personal professional project report

IDP-X

next-generation Intelligent Internal Developer Platform

Bzeouich Naoures
Mejdi Omar

supervisor : Ms. Saloua BEN YAHIA

2025-06-03

Contents

1. Introduction

In this introductory part, we will explore how DevOps has evolved in the era of Artificial Intelligence. Through a concise and comprehensive overview, we will examine what DevOps is, how AI is transforming its processes, and why intelligent automation is becoming essential. This section also highlights the challenges developers face today, the limitations of current platforms, and the motivation behind building a smarter, AI-driven developer platform.

1.1 DevOps in the Era of Artificial Intelligence

In this section, we delve into the foundational concepts of DevOps and examine how Artificial Intelligence is reshaping its landscape.
We begin by defining DevOps and its core principles, then explore the growing role of AI in optimizing DevOps workflows. Finally, we discuss the need for intelligent automation to overcome the limitations of traditional pipelines and meet the demands of modern software delivery.

1.1.1 What is DevOps?

DevOps is a combination of practices, tools, and cultural philosophies that aim to bridge the gap between software development and IT operations. Its core objective is to shorten the system development life cycle while delivering high-quality software continuously. DevOps promotes collaboration between development and operations teams, automated workflows, continuous integration/continuous delivery - deployment (CI/CD), and real-time monitoring and feedback loops.

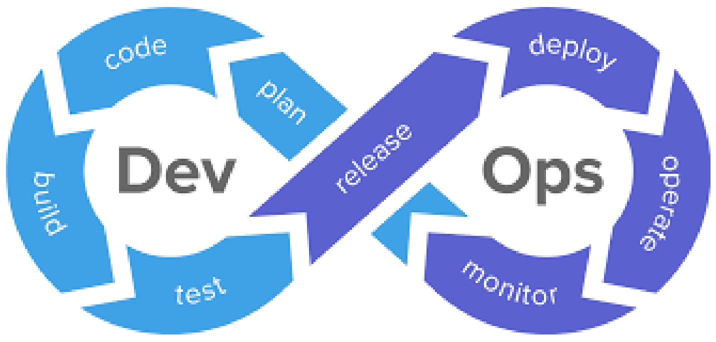


Figure 1.1: devops logo

Figure 1.1 : devops logo illustrates the DevOps lifecycle, highlighting the workflow stages: plan, code, build, test, release, deploy, operate, and monitor. It emphasizes the seamless collaboration between development and operations teams, enabling faster, more reliable, and iterative software delivery.

Building upon this foundational model, the traditional DevOps pipeline consists of distinct stages such as planning, development, testing, deployment, and monitoring. Tools like Git, Jenkins, Docker, Kubernetes, Ansible, and Terraform have become integral to automating both infrastructure and application lifecycles. However, as modern systems increase in complexity and the demand for rapid, error-free deployment grows, traditional DevOps practices begin to face challenges in terms of speed, scalability, and accuracy.

1.1.2 How AI is transforming DevOps

Artificial Intelligence (AI) has rapidly become a cornerstone of innovation across nearly every industry — from healthcare and finance to manufacturing, automotive, and education. Its ability to process massive amounts of data, learn from patterns, and make intelligent decisions has made it indispensable in a world that demands speed, personalization, and automation.
In recent years, AI has taken a central role in automated systems, such as self-driving cars, predictive maintenance in manufacturing, and intelligent personal assistants. This growing presence has naturally extended to software engineering and IT operations, where complexity, scale, and the need for rapid response make AI not just useful, but essential.

Within the context of DevOps, the integration of AI — often referred to as AIOps (Artificial Intelligence for IT Operations) — brings a new level of intelligence and automation to development and operations pipelines. Traditional DevOps, while powerful, often relies on predefined rules, static monitoring, and human decision-making. AI, by contrast, introduces adaptive, predictive, and autonomous capabilities.

Here are some of the key ways AI is transforming DevOps:

- **Automated Anomaly Detection:** AI can continuously analyze logs, metrics, and system behaviors to identify anomalies or performance regressions in real-time, without requiring manual thresholds.
- **Predictive Maintenance and Failure Forecasting:** Machine learning models can predict system failures, resource bottlenecks, or capacity issues before they happen.
- **Optimized Infrastructure Configuration:** AI algorithms can analyze usage patterns to suggest (or automatically apply) optimal infrastructure settings for performance, cost, and scalability.
- **CI/CD Pipeline Optimization:** AI can analyze past deployments and recommend improvements to CI/CD workflows — such as adjusting the order of tests, reducing redundant steps, or parallelizing jobs — to speed up delivery and reduce errors.
- **Smart Debugging and Deployment Decisions:** AI-powered assistants can analyze code changes, recent incidents, and system health to suggest safer deployment strategies or highlight risky commits.
- **Intelligent ChatOps and Assistants:** Integrated with communication tools, AI assistants can help developers query logs, roll back deployments, or get environment summaries using natural language — significantly reducing the learning curve and improving responsiveness.

By embedding AI into the DevOps lifecycle, organizations unlock higher automation, better insights, faster recovery, and smarter decision-making. This shift transforms DevOps from a rule-based, reactive approach into a data-driven, proactive, and adaptive system.

1.1.3 The need for intelligent automation

While DevOps has significantly streamlined software delivery and deployment, its adoption still presents challenges, particularly for individual developers and small teams. Setting up CI/CD pipelines, provisioning cloud infrastructure, managing environments, and maintaining deployment workflows often demand deep technical expertise and time-consuming manual effort.
Moreover, developers frequently encounter repetitive and error-prone tasks — such as writing infrastructure-as-code templates, debugging deployment scripts, or aligning environments — that divert attention from core development work.

This is where intelligent automation becomes essential. By embedding AI-driven capabilities into automation tools, we can reduce cognitive load, eliminate redundancy, and minimize the risks associated with manual configuration.

Key benefits of intelligent automation include:

- **Auto-generation of Infrastructure-as-Code (IaC):** Automatically creating templates for tools like Terraform or Ansible based on high-level user inputs or detected project structure.

- **Context-aware Architecture Recommendations:** Suggesting the most suitable infrastructure or microservices layout depending on the application's type, language, and scalability needs.
- **Dynamic Workflow Adaptation:** Adjusting build and deployment pipelines in real-time based on historical behavior, performance metrics, or system changes.
- **Reduced Human Error:** Minimizing misconfigurations and increasing reliability by validating infrastructure setups and deployment strategies using intelligent agents.

The future of DevOps lies in the fusion of automation and artificial intelligence — not just to automate tasks, but to understand, anticipate, and improve the DevOps lifecycle. Intelligent automation enables faster onboarding, reduces operational overhead, and empowers developers to focus on innovation rather than infrastructure.

1.2 Motivation and Problem Statement

This section outlines the key challenges that modern developers face when working with infrastructure and deployment workflows. We highlight the limitations of existing platforms like Railway, Vercel and heroku, which, while user-friendly, lack flexibility and deep integration with DevOps ecosystems. These limitations motivate the need for a smarter, AI-assisted Intelligent Developer Platform (IDP) that can bridge the gap between simplicity and full-stack control.

1.2.1 Developer challenges with infrastructure

In today's software developers are expected to go beyond writing code — they are now responsible for managing the full lifecycle of their applications, including deployment, infrastructure provisioning, and operational maintenance. This growing expectation can introduce significant friction and complexity into the development process.

Common challenges developers face include:

- **Creating and Managing Cloud Environments:** Choosing the right cloud provider, provisioning resources, and ensuring scalability, security, and availability are no longer optional tasks—they are now essential responsibilities.
- **Setting Up and Maintaining CI/CD Pipelines:** Developers must configure and maintain automated pipelines for building, testing, and deploying code. This requires deep knowledge of tools like GitHub Actions, Jenkins, or GitLab CI.
- **Handling Multiple Environments:** Managing separate development, testing, staging, and production environments introduces overhead in maintaining consistency and ensuring smooth rollouts and rollbacks.
- **Writing and Managing Configuration Files:** Tools like Docker, Kubernetes, Ansible, and Terraform require accurate and optimized configuration files, which can be overwhelming for developers without infrastructure experience.

These tasks are often:

- **Time-consuming:** They slow down the development cycle, particularly for small teams or solo developers.
- **Error-prone:** Manual configurations frequently lead to mistakes, misalignments, or security gaps.
- **Distracting:** Infrastructure responsibilities take valuable time and focus away from writing business logic and building product features.
- **Writing and Managing Configuration Files:** Tools like Docker, Kubernetes, Ansible, and Terraform require accurate and optimized configuration files, which can be overwhelming for developers without infrastructure experience.

As applications grow in complexity, these pain points become more pronounced, highlighting the need for an intelligent, developer-friendly platform that simplifies infrastructure management and removes the burden of repetitive DevOps tasks.

1.2.2 Gaps in existing platforms

Platforms like Vercel, Heroku and Railway have become popular for their ability to streamline deployment workflows and abstract away infrastructure complexity. They offer seamless integration with Git-based repositories, automated deployments, and excellent developer experience out of the box.

However, despite their strengths, these platforms present several limitations — especially for teams with more advanced needs or enterprise — level requirements:

- **Limited Customization and Control:** While Vercel and Railway are ideal for rapid prototyping and small to mid-scale applications, they often lack fine-grained control over infrastructure components, networking, security policies, and environment configurations.
- **No Infrastructure as Code (IaC):** These platforms do not expose the underlying infrastructure through code, making it difficult to manage environments declaratively or maintain reproducibility across teams and projects.
- **Restricted to Single-Cloud Paradigms:** Multi-cloud or hybrid cloud strategies are not natively supported. This limits flexibility for organizations that wish to distribute workloads across AWS, Azure, GCP, or on-premise environments.
- **Minimal AI and Intelligence Integration:** Current platforms lack intelligent tooling to assist with infrastructure decisions, CI/CD optimization, or proactive system diagnostics using AI or machine learning.
- **Usage Limits and Pricing Constraints:** While both platforms - Heroku and Railway - offer free tiers, they are limited in build minutes, storage, environments, and team collaboration features. Scaling beyond basic usage often requires moving to paid plans that may not offer proportional value or flexibility.

These gaps highlight the need for a smarter, more flexible DevOps platform—one that combines ease of use with infrastructure transparency, AI-enhanced workflows, and seamless integration with modern DevOps ecosystems.

1.2.3 Why build an intelligent IDP

The emergence of cloud-native development, rapid deployment cycles, and the growing complexity of infrastructure have revealed the limitations of traditional DevOps tools and simplified platforms. There is a clear need for a next-generation Intelligent Developer Platform (IDP) that not only abstracts complexity but also augments the developer experience through intelligence and automation.

A Intelligent IDP can address the existing gaps in current platforms by:

- **Empowering Developers Without Requiring Deep DevOps Expertise:** By automating infrastructure provisioning, pipeline setup, and environment management, developers can focus on delivering code without needing advanced knowledge of tools like Terraform, Kubernetes, or Ansible.
- **Delivering AI-Driven Recommendations:** The integration of artificial intelligence enables the platform to offer context-aware guidance on environment configuration, architecture choices, and optimization strategies — reducing trial-and-error cycles.
- **Integrating Seamlessly with Git Repositories:** Users can either create new projects or connect existing GitHub repositories, enabling continuous deployment pipelines to be generated and managed automatically.
- **Auto-Generating Infrastructure Based on Tech Stack:** By selecting a technology stack (e.g., Node.js with PostgreSQL on AWS), the IDP can generate infrastructure-as-code templates tailored to the application's architecture and deploy them across supported environments.

Ultimately, such a platform democratizes DevOps, making it more accessible, intelligent, and scalable. It enables individuals and teams — regardless of size or expertise — to build, deploy, and manage full-stack applications with confidence and efficiency.

1.3 Objectives of the Project

This section presents the main goals of the project, which aim to simplify and enhance the DevOps experience through intelligent automation. By integrating AI and modern development practices, the project seeks to provide a platform that not only automates full-stack deployments but also assists developers in architectural decisions, environment configuration, and infrastructure management — all with minimal manual effort.

To achieve these goals, we designed and developed a platform named **IDP-X**, which stands for *Internal Developer Platform – Next Generation.* The "X" highlights the innovative and forward-thinking nature of the solution, positioning it as a modern evolution of traditional developer platforms.

IDP-X aims to democratize DevOps by reducing operational burdens and enabling intelligent, user-guided workflows.

Build an Intelligent Platform for DevOps

Develop a user-friendly platform that empowers developers to create infrastructure, deployments, and environments without requiring deep DevOps expertise. IDP-X offers seamless integration with modern tools such as GitHub, Jenkins, Docker, Terraform, and Ansible, while abstracting away the underlying complexity through a unified interface.

Automate Full-Stack Deployment

Enable end-to-end automation for deploying backend, frontend, and infrastructure. The system should support one-click deployments, Destroy mechanisms, and real-time status monitoring — allowing teams to iterate and scale more rapidly.

AI Assistant for Architecture and Setup

Leverage AI to assist developers in designing system architectures, configuring environments, and optimizing deployments. The assistant analyzes project requirements and suggests best practices for scalability, reliability, and cost-efficiency. This transforms DevOps from a manual and error-prone activity into a smart, guided experience. IDP-X includes a chatbot interface that communicates via API with **LLaMA 3**, augmented by contextual documentation, enabling interactive assistance throughout the development and deployment lifecycle.

2. Review of the Existing

In this chapter, we provide a comprehensive overview of the current landscape of DevOps platforms and automation technologies. We begin by exploring modern deployment platforms such as Vercel and Railway. Then, we examine widely-used automation engines — Terraform, Ansible, and Kubernetes — which serve as the backbone of infrastructure management today. This review sets the foundation for understanding how the proposed intelligent platform can build upon and enhance these existing solutions.

2.1 Existing IDPs (Internal Developer Platforms)

In this section, we examine widely-used platforms that simplify application deployment and abstract much of the underlying infrastructure complexity. We focus on Vercel and Railway — two developer-friendly platforms known for their streamlined workflows and ease of use. By analyzing their features, strengths, and limitations, we aim to understand the current developer experience and identify the gaps that a smarter DevOps platform can address.

2.1.1 Vercel

According to the official Vercel documentation [15], the platform streamlines the deployment of web applications, particularly those built using modern frontend technologies such as React, Next.js, Vue.js, and Svelte. It enables developers to launch websites and web services with immediate deployment, automatic scalability, and minimal configuration.

Vercel natively supports popular frontend frameworks and operates on a globally distributed, secure infrastructure that delivers content efficiently. This global network ensures optimal speeds by serving data from locations closest to end users.

Figure 2.1: Vercel logo

Figure 2.1 illustrates the official logo of Vercel, a modern deployment platform widely used in developer workflows.

Context within an Intelligent Developer Platform (IDP)
In the context of our Intelligent Developer Platform (IDP), Vercel serves as a foundational reference. It demonstrates how a platform can abstract away much of the infrastructure complexity, empowering developers with a clean, efficient, and intuitive deployment pipeline. While Vercel focuses primarily on frontend applications, its approach to automation and user experience is essential to consider when designing a more intelligent and versatile platform.

Key Features:

- **Native Support for Frontend Frameworks:** Vercel works seamlessly with popular web technologies such as Next.js, React, Vue.js, and Svelte, enabling fast and simplified deployment for modern frontend applications.

- **Globally Distributed Content Delivery:** The platform uses an international network of servers to deliver content from locations closest to end users, which minimizes latency and improves load times.
- **Dynamic Auto-Scaling:** Applications deployed on Vercel automatically adjust their resource usage in response to traffic fluctuations, ensuring stable performance without manual scaling.
- **Instant Preview Links:** For every code change or pull request, Vercel creates a unique deployment preview, making it easier for teams to test and review updates before pushing to production.
- **Integrated CI/CD Pipeline:** Vercel includes continuous integration and deployment capabilities out-of-the-box, simplifying the release process and reducing operational overhead.
- **Environment Configuration:** The platform allows easy management of environment variables and supports multiple environments such as development, staging, and production, ensuring flexibility across deployment stages.

Limitations:

- **Backend Integration Constraints:** Vercel is primarily tailored for frontend-focused projects. Incorporating complex backend logic often requires external services or additional setup beyond what the platform directly offers.
- **Resource and Execution Limits:** There are predefined constraints on serverless functions, such as maximum file size and execution time, which may not suit applications with intensive processing needs.
- **Risk of Vendor Lock-In:** Leveraging Vercel-specific capabilities can make migration to other platforms more challenging due to compatibility and configuration differences.
- **Limited Control over Infrastructure:** While abstraction simplifies the deployment process, it also reduces the ability to fine-tune or customize the underlying infrastructure components according to specific requirements.

2.1.2 Railway

According to the official Railway documentation [13], Railway is a development platform that abstracts away infrastructure concerns, enabling developers to build, deploy, and manage applications effortlessly. It provides an all-in-one solution that supports backend and full-stack projects, offering an intuitive user interface, prebuilt templates, and integrations with common services such as PostgreSQL, Redis, and more. Railway emphasizes ease of use, rapid iteration, and automation.

Railway

Figure 2.2: Figure 2.2: Railway logo

Figure 2.2 shows the official logo of Railway - a modern deployment platform widely adopted by developers for its simplicity and powerful infrastructure abstraction.

Railway in the Context of an Intelligent Developer Platform (IDP):

Railway serves as a model of a modern Internal Developer Platform that simplifies both frontend and backend deployments. Within our IDP vision, Railway illustrates how intelligent automation and minimal configuration can enhance developer productivity. Its unified interface and automatic infrastructure handling make it particularly valuable for teams seeking to deploy full-stack applications without managing infrastructure manually.

Key Features:

- **Full-Stack Support:** Unlike platforms focused only on frontend frameworks, Railway accommodates full-stack development with backend services, databases, and APIs.

- **Instant Project Setup:** Developers can deploy prebuilt templates or connect GitHub repositories to launch projects with minimal setup time.
- **Integrated Service Management:** Railway provides built-in support for managing databases and third-party services directly from its dashboard.
- **Continuous Deployment:** Offers seamless Git-based CI/CD pipelines that automatically deploy on code push.
- **Environment Configuration:** Simplifies management of environment variables and secrets across different deployment stages.
- **Team Collaboration:** Facilitates multi-user collaboration with shared environments and deployment previews.

Limitations:

- **Limited Custom Infrastructure Control:** Similar to other abstracted platforms, Railway limits user control over the underlying infrastructure, which may not suit advanced or highly specific deployment needs.
- **Platform Dependency:** Heavy reliance on Railway-specific workflows and configurations could introduce vendor lock-in challenges.
- **Scalability for Complex Projects:** While suitable for small to medium-sized applications, large enterprise-grade systems may face constraints in customization or performance tuning.
- **Pricing at Scale:** Though user-friendly, the pricing model may become less economical as usage grows, especially for resource-intensive applications.

2.1.3 Comparison Between Vercel and Railway:

To better understand the positioning of Vercel and Railway within the ecosystem of modern development platforms, the following table provides a comparative overview of their key features, strengths, and limitations. This comparison is particularly relevant in the context of Internal Developer Platforms (IDPs), as both solutions aim to simplify deployment processes while abstracting infrastructure concerns, with different focuses and capabilities.

Table 2.1: Comparison between Vercel and Railway as Internal Developer Platforms

Criteria	**Vercel**	**Railway**
Primary Use Case	Optimized for frontend applications and static sites	Designed for backend and full-stack applications, with built-in support for databases and services
Framework Support	Excellent support for frontend frameworks: React, Next.js, Vue, etc.	Supports a wide range of backend technologies, with flexibility in full-stack development
Infrastructure Abstraction	**High:** hides most infrastructure complexity to focus on frontend deployment	**High:** abstracts backend infrastructure, databases, and deployments
CI/CD Integration	Built-in, seamless continuous deployment and preview environments	Built-in CI/CD pipelines with deployment automation
Backend Capabilities	**Limited:** requires external services or APIs for backend logic	**Strong:** supports deploying backend services and integrates with databases
Scaling & Performance	Auto-scaling with global CDN	Auto-scaling with hosted compute and database instances
Learning Curve	Very beginner-friendly for frontend developers	Beginner-friendly with slightly more complexity for backend management
Limitations	Limited backend functionality, vendor lock-in, and customization control	More flexibility, but limitations on free tier resources and advanced configuration

2.2 Automotive tools

In this section, we explore the essential automation tools that underpin the DevOps workflows. Technologies such as Docker, Terraform, Ansible, and Jenkins play pivotal roles in containerizing applications, provisioning infrastructure, automating configurations, and orchestrating deployment pipelines at scale. By analyzing their core functionalities and practical use cases, we aim to highlight the foundational technologies on which intelligent DevOps platforms are built.

To establish a clear link between theory and practice—and to strengthen the technical rigor of this report—we focus exclusively on detailing the automation engines that we have actively used throughout the IDP-X project. This approach ensures that readers gain a precise understanding of the components that form part of our system, avoiding any confusion about which tools are integrated within IDP-X.

2.2.1 Docker

According to the official Docker documentation [7], Docker is an open-source platform designed to automate the deployment, scaling, and management of applications using containerization. It allows developers to package applications and their dependencies into standardized units called containers, which run consistently across different computing environments.

Docker simplifies the development lifecycle by enabling a uniform environment from development to production. Its lightweight nature ensures quick startup times and efficient resource usage. Docker images, which define the container content, can be versioned, shared, and deployed repeatedly with ease, improving reproducibility and collaboration.

In the context of Intelligent Developer Platforms (IDPs), Docker serves as a foundational component for application isolation, scalability, and portability. It integrates seamlessly with CI/CD tools, cloud platforms, and orchestration systems like Kubernetes, making it an essential part of modern DevOps toolchains.

Figure 2.3: Docker logo

Figure 2.3 displays the official Docker logo, representing a key technology in modern software delivery workflows.

2.2.2 Terraform

According to the official Terraform documentation [9], Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp that enables users to define and provision infrastructure using a high-level configuration language known as HCL (HashiCorp Configuration Language). It allows for consistent and repeatable deployments by managing resources across multiple cloud providers (such as AWS, Azure, GCP) and on-premise systems. One of Terraform's key strengths is its ability to track infrastructure changes through its execution plan and state file, ensuring that deployments remain predictable and manageable over time. Its modular approach and support for reusable components make it ideal for teams seeking scalable and maintainable infrastructure setups.

Figure 2.4: Terraform logo

Figure 2.4 presents the official Terraform logo, symbolizing its role as a foundational tool for Infrastructure as Code (IaC) in modern DevOps practices.

2.2.3 Ansible

Ansible is an open-source automation tool developed by Red Hat, designed primarily for configuration management, application deployment, and task automation. Unlike other tools that require agent installations, Ansible operates in an agentless manner using SSH, making it simpler to set up and maintain. It uses YAML-based playbooks to describe the desired system state in a human-readable way, enabling teams to automate complex workflows with minimal effort. Ansible is especially favored for its ease of use, low learning curve, and powerful integrations across cloud platforms and DevOps pipelines, helping reduce manual operations and enhance system consistency [10].

Figure 2.5: Ansible logo

Figure 2.5 displays the official Ansible logo, representing a widely adopted tool in configuration management and IT automation workflows.

2.2.4 Jenkins

According to the official Jenkins documentation [11], Jenkins is an open-source automation server that facilitates continuous integration and continuous delivery (CI/CD) in software projects. It enables developers to automatically build, test, and deploy applications, integrating seamlessly with various version control systems and external tools.

Jenkins is highly extensible through its large ecosystem of plugins, allowing teams to customize pipelines that suit their workflows. It supports both declarative and scripted pipeline definitions, enabling flexibility for various project sizes and complexities. Jenkins plays a critical role in automating repetitive tasks, minimizing integration issues, and accelerating software delivery processes in modern DevOps environments.

Figure 2.6: Jenkins logo

Figure 2.6 presents the official Jenkins logo, representing a core tool in continuous integration and delivery automation.

2.2.5 Additional Tools in the DevOps workflow:

While IDP-X primarily leverages tools such as Docker, Terraform, Ansible, and Jenkins, it is worth noting that several other automation tools — including Kubernetes, Puppet, Chef, and Spinnaker — are widely used in enterprise environments. These alternatives offer diverse approaches to configuration management and CI/CD orchestration, and may be evaluated for integration in future versions of the platform to expand its compatibility and flexibility.

2.3 AI-Powered Tools in the DevOps Landscape

While Chapter 1 introduced the transformative role of AI in modern DevOps practices, this section offers a concise review of prominent tools and technologies currently leveraging artificial intelligence in the DevOps ecosystem.

Several platforms now integrate AI capabilities to streamline continuous integration, delivery, and monitoring workflows. Tools like GitHub Copilot, Amazon CodeGuru, Datadog AIOps, Dynatrace Davis and Splunk ITSI

GitHub Copilot:

Developed by GitHub and OpenAI, Copilot acts as an AI pair programmer by suggesting whole lines or blocks of code based on the developer's context. It helps accelerate coding tasks, reduce boilerplate writing, and increase productivity by learning from billions of lines of open-source code [8].

Amazon CodeGuru:

An AI-powered code reviewer by AWS that provides intelligent recommendations to improve code quality and application performance. It integrates with CI/CD pipelines to identify security vulnerabilities, detect resource leaks, and suggest optimizations based on runtime profiling [4].

Datadog AIOps:

Datadog's AIOps engine uses machine learning to automatically detect anomalies, correlate events, and reduce alert fatigue. It enables predictive incident detection and helps teams focus on high-priority signals during system outages or unusual behavior [2].

Dynatrace Davis:

Davis is the AI engine behind Dynatrace, capable of automatically discovering application dependencies, analyzing root causes, and predicting potential service degradations. It helps operations teams proactively address issues before they impact users [1].

Splunk ITSI (IT Service Intelligence):

Splunk's AI-driven ITSI platform offers anomaly detection, event clustering, and root cause analysis by analyzing machine data. It provides real-time health scores for services and applications, improving observability and mean time to resolution (MTTR) [3].

Summary and Outlook:

These tools—each targeting a specific stage of the DevOps lifecycle—demonstrate the growing integration of AI into software engineering workflows. From intelligent code generation to predictive monitoring, AI augments the capabilities of DevOps teams by increasing speed, accuracy, and resilience.

It is important to note that this list is not exhaustive. Numerous other platforms and research initiatives are pushing the boundaries of what AI can accomplish in DevOps, including automated testing tools, AI-enhanced deployment strategies, and security-focused analytics.

In this evolving landscape, our platform follows the same trend by integrating an AI assistant based on the LLaMA 3 language model. This assistant not only guides users through deployment steps but also analyzes documentation and project context to deliver personalized, context-aware recommendations. As AI continues to advance, its role in DevOps will become even more prominent—leading to smarter pipelines, faster iterations, and more autonomous systems.

3. Concept and Design of the Intelligent IDP

In this chapter, we present the architectural vision and design of the Intelligent Internal Developer Platform (IDP) that we developed. The primary goal of this platform is to automate and simplify the deployment lifecycle by seamlessly integrating infrastructure provisioning, CI/CD workflows, and AI-driven assistance.

We begin by introducing a high-level functional overview of the platform. This section outlines the major capabilities of the IDP and the interactions between key actors and system components, illustrated through a simplified use case diagram. It serves as a foundation for understanding the platform's primary functionalities at a glance.

Following this, we outline the global workflow of the platform, highlighting the interactions between its various modules and services. This includes the provisioning logic, CI/CD orchestration, and the integration of a context-aware AI chatbot powered by LLaMA 3.

Next, we describe the technology stack employed at each layer of the system — from the frontend user interface to the backend orchestration engine, as well as the infrastructure automation components, including Terraform, Ansible, and Docker.

Finally, we provide an overview of the repository structure, explaining the rationale behind the directory layout to ensure modularity, scalability, and maintainability.

This foundational chapter sets the stage for the detailed implementation and deployment discussions in the subsequent sections.

3.1 High-Level Functional View :

This section provides a simplified functional overview of the Intelligent Internal Developer Platform (IDP). The purpose of this high-level use case view is to highlight the core capabilities of the platform, as well as the main interactions between the user and the system. This diagram serves as a foundation for understanding the primary services offered by the platform without delving into implementation specifics.

To begin using the platform, users must authenticate via GitHub. Once authenticated, users are presented with two primary options: interacting with the integrated AI assistant, or directly initiating a new deployment configuration.

The AI assistant is designed to provide context-aware support by suggesting optimal deployment strategies or even generating a complete deployment configuration tailored to the user's repository and project needs.

Alternatively, users may choose to manually generate a deployment configuration by selecting key parameters, including: the GitHub repository and project name, the architecture of the target deployment server, server configuration settings, and more. This process enables the platform to assemble a complete set of deployment files. Further details on this functionality and its interface are provided in Section 6.3.

This process is encapsulated in the action called **Create New Project**, which refers to a combination of repository metadata and deployment method, along with the automatically generated configuration files such as: `Dockerfile`, `Jenkinsfile`, `main.tf`, `inventory.ini`, and `deploy.yml`.

Once a new project is created, the user can choose to deploy it. This triggers the CI/CD pipeline on the selected server, ultimately initiating the deployment process (explained in detail later in the chapter).

Users can also access the sidebar to view a list of all existing projects. From this interface, they can select a specific project to inspect its configuration files and current deployment status. Additionally, users are given the ability to either redeploy or destroy the selected project if needed.

The following diagram summarizes these primary functionalities and interactions at a high level.

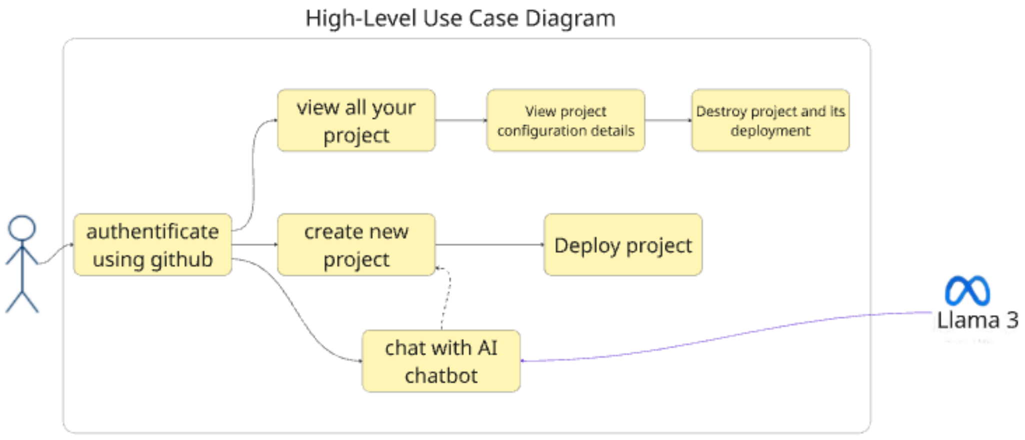

High-Level Use Case Diagram
view all your project
View project configuration details
Destroy project and its deployment
authentificate using github
create new project
Deploy project
chat with AI chatbot
Llama 3

Figure 3.1: High-Level Use Case Diagram

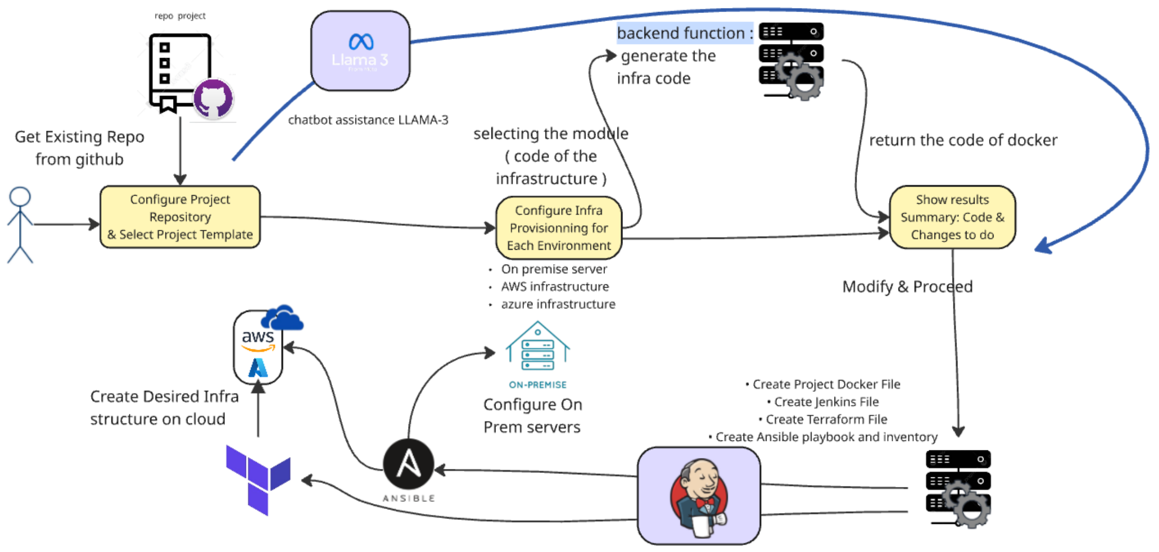
Figure 3.1 illustrates the high-level functional interactions within the Intelligent IDP platform, summarizing the key actions users can perform and the major system components involved.

3.2 Global Workflow of the Platform

In this section, we present the overall architecture of the platform, which is primarily designed to serve as a boilerplate solution for automating the provisioning and CI/CD processes of developers' projects. The platform simplifies project setup by integrating repository management, environment configuration, infrastructure code generation, and deployment workflows into a single unified system.

3.2.1 Provisioning Workflow

Provisioning in this platform is designed to simplify and automate the setup of infrastructure required for developers' applications across multiple environments. By abstracting low-level cloud and on-prem configurations into reusable templates and modules, the platform ensures consistency and scalability.



repo project
Llama 3
chatbot assistance LLAMA-3
backend function : generate the infra code
Get Existing Repo from github
selecting the module (code of the infrastructure)
return the code of docker
Configure Project Repository & Select Project Template
Configure Infra Provisionning for Each Environment
Show results Summary: Code & Changes to do
• On premise server
• AWS infrastructure
• azure infrastructure
Modify & Proceed
aws
Create Desired Infra structure on cloud
ON-PREMISE
Configure On Prem servers
• Create Project Docker File
• Create Jenkins File
• Create Terraform File
• Create Ansible playbook and inventory
ANSIBLE

Figure 3.2: Workflow diagram for provisioning

- **Project Initialization** The process starts with the user either creating a new Git repository or linking an existing one from GitHub. This flexibility allows both greenfield and legacy projects to be on-boarded onto the platform. Once the repository is selected, the user chooses a project template. This template dictates the initial structure, languages, and tech stack (Node.js web app, Python backend, etc.), and also sets expectations for the types of infrastructure components needed.

- **Deployment Environment Configuration**

  After project selection, the user defines the deployment environments (e.g., dev, staging, prod). These environments are essential units of separation for infrastructure and CI/CD pipelines. Each environment will eventually correspond to:

  - A separate Git branch
  - Its own set of infrastructure configuration files
  - Unique deployment workflows and secrets

  This configuration phase sets the foundation for infrastructure provisioning by clearly scoping what needs to be provisioned and maintained.

- **Infra Module Selection and Configuration**

  Once environments are set, the user proceeds to configure the infrastructure provisioning needs for each one. This involves selecting the desired platform:

  - Cloud-based (e.g., AWS, Azure)
  - On-premise servers

  Behind the scenes, the platform uses a backend function that intelligently maps these selections to prebuilt infrastructure modules — often based on Terraform for cloud and Ansible for on-prem. This function determines which code blocks/modules to include and how to parameterize them based on user inputs (such as region, instance type, VPC settings, etc.).

- **Backend Code Generation**

  The backend service now dynamically generates the necessary Infrastructure as Code (IaC):

  - For cloud: Terraform code is structured per environment, containing resources like EC2, S3, RDS, VPC, etc.
  - For on-prem: Ansible playbooks or roles are built to match the server setup and application deployment requirements.

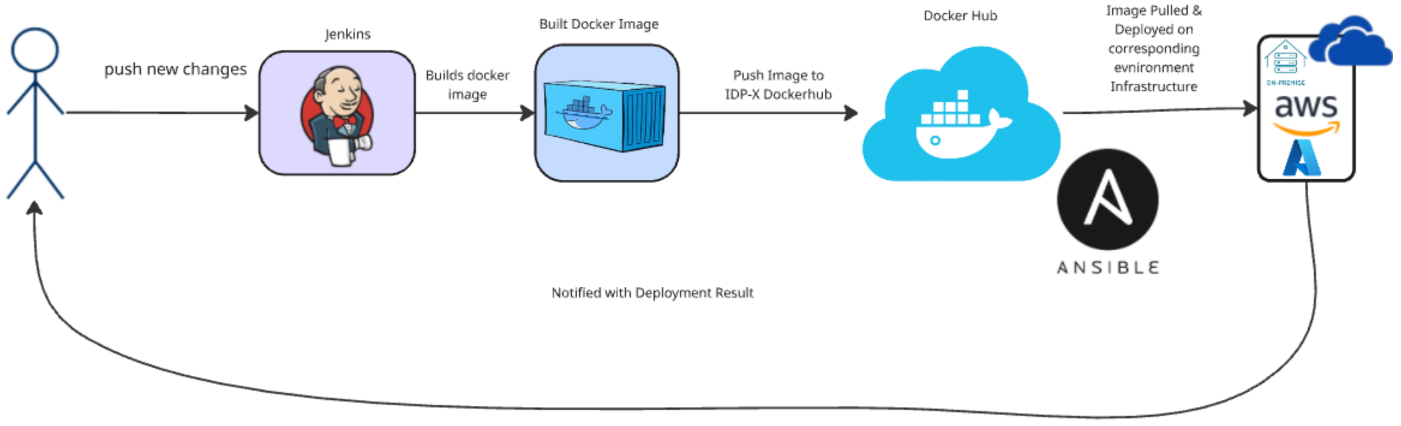

push new changes
Jenkins
Builds docker image
Built Docker Image
Push Image to IDP-X Dockerhub
Docker Hub
Image Pulled & Deployed on corresponding evnironment Infrastructure
ANSIBLE
aws
Notified with Deployment Result

Figure 3.3: CI/CD pipeline flow

These artifacts are not directly applied yet. Instead, they are returned to the user through the interface as a preview, showing:

- The actual code
- A summary of changes
- The components that will be created/modified/deleted

This allows users to review, audit, or modify the generated code before proceeding.

- **Commit and Integration into Git**

  Once the user is satisfied with the generated code, they proceed with modifying and finalizing it if needed. Then, the platform:

  Commits the infrastructure code to the Git repository under appropriate folders

  Creates environment-specific branches

  Adds a standardized GitHub Actions workflow file for CI/CD automation

  Optionally adds a base Dockerfile to containerize the application

  This sets up the repository for end-to-end automation — from infrastructure provisioning to deployment.

- **Infrastructure Provisioning**

  With the IaC committed and the workflows ready, provisioning can begin:

  - Terraform workflows are triggered to apply the cloud infrastructure.
  - Ansible workflows are used to configure on-premise servers via SSH or automation agents.

  This action results in real infrastructure being created and ready for application deployment — including servers, storage, networking, and runtime configurations.

3.2.2 CI/CD Workflow

The CI/CD pipeline in this platform is tailored to streamline the full software delivery lifecycle. From the moment a developer pushes new code to the repository, the platform automates image creation, registry updates, and deployment to target infrastructure, while also providing clear feedback to the contributor. This end-to-end automation ensures rapid and reliable delivery across multiple environments.

- **Code Change Pushed to Repository**

  The developer initiates the workflow by pushing changes (new code, bug fixes, updates) to a GitHub repository. This push event automatically triggers a predefined GitHub Actions workflow.

- **CI: GitHub Actions Builds Docker Image**

  Upon receiving the push event, GitHub Actions handles the Continuous Integration (CI) process. A YAML-based workflow executes the following:

  - Pulls the latest code
  - Optionally runs tests or linting
  - Builds a Docker image using a Dockerfile in the repository

- **Docker Image Is Created**

  The output of the CI process is a Docker image. This image includes the application code and all required dependencies, ensuring consistent behavior across development, staging, and production environments.

- **CD: Push to Docker Hub**

  Following image creation, the CI workflow tags and pushes the image to the container registry, specifically the IDP-X namespace within Docker Hub.

- **Image Pulled & Deployed on Target Infrastructure**

  The platform determines the appropriate deployment target (e.g., AWS EC2, ECS, or on-prem servers) based on branch names or configuration files. The deployment system then pulls the image from Docker Hub and deploys it. This marks the Continuous Deployment (CD) stage.

  Deployment may be facilitated by:

  - Ansible playbooks for remote execution
  - Terraform with remote-exec provisioners
  - A custom pull agent running on the target infrastructure

- **Deployment Feedback**

  After deployment, the developer is notified of the result—success or failure—through channels like GitHub status checks, Slack webhooks, or email.

3.2.3 Chatbot Assistant Integration

To enhance developer productivity and ease the interaction with the Internal Developer Platform, a conversational chatbot is integrated directly into the application interface. This assistant provides contextual support throughout the provisioning and CI/CD workflows. Its key functionalities are:

- Guiding users through project initialization, environment setup, and infrastructure module selection.
- Explaining IaC templates and CI/CD workflows in natural language.
- Recommending infrastructure or CI strategies based on project metadata.

3.3 Technology Stack

The Internal Developer Platform (IDP) is built on a modular, multi-layered architecture that combines modern web development frameworks, automation tooling, and containerization technologies. This section outlines the key components of the tech stack from the user interface down to the infrastructure orchestration layer.

3.3.1 Frontend Platform (User Interface)

The web interface provides an intuitive portal for developers to initiate projects, define environments, and manage infrastructure and CI/CD pipelines.

- **Framework:** Angular
- **Styling:** TailwindCSS or Angular Material
- **API Communication:** REST (Axios, Angular `HttpClient`) or GraphQL

3.3.2 Backend Platform (Orchestration Engine)

This layer handles business logic, infrastructure code generation, and interaction with version control and CI/CD systems.

- **Framework:** Flask (Python)
- **Templating:** Jinja2 for dynamic code generation (Terraform, Ansible, YAML)
- **GitHub Integration:** API calls for repo commit, webhook registration, and workflow triggering

3.3.3 Infrastructure & CI/CD Automation Layer

This layer automates the provisioning of cloud/on-prem infrastructure and defines the continuous delivery lifecycle.

- **Provisioning Tools:** Terraform (Cloud), Ansible (On-Prem)
- **CI/CD Engine:** GitHub Actions (generated YAML workflows)

Containerization, Storage & Execution

Ensures that both the platform services and user applications are containerized and orchestrated efficiently.

- **Containerization:** Docker
- **Image Registry:** Docker Hub (namespace: `idp-x/`)

4. Implementation and Deployment

This chapter presents the practical realization of the Intelligent Developer Platform (IDP-X), detailing how its various components were implemented, integrated, and deployed. We cover the modular software architecture, the incorporation of an AI assistant based on the LLaMA 3 language model, and the decisions made regarding infrastructure automation.

The chapter is organized as follows: we begin with the frontend module, which offers an intuitive interface for interacting with the platform. We then detail the backend module, which orchestrates API calls, user management, and communication with the automation and AI layers. The integration of the AI assistant is examined next, with a justification of our choice of LLaMA 3 through a comparative analysis with other NLP models. Finally, we address deployment strategies and testing methodologies employed to ensure platform stability and functionality.

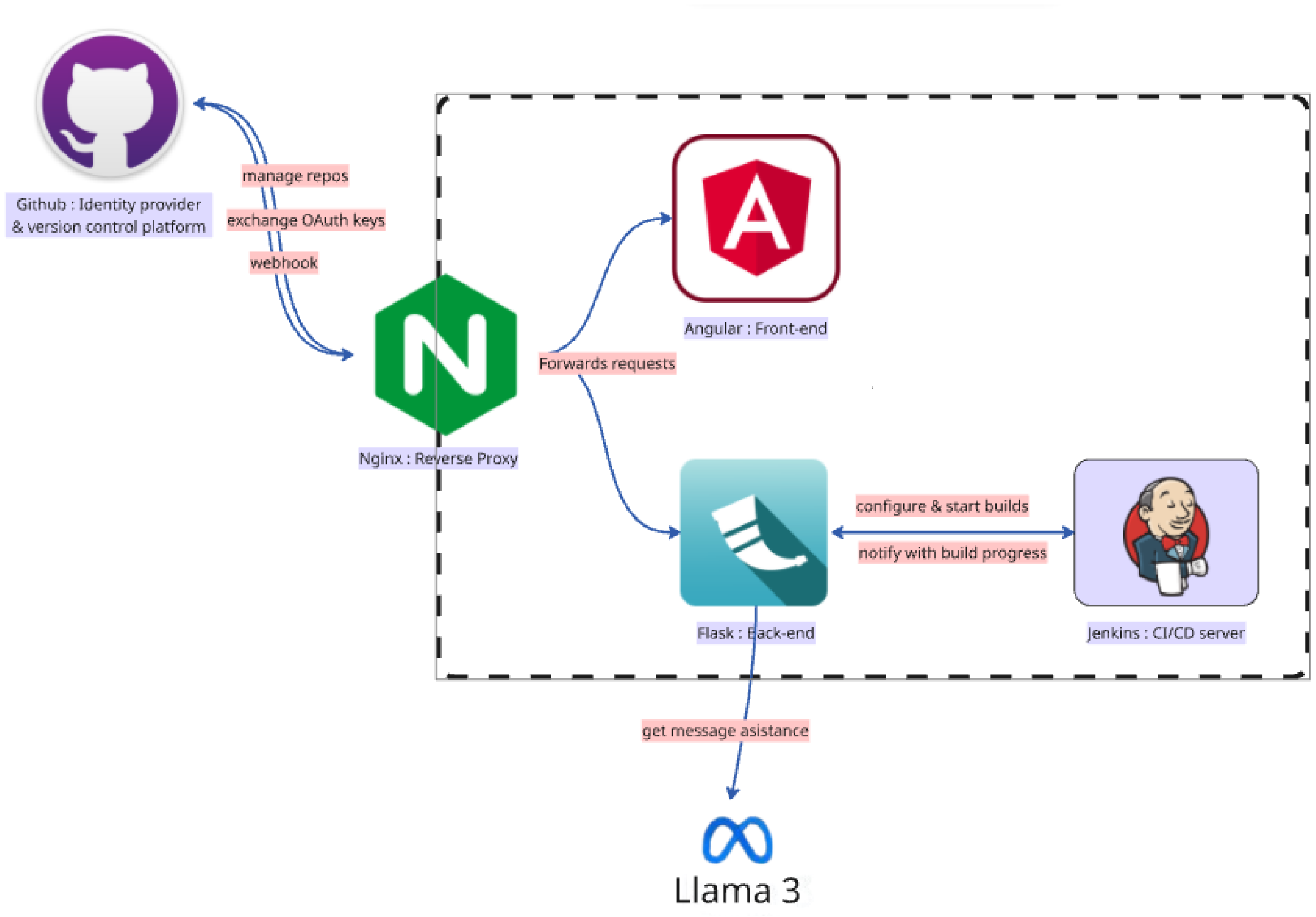


Figure 4.1: System Components and interactions

4.1 Frontend Module

The frontend is developed using **Angular 16**, a modern JavaScript framework known for its modularity and performance. It provides an interactive and user-friendly interface that allows users to configure, visualize, and deploy applications with ease.

To enhance performance, the application leverages **Angular Signals**, a reactive primitive introduced for fine-grained change detection, enabling more efficient state updates across components. For styling and responsive design, we employ **Tailwind CSS**, a utility-first CSS framework that facilitates rapid UI development and ensures a clean, modern appearance across different screen sizes.

The frontend communicates with the backend via **RESTful APIs**, enabling real-time interaction with the platform's core services, including deployment logic, infrastructure automation tools, and the AI assistant. The assistant's responses and suggestions are displayed dynamically within the user interface, offering context-aware support to guide users through the deployment process.

4.2 Backend Module

The backend of the platform is developed using **Flask**, a lightweight and flexible Python web framework. Flask is chosen for its simplicity, extensibility, and ease of integration, making it an ideal choice for building modular microservices and RESTful APIs.

The backend is responsible for several core functionalities, including:

- **API routing:** Managing HTTP endpoints that connect the frontend with the platform's internal services.
- **Authentication and user management:** Securing access using Github authentication.
- **Infrastructure integration:** Orchestrating the execution of tools such as Docker, Terraform, Ansible, and Jenkins to automate deployment workflows.
- **Template generation:** Dynamically creating configuration files (e.g., `Dockerfile`, `main.tf`, `deploy.yml`) tailored to the selected project and deployment strategy.
- **AI assistant integration:** Communicating with the AI assistant via a dedicated endpoint to forward user prompts and return context-aware suggestions.

The modular design of the backend ensures clear separation of concerns and simplifies the integration of external services and DevOps tools. Its service-oriented architecture supports scalability, maintainability, and straightforward deployment.

For more information about how these services are coordinated during deployment, refer to Subsection 3.2.1.

For a detailed discussion on the automation tools used in this platform, see Section 2.2.

4.3 AI Assistant

The AI assistant is powered by the LLaMA 3 model, integrated via an API call to an external endpoint. This assistant uses context-aware prompts derived from documentation files and user queries. These documents (e.g., tool specs, deployment scenarios) are dynamically provided to the API to improve relevance and accuracy.

Why LLaMA 3?

The choice of LLaMA 3 is driven by the following factors:

- **Free to use:** LLaMA 3 is available under a permissive license.
- **Strong performance:** LLaMA 3 achieves competitive results against commercial models.
- **Context-aware input:** It allows detailed prompts with appended documentation, enhancing the assistant's ability to reason over the current task.

4.4 Comparison: LLM vs. Traditional NLP Models

To justify the choice of LLaMA 3 as the AI assistant model, we compare it against two other categories of language models commonly used in natural language processing tasks: traditional NLP models such as BERT and GPT-2, and generic large language models (LLMs) like GPT-3 or GPT-4. Traditional NLP models are widely used for many tasks but tend to have limitations in handling long context and reasoning [6] [12]. Generic LLMs offer impressive capabilities but often come with high computational costs and limited accessibility due to licensing constraints [5] [14]. This comparison focuses on key criteria such as training cost, inference speed, context handling capacity, ease of integration, open-source availability, and suitability for an AI assistant role. For more detailed information on these models, see the original papers and resources: the BERT paper by Google [6], OpenAI's publications on GPT-2 and GPT-3 [12][5], and Meta AI's research on LLaMA 3 [14].

Table 4.1: Comparison between NLP models and LLaMA 3 for assistant integration

Metric	**Traditional NLP (BERT, GPT-2)**	**Generic LLMs**	**LLaMA 3 (Used)**
Training Cost	Low	Very High	Moderate
Inference Speed	Fast	Slow on large models	Optimized for fast inference
Context Handling	Limited (512–1024 tokens)	Up to 32k tokens	Up to 32k+ tokens
Ease of Integration	High	Medium	High
Open Source	Partially	Often closed	Fully open
Suitability for Assistant Role	Limited reasoning	General-purpose	Fine-tuned for reasoning

Based on the comparison above, LLaMA 3 clearly represents the best fit for our use case. Its balance of moderate training cost, optimized inference speed, extended context handling, open-source availability, and strong reasoning capabilities makes it the ideal choice for powering the AI assistant within the IDP-X platform. This is why we chose LLaMA 3 as the foundation of our intelligent assistant.

4.5 Deployment and Testing

4.5.1 Deployment Targets

Due to the storage and cost implications of bundling individual Docker, Terraform, and Ansible files for each product configuration, the application is currently deployed locally. These infrastructure files make cloud deployment costly and complex, as each instance holds heavy static resources.

4.5.2 Testing Strategies

Testing includes:

- **Unit Testing:** Ensures individual modules (frontend/backend) function as expected.
- **Integration Testing:** Verifies the interoperability of components like AI assistant, CI/CD, and DevOps tools.
- **Manual Testing:** Used to validate the recommendations and output of the AI assistant.

5. Conclusion and Future Work

This chapter summarizes the achievements of the project, reflects on its current limitations, and outlines future development directions, including the integration of microservices and security enhancements.

5.1 Summary of Achievements

Throughout this project, we successfully developed an intelligent Internal Developer Platform (IDP) that integrates widely-used DevOps tools such as Docker, Terraform, Ansible, and Jenkins. The platform includes a responsive frontend, a modular backend, automated infrastructure provisioning, a CI/CD pipeline, and an AI assistant to streamline deployment and configuration tasks.

5.2 Limitations

Despite its functionality, the current platform has a few limitations:

- The AI assistant is still basic and does not offer deep learning or self-adaptive capabilities.
- Support for multi-cloud deployments and hybrid infrastructure is limited.
- Security mechanisms are minimal, leaving the platform vulnerable to various attack vectors.
- The deployment model is currently monolithic and lacks microservices support.

5.3 Improvements

Future improvements will focus on:

- Enhancing the user interface with real-time validations and feedback.
- Adding more third-party DevOps and monitoring integrations.
- Introducing granular access control and permission systems.
- Supporting GitOps workflows for environment-specific deployment configurations.

5.4 AI Assistant Upgrades

Planned upgrades to the AI assistant include:

- Using larger language models (LLMs) for smarter and more contextual decision-making.
- Adding support for natural language to YAML/JSON translation.
- Personalized suggestions based on usage history and past deployment patterns.
- Proactive error detection and automated troubleshooting recommendations.

5.5 Scalability and Security

As we move toward production-grade maturity, scalability and security will become central:

- **Microservices Architecture:** A future version of the platform will support deployment using a microservices-based model, allowing independent scaling, versioning, and better fault isolation.
- **Security Layers:** Given the current website's vulnerability, an important next step is the integration of robust security practices, including input sanitization, token-based authentication, HTTPS enforcement, and logging/auditing mechanisms.
- Container orchestration with Kubernetes and advanced load balancing will also be explored to ensure reliability and scalability.

5.6 Final Thoughts

This project demonstrates the potential of combining DevOps tools with AI to improve infrastructure automation and software delivery. While the initial prototype delivers essential functionality, future iterations—especially those introducing microservices and enterprise-grade security—will greatly enhance usability, reliability, and trustworthiness. The path ahead lies in making the platform smarter, safer, and scalable to meet the evolving demands of modern software teams.

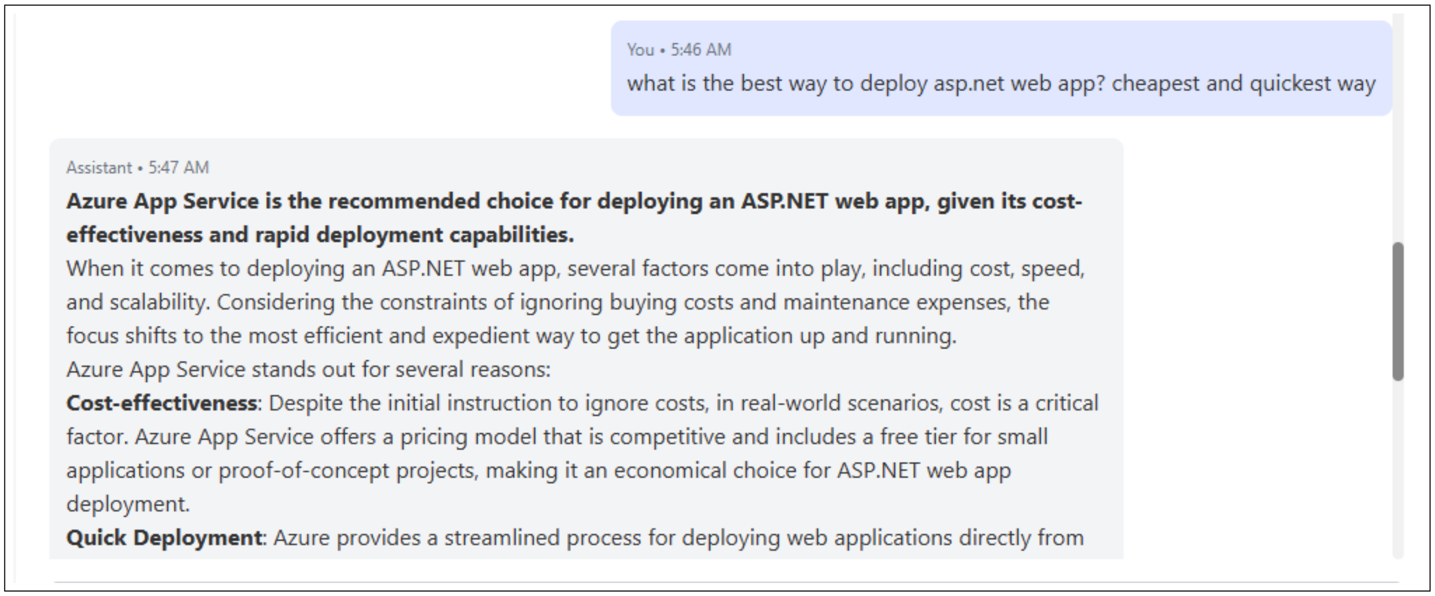
6. Annexes

6.1 AI Prompt Examples

Deployment Plan suggestion - ASP.NET example:

As part of enhancing user experience and simplifying decision-making during the configuration process, the chat assistant enables users to interact conversationally to seek advice, recommendations, or clarification related to their deployment needs.

For instance, a user might ask, *"What is the best way to deploy an ASP.NET web app? Cheapest and quickest way"*, and the assistant would respond contextually with tailored suggestions—such as proposing Azure App Service as an optimal choice based on factors like ease of deployment, cost-effectiveness, and compatibility with .NET technologies. Beyond providing mere guidance, the assistant is context-aware and can influence the configuration pane dynamically; when a recommendation is made, relevant fields in the setup interface are automatically suggested or pre-filled, saving time and reducing errors. This integration turns the configuration process from a static form-filling task into a responsive and intuitive workflow guided by conversational input.



You • 5:46 AM

what is the best way to deploy asp.net web app? cheapest and quickest way

Assistant • 5:47 AM

Azure App Service is the recommended choice for deploying an ASP.NET web app, given its cost-effectiveness and rapid deployment capabilities.

When it comes to deploying an ASP.NET web app, several factors come into play, including cost, speed, and scalability. Considering the constraints of ignoring buying costs and maintenance expenses, the focus shifts to the most efficient and expedient way to get the application up and running.

Azure App Service stands out for several reasons:

Cost-effectiveness: Despite the initial instruction to ignore costs, in real-world scenarios, cost is a critical factor. Azure App Service offers a pricing model that is competitive and includes a free tier for small applications or proof-of-concept projects, making it an economical choice for ASP.NET web app deployment.

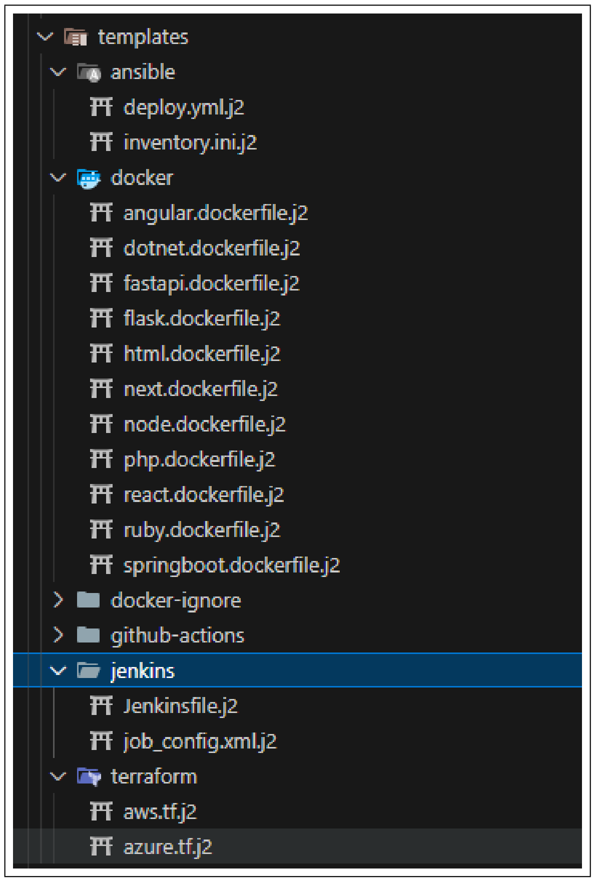
Quick Deployment: Azure provides a streamlined process for deploying web applications directly from

Figure 6.1: Example of chat prompt asking suggestion

6.2 Templating strategy for Scripts

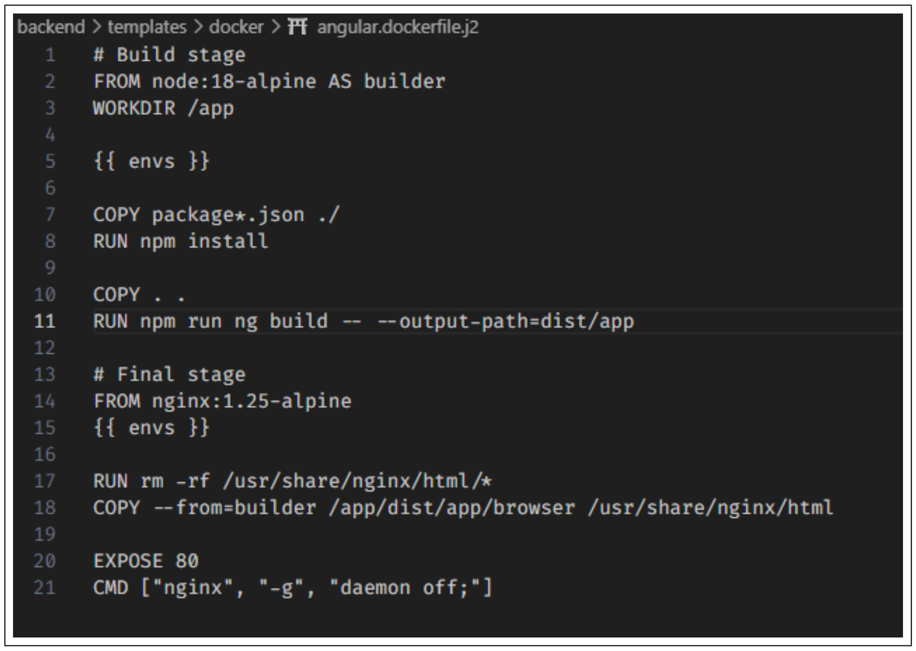
At the core of our IDP-X lies a powerful templating system that ensures flexibility, scalability, and consistency across deployments. The application is designed around a library of pre-prepared templates, covering all the essential components required for automated deployment workflows. These include templates for `Dockerfile`, `.dockerignore`, `Terraform`, `Ansible`, and Jenkins CI/CD pipelines. Each template is tailored to specific technology stacks and cloud providers—such as AWS and Azure for infrastructure provisioning, and various Docker stack configurations for different application types (e.g., Node.js, Python, ASP.NET).

When a user initiates a deployment, the platform dynamically selects and customizes these templates based on the user's input and configuration selections. This modular and reusable template system enables the platform to generate the appropriate configuration files at the right stage in the pipeline, ensuring both consistency and adaptability. Whether deploying to a cloud provider or an on-premise environment, the templating engine ensures that all generated scripts and infrastructure definitions are compatible and optimized for the selected environment.



```
templates
  ansible
    deploy.yml.j2
    inventory.ini.j2
  docker
    angular.dockerfile.j2
    dotnet.dockerfile.j2
    fastapi.dockerfile.j2
    flask.dockerfile.j2
    html.dockerfile.j2
    next.dockerfile.j2
    node.dockerfile.j2
    php.dockerfile.j2
    react.dockerfile.j2
    ruby.dockerfile.j2
    springboot.dockerfile.j2
  docker-ignore
  github-actions
  jenkins
    Jenkinsfile.j2
    job_config.xml.j2
  terraform
    aws.tf.j2
    azure.tf.j2
```

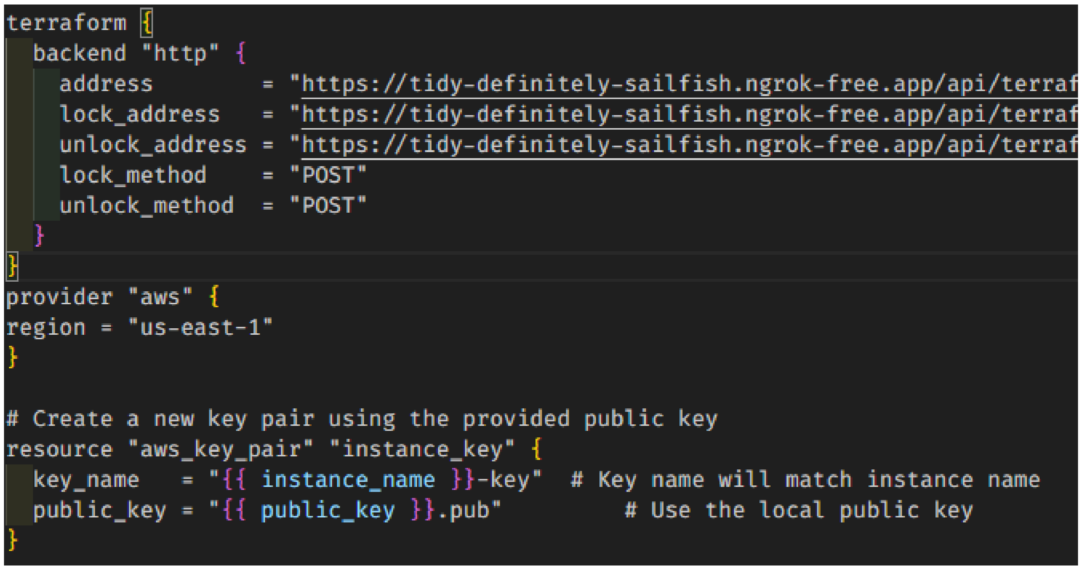
Figure 6.2: Folder hierarchy for templates



backend > templates > docker > angular.dockerfile.j2

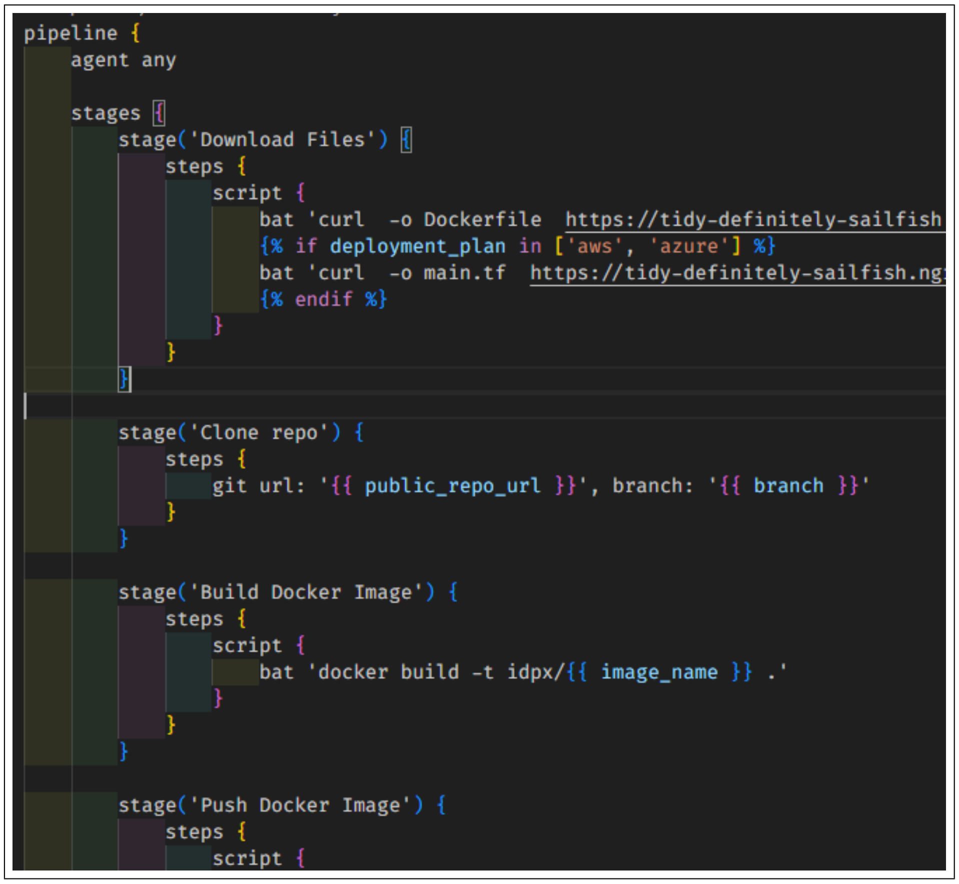
```
# Build stage
FROM node:18-alpine AS builder
WORKDIR /app

{{ envs }}

COPY package*.json ./
RUN npm install

COPY . .
RUN npm run ng build -- --output-path=dist/app

# Final stage
FROM nginx:1.25-alpine
{{ envs }}

RUN rm -rf /usr/share/nginx/html/*
COPY --from=builder /app/dist/app/browser /usr/share/nginx/html

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Figure 6.3: Example of Docker snippet for an Angular stack



```
terraform {
  backend "http" {
    address        = "https://tidy-definitely-sailfish.ngrok-free.app/api/terraf
    lock_address   = "https://tidy-definitely-sailfish.ngrok-free.app/api/terraf
    unlock_address = "https://tidy-definitely-sailfish.ngrok-free.app/api/terraf
    lock_method    = "POST"
    unlock_method  = "POST"
  }
}
provider "aws" {
region = "us-east-1"
}

# Create a new key pair using the provided public key
resource "aws_key_pair" "instance_key" {
  key_name   = "{{ instance_name }}-key"  # Key name will match instance name
  public_key = "{{ public_key }}.pub"         # Use the local public key
}
```

Figure 6.5: Terraform template snippet for AWS infra



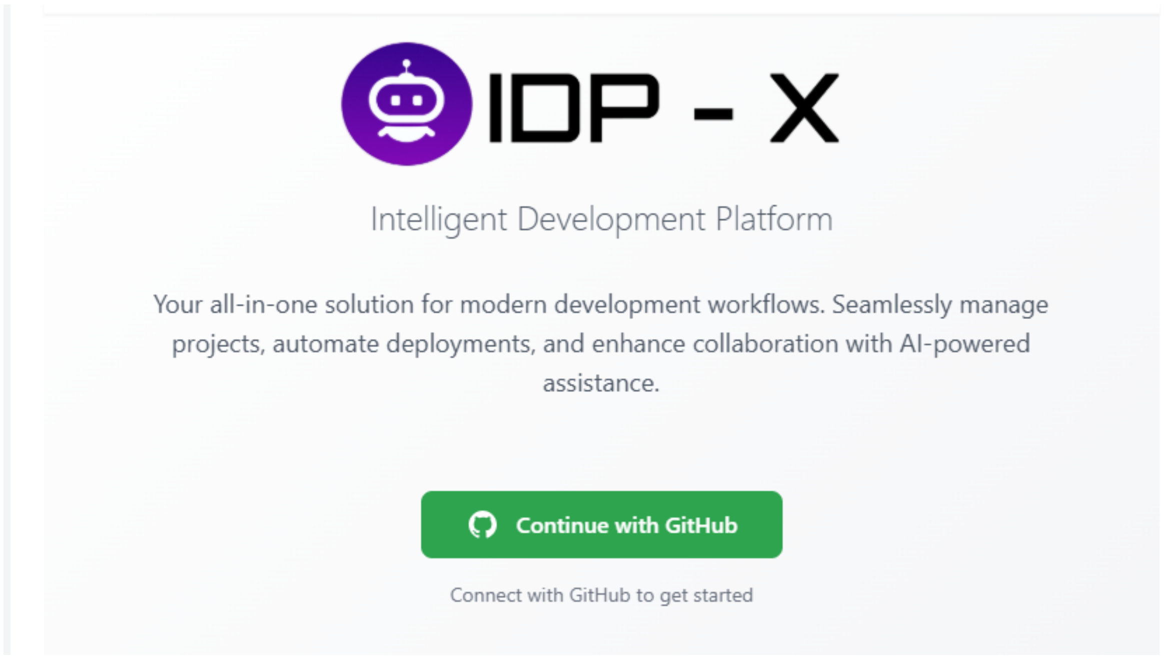
```
pipeline {
    agent any

    stages {
        stage('Download Files') {
            steps {
                script {
                    bat 'curl  -o Dockerfile  https://tidy-definitely-sailfish
                    {% if deployment_plan in ['aws', 'azure'] %}
                    bat 'curl  -o main.tf  https://tidy-definitely-sailfish.ng
                    {% endif %}
                }
            }
        }

        stage('Clone repo') {
            steps {
                git url: '{{ public_repo_url }}', branch: '{{ branch }}'
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    bat 'docker build -t idpx/{{ image_name }} .'
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
```

Figure 6.4: The Jenkins template snippet

6.3 Platform Walkthrough

6.3.1 Initial page

This is the initial screen users encounter before accessing the app's features. At this stage, the application requires users to authenticate before proceeding. The only available action is to connect using GitHub as an identity provider. This connection is essential to ensure secure access and personalized experience based on each user's GitHub profile.

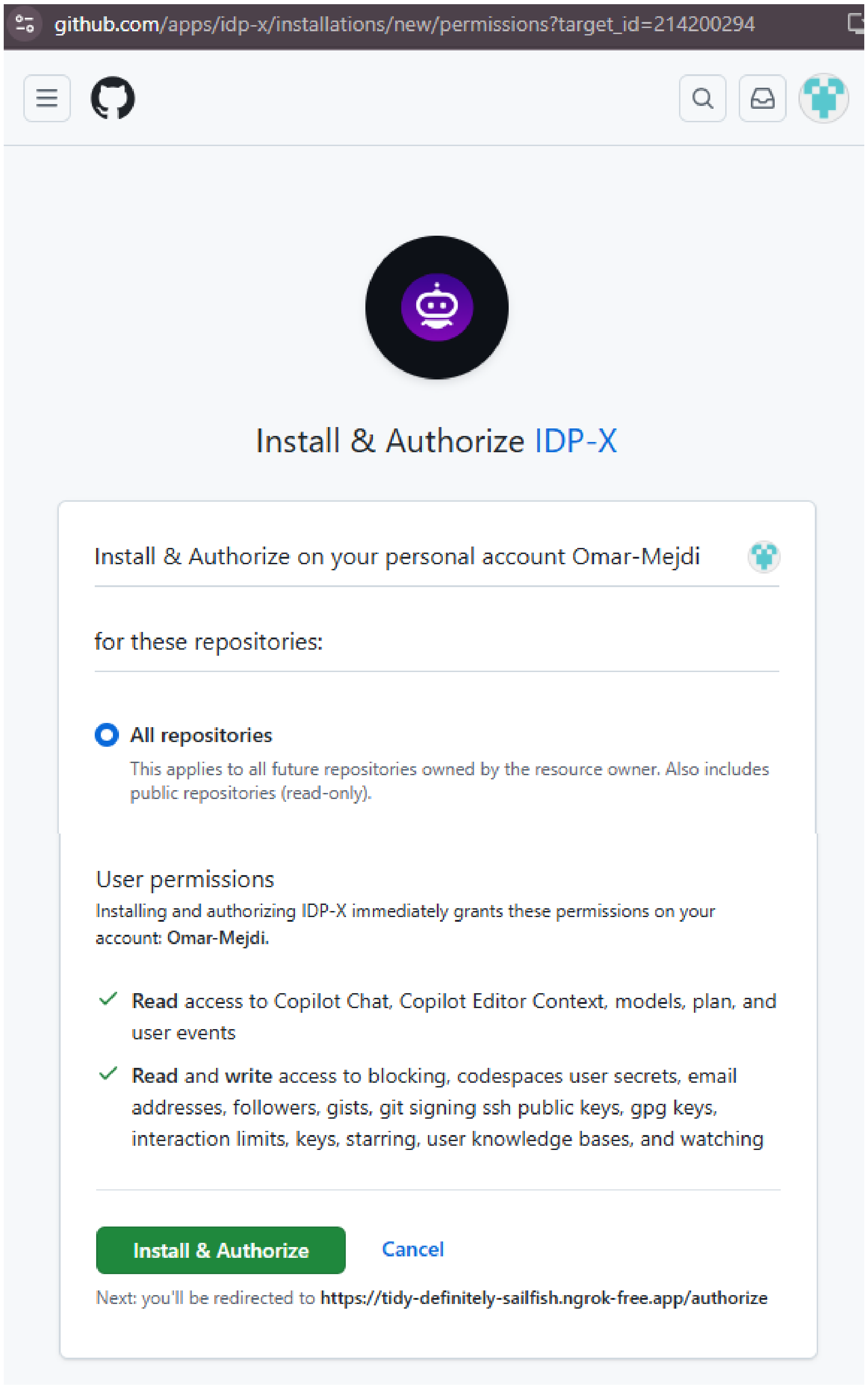

IDP - X
Intelligent Development Platform
Continue with GitHub
Connect with GitHub to get started

Your all-in-one solution for modern development workflows. Seamlessly manage projects, automate deployments, and enhance collaboration with AI-powered assistance.

Figure 6.6: Landing Page - disconnected

6.3.2 Connecting to Github

To proceed, the application requires you to connect using GitHub. This is because IDPX functions as a GitHub App, and as such, it needs to be installed on your currently active GitHub account. This installation is crucial to grant the necessary permissions that allow our platform to manage your repositories, access metadata, and subscribe to repository events. Once connected, IDPX will use GitHub's OAuth2 strategy to securely exchange authorization codes for access tokens. These tokens are essential to perform actions on your behalf, such as listing your repositories, reading user profile information, and responding to specific GitHub webhook events like pushes, merges, and pull requests. All interactions are handled through secure authentication flows, ensuring that your data and identity are protected


github.com/apps/idp-x/installations/new/permissions?target_id=214200294

Install & Authorize IDP-X

Install & Authorize on your personal account Omar-Mejdi

for these repositories:

[✓ All repositories]
This applies to all future repositories owned by the resource owner. Also includes public repositories (read-only).

User permissions
Installing and authorizing IDP-X immediately grants these permissions on your account: **Omar-Mejdi.**

✓ **Read** access to Copilot Chat, Copilot Editor Context, models, plan, and user events

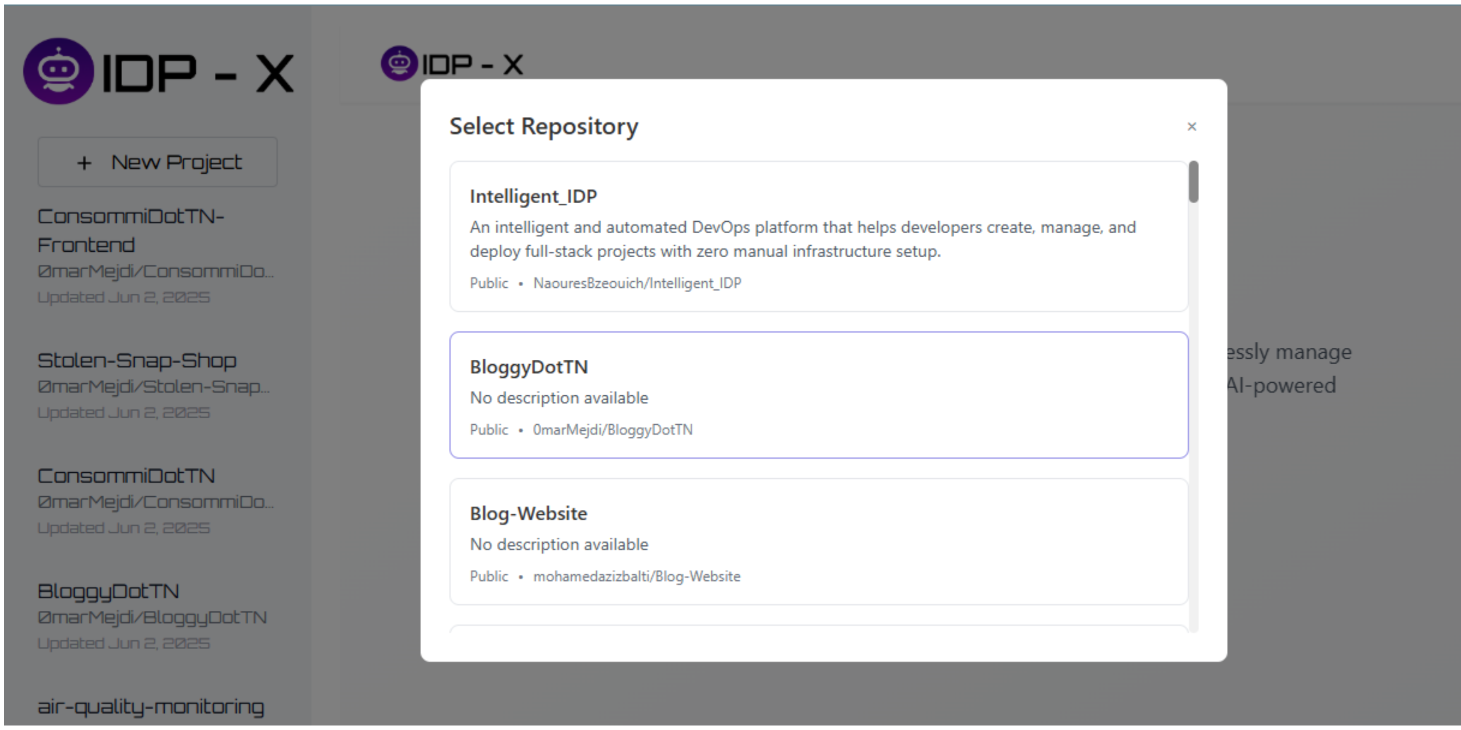
✓ **Read** and **write** access to blocking, codespaces user secrets, email addresses, followers, gists, git signing ssh public keys, gpg keys, interaction limits, keys, starring, user knowledge bases, and watching

Next: you'll be redirected to **https://tidy-definitely-sailfish.ngrok-free.app/authorize**

Figure 6.7: Connecting with github - installing idp-x

6.3.3 Selecting Repository to start project

After successfully connecting your GitHub account, the next step is to choose an existing repository from your account. IDPX will fetch and display a list of your accessible repositories, allowing you to select one as the base for your deployment projec



IDP - X

+ New Project

ConsommiDotTN-Frontend
0marMejdi/ConsommiDo...
Updated Jun 2, 2025

Stolen-Snap-Shop
0marMejdi/Stolen-Snap...
Updated Jun 2, 2025

ConsommiDotTN
0marMejdi/ConsommiDo...
Updated Jun 2, 2025

BloggyDotTN
0marMejdi/BloggyDotTN
Updated Jun 2, 2025

air-quality-monitoring

Select Repository

Intelligent_IDP
An intelligent and automated DevOps platform that helps developers create, manage, and deploy full-stack projects with zero manual infrastructure setup.
Public • NaouresBzeouich/Intelligent_IDP

BloggyDotTN
No description available
Public • 0marMejdi/BloggyDotTN

Blog-Website
No description available
Public • mohamedazizbalti/Blog-Website

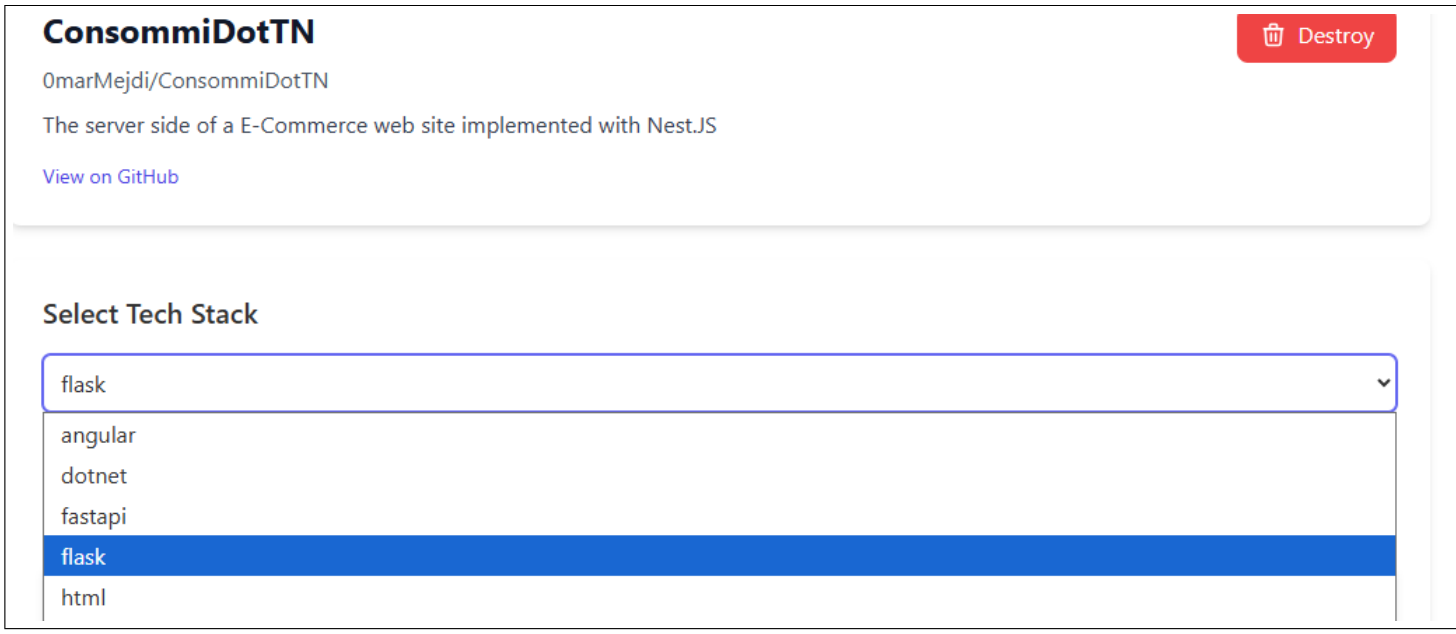
Figure 6.8: User's Repository selection

6.3.4 Configuring project setup

Once you've selected a repository and started a project, you'll be redirected to a configuration page. Here, you can define all the essential parameters that shape how your application will be built and deployed.

You'll be able to:

- Choose the tech stack to use when building your Docker image (e.g., Node.js, Python, Java, etc.).
- Set up environment variables required by your application to function properly.
- Define the deployment strategy, selecting between cloud providers like AWS or Azure, or opting for an on-premises deployment.
- If deploying on-prem, provide additional parameters: the IP address of the remote server, the SSH username, and the SSH public key used for communication.


ConsommiDotTN

Destroy

0marMejdi/ConsommiDotTN

The server side of a E-Commerce web site implemented with Nest.JS

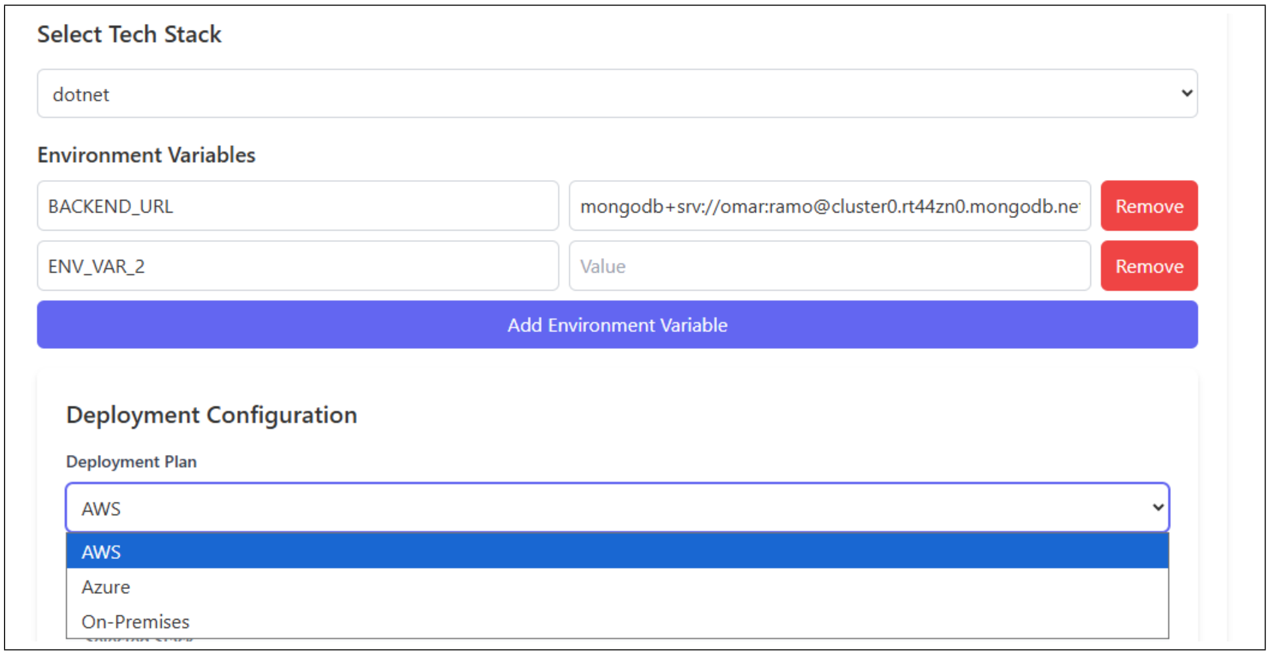
View on GitHub

Select Tech Stack

flask

angular
dotnet
fastapi
flask
html

Figure 6.9: Deployment Config - Stack Selection


Select Tech Stack

dotnet

Environment Variables

BACKEND_URL	mongodb+srv://omar:ramo@cluster0.rt44zn0.mongodb.ne	Remove
ENV_VAR_2	Value	Remove

Add Environment Variable

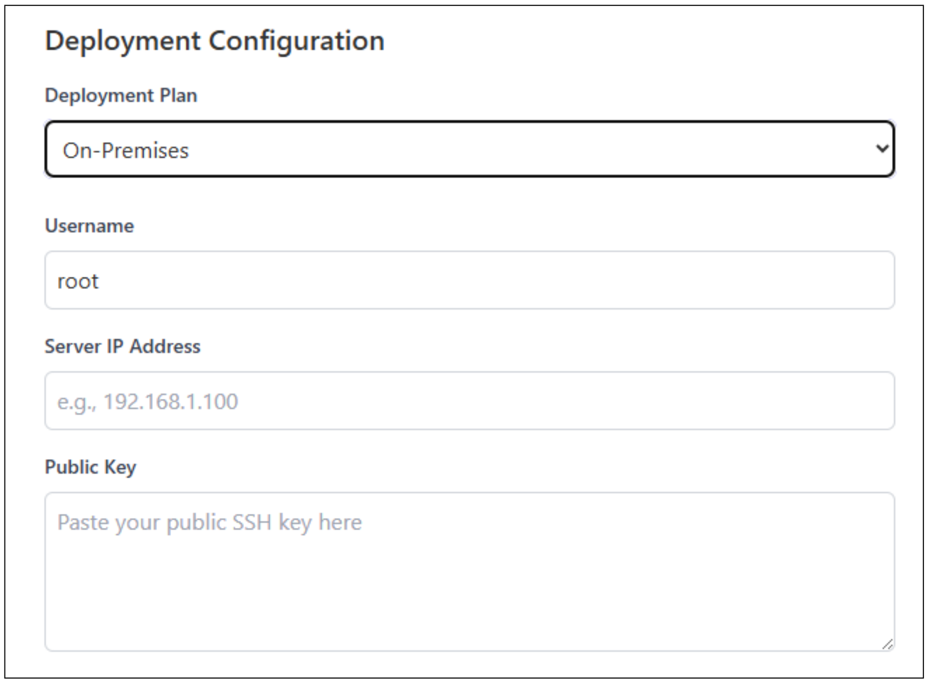
Deployment Configuration

Deployment Plan

AWS

AWS
Azure
On-Premises

Figure 6.10: Deployment Config - Env Vars & Deployment Plan


Deployment Configuration

Deployment Plan: On-Premises

Username: root

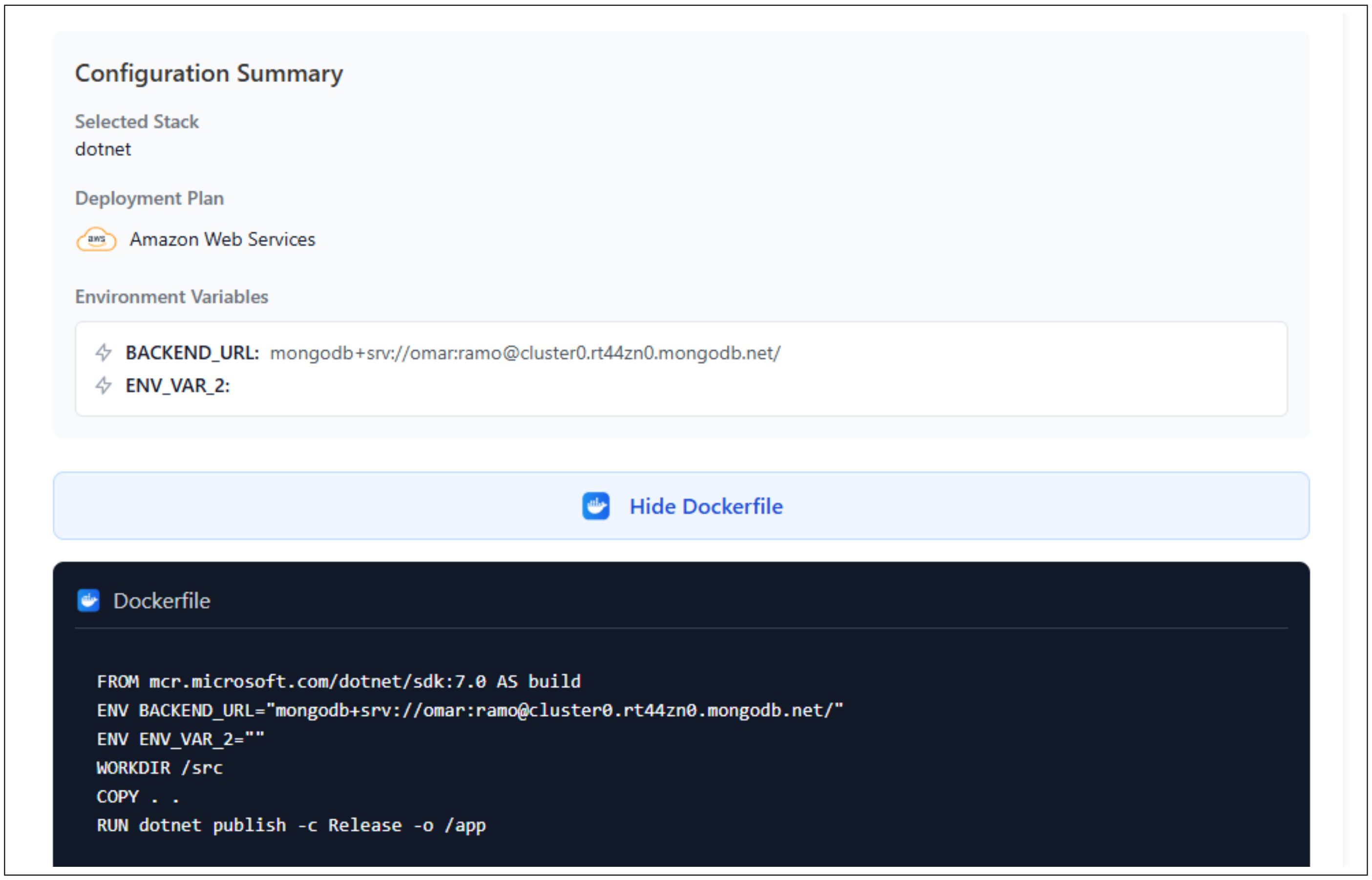
Server IP Address: e.g., 192.168.1.100

Public Key: Paste your public SSH key here

Figure 6.11: Deployment Config - On prem Options

6.3.5 Configuration Summary

After defining all necessary parameters such as the tech stack, environment variables, and deployment strategy, you are redirected to the **Configuration Summary** screen. This final step serves as a recap interface, allowing you to preview and confirm the full set of configurations before triggering the actual deployment. It displays the selected repository, deployment target (e.g., AWS, Azure, on-prem), the environment values, and other essential settings in a concise format. Additionally, this screen exposes the generated `Dockerfile` that will be used to build the container image, giving users transparency and the opportunity to review the instructions that will run inside the build phase. This step ensures correctness and gives a last chance to make any required adjustments before validating and launching the deployment pipeline.



Configuration Summary

Selected Stack
dotnet

Deployment Plan
Amazon Web Services

Environment Variables

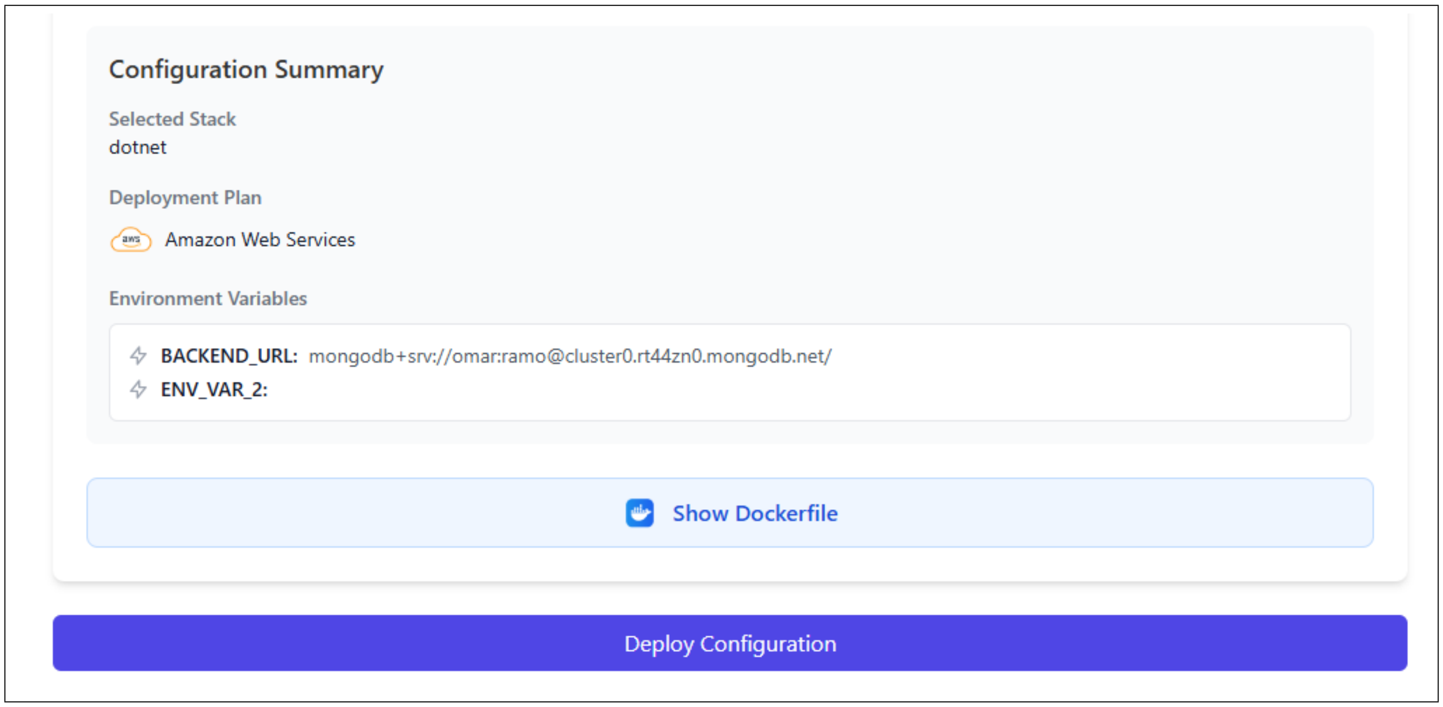
BACKEND_URL: mongodb+srv://omar:ramo@cluster0.rt44zn0.mongodb.net/
ENV_VAR_2:

```
FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
ENV BACKEND_URL="mongodb+srv://omar:ramo@cluster0.rt44zn0.mongodb.net/"
ENV ENV_VAR_2=""
WORKDIR /src
COPY . .
RUN dotnet publish -c Release -o /app
```

Figure 6.12: Deployment Config - Summary

6.3.6 Submission and Launch

Now that you are confident in the configurations you've defined, the final step is to hit the **Deploy Configuration** button. This action triggers the actual deployment workflow, handling all the heavy lifting behind the scenes. It will generate the corresponding configuration files including the `Dockerfile`, `Terraform` scripts, `Ansible` playbooks and inventory files—each tailored to your setup and executed in the proper sequence. Additionally, it will create a `Jenkinsfile` that defines the CI/CD pipeline. Once generated, the system communicates with the Jenkins server to retrieve these files and initiate the pipeline. Jenkins will then proceed to provision the required infrastructure (if applicable), build and push the Docker image, and finally configure the destination servers to pull and run the built image. This step consolidates all configurations into a seamless and automated deployment process.



Configuration Summary

Selected Stack
dotnet

Deployment Plan
Amazon Web Services

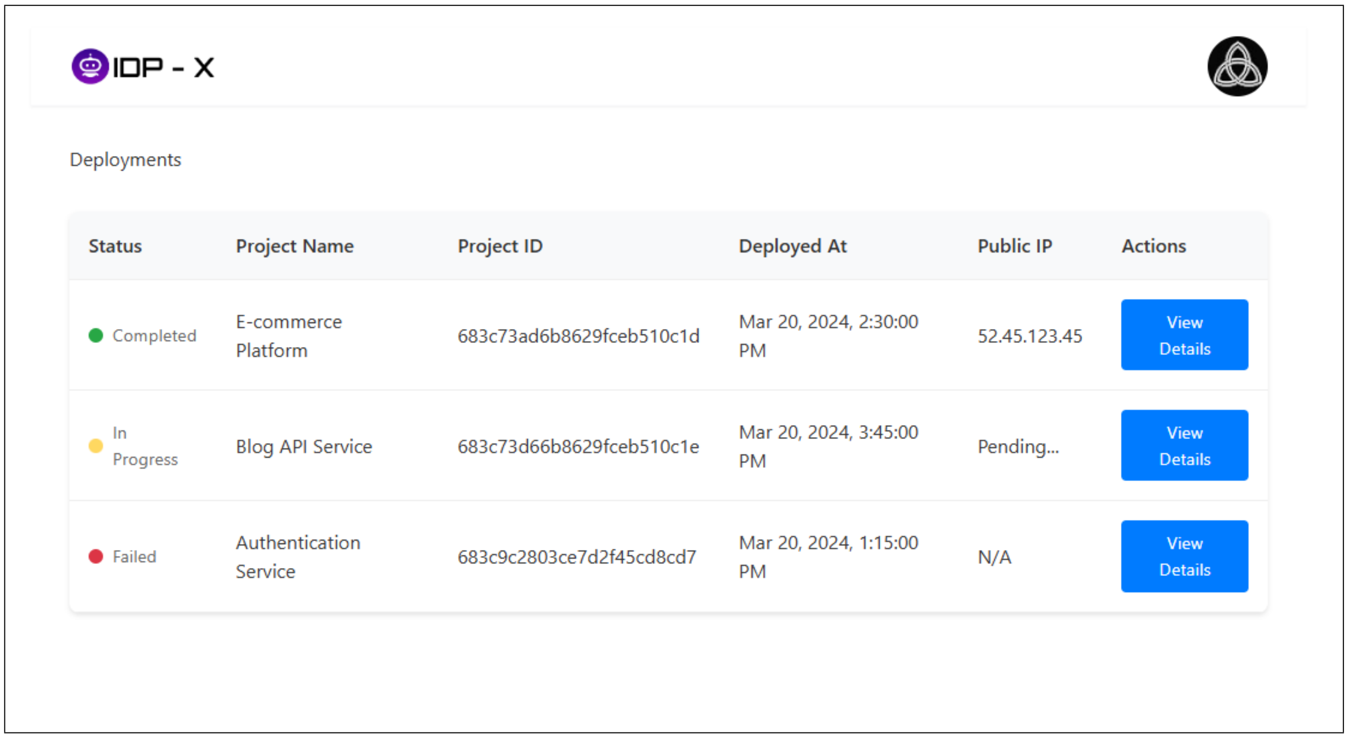
Environment Variables

BACKEND_URL: mongodb+srv://omar:ramo@cluster0.rt44zn0.mongodb.net/
ENV_VAR_2:

Figure 6.13: Deployment Config - Submit

6.3.7 Deployments dashboard

The Deployments Dashboard serves as a centralized view where users can track all deployment activities executed through the platform. Each entry in the dashboard reflects the real-time status of a deployment—whether it is in progress, completed successfully, or encountered an error. For successful deployments, the dashboard conveniently displays the associated IP address of the target server or endpoint, enabling direct access to the deployed application. This summary view helps users quickly assess the outcome of their configurations and take action if needed, such as re-deploying failed setups or validating running services.



Deployments

Status	Project Name	Project ID	Deployed At	Public IP	Actions
● Completed	E-commerce Platform	683c73ad6b8629fceb510c1d	Mar 20, 2024, 2:30:00 PM	52.45.123.45	View Details
● In Progress	Blog API Service	683c73d66b8629fceb510c1e	Mar 20, 2024, 3:45:00 PM	Pending...	View Details
● Failed	Authentication Service	683c9c2803ce7d2f45cd8cd7	Mar 20, 2024, 1:15:00 PM	N/A	View Details

Figure 6.14: Deployments dashboard

6.4 Glossary of Terms

- **IDP (Internal Developer Platform)**: A self-service platform that enables developers to build, deploy, and manage applications autonomously, using infrastructure and tools abstracted by the platform team.
- **LLM (Large Language Model)**: A type of AI model trained on vast amounts of text data, used for generating code, documentation, or assisting in configuration within IDPs.
- **NLP (Natural Language Processing)**: A subfield of AI focused on enabling machines to understand and generate human language. Useful in IDPs for chatbots, auto-completion, and intelligent config generation.
- **IaC (Infrastructure as Code)**: Practice of defining infrastructure (servers, networks, policies) using code (e.g., Terraform, Pulumi). Key to automating deployments in IDPs.
- **CI/CD (Continuous Integration / Continuous Deployment)**: Automation pipelines for building, testing, and releasing software. Central to IDPs to streamline DevOps.
- **OAuth2**: An industry-standard protocol for authorization. Allows the platform to securely access user data (like GitHub repos) without requiring passwords.
- **Token**: A credential (like a string) used to authenticate users or apps and authorize actions (e.g., API access). Access tokens, refresh tokens, installation tokens, etc. are common types.
- **Webhook**: A mechanism for receiving real-time updates/events (e.g., code push, build complete). IDPs use webhooks to trigger workflows on events like GitHub pushes.
- **DevOps**: A set of practices aimed at unifying software development and operations. IDPs aim to make DevOps practices easier and more accessible for developers.
- **Environment Variables**: Key-value pairs used to configure applications without hardcoding values. IDPs often allow teams to manage and inject these into deployments.
- **Container**: A lightweight, portable software unit that includes everything needed to run an app. IDPs typically use Docker or similar tools to build and run containers.

- **Kubernetes (K8s)**: An orchestration system for automating deployment, scaling, and management of containerized applications.
- **Jenkins**: An open-source automation server that facilitates CI/CD pipelines by automating parts of the software development process.
- **Reverse Proxy**: A server that sits in front of web servers and forwards client (e.g., browser) requests to those web servers.
- **Provisioning**: The process of setting up IT infrastructure such as servers, storage, or networking services.
- **Access Key / SSH Key**: A cryptographic key used to authenticate access to a remote server, often used in secure automated deployments.
- **Deployment Plan**: A predefined workflow that describes how an application should be built, tested, deployed, and monitored.
- **Build Pipeline**: A series of steps that transform source code into a deliverable product, usually including compilation, testing, and packaging.

Bibliography

[1] *dynatrace.* Accessed on Avril 2, 2025.

[2] *datadoghq documentation* . Accessed on Avril 2, 2025.

[3] *newrelic.com documentation* . Accessed on Avril 5, 2025.

[4] AWS. *CodeGuru website.* Accessed on Mars 31, 2025.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[7] Docker. Docker documentation. `https://docs.docker.com/`, 2025. Accessed on: Mars 25, 2025.

[8] GitHub and OpenAI. *Github Copilot website.* Accessed on Mars 31, 2025.

[9] HashiCorp. Terraform documentation. `https://developer.hashicorp.com/terraform/docs`, 2025. Accessed on: Mars 25, 2025.

[10] Red Hat. Ansible documentation. https://docs.ansible.com/, 2025. Accessed on Mars 27, 2025.

[11] Jenkins Community. *Jenkins Documentation*, 2025. Accessed on Mars 27, 2025.

[12] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI Blog, 2019.

[13] Railway. Railway documentation. `https://docs.railway.app/`. Accessed: Mars 21, 2025.

[14] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971, 2023.

[15] Vercel. Vercel documentation. `https://vercel.com/docs`, 2025. Accessed: Mars 20, 2025.