**MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE**

**UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM**

**Faculté des Sciences Exactes et d'Informatique**

**Département de Mathématiques et informatique**

**Filière : Informatique**

MEMOIRE DE FIN D'ETUDE

Pour l'Obtention du Diplôme de Master en Informatique

Option : **Réseaux et Systèmes**

Présenté par :

**ADOUL Sidali Abdelkader Benmokhtar**

**AIT ALI SAID Yacine**

THEME :

# Proposition d'une nouvelle stratégie d'ordonnancement pour Kubernetes

Soutenu le : 03/07/2022

Devant le jury composé de :

| | | |
|---|---|---|
| Mr. MECHAOUI Moulay Driss | Université de Mostaganem | Président |
| Mr.MIROUD Mohammed El Mustapha | Université de Mostaganem | Examinateur |
| Mme.FILALI Fatima Zohra | Université de Mostaganem | Encadrante |

Année Universitaire : 2021-2022

# Table des matières

**Page**

# Table des figures

# Liste des tableaux

# Liste des abréviations

**Iaas :** Infrastructure as a Service

**Paas :** Platform as a Service

**Saas :** Software as a Service

**AWS :** Amazon Web Service

**OS :** Operating System ( Système d'exploitation )

**VM :** Virtual Machine ( Machine Virtuelle )

**E/S :** Entré/Sorie

**API :** Application Programming Interface

**HTTP :** Hypertext Transfer Protocol

**Amazon ECR :** Amazon Elastic Container Registry

**K8s :** Kubernetes

**DNS :** Domain Name System

**IP :** Internet Protocol

**POD :** Point Of Delivery

**ILP :** Programmation Linéaire en nombre Entier

**CPU :** Central Processing Unit

**RAM :** Random Acces Memory

**Amazon EBS :** Amazon Elastic Block Store

**ITIL :** Information Technology Infrastructure Library

**ACO :** Ant Colony Optimization

**PSO :** Particle Swarm Optimization

**ProCon :** Progres-Based Container Scheduling

**CSML :** Container Scheduling based on Machine Learning

**PM :** Physical Machine ( Machine Physique )

**K8s-BWA :** Kubernetes BandWidth Aware Scheduler

**YAML :** YAML Ain't Markup Language

# Resumé

Kubernetes fait l'orchestration des conteneurs pour exploiter au mieux les ressources disponibles. Cette opération est réalisée à travers sa stratégie d'ordonnancement par défaut.
Cette stratégie étant statique, elle montre ses limites dans des conditions dynamiques et changeantes. Il est donc nécessaire d'étudier le problème d'ordonnancement des conteneurs de façon plus approfondie.
Le but de cette étude consiste à proposer une nouvelle stratégie d'ordonnancement pour kubernetes qui vise à réduire la consommation d'énergie et la bande passante.


**Mots clés :** Conteneur, Docker, Kubernetes, Ordonnancement.



# Abstract

Kubernetes orchestrates containers to make the most of the available resources. This operation is carried out through a default scheduling strategie.
Because of kubernetes default policy being static, it shows its limits under dynamic and changing conditions. It is therefore necessary to study the problem of container scheduling more in depth.
The purpose of our study is to come out with a new scheduling strategy that aims to reduce the energy consumption and bandwidth.


**Key words :** Container, Docker, Kubernetes, Scheduling.



# ملخص

kubernetes تنسق الحاويات لتحقيق الاستغلال الامثل للموارد المتاحة.يتم تنفيذ هذه العملية عن طريق استراتيجيته الخاصة.
الا انها لم تثبت جدارتها في الظروف الديناميكية و المتغيرة، لذلك اصبح من الظروري دراسة مشكلة جدولة الحاويات بتعمق.
الهدف من دراستنا هو اقتراع استراتيجية جديدة ل kubernetes تهدف الى تخفيض استهلاك الطاقة و passante bande


**كلمات مفتاحية:** حاوية،Kubernetes،Docker،جدولة

# Dédicace

*Je dédie ce travail :*

*À mes chers parents et à mes frères, qui ont été mon plus grand Soutien.*

*À tous ceux qui me tiennent à cœur, en particulier ma mère, pour son aide, son temps et ses encouragements tout au long de mes études.*

*Sidali Abdelkader Benmokhtar*

# Remerciments

Avant toute personne, je tiens à remercier ALLAH le tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce modeste travail.

Je tiens à exprimer ma sincère gratitude à notre encadrante Mr.FILALI Fatima Zohra pour ses conseils et ses encouragements tout au long de ce projet.

Je tiens à exprimer toute ma gratitude à tous les membres de jury, pour avoir bien voulu juger notre travail.

Je veux aussi adresser mes sincères remerciements à tous les enseignants du département de l'informatique de l'université de Mostaganem qui ont contribué à notre formation.

Aussi à mes parents, ma mère et mon père Abdelkader, qui m'ont toujours soutenue et encouragé tout au long de mon cursus.

Par crainte d'avoir oublié quelqu'un, que tous ceux et toutes celles à qui je suis redevables se voient ici vivement remerciés.

***ADOUL Sidali Abdelkader Benmokhtar***

# Remerciments

La réalisation de ce mémoire a été possible grâce au concours de plusieurs personnes à qui je voudrais témoigner toute ma gratitude.

Je voudrais tout d'abord adresser toute ma reconnaissance à la directrice de ce mémoire, Madame FILALI Fatima Zohra, pour sa patience, sa disponibilité et surtout ses judicieux conseils, pour le temps qu'elle a consacré à nous apporter les outils méthodologiques indispensables à la réalisation de ce travail.

Je désire aussi remercier les professeurs de l'université de Mostaganem pour l'enseignement dispensé, qui m'ont fourni les outils nécessaires à la réussite de mes études universitaires.

Je voudrais exprimer ma reconnaissance envers les amis et collègues qui m'ont apporté leur soutien moral et intellectuel tout au long de ma démarche, et en particulier mon binôme pour tous les moments partagés et les efforts fournis.

Je remercie en dernier et surtout mon très cher père, Youssef, qui a toujours été là pour moi et qui n'a ménagé aucun effort pour mon bien être et mon épanouissement tout au long de mon cursus.

Enfin, je tiens à dédier ce modeste travail à ma chère maman qui m'a quitté bien trop tôt en espérant que je pourrais un jour la rendre fière. Allah yerahmek.

***AIT ALI SAID Yacine***

# Introduction Générale

Le cloud computing est un sujet qui a reçu beaucoup d'attention de la part d'individus et d'organisations de différentes disciplines au cours de la dernière décennie. Ce nouvel environnement implique une grande flexibilité et une disponibilité des ressources de calcul à différents niveaux d'abstractions et à moindre coût. Les fournisseurs de services cloud (par exemple, Google, Microsoft, Amazon) louent à leurs clients des ressources et des services de cloud computing qui sont utilisés de manière dynamique en fonction de la demande du client selon un certain modèle commercial. Des services généraux dans différents domaines d'application tels que les affaires, l'éducation et la gouvernance sont fournis aux clients en ligne et sont accessibles via un navigateur Web, tandis que les données et les logiciels sont stockés sur les serveurs cloud situés dans les centres de données.

La virtualisation a permis la banalisation du cloud computing, car les ressources matérielles sont devenues disponibles pour exécuter différents environnements, et partager des ressources informatiques entre différentes entreprises. Comme l'adoption du cloud computing est devenue une norme au niveau des entreprises à grande échelle, l'amélioration du partage des ressources est une prochaine étape logique. La technologie des conteneurs améliore le partage des ressources matérielles en éliminant une couche de configuration et d'installation de l'infrastructure.

L'exécution d'une instance virtuelle d'un système informatique dans une couche abstraite du matériel physique est le concept de base de la virtualisation. Les applications qui s'exécutent sur la machine virtualisée fonctionnent sans connaissance de couches intermédiaires.

Les technologies de conteneurs offrent une agilité sans précédent dans le développement et l'exécution d'applications dans un environnement cloud, en particulier lorsqu'elles sont combinées à une architecture de type microservice.

Kubernetes est un exemple de ces technologies de conteneurs, Kubernetes est un orchestrateur open source pour le déploiement d'applications conteneurisées. Il a été développé à l'origine par Google, inspiré par une décennie d'expérience dans le déploiement de systèmes évolutifs et fiables dans des conteneurs via des API orientées applications.

Kubernetes fait l'ordonnancement des conteneurs avec son ordonnanceur en utilisant des stratégies d'ordonnancement, ces stratégies permettent de répondre à des besoins standards et font le job la plupart du temps, mais elles sont statiques et ne peuvent pas répondre aux besoins dynamiques en ressources des applications car elles ne tiennent pas compte des caractéristiques des environnements et des applications microservices.

Récemment, l'ordonnancement des conteneurs dans les environnements cloud est devenu un sujet de recherche très important, les études à ce sujet se sont multipliées afin de proposer de nouvelles stratégies plus pertinentes, plus adaptables aux différents environnements. Car un bon ordonnancement permet l'utilisation efficace de tous les dispositifs informatiques, réduit le temps de traitement total et assure la satisfaction des clients.
Au vu de l'importance de l'ordonnancement, nous avons fait ce travail afin de proposer une stratégie d'ordonnancement pour Kubernetes qui corrige les limites de cette dernière en prenant en compte la consommation d'énergie et de la bande passante.
Ce mémoire est composé de quatres chapitres divisés comme suit :

- **Chapitre 1:** Nous évoquons dans ce chapitre les concepts du cloud et de la virtualisation, ainsi que la conteneurisation et l'architecture microservice.
- **Chapitre 2:** Nous parlons dans ce chapitre de Docker et son architecture, et de l'ordonnanceur de conteneurs et son fonctionnement, et nous présentons en détails Kubernetes ( ses composants et son architecture et la stratégie d'ordonnancement utilisée par défaut), les différentes stratégies d'ordonnancement existantes, ainsi que les métriques de performances prises en compte pour évaluer ces stratégies, et nous faisons aussi l'étude et l'analyse de 3 travaux proposant chacun une stratégie d'ordonnancement.
- **Chapitre 3:** Dans ce chapitre, nous présentons la description de notre solution et ses phases ainsi que les travaux dont nous nous sommes inspirés pour sa réalisation.
- **Chapitre 4:** Dans ce chapitre nous parlons sur les détails de l'implémentation, l'environnement matériel et logiciels utilisés ainsi que les résultats des tests.

# Chapitre 1

# Concepts du cloud et de la virtualisation

## Sommaire

# I. Introduction

Le cloud computing est devenu un concept majeur faisant référence à l'utilisation de la mémoire et des capacités de calcul des ordinateurs et des serveurs répartis dans le monde et liés par un réseau. Cette technologie a permis d'offrir aux utilisateurs des ressources quasi infinies. Dans ce chapitre, nous allons parler en détails de cette technologie ainsi que d'autre notions qui lui sont reliées tels que la virtualisation et la conteneurisation.

# II. Informatique en nuage (Cloud Computing)

Le cloud computing ou informatique en nuage est une infrastructure dans laquelle la puissance de calcul et le stockage sont gérés par des serveurs distants auxquels les usagers se connectent via une liaison Internet sécurisée. L'ordinateur de bureau ou portable, le téléphone mobile, la tablette tactile et autres objets connectés deviennent des points d'accès pour exécuter des applications ou consulter des données qui sont hébergées sur les serveurs. Le cloud se caractérise également par sa souplesse qui permet aux fournisseurs d'adapter automatiquement la capacité de stockage et la puissance de calcul aux besoins des utilisateurs. Cette technologie consiste à externaliser les données informatiques vers des serveurs distants. Ces services s'adressent surtout aux entreprises et aux organismes officiels. Les données du client sont envoyées via Internet vers des serveurs distants situés dans des centres de stockage sécurisés et vidéosurveillés avec accès limité. Le fournisseur se charge de conserver les données de ses clients en lieu sûr. Le cloud computing se décline en plusieurs niveaux de services : IAAS (Infrastructure As A Service) : Seule l'infrastructure matérielle est externalisée. PAAS (Platform As A Service) : L'externalisation concerne l'infrastructure matérielle, les données et les applications. SAAS (Software As A Service) : Le tout-compris qui inclut l'externalisation complète, la mise en fonctionnement et la maintenance. Il s'agit de la formule la plus courante.

## 1. Les services de l'informatique en nuage

### 1.1. Infrastructure As A Service (IAAS)

L'infrastructure en tant que service (IaaS) fournit le matériel informatique (serveurs, technologie réseau, stockage et espace de centre de données) en tant que service. Cela peut également inclure la mise à disposition de systèmes d'exploitation et de technologies de virtualisation pour

gérer les ressources. Le client IaaS loue des ressources informatiques au lieu de les acheter et de les installer dans son propre centre de données. Le service est généralement payé sur la base de l'utilisation.[1]

### 1.2. Platform As A Service (PAAS)

Platform as a Service (PaaS) : C'est là que les applications sont développées à l'aide d'un ensemble de langages de programmation et outils pris en charge par le fournisseur PaaS. Le PaaS offre aux utilisateurs un niveau élevé d'abstraction qui leur permet de se concentrer sur le développement de leurs applications et de ne pas se soucier de l'infrastructure sous-jacente. [2]

### 1.3. Software As A Service (SAAS)

Software as a Service (SaaS) : C'est là que les utilisateurs utilisent simplement un navigateur Web pour accéder à des logiciels que d'autres ont développé et offrent en tant que service sur le Web. Au niveau SaaS, les utilisateurs n'ont pas le contrôle ou l'accès à l'infrastructure sous-jacente utilisée pour héberger le logiciel. Salesforce, Google Docs sont des exemples populaires qui utilisent le modèle SaaS du cloud computing.. [2]

## III. La virtualisation

### 1. Définition :

La virtualisation est une technique qui permet de partionner, étendre ou remplacer une interface existante en plusieurs interfaces virtuelles complètement séparées pour imiter le comportement de l'interface/système réel. La virtualisation peut également être définie comme la technique qui encapsule l'interface virtuelle (ressource ou requête ou application) à partir de la couche matérielle sous-jacente de cette interface.[3]
Voici la figure 1.1 qui montre la différence entre un environnement avec et sans virtualisation.

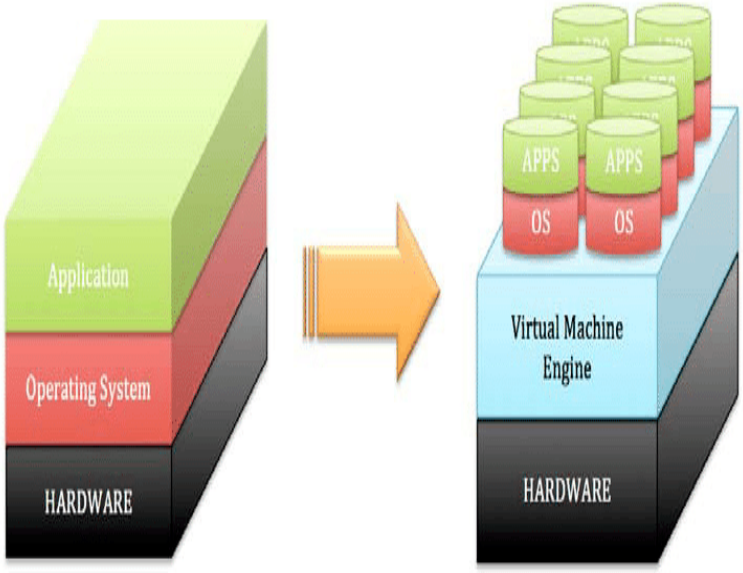


FIG. 1.1 : Comparaison entre une architécture sans (gauche) et avec (droite) virtualisation [13]

## 2. Hyperviseur

Le logiciel qui permet la virtualisation est appellé hyperviseur. Il s'agit d'une couche logicielle légère qui se situe entre le matériel physique et les environnements virtualisés et permet à plusieurs systèmes d'exploitation (OS) de fonctionner en tandem sur le même matériel. L'hyperviseur est l'intermédiaire qui extrait les ressources des matières premières de l'infrastructure et les oriente vers les différentes instances de calcul.[37]

## 3. Fonctionnement

La virtualisation consiste à créer plusieurs machines virtuelles (aussi appelées ordinateurs virtuels, instances virtuelles, versions virtuelles, VM ou virtual machine) à partir d'une machine physique, à l'aide d'un logiciel appelé hyperviseur. Parce que ces machines virtuelles fonctionnent de la même manière que des machines physiques, mais ne s'appuient sur les ressources que d'un seul système informatique, la virtualisation permet aux directions informatiques d'exécuter plusieurs systèmes d'exploitation sur un seul serveur (connu également sous le nom d'hôte). Pendant ce temps, l'hyperviseur attribue des ressources informatiques à chaque ordinateur virtuel, en fonction des besoins. Les opérations informatiques deviennent ainsi beaucoup plus efficaces et rentables.

# 4. Les types de virtualisation

## 4.1. Virtualisation de serveurs

Elle consiste à diviser les serveurs physiques en plusieurs serveurs virtuels, chacun exécutant son propre système d'exploitation. Ce qui permet ainsi d'exploiter toute la puissance des serveurs physiques afin de réduire considérablement les coûts matériels et d'exploitation.

## 4.2. Virtualisation de réseau

Ce type de virtualisation consiste à diviser la bande passante disponible en canaux indépendants, chacun étant affecté à un serveur ou un appareil, en fonction des besoins. La virtualisation de réseau facilite les tâches de programmation et de provisioning du réseau, telles que l'équilibrage des charges et la protection par pare-feu, sans avoir à toucher à l'infrastructure sous-jacente.

## 4.3. Virtualisation de stockage

La virtualisation de stockage a lieu lorsque l'espace de stockage physique de plusieurs appareils d'un réseau est unifié au sein d'un appareil de stockage virtuel, géré depuis une console centrale. Elle se fait à travers un logiciel de virtualisation qui doit identifier les capacités disponibles sur les appareils physiques, et d'agréger ces capacités au sein d'un environnement virtuel. Aux yeux des utilisateurs finaux, le stockage virtuel ressemble à un disque dur physique standard.

## 4.4. Virtualisation de données

Permet à une application d'accéder aux données et de les exploiter sans avoir besoin des détails sur l'emplacement physique ou le format de ces données. Les utilisateurs peuvent ainsi créer une représentation de données à partir de plusieurs sources, sans déplacer ni copier ces données. Cette agrégation de données se fait avec un logiciel de virtualisation de données, qui intègre et visualise virtuellement ces données à travers un tableau de bord, permettant aux utilisateurs d'accéder à de grands ensembles de données depuis un point unique, où qu'elles soient stockées. [4]

### 4.5. Virtualisation des applications

La virtualisation d'applications est un moyen d'exécuter une application isolée des autres applications. L'application s'exécute dans une bulle plutôt que d'avoir à être physiquement installée sur un PC. Les résultats finaux sont que les paramètres du système de fichiers et du registre sous-jacents ne sont jamais modifiés. [5]

### 4.6. Virtualisation de poste de travail

Dans la virtualisation de poste de travail, l'application est hébergée dans une machine virtuelle (qui inclut également le système d'exploitation), donc plutôt que de donner aux employés un PC physique, l'entreprise peut leur donner une machine virtuelle personnelle s'exécutant sur un serveur central, ce qui permet également d'économiser des dépenses et de l'espace. [3]

## IV. La Conteneurisation

Il s'agit d'une forme de virtualisation du système d'exploitation dans laquelle les applications peuvent s'exécuter dans des espaces utilisateurs isolés appelés conteneurs qui utilisent le même système d'exploitation partagé.
La conteneurisation d'une application permet d'isoler le conteneur du système d'exploitation hôte, avec un accès limité aux ressources sous-jacentes, à l'instar d'une machine virtuelle légère. L'application conteneurisée peut être exécutée sur différents types d'infrastructures, tels qu'un serveur bare metal, cloud ou sur des VMs, sans la remanier pour chaque environnement.
La conteneurisation permet de réduire les charges au démarrage et de supprimer la nécessité de configurer des systèmes d'exploitation invités distincts pour chaque application, car ils partagent tous un seul noyau de système d'exploitation. En raison de cette efficacité élevée, les développeurs de logiciels utilisent couramment la conteneurisation des applications pour regrouper plusieurs microservices individuels constituant les applications modernes. [6]

### 1. Les Conteneurs

Les conteneurs sont une abstraction au niveau de la couche application qui empaquette le code et les dépendances. Ils fournissent une virtualisation au niveau du système d'exploitation en tirant parti des fonctionnalités du noyau pour isoler les processus et définir les limites d'utilisation du système pour les ressources telles que le processeur, la mémoire, les E/S disque

et le réseau. Plusieurs conteneurs peuvent s'exécuter sur la même machine et partager le noyau du système d'exploitation avec d'autres conteneurs, chacun s'exécutant en tant que processus isolé dans l'espace utilisateur. [7]

## 2. Intérêts de la conteneurisation

- **L'accélération des développements :** Le développeur travaille dans un cadre restreint à son strict nécessaire, ce qui lui épargne le codage de tests d'interactions notamment. Cela favorise aussi la création de bacs à sable et donc une montée en compétence et une capacité d'innovation plus rapide.
- **La portabilité :** Permet l'accélération des déploiements, le conteneur créé est cohérent et ne souffrira pas d'être exécuté sur un autre environnement, tout en étant moins gourmand qu'une VM et peut donc être plus facilement déplacé, copié, relancé.
- **L'impact moindre sur les performances du serveur :** Un conteneur pouvant libérer rapidement les ressources (mémoire, stockage) inutilisées. [8]

## 3. Les Microservices

Les microservices ont gagné en popularité dans l'industrie au cours des dernières années. L'idée clé est que plutôt que d'architecturer des applications monolithiques, on peut obtenir une pléthore d'avantages en créant de nombreux services indépendants qui fonctionnent ensemble de concert. Les avantages obtenus incluent des bases de code plus simples pour les services individuels, la possibilité de mettre à jour et de mettre à l'échelle les services de manière isolée, permettant aux services d'être écrits dans différentes langues si besoin et d'utiliser différentes piles de middleware et même des niveaux de données pour différents services. [9]

### 3.1. Caractéristiques des microservices

- Composants multiples : Les logiciels construits sous forme de microservices peuvent, par définition, être décomposés en plusieurs services à composants. Ainsi, chacun de ces services peut être déployé, modifié, puis redéployé indépendamment sans compromettre l'intégrité d'une application.
- Conçu pour les entreprises : Le style des microservices est généralement organisé autour des capacités et des priorités de l'entreprise.

- Routage simple : Les microservices fonctionnent un peu comme le système UNIX classique : ils reçoivent les requêtes, les traitent et génèrent une réponse en conséquence.
- Décentralisation.
- Résistance aux pannes.
- Évolutionniste : L'architécture des microservices est une conception évolutive, et encore une fois, est idéale pour les systèmes évolutifs où il n'est pas possible d'anticiper les types d'appareils qui pourraient un jour accéder à l'application. [10]

Voici le tableau 1.1 qui montre la comparaison selon certains critères entre les conteneurs et les VMs

| Caractéristique | Conteneur | Machine virtuelle |
|---|---|---|
| Installation | Logiciel qui permet d'installer différentes fonctionnalités d'une application de manière indépendante. | Permet l'installation d'un autre logiciel à l'intérieur afin qu'il puisse être contrôlé virtuellement au lieu d'installer le logiciel directement sur l'ordinateur. |
| OS | Les conteneurs partagent un seul système d'exploitation | Chaque VM s'exécute dans son propre système d'exlpoitation |
| Virtualisation | Virtualise uniquement le système d'exploitation. | Virtualisation au niveau matériel. |
| Taille | La taille du conteneur est très petite : c'est-à-dire quelques mégaoctets. | La taille est très grande. |
| Temps d'execution | S'exécute en quelques secondes. | Prend quelques minutes pour s'exécuter, en raison de sa grande taille. |
| Mémoire | Nécessite peu de mémoire système. | Utilise beaucoup de mémoire système. |
| Isolation | Isolation au niveau des processus, peut-être moins sécurisé. | Entièrement isolé et donc plus sécurisé. |
| Utilité | Utile lorsqu'il est nécessaire de maximiser les applications en cours d'exécution en utilisant un minimum de serveurs. | Utile lorsque toutes les ressources du système d'exploitation sont nécessaires pour exécuter diverses applications. |
| Fonctionnement | Fournit des services de système d'exploitation à partir de l'hôte sous-jacent et isole les applications en utilisant de la mémoire virtuelle | Géré par un hyperviseur et utilise le matériel de la VM. |

TAB. 1.1 : Tableau comparatif entre les conteneurs et les machines virtuelles [11]

Et voici la figure 1.2 qui montre la différence de fonctionnement entre un conteneur et une machine virtuelle.

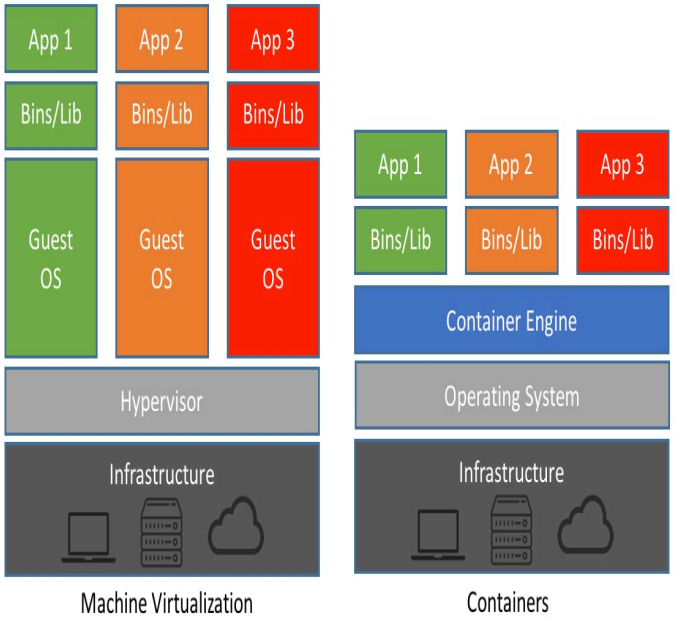


FIG. 1.2 : Différence de fonctionnement entre un conteneur et une machine virtuelle [12]

# V. Conclusion

Le cloud computing est une amélioration technologique récente qui a le potentiel d'avoir un grand impact sur le monde. Il présente de nombreux avantages qu'il offre aux utilisateurs et aux entreprises.

Au cours de ce chapitre, nous avons discuté les principes du cloud computing et les notions gravitant autour de ce paradigme.

Dans le prochain chapitre, nous allons présenter en détails le gestionnaire des conteneurs le plus utilisé **Docker** et le gestionnaire de groupes de conteneurs le plus célèbre **Kubernetes**, les différentes stratégies d'ordonnancements ainsi que les métriques de performanes.

# Chapitre 2

# Docker, kubernetes et les stratégies d'ordonnancements

## Sommaire

# Partie 1: Docker et kubernetes

## I. Introduction

Dans le développement traditionnel d'un logiciel, les ingénieurs travaillent sur leur base de code dans un environnement informatique spécifique. Bien qu'une application puisse fonctionner sur la machine d'un développeur, cela ne garantit pas qu'elle fonctionnera sur une machine différente.
L'utilisation des conteneurs vient remédier à ce problème grâce à leur haut degré de portabilité, mais pour créer et déployer des conteneurs, les développeurs ont besoin d'un environnement d'exécution de conteneur. C'est là qu'entre en jeu Docker en raison de sa capacité à gérer les cycles de vie des conteneurs et à définir des limites de ressources sur les conteneurs.
Une fois qu'une application est déployée dans un conteneur, elle peut théoriquement être utilisée par n'importe qui, n'importe où. Mais comment gérer un grand nombre de conteneur à la fois ? C'est là qu'intervient Kubernetes. Il automatise le processus de déploiement, de mise à l'échelle et de gestion des applications et services conteneurisés. Dans ce chapitre, nous allons introduire en détails le gestionnaire des conteneurs **Docker** et l'orchestrateur de conteneurs **Kubernetes**.

## II. Docker

### 1. Définition

Docker est une plate-forme open source qui exécute des applications et facilite leurs développements et leurs distributions. Les applications générées par ce dernier sont empaquetées avec toutes les dépendances et les bibliothèques nécessaires dans un conteneur. Ces conteneurs continuent de fonctionner de manière isolée au-dessus du noyau du système d'exploitation. [13]

### 2. Architécture de docker

Docker utilise une architécture client-serveur. Le client Docker parle au daemon Docker, qui s'occupe de la construction, de l'exécution et de la distribution des conteneurs Docker. Le client et le daemon Docker peuvent s'exécuter sur le même système, il est aussi possible de connecter un client Docker à un daemon Docker distant. Le client et le daemon Docker communiquent à

l'aide d'une API REST, via des sockets UNIX ou une interface réseau.
La plateforme Docker repose sur plusieurs technologies et composants. Voici les principaux éléments :

### 2.1. Docker Engine

Le Docker Engine est l'application à installer sur la machine hôte pour créer, exécuter et gérer des conteneurs Docker. Comme son nom l'indique, il s'agit du moteur du système Docker. C'est ce moteur qui regroupe et relie les différents composants entre eux. C'est la technologie client-serveur permettant de créer et d'exécuter les conteneurs, et le terme Docker est souvent employé pour désigner Docker Engine.
On distingue le Docker Engine Enterprise et le Docker Engine Community. La Docker Community Edition est la version originale, proposée en open source gratuitement.

### 2.2. Docker Daemon

Le Docker Daemon traite les requêtes API afin de gérer les différents aspects de l'installation tels que les images, les conteneurs ou les volumes de stockage.

### 2.3. Docker Client

Le client Docker est la principale interface permettant de communiquer avec le système Docker. Il reçoit les commandes par le biais de l'interface de ligne de commande et les transmet au Docker Daemon.

### 2.4. Dockerfile

Chaque conteneur Docker débute avec un "Dockerfile" . Il s'agit d'un fichier texte rédigé dans une syntaxe compréhensible, comportant les instructions de création d'une image Docker. Un Dockerfile précise le système d'exploitation sur lequel sera basé le conteneur, et les langages, variables environnementales, emplacements de fichiers, ports réseaux et autres composants requis.

### 2.5. Les images Docker

Une image Docker est un modèle en lecture seule, utilisée pour créer des conteneurs Docker. Elle est composée de plusieurs couches empaquetant toutes les installations, dépendances, bi-

bliothèques, processus et codes d'applications nécessaires pour un environnement de conteneur pleinement opérationnel.
Après avoir écrit le Dockerfile, on invoque l'utilitaire "build" pour créer une image basée sur ce fichier. Cette image se présente comme un fichier portable indiquant quels composants logiciels le conteneur exécutera et de quelle façon.

## 2.6. Les conteneurs Docker

Un conteneur Docker est une instance d'image Docker exécutée sur un microservice individuel ou une pile d'application complète. En lançant un conteneur, on ajoute une couche inscriptible sur l'image. Ceci permet de stocker tous les changements apportés au conteneur durant le runtime.

## 2.7. Le registre Docker (dépôt)

Le registre Docker est un système de catalogage permettant l'hébergement et le "push and pull " des images Docker. Il est possible d'utiliser un registre local, ou l'un des nombreux services de registre hébergés par des tiers comme Red Hat Quay, Amazon ECR, Google Container Registry.
Le Docker Hub est le registre officiel de Docker. Il s'agit d'un répertoire SaaS permettant de gérer et de partager les conteneurs. On peut y trouver des images Docker de projets open source ou de vendeurs logiciels. Il est possible de télécharger ces images et de partager d'autres.
Un registre Docker organise les images dans différents répertoires de stockage. Chacun d'entre eux contient différentes versions d'une image Docker partageant le même nom d'image. [14]

Pour bien comprendre Docker voici la figure 2.1 qui montre en détails l'architécture ainsi que les différents composants.

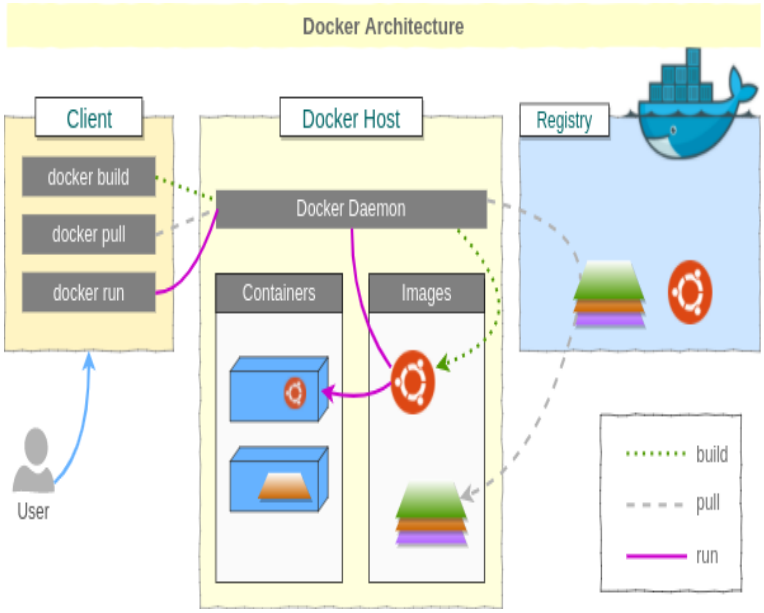


FIG. 2.1 : Architécture de Docker [15]

# III. Ordonnancement

L'ordonnancement est une méthode utilisée pour distribuer les ressources informatiques, généralement du temps processeur, de la bande passante et de la mémoire, aux différents processus, threads, flux de données et applications qui en ont besoin. Il est effectué pour équilibrer la charge sur le système et assurer une répartition égale des ressources et donner une certaine priorisation selon des règles établies. Cela garantit la capacité d'un système informatique à répondre à toutes les demandes et atteindre une certaine qualité de service.
L'ordonnancement est largement basé sur plusieurs facteurs et varie en fonction du système d'exploitation et de sa programmation ou des préférences et objectifs de l'utilisateur.[16]

## 1. Fonctionnement de l'ordonnancement

L'ordonnancement de conteneurs fonctionne avec des outils tels que Kubernetes et Docker Swarm. Les fichiers de configuration indiquent à l'outil d'ordonnancement de conteneurs comment établir un réseau entre les conteneurs et où stocker les journaux. L'outil d'ordonnancement planifie également le déploiement de conteneurs dans des clusters et détermine le meilleur hôte

pour le conteneur. Une fois l'hôte choisi, l'outil d'ordonnancement gère le cycle de vie du conteneur en fonction de spécifications prédéterminées. Les outils d'ordonnancement de conteneurs fonctionnent dans n'importe quel environnement qui exécute des conteneurs. [17]

## 2. Exemple d'ordonnanceurs

Il existe plusieurs ordonnanceurs de conteneurs, nous citerons trois ordonnanceurs qui sont Docker swarm, Mesos et Kubernetes. Nous allons présenter brièvement les deux premiers et parlerons en détails du dernier.

### 2.1. Docker Swarm

Docker Swarm est un groupe de machines physiques ou virtuelles qui exécutent l'application Docker et qui ont été configurées pour se réunir dans un cluster. Une fois qu'un groupe de machines a été regroupé, il est toujours possible d'exécuter les commandes Docker auxquelles l'utilisateur est habitué, mais elles seront désormais exécutées par les machines du cluster.
Les activités du cluster sont contrôlées par un gestionnaire de swarm et les machines qui ont rejoint le cluster sont appelées nœuds travailleurs. Docker swarm est un outil d'ordonnancement de conteneurs, ce qui signifie qu'il permet à l'utilisateur de gérer plusieurs conteneurs déployés sur plusieurs machines. [18]

### 2.2. Mesos

Apache Mesos est un gestionnaire de cluster open source qui gère les charges de travail dans un environnement distribué via le partage et l'isolement dynamiques des ressources. Mesos est adapté au déploiement et à la gestion d'applications dans des environnements en cluster à grande échelle. Mesos rassemble les ressources existantes des machines/nœuds d'un cluster dans un pool unique à partir duquel diverses charges de travail peuvent être utilisées. Également connu sous le nom d'abstraction de nœud, cela élimine le besoin d'allouer des machines spécifiques pour différentes charges de travail. Des entreprises telles que Twitter, Airbnb et Xogito utilisent Apache Mesos. Mesos isole les processus exécutés dans un cluster, tels que la mémoire, le processeur, le système de fichiers, la localité du rack et les E/S, pour les empêcher d'interférer les uns avec les autres. Une telle isolation permet à Mesos de créer un seul et grand pool de ressources pour offrir différentes charges de travail. [19]

# IV. Kubernetes(k8s)

## 1. Définition

Kubernetes est une plate-forme d'orchestration open-source pour le déploiement, la mise à l'échelle et la gestion automatiques des applications conteneurisées. Bien que les conteneurs fournissent déjà un haut niveau d'abstraction, ils doivent encore être correctement gérés, en particulier en termes d'ordonnancement des ressources, d'équilibrage de charge et de distribution des serveurs, et c'est là que les solutions intégrées comme Kubernetes prennent tout leur sens. Kubernetes simplifie le déploiement de systèmes distribués fiables et évolutifs en gérant leur flux de travail complet tout au long de leur cycle de vie.[20]

## 2. Historique

Kubernetes (du grec « timonier » ou « pilote » ou « gouverneur », et la racine étymologique de la cybernétique) a été fondée par Ville Aikas, Joe Beda, Brendan Burns et Craig McLuckie, qui ont été rapidement rejoints par d'autres ingénieurs de Google, dont Brian Grant et Tim Hockin. Il a été annoncé pour la première fois par Google à la mi-2014. Son développement et sa conception sont fortement influencés par le système Borg de Google, et bon nombre des principaux contributeurs au projet ont déjà travaillé sur Borg.
Le nom de code original de Kubernetes au sein de Google était Project 7, une référence au personnage de Star Trek ex-Borg Seven of Nine. Les sept rayons sur la roue du logo Kubernetes font référence à ce nom de code. Le projet Borg original a été entièrement écrit en C++, mais le système Kubernetes réécrit est implémenté en Go.
Kubernetes v1.0 est sorti le 21 juillet 2015. Parallèlement à la version Kubernetes v1.0, Google s'est associé à la Linux Foundation pour former la Cloud Native Computing Foundation (CNCF) et a proposé Kubernetes comme technologie de départ. En février 2016, le gestionnaire de packages Helm pour Kubernetes a été publié. Le 6 mars 2018, le projet Kubernetes a atteint la neuvième place des commits sur GitHub et la deuxième place des auteurs et des problèmes, après le noyau Linux.

## 3. Les concepts de K8s

### 3.1. Pods

Un Pod est l'unité d'exécution de base d'une application Kubernetes considéré comme étant l'unité la plus petite et la plus simple dans le modèle d'objets de Kubernetes qui peut être créé ou déployée. Un Pod représente des processus en cours d'exécution dans le cluster.
Il encapsule un conteneur applicatif (ou, dans certains cas, plusieurs conteneurs), des ressources de stockage, une identité réseau (adresse IP) unique, ainsi que des options qui contrôlent comment le ou les conteneurs doivent s'exécuter. Un Pod représente une unité de déploiement : Une instance unique d'une application dans Kubernetes, qui peut consister soit en un unique conteneur soit en un petit nombre de conteneurs qui sont étroitement liés et qui partagent des ressources.
Chaque Pod est destiné à exécuter une instance unique d'une application donnée. [21]

### 3.2. Services

Un service permet d'exposer de manière abstraite une application s'exécutant sur un ensemble de pods en tant que service réseau. Il définit un ensemble logique de pods et une politique permettant d'y accéder (parfois ce modèle est appelé micro-service). L'ensemble des pods ciblés par un service est généralement déterminé par un sélecteur. [22]

### 3.3. Volume

Un volume est un annuaire accessible à tous les conteneurs d'un pod. La source du volume déclaré dans la spécification du pod détermine le mode de création du répertoire, le support de stockage utilisé et le contenu initial du répertoire. Un pod spécifie les volumes qu'il contient et le chemin d'accès où les conteneurs installent le volume. [23]

### 3.4. Load Balancer

Un service d'équilibrage de charge qui agit comme un contrôleur de trafic, réalisant le routage des demandes des clients vers les nœuds capables de les servir rapidement et efficacement. Lorsqu'un hôte tombe en panne, l'équilibreur de charge redistribue sa charge de travail entre les nœuds restants. D'autre part, lorsqu'un nouveau nœud rejoint un cluster, le service commence automatiquement à envoyer des requêtes aux pods qui lui sont attachés. Dans les clusters

Kubernetes, un service Load Balancer effectue les tâches suivantes :

- Répartition efficace des charges réseaux et des demandes de service sur plusieurs instances.
- Activation de l'autoscaling (mise à l'échelle automatique) en réponse aux changements de la demande
- Assurer la haute disponibilité en envoyant des charges de travail à des pods sains.[24]

## 4. Les composants de K8s

Il existe de nombreux composants dans la plateforme d'orchestration kubernetes, nous citons :

### 4.1. Composants du noeud principal

Les composants du noeud principal fournissent le plan de contrôle (control plane) du cluster. Ils prennent des décisions globales à propos du cluster (par exemple, l'ordonnancement (scheduling)). Ils détectent et répondent aux événements du cluster.
Ils peuvent être exécutés sur n'importe quelle machine du cluster. Toutefois, par soucis de simplicité, les scripts de mise en route démarrent typiquement tous ces composants sur la même machine et n'exécutent pas de conteneurs utilisateur sur cette machine.
Les composants du noeud principal sont les suivants :

a. **kube-apiserver :** Composant qui expose l'API Kubernetes. Il s'agit du front-end pour le plan de contrôle Kubernetes. Il est conçu pour une mise à l'échelle horizontale, ce qui veut dire qu'il réalise la mise à l'échelle en déployant des instances supplémentaires.

b. **Etcd :** Base de données clé-valeur consistante et hautement disponible utilisée comme mémoire de sauvegarde pour toutes les données du cluster.

c. **kube-scheduler :** Composant qui surveille les pods nouvellement créés qui ne sont pas assignés à un nœud et sélectionne un nœud sur lequel ils vont s'exécuter.
   Les facteurs pris en compte pour les décisions d'ordonnancement (scheduling) comprennent les exigences individuelles et collectives en ressources, les contraintes matérielles, logicielles, politiques, les spécifications d'affinité et d'anti-affinité, la localité des données, les interférences entre charges de travail et les dates limites.

d. **kube-controller-manager :** Composant qui exécute les contrôleurs.[24]

### 4.2. Composants des nœuds travailleurs

Ils s'exécutent sur chaque nœud travailleur, en maintenant les pods en exécution et en fournissant l'environnement d'exécution Kubernetes. On cite :

a. **Kubelet :** Un agent qui s'exécute sur chaque nœud travailleur du cluster. Il s'assure que les conteneurs fonctionnent dans un pod. Le kubelet prend un ensemble de PodSpecs fournis par divers mécanismes et s'assure du fonctionnement et de la santé des conteneurs décrits dans ces PodSpecs. Le kubelet ne gère que les conteneurs créés par Kubernetes.

b. **Kube-proxy :** C'est un proxy réseau qui s'exécute sur chaque nœud travailleur du cluster et implémente une partie du concept Kubernetes de Service. Il maintient les règles réseau sur les nœuds. Ces règles réseau permettent une communication réseau vers les Pods depuis des sessions réseau à l'intérieur ou à l'extérieur du cluster. kube-proxy utilise la couche de filtrage de paquets du système d'exploitation s'il y en a une et si elle est disponible. Sinon, il transmet le trafic lui-même.

c. **Container runtime :** L'environnement d'exécution de conteneurs est le logiciel responsable de l'exécution des conteneurs. Kubernetes est compatible avec plusieurs environnements d'exécution de conteneur.

d. **Addons :** Les addons utilisent les ressources Kubernetes pour implémenter des fonctionnalités cluster.

e. **DNS :** Le DNS Cluster est un serveur DNS, en plus des autres serveurs DNS dans l'environnement, qui sert les enregistrements DNS pour les services Kubernetes. Les conteneurs démarrés par Kubernetes incluent automatiquement ce serveur DNS dans leurs recherches DNS. [24]

## 5. Architécture

L'architécture suit le modèle maître-esclave, où au moins un nœud maître gère les conteneurs Docker sur plusieurs nœuds de travail (esclaves). Ces nœuds peuvent être des serveurs physiques locaux et des machines virtuelles ou même des clouds publics et privés. Le maître est responsable de l'exposition du serveur API (Application Program Interface), de l'ordonnancement des services de déploiements et de la gestion globale du cluster. [20]. Comme le montre la figure 2.2.

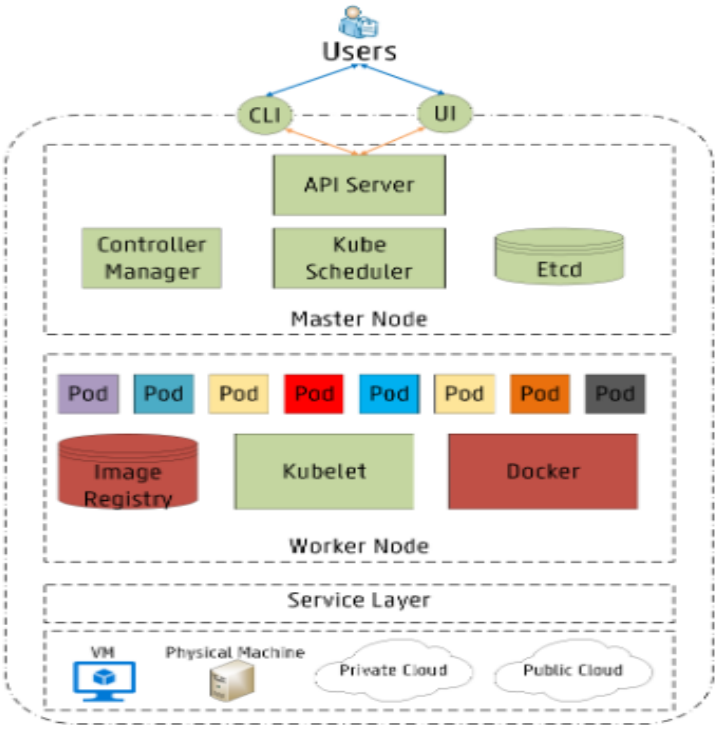


FIG. 2.2 : Architécture de Kubernetes [20]

# Partie 2: Stratégies d'ordonnancements

## V. Les stratégies d'ordonnancement de conteneurs

Elles permettent d'organiser les conteneurs et de gérer les ressources selon des critères définis. Elles sont classifiées en quatre catégories qui sont :

### 1. Modélisation mathématique

Les techniques de modélisation mathématique modélisent le problème d'ordonnancement sous forme d'ensemble d'équations telles que la formulation de la programmation linéaire en nombre entier (ILP), puis utilisent des techniques standards pour trouver une solution optimale au problème. Cependant, en raison de son coût de calcul prohibitif, cette technique ne peut être utilisée que pour des problèmes de petite taille.
Comme le problème d'ordonnancement de conteneur est un problème NP-difficile, il n'y a pas d'algorithme de complexité polynomiale pour trouver un ordonnancement optimal pour les problèmes de grande taille.[25]

### 2. Technique heuristique

C'est une technique de résolution de problèmes qui donne une solution simple et facile à mettre en œuvre, qui n'est pas nécessairement optimale. Cette technique n'est pas parfaite, mais elle donne rapidement des solutions quand les méthodes classiques sont trop lentes, et peut trouver une solution approximative lorsqu'il n'y a pas de solution exacte.
Elle utilise une méthode pratique qui n'est pas garantie optimale, parfaite ou rationnelle, mais qui est néanmoins suffisante pour atteindre un objectif ou une approximation immédiate et à court terme. Ceci est réalisé en sacrifiant soit l'optimalité, l'exhaustivité, l'exactitude ou la précision pour la vitesse.[26]

### 3. Technique méta-heuristique

Les métaheuristiques sont des stratégies qui « guident » le processus de recherche. Ils sont approximatifs et généralement non déterministes. L'objectif est d'explorer efficacement l'espace

de recherche afin de trouver des solutions (presque) optimales. Les techniques qui constituent les algorithmes méta-heuristiques vont des simples procédures de recherche locale aux processus d'apprentissage complexes.[27]

### 4. Technique de Machine Learning

L'apprentissage automatique se concentre sur le développement de programmes informatiques capables d'accéder aux données et de les utiliser pour apprendre par eux-mêmes. Le processus d'apprentissage commence par des observations ou des données, telles que des exemples, une expérience directe ou des instructions, afin de rechercher des modèles dans les données et de prendre de meilleures décisions à l'avenir en fonction des exemples que nous fournis.[28]
L'ordonnancement avec l'apprentissage machine consiste à réaliser des stratégies ou des séquences d'actions.
Dans des environnements connus avec des modèles disponibles. Les solutions peuvent être trouvées et évaluées avant l'exécution.
Dans des environnements dynamiquement inconnus, la stratégie doit souvent être révisée en ligne. Les modèles et les politiques doivent être adaptés. Les solutions ont généralement recours à des processus d'essais et d'erreurs itératifs couramment observés dans l'intelligence artificielle.

## VI. Stratégies d'ordonnancement de k8s

Il existe une stratégie d'ordonnancemnt par défaut dans Kubernetes afin de trouver un nœud pour un Pod nouvellement créé. Si un pod est ajouté à la file d'attente, l'ordonnanceur recherche un nœud approprié pour le déploiement en fonction d'un processus qui se fait en deux phases.

### 1. Phase de filtrage des noeuds(Prédicats)

La première phase est appelée filtrage des nœuds, où l'ordonnanceur vérifie quels nœuds sont capables d'exécuter le pod en appliquant un ensemble de filtres, également appelés prédicats.
Parmi les prédicats supportés par l'ordonnanceur de k8s on cite :

### 1.1. PodFitsResources

Si la quantité libre de ressources (CPU et mémoire) sur un nœud donné est inférieure à celle requise par le pod, le nœud ne doit pas être pris en compte dans le processus d'ordonnancement. Par conséquent, le nœud est disqualifié.

### 1.2. CheckNodeMemoryPressure

Ce prédicat vérifie si un pod peut être alloué sur un nœud signalant un état de pression de la mémoire.

### 1.3. CheckNodeDiskPressure

Ce prédicat évalue si un pod peut être ordonnancé sur un nœud signalant un état de pression du disque.

### 1.4. MatchNodeSelector (Affinity/Anti-Affinity) :

En utilisant des sélecteurs de nœuds (étiquettes), il est possible de définir qu'un pod donné ne peut s'exécuter que sur un ensemble particulier de nœuds avec une valeur d'étiquette exacte (affinité de nœud), ou même qu'un pod doit éviter d'être alloué sur un nœud qui a déjà certains pods déployés (pod-anti-affinité). Ces règles peuvent être créées en déclarant des tolérances dans les fichiers de configuration de pod pour correspondre à des taintes de nœud spécifiques. Essentiellement, les règles d'affinité sont des propriétés des pods qui les attirent vers un ensemble de nœuds ou de pods tandis que les taintes permettent aux nœuds de repousser un ensemble donné de pods.[20]

## 2. Phase de calcul de priorité(Priorité)

La deuxième phase est appelée calcul de la priorité du nœud, cette phase est basée sur un ensemble de priorités, avec un score de 0 à 10 attribué à chaque nœud restant. Chaque priorité est pondérée par un nombre positif, et le score ultime de chaque nœud est généré en additionnant tous les scores pondérés. Le pod est exécuté dans le nœud qui a le score le plus élevé.

S'il y a plusieurs nœuds qui ont le même meilleur score, alors l'un de ces noeuds est choisi

alééatoirement.
Parmi les priorités supportées par l'ordonnanceur de k8s on cite :

### 2.1. LeastRequestedPriority

Les fractions CPU et mémoire (libres/allouées) sont utilisées pour attribuer un score au nœud. Le déploiement doit être effectué sur le nœud qui a la fraction libre la plus élevée. En se basant sur l'utilisation des ressources, cette fonction de priorité distribue les pods sur le cluster.[20]

### 2.2. NodeAffinityPriority

Dans ce cas, les nœuds sont triés selon les règles d'affinité de nœud. Par exemple, les nœuds ayant une étiquette spécifique reçoivent un score plus élevé que les autres.[20]

### 2.3. TaintTolerationPriority

Cette fonction de priorité attribue un score aux nœuds en fonction de leurs taintes, ainsi que les tolérances qui sont définies dans les fichiers de configuration de pod. Les nœuds restants sont choisis en fonction du nombre de taintes intolérables qu'ils ont pour le pod.[20]

### 2.4. InterPodAffinity

Les règles d'affinité des pods sont utilisées pour attribuer un score aux nœuds dans cet algorithme de priorité.
Les nœuds avec certains pods déjà deployés reçoivent un score plus élevé car il est préférable de déployer le pod fourni à proximité de ces pods.[20]

Les prédicats sont utilisés pour éliminer les nœuds qui ne peuvent pas exécuter le pod fourni, tandis que les priorités sont utilisées pour classer tous les nœuds restants qui sont capables de déployer le pod.[20]

## VII. Les métriques de performance

Les métriques de performance sont des critères qui permettent de déterminer la qualité et l'efficacité d'une stratégie d'ordonnancement. Il existe de nombreuses métriques parmi lesquels :

## 1. Efficacité énergétique

L'efficacité énergétique signifie utiliser moins d'énergie lors du déploiement de conteneurs sur des nœuds de travail.[25]
L'objectif de cette métrique est de minimiser la consommation électrique globale du cluster. C'est l'un des moyens les plus simples d'éliminer le gaspillage d'énergie et de réduire les coûts énergétiques.[29]

## 2. Coût

Le coût global de l'exécution de l'application dépend du coût de divers services tels que le calcul, le stockage et le coût de communication. Le coût de calcul fait référence au temps passé à exécuter une application sur les nœuds disponibles dans le cluster. Plus l'application passe de temps sur un processeur, plus le coût est élevé. Le coût de stockage fait référence à l'espace de stockage occupé par les applications. De plus, les coûts de communication font référence au coût payé aux fournisseurs de réseau pour fournir les services de télécommunications nécessaires à l'exécution de l'application. L'augmentation de la communication nécessaire entre les nœuds/clusters augmente les coûts de communication.[25]

## 3. Utilisation des ressources

Cette métrique permet de comprendre quelles ressources sont utilisées, comment elles sont utilisées et si leur application est appropriée. L'objectif est d'identifier les utilisations inefficaces des ressources, puis d'élaborer une stratégie pour améliorer l'utilisation des ressources.[30]

## 4. Equilibrage de charge

Il s'agit du processus de répartition uniforme de la charge de travail entre les ressources informatiques de sorte qu'aucun nœud informatique ne soit surchargé alors que d'autres sont inactifs. Cet objectif a également un impact sur le temps de réponse, le coût et le débit. Cette métrique est importante, en particulier dans les applications de conteneurs qui utilisent des architéctures de micro-services en raison de la fluctuation dynamique de la charge de travail.[25]

# VIII. Les travaux existants

## 1. Stratégie d'ordonnancement basée sur la technique méta-heuristique

### 1.1. Description de la solution

#### 1.1..1 Algorithme de colonies de fourmis (ACO)

L'optimisation des colonies de fourmis (ACO) est un algorithme méta-heuristique basé sur une population importante de fourmis qui peut être utilisé pour trouver des solutions approximatives à des problèmes d'optimisation difficiles.

Dans ACO, un ensemble d'agents logiciels appelés fourmis artificielles recherchent de bonnes solutions à un problème d'optimisation donné. Pour appliquer ACO, le problème d'optimisation est transformé en problème de recherche sur un graphe pondéré. Les fourmis artificielles construisent progressivement des solutions en se déplaçant sur le graphe.

Le processus de construction de la solution est stochastique et est biaisé par un modèle de phéromone, c'est-à-dire un ensemble de paramètres associés à des composants de graphe (nœuds ou arêtes) dont les valeurs sont modifiées au moment de l'exécution par les fourmis. [31]

Voici la figure 2.3 qui montre l'organigramme du fonctionnement de l'algorithme d'optimisation de colonies de fourmis(ACO).

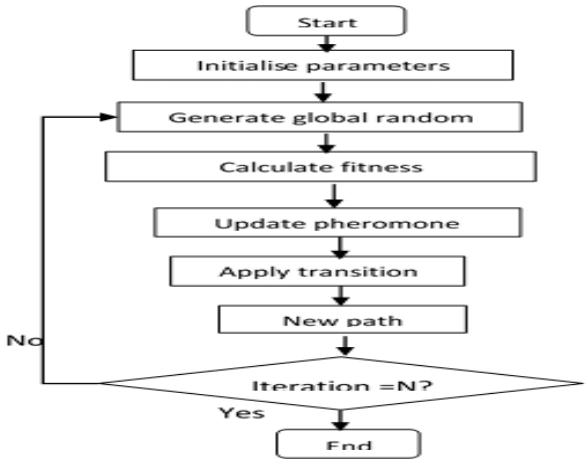


FIG. 2.3 : Organigramme du fonctionnement de l'algorithme ACO[32]

### 1.1..2 Algorithme d'optimisation par essaim particulaire (PSO)

L'optimisation des essaims de particules (PSO) est un algorithme modélisé sur l'intelligence en essaim, qui trouve une solution à un problème d'optimisation dans un espace de recherche, ou modélise et prédit les comportements sociaux en présence d'objectifs. PSO est un algorithme stochastique basé sur la population, modélisé sur l'intelligence en essaim. L'intelligence en essaim est basée sur les principes de la psychologie sociale et donne un aperçu des comportements sociaux, tout en contribuant à l'ingénierie des applications.
Un problème est donné, une méthode d'évaluation de la solution proposée existe sous la forme d'une fonction de fitness. Une structure de communication est également définie, attribuant des voisins avec lesquels chaque individu peut intéragir. Ensuite, une population d'individus définie comme des suppositions aléatoires aux solutions du problème est initialisée. Ces individus sont des solutions candidates. Ils sont également connus sous le nom de particules, d'où le nom d'essaim de particules. Un processus itératif d'amélioration de ces solutions candidates est mis en place.
Les particules évaluent de manière itérative l'adéquation des solutions candidates et mémorisent l'endroit où elles ont eu leur meilleur succès. La meilleure solution de l'individu est appelée la meilleure particule ou la meilleure solution locale. Chaque particule met cette information à disposition de ses voisines.
Chaque particule représente une solution candidate au problème d'optimisation. La position d'une particule est influencée par la meilleure position visitée par elle-même, c'est-à-dire sa propre expérience et la position de la meilleure particule dans son voisinage, (l'expérience des particules voisines). [33]

Voici la figure 2.4 qui montre l'organigramme du fonctionnement de l'algorithme d'optimisation par essaim particulaire (PSO).

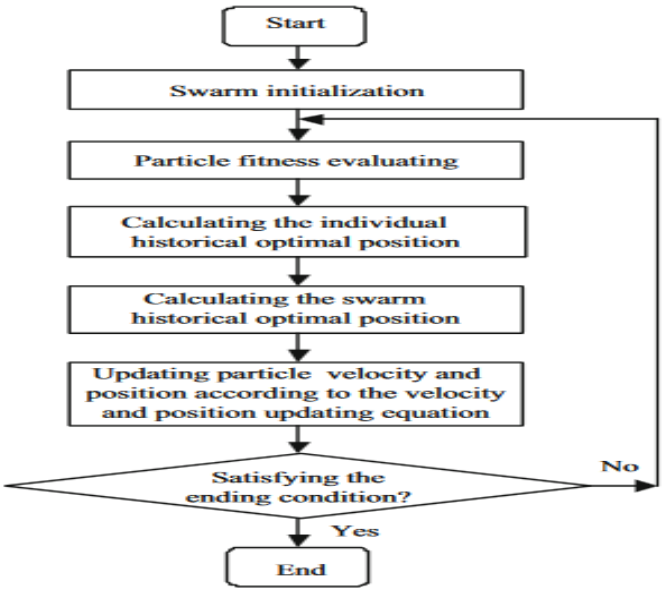


FIG. 2.4 : Organigramme du fonctionnement de l'algorithme PSO [34]

### 1.1..3 Hybridation entre ACO et PSO

Dans ce travail, les auteurs de [31] introduisent la fonction objective pour la minimisation du coût dans le but d'améliorer le modèle d'ordonnancement par défaut de kubernetes en assurant la haute disponibilité et la fiabilité des conteneurs.

Avec l'amélioration de l'algorithme ACO par l'ajout de la notion de volatilité adaptative de phéromone. Cet algorithme permet d'ordonnancer les ressources d'une manière efficace et d'obtenir des solutions optimales en peu de temps.

Si chaque fourmi attribue une tâche à un nœud avec la concentration de phéromones la plus élevée, la stagnation se produit. C'est-à-dire que l'algorithme converge prématurément vers une solution optimale locale et que la solution optimale globale ne peut pas être trouvée. Par conséquent, certaines fourmis doivent suivre la stratégie d'allocation de phéromones la plus élevée, et certaines fourmis doivent suivre une stratégie assignée aléatoirement pour découvrir de nouvelles stratégies locales afin d'obtenir des solutions optimales.

L'algorithme PSO est amélioré avec la combinaison des choix aléatoires des facteurs et la masse inertie pour optimiser le schéma d'ordonnancement des ressources.

En comparant la valeur optimale historique de la particule avec la solution optimale globale de toutes les particules, le rapport pondéral de la vitesse de chaque particule itérative est déterminé. Si la valeur optimale de la particule itérative à l'instant $t$ est inférieure à la valeur optimale de l'itération précédente (l'instant $t-1$) de la particule et inférieure à la valeur opti-

male globale de la particule, cela indique que la direction de vol de la particule est correcte et que l'effet est bon.

### 1.2. Résultat des tests de la solution

Les auteurs ont montré avec des expérimentations que lorsqu'il y a une augmentation du nombre de tâches, l'algorithme d'ordonnancement par défaut kube-scheduler a un coût d'exécution supérieur à l'algorithme ACO et l'algorithme ACO-PSO, tandis que l'algorithme combiné et l'algorithme ACO ont un coût quasi similaire comme le montre la figure 2.5 :

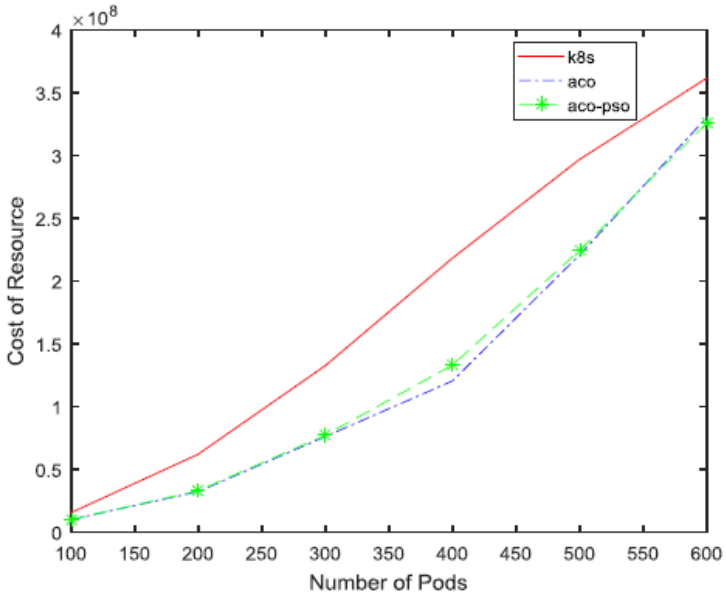


FIG. 2.5 : Comparaison des coûts d'exécution des tâches [31]

On peut aussi voir sur la figure 2.6 que la charge maximale de l'algorithme combiné et l'algorithme ACO est inférieure à la charge maximale de l'algorithme d'ordonnancement kube-scheduler, tandis que l'algorithme combiné rend la charge globale de l'ordonnanceur du cluster Kubernetes plus équilibrée et plus stable par rapport à l'algorithme ACO comme le montre la figure 2.6 :

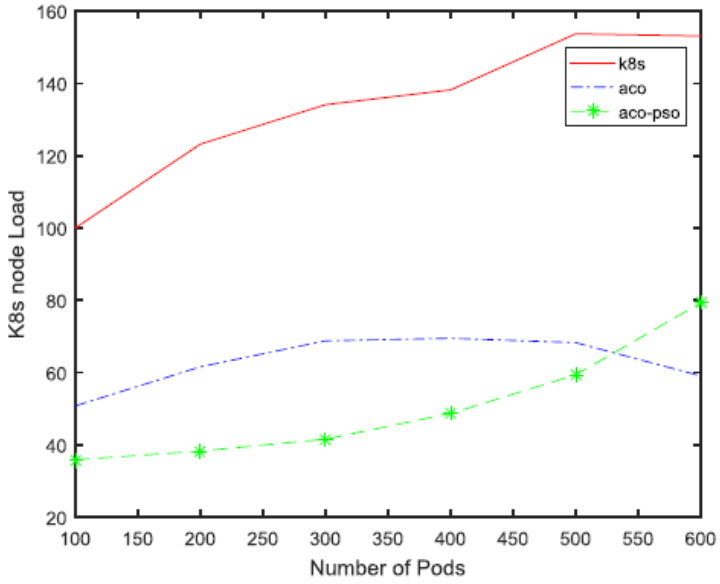


FIG. 2.6 : Comparaison de la charge maximale d'affectation des tâches [31]

A travers les résultats de simulation ci-dessus, les auteurs ont conclu que l'ordonnancement des pods basé sur l'algorithme ACO-PSO réduit non seulement le coût des tâches, mais équilibre également la charge de l'ensemble du système, améliorant ainsi l'utilisation des ressources du système.

Les auteurs ont considéré le coût des ressources, et en même temps de l'équilibrage de charge afin de réaliser l'allocation la plus raisonnable des pods sur Kubernetes. L'algorithme qu'ils ont proposé a pour but d'améliorer le schéma d'allocation de l'ordonnanceur de k8s et réduire le coût des ressources et la charge maximale.

## 1.3. Avantages et inconvénients

### Avantages

- Réduction du coût d'exécution.
- Meilleur équilibrage de charges.
- Prend en considération la disponibilité.

**Inconvénients**

- Consommation d'énergie.
- Pas de mise à l'échelle.

## 2. Stratégie d'ordonnancement basée sur la technique heuristique

Dans l'article [35], les auteurs ont proposé une solution d'ordonnancement basée sur une technique heuristique appelée ProCon qui a pour objectif d'améliorer le temps d'exécution, la performance tout en prenant en considération l'utilisation instantanée des ressources et l'estimation de l'utilisation future des ressources.

### 2.1. Description de la solution

Dans ce travail, les auteurs ont changé les critères de choix du nœud qui exécute les conteneurs. Par défaut la sélection des nœuds se base sur différents critères parmi lesquels l'état des nœuds travailleurs, la disponibilité des ressources ainsi que les spécifications des conteneurs. Cette méthode diffère de la solution par défaut de kubernetes dans le fait qu'elle supervise la progression de l'exécution des conteneurs en tenant compte de l'utilisation actuelle des ressources dans les nœuds travailleurs. En outre, elle utilise les données collectées pour calculer le taux de conflit attendu et affecter les conteneurs entrants pour équilibrer le conflit de ressources sur les nœuds travailleurs réduisant ainsi le temps d'exécution des conteneurs.
Le système ProCon se compose de trois principaux modules qui sont :

a. **Container monitor (Superviseur de conteneur) :**
   Ce module est exécuté dans chaque nœud travailleur, le moniteur de conteneur assure le suivi de l'état des tâches en cours d'exécution en faisant le marquage (recording) du temps et de la consommation de ressources moyenne pour l'exécution de chaque conteneur. Ces données vont être stockées sous forme des fichiers log dans un volume persistant.

b. **Log analyst (Analyseur des fichiers journaux) :**
   Ce module est exécuté dans le nœud principal qui a une vue globale sur les nœuds travailleurs, Il analyse les fichiers journaux générés par les moniteurs de conteneurs dans ces nœuds pour calculer les ressources requises pour l'exécution des conteneurs et prendre les décisions de placement des conteneurs.

c. **ProCon Schedular (L'ordonnanceur ProCon) :**

Ce module est exécuté dans le nœud principal, il a pour but de collecter les données à partir de l'analyseur des fichiers journaux et de calculer un score pour chaque nœud travailleur. Il sélectionne le nœud travailleur pour héberger un conteneur entrant en fonction du score.

En considérant que des conteneurs multiples sont exécutés en parallèle dans le cluster sur différents nœuds travailleurs.
L'estimation du temps d'exécution des conteneurs est basée sur le calcul de la différence du taux de progression entre différents nœuds qui exécutent le même conteneur à un instant t donné.
L'utilisation des ressources est calculée à partir du rapport entre la quantité de ressources utilisée par un conteneur et la somme des quantités utilisées par l'ensemble des conteneurs puisque les ressources sont partagées entre les conteneurs.

## 2.2. Résultat des tests

Cette solution a été implémenté dans Kubernetes et testée dans un environnement CloudLab avec l'utilisation de deux clusters :

- **Cluster 1:** 1 nœud principal et quatre nœuds travailleurs.
- **Cluster 2:** 1 nœud principal et sept nœuds travailleurs.

Les expérimentations ont montré que la solution ProCon réduit le temps d'exécution des conteneurs de 53.3% et garantit l'amélioration de la performance avec un taux de 37.4% quand elle est comparée avec la solution par défaut de kubernetes.

## 2.3. Avantages et inconvénients

### Avantages

- Utilisation efficace des ressources avec la prédiction des futurs besoins.
- Equilibrage de charge.
- Temps d'exécution des conteneurs optimisé.

### Inconvénients

- Besoin de supervision pour un bon fonctionnement.
- Consommation importante d'énergie.

# 3. Stratégie d'ordonnancement basée sur l'apprentissage machine dans une architécture microservice

## 3.1. Description de la solution

Les auteurs dans [36] ont essayé de se pencher sur un problème peu traité d'après eux, qui est, comment ajuster le nombre de conteneurs en fonction de la pression de charge actuelle des services ?

Pour résoudre ce problème, ils ont proposé un algorithme d'ordonnancement de conteneur basé sur l'apprentissage automatique dans l'architécture de micro services appelé algorithme CSML. Il est combiné avec la technique de l'algorithme de régression de forêt aléatoire qui combine les performances de nombreux algorithmes d'arbres de décision pour classer ou prédire la valeur d'une variable. Selon les quatre vecteurs propres de services de la fenêtre de temps actuelle (accès total, simultanéité, temps de réponse moyen des utilisateurs, taux d'erreur), le nombre de nouveaux conteneurs à exécuter dans la prochaine fenêtre de temps est prédit. Sur la base des résultats prévus, le nombre de conteneurs de services pour la prochaine fenêtre de temps sera ajusté à la fin de la fenêtre de temps actuelle.

Les auteurs ont conduit les expérimentations sur le réseau de l'université. Ils ont utilisé la plate-forme Kubernetes pour gérer les charges de travail et les services conteneurisés. Ainsi que l'application JMeter qui est utilisé pour simuler la pression de charge des applications micro services en modifiant l'accès total (TA) et la concurrence.

Et pour le monitoring, Prometheus qui est un outil de surveillance a été utilisé, il permet de surveiller la qualité de service et la pression de charge des applications micro services en observant le temps de réponse moyen des utilisateurs (ARTU) et le taux d'erreur (ER).

Quatre valeurs seuils appelées $A_{min}$, $A_{max}$, $B_{min}$, $B_{max}$ sont définies. La pression de charge des services d'application dans cet état est considérée comme normale, la valeur d'ARTU est comprise entre $A_{min}$ et $A_{max}$, et la valeur de ER est comprise entre $B_{min}$ et $B_{max}$. Le maintien de l'équilibre de la pression de charge est défini comme le contrôle de ces deux indicateurs dans les limites définies en ajustant le nombre de conteneurs. Dans chaque test, le nombre initial de conteneurs n'est qu'un seul.

En même temps, un ensemble de données expérimentales est enregistré. Ces données composées du vecteur d'entité X et de la cible de régression Y sont appelées ensemble d'entraînement. Afin d'évaluer la qualité des tâches du modèle d'apprentissage, 20% des échantillons sont ex-

traits aléatoirement de l'ensemble de formation pour construire l'ensemble de tests. La rapidité d'ajustement des conteneurs et la précision de la prédiction du nombre de conteneurs sont les critères permettant de juger des mérites et des démérites de l'algorithme.
Une fois le travail préliminaire terminé, un modèle plus précis est obtenu, à l'aide de l'algorithme de régression de forêt aléatoire. En surveillant les données du service sous la fenêtre de temps actuelle en temps réel, le nombre de conteneurs est ajusté dynamiquement pour se préparer à la prochaine fenêtre de temps.

### 3.2. Résultat des tests

Les résultats obtenus en comparant l'effet de l'algorithme CSML et l'algorithme feedback (un algorithme qui ajoute un conteneur lorsque la pression de charge est trop lourde et réduit un conteneur lorsque la pression de charge est trop légère) ont montré que lorsqu'il y a un changement lent de la pression de charge, les performances des deux algorithmes sont à peu près les mêmes en termes de temps de réponse moyen des utilisateurs et de taux d'erreur.
Mais lorsque la pression de charge change radicalement, par rapport à l'algorithme de feedback, l'algorithme CSML peut réagir plus rapidement et plus précisément, maintenant ainsi l'équilibre de charge et évitant le gaspillage de conteneurs.
Les résultats expérimentaux montrent que par rapport à l'algorithme feedback, en utilisant l'algorithme CSML, le temps d'expansion du conteneur est réduit de 50% et la précision est 10% à 38% plus élevée. En résumé, l'algorithme CSML peut fournir aux utilisateurs une meilleure qualité de service que les algorithmes de feedback.
Les auteurs ont conclu que l'algorithme proposé équilibre non seulement la pression de charge actuelle des services, mais améliore également les performances du système. Les résultats de l'expérience montrent que l'utilisation d'algorithmes d'apprentissage automatique permet d'ajuster avec précision et rapidité le nombre de conteneurs en temps réel en fonction de la pression de charge actuelle des services.

### 3.3. Avantages et inconvénients

**Avantages**

- Haute précision.
- Equilibrage de charge et performance.
- Qualité de service.

**Inconvénients**

- Ne fait que de la prédiction des ressources.
- Manque de tests en situation réelle.
- Pas d'améliorations notables dans des conditions normales.

# IX. Analyse et discussion

Selon l'état de l'art vu précédemment Kubernetes a plusieurs limitations parmi lesquelles on cite :

- Kubernetes est conçu pour les entreprises à l'échelle du Web.
- Kubernetes a trop de composants : Par rapport à d'autres plates-formes complexes, Kubernetes gère assez mal l'intégration de différentes parties dans un environnement assez simple.
- L'architécture Kubernetes ne fait pas grand-chose pour surveiller les charges de travail ou s'assurer qu'elles fonctionnent de manière optimale. Il n'alerte pas en cas de problème et ne facilite pas vraiment la collecte de données de surveillance à partir d'un cluster.
- Malgré le fait que Kubernetes nécessite une bonne quantité d'intervention manuelle pour fournir une haute disponibilité, il parvient à rendre assez difficile le contrôle manuel des choses.

Dans la stratégie vue dans le travail [31], nous avons vu que cette stratégie permet de réduire le coût de l'exécution et d'avoir un meilleur équilibrage de charge, mais elle souffre d'une grande consommation d'énergie et ne permet pas d'évolutivité.

Dans la stratégie vue dans le travail [35], nous avons vu que cette stratégie offre une bonne optimisation du temps d'exécution et une utilisation efficace des ressources avec une possibilité de prédiction des besoins futurs, mais souffre elle aussi d'une grosse consommation d'énergie et à besoin de supervision pour bien fonctionner.

Dans la stratégie vue dans le travail [36], nous avons vu que cette stratégie fournit un haut degré de précision et une qualité de service appréciable et de bonnes performances, mais cette différence de performance ne se voit que dans des conditions de charges élevées et que cette stratégie manque de test dans un contexte réel et ne fait principalement que de la prédiction de ressources.

# X. Conclusion

Docker et Kubernetes sont des technologies avec des portées différentes. Docker peut être utilisé sans Kubernetes et vice versa, mais ils fonctionnent bien ensemble. Du point de vue d'un cycle de développement logiciel, le terrain d'accueil de Docker est le développement. D'autre part, Kubernetes brille dans les opérations, permettant l'utilisation de conteneurs Docker existants tout en s'attaquant aux complexités du déploiement, de la mise en réseau, de la mise à l'échelle et de la surveillance.
Dans ce chapitre, nous avons abordé Docker et Kubernetes. Dans le prochain chapitre, nous allons décrir notre solution d'ordonnancement proposée.

# Chapitre 3

# Conception et développement

## Sommaire

# I. Introduction

Après avoir étudié quelques travaux sur les différentes stratégies d'ordonnancement et plus particulièrement celle utilisée par kubernetes, nous avons constaté qu'elle ne répond pas aux besoins dynamiques et effectue l'ordonnancement des pods de la même manière quelque soit la charge de travail.
Tout ceci nous a poussé à établir notre prope stratégie d'ordonnancement qui a pour but de réduire la consommation d'énergie et de la bande passante.
Dans ce chapitre, nous évoquons les travaux dont on s'est inspirés pour notre solution et son processus de réalisation, ainsi que la problématique qu'elle est sensée résoudre, ensuite nous décrivons les différentes phases de notre startégie, et nous cloturons ce chapitre avec l'architecture globale de notre stratégie.

# II. Travaux existants et limites

Bien que kubernetes offre plusieurs avantages tels que le déploiement automatique des pods et leur orchestration dans plusieurs hôtes. Cependant, il possède des limites qui nous ont poussé à proposer notre stratégie d'ordonnancement. Parmi ces limites on peut citer :

- Kubernetes ne peut pas garantir automatiquement que les ressources soient correctement allouées entre différentes charges de travail exécutées dans un cluster.
- L'efficacité énergétique ne s'adapte pas aux différentes charges de travail.
- L'architecture de Kubernetes ne permet pas de surveiller les charges de travail ou de s'assurer qu'elles fonctionnent de manière optimale.

# III. Réalisation de la solution

Pour réaliser notre solution, nous allons nous focaliser sur deux travaux :

## 1. Une stratégie d'ordonnancement méta-heuristique basée sur l'hybridation entre ACO et PSO

Cette solution a été réalisée dans le but de résoudre le problème de coût des ressources en utilisant un algorithme méta-heuristique et en ajoutant la fonction objective de coût pour

améliorer le modèle de l'ordonnanceur de k8s.

### 1.1. Détails de la solution

Les auteurs de ce papier [31] ont combiné deux algorithmes ACO(Ant Colony Optimization) et PSO(Particle Swarm Optimization) pour réaliser leur solution, mais dans notre solution nous retenons seulement l'algorithme ACO dans un souci de simplicité.

L'optimisation des colonies de fourmis (ACO) est un algorithme méta-heuristique basé sur une population importante de fourmis qui peut être utilisé pour trouver des solutions approximatives à des problèmes d'optimisation difficiles.

Dans ACO, un ensemble d'agents logiciels appelés fourmis artificielles recherchent de bonnes solutions à un problème d'optimisation donné. Pour appliquer ACO, le problème d'optimisation est transformé en un problème de recherche sur un graphe pondéré. Les fourmis artificielles construisent progressivement des solutions en se déplaçant sur le graphe. Le processus de construction de la solution est stochastique et est biaisé par un modèle de phéromone, c'est-à-dire un ensemble de paramètres associés à des composants de graphe (nœuds ou arêtes) dont les valeurs sont modifiées au moment de l'exécution par les fourmis.

La première phase consiste à initialiser un nombre de fourmis à $n$ et un nombre de nœuds à $m$. $\pi_{ij}(t)$ Représente la concentration de phéromone sur le chemin entre le nœud $i$ et le nœud $j$ à l'instant $t$. $allow_k$ représente la liste des noeuds qui peuvent être visités par la fourmi $k$. $\mu_{ij}$ est égal au score total assigné au nœud $i$ par le pod $j$.

Le calcul de la probabilité qu'une fourmi $k$ sélectionne un nœud $j$ à partir du nœud $i$ est :

$$P_{i,j}^{k} = \begin{cases} \frac{[\pi_{i,j}(t)]^{\alpha}[\mu_{i,j}(t)]^{\beta}}{\sum_{k \in allow}[\pi_{i,j}(t)]^{\alpha}[\mu_{i,j}(t)]^{\beta}} & j \in allow_k \\ 0 & other \end{cases} \tag{1}$$

La mise à jour de la concentration de phéromone est calculée selon les équations suivantes :

$$\pi_{i,j}(t+1) = (1-\rho)\pi(t) + \Delta\pi_{i,j}(t) \tag{2}$$

$$\Delta\pi_{i,j}(t) = \frac{S_2}{f_k(m_{i,j})} \tag{3}$$

tels que :

$\Delta_{ij}$ : La variation de l'équilibre des ressources du nœud j après le déploiement de la tâche $i$.

$S_2$ : est une constante.

$f_k(m_ij)$ : Représente le score d'égalisation de la ressource de nœud après l'affectation de Pod $j$ au nœud $i$. Le calcul de score total se fait par l'équation suivante :

$$\mu = Score1 + Score2 \tag{4}$$

tels que :

$\mu_{ij}$ : Score assigné au nœud $i$ par le pod $j$.

Où le $Score1$ est calculé à travers l'équation suivante :

$$Score1 = \frac{S_{mem} + S_{cpu}}{2} \tag{5}$$

tels que :

$$S_{mem} = \frac{T_{mem} - S_{Reqmem} - P_{Reqmem}}{T_{mem}} \tag{6}$$

Où :

$T_{mem}$ : La quantité de mémoire totale allouée au pod dans un nœud.

$S_{Reqmem}$ : La quantité de mémoire consommée par le pod.

$P_{Reqmem}$ : La quantité de mémoire nécessaire pour le prochain pod.

Et :

$$S_{cpu} = \frac{T_{cpu} - S_{Reqcpu} - P_{Reqcpu}}{T_{cpu}} \tag{7}$$

Où :

$T_{cpu}$ : La quantité de cpu totale allouée au pod dans un nœud.

$S_{Reqcpu}$ : La quantité de cpu consommée par le pod.

$P_{Reqcpu}$ : La quantité de cpu nécessaire pour le prochain pod.

Le score 2 est calculé avec l'équation suivante :

$$Score2 = \frac{(1 - abs(S_{cpu} - S_{mem})}{2} \tag{8}$$

Nous avons considéré cette solution parce qu'elle offre un bon placement des conteneurs dans les nœuds et une meilleure disponibilité tout en réduisant le coût d'exécution. Par contre elle a certaines limites en termes de consommation d'énergie et n'offre pas d'évolutivité.

## 2. Placement économe en énergie de machine virtuelle dans les centres de données cloud

Dans ce papier [38], les auteurs se sont intéressés au placement des machines virtuelles (VMs) dans les machines physiques (PMs) dans le but d'améliorer l'utilisation des ressources et réduire la consommation d'énergie.

Ils ont proposé un schéma de placement de VMs répondant à de multiples contraintes de ressources telles que la taille du serveur physique (CPU, RAM,..) ,la capacité des liens réseaux pour améliorer l'utilisation des ressources et réduire le nombre de serveurs physiques actifs entrainant ainsi la réduction de la consommation d'énergie.

Afin d'optimiser les ressources réseaux, les auteurs ont pensé à minimiser le trafic réseaux dans les centres de données en convergeant le trafic entre les VMs dans une même PM ou Switch. Pour cela, ils ont utilisé deux matrices $A$ et $B$ tels que :

$A$ est une matrice de trafic de communication entre les VMs.

$B$ est une matrice de coût de trafic de communication équivalent au nombre de Switch que traverse le trafic entre les VMs.

La fonction objective est formulée de la manière suivante :

$$\begin{aligned} \min \ & cost_{net} \sum_{i,j=1}^{N} a_{i,j} b_{m,p} \\ \text{s.t.} \ & \sum_{i=1}^{N} X_{i,m}.\vec{S_i} \leq Y_m.\vec{H_m}, \quad \forall m \\ & \sum_{j=1}^{N} X_{j,p}.\vec{S_p} \leq Y_p.\vec{H_p}, \quad \forall p \\ & \{X_{i,m}, X_{j,p} \in F\} \end{aligned} \tag{9}$$

Les paramètres sont cités dans la figure 3.1

| Symbol | Description |
|---|---|
| M | Number of PMs , indexed by $m = 1,..,M$ |
| $N$ | Number of VMs , indexed by $i = 1..N$ |
| $\vec{H}_m$ | $d$ dimensional resource vector of PM $m$ ,its value $\{H_{m,1}, H_{m,2}, \ldots, H_{m,d}\}$, $d$ is the number of resource types |
| $\vec{S}_i$ | $d$ dimensional resource vector of VM $i$ its value $\{S_{i,1},S_{i,2},\ldots,S_{i,d}\}$ |
| $Y_m$ | Binary variable , 1 indicates PM $m$ is in the activation status ; 0 indicates that PM $m$ is sleep |
| $X_{i,m}$ | Binary variable , 1 indicates VM $i$ is placed on the PM $m$ , whereas is 0 |
| A | Communication traffic matrix, $(a_{i,j})_{N\times N}$ is the traffic between VM $i$ and VM $j$ |
| B | Communication cost matrix B, The communication cost is equivalent to the number switch that the traffic between PMs traverse. |

FIG. 3.1 : Figure indiquant les symboles et leurs descriptions [38]

Pour les ressources serveurs, ils se sont focalisés sur la minimisation du nombre de PMs active.

La fonction objective est décrite dans l'équation suivante :

$$\begin{aligned} &\min \ cost_{ser} \sum_{m=1}^{M} \\ &\text{s.t.} \ \sum_{i=1}^{N} X_{i,m}.\vec{S_i} \leq Y_m.\vec{H_m}, \quad \forall m \\ &\{X_{i,m}, X_{j,p} \in F\} \end{aligned} \tag{10}$$

Pour l'optimisation de l'énergie des ressources physiques, ils ont utilisé un problème d'optimisation multi-objectif qui est décrit dans l'équation suivante :

$$\min cost_{net} + r.\min cost_{ser} \tag{11}$$

Dans cette solution, pour placer les VMs dans les PMs, les auteurs utilisent l'algorithme du Clusturing Hiérarchique suivi de l'algorithme Best Fit.

L'algorithme du Clusturing Hiérarchique consiste à réunir les VMs qui ont un grand trafic entre elles dans un même cluster pour assurer une meilleure performance des applications, et pour réduire le nombre d'éléments réseaux réduisant ainsi la consommation d'énergie.

Les VMs et leurs trafics sont présentés par un graphe unidirectionnel connecté $G = (V, E)$ tels que :

$V$ : Le nombre de VMs.

$E$ : Le trafic entre les VMs.

Le clustering hiérarchique est réalisé en utilisant le minimum de coupes (une coupe se compose de toutes les arêtes qui ont une extrémité dans $Q$ et l'autre extrémité dans $V \setminus Q$ , où $Q$ est un ensemble de nœuds avec $Q \neq \varnothing$ et $Q \neq V$. Cet algorithme prend en entrée le graphe G et fournit en sortie le résultat T(V) qui est un arbre binaire représentant la coupe minimal de G. L'algorithme Best Fit prend en entrée l'arbre binaire T(V) puis parcours cet arbre pour générer un vecteur appelé VMlist (se compose des feuilles successives de l'arbre T).

Ce vecteur est utilisé pour placer les VMs.

L'algorithme place les différentes VMs présentes dans le vecteur dans les PMs correspondantes, pour chaque nouvelle VM il fait la recherche à partir de la première PM pour trouver celle qui peut accommoder cette nouvelle VM. Si aucune n'est trouvée alors une nouvelle PM est allouée.

Nous avons considéré cette solution car elle permet de réduire le trafic ainsi que le nombre de PMs nécessaire pour satisfaire les besoins de placement des VMs réalisant ainsi une économie considérable d'énergie. Par contre si on place un grand nombre de VM dans la même PM alors les ressources physiques seront surchargées, et il y'a aussi le risque de congestion du réseau.

# IV. Problématique

En vue des limites et problèmes présentés dans la section précédente, nous allons présenter dans ce qui suit une solution pour essayer de résoudre la problématique étudiée qui est : Comment placer les pods dans les nœuds appropriés tout en assurant une consommation minime de l'énergie et de bande passante ?

# V. Description de la solution

Selon les travaux que nous avons étudié, nous avons décidé de combiner les deux travaux cités ci-dessus en faisant une hybridation de l'algorithme de colonie de fourmis du premier pour le placement des pods dans les nœuds appropriés avec l'algorithme du clustering hiérarchique du second qui permet d'obtenir de meilleurs résultats en termes de consommation et d'économie

énergie. De plus, nous allons utiliser l'API Pixie pour récupérer et gérer les besoins en bande passante des nœuds et des pods.
Notre solution se fera en trois phases :

- La première phase consiste à récupérer les valeurs de la bande passante des pods à travers l'API Pixie, et les fournir comme entrée à l'algorithme du clustering hiérarchique.
- La deuxième phase consistera à placer les pods de façon aléatoire dans les nœuds puis faire appel à l'algorithme du clustering hiérarchique pour regrouper les pods ayant un trafic réseaux important entre eux, et les réunir dans un même cluster pour optimiser la consommation d'énergie. Cette phase produira la liste des clusters (groupe de pods) qui sera fournie en entrée à l'algorithme ACO pour effectuer le placement de ces derniers.
- La troisième phase consistera à choisir le meilleur nœud pour placer le groupe de pods résultant de la phase précédente en utilisant l'algorithme de colonie de fourmis(ACO).

## 1. Phase1

La première phase consiste à récupérer les valeurs de la bande passante des pods dans notre cluster, pour cela nous utilisons l'API Pixie qui va nous servir à collecter les différentes données relatives au cluster et aux pods présents dedans.

## 2. Phase 2

Pour la deuxième phase, nous allons appliquer l'algorithme du clustering hiérarchique à l'ensemble des pods, en nous focalisant sur l'optimisation des ressources réseaux afin de réduire la consommation d'énergie. Après avoir récupérer les valeurs de la bande passante, l'objectif principal est de regrouper ensemble les pods ayant un trafic réseau important et ceci afin de minimiser le nombre de nœuds nécessaire pour le déploiement des conteneurs et réduire le trafic réseau.
Pour minimiser le trafic réseau entre les pods, nous utilisons deux matrices A et B :
$A = (a_{ij})_{N,N}$
$B = (b_{mp})_{M,M}$.
Plus le coût de communication est grand plus la consommation d'énergie du réseau est élevée.
La fonction objective est formulée avec équation 10.
Soit un graphe $G = (V, E)$ tels que :

$V$ : Une collection de pods.

$E$ : Trafic réseau entre ces pods.

L'algorithme de clustering hiérarchique est achevé en utilisant le minimum de coupe du graphe G, qui est décrit dans ce qui suit :

Soit un ensemble $Q$ inclus dans $V$, $\delta(Q)$ désigne l'ensemble de toutes les arêtes avec une extrémité en $Q$ et l'autre extrémité en $V \setminus Q$.

Une coupe désigne l'ensemble de toutes les arêtes avec une extrémité en $Q$ et l'autre extrémité en $V \setminus Q$.Où Q est un ensemble tels que $Q \neq \varnothing$ et $Q \neq V$, la coupe est dénotée $(Q, V \setminus Q)$.

On assigne à chaque arrête $ij \in E$ une capacité $c(ij)$ non négative, cette capacité est définie comme étant la somme de toutes les arrêtes contenues dedans c'est à dire :

$c(Q, V \setminus Q) = \sum_{ij \in \delta(Q)} c(ij)$.

Le problème de coupe minimum est de trouver une coupe en $G$ avec la plus petite capacité.

La coupe minimale de $G$ est exprimée avec un arbre binaire $T(V)$, où le sous arbre gauche $TL$ est dans $Q$, dont le poids est la somme des arrêtes qui sont dans $Q$, tels que :

$W(TL) = \sum_{ij \in \delta(Q)} c(ij)$, et le sous arbre droit $TR$ est dans $V \setminus Q$ dont le poids est calculé avec $W(TR) = \sum_{ij \in V \setminus Q} c(ij)$.

La feuille dans $T(V)$ représente un pod et la branche représente une collection de pods après le clustering. Le déroulement de l'algorithme est comme suit :

**Algorithm 1** Algorithme du clustering hiérarchique

**Entrée : Graphe** $G = (V, E)$
**Sortie : Arbre binaire** $T(V)$

Coupe initiale :$S$
Arbre initial : $T(V)$
**while** G contient des noeuds **do**
    Chosir 2 nouds distincts $s$ et $t$
    Calcul de la coupe $S2$ séparant $s$ et $t$
    **if** $c(S2, V \setminus S2) < C$ **then**
        $C \leftarrow c(S2, V \setminus S2)$
        $S \leftarrow S2$
    **end if**
    $TL \leftarrow G_s(V)$, Calcul $W(TL)$
    $TR \leftarrow G_t(V)$, Calcul $W(TR)$
    **if** $W(TL) < W(TR)$ **then**
        $W(TL) \leftrightarrow W(TR)$
        $G_s \leftrightarrow G_t$
    **end if**
    Remplacer $G$ par $G_s$ et $G_t$
**end while**
Retourner $T(V)$

## 3. Phase 3

La troisième phase de notre solution, consiste à placer les groupes de pods résultants de la deuxième étape dans les nœuds les plus appropriés. Ce placement est réalisé avec l'algorithme de colonie de fourmis. Pour se faire, l'algorithme reçoit en entrée le résultat de l'algorithme du clustering hiérarchique qui est une collection de groupes de pods ayant un trafic réseau important entre eux.

Les équations relatives au calcul de la probabilité et de la mise à jour de la concentration de phéromone restent inchangées comme mentionné précédemment.

Afin de réaliser le placement des pods dans les noeuds adéquats nous devons calculer les scores ($Score1$ et $Score2$) sur la base des resources utilisées par les groupes de pods qui sont le CPU et la RAM. Les valeurs de CPU, mémoire seront calculées avec les formules suivantes :

**Pour CPU :**

En premier lieu nous récupérons la quantité de cpu utilisée par chaque pod ($P_{cpuUsage}(i)$) appartenant au même cluster , puis nous calculons la quantité totale de cpu utilisée par tous les

pods avec la formule suivante :

$$T_{cpuUsage} = \sum_{i=1}^{N} P_{cpuUsage}(i) \tag{12}$$

tels que :

$Pod_{cpuUsage}(i)$ : La quantité de cpu utilisée par le $Pod(i)$.

$N$ : Le nombre de pods dans un cluster.

En deuxième lieu nous calculons le score du noeud par rapport à la quantité de cpu utilisée par l'ensemble des pod ($S_{cpu}$) qui sont dans le cluster avec la formule suivante :

$$S_{cpu} = \frac{T_{nodeCpu} - T_{cpuUsage}}{T_{nodeCpu}} \tag{13}$$

tels que :

$T_{nodeCpu}$ : La quantité de cpu d'un noeud.

**Pour RAM :**

En premier lieu nous récupérons la quantité de mémoire utilisée pour chaque pod ($P_{memUsage}(i)$) appartenant au même cluster, ensuite nous calculons la quantité totale de mémoire utilisée par l'ensemble des pods ($T_{memUsage}$) avec la formule suivante :

$$T_{memUsage}(i) = \sum_{i=1}^{N} P_{memUsage}(i) \tag{14}$$

tels que :

$P_{memUsage}(i)$ : La quantité de mémoire utilisée par le $Pod(i)$.

$N$ : Le nombre total des pods du cluster.

En deuxième lieu nous calculons le score du noeud par rapport à la quantité de mémoire utilisée par l'ensemble des pod ($S_{mem}$) qui sont dans le cluster avec la formule suivante :

$$S_{mem} = \frac{T_{nodeMem} - T_{memUsage}}{T_{nodeMem}} \tag{15}$$

tels que :

$T_{nodeMem}$ : La quantité de mémoire d'un noeud.

L'équation du $Score1$ représentant le ratio d'inactivité des ressources du nœud est calculé comme suit :

$$Score1 = \frac{S_{mem} + S_{cpu}}{2} \tag{16}$$

Plus le score du nœud est élevé et plus il est susceptible d'être choisi pour exécuter le groupe de pods.

Le $Score2$ représente le degré d'équilibrage des ressources du noeud, est calculé comme suit :

$$Score2 = \frac{(1 - abs(S_{cpu} - S_{ram})}{2} \tag{17}$$

Plus la valeur est grande, meilleur est l'équilibre des ressources du nœud.

Le déroulement de l'algorithme se fait comme suit :

1. Initialiser un nombre de fourmis ($nbFourmis$) à m et initialiser les valeurs des scores des nœuds présents dans la liste résultante de l'algorithme du clustering hiérarchique. Le nombre maximal d'itérations est $maxIter$ et le nombre d'itérations actuel est $iterI$. Initialiser aussi les valeurs du facteur heuristique des phéromones $\alpha$, le facteur de volatilisation $\rho$, Le facteur heuristique $\beta$ et les valeurs de la fonction objective.
2. Placer les fourmis aléatoirement dans les nœuds et traiter le $j^{ime}$ groupe de pods.
3. Chaque fourmi sélectionne le nœud approprié pour la prochaine tâche en se basant sur le score du nœud qui est calculé à l'aide de la formule (4), ainsi que la valeur de la fonction objective.
4. Après l'affectation de chaque tâche à un nœud avec la plus grande concentration de phéromone, la stagnation à lieu alors l'algorithme converge vers une solution locale.
5. Une fois que toutes les fourmis ont terminé toutes les allocations des pods, la mise à jour des phéromones est effectuée sur le schéma d'allocation global optimal par les formules (2) et (3).
6. Faire la comparaison entre le nombre actuel d'itérations et $maxIter$. Si le nombre actuel d'itérations est inférieur à $maxIter$, on va passer à l'étape 3, sinon on sort de la boucle, terminant ainsi l'algorithme.

**Algorithm 2** Algorithme de colonis de fourmi

Initialiser le nombre de fourmi à m (le nombre de groupes de pods)
Initialiser les valeurs du facteur heuristique des phéromones $\alpha$
Initialiser la valeur de facteur de volatilisation $\rho$
Initialiser le facteur heuristique $\beta$
Initialiser les valeurs de la fonction objective
**while** $nb_iter$ != $max_iter$ **do**
    Affecter aléatoirement chaque groupe de pods à un nœud
    **while** fourmis n'a pas encore construit la solution **do**
        **for** Chaque fourmis **do**
            Calculer score (équation 8) et valeur fonction objective.
            Sélectionner le nœud approprié
            Mise à jour local des phéromones
        **end for**
        Applique la règle de mise à jour globale
    **end while**
**end while**

# VI. Architecture de la solution

Voici la figure 3.2 que nous avons réalisé qui représente l'architecture globale de notre solution proposée.

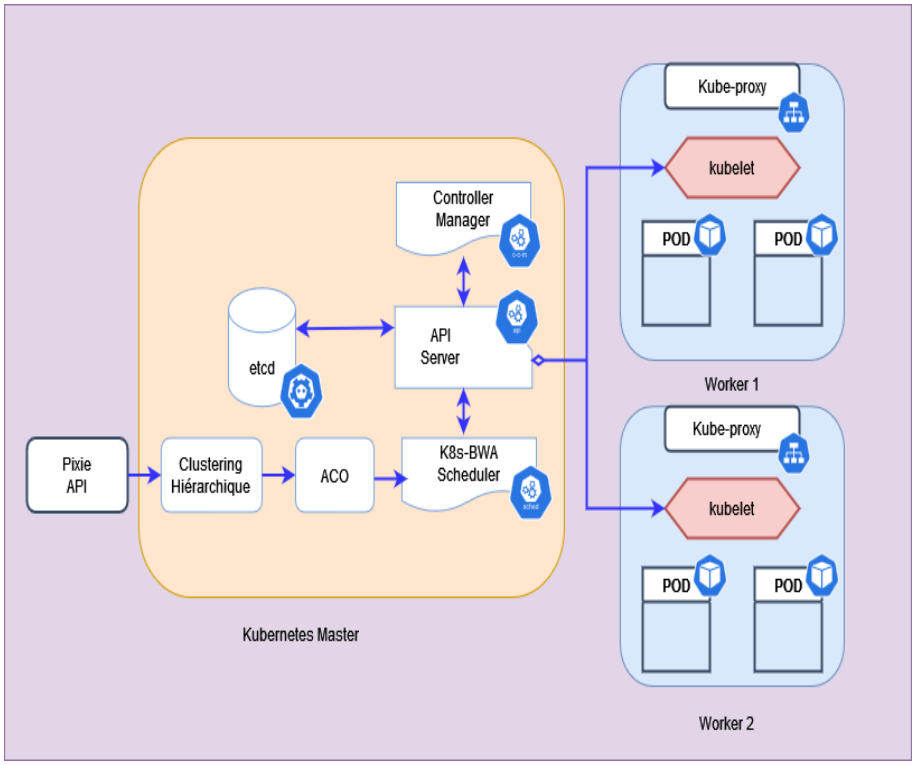


FIG. 3.2 : Architécture de la solution proposée

# VII. Conclusion

Dans ce chapitre, nous avons abordé la partie la plus importante de notre travail qui est la description et conception de la solution proposée. Cette solution a pour objectif la minimisation de la consommation de la bande passante et de l'énergie. Nous avons vu le processus de sa réalisation, ainsi que les différentes phases qui la compose et son architecture générale. Dans le prochain chapitre, nous parlerons en détails de l'environnement matériel utilisé pour l'implémentation, ainsi que les outils logiciels nécessaires à la réalisation de notre startégie.

# Chapitre 4

# Implémentation et tests

## Sommaire

# I. Introduction

Dans les chapitres précédents, nous avons abordé les divers concepts de notre domaine d'intérêt liés à l'ordonnancement des applications basées sur les conteneurs dans kubernetes. Ensuite, nous avons étudié certains travaux qui ont des éléments communs avec notre projet en termes de contexte et d'objectifs.

Nous nous sommes basé sur ces travaux afin de proposer une nouvelle stratégie d'ordonnancement pour kubernetes visant à réduire non seulement la consommation des ressources mais aussi celle de la bande passante, qui n'est pas prise en compte dans la stratégie d'ordonnancement par défaut, nous avons appelé notre proposition « K8s-BWA » (Kubernetes Bandwidth Aware Scheduler).

Dans ce chapitre, nous allons présenter l'implémentation de notre approche en commençant par la description de l'environnement physique de développement dans lequel nous avons implémenté notre stratégie, ainsi que les différents outils utilisés pour cette réalisation. À la fin de ce chapitre, nous exposerons les différents résultats et l'évaluation de notre stratégie dans le but de montrer la valeur ajoutée et le mérite de notre solution en faisant une comparaison avec la stratégie par défaut de kubernetes.

# II. Objectif

Le but principal de ce travail est de réaliser une stratégie d'ordonnancement pour kubernetes qui prend en compte non seulement les ressources (CPU, RAM) mais aussi la consommation de la bande passante. L'objectif étant de réduire la consommation d'énergie en minimisant la consommation de bande passante avec l'algorithme du clustering hiérarchique et en choisissant le meilleur emplacement des pods avec l'algorithme de colonie de fourmis (ACO). Pour cela, nous avons proposé une approche qui combine deux algorithmes (Algorithme de colonie de fourmis et l'algorithme de clustering hiérarchique).

# III. Environnement de travail

## 1. Environnement matériel

Pour notre environnement physique nous avons utilisé la plateforme CloudLab qui nous fournit les machines et les ressources nécessaire pour notre cluster.
Le tableau 4.1 suivant montre la fiche technique de notre environnement physique :

| Noms | Rôles | Addresses IP | Ressources |
|---|---|---|---|
| node0.k8s.labs1.pg0.utah.cloudlab.us | Nœud Master | 128.110.219.183 | **RAM :**131GB **CPU :**AMD EPYC 7402P 24 Coeurs **OS :**Ubuntu 20.04 LTS. |
| node1.k8s.labs1.pg0.utah.cloudlab.us | Nœud Worker | 128.110.219.159 | **RAM :**128GB **CPU :**AMD EPYC 7402P 24 Coeurs **OS :**Ubuntu 20.04 LTS. |
| node2.k8s.labs1.pg0.utah.cloudlab.us | Nœud Worker | 128.110.219.177 | **RAM :**128GB **CPU :**AMD EPYC 7402P 24 Coeurs **OS :**Ubuntu 20.04 LTS. |
| node3.k8s.labs1.pg0.utah.cloudlab.us | Nœud Worker | 128.110.219.176 | **RAM :**128GB **CPU :**AMD EPYC 7402P 24 Coeurs **OS :**Ubuntu 20.04 LTS. |
| node4.k8s.labs1.pg0.utah.cloudlab.us | Nœud Worker | 128.110.219.190 | **RAM :**128GB **CPU :**AMD EPYC 7402P 24 Coeurs **OS :**Ubuntu 20.04 LTS. |
| node5.k8s.labs1.pg0.utah.cloudlab.us | Nœud Worekr | 128.110.219.157 | **RAM :**128GB **CPU :**AMD EPYC 7402P 24 Coeurs **OS :**Ubuntu 20.04 LTS. |

Tab. 4.1 : Tableau présentatnt la fiche technique de l'environnement physique

## 2. Environnement logiciel

L'environnement logiciel comprend les composants suivants :

### 2.1. CloudLab

CloudLab est un banc d'essai, conçu pour permettre aux chercheurs d'expérimenter les architectures cloud et les nouvelles applications qu'elles permettent.
Cela signifie qu'il est parfaitement adapté aux expériences qui ne peuvent pas être exécutées dans des clouds traditionnels car elles nécessitent un contrôle et/ou une visibilité sur des parties du système qui seraient donnée dans d'autres clouds, telles que la virtualisation, le stockage ou les couches réseau.[39]

### 2.2. Ubuntu 20.04 LTS OS

Ubuntu est un système d'exploitation de bureau gratuit. Il est basé sur Linux, un projet massif qui permet à des millions de personnes dans le monde d'exécuter des machines alimentées par des logiciels libres et ouverts sur toutes sortes d'appareils. Linux se présente sous de nombreuses formes et tailles, Ubuntu étant l'itération la plus populaire sur les ordinateurs de bureau et les ordinateurs portables. Ubuntu est géré et financé par une société privée appelée Canonical Ltd.[40]
Pour notre travail, nous avons utilisé la version Ubuntu 20.04.
Ubuntu 20.04 utilise une version plus récente du noyau Linux (5.4) et Gnome (3.36) que Bionic Beaver. Il apporte également un nouveau thème sombre, des applications et des éléments visuels repensés, un nouveau mode de jeu et des performances de démarrage améliorées.[41]

### 2.3. Docker

Dans notre stratégie la version de Docker utilisée est 20.10.16, les informations sur ce dernier sont mentionnées plus haut.

### 2.4. Kubernetes

Dans notre stratégie nous avons utilisé la version 1.18.5 de kubernetes, les informations sur kubernetes sont mentionnées plus haut.

### 2.5. YAML

YAML est un langage de sérialisation de données basé sur Unicode, convivial et inter-langage, conçu autour des structures de données natives communes des langages de programmation agiles. Il est largement utile pour les besoins de programmation allant des fichiers de configuration à la messagerie Internet en passant par la persistance des objets et l'audit des données.[42]
L'utilisation de YAML pour les définitions K8s (Kubernetes) nous offre plusieurs avantages, notamment :

- YAML est facile à implémenter et à utiliser, donc il est facilement lisible par les humains.
- Il ne sera plus nécessaire d'ajouter tous les paramètres à la ligne de commande.
- Les fichiers YAML peuvent être ajoutés au contrôle des sources, ce qui permet le contrôle des changements, facilitant ainsi la maintenance du système.
- Possibilité de créer des structures beaucoup plus complexes en utilisant YAML que sur la ligne de commande.
- YAML dispose d'un modèle cohérent pour prendre en charge les outils génériques, ce qui permet la portabilité des données entre les langages de programmation. [43]

### 2.6. Pixie

Pixie est un outil de supervision open source pour les applications qui utilisent Kubernetes comme orchestrateur de conteneurs. Il capture automatiquement les données de télémétrie sans avoir besoin d'instrumentation manuelle. Pixie permet aussi la collecte, le stockage et l'interrogation de toutes les données de télémétrie localement dans le cluster. Il est composé de :

- **Pixie Edge Module (PEM) :** Agent pixie installé sur chaque nœud dans le cluster, utilisé pour la collecte des données stockées localement dans le nœud.
- **Vizier :** Collecteur pixie installé sur chaque cluster, responsable de l'exécution des requêtes et la gestion des PEMs.
- **Pixie Cloud :** Utilisé pour la gestion des utilisateurs ainsi que l'authentification.
- **Pixie CLI :** Utilisé pour déployer Pixie, il peut être aussi utilisé pour l'exécution des requêtes ainsi que la gestion des ressources.
- **Pixie Client API :** Utilisé pour interagir avec Pixie. [44]

## 3. Langage de programmation

### 3.1. Python

Pour implémenter notre stratégie, nous avons utilisé python comme langage de programmation afin d'intéragir avec l'API de l'orchestrateur des conteneurs Kubernetes, ce qui requiert l'utilisation d'une bibliothèque client qui s'appelle **client-python**. Python est un langage de programmation interprété, orienté objet et de haut niveau avec une sémantique dynamique. Ses structures de données intégrées de haut niveau, combinées au typage dynamique et à la liaison dynamique, le rendent très attrayant pour le développement rapide d'applications. La syntaxe simple et facile à apprendre de Python met l'accent sur la lisibilité et réduit donc le coût de maintenance du programme. Python prend en charge les modules et les packages, ce qui encourage la modularité du programme et la réutilisation du code. L'interpréteur Python et la vaste bibliothèque standard sont disponibles gratuitement sous forme source ou binaire pour toutes les principales plates-formes et peuvent être librement distribués.[45]
Dans l'implémentation de notre stratégie nous avons utilisé la version 3.8.10.

### 3.2. Bibliothèques clientes

Les bibliothèques clientes gèrent souvent des tâches courantes telles que l'authentification. La plupart des bibliothèques clientes peuvent découvrir et utiliser le compte de service Kubernetes pour s'authentifier si le client API s'exécute dans le cluster Kubernetes, ou peuvent comprendre le format de fichier kubeconfig pour lire les informations d'identification et l'adresse du serveur API.

### 3.3. Bibliothèques clientes officielles supportées par Kubernetes

Il existe de multiples façons de personnaliser et d'étendre Kubernetes :
Comme l'utilisation des fichiers de configuration, l'intérêt d'étendre kubernetes, c'est la possibilité d'accéder à des bibliothèques clientes pour l'utilisation de son API à partir de divers langages de programmation tels qu'il est mentionné dans la figure 4.1.

| Language | Client Library | Sample Programs |
|---|---|---|
| Go | github.com/kubernetes/client-go/ | browse |
| Python | github.com/kubernetes-incubator/client-python/ | browse |

FIG. 4.1 : Bibliothèques clients officielles supportées par Kubernetes
[46]

# IV. Implémentation

## 1. Implémentation de la stratégie proposée

### 1.1. Phase 1 : Récupération des valeurs de la bande passante des pods

Avant de pouvoir implémenter notre stratégie, nous devons d'abord installer l'outil Pixie dans notre cluster kubernetes afin de récupérer les valeurs de la bande passante des pods, pour cela nous utilisons le script de l'installation de cet outil qui peut être trouvé sur le site officiel `https://docs.px.dev/installing-pixie/install-guides/community-cloud-for-pixie`. Ensuite, nous utilisons le script **net_flow_graph** de Pixie qui affiche la communication entre les pods d'un même espace de nom en spécifiant l'espace de nom utilisé. Le script fournit, en premier lieu un graphe qui montre la communication entre les différents pod qui appartiennent au même espace de nom.

Voici la figure 4.2 qui montre la communication entre les pods de l'espace de nom px-sock-shop.

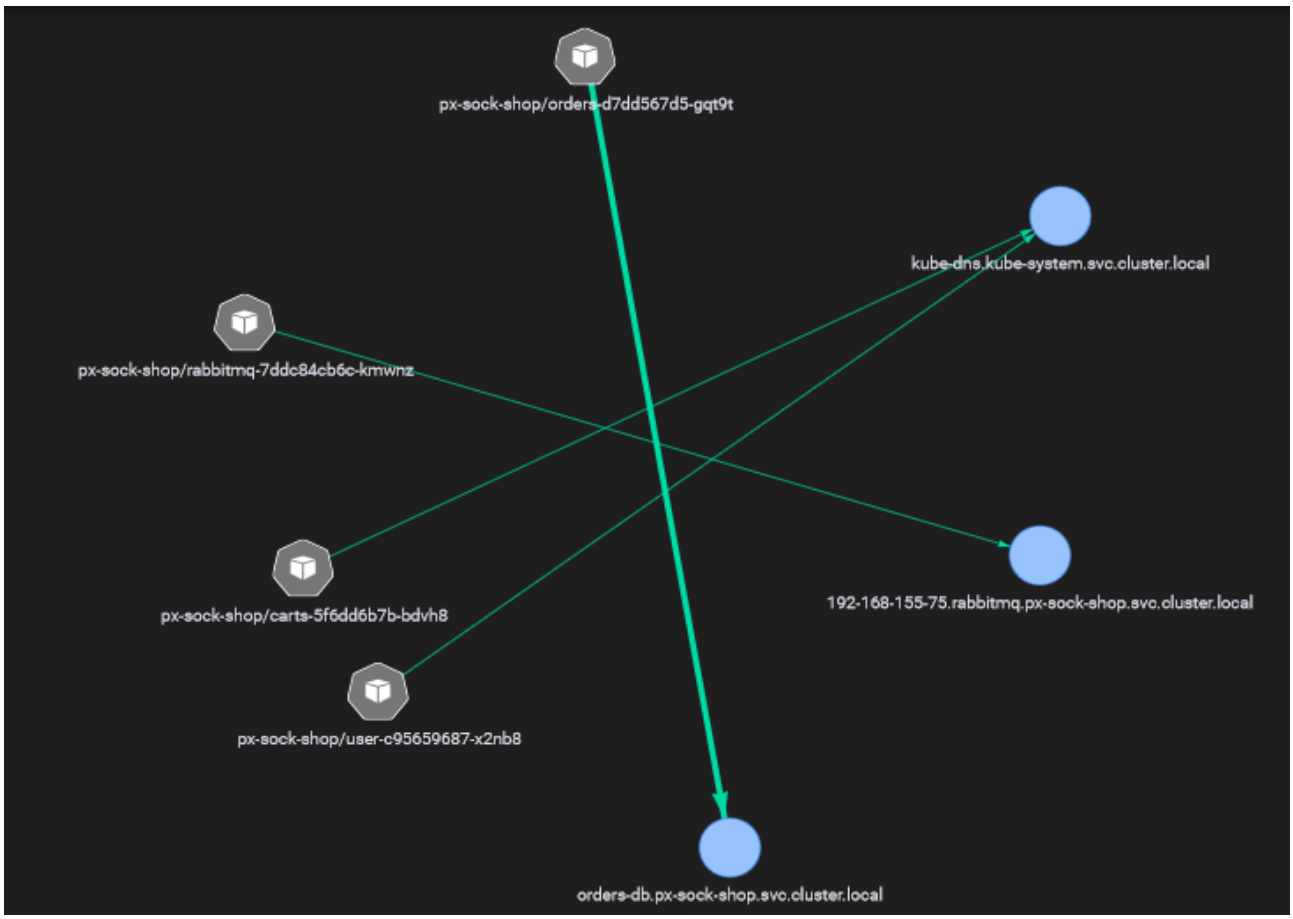


FIG. 4.2 : Graphe qui montre la communication entre les pods de l'espace de nom px-sock-shop

En deuxième lieu, un tableau composé de cinq colonnes qui sont la source, destination, nombre d'octets envoyés, nombre d'octets reçus, et le nombre d'octets totals, comme le montre figure 4.3 suivante :

| FROM_ENTITY | TO_ENTITY | BYTES_SENT | BYTES_RECV | BYTES_TOTAL |
|---|---|---|---|---|
| px-sock-shop/carts-5f6dd6b7b... | kube-dns.kube-system.svc.clus... | 0.2 B/s | 0.4 B/s | 0.6 B/s |
| px-sock-shop/orders-d7dd567... | orders-db.px-sock-shop.svc.clu... | 33.1 B/s | 1.1 KB/s | 1.1 KB/s |
| px-sock-shop/rabbitmq-7ddc8... | 192-168-155-75.rabbitmq.px-s... | 0.1 B/s | 0.3 B/s | 0.4 B/s |
| px-sock-shop/user-c95659687-... | kube-dns.kube-system.svc.clus... | 3.5 B/s | 8.1 B/s | 11.6 B/s |

FIG. 4.3 : Tableau qui résume le nombre de paquets échangés entre les pods de l'espace de nom px-sock-shop

### 1.2. Phase 2 : Regroupement des pods dans des clusters avec l'algorithme de clustering hiérarchique

Après la récupération des valeurs de la bande passante, les pods ayant un trafic de communication important entre eux sont regroupés dans un même cluster, réduisant ainsi le nombre de cluster, ceci sera fait à travers l'algorithme de clustering hiérarchique qui fournit en résultat un graphe montrant les différents clusters et leur nombre, comme le montre figure 4.4 suivante :

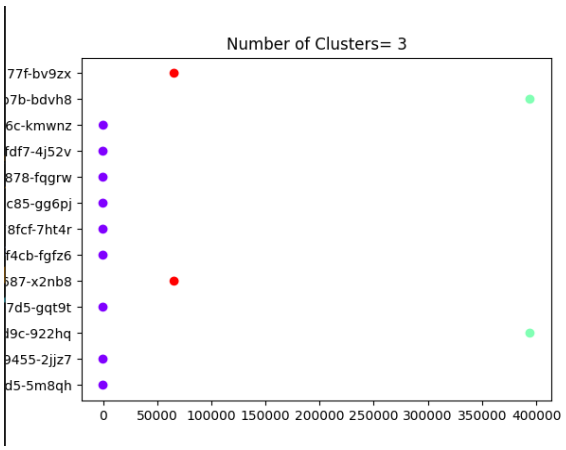


FIG. 4.4 : Regroupement des pods dans des clusters

### 1.3. Phase 3 : Placement des groupes de pods dans les nœud avec l'algorithme ACO

La liste résultante de l'algorithme de clustering hiérarchique contenant les identifiants des clusters avec les pods contenus dans ceux-ci et la liste des clusters ainsi que les scores des noeuds par rapport aux clusters sont donnés comme paramètres à l'algorithme de colonie de fourrmis pour faire leurs placement dans les noeuds adéquats en se basant sur les scores comme le montre la figure 4.5 et la figure 4.6 suivantes :

```
Clusters With Pods:
 {2: ['user-c95659687-x2nb8', 'user-db-79b99f877f-bw9zx'], 0: ['session-db-8f64655d5-5m8qh', 'orders-d7dd567d5-gqt9t', 'catalogue-db-5579f7f4
cb-fgfz6', 'front-end-6598d7fdf7-4j52v', 'catalogue-5b88655878-fqgrw', 'orders-db-77c46b9c85-gg6pj', 'queue-master-86dd578fcf-7ht4r', 'shippi
ng-5cfbb59455-2jjz7', 'rabbitmq-7ddc84cb6c-kmwnz'], 1: ['carts-5f6dd6b7b-bdvh8', 'carts-db-84b777d9c-922hq']}
```

FIG. 4.5 : Identifiants des clusters avec leurs pods

```
Clusters With Nodes Affectation:
 {2: 'node4.cls-k8s.labs1-pg0.utah.cloudlab.us', 0: 'node4.cls-k8s.labs1-pg0.utah.cloudlab.us', 1: 'node2.cls-k8s.labs1-pg0.utah.cloudlab.us'
}
```

FIG. 4.6 : Identifiants des clusters avec le meilleur placement

L'algorithme de colonie de fourmis nous donnera ainsi les meilleurs noeuds où seront placés chaque cluster.

Nous exécutons notre ordonnanceur comme un pod dans l'espace de nom kube-system, voici la figure 4.7 qui montre la configuration de notre ordonnanceur :

```
root@node0: ~
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    component: scheduler
    tier: control-plane
  name: k8s-bwa-scheduler
  namespace: kube-system
spec:
  selector:
    matchLabels:
      component: scheduler
      tier: control-plane
  replicas: 1
  template:
    metadata:
      labels:
        component: scheduler
        tier: control-plane
        version: second
    spec:
      serviceAccountName: k8s-bwa-scheduler
      containers:
      - image: sidali99/k8s-bwa-scheduler_i:latest
        name: k8s-bwa-scheduler
~
```

FIG. 4.7 : Fichier de configuration de l'ordonnanceur

Afin de tester notre ordonnanceur, nous deployons un nombre de pods qui seront exécutés dans l'espace de nom px-sock-shop, voici la figure 4.8 qui montre la configuration des pods.

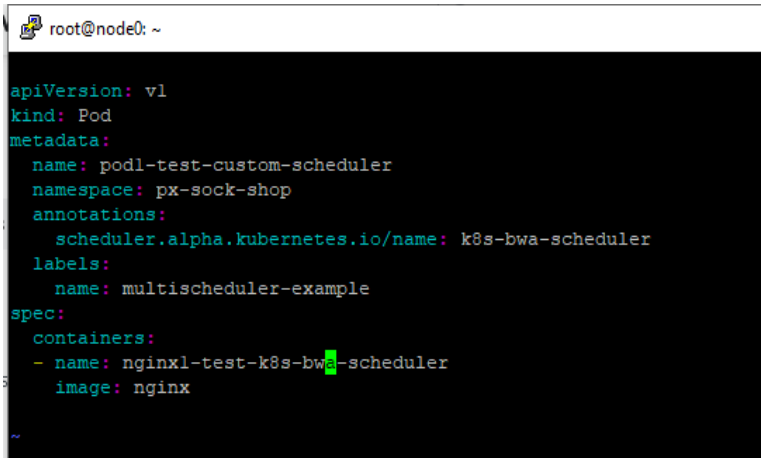


FIG. 4.8 : Fichier de configuration de pod

En dernière étape, nous vérifions que les pods déployés sont correctement exécutés, comme montre la figure 4.9 suivante :

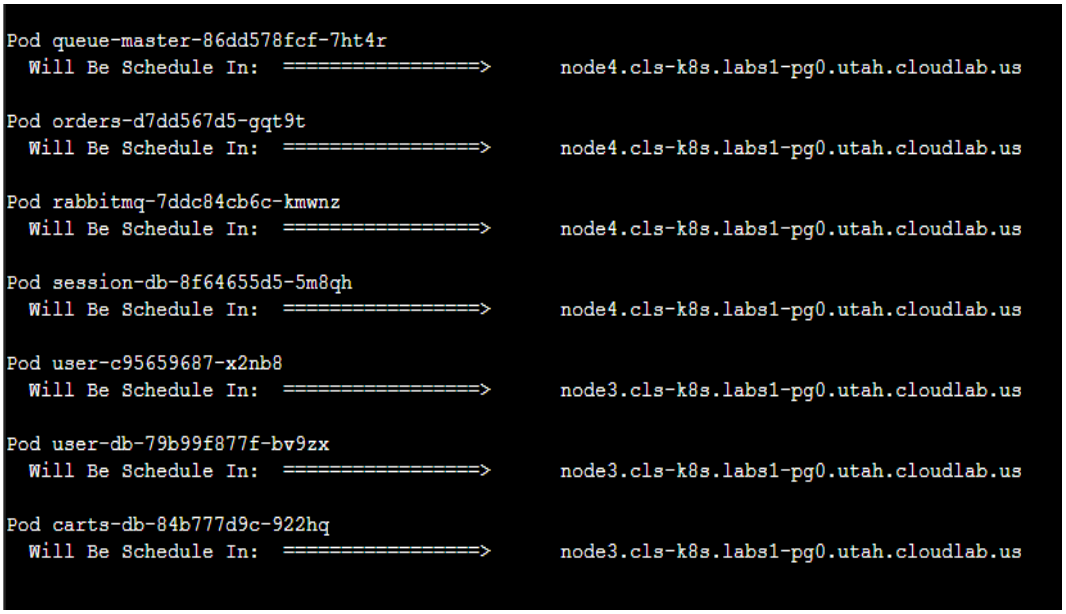


FIG. 4.9 : Exécution correcte des pods

## 2. Comparaison

Nous comparons notre stratégie avec la stratégie d'ordonnancement par défaut de kubernetes en deployant les pods de l'application px-sock-shop, voici les figures qui montrent le deploiement des pods avec la stratégie par défaut et avec notre stratégie.

```
root@node0:~# kubectl get pods -n px-sock-shop -o wide
NAME                           READY   STATUS    RESTARTS   AGE   IP              NODE
   READINESS GATES
carts-5f6dd6b7b-bdvh8          1/1     Running   1          29m   192.168.71.142   node1.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
carts-db-84b777d9c-922hq       1/1     Running   0          29m   192.168.151.79   node4.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
catalogue-5b88655878-fqgrw     1/1     Running   0          29m   192.168.98.10    node5.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
catalogue-db-5579f7f4cb-fgfz6  1/1     Running   0          29m   192.168.60.139   node2.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
front-end-6598d7fdf7-4j52v     1/1     Running   0          29m   192.168.71.141   node1.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
orders-d7dd567d5-gqt9t         1/1     Running   0          29m   192.168.98.11    node5.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
orders-db-77c46b9c85-gg6pj     1/1     Running   0          29m   192.168.60.140   node2.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
payment-7b674db4bc-7zlwz       1/1     Running   0          29m   192.168.155.74   node3.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
queue-master-86dd578fcf-7ht4r  1/1     Running   0          29m   192.168.71.143   node1.cls-k8s.labs1-pg0.utah.cloudlab.us
   <none>
rabbitmq-7ddc84cb6c-kmwnz      2/2     Running   0          29m   192.168.155.75   node3.cls-k8s.labs1-pg0.utah.cloudlab.us
```

FIG. 4.10 : Déploiement des pods avec la stratégie par défaut

```
Pod queue-master-86dd578fcf-7ht4r
  Will Be Schedule In:  ================>      node4.cls-k8s.labs1-pg0.utah.cloudlab.us

Pod orders-d7dd567d5-gqt9t
  Will Be Schedule In:  ================>      node4.cls-k8s.labs1-pg0.utah.cloudlab.us

Pod rabbitmq-7ddc84cb6c-kmwnz
  Will Be Schedule In:  ================>      node4.cls-k8s.labs1-pg0.utah.cloudlab.us

Pod session-db-8f64655d5-5m8qh
  Will Be Schedule In:  ================>      node4.cls-k8s.labs1-pg0.utah.cloudlab.us

Pod user-c95659687-x2nb8
  Will Be Schedule In:  ================>      node3.cls-k8s.labs1-pg0.utah.cloudlab.us

Pod user-db-79b99f877f-bv9zx
  Will Be Schedule In:  ================>      node3.cls-k8s.labs1-pg0.utah.cloudlab.us

Pod carts-db-84b777d9c-922hq
  Will Be Schedule In:  ================>      node3.cls-k8s.labs1-pg0.utah.cloudlab.us
```

FIG. 4.11 : Déploiement des pods avec notre stratégie

Comme le montre les captures d'écrans, notre stratégie solicite moins de noeud pour déployer les même pods faisant ainsi une économie d'énergie, de plus vu que les pods sont regroupés dans des clusters en fonction de leur communication, on économise ainsi de la bande passante.

# V. Conclusion

Dans ce dernier chapitre, nous avons abordé les différentes phases de l'implémentation de notre solution proposée, en commençant par la phase de récupération des valeurs de la bande passante des différents pods appartenant au même espace nom, ensuite nous utilisons ces valeurs pour initier la second phase qui consiste à regrouper les pods qui ont un trafic important entre eux dans un même cluster en utilisant l'algorithme de clustering hiérarchique, et en dernier nous procédons au placement des groupes de pods résultant dans les nœuds appropriés qui est fait à travers l'algorithme de colonie de fourmis.

Nous avons montré l'implémentation de notre stratégie dans un environnement physique ainsi que l'ensemble des outils utilisés pour réaliser cette tâche.

En dernier, nous avons illustré les résultats des expérimentations de notre ordonnanceur proposé, dans le but d'analyser son comportement. Et nous avons comparé notre ordonnanceur **K8s-BWA-Scheduler** avec la stratégie d'ordonnancement par défaut de kubernetes.

# Conclusion Générale

Dans le cadre de ce travail, nous avons parlé du cloud computing et des notions gravitant autour de ce domaine comme les conteneurs et la virtualisation. Les services offerts par la conteneurisation étant vastes et diversifés ont fait que l'usage d'ordonnanceur de conteneurs est devenu nécessaire et vital pour répondre aux exigences et offrir une qualité de service adéquate aux utilisateurs. Dans ce contexte, Kubernetes s'est vite fait une place et s'est imposé comme la référence de la gestion et de l'orchestration des conteneurs.

Kubernetes se concentre sur l'ordonnancement open source et modulaire, offrant une solution d'ordonnancement de conteneurs efficace pour les applications à forte demande avec une configuration complexe. L'ordonnancement est réalisé par le biais de son ordonnanceur. Ce dernier utilise une stratégie d'ordonnancement pour gérer les conteneurs de la meilleure façon possible. Mais sa stratégie ne parvient pas à satisfaire tous les besoins et ne s'adapte pas aux charges dynamiques. Tout ceci a entrainé l'émergence de plusieurs travaux et études qui proposent différentes nouvelles stratégies d'ordonnancements de conteneurs essayant de combler les lacunes de la stratégie par défaut de l'ordonnanceur de Kubernetes.

Nous avons utilisé ces travaux comme point de départ afin de concevoir notre propre stratégie d'ordonnancement visant à réduire la consommation d'énergie ainsi que la bande passante.

L'ordonnanceur proposé, nommé « K8s-BWA-Scheduler » ( **K**ubernetes **B**and**W**idth **A**ware **S**cheduler) est réalisé en combinant deux algorithmes qui sont l'algorithme de clustering hiérarchique et l'algorithme de colonie de fourmis , dans le but de réaliser notre objectif.

Nous avons montré à travers les expérimentations la valeur ajoutée de notre ordonnanceur en terme de consommation d'énergie et de bande passante.

Ce travail, nous a permis de découvrir l'environnement de « **Kubernetes** » et les différents méchanismes qui entrent en jeux dans le placement des pods dans les noeuds adéquats et d'acquérir de nouvelles connaissances dans le domaine de l'orchestration des conteneurs.

Comme perspectives, nous proposons l'amélioration de ce travail par :

- L'utilisation d'autres algorithmes autre que l'algorithme de colonie de fourmis tels que l'algorithme génétique pour le placements des groupes de pods dans les noeuds adéquats.
- Comparer notre solution avec d'autre stratégies existantes.
- Trouver un méchanisme pour réduire le nombre de groupes de pods exécutés sur un noeud pour éviter la surcharge de ce dernier ainsi que la congestion du réseaux.

# Bibliographie

## Documents Web et Articles

[1] Hurwitz, J. S., Bloor, R., Kaufman, M., & Halper, F. (2009). Cloud Computing For Dummies (1re éd.). For Dummies.

[2] Kayte, Sangramsing. (2015). Review and Classification of Cloud Computing Research. Journal of VLSI Signal Processing. 5. 5.

[3] Singh, Er.Gursimran & Bhathal, Gurjit. (2006). An Overview Of Virtualization. INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY. 5. 167-171. 10.24297/ijct.v5i3.3518.

[4] **gl-service. (s. d.). Qu'est-ce que la virtualisation ? - Définition de la virtualisation - Citrix France. Citrix.com.** à l'addresse `https://www.citrix.com/fr-fr/solutions/vdi-and-daas/what-is-virtualization.html`. Consulter le 6 décembre 2021

[5] Kaur, Kirandeep & Bhathal, Gurjit. (2013). Trend and Need of Application Virtualization in Cloud Computing. Global Journal of Computer Science and Technology Cloud and Distributed. Volume 13.

[6] **Qu'est-ce que la conteneurisation ? Quels sont les avantages ? (s. d.). Veritas.** à l'addresse `https://www.veritas.com/fr/ch/information-center/containerization`. Consulter le 6 décembre 2021

[7] Silva, Vitor & Kirikova, Marite & Alksnis, Gundars. (2018). Containers for Virtualization : An Overview. Applied Computer Systems. 23. 21-27. 10.2478/acss-2018-0003.

[8] **Hercé, L. (2020, 29 septembre). Conteneurisation informatique : définition, avantages, différence virtualisation, solutions. appvizer.fr.** à l'addresse `https://www.appvizer.fr/magazine/services-informatiques/virtualisation/conteneurisation-informatique`. Consulter le 6 décembre 2021

[9] Amaral, Marcelo et al. « Performance Evaluation of Microservices Architectures Using Containers ». 2015 IEEE 14th International Symposium on Network Computing and Applications, 2015. Crossref

[10] **What are Microservices ? (s. d.). smartbear.com.** à l'addrsse `https://smartbear.com/solutions/microservices/`. Consulter le 6 décembre 2021

[11] **GeeksforGeeks. (2020, 3 janvier). Difference between Virtual Machines and Containers.** à l'addresse `https://www.geeksforgeeks.org/difference-between-virtual-machines-and-containers`. Consulter le 6 décembre 2021

[12] **Doug Jones. (2018, 16 mars). Containers vs. Virtual Machines (VMs) : What's the Difference ? | NetApp Blog|Containers vs. Virtual Machines (VMs) : What's the Difference ? | NetApp Blog. NetApp.** à l'addresse `https://www.netapp.com/blog/containers-vs-vms/`. Consulter le 6 décembre 2021

[13] Bashari Rad, Babak & Bhatti, Harrison & Ahmadi, Mohammad. (2017). An Introduction to Docker and Analysis of its Performance. IJCSNS International Journal of Computer Science and Network Security. 173. 8.

[14] **Margot P. (2020, 23 décembre). Docker : qu'est-ce que c'est et comment l'utiliser ? Le guide complet. Formation Data Science | DataScientest.com.** à l'addresse `https://datascientest.com/docker-guide-complet/`. Consulter le 21 décembre 2021

[15] **Docker Architecture. (2021, 21 janvier). TutorialKart.** à l'addresse `https://www.tutorialkart.com/docker/docker-architecture/`. Consulter le 21 décembre 2021

[16] **Techopedia. (2017, 4 janvier). Scheduling. Techopedia.Com** à l'addresse `https://www.techopedia.com/definition/9654/scheduling` . Consulter le 17 janvier 2022

[17] **What is Container Orchestration ? Definition & Related FAQs. (2021, 11 janvier). Avi Networks.** à l'addresse `https://avinetworks.com/glossary/container-orchestration/`. Consulter le 21 décembre 2021

[18] **Sumo Logic, Inc. (s. d.). What is Docker Swarm ? Sumo Logic.** à l'addresse `https://www.sumologic.com/glossary/docker-swarm/`. Consulter le 21 décembre 2021

[19] **Allen, A. (2016, 5 juillet). Apache Mesos. SearchITOperations.** à l'addresse `https://searchitoperations.techtarget.com/definition/Apache-Mesos`. Consulter le 21 décembre 2021

[20] Santos, J., Wauters, T., Volckaert, B., & de Turck, F. (2021). Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing applications. Journal of Network and Computer Applications, 175, 102915.

[21] **Concepts. (s. d.). Kubernetes** à l'addresse `https://kubernetes.io/fr/docs/concepts/_print/#pg-4d68b0ccf9c683e6368ffdcc40c838d4`. Consulter le 21 décembre 2021

[22] **Service. (2022b, juin 22). Kubernetes** à l'addresse `https://kubernetes.io/docs/concepts/services-networking/service/`. Consulter le 21 décembre 2021

[23] **Volumes | Kubernetes Engine Documentation |. (s. d.). Google Cloud** à l'addresse `https://cloud.google.com/kubernetes-engine/docs/concepts/volumes`. Consulter le 21 décembre 2021

[24] **Concepts. (s. d.-b). Kubernetes.** à l'addresse `https://kubernetes.io/fr/docs/concepts/_print/#pg13b0f1dbe89228e3d76d2ac231e245f1`. Consulter le 21 décembre 2021

[25] I. Ahmad, Mohammad Gh. AlFailakawi, A. AlMutawa et al., Container scheduling techniques : A Survey and assessment, Journal of King Saud University –Computer and Information Sciences.

[26] **What is heuristic scheduling ? (2020, 16 juin). ASKINGLOT.** à l'addresse `https://askinglot.com/what-is-heuristic-scheduling` . Consulter le 17 janvier 2022

[27] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization. ACM Computing Surveys, 35(3), 268-308 https ://doi.org/10.1145/937503.937505

[28] Abraham Iorkaa, Asongo & Barma, Modu & Muazu, Hamandikko. (2021). Machine Learning Techniques, methods and Algorithms : Conceptual and Practical Insights. International Journal of Engineering Research and Applications. 11. 55-64. 10.9790/9622-1108025564.

[29] **What is Energy Efficiency ? (s. d.). About ENERGY STAR | ENERGY STAR.** à l'addresse `https://www.energystar.gov/about/about_energy_efficiency`. Consulter 17 janvier 2022

[30] **T. (s. d.). The 6 Most Important Resource Planning Metrics to Track. Ganttic.** à l'addresse `https://www.ganttic.com/blog/resource-planning-metrics`. Consulter 17 janvier 2022

[31] Wei-guo, Z., Xi-lin, M., & Jin-zhong, Z. (2018). Research on Kubernetes' Resource Scheduling Scheme. Proceedings of the 8th International Conference on Communication and Network Security - ICCNS 2018. https ://doi.org/10.1145/3290480.3290507

[32] C. I. Okonta1, A.H. Kemp, R.O. Edopkia, G.C. Monyei and E.D Okelue, A HEURISTIC BASED ANT COLONY OPTIMISATION ALGORITHM FOR ENERGY EFFICIENT SMART HOMES, University of Leeds, United Kingdom , University of Benin, Nigeria

[33] Selvi, V., & Umarani, D. (2010). Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques. International Journal of Computer Applications, 5(4), 1-6. https ://doi.org/10.5120/908-1286

[34] Dongshu Wang1, Dapei Tan1 and Lei Liu, Particle swarm optimization algorithm : an overview, School of Electrical Engineering, Zhengzhou University,Zhengzhou 450001, Henan, China

[35] Yuqi Fu, Shaolun Zhang, Jose Terrero, Ying Mao, Guangya Liu, Sheng Li, Dingwen Tao,Progress-based container scheduling for short-lived applications in a kubernetes cluster, Computer and Information Science, Fordham University

[36] Jingze Lv, Mingchang Wei,YangYu , A container scheduling strategy based on machine learning in microservice architecture, In : 2019 IEEE International Conference on Services Computing (SCC). IEEE, pp. 65–71.

[37] **What Is a Hypervisor and How Does It Work ? - Citrix France. (s. d.). Citrix.com** à l'addresse `https://www.citrix.com/fr-fr/solutions/vdi-and-daas/what-is-hypervisor.html`. Consulter le 6 décembre 2021

[38] J. Dong, X. Jin, H. Wang, Y. Li, P. Zhang and S. Cheng, "Energy-Saving Virtual Machine Placement in Cloud Data Centers," 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2013, pp. 618-624, doi : 10.1109/CCGrid.2013.107.

[39] **University of Utah. (s. d.). CloudLab.** à l'addresse `https://www.cloudlab.us/`. Consulter le 5 juin 2022

[40] **Bertel King, B. K. (2017, 9 octobre). Ubuntu : A Beginner's Guide. Makeuseo** à l'addresse `https://www.makeuseof.com/tag/ubuntu-an-absolute-beginners-guide/`. Consulter le 5 juin 2022

[41] **PCMag. (2021, 27 avril). Ubuntu 20.04 (Focal Fossa) Review** à l'addresse `https://www.pcmag.com/reviews/ubuntu-2004-focal-fossa`. Consulter le 4 juin 2022

[42] Oren. Ben-Kiki, Clark Evans, and Brian Ingerson. YAML Ain't Markup Language (YAML) Version 1.1. Working Draft, 2004-12-28

[43] Harichane Ishak, Université Oran 1, UNE MACHINE LEARNING POUR LA GESTION DES APPLICATIONS SUR DES RESSOURCES HETEROGENES CPU/GPU

[44] **About Pixie | Pixie Overview. (s. d.).** à l'addresse `https://docs.px.dev/about-pixie/what-is-pixie/`. Consulter le 4 juin 2022

[45] **What is Python ? Executive Summary. (s. d.). Python.Org** à l'addresse `https://www.python.org/doc/essays/blurb/`. Consulter le 4 juin 2022

[46] **Client Libraries. (s. d.). Kubernetes** à l'addresse `http://pwittrock.github.io/docs/reference/client-libraries/`. Consulter le 4 juin 2022