NewgenONE OmniDocs RMS

Newgen Enterprise Products Containerization Guide for AWS

Version: 4.0 SP1

Newgen Software Technologies Ltd.

Table of contents

1 Preface 2
1.1 Revision history 2
1.2 Intended audience 2
1.3 Documentation feedback 2
2 Containerization of Newgen Enterprise Products 3
2.1 Key advantages 3
2.2 Deliverables – Product docker images for initial deployment 4
2.2.1 ECM Suite - OmniDocs 4
2.3 Deployment architecture 5
2.3.1 OmniDocs Vanilla deployment architecture 5
2.3.2 OmniDocs Multi-Tenant Deployment 7
2.4 Used tools and technologies 8
2.5 Kubernetes Autoscaling 10
2.6 Logging architecture 11
2.7 Container security 12
2.8 CI/CD pipeline 13
2.8.1 CI/CD pipeline of base products 14
2.8.2 CI/CD pipeline of custom code 15
2.8.3 CI/CD pipeline of product's hotfix 16
2.8.4 Deployment changes against file types 17
2.8.5 Deployment downtime 18
2.8.6 Docker image management for rollbacks 18
***Appendix* 18**

1 Preface

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 NewgenONE flagship products OmniDocs & RMS on AWS (Amazon Web Services).

1.1 Revision history

Revision Date	Description
April 2024	Initial publication

1.2 Intended audience

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

1.3 Documentation feedback

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

To help capture your feedback effectively, request you to share the following information in your email.

- Document Name:
- Version:
- Chapter, Topic, or Section:
- Feedback or Suggestions:

2 Containerization of Newgen Enterprise Products

The containerization support is enabled for Newgen flagship product OmniDocs & RMS on **AWS EKS (Elastic Kubernetes Service).** 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 is deployed on the underlying web server JBoss WebServer 5.7.x. EJB components is deployed on the underlying application server JBoss EAP 7.4.x.

2.1 Key advantages

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 the environment consistency that means it never face 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 of containers is far faster and easier than deploying additional VMs. Docker containers are auto-scaled up and scaled down depending on the CPU utilization and memory consumption. Not only containers but worker node instances can also scale up and scale down automatically in Kubernetes.
- **Easy distribution and portability:** As our application and their 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-restart that application inside the container.
- **Seamless deployment of updates (Rolling Deployment):** Deploying updates as a Docker image are far faster and very efficient. Whenever we 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 the user session also not gets expired, and the latest updates gets deployed.

2.2 Deliverables – 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 ECM Suite – OmniDocs & RMS

In the case of ECM Suite – OmniDocs & RMS, Newgen delivers the following Docker images for initial product deployment:

- OmniDocs+RMS Web Components
- OmniDocs Web Service Components
- OmniDocs+RMS EJB Components
- OmniDocs Add-on Services (Wrapper, Alarm Mailer, Scheduler, Thumbnail Manager and LDAP)
- EasySearch (Apache Manifold only)
- Text Extraction Manager or Full-Text Search (TEM or FTS)
- OmniScan Web Components

OmniDocs Services container consists of seven different add-on services of OmniDocs & RMS. If there is a requirement, you can split them into separate containers.
EasySearch container contains the Apache Manifold which is freeware software.
If OmniScan Web components are required for scanning services, then it also constitute a separate Docker instance.

Docker Image Name	Docker Image Size
OmniDocs Web	771 MB
OmniDocs Web Service	482 MB
OmniDocs EJB	706 MB
OmniDocs Services	1.02 GB
EasySearch (Apache Manifold only)	695 MB
TEM or FTS	251 MB
OmniScan Web	439 MB
SharePointAdapter	176 MB
OmniDocs Wopi	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.

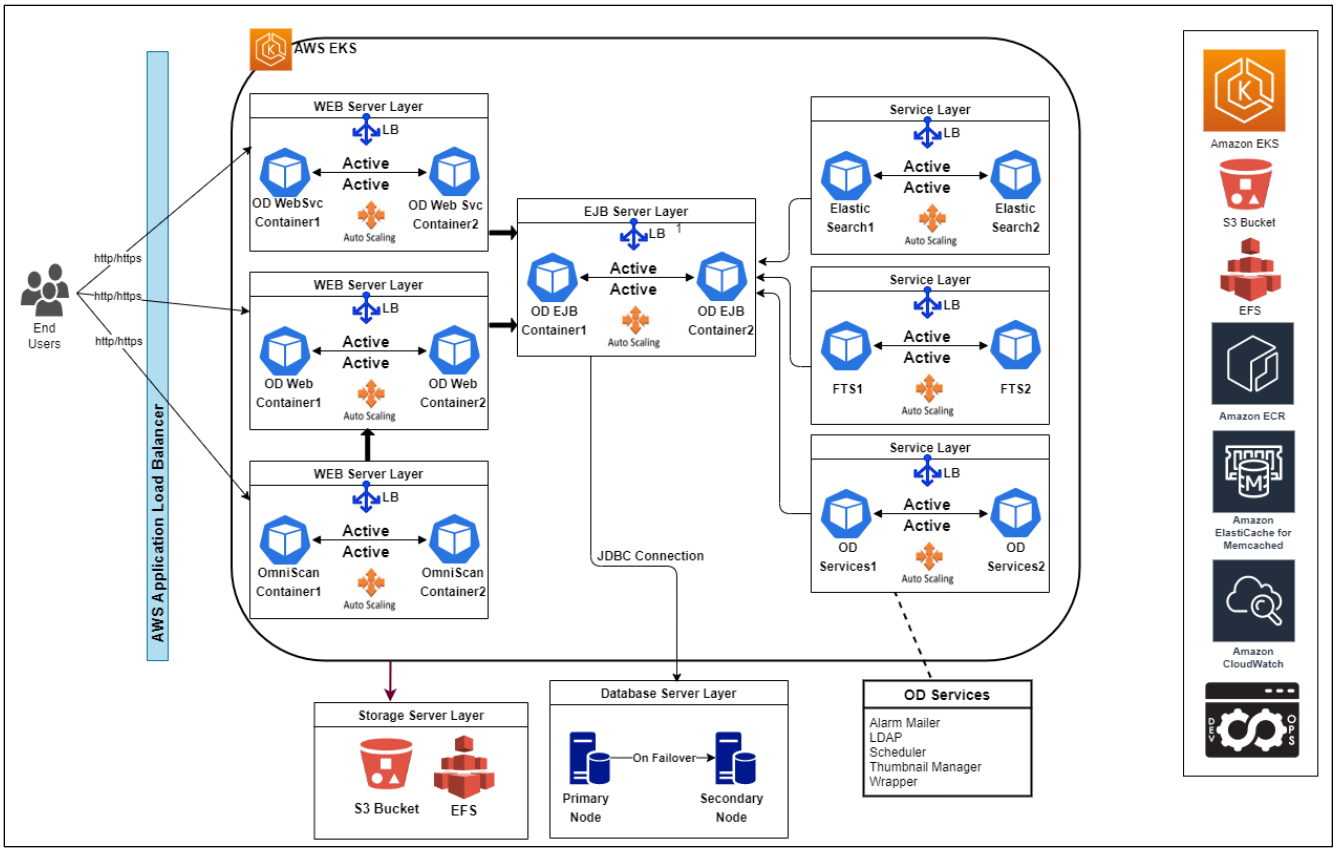

Figure 2.1

- In this architecture, all the user requests come to AWS Application Load Balancer, which is created and managed by the Ingress controller. An ingress controller is an object running inside the Kubernetes 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.newgendocker.com* then the ingress controller redirects the user request to OmniDocs & RMS WEB containers. If the URL is *omnidocswebservice.newgendocker.com* then the ingress controller redirects the user request to the OmniDocs 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 ElasticSearch, 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.
- Using the AWS services to configure and manage the Docker containers in the left panel.
- Using AWS EKS as container orchestration, an S3 bucket to store the product's PN files, AWS EFS for the configuration files, and log files persistence. The Docker images are stored in a private container repository like AWS ECR (Elastic Container Registry). The AWS Elastic MemCached is used to persist the user's web session. The AWS CloudWatch is used to monitor the container's health. The CI or CD pipeline is implemented for end-to-end deployment.

2.3.2 OmniDocs & RMS Multi-Tenant Deployment

This section describes the deployment of the OmniDocs & RMS Multi-Tenant.

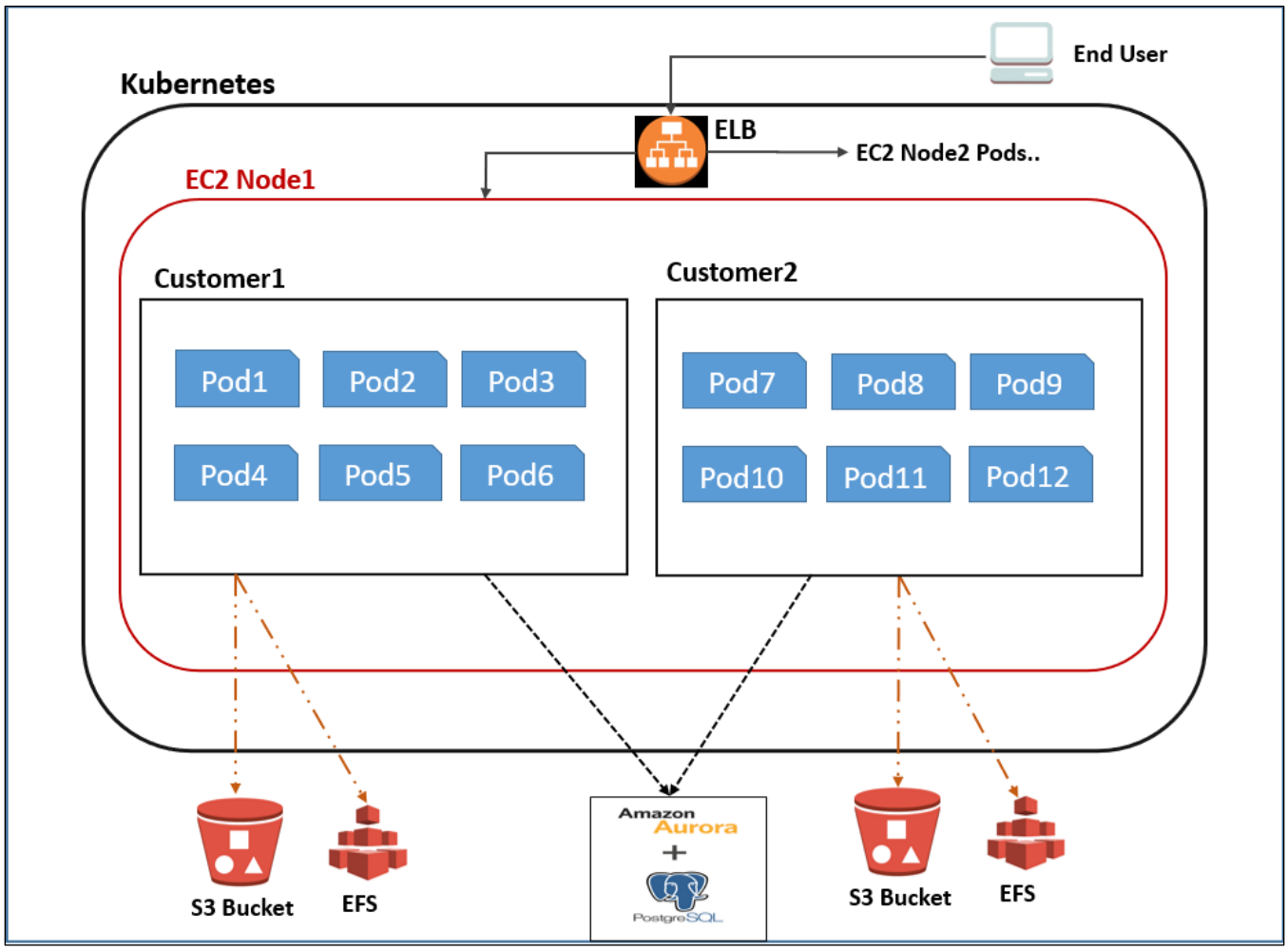

Figure 2.2

- In the case of Multi-Tenant Deployment, containers of multiple customers are running on the same EC2 nodes.
  For example, in the above architecture diagram, POD1 to POD6 are running for customer1 and from POD7 to POD12 are running for customer2. Also, all the containers are running on the same EC2 node.
- In container orchestration, define the resource limit to each container so that one container must not consume all the resources available on the EC2 node due to which other container's performance remains intact.
- In this architecture, end-users' requests come to the Elastic Load Balancer (ELB) which is managed by the Ingress controller in Kubernetes. After that Ingress controller routes the

incoming requests to the target Kubernetes services for different customers according to host-based routing rules. Host-based routing is a capability of ALB redirects the user requests to the right service based on the request-host header.
For example, you can set the rules as described below:
    - IF URL is *customer1dev.aws.co.in* THEN redirect to POD1.
    - IF URL is *customer2dev.aws.co.in* THEN redirect to the POD7.
      Where POD1 and POD7 are OmniDocs Web containers for customer1 and customer2 respectively.
- In this architecture, the same ELB and EC2 nodes can be used for multiple customers.
- Containers can be different for each customer depending on the changes in custom code.
- S3 bucket and EFS is different for each customer to maintain the data security.
- The database server can be either Microsoft SQL, Oracle, PostgreSQL, or AWS Aurora PostgreSQL. You can use either the same database server for all customers or different database servers for each customer. It depends on the business requirement.

2.4 Used tools and technologies

The following tools and technologies are used for this containerization approach

- **Container Orchestration: AWS Elastic Kubernetes Service**
  Amazon Elastic Kubernetes Service (Amazon EKS) is a fully managed Kubernetes service. EKS runs the Kubernetes management infrastructure across multiple AWS Availability Zones, automatically detects and replaces unhealthy control plane nodes, and provides on-demand, zero downtime upgrades, and patching. EKS automatically applies the latest security patches to your cluster control plane.
- **Worker Node Host OS: Amazon Linux 2**
  Here, the EC2 host can also be referred to as the Kubernetes worked node which 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 is built with another container OS on-demand basis.
- **Docker Image Registry: AWS Elastic Container Registry (ECR)**
  AWS Elastic Container Registry is used to store the Product's Docker Images and by default Image scanning is enabled.
- **AppServer: JBossEAP 7.4.x**
  JBoss EAP 7.4.x is used to deploy 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 1.11.x**
  Inside the Docker images, OpenJDK 1.11.x is bundled. If there is some special requirement, then you can build the Docker images with Oracle JAVA on-demand basis.
- **Database: Aurora PostgreSQL Compatible 14.x**
  Newgen Products support three database servers: Microsoft SQL Server, Oracle & PostgreSQL. The supported database versions for products are available in the Product Support Matrix.
- **PN Files Storage Server: AWS S3 Bucket**
  PN files are encrypted files that contains Newgen products added, uploaded, and scanned documents. Also, Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance.
- **Log Files Persistence: AWS EFS**
  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 is 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 the AWS EFS. Amazon Elastic File System (Amazon EFS) provides a simple, scalable, fully managed elastic NFS file system to use with AWS Cloud services and on-premises resources. Here, using EFS to persist the Newgen Product's logs as well as Newgen Product's configuration files.
- **Configuration Files Persistence: AWS EFS**
  Same as log files, using the AWS EFS to persist the configuration files.
- **Load Balancer: AWS Elastic Application Load Balancer**
  Here, not using the AWS ALB directly. Instead, using the AWS ALB Ingress controller. The AWS ALB Ingress Controller for Kubernetes is a controller that triggers the creation of an Application Load Balancer (ALB). The Ingress resource configures the ALB to route HTTP or HTTPS traffic to different pods within the cluster.
- **Session Persistence: AWS Elastic MemCached**
  MemCached 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.
- **Container Health Monitoring: CloudWatch Container Insight**
  Container insights is a fully managed CloudWatch service used to collect, aggregate, and summarize metrics and logs of containerized applications deployed on ECS or EKS service. The metrics include the utilization of resources such as CPU, memory, disk, and network. Container insights also provide diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly. The metrics that Container Insights collects are available in CloudWatch automatic dashboards.

- **Container Runtime Security: Falco**
  Falco is an open-source CNCF Sandbox project from Sysdig, that detects abnormal behavior in the EKS cluster and applications deployed to that cluster. It provides runtime security for containerized applications and their underlying host systems. By instrumenting the Linux kernel of the container host, Falco can create an event stream from the system calls made by containers and the host. Rules can then be applied to this event stream to detect abnormal behavior.

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).
   a. **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 autoscaler thrashing.
      - In HPA, the autoscaler 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.
   b. **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 the resource utilization.

2.6 Logging architecture

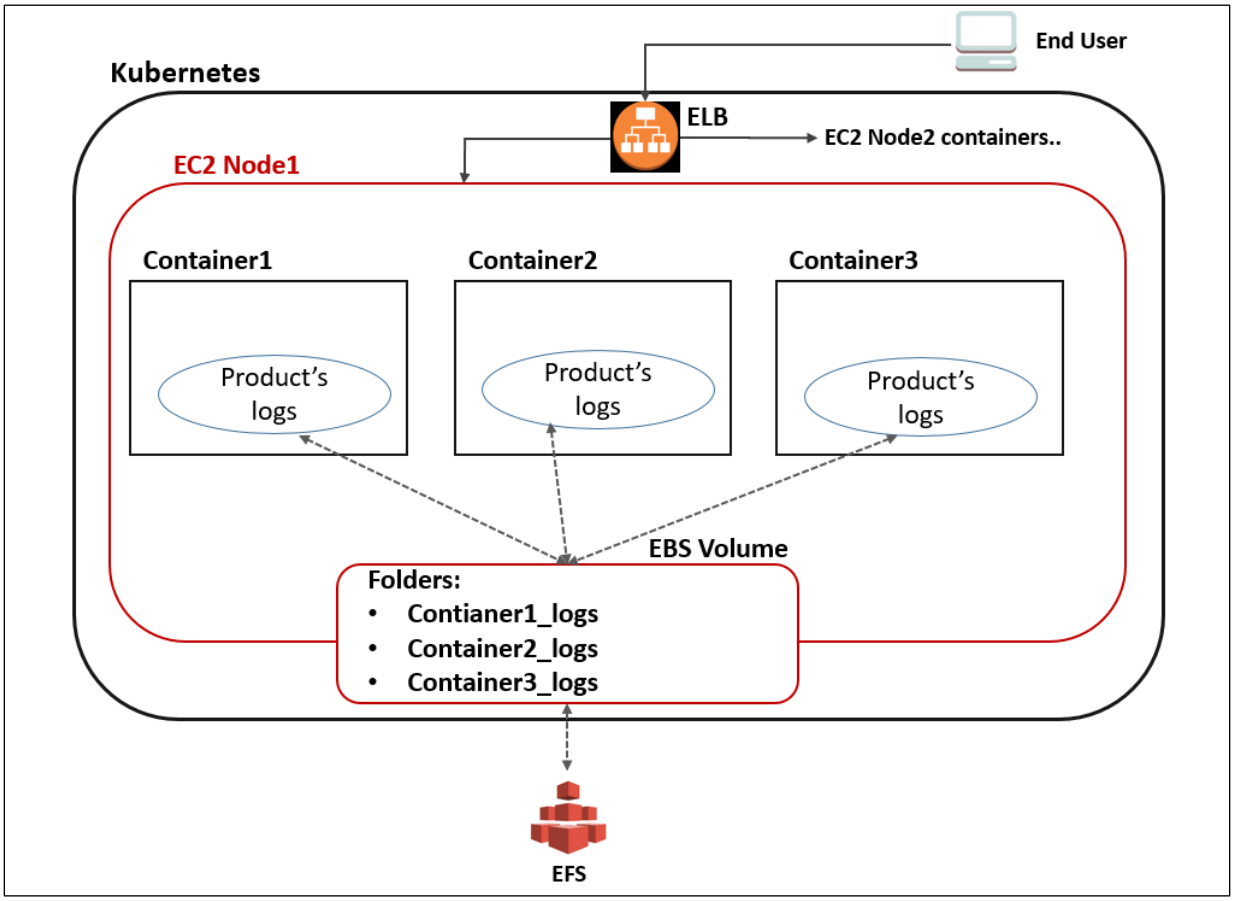

Figure 2.2

- 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, and it can be difficult to get the data out of the container if it is required later. Thus, to persist these types of data the AWS EFS. Amazon Elastic File System (Amazon EFS) provides a simple, scalable, fully managed elastic NFS file system to use with AWS Cloud services and on-premises resources.
- Since the product's log files are dynamic in nature and are created at runtime so they must be kept outside the container. Therefore, AWS EFS is used to persist in the Newgen Product's log files.
- To persist the data in EFS, the product's logs are mounted at two levels:
  - Container to EC2 node's EBS volume.
  - EBS volume to EFS
- Thus, all the product's logs created inside the containers is mounted to the EC2 instance's EBS volume at some predefined folder structure.

For example, container1's product logs are stored inside the container1_logs folder on an EC2 instance, and container2's product logs are stored inside the Container2_logs folder on an EC2 instance, and so on.

- Further, all the log folders created in EBS volume is mounted to EFS so that whenever an EC2 instance goes down, our product's logs must always be available in EFS. 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 EBS volume as well as in EFS 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 AWS Elastic Container Registry.
  AWS ECR 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. Amazon ECR uses the Common Vulnerabilities and Exposures (CVEs) database from the open-source **Clair** project and provides a list of scan findings.
- **No container should 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 absolutely 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.
- **Falco Implementation:** Falco Implementation is for container runtime security. Falco is an open-source CNCF Sandbox project from Sysdig, that detects abnormal behavior in the EKS

cluster and applications deployed to that cluster. It provides runtime security for containerized applications and their underlying host systems. By instrumenting the Linux kernel of the container host, Falco can create an event stream from the system calls made by containers and the host.

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. We 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.

Persistent Storage:

AWS EFS persists the Newgen product's configuration files and log files. EFS is the fully managed storage service of AWS. Access to Amazon EFS requires credentials that AWS can use to authenticate your requests and those credentials must have permission to access AWS resources such as EFS. All the data is kept in EFS is highly secure and no one can access the EFS without the required permission.

2.8 CI/CD pipeline

Use the CI/CD pipeline to manage deployments with Kubernetes orchestration on cloud platforms. Here at Newgen, there is a segregation of the Build Pipeline and Release Pipeline into two parts. The Build Pipeline is done by the Jenkins server which can be installed on an on-premises machine or a cloud machine. The Release Pipeline is managed by AWS CodePipeline cloud service to manage the Release pipeline.

2.8.1 CI/CD pipeline of base products

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

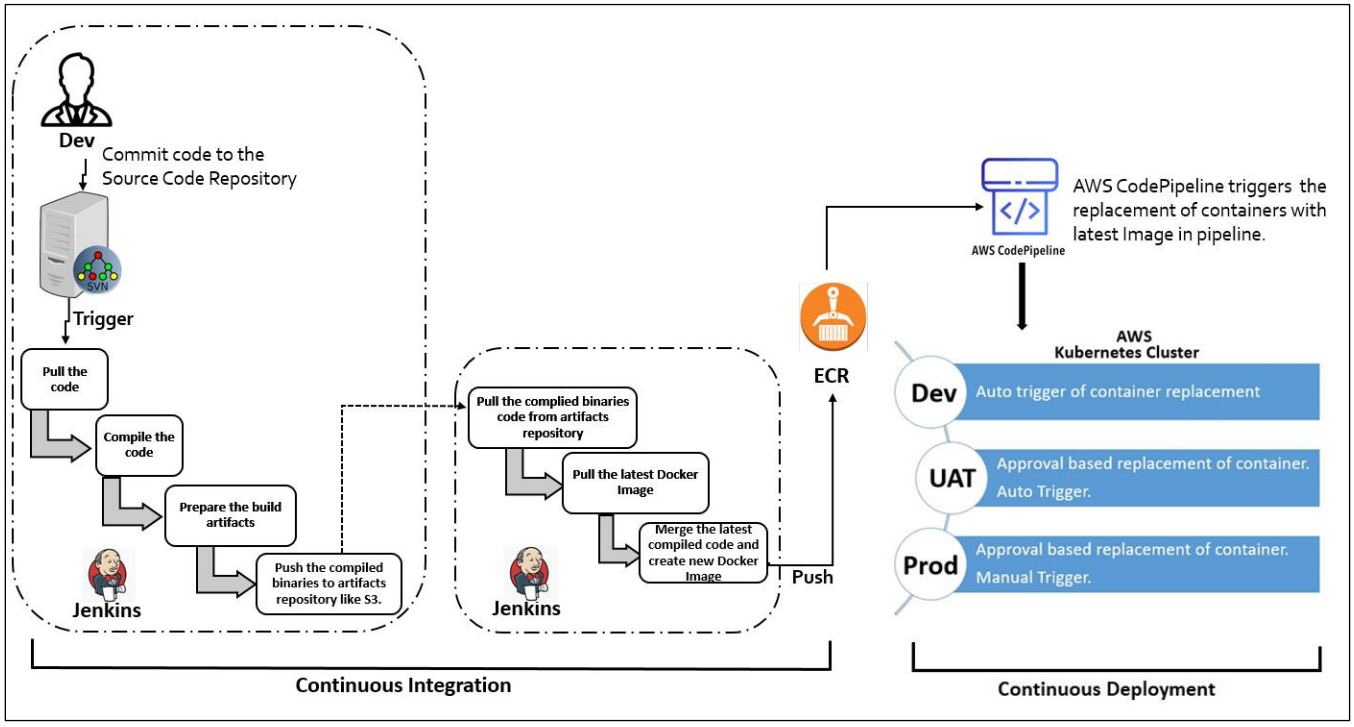

Figure 2.3

- The build pipeline is segregated into two teams: the implementation team and the operational team. The implementation team keep pushing their code to the source code repository. As soon as they commit their code to the source code repository, the 1st Jenkins pipeline is triggered. It pulls the latest code, compile them and prepare the build artifacts and push them to the artifact's repository like AWS S3 bucket or some other similar kind of media. With this, the implementation team support is not required in Docker's image creation. After that 2nd Jenkins pipeline gets triggered. It pulls the compiled build artifacts from the artifacts repository, pull the latest Docker image and create a new Docker after merging the latest code changes. Last, it pushes the newly created Docker images to Image Repository like AWS ECR.
- In this architecture, there are three stages: Dev, UAT, and Production and in each stage, deployment is quite different. You can have some more stages depending on the requirements.
- As soon as any Docker Image is pushed to the AWS ECR, AWS CodePipeline triggers the deployment to the Dev environment.

- UAT and Production deployments are approval based and they are called on-demand. Once you are ready to deploy to the UAT environment, trigger the UAT deployment. When that deployment is triggered, an approval mail is sent for the approval. After receiving approval, the UAT deployment starts automatically.
- The production deployment is also approval-based, but it is multi-level approval. To deploy to a production environment, approvals from all stakeholders are required. Most importantly, after receiving approvals from all the stakeholders, deployment to the production environment cannot be triggered automatically. A manual intervention mail is sent to the engineer who is supposed to deploy to production with a checklist. After verifying that the checklist points are covered or not. If not, then the deployment to the production gets rejected.

2.8.2 CI/CD pipeline of custom code

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

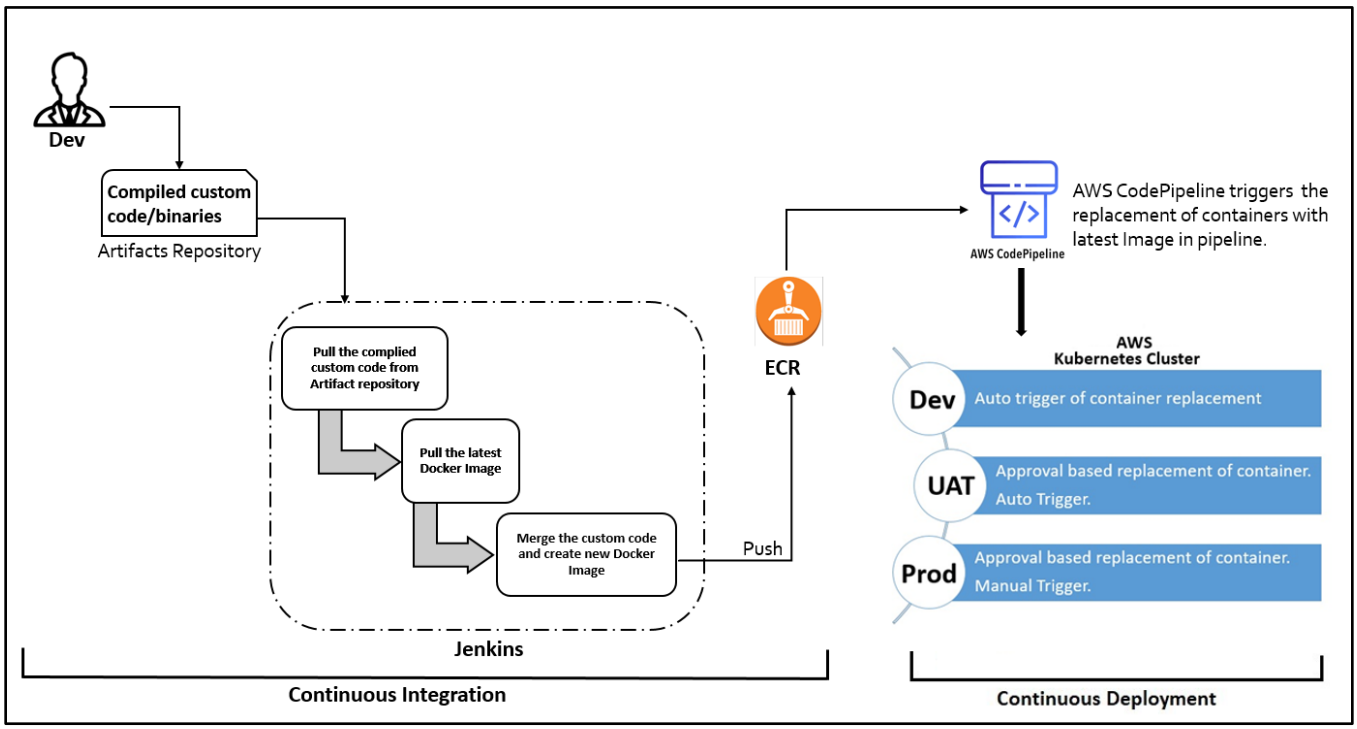

Figure 2.4

- The Custom Code deployment pipeline can also be configured and maintained in Jenkins.
- For initiating custom code deployment, the Implementation team pushes the compiled custom code to the artifacts repository. Artifacts repository could be anything like AWS S3 bucket, SVN, FTP, JFrog, and so on.

- After that Jenkins pulls the compiled custom code from the artifacts repository, it pulls the latest Docker image and create a new Docker. After merging the custom code changes, it pushes the newly created Docker images to Image Registry.
- In this architecture, Cloud/Infra team have full access to initiate the build pipeline configured on the Jenkins server. The Implementation team having no access to this Jenkins server. However, a common artifacts repository is shared for both teams.
- As soon as any Docker Image is pushed to the AWS ECR (Elastic Container Registry), AWS CodePipeline triggers the deployment to the Dev environment.
- UAT and Production deployments are approval based and they are called on-demand. To deploy the UAT environment, trigger the UAT deployment. When that deployment is triggered, an approval mail is sent to the project manager or team concerned. After receiving approval, the UAT deployment started automatically.
- The production deployment is also approval based but it is multi-level approval. To deploy to a production environment, approvals from all stakeholders are required. Most importantly, after approvals from all the stakeholders, deployment to the production environment cannot be triggered automatically. A manual intervention mail is sent to the engineer who is supposed to deploy to production with a checklist. After verifying that all the checklist points are covered or not. If not, then the deployment to the production gets rejected.

2.8.3 CI/CD pipeline of product's hotfix

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

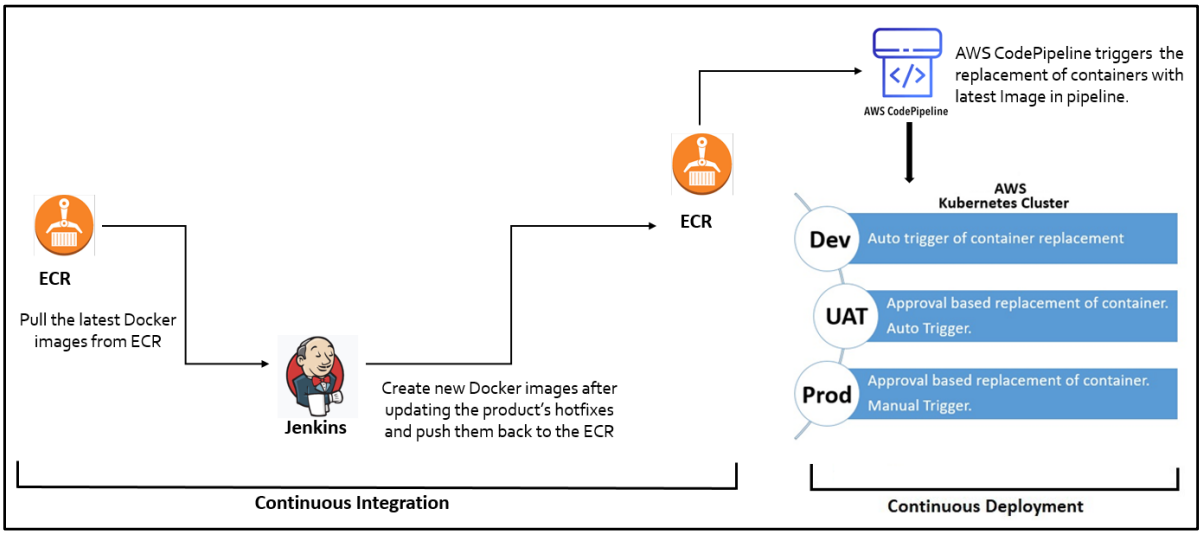

Figure 2.5

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 Docker images and create new Docker images. The deployment structure of these hotfix files (Dockerfile) is shared along with the hotfix files, which indicates how to update the Docker images.
3. Push the newly created images to the container registry.
4. As soon as any Docker image is pushed to the AWS ECR (Elastic Container Registry), AWS CodePipeline triggers the deployment to the Dev environment.
5. UAT and Production deployments are approval based and they are called on-demand. To deploy the UAT environment, trigger the UAT deployment. When that deployment is triggered, an approval mail is sent to the project manager or team concerned. After receiving approval, the UAT deployment starts automatically.
6. The production deployment is also approval-based, but it is multi-level approval. Approvals of all the stakeholders are required to deploy to a production environment. Most importantly, after approval from all the stakeholders, deployment to the production environment cannot be triggered automatically. A manual intervention mail is sent to the engineer who is supposed to deploy to production with a checklist. After verifying that whether all the checklist points are covered or not. If not, the deployment to the production gets rejected.

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 in nature so they must be kept outside the container.

For which the volume persistence is used and mapped them to external disk storage like AWS EFS. 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 scripts 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 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 DockerImage management, two tags 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:latest
- omnidocs11.0web:build-10 (where build-10 is the build pipeline number)

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

Appendix

This guide contains third-party product information about configuring Amazon Web Services (AWS) CodePipeline for Container Deployment on EKS and AWS Kubernetes Cluster. Newgen Software Technologies Ltd does not claim any ownership on such third-party content. This information is shared in this guide only for convenience of our users and could be an excerpt from the AWS documentation. For latest information on configuring the AWS Kubernetes Cluster and AWS CodePipeline refer to the AWS documentation.