



PROJET PROFESSIONNEL DÉVOPS

DRIVEMANAGER

OUTIL DE GESTION DES TAXIS MÉDICALISÉS

**SAYFOUTDINOV SAID-TIMOUR
2023/2024**

REMERCIEMENT

Ce dossier représente l'aboutissement de mon année d'études chez Simplon (à distance). Je tiens à remercier l'ensemble des personnes m'ayant accompagné dans cette tâche.

- Patrice Guillaume (DSI), Responsable d'activité Service Informatique (AURA), Tiphaine Hanry (Simplon) et Leslie Bitene-Verrier (Simplon), Responsables de la promo, pour leur accompagnement et leurs conseils.*
- DSI pour m'avoir accepté et accompagné malgré les difficulté possiblement contraignant et un profil peu orthodoxe.*
- Simplon Grenoble pour m'avoir accepté malgré les aménagements peu compatible avec la formation.*
- Bien sûr je ne peux omettre ma reconnaissance envers les enseignants Thomas Laforge et Anthony Youssef pour leur implication dans notre réussite, leur organisation et leur enseignement de façon général.*
- Et pour finir à mes proches pour leur soutien moral et leurs conseils tout au long de cette formation.*

A toutes ces personnes, je présente mes remerciements, mon respect et ma gratitude.

SOMMAIRE

Introduction	4
Mise en place de la méthodologie devops.	5
Optimisation du Processus de Développement.	6
<i>Gestion du Versionnement, Collaboration, Assurance Qualité et Intégration Continue</i>	6
Bilan du projet.	15
<i>Problématiques rencontrés. Améliorations envisagés.</i>	15
<i>Discussion</i>	16
Conclusion	17
Bibliographie.....	18

INTRODUCTION

Présent écrit, est le fruit du travail, fait pour la certification DevOps, que nous faisons dans la cadre de la formation préparante au titre de niveau II, s'intitulant Concepteur Développeur d'Application, dans un environnement DevOps.

A la fin de cette introduction, nous allons présenter notre projet « chef d'œuvre » de façon sommaire, afin de donner une vision plus global sur le travail que nous menons.

Ensuite, dans la première partie du dossier, nous allons exposer la mise en place de la méthodologie DevOps. Egalement nous aborderons qu'est-ce que peut apporter cette approche.

Dans la suite de l'écrit, nous exposerons le pipeline que nous suivons et comment nous avons mis en place une infrastructure programmable. De ce fait, nous présenterons les outils de travail.

Notre projet porte sur la création d'un outil pour les entreprises de taxis médicalisé/VSL. Pour être plus précis, ce sont des entreprises agrémentées et sont spécialisées dans le transport des malades, dans le cadre des traitements ou de consultations médicales. L'outils que nous souhaitons proposer, est un outils de gestion centralisé des courses, clients et chauffeurs.

Au moment de la production de cet écrit, le travail sur le projet avance de plus en plus vite car le passage de titre se rapproche. A ce moment même, nous commençons le travail sur le front-end et l'application pour les chauffeurs. Cependant les discussions sur la façon de travail ne sont pas terminées. Toutefois nous faisons de notre mieux pour accomplir ce dossier, en donnant des exemples issu de notre projet.

MISE EN PLACE DE LA MÉTHODOLOGIE DEVOPS.

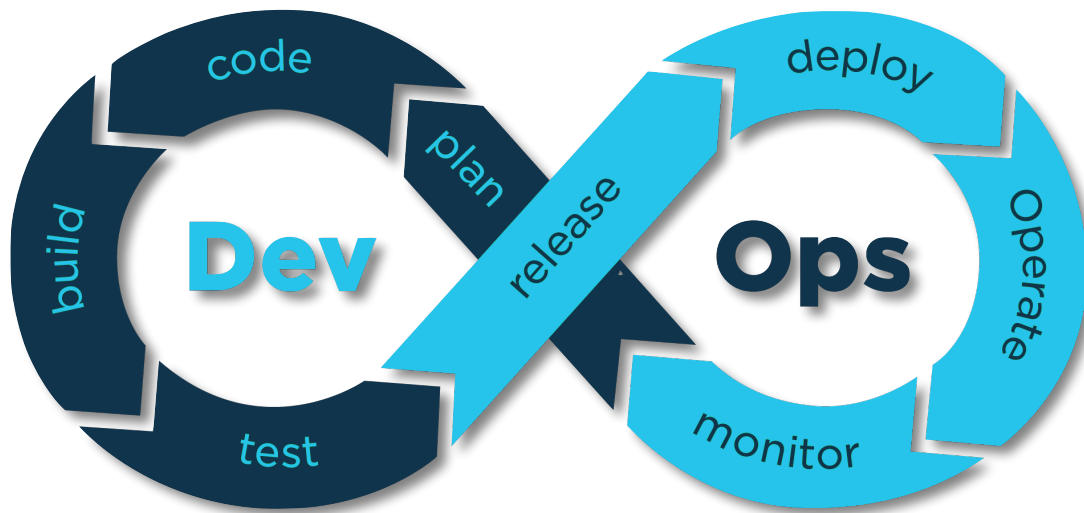


Fig. 1

Dans cette partie, nous allons aborder les outils que nous avons utilisé pour la mise en place de l'approche DevOps et nous argumenterons sur nos choix.

Les intitulés des sections qui suivront, sont des reformulations des intitulés des compétences listées dans le référentiel « Certification DevOps ».

Il serait probablement possible de constater, que certains arguments, sont des rappels directs des compétences que nous voulons valider par le biais de l'outil ou approche utilisé.

Globalement, notre façon de travailler est basé sur l'approche dite déclarative, car nous avons défini les environnements de travail avant de commencer notre travail.

Egalement, nous sommes parti du postula suivant:

« C'est le cumul d'itérations et de feedback sur le code qui vont nous permettre d'améliorer sa fiabilité, sa stabilité, sa performance et sa lisibilité. »

OPTIMISATION DU PROCESSUS DE DÉVELOPPEMENT.

GESTION DU VERSIONNEMENT, COLLABORATION, ASSURANCE QUALITÉ ET INTÉGRATION CONTINUE

Pour commencer l'exposition du travail effectué, nous avons décidé de débiter par la présentation de la gestion du versionnement du code. Car l'outil que nous avons adopté nous était déjà familier.

Ainsi, nous poser trop questions nous avons décidé d'utiliser Git et GitHub. Dans un premier temps, ce choix vient du fait que, tout au long du parcours d'apprentissage nous avons utilisé ces outils.

Dans un second temps, ayant prévu utiliser GitHub Actions pour l'automatisation, le GitHub est juste incontournable. Alors nous nous sommes demandé:

- *Pourquoi changer l'équipe qui gagne?*

Également, dans le futur, quand de nouveaux collaborateurs intégreront le projet, nous serons certains qu'ils sauront l'utiliser.

En utilisant GitHub pour ce projet, nous avons fait le choix de créer plusieurs repositories. C'est à dire, nous allons nous servir d'un repository pour le back-end, d'un autre pour le front-end et d'un dernier pour le code de l'application mobile des chauffeurs.

Nous pensons que cette approche nous convient le plus, car l'application de standardiste et l'application de chauffeurs ne sont pas similaires et nous ne risquons pas de nous retrouver face à la question:

- Combien de fois tel ou tel composant est-il utilisé?

Aussi dans le futur quand ce projet prendre l'ampleur, il sera plus facile d'attribuer les permissions d'accès aux collaborateurs.

Afin de mieux articuler la présentation, le fonctionnement de l'intégration continue sera abordé dans la suite de cette partie. Avant nous exposerons l'approche de gestion de versionnement du code, que nous appliquerons à nos

trois référentiels, et puis nous passerons à la présentation de notre pipeline. Dans la figure 2, nous avons voulu illustrer le modèle de gestion de branche que nous avons adopté.

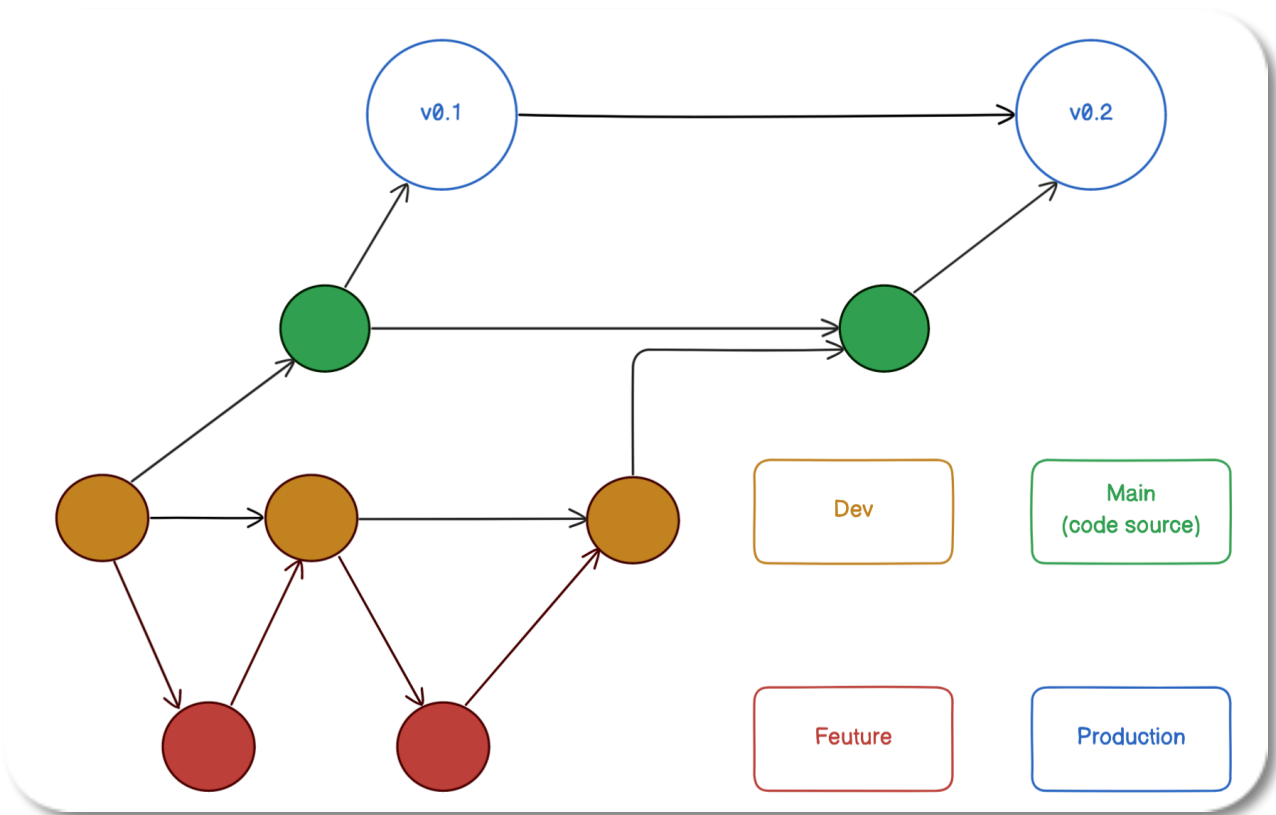


Fig. 2

Pour les trois repositories, il était décidé d'utiliser trois branches principales, « Dev », « Main » et « Production ». Ces branches sont permanentes, alors que la branche « Feature » va être utilisée pour la mise en partage de nouvelles fonctionnalités ou de correctifs. Ce ne sera pas une branche permanente. Elle sera supprimée, après chaque fusion avec la branche « Dev » et recrée au besoin.

La branche « Dev », sera utilisée pour faire passer les premiers tests automatisés et puis être confronté à un « build », dans un environnement autre que celui de la machine du développeur, avant d'être fusionnée avec la branche « Main ».

La branche « Main » aura plus au moins le même fonctionnement que « Dev ». C'est à partir de cette branche que nous allons envoyer le code dans la

branche « Production ». Suite à cela, la branche « Production », sera utilisé pour le déploiement de notre repository.

Afin d'avancer l'exposé du travail effectué dans le cadre du projet et de la certification DevOps, nous présenterons la pipeline adopté.

De ce fait, cette section du chapitre présentera nos compétences du contrôle du code produit, de l'automatisation des tests dans un environnement configuré pour la CI et CD. Et après cela, nous aborderons la veille technologique permettant le suivi de l'application en temps réel.

Ainsi, pour mieux expliquer cette partie de travail, nous avons joins l'illustration de pipeline (fig. 3). Cependant, étant donné que nous avons séparé en trois repository différents notre projet, il nous fallait « trois » pipelines pour chaque repository. Nous avons décidé de joindre ici la pipeline de Front-end, la seul différence que nous aurons avec celle pour le back, c'est qu'il n'y aura pas de tests end-to-end.

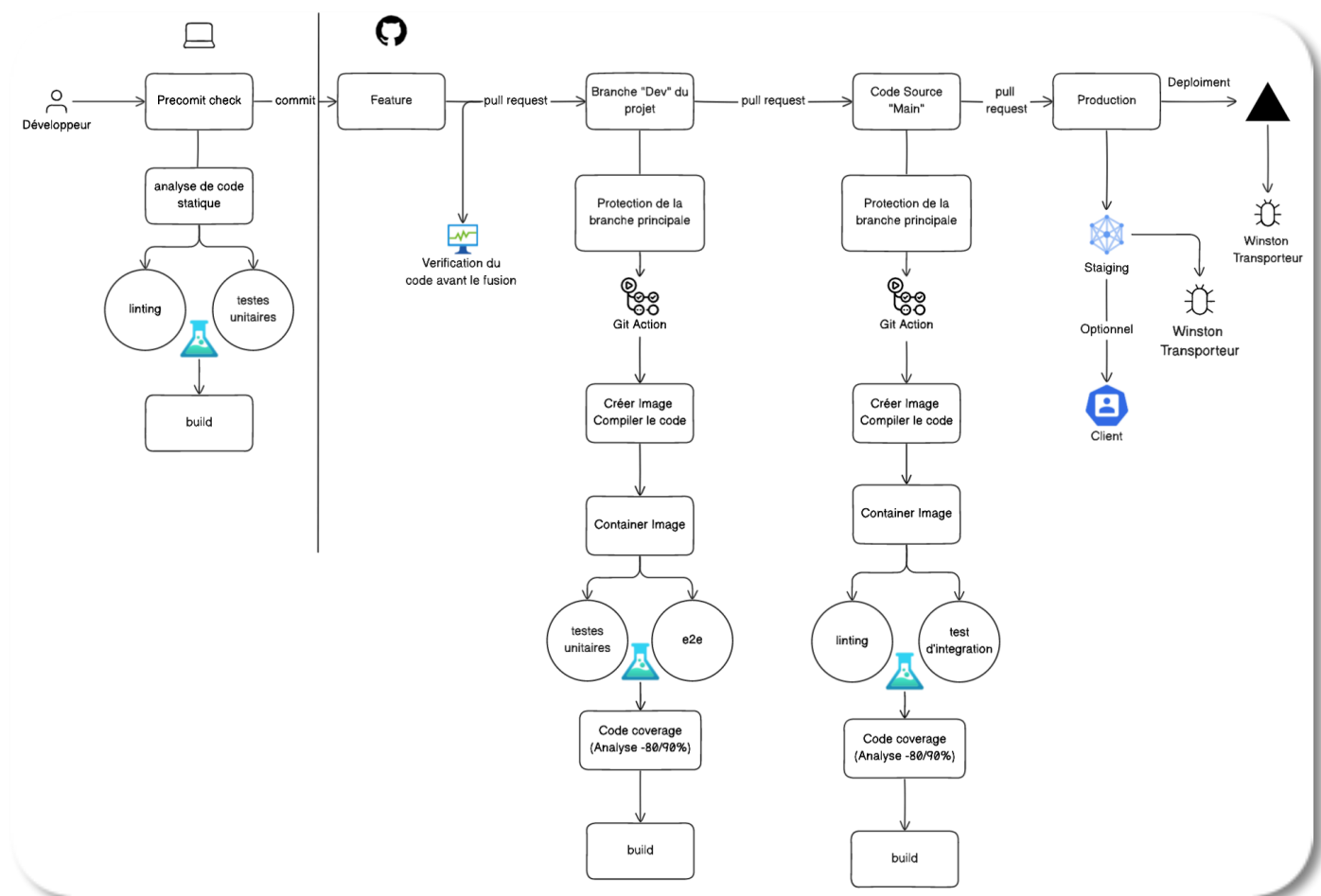


Fig. 3

Sur cette illustration, il est possible de constater que, nous avons fait une séparation en deux, entre la machine du développeur et GitHub, avec le trait vertical. Également, il est possible de voir que nous avons plusieurs série de tests pour chaque branche permanente.

Cette mise en place de tests, était inspiré par l'article s'intitulant « *Traversée de la pyramide d'automatisation des tests* »¹ et l'illustration de la pyramide qui est présente dans cet article et que nous avons intégré dans ce dossier (fig. 4). Cette illustration se lit de bas vers le haut.

Sur cette illustration, nous pouvons constater que les auteurs présentent les tests unitaires comme des tests de « base » et donc ils devraient être fait en premier. Une fois ces tests passés, nous passons à la couche « service » et elle fait référence aux tests d'intégration. A la fin nous retrouvons la couche e2e, qui fait référence aux tests end-to-end, donc ce sont les tests qui vont vérifier le fonctionnement de l'application dans sa globalité.



Fig. 4

Passons maintenant à la lecture de notre pipeline. Dans un premier temps, nous avons fait le choix de mettre en place des pre-commit checks. Les pre-commit checks s'expliquent de façon suivante: avant d'envoyer le travail vers le GitHub, nous allons faire les tests à l'issue desquels nous allons pouvoir envoyer notre travail dans la branche « Feature », uniquement si les tests et le build sont passés sans erreurs. Dans le cadre du pre-commit check nous souhaitons que les développeurs dans le futur et nous même fassions des tests de linting et des tests unitaires.

¹ Cet article est peut être lu à l'adresse suivante: <https://fr.parasoft.com/blog/testing-automation-pyramids-for-software-development/>

Dans les figures 6(package.json), 7 (eslintrc.json) et 8 (.prettierrc), nous exposons nos fichiers de configuration, pour faire fonctionner les tests de linting.



Fig. 5

Les tests de linting vont être assurés par ESLint et Prittier. Les raisons pour lesquelles, nous les avons choisi, sont assez simples. Nous utilisons cette combinaison d'outils car ESLint va vérifier la syntaxe et des variables fantômes « non utilisées » et Prettier va s'occuper uniquement du format stylistique car nous voulons avoir un code formaté horizontalement et verticalement comme sur la figure 5. Nous considérons ce point, comme important car, il permet une meilleur lisibilité au code, puis notre but est de pouvoir travailler facilement avec le code. Pour cela, nous nous imposons la convention de nommage des variables, « camelCase », mais aussi nous allons nous obliger de nommer des variables de façon explicite, afin de comprendre leur utilité dès la première lecture du code. Puisque un code autodocumenté est un code qui peut être lu, compris et maintenu rapidement par les développeurs extérieurs.

```

{
  ...
  "scripts": {
    "build": "npx tsc",
    "start": "node dist/server.js",
    "dev": "concurrently \"npx tsc --watch\" \"nodemon -q dist/server.js\"",
    "lint": "eslint . --ext .ts",
    "lint:fix": "eslint . --ext .ts --fix",
    "format": "prettier --write 'src/**/*.ts' --config ../.prettierrc",
    "test": "cross-env NODE_ENV=test jest --testTimeout=10000"
  },
  ...
}

```

Fig. 6

Maintenant nous allons argumenter nos choix de ces outils. ESLint, étant un outils configurable, il nous permettra d'avoir une cohérence syntaxique. Egalement, c'est un outil qui nous donnera la possibilité d'identifier des bugs directement à l'écriture, plutôt que pendant le build de l'application.

Le choix d'utiliser Prettier était assez évident car dans les articles que nous avons pu lire, la combinaison de ces outils est encouragé.

Pour terminer cette partie, portant sur linting, nous allons aborder rapidement l'outil que nous voulons, en tant que générateur de

```

{
  "root": true,
  "env": {
    "browser": true,
    "es2021": true,
    "node": true
  },
  "extends": [
    "eslint:recommended",
    "plugin:@typescript-eslint/eslint-recommended",
    "plugin:@typescript-eslint/recommended"
  ],
  "parser": "@typescript-eslint/parser",
  "parserOptions": {
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "plugins": [
    "@typescript-eslint",
    "prettier"
  ],
  "rules": {
    "quotes": [
      "error",
      "double"
    ],
    "camelcase": [
      "error",
      {
        "properties": "never",
        "ignoreDestructuring": true
      }
    ],
    "no-duplicate-imports": "error",
    "no-unused-vars": "error"
  }
}

```

Fig. 7

documentation. Cette tâche peu amusante, sera fait par Storybook, car c'est un outil « open source » qui permet de créer des composants isolés à des fins de documentations, et il est compatible avec les frameworks, front-end, utilisés.

Dans la suite de pre-commit check, nous lançons les tests unitaires localement pour nous assurer du bon fonctionnement de chaque composant du code.

Pour cette étape, nous utiliserons Jest en le combinant avec supertest. Jest est un framework de test JavaScript maintenu par une communauté de contributeurs open source et d'employés de Facebook. Nous l'avons choisi car il est compatible avec TypeScript et il est capable de compiler le code TS avant d'exécuter les tests automatiquement. Également, cet outil a une documentation abondante et dû à son large adaptation il bénéficie de nombreux tutoriels, facilitant ainsi sa maîtrise.

Après la validation des tests unitaires, nous procéderons à un build en local pour vérifier la compilation du code. A ce stade, nous utilisons le TypeScript Compiler (tsc), puisque c'est un outil officiel, il est conçu pour être conforme aux normes et spécifications de TypeScript (*alors nous nous sommes demandé: pourquoi changer l'équipe qui gagne?*).

Une fois que l'étape de pre-commit check est accompli, nous allons pousser notre branche local « Feature » vers la branche distante, pour que l'utilité du code soit vérifié. Après la validation, le code va être ajouté au code se trouvant dans la branche « Dev ». C'est à ce moment la que GitHub Action va prendre le relais,



Fig. 8

grâce à la configuration du fichier YAML dans le gitflow (fig. 9). De façon général nos fichier YAML se ressemblent donc c'est pour cela que nous avons joint un seul fichier.

GitHub Actions procédera à la compilation du code pour faire des tests unitaire et des tests end-to-end (si c'est le repository du front de l'application). Encore une fois, nous aurons recours aux tests unitaire avec Jest, seule différence, c'est que cette fois le lancement est automatisé. Pour les tests end-to-end, nous allons utiliser Playwright. Ce choix était fait suite à la lecture du témoignage de Diaz, L. (2023). *Why I switched from Cypress to Playwright*. Medium.com. A la fin des tests GitHub Action va refaire un build pour nous assurer un code fonctionnel.

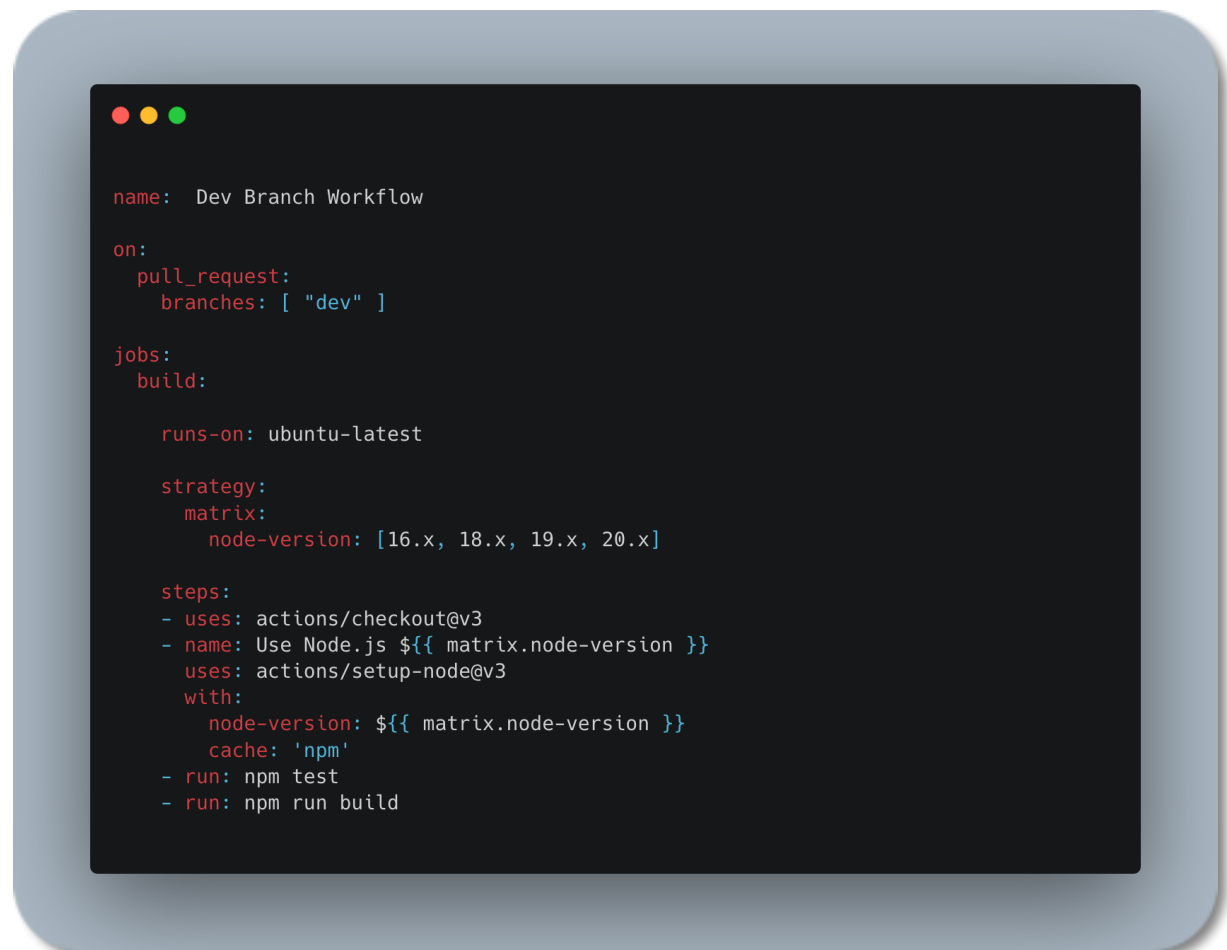
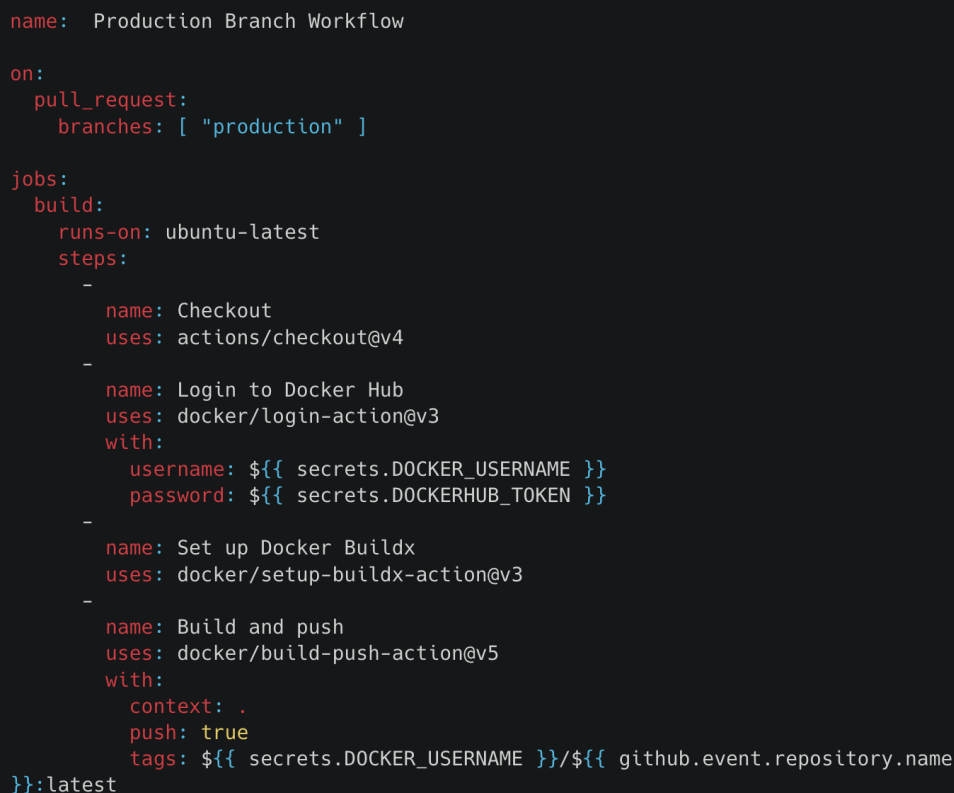


Fig. 9

A ce stade notre branche « Dev » ne doit pas avoir d'erreur, pour que nous puissions continuer la CI et évoluer vers la CD. Donc, il nous restera encore une batterie de tests. Ces sont les tests linting et les tests d'intégration. Pour linting, nous

utiliserons les même outils qu'en local. Pour les tests d'intégrations nous avons choisi d'utiliser SuperTest avec Chai. Une fois les tests validés, cette branche enverra le code vers la branche de production. À son tour la branche « Production » aura deux fonction assez importantes, déployer et envoyer vers DockerHub son code, pour qu'il soit testable dans un environnement de staging et éventuellement présenté au client pour un retour.

Pour suivre notre application de manière quotidienne nous allons mettre en place « Winston », un outil de journalisation. Il va nous aider à tracer des éventuels erreurs.



```
name: Production Branch Workflow

on:
  pull_request:
    branches: [ "production" ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      -
        name: Checkout
        uses: actions/checkout@v4
      -
        name: Login to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${ secrets.DOCKER_USERNAME }
          password: ${ secrets.DOCKERHUB_TOKEN }
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      -
        name: Build and push
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ${ secrets.DOCKER_USERNAME }/${ github.event.repository.name
}}:latest
```

Fig. 10

BILAN DU PROJET.

PROBLÉMATIQUES RENCONTRÉS. AMÉLIORATIONS ENVISAGÉS.

Comme il était dit dans l'introduction, le travail sur le dossier était tombé juste au moment où nous travaillons sur le projet, ce qui fait que nous ne sommes pas complètement sûr de nos choix, comme par exemple l'élaboration des pipelines qui nous fait douter sur le choix d'utiliser plusieurs repositories, car nous n'avons pas encore assez de connaissances pour mener à bien certains tests avec les trois repositories. Et donc nous réfléchissons encore sur ce point. Par conséquent nous pensons de refaire un repository unique, en parallèle, pour tout le projet et refaire les tests de pipeline. Cela nous permettra de comparer les deux approches et de choisir la meilleur. Certainement si nous changeons notre approche il nous faudra revoir la configuration de pipeline.

Autre soucis que nous avons pu rencontrer, sont liés aux choix des framework pour les tests, car ne connaissant pas tous les outils existants il est facile de se tromper d'outil. Pour surmonter ce problème, nous nous sommes résolu à lire et regarder plusieurs témoignages et articles possibles, ce qui a bien enrichie notre bibliographie. Bien sûr il est peu probable que nous allons changer ses outil dans le cas du changement de gestion de notre code sur GitHub. Toutefois, nous continuons la recherche au sujet des tests unitaires, d'intégration et d'end-to-end.

Dans le cadre de certification, nous devons mettre en place des micro-services. Cependant, nous ne l'avons pas fait car nous n'avons pas trouvé de réel utilité, étant donné que le projet est nouveau, il n'est pas était confronté à la « vie réelle » et donc nous ne pouvons pas savoir avec exactitude quels sont les services qui vont être surchargés et à quel endroit il vaut mieux « scier » l'application.

Également nous avons lu un article qui s'intitule en russe « *Сначала – монолит, или правильный путь к микросервисной архитектур* » et en français, ce titre se traduit comme « D'abord - monolithe, ou meilleur chemin vers l'architecture en micro-services ». Dans cet article, l'auteur stipule que la grande partie des applications au début étaient monolithiques, et avec le temps et la prise de l'ampleur, elles étaient allé vers les micro-services. Aussi l'auteur conseille de

suivre le principe « YAGNI² », et dans notre projet cela est bien d'actualité car, il est pas si grand pour être « scié » en plusieurs morceaux. Ainsi nous pensons que ces raisons suffisent pour l'instant le choix de ne pas développer de micro-services et faire une application monolithique.

DISCUSSION

Au cours de nos recherches pour ce projet, nous avons lu plusieurs témoignages et articles sur la méthodologie DevOps et les outils. Le fait de voir différentes propositions, fait que même au cours de l'écriture du dossier nous ne sommes pas sûr des choix, mais nous avons décidé quand même de les tester et voir si cela fonctionne ou pas. Cette approche nous oblige à beaucoup travailler et vite pour être dans les temps. Il y a aussi des bons côtés, les témoignages que nous avons exploré, nous ont permis de rejeter certaines propositions et en garder d'autres. Comme par exemple, nous n'avons pas adopté l'utilisation de « Husky Precommit Hook » alors que nous trouvons que cet outil est assez pratique. Nous ne l'utilisons pas car dans notre infrastructure automatisé, il serait préférable que nous regardions quand même le code, car cela nous permettrait de mieux intégrer les conventions. Il est possible que cet outil ferait partie de Stack Technique dans le futur proche, une fois que nous aurons plus d'aisance avec notre projet.

Aussi nous avons intégré l'outil de journalisation, mais c'est un outil que nous n'avons jamais utilisé et donc nous n'avons pas pu en donner d'exemples car nous ne sommes pas encore arrivés à l'étape, au cours duquel nous devons l'utiliser. Cependant, il est éventuellement possible qu'au moment du passage de titre nous aurions l'occasion de le tester.

² Yagni ou You aren't gonna need it en anglais et en français « Vous n'en aurez pas besoin » - Dans le contexte de la programmation expérimentale, c'est une méthode souvent utilisée dans les équipes agiles de développement de logiciels. Elle peut être définie comme un mantra qui indique qu'une fonctionnalité que nous présumons nécessaire pour notre logiciel dans le futur, ne devrait pas être développée maintenant car « vous n'en aurez pas besoin ». Source: <https://martinfowler.com/bliki/Yagni.html>

CONCLUSION

Pour conclure ce dossier et partager mon avis général sur le dossier, le travail qui était fait et qui encore devrait être fait, je voudrais écrire un constat qui serait peu pertinent mais tout de même important. Ce dossier peut être encore perfectionné et retravaillé afin de pleinement démontrer les compétences nécessaires pour la validation de la certification DevOps.

En relisant le dossier je vois certaines choses qui sont selon mon avis novice, parfois maladroites, parfois compliqués pour être utilisée ou même mal réfléchies. Cependant pour me justifier, mais surtout pas me défendre car cette fois je n'ai pas su m'adapter au travail avec la personne avec laquelle je devais travailler, le projet présenté ici et toute la réflexion qui est faite autour, est le fruit de mon travail avec moi-même ce qui assez dommage, car je pense qu'une collaboration à plusieurs aurait été bien plus bénéfique pour la mise en place de l'environnement et du projet en lui même.

Je voudrais ajouter aussi que l'écriture de ce dossier m'a permis de remettre en question certaines approches que j'ai employé. Cela m'a permis de réfléchir sur l'utilité des outils que j'utilise et je n'ai pas manqué de le dire plus haut dans la discussion. Aussi il m'a permis de trouver des articles qui me seront très utiles dans le futur. Ce dossier est un bon point sur les connaissances que j'ai acquis et que je devrais acquérir au plus vite pour pouvoir intégrer un environnement DevOps.

BIBLIOGRAPHIE

A successful Git branching model. (s. d.). nvie.com. <https://nvie.com/posts/a-successful-git-branching-model/>

Atlassian. (s. d.-a). *Principes des microservices* | Atlassian. <https://www.atlassian.com/fr/microservices>

Atlassian. (s. d.-b). *SLA, SLO et SLI : les différences* | Atlassian. <https://www.atlassian.com/fr/incident-management/kpis/sla-vs-slo-vs-sli>

Atlassian. (s. d.-c). *SLA : Tout ce que vous devez savoir* | Atlassian. <https://www.atlassian.com/fr/itsm/service-request-management/slas>

Benbrahim, R. (2022, août 31). *ESLINT : Comment coder proprement en JavaScript* - WeLoveDevs.com. <https://welovedevs.com/fr/articles/eslint/>

Benefits of a DevOps environment | MuleSoft. (s. d.). MuleSoft. <https://www.mulesoft.com/resources/api/devops-environment-benefits>

Bogdan Stashchuk. (2021a, novembre 4). *Back-end development and APIs - FreeCodeCamp tutorial* [Vidéo]. YouTube. <https://www.youtube.com/watch?v=hHLmb3OD7Mo>

Bogdan Stashchuk. (2021b, novembre 4). *Back-end development and APIs - FreeCodeCamp tutorial* [Vidéo]. YouTube. <https://www.youtube.com/watch?v=hHLmb3OD7Mo>

Catoire, L. (2023, 16 mai). *Quelle architecture de projet choisir entre micro-services et monolithe modulaire ?* *Efficience IT*. <https://www.itefficiency.com/article/quelle-architecture-de-projet-choisir-entre-micro-service-ou-monolithe-modulaire>

Chaitanya, K. (2021, 13 décembre). *Rest API testing with Mocha, Chai, SuperTest* - Kishan Chaitanya - Medium. <https://kishanchaitanya.medium.com/api-testing-using-mocha-chai-and-supertest-a7c7edc96c24>

Chazelle, J. (2020, 14 juillet). *Utiliser ESLINT et Prettier pour un code de qualité*. Jérémie Chazelle. <https://jeremiechazelle.dev/utiliser-eslint-et-prettier-sous-visual-studio-code-webstorm-phpstorm-pour-un-code-de-qualite/>

Code documentation for JavaScript with JSDOC : An Introduction. (2020, 2 février). <https://www.valentinog.com/blog/jsdoc/>

Contributeurs aux projets Wikimedia. (2023, août 2). *JSDOC*. Wikipédia. <https://fr.wikipedia.org/wiki/JSDoc>

Cypress, Tests de bout en bout (E2E) | Fullwave. (s. d.). Fullwave. <https://fullwaveagency.com/blog/cypress-tests-de-bout-en-bout-e2e/>

Darras, G. (2019, 17 septembre). *Documenter son projet avec Storybook*. Blog Ineat. <https://blog.ineat-group.com/2019/09/documenter-son-projet-avec-storybook/>

De Best, B. (2022, 31 mai). *Plan DevOps - SLA et exigences non fonctionnelles*. Welcome IT Professional. <https://fr.itpedia.nl/2017/07/14/devops-plan-slases-and-non-functional-requirements/>

Delattre, L. (2022, 7 septembre). *DevOps : de la philosophie aux outils*. IT For Business. <https://www.itforbusiness.fr/devops-de-la-philosophie-aux-outils-40753>

DevOps Journey. (2023, 17 octobre). *How to design a modern CI/CD pipeline* [Vidéo]. YouTube. <https://www.youtube.com/watch?v=KnSBNd3b0ql>

Diaz, L. (2023, 7 novembre). *Why I switched from Cypress to Playwright* - Lucy Diaz - medium. Medium. <https://medium.com/@oldiazg/why-i-switched-from-cypress-to-playwright-dc41ce4d5e1b>

Documentation - JSDOC reference. (s. d.). <https://www.typescriptlang.org/docs/handbook/jsdoc-supported-types.html>

Doglio, F. (2023, 12 avril). *Documenting your TypeScript projects : There are options*. Medium. <https://blog.bitsrc.io/documenting-your-typescript-projects-there-are-options-da7c8c4ec554>

Driat, L. (2021, 6 janvier). *Écrire des commentaires de code PARFAITS : le guide ultime*. IT Expert. <https://itexpert.fr/blog/commentaires-parfaits/>

Écrivez du code autodocumenté. (s. d.). OpenClassrooms. <https://openclassrooms.com/fr/courses/6398056-ecrivez-la-documentation-technique-de-votre-projet/6904236-ecrivez-du-code-autodocumente>

Frontend Channel. (2021, août 19). *Ликбез по CI/CD для frontend'а на примере GitLab / Тимофей Тиунов* [Vidéo]. YouTube. https://www.youtube.com/watch?v=BIY_J0Ba4Cc

Gérer les assertions avec CHAi – formation tests en JavaScript. (s. d.). Grafikart.fr. <https://grafikart.fr/tutoriels/assertions-chai-654>

Goffinet, F. (2020, août 9). *Introduction à l'Infrastructure as Code*. cisco.goffinet.org. <https://cisco.goffinet.org/ccna/automation-programmabilite-reseau/infrastructure-as-code/>

Haaker, W. (2024, 25 janvier). *Comment les pyramides d'automatisation des tests sont-elles utilisées dans le développement de logiciels ?* Parasoft. <https://fr.parasoft.com/blog/testing-automation-pyramids-for-software-development/>

Isaiah, A. (2023, 25 octobre). *A complete guide to Winston logging in Node.js*. Better Stack Community. <https://betterstack.com/community/guides/logging/how-to-install-setup-and-use-winston-and-morgan-to-log-node-js-applications/>

Jacobsen, T. (2023, 14 décembre). *Mono-repository : nos développeurs réalisent une transition majeure pour collaborer plus efficacement*. Infomaniak Network News. <https://news.infomaniak.com/monorepo-collaborer-efficacement/>

Khatrī, M. F. (2023, 6 janvier). *API testing using SuperTest !* - Mohammad Faisal Khatrī - Medium. Medium. <https://medium.com/@iamfaisalkhatrī/api-testing-using-supertest-ea37522fa329>

Losoviz, L. (2023, 12 septembre). Mono-repo vs multi-repo : avantages et inconvénients des stratégies de dépôt de code. Kinsta®. <https://kinsta.com/fr/blog/mono-repo-vs-multi-repo/>

Maniez, D. (s. d.). *La notation hongroise*. Developpez.com. <https://dominiquemaniez.developpez.com/Nothongroise/>

Martinez, J. (2023, 9 mars). À la découverte des monorepos pour partager son code JS | DARVA. DARVA. <https://www.darva.com/fr/2023/03/02/a-la-decouverte-des-monorepos-pour-partager-son-code-js/>

Martins, J. (2023, 26 septembre). Qu'est-ce qu'un KPI (indicateur de performance) ? [2023] • Asana. Asana. <https://asana.com/fr/resources/key-performance-indicator-kpi>

Mazzoni, L. (2023, 17 février). Pourquoi utiliser Cypress ? - UpSkill4IT. UpSkill4IT. <https://upskill4it.com/pourquoi-utiliser-cypress/>

McKendrick, J. (2017, 20 avril). Pourquoi les microservices ne sont peut-être pas faits pour tout le monde. ZDNetFR. <https://www.zdnet.fr/amp/actualites/pourquoi-les-microservices-ne-sont-peut-etre-pas-faits-pour-tout-le-monde-39851458.htm>

Monolithic vs. Microservices : Why decoupled and headless architectures are the future | Contentstack. (s. d.). <https://www.contentstack.com/cms-guides/monolithic-vs-microservices-cms-architectures>

Monorepo : Est-ce que ça vaut le coup ? | Maxence Poutord. (2020, 22 décembre). Monorepo : est-ce que ça vaut le coup ? | Maxence Poutord. <https://www.maxpou.fr/monorepo-pros-and-cons-fr>

NPM : Winston. (s. d.). Npm. <https://www.npmjs.com/package/winston>

Notation hongroise : Définition et explications. (s. d.). Techno-Science.net. <https://www.techno-science.net/definition/11395.html>

Orie, C. (2019, 8 novembre). Testing NodeJS/Express API with JEST and SuperTest. DEV Community. <https://dev.to/nedsoft/testing-nodejs-express-api-with-jest-and-supertest-1km6>

PlayerZero. (2023, 15 février). Les 10 meilleures bibliothèques de journalisation Node.js. HackerNoon. <https://hackernoon.com/fr/les-10-meilleures-biblioth%C3%A8ques-de-journalisation-nodejs>

Pourquoi et comment créer des modules sur NodeJS. (2016, 23 septembre). M@XCode. <https://maximilienandile.github.io/2016/09/23/Pourquoi-et-comment-cree-un-module-avec-nodejs/>

Pourquoi la culture DevOps est-elle bénéfique pour votre entreprise ? (s. d.). <https://carrieres.groupeozitem.com/blog/culture-devops>

Qu'est-ce que le Daily Scrum Meeting ? (s. d.). <https://www.hubvisory.com/fr/blog/le-daily-scrum-meeting>

Réalisez vos premiers tests unitaires avec JEST. (s. d.). OpenClassrooms. <https://openclassrooms.com/fr/courses/7159306-testez-vos-applications-front-end-avec-javascript/7332796-realisez-vos-premiers-tests-unitaires-avec-jest>

Sécurisez le traitement des erreurs et des logs. (s. d.). OpenClassrooms. <https://openclassrooms.com/fr/courses/1761931-securisez-vos-applications/5702858-securisez-le-traitement-des-erreurs-et-des-logs>

Solderea, I. (2024, 6 janvier). Cypress vs Playwright : A Detailed Comparison | LambdaTest. LambdaTest. <https://www.lambdatest.com/blog/cypress-vs-playwright/>

Staging Environment – RYTE Wiki - Wiki du marketing digital. (s. d.). https://fr.ryte.com/wiki/Staging_environment

Stemmler, K. (2021, 19 décembre). How to use ESLint with TypeScript | Khalil Stemmler. [khalilstemmler. https://khalilstemmler.com/blogs/typescript/eslint-for-typescript/](https://khalilstemmler.com/blogs/typescript/eslint-for-typescript/)

Test d'intégration : quand et pourquoi ? (s. d.). <https://www.mobiapps.fr/blog/test-dintegration-quand-et-pourquoi>

Testim. (2022, 2 février). TypeScript Unit Testing 101 : A Developer's Guide. AI-driven E2E automation with code-like flexibility for your most resilient tests. <https://www.testim.io/blog/typescript-unit-testing-101/>

Thomas. (2021, 9 décembre). Partage d'expérience : un tuto inspiré des méthodes de travail CentreOn pour améliorer le travail d'équipe sur REACT avec ESLINT et Prettier - Centreon. Centreon. <https://www.centreon.com/fr/partage-dexperience-un-tuto-inspire-des-methodes-de-travail-centreon-pour-ameliorer-le-travail-dequipe-sur-react-avec-eslint-et-prettier/>

Veiller à la qualité de votre code et sa documentation. (s. d.). CNIL. <https://www.cnil.fr/fr/veiller-la-qualite-de-votre-code-et-sa-documentation>

Vérifiez que votre code respecte les conventions de codage. (s. d.). OpenClassrooms. <https://openclassrooms.com/fr/courses/7697016-creez-des-pages-web-dynamiques-avec-javascript/7911268-verifiez-que-votre-code-respecte-les-conventions-de-codage>

Wallen, J. (2022, 7 novembre). How to build a docker image and upload it to Docker Hub. TechRepublic. <https://www.techrepublic.com/article/how-to-build-a-docker-image-and-upload-it-to-docker-hub/>

webDev. (2020, 4 février). Просто о CI/CD (Непрерывная интеграция и доставка) [Vidéo]. YouTube. <https://www.youtube.com/watch?v=7S1ndRRht6M>

What Is an API Gateway ? (s. d.). Nginx. <https://www.nginx.com/learn/api-gateway/>

YeePLY. (2022, 15 novembre). Qu'est-ce que le test unitaire ? comment s'y prendre ? YeePLY. <https://www.yeeply.com/fr/blog/test-unitaire-comment-sy-prendre/>

Zerial, A. (2024, 11 janvier). Les avantages de l'architecture modulaire informatique. Organisation Performante. <https://www.organisation-performante.com/les-avantages-de-larchitecture-modulaire-informatique/>

Мир IT с Антоном Павленко. (2023, 10 janvier). Что такое SLI, SLO, SLA КАК СЧИТАТЬ, что такое ? [Vidéo]. YouTube. <https://www.youtube.com/watch?v=14YSD5b0jHE>

Программист, Т. (2018, 7 décembre). Сначала – монолит, или правильный путь к микросервисной архитектуре. Tproger. <https://tproger.ru/translations/monolithfirst>