

## GRADUATION PROJECT REPORT

Presented for the purpose of obtaining

Diplôme National d'Ingénieur en Sciences Appliquées et Technologiques

Speciality : Computer Science and Engineering

By

Ben Hadj Nasr Mohamed

---

# Orchestrating Critical Application Deployment with Minimal Downtime

---

Professional Supervisor: Mr. Yazid Missaoui

Ingénieur R&D

Academic Supervisor: Mrs. Yousra najjar

Professor

Developed within Adactim





## GRADUATION PROJECT REPORT

Presented for the purpose of obtaining

Diplôme National d'Ingénieur en Sciences Appliquées et Technologiques

Speciality : Computer Science and Engineering

By

Ben Hadj Nasr Mohamed

---

# Orchestrating Critical Application Deployment with Minimal Downtime

---

Professional Supervisor: Mr. Yazid Missaoui

Ingénieur R&D

Academic Supervisor: Mrs. Yousra najjar

Professor

Developed within Adactim



# Acknowledgement

The completion of this project would not have been possible without the unwavering support of my family and friends and their constant encouragement and understanding throughout this journey.

I would like to express my sincere gratitude to **Mr. Yazid Missaoui** for providing me with the opportunity to undertake this graduation project. Their trust in my abilities allowed me to delve deeper and gain valuable knowledge and skills in this fascinating field. Furthermore, their expertise in Cloud/Devops proved invaluable throughout the project. Whenever I encountered technical hurdles, he offered insightful guidance and solutions, which significantly improved the project's overall quality.

I am also deeply grateful to **Mrs. Yousra Najjar** for their mentorship throughout this project. her knowledge of project management methodologies, particularly Scrum, was instrumental in effectively structuring my workflow and achieving project milestones. Additionally, she provided valuable feedback and suggestions during the report writing process, ensuring it was clear, concise, and well-organized. her guidance undoubtedly contributed to the final outcome of this project.

Ben Hadj Nasr Mohamed

# Dedication

*I dedicate this work to my parents, my siblings, my friends, and my teachers.*

*I also dedicate it to the open-source community, who taught me so much about the different technologies I got intrested in. and I wish someday to contribute back to this amazing community.*

# Contents

<b>General Introduction</b>	<b>1</b>
<b>1 General framework of the project</b>	<b>2</b>
I. The Host Organization . . . . .	3
II. Problematic . . . . .	3
II.1. Existing study . . . . .	3
II.2. Problematic . . . . .	4
III. Proposed Deployment Optimization Solution . . . . .	5
IV. Methodology of the project . . . . .	5
<b>2 Fundamentals</b>	<b>8</b>
I. Introduction . . . . .	9
II. Cloud Computing . . . . .	9
III. Cloud Computing Services . . . . .	10
IV. DevOps . . . . .	11
IV.1. Devops lifecycle . . . . .	11
IV.2. Implementation of DevOps . . . . .	12
<b>3 Analatycs and Requirements specifications</b>	<b>14</b>
I. Infrastructure Study . . . . .	15
I.1. Componants of the architecture . . . . .	17
I.2. Network flows . . . . .	18
I.3. Architecture characteristics . . . . .	18
II. Deployment Process . . . . .	18
III. Needs and requirements . . . . .	19
III.1. Functional Needs . . . . .	19
III.2. Non-functional Needs . . . . .	20
IV. The implementation of Scrum process . . . . .	20
IV.1. Scrum Roles . . . . .	20
IV.2. Scrum events . . . . .	20

IV.3.	Scrum implementation . . . . .	21
IV.4.	Product backlog . . . . .	21
IV.5.	Definition of Done . . . . .	22
IV.6.	Sprint planning . . . . .	22
V.	Description of available tools . . . . .	24
V.1.	Terraform . . . . .	24
V.2.	Azure DevOps . . . . .	25
<b>4</b>	<b>The provisioning of the infrastructure</b>	<b>27</b>
I.	Sprint backlog . . . . .	28
II.	Realisation . . . . .	28
II.1.	The difference between the environments . . . . .	29
II.2.	The modified architecture . . . . .	30
III.	Challenges . . . . .	31
IV.	Testing the connections . . . . .	32
<b>5</b>	<b>Continuous Integration and Deployment</b>	<b>34</b>
I.	Sprint backlog . . . . .	35
II.	Version Control strategy . . . . .	35
III.	The CI/CD pipelines . . . . .	39
IV.	Challenges Faced . . . . .	42
<b>6</b>	<b>The deployment strategy</b>	<b>43</b>
I.	Introduction . . . . .	44
II.	Sprint backlog . . . . .	44
III.	The different deployment strategies . . . . .	44
IV.	The different ways to implement the deployment strategy . . . . .	47
V.	The rollback strategy . . . . .	50
VI.	Overview of the flow of the deployment strategy . . . . .	51
VII.	The monitoring strategy . . . . .	53
	<b>General Conclusion</b>	<b>57</b>
	<b>Bibliography</b>	<b>58</b>

# List of Figures

1.1	Logo Entreprise Adactim . . . . .	3
1.2	existing strategy of the application deployment . . . . .	4
1.3	Scrum process [2] . . . . .	7
2.1	DevOps lifecycle . . . . .	12
3.1	global architecture . . . . .	16
3.2	This is the gantt chart of the scrum process . . . . .	23
3.3	Terraform . . . . .	24
3.4	Azure DevOps . . . . .	25
4.1	The file structure . . . . .	29
4.2	The new architecture . . . . .	30
4.3	connectivity problem . . . . .	31
4.4	The DNS configuration . . . . .	32
4.5	the simple web app that was used to test the infrastructure. . . . .	33
5.1	gitlab flow strategy . . . . .	38
5.2	development pipeline . . . . .	40
6.1	blue-green deployment strategy. . . . .	45
6.2	canary deployment strategy. . . . .	46
6.3	Azure App Service Deployment Slots implementation. . . . .	48
6.4	the full workflow of the deployment strategy. . . . .	49
6.5	the rollback strategy implementation . . . . .	50
6.6	seccuss rollback . . . . .	51
6.7	the flow of the deployment strategy . . . . .	52
6.8	monitoring insights set up . . . . .	53
6.9	in this dashboard, you can check the status of the alerts. . . . .	54
6.10	here you can set up the alerts. . . . .	54
6.11	metrics dashboard . . . . .	55



6.12 log analytics dashboard . . . . . 55

# List of Tables

1.1	the significant differences between traditional and agile project methodologies. <b>TradvsAgile1</b>	5
3.1	the key technical objectives for the project . . . . .	20
3.2	the key technical objectives for the project . . . . .	21
4.1	Sprint 1 backlog . . . . .	28
5.1	Sprint 2 backlog . . . . .	35
5.2	the different situations where each branching strategy is suitable . . . . .	36
6.1	Sprint 3 backlog . . . . .	44

# List of acronyms

- **ARM** = **A**zure **R**esource **M**anager
- **AWS** = **A**maزون **W**eb **S**ervices
- **BI** = **B**usiness **I**ntelligence
- **CD** = **C**ontinuous **D**elivery
- **CI** = **C**ontinuous **I**ntegration
- **CIDR** = **C**lassless **I**nter-**D**omain **R**outing
- **CLI** = **C**ommand **L**ine **I**nterface
- **CSP** = **C**loud **S**ervice **P**rovider
- **DNS** = **D**omain **N**ame **S**ystem
- **ERP** = **E**nterprise **R**esource **P**lanning
- **GCP** = **G**oogle **C**loud **P**latform
- **IaC** = **I**nfrastructure **a**s **C**ode
- **IaaS** = **I**nfrastructure **a**s **a** **S**ervice
- **NSG** = **N**etwork **S**ecurity **G**roup
- **PaaS** = **P**latform **a**s **a** **S**ervice
- **QA** = **Q**uality **A**ssurance
- **SKU** = **S**tock **K**eeping **U**nit
- **SaaS** = **S**oftware **a**s **a** **S**ervice
- **TFVC** = **T**eam **F**oundation **V**ersion **C**ontrol
- **URL** = **U**niform **R**esource **L**ocator
- **VM** = **V**irtual **M**achine

- **WAF** = **W**eb **A**pplication **F**irewall
- **YAML** = **Y**AML **A**nother **M**arkup **L**anguage

# General Introduction

In the fast-paced world of web development, users expect applications and websites to be constantly available and up-to-date. This creates a crucial challenge: how to implement new features, fix bugs, and improve performance without disrupting the user experience.

Furthermore, to guarantee an impeccable user experience, testing is a vital step in the development process. However, this testing needs to be streamlined to avoid slowing down development.

This need for efficiency necessitates an automated deployment process. For cloud-hosted websites, setting up such a process presents its own challenges, such as connecting various cloud services and development tools. However, it also offers unique opportunities, like streamlining the creation of the cloud infrastructure itself.

ADACTIM, a cloud and outsourcing services company whose introduction can be found in the **first chapter**, aims to establish an optimized deployment and testing process for a critical cloud-hosted web application. This will accelerate development and allow developers to focus on core coding rather than deployment tasks.

The technical development will follow the DevOps process, requiring an understanding of cloud computing and DevOps practices that will be detailed in the **second chapter**.

After that we will explore the thought process behind selecting the process components in the **third chapter**.

The realisation of the project is detailed in the last three chapters. First in the **fourth chapter**, we'll establish a solid foundation for our cloud environment using Infrastructure as Code (IaC). This involves creating scripts that automate the provisioning of cloud resources. These scripts ensure our infrastructure is not only built efficiently but also remains reusable and maintainable for future changes.

Next in the **fifth chapter**, we'll implement the (CI/CD) pipeline. This pipeline acts like an automated assembly line, taking code changes and automatically triggering builds, tests, and potentially even deployments. all while establishing a DevOps workflow will bridge the gap between development and operations teams, fostering collaboration and a smoother development process.

Finally in the **sixth chapter**, we'll focus on creating deployment scripts and the pipeline flow that guarantees minimal downtime during updates to the live application.

# GENERAL FRAMEWORK OF THE PROJECT

---

## Plan

I.	The Host Organization . . . . .	3
II.	Problematic . . . . .	3
III.	Proposed Deployment Optimization Solution . . . . .	5
IV.	Methodology of the project . . . . .	5

## Introduction

In this chapter we will present a global overview of the project, starting with the presentation of the host organization, followed by the problematic that was the reason behind the project, and finally we will into the methodology that was adopted to achieve this project objectives.

### I. The Host Organization

**ADACTIM** helps businesses leverage technology to improve their operations. They specialize in cloud computing, application integration and management, enterprise resource planning (ERP), and business intelligence (BI). They operate internationally across Europe, North Africa, and Africa.



**Figure 1.1:** Logo Entreprise Adactim

#### ● Adactim Mission

ADACTIM can enable a company to benefit from technological transformations in the areas of IT infrastructure and integrated business systems allowing it to focus its energy on its core business.

*"Their mission is to facilitate businesses' access to technological innovations, to simplify their daily use, allowing the company to be more efficient and competitive. With this, the client company can focus its resources on its development and its customers. the company will be well-equipped with business software and IT infrastructure and outsource appropriate operations processes."*[1]

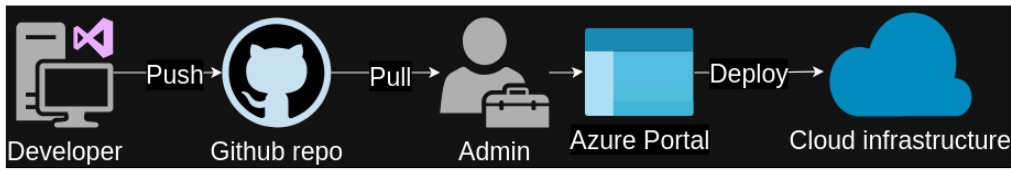
## II. Problematic

### II.1. Existing study

#### **Description of the Current Environment:**

Some applications that are managed by ADACTIM currently runs on a manually provisioned Azure cloud infrastructure managed through the web portal. Which means the actions to these infrastructures are done through a graphical interface provided by azure.

### Analysis of Existing Deployment Processes:



**Figure 1.2:** existing strategy of the application deployment

The deployment of applications in our current setup involves several steps, including code preparation, testing, deployment scheduling, and monitoring. Each deployment requires coordination between multiple teams, including developers, quality assurance, and system administrators. Strengths and Weaknesses of the Current Approach:

#### Strengths:

- Our current deployment process follows a structured workflow, ensuring thorough testing before releasing applications into production.
- Effective communication and collaborative problem-solving are facilitated by teamwork during deployments.

#### Weaknesses:

- Extended outages caused by inefficient deployment practices negatively impact business continuity and revenue.
- Limited automation in certain areas leads to manual errors and delays during deployments.

## II.2. Problematic

While our current deployment process prioritizes rigorous testing and cross-team collaboration, it faces significant challenges impacting both efficiency and reliability. One critical issue lies in the protracted nature of deployment timelines. This stems from the inherent complexity of coordinating and executing numerous manual testing processes. Consequently, not only are the releases of crucial applications delayed but the potential for human error during manual interventions is also amplified.

In essence, our current application deployment process faces a critical challenge: balancing the strengths of its structured workflow and collaborative approach with the need for faster, more automated deployments.



### III. Proposed Deployment Optimization Solution

In order to address the identified challenges, we propose a solution that leverages modern technologies and methodologies like Devops.

- **Provisioning and Configuration Management:** Automate the provisioning and configuration of cloud infrastructure and application environments to ensure consistency and reliability using Infrastructure as Code (IaC) and Configuration Management tools.
- **Automating the build and testing process:** minimize manual intervention and expedite development cycles by leveraging automation across the build and testing pipeline. This not only reduces human error but also frees up valuable time for developers to focus on core tasks.
- **implementing a Deployment strategy:** Implement a deployment strategy that leverages automation to ensure seamless and efficient application deployments, minimizing downtime and errors. By applying these pattern principles:
  - **Rightsize resources for each environment:** ensuring that the resources allocated to each environment are appropriate for the expected load. We can do this by implementing workspaces in Terraform.
  - **Delete non-production environments:** ensuring that non-production environments are deleted when they are no longer needed.

### IV. Methodology of the project

While some projects might seem straightforward, a defined methodology is crucial for achieving success. This framework provides a roadmap, ensuring tasks are completed efficiently and in the right order. By following a methodology, we can avoid common pitfalls, manage our time effectively, and ultimately deliver a successful project.

#### ● Agile vs traditional methodologies

**Tableau 1.1:** the significant differences between traditional and agile project methodologies.  
**TradvsAgile1**

Traditional Methodology	Agile Methodology
-------------------------	-------------------

The Waterfall model is used.	Iterative and incremental development.
Emphasis on planning and design.	Emphasis on flexibility and adaptability.
Emphasis on deliverables and completion	Emphasis on customer satisfaction
Projects are completed in phases.	Projects are completed in sprints.
The strict change management process.	Encourages changes and improvements.
Team roles and responsibilities are fixed.	Team roles and responsibilities are flexible.
Limited customer involvement	High customer involvement
Risk management is proactive	Risk management is reactive

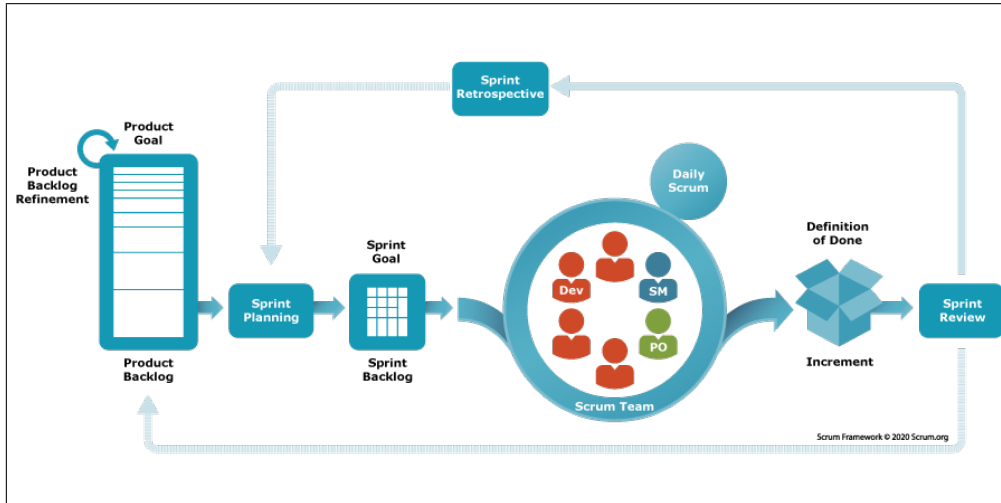
the choice between agile and traditional methodologies hinges on how much flexibility is needed. Agile thrives on an iterative approach. The team continuously works in short cycles, delivering features and gathering feedback to adapt and refine the project as it goes. This makes it ideal for DevOps, where requirements can evolve quickly. Traditional methods, on the other hand, emphasize upfront planning with a detailed roadmap. While this can be beneficial for projects with clear goals from the beginning, it can hinder DevOps teams who need to be adaptable and responsive to new information or feedback throughout the project lifecycle.

### ● The Agile methodology

*"While seemingly designed for larger teams, Agile methodologies offer valuable structure even for single-person projects. Their core principles of iterative development, continuous improvement, and flexibility empower individuals to efficiently manage projects."*[2] **scrum:** Scrum is an agile framework that helps teams structure their work into short development cycles called sprints. Scrum teams commit to shipping work at the end of each sprint and adopt practices and a team structure that helps them achieve this cadence. Scrum takes the agile principles one step further, creating a structure that helps teams live the agile principles in their day-to-day work. Scrum is a well-documented agile framework that many teams can adopt without much disruption.

### ● Verdict

Considering the project's potential for evolving requirements and the benefits of iterative development, Scrum emerges as the optimal choice. Its focus on defined sprints, adaptability, and a structured approach to project management, even for solo developers, aligns perfectly with our project needs.



**Figure 1.3:** Scrum process [2]

By following the Scrum process and maintaining open communication, we can ensure a successful project that delivers value iteratively while continuously adapting to feedback and changes.

## Conclusion

This chapter outlined the challenges associated with our current applications deployment process, which prioritizes thorough testing and collaboration but suffers from lengthy timelines and manual errors. To address these shortcomings, we proposed a new approach that leverages modern technologies and methodologies to achieve faster, more reliable deployments.

The proposed solution emphasizes automation throughout the deployment lifecycle, encompassing infrastructure provisioning, build and testing processes, and the deployment strategy itself. This will minimize manual intervention, reduce errors, and expedite deployments. Additionally, the strategy incorporates security best practices and performance optimization techniques to ensure the applications' continued reliability and availability.

Finally, we discussed the methodology Scrum that will be adopted to manage the project. This agile framework will provide a structured approach to project management.

The next chapter, we will provide a comprehensive exposition to many concepts in the field of DevOps and cloud computing, which will be prove essential to understanding the project's technical aspects.

# FUNDAMENTALS

---

## Plan

I.	Introduction . . . . .	9
II.	Cloud Computing . . . . .	9
III.	Cloud Computing Services . . . . .	10
IV.	DevOps . . . . .	11

## I. Introduction

This chapter dives into several exciting concepts that power this project. We'll begin by exploring cloud computing, its various service offerings, and the available options in this domain. Next, we'll delve into DevOps principles and the tools that empower its implementation. Finally, we'll unveil the specific tools chosen for this project and the reasoning behind their selection.

## II. Cloud Computing

*Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. A simple way to describe the Cloud is its multiple data centers available to many users over the Internet. [3]*

Cloud computing services are offered by various providers, with Amazon Web Services, Microsoft Azure, and Google Cloud Platform being some of the major players in the field.

### ● Comparative study between cloud providers

So it makes sense to compare the three major cloud providers to see which one is the best for our project.

### ● Strengths of the three major cloud providers [3]

- **Amazon Web Services (AWS):** AWS has had almost a 7-year head-start and vastly more offerings at present than other competitors. With that head start, the available talent pool is larger, meaning that more people know AWS.
- **Microsoft Azure:** Azure provides a pretty compelling transition path to the cloud, also Microsoft has a large number of enterprise customers, and many of them are already using Microsoft products. This makes it easier for them to use Azure.
- **Google Cloud Platform (GCP):** Google just happens to originated one of the most popular container orchestration systems, that being Kubernetes. and with that, it has been able to leverage that reputation to attract customers to its cloud platform.

- **Weaknesses of the three major cloud providers [3]**

- **Amazon Web Services (AWS):** With its vast growth Amazon (the company that owns AWS) has become a direct rival to many retailers, and that has led to some customers looking for alternatives.
- **Microsoft Azure:** For the past decade or so, open-source software has found great acceptance, both on-prem and in the cloud, largely due to organizations seeking alternatives to commercial software vendors like Microsoft.
- **Google Cloud Platform (GCP):** Google has a reputation for killing off products that don't meet its expectations, and that has led to some customers being wary of using Google Cloud Platform.

- **Verdict**

since this project is about transitioning to the cloud, and the fact that the company Adactim has a golden partnership with Microsoft, we will be using Microsoft Azure as our cloud provider.

### III. Cloud Computing Services

Cloud computing services are a broad set of services that are delivered over the internet. These services are divided into three main categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

- **IaaS**

IaaS provides virtualized computing resources that require the developer to manage the infrastructure, including the network, servers, and operating systems. and with this comes great flexibility and scalability.

**Offered Services:** Azure offers a wide range of IaaS services, including virtual machines, storage, and networking. With this, we can simulate the on-premises infrastructure in the cloud with minimal changes.

- **PaaS**

PaaS provides a platform allowing developers to build, run, and manage applications without the complexity of building and maintaining the infrastructure. This allows developers to focus on the application itself.

**Offered Services:** Some of the PaaS services offered by Azure include Azure App Service and Azure Functions. and these services have built-in deployment strategies that can be selected.

- **SaaS**

SaaS is a software distribution model in which applications are hosted by a third-party provider so developers don't have to install, maintain, or update the software.

**Offered Services:** Azure offers a wide range of SaaS services, including Office 365, Dynamics 365, and many more. Unfortunately, these services are not relevant to us since we cannot use them to host our application.

- **Verdict**

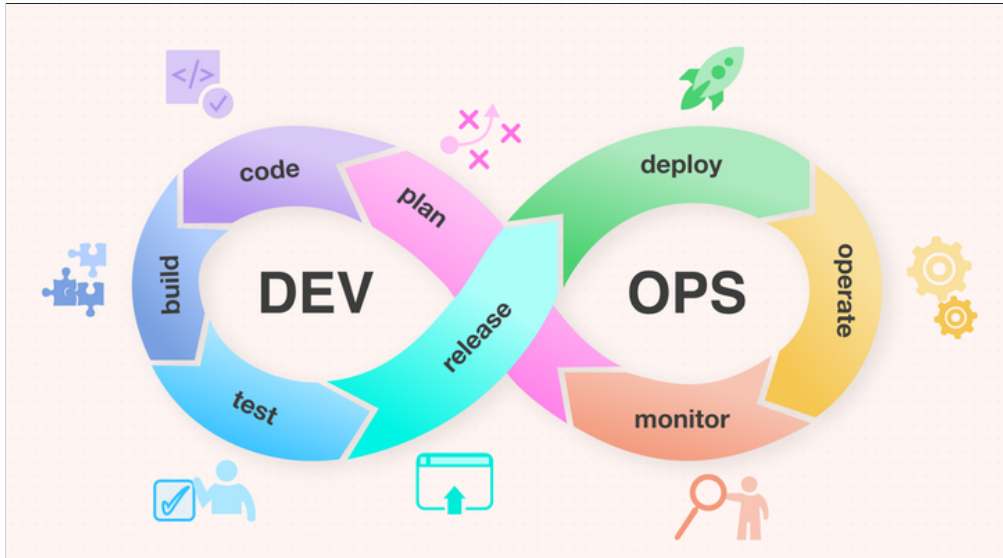
After this brief overview of the cloud computing services, we can conclude that the IaaS and PaaS services are the most relevant to us, so I will implement the deployment plan using both of them and choose the most suitable one.

## IV. DevOps

The goal of DevOps is to increase an organization's speed when it comes to delivering applications and services. Many companies have successfully implemented DevOps to enhance their user experience including Amazon, Netflix, etc. DevOps streamlines the application deployment process. Following each code contribution (commit) to a designated branch, DevOps automates the necessary tasks to ensure the appropriate deployment occurs while ensuring the appropriate testing is completed.

### IV.1. Devops lifecycle

*"DevOps lifecycle is the methodology where professional development teams come together to bring products to market more efficiently and quickly. The structure of the DevOps lifecycle consists of Plan, Code, Building, Test, Releasing, Deploying, Operating, and Monitoring." [4]*



**Figure 2.1:** DevOps lifecycle

- **Plan:** gathering the opinions of end-users by professionals in this level of the DevOps lifecycle.
- **Code:** This is where the development team writes the code for the project.
- **Build:** This is where the development team compiles the code into a common code source.
- **Test:** Various sorts of tests are done such as user acceptability testing, safety testing, speed testing, and many more.
- **Release:** At this level, everything is ready to be deployed in the production environment.
- **Deploy:** In this level, Infrastructure-as-Code assists in creating the operational infrastructure and subsequently publishes the build.
- **Operate:** At this level, the available version is ready for users to use. Here, the department looks after the server configuration.
- **Monitor:** The observation is done at this level that depends on the data which is gathered from consumer behavior.

## IV.2. Implementation of DevOps

There are multiple ways to implement the DevOps methodology into a project:

- **Leveraging Cloud-Based DevOps Solutions:** Cloud providers like AWS, Azure, and GCP offer comprehensive DevOps platforms that integrate seamlessly with their respective



cloud environments. These platforms provide pre-built tools for infrastructure provisioning, configuration management, CI/CD pipelines, and monitoring.

- **Using Open-source DevOps Tools:** DevOps tools like Jenkins, GitLab, Ansible, and Terraform provide the necessary automation and orchestration capabilities to streamline the development and deployment process. These tools can be integrated to create a customized DevOps pipeline tailored to the project's specific requirements.

Given that our web application resides in the cloud, leveraging a cloud-based DevOps solution offered by your chosen Cloud Service Provider (CSP) becomes an ideal approach. Here's why:

- **Seamless Integration:** Cloud-based DevOps solutions integrate seamlessly with the underlying cloud infrastructure, offering pre-configured tools and services specifically designed for that environment. This translates to faster setup, easier management, and optimized performance for your cloud-hosted project.
- **Scalability and Elasticity:** Cloud-based solutions are inherently scalable and elastic. They can automatically adjust resources based on your project's demands, ensuring efficient resource utilization and cost optimization.
- **Managed Services:** Many cloud providers offer managed DevOps services that take care of infrastructure management, patching, and security updates. This frees up your team to focus on core development tasks.

## Conclusion

In this chapter we have explored the fundamentals of cloud computing, DevOps principles, and the tools that empower its implementation. Later on, in the next chapter we will dive into the analytics of the existing infrastructure and the specification of our needs.

# ANALATYCS AND REQUIREMENTS SPECIFICATIONS

---

## Plan

I.	Infrastructure Study . . . . .	15
II.	Deployment Process . . . . .	18
III.	Needs and requirements . . . . .	19
IV.	The implementation of Scrum process . . . . .	20
V.	Description of available tools . . . . .	24

## **Introduction**

This chapter meticulously examines the current environment, including its existing architecture and deployment process. This in-depth analysis will ensure comprehensive coverage of both functional and non-functional needs and requirements. We'll then establish a roadmap for utilizing the Scrum process, defining objectives and deliverables along the way.

### **I. Infrastructure Study**

The company is currently using the baseline web application architecture provided by Microsoft[5]. This figure 3.1 presents the global architecture of the application.

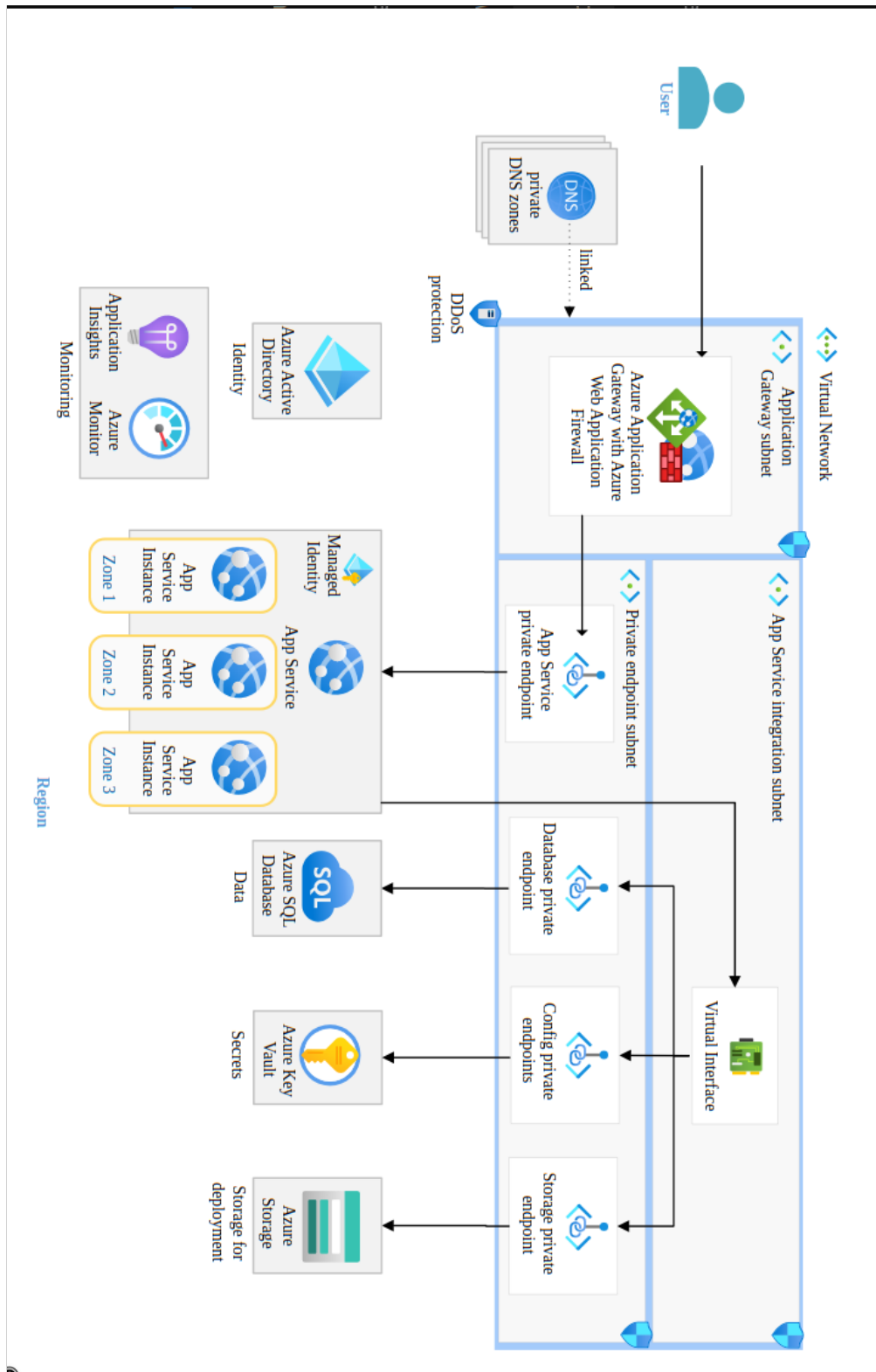


Figure 3.1: global architecture

- **Description**

The architecture exposes a public endpoint via Azure Application Gateway with a Web Application Firewall. The App Service application uses virtual network integration to securely communicate to Azure PaaS services such as Azure Key Vault and Azure SQL Database.

### **I.1. Componants of the architecture**

- **Virtual Network:** This is the fundamental building block for your private network in Azure. It provides isolation and protection for your resources.
- **App Service:** This service is used to host the web application. It provides a fully managed platform for building, deploying, and scaling web apps.
- **Azure SQL Database:** This service is used to store the application data. It provides a fully managed relational database with built-in high availability and security.
- **Azure Key Vault:** This service is used to store and manage application secrets. It provides a secure and centralized storage for application secrets.
- **Azure Application Gateway:** This service is used to protect the web application from common web vulnerabilities. It provides a web application firewall and other security features.
- **Azure Monitor:** This service is used to monitor the health of the web application. It provides logging and application telemetry to monitor the health of the application.
- **Azure DevOps:** This service is used to automate the deployment of the web application. It provides a set of tools for building, testing, and deploying applications.
- **Virtual Interface:** This service is used to connect the web application to the virtual network. It provides a secure and private connection to the web application.
- **Application Insights:** This service is used to monitor the performance of the web application. It provides real-time monitoring and analytics for the web application.
- **Private DNS Service:** This service is used to resolve the DNS names of the Azure PaaS services. It provides a secure and private DNS resolution for the web application.
- **Private endpoint:** This service is used to connect the web application to the Azure PaaS services. It provides a secure and private connection to the Azure PaaS services.

## **I.2. Network flows**

### **Inbound flow:**

- The user issues a request to the Application Gateway public IP.
- The WAF rules are evaluated.
- The request is routed to an App Service instance through the private endpoint.

### **App Service to Azure PaaS services flow:**

- App Service makes a request to the DNS name of the required Azure service. The request could be to Azure Key Vault to get a secret, Azure SQL Database.
- The request is routed to the service through the private endpoint.

### **Deploying to the app service:**

The deployment process is initiated from the Azure portal by the admin from his machine.

## **I.3. Architecture characteristics**

- For security reasons, the network in this architecture has separate subnets for the Application Gateway, App Service integration components, and private endpoints. Each subnet has a network security group that limits both inbound and outbound traffic for those subnets to just what is required.
- The App Service baseline configures authentication and authorization for user identities (users) and workload identities (Azure resources) and implements the principle of least privilege.
- Azure Monitor collects and analyzes metrics and logs from your application code, infrastructure (runtime), and the platform (Azure resources).

## **II. Deployment Process**

The process of releasing a new application version or update in our system follows a multi-step approach. This approach ensures a smooth transition from development to production and minimizes the risk of introducing issues. Here's a breakdown of the key stages involved:

- **Code Preparation:** During this stage, developers finalize the code for the new application version or update. This may involve tasks like code reviews, bug fixing, and integration with existing systems.
- **Testing:** Once the code is prepared, it undergoes rigorous testing by the Quality Assurance (QA) team. This testing verifies that the application functions as intended identifies and resolves any bugs or errors, and ensures compatibility with different environments.
- **Deployment Scheduling:** Following successful testing, a deployment plan is created. This plan defines the specific time and method for releasing the application to users. The plan often involves coordination between development, QA, and system administration teams to ensure a smooth rollout and minimal disruption to ongoing operations.
- **Deployment:** During deployment, the application is transferred from its development environment to the production environment where it will be used by end-users. This process may involve tasks like uploading application files, configuring settings, and integrating with databases or other systems.
- **Monitoring:** After deployment, the system administrators closely monitor the application's performance and functionality. This monitoring helps to identify any issues that may arise after the release and allows for prompt intervention if necessary.

Effective collaboration between development, QA, and system administration teams is crucial throughout this process. Clear communication and well-defined roles ensure a successful application deployment with minimal downtime and a positive experience for end-users.

### III. Needs and requirements

To be able to improve the current deployment process while ensure the security and performance of our applications, we have identified the following needs and requirements:

#### III.1. Functional Needs

- **Patch Deployment Automation:** Implement a system for automated deployment of patches to applications and systems.

- **Version Rollback Capability:** Provide the ability to revert to a previous version of an application during deployment in case of failure or unexpected issues.

### III.2. Non-functional Needs

- **Availability:** Ensure high availability of applications and services, minimizing downtime during deployments.
- **Security:** Implement Web Application Firewall (WAF)) to safeguard applications from cyber threats.
- **Performance:** Optimize deployment processes to maintain optimal performance levels of applications and systems.
- **Cost optimization:** Minimize the costs associated with deployment processes, including recourses and infrastructure.

## IV. The implementation of Scrum process

This overview outlines how we will be implementing the Scrum process to manage our project:

### IV.1. Scrum Roles

**Tableau 3.1:** the key technical objectives for the project

Roles	Members
Scrum Master	Mme. Yosra najjar
Product owner	Mr. Yazid Missaoui
Developer team	Mr. Mohamed ben hadj nasr

### IV.2. Scrum events

- **Retrospective (Weekly):** Held with the company supervisor to review the previous sprint, identify areas for improvement, and adapt the process for the upcoming sprint. This meeting can take place in person at the supervisor's office or virtually using Microsoft Teams.
- **Sprint Planning (Before Each Sprint):** Select items from the backlog and commit to delivering them during the upcoming sprint (a short iteration typically lasting 2 weeks).



### IV.3. Scrum implementation

- **Product Backlog:** I will maintain a product backlog that outlines all the features and functionalities desired in the final product.
- **Sprint Planning:** Before each sprint, a sprint planning meeting is held by the supervisor and me to select a set of items from the top of the product backlog that we commit to delivering during the upcoming sprint. This selection is called the sprint backlog.
- **Sprint Execution:** I deliver the items in the sprint backlog. with each sprint lasting two weeks.
- **Sprint Retrospective:** At the end of each sprint, I will hold a sprint retrospective meeting with the company supervisor. This meeting will focus on reviewing what areas need improvement in the upcoming sprint.

### IV.4. Product backlog

By analysing the needs and requirements, we have defined the following product backlog that will be provided during multiple sprints:

**Tableau 3.2:** the key technical objectives for the project

num	epics	effort
1	Define the structure and variables for the web Terraform module	4 days
2	Define the structure and variables for the database Terraform module	4 days
3	Define the structure and variables for the storage account Terraform module	2 days
4	Implement the DNS zones for each module and set up the connections between them.	4 days
5	Configure security groups for resources in each module.	1 days
6	Define environment-specific variables for the rest of the environments: QA and prod.	4 days
7	Deploy a simple web application to test the infrastructure.	1 days
8	write the necessary documentation on how to use the different workspaces.	2 days
9	Test connectivity between the web application and the database for each environment.	3 days
10	Configure the production environment for optimal performance.	2 days
11	research the different branching strategies and pick the appropriate one.	2 days
12	create the build pipeline and publish the artifacts for each environment.	4 days

13	create the provisioning pipeline for the development environment	4 days
14	set up the management of the state file of the terraform in the pipeline	2 days
15	create the release pipelines for the staging and production environments	3 days
16	test the pipeline by pushing code to the different branches and checking the results	2 days
17	research the different deployment strategies and select the most appropriate one	3 days
18	research the different rollback strategies and select the most appropriate one	3 days
19	compare the different ways to implement the deployment strategy.	2 days
20	implement the deployment strategy in the production release pipeline	4 days
21	implement the rollback strategy in the production release pipeline	4 days
22	test the downtime of the deployment during an update.	2 days
23	test the rollback procedure.	2 days

#### IV.5. Definition of Done

The definition of done is a checklist of requirements that must be met before a task can be considered complete.

- the deployment must be done with no manual intervention.
- the production environment must not be down for more than 3 minutes during the deployment.
- the rollback procedure must be tested and working in under than 4 minutes.
- commits in the dev branch must result in the creation of a new testing environment.
- the infrastructure must provisioned in a maintainable and reproducible code.

#### IV.6. Sprint planning

sprint planning can be found in the gantt diagram below:

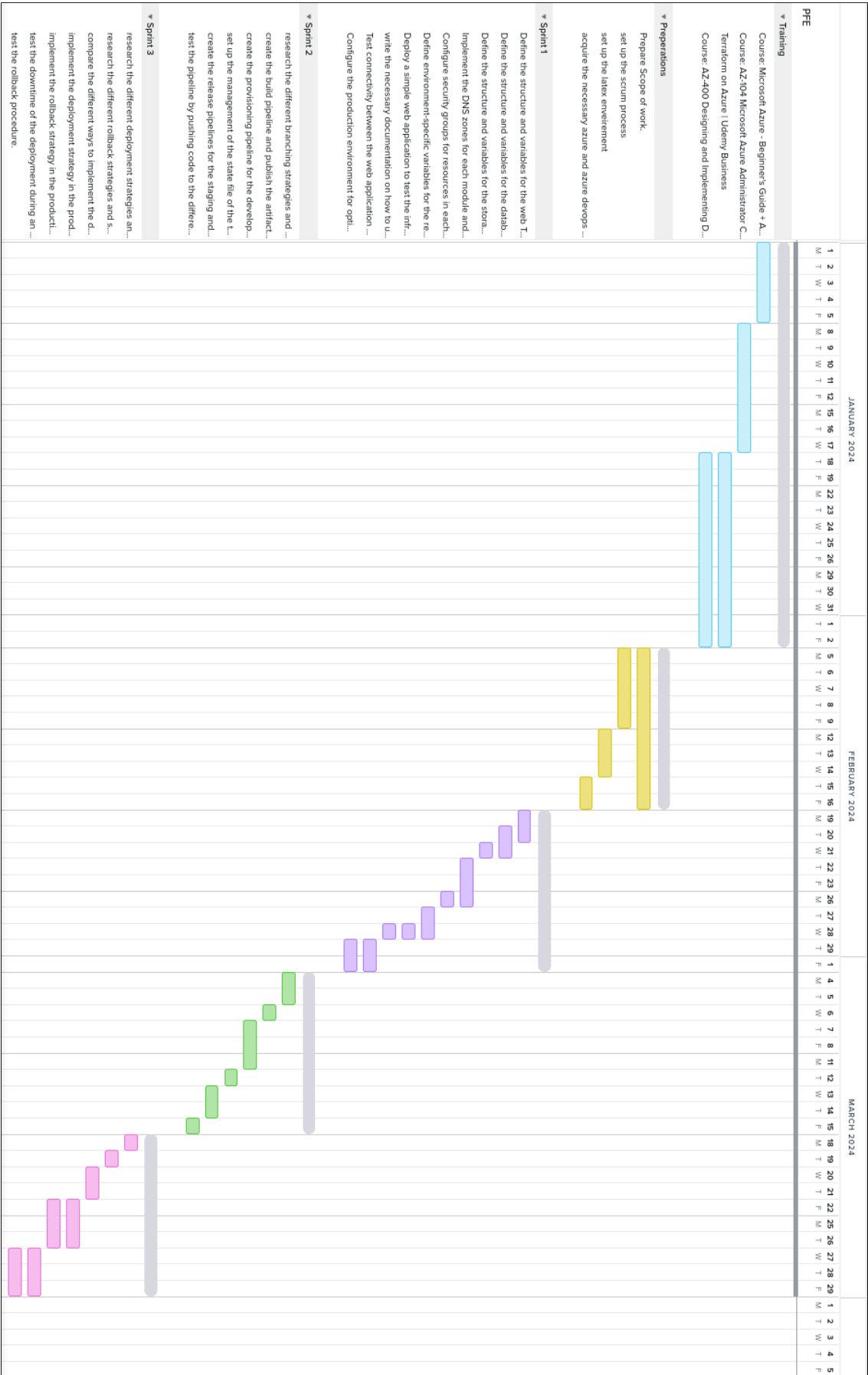


Figure 3.2: This is the gantt chart of the scrum process

## V. Description of available tools

### V.1. Terraform



**Figure 3.3:** Terraform

Imagine building your software on a foundation pre-designed with specific instructions, rather than individually placing each brick. This is the essence of Terraform, an open-source IaC tool. It allows you to define the infrastructure your application needs using a simple language, similar to writing instructions. This simplifies managing resources across different environments (cloud-based or on-premises) with consistent configurations, ensuring everything is built according to your specifications.

#### **Alternatives:**

- **Azure Resource Manager (ARM) templates:** These templates are native to Azure, offering familiarity and direct management within the platform. However, they require more technical knowledge and lack the flexibility and reliability of IaC tools like Terraform.
- **Bicep** Think of Bicep as a specialized architect fluent in Azure, Microsoft's cloud platform. It speaks Azure's language directly, making it easier to design and manage resources within that specific environment. However, since its expertise is limited to Azure it does not have the community support offered by an open-source project like Terraform.

By understanding these factors, we can make the informed decision that the IaC tool that best suits our requirements is Terraform.

## V.2. Azure DevOps



**Figure 3.4:** Azure DevOps

This platform acts as a comprehensive toolkit for software teams, offering various features to manage the entire development lifecycle efficiently.

### Features:

- **Azure Repos:** This feature keeps your code organized and secure, just like a well-structured library holding all your project versions. It can use either Git or Team Foundation Version Control (TFVC) to manage your code.
- **Azure Pipelines:** This service automates tasks like compiling code, running tests, and deploying new versions, saving time and minimizing errors.
- **Azure Boards:** Planning and tracking progress becomes transparent with this feature. It provides Kanban boards visually displaying tasks, backlogs listing upcoming work, and sprint planning tools.
- **Azure Artifacts:** Sharing reusable components becomes effortless with this feature. Think of it as a shared storage space for code modules, containerized applications, and other resources your team can easily access and reuse across projects.

## Conclusion

This chapter's comprehensive analysis of our cloud environment, infrastructure, and deployment process provides a firm foundation for the next crucial step: constructing a secure, efficient, and automated cloud-based deployment framework. With this groundwork laid, we can confidently

transition to the realization phase, detailed in the next chapter, where we'll navigate the provisioning of our current infrastructure.

# THE PROVISIONING OF THE INFRASTRUCTURE

---

## Plan

I.	Sprint backlog . . . . .	28
II.	Realisation . . . . .	28
III.	Challenges . . . . .	31
IV.	Testing the connections . . . . .	32

## Introduction

The first sprint of this project focused on provisioning the necessary cloud resources and setting the connections between them. In this chapter, we will present how we customized the architecture to our needs and the challenges we faced during the process, and we will also showcase the features implemented in the IaC scripts.

### I. Sprint backlog

**Tableau 4.1:** Sprint 1 backlog

Define the structure and variables for the web Terraform module.	(2 days)
Define the structure and variables for the database Terraform module.	(1 days)
Define the structure and variables for the storage account Terraform module.	(1 days)
Develop Terraform code to provision a virtual network with subnets.	(4 days)
Implement the DNS zones for each module and set up the connections between them.	(2 days)
Configure security groups for resources in each module.	(2 days)
implement the characteristics of the development environment.	(2 days)
Define environment-specific variables for the rest of the environments: QA and prod.	(1 days)
Deploy a simple web application to test the infrastructure.	(2 days)
write the necessary documentation on how to use the different workspaces.	(2 days)
Test connectivity between the web application and the database for each environment.	(1 day)
Configure the production environment for optimal performance.	(1 day)

### II. Realisation

in this sprint we focused on provisioning the cloud infrastructure for our project. This involved defining and configuring the necessary cloud resources, setting up the connections between them. and since the infrastructure will be used in three diffrent distinct situations, the first one being the development environment, in the dev team cant test the application , the second one is the quality assurance (QA) environment, where the application is tested thouroly by the QA team, and the last one is the production environment where the application is deployed to the public. the code I wrote need to adapt depending on the environment while also maintaining modularity and reusability. To promote code reusability, maintainability, and clarity, we adopted a modular



approach for our Terraform configuration. This involved structuring the code into distinct modules, each encapsulating a specific aspect of the infrastructure (e.g., web app, database, storage account). This approach allows us to manage and update each module independently, and to reuse them across different environments. Furthermore, to ensure consistency and streamline configuration changes, we grouped frequently used variables like SKU names, VNet prefixes, and subnet prefixes into a dedicated `local.tf` file. This combined approach of modular structure and centralized variable management promotes efficient and well-organized Terraform configuration, fostering long-term project maintainability and scalability.

this figure 4.1 shows the file structure of the terraform configuration.

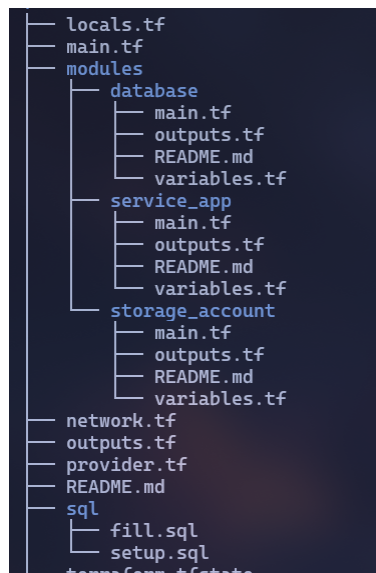


Figure 4.1: The file structure

## II.1. The difference between the environments

using the terraform workspaces to manage the different environments, we can define the differences between the environments.

➤ **dev:** the development environment:

- the resources are the most basic cost to optimize the cost.
- It also has a VM with a public IP inside the virtual network to access the database and the storage account.
- the web app and the VM can only be accessed from the CIDN given in the variables.

➤ **QA:** the quality assurance environment:

- the resources are a bit more expensive to better simulate the production environment.
- It also has a VM with a public IP inside the virtual network to access the database and the storage account.
- it is open to the public.

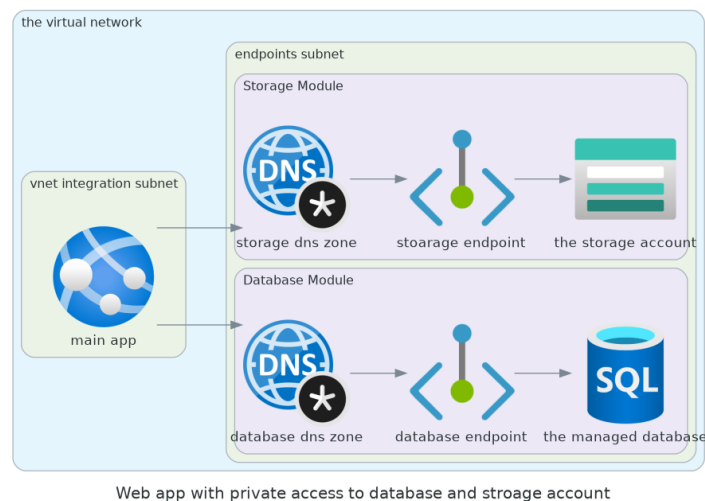
➤ **prod:** the production environment.

- the resources are the most expensive to ensure the best performance.
- it is open to the public.

## II.2. The modified architecture

a slight modification made to the original architecture consists of the removal of the application gateway due to its high price so the user will just have to access the web app directly which is acceptable since we only have one web app.

the changes in the architecture are shown in figure 4.2.

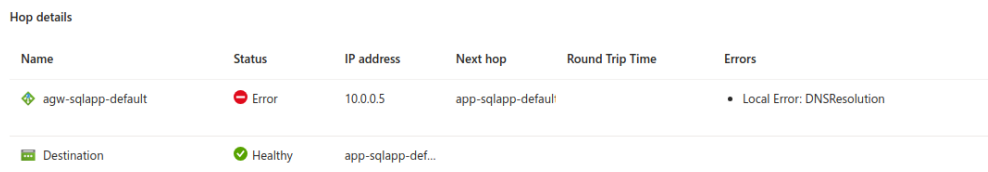






**Figure 4.2:** The new architecture

### III. Challenges

#### ● The problem

The main challenge we faced during Sprint 1 was configuring the DNS zones to enable communication between the web application and the database. Despite successfully provisioning the necessary cloud resources, the web application encountered issues resolving the hostname of the database. This meant the application couldn't locate the database to retrieve or store data, hindering core functionality. Troubleshooting involved verifying several aspects:



Name	Status	IP address	Next hop	Round Trip Time	Errors
 agw-sqlapp-default	 Error	10.0.0.5	app-sqlapp-default		<ul style="list-style-type: none"><li>• Local Error: DNSResolution</li></ul>
 Destination	 Healthy	app-sqlapp-def...			

**Figure 4.3:** connectivity problem

- **DNS record configuration:** We double-checked the DNS record types (likely A record) and their values (database hostname and IP address) within the configured DNS zone. Any typos or incorrect mappings could have caused resolution issues.
- **Network security group (NSG) rules:** We had to verify the NSG rules to ensure the web application could communicate with the database.
- **verify connections:** We had to verify that the database was accessible from the web application. and that the web could access the DNS server.

By systematically examining these potential causes, we were able to identify that the web app did not have access to the DNS server provided by Azure. This experience highlights the importance of careful configuration and understanding of how DNS plays a crucial role in enabling communication between different components within a cloud infrastructure.

#### ● The solution

To address the DNS resolution issue between the web application and the database, we capitalized on a core Azure concept: **the Wire Server**. This managed DNS server, automatically created within each virtual network, plays a critical role in resolving internal DNS queries using private

DNS zones. Although offered as a free service, the Wire Server isn't automatically configured as the default DNS server for the virtual network.

```
resource "azurerm_virtual_network" "the_network" {
  name            = module.naming.virtual_network.name
  location        = local.location
  resource_group_name = azurerm_resource_group.the_group.name
  address_space   = [local.vnet_prefix]
  dns_servers     = ["168.63.129.16"]

  tags = {
    environment = "${terraform.workspace}"
  }
}
```

**Figure 4.4:** The DNS configuration

Our solution involved leveraging Terraform to explicitly set the Wire Server's static IP address (168.63.129.16) as the default DNS server within our virtual network configuration. By making this configuration change, we ensured that the web application could effectively utilize the Wire Server to resolve the database hostname and establish the necessary communication channel. This approach eliminated the initial DNS resolution obstacle and facilitated seamless communication between the application and the database.

## IV. Testing the connections

### • The user to the application

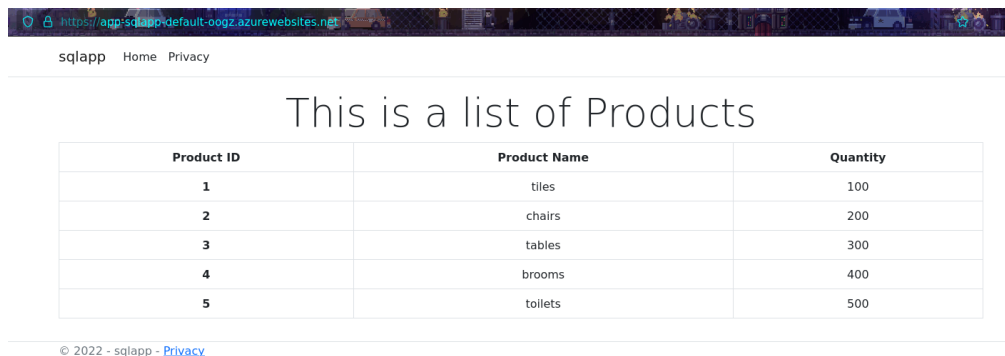
To test the connectivity between the web application and the user, I deployed a simple web application that displays a welcome message. We then accessed the web application using a web browser to verify that the application was accessible from the designated CIDR range in the development environment.

### • The application to the database

To test the connectivity between the web application and the database, I used the Azure Portal to verify that the web application could successfully communicate with the database. This involved checking the database connection status and ensuring that the web application could read and write data to the database.

### ● Integration testing

To ensure that the web application and database could communicate effectively, I performed integration testing by simulating user interactions with the application. This involved submitting data through the web application and verifying that the data was correctly stored and retrieved from the database. By testing the end-to-end functionality of the application, I confirmed that the web application and database were successfully connected and operational.



**Figure 4.5:** the simple web app that was used to test the infrastructure.

## Conclusion

This chapter detailed the successful completion of Sprint 1, focusing on provisioning the cloud infrastructure for our project. We presented a modified architecture that removed the application gateway due to cost considerations. The new architecture leverages Terraform modules for efficient code organization and utilizes workspaces to manage different environments (dev, QA, prod) with appropriate resource configurations.

We encountered a challenge during this sprint related to DNS configuration, where the web application struggled to resolve the database hostname. This issue was resolved by leveraging the Azure Wire Server as the default DNS server within the virtual network configuration.

By successfully provisioning the infrastructure and addressing the DNS resolution challenge, Sprint 1 laid the foundation for future sprints to focus on building and deploying the web application and its functionalities within the cloud environment as will be detailed in the next chapter.

# CONTINUOUS INTEGRATION AND DEPLOYMENT

---

## Plan

I.	Sprint backlog . . . . .	35
II.	Version Control strategy . . . . .	35
III.	The CI/CD pipelines . . . . .	39
IV.	Challenges Faced . . . . .	42

## Introduction

This chapter delves into the core components that establish a robust CI/CD pipeline for our project. We'll explore the setup process, delve into the selection of a branching strategy, and dissect the configuration of pipelines designed for development, staging, and production environments. Each stage within these pipelines will be meticulously explained, highlighting its function and contribution to the overall workflow.

### I. Sprint backlog

**Tableau 5.1:** Sprint 2 backlog

research the different branching strategies and pick the appropriate one for this project	(2 days)
create the build pipeline and publish the artifacts for each environment	(3 days)
create the provisioning pipeline for the development environment	(3 days)
set up the management of the state file of the terraform in the pipeline	(2 days)
create the release pipelines for the staging and production environments	(3 days)
test the pipeline by pushing code to the different branches and checking the results	(2 days)

### II. Version Control strategy

- **The different branching strategies[6]**

**trunc based development:** In this strategy, requires no branches but instead, developers integrate their changes into a shared trunk at least once a day. This shared trunk should be ready for release anytime.

- **Advantages:** Have better visibility over what changes other developers are making as commits are made directly into the trunk without the need for branches.
- **Desadvantages:** It can be difficult to monitor the quality of the codebase as there are no branches to isolate changes. This approach can be daunting for junior developers as they are interacting directly with the shared trunk.

**github flow:** In this strategy, there is only one branch called the main branch. Developers create feature branches from the main branch, work on their features, and then create a pull request

to merge their changes back into the main branch.

- **Advantages:** This strategy is particularly suited for small teams and web applications and it is ideal when you need to maintain a single production version.
- **Desadvantages:** The lack of development branches makes this strategy more susceptible to bugs and so can lead to an unstable production code if branches are not properly tested before merging with the master-release preparation and bug fixes happen in this branch.

**gitflow:** In this strategy, there are two main branches: the master branch and the develop branch. The master branch contains the production-ready code while the develop branch contains the latest code that is ready for release.

- **Advantages:** Perhaps the most obvious benefit of this model is that it allows for parallel development to protect the production code so the main branch remains stable for release while developers work on separate branches.
- **Desadvantages:** Due to GitFlow's complexity, it could slow down the development process and release cycle. In that sense, GitFlow is not an efficient approach for teams wanting to implement continuous integration and continuous delivery.

**gitlab flow:** is a simpler alternative to GitFlow that combines feature-driven development and feature branching with issue tracking. GitLab Flow is great when you want to maintain multiple environments and when you prefer to have a staging environment separate from the production environment.

- **Advantages:** It promotes teamwork and emphasizes code quality with a lean approach, encouraging practices like unit testing and CI/CD.
- **Desadvantages:** With more frequent integration, there's a higher chance of merge conflicts, especially in larger teams.

**Tableau 5.2:** the different situations where each branching strategy is suitable

Product type and its release method	Team size	Applicable branching strategy
All	Small	Trunk based development



Products that support continuous deployment and release, such as SaaS products	Middle	GitHub-Flow and TBD
Products with a definite release window and a periodic version release cadence, such as iOS apps	Middle	Git-Flow and GitLab-Flow with release branch
Products that are demanding for product quality and support continuous deployment and release, such as basic platform products	Middle	GitLab-Flow
Products that are demanding product quality and have a long maintenance cycle for released versions	Large	Git-Flow

**Verdict:** Considering the table’s comparison and the advantages and disadvantages listed, GitLab Flow emerges as the optimal choice for our project’s workflow since we can leverage its strengths to establish a smooth, efficient, and quality-focused development process for our continuous deployment pipeline.

- **Description of the GitLab flow strategy**

the GitLab flow strategy consists of 3 main branches: the main branch(develop), the staging branch(pre-prod), and the production branch(prod).

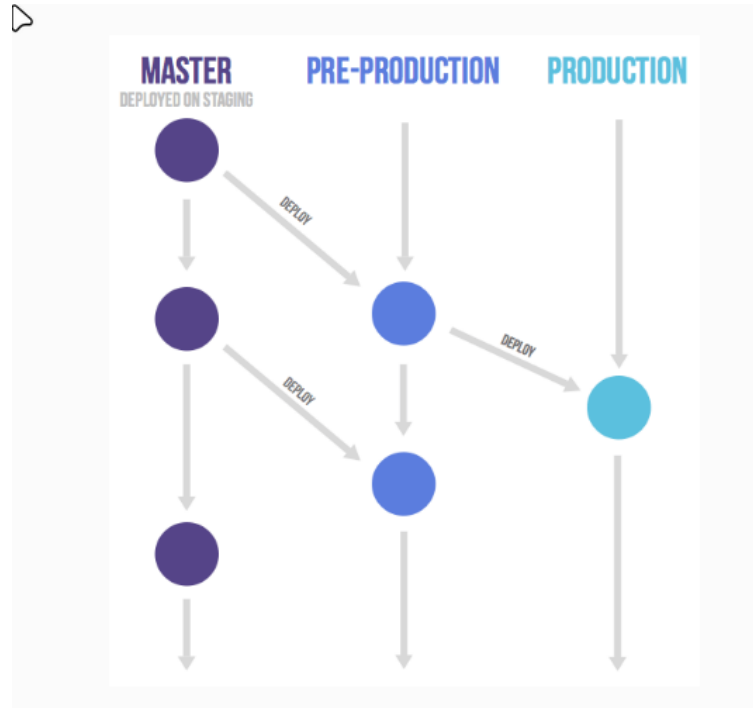


Figure 5.1: gitlab flow strategy

- **Main branches**

- **the main branch(develop):** This branch is the default branch and contains the latest code that is ready for release. Developers create feature branches from the main branch, work on their features, and then create a pull request to merge their changes back into the main branch.
- **the staging branch(pre-prod):** This branch is used for testing the code by the QA team before it is deployed to the production environment. The staging branch is created from the main branch and is used to test the code in a production-like environment.
- **the production branch(prod):** This branch contains the production-ready code that is deployed to the production environment. The production branch is created from the staging branch and is used to deploy the code to the production environment.

- **Secondary branches**

these branches are short-lived and are used to develop new features or fix bugs.

- **feature branches:** These branches are created from the main branch and are used to develop new features. Once the feature is complete, a pull request is created to merge the changes back into the main branch.

- **Hotfix branches:** These branches are created from the production branch and are used to fix critical bugs in the production code. Once the bug is fixed, a pull request is created to merge the changes back into the production branch.

### III. The CI/CD pipelines

- **The build pipeline**

here's a detailed breakdown of the build pipeline for the .NET Core application:

- **Fetch the code:** The pipeline starts by fetching the latest code from your GitHub repository or a pull request. This is done using a service connection that links our Azure DevOps account to the GitHub repository.
- **Build the code:** This stage involves two main tasks:
  - **Restore dependencies:** The pipeline installs NuGet and restores the packages required for the project. Nuget is the package manager responsible for managing the various dependencies of this application.
  - **Build the project:** The pipeline builds the .NET Core project using the dotnet build command.
- **Run tests:** This stage involves running the unit tests and the integration tests to ensure the application functions as expected.
- **Publish artifacts:** The pipeline publishes the build artifacts to the Azure DevOps server. These artifacts are to be used by the release pipeline to deploy the application to the different environments.

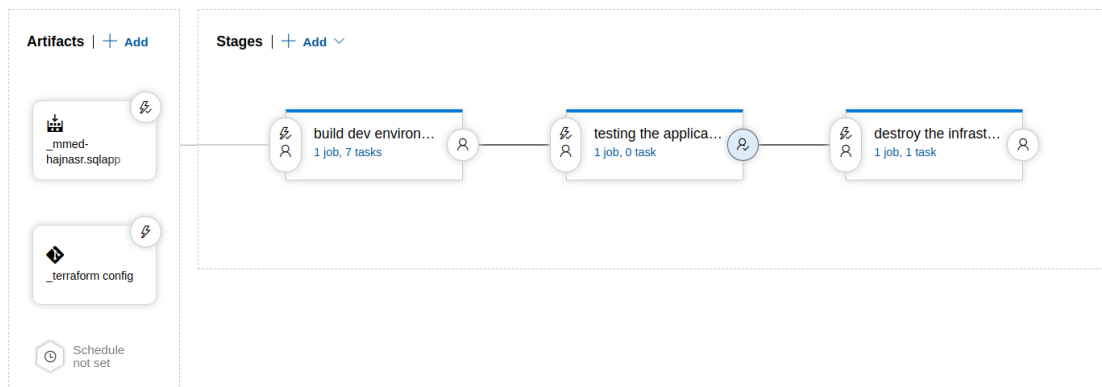
- **The release pipeline**

the release pipeline is the combination of the provisioning and deployment process. Here's a detailed breakdown of the different pipelines for the different environments:

- **The development environment**

the dev environment is used for testing the code by the developers when a push request is made into the main branch and approval from another developer is required. it consists of the following stages:

- **Fetch the artifacts:** The pipeline starts by fetching the latest artifact that contains the build of the application and the artifact that contains the terraform source code.
- **Deploy the infrastructure:** The pipeline deploys the infrastructure using the terraform source code using the service connection to the Azure subscription.
- **assign the variables:** The pipeline assigns the variables required for the deployment of the application like the name of the web app that is generated during deployment.
- **deploy the application:** The pipeline deploys the application to the web app using the artifact that contains the build of the application.
- **the approval stage:** After the pipeline notifies the responsible developer it waits for his approval to merge the code into the main branch after he is done reviewing the changes using the deployed application.
- **the destruction of the infrastructure:** This stage is used to destroy the infrastructure after the request has been approved or rejected.



**Figure 5.2:** development pipeline

**the benefits of this pipeline are:**

- **fast feedback:** The pipeline provides fast feedback to the developers on the quality of the code.
- **cost savings:** The pipeline saves costs by destroying the infrastructure after the request has been approved or rejected. so that the infrastructure doesn't keep running when not needed.

- **multiple deployments:** The pipeline allows for multiple deployments in different infrastructures at once due to the randomly generated names. giving us the ability to test multiple pull requests at once.

- **The staging and the deployment pipelines**

These pipelines share similar stages but differ in their triggers:

- **Staging Pipeline:** Triggered by a push or merge to the pre-production (pre-prod) branch. This environment mirrors production as closely as possible for testing purposes.
- **Deployment Pipeline:** Triggered by a push or merge to the production (prod) branch. This pipeline deploys the application to the live environment used by end users.

The stages within these pipelines typically involve:

- **Fetch the artifacts:** Similar to the development pipeline, this stage retrieves the latest Terraform configuration files and the recently built application artifact.
- **Deploy the infrastructure:** In the staging and production environments, the infrastructure is already deployed using the terraform files. so we directly deploy the application to the web app.
- **Notify Deployment Status:** Once the deployment is complete, the pipeline automatically notifies the designated manager (or team) about the success or failure of the process.

**the benefits of these pipelines are:**

- **Increased Efficiency:** Automating deployments frees up valuable developer and operations time. They can focus on core activities like building new features, improving code quality, and resolving critical issues.
- **Reduced Risk of Errors:** Manual deployments are prone to human error. Staging and deployment pipelines automate the entire process, minimizing the chance of mistakes that could lead to application downtime or malfunctions.
- **Faster Release Cycles:** By automating the deployment process, teams can push new features and bug fixes to staging and production environments much quicker. This eliminates the time spent on manual tasks and allows for more frequent releases, keeping your software up-to-date and competitive.

## IV. Challenges Faced

### ● Problem(Secure Terraform State Management in a Multi-Stage Pipeline)

A critical hurdle encountered during CI/CD pipeline implementation was handling Terraform state files across stages. These files, generated during infrastructure deployment, contain crucial details about the deployed infrastructure. Terraform relies on them to determine the current state for subsequent actions and this file was required by Terraform to destroy the infrastructure.

The initial solution involved saving the state file in an Azure storage account. However, this necessitated granting pipeline access to the storage, incurring potential costs.

### ● Solution

Further exploration revealed a more efficient approach: utilizing the Azure CLI to destroy deployed resources. By relying solely on the readily accessible resource group name, the Azure CLI could effectively destroy resources after approval/rejection decisions, eliminating the need for storage access and associated costs.

## Conclusion

The meticulously crafted CI/CD pipeline significantly enhances the development process by automating deployments and infrastructure management. This automation not only minimizes manual work and the potential for errors but also facilitates faster deployments and cost savings through efficient infrastructure utilization. Our selection of the GitLab Flow branching strategy fosters collaboration and ensures code quality throughout the development lifecycle.

In conclusion, our new CI/CD pipeline empowers our team to deliver features and updates with remarkable efficiency while maintaining an unwavering commitment to quality. As we move forward, the next chapter will explore the various deployment strategies available and delve into the process of selecting and implementing the most suitable option for our specific project requirements.

---

# THE DEPLOYMENT STRATAGY

---

## Plan

I.	Introduction . . . . .	44
II.	Sprint backlog . . . . .	44
III.	The different deployment strategies . . . . .	44
IV.	The different ways to implement the deployment strategy . . . . .	47
V.	The rollback strategy . . . . .	50
VI.	Overview of the flow of the deployment strategy . . . . .	51
VII.	The monitoring strategy . . . . .	53

## I. Introduction

in this day and age, users expect uninterrupted access to websites around the clock. To ensure this high level of availability, selecting the right deployment strategy is critical. This strategy should facilitate seamless updates with minimal to no downtime for the website. Furthermore, a robust rollback procedure should be implemented to address any unforeseen issues that may arise during deployment. This chapter will delve into the specific deployment strategy chosen for this project, along with the rollback procedure designed to ensure a smooth recovery process in the event of failure. and the monitoring tools that will be used to monitor the web application once deployed.

## II. Sprint backlog

**Tableau 6.1:** Sprint 3 backlog

research the different deployment strategies and select the most appropriate one	(3 days)
research the different rollback strategies and select the most appropriate one	(2 days)
compare the different ways to implement the deployment strategy.	(1 days)
implement the deployment strategy in the production release pipeline	(2 days)
implement the rollback strategy in the production release pipeline	(2 days)
test the downtime of the deployment during an update.	(2 days)
test the rollback procedure.	(1 days)

## III. The different deployment strategies

A deployment strategy is a way to change or upgrade an application. The aim is to make the change without downtime in a way that the user barely notices the improvements.

Now that we know what a deployment strategy is, let's dive into the different types of deployment strategies:

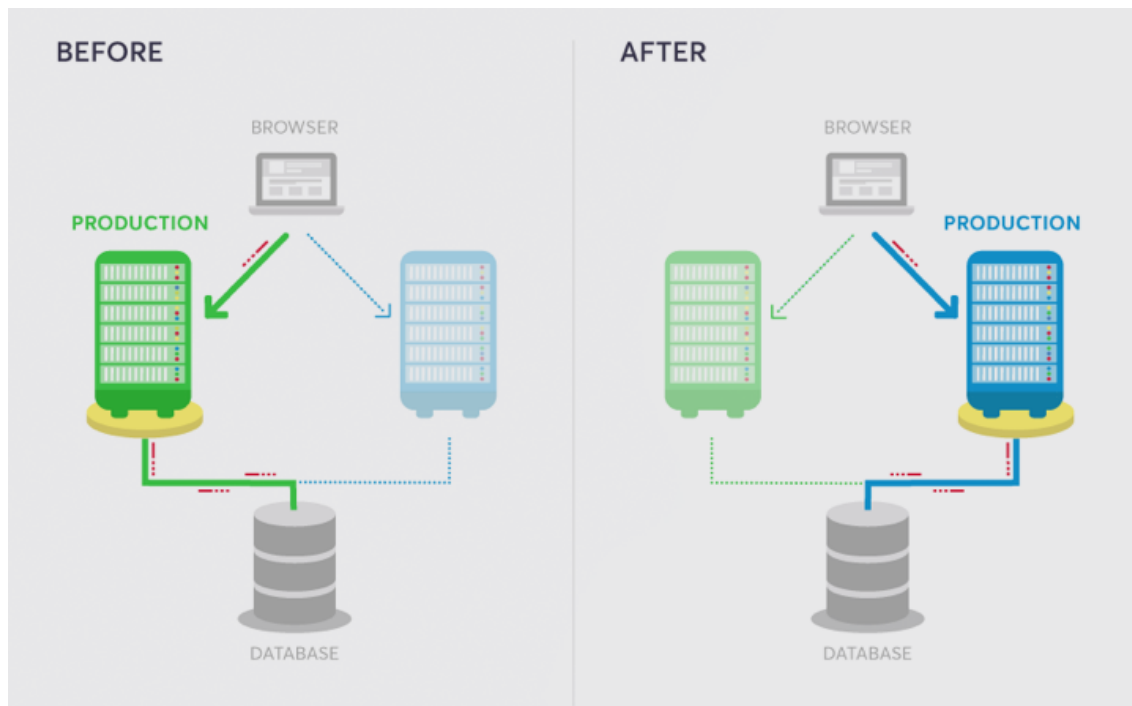
- **Blue-green deployment**

*"This deployment strategy involves running the new version of the software alongside the old version. Stable or older versions of the application are always blue while newer versions are green. When the new version has been tested and certified to meet all the requirements, the load balancer automatically switches traffic from the older version to the newer one."*[7]



**Advantages:** This strategy offers a quick update with simplicity and minimal impact on the users.

**Desadvantages:** expensive since both the new and old versions must run simultaneously.



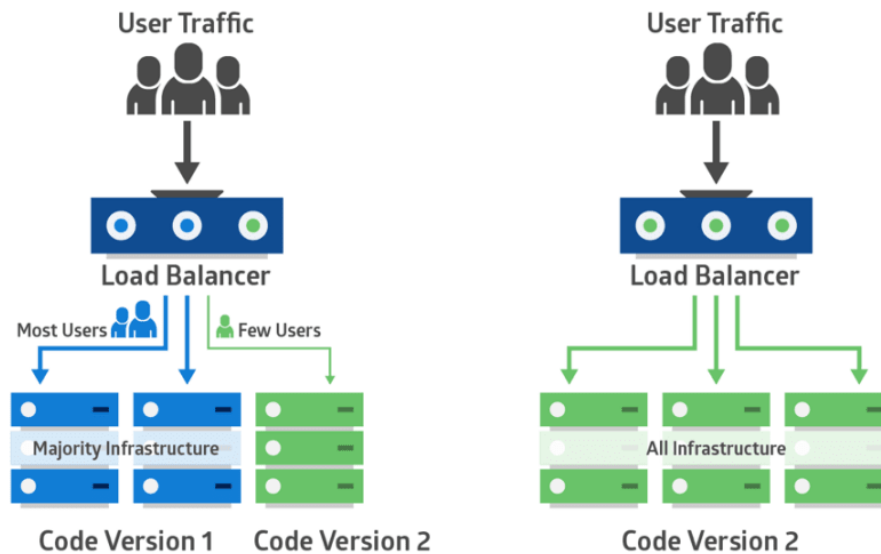
**Figure 6.1:** blue-green deployment strategy.

- **Canary deployment**

During canary deployment, the team responsible for deployment gradually redirects traffic from the older version to the new one. This is done in small increments, allowing the team to monitor the new version's performance and quickly roll back if necessary.

**Advantages:** This strategy allows for better performance monitoring as well as faster and easier software rollbacks.

**Desadvantages:** It has a slow deployment cycle and requires more time.



**Figure 6.2:** canary deployment strategy.

- **Ramped/Rolling deployment**

The ramped deployment strategy gradually changes the older version to the new version. Unlike canary deployment, the ramped deployment strategy replaces instances of the old application version with instances of the new application version one instance at a time.

**Advantages:** This strategy costs less than canary and blue-green deployment strategies since every instance not used is destroyed.

**Desadvantages:** In case of an unexpected event, the rollback process takes longer than the other strategies.

- **A/B testing**

Developers deploy the new version alongside the older version in A/B testing deployment. However, this new version is only available to a limited number of users, who are selected according to certain conditions and parameters. Location, device type, UI language, and operating system can serve as parameters for selecting these users.

**Advantages:** The use of real-time statistical data can help developers make informed decisions about their software.

**Desadvantages:** This strategy is complex and requires a high-end load balancer that can access the application layer.

- **Verdict**

Given our need for swift deployments and near-instant rollbacks, coupled with a cost-effective and manageable infrastructure, a blue-green deployment strategy is the ideal choice.

## IV. The different ways to implement the deployment strategy

- **Utilizing Infrastructure as Code (IaC) and Traffic Management**

this method needs these steps to be implemented in the pipeline:

- Create a green environment using IaC tools.
- deploy the new version of the application to the green environment.
- Use the traffic management resource to switch the traffic from the blue environment to the green environment.

**Advantages:** This method is cost-effective and more customizable(because you can choose any SKU for the requirements).

**Desadvantages:** It takes longer during updates since we need to create an environment each time. Also, it adds the complexity of adding a load balancer to the infrastructure.

- **Leveraging Azure App Service Deployment Slots**

Deployment slots are live apps with their host names. App content and configuration elements can be swapped between two deployment slots, including the production slot.

- Create a deployment slot for the new version of the application.
- Deploy the new version of the application to the deployment slot.
- Swap the deployment slot with the production slot.

**Advantages:** It is way simpler to implement than the first method, and it is faster during updates.

**Desadvantages:** This method requires at least the Standard SKU for the App Service Plan.

- **Verdict**

since this strategy is implemented in the production environment choosing the Standard SKU is acceptable. So to avoid unnecessary complexity and to speed up the deployment process, we will

use the Azure App Service Deployment Slots method.

Agent job		Started: 4/7/2024, 7:46:12 AM
Pool: <a href="#">Hosted Windows 2019 with...</a> · Agent: Hosted Agent		... 3m 46s
✓ Initialize job · succeeded		4s
✓ Download artifact - _mmed-hajnasr.sqlapp (1) - sqlapp-pre-prod-artifact · succeeded		4s
✓ Azure App Service Deploy: app-sqlapp-preprod-qog0 · succeeded		25s
✓ Swap Slots: app-sqlapp-preprod-qog0 · succeeded		3m 8s
✓ Finalize Job · succeeded		<1s

**Figure 6.3:** Azure App Service Deployment Slots implementation.

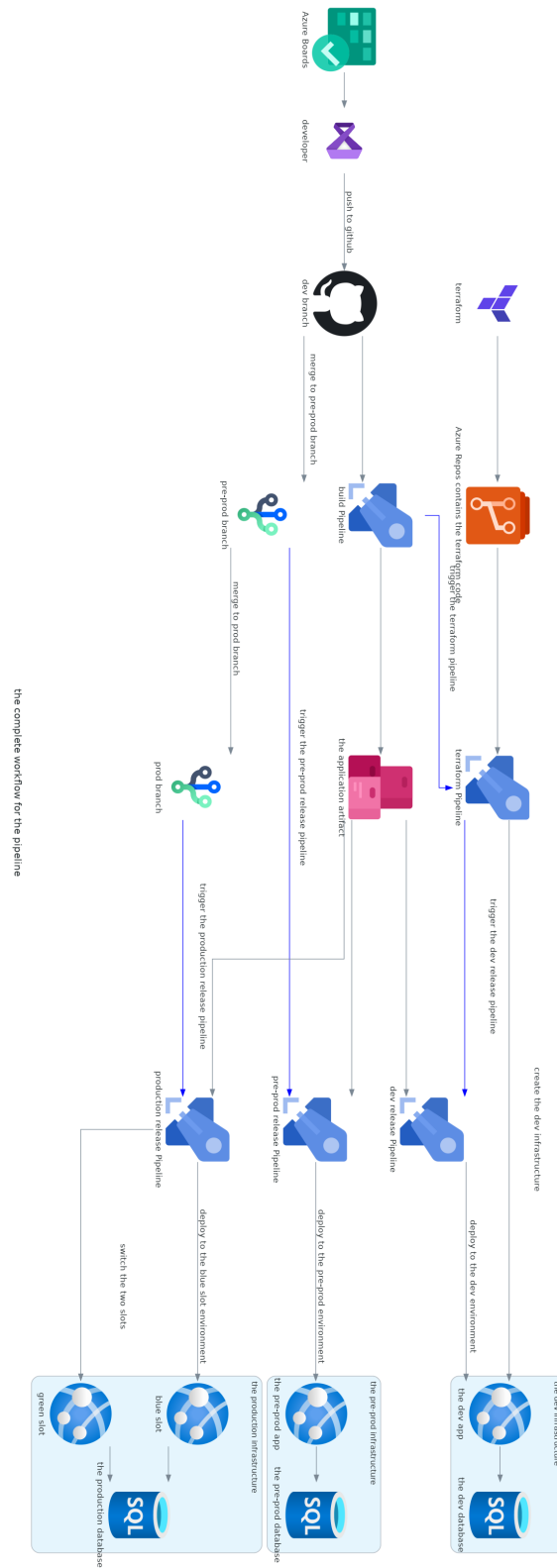


Figure 6.4: the full workflow of the deployment strategy.

## V. The rollback strategy

the deployment pipeline incorporates a robust rollback strategy to ensure minimal disruption and user satisfaction.

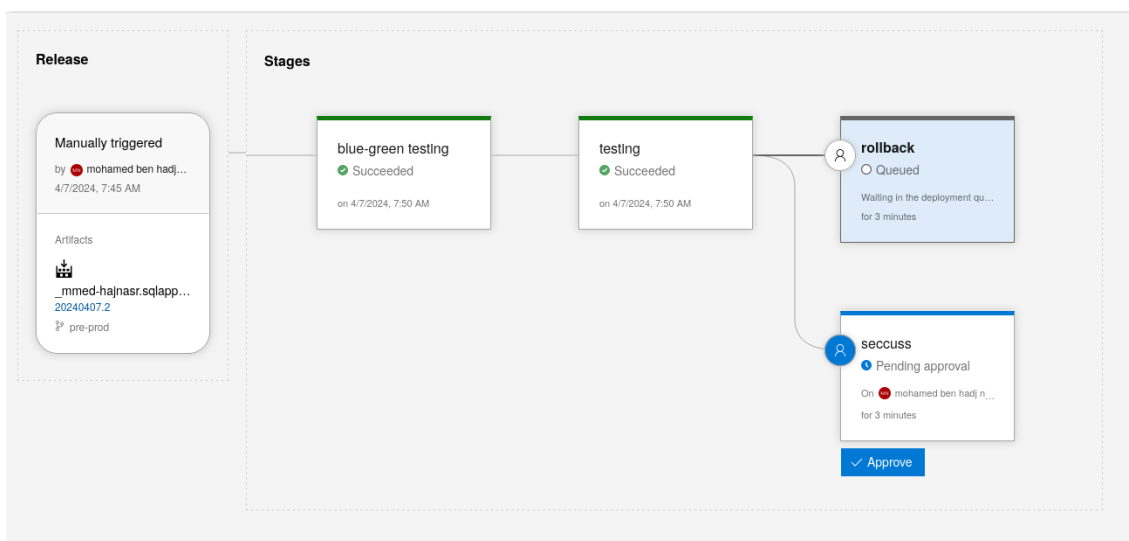
### ● Deployment process

- **New Version Deployment:** The new version is deployed through the pipeline.
- **Automatic Rollback on Deployment Failure:** Any errors encountered during deployment will trigger an automatic rollback. This ensures the system quickly reverts to the previous stable version, minimizing downtime and potential data loss.
- **User Feedback and Rollback Option:** Users have the opportunity to provide feedback on the new version. If a significant number of users express dissatisfaction, an admin can initiate a rollback.

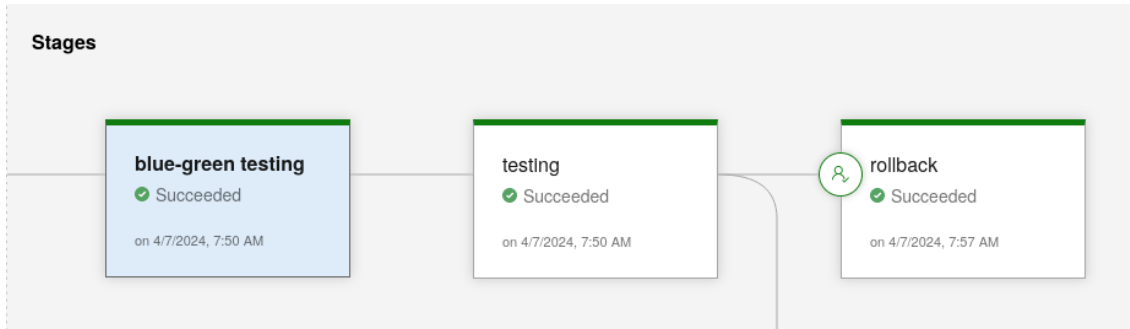
### ● The rollback retention

**30-Day Rollback Window:** The pipeline, including the rollback option, remains accessible for 30 days after deployment. This window provides ample time for user feedback and admin decision-making.

**Automatic Rollback Rejection After 30 Days:** If no rollback is initiated within 30 days, the rollback stage is automatically deactivated. This prevents holding onto outdated rollback options indefinitely and we can delete the old version slots to avoid extra costs.



**Figure 6.5:** the rollback strategy implementation



**Figure 6.6:** seccuss rollback

## VI. Overview of the flow of the deployment strategy

This diagram follow bpmn to show what happens in each branch during a commit.

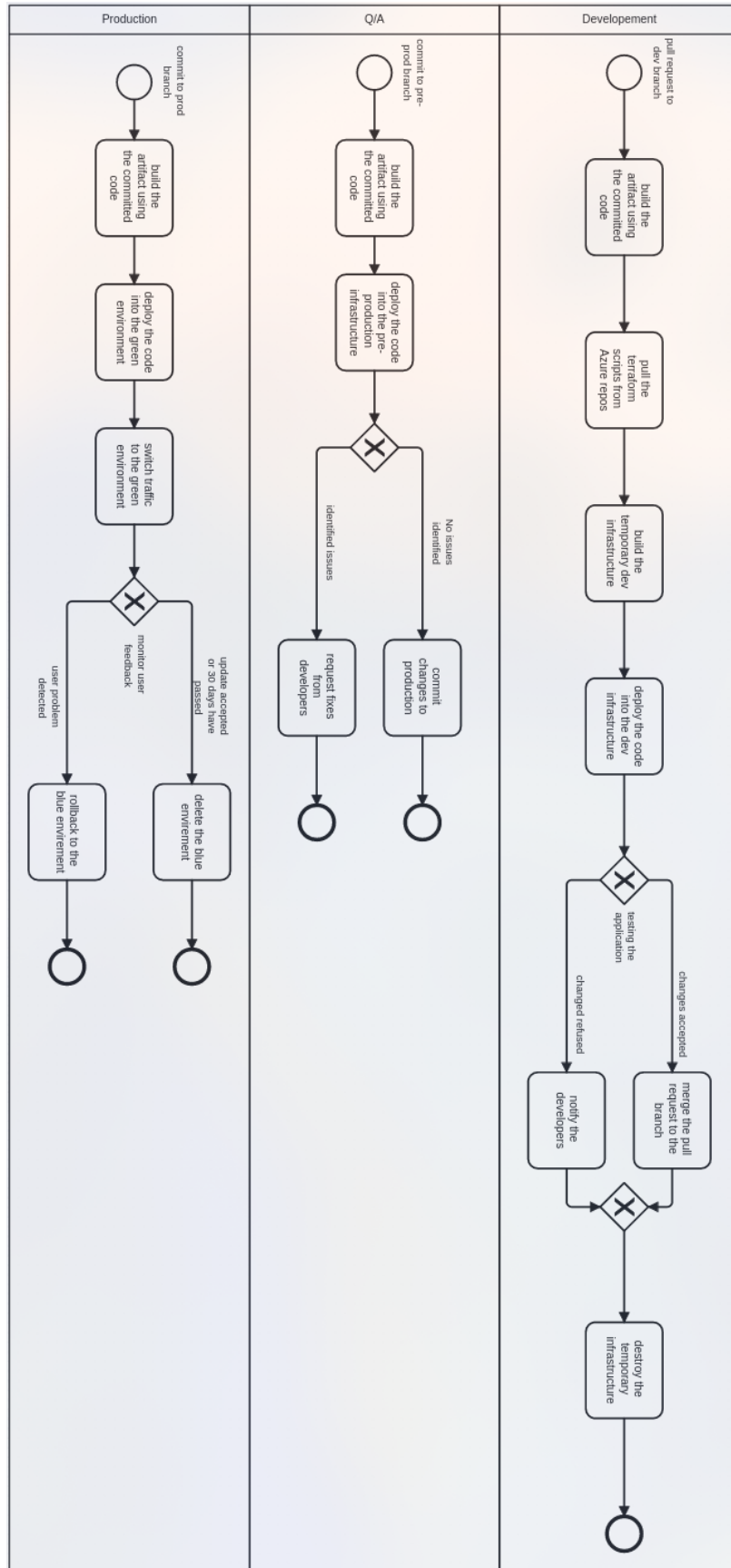


Figure 6.7: the flow of the deployment strategy



## VII. The monitoring strategy

we will use different monitoring tools offered by Azure to monitor the web application once deployed [8]:

- **Azure Health check:** This tool will monitor the health of the web application and notify the admin if any issues arise.
- **Insights:** Application Insights monitors the availability, performance, and usage of your web applications, so you can identify and diagnose errors without waiting for a user to report them.

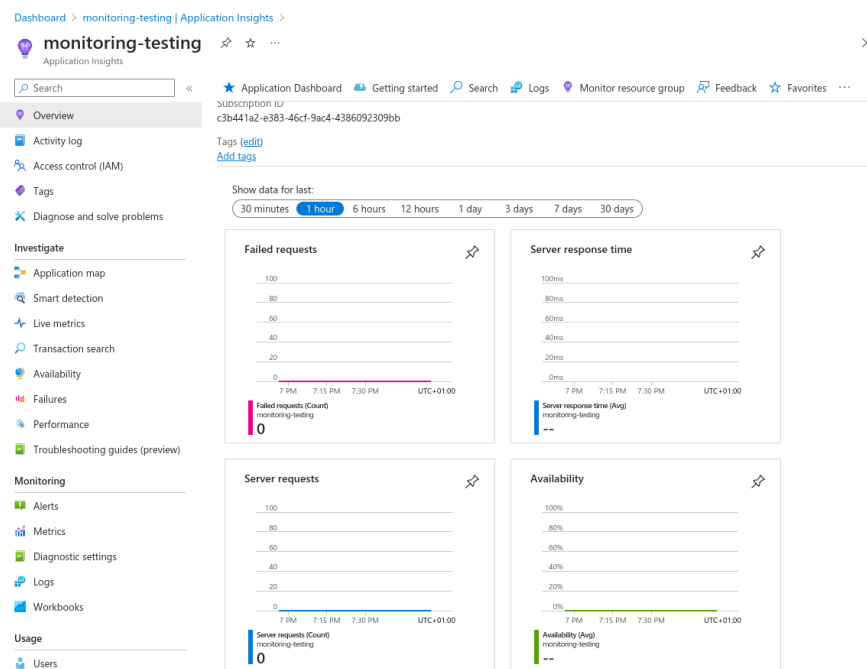
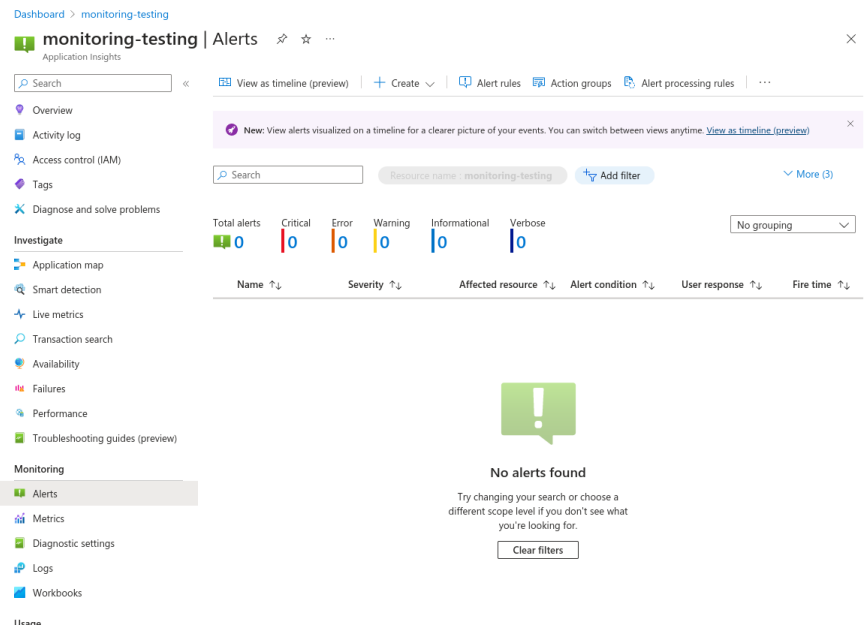


Figure 6.8: monitoring insights set up

- **Alerts:** Alerts can be set up to notify the admin of any issues that arise with the web application. Unlike Health check, this option is highly customizable.



**Figure 6.9:** in this dashboard, you can check the status of the alerts.

**Create an alert rule**

Scope **Condition** Actions Details Tags Review + create

Configure when the alert rule should trigger by selecting a signal and defining its logic.

Signal name \*  [See all signals](#)

**Alert logic**

**Threshold** ☒ Static ☐ Dynamic

**Aggregation type**

**Operator**

**Threshold value** \*  %

**Split by dimensions**

Use dimensions to monitor specific time series and provide context to the fired alert. Dimensions can be either number or string columns. If you select more than one dimension value, each time series that results from the combination will trigger its own alert and will be charged separately. [About monitoring multiple time series](#)

Dimension name	Operator	Dimension values	Include all future values
<input type="text" value="Select dimension"/>	<input type="text" value="="/>	<input type="text" value="0 selected"/> <a href="#">Add custom value</a>	<input type="checkbox"/>

**Figure 6.10:** here you can set up the alerts.

- **Metrics:** Metrics provide a detailed view of the web application's performance, including response time, request rate, and error rate.

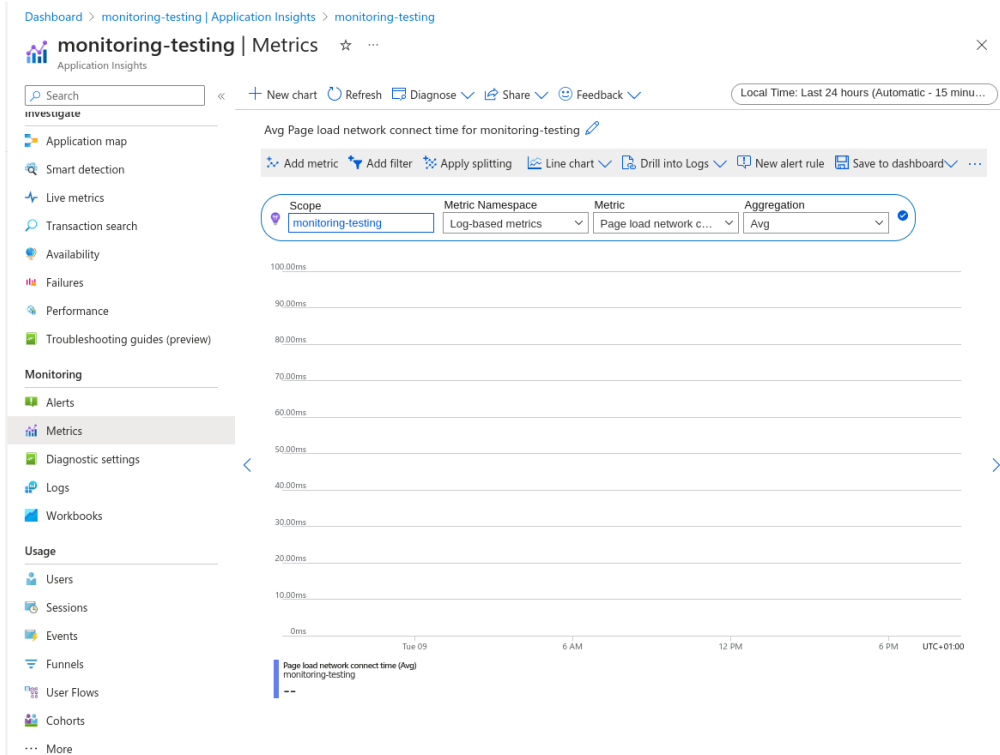


Figure 6.11: metrics dashboard

- **Log Analytics:** Log Analytics provides a centralized location for monitoring and analyzing the web application's logs. This tool can help identify issues and troubleshoot problems quickly.

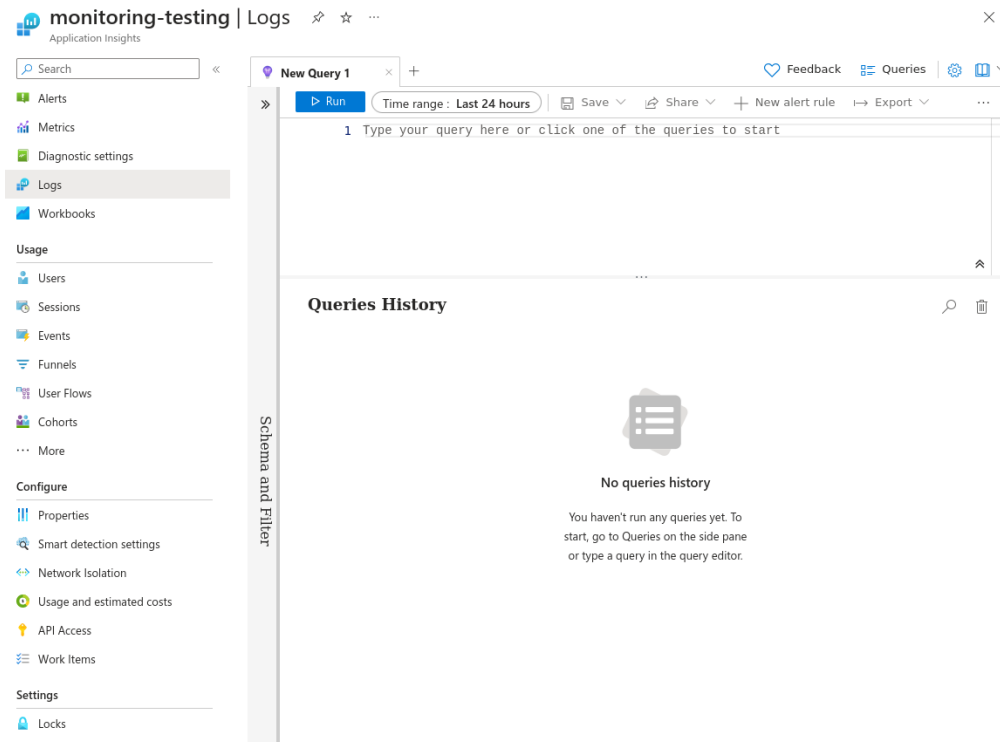


Figure 6.12: log analytics dashboard

## Conclusion

In conclusion, this chapter has outlined the deployment strategy chosen for this project, which is a blue-green deployment strategy. This strategy was selected due to its ability to provide swift deployments with minimal downtime and near-instant rollbacks. Additionally, a rollback strategy has been implemented to ensure a smooth recovery process in the event of any unforeseen issues during deployment. The rollback strategy includes a 30-day rollback window to allow for user feedback and admin decision-making. Finally, a monitoring strategy has been established to monitor the health, performance, and usage of the web application once deployed. This monitoring strategy utilizes various Azure monitoring tools, including Application Insights, Health Checks, Alerts, Metrics, and Log Analytics. By implementing these strategies, we can ensure a smooth and successful deployment process for our website.

# General Conclusion

The project successfully transitioned the deployment process from a manual system to a streamlined, automated approach. This was achieved by adopting a DevOps methodology, leveraging cloud infrastructure, and implementing modern tools. The initial groundwork has been laid by provisioning the cloud environment and establishing a CI/CD pipeline. Future efforts will focus on building and deploying the web application within this new framework.

Personally and professionally, this project provided me valuable learning experiences. It highlighted the importance of collaboration between development and operations teams (DevOps) for efficient software delivery. I gained practical experience with cloud computing tools like Terraform and GitLab, along with continuous integration and delivery (CI/CD) practices.

The world of DevOps is constantly evolving, and there's always room to push the boundaries. One intriguing idea would be to develop a developer-centric interface that integrates directly with the CI/CD pipeline. This interface could provide real-time feedback on code quality, suggest deployment strategies based on historical data, and even automate some aspects of the deployment process itself. Imagine a scenario where developers could visually drag and drop code modules to create deployment pipelines, or receive automated recommendations for infrastructure configurations based on their application's resource requirements. By building an intelligent and interactive interface, we could empower developers to take ownership of the deployment process while still maintaining governance and security best practices.

# Bibliography

- [1] . “Adactim.” [10-05-2024], Adactim. (-), [Online]. Available: <https://adactim.com/our-mission>.
- [2] . “Traditional vs agile project management: comparing and contrasting.” [10-04-2024], Atlassian. (-), [Online]. Available: <https://quixy.com/blog/traditional-vs-agile-project-management/>.
- [3] S. Pletcher. “Aws vs azure vs gcp: the big 3 cloud providers compared.” [16-02-2024], pluralsight. (8June 2023), [Online]. Available: <https://www.pluralsight.com/resources/blog/cloud/aws-vs-azure-vs-gcp-the-big-3-cloud-providers-compared>.
- [4] . “Introduction to devops.” [10-04-2024], geeksforgeeks. (6Feb, 2023), [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-devops/>.
- [5] . “Baseline highly available zone-redundant web application.” [29-02-2024], Microsoft. (-), [Online]. Available: <https://learn.microsoft.com/en-us/azure/architecture/web-apps/app-service/architectures/baseline-zone-redundant>.
- [6] R. Haddad. “What are the best git branching strategies.” [24-03-2024], ABTasty. (8Mars 2022), [Online]. Available: <https://www.abtasty.com/blog/git-branching-strategies/>.
- [7] . “Deployment strategies: 6 explained in depth.” [2-05-2024], Plutora. (May 3, 2022), [Online]. Available: <https://www.plutora.com/blog/deployment-strategies-6-explained-in-depth>.
- [8] . “Monitor azure app service.” [6-05-2024], Microsoft. (-), [Online]. Available: <https://learn.microsoft.com/en-us/azure/app-service/monitor-app-service>.

# Abstract

The document outlines a comprehensive plan to transition from a slow and error-prone deployment process to a significantly faster and more reliable one. This is achieved by embracing a DevOps approach that leverages cloud computing, automation, and modern tools and methodologies. The new approach will automate the entire deployment lifecycle, incorporate security best practices, and utilize Scrum for structured project management.

**Keywords :** DevOps, Cloud Computing, Automation, CI/CD, Scrum