

Métrologie centralisée d'un cluster Docker

Par : Maxence Luongo, Sébastien Saintot, Evan Daussin, Tom
Thioulouse

Table des matières

1	Remerciements	4
2	Introduction	5
2.1	Contexte	5
2.2	Objectifs	5
2.3	Principe de métrologie centralisée	6
3	Docker	8
3.1	La conteneurisation	8
3.2	Un peu d'histoire	10
3.3	Fonctionnalité Swarm	10
4	Métrologie	12
4.1	Schéma d'architecture général	12
4.2	Communication entre logiciels	13
5	Outils de métrologie	14
5.1	Les Sondes	14
5.1.1	cAdvisor	14
5.1.2	Node Exporter	14
5.2	Prometheus	15
5.2.1	Exportateurs Prometheus	15
5.2.2	Fonctionnement de Prometheus	16
5.2.3	Base de données de séries temporelles	17
5.3	InfluxDB	19
5.3.1	Présentation	19
5.3.2	Fonctionnement	19
5.4	Grafana	21
5.4.1	Fonctionnalités de Grafana	21
5.4.2	Data Source	21
5.4.3	Les Dashboards	21
6	Les applications web	23
6.1	WordPress	23
6.2	MailCow	24
6.3	Gatling	25

7	Cas d'expérimentation	27
7.1	Schéma d'architecture	27
7.2	Mise en place	30
7.2.1	Docker Swarm	30
7.2.2	Sondes	32
7.2.3	Prometheus	33
7.2.4	InfluxDB	34
7.2.5	Grafana	38
7.2.6	WordPress	40
7.2.7	MailCow	42
7.2.8	Gatling	44
7.3	Problèmes rencontrés	46
7.3.1	MailU	46
7.3.2	Netdata	47
7.3.3	InfluxDB	47
7.3.4	Infrastructure	48
8	Organisation du projet	49
8.1	Outils collaboratifs	49
8.2	Répartition des tâches	50
9	Conclusion	51
10	Bibliographie	52
11	Annexes	56
11.1	Vagrantfile	56
11.2	provision.sh	57
11.3	docker-compose-manager.yml	58
11.4	daemon.json	59
11.5	docker-compose-monitoring.yml	59
11.6	prometheus.yml	60
11.7	SimWordPress.java	60

Chapitre 1

Remerciements

Avant d'introduire ce rapport de projet, nous souhaitons d'abord remercier l'intégralité de la licence professionnelle ASRALL à l'IUT Nancy-Charlemagne, enseignants comme étudiants, pour ces 7 mois d'enseignements autour du Logiciel Libre auxquels nous avons eu la chance d'assister tous ensemble en physique.

Nous remercions plus spécifiquement les étudiants pour avoir distillé bienveillance et partage au sein de la salle de classe.

Ainsi que le corps enseignant pour leur volonté de transmettre leurs connaissances et leurs expériences.

Nous remercions également Monsieur Philippe Dosch, responsable de cette licence professionnelle. D'une part, pour avoir rendu celle-ci possible. D'autre part, pour nous avoir guidé tout au long de ce projet et avoir répondu à nos questionnements, souvent par des principes qui nous permettaient de trouver les réponses à des questions complexes par nous-mêmes. Mais aussi pour ses conseils en matière de rédaction et plus spécifiquement pour ses astuces de passionné sous LaTeX, solution que nous avons choisie pour rédiger ce rapport.

Chapitre 2

Introduction

2.1 Contexte

Pour tout administrateur système, le critère de succès le plus important se résume simplement à une infrastructure qui fonctionne. Avec l'émergence de la haute disponibilité, plusieurs solutions permettant d'aboutir à une infrastructure bien pensée, ont vu le jour, notamment le failover ou la redondance. Or, comme démontré par la loi de Murphy, ce qui peut mal se passer finira par arriver.

C'est en suivant cette idéologie, qu'on finit par s'intéresser au concept de métrologie. Celui-ci permet de récolter des informations sur l'état de notre infrastructure et détecter voire prédire les problèmes avant que les utilisateurs ne nous les signalent. La métrologie est à ne pas confondre avec la supervision, nous cherchons bien à récupérer la charge et la tracer dans le temps et non pas à récupérer l'état d'un service à un instant T pour vérifier son bon fonctionnement.

Dans notre cas, nous utilisons alors plusieurs composantes inhérentes à la métrologie dont les sondes pour obtenir les données, les bases de données pour les stocker, les dashboards pour afficher les résultats de manière centralisée, et éventuellement un système de notifications pour notifier d'incidents imminents.

2.2 Objectifs

Le projet consiste à conteneuriser des applications web au sein d'un cluster Docker et d'être capable de récupérer les charges du cluster et des applications web sur diverses échelles de temps. Il faudra ensuite les tracer dans le temps à partir des métriques récoltés, de manière centralisée pour avoir une vue d'ensemble sur les performances de l'infrastructure complète.

Dans le cadre de ces objectifs nous utiliserons d'abord Docker Swarm pour mettre en place notre cluster et déployer nos applications web conteneurisées. L'avantage de cette solution est que l'équilibrage de charge se fait automatiquement et nos services sont facilement extensibles.

Pour aboutir à la métrologie souhaitée de notre infrastructure, nous utiliserons plusieurs sondes : cAdvisor et Node Exporter qui permettent respectivement de récupérer les métriques des conteneurs et des machines hôtes, ce sont les technologies qui constituent notre

métrologie fine.

Quant à la métrologie moyenne et la rétention longue nous nous sommes tournés vers Prometheus et InfluxDB.

Enfin, nous avons choisi Grafana pour centraliser et visualiser les métriques sous forme de dashboards.

Pour simuler des montées en charge sur les applications web, nous utiliserons Gatling avec un programme, écrit en Java, dont le rôle sera de générer du trafic.

2.3 Principe de métrologie centralisée

La métrologie centralisée de notre cluster Docker représente notre résultat final. C'est une solution sous-jacente au concept de monitoring, tout comme la supervision. Pourtant, ces deux notions sont bien différentes et il est important d'être capable de les distinguer. Quand on parle de métrologie, cela signifie qu'on cherche à récupérer la charge d'un système et la tracer dans le temps, ainsi nous pourrions afficher et visualiser l'évolution de la charge, qui sera construite par l'ensemble des métriques récupérées dans le temps. En revanche, la supervision consiste seulement à récupérer l'état d'un service à l'instant T. Comprenez bien que les valeurs numériques jouent un rôle moins prépondérant dans le cas de la supervision. Néanmoins, elles restent présentes, par exemple dans le cas où on voudrait connaître l'espace de stockage sur un disque dur. Cependant, la plupart du temps nous cherchons seulement à savoir si le service supervisé est joignable ou non, auquel cas les valeurs numériques et l'aspect historique de charge n'entrent pas en compte contrairement à la métrologie.

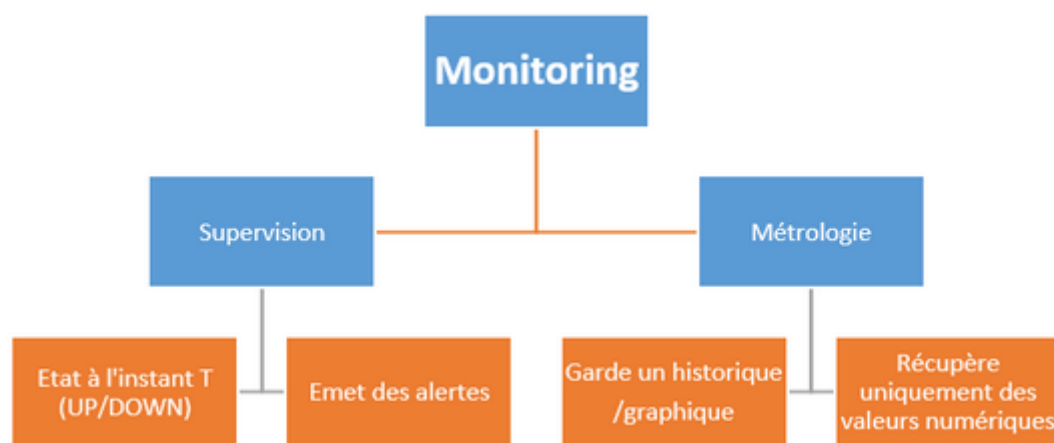


FIGURE 2.1 – Représentation schématique du monitoring et ses concepts sous-jacents

Mais revenons à ce qui nous intéresse, la métrologie centralisée. Pour obtenir notre évolution de la charge, nous avons besoin de métriques que nous allons récupérer, garder et tracer dans le temps. Pour ce faire, nous aurons besoin de plusieurs outils : des sondes, des bases de données, des dashboards, et possiblement des systèmes de notifications. En revanche, il reste une contrainte à aborder : la centralisation. En effet, il est possible selon nos besoins de métrologie, que plusieurs sondes ou plusieurs échelles de temps soient nécessaires pour avoir une vue représentative de notre système, ce qui impliquerait que nous ayons plusieurs

groupes de métriques à différents endroits. Malgré que nous pourrions nous affranchir de cette problématique de centralisation, la possibilité de visualiser l'évolution de toutes nos métriques au même endroit constitue un confort non négligeable.

Chapitre 3

Docker

Puisque nous ne nous limitons pas au thème de la métrologie centralisée mais que nous l'appliquons au cas d'un cluster Docker, cette notion nous est également essentielle à la compréhension de ce qui suit.

Docker est une technologie qui permet d'empaqueter une application avec ses dépendances dans un conteneur virtuel et isolé qu'on pourra envoyer et faire fonctionner vers n'importe quel serveur Linux. Alors que Docker ne fait pas partie des outils de métrologie à proprement parler, c'est une technologie qui n'en reste pas moins essentielle. Elle constitue souvent la base même de solutions de métrologie en raison de sa rapidité et sa facilité de mise en place d'applications, ainsi que pour ses aspects de transportabilité et de scalabilité.



FIGURE 3.1 – Logo Docker

3.1 La conteneurisation

Pour arriver à ses fins, Docker utilise ce que l'on appelle la technologie de conteneurisation. Si l'on veut comprendre le fonctionnement de ces conteneurs, il est alors intéressant de les comparer aux machines virtuelles.

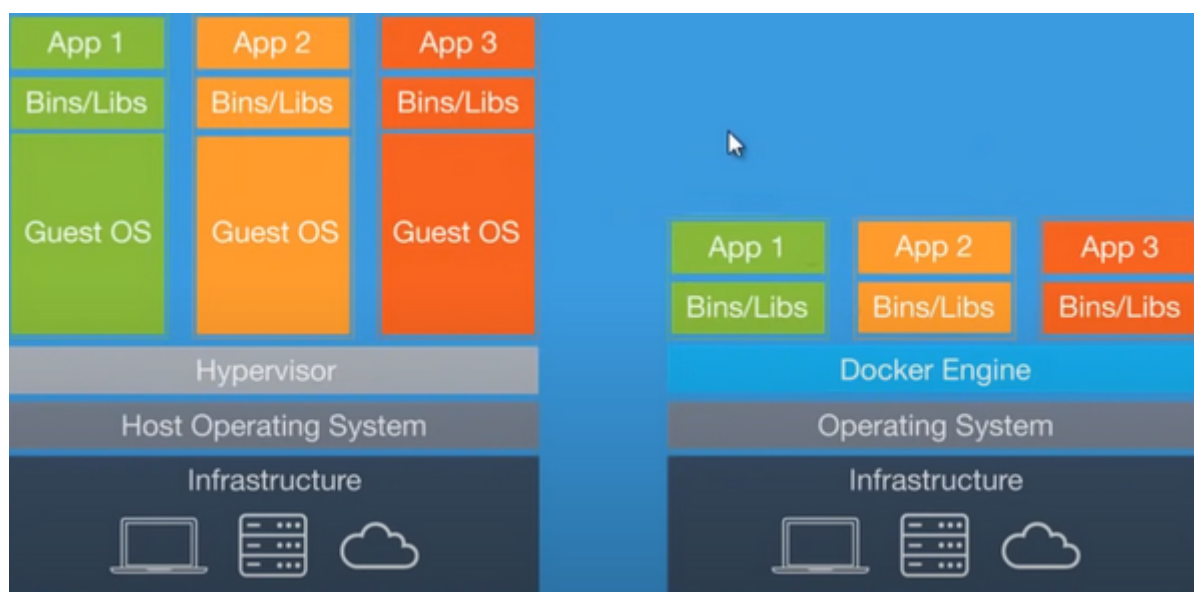


FIGURE 3.2 – Schéma comparatif entre le fonctionnement de machines virtuelles et de conteneurs Docker

Tout d'abord, les technologies de conteneurisation (dont Docker fait partie) et la virtualisation permettent les unes comme les autres d'isoler une application pour la rendre opérationnelle dans plusieurs environnements. Quant aux différences, trois points sont à retenir : la transportabilité, la scalabilité et les performances.

D'une part, une machine virtuelle représente une émulation d'une machine par un logiciel sur une machine hôte. Ce logiciel d'émulation y simule la présence de ressources matérielles et logicielles telles que la mémoire, le processeur, le disque dur, le système d'exploitation et les pilotes. La machine virtuelle est capable d'exécuter des programmes grâce à l'allocation de ressources de l'hôte. En revanche, le poids se comptera en giga-octets.

D'autre part, les conteneurs regroupent en un ensemble cohérent et prêt à être déployé sur un serveur et son OS, une application et les éléments nécessaires à son bon fonctionnement (code de l'application, bibliothèques, fichiers de configuration et dépendances requises). Le point fort de la conteneurisation réside dans le fait que les conteneurs ne contiennent pas d'OS puisque c'est le noyau du système hôte qui est utilisé (partagé si on a plusieurs conteneurs), ce qui apporte une certaine légèreté à ce type de technologie.

Cela se traduit par une facilité accrue de migration/téléchargement ou sauvegarde/restauration. D'où la notion de transportabilité. Mais en raison de leurs poids réduits, les conteneurs peuvent également redémarrer plus rapidement après chaque modification apportée à une application, c'est le concept de scalabilité.

Enfin, étant donné que les moteurs de conteneurs n'ont pas besoin d'émuler un OS complet contrairement aux hyperviseurs des machines virtuelles, on obtiendra de meilleures performances.

3.2 Un peu d'histoire

Auparavant, Docker était basé sur LXC (Linux Containers), autrefois l'implémentation de référence de conteneurs dans Linux. L'idée consistait donc à utiliser LXC comme base et ajouter des capacités de niveau supérieur. Mais depuis la version 0.9 du logiciel, Docker a abandonné LXC comme environnement d'exécution par défaut en le remplaçant par son propre libcontainer, écrit en Go. Celui-ci, permet d'avoir accès à des fonctionnalités du kernel Linux qui sont essentielles au bon fonctionnement des conteneurs. Deux des plus importantes sont les espaces de noms, qui permettent d'empêcher un système de voir les ressources utilisées par le système hôte ou un autre conteneur, et les groupes de contrôle qui permettent de délimiter et isoler l'utilisation des ressources (processeur, mémoire, utilisation disque,).

Texte traduit du blog officiel de Docker :

Docker peut désormais manipuler les espaces de noms, les groupes de contrôle, les capacités, les profils d'apparence, les interfaces réseau et les règles de pare-feu - le tout de manière cohérente et prévisible, et sans dépendre de LXC ou de tout autre package utilisateur. Cela réduit considérablement le nombre de pièces mobiles et isole Docker des effets secondaires introduits dans les versions et distributions de LXC. En fait, libcontainer a tellement amélioré la stabilité que nous avons décidé d'en faire la valeur par défaut. En d'autres termes, à partir de Docker 0.9, LXC est désormais facultatif.

3.3 Fonctionnalité Swarm

Un Swarm est un cluster de machines qui exécutent le moteur Docker qui est un outil client-serveur. Ce cluster sera d'abord constitué d'un manager (il peut y avoir plusieurs managers mais un seul est élu leader des managers) puis ensuite nous joignons au Swarm d'autres machines qui constitueront des worker, chaque machine qui rejoint le Swarm est considérée comme une node.

Le rôle de ce manager est d'orchestrer et de déployer les différents services (terme qui désigne des conteneurs dans le contexte d'un Swarm).

Le rôle des worker est seulement de fournir de la capacité, elles ne peuvent pas ordonner à d'autres machines ce qu'elles peuvent faire ou non contrairement aux machines managers.

Une fois un conteneur (maintenant appelé service dans le contexte Swarm) lancé, une ou plusieurs tâches (en fonction de ce qu'on a défini) sont exécutées sur les nodes disponibles. Cette fonctionnalité de Docker contient également un équilibreur de charge qui permet de répartir les tâches afin de pouvoir mieux supporter la charge.

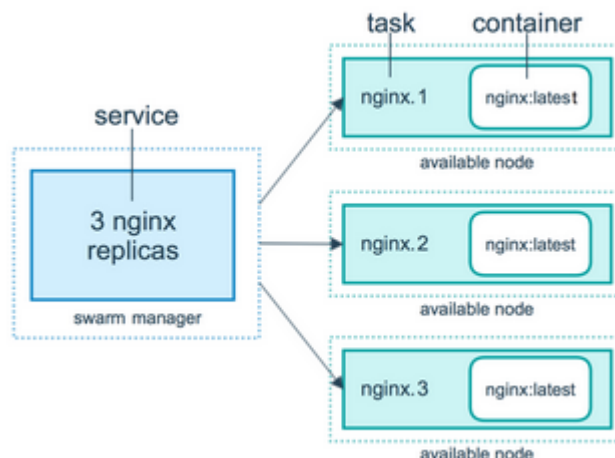


FIGURE 3.3 – Schéma d'une réplication avec Docker Swarm

Le fonctionnement de Swarm est linéaire. Par-là, nous sous-entendons que le Swarm manager passe par différents états lors de la création de service (assigné, préparé, en cours d'exécution). Le manager fait office d'orchestrateur / planificateur pour l'usage général du cluster où il a la possibilité d'équilibrer la charge sur les différentes nodes à sa disposition. Mais le manager ne fait pas cela bêtement, en effet lorsqu'un incident dans la création du service intervient, le manager crée une nouvelle tâche et supprime le conteneur "corrompu" en fonction de l'état spécifié pour le service par le manager.

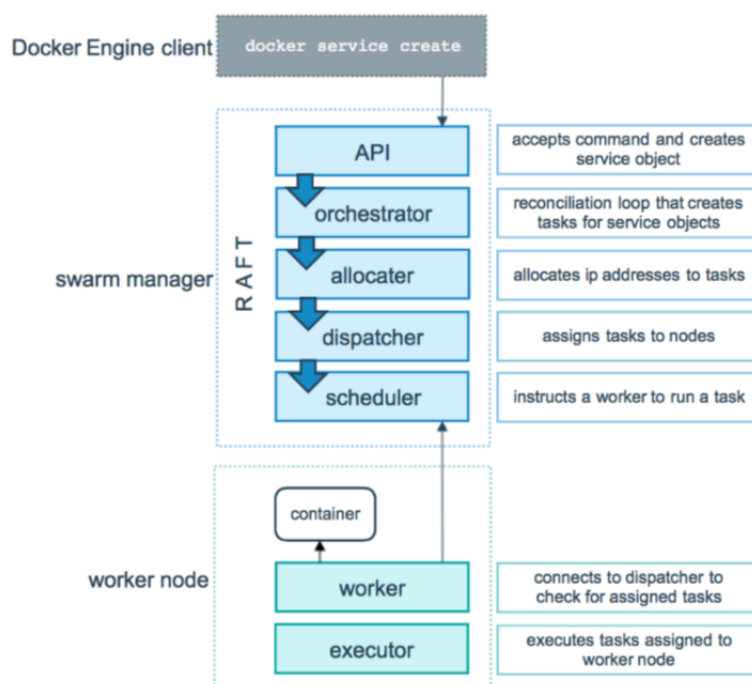


FIGURE 3.4 – Schéma des différentes étapes par lesquelles Docker Swarm passe pour la création d'un service

Chapitre 4

Métrologie

4.1 Schéma d'architecture général

Ci-dessous, nous allons voir un cas général d'une solution de monitoring d'un cluster Docker, représentatif de ce qui pourrait être utilisé en entreprise.

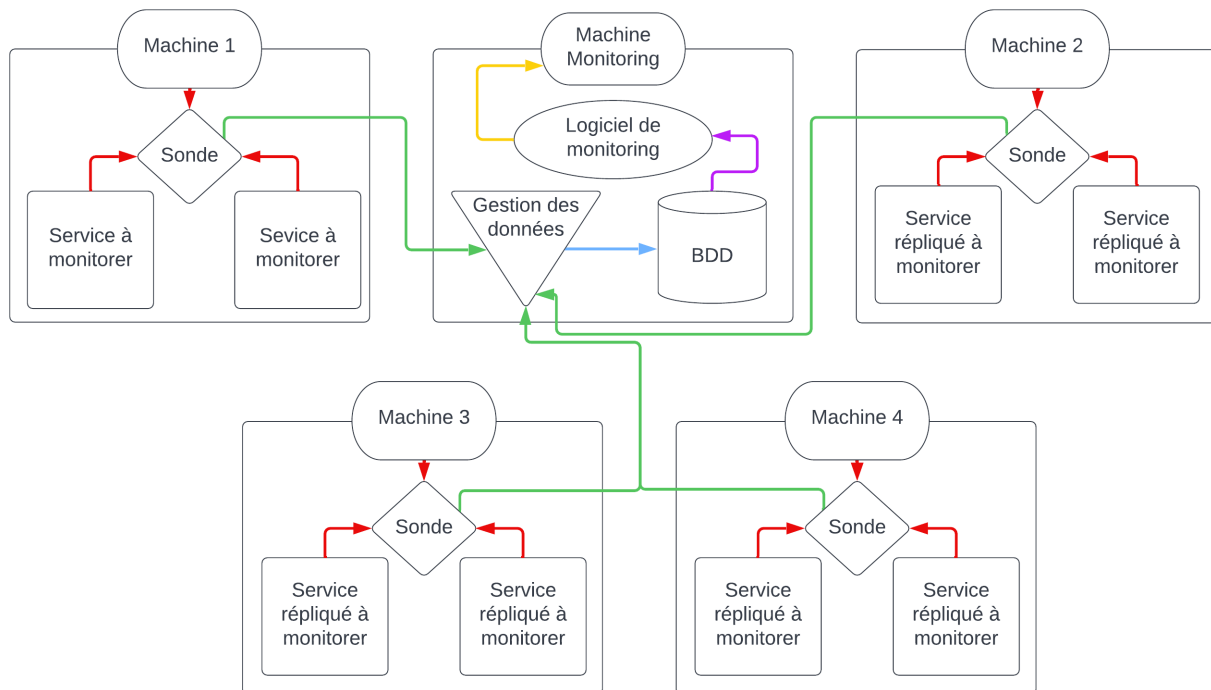


FIGURE 4.1 – Schéma du cas général d'un monitoring clusterisé - légende : rouge = récupération des métriques par la sonde, vert = récupération des métriques de la sonde par le logiciel de gestion des données, bleu = transfère des données vers la BDD, violet = envoi des données au logiciel de monitoring, jaune = visualisation des données par la machine monitoring

Dans le cas où on veut monitorer une application ou une machine il faut avant tout un logiciel qui va permettre d'analyser l'activité de la cible, une sonde, la sonde sera souvent adaptée à cette cible s'il s'agit d'une application, application web, application conteneurisée

ou une machine physique, quelques exemples de ces sondes sont cAdvisor ou Node Exporter. Ensuite si on veut pouvoir visualiser des métriques de plusieurs applications en même temps, il faut un logiciel de monitoring comme Grafana mais il n'est pas capable nativement de lire les métriques de toutes les sondes. Si le logiciel de monitoring ne peut pas lire les métriques qu'il reçoit, il faut un logiciel intermédiaire de gestion des données comme Prometheus qui va être capable de lire la plupart des métriques pour les rendre à un format accessible aux outils de monitoring. Et pour finir, si on veut avoir accès aux données sur le long terme une base de données sera généralement mise en place afin de stocker les métriques.

4.2 Communication entre logiciels

Au fil des technologies que nous présentons dans ce rapport nous mentionnons plusieurs fois la notion d'endpoint, étroitement liée aux notions de sockets et d'API qui permettent de communiquer des ressources entre nos solutions. C'est pourquoi nous avons décidé d'y consacrer une partie.

Souvent en informatique, pour permettre à deux systèmes qui n'ont pas la même architecture de communiquer entre eux pour échanger des ressources, nous utilisons une API. Celle-ci fonctionne par demandes et réponses, elle va définir ce qui est attendu d'un côté comme de l'autre.

Mais cette communication n'est pas possible sans la notion d'endpoints, ou points de terminaisons, puisque chacun représente une extrémité d'un canal de communication. Concrètement, ce sont une combinaison d'une IP et d'un numéro de port. C'est l'endroit où les API envoient les demandes et où réside la ressource qu'on souhaite échanger.

Enfin, nous avons les sockets. Ce sont eux qui permettent une communication interprocessus bidirectionnelle entre deux endpoints. Dans le cas d'une communication distante, nous avons recours à des sockets tcp. Dans le cas contraire, nous avons aussi des sockets unix qui utilisent le système de fichiers local.

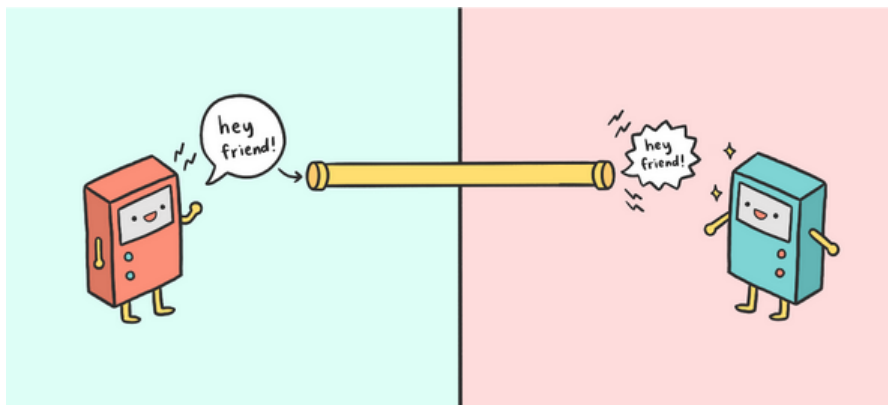


FIGURE 4.2 – Illustration ludique d'une communication par API

Chapitre 5

Outils de métrologie

5.1 Les Sondes

Maintenant que les bases ont été posées, nous allons présenter plusieurs outils viables pour aboutir à une quelconque solution de métrologie. Commençons par les sondes.

5.1.1 cAdvisor

Aujourd'hui, les conteneurs sont largement utilisés du développement jusqu'en production. En revanche, les fonctionnalités fournies par les solutions de conteneurisation comme Docker ne permettent pas d'anticiper les dysfonctionnements de son environnement de production. C'est alors que de nouvelles solutions sont nées pour répondre à des besoins nouveaux émergeant de cette constatation, l'une d'entre elles se nomme cAdvisor.



FIGURE 5.1 – Logo de cAdvisor

C'est un outil qui va nous permettre de récupérer les métriques des conteneurs qui s'exécutent sur notre machine, ou node Docker dans notre cas. Au sens général les métriques représentent une compilation de mesures issues des propriétés techniques ou fonctionnelles d'un logiciel, elles peuvent être classées sous différentes catégories mais dans notre cas il est question de qualité applicative. Plusieurs indications quant aux ressources utilisées s'offrent à nous (cpu, ram, network pour chaque conteneur). Enfin, si l'on souhaite exporter ces données, cAdvisor expose un endpoint (<http://cadvisor:8080/metrics>) regroupant l'ensemble des métriques des conteneurs.

5.1.2 Node Exporter

Désormais, nous sommes en capacité de collecter les métriques de nos conteneurs mais pas encore celles de nos hôtes Linux. Pour répondre à cette problématique nous utiliserons la solution Node Exporter, fournie par Prometheus.



FIGURE 5.2 – En-tête de la page GitHub officielle de Node Exporter

Tout comme cAdvisor, un endpoint sera exposé (`http://localhost:9100/metrics`) regroupant les métriques de notre machine (processeur, mémoire, disques, systèmes de fichiers, suivi du réseau). Nous déploierons également cette solution sous forme de service au sein de notre cluster (une instance sur chaque node).

5.2 Prometheus

Prometheus est un projet open-source issu de la plateforme musicale SoundCloud, son objectif est de monitorer les métriques de fonctionnement des serveurs et de créer une gestion d'alertes en fonction de seuils considérés critiques. Il repose sur un modèle d'extraction de endpoints HTTP pour enregistrer les données collectées en temps réel.



FIGURE 5.3 – Logo de Prometheus

5.2.1 Exportateurs Prometheus

Ces données en question sont collectées par des exportateurs (dont cAdvisor et Node Exporter font partie). Prometheus en propose un large catalogue dans sa documentation, certains sont maintenus officiellement par Prometheus et d'autres proviennent de contributions tierces (il existe aussi des logiciels tiers qui exportent directement leurs métriques au format Prometheus, comme Kubernetes par exemple). L'exportateur est composé de deux éléments :

- Des capacités logicielles qui vont générer des données métriques
- Un serveur HTTP qui expose la métrique générée disponible via un point de terminaison particulier.

Enfin, le serveur Prometheus peut lire et capturer (ou *scrape* pour les anglophones) les métriques qui auront été publiées selon un format spécifique.

Il existe plusieurs types de métriques Prometheus mais les principaux sont les suivants :

- Le compteur, une métrique cumulative représentant un compteur croissant, dont la valeur peut seulement augmenter ou être remise à zéro au redémarrage.
- La jauge, une métrique qui représente une valeur numérique qui peut monter et descendre arbitrairement.

5.2.2 Fonctionnement de Prometheus

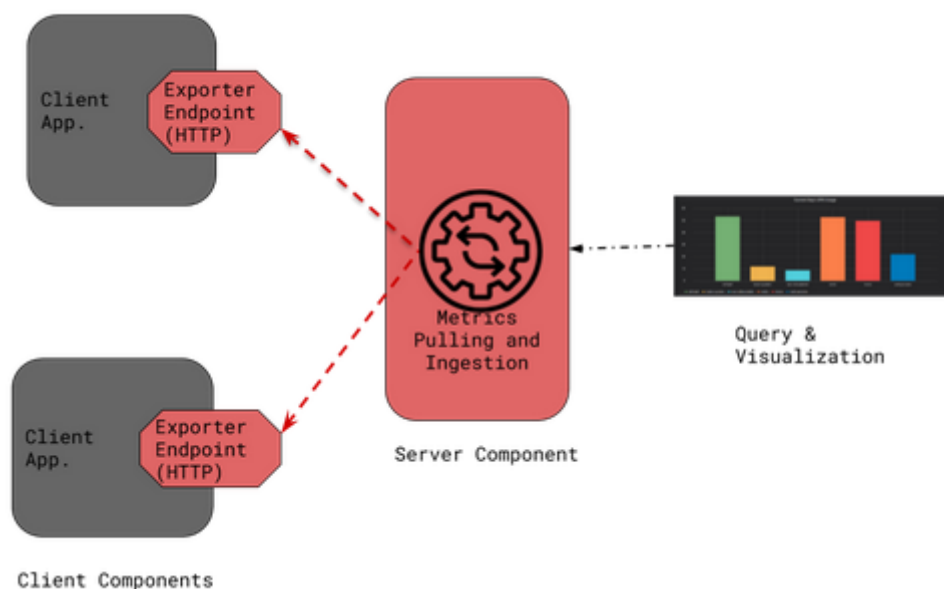


FIGURE 5.4 – Schéma d'architecture de base d'un environnement Prometheus avec un composant serveur, deux composants client et un système de visualisation externe

Pour la collecte de données Prometheus est basé sur le pull, un concept largement répandu dans différents secteurs qui a pour principe de créer un flux de travail dans lequel le travail n'est effectué que s'il y a une demande pour celui-ci. Le but est de réduire les gaspillages dans tout processus de production. C'est donc Prometheus qui se connecte aux agents pour récupérer les métriques et pas l'inverse.

- Les exportateurs collectent leurs métriques et les exposent en HTTP
- Le serveur Prometheus va récupérer les métriques auprès des exportateurs, et les stocker sur disque ou ailleurs (via une fonction `remote_write`)
- Il va exposer un endpoint en HTTP pour permettre aux autres composants d'exécuter des requêtes PromQL pour accéder aux métriques

Enfin, nous avons aussi l'interface web qui nous permet de voir l'état de notre instance Prometheus mais aussi de requêter les métriques. En revanche, Grafana apporte plus de fonctionnalités en termes de visualisation.

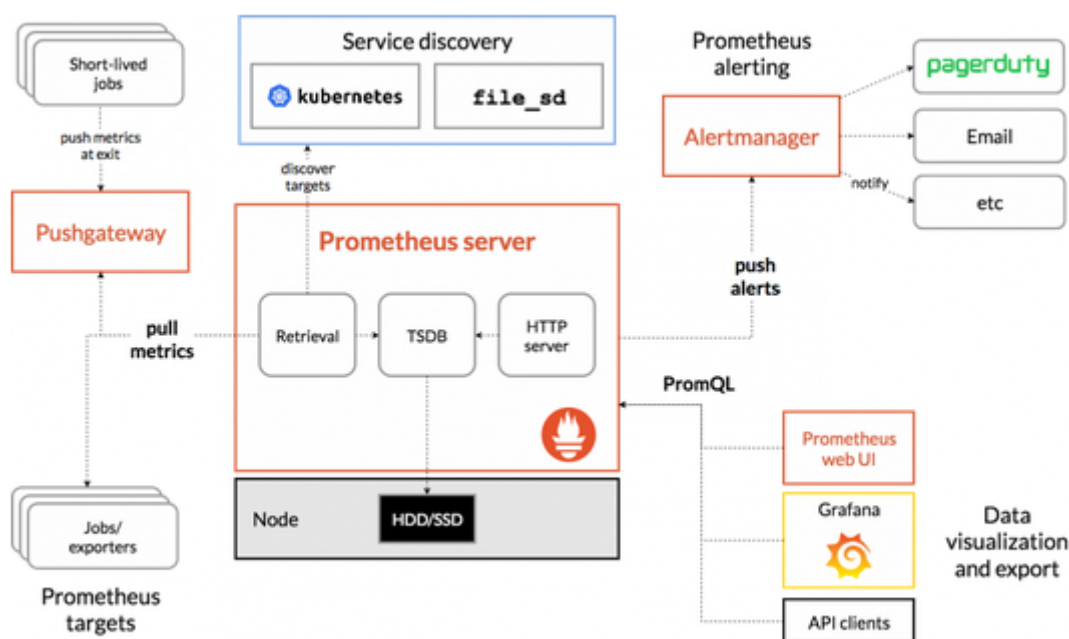


FIGURE 5.5 – Schéma présentant le fonctionnement global de Prometheus

5.2.3 Base de données de séries temporelles

Pour stocker les métriques, Prometheus comprend une base de données conçue spécialement pour gérer des données temporelles. Elle est à dénoter d'une base de données relationnelle, ou SGBD, dans laquelle des tables contiennent des colonnes et des lignes où chacune d'entre elles représente une entrée dans notre table. Dans une base de données de séries temporelles, les données sont toujours stockées dans des collections mais elles sont désormais agrégées au fil du temps, c'est-à-dire qu'un horodatage est associé à chaque entrée.

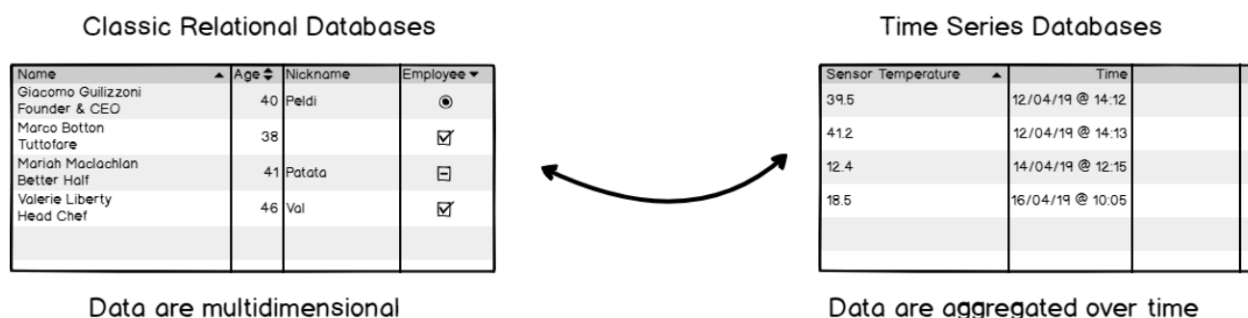


FIGURE 5.6 – Schéma comparatif des principes d'une base de données relationnelle et d'une base de données de séries temporelles

L'avantage des bases de données de séries temporelles repose dans leur conception, pensée pour insérer des données de manière rapide et efficace. Au contraire, les SGBD perdent vite en performances au fil des nouvelles entrées qui s'accumulent (la présence d'index dans les tables fait également pencher la balance) et il devient de plus en plus difficile de lire nos

données avec la charge qui augmente. Ainsi, les deux types de bases de données démarrent sur des performances assez uniformes, mais avec de gros volumes de données celles-ci déclinent rapidement pour les SGBD tandis que les bases de données de séries temporelles parviennent à garder un taux d'insertion plutôt constant en raison de leur optimisation.

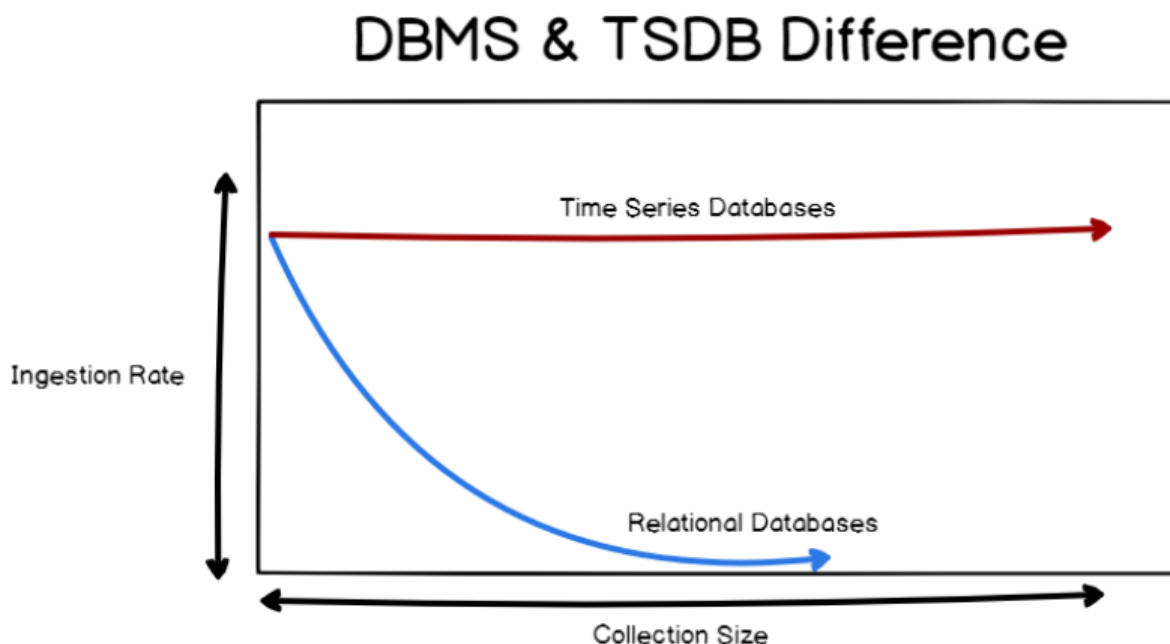


FIGURE 5.7 – Schéma comparatif entre les performances des bases de données relationnelles et des bases de données de séries temporelles

En revanche, la limite de Prometheus et sa base de données se trouve dans la conservation des données. En effet, il est fort probable que les données que stocke Prometheus ne nous intéressent pas sur le long terme. Là où nous voulons en venir est que ces données vont continuer de s'accumuler au fil du temps et prendre beaucoup d'espace de stockage. Ainsi, à moins que nous soyons intéressés par d'anciennes données, nous allons vouloir les supprimer pour des raisons de coûts. C'est alors qu'une solution nommée InfluxDB prend tout son intérêt.

5.3 InfluxDB

5.3.1 Présentation

InfluxDB est une base de données open-source spécialisée dans le stockage de séries temporelles. Ces séries temporelles peuvent correspondre à n'importe quelles mesures (exemples : température, utilisation processeur, espace disque utilisé, ...) prises régulièrement par des sondes. Cependant, cette technologie s'est améliorée et est devenue un outil performant de supervision, tout comme Prometheus.



FIGURE 5.8 – Logo d'InfluxDB

Dans le cadre de ce projet tutoré, nous utilisons InfluxDB sous sa version 1.8. C'est-à-dire qu'elle n'est simplement considérée que comme une base de données stockant les métriques issues de la métrologie, et rien d'autre.

Cette technologie vient donc substituer la base de données de séries temporelles déjà présente sur Prometheus. Le paramétrage de cette substitution se fait essentiellement dans la configuration de Prometheus. Préalablement, il est possible de dire à la première instance d'InfluxDB de créer une première base, afin que Prometheus puisse y écrire et lire.

Une des fonctionnalités essentielles pour ce projet que nous allons détailler par la suite est l'échantillonnage des données, ce qui va nous permettre de garder un grand nombre d'informations concernant la métrologie, en consommant le moins d'espace disque possible.

5.3.2 Fonctionnement

Cette technologie nous offre les fonctionnalités suivantes :

- Premièrement, afin de pouvoir installer et configurer proprement InfluxDB, celui-ci propose un *shell* (invité de commande). Ce shell attend en entrée des requêtes InfluxQL (proche du SQL) ainsi que quelques autres commandes pour la configuration globale du logiciel. Une autre façon de faire est de modifier le fichier de configuration d'InfluxDB (ou par les variables d'environnements sous Docker) pour créer l'utilisateur initial, son organisation, ainsi qu'une première base de données.
- Comme dit précédemment, InfluxDB utilise le langage InfluxQL, puis Flux depuis sa version 2. InfluxQL est proche du langage SQL, ce qui n'est pas le cas de Flux.
- InfluxDB met en place une *API* lui permettant de recevoir des requêtes. Elle est accessible par des requêtes HTTP en accédant aux différents endpoints. Les informations nécessaires pour lire et écrire les données sont à la fois dans l'URL de la requête

mais aussi dans l'en-tête HTTP. C'est de cette façon que Prometheus peut se servir d'InfluxDB comme stockage distant.

- *Authentication par HTTP* : Il y a deux manières de se connecter à l'API. La première est l'authentification par un *jeton*, qui doit être transmis au client distant. Celui-ci le renseigne dans son en-tête HTTP. La deuxième façon de s'identifier est de renseigner *un nom d'utilisateur et un mot de passe* dans l'en-tête HTTP ou directement dans l'URL.
- Pour répondre au besoin d'une supervision à différentes échelles de temps, nous pouvons mettre en place des *politiques de rétention* ainsi que des *requêtes continues* (permet de définir une résolution de temps aux données enregistrées, mais surtout la durée de vie de ces données). C'est sur cette fonctionnalité qu'est possible l'échantillonnage des données. Nous détaillons plus en détails la façon dont nous pouvons conserver les données dans notre cas d'expérimentation.

5.4 Grafana

Grafana est un logiciel libre qui permet de créer des graphiques et des tableaux de bord venant de sources variées avec des données brutes. Il peut également avoir comme source des bases de données temporelles comme InfluxDB, ce qui en fait un outil de monitoring centralisé très complet et efficace.



FIGURE 5.9 – Logo de Grafana

5.4.1 Fonctionnalités de Grafana

Grafana est la partie visible et visuel du monitoring, depuis celui-ci on pourra :

- Sélectionner des sources de données variées à exploiter
- Les organiser dans des dashboards que l'on peut soit créer nous-même ou les récupérer dans le Grafana Lab
- Un menu qui répertorie tous les dashboards pour pouvoir les partager, faire des snapshots,

5.4.2 Data Source

Pour récupérer les données Grafana utilise les data source qui permettent de configurer rapidement une source de données à partir de l'endpoint d'une des nombreuses applications de gestion des données existante comme Azure Monitor, AWS CloudWatch, PostgreSQL, Prometheus ou encore InfluxDB, chaque data Source se paramètre de manière différente en fonction de ses besoins ce qui donne un aspect très versatile et inter-compatible à Grafana.

5.4.3 Les Dashboards

Une fois notre Data Source crée on peut ensuite configurer nos dashboards, un dashboard est un menu regroupant des graphiques très visuels appeler panel qui vont permettre de voir l'état des métriques reçus. Chaque dashboard va pouvoir posséder plusieurs panels pour visualiser les informations les plus intéressantes. Si on démarre d'un dashboard vide aucun panel ne sera visible, il faudra crée ses propres panels ce qui peut être complexe et assez fastidieux. Une autre solution consiste à utiliser des dashboards préconçus disponible sur le GrafanaLabs en fonction de nos besoins, le GrafanaLabs est un site qui entre autres permet à la communauté d'échanger ses dashboards en fonction de leur utilité.

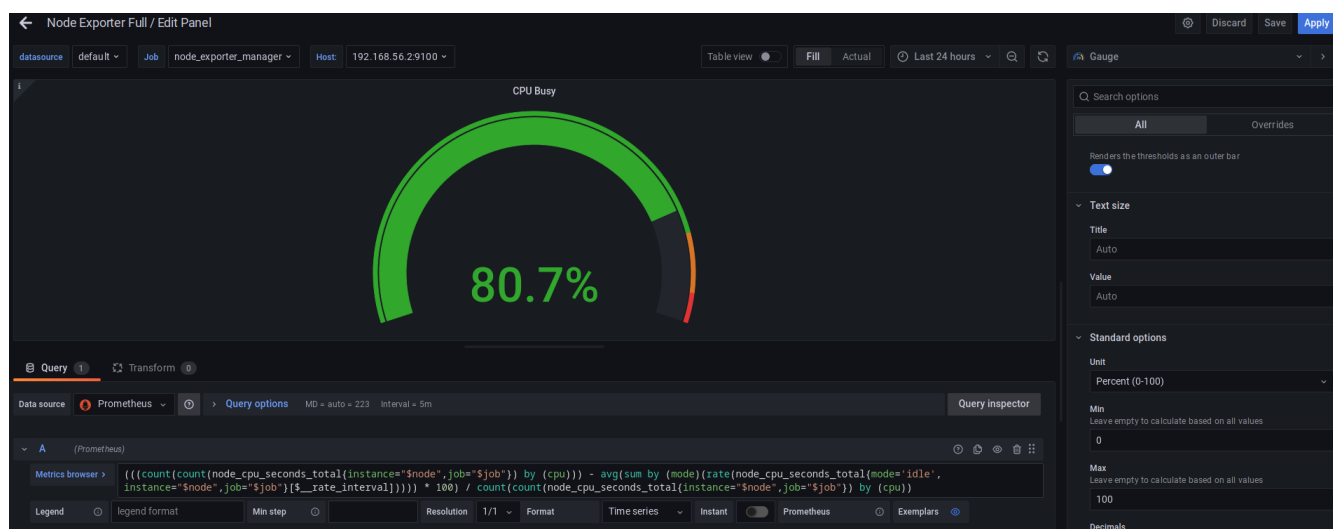


FIGURE 5.10 – Le menu d'édition d'un panel Grafana

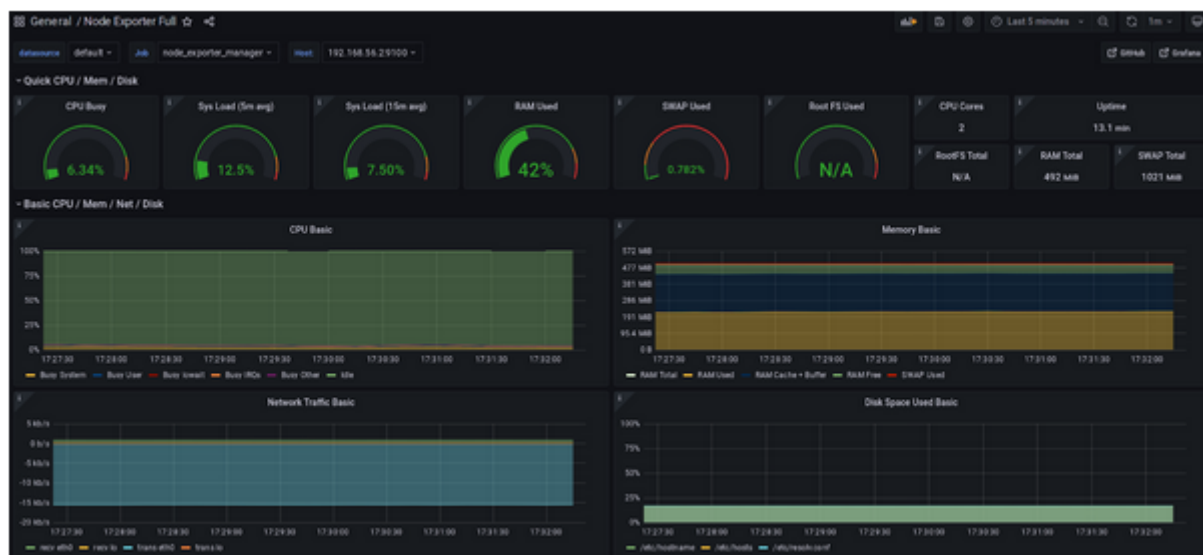


FIGURE 5.11 – Capture d'écran du dashboard des métriques de Node Exporter

Chapitre 6

Les applications web

6.1 WordPress



FIGURE 6.1 – Logo de WordPress

WordPress est un CMS (système de gestion de contenu) open-source et est le CMS le plus utilisé au monde, 41,5 % des sites web dans le monde utilisent WordPress. Il est si utilisé car cette solution gratuite permet de mettre en place très rapidement un site avec des objectifs variés comme un blog, un site vitrine ou un site de ventes sans aucune connaissances en programmation mais en ayant tout de même une personnalisation poussée grâce aux nombreuses extensions développées autour de WordPress par des entreprises professionnelles ou encore des templates qui permettent d'avoir une identité visuelle pré-faite et préconçus en fonction de nos besoins comme l'e-commerce ou la promotion d'un produit.

6.2 MailCow

Typiquement, sur un cluster Docker, nous pouvons également héberger un serveur mail. C'est pourquoi nous avons choisi de présenter MailCow.



FIGURE 6.2 – Logo de MailCow

MailCow est une suite de serveurs de messagerie basée sur plusieurs logiciels open-source. Nous l'avons mis en place via sa version Dockerized qui utilise plusieurs conteneurs Docker, dont chacun contiendra une seule application, liés par un réseau bridge. Ces conteneurs utilisent également des volumes pour conserver des données dynamiques. Parmi les logiciels utilisés par MailCow, nous pouvons citer :

- Dovecot
- Oletools via Olefy
- Memcached
- Redis
- MariaDB
- Unbound
- PHP
- Postfix
- Let's Encrypt
- Nginx
- Rspamd
- SOGo
- Netfilter
- Un Watchdog pour la surveillance de base

Une fois que tous les conteneurs sont lancés et fonctionnels, nous pouvons accéder à l'interface utilisateur intégrée qui va nous permettre d'administrer notre instance de serveur de messagerie. D'ailleurs celle-ci nous fournit un accès séparé pour l'administrateur de domaine et l'utilisateur de la boîte aux lettres. Parmi les fonctionnalités de MailCow nous retrouverons celles-ci :

- Prise en charge de DKIM et ARC
- Blacklists et Whitelists par domaine et par utilisateur
- Gestion des Spam par utilisateur (reject spam, mark spam, greylist)
- Autoriser les utilisateurs de boîtes aux lettres à créer des alias de spam temporaires
- Ajouter des balises de courrier au sujet ou déplacer le courrier dans un sous-dossier (par utilisateur)

- Autoriser les utilisateurs de boîtes aux lettres à basculer entre l'application TLS entrante et sortante
- Autoriser les utilisateurs à réinitialiser les caches d'appareils SOGo ActiveSync
- imapsync pour migrer ou extraire régulièrement des boîtes aux lettres distantes
- Authentification à deux facteurs : Yubikey OTP, U2F USB ou TOTP
- Ajouter des domaines, des boîtes aux lettres, des alias, des alias de domaine et des ressources SOGo
- Ajouter des hôtes sur liste blanche pour transférer le courrier vers MailCow
- Intégration d'un Fail2ban-like
- Système de quarantaine
- Analyse antivirus
- Surveillance de base intégrée

6.3 Gatling

Gatling est un outil open-source de test de charge et de performance pour les applications web. Il permet de simuler des réelles charges utilisateurs sur une infrastructure contenant un ou plusieurs services web. Les tests de charge, appelés simulations par Gatling, sont écrits sous forme de programmes en Java, Kotlin, ou Scala.



FIGURE 6.3 – Logo de Gatling

Les simulations introduisent la notion de scénario. Un scénario est une succession logique d'évènements utilisateurs (appui sur un bouton, chargement de média, création d'un commentaire,) qu'un utilisateur est amené à faire sur une application web. Parallèlement, dans ces programmes sont écrits les en-têtes HTTP utilisées dans les différents scénarios. On a donc un contrôle total de l'interaction entre le client et le serveur web.

La dernière partie de ces programmes sont le paramétrage de la simulation. Il s'agit d'introduire un nombre quelconque d'utilisateurs pour chaque scénario. Les méthodes pour introduire les utilisateurs sont nombreuses et précises, donc un haut niveau de personnalisation est possible sur ce point-là. Par exemple, on est capable d'insérer un certain nombre d'utilisateur par seconde de façon constante, progressive ou dégressive pendant une période voulue.

Finalement, les résultats des tests de charge, donc de l'exécution d'un programme, est exporté à la fin de la simulation. Le format du résultat est en HTML, et il est plutôt plaisant à analyser.



FIGURE 6.4 – Exemple de rendu sur Gatling

En soit, Gatling n'est pas un outil nécessaire pour la métrologie et sa supervision, mais c'est un bon moyen de simuler des charges réelles qu'une entreprise supporte au quotidien. Sa mise en place est externalisée à l'infrastructure dont nous voulons tester les performances, puisque Gatling simule des utilisateurs externes.

Chapitre 7

Cas d'expérimentation

Désormais nous allons mettre en place un cadre plus restreint dans lequel nous chercherons à approcher le problème général.

7.1 Schéma d'architecture

Pour ce faire, voici les schémas qui permettront de mieux se situer avant de suivre le cheminement des différentes technologies qui nous permettront d'aboutir à notre solution de métrologie :

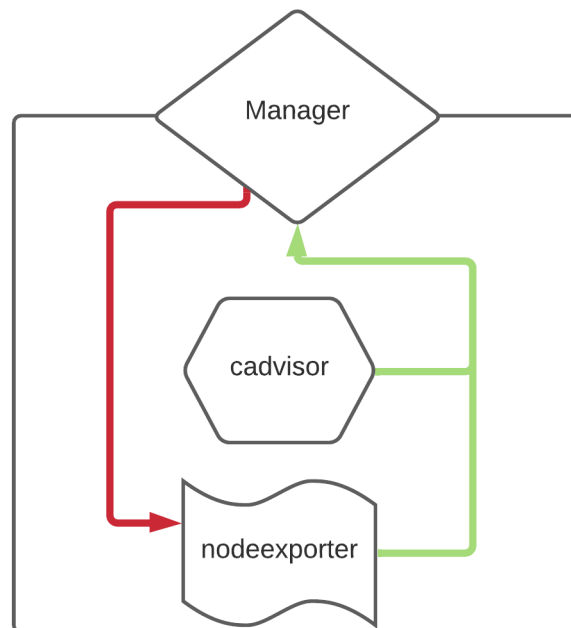


FIGURE 7.1 – Schéma de notre machine manager - légende : vert = logiciel que l'on peut visualiser, rouge = trajet des métriques

La machine manager est en soit très simple, la machine créatrice du Docker Swarm qui va permettre de répliquer les conteneurs présents dans les autres machines du Swarm comme celle de worker par exemple. Malgré sa fonction importante ses composantes sont plutôt simples, il n'y a pas d'application importante à part le Swarm, on le monitore donc

avec cAdvisor pour récupérer les métriques des conteneurs et Node_exporter pour avoir les métriques de la machine pour être sûr que lors d'une réplication des services tout se passe normalement.

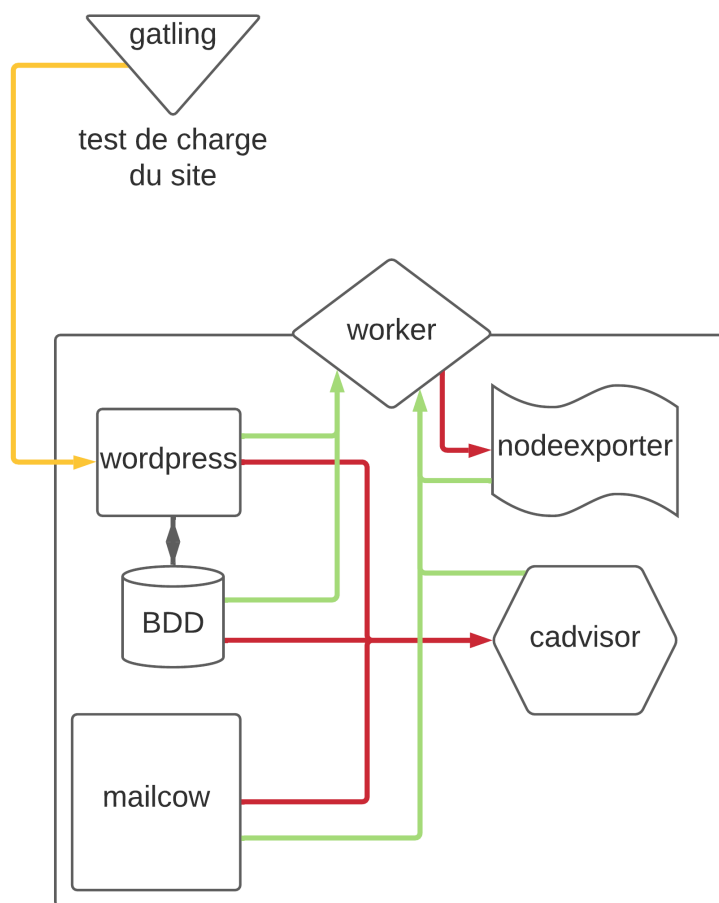


FIGURE 7.2 – Schéma de notre machine worker - légende : vert = logiciel que l'on peut visualiser, rouge = trajet des métriques

La machine worker est, comme son nom l'indique, la machine qui aura pour objectif d'exécuter les services orchestrés par la machine manager, nous avons dans notre cas deux services standard que l'on peut retrouver dans n'importe quelle entreprise. Un site web WordPress avec une base de données et un serveur de mail MailCow. Comme pour le manager cAdvisor va monitorer les conteneurs et Node Exporter va nous donner une vision globale de la machine, nous avons en dehors de la machine Gatling qui va nous permettre de faire les différents tests de charges pour tester l'infrastructure et le site WordPress.

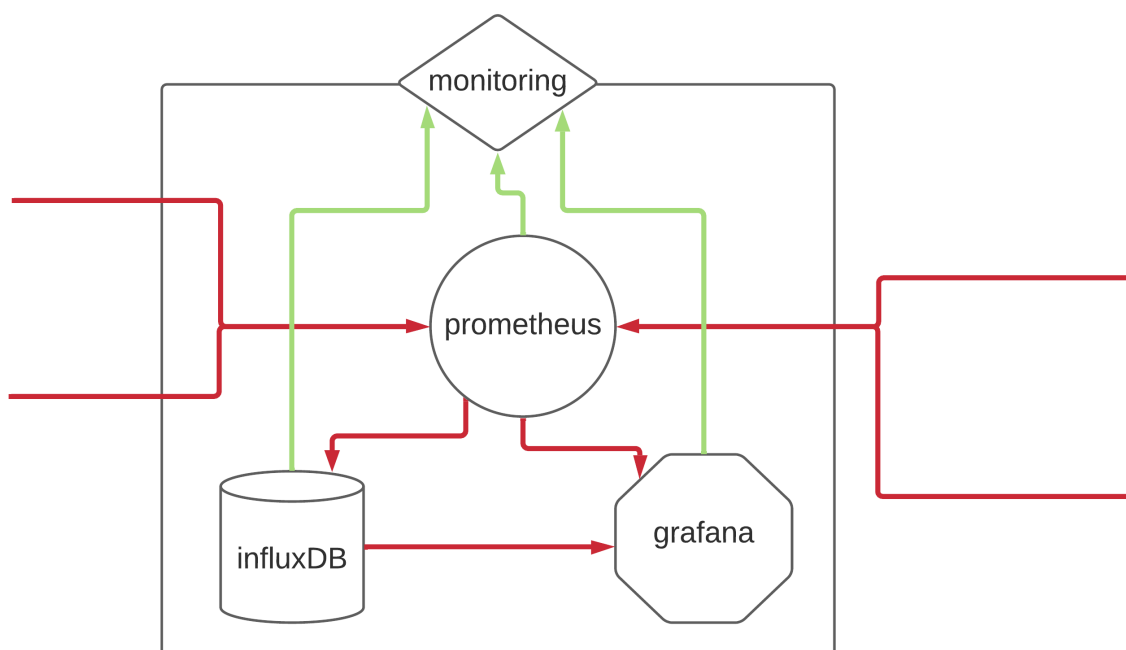


FIGURE 7.3 – Schéma de notre machine monitoring - légende : vert = logiciel que l'on peut visualiser, rouge = trajet des métriques

La machine monitoring a encore une fois un nom explicite, extérieur au Docker Swarm cette machine va regrouper toutes les solutions de monitoring, Prometheus qui va recueillir les métriques des machines worker et manager qui va ensuite les stocker et les relire depuis InfluxDB, en les visualisant grâce à Grafana. InfluxDB va d'ailleurs stocker les données avec des règles de rétentions particulières pour les besoins de supervision de données à longues rétention. Avec tout ce système, Grafana aura accès aux données avec trois temporalités différentes. La première, à court terme, est directement envoyée par Prometheus. Les deux autres, moyens et long terme, sont stockés et gérés par InfluxDB puis envoyés. Grafana nous permet donc de voir toutes nos données en fonction de la temporalité voulue.

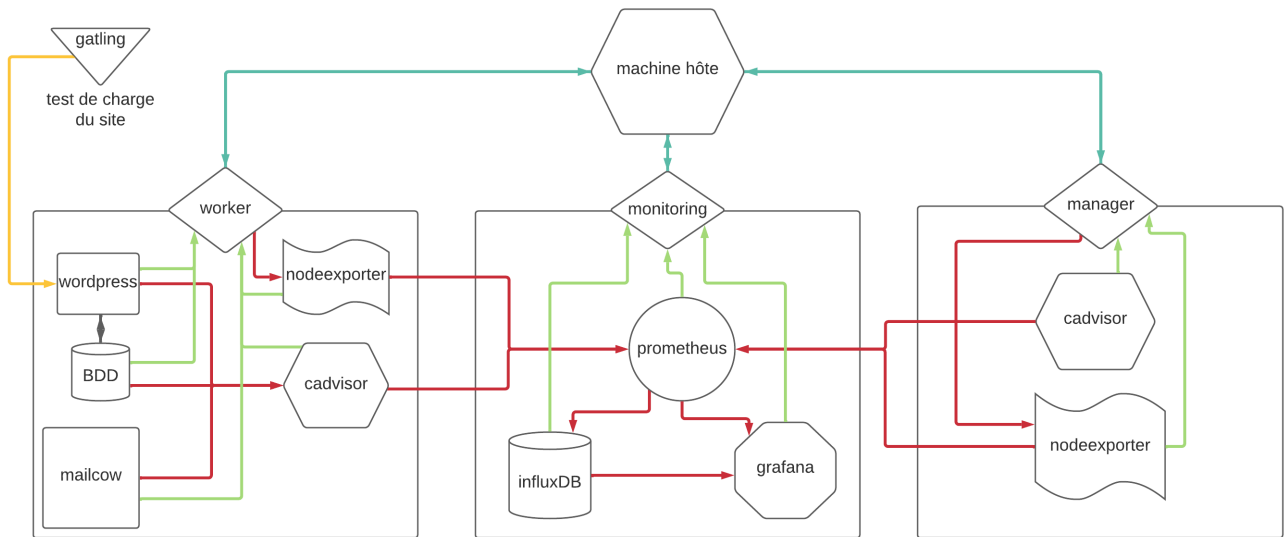


FIGURE 7.4 – Schéma de notre infrastructure légende : vert = logiciel que l'on peut visualiser, rouge = trajet des métriques, jaune = test de Gatling sur WordPress, turquoise = accès entre la machine hôte et les machines virtuelle

Voici le schéma global de notre infrastructure, nous avons donc nos trois machines : worker et manager qui font partie du Swarm et monitoring qui est extérieure au Swarm. Les sondes des machines du Swarm (Node Exporter et cAdvisor) récupèrent les métriques des conteneurs et des machines puis les envoient à Prometheus qui se trouve dans la machine monitoring. Prometheus peut ensuite envoyer ses métriques à nos outils de monitoring, il les envoie à Grafana qui va pouvoir afficher les métriques directement. Pour pouvoir afficher les métriques sur le moyen et le long terme Prometheus envoie les métriques à InfluxDB qui va traiter et stocker les données qui pourra ensuite les envoyer à Grafana qui pourra donc visualiser les métriques sur trois temporalités différentes.

7.2 Mise en place

7.2.1 Docker Swarm

Pour mettre en place notre cluster docker Swarm tout d'abord nous initions sur notre machine qui sera Manager un token unique qui aura pour but de reconnaître cette en tant que Manager et aussi d'utiliser ce token pour ajouter les machine dites worker à notre Swarm.

Voici à quoi devrait ressembler le token en question :

```
SWMTKN-1-0rwlvo0d11hxmivbeh41qe2rfgw00y13msi9msdpq9b04bh-axhj2igyw7z56o5erdycjlwdp
192.168.56.2 :2377
```

Maintenant que le token est en notre possession nous pouvons ajouter notre machine worker à notre Swarm. Nous disposons donc de nos machines dans le Swarm, nous pouvons alors commencer la procédure de déploiement de services. Mais avant cela, nous activons les fonctionnalités expérimentales (réservées à des environnements de test) car la commande docker deploy ne fonctionne que dans cet environnement expérimental spécifique. Pour cela, il faut éditer un fichier daemon.json dans le répertoire /etc/docker et mettre le paramètre

"expérimental" à la valeur true, puis relancer le service docker. Afin de lancer nos différents services pour notre infrastructure, nous avons un fichier nommé docker-compose.yml qui permet d'initier tous nos services avec les paramètres nécessaires, comme les images des services directement cherchées sur le Docker Hub, les machines sur lesquelles nous voulons lancer nos services, les volumes, les ports d'usage, et les dépendances. Grâce à ce fichier les services vont être lancés sur les machines seulement à partir de la machine Manager.

Avec la commande qui suit :

```
vagrant@manager:~$ sudo docker deploy --compose-file docker-compose-manager.yml stackdemo
Ignoring unsupported options: restart

Ignoring deprecated options:
container_name: Setting the container name is not supported.

Creating network stackdemo_default
Creating service stackdemo_node-exporter
Creating service stackdemo_db
Creating service stackdemo_wordpress
Creating service stackdemo_cadvisor
Creating service stackdemo_redis
```

FIGURE 7.5 – Capture de la création des services avec la machine Manager

Nous pouvons constater que les services sont bien lancés via le manager :

```
vagrant@manager:~$ sudo docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
ke8twvl1ouo8	stackdemo_cadvisor	global	2/2	gcr.io/cadvisor/cadvisor:latest	*:8080->8080/tcp
l9qhic8uqx2g	stackdemo_db	replicated	1/1	mysql:5.7	
p4jwn5m8wzyd	stackdemo_node-exporter	global	2/2	prom/node-exporter:latest	*:9100->9100/tcp
pdwa8pyr5bjz	stackdemo_redis	global	2/2	redis:latest	*:6379->6379/tcp
p4yi6f4970km	stackdemo_wordpress	replicated	1/1	wordpress:latest	*:8000->80/tcp

```
vagrant@manager:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
0b1ad827e0e1	redis:latest	"docker-entrypoint.s..."	20 minutes ago	Up 20 minutes	6379/tcp
3ad8f51ddb4	gcr.io/cadvisor/cadvisor:latest	"/usr/bin/cadvisor -..."	20 minutes ago	Up 20 minutes (healthy)	8080/tcp
d8ebb273cb8f	prom/node-exporter:latest	"/bin/node_exporter"	20 minutes ago	Up 20 minutes	9100/tcp
18879ce99812	dockersamples/visualizer	"/sbin/tini -- node -..."	34 minutes ago	Up 34 minutes (healthy)	0.0.0.0:5000->8080/tcp

FIGURE 7.6 – Capture des services lancé sur la machine Manager

Et nous pouvons constater que les services sont lancés sur la machine worker :

```
vagrant@worker:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
37497f4d9982	wordpress:latest	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes	80/tcp
21d33c2909c9	gcr.io/cadvisor/cadvisor:latest	"/usr/bin/cadvisor -..."	14 minutes ago	Up 14 minutes (healthy)	8080/tcp
debeff6ab60c	redis:latest	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes	6379/tcp
d945de34bd87	mysql:5.7	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes	3306/tcp, 33060/tcp
d9ebbb8f63d4	prom/node-exporter:latest	"/bin/node_exporter"	14 minutes ago	Up 14 minutes	9100/tcp

FIGURE 7.7 – Capture des services lancé sur la machine worker

Afin de nous assurer que nos services sont bien lancés sur les machines voulues nous ajoutons un service de visualisation sur notre machine manager et nous pouvons bel et bien constater que nos services sont bien attribués aux bonnes machines :

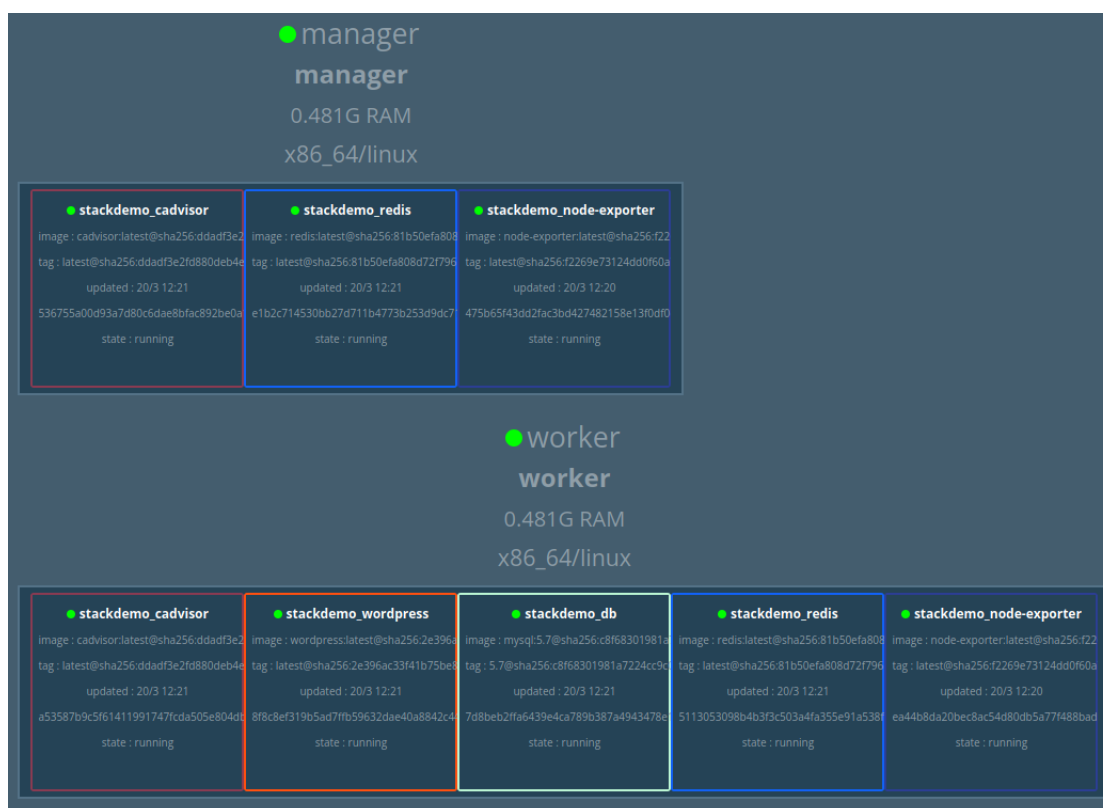


FIGURE 7.8 – Capture du service de visualisation

7.2.2 Sondes

Pour collecter les données de nos applications web, nous allons donc mettre en place des sondes. Étant donné que nous sommes toujours dans le cas d'un cluster, il faut garder à l'esprit qu'une sonde collecte les métriques seulement sur la machine sur laquelle elle s'exécute. Ainsi, nous devons avoir une instance de chacune de nos sondes sur chaque machine de notre cluster. Commençons par mettre en place cAdvisor, la sonde qui nous permet de collecter et exporter les métriques de nos conteneurs Docker.

L'installation se fait sous forme de conteneurs que nous allons créer à partir d'une image provenant du Docker Hub. Nous l'adaptions à nos besoins en lui spécifiant des paramètres spécifiques et nous l'instancions de manière à obtenir notre conteneur. Une fois mis en place, nous avons accès à une interface de visualisation à l'URL `http://<ip_de_la_machine>:8080`

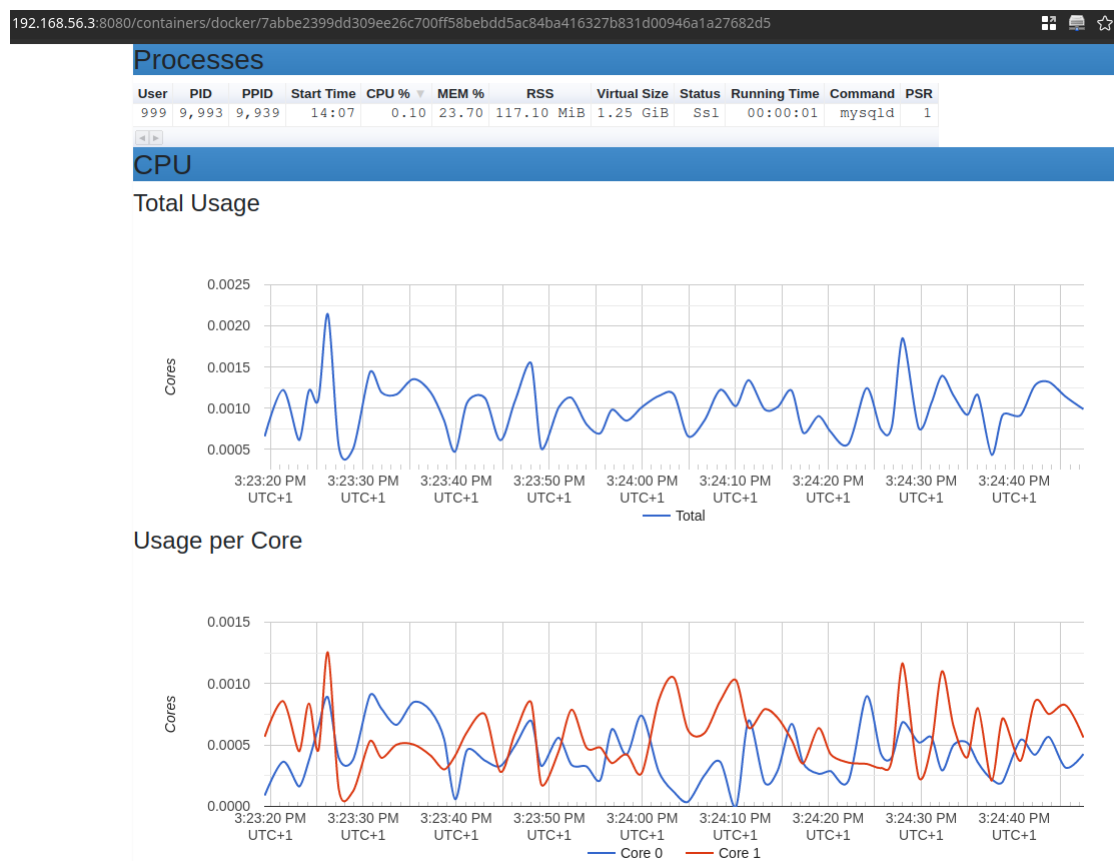


FIGURE 7.9 – Capture de l'interface cAdvisor

Mais cette interface est plus limitée que Grafana et ne nous intéresse donc pas. De plus, nous cherchons à centraliser nos métriques. C'est pourquoi cAdvisor expose un endpoint à l'adresse : `http://<ip_de_la_machine>:8080/metrics` dans un format compatible avec des logiciels de gestion de données que nous verrons dans la partie suivante.

Avant cela il nous reste à parler brièvement de Node Exporter. Pour rappel, cette sonde nous permet de collecter et exporter les métriques concernant les performances de nos machines. L'installation se fait de la même manière que pour cAdvisor et le mode de fonctionnement est similaire aussi, l'endpoint exposé est le suivant : `http://192.168.56.3:8080/metrics`. La différence est que Node exporter ne possède aucune interface graphique.

7.2.3 Prometheus

Pour gérer les données collectées nous utilisons Prometheus, c'est lui qui va rassembler les données de nos différentes sondes dans sa base de données de séries temporelles.

Pour le mettre en place nous lui définissons un service dans lequel nous spécifions notamment son fichier de configuration ainsi que des volumes qui permettent la persistance des données après un redémarrage de Prometheus :

```

prometheus:
  image: prom/prometheus:latest
  container_name: prometheus
  ports:
    - "9090:9090"
  command:
    - --config.file=/etc/prometheus/prometheus.yml
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheus-data:/prometheus

```

FIGURE 7.10 – Définition du service prometheus

Une fois notre service prometheus défini, nous aurons également besoin d'un fichier de configuration avant de pouvoir le lancer. C'est dans ce fichier de configuration que nous allons lui dire de récupérer les métriques auprès de "jobs". Nous leur donnons un nom, un intervalle à laquelle Prometheus va récupérer les données et l'endpoint exposé par la sonde auprès de laquelle nous voulons récupérer les métriques.

```

scrape_configs:
  - job_name: cadvisor_manager
    scrape_interval: 5s
    static_configs:
      - targets: ['192.168.56.2:8080']
  - job_name: node_exporter_manager
    scrape_interval: 5s
    static_configs:
      - targets: ['192.168.56.2:9100']
  - job_name: cadvisor_worker
    scrape_interval: 5s
    static_configs:
      - targets: ['192.168.56.3:8080']
  - job_name: node_exporter_worker
    scrape_interval: 5s
    static_configs:
      - targets: ['192.168.56.3:9100']

```

FIGURE 7.11 – Fichier de configuration de Prometheus (prometheus.yml)

7.2.4 InfluxDB

Dans notre cas d'expérimentation, l'utilisation que nous voulons faire d'InfluxDB c'est de remplacer la base de données de séries temporelles déjà présente sur Prometheus.

Premièrement, il faut configurer la base pour qu'elle soit opérationnelle. Il nous faut donc :

- Un utilisateur et son mot de passe
- Une première base de données
- Une API fonctionnelle pour faire des requêtes depuis Prometheus
- Avoir différents niveaux de rétention des données

Première instance

Pour installer InfluxDB, il suffit d'installer le paquet associé, ou, comme nous le faisons, de lancer la version conteneurisée de DockerHub. A savoir que l'image produite est bel et bien officielle.

Voici le paramétrage que nous avons introduit à notre cas d'expérimentation :

```
INFLUXDB_DB=prometheus
INFLUXDB_USER=influx
INFLUXDB_ADMIN_ENABLED=true
INFLUXDB_ADMIN_USER=influxadmin
INFLUXDB_ADMIN_PASSWORD=influxadmin
```

FIGURE 7.12 – Variables d'environnement concernant InfluxDB

Par ces variables là nous créons une première base nommée "prometheus" avec un compte "influxadmin" administrateur. Ces noms de variable peuvent changer d'une version à une autre, notamment pour la version 2 où les bases de données sont remplacées par la notion de bucket.

Prometheus

Ajoutons maintenant à la configuration de Prometheus les paramètres nécessaires pour qu'il puisse contacter l'API de InfluxDB.

```
remote_write:
- url: "http://192.168.56.4:8086/api/v1/prom/write?u=influxadmin&p=influxadmin&db=prometheus"

remote_read:
- url: "http://192.168.56.4:8086/api/v1/prom/read?u=influxadmin&p=influxadmin&db=prometheus"
```

FIGURE 7.13 – Configuration de l'API InfluxDB sur Prometheus

Il y a deux paramètres : le "remote_write", et le "remote_read". L'adresse "192.168.56.4" correspond à la machine de monitoring, c'est-à-dire celle où nous avons Prometheus, Grafana et Influx qui sont actifs. Nous sommes contraints de renseigner cette adresse plutôt que le nom "localhost" car nous avons des problèmes de résolution de nom de domaine à ce niveau-là dans notre environnement de test.

A cette adresse-là nous pointons le port "8086" qui correspond au port de InfluxDB, et les suites "/api/v1/prom/write" et "/api/v1/prom/read" sont les endpoints accessibles pour écrire les lire les données. Il n'y a pas besoin d'activer l'API car elle est mise en place par défaut lors de l'installation.

Le reste de l'url "?u=influxadmin&p=influxadmin&db=prometheus", c'est l'utilisateur, son mot de passe et la base de données que Prometheus va utiliser pour effectuer ses requêtes.

Nous on utilise l'utilisateur administrateur, la sécurité de l'infrastructure n'étant pas le but premier de notre projet.

Vérification de l'opération :

```
> select * from node_timex_status
name: node_timex_status
time          __name__          instance          job          value
-----
1646148276199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148277067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148281214000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148282071000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148286199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148287067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148291199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148292067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148296199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148301199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148302120000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148306199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148307067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148311202000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148312067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148316199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148317067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148321199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148322067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148326199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148327067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148331199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148332067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148336204000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148337067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148341199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148342067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148346199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148351199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148352070000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148356199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148357067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148361199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148362067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
1646148366199000000 node_timex_status 192.168.56.2:9100 node_exporter_manager 64
1646148367067000000 node_timex_status 192.168.56.3:9100 node_exporter_worker 64
```

FIGURE 7.14 – Capture d'écran des données d'InfluxDB

Les données issues des métriques se retrouvent bien dans la base "prometheus", dans l'exemple ci-dessus nous pouvons voir les données issues de la table "node_timex_status". De par son nom nous comprenons ici qu'il s'agit de métriques récoltées par les sondes Node Exporter. Nous avons précisément la source de chaque donnée dans la colonne "instance".

Politiques de rétention

Dans cette partie, nous faisons l'échantillonnage des données issues de Prometheus. Le niveau de précision de ces données dépend directement de l'intervalle de temps défini sur les sondes pour récolter les métriques.

Voici ce que nous mettons en place :

- Les données à courte rétention sont générées par Prometheus
- La rétention moyenne par échantillonnage des données à courte rétention
- La rétention longue par échantillonnage des données à moyenne rétention

Pour chacun de ces 3 points, il est nécessaire de mettre ce que l'on appelle une politique de rétention, c'est-à-dire la durée de temps pendant laquelle nous conservons les données.

Pour la rétention courte :

```
CREATE RETENTION POLICY "rp_court" ON "prometheus" DURATION 4w REPLICATION 1 DEFAULT
```

Pour la rétention moyenne :

```
CREATE RETENTION POLICY "rp_moy" ON "prometheus" DURATION 12w REPLICATION 1
```

Pour la rétention longue :

```
CREATE RETENTION POLICY "rp_long" ON "prometheus" DURATION 1y REPLICATION 1
```

En faisant comme cela, nous définissons la première politique de rétention comme par défaut sur les données reçues, grâce au mot-clé "DEFAULT". Les autres politiques sont utilisées dans ce qui suit.

Requêtes continues

Nous allons définir des requêtes continues. Comme leur nom l'indique, ce sont des requêtes InfluxQL qui sont exécutés, non pas vraiment en continu, mais à intervalles de temps régulières. Le but de ces requêtes, c'est de regrouper toutes les données de la base, et d'en effectuer l'échantillonnage selon une échelle de temps donnée.

Pour la rétention courte : Pas besoin de requêtes, nous avons ces données naturellement. Leur résolution est de 5 secondes sur les 4 dernières semaines.

Pour la rétention moyenne :

```
CREATE CONTINUOUS QUERY "cq_moy" ON "prometheus" BEGIN SELECT mean(value) AS value, min(value) AS min, max(value) AS max INTO "prometheus"."rp_moy". :MEASUREMENT FROM "prometheus"."autogen". :*/ GROUP BY time(60s),* END
```

Soit une résolution d'une minute sur les 12 dernières semaines, en se basant sur les données à courte rétention.

Pour la rétention longue :

```
CREATE CONTINUOUS QUERY "cq_long" ON "prometheus" BEGIN SELECT mean(value) AS value, mean(max) AS max, mean(min) AS min INTO "prometheus"."rp_long". :MEASUREMENT FROM "prometheus"."rp_moy". :*/ GROUP BY time(1800s),* END
```

Soit une résolution de 30 minutes sur la dernière année, en se basant sur les données à moyenne rétention.

Concernant tous ces niveaux, il est possible de voir les rendus graphiques sur Grafana, que nous allons maintenant ajouter à notre cas d'expérimentation.

7.2.5 Grafana

Grafana va nous permettre de visualiser les métriques centralisées par Prometheus et stockés par InfluxDB.

Comme pour les autres services conteneuriser sont installation se fait dans la machine monitoring seulement via le fichier docker-compose-monitoring.yml

```
grafana:
  image: grafana/grafana
  container_name: grafana
  ports:
    - "3000:3000"
  volumes:
    - grafana-data:/var/lib/grafana
```

FIGURE 7.15 – Définition du service Grafana dans le fichier docker-compose-monitoring.yml

On définit ce service d'abord par l'image qui va être prise sur le Docker Hub qui est l'image standard de Grafana, le nom du conteneur, les ports d'accès qui sont laissés par défauts et le volume qui est l'emplacement fichier ou Grafana va pouvoir stocker les informations dont il aura besoin.

Pour le reste de la mise en place tout se passe sur l'API web de Grafana, on s'y connecte à l'adresse : `http://<ip_de_la_machine>:3000`

On tombe sur la page de connexion où le login / mot de passe de base est admin / admin, on choisit un nouveau mot de passe et nous sommes connecter à l'interface Grafana.

La première chose à faire dans Grafana est d'implémenter notre Data Source, pour ce faire, on va dans le menu (à gauche), configuration (engrenages) et le menu Data Source, à partir de là on arrive dans le menu des data source où toutes nos data sources seront répertoriées. Il n'y en a de base aucune on peut donc rajouter la première en cliquant sur "Add data source".

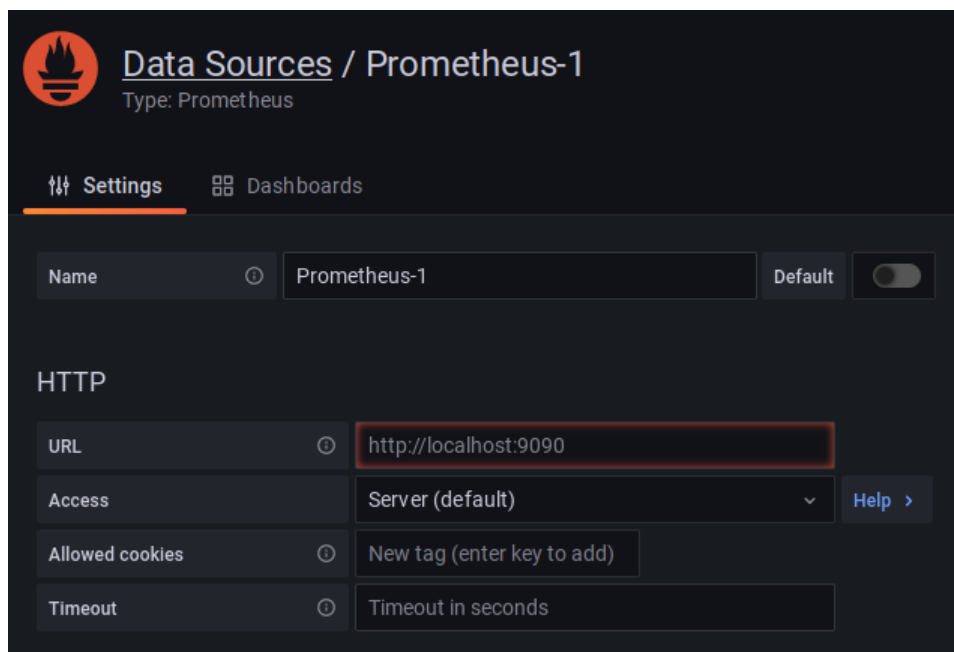


FIGURE 7.16 – Menu de configuration de la data source de Prometheus

Ici nous importons une Data Source Prometheus qui a donc ses propres paramètres, nous gardons toutefois une installation par défaut, nous allons simplement dans notre cas indiquer l'URL de l'endpoint de Grafana qui est le : `http :///<ip_de_la_machine> :9090` . Ensuite, on configure les sources de données avec l'URL de la source, ce qui permet d'avoir plusieurs sources du même type (plusieurs sources Prometheus par exemple) on peut également choisir la manière d'y accéder soit via le serveur qui l'héberge soit par l'API en ligne.

On peut également configurer l'authentification et les protocoles TLS pour renforcer la sécurité et la fiabilité des données transmises.

Avec Grafana on peut gérer l'accès de l'AlertManager de Prometheus, nous ne l'avons pas activé mais s'il est activé on peut simplement sélectionner la data source de l'AlertManager et activer l'alerting pour recevoir des mails d'alerte dès que la data source prometheus repère un problème.

Dans notre cas, nous devons également créer la Data Source d'InfluxDB, la manière de la créer est très similaire à celle de Prometheus puisque l'on garde une configuration de base on saisit seulement l'url : `http :///<ip_de_la_machine> :8086`

Pour finir la mise en place de Grafana, il faut mettre en place les dashboards pour chacune de nos solutions de monitoring. Pour ça on utilise la fonction qui permet d'importer des dashboards pré-faits avec les panels adéquats, il suffit de lier ces dashboards aux data sources adéquates et nous avons nos dashboards fonctionnels pour monitorer les résultats de Node Exporter, cAdvisor et influxDB.

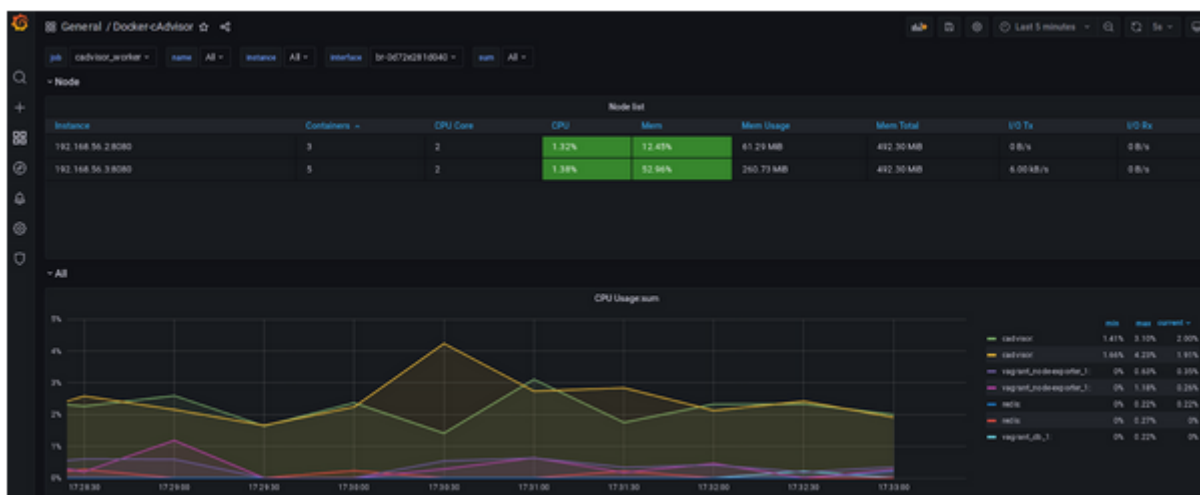


FIGURE 7.17 – Capture d'écran du dashboard des métriques de cAdvisor

7.2.6 WordPress

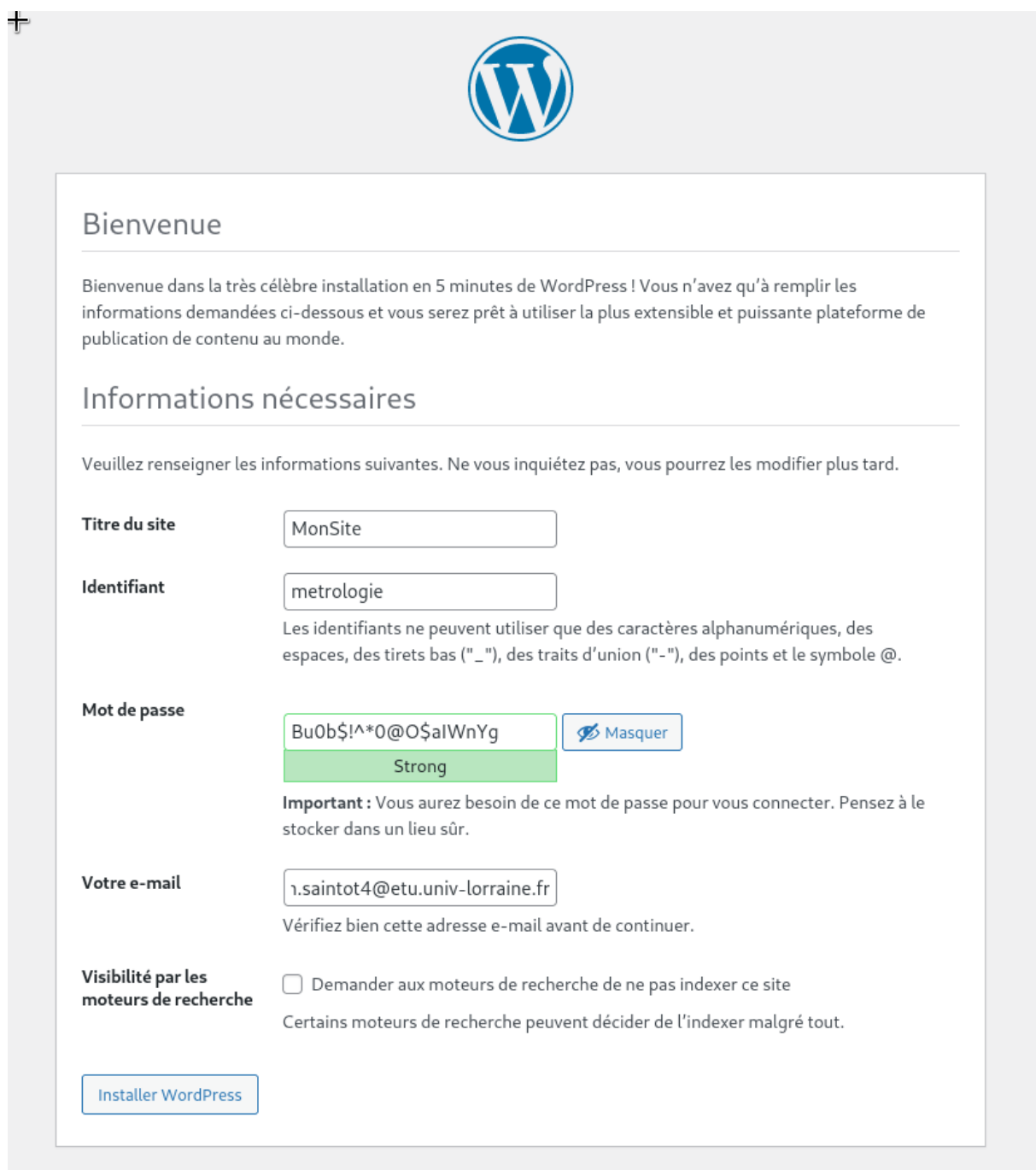
Nous avons mis en place un site Wordpress sous conteneur Docker. Nous utilisons l'image officielle mise en place sur le site Docker Hub.

Nous configurons les paramètres suivants :

- WORDPRESS_DB_HOST : db :3306
- WORDPRESS_DB_USER : wordpress
- WORDPRESS_DB_PASSWORD : wordpress

Pour la base utilisée par le WordPress.

Ensuite, nous en faisons la rapide mise en place graphique.

The image shows the WordPress installation welcome screen. At the top is the WordPress logo. Below it, a heading 'Bienvenue' is followed by a paragraph welcoming the user and stating that they will be able to use the platform after filling in the required information. A section titled 'Informations nécessaires' contains a prompt to provide the following information. The form fields include: 'Titre du site' with the value 'MonSite'; 'Identifiant' with the value 'metrologie', accompanied by a note about allowed characters; 'Mot de passe' with a strong password 'Bu0b\$!^*0@O\$aIWnYg' and a 'Masquer' button; 'Votre e-mail' with the value 'l.saintot4@etu.univ-lorraine.fr', accompanied by a verification prompt; and a checkbox for 'Visibilité par les moteurs de recherche' which is currently unchecked. A final 'Installer WordPress' button is at the bottom.

Bienvenue

Bienvenue dans la très célèbre installation en 5 minutes de WordPress ! Vous n'avez qu'à remplir les informations demandées ci-dessous et vous serez prêt à utiliser la plus extensible et puissante plateforme de publication de contenu au monde.

Informations nécessaires

Veuillez renseigner les informations suivantes. Ne vous inquiétez pas, vous pourrez les modifier plus tard.

Titre du site

Identifiant

Les identifiants ne peuvent utiliser que des caractères alphanumériques, des espaces, des tirets bas (" _ "), des traits d'union (" - "), des points et le symbole @.

Mot de passe [Masquer](#)

Strong

Important : Vous aurez besoin de ce mot de passe pour vous connecter. Pensez à le stocker dans un lieu sûr.

Votre e-mail

Vérifiez bien cette adresse e-mail avant de continuer.

Visibilité par les moteurs de recherche ☐ Demander aux moteurs de recherche de ne pas indexer ce site

Certains moteurs de recherche peuvent décider de l'indexer malgré tout.

[Installer WordPress](#)

FIGURE 7.18 – Mise en place du site WordPress

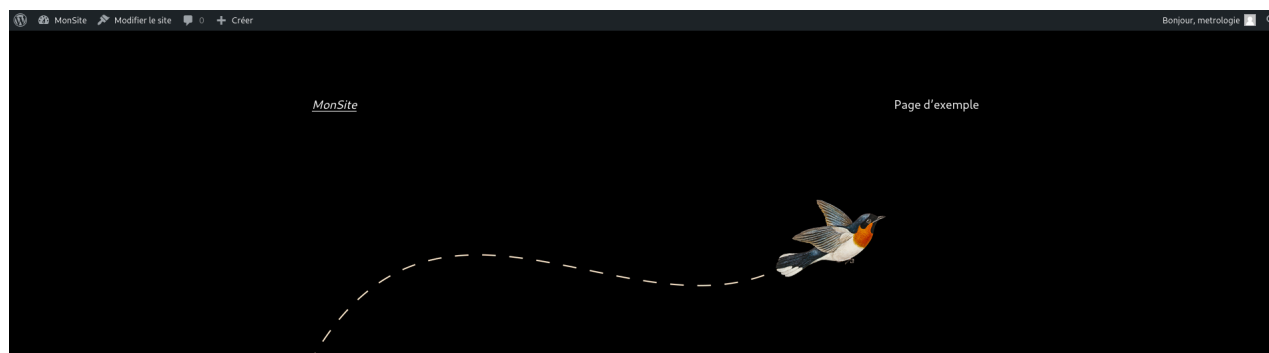


FIGURE 7.19 – Mise en place du site WordPress

7.2.7 MailCow

En soit MailCow est assez complexe car, comme nous l'avons dit lors de sa présentation, il est constitué de plusieurs conteneurs qui fonctionnent par réseau bridge. Heureusement, la procédure a été plutôt simplifiée pour nous, utilisateurs. Nous avons juste à récupérer le repository officiel de MailCow sur GitHub et dans celui-ci nous retrouverons un script en .sh qui va générer tout ce dont nous avons besoin dans un fichier YAML que nous aurons plus qu'à lancer avec Docker.

Parfois, des ports requis par MailCow sont déjà pris, ce qui entraîne un conflit. On peut vérifier que les ports nécessaires à sa mise en place ne sont pas déjà occupés avec la commande suivante : `ss -tlnp | grep -E -w '25|80|110|143|443|465|587|993|995|4190'`. Le plus souvent c'est le service `exim4` qu'il faudra arrêter et désactiver pour que postfix puisse utiliser le port 25.

```
Status: Downloaded newer image for robbertkl/ipv6nat:latest
Creating mailcowdockerized_watchdog-mailcow_1 ... done
Creating mailcowdockerized_memcached-mailcow_1 ... done
Creating mailcowdockerized_clamd-mailcow_1 ... done
Creating mailcowdockerized_olefy-mailcow_1 ... done
Creating mailcowdockerized_redis-mailcow_1 ... done
Creating mailcowdockerized_sogo-mailcow_1 ... done
Creating mailcowdockerized_dockerapi-mailcow_1 ... done
Creating mailcowdockerized_unbound-mailcow_1 ... done
Creating mailcowdockerized_solr-mailcow_1 ... done
Creating mailcowdockerized_mysql-mailcow_1 ... done
Creating mailcowdockerized_php-fpm-mailcow_1 ... done
Creating mailcowdockerized_postfix-mailcow_1 ... done
Creating mailcowdockerized_dovecot-mailcow_1 ... done
Creating mailcowdockerized_nginx-mailcow_1 ... done
Creating mailcowdockerized_ofelia-mailcow_1 ... done
Creating mailcowdockerized_rspamd-mailcow_1 ... done
Creating mailcowdockerized_acme-mailcow_1 ... done
Creating mailcowdockerized_netfilter-mailcow_1 ... done
Creating mailcowdockerized_ipv6nat-mailcow_1 ... done
vagrant@worker: /opt/mailcow-dockerized$
```

FIGURE 7.20 – Capture de la création des conteneurs MailCow sous Docker

Lorsque nous accédons à l'interface web, on se connecte avec les identifiants par défaut (ID : admin, MDP : moohoo). A partir de là, les fonctionnalités de MailCow s'offrent à nous, comme par exemple la création de domaines et de boîtes mail.

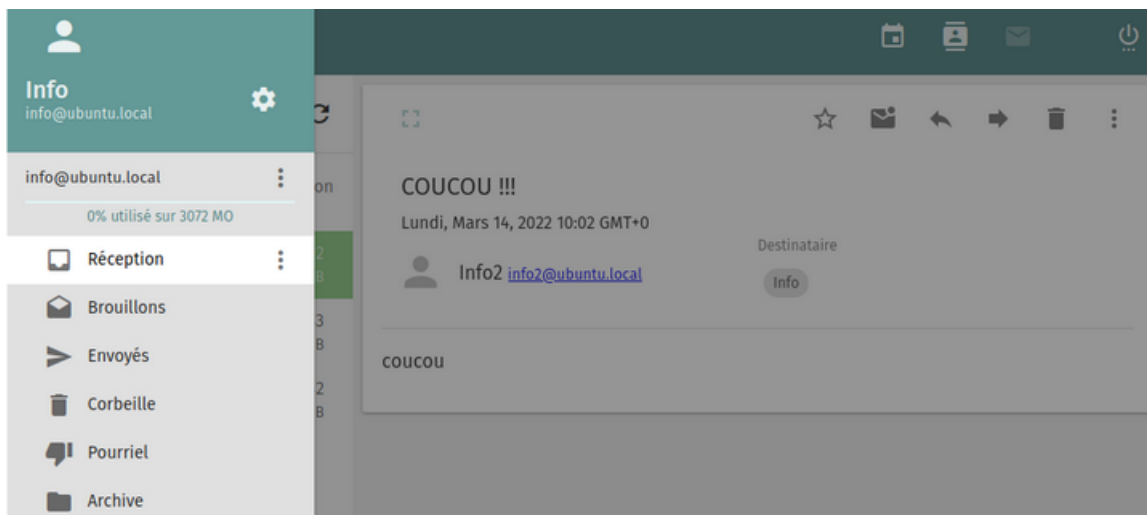


FIGURE 7.21 – Capture d'un test d'envoi de mail entre deux boîtes mail

7.2.8 Gatling

Maintenant que tout est en place, il est grand temps de tester l'infrastructure. Ceci, nous pouvons le faire grâce à Gatling. Dans notre cas, nous avons configuré préalablement Wordpress pour qu'il héberge un site fonctionnel. Sur ce site, nous pouvons y naviguer entre différents pages contenant des ressources média, et poster des commentaires.

Le but de ces tests de charge et de vérifier le bon fonctionnement de l'infrastructure, à la fois sur sa conception, mais aussi sur sa métrologie mise en place. Pendant ces tests, certaines données des métriques changent et sont visibles sur les dashboards de Grafana.

Dans la simulation (mise en annexe 11.7) nous testons d'insérer 3 utilisateurs par seconde au Wordpress en place. Ils vont premièrement récupérer la racine du site, ensuite naviguer vers un article précis, pour finalement ajouter un commentaire en envoyant une requête POST avec de bons paramètres.

Voici le résultat de cette simulation :



FIGURE 7.22 – Résultat de la simulation sous Gatling



FIGURE 7.23 – Résultat de la simulation sous Gatling



FIGURE 7.24 – Résultat de la simulation sous Grafana

Nous pouvons, sur cette troisième capture, observer l'activité qui change depuis le lancement du test, qui a duré une minute.

7.3 Problèmes rencontrés

7.3.1 MailU

Lors de la mise en place des applications web nous avons décidé de choisir une application web WordPress pour un site internet simple et MailU qui est un serveur de messagerie. Nous avons commencé à faire nos recherches sur MailU, car il propose plusieurs services tel qu'un antispam intégré et est facilement "conteneurisable" grâce à un outil en ligne qui permet de générer un fichier pour déployer simplement le service. Mais c'est ici que notre problème se complique lors du déploiement de MailU sur notre infrastructure nous rencontrons des erreurs liées au fichier que l'outil en ligne a généré.

```
ERROR : Invalid interpolation format for "image" option in service "admin" :
"$DOCKER_ORG :-mailu/$DOCKER_PREFIX :-admin :$MAILU_VERSION :-1.9"
```

Une fois ces erreurs résolues nous rencontrons des erreurs liées à notre réseau. Par ici nous parlons plus précisément du serveur DNS, nos différents conteneurs n'arrivent pas à contacter le serveur DNS malgré les modifications que nous avons pu apporter pour résoudre cela, nous avons donc décidé de mettre fin à cette solution au profit de MailCow.

7.3.2 Netdata

Au début du projet on nous a conseillé d'utiliser le duo Netdata et cAdvisor, si nous n'avons eu aucun problème avec cAdvisor nous avons eu en revanche quelques difficultés avec Netdata. En effet quand nous avons découvert la solution, nous avons essayé de l'installer dans sa version la plus récente car elle possédait des fonctionnalités très intéressantes comme le principe de "war room" qui permettait de centraliser les données et les partager au reste d'une équipe tout en y ajoutant un système d'alerting par mail. Mais malheureusement cette version n'était pas compatible avec notre manière d'installer l'application donc nous avions une application Netdata fonctionnelle mais dans une version qui ne nous intéressait pas vraiment, nous avons donc décidé de remplacer Netdata par Node Exporter qui nous permettait d'accomplir nos objectifs plus facilement tout en étant plus léger à l'utilisation que Netdata.

7.3.3 InfluxDB

Avant de vouloir configurer la version 1.8 d'InfluxDB pour notre infrastructure, nous nous sommes d'abord penchés sur sa version la plus récente, c'est-à-dire la 2.1 au moment de ce projet. Seulement, InfluxDB en version 2 n'est plus simplement que des bases de données de séries temporelles, elle est devenue capable de faire tout comme Prometheus. Ceci inclut la récolte des métriques avec les sondes Telegraf.

Nous n'utilisons évidemment pas les sondes Telegraf. Nous avons pu observer également que la documentation de InfluxDB changeait de structure entre chacune de ses versions, ce qui rendait son exploitation compliquée. Nous avons dû essentiellement nous référer à des tutoriels pour cette partie-là.

En raison des observations faites précédemment, nous avons décidé de continuer avec la version 1.8

7.3.4 Infrastructure

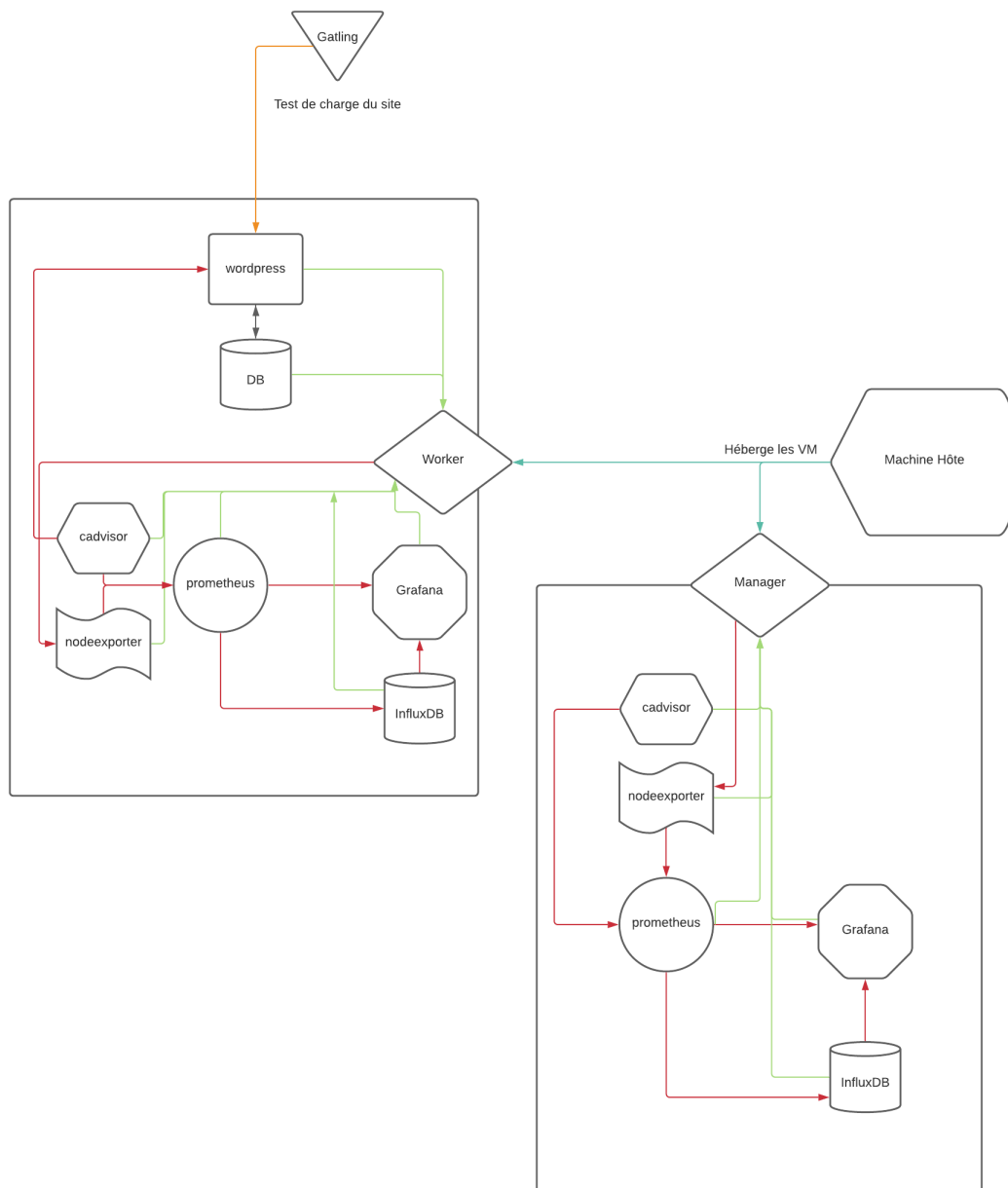


FIGURE 7.25 – Infrastructure au 25 janvier 2022

Voici l’une des toutes premières formes d’infrastructures que nous avons mise en place. Elle est globalement similaire en termes d’applications comparé à notre infrastructure actuelle. En revanche, on voit ici que chaque machine possède son propre système de monitoring ce qui n’est pas vraiment compatible avec ce qui est attendu comme le fait d’avoir toutes les métriques centralisées en un même point. Cette technique a également l’inconvénient de la taille que prenait un système de monitoring dédoublé et la difficulté d’aller de l’un à l’autre. Nous sommes donc partis ensuite sur l’infrastructure que nous vous avons présentée durant ce rapport, basée sur trois machines, une complètement dédiée au monitoring et les deux autres machines manager et worker, allégées des solutions de métrologie, n’ont que les sondes et les applications propres à leur fonctionnement.

Chapitre 8

Organisation du projet

8.1 Outils collaboratifs

La première chose que nous avons mis en place pour notre projet sont les outils pour communiquer et travailler au sein de l'équipe. En effet pour travailler en groupe de quatre, nous avons besoin de nous organiser de manière efficace. Surtout pendant la période de pandémie que nous avons connue. Notre objectif est donc que chaque membre de l'équipe ait accès à toutes les composantes du projet à n'importe quel moment et n'importe où.

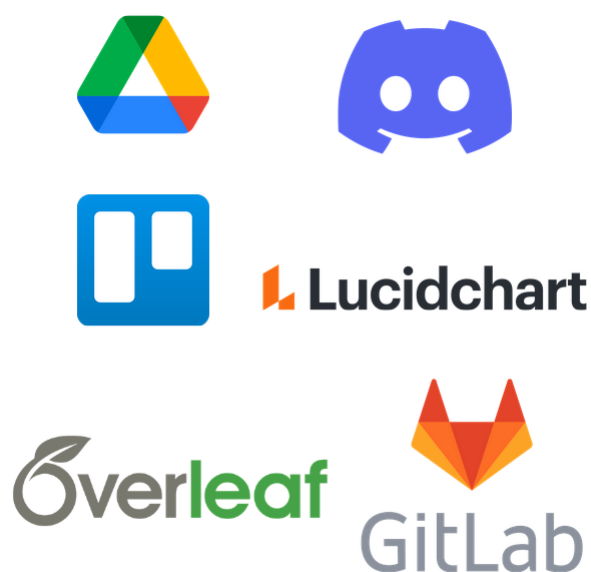


FIGURE 8.1 – Ensemble des logos d'outils collecticiels et organisationnels utilisés

- Google drive est la suite Google Documents qui nous a permis de mettre en commun toutes nos prises de notes ou ébauches de rapport. Nous utilisons Google Slides pour créer nos diapositives de présentation des soutenances.
- Discord est notre moyen de communication privilégié, toute l'équipe le possédait déjà avant le projet et mettre en place un serveur pour nous permettre d'échanger rapidement des messages ou faire des "réunions" à distance via le vocal ont été très pratiques.

- Trello a été un outil très important de notre organisation, il a permis en effet durant tous le projet de répartir efficacement les tâches et de voir l'avancement de chacune d'entre elles.
- Lucidchart nous a permis de travailler à plusieurs sur des schémas comme celui de notre infrastructure.
- Overleaf est l'outil en ligne qui nous a permis de réaliser notre rapport de soutenance, en effet c'est un outil de rédaction de documents en LaTeX qui est très complet et accessible partout et pour tous.
- GitLab nous a permis de partager et travailler en parallèle sur les fichiers de configurations des machines et des logiciels.

8.2 Répartition des tâches

Voici un tableau récapitulatif des différentes tâches menées au cours de notre expérimentation que chaque personne a réalisé ou aidé à réaliser :

Tâches	Maxence	Sébastien	Evan	Tom
Cluster Docker	X		X	
Sondes (cAdvisor, Node Exporter)				X
Prometheus	X			X
Grafana	X			X
InfluxDB	X	X		
WordPress	X		X	
MailCow	X			
Tests de charge Gatling		X		
Provisionnement de l'infrastructure	X		X	
Conception de l'architecture	X	X		X
Rédaction du rapport sous LaTeX	X			X
Rédaction du rapport	X	X	X	X
Mise en place des outils organisationnels	X	X	X	X
Tâche abandonnées ou pas terminées				
Netdata	X			X
MailU			X	
Déploiement des services avec Docker Swarm	X		X	

Chapitre 9

Conclusion

En conclusion, nous sommes plutôt satisfaits de notre expérience de projet. Celui-ci nous aura permis d'aborder à notre échelle une problématique clé du métier d'administrateur systèmes et réseaux, à savoir anticiper les temps d'interruption et évaluer la robustesse d'une infrastructure en suivant l'évolution d'indicateurs jugés représentatifs. Nous avons pu découvrir une large panoplie de solutions, de la conteneurisation aux logiciels de tests de charge en passant par les outils de métrologie et les applications web. Nous avons dû travailler également sur la composante organisationnelle du projet pour rendre l'avancement de ce-dernier plus efficace.

Nous avons plusieurs poursuites de projet possibles, la suite la plus évidente est l'agrandissement de l'infrastructure avec l'ajout de plus de machines worker, et dans la suite logique plus de services peuvent être mis à disposition, tel qu'AlertManager pour Prometheus qui aurait pu nous aider à gérer les alertes envoyées par les différentes applications afin d'être averti par mail notamment.

Nous aurions pu également affiner les rétentions variables sur InfluxDB, en ne gardant par exemple qu'une minute d'échantillonnage sur les trois premiers mois, puis deux minutes pour l'échantillonnage entre trois et six mois, et enfin ne garder que cinq minutes d'échantillonnage sur les six derniers mois.

Chapitre 10

Bibliographie

<https://gatling.io/>
<https://gatling.io/open-source/>
<https://www.influxdata.com/>
<https://fr.wikipedia.org/wiki/InfluxDB>
[https://fr.wikipedia.org/wiki/Gatling_\(logiciel\)](https://fr.wikipedia.org/wiki/Gatling_(logiciel))
<https://docs.influxdata.com/>
<https://gatling.io/docs/>
<https://devconnected.com/the-definitive-guide-to-influxdb-in-2019/>
https://prometheus.io/docs/concepts/metric_types/
[https://fr.wikipedia.org/wiki/M%C3%A9trique_\(logiciel\)](https://fr.wikipedia.org/wiki/M%C3%A9trique_(logiciel))
<https://www.mayasquad.com/glossaire/endpoint/>
<https://www.youtube.com/watch?v=zMC3SyeaDGU>
<https://github.com/dockersamples/docker-swarm-visualizer>
<https://docs.docker.com/compose/compose-file/#deploy>
<https://www.cerenit.fr/blog/influxdb-shard-duration-retention-policy/>
<https://mailcow.github.io/mailcow-dockerized-docs/#demo>
<https://cicd.life/intro-to-docker-swarm-pt2-config-options-requirements/>
<https://stefanjarina.gitbooks.io/docker/content/swarm-mode/swarm-configs.html>
<https://blog.ruanbekker.com/blog/2019/02/28/use-swarm-managed-configs-in-docker-swarm-to-store-your-application-configs/>
<https://gabrieltanner.org/blog/docker-swarm>
<https://stackoverflow.com/questions/48396459/docker-swarm-build-configuration-in-docker-compose-file-ignored-during-stack>
<https://community.icinga.com/t/retention-policies-and-continuous-queries-made-simple/117>
https://docs.influxdata.com/influxdb/v1.8/query_language/continuous_queries/
<https://youtu.be/6ADtXo7uHPg>
<https://docs.influxdata.com/influxdb/v1.8/administration/config/>
<https://www.infoq.com/fr/articles/prometheus-monitor-applications-at-scale/>
https://www.docker.com/increase-rate-limits?utm_source=docker&utm_medium=web%20referral&utm_campaign=increase%20rate%20limit&utm_budget=
https://mailcow.github.io/mailcow-dockerized-docs/post_installation/firststeps-rp/
<https://mailcow.github.io/mailcow-dockerized-docs/prerequisite/prerequisite-system/#usage-examples>

<https://pkgs.org/search/?q=docker-ce>
<https://www.youtube.com/watch?v=4rzc0hWRSPg>
<https://www.the-digital-life.com/mail-server-on-linux/>
https://docs.influxdata.com/influxdb/v1.8/query_language/manage-database/#retention-policy-management
<https://github.com/itzg/docker-minecraft-server#using-docker-compose>
<https://github.com/itzg/docker-minecraft-server/blob/master/examples/docker-compose-paper.yml>
<https://linuxtut.com/fr/c67f198718286b895308/>
<https://www.journaldunet.fr/web-tech/guide-de-l-entreprise-digitale/1443880-prometheus-le-monitoring-orienté-alerting-open-source-gratuit/>
<https://www.linuxtechi.com/install-configure-bind-9-dns-server-ubuntu-debian/>
<https://www.geco-it.fr/2021/12/24/prometheus/>
<https://sematext.com/blog/docker-container-monitoring/>
<https://blog.eleven-labs.com/fr/monitorer-ses-containers-docker/>
<https://www.youtube.com/watch?v=gb6AiqCJqP0>
https://chowdera.com/2021/12/202112192124161505.html#22prometheusinfluxdb_67
https://docs.influxdata.com/influxdb/v1.8/guides/downsample_and_retain/
https://runebook.dev/fr/docs/influxdata/influxdb/v1.3/guides/downsampling_and_retention/index
https://www.youtube.com/watch?v=Vq4cDIIdz_M8&list=PLY_rQVYyU1sB06FFBp18B2wEFsL0E_m8C
<https://leandeeep.com/installer-influxdb-1.8-via-docker/>
<https://www.influxdata.com/blog/prometheus-remote-write-support-with-influxdb-2-0/>
<https://github.com/prometheus/prometheus/issues/5657>
<https://twitter.com/influxdb/status/1416001421124702208>
<https://community.influxdata.com/t/influxdb2-prometheus-endpoint-problem/17380>
<https://community.influxdata.com/t/influxdb2-prometheus-endpoint-problem/17380>
https://github.com/prometheus/influxdb_exporter
<https://prometheus.io/docs/instrumenting/exporters/>
<https://grafana.com/grafana/dashboards/13946>
<https://grafana.com/grafana/dashboards/1860>
<https://www.aneu.eu/orchestration-docker-swarm/>
https://docs.docker.com/engine/reference/commandline/stack_deploy/
<https://theogindre.fr/2018/02/16/mise-en-place-dune-stack-de-monitoring-avec-influxdb-grafana-et-telegraf/>
<https://setup.mailu.io/1.9/>
<https://github.com/Mailu/Mailu/issues/853>
<https://l-informaticien-libre.com/installer-serveur-mailu/>
<http://logidee.com/asrall/postfix.pdf>
<http://logidee.com/asrall/postfix.pdf>
<http://x.guimard.free.fr/postfix/>
https://postfix.traduc.org/index.php/BASIC_CONFIGURATION_README.html
<https://postfix.traduc.org/index.php/INSTALL.html#install>

<https://hub.docker.com/r/mailu/postfix>
<https://mailu.io/master/compose/setup.html#bind-address>
<https://mailu.io/1.9/>
https://www.youtube.com/watch?v=o66UFsodUYo&ab_channel=TheDigitalLife
https://docs.influxdata.com/influxdb/v1.8/administration/authentication_and_authorization/
<https://grafana.com/grafana/dashboards/893>
<https://jeckel-lab.fr/2017/12/20/pre-configurer-grafana-avec-docker-compose/>
<https://www.syloe.com/glossaire/docker-swarm/>
<https://prometheus.io/docs/guides/cadvisor/>
<https://learn.netdata.cloud/docs/get-started#run-netdata-with-docker>
<https://wordpress.org/support/article/optimization/>
<https://fr.wordpress.org/support/article/optimization-caching/>
<https://fr.wordpress.org/support/article/how-to-install-wordpress/#instructions-detaillées>
<https://www.techrepublic.com/article/how-to-install-phpmyadmin-on-ubuntu-18-04/>
<https://fr.wordpress.org/support/article/how-to-install-wordpress/>
<https://docs.docker.com/engine/install/debian/>
<https://www.grottedubarbu.fr/introduction-docker-swarm/>
<https://prometheus.io/docs/guides/cadvisor/>
<https://geekflare.com/fr/docker-swarm/>
<https://www.learncloudnative.com/blog/2021-08-25-cadvisor>
<https://www.aukfood.fr/creation-dun-cluster-docker/>
<https://shahbhargav.medium.com/monitoring-docker-containers-using-cadvisor-and-prometheus-5350ae038f45>
<https://hub.docker.com/r/grafana/grafana>
<https://hub.docker.com/r/netdata/netdata>
https://hub.docker.com/_/wordpress
https://hub.docker.com/_/influxdb
<https://devopssec.fr/article/comprendre-gerer-manipuler-un-cluster-docker-swarm>
<https://www.editions-eni.fr/open/mediabook.aspx?idR=481f99f9659179b81324524f3f5e91b6>
<https://chambreuil.com/public/prof/csi/totalreport.pdf>
<http://www.novagen.tech/deployer-cluster-applicatif-10mn-docker-swarm-/>
<https://www.it-connect.fr/monitoring-supervision-et-metrologie/>
https://runebook.dev/fr/docs/influxdata/influxdb/v1.3/guides/downsampling_and_retention/index

[https://www.davidguida.net/
how-to-scale-your-services-with-docker-during-development/](https://www.davidguida.net/how-to-scale-your-services-with-docker-during-development/)
<https://forums.docker.com/t/autoscaling-in-docker-swarm/44353/10>
[https://www.ionos.fr/digitalguide/serveur/know-how/
docker-orchestration-avec-swarm-et-compose/](https://www.ionos.fr/digitalguide/serveur/know-how/docker-orchestration-avec-swarm-et-compose/)
<https://www.cloudbees.com/blog/running-services-within-docker-swarm>
<https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/>
<https://docs.docker.com/engine/swarm/>
<https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/>
<https://docs.docker.com/engine/swarm/swarm-tutorial/>
<https://www.lebigdata.fr/docker-definition>
<https://docs.docker.com/engine/swarm/stack-deploy/>
<https://docs.docker.com/get-started/swarm-deploy/>
<https://docs.docker.com/compose/compose-file/compose-file-v3/>
[https://docs.docker.com/engine/reference/commandline/service_create/
#specify-service-constraints---constraint](https://docs.docker.com/engine/reference/commandline/service_create/#specify-service-constraints---constraint)
[https://askubuntu.com/questions/1256246/
invalid-yaml-mapping-values-are-not-allowed-in-this-context-network](https://askubuntu.com/questions/1256246/invalid-yaml-mapping-values-are-not-allowed-in-this-context-network)
<https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/>
<https://prometheus.io/docs/alerting/latest/alertmanager/>
[https://www.journaldunet.fr/web-tech/guide-de-l-entreprise-digitale/
1146290-docker-definition-docker-compose-docker-hub-docker-swarm-160919/](https://www.journaldunet.fr/web-tech/guide-de-l-entreprise-digitale/1146290-docker-definition-docker-compose-docker-hub-docker-swarm-160919/)

Chapitre 11

Annexes

11.1 Vagrantfile

```
1 Vagrant.configure("2") do |config|
2
3   # Exécuté sur toutes les VMs
4   config.vm.provision "shell", inline: <<-SHELL
5     # Mise à jour
6     apt-get update
7     #apt-get upgrade
8     # Prérequis
9     apt-get install apt-transport-https
10    apt-get -y install ca-certificates curl gnupg lsb-release
11    # Téléchargement et installation de Docker
12    curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --
13    dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
14    echo \
15      "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
16      keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/
17      debian $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.
18      list > /dev/null
19    apt-get update
20    apt-get install docker-ce docker-ce-cli containerd.io -y
21    # Téléchargement et installation de docker-compose
22    curl -L "https://github.com/docker/compose/releases/download/1.29.2/
23    docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-
24    compose
25    chmod +x /usr/local/bin/docker-compose
26  SHELL
27  # Manager
28  config.vm.define "manager" do |b|
29    b.vm.box = "debian/contrib-stretch64"
30    b.vm.hostname = "manager"
31    b.vm.network "private_network", ip: "192.168.56.2"
32    b.vm.provision "shell", path: "provision.sh", args: "manager"
33  end
34  # Worker
35  config.vm.define "worker" do |b|
36    b.vm.box = "debian/contrib-stretch64"
37    b.vm.hostname = "worker"
38    b.vm.network "private_network", ip: "192.168.56.3"
39    b.vm.provision "shell", path: "provision.sh", args: "worker"
40  end
```



```

35 # Monitoring
36 config.vm.define "monitoring" do |b|
37   b.vm.box = "debian/contrib-stretch64"
38   b.vm.hostname = "monitoring"
39   b.vm.network "private_network", ip: "192.168.56.4"
40   b.vm.provision "shell", path: "provision.sh", args: "monitoring"
41 end
42 end

```

11.2 provision.sh

```

1 #!/bin/bash
2
3 VM=$1
4 if [ "$VM" = "manager" ];then
5   echo "-----"
6   echo "      Manager (cAdvisor + NodeExporter)      "
7   echo "-----"
8   #Récupération du token pour l'ajout du worker
9   docker swarm init --advertise-addr 192.168.56.2 > /vagrant/token.txt
10  #Mise en place des conteneurs
11  cp /vagrant/manager.local/docker-compose-manager.yml /home/vagrant/
12  cp /vagrant/manager.local/daemon.json /etc/docker/
13  #docker-compose -f docker-compose-manager.yml up -d
14  sudo service docker restart
15  sudo docker run -it -d -p 5000:8080 -v /var/run/docker.sock:/var/run/
docker.sock dockersamples/visualizer
16  sudo docker deploy --compose-file docker-compose-manager.yml stackdemo
17 fi
18
19 if [ "$VM" = "worker" ];then
20   echo "-----"
21   echo "      Worker (Wordpress + MAILU + cAdvisor + NodeExporter)      "
22   echo "-----"
23   #Utilisation du token récupéré
24   sed -n 5p /vagrant/token.txt | while read line ; do echo $line ; done
| sh
25   #Mise en place des conteneurs
26   cp /vagrant/worker.local/docker-compose-worker.yml /home/vagrant/
27   #docker-compose -f docker-compose-worker.yml up -d
28 fi
29
30 if [ "$VM" = "monitoring" ];then
31   echo "-----"
32   echo "      Monitoring (Prometheus + InfluxDB + Grafana)      "
33   echo "-----"
34   #Mise en place des conteneurs et conf Prometheus
35   cp /vagrant/monitoring.local/prometheus.yml /home/vagrant/
36   cp /vagrant/monitoring.local/docker-compose-monitoring.yml /home/
vagrant/
37   docker-compose -f docker-compose-monitoring.yml up -d
38 fi
39 # vim: et sw=4

```

11.3 docker-compose-manager.yml

```
1 version: '3.0'
2 services:
3   cadvisor:
4     image: gcr.io/cadvisor/cadvisor:latest
5     deploy:
6       mode: global
7     container_name: cadvisor
8     ports:
9       - 8080:8080
10    volumes:
11      - /:/rootfs:ro
12      - /var/run:/var/run:rw
13      - /sys:/sys:ro
14      - /var/lib/docker:/var/lib/docker:ro
15    depends_on:
16      - redis
17  redis:
18    image: redis:latest
19    deploy:
20      mode: global
21    container_name: redis
22    ports:
23      - 6379:6379
24  node-exporter:
25    image: prom/node-exporter:latest
26    deploy:
27      mode: global
28    ports:
29      - 9100:9100
30  db:
31    image: mysql:5.7
32    deploy:
33      placement:
34        constraints:
35          - "node.role==worker"
36    volumes:
37      - db_data:/var/lib/mysql
38    restart: always
39    environment:
40      MYSQL_ROOT_PASSWORD: somewordpress
41      MYSQL_DATABASE: wordpress
42      MYSQL_USER: wordpress
43      MYSQL_PASSWORD: wordpress
44  wordpress:
45    depends_on:
46      - db
47    image: wordpress:latest
48    deploy:
49      placement:
50        constraints:
51          - "node.role==worker"
52    ports:
53      - "8000:80"
54    restart: always
55    environment:
56      WORDPRESS_DB_HOST: db:3306
```

```
57     WORDPRESS_DB_USER: wordpress
58     WORDPRESS_DB_PASSWORD: wordpress
59
60 volumes:
61     db_data:
```

11.4 daemon.json

```
1 {
2   "experimental": true
3 }
```

11.5 docker-compose-monitoring.yml

```
1 version: '2.0'
2
3 volumes:
4     prometheus-data:
5         driver: local
6     grafana-data:
7         driver: local
8     # influxdb:
9     influxdb-volume:
10
11 services:
12     prometheus:
13         image: prom/prometheus:latest
14         container_name: prometheus
15         ports:
16             - "9090:9090"
17         command:
18             - --config.file=/etc/prometheus/prometheus.yml
19         volumes:
20             - ./prometheus.yml:/etc/prometheus/prometheus.yml
21             - prometheus-data:/prometheus
22     grafana:
23         image: grafana/grafana
24         container_name: grafana
25         ports:
26             - "3000:3000"
27         volumes:
28             - grafana-data:/var/lib/grafana
29     influxdb:
30         image: influxdb:1.8
31         container_name: influxdb
32         restart: always
33         ports:
34             - 8086:8086
35         volumes:
36             - influxdb-volume:/vol01/Docker/monitoring
37     environment:
38         - INFLUXDB_DB=prometheus
39         - INFLUXDB_USER=influx
40         - INFLUXDB_ADMIN_ENABLED=true
```

```
41 - INFLUXDB_ADMIN_USER=influxadmin
42 - INFLUXDB_ADMIN_PASSWORD=influxadmin
```

11.6 prometheus.yml

```
1 scrape_configs:
2 - job_name: cadvisor_manager
3   scrape_interval: 5s
4   static_configs:
5   - targets: ['192.168.56.2:8080']
6 - job_name: node_exporter_manager
7   scrape_interval: 5s
8   static_configs:
9   - targets: ['192.168.56.2:9100']
10 - job_name: cadvisor_worker
11   scrape_interval: 5s
12   static_configs:
13   - targets: ['192.168.56.3:8080']
14 - job_name: node_exporter_worker
15   scrape_interval: 5s
16   static_configs:
17   - targets: ['192.168.56.3:9100']
18
19 remote_write:
20 - url: "http://192.168.56.4:8086/api/v1/prom/write?u=influxadmin&p=
    influxadmin&db=prometheus"
21
22 remote_read:
23 - url: "http://192.168.56.4:8086/api/v1/prom/read?u=influxadmin&p=
    influxadmin&db=prometheus"
```

11.7 SimWordPress.java

```
1 // required for Gatling core structure DSL
2 import io.gatling.javaapi.core.*;
3 import static io.gatling.javaapi.core.CoreDsl.*;
4
5 // required for Gatling HTTP DSL
6 import io.gatling.javaapi.http.*;
7 import static io.gatling.javaapi.http.HttpDsl.*;
8
9 // can be omitted if you don't use jdbcFeeder
10 import io.gatling.javaapi.jdbc.*;
11 import static io.gatling.javaapi.jdbc.JdbcDsl.*;
12
13 // used for specifying durations with a unit, eg Duration.ofMinutes(5)
14 import java.time.Duration;
15
16
17 public class SimWordPress extends Simulation {
18
19   HttpProtocolBuilder httpProtocol = http
20     .baseUrl("http://192.168.56.3:8000")
```

```
21 .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,
    image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;
    v=b3;q=0.9")
22 .doNotTrackHeader("1")
23 .acceptLanguageHeader("fr-FR,fr;q=0.9")
24 .acceptEncodingHeader("gzip, deflate, br")
25 .userAgentHeader("Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
    KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36");
26
27
28 ScenarioBuilder scn3 = scenario("Stress nr3")
29     .exec(http("request_1")
30         .get("/"))
31     .pause(2)
32     .exec(http("request_2")
33         .get("/2022/03/18/bonjour-tout-le-monde/"))
34     .pause(2)
35     .exec(http("request_3")
36         .post("/wp-comments-post.php")
37         .formParam("comment","Ceci est un commentaire")
38         .formParam("author","BotTestGatling")
39         .formParam("email","jenaipasdemail@rien.com")
40         .formParam("url","google.com")
41         .formParam("comment_post_ID","1")
42         .formParam("comment_parent","0"));
43 {
44     setUp(
45         scn3.injectOpen(constantUsersPerSec(3).during(60))
46     )
47     .protocols(httpProtocol);
48 }
49 }
```