A Journey through expertise...

@RedHat_Summit

Deepak Mishra

(CEO and Founder of Prodevans Technologies)

Graham Dumpleton

(The author of two O'Reilly books on OpenShift: OpenShift for Developers, and Deploying to OpenShift)

AGENDA

1. Overview of container technology
2. Overview of container architecture
3. Overview of Kubernetes and OpenShift
4. Provisioning a containerized database server
5. Building custom container images with Dockerfile
6. Creating basic Kubernetes and OpenShift resources
7. Creating routes
8. Creating applications with the source-to-image facility
9. Creating applications with Red Hat OpenShift web console

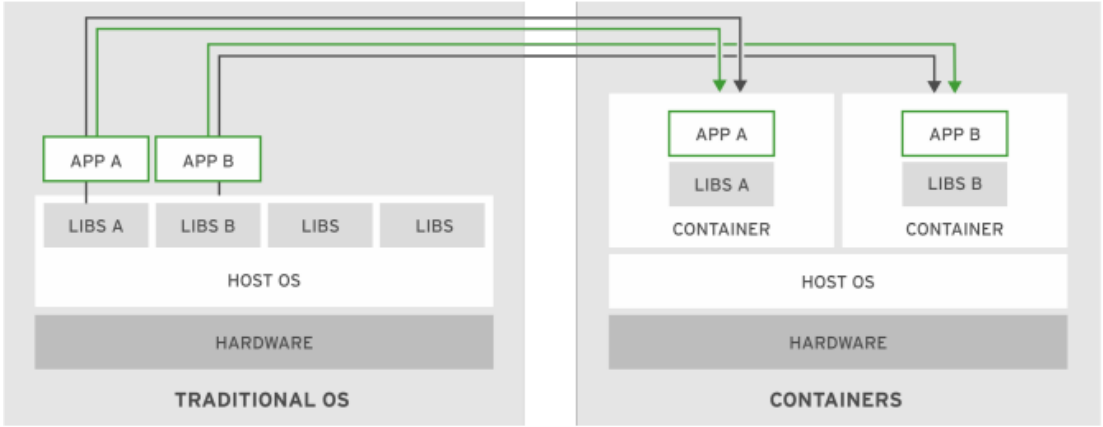
CONTAINER TECHNOLOGY

What are containers?

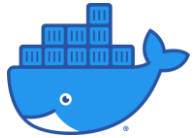
Containers are a set of one or more processes that are isolated from the rest of the system.

Enterprise Requirements

- Low Hardware Footprint.
- Quick and Reusable Deployment.
- Multiple Environment Deployment.



CONTAINER ARCHITECTURE

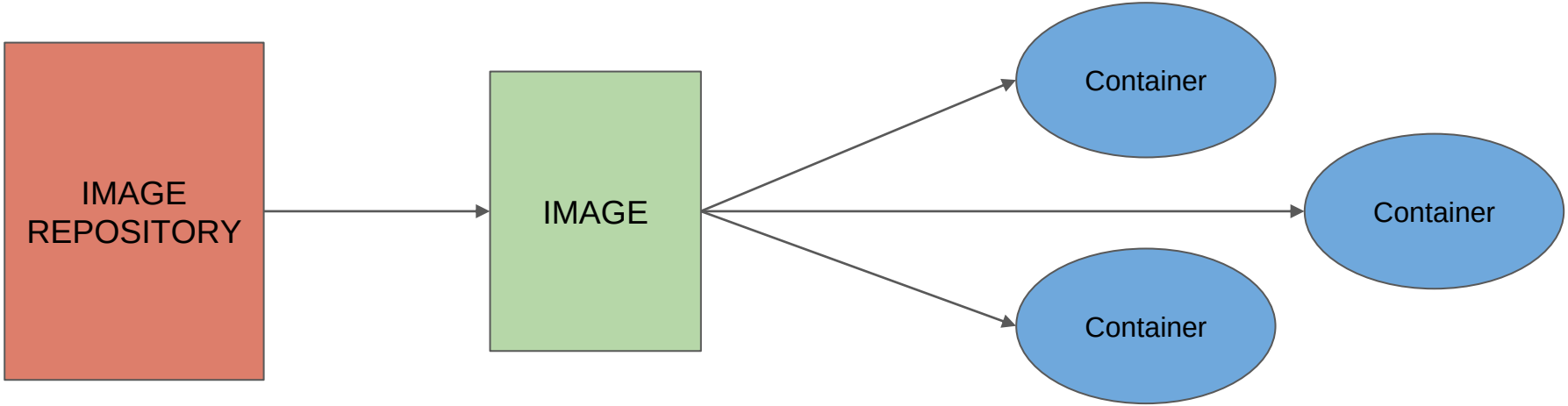


How do containers work?

- Namespaces
- Control groups (cgroups)
- Seccomp
- SELinux

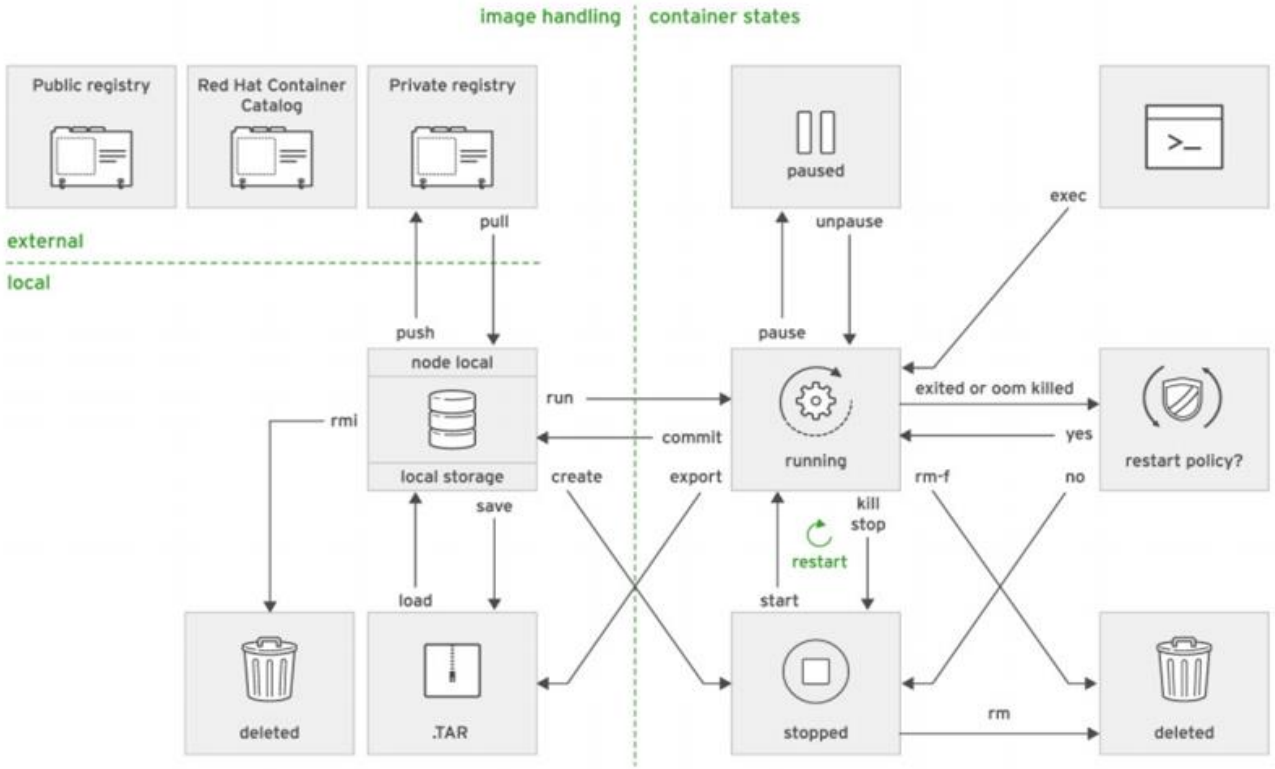
Linux Container Architecture Terms

- Containers.
- Image
- Image Repository
- Podman



CONTAINER LIFE CYCLE MANAGEMENT WITH PODMAN

Podman Managing Subcommands



Need of OPENSHIFT AND KUBERNETES

Limitation of Containers

As the number of containers managed by an organization grows, so is the manually starting too exponentially which is directly proportional to quick response to external demands.

Enterprise Needs:

- Easy communication between a large number of services.
- Resource limits on applications regardless of the number of containers running them.
- To respond to application usage spikes to increase or decrease running containers.
- To react to service deterioration with health checks.
- Gradual/Smooth roll out of a new release to a set of users.

OPENSHIFT AND KUBERNETES

ORCHESTRATION + SCHEDULING = HIGH AVAILABILITY

Three challenges of a container cluster architecture:

- Orchestration
- Scheduling
- Isolation

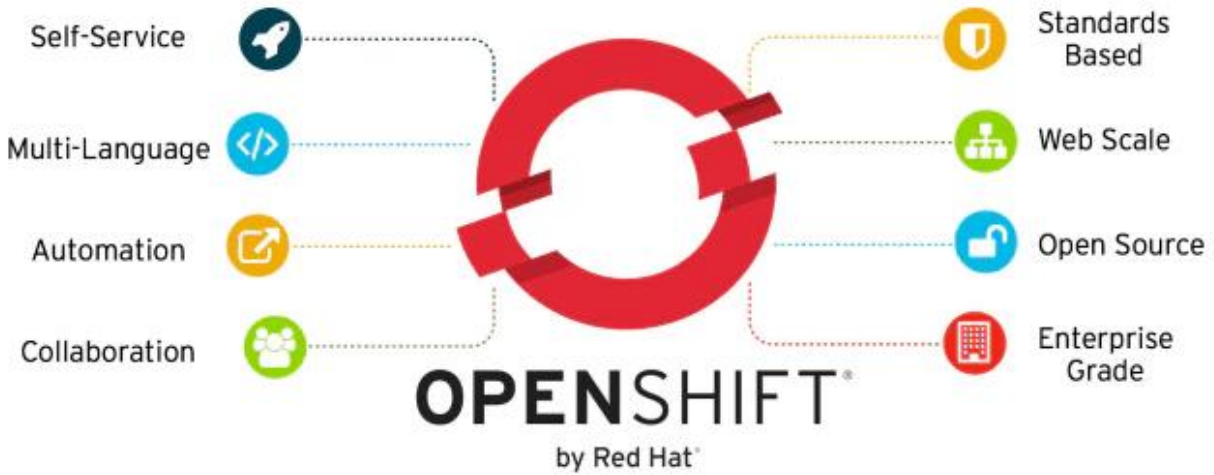
Kubernetes Features:

- Service discovery and load balancing
- Horizontal Scaling
- Self-healing with user defined health checks
- Automated rollout and rollback
- Secrets and Configuration Management.
- Operators

kubernetes

RH OCP Features

- Integrated developer workflow.
- Routes
- Metrics and logging
- Unified UI



Fetching Image From Image Registry

Running an application inside a container, requires a container image, a file system bundle providing all application files, libraries, and dependencies and the application needs to run.

Container images can be found in image registries. Some of the registries are as followed :

- Docker Hub.
- Red Hat Quay.
- Red Hat Container Catalog.

Container Image Name Syntax
Container images are named, based on the following syntax :

registry_name/user_name/image_name:tag

```
[student@workstation ~]$ sudo podman pull rhel
Trying to pull registry.access.redhat.com/rhel...Getting image source signatures
Copying blob sha256: ...output omitted...
 72.25 MB / 72.25 MB [=======================================================] 8s
Copying blob sha256: ...output omitted...
 1.20 KB / 1.20 KB [=========================================================] 0s
Copying config sha256: ...output omitted...
 6.30 KB / 6.30 KB [=========================================================] 0s
Writing manifest to image destination
Storing signatures
699d44bc6ea2b9fb23e7899bd4023d3c83894d3be64b12e65a3fe63e2c70f0ef
```

Running Containers

The **podman run** command runs a container locally based on an image

```
[student@workstation containers]$ sudo podman run ubi7/ubi:7.7 echo "Hello!"
Hello world
```

*Use the **-d** option to run the process in the background*

Some Common Container Options

- --name : Container Name
- -t : Pseudo Terminal
- -i : Interactive

```
[student@workstation ~]$ sudo podman run -it ubi7/ubi:7.7 /bin/bash
bash-4.2# ls
...output omitted...
bash-4.2# whoami
root
bash-4.2# exit
exit
[student@workstation ~]$
```

Docker and its Terminologies

To build base container images with Dockerfiles

1. Create a working directory.
2. Write the **Dockerfile** Specifications.
3. Build the image with the **podman** or **docker** command.

A Dockerfile is a simple text file and each line uses the following syntax:

INSTRUCTION *arguments*

The instructions are executed in the order they appear.

CMD and ENTRYPOINT

Defining an ENTRYPOINT in the Dockerfile creates containers that are Executables.
The ENTRYPOINT can be script that is added to the container with an ADD instruction.

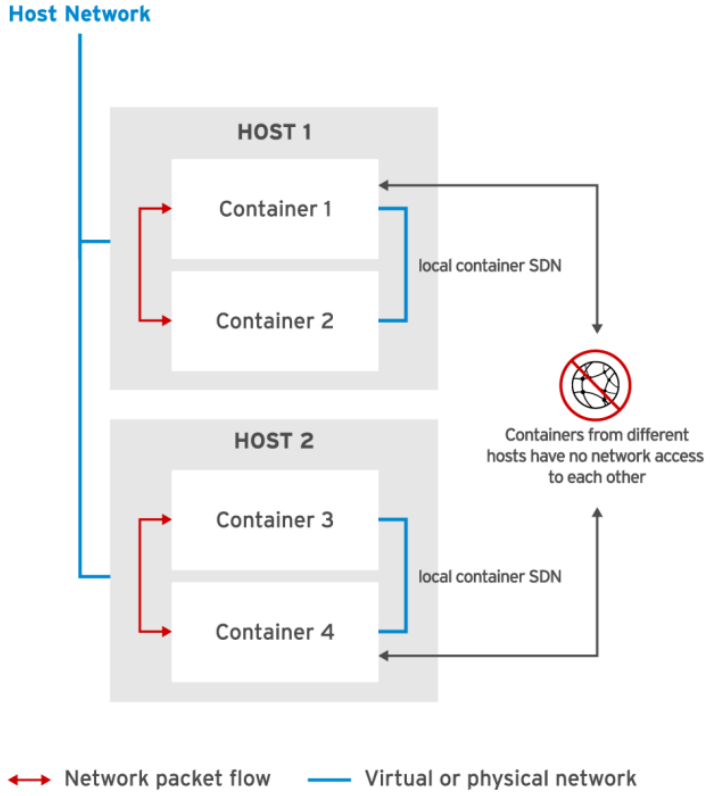
Sample Dockerfile

```
FROM ubi7/ubi:7.7

MAINTAINER Your Name <youremail>

LABEL description="A basic Apache container on RHEL 7 UBI"

RUN yum install -y httpd && \
    yum clean all

RUN echo "Hello from Dockerfile" > /usr/share/httpd/noindex/index.html

EXPOSE 80

ENTRYPOINT  ["httpd", "-D", "FOREGROUND"]
```

Basic Linux Container Networking

The Cloud Native Computing Foundation (CNCF) sponsors the *Container Networking Interface (CNI)* open source project. The CNI project aims to standardize the network interface for containers in cloud native environments, such as Kubernetes and Red Hat OpenShift Container Platform.

Podman uses the CNI project to implement a *software-defined network (SDN)* for containers on each host. Podman attaches each container to a virtual bridge and assigns each container a private IP address. The configuration file that specifies CNI settings for Podman is /etc/cni/net.d/87-podman-bridge.conflist.



KUBERNETES and OpenShift Resources

Kubernetes Architecture

- The smallest unit manageable in **Kubernetes** is a **pod**.
- A **pod** consists of one or more containers with its storage resources and IP.
- A **master node** provides basic cluster services such as **APIs** controllers.
- A **worker node** performs work in a Kubernetes cluster. **Application pods** are scheduled onto worker nodes.
- A **controller** is a Kubernetes process that watches the resources and make changes based on that state.
- **Services** : Define a single, persistent IP/port combination that provides access to a pool of pods.
- **Replication Controllers** : Defines how pods are replicated(horizontally scaled) into different nodes.
- **Persistent Volumes** and **Persistent Volume Claims.**
- **ConfigMaps** and **Secrets** : Contains keys and values that can be used by other resources.

KUBERNETES and OpenShift Resources

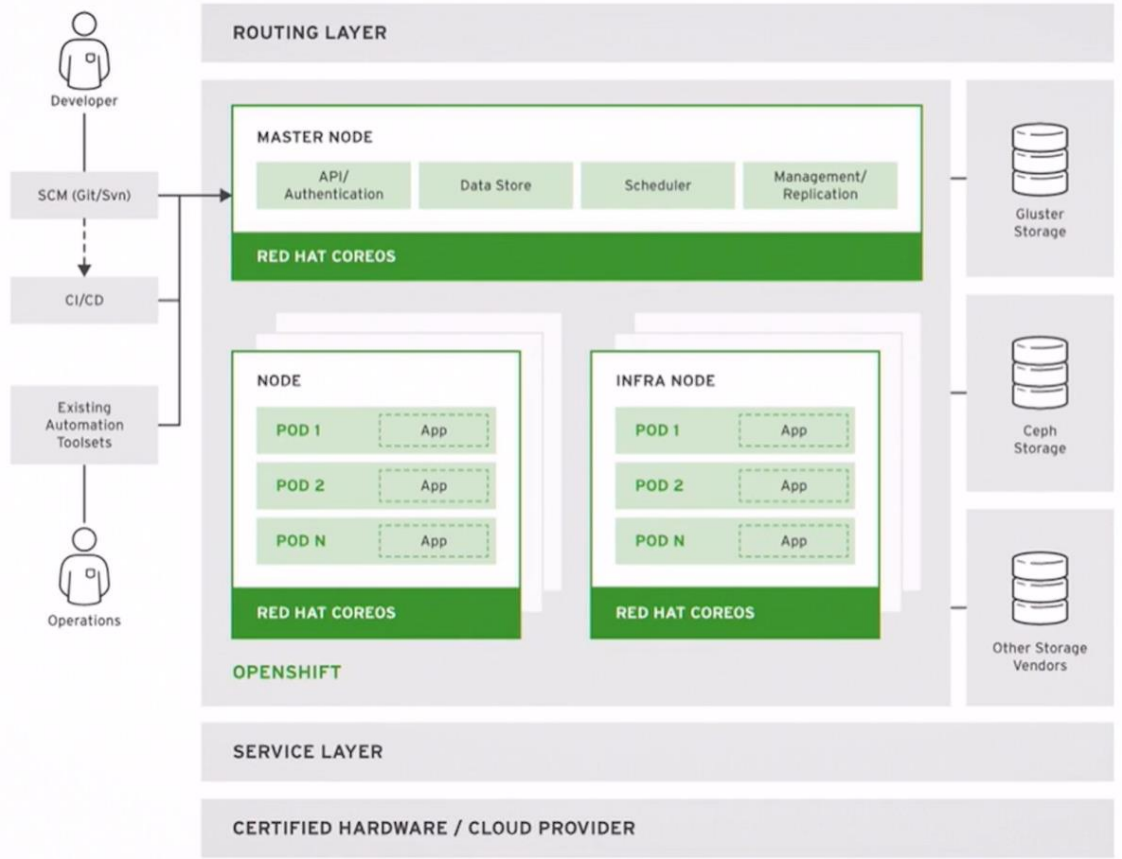
OpenShift Resource Types

- **Deployment Config (dc)** : Represents the set of containers included in a pod, and the deployment strategies to be used. A dc can also provide a basic but extensible continuous delivery workflow.

- **Build Config (bc)** : Defines a process to be executed in the OpenShift project. Source-to-image (S2I) feature to build a image from application source code stored in a Git repository.

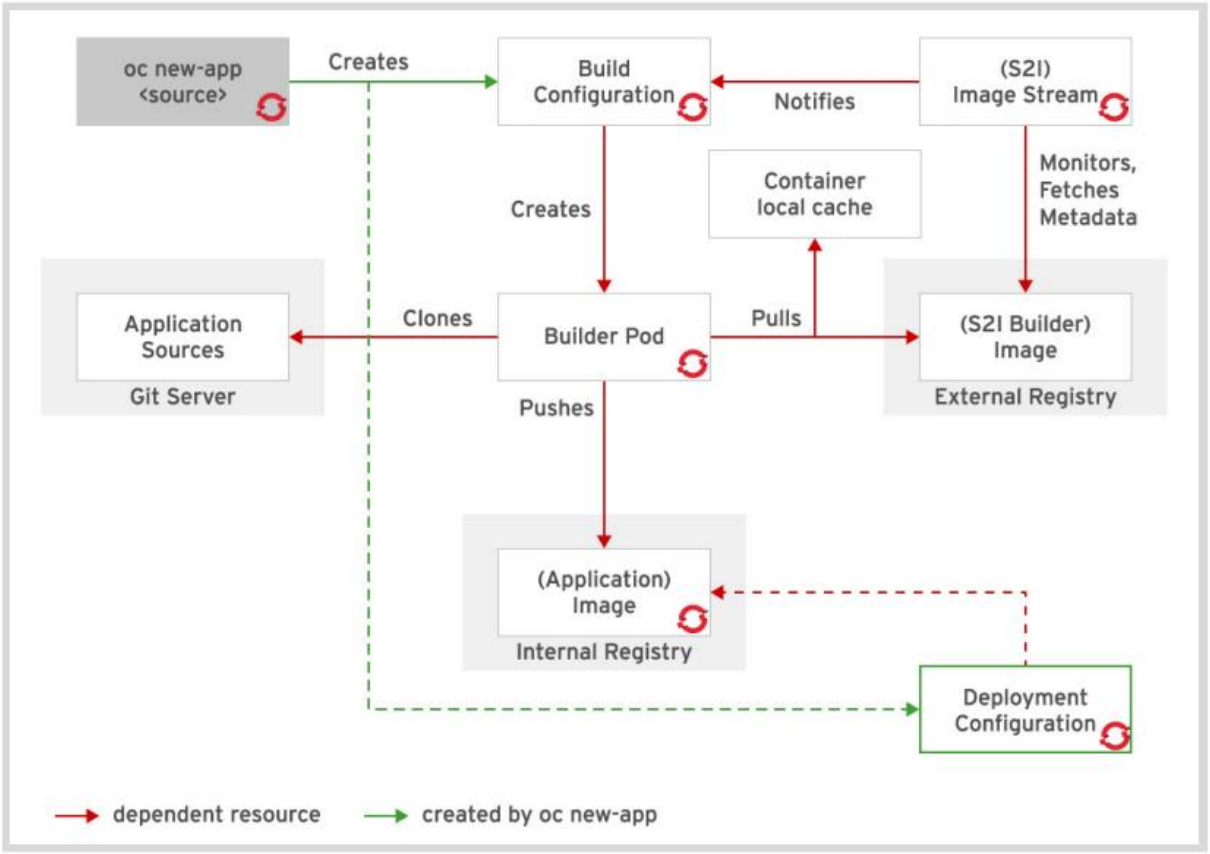
- **Routes** : Represent a DNS host name recognized by the OpenShift router as an ingress point for applications and microservices.

OpenShift Architecture



OpenShift Source-To-Image (S2I)

Deployment Configuration And Dependent Resources



OpenShift Web Console

Web Console Home Page