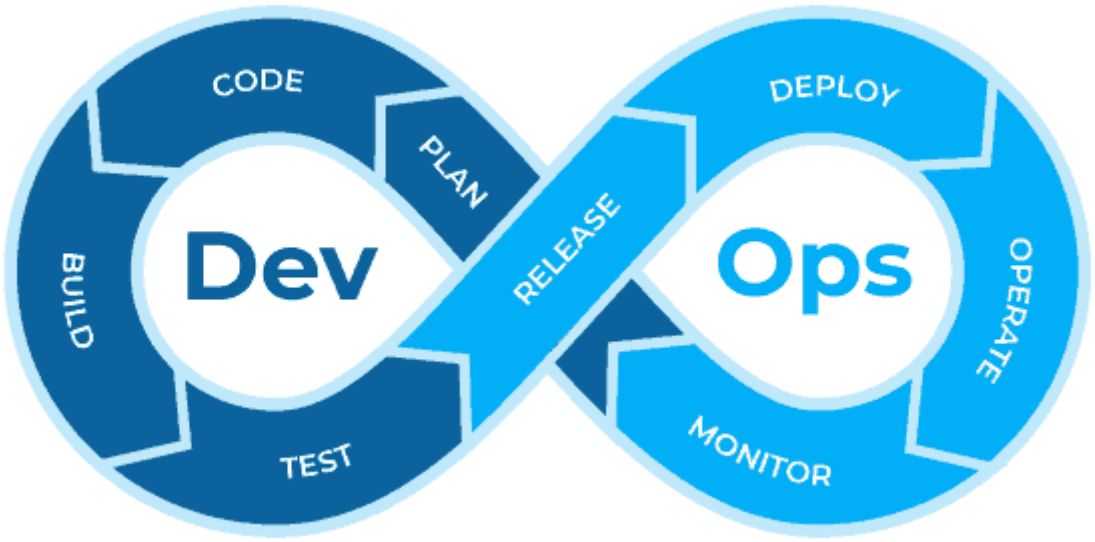
PARIS PANTHÉON-ASSAS UNIVERSITÉ

DEVOPS
Final Project



LE CORVEC Malo – CARRETERO Enrique – KEUNEBROEK Baptiste - Othmane OUBOUSELHAM - Pape Mouhamadou Mamoune SOCK – Bryan KESSEL

Strategy roadmap

Blueprint of the existing system

1.1 Organization and Locations:
Agile Auto Parts (AAP) is a major worldwide auto parts retailer operating in 36 countries with 452 stores. The company has nearly 6800 employees across various roles and departments. AAP's development and IT operations are distributed across three different sites: Bordeaux, Nantes, and Brussels.

1.2 IT Department Profiles:
The IT department consists of the following roles:

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

1.3 Applications:
AAP utilizes several applications to support its operations:
Public Web site for customers (online store) + backend
Web application for stores (parts availability, partner communication)
Standalone local application used in retail stores
Intranet Web application for HR purposes
All these applications are connected to a common backend.

1.4 Environments:
The existing environments used in the development and deployment process are as follows:
Dev
Integration
User Acceptance Tests
Security
Performances
Preproduction
Production

1.5 Infrastructure:

AAP currently operates on physical servers, with no virtualization in place. The infrastructure comprises 21 physical servers, including backup servers. The servers host various components and technologies such as HTTP (Apache and nginx), Web (Tomcat), databases (MySQL and PostgreSQL), applications (PHP, Java EE, Java Swing), WildFly application server, and operating systems like Debian, Ubuntu, and CentOS.

Blueprint of the target solution

We chose to use Github Actions for:

On the main branch: Build, test, and deploy the application on Amazon EC2 whenever there is a change.

A dev branch that does the same but for the development environment application.

An application in two parts, api and app, which is a server and a client part. And for each part, it is deployed on Amazon EC2.

Slack and Discord notifications for errors or success of the build and updates.

Version Control:

Utilize Git and create a repository on GitHub to manage the application code.

Branching Strategy:

a) Main Branch: This branch will be used for production-ready code. Set up Github Actions to trigger the build, test, and deployment processes on Amazon EC2 whenever changes are detected on this branch. Upon successful deployment, the application will be available for use in the production environment.

b) Dev Branch: Create a separate branch, "dev," to facilitate development and testing. Implement Github Actions to build, test, and deploy the application on an environment specific to development. This branch will allow developers to collaborate and test new features before merging them into the main branch.

Continuous Integration and Deployment (CI/CD):

Configure Github Actions workflows to automate the build, test, and deployment processes for both the main and dev branches. These workflows should include the following steps:

a) Build: Compile the application source code and generate the executable or deployable artifact.

b) Test: Execute automated tests, including unit tests, integration tests, and any other relevant test suites to ensure the application's quality.

c) Deployment:

Main Branch: Deploy the application on Amazon EC2 instances in the production environment upon successful completion of the build and test stages.

Dev Branch: Deploy the application on a separate environment dedicated to development and testing purposes.

Infrastructure Provisioning:

a) Amazon EC2: Set up and configure EC2 instances to host the application in both the production and development environments. Ensure that the instances are properly provisioned with the required resources, such as CPU, memory, and storage, to support the application's requirements.

Error and Update Notifications:

Integrate Slack and Discord to receive notifications for errors and updates. Configure Github Actions to send alerts to designated channels on Slack and Discord whenever an error occurs during the build,

test, or deployment process. Additionally, configure notifications for successful deployments and updates to keep the team informed.

By implementing this target solution, the development and deployment processes will be streamlined using Github Actions. The application will be automatically built, tested, and deployed on the appropriate environments based on the branch. Slack and Discord notifications will ensure effective communication regarding errors and updates, enabling prompt resolution and keeping the team up to date with the application's status.

Main concerns

PIPELINE/CI/CD

What does the word pipeline refer to? How does it fit in our DevOps strategy? Do we need a pipeline in order to be DevOps compliant? What about Continuous Integration? What are your recommendations? Should we push til Continuous Deployment? What are the differences?

In the context of DevOps, the term "pipeline" refers to a set of automated processes that allow for the continuous integration, testing, and deployment of software applications. It is a key component of the DevOps strategy and facilitates the efficient and reliable delivery of software products.

A pipeline in DevOps typically consists of several stages, including:
-Continuous Integration (CI): This stage involves automatically integrating code changes from multiple developers into a shared repository. It ensures that the changes do not introduce conflicts or errors by running automated tests and code quality checks.
-Continuous Delivery (CD): In this stage, the software is automatically built, tested, and prepared for deployment to a staging or production environment. CD focuses on ensuring that the software is always in a deployable state.
-Continuous Deployment: This stage takes the automation a step further by automatically deploying the software to the production environment after passing all the necessary tests and checks. It eliminates manual intervention in the deployment process, enabling faster and more frequent releases.

Having a pipeline is not mandatory to be DevOps compliant, but it is highly recommended. A well-designed and automated pipeline can significantly improve the efficiency, quality, and speed of software development and delivery processes. It helps in reducing manual errors, enables faster feedback loops, and enhances collaboration among development, testing, and operations teams.

Continuous Integration (CI) is an essential practice in DevOps. It involves frequently integrating code changes into a shared repository and running automated tests to detect issues early. CI ensures that the codebase remains in a consistent state and helps identify and resolve conflicts or defects quickly.

Regarding recommendations, we would suggest implementing a comprehensive DevOps pipeline that includes Continuous Integration (CI) and Continuous Delivery (CD) practices. This pipeline should automate the build, test, and deployment processes to enable faster and more reliable software delivery.

Continuous Deployment can be considered as the next step beyond Continuous Delivery. It involves automatically deploying the software to production without any manual intervention. Continuous Deployment is suitable for organizations that have robust testing and monitoring practices in place, ensuring that the deployed software is of high quality and meets the required standards.

However, before moving to Continuous Deployment, it is crucial to establish a solid foundation with Continuous Integration and Continuous Delivery. This involves implementing automated tests, ensuring code quality, and having effective monitoring and rollback mechanisms. Gradually, as confidence in the process and system grows, the organization can consider adopting Continuous Deployment. Ultimately, the choice between Continuous Delivery and Continuous Deployment depends on the organization's specific needs, level of automation, and risk tolerance.

JENKINS (OR GITHUB-ACTION / GITLAB-CI)

We are seriously considering adopting Jenkins. What it is? Where does it fit in the pipeline (diagram please)? Any best practices you'd recommend? Is Jenkins the only tool in its category? Give us an overview of the market in 2020.

Jenkins is an open-source automation server that facilitates the continuous integration and delivery of software applications. It is one of the most popular tools in the DevOps ecosystem and is widely used for building, testing, and deploying software.

In the DevOps pipeline, Jenkins typically fits in the Continuous Integration (CI) stage. Here's a diagram of how Jenkins fits into the pipeline:

Source Code Repository (e.g., GitHub) -> Jenkins -> Build -> Test -> Package -> Deploy

Jenkins acts as the orchestrator in the pipeline. It connects to the source code repository (such as GitHub) and triggers the build process whenever changes are detected. It fetches the latest code, compiles/builds the software, runs tests, and packages the application artifacts.

Here are some best practices for using Jenkins effectively:

Automate the Build Process: Configure Jenkins to automatically build the software whenever changes are pushed to the repository. This ensures that the build is triggered consistently and reduces manual effort.

Use Jenkins Plugins: Jenkins has a vast collection of plugins that extend its functionality. Explore and leverage plugins to integrate with other tools, enable additional features, and customize your build and deployment processes.

Parallelize Builds: Jenkins supports parallel execution of build jobs across multiple agents or nodes. Utilize this feature to distribute the workload and speed up the build process, especially for larger codebases.

Implement Build Triggers and Notifications: Set up triggers to initiate builds based on events such as code commits or scheduled intervals. Configure email notifications or integrate with messaging platforms to receive alerts and updates about build status and failures.

Version Control and Configuration Management: Ensure that your Jenkins configuration, including jobs and plugins, is version controlled. This helps in tracking changes, maintaining consistency, and facilitating easy recovery or replication of the Jenkins environment.

While Jenkins is a widely adopted and feature-rich tool, it is not the only one in its category. Other popular tools for CI/CD include GitLab CI/CD, GitHub Actions, and TeamCity. These tools offer similar functionality with varying degrees of integration and ease of use.

As for an overview of the market in 2020, it's important to note that the DevOps tool landscape is continuously evolving. In 2020, Jenkins remained one of the dominant players due to its extensive plugin ecosystem, community support, and compatibility with various technologies and platforms.

However, there has been increased adoption of cloud-native CI/CD solutions like GitLab CI/CD and GitHub Actions, which offer tighter integration with their respective source code management platforms. These solutions provide a seamless experience for developers by combining code hosting, CI/CD pipelines, and collaboration features within a single platform. Overall, the DevOps tool market in

2020 showcased a diverse range of options, allowing organizations to choose tools that best suit their specific requirements, preferences, and existing technology stack.

VIRTUALIZATION / CONTAINERIZATION (DOCKER)

Virtualization vs Containerization: we want to understand the differences before making any decision. What are the benefits and various scenarios of using Docker (Podman/ContainerD) instead of our good old VMs?

Virtualization and containerization are two distinct approaches to running applications and managing software environments. Here are the differences between the two:

Virtualization:

Virtualization involves running multiple virtual machines (VMs) on a physical server, with each VM having its own operating system (OS) instance.

Each VM is isolated and provides a full OS environment, including its own kernel, libraries, and resources.

VMs can be resource-intensive due to the duplication of OS instances, requiring more memory and storage.

Migration and scalability of VMs can be complex and time-consuming.

Containerization (Docker):

Containerization, on the other hand, uses containers to run applications. Containers are lightweight and isolated execution environments.

Containers share the host OS kernel and only include the necessary application dependencies, libraries, and binaries, making them more lightweight and efficient.

Containers provide consistent behavior across different environments, ensuring that applications run consistently regardless of the underlying infrastructure.

Containers offer faster startup times, efficient resource utilization, and easy scalability.

Docker is one of the most popular containerization platforms, providing tools and APIs for building, packaging, and deploying containers.

Benefits of using Docker (or other containerization technologies like Podman or ContainerD) instead of VMs include:

Resource Efficiency: Containers are lightweight and share the host OS kernel, allowing for more efficient resource utilization. Multiple containers can run on a single host, maximizing server capacity.

Portability: Containers encapsulate applications and their dependencies, making them highly portable across different environments (development, testing, production). They provide consistent behavior and reduce the "it works on my machine" problem.

Scalability: Containers can be easily replicated and scaled horizontally, enabling quick and efficient application scaling to meet increased demand.

Fast Deployment: Containers can be started and stopped rapidly, resulting in faster application deployment and updates. This supports continuous integration and delivery practices.

Isolation and Security: Containers provide isolation between applications, minimizing the impact of potential vulnerabilities. Each container operates in its own isolated environment, enhancing security.

DevOps Integration: Containers are well-suited for DevOps practices, facilitating automation, version control, and collaboration. They can be easily integrated into CI/CD pipelines, enabling faster and more reliable software delivery.

It's important to note that virtualization and containerization are not mutually exclusive, and they can be used together in certain scenarios. For example, virtualization can be used at the infrastructure level to provide the underlying host environment for containerization platforms like Docker.

Ultimately, the choice between virtualization and containerization depends on factors such as application requirements, resource utilization, scalability needs, and the level of isolation desired. Containerization, with tools like Docker, offers significant benefits in terms of efficiency, portability, scalability, and faster application deployment, making it a popular choice for many modern software development and deployment scenarios.

COMMUNICATION: SLACK

We might choose Slack to improve the general communication between the teams. How could we do that? Please include a Communication tool demo in your POC.

Introducing Slack as a communication tool can greatly improve collaboration and communication among teams. Here's a step-by-step demonstration of how you can utilize Slack to enhance communication within your organization:

1. Create a Workspace: Start by creating a Slack workspace for your organization. You can sign up for a new workspace on the Slack website or use an existing workspace if one already exists.

2. Set Up Channels: Channels in Slack are dedicated spaces for conversations related to specific topics, projects, or teams. Create channels based on your organization's structure, projects, or any other relevant criteria. For example, you could have channels for development, marketing, sales, and specific project-related channels.

3. Invite Team Members: Invite team members to join the Slack workspace. Users can be added by sending them email invitations or sharing an invitation link. Make sure to assign users to relevant channels to ensure they receive relevant updates and can participate in discussions.

4. Sending Messages: Users can send messages in channels or privately to individuals or groups. Encourage team members to use appropriate channels for relevant discussions. For example, project-related discussions should happen in project-specific channels, while company-wide announcements can be shared in a general channel.

5. Collaboration Features: Slack provides numerous features to enhance collaboration. Some key features include:

- File Sharing: Users can share files (documents, images, etc.) directly in Slack channels or through private messages.

- Mentioning and Notifications: Users can mention specific individuals or teams using the '@' symbol to notify them of relevant messages.

- Threads: Threads allow for focused discussions within specific messages, reducing clutter in the main conversation.

- Integrations: Slack integrates with various third-party tools, allowing you to receive notifications, updates, and information directly in Slack.

6. Collaboration Tools: Slack offers several built-in collaboration tools, such as:

- Calls and Video Conferencing: Users can initiate audio or video calls within Slack, reducing the need for external conferencing tools.

- Screen Sharing: Screen sharing allows team members to share their screens during meetings or troubleshooting sessions.

- App Integration: Slack supports integration with a wide range of apps and services, enabling automation, task management, and more.

7. Mobile and Desktop Apps: Slack provides mobile and desktop apps for seamless communication across devices. Encourage team members to install the app on their preferred devices for quick and easy access to communication channels and messages.

8. Etiquette and Guidelines: Establish communication guidelines and best practices to ensure effective and efficient communication within Slack. This can include guidelines for using appropriate channels, response times, and expectations for communication etiquette.

By implementing Slack as your communication tool, you can centralize communication, streamline collaboration, and improve overall team productivity and engagement.

Remember that this demonstration is just an overview, and there are numerous features and customization options available in Slack. Feel free to explore and customize Slack to fit your organization's specific needs and workflows.

TESTING & KPIS

What do you recommend in order to industrialize, automate and increase the quality of our tests? Should and can we automate all the tests? What are the best practices in the field? Please describe in great details your solution. It is extremely important to us. DevOps strongly advocates the use of indicators to know if we are doing things right and how well we're doing things. Any clues on the indicators to use?

To industrialize, automate, and increase the quality of your tests, I recommend implementing a robust testing framework and adopting best practices in the field of test automation. Here's a detailed solution that can help you achieve these goals:

1. Test Automation Framework:

- Develop or choose a suitable test automation framework that aligns with your technology stack, application architecture, and testing requirements. Popular frameworks include Selenium, Cypress, Appium, and JUnit.

- The framework should support various types of tests, including unit tests, integration tests, functional tests, and end-to-end tests.

2. Test Strategy and Coverage:

- Define a comprehensive test strategy that outlines the types of tests to be automated and the level of coverage required. This includes determining which tests can be automated and which should remain as manual tests.

- Prioritize your test cases based on criticality, risk, and business impact to ensure efficient allocation of testing resources.

3. Continuous Integration and Continuous Testing:

- Integrate your test automation framework with your CI/CD pipeline to trigger tests automatically with every code commit or deployment.

- Automate the execution of unit tests and integration tests as part of the CI process to detect issues early and ensure code stability.

- Consider running automated functional and end-to-end tests in dedicated testing environments or using virtualization/containerization technologies to create isolated and reproducible test environments.

4. Test Data Management:

- Establish a robust test data management strategy to ensure consistent and reliable test data for automated tests.

- Define and maintain test data repositories that provide the required data sets for different test scenarios.

- Utilize techniques such as data masking, synthetic data generation, or database snapshots to create and manage test data efficiently.

5. Test Automation Code and Version Control:

- Follow best practices for writing test automation code, including using meaningful variable and method names, employing proper commenting, and adhering to coding standards and guidelines.

- Version control your test automation code using a source code management system (e.g., Git) to track changes, collaborate with team members, and facilitate code review processes.

6. Test Reporting and Analysis:

- Implement a test reporting and analysis framework that provides insights into test execution results, test coverage, and overall quality metrics.

- Track key performance indicators (KPIs) such as test execution time, test success rate, defect density, and code coverage to evaluate the effectiveness of your testing efforts.

- Leverage visualization tools or dashboards to present test metrics in a meaningful and actionable way for stakeholders.

7. Test Environment Monitoring:

- Monitor the health and availability of your test environments to ensure the stability and reliability of your automated tests.

- Implement alerting mechanisms to notify teams in case of environment issues or failures that may impact test execution.

8. Test Maintenance and Review:

- Regularly review and update your automated test suite to align with application changes and evolving business requirements.

- Conduct periodic code reviews and refactor tests to improve code maintainability, readability, and reusability.

- Continuously analyze and optimize your test suite to identify and remove redundant or flaky tests.

In terms of KPIs, here are some indicators you can consider tracking:

1. Test Coverage: Measure the percentage of code or functionality covered by automated tests.

2. Test Execution Time: Monitor the time taken to execute automated tests to ensure efficiency and timely feedback.

3. Test Success Rate: Track the percentage of tests that pass successfully.

4. Defect Density: Calculate the number of defects discovered per unit of code or functionality.

5. Mean Time to Detect (MTTD) and Mean Time to Resolve (MTTR): Measure the average time taken to detect and resolve defects or issues identified during testing.

6. Code Coverage: Assess the percentage of

ORGANIZATION / CHANGE MANAGEMENT

Impact of DevOps on Different Aspects:

Team Structure: DevOps promotes cross-functional teams and encourages collaboration between developers, operations, and other stakeholders. Siloed roles and responsibilities may shift towards more integrated and collaborative teams, where individuals take shared ownership of delivering and maintaining software.

Mentality: DevOps fosters a culture of collaboration, communication, and continuous improvement. It encourages a mindset of automation, efficiency, and accountability. Team members are empowered to experiment, learn from failures, and embrace a continuous learning culture.

IT Delivery Process: DevOps emphasizes automation, continuous integration, continuous delivery, and continuous deployment. It streamlines the software delivery process, enabling faster and more frequent releases while maintaining quality and stability. This requires a shift towards practices like infrastructure as code, automated testing, and deployment pipelines.

Technical Architecture: DevOps promotes the use of scalable, modular, and loosely coupled architectures. It encourages the adoption of cloud-based infrastructure, microservices, and containerization technologies. These architectural changes support rapid scalability, easier deployment, and better resilience.

Toolchain: DevOps encourages the integration of various tools and technologies into a cohesive toolchain. This may include version control systems (e.g., Git), CI/CD platforms (e.g., Jenkins), configuration management tools (e.g., Ansible), containerization platforms (e.g., Docker), monitoring and alerting tools, and collaboration platforms (e.g., Slack). The toolchain should support automation, collaboration, and visibility across the software development and delivery lifecycle.

Risks of Human Resistance to Change:

Change, especially when it affects established processes and roles, can be met with resistance. Some potential risks associated with human resistance to DevOps adoption include:

Fear of Job Loss: Employees may fear that automation and process changes will make their roles obsolete or reduce job security. Clear communication about the purpose of DevOps and how it can enhance their work rather than replace them is crucial.

Lack of Awareness and Training: Insufficient understanding and lack of training on DevOps principles, practices, and tools can lead to resistance. Providing comprehensive training programs and resources to upskill and educate employees about DevOps is essential.

Change Fatigue: If an organization has recently undergone significant changes or is undergoing multiple concurrent changes, employees may experience change fatigue. Proper change management practices, including effective communication, involvement of employees in decision-making, and phased implementation, can help address this risk.

Cultural Shift: DevOps requires a cultural shift towards collaboration, shared responsibility, and continuous improvement. Resistance can arise when existing organizational culture and individual mindsets are deeply rooted in traditional practices. Leadership support, fostering a learning culture, and incentivizing collaboration can mitigate this risk.

Change Management Plan for DevOps Adoption:

To facilitate AAP's efficient adoption of DevOps, a comprehensive change management plan should be implemented. Here's an approach to consider:

Leadership Support: Ensure strong support and commitment from top-level management to drive the change and foster a culture of collaboration and continuous improvement.

Create Awareness: Conduct workshops, training sessions, and town hall meetings to educate employees about DevOps concepts, benefits, and the need for change. Highlight success stories from other organizations to inspire and motivate.

Assess Current State: Conduct a thorough assessment of the existing IT delivery process, team structure, technical architecture, and toolchain. Identify pain points, bottlenecks, and areas that need improvement to tailor the DevOps transformation plan accordingly.

Define Vision and Objectives: Clearly articulate the vision, goals, and expected outcomes of the DevOps transformation. This helps align everyone towards a common purpose and provides a direction for change.

Engage and Empower Employees: Involve employees in the decision-making process, seek their feedback, and address their concerns. Encourage cross-functional collaboration, knowledge sharing, and experimentation. Empower teams to take ownership of the transformation process.

Pilot Projects: Select a few pilot projects to showcase the benefits of DevOps. This helps build confidence, gain early wins, and provide tangible examples of success.

Communication Plan: Develop a comprehensive communication plan to ensure transparent and consistent communication at all levels. Regularly update employees on progress, milestones, and the impact of DevOps adoption on individuals and teams.

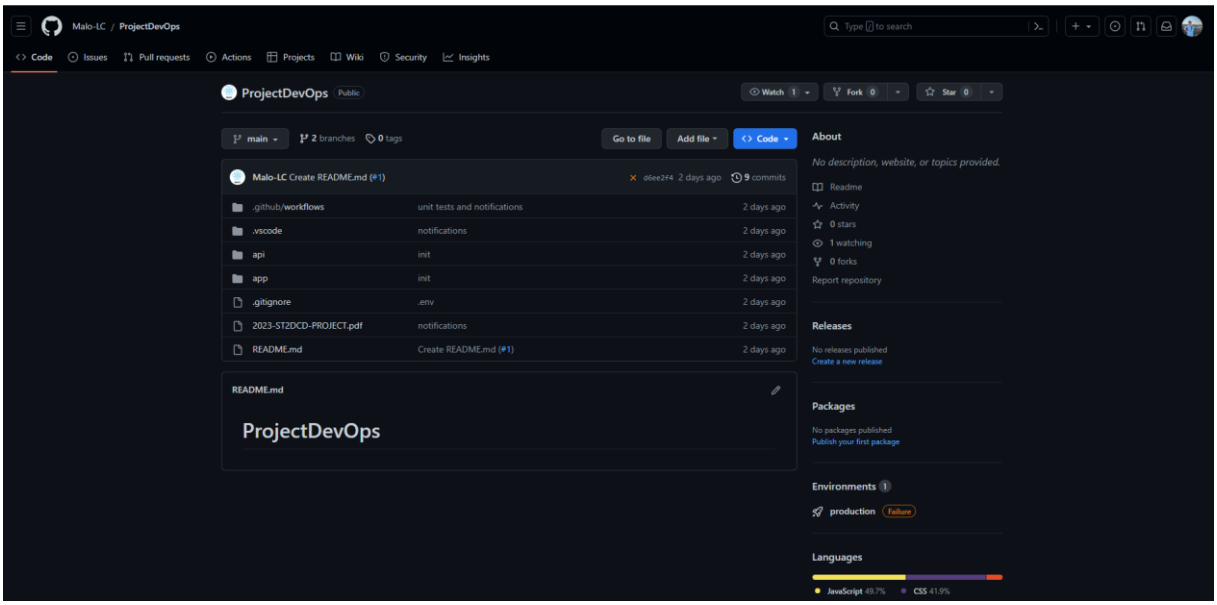
Training and Skill Development: Provide training programs and resources to upskill employees in relevant DevOps practices, tools, and technologies. Encourage continuous learning and create platforms for knowledge sharing.

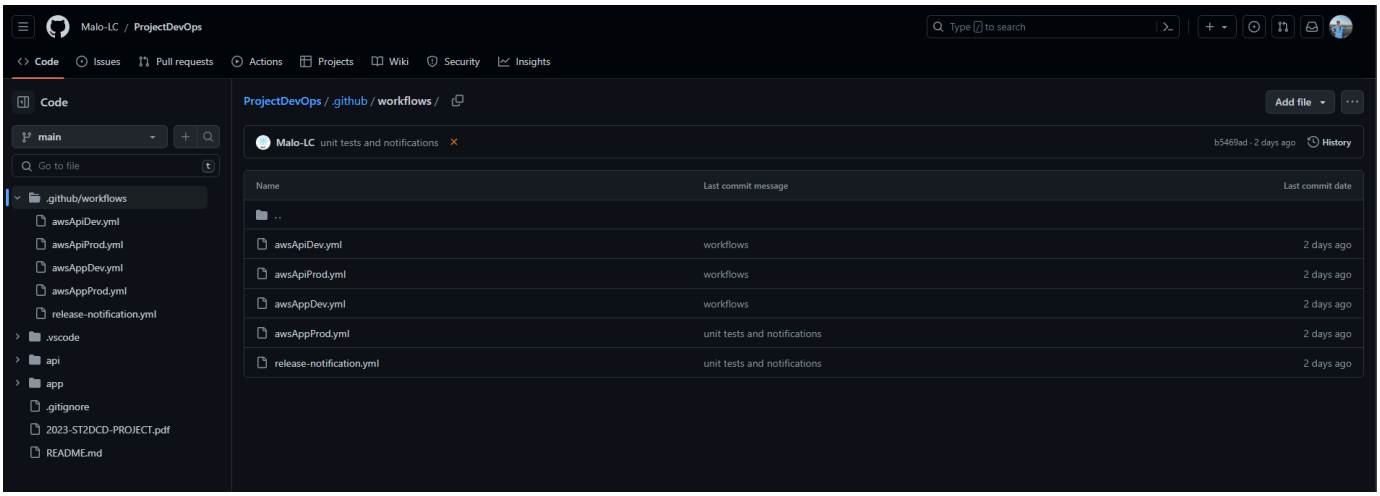
Measure and Celebrate Success: Establish key performance indicators (KPIs) to measure the progress and impact of DevOps adoption. Celebrate milestones and achievements to recognize and reinforce positive change.

Continuous Improvement: DevOps is a journey of continuous improvement. Encourage teams to share feedback, ideas, and lessons learned. Foster a culture of experimentation, innovation, and adapting to changing circumstances.

By following this change management plan, AAP can ensure a smooth and successful transition towards DevOps, empowering teams, improving collaboration, and enhancing IT delivery processes.

Proof Of Concept





```yaml
name: Deploy dev api to Amazon EC2
# on branch main and api folder
on:
  push:
    branches:
      - dev
    paths:
      - "api/**"

env:
  AWS_REGION: eu-west-3
  ECR_REPOSITORY: MY_ECR_REPOSITORY # set this to your Amazon ECR repository name
  ECS_SERVICE: MY_ECS_SERVICE # set this to your Amazon ECS service name
  ECS_CLUSTER: MY_ECS_CLUSTER # set this to your Amazon ECS cluster name
  ECS_TASK_DEFINITION:
    MY_ECS_TASK_DEFINITION # set this to the path to your Amazon ECS task definition
    # file, e.g. .aws/task-definition.json
  CONTAINER_NAME:
    MY_CONTAINER_NAME # set this to the name of the container in the
    # containerDefinitions section of your task definition

permissions:
  contents: read

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    environment: production

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ env.AWS_REGION }}

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1

      - name: Build, tag, and push image to Amazon ECR
        id: build-image
```

```yaml
name: Send release notifications
# Send notification on build complete, either success or failure
on:
  push:
    branches:
      - main
jobs:
  send-notification:
    runs-on: ubuntu-latest
    steps:
      - name: Notify on completion
        if: always()
        uses: rtCamp/action-slack-notify@v2.1.0
        with:
          status: ${{ job.status }}
          fields: repo,message,author,commit,action,eventName,ref,workflow,job,took
        env:
          SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}

      - name: Discord Notification
        id: discord-notification
        env:
          DISCORD_WEBHOOK: ${{ secrets.DISCORD_RELEASE_WEBHOOK }}
        uses: Ilshidur/action-discord@0.3.2
        with:
          args: ${{ steps.notification.outputs.message }}
```

```yaml
name: Deploy main app to Amazon EC2
# on branch main and app folder
on:
  push:
    branches:
      - main
    paths:
      - "app/**"

env:
  AWS_REGION: eu-west-3
  ECR_REPOSITORY: MY_ECR_REPOSITORY # set this to your Amazon ECR repository name
  ECS_SERVICE: MY_ECS_SERVICE # set this to your Amazon ECS service name
  ECS_CLUSTER: MY_ECS_CLUSTER # set this to your Amazon ECS cluster name
  ECS_TASK_DEFINITION:
    MY_ECS_TASK_DEFINITION # set this to the path to your Amazon ECS task definition
    # file, e.g. .aws/task-definition.json
  CONTAINER_NAME:
    MY_CONTAINER_NAME # set this to the name of the container in the
    # containerDefinitions section of your task definition

permissions:
  contents: read

jobs:
  deploy:
    name: Deploy
    runs-on: ubuntu-latest
    environment: production

    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ env.AWS_REGION }}

      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1

```