**Project : Automated Flask Application Deployment to Amazon ECR and EC2 Using Jenkins Pipeline**

**Obejctive :**

The objective of this project is to **automate the end-to-end deployment** of a Flask web application using a **Jenkins CI/CD pipeline**, ensuring:

* **Automatic Docker image creation** from application code.
* **Seamless image push to Amazon ECR** (Elastic Container Registry).
* **Remote deployment to an EC2 instance** using SSH and Docker.
* **Repeatable and consistent deployments** with minimal manual intervention.
* **Streamlined DevOps workflow** for faster and more reliable application delivery.

This project showcases best practices for containerization, AWS integration, and CI/CD automation.

**Introduction :**

In today's cloud-native ecosystem, deploying applications manually is time-consuming and error-prone. To address this, **CI/CD (Continuous Integration and Continuous Deployment)** pipelines are used to automate the process of building, testing, and deploying applications.

This project demonstrates how to **automatically deploy a Flask-based Python web application** using a Jenkins pipeline. The application is containerized using Docker, stored securely in **Amazon Elastic Container Registry (ECR)**, and deployed on a remote **Amazon EC2** instance.

By integrating **Jenkins, Docker, AWS ECR, and EC2**, this project enables teams to:

* Rapidly deploy code changes,
* Maintain a consistent deployment process, and
* Reduce human errors in production environments.

Whether you're a DevOps engineer or a developer exploring automation, this project provides a **hands-on example of modern CI/CD practices** in action.

**Techonology Stack :**

|  |  |
| --- | --- |
| Jenkins | Automation server used for building, testing, and deploying the application. |
| Docker | Containerization platform used to package the Flask application. |
| Amazon ECR | AWS-managed Docker container registry used to store and manage Docker images. |
| GitHub | Source code repository integrated with Jenkins for version control. |
| AWS CLI | Command-line interface to interact with AWS services within Jenkins and EC2. |

**Implementation Steps :**

### **Step 1: Launch EC2 Instance**

* Launch an EC2 instance (Amazon Linux 2 or Ubuntu).
* Allow these ports in the **security group**:
  + Port 22 (SSH)
  + Port 8080 (Jenkins)
  + Port 80 (for application, if needed)
* Connect via SSH:

- ssh -i <your-key.pem> ec2-user@<EC2-PUBLIC-IP>

**Step 2: Install Docker**

* **sudo yum update -y**
* **sudo amazon-linux-extras install docker -y**
* **sudo service docker start**
* **sudo usermod -aG docker ec2-user**

**Step 3: Run Jenkins in Docker**

* **docker run -d --name jenkins -p 8080:8080 -p 50000:50000 -v jenkins\_home:/var/jenkins\_home -v /var/run/docker.sock:/var/run/docker.sock jenkins/jenkins:lts**

### **Step 4: Access Jenkins**

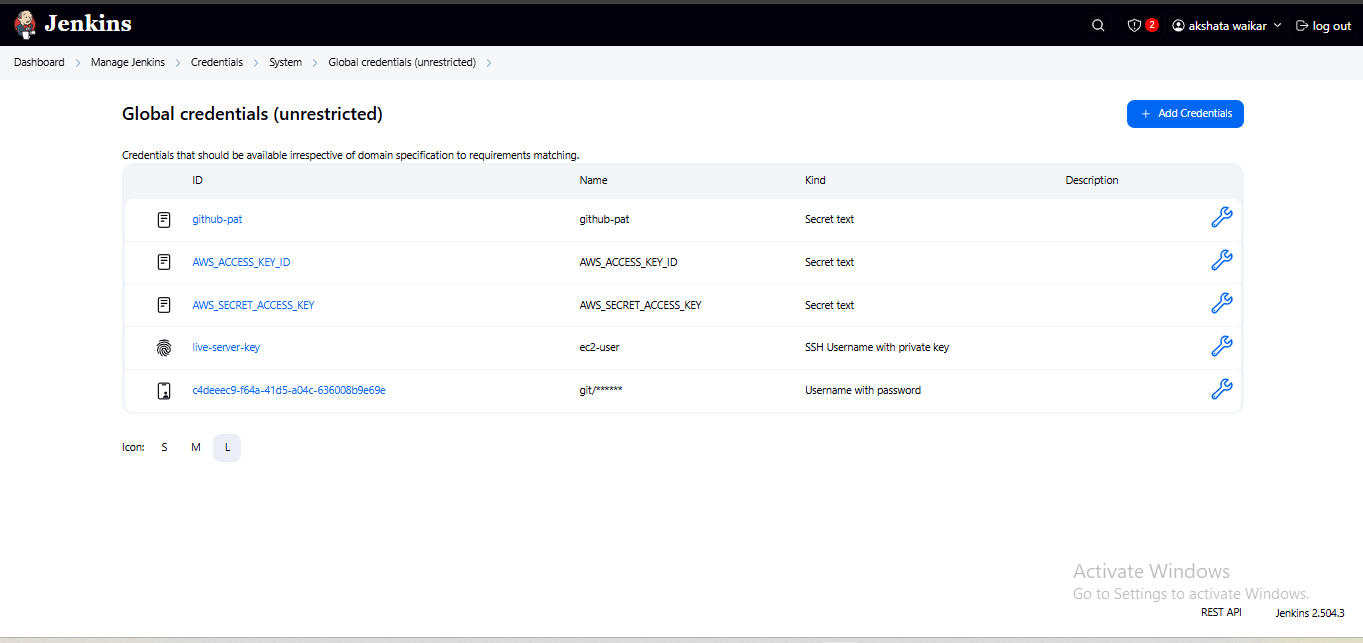
* Open browser → http://<EC2\_PUBLIC\_IP>:8080
* Unlock Jenkins:
* docker exec -it jenkins cat /var/jenkins\_home/secrets/initialAdminPassword

### **Step 5: Install Plugins & Setup Admin**

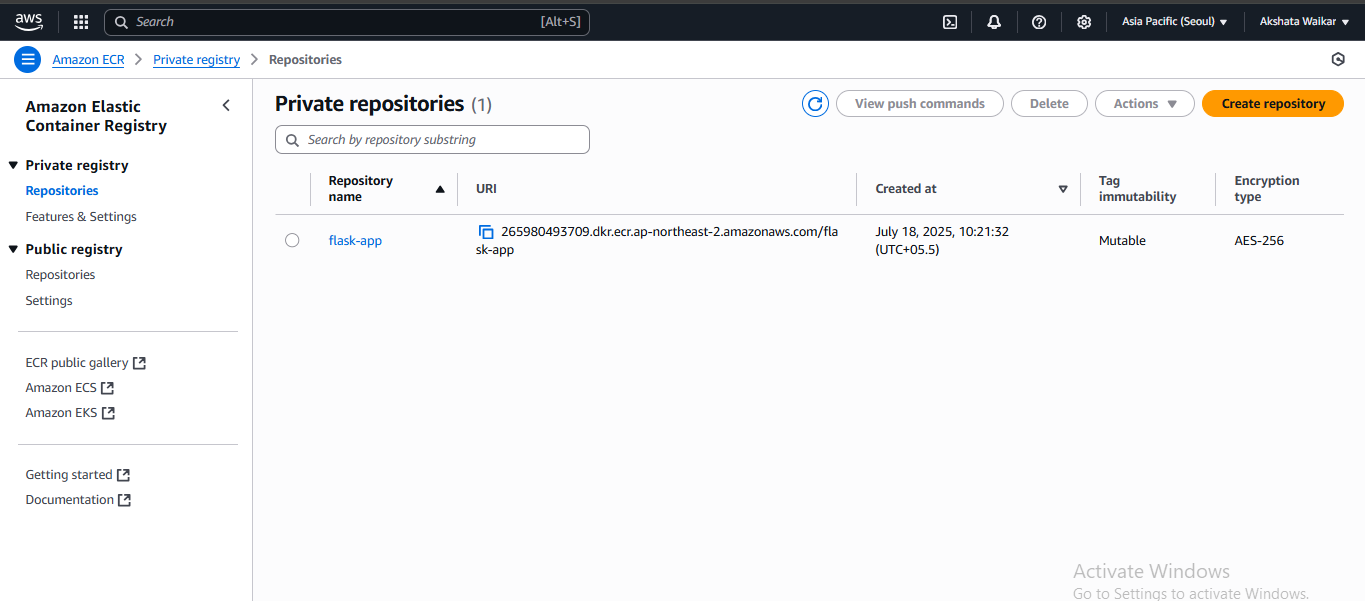
* Install **Suggested Plugins**
* Create **admin user**
* Jenkins is now ready to use

#### ****Step 6: Configure AWS Credentials in Jenkins****

* Go to **Manage Jenkins → Credentials → System → Global credentials**.
* Add the following:
  + **AWS\_ACCESS\_KEY\_ID** (as secret text)
  + **AWS\_SECRET\_ACCESS\_KEY** (as secret text)
  + **GitHub Personal Access Token**
  + **EC2 SSH Private Key** (as SSH Username with private key)



**Step 7: Create ECR Repository**



#### ****Step 8: Configure Jenkins Pipeline****

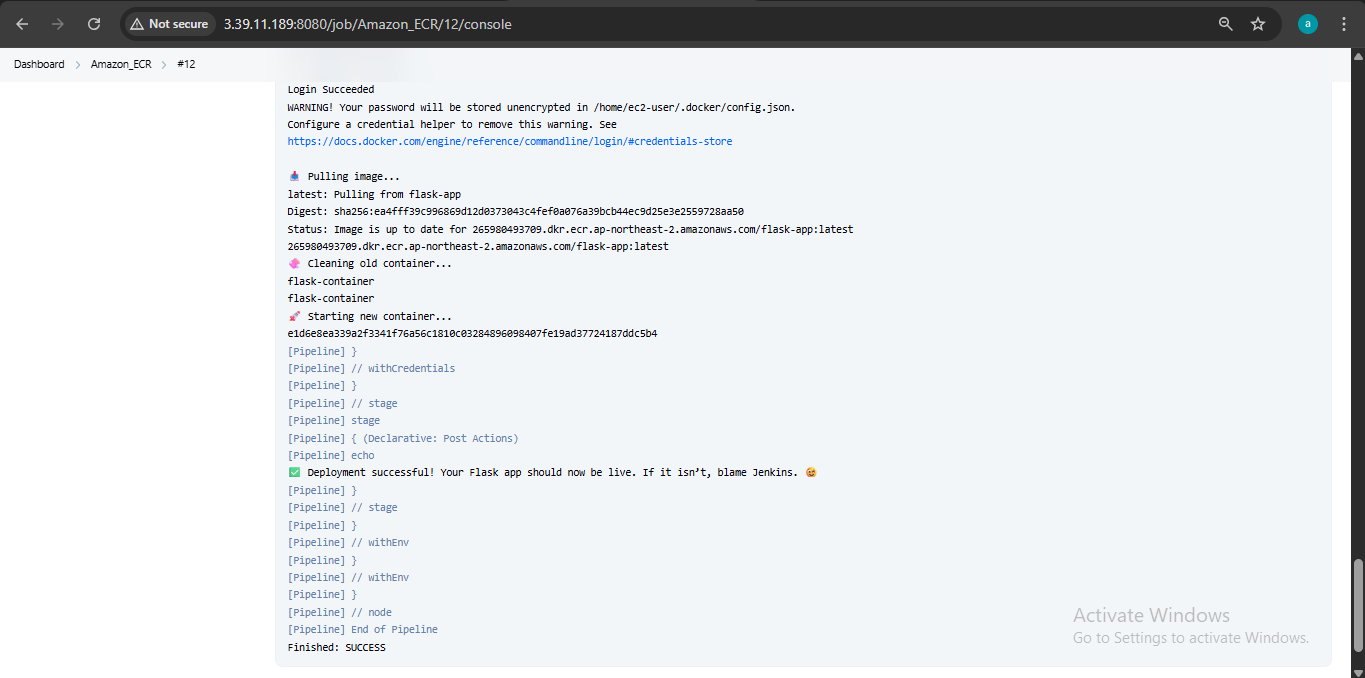
* Use a **Declarative Pipeline** (Jenkinsfile) with stages:
  + **Clone Repo**
  + **Build Docker Image**
  + **Login to ECR**
  + **Push Image to ECR**
  + **Deploy to EC2 via SSH**

#### ****Step 9: Prepare EC2 Live Server (Target)****

* Install Docker and AWS CLI.
* Allow port **80** in EC2 security group.
* Authorize Jenkins IP or allow SSH from anywhere for testing.

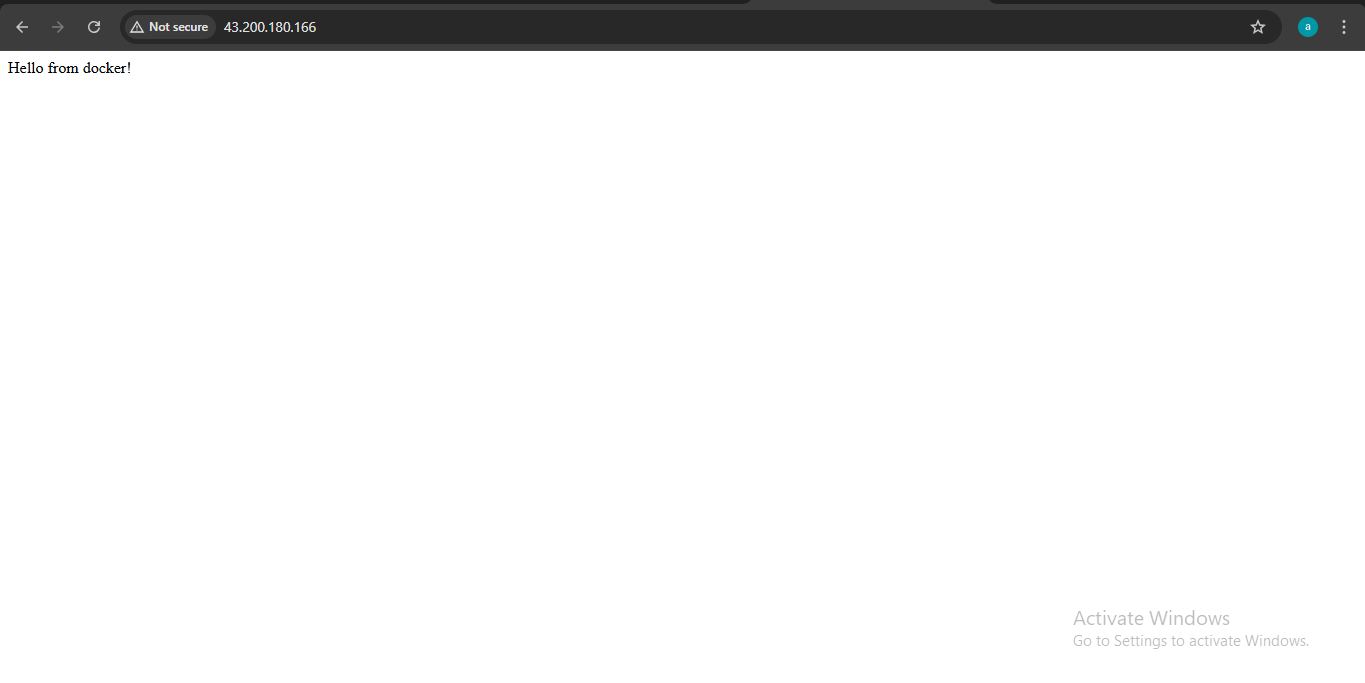
#### ****Step 10: Run Jenkins Pipeline****

* Trigger build from Jenkins dashboard.
* Jenkins will:
  + Build the Docker image.
  + Push it to Amazon ECR.
  + Connect to EC2 via SSH.
  + Pull and run the Docker container on EC2.



#### ****Step 11: Verify Deployment****

* Open a browser and visit:  
  http://<EC2\_PUBLIC\_IP>  
  You should see: **"Hello from Docker!"**



**Conclusion :**

In this project, we successfully implemented a **CI/CD pipeline using Jenkins inside a Docker container on an EC2 instance**, integrated with **Amazon ECR** and deployed a **Flask application** to another **EC2 instance**.

By the end of this setup:

* Jenkins (running in Docker) builds the Docker image for the Flask app.
* The image is pushed to **Amazon ECR (Elastic Container Registry)**.
* Jenkins connects securely to a **live EC2 instance** using SSH credentials and deploys the latest image by pulling from ECR.
* The old container is removed, and the new one is started automatically.

This setup ensures:

* **Automated deployments** on every push or build.
* **Seamless integration** between Jenkins, Docker, ECR, and EC2.
* A strong foundation for building more complex pipelines (Blue/Green, Canary, multi-environment).

**GitHub Repository link** : https://github.com/Akshata-Waikar/Amazon\_ECR

**Prepared By:**  
**Akshata Waikar**  
Cloud & DevOps Intern  
Cravita Technologies