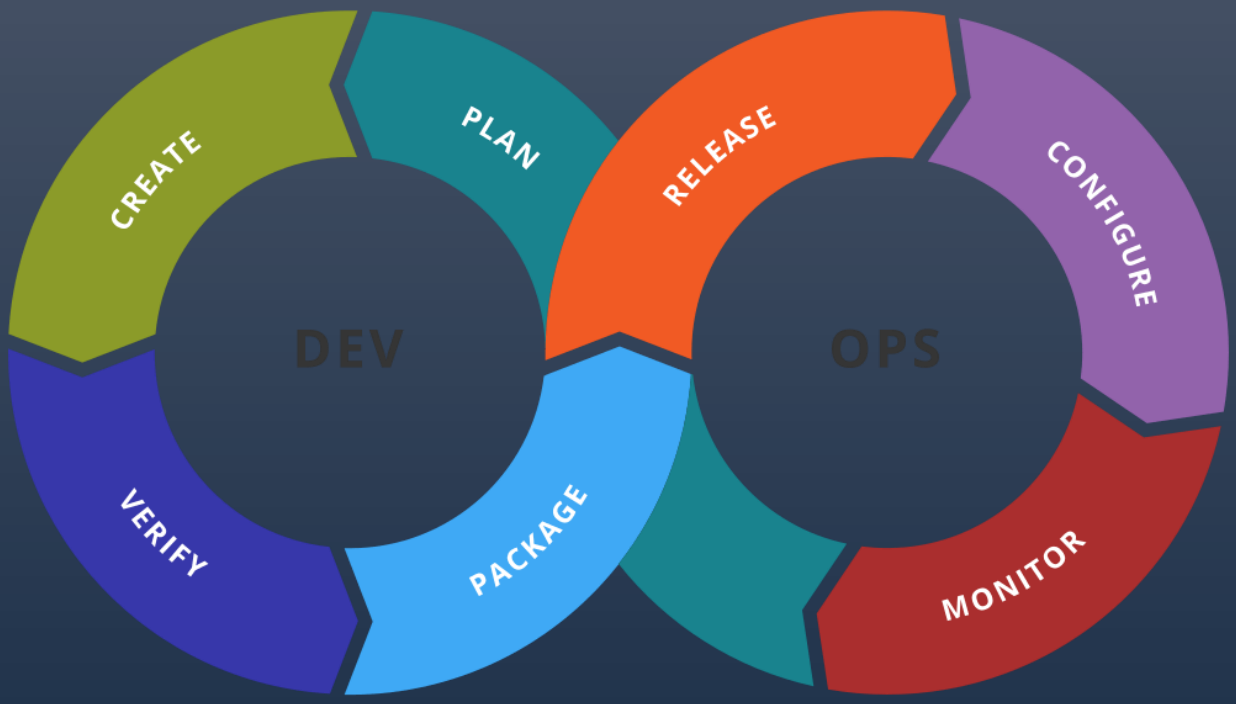


DevOps Projet - Report

IT INFRASTRUCTURE FOR AGILE AUTO PARTS COMPANY

ESTEBAN VINCENT, NELSA AISSA YAGO, CLORINDA NGUOKO, ARTHUR ROULET, LAMECHE INSAF, ROMAIN VILLENEUVE, YOHANN SABA, LÉO TORRADO

Table des matières

I – Introduction

Agile Auto Parts (AAP) has been a leading worldwide auto parts retailer for over two decades. However, since 2005, AAP has faced a decline in revenues, leading the board to investigate and identify areas for improvement. The IT department was identified as the primary source of inefficiencies, particularly in the Time to Market (TTM) for new releases.

In response, the board has shown interest in adopting DevOps practices, which have proven to significantly enhance TTM for competitors. This report outlines a comprehensive strategy to implement a DevOps CI/CD pipeline, using modern tools and methodologies to streamline AAP's software delivery process. The focus is on configuring Jenkins to manage the pipeline, incorporating build, test, and deployment stages, and ensuring seamless communication and deployment workflows.

To demonstrate the efficiency of the proposed CI/CD pipeline, we have selected a simple yet practical to-do list application for the proof of concept. This application will serve to illustrate how the pipeline can automate the build, testing, and deployment processes, while significantly reducing TTM and enhancing overall efficiency.

II - Strategy roadmap

II – 1. Blueprint of the existing system

II – 1. 1. Resources

Human Resources:

- Developers: 28
- Integrators: 2
- Designers: 5
- Architects: 2
- System Administrators: 3
- Database Administrators: 2
- Infrastructure Technicians: 5
- Release Managers: 3
- Security Expert: 1
- Chief Technology Officer: 1

Applications:

- Public Website: An online store accessible to customers, backed by a comprehensive backend system.
- Web Application for Stores: Used by retail stores to check parts availability and contact partners, connected to the same backend.
- Standalone Local Application: Employed within retail stores, interfacing with the backend system.
- Intranet Web Application: Dedicated to HR purposes, used internally by AAP employees.

II – 1. 2. Environments

AAP maintains multiple environments to support various stages of the development and deployment lifecycle:

- Development (Dev): Where initial development and integration of new features occur.
- Integration: For combining and testing integrated components.
- User Acceptance Testing (UAT): Where end-users validate the functionality before production.
- Security: Ensuring the application meets all security requirements and standards.
- Performance: Testing the application's performance under different conditions.
- Preproduction: A staging environment that mirrors the production setup for final checks.

- Production: The live environment where the application is available to end-users.

II – 1. 3. Infrastructure

AAP's current infrastructure relies entirely on physical servers, with no virtualization in place:

Physical Servers: 21 servers, including backup servers, running various services:

- HTTP Servers: Apache, Nginx
- Web Servers: Tomcat
- Databases: MySQL, PostgreSQL
- Applications: PHP, Java EE, Java Swing
- Application Server: WildFly
- Operating Systems: Debian, Ubuntu, CentOS

II – 1. 4. Geographical Locations

AAP's developers and servers are distributed across three primary locations:

- Bordeaux
- Nantes
- Brussels

II – 1. 5. Delivery Process

The current delivery process is manual and time-consuming:

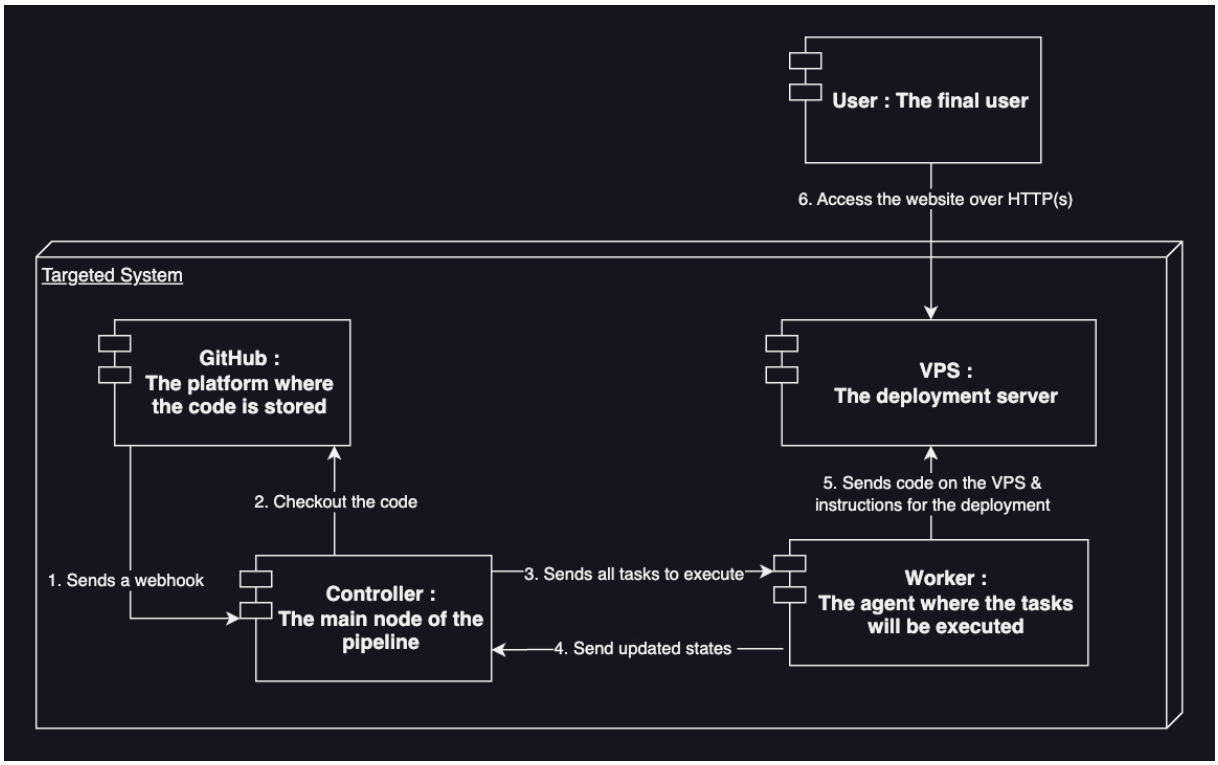
- Developers send their code as zip files via WeTransfer.
- No version control system is in place, leading to manual code merging by release managers.
- Deployment is slow, taking at least seven weeks, with limited automated testing.
- Environments are outdated and inconsistently maintained.

II – 2. Blueprint of our target solution

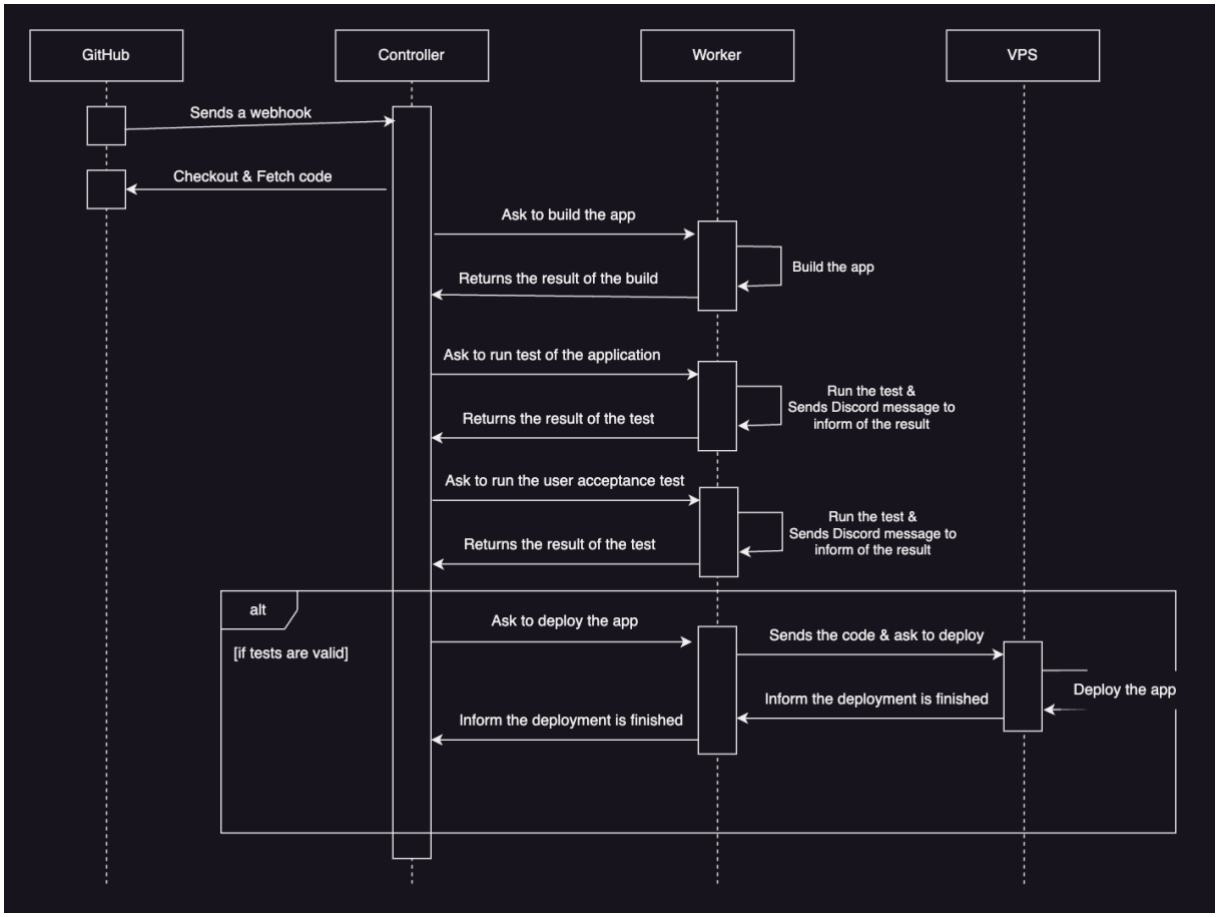
To have a better understanding of the project and the blueprint of the targeted solution, we've made 2 diagrams :

- one deployment diagram to understand how we can made our deployment
- and one sequence diagram, to understand the different steps of the deployment

II – 2. 1. Deployment diagram



II – 2. 2. Sequence diagram



III - Main concerns

III – 1. Pipeline / CI / CD

A DevOps pipeline is a combination of automation, tools, and practices across the software development life cycle to facilitate the development and deployment of software into the hands of end users. A pipeline is a series of automated steps that software goes through from code commits to production release.

The primary benefit of a DevOps pipeline is speed to deployment. As of today, each deployment takes 7 weeks because our 3 release managers must go through the work done by each of our 28 developers to do a manual merge. By using a pipeline, we will be able to quickly build, test and deploy using automated processing and tooling.

- Automated Builds: Every time a developer commits changes to the version control system (which you will need to adopt), the application is automatically built. This ensures that the main branch is always in a deployable state.
- Automated Testing: Integrate automated testing in the CI process to catch bugs early. Given AAP's current lack of testing, start with basic unit tests and progressively integrate more comprehensive tests (integration, system).

Initially, we recommend AAP adopt a continuous delivery scheme of work that packages code and prepares it for release once the automated tests have been passed. Continuous delivery will ease the deployment to production process by enabling it to be done by simply clicking on a button.

Implement a staging environment that mirrors production to ensure that deployments can be tested before they go live, thus reducing deployment risks.

Once AAP is comfortable with continuous delivery, we will introduce continuous deployment where every change that passes all stages of the production pipeline is released to customers automatically. This will reduce the time and effort required for manual reviews and interventions.

III – 2. Jenkins

Jenkins is an open-source automation server that is commonly used for continuous integration (CI) and continuous delivery (CD). It helps automate the parts of software development related to building, testing, and deploying, facilitating the integration and delivery of code changes more efficiently and effectively.

Jenkins can be integrated into various stages of the CI/CD pipeline. Here’s a typical pipeline with Jenkins:

- **Source Code Management**: Jenkins integrates with source code repositories like GitHub, GitLab, Bitbucket, etc.
- **Build**: Jenkins can allow us to compile the code, run unit tests, and generate build artifacts.
- **Testing**: Jenkins can allow us to run automated tests (unit, integration, and functional tests).

- **Deployment**: Jenkins can allow us to deploy applications to different environments (development, staging, production).
- **Monitoring**: Jenkins can allow us to trigger monitoring tools to ensure the application is running smoothly post-deployment.

III – 2. 1. Best Practices for Using Jenkins

- **Use Declarative Pipelines**: Prefer declarative pipelines for better readability and maintainability.
- **Keep Jobs Simple**: Break complex jobs into simpler steps or jobs.
- **Use Shared Libraries**: Share common code across different pipelines using shared libraries.
- **Security**: Secure your Jenkins server with proper authentication and authorization mechanisms.
- **Backup and Restore**: Regularly back up your Jenkins configuration and job details.
- **Monitor Jenkins Performance**: Use monitoring tools to keep track of Jenkins' performance.

III – 2. 2. Alternatives to Jenkins

While Jenkins is popular, there are several other CI/CD tools available:

- **GitLab CI/CD**: Integrated with GitLab, providing seamless CI/CD processes.
- **Travis CI**: A hosted CI/CD service for GitHub repositories.
- **CircleCI**: Offers fast builds and can be used with GitHub, GitLab, and Bitbucket.
- **TeamCity**: A CI/CD server from JetBrains, known for its robust features and integrations.
- **Azure DevOps**: Microsoft's CI/CD platform that integrates with Azure services.

III – 3. Virtualization / Containerization

III – 3. 1. Virtualization vs Containerization

Virtualization involves creating virtual machines (VMs) that run on a hypervisor, allowing multiple OS instances on a single physical machine. Each VM includes a full copy of an operating system, along with the application and its dependencies, which leads to larger sizes and slower start times.

Containerization uses containers to encapsulate an application and its dependencies. Containers share the host system's OS kernel but run in isolated user spaces. This results in lightweight, faster, and more efficient application deployment.

III – 3. 2. Benefits of Using Docker Over VMs

- **Lightweight**: Containers share the host OS kernel, making them smaller and faster to start compared to VMs.
- **Efficiency**: Containers use system resources more efficiently, allowing higher density of applications on a single host.
- **Consistency**: Containers ensure the application runs the same, regardless of where they are deployed.
- **Scalability**: Containers can be easily scaled horizontally to handle increased load.
- **Portability**: Docker containers can run on any system that supports Docker, facilitating easy movement between development, testing, and production environments.

III – 4. Communication: Slack

Effective communication is a cornerstone of successful DevOps implementation. To facilitate seamless communication among Agile Auto Parts' teams, we propose the integration of Slack as the primary communication tool. Slack offers a range of features that can streamline collaboration, enhance information sharing, and improve overall workflow efficiency.

III – 4. 1. Benefits of using Slack

1. **Real-Time Communication:**
    - Instant messaging allows team members to communicate in real-time, reducing delays in decision-making and problem-solving.
    - Channels can be created for specific projects, teams, or topics, ensuring that relevant discussions are organized and easily accessible.

2. **Integration with CI/CD Tools:**
    - Slack can be integrated with Jenkins and other CI/CD tools to provide real-time updates on build status, deployment progress, and test results.
    - Notifications and alerts can be set up to inform the team about critical events, such as build failures or successful deployments.

3. **Collaboration Features:**
    - File sharing, screen sharing, and video conferencing capabilities facilitate comprehensive collaboration among team members, regardless of their geographical location.
    - Bots and automation tools can be used to streamline routine tasks, such as scheduling meetings or reminders.

4. **Enhanced Transparency:**
    - All communication is archived and searchable, ensuring that important information is preserved and can be referenced later.

- Status updates and progress reports can be shared in relevant channels, keeping everyone informed about project developments.

III – 4. 2. Implementation Strategy

5. **Set Up Slack Workspace:**
   - Create a Slack workspace dedicated to Agile Auto Parts and invite all relevant team members.
   - Organize channels based on teams

6. **Integrate Slack with Jenkins:**
   - Use the Jenkins Slack plugin to configure notifications for build and deployment events.
   - Set up automated alerts for critical issues, such as build failures or security vulnerabilities.

7. **Develop Communication Protocols:**
   - Establish guidelines for channel usage, including naming conventions and purposes.
   - Define protocols for escalation and resolution of issues to ensure timely responses and accountability.

8. **Training and Onboarding:**
   - Provide training sessions for team members to familiarize them with Slack's features and best practices.
   - Create documentation and resources to support ongoing use and troubleshooting.

III – 5. Testing & KPIs

Testing is a crucial component of the DevOps pipeline, ensuring that code changes meet quality standards before they are deployed to production. Implementing a robust testing framework within the CI/CD pipeline can significantly enhance the reliability and performance of applications.

III – 5. 1. Testing Strategy

1. **Automated Testing:**
   - **Unit Testing:** Validate individual components or functions to ensure they work as intended.
   - **Integration Testing:** Verify that different modules or services interact correctly.

- **Functional Testing:** Check the end-to-end functionality of the application against business requirements.
- **Regression Testing:** Ensure that new code changes do not adversely affect existing functionality.

2. **Performance Testing:**
   - Conduct load testing to assess how the application performs under high user traffic.
   - Execute stress testing to determine the application's breaking point and its ability to recover gracefully.

3. **Security Testing:**
   - Implement automated security scans to identify vulnerabilities in the codebase.
   - Conduct penetration testing to simulate attacks and uncover security weaknesses.

4. **User Acceptance Testing (UAT):**
   - Facilitate UAT in a dedicated environment where end-users can validate that the application meets their expectations and requirements.

III – 5. 2. Key Performance Indicators (KPIs)

To measure the effectiveness of the CI/CD pipeline and the overall DevOps implementation, we will track the following KPIs:

5. **Deployment Frequency:**
   - Measure how often new code is deployed to production. Higher frequency indicates a more efficient and responsive development process.

6. **Change Lead Time:**
   - Track the time taken from code commit to deployment in production. Shorter lead times reflect a faster TTM.

7. **Mean Time to Recovery (MTTR):**
   - Calculate the average time taken to restore service after a failure. A lower MTTR indicates a more resilient system.

8. **Change Failure Rate:**
   - Monitor the percentage of deployments that result in failures in production. Lower failure rates signify higher code quality and stability.

9. **Test Coverage:**
   - Assess the percentage of code covered by automated tests. Higher test coverage ensures better validation and reduces the risk of defects.

10. **Defect Density:**
    - Measure the number of defects per unit of code (e.g., per 1,000 lines of code). Lower defect density indicates higher code quality.

III – 5. 3. Implementation Plan

11. **Integrate Testing Tools:**
    - Use tools like JUnit for unit testing, Selenium for functional testing, JMeter for performance testing, and OWASP ZAP for security testing.
    - Configure Jenkins to run these tests automatically as part of the CI/CD pipeline.

12. **Monitor KPIs:**
    - Implement monitoring and reporting tools to track KPIs continuously.
    - Use dashboards and visualizations to provide real-time insights into the pipeline's performance.

13. **Continuous Improvement:**
    - Regularly review KPI data to identify areas for improvement.
    - Implement feedback loops to ensure that insights from testing and KPIs drive ongoing enhancements to the pipeline and processes.

III – 6. Organization / Change Management

III – 6. 1. Team Structure Changes

- **Integrated Teams:** AAP currently has segregated roles like developers, system administrators, and database administrators. Introduce cross-functional teams where these roles collaborate from the beginning of a project, promoting faster decision-making and reducing bottlenecks in the deployment process.

- **Role Expansion:** Encourage existing roles to expand their skill sets. For instance, developers should gain basic operations knowledge and vice versa, promoting a 'you build it, you run it' mentality.

III – 6. 2. Mentality Shift :

- **From Project to Product:** Shift the organizational focus from project-based to product-based, which entails ongoing improvement and support rather than considering 'projects' as temporary and finite.

- **Embrace Failure as a Learning Tool:** Instill a culture where failures are seen as opportunities to learn and improve, not just setbacks. This is crucial for fostering an innovative and resilient team.

III – 6. 3. Revamping the IT delivery process

- **Automation First:** Given AAP's outdated manual processes and the long release cycles that can go up to 7 weeks, we will introduce automation in testing, builds, and deployments to reduce errors and speed up the time to market of the company's products.

- **Regular Releases:** We intend to move from infrequent and large updates to regular, smaller releases. This reduces risk and allows quicker responses to market changes.

III – 6. 4. Technical Architecture Adaptation

- **Cloud Adoption:** Considering the lack of virtualization and reliance on physical servers, transitioning to cloud environments would offer scalability and better disaster recovery options. This would support CI/CD processes effectively.

- **Decompose Monolithic Systems:** If applicable, start breaking down large, monolithic applications into more manageable microservices. This will facilitate easier updates and better fault isolation.

III – 6. 5. Enhancing the tool chain

- **Version Control System:** Introduce a robust version control system (like Git) to replace the current practice where developers send zip files via WeTransfer. This will support collaboration and track changes effectively.
- **CI/CD Tools:** We will encourage the use of Jenkins for automation and train the team on how to use them to ensure these tools are utilized effectively.

III – 6. 6. Change Management Strategy

- **Stakeholder Engagement:** Engage with stakeholders early by showcasing the direct benefits of DevOps through a pilot project. Highlight improvements in deployment times, reduction in bugs, and better response times to customer demands.

- **Training and Upskilling:** Organize workshops and training sessions for all technical staff to get accustomed to new tools and practices. This could be done in phases, starting with the basics of Git and Jenkins, and gradually moving to more advanced topics like containerization and orchestration.

- **Feedback and Adaptation:** Establish feedback loops with all teams to gather insights on the new processes and adapt strategies as required. This might involve regular retrospective meetings and suggestion boxes to collect anonymous feedback.

IV - Proof of Concept

IV. 1 – Prerequisites

For testing our proof of concept, you'll need a VPS. After that, you'll need to setup and secure your VPS. Then, you'll have to follow this steps :

1. Install Git
2. Clone the project in a first time : $ git clone https://github.com/Serquand/DevOps-M1
3. Navigate to the app folder : $ cd DevOps-M1/app
4. Install NodeJS
5. Install globally PM2 : $ sudo npm install -G pm2
6. Install the dependencies : $ npm install --omit=dev
7. Copy and paste the .env.template in .env and fill with your MongoDB database URL
8. Starts a first time the application : $ npm start

In addition of that, you'll need a BOT Discord. For that, follow this steps :

1. Go on the https://discord.com/developers/applications
2. Click on New Application and give a name to the application
3. Go on the OAuth2 section, click on "BOT", and then on "Send Messages".
4. Copy and paste the generated URL in a new windows, and invite the BOT in your server.
5. Go on the BOT section
6. Reset a BOT token and copy it.

IV. 2 – Installation

For the installation part, follow these steps:

1. Clone the Repository from GitHub to obtain the application's source code.
2. Navigate to the Project Directory
3. Build and Run the Application with Docker Compose by executing this command:

$ docker compose up --build -d

IV. 3 – Setup

For the setup part, configure Jenkins to automate the pipeline:

1. Go on localhost:8080, or on the server you've installed your project.
2. Install Plugins: Once Jenkins is up and running, install the necessary plugins (e.g., NodeJs)
3. Setup the NodeJS tools and call it Node22.

4. Configure Nodes: Set up a worker node to distribute the build load

S	Nom ↓	Architecture	Différence entre les horloges	Espace disque disponible	Espace de swap disponible	Free Temp Space	Temps de réponse
	contrôleur	Linux (aarch64)	Synchronisé	41.67 GiB	1024.00 MiB	41.67 GiB	0ms
	Worker	Linux (aarch64)	Synchronisé	41.67 GiB	1024.00 MiB	41.67 GiB	32ms
	Données obtenues	37 mn	37 mn	37 mn	37 mn	37 mn	37 mn

5. Create a Jenkins Pipeline
   - Create a new pipeline job in Jenkins : Click on **New Item**, enter a name for the job (e.g., To-Do List App Pipeline), and select **Pipeline**.
   - Define the pipeline as “Pipeline script from SCM”: Enter the Git repository URL where the pipeline script is located (https://github.com/Serquand/DevOps-M1)

6. You’ll need to set up your env variables. For that, go on Dashboard > Administer Jenkins > System > Environnement variables and setup the following variables :
   a. DISCORD_BOT_TOKEN : The copied token
   b. DISCORD_CHANNEL_ID : The channel ID where you want the messages arrive
   c. HOST_IP : The IP of your VPS
   d. MONGODB_URL : The URL of your MongoDB database

7. You can now start a build.

IV. 4 – Use

To access and use the application, navigate to the URL “http://<server-ip>:3000” in your web browser, replacing “<server-ip>” with the IP address or domain name of the server where the application is deployed. This will open the application we’ve created for this proof of concept.

V – Conclusion

Agile Auto Parts (AAP) faces significant challenges in improving its Time to Market (TTM) due to inefficiencies in its IT department. Adopting DevOps practices can transform AAP's software development and delivery processes, enhancing agility, efficiency, and reliability.

This report detailed the current system's limitations and proposed a target solution with modern DevOps tools and methodologies. The proof of concept, demonstrated with a to-do list application, showed how the CI/CD pipeline can automate build, testing, and deployment, reducing TTM and improving overall efficiency.

By implementing this DevOps strategy, AAP can overcome its current challenges, improve operational efficiency, and regain its competitive edge in the market.