**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 Springboot and Spring Cloud Frameworks and integrated with 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 App Docker Images.
5. We prepare helm charts for the application and push 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 day 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 applications in the cluster with Prometheus and Grafana.

Petclinic Mikrohizmetler Projesi Hikayesi

Son proje deneyimimde, tamamen otomasyonlu bir mikrohizmetler uygulaması üzerinde çalıştım. Bu, Java Spring Boot ve Spring Cloud Frameworks kullanılarak geliştirilmiş, Dockerize edilmiş bir Web Uygulamasıydı ve MySQL veritabanıyla entegre edilmişti. Projenin amacı, tam mikrohizmetler oluşturmak, CI/CD Boruları kurmak ve izleme ile bir Kubernetes kümesine dağıtmaktı. Kubernetes kümesi, Rancher ile oluşturulup yönetiliyordu. CI/CD otomasyon aracı olarak Jenkins kullanıldı ve tüm altyapıyı AWS EC2 Hizmeti üzerine oluşturduk. Ayrıca, süreç boyunca sürüm kontrol sistemi olarak Git'i kullandık. Ana dallar olan ana (master), geliştirme (dev) ve yayın (release) dallarını DevOps döngüsü için hazırladık.

Kod Java dilinde geliştirildi ve derleme aracı olarak Maven kullanıldı. Test, paketleme ve kurulum aşamaları için Maven Wrapper'ı kullandım. Geliştirme sunucusunu terraform ile hızla ayağa kaldırdım. Ayrıca, her mikrohizmet için Docker dosyaları ve uygulamayı Kubernetes Kümesine dağıtmak için helm şablonları hazırladım. İlk olarak, uygulamayı docker-compose kullanarak yerel ortamda test ettik. Ardından uygulamaları Kubernetes Kümesine dağıttık. Kubernetes belirtim dosyaları oluşturmak için Kompose gibi araçlar kullandık. Docker-compose dosyalarını Kubernetes tanım dosyalarına Kompose aracını kullanarak çevirdik. Ayrıca, uygulamaların ve hizmetlerin oluşturulması, paketlenmesi, yapılandırılması ve dağıtılmasını otomatikleştirmek için Helm aracını kullandık.

Uygulamamız AWS üzerinde çalışıyordu, bu nedenle altyapı oluşturmayı otomatikleştirmek için Altyapı olarak Kod (IaaS) olarak Terraform kullandık. Örneklerin yapılandırılması için Ansible kullanıldı ve bu amaçla bazı playbook'lar hazırladık. AWS Cloud platformunu kullandığımızdan, etiketlerini kontrol ederek EC2 örneklerini envanterimize dahil eden dinamik bir envanter hazırladım. Ayrıca, Ansible'ı EC2 örneklerimize bağlayabilmek için bir Ansible yapılandırma dosyası yazdım ve bunu Jenkins boru hattı içine ekledim.

DevOps mühendisleri olarak geliştiriciler için geliştirme sunucuları hazırlarız. Jenkins Sunucusunu kurup Docker Pipeline, GitHub Entegrasyonu ve Jacoco gibi bazı eklentileri ekledim. Sorumluluklarım şunlardı:

1. CI/CD Borusu: Her aşama için CI/CD boru hattı komut dosyalarını yazmak ve bunları çalışır durumda tutmak. Jenkins'i kullanarak gece ve haftalık boru hatları oluşturdum. Docker dosyalarını oluşturup görüntülerin yapımını üstlendim.
2. Projemiz AWS üzerinde çalışıyordu, bu yüzden AWS'de kullandığımız tüm hizmetleri yapılandırmaktan sorumluydum.
3. Ayrıca, Kubernetes belirtim dosyalarını ve Helm şablonlarını yazmak da benim görevimdi. Staging ve üretim ortamı için Helm şablonları hazırladım.

Projenin gece yapısına ait adımları sırasıyla açıklamak istiyorum:

1. İlk olarak, görüntüleri depolamak için AWS ECR Repo oluşturuldu, bunu AWS CLI kullanarak gerçekleştirdik.
2. İkinci olarak, Maven Wrapper ile uygulamayı jar dosyalarına paketledik.
3. Ardından, Docker Görüntüleri için etiketleri hazırladık.
4. Sonrasında, Uygulama Docker Görüntüleri oluşturduk.
5. Uygulama için helm şablonlarını hazırladık ve AWS S3 hizmetine yükledik.
6. Bundan sonra, Görüntüleri AWS CLI kullanarak ECR Repo'ya yükledik.
7. Sonrasında, Ansible için bir anahtar çifti oluşturduk ve ansible.config dosyasını ile envanter dosyalarını hazırladık.
8. Bu adımdan sonra, Terraform kullanarak QA otomasyon altyapısını oluşturduk.
9. Ardından, Ansible kullanarak QA otomasyon oluşturmak için bir Kubernetes kümesi oluşturduk.
10. Daha sonra, uygulamayı Kubernetes kümesine Helm şablonları ile dağıttık.
11. Sonrasında, Selenium ile fonksiyonel testleri otomasyonla çalıştırdık.
12. En son olarak, boruyu tüm yerel görüntüleri, deposunu ve altyapıyı otomatik olarak silmek üzere ayarladık.
13. Ertesi sabah, Jenkins günlüklerini kontrol ederiz, herhangi bir hata veya başarısız iş olup olmadığını kontrol ederiz. Ayrıca, performans veya metrik hataları için Grafana panolarını kontrol ederiz.

Haftalık yapıya ait adımları da sırasıyla açıklamak istiyorum:

1. İlk olarak, AWS EKS hizmeti üzerinde manuel QA testleri için bir Kubernetes kümesi oluşturduk.
2. AWS CLI kullanarak görüntüleri depolamak için AWS ECR Repo oluşturduk.
3. Her hafta Pazar günü, Maven Wrapper ile uygulamayı jar dosyalarına paketledik.
4. Ardından, Docker Görüntüleri için etiketleri hazırladık.
5. Sonrasında, Uygulama Docker Görüntüleri oluşturduk.
6. Bundan sonra, Görüntüleri AWS CLI kullanarak ECR Repo'ya yükledik.
7. Uygulama için helm şablonlarını hazırladık ve AWS S3 hizmetine yükledik.
8. Daha sonra, uygulamayı Kubernetes kümesine Helm şablonları ile dağıttık.

Projemizde kullandığımız Rancher -Kubernetes küme yönetim aracı- hakkında konuşmak istiyorum: Rancher'ı Kubernetes kümesi oluşturmak ve yönetmek için kullandık. Rancher'ı kurmak için Helm şablonunu kullandım. Rancher ile kümede kolayca değişiklikler yapabiliyor, görsel arayüzü üzerinden düğme ekleyebilir, düğme silebilir, yapılandırma dosyalarını düzenleyebilir ve terminalinde kubectl kullanabilirsiniz. İzleme araçları olarak, Prometheus ve Grafana ile kümedeki uygulamaları izledik.