# DevOps Coding Assessment Submission

Candidate Name: [Your Name]

Submission Date: [Insert Date]

Position: DevOps Engineer

Assessment By: Makerble

Submission Link: https://forms.gle/FmYFZgpyw3VSrn7t6

## Step 1 – Docker Setup

Objective:  
Create a Dockerfile for deploying a Ruby on Rails application using a separate PostgreSQL container.

Solution Summary:  
- Created a Dockerfile to build the Rails application image.  
- Used docker-compose.yml to orchestrate the Rails app and PostgreSQL as separate containers.  
- The database.yml file configured PostgreSQL settings for container communication.

Key Files:  
- Dockerfile  
- docker-compose.yml  
- config/database.yml

Verification:  
Rails app and PostgreSQL communicate over the bridge network; app launches successfully via docker-compose up.

## Step 2 – Kubernetes Deployment

Objective:  
Deploy the application on Kubernetes using:  
- Deployment for the Rails app.  
- StatefulSet for PostgreSQL.  
- Ingress for external access.

Solution Summary:  
- Used Minikube as the Kubernetes environment.  
- Created rails-deployment.yaml, postgres-statefulset.yaml, rails-service.yaml, and postgres-service.yaml.  
- Deployed ingress using ingress-nginx as per official documentation.  
- Applied persistent storage for PostgreSQL using PVC.

Key Files:  
- rails-deployment.yaml  
- postgres-statefulset.yaml  
- services.yaml  
- ingress.yaml

Verification:  
Application was successfully deployed and accessible via configured Ingress route on Minikube.

## Step 3 – ArgoCD GitOps

Objective:  
Manage application deployment using ArgoCD and GitOps principles.

Solution Summary:  
- Installed ArgoCD on Minikube using YAMLs from the official repo.  
- Created a private GitHub repository to host all manifests.  
- Added ArgoCD Application definition in application.yaml.  
- Configured ArgoCD to access the private repo via Git credentials stored in a Kubernetes Secret.  
- Used argocd-cm.yaml and argocd-rbac-cm.yaml for additional configuration.

Key Files:  
- application.yaml  
- argocd-cm.yaml  
- argocd-rbac-cm.yaml  
- Kubernetes manifests for Rails and PostgreSQL  
- Repository config file for GitOps

Verification:  
Successful sync of manifests from the GitHub repo to the cluster using ArgoCD Web UI.

## Step 4 – Tekton Pipelines

Objective:  
Use Tekton Pipelines to automate building and pushing the Docker image.

Solution Summary:  
- Installed Tekton Pipelines and Tekton Dashboard.  
- Created Tasks using kaniko to build and push Docker images.  
- Designed a pipeline that:  
 1. Clones the public fork of the sample Rails app.  
 2. Builds the Docker image using Kaniko.  
 3. Pushes to Docker Hub.

Key Files:  
- pipeline.yaml  
- task-clone.yaml  
- task-kaniko.yaml  
- pvc.yaml (for Tekton workspace)

Verification:  
Manually executed the pipeline from the Tekton dashboard. Docker image was successfully pushed to Docker Hub.

## Files for Submission

Included in ZIP:  
- All Kubernetes Manifests  
- Dockerfile and Compose setup  
- ArgoCD configuration files  
- Tekton pipeline files  
- README.md with setup instructions

Not Included:  
- Deployment keys or any sensitive secrets

## Demo Video

A recorded walkthrough of:  
- Application deployment on Docker & Kubernetes  
- ArgoCD GitOps workflow  
- Tekton pipeline execution

## Additional Notes

- Ruby on Rails version used: [Specify if applicable]  
- Docker Hub Repository: [your-dockerhub-username]/rails-app  
- GitHub Repo (private): [Provide link, add to README]  
- Sample Rails App Source: [Mention original repo or fork]