Automated deployment pipeline for Mono-repo system

DISSERTATION

Submitted in partial fulfillment of the requirements of the

M. Tech. Software Engineering Degree programme

By

Veena Sri Varadharajulu
2017HS70012

Under the supervision of

Sudhanva Laxminarayan
Senior Developer

Dissertation work carried out at
SAP Labs, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA
(April 2021)
SE SAP ZG629T DISSERTATION

Automated deployment pipeline for Mono-repo system

Submitted in partial fulfillment of the requirements of the

M. Tech. Software Engineering Degree programme

By

Veena Sri Varadharajulu
2017HS70012

Under the supervision of

Sudhanva Laxminarayan
Senior Developer

Dissertation work carried out at

SAP, Bangalore

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI (RAJASTHAN)

(April 2021)

SESAP ZG629T DISSERTATION

Dissertation Title : Automated deployment pipeline for Mono-repo system

Name of Supervisor : Sudhanva Laxminarayan

Name of Student : Veena Sri Varadharajulu

ID No. of Student : 2017HS70012

Abstract

A Jenkins job will trigger the Automated deployment pipeline 4 times a day, this deployment pipeline will take the responsibility to deploy all the component to the Production SCP environment. These deployment details will be stores the RDS database in AWS, data stored in this AWS region will be used for monitoring purpose.
We will have the deployment monitoring dashboards providing information about everyday deployment along with the regions to which the applications are deployed, the deployment status, etc.,
Keywords: Jenkins, deployment, AWS RDS, Grafana

List of Symbols & Abbreviations used

AWS: Amazon Web Service
SCP: SAP Cloud Platform
RDS: Relational Database System
IST: Indian Standard Time
SLA: Service Level Agreement

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

AUTOMATED DEPLOYMENT PIPELINE FOR MONO-REPO SYSTEM

Project Objective

Automated deployment trigger pipeline will let a deployment be trigger 4 times a day starting from morning 9.30 AM IST – 9.30 PM IST with an interval of 4 hours between each deployment. This deployment pipeline will take the responsibility to deploy the new version of the components and bind required service instances , switch routes from the old version to the newly deployed version , removing service instance binding from the old components and delete these old version application.

After a successful deployment, the pipeline will run tests as a sanity check to validate if the newly deployed services are performing as expected , if not then appropriate alerting will be made.

Benefits of the project

- No manual effort will be needed to perform deployment
- When new production environment is added, this pipeline will be stable enough to make deployment to this new region
- Latest version of the components will be automatically detected and get deployed
- This pipeline will run sanity checks after each deployment to validate if the new versions of the application is functioning as performed

CHAPTER 2

PROBLEM STATEMENT

Manual deployment can be a complicated process, especially when there are lot of applications to deploy. In a growing organization where, new components get added very frequently, manual deployment is complex. The Mono-repo system for which we are building the automatic deployment will have nearly 54 components and each component will be need set of service instances to be bounded with them.

Manually binding services and then deploying the applications, along with the route switching from old version to new version, deleting the old deployed version , running sanity check after deploying all the applications from our Mono-repo system just to validate if the newly deployed version is functioning as expected is becoming a tedious task. This manual process if very time consuming and lot of resources will be needed.

Automated deployment pipeline will now help us get rid of all these complexions. We will maintain a Jenkins seed job which will act as trigger pipeline to deploy applications to our production environment 4 time a day. Also, we will integrate this pipeline with AWS RDS and store the deployment related data which will help us monitor about the deployments to different production regions and their deployment status.

So, this project will help us eliminate the manual deployment process and the gap in monitoring of the deployment becomes way easier.

CHAPTER 3

OBJECTIVE OF THE PROJECT

Automated deployment pipeline will trigger deployment to all the production environments 4 times a day starting from 9.30 AM – 9.30 PM IST, with an interval of 4 hours between each deployment. We will check if the new versions are performing as expected after each deployment, on failure we trigger appropriate notification.

This project aims to deploy new versions to the production environment and record deployment details for monitoring purpose. This will eliminate burdens in tracking the versions deployed, regions, etc.,

CHAPTER 4

UNIQUENESS OF THE PROJECT

In my current organization, there is no powerful pipeline which can perform the task of multiple deployments a day, tracking the deployed versions, etc.,
Also, the pipeline which we are building now will be dynamic enough that we define the list of apps to be deployed in each production environment.
Example: We can mention which application should be deployed in each production region.

- Production region 1 can have
  - App 1
  - App 2
  - App 3
- Production region 2 can have
  - App 1
  - App 2
  - App 4

CHAPTER 5

BENEFIT TO THE ORGANIZATION

- Delivery latest feature to the customer
- Easy roll back to previous version if needed
- New features get deployed every day, so fast delivery of features to customers
- Lesser deployment time (in comparison with the efforts need for manual deployment)

CHAPTER 6

SCOPE OF WORK

The purpose of the CICD tool is to address the need of continuous feature delivery for the cloud application, without the pressure of timelines/deadlines to the development teams. This project involves setting-up an end-to-end deployment pipeline which will make sure the services required for applications are created in the live regions .After deploying latest version of application , old components should be deleted and all the bindings made to those component must also be unbonded along with routes switching .We run tests in prod as a sanity check after each deployment.

These deployment data should be recorded in the our RDS in AWS region which will later be consumed by out Grafana monitoring dashboards.

CHAPTER 7

SOLUTION ARCHITECTURE

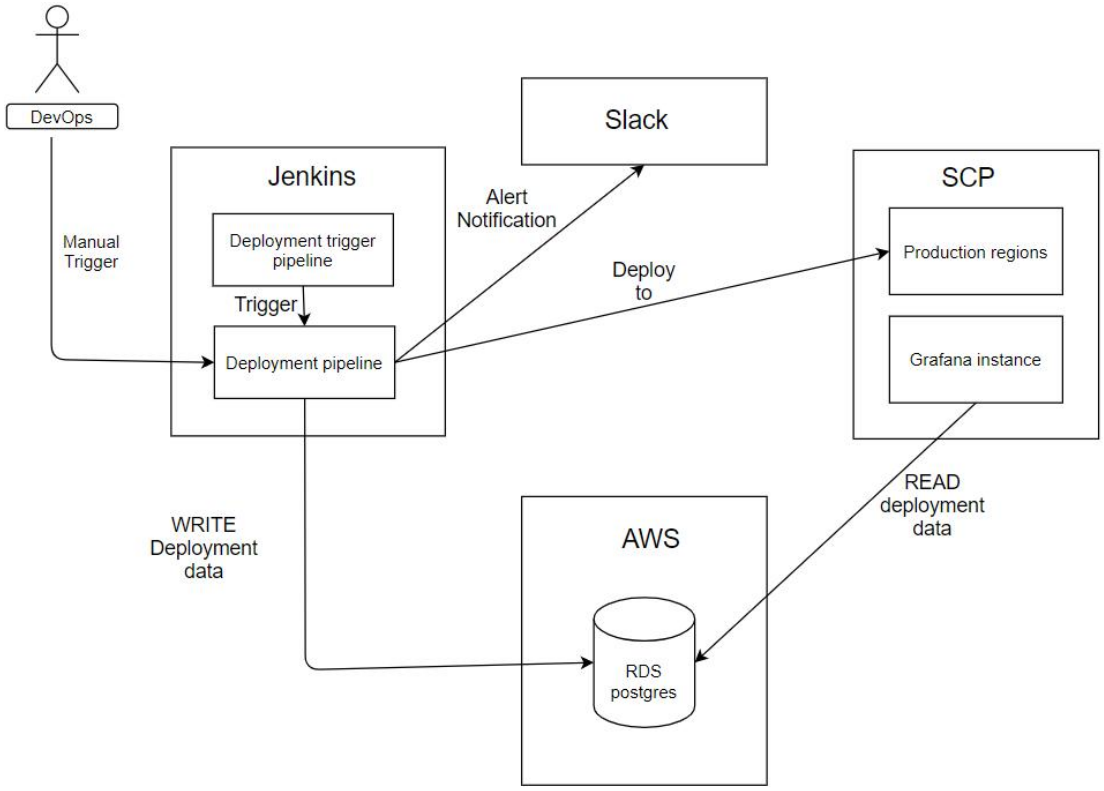

Figure 1: Business Process Flow

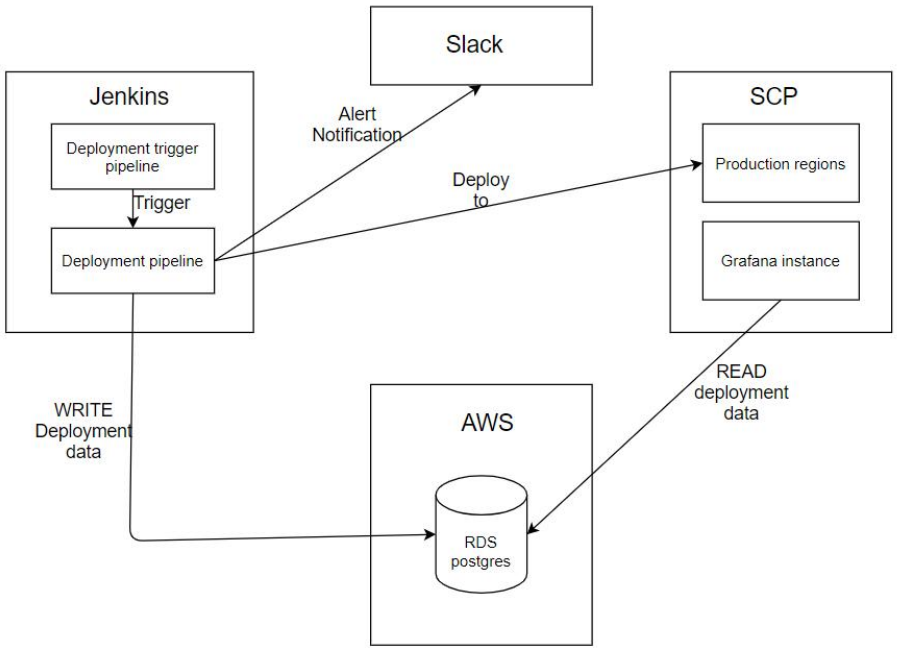

Figure 2: Jenkins integration

The above diagrams show how Jenkins is integrated with different components. Jenkins uses CF-CLI to login to the cf regions and performs the deployment actions. We use Slack webhooks to make notification in slack channel.

A webhook is a user-defined callback over HTTP. Slack webhooks can be used to notify about the deployment start, success/failure. A webhook in Jenkins is defined by the following information that you need to provide when creating a slack notification:

- Slack info message
- Production environment details
- Deployment Status
- Deployment commit version

Monitoring Dashboards will be created in the Grafana instance running in the SCP. The deployment data will be written to AWS RDS postgres and will be consumed by the dashboards in Grafana.

CHAPTER 8

WORK ACCOMPLISHED SO FAR

Jenkins Deployment trigger job

The seed job to trigger the deployment pipeline. The Bootstrap code to trigger the Deployment pipeline will be configured in the Jenkins job. This seed job is supposed to trigger the Deployment pipeline 4 times a day, this interval is set in the Jenkins job.

The deployment pipeline needs following parameters:

- **Latest deployable version**: will be picked from the GitHub
- **Deployment region**: The prod region to deploy
- **Action**: deploy/switch/test/clean
- **Alerting**: Notify about the deployment in Slack channel

Development Environment setup

CF deployment needs .yml file to deploy applications. This project supports dynamic deployment where multiple CF regions can have different applications deployed. So we maintain .yml files for different regions along with the routes, services to be bounded with the application. Service instance details and the plans will be maintained in the GitHub branch , we have tools written in NodeJS which will create the services mentioned .yml file in CF space.

Jenkins Deployment pipeline

Input: Will be supplied from the Deployment trigger job
The script for deployment is maintained in the GitHub master branch. The Jenkins job is configured to execute this script for the regions defined, the pipeline will execute the following steps:

- Create required service instances in the CF Space
- Deploy the applications
- Bind the service instance with the application
- Switch the routes for the applications
- Start the newly deployed application
- Stop the old running applications
- Sanity test to check if the newly deployed version is working as expected
- Delete the old versions of the applications from CF space

CHAPTER 9

KEY CHALLENGES

The current project depends on few pre-requisites which is something to be done manually and cannot be automated, following are few if the issue with/in capabilities in current pipeline and required manual intervention,

- The ideal approach should have retry mechanisms, on certain kind of failures, automatic retry should get enabled.

- After route switching if the application fails to start, then service will become un-operable, so if we don't revert quickly to the stable version then downtime will be long. This will affect the SLA's made.

- Deploying applications in CF will need few services to be bounded, our pipeline is capable for creating service instances in the CF given that appropriate entitlements are added to the particular CF Space. This entitlement is a manual process and cannot be automated as this needs Admin rights to increase are decrease the entitlements for a particular service.

CHAPTER 10

PLAN FOR REMINDER OF THE PROJECT

Detailed Plan of Work (for 16 weeks)

Serial Number of Task	**Tasks or subtasks to be done**	**Planned duration in weeks**	**Deliverable**	**Break-down of task**	**Status**
1.	Jenkins job to trigger the deployment pipeline in mentioned interval of time	3	Jenkins job which can trigger another pipeline for a defined interval		Completed
2.	Set-up deployment environment	3	Files with services and components required for different production environment	- Setup yaml files for deployment 1.Production regions (**1 week)** 2.Development/test regions (**1 week)** - Dynamic service creation yaml files (**1 week)**	Completed
3.	Deployment pipeline	4	Pipeline which can deploy applications and services to production regions	1.Category of Actions to performed (push/deploy/switch/clean/test) (**2 weeks)** 2.Scripts to perform the actions defined during deployment (**2 weeks)**	Completed
4	Alert Notification	1	Send deployment notifications to Slack channel		
5	Store deployment metrics	2	Writing data to the postgres database		

6	Grafana – for monitoring dashboards	1	A running Grafana instance		
7	Monitoring dashboards	2	Dashboards on Grafana showing deployment related data’s		

REFERENCES

About Jenkins: https://www.jenkins.io/

Slack webhook for sending notifications: https://api.slack.com/messaging/webhooks

Connecting to AWS RDS postgres: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToPostgreSQLInstance.html

About Grafana: https://grafana.com/docs/grafana/latest/installation/

CF-CLI: https://docs.cloudfoundry.org/cf-cli/