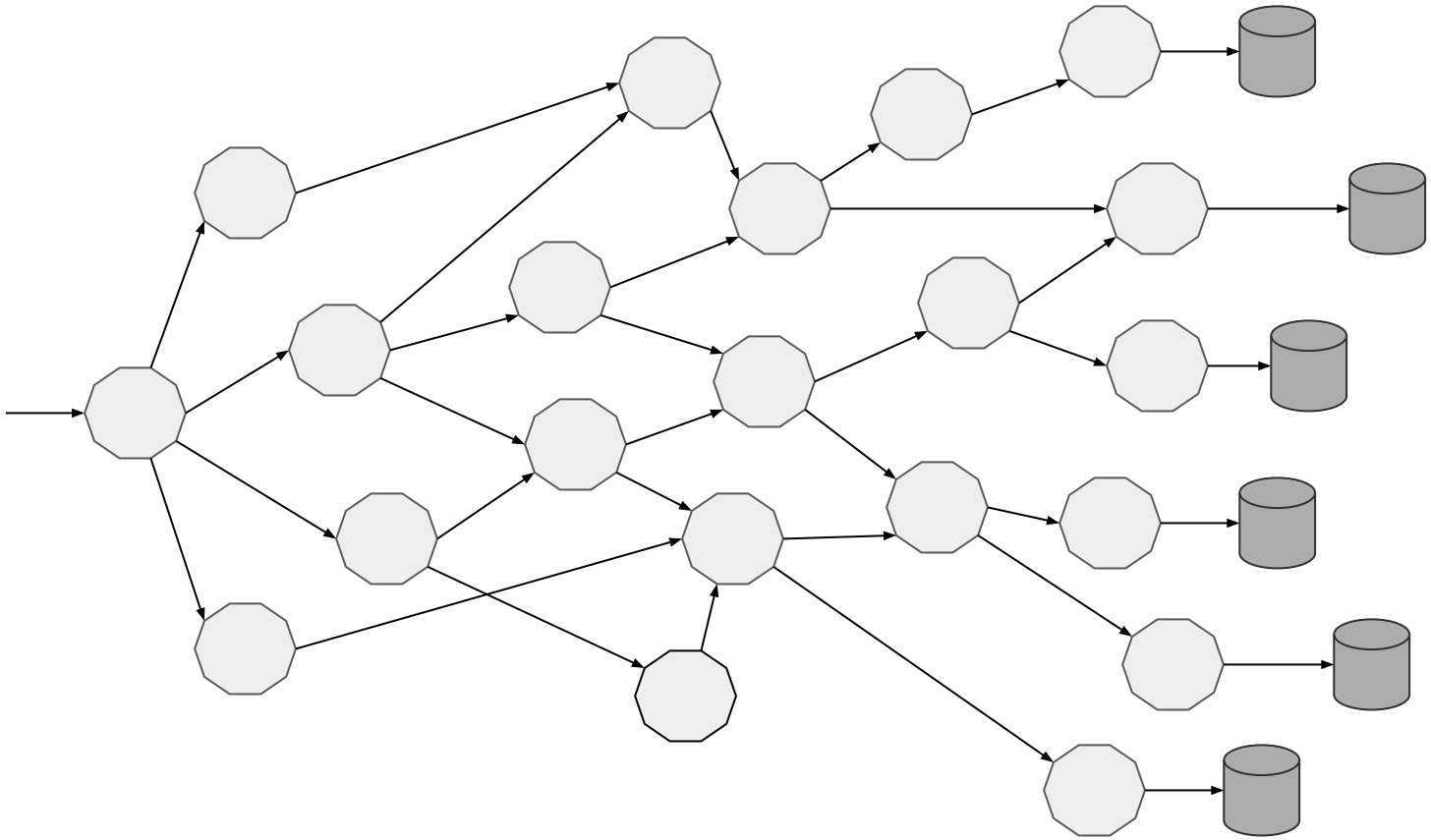
Microservices



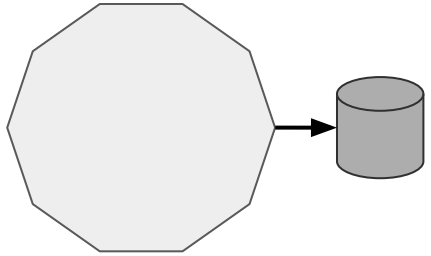
Network of Services



Microservices own their Data

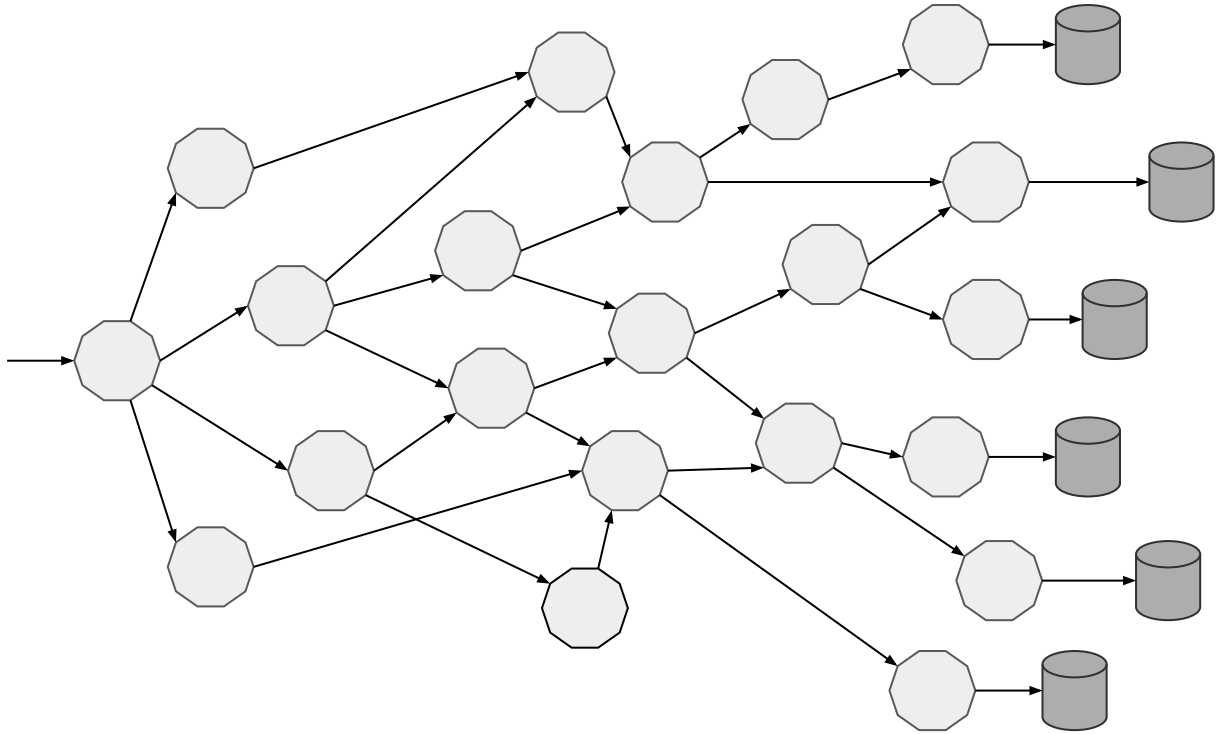


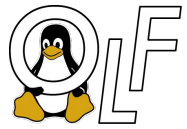
Old School



Love Thy Mono

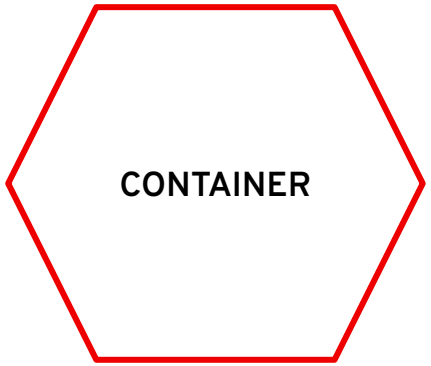
New School

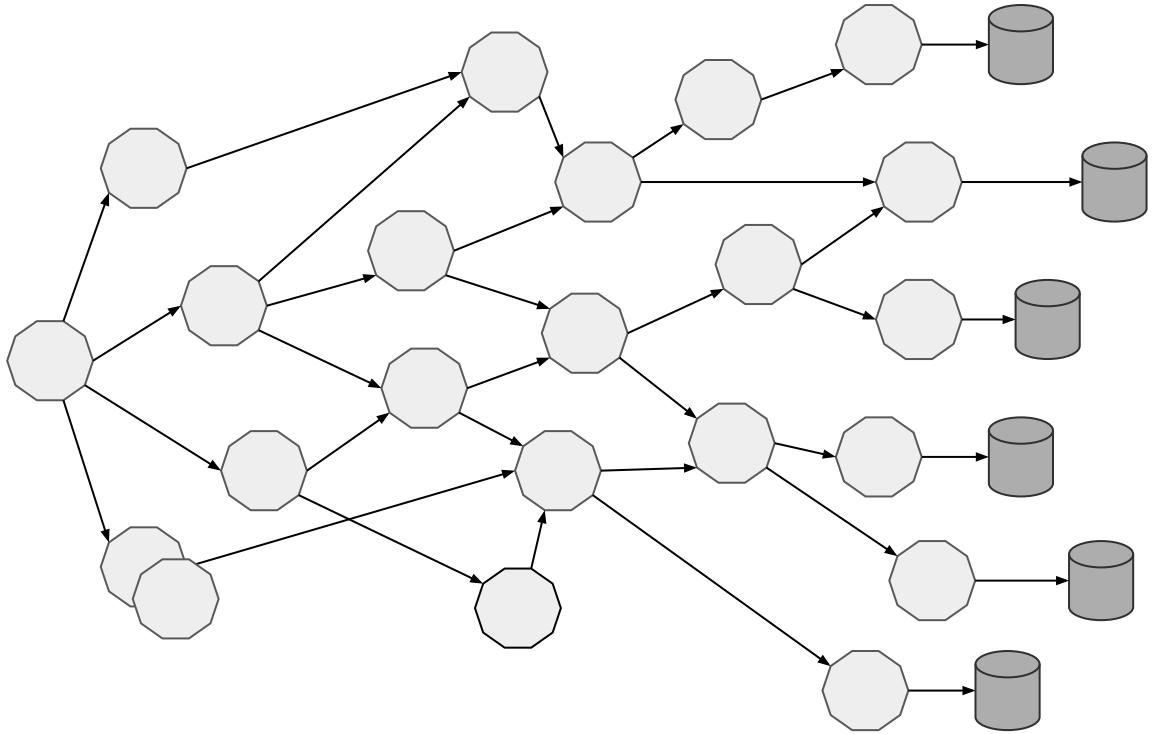


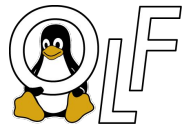


Red Hat

Containers

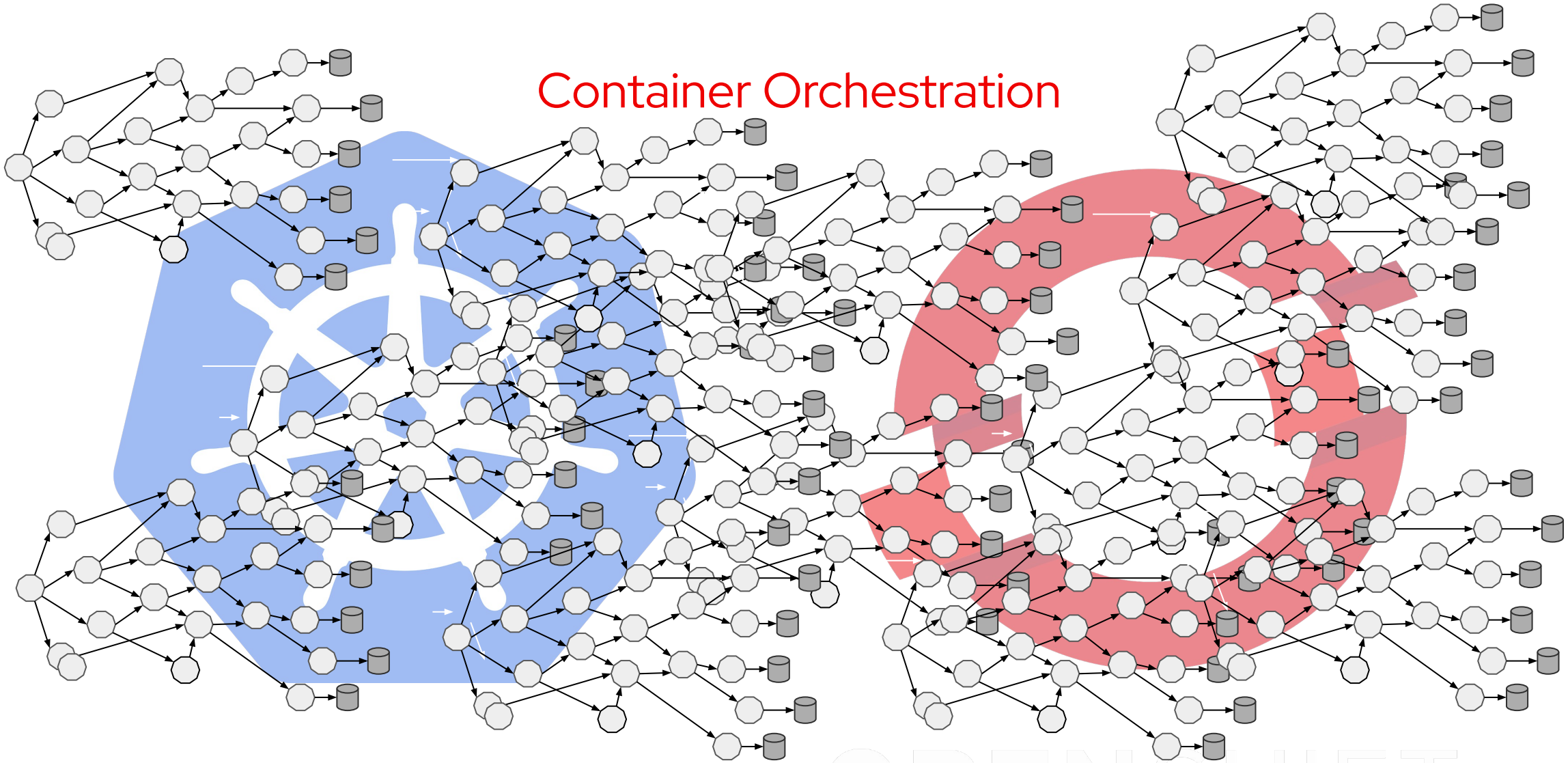






Red Hat

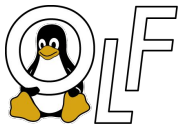
Container Orchestration



Container Technology Introduction

What is a container?

- A unit of software that contains code and all its dependencies so that the application can be run in the same way in different computational environments.
- **Operating system** - A process in a sandbox
- **System Administrator** - Portability for an app and its runtime dependencies
- **Developer** - Universal Application Packager



Red Hat

The History of Containers

1979: UNIX®
CHROOT SYSCALL ADDED

2000:
JAILS ADDED TO FREEBSD

2001:
LINUX -VSERVER PROJECT

2003:
SELINUX ADDED TO LINUX MAINLINE

2005:
FULL RELEASE OF SOLARIS ZONES

2006:
PROCESS CONFINEMENT

2007:
GPC RENAMED CGROUPS

2008:
LINUX CONTAINER PROJECT (LXC)

2008:
KERNEL & USER NAMESPACES

2013:
DOTCLOUD PYCON LIGHTNING TALK

2013:
DOTCLOUD BECOMES DOCKER

2013:
RED HAT ENTERPRISE LINUX

2014:
GOOGLE KUBERNETES

1979 2000 2005 2010

https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

Red Hat

CONTAINER INNOVATION ...

2015:
OPENSHIFT by Red Hat
RED HAT CONTAINER PLATFORM 3.0

2015:
STANDARDS VIA OCI AND CNCF

2015:
Tectonic Announced

2016:
Skopeo project launched under the name OCID

2016:
CRI-O project launched under the name OCID

2016:
Docker engine 1.12 adds swarm

2016:
Containerd project launched

2017:
Moby project Announced

2017:
V1.0 of image & runtime spec

2017:
Buildah released and ships in RHEL

2017:
Docker includes the new containerd

2017:
Kata merges Clear & RunV projects

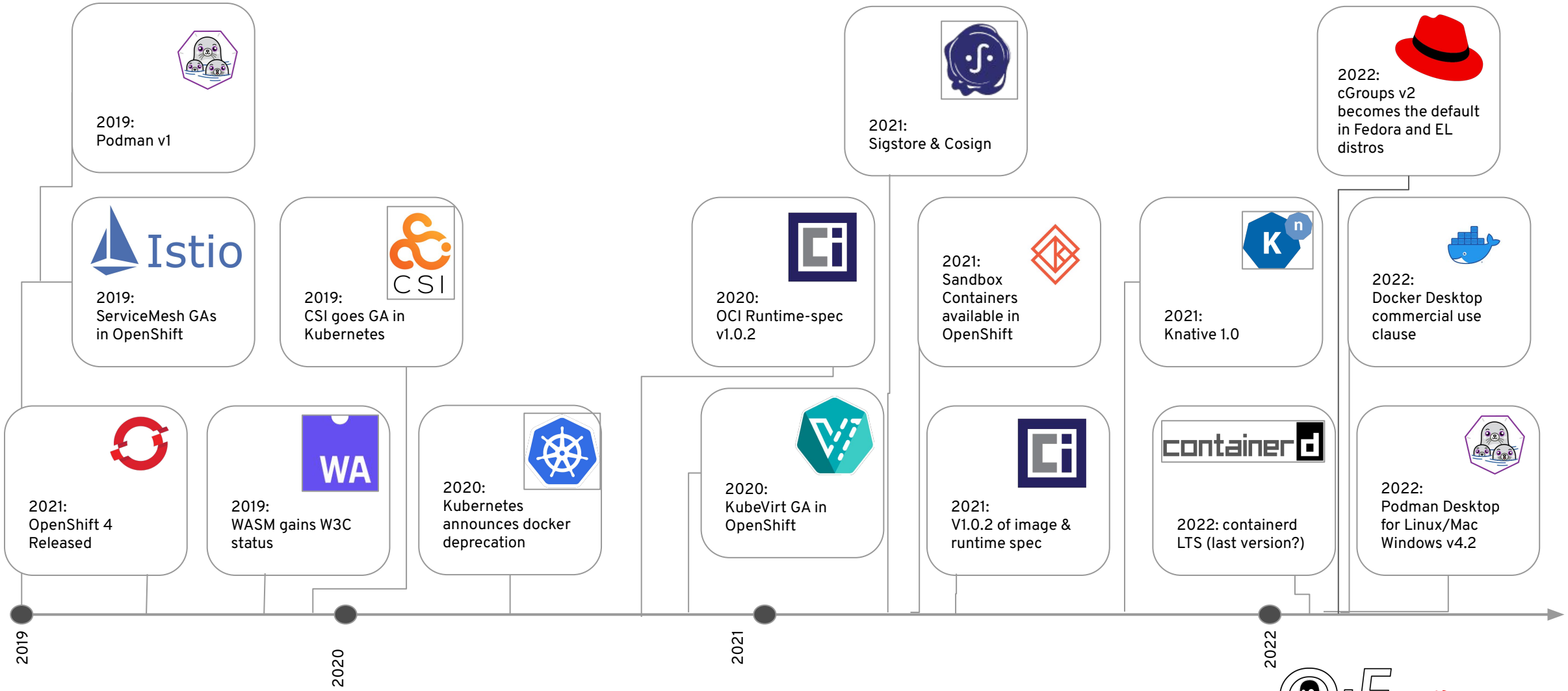
2018:
V1.0 of distribution spec

2018:
CRI-O is GA and powers OpenShfit Online

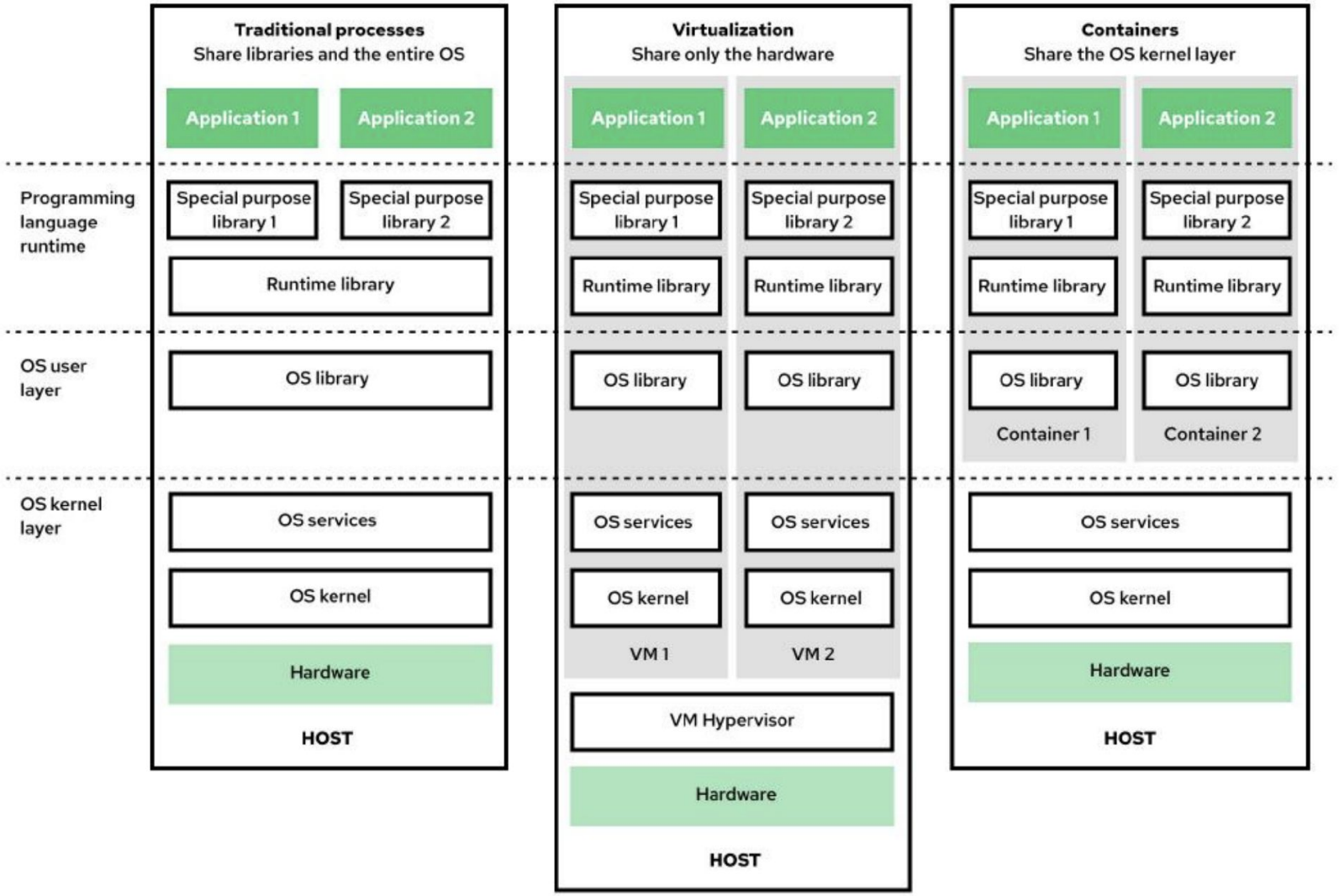
2018:
Podman released and ships in RHEL

2015 2016 2017 2018

CONTAINER INNOVATION IS NOT FINISHED

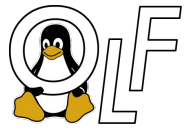


What is a container?



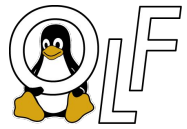
How do we make one?

- It all starts with a process
- The we wrap some boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - Limit resource usage
  - Prioritize or de-prioritize CPU share and Disk I/O
  - Control freezing, stopping, starting and checkpointing
  - Account for usage of resources managed above



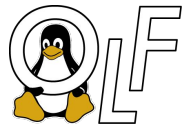
Namespaces

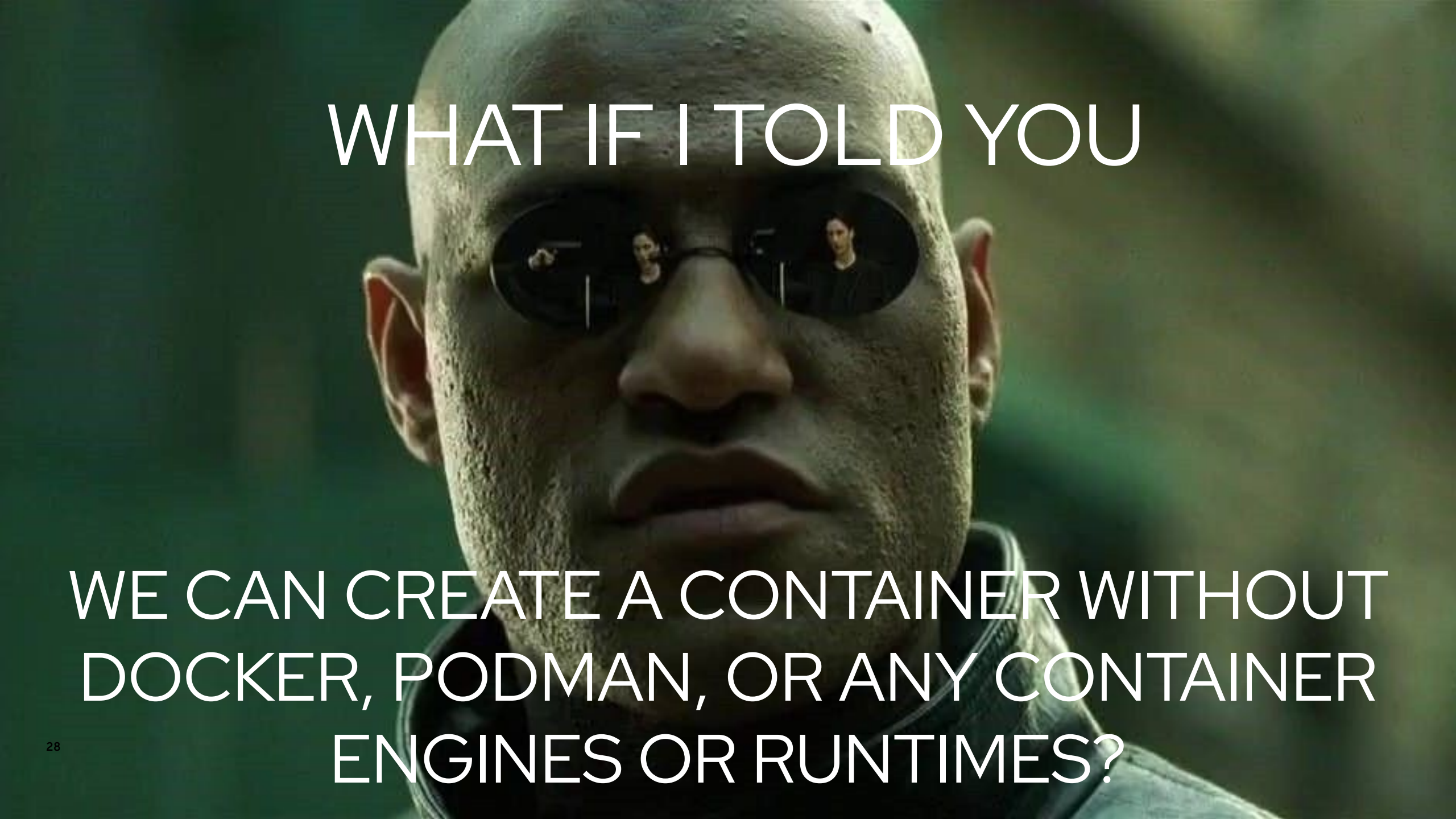
- It all starts with a process
- The we wrap some boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - Limit resource usage
  - Prioritize or de-prioritize CPU share and Disk I/O
  - Control freezing, stopping, starting and checkpointing
  - Account for usage of resources managed above



CGroups

- It all starts with a process
- The we wrap some boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - **Limit** resource usage
  - **Prioritize** or de-prioritize CPU share and Disk I/O
  - **Control** freezing, stopping, starting and checkpointing
  - **Account** for usage of resources managed above



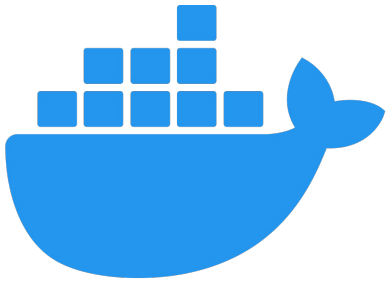

WHAT IF I TOLD YOU
WE CAN CREATE A CONTAINER WITHOUT DOCKER, PODMAN, OR ANY CONTAINER ENGINES OR RUNTIMES?

Technically Speaking....

```
# Creates new linux namespaces for a process (shell) and starts it
unshare --mount --uts --ipc --net --pid --fork --user --map-root-user chroot ${CONTAINER_ROOT} /bin/sh

# mount up the kernel stuff
mount -t proc none /proc
mount -t sysfs none /sys
mount -t tmpfs none /tmp

# OUTSIDE THE CONTAINER!!!
sudo ip link add vethlocal type veth  peer name vethNS
sudo ip link set vethlocal up
sudo ip link set vethNS up
sudo ps -ef |grep '/bin/sh'
sudo ip link set vethNS netns <pid of /bin/sh>

# Back in the container
ip link
```

Bonus Lab - If you have time today!

Complexity begets docker



docker

- 2013 - A Star is born!
- Makes all of this hard stuff easy for developers
- New constructs to build, deploy, store, run containers
    - Container Host - brings the kernel to the party
    - Container Runtime - Interface to the kernel
    - Container Engine - Interface to the operator (API/CLI)

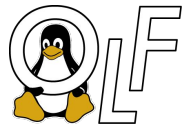
rkt

kata containers

Firecracker

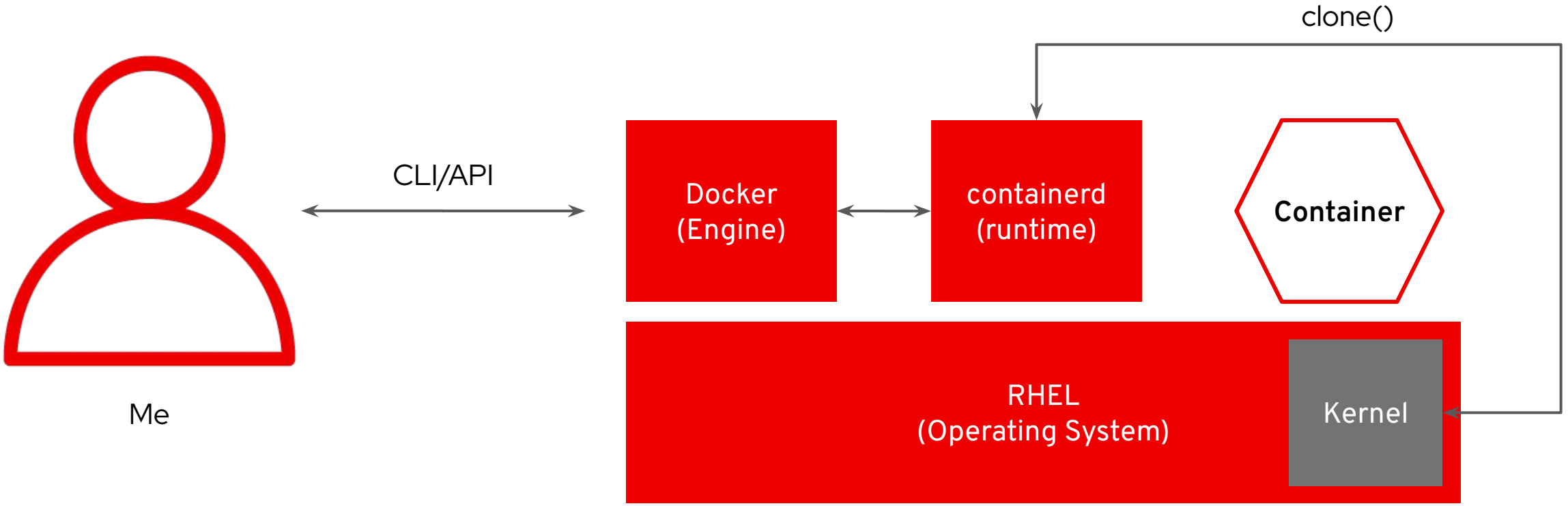
gVisor

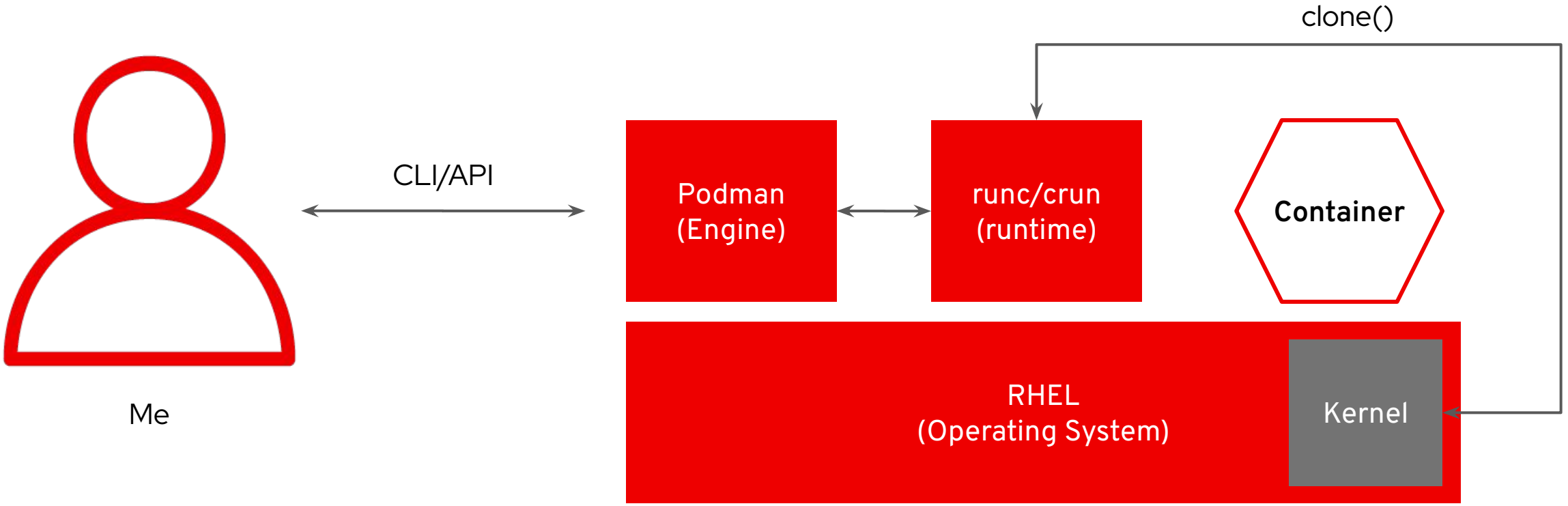
cri-o

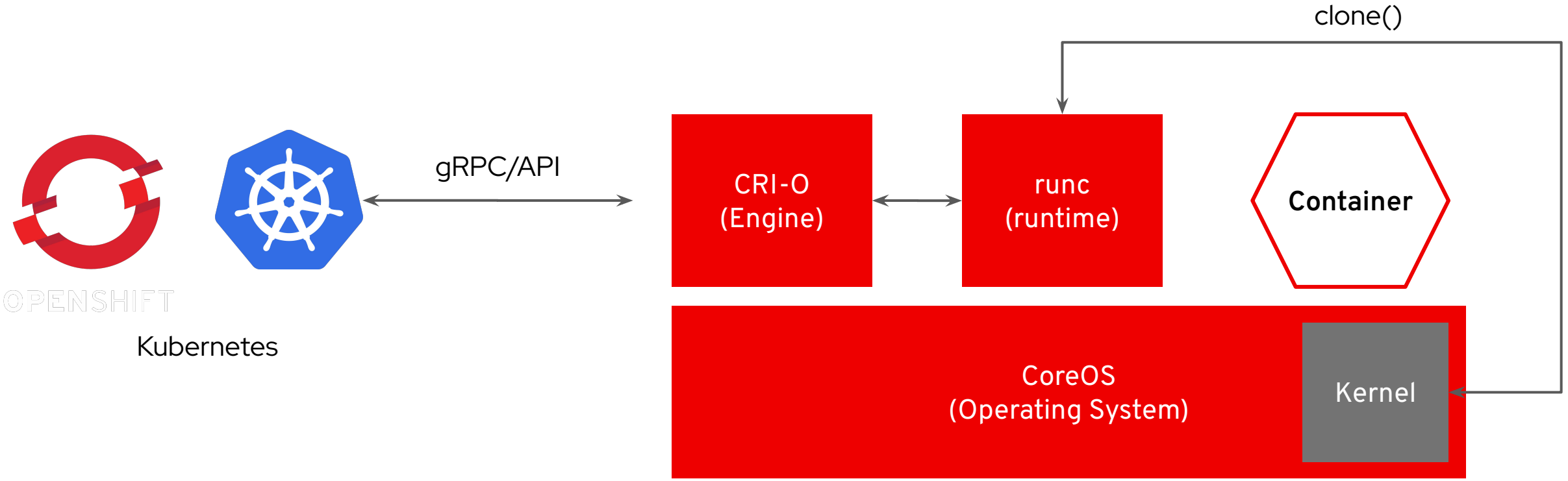


Red Hat

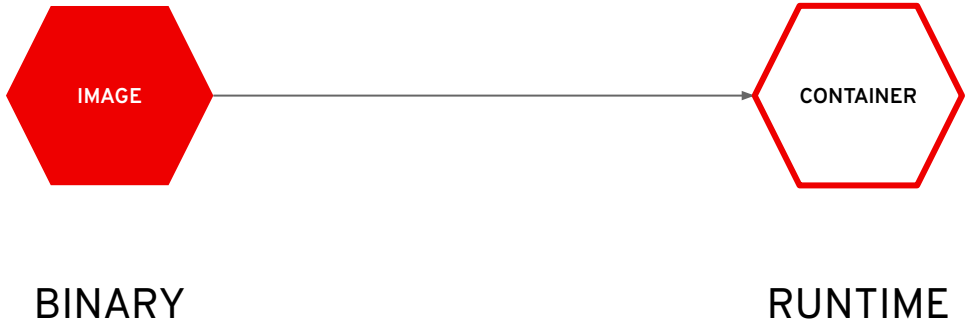

Google
docker architecture
Sign in
Docker Documentation
Docker overview | Docker Documentation
Whizlabs
Docker Archi
Geekflare
Docker Architecture and its Components ...
clone()
eduCBA
Docker Architecture | Learn the Objects ...
Docker Architecture & Components
Container
Runtir
Kernel
eduCBA
Docker Architecture | Learn the Objects ...
Simplilearn
Docker Architect
DZone
Docker Containers and Kubernetes: An ...
ResearchGate
Docker architecture ...
Docker Client
Docker Host/Server
Docker Registry/Hub
Docker Daemon
31
LINUX OS
CONTAINER
CONTAINER
NODE.JS
NGINX
ALPINE
BUSYBOX
DOCKER ENGINE
KERNEL
Docker Engine Architecture
Docker Engine - Enterprise
Docker Engine - Community
containerd
runc


clone()
CLI/API
Docker
(Engine)
containerd
(runtime)
Container
Me
RHEL
(Operating System)
Kernel

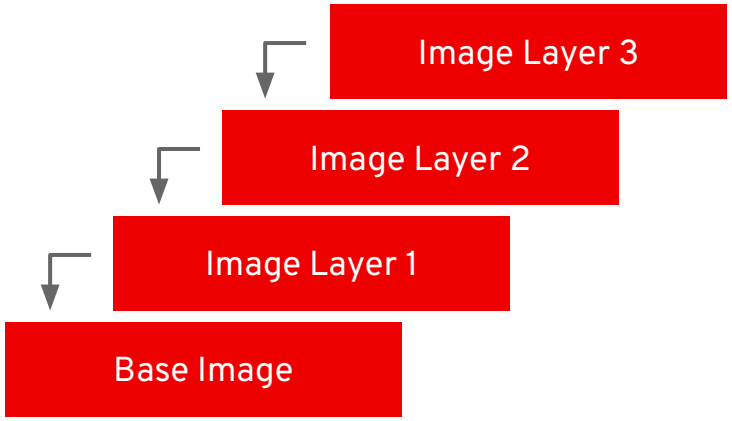

clone()
CLI/API
Podman
(Engine)
runc/crun
(runtime)
Container
Me
RHEL
(Operating System)
Kernel


clone()
OPENSHIFT
Kubernetes
gRPC/API
CRI-O
(Engine)
runc
(runtime)
Container
CoreOS
(Operating System)
Kernel

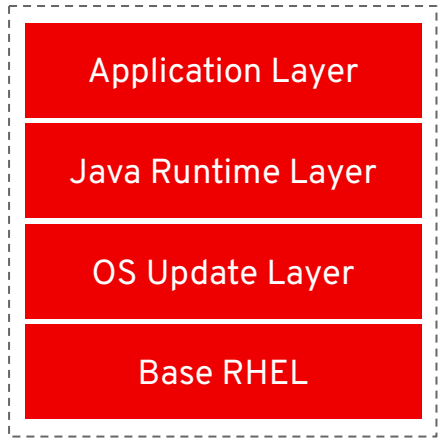
containers are created from container images



container images are structured in layers



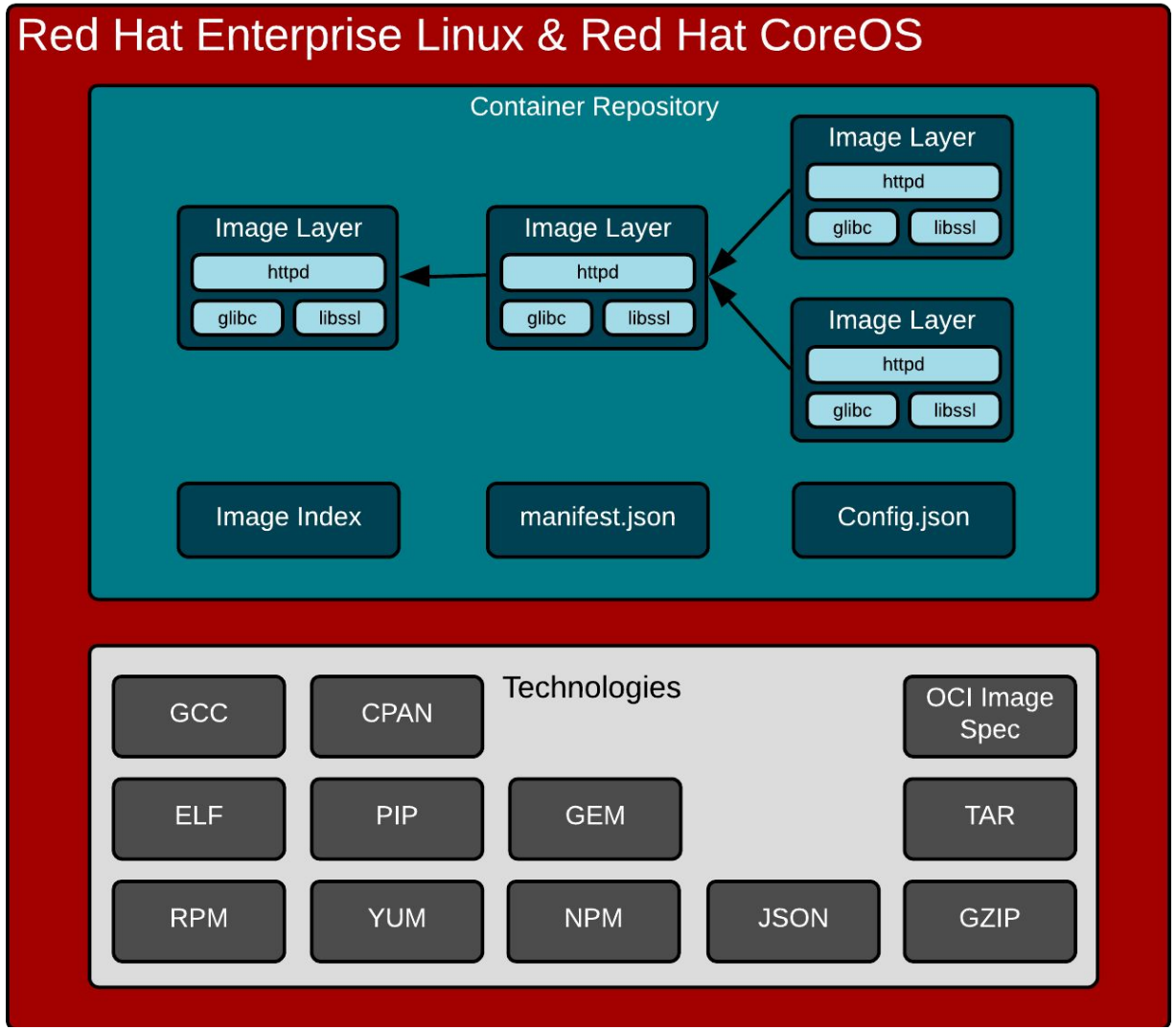
Container Image Layers



Example Container Image

container image parts (OCI Spec)

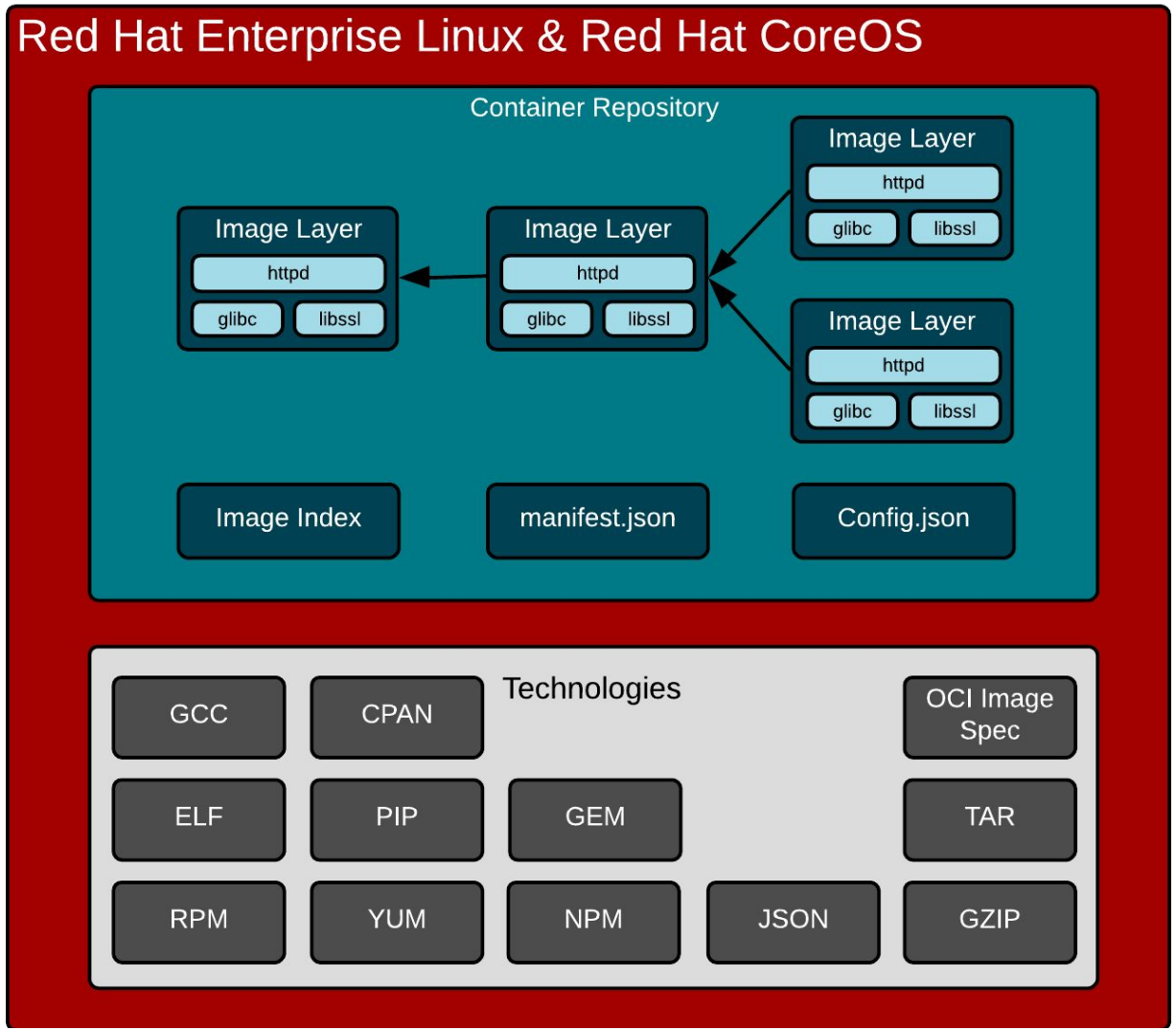
- Image layers
  - The Bits
- Image Index
  - Manifest of manifests
  - Multiple architectures
- Manifest.json
  - Unique hashes of layers
- Config.json
  - Metadata about the image
- https://github.com/opencontainers/runtime-spec/blob/main/spec.md
- V1.1.0 was released in July!



Courtesy of @fatherlinux - Container Internals

container image compatibility

- OCI Standard guarantees the engine can accept the image
- Nothing guarantees it is compatible!
- red.ht/2X9oq0i



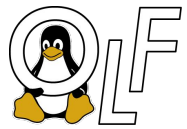
Courtesy of @fatherlinux - Container Internals

anatomy of a Dockerfile/Containerfile

```
FROM registry.access.redhat.com/ubi8/ubi
ENV foo=text
RUN dnf install -y java-11-openjdk
ADD my-app.jar /home/my-app.jar
EXPOSE 8080
CMD java -jar /home/my-app.jar
```

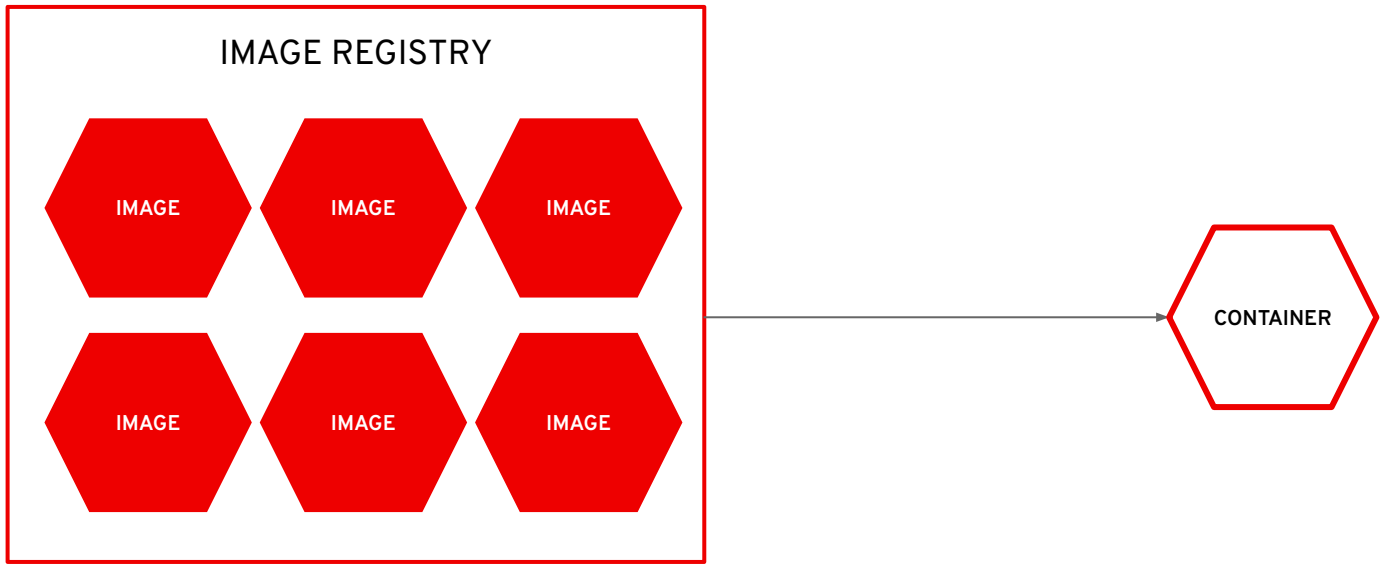
1. Inherit from a base image
2. Parameters as environment variables
3. Install dependencies (tooling from base image)
4. Add your app as a new Layer
5. Expose the port your app will use
6. Run the app

Example for Java app

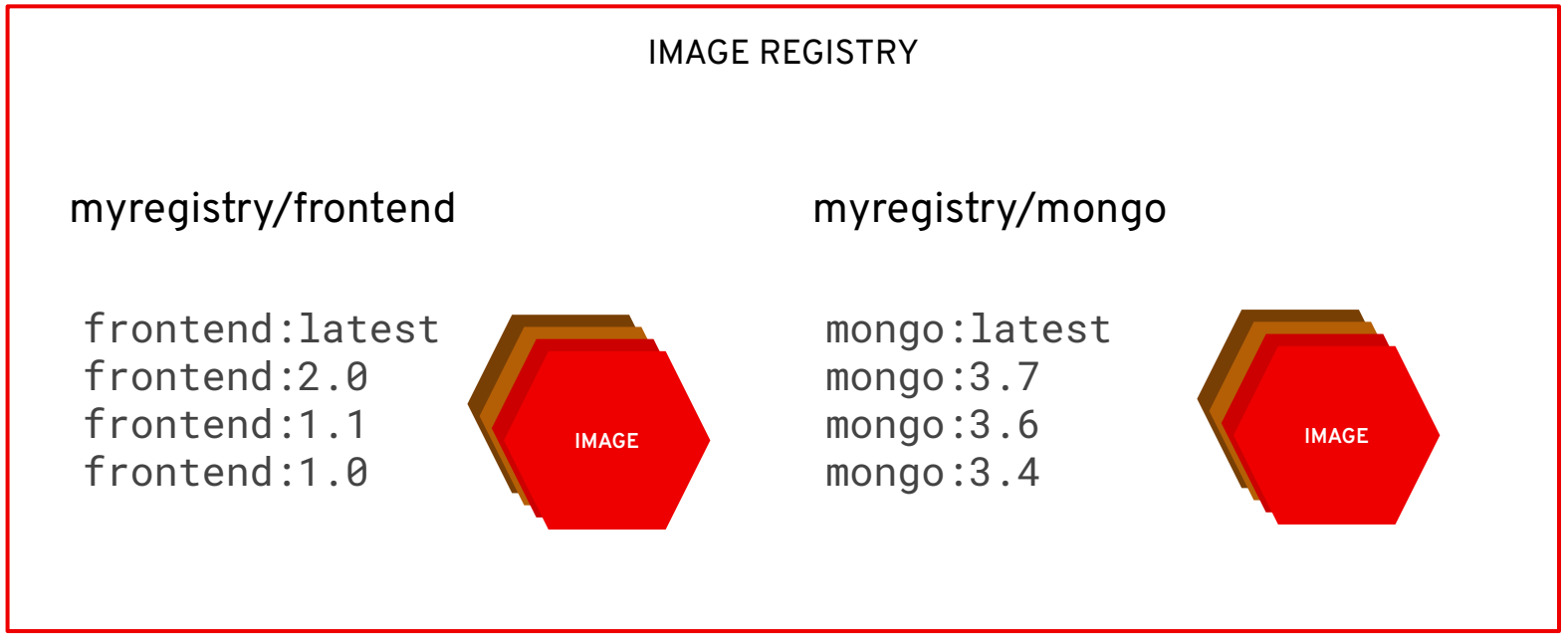


Red Hat

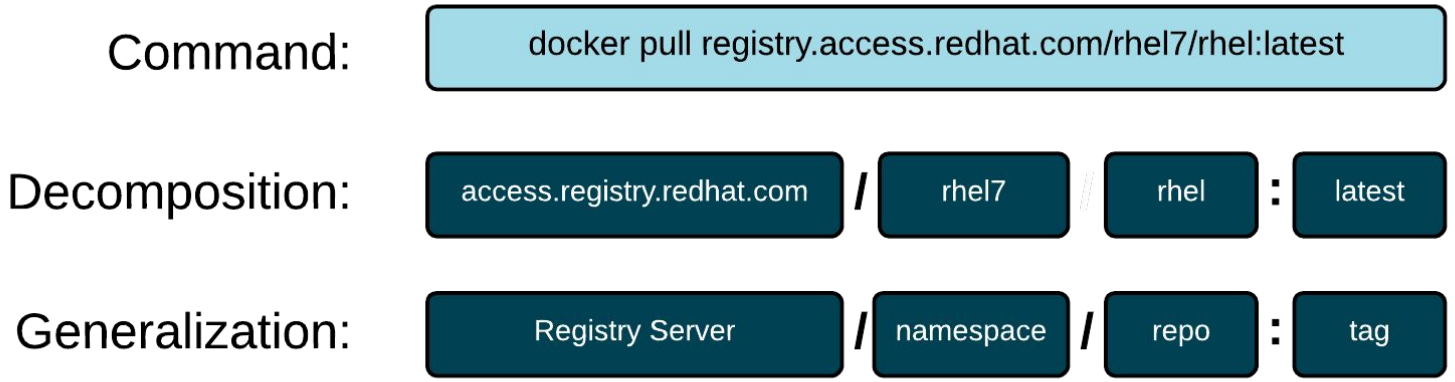
container images are stored in an image registry



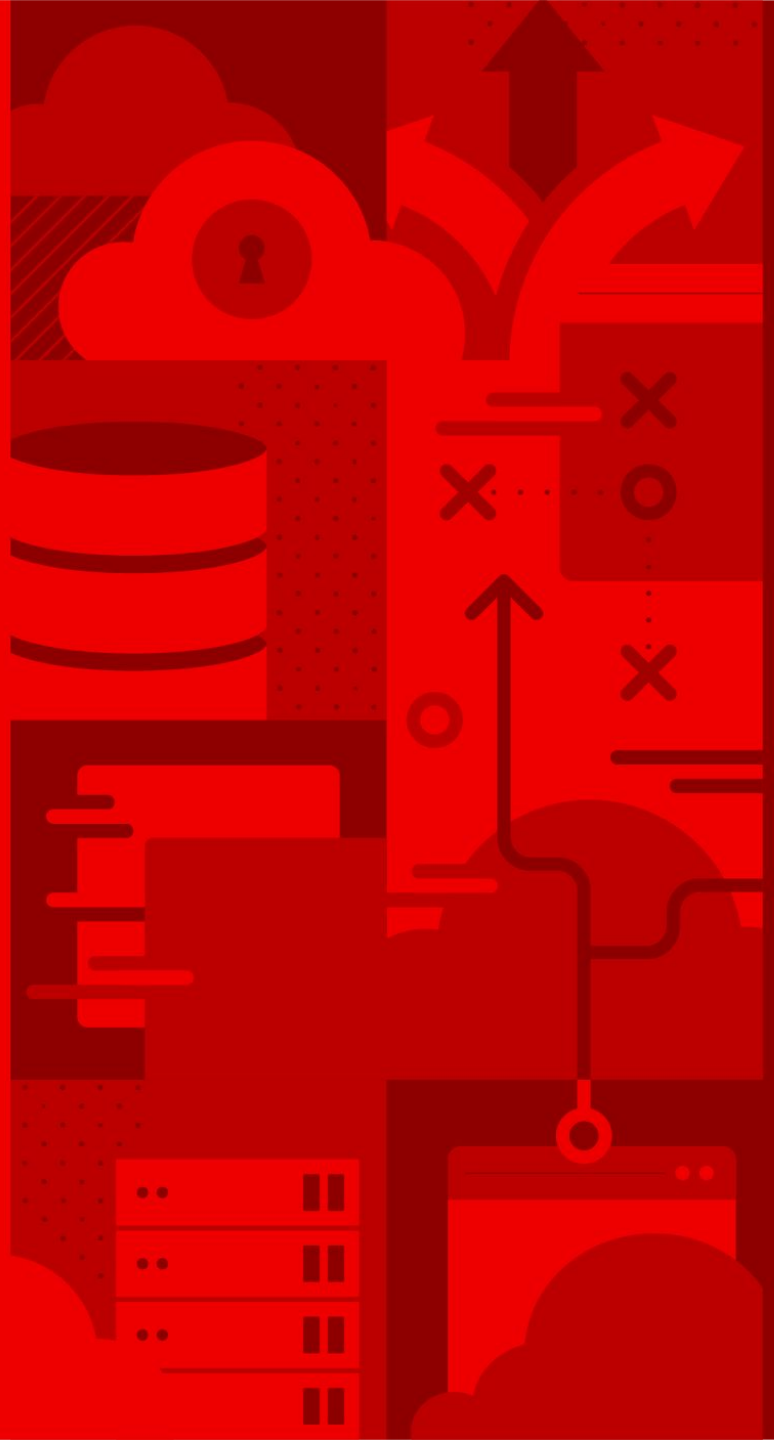
an image repository contains all versions of an image in the image registry



Registry Command Syntax



Courtesy of @fatherlinux - Container Internals

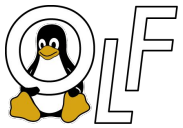


What is Kubernetes?

What is Kubernetes?

- “An open-source system for automating deployment, scaling, and management of containerized applications” (https://kubernetes.io/)
- Layman terms – Kubernetes is a scheduler for containers (and other things!)
- Abstracts away the details of infrastructure
- “Kubernetes is the new kernel.” – Jessie Frazelle

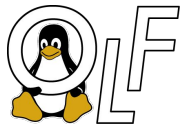
An open source orchestration system for managing containerized workloads across a cluster of nodes.



Red Hat

History of Kubernetes

- Came from Google (announced in 2014)
- Influenced by Google's own cluster manager (Borg/Omega)
- Greek word for Helmsman or Governor
  - The captain of the container ship
- Written in GOLANG
- Google and the Linux foundation formed the CNCF (Cloud Native Computing Foundation) and donated the first project (2015)



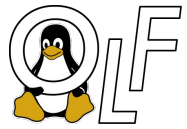
Red Hat

Is Kubernetes the right tool for the job?

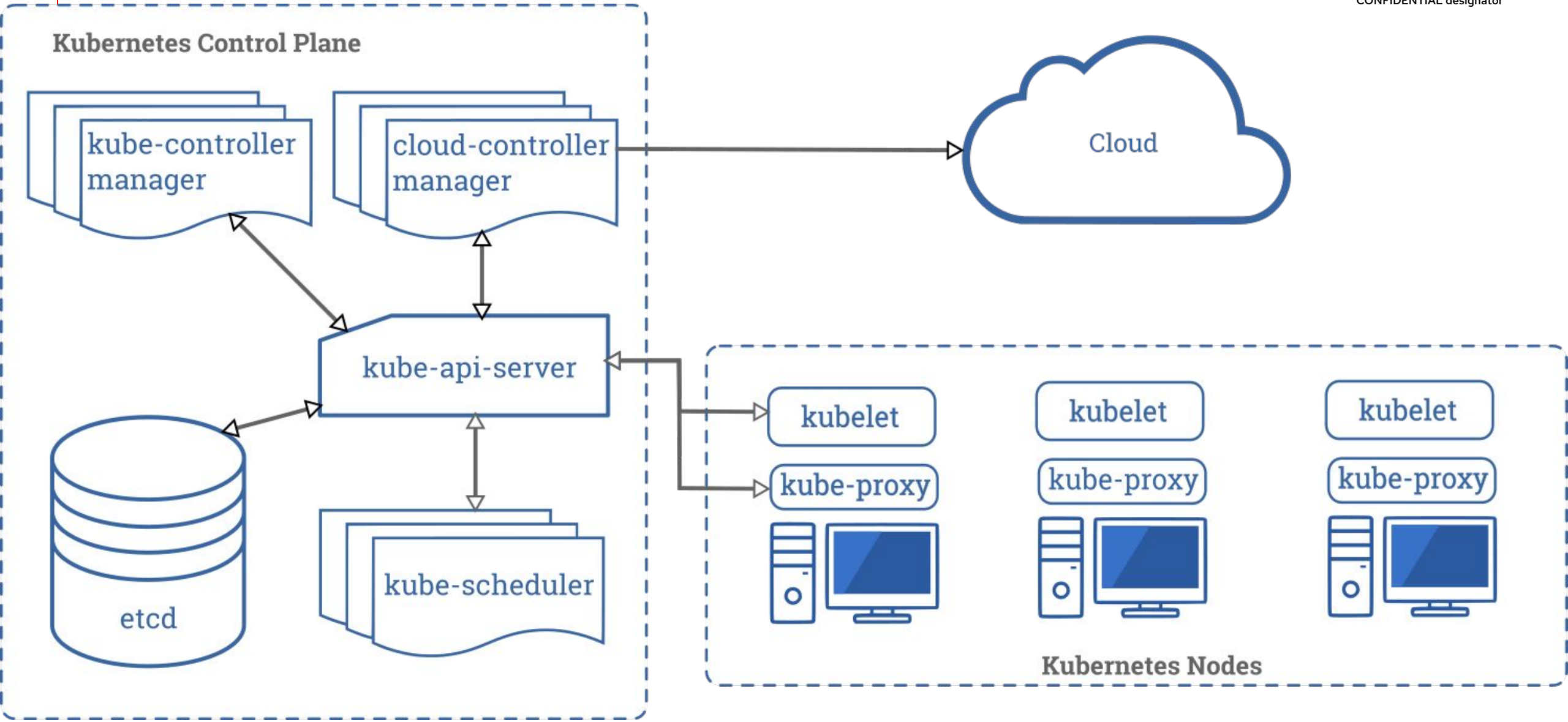
“You should be using tools and platforms that enable you to get business logic to production as fast as safe and as secure as possible”

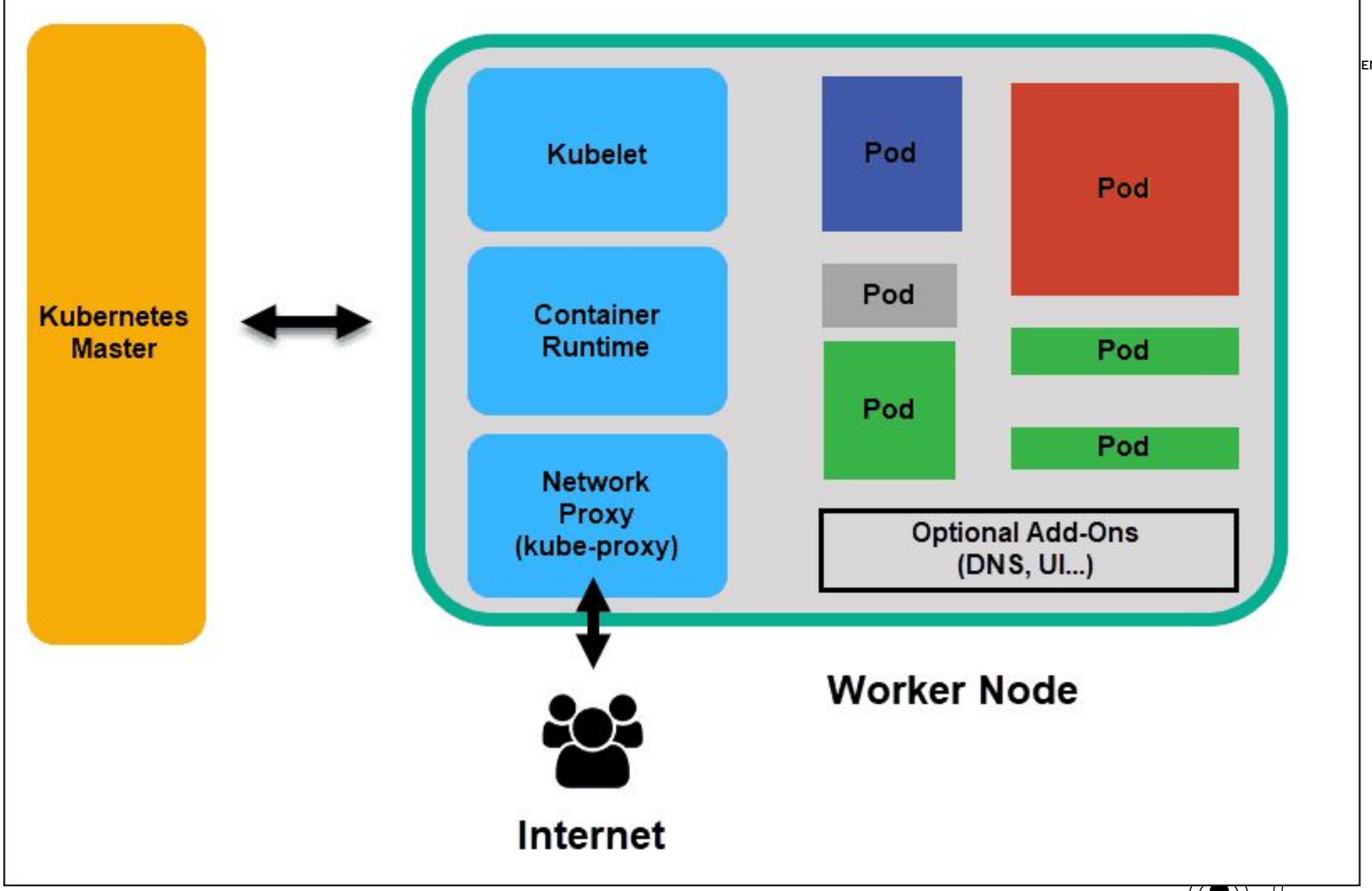
- Me

Hint: Kubernetes is a great tool for this!

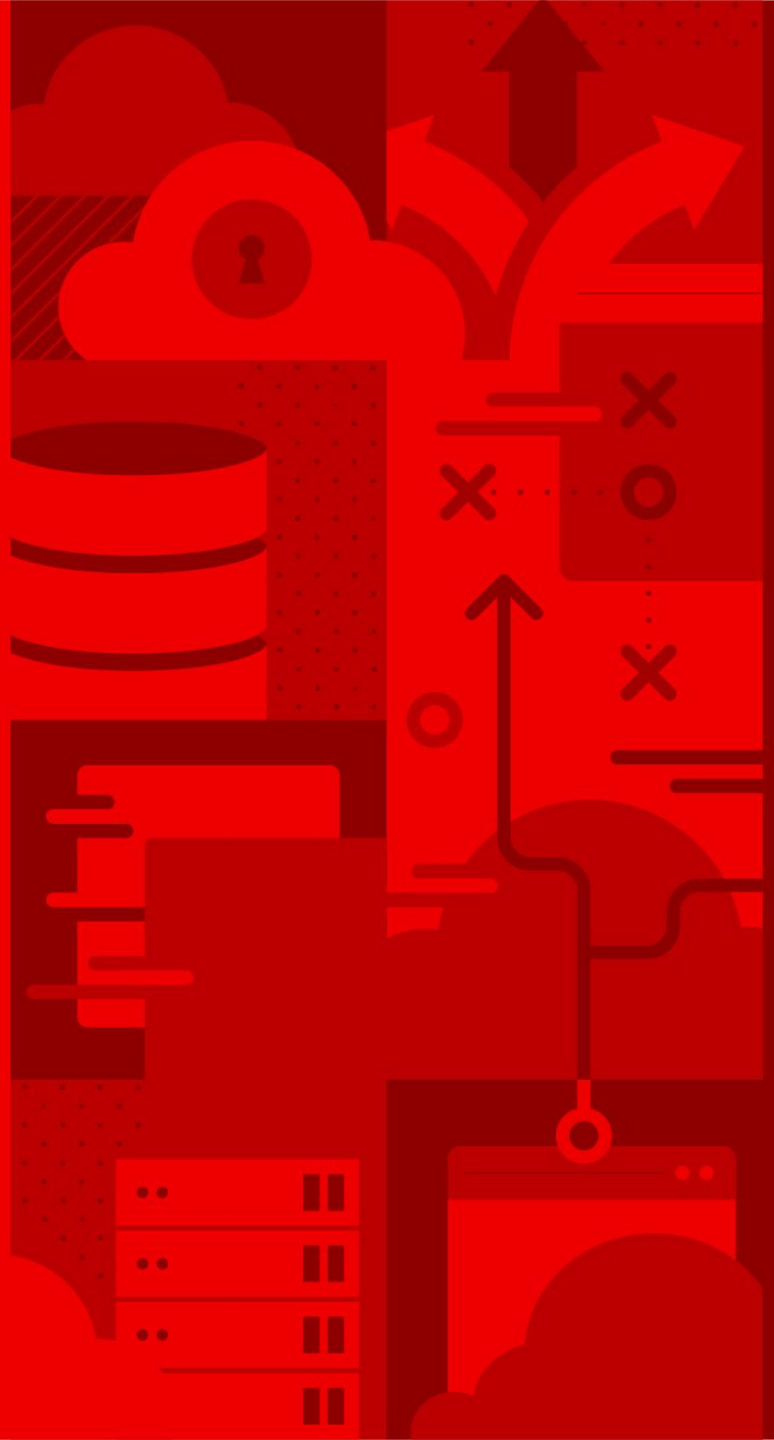


Red Hat


Kubernetes Control Plane
kube-controller manager
cloud-controller manager
Cloud
kube-api-server
kubelet
kube-proxy
kubelet
kube-proxy
kubelet
kube-proxy
etcd
kube-scheduler
Kubernetes Nodes



https://phoenixnap.com/kb/wp-content/uploads/2019/10/components-worker-node-kubernetes.png



Kubernetes basic building blocks

Red Hat

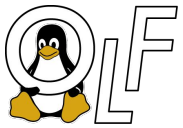
Declarative API - The unsung hero

The Control Plane's Declarative API is the most underrated and important part of Kubernetes

- Inception: Orchestrator of containers
- ~~Today~~Yesterday: Orchestrator of containers and container adjacent things
- ~~Tomorrow~~Today: Orchestrator of Orchestrators and Clouds
- Tomorrow: Orchestrator of WebAssembly (WASM/WASI) / Anything!

"Kubernetes is Infrastructure as Data" - Kelsey Hightower of Google

"Containers is just how it started, Kubernetes is bound for much more" - Bassam Tabbara of Upbound



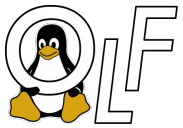
Red Hat

Object Model

All Objects have:

- GVK (Group/Version/Kind)
- Metadata
- “spec” - This is what I want to happen
- “status” - This is what the actual state is

**Objects are described in YAML and converted to JSON when sent to the API server.*



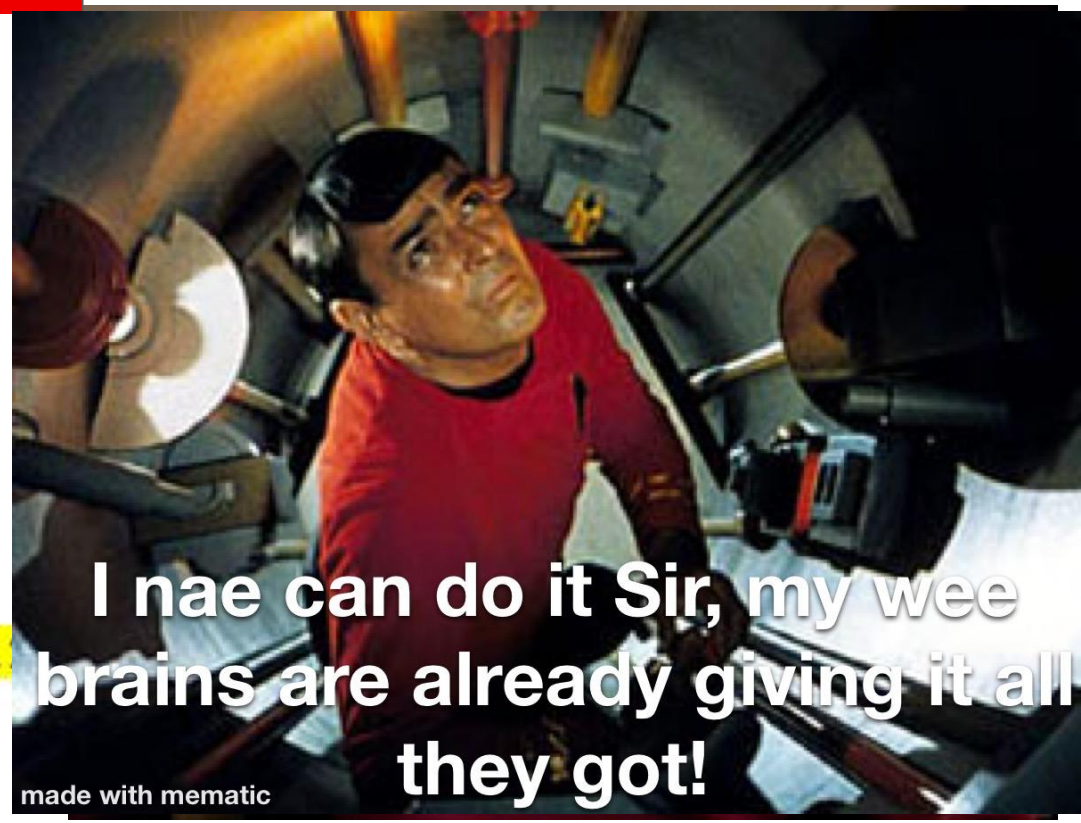
Red Hat

Resource definition

```
 status:
     availableReplicas: 2
     conditions:
     - lastTransitionTi
status:
  containerStatuses:
  - image: nginx:1.14.22
    imageID: ""
    lastState: {}
    name: nginx
    ready: false
    restartCount: 0
    state:
      waiting:
        message: Back-off pulling image "nginx:1.14.
        reason: ImagePullBackOff
     replicas: 2
     updatedReplicas: 2
```

Group: apps Version: v1beta

Kind: Deployment

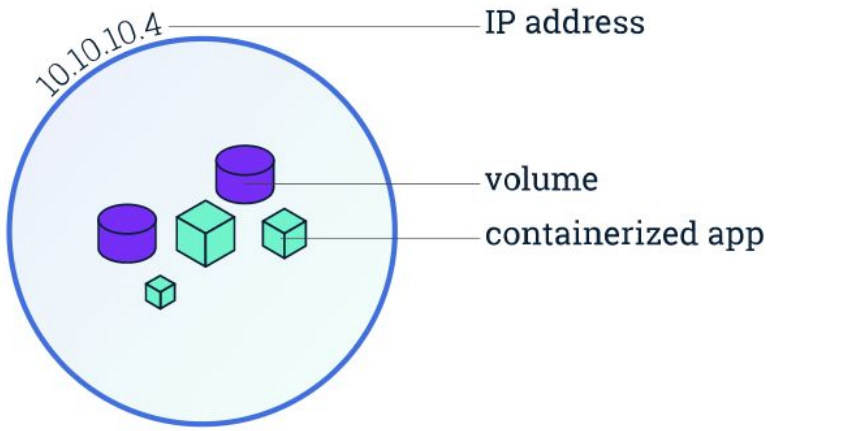


Pod

A pod is the atomic unit of an application in Kubernetes.

One (or more) containers that share:

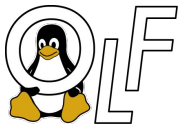
- Networking (IP address)
- Linux namespace
- Storage
- Memory



Volumes / Configmaps / Secrets

Volumes, Configmaps, and Secrets are used to insert data into pods at runtime

- **Volumes** are essentially just a directory on disk
  - Ephemeral or Persistent (PV/PVC)
- **Configmaps** are a declarative way to store and insert configuration data in to pods as volumes
- **Secrets** are just like configmaps only they store their data in RAM and can be encrypted and obfuscated from the host.



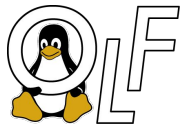
Red Hat

Service

A networking construct to abstract the Pod

3 basic types of Services:

- ClusterIP – basic load balancing (internal to cluster only)
- NodePort – Translates ClusterIP:port to NodeIP:port
- LoadBalancer – integrates with 3rd party / external LB



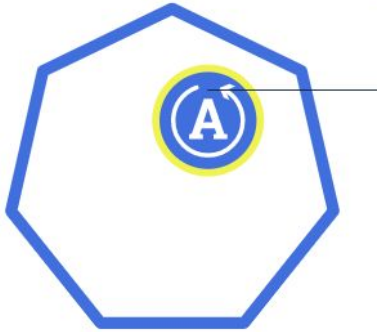
Red Hat

ReplicaSets / Deployments

ReplicaSets - Scales Pods...That's it...

Deployments - Declarative updater for pods and ReplicaSets

- What is running and where?
- What services and resources are available to it?
- Policies for how things should behave (IFTTT logic)
- Rollback / Scaleup / Rollout definitions



Stateful Sets / Daemon Sets / Cron Jobs

Stateful Sets

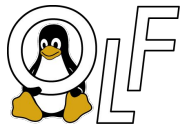
- A special deployment type for stateful workloads
- Stable and ordered constructs
- Volumes scale out with pods

Daemon Sets

- A special deployment type that runs on a defined set of nodes
- As nodes get added that match, the pods scale
- Useful for infrastructure services

Cron Jobs

- Batch processing
- Serverless (like)
- Schedule **when** you want to run a pod

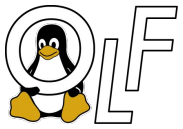


Red Hat

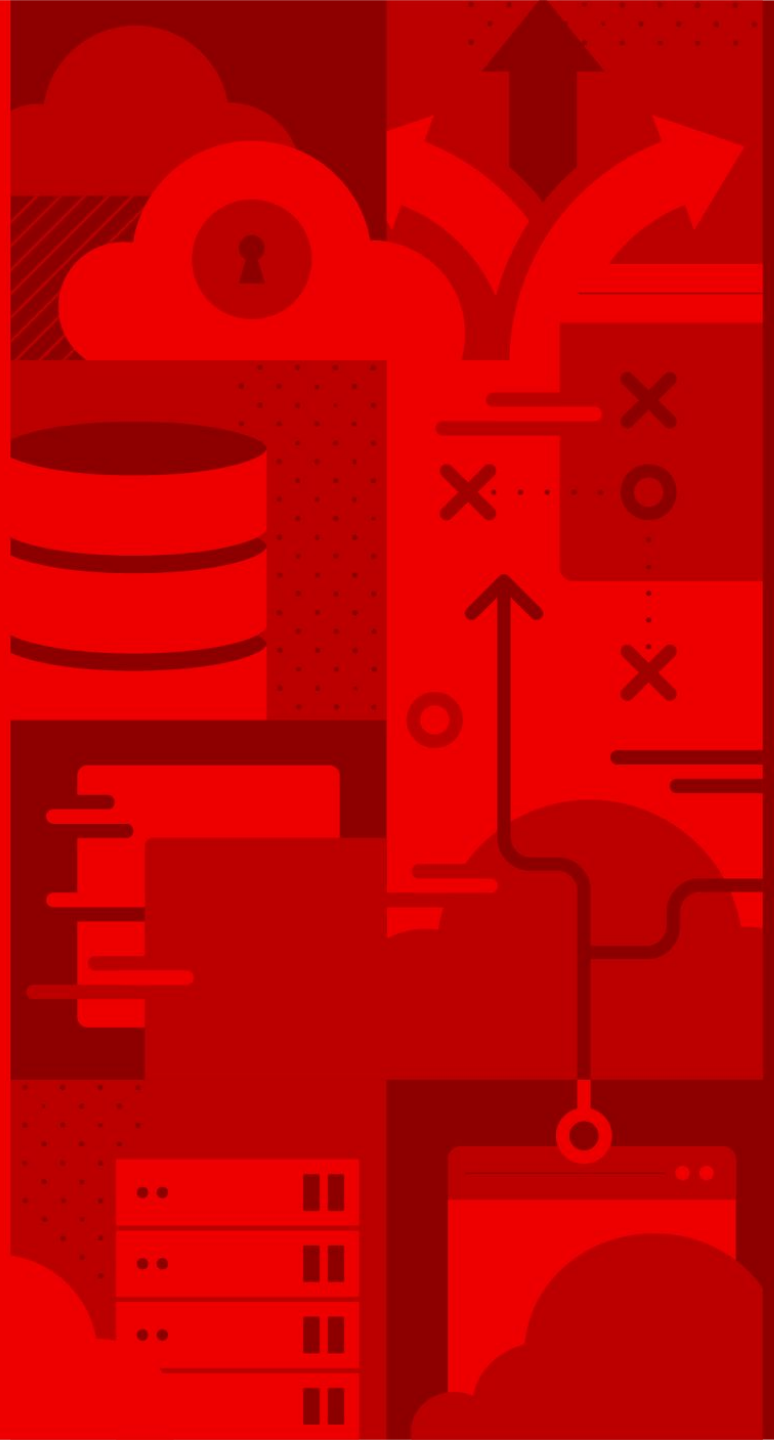
Extending Kubernetes: Custom Resource Definitions

Custom Resource Definitions

- Extend the Kubernetes API with your own GVK
- Inherit the rich high-level API abstractions and data model
- Your resources become first class citizens in the Kubernetes ecosystem
- Operator pattern pairs custom resources with custom controllers to program specific knowledge of your applications into the Kubernetes ecosystem.



Red Hat



Workshop

Please choose the lab code for this session, enter the activation key, and your e-mail address then click **Submit**.

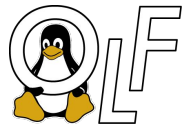
Lab Code: 348f - OCP4 and Container Storage for Admins

Activation Key:

E-Mail Address:

All fields are **required**.

- **Unless your event organizer says otherwise, we will not e-mail you and your e-mail address will be deleted from this system after this session is over**. Normally it is only used for tracking this session.
- You may need to refresh this page if you do not see an option for this lab session in the dropdown.
- If you are unsure which lab code to choose or what the activation key is please notify a lab assistant.



Red Hat

Welcome to: OCP4 and Container Storage for Admins

Your assigned lab GUID is **e590**

Let's get started! Please read these instructions carefully before starting to have the best lab experience:

- Save the above **GUID** as you will need it to access your lab's systems from your workstation.
- Consult the lab instructions *before* attempting to connect to the lab environment.
- Please note the following information about your lab environment:
  - Openshift Console: https://console-openshift-console.apps.cluster-qq8pz.qq8pz.sandbox1506.opentlc.com
  - Openshift API for command line 'oc' client: https://api.cluster-qq8pz.qq8pz.sandbox1506.opentlc.com:6443
  - Download oc client from http://d3s3zqyaz8cp2d.cloudfront.net/pub/openshift-v4/clients/ocp/stable-4.7/openshift-client-linux.tar.gz
  - Access the workshop at 'https://dashboard-lab-ocp-cns.apps.cluster-qq8pz.qq8pz.sandbox1506.opentlc.com'
  - Login with 'kubeadmin' and '[redacted]'
  - Workshop may not be accessible until rollout finishes shortly.
  - You can access your bastion via SSH:
  - ssh lab-user@bastion.qq8pz.sandbox1506.opentlc.com
  - Make sure you use the username 'lab-user' and the password '[redacted]' when prompted.
- The following URLs and information will be used in your lab environment. Please only access these links when the lab instructions specify to do so:
  - **Username:** kubeadmin
  - **Password:** [redacted]
  - Note: The lab instructions may specify other host names and/or URLs.

WARNING: You should only click FORGET SESSION if **requested** to do so by a lab attendant.

FORGET SESSION

Red Hat

Authorize Access

Service account dashboard-user in project lab-ocp-cns is requesting permission to access your account (kube:admin)

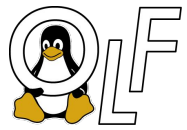
Requested permissions

- [x] **user:info**

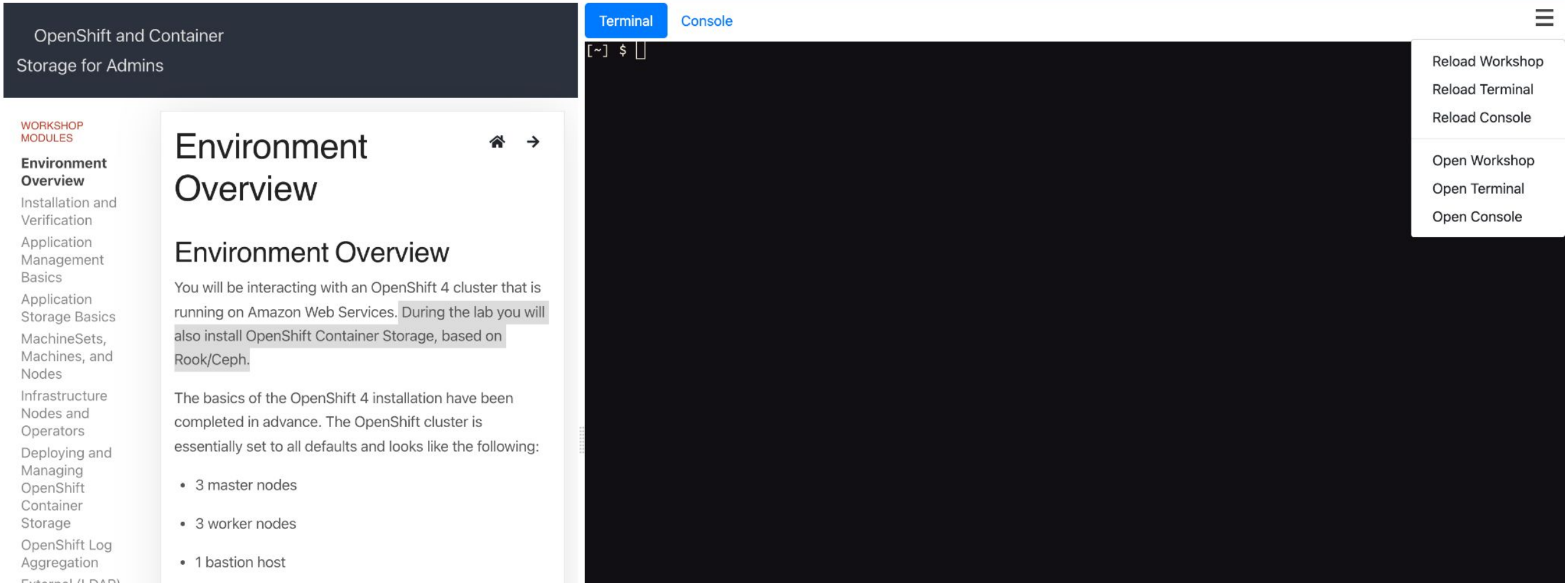
  Read-only access to your user information (including username, identities, and group membership)

You will be redirected to https://dashboard-lab-ocp-cns.apps.cluster-qq8pz.qq8pz.sandbox1506.opentlc.com/oauth_callback

Allow selected permissions | Deny

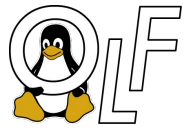


Red Hat


OpenShift and Container Storage for Admins
WORKSHOP MODULES
Environment Overview
Installation and Verification
Application Management Basics
Application Storage Basics
MachineSets, Machines, and Nodes
Infrastructure Nodes and Operators
Deploying and Managing OpenShift Container Storage
OpenShift Log Aggregation
Environment Overview
Environment Overview
You will be interacting with an OpenShift 4 cluster that is running on Amazon Web Services. During the lab you will also install OpenShift Container Storage, based on Rook/Ceph.
The basics of the OpenShift 4 installation have been completed in advance. The OpenShift cluster is essentially set to all defaults and looks like the following:
3 master nodes
3 worker nodes
1 bastion host
Terminal
Console
[~] $
Reload Workshop
Reload Terminal
Reload Console
Open Workshop
Open Terminal
Open Console

Workshop Instructions

- **Everything is done in browser** - no local commands or installs needed on your laptop
- Works best in Chrome or Firefox
- **Turn off VPN** (we use websockets extensively), **pause AdBlock** for the lab domain (there are no ads)
- Go slow, read, re-read
- Type things (I know it's painful)
- **Ask Questions!**



Red Hat

Lab Request URL:
https://red.ht/olf-kube

Password: r3dh4t!

Exercise 1: Environment Overview

What you're working with

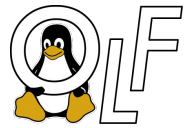
Red Hat

Environment Overview

You will be interacting with an OpenShift 4 cluster that is running on Amazon Web Services.

The basics of the OpenShift 4 installation have been completed in advance. The OpenShift cluster is essentially set to all defaults and looks like the following:

- 3 control (master) nodes
- 2 worker nodes
- 1 bastion host



Red Hat

Exercise 2: Installation and Verification

Checking that everything

is OK

Red Hat

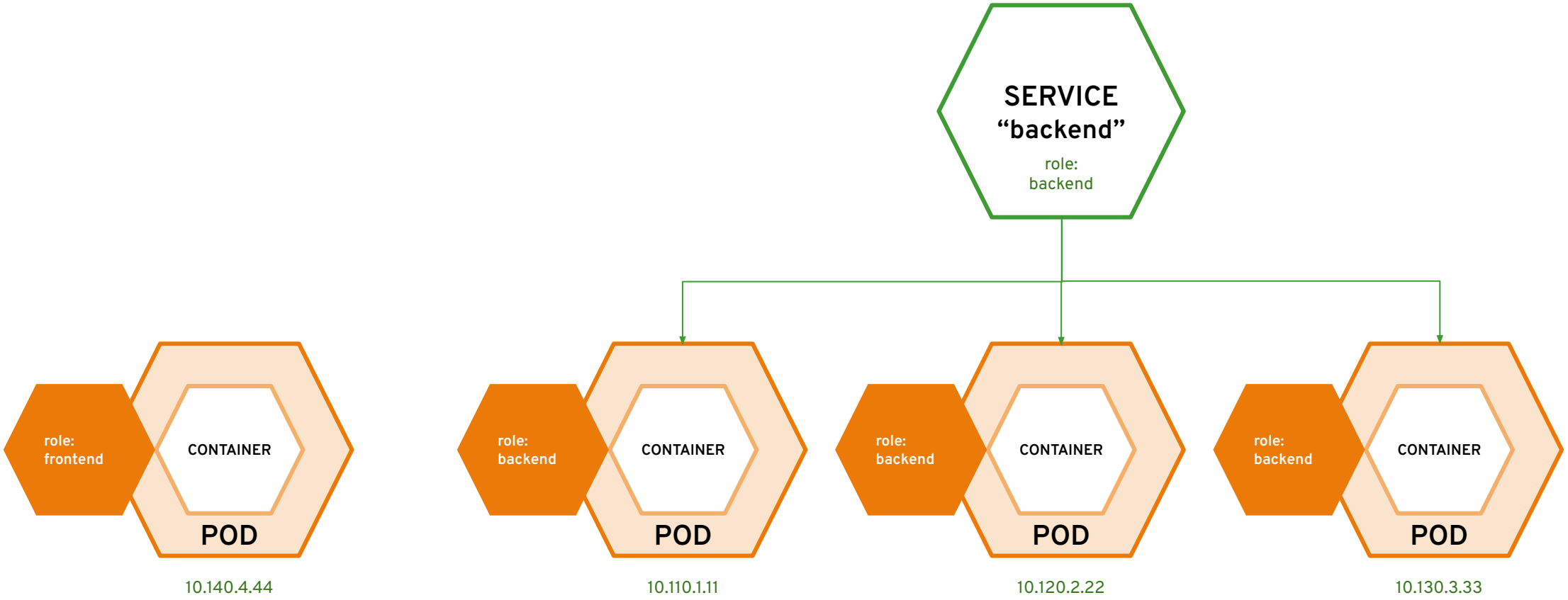
Exercise 3: Application Management Basics

Fundamental Kubernetes concepts

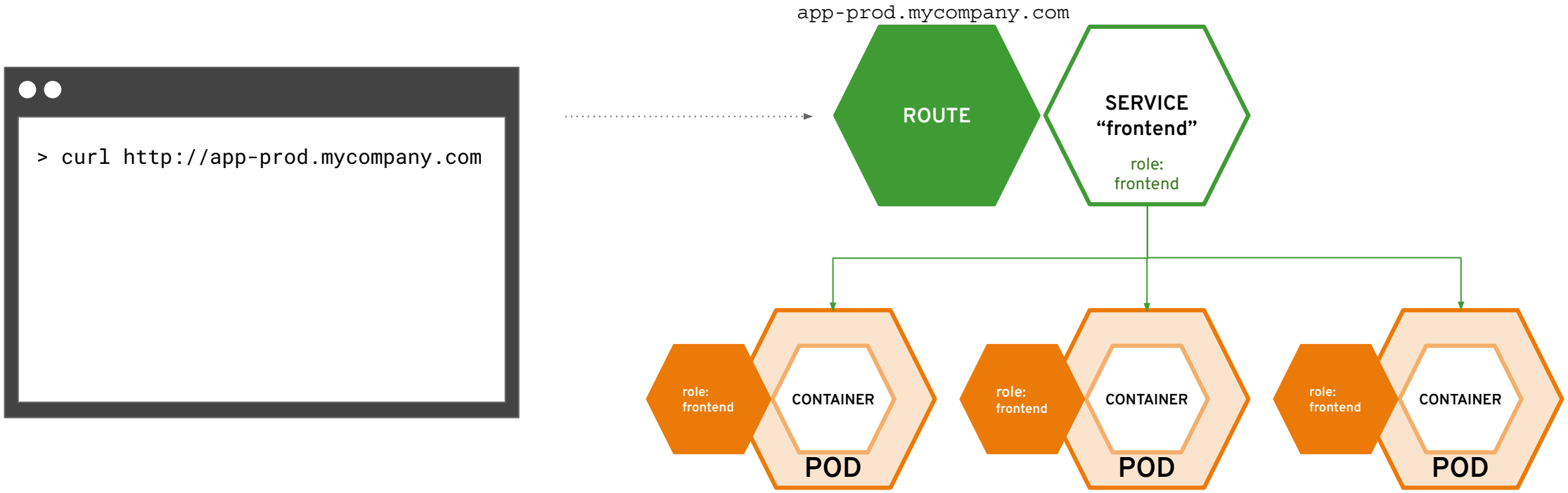
Red Hat

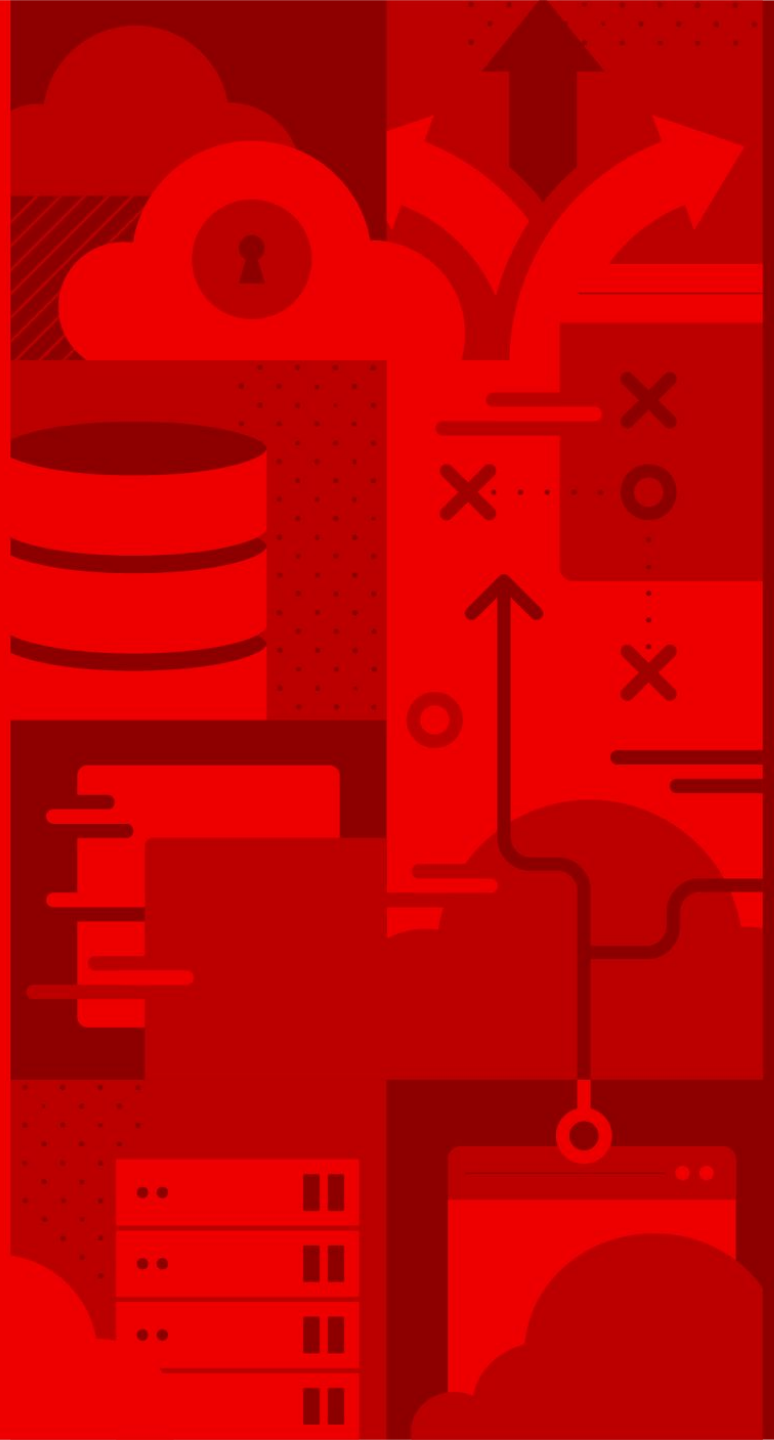
containers are wrapped in pods which are units of deployment and management

services provide internal load-balancing and service discovery across pods



`routes` make services accessible to clients outside the environment via real-world urls

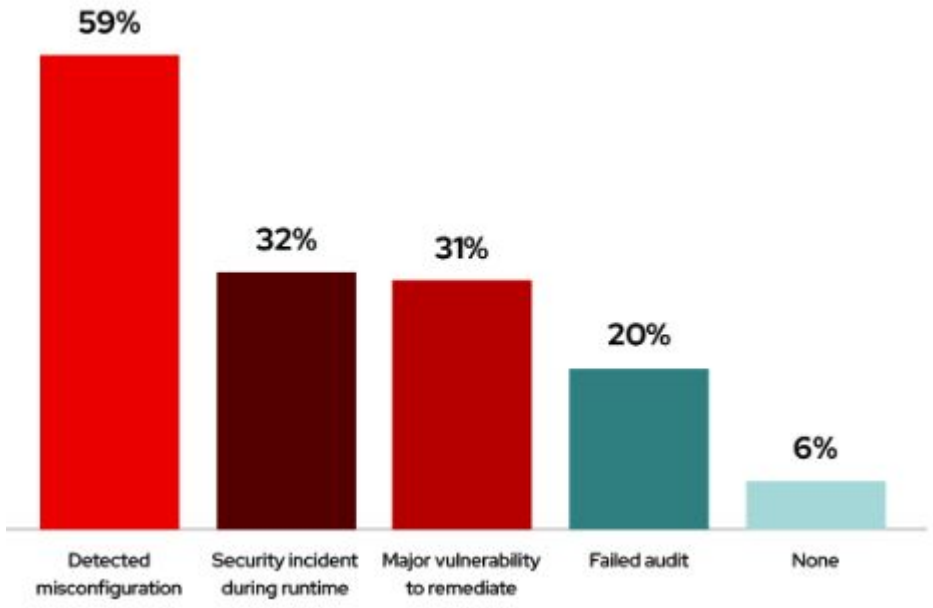




Kubernetes Security Best Practices

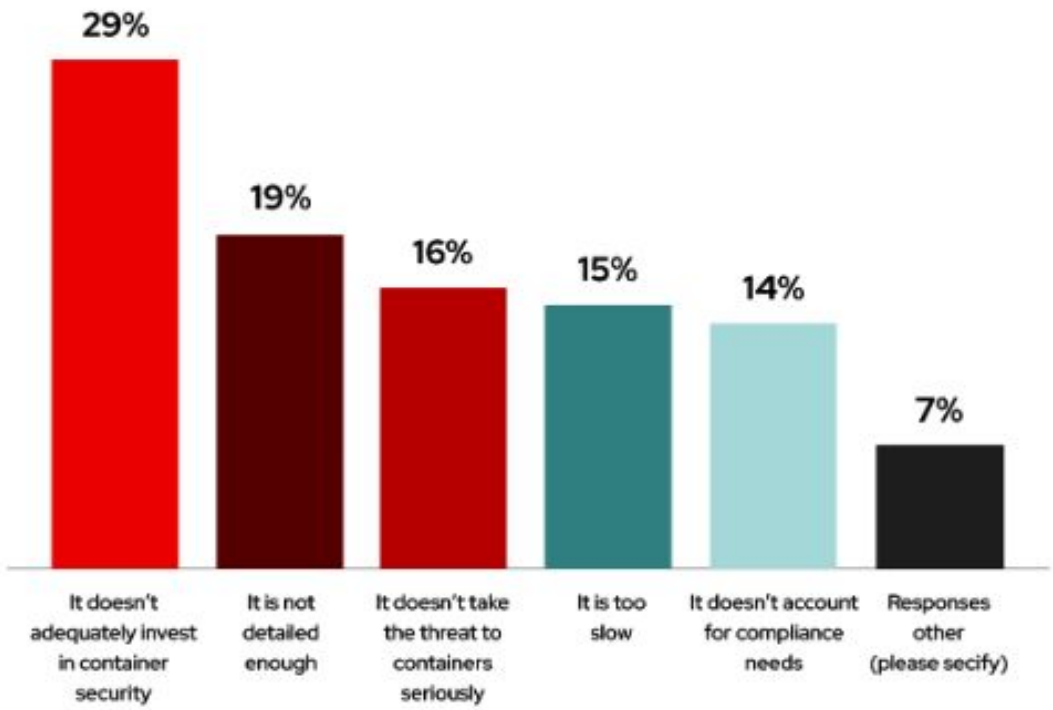
94% of respondents experienced at least one security incident in their Kubernetes environments in the last 12 months

Misconfiguration is the leading cause of security incidents, by a wide margin



Red Hat

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



Considerations for Securing Containers and Kubernetes

NIST 800-190

"Use container-specific host OSs instead of general-purpose ones to reduce attack surfaces."

CNCF Kube Security Audit

"...the underlying hosts, components, and environment of a Kubernetes cluster must be configured and managed. This management has a direct impact on the capabilities of the cluster..."

Gartner Market Guide for Cloud Workload Protection

"The best way to secure these rapidly changing and short-lived workloads is to start their protection proactively in the development phase ..."

"Replace antivirus (AV)-centric strategies with a "zero-trust execution"/default deny/application control approach to workload protection where possible."

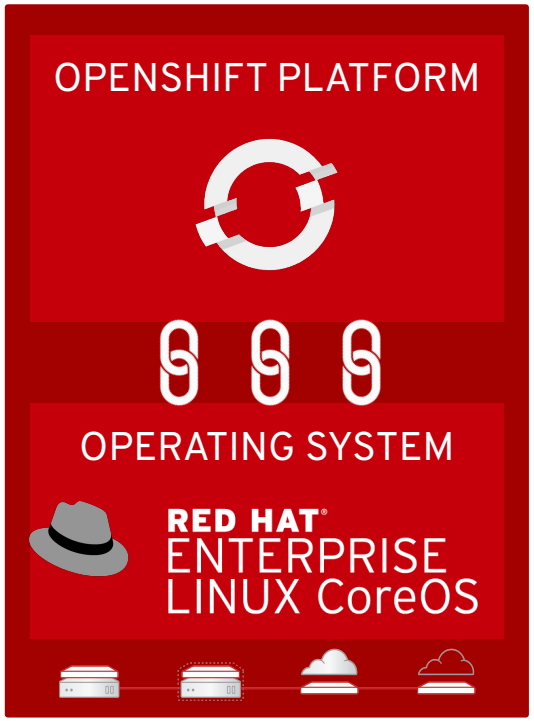
Sources

NIST Special Publication 800-190 *Application Container Security Guide*
CNCF Cloud Native Security Whitepaper
Kubernetes Security Whitepaper, Trail of Bits, May 31, 2019
Gartner: Market Guide for Cloud Workload Protection Platforms, ID G00356240, April 8, 2019

Red Hat Enterprise Linux CoreOS

Controlled Immutability / Container Optimized

OPENSHIFT 4



Role in OpenShift Ecosystem

- Versioned and validated for specific OpenShift version
- User space read-only

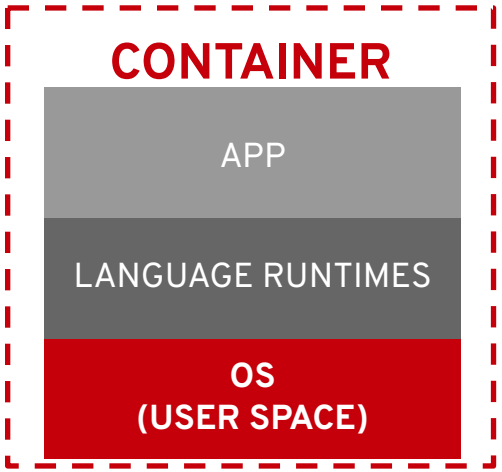
Managed by the OpenShift Cluster

- Considered a member of an OpenShift Deployment
- Configuration managed by the Machine Config Operator
  - Container runtime
  - Kubelet configuration
  - Authorized container registries
  - SSH Configuration
  - Multiple machine pools can be created
- Continuously monitoring for configuration drift
- Deploy the File Integrity operator to monitor for changes to files

Red Hat

The Red Hat Universal Base Image

The base image for all your needs -- enterprise architecture, security and performance



The Red Hat Universal Base Image is based on RHEL and made available at no charge by a new end user license agreement.

Development

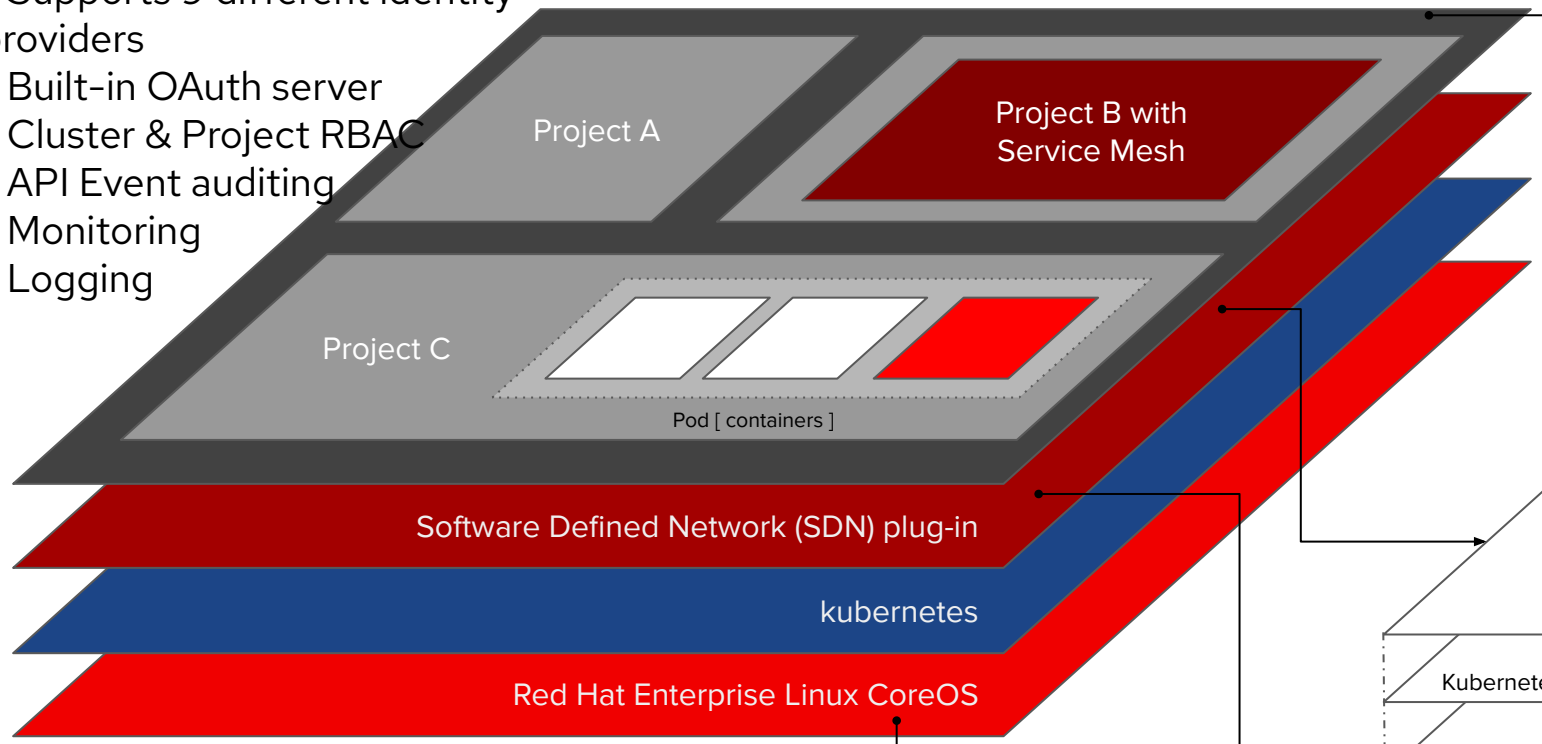
- Minimal footprint (~90 to ~200MB)
- Programming languages (Modularity & AppStreams)
- Enables a single CI/CD chain

Production

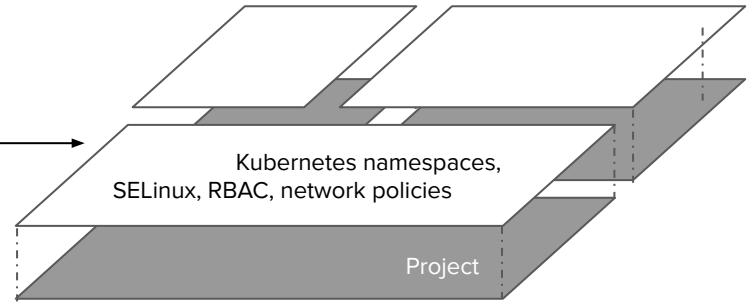
- Supported as RHEL when running on RHEL
- Same Performance, Security & Life cycle as RHEL
- Can attach RHEL support subscriptions as RHEL

Defense in Depth

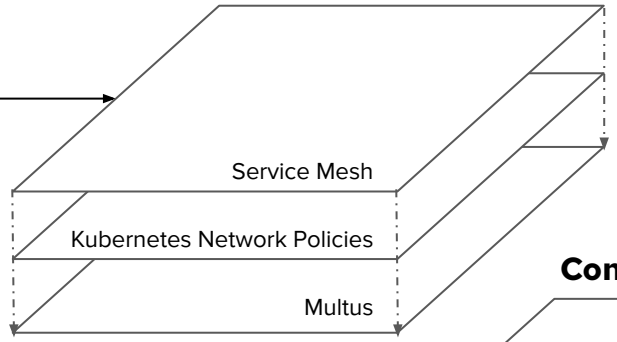
OpenShift Container Platform
- Supports 9 different identity providers
- Built-in OAuth server
- Cluster & Project RBAC
- API Event auditing
- Monitoring
- Logging



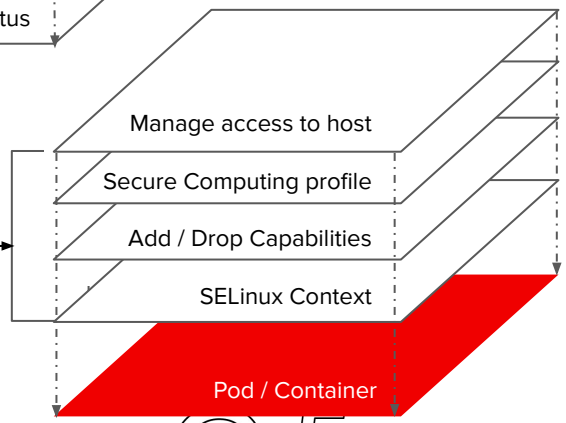
Compartmentalized Projects



Network Security



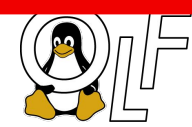
Container Security



Red Hat CoreOS
- Minimized attack surface
- Controlled Immutability
- SELinux on by default
- Kernel namespaces and Cgroups
- CRI-O container runtime, Kubelet
- Auditd for host-level audit
- FIPS enablement
- RHCOS volume encryption

Security Context Constraints
(Kubernetes Admission Controller)

Red Hat Universal Base Image
containers can leverage RHEL FIPS capabilities



Red Hat

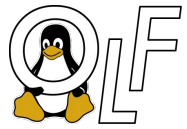
Comprehensive, Layered Approach to Security

	Control	Protect	Detect & Respond
Fleet	Application Lifecycle and Locality	Fleet Management	Fleet Observability & Alerts
Workload	Vulnerability analysis	Policy admission controller	Runtime behavioral analysis
Workload	App config analysis	Compliance assessments	Auto-suggest network policies
Workload	APIs for CI/CD integrations	Risk profiling	Threat detection / incident response
Platform	Trusted content	Kubernetes platform lifecycle	Container isolation
Platform	Container registry	Identity and access management	Network isolation
Platform	Build management	Platform data	Application access and data
Platform	CI/CD pipeline	Deployment policies	Observability
	BUILD	DEPLOY	RUN

DevSecOps

Build Security into the App

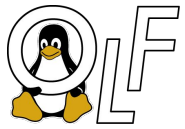
- Trusted Content
- Enterprise Registry
- Control and Automate builds
- DevSecOps - Must be in the pipeline



Red Hat

Manage deployment config, security, compliance

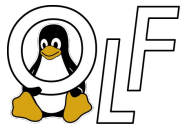
- Platform configuration and life-cycle management
- Identity and access management
- Security for platform data and attached storage
- Deployment Policies



Red Hat

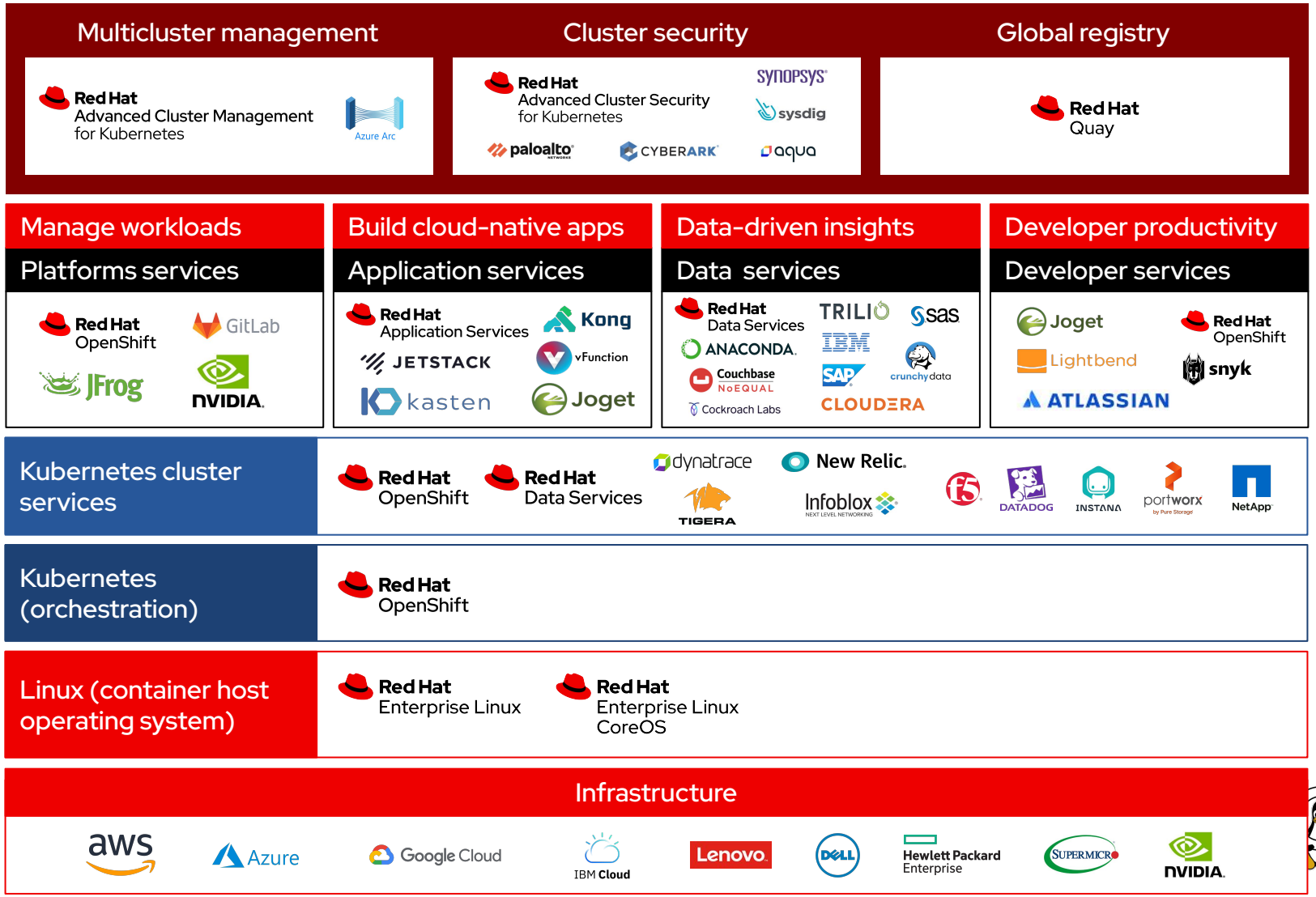
Runtime Protection

- Container Isolation
- Application and Network Isolation
- Secure Access to Apps
- Observability

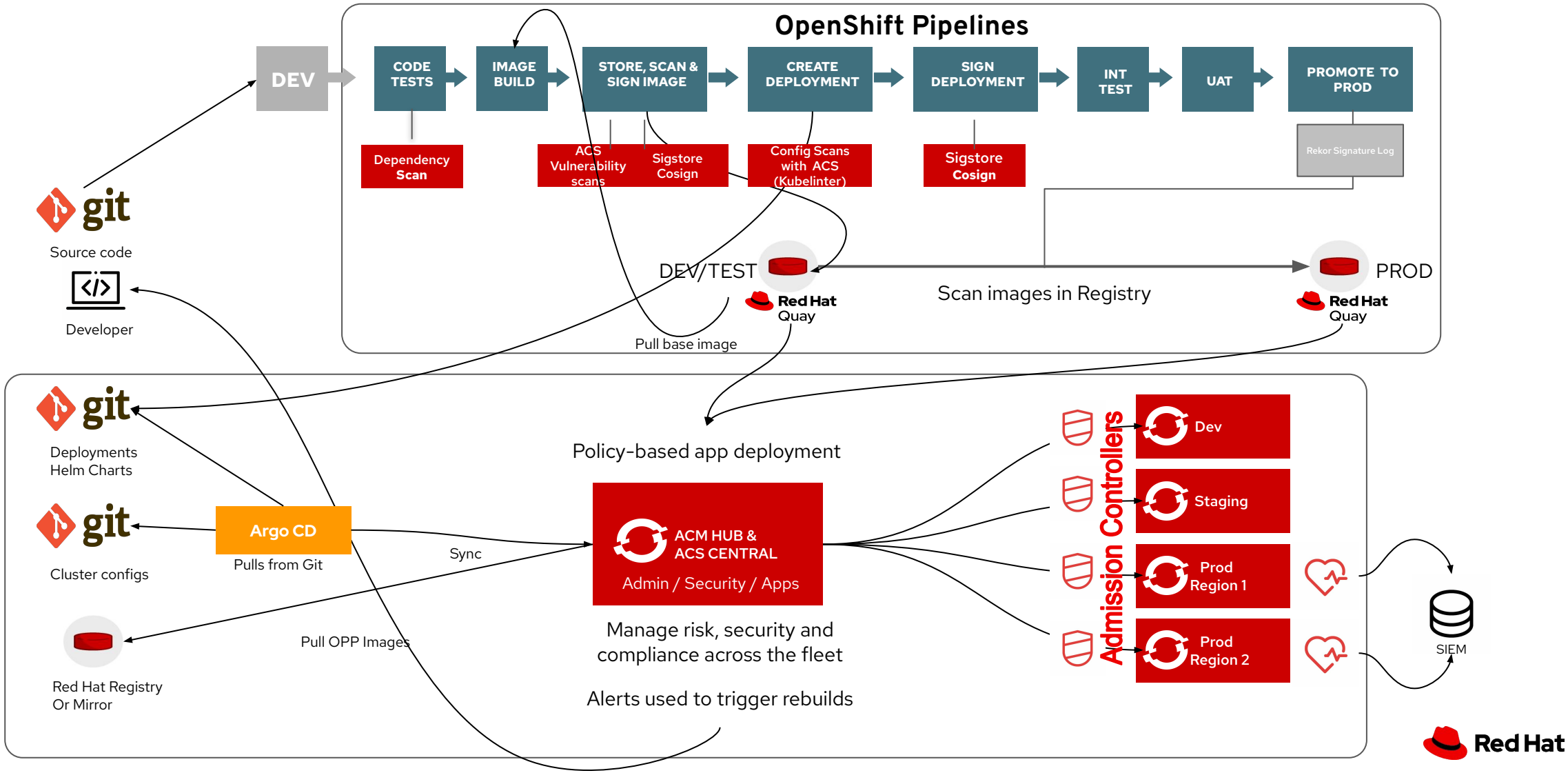


Red Hat

Extend security with best in class tooling



Policy Based Multi-cluster DevSecOps



Red Hat

Resources

Container security maturity model -

https://www.redhat.com/en/resources/container-security-maturity-model-whitepaper

Managing NIST 800-53 Controls in a Multicluster OpenShift Environment -

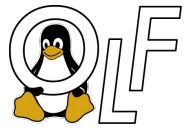
https://cloud.redhat.com/blog/managing-nist-800-53-controls-in-a-multicluster-openshift-environment-part-1

Container and Kubernetes security: An evaluation guide -

https://www.redhat.com/en/resources/container-kubernetes-security-evaluation-detail

State of Kubernetes security report -

https://www.redhat.com/en/resources/state-kubernetes-security-report



Red Hat

Resources

How OpenShift enables container security -

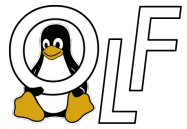
https://www.redhat.com/en/technologies/cloud-computing/openshift/security

A layered approach to container and Kubernetes security -

https://www.redhat.com/en/resources/layered-approach-container-kubernetes-security-whitepaper

OpenShift Security Guide (300 page in depth guide!) -

https://www.redhat.com/rhdc/managed-files/cl-openshift-security-guide-ebook-us287757-202103.pdf



Red Hat

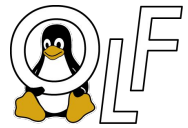
Exercise 4: Application Storage Basics

Persistent Volumes and
Persistent Volume Claims

Red Hat

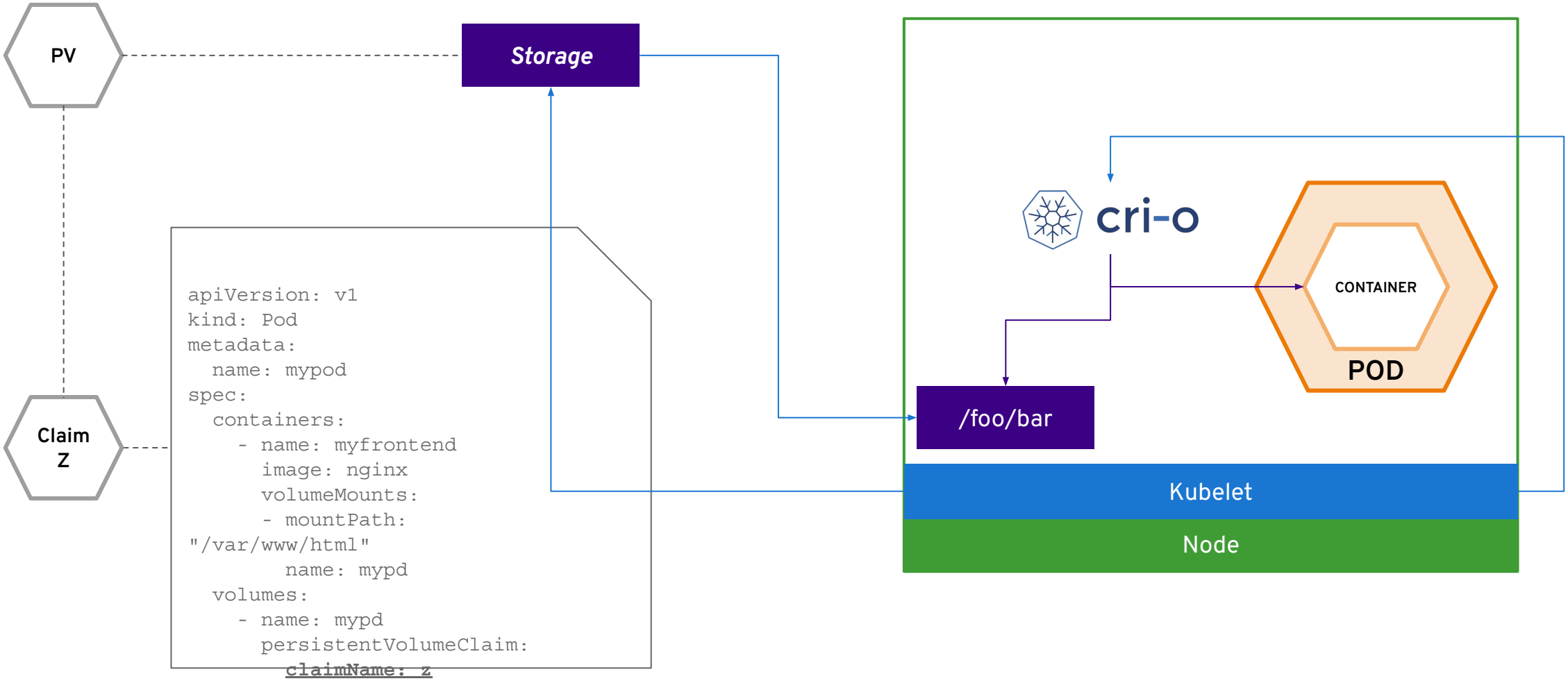
A broad spectrum of static and dynamic storage endpoints

NFS	OpenStack Cinder	iSCSI	Azure Disk	AWS EBS	FlexVolume
GlusterFS	Ceph RBD	Fiber Channel	Azure File	GCE Persistent Disk	VMWare vSphere VMDK
		NetApp Trident*	Container Storage Interface (CSI)**		

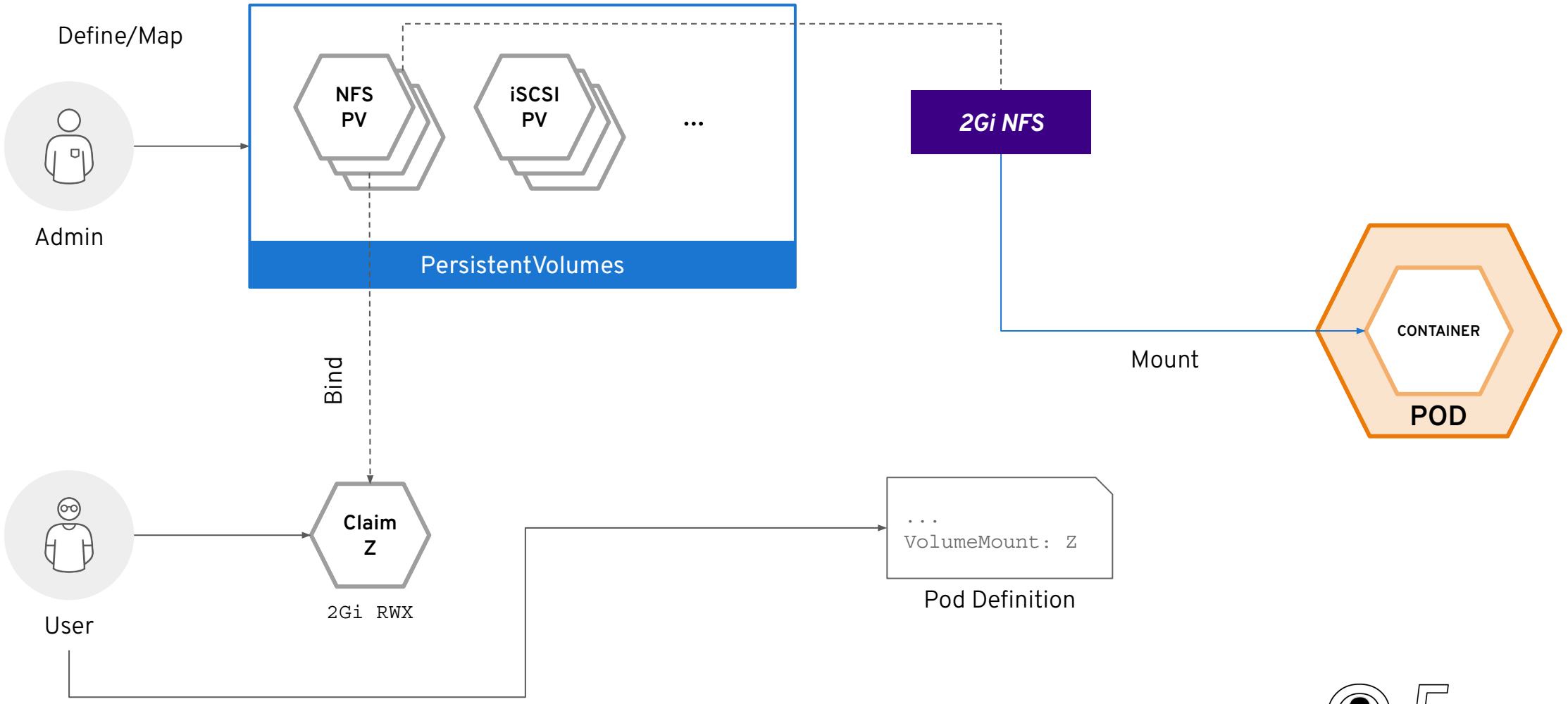


Red Hat

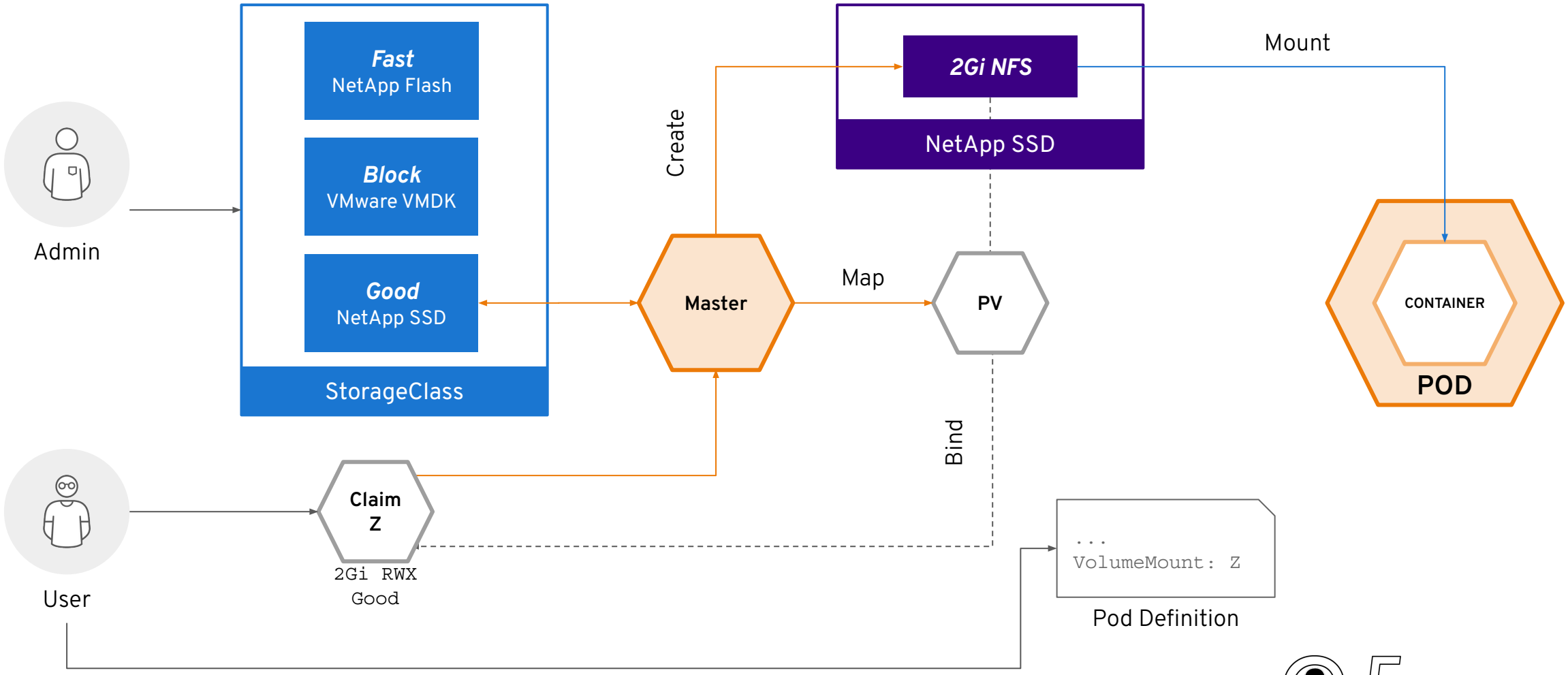
PV Consumption



Static Storage Provisioning



Dynamic Storage Provisioning

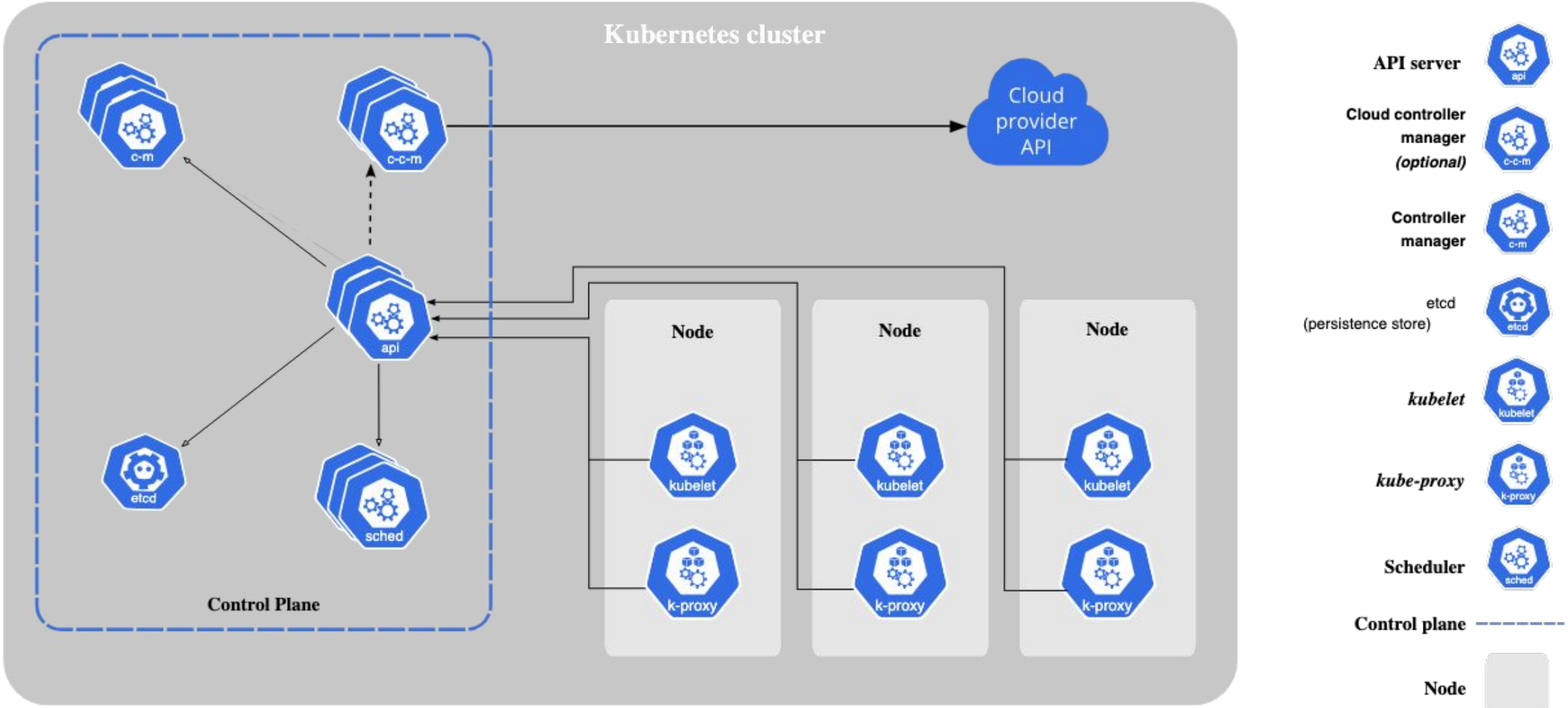


Exercise 5: Machinesets, Machines and Nodes

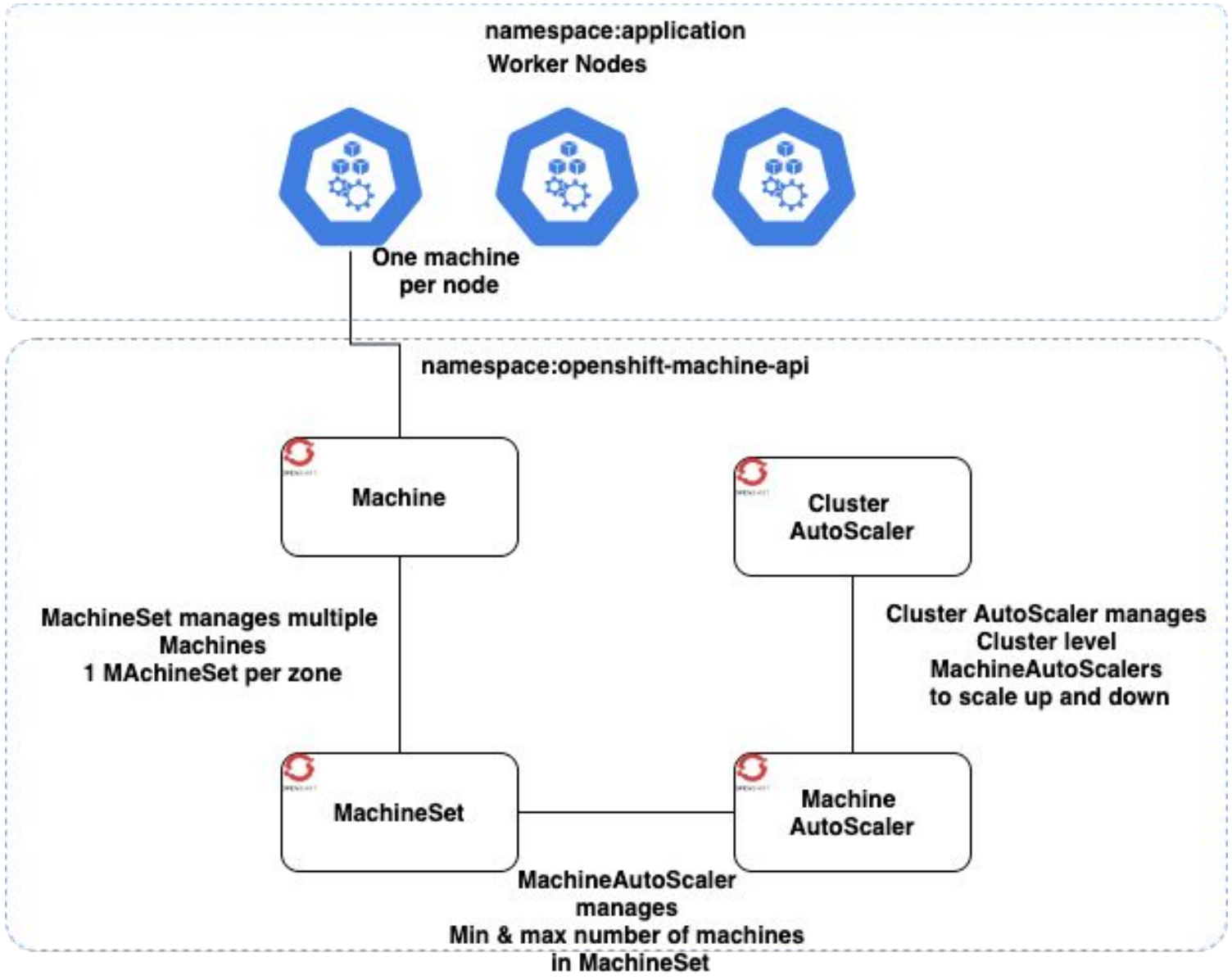
Expanding the cluster capacity quickly and easily

Red Hat

Cloud Controller Manager



Cluster API


namespace:application
Worker Nodes
One machine
per node
namespace:openshift-machine-api
Machine
Cluster
AutoScaler
MachineSet manages multiple
Machines
1 MAchineSet per zone
Cluster AutoScaler manages
Cluster level
MachineAutoScalers
to scale up and down
MachineSet
Machine
AutoScaler
MachineAutoScaler
manages
Min & max number of machines
in MachineSet

Exercise 6: Infrastructure Nodes and Operators

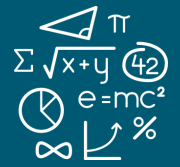
Common Day2 configuration tasks

Red Hat

Kubernetes-native day 2 management

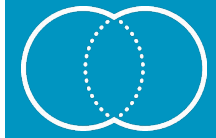
OPERATOR FRAMEWORK

Flexible app architectures



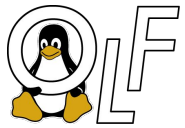
No reinvention of core concepts

Uniform deploy and debug



Truly hybrid

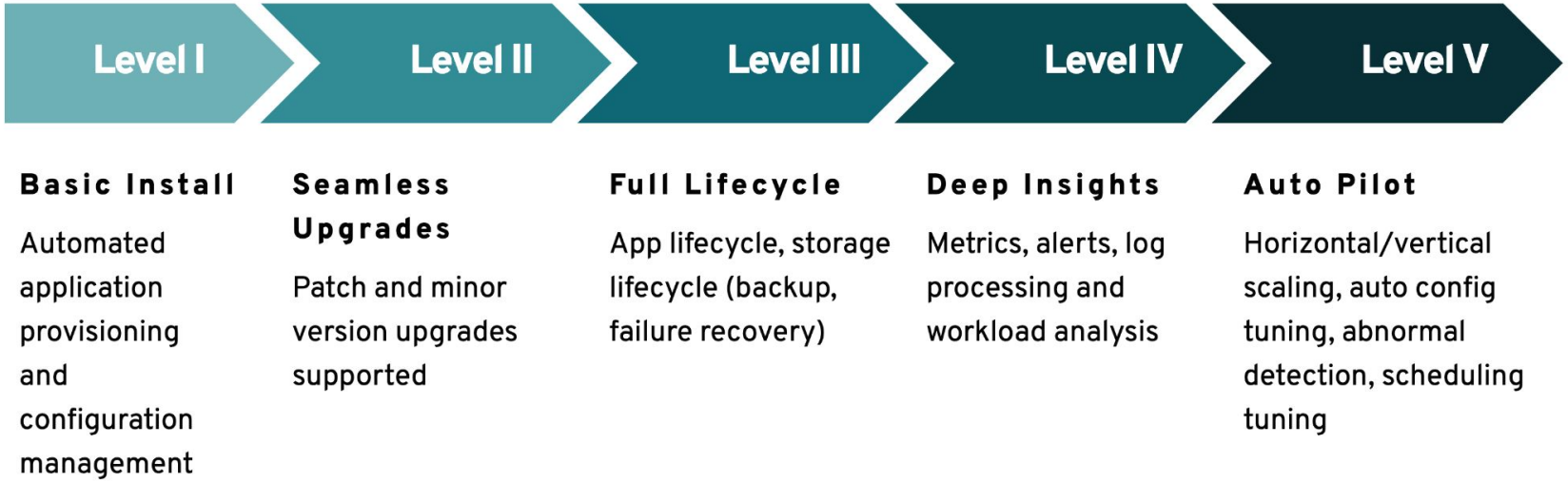
Operators codify operational knowledge and workflows to automate life-cycle management of containerized applications with Kubernetes

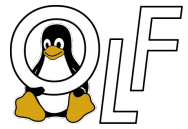


Red Hat

OPERATOR CAPABILITY LEVELS

Operators come in different maturity levels in regards to their lifecycle management capabilities for the application or workload they deliver. The capability models aims to provide guidance in terminology to express what features users can expect from an Operator.

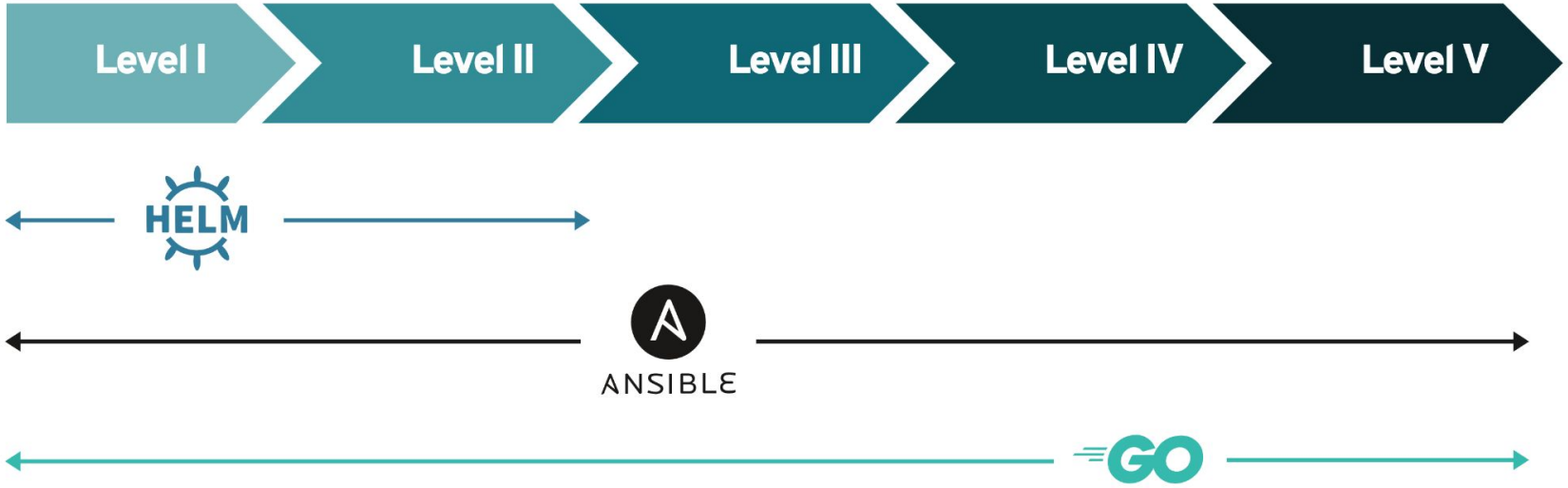




Red Hat

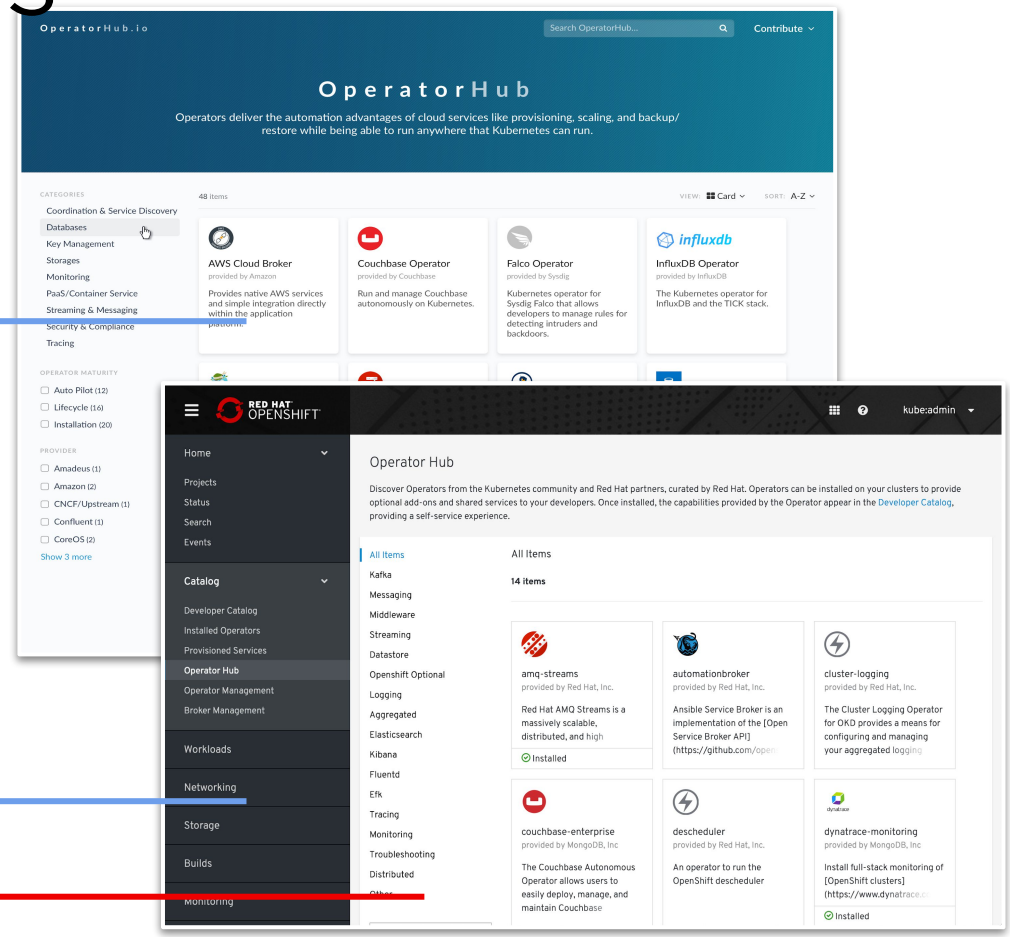
OPERATOR CAPABILITY LEVELS

Operators come in different maturity levels in regards to their lifecycle management capabilities for the application or workload they deliver. The capability models aims to provide guidance in terminology to express what features users can expect from an Operator.



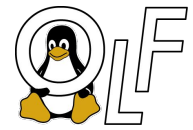
Operator Hub and certified Operators

- OperatorHub.io launched by Red Hat, AWS, Microsoft and Google
- OpenShift Operator Certification
- OperatorHub integrated into OpenShift 4



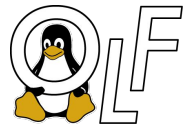
COMMUNITY OPERATORS

OPENSHIFT CERTIFIED OPERATORS



Red Hat

Operator Hub and certified Operators



Red Hat

PGO is developed with many years of production experience in automating Postgres management on Kubernetes, providing a seamless cloud native Postgres solution to keep your data always available.

- **PostgreSQL Cluster Provisioning**: Create, Scale, & Delete PostgreSQL clusters with ease, while fully customizing your Pods and PostgreSQL configuration!
- **High-Availability**: Safe, automated failover backed by a distributed consensus based high-availability solution. Uses Pod Anti-Affinity to help resiliency; you can configure how aggressive this can be! Failed primaries automatically heal, allowing for faster recovery time. You can even create regularly scheduled backups as well and set your backup retention policy
- **Disaster Recovery**: Backups and restores leverage the open source pgBackRest utility and includes support for full, incremental, and differential backups as well as efficient delta restores. Set how long you want your backups retained for. Works great with very large databases!
- **Monitoring**: Track the health of your PostgreSQL clusters using the open source pgMonitor library.
- **Clone**: Create new clusters from your existing clusters or backups with efficient data cloning.
- **TLS**: All connections are over TLS. You can also bring your own TLS infrastructure if you do not want to use the provided defaults.
- **Connection Pooling**: Advanced connection pooling support using pgBouncer.
- **Affinity and Tolerations**: Have your PostgreSQL clusters deployed to Kubernetes Nodes of your preference. Set your pod anti-affinity, node affinity, Pod tolerations and more rules to customize your deployment topology!
- **Full Customizability**: Crunchy PostgreSQL for Kubernetes makes it easy to get your own PostgreSQL-as-a-Service up and running and fully customize your deployments, including:
  - Choose the resources for your Postgres cluster: container resources and storage size. Resize at any time with minimal disruption.
  - Use your own container image repository, including support `imagePullSecrets` and private repositories
  - Customize your PostgreSQL configuration

and much more!

Red Hat

Exercise 8: OpenShift Logging

A solution to visualize and corroborate log messages across pods

Red Hat

Vector & Loki - Logging Subsystem

The logging subsystem for Red Hat OpenShift collects container logs and node logs. These are categorized into types:

- **application** - Container logs generated by non-infrastructure containers.
- **infrastructure** - Container logs from namespaces kube-* and openshift-*, and node logs from journald.
- **audit** - Logs from auditd, kube-apiserver, openshift-apiserver, and ovn if enabled.

Vector & Loki - Logging Subsystem

Components

- **Loki:** a search and analytics engine to store logs
- **Vector:** gathers logs and sends to Elasticsearch.
- **Kibana:** A web UI for Elasticsearch.

Access control

- Cluster administrators can view all logs
- Users can only view logs for their projects

Ability to forward logs elsewhere

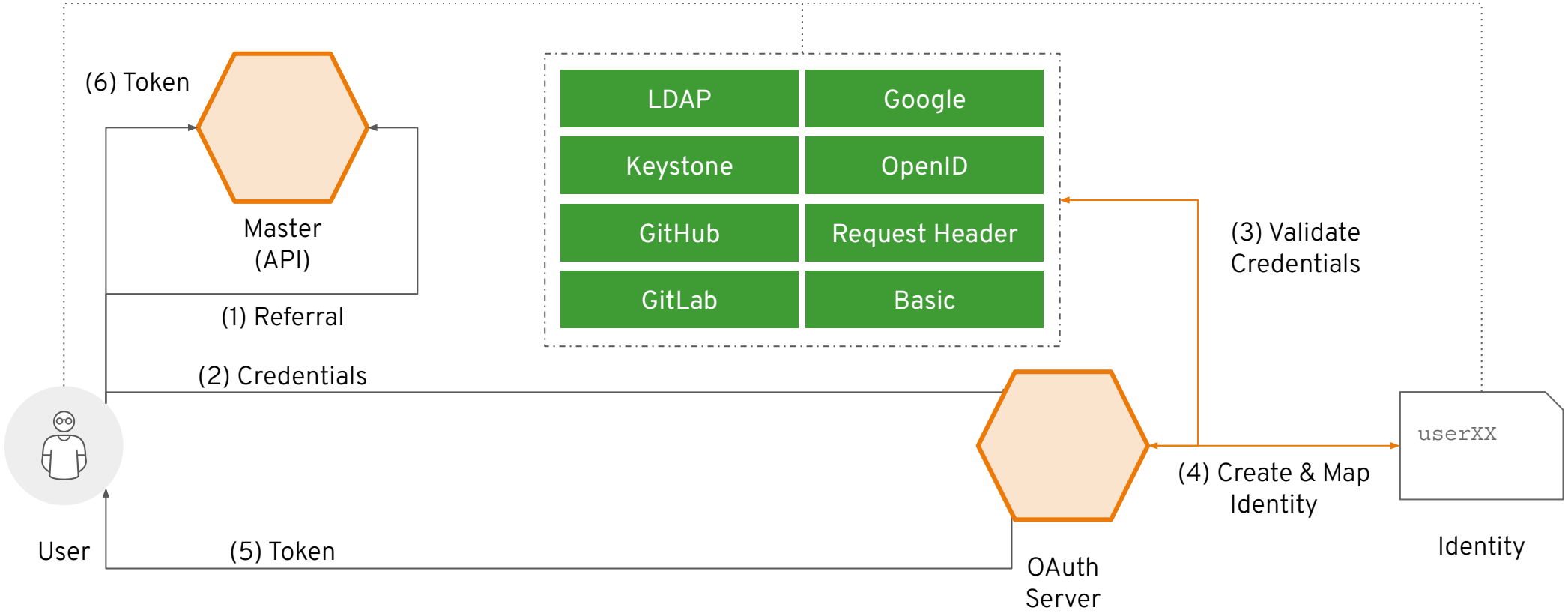
- External syslog, elasticsearch, Splunk, Cloudwatch, Kafka

Exercise 9: Authentication Providers, Users, and Groups

Configuring RBAC alongside LDAP

Red Hat

Identity and Access Management



Fine-Grained RBAC

- Project scope & cluster scope available
- Matches request attributes (verb,object,etc)
- If no roles match, request is denied (deny by default)
- Operator- and user-level roles are defined by default
- Custom roles are supported

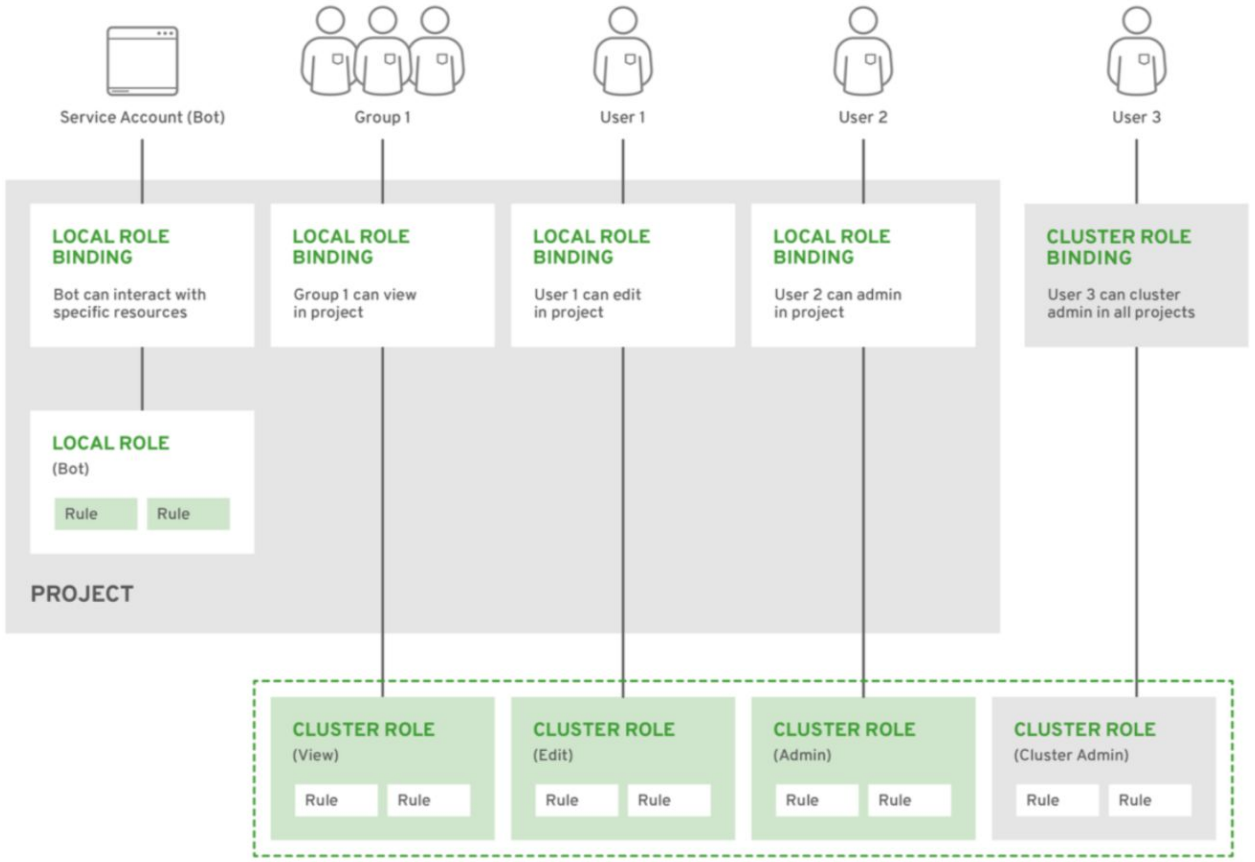


Figure 12 - Authorization Relationships

Exercise 10: OpenShift Cluster Monitoring

A no-touch built-in monitoring solution implemented with Red Hat SRE best practices

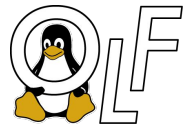
Red Hat

OpenShift Cluster Monitoring

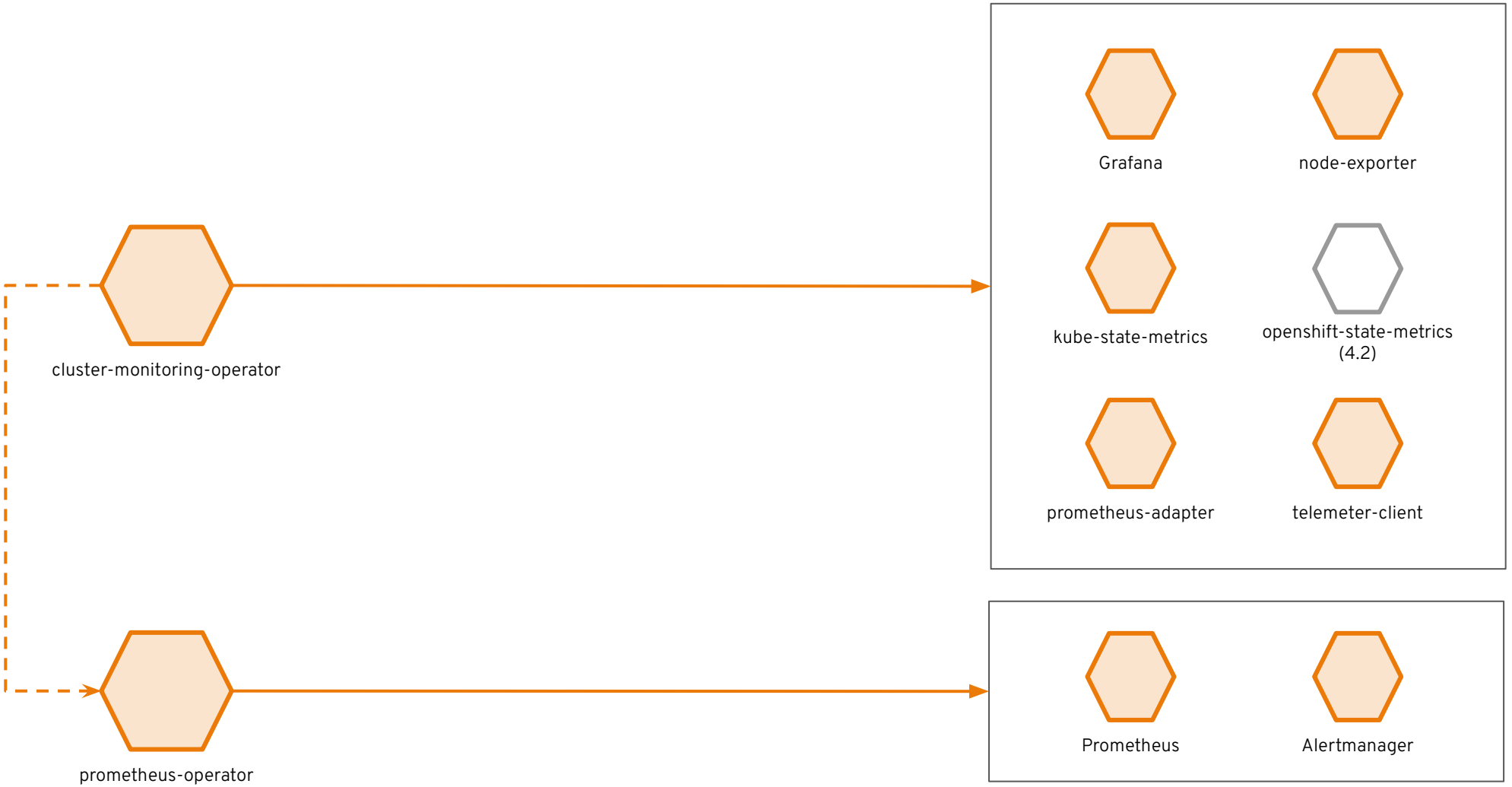
Metrics collection and storage via Prometheus, an open-source monitoring system time series database.

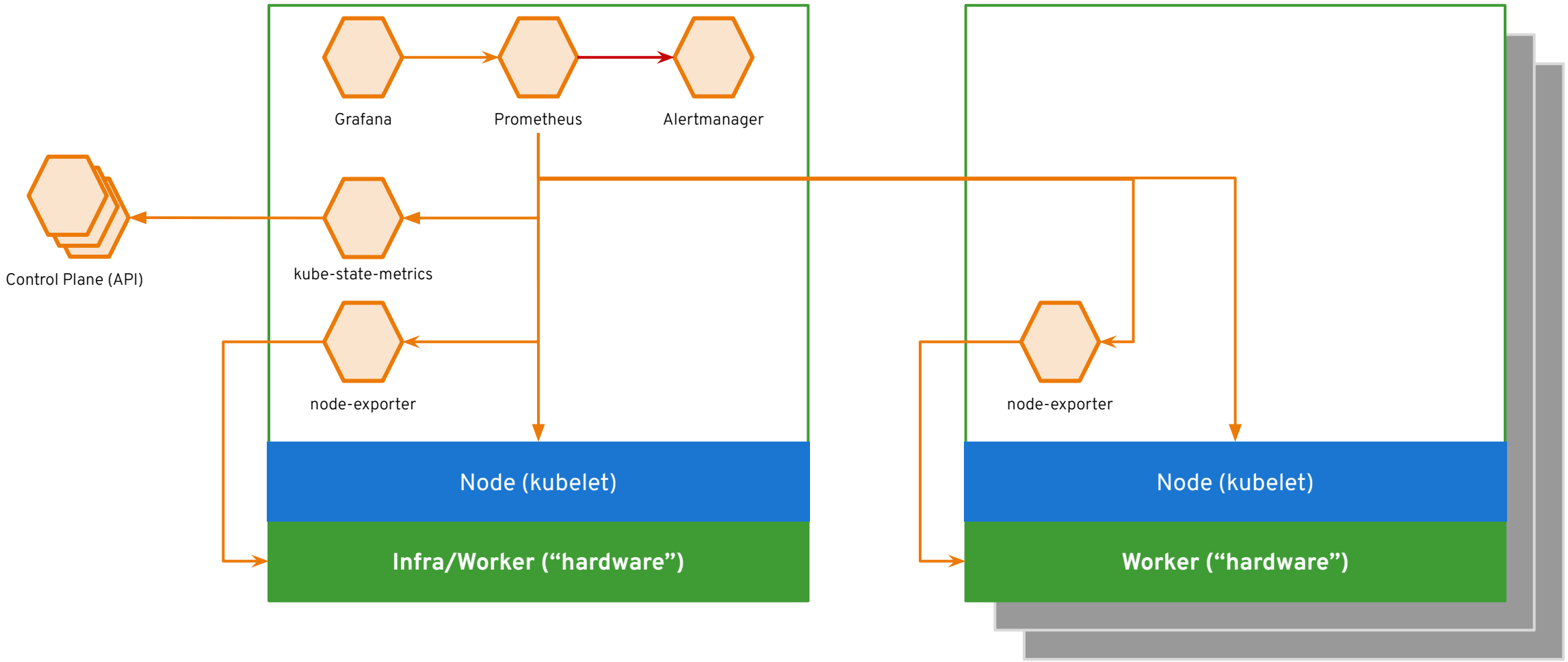
Alerting/notification via Prometheus' Alertmanager, an open-source tool that handles alerts send by Prometheus.

Metrics visualization via Grafana, the leading metrics visualization technology.



Red Hat


Grafana
node-exporter
kube-state-metrics
openshift-state-metrics
(4.2)
prometheus-adapter
telemeter-client
cluster-monitoring-operator
prometheus-operator
Prometheus
Alertmanager


Grafana
Prometheus
Alertmanager
Control Plane (API)
kube-state-metrics
node-exporter
node-exporter
Node (kubelet)
Infra/Worker (“hardware”)
Node (kubelet)
Worker (“hardware”)

Exercise 11: Project Template, Quota, and Limits

Forcing users to be well-behaved with respect to resource consumption

Red Hat

Types of resources

Compressible: there is always more, you just have to wait for the next unit of time

- Cpu, network bandwidth, iops ...
- Compressible resources are always expressed as qyt/unit of time for example cpu/millisec

Incompressible: once used up, there is no more.

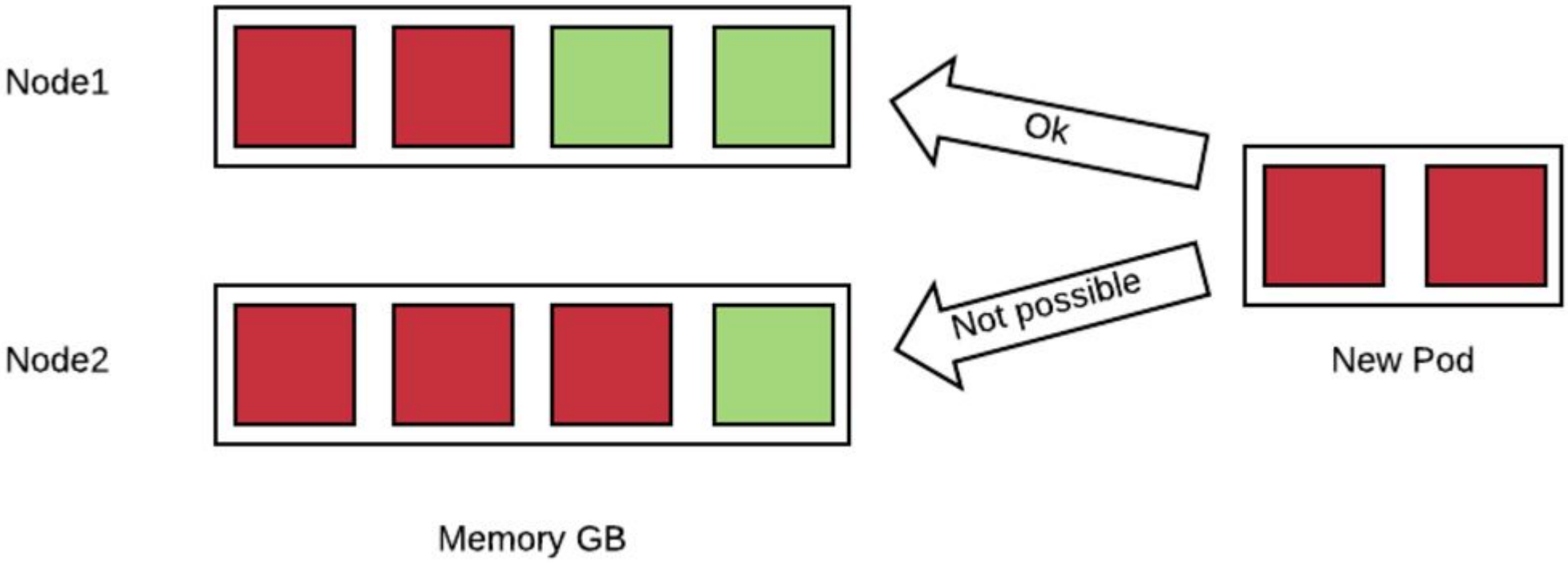
- Memory, diskspace ...

Quotas, requests and limits

Constraint	When it's Evaluated	Purpose
Quotas	Request Time	To limit how many resources a single tenant can request so that they cannot take over the cluster.
Requests	Scheduling Time	Requests are used to choose which node is the best fit for the given workload (together with other criteria).
Limits	Run Time	Limits are used to limit the maximum amount of resources that a container can use at runtime.

Scheduling a pod

(how memory requests affects scheduling a pod)



Red squares on the nodes represent the sum of the requests of previously scheduled pods.

Compare the output of “oc describe node” and “oc adm top”.

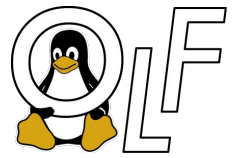
Red Hat

Exercise 12: Networking and NetworkPolicy

Fine-grained control over network connectivity between and even inside tenants

Red Hat

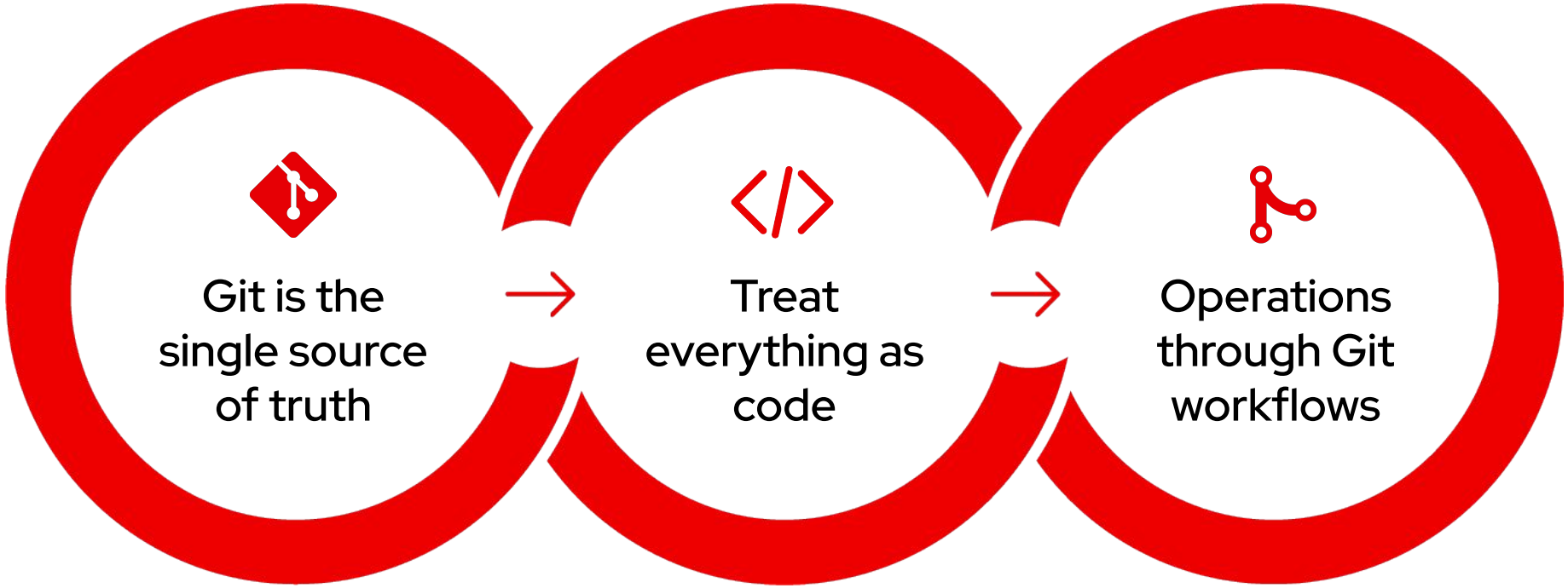
OpenShift GitOps



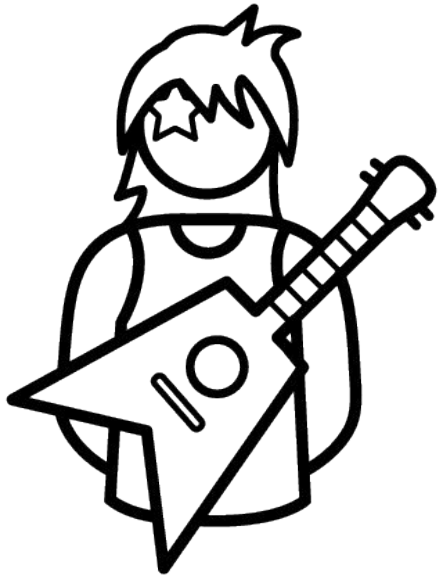
Red Hat

What is GitOps?

An developer-centric approach to Continuous Delivery and infrastructure operation



GitOps is for Everyone

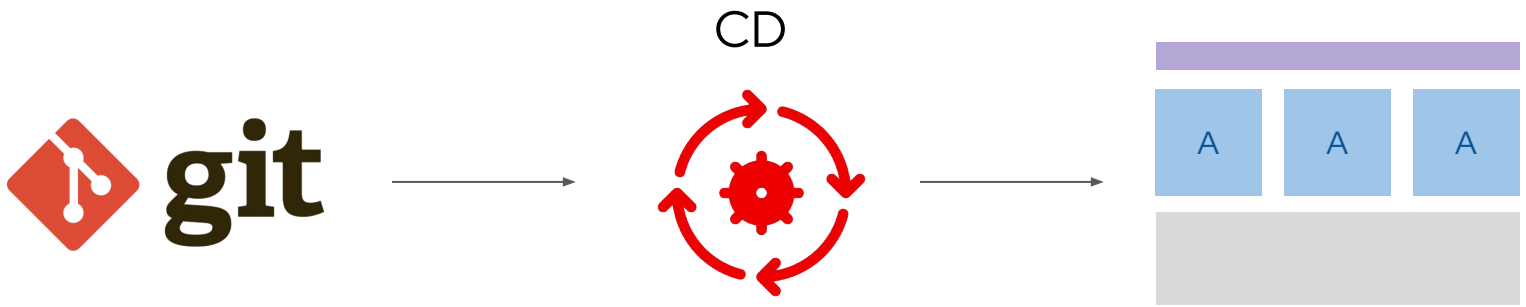


Developers

Operations

GitOps Workflow

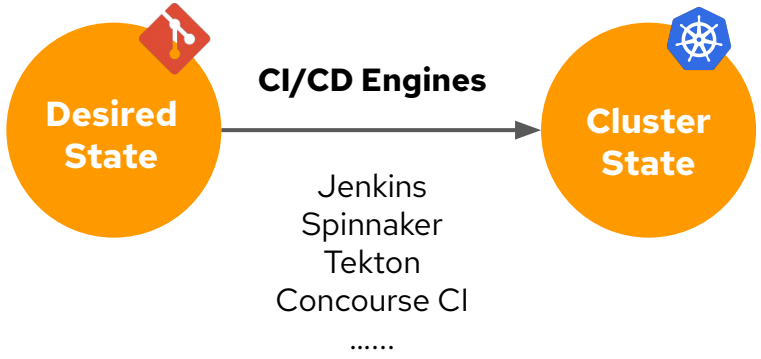
a declarative approach to application delivery

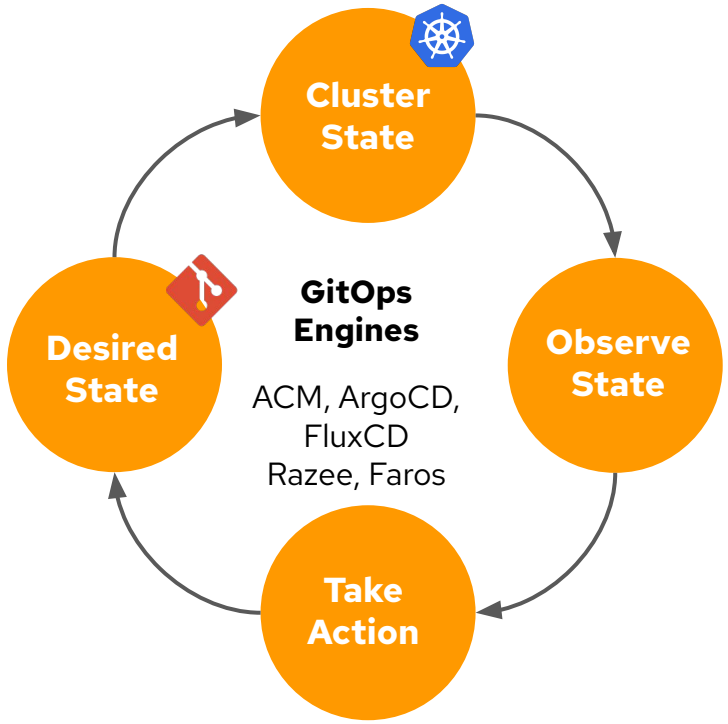


What you want
(desired state)

What you have
(current state)

GitOps versus CI/CD





Red Hat

Why GitOps?

Standard Workflow

Familiar tools and Git workflows from application development teams

Enhanced Security

Review changes beforehand, detect configuration drifts, and take action

Visibility and Audit

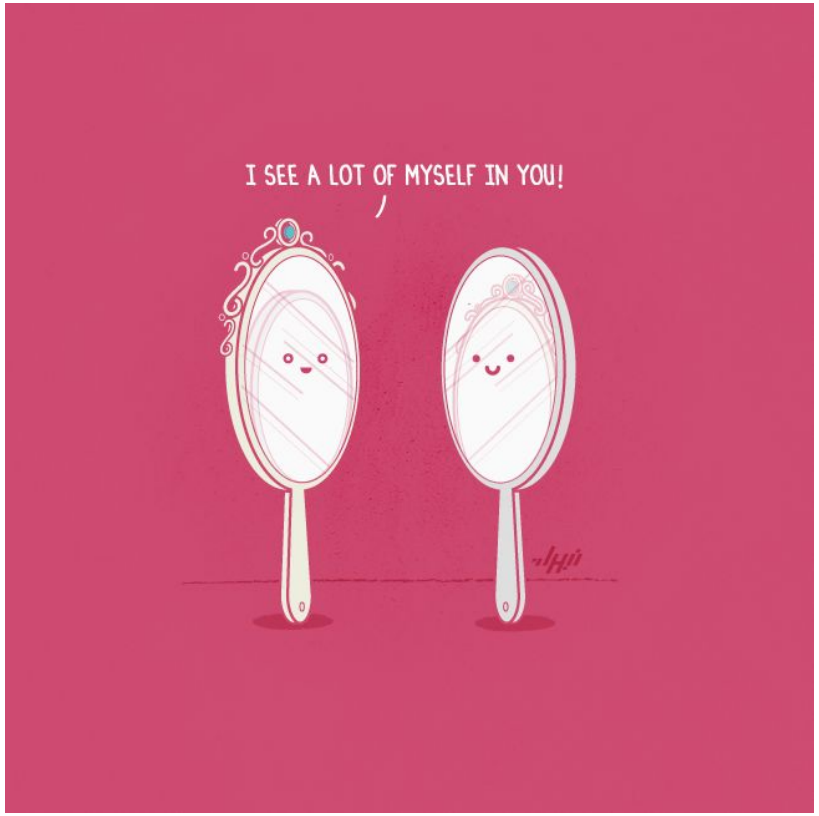
Capturing and tracing any change to clusters through Git history

Multi-cluster consistency

Reliably and consistently configure multiple Kubernetes clusters and deployment

Kubernetes and GitOps - A Perfect Match

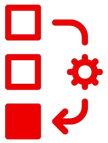
- Kubernetes is a declarative environment
  - Application deployments are declared and Kubernetes scheduler makes it happen
  - OpenShift goes further in that cluster configuration is declared and Operators make it happen
- GitOps in traditional environments requires automation/scripting, declarative environment minimizes or eliminates this need
- Declarations are yaml files which are easily stored and managed in git



OpenShift GitOps

Multi-cluster config management

Declaratively manage cluster and application configurations across multi-cluster OpenShift and Kubernetes infrastructure with Argo CD



Automated Argo CD install and upgrade

Automated install, configurations and upgrade of Argo CD through OperatorHub

Opinionated GitOps bootstrapping

Bootstrap end-to-end GitOps workflows for application delivery using Argo CD and Tekton with GitOps Application Manager CLI

Deployments and environments insights

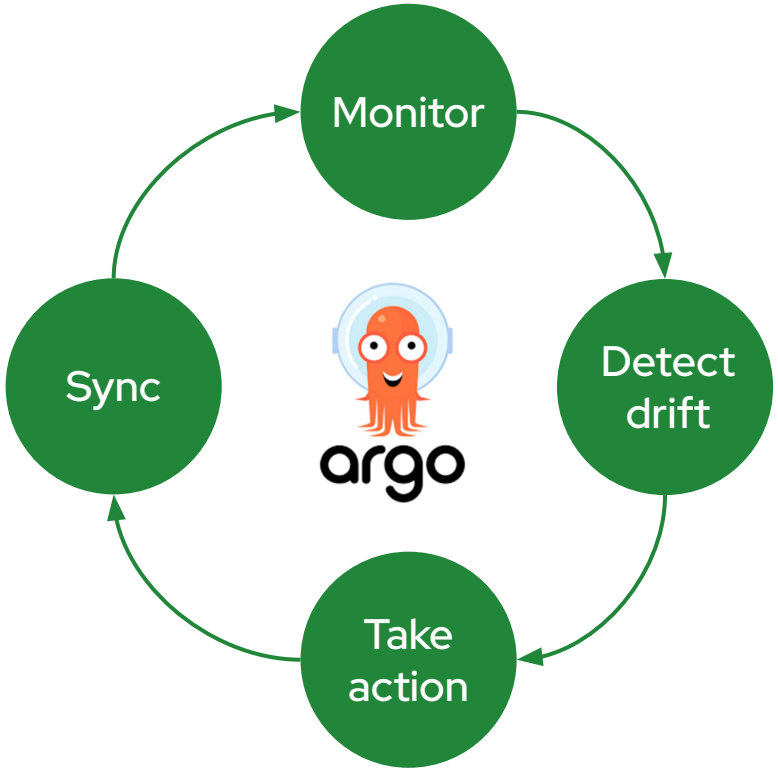
Visibility into application deployments across environments and the history of deployments in the OpenShift Console

Powered by argo

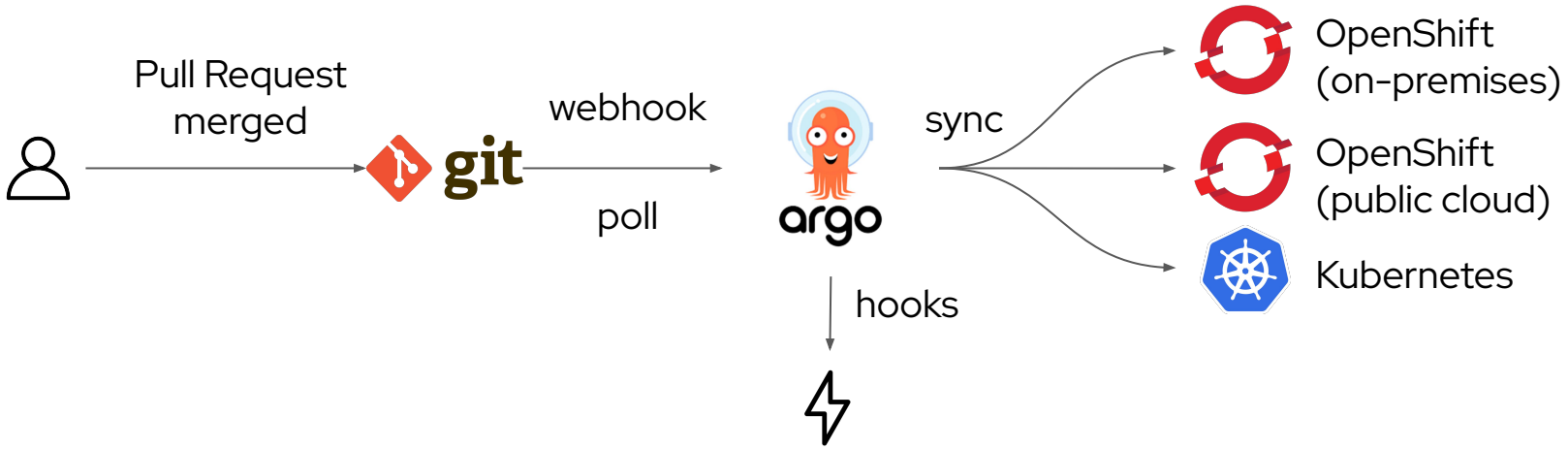
Red Hat

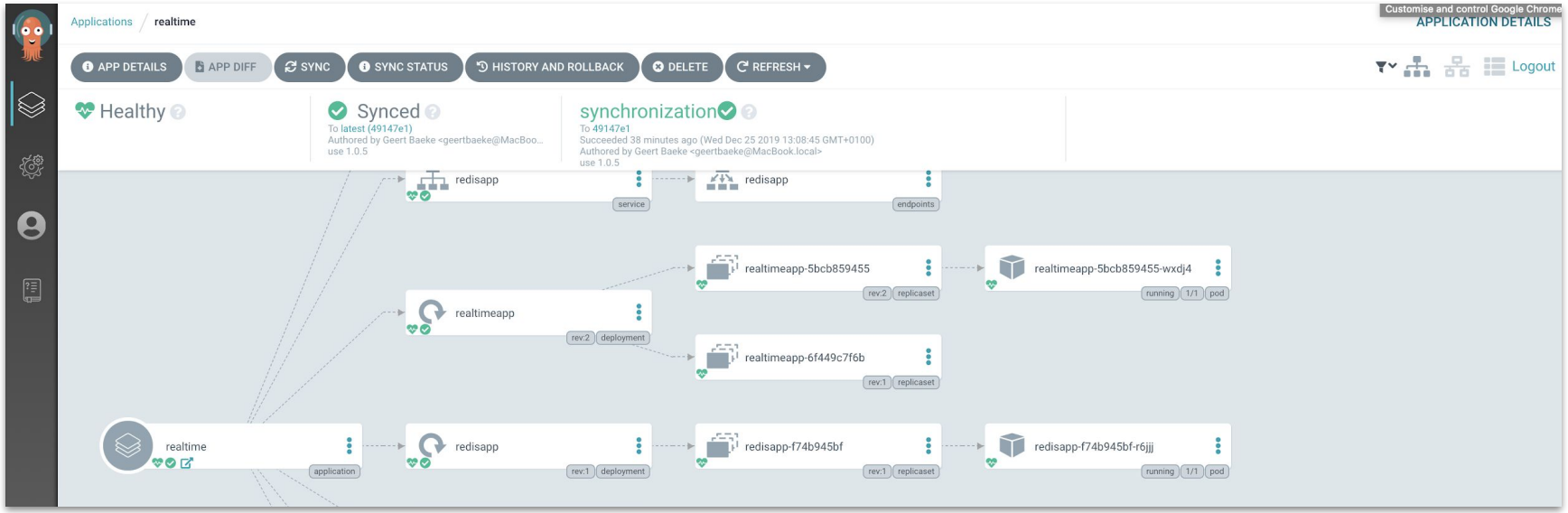
Argo CD

- Cluster and application configuration versioned in Git
- Automatically syncs configuration from Git to clusters
- Drift detection, visualization and correction
- Granular control over sync order for complex rollouts
- Rollback and rollforward to any Git commit
- Manifest templating support (Helm, Kustomize, etc)
- Visual insight into sync status and history



Argo CD





Red Hat

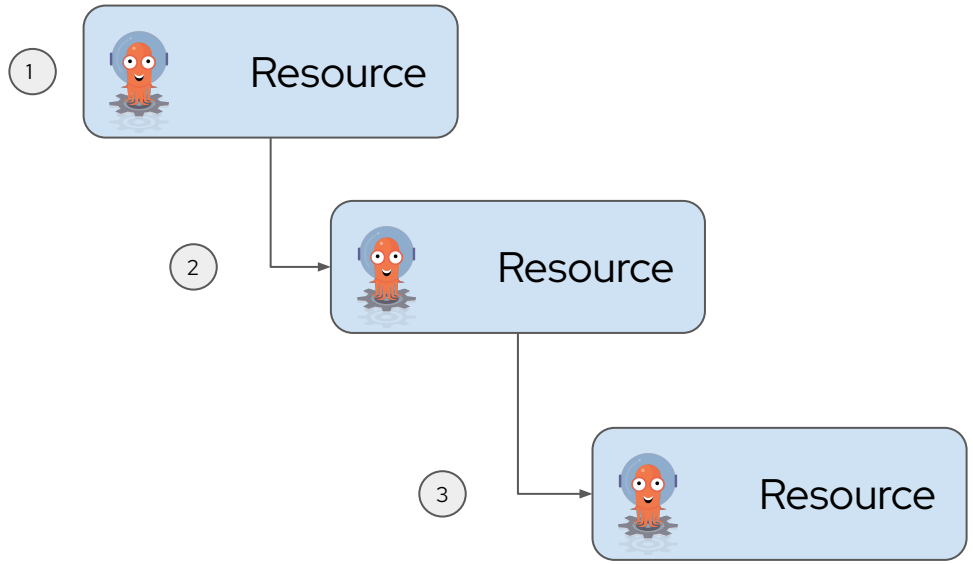
What is an Argo CD Application?

- Argo CD Application is a Custom Resource (CR) that defines the app in a declarative manner
- Application definition includes:
  - Name
  - Cluster
  - Git repository
  - Synchronization Policy
- Applications can be deployed from Argo CD GUI or CLI (argocd or kubectl or oc)

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
 name: product-catalog-dev
 namespace: argocd
spec:
 destination:
  namespace: argocd
  server: https://kubernetes.default.svc
 project: product-catalog
 source:
  path: manifests/app/overlays/dev-quay
  repoURL: https://github.com/gnunn-gitops/product-catalog.git
  targetRevision: master
 syncPolicy:
  automated:
   prune: false
   selfHeal: false
```

Argo CD Sync Waves

- Sometimes you need to deploy resources in a specific order due to dependencies, pre-reqs, etc
- Argo CD enables you to order the deployment using Sync Waves. Next resource will not deploy until the first resource is “healthy”
- Very useful when resources have hard dependencies versus eventual consistency



GitOps - Avoiding Duplication

GitOps enables deployment across multiple environments and clusters, awesome!

Wait, how do we manage configuration without copying and pasting yaml everywhere?

Helm is a package manager for Kubernetes applications that manages manifests via **templating**

define, install and update applications

Kustomize is a tool for customizing Kubernetes manifests via **patching**, it is built natively into the “kubectl/oc” cli with a standalone “kustomize” cli for advanced features.

What does it look like?

Kustomization

A file that declares any resources, and any customization to apply to them, e.g. add a common label or namespace

Overlay

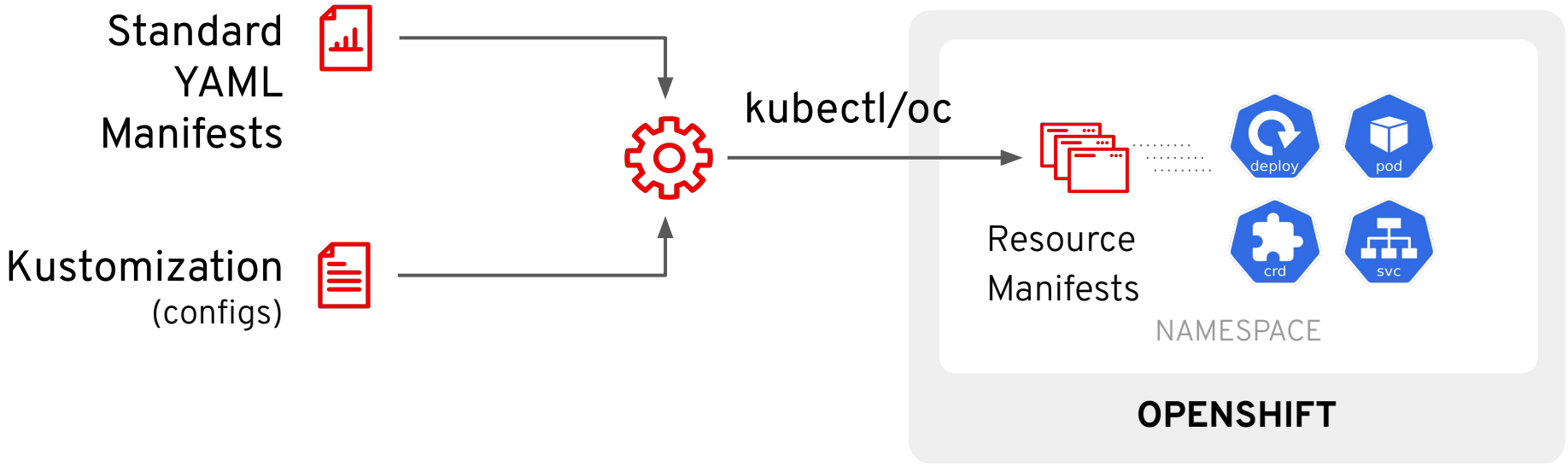
Manage variants of a configuration using overlays that modify a common base.

Example folder structure

```
~/someApp
├── base
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   └── service.yaml
└── overlays
    ├── development
    │   ├── cpu_count.yaml
    │   ├── kustomization.yaml
    │   └── replica_count.yaml
    └── production
        ├── cpu_count.yaml
        ├── kustomization.yaml
        └── replica_count.yaml
```

Red Hat

How does it work?



Thank you

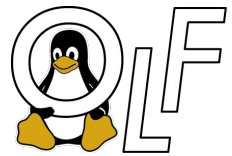
Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat



Red Hat

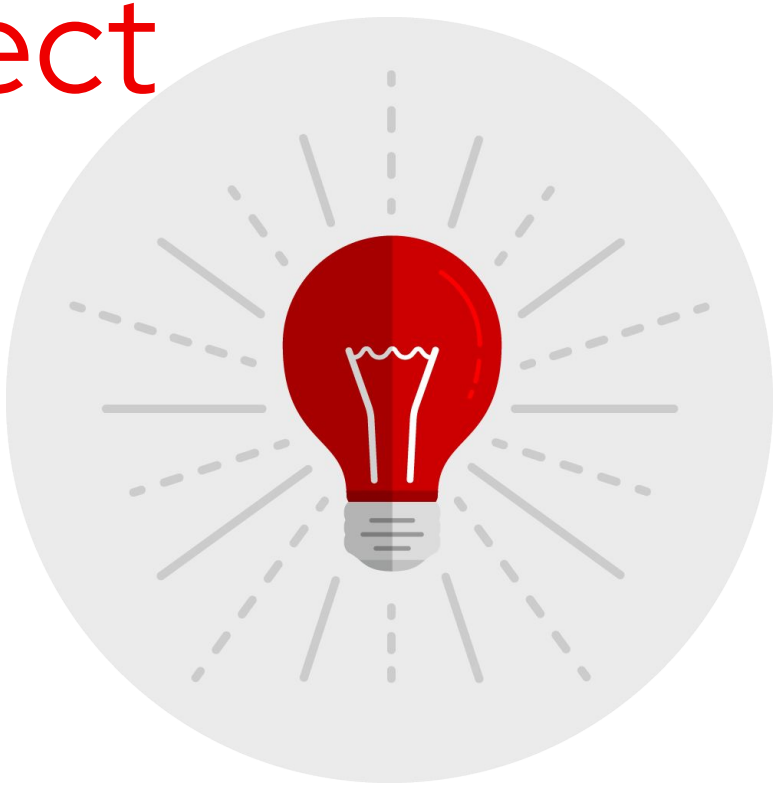
Red Hat Summit Connect

Cincinnati - Sept 14th

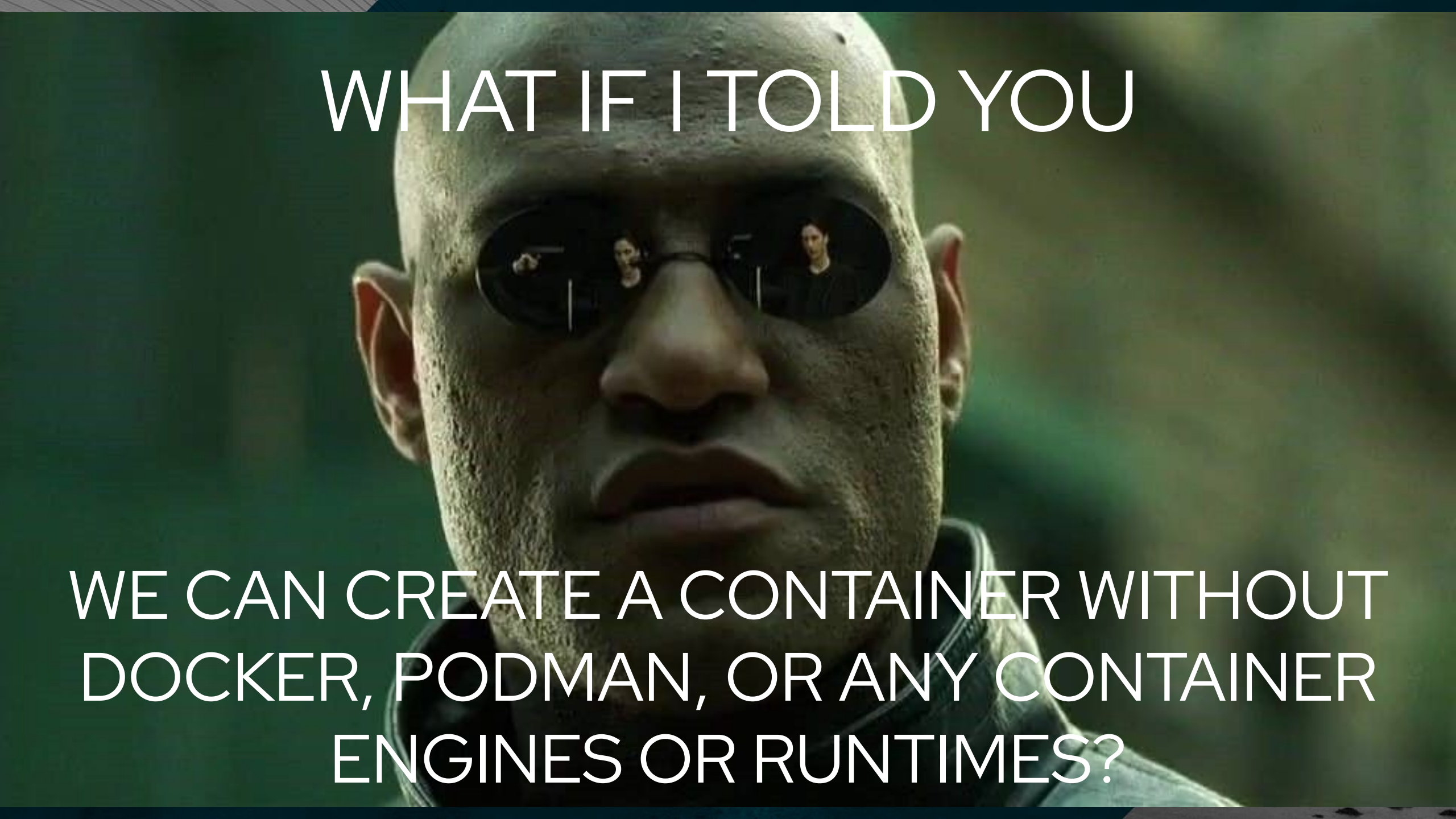
Other Cities too!

Hands On Labs

- Openshift Virtualization
- Event Driven Ansible



CONTAINERS FROM SCRATCH


WHAT IF I TOLD YOU
WE CAN CREATE A CONTAINER WITHOUT DOCKER, PODMAN, OR ANY CONTAINER ENGINES OR RUNTIMES?

How do we make one?

- It all starts with a process
- The we wrap some Linux/Unix boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - Limit resource usage
  - Prioritize or de-prioritize CPU share and Disk I/O
  - Control freezing, stopping, starting and checkpointing
  - Account for usage of resources managed above
  - Limit what/who they can talk to
  - Limit what they can say

Namespaces

- It all starts with a process
- The we wrap some boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - Limit resource usage
  - Prioritize or de-prioritize CPU share and Disk I/O
  - Control freezing, stopping, starting and checkpointing
  - Account for usage of resources managed above
  - Limit what/who they can talk to (MAC)
  - Limit what they can say

CGroups

- It all starts with a process
- The we wrap some boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - Limit resource usage
  - Prioritize or de-prioritize CPU share and Disk I/O
  - Control freezing, stopping, starting and checkpointing
  - Account for usage of resources managed above
  - Limit what/who they can talk to (MAC)
  - Limit what they can say

SELinux / AppArmor

- It all starts with a process
- The we wrap some boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - Limit resource usage
  - Prioritize or de-prioritize CPU share and Disk I/O
  - Control freezing, stopping, starting and checkpointing
  - Account for usage of resources managed above
  - Limit what/who they can talk to (MAC)
  - Limit what they can say

Stop Disabling SELINUX!

https://stopdisablingselinux.com/

#MakeSELINUXenforcingAgain
#SETENFORCE1

SECCOMP

- It all starts with a process
- The we wrap some boundaries around it that....
  - Partition kernel resources (filesystem, IPC, network, PID, users, more)
  - Limit resource usage
  - Prioritize or de-prioritize CPU share and Disk I/O
  - Control freezing, stopping, starting and checkpointing
  - Account for usage of resources managed above
  - Limit what/who they can talk to (MAC)
  - Limit what they can say

LAB

CONTAINERS FROM SCRATCH (~20 min)

https://red.ht/olf-container