NewgenONE OmniDocs RMS

Newgen Enterprise Products Containerization Guide for OpenShift

Version: 4.0 SP1

Newgen Software Technologies Ltd.

Table of contents

1 Preface

This chapter provides information about the purpose of this guide, details on the intended audience, revision history, and related documents for OmniDocs RMS Containerization.

1.1 Revision history

Revision date	Description
April 2024	Initial publication

1.2 About this guide

This guide describes the containerization approach for Newgen Enterprise products, which provides the details of the container-based deployment architecture and container-based DevOps pipeline for OmniDocs RMS on OpenShift Container Platform.

NOTE:

To ensure you are referring to the latest and most recent revision of this guide, download it from one of the following locations:

- Newgen Internal Doc Portal, if you are a Newgen employee.
- Newgen Partner Portal, if you are Newgen partner.

1.3 Intended audience

This guide is intended for System Administrators, developers, and all other users who are seeking information on the OmniDocs Container Architecture and DevOps Pipeline to manage deployments. The reader must be comfortable understanding the computer terminology. The reader must have administrative rights for deployment and configuration.

1.4 Related documents

The following documents are related to the OmniDocs Docker Document:

- OmniDocs RMS 4.0 SP1 Configuration and Deployment Guide
- OmniDocs RMS 4.0 SP1 Docker Containers Hotfix Deployment Guide
- OmniDocs RMS 4.0 SP1 Newgen Enterprise Product Containerization Guide
- OmniDocs RMS 4.0 SP1 Docker Containers Custom Code Deployment Guide
- OmniDocs RMS 4.0 SP1 Release Notes

1.5 Documentation feedback

To provide feedback or any improvement suggestions on technical documentation, write an email to docs.feedback@newgensoft.com.

To help capture your feedback effectively share the following information in your email.

- Document name
- Version
- Chapter, topic, or section
- Feedback or suggestions

1.6 Third-party information

This guide contains third-party product information about configuring OpenShift Container Platform and Jenkins CICD Pipeline for Container Deployment on OpenShift. Newgen Software Technologies Ltd does not claim any ownership of such third-party content. This information is shared in this guide only for the convenience of our users and could be an excerpt from the OpenShift documentation. For the latest information on configuring the OpenShift Container Platform and Jenkins CICD Pipeline refer to the respective official documentation.

2 Containerization of Newgen Enterprise Products

The containerization support is enabled for Newgen's flagship product OmniDocs RMS on OpenShift Container Platform. Newgen has isolated the product suite’s application modules that must be deployed in a separate docker container to enable independent scalability for each of these components or modules. Newgen product suite’s web components and EJB components are isolated for deployment in separate Docker Container Instances. Web components are deployed on the underlying web server JBoss WebServer 5.7.x. EJB components are deployed on the underlying application server JBoss EAP 7.4.x.

2.1 Advantages of docker containerization

The key advantages are as follows:

- **Reduce time and effort in environment configuration** — With Docker containers, you can reduce the time and effort in environment configuration, you don't need to bother about the installation parts. Whenever you want to use these products, pull the docker images and run them along with some configuration. Also, Containers are immutable by design so no need to worry about corrupted VMs.
- **Consistency across the environments** (Dev, UAT, Production, and so on) — Docker container provides environment consistency which means it never faces such issues that this application is working on a Dev environment, but it is not working on UAT and stage environment due to O.S updates or patches.
- **Auto Scaling of Containers** — Scaling containers is far faster and easier than deploying additional VMs. Docker containers are auto-scaled up and scaled down depending on CPU utilization and memory consumption.
- **Easy distribution and portability** — As our application and its dependencies all are bundled inside the Docker image then it becomes easy to distribute our applications using the container registry.
- **Maintain the Liveness of applications running inside a container** — Docker containers have healing power, if an application running inside the container gets down due to any reason or becomes unresponsive then Kubernetes auto-restarts that application inside the container.
- **Seamless deployment of updates (Rolling Deployment)** — Deploying updates as a Docker image is far faster and very efficient. Whenever you push new images to the container registry, Kubernetes 1st of all creates new containers to make it up and running once new containers become fully functional after that Kubernetes start deleting the existing containers. In such a case, the end-user never faces any downtime even if the user session also not gets expired, and the latest updates get deployed.

2.2 Deliverables

The deliverables contain the Product docker images for initial deployment. Newgen has isolated the product suite into multiple Docker containers to enable the independent scalability of each Docker container. This separation is done based on the product's usability. At a broad level, Web components and EJB components are isolated for deployment in separate container Instances. Web components are deployed on the underlying web server JBoss WebServer 5.7.x. EJB components are deployed on the underlying application server JBoss EAP 7.4.x. Newgen releases multiple Docker images for the different product suites along with some configuration files for data persistence, YAML files for deployment, and some documentation for end-to-end configurations and deployments.

2.2.1 Docker images of OmniDocs RMS

There's a segregation of product suite into multiple Docker containers to enable the independent scalability of each Docker container. This segregation is done based on the product's usability. At the broad level, the OmniDocs Services container consists of five different add-on services of OmniDocs RMS. If there is a requirement, you can split them into separate containers.

EasySearch container consists of Apache Manifold which is freeware software.

If OmniScan Web components are required for scanning services, then it also constitute a separate Docker instance.

OmniDocs SMS (Storage Management System) service is required to store the PN files. PN files are encrypted files that contain all the added/uploaded/scanned documents by Newgen products.

Docker Image Name	Docker Image Size
OmniDocs Web	893 MB
OmniDocs Web Service	541 MB
OmniDocs EJB	701 MB
OmniDocs Services	1.15 GB
EasySearch(Apache Manifold only)	741 MB
TEM or FTS	408 MB
OmniScan Web	498 MB
OmniDocs SMS	354 MB
SharePointAdapter	176 MB
OmniDocsWopi	315 MB

2.3 Deployment architecture

This section describes the deployment architecture of OmniDocs RMS Docker Containers.

2.3.1 OmniDocs RMS Vanilla deployment architecture

This section describes the deployment of the vanilla flavor of OmniDocs RMS.

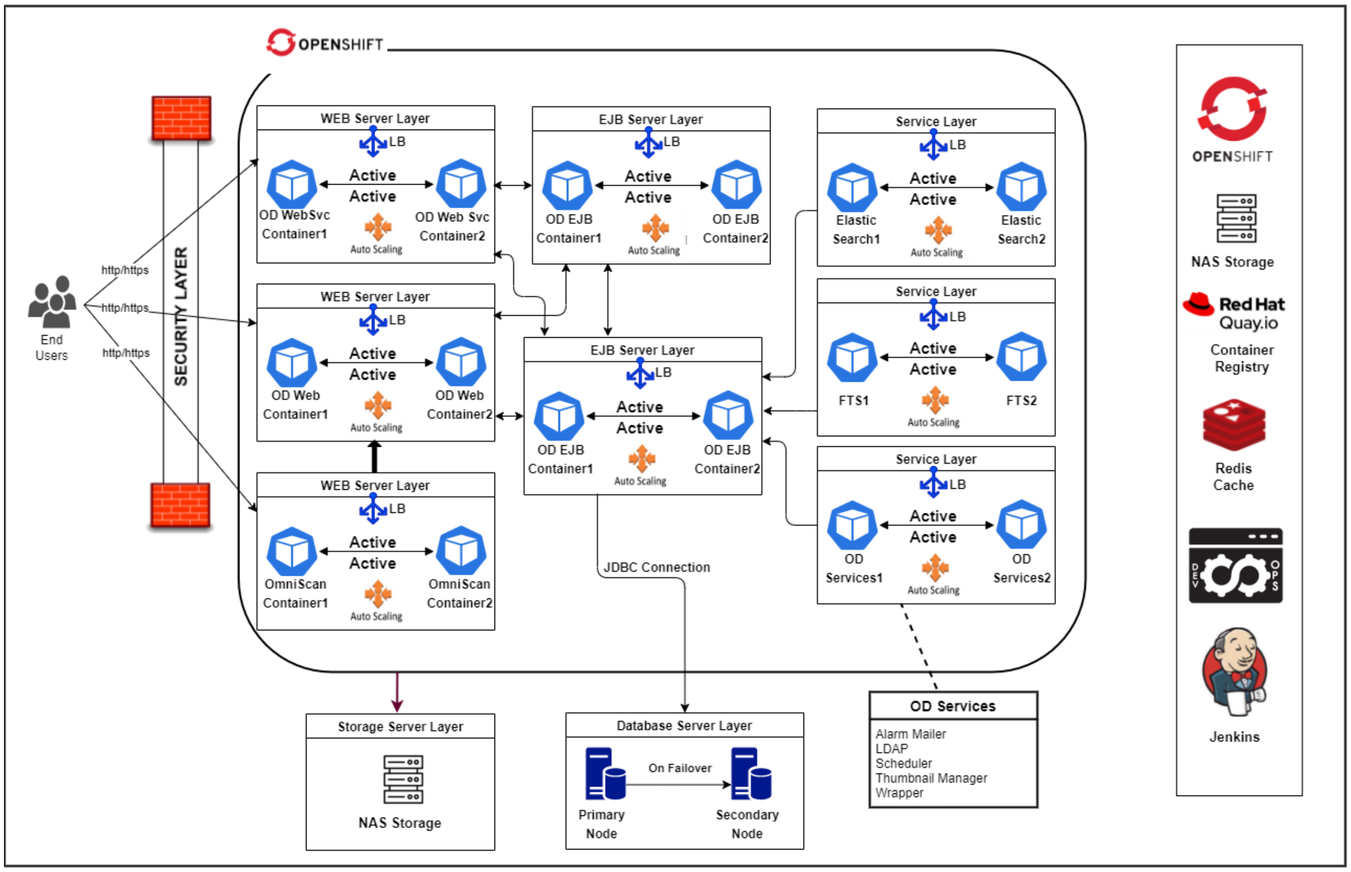

Figure 2.1

- In this architecture, all user requests, either coming from end-users, mobile users, admin users or portal users comes to Load Balancer, which is internally managed by the OpenShift Ingress controller. An ingress controller is an object running inside the Openshift cluster that is used to manage the host-based routing rules.
  For example, you can set the host-based routing rules like if the URL is *omnidocs.apps.newgenopenshiftcluster.newgensoftware.net* then the ingress controller redirects the user request to OmniDocs RMS WEB containers. If the URL is *omnidocswebservice.apps.newgenopenshiftcluster.newgensoftware.net* then the ingress controller redirects the user request to the OmniDocs RMS WEB container.

- Now, there is a separate container for web services that can directly connect with the EJB's present in another container. In case, users want to connect their business or productivity applications with OmniDocs, they can call out these web services directly (which are now deployed in separate container). Web server components and App server components are still deployed in separate containers for accessing OmniDocs through the GUI.
- Here, each Docker image is running in HA mode, one container in each availability zone, and both the containers is running in Active-Active mode so that they can serve the user requests at the same time and manage the load properly. All the containers are also running with auto-scaling enabled so that whenever user requests increase drastically, new pods can auto-scaled up and start serving the user requests.
- Here, a separate Docker container is created for OmniDocs Addon services like EasySearch, FTS, Alarm Mailer, and all these services is connected to the OD RMS EJB container. Additionally, OD RMS EJB is connected to the database servers.
- In this architecture, the database is running in Active-Passive mode and sync the data in real-time. Thus, primary instance is going down then secondary instance can take place and start serving the requests meanwhile you can fix the issues on primary nodes.
- OpenShift Container Platform (OCP) works for container orchestration.
- Here, the NAS Server or NFS Server is used for the Storage layer. It is used to store the PN files. A PN File is nothing but an encrypted file that contains all the added, uploaded, or scanned documents by Newgen products.
- The NAS Server is also used to persist the product's configuration and log files outside the container because these are dynamic and modified configuration files or newly created log files that cannot be recovered once a container is terminated due to any reason.
- To store the Docker images in a private repository, RedHat Quay Registry is used. However, you can use other Image repositories as well like Azure Container Repository or DockerHub.
- For the product's web session persistence, Redis Cache is used.
- For end-to-end CICD Pipeline, Jenkins is used.

2.4 Tools and technologies

The following tools and technologies are used for this containerization approach:

- **Container Orchestration — OpenShift Container Platform**
  OpenShift Container Platform is a popular enterprise-grade container management platform developed by Red Hat. It is built on top of open-source technologies, primarily Kubernetes, and provides a comprehensive set of tools and features for deploying, managing, and scaling containerized applications and services. OpenShift uses Kubernetes as its core container orchestration engine. This means it provides features for deploying, scaling, load balancing, and

managing containerized applications, making it easier to manage complex microservices architectures.

- **WorkerNode Host OS — RHEL CoreOS (RHCOS) or RHEL 8.4/8.5**
  Here, RedHat CoreOS or RHEL 8.4 or RHEL 8.5 can be used as the OpenShift worked node. Worker node is used to run the containerized applications. Every Kubernetes cluster has at least one worker node.
- **Container OS — Alpine**
  Container OS is a lightweight, optimized operating system that is used to run Docker containers and bundled inside the Docker images. It defines the file system structure of a container.
  For example, Ubuntu, CentOS, Alpine, Windows, RHEL, and so on. In Newgen, the Alpine is used whose size is about 5 MB only. If there is some special requirement, the Docker images are built with another container OS on-demand basis.
- **Docker Image Registry — RedHat Quay Registry**
  RedHat Quay Registry is used to store the Product's Container Images.

- **AppServer — JBossEAP 7.4.x**
  JBoss EAP 7.4.x deploys the Newgen Product's EJB components.
- **WebServer — JBossWebServer 5.7.x**
  JWS 5.7.x is to deploy the Newgen Product's WEB components.
- **JDK — OpenJDK 1.17.x**
  Inside the Docker images, OpenJDK 1.17.x is bundled. If there is some special requirement, then you can build the Docker images with Oracle JAVA on-demand basis.
- **Database — Oracle 19C**
  Newgen Products support three database servers: Microsoft SQL Server, Oracle, and PostgreSQL. The supported database versions for products are available in the Product Support Matrix.
- **PN Files Storage Server — NAS Storage or NFS Storage**
  PN files are encrypted files that contain Newgen products added, uploaded, and scanned documents. Here, NAS Server or NFS Server is used for the storage layer.
- **Log Files Persistence — NAS Storage or NFS Storage**
  All the files created inside a container are stored on a writable container layer. This means that the data doesn't persist when that container no longer exists or is terminated due to any reason, and it can be difficult to get the data out of the container if another process requires the same. Thus, to persist these types of data NAS Server or NFS Server is used.
- **Configuration Files Persistence — NAS Storage or NFS Storage**
  Same as log files, NAS Server or NFS Server is used to persist the configuration files.
- **Load Balancer —Ingress Controller**

Here, OpenShift Ingress controller is used to route the incoming user requests to the backend containers. An ingress controller is an object running inside the Kubernetes cluster that is used to manage the host-based routing rules.

- **Session Persistence** —**Redis Cache**
  Redis Cache is an easy-to-use, high-performance, in-memory data store. It offers a mature, scalable, open-source solution for delivering sub-millisecond response times making it useful as a cache or session store.

2.5 Kubernetes autoscaling

In Kubernetes, auto-scaling happens at two-level depending on multiple parameters like CPU utilization, memory consumption, and so on.

1. **Pod level** — This is controlled by Horizontal Pod Autoscaling (HPA) or Vertical Pod Autoscaling (VPA).
   i. **Horizontal Pod Autoscaler (HPA)** automatically scales the number of pods depending on CPU utilization and memory consumption. Like if CPU utilization goes more than 80% then a new pod automatically gets launched and is mapped to the load balancer. Below are some points regarding HPA’s default behavior:
      - The default HPA check interval is 15 seconds.
      - HPA waits for 3 minutes after the last scale-up events to allow metrics to stabilize.
      - HPA waits for 5 minutes from the last scale-down event to avoid auto scaler thrashing.
      - In HPA, the autoscale works on the resource's request parameter instead of the resource's limit parameter.
      - HPA can be configured on CPU utilization, memory utilization, or both with an OR condition.

   ii. **Vertical Pod Autoscaler (VPA)** — just like HPA, a Vertical pod autoscaler does not add more replicas but increases the number of system resources available to the particular pod. It automatically adjusts the CPU and memory reservations for the pod. VPA gives some recommendations based on the data collected from the matrix server and updates the existing pod with recommended values.
2. **Cluster** or **Node level** — It is also known as Cluster Autoscaler. It automatically adjusts the number of nodes in your cluster when pods fail to launch or are pending due to lack of resources or when nodes in the cluster are underutilized and their pods can be rescheduled onto other nodes in the cluster. In other words, Cluster Autoscaler adds nodes to a cluster whenever it detects pending pods that cannot be scheduled due to resource shortages and delete nodes from a cluster whenever the utilization of a node falls below a certain threshold defined by the cluster administrator. Thus, it automatically scales up or down depending on resource utilization.

NOTE:
However, Cluster Autoscaler is primarily associated with cloud-based Kubernetes deployments. It can also be used with OpenShift running on bare metal hardware or VMWare. But there are some important considerations and steps to set up Cluster Autoscaler in such an environment. Please contact RedHat team for more information.

2.6 Logging architecture

This section explains the logging architecture.

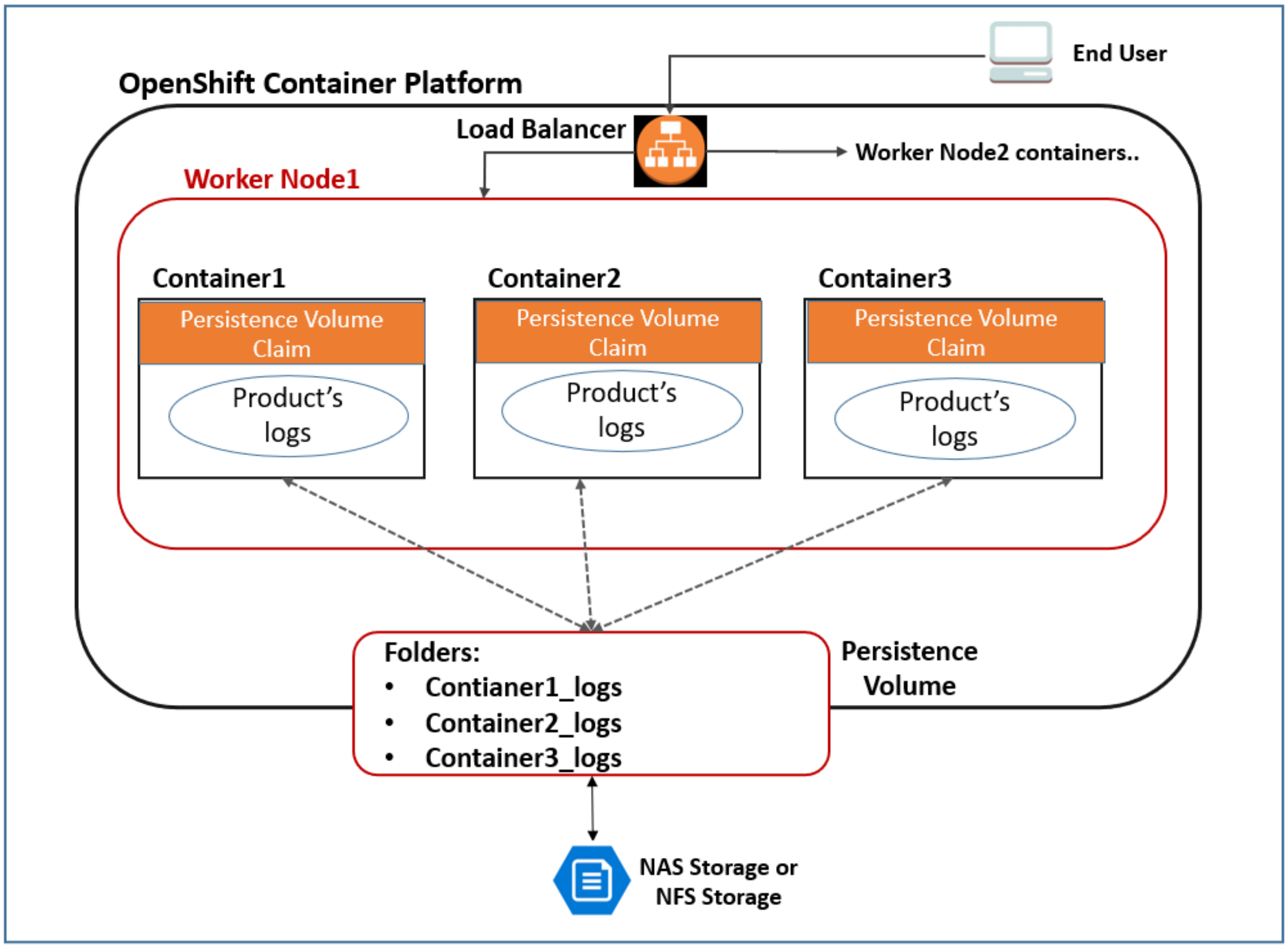

Figure 2.2

- All the files are created inside a container are stored on a writable container layer. This means that the data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if you need it later. Thus, to persist these types of data, use NAS storage or NFS storage. It helps to share data across multiple nodes and pods.
- Since the product's log files are dynamic in nature and are created at runtime so they must be kept outside the container. Therefore, NAS storage is used to persist in the Newgen Product's log files.
- To persist the data in NAS storage, there's a concept of PV or PVC (Persistence Volume or Persistence Volume Claim):

    - Persistence Volume: Persistent Volume (PV) is a piece of storage in the cluster that has provisioned using Storage Classes. It contains the NAS server IP address and share path that is already created on the NAS server. It is also used to set the access permission on NAS share path using accessModes attribute.
    - Persistence Volume Claim: A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (for example, they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany).
- So, all the product's logs created inside the containers can be claimed by the PVC in some predefined folder structure.

  For example, container1's product logs are stored inside the container1_logs folder on a worker node, and container2's product logs are stored inside the Container2_logs folder on a worker node and so on.
- Further, all the log folders created, is mounted to NAS server so that whenever a Pod goes down, our product's logs must always be available in NAS server. Whenever it is required, you can fetch it from there.
- Mounted product's logs are in sync in real-time. So, whenever the product's logs are generated inside the container, these logs are available in NAS server as a persistence volume at the same time. There is no delay.

2.7 Container security

The Container Images are as follows:

- **Using Secure Images** — Since you are bundling numerous third-party libraries as dependencies, these dependencies contain some vulnerabilities. Also, Docker images are created using nested Docker images at multiple layers and any parent Docker image may contain vulnerabilities. So, it becomes necessary to scan all the Docker images against vulnerabilities. As all the Docker images are stored inside the RedHat Quay Container Registry.

  RedHat Quay image scanning gets enabled by default. Whenever you push an image to the container registry, Security Centre automatically scans it and then checks for known vulnerabilities in packages or dependencies defined in the file. Quay uses the Common Vulnerabilities and Exposures (CVEs) database from the open-source **Clair** project and provides a list of scan findings.
- **No container must be running with root privileges** — The security context **RunAsNonRoot** is configured for all the containers in the deployment manifest file. This ensures that the application must not be running with administrator or root privileges inside the container.

Container Runtime Environments are as follows:

- **Restricted communication between containers** — In a Kubernetes cluster, containers can communicate with any other container running in the same cluster. However, containers need to communicate with each other to accomplish their goals, but they must not communicate with all the running containers. The reason is that if a hacker gets access to one container, then that hacker can communicate with any other container as well. So, allow containers to communicate to only those containers that are required to minimize the attack surface. To handle this case, the **Network Security Policy** is implemented between the containers by defining some ingress and egress rules.
- **ServiceMesh** — ServiceMesh is a dedicated infrastructure layer for managing secure communication between containers. It consists of a set of network proxies deployed alongside application code, which intercept and manage traffic between containers.

Orchestrators:

- By using Kubernetes for container orchestration, it automates many of the tedious tasks required to deploy containerized apps, but it does not manage container security. However, it can enforce role-based access control policies and network policies.
  For example, it can restrict the ability of a container to access resources on the host server.

Container Registry:

- Container registry providers already provide security on the container registry. You cannot push or pull any Docker images without authorization.
- A private container repository is available on the cloud, but it is accessible through whitelisted users only.
- Vulnerability scanning is enabled to scan the pushed Docker images.

2.8 CI/CD pipeline

The Build Pipeline (aka Continuous Integration) and Release Pipeline (aka Continuous Deployment) are separated into two parts. Build Pipeline and Release Pipeline both are done through the Jenkins server which can be installed on an on-premises machine or a bastion server. In this architecture, three stages are created, Dev, UAT, and Production, and in each stage, deployment is quite different. You can have some more stages depending on the requirements. This document describes the configuration of the Jenkins Release Pipeline for container deployment on the OpenShift Container Platform

2.8.1 CI/CD pipeline of base products

This section describes the CI/CD pipeline for base products.

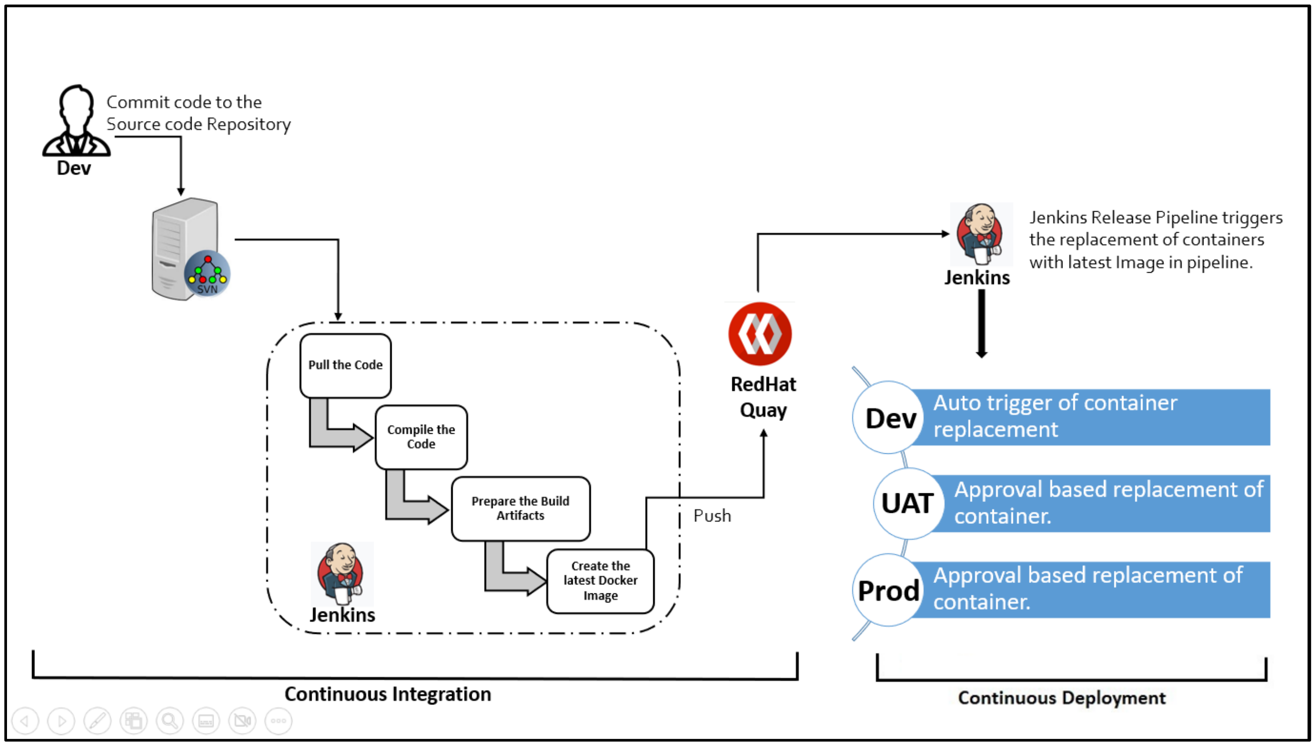
Commit code to the Source code Repository
Dev
SVN
Pull the Code
Compile the Code
Prepare the Build Artifacts
Create the latest Docker Image
Jenkins
Push
RedHat Quay
Jenkins
Jenkins Release Pipeline triggers the replacement of containers with latest Image in pipeline.
Dev
Auto trigger of container replacement
UAT
Approval based replacement of container.
Prod
Approval based replacement of container.
Continuous Integration
Continuous Deployment

Figure 2.3

- The Newgen representative builds the product's base container images on the company's on-premises servers using Jenkins.
- As soon as the Dev team commits the code to the source code repository, the Jenkins pipeline gets triggered. It pulls the code then compiles them and prepares the build artifacts as well as creates container images and pushes the newly created container images to the RedHat Quay Registry.
- As soon as any container image is pushed to the RedHat Quay Registry, Jenkins Release Pipeline triggers the deployment to the Dev environment. Here, you can configure the performance testing as well as security testing of the application. In Addition, you can perform manual testing as required.
- UAT and Production deployments are based on approval and are available on-demand. To deploy to the UAT/Production environment, you need to trigger the UAT/Production deployment. Upon deployment trigger, an approval mail is sent to the project manager or the concerned team. As soon as the project manager or concerned team approves the go-ahead, UAT/Production deployment gets started.

2.8.2 CI/CD pipeline of custom code

This section describes the CI/CD pipeline of custom code.

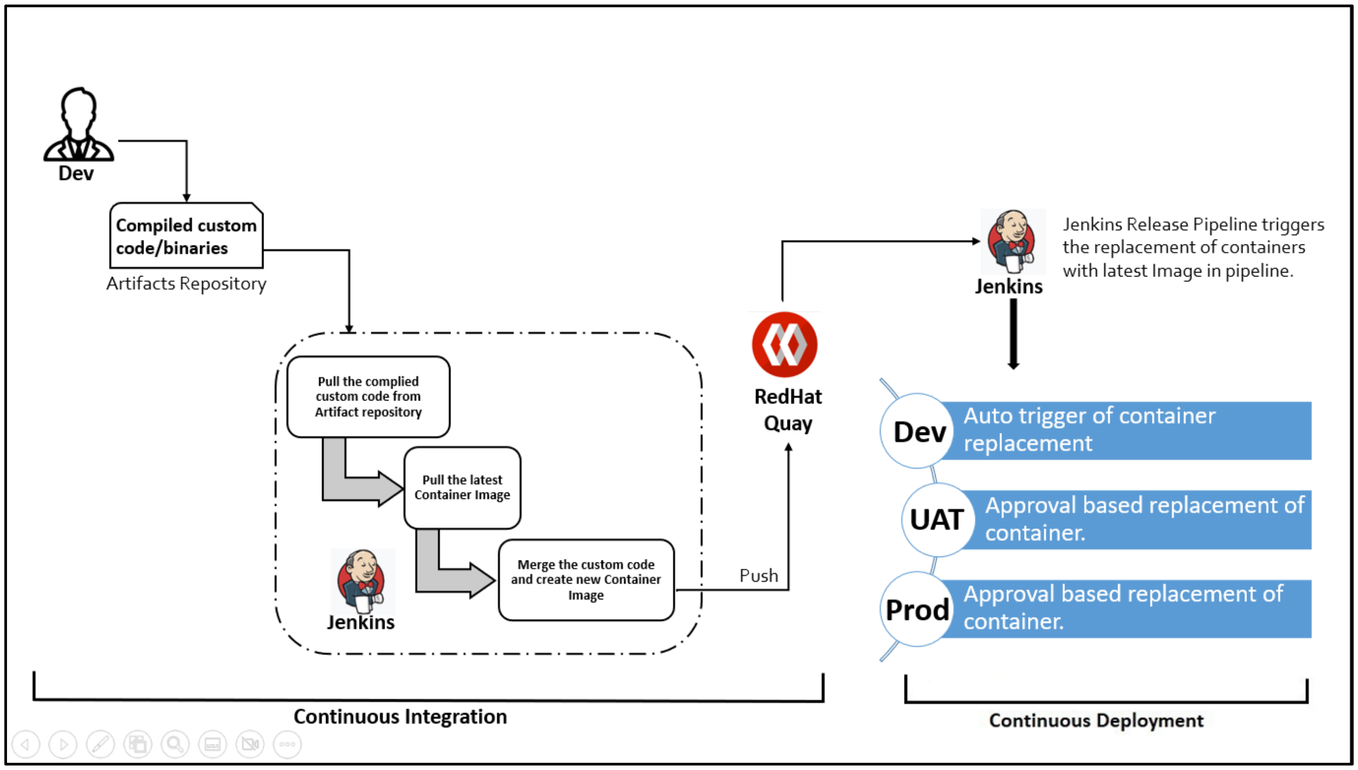
Dev
Compiled custom code/binaries
Artifacts Repository
Pull the complied custom code from Artifact repository
Pull the latest Container Image
Merge the custom code and create new Container Image
Jenkins
Push
RedHat Quay
Jenkins
Jenkins Release Pipeline triggers the replacement of containers with latest Image in pipeline.
Dev
Auto trigger of container replacement
UAT
Approval based replacement of container.
Prod
Approval based replacement of container.
Continuous Integration
Continuous Deployment

Figure 2.4

- For initiating custom code deployment, the Implementation team pushes the compiled custom code to the artifacts repository. An artifacts repository can be SVN, FTP, a Network drive, and so on.
- After that Jenkins pulls the compiled custom code from the artifacts repository and pulls the latest container image. Then, it creates a new container image after merging the custom code changes and pushes the newly created container images to the RedHat Quay Registry.
- In this architecture, the operation team have full access to initiate the build pipeline configured on the Jenkins server. The Implementation team does not have access to this Jenkins server. However, a common artifacts repository is shared for both teams.
- As soon as any container image is pushed to the RedHat Quay Registry, Jenkins Release Pipeline triggers the deployment to the Dev environment.
- UAT and Production deployments are based on approval and are available on-demand. To deploy to the UAT/Production environment, you need to trigger the UAT/Production deployment. Upon deployment trigger, an approval mail is sent to the project manager or the

concerned team. As soon as the project manager or concerned team approves the go-ahead, UAT/Production deployment gets started.

2.8.3 CI/CD pipeline of product’s hotfix

This section describes the CI/CD pipeline of the product’s hotfix.

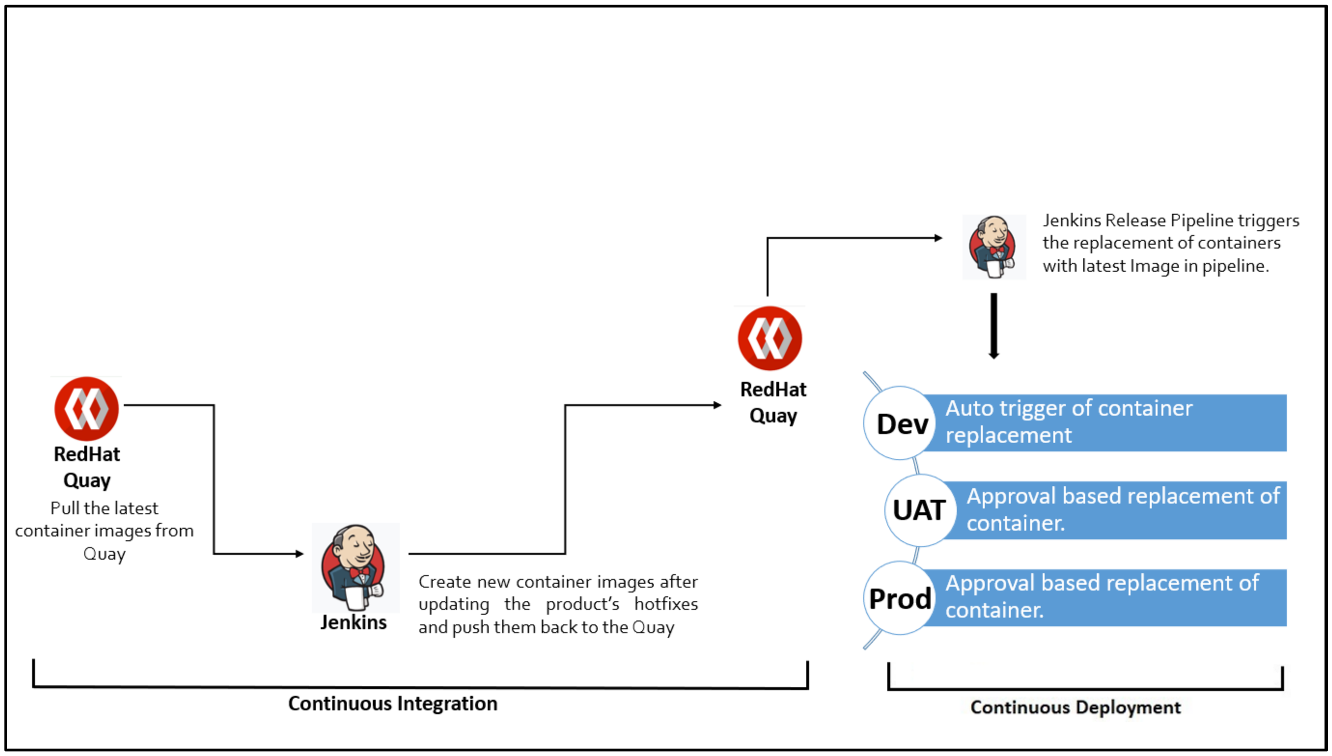
RedHat Quay
Pull the latest container images from Quay
Jenkins
Create new container images after updating the product’s hotfixes and push them back to the Quay
RedHat Quay
Jenkins Release Pipeline triggers the replacement of containers with latest Image in pipeline.
Dev
Auto trigger of container replacement
UAT
Approval based replacement of container.
Prod
Approval based replacement of container.
Continuous Integration
Continuous Deployment

Figure 2.5

To deploy the Newgen product’s hotfix, follow the below steps:

1. Pull the product’s base images or latest images that are already deployed in the current environment, from the container registry.
2. Update the hotfix files in the earlier running container images and create new container images. The deployment structure of these hotfix files (Dockerfile) is shared along with the hotfix files, which indicates how container images require update.
3. Push the newly created images to the container registry.
4. As soon as any container Image is pushed to the RedHat Quay Registry, Jenkins Release Pipeline triggers the deployment to the Dev environment.
5. UAT and Production deployments are based on approval and are available on-demand. To deploy to the UAT/Production environment, you need to trigger the UAT/Production deployment. Upon deployment trigger, an approval mail is sent to the project manager or the

concerned team. As soon as the project manager or concerned team approves the go-ahead, UAT/Production deployment gets started.

2.8.4 Deployment changes against file types

In our products, majorly there are four types of files:

- Deployable files *(*.war*, **._ejb.ear*)
- Dependent Libraries (**.jar*)
- Configuration Files (**.ini*, **.xml*)
- Database Scripts

Deployable files and libraries can be merged inside the Docker container as there are no dynamic changes in these types of files.

But as configuration files are dynamic so they must be kept outside the container.

For which the volume persistence is used and mapped to external disk storage like NAS server. So, whenever configuration changes are found in a custom code or product's hotfix, update the configuration files located at external disk storage along with updating Docker images.

For the database scripts, provide the DB scripts, which can be manually executed through Database Client software. However, if this operation requires automation, configure the DB script execution in Jenkins. In that case, create DB script execution job types in Jenkins.

2.8.5 Deployment downtime

Deployment downtime is dependent on the types of binaries used in custom code or hotfixes that need to be deployed. If changes are in deployable files or into dependent libraries, then downtime is not required, as Kubernetes provides the rolling deployment feature. In this case, deploy the latest Docker image, Kubernetes first launch the new container with the latest updated Docker image, make it runnable and when the latest container is ready to serve the requests, it starts deleting the existing container.

But, if changes required are in configuration files and database scripts then some downtime is required to update the configuration files and to execute the DB scripts, in the persistence mapped location.

For the Production environment, must take the requisite downtime to ensure that the product is working fine. To ensure that recheck the prerequisites and perform the smoke testing. Also, perform some regression testing before making it live.

2.8.6 Docker image management for rollbacks

For Docker Image management, two tag creation is recommended for each Docker image through Jenkins while building the Docker Images, before pushing them to the container registry.

For Example,

- omnidocs11.0web: sp1
- omnidocs11.0web: sp1 -build10 (where build-10 is the build pipeline number)

This means that you always have a copy of the Docker image of each build pipeline in the container registry so that whenever you require to rollback the current deployment, you can use the previous build pipeline number tag – Docker Image for rolling back the deployments.