Design and Development of AI Agent-Based Travel Planner Platform using Agile Scrum and DevOps Integration

by

Abdur Rahman
ID: CSE2203027171

Farhana Yesmain
ID: CSE2203027175

Anika Tabassum Sinjim
ID: CSE2203027130

Mia Md. Siam
ID: CSE2203027129

Naima Islam Keya
ID: CSE2203027153

Supervised by
Prof. Bulbul Ahamed

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)
September 2026

APPROVAL

The project titled "**Design and Development of AI Agent-Based Travel Planner Platform using Agile Scrum and DevOps Integration**" submitted by Abdur Rahman (CSE2203027171), Farhana Yesmain (CSE2203027175), Anika Tabassum Sinjim (CSE2203027130), Mia Md. Siam (CSE2203027129) and Naima Islam Keya (CSE2203027153) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Bulbul Ahamed — **Supervisor**
Professor and Head,
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 1**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 2**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 3**
Department of Computer Science and Engineering
Sonargaon University (SU)

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Bulbul Ahamed, Professor** and **Head,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Bulbul Ahamed)
Supervisor

Signature

Abdur Rahman
ID: CSE2203027171

Farhana Yesmain
ID: CSE2203027175

Anika Tabassum Sinjim
ID: CSE2203027130

Mia Md. Siam
ID: CSE2203027129

Naima Islam Keya
ID: CSE2203027153

ABSTRACT

Planning a trip often turns into a tedious chore as travelers are forced to jump between disjointed platforms to organize itineraries, evaluate accommodations, and execute bookings. This project resolves that complexity by delivering a cloud-native, end-to-end travel platform powered by autonomous AI agents. Unlike traditional engines that merely provide static recommendation lists, this system uses intelligent agents to interpret individual user preferences, dynamically generate personalized travel itineraries, and autonomously execute complex booking workflows. The frontend is built with Next.js to ensure a responsive, seamless user experience, while Supabase serves as the backend infrastructure, providing flexible database management and vector search capabilities to power high-dimensional context retrievals for the AI models. To handle the operational demands of a multi-service architecture, the project was managed using the Agile Scrum framework, enabling continuous iterative development and feature delivery. Robust DevOps practices were integrated from the ground up to establish an automated CI/CD pipeline using Jenkins, streamline application containerization via Docker, and orchestrate scalable container deployments across Amazon Elastic Kubernetes Service (AWS EKS). Managed via Infrastructure as Code, this automated pipeline guarantees consistent testing, rapid deployment, and high system availability under varying workloads. Ultimately, this capstone project demonstrates how synthesizing agentic AI architectures with modern cloud-native engineering principles can transform a fragmented, overwhelming workflow into a cohesive, intelligent, and effortless travel planning solution.

ACKNOWLEDGMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the scheduled time.

We are auspicious that we had the kind association as well as supervision of **Bulbul Ahamed**, Professor and Head, Department of Computer Science and Engineering, Sonargaon University whose hearted and valuable support with best concern and direction acted as necessary recourse to carry out our project.

We are also thankful to all our teachers during our whole education, for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
BaaS	Backend as a Service
CI/CD	Continuous Integration / Continuous Deployment
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
DevOps	Development and Operations
EKS	Elastic Kubernetes Service
GraphQL	Graph Query Language
HTML	HyperText Markup Language
MVPs	Minimum Viable Products
RAG	Retrieval-Augmented Generation
RESTful	Representational State Transfer
RLS	Row Level Security
SDLC	Software Development Life Cycle
UAT	User Acceptance Testing
UI/UX	User Interface / User Experience
VPC	Virtual Private Cloud

TABLE OF CONTENTS

Title			Page No.
DECLARATION …….……..			iii
ABSTRACT ……...…….			iv
ACKNOWLEDGEMENT ……..……			v
LIST OF ABBREVIATIONS ……..……..			vi
CHAPTER 1			1 – 2
INTRODUCTION TO AI AGENT-BASED TRAVEL PLANNER PLATFORM USING AGILE SCRUM AND DEVOPS INTEGRATION			
1.1	Introduction ……..…….....		1
1.2	Objectives ……...……..		1 – 2
1.3	Technology used ……..…….		2
1.4	Expected benefit ……..……..		3
CHAPTER 2			3 – 8
INITIALIZE THE SOFTWARE DEVELOPMENT LIFE CYCLE			
2.1	Introduction …….…….		3
2.2	Agile Scrum …….…….		3 – 4
2.3	Outline of the Scrum Structure ……		4
2.4	Scrum Team Setup …….....		5
2.5	Sprint Planning …….		5 – 6
2.6	Sprint Backlog ……..		6 – 8
	2.6.1	Sprint 1 — Discovery & Design ……	6
	2.6.2	Sprint 2 — Core Frontend ……..	6 – 7
	2.6.3	Sprint 3 — Backend & Data ……...	7
	2.6.4	Sprint 4 — AI Assistant ……..	7 – 8
	2.6.5	Sprint 5 — Polish & Release Readiness ……	8
2.7	Scrum Overview of this Project ……..		8

CHAPTER 3 9 – 29

CREATE THE DEVOPS STEPS AND AUTOMATED THE CI/CD PIPELINE

3.1 Introduction …….. 9

3.2 Build Infrastructure with Terraform for AWS ……. 10 – 13

3.2.1 Setup the Security Group for EKS ……. 11

3.2.2 Setup the EKS ……..……... 11 – 13

3.3 Managing Server with Ansible ……..……... 14 – 19

3.3.1 Inventory file …….. 14

3.3.2 Playbook: Install Kubectl and AWS CLI …… 14 – 15

3.3.3 Playbook: Install Java and Docker ……..……. 15 – 16

3.3.4 Playbook: Install Prometheus and Grafana on AWS EC2 ….. 16 – 18

3.3.5 Configure AWS Credentials on Managed Nodes ……... 18 – 19

3.3.6 Run the Ansible files ……..……. 19

3.4 Create a Github Repository ……..……..……... 20 – 21

3.5 Create a Docker File ……..……..…….. 21 – 22

3.6 Create Jenkins CI/CD Pipeline ……..…… 22 – 27

3.6.1 Creating Jenkins Credential ……..……... 23

3.6.2 Creating Nodes in Jenkins ……..……. 24 – 25

3.6.3 Creating a Pipeline According to this Node …….. 25 – 27

3.6.4 Run the Pipeline ……..…… 27

3.7 Create Kubernetes files in the Project Repository …… 27 – 29

3.7.1 Configuration (ConfigMap) ……..……... 28

3.7.2 Secrets (Secret) ……..…….. 28

3.7.3 Application Life Cycle (Deployment) ……..…... 28 – 29

3.7.4 Networking (Service) ……..…… 29

CHAPTER 4 30 – 39

INTEGRATING THE SUPABASE SERVER AND BUILD THE NECESSARY LOGIC FOR THE APPLICATION

4.1 The Purpose of using Supabase ……..……. 30 – 31

4.2 Create an AWS EC2 for Supabase Server …….. 31 – 32
4.3 Allocated an Elastic IP ……..……... 33
4.4 Install Docker and Supabase throw the Docker Image in EC2 Server 33 – 35
4.5 Setup the .env file and Run the Server …….……... 35 – 36
4.6 Supabase(PostgreSQL) Query …….…… 36 – 39
4.6.1 Extension & Setup ……..…… 36
4.6.2 Profiles Table & Auth Integration …….. 36
4.6.3 Categories & Tours …….…… 37
4.6.4 Bookings Engine …….…… 37
4.6.5 Payments Processing …….…….. 37
4.6.6 Reviews & Ratings System …….…… 37
4.6.7 AI Chat Engine (chat_conversations & chat_messages) …… 37 – 38
4.6.8 Database Performance Indexes …….……. 38
4.6.9 Seed Data Initialization …….……. 38 – 39

CHAPTER 5 40 – 44
DESIGN THE UI/UX AND DEVELOP FRONTEND SERVER WITH NEXTJS
5.1 Design UI/UX and make a Design file ……. 40
5.2 Initialize the Nextjs App ……. 41 – 42
5.3 Sitemap for the Application ……... 42
5.4 Authentication Module …….…… 42
5.5 Tour Browsing and Details Module …….…… 43
5.6 Booking and Checkout Module …….…….. 43
5.7 AI Assistant Module …….…… 43 – 44
5.8 Security Implementation …….……. 44

CHAPTER 6 45 – 47
TESTING AND MONITORING
6.1 Testing Approach …….. 45
6.2 Monitoring with Prometheus and Grafanas …… 45 – 47
6.2.1 Connect Datasource …… 34

6.2.2 Import a Dashboard ……..…….. 34
6.2.3 Monitoring the Server from the Dashboard …….. 47

CHAPTER 7 48
CONCLUSION
7.1 Future Work ……..……..……..…….. 48
7.2 Conclusion ……..……..……..…… 48

REFERENCES ……..…….. 49

LIST OF TABLES

Table No.	Title	Page No.
Table 1.3	Technology Used	2
Table 2.4	Scrum Team Setup	5
Table 2.5	Sprint Planning	5 – 6
Table 2.6.1	Sprint 1 — Discovery & Design	6
Table 2.6.2	Sprint 2 — Core Frontend	6 – 7
Table 2.6.3	Sprint 3 — Backend & Data	7
Table 2.6.4	Sprint 4 — AI Assistant	7 – 8
Table 2.6.5	Sprint 5 — Polish & Release Readiness	8
Table 2.7	Scrum Overview	8
Table 4.1	Common Use Cases	30 – 31
Table 6.1	Testing Approach	45

LIST OF FIGURES

Figure No.	Title	Page No.
Figure 2.3	Scrum process overview	5
Figure 3.1	Overall DevOps Steps	10
Figure 3.2	Create A VPC With 2 Subnet	11
Figure 3.2.2	Creating EKS Worker Nodes Access	13
Figure 3.3.6	Install And Setup The Additional Tool In The Target Build-Server	19
Figure 3.4	Creating A Github Repository	20
Figure 3.5	Create Docker Images For This Project	21
Figure 3.6	Create CI/CD Fully Automate Pipeline With Jenkins	22
Figure 3.6.1	Necessary Credentials For This Pipeline	23
Figure 3.6.2	(a, b) Creating A Node For The Pipeline, (c) Connect The Node With The Server	24 – 25
Figure 3.6.3	(a, b, c, d) Creating A Jenkins Pipeline	25 – 27
Figure 3.6.4	Run The Pipeline	27
Figure 3.7.4	Kubernetes Cluster Setup	29
Figure 4.2	(a, b, c, d) Creating EC2 Server	31 – 32
Figure 4.3	Associate The Supabase Server for Elastic IP address	33
Figure 4.6.9	(a) Flowchart Of The Database, (b) Database Schema Visualization	38 – 39
Figure 5.1	(a) Design File, (b) UI/UX Design For The Frontend Application	40
Figure 6.2.1	(a) Connect With Datasource, (b) Connect With Datasource	46
Figure 6.2.2	Import The Dashboard	47
Figure 6.2.3	Monitoring The Server Health	47

CHAPTER 1

INTRODUCTION TO AI AGENT-BASED TRAVEL PLANNER PLATFORM USING AGILE SCRUM AND DEVOPS INTEGRATION

1.1 Introduction

At this time, integration of DevOps , Cloud and Agile methodology play a crucial role in the software industry.[1] Also, AI agent based software has high demand in this industry. So, they are trying to implement an application Agile methodology with fully automated DevOps steps and also integrating AI agents that will deploy on cloud storage.

What is Agile methodology and why Agile?

Agile software development offers a flexible, lightweight alternative designed to address the inefficiencies and rigid constraints of traditional waterfall methodologies. By minimizing administrative overhead and reducing delivery costs, Agile enables teams to seamlessly adapt to evolving requirements throughout the development lifecycle. This adaptability is achieved by structuring task execution and team collaboration around a core set of foundational values and principles.[2] In this project, Agile scrum is used.

What is DevOps and why DevOps?

DevOps represents a modern framework that bridges the traditional gap between software development and IT operations within the broader landscape of software process models. Originating from the need to align development efforts with system operations, this paradigm emphasizes tight cross-functional integration to drive organizational success. By unifying these stages, DevOps practices act as an enhancing mechanism that optimizes performance, delivery speed, and operational stability across the entire software lifecycle.[3]

1.2 Objectives

The specific, measurable, achievable, relevant, and time-bound goals for the project are:

- **Supabase:** Supabase is a platform for application development that provides a backend as a service (BaaS).[4] It is open source and an alternative for Google Firebase. Supabase project is a dedicated PostgreSQL database. Exposes RESTful APIs (via PostgREST) and GraphQL APIs directly from your database schema, removing the need to write boilerplate CRUD endpoints.
- **AI Agent:** This project also integrates an AI agent. This agent can assist clients to make their decision and can make some CRUD operations as required.

- **Automated the CI/CD pipeline:** As there integrated DevOps, the CI/CD pipeline is fully automated. As a developer just gave the comment on his own pc “git push”, the application automatically deployed after some additional testing. Those all steps are fully automated.
- **Monitoring:** After deploying the application, they use Prometheus and Grafana to monitor the cluster health of the application.

1.3 Technology used

Category	Technology	Purpose
Server-Side Logic & Database	Supabase	Execute business logic and process forms and manage the database.
Frontend	NextJS	A reactjs framework for frontend design.
Cloud	AWS	A cloud platform for running servers and databases.
CI/CD	Jenkins	CI/CD orchestration for building and testing application code.
Version Control	Git & Github	Writing code and pushing it to source control.
Containeriz Images	Docker & Docker Hub	Packaging the application into container images and storing them in a registry.
Orcastration	Kubernetes /EKS	Deploying, scaling, and managing containerized applications in production.
Monitoring	Prometheus & Grafana	Monitoring cluster metrics, system health, and visualizing performance dashboards.
AI agent	OpenRouter	An unified API gateway that aggregates multiple AI models.

1.4 Expected benefit

- **Agentic application:** This application is assisted by an AI agent. So, the application is more user friendly and fulfilled the current industry demand.
- **Secured:** They use Supabase as backend and database. Supabase is an open source project. So, the application is mostly secured.
- **Developer friendly:** As the CI/CD is fully automated, the application development part is hassle free.
- **User friendly:** This application UI/UX is very user friendly.

CHAPTER 2

INITIALIZE THE SOFTWARE DEVELOPMENT LIFE CYCLE

2.1 Introduction

The Software Development Life Cycle (SDLC) provides a structured framework for designing, building, and maintaining high-quality, reliable, and cost-effective software within strict schedule constraints [5]. Often referred to as a software process model, the typical SDLC encompasses five core sequential phases:

1. **Planning and Feasibility Analysis:** Senior stakeholders conduct initial consultations with clients and project sponsors to establish core project goals. Primary deliverables from this phase include risk assessments, quality assurance guidelines, and detailed feasibility reports.
2. **Requirements Definition:** Building upon initial planning, this stage captures and formalizes complete functional and technical specifications. The primary output is a Software Requirements Specification (SRS) document, which serves as a verified agreement between clients and engineering teams.
3. **Architectural Design:** Engineers utilize the SRS document to evaluate and formulate potential system designs. Multiple candidate architectures are systematically assessed against modularity, risk, cost, and performance criteria before selecting the optimal system architecture.
4. **Implementation and Coding:** Developers write source code based on the approved architecture. Using appropriate programming languages and development tools—including compilers, interpreters, and debuggers—the physical software product is constructed according to system specifications.
5. **Testing, Deployment, and Maintenance:** The application undergoes rigorous verification to ensure compliance with SRS requirements. Uncovered defects are logged, resolved, and re-tested. Once validated, the software is deployed to the production environment for user acceptance testing (UAT), followed by ongoing maintenance based on operational feedback.

2.2 Agile Scrum

Scrum execution relies on a structured sequence of time-boxed ceremonies that guide iterative software development. At the heart of this framework is the **Sprint**—a short, fixed-duration iteration that caps work-in-progress to maintain focus and prevent team overload [6].

The primary Scrum events include:

- **Sprint Planning:** Establishes clear goals for the upcoming iteration. Team members self-assess and select backlog items, defining task scope in alignment with the agreed Definition of Done (DoD).
- **Daily Scrum:** A brief daily alignment meeting where team members synchronize progress, discuss immediate goals, and improve coordination.
- **Sprint Review:** A collaborative session held at the end of a sprint involving the Scrum team and key stakeholders to demonstrate the increment, inspect progress, and gather user feedback.
- **Sprint Retrospective:** An internal team reflection held post-review to analyze challenges encountered during the iteration and identify actionable improvements for future Sprints.

2.3 Outline of the Scrum Structure

- **Product Backlog Creation:** The Product Owner creates and maintains a prioritized list of features, enhancements, and bug fixes, ensuring alignment with project goals.
- **Sprint Planning:** During planning sessions, the development team selects high-priority items from the Product Backlog, decomposing large requirements into smaller, manageable tasks.
- **Sprint Backlog Management:** The team establishes and continuously updates the Sprint Backlog, which acts as the dedicated task board for the active iteration.
- **Cadence Selection:** The team determines the fixed iteration length, which typically spans a two-week development window.
- **Daily Standups:** Brief daily synchronization meetings are conducted to review task progression, share updates, and identify operational bottlenecks.
- **Scrum Master Facilitation:** A certified Scrum Master guides the development team, ensuring adherence to Agile practices, removing roadblocks, and maintaining team focus.
- **Sprint Review & Stakeholder Evaluation:** At the conclusion of each iteration, key stakeholders and the Product Owner review the working increment to provide feedback and refine future priorities.

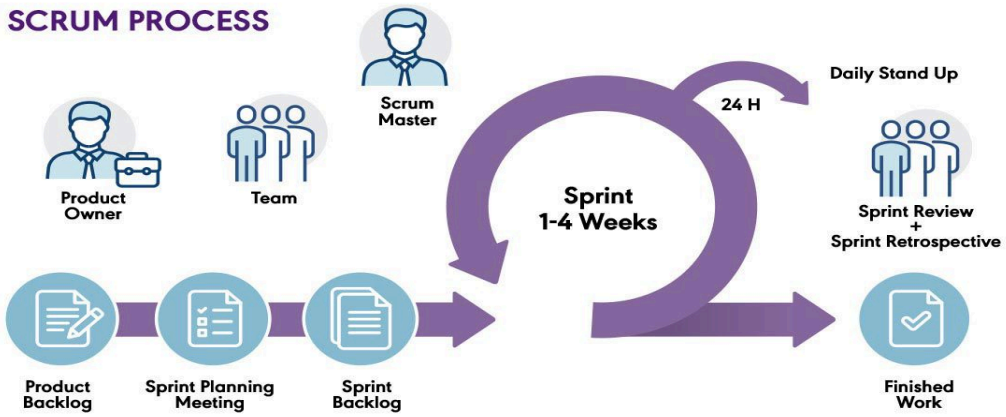


Figure 2.3 Scrum process overview

2.4 Scrum Team Setup

Role	Person
Product Owner	Bulbul Ahamed
Scrum Master	Self-managed
Development Team	Self (Frontend + Backend + AI Integration)
Sprint Length	1–2 weeks each
Story Point Scale	Fibonacci: 1, 2, 3, 5, 8, 13

2.5 Sprint Planning

Sprint	Duration	Sprint Goal	Key User Stories	Points	Deliverable
Sprint 1	Week 1–2	Define scope, design system, and information architecture	As a stakeholder, I want a clear sitemap and UI design so the team agrees on scope before coding	13	Sitemap, design system, static homepage mockup
Sprint 2	Week 3–4	Build the core frontend pages in Next.js + Tailwind	As a visitor, I want to browse tours, view details, and create an account so I can plan a trip	21	Next.js + Tailwind app: Home, Login, Signup, Tour Details, error/loading/404 pages

Sprint	Duration	Sprint Goal	Key User Stories	Points	Deliverable
Sprint 3	Week 5–6	Stand up the backend and connect real data	As a user, I want my account and bookings to persist so I can check out securely	21	Supabase schema (7 tables, RLS), real auth, live tour data, checkout + payments
Sprint 4	Week 7–8	Add AI-assisted trip planning	As a traveler, I want to chat with an AI assistant that recommends real tours and takes me to checkout	18	OpenRouter-powered chat, conversation persistence, AI-to-checkout handoff
Sprint 5	Week 9–10	Harden, polish, and prepare for release	As a user, I want a bug-free, responsive, branded experience across the whole site	13	Cross-page QA, navbar auth state, branding pass, deployment-ready build

2.6 Sprint Backlog

2.6.1 Sprint 1 — Discovery & Design

Task ID	Task	Priority	Points	Status
S1-T1	Gather requirements, define target users (travelers booking tours)	High	2	Done
S1-T2	Create sitemap (Home, Tours, Details, Auth, Chat, Checkout, Admin)	High	3	Done
S1-T3	Define design system: colors, typography, components	High	3	Done
S1-T4	Build static HTML homepage mockup with dummy tour data	High	5	Done

2.6.2 Sprint 2 — Core Frontend

Task ID	Task	Priority	Points	Status
S2-T1	Scaffold Next.js (App Router) + Tailwind CSS v4 project	High	3	Done

Task ID	Task	Priority	Points	Status
S2-T2	Build Home page (hero, search bar, featured tours grid, category filter)	High	5	Done
S2-T3	Build Login & Signup pages	High	3	Done
S2-T4	Build Tour Details page (dynamic route, gallery, itinerary)	High	5	Done
S2-T5	Build error.js, loading.js, not-found.js (Next.js special pages)	Medium	3	Done
S2-T6	Build Checkout / Payment UI (traveler details, payment method form)	High	2	Done

2.6.3 Sprint 3 — Backend & Data

Task ID	Task	Priority	Points	Status
S3-T1	Design Supabase schema: profiles, categories, tours, bookings, payments, reviews, chat tables	High	5	Done
S3-T2	Write RLS policies for every table (owner-scoped access)	High	3	Done
S3-T3	Set up Supabase client/server helpers + auth middleware (proxy.js)	High	3	Done
S3-T4	Wire Home + Tour Details pages to real Supabase data	High	3	Done
S3-T5	Wire Login/Signup to supabase.auth (real sessions)	High	3	Done
S3-T6	Wire Checkout to insert real bookings + payments rows	High	3	Done
S3-T7	QA: fix RLS gap (missing payments INSERT policy)	Medium	1	Done

2.6.4 Sprint 4 — AI Assistant

Task ID	Task	Priority	Points	Status
S4-T1	Integrate OpenRouter API (google/gemma-4-26b-a4b-it:free) via secure server route	High	5	Done

Task ID	Task	Priority	Points	Status
S4-T2	Ground AI responses in live tour data (system prompt injection)	High	3	Done
S4-T3	Persist chat conversations/messages to Supabase for logged-in users	Medium	3	Done
S4-T4	Implement AI to Checkout handoff (structured booking marker + validation)	High	5	Done
S4-T5	Add multi-model fallback + friendly error/retry handling for rate limits	Medium	2	Done

2.6.5 Sprint 5 — Polish & Release Readiness

Task ID	Task	Priority	Points	Status
S5-T1	Fix Server/Client Component boundary bugs (hero search form)	High	2	Done
S5-T2	Add dynamic Navbar auth state (user name + logout vs. login/signup)	High	3	Done
S5-T3	Migrate Tailwind v3 to v4 config	Medium	3	Done
S5-T4	Rebrand project (VoyageAI to FloraFlyers) across all pages/config	Low	2	Done
S5-T5	Cross-browser / responsive QA pass	Medium	3	Backlog

2.7 Scrum Overview of this Project

Software development life cycle is a continuous process. So, the Sprint will be created if it is needed. Those all processes were complete that are explained above in this paper. The process is described below in this paper.

The sprint summary is,

Sprint	Focus	Key Deliverable
Sprint 1	Discovery & Design	Sitemap, design system, static homepage mockup
Sprint 2	Core Frontend	Home, Login, Signup, Tour Details, error/loading/404 pages
Sprint 3	Backend & Data	Supabase schema (7 tables, RLS), real auth, live data, checkout
Sprint 4	AI Assistant	OpenRouter-powered chat, grounding, conversation persistence, booking handoff
Sprint 5	Polish & Release Readiness	Bug fixes, dynamic navbar auth state, Tailwind v4 migration, rebranding

CHAPTER 3

CREATE THE DEVOPS STEPS AND AUTOMATED THE CI/CD PIPELINE

3.1 Introduction

In present industry demand, DevOps is a compulsory part of any type of project. It makes the software development process easier. Also, the deployment process will be completely automated. So, it saves the time and effort of the production team.

In this project, there using DevOps tools and technology are,

Dev (Development) Tools Focus on coding, building, continuous integration, and image creation:

- **Developer & GitHub:** Writing code and pushing it to source control.
- **Jenkins:** CI/CD orchestration for building and testing application code.
- **Docker / Docker Hub:** Packaging the application into container images and storing them in a registry.

Ops (Operations) Tools Focus on infrastructure provisioning, container orchestration, configuration, and monitoring:

- **Kubernetes (EKS) & Helm:** Deploying, scaling, and managing containerized applications in production using Helm charts.
- **Prometheus & Grafana:** Monitoring cluster metrics, system health, and visualizing performance dashboards.
- **Terraform:** Provisioning cloud infrastructure (AWS resources) via Infrastructure as Code (IaC).
- **Ansible:** Managing server configurations, deployments, and maintenance tasks.

In the next stages, they describe how they implement those tools and technology in this project.

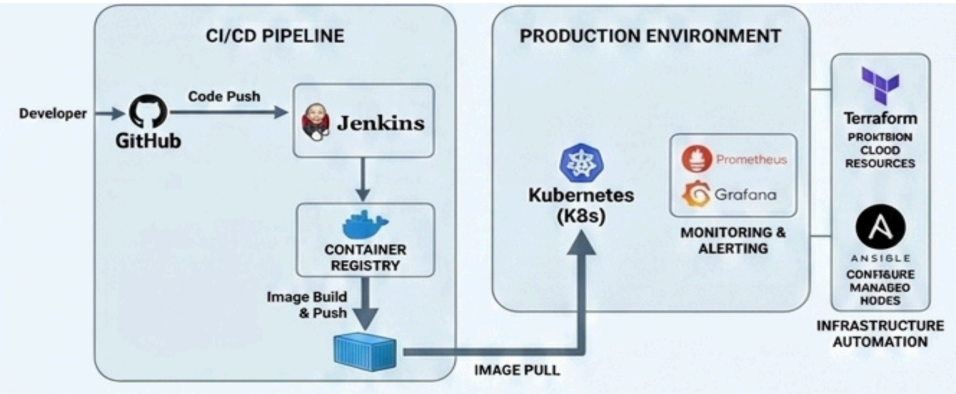


Figure 3.1 Overall DevOps Steps

3.2 Build Infrastructure with Terraform for AWS

On the local pc, they use Ubuntu OS. Now, describe the terraform files that are used on this project.

- **Provider Configuration (provider "aws"):** Specifies AWS as the cloud platform and sets us-east-1 as the deployment region.
- **Network Topology (aws_vpc, aws_subnet, aws_internet_gateway):** Provisions an isolated Virtual Private Cloud (10.0.0.0/16) spanning across two Availability Zones (us-east-1a and us-east-1b) for high availability. Public route tables and an Internet Gateway enable bidirectional internet traffic.
- **Compute Infrastructure (aws_instance, aws_key_pair):** Deploys an EC2 instance tagged frontend-server using an m7i-flex.large instance type with 20 GB of GP3 storage in us-east-1a. Public SSH access is managed via the imported public key file.
- **Security Group Rules (aws_security_group):** Functions as a stateful firewall controlling inbound network access:
    - **Port 22 (SSH):** Administrative remote terminal access.
    - **Ports 80 / 3000:** Web access for standard HTTP and the Next.js Web UI.
    - **Port 8080:** Jenkins Continuous Integration / Continuous Deployment dashboard.
    - **Ports 3001 / 9090 / 9100:** Observability and metrics stack (Grafana, Prometheus, and Node Exporter).
    - **ICMP:** Allows ping commands for network troubleshooting.
- **Managed Services (module "eks"):** Connects the VPC and public subnets directly into custom Kubernetes (EKS) modules for container orchestration.
- **Variable Management (variables.tf / terraform.tfvars):** Parameterizes the infrastructure (e.g., CIDR blocks, AMI IDs, instance types) to decouple environment configurations from executable Terraform code.

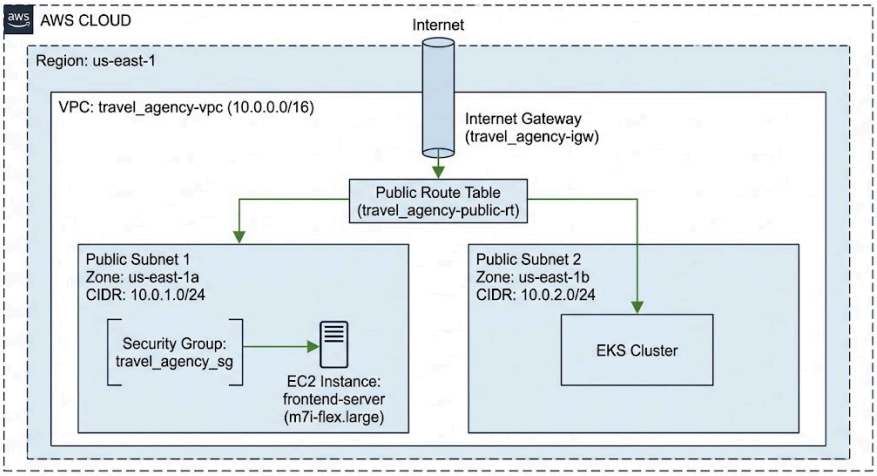


Figure 3.2 Create A VPC With 2 Subnet

3.2.1 Setup the Security Group for EKS

Inbound Traffic:

- Traffic using port 22 (SSH) is allowed *from* any IP address (0.0.0.0/0) on the internet.
- All other inbound traffic is blocked.

Outbound Traffic:

- All traffic, using any port and protocol, is allowed *to* any IP address (0.0.0.0/0) on the internet.

3.2.2 Setup the EKS

These Terraform files deploy a managed Kubernetes cluster on AWS using the Elastic Kubernetes Service (EKS).

The configuration is divided logically into three stages: setting up Identity and Access Management (IAM) for security, defining the EKS Control Plane (the master), and defining the EKS Node Groups (the workers).

Part 1: IAM Configuration (Security)

EKS requires distinct IAM roles to function. One role allows the AWS EKS service itself to manage resources (like network interfaces), and another role allows the EC2 instances acting as worker nodes to check in with the control plane and pull container images.

1. EKS Control Plane (Master) Role

This role grants the EKS service permissions to create and manage the cluster components.

- **aws_iam_role.master**: Creates the role named ed-eks-master.

    - *Assume Role Policy*: Allows the eks.amazonaws.com service principal to assume this role.
- **Policy Attachments**: The code attaches three AWS-managed policies that are required for basic EKS functionality:
    - AmazonEKSClusterPolicy: Core permissions for cluster creation.
    - AmazonEKSServicePolicy: Permissions to manage services.
    - AmazonEKSVPCResourceController: Allows EKS to manage security groups and network interfaces.

2. EKS Worker Nodes Role

This role is assumed by the EC2 instances that join the Kubernetes cluster.

- **aws_iam_role.worker**: Creates the role named ed-eks-worker.
    - *Assume Role Policy*: Allows the ec2.amazonaws.com service principal (the instances themselves) to assume this role.
- **Policy Attachments**:
    - AmazonEKSWorkerNodePolicy: Minimum permissions needed to connect to the EKS cluster.
    - AmazonEKS_CNI_Policy: Allows the Amazon VPC CNI plugin to manage networking.
    - AmazonEC2ContainerRegistryReadOnly: Allows nodes to pull images from ECR.
    - AmazonSSMManagedInstanceCore: Allows using Systems Manager to manage the nodes (e.g., SSH-less access).
    - AWSXRayDaemonWriteAccess / AmazonS3ReadOnlyAccess: Custom additions likely needed for your application's monitoring or storage needs.
- **Autoscaler Support**:
    - **aws_iam_policy.autoscaler**: Defines a custom policy allowing instances to read Auto Scaling Group settings and modify the "desired capacity". This is needed if you intend to deploy the Kubernetes Cluster Autoscaler.
    - This custom policy is attached to the worker role.
- **aws_iam_instance_profile.worker**: This is a container for the IAM role. In AWS, you cannot attach an IAM role directly to an EC2 instance; you must attach an Instance Profile that contains that role.

Part 2: EKS Control Plane (aws_eks_cluster)

This resource provisions the Kubernetes control plane—the "master" nodes managed by AWS.

- **name**: Set to devopssteps-eks-1.

- **role_arn**: Attaches the ed-eks-master IAM role created earlier. This gives AWS the permissions needed to set up the control plane.
- **vpc_config**: Specifies the subnets where EKS will create elastic network interfaces (ENIs) to communicate with the worker nodes. It uses two subnets provided via variables.
- **depends_on**: Crucial safety setting. It ensures that Terraform creates the IAM role *and* attaches all three required policies *before* attempting to create the cluster[cite: 1.2.4]. Without this, cluster creation might fail due to missing permissions.

Part 3: EKS Node Group (aws_eks_node_group)

This resource provisions the actual worker nodes where your containers will run. EKS Managed Node Groups automate the creation and updating of the underlying EC2 Auto Scaling Group.

- **cluster_name**: References the name of the control plane created above to link them.
- **node_role_arn**: Attaches the ed-eks-worker IAM role. Every instance created by this node group will use this role.
- **instance_types**: Configured to use t3.small instances.
- **scaling_config**: Sets the minimum (1), desired (2), and maximum (3) number of nodes in the group.
- **update_config**: During a Kubernetes version upgrade, this ensures that only 1 node is unavailable at any given time, maintaining application availability.
- **remote_access**: Specifies the SSH key (key-s) used to log into the nodes, and restricts SSH access to the source security groups defined by the variable sg_ids.

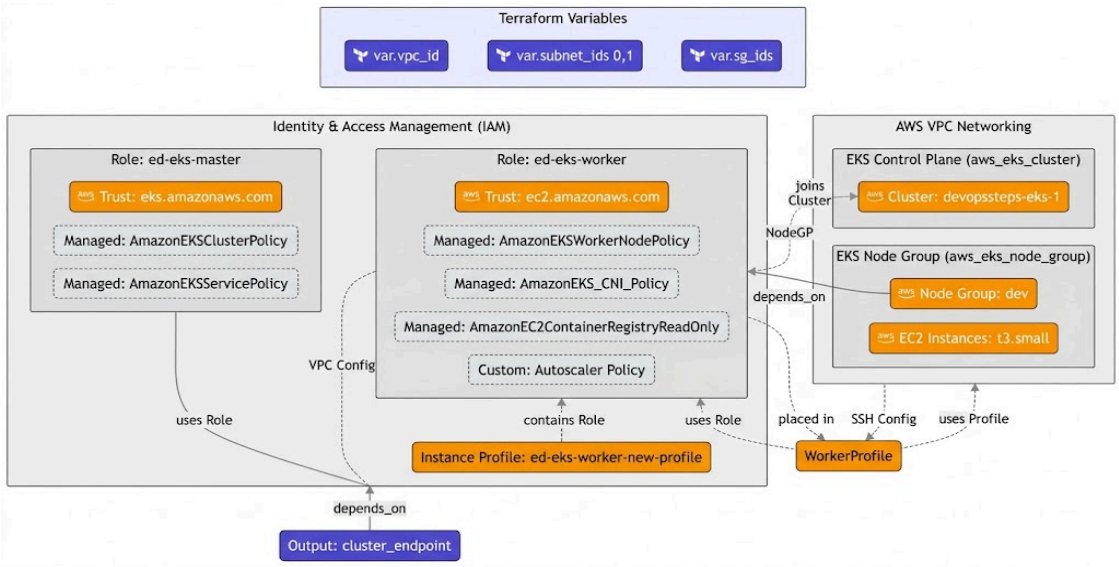


Figure 3.2.2 Creating EKS Worker Nodes Access

3.3 Managing Server with Ansible

At first connection the server with the personal pc, they need a ssh key the command to generate the ssh key is,

```
ssh-keygen -t ed25519 -C "ssh-key"
```

Then, creating the yml files to manage the ansible server

3.3.1 Inventory file

The unlabelled snippet at the beginning is an **Ansible Inventory File**.

- It defines a host group [all].
- Within this group, it defines a logical host named build-server.
- It provides the connection details:
    - **ansible_host=3.220.167.48**: The actual public IP address of the target machine (likely an AWS EC2 instance).
    - **ansible_user=ubuntu**: The remote user Ansible should use to log in (default user for Ubuntu images on AWS).
    - **ansible_ssh_private_key_file=~/.ssh/ssh-key**: The path to the SSH private key used to authenticate with the server.

The other playbook files reference these hosts: build-server group to know where to execute their tasks.

3.3.2 Playbook: Install Kubectl and AWS CLI

This playbook, titled **"Install kubectl and AWS CLI"**, focuses on equipping the build-server with command-line tools necessary for interacting with Kubernetes clusters and AWS services. These tools are essential for any server involved in CI/CD pipelines (such as a Jenkins node).

- **Host/Privilege**: Target is build-server, using elevated (become: true) privileges.
- **Task 1: System Package Prerequisites**: Installs several standard Linux utilities and libraries using the apt module.
    - curl, unzip: Required to download and extract the CLI binaries.
    - apt-transport-https, ca-certificates, gnupg, lsb-release: Prerequisites for securely adding external repositories (like those for Docker or Grafana).
- **Task 2: Download and install AWS CLI v2**:
    - Uses a **shell** block to:
        1. **curl**: Download the specific AWS CLI v2 zip installer from the official AWS source.
        2. **unzip**: Extract the installer to /tmp.

3. **install --update**: Run the installation script, updating the binary if it already exists.
    - **creates: /usr/local/bin/aws**: This is a critical argument. Ansible first checks if the final binary (/usr/local/bin/aws) already exists. If it does, Ansible skips this task, ensuring it does not redownload and reinstall the CLI unnecessarily.
- **Task 3: Download and install kubectl**:
    - Uses a **shell** block to:
        1. **Nested curl**: Dynamically determine the URL for the current stable release of kubectl for Linux x86_64, then download it.
        2. **install**: Copy the downloaded binary to a location in the system PATH (/usr/local/bin/), set ownership to root, and set executable permissions (0755).
        3. **rm**: Clean up the temporary download file.
    - **creates: /usr/local/bin/kubectl**: As with the AWS CLI, this ensures the task is skipped if kubectl is already installed.

3.3.3 Playbook: Install Java and Docker

The second playbook, which is unlabelled but contains tasks like **"Install Java and Docker"**, prepares the server to host CI/CD tools (like Jenkins) and execute containerized builds.

- **Host/Privilege**: Target is build-server, using elevated (become: true) privileges.
- **Task 1: Lock Wait (Retry Mechanism)**:
    - Background APT processes (like automatic updates) can often lock the package database (/var/lib/dpkg/lock-frontend), causing the apt module to fail.
    - This task uses the **shell** module with a while loop that continuously checks (using fuser) if the lock is present[cite: 3]. It sleeps for 5 seconds between checks and only finishes once the lock is released.
    - changed_when: false ensures that Ansible doesn't incorrectly report that this task "changed" the system.
- **Task 2: Install Java and Docker**:
    - Installs standard system packages:
        - default-jre: Java Runtime Environment (required to run Jenkins master or agent).
        - docker.io: The primary Docker engine.
    - **apt**: Updates the cache and installs the packages.
    - **register: apt_status**, **until: apt_status is success**, **retries: 20**, **delay: 10**: This is an internal Ansible retry loop for added robustness. It will attempt the installation up to 20 times, waiting 10 seconds between attempts, ensuring it can survive temporary network glitches or brief package database locks.

- **Task 3: Enable Docker Service**:
    - Uses the **service** module to ensure that the newly installed docker daemon is running (state: started) and configured to start automatically on system reboot (enabled: yes).
- **Task 4: Docker Group Association**:
    - By default, only the root user can interact with the Docker socket (/var/run/docker.sock).
    - This task uses the **user** module to add the standard login user (ubuntu) to the supplementary docker group[cite: 6]. The append: yes argument is vital: it *adds* the group rather than replacing the user's existing groups. This allows CI/CD tools running as the ubuntu user to execute Docker commands (like building or pushing images) without needing sudo.

3.3.4 Playbook: Install Prometheus and Grafana on AWS EC2

This third playbook, **"Install Prometheus and Grafana on AWS EC2"**, is the most complex. It is entirely dedicated to deploying a full monitoring and observability stack on the build-server. It installs Prometheus for metric storage, Node Exporter for gathering OS metrics, and Grafana for visualization. It also implements best-practice system hardening by running services as dedicated unprivileged system users.

- **Host/Privilege/Vars**: Target is build-server, using elevated (become: true) privileges. It defines variables for the desired versions of Prometheus and Node Exporter (prometheus_version and node_exporter_version), allowing easy upgrades by changing these values.
- **Phase 1: System Setup**: Updates the package manager cache (apt) and installs fundamental network and repository utilities needed for the subsequent tasks: wget, curl, apt-transport-https, software-properties-common, and gpg.
- **Phase 2: Prometheus Setup**:
    1. **group**, **user**: Creates a dedicated system group and user named prometheus. Crucially, the user is created with /bin/false as its shell (meaning it cannot log in) and create_home: no. This is a best practice for securing system daemons.
    2. **file**: Creates the necessary directories for Prometheus:
        - /etc/prometheus: For configuration files.
        - /var/lib/prometheus: For storing the time-series database (TSDB).
        - It sets the owner and group of these directories to the new prometheus user.
    3. **unarchive**: Downloads the official Prometheus binary release (using the defined version variable) from GitHub and extracts it directly to the remote /tmp directory.

4. **copy**: Moves the core executable binaries (prometheus, promtool) from /tmp into the standard system PATH (/usr/local/bin), setting the owner and executable permissions.
5. **copy**: Moves the default console templates and libraries (consoles, console_libraries) from /tmp into the configuration directory (/etc/prometheus/).
6. **copy**: Creates a basic **prometheus.yml** configuration file. This file is simple:
    - Sets a global scrape_interval of 15 seconds.
    - Defines two scrape jobs:
        1. prometheus: Targets Prometheus's own internal metrics (on localhost:9090).
        2. node_exporter: Targets metrics from the Node Exporter instance running locally (on localhost:9100).
7. **copy**: Creates a **Systemd Service Unit File** (/etc/systemd/system/prometheus.service). This unit file tells the OS how to manage the Prometheus daemon. It specifies:
    - It should run after network services are online.
    - It should execute as the prometheus user/group.
    - The exact ExecStart command, including arguments pointing to the configuration file (--config.file=...) and the database storage path (--storage.tsdb.path=...).

- **Phase 3: Node Exporter Setup**:
    1. **unarchive**: Downloads and extracts the official Node Exporter binary release (using the version variable) to /tmp.
    2. **copy**: Moves the core node_exporter binary to /usr/local/bin, setting ownership and executable permissions. It runs as the same prometheus user.
    3. **copy**: Creates a **Systemd Service Unit File** for Node Exporter (/etc/systemd/system/node_exporter.service), ensuring the OS can manage it automatically.
- **Phase 4: Grafana Setup**:
    1. **get_url**, **command (gpg --dearmor)**: Downloads the official Grafana GPG signing key, demurs it, and places it into the trusted system keychain (/etc/apt/keyrings/). This ensures the OS can verify that the packages are authentic.
    2. **copy**: Creates a new APT repository source file (/etc/apt/sources.list.d/grafana.list) pointing to the official Grafana repository, signed by the key added above.
    3. **apt**: Installs the grafana package, automatically updating the package cache first to pick up the new repository.

4. **lineinfile**: Performs a localized configuration update. It searches the main Grafana configuration file (/etc/grafana/grafana.ini) for the default HTTP port setting (3000) and changes it to http_port = 3001. This is useful if port 3000 is already in use by another service on the build server.

- **Phase 5: Service Activation**:
    1. **systemd (daemon_reload: yes)**: Instructs Systemd to reload its configuration from disk, forcing it to recognize the new unit files created for Prometheus and Node Exporter.
    2. **systemd (restart/enable loop)**: A loop using the systemd module that performs two actions on prometheus, node_exporter, and grafana-server:
        - **state: restarted**: Immediately restarts the services to ensure they are running with any new configurations applied.
        - **state: enabled**: Configures them to start automatically when the system boots.

3.3.5 Configure AWS Credentials on Managed Nodes

After installing cli, it needs configuration. For securing the configuration values, they create ansible **vault.yml** file,

```
ansible-vault create vault.yml
```

Then, gives a password and necessary values,

```
vault_aws_access_key: "YOUR_ACTUAL_AWS_ACCESS_KEY_ID"
vault_aws_secret_key: "YOUR_ACTUAL_AWS_SECRET_ACCESS_KEY"
```

After that, they create an **access_aws.yml** file to configure the AWScli. It securely deploys the authentication credentials needed by the previously installed tools (the AWS CLI and kubectl/EKS) to interact with AWS APIs.

- **Host/Privilege**: Target is build-server. No elevated privilege (become: true) is needed, as these files are created in the user's home directory.
- **vars_files: - vault.yml**: This is a critical security step. Instead of defining sensitive keys in the playbook itself, it loads them from an external file named vault.yml. The vault naming strongly implies this file is encrypted using Ansible Vault, protecting the secrets. The loaded variables are vault_aws_access_key and vault_aws_secret_key.
- **Task 1: Ensure .aws directory exists**: Uses the **file** module to create the standard hidden directory (~/.aws) for storing AWS configuration. It sets the mode to 0700 (read/write/execute *only* for the user), protecting the contents from other users.
- **Task 2: Deploy AWS credentials file**:
    - Uses the **copy** module to create the specific **credentials** file inside that directory.

- Sets the mode to 0600 (read/write *only* for the user), as this file contains highly sensitive secrets.
- **content**: Uses Jinja2 templating to inject the unencrypted secrets loaded from the vault.yml file into the standard AWS credentials file format.
  - It defines a [default] profile.
  - It maps the vault variables to aws_access_key_id and aws_secret_access_key.

3.3.6 Run the Ansible files

Run that command on terminal, then the ansible server do it's work,

```
ansible-playbook -i hosts \
  install-kubectl-awscli.yml \
  instal_docker_build_server.yml \
  access_aws.yml \
  deploy-monitoring.yml \
  --ask-vault-pass
```

Once complete, the server is now fully equipped with a modern build environment, monitoring, and the necessary credentials to begin deploying applications to EKS.

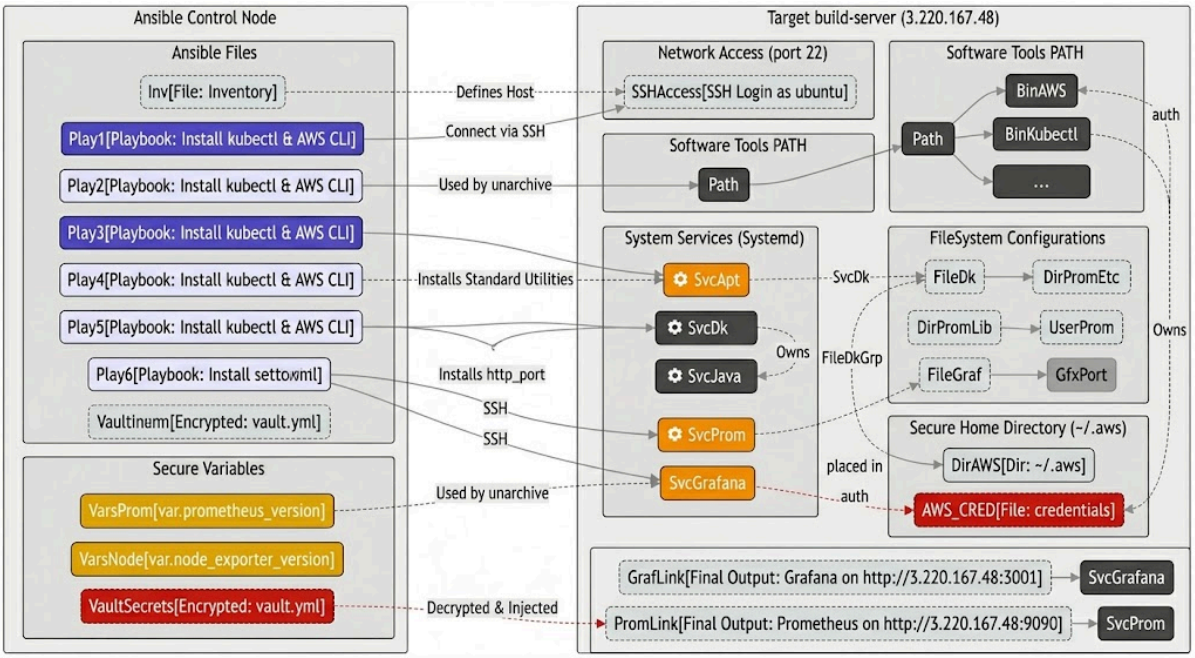


Figure 3.3.6 Install And Setup The Additional Tool In The Target Build-Server

3.4 Create a Github Repository

For version control and development necessity, a github repository is a necessary platform. Now, create a github repository for this project.

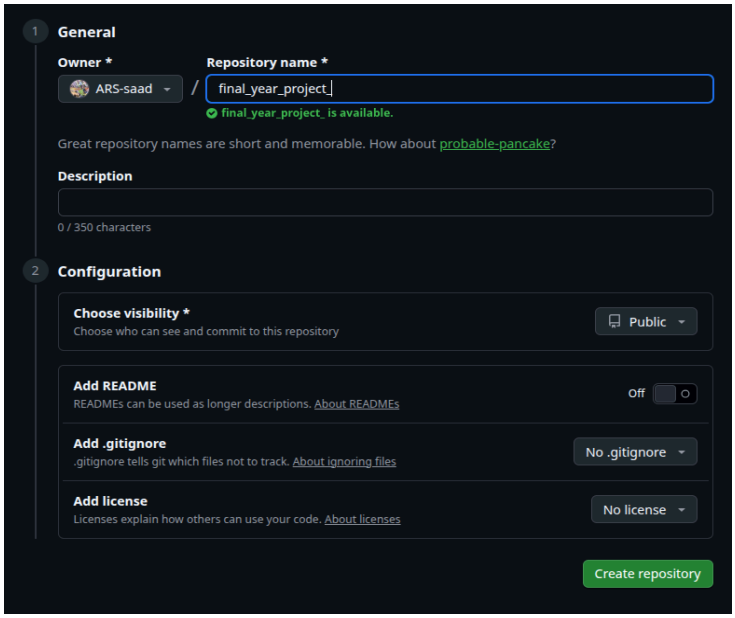


Figure 3.4 Creating A Github Repository

Connect the local folder with this repository according to the terminal command.

```
git remote add origin https://github.com/ARS-saad/final_year_project_.git
git branch -M main
git push -u origin main
```

As they build a **Nextjs** application, they need a **.gitignore** file to ignore unnecessary files from git push,

- **Dependencies (/node_modules, /.pnp, .pnp.*, .yarn/*, etc.)**: Ignores installed packages and zero-install/Plug'n'Play cache files to keep the repository lightweight. It includes negation rules (!) to allow committing essential Yarn release scripts, patches, and plugins if using Yarn Berry.
- **Testing (/coverage)**: Excludes code coverage reports generated during test runs.
- **Next.js & Production (/.next/, /out/, /build)**: Prevents build output directories, static exports, and compiled distribution files from being tracked, ensuring clean production builds on deploy servers.
- **Misc (.DS_Store, *.pem)**: Filters out macOS system metadata files and private SSL/TLS certificate key files to prevent accidental commits of sensitive keys.
- **Debug Logs (*debug.log*, yarn-error.log*)**: Ignore error logs created during package installation or execution across npm, yarn, and pnpm.

- **Environment Files (.env*)**: Prevents local environment files containing API keys, database URLs, or secret credentials from leaking into public or shared repositories.
- **Vercel (.vercel)**: Ignores local project deployment configurations and cache managed by the Vercel CLI.
- **TypeScript (*.tsbuildinfo, next-env.d.ts)**: Excludes incremental build metadata generated by TypeScript and auto-generated Next.js environment type declarations.

3.5 Create a Docker File

In the same directory, create a **Docker** file to generate the image of this project. This is a **3-stage Dockerfile** designed to build and run a production-ready **Next.js** application:

1. **Dependencies (deps):** Detects whether npm, yarn, or pnpm is used and installs node packages based on the project's lockfile.
2. **Builder (builder):** Copies installed dependencies and application code. It injects build-time environment variables (like site URL and Supabase keys) passed from Jenkins and compiles the application (npm run build).
3. **Runner (runner):** Configures a lightweight production container using Next.js standalone mode. It enhances security by creating and switching to a non-root system user (nextjs), copies only essential static files, and exposes port 3000 to serve server.js.

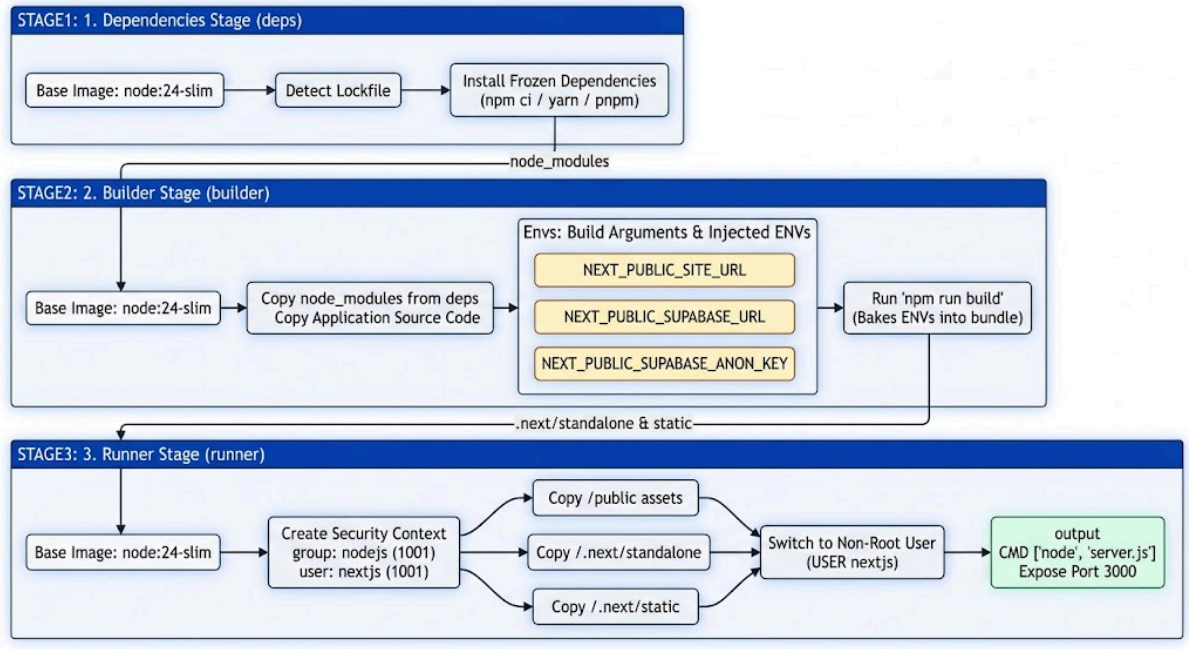


Figure 3.5 Create Docker Images For This Project

Also add a **.dockerignore** file to specify which local files and directories Docker should ignore when copying assets into the image during the build process.

- **node_modules**: Local packages are massive and platform-dependent. Clean ones are always re-installed on the production server.

- **.next & out**: Local build artifacts. Fresh, optimized production builds are always generated during deployment.
- **.env*.local**: Local development secrets (API keys, passwords). Production secrets are injected securely via the deployment platform (e.g., Jenkins or Docker secrets).
- **README.md & .git**: Documentation and version history. They waste space and have zero function in the production environment.

3.6 Create Jenkins CI/CD Pipeline

Creating a **Jenkinsfile** to pipeline automates the end-to-end build, containerization, and deployment process of a Next.js application to an AWS EKS Kubernetes cluster.
Environment & Setup: Runs on a Jenkins node labeled "node". It sets up AWS variables, credentials for Docker Hub/GitHub/Supabase, and tags the container image using Jenkins' ${BUILD_NUMBER}.

Stage 1: Fetch or Provision Load Balancer IP:

- Authenticates with the AWS EKS cluster (aws eks update-kubeconfig).
- Applies k8s/service.yaml to ensure an AWS Elastic Load Balancer (ELB) service exists.
- Polls Kubernetes in a loop (up to 30 retries, 10-second sleep intervals) until the ELB external hostname or IP is assigned.
- Saves that address into NEXT_PUBLIC_SITE_URL so Next.js knows its public URL.

Stage 2: Docker Build and Push: Log into Docker Hub securely using credentials, builds the Docker image passing Next.js client-side build arguments (--build-arg), and pushes 890iop/travel_agency:<BUILD_NUMBER> to Docker Hub.
Stage 3: Deploy Application to K8s: Uses envsubst to dynamically replace variables inside configmap.yaml, secret.yaml, and deployment.yaml, applies them to Kubernetes, and triggers a kubectl rollout restart to force pods to pull the new image tag immediately.
Post Actions: Cleans up local Docker images generated on the Jenkins worker agent (docker rmi) and deletes workspace files (cleanWs()) to conserve disk space.

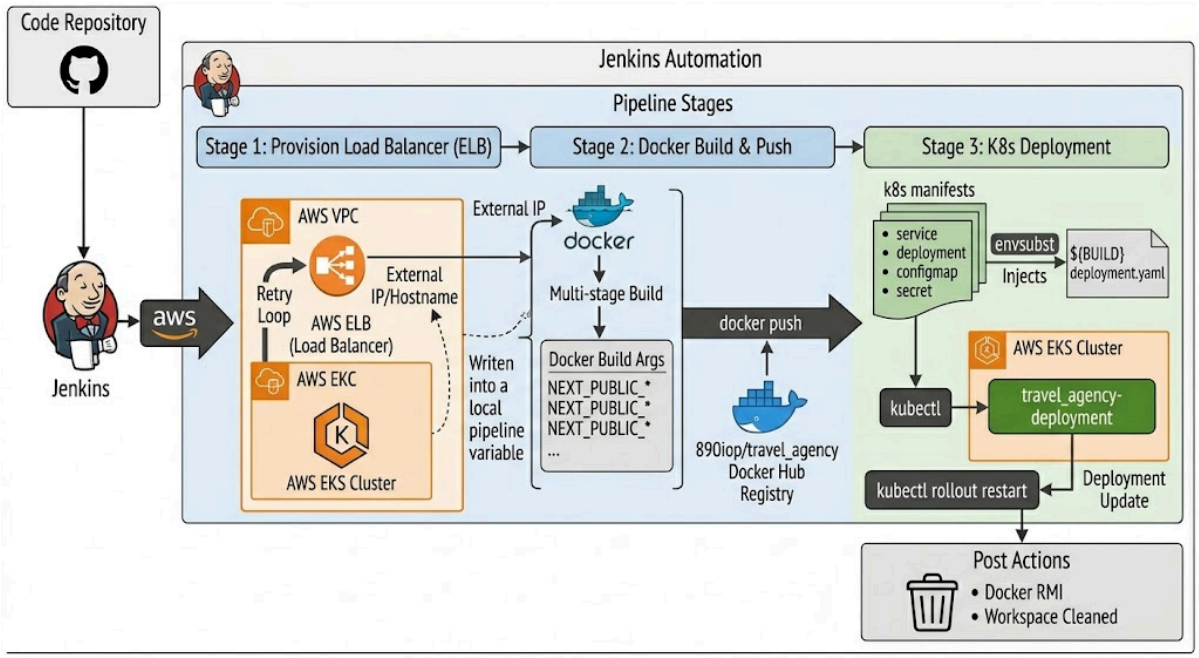


Figure 3.6 Create CI/CD Fully Automate Pipeline With Jenkins

3.6.1 Creating Jenkins Credential

To deploy an application, it needs some environment variables, some access keys and tokens. For security purposes, there is no option to push those secure text in github or some publicly available repository. In this case, as they want to deploy our application automatically, so they create some credentials and throw the pipeline script they can access those secret keys. When the pipeline is running, Jenkins places those secret keys as they need and deploy the application automatically. Now, creating Jenkins they need to go Jenkins>Manage Jenkins>Credentials then Add Credentials. Then as needed, choose the options and past the secret keys.

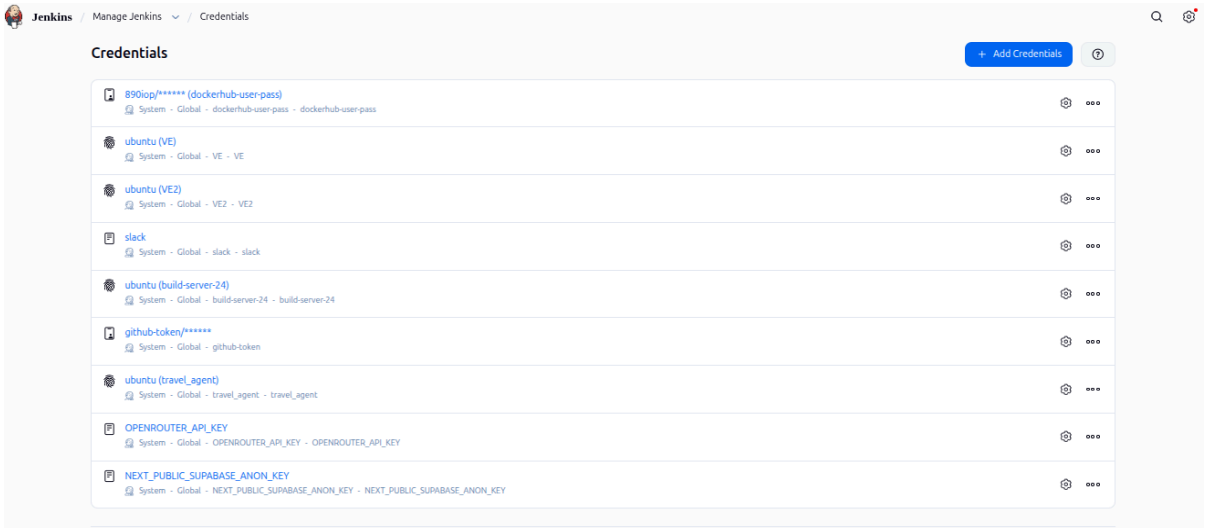


Figure 3.6.1 Necessary Credentials For This Pipeline

3.6.2 Creating Nodes in Jenkins

In Jenkins, a **Node** is any machine that is capable of executing build jobs and running pipelines. To creating node, need to go Jenkins>Nodes>New node and then,

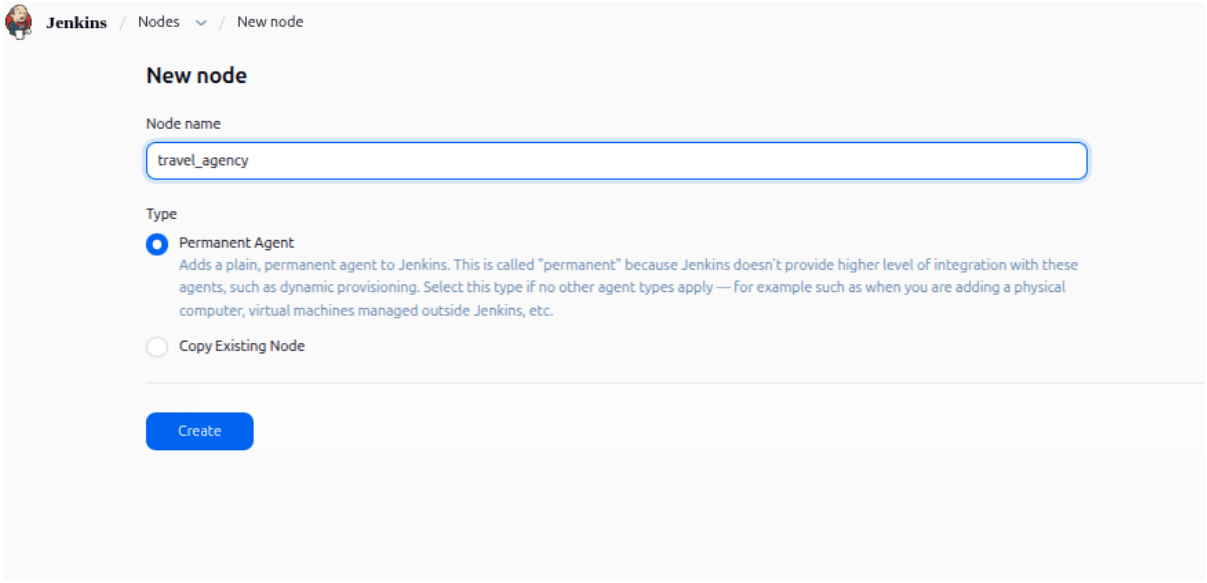


Figure 3.6.2 (a) Creating A Node For The Pipeline

Then give the necessary informations for the project,

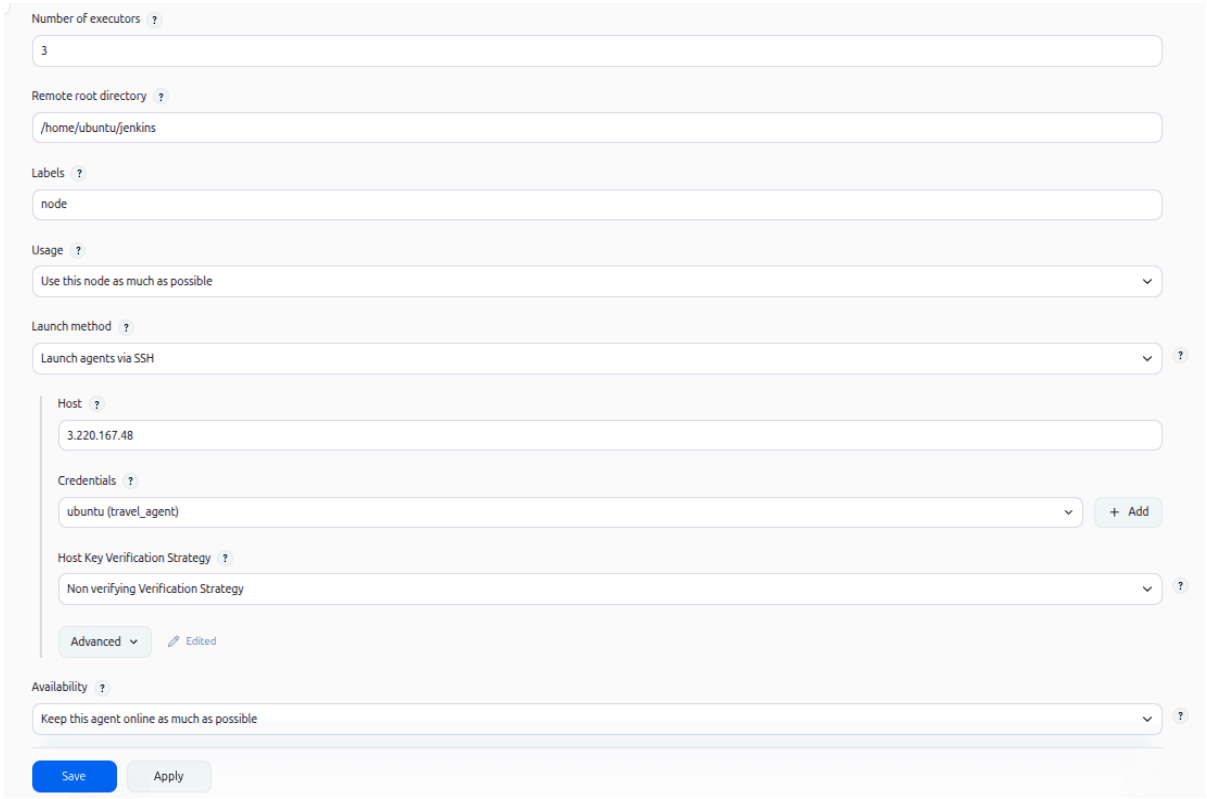


Figure 3.6.2 (b) Creating A Node For The Pipeline

Then apply and connect the node,

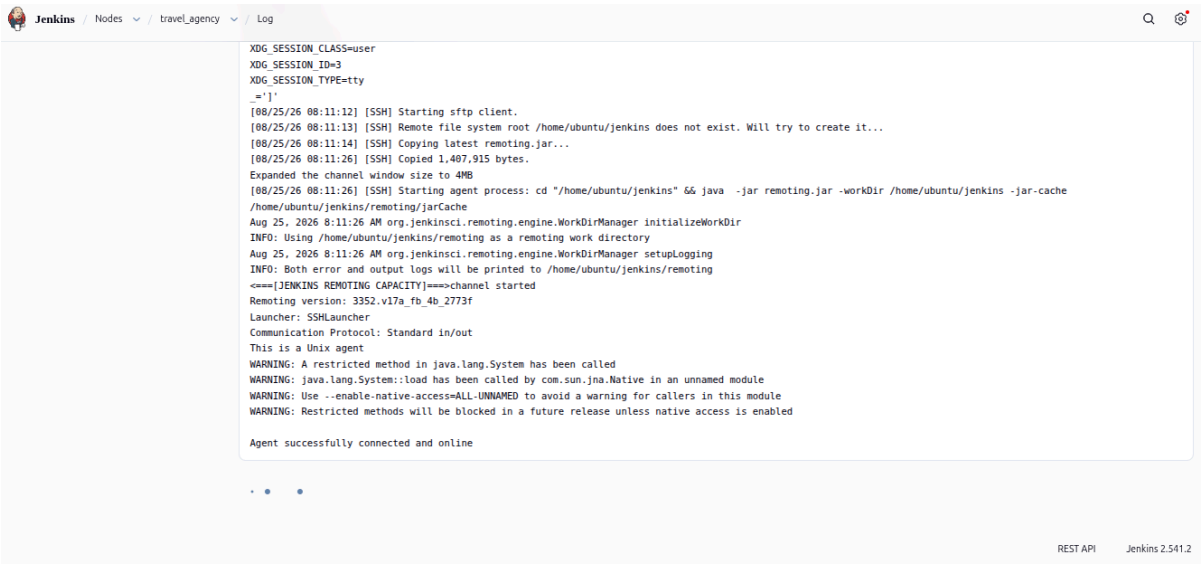


Figure 3.6.2 (c) Connect The Node With The Server

3.6.3 Creating a Pipeline According to this Node

After creating the node a pipeline needs to be created. According to this pipeline Jenkins runs its own operation.

At first, go to the Jenkins>All>New Item, then create a pipeline to this process.

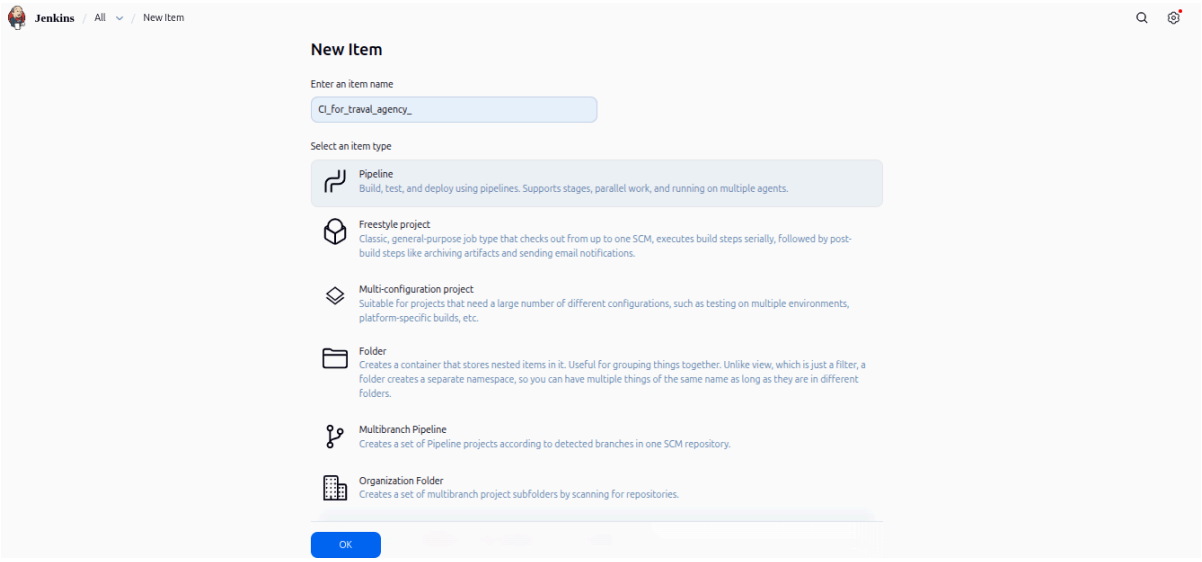


Figure 3.6.3 (a) Creating A Jenkins Pipeline

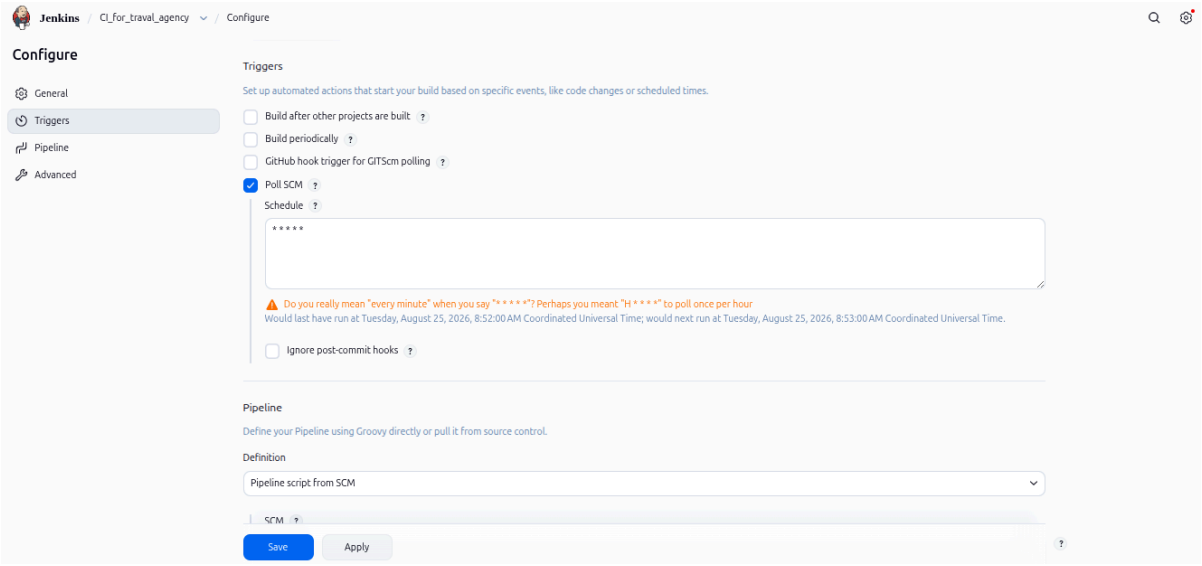


Figure 3.6.3 (b) Creating A Jenkins Pipeline

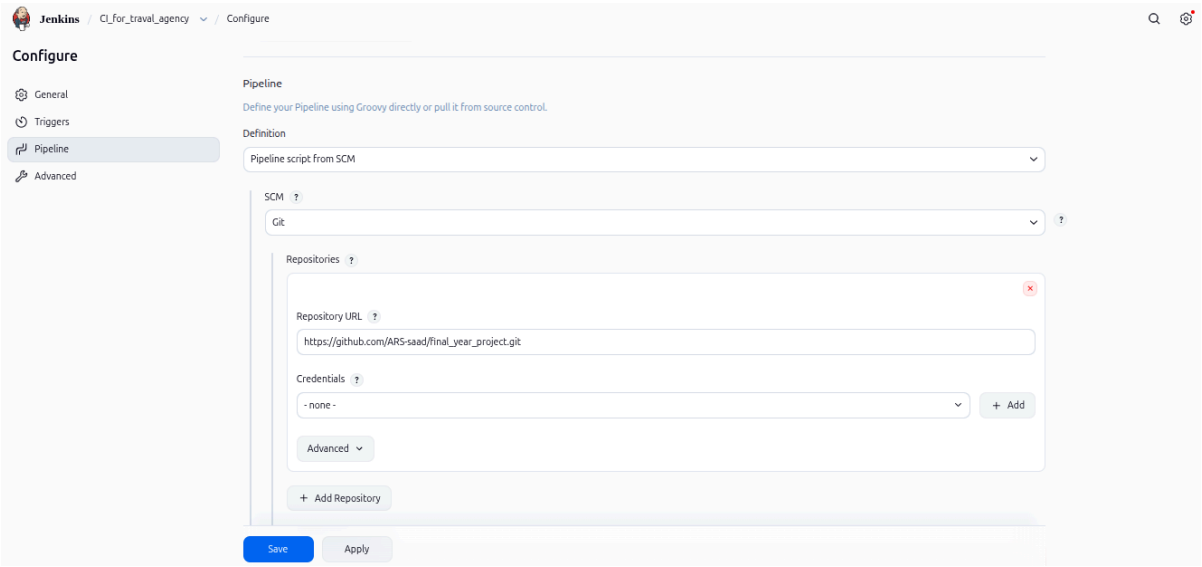


Figure 3.6.3 (c) Creating A Jenkins Pipeline

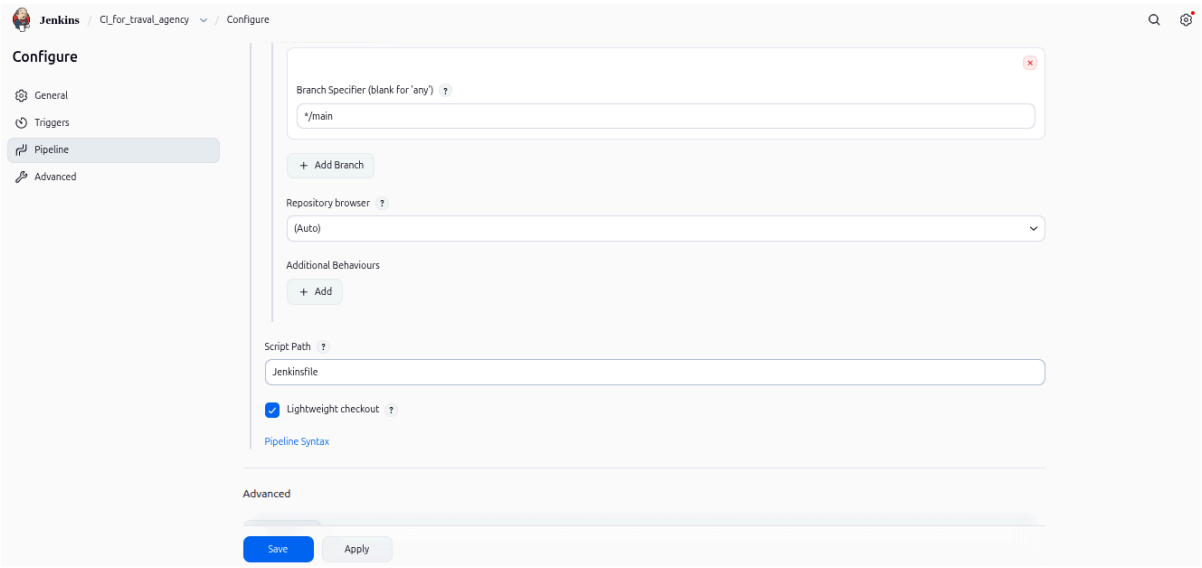


Figure 3.6.3 (d) Creating A Jenkins Pipeline

3.6.4 Run the Pipeline

After creating a pipeline, the pipeline runs automatically. And creating the application fully operational.

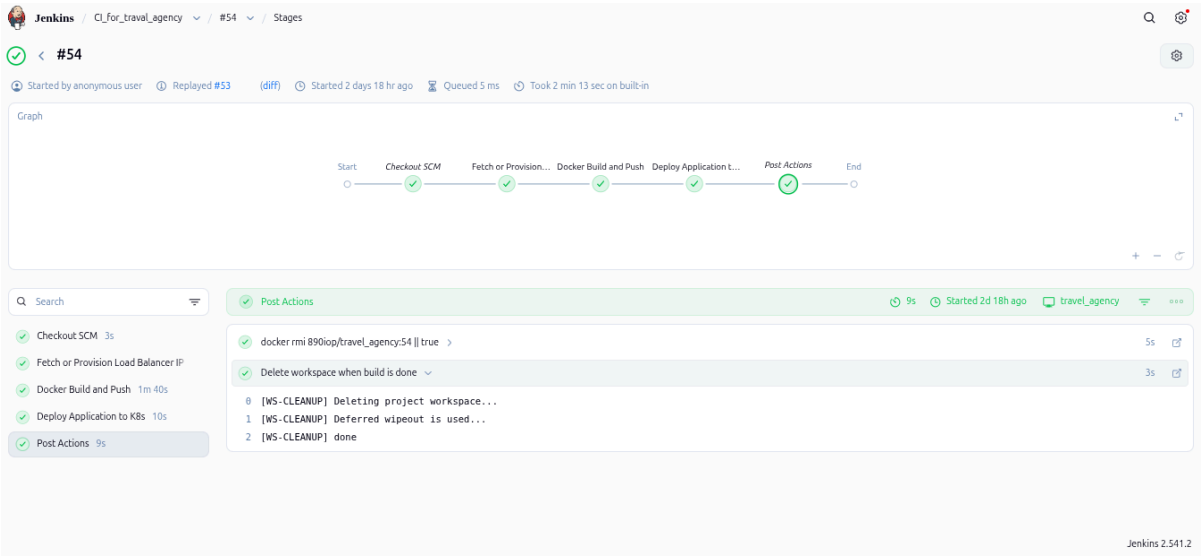


Figure 3.6.4 Run The Pipeline

3.7 Create Kubernetes Files in the Project Repository

Now, creating a k8s folder. In this folder, they are creating 4 yaml files for kubernetes setup in this project. They use standard Kubernetes resources to manage configuration, application life cycle, sensitive data, and network access.The configuration follows modern best

practices by separating configuration from code (using ConfigMaps and Secrets) and defining resource constraints and health checks for operational stability.

3.7.1 Configuration (ConfigMap)

- **File:** apiVersion: v1, kind: ConfigMap, metadata.name: travel-agency-config
- **Purpose:** Stores non-sensitive, plaintext configuration parameters as key-value pairs.
- **Details:** This file defines standard runtime variables like NODE_ENV (set to production) and PORT (3000). It also includes the placeholders ${NEXT_PUBLIC_SUPABASE_URL} and ${NEXT_PUBLIC_SITE_URL}. These placeholders are likely substituted by an external tool (like envsubst in a CI/CD pipeline) before the manifest is applied to the cluster.

3.7.2 Secrets (Secret)

- **File:** apiVersion: v1, kind: Secret, metadata.name: travel-agency-secrets
- **Purpose:** Stores sensitive information, such as API keys and tokens. Secrets are stored in the cluster as base64-encoded strings.
- **Details:** It contains the placeholders ${NEXT_PUBLIC_SUPABASE_ANON_KEY} and ${OPENROUTER_API_KEY}. The use of stringData allows you to provide the plaintext values, which Kubernetes will automatically encode into base64 upon creation. *Note: For production environments, it is recommended to manage these secrets using Jenkins credentials rather than storing them directly in Git.*

3.7.3 Application Life Cycle (Deployment)

- **File:** apiVersion: apps/v1, kind: Deployment, metadata.name: travel-agency-deployment
- **Purpose:** The primary controller responsible for creating and updating instances of your application.
- **Key Features:**
    - **Replicas:** It ensures that 2 identical Pods are running at all times for high availability.
    - **Image:** It uses the image 890iop/travel_agency with a placeholder tag ${IMAGE_TAG} (substituted during build time).
    - **Environment Variables:** It injects *all* variables defined in *both* the travel-agency-config ConfigMap and the travel-agency-secrets Secret directly into the container as environment variables.
    - **Resource Management:** Defines standard memory and CPU limits (256Mi/250m requests, 512Mi/500m limits) to prevent the application from consuming all node resources.

- **Health Checks:** Uses livenessProbe to restart the container if it stops responding (at /), and readinessProbe to ensure the application is fully started (at /) before the Service directs traffic to it.

3.7.4 Networking (Service)

- **File:** apiVersion: v1, kind: Service, metadata.name: travel-agency-service
- **Purpose:** Provides a stable network endpoint (IP address and port) to access the dynamic set of Pods managed by the Deployment.
- **Details:** It uses a LoadBalancer type. In a cloud environment (like AWS EKS), this will automatically provision an external cloud load balancer (ELB/ALB) and direct public internet traffic (listening on Port 80) to the application's Port 3000 inside the cluster. It selects the correct Pods using the label app: travel-agency.

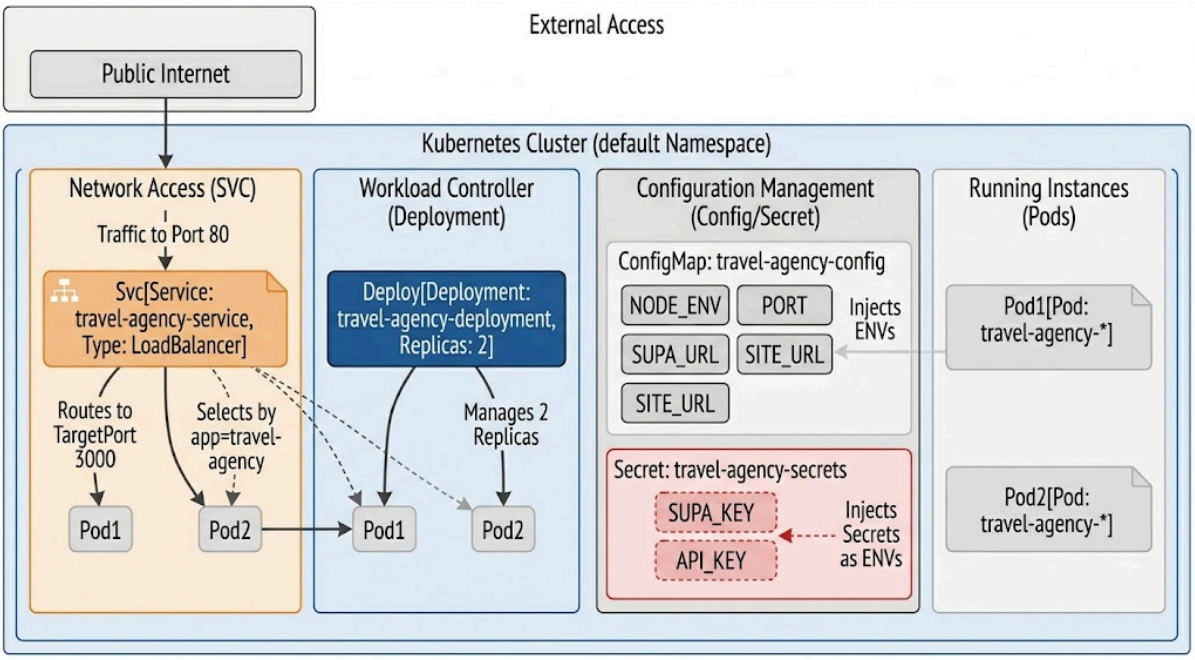


Figure 3.7.4 Kubernetes Cluster Setup

CHAPTER 4

INTEGRATING THE SUPABASE SERVER AND BUILD THE NECESSARY LOGIC FOR THE APPLICATION

4.1 The Purpose of Using Supabase

Supabase is a **Backend-as-a-Service (BaaS)** platform that gives developers an instant, production-ready backend built directly on **PostgreSQL**.

Primary Core Purposes

- **Relational Data Management:** Provides a fully managed PostgreSQL database with native support for complex joins, transactions, JSON queries, and schema migrations.
- **Auto-Generated APIs:** Automatically creates instant, secure RESTful and GraphQL APIs directly from your database schema without writing custom server code.
- **Built-in Authentication:** Handles user sign-up, session management, magic links, social logins (Google, GitHub, Apple), and Multi-Factor Authentication (MFA) out of the box.
- **Row-Level Security (RLS):** Integrates authentication with Postgres RLS policies, allowing fine-grained data access control directly inside the database tier.
- **Realtime Subscriptions:** Listens to database changes (inserts, updates, deletes) via WebSockets to power live features like chats, collaborative tools, and live dashboards.
- **File Storage:** Offers S3-compatible object storage for uploading and serving media (images, videos, documents) linked directly to database policies.
- **Vector Search for AI:** Native support for the pgvector extension allows developers to store vector embeddings for semantic search, retrieval-augmented generation (RAG), and AI agent memory in the same database as application data.
- **Serverless Logic:** Executes TypeScript/JavaScript serverless Edge Functions for background processing, payment webhooks, or third-party integrations.

Common Use Cases

Category	Typical Project Application
Rapid Prototyping & MVPs	Launching full-stack web/mobile apps without writing custom backend code.

Realtime Apps	Live messaging platforms, multiplayer collaboration tools, and live notification feeds.
AI & LLM Applications	Storing document embeddings for RAG pipelines alongside traditional structured user data.
Self-Hosted Infrastructure	Projects requiring full data sovereignty or specialized cloud setups via Docker deployment.

4.2 Create an AWS EC2 for Supabase Server

First, go to the ec2>instance. Then, select Launch instance and then give the necessary details as needed.

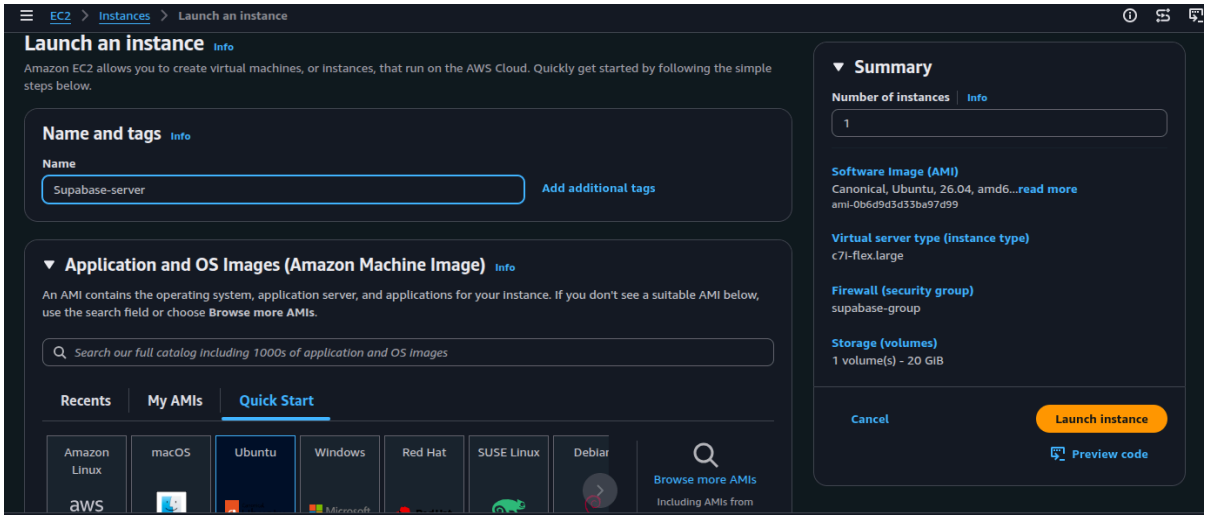


Figure 4.2 (a) Creating EC2 Server

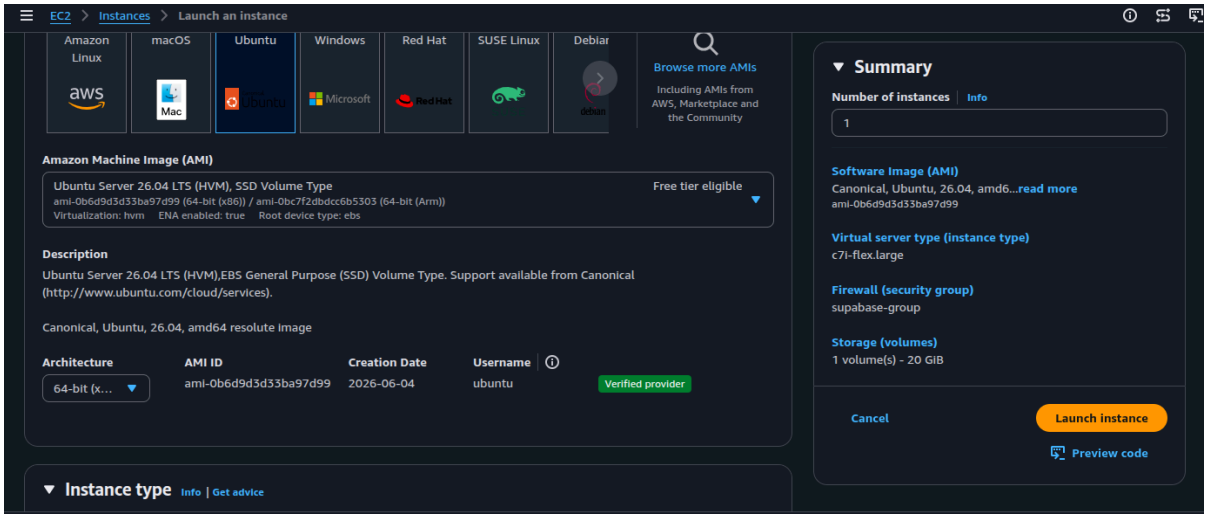


Figure 4.2 (b) Creating EC2 Server

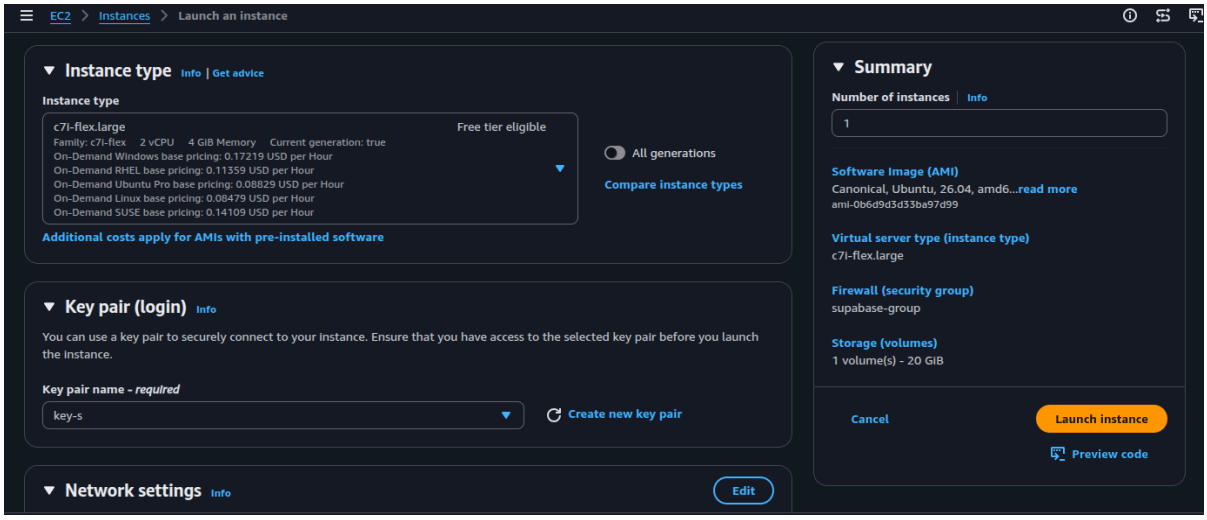


Figure 4.2 (c) Creating EC2 Server

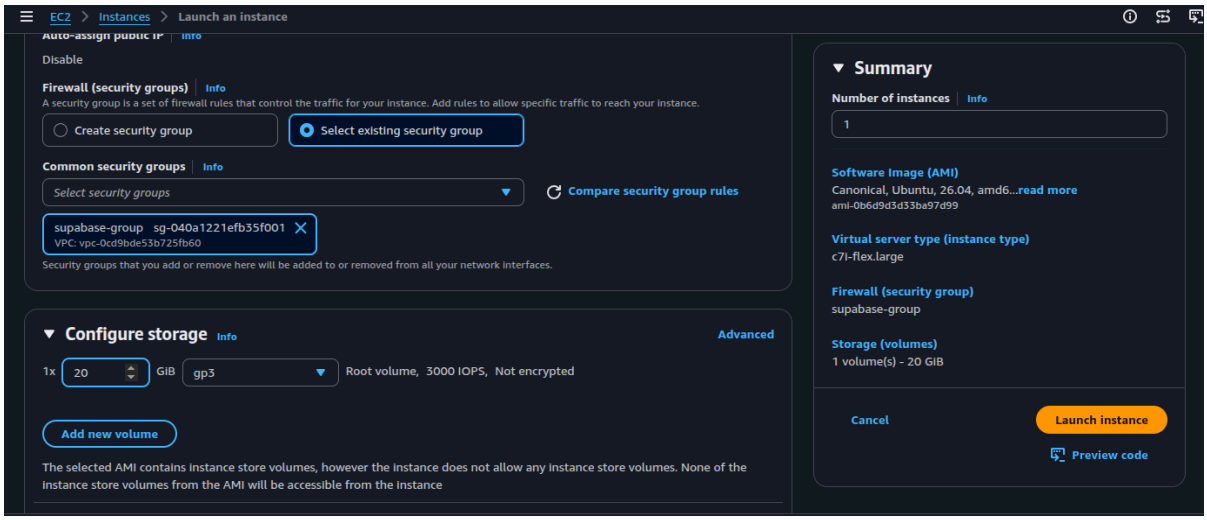


Figure 4.2 (d) Creating EC2 Server

4.3 Allocated an Elastic IP

The AWS ec2 instance suddenly changed its ip address. So, if this ec2 is allocated by an elastic ip then the ip address of this instance is not changed.
First go to ec2>Elastic Ip Address>Allocated Elastic Ip Address then click Allocate. Then click the recently created Elastic IP and then click Associate Elastic IP address, then select the supabase-server instance and click Associate.

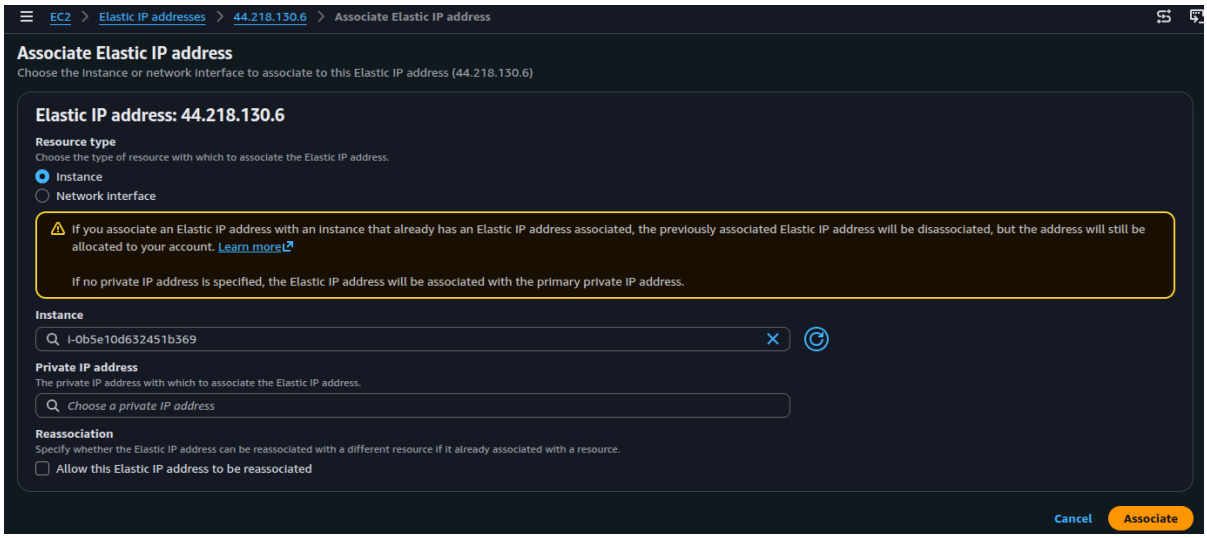


Figure 4.3 Associate The Supabase Server for Elastic IP address

4.4 Install Docker and Supabase throw the Docker Image in EC2 Server

Get access the ec2 server, they need to run this command,

```
$ ssh -i key-s.pem ubuntu@44.218.130.6
Welcome to Ubuntu 26.04 LTS (GNU/Linux 7.0.0-1011-aws x86_64)

 * Documentation:  https://docs.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Sat Aug 15 08:49:30 UTC 2026

  System load:  1.47              Temperature:           -273.1 C
  Usage of /:   71.4% of 18.25GB  Processes:               233
  Memory usage: 35%               Users logged in:         0
  Swap usage:   0%                IPv4 address for enp39s0: 172.31.1.161

 * Ubuntu Pro delivers the most comprehensive open source security and
```

```
   compliance features.

   https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Sat Aug 15 14:11:49 2026 from 103.80.2.149
ubuntu@ip-172-31-1-161:~$
```

Then, install docker and docker compose in this server

- **Set up Docker's official APT repository**

```
# Update package index and install prerequisites
sudo apt-get update
sudo apt-get install -y ca-certificates curl gnupg

# Add Docker's official GPG key
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Set up the repository source list
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

- **Install Docker Engine and Docker Compose Plugin**

```
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
```

- **Enable and start the Docker daemon**

```
sudo systemctl enable --now docker
```

- **Configure non-root user permissions (Optional but recommended)** Allow running docker commands without prepending sudo:

```
# Add your user to the docker group
sudo usermod -aG docker $USER

# Apply group changes to current session
newgrp docker
```

Fetch the Official Supabase Configuration

Clone the Supabase Git repository to pull the Docker configuration files:

```
# Clone only the latest commit to save space
git clone --depth 1 https://github.com/supabase/supabase

# Navigate to the Docker setup folder
cd supabase/docker

# Create your active environment file from the example template
cp .env.example .env
```

4.5 Setup the .env file and Run the Server

Before starting the containers, update .env to secure your setup:

1. Open .env in your editor (nano .env).
2. Update key secrets:
    - **POSTGRES_PASSWORD**: Set a strong database password.
    - **JWT_SECRET**: Replace with a secure random string (minimum 32 characters).
    - **ANON_KEY** and **SERVICE_ROLE_KEY**: Generate new JWTs signed with your JWT_SECRET for production.
    - **SITE_URL** & **API_EXTERNAL_URL**: Change localhost to your server's domain or public IP address.

3. Launch the Stack

Pull the official Docker images and start the services in detached mode:

```
# Pull container images
docker compose pull

# Start all Supabase services
docker compose up -d
```

4. Access Supabase

Once the deployment finishes and services status becomes healthy:

- **Supabase Studio (Dashboard):** Open http://44.218.130.6:8000 in your browser.
- **Default Credentials:** Username: supabase, Password: this_password_is_insecure_change_it (you can update DASHBOARD_USERNAME and DASHBOARD_PASSWORD in .env).

4.6 Supabase(PostgreSQL) Query

This SQL script sets up a production-ready relational database schema for **Floraflyers**, a travel and tour booking web application with an integrated AI chat assistant. Built specifically for **Supabase (PostgreSQL)**, it handles user profiles, tour catalogs, bookings, payments, reviews, and AI conversation histories while enforcing database-level security and automated data flows.

4.6.1 Extension & Setup

- **create extension if not exists "pgcrypto";** Activates cryptographic functions inside Postgres, allowing the engine to generate cryptographically secure Version 4 UUIDs using gen_random_uuid().

4.6.2 Profiles Table & Auth Integration

- **public.profiles**: Maps directly to Supabase's internal authentication schema (auth.users) via a 1:1 foreign key.
- **RLS Policies**: Restricts access so users can only SELECT and UPDATE their own profile (auth.uid() = id).
- **Automated User Synchronization (handle_new_user)**: A PL/pgSQL function paired with a database trigger (on_auth_user_created). Whenever a new user registers via Supabase Auth (auth.users), Postgres automatically inserts a matching row into public.profiles and extracts their full_name from metadata.

4.6.3 Categories & Tours

- **public.categories**: Stores tour classifications (e.g., Beach, Mountain). Uses unique constraints on name and URL-friendly slug.
- **public.tours**: The core catalog containing pricing, duration, ratings, and rich payload arrays stored as JSON (gallery, itinerary, includes).
    - Includes domain integrity constraints via CHECK clauses (e.g., days > 0, rating between 0 and 5, match_score between 0 and 100).
- **RLS Policies**: Unauthenticated visitors and authenticated users alike can view published tours (is_published = true).

4.6.4 Bookings Engine

- **public.bookings**: Tracks reservations made by users for specific tours.
- **State Management**: Enforces status transitions via a CHECK constraint restricting values to 'pending', 'confirmed', 'cancelled', or 'completed'.
- **RLS Policies**:
    - **View & Insert**: Users can only see or insert bookings where user_id matches their authenticated session ID (auth.uid()).
    - **Cancellation**: Users can only modify a booking if it belongs to them **and** its current state is 'pending'.

4.6.5 Payments Processing

- **public.payments**: Manages transactions linked to specific bookings.
- **Supported Payment Methods**: Validates payment methods ('card', 'mobile_banking') and payment lifecycle statuses.
- **Subquery RLS Security**: Users can view or create payment entries *only* if a subquery confirms the parent booking (booking_id) belongs to their authenticated user ID (auth.uid()).

4.6.6 Reviews & Ratings System

- **public.reviews**: Stores user feedback for tours.
- **Unique Constraint (tour_id, user_id)**: Prevents duplicate reviews by ensuring a single user can only leave one review per tour.
- **RLS Policies**: Reviews are publicly viewable by anyone, but only the creator can post or edit their review.

4.6.7 AI Chat Engine (chat_conversations & chat_messages)

- **Stateful AI Sessions**: Provides structure for an LLM/agentic chat interface.

- **chat_conversations**: Tracks distinct user threads.
- **chat_messages**: Stores message exchanges within a conversation. Roles are constrained to 'user' or 'ai'. The metadata JSONB column can store token counts, tool execution calls, or agent execution states.

- **Subquery Security**: Message visibility and insertion policies verify session ownership by checking the parent conversation's user_id.

4.6.8 Database Performance Indexes

Explicit B-Tree indexes created on foreign key columns (idx_tours_category, idx_bookings_user, idx_chat_messages_conversation, etc.) optimize foreign key lookups and standard JOIN operations, preventing full-table scans during user queries.

4.6.9 Seed Data Initialization

Populates default records into categories and tours using ON CONFLICT (slug) DO NOTHING, ensuring initial catalog data exists while preventing duplicate insertion errors when executing scripts repeatedly.

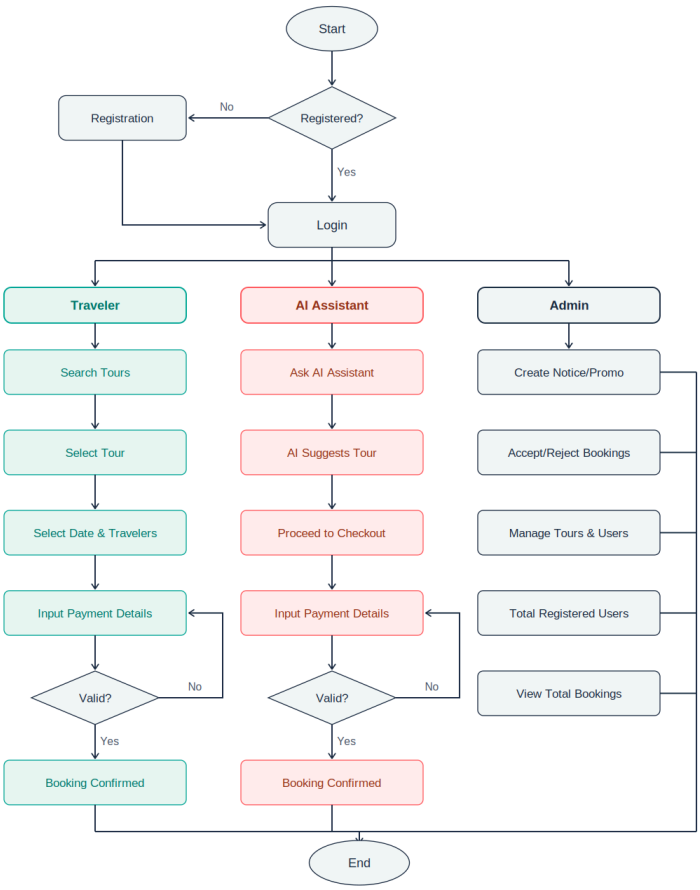


Figure 4.6.9 (a) Flowchart Of The Database

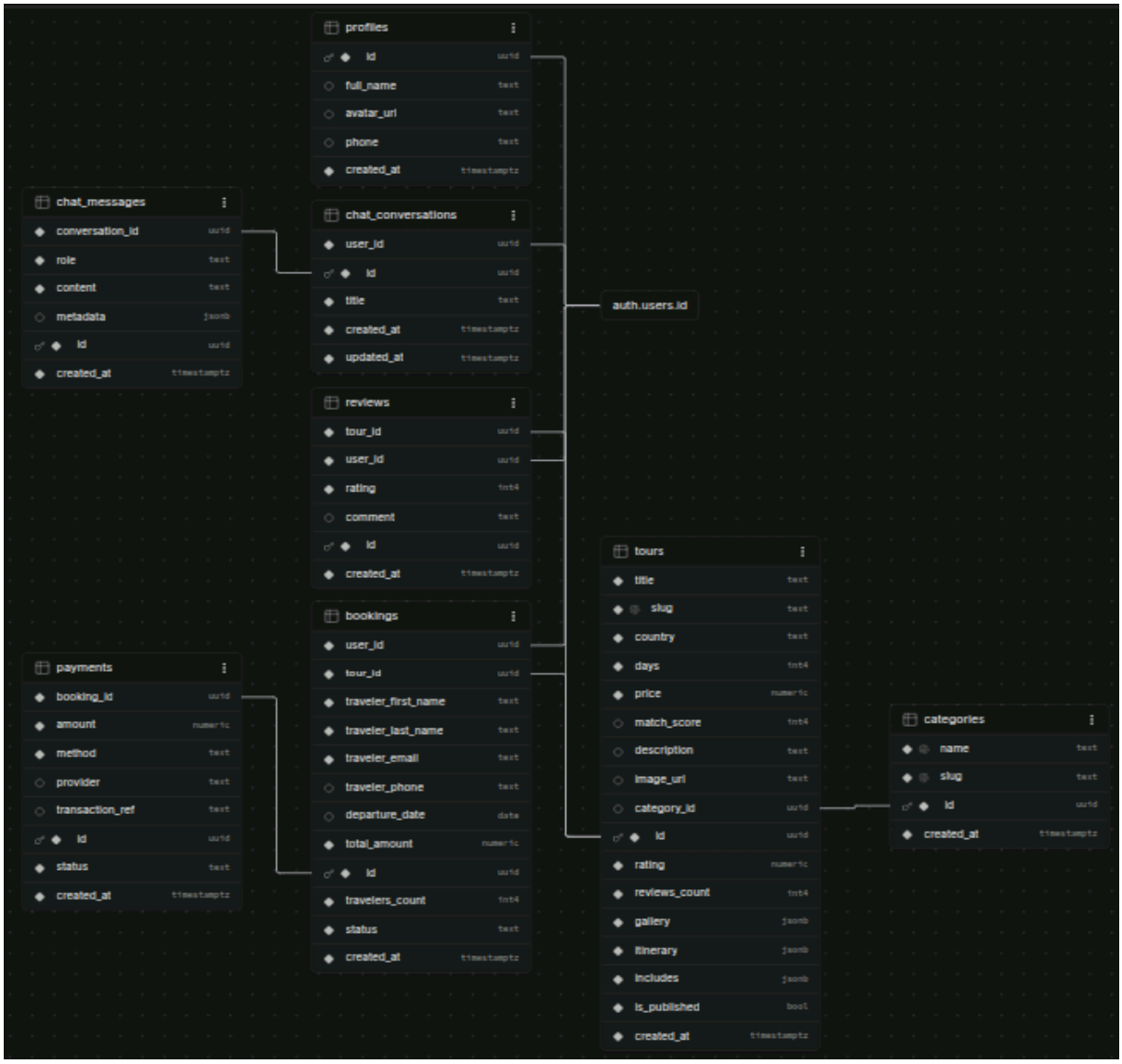


Figure 4.6.9 (b) Database Schema Visualization

CHAPTER 5

DESIGN THE UI/UX AND DEVELOP FRONTEND SERVER WITH NEXTJS

5.1 Design UI/UX and Make a Design file

At first, make a design file for frontend design,

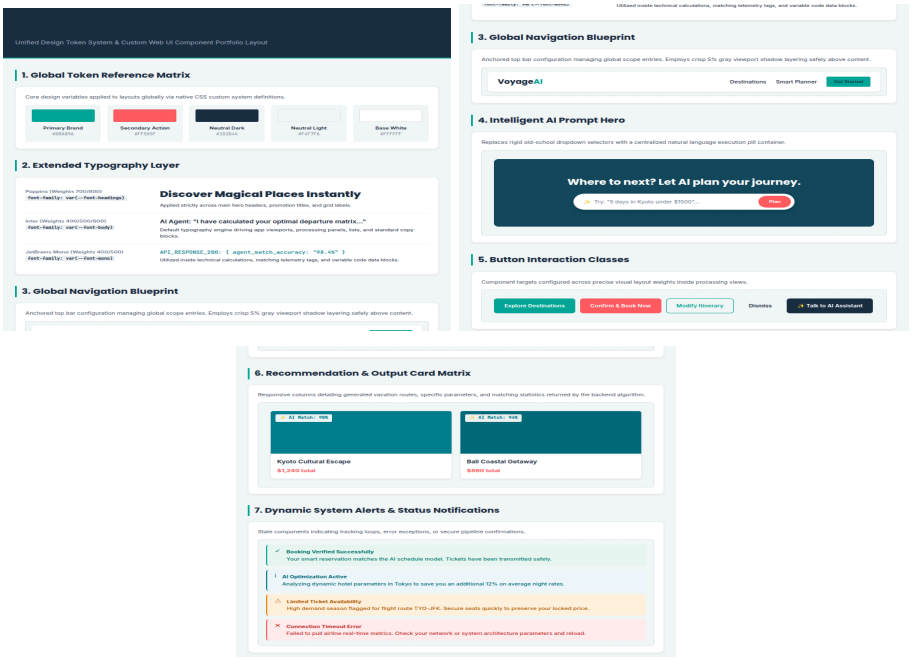


Figure 5.1 (a) Design File

Now, design 4 pages for the frontend application. Those 4 pages design are given below,

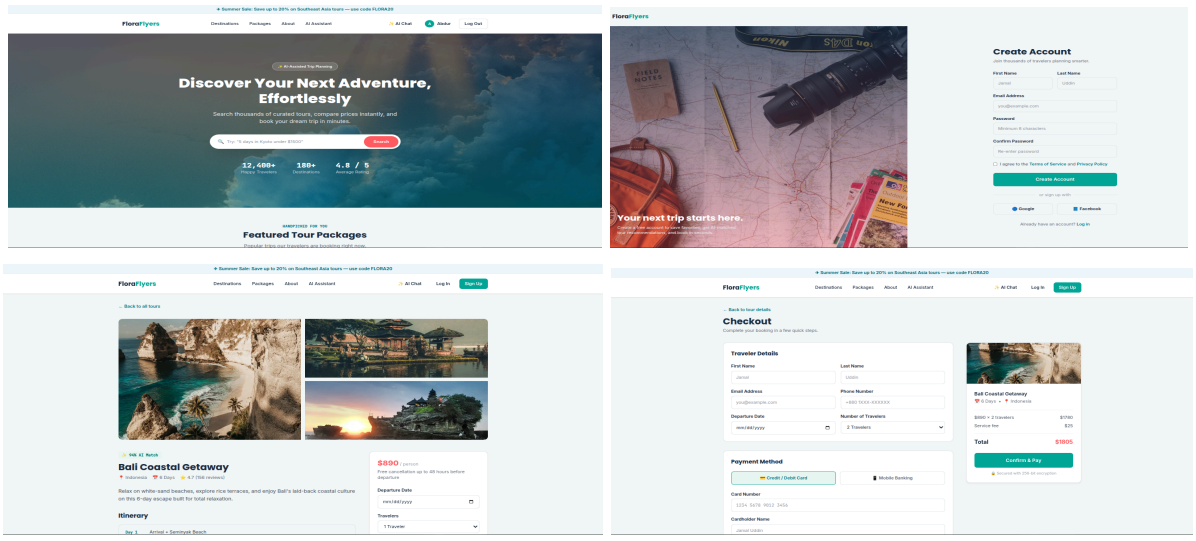


Figure 5.1 (b) UI/UX Design For The Frontend Application

5.2 Initialize the Nextjs App

As planned, creating a Nextjs app with tailwind CSS. The process of the creating project on terminal is,

```
$ npx create-next-app@latest
Need to install the following packages:
create-next-app@16.3.3
Ok to proceed? (y)

✔ What is your project named? ... flora-flyers
✔ Would you like to use the recommended Next.js defaults? > No, reuse previous settings
Creating a new Next.js app in /home/…/flora-flyers.

Using npm.

Initializing project with template: app-tw


Installing dependencies:
- next
- react
- react-dom

Installing devDependencies:
- @tailwindcss/postcss
- eslint
- eslint-config-next
- tailwindcss

npm warn deprecated eslint@9.39.5: This version is no longer supported. Please see
https://eslint.org/version-support for other options.

added 433 packages, and audited 434 packages in 1m

182 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

Generating route types...
```

```
✓ Types generated successfully

Initialized a git repository.

Success! Created flora-flyers at /home/abdur-rahman-saad/Desktop/webdev/flora-flyers
```

Then, for supabase attach necessary package is,

```
npm install
npm install @supabase/supabase-js @supabase/ssr
cp .env.local.example .env.local
```

Now set the .env file to attach the supabase server,

```
NEXT_PUBLIC_SUPABASE_URL=https://xxxxx.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=eyJ...
```

5.3 Sitemap for the Application

The site is organized around the following pages:

- Home — hero section, search bar, featured tour packages, category filter, testimonials.
- Login / Signup — authentication forms backed by Supabase Auth.
- Tour Details — a dynamic route displaying a single tour's gallery, itinerary, and inclusions.
- AI Assistant (Chat) — a conversational interface with persisted history for logged-in users.
- Checkout — traveler details, payment method selection, and order summary.
- Error, Loading, and Not-Found pages — Next.js framework-level fallback pages for a consistent experience under failure or navigation-miss conditions.

5.4 Authentication Module

User registration and login are implemented using Supabase Auth's email/password provider. On sign-up, the traveler's full name is stored in Supabase Auth's user metadata; a database trigger (handle_new_user) automatically creates a corresponding row in the profiles table. The navigation bar is implemented as a client component that subscribes to Supabase's onAuthStateChange event, so the interface updates immediately between a “Log In / Sign Up” state and an authenticated state showing the user's name and a log-out action, without requiring a page reload.

5.5 Tour Browsing and Details Module

The home page is implemented as a Next.js server component that queries the tours and categories tables directly from the server before the page is sent to the browser, improving initial load performance and avoiding an extra client-side round trip. Category filtering is delegated to a small client component so that it can respond instantly to user interaction without a further server request. The tour details page uses a dynamic route (tours/[id]) to fetch and display a single tour's gallery, day-by-day itinerary, and inclusions.

5.6 Booking and Checkout Module

The checkout page accepts a tour identifier via a query parameter, requires the traveler to be authenticated, and on submission inserts a row into the bookings table followed by a corresponding row into the payments table, both scoped to the authenticated user by Row Level Security. The payment step in the current implementation is a simulated form (no real payment gateway is connected); its purpose is to demonstrate the end-to-end data flow that a production payment integration would ultimately hook into.

5.7 AI Assistant Module

The AI assistant is implemented as a Next.js Route Handler (app/api/chat) that keeps the OpenRouter API key server-side. On each request, the route first queries the tours table for all published tours and constructs a system prompt that lists each tour's title, country, duration, price, rating, and description, explicitly instructing the model to recommend only from this list. The route then forwards the conversation, together with this grounded system prompt, to OpenRouter's chat completions endpoint using the google/gemma-4-26b-a4b-it:free model, with two additional free models configured as automatic fallbacks in case the primary model is temporarily rate-limited by its upstream provider.

A key design decision was giving the AI assistant the ability to trigger a real application action rather than only producing text. The system prompt instructs the model that, once the traveler has clearly settled on a specific tour, it should end its reply with a hidden marker of the form [BOOK:<tour-id>], using only real identifiers supplied in the grounding data. The server-side route parses this marker out of the model's response, validates the referenced ID against the actual tours table (so the model cannot fabricate a booking target), strips the marker from the visible text, and returns the validated tour ID to the frontend. The chat interface then renders a genuine “Proceed to Payment” button linking directly to the checkout page for that tour. For logged-in users, each conversation and its messages — including any

booking marker metadata — are persisted to the chat_conversations and chat_messages tables, allowing a past conversation to be reopened with its payment button intact.

5.8 Security Implementation

Authorization is enforced primarily at the database layer through Row Level Security. Public-read tables (tours, categories) allow unauthenticated SELECT queries; user-owned tables (bookings, payments, reviews, chat_conversations, chat_messages) restrict SELECT, INSERT, and UPDATE operations to rows where the relevant user_id column matches the currently authenticated user's ID, evaluated via Postgres's auth.uid() function. During development, a gap was identified and corrected in which the payments table had a SELECT policy but no corresponding INSERT policy, which would have silently blocked the checkout flow's payment record from ever being created; this is discussed further in Chapter 6. Separately, the OpenRouter API key used by the AI assistant is read only inside the server-side Route Handler and is never sent to or exposed in the browser.

CHAPTER 6

TESTING AND MONITORING

6.1 Testing Approach

Testing was carried out manually and incrementally as each sprint's features were completed, rather than as a single end-of-project pass. Each new page or module was exercised directly in the running application (npm run dev) immediately after implementation, and issues were fixed before moving to the next task. This approach is consistent with the Agile Scrum methodology adopted for the project, where each sprint is expected to end in a working, testable increment.

Test case:

ID	Test Case	Expected Result	Status
TC-01	Register with a new email/password	Account created; profile row auto-created	Pass
TC-02	Log in with valid credentials	Session established; navbar shows user name	Pass
TC-03	Log in with invalid credentials	Error message displayed; no session created	Pass
TC-04	Browse tours on the home page	Published tours load from Supabase and render	Pass
TC-05	Filter tours by category	Grid updates instantly to matching tours only	Pass
TC-06	Attempt checkout while logged out	Redirected to login before booking is created	Pass
TC-07	Complete checkout while logged in	Booking and payment rows created in database	Pass (after RLS fix)
TC-08	Ask AI assistant for a tour recommendation	Reply grounded in real, published tours only	Pass
TC-09	Confirm a tour via chat	“Proceed to Payment” button links to correct checkout page	Pass
TC-10	AI provider rate-limited (simulated)	Friendly error shown with a working Retry action	Pass

6.2 Monitoring with Prometheus and Grafana

In the server, prometheus and grafana are installed according to the ansible server. Now, just open the server in the port of 3001. The 3001 port for grafana. 3000 port is resub for Nextjs application. After entering the grafana interface, the login with default username and password as admin.

6.2.1 Connect Datasource

Go to Connection>Data sources and select prometheus.

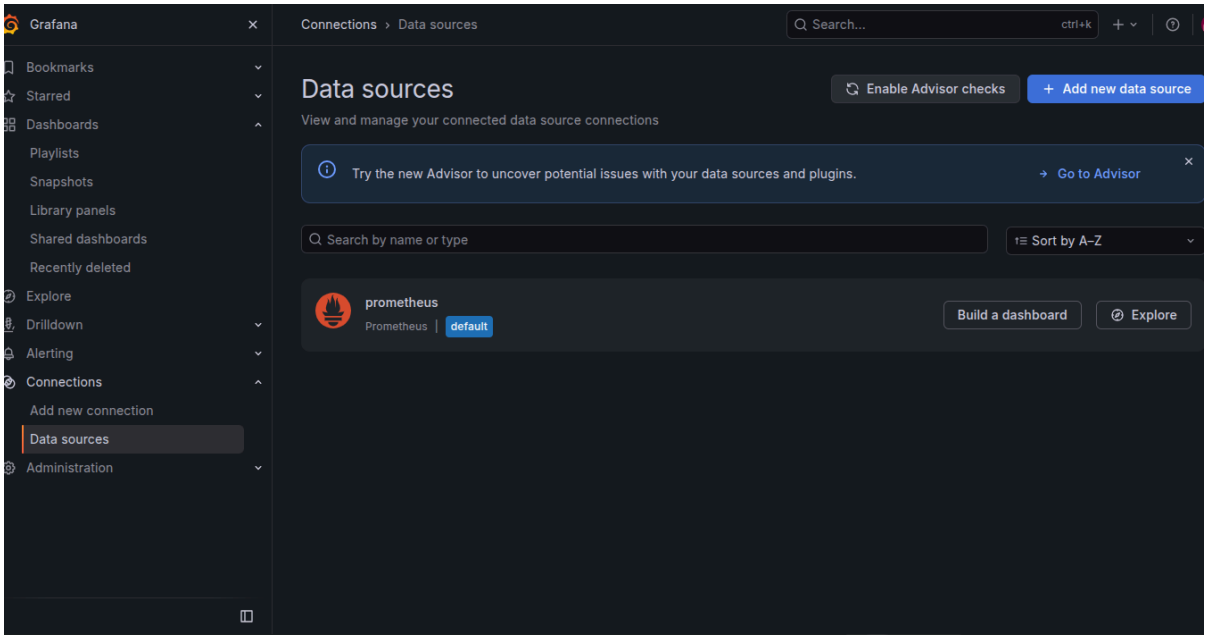


Figure 6.2.1 (a) Connect With Datasource

Then give the url of prometheus on the Prometheus server URL segment. Then click the save and test button.

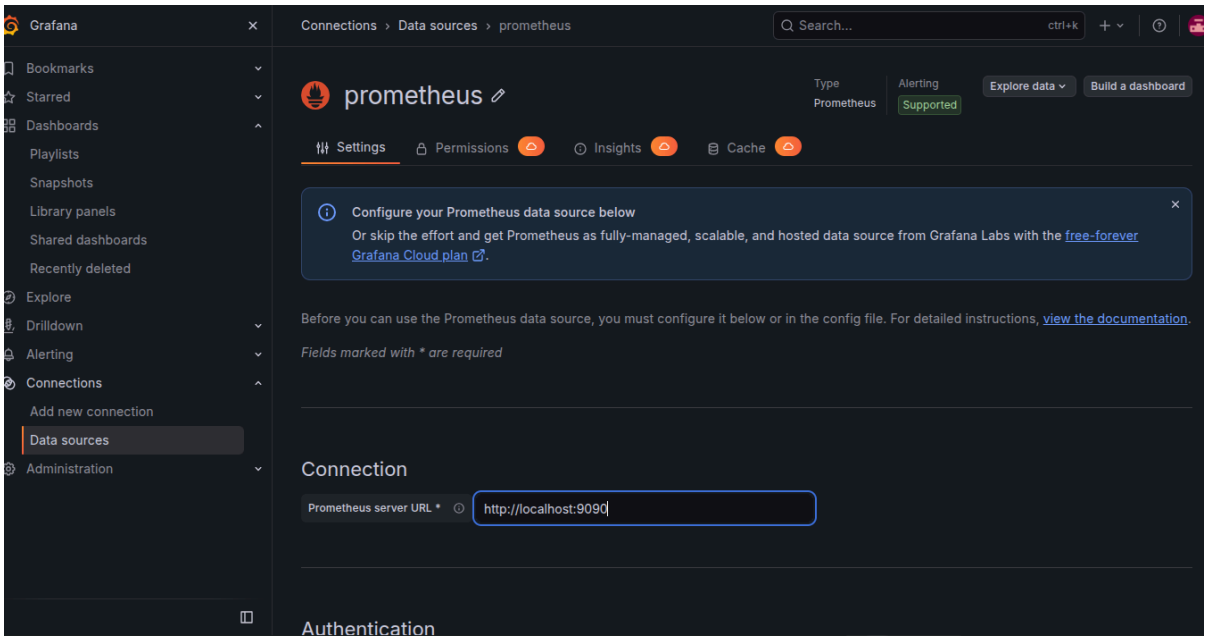


Figure 6.2.1 (b) Connect With Datasource

6.2.2 Import a Dashboard

Go to the Dashboard, then click import dashboard. Then load the 1860 dashboard for my project and then click the Import button.

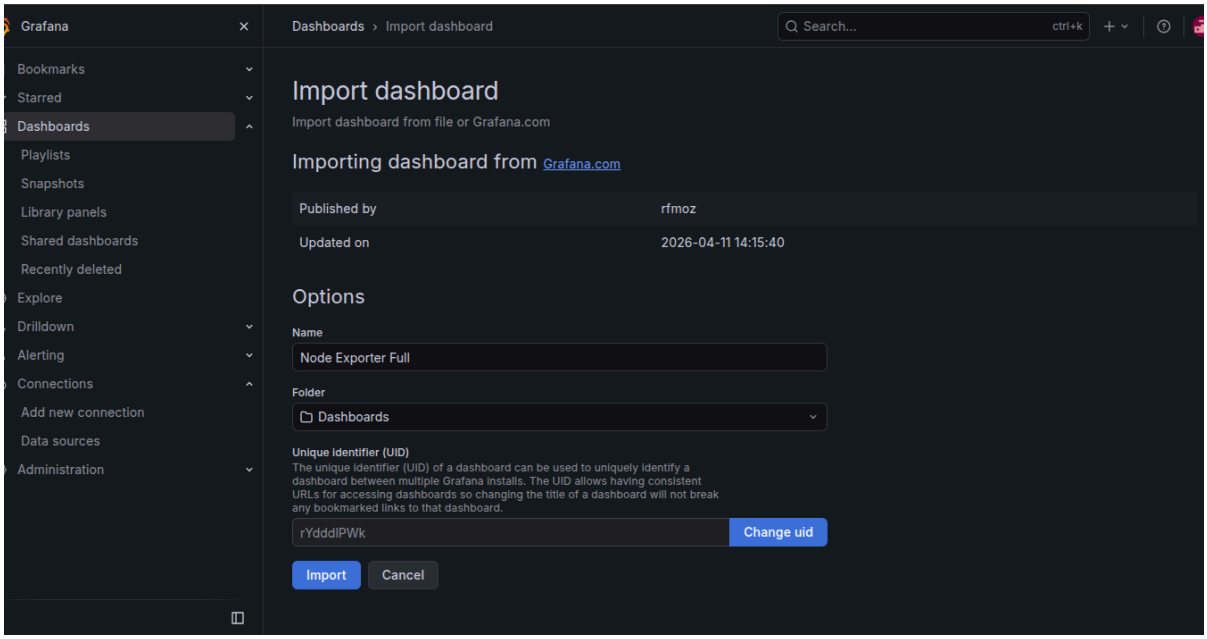


Figure 6.2.2 Import The Dashboard

6.2.3 Monitoring the Server from the Dashboard

After importing the dashboard, grafana fetches data from prometheus and previews that data by this dashboard.

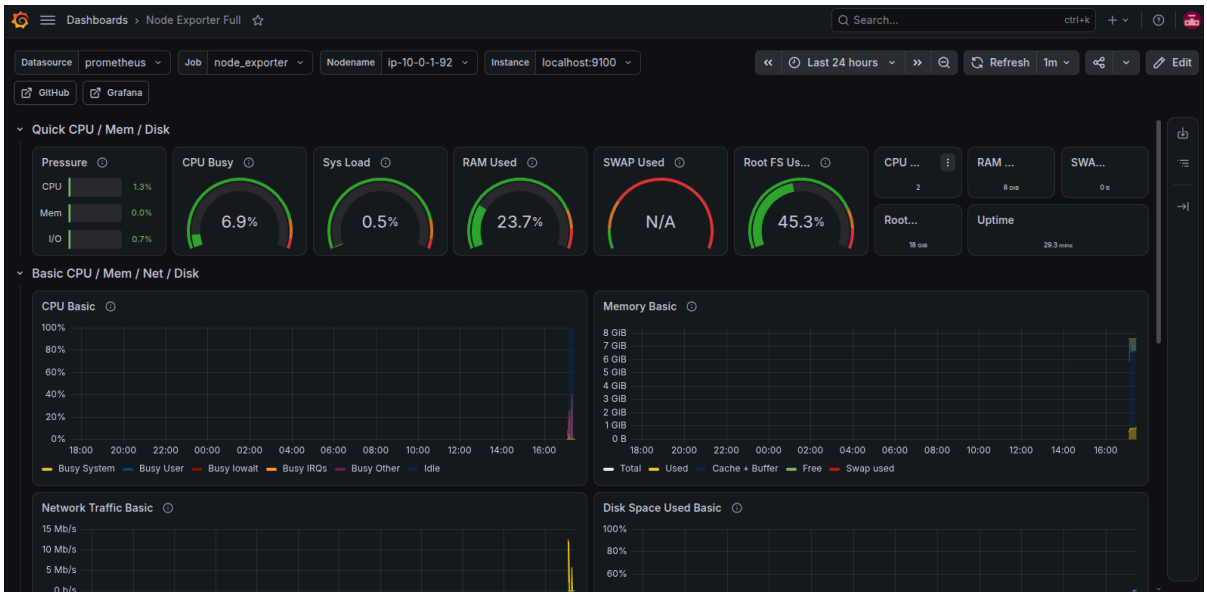


Figure 6.2.3 Monitoring The Server Health

CHAPTER 7

CONCLUSION

7.1 Future Work

- Integrate a real payment gateway and an admin panel for fully functional this application.
- Integrate email/slack notification system in prometheus to notify the server state.
- Introduce automated testing to complement the manual testing process used during initial development.
- Extend the AI assistant with function-calling for structured actions beyond checkout handoff, such as filtering results or checking booking status conversationally.

7.2 Conclusion

This project set out to design and implement a secure, AI-assisted tourist agency web application, and, over five Agile Scrum sprints, delivered a working system covering authentication, tour browsing, booking and checkout, and a grounded conversational AI assistant capable of taking real action within the application. The project demonstrates a practical, end-to-end integration of a modern frontend framework, a Backend-as-a-Service with database-level authorization, and a third-party large language model, and illustrates how careful prompt design and server-side validation can make an LLM-based feature both useful and trustworthy within a small-scale production-style application.

REFERENCES

[1] El Aouni, F., Moumane, K., Idri, A., Najib, M., & Jan, S. U. (2025). A systematic literature review on Agile, Cloud, and DevOps integration: Challenges, benefits. Information and Software Technology, 177, 107569.

[2] Al-Saqqa, S., Sawalha, S., & Abdelnabi, H. (2020). Agile software development: Methodologies and trends. *iJIM*, *14*(11), 246-70.

[3] Azad, N., & Hyrynsalmi, S. (2023). DevOps critical success factors—A systematic literature review. *Information and Software Technology*, *157*, 107150.

[4] Bariah, S. H., Imania, K. A. N., Nasruloh, I., Purwanti, Y., Rahadian, D., & Ranti, S. S. (2025, July). Enhancing backend development efficiency with Supabase. In *AIP Conference Proceedings* (Vol. 3141, No. 1, p. 040007). AIP Publishing LLC.

[5] Shylesh, S. (2017, April). A study of software development life cycle process models. In *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences* (pp. 534-541).

[6] Hema, V., Thota, S., Naresh Kumar, S., Padmaja, C., Rama Krishna, C. B., & Mahender, K. (2020, December). Scrum: An effective software development agile tool. In *IOP Conference Series: Materials Science and Engineering* (Vol. 981, No. 2, p. 022060). IOP Publishing.