DEVOPS | Project Report

Group 10

Summary :

The team :

LACHAUD Antoine,

LE LAY Logan,

MARGUET Vincent,

MARQUES Romain,

NASS Michaël,

RINCHEVAL Antoine,

THAVARASA Jathoosh

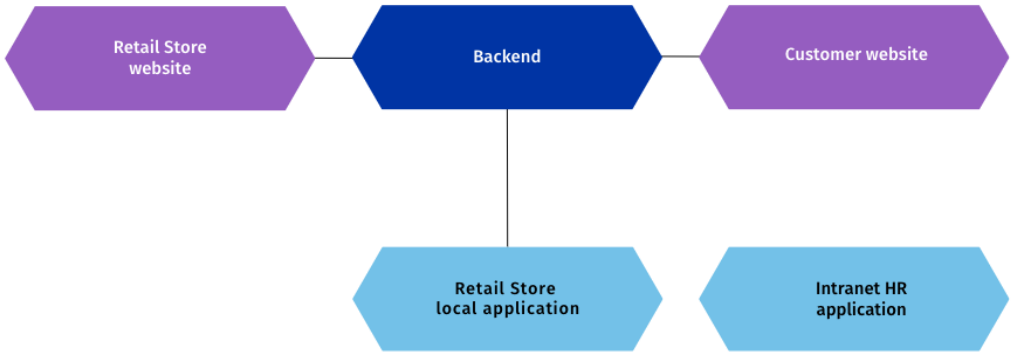
Strategy Roadmap

Existing System

AAP is a big international company, needing our help to improve their DevOps cycle by investigating their IT department and services branch. In fact, this branch has existed since the creation of the company, as 2 developers were doing all the work.

The IT department is composed of 52 members with more than half of them being developers. There is only one security expert, which raises questions about the security level of their IT infrastructure. The environments are well decomposed with lots of different steps to deploy the updates correctly.

Regarding their applications, AAP maintains a public website, a backend, a web application dedicated to their physical stores, a local application used in these stores which is connected to the same backend as the customer website, and one intranet web application for HR purposes. We can summarise this situation in the following diagram :



We can see a first issue here : connecting all the online platforms to the same backend isn't a good idea as all can be down if the backend is down.

Regarding the infrastructure, nothing is virtualized as the company didn't invest time on it when they built the IT system. Using containers with Docker (or similar software) could greatly improve the infrastructure. Other issues are occuring here, making the infrastructure hardly maintainable :

- AAP uses two different frameworks to run their HTTP servers
- Same with databases
- Some applications are coded in PHP, while others use Java EE or Java Swing, which makes everything hardly maintainable without problems.
- Servers runs on three different linux distributions which all have their own specifications. CentOS for example has a different package manager (yum) compared to Debian and Ubuntu (apt). This is a noticeable problem on its own as applications could have different versions available depending on the package manager.

While having their developers located close to each other around France is fine, it isn't a good idea for an international company to have all its servers located in the same country. If an american client tries to access the customer website, it will take a lot more time to load the resources as its device has to travel all the ocean to reach the servers.

Finally, the delivery process is awfully bad :

- It isn't secured at all : all the release code is sent via WeTransfer, meaning anyone who has a WeTransfer link can access ALL the code used on the different company's platform
- Having a code repository is one of the basics of maintenance nowadays, and AAP needs to implement one (which will also fix the WeTransfer issue)
- Having a code repository will also fix the manual merge done by the release managers since GitHub and others can do automatic merge with the appropriate pipeline tools
- Speaking of pipeline tools, it's also clearly needed here since every deployment is done manually. GitLab CI or GitHub Actions should be good. These deployment tools will also be able to run all the needed tests in the cloud.

The company also clearly indicated that all environments aren't maintained and are outdated. The strategy explained in the next section will help to correct all of the issues mentioned.

Target Solution

Blueprint

The target solution should use a git repository to store all the code used by the project. To keep the project organized, it is recommended to split the components from different applications and services into separate repositories, allowing for easier version control, collaboration, and maintenance. Setting up a code repository will secure the release process and facilitate release management through the use of automatic merge and continuous integration and deployment tools, such as Jenkins or GitHub Actions, which will allow automated testing and simplify deployments, as well as to avoid errors in the early stages of the development.

Also, to improve the AAP company's DevOps cycle, we will need a holistic approach that will require a significant change in the structure and skills of their IT department and services. First, infrastructure security is a priority. Building a dedicated team of security specialists is essential to protect systems and data against the growing threats of cybercrime. Next, we will need to seek virtualization and containerization skills to modernize the infrastructure and use technologies like Docker to easily deploy, manage, and scale applications, making the whole infrastructure more flexible and efficient and using cloud services.

Regarding developing skills, it will be crucial to put in place a standardization strategy to simplify development and maintenance while there are different frameworks and programming languages. In addition, for an international company such as AAP, it will be essential to diversify the location of the servers. The cloud infrastructure allows us to easily improve performance and the user experience on an international scale, while making the deployment easier than with centralized servers thanks to Continuous Delivery.

To improve the DevOps cycle within the AAP enterprise, it will be essential to form a dedicated team that will be responsible for implementing these changes. This team should be composed of professionals skilled in the fields of security, virtualization, containerization, software architecture and deployment. The team will work on rewriting or adapting existing applications to use consistent frameworks and programming languages. This will simplify maintenance and improve collaboration between developers.

Architecture

Our solution includes an overhaul of the architecture, by getting rid of all the servers (of course, with a transition period to ensure everything works fine). All the environments are shifted to become cloud based on the cloud provider chosen by the company. In our Proof-Of-Concept, we have adopted the use of AWS, by using AWS Lambda Functions and Amazon S3, with a possible upgrade to Amazon DynamoDB for the backend and GitHub Pages for the static distribution of the frontend. Each environment gets the right set of resources and network in order to operate in a secured and isolated environment. This allows them to remove a layer of administration for the servers, and rely on the cloud

services instead which require way less effort and have an unprecedented uptime rate. For more complex applications they deploy, they can use Virtual Machines on the cloud, like Amazon Elastic Compute EC2, which can be deployed using container images. For the local testing, before the code is being pushed on the git repository, they can use Docker using the same configuration profiles and images as in AWS.

Tools used

The only tool we used is GitHub action. GitHub Actions is a continuous integration and continuous delivery (CI/CD) service provided by GitHub, the popular source code management platform. It allows developers to automate their software workflows in response to specific events on their GitHub repository, such as creating a new branch, creating a pull request, uploading a new version of code, etc. GitHub Actions offers many benefits for development teams:

- Workflow automation: GitHub Actions allows defining custom workflows using YAML files to automate various tasks such as code compilation, unit testing, deployment, team notification, etc.
- Ease of use: With native integration with GitHub, setting up workflows is relatively simple and seamlessly integrates into the existing development environment.
- Integration with the GitHub ecosystem: GitHub Actions can be configured to react to specific events in a GitHub repository, such as pushes, pull requests, tags, etc.
- Customization of triggers: You can define specific triggers for workflows, responding to specific actions performed by developers or other external services.
- Distributed Execution: Actions are executed in containers, enabling distributed execution for increased scalability and speed.
- Library of predefined actions: GitHub Actions has an extensive library of predefined actions that can be reused for common tasks, saving developers time and benefiting from community expertise.
- Visualisation of workflows: GitHub Actions provides a graphical interface for visualising workflows and monitoring the status of executions in real time.
- Integration with notifications: You can set up notifications to be notified of workflow results, test failures, etc., making it easier for team members to collaborate.

To summarize, GitHub Actions is a powerful and flexible tool for automating development, testing, and deployment processes, helping to accelerate the delivery of high-quality software while reducing human error and improving collaboration within the development team.

Main concerns

Pipeline/CI/CD

In computer science, the term pipeline refers to the continuous delivery pipeline : a set of automated processes that facilitate the development, testing and deployment of software. It interacts during the entire lifecycle of a project, from source code management to production deployment. The pipeline serves as the backbone of the DevOps strategy by enabling teams to automate and optimise the software delivery process. It ensures that code changes are efficiently built, tested, and deployed, leading to faster and more reliable software releases.

Implementing a pipeline is essential for achieving DevOps goals as it provides the necessary automation and structure such as continuous integration, continuous delivery and continuous deployment. Continuous Integration (CI) is a crucial aspect of the pipeline, where developers regularly integrate their code changes into a shared repository, allowing for early detection of integration issues, ensuring a smoother and more efficient development cycle.

We recommend pushing towards Continuous Integration as a foundational practice within DevOps strategy and for all domains.

While Continuous Integration focuses on frequent code integration and testing, Continuous Deployment takes it a step further by automatically deploying code changes to production after passing all automated tests. Continuous Deployment enables AAP to rapidly and consistently release new features, bug fixes, and enhancements to customers. It further accelerates the Time to Market and ensures a faster feedback loop between development and end-users.

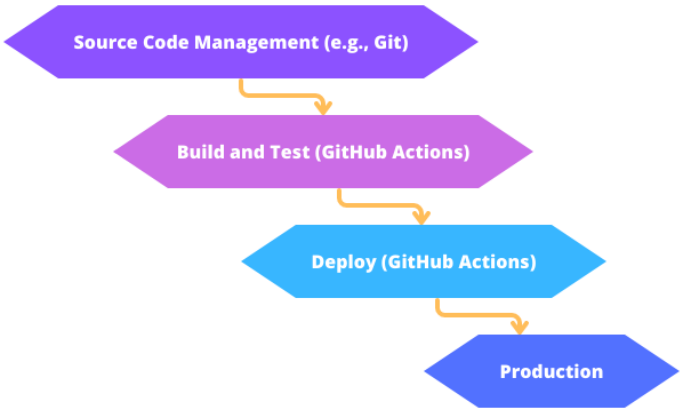
For a company like AAP, it is essential to have multiple pipelines segmented according to their needs, more specifically according to their environments. Each environment needs to have its own pipeline depending on what the objective is. The way to segment these pipelines and environments is by using branches in the git repository storing the code. We will be storing the git repository on GitHub. The main branch is considered as the production branch and is therefore protected for direct editing, as the commits must transit through a development or pre-production environment where the Continuous Integration pipelines for testing will be executed, before being manually merged after a review for the Continuous Delivery pipelines to kick in and deploy the solution.

GitHub-Action

Jenkins is a powerful CI/CD software that can be installed locally or on a proprietary server. It mostly acts as an orchestrator, and executes CI/CD actions like testings and deployments in containers that are created on the fly using Vagrant or Docker. While Jenkins is well suited for local testing and deployment, we chose to use GitHub Actions, which is a powerful and widely used workflow automation platform provided by GitHub. It shares the

same goal as Jenkins, however it uses GitHub's proprietary workflow syntax and features, to be perfectly integrated with the GitHub repository. As GitHub Actions is a service, its orchestrator features are built-in which makes it really easy to use on any environment and removes the hassle of changing the OS and architectures to start docker containers. It allows us to automate various tasks and processes within our software development lifecycle. Furthermore, if AAP decides to make open source software, GitHub Actions is the best way to ensure that external contributors get their code properly tested according to the company pipelines. With GitHub Actions, we can define custom workflows as code, enabling the automation of build, test, and deployment processes.

In the DevOps pipeline, GitHub Actions (which can be replaced by the Jenkins orchestrator after the SCM, and Docker or Vagrant for the Build, Test and Deploy, where each of these steps would be executed in a dedicated container launched by Jenkins), can be integrated at multiple stages to streamline and automate the software delivery process. Here is a simplified diagram showcasing the role of GitHub Actions within the pipeline:



In this diagram, GitHub Actions is responsible for executing the build and test stages of the pipeline. It pulls the source code from the version control system (e.g., Git), compiles and builds the code, runs automated tests, and generates artefacts.

For best practices when using GitHub Actions, here are a few recommendations:

- Utilise GitHub's secret management feature to securely store sensitive information such as API keys and credentials.
- Implement comprehensive tests and validation steps to ensure the quality and integrity of the code.
- Leverage the GitHub Actions marketplace to explore and utilise pre-built actions shared by the community, saving time and effort.
- Divide workflows into separate jobs for running the shell script and the Cypress tests, offering better organization and parallel execution.

While GitHub Actions is a popular and feature-rich automation tool, like we said, it can be replaced by popular alternatives like Jenkins and GitLab CI which offer similar functionalities and have their own strengths. We can also talk about CircleCI, Travis CI, TeamCity, etc.

GitHub Actions, being tightly integrated with GitHub, has gained significant traction due to its seamless workflow setup within the GitHub ecosystem. Its vast library of pre-built actions and continuous updates have made it a compelling choice for our project.

Jenkins, on the other hand, has been one of the oldest and most widely used CI/CD tools in the industry. Its extensibility through plugins and long-standing reputation have made it a reliable choice for organizations with complex CI/CD needs and those preferring on-premises setups. It is also the best suited option for local CI/CD pipelines.

GitLab CI, as part of the GitLab platform, offers a comprehensive solution for version control, CI/CD, and project management, making it a popular choice for teams seeking a single integrated platform for their software development lifecycle.

As of 2020, the market for Continuous Integration/Continuous Deployment (CI/CD) tools was dynamic and competitive, with several popular options available to support the automation of software development processes. Among these options, for GitHub-based projects, GitHub Actions emerges as the recommended solution due to its ease of use, tight integration with GitHub repositories, and extensive community support.

Virtualization / Containerization (Docker)

Virtual machines are emulations of a physical computer running an OS on a virtualisation layer. Indeed a VM includes a complete installation and replication of an OS and its resources. They provide full isolation between the host machine and the virtual environment. This allows different OS to run concurrently but it also costs many resources and slow startup times.

On the other hand, containerization provides a lightweight alternative. Indeed, using containers does not need a complete OS installation to provide an isolated environment. They are faster to install and to run.

Containerization, particularly with tools like Docker, Podman, or ContainerD, offers several benefits over traditional virtualization with VMs.

As mentioned earlier, containers are lightweight and have faster startup times compared to VMs. Containers share the host OS kernel, which means they don't require a full OS installation like VMs. This allows containers to start and stop almost instantly, making them ideal for rapidly scaling applications and microservices.

Since containers share the host OS resources, they are more resource-efficient than VMs which need a separate OS for each instance, leading to higher memory and storage consumption.

Containers encapsulate the application and its dependencies, making them highly portable across different environments, ensuring consistency between each of them.

They are a perfect fit for DevOps practices. By packaging the application and its dependencies together, containers streamline the deployment process and facilitate continuous integration and continuous delivery (CI/CD) pipelines. This reduces the time and effort required to move from development to production.

Containers support versioning, enabling you to maintain different versions of your application simultaneously. This makes it easier to roll back to a previous version in case of issues, ensuring more robust and reliable software deployments.

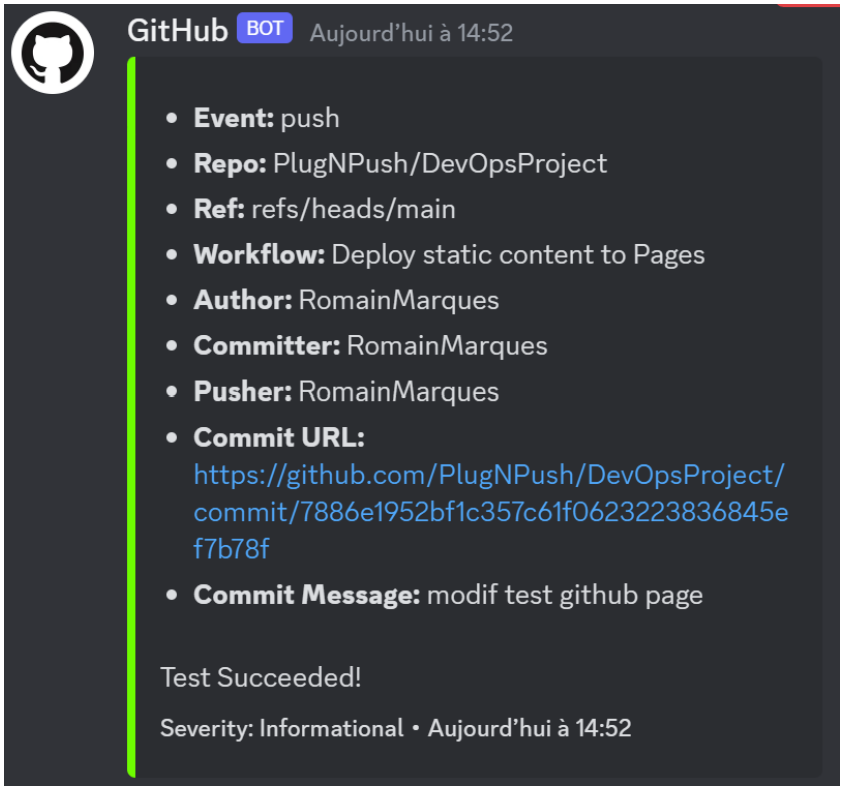
We can take some scenarios where Containerization Excels like in microservices architectures, cloud-native applications, development and testing environments and some others.

Communication: Slack (or Discord)

> "Made for people. Built for productivity.

Connect the right people, find anything that you need and automate the rest. That's work in Slack, your productivity platform." - Slack

It is a very popular and powerful tool for team communication and collaboration. By adopting Slack, AAP can enhance real-time communication, facilitate information sharing, and provide a centralised space for discussions and updates. It provides an organised space for everyone and everything that you need for work by using channels and chats where you can chit-chat, send audio and video clips, or join a huddle to talk things through live. Slack gives the possibility to connect other work apps and with the Workflow Builder, you can automate away routine tasks : now there is no reason to not add Slack to the APP workflow.



In our case we decided to use another very powerful tool : Discord! It can be used like Slack, offering a place where we can create channels, chit-chat, send a lot of documents of many types, make a vocal call with the camera and the screen if we want, share code snippets... There are so many functionalities, and we can also automate some actions like you can see on the screen above : here we receive automatically a message from GitHub each time someone makes a commit. Discord is hands down one of the best online communication and collaboration tools in the world!

Testing & KPIs

To enhance the efficiency and reliability of our testing processes and ensure high-quality software delivery, a few crucial steps should be taken. First, we should make continuous testing throughout the development lifecycle to identify potential issues early on and ensure that new code changes do not adversely impact the application's functionality. This can be measured using the KPI of "Mean Time to Detect" (MTTD), which tracks the average time taken to identify issues from the moment they are introduced.

Second, leverage automation thanks to GitHub Actions which execute tests automatically. This reduces manual effort, provides consistent results, and enables quicker feedback on code modifications. The effectiveness of test automation can be assessed through the KPI of "Test Automation Coverage," which measures the percentage of test cases that are automated in the testing process.

Furthermore, we can strive for comprehensive test coverage, including unit tests to examine individual code units in isolation, integration tests to validate interactions between different components, and end-to-end tests to verify the application's functionality as a whole. This ensures that we make it less prone to bugs and vulnerabilities. The success of achieving comprehensive test coverage can be monitored through the KPI of "Code Coverage," which measures the percentage of code that is exercised by the test suite.

While it's ideal to automate as many tests as possible, it may not be feasible or necessary to automate every single test. While unit tests or integration tests can be easily automated, tests that include subjective evaluation or complex user interactions can be difficult to automate. In such cases, we can focus on the KPI of "Defect Detection Efficiency," which measures the effectiveness of the testing process in identifying defects during manual testing.

There are many practices that are important in tests. First of all it is important to test continuously, the software should be tested through the whole development phase to verify the impact of the new code. It is also important to Automate the test as much as possible with a framework such as github action or Jenkins. We should pay attention to the unit tests and deployment environment configuration, to ensure that we cover all the sensitive parts of the code to reduce the risk of deploying a broken code or code that contains errors in production. The more detailed and self-explanatory the unit tests are, the quickest it will be to solve the issues early in development and avoid carrying bugs from feature to feature in the development environment and later even in production. As the unit tests are integrated in our CI/CD pipelines, they are a great indicator of code stability. Some companies even include linters in their unit tests, to ensure the code has the same level of consistency and quality across all the employees such that it is as clean as possible when it is released in the main branch, in addition to be functional.

Organization / Change management

DevOps brings about significant changes to various aspects of an organization. Our proposal includes an analysis of how DevOps will affect AAP in terms of team structure, mentality, IT delivery process, technical architecture, and toolchain. We will provide insights into the potential risks associated with human resistance to change and propose mitigation strategies to ensure a smooth transition. One potential risk is the resistance from system administrators and infrastructure technicians to the job conversion. Shifting their roles to focus on building and managing CI/CD pipelines might lead to concerns about job security and the need to acquire new skills. To mitigate this, a comprehensive training program can be implemented to upskill and reskill these employees for their new roles. Communication and transparency throughout the process are crucial to gaining their support and ensuring a successful transition. Additionally, we will recommend a comprehensive change management plan that includes educating and training teams, fostering a culture of collaboration and continuous improvement, and promoting cross-functional teams to enable faster and more efficient adoption of DevOps practices.

To address the risk of disruptions during the migration process, it is essential that the move to DevOps is made in parallel with the current deployment. This means that during the transition phase, teams will have to manage both the existing deployment and the new CI/CD pipelines. This increased workload might cause temporary challenges for the teams. However, careful planning, adequate resources, and effective communication will help minimize disturbances for end-users and ensure a smooth transition. The main challenge will be to succeed in a job conversion for the system administrators and infrastructure technicians, and dispatch them to increase the team of integrators and release managers, to create, manage and monitor the CI/CD pipelines. This presents a risk as their core job is more or less discarded from the company, but moving to the cloud represents a great advantage for the rest of the team as they will more easily be able to focus on the code without making mistakes, improving the set to production times and rates, and therefore improving their Time-to-Market. The jobs of the system administrators and infrastructure technicians will also get easier and way less stressful, as they won't have to monitor the servers 24/7, since the cloud services have a high reliability rate and are self-managed by the provider.

In order to successfully handle the change, it is important that the move to DevOps is being made in parallel with the current deployment, even if it will temporarily increase the work loads across all teams. This phase might lead to potential risks such as increased pressure on the teams and a learning curve for using the new CI/CD pipelines. However, by providing adequate support, training, and clear documentation, these challenges can be successfully managed. It is important that such a migration does not cause disturbances for the end users. While modifications are being made to the code and shared via WeTransfer to the servers, the integrator teams can focus on building the first CI pipeline for testing using a sample of code. Once the first CI is up, the CTO should create git repositories for the projects. The Lead Developer should transfer the current code on the git remote, and inform all the developers to use the git repository to fetch and update the code, while still sending it in parallel via email to not disturb the old workflow. Once this step is achieved, the architects and release managers should establish the CD pipeline with the cloud provider of their choice, by using the git repository directly as a reference. Once the CI/CD pipelines work and the developers are familiar with the git repositories, there are two ways of finishing the transition: either AAP slowly shuts down one server at a time, ensuring that its equivalent has been deployed on the cloud using the Continuous Deployment, which leaves time for the system administrators and infrastructure technicians to verify that no specific configuration files were lost in the process by gracefully terminating the servers or rollback the changes in the event of unforeseen issues, or AAP can also opt for a brutal transition after an extended period of testing, by setting all their servers in maintenance mode (business closed for a couple of hours) and switching to the cloud deployed servers overnight. Once all the servers are terminated, the old infrastructure can be destroyed and the old employees can start their new full time position on the cloud platforms.

Throughout the change process, communication and collaboration with members of the company will be essential. It will be necessary to explain the advantages and the objectives

of each stage of the change to promote support and minimize resistance. Regularly monitoring key performance indicators (KPIs) related to the adoption of DevOps, such as time-to-market, deployment frequency, and mean time to recovery, will help identify areas that need improvement and provide valuable insights for optimizing the DevOps implementation further. Finally, once the changes are implemented, it will be essential to regularly monitor and evaluate the performance of the new architecture and development processes. This will allow for possible improvements and ensure that business objectives are met. By addressing these main concerns, we aim to provide Agile Auto Parts with a thorough understanding of how DevOps can address their challenges and help them achieve their goal of reducing Time to Market and restoring growth.

POC

For best viewing, our POC is available at :https://github.com/PlugNPush/DevOpsProject.

For the CI/CD pipeline, we decided to go fully in the GitHub ecosystem. We will be using GitHub Actions for continuous integration, and a combination of GitHub Actions and GitHub Pages for continuous delivery, and Cyclic as a bonus.

First of all, the branch “main” is protected against any direct commit by anyone.

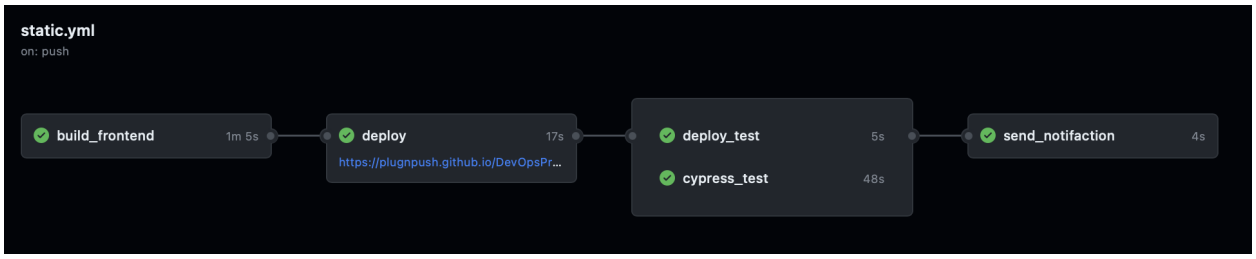
Then, we have a first workflow that allows or not the possibility to validate a pull request on the branch “main” from any other branch or forked repository. The requirements to validate it are based on two criteria :

- First, the backend must go through unit tests
- Second, the frontend must be able to be build
- (In any cases, met or not, a discord notification is sent with a webhook URL to a specific channel informing the team about the final status of these two tasks)

When these two criteria are met, the pull request can be made, or else it will require further changes. In order to further enhance the protection of the main branch, a manual review by a project team member is required to merge a branch into the main branch.

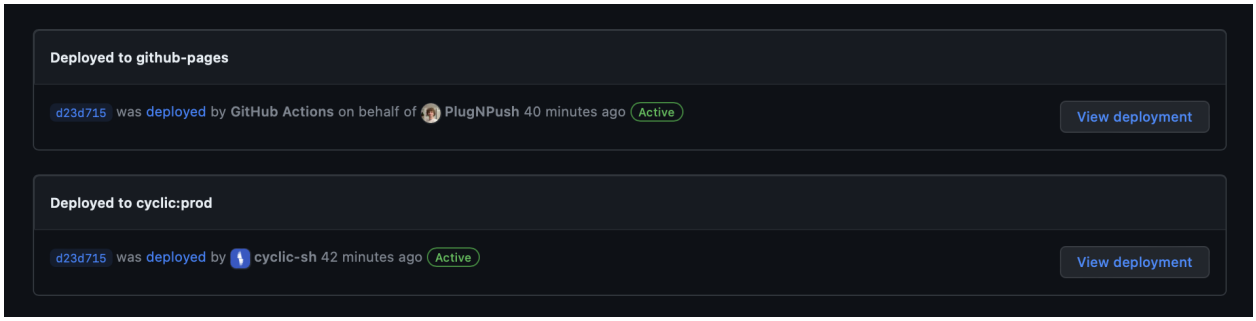
Our second workflow is launched only when changes are committed to the “main” branch through a validated pull request and it will do multiple things like :

- Build our frontend, generating a static website, and deploy it through GitHub Pages
- Then it runs a bash script to check that the website answers 200 when requested.
- In parallel, we are doing a test through Cypress to check everything is alright on the deployed website, in terms of the deployed content (the website is loading and the buttons are on the page)
- And finally, we send a discord notification with a webhook URL in a specific channel, informing us about the final status of the deployment (completed, cancelled by a team member or failed)



As we wanted to go even further, we added a third workflow, but this time external from GitHub Actions. We used Cyclic to handle the deployment of a functioning backend as well as database as part of the continuous delivery, and integrated it to the frontend. Cyclic is a

free service that allows anyone to host an app, with paid tiers for more powerful usages. Cyclic is based on AWS, using Amazon Lambda to run our backend in a serverless architecture. Thanks to the AWS integration, we were also able to manage our database using AWS services, automatically thanks to environment variables and private networks provided by Cyclic, so we linked them both.



While we should use Amazon DynamoDB for the database as it is more efficient, the project we started to build on was not designed with a proper database management system, and was relying on raw file storage, that is similar to JSON but not compliant to the format. Therefore, as we did not have time to rewrite the project, we chose to use Amazon S3 to store our database files, and it allows us to have a working backend.

Obviously, as it is a proof-of-concept, we only implemented two git branches that represent two environments: main for the production environment and develop for the development environment. The actual project would have as many branches as they have environments, and as many workflows to run tests, deploy in different environments or apply quick security patches.