

Cycle de formation des ingénieurs en Télécommunications

*Option :*

Network Broadband Access (NBA)

## **Rapport de Projet de fin d'études**

*Thème :*

### **Mise en place d'une usine logicielle automatisée & Migration vers une architecture microservices sans serveur**

*Réalisé par :*

**Wael BEN SALAH**

*Encadrants :*

M. Nabil TABBANE: Professeur à Sup'Com

M. Rachid BERKANE: Ingénieur chez NTI Solutions

Mme. Fanta CISSE : Ressources Manager chez NTI Solutions

*Travail proposé et réalisé en collaboration avec :*

**NTI Solutions**



Année universitaire : 2017/2018

# ***Dédicaces***

*Je dédie ce travail*

*à mes chers parents*

# **Remerciements**

*Je ne saurais commencer la rédaction de ce rapport sans adresser mes sincères remerciements à l'administration de la société **NTI Solutions** pour l'occasion qu'elle m'a offerte pour passer mon stage dans son entreprise.*

*Mes vifs remerciements s'adressent à toutes les personnes de **NTI Solutions** et particulièrement Monsieur **Rachid Berkane** qui a toujours trouvé le temps de faire le suivi de mon travail et d'être à l'écoute et Madame **Fanta Cissé** pour ses conseils, ses orientations et ses efforts pour me faire passer les informations nécessaires.*

*Mes remerciements s'adressent également à Monsieur **Nabil Tabbane**, mon encadrant de **Sup'Com**. Ses directives et son sens du détail m'ont permis de soigner et d'améliorer constamment la qualité de ce travail.*

*Mon dernier mot s'adresse à tous les membres du jury pour l'honneur qu'ils me font de participer à l'examen de ce travail. J'espère que le présent projet soit à la hauteur de vos attentes.*

# ***Résumé***

*Le présent rapport synthétise le travail effectué dans le cadre du projet de fin d'études en vue de l'obtention du diplôme national d'ingénieur en Télécommunications de Sup'Com. Il est réalisé au sein de l'entreprise NTI Solutions IDF en France.*

*L'objectif principal de ce stage est de mettre en place une usine logicielle d'une manière automatisée, qui met en agilité le processus de production logicielle, et de migrer vers une architecture microservices sans serveur dans le cloud pour réduire les coûts de développement et améliorer la productivité de l'entreprise.*

**Mots clés :** *DevOps, Agile, Cloud computing, AWS, Terraform, Ansible, Git, Jenkins, SonarQube, CI/CD, Elastic Stack, Microservices, Serverless.*

# ***Table des matières***

<b>Introduction générale.....</b>	<b>1</b>
<b>Chapitre I : Contexte général &amp; Etude préalable.....</b>	<b>3</b>
I.1 Introduction.....	4
I.2 Contexte général .....	4
I.2.1 Cadre du projet.....	4
I.2.2 Présentation de l'organisme d'accueil .....	4
I.2.2.1 Le groupe NTI Solutions.....	4
I.2.2.2 NTI Solutions IDF .....	6
I.3 Problématique du projet.....	6
I.4 Diagnostic technique.....	7
I.4.1 L'approche DevOps et le processus de production logicielle.....	7
I.4.1.1 Git .....	8
I.4.1.2 Jenkins .....	8
I.4.1.3 SonarQube .....	9
I.4.1.4 Cucumber.....	9
I.4.1.5 JMeter.....	9
I.4.1.6 Slack .....	10
I.4.2 L'approche DevOps et le cloud computing .....	10
I.4.3 L'approche DevOps et les microservices .....	11
I.4.4 L'approche DevOps et l'architecture « Serverless » .....	11
I.4.5 Migration vers les applications sans serveur .....	13
I.5 Conclusion .....	13
<b>Chapitre II : Architecture &amp; Conception de la solution.....</b>	<b>14</b>
II.1 Introduction.....	15
II.2 Architecture de l'infrastructure dans le cloud AWS.....	15
II.2.1 Présentation .....	15
II.2.2 Routage.....	16
II.2.3 Sécurité.....	16

II.3 Conception de l'environnement de développement.....	17
II.3.1 Conception de l'usine logicielle .....	17
II.3.2 Conception d'une solution de monitoring centralisée.....	19
II.4 Conception d'une application sans serveur .....	20
II.4.1 Présentation .....	20
II.4.2 Architecture globale et microservices AWS utilisés .....	20
II.4.3 Cahier des charges de l'application « BillYourself » .....	23
II.4.3.1 Problématique .....	23
II.4.3.2 Solution proposée.....	23
II.4.3.3 Description des besoins fonctionnels.....	23
II.4.3.4 Description des besoins non-fonctionnels .....	24
II.4.4 Choix technologique .....	24
II.4.5 Conception de l'application .....	25
II.4.5.1 Diagramme de classes .....	25
II.4.5.2 Diagramme de cas d'utilisation .....	26
II.4.5.3 Diagramme de séquences .....	28
II.5 Conclusion .....	29
<b>Chapitre III : Réalisation du projet.....</b>	<b>30</b>
III.1 Introduction .....	31
III.2 Mise en place automatisée de l'infrastructure .....	31
III.2.1 Présentation .....	31
III.2.2 Création automatisée de l'infrastructure à l'aide de Terraform.....	32
III.3 Mise en place automatisée, configuration et sécurisation de l'usine logicielle .....	38
III.3.1 Présentation .....	38
III.3.2 Mise en place d'une usine logicielle automatisée avec Ansible .....	38
III.3.3 Configuration de l'usine logicielle .....	39
III.3.3.1 Configuration de AWS CLI.....	40
III.3.3.2 Configuration de NodeJS .....	41
III.3.3.3 Intégration de GitHub avec Jenkins.....	41
III.3.3.4 Intégration de SonarQube avec Jenkins.....	42
III.3.3.5 Intégration de Slack avec Jenkins .....	42
III.3.4 Sécurisation de l'usine logicielle avec SSL .....	44

III.4 Mise en place de la solution de monitoring .....	45
III.4.1 Génération d'un certificat SSL .....	45
III.4.2 Configuration de Logstash .....	46
III.4.3 Configuration de FileBeat .....	47
III.5 Développement et déploiement de l'application .....	48
III.5.1 Développement et déploiement de la partie frontend.....	48
III.5.1.1 Développement .....	48
III.5.1.2 Déploiement .....	51
III.5.1.3 Création d'un enregistrement .....	52
III.5.2 Gestion des utilisateurs .....	53
III.5.2.1 Présentation .....	53
III.5.2.2 Création d'un groupe d'utilisateurs Amazon Cognito .....	53
III.5.3 Intégration d'un backend sans serveur .....	53
III.5.4 Déploiement d'une API RESTful .....	54
III.6 Automatisation des tests de développement .....	55
III.6.1 Tests fonctionnels.....	55
III.6.1.1 Création du projet .....	55
III.6.1.2 Ecriture des scénarios.....	56
III.6.1.3 Définition des étapes en JAVA.....	57
III.6.1.4 Exécution de tests et génération de rapports.....	58
III.6.2 Tests de performances .....	58
III.6.2.1 Configuration de JMeter.....	58
III.6.2.2 Exécution de tests et génération de courbes.....	59
III.7 Implémentation des workflows CI/CD .....	60
III.7.1 CI/CD de la partie frontend .....	61
III.7.2 CI/CD de la partie backend .....	63
III.8 Conclusion .....	65
<b>Conclusion générale.....</b>	<b>66</b>
<b>Webographie.....</b>	<b>67</b>
<b>Annexes.....</b>	<b>69</b>

# Liste des figures

Figure 1: Localisation du siège, des agences et des centres de support de NTI Solutions .....	5
Figure 2: Logo de NTI Solutions .....	6
Figure 3: Architecture de l'infrastructure dans le cloud AWS .....	15
Figure 4: Les règles entrantes du groupe de sécurité associé à l'instance EC2 .....	16
Figure 5: Les étapes de la chaîne de production logicielle .....	17
Figure 6: Enchaînement des composants de la pile Elastic Stack .....	20
Figure 7: Architecture microservices sans serveur.....	21
Figure 8: Courbe de performance Blaze vs React vs AngularJS vs Angular .....	25
Figure 9: Diagramme de classes .....	26
Figure 10: Diagramme de cas d'utilisation coté front office .....	26
Figure 11: Diagramme de cas d'utilisation coté back office .....	27
Figure 12: Diagramme de séquences : Scénario de génération d'une facture .....	28
Figure 13: Ansible et les machines distantes .....	38
Figure 14: Intégration des différents composants de la chaîne de développement .....	40
Figure 15: Configuration de NodeJS .....	41
Figure 16: Vérification de l'intégration de GitHub avec Jenkins .....	41
Figure 17: Configuration du serveur SonarQube dans Jenkins .....	42
Figure 18: Ajout du chemin de SonarRunner dans Jenkins .....	42
Figure 19: Ajout de l'application « Jenkins CI » à Slack .....	43
Figure 20: Configuration des paramètres de Slack dans Jenkins .....	43
Figure 21: Sécuriser le trafic entre serveur/client en utilisant HTTPS .....	44
Figure 22: Visualisation des logs à travers Kibana.....	48



Figure 23: L'interface « Sign Up » de l'application .....	49
Figure 24: L'interface « Sign In » de l'application.....	49
Figure 25: L'interface « Billing » de l'application .....	50
Figure 26: L'interface de la génération de la facture de l'application .....	50
Figure 27: Hébergement de site Web en utilisant AWS S3 .....	52
Figure 28: Le rapport des tests fonctionnels de Cucumber .....	58
Figure 29: Configuration JMeter.....	59
Figure 30: Requêtes/s et temps de réponses des tests de performances .....	59
Figure 31: Taux d'erreurs des tests de performances .....	60
Figure 32: Frontend : Gestion du code source .....	61
Figure 33: Frontend : Ce qui déclenche le build.....	61
Figure 34: Frontend : Environnement de build .....	61
Figure 35 : Frontend : Lancer une analyse avec SonarQube .....	62
Figure 36 : Frontend : Exécuter un script .....	62
Figure 37 : Frontend : Notifier via Slack .....	63
Figure 38 : Backend : Gestion de code source .....	63
Figure 39 : Backend : Ce qui déclenche le build .....	64
Figure 40 : Backend : Environnement de build .....	64
Figure 41 : Backend : Build du projet .....	64
Figure 42 : Backend : Déploiement AWS Lambda .....	65

# *Liste des abréviations*

- AWS : Amazon Web Services
- IAM : Identity Access Manager
- VPC : Virtual Private Cloud
- ACL : Access Control List
- EC2 : Elastic Compute Cloud
- AMI : Amazon Machine Images
- IP : Internet Protocol
- CIDR : Classless Inter-Domain Routing
- SSH : Secure Shell
- SSL : Secure Sockets Layer
- CLI : Command Line interface
- IaC : Infrastructure as Code
- ELK : Elasticsearch Logstash Kibana
- S3 : Simple Storage Service
- API : Application Programming Interface
- YAML : Yet Another Markup Language
- NoSQL : No Structured Query Language
- HTTP : Hypertext Transfer Protocol
- HTTPS : Hypertext Transfer Protocol Secure
- HTML : HyperText Markup Language
- CSS : Cascading Style Sheets

- JSON : JavaScript Object Notation
- NPM : Node Package Manager
- REST : REpresentational state transfer
- DNS : Domain Name System
- URL : Uniform Resource Locator
- SaaS : Software as a Service
- CPU : Central Processing Unit
- CVS : Concurrent Versions System
- BDD : Behavior Driven Development
- IDE : Integrated Development Environment
- CI/CD : Continuous Integration / Continuous Delivery

# ***Introduction générale***

Le défi de toute entreprise, opérant dans le secteur de développement logiciel, est d'optimiser sa chaîne de production des solutions informatiques afin de gagner du temps et optimiser les coûts. Aujourd'hui, vu la concurrence du marché, il ne suffit plus désormais de produire des applications qui répondent à certains besoins, il faut que ces solutions logicielles soient concurrentielles à la fois avec leur haute qualité et leur prix bas. Ceci nécessite la réduction du cycle de livraison du produit final tout en améliorant sa qualité. En effet, les applications produites doivent être évolutives, disponibles, hyperperformantes avec une latence plus faible et, bien évidemment, à moindre coût.

Le mouvement DevOps apporte des réponses à cette problématique par la mise en application de différents concepts d'ordre culturel et technologique en mettant en place une infrastructure de développement agile, automatisée et sécurisée dans le but d'optimiser et accélérer la chaîne de mise en production. Ceci, permet aux équipes de développement et d'infrastructure d'être plus réactives face à ces nouvelles exigences afin de concrétiser cette évolution.

Concrètement, cela suppose des évolutions centrées sur l'innovation, l'adaptation et l'agilité, le cloud et l'automatisation. Cela suppose aussi de bons choix technologiques et logiciels en étudiant les avantages et les limites de chacun, en particulier, le choix architectural des applications développées afin de rendre les solutions plus fiables et plus robustes tout en répondant aux besoins évolutifs des clients.

C'est dans ce cadre que se situe mon stage de fin d'études. En effet, le présent rapport détaillera toute une solution pour améliorer la productivité et construire un milieu de travail numérique professionnel. Il s'articule autour de trois chapitres qui reflètent la démarche que nous avons adoptée pour ce projet.

Le premier chapitre est consacré au contexte général du stage, c'est-à-dire le cadre du projet et la présentation de l'établissement d'accueil, et à l'étude préalable du projet, c'est-à-dire ce qui est déjà existant, sa critique et sur la solution proposée.

Le deuxième chapitre présente les concepts clés, liés au style architectural du projet. C'est-à-dire la conception de l'infrastructure et de l'environnement de développement, l'architecture microservices adoptée ainsi que la conception.

Le troisième chapitre détaille les différentes étapes de la réalisation du projet (migration vers le cloud, automatisation, configuration, supervision, développement, tests, déploiement, etc.).

---

# ***Chapitre I : Contexte général & Etude préalable***

---

## **I.1 Introduction**

Dans ce premier chapitre introductif, nous présentons le contexte général du stage, à savoir, le cadre du projet et l'organisme d'accueil. Nous présentons par la suite la problématique traitée et un diagnostic technique pour la solution proposée.

## **I.2 Contexte général**

### **I.2.1 Cadre du projet**

Le présent travail s'inscrit dans le cadre du projet de fin d'étude en vue de l'obtention du diplôme national d'ingénieur en Télécommunications de Sup'Com. Le projet s'intitule « Mise en place d'une usine logicielle automatisée & Migration vers une architecture microservices sans serveur ». Il est réalisé au sein de la société NTI Solutions IDF en France.

### **I.2.2 Présentation de l'organisme d'accueil**

#### **I.2.2.1 Le groupe NTI Solutions**

##### **I.2.2.1.1 Présentation**

Depuis vingt ans, NTI Solutions est une Société d'Expertise et de Services en Informatique, plus spécifiquement dans le domaine des Systèmes, Réseaux et Télécommunications.

NTI Solutions, membre du groupe « Ice Consulting », propose une offre complète, du conseil en architecture jusqu'au maintien en condition opérationnelle des systèmes informatiques.

La société représente un chiffre d'affaire annuel d'environ douze millions d'euro et d'une soixantaine de collaborateurs en production.

NTI Solutions est installée à Beauvais (Oise), mais grâce à ces plusieurs agences et centres de supports, elle intervient sur tout le territoire français. La figure ci-dessous montre les localisations du siège social, les agences et les différents centres de supports de NTI Solutions.

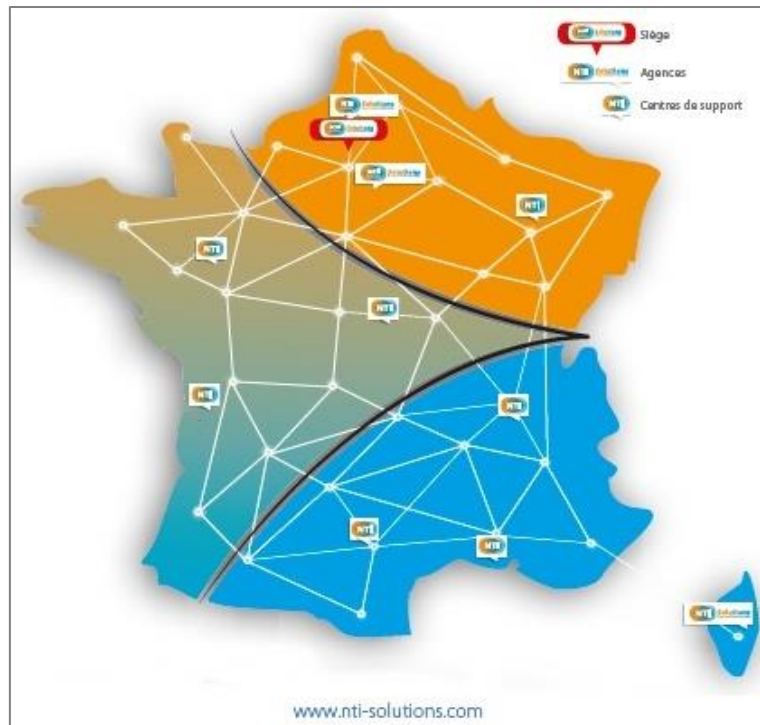


Figure 1: Localisation du siège, des agences et des centres de support de NTI Solutions

#### I.2.2.2.1 Domaines d'expertises

- **Infrastructure informatique**
  - ✓ Câblage : tisser les mailles d'un réseau sécurisé, rapide et performant.
  - ✓ Réseau d'entreprise : audit, intégration LAN/WAN/WIFI, boucle locale radio, optimisation des performances, interconnexion de sites, haute disponibilité, etc.
  - ✓ Sécurité informatique : assurer la parfaite sécurité des systèmes, des données et des échanges (sauvegarde, archivage, pare-feu, filtrage, etc.).
- **Solutions de sécurité physique**
  - ✓ Vidéo protection/Vidéosurveillance.
  - ✓ Contrôle d'accès.
- **Communications unifiées**
  - ✓ Solution de téléphonie IP : solution cloud, gestion multisite, haute disponibilité, etc.
  - ✓ Mobilité : messagerie unifiée, téléphonie mobile WIFI/DECT, Numéro unique, etc.
  - ✓ Travail collaboratif : réunion en ligne, vidéoconférence, gestion de présence, messagerie instantanée, etc.
  - ✓ Centre d'appels : routage dynamique, services vocaux intégrés, supervision, couplage téléphonie informatique etc.



- **Services managés**

- ✓ Services de support.
- ✓ Services de supervision.
- ✓ Gestion des performances réseau.

#### **I.2.2.3.1 Secteurs d'activités**

- ✓ Secteur public : mairies, collectivités locales, conseils départementaux, communautés d'Agglomérations, centres de gestion, etc.
- ✓ PME/PMI : industrie, transport, banques, assurances, parcs de loisirs, etc.
- ✓ Santé : hôpitaux, cliniques, laboratoires, maisons de retraite, etc.

#### **I.2.2.2 NTI Solutions IDF**

NTI Solutions IDF est la filiale du groupe NTI Solution à l'Île-De-France. C'est une société à responsabilité limitée est en activité depuis trois ans. Située au 22 Rue Gustave Eiffel à Poissy (78300) en France, elle est spécialisée dans le secteur d'activité de la programmation informatique. Sur l'année 2016, elle a réalisé un chiffre d'affaires de 2 680 300 €.

NTI Solutions IDF accompagne ses clients dans leur transformation digitale dans différents secteurs. Parmi ses clients, on cite « JCDecaux » le groupe industriel français spécialisé dans la publicité urbaine, la « Banque de France » et le leader de l'audit et du conseil « KPMG ».



*Figure 2: Logo de NTI Solutions*

### **I.3 Problématique du projet**

L'infrastructure de développement informatique est souvent coûteuse et difficile à mettre en œuvre, ce qui nécessite toute une équipe qui la met en place, l'administre et assure sa sécurité et son bon fonctionnement. Le choix des outils de production de code, la

spécification et le déploiement de cette infrastructure nécessitent à la fois des compétences de développeurs et des compétences d'administrateurs système et réseaux.

Dans ce contexte, un problème, qui s'impose et présente un défi pour les boîtes de développement, depuis toujours, concerne l'agilité au niveau d'équipe qui est constituée traditionnellement de deux sous-équipes : une équipe de développement (Dev) et une équipe opérationnelle (Ops). L'équipe de développement écrit le code des applications, l'exécute et le teste dans un environnement isolé sans se préoccuper par l'impact de leur code sur la production ni par les erreurs au niveau des serveurs. L'équipe opérationnelle est plutôt occupée par cela et par la stabilisation des services et par l'architecture ainsi que la performance des instances mises en place. Cette organisation traditionnelle entre les équipes n'est pas trop efficace et alourdit énormément la production. En effet, plus le projet est énorme, plus le temps pour la mise en production augmente. Potentiellement, cela rend le travail de l'équipe plus gênant et plus coûteux en termes d'homme/jours ce qui implique l'augmentation du stress pour l'équipe et bien évidemment de l'argent perdu pour l'entreprise. De plus, de point de vue qualité de service, cela peut augmenter le temps d'indisponibilité pour l'application ce qui peut augmenter le risque d'insatisfaction des utilisateurs finaux.

Le mouvement DevOps aborde cette problématique sous l'angle des relations entre l'administration des systèmes et des réseaux et le développement pour faciliter la mise en place d'une infrastructure agile, automatisée et sécurisée adaptée aux besoins et dans le but d'accélérer et d'optimiser toute la chaîne de mise en production, depuis la phase de développement jusqu'au déploiement en production [14].

Dans le but de surmonter ces limites, nous avons décidé de passer vers la culture DevOps et vers l'architecture microservices dans le cloud. Ainsi, deux sous-projets sont affleurés, un sous-projet d'automatisation de l'infrastructure et de chaîne de déploiement et un autre sous-projet de migration vers une architecture microservices et sans serveur.

## **I.4 Diagnostic technique**

### **I.4.1 L'approche DevOps et le processus de production logicielle**

L'approche DevOps a optimisé le processus de développement et l'a transformé en chaînes de production logicielle automatisées. Les activités manuelles répétitives des

analystes, développeurs, testeurs et opérateurs sont converties, autant que possible, en activités automatiques en utilisant certains outils de développement logiciel.

Ceci se fait à travers des solutions d'automatisation et de virtualisation de l'infrastructure dans le cloud et d'une usine logicielle configurable et automatisée qui répond aux besoins des développeurs et architectes logiciels de mener à bien leur mission d'une manière agile et itérative. Ainsi, l'automatisation permet d'augmenter la productivité des équipes, de réduire le nombre d'erreurs, d'améliorer la collaboration et de consacrer plus de temps à des tâches plus importantes.

En outre, l'usine logicielle permet aux développeurs d'adopter de bonnes pratiques (gestion de versions, intégration continue, tests de performances, couverture de code, automatisation du déploiement, etc.) dans le but de mettre en valeur les applications développées. Une bonne usine logicielle devrait inclure un gestionnaire de versions de code centralisé comme Git, un logiciel d'intégration continue comme Jenkins et un logiciel permettant de mesurer la qualité du code source en continu comme SonarQube.

#### **I.4.1.1 Git**

Git [2] est un gestionnaire de code source. Il permet de tracer l'historique de toutes les modifications de code effectuées par les différents développeurs. De plus, il est possible de publier le code gratuitement sur GitHub par exemple (<http://www.github.com>) de manière publique.



#### **I.4.1.2 Jenkins**

Jenkins [3] est un outil open source, écrit en JAVA, d'intégration continue extrêmement simple à mettre en œuvre. Jenkins fonctionne dans un conteneur de servlets tel qu'Apache Tomcat et s'interface avec des systèmes de gestion de versions tels que Git, CVS et Subversion et avec d'autres outils comme SonarQube, Slack et Maven grâce à son système de configuration en plugins.



Jenkins permet de sécuriser le développement en garantissant la présence d'une version intermédiaire du projet sans se lancer dans un processus manuel coûteux.

#### **I.4.1.3 SonarQube**

SonarQube [4] est un logiciel libre développé par la société « Sonarsource » permettant de mesurer la qualité de code des projets. Il supporte plus de vingt-cinq langages (Java, C, C++, C#, PHP, JavaScript, Python, etc.).



SonarQube permet de détecter des erreurs classiques effectuées lors du développement (fuites mémoire possibles, etc.) et d'obtenir des métriques représentatives (complexité du code, taux de commentaire, duplication de code, etc.). Cet outil peut se brancher sur un outil d'intégration continue comme Jenkins et remonter les erreurs lors de la construction du produit.

En plus de l'usine logicielle, la chaîne de production logicielle pourrait intégrer d'autres outils qui peuvent ajouter d'autres fonctionnalités utiles dans le processus de développement tels que les outils de tests, les plateformes de collaboration et de communication etc.

#### **I.4.1.4 Cucumber**

Cucumber [10] est un outil qui permet d'automatiser les tests fonctionnels de développement. Cet outil exécute des spécifications



exécutables écrites en langage clair, avec un vocabulaire précis (Given, When, Then, etc.) et produit des rapports indiquant si le logiciel se comporte ou non selon les spécifications.

#### **I.4.1.5 JMeter**

JMeter [11] est un projet de logiciel libre permettant d'effectuer des tests de performances d'applications et de serveurs selon différents protocoles. Il est développé au sein de la fondation Apache.



JMeter permet de simuler le comportement de plusieurs utilisateurs agissant de manière simultanée sur une application Web. Il mesure le temps de réponse de chaque requête et produit des statistiques de ces temps de réponse.

#### **I.4.1.6 Slack**

Slack [12] est une plateforme collaborative qui centralise tous les flux de communication des équipes DevOps pour optimiser la productivité. En effet, Slack peut regrouper les membres d'une équipe et ses conversations dans un seul espace de travail. Les discussions sont organisées dans des chaînes. Ainsi, chaque projet, chaque équipe ou chaque département trouve sa place. Il est possible de créer une chaîne pour les alertes de service, une autre pour l'intégration de nouveaux utilisateurs, etc. De plus, Il est possible d'intégrer Slack avec les autres outils comme Jenkins.



#### **I.4.2 L'approche DevOps et le cloud computing**

Le cloud permet une plus grande agilité du business en rendant les infrastructures informatiques plus flexibles. Il rend possible la création d'un lien numérique entre les entreprises et leurs clients. Le cloud n'est qu'une partie de la réponse à la question du business rendu adaptatif par l'informatique. Qu'une application tourne dans le data center de l'entreprise ou sur un cloud, qu'il soit privé ou public, elle doit toujours être en phase avec les besoins business et non pas l'inverse.

Une étude récente [15] affirme qu'en adoptant une stratégie combinant cloud computing et DevOps, les entreprises peuvent doubler leur vitesse de fourniture de logiciels et obtenir une amélioration de 81 % des performances globales de fourniture de logiciels.

L'étude montre que le duo DevOps et cloud computing offre des avantages à tous les niveaux, mais plus particulièrement en termes de rapidité et de coût. De plus, les entreprises qui utilisent ce duo ont obtenu une amélioration de 69 % de l'expérience des clients ainsi une meilleure prévisibilité de 80 % sur les performances logicielles, par rapport à la référence des modèles traditionnels de développement et fourniture de logiciels.

### **I.4.3 L'approche DevOps et les microservices**

Actuellement, les microservices et DevOps sont deux tendances en effervescence. Elles présentent des pratiques qui offrent une plus grande agilité et une efficacité opérationnelle pour l'entreprise, selon Uri Sarid, le directeur de la technologie de « MuleSoft » : « L'excellence de DevOps est un élément clé de l'excellence des microservices ». C'est-à-dire des bons résultats auront lieu lorsque DevOps et les microservices sont appliqués ensemble.

Le mariage entre l'architecture des microservices et l'approche DevOps a vu le jour dans des entreprises comme Amazon, Netflix, SoundCloud, Facebook, Google, etc. Dans de nombreux cas, ces entreprises ont commencé par des applications monolithiques, qui ont rapidement évolué vers des services décomposés. Avec cette évolution architecturale, ces entreprises ont également excellé avec DevOps, elles ont partagé des approches communes pour le développement de logiciels. En outre, les microservices apportent une productivité supplémentaire à DevOps en adoptant un ensemble d'outils communs, qui peut être utilisé à la fois pour le développement (Dev) et les opérations (Ops). Ceci a facilité les tâches en permettant aux membres des équipes Dev et Ops de travailler ensemble sur un problème et de le résoudre avec succès [16].

### **I.4.4 L'approche DevOps et l'architecture « Serverless »**

Même si le DevOps et la conteneurisation ont apporté beaucoup dans la gestion et l'automatisation de l'infrastructure, ils ne l'ont pas résolue pour autant. Aujourd'hui, une nouvelle architecture fait parler d'elle dans le monde de la technologie, c'est le « Serverless ». Alors qu'est-ce qu'une architecture sans serveur ?

Une architecture sans serveur est un moyen de créer et exécuter des applications et services sans avoir à gérer d'infrastructure. En effet, les applications s'exécutent toujours sur des serveurs, mais la gestion des serveurs est effectuée par un fournisseur de services cloud (Amazon, Google, Microsoft, etc.). C'est à ces fournisseurs de mettre en service, de mettre à l'échelle et d'entretenir les serveurs pour exécuter les applications, les bases de données et les systèmes de stockage [1].

Le « Serverless » permet de payer quelqu'un pour gérer les serveurs, les bases de données et même la logique des applications. Ainsi, ce paiement suit le principe « Pay as you go », c'est-à-dire payer uniquement les frais d'utilisations et de consommation (bande passante, temps de calcul, volume de stockage, etc.), ce qui permet d'avoir une facturation très fine et de ne payer que la charge dont nous avons besoin.

De plus, étant donné que le service consommé fait partie d'un ensemble de services similaires, la notion d'économie d'échelle va alors s'appliquer : On paye moins cher les coûts de gestion vu que le même service est utilisé par de nombreuses personnes, ce qui permet de réduire les coûts.

Un bon exemple qui montre l'économie de l'argent est l'utilisation occasionnelle d'une fonction. Par exemple, si nous exécutons une application serveur qui ne traite qu'une seule demande chaque minute et disons qu'il faut 50 ms pour traiter chaque requête et que notre utilisation moyenne de CPU pendant une heure est de 0,1%. D'un point de vue charge de travail serveur, cela est extrêmement inefficace. La technologie « Serverless » capte cette inefficacité et permet ainsi de ne payer que ce qui est consommé, c'est-à-dire 100 ms de calcul par minute, soit 0.15% du temps global [17].

En outre, grâce à l'architecture « Serverless », les développeurs peuvent se concentrer sur leur code au lieu de se soucier de la gestion et de l'exploitation des serveurs, que ce soit dans le cloud ou sur site. Cette réduction des opérations nécessaires leur permet de consacrer davantage de temps et d'énergie au développement de produits fiables, évolutifs et formidables ce qui réduit les coûts en termes d'homme/jours et augmente la qualité du produit développé.

Afin d'illustrer ce point, prenons en exemple la gestion manuelle des bases de données qui peut prendre un temps précieux et peut conduire à l'utilisation inefficace des ressources de cette dernière, alors que les bases de données sans serveur s'exécutent dans le cloud sans gérer d'instances ni de clusters de bases de données. Ce système permet une gestion plus fine des données accessibles en fonction des différents profils utilisateur.

Un autre exemple qui se prête bien au jeu est le cas de l'authentification. De nombreuses applications codent leurs propres services d'authentification et de gestion des utilisateurs, implémentant par la même occasion leur propre niveau de sécurité alors qu'avec du « Serverless », la logique d'authentification est remplacée par un service (Auth0, AWS Cognito, etc.) dans le cloud assurant l'enregistrement, l'authentification et l'accès aux différents services d'une manière sécurisée.

#### **I.4.5 Migration vers les applications sans serveur**

Avec tous les avantages que porte le « Serverless » cités ci-dessus, nous allons migrer vers une architecture sans serveur dans le cloud AWS pour le développement des applications. Par conséquent, un certain nombre de changements importants aura lieu au niveau de l'architecture des applications.

Premièrement, nous n'aurons plus besoin d'héberger nos fichiers Web sur un serveur, un simple compartiment (un bucket S3) et un outil de gestion du cache peuvent suffire. Deuxièmement, la rebrique d'authentification sera assurée par le service AWS Cognito au lieu de la coder à la main. Troisièmement, au lieu d'avoir un serveur toujours en cours d'exécution, il y aura un nouveau panel d'outils à notre disposition qui répondra aux requêtes HTTP via une API Gateway. Cette API permet de définir des points finaux sécurisés accessibles de l'extérieur et de rediriger l'appel vers un service de traitement (une fonction AWS Lambda) qui retournera des données selon les besoins à partir d'une base de données sans serveur (DynamoDB, Aurora, etc.).

#### **I.5 Conclusion**

À travers ce premier chapitre, nous avons exposé le cadre de notre travail, la problématique et un diagnostic technique de la solution proposée. Nous pouvons passer au chapitre suivant qui est réservé à l'architecture et la conception de la solution.



---

## ***Chapitre II : Architecture & Conception de la solution***

---

## II.1 Introduction

Dans ce chapitre, nous allons nous intéresser à l'architecture et la conception de la solution : Nous allons faire une conception architecturale de l'infrastructure et de l'environnement du développement. Puis, nous allons faire la conception de l'application à développer.

## II.2 Architecture de l'infrastructure dans le cloud AWS

### II.2.1 Présentation

Vu les avantages, bien détaillés dans [I.4.2], que le cloud apporte pour une entreprise, nous allons opter pour une architecture basée sur un cloud virtuel privé (VPC) sur la région Ohio (us-east-2) d'AWS. Ce VPC utilise le bloc d'adresse CIDR IPv4 10.0.0.0/16. La figure ci-dessous présente l'architecture globale de notre environnement cloud AWS.

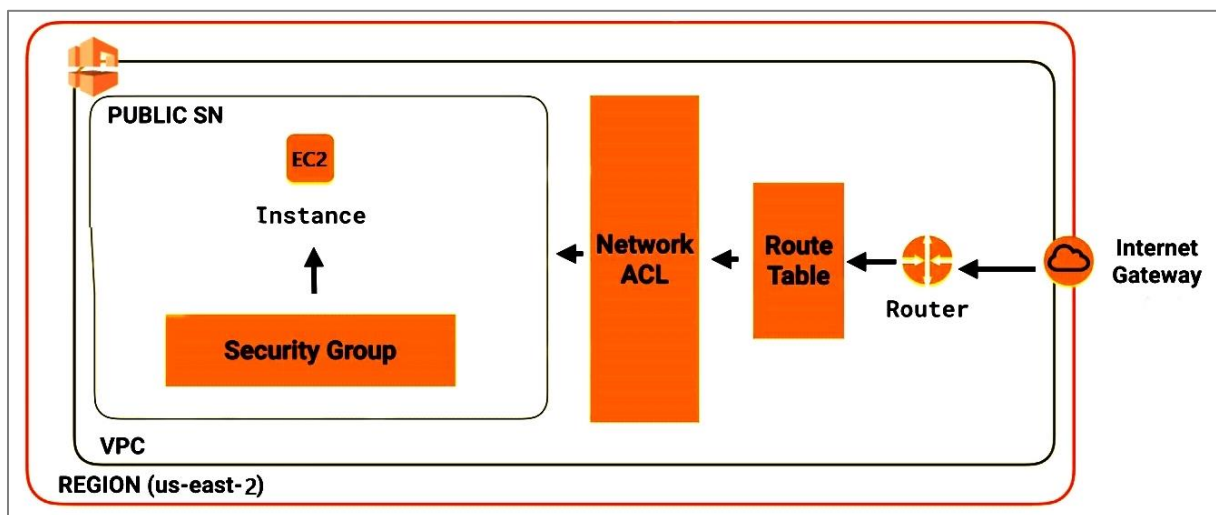


Figure 3: Architecture de l'infrastructure dans le cloud AWS

Le VPC permet de mettre en service une section du cloud AWS qui a été isolée d'une manière logique et dans laquelle nous pouvons lancer des ressources AWS dans un réseau virtuel.

La configuration de ce scénario consiste à créer un sous-réseau public avec bloc d'adresse CIDR IPv4 10.0.0.0/24 et une passerelle Internet (Internet Gateway) pour permettre la connexion à Internet et la communication aux autres services AWS.

L'usine logicielle va être installée sur une instance EC2 que nous allons créer dans notre sous-réseau. Une adresse IPv4 dans la plage du sous-réseau va être allouée à cette instance.

## II.2.2 Routage

Le VPC intègre un routeur implicite. En effet, il crée une table de routage qui achemine tout le trafic destiné à une adresse se trouvant à l'extérieur du cloud privé vers l'Internet Gateway et associe cette table de routage au sous-réseau public.

Le tableau suivant présente la table de routage correspondante au schéma de configuration ci-dessus. La première entrée est dédiée pour le routage IPv4 local dans le VPC, elle permet aux instances du VPC de communiquer entre elles alors que la deuxième entrée achemine tout le reste du trafic du sous-réseau IPv4 vers la passerelle Internet.

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	<i>igw-id</i>

## II.2.3 Sécurité

AWS propose deux techniques à utiliser pour renforcer la sécurité d'un VPC qui sont les groupes de sécurité et les listes ACL réseau.

Les groupes de sécurité contrôlent le trafic entrant et sortant pour les instances créées dans un sous-réseau, et les ACLs réseau contrôlent le trafic entrant et sortant pour les sous-réseaux. Dans notre cas, où nous avons une instance EC2 dans un sous réseau, les groupes de sécurité sont assez suffisants pour répondre à notre besoin.

Le tableau ci-dessous décrit les règles entrantes pour le trafic IPv4 du groupe de sécurité qui contrôle le trafic entrant pour notre instance EC2 sur laquelle l'usine logicielle sera installée.

Type ⓘ	Protocole ⓘ	Plage de ports ⓘ	Source ⓘ	Description ⓘ
HTTP	TCP	80	0.0.0.0/0	Web Access
Règle TCP personnalisée	TCP	9000	0.0.0.0/0	SonarQube
Règle TCP personnalisée	TCP	8080	0.0.0.0/0	Jenkins
SSH	TCP	22	193.52.24.27/32	SSH
Règle TCP personnalisée	TCP	5044	0.0.0.0/0	FileBeat
HTTPS	TCP	443	0.0.0.0/0	Secure Web Access

Figure 4: Les règles entrantes du groupe de sécurité associé à l'instance EC2

Nous allons autoriser le port 22 pour le trafic entrant afin de pouvoir accéder à distance à l'instance EC2 en utilisant le protocole de communication sécurisé SSH, les ports 80 et 443 pour l'accès Web, les ports 8080 et 9000 sur lesquels les services Jenkins et SonarQube de notre usine logicielle se mettront en écoute et le port 5044 pour l'agent FileBeat.

Il est possible également d'utiliser les ACLs réseau si nous souhaitons ajouter une couche de sécurité supplémentaire à notre VPC.

## II.3 Conception de l'environnement de développement

### II.3.1 Conception de l'usine logicielle

Dans cette partie, nous allons concevoir notre chaîne de développement basée sur une usine logicielle contenant les différents outils décrits dans la partie [I.4.1].

L'usine logicielle permet d'orchestrer de bout en bout et en continu tous les outils de la chaîne logicielle et d'automatiser le processus de production logicielle du développement jusqu'au déploiement. La figure ci-dessous montre la chaîne de production que nous allons opter pour dans notre projet.

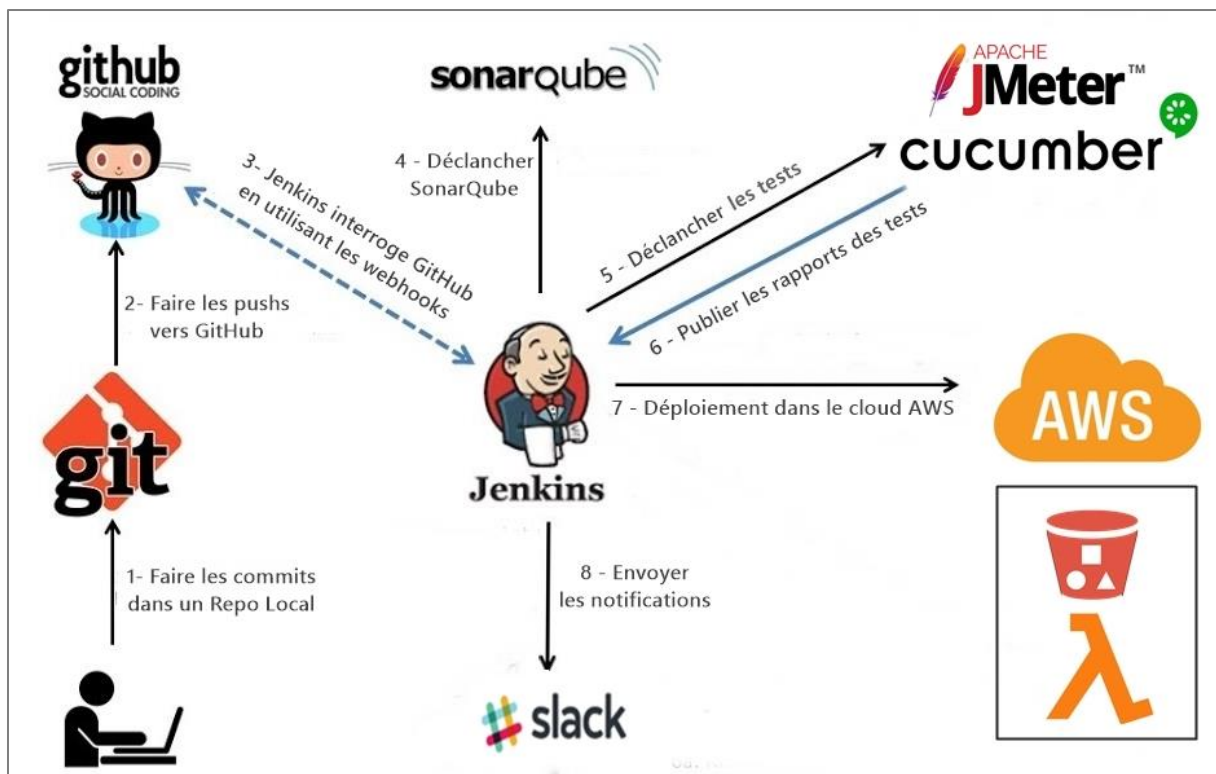


Figure 5: Les étapes de la chaîne de production logicielle

Au cours de ce projet, le processus de production logicielle va passer par les étapes suivantes :

Premièrement, après l'écriture du code en utilisant un IDE, le développeur fait ses commits dans un repo local pour sauvegarder les modifications qu'il a faites.

Deuxièmement, pour centraliser la version du code, le développeur doit faire un push pour mettre à jour son code sur GitHub.

Troisièmement, le commit doit déclencher un build automatique du projet sur Jenkins grâce à un « webhook » qui devra être configuré entre GitHub et Jenkins.

Quatrièmement, Jenkins doit déclencher SonarQube pour mesurer la qualité du code grâce à un plugin qui devra être installé et configuré sur Jenkins.

Cinquièmement, Jenkins doit déclencher Cucumber et JMeter pour automatiser certains tests de développement que nous détaillerons par la suite.

Sixièmement, Cucumber et JMeter doivent publier les rapports de tests sur Jenkins grâce à deux plugins qui devront être installés.

Septièmement, si les tests réussissent, et si une certaine qualité de code est garantie par SonarQube, le code peut être déployé dans le cloud AWS. Dans ce projet, nous allons utiliser les services S3 et Lambda d'AWS pour déployer notre code.

Huitièmement, une fois le code est bien déployé, il est possible de notifier les membres de l'équipe en utilisant Slack grâce à un plugin qui devra être installé et configuré sur Jenkins.

Avec une telle chaîne de production, le processus de développement sera automatisé et optimisé. Cependant, notre usine logicielle pourrait rencontrer des incidents au niveau du système. Ainsi, les serveurs Jenkins et SonarQube pourront générer les logs pour des échecs de connexion ou de build par exemple. Pour cela, il vaut mieux centraliser les journaux d'événements (logs) de nos serveurs et de nos applications afin de les bien surveiller. À cet égard, nous allons étudier, dans ce qui suit, une solution de monitoring centralisée.

### II.3.2 Conception d'une solution de monitoring centralisée

La centralisation des logs peut être très utile lorsque nous voulons identifier les problèmes de nos différents serveurs et/ou applications, car elle permet de rechercher tous les logs au même endroit pendant une période donnée. À ce propos, nous avons mis en place la pile « Elastic Stack » [7] qui est en fait une évolution de la pile « ELK » qui se compose elle-même de :

- « Logstash » qui permet de traiter les logs, les filtrer, les découper, pour les transformer en documents compréhensibles pour Elasticsearch.
- « Elasticsearch » qui utilise les documents formatés par Logstash, il les stocke et les indexe. Il permet aussi d'analyser ces documents.
- « Kibana » qui est un tableau de bord paramétrable sous forme d'une interface Web qui permet de visualiser les données d'ElasticSearch.

Ce trio simplifie la vie de l'administrateur système en lui proposant rapidement une vue d'ensemble des serveurs et services dont il a la charge.

En addition de la pile ELK, la pile Elastic Stack intègre un module supplémentaire qui s'appelle « Beats ». Ce module permet d'installer un agent sur notre machine distante (l'usine logicielle). Cet agent s'occupera d'envoyer les données voulues (contenus des logs) vers notre serveur de monitoring. Il regroupe plusieurs outils différents :

- « PacketBeat » pour la supervision du réseau.
- « TopBeat » pour superviser les « tops », c'est-à-dire les processus ayant consommé le plus de mémoire vive, de CPU, etc.
- « FileBeat » qui est un moniteur temps-réel des fichiers.

Ces outils se marient parfaitement avec la pile ELK, et puisque nous allons nous intéresser aux fichiers logs générés par le système (Syslog), Jenkins et SonarQube, nous allons utiliser alors l'outil FileBeat. En effet, les données collectées par FileBeat seront traitées et

filtrées par Logstash, puis indexées et stockées dans Elasticsearch et finalement visualisées par Kibana comme indiqué dans la figure ci-dessous.

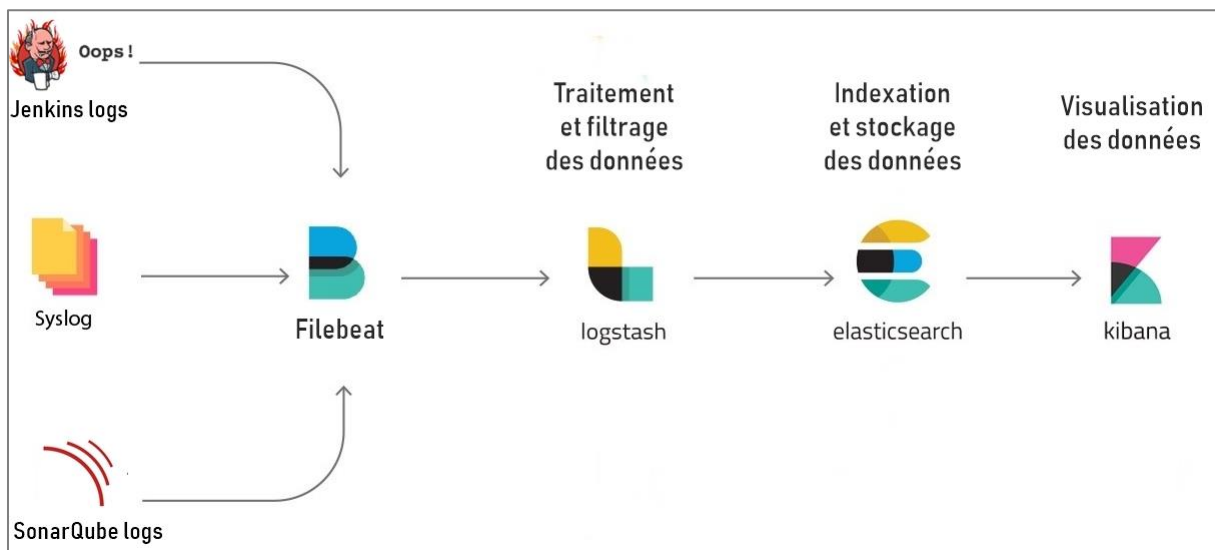


Figure 6: Enchaînement des composants de la pile Elastic Stack

Cette pile présente une solution d'analyse de logs performante et complète. À ce stade, l'architecture de notre environnement est bien claire. Passons maintenant à la conception de la partie de développement du projet.

## II.4 Conception d'une application sans serveur

### II.4.1 Présentation

Dans cette partie, nous allons migrer alors vers l'architecture « Serverless » décomposée en microservices, pour les raisons expliquées dans [I.4.3], [I.4.4] et [I.4.5], dans le but de développer une application sans serveur, entièrement hébergée dans le cloud AWS.

### II.4.2 Architecture globale et microservices AWS utilisés

L'architecture globale d'une application sans serveur utilise principalement les microservices AWS suivants : S3, Cognito, API Gateway, Lambda et DynamoDB, Route 53 et CloudWatch comme illustré ci-dessous.

Dans ce qui suit, nous allons opter pour une telle architecture pour le développement de notre application intitulée « BillyYourself ».

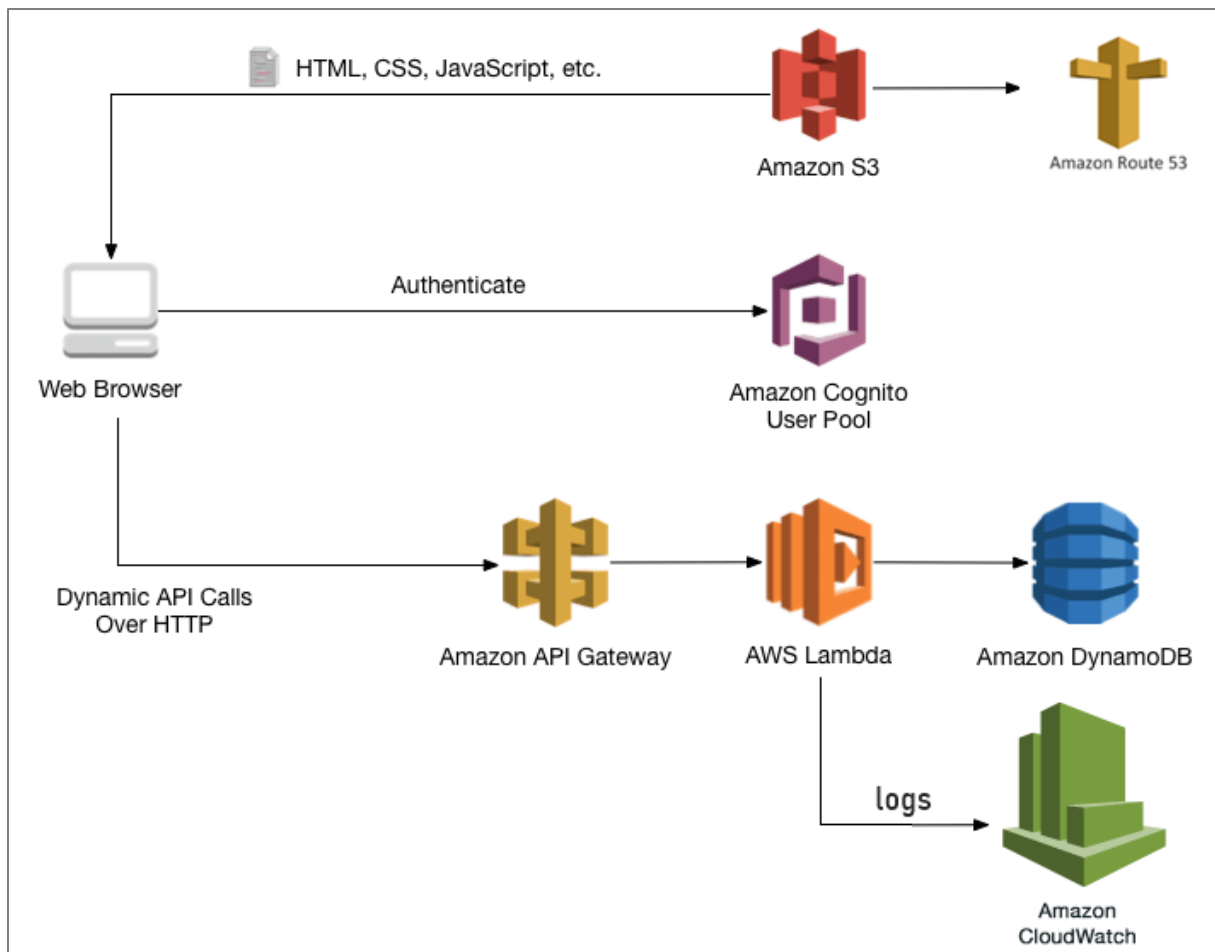


Figure 7: Architecture microservices sans serveur

- **Amazon S3**

Amazon S3 est une solution AWS de stockage sur Internet. Il est possible d'utiliser ce service pour stocker et récupérer n'importe quelle quantité de données, n'importe quand et depuis n'importe quel emplacement sur le Web [1].



Nous allons utiliser ce service pour héberger la partie frontale statique (HTML, CSS, JavaScript et TypeScript) de notre application.

- **Amazon Cognito**

Amazon Cognito est un service AWS qui permet d'ajouter facilement les fonctions de connexion et d'inscription des utilisateurs et de gérer les autorisations pour les applications Web et mobiles. Il permet de créer un répertoire utilisateur sécurisé dans Amazon [1].





Nous allons profiter de ce service pour la gestion des utilisateurs et afin de sécuriser l'authentification à notre application.

- **Amazon API Gateway**

Amazon API Gateway est un service AWS qui permet de créer, de publier, de gérer, de superviser et de sécuriser facilement les APIs [1].

Nous allons utiliser API Gateway pour créer les APIs RESTful nécessaires qui seront une « porte d'entrée » pour l'applications, afin d'accéder aux données.



API Gateway

- **Amazon Lambda**

Amazon Lambda est un service AWS qui permet d'exécuter du code pour n'importe quel type d'application ou de service backend sans avoir à allouer ou gérer des serveurs et sans aucune administration. Il suffit de charger le code et Lambda fait le nécessaire pour l'exécuter et en assurant une haute disponibilité [1].



Amazon Lambda

- **Amazon DynamoDB**

Amazon DynamoDB est un service AWS de base de données NoSQL qui prend en charge les modèles de stockage de documents et de clé-valeurs et qui offre des performances exceptionnelles et prévisibles en termes de rapidité et d'évolutivité [1].



DynamoDB

Nous allons profiter de la flexibilité et la performance de cette base de données pour le stockage des données de notre application.

- **Amazon Route 53**

Amazon Route 53 est un service AWS équivalent au service DNS hautement disponible et évolutif. Il est conçu pour donner un moyen fiable et rentable de traduction des noms par des adresses IP [1].



Amazon Route 53

Nous allons utiliser ce service pour créer un enregistrement de noms de domaine pour l'URL de notre application.

- **Amazon CloudWatch**

Amazon CloudWatch est un service AWS qui permet de surveiller les ressources du cloud et les applications exécutées sur AWS. En outre, il est possible d'utiliser CloudWatch pour collecter et suivre des métriques, regrouper et contrôler des fichiers journaux (logs), régler des alarmes et réagir automatiquement aux modifications apportées aux ressources AWS [1].



Nous allons utiliser ce service pour surveiller le fonctionnement des fonctions Lambda développées dans la partie backend de l'application.

### **II.4.3 Cahier des charges de l'application « BillYourself »**

#### **II.4.3.1 Problématique**

D'habitude, quand on est freelance, on a souvent du mal à organiser sa paperasse vu la quantité de travail et les différentes tâches à faire pour satisfaire tous ses clients et avoir plus de marchés et gagner plus d'argent. En outre, lorsqu'on est freelance, on a besoin de garder le focus sur le travail à délivrer et on évite de casser la tête avec de la comptabilité et de la facturation des clients, et parfois, on n'a pas les compétences suffisantes pour le faire.

#### **II.4.3.2 Solution proposée**

L'objectif de cette application est de fournir une plateforme en mode SaaS (hébergée dans le cloud) pour simplifier la gestion de facturation pour les freelances en envoyant les factures générées automatiquement à leurs clients. En effet, tenir un registre de facturation et de paiement est quelque chose de très important, et permet de savoir précieusement quelle facture a été émise, et si le client a payé. Négliger sa gestion c'est perdre beaucoup de temps, et de l'argent.

#### **II.4.3.3 Description des besoins fonctionnels**

Afin d'assurer le bon fonctionnement de notre application, nous avons dégagé les besoins fonctionnels. Nous avons des besoins fonctionnels côté front office et des besoins fonctionnels côté back office.

#### **II.4.3.3.1 Besoins fonctionnels « front office »**

- Générer les factures automatiquement : chaque utilisateur a le droit de générer ses factures en entrant les détails nécessaires.
- Télécharger les factures : chaque utilisateur a le droit de télécharger ses factures générées en format PDF.
- Consulter les factures : chaque utilisateur peut consulter la liste de toutes ses factures.
- Modifier et supprimer les factures : chaque utilisateur a le droit de modifier et supprimer ses anciennes factures.

#### **II.4.3.3.2 Besoins fonctionnels « backoffice »**

- Gérer les utilisateurs : l'administrateur a la possibilité de créer, modifier, désactiver et supprimer les utilisateurs.
- Gestion de la base de données : l'administrateur a le droit d'apporter n'importe quel type de modification sur la base de données.

#### **II.4.3.4 Description des besoins non-fonctionnels**

En plus des besoins fonctionnels, il faut cerner les besoins non-fonctionnels de notre application :

##### **II.4.3.4.1 Besoins non-fonctionnels « front office »**

- S'inscrire à l'application puis s'identifier : chaque utilisateur doit créer un compte et s'identifier pour pouvoir utiliser l'application.
- L'utilisateur doit être guidé lors de la saisie de certaines informations, afin de respecter les formats des champs de notre base de données.

##### **II.4.3.4.2 Besoins non-fonctionnels « back office »**

L'administrateur doit garantir la sécurité d'accès à l'espace administrateur afin de protéger les données personnelles des utilisateurs.

#### **II.4.4 Choix technologique**

Pour commencer la phase de développement, il faut tout d'abord choisir les technologies à utiliser. Pour la partie frontend, plusieurs technologies se présentent. De point de vue open source, notre choix va être restreint sur les deux frameworks Angular [8] et React [9], mais notre choix doit satisfaire certains critères. En effet, notre application doit être

performante et notre code doit être maintenable, modulaire et compréhensible par les autres développeurs.

La figure ci-dessous montre une analyse [18] du temps nécessaire pour afficher un certain nombre d'articles. L'analyse est faite pour comparer les quatre frameworks AngularJS, Angular 2+, ReactJS et Blaze, mais nous nous intéressons seulement aux résultats d'Angular 2+ et ReactJS.

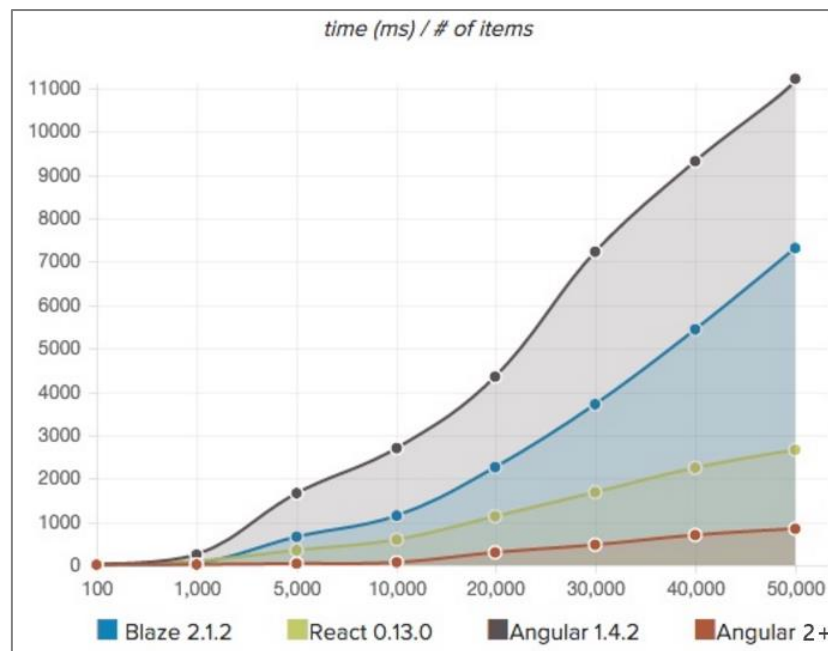


Figure 8: Courbe de performance Blaze vs React vs AngularJS vs Angular

Selon la courbe, Angular 2+ est le plus performant, il affiche 50 000 articles dans 1000 ms. Pour cela, nous allons utiliser Angular 5 qui est une version améliorée d'Angular 2.

Concernant la partie backend, nous allons opter pour un « Serverless Framework » en utilisant les microservices AWS pour les raisons bien expliquées dans le premier chapitre.

## II.4.5 Conception de l'application

### II.4.5.1 Diagramme de classes

L'application intitulée « Billyourself » comporte deux classes : une classe pour les utilisateurs et une autre pour les factures.

Chaque utilisateur possède un identificateur unique, un email et un mot de passe. Chaque facture possède un identificateur unique, un attribut pour le nombre de jours de

travail, un autre pour le revenu par jour, un nom de client, une adresse de client et une description.

Chaque utilisateur a le droit d'ajouter une ou plusieurs factures, supprimer une ou plusieurs factures ou afficher la liste de ses factures comme la figure ci-dessous l'indique.



Figure 9: Diagramme de classes

## II.4.5.2 Diagramme de cas d'utilisation

### II.4.5.2.1 Diagramme de cas d'utilisation coté front office

Le diagramme de cas d'utilisation ci-dessous présente les fonctionnalités que l'application « BillYourself » pourrait offrir du coté front office.

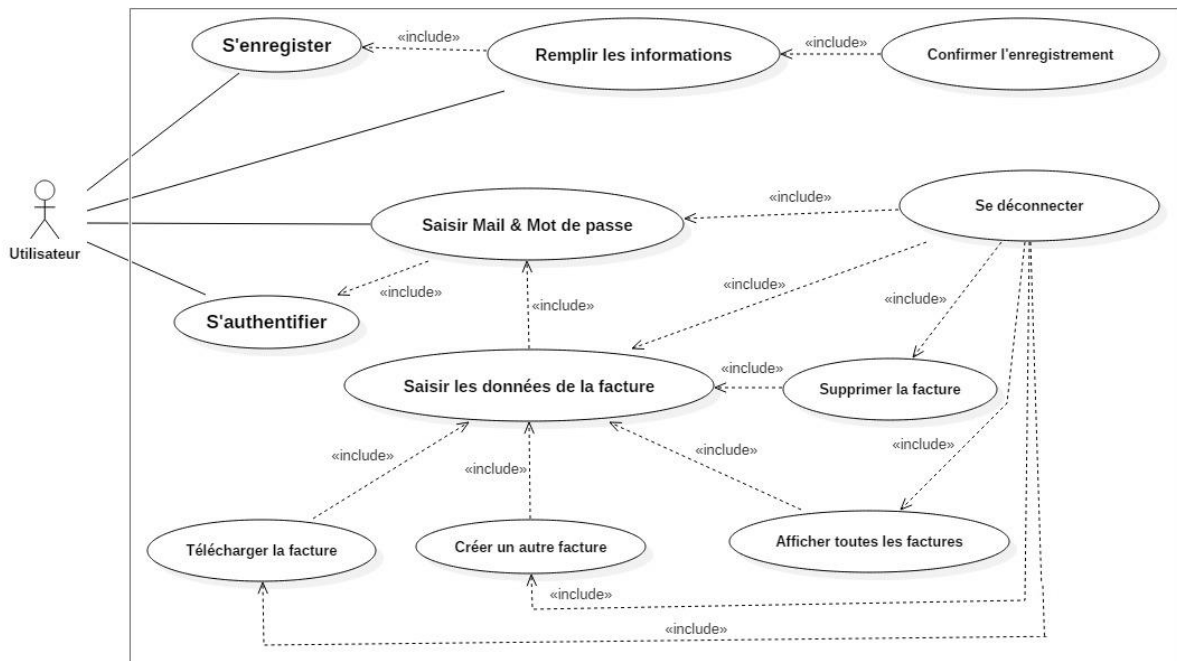


Figure 10: Diagramme de cas d'utilisation coté front office

Chaque utilisateur doit, tout d'abord, s'enregistrer en remplissant les informations nécessaires et en confirmant son enregistrement par email.

Après la création de son compte, l'utilisateur pourrait désormais s'authentifier à l'application en saisissant son email et son mot de passe.

Si l'authentification réussie, l'utilisateur se trouve devant une nouvelle interface où il est demandé de remplir les données de la facture et de confirmer.

Après avoir confirmé, l'utilisateur se trouve devant une nouvelle interface où sa facture est générée avec la possibilité de la télécharger en format PDF. En outre, cette interface donne la possibilité aussi de créer une autre facture, de supprimer la facture ou bien de consulter la liste de toutes les factures.

À chaque étape décrite, l'utilisateur a la possibilité de tout interrompre et de se déconnecter de l'application

#### II.4.5.2.2 Diagramme de cas d'utilisation coté back office

Du coté back office, l'administrateur doit accéder à sa console AWS pour pouvoir gérer les utilisateurs et la base de données. La figure ci-dessous présente le diagramme de cas d'utilisation coté back office.

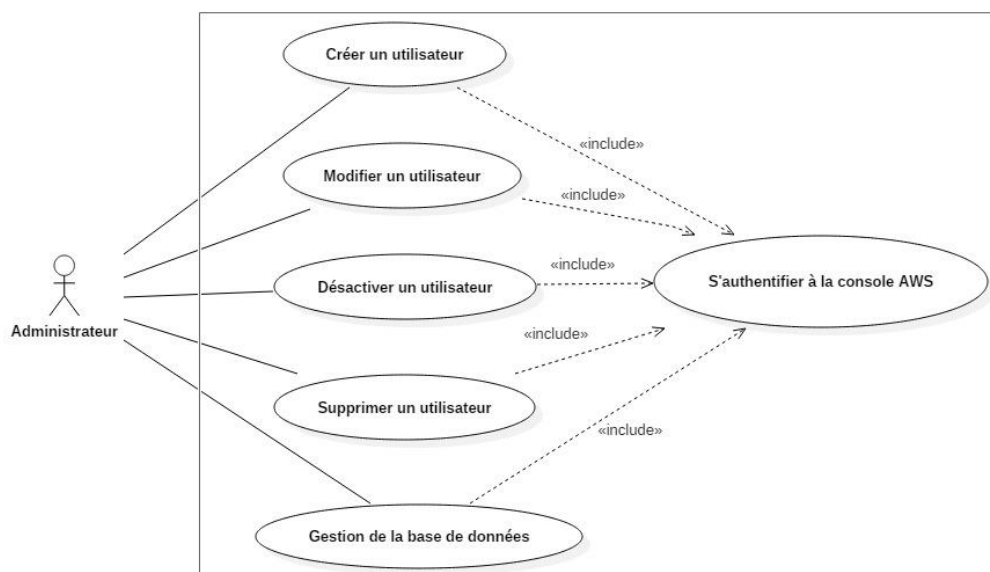


Figure 11: Diagramme de cas d'utilisation coté back office

Après avoir accédé à la console AWS, chaque administrateur (qui est un utilisateur AWS ayant la stratégie « AdministratorAccess ») a le droit de créer, modifier, désactiver et supprimer des utilisateurs en utilisant le service AWS Cognito. De plus, l'administrateur gère la base de données : il peut modifier et supprimer des données.

### II.4.5.3 Diagramme de séquences

Le diagramme de cas d'utilisation est un diagramme de plusieurs scénarios, chaque scénario contient des interactions entre l'utilisateur et l'application « Billyourself ».

Le diagramme ci-dessous s'intéresse à un scénario où l'utilisateur s'enregistre à l'application puis s'authentifie, saisie les données de sa facture et la télécharge en format PDF.

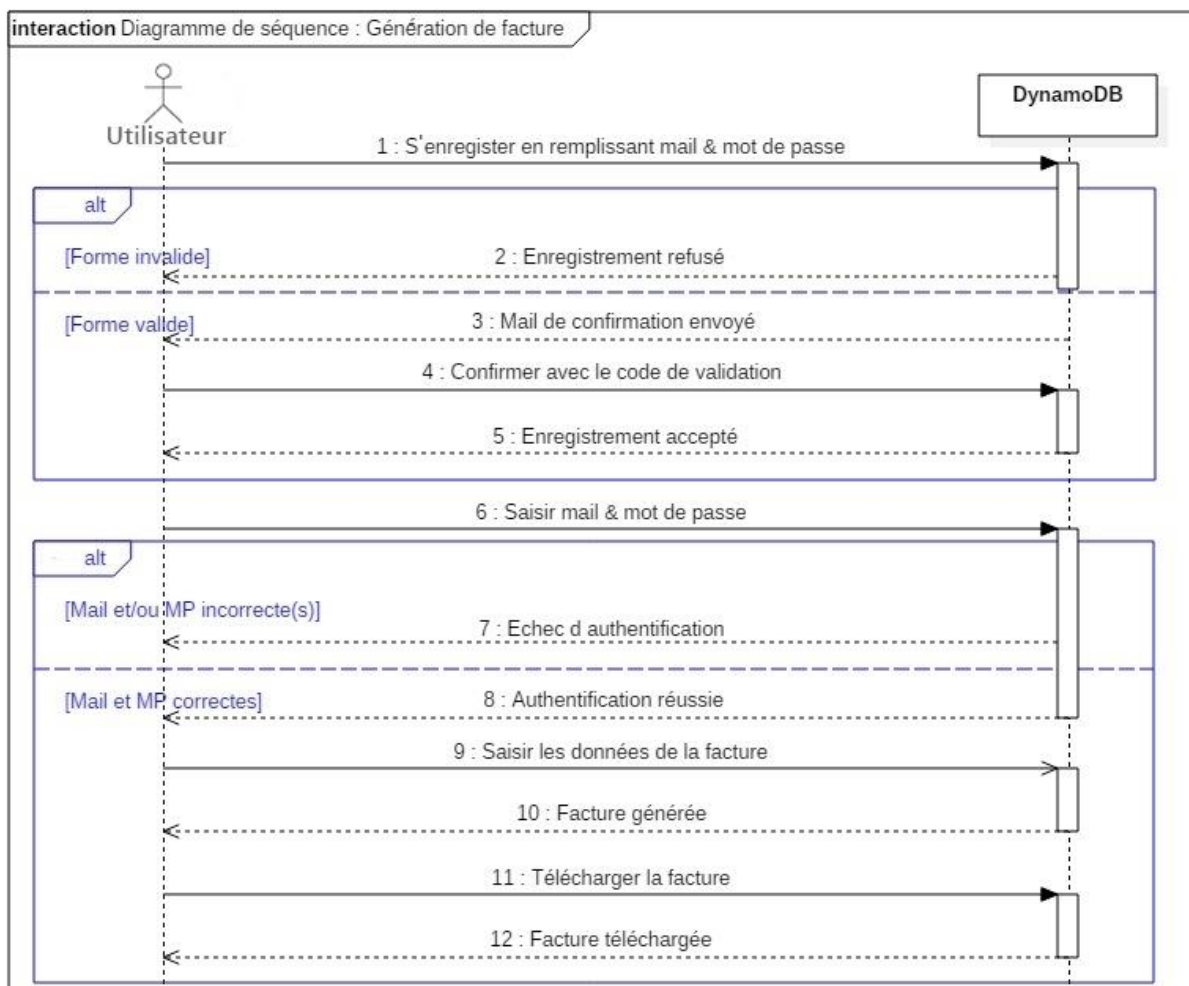


Figure 12: Diagramme de séquences : Scénario de génération d'une facture

Au cours de l'enregistrement, une vérification des données aura lieu : Il faut saisir un email et un mot de passe à au moins six caractères. Un email de confirmation sera envoyé par email. Si le code de confirmation est saisi correctement, l'enregistrement sera accepté. Dans le cas contraire, l'enregistrement sera refusé.

Au cours de l'authentification, l'application vérifie si l'email et le mot de passe sont bien valides pour pouvoir s'authentifier. Dans le cas contraire, il y aura un échec d'authentification.

Au cours de la saisie des données de la facture, il faut respecter la forme des champs de saisie (champs numériques, champs à chaîne de caractères) pour pouvoir valider et générer la facture.

## **II.5 Conclusion**

Dans ce chapitre, nous avons présenté l'architecture globale du projet, la conception architecturale de l'infrastructure et de l'environnement de développement et la conception de l'application à développer. Le prochain chapitre s'étalera sur les différentes étapes à suivre pour la réalisation du projet.



---

## ***Chapitre III : Réalisation du projet***

---

### **III.1 Introduction**

Ce chapitre détaille les différentes étapes de la réalisation du projet, Premièrement, nous allons mettre en place et configurer notre environnement de développement dans le cloud. Ensuite, nous allons entamer le développement de l'application en mode SaaS et en culture DevOps.

### **III.2 Mise en place automatisée de l'infrastructure**

#### **III.2.1 Présentation**

Pour déployer automatiquement l'infrastructure avec plus d'agilité et de flexibilité, l'approche DevOps considère l'infrastructure comme du code (Infrastructure as Code). Le concept « IaC » est similaire à celui des scripts de programmation utilisés pour automatiser les processus importants.

Toutefois, nous avons une série d'étapes sensibles et/ou qui doivent être répétées de nombreuses fois, le concept « IaC » fait appel à un langage descriptif pour coder ces étapes de déploiement et de mise à disposition afin de réduire le nombre d'erreurs et gagner du temps. De nombreux outils et solutions permettent d'automatiser le déploiement et la gestion du cycle de vie de l'infrastructure. Nous citons Terraform.

Terraform [5] est un outil open source, édité par « Hashicorp », qui s'inscrit dans la mouvance « IaC ». Terraform permet de créer et de modifier de façon sûre et prévisible une infrastructure de production, et de décrire les services qui gravitent autour sous la forme de fichiers de configuration.



Terraform orchestre les appels d'API pour un nombre important de fournisseurs d'infrastructure (AWS, Google Cloud, Microsoft Azure, OVH, Openstack, Kubernetes, VMWare, etc.). Il se base sur des fichiers de configuration déclaratifs, traités comme du code afin de réduire les processus manuels au moyen de l'automatisation.

### III.2.2 Création automatisée de l'infrastructure à l'aide de Terraform

Afin d'automatiser le processus de la création de l'infrastructure décrite précédemment, nous avons utilisé Terraform comme un outil « IaC » en se basant sur l'ensemble des fichiers de configuration suivant :

```
SoftwareFactory
|__ provider.tf
|__ vpc.tf
|__ security-group.tf
|__ key.tf
|__ instance.tf
|__ vars.tf
|__ terraform.tvars
|__ requirements.sh
```

Tous les fichiers sont regroupés dans un même répertoire. Les fichiers d'extension « tf » (\*.tf) définissent l'infrastructure à générer (Fournisseurs, VPCs, groupes de sécurité, instances, clés, etc.). La configuration dans son ensemble peut être répartie dans plusieurs fichiers.

Dans notre cas, nous voulons déployer une infrastructure AWS. Pour cela, nous avons créé un utilisateur en utilisant le service IAM d'AWS et nous avons affecté la stratégie « AdministratorAccess » pour cet utilisateur dans le but d'avoir un accès total à tous les services AWS disponibles.

Maintenant, nous pouvons créer notre infrastructure en utilisant les services AWS adéquats et pour commencer, nous avons créé une ressource « aws » dans le fichier « provider.tf » en précisant la région sur laquelle notre infrastructure va être créée, la version du « provider » utilisée ainsi que l'« Access Key » et le « Secret Key » de notre utilisateur IAM créé :

```
provider
  "aws"
  {
    access_key = "${var.AWS_ACCESS_KEY}"
    secret_key = "${var.AWS_SECRET_KEY}"
    region     = "${var.AWS_REGION}"
    version    = "~> 1.20"
```

Le fichier « vpc.tf » permet de créer nos ressources indiquées précédemment :

- Un VPC intitulé « main » avec un bloc d'adresse CIDR IPv4 10.0.0.0/16.

```
resource "aws_vpc" "main" {
    cidr_block = "10.0.0.0/16"
    instance_tenancy = "default"
    enable_dns_support = "true"
    enable_dns_hostnames = "true"
    enable_classiclink = "false"
    tags { Name = "main" }
}
```

- Un sous-réseau public attaché au VPC « main » avec un bloc d'adresse CIDR IPv4 10.0.0.0/24 et dont la zone de disponibilité est « us-east-2a ».

```
resource "aws_subnet" "main-public" {
    vpc_id = "${aws_vpc.main.id}"
    cidr_block = "10.0.1.0/24"
    map_public_ip_on_launch = "true"
    availability_zone = "us-east-2a"
}
```

- Une passerelle attachée au VPC « main » pour l'accès à Internet.

```
resource "aws_internet_gateway" "main-gw" {
    vpc_id = "${aws_vpc.main.id}"
}
```

- Une table de routage attachée au VPC « main » et possède deux entrées :  
Une première entrée par défaut pour le routage local dans le VPC et une entrée vers la passerelle comme expliqué dans la partie [II.2.2].

```
resource
    "aws_route_table"
    "main-public" {
        vpc_id = "${aws_vpc.main.id}"
        route {
            cidr_block = "10.0.0.0/16"
            gateway_id = "${aws_internet_gateway.main-gw.id}"
        }
    }
}
```

- Une route qui associe la table de routage au sous-réseau public.

```
resource
  "aws_route_table_association"
  "main-public-a" {
    subnet_id = "${aws_subnet.main-public.id}"
    route_table_id = "${aws_route_table.main-public.id}"
  }
```

Concernant l'ACL, elle va être créée automatiquement avec le VPC avec une configuration par défaut qui autorise tout le trafic entrant et sortant, puisque nous n'avons qu'une seule couche de sécurité au niveau du sous-réseau et nous n'en avons pas une au niveau du VPC.

Le fichier « security-group.tf » permet la configuration d'un groupe de sécurité qui requiert les attributs « name » et « description ». Chaque règle de sécurité est définie par les attributs « ingress » pour un trafic entrant ou « egress » pour le trafic sortant.

```
resource
  "aws_security_group"
  "SoftwareFactory" {
    vpc_id = "${aws_vpc.main.id}"
    name = "usine_logicielle"
    description = "security group of the Software Factory"
    egress {
      from_port = 0
      to_port = 0
      protocol = "-1"
      cidr_blocks = ["0.0.0.0/0"]
    }
    # http access from anywhere
    ingress {
      from_port = 80
      to_port = 80
      protocol = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
    ...
    ...
  }
```

Le fichier « key.tf » permet de créer la ressource clé SSH avec laquelle nous pouvons se connecter à l'instance EC2 à distance.

```
resource "aws_key_pair"
  "mykeypair" {
    key_name = "my-key"
    public_key = "${file("${var.PATH_TO_PUBLIC_KEY}")}"
  }
```

Le fichier « instance.tf » permet de créer une ressource de type instance EC2 en indiquant son nom, son type, le VPC auquel elle est attachée et son groupe de sécurité ainsi que sa clé SSH avec laquelle nous y pouvons se connecter à distance.

Terraform aborde la notion de « provisionning » pour le transfert de fichiers et l'exécution de commandes locales ou distantes. Le provisionner « remote-exec » utilise la connexion déclarée dans la même instance EC2 pour faire appel au fichier « requirements.sh » pour exécuter des commandes Shell dont le rôle est d'installer des dépendances et des packages dont nous aurons besoin pour installer notre usine logicielle plus tard tels que :

- « AWS CLI » pour accéder à la console AWS et gérer ses différents services en utilisant la ligne de commande.
- « Python » pour l'exécution des « playbooks » d'Ansible.
- « JAVA » pour pouvoir installer Jenkins.
- « Maven » pour pouvoir exécuter les tests de développement avec Jenkins.
- « NPM » pour construire (build) des projets de type NodeJS/AngularJS avec Jenkins.

Il est possible aussi d'exporter toutes les valeurs de notre infrastructure dans des fichiers texte ou de les afficher à la fin de l'exécution de Terraform en utilisant l'instruction « outputs » qui affichera l'adresse IPv4 publique de notre instance EC2.

L'instruction « connection » définit la manière de connexion à l'instance. Ici, la connexion se fait via l'utilisateur par défaut « ubuntu » en utilisant la clé SSH spécifiée.

```

resource "aws_instance"
  "main" {

    ami = "${lookup(var.AMIS, var.AWS_REGION)}"
    instance_type = "m5.large"
    tags {
      Name = "SoftwareFactory"
    }
    # the VPC subnet
    subnet_id = "${aws_subnet.main-public-1.id}"
    # the security group
    vpc_security_group_ids =
      ["${aws_security_group.usine_logicielle.id}"]
    # the public SSH key
    key_name = "${aws_key_pair.mykeypair.key_name}"
    provisioner "file" {
      source = "requirements.sh"
      destination = "/tmp/requirements.sh"
    }
    provisioner "remote-exec" {
      inline = [
        "chmod +x /tmp/requirements.sh",
        "sudo /tmp/requirements.sh"
      ]
    }
    connection {
      user = "${var.INSTANCE_USERNAME}"
      private_key = "${file("${var.PATH_TO_PRIVATE_KEY}")}"
    }
  }
  # outputs
  output "ip" { value = "${aws_instance.main.public_ip}" }

```

Le fichier « var.tf » est utilisé pour déclarer les variables non sensibles tels que la région AWS, le nom de l'instance, son AMI dans la région spécifiée et les chemins vers les deux clés publique et privée.

L'AMI est une image virtuelle délivrée par AWS qui sert à créer des machines virtuelles EC2. Nous avons choisi d'utiliser une image Ubuntu 16 dans la région « us-east-2 ».

```

variable "AWS_REGION" {
    default = "us-east-2"
}
variable "PATH_TO_PRIVATE_KEY" {
    default = "my-key"
}
variable "PATH_TO_PUBLIC_KEY" {
    default = "my-key.pub"
}
variable "INSTANCE_USERNAME" {
    default = "ubuntu"
}
variable "AMIS" {
    type = "map"
    default = {
        us-east-2 = "ami-98023efd"
    }
}

```

Le fichier « terraform.tfvars » contient les identifiants des différents fournisseurs (« AWS Access Key » et « AWS Secret Key » dans notre cas), auquel cas il ne devrait pas être enregistré dans le gestionnaire de source.

Avant de se lancer, prévisualisons l'impact de notre configuration en utilisant la commande « terraform plan » qui affiche le plan d'exécution sans l'appliquer. Cette commande montre par la suite les différences de configuration avec l'existant.

```
$ terraform plan
```

La configuration peut désormais être appliquée en utilisant la commande « terraform apply » qui crée l'ensemble de l'infrastructure la première fois puis applique que les modifications par la suite.

```
$ terraform apply
```

Lors de la création de l'infrastructure, Terraform génère un fichier d'état intitulé « terraform.tfstate ». Ce fichier est lisible humainement à des fins de débogage. Il contient l'état du projet et il est indispensable pour définir les opérations de mise à jour.



### III.3 Mise en place automatisée, configuration et sécurisation de l'usine logicielle

#### III.3.1 Présentation

Mettre en place un serveur manuellement peut rapidement s'avérer pénible, surtout si nous avons pas mal de chose à configurer. Pour une telle situation, nous proposons d'opter pour une solution automatisée en utilisant Ansible.

Ansible [6] est un logiciel de configuration et déploiement à distance et est écrit en Python. Il utilise seulement SSH et ne nécessite qu'une simple station de travail équipé de Python 2.4+. Ansible permet de créer des « playbooks » qui ne sont autre que des scripts permettant de configurer vos serveurs.



#### III.3.2 Mise en place d'une usine logicielle automatisée avec Ansible

Ansible repose sur le protocole SSH et c'est via ce protocole qu'il se connecte aux serveurs distants. Ces serveurs sont définis par le fichier « hosts » dont le chemin est « /etc/ansible/hosts ». La définition des installations et des configurations se fait dans un « playbook ». Ceci est illustré dans la figure ci-dessous.

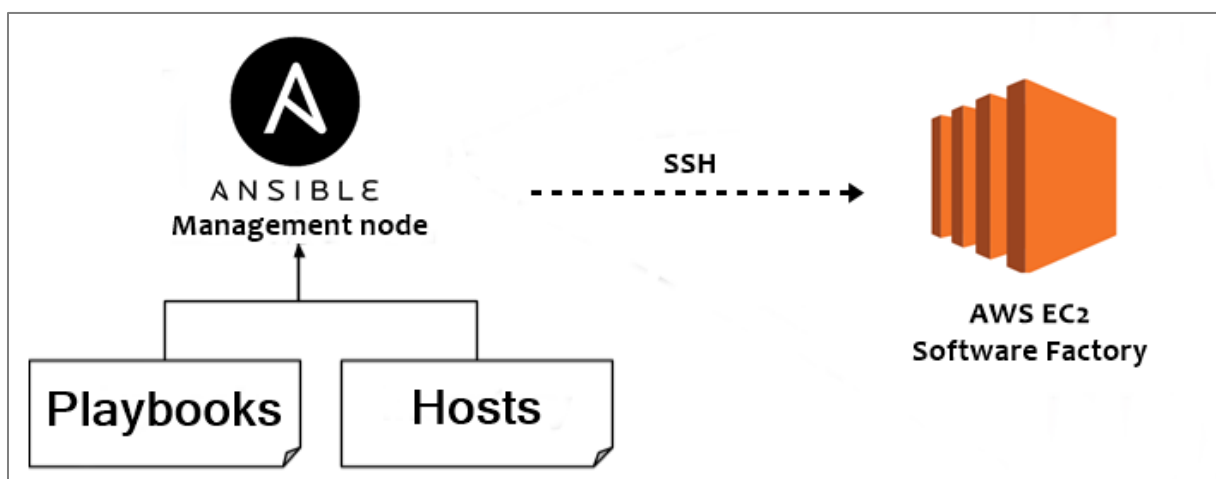


Figure 13: Ansible et les machines distantes

Après avoir bien défini notre serveur qui est l'instance EC2 qu'on a créée dans le cloud AWS pour placer notre usine logicielle, il faut maintenant créer un « playbook » Ansible où nous définissons la configuration et les commandes à exécuter sur notre serveur. Ce « playbook » doit contenir les instructions suivantes :

- Installation du gestionnaire de version Git.
- Installation et lancement du logiciel d'intégration continue Jenkins sur le port 8080.
- Installation et lancement du logiciel de la mesure de la qualité du code SonarQube sur le port 9000.

Nous avons créé alors un fichier YAML intitulé « SoftwareFactory.yml » qui va permettre de décrire notre recette. Ce fichier a la forme ci-dessous. La configuration est bien détaillée dans l'annexe A.

```
---
- hosts: # EC2 IPv4 @
  become: true
  tasks:
  - name: Install Software Factory components.
    apt: name = {{item}} state = present update_cache=yes
    with_items:
      - ...
      - ...
```

Enfin, il faut utiliser la commande « ansible-playbook » pour lancer le déploiement :

```
$ ansible-playbook SoftwareFactory.yml
```

Maintenant, nous avons mis en place les différents composants de notre usine logicielle. Il est temps maintenant de faire la liaison entre ces éléments et les intégrer ensemble.

### III.3.3 Configuration de l'usine logicielle

Dans cette partie, nous allons intégrer les différents composants de la chaîne de développement ensemble à savoir Jenkins, AWS, GitHub, SonarQube et Slack, comme illustré dans la figure ci-dessous. Ainsi, nous allons configurer NodeJS dans Jenkins pour le build des projets.

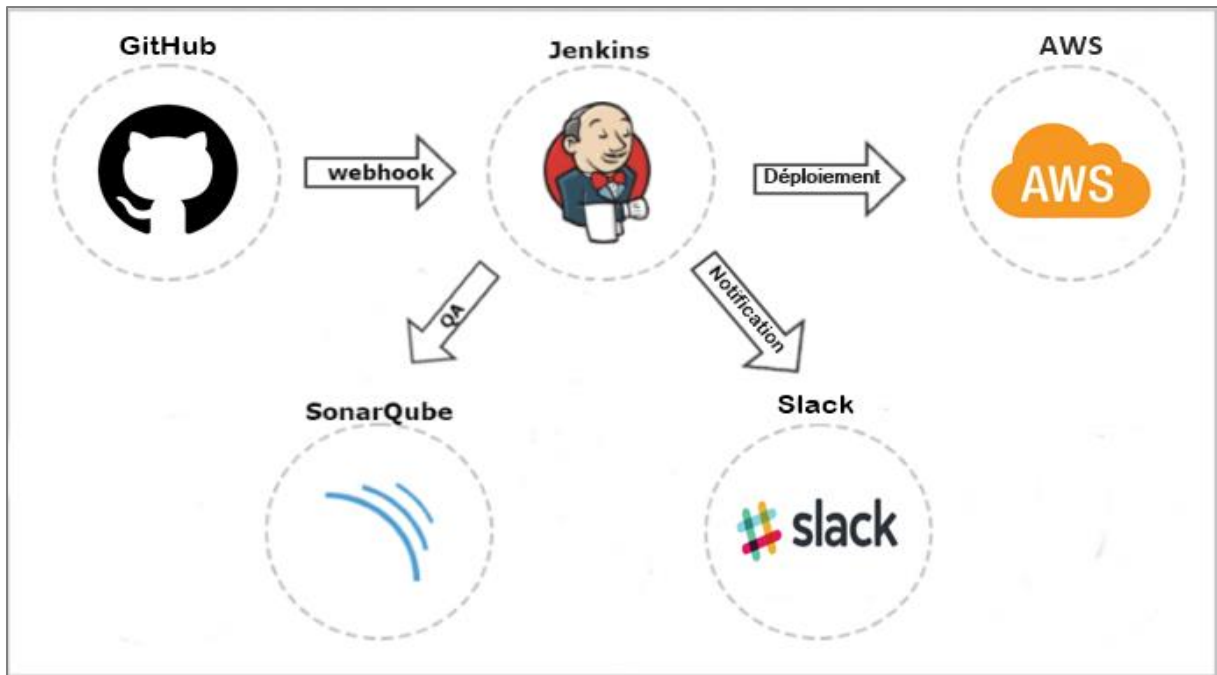


Figure 14: Intégration des différents composants de la chaîne de développement

### III.3.3.1 Configuration de AWS CLI

Pour que Jenkins puisse interagir avec AWS en utilisant AWS CLI, il faut configurer ses paramètres et donner leur accès à Jenkins. La commande « `aws configure` » est le moyen le plus rapide pour configurer l'installation de notre AWS CLI. En tapant cette commande, AWS CLI invite à fournir quatre éléments d'information :

- « AWS Access Key » et « AWS Secret Key » : ils constituent les informations d'identification de compte.
- Le format de sortie par défaut : il peut être « json », « text » ou « table ».
- La région par défaut : c'est le nom de la région dans laquelle nous souhaitons effectuer des appels par défaut. Il s'agit généralement de la région la plus proche, mais il peut aussi s'agir de n'importe quelle autre région.

Cette configuration peut être spécifique pour les différents utilisateurs de l'usine logicielle. Dans notre cas, nous allons configurer l'utilisateur « jenkins » comme suit :

```

$ aws configure --profile jenkins

AWS Access Key ID [None]: *****
AWS Secret Access Key [None]: *****
Default region name [None]: us-east-2
Default output format [None]: json
  
```

### III.3.3.2 Configuration de NodeJS

Cette étape consiste à configurer NodeJS dans Jenkins pour que ce dernier puisse faire le build des projets. Pour ce faire, il faut installer le plugin « NodeJS Plugin » en utilisant le gestionnaire des plugins de Jenkins, puis configurer NodeJS à travers la configuration globale des outils de Jenkins comme suit :

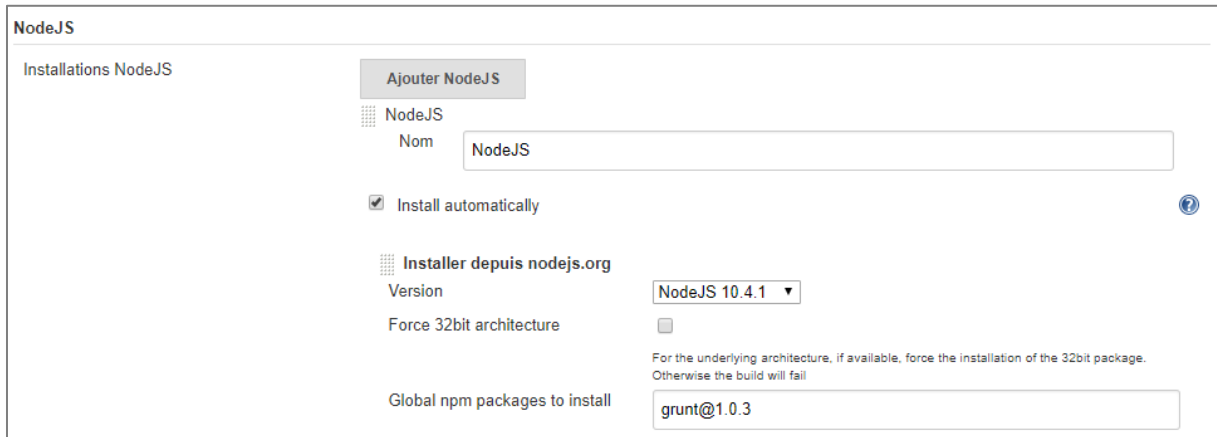


Figure 15: Configuration de NodeJS

### III.3.3.3 Intégration de GitHub avec Jenkins

Cette étape consiste à configurer Jenkins pour parler à GitHub. Pour cela, nous avons installé le plugin « GitHub Plugin » à travers le gestionnaire des plugins de Jenkins.

Du côté de GitHub, il faut ajuster les paramètres du projet, qu'on désire le construire avec Jenkins, en créant un « webhook » pour notre serveur Jenkins en activant l'option « Jenkins (GitHub plugin) » et en le remplissant avec une URL similaire à celle-ci :

```
http://Jenkins_Server_IPv4@:8080/github-webhook/
```

Si la configuration est bien faite, une coche en vert doit s'afficher :



Figure 16: Vérification de l'intégration de GitHub avec Jenkins

Enfin, il faut cocher la case « Build when a change is pushed to GitHub » dans le projet sur Jenkins pour assurer son build automatique après chaque nouveau commit sur GitHub.

### III.3.3.4 Intégration de SonarQube avec Jenkins

Cette étape consiste à interconnecter Jenkins et SonarQube. Pour cela, nous avons installé le plugin « SonarQube Scanner for Jenkins » à travers le gestionnaire des plugins de Jenkins.

Ensuite, il faut configurer le serveur de SonarQube en ajoutant son URL dans la section « SonarQube servers » de la configuration système de Jenkins :



Figure 17: Configuration du serveur SonarQube dans Jenkins

Enfin, il faut ajouter le chemin de « SonarRunner » dans la section « SonarQube Scanner » de la configuration globale des outils de Jenkins :

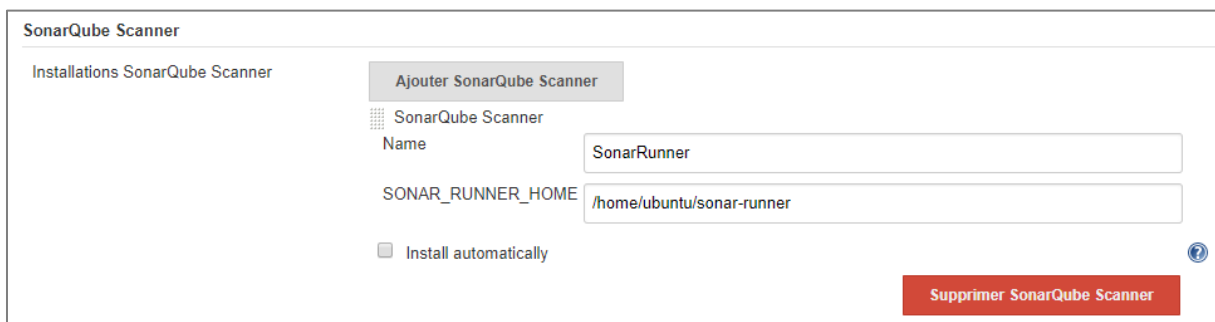


Figure 18: Ajout du chemin de SonarRunner dans Jenkins

### III.3.3.5 Intégration de Slack avec Jenkins

Slack est une excellente plateforme de communication collaborative pour la communication d'équipe que nous avons utilisé pendant ce projet. Parmi les fonctionnalités de Slack, la réception des notifications des builds de Jenkins pour informer les membres de l'équipe des builds faits et publier les URLs des serveurs mis en place.

Pour cela, nous avons installé le plugin « Slack Notification » à travers le gestionnaire des plugins de Jenkins et nous avons ajouté l'application « Jenkins CI » à Slack :

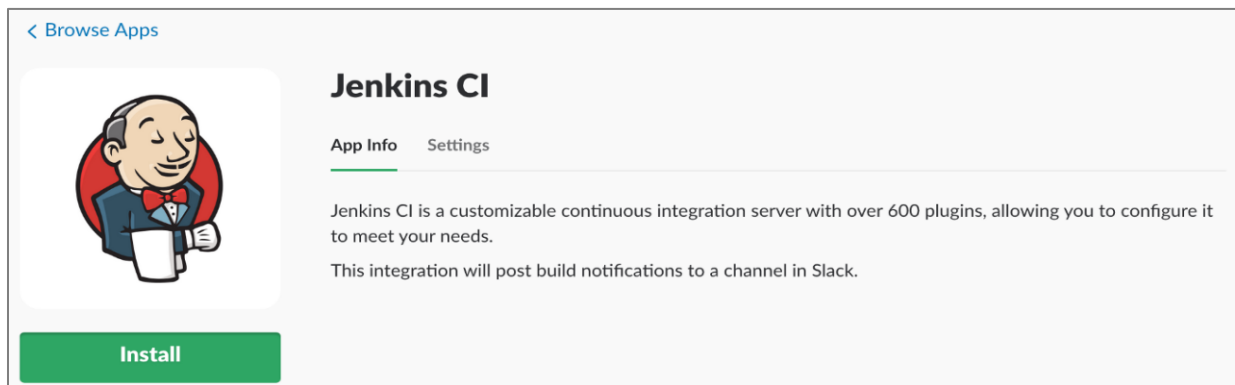


Figure 19: Ajout de l'application « Jenkins CI » à Slack

Ensuite, nous avons configuré le plugin dans la section « Global Slack Notifier Settings » de la configuration système de Jenkins où il faut remplir l'URL de notre service « Jenkins CI » de Slack, le sous-domaine de l'équipe (« oofreelance » dans notre cas), le canal utilisé (« général » dans notre cas) et les informations d'identification du jeton d'intégration (Integration Token Credential ID).

Enfin, il est possible de tester la connexion. Si la configuration est bien faite, le mot « Success » s'affiche comme indiqué dans la figure ci-dessous.

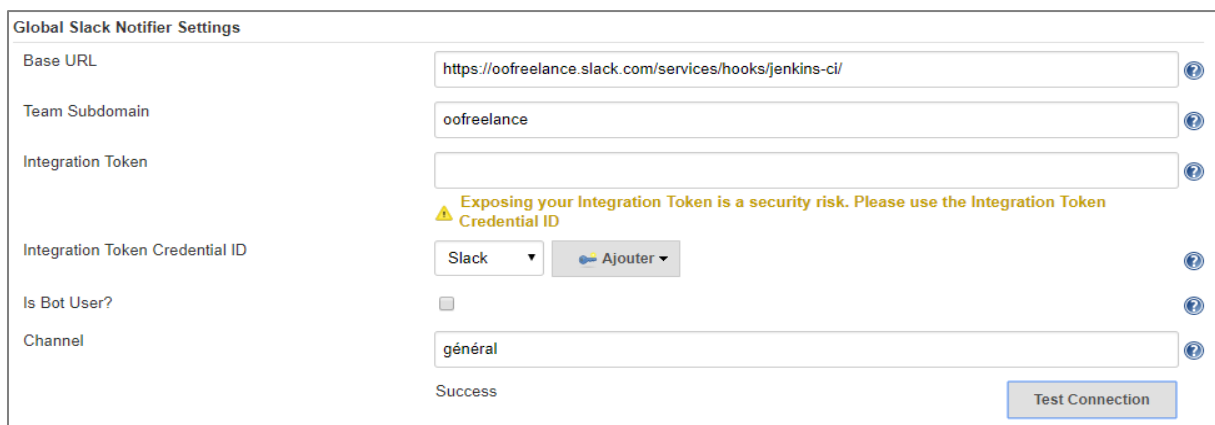


Figure 20: Configuration des paramètres de Slack dans Jenkins

En ce moment, notre chaîne de production logicielle est bien configurée et liée entre elle et nous pouvons maintenant importer nos projets à partir de GitHub, mesurer leur qualité du code avec SonarQube, les construire avec Jenkins si une certaine qualité est bien garantie et enfin envoyer les notifications de build vers Slack.

### III.3.4 Sécurisation de l'usine logicielle avec SSL

Par défaut, Jenkins se met à l'écoute sur le port 8080. Tant que nous avons de vraies données de production, il est préférable d'utiliser un serveur Web plus sécurisé comme Nginx. Pour cela, nous allons encapsuler notre site en configurant un certificat SSL auto-signé à utiliser avec le serveur Web Nginx comme étant un proxy inverse pour Jenkins.

SSL est un protocole Web utilisé pour rendre le trafic entre le serveur et ses clients chiffré et protégé. Grâce à cette technologie, les serveurs peuvent envoyer du trafic en toute sécurité à leurs clients sans que les messages puissent être interceptés par des tiers.

Tout d'abord, nous créons un certificat SSL. Nous avons défini le nombre de bits à 2048 car c'est le minimum nécessaire pour le faire signer par une autorité de certification :

```
$ mkdir -p /etc/nginx
$ cd /etc/nginx
$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/cert.key -
out /etc/nginx/cert.crt
```

Ensuite, nous installons Nginx et nous le configurons en utilisant Ansible. Nous créons alors un fichier YAML intitulé « Nginx.yml » qui décrira l'installation et la configuration de Nginx en tant que proxy inverse. Ceci est bien détaillé dans l'annexe B.

A ce stade, nous avons bien configuré notre serveur Nginx pour gérer nos requêtes en toute sécurité comme montré dans la figure ci-dessous, ce qui permet d'éviter que des tiers ne puissent intercepter le trafic.

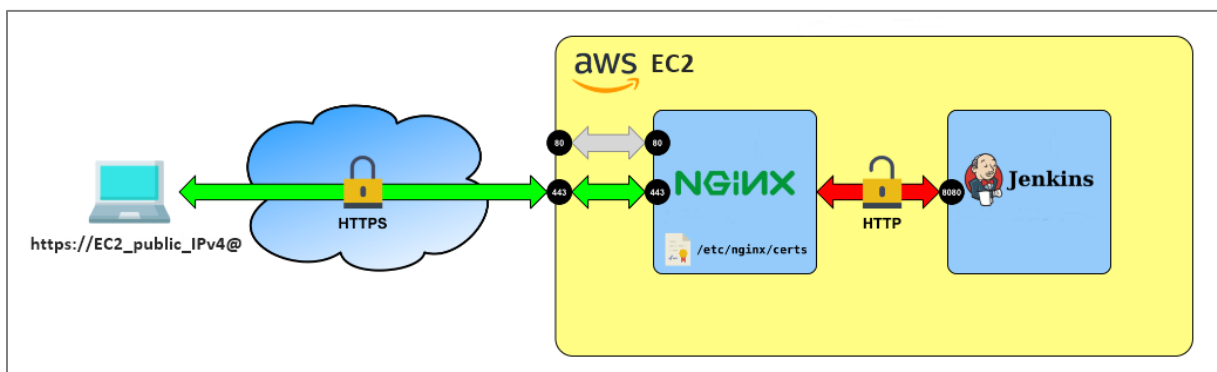


Figure 21: Sécuriser le trafic entre serveur/client en utilisant HTTPS

Passons maintenant à l'étape suivante qui est la mise en place d'un serveur de monitoring en utilisant la stack ELK afin de centraliser les journaux d'événements (logs) de nos serveurs et de nos applications.

### III.4 Mise en place de la solution de monitoring

Tout d'abord, Nous allons installer la pile ELK sur une instance EC2 dans le cloud AWS et FileBeat sur l'instance de l'usine logicielle. Ces installations sont bien détaillées dans l'annexe C. Puis, nous allons faire les configurations adéquates afin de collecter et visualiser les différents logs de l'usine logicielle dans un emplacement centralisé.

#### III.4.1 Génération d'un certificat SSL

Puisque nous allons utiliser FileBeat pour expédier les logs de notre serveur distant vers notre serveur ELK, nous devons créer un certificat SSL et une paire de clés. Le certificat est utilisé par FileBeat pour vérifier l'identité du serveur ELK.

Pour cela, il faut créer les répertoires qui vont stocker le certificat et la clé privée en utilisant les commandes suivantes :

```
$ sudo mkdir -p /etc/pki/tls/certs
$ sudo mkdir -p /etc/pki/tls/private
```

Ensuite, il faut ajouter l'adresse IP privée de notre serveur ELK au champ « subjectAltName » du certificat SSL, que nous sommes sur le point de générer, dans la section « [v3\_ca] » du fichier « /etc/ssl/openssl.cnf ». Pour ce faire, il faut modifier le fichier de configuration OpenSSL :

```
subjectAltName = IP: ELK_server_private_IPv4_@
```

Puis, il faut générer le certificat SSL et la clé privée dans leur emplacement approprié (/etc/pki/tls/) avec les commandes suivantes :

```
$ cd /etc/pki/tls
$ sudo openssl req -config /etc/ssl/openssl.cnf -x509 -days 3650 -batch -nodes -newkey
rsa:2048 -keyout private/logstash-forwarder.key -out certs/logstash-forwarder.crt
```



Le fichier « logstash-forwarder.crt » sera copié sur le serveur distant, qui enverra ses logs à Logstash, dans le répertoire « /etc/pki/tls/certs/ ». Ce fichier est requis pour la communication entre les deux serveurs.

### III.4.2 Configuration de Logstash

Les fichiers de configuration de Logstash sont au format JSON et se trouveront dans « /etc/logstash/conf.d ». La configuration comprend trois sections : entrées, filtres et sorties. Commençons par la création du fichier de configuration appelé « 02-beats-input.conf » où nous allons configurer notre entrée FileBeat en y ajoutant la configuration suivante :

```
input {
  beats {
    port => 5044
    ssl => true
    ssl_certificate => "/etc/pki/tls/certs/logstash-forwarder.crt"
    ssl_key => "/etc/pki/tls/private/logstash-forwarder.key"
  }
}
```

Cela spécifie une entrée FileBeat qui se met à l'écoute sur le port TCP 5044, et utilise le certificat SSL et la clé privée que nous avons créés précédemment.

Ensuite, créons un fichier de configuration appelé « 10-syslog-filter.conf », où nous allons ajouter un filtre pour les messages « Syslog » :

```
filter {
  if [type] == "syslog" {
    grok {
      match => {"message" => "%{SYSLOGTIMESTAMP:syslog_timestamp}
%{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[ %{POSINT:syslog_pid} \])?":
%{GREEDYDATA:syslog_message}" }
      add_field => ["received_at", "%{@timestamp}" ]
      add_field => ["received_from", "%{host}" ]
    }
    syslog_pri { }
    date {
      match => [ "syslog_timestamp", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ]
    }
  }
}
```

Ce filtre recherche les logs étiquetés comme du type « Syslog », et il essayera d'utiliser l'instruction « grok » pour les analyser afin de les structurer et de les interroger.

Egalement, il est possible d'ajouter des filtres pour des autres applications (Jenkins et SonarQube) en utilisant la même entrée FileBeat.

Puis, créons un fichier de configuration appelé « 30-elasticsearch-output.conf » où nous allons configurer notre sortie FileBeat en y ajoutant la configuration suivante :

```
output {
  elasticsearch {
    hosts => ["localhost:9200"]
    sniffing => true
    manage_template => false
    index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
    document_type => "%{[@metadata][type]}"
  }
}
```

Cette sortie configure essentiellement Logstash pour stocker les données venant de FileBeat dans ElasticSearch qui s'exécute à « localhost:9200 ».

Finalement, il suffit de redémarrer Logstash pour appliquer nos modifications de configuration en tapant la commande suivante :

```
$ sudo service logstash restart
```

### III.4.3 Configuration de FileBeat

Nous allons maintenant configurer FileBeat pour se connecter à Logstash sur notre serveur ELK en tapant la commande suivante :

```
$ sudo vi /etc/filebeat/filebeat.yml
```

Le fichier de configuration doit avoir la forme suivante :

```
filebeat.prospectors:
- type: log
  enabled: true
  paths:
  - /var/log/syslog
  - /var/log/jenkins/jenkins.log
  - /var/log/sonar/sonar.log
output.logstash:
  # The Logstash hosts
  hosts: ["ELK_Private@:5044"]
  ssl.certificate_authorities: ["/etc/pki/tls/certs/logstash-forwarder.crt"]
```

Dans cette configuration, nous avons spécifié les fichiers entrants qui sont les fichiers logs du système (Syslog), de Jenkins et de SonarQube. En outre, cette configuration permet de se connecter à Logstash sur notre serveur ELK sur le port 5044 (le port pour lequel nous avons spécifié une entrée Logstash) en utilisant le certificat SSL généré précédemment.

Maintenant, il suffit de redémarrer FileBeat pour mettre en place nos modifications de configuration en tapant la configuration suivante :

```
$ sudo service filebeat restart
```

Finalement, en regardant Kibana, l'interface Web que nous avons mise en place, nous pouvons visualiser les logs de notre usine logicielle en précisant l'intervalle de temps désiré. De plus, il est possible de créer des dashboards, diagrammes en barre, en ligne, des nuages de points, des camemberts et des cartes pour mieux visualiser les données.

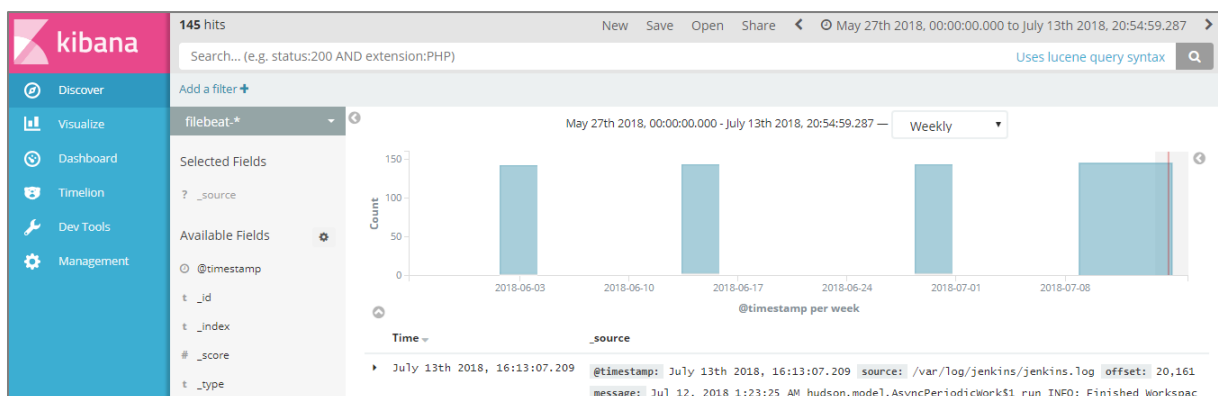


Figure 22: Visualisation des logs à travers Kibana

### III.5 Développement et déploiement de l'application

Dans cette partie, nous allons développer une application Web en mode SaaS, c'est-à-dire l'application va être hébergée entièrement dans le cloud AWS. Nous rappelons que nous avons choisi le framework Angular 5 pour la partie frontale et les microservices AWS pour la partie backend sans serveur.

#### III.5.1 Développement et déploiement de la partie frontend

##### III.5.1.1 Développement

Angular est à base de composants. Utilisant ce framework, notre application, intitulée « BillyYourself », contient principalement deux composants : « user » et « bill » et chacun

contient des sous-composants et des services. Le composant « user » contient les sous-composants « signin » et « signup » qui sont dédiés pour l'enregistrement et l'authentification. Ces derniers font appel au service « auth-service » qui utilisera AWS Cognito pour l'authentification. Le deuxième composant « bill » contient les sous-composants « input » pour saisir les données de la facture, « result » pour générer la facture et « list » pour lister toutes les factures. Ces trois composants font appel au service « bill-service » où les APIs RESTful seront appelées.

L'interface « Sign Up » permet de s'enregistrer à l'application en saisissant un mail valide et un mot de passe à au moins six caractères. Pour que l'utilisateur soit créé, il faut saisir le code de validation envoyé par mail.

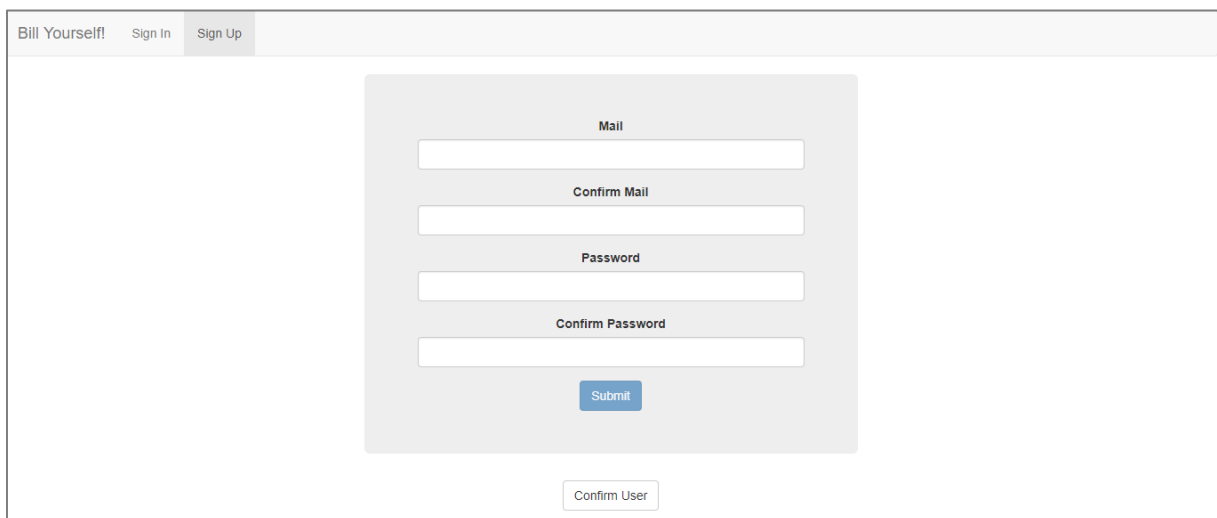


Figure 23: L'interface « Sign Up » de l'application

L'interface « Sign In » permet de s'authentifier à l'application en saisissant le mail et le mot de passe d'un utilisateur valide chacun.

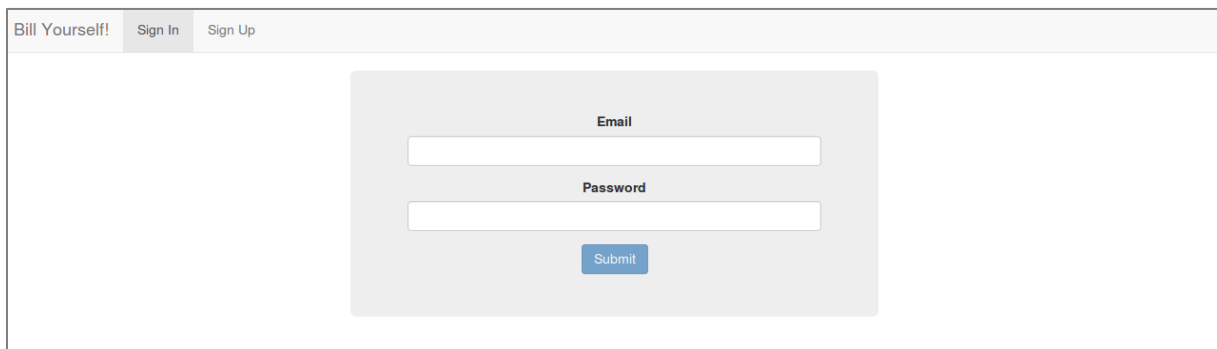


Figure 24: L'interface « Sign In » de l'application

L'interface « Billing » est la page d'accueil après l'authentification. Cette interface permet de saisir les données nécessaires pour générer une facture. Les différents champs à remplir sont : le nombre de jours de travail, le revenu par jour, le nom et l'adresse du client à facturer et la description de la facture.

The screenshot shows a web interface with a navigation bar at the top containing 'Bill Yourself!', 'Billing' (highlighted), and 'Logout'. The main content area is titled 'Set your bill details' and contains a form with the following fields: 'Quantity (Days)', 'Daily Income (€)', 'Client Name', 'Client Address', and 'Description'. Each field is represented by a text input box. At the bottom of the form is a blue 'Submit' button.

Figure 25: L'interface « Billing » de l'application

Après avoir rempli les champs et confirmé, une facture sera créée sous la forme ci-dessous. Cette interface donne la possibilité aux utilisateurs de télécharger la facture en format PDF, de la supprimer, de créer une autre facture et de lister toutes les factures créées.

The screenshot shows the 'Facture' (Invoice) page. It features a navigation bar with 'Bill Yourself!', 'Billing' (highlighted), and 'Logout'. The main content area displays a completed invoice with the following details:

- Facture** (Section Header)
- Préstatataire:** Wael Ben Salah, wael.ben.salah@gmail.com, 1 BOULEVARD JOURDAN 75014 PARIS
- Client:** NTI SOLUTIONS, ESPACE CRISTAL, LE TECHNOPARC, 22 RUE GUSTAVE EIFFEL 78306 POISSY CEDEX
- Application en mode SaaS:** 60 jours à 32 €
- Montant (€):** 1920
- Total HT:** 1920
- TVA:** 384.00
- Total TTC:** 2304.00
- Montant à reverser:** 2304.00 €
- TVA 20%, article 283-2 du Code Général des Impôts

On the right side of the invoice, there are four buttons: 'Télécharger la facture', 'Créer une autre facture', 'Supprimer la facture', and 'Afficher toutes les factures'.

Figure 26: L'interface de la génération de la facture de l'application

Maintenant, nous avons clôturé la phase de développement de la partie frontale de l'application. Cette dernière est prête pour être déployée dans le cloud AWS.

### III.5.1.2 Déploiement

Dans cette partie, nous allons configurer AWS S3 pour héberger les ressources statiques de notre application Web. Cette application peut être déployée dans n'importe quelle région AWS. Nous avons choisi de travailler sur la région « Ohio » comme dans la première partie de la réalisation.

Premièrement, nous avons créé un compartiment (bucket) S3, qui sera utilisé pour contenir les différents fichiers de l'application (HTML, CSS, TypeScript, images, etc.), en utilisant la console AWS ou bien l'interface de ligne de commande (AWS CLI) puisque nous avons déjà installé et configuré « awscli » en créant l'usine logicielle. Nous avons nommé ce compartiment « bill-yourself ». Ce nom doit être unique dans le monde puisque AWS S3 est une solution universelle.

Deuxièmement, nous avons téléchargé les fichiers de l'application après son build, c'est-à-dire le contenu du répertoire « dist/BillYourself » dans le projet Angular. Ce téléchargement est possible en utilisant la console AWS ou bien l'interface de ligne de commande (AWS CLI) en utilisant les commandes suivantes :

```
$ cd ~/DevOps/GitHub/BillYourself
$ aws s3 cp dist/BillYourself s3://bill-yourself.oofreelance.com/
```

Troisièmement, nous avons ajouté la stratégie de compartiment ci-dessous à « bill-yourself » pour permettre aux différents utilisateurs de voir notre site Web car par défaut, il n'est pas accessible au public :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::bill-yourself.oofreelance.com/*"
    }
  ]
}
```

Finalement, nous avons activé la fonction d'hébergement de site du compartiment. Cette fonction rendra nos objets disponibles par le point de terminaison suivant :

⇒ <http://bill-yourself.oofreelance.com.s3-website.us-east-2.amazonaws.com>

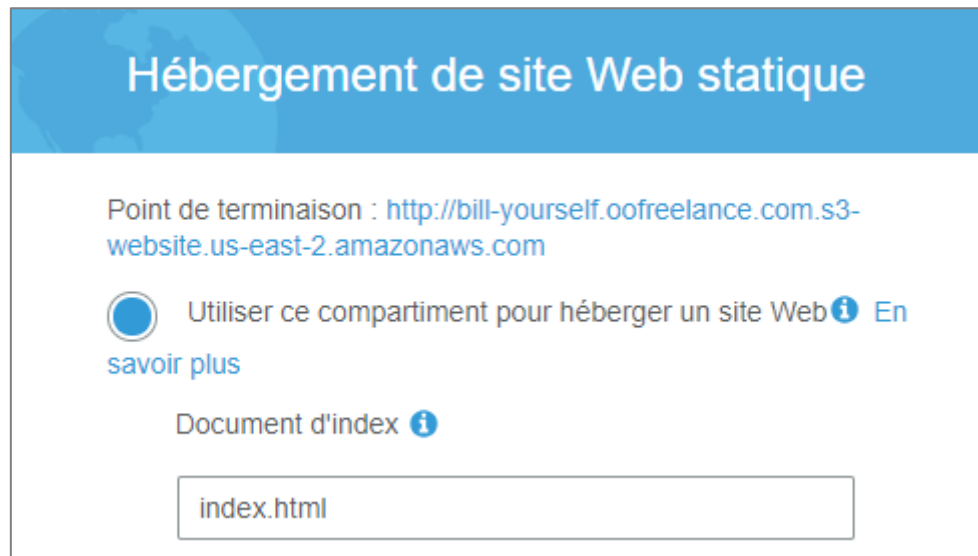


Figure 27: Hébergement de site Web en utilisant AWS S3

Maintenant, les utilisateurs pourront accéder à notre application à l'aide de l'URL du site Web public indiquée dans la figure ci-dessous et nous n'avons pas besoin d'exécuter un serveur ni d'utiliser d'autres services pour le rendre disponible.

### III.5.1.3 Création d'un enregistrement

L'objectif de cette partie est de configurer AWS Route 53 en tant que fournisseur de nom de domaine (DNS). Pour cela, nous avons créé une zone hébergée pour le domaine et nous avons créé un enregistrement de type « CNAME » dont le nom est celui de notre compartiment S3 (sinon ça ne va pas fonctionner). Maintenant, notre application est accessible depuis l'URL suivante :

⇒ <http://bill-yourself.oofreelance.com/>

Dans les modules suivants, nous ajouterons des fonctionnalités à notre application en faisant appel à des APIs RESTful à l'aide de TypeScript afin de la rendre dynamique. Commençons par la gestion des utilisateurs.

## III.5.2 Gestion des utilisateurs

### III.5.2.1 Présentation

Dans ce module, nous créerons un groupe d'utilisateurs AWS Cognito pour gérer les comptes des utilisateurs de notre application.

Lorsque les utilisateurs accèdent à notre site Web, ils devront d'abord créer un nouveau compte utilisateur. Une fois qu'ils ont soumis leur inscription, AWS Cognito leur envoie un email de confirmation avec un code de vérification pour valider leurs inscriptions à notre plateforme.

Pour se connecter, les utilisateurs doivent saisir leur email et leur mot de passe. Une fonction JavaScript, qui contient les informations sur l'identité de l'utilisateur, communique alors avec AWS Cognito. Ces informations seront utilisées dans le prochain module pour procéder à l'authentification auprès de l'API RESTful que nous allons créer avec AWS API Gateway.

### III.5.2.2 Création d'un groupe d'utilisateurs Amazon Cognito

AWS Cognito propose la possibilité d'utiliser les groupes d'utilisateurs Cognito pour ajouter des fonctionnalités d'inscription et de connexion aux applications.

A cet égard, nous avons créé un groupe d'utilisateurs et nous l'avons affecté un nom depuis la console AWS. Puis, nous avons ajouté une nouvelle application cliente associée à ce groupe sur Cognito et nous l'avons définie dans l'application développée comme suit :

```
const
  POOL_DATA
= {
    UserPoolId: 'us-east-2_qYpPA9u9Y',
    ClientId: '1epsdgvrh9avu392lrog3o2kbt' };
```

A cette étape, nos fonctionnalités d'authentification fonctionnent parfaitement. Passons maintenant au développement des autres parties backend de l'application.

## III.5.3 Intégration d'un backend sans serveur

Dans ce module, nous allons utiliser AWS Lambda et AWS DynamoDB pour créer un processus backend qui traitera les requêtes destinées à notre application Web. En effet, le



code TypeScript exécuté dans la partie frontale devra invoquer une fonction Lambda chaque fois qu'un utilisateur fait une requête HTTP. Cette fonction Lambda enregistrera la requête dans une table DynamoDB, puis répondra à l'application frontend.

Pour ce faire, nous avons créé tout d'abord une table intitulée « bill-yourself » sur DynamoDB. Cette table contient une clé de partition qui est « bill-ID », c'est-à-dire l'identifiant unique pour chaque facture enregistrée dans la table « bill-yourself ». De plus, elle contient les différents autres attributs qui définissent une facture.

Chaque fonction Lambda dispose d'un rôle IAM qui lui est associé. Ce rôle définit les autres services AWS avec lesquels la fonction est autorisée à interagir. Dans ce cadre, nous avons créé un rôle IAM qui octroie à nos fonctions Lambda l'autorisation d'écrire dans les logs d'AWS CloudWatch et d'accéder à notre table DynamoDB.

Dans notre cas, la partie backend de l'application nécessite trois fonctions Lambda que nous avons développées en NodeJS et qui sont les suivants :

- Une fonction « POST » qui ajoute des éléments à la base de données. Cette fonction est appelée à chaque fois qu'un utilisateur veut créer une nouvelle facture et la sauvegarder dans la base de données.
- Une fonction « DELETE » qui supprime des éléments de la base de données. Cette fonction est appelée à chaque fois qu'un utilisateur veut supprimer une facture de la base de données.
- Une fonction « GET » qui prend des éléments de la base de données. Cette fonction est appelée à chaque fois qu'un utilisateur veut afficher la liste de ses factures sauvegardées dans la base de données.

#### **III.5.4 Déploiement d'une API RESTful**

AWS Lambda s'exécute à la suite d'une requête HTTP. Ces requêtes sont traitées par une fonction centrale qui est AWS API Gateway. Ce service expose un point de terminaison HTTP pouvant être invoqué depuis le navigateur des utilisateurs.

Le trio API Gateway, Lambda et DynamoDB s'intègre ensemble dans le but d'établir un backend entièrement fonctionnel pour notre application Web.

API Gateway est accessible sur Internet en public. Pour cela, nous l'avons sécurisée à l'aide du groupe d'utilisateurs AWS Cognito créé dans le module précédent de manière que seulement les utilisateurs inscrits dans le groupe d'utilisateurs de notre application puissent utiliser cette API et fassent des requêtes HTTP vers notre base de données DynamoDB.

Finalement, notre API est prête à être déployée. Son déploiement se fait à partir de la console AWS API Gateway en cliquant sur l'action « Déploiement d'API ». Maintenant, il est possible de faire appel à cette API à partir de la partie frontale en y incluant son URL d'appel.

### **III.6 Automatisation des tests de développement**

#### **III.6.1 Tests fonctionnels**

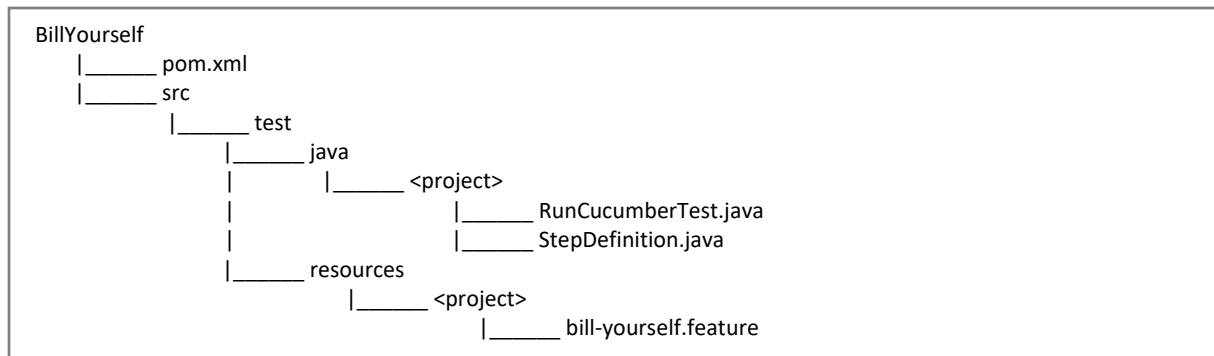
Un test fonctionnel est un test dont l'objectif est d'analyser le produit fini et de déterminer s'il répond bien aux attentes sur le plan technique. Les tests fonctionnels peuvent être effectués durant la phase de développement, au cas par cas pour des sections spécifiques développées, ou au terme du développement du projet, une fois qu'il est complet.

Afin d'automatiser les tests fonctionnels, nous allons utiliser le processus BDD (Behavior Driven Development) qui automatise ces tests, afin d'améliorer la qualité de développement et réduire les coûts de maintenance, et nous allons utiliser Cucumber pour faire ça en suivant les étapes suivantes :

##### **III.6.1.1 Création du projet**

Pour commencer, il faut tout d'abord installer JAVA, Maven et un IDE comme Eclipse. Puis, il faut intégrer Cucumber en installant son plugin sur Eclipse.

Ensuite, nous utilisons Maven pour importer des dépendances externes. Nous commençons par créer un nouveau projet Maven, qui créera automatiquement certains des répertoires et fichiers dont nous aurons besoin. Le projet doit avoir la forme suivante :



Le fichier « pom.xml » contient les dépendances du projet. Les fonctionnalités sont définies dans les fichiers « .feature », stockés dans le répertoire « src/test/resources ». Les fichiers de ce dossier de type « .feature » sont automatiquement reconnus comme des fichiers de fonctionnalités. Chaque fichier de fonctionnalité décrit une entité unique ou une partie d'une fonctionnalité. Le fichier « RunCucumberTest.java » est un « runner » qui exécute les tests. Quant au fichier « StepDefnition.java », il contient la définition des différentes étapes en JAVA.

### III.6.1.2 Ecriture des scénarios

Les scénarios définissent des exemples du comportement attendu pour tester les fonctionnalités de l'application. Un scénario est défini en utilisant les mots-clés (Given, When et Then) suivis d'une étape chacun. L'étape est, par la suite, associée à une définition d'étape, qui mappe l'étape de texte écrit en langage Gherkin [13] au code de programmation. L'exemple ci-dessous définit un scénario qui teste la fonctionnalité d'authentification :

```

Feature: Login to BillYourself website
Scenario: Enter Email and Password and Login
Given User need to be on BillYourself Sign In page
When User enters Email
And User enters Password
Then User checks user Email is present
And User checks user Password is present
And User login
  
```

- « Given » précède le texte définissant le contexte, la condition préalable du système.
- « When » précède le texte définissant une action.
- « Then » précède le texte définissant le résultat de l'action ou bien le résultat attendu.

### III.6.1.3 Définition des étapes en JAVA

Cette partie consiste à traduire les étapes écrites en langage Gherkin en code de programmation JAVA. Les étapes sont définies comme suit :

```
@Given("^User need to be on BillYourself Sign In page$")
public void User_need_to_be_on_BillYourself_signin_page() {
    System.setProperty("webdriver.chrome.driver",
        "C:\\Users\\wbensalah\\Desktop\\Cucumber\\chromedriver_win32\\chromedriver.exe");
    driver = new ChromeDriver();
    driver.get("http://bill-yourself.oofreelance.com/"); }

@When("^User enters Email$")
public void User_enters_Email() {
    driver.findElement(By.xpath("//input[@id=\"username\"]")).sendKeys(
        "wael.bensalah@supcom.tn"); }

@And("^User enters Password$")
public void User_enters_Password() {
    driver.findElement(By.xpath("//input[@id=\"password\"]")).sendKeys("*****"); }

@Then("^User checks user Email is present$")
public void User_checks_user_Email_is_present() {
    String userEmailActual = driver.findElement(
        By.xpath("//input[@id=\"username\"]")).getAttribute("value");
    Assert.assertEquals("wael.bensalah@supcom.tn", userEmailActual); }

@And("^User checks user Password is present$")
public void User_checks_user_Password_is_present() {
    String userPasswordActual = driver.findElement(
        By.xpath("//input[@id=\"password\"]")).getAttribute("value");
    Assert.assertEquals("*****", userPasswordActual); }

@And("^User login$")
public void User_login() throws InterruptedException {
    driver.findElement(By.id("login_btn")).click(); }
```

Nous avons défini l'URL de l'application en utilisant « ChromeDriver » que nous avons installé. Puis, nous avons utilisé la méthode « sendKeys » pour envoyer les clés de tests aux éléments adéquats qui sont définis avec leurs « xpath » ou bien leurs « id ».

Nous répétons ces étapes pour les différentes fonctionnalités de l'application (enregistrement, création de factures, génération en PDF, suppression, etc.) jusqu'à la couvrir entièrement avec ces tests fonctionnels.

#### III.6.1.4 Exécution de tests et génération de rapports

Afin d'exécuter ces tests, nous faisons, tout d'abord, un commit de tout le projet vers GitHub. Ensuite, nous faisons son build avec Jenkins en faisant appel au fichier « pom.xml ». Enfin, nous allons générer le rapport des tests en utilisant le plugin « Cucumber reports » après avoir l'installé en utilisant le gestionnaire des plugins de Jenkins :

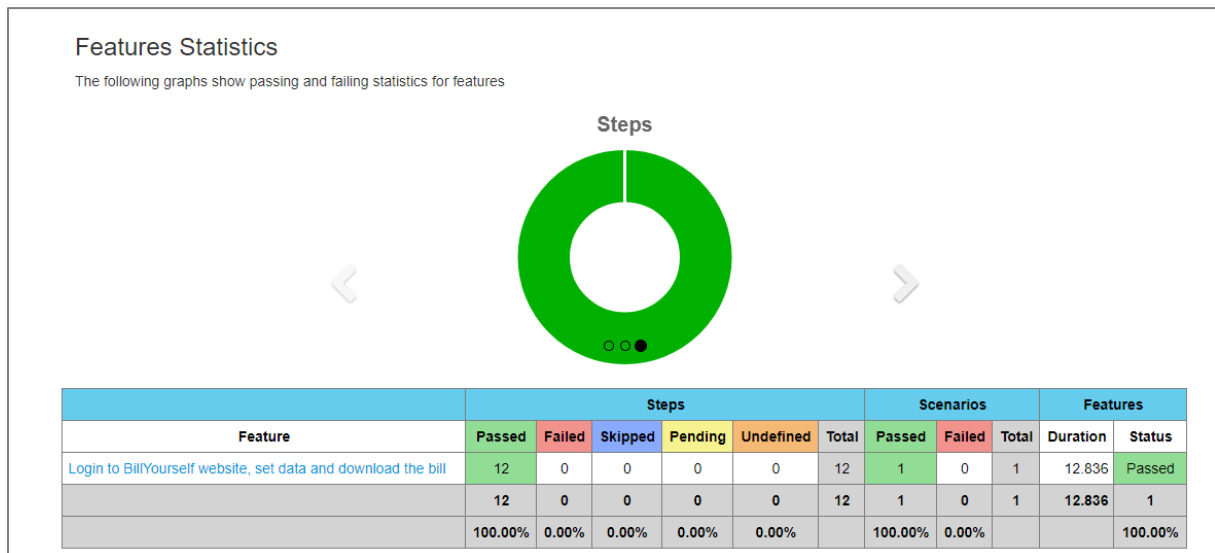


Figure 28: Le rapport des tests fonctionnels de Cucumber

Le graphe ci-dessus montre les statistiques de réussite et d'échec pour les fonctionnalités testées, à savoir l'authentification à l'application, la saisie des données de la facture et son téléchargement en format PDF. Nous constatons que tous les tests sont passés et ont réussi à 100%.

#### III.6.2 Tests de performances

Un test de performance est un test dont l'objectif est de déterminer la performance d'un système informatique en mesurant son temps de réponse en fonction de sa sollicitation. Cette capacité est très importante pour tester la stabilité des applications.

##### III.6.2.1 Configuration de JMeter

Afin d'automatiser les tests de performances, nous allons installer et puis utiliser JMeter que nous configurons en indiquant l'URL de notre application Web ainsi que la requête HTTP qui est « GET » dans notre cas, tel qu'il est indiqué dans la figure ci-dessous.

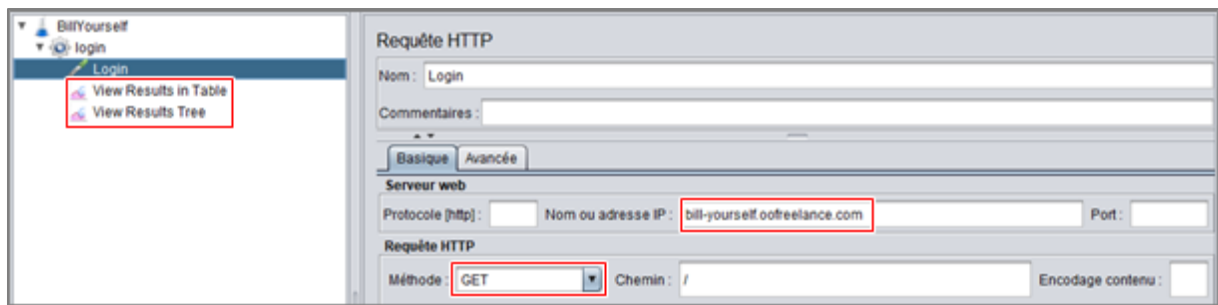


Figure 29: Configuration JMeter

A ce stade, nous pouvons enregistrer cette configuration et exécuter des tests de performance manuellement en cliquant sur le bouton « Run » et puis visualiser les résultats sous forme de table ou d'arbre. Cependant, notre objectif est d'automatiser ces tests et ceci en intégrant JMeter avec Jenkins.

### III.6.2.2 Exécution de tests et génération de courbes

Afin d'exécuter les tests de performances automatiquement avec Jenkins, nous installons, tout d'abord, le plugin « Performance Plugin » en utilisant le gestionnaire des plugins de Jenkins. Ensuite, sous la section « Build » du projet, nous choisissons « Run performance tests » et nous indiquons le chemin du fichier que nous avons enregistré avec JMeter. Enfin, nous faisons le build du projet et nous générons les courbes des tests présentées ci-dessous.

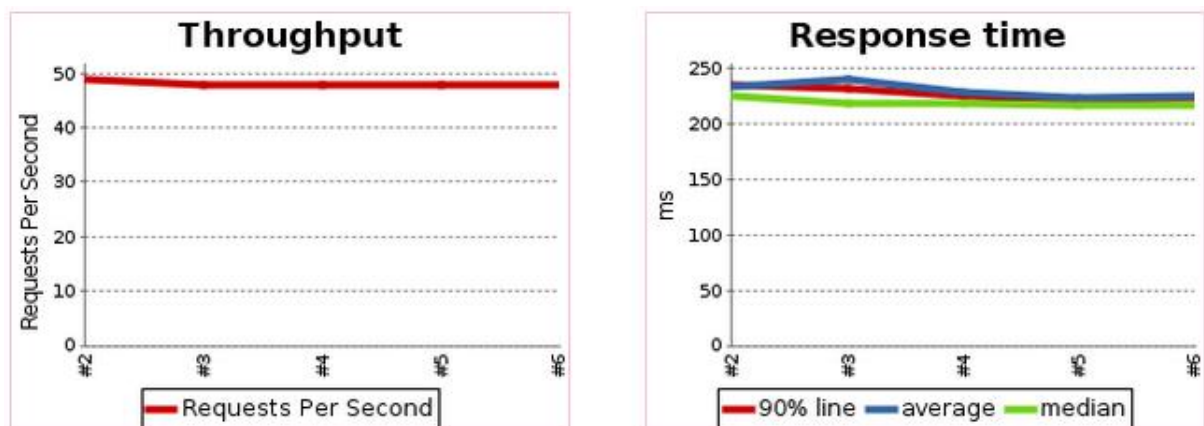


Figure 30: Requête/s et temps de réponses des tests de performances

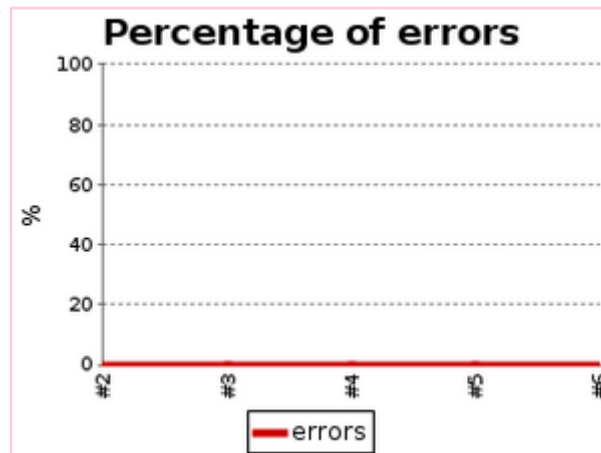


Figure 31: Taux d'erreurs des tests de performances

L'intégration de JMeter avec Jenkins permet de déclencher les tests de performance à chaque version du logiciel, afin de mieux comprendre si l'application est stable sous une certaine charge. L'exécution de tests de performances dans chaque génération peut nous aider à déterminer si les modifications récentes causent des problèmes, s'il existe une dégradation plus progressive des performances du système ou si notre système est capable de gérer sa charge de trafic de façon optimale.

En effectuant 5 tests de performances, les deux premières courbes montrent que lorsque nous avons environ 50 requêtes par seconde, le temps de réponse est resté quasiment stable (entre 220 ms et 240 ms). Nous pouvons remarquer aussi une convergence vers la valeur 220 ms dans les deux derniers tests (#5 et #6). La troisième courbe montre l'absence des erreurs au cours des 5 tests.

### III.7 Implémentation des workflows CI/CD

La CI/CD repose sur la mise en place d'une brique logicielle permettant l'automatisation des tâches de production logicielle (compilation, tests fonctionnels, tests de performances, validation du produit, déploiement etc.). À chaque changement du code, cette brique logicielle va exécuter un ensemble de tâches pour intégrer les changements et déployer la nouvelle version du code en continu.

Dans ce qui suit, nous allons mettre en œuvre un environnement d'intégration continue, d'exploiter les fonctionnalités de Jenkins et des différents outils qui constituent l'usine logicielle.

### III.7.1 CI/CD de la partie frontend

L'objectif de cette partie est d'intégrer l'approche CI/CD au processus de développement de la partie frontale de l'application, qui est développée en Angular 5.

Pour commencer, nous avons créé un projet sur GitHub en intégrant des « webhooks » pour notre serveur Jenkins comme expliqué dans la partie [II.3.2.1]. Ensuite, nous avons créé un nouveau projet sur Jenkins et nous l'avons configuré en suivant les étapes suivantes :

- Sous la section « Gestion du code source », nous avons indiqué l'URL du référentiel Git :



**Gestion de code source**

☐ Aucune

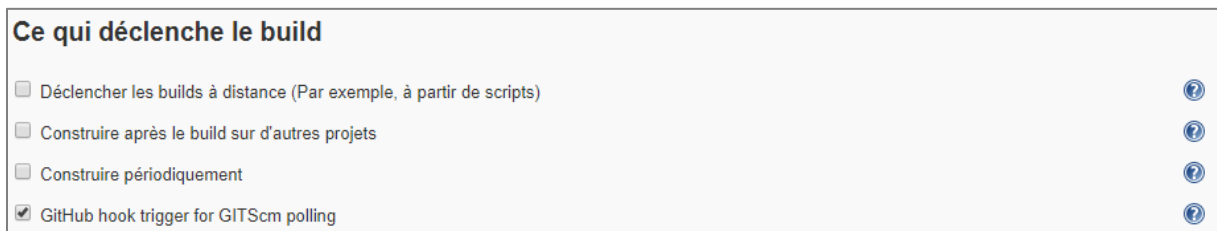
☒ Git

Repositories

Repository URL

Figure 32: Frontend : Gestion du code source

- Sous la section « Ce qui déclenche le build », nous avons sélectionné l'option « GitHub hook trigger for GITScm polling » afin de déclencher le build du projet automatiquement après chaque nouveau commit sur GitHub :



**Ce qui déclenche le build**

☐ Déclencher les builds à distance (Par exemple, à partir de scripts)

☐ Construire après le build sur d'autres projets

☐ Construire périodiquement

☒ GitHub hook trigger for GITScm polling

Figure 33: Frontend : Ce qui déclenche le build

- Sous la section « Environnements de Build », nous avons ajouté le répertoire de NodeJS :



**Environnements de Build**

☒ Ajouter le répertoire bin/ de Node/npm au PATH

NodeJS Installation

Figure 34: Frontend : Environnement de build



- Sous la section « Build », nous avons lancé une analyse avec « SonarQube Scanner » afin de mesurer la qualité de notre code. Pour cela, il faut ajouter les propriétés d'analyses suivantes :

The screenshot shows the Jenkins configuration interface for a build step titled "Lancer une analyse avec SonarQube Scanner". The configuration includes the following fields:

- Tâche à lancer:** A text input field.
- JDK:** A dropdown menu currently set to "(Hérite du job)".
- Le JDK à utiliser pour cette analyse SonarQube:** A label indicating the purpose of the selected JDK.
- Chemin vers les propriétés du projet:** A text input field.
- Propriétés de l'analyse:** A text area containing the following properties:
 

```
#Required metadata
sonar.projectKey=BillYourself
sonar.projectName=BillYourself
sonar.projectVersion=1.0

#Path to source directory
sonar.sources=/var/lib/jenkins/workspace/$JOB_NAME
```

Figure 35 : Frontend : Lancer une analyse avec SonarQube

- Sous cette même section, nous avons ajouté une étape « Exécuter un script Shell » et nous avons ajouté les commandes adéquates pour installer les packages requis, construire le projet en mode « production » dans le dossier « dist/BillYourself » et synchroniser le contenu de ce dossier avec celui du compartiment S3 « bill-yourself » comme indiqué dans la figure ci-dessous.

The screenshot shows the Jenkins configuration interface for a build step titled "Exécuter un script shell". The configuration includes the following field:

- Commande:** A text area containing the following shell commands:
 

```
npm install
ng build --prod
aws s3 sync dist/BillYourself s3://bill-yourself.oofreelance.com/
```

Figure 36 : Frontend : Exécuter un script

- Sous la section « Actions à la suite du build », nous avons ajouté « Slack Notifications » et coché « Notify Success » pour notifier les autres membres de l'équipe lorsqu'un build a réussi.

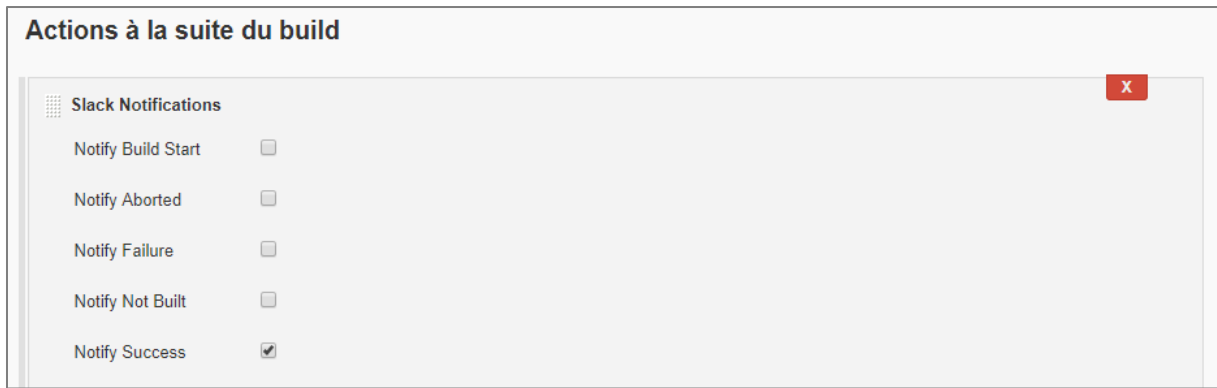


Figure 37 : Frontend : Notifier via Slack

### III.7.2 CI/CD de la partie backend

Dans cette partie, notre objectif est l'automatisation du build et du déploiement de la partie backend de l'application qui se base sur les trois fonctions Lambda créées précédemment. Pour ce faire, nous avons installé le plugin Jenkins nécessaire au déploiement des fonctions Lambda qui est « AWS Lambda Plugin » en utilisant le gestionnaire des plugins de Jenkins.

Par la suite, nous avons créé trois projets sur GitHub chacun pour une fonction Lambda sur GitHub en intégrant des « webhooks » pour notre serveur Jenkins. Ensuite, nous avons créé trois nouveaux projets sur Jenkins et nous avons configuré chacun en suivant les étapes suivantes :

- Sous la section « Gestion du code source », nous avons indiqué l'URL du référentiel Git :



Figure 38 : Backend : Gestion de code source

- Sous la section « Ce qui déclenche le build », nous avons sélectionné l'option « GitHub hook trigger for GITScm polling » afin de déclencher le build du projet automatiquement après chaque nouveau commit sur GitHub :

**Ce qui déclenche le build**

☐ Déclencher les builds à distance (Par exemple, à partir de scripts)

☐ Construire après le build sur d'autres projets

☐ Construire périodiquement

☒ GitHub hook trigger for GITScm polling

Figure 39 : Backend : Ce qui déclenche le build

- Sous la section « Environnements de Build », nous avons ajouté le répertoire de NodeJS :

**Environnements de Build**

☒ Ajouter le répertoire bin/ de Node/npm au PATH

NodeJS Installation      NodeJS

Figure 40 : Backend : Environnement de build

- Sous la section « Build », nous avons ajouté une étape « Exécuter un script Shell » et nous avons ajouté les commandes pour installer les packages requis et pour créer le bundle des fonctions Lambda :

**Build**

Exécuter un script shell

Commande `npm install`  
`grunt lambda_package`

Figure 41 : Backend : Build du projet

- Sous cette même section, nous avons ajouté une étape de déploiement AWS Lambda, en entrant les informations d'identification AWS, la région AWS, le nom et la description de la fonction Lambda, le rôle IAM utilisé ainsi que l'emplacement de l'artefact sur « dist/ », car c'est là que Grunt génère le bundle des fonctions Lambda, comme montée dans la figure ci-dessous.

Deploy AWS Lambda functions

☐ Use instance credentials

AWS Access Key Id

AWS Secret Key

AWS Region: us-east-2

Function Name: # Lambda Function Name

Description: # Lambda Function Description

Role: arn:aws:iam::890472922539:role/LambdaAdministratorAccess

Artifact Location (zip, directory, s3): dist/by-get-data.zip

Figure 42 : Backend : Déploiement AWS Lambda

A cette étape, chaque nouveau commit sur GitHub, un build automatique se déclenche pour mettre à jour le code exécuté par la fonction Lambda concernée.

### III.8 Conclusion

Ce dernier chapitre était consacré pour la réalisation des différentes tâches du projet. Premièrement, nous avons réussi à créer notre infrastructure dans le cloud AWS d'une manière automatisée. Deuxièmement, nous avons mis en place toute une usine logicielle sur cette infrastructure d'une manière automatisée aussi. Troisièmement, nous avons mis en place un serveur de monitoring pour superviser cet environnement de travail. Quatrièmement, nous avons développé une application Web en mode SaaS, sans serveur en utilisant les différents microservices AWS adéquats. Cinquièmement, nous avons automatisé les tests de développement de l'application développée. Finalement, nous avons intégré les workflows CI/CD pour l'intégration et la livraison continue de notre application.

# ***Conclusion générale***

Adopter l'approche DevOps n'est pas un sujet de prospective mais bien une réalité pour les entreprises, surtout que les géants du Web comme Amazon et Google la trouvent efficace et fiable. En effet, les entreprises doivent désormais comprendre que le choix architectural est un enjeu très important qui conditionne la performance de leurs activités.

Ce projet constitue un pas en avant vers une agilité de bout en bout. L'objectif ultime est issu d'une pure volonté stratégique, vise à faire évoluer son organisation, mais aussi à apporter des nouveaux projets à l'entreprise plus robustes et plus performants.

Pour mener à terme ce projet, nous avons procédé par une suite d'étapes bien étudiées : Nous étions responsables de la mise en place de l'architecture de la nouvelle solution ainsi que sa conception, sa réalisation et les tests nécessaires. En se basant sur une infrastructure dans le cloud, une usine logicielle automatisée et une architecture en microservices sans serveur, nous avons résolu plusieurs problèmes présentés par la méthode traditionnelle de développement logiciel.

Dans le présent rapport, nous avons détaillé les étapes par lesquelles nous sommes passés pour réaliser ce projet. Afin d'aboutir à ce résultat, nous avons tout d'abord commencé par présenter le cadre général de notre travail. Ensuite, nous avons expliqué la problématique traitée par notre projet. Puis, nous avons présenté les concepts clés liés à notre solution. En outre, nous avons abordé la modélisation et la conception de cette solution. Finalement, nous avons passé à la réalisation des différentes tâches.

Durant ce projet, nous avons été confrontés à plusieurs problèmes et obstacles. En effet, nous avons fait face aux défis exigés par l'architecture microservices comme la communication entre les différents microservices.

En perspective, nous proposons de mieux sécuriser l'application en utilisant HTTPS et de l'enrichir en s'intéressant à certains points : Nous envisageons d'ajouter d'autres fonctionnalités qui seront utiles pour l'utilisateur et nous envisageons aussi d'améliorer le design pour que notre application soit plus élégante et attirante.

# Webographie

- [1] Amazon AWS Documentation <https://docs.aws.amazon.com>
- [2] Git Documentation <https://git-scm.com/doc>
- [3] Jenkins Documentation <https://jenkins.io/doc/>
- [4] SonarQube Documentation <https://docs.sonarqube.org>
- [5] Terraform Documentation <https://www.terraform.io/docs/>
- [6] Ansible Documentation <https://docs.ansible.com/>
- [7] Elastic Stack Documentation <https://www.elastic.co/guide>
- [8] Angular Documentation <https://angular.io/docs>
- [9] React Documentation <https://reactjs.org/docs>
- [10] Cucumber Documentation <http://docs.cucumber.io>
- [11] JMeter Documentation <https://jmeter.apache.org/>
- [12] Slack <https://www.slack.com> (Consulté le 07/07/2018)
- [13] Gherkin <https://docs.cucumber.io/gherkin/reference> (Consulté le 05/08/2018)
- [14] Gwenaël Perier « Le mouvement DevOps » <https://blog.clever-age.com/fr/2012/04/17/le-mouvement-devops/> (Consulté le 30/04/2018)
- [15] CA Technologies « DevOps and Cloud Computing » <https://www.ca.com/us/modern-software-factory/content/devops-and-cloud-computing-exploiting-the-synergy-for-business-advantage.html> (Consulté le 02/05/2018)
- [16] MuleSoft « Microservices and DevOps : Better together » <https://www.mulesoft.com/resources/api/microservices-devops-better-together> (Consulté le 22/05/2018)

- [17] Mike Roberts « Serverless Architectures » <https://martinfowler.com/articles/serverless.html> (Consulté le 30/05/2018)
- [18] Shawn McKay « Comparing Performance of Blaze, React, Angular and Angular 2 » <http://tiny.cc/7kclly> (Consulté le 03/08/2018)

---

# ***Annexes***

---



## Annexe A : Mise en place d'une usine logicielle automatisée avec Ansible

### ❖ Le fichier « software\_factory.yml »

```
---  
  
- hosts: EC2 public IPv4 @  
  become: true  
  tasks:  
  
    #Git  
    - name: Install Git.  
  
    #Jenkins  
    - name: Ensure dependencies are installed.  
      apt:  
        name:  
        - curl  
        - apt-transport-https  
        state: present  
  
    - name: Add Jenkins apt repository key.  
      apt_key:  
        url: "https://pkg.jenkins.io/debian/jenkins.io.key"  
        state: present  
  
    - name: Add Jenkins apt repository.  
      apt_repository:  
        repo: "deb https://pkg.jenkins.io/debian binary/"  
        state: present  
        update_cache: yes  
  
    - name: Download specific Jenkins version.  
      get_url:  
        url: "https://pkg.jenkins.io/debian/binary/jenkins_2.127_all.deb"  
        dest: "/tmp/jenkins_2.127_all.deb"  
  
    - name: Check if we downloaded a specific version of Jenkins.  
      stat:  
        path: "/tmp/jenkins_2.127_all.deb"  
        register: specific_version  
  
    - name: Install our specific version of Jenkins.  
      apt:  
        deb: "/tmp/jenkins_2.127_all.deb"  
        state: present
```

#SonarRunner

- name: Download and expand sonar-runner.

unarchive:

src: "http://repo1.maven.org/maven2/org/codehaus/sonar/runner/sonar-runner-dist/2.3/sonar-runner-dist-2.3.zip"

dest: "/home/ubuntu"

copy: no

creates: /usr/local/sonar-runner/bin/sonar-runner

- name: Move sonar-runner into place.

shell: >

mv /home/ubuntu/sonar-runner-2.3 /home/ubuntu/sonar-runner

creates=/home/ubuntu/sonar-runner/bin/sonar-runner

#SonarQube

- name: Download Sonar.

get\_url:

url: "https://sonarsource.bintray.com/Distribution/sonarqube/sonarqube-7.2.zip"

dest: "/root/sonarqube-7.2.zip"

validate\_certs: yes

- name: Unzip Sonar.

unarchive:

src: "/root/sonarqube-7.2.zip"

dest: /home/ubuntu

copy: no

creates: /home/ubuntu/sonar/COPYING

- name: Move Sonar into place.

shell: >

mv /home/ubuntu/sonarqube-7.2 /home/ubuntu/sonar

creates=/home/ubuntu/sonar/COPYING

- name: Symlink sonar bin.

file:

src: /home/ubuntu/sonar/bin/linux-x86-64/sonar.sh

dest: /usr/bin/sonar

state: link

register: sonar\_symlink

- name: Add sonar as init script for service management.

file:

src: /home/ubuntu/sonar/bin/linux-x86-64/sonar.sh

dest: /etc/init.d/sonar

state: link

## Annexe B : Installation et configuration de Nginx pour créer un certificat SSL

### ❖ Le fichier « Nginx.yml »

```
---
- hosts: EC2 public IPv4 @
  become: true
  tasks:

#Nginx
- name: Install Nginx.
  apt:
    name=nginx
    update_cache=yes
    state=latest
  tags:
    - apt
    - setup

- name: Delete default configuration.
  shell: >
    sudo rm /etc/nginx/sites-enabled/default

- name: Nginx site for Jenkins.
  template: src=jenkins-nginx.j2 dest=/etc/nginx/sites-available/jenkins
  sudo: yes

- name: Enable the Nginx jenkins site.
  file: src=/etc/nginx/sites-available/jenkins dest=/etc/nginx/sites-enabled/jenkins
    state=link
  sudo: yes

- name: Start Nginx.
  service: name=nginx state=started
  sudo: yes
```

### ❖ Le fichier « jenkins-nginx.j2 »

La première section indique au serveur Nginx d'écouter les requêtes qui arrivent sur le port 80 (HTTP par défaut) et de les rediriger vers HTTPS.

La deuxième section contient les paramètres SSL. Les fichiers « cert.crt » et « cert.key » reflètent l'emplacement où nous avons créé notre certificat SSL.

La troisième section contient la configuration du proxy. Ce dernier prend essentiellement toutes les demandes entrantes sur le port 8080 et les redirige vers le port 443 dédié au protocole HTTPS.

```
server
{
    listen 80;
    return 301 https://$host$request_uri;
}

server {

    listen 443;
    server_name localhost;

    ssl_certificate      /etc/nginx/cert.crt;
    ssl_certificate_key  /etc/nginx/cert.key;

    ssl on;
    ssl_session_cache  builtin:1000  shared:SSL:10m;
    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers  HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    ssl_prefer_server_ciphers on;

    access_log    /var/log/nginx/jenkins.access.log;

    location / {

        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto $scheme;

        # Fix the "It appears that your reverse proxy set up is broken" error.
        proxy_pass    http://localhost:8080;
        proxy_read_timeout 90;

        proxy_redirect    http://localhost:8080 https://localhost;
    }
}
```

Pour exécuter cette configuration, il faut utiliser la commande suivante :

```
$ ansible-playbook Nginx.yml
```

## Annexe C : Installation de la pile ELK & FileBeat

Pour mettre en place le serveur de monitoring ELK, notre système doit répondre aux caractéristiques suivantes :

- ✓ Possesseur : multi-core recommandé, au moins 2 GHZ.
- ✓ Mémoire : au moins 4 GHZ de mémoire libre pour les services et les couches de système de fichiers.
- ✓ Environnement JAVA : Java 8+
- ✓ Espace disque : 10 GO au minimum.

### ❖ Installation de l'environnement JAVA

La pile ELK se base sur le langage Java. On installe donc Oracle Java 8 sur notre machine virtuelle Ubuntu 14 en utilisant les lignes de commandes suivantes :

```
$ add-apt-repository ppa:webupd8team/java  
$ apt-get update  
$ apt-get install oracle-java8-installer
```

Ensuite les variables d'environnement par default sont définies en utilisant la commande :

```
$ apt-get install oracle-java8-set-default
```

### ❖ Installation et configuration de ElasticSearch

ElasticSearch peut être installé avec un gestionnaire de paquets en ajoutant la liste des sources de paquets d'Elastic. Créons la liste source et installons ElasticSearch :

```
$ wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -  
$ sudo apt-get update  
$ sudo apt-get install elasticsearch
```

ElasticSearch est maintenant installé. Modifions alors sa configuration :

```
$ sudo vi /etc/elasticsearch/elasticsearch.yml
```

Dans ce fichier, il faut décommenter et modifier les lignes suivantes :

- ✓ cluster.name: DevOps
- ✓ node.name: SF\_Logs
- ✓ network.host: 0.0.0.0
- ✓ http.port: 9200

Enfin, il faut activer le service en utilisant la commande suivante :

```
$ sudo service elasticsearch start
```

### ❖ Installation et configuration de Kibana

Kibana peut être installé avec un gestionnaire de paquets en ajoutant la liste des sources de paquets d'Elastic. Créons la liste source et installons Kibana :

```
$ echo "deb http://packages.elastic.co/kibana/4.5/debian stable main" | sudo tee -a  
/etc/apt/sources.list.d/kibana-4.5.x.list  
$ sudo apt-get update  
$ sudo apt-get install kibana
```

Kibana est maintenant installé. Modifions alors sa configuration :

```
$ sudo vi /opt/kibana/config/kibana.yml
```

Dans ce fichier, il faut décommenter et modifier les lignes suivantes :

- ✓ elasticsearch.url: "localhost"
- ✓ server.host: "0.0.0.0"

Enfin, il faut activer le service en utilisant la commande suivante :

```
$ sudo service kibana start
```

## ❖ Installation de Logstash

Logstash peut être installé avec un gestionnaire de paquets en ajoutant la liste des sources de paquets d'Elastic. Créons la liste source et installons Logstash :

```
$ echo 'deb http://packages.elastic.co/logstash/2.2/debian stable main' | sudo tee  
/etc/apt/sources.list.d/logstash-2.2.x.list  
$ sudo apt-get update  
$ sudo apt-get install logstash
```

Ensuite, il faut activer le service en utilisant la commande suivante :

```
$ sudo service logstash start
```

## ❖ Installation de FileBeat

FileBeat peut être installé avec un gestionnaire de paquets en ajoutant la liste des sources de paquets d'Elastic. Créons la liste source et installons FileBeat :

```
$ echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | sudo tee -a  
/etc/apt/sources.list.d/elastic-6.x.list  
$ sudo apt-get update  
$ sudo apt-get install filebeat
```

Ensuite, il faut activer le service en utilisant la commande suivante :

```
$ sudo service filebeat start
```

## Annexe D : Les tarifs des services AWS utilisés dans la région us-east-2 (Ohio)

### ❖ AWS IAM

AWS IAM est service offert par Amazon sans frais supplémentaires. On est facturé uniquement pour l'utilisation d'autres services AWS par nos utilisateurs.

### ❖ AWS VPC

En créant un point de terminaison de VPC, La facturation se fait pour chaque heure pendant laquelle le point de terminaison de VPC est en service dans chaque zone de disponibilité. Des frais de traitement des données sont appliqués pour chaque Go traité, quelles que soient l'origine ou la destination du trafic. Chaque heure partielle consommée par le point de terminaison de VPC sera facturée comme une heure pleine.

Prix par point de terminaison de VPC par AZ (USD/heure)	Prix par Go de données traitées (USD)
0,01	0,01

### ❖ AWS EC2

Les instances à la demande permettent de payer la capacité de calcul à l'heure ou à la seconde (60 secondes minimum) sans engagement. De cette manière, le coût et la complexité de la planification, de l'achat et de la maintenance du matériel n'est pas à subir : L'ensemble de ces frais fixes, habituellement élevés, est transformé en des coûts variables bien moindres.

Type	CPU	ECU*	Mémoire (Go)	Utilisation de Linux/UNIX (USD/heure)
t2.micro	1	Variable	1	0,0116
t2.small	1	Variable	2	0,023
t2.medium	2	Variable	4	0,0464
t2.large	2	Variable	8	0,0928
m5.large	2	8	8	0,096
m5.xlarge	4	16	16	0,192



\*ECU (EC2 Compute Unit) : c'est une unité de calcul EC2 fournit la mesure relative de la puissance de traitement d'une instance EC2.

#### ❖ AWS S3

Avec AWS S3, on ne paye que ce qu'on utilise. Il n'y a pas de frais minimum. Une fois inscrits, les nouveaux clients AWS bénéficient, chaque mois pendant un an, de 5 Go de stockage Amazon S3 dans la classe de stockage Standard, de 20 000 requêtes « GET », de 2 000 requêtes « PUT » et de 15 Go de transfert de données sortantes gratuitement.

Tranche	Prix (USD/Go)
Première tranche de 50 To/mois	0,023
450 To suivants/mois	0,022
Plus de 500 To/mois	0,021

#### ❖ AWS Cognito

Avec AWS Cognito, on ne paye que ce qu'on utilise. Il n'y a pas de frais minimum et aucun engagement initial n'est requis. Les frais liés à la gestion des identités et à la synchronisation des données, ainsi que la tarification de ces options sont détaillés ci-dessous.

Pour utiliser AWS Cognito pour créer un pool d'utilisateurs, on doit payer uniquement sur la base des utilisateurs actifs mensuels (MAU). Un utilisateur est considéré comme étant un MAU s'il est associé, dans le mois civil, à une opération faisant intervenir l'identité, telle que l'inscription, la connexion, l'actualisation d'un jeton ou la modification du mot de passe. Les sessions suivantes et les utilisateurs inactifs de ce mois civil ne sont pas facturés.

Niveau de tarification (MAUs)	Tarif (USD)
Premiers 50 000	Gratuit
50 000 suivants	0,00550 USD
900 000 suivants	0,00460 USD
9 000 000 suivants	0,00325 USD
Supérieure à 10 000 000	0,00250 USD

### ❖ AWS API Gateway

Avec AWS API Gateway, on ne paye que lorsque les API sont en cours d'utilisation. Il n'y a pas de frais minimaux et aucun engagement initial n'est requis. Uniquement les appels d'API reçus et la quantité de données en transfert sortant sont à payer.

Le niveau gratuit d'API Gateway inclut un million d'appels d'API reçus par mois pendant 12 mois maximum. Si ce nombre d'appels est dépassé, les tarifs ci-dessous sont à appliquer.

Appels d'API	Coûts des transferts de données
3,50 USD par million d'appels d'API reçus	0,09 USD/Go pour les premiers 10 To 0,085 USD/Go pour les 40 To suivants 0,07 USD/Go pour les 100 To suivants 0,05 USD/Go pour les 350 To suivants

### ❖ AWS Lambda

Avec AWS Lambda, on paye uniquement en fonction de l'utilisation. En effet, le paiement se fait en fonction du nombre de requêtes pour les fonctions utilisées et pour la durée, c'est-à-dire le temps nécessaire à l'exécution des fonctions.

Lambda compte une requête chaque fois qu'il s'exécute en réponse à une notification d'événement ou un appel de demande, notamment les appels d'essai provenant de la console. Le paiement est en fonction du nombre total de requêtes dans l'ensemble des fonctions.

Mémoire (Mo)	Nombre de secondes offertes par mois	Tarif par 100 ms (USD)
128	3 200 000	0,000000208
256	1 600 000	0,000000417
384	1 066 667	0,000000625
448	914 286	0,000000729
512	800 000	0,000000834

### ❖ AWS DynamoDB

En créant une table AWS DynamoDB, on doit indiquer l'utilisation cible ainsi que les capacités minimale et maximale pour la mise à l'échelle automatique, ou bien le volume de capacité à réserver pour les lectures et écritures. Un tarif horaire fixe s'applique en fonction de la capacité allouée.

Type de débit alloué	Prix à l'heure	Performances
Unité de capacité d'écriture	0,00065 USD par unité de capacité d'écriture	1 unité de capacité d'écriture offre jusqu'à 3 600 écritures par heure
Unité de capacité de lecture	0,00013 USD par unité de capacité de lecture	1 unité de capacité de lecture offre jusqu'à 7 200 lectures par heure

### ❖ AWS Route 53

Avec AWS Route 53, on ne paye que ce qu'on utilise. Il n'y a pas de frais minimum. La facturation se fait selon le nombre des zones hébergées et le nombre des requêtes.

Zones hébergées	Requêtes standards
0,50 USD par zone hébergée/mois pour les 25 premières zones hébergées	0,400 USD par million de requêtes – premier milliard de requêtes/mois
0,10 USD par zone hébergée/mois pour chaque zone hébergée supplémentaire	0.200 USD par million de requêtes – au-delà d'un milliard de requêtes/mois

### ❖ AWS CloudWatch

Avec AWS CloudWatch, on ne paye que ce qu'on utilise. Il n'y a pas de frais minimum et aucun engagement initial n'est requis. La plupart des services AWS, par exemple EC2, S3, Lambda et Kinesis, fournissent automatiquement et gratuitement les métriques sur

CloudWatch. De nombreuses applications devraient pouvoir fonctionner dans ces limites de l'offre gratuite détaillée dans le tableau ci-dessous.

<b>Métriques</b>	Métriques de surveillance de base (fréquence de 5 min)  10 métriques de surveillance de base (fréquence de 1 min)  1 million de demandes API
<b>Tableau de bord</b>	3 tableaux de bord avec jusqu'à 50 métriques chacun par mois
<b>Alarmes</b>	10 alarmes (sans objet pour les alarmes haute résolution)
<b>Journaux</b>	5 Go de données (ingestion et stockage des archives)
<b>Événements</b>	Tous les événements sont inclus, sauf ceux personnalisés