TM DevOps – Use Case

TM DevOps – Use Case

Document Details

Use Case Name	**TMDevOps – Use Case04**
First Draft	**10th Dec 2017**
Author	**Amrendra Kumar**
Reviewed By	**Pradeep Narayanaswamy**

TM DevOps – Use Case

Contents

TM DevOps – Use Case

Scope

This document provides a detailed use case study on development and operations (DevOps) using Continuous integration and continuous delivery (CI/CD) including AWS services and third-party application, with guidelines for implementation.

About Customer

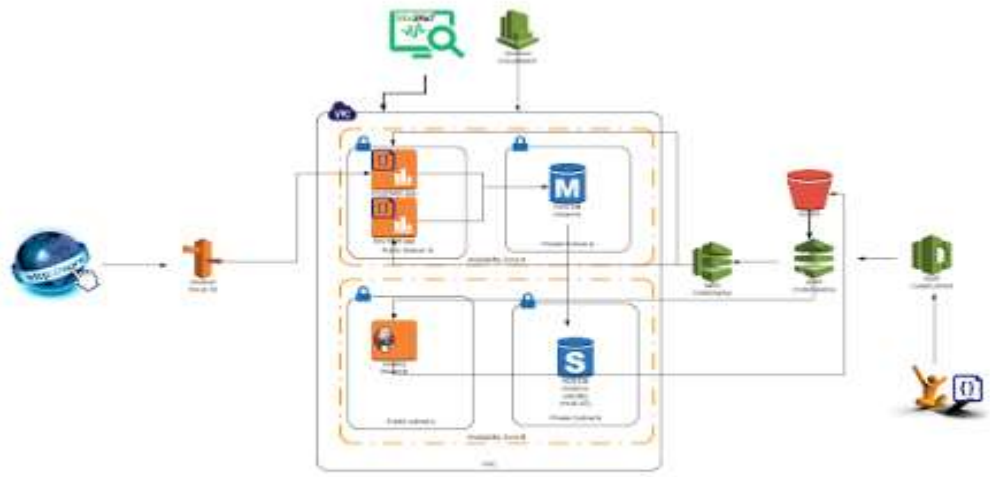
Laranya Infoedge Pvt Ltd (LIPL) aims to bring a digital platform to empower both consumers and retailers and bridge the information gap between them. The platform allows consumers to identify stores & products with ease and allows retailers to build their brand loyalty.

Pre-Conditions/Trigger

- Rapidly changing competitive landscapes, evolving security requirements, and performance scalability triggered the need of implanting CI/CD
- Requirement for frequent updates, regular patching and frequent configuration changes was also one of the reasons.
- Laranya Infoedge Pvt Ltd was not full or minimal DevOps compliance for their software development life cycle policy (SDLC) for their stack of PHP for main app, Angular for front UI

Architecture

DevOps – CI/CD Architecture for Laranya Infoedge Pvt Ltd
Code Repository: AWS CodeCommit
Build: Jenkins 2.7.9 using git and maven build plugin
Deployment: AWS EC2 + Apache + PHP 5.6



TM DevOps – Use Case

CI Integration Process

- We enabled freestyle job using AWS CodeCommit Repository to trigger build when developers pushed any code changes to the repo and its branches using SCM poll
- Build use maven build plugin was used to generate jar or war file
- Using AWS Code Pipeline and CodeDeploy Plugin zip file we deployed Apache server on AWS EC2
- Build Rollback plan was enabled for build failure
- Email notifications was sent to the team on build status

Implementation

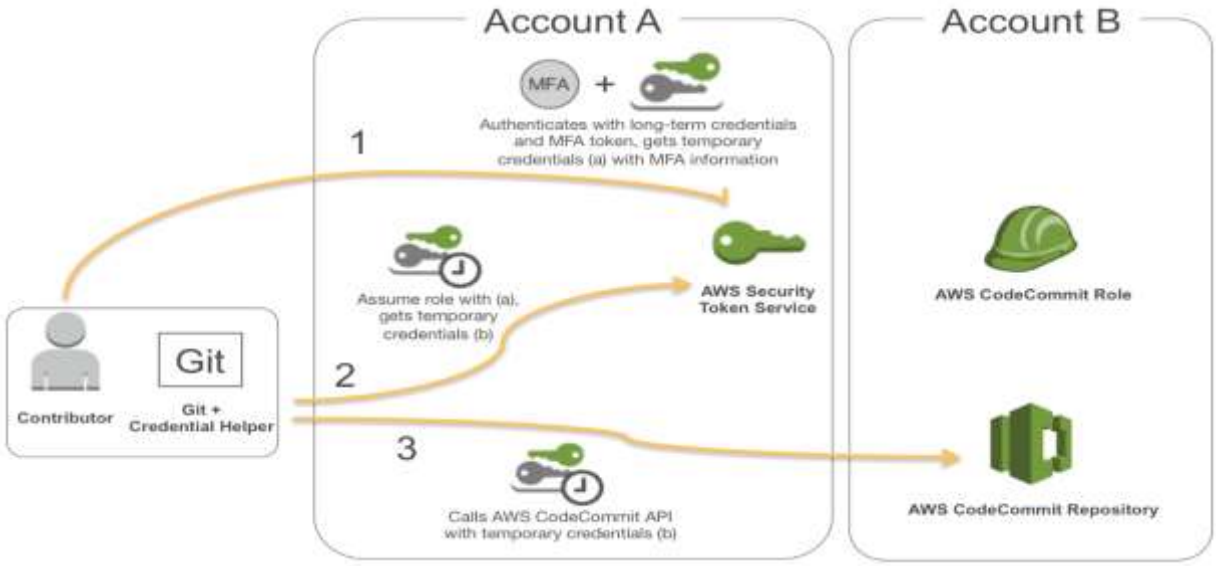
- New code was submitted on one end, tested over a series of stages (source, build, staging, and production), and then published as production-ready code.
- Each stage of the CI/CD pipeline was structured as a logical unit in the delivery process. In addition, each stage acted as a gate that vets a certain aspect of the code.
- As the code progressed through the pipeline, the assumption was that the quality of the code was higher in the later stages because more aspects of it continued to be verified.
- Results from the tests were immediately sent to the team.
- We made sure that all of the developers regularly commit their code to a central repository (such as one hosted in CodeCommit or GitHub) and merged all changes to a release branch for the application
- Frequent commits and merges with complete units of work were recommended for the team to develop discipline and are encouraged by the process.
- We created unit tests for their applications and to ran these tests before pushing the code to the central repository.
- When the code is pushed to a branch in a source code repository, a workflow engine monitoring that branch will sent a command to a builder tool to build the code and run the unit tests in a controlled environment.
- The build process was sized appropriately to handle all activities, including pushes and tests that happened during the commit stage, for faster feedback.
- Other quality checks, such as unit test coverage, style check, and static analysis, was processed in this stage.
- Finally, the builder tool created one or more binary builds and other artefacts, like images, stylesheets, and documents, for the application.
- After the staging environment was built using infrastructure as code (LAC), we successfully built a production environment in the same way.
- The final phase in the CI/CD pipeline was continuous deployment, which included full automation of the entire software release process including deployment to the production environment

TM DevOps – Use Case

Security with DevOps

Continuous Integrations (Code)

We authenticated Private Code repositories on AWS CodeCommit using two methods, one is SSH authentication and the other HTTPS authentication. We used HTTPS git credentials to get authenticated with AWS CodeCommit. AWS Code Commit also supports MFA authentication to prevent accidental pushes. Code Pull for Jenkins build gets authenticated using https git credentials using AWS CLI commands. Here is how we implemented AWS CodeCommit authentication and how it works.



Here is steps how AWS CodeCommit was securely integrated with Jenkins

Configure the AWS CLI.

```
cd ~jenkins
sudo -u jenkins aws configure
```

Accept the defaults for the AWS access key and AWS secret access key; enter us-east-1 for the region name; and enter json for the output format.

```
AWS Access Key ID [None]: Press Enter
AWS Secret Access Key [None]: Press Enter
Default region name [None]: Type us-east-1 here, and then press Enter
Default output format [None]: Type json here, and then press Enter
```

Configure Git to use IAM credentials and an HTTP path to access the repositories hosted by AWS CodeCommit.

```
sudo -u jenkins git config --global credential.helper '!aws codecommit credential-helper $@'
sudo -u jenkins git config --global credential.useHttpPath true
sudo -u jenkins git config --global user.email "me@mycompany.com"
sudo -u jenkins git config --global user.name "MyJenkinsServer"
```

Continuous Delivery

Jenkins build process from DEV to higher environment i.e. QA, UAT, PIT, SIT, Staging and Production was securely controlled using Jenkins Authorize Project plugin .This enabled projects to have their builds run with specified authorization.

Continuous Deployment

Deployments were securely authenticated using AWS access key and secret keys for deployment on AWS EC2 instances

Rollback for deployment failures

Build rollback was enabled using Jenkins Deployment plugin and configured as Parameterized rollback plan, in case build is success but functionality broken due to last code commit

Build Versions

All builds are archived and build versions are tracked for specific build version deployment and also for the roll back purpose. All build versions are controlled with parameterized build option in Jenkin Jobs

DevOps – CI / CD overall benefits

Saves Time

- Laranya InfoEdge used to run repeated builds before implementation of CI and CD, around 10-15 builds on daily basis.
- Post CI and CD implementation, repeated/deplulicated builds were prevented and this saved a lot of time
- Also CI and CD helps Laranya InfoEdge to prevent wrong code commits as it is integraated and automated for each and every build
- Continuous Integration saves time for Laranya InfoEdge on code commit, merge, pull and prevents any accidental pushes to wrong repositories and branches
- Continuous delivery also saves time for Laranya InfoEdge as developer need not spend time on manual build and log tracking and can spend time only on build error fixes.
- Continuous deployment saves time of for Laranya InfoEdge - IT Ops Manager/System Admins as there is no need of manual intervention during deployments
- CI/CD implementation enabled automated email notification so that for Laranya InfoEdge developers and IT Ops team are informed of each and every build sand deployments, Dev and Ops Team get notified on the builds and deployments stayis, this saves lot time for Dev and Ops team from manual tracking

- Faster build time when compared to manuall build and deployment
- Laranya InfoEdge requires no manual maintenance as builds and logs cleaner is automated with Jenkins workspace cleanup plugin , saving time for IT Ops team from manual disk space cleaner
- Laranya InfoEdge need not manually track builds and deployments history, automated build and deployment tracking with builds history for any future analysis
- Jenkins rollback features saves time in planning for rollback, as the functionality helps to rollback with last success build version

Productiveness

- Dev team and Ops team can now focus more on productivity as they don't have to spend much time on manual build, error tracking and deployment failures.
- They can focus more on bug fixes of build and deployments
- Better builds release management helps Laranya InfoEdge Ops team to stream line deployments on appropriate targets resources

Portability

- Builds and deployments can be portable upon build success, each and every successful builds can be deployed from one environment to another environment easily on any target environments
- Better build release control makes sure that same working version are deployed to all customer environments and this avoids unnecessary conflicts and incorrect version deployments

Version Control

- Each and every builds artefacts are now archived and tracked and can be redeployed to any target environments

Security

- We enabled controlled build process using authentication methods , AWS HTTPS Git credentials and AWS access keys and secret keys
- We enabled build authorization to prevent accidental push, builds and deployment
- Track and catch builds and deployments breaking
- Enabled Jenkins roll based authentication to make sure only authenticated and allowed users can perform a builds and deployments