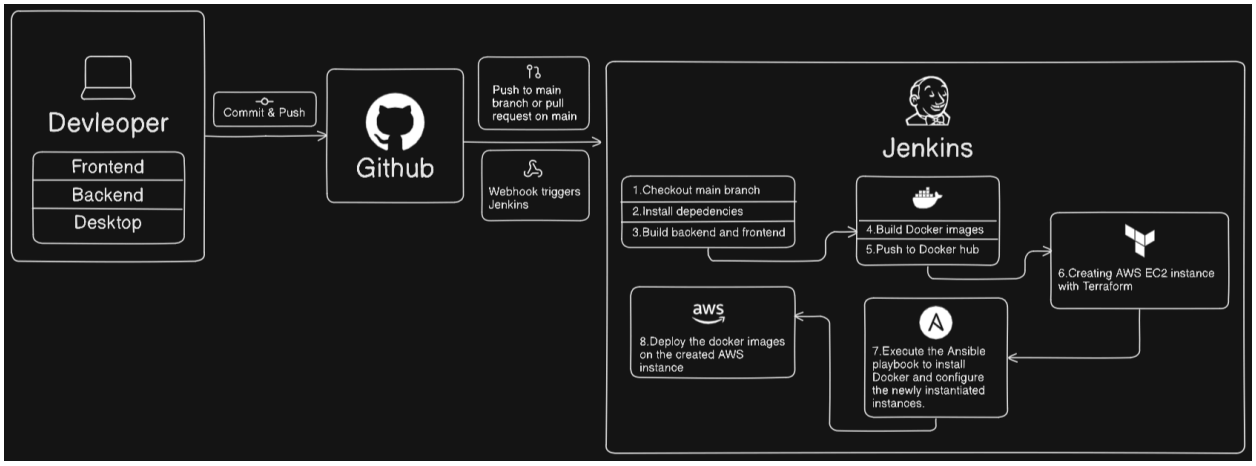
EC5207

ASSIGNMENT-CI/CD DIAGRAM

NANDASIRI R.J.C.
EG/2022/5203

Part 1: CI/CD Design Diagram



This CI/CD architecture establishes a fully automated pipeline that synchronizes application development with cloud-native deployment on AWS. The workflow is initiated by the developer committing and pushing frontend, backend, and desktop code to a GitHub repository. This action, specifically a push to the main branch or a pull request, triggers a Jenkins automation server via webhooks to begin the orchestration process. Within Jenkins, the pipeline follows a structured lifecycle.It first checks out the code from main branch, installs necessary dependencies, and builds both the backend and frontend components. These builds are then containerized into Docker images and pushed to Docker Hub. Simultaneously, the pipeline uses Terraform to provision the underlying infrastructure, specifically creating AWS EC2 instances. Once the hardware is ready, Ansible playbooks are executed to install Docker and configure the environment on the newly instantiated instances. The workflow concludes by automatically deploying the Docker images to the configured AWS infrastructure, enabling a fast and repeatable delivery process.

Part 2: Automation approach

DevOps Tools and Purpose

The following tools were selected to create a scalable and reproducible deployment environment:

- GitHub: Serves as the primary version control system to manage application source code and infrastructure scripts.
- Jenkins: Acts as the central CI/CD orchestrator, automating the build, test, and deployment phases through a declarative pipeline.
- Terraform: Used as the Infrastructure as Code (IaC) tool to programmatically provision AWS EC2 instance and networking components.
- Ansible: Used for configuration management to install Docker and prepare the runtime environment on newly created instances.
- Docker: Provides containerization.

Application Tools and Dependencies

The application stack relies on the following core technologies:

- Node.js & npm: Used for the backend (NestJS) and frontend (Next.js) build processes.
- PostgreSQL: The primary database for data persistence.
- C# & Avalonia UI: A cross-platform UI framework used to build the desktop application interface.
- Prisma ORM: Used for database schema management and type-safe queries.

Automated Deployment Workflow

The deployment process is fully automated through the following pipeline stages:

1. Source Checkout: Jenkins identifies a push to the main branch and pulls the latest code from GitHub.
2. Build Phase: The pipeline installs dependencies and builds the application components using npm.
3. Containerization: Docker images are created for the frontend and backend, then pushed to Docker Hub.
4. Infrastructure Provisioning: Terraform scripts run to ensure the AWS EC2 instances are active and correctly configured.
5. Environment Configuration: Ansible playbooks execute to install the Docker engine and necessary security updates on the instances.
6. Final Deployment: The latest Docker images are pulled onto the AWS instances and started, completing the delivery cycle.