Cloud Native Applications Workshop

What is Cloud Native?

WW Developer Advocacy

Contents

- App Modernization
- Docker Overview
- Kubernetes Overview
- OpenShift Overview
- 12 Factor Apps

App Modernization is inevitable

Evolution of application architectures

Late 90's	**Enterprise Application (EAI) Services and Models** Addressed integration and transactional challenges primarily by using message oriented middleware. Mostly proprietary systems needing a proliferation of custom interfaces.
Mid 00's	**Service Oriented Architectures** Based on open protocols like SOAP and WSDL making integration and adoption easier. Usually deployed on an Enterprise ESB which is hard to manage and scale.
Early 10's	**API Platforms and API Management** REST and JSON become the defacto standard for consuming backend data. Mobile apps become major consumers of backend data. New Open protocols like OAuth become available further simplifying API development .
2015 and beyond	**Cloud Native and Microservice Architecture** Applications are composed of small, independently deployable processes communicating with each other using language-agnostic APIs and protocols.

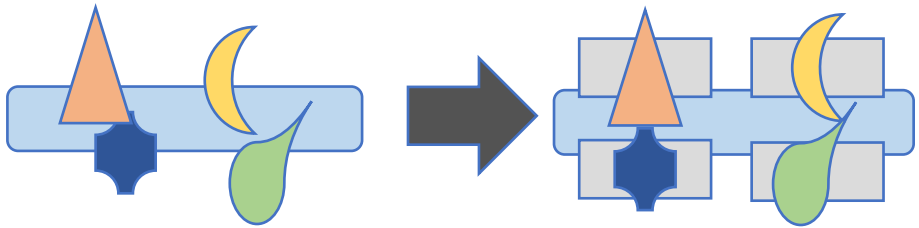
Key tenets of a cloud native application

1. Packaged as light weight **containers**
2. Developed with best-of-breed languages and frameworks
3. Designed as loosely coupled **microservices**
4. Centered around **APIs** for interaction and collaboration
5. Architected with a clean separation of stateless and stateful services
6. Isolated from server and operating system dependencies
7. Deployed on self-service, elastic, **cloud infrastructure**
8. Managed through agile **DevOps** processes
9. Automated capabilities
10. Defined, policy-driven resource allocation

https://thenewstack.io/10-key-attributes-of-cloud-native-applications/

Key tenets of a microservices architecture

1. Large monoliths are broken down into many small services
2. Services are optimized for a single function or business capability
3. Teams that write the code should also deploy the code
4. Smart endpoints, dumb pipes (message brokers)
5. Decentralized governance
6. Decentralized data management

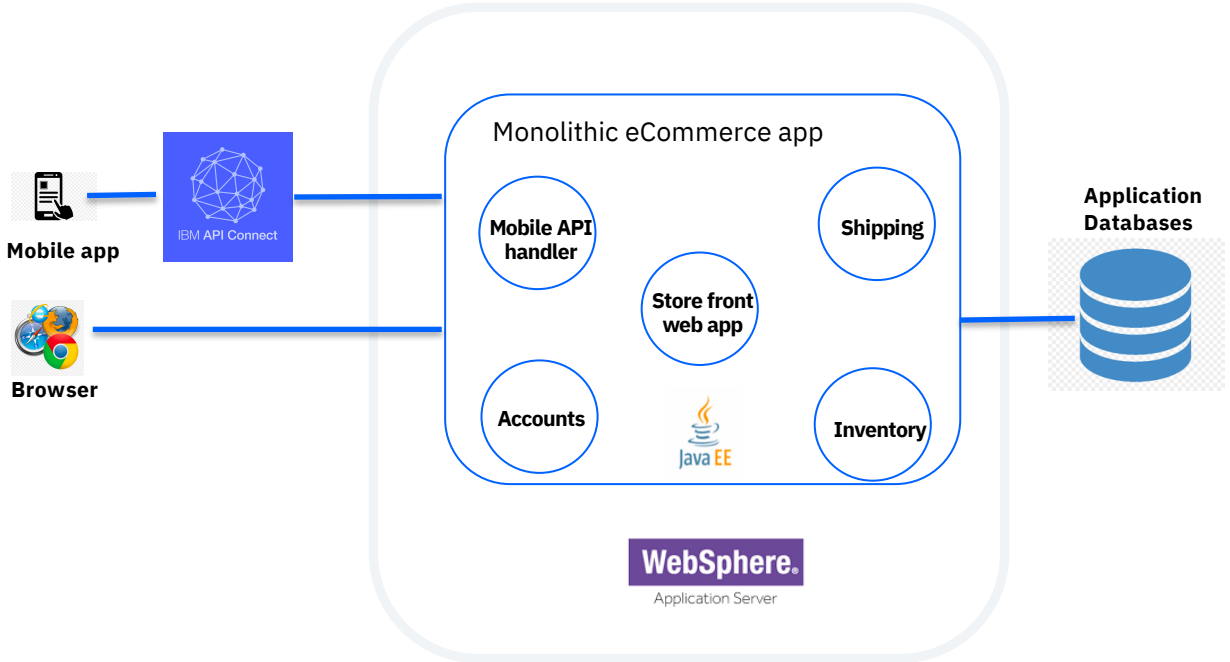


https://martinfowler.com/articles/microservices.html

Example monolithic application

eCommerce app

- Store front web interface
- Customer Accounts
- Inventory
- Shipping
- Back end for mobile app

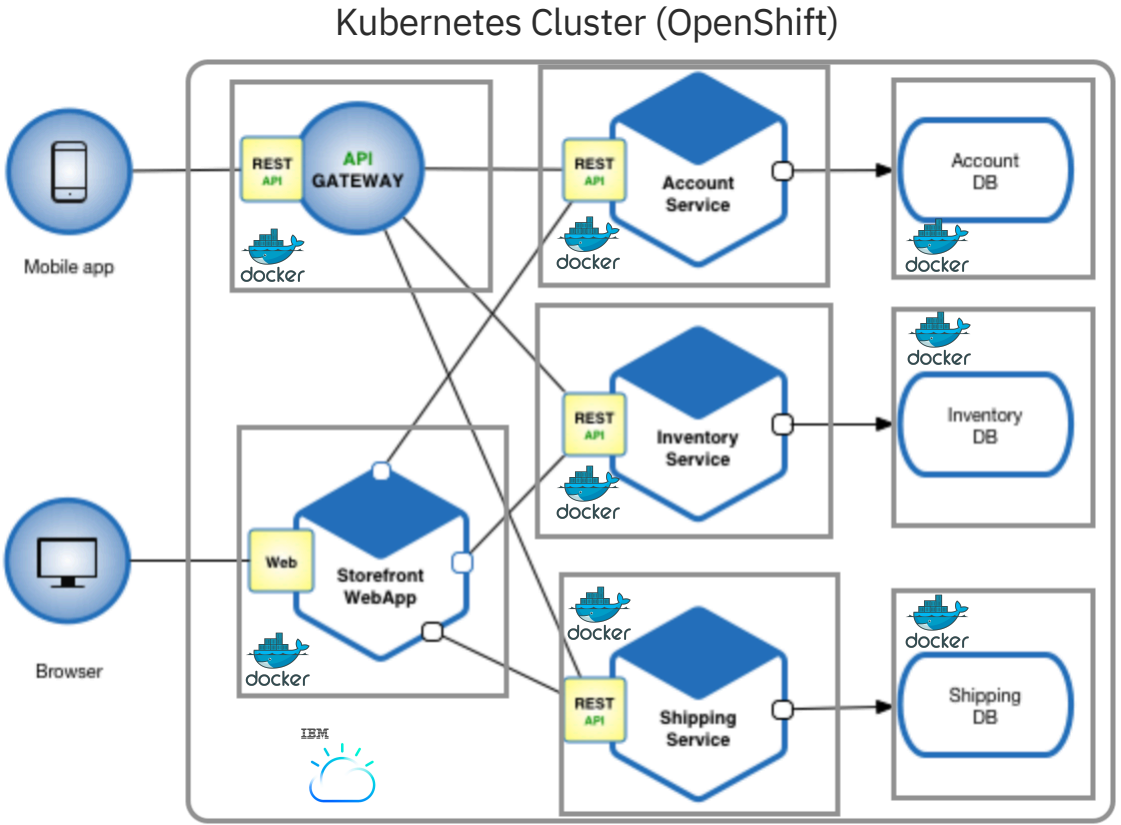


An eCommerce Java EE app on Websphere

Transformed application

Key technologies

- Containers (Docker)
- Container orchestration (Kubernetes)
- Transformation Advisor
- 12-Factor Best Practices
- CI/CD tools (e.g Jenkins)



An eCommerce microservices app on a Kubernetes cluster

Why microservices and cloud native?

Efficient teams	Simplified deployment	Right tools for the job	Improved application quality	Scalability
• End to end team ownership of relatively small codebases	• Each service is individually changed, tested, and deployed without affecting other services	• Teams can use best of breed technologies, libraries, languages for the job at hand	• Services can be tested more thoroughly in isolation	• Services can be scaled independently at different rates as needed
➢Teams can innovate faster and fix bugs more quickly	➢Time to market is accelerated.	➢Leads to faster innovation	➢Better code coverage	➢Leads to better overall performance at lower cost

Cultural change considerations

- **Smaller teams with broader scope**
  - Mini end to end development orgs in each team vs large silos across the entire development team
- **Top down support with bottom up execution**
  - Change can't happen effectively w/o executive sponsorship
  - Change needs to be executed at the smallest organizational unit to take hold
- **Teams own all metrics related to operations and development**
  - Have to minimize downtime + number of bugs while also maximizing the rate at which needed features are added and minimizing the time to market of those new features
- **Trust**
  - Teams need to build trust with other teams that they collaborate with rather than relying on one size fits all checklists and rules
- **Reward based on results not compliance**
  - Cultures only change when people are measured and rewarded for outcomes consistent with the changes
  - Smaller more autonomous teams work better with less central micromanagement and more focus on broad measurable goals

Docker Overview

Containers provide process isolation.

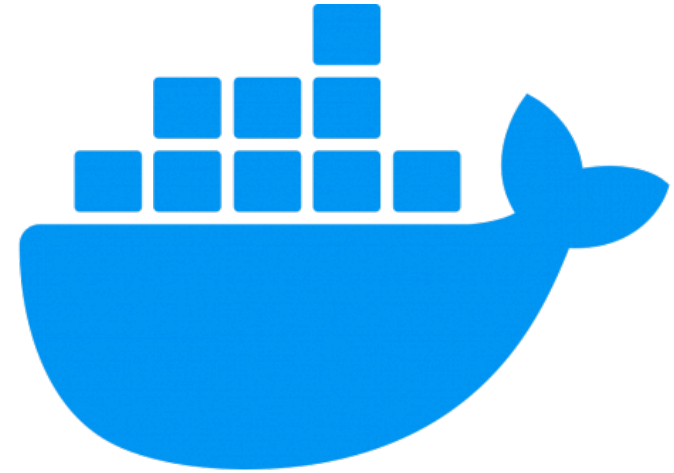
Docker is a container runtime.

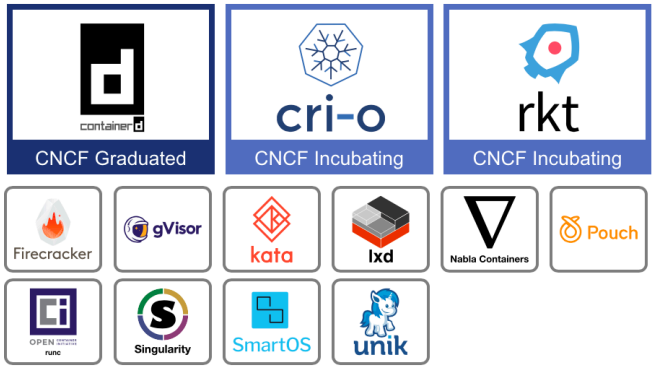
Docker and containers

- **Docker** is one of many **container runtimes**
  - containerD, cri-o, rkt, are other examples

- Docker has the **best developer tools**
- Other runtimes more suited for **production**

- Containers provide **process isolation**
- Containers are **not virtual machines**

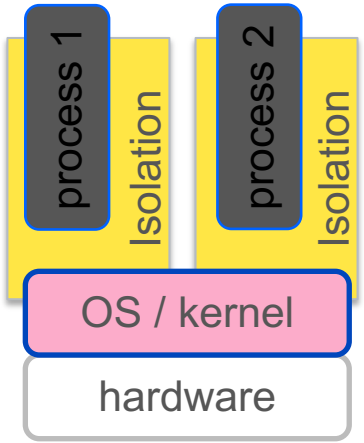




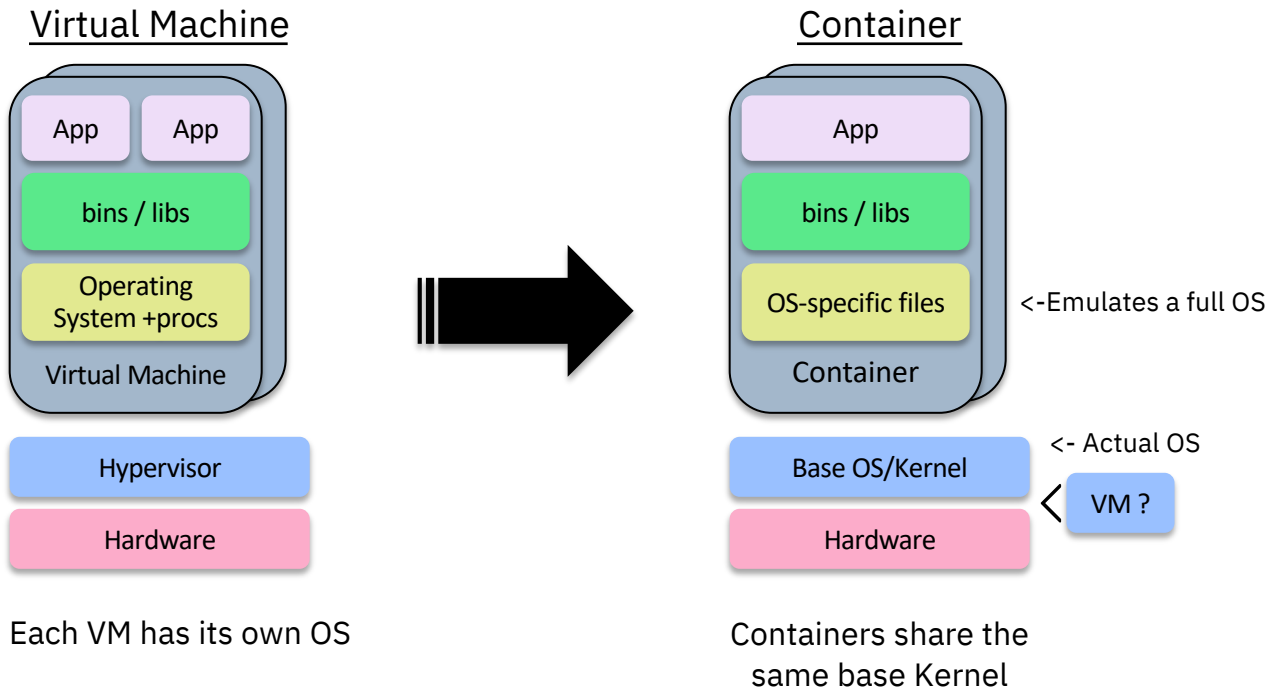
Containers are not a new idea

- Other process isolation technologies:
  - chroot ('80s) process spawned in isolated file space
  - FreeBSD jails
  - OS-level virtualization (user-mode-linux, virtuozzo)
  - Solaris Containers
  - LinuX Containers (LXC)
  - Cloud Foundry (Warden, Garden)

- Docker provided an ecosystem approach that transformed perception
  - Building application-centric containers
  - Mechanism for sharing images (Docker Registry)
  - Open-source enabled



VM vs Container



Our First Container

```
$ docker run ubuntu echo Hello World
Hello World

$ docker run -ti ubuntu bash
root@62deec4411da:/# pwd
/
```

Docker Images and Image Registry

Docker Images

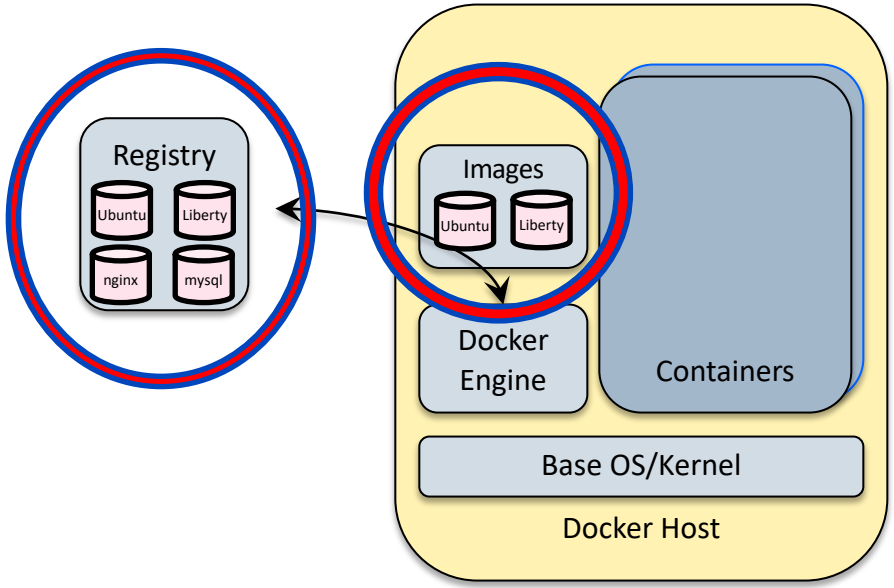
- Tar file containing a container's filesystem and metadata

Docker Registry

- The central place to share images
- Enterprises will want to use a private registry
- OpenShift has a built-in registry

DockerHub - http://hub.docker.com

- Public registry of Docker Images
- Also useful to find prebuilt images for web servers, databases, and much more



Build your own image with a Dockerfile

1. Create a **Dockerfile** to script how you want the image to be built
2. Run **docker build** to build an image
3. Run **docker run** to run it locally
4. Run **docker push** to push it to a registry
5. Run **docker pull** to download an image

```
FROM java:8 # This might be an ubuntu or...
COPY *.jar app.jar
CMD java -jar app.jar
```

Are you testing these on every commit?

Traditional Deployment (Dev, Staging, Prod)

1. Code ✓
2. App server ✗
3. Runtime versions ✗
4. System libraries and versions ✗

Containerized Deployment

1. Code ✓
2. App server ✓
3. Runtime versions ✓
4. System libraries and versions ✓

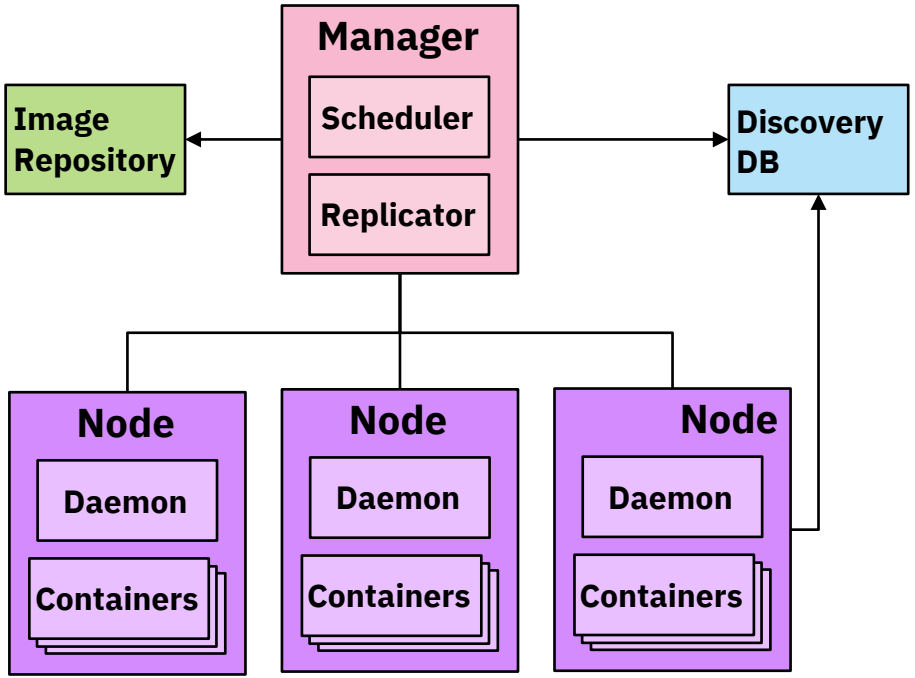
Kubernetes Overview

Container orchestration with Kubernetes unlocks value of containers as application components

What is container orchestration?

Container Orchestration handles the following:

- Cluster management
- Scheduling
- Service discovery
- Replication
- Health management



Kubernetes and Container Orchestration

Kubernetes is a Container Orchestrator

- Provisions, manages, and scales containerized applications

- Supports:
  - Automated scheduling and scaling
  - Zero downtime deployments
  - High availability and fault tolerance
  - A/B deployments

- Manage infrastructure resources needed by applications
  - Volumes
  - Networks
  - Secrets
  - And many many many more..

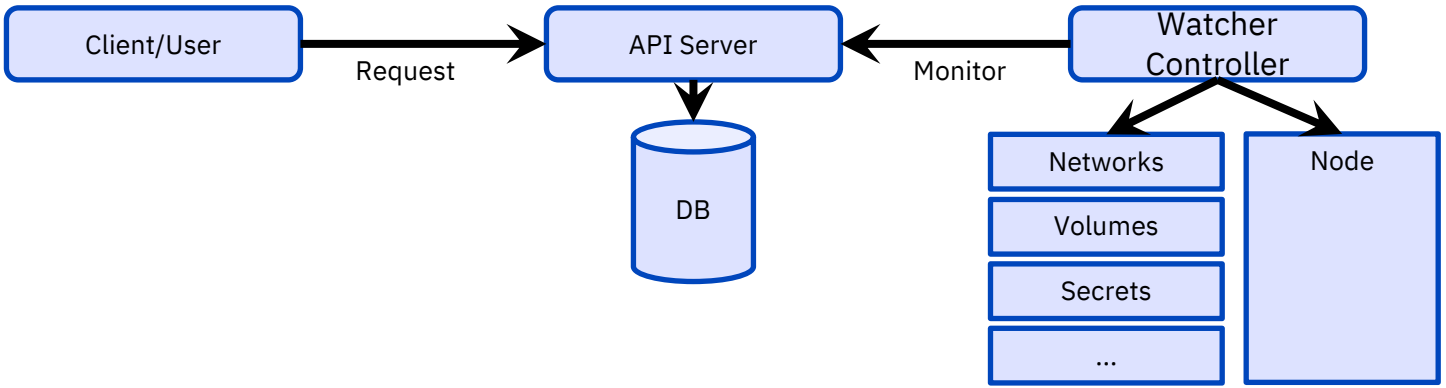
- Declarative model
  - Provide the "desired state" and Kubernetes will make it happen

Kubernetes fun facts:

- Builds upon 15 years of experience of running production workloads at Google
- Open Governance via CNCF
- Adopted by IBM, Amazon, Microsoft, Red Hat, Google,
- Means “helmsman” in Greek

Kubernetes Architecture

- At its core, Kubernetes is a database (etcd) with "watchers" & "controllers" that react to changes in the DB.
  The controllers are what make it Kubernetes. This extensibility is part of its "secret sauce".
- The database represents the user's desired state. Watchers attempt to make reality match the desired state
- The API server is the HTTP/REST front-end to the DB



Kubernetes Resource Model

A **resource** for every purpose

- Config Maps
- Daemon Sets
- **Deployments**
- Events
- Endpoints
- Ingress
- Jobs
- Nodes
- Namespaces
- **Pods**
- Persistent Volumes
- Replica Sets
- Secrets
- Service Accounts
- **Services**
- Stateful Sets, and more...

- Kubernetes aims to have the building blocks on which you build a cloud native platform.

- Therefore, the internal resource model **is** the same as the end user resource model.

Key Resources

- Pod: set of co-located containers
  - Smallest unit of deployment
  - Several types of resources to help manage them
  - Replica Sets, Deployments, Stateful Sets, ...

- Services
  - Define how to expose your app as a DNS entry
  - Query based selector to choose which pods apply

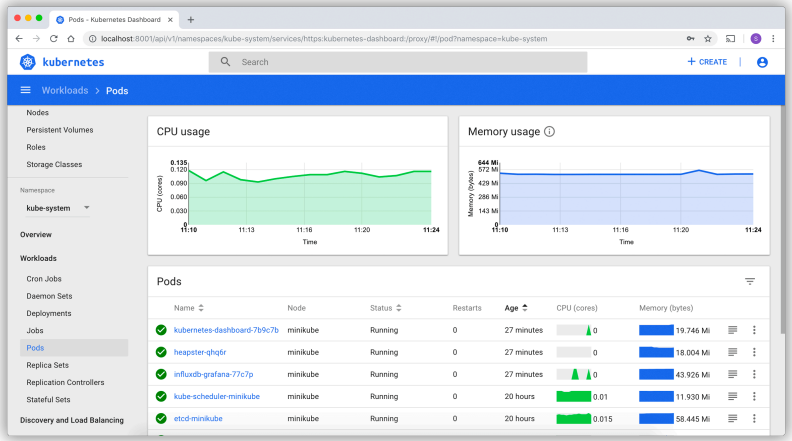
Kubernetes Clients (CLI and Dashboard)

Kubernetes CLI

- Directly manipulate YAML
  - `kubectl (create|get|apply|delete) -f myResource.yaml`
- https://kubernetes.io/docs/tasks/tools/install-kubectl

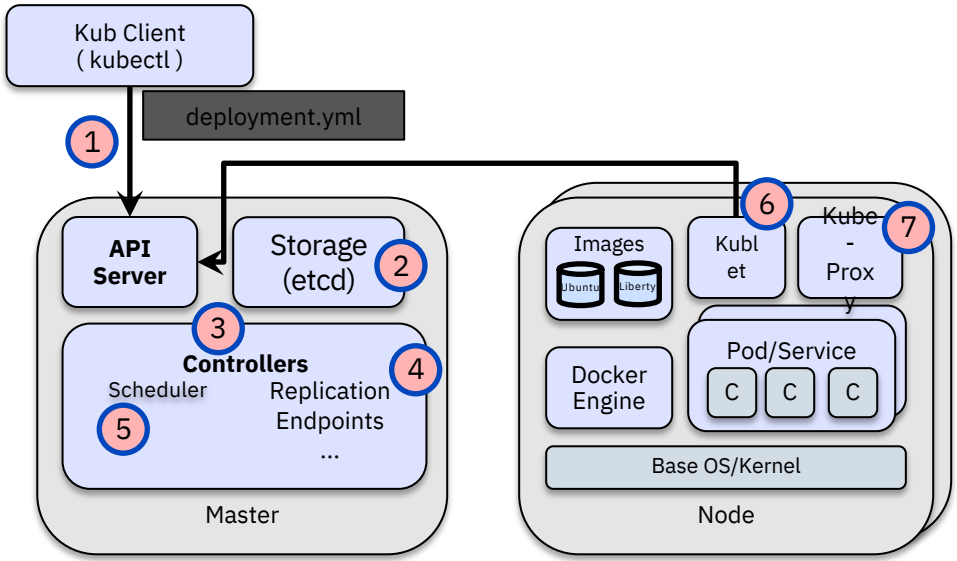
Kubernetes Dashboard

- Another way to view and modify resources



Kubernetes in Action!

1. User via "**kubectl**" deploys a new application
2. API server receives the request and stores it in the DB (etcd)
3. Watchers/controllers detect the resource changes and act upon it
4. ReplicaSet watcher/controller detects the new app and creates new pods to match the desired # of instances
5. Scheduler assigns new pods to a kubelet
6. Kubelet detects pods and deploys them via the container runtime (e.g. Docker)
7. Kubeproxy manages network traffic for the pods – including service discovery and load-balancing

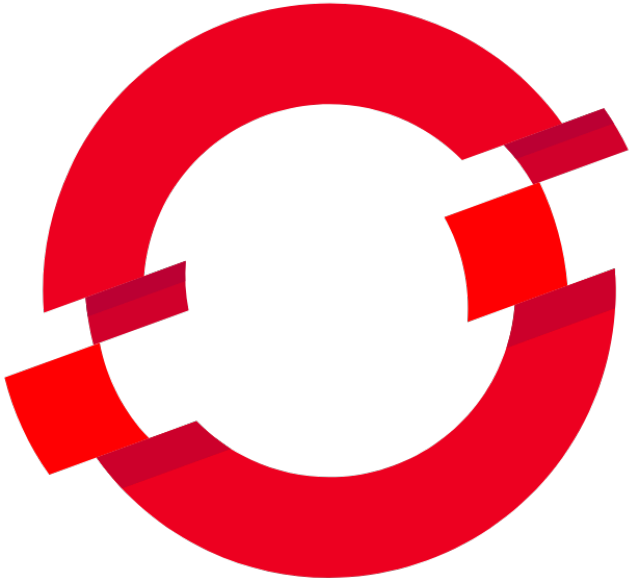


OpenShift Overview

Red Hat OpenShift is an enterprise-ready Kubernetes container platform

OpenShift key points

- OpenShift is Red Hat's Kubernetes platform
- Runs on RHEL (Red Hat Enterprise Linux)
- Provides built-in security for container-based applications
  - Role-based access controls
  - Security-Enhanced Linux (SELinux)-enabled isolation
- Supports a hybrid cloud architecture
  - OpenShift runs on Amazon Web Services, Azure, Google Cloud Platform, VMware, and more
- Current latest release: v4.1
- IBM Cloud uses v3.11
- v3 uses Docker, v4 uses CRIO



OPENSHIFT

Container runtime

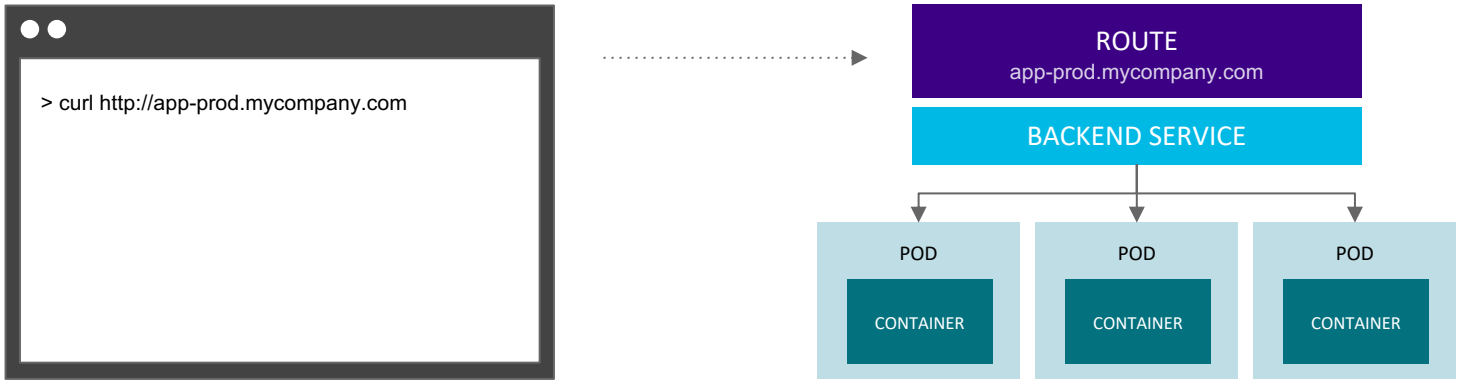
cri-o

A lightweight, OCI-compliant container runtime

Minimal and Secure Architecture	Optimized for Kubernetes	Runs any OCI-compliant image (including docker)

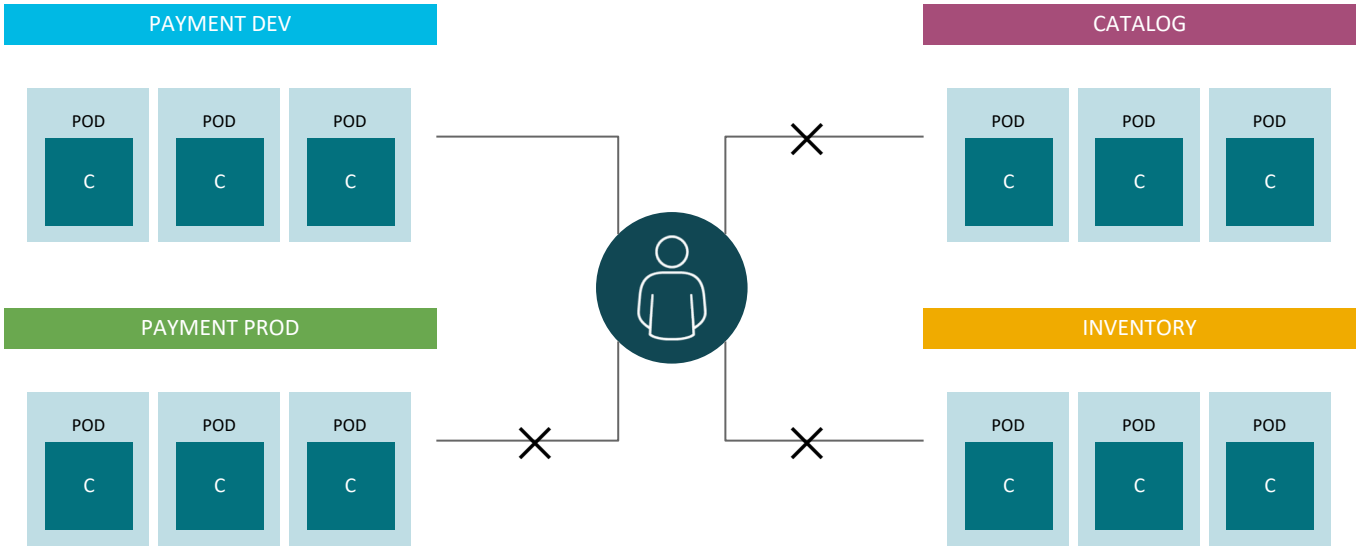
Routes

routes add services to the external load-balancer and provide readable URLs for the app

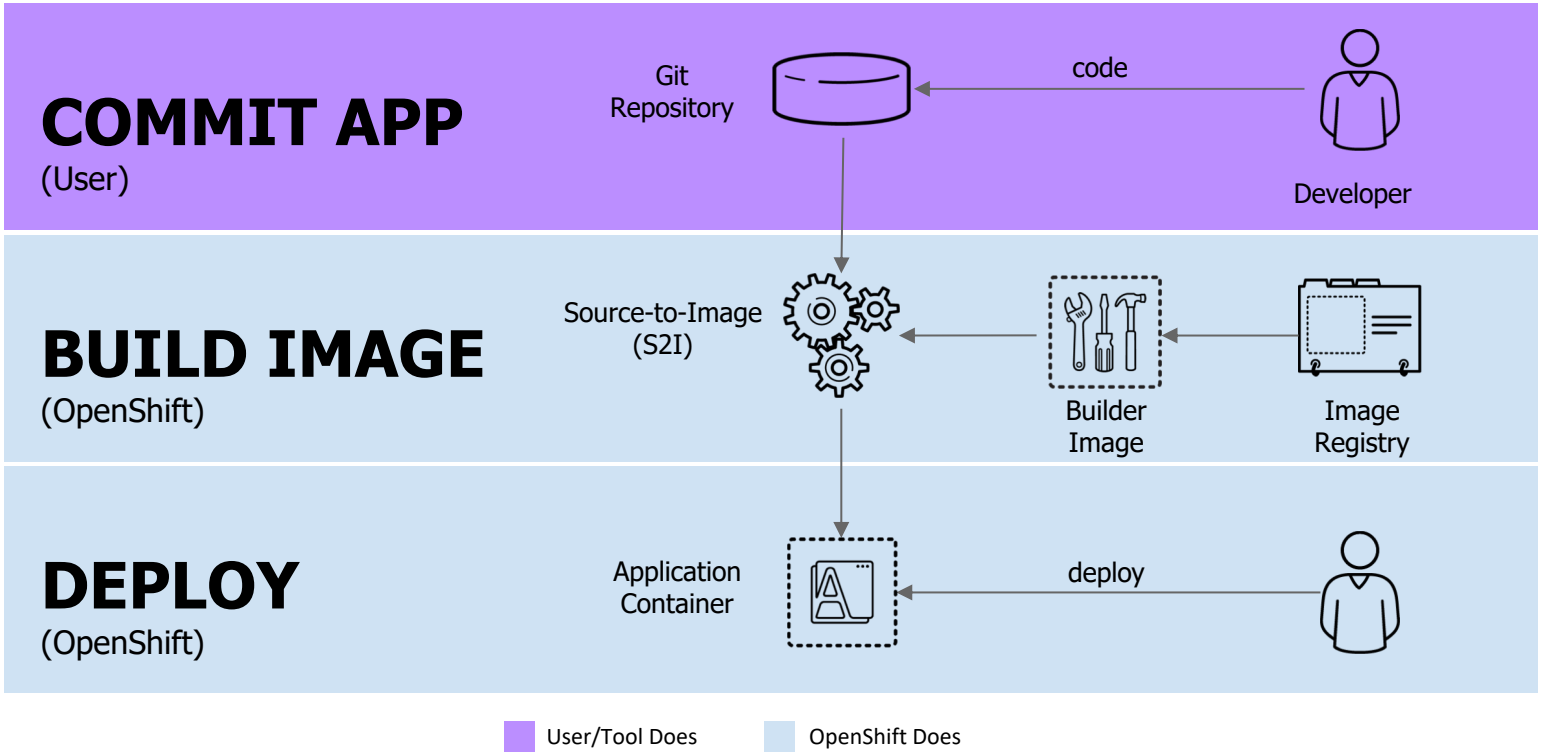


Projects extend k8s namespaces

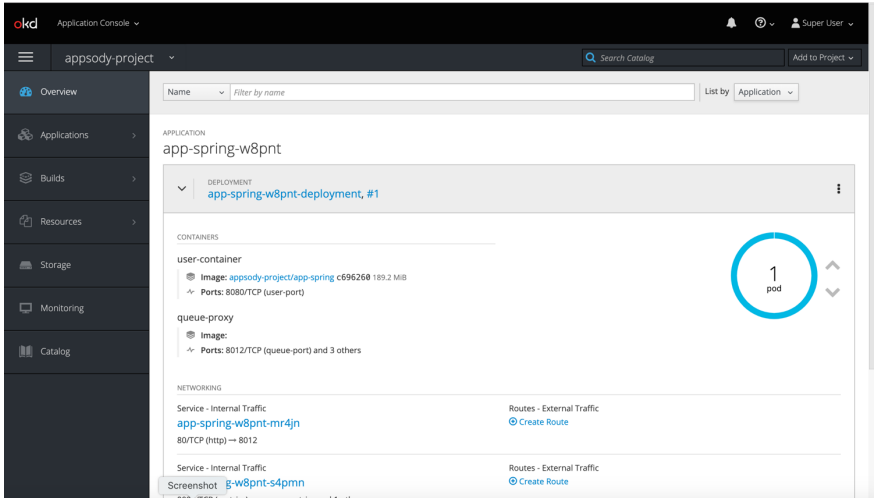
projects isolate apps across environments, teams, groups and departments



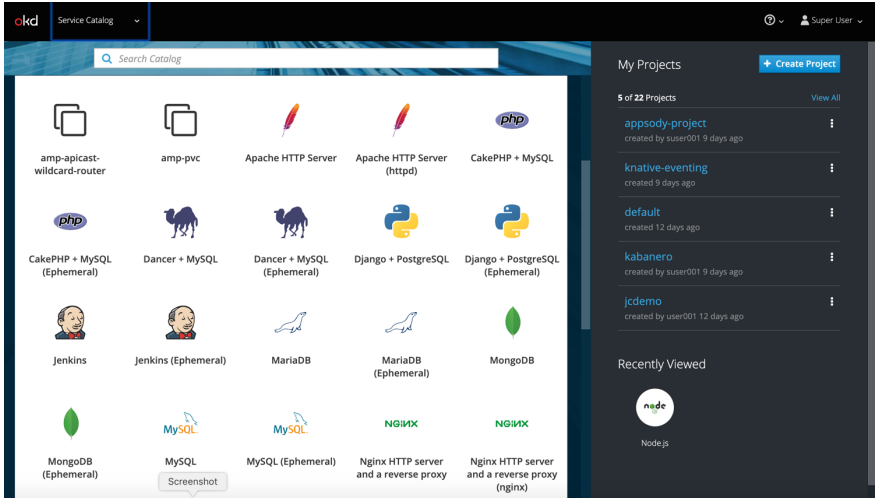
Source-to-Image



Web Console



Application dashboard



Service Catalog

12 Factor Apps

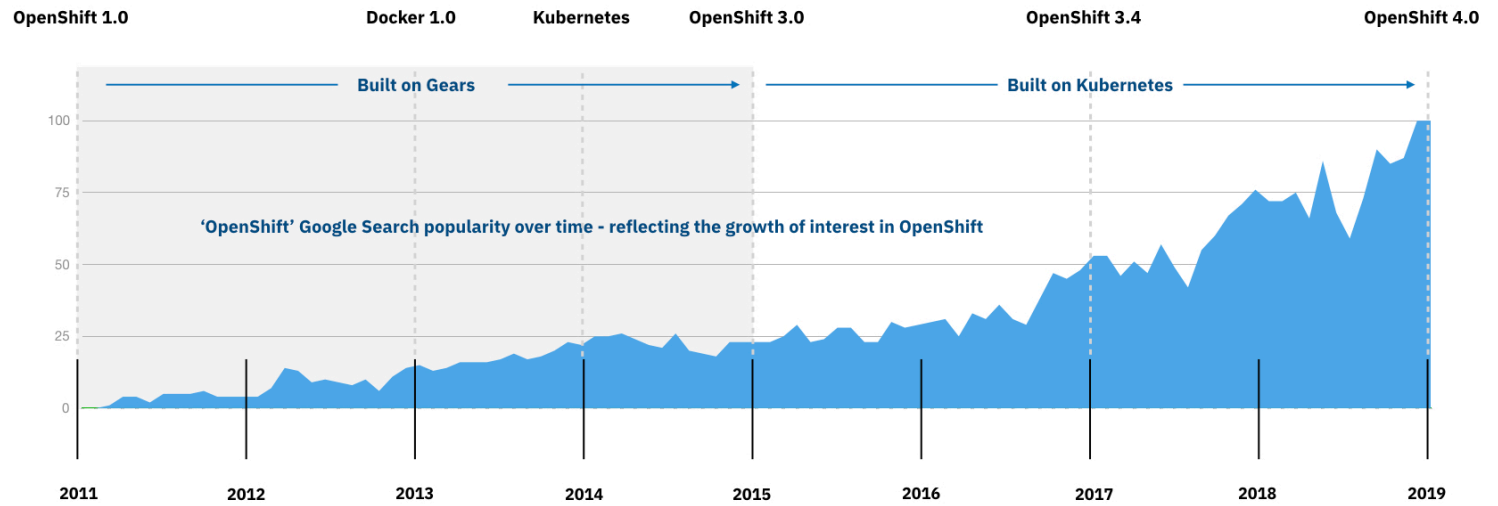
12 Factor is a methodology for building software

Tenets for a 12 Factor App

1. Codebase
2. Dependencies
3. Config
4. Backing Services
5. Build, Release, Run
6. Processes
7. Port Binding
8. Concurrency
9. Disposability
10. Dev/Prod Parity
11. Logs
12. Admin processes

https://12factor.net

OpenShift history: v1 – v4



Linux, building on Red Hat's heritage and deep expertise in Linux and the reliability of **Red Hat Enterprise Linux** which served as the foundation for OpenShift 3

RedHat and Docker announced a collaboration around

Fedora RedHat Enterprise Linux and OpenShift

IBM begin contributing code to the Docker Open Source Project

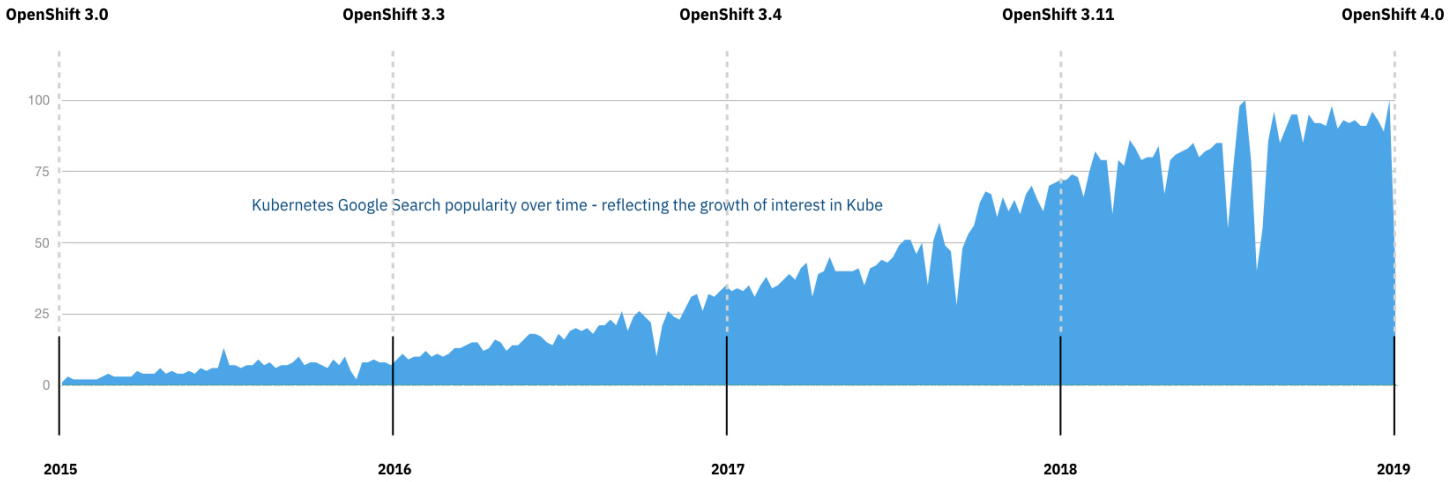
Kubernetes (κυβερνήτης, Greek for "governor", "helmsman" or "captain")[3] was founded by Joe Beda, Brendan Burns and Craig McLuckie,[6] who were quickly joined by other Google engineers including Brian Grant and Tim Hockin, and was first announced by Google in mid-2014.[7] Its development and design are heavily influenced by Google's Borg system.

IBM contributing code to the Kubernetes Open Source Project

Kubernetes for the Enterprise

includes Kubernetes core services, along with Prometheus, Grafana, Elasticsearch, software defined networking, storage, registry and other components that make up the OpenShift Kubernetes platform.

OpenShift history: v3 – v4



Linux, building on Red Hat's heritage and deep expertise in Linux and the reliability of **Red Hat Enterprise Linux** which served as the foundation for OpenShift 3

Containers, designed to provide efficient, immutable and standardized application packaging that enables application portability across a hybrid cloud environment

Kubernetes, providing powerful container orchestration and management capabilities and becoming one of the fastest growing open source projects of the last decade

User Experience additions with a new web console for editing YAML files and for the container registry.

Build automation and app deployment a beloved feature of OpenShift 3.0 now comes

Build pipelines and web hooks - paving the way for integrated GitHub repository deployment.

RedHat second only to Google for **Kubernetes open source commits**

Filtering, Sorting, Managing Projects: To make things easier for people with a large number of projects, the project list now has a text filter on name, display name, description, and project creator. It also allows sorting on several of these attributes.

Identity and Access management

Filtering and Sorting, Managing Projects: To make things easier for people with a large number of projects, the project list now has a text filter on name, display name, description, and project creator. It also allows sorting on several of these attributes.

Prometheus Monitoring, CoreOS and Quay.

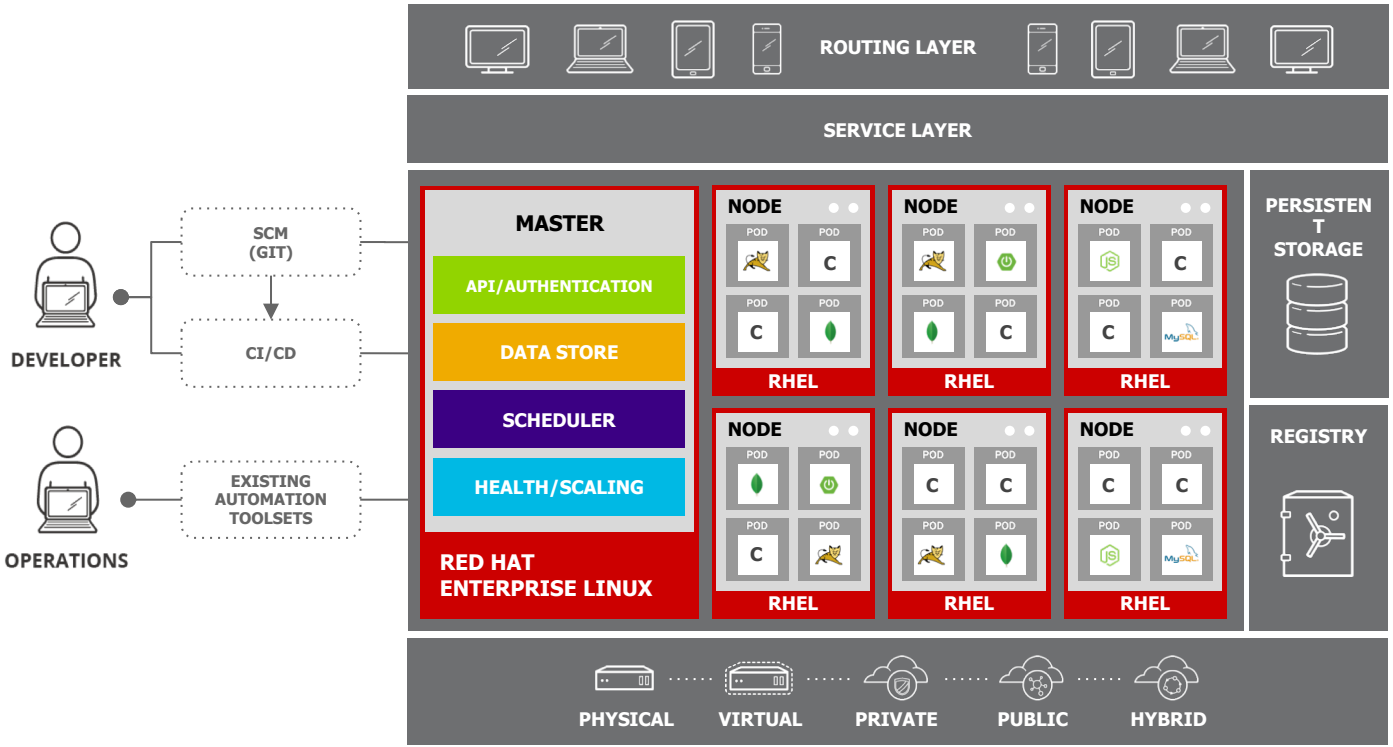
Operator Framework single-step installation for Kubernetes applications and services, and automated, over-the-air updates and performance tuning.

OpenShift Multi-Cluster Manager allows you to manage multiple clusters, running across multiple clouds or on-premise environments

OpenShift Service Mesh - Istio in OpenShift 4 through what's called the OpenShift service mesh

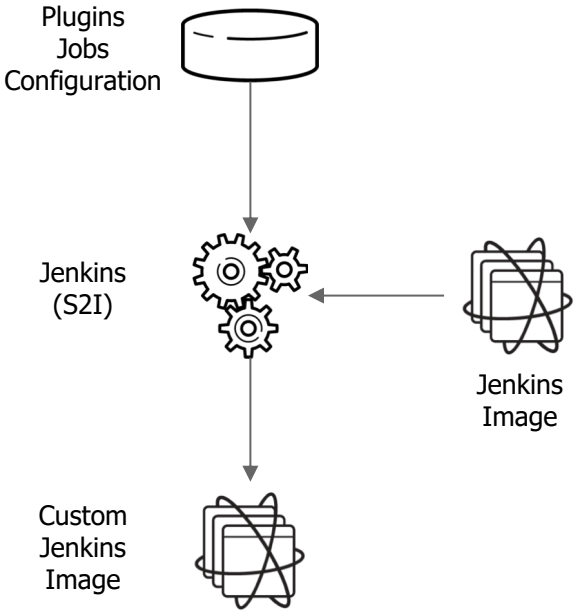
Knative Framework - a new developer-friendly serverless framework for building, serving and running event-driven applications.

OpenShift Architecture



Jenkins-as-a-Service

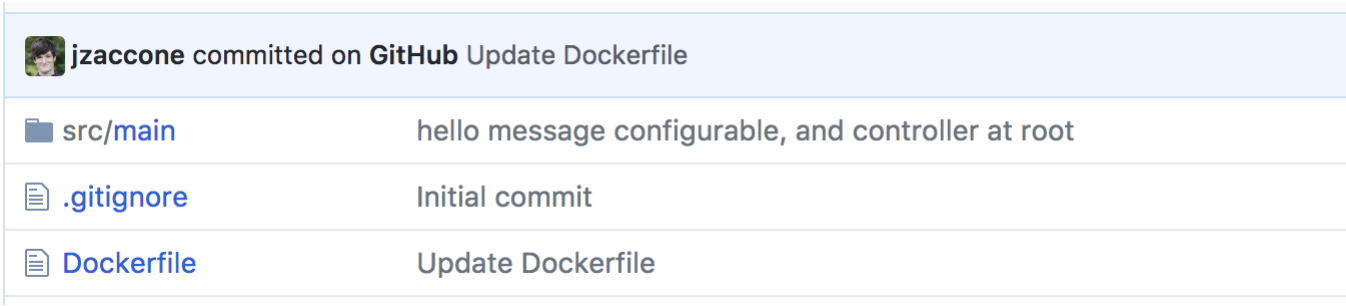
- Certified Jenkins images with pre-configured plugins
  - Provided out-of-the-box
  - Follows Jenkins 1.x and 2.x LTS versions
- Jenkins S2I Builder for customizing the image
  - Install Plugins
  - Configure Jenkins
  - Configure Build Jobs
- OpenShift plugins to integrate authentication with OpenShift and also CI/CD pipelines
- Dynamically deploys Jenkins slave containers



I. Codebase

Code for a single application should be in a single code base

- Track running applications back to a single commit
- Use Dockerfile Maven, Gradle, or npm to manage external dependencies
- Version pinning! Don’t use latest
- No changing code in production



II. Dependencies

Explicitly declare and isolate dependencies. AKA: Remove system dependencies

How?

- Step 1: Explicitly declare dependencies (Dockerfile)
- Step 2: Isolate dependencies to prevent system dependencies from leaking in (containers)

```
1  FROM openjdk:8-jdk-alpine
2  EXPOSE 8080
3  WORKDIR /data
4  CMD java -jar *.jar
5  COPY target/*.jar /data/
```

III. Config

Store config in the environment (not in the code).

How?

- Inject config as environment variables (language agnostic)
- ConfigMap in Kubernetes does this ^

```
$ docker run -e POSTGRES_PASSWORD=abcd postgres
```

IV. Backing Services

Treat backing resources as attached services. Swap out resources.

How?

- Pass in URLs via config (see III.)
- K8s built in DNS allows for easy service discovery

```
services:

  account-api:
    build:
      context: ./compute-interest-api
    environment:
      DATABASE_URL: http//account-database

  account-database:
    image: jzaccone/account-database
```

V. Build, Release, Run

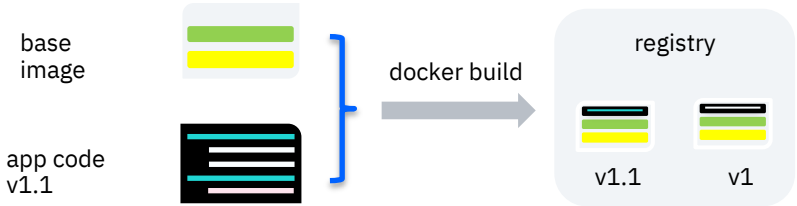
Strictly separate build and run stages.

Why?

Rollbacks, elastic scaling without a new build

How?

- Use Docker images as your handoff between build and run
- Tag images with version. Trace back to single commit (see I. Codebase)
- Single command rollbacks in Kubernetes



VI. Process

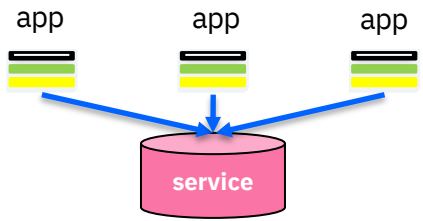
Execute app as stateless process

Why?

Stateless enables horizontal scaling

How?

- Remove sticky sessions
- Need state? Store in volume or external data service
- Use persistent volumes in Kubernetes for network wide storage



VII. Port Binding

Export services via port binding. Apps should be self-contained.

Why?

Avoid “Works on my machine”

How?

- Web server dependency should be included inside the Docker Image
- To expose ports from containers use the —publish flag

VIII. Concurrency

Scale out via the process model. Processes are **first-class citizens**

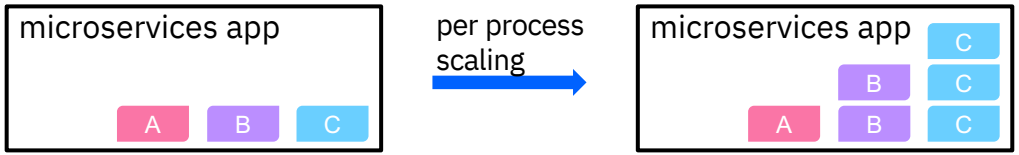
Why?

Follow the Unix model for scaling, which is simple and reliable

How?

- Scale by creating more processes
- **Docker**: really just a process running in isolation
- **Kubernetes**: Acts as process manager: scales by creating more pods

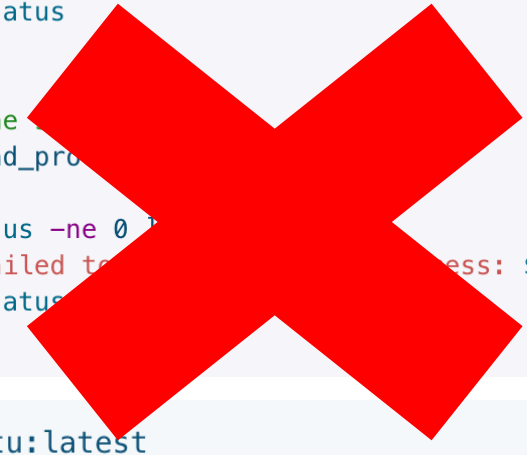
  Don't put process managers in your containers



Bad Example

```
# Start the first process
./my_first_process -D
status=$?
if [ $status -ne 0 ]; then
  echo "Failed to start my_first_process: $status"
  exit $status
fi

# Start the
./my_second_pro
status=$?
if [ $status -ne 0
  echo "Failed t                ess: $status"
  exit $statu
fi
```



```
FROM ubuntu:latest
COPY my_first_process my_first_process
COPY my_second_process my_second_process
COPY my_wrapper_script.sh my_wrapper_script.sh
CMD ./my_wrapper_script.sh
```

Containers should be a single process!

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

Why?

- Enables fast elastic scaling, robust production deployments. Recover quickly from failures.

How?

- multi-minute app startups!
- Docker enables fast startup: Union file system and image layers
- In best practice: Handle SIGTERM in main container process.

X. Dev/Prod Parity

Keep development, staging and production as similar as possible. Minimize time gap, personnel gap and tools gap

How?

- **Time gap:** Docker supports delivering code to production faster by enabling automation and reducing bugs caused by environmental drift.
- **Personnel gap:** Dockerfile is the point of collaboration between devs and ops
- **Tools gap:** Docker makes it very easy to spin up production resources locally by using `docker run ...`

XI. Logs

Treat logs as event streams

How?

- Write logs to stdout (Docker does by default)
- Centralizes logs using ELK or [your tool stack here]

  Don’t

  Don’t retroactively inspect logs! Use ELK to get search, alerts

  Don’t throw out logs! Save data and make data driven decisions

XII. Admin Processes

Run admin/management tasks as one-off processes.

Don’t treat them as special processes

How?

- Follow 12-factor for your admin processes (as much as applicable)
- Option to collocate in same source code repo if tightly coupled to another app
- “Enter” namespaces to run one-off commands via `docker exec ...`