IT Business School

**Logo**

Mini-project report Devops Tools

**Automated Deployment of a React Application Using DevOps Practices**

Prepared by

MED ALI CHAKHARI

Under the supervision of Mr. Firas Khlil

ACADEMIC YEAR: 2024-2025

**Acknowledgment**

I would like to express my sincere gratitude to all those who contributed to the successful realization of this project.  
First and foremost, I extend my heartfelt thanks to my academic supervisor for their continuous support, valuable insights, and encouragement throughout the entire process. Their guidance was instrumental in shaping the direction and quality of this work.

I am also deeply grateful to my instructors and professors who provided me with the technical foundation and critical thinking skills necessary to undertake this project.

Special appreciation goes to my classmates and peers for the productive discussions, technical advice, and moral support that kept me motivated.

Finally, I wish to thank my family for their unwavering support, patience, and motivation during the most demanding phases of this work. Their belief in me has been a constant source of strength.

**Abstract**

This report presents a complete, practical, and step-by-step guide to deploying a React.js web application using Jenkins CI/CD pipelines and hosting it on a cloud-based virtual machine using Amazon EC2.

The objective of this project is to automate the build, test, and deployment phases of a front-end application using modern DevOps tools. Beginning with infrastructure provisioning using EC2, the report covers essential configurations like Elastic IPs, SSH connections, security group rules, and software installations such as Node.js, Jenkins, and Nginx.

A key component of this deployment process is the creation and automation of a Jenkins Freestyle pipeline integrated with GitHub via webhooks to enable continuous deployment. We also explore a real-world use-case of ChatGPT to generate configuration commands for server setup.

Through this project, the importance of CI/CD, cloud computing, and infrastructure automation is highlighted, demonstrating their ability to enhance development velocity, scalability, and reliability in software delivery.

**Introduction**

In today’s fast-paced software development landscape, Continuous Integration and Continuous Deployment (CI/CD) have become integral to ensuring rapid and consistent delivery of high-quality applications. React, as a popular front-end JavaScript library, is widely used in developing dynamic and responsive user interfaces. However, deploying React applications to production requires a solid understanding of cloud services, automation tools, and secure hosting environments.

This project focuses on the deployment of a React application using a CI/CD pipeline powered by Jenkins, hosted on Amazon Web Services (AWS) EC2. By automating repetitive deployment tasks, we not only improve efficiency but also minimize human error.

The project begins with the creation and configuration of an EC2 instance, followed by SSH access and software installations. Jenkins is installed to act as the orchestration engine for automating the deployment. A Jenkins Freestyle pipeline is then configured to fetch source code from a GitHub repository, build the React app, and deploy it to the server.

The report also demonstrates the integration of webhooks for auto-triggering deployments on code updates, and the configuration of Nginx for serving the application over HTTP/HTTPS.

This deployment process reflects a production-grade DevOps pipeline and provides a practical experience in full-stack automation, cloud provisioning, and software engineering best practices.

# **Table of Contents**

1. **General Presentation and State of the Art**  
   1.1. Context of the Project  
   1.2. Project Objectives  
   1.3. Justification of the Project Choice  
   1.4. Overview of Similar Existing Approaches  
   1.5. Conclusion of Chapter 1
2. **Tools Used and DevOps Strategies Implemented**  
   2.1. Overview of the DevOps Methodology  
   2.2. Chosen Technological Stack  
   2.3. Cloud Infrastructure and EC2 Instance Setup  
   2.4. Jenkins as a CI/CD Orchestrator  
   2.5. Source Code Management and GitHub Integration  
   2.6. Deployment Automation and Webhooks  
   2.7. HTTP/HTTPS Configuration and Security Rules  
   2.8. Summary of Implementation Steps  
   2.9. Conclusion of Chapter 2
3. **Conclusion and Future Perspectives**  
   3.1. General Conclusion  
   3.2. Challenges Encountered  
   3.3. Key Takeaways  
   3.4. Future Perspectives  
   3.5. Final Thoughts

# **Chapter 1: General Presentation of the Project and State of the Art**

## **1.1 General Context**

In the current landscape of software engineering, **agility**, **automation**, and **continuous delivery** have become essential elements of successful application development and deployment. Traditional software lifecycles, which often separated development from operations, created barriers that slowed down releases, introduced configuration inconsistencies, and limited scalability.

**DevOps** is a cultural and technical movement that emerged to **bridge the gap** between development and operations teams. It emphasizes **automation**, **collaboration**, and **monitoring** across all stages of the software lifecycle: from code commit to deployment and maintenance.

Within this DevOps culture, the **CI/CD pipeline** has become a critical component. CI/CD stands for:

* **Continuous Integration (CI)**: Developers merge their changes frequently into the main branch. These changes are automatically tested and built.
* **Continuous Deployment (CD)**: Once tested, the software is automatically deployed to production or staging environments.

The result is a **faster, more reliable, and more scalable** development and release process.

In this project, we apply DevOps principles using **Jenkins** to automate the deployment of a **React.js application** to an **AWS EC2** instance, integrating tools such as **Nginx**, **GitHub**, and **Webhooks**.

## **1.2 Project Objectives**

The goal of this project is both practical and pedagogical: to demonstrate how to design and implement a **DevOps workflow** in a cloud context for a frontend application. The key objectives are:

### Technical Objectives

* **Set up an EC2 virtual machine** on AWS for hosting the application and CI/CD tools.
* **Install and configure Jenkins**, an open-source automation server, to act as the backbone of the deployment pipeline.
* **Integrate GitHub with Jenkins** using Webhooks, so that every code change in the GitHub repository automatically triggers a build.
* **Use Nginx** as a reverse proxy and static file server to serve the React application securely.
* Ensure the application is **deployed automatically** upon every change (push) to the main branch.

### Educational Objectives

* Learn to work with cloud infrastructure (AWS EC2) and understand basic cloud provisioning.
* Understand the role of each component in the DevOps toolchain.
* Gain experience in pipeline design and automation using Jenkins.
* Apply best practices in infrastructure management and application delivery.

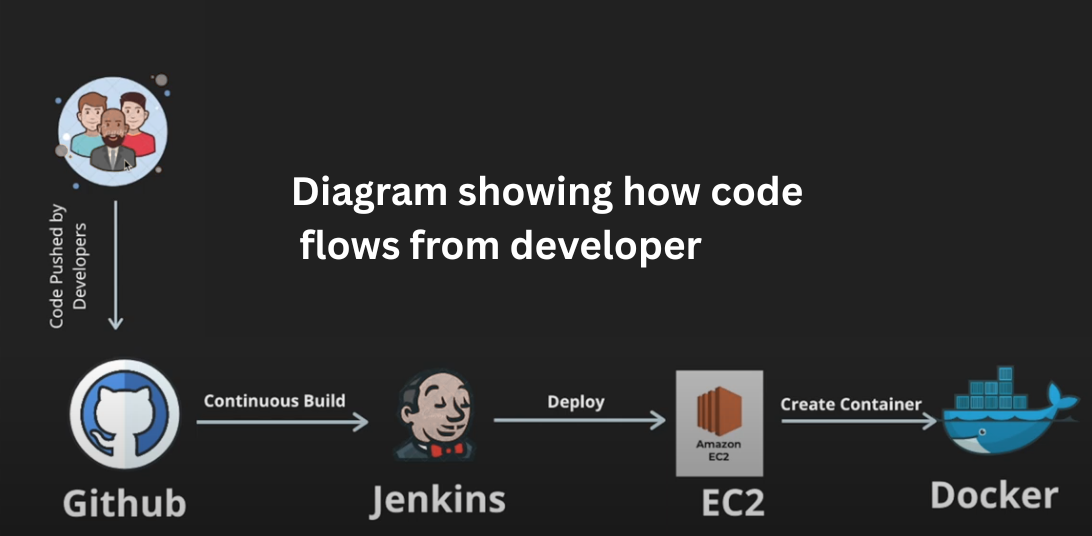


Figure 1 : Diagram showing how code flows from developer

## **1.3 Project Rationale and Technology Choices**

### Why Automate with DevOps?

DevOps offers several compelling benefits that justify the project choice:

* **Speed**: Deployments that previously took days or hours can be done in minutes.
* **Consistency**: Infrastructure as Code and pipelines ensure environments are reproducible.
* **Collaboration**: Developers and operations teams work from a shared pipeline and can monitor the same system.
* **Early error detection**: Automated testing and builds catch bugs early in the lifecycle.

### Why React?

* React is a popular JavaScript library for building user interfaces.
* It uses a **component-based architecture** that is modular and easy to test.
* A large community and ecosystem make it a natural choice for modern frontend development.
* The output of a React build is a static bundle, ideal for deployment via web servers like Nginx.

### Why AWS EC2?

Amazon Web Services (AWS) is the market leader in cloud computing. **EC2 (Elastic Compute Cloud)** offers:

* On-demand scalability and elasticity.
* Full control over the operating system and configurations.
* A wide choice of regions and availability zones for redundancy and performance.

### Why Jenkins?

Jenkins was chosen over other CI/CD tools for the following reasons:

| **Feature** | Une image contenant dessin, clipart, illustration, Dessin animé  Le contenu généré par l’IA peut être incorrect. | Une image contenant symbole, logo, Graphique, conception  Le contenu généré par l’IA peut être incorrect. | **Une image contenant orange, triangle, Graphique, conception  Le contenu généré par l’IA peut être incorrect.** |
| --- | --- | --- | --- |
| Hosting | Self-hosted | Cloud-based | Cloud/self-hosted |
| Customization | Extremely flexible | Moderately flexible | Flexible with some restrictions |
| Plugin Ecosystem | Rich (1500+ plugins) | Limited | Limited |
| Community | Very large | Growing | Growing |

## **1.4 State of the Art and Similar Approaches**

The shift towards DevOps and CI/CD practices is industry-wide. According to the **2023 DevOps Trends Report**, over 80% of software teams have adopted some form of CI/CD process.

### DevOps Tools Landscape

Some major tools used in CI/CD pipelines include:

| **Category** | **Tools** | **Role in Pipeline** |
| --- | --- | --- |
| Version Control | Git, GitHub, GitLab | Code repository and change history |
| CI/CD Servers | Jenkins, GitHub Actions, CircleCI | Orchestrate build, test, deploy |
| Cloud Platforms | AWS, Azure, GCP | Host infrastructure and apps |
| Containerization | Docker, Kubernetes | Consistent runtime and scalable deployments |
| Monitoring | Prometheus, Grafana | Observe and visualize metrics |

**Jenkins**, despite being one of the oldest tools, remains **highly relevant** due to its extensibility and strong community support.

### Case Studies and Related Work

Several companies have published case studies on how they transitioned to DevOps. For example:

* **Netflix** uses Spinnaker (built on top of Jenkins) to deploy thousands of microservices daily.
* **Airbnb** uses a combination of Jenkins, Docker, and Kubernetes to manage deployments.
* **Startups** often use Jenkins with AWS to balance control and cost-efficiency.

These examples confirm the **real-world relevance** of the approach chosen in this project.

## **1.5 Added Value of the Project**

This project provides a complete, real-world scenario for learning and applying:

* **DevOps concepts** in a practical context.
* Use of **cloud computing platforms (AWS)**.
* Creation of a **CI/CD pipeline** from scratch.
* Deployment and hosting of a **React application** using modern tools.

Moreover, it creates a **reusable framework** that can be adapted to backend services, mobile apps, or full-stack projects.

## **Conclusion**

This chapter has laid the groundwork for the rest of the report by explaining **why DevOps matters**, **what the project aims to achieve**, and **how it compares to other approaches** in the field. Through a mix of theoretical concepts and real-world considerations, we’ve built a strong case for our chosen technologies.

In the next chapter, we’ll move into the **practical implementation**, covering how we provision the EC2 instance, install Jenkins, configure Webhooks, and automate the deployment process.

# **Chapter 2: Tools Used and DevOps Strategies Implemented**

## **2.1 Introduction**

This chapter provides a detailed explanation of the tools used in the deployment process and the DevOps strategies that were implemented. Each tool was carefully selected for a specific purpose, and every step of the CI/CD pipeline is broken down to provide a clear understanding of the automation of a React application deployment on an AWS EC2 virtual machine.

## **2.2 Tools and Technologies Used**

Here is an overview of the key tools used, categorized by functionality:

### Hosting Environment: ****AWS EC2****

* **Role**: Host the React application and the Jenkins CI/CD tool.
* **Why AWS?**: It’s a reliable and widely adopted cloud provider offering flexible virtual machines.
* **Benefits**: Secure SSH access, customizable VM specs, and extensive documentation.

### CI/CD Server: ****Jenkins****

* **Role**: Automate the build, test, and deployment processes.
* **Why Jenkins?**: It's open-source, highly customizable via plugins, and well-established in the industry.
* **Benefits**: User-friendly web interface, flexible job configuration, and GitHub integration.

### Version Control: ****Git & GitHub****

* **Role**: Host and manage the source code.
* **Why GitHub?**: It supports collaboration and integrates easily with Jenkins using webhooks.
* **Benefits**: Version tracking, automated job triggers, and seamless pull/push operations.

### Web Server: ****Nginx****

* **Role**: Serve the static build files of the React application over HTTP.
* **Why Nginx?**: It’s lightweight, fast, and ideal for serving frontend apps.
* **Benefits**: Easy to configure, high performance, and low resource consumption.

## **2.3 Preparing the Cloud Environment (AWS EC2)**

The first step was creating a virtual machine on AWS EC2:

1. **Choosing the Image**: Ubuntu 22.04 LTS was selected for its stability and compatibility.
2. **Instance Configuration**:
   * Instance type: t2.micro (free tier and suitable for small-scale testing).
   * Security group: Configured to allow SSH (22), Jenkins (8080), and HTTP (80) traffic.
   * SSH Key: A key pair was generated to connect securely to the instance.
3. **Connecting to the VM**: Using a terminal with the private SSH key.

## **2.4 Installing and Configuring Jenkins**

After accessing the EC2 instance:

1. **Adding Jenkins Repository**: To ensure official and updated packages are used.
2. **Installing Java**: Jenkins requires Java as a dependency to run.
3. **Installing Jenkins**: Using the system's package manager.
4. **Accessing Jenkins in Browser**:
   * Available at http://<your-ec2-ip>:8080
   * Initial admin password retrieved for the first login.
5. **Installing Essential Plugins**:
   * Plugins for Git, Pipelines, GitHub integration, Webhooks, etc.
6. **Creating a Basic Job**: To test connectivity and functionality.

## **2.5 GitHub Integration and Webhook Configuration**

To automate deployments upon source code changes:

1. **Creating a GitHub Repository**: Containing the React application.
2. **Configuring Webhooks**:
   * Jenkins exposes a webhook endpoint.
   * GitHub is set to send a POST request to Jenkins on every push.
3. **Authentication**:
   * A GitHub token is used for Jenkins to access private repositories.
4. **Validation**:
   * A test push triggers Jenkins automatically, confirming integration.

Une image contenant texte, capture d’écran, Police, logo

Le contenu généré par l’IA peut être incorrect.

Figure 2 : Diagram showing how code flows from developer

## **2.6 Automated Deployment via Jenkins**

Once Jenkins receives a new commit, the pipeline performs:

1. **Cloning the Repository**: Jenkins pulls the latest code from GitHub.
2. **Installing Dependencies**: Based on the package.json of the React app.
3. **Building the Project**: The app is compiled and the static files are placed in a build/ directory.
4. **Deploying to Nginx**:
   * The build output is moved to /var/www/html, the default web root for Nginx.
5. **Restarting Nginx**: Ensures that new changes are served.

## **2.7 Overall CI/CD Strategy**

Here’s how the DevOps automation works:

* **Continuous Integration (CI)**:
  + On every push, Jenkins builds the code and catches any errors early.
* **Continuous Deployment (CD)**:
  + If the build passes, the changes are deployed automatically.
  + Users can see the updated application almost immediately.

Une image contenant texte, Police, logo, Graphique

Le contenu généré par l’IA peut être incorrect.

Figure 3 : Devops infinity loop

## **Conclusion**

This chapter explained in detail the tools and the DevOps strategy used in the project. From setting up a virtual machine to automating the deployment process with Jenkins and GitHub, each component was chosen to ensure efficiency, reliability, and scalability.

The end result is a system where every code update leads to an automated deployment, significantly speeding up development and reducing human error—meeting the standards of modern software engineering practices.

# **Chapter 3: Conclusion and Future Perspectives**

## **3.1 General Conclusion**

The deployment of a React application using DevOps methodologies demonstrates how automation, scalability, and modern tooling can enhance software development and delivery. Through this project, we successfully implemented a complete CI/CD pipeline using open-source tools such as **Jenkins**, **GitHub**, **AWS EC2**, and **Nginx**, which collectively enabled us to automate the build and deployment process effectively.

We began by provisioning and securing a cloud environment on AWS, followed by installing Jenkins to serve as our CI/CD orchestrator. We integrated Jenkins with a GitHub repository, configured webhooks for automatic triggers, and ensured that each code change led to a fully automated build and deployment. Throughout this process, the project adhered to best practices in terms of security, maintainability, and continuous delivery.

This setup not only facilitated the automation of repetitive deployment tasks but also ensured faster feedback loops, reduced manual errors, and improved the reliability of the application deployment lifecycle.

## **3.2 Challenges Encountered**

During the project, several challenges were encountered and overcome:

* **Permissions Issues**: Early Jenkins builds failed due to insufficient permissions when accessing deployment directories. This was resolved by adjusting file permissions and user privileges.
* **Security Configurations**: Setting up the correct inbound rules on AWS required careful attention to avoid security vulnerabilities while ensuring necessary access.
* **Tool Compatibility**: Ensuring that versions of Jenkins, Node.js, and Nginx were compatible with each other and the Ubuntu server environment required careful planning and testing.

These challenges provided valuable learning opportunities and reinforced the importance of testing, version control, and configuration management.

## **3.3 Key Takeaways**

* **Automation is Crucial**: Automating the build and deployment process improves productivity and ensures consistency.
* **Jenkins is Highly Adaptable**: Jenkins, though complex, proves to be a powerful tool for CI/CD workflows.
* **Cloud Deployment Brings Flexibility**: AWS EC2 provided the flexibility needed to configure the environment exactly as required.

These insights can be generalized and applied to more complex production-level systems in future professional settings.

## **3.4 Future Perspectives**

Several opportunities exist to build upon this project:

### Add Security Enhancements

Implement HTTPS with SSL/TLS certificates to secure communication. This includes installing Certbot with Let's Encrypt to serve the application over HTTPS.

### Dockerize the Entire Setup

Containerizing the Jenkins server, React application, and Nginx with Docker would improve portability, scalability, and consistency across environments.

### Implement Kubernetes (K8s)

A future evolution could involve migrating to a Kubernetes cluster to manage container orchestration and achieve high availability and self-healing capabilities.

### Integrate Testing Tools

Adding automated unit and integration tests to the Jenkins pipeline ensures higher software quality and better reliability.

### Monitoring and Logging

Introduce tools like **Prometheus**, **Grafana**, and **ELK Stack** to monitor performance and logs in real time, improving observability.

## **Final Thoughts**

This project demonstrates how DevOps practices can streamline the deployment of web applications. By implementing CI/CD pipelines, we reduce deployment times, increase reliability, and lay a foundation for scalable infrastructure. The experience gained here provides a strong base for tackling more advanced deployment architectures and integrating more sophisticated DevOps tools and workflows in the future.

**General Conclusion**

The deployment of a React application using a Jenkins-based CI/CD pipeline on an AWS EC2 instance illustrates the practical implementation of modern DevOps practices. This report has walked through the end-to-end process, from provisioning secure cloud infrastructure to automating deployment with webhooks and Nginx. Each chapter emphasized not only the technical steps involved, but also the rationale behind adopting such tools and methods.

By leveraging cloud services like Amazon EC2, we achieved flexibility, scalability, and availability of the application environment. Jenkins, serving as the backbone of continuous integration and delivery, allowed for robust pipeline configurations and automatic code deployments, drastically reducing manual effort and potential human error.

Additionally, integrating Node.js and Nginx ensured that the frontend application was served efficiently and reliably. The use of GitHub webhooks brought the process full-circle by enabling real-time, trigger-based deployments upon code pushes.

This hands-on deployment project demonstrates how cloud infrastructure, CI/CD tools, and automation can work in unison to achieve rapid, secure, and repeatable application delivery. It reflects the industry’s shift towards infrastructure as code, continuous delivery, and DevOps culture. The knowledge and experience gained through this deployment serve as a strong foundation for further exploration into containerization (Docker), orchestration (Kubernetes), and advanced monitoring tools (like Prometheus and Grafana), which represent the next steps in a full DevOps pipeline.