Université de Strasbourg
UFR de mathématique et d'informatique
Université de Strasbourg

Master de Science, mention Informatique spécialité Science et Ingénierie des Réseaux, de l'Internet et des Systèmes

Mémoire de projet de Master, groupe n°2

ALGGOT

Adrien Roberty	Ludovic Muller	Oussema Tourki
roberty@etu.unistra.fr	ludovic.muller2@etu.unistra.fr	oussema.tourki@etu.unistra.fr
Govindaraj Vetrivel	Thomas Froeliger	Gabriel Poittevin
govindaraj.vetrivel@etu.unistra.fr	thomas.froeliger@etu.unistra.fr	gabriel.poittevin@etu.unistra.fr

ChaTalK, application de télécommunications sécurisées et centralisées

ChaTalK

Projet encadré et commandité par

Julien Montavont montavont@unistra.fr et Cristel Pelsser pelsser@unistra.fr

iCUBE

Journalistes Sans Papiers

8 janvier 2020

Table des matières

Table des figures

Résumé

Ceci est le mémoire de projet du groupe n°2 de projet de Master de la promotion 2018-2020 du Master SIRIS de l'Université de Strasbourg.
L'objectif de ce document est de présenter le projet dans son ensemble, avec ses problématiques, les solutions qui leur ont été apportées et les difficultés rencontrées au cours de sa réalisation.

Partie n^o 1

Introduction

1.1 Le projet

L'objectif de ce projet était de répondre à une série de problèmes rencontrés par l'organisation (fictive) Journalistes Sans Papiers. JSP souhaitait un moyen le plus sécurisé possible de permettre à ses membres de communiquer les uns avec les autres. La solution devait également supporter d'importants flux de données, et pouvoir fournir notamment un mode audio voire vidéo en plus du mode texte classique. Enfin, l'organisation souhaitait la possibilité de déployer à la volée le service sur des réseaux isolés, permettant à ses membres d'utiliser la solution de communication dans des situations coupées du réseau Internet et des serveurs centraux de l'organisation.

1.2 La solution

Notre équipe a conçu la solution ChaTalK afin de répondre aux besoin de l'entreprise. Il n'existait pas à notre connaissance de solution permettant les communication entièrement sécurisées de manière centralisée, dont un développement facile pour un néophyte puisse être possible. ChaTalK est une application Web 2.0, dont l'infrastructure repose sur des cluster Kubernetes sur des machines Ubuntu Server, le côté serveur de l'application est un ensemble de services Go et PostgresQL, et le côté client est une interface écrite en React JS pour la partie navigateur et en Kotlin pour l'application mobile.

Partie n^o 2

Réalisation technique

2.1 Descriptif général

Nous avons fait le choix de déployer des grappes Kubernetes [2] pour un certain nombre de raisons. Tout d'abord, Kubernetes est une technologie répandue et approuvée à très grande échelle, ce qui en fait une référence incontournable en terme de fiabilité pour architecturer une infrastructure devant répondre à un certain nombre de problématiques bien précises, telles que la redondance, la tolérance aux pannes et à la charge, etc. Ce qui correspond à la demande de nos clients.

En outre, découper notre projet en micro-services nous a permis de découper les tâches de manière plus fines et d'avancer parallèlement sur ces derniers. Chaque service est testé et compilé avant d'être conteneurisé automatiquement sous forme de conteneur Docker [3]. Ces derniers peuvent ainsi être déployés très facilement sur une infrastructure Kubernetes pour la grande instance de production, ainsi que sur une instance minimaliste avec l'aide de *docker-compose* [4], qui va simplement lancer une instance de chaque service. Cette dernière solution est pratique à la fois pour les développeurs pour faire tourner l'ensemble de l'application en local chez eux, ainsi que pour nos clients qui auront la possibilité de lancer une nouvelle instance minimaliste lors de leurs déplacements alors qu'ils n'ont pas de connexion à leur serveur principal pour continuer à communiquer.

Nous avons également choisi de faire des micro-services car notre application était supposée être modulaire. Nous n'avons pas retravaillé cette partie de l'architecture logicielle lorsque nous avons abandonné cet objectif, car il aurait été trop long de tout redévelopper..

Les machines virtuelles que nous avons ont été configurées avec l'aide d'Ansible [5], de même que la base des grappes Kubernetes avec l'aide de *Kubespray* [6], un Playbook Ansible, ce qui nous permet d'avoir un déploiement automatisé assez rapide. Un nouveau grappe complet peut être monté en une heure environ.

2.2 Gestion de flux de données très importants

L'utilisation de Kubernetes ainsi que la pile de surveillance que nous avons mis en place nous permettent de récupérer un certain nombre de métriques. Le fait d'avoir architecturé notre application sous forme de micro-services nous permet de monter ou diminuer le nombre d'instances d'un de ces services de manière indépendante avec l'aide des métriques que nous avons récupérées, ce qui permet d'adapter les ressources de nos grappes en fonction de la demande courante, et donc de pouvoir s'adapter à des flux de données très importants peu importe le service sollicité.

De plus, le bus de données permet d'éviter aux services d'avoir à gérer la notion de charge. En effet, c'est le bus de données qui temporise l'ensemble des requêtes en attente avec l'aide de *Nats-Streaming*

2. `https://kubernetes.io/fr/`
3. `https://www.docker.com/`
4. `https://docs.docker.com/compose/`
5. `https://www.ansible.com/`
6. `https://kubespray.io/#/`

(aka. stan)[1]

2.3 Tolérance aux pannes

Nous avons deux grappes Kubernetes répartis sur deux sites géographiquement distincts : l'un à Illkirch et l'autre à Esplanade. Si l'un des deux venait à tomber, l'autre sera là pour prendre la charge et répondre à la demande.

Nos grappes Kubernetes sont chacun constituées de trois nuds, permettant de mutualiser l'ensemble des ressources au sein d'une grappe. Si au sein d'une grappe on perd un nud, il en restera toujours deux qui seraient disponibles, soit la majorité ; Kubernetes peut donc continuer à instancier de nouvelles instances sans difficultés. Par contre si on venait à perdre un second nud au sein de cette grappe, il n'y aurait plus la majorité. Les services actuellement en place sur le dernier nud resteront accessibles, mais les services se trouvant sur les autres nuds ne pourront pas être déployés sur ce dernier nud comme cela pourrait être le cas dans le cas précédent.

Chaque grappe expose l'application depuis une adresse IP qui est assignée à l'un des nuds. Si le nud en question venait à tomber, cette adresse IP sera attribuée à un autre nud de la grappe qui est disponible. Ce point sera davantage détaillé dans la suite de ce mémoire.

En termes de maintenance, remplacer un service, par exemple dans le cadre d'une mise à jour, est invisible pour l'utilisateur final, car Kubernetes va d'abord instancier la nouvelle instance du micro-service et va arrêter l'ancienne seulement une fois que la nouvelle instance sera fonctionnelle ; c'est-à-dire que si une nouvelle instance n'arrive pas à se lancer correctement, l'ancienne instance, qui elle est fonctionnelle, continuera à répondre aux différentes demandes. De plus, il est possible d'avoir plusieurs instances d'un même service au sein d'une grappe pour avoir de la redondance et mieux résister aux pannes, ainsi que pour mieux gérer la charge.

Au niveau de l'application web, si la connexion au WebSocket plante, la connexion va se rétablir automatiquement ultérieurement, et ce de manière périodique toutes les deux secondes jusqu'à ce que la connexion s'effectue à nouveau (le temps que le serveur soit à nouveau accessible en cas d'une panne de forte envergure par exemple).

Du côté de la base de données, étant donné qu'on déploie des grappes de PostgreSQL avec Stolon qu'on verra plus tard dans ce mémoire, nous avons une redondance qui évite une perte de données. De plus ces données sont persistées sur le disque contrairement aux autres données qui n'ont pas besoin de persistance. Le site d'Esplanade héberge notre grappe Postgres maître. S'il tombe, alors seules les lectures pourront être possibles ce qui constitue un mode dégradé. Il en va de même avec notre bus de données, qui est fédéré entre les deux sites, avec une instance maître sur Esplanade.

2.4 Tolérance à la charge, redimensionnement automatique

Comme dit dans la partie précédente, il est possible d'avoir plusieurs instances d'un même service qui sont lancées au sein d'une même grappe. Il est possible de mettre en place des règles afin que ces derniers soient par exemple lancés de préférence sur des nuds différents. C'est par exemple le cas du contrôleur d'*ingress* qui a une instance lancée par nud.

Il est également possible de changer le nombre d'instances d'un service en fonction de certaines métriques, notamment les ressources utilisées telles que la RAM et le CPU, ou bien des métriques personnalisées récupérées par notre pile de surveillance en explorant des URL précises exposées par les services.

Si jamais notre application deviendrait très populaire, il serait toujours possible de rajouter de nouvelles machines au sein des grappes existantes, voire même ajouter des grappes entières.

1. `https://docs.nats.io/nats-streaming-concepts/intro`

2.5 Temps de réponse garantis

Nous ne garantissons pas spécialement de temps de réponse pour notre application. Néanmoins le fait qu'elle supporte bien la charge et que l'on peut faire évoluer le nombre d'instances des différents services en fonction de certaines métriques, on peut limiter le temps d'attente pour le traitement des différentes requêtes en cas de montée en charge.

Le fait d'avoir une tolérance aux pannes plutôt convenable nous aide également à limiter les temps de réponses : puisque les requêtes ont moins de chances d'échouer, il y aura moins de tentatives à effectuer avant d'accéder aux différentes ressources.

2.6 Hébergement sur deux sites

Nous n'avions aucune ressources à disposition, ni matérielles, ni financières pour être en mesure de répondre à la demande, excepté les machines en salles de TP. L'utilisation de ces dernières n'étaient malheureusement pas envisageables dans le cadre de ce projet étant donné que ces machines sont utilisées lors de différents TP et qu'il faudrait par conséquent réinstaller toute l'infrastructure à chaque séance de travail fois avoir un environnement maîtrisé.

Ludovic est donc allé voir différentes personnes au sein de l'Université afin de voir s'il était possible de venir avec son propre serveur 1U ou s'il était possible d'obtenir des machines virtuelles quelque part. Déployer un nouveau serveur au sein de l'Université est malheureusement trop complexe. Il a finalement pu trouver des machines virtuelles pour l'ensemble des groupes de projet à Illkirch ; c'est ainsi que nous avions eu nos quatre première machines virtuelles pour y déployer notre première grappe Kubernetes.

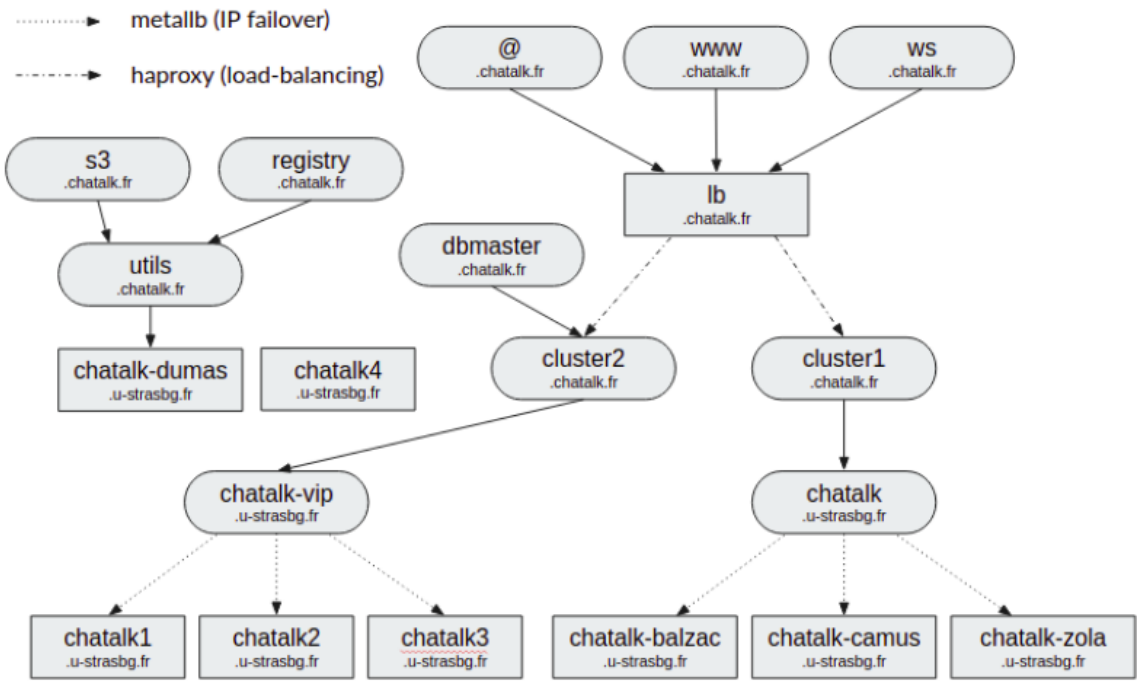
La première grappe située à Illkirch est constituée de trois machines virtuelles (`chatalk-balzac`, `chatalk-camus` et `chatalk-zola`) qui sont configurées pour être les trois nuds maîtres de cette grappe. Ces machines virtuelles sont dotées de 8vCPU, de 8Go de RAM, de 50Go de disque et ont pour système d'exploitation Ubuntu Server 18.04, qui correspond à la dernière LTS.

Une quatrième machine virtuelle (chatalk-dumas) est déployée sur ce site pour des fins utilitaires. Elle dispose des mêmes ressources que les trois autres machines virtuelles, sauf en termes de disque où il y a 75Go à sa disposition. Cette machine virtuelle est utilisée pour y héberger Minio (s3), qui est un stockage compatible S3, un type stockage qui a la possibilité de passer à l'échelle de manière plus efficace que les solutions classiques à base de montages distants. Ce stockage est utilisé pour héberger nos sauvegardes ainsi que ce qu'il faut pour initialiser une nouvelle instance de la base de données. Cette machine virtuelle est également utilisée pour héberger des exécuteurs *GitLab* qui nous permettent de tester, compiler et déployer notre application très rapidement en production. Cette machine virtuelle nous sert également pour héberger nos propres images Docker dans notre base de registre (*registry*).

Ludovic a pu obtenir dans un second temps quatre autres machines virtuelles à Esplanade pour former un second site. Les machines virtuelles chatalk1, chatalk2 et chatalk3 sont les trois nuds utilisés pour la seconde grappe Kubernetes, qui ont les mêmes ressources que la première grappe. La dernière machine virtuelle, chatalk4, n'a pas été utilisée pour le moment. Elle aurait pu nous permettre de redonder le stockage s3 par exemple, et par conséquence de s'assurer qu'il soit toujours accessible même si le site d'Illkirch venait à tomber ; hélas par manque de temps nous n'avons pas pu le faire.

Nous avons donc réussi à avoir toutes nos ressources au sein de l'Université de Strasbourg, et ce sur deux sites différents géographiquement (Illkirch et Esplanade), même si cela n'a pas forcément été très simple de les avoir. De surcroît, chaque demande (demande d'ouverture de ports, augmentation des ressources, demande d'IP publiques, ...) devait être faite par mail ou en personne, une partie d'entre elles étant remontées jusqu'au support informatique de l'Université, ce qui peut prendre du temps. Il n'était donc pas possible de faire des tests très poussés concernant l'extinction de machines virtuelles. En effet, il aurait fallu attendre que quelqu'un nous les rallume. Enfin, accéder aux machines virtuelles nécessite de passer par le réseau de l'Université, ce qui oblige à passer par une connexion VPN si l'on souhaite travailler à distance. L'université n'a également pas été en mesure de nous fournir de l'IPv6 pour nos différentes machines virtuelles.

FIGURE 2.1 – Architecture de l'infrastructure de ChaTalK



Le schéma suivant illustre l'architecture actuelle de notre projet :

Chaque grappe Kubernetes fait tourner les éléments suivants :

— un contrôleur d'entrée (nous utilisons *nginx-ingress-controller*[1] qui est celui pris officiellement en charge par Kubernetes), qui permet de rediriger le trafic entrant vers les bons services hébergés sur la grappe,

— une solution de basculement d'IP avec l'aide du projet *metallb*[2] ; en demandant une IP supplémentaire pour chaque grappe (*chatalk* pour grappe1 et *chatalk-vip* pour grappe2) metallb va l'attribuer à l'un des nuds de la grappe. Si un des nuds de la grappe en question tombe, cette IP sera automatiquement attribuée à un autre nud qui est disponible si c'était ce nud qui possédait cette adresse. Le fait d'avoir une adresse IP qui est certaine de pointer sur un nud qui est disponible nous permet de faire pointer les entrées DNS dessus, le contrôleur d'*ingress* se charge de répartir la charge convenablement au sein de la grappe. Il s'agit d'un mécanisme qui est habituellement déjà en place lorsque l'on est chez un fournisseur de services infonuagiques avec l'utilisation de *load-balancers* physiques, ce qui n'est pas notre cas.
Metallb est une solution reconnue et est utilisée massivement pour les cas similaires au nôtre, c'est la raison pour laquelle nous l'avons choisie.
Utiliser de la répartition de charge au niveau DNS n'est pas efficace, car si un des trois nuds d'une grappe tomberait en panne, un tiers des requêtes échouerait.
Nous ne pouvons également pas utiliser de l'*anycast* car on n'aurait pas la garantie qu'un flux TCP irait sur un même nud, ce qui pourrait s'avérer problématique dans le cas où on récupérerait un gros fichier par exemple.

— un bus de données (*nats* / *nats-streaming*[3]), qui est une solution vraiment légère et performante. Apache Kafka[4] ou RabbitMQ[5] sont des alternatives qui sont relativement plus grosses, et NSQ[6] est une alternative similaire à Nats, mais semble être moins utilisé.
Utiliser un bus de données nous permet d'empiler les différents messages avec *nats-streaming* en cas de montée en charge soudaine, ce qui évite d'avoir à implémenter cette logique dans chacun des micro-services.

1. `https://kubernetes.github.io/ingress-nginx/`
2. `https://metallb.universe.tf/concepts/layer2/`
3. `https://nats.io/`
4. `https://kafka.apache.org/`
5. `https://www.rabbitmq.com/`
6. `https://nsq.io/`

— un gestionnaire de base de données PostgreSQL (*stolon*[1]) qui nous permet d'avoir des instances de PostgreSQL en haute-disponibilité et de faire de la fédération entre deux sites. Au départ nous étions partis sur *postgres-operator*[2] de Zalando, mais les logs n'étaient pas assez précis pour permettre de déboguer convenablement la solution.

— un ensemble d'outils pour faire de la surveillance, avec l'aide du projet *kube-prometheus*[3], qui nous permet de déployer et configurer assez rapidement *AlertManager*[4], *Prometheus*[5] et *Grafana*[6] (voir la partie concernant la supervision pour plus d'informations) qui sont les outils les plus couramment utilisés pour faire de la supervision au sein d'une grappe Kubernetes.

— notre application, que nous allons détailler dans la partie suivante.

Afin de rendre notre application accessible en IPv4 et en IPv6 (plus d'information dans la partie concernant la double-pile), ainsi que pour faire de la répartition de charge entre les deux sites, Ludovic a mis en place une machine virtuelle à l'AIUS (lb). Encore une fois, utiliser de l'*anycast* n'était pas possible, et utiliser de la répartition de charge au niveau DNS ferait que si l'un des deux sites tombe, la moitié des requêtes échouerait, ce qui n'est pas envisageable. L'ensemble de l'infrastructure étant accessible uniquement depuis le réseau de l'Université, seuls l'interface web et un point d'entrée pour établir une connexion WebSocket vers l'arrière-plan sont exposés sur Internet via cette machine virtuelle.

2.7 Application

L'application est décomposée en deux grandes parties : la partie *frontale* qui est l'interface utilisateur, une application web sur navigateur Internet ou une application mobile sur téléphone, ainsi que la partie *arrière-plan* qui est l'ensemble des micro-services déployés pour traiter les différentes requêtes.

2.7.1 Application web

Aujourd'hui un site web peut se comporter comme une application mobile[7]. De plus, une application web à l'avantage de ne nécessiter qu'un simple navigateur web moderne à jour et respectant les différents standards ; pas besoin d'installer quoi que ce soit en plus, ce qui est pratique pour les utilisateurs, et ce indépendamment de la plateforme utilisée (ordinateur, télévision, tablette ou smartphone).

L'application web a été développée avec ReactJS[8], une bibliothèque JavaScript largement reconnue et utilisée, qui permet de créer une application mobile complète et dynamique en évitant de réinventer la roue. Nous utilisons TypeScript[9] qui génère du code JavaScript en offrant l'avantage de faire de la vérification sur les types, évitant ainsi un grand nombre de bogues en production.

Cette application web se connecte à l'arrière-plan avec l'aide d'une connexion WebSocket sécurisée. Un mécanisme de reconnexion au WebSocket a été mis en place dans le cas où la connexion au WebSocket est momentanément indisponible ou est interrompue.

2.7.2 Application mobile

Ludovic a réalisé une application mobile pour Android avec Kotlin[10] dans le cadre de l'UE Programmation Mobile. L'application a été codée en utilisant les dernières recommandations Android dans l'optique d'avoir un code propre et maintenable.

L'application a été développée sur Android Studio et a été testée sur les appareils suivants :

1. `https://github.com/sorintlab/stolon`
2. `https://github.com/zalando/postgres-operator`
3. `https://github.com/coreos/kube-prometheus`
4. `https://prometheus.io/docs/alerting/alertmanager/`
5. `https://prometheus.io/`
6. `https://grafana.com/`
7. `https://www.howtogeek.com/196087/how-to-add-websites-to-the-home-screen-on-any-smartphone-or-tablet/`
8. `https://reactjs.org/`
9. `https://www.typescriptlang.org/`
10. `https://kotlinlang.org/`

— Pixel 3a, Android Q, API 29 (émulateur d'Android Studio),

— Honor 10, Android P, API 28 (téléphone personnel).

L'application mobile dispose d'un certain nombre de fonctionnalités communes par rapport à l'application web, telles que la connexion, l'inscription, la déconnexion, la liste des conversations, l'affichage d'une conversation ainsi que l'envoi et la réception de messages dans une conversation.

Cependant, par manque de temps, il n'est pas possible de créer une nouvelle conversation et d'en gérer (la renommer, ajouter ou supprimer des personnes). De plus elle ne respecte ni la charte graphique [1], ni les maquettes [2].

Néanmoins, cette application prends en charge le support du thème sombre. Si un utilisateur a configuré son appareil Android pour utiliser un thème sombre, l'application sera alors lancée avec un thème sombre. Sinon un thème clair sera utilisé à la place.

De plus, elle prend en charge le multi-langues. Par défaut l'application a été développée en langue anglaise. Mais si l'utilisateur a son appareil Android configuré en français, alors l'application sera affichée en français.

2.7.3 Partie arrière-plan/serveur

Cette partie est constitué de différentes parties :

— un service *entrypoint* qui est le point d'entrée sur lequel l'application web ou mobile va se connecter. Ce service est chargée de maintenir les connexions WebSocket et de relayer les différentes requêtes sur les bons canaux du bus de données afin de les transmettre au service apte à traiter la demande.

— un service *login* qui suite à un message récupéré depuis le bus, va permettre à un utilisateur de se connecter avec ses identifiants ou un token JWT, ou bien de se déconnecter.

— un service *register* qui va permettre à un utilisateur de s'inscrire,

— ainsi que différents services qui sont chargés de renvoyer la liste des conversations, diffuser les messages, etc.

2.8 Stockage des données

Nous utilisons une base PostgreSQL composé de différentes tables afin de stocker les différentes données, telles que les informations de l'utilisateur, les messages, etc.

2.8.1 Architecture

Afin d'avoir une base de données en haute disponibilité, nous avons déployé Stolon (voir plus haut).

Chaque grappe fait tourner une instance de Stolon, et les deux sont fédérées entre-elles. Cependant seule l'instance de la grappe d'Esplanade (grappe2) est maître. C'est-à-dire que les lectures peuvent être réalisées depuis n'importe quel instance. Cependant les écritures doivent être transmises à l'instance maître (*dbmaster*). Par manque de temps, nous n'avions pas pu mettre en place un basculement automatique, car cela est relativement complexe à faire, mais faisable si nous avions pu avoir quelques mois supplémentaires dans le but de tout finaliser correctement.

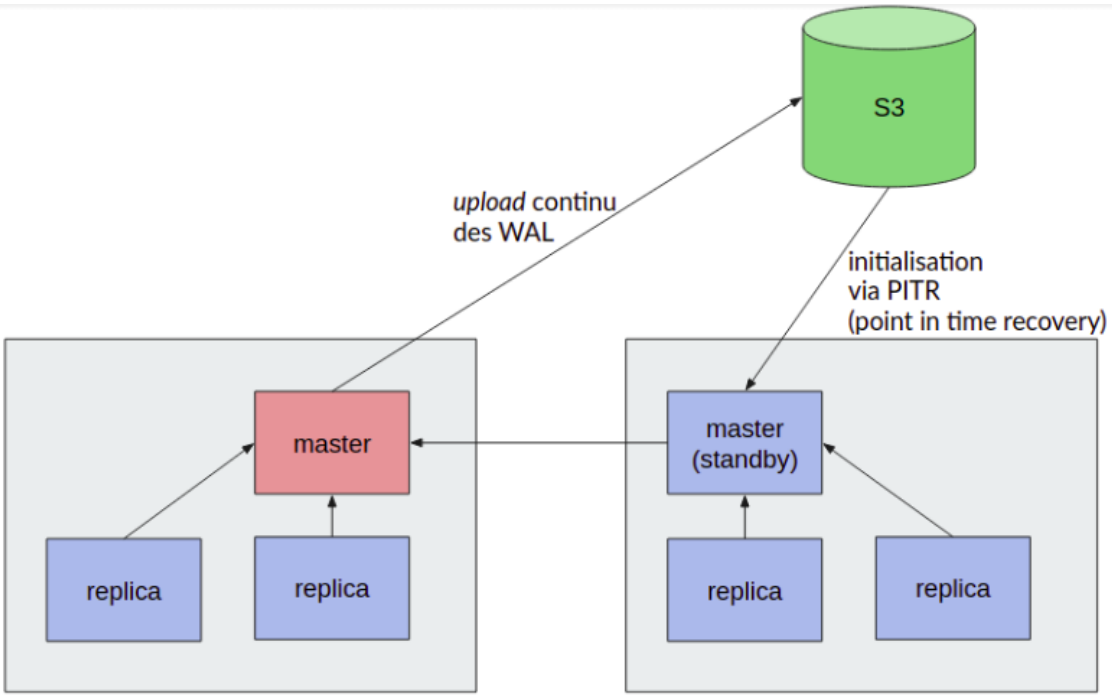
Voici à quoi ressemble l'architecture de notre base de données :

Nous avons configurés Stolon pour qu'il déploie une instance de Postgres par nud (donc trois par grappe). L'une de ces instance est désignée en tant que *master* et les autres sont configurées comme étant des réplicas.

Sur la grappe1, qui est configurée pour ne pas être *master*, l'architecture sera similaire. Cependant la *master* sera mise en *stand-by* et ne fera que des lectures.

1. `https://blog.chatalk.fr/graphic-charter/`
2. `https://blog.chatalk.fr/mock-ups-update/`

FIGURE 2.2 – Architecture de la base de données



Une grappe de Postgres est initialisée avec l'aide d'un point de récupération d'un instantané stocké sur notre instance de stockage S3 qui a été envoyé par le *master* de l'instance principale en envoyant les journaux de transactions (WAL).

2.9 Sécurisation et analyse des risques

Seuls l'application web et un micro-service (entrypoint) sont exposés vers l'extérieur à travers un proxy hébergé à l'AIUS (qui forward uniquement le trafic TCP des ports 80 et 443 vers les grappes) la surface d'exposition est minimisée.

L'interface web ainsi que le websocket sont uniquement joignables de manière sécurisé avec de l'HTTPS.

Cependant n'importe qui depuis le réseau de l'Université ou utilisant le VPN peut atteindre certains points auxquels ils ne devraient pas avoir accès. Par exemple, le bus de données que nous utilisons possède un mécanisme d'authentification pour que seuls les services autorisés puissent s'y connecter ; hélas par manque de temps cela n'a pas été configuré et des personnes malintentionnées pourraient y brancher des services pour écouter ce qui se passe sur le bus et récupérer des identifiants par exemple.

La machine virtuelle que nous avons à l'AIUS est également un point critique important : si elle tombe, plus aucun service ne sera accessible. Le fait d'utiliser *haproxy* de la manière dont on le fait nous prive de voir l'IP réelle du client depuis les services hébergés sur les grappes, bien que nous ne nous servons pas, cela pourrait être utile pour l'analyse de certains logs.

2.10 Supervision

La supervision peut s'effectuer à différents niveaux. Tout d'abord il est possible de récupérer les différentes statistiques du proxy en place à l'AIUS grâce à l'activation de l'interface web de *haproxy*.

Nous utilisons également un service tiers (UptimeRobot [1]) pour vérifier si le site est toujours joignable depuis l'extérieur. Le statut actuel peut être consulté ici : `https://status.chatalk.fr/`.

Ce service va effectuer un ping régulier et effectuer régulièrement des requêtes HTTP vers la page d'accueil et va envoyer un mail à Ludovic en cas d'incident. L'incident de Renater du 18 décembre

1. `https://uptimerobot.com/`

2019 a par exemple été suivi en temps réel.

Concernant la surveillance au niveau des grappes, nous avons, avec le projet *kube-prometheus*[1], déployé une pile de surveillance qui comporte trois éléments principaux :

— Alert Manager, qui permet la configuration d'un certain nombre d'alertes,

— Prometheus, qui permet d'afficher ces alertes, de récupérer des métriques exposées par Kubernetes ainsi que certaines qui sont exposées par certains services en explorant des URL particulières sur ces derniers,

— Grafana, qui avec l'aide des métrique récupérées depuis Prometheus, va afficher un grand nombre de tableaux de bords avec des graphes, des valeurs...

La figure suivante montre la manière dont tout est connecté ensemble. Le plan de contrôle de Kubernetes va régulièrement vérifier les services qui sont configurés pour l'être afin de voir s'ils sont bien fonctionnels. Prometheus va explorer des URL spécifiées par les services qui proposent d'exposer leur propres métriques ainsi que celles exposées par la grappe Kubernetes. Alert Manager va récupérer ces métriques et voir si certaines d'entre elles dépassent certaines valeurs et va donc lever certaines alertes. Grafana est une belle interface pour avoir une vue d'ensemble des métriques récupérées sur Prometheus avec l'aide de beaux graphes. Les métriques récupérées par Prometheus sont rendues accessibles au plan de contrôle avec l'aide d'un *Prometheus Adapter* qui va se charger de rendre les différentes métriques compréhensibles pour le plan de contrôle. Le plan de contrôle pourra donc être en mesure de pouvoir augmenter ou diminuer le nombre d'instances de certains services (utilisation de *Horizontal Pod Autoscaler*[2]).

En cas de soucis, les logs peuvent être facilement récupérés avec l'aide de l'outil *kubectl*, qui est un outil en ligne de commandes utilisé pour gérer une grappe Kubernetes.

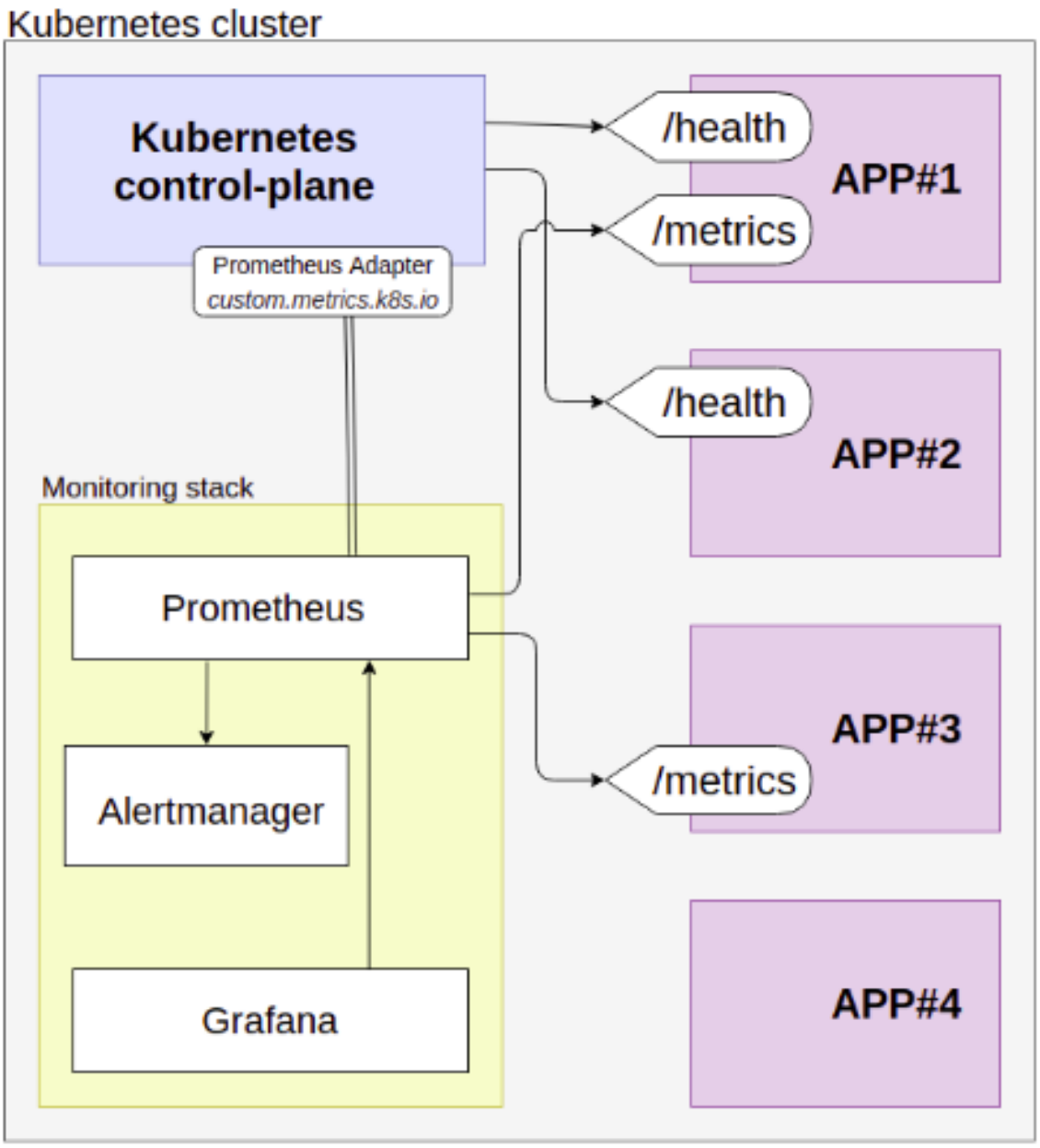
2.11 Double pile v4/v6 + v6-seul

Nos deux sites ne disposent pas de connectivité IPv6 du fait que l'Université de Strasbourg n'a pas été en mesure de pouvoir nous en router. Néanmoins, même avec de l'IPv6 il aurait été compliqué de configurer convenablement Kubernetes pour prendre en charge la double pile IPv4/IPv6 qui est encore en stade d'alpha à l'heure actuelle. C'est donc la raison pour laquelle en interne seul de l'IPv4 est utilisé.

Cependant nous avons fait en sorte que notre application puisse être accessible pour les utilisateurs que ce soit en IPv4 et en IPv6. Pour ce faire, Ludovic a déployé un proxy sur une machine virtuelle de l'AIUS, qui elle est accessible publiquement en IPv4 et en IPv6. Lorsqu'une connexion d'un utilisateur arrive, le proxy (*haproxy* en l'occurence), va ouvrir une nouvelle connexion TCP vers les deux sites en faisant de la répartition de charge sur les grappes disponibles.

1. `https://github.com/coreos/kube-prometheus`
2. `https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/`

FIGURE 2.3 – La surveillance sous Kubernetes



Partie n^o 3

Solution

La solution telle que nous l'avons conçue comprenait plusieurs éléments que nous allons rappeler ci-dessous.

Tolérance aux pannes et passage à l'échelle Grâce à son infrasctucture basée sur Kubernetes et à son architecture en micro-services, notre application est aisément capable de passer à l'échelle et de supporter la montée en charge, ainsi que de résister à la perte de machines dans les grappes Kubernetes. Dans les pires cas, les les fonctionnalités passent en mode dégradé. Ce mode dégradé permet la consultation des archives de conversations et donc de conserver une partie des activités des utilisateurs du service.

Sécurité Notre application devait garantir la sécurité des flux de données, en particulier d'un point de vue intégrité et confidentialité. Seuls les envoyeurs et récepteurs des messages devaient avoir accès aux données.
Nous avons partiellement rempli cet objectif, des contraintes de temps liées à nos retards nous ayant forcé à abandonner le chiffrement des flux audiovisuels. L'authentification des utilisateurs a néanmoins été réalisée. Une couche de sécurité minimale basée sur SSL/TLS assure un niveau de sécurité décent, tant que les utilisateurs du service font confiance à l'administrateur des serveurs sur lesquels fonctionne la solution.

Déploiement facile Notre application devait pouvoir être facilement déployée par un néophyte, sur n'importe quelle machine de type Linux, rapidement et sans connexion Internet. Cette tâche était d'une priorité maximale

Temps réel Notre application devait assurer des conversations en temps réel entre les utilisateurs. Nos tests actuels ne sont pas représentatifs d'une utilisation réelle, mais montrent que la latence des communications textuelles sur notre solution est imperceptible à l'humain pour un petit nombre d'utilisateurs.

Interface utilisateur Notre solution devait proposer une interface utilisateur compatible avec un maximum d'appareils.
Notre application frontale Web est utilisable sur tous les navigateurs modernes sans modification et sans distinction de système d'expoitation. De plus elle s'adapte à des tailles d'écran diverses et à des appareils de puissance variée. Bien que cette interface Web puisse être utilisée en tant qu'application mobile, une application mobile pour Android en Kotlin est également disponible, plus adaptée à l'utilisation sur des téléphones intelligents.
Nous avons choisi une interface utilisateur épurée, en conservant les icônograpies habituelles, afin de permettre une meilleure expérience utilisateur et une plus grande ergonomie.

Modularité Notre solution devait permettre à un individu déployant un serveur de choisir les capacités qu'il souhaitait donner à celui-ci.

À cause des contraintes de temps, et puisque la tâche n'était pas prioritaire, elle a été la première abandonnée totalement. Les fonctionnalités supplémentaires audio et vidéo ont été incluses directement dans l'application d'arrière-plan principale et le concept de modules serveur a été laissé à l'abandon.

Compatibilité et standards Notre application devait être compatible avec un maximum de standards et ne pas utiliser de technologies trop peu orthodoxes.
Puisque l'interface utilisée est une interface Web, les quatre systèmes d'exploitation les plus courants, Microsoft Windows, Apple macOS, Apple iOS et Android, sont couverts tant qu'ils possèdent un navigateur Internet à jour. De plus, les systèmes d'exploitation GNU/Linux et BSD sont également couverts tant qu'ils possèdent un navigateur Internet à jour. Enfin, s'il n'existe pas d'application native pour iOS, il en existe une pour Android.
D'un point de vue des standards, notre application est accessible à des clients utilisant indifféremment de l'IPv4, de l'IPv6 ou les deux.

Internationalisation Notre application se devait de supporter les principaux alphabets du monde et d'être disponible à minima en Anglais.
Puisque tous les textes que nous utilisons sont encodés en UTF-8, les alphabets divers sont gérés dans l'application. L'interface que nous avons développée l'a été en Anglais et est donc utilisable par des utilisateurs anglophones.

Partie n° 4

Gestion de projet

4.1 Organisation et commentaire général du projet

Au départ du projet, nous avions prévu des binômes s'occupant des tâches. Au fil du déroulement du projet, nous nous sommes aperçus que certaines parties du projet avançaient plus vite que d'autres, et que l'idée de binômes a changé pour confier à chacun les tâches dans lesquelles il se sentait à l'aise.

Le suivi du projet était assuré par des réunions régulières avec les clients / professeurs, entre une réunion par semaine et une réunion pour deux semaines selon les périodes. Ces réunions étaient suivies d'un compte-rendu à l'équipe projet, durant lequel le nouveau planning était généralement présenté, et où les tâches à venir étaient réparties entre les membres de l'équipe.

Les membres de l'équipe étaient informés de leurs missions via trois moyens principaux :

— des *issues* GitLab, avec une date, une mission et un responsable

— une notification écrite sur l'application de communication Discord

— une notification orale, généralement à la fin de la réunion de répartition des tâches

Malgré les efforts mis par le chef de projet pour notifier son équipe des missions à réaliser, il est arrivé trop souvent que les issues soient créées tardivement et que la notification orale reste la seule information donnée à l'équipe.

La plus grosse difficulté pour l'organisation de ce projet a été de l'ordre de la communication. Gabriel a souvent été peu réactif à officialiser par écrit les tâches à réaliser d'une semaine sur l'autre. La majorité l'équipe a été peu proactive, en particulier au début, pour donner des retours sur ses avancements et les tâches sur lesquelles elle travaillait et comment ces tâches avançaient.

4.2 Outils utilisés pour l'organisation et la gestion du projet

Discord L'outil principal utilisé pour l'organisation et la gestion du projet a été le logiciel de messagerie textuelle et vocale Discord. Nous l'avons choisi car nous l'utilisons tous quotidiennement et qu'il nous permet l'utilisation de certaines fonctionnalités pratiques comme la création de canaux thématiques pour ne pas mélanger les communication à propos de plusieurs tâches, ou l'épinglage de messages pour les retrouver facilement.

Réunions en présentiel Un outil très efficace que nous avons utilisé lors de l'organisation du projet a été les réunions en présentiel, avec autant de membres de l'équipe que possible. Rétrospectivement, nous aurions du en faire plus souvent. Lors de ces réunions, l'utilisation d'un tableau blanc nous permettait de schématiser et de planifier. Ces réunions étaient également propices à l'entraide et à la pédagogie, permettant des explications en face à face direct.

Git et GitLab Pour gérer le code source Git est devenu un outil indispensable. Toutefois, outre cette utilisation, la création de problèmes suivis a permis de communiquer sur les spécification des tâches à plusieurs reprises. Les historiques de *commit* permettent plus ou moins de voir qui a travaillé à quel moment et de détecter les passages à vide.

Gantt Project Pour créer et officialiser le planning, l'outil Gantt Project a été utilisé, permettant de créer simplement des diagrammes de Gantt. Ces diagrammes ont ensuite été partagés sur le dépôt GitLab pour que tous puissent les consulter.

4.3 Réactivité et gestion humaine

L'un des plus gros problèmes de ce projet a été la gestion de la communication. D'un côté certains membres de l'équipe ont été peu proactifs en ce qui concerne le fait d'informer les autres de leur avancement, mais également de leurs difficultés. Certains membres de l'équipe ont également été peu réactifs à informer le chef du projet de leur impossibilité temporaire à remplir leurs fonctions (maladie, problèmes personnels grâves). D'un autre côté, le chef de projet n'a pas été s'enquérir rapidement de l'état de ses équipiers lorsque ceux-ci étaient absents ou ne donnaient pas de nouvelles pendant plusieurs jours.

Ce manque de communication a posé problème pour la redistribution des tâches auprès des autres membres de l'équipe. Si l'informations avait été obtenue plus tôt, les tâches non assurées auraient pu l'être par quelqu'un d'autre et le retard aurait été mitigé.

Toutefois, lors des cinq dernières semaines du projet, lorsqu'une tâche devait être réattribuée, elle l'a été sans délai, pendant la période nécessaire.

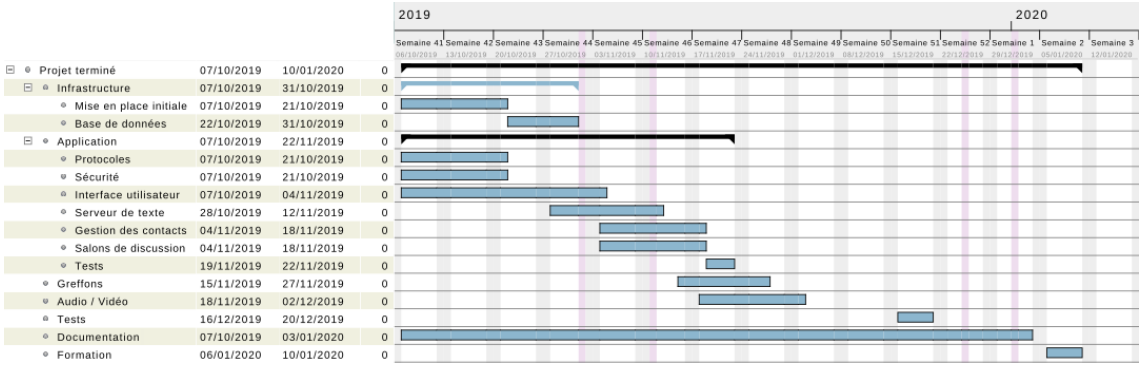
Partie n° 5

Planning et évolutions

5.1 Planning initial

Le planning initial illustré par la figure 5.1 prévoyait que la majorité du projet serait terminée avant mi-décembre, ne laissant pour la fin d'année que la correction de détails et la rédaction du mémoire final.

FIGURE 5.1 – Planning prévisionnel initial



Les tâches du planning initial étaient très peu subdivisées, ce qui n'a pas facilité l'estimation de la charge de travail requise pour les accomplir et est sans doute l'un des facteurs ayant conduit à certains retards.

Nos prévision étaient que chaque tâche majeur prendrait deux semaines à réaliser. Si cette estimation était probablement vrai pour certaines tâches de conception, un découpage plus fin comme celui du dernier Gantt illustré par la figure 5.2 nous aurait permis de voir plus facilement le travail impliqué par chaque tâche.

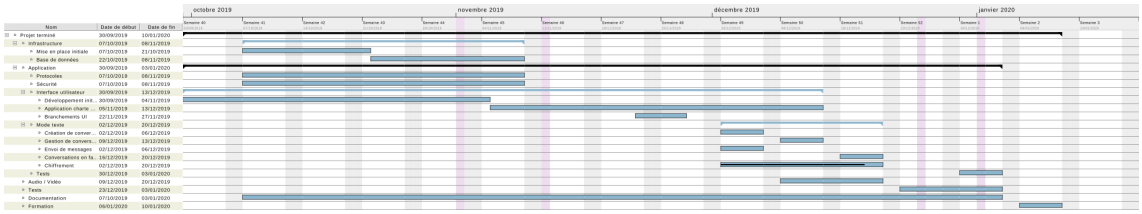
De plus, notre inexpérience a conduit à des imprécisions dans nos documents initiaux, et des flous en terme de choix techniques. Les enjeux en la matière ayant été mal compris, nous avons du faire valider plusieurs choix par nos clients, ce qui nous a ralenti davantage, car plusieurs échanges de requête réponse entre humains prennent rapidement une à deux semaines.

5.1.1 Planning final

Au fil des retards successifs notre organisation a pris la forme illustrée par la figure 5.2, présentant une liste de tâches plus détaillée, avec des tâches divisées en tâches plus petites et plus courtes, aboutissant à des tâches effectives plus longues que ce que nous avions prévu à l'origine.

Finalement le projet a avancé beaucoup moins rapidement que ce que nous prévoyions, et ce que Gabriel avait prévu dès le départ arrivé sur la fin, la période des fêtes de fin d'année et les vacances associées ont été une période globalement très peu productive.

FIGURE 5.2 – Planning prévisionnel final



5.1.2 Outil utilisé

L'outil *Gantt Project* utilisé pour réaliser ces diagrammes possède des capacités peu utilisées par le chef de projet : les graphes de dépendances notamment, qui auraient permis une représentation plus visuelle du travail à faire, et du travail déjà fait. La gestion des ressources également, une fonctionnalité mal documentée et mal compris du logiciel qui auraient permis de facilité le calcul des coûts lors des phases initiales du projet.

Partie n^o 6

Emplois du temps et répartition du travail

Partie n^o 7

Conclusion