République Algérienne Démocratique et Populaire
الجمهوريـــة الجـزائريـة الديمـقراطيـة الشـعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التـعليـم العـالى و البـحث العلـمى

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
Ecole nationale Supérieure d'Informatique

2ème année cycle supérieure(2CS)

Option: Systèmes Informatiques (SQ)

Thème:

Mise en place et administration d'un cluster Kubernetes pour la Gestion des TP au niveau de l'ESI

Equipe N° 10:
- BELKESSA Linda (CE)
- DJABELKHIR Sarah
- CHELLAT Hatem
- BENHAMADI Yasmine
- BOUZOUAD Meriem
- LAMDANI Wilem
- Yasmine Benhamadi

Encadré par:
- Mr AMROUCHE Hakim
- Mr SEHAD Abdenour
- Mr HAMANI Nacer

Promotion : 2021-2022

Résumé

Pour développer et construire une infrastructure cloud moderne ou une implémentation DevOps, Docker et Kubernetes ont révolutionné l'ère du développement et des opérations logicielles. Bien que les deux soient différents, ils unifient le processus de développement et d'intégration, il est désormais possible de construire n'importe quelle architecture en utilisant ces technologies. Docker est utilisé pour créer, expédier et exécuter n'importe quelle application n'importe où tout en permettant d'utiliser les mêmes ressources disponibles. Ces conteneurs peuvent être utilisés pour accélérer les déploiements et réduire l'espace consommé, sont fiables et sont très rapides. Kubernetes quant à lui est une plateforme automatisée de gestion, de déploiement et de mise à l'échelle des conteneurs.

Ce projet a pour but de faciliter l'exploitation des différents logiciels utilisés dans le cadre académique durant les séances de TP. Cette première partie du travail élabore le résultat d'une étude approfondie des solutions existantes, ressources, besoins des utilisateurs (enseignants et étudiants) ainsi qu'une étude théorique de l'architecture et fonctionnement des outils que nous allons utiliser, une étude comparative accompagne cette partie pour justifier nos choix de technologies. A la fin de notre travail, nous avons opté pour la combinaison docker pour la conteneurisation et kubernetes pour l'orchestration. De ce fait, une réponse aux besoins et exigences des clients a été formalisée sous forme de solution que nous traiterons durant la prochaine phase du projet.

Mots Clés : containers, orchestration, Kubernetes, Docker

Table des matières

1 Introduction **5**

2 Présentation de l'entreprise **6**
2.1 Qui sommes-nous ? . . . 6
2.2 Solutions et prestation de services : . . . 6
2.3 Pourquoi nous-choisir ? . . . 6
2.4 Rôles des membres de l'équipe technique . . . 7
2.5 Organigramme . . . 10

3 Étude de l'existant **11**
3.1 Description du contexte . . . 11
3.2 Description du fonctionnement des salles machines au niveau de l'ESI . . . 11
3.3 Solutions existantes . . . 12
3.3.1 Technologie de virtualisation : Machine Virtuelle . . . 12
3.3.2 Externalisation de ressources avec le Cloud Computing : OpenStack . 13
3.4 Expression des besoins du client . . . 14
3.5 Problématique . . . 15
3.6 Solution proposée . . . 15

4 Étude théorique **16**
4.1 Architecture en microservices . . . 16
4.2 Conteneurisation . . . 16
4.2.1 Définition . . . 16
4.2.2 Principe de la conteneurisation . . . 17
4.2.3 Avantages et inconvénients . . . 18
4.2.4 Outils de conteneurisation . . . 19
4.3 Docker . . . 20
4.3.1 Définition . . . 20
4.3.2 Terminologie . . . 20
4.3.3 Architecture de la technologie . . . 21
4.3.4 Principe de fonctionnement . . . 22
4.4 Orchestration . . . 23
4.4.1 Définition . . . 23
4.4.2 Principe de fonctionnement . . . 23
4.4.3 Outils . . . 24

4.5 Kubernetes 25
4.5.1 Définition 25
4.5.2 Infrastructure Kubernetes 25
4.5.3 Principe de fonctionnement 27
4.5.4 Fonctionnalités 29
4.5.5 Avantages et inconvénients 30

5 Gestion du projet **31**
5.1 Processus de Flux de Travail 31
5.2 Planning et Etat d'avancement 32
5.2.1 Planning 32
5.2.2 Etat d'avancement 33

6 Conclusion **34**

Table des figures

2.1 Organigramme de l'entreprise 10

3.1 Modèle de la virtualisation par hyperviseur 12
3.2 Schéma explicatif de l'informatique en nuage 13

4.1 Principe de la conteneurisation 17
4.2 Architecture Docker 22
4.3 Conteneur basé sur l'image Ubuntu 15.04 22
4.4 Principe de fonctionnement de l'orchestration 24
4.5 Infrastructure Kubernetes 26
4.6 Architecture Kubernetes 26

5.1 Diagramme de processus de flux de travail 31
5.2 Diagramme de Gantt 32

Liste des tableaux

4.1 Etude comparative entre les outils de conteneurisation 20
4.2 Etude comparative entre les outils d'orchestration [4] 24

1. Introduction

Confrontées à l'augmentation continuelle des coûts de mise en place et de maintenance des systèmes informatiques accessibles depuis partout et à tout moment, les entreprises externalisent de plus en plus leurs services informatiques en les confiant à des entreprises spécialisées comme des fournisseurs de services nuagiques (cloud). L'intérêt principal de cette stratégie réside dans le fait que ces entreprises peuvent se prévaloir d'un excellent niveau de service, en ne payant que les ressources nécessaires et effectivement consommées. Quant au fournisseur du cloud, son but est de répondre aux besoins des clients en déployant le minimum de ressources nécessaires. Une approche courante du fournisseur consiste à mutualiser ses ressources (slicing) pour les partager entre plusieurs entreprises clientes. Dans ce contexte, plusieurs défis se dressent afin d'offrir un environnement cloud efficace. Nous pouvons citer parmi elles la garantie de bonnes performances, la gestion des ressources et la continuité de service.

De plus, gérer la performance de grandes applications reste compliqué, car scaler (s'adapter à un changement d'ordre de grandeur des requêtes) horizontalement (dupliquer une ou plusieurs instances du produit) est impossible puisque les composants sont fortement couplés et les applications ne sont pas indépendantes. Scaler verticalement en ajoutant du CPU ou autre sur un serveur ne nécessite pas de changement au niveau du code mais est très coûteux à l'inverse de l'horizontale. De ce fait, des technologies de conteneurisation et d'orchestration sont mises en place pour répondre à plusieurs problématiques et besoins des entreprises et particuliers.

2. Présentation de l'entreprise

2.1 Qui sommes-nous ?

Barbaros - Ingénierie des Systèmes Informatiques est une société de prestation de services informatiques, spécialisée dans les solutions à base de Linux, ingénierie système, réseau et Sécurité informatique. Son équipe technique est composée de jeunes ingénieurs algériens diplômés de l'École nationale Supérieure d'Informatique (ESI).

Nos ingénieurs et techniciens certifiés sont à votre service afin de vous apporter l'expertise dont votre entreprise a besoin pour évoluer et assurer sa place dans le marché.

Nos consultants financiers vous proposent des contrats récurrents, étalés dans le temps permettant un rapport qualité/prix optimum.

2.2 Solutions et prestation de services :

Nous vous proposons les services suivants :

- Services sécurité informatique du sol au plafond : Back-End et Front-End.
- Solutions d'architectures Clusters physiques et virtualisées : Haute disponibilité et Haute performance.
- Solutions de gestion et de supervision des DataCenters.
- Installation et mise en service de réseaux informatiques et téléphoniques.
- Formations professionnelles qualifiantes.
- Déploiement de l'infrastructure dans un Cloud privé de BarbarosTeam. Audit, diagnostic et préconisations.

2.3 Pourquoi nous-choisir ?

Pour la satisfaction des entreprises et leur permettre d'évoluer dans la compétitivité, Barbaros offre des lignes de service dans divers domaines pour des solutions adaptées à vos besoins.

Vous écouter, vous accompagner, vous assister, est notre priorité. Au travers de nos contrats de services et de nos offres de formation, notre engagement est de nourrir notre relation afin de vous faire vivre une expérience client extraordinaire.

Voici cinq principales raisons qui vous assurent d'avoir fait le bon choix en acceptant notre réponse d'appel d'offres :

- **Meilleur rapport qualité/prix :** Le coût de nos prestations est largement inférieur à ceux de la concurrence grâce à nos experts financiers qui vous proposent des contrats récurrents, étalés dans le temps permettant un rapport qualité/prix optimum.
- **Une Équipe compétente et qualifiée :** Composée d'ingénieurs issus des grandes écoles algériennes disposant d'expériences internationales.
- **Respect des engagements contractuels :** Nos équipes sont engagées à respecter à la lettre les consignes mentionnées dans les contrats signés avec le client.
- **Solutions efficaces et extensibles :** Nos ingénieurs vous proposent des solutions complètes, performantes et simples d'utilisation conçues pour répondre à vos besoins et satisfaire vos attentes.
- **Indépendance vis-à-vis des marques :** Nous œuvrons en continu pour vous fournir la meilleure solution informatique et répondre à tous vos besoins en toute objectivité.

2.4 Rôles des membres de l'équipe technique

1. **Chef de projet :**
   Capable de :
   - Dynamiser la coopération étendue.
   - Développer les outils d'assistance à l'anticipation.
   - Faciliter l'intégration et accompagner le changement.
   - Évaluer les changements, la performance, les risques et les opportunités.
   - Gérer les conflits, les crises.
   - Développer les compétences.
   - Créer un climat de confiance.
   - Travailler en équipe.

   Maîtriser :
   - Avoir un sens de communication avec ses membres d'équipe.
   - Organisation du projet.
   - Être un véritable leader.
   - Une grande organisation et une maîtrise de la planification.
   - Un sens de gestion du temps et des risques.
   - Savoir négocier.
   - Savoir prendre des décisions réfléchies.

2. **Responsable qualité :**
   Capable de :
   - Optimiser la performance qualité de l'entreprise.
   - Mettre en œuvre la prévention des risques.
   - Connaître les normes, standards et procédures QHSE (ISO, OHSAS, ERP/IGH...).
   - Manager l'activité du service QSE.
   - Communiquer et promouvoir un projet QSE.
   - Travailler en équipe.

   Maîtriser :
   - Les exigences en santé et en sécurité.
   - Les exigences environnementales.
   - Un excellent esprit d'analyse et de synthèse.

3. **Support technique :**
   Capable de :
   - Encadrer et animer une équipe de consultants techniques.
   - Gérer les relations avec les partenaires.
   - Garantir la fiabilité, la performance et l'évolution du système d'information.
   - Garantir l'intégration et l'encadrement technique des projets.
   - Conseiller et coordonner la production et le développement technique de l'entreprise.
   - Définir et suivre le budget.
   - Conseiller les équipes projets.

   Maîtriser :
   - Management, organisation, planification et contrôle des activités d'une équipe de consultants techniques.
   - Organisation et animation des réunions liées à l'évolution technique.
   - Supervision des projets techniques.
   - Contrôle des procédures et des documents de synthèse.
   - Garantie du respect des procédures et des méthodes d'assurance qualité.
   - Compétences techniques (système, bases de données, réseaux, sécurité, performance...).
   - Animation d'équipe.

4. **Designer et multimédia :**
   Capable de :

- Gestion des propositions des clients, de l'idée à la conception, l'impression et la production.
- Développer des concepts, des graphiques et des mises en page pour les illustrations de produits, les logos d'entreprise et les sites Web.
- Déterminer la taille et l'agencement du matériel de copie et d'illustration, ainsi que le style et la taille des polices.
- Préparation de brouillons de documents sur la base d'un mémoire convenu.
- Examiner les dispositions finales et suggérer des améliorations au besoin.

Maîtriser :

- Méthodologie de la création (créativité).
- Dessin – Rough.
- Écriture, scénarisation, dramaturgie.
- Capture Vidéo et gestion du Son.
- Découverte du Son Design.
- Animation layout posing et narration visuelle (storytelling).
- Design d'interfaces animées et ergonomies interactives.
- Animation typographique et de titrage.
- Principes universels d'animation.
- Colorimétrie et colorisation.

5. **Développeur Full-Stack :** Capable de :
   - Analyser les besoins du client.
   - Établir un cahier des charges.
   - Rédiger une spécification Technique de Besoin (STB).
   - Réaliser un prototype de la solution technique pour validation.
   - Concevoir et développer les programmes et applications informatiques.
   - Déterminer les phases et procédures de tests techniques et fonctionnels de programmes et applications informatiques.
   - Analyser des problèmes techniques.
   - Déterminer des mesures correctives.
   - Traiter l'information (collecter, classer et mettre à jour).

   Maîtriser :

   - Algorithmique et Architecture des systèmes d'information.
   - Technologies de l'accessibilité numérique.
   - Protocoles et règles de sécurité informatique et télécoms.
   - Langages de programmation informatique (Python, JavaScript, Java, C).

- Systèmes d'exploitation informatique.
- Architecture d'applications.

6. **Architecte Réseaux :** Capable de :
   - Analyser le réseau existant.
   - Élaborer et exécuter les configurations réseaux.
   - Superviser et gérer l'architecture réseau.
   - Faire le bilan des besoins techniques en matière de communication pour proposer des solutions adaptées.
   - Établir et maintenir une politique de sécurité.
   - Travailler en équipe et collaborer avec le directeur technique.
   - Tester la fiabilité du réseau mis en place.

   Maîtriser :

   - L'évolution des technologies propres aux réseaux.
   - Les normes utilisées dans les systèmes de communication.
   - Protocoles et règles de sécurité informatique.
   - Matériel technique utilisé (maîtrise parfaite du lexique informatique anglais).

2.5 Organigramme

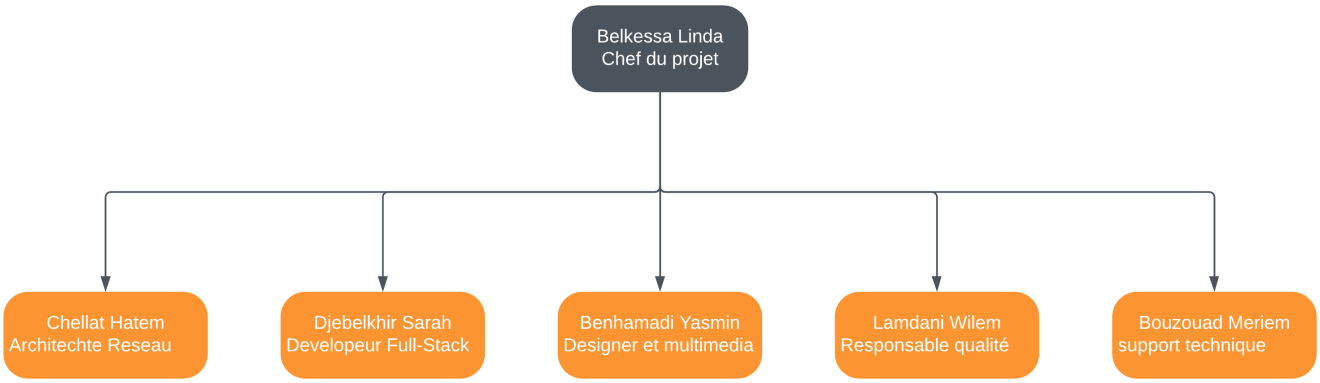

FIGURE 2.1 – Organigramme de l'entreprise

3. Étude de l'existant

3.1 Description du contexte

L'objectif principal de ce projet est de concevoir et mettre en place une solution qui facilitera la gestion des TPs au niveau de l'ESI d'une part, et qui d'autre part, réduira le temps nécessaire pour préparer les environnements d'exécution des applications par les enseignants responsables.

Dans cette phase préalable, une étude de l'existant sera effectuée afin de critiquer la procédure actuelle dans le but de donner un aperçu sur le degré de pertinence que le nouveau système se doit d'atteindre, pour ensuite proposer une solution tenant compte des différents besoins exprimés dans le cahier de charges.

3.2 Description du fonctionnement des salles machines au niveau de l'ESI

Compte tenu de la diversité des modules enseignés au sein de l'ESI, les séances de travaux pratiques nécessitent souvent l'utilisation des salles machines et comprennent l'utilisation de plusieurs applications sur les machines, qu'il s'agisse de celles des salles en y accédant à travers des terminaux ou les machines personnelles des étudiants (programmation, développement, base de données, conception objet...). Ces derniers sont souvent répartis sur des groupes comportant 12 à 15 binômes, chaque binôme travaillant sur la même instance.

Après discussion avec les personnes impliquées dans cette activité pédagogique, notamment les enseignants, il nous a été permis de déceler les différents problèmes rencontrés durant ces séances.

Parmi les problèmes les plus récurrents perturbant le bon déroulement des séances TPs :

- Lenteur de créations des VMs et d'installation des applications.
- Conflits de versions et compatibilité des applications entre les différents systèmes d'exploitation des machines.
- Lenteur des machines.
- Utilisation importante de ressources.

3.3 Solutions existantes

3.3.1 Technologie de virtualisation : Machine Virtuelle

Une salle virtuelle peut être considérée comme un ensemble de machines virtuelles, agencées en fonction d'objectifs pédagogiques. Le couplage de techniques de virtualisation d'une part, et le déport d'interface d'autre part, permet la mise en place de machines virtuelles à la disposition des étudiants.

Cette solution permet de faire fonctionner simultanément plusieurs machines virtuelles appelées "ordinateurs invités" sur une seule machine physique appelée "ordinateur hôte".

Chaque machine virtuelle invitée dispose de son propre système d'exploitation, ses propres applications et son propre environnement de connexion.

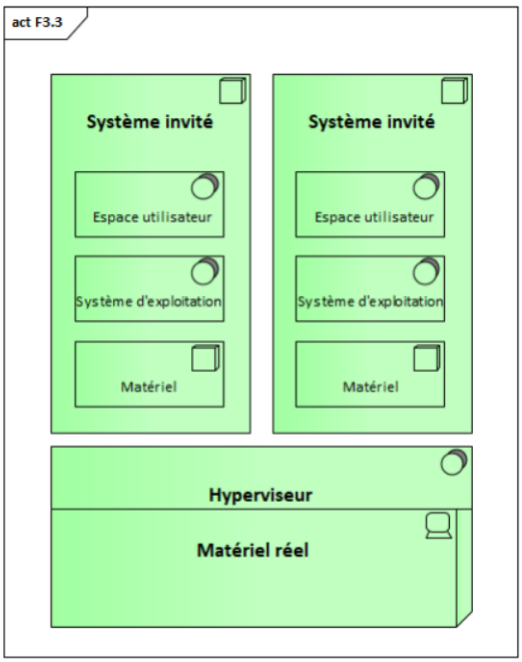

FIGURE 3.1 – Modèle de la virtualisation par hyperviseur [6]

Défauts de cette solution Bien que l'utilisation des machines virtuelles ait apporté plusieurs bénéfices aux enseignants lors des séances TPs comparé au fait de configurer des machines physiques, nous pouvons citer le fait de pouvoir héberger plusieurs systèmes différents sur la même machine, ce qui réduit les besoins du câblage, et permet de tirer un meilleur avantage des ressources physiques, cette technologie présente aussi des obstacles se rapprochant de la problématique de départ :

- La complexité de gérer les VMs
- Erreurs et dégradation des applications de la part des étudiants - aucune restriction quant à l'utilisation des VMs -

– Le nombre limité de VMs par hyperviseurs
– La consommation importante de ressources et dégradation de performance si le nombre de VMs est important - dû à la communication avec le matériel à travers l'hyperviseur.
– Les machines virtuelles invitées dépendent de la machine physique hôte.

3.3.2 Externalisation de ressources avec le Cloud Computing : OpenStack

C'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.

Les machines des salles TPs - et n'importe quel autre terminal - deviennent donc des points d'accès pour les étudiants leur permettant d'exécuter les applications et consulter les données hébergées sur les serveurs. Cette technologie offre plusieurs avantages entre autres la délocalisation du matériel et donc des applications, de plus les données conservées seront accessibles depuis n'importe où par les personnes ayant les permissions d'accès.

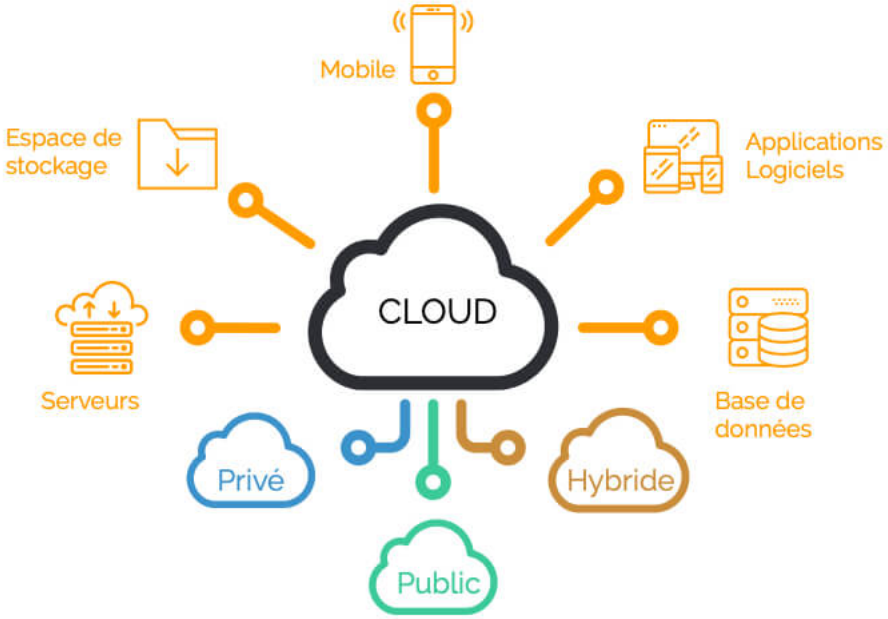

FIGURE 3.2 – Schéma explicatif de l'informatique en nuage [2]

Obstacles rencontrés avec cette solution : L'externalisation de l'infrastructure informatique requiert la signature d'un contrat avec un fournisseur de solutions de cloud computing (abonnement mensuel ou annuel), ce qui n'était pas possible à cause des coûts élevés et le manque de ressources nécessaires pour le réaliser au niveau de l'école. En outre, le risque de confidentialité entre en jeu vu que tout sera hébergé en dehors de l'organisation, dans le cloud, ou aucune sécurité est garantie.

Un autre problème qui soulève est la dépendance sur le fournisseur, surtout si les fonctionnalités dont l'école aura besoin risquent de changer à travers le temps ou deviennent très spécifiques. Cela demande le choix d'un fournisseur en qui l'on a confiance.

De plus, héberger sa propre infrastructure cloud privée au sein de l'organisation demande des besoins matériels assez importants que l'ESI ne pouvait se permettre.

3.4 Expression des besoins du client

Pour estimer la charge de travail et le temps nécessaire afin de délivrer les résultats attendus de la part du client, il est essentiel de procéder d'abord à une expression des besoins et des exigences qui ont été collectés à travers différents outils :

- Entrevues avec les enseignants encadreurs.
- Envoi de questionnaires.
- Observation des méthodes actuelles de travail.
- Organisation d'interviews avec les responsables des salles machines.

Besoins Fonctionnels : Les besoins fonctionnels d'un système s'expriment par une liste non exhaustive de ce que ce dernier doit réaliser.

Voici l'ensemble d'exigences fonctionnelles que le nouveau système mis en place se doit de satisfaire :

- Le système doit permettre l'accès simultané au cluster d'un groupe composé de 12 à 15 binômes.
- Le système doit donner la possibilité de modifier/planifier l'exécution des applications de la part des administrateurs.
- Le système doit permettre l'authentification des étudiants lors des tentatives de connexions.
- Le système doit permettre la traçabilité des connexions récentes.
- Le système doit garder l'état d'exécution de l'application pour chaque utilisateur.
- Le système doit permettre un accès GUI à travers une URL/adresse IP.

Besoins Techniques : Les besoins non fonctionnels d'un système représentent le comportement et la performance que le système doit avoir dans un environnement technique bien spécifié.

Voici l'ensemble d'exigences techniques que le nouveau système mis en place se doit de satisfaire :

- La solution proposée doit prévoir le minimum de ressources à utiliser.
- Le système doit avoir un niveau de sécurité acceptable.
- Le système doit être capable de gérer l'accès simultané de plusieurs étudiants (montée en charge) d'au minimum un groupe - 12 à 15 binômes- .

3.5 Problématique

Les machines virtuelles sont de plus en plus utilisées au niveau de l'ESI pour pallier aux problèmes liés à l'utilisation de machines physiques permettant l'utilisation d'applications durant les séances TPs. Cependant, d'autres difficultés ont été rencontrées en optant pour une virtualisation complète du système d'exploitation, notamment la complexité des machines virtuelles qui demandent une gestion pertinente, consommant beaucoup de temps aux administrateurs.

La technologie de conteneurisation permet d'avoir une virtualisation légère au niveau application afin d'éviter tout problème lié au système d'exploitation. Le principe de conteneuriser un logiciel consiste à rassembler son code et tous ses composants (bibliothèques, frameworks...) de manière à isoler son exécution dans son propre conteneur.

Néanmoins, la gestion des conteneurs, surtout quand le taux d'utilisation augmente, devient une mission défiante, à savoir le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs.

L'utilisation d'un outil qui permettra d'automatiser toutes ces tâches est un impératif. C'est pour cela qu'on nous a incombé de proposer une solution salvatrice à cette problématique en assurant deux objectifs principaux :

- Simplifier la gestion des applications conteneurisées.
- Centraliser l'emplacement et la gestion des applications.

3.6 Solution proposée

Avec les conteneurs, les applications basées sur des microservices disposent d'une unité de déploiement et d'un environnement d'exécution parfaitement adaptés. Les conteneurs permettent d'exécuter plusieurs parties d'une application dans des microservices, indépendamment les unes des autres, sur le même matériel, avec un niveau de contrôle bien plus élevé sur leurs éléments et cycles de vie.

La gestion du cycle de vie des conteneurs avec l'orchestration permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs.

Cette technologie est compatible avec tous les environnements exécutant des conteneurs. Elle permet de déployer la même application dans différents environnements sans modifier sa conception. De plus, les microservices stockés dans les conteneurs simplifient l'orchestration des services, notamment les services de stockage, de réseau et de sécurité.

Les outils d'orchestration de conteneurs fournissent un cadre pour la gestion de l'architecture de conteneurs. Beaucoup de solutions sont disponibles sur le marché pour aider à gérer le cycle de vie des conteneurs. Parmi les plus connues, on peut citer Kubernetes, Docker Swarm et Apache Mesos.

En réponse à l'appel d'offres lancé par l'ESI, notre équipe propose une solution Kubernetes personnalisée qui permettra de centraliser et d'automatiser l'administration d'un cluster des conteneurs. C'est une solution se montrera extensible selon les besoins changeants dans le futur et les procédures correctives pouvant avoir lieu sans impacter son efficacité.

4. Étude théorique

4.1 Architecture en microservices

Le but des architectures est de répondre à un besoin en suivant la logique du métier, souvent les développeurs ont tendance à utiliser une architecture Monolithique où toute l'application développée est déployée comme une seule unité .

L'architecture en microservices consiste à découper une application comme un ensemble de services simples et indépendants les uns des autres ou bien faiblement couplés qui se caractérisent par la facilité de maintenance, de test et d'autonomie, chaque service exécute une seule tâche spécifique. Ainsi les microservices peuvent utiliser des langages de programmation différents, assurant l'adaptabilité.

Ces services communiquent entre eux en échangeant des requêtes et des réponses notamment à travers des API REST en se basant sur le protocole HTTP. Cette architecture offre une très grande scalabilité et s'adapte ainsi au nombre croissant d'utilisateurs, elle se caractérise par :

- Une facilité de maintenance, étant donné que chaque service est indépendant et lié à une tâche spécifique.
- Une grande tolérance aux pannes car en cas d'arrêt d'un certain service, les fonctionnalités fournies par les autres services continuent toujours de s'exécuter
- La réutilisabilité, car les services offrent des interfaces (api) aux autres services ou même aux applications,
- Une facilitation de l'intégration et même du déploiement des applications.

4.2 Conteneurisation

4.2.1 Définition

Il s'agit d'un type de virtualisation utilisé au niveau des applications. Elle consiste à rassembler le code du logiciel et tous ses composants (bibliothèques, Framework et autres dépendances) de manière à les isoler dans leur propre « conteneur ». Le logiciel ou l'application dans le conteneur peut ainsi être déplacé et exécuté de façon cohérente dans tous les environnements et sur toutes les infrastructures, indépendamment de leur système d'exploitation.

Le conteneur fonctionne comme une sorte de bulle, soit un environnement de calcul qui enveloppe l'application et l'isole de son entourage. C'est en fait un environnement de calcul portable complet.

4.2.2 Principe de la conteneurisation

Le principe de la conteneurisation repose sur la création de plusieurs espaces utilisateurs isolés les uns des autres sur un noyau commun. Elle permet aux développeurs de créer et de déployer des applications plus rapidement et de manière plus sécurisée. Avec les méthodes traditionnelles, le code est développé dans un environnement informatique spécifique. Lorsqu'il est transféré vers un nouvel emplacement, cet environnement entraîne souvent des bogues et des erreurs.

Par exemple, lorsqu'un développeur transfère du code d'un ordinateur de bureau vers une machine virtuelle (VM) ou d'un système d'exploitation Linux vers un système d'exploitation Windows.

La conteneurisation élimine ce problème en regroupant le code de l'application avec les fichiers de configuration, les bibliothèques et les dépendances associés nécessaires à son exécution. Ce package unique de logiciel ou "conteneur" est extrait du système d'exploitation hébergeur.

Par conséquent, il est autonome et devient portable – capable de fonctionner sur n'importe quelle plateforme ou cloud, sans problèmes

FIGURE 4.1 – Principe de la conteneurisation [7]

4.2.3 Avantages et inconvénients

Avantages :

- **Portabilité :** un conteneur créé un package exécutable de logiciels extrait du système d'exploitation hébergeur (non lié ou dépendant de celui-ci). Par conséquent, ce package est portable et capable de fonctionner de manière uniforme et cohérente sur n'importe quelle plateforme ou cloud.
- **Agilité :** Le Docker Engine open source qui exécute les conteneurs a lancé la norme de l'industrie pour les conteneurs avec des outils de développement simples et une approche de packaging universelle qui fonctionne à la fois sur les systèmes d'exploitation Linux et Windows.
- **Vitesse :** les conteneurs sont souvent qualifiés de « légers ». Cela signifie qu'ils partagent le noyau du système d'exploitation (OS) de la machine et ne sont pas embourbés par ces frais supplémentaires. Non seulement cela améliore l'efficacité des serveurs, mais cela réduit également les coûts de matériel et de licence. De plus, les temps de démarrage sont accélérés étant donné qu'il n'y a pas de système d'exploitation à lancer.
- **Isolation des erreurs :** chaque application conteneurisée est isolée et fonctionne indépendamment des autres. La défaillance d'un conteneur n'affecte pas le fonctionnement continu des autres conteneurs. Les équipes de développement peuvent identifier et corriger tout problème technique dans un conteneur sans aucun temps d'arrêt de ceux en cours de fonctionnement. En outre, le moteur de conteneur peut tirer partie de toutes les techniques d'isolation et de sécurité du système d'exploitation, telles que le contrôle d'accès SELinux, pour isoler les erreurs dans les conteneurs.
- **Efficacité :** les logiciels exécutés dans des environnements conteneurisés partagent le noyau du système d'exploitation de la machine. De plus, les couches d'application d'un conteneur peuvent être partagées avec les autres. Ainsi, les conteneurs ont une capacité intrinsèquement plus petite qu'une machine virtuelle et nécessitent moins de temps de démarrage. Cela permet à beaucoup plus de conteneurs de s'exécuter avec la même capacité de calcul d'une seule machine virtuelle. Ce qui améliore l'efficacité des serveurs, réduisant les coûts de matériel et de licence.
- **Facilité de gestion :** une plateforme d'orchestration de conteneurs automatise l'installation, la mise à l'échelle et la gestion des charges de travail et des services conteneurisés. Les plateformes d'orchestration de conteneurs peuvent faciliter les tâches de gestion telles que la mise à l'échelle des applications conteneurisées, le déploiement de nouvelles versions d'applications et la surveillance, la journalisation et le débogage, entre autres fonctions. Kubernetes est une technologie open source (à l'origine open-source par Google, basée sur leur projet interne appelé Borg). Il se classe parmi les systèmes d'orchestration de conteneurs les plus populaires disponibles. A l'origine, il automatise les fonctions des conteneurs Linux. Kubernetes fonctionne avec de nombreux moteurs de conteneur, tels que Docker, mais aussi avec tout système de conteneur conforme aux normes Open Container Initiative (OCI) pour les formats d'image de conteneur et les environnements d'exécution.
- **Sécurité :** l'isolement des applications en tant que conteneurs empêche par nature l'invasion de codes malveillants les empêchant d'affecter d'autres conteneurs ou le système

hôte. De plus, des autorisations de sécurité peuvent être définies pour empêcher automatiquement les composants indésirables d'entrer dans des conteneurs ou limiter les communications avec des ressources inutiles.

Inconvénients

- **Incompatibilité avec certaines tâches :** Thomas Bittman fait remarquer que les conteneurs, bien que polyvalents, sont loin de pouvoir remplacer tous les déploiements de machines virtuelles (VM) existants. En effet, tout comme d'anciennes applications se prêtaient mieux à des déploiements physiques aux premiers temps de la virtualisation, certaines applications ne conviennent pas du tout à une virtualisation en conteneurs.
- **Problème des dépendances :** Les VM classiques sont extrêmement autonomes, chacune comprenant un système d'exploitation (OS) à part entière, des pilotes et des composants d'application. Elles peuvent également migrer vers n'importe quel autre système du moment qu'un hyperviseur approprié est disponible. De leur côté, les conteneurs s'exécutent sur un OS et partagent la majeure partie du noyau sous-jacent ainsi qu'un grand nombre de fichiers binaires et de bibliothèques. D'après Bittman, les dépendances imposées aux conteneurs peuvent limiter la portabilité entre serveurs.
- **Faiblesse relative de l'isolement :** Les VM reposant sur un hyperviseur offrent un degré élevé d'isolement les unes des autres, car les ressources matérielles du système sont toutes virtualisées et présentées aux VM par le biais de l'hyperviseur. Autrement dit, un bug, un virus ou une intrusion peut porter atteinte à une VM sans se propager aux autres.
- **Risque de prolifération :** Si la gestion du cycle de vie des VM est importante dans les environnements basés sur un hyperviseur, elle s'avère absolument essentielle pour les conteneurs. En effet, ces derniers offrent l'avantage non négligeable de pouvoir être mis en service et dupliqués à la vitesse de l'éclair. Le revers de la médaille est qu'il est également possible de consommer une grande quantité de ressources informatiques sans vraiment s'en rendre compte. Ce n'est pas très grave si les conteneurs qui composent l'application sont arrêtés ou supprimés dès lors qu'ils ne sont plus nécessaires. Mais en cas d'oubli, la montée en charge d'une application conteneurisée peut se traduire par des coûts de Cloud Computing tout aussi importants qu'inutiles pour l'entreprise.

4.2.4 Outils de conteneurisation

Avec les nouvelles tendances de la technologie, l'utilisation des conteneurs est présente presque partout dans le monde .

Cette technologie de conteneurisation a été élaborée grâce à des projets open source tels que LXC et son évolution en LXD, DOCKER, RKT de CoreOs.

Dans le tableau ci-dessous nous allons comparer entre les technologies les plus utilisées pour la conteneurisation docker et LXD et rkt qui sont les trois open source.

LXD	Docker	Rkt
- Les conteneurs sont vus comme des machines virtuelles mais sans système d'exploitation. - Ne supporte que les distributions linux - Peut être integré en cloud avec OpenStack ou OpenNebula - Exécute plusieurs processus sur le même conteneur - N'est pas portable	- Les conteneurs sont vus comme des machines virtuelles mais sans système d'exploitation. - Ne supporte que les distributions linux - Peut être integré en cloud avec OpenStack ou OpenNebula - Exécute plusieurs processus sur le même conteneur - N'est pas portable	- virtualisation des applications. - Est utilisé sous linux uniquement - Compatible avec les orchestrateurs tels que Kubernetes et Nomad - Est portable - Supporte les images docker - Est plus sécurisé

TABLE 4.1 – Etude comparative entre les outils de conteneurisation

4.3 Docker

4.3.1 Définition

Docker est une plateforme logicielle Open source qui permet de concevoir, tester et déployer des applications conteneurisées. Ce dernier regroupe le code et les dépendances d'une application comme un environnement de calcul isolé indépendamment du système d'exploitation et de l'infrastructure utilisés.

4.3.2 Terminologie

image docker : Est un fichier utilisé pour exécuter le code dans un conteneur, ce dernier regroupe toutes les informations et les dépendances, ainsi que le code source et les configurations de déploiement et d'exécution à utiliser.

Fichier dockerfile : Fichier texte contenant toutes les instructions pour la création d'images docker.

Création (build) : Processus de création d'une image à partir du fichier Dockerfile, se lance avec la commande docker build.

Registre (registry) : Permet le stockage et la gestion des images docker. Il est comparable à un répertoire où l'on peut extraire ou héberger les images . Le registre par défaut utilisé

pour la plupart des images publiques est Docker Hub et Docker Cloud mais les développeurs peuvent avoir leur propre registre privé.

Conteneur Docker : Est une instance de l'image docker. Un conteneur fonctionne avec une image docker, un environnement d'exécution et un ensemble de commandes. Un conteneur est lancé très rapidement et consomme peu de ressources, ce dernier est éphémère ce qui veut dire qu'il peut être détruit et reconstruit autant de fois que l'on souhaite.

Volumes : Permettent d'avoir un système de fichiers accessible en écriture que le conteneur peut utiliser. Ces fichiers sont stockés au niveau du système hôte et contiennent des données persistantes ou des données partagées.

Docker Compose : Est un outil de commande fournissant des métadonnées pour la définition et l'exécution d'applications multi-conteneurs. Il se base sur un fichier YAML pour la configuration et assure la création d'un conteneur par image en utilisant la commande :

```
docker-compose up
```

4.3.3 Architecture de la technologie

Docker utilise une architecture client serveur avec trois composants principaux : le client, le docker daemon [1] (serveur) et l'API rest et utilise le registre docker ainsi que les composants de réseaux et les volumes de stockages.

Client Docker : Communément appelé Docker, ce dernier joue le rôle client, il interagit avec le docker daemon à travers l'invité de commandes (CLI).

Les commandes les plus utilisées sont :

```
$ docker run: Pour instancier un nouveau conteneur
$ docker pull: Pour récupérer une image du registre
$ docker build: Pour construire une nouvelle image
```

Un Docker daemon appelé dockerd joue le rôle de serveur. Il se charge d'exécuter les requêtes des clients docker et est responsable de la création et de la gestion d'objets docker tels que les conteneurs, les images, volumes et réseaux. Il peut également communiquer avec les autres dockerd.

Rest API : Interface utilisée pour la communication client/serveur à travers des sockets UNIX ou l'interface réseau.

Lors de la création d'un container à partir des commandes clients comme docker run ou bien docker pull, docker utilise le registre pour récupérer les images configurées, par défaut, il utilise le docker hub qui héberge les images docker de toute la communauté, ou bien à partir d'un registre privé au niveau local.

1. daemon : processus qui s'exécute en arrière-plan sans interaction avec l'utilisateur.

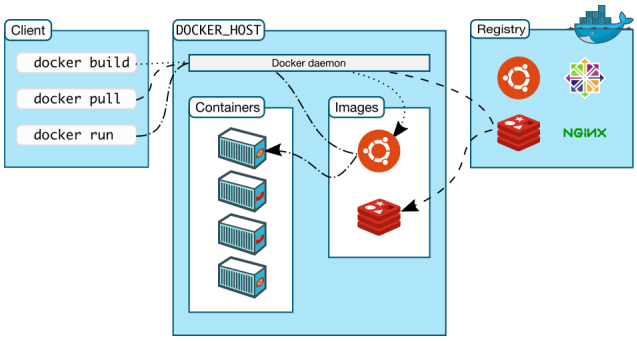

FIGURE 4.2 – Architecture Docker [5]

4.3.4 Principe de fonctionnement

Docker repose sur le principe d'isolation qui sépare les ressources du système d'exploitation hôte. Il utilise le noyau de linux et ses fonctionnalités comme les espaces noms (namespace) et les espaces de contrôles (Cgroups) et ainsi chaque conteneur docker aura sa propre adresse ip, son propre système de fichiers avec des limitations des ressources telles que le processeur et la mémoire.

Les images Docker sont basées sur un système de fichiers en couches. une couche représente un ensemble de changement que l'on effectue sur le système de fichiers de base. Ces changements sont effectués à partir des instructions au niveau du dockerfile. Ces couches sont empilées dans un ordre précis afin de construire une image, et peuvent être partagées entre plusieurs images. Ainsi, lorsque l'on récupère une image à partir du registre, cette dernière se télécharge couche par couche. Ces couches sont de type lecture seule, pour créer un conteneur une couche supplémentaire de lecture/écriture est ajoutée au sommet de la pile.

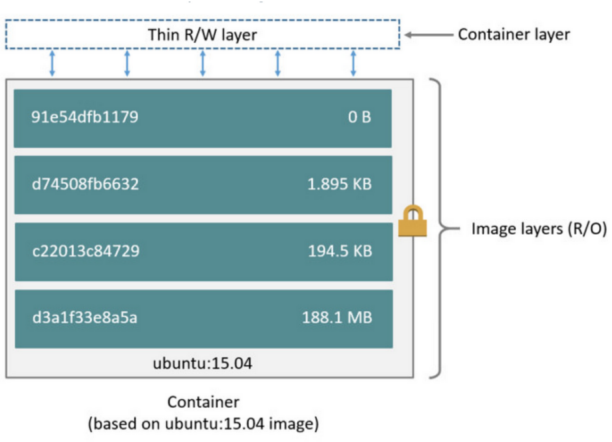

FIGURE 4.3 – Conteneur basé sur l'image Ubuntu 15.04 [3]

4.4 Orchestration

La conteneurisation est née d'un besoin répondant aux préoccupation des équipes de développement, mais de l'utilisation croissantes de ces derniers et de la multiplication des conteneurs par services (notion de micro-services) , d'autres problèmes ont vu le jour : la gestions des différents conteneurs, entre déploiement, évolutivité, mise en réseaux...

Effectivement, les conteneurs résolvent d'eux-mêmes de nombreux problèmes liés au processus de développement, tels que les dépendances et la compatibilité. De ce fait, la réflexion se porte maintenant sur la manière dont les conteneurs seront déployés. Comment faire évoluer les services déployés selon le besoin, comment s'assurer de la bonne utilisation des ressources fournies, et enfin, comment donner l'accès aux services hébergés.

4.4.1 Définition

"L'orchestration des conteneurs permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs." [9]

On peut dire aussi que l'orchestration est l'automatisation et la coordination des tâches, d'une manière à optimiser les performances, gérer les concurrences, et minimiser les problèmes et le temps de production.

4.4.2 Principe de fonctionnement

L'orchestration sert à gérer plusieures tâches, en créant un workflow, c'est-à-dire décrire comment les tâches seront exécutés et par qui. Sur un système basé sur l'orchestration, un administrateur décrit l'état désiré de système (performance, nombre des tâches exécutés, ressources utilisées...), à travers un ensemble des règles et spécifications. L'orchestrateur, en se basant sur ces paramètres, ajuste l'état des systèmes automatiquement, ses tâches incluent : la gestion des ressources physiques, l'ordonnancement des tâches, la mise à l'échelle... ce qui implique que l'orchestration nécessite une communication continue entre l'orchestrateur et les environnements d'exécution des tâches.

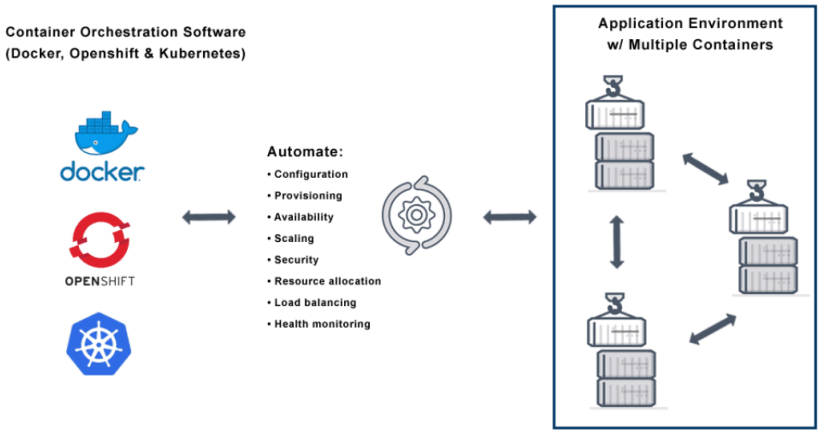

FIGURE 4.4 – Principe de fonctionnement de l'orchestration [10]

4.4.3 Outils

4.4.3.1 Etude comparative

	Kubernetes	DockerSwarm	Apache Mesos
Développé par	Google	Docker, Inc.	Apache Software Foundation
Open Source	oui	oui	oui
Type de conteneurs	Docker, Rkt	Docker	Docker, Rkt
Déploiement basé sur	YAML	Docker	JSON
nombre de noeuds minimum	un noeud maître et un noeud de travail	un noeud de travail	un noeud maître et un noeud de travail
taille de cluster	petit-moyen	petit	moyen-grand
Difficulté d'installation	difficile	facile	moyen
Difficulté d'usage	moyen	facile	difficile

TABLE 4.2 – Etude comparative entre les outils d'orchestration [4]

4.5 Kubernetes

Ainsi, bien que les conteneurs n'aient besoin que d'un runtime pour s'exécuter, de nombreuses organisations préfèrent les lancer avec un orchestrateur de conteneurs. Parmi ceux-ci, Kubernetes est de loin le plus populaire.

Un orchestrateur de conteneurs gère les aspects opérationnels d'exécution d'applications dans des conteneurs - Comme la découverte de services, le redémarrage des conteneurs en cas d'arrêt, l'allocation des ressources systèmes et la gestion des groupes de conteneurs qui doivent être étroitement liés, pour n'en nommer que quelques-uns.

4.5.1 Définition

Kubernetes est une solution d'orchestration de conteneurs développée par Google et publiée en 2014.

Selon le site officiel de Kubernetes [12] :

"Kubernetes est une plate-forme open-source extensible et portable pour la gestion de charges de travail (workloads) et de services conteneurisés. Elle favorise à la fois l'écriture de configuration déclarative (declarative configuration) et l'automatisation. C'est un large écosystème en rapide expansion. Les services, le support et les outils Kubernetes sont largement disponibles."

Autrement dit, Kubernetes est un logiciel utilisé pour automatiser les opérations de gestion des conteneurs, notamment la création, le déploiement et la mise à l'échelle. Cet outil permet de coordonner un grand nombre de conteneurs déployés sur plusieurs machines qui travaillent ensemble, d'une manière efficace.

4.5.2 Infrastructure Kubernetes

Une infrastructure kubernetes peut être divisée en 5 parties :

- **Couche physique :** il s'agit de l'infrastructure physique sur lequel le cluster est déployé, elle peut être purement physique, virtuelle (en site ou sur le cloud), publique, privée ou hybride.
- **Le cluster Kubernetes :** c'est la première couche d'abstraction sur l'infrastructure physique, c'est une vue logique, des machines sur lesquelles kubernetes est déployé.
- **Les services liés au cluster Kubernetes :** c'est des services (logicielles) qui permettent de monitorer, gérer et administrer le cluster Kubernetes.
- **Les services liés au applications :** c'est des services (logicielles) qui permettent de monitorer, gérer et administrer les applications qui s'exécutent sur le cluster.
- **Les applications :** les services et les applications qui s'exécutent sur le cluster.

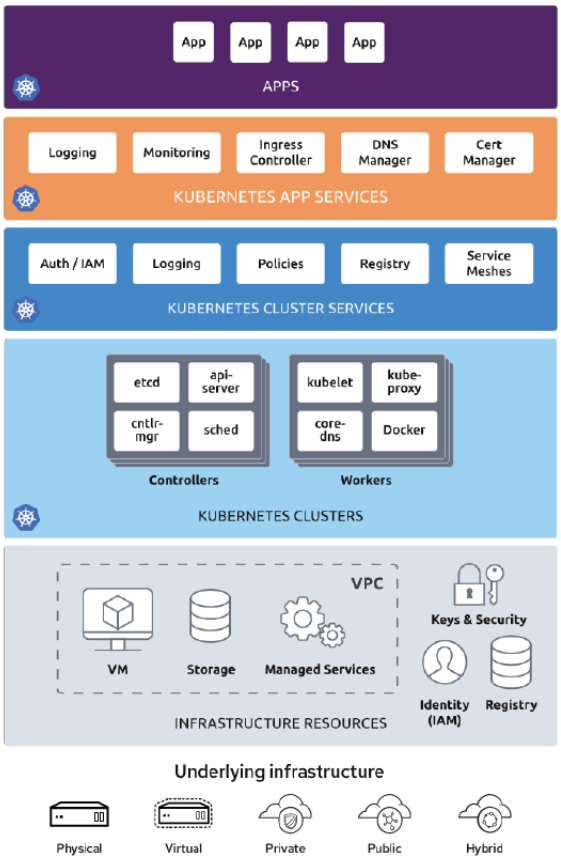

FIGURE 4.5 – Infrastructure Kubernetes [1][11]

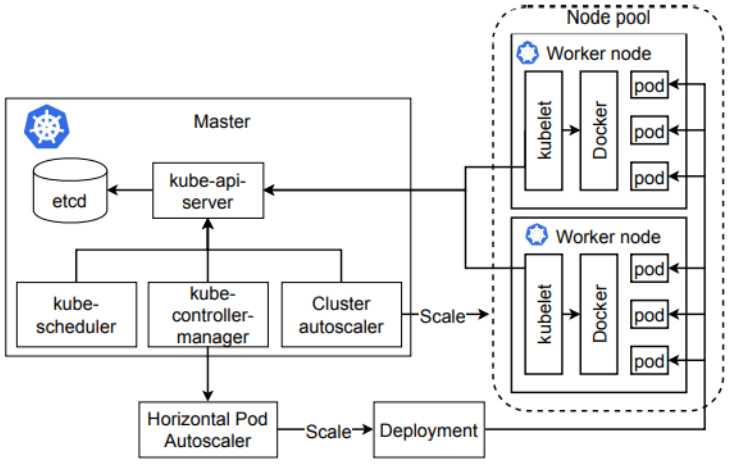

FIGURE 4.6 – Architecture Kubernetes [8]

4.5.3 Principe de fonctionnement

4.5.3.1 Architecture

Un cluster Kubernetes se compose de 2 parties importantes : une partie plan de contrôle, constituée d'un ou plusieurs nœuds maîtres, et d'une partie responsable de l'exécution des conteneurs constitué de un ou plusieurs nœuds de travail -workers-. La communication entre ces deux parties se fait à travers un API REST, géré par le kube-api-server.

Un nœud étant responsable de l'exécution d'un conteneur, il comprend un container runtime, et un ensemble de PODs, géré par un Kubelet qui est toujours en communication avec le plan de contrôle.

Le plan de contrôle contient plusieurs composantes responsables de maintenir ou d'atteindre un état défini par l'administrateur de cluster.

4.5.3.2 Modules :

Pod : c'est la plus petite unité d'exécution qui peut être créée et déployée sur un cluster kubernetes. Un pod peut contenir un ou plusieurs conteneurs (cas des applications qui sont fortement couplées), qui seront toujours placés et ordonnancés ensemble, et qui partagent le même espace de stockage, la même adresse IP et la même plage de ports. Un pod est muni également d'un ensemble d'options qui contrôlent comment ses conteneurs doivent s'exécuter.

Cluster : L'ensemble des machines, physiques ou virtuelles, appelées nœuds, sur lesquelles Kubernetes est déployé.

Nœuds : Une unique machine du cluster, qui peut être soit physiques ou virtuelles. Un noeud peut se voir attribuer des rôles :

- **Noeud de travail (Worker node) :** c'est un nœud responsable de l'exécution de conteneurs, contrôlé par l'orchestrateur de cluster. Chaque nœud contient un ou plusieurs unités logiques appelées pod, et les services suivants qui permettent leur exécution : Container Runtime, Kubelet et Kube-proxy.
- **Nœud maître (Master Node) :** c'est le nœud responsable de la gestion de cluster, i.e l'orchestrateur de l'ensemble des nœuds de travail. Un cluster peut avoir plusieurs nœuds maîtres, pour garantir la haute disponibilité, dans ce cas il y aura, parmi les nœuds maîtres, un maître principal suivi par les autres.

Objet Kubernetes : se sont des entités persistantes sur le cluster qui servent à décrire son état. Il sont souvent stockés suivant le format .yaml

ETCD : une base de données clé-valeur consistante et hautement disponible utilisée comme mémoire de sauvegarde pour toutes les données du cluster.

Container Runtime : c'est le logiciel qui permet de gérer et d'exécuter des conteneurs, par exemple : Docker Engine.

Kubelet : c'est un "agent" qui s'exécute sur chaque nœud de travail pour gérer l'ensemble des pods et des conteneurs et assurer leur santé (bon état), en communiquant avec le nœud maître a travers l'API de kubernetes.

Kube-proxy : c'est un proxy qui s'exécute sur chaque nœud de travail. Sa responsabilité est d'assurer communication vers les Pods depuis l'intérieur ou à l'extérieur du cluster. Il intervient aussi dans l'équilibrage de charge.

Kube-apiserver : kubernetes utilise l'API REST pour les communications internes et externes. Le serveur API constitue l'interface de communication au sein de cluster qui fournit les opérations REST est définie les objets qui peuvent être échangés entre les nœuds ou avec l'extérieur.

kube-scheduler : c'est l'ordonnanceur de kubernetes qui s'exécute sur le nœud maître. Il est responsable d'affecter un pod au nœud qui convient. Le choix est pris selon les besoins de pod et même de conteneurs au sein de pod.

kube-controller-manager : c'est un logiciel toujours en cours d'exécution (daemon) qui contient la boucle de contrôle [2] qui régule l'état du système. Il surveille l'état de système à travers le kube-apiserver, et il essaye de maintenir ou d'atteindre l'état désiré de cluster.

Deployment : c'est un objet Kubernetes qui sert à décrire un état désiré de déploiement, c'est-à-dire le nombre des PODs dans le cluster. Ces objets sont utilisés par des contrôleurs qui travaillent à assurer cet état.

Cluster-Autoscale : c'est un composant de kubernetes qui contrôle la taille de Cluster, en créant ou supprimant des nouveaux pods ou nœuds selon le besoin.

Vertical-Pod-Autoscaler : c'est un composant de kubernetes qui contrôle les ressources physique (CPU et mémoire) de cluster selon la charge de travail : mise à l'échelle verticale [3].

Horizontal-Pod-Autoscaler : c'est un composant de kubernetes qui contrôle le nombre des PODs dans le cluster selon la charge de travail : une mise à l'échelle horizontale [4].

2. Boucle de contrôle : élément, aux composants physiques ou logiciels, qui surveille et régule l'état d'un système.

3. mise à l'échelle verticale (Scale-In) : consiste à ajouter plus de ressources (ex : CPU, RAM...) aux machines ou périphériques existants.

4. Mise à l'échelle horizontale (Scale-Out) : consiste à ajouter plus de machines ou de périphériques informatiques au système.

4.5.4 Fonctionnalités

Kubernetes offre plusieurs fonctionnalités, parmi les plus importantes on retrouve :

Configuration déclarative : kubernetes permet de décrire un état désiré du système, à travers des fichiers de configuration. Après cette déclaration, le logiciel va toujours essayer de maintenir et/ou d'atteindre cet état.

Réparation et scalabilité automatique : pour s'assurer que l'état actuel de cluster est conforme au état désiré déclaré, Kubernetes se charge de créer, redémarrer, remplacer, ou détruire des conteneurs d'une manière automatique selon les besoins.

Infrastructure immuable : kubernetes permet de mettre en place une infrastructure qui reste toujours inchangée pour une application donnée, mais plutôt destructible. Par exemple, on peut remplacer un conteneur ayant certaines caractéristiques (dépendances, ressources...) par un autre complètement différent. Le but de l'immuabilité est de réduire la complexité et le risque aux pannes, et d'améliorer la sécurité et facilite la maintenance.

Allocation des ressources : ayant les informations sur les ressources en mémoire et en CPU nécessaire pour un conteneur sur le cluster, Kubernetes peut l'associer à une machine d'une manière automatique et efficace.

Découverte de services : Kubernetes permet d'annoncer certaines applications comme étant des services réseaux accessibles de l'extérieur, en attribuant des adresses IP aux pods et un nom DNS unique pour un ensemble de pods. Et lorsqu'une requête est reçue de l'extérieur, il assure un équilibrage du travail, et la redirection de requêtes.

Gestion de cycle de vie : Kubernetes facilite la gestion de vie des services et des applications, notamment le déploiement, la maintenance, la mise à l'échelle et l'extensibilité.

Stockage : Kubernetes permet d'ajouter plusieurs supports et solutions de sauvegarde de fichiers, comme le stockage local, le cloud...

Sécurité : Kubernetes offre la possibilité d'intégrer les processus DevSecOps avec la gestion des conteneurs tout au long de leur cycle de vie.

4.5.5 Avantages et inconvénients

Advantages :

- **Gain de temps :** l'autonomisation des différentes opérations de gestion de conteneurs permet de gagner du temps, et d'améliorer les performances.
- **Scalabilité :** les clusters kubernetes sont évolutifs, c'est-à-dire que l'on peut ajouter des nodes, et des conteneurs à la demande d'une manière facile, rapide et efficace.
- **Economisation :** Kubernetes permet également d'économiser de l'argent et de l'énergie, parce qu'il offre une infrastructure évolutive où le minimum de ressources physiques est nécessaire.
- **Abstraction d'infrastructure :** Kubernetes offre une abstraction de l'infrastructure physique, soit le nombre de machines physiques, leurs caractéristiques, systèmes d'exploitation...ne sont connus ni par l'équipe de développement ni par les utilisateurs.
- **Haut disponibilité :** Kubernetes facilite la mise en place des clusters hautement disponibles, grâce à ses fonctionnalités d'équilibrage de charge, et de réparation et scalabilité automatique, qui peuvent même être déployés sur des régions distantes dans le monde.
- **Flexibilité et Conformité :** Kubernetes permet à la fois de gérer une infrastructure qui est flexible (scalable, extensible et indépendante de placement géographique), et répondant aux besoins et exigences de QoS (sécurité...)ou autres(ressources disponibles...) grâce aux configurations déclaratives.

Inconvénients :

- **Complexité :** la migration vers Kubernetes (installation et de configuration) est souvent difficile surtout si la taille du système existant est importante. D'autres part, si le système est de taille réduite, l'utilisation d'une solution aussi complexe et sophistiquée que Kubernetes ne représente pas le choix le plus judicieux.
- **Approche centralisée :** comme toute solution d'orchestration, le cluster kubernetes dépend d'un point central de coordination. Ce qui devient un point de défaillance unique(SPOF), c'est-à-dire que s'il tombe en panne, le système dans son entièreté ne fonctionnera plus.
- **Trafic Réseau :** Kubernetes nécessite une communication permanente entre l'orchestrateur et les nœuds de cluster, ce qui peut encombrer le réseau et diminuer les performances.

5. Gestion du projet

Dans cette partie, on décrit la démarche de gestion de projet adoptée pour assurer le bon déroulement du projet.

5.1 Processus de Flux de Travail

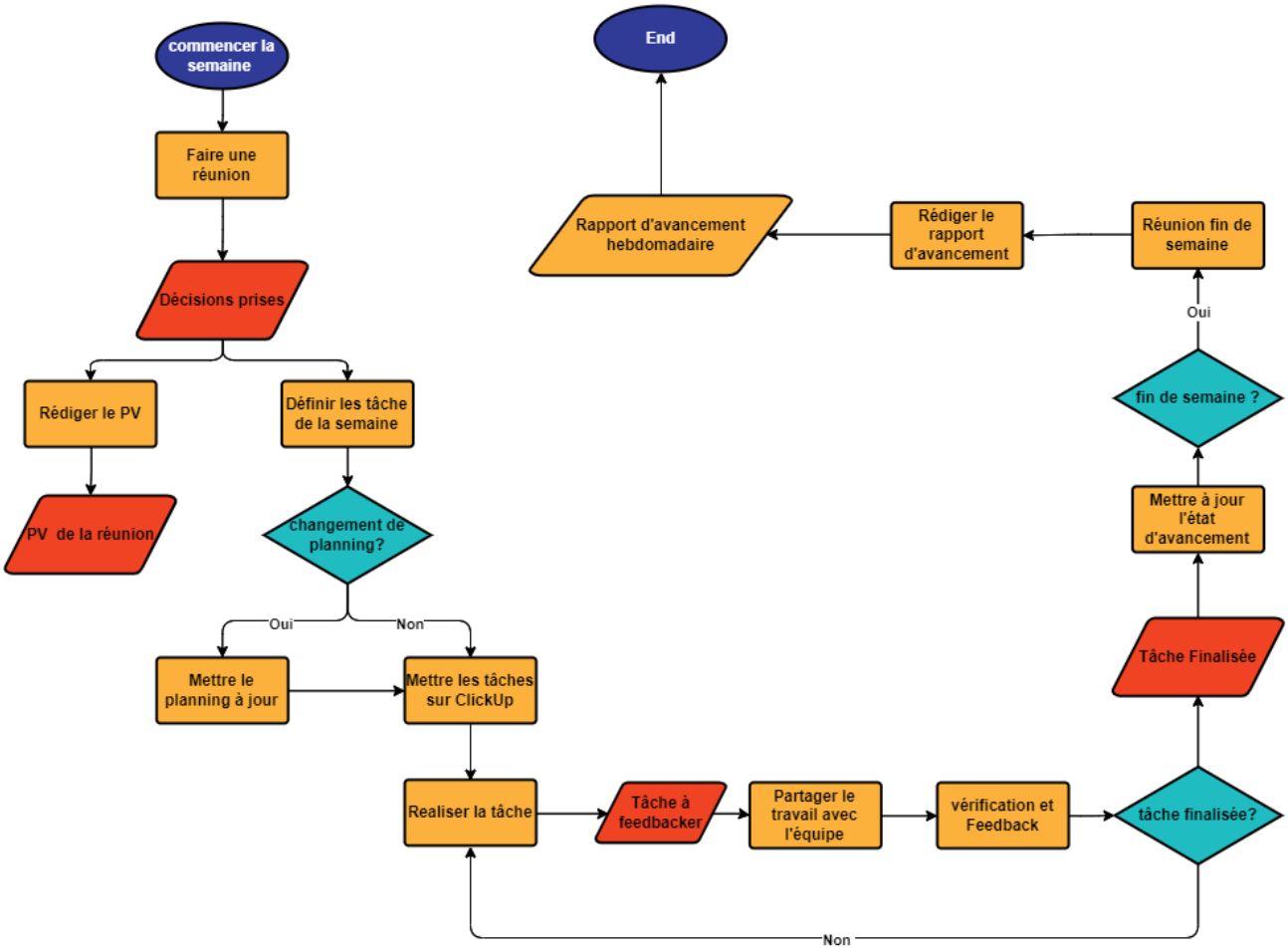

FIGURE 5.1 – Diagramme de processus de flux de travail

5.2 Planning et Etat d'avancement

5.2.1 Planning

On a adopté un planning d'exécution à 4 phases, illustrée par le diagramme de GANTT suivant, dont les durées ont été fournies par des délais exigés par le client :

Phase 1 : initialisation (construction l'équipe, mise en place d'environnement de travail, et compréhension de problématiques et de cadre d'étude)

Phase 2 : étude de l'existant et étude théorique (familiarisation avec Kubernetes et Docker)

Phase 3 : mise en place et exploitation d'une application conteneurisée : création et exploitation des conteneurs Docker sur des applications utilisées pour les travaux dirigés dans L'ESI.

Phase 4 : mise en place du cluster Kubernetes : création et configuration de cluster Kubernetes.

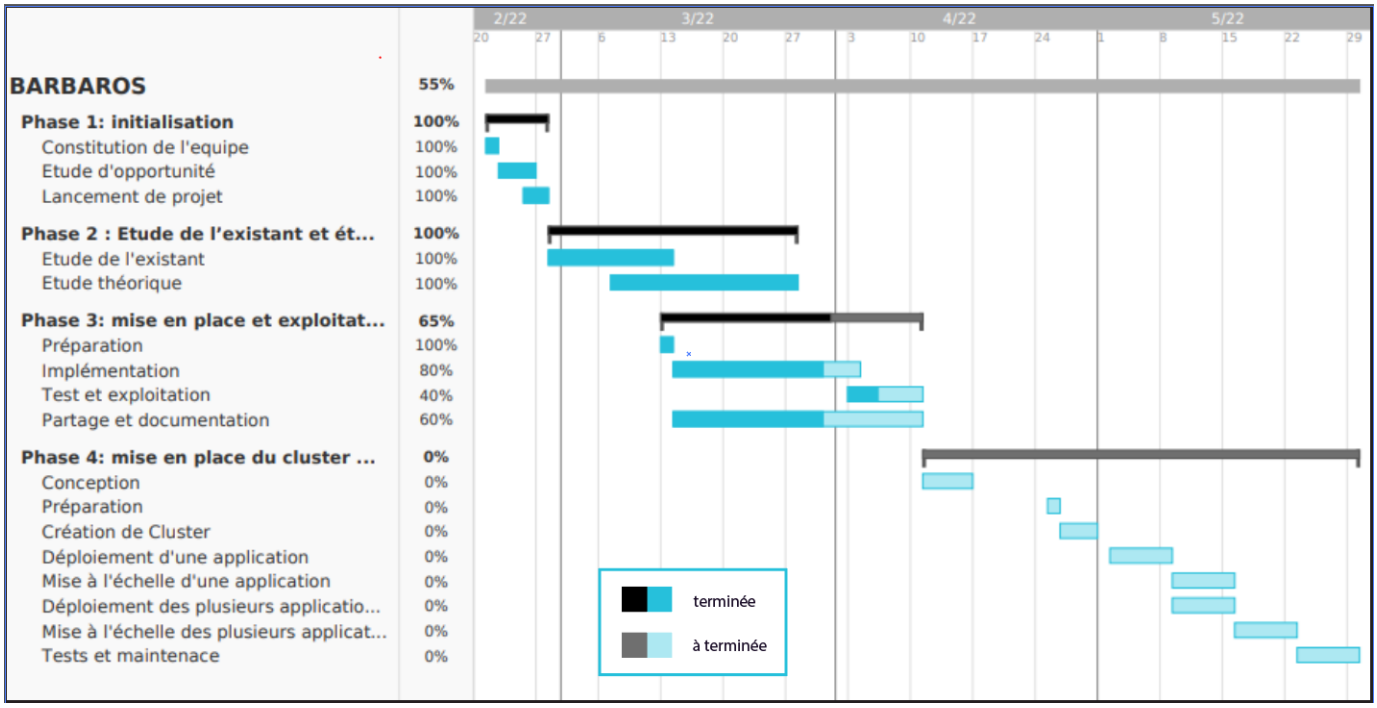

FIGURE 5.2 – Diagramme de Gantt

5.2.2 Etat d'avancement

Tâches réalisées :

- Finalisation de la phase 1.
- Élaboration de l'étude de l'existant sur la gestion des travaux pratiques au sein de l'ESI, et de l'étude théorique.
- Lancement de la phase 3 : 3 images docker ont été créées avec succès : application Bash de linux, Cisco Packet Tracer et IntelliJ IDEA.

Tâches en progrès :

- Tests unitaires sur les images Docker créées.
- Rédaction des comptes rendus de l'implémentation des images Docker.

6. Conclusion

Pour conclure, nous aimerions faire un retour sur le mandat. Dans le cadre de cette première phase du projet, nous avions étudié les différentes solutions de conteneurisation et orchestration d'applications. Nous avions à choisir les technologies d'une solution moderne qui viendrait remplacer un système archaïque basé sur les VM. Pour rendre cela possible, nous avons eu à mener une analyse en profondeur de la solution existante et des besoins du client. Finalement, nous aimerions souligner le fait que cette phase nous permettra de mettre en pratique les notions théoriques afin de délivrer des produits de qualité au bénéfice de l'ensemble des enseignants et étudiants de l'Ecole Nationale Supérieure d'Informatique d'Alger .

Bibliographie

1.[1] *A Year of Helping Customers Build Production-Ready Kubernetes Infrastructure* [2019], https://www.pulumi.com/blog/crosswalk-kubernetes/.

2.[2] BEKKALI, H. [2018], '5 questions à vous poser avant de migrer vers le cloud', https://www.syloe.com/migrer-vers-le-cloud-questions/.

3.[3] Burillo, M. [2018], 'How to implement docker image scanning with open source tools ?', https://sysdig.com/blog/docker-image-scanning/.

4.[4] Chigira, M. [2019], 'Container orchestration in 2019', https://scoutapm.com/blog/container-orchestration-in-2019.

5.[5] *Docker overview* [n.d.], https://docs.docker.com/get-started/overview/.

6.[6] Eddine, M. D. [2021], Cours de virtualisation et cloud computing (vcl), chapitre 1 : Introduction à la virtualisation.

7.[7] Gourvennec, Y. [2020], 'Définition et état des lieux de la conteneurisation en cloud computing', https://www.selceon.com/glossary/conteneurisation-cloud-computing/.

8.[8] Tamiru, M. A., Tordsson, J., Elmroth, E. and Pierre, G. [2020], An experimental evaluation of the kubernetes cluster autoscaler in the cloud.

9.[9] *What is container orchestration ?* [2019], https://www.redhat.com/en/topics/containers/what-is-container-orchestration.

10.[10] *What is Container Orchestration ?* [n.d.], https://avinetworks.com/glossary/container-orchestration/.

11.[11] *What is Kubernetes ?* [2020], https://www.redhat.com/en/topics/containers/what-is-kubernetes.

12.[12] *What is Kubernetes ?* [2021], https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.