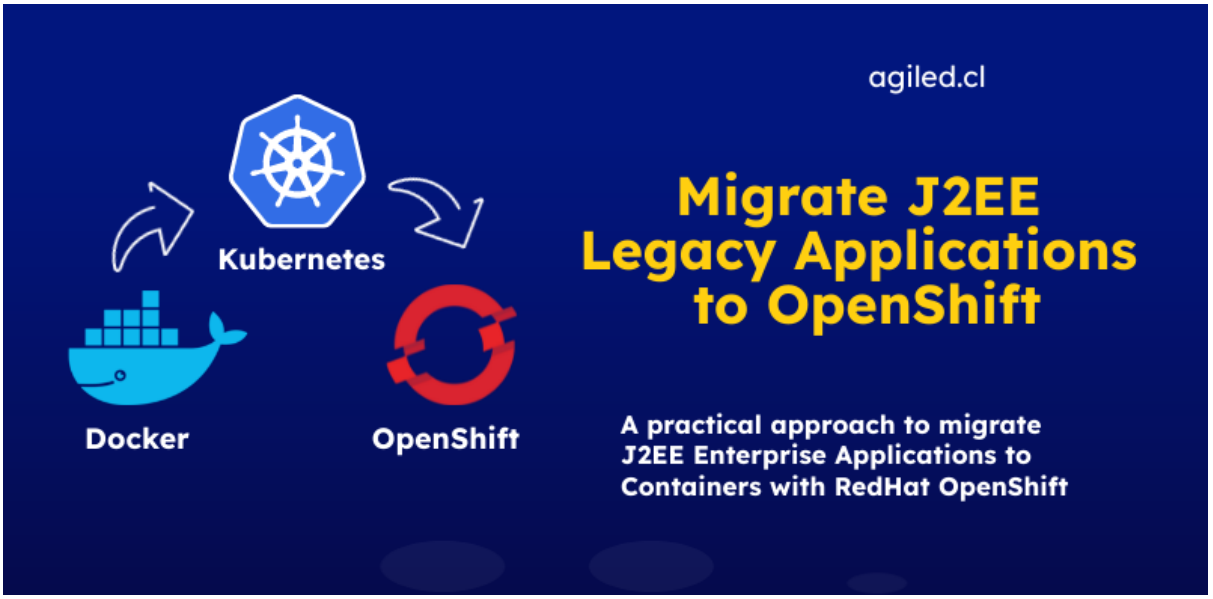


Rodrigo León Nanjarí | 22/12/2023

Migrate J2EE Applications to Containers with RedHat OpenShift

This article shows a practical approach to migrate J2EE enterprise and web applications running in IBM WebSphere to containers using RedHat OpenShift.

Many companies in the world keep using IBM WebSphere Application Server as a main platform for running enterprise Java based applications. This application server supports the full JEE standard including technologies like Servlets, JSP, EJB, JMS and much more. However, using IBM WebSphere in a production environment requires many resources in terms of host machines (virtual), processing capacity and memory.

Big enterprises use IBM WebSphere in cluster topology to support the deployment of one or more applications. This approach is highly demanding in infrastructure resources and does not consider the complexity of different applications. Not all applications consume the same amount of resources and not all applications need to operate in a high availability environment. Furthermore, this approach is not flexible enough to support big increases in demand in short periods of time. It commonly requires years of tuning and a vast amount of money to achieve a stable configuration to support all applications in a decent way.

Here is when the use of docker and containers comes very attractive, because enterprises can now deploy their applications in small containers, using only the resources required for every application. If an application consumes more resources, simply you can assign more resources to that container. Alternatively, if your application requires more computing power, you can start the same application in another container and manage the requests using a HTTP load balancer.

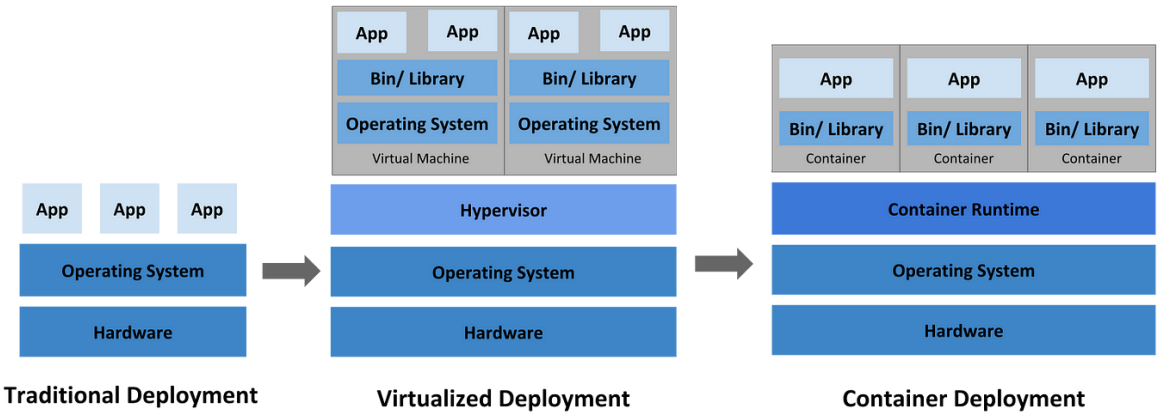
Containers provide benefits such as:

- Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and efficient rollbacks.
- Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Environmental consistency across development, testing, and production: runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Resource isolation: predictable application performance.
- Resource utilization: high efficiency and density.

Kubernetes and Containers

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The following figure describes the transition from traditional software development to containers.



Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Kubernetes provides you with:

- **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
- **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
- **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
- **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.
- **Batch execution** In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.

- **Horizontal scaling** Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.

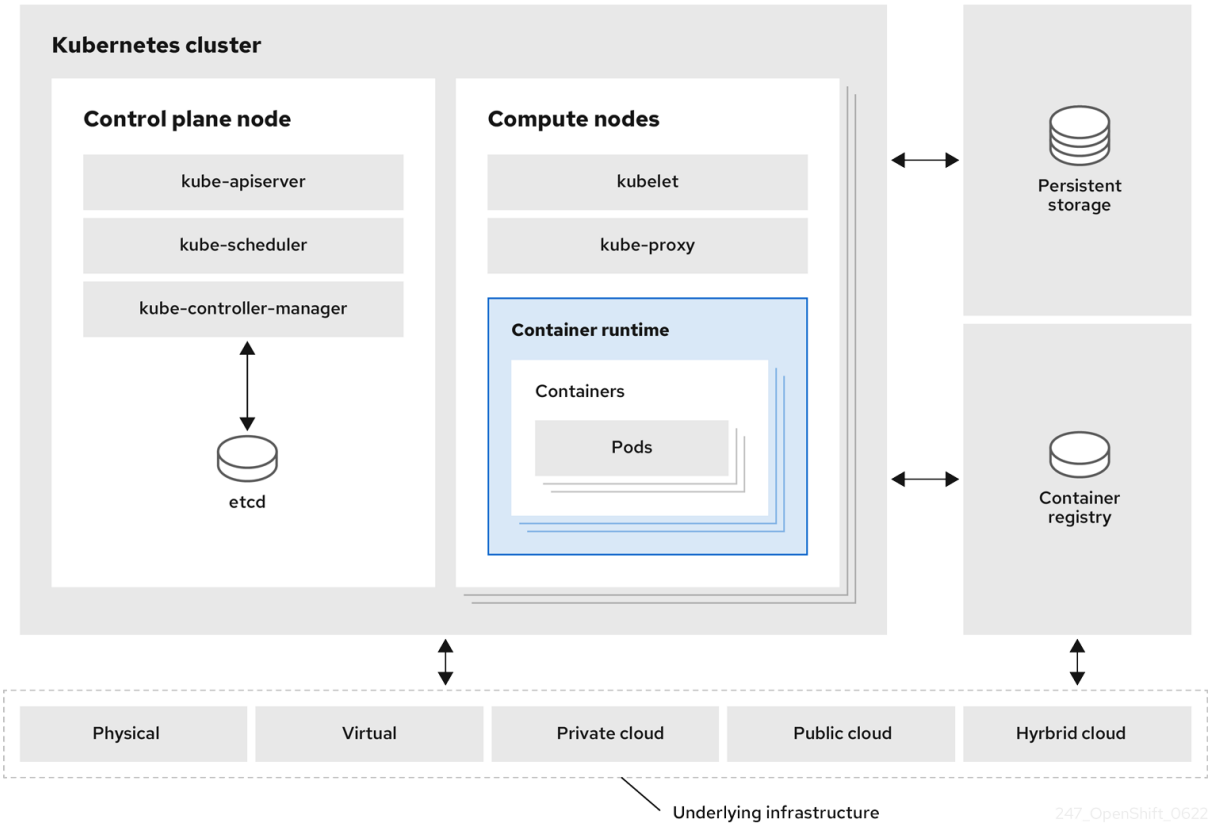
Introduction to RedHat OpenShift

RedHat OpenShift Container Platform is a Kubernetes environment for managing the lifecycle of container-based applications and their dependencies on various computing platforms, such as bare metal, virtualized, on-premise, and in cloud. OpenShift Container Platform deploys, configures and manages containers. OpenShift Container Platform offers usability, stability, and customization of its components.

OpenShift Container Platform utilizes a number of computing resources, known as nodes. A node has a lightweight, secure operating system based on Red Hat Enterprise Linux (RHEL), known as Red Hat Enterprise Linux CoreOS (RHCOS).

OpenShift Container Platform configures and manages the networking, load balancing and routing of the cluster. OpenShift Container Platform adds cluster services for monitoring the cluster health and performance, logging, and for managing upgrades.

You can manage applications within the cluster either manually by configuring deployments of containers running from pre-built images or through resources known as Operators. You can build custom images from pre-build images and source code, and store these custom images locally in an internal, private or public registry.

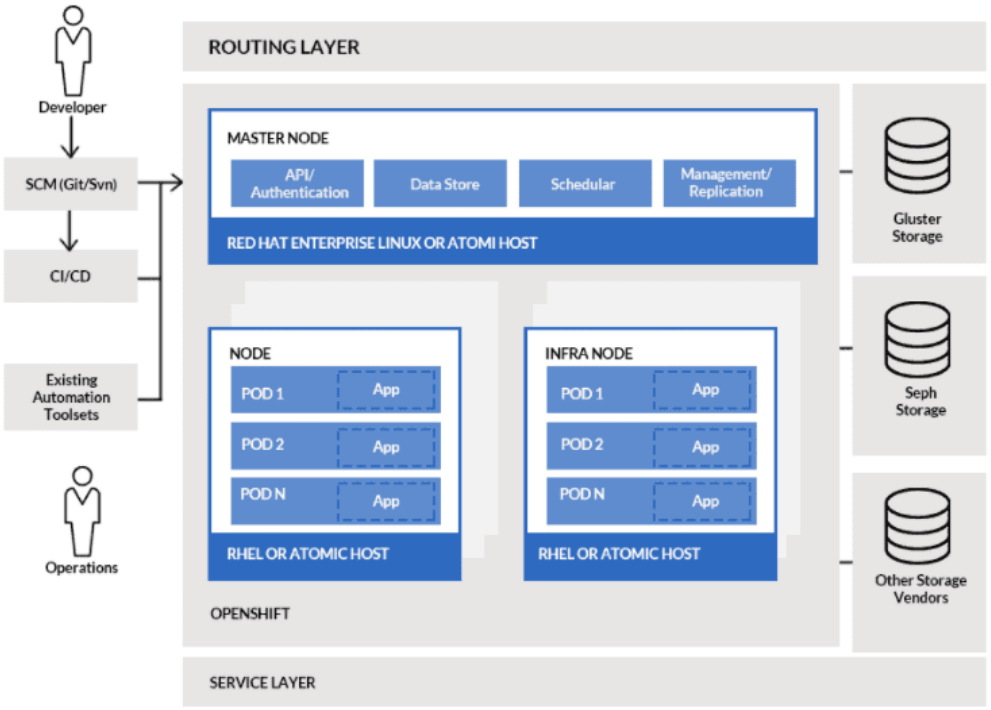


RedHat OpenShift Components

RedHat OpenShift Container Platform is designed with several components in order to implement Kubernetes over the RedHat environment. Below the main components:

- **Containers** are application instances and components that run in OCI-compliant containers on the worker nodes. An image is a binary application. A worker node can run many containers. A node capacity is related to memory and CPU capabilities of the underlying resources whether they are cloud, hardware, or virtualized.
- **Pod** is one or more containers deployed together on one host. It consists of a colocated group of containers with shared resources such as volumes and IP addresses. A pod is also the smallest compute unit defined, deployed, and managed.
- **Service** defines a logical set of pods and access policies. It provides permanent internal IP addresses and hostnames for other applications to use as pods are created and destroyed. Service layers connect application components together. For example, a front-end web service connects to a database instance by communicating with its service.
- **Route** is a way to expose a service by giving it an externally reachable hostname, such as www.example.com. Each route consists of a route name, a service selector, and optionally a security configuration.

- **Build** is the process of transforming input parameters into a resulting object. Most often, the process is used to transform input parameters or source code into a runnable image
- **Project** OpenShift Container Platform uses projects to allow groups of users or developers to work together, serving as the unit of isolation and collaboration

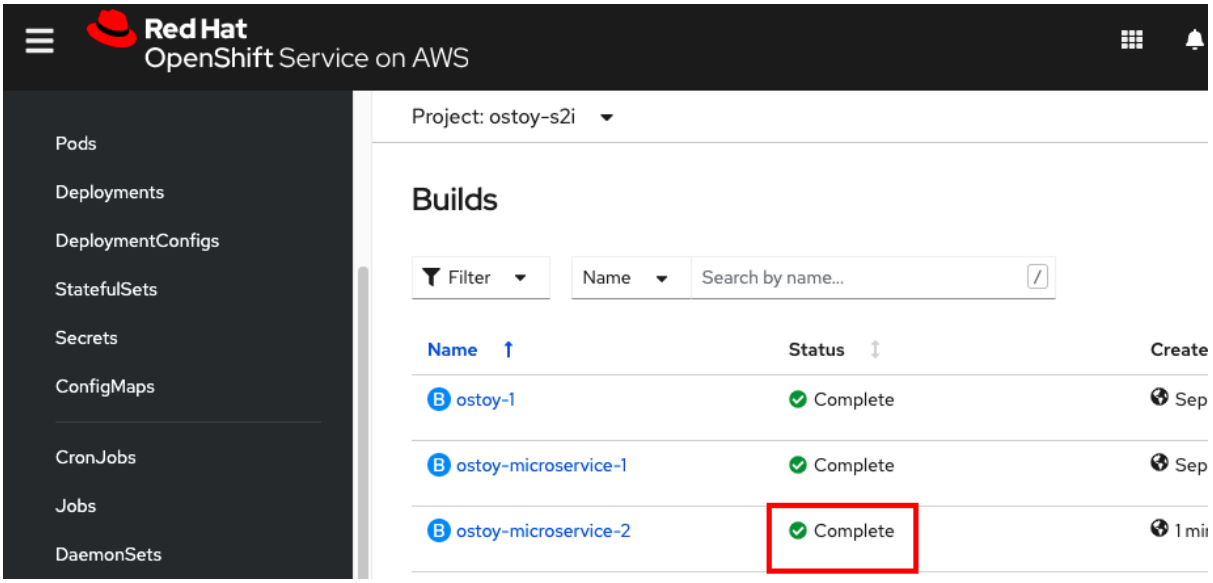


Keys Factors to Migrate IBM WebSphere Apps to OpenShift

In order to achieve a successful migration of an enterprise application running in IBM WebSphere Application Server environment to RedHat OpenShift Platform, we should take several considerations:

- Many JEE legacy applications use Enterprise Java Bean EJB 2.1 components. These components are supported only in JBoss 6.3 or minor versions. To deploy EJB 2.1 components to OpeShift, you must migrate these components to EJB 3.0, JPA or Spring.
- For legacy applications that contain web services, WebSphere includes the IBM JAX-WS runtime that generates skeleton classes that can be used in your Java project. These generated classes must be replaced with a standard supported by OpenShift Platform. We strongly recommend Axis to generate the implementation classes using the same WSDL contract service. In this way, there is no impact on other applications consuming these web services.

- When maven is used to compile and package your application, you must replace these dependencies with OpenShift and RedHat libraries supported (https://maven.repository.redhat.com/ga/).



Using GITLAB CI/CD Platform with OpenShift

If using GITLAB to implement the CI/CD Pipeline to deploy the migrated application into OpenShift, you must apply the following steps.

Add a DockerFile in the root directory of your application.
This file contains information about the Docker image. Example:

```
FROM registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-rhel8
COPY artifacts/*.ear /opt/eap/standalone/deployments/
COPY modules /opt/eap/modules
COPY                                          standalone-openshift.xml
/opt/eap/standalone/configuration/standalone-openshift.xml
EXPOSE 9990
```

Add a .gitlab-ci.yml in the root directory of your application.
This file contains the following information:

```
include:
  - project: 'my-group/my-project'
    ref: main   # Git branch
    file: '/templates/.gitlab-ci-template.yml'
```

Add a deployment.yml in the root directory of your application.

This file contains information about the deployment process in OpenShift, HTTP ports, web context and more.

Add a standalone-openshift.yml in the root directory of your application.
This file contains information about the configuration of the application in the instance of RedHat EAP Application Server, including JDBC drivers, DataSources and other resources.

Add a folder "modules/" in the root directory of your application.
This file contains information about modules used by your application, for example, JDBC drivers like Oracle or SQL Server.

Add a ConfigMap.yaml in the OpenShift console
This file contains information about metadata used by the instance, for example, database.

Add a Secrets.yaml in the OpenShift console
This file contains information about credentials used by the instance, for example, database credentials.

Create a Custom DNS for your Application
Is best practice to create a custom DNS to access your application deployed in OpenShift, in order to simplify the access for users and others systems.

Summary

This article shows a practical approach to migrate legacy applications running into IBM WebSphere Application Server to Containers with Redhat OpenShift Platform. We cover the principal concepts of containers, his benefits, Kubernetes and the RedHat OpenShift Platform.

We hope that this article will invite you to migrate your legacy applications running, not only in IBM WebSphere environments, but also Oracle Weblogic, GlassFish and others application servers with monolithic applications.

Bibliography

- Kubernetes Overview
  https://kubernetes.io/docs/concepts/overview/
- RedHat OpenShift Platform Getting Started
  https://access.redhat.com/documentation/en-us/openshift_container_platform/
- GITLAB Docs

https://docs.gitlab.com/

Rodrigo León Nanjarí
CEO Agiled and Software Architect

Since 2018, Rodrigo has been working in the migration of hundreds of mission critical enterprise applications and services to RedHat EAP or containers with OpenShift.

rodrigo.leon@agiled.cl
http://www.agiled.cl
https://www.linkedin.com/in/rodrigoleonnanjari/