The PetClinic Microservices Project Story

In my recent project, I worked mainly on a micro-services application fully automated. It was a Dockerized Web Application developed in Java Spring Boot and Spring Cloud Frameworks and integrated with a MySQL database. The project was to create full microservices, CI/CD Pipelines, and deployment on a Kubernetes cluster with monitoring. Kubernetes cluster was created and managed with Rancher. Jenkins was used as the CI/CD automation tool and we created all the infrastructure on AWS EC2 Service. Also, we used Git as the version control system during the whole process. We prepared base branches namely master, dev, and release for the DevOps cycle.

The code was developed in Java and Maven was used as the build tool. So I used Maven Wrapper for the testing, packaging, and installing phases. I spun up the development server through terraform. I also prepared Dockerfiles for each microservice and helm charts to deploy the application to the Kubernetes Cluster.

Firstly, we test the application on a local environment using docker-compose. Then we deployed the applications to the Kubernetes Cluster.

To create Kubernetes manifest files we used some tools like Kompose. We converted the Docker-compose files to Kubernetes definition files by using the Kompose tool. We also used the Helm tool to automate the creation, packaging, configuration, and deployment of applications and services.

Our app was running on AWS, we used Terraform as Infrastructure as a Code (IaaS) to automate the creation of infrastructure in each pipeline. We used Ansible to configure the instances and prepared some playbooks for this purpose. Since we used the AWS Cloud platform, I prepared a dynamic inventory that includes EC2 instances into the inventory by checking their tags. In addition, to be able to connect Ansible to our EC2 instances, I wrote an Ansible config file within the Jenkins pipeline.

As DevOps engineers, we prepare development servers for developers. I set up Jenkins Server and installed some plugins such as Docker Pipeline, GitHub Integration, and Jacoco. My responsibilities were:

1. CI/CD Pipeline. I was responsible for writing CI/CD pipeline scripts for each stage and keeping them up and running. I've created the nightly and weekly pipelines by using Jenkins. I was responsible for creating Dockerfiles and building images.
2. Our Project was running on AWS, so I was in charge of configuring all services that we used in AWS.
3. I'm also responsible for writing the Kubernetes manifest files and Helm charts. I’ve created the Helm charts for the staging and production environment.

I want to mention the steps in the nightly build of the project one by one,

1. First, we created AWS ECR Repo to store images using AWS CLI.
2. Secondly, we packaged the app into jars with Maven Wrapper.
3. Then we prepared image tags for Docker Images.
4. Next, we built an App Docker Images.

5. We prepared helm charts for the application and pushed AWS S3 Service.
6. After that, we pushed the images to the ECR Repo using AWS CLI.
7. And then we created a key pair for Ansible and prepared ansible.config file and inventory files.
8. Following this, we created a QA automation infrastructure by using Terraform.
9. Then, we created a Kubernetes cluster for QA automation build using Ansible.
10. Next, we deployed the App on the Kubernetes cluster using the Helm charts.
11. After that, we run Functional Tests on QA Environment using the Ansible playbook file. We automated functional tests with Selenium.
12. Finally, we set the pipeline to delete all local images, the repository, and the infrastructure automatically.
13. The next morning, we check the Jenkins logs to see if there is any error or failed job and then check the Grafana dashboards to see the performance or any metric fails.

I want to mention the steps in the weekly build of the project one by one,

1. First of all, we created a Kubernetes cluster on AWS EKS service for manual QA tests using eksctl.
2. we have created AWS ECR Repo to store images using AWS CLI
3. Every week on Sunday, we packaged the app into jars with Maven Wrapper.
4. Then we prepared image tags for Docker Images
5. Next, we built App Docker Images
6. After that, we pushed the images to the ECR Repo using AWS CLI.
7. We prepare helm charts for the application and push AWS S3 Service.
8. Next, we deployed the App on the Kubernetes cluster using the Helm charts.

I want to talk about the Rancher - Kubernetes cluster orchestration tool- used in our project:

I used Rancher to create and manage our Kubernetes clusters. To install the Rancher, I used the Helm chart. With Rancher, we easily made changes in the cluster via its dashboard, add nodes, delete nodes, edit configuration files, and used kubectl on its terminal.

As monitoring tools: we monitored the application in the cluster with Prometheus and Grafana.