M4





CICD PIPELINE DOCUMENT

F:\Excel-SARAS_Logos\logo.gif



*CICD PIPELINE DOCUMENT*

**Submitted by Excelsoft Technology**

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Author** |
| 1.1 | 05-Sep-2024 | Sandesh Gowda P |

**CONTENT**

# [1. An Overview of CI/CD Pipeline](#_An_Overview_of)……………………………...………………….....4

# [2. Activities performed within ES Network](#_Activities_executed_in)……………………………………....5

# [3. Activities performed within SHIPHATS](#_Activities_which_are)…………………………………...…….6

# [4. Activities performed within GCC](#_Activities_which_are_1)………………………………………...…….....7

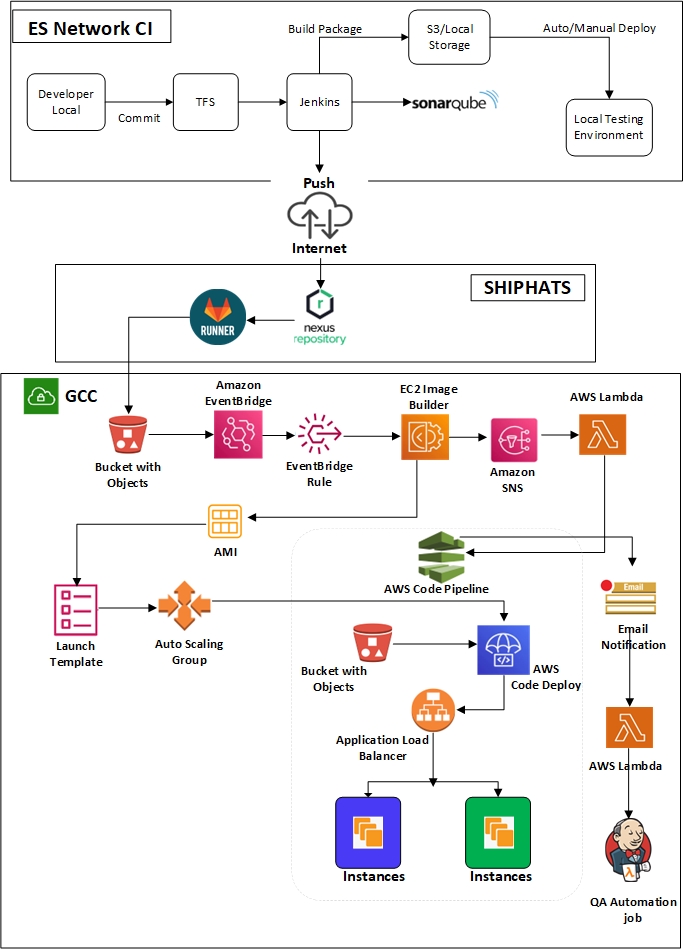
# [5. Deployment description](#_Deployment_description_1)…………………………………………………...……......8

# [6. Pros and Cons](#_Pros_and_Cons)……………………………………………………………………….….....9-10

# An Overview of CI/CD Pipeline

# Introduction

This document provides an overview of CI/CD Pipeline configuration to Build, Setup, Test, Release and Deploy the Package/Application for all the modules. This document represents a ‘point in time’ setup, or snapshot of all the environments such as UAT, Pre-Prod and Production.



The above diagram explains the complete CI/CD Pipeline workflow, and this can be divided into three parts.

* Part 1 - Activities that are performed within Excelsoft (ES) using CI/CD tools.
* Part 2 - The tasks that are executed within SHIPHATS.
* Part 3 - Activities that are performed within GCC.

# Activities performed in ES network

A diagram of a cloud computing process

Description automatically generated

# Block Diagram – Part 1

1. The developer performs code changes and checks in to TFS.
2. SonarQube will be used for Static scan code analysis and will be executed based on requirement, reports are shared only during major releases.
3. CI jobs are configured and executed in Jenkins every 30 minutes to ensure that code changes made by developers have no errors or code breakage.
4. Every night, a scheduled job in Jenkins triggers the nightly build process, collecting the latest source code from TFS to generate the latest package/artifacts.
5. After successful package generation, an automated deployment job will be triggered to deploy the application to the relevant internal servers. In the event of a build failure, an email notification will be sent to the project teams, prompting their intervention the following morning.
6. A copy of the deployed package will be saved in the TFS server with a unique label number. The previously deployed package will be retained as a backup on the respective servers. In the event of any failure, the previous package can be easily deployed from the TFS server or the backup package on the respective server can be used for rollback.
7. The successful build package also triggers the Automation Test Suite, and the generated report will be emailed to the key team members. Additionally, the QA team executes a smoke test the following day. Upon confirmation, the same build package/artifact will be moved to UAT/CAT based on demand. The build/package with the assigned label number will also be uploaded to the Shiphats nexus repository.

# Activities performed within SHIPHATS and CD tools used.

A logo with a hexagon and a letter r

Description automatically generated

# Block Diagram – Part 2

1. Nexus Repo is used to manage the uploaded Artifacts. The latest build package from the ES network will be uploaded to nexus repository manually by using SEED device.
2. GitLab Runner is configured in such a way that it uploads the package to the S3 bucket which is embedded within the GCC network. This can either be done manually or by scheduling the jobs.
3. Separate jobs have been configured for each module.

# Activities performed within GCC

A diagram of a software

Description automatically generated

# Block Diagram – Part 3

1. Once the package reaches from GitLab to the S3 bucket, the Event bridge rule will trigger the EC2 ImageBuilder Pipeline.
2. The EC2 ImageBuilder pipeline will perform the application deployment, the output AMI will be updated to the respective Launch Template.
3. The Auto Scaling Group is used to spin up the instances by referring to the latest AMI which is associated in the launch template to ensure that the instances are up to date.
4. As soon as the EC2 Image Builder execution is completed, the SNS notification is sent to the Lambda function.
5. The Lambda function will trigger the respective AWS Code Pipeline.
6. Code Pipeline will initiate the Blue-Green Deployment process.
7. Switching of the environments will be done via code pipeline with an option called Re-route traffic.
8. Re-routing traffic can either be done manually or it can be set to re-route traffic immediately once the replacement instances are up.
9. Notification rule has been configured such that when the pipeline completes, the SNS notification will be sent to Lambda function to trigger the respective Jenkins jobs (This applies for the UAT/Pre-Prod environment only).
10. The old servers will be retained and will be terminated later manually.
11. Step 1 to Step 9 will be repeated for all the modules.
12. The same process will be followed in each environment (UAT, CAT, and Prod environments).

# Deployment description

**5.1 Full Build Deployment**

The latest package will be uploaded manually from ES Network to NexusRepo in Shiphats network via internet, and then the Gitlab job will be triggered manually (can be scheduled if required) so that it downloads the package from the NexusRepo and pushes it to the S3 bucket.

Once the above process is completed, the pipeline will be triggered automatically in the GCC network to perform the deployment process (using CD Pipeline deployment approach) by using the AWS services.

**5.2 Incremental Deployment (Manual process)**

A new instance will be created from the live instance AMI and incremental patch will be deployed to the new instance.

An incremental patch typically includes updates, bug fixes, security patches, or other changes such as the software running on the instance. This step ensures that the new instance is up to date and includes any necessary improvements.

Once the downtime is confirmed, the latest AMI will be attached to the ASG and deployed to the live environment.

The application or workload running on the new instances will be tested and verified.

This testing phase aims to confirm that the patch deployment did not introduce any issues or disruptions, and it also confirms that the updated changes are working as expected.

# Pros and Cons

Here are some pros and cons of the described approach for Continuous Deployment (CD) using AWS services.

**Pros**

**1. Automated Deployment:** Automating the deployment process ensures that new versions of the application are deployed consistently, reducing the manual effort required. Automation reduces the risk of human error, leading to more reliable deployments.

**2. Blue/Green Deployment**

**Minimized Downtime:** Blue/green deployments allow for near-zero downtime by routing traffic to new instances only once they are verified to be ready.

**Safe Rollback:** If there is an issue with the new deployment, traffic can be quickly switched back to the previous version, ensuring minimal disruption.

**3. AMI-based Deployment:**

**Reproducibility:** Using Amazon Machine Images (AMIs) ensures that the exact same environment is deployed each time, leading to consistent performance and behavior across instances.

**4. Integration with S3 and EC2 Image Builder:**

**Centralized Artifact Storage:** Using S3 for storing the published copy ensures that the artifacts are centrally available and durable.

**Automated Image Creation:** EC2 Image Builder automates the creation and distribution of AMIs, integrating seamlessly into the deployment pipeline.

**5. Code Pipeline Integration:**

**Orchestration:** AWS Code Pipeline provides a visual interface to manage and visualize the entire deployment process, ensuring that all steps are executed in order.

**Notifications and Monitoring:** Code Pipeline can be integrated with AWS SNS, CloudWatch, and other services to provide notifications and detailed monitoring of the pipeline.

**Cons**

**1. Maintenance:** Maintaining the pipeline, managing dependencies, and handling updates to AWS services require ongoing effort and expertise.

**2. Cost:**

**AWS Charges:** Using multiple AWS services can incur significant costs, especially for storage (S3), compute resources (EC2 instances for Image Builder), and pipeline execution (Code Pipeline).

**3. Resource Usage:** Running instances for image creation and maintaining multiple AMIs (blue/green) can lead to higher resource usage and associated costs.

**4. Deployment Time**

**Patch Deployments:** Incremental patches/hotfixes cannot be deployed into the live servers using the CD Deployment approach.

**Image Creation Time:** The process of building and testing new AMIs can take time, potentially slowing down the deployment process.

**Pipeline Execution:** Each step in the Pipelines can introduce delays, particularly if there are long-running integration or deployment tests.

**Package Extraction:** For UAT/Pre-Prod, as the package preparation for each environment differs, the extraction takes more time since the size will be larger for all the modules.

**BIL issues:** Resolving BIL issues will take more than the given designated time. Pipeline will take at least 4-6hrs in prod and 8-10hrs in UAT to release the new changes. In addition to this, QA verification and sign-off must be provided to release the newly deployed changes.

**Deployment requests:** Frequent deployment requests will not be possible as the entire package preparation along with the pipeline execution takes a lot of time.

**5. Error Handling and Debugging**

**Complex Debugging:** Troubleshooting issues within a multi-stage pipeline can be complex, particularly when problems arise during image creation or instance launching.

**Rollback Complexity:** While blue/green deployments simplify rollback in production, managing AMIs and ensuring that all stages of the pipeline correctly handle rollbacks can be intricate.

**Conclusion:**

The described approach leverages powerful AWS services to automate and streamline the deployment process, providing benefits like reduced downtime, consistent deployments, and strong integration capabilities. However, it also introduces complexity, potential costs, and some inflexibility that need to be managed effectively. Ensuring the right balance between automation, cost, and complexity is key to a successful CI/CD pipeline.