

Elyes Gassara

# Docker/ Kubernetes

Pour optimiser et accélérer les développements  
d'applications conteneurisées



● Éditions  
**EYROLLES**

# Docker/ Kubernetes

Kubernetes, communément appelé « k8s », est un large écosystème en rapide expansion. Initialement développé par Google, il permet de gérer les applications conteneurisées dans un environnement en cluster. Ce système d'orchestration de conteneurs permet l'automatisation du déploiement, la mise à l'échelle et la gestion de l'application. Le projet Kubernetes est devenu open source en 2014.

Ce livre a pour objectif d'accompagner les développeurs dans l'utilisation de Docker et de Kubernetes pour leur permettre d'optimiser et d'accélérer leurs développements d'applications conteneurisées. Il est idéal pour les architectes d'infrastructure cloud, les ingénieurs DevOps, les administrateurs système et les responsables de l'ingénierie qui ont besoin de connaître les bases de Kubernetes et sont prêts à appliquer les meilleures pratiques de l'industrie du cloud pour concevoir, créer et exploiter des clusters Kubernetes de qualité en production.

Au fil des chapitres, l'auteur met en lumière les pratiques nécessaires à suivre par les personnes en charge de la création d'une application dans Kubernetes et apporte les informations nécessaires sur l'utilisation de services externes tels qu'une base de données ou un serveur web. Une démarche pédagogique vous accompagne pas à pas, avec des exemples simples et une continuité dans les travaux qui sont à la base des activités pratiques, et vous garantit donc une bonne compréhension des notions théoriques.

## À qui s'adresse cet ouvrage ?

- Développeurs
- Architectes d'infrastructure cloud
- Ingénieurs DevOps
- Administrateurs systèmes & réseaux

## Au sommaire

**Introduction à Docker.** Installation de Docker • Utilisation de la commande docker • Gestion du réseau sous Docker • **Création et gestion d'images Docker.** Persistance des données • Créer sa propre image à l'aide d'un fichier Dockerfile • Conteneur en production • Docker Compose • Application • **Docker Swarm.** Créer un cluster Swarm • Déployer une application à service individuel dans un Swarm avec Docker Service • Virtual IP et Service Discovery • Déployer une application multiservice dans un Swarm • **Kubernetes.** Kubernetes et Docker Swarm • Présentation de l'architecture de Kubernetes • Fonctionnalités de Kubernetes • Minikube • Application • **Kubernetes en production.** Création d'un cluster Kubernetes sous AKS • Mise à l'échelle du cluster Kubernetes sous AKS • Déploiement d'une application sous AKS • **Développement et conteneurisation d'une application web moderne.** Étape 1 : développer et tester l'application en local avec un IDE • Étape 2 : embarquer l'application dans des conteneurs Docker • Étape 3 : déployer l'application sur un serveur de production avec Kubernetes

**Elyes Gassara** est ingénieur et Maître technologue à ISET Charguia, Tunisie. Il est enseignant depuis 2008 et instructeur accrédité dans le programme AWS Academy et LPI, CISCO et Huawei HCAL.

[www.editions-eyrolles.com](http://www.editions-eyrolles.com)

# Docker/Kubernetes

DANS LA MÊME COLLECTION

M. BIDAULT. – **Programmation Excel avec VBA.**

N° 0100544, 3<sup>e</sup> édition, 2022, 544 pages.

R. BRUCHEZ. – **Les bases de données NoSQL - Comprendre et mettre en œuvre.**

N° 67866, 2021, 290 pages.

H. BERSINI, K. HASSELMANN. – **L'intelligence artificielle en pratique avec Python.**

N° 00456, 2021, 136 pages.

S. AKBARZADEH, J. SCHWOERER, B. BAILLY, W. LABIDI. – **Les réseaux 5G.**

N° 67898, 2020, 580 pages.

Y. BENZAKI. – **Les data sciences en 100 questions/réponses.**

N° 67951, 2020, 126 pages.

K. NOVAC. – **Administration Linux par la pratique. Configurer les services les plus courants - Tome 2.**

N° 67949, 2020, 418 pages.

É. SARRION. – **React.js.**

N° 67756, 2019, 358 pages.

R. GOETTER. – **Grid Layout.**

N° 67683, 2019, 144 pages.

C. BLAESS. – **Solutions temps réel sous Linux.**

N° 67711, 3<sup>e</sup> édition, 2019, 320 pages.

C. PIERRE DE GEYER, J. PAULI, P. MARTIN, E. DASPET. – **PHP 7 avancé.**

N° 67720, 2<sup>e</sup> édition, 2018, 736 pages.

H. WICKHAM, G. GROLEMUND. – **R pour les data sciences.**

N° 67571, 2018, 496 pages.

Retrouvez nos bundles (livres papier + e-book) et livres numériques sur  
<http://izibook.eyrolles.com>

Elyes Gassara

# Docker/Kubernetes

Pour optimiser et accélérer les développements  
d'applications conteneurisées

● Éditions  
**EYROLLES**

ÉDITIONS EYROLLES  
61, bd Saint-Germain  
75240 Paris Cedex 05  
[www.editions-eyrolles.com](http://www.editions-eyrolles.com)

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre Français d'exploitation du droit de copie, 20, rue des Grands Augustins, 75006 Paris.

© Éditions Eyrolles, 2022, ISBN : 978-2-416-00569-5

# Table des matières

---

<b>Introduction .....</b>	<b>1</b>
---------------------------	----------

## CHAPITRE 1

<b>Introduction à Docker .....</b>	<b>3</b>
------------------------------------	----------

Introduction générale .....	3
Sans l'utilisation de Docker .....	4
Avec l'utilisation de Docker .....	6
Installation de Docker .....	11
Installation depuis les dépôts officiels Ubuntu .....	11
Installation à partir du référentiel officiel de Docker .....	11
Utilisation de la commande docker .....	13
Utilisation des images Docker .....	15
Exécuter un conteneur Docker .....	18
Publiez votre première image Docker sur Docker Hub .....	19
<i>Publier l'image.</i> .....	20
Gestion du réseau sous Docker .....	22
Les types de réseaux .....	22
Réseaux définis par l'utilisateur .....	26
<i>Création d'un réseau de type bridge nommé « monbridge ».</i> .....	27
<i>Test du serveur DNS intégré.</i> .....	29
Docker sous Windows .....	29
<i>Docker Toolbox</i> .....	29
<i>Docker Desktop</i> .....	29
<i>Docker pour Windows Server</i> .....	30
PWD .....	30
Conclusion .....	31

## CHAPITRE 2

<b>Création et gestion d'images Docker .....</b>	<b>35</b>
--	-----------

Persistance des données .....	35
Répertoire hôte monté (bind mounts) .....	36
Volumes .....	38
Créer sa propre image à l'aide d'un fichier Dockerfile .....	42
Exemple 1 .....	42
<i>Différence entre CMD et ENTRYPOINT.</i> .....	46
<i>Différence entre COPY et ADD</i> .....	47

<i>Différence entre MAINTAINER et LABEL</i> .....	47
Exemple 2 .....	47
Conteneur en production .....	49
Docker Compose .....	57
Application .....	61
Conclusion .....	65

## CHAPITRE 3

### **Docker Swarm**..... 69

Créer un cluster Swarm .....	70
Déployer une application à service individuel dans un Swarm avec Docker Service .....	72
Virtual IP et Service Discovery .....	78
Déployer une application multiservice dans un Swarm .....	80
Déployer une application multiservice dans un Swarm avec Docker Stack .....	81
Conclusion .....	85

## CHAPITRE 4

### **Kubernetes**..... 89

Kubernetes et Docker Swarm .....	89
Présentation de l'architecture de Kubernetes .....	90
Plan de contrôle .....	92
<i>kube-apiserver</i> .....	92
<i>kube-scheduler</i> .....	92
<i>ETCD</i> .....	92
<i>kube-controller-manager</i> .....	93
Nœud Kubernetes .....	93
<i>Container runtime</i> .....	93
<i>Kubelet</i> .....	94
<i>kube-proxy</i> .....	94
Stockage persistant .....	94
Registre de conteneurs .....	95
Fonctionnalités de Kubernetes .....	95
Interfaces Kubernetes .....	96
<i>Plug-ins réseau pour Kubernetes</i> .....	96
<i>Runtimes de conteneurs améliorés</i> .....	96
<i>Plug-ins de volume avec CSI</i> .....	96
Résolution DNS .....	97
Déploiement d'une application .....	97
Minikube .....	98
Créer un pod avec la méthode impérative .....	100
Créer un pod avec la méthode déclarative .....	102
Kubernetes Dashboard .....	103
ReplicaSets .....	104
Déploiements .....	107
Services .....	111
Application .....	113
Rolling Update .....	117



Job .....	119
CronJob .....	119
DaemonSet .....	119
ConfigMap .....	119
Secrets .....	120
Conclusion .....	120

## CHAPITRE 5

### **Kubernetes en production..... 123**

Création d'un cluster Kubernetes sous AKS .....	124
Mise à l'échelle du cluster Kubernetes sous AKS .....	134
Déploiement d'une application sous AKS .....	135
Conclusion .....	145

## CHAPITRE 6

### **Développement et conteneurisation d'une application web moderne..... 147**

Étape 1 : développer et tester l'application en local avec un IDE .....	147
Partie backend .....	149
Partie frontend .....	161
Étape 2 : embarquer l'application dans des conteneurs Docker .....	168
Étape 3 : déployer l'application sur un serveur de production avec Kubernetes .....	176
Conclusion .....	184

### **À vous de jouer..... 187**

### **Références..... 189**

### **Index..... 191**



# Introduction

---

Kubernetes, ou communément appelé « k8s », est un large écosystème en rapide expansion. Initialement développé par Google, il permet de gérer les applications conteneurisées dans un environnement en cluster. Ce système d'orchestration de conteneurs permet l'automatisation du déploiement, la mise à l'échelle et la gestion de l'application. Google a rendu open source le projet Kubernetes en 2014.

La Cloud Native Computing Foundation (CNCF) est un projet de la Linux Foundation qui a été fondé en 2015 pour aider à faire progresser la technologie des conteneurs et rassembler les industries technologiques autour de son évolution. Il a été annoncé en même temps que Kubernetes 1.0, qui a contribué à la Linux Foundation par Google en tant qu'initiateur de cette technologie. Une approche cloud native consiste à développer une application en profitant des différents avantages offerts par le cloud.

Tout d'abord, elle permet aux entreprises de transformer plus rapidement leurs idées d'applications en produits disponibles sur le marché.

En outre, elle permet une scalabilité accrue pour les applications. Elle offre également une réduction efficace des charges car les entreprises dépensent moins d'argent dans l'hébergement.

De plus, les applications cloud natives présentent un design modulaire, car un grand nombre de leurs fonctions peuvent être décomposées en microservices. Il est donc possible de désactiver certaines fonctions ou de déployer des mises à jour pour des modules spécifiques plutôt que pour l'application complète.

Kubernetes est un système permettant d'exécuter et de coordonner des applications conteneurisées sur un cluster de machines. Il s'agit d'une plate-forme conçue pour gérer complètement le cycle de vie des applications et des services conteneurisés à l'aide de méthodes offrant prévisibilité, évolutivité et haute disponibilité.

En tant qu'un des meilleurs outils DevOps, l'objectif principal de Kubernetes est de fournir une plate-forme qui permet de rationaliser le déploiement, la mise à l'échelle et les opérations des conteneurs d'applications sur un cluster d'hôtes.

De nombreux services cloud fournissent une plate-forme en tant que service (PaaS) basée sur Kubernetes ainsi qu'une infrastructure en tant que service (IaaS). K8s fournit donc une partie

des fonctionnalités des offres PaaS telles que le déploiement, la mise à l'échelle, l'équilibrage de charge (*load balancing*), la journalisation (*logging*) et la surveillance (*monitoring*).

DevOps est une évolution naturelle du développement de logiciels qui rassemble les équipes des opérations et du développement.

Elle vise à aligner les équipes Dev et Ops avec des objectifs communs car elles avaient des objectifs différents.

- L'objectif de l'équipe Dev est de prendre autant de nouvelles fonctionnalités à la production que possible.
- L'objectif de l'équipe Ops est de maintenir l'environnement de production aussi stable que possible.

L'utilisation des conteneurs est une tendance DevOps pour éviter une adhérence système de l'application en déployant une application et ses bibliothèques dans un container (une enveloppe) qui va être complètement abstrait de l'OS de base : votre application est alors conteneurisée et donc isolée de l'OS.

Kubernetes offre les outils qui permettent d'orchestrer une application conteneurisée complexe et volumineuse. Vous devez alors choisir le système d'exploitation, l'environnement d'exécution des conteneurs, les outils d'intégration et de déploiement continus (CI/CD), les services d'applications, le système de stockage et de nombreux autres composants.

Ce livre a pour objectif de vous transmettre les compétences nécessaires pour conteneuriser et déployer une application sur un cluster Kubernetes et notamment sur une plate-forme Kubernetes gérée par Microsoft Azure, sans vous soucier de l'ordinateur sous-jacent et des frais de gestion.

Vous apprendrez tout d'abord les principes de base de Docker et vous verrez comment cela s'applique à une architecture de microservices.

Vous découvrirez ensuite comment déployer une application sur Docker Swarm et sur un cluster Kubernetes local.

Vous verrez également comment cette expérience se traduit par le déploiement de la même application sur Azure Kubernetes Service sans aucune modification de la démarche de déploiement.

Enfin, pour mettre en œuvre tout ce savoir-faire et le transposer dans le monde professionnel, vous découvrirez la démarche complète de développement et de conteneurisation d'une application web moderne avec déploiement de cette dernière dans un serveur de production avec Kubernetes.

# Introduction à Docker

---

## Introduction générale

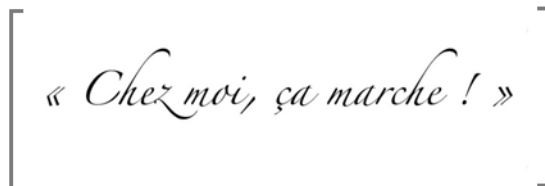
La technologie de conteneurs Docker s'est imposée au cours des deux dernières années, comme une alternative flexible et légère à la virtualisation, pour le déploiement d'applications web et d'applications cloud.

Docker est un programme de virtualisation au niveau du système d'exploitation, développé par Docker Inc. Il a été purement écrit dans le langage de programmation Go.

Docker a été initialement publié en mars 2013, il permet de créer des conteneurs applicatifs, architecturés sous forme de microservices. Le code source de Docker est hébergé sur le référentiel GitHub.

La problématique à laquelle on répond avec les conteneurs et Docker (le moteur d'exécution des conteneurs le plus utilisé aujourd'hui) est celle qui a été accentuée avec le DevOps et qui existait déjà auparavant. Il s'agit de l'adage « Chez moi ça marche ! »

Figure 1-1



« *Chez moi, ça marche !* »

Les administrateurs réseau et système viennent nous voir, en nous exposant le problème qu'ils ont rencontré, par exemple une erreur qui s'affiche lors de l'exécution d'un logiciel que nous

avons développé, mais nous n'avons jamais rencontré ce problème. Ceci vient du fait que l'environnement de production est différent de l'environnement de développement.

On appelle « serveur de production » le serveur sur lequel sera installée l'application une fois terminée, donc accessible aux utilisateurs finaux. Par opposition, on parle de « serveur de développement » pour désigner le serveur sur lequel vous développez l'application. Le serveur de production est donc ouvert à l'accès de l'utilisateur final.

Sur ma machine de développement, des composants sont installés (SDK, python .Net, etc.), ainsi que des dépendances et des environnements de développement intégrés (EDI). Sur la machine de déploiement, en revanche, nous trouvons juste l'environnement d'exécution et pas forcément de la même version que celle utilisée sur le serveur de développement.

Comment alors uniformiser les environnements sur les postes des développeurs et sur les serveurs de production ?

## Sans l'utilisation de Docker

Pour faire fonctionner un serveur, il faut installer le système d'exploitation, les environnements d'exécution des logiciels nécessaires et ajouter les fichiers propres à notre application. Vous pouvez le faire soit sur un serveur physique, soit sur une machine virtuelle.

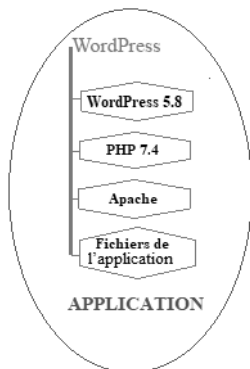
La virtualisation génère des ressources virtuelles qui peuvent être utilisées tout comme n'importe quelle ressource physique ou application. Une infrastructure virtuelle peut être une solution technologique abordable pour les organisations qui ne disposent pas du capital permettant d'acquérir des licences matérielles et logicielles, ni du budget nécessaire pour assurer la maintenance en continu du centre de données.

Les ressources sont allouées aux machines virtuelles via un programme appelé « hyperviseur », qui ressemble à un système d'exploitation pour les environnements virtualisés.

« C'est un processus qui crée et exécute des machines virtuelles (VM). Il permet à un ordinateur hôte de prendre en charge plusieurs VM clientes en partageant virtuellement ses ressources, telles que la mémoire et la capacité de traitement. » (VMware Inc, 2021)

Dans le cas ici d'une application WordPress, il faut installer le CMS WordPress 5.8, le PHP 7.4 et le serveur Apache ainsi que notre application.

**Figure 1-2**  
Serveur sans Docker



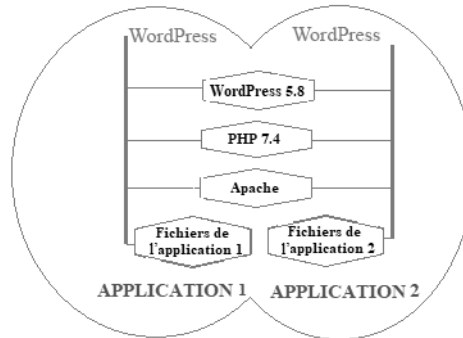
Tout ça a l'air parfait sur le papier, mais nous n'allons pas laisser notre serveur faire fonctionner une seule application.

Nous allons installer plusieurs applications qui partageraient des composants communs installés sur ce serveur tout en ayant des fichiers différents pour les différentes applications.

Le bon fonctionnement de ces applications requiert un serveur performant, efficace et disponible en permanence.

**Figure 1-3**

Composants partagés  
par plusieurs applications

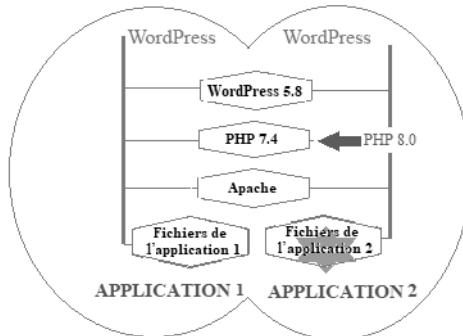


Jusqu'ici tout va bien, jusqu'au moment où je vais vouloir installer une version plus récente de PHP pour des raisons de sécurité, par exemple. Il faudra s'assurer que toutes les applications seront compatibles avec cette modification.

Dans cet exemple, il n'y a que deux applications mais sur un serveur de production il peut y en avoir une centaine qui tourne, alors vous pouvez imaginer l'effet de ces changements.

**Figure 1-4**

Les modifications provoquent  
des problèmes de compatibilité.



La modification provoque un dysfonctionnement de l'application 2, cela ne va pas du tout. De plus, vos applications n'ont pas besoin que de PHP, de nombreuses autres technologies sont installées sur le serveur.

Pour résoudre ce problème, il est possible de recourir à la virtualisation. En effet, elle permet de mettre chaque application (y compris le système d'exploitation) dans une image virtuelle, et c'est cette image qui sera délivrée aux clients.

En pratique, la virtualisation consomme beaucoup de ressources. Sa mise en œuvre est donc extrêmement lourde car on va embarquer dans une image virtuelle tout le système de disque, la mémoire, l'OS ainsi que les applications. C'est là que les conteneurs deviennent intéressants.

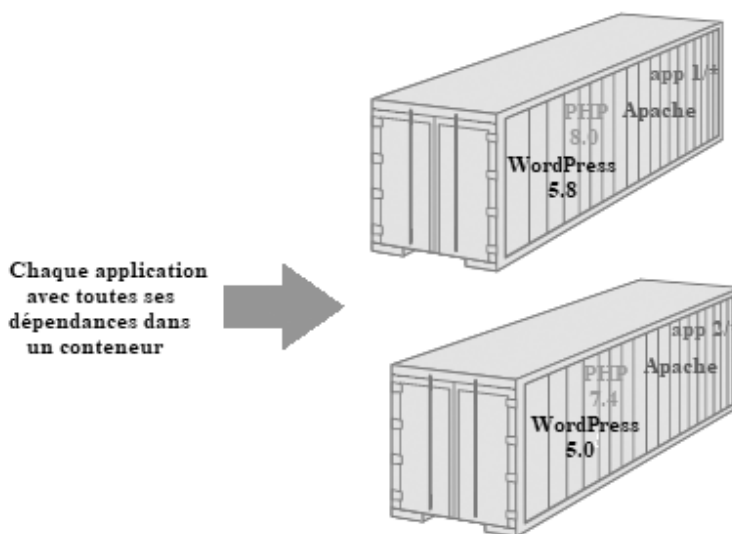
## Avec l'utilisation de Docker

La possibilité de partager des composants entre les applications rend inutile l'utilisation d'une image virtuelle qui contient tout. C'est ce que fait Docker, qui permet donc de remédier aux inconvénients de la virtualisation.

Avec Docker, on va faire tourner nos applications sous forme de conteneurs.

**Figure 1-5**

Conteneuriser les applications



Chaque « boîte » représente un conteneur qui contient une application avec toutes ses dépendances. Nous avons donc un WordPress, un PHP et un Apache propres à chaque conteneur et dédiés à chaque application.

Cela semble proche de la virtualisation, mais les conteneurs offrent quant à eux le partage des ressources et notamment l'OS.

Ainsi, d'un point de vue conceptuel, les deux applications semblent être isolées. Il n'y a pas de conflits entre les dépendances. Nous bénéficions des avantages de la virtualisation sans ses inconvénients.



*Lors de la décennie précédente, les hyperviseurs ont rapidement pris l'avantage dans le milieu professionnel, créant des écosystèmes complets avec des fonctionnalités avancées (migration à chaud des machines virtuelles, redondance, scripting, support d'OS différents, API...).*

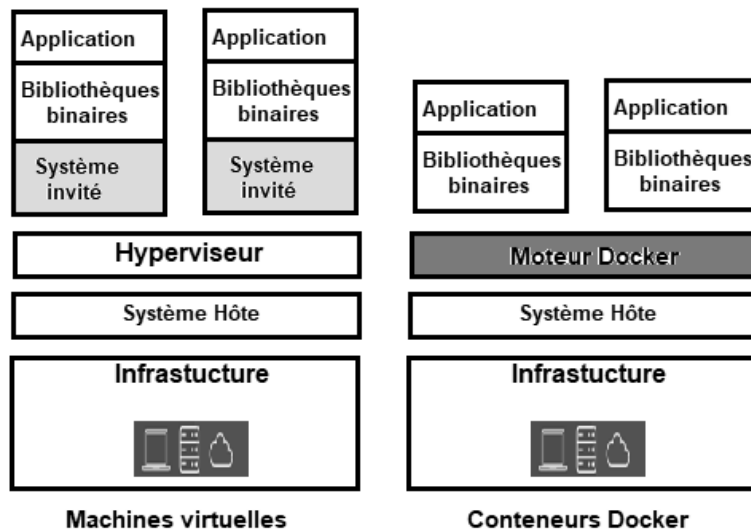
Jean-Marc Pouchoulon, *Docker : outil et sujet d'enseignement en D.U.T.*, 2016

Il est notamment possible de lancer plusieurs environnements d'OS sur la même machine, en les isolant les uns des autres.

De même, la virtualisation permet de réduire les coûts au sein d'une entreprise en diminuant le nombre de machines virtuelles nécessaires. Cependant, les hyperviseurs de machines virtuelles reposent sur une émulation du hardware, et requièrent donc beaucoup de puissance de calcul.

Les conteneurs sont donc proches des machines virtuelles, mais se partagent le même noyau de système d'exploitation et isolent les processus de l'application du reste du système. Contrairement à une machine virtuelle, qui virtualise l'ordinateur sous-jacent, un conteneur virtualise le système d'exploitation.

**Figure 1-6**  
Machines virtuelles versus  
conteneurs Docker



Chaque conteneur partage le noyau du système d'exploitation hôte, les fichiers binaires et les bibliothèques. De plus, il est uniquement possible de lire les composants partagés dans un conteneur.

Le partage des ressources du système d'exploitation, telles que les bibliothèques, minimise la nécessité de générer le code du système d'exploitation. Un serveur peut donc exécuter plusieurs charges de travail avec une seule installation du système d'exploitation, ce qui réduit la taille d'un conteneur, qui est généralement en mégaoctets. Cela prend un minimum de temps pour commencer.

Cette technologie de conteneurisation permet donc de créer un environnement d'exploitation portable pour le développement, les tests et le déploiement. En outre, plusieurs conteneurs peuvent fonctionner côte à côte sur la même plate-forme.

La plate-forme Docker offre un haut degré de portabilité, ce qui permet aux utilisateurs de s'enregistrer et de partager des conteneurs sur une grande variété d'hôtes au sein d'environnements publics et privés. Il est alors possible de développer des applications de façon plus efficace, en utilisant moins de ressources, et de déployer ces applications plus rapidement.

Juste avant de rentrer dans le concret, prenons un exemple pour bien comprendre la philosophie de Docker. Dans le domaine du transport maritime se pose souvent le problème de comment livrer.

Par analogie avec le monde informatique, on peut imaginer que chaque fournisseur (semblable au développeur) a des objets à livrer. Juste à côté se trouve un autre fournisseur possédant également des marchandises à livrer. À proximité de ces fournisseurs sont amarrés des bateaux (l'équivalent du serveur de production) structurés et organisés de manière différente d'un bateau à l'autre.

Auparavant, les bateaux restaient à quai pendant des jours, le temps pour les différents fournisseurs de venir livrer leurs marchandises et les charger.

**Figure 1-7**  
Chargement des marchandises  
dans les bateaux



Pour regrouper les marchandises et les transporter, elles étaient placées dans des conteneurs. Ces gros caissons métalliques étaient chargés soit sur le quai, soit dans les locaux de l'entreprise, pendant que le bateau restait à quai. Les conteneurs pouvaient aussi être acheminés par train ou par camion jusqu'au bateau, lequel pouvait en transporter beaucoup grâce à un système d'empilement selon une grille.

**Figure 1-8**  
Transporter les conteneurs



Ce type de fonctionnement et l'utilisation des conteneurs se retrouvent dans le domaine du développement.

En effet, Docker construit ses conteneurs en superposant des couches logicielles. Certaines d'entre elles sont en lecture seule afin d'être partagées par plusieurs conteneurs en même temps, ce qui permet d'envoyer une quantité minimale de données lors du transfert des images sur le réseau.

Par ailleurs, Docker fournit des images dans des registres, qui sont des entrepôts logiciels qui stockent les images.

Généralement, les images sont créées avec la commande `docker build` et vont produire un conteneur quand elles sont lancées avec la commande `run`.

Sur le registre officiel Docker, vous pouvez trouver des images de distribution Linux ou de logiciels. Ces images sont fournies officiellement par des entités (Fondation Python, Debian, Ubuntu, etc.) ou construites par des particuliers.

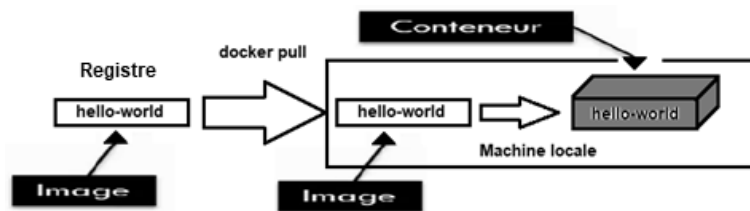
Un registre Docker permet aux utilisateurs d'extraire et d'insérer des images Docker dans un dépôt avec les autorisations d'accès appropriées. La même image peut avoir plusieurs versions différentes, identifiées par leurs tags.

Par défaut, le moteur Docker interagit avec le DockerHub. Ce dernier fournit un registre gratuit, des fonctionnalités supplémentaires telles que les comptes d'organisation et une intégration à des solutions de contrôle de source (GitHub, Bitbucket, etc.).

Vous pouvez utiliser DockerHub pour créer et télécharger des images, mais celles-ci deviennent alors publiques et tout le monde peut accéder à vos images et les utiliser.

Il est donc recommandé d'utiliser un registre privé Docker qui vous permet de contrôler et de protéger vos images, ou encore d'utiliser des services clés en main disponibles sur Azure ou AWS (*Amazon Web Services*) pour constituer vos hubs privés.

**Figure 1-9**  
Image et conteneur Docker



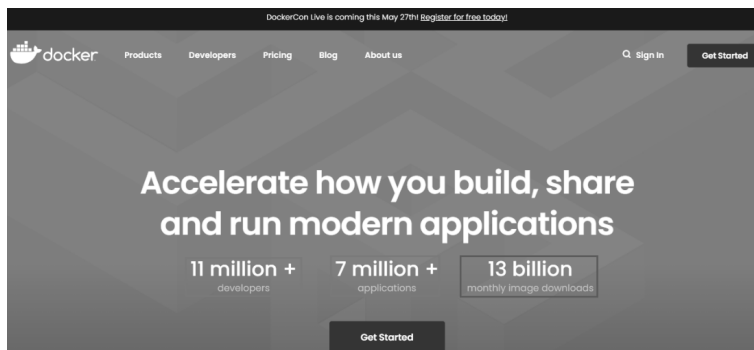
Un conteneur Docker est donc une instance exécutable d'une image. En utilisant l'API ou la CLI de Docker, nous pouvons créer, démarrer, arrêter, déplacer ou supprimer un conteneur.

Rendez-vous sur le site <https://www.docker.com/>, utilisé par des millions de développeurs (voir figure 1-10).

Aujourd'hui, tous les principaux fournisseurs de clouds et les principaux frameworks open source utilisent cette plate-forme. Beaucoup utilisent Docker pour leurs offres IaaS natives pour les conteneurs.

**Figure 1–10**

La plate-forme Docker



Cliquez sur *Pricing* dans le menu afin de consulter les offres disponibles. Les abonnements Docker se déclinent en trois options.

- Plans gratuits pour les individus et les équipes de développement qui incluent des dépôts publics illimités.
- Abonnements Docker Pro pour les particuliers avec des dépôts publics et privés illimités, aucune limitation de taux sur les requêtes pull et un support Premium avec un abonnement annuel.
- Abonnements Docker Team pour les équipes de développement avec des dépôts publics et privés illimités avec un abonnement annuel. Le plan Team offre des outils de collaboration et de gestion avancés comme la gestion de l'organisation et de l'équipe avec des contrôles d'accès basés sur les rôles.

**Figure 1–11**

Les abonnements Docker

## Pricing & Subscriptions

Choose a plan that's right for you.

Free	Pro	Team	Large
FOR EVERYBODY	FOR INDIVIDUALS	FOR ORGANIZATIONS	FOR ORGANIZATIONS
<ul style="list-style-type: none"> <li>Unlimited public repositories</li> <li>Docker Desktop continuously updated</li> <li>Docker Desktop includes Docker Engine and Kubernetes</li> <li>Limited container image requests</li> <li>Two-factor authentication</li> </ul>	<ul style="list-style-type: none"> <li>Everything in Free</li> <li>Unlimited private repositories</li> <li>Unlimited container image requests</li> <li>2 parallel builds</li> <li>300 monthly Hub image vulnerability scans</li> <li>Premium customer support for Desktop and Hub</li> </ul>	<ul style="list-style-type: none"> <li>Everything in Pro</li> <li>Unlimited teams</li> <li>Unlimited monthly Hub image vulnerability scans</li> <li>3 parallel builds per org</li> <li>Role-based access control</li> <li>Audit log</li> <li>First five members is \$25, \$7 per additional team member</li> </ul>	<ul style="list-style-type: none"> <li>Everything in Team</li> <li>Whitelist service up to 20 IPs</li> <li>Invoicing available</li> <li>Minimum 500 users</li> <li>\$84/team seat per year</li> <li>Governed by Terms of Service</li> </ul>
<b>\$0</b> /month	<b>\$5</b> /month With annual plan	<b>\$7</b> user/month With annual plan	
<a href="#">Signup for Free</a>	<a href="#">Buy Now</a>	<a href="#">Buy Now</a>	<a href="#">Contact Sales</a>

## Installation de Docker

Pour les opérations qui vont suivre, j'utiliserai une machine Ubuntu 20.04 LTS, sur laquelle j'installerai Docker et je ferai toutes les démonstrations. Les commandes Docker restent valides même si vous utilisez une autre distribution Linux.

Commencez par mettre à jour votre liste de paquets :

```
elyes@ubuntu:~$ sudo apt update
```

## Installation depuis les dépôts officiels Ubuntu

Cette première méthode consiste à installer Docker depuis le référentiel officiel Ubuntu. Tapez la commande suivante :

```
elyes@ubuntu:~$ sudo apt install docker.io
```

Vérifiez la version de Docker installée :

```
elyes@ubuntu:~$ docker --version
```

```
Docker version 19.03.8, build afac8b7f0
```

## Installation à partir du référentiel officiel de Docker

Pour installer la version la plus récente de Docker, nous utiliserons le référentiel officiel de Docker car il se peut que le package d'installation de Docker disponible dans le référentiel officiel Ubuntu ne soit pas la version la plus récente.

Pour ce faire, nous allons ajouter une nouvelle source de paquet. Pour garantir la validité des téléchargements, ajoutez la clé GPG de Docker, puis installez le paquet.

Tout d'abord, installez quelques paquets prérequis qui permettent à apt d'utiliser les paquets via HTTPS :

```
elyes@ubuntu:~$ sudo apt install apt-transport-https ca-certificates curl  
software-properties-common
```

Ensuite, ajoutez la clé GPG du référentiel Docker officiel à votre système :

```
elyes@ubuntu:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg |  
sudo apt-key add -
```

```
OK
```

Maintenant, ajoutez le référentiel Docker aux sources APT :

```
elyes@ubuntu:~$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu focal stable"
```

Ensuite, mettez à jour la base de données des paquets avec les paquets Docker du référentiel que vous avez ajouté :

```
elyes@ubuntu:~$ sudo apt update
```

Pour vous assurer que vous êtes sur le point d'installer à partir du référentiel Docker au lieu du référentiel par défaut Ubuntu, tapez la commande suivante :

```
elyes@ubuntu:~$ apt-cache policy docker-ce
```

Vous verrez une sortie de données comme celle-ci (à noter que le numéro de version de Docker peut être différent) :

```
docker-ce:
  Installed: (none)
  Candidate: 5:20.10.6~3-0~ubuntu-focal
  Version table:
 *** 5:20.10.6~3-0~ubuntu-focal 500
```

Vous voyez que `docker-ce` n'est pas encore installé, mais que le candidat à l'installation vient du référentiel Docker pour Ubuntu.

Installez alors Docker à l'aide de cette commande :

```
elyes@ubuntu:~$ sudo apt install docker-ce
```

Vérifiez maintenant la version de Docker installée :

```
elyes@ubuntu:~$ docker --version
```

```
Docker version 20.10.6, build 370c289
```

Comme vous pouvez le constater, cette version est plus récente que celle installée à partir des dépôts officiels d'Ubuntu.

Vérifiez ensuite que Docker est en cours d'exécution :

```
elyes@ubuntu:~$ sudo systemctl status docker
```

```
? docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset>
   Active: active (running) since Thu 2021-10-07 08:02:35 PDT; 6min ago
```

```
TriggeredBy: ? docker.socket
Docs: https://docs.docker.com
Main PID: 1222 (dockerd)
Tasks: 13
Memory: 140.7M
```

Ajoutez votre nom d'utilisateur au groupe `docker` pour éviter de taper `sudo` chaque fois que vous exécutez la commande `docker`.

En fait, la commande `docker` ne peut être exécutée que par l'utilisateur `root` ou par un utilisateur du groupe `docker` créé automatiquement lors du processus d'installation de Docker :

```
elyes@ubuntu:~$ sudo usermod -aG docker ${USER}
```

Pour appliquer la nouvelle appartenance à un groupe et éviter la déconnexion et la reconnexion à la machine, tapez ce qui suit :

```
elyes@ubuntu:~$ su - ${USER}
```

Confirmez que votre utilisateur est maintenant ajouté au groupe `docker` en tapant :

```
elyes@ubuntu:~$ id -nG
```

```
elyes adm cdrom sudo dip plugdev lpadmin lxd sambashare docker
```

## Utilisation de la commande `docker`

Utiliser la commande `docker` consiste à lui transmettre une chaîne d'options et de commandes suivie d'arguments. La syntaxe prend cette forme :

```
Usage: docker [OPTIONS] COMMAND
```

Le tableau 1-1 présente quelques commandes de base ainsi que leurs rôles.

**Tableau 1-1** Commandes Docker de base

Commandes	Rôles
<code>docker ps</code>	Visualiser les conteneurs actifs
<code>docker ps -a</code>	Visualiser tous les conteneurs
<code>docker rm [container]</code>	Supprimer un conteneur inactif
<code>docker rm -f [container]</code>	Forcer la suppression d'un conteneur actif
<code>docker images</code>	Lister les images existantes
<code>docker rmi [image]</code>	Supprimer une image Docker
<code>docker exec -t -i [container] /bin/bash</code>	Exécuter des commandes dans un conteneur actif
<code>docker inspect [container]</code>	Inspecter la configuration d'un conteneur

Tableau 1-1 Commandes Docker de base (suite)

Commandes	Rôles
<code>docker build -t [image] .</code>	Construire une image à partir d'un Dockerfile
<code>docker history [image]</code>	Afficher l'historique d'une image
<code>docker logs --tail 5 [container]</code>	Visualiser les logs d'un conteneur (les 5 dernières lignes)
<code>docker login</code>	Se connecter au registre
<code>docker search [name]</code>	Rechercher une image
<code>docker pull [image]</code>	Récupérer une image
<code>docker push [image]</code>	Pousser une image du cache local au registre
<code>docker tag [UUID] [image]:[tag]</code>	Taguer une image

La commande `docker` permet d'afficher toutes les sous-commandes disponibles :

```
eIyes@ubuntu:~$ docker

Management Commands:
  app*      Docker App (Docker Inc., v0.9.1-beta3)
  builder   Manage builds
  buildx*   Build with BuildKit (Docker Inc., v0.5.1-docker)
  config    Manage Docker configs
  container Manage containers
  context   Manage contexts
  image     Manage images
  manifest  Manage Docker image manifests and manifest lists
  network   Manage networks
  node      Manage Swarm nodes
  plugin    Manage plugins
  scan*     Docker Scan (Docker Inc., v0.7.0)
  secret    Manage Docker secrets
  service   Manage services
  stack     Manage Docker stacks
  swarm     Manage Swarm
  system    Manage Docker
  trust     Manage trust on Docker images
  volume    Manage volumes
```

Voici la liste complète de toutes les commandes Docker disponibles :

```
Commands:
  attach    Attach local standard input, output, and error streams to a running
            container
  build     Build an image from a Dockerfile
  commit    Create a new image from a container's changes
  cp        Copy files/folders between a container and the local filesystem
  create    Create a new container
  diff      Inspect changes to files or directories on a container's filesystem
```



events	Get real time events from the server
exec	Run a command in a running container
export	Export a container's filesystem as a tar archive
history	Show the history of an image
images	List images
import	Import the contents from a tarball to create a filesystem image
info	Display system-wide information
inspect	Return low-level information on Docker objects
kill	Kill one or more running containers
load	Load an image from a tar archive or STDIN
login	Log in to a Docker registry
logout	Log out from a Docker registry
logs	Fetch the logs of a container
pause	Pause all processes within one or more containers
port	List port mappings or a specific mapping for the container
ps	List containers
pull	Pull an image or a repository from a registry
push	Push an image or a repository to a registry
rename	Rename a container
restart	Restart one or more containers
rm	Remove one or more containers
rmi	Remove one or more images
run	Run a command in a new container
save	Save one or more images to a tar archive (streamed to STDOUT by default)
search	Search the Docker Hub for images
start	Start one or more stopped containers
stats	Display a live stream of container(s) resource usage statistics
stop	Stop one or more running containers
tag	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
top	Display the running processes of a container
unpause	Unpause all processes within one or more containers
update	Update configuration of one or more containers
version	Show the Docker version information
wait	Block until one or more containers stop, then print their exit codes

Pour afficher les options disponibles pour une commande spécifique, tapez :

```
docker COMMAND --help
```

Pour afficher des informations sur Docker, tapez :

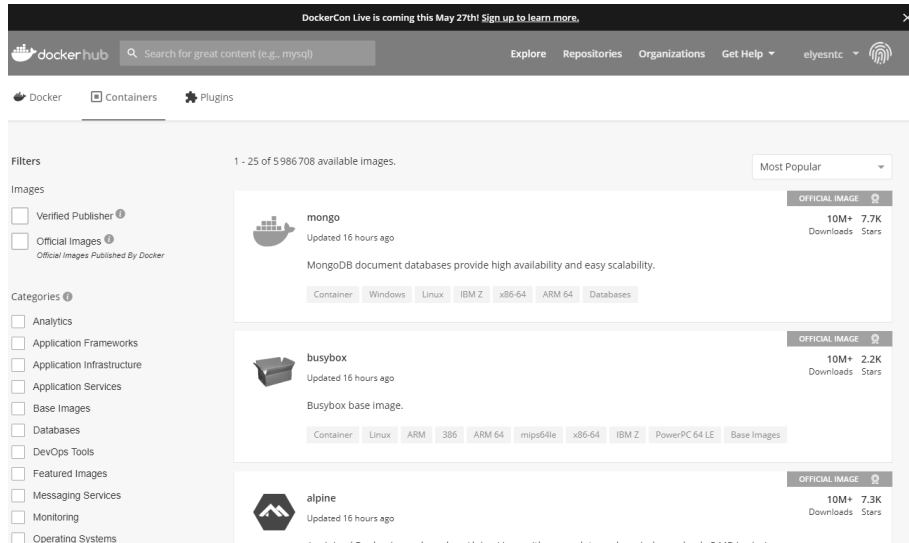
```
elyes@ubuntu:~$ docker info
```

## Utilisation des images Docker

Les conteneurs Docker sont construits à partir d'images Docker. Par défaut, Docker extrait ces images de Docker Hub (Docker Inc, 2021), un registre de conteneurs géré par Docker, la société à l'origine du projet Docker.

Rendez-vous à cette adresse pour consulter les images disponibles : <https://hub.docker.com/>.

**Figure 1–12**  
Docker Hub



Tout le monde peut héberger ses images Docker sur Docker Hub. Ainsi, la plupart des applications et des distributions Linux dont vous aurez besoin auront des images hébergées sur cet espace.

Pour vérifier si vous pouvez accéder aux images et les télécharger à partir de Docker Hub, tapez :

```
elyes@ubuntu:~$ docker run hello-world
```

Le message `Hello from Docker !` indiquera que Docker fonctionne correctement :

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:9ade9cc2e26189a19c2e8854b9c8f1e14829b51c55a630ee675a5a9540ef6ccf
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

Docker a téléchargé l'image à partir de Docker Hub (le référentiel par défaut) car il n'a pas pu trouver initialement l'image `hello-world` localement.

Une fois l'image téléchargée, Docker a créé un conteneur à partir de l'image et de l'application exécutée dans le conteneur. C'est un exemple de conteneur qui s'exécute et se ferme après avoir émis un message de test.

Pour rechercher des images disponibles sur Docker Hub vous pouvez utiliser la commande `docker` avec la sous-commande `search`.

Par exemple, pour rechercher l'image Ubuntu, tapez :

```
elyes@ubuntu:~$ docker search ubuntu
```

Le script analysera Docker Hub et renverra une liste de toutes les images dont le nom correspond à la chaîne de recherche.

Dans la colonne **OFFICIAL**, **OK** indique une image construite et prise en charge par la société derrière le projet.

Figure 1-13

NAME	DESCRIPTION	STARS	OFFICIAL
<b>ubuntu</b>	Ubuntu is a Debian-based Linux operating sys...	<b>12915</b>	<b>[OK]</b>
dorowu/ubuntu-desktop-lxde-vnc	Docker image to provide HTML5 VNC interface ...	576	
websphere-liberty	WebSphere Liberty multi-architecture images ...	280	[OK]
rastasheep/ubuntu-sshd	Dockerized SSH service, built on top of offi...	255	
consol/ubuntu-xfce-vnc	Ubuntu container with "headless" VNC session...	242	
ubuntu-upstart	DEPRECATED, as is Upstart (find other proces...	113	[OK]

La commande **pull** permet de télécharger sur votre ordinateur l'image que vous souhaitez utiliser. Exécutez la commande suivante pour télécharger l'image officielle Ubuntu sur votre ordinateur :

```
elyes@ubuntu:~$ docker pull ubuntu
```

Notez que si vous ne spécifiez aucun tag, Docker utilisera le tag par défaut **latest**. La dernière version de l'image sera donc téléchargée.

```
Using default tag: latest
latest: Pulling from library/ubuntu
f3ef4ff62e0d: Pull complete
Digest: sha256:a0d9e826ab87bd665cfc640598a871b748b4b70a01a4f3d174d4fb02adad07a9
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

La sous-commande **run** permet d'exécuter un conteneur en utilisant l'image téléchargée.

Comme vous l'avez vu avec l'exemple **hello-world**, si une image n'a pas été téléchargée lorsque Docker est exécuté avec la sous-commande **run**, le client Docker télécharge d'abord l'image, puis exécute un conteneur en l'utilisant.

Pour voir les images téléchargées sur votre ordinateur, tapez :

```
elyes@ubuntu:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	597ce1600cf4	6 days ago	72.8MB
hello-world	latest	feb5d9fea6a5	13 days ago	13.3kB

Notez la taille réduite de l'image Ubuntu.

Les images que vous utilisez pour exécuter des conteneurs peuvent être modifiées et utilisées pour générer de nouvelles images, qui peuvent ensuite être transmises (*pushed* est le terme technique) vers Docker Hub ou d'autres registres Docker.

## Exécuter un conteneur Docker

Les conteneurs ressemblent aux machines virtuelles, mais ils sont plus conviviaux. Exécutez comme exemple un conteneur en utilisant la dernière image d'Ubuntu.

La combinaison des options `-i` et `-t` vous donne un accès interactif au shell dans le conteneur :

```
iset@ubuntu:~$ docker run -it --name ubuntu-nginx ubuntu
root@98bd6d05c235:/#
```

Vous travaillez maintenant à l'intérieur du conteneur. Notez l'ID de conteneur dans l'invite de commande. Dans cet exemple, il s'agit de `98bd6d05c235`.

Vous pouvez maintenant exécuter n'importe quelle commande dans le conteneur.

Par exemple, mettez à jour la base de données de paquets à l'intérieur du conteneur en utilisant la commande `apt update` et installez ensuite le serveur web `nginx` :

```
root@98bd6d05c235:/# apt install nginx
```

Notez l'adresse IP du conteneur et démarrez Nginx.

```
root@98bd6d05c235:/# ip a s
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
6: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
```

```
root@98bd6d05c235:/# service nginx start
```

```
* Starting nginx nginx
```

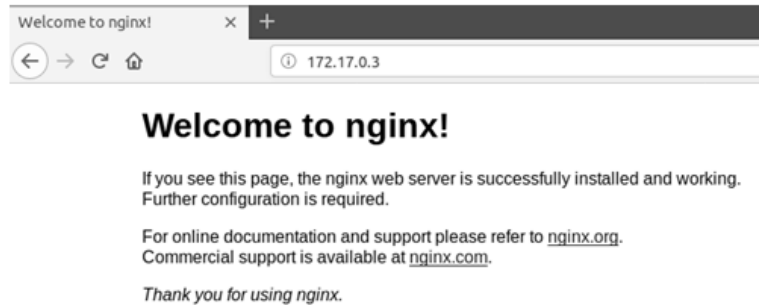
```
[ OK ]
```

```
root@98bd6d05c235:/# ss -anltp
```

State	Recv-Q	Send-Q	Local Address:Port
LISTEN	0	511	0.0.0.0:80
LISTEN	0	511	:::80

Saisissez cette adresse dans votre navigateur web et vérifiez que Nginx fonctionne correctement (voir figure 1-14).

Figure 1-14



Vous pouvez quitter ce conteneur, le démarrer et exécuter le shell bash à nouveau sur ce conteneur :

```
root@98bd6d05c235:/# exit
```

```
exit
```

```
elyes@ubuntu:~$ docker start ubuntu-nginx
```

```
ubuntu-nginx
```

```
elyes@ubuntu:~$ docker exec ubuntu-nginx /bin/bash
elyes@ubuntu:~$ docker exec -it ubuntu-nginx /bin/bash
```

```
root@98bd6d05c235:/#
```

## Publiez votre première image Docker sur Docker Hub

Il peut être utile de valider les modifications ou les paramètres d'un fichier de conteneur dans une nouvelle image à l'aide de la commande `docker commit`.

```
root@98bd6d05c235:/# exit
```

```
exit
```

```
elyes@ubuntu:~$ docker commit ubuntu-nginx elyesntc/ubuntu-nginx:1.0
```

```
sha256:05834c35c335bc10f90a780e84a01bcbc04c3be6c0fd467e26a37f036241cea6
```

```
elyes@ubuntu:~$ docker login
```

```
Login with your Docker ID to push and pull images from Docker Hub. If you don't have
a Docker ID, head over to https://hub.docker.com to create one.
```

```
Username: elyesntc
Password:
WARNING! Your password will be stored unencrypted in /home/elyes/.docker/
config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

## Publier l'image

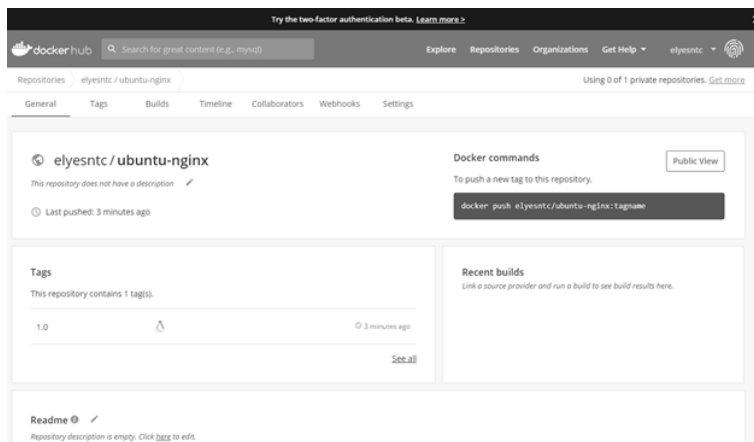
Mettez votre image taguée dans le référentiel à l'aide de la commande `docker push`.

```
elyes@ubuntu:~$ docker push elyesntc/ubuntu-nginx:1.0
```

```
The push refers to repository [docker.io/elyesntc/ubuntu-nginx]
E3d47dc27759: Pushed
16542a8fc3be: Mounted from elyesntc/ubuntu_nginx
6597da2e2e52: Mounted from elyesntc/ubuntu_nginx
977183d4e999: Mounted from elyesntc/ubuntu_nginx
c8be1b8f4d60: Mounted from elyesntc/ubuntu_nginx
1.0: digest: sha256:2163adb7f02a8855625478ef0a7dc5a5b2ad5d8ce75eb92db4e2166eaaa76a98
size: 1364
```

Une fois terminé, les résultats de ce dépôt sont accessibles au public. Si vous vous connectez à Docker Hub, vous y verrez la nouvelle image, avec sa commande `pull` (voir figure 1-15).

Figure 1-15



### Remarque

Docker Hub héberge des millions d'images de conteneurs et il contient déjà une image Nginx prête. Vous pouvez l'utiliser directement.

Il y a aussi de très nombreuses images officielles et personnelles créées par différents membres de la communauté, et même par certains éditeurs de solutions open source.

```
elyes@ubuntu:~$ docker run -itd --name nginx-test -p 8080:80 nginx
```

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
07aded7c29c6: Pull complete
bbe0b7acc89c: Pull complete
44ac32b0bba8: Pull complete
91d6e3e593db: Pull complete
8700267f2376: Pull complete
4ce73aa6e9b0: Pull complete
Digest: sha256:06e4235e95299b1d6d595c5ef4c41a9b12641f6683136c18394b858967cd1506
Status: Downloaded newer image for nginx:latest
46c876ccf93d27d089305d9707f26c268bc61e451e51c796dfb05bff23f9f1d8
```

Repérez l'adresse IP de la machine hôte et testez le mappage de port vers le conteneur.

```
elyes@ubuntu:~$ ip a s
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 00:0c:29:6a:9d:70 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.4/24 brd 192.168.1.255 scope global dynamic noprefixroute ens33
        valid_lft 81393sec preferred_lft 81393sec
    inet6 fe80::adc4:dd00:8956:99c4/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

Figure 1–16



# Gestion du réseau sous Docker

## Les types de réseaux

Lors de l'installation de Docker, trois réseaux sont créés automatiquement. On peut voir ces réseaux avec la commande `docker network ls`.

```
elyes@ubuntu:~$ docker network ls
```

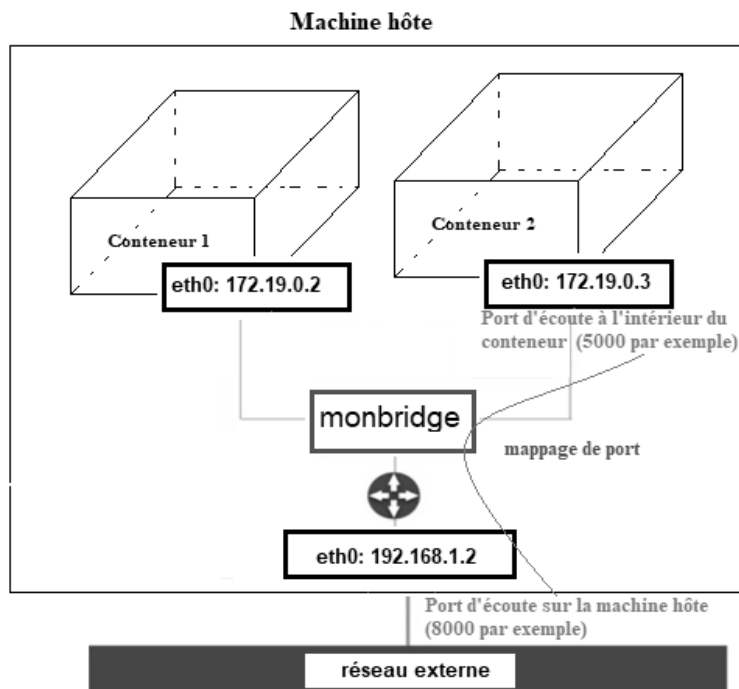
NETWORK ID	NAME	DRIVER	SCOPE
3a47f3a09bf9	bridge	bridge	local
0b7ab4520aaf	host	host	local
a082ea7426a3	none	null	local

Le driver `bridge` est présent sur tous les hôtes Docker. Par ailleurs, le réseau `bridge` est le type de réseau le plus couramment utilisé.

Il est limité aux conteneurs d'un hôte unique exécutant le moteur Docker. Les conteneurs qui utilisent ce driver ne sont pas accessibles depuis l'extérieur et ne peuvent communiquer qu'entre eux.

**Figure 1-17**

Le driver bridge





Lors de la création d'un conteneur, si l'on ne spécifie pas un réseau particulier, les conteneurs sont connectés au réseau `bridge` `docker0`.

```
elyes@ubuntu:~$ ip addr show docker0
```

```
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group
default
    link/ether 02:42:c7:54:e9:91 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
    inet6 fe80::42:c7ff:fe54:e991/64 scope link
        valid_lft forever preferred_lft forever
```

La commande `docker network inspect bridge` retourne les informations concernant ce réseau :

```
elyes@ubuntu:~$ docker network inspect bridge
```

```
[
  {
    "Name": "bridge",
    "Id": "4ec087dc338e80b2a7d618ead21faf3d2c97b29b4a61bbea7e8c59ecceb7e98",
    "Created": "2021-10-07T01:34:38.395917633-07:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    }
  },
]
```

```

    "Labels": {}
  }
]

```

Créez deux conteneurs à partir de l'image Ubuntu.

```
elyes@ubuntu:~$ docker run -itd --name container1 ubuntu
```

```
4d98f8e3d0976aaca6b73e4893c68f17ce22069bdbd9a5ce0fc916d835efb3e0
```

```
elyes@ubuntu:~$ docker run -itd --name container2 ubuntu
```

```
619b5a3f2611849c337df922269f3f0077d5157c9ec2c5a9739f22f6ac91ad8e
```

Et visualisez les informations du réseau avec `docker network inspect bridge` :

```

"Containers": {
  "6b4a42d0d7ba964c5f8ab66b3552ab81c90f05f6e3debf2c660415ccb4679f8c": {
    "Name": "container2",
    "EndpointID":
"6fc393c8bc4b003c4c9501ee5f9c11a7c589ccc102f484d6ee363369f276ddc7",
    "MacAddress": "02:42:ac:11:00:03",
    "IPv4Address": "172.17.0.3/16",
    "IPv6Address": ""
  },
  "dabacd7807b210e6e032f5bfaa989a58c0c15989ab53296502b0651a43d4d3a92": {
    "Name": "container1",
    "EndpointID":
"c2779e7afef417ab05a2549069ee1424e1099e08eb93494cd38e80b6421ed3f3",
    "MacAddress": "02:42:ac:11:00:02",
    "IPv4Address": "172.17.0.2/16",
    "IPv6Address": ""
  }
},

```

Notez les adresses des deux conteneurs et effectuez un ping sur chacune d'elles :

```
elyes@ubuntu:~$ ping 172.17.0.2
```

```

PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.040 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.096 ms
^C
--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.040/0.068/0.096/0.028 ms

```

```
elyes@ubuntu:~$ ping 172.17.0.3
```

```

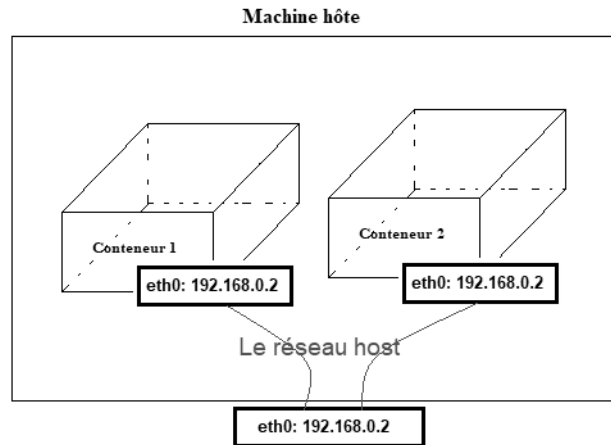
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.117 ms

```

```
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.098 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.098 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.101 ms
^C
--- 172.17.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3055ms
rtt min/avg/max/mdev = 0.098/0.103/0.117/0.007 ms
```

- Le driver `none` permet d'interdire toute communication (interne et externe) avec votre conteneur puisque ce dernier sera dépourvu de toute interface réseau mis à part l'interface `loopback`.
- Le driver `host` permet aux conteneurs d'utiliser la même interface réseau que l'hôte. Un conteneur lié à ce type de réseau prendra donc la même adresse IP que la machine hôte. Les conteneurs seront par défaut accessibles de l'extérieur et il n'y a pas d'isolation réseau entre eux.

**Figure 1-18**  
Le driver `host`



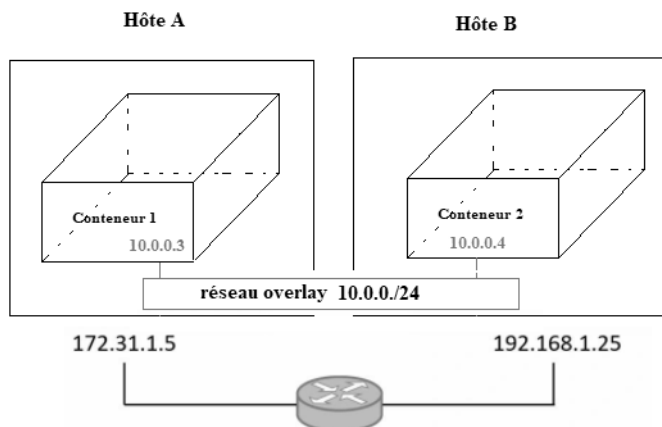
À part ces trois réseaux créés à l'installation, il existe d'autres drivers que vous pouvez utiliser comme `overlay` et `macvlan`.

- Le driver `overlay` : ce réseau se trouve au-dessus des réseaux spécifiques à l'hôte (superpositions), ce qui permet aux conteneurs qui y sont connectés de communiquer en toute sécurité lorsque le chiffrement est activé.

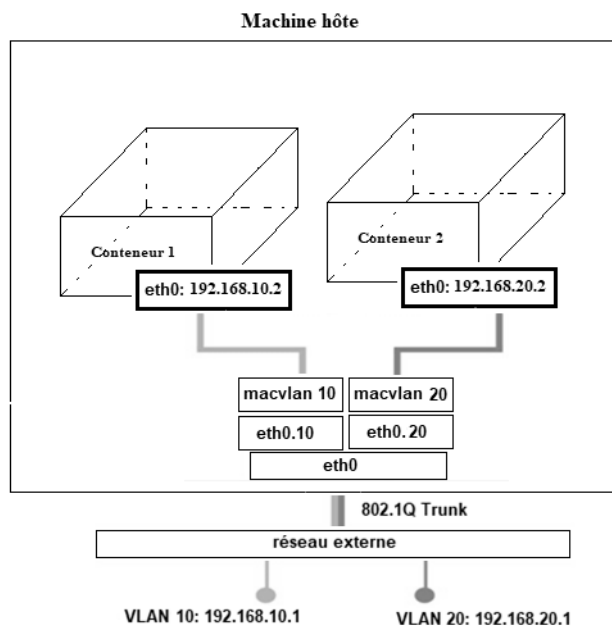
Docker gère de manière transparente le routage de chaque paquet vers et depuis le bon hôte et le bon conteneur (figure 1-19).

- Le driver `macvlan` permet d'attribuer une adresse MAC à un conteneur, le faisant apparaître comme un périphérique physique sur votre réseau. Le moteur Docker route donc le trafic vers les conteneurs en fonction de leurs adresses MAC. Il est utile dans le cas de l'utilisation des applications qui s'attendent à être directement connectées au réseau physique.

**Figure 1–19**  
Le driver overlay



**Figure 1–20**  
Le driver macvlan



## Réseaux définis par l'utilisateur

Les conteneurs sont connectés au réseau `bridge docker0` par défaut et peuvent communiquer entre eux par adresses IP. Ils se trouvent alors sur le même réseau.

Afin que les conteneurs puissent résoudre leurs noms en adresses IP, on crée des réseaux définis par l'utilisateur.

Docker exécute un serveur DNS intégré qui fournit une résolution de noms aux conteneurs connectés au réseau créé par les utilisateurs, de sorte que ces conteneurs peuvent résoudre les noms d'hôtes en adresses IP. Utiliser des réseaux `bridge` définis par l'utilisateur permet aussi de contrôler quels conteneurs peuvent communiquer entre eux.

Docker fournit des pilotes pour créer des réseaux :

- `bridge` ;
- `overlay` (multi host) ;
- `macvlan`.

Un réseau `bridge` est le type de réseau le plus utilisé dans Docker. Les réseaux `bridge` créés par les utilisateurs sont semblables au réseau `bridge` par défaut créé à l'installation de Docker. Cependant, de nouvelles fonctionnalités peuvent être ajoutées, telles que la gestion du DNS.

Lors de la création d'un nouveau réseau, une nouvelle interface est également créée.

### Création d'un réseau de type `bridge` nommé « `monbridge` »

Utilisez la commande `docker network create` en spécifiant le type du réseau avec l'option `--driver` pour créer un réseau que vous nommerez `monbridge`.

Ensuite, inspectez ce réseau et notez la plage réseau et la passerelle par défaut.

```
elyes@ubuntu:~$ docker network create --driver bridge monbridge
```

```
b736b9c8c2ab7363329f7ddd59c9f5149695739d221ce86de1bdfa2f78b744ad
```

```
elyes@ubuntu:~$ docker network inspect monbridge
```

```
[
  {
    "Name": "monbridge",
    "Id": "b736b9c8c2ab7363329f7ddd59c9f5149695739d221ce86de1bdfa2f78b744ad",
    "Created": "2021-10-07T08:41:30.33884955-07:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
```

```

    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]

```

Affichez la liste des réseaux et vérifiez l'ajout du réseau `monbridge` dans cette liste.

```
elyes@ubuntu:~$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
3a47f3a09bf9	bridge	bridge	local
0b7ab4520aaf	host	host	local
b736b9c8c2ab	monbridge	bridge	local
a082ea7426a3	none	null	local

Connexion d'un premier conteneur nommé `conteneur1` au réseau `monbridge` directement avec l'option `--network` :

```
elyes@ubuntu:~$ docker run --network monbridge -itd --name conteneur1 ubuntu
ed130ed05d04f9c9f1c0872266a224de8b003e218b128bacee49bbfa067a766a
```

Connexion d'un deuxième conteneur nommé `conteneur2` au réseau `monbridge` après sa création avec la commande `docker network connect` :

```
elyes@ubuntu:~$ docker run -itd --name conteneur2 ubuntu
592a4fe9d8fee879f98b9b7c170f5e2bf986518e4dcf87e2172f18e037b5037e
elyes@ubuntu:~$ docker network connect monbridge conteneur2
```

Listez les conteneurs en cours d'exécution.

```
elyes@ubuntu:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
592a4fe9d8fe	ubuntu	"bash"	33 seconds ago	Up 32 seconds		conteneur2
ed130ed05d04	ubuntu	"bash"	About a minute ago	Up About a minute		conteneur1
6b4a42d0d7ba	ubuntu	"bash"	9 minutes ago	Up 9 minutes		container2
dabcd7807b21	ubuntu	"bash"	9 minutes ago	Up 9 minutes		container1

Maintenant, nous allons nous connecter au `conteneur1` et vérifier le fonctionnement du DNS.

Il est possible d'installer le paquet `dnsutils` contenant la commande d'interrogation du service DNS `nslookup`.

```
elyes@ubuntu:~$ docker exec -it ed130ed05d04 bash
root@ed130ed05d04:/# apt install dnsutils
```

## Test du serveur DNS intégré

Si le serveur DNS intégré est incapable de résoudre la demande, il sera transmis à tous les serveurs DNS externes configurés pour le conteneur.

Pour faciliter cela, lorsque le conteneur est créé, seul le serveur DNS intégré 127.0.0.11 est renseigné dans le fichier `resolv.conf` du conteneur.

```
root@ed130ed05d04:/# nslookup conteneur2
```

```
Server: 127.0.0.11
Address: 127.0.0.11#53

Non-authoritative answer:
Name: conteneur2
Address: 172.18.0.3
```

## Docker sous Windows

### Docker Toolbox

Cet outil est destiné aux anciens systèmes macOS et Windows qui ne répondent pas aux exigences de Docker Desktop pour macOS et Windows. Étant donné que le démon Docker Engine utilise des fonctionnalités de noyau spécifiques à Linux, vous ne pouvez pas exécuter Docker Engine en mode natif sous Windows.

Vous devez alors utiliser la commande `docker-machine` pour créer et vous connecter à une machine virtuelle Linux sur votre machine qui héberge Docker Engine sur votre système Windows.

### Docker Desktop

C'est le meilleur moyen de démarrer avec Docker sous Windows. L'image du conteneur se compose des fichiers du système d'exploitation en mode utilisateur nécessaires pour prendre en charge votre application, les exécutions ou les dépendances de votre application et tout autre fichier de configuration dont votre application a besoin pour fonctionner correctement.

Microsoft propose plusieurs images (appelées « images de base ») que vous pouvez utiliser comme point de départ pour créer votre propre image de conteneur (Microsoft, 2021).

- Windows : contient l'ensemble complet des API Windows et des services système (sans les rôles de serveur).
- Windows Server Core : image plus petite qui contient un sous-ensemble des API Windows Server, à savoir le framework .NET complet. Il inclut également la plupart des rôles de serveur, bien que malheureusement peu nombreux, et pas le serveur de télécopie.
- Nano Server : il s'agit de la plus petite image Windows Server, avec prise en charge des API .NET Core et de certains rôles de serveur.

- Windows 10 IoT Core : version de Windows utilisée par les fabricants de matériel pour les petits appareils utilisés dans l'Internet des objets qui exécutent des processeurs ARM ou x86/x64.

En guise d'image de base, Microsoft recommande le Nano Server.

Windows est actuellement la seule plate-forme permettant d'exécuter aussi bien des conteneurs Windows que Linux grâce à la virtualisation proposée par Hyper-V. Il est néanmoins nécessaire de choisir entre les conteneurs Linux ou les conteneurs Windows.

## Docker pour Windows Server

Depuis Windows Server 2016, Windows dispose de sa propre technologie de conteneurs basée sur Docker à laquelle Microsoft ajoute la prise en charge de la ligne de commande PowerShell et permet davantage d'isolation grâce à Nano Server et Hyper-V Containers.

Les conteneurs Windows peuvent être installés via Windows Features Dialogue ou PowerShell.

Il sera aussi nécessaire d'activer la virtualisation Hyper-V si vous souhaitez utiliser les conteneurs Hyper-V.

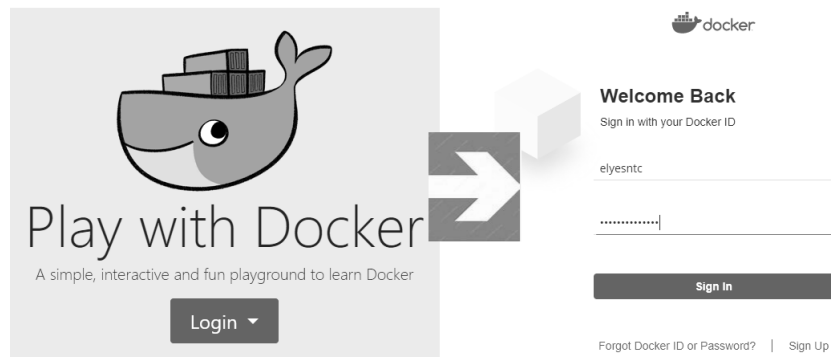
## PWD

PWD (*Play With Docker*), connu également sous le nom de Docker Playground, est un service gratuit qui permet de créer un cluster de VM prêtes pour utiliser Docker. Il est possible de créer jusqu'à cinq instances.

Le cluster a une durée de vie limitée à quatre heures. Ce service est vraiment utile pour tester des images Docker ou bien pour des ateliers ou des formations.

Créez un compte Docker et utilisez-le pour vous connecter au site <https://labs.play-with-docker.com/>.

**Figure 1-21**  
PWD (Play with Docker)



Créez une nouvelle instance et commencez directement à utiliser Docker qui est déjà installé.



Cette plate-forme vous permet d'avoir une machine virtuelle Alpine Linux gratuite dans le navigateur, où vous pouvez créer et exécuter des conteneurs Docker et même créer des clusters en mode Docker Swarm.

Elle comprend également un site de formation composé d'un grand ensemble de Tps Docker et de quiz du niveau débutant au niveau avancé, disponibles en cliquant sur le lien <https://training.play-with-docker.com/>.

**Figure 1-22**

Créer une instance sous le service PWD

The screenshot shows the Play with Docker web interface. On the left, a sidebar contains a clock showing 03:57:53, a 'CLOSE SESSION' button, an 'Instances' section with a wrench and gear icon, and a '+ ADD NEW INSTANCE' button. Below this, a list of instances shows one instance with IP 192.168.0.18 and name 'node1'. The main panel displays details for a new instance named 'c21k3kfn\_c21k3svqf8u000fj7arg'. It shows the IP 192.168.0.18, an 'OPEN PORT' button, memory usage at 3.19% (127.7MiB / 3.906GiB), CPU usage at 0.03%, and an SSH command: 'ssh ip172-18-0-72-c21k3kfnjsv000cb6gg0@direct.labs.play-'. There are 'DELETE' and 'EDITOR' buttons. Below this is a terminal window showing a warning about user responsibility, the PWD team notice, and a successful 'docker pull ubuntu' command execution.

```
# is HIGHLY! discouraged. Any consequences of doing so are #
# completely the user's responsibilities. #
# #
# The PWD team. #
#####
[node1] (local) root@192.168.0.18 ~
$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
a70d879fa598: Pull complete
c4394e92d1f8: Pull complete
10e6159c56c0: Pull complete
Digest: sha256:3c9c713e0979e9bd6061ed52ac1e9e1f246c9495aa063619d9d695fb8039aa1f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
[node1] (local) root@192.168.0.18 ~
$
```

## Conclusion

Dans cette partie, nous avons vu tous les aspects majeurs de Docker rendant son utilisation très simple, intéressante et puissante.

Docker est utilisé par des millions de professionnels de l'informatique dans le monde entier, et comprend la plus grande bibliothèque d'images de conteneurs provenant de grands fournisseurs de logiciels, de projets open source et de la communauté.

Docker apporte une valeur immédiate dans les entreprises, augmentant rapidement leur productivité, ce qui fournit de nouvelles expériences client allant des applications monolithiques traditionnelles aux applications cloud natives.

## Validation des acquis

### Questions :

- ① Quels sont les avantages d'utiliser Docker plutôt que la virtualisation ?
- ② Quelle est la différence entre Docker et un hyperviseur ?
- ③ Quels sont les composants de l'architecture Docker ?
- ④ Quel est le cycle de vie d'un conteneur Docker ?
- ⑤ Est-il possible de faire fonctionner des images créées avec Linux sur Windows, et réciproquement ?
- ⑥ Quels sont les principaux facteurs qui déterminent le nombre de conteneurs que vous pouvez exécuter ?
- ⑦ Une image Docker :
  - est la même chose qu'un conteneur Docker ;
  - sera modifiée si j'apporte des modifications à un conteneur lancé à partir de cette image ;
  - permet de créer plusieurs conteneurs identiques.
- ⑧ Le référentiel public d'images Docker est :
  - GitHub ;
  - Docker Hub ;
  - Docker Trusted Registry ;
  - Azure MarketPlace.
- ⑨ Pour instancier et exécuter un nouveau conteneur Docker, on utilise la commande :
  - `docker start` ;
  - `docker pull` ;
  - `docker run` ;
  - `docker new`.

**Réponses :****① Quels sont les avantages d'utiliser Docker plutôt que la virtualisation ?**

La technologie des conteneurs se rapproche de la virtualisation, tout en étant beaucoup plus légère puisque les conteneurs n'embarquent pas forcément un système d'exploitation et sont donc plus appropriés pour rendre les applications portables que les VM.

Avec Docker, si l'application fonctionne sur votre machine, vous serez certain qu'elle fonctionnera sur l'environnement de production. Cette portabilité donne même la possibilité de faire tourner des images Docker sur des objets connectés.

**② Quelle est la différence entre Docker et un hyperviseur ?**

Un hyperviseur nécessite un matériel important pour fonctionner correctement, tandis que Docker s'exécute sur le système d'exploitation réel. Cela explique le fait que Docker est très rapide et permet d'effectuer des tâches de manière fluide.

**③ Quels sont les composants de l'architecture Docker ?**

Docker fonctionne sur une architecture client-serveur. Le client Docker établit la communication avec le démon Docker.

Le client Docker et le démon peuvent s'exécuter sur le même système. Les différents types de composants Docker sont les suivants :

- Client Docker : il est le principal moyen par lequel de nombreux utilisateurs interagissent avec Docker. Lorsque vous utilisez des commandes telles que `docker run`, le client envoie ces commandes à `dockerd`, qui les exécute. La commande Docker utilise l'API Docker pour appeler les requêtes à exécuter.
- Hôte Docker : ce composant contient le démon Docker, les conteneurs et les images. Le démon Docker établit une connexion avec le registre.
- Registre : ce composant stockera les images Docker.

**④ Quel est le cycle de vie d'un conteneur Docker ?**

Le cycle de vie du conteneur Docker est le suivant :

- Création d'un conteneur.
- Exécution du conteneur Docker.
- Mise en pause du conteneur et remise en marche du conteneur.
- Démarrage, arrêt et redémarrage du conteneur.
- Destruction du conteneur.

**⑤ Est-il possible de faire fonctionner des images créées avec Linux sur Windows, et réciproquement ?**

Il n'est pas possible de faire fonctionner des images créées avec Linux sur Windows, et réciproquement car les conteneurs partagent le noyau du système de la machine hôte.

Cependant, les équipes de Docker travaillent activement avec les équipes de Microsoft pour pallier ce manque.

**⑥ Quels sont les principaux facteurs qui déterminent le nombre de conteneurs que vous pouvez exécuter ?**

Deux facteurs peuvent en principe limiter le nombre de conteneurs que vous pouvez exécuter : la taille de votre application et la puissance de votre processeur.

- ⑦ **Une image Docker :**
  - permet de créer plusieurs conteneurs identiques.
- ⑧ **Le référentiel public d'images Docker est :**
  - Docker Hub.
- ⑨ **Pour instancier et exécuter un nouveau conteneur Docker, on utilise la commande :**
  - `docker run`.

# 2

## Création et gestion d'images Docker

---

Il est important qu'une application soit en mesure de conserver des données dans un conteneur. Nous allons donc commencer par faire le point sur les différentes façons de persister des données avec Docker.

Nous verrons ensuite comment valider les modifications apportées à un conteneur en cours d'exécution pour créer une nouvelle image Docker et comment automatiser cette tâche en écrivant toutes les instructions dans un fichier Dockerfile.

Enfin, nous étudierons Docker Compose, outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services interconnectés.

### Persistance des données

Les conteneurs Docker ne stockent pas de données persistantes. Conçu à l'origine pour faciliter le déploiement d'applications sans état, Docker est de plus en plus utilisé pour des applications ayant besoin de stocker des données de façon persistante.

Cependant, les données écrites dans la couche inscriptible d'un conteneur ne seront plus disponibles une fois que le conteneur aura cessé de fonctionner.

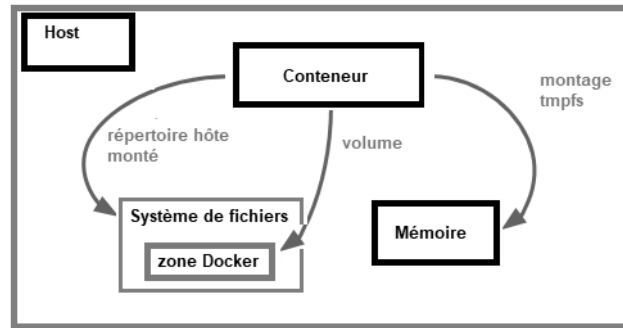
En fait, lorsqu'une image Docker est exécutée, le Docker Engine crée un système de fichiers temporaire sur lequel est stocké l'ensemble des composants et des données générées par le conteneur. En outre, il peut être difficile d'obtenir des données écrites dans un conteneur pour un autre processus.

Pour résoudre le problème de la persistance des données d'un conteneur, Docker a deux options :

- le répertoire hôte monté ;
- le volume.

**Figure 2-1**

Persistance des données



## Répertoire hôte monté (bind mounts)

Depuis les débuts de Docker, il est possible de partager des dossiers sur la machine hôte. Lorsque vous utilisez ce partage, un fichier ou un répertoire sur la machine hôte est monté dans un conteneur.

Le fichier ou répertoire est référencé par son chemin absolu sur la machine hôte. Il s'agit donc d'un fichier ou d'un dossier stocké n'importe où sur le système de fichiers hôte du conteneur, monté dans un conteneur en cours d'exécution.

La principale différence entre ce montage et un volume est que, puisqu'il peut exister n'importe où sur le système de fichiers hôte, les processus en dehors de Docker peuvent également le modifier. En revanche, lorsque vous utilisez un volume, un nouveau répertoire est créé dans le répertoire de stockage de Docker sur la machine hôte et Docker gère le contenu de ce répertoire. Il n'est pas donc nécessaire que le fichier ou le répertoire existe déjà sur l'hôte Docker. Il est créé à la demande s'il n'existe pas encore.

Les points de montage sont très performants, mais ils reposent sur le système de fichiers de la machine hôte ayant une structure de répertoire spécifique disponible.

Si vous développez de nouvelles applications Docker, envisagez plutôt d'utiliser des volumes nommés. Vous ne pouvez pas utiliser les commandes Docker CLI pour gérer directement les points de montage.

Envisagez d'utiliser un montage temporaire `tmpfs` si votre conteneur génère des données d'état non persistantes pour éviter de stocker les données n'importe où de façon permanente et pour augmenter les performances du conteneur en évitant d'écrire dans sa couche inscriptible.

Un montage `tmpfs` est donc temporaire et ne persiste que dans la mémoire de l'hôte.

Lorsque le conteneur s'arrête, le montage `tmpfs` est supprimé et les fichiers qui y sont écrits ne seront pas conservés.

Dans l'exemple suivant, je vais créer un répertoire `webapp` sous mon répertoire personnel que je vais lier au répertoire `/usr/share/nginx/html` d'un conteneur lancé à partir de l'image Nginx. Je vais donc garder sur ma machine hôte les sites web hébergés par Nginx.

```
iset@ubuntu:~$ mkdir $HOME/webapp
iset@ubuntu:~$ docker run -itd --name nginx1 --mount type=bind,source=$HOME/webapp,target=/usr/share/nginx/html -p 8080:80 nginx:latest
```

### Remarques

1. Si vous effectuez un montage dans un répertoire non vide du conteneur, le contenu existant du répertoire est masqué par le contenu du répertoire de la machine hôte. Cela peut être bénéfique, par exemple lorsque vous souhaitez tester une nouvelle version de votre application sans créer une nouvelle image, mais ce comportement diffère de celui des volumes Docker.
2. Le type du montage peut être `bind`, `volume` ou `tmpfs`.

Pour tester le montage d'un répertoire existant sur la machine hôte, créez et modifiez un fichier `index.html` dans le répertoire `webapp` que vous avez créé.

Vous allez le lier dans le conteneur au répertoire `/usr/share/nginx/html`. Vous modifierez par la suite la page d'accueil de Nginx située à l'intérieur du conteneur.

```
iset@ubuntu:~$ nano webapp/index.html
```

Figure 2-2

Testez la modification en vous connectant au serveur Nginx à partir de l'adresse IP de la machine hôte :

Figure 2-3

Maintenant, arrêtez et supprimez le conteneur `nginx1` et créez un nouveau conteneur `nginx2` à partir de l'image Nginx.

Effectuez le même type de montage sur le répertoire `webapp` et modifiez le port d'écoute sur la machine hôte par 9090 (notez que vous pouvez utiliser l'option `-v` (`--volume`) à la place de l'option `--mount`).

```
iset@ubuntu:~$ docker stop nginx1
```

```
nginx1
```

```
iset@ubuntu:~$ docker rm nginx1
```

```
nginx1
```

```
iset@ubuntu:~$ docker run -itd --name nginx1 -v $HOME/webapp:/usr/share/nginx/html -p 9090:80 nginx:latest
```

Puis, testez la persistance des données en vous connectant à ce nouveau serveur Nginx :

Figure 2-4

Non sécurisé | 192.168.1.6:9090

Serveur Nginx sur votre conteneur

Montage de type BIND

Utilisez la commande `docker inspect nginx2` pour vérifier que le montage de type `bind` a été créé correctement.

```
iset@ubuntu:~$ docker inspect nginx2
```

Recherchez la section `Mounts` :

Figure 2-5

```
"Mounts": [
  {
    "Type": "bind",
    "Source": "/home/iset/webapp",
    "Destination": "/usr/share/nginx/html",
    "Mode": "",
    "RW": true,
    "Propagation": "rprivate"
  },
]
```

## Volumes

Afin de pouvoir sauvegarder (persister) les données et les partager entre différents conteneurs, Docker a développé le concept de volumes.

Les volumes sont le moyen préféré pour stocker ou utiliser des données persistantes dans des conteneurs Docker. Ils sont entièrement gérés par Docker.

Contrairement à un montage `bind`, vous pouvez créer et gérer des volumes en dehors de la portée de n'importe quel conteneur.



Les volumes présentent plusieurs avantages par rapport aux répertoires partagés.

- Vous pouvez gérer les volumes à l'aide des commandes CLI Docker ou de l'API Docker.
- Les volumes sont plus faciles à sauvegarder ou à migrer que les répertoires partagés.
- Les volumes fonctionnent sur les conteneurs Linux et Windows et peuvent être partagés de manière plus sûre entre plusieurs conteneurs.
- Les pilotes de volume vous permettent de stocker des volumes sur des hôtes distants ou des fournisseurs de cloud et de crypter le contenu des volumes ou d'ajouter d'autres fonctionnalités.
- Les nouveaux volumes peuvent avoir leur contenu prérempli par un conteneur contrairement aux répertoires montés qui masquent le contenu des répertoires à l'intérieur du conteneur.

Contrairement à un montage de répertoire, vous pouvez créer et gérer des volumes en dehors de la portée de n'importe quel conteneur.

Pour commencer, créez un volume :

```
iset@ubuntu:~$ docker volume create elyesvolume
```

```
elyesvolume
```

Listez ensuite les volumes :

Figure 2-6

```
iset@ubuntu:~$ docker volume ls
DRIVER      VOLUME NAME
local       2e98edc41fead97dd01eac98bfd26e7378a268c46a50e988e094c4e46ab412e
local       9e6807c3f096c60527ead619ea6457d8741aac38515b449ba4eb216c8288639a
local       elyesvolume
local       f1b69ed346e4b121af176aa5551290c16fe8b5b4b4cb35e85d15bb6d0b376312
local       f1c89373d70229ea82b34a7acd3431a8670cdd8d5fd1003ff6f84ad9191271da
local       faff62ea5ade57a5e6051441cf1eba55eeb24dee6f1e1d17e9ad99ad0ddc3b70
iset@ubuntu:~$
```

Inspectez un volume et notez l'emplacement de ses données sur la machine hôte :

Figure 2-7

```
iset@ubuntu:~$ docker volume inspect elyesvolume
[
  {
    "CreatedAt": "2020-04-10T09:51:26-07:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/elyesvolume/_data",
    "Name": "elyesvolume",
    "Options": {},
    "Scope": "local"
  }
]
iset@ubuntu:~$
```

Démarrez un conteneur à partir de l'image Nginx en liant notre volume au répertoire `/usr/share/nginx/html` :

```
iset@ubuntu:~$ docker run -itd --name nginx3 --mount
type=volume,source=elyesvolume,target=/usr/share/nginx/html -p 7070:80 nginx:latest
```

Pensez à consulter la section `Mounts` en inspectant ce conteneur avec la commande `docker inspect nginx3`.

Si le conteneur a des fichiers ou des répertoires dans le répertoire à monter, le contenu du répertoire est copié dans le volume.

Le conteneur monte ensuite le volume et l'utilise. D'autres conteneurs qui utilisent le volume ont également accès au contenu prérempli.

```
iset@ubuntu:~$ sudo ls /var/lib/docker/volumes/elyesvolume/_data
```

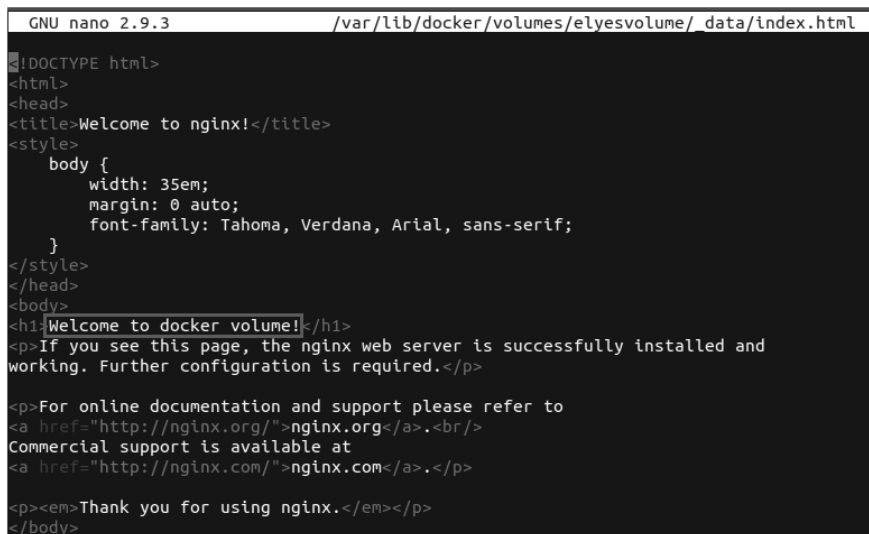
```
[sudo] password for iset:
50x.html index.html
```

Vous pouvez modifier la page `index.html` directement dans le répertoire `_data` du volume et supprimer ensuite le conteneur `nginx3` pour tester la persistance des données.

Créez un conteneur nommé `nginx4` en montant le volume précédent et vérifiez que votre page web persiste avec son contenu modifié.

```
iset@ubuntu:~$ sudo nano /var/lib/docker/volumes/elyesvolume/_data/index.html
```

Figure 2-8



```
GNU nano 2.9.3 /var/lib/docker/volumes/elyesvolume/_data/index.html
!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to docker volume!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
```

```
iset@ubuntu:~$ sudo rm -f nginx3
iset@ubuntu:~$ docker run -itd --name nginx4 --mount
type=volume,source=elyesvolume,target=/usr/share/nginx/html -p 6060:80 nginx:latest
```

Vérifiez que la nouvelle page est hébergée par le conteneur :

Figure 2–9

ⓘ Non sécurisé | 192.168.1.6:6050

## Welcome to docker volume!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

### Remarque

Si vous créez un nouveau conteneur en le montant sur un volume inexistant, ce volume sera créé automatiquement.

```
iset@ubuntu:~$ docker run -itd --name nginx5 --mount
type=volume,source=elyesvolume1,target=/usr/share/nginx/html -p 5050:80 nginx:latest
```

```
elyes@ubuntu:~$ docker volume ls
```

DRIVER	VOLUME NAME
local	eb7e7c58fbf5ec7f23d3734541fd914754466fa89342917ba8b6a337013011f1
local	ed31f8796433977ed1a2564eba37be48f37055be6b06b9b50a934a15f71dda60
local	elyesvolume
local	elyesvolume1

### Remarque

L'option `readonly` (lecture seule), si elle existe, entraîne le montage du volume en lecture seule. Notez le message d'erreur indiquant que le répertoire est protégé en écriture.

```
elyes@ubuntu:~$ docker run -it --name elyes --mount type=volume,source=vol1,
target=/bin,readonly ubuntu
```

```
root@9b2188b8e461:/# ls /bin
 '['                getopt             rgrep
 addpart            gpasswd           rm
 apt                gpgv              rmdir
 apt-cache          grep              run-parts
 apt-cdrom           groups            runcon
 apt-config          gunzip            savelog
 apt-get            gzexe            script
 apt-key            gzip              scriptreplay
 apt-mark           head              sdiff
 arch               hostid            sed
 awk                hostname          select-editor
 b2sum              i386              sensible-browser
 base32             iconv             sensible-editor
 ...
```

```
root@9b2188b8e461:/# mkdir /bin/elyes
```

```
mkdir: cannot create directory '/bin/elyes': Read-only file system
```

## Créer sa propre image à l'aide d'un fichier Dockerfile

Jusqu'à présent, nous nous sommes cantonnés au fonctionnement de base de Docker, qui limite à l'utilisation des images disponibles sur le Docker Hub.

Afin de pouvoir utiliser Docker au maximum, il est indispensable de pouvoir créer des images adaptées à vos projets, ce qui est possible grâce aux Dockerfiles.

### Exemple 1

Les Dockerfiles sont des fichiers qui permettent de construire une image Docker adaptée à vos besoins, étape par étape.

Pour commencer, créez un nouveau fichier Dockerfile à la racine de votre projet. La première chose à faire dans un Dockerfile est de définir de quelle image vous héritez.

Pour cet exemple, je vous propose d'utiliser une image Ubuntu comme base.

- 1 Indiquez la distribution de départ avec la ligne de commande suivante :

```
FROM ubuntu
```

- 2 Mentionnez la personne qui a écrit ce Dockerfile.

```
MAINTAINER Elyes Gassara <elyes.gassara@sfax.r-iset.tn>
```

- 3 Mettez à jour le cache des paquets disponibles et installez Nginx.

```
RUN apt-get update \  
    && apt-get install -y \  
        nginx
```

- 4 Copiez successivement les configurations et scripts de votre système hôte vers votre image.

```
COPY default /etc/nginx/sites-enabled  
COPY index.html /var/www/html  
COPY service_start.sh /home/docker/script/service_start.sh
```

- 5 Ajoutez le droit d'exécution pour votre script.

```
RUN chmod 744 /home/docker/script/service_start.sh
```

- 6 Définissez un point d'entrée (ENTRYPOINT), c'est-à-dire le script qui va se lancer au démarrage du conteneur.

```
ENTRYPOINT /home/docker/script/service_start.sh
```

- 7 Définissez le répertoire de travail quand vous exécuterez un nouveau conteneur par WORKDIR.

```
WORKDIR /home/docker
```

Nous avons toutes les lignes pour créer notre Dockerfile. Le voici dans son intégralité :

```
FROM ubuntu
MAINTAINER Elyes Gassara <elyes.gassara@sfax.r-iset.tn>
RUN apt-get update && apt-get install -y nginx
COPY default /etc/nginx/sites-enabled
COPY index.html /var/www/html
COPY service_start.sh /home/docker/script/service_start.sh
RUN chmod 744 /home/docker/script/service_start.sh
ENTRYPOINT /home/docker/script/service_start.sh
WORKDIR /home/docker
```

Contenu du fichier `default` (le fichier de configuration de Nginx) :

```
server {
    listen 2080 default_server;
    listen [::]:2080 default_server;

    root /var/www/html;
    index index.htm index.html

    server_name_;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Contenu du fichier `index.html` (la page d'accueil) :

```
<html><title>Test Dockerfile</title>
<body><center><b>Test Dockerfile</b></center></body>
</html>
```

Contenu du script `service_start.sh` (qui sera exécuté au lancement du conteneur pour afficher un message. Démarrez Nginx et lancez le Shell Bash) :

```
#!/bin/bash
echo bonjour mes amis
service nginx start
/bin/bash
```

Maintenant que nous avons un Dockerfile, nous allons créer notre image. Pour cela, la commande à utiliser est `docker build`, en spécifiant un nom pour l'image et en indiquant l'emplacement du Dockerfile.

```
elyes@ubuntu:~/exemple1$ docker build -t elyesnginx .
```

```
Sending build context to Docker daemon   5.12kB
Step 1/9 : FROM ubuntu
----> 597ce1600cf4
Step 2/9 : MAINTAINER Elyes Gassara <elyes.gassara@yahoo.fr>
----> Using cache
----> ba98e03c9842
Step 3/9 : RUN apt-get update && apt-get install -y nginx
----> Using cache
----> e2cacb851ced
Step 4/9 : COPY default /etc/nginx/sites-enabled
----> Using cache
----> 6b9cd0ccd5ce
Step 5/9 : COPY index.html /var/www/html
----> Using cache
----> e963ac13ecf1
Step 6/9 : COPY service_start.sh /home/docker/script/service_start.sh
----> Using cache
----> 511c17e84843
Step 7/9 : RUN chmod 744 /home/docker/script/service_start.sh
----> Using cache
----> 77ba30d02f24
Step 8/9 : ENTRYPOINT /home/docker/script/service_start.sh
----> Using cache
----> fc1d675c8187
Step 9/9 : WORKDIR /home/docker
----> Using cache
----> 794c3f3ae01c
Successfully built 794c3f3ae01c
Successfully tagged elyesnginx:latest
```

Lorsque vous exécutez `docker build`, vous devez spécifier le chemin du Dockerfile (d'où le point à la fin de la commande si vous lancez la commande depuis le même endroit).

Durant la construction, vous allez voir s'exécuter toutes les étapes les unes après les autres. Le premier lancement est long, car aucune des étapes n'a été appelée auparavant. Lors d'un second `build`, vous verrez que les étapes sont expédiées à grande vitesse, à condition que les étapes précédentes n'aient pas été modifiées dans votre Dockerfile bien sûr.

Lancez un conteneur à partir de cette image comme suit :

```
iset@ubuntu:~/exemple1$ docker run -it --name elyesnginx1 -p 8080:2080 elyesnginx
```

```
bonjour mes amis
* Starting nginx nginx          [ OK ]
```

```
root@c450da80859d:/home/docker# ss -anltp
```

State	Recv-Q	Send-Q	Local Address:Port
LISTEN	0	511	0.0.0.0:2080
LISTEN	0	511	:::2080

Notez le message « bonjour mes amis » qui s'affiche après le lancement du conteneur, ainsi que le port d'écoute de Nginx qui a été modifié dans le fichier de configuration par 2080.

Effectuez une requête sur votre serveur web depuis la machine hôte :

Figure 2-10

① Non sécurisé | 192.168.1.6:8080

Test Dockerfile

Le tableau 2-1 présente quelques instructions Dockerfile utiles.

Tableau 2-1 Instructions Dockerfile

Instructions	Description
FROM	Image parente
MAINTAINER	Auteur
LABEL	Ajout de métadonnées
RUN	Exécution des commandes dans le conteneur
ADD	Ajout des fichiers dans le conteneur
COPY	Ajout d'un fichier dans l'image
WORKDIR	Définition du répertoire de travail
EXPOSE	Définition des ports d'écoute par défaut
VOLUME	Définition des volumes utilisables
CMD	Exécution d'une commande au démarrage du conteneur
ENTRYPOINT	Exécution d'une commande au démarrage du conteneur
ARG	Variables passées comme paramètres à la construction de l'image

Tableau 2–1 Instructions Dockerfile (suite)

Instructions	Description
ENV	Variable d’environnement
USER	Nom d’utilisateur ou UID à utiliser
ONBUILD	Instructions exécutées lors de la construction d’images enfants

Différence entre CMD et ENTRYPOINT

Les instructions `CMD` et `ENTRYPOINT` sont similaires dans la mesure où elles spécifient toutes deux les commandes qui s’exécutent dans une image. Elles présentent néanmoins une différence importante : `CMD` définit simplement une commande à exécuter dans l’image si aucun argument n’est transmis à `docker run`, tandis que `ENTRYPOINT` permet que votre image se comporte comme un binaire. Par conséquent :

- `ENTRYPOINT` spécifie une commande qui sera toujours exécutée au démarrage du conteneur ;
- `CMD` spécifie les arguments qui seront transmis à `ENTRYPOINT`.

Par exemple, si votre fichier Dockerfile est :

```
FROM debian:latest
RUN apt-get update
RUN apt-get -y install iputils-ping
ENTRYPOINT ["/bin/ping"]
CMD ["localhost","-c 2"]
```

Créez une image `test` à partir de ce fichier :

```
elyes@ubuntu:~$ docker build -t test .
```

L’exécution de cette image sans aucun argument enverra une requête ping à l’hôte local :

```
elyes@ubuntu:~$ docker run -it test

PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.050 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.066 ms

--- localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.050/0.058/0.066/0.008 ms
```

L’exécution de l’image avec un argument fait maintenant le ping sur l’argument :

```
elyes@ubuntu:~$ docker run -it test www.isetsf.net

PING isetsf.net (185.62.136.67) 56(84) bytes of data.
64 bytes from server.hosting-tunisia.com (185.62.136.67): icmp_seq=1 ttl=46 time=74.5 ms
64 bytes from server.hosting-tunisia.com (185.62.136.67): icmp_seq=2 ttl=46 time=141 ms
^C
```



```
--- isetsf.net ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 74.477/107.671/140.866/33.194 ms
```

## Différence entre COPY et ADD

Les instructions `COPY` et `ADD` ont la même finalité : permettre de copier des fichiers dans une image Docker :

- `COPY` permet de copier un fichier ou un répertoire local de la machine qui a créé l'image Docker dans l'image Docker ;
- `ADD` fait la même chose mais prend également en charge des sources distantes, telles que des URL ou un fichier archive, qui seront extraites dans l'image Docker.

## Différence entre MAINTAINER et LABEL

L'instruction `MAINTAINER` permet de définir le champ `Auteur` des images générées. Mais selon la documentation officielle de Docker, l'instruction `MAINTAINER` est obsolète.

Vous pouvez utiliser à la place l'instruction `LABEL` qui est plus flexible. Cette dernière permet de définir des métadonnées et peut être facilement visualisée avec la commande `docker inspect`.

## Exemple 2

Dans cet exemple, nous allons créer ensemble une image Docker dans laquelle nous allons installer Node.js ainsi que les différentes dépendances de notre application qui sert à créer un simple serveur web.

Node (ou plus formellement Node.js) est un environnement d'exécution open source, multi-plate-forme, qui permet aux développeurs de créer toutes sortes d'applications et d'outils côté serveur en JavaScript.

Pour commencer, nous allons créer un fichier `package.json` pour stocker des métadonnées utiles sur le projet, ce qui aide à gérer les modules Node.js dépendants du projet.

Express est une infrastructure d'application (framework) écrite en JavaScript et hébergée dans l'environnement d'exécution Node.js.

**Fichier `package.json` :**

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "Elyes Gassara <elyes.gassara@sfax.r-iset.tn>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
}
```

```

    "dependencies": {
      "express": "^4.16.1"
    }
  }
}

```

#### Fichier server.js :

```

'use strict';
const express = require('express');
// Constantes
const PORT = 8080;
const HOST = '0.0.0.0';

var os = require("os");
// Nous indiquons à notre serveur Express comment gérer une demande
const app = express();
app.get('/', (req, res) => {
  res.send('<center>Formation présentée par Elyes Gassara<br>Application web  
<font color="0000FF"><b>version 2</b> </font><br> Conteneur : <b>' +
String(os.hostname()));
});

app.listen(PORT, HOST);
console.log(String(os.hostname()));

```

#### Fichier Dockerfile :

```

FROM node
MAINTAINER Elyes Gassara <elyes.gassara@sfax.r-iset.tn >
# Création du répertoire de travail
WORKDIR /usr/src/app
# Installation des dépendances
COPY package.json .
RUN npm install
# Copie du code à l'intérieur du conteneur pour pouvoir l'utiliser ensuite
COPY server.js .

# Ouverture (ou exposition) du port 8080 pour pouvoir accéder au serveur à partir de
la machine hôte
EXPOSE 8080
# Démarrage du serveur
CMD [ "node", "server.js" ]

```

On utilise ensuite la commande `docker build` pour créer une image `http-server`.

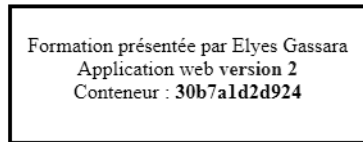
```
$ docker build -t http-server .
```

Nous lançons à présent un conteneur nommé `http` à partir de cette image :

```
$ docker run -d --name http -p 80:8080 http-server
```

Test du serveur web :

Figure 2-11



En fait, j'ai créé deux versions pour ce serveur web : une version avec le tag `v1` qui affiche « Application web version 1 » et une autre avec le tag `v2` qui affiche le même message mais pour la version 2.

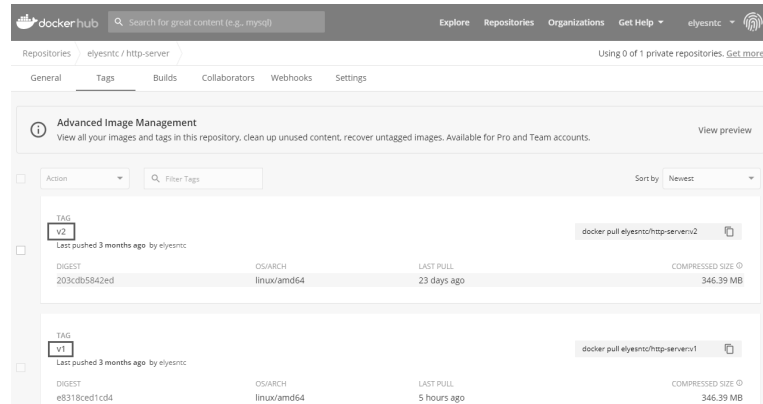
Nous avons besoin de ces deux images dans les activités suivantes, je les ai donc déposées sur Docker Hub.

Par exemple, pour la version 2, j'ai utilisé les commandes suivantes :

```
docker tag http-server elyesntc/http-server:v2
docker push elyesntc/http-server:v2
```

Vous voyez alors l'image sur Docker Hub avec ses deux tags (voir figure 2-12).

Figure 2-12



## Conteneur en production

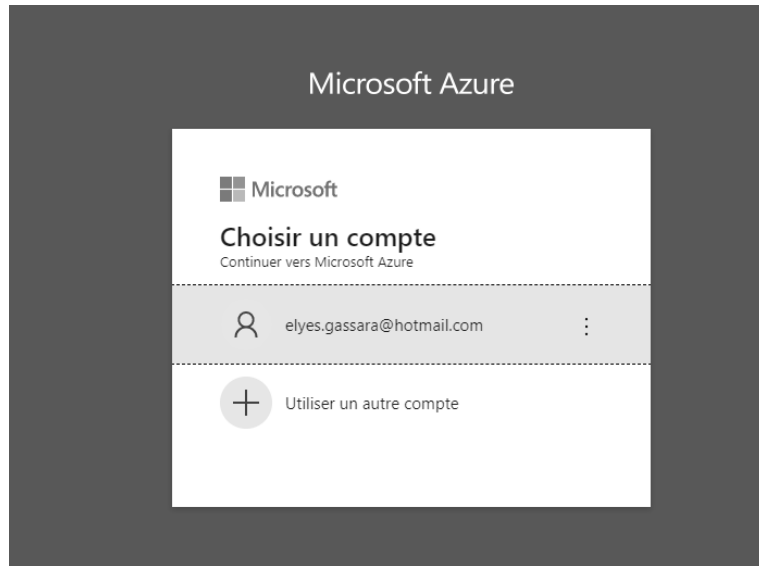
Nous allons ici voir comment mettre notre serveur web sur la plate-forme d'un fournisseur de cloud public en lançant un conteneur à partir de l'image `elyesntc/http-server:v1` créée précédemment.

Dans cette démonstration, j'ai choisi Microsoft Azure comme plate-forme d'hébergement. Azure propose de nombreuses options pour exécuter des conteneurs dans le cloud, chacune avec ses propres fonctionnalités, prix et complexité. Le moyen le plus simple et rapide d'exécuter un conteneur dans Azure consiste à utiliser le service ACI (*Azure Container Instances*).

Ce dernier permet de ne pas avoir à gérer de machines virtuelles et à adopter un service de niveau supérieur.

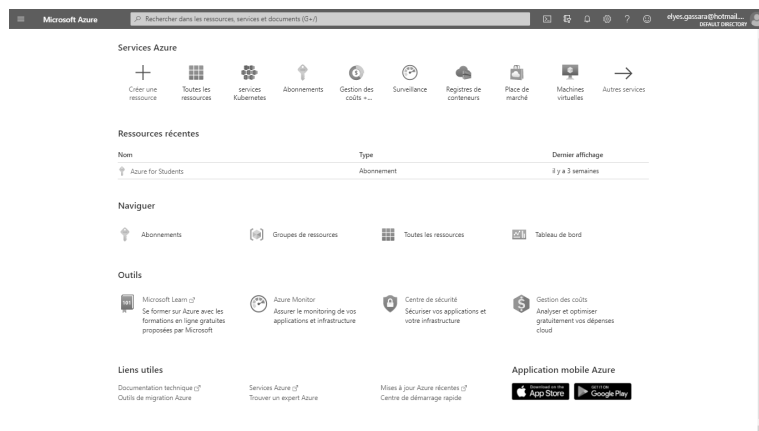
Pour accéder au portail Microsoft Azure, rendez-vous directement sur le site <https://portal.azure.com/>.

Figure 2–13



Pour pouvoir accéder à l'interface du portail Azure, vous avez besoin d'un compte Microsoft et d'un abonnement Azure. Si ce n'est pas encore le cas, vous pouvez créer un compte Azure gratuitement.

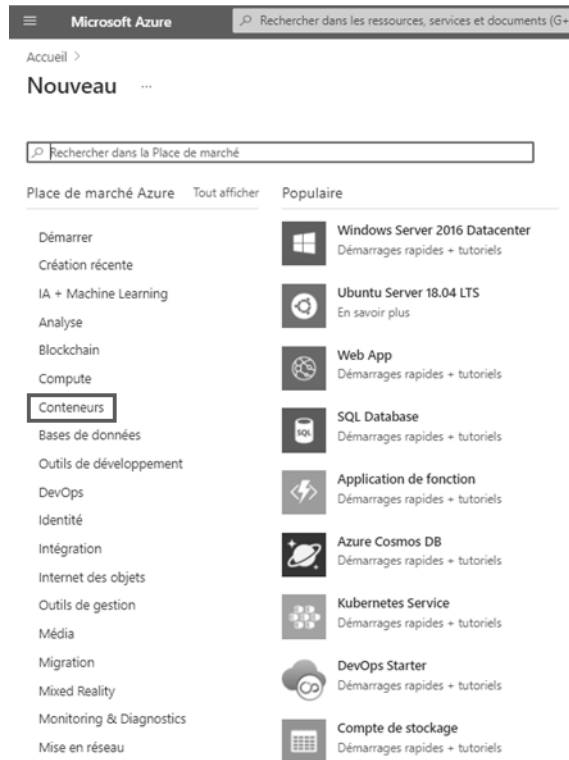
Figure 2–14  
Interface du portail Azure



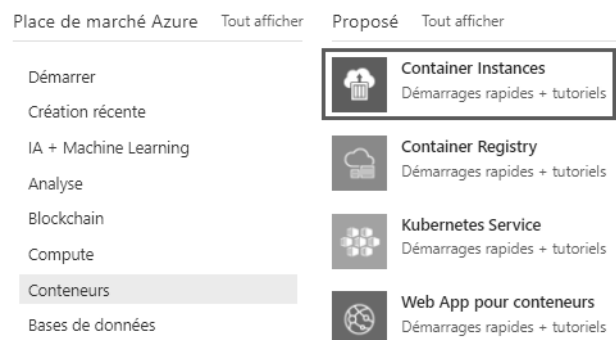
La page d'accueil contient des informations concernant les tâches courantes, les ressources récemment utilisées, les principaux services Azure, etc. Vous pouvez créer une ressource ou afficher un large éventail de services dans la place de marché Azure.

Cliquez sur *Créer une ressource* et notez que les services sont répertoriés par catégorie. Vous trouverez différents types de ressources comme celles liées à l'intelligence artificielle, aux bases de données, au DevOps et à l'Internet des objets.

Choisissez *Conteneurs* pour cet exemple (voir figure 2-15).

**Figure 2-15**

Dans la fenêtre qui s'affiche, choisissez la première proposition affichée, soit *Container Instances* (voir figure 2-16).

**Figure 2-16**

Azure Container Instances est une excellente solution pour tout scénario pouvant fonctionner dans des conteneurs isolés, y compris les applications simples, l'automatisation des tâches et le build des tâches.

Pour les scénarios dans lesquels nous avons besoin d'une orchestration complète des conteneurs, y compris la découverte de services sur plusieurs conteneurs, la mise à l'échelle automatique et les mises à niveau d'applications coordonnées, nous allons utiliser Azure Kubernetes Service (AKS).

J'ai créé un nouveau groupe de ressources « webserver » et j'ai nommé le conteneur « http-server ».

Sélectionnez une zone géographique qui correspond au mieux à vos besoins, qu'il s'agisse de conformité ou de fonctionnalités de résilience. Pour ma part, j'ai choisi la région « France-Centre » (voir figure 2-17).

Figure 2-17

Accueil > Nouveau >

## Créer une instance de conteneur ...

De base Réseau Avancé Étiquettes Vérifier + créer

Azure Container Instances (ACI) vous permet d'exécuter rapidement et facilement des conteneurs sur Azure sans avoir à gérer de serveurs ni à apprendre de nouveaux outils. ACI offre une facturation par seconde pour minimiser le coût d'exécution des conteneurs sur le cloud. En savoir plus sur Azure Container Instances

Détails du projet

Sélectionnez l'abonnement pour gérer les coûts et les ressources déployées. Utilisez les groupes de ressources comme les dossiers pour organiser et gérer toutes vos ressources.

Abonnement \* ⓘ Azure for Students ✓

Groupe de ressources \* ⓘ (Nouveau) webserver ✓  
Créer nouveau

Détails du conteneur

Nom de conteneur \* ⓘ http-server ✓

Région \* ⓘ (Europe) France-Centre ✓

Source d'image \* ⓘ

☐ Images de démarrage rapide

☐ Azure Container Registry

☒ Docker Hub ou autre registre

Type d'image \* ⓘ

☒ Public ☐ Privé

Image \* ⓘ elyesntc/http-server:v1 ✓

ⓘ Si rien n'est spécifié, Docker Hub est utilisé pour le registre de conteneurs et la dernière version de l'image est tirée.

Les services Microsoft Azure sont disponibles dans le monde entier. Vous pouvez choisir la meilleure région pour vos besoins en fonction des considérations techniques et réglementaires. Cela inclut les fonctionnalités de service, la résidence des données, les exigences de conformité et la latence.

La figure 2-18 présente un aperçu des régions et zones de disponibilité Azure.

**Figure 2-18**  
Zones géographiques Azure



Choisissez ensuite le type du système d'exploitation. Dans notre cas, il s'agit de « Linux ».

Nous utilisons une simple page web pour cette démonstration. Nous allons donc minimiser les ressources de notre conteneur. Cliquez sur le lien *Changer la taille* (voir figure 2-19).

**Figure 2-19**

Pour pouvoir accéder à notre serveur, nous avons besoin d'une adresse IP publique et d'une étiquette de nom DNS.

Il faut bien sûr exposer le port qui a été défini dans le fichier Dockerfile à partir duquel nous avons construit l'image `http-server`. Pour rappel, il s'agissait de l'instruction `EXPOSE 8080` (voir figure 2-20).

Figure 2-20

Microsoft Azure

Rechercher dans les ressources, services et documents (G+)

Accueil > Nouveau >

## Créer une instance de conteneur

De base Réseau Avancé Étiquettes Vérifier + créer

Choisissez entre trois options de réseau pour votre instance de conteneur :

- « **Public** » crée une adresse IP publique pour votre instance de conteneur.
- « **Privée** » vous permet de choisir un réseau virtuel nouveau ou existant pour votre instance de conteneur. Cette option n'est pas encore disponible pour les conteneurs Windows.
- « **Aucune** » ne crée pas d'adresse IP publique ni de réseau virtuel. Vous pouvez toujours accéder à vos journaux de conteneur à l'aide de la ligne de commande.

Type de réseau ☒ Public ☐ Privé ☐ Aucun

Étiquette du nom DNS  ☒ .francecentral.azurecontainer.io

Ports

La stratégie de redémarrage que nous allons choisir dans cet exemple permet de redémarrer le conteneur en cas d'échec (voir figure 2-21).

Figure 2-21

De base Réseau Avancé Étiquettes Vérifier + créer

Configurez des variables et des propriétés de conteneur supplémentaires.

Stratégie de redémarrage

Si vous voulez catégoriser vos ressources, vous pouvez les marquer par une ou plusieurs étiquettes (voir figure 2-22).

Figure 2-22

Nom	Valeur
Département	TI

À la fin de la configuration, un récapitulatif de tous les paramètres spécifiés s'affichera pour vérification. Cliquez sur *Créer* si tout vous convient afin de procéder au déploiement (voir figure 2-23).

Le déploiement va prendre quelques minutes, le temps que l'image soit téléchargée et que les ressources nécessaires pour lancer le conteneur soient provisionnées (voir figure 2-24).



Figure 2-23

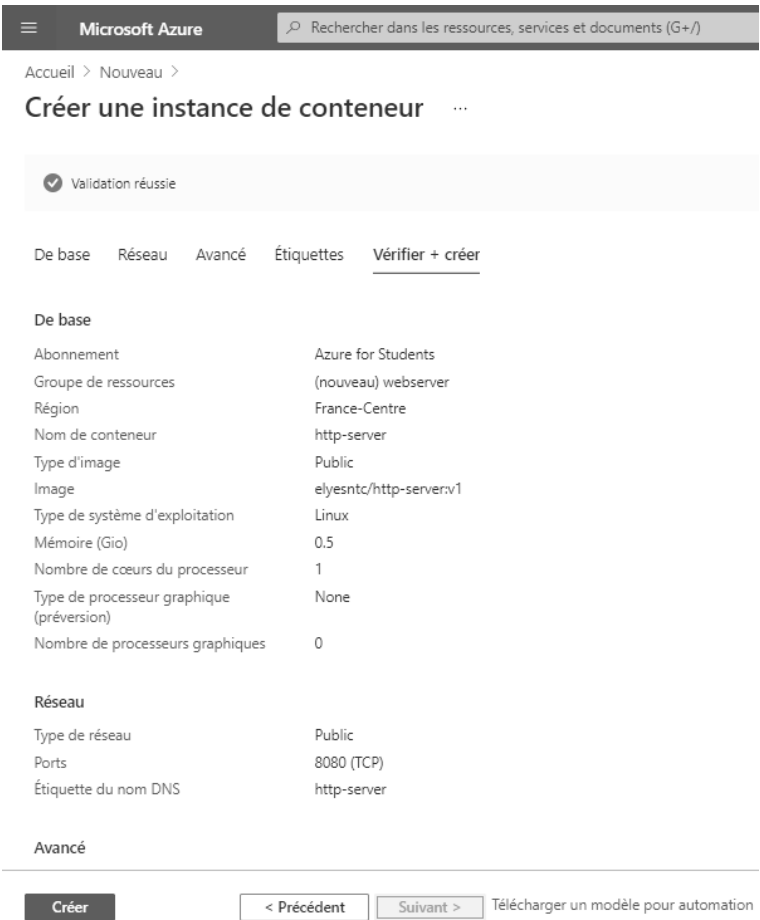
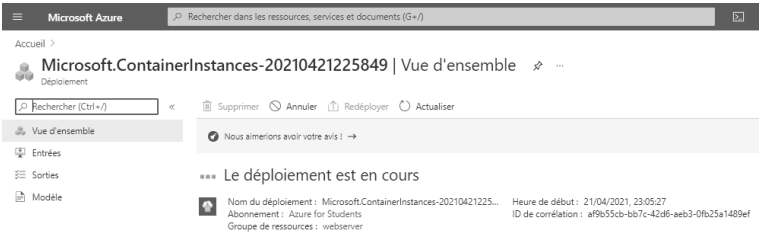


Figure 2-24



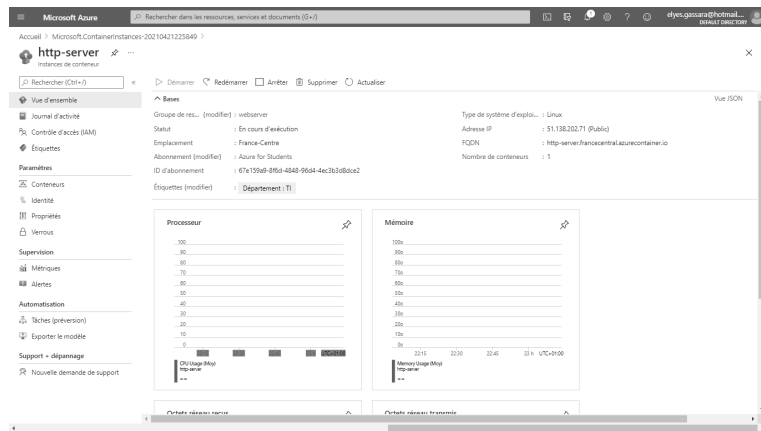
Une fois le déploiement effectué, vous pouvez accéder à la nouvelle ressource créée (voir figure 2-25).

Figure 2-25



Une vue d'ensemble de cette ressource est alors affichée (voir figure 2-26).

Figure 2-26



Cliquez sur le menu *Conteneurs* pour voir les conteneurs en cours d'exécution. Vous pouvez aussi visualiser les événements liés à votre conteneur, les propriétés et les journaux (voir figure 2-27).

Figure 2-27

1 conteneur

Nom	Image	État	État précédent	Heure de début	Nombre de redémarrages
http-server	elyesntc/http-server:1	Running	-	2021-04-21T22:07:34.953Z	0

Événements

Propriétés

Journaux

Connecter

Afficher le fuseau horaire

☒ Heure locale

☐ UTC

Nom	Type	Premier horodatage	Dernier horodatage	Message	Nombre
Started	Normal	21/04/2021, 23:07 UTC+1	21/04/2021, 23:07 UTC+1	Started container	1
Pulled	Normal	21/04/2021, 23:07 UTC+1	21/04/2021, 23:07 UTC+1	Successfully pulled image 'elye...	1
Pulling	Normal	21/04/2021, 23:06 UTC+1	21/04/2021, 23:06 UTC+1	pulling image 'elyesntc/http-se...	1

Le conteneur est en cours d'exécution et il ne reste plus qu'à l'interroger.

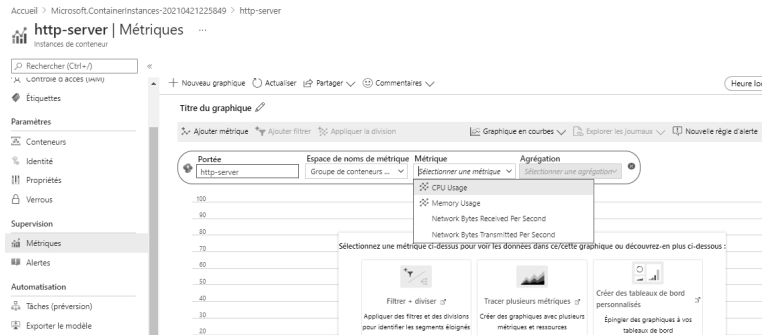
Pour cela, lancez une requête sur le serveur en utilisant le nom DNS attribué par Azure et vérifiez son bon fonctionnement (voir figure 2-28).

Figure 2-28



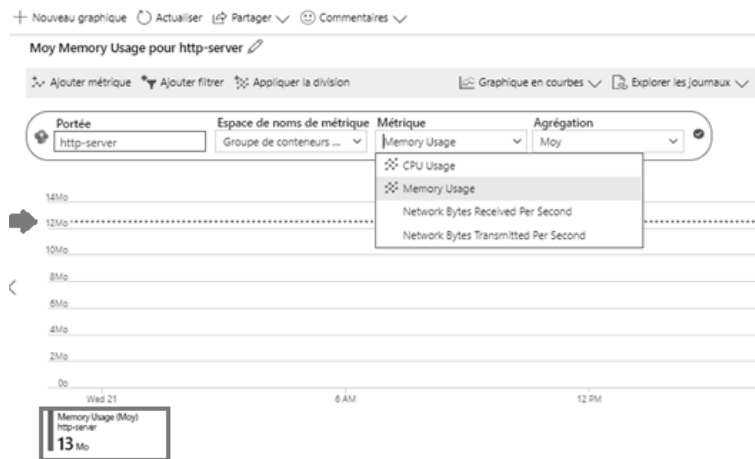
Il est intéressant de consulter la section *Supervision* qui affiche les métriques liées au conteneur, telles que l'utilisation du CPU, de la mémoire et du réseau (voir figure 2-29).

Figure 2-29



Nous avons configuré notre conteneur avec 0,5 Go de RAM. Allons vérifier que c'est largement suffisant en lançant notre métrique sur l'utilisation de la mémoire (voir figure 2-30).

Figure 2-30



## Docker Compose

Docker Compose est un outil qui vous permet de définir et de gérer des applications Docker contenant plusieurs conteneurs. Il utilise un fichier YAML pour configurer les services, les réseaux et les volumes de l'application.

Les déploiements d'applications à hôte unique, les tests automatisés et le développement local sont les cas d'utilisation les plus courants de Docker Compose.

Le package d'installation de Docker Compose est disponible dans les référentiels officiels d'Ubuntu, mais il ne s'agit peut-être pas toujours de la version la plus récente. Il est donc recommandé d'installer Docker Compose à partir du référentiel GitHub de Docker.

Avant de télécharger le fichier binaire, visitez la page de publication du référentiel Compose sur GitHub et vérifiez si une nouvelle version est disponible au téléchargement.

Pour installer Docker Compose sur Ubuntu, procédez comme suit :

- 1 Téléchargez le fichier binaire Docker Compose dans le répertoire `/usr/local/bin` à l'aide de la commande `curl` suivante :

```
elyes@tunisie:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

- 2 Une fois le téléchargement terminé, appliquez des autorisations exécutables au binaire Docker Compose :

```
elyes@tunisie:~$ sudo chmod +x /usr/local/bin/docker-compose
```

- 3 Vérifiez l'installation en exécutant la commande suivante qui affichera la version de Docker Compose :

```
elyes@tunisie:~$ docker-compose -version
```

```
docker-compose version 1.26.0, build d4451659
```

Ici nous allons voir comment utiliser Docker Compose pour configurer une application WordPress contenant plusieurs conteneurs.

Commencez par créer un répertoire de projet et naviguez dans celui-ci :

```
elyes@tunisie:~$ mkdir my_app
elyes@tunisie:~$ cd my_app/
elyes@tunisie:~/my_app$
```

Lancez votre éditeur de texte et créez un fichier nommé `docker-compose.yml` dans le répertoire du projet :

```
elyes@tunisie:~/my_app$ gedit docker-compose.yml
```

Ajoutez ensuite le contenu suivant dans ce fichier :

```
version: '3.3'

services:
  db:
    image: mysql:5.7
    restart: always
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wordpress
```

```
wordpress:
  image: wordpress
  restart: always
  volumes:
    - wp_data:/var/www/html
  ports:
    - "8080:80"
  environment:
    WORDPRESS_DB_HOST: db:3306
    WORDPRESS_DB_NAME: wordpress
    WORDPRESS_DB_USER: root
    WORDPRESS_DB_PASSWORD: password
  depends_on:
    - db

volumes:
  db_data:
  wp_data:
```

Analysons le code ligne par ligne.

Dans la première ligne, nous avons spécifié la version du fichier Compose. Il existe plusieurs versions du format de fichier Compose prenant en charge des versions spécifiques de Docker.

Ensuite, nous avons défini deux services, à savoir `db` et `wordpress`. Chaque service exécute une image et crée un conteneur distinct lorsque `docker-compose` est exécuté.

- Le service `db` :
  - utilise l'image `mysql:5.7`. Si l'image n'est pas présente localement, elle sera téléchargée à partir du référentiel public de Docker Hub ;
  - utilise la stratégie de redémarrage qui demandera au conteneur de toujours redémarrer ;
  - crée un volume nommé `db_data` pour rendre la base de données MySQL persistante ;
  - définit les variables d'environnement pour l'image `mysql:5.7` : le nom de la base de données et le mot de passe du super utilisateur ;
  - crée un volume `db_data` et le lie au répertoire `/var/lib/mysql` à l'intérieur du conteneur.
- Le service `wordpress` :
  - utilise l'image `wordpress`. Si l'image n'est pas présente sur votre système, Compose l'extraira du référentiel public de Docker Hub ;
  - utilise la stratégie de redémarrage qui demandera au conteneur de toujours redémarrer ;
  - crée un volume `wp_data` et le lie au répertoire `/var/www/html` à l'intérieur du conteneur ;
  - expose le port 8080 sur la machine hôte ;
  - définit les variables d'environnement pour l'image `wordpress` et notamment les paramètres d'accès à la base de données ;
  - l'instruction `depends_on` définit la dépendance entre les deux services. Dans cet exemple, `db` sera démarré avant `wordpress`.

À partir du répertoire du projet, démarrez l'application WordPress en exécutant la commande suivante :

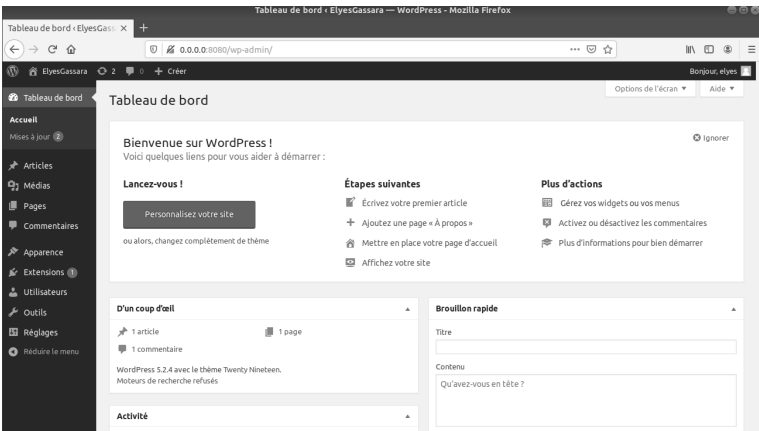
```
eIyes@tunisie:~/my_app$ docker-compose up
```

Docker Compose extrait les deux images, démarre deux conteneurs et crée les volumes nécessaires.

Entrez `http://0.0.0.0:8080/` dans votre navigateur et vous verrez l'écran d'installation de WordPress apparaître.

À ce stade, l'application WordPress est opérationnelle et vous pouvez commencer à travailler sur votre thème ou votre plug-in.

**Figure 2-31**  
Tableau de bord WordPress



Pour arrêter la composition, appuyez sur les touches `CTRL+C`.

Vous pouvez également démarrer la composition en mode détaché en passant l'option `-d`.

```
eIyes@tunisie:~/my_app$ docker-compose up -d
```

```
Starting my_app_db_1 ... done
Starting my_app_wordpress_1 ... done
```

Pour vérifier les services en cours, utilisez l'option `ps` :

```
eIyes@tunisie:~/my_app$ docker-compose ps
```

Name	Command	State	Ports
-----			
my_app_db_1	docker-entrypoint.sh mysqld	Up	3306/tcp, 33060/tcp
my_app_wordpress_1	docker-entrypoint.sh apach ...	Up	0.0.0.0:8080->80/ tcp,:::8080->80/tcp

Pour arrêter l'utilisation des services lorsque Docker Compose est exécuté en mode détaché, utilisez la commande suivante :

```
elyes@tunisie:~/my_app$ docker-compose stop
```

```
Stopping my_app_wordpress_1 ... done
Stopping my_app_db_1         ... done
```

Si vous souhaitez supprimer entièrement les conteneurs, utilisez l'option `down` :

```
elyes@tunisie:~/my_app$ docker-compose down
```

```
Removing my_app_wordpress_1 ... done
Removing my_app_db_1         ... done
Removing network my_app_default
```

Passer l'option `--volume` supprimera également les volumes de données :

```
elyes@tunisie:~/my_app$ docker-compose down --volume
```

```
Removing network my_app_default
WARNING: Network my_app_default not found.
Removing volume my_app_db_data
Removing volume my_app_wp_data
```

Si, pour une raison quelconque, vous souhaitez désinstaller Docker Compose, vous pouvez simplement supprimer le fichier binaire en tapant :

```
elyes@tunisie:~/my_app$ sudo rm /usr/local/bin/docker-compose
```

## Application

Nous allons maintenant améliorer notre serveur web créé précédemment avec Node.js en utilisant une base de données pour stocker le nombre de visites.

**Fichier `package.json` :**

```
{
  "name": "sample_nodejs_with_redis",
  "version": "1.0.0",
  "description": "Node.js and redis on Docker",
  "author": "Elyes Gassara <elyes.gassara@sfax.r-iset.tn >",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
```

```

    "express": "^4.13.3",
    "redis": "^2.2.5"
  }
}

```

#### Fichier server.js :

```

'use strict';

const express = require('express'),
    http = require('http'),
    redis = require('redis'),
    os = require('os');

const client = redis.createClient(6379, 'redis');
const app = express();

app.get('/', function(req, res, next) {
  client.incr('visits', function(err, visits) {
    if(err) return next(err);
    const response = '<center><br> <font color="0000FF">Formation animée par Elyes Gassara </font><br>Cette requête a été traitée par le conteneur <b>' + os.hostname() + ' </b><br> Vous avez vu cette page <b> <font color="0F000F">' + visits + ' fois.</font></b> <br>';

    res.send(response);
  });
});

const appPort = 8080;
http.createServer(app).listen(appPort, function() {
  console.log('Listening on port ' + appPort);
});

```

#### Fichier Dockerfile :

```

FROM node
MAINTAINER Elyes Gassara <elyes.gassara@sfax.r-iset.tn >

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json /usr/src/app/
RUN npm install

COPY server.js /usr/src/app/

EXPOSE 8080
CMD [ "npm", "start" ]

```



Fichier `docker-compose.yml` :

```
version: "3"
services:
  http:
    image: nodejs-redis-app
    ports:
      - "80:8080"
    depends_on:
      - redis
    networks:
      - frontend
      - backend

  redis:
    image: redis
    networks:
      - backend
    volumes:
      - elyesvolume:/data
networks:
  frontend:
  backend:
volumes:
  elyesvolume:
```

Commençons par créer l'image `nodejs-http-server` :

```
$ docker build -t nodejs-redis-app .
```

Lançons ensuite l'application avec `docker-compose` :

```
elyes@ubuntu:~/application$ docker-compose up -d
```

```
Creating network "application_backend" with the default driver
Creating network "application_frontend" with the default driver
Creating volume "application_elyesvolume" with default driver
Creating application_redis_1 ... done
Creating application_http_1 ... done
```

Test de l'application :

**Figure 2-32**

Formation animée par Elyes Gassara  
Cette requête a été traitée par le conteneur 3e250824b9e0  
Vous avez vu cette page 1 fois.

Nous avons besoin de cette image dans les activités suivantes, je l’ai donc déposée sur Docker Hub en utilisant ces commandes :

```
docker tag nodejs-redis-app elyesntc/nodejs-redis-app
docker push elyesntc/nodejs-redis-app
```

Supprimez puis lancez les conteneurs pour tester la persistance des données :

```
elyes@ubuntu:~/application$ docker-compose down
```

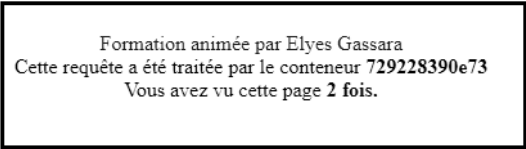
```
Stopping application_http_1 ... done
Stopping application_redis_1 ... done
Removing application_http_1 ... done
Removing application_redis_1 ... done
Removing network application_backend
Removing network application_frontend
```

```
elyes@ubuntu:~/application$ docker-compose up -d
```

```
Creating network "application_backend" with the default driver
Creating network "application_frontend" with the default driver
Creating application_redis_1 ... done
Creating application_http_1 ... done
```

Notez que le nombre de visites est maintenant égal à deux car la première visite a été stockée dans le volume elyesvolume :

Figure 2–33



Le tableau 2-2 présente quelques commandes Docker Compose utiles :

Tableau 2–2 Commandes Docker Compose de base

Commandes	Rôles
docker-compose up -d	Démarre un ensemble de conteneurs en arrière-plan.
docker-compose down	Stoppe un ensemble de conteneurs.
docker-compose exec [service] [command]	Exécute une commande au sein d’un service
docker-compose logs (<ID>/<NAME>)	L’option (<ID>/<NAME>) avec docker-compose logs vous permet de voir les logs d’un conteneur uniquement, au lieu de voir tous les logs.
docker-compose stop	Arrête tous les services situés dans votre fichier docker-compose.yml mais ne supprime pas vos conteneurs.

Tableau 2-2 Commandes Docker Compose de base *(suite)*

Commandes	Rôles
<code>docker-compose start</code>	Démarre tous les services situés dans votre fichier <code>docker-compose.yml</code> .
<code>docker-compose pause</code>	Met en pause les conteneurs dans un service qui est répertorié dans un fichier <code>docker-compose.yml</code> .
<code>docker-compose unpause</code>	Reprend les conteneurs en pause dans un service qui est répertorié dans un fichier <code>docker-compose.yml</code> .
<code>docker-compose restart</code>	Redémarre tous les conteneurs d'un service décrit dans un fichier <code>docker-compose.yml</code> .
<code>docker-compose rm</code>	Supprime tous les conteneurs répertoriés dans un fichier qui sont arrêtés.
<code>docker-compose pull</code>	Télécharge uniquement les images à partir du fichier <code>docker-compose.yml</code> .
<code>docker-compose build</code>	Construit uniquement les images à partir du fichier <code>docker-compose.yml</code> .
<code>docker-compose ps</code>	Affiche la liste des conteneurs qui sont déployés à partir d'un fichier <code>docker-compose.yml</code> .
<code>docker-compose images</code>	Énumère toutes les images qui sont sur le système dans le cadre d'un fichier <code>docker-compose.yml</code> .
<code>docker-compose top</code>	Affiche l'utilisation des conteneurs qui sont déployés à partir d'un fichier <code>docker-compose.yml</code> .
<code>docker-compose config</code>	Visualise et valide un fichier <code>docker-compose.yml</code> .

## Conclusion

Dans ce chapitre, vous avez appris à utiliser un stockage persistant pour les conteneurs Docker, à créer votre propre image Docker grâce à l'utilisation du fichier `Dockerfile` et à installer et utiliser Docker Compose sur Ubuntu.

L'utilisation de Docker Compose peut considérablement améliorer votre flux de travail et votre productivité. Vous pouvez définir votre environnement de développement avec Docker Compose et le partager avec les collaborateurs du projet.

## Validation des acquis

### Questions :

- ❶ Où sont stockés les volumes Docker ?
- ❷ Un volume Docker est :
  - Un espace de stockage connecté à un ou plusieurs conteneurs Docker.
  - Une image fonctionnelle à partir de laquelle on crée des conteneurs identiques.
  - Un snapshot de l'application que l'on déploie dans un cluster comme Swarm.
- ❸ Est-ce que vous perdrez votre travail si vous sortez accidentellement d'un conteneur ?
- ❹ Comment exécuter plusieurs copies d'un projet Docker Compose sur le même hôte ?
- ❺ Est-ce que je peux utiliser « json » au lieu de « yaml » pour mon fichier Compose ?
- ❻ Lequel des éléments suivants est un fichier qui contient toutes les commandes qu'un utilisateur pourrait appeler sur la ligne de commande pour créer une image ?
  - Docker Cloud
  - Dockerfile
  - Docker Kitematic
  - Docker Compose
- ❼ Vous ne pouvez pas créer plusieurs conteneurs à partir de la même image.
  - Vrai
  - Faux
- ❽ Comment pouvez-vous vous assurer que vos conteneurs Docker et leurs données sont sauvegardés en toute sécurité ?
  - En sauvegardant manuellement le répertoire `/var/lib/docker/`.
  - Les données stockées dans les conteneurs Docker sont sauvegardées sur Docker Hub.
  - En configurant le mappage de volume sur tous les conteneurs pour stocker leurs données dans un seul emplacement sur le serveur (par exemple, `/data/`) et en sauvegardant cet emplacement.
- ❾ Est-il possible de contrôler l'ordre de démarrage des services avec Docker Compose ?

**Réponses :****① Où sont stockés les volumes docker ?**

Vous devez naviguer à : `/var/lib/docker/volumes`.

**② Un volume Docker est :**

Un espace de stockage connecté à un ou plusieurs conteneurs Docker.

**③ Est-ce que vous perdrez votre travail si vous sortez accidentellement d'un conteneur ?**

Quitter accidentellement un conteneur n'aura aucun impact sur les fichiers. Le seul moyen de perdre votre progression serait d'émettre une commande spécifique pour supprimer le conteneur Docker.

**④ Comment exécuter plusieurs copies d'un projet Compose sur le même hôte ?**

Compose utilise le nom de projet pour attribuer des identifiants uniques pour tous les conteneurs et ressources d'un projet.

Pour exécuter plusieurs copies d'un projet, vous pouvez définir un nom de projet personnalisé à l'aide de la variable d'environnement `COMPOSE_PROJECT_NAME` ou de l'option de ligne de commande `-p`.

**⑤ Est-ce que je peux utiliser « json » au lieu de « yaml » pour mon fichier Compose ?**

Oui. Yaml est un sur-ensemble de json, donc n'importe quel fichier JSON devrait être valide pour Yaml.

**⑥ Lequel des éléments suivants est un fichier qui contient toutes les commandes qu'un utilisateur pourrait appeler sur la ligne de commande pour créer une image ?**

- Dockerfile

**⑦ Vous ne pouvez pas créer plusieurs conteneurs à partir de la même image.**

- Faux

**⑧ Comment pouvez-vous vous assurer que vos conteneurs Docker et leurs données sont sauvegardés en toute sécurité ?**

- En configurant le mappage de volume sur tous les conteneurs pour stocker leurs données dans un seul emplacement sur le serveur (par exemple, `/data/`) et en sauvegardant cet emplacement.

**⑨ Est-il possible de contrôler l'ordre de démarrage des services avec Docker Compose ?**

Oui, il est possible de contrôler l'ordre de démarrage et d'arrêt des services avec l'option `depends_on`.

Compose démarre et arrête toujours les conteneurs dans l'ordre des dépendances, où les dépendances sont déterminées par `depends_on`, `links`, `volumes_from` et `network_mode` : "service :..." (Docker Inc, 2013-2021).



# 3

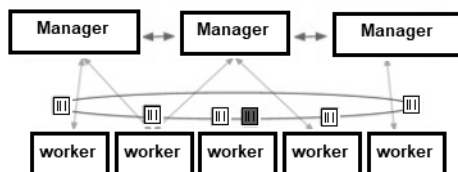
## Docker Swarm

---

Un Swarm est un groupe de machines exécutant le moteur Docker et faisant partie du même cluster. Docker Swarm vous permet de lancer des commandes Docker auxquelles vous êtes habitué sur un cluster depuis une machine maître nommée « manager/leader Swarm ».

Quand des machines rejoignent un Swarm, elles sont appelées « nœuds de travail » (*workers* en anglais).

**Figure 3-1**  
Architecture du mode Swarm



- Les nœuds de travail reçoivent des tâches (conteneurs) distribuées par les managers.
- Les managers peuvent également être des workers.

Les managers Swarm sont les seules machines du Swarm qui peuvent exécuter des commandes Docker ou autoriser d'autres machines à se joindre au Swarm en tant que workers.

Les workers ne sont là que pour fournir de la capacité et n'ont pas le pouvoir d'ordonner à une autre machine ce qu'elle peut ou ne peut pas faire.

Jusqu'à présent, vous utilisiez Docker en mode hôte unique sur votre ordinateur local. Mais Docker peut également être basculé en mode Swarm, ce qui permet l'utilisation des commandes liées au Swarm.

L'activation du mode Swarm sur un hôte Docker transforme instantanément la machine actuelle en manager Swarm. À partir de ce moment, Docker exécute les commandes que vous effectuez sur le Swarm que vous gérez, plutôt que sur la seule machine en cours.

## Créer un cluster Swarm

Pour activer le mode Swarm sur une machine, tapez la commande suivante :

```
[node1] (local) root@192.168.0.18 ~  
$ docker swarm init --advertise-addr 192.168.0.18
```

Cela a pour effet d'initialiser le Swarm. Le moteur Docker ciblé par cette commande devient un gestionnaire dans le Swarm à nœud unique nouvellement créé.

L'option `--advertise-addr` spécifie l'adresse qui sera annoncée aux autres membres du Swarm pour l'accès aux API et la mise en réseau `overlay`. Si cette option n'est pas spécifiée, Docker vérifiera si le système a une seule adresse IP et utilisera cette dernière avec le port d'écoute.

Si le système a plusieurs adresses IP, l'option `--advertise-addr` doit être spécifiée afin que l'adresse correcte soit choisie pour la communication entre gestionnaires et la mise en réseau `overlay`.

Résultat :

```
Swarm initialized : current node (kk4h8vk1vv3y3u70r9wccwe31) is now a manager.  
  
To add a worker to this swarm, run the following command :  
    docker swarm join --token SWMTKN-1-  
    2u7rdhw4bh6m8u2vne1vq36uv2xzitb51peuo05w3qa8b2ep8y-6wuqe9tyk67rrxj96uae65jur  
    192.168.0.18 :2377  
  
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the  
instructions.
```

Votre cluster Swarm est prêt maintenant à accueillir de nouvelles machines. De plus, le résultat nous indique clairement comment ajouter une machine au Swarm (manager ou worker).

`docker swarm init` génère deux jetons aléatoires : un jeton de worker et un jeton de manager. Lorsque vous ajoutez un nouveau nœud au Swarm, celui-ci se joint en tant que nœud de travail ou de manager en fonction du jeton que vous passez à la jointure du Swarm.

Ajoutez deux nœuds workers à ce Swarm et exécutez la commande suivante sur chacune des deux machines :

```
[node2] (local) root@192.168.0.17 ~  
$ docker swarm join --token SWMTKN-1-  
    2u7rdhw4bh6m8u2vne1vq36uv2xzitb51peuo05w3qa8b2ep8y-6wuqe9tyk67rrxj96uae65jur  
    192.168.0.18 :2377  
This node joined a swarm as a worker.
```



```
[node3] (local) root@192.168.0.16 ~
$ docker swarm join -token SWMTKN-1-
2u7rdhw4bh6m8u2vne1vq36uv2xzitb5lpeuo05w3qa8b2ep8y-6wuqe9tyk67rrxj96uae65jur
192.168.0.18 :2377
This node joined a swarm as a worker.
```

Pour ajouter un manager à ce Swarm, exécutez la commande suivante sur le manager actuel :

```
[node1] (local) root@192.168.0.18 ~
$ docker swarm join-token manager
```

Résultat :

```
To add a manager to this swarm, run the following command :
    docker swarm join -token SWMTKN-1-
2u7rdhw4bh6m8u2vne1vq36uv2xzitb5lpeuo05w3qa8b2ep8y-bqkaynfnax7ny3psa6a0baq
8u 192.168.0.18 :2377
```

Dans un environnement de production, il faut configurer au minimum trois managers pour la haute disponibilité. Ajoutez donc deux autres managers comme suit :

```
[node4] (local) root@192.168.0.15 ~
$ docker swarm join -token SWMTKN-1-
2u7rdhw4bh6m8u2vne1vq36uv2xzitb5lpeuo05w3qa8b2ep8y-bqkaynfnax7ny3psa6a0baq8u
192.168.0.18 :2377
This node joined a swarm as a manager.
```

```
[node5] (local) root@192.168.0.14 ~
$ docker swarm join --token SWMTKN-1-
2u7rdhw4bh6m8u2vne1vq36uv2xzitb5lpeuo05w3qa8b2ep8y-bqkaynfnax7ny3psa6a0baq8u
192.168.0.18:2377
This node joined a swarm as a manager.
```

Voici la commande qui permet d'afficher les différents nœuds de votre Swarm :

```
[node1] (local) root@192.168.0.18 ~
$ docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
ENGINE VERSION				
kk4h8vk1vv3y3u70r9wccwe31 *	node1	Ready	Active	Leader
20.10.0				
712uw72pa8t09tnpoecup1jzk	node2	Ready	Active	
20.10.0				
wg9u65pbx0tk4vvpnhssputk1	node3	Ready	Active	
20.10.0				
nix59u10sdzhktpmzxli9n883	node4	Ready	Active	Reachable
20.10.0				
t47sv1gkr0tti12gdb6p00yhc	node5	Ready	Active	Reachable
20.10.0				

Le résultat indique dans la colonne `MANAGER STATUS` le type de chaque nœud.

## Déployer une application à service individuel dans un Swarm avec Docker Service

La commande `docker service` est utilisée lors de la gestion d'un service individuel dans un cluster Swarm.

Voici un aperçu de ce qu'un service peut définir comme comportement et état d'un conteneur :

- Le nom de l'image et le tag que les conteneurs du nœud doivent exécuter.
- Combien de conteneurs participent au service.
- Les ports à exposer à l'extérieur du cluster Swarm.
- Comment doit agir le conteneur à la suite d'une erreur.
- Les caractéristiques des nœuds sur lesquels le service peut s'exécuter (telles que des contraintes de ressources et/ou de préférence de placement sur tel ou tel nœud).
- Etc.

La commande suivante crée un service (en se basant sur l'image `nginx`) avec ses options respectant les caractéristiques définies précédemment :

```
[node1] (local) root@192.168.0.18 ~
$ docker service create --name nginx-app --replicas=3 --publish
published=8080,target=80 --restart-condition=on-failure --limit-memory=100M nginx
```

Résultat :

```
103430sk05pwsao0i78iv51ou
overall progress: 3 out of 3 tasks
1/3: running [=====>]
2/3: running [=====>]
3/3: running [=====>]
verify: Service converged
```

Vous pouvez connaître l'état d'avancement du service dans votre Swarm en lançant la commande suivante :

```
[node1] (local) root@192.168.0.18 ~
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
103430sk05pw	nginx-app	replicated	3/3	nginx:latest	*:8080->80/tcp

Vous pouvez également lister les différentes tâches de votre service afin de vérifier, par exemple, sur quel nœud votre tâche s'est exécutée.

```
[node1] (local) root@192.168.0.18 ~
$ docker service ps nginx-app
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
ERROR	PORTS				
iipwsit58okr	nginx-app.1	nginx:latest	node3	Running	Running 2 minutes ago

24ugb46iffba minutes ago	nginx-app.2	nginx:latest	node4	Running	Running 2
itr6sq6qhgrw minutes ago	nginx-app.3	nginx:latest	node2	Running	Running 2

Actuellement, nous avons trois conteneurs Nginx tournant dans notre Swarm, tous cyclant de manière aléatoire, grâce à un équilibreur de charge créé automatiquement par Swarm.

Accédez à n'importe quelle adresse des cinq machines sur le port 8080 pour tester le service.

Dans l'exemple suivant, je demande le service en tapant l'adresse du nœud 5, 192.168.0.14, qui n'exécute pas un conteneur Nginx (vous pouvez le vérifier dans le résultat de la commande précédente) et pourtant le service est fonctionnel :

```
[node1] (local) root@192.168.0.18 ~
$ curl 192.168.0.14:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Pour tester l'équilibreur de charge, nous allons déployer un service en nous basant sur une image qui affiche le hostname du conteneur. Par défaut le hostname du conteneur correspond à son ID.

Mais avant, nous allons supprimer le service Nginx.

```
[node1] (local) root@192.168.0.18 ~
$ docker service rm nginx-app
nginx-app
[node1] (local) root@192.168.0.18 ~
$ docker service ls
ID            NAME          MODE          REPLICAS    IMAGE           PORTS
[node1] (local) root@192.168.0.18 ~
```

```
[node1] (local) root@192.168.0.18 ~
$ docker service create --name http-app --replicas 2 -p 80:8080 elyesntc/http-
server:v1
x34bpjua1ttfmluiciest4cwo
overall progress: 2 out of 2 tasks
1/2: running [=====>]
2/2: running [=====>]
verify: Service converged
```

Demandez ce service à partir du nœud 1 deux fois consécutives et notez que ce n'est pas le même conteneur qui répond toujours.

```
[node1] (local) root@192.168.0.18 ~
$ curl 192.168.0.18
<center>Formation présentée par Elyes Gassara<br>Application web <font
color="00FF00"><b>version 1</b> </font><br> Conteneur : <b>49bb75496343[node1]
(local) root@192.168.0.18 ~
$ curl 192.168.0.18
<center>Formation présentée par Elyes Gassara<br>Application web <font
color="00FF00"><b>version 1</b> </font><br> Conteneur : <b>0feaa0dc5695[node1]
(local) root@192.168.0.18 ~
```

Vous pouvez le tester aussi en lançant deux fois la page à partir d'un navigateur :

Figure 3-2



Si une erreur survient sur un conteneur, Swarm relance par défaut le service sur un autre conteneur pour assurer la haute disponibilité.

Allons voir sur quel nœud s'exécutent nos conteneurs puis essayons d'arrêter un conteneur en cours d'exécution.

```
[node1] (local) root@192.168.0.18 ~
$ docker service ps http-app
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
y0twest01etp	http-app.1	elyesntc/http-server:v1	node1	Running	Running 4 minutes ago		
mvjhledn9xpt	http-app.2	elyesntc/http-server:v1	node2	Running	Running 4 minutes ago		

Pour cet exemple, arrêtons Nginx sur le nœud 2.

```
[node2] (local) root@192.168.0.17 ~
$ docker ps
CONTAINER ID   IMAGE                                COMMAND                                  CREATED        STATUS
PORTS         NAMES
a886f809e1a7   elyesntc/http-server:v1            "docker-entrypoint.s..." 6 minutes ago   Up
6 minutes     8080/tcp    http-app.2.mvjhledn9xpt8c7w96l1l1n11m
[node2] (local) root@192.168.0.17 ~
$ docker stop http-app.2.mvjhledn9xpt8c7w96l1l1n11m
http-app.2.mvjhledn9xpt8c7w96l1l1n11m
```

Vérifions à présent l'état de notre service :

```
[node1] (local) root@192.168.0.18 ~
$ docker service ps http-app
ID                NAME          IMAGE                                NODE     DESIRED STATE
CURRENT STATE    ERRO          PORTS
R
y0twest0letp     http-app.1    elyesntc/http-server:v1            node1    Running
Running 7 minutes ago

qlom0brxxj9h     http-app.2    elyesntc/http-server:v1            node4    Running
Running 14 seconds ago

mvjhledn9xpt     \_ http-app.2  elyesntc/http-server:v1            node2    Shutdown
Failed 49 seconds ago
k: non-zero exit (137)"
```

Le conteneur a été relancé sur le nœud 4. Notez également que le manager peut jouer le rôle de worker.

Affichez plus de détails sur votre service en utilisant la commande suivante :

```
[node1] (local) root@192.168.0.18 ~
$ docker service inspect --pretty http-app

ID:                x34bpjua1ttfmliuciest4cwo
Name:              http-app
Service Mode:      Replicated
  Replicas:        2
Placement:
UpdateConfig:
  Parallelism:     1
  On failure:      pause
  Monitoring Period: 5s
  Max failure ratio: 0
  Update order:    stop-first
RollbackConfig:
  Parallelism:     1
  On failure:      pause
  Monitoring Period: 5s
  Max failure ratio: 0
```

```
Rollback order:    stop-first
ContainerSpec:
  Image:            elyesntc/http-
server:v1@sha256:e8318ced1cd4e5e2a2cd4244f61927f16fad59f34a4deca312d5caff0636960a
  Init:            false
Resources:
Endpoint Mode:    vip
Ports:
  PublishedPort = 80
  Protocol = tcp
  TargetPort = 8080
  PublishMode = ingress
```

Pour examiner l'état des conteneurs vous pouvez aussi utiliser cette commande :

```
[node1] (local) root@192.168.0.18 ~
$ docker node ps self
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	PORTS
y0twest0letp	http-app.1	elyesntc/http-server:v1	node1	Running		

```
Running 11 minutes ago
```

```
[node1] (local) root@192.168.0.18 ~
$ docker node ps node4
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	PORTS
qlom0brxxj9h	http-app.2	elyesntc/http-server:v1	node4	Running		

```
Running 4 minutes ago
```

Si votre application requiert encore plus de puissance, voici la commande qui permet de « scaler » (mettre à l'échelle) automatiquement vos conteneurs sans les redémarrer :

```
[node1] (local) root@192.168.0.18 ~
$ docker service scale http-app=5
http-app scaled to 5
overall progress: 5 out of 5 tasks
1/5: running [=====>]
2/5: running [=====>]
3/5: running [=====>]
4/5: running [=====>]
5/5: running [=====>]
verify: Service converged
```

Vérifions maintenant le nombre de tâches s'exécutant sur les nœuds afin de s'assurer du nombre de répliques :

```
[node1] (local) root@192.168.0.18 ~
$ docker service ps http-app
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE
swarm-1	http-app	http-app	node1	Running	Running

ERROR		PORTS		
y0twest0letp	http-app.1	elyesntc/http-server:v1	node1	Running
Running 14 minutes ago				
qlom0brxxj9h	http-app.2	elyesntc/http-server:v1	node4	Running
Running 7 minutes ago				
mvjhledn9xpt	\_ http-app.2	elyesntc/http-server:v1	node2	Shutdown
Failed 7 minutes ago				
"task: non-zero exit (137)"				
1l2g1zd8s99k	http-app.3	elyesntc/http-server:v1	node5	Running
Running about a minute ago				
r2xbaap3glu1	http-app.4	elyesntc/http-server:v1	node2	Running
Running about a minute ago				
38f1hfntx74d	http-app.5	elyesntc/http-server:v1	node3	Running
Running about a minute ago				

Il est possible de mettre à jour votre application depuis votre nouvelle image sans interruption de service.

Dans cet exemple, nous allons simuler une nouvelle version de notre application grâce à la nouvelle image que j'ai déposée dans le Docker Hub :

```
[node1] (local) root@192.168.0.18 ~
$ docker service update --update-delay=10s --update-parallelism=1 --image elyesntc/
http-server:v2 http-app
http-app
overall progress: 3 out of 5 tasks
1/5: running  [=====>]
2/5: running  [=====>]
3/5: running  [=====>]
4/5:
```

Notez que j'ai demandé que la mise à jour se fasse conteneur par conteneur et avec un espace-ment de 10 secondes.

Entre-temps, testez que les deux versions sont disponibles en même temps (en actualisant votre navigateur) avant que la mise à jour s'effectue sur tous les conteneurs (voir figure 3-3).

Figure 3-3



Supprimez le service avant de passer à l'étape suivante.

## Virtual IP et Service Discovery

Le mode Docker Swarm comprend un maillage de routage qui permet la mise en réseau multihôte. Il permet aux conteneurs situés sur deux hôtes de communiquer comme s'ils étaient sur le même hôte.

Pour ce faire, Docker Swarm crée un réseau local extensible virtuel (VXLAN), conçu pour les réseaux basés sur le cloud.

Le routage fonctionne de deux manières différentes :

- sur la base du port public exposé sur le service. Toutes les demandes adressées au port seront distribuées ;
- le service reçoit une adresse IP virtuelle qui n'est routable qu'à l'intérieur du réseau Docker.

Lorsque des demandes sont adressées à l'adresse IP virtuelle, elles sont distribuées aux conteneurs sous-jacents. Cette adresse IP est enregistrée auprès du serveur DNS intégré dans Docker.

Lorsqu'une recherche DNS est effectuée en fonction du nom du service, l'adresse IP virtuelle est donc renvoyée.

Dans cette étape, nous allons créer un service qui est attaché à un réseau `overlay` que nous allons créer.

```
[node1] (local) root@192.168.0.18 ~
$ docker network create --attachable -d overlay eg1
mc8bz8ygce43xuuh06v6azplz
```

Ce réseau sera un « réseau à portée de Swarm ». Cela signifie que seuls les conteneurs lancés en tant que services peuvent s'attacher au réseau.

```
[node1] (local) root@192.168.0.18 ~
$ docker service create --name http --network eg1 --replicas 2 elyesntc/http-
server:v1
e19ym4j9pu5rjv8cil17imfcn
overall progress: 2 out of 2 tasks
1/2: running [=====>]
2/2: running [=====>]
verify: Service converged
```

En appelant le service `http`, Docker ajoute une entrée à celui-ci dans un serveur DNS intégré. Les autres conteneurs sur le réseau peuvent utiliser le nom convivial pour découvrir l'adresse IP.

Avec les ports, c'est cette adresse IP qui peut être utilisée à l'intérieur du réseau pour réaliser l'équilibrage de la charge.

Utilisez l'utilitaire `dig` pour trouver l'adresse IP virtuelle interne. En utilisant l'option `--attachable`, un conteneur situé en dehors du service Swarm peut accéder au réseau.



```
[node1] (local) root@192.168.0.18 ~
$ docker run --name=dig --network egl benhall/dig dig http
Unable to find image 'benhall/dig:latest' locally
latest: Pulling from benhall/dig
12b41071e6ce: Pull complete
d23aaa6caac4: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:ed7d241f0faea3a015d13117824c04a433a79032619862e4e3741a31eb9e4272
Status: Downloaded newer image for benhall/dig:latest

; <<>> DiG 9.10.2 <<>> http
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62545
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;http.                                IN      A

;; ANSWER SECTION:
http.                                600      IN      A      10.0.1.2

;; Query time: 0 msec
;; SERVER: 127.0.0.11#53(127.0.0.11)
;; WHEN: Thu Oct 07 18:34:01 UTC 2021
;; MSG SIZE rcvd: 42
```

Le ping du nom devrait également découvrir l'adresse IP.

```
[node1] (local) root@192.168.0.18 ~
$ docker run --name=ping --network egl alpine ping -c5 http
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
a0d0a0d46f8b: Pull complete
Digest: sha256:e1c082e3d3c45ccac829840a25941e679c25d438cc8412c2fa221cf1a824e6a
Status: Downloaded newer image for alpine:latest
PING http (10.0.1.2): 56 data bytes
64 bytes from 10.0.1.2: seq=0 ttl=64 time=0.997 ms
64 bytes from 10.0.1.2: seq=1 ttl=64 time=0.117 ms
64 bytes from 10.0.1.2: seq=2 ttl=64 time=0.098 ms
64 bytes from 10.0.1.2: seq=3 ttl=64 time=0.107 ms
64 bytes from 10.0.1.2: seq=4 ttl=64 time=0.121 ms

--- http ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.098/0.288/0.997 ms
```

Cela doit correspondre à l'adresse IP virtuelle donnée au service. Vous pouvez la découvrir aussi en inspectant le service.

```
[node1] (local) root@192.168.0.18 ~
$ docker service inspect http --format="{{.Endpoint.VirtualIPs}}"
[{{mc8bz8ygce43xuuh06v6azplz 10.0.1.2/24}}
```

À part cette adresse du service, chaque conteneur reste toujours attribué à une adresse IP unique.

```
[node3] (local) root@192.168.0.16 ~
$ docker inspect --format="{{.NetworkSettings.Networks.eg1.IPAddress}}" $(docker
ps|grep http-server |head -n1 |awk '{print $1}')
10.0.1.4
```

Cette adresse IP virtuelle garantit que l'équilibrage de charge fonctionne comme prévu dans le cluster. L'adresse IP garantit quant à elle qu'il fonctionne en dehors du cluster.

## Déployer une application multiservice dans un Swarm

L'IP virtuelle, l'équilibrage de charge ainsi que la découverte de service peuvent être utilisés dans un scénario multihôte avec applications communiquant avec différents services sur plusieurs hôtes.

Dans cette étape, nous déploierons une application Node.js répliquée qui communique avec Redis pour stocker des données.

Commençons par créer un réseau `overlay` sur lequel nous attacherons nos services :

```
[node1] (local) root@192.168.0.18 ~
$ docker network create -d overlay app1-network
yfivwtmf0ctnk0u1www8se7ck
```

Lors du déploiement de Redis, le réseau peut être connecté. L'application s'attend à pouvoir se connecter à une instance Redis, nommée `Redis`.

Pour permettre à l'application de découvrir l'adresse IP virtuelle via le DNS intégré, nous appelons le service `redis`.

```
[node1] (local) root@192.168.0.18 ~
$ docker service create --name redis --network app1-network redis:alpine
slesutlvu64a3fk635u4elpak
overall progress: 1 out of 1 tasks
1/1: running [=====>]
verify: Service converged
```

Lors du déploiement de l'application, un port public peut être exposé, ce qui lui permet d'équilibrer la charge des requêtes entre les quatre conteneurs.

```
[node1] (local) root@192.168.0.18 ~
$ docker service create --name app1-web --network app1-network --replicas 4 -p
80:8080 elyesntc/nodejs-redis-app
wscsewpz7egy61vxvlhm2k6vh
overall progress: 4 out of 4 tasks
1/4: running  [=====>]
2/4: running  [=====>]
3/4: running  [=====>]
4/4: running  [=====>]
verify: Service converged
```

Accédez à ce service plusieurs fois pour tester l'incrémentation du nombre de visites et la répartition de charge sur les quatre conteneurs.

```
[node1] (local) root@192.168.0.18 ~
$ curl 192.168.0.18
<center><br> <font color="0000FF">Formation animée par Elyes Gassara </
font><br>Cette requête a été traitée par 1
e conteneur <b>f111ff551898 </b><br> Vous avez vu cette page <b> <font
color="0F000F">1 fois.</font></b> <br>[nod
e1] (local) root@192.168.0.18 ~
$ curl 192.168.0.18
<center><br> <font color="0000FF">Formation animée par Elyes Gassara </
font><br>Cette requête a été traitée par 1
e conteneur <b>cebe6dd86e3e </b><br> Vous avez vu cette page <b> <font
color="0F000F">2 fois.</font></b> <br>[nod
e1] (local) root@192.168.0.18 ~
$ curl 192.168.0.18
<center><br> <font color="0000FF">Formation animée par Elyes Gassara </
font><br>Cette requête a été traitée par 1
e conteneur <b>fdce4c550b4e </b><br> Vous avez vu cette page <b> <font
color="0F000F">3 fois.</font></b> <br>[nod
e1] (local) root@192.168.0.18 ~
```

## Déployer une application multiservice dans un Swarm avec Docker Stack

Docker Stack peut être utilisé pour gérer une application multiservice dans votre cluster Swarm.

Pour faire simple, vous pouvez considérer que la commande `docker service` est identique à la commande `docker run` et que la commande `docker stack` est comparable à la commande `docker-compose`. Nous allons utiliser un nœud manager pour déployer notre application Docker multiservice dans notre cluster Swarm.

Comme pour la création de notre service précédemment, nous allons définir les différentes caractéristiques de nos conteneurs :

- Deux API sous forme de deux services différents.
- Trois conteneurs pour le service `http` doivent être exécutés dans cet exemple.
- Redémarrer un service s'il se ferme à la suite d'une erreur.
- Limiter l'utilisation de la mémoire à 50 Mo.

Nous n'avons pas besoin d'apprendre de nouvelles choses pour créer nos services. Nous allons réutiliser les connaissances vues dans la partie précédente puisqu'il suffit de créer un fichier `docker-compose.yml` :

```
version: "3"
services:
  http:
    image: elyesntc/nodejs-redis-app
    deploy:
      replicas: 3
      resources:
        limits:
          memory: 50M
    ports:
      - "80:8080"
    depends_on:
      - redis
    networks:
      - frontend
      - backend

  redis:
    image: redis:alpine
    deploy:
      replicas: 1
      resources:
        limits:
          memory: 50M
    networks:
      - backend
    volumes:
      - elyesvolume:/data
networks:
  frontend:
  backend:
volumes:
  elyesvolume:
```

Déployons à présent l'application avec la commande `docker stack` en donnant comme arguments le fichier `docker-compose` et le nom de notre application.

```
[node1] (local) root@192.168.0.18 ~
$ docker stack deploy -c docker-compose.yml http
Creating network http_frontend
Creating network http_backend
Creating service http_redis
Creating service http_http
```

Vérifiez que deux services sont bien créés (`http` et `redis`).

```
[node1] (local) root@192.168.0.18 ~
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE
finrqmvdvrv2	http_http	replicated	3/3	elyesntc/nodejs-redis-app:latest
*:80->8080/tcp				
zflx1y05422b	http_redis	replicated	1/1	redis:alpine

Vous pouvez aussi vérifier que vous avez bien trois conteneurs `http` et un seul conteneur `redis` comme indiqué dans le fichier `docker-compose` (voir le nombre de répliques dans chaque déploiement).

```
[node1] (local) root@192.168.0.18 ~
$ docker stack ps http
```

ID	NAME	IMAGE	NODE	DESIRED
pagwh8avqg53	http_http.1	elyesntc/nodejs-redis-app:latest	node2	Running
Running 2 minutes ago				
mmp1m4zhdrjc	http_http.2	elyesntc/nodejs-redis-app:latest	node1	Running
Running 2 minutes ago				
dtmip7tc2940	http_http.3	elyesntc/nodejs-redis-app:latest	node4	Running
Running 2 minutes ago				
k16w5bv9grtc	http_redis.1	redis:alpine	node1	Running
Running 2 minutes ago				

Examinez les journaux des conteneurs pour vérifier leur fonctionnement normal.

```
[node1] (local) root@192.168.0.18 ~
$ docker logs http_redis.1.k16w5bv9grtc9ztgbnii9c0xk
1:C 07 Oct 2021 19:31:05.516 # o000o000o000o Redis is starting o000o000o000o
1:C 07 Oct 2021 19:31:05.516 # Redis version=6.2.6, bits=64, commit=00000000,
modified=0, pid=1, just started
1:C 07 Oct 2021 19:31:05.516 # Warning: no config file specified, using the default
config. In order to specify a
config file use redis-server /path/to/redis.conf
1:M 07 Oct 2021 19:31:05.517 * monotonic clock: POSIX clock_gettime
1:M 07 Oct 2021 19:31:05.518 * Running mode=standalone, port=6379.
```

```
1:M 07 Oct 2021 19:31:05.518 # WARNING: The TCP backlog setting of 511 cannot be
enforced because /proc/sys/net/c
ore/somaxconn is set to the lower value of 128.
1:M 07 Oct 2021 19:31:05.518 # Server initialized
1:M 07 Oct 2021 19:31:05.518 * Ready to accept connections
```

```
[node2] (local) root@192.168.0.17 ~
$ docker logs http_http.1.pagwh8avqg5300vk56rcht3h
npm info it worked if it ends with ok
npm info using npm@3.8.3
npm info using node@v5.10.1
npm info lifecycle sample_nodejs_with_redis@1.0.0~prestart:
sample_nodejs_with_redis@1.0.0
npm info lifecycle sample_nodejs_with_redis@1.0.0~start:
sample_nodejs_with_redis@1.0.0

> sample_nodejs_with_redis@1.0.0 start /usr/src/app
> node server.js

Listening on port 8080
```

Arrêtez par exemple la base de données `redis`.

```
[node1] (local) root@192.168.0.18 ~
$ docker stop http_redis.1.kl6w5bv9grtc9ztgbnii9c0xk
http_redis.1.kl6w5bv9grtc9ztgbnii9c0xk
```

Affichez à nouveau les journaux du conteneur `http` et notez le message d'erreur suite à l'interruption de la connexion avec la base.

```
[node2] (local) root@192.168.0.17 ~
$ docker logs http_http.1.pagwh8avqg5300vk56rcht3h
```

```
events.js:154
    throw er; // Unhandled 'error' event
    ^

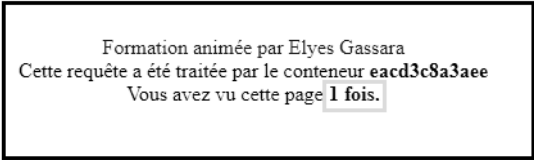
Error: Redis connection to redis:6379 failed - connect EHOSTUNREACH 10.0.4.9:6379
    at Object.exports._errnoException (util.js:890:11)
    at exports._exceptionWithHostPort (util.js:913:20)
    at TCPConnectWrap.afterConnect [as oncomplete] (net.js:1059:14)
```

La haute disponibilité assure le relancement de la base de données après l'arrêt du conteneur et le service `http` est de nouveau fonctionnel.

```
[node1] (local) root@192.168.0.18 ~
$ docker stack ps http
ID                NAME                IMAGE                NODE
DESIRED STATE    CURRENT STATE      ERROR                PORTS
ozd06id0av25     http_http.1        elyesntc/nodejs-redis-app:latest node2
Running          Running 6 minutes ago
pagwh8avqg53     \_ http_http.1     elyesntc/nodejs-redis-app:latest node2
Shutdown        Failed 6 minutes ago "task: non-zero exit (1)"
ntom3xteu3vm     http_http.2        elyesntc/nodejs-redis-app:latest node3
Running          Running 6 minutes ago
mmp1m4zhdrjc     \_ http_http.2     elyesntc/nodejs-redis-app:latest node1
Shutdown        Failed 6 minutes ago "task: non-zero exit (1)"
4q48qoudxkgp     http_http.3        elyesntc/nodejs-redis-app:latest node4
Running          Running 6 minutes ago
dtmip7tc2940     \_ http_http.3     elyesntc/nodejs-redis-app:latest node4
Shutdown        Failed 6 minutes ago "task: non-zero exit (1)"
z25lmyo5f7wq     http_redis.1       redis:alpine        node5
Running          Running 6 minutes ago
kl6w5bv9grtc     \_ http_redis.1    redis:alpine        node1
Shutdown        Complete 6 minutes ago
```

Il faut penser à ajouter un stockage persistant si vous voulez garder le nombre de visites même en cas de redémarrage de la base.

Figure 3-4



# Conclusion

Ce chapitre vous a montré la simplicité d'utilisation du Docker Swarm. L'un de ses plus gros avantages est qu'il vous permet de vous concentrer sur le développement de votre application et de ne pas vous soucier de l'endroit où cette dernière va s'exécuter. Par ailleurs, les services sont constamment monitorés par Swarm.

En plus du monitoring, Docker Swarm se charge également de la réparation automatique, vous aidant ainsi à garder les services du cluster en bon état de fonctionnement en comparant en permanence l'état désiré avec l'état actuel.

## Validation des acquis

### Questions :

- ❶ Qu'est-ce que Docker Swarm ?
  - un outil d'orchestration de conteneurs ;
  - un document texte qui contient toutes les commandes ;
  - un registre de conteneurs.
- ❷ Qu'est-ce que le mode Swarm de Docker ?
  - Swarm est le nom de la dernière version de Docker ;
  - Docker Swarm est également connu sous le nom de Docker Hub ;
  - Swarm est un groupe de machines qui exécutent Docker et qui sont réunies en un cluster ;
  - Docker Swarm est un outil pour gérer vos pipelines CI/CD.
- ❸ Quel est le nombre recommandé de nœuds de gestion dans Docker Swarm ?
  - 0 manager ;
  - le plus grand nombre possible ;
  - 1 manager ;
  - 3 ou 5 managers.
- ❹ Comment récupérer le jeton de jointure pour les nœuds de gestionnaire dans Docker Swarm ?
  - `docker get join-token manager`
  - `docker swarm join-token manager`
  - `docker print join-token manager`
  - `docker swarm token manager`
- ❺ Comment lister tous les nœuds dans votre cluster Docker Swarm ?
  - `docker node get`
  - `docker node get-all`
  - `docker node print`
  - `docker node ls`
- ❻ Comment lister tous vos services dans Docker Swarm ?
  - `docker ls service`
  - `docker service ps`
  - `docker service ls`
  - `docker get service`
- ❼ Comment mettre à l'échelle un service à 6 répliques dans Docker Swarm ?
  - `docker service scale nom_du_service=6`
  - `docker service update nom_du_service =6`
  - `docker service replica-set nom_du_service =6`
  - `docker service upgrade nom_du_service =6`



**Réponses :**

- ① **Qu'est-ce que Docker Swarm ?**
  - Docker Swarm est un outil d'orchestration de conteneurs.
- ② **Qu'est-ce que le mode Swarm de Docker ?**
  - Swarm est un groupe de machines qui exécutent Docker et qui sont réunies en un cluster.
- ③ **Quel est le nombre recommandé de nœuds de gestion dans Docker Swarm ?**
  - 3 ou 5 managers.
- ④ **Comment récupérer le jeton de jointure pour les nœuds de gestionnaire dans Docker Swarm ?**
  - `docker swarm join-token manager`
- ⑤ **Comment lister tous les nœuds dans votre cluster Docker Swarm ?**
  - `docker node ls`
- ⑥ **Comment lister tous vos services dans Docker Swarm ?**
  - `docker service ls`
- ⑦ **Comment mettre à l'échelle un service à 6 répliques dans Docker Swarm ?**
  - `docker service scale nom_du_service=6`



# 4

## Kubernetes

---

Docker Compose permet de regrouper et d'orchestrer un ensemble de conteneurs sur une même machine. Il va lancer les conteneurs et leurs éventuels liens à partir d'un fichier de configuration écrit en langage Yaml. Mais comment automatiser et industrialiser le déploiement, la montée en charge et la gestion des applications fonctionnant dans des conteneurs ?

C'est à cette question que Kubernetes (K8S) se propose de répondre.

Kubernetes est une solution open source créée par Google et c'est actuellement le leader du marché de l'orchestration de conteneurs. Il est maintenant disponible en version managée chez les grands fournisseurs du cloud comme Azure Kubernetes Service (Microsoft), Google Kubernetes Engine ou Amazon Elastic Container Service for Kubernetes. Il peut également être installé sur des parcs de machines dédiées (sur site).

Kubernetes et Docker Swarm sont deux des plates-formes open source les plus utilisées et offrent des fonctionnalités très similaires. Cependant, en y regardant de plus près, on peut remarquer plusieurs différences fondamentales en ce qui concerne le fonctionnement de ces deux plates-formes.

### Kubernetes et Docker Swarm

Le tableau 4-1 présente les principales différences entre Kubernetes et Docker Swarm.

Tableau 4-1 Kubernetes versus Docker Swarm

Docker Swarm	Kubernetes
Pas de mise à l'échelle automatique	Mise à l'échelle automatique
Équilibrage automatique de la charge	Configuration manuelle des paramètres d'équilibrage de charge
Mise à l'échelle plus rapide que Kubernetes, mais la puissance du cluster n'est pas aussi robuste	Mise à l'échelle lente par rapport à Docker, mais garantit un état de cluster plus fort
Démarrage d'un cluster facile	Démarrage d'un cluster difficile
Limitation aux capacités de l'API Docker	Peut surmonter les contraintes de Docker et de l'API Docker
N'a pas une aussi grande expérience des déploiements de production à grande échelle	Déployé à grande échelle au sein des organisations

Pour bien comprendre Kubernetes, vous devez avoir une bonne compréhension du fonctionnement de Docker, de la création des images Docker et de leur fonctionnement en tant qu'unités autonomes.

Si cela n'est pas encore bien clair pour vous, il est vivement recommandé de reprendre les activités proposées dans les précédents chapitres avant de continuer.

Pour parvenir à une configuration avancée dans Kubernetes, il faut aussi comprendre les bases de la mise en réseau et le fonctionnement des protocoles de communication.

Dans ce chapitre, nous verrons donc :

- l'intérêt et le contexte d'utilisation de Kubernetes ;
- les différents composants de l'architecture de Kubernetes ;
- les concepts de base liés à cet orchestrateur ;
- comment déployer des applications multiconteneurs sur Kubernetes.

## Présentation de l'architecture de Kubernetes

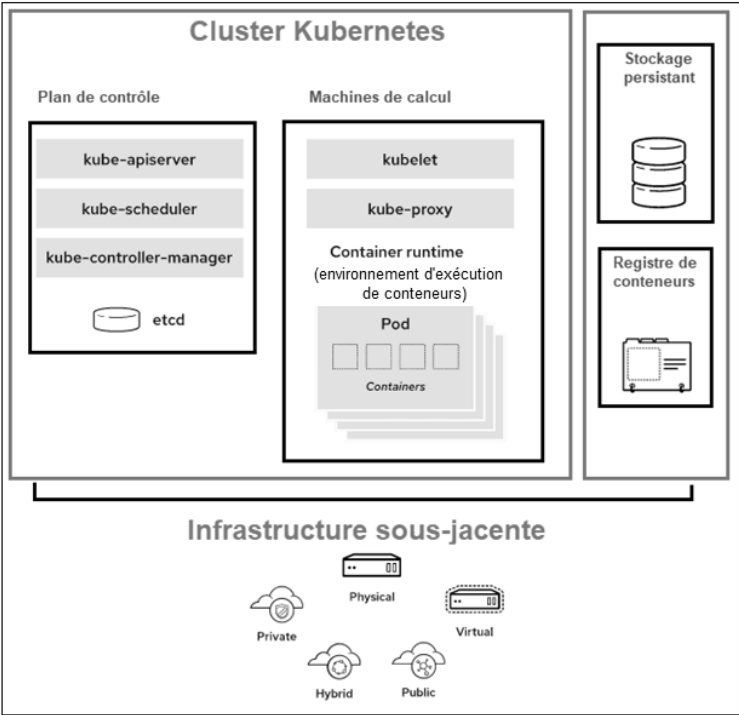
Un cluster informatique, cluster de serveurs ou grappe de serveurs, est un groupe de serveurs indépendants fonctionnant comme un seul et même système. Ainsi, les clusters permettent de profiter d'une ressource de traitement de données centralisée. Un client dialogue avec le groupe de serveurs comme s'il s'agissait d'une seule machine.

Un cluster Kubernetes est généralement déployé sur plusieurs nœuds (*nodes*) : au moins un maître et plusieurs nœuds.

Kubernetes peut fonctionner sur de nombreux types d'infrastructures et vous pouvez choisir son environnement d'exécution : serveurs physiques, machines virtuelles ou clouds (publics, privés et hybrides).

Kubernetes fournit plusieurs couches de fonctionnalités de mise à l'échelle automatique : mise à l'échelle horizontale et verticale basée sur les POD, ainsi que mise à l'échelle basée sur les nœuds.

**Figure 4–1**  
Présentation de l’architecture  
de Kubernetes



**Tableau 4–2** Mise à l’échelle automatique dans Kubernetes

	Mise à l’échelle horizontale	Mise à l’échelle verticale
<b>Pod</b>	Ajoute ou retire des pods	Modifie les ressources CPU et/ou RAM allouées au pod
<b>Nœud</b>	Ajoute ou retire des nœuds	Modifie les ressources CPU et/ou RAM allouées au nœud

Un maître héberge le plan de contrôle Kubernetes : un ensemble de services qui administrent et orchestrent l’ensemble du cluster. Ces services se trouvent sous la forme de pods dans l’espace de nommage (*namespace*) *kube-system*.

À des fins de test, j’utilise Kubernetes en local avec Minikube. La commande suivante permet d’afficher les pods dans cet espace de nommage.

```
C:\Users\Elyes>kubect1 get pods --namespace kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-78fcd69978-nq6wh	1/1	Running	1 (20h ago)	20h
etcd-minikube	1/1	Running	1 (20h ago)	20h
kube-apiserver-minikube	1/1	Running	1 (20h ago)	20h
kube-controller-manager-minikube	1/1	Running	1 (20h ago)	20h
kube-proxy-zfwvk	1/1	Running	1 (20h ago)	20h
kube-scheduler-minikube	1/1	Running	1 (20h ago)	20h
storage-provisioner	1/1	Running	3 (38s ago)	20h

## Plan de contrôle

Le plan de contrôle contient les composants Kubernetes qui contrôlent le cluster, ainsi que des données sur l'état et la configuration du cluster.

Il est en contact permanent avec vos machines de calcul et ses composants s'assurent que suffisamment de conteneurs peuvent fonctionner avec les ressources nécessaires.

### kube-apiserver

Le serveur API est la partie centrale du plan de contrôle Kubernetes. Il s'agit d'une API REST qui est le point d'entrée pour envoyer des commandes au cluster.

Une API REST est un service web qui utilise l'architecture REST (*Representational State Transfer*) pour traiter une demande sur un service web *frontend*.

L'API Kubernetes vous permet d'interroger et de manipuler l'état des objets API dans Kubernetes (par exemple : Pods, Namespaces, ConfigMaps et Events). La plupart des opérations peuvent être effectuées via l'interface `kubectl` de ligne de commande ou d'autres outils de ligne de commande, tels que `kubeadm`, qui utilise aussi l'API. Toutefois, vous pouvez également accéder directement à l'API à l'aide d'appels REST (The Linux Foundation, 2020).

### kube-scheduler

Le *scheduler*, « planificateur » en français, surveille les ressources disponibles sur les différents nœuds et s'assure que la charge de travail est répartie uniformément sur l'ensemble du cluster.

En effet, lorsque le scheduler découvre un pod, il est en charge de trouver le meilleur nœud sur lequel ce pod doit être exécuté. Il vérifie si un nœud candidat détient suffisamment de ressources disponibles pour répondre aux demandes de ressources spécifiques au pod et élabore une liste avec tous les nœuds appropriés. Il attribue un score à chaque nœud à partir de cette liste et assigne ensuite le pod au nœud ayant le rang le plus élevé.

Plusieurs facteurs sont pris en compte pour les décisions de planification comme les exigences individuelles et collectives en ressources, les contraintes (matérielles, politiques, etc.), les spécifications d'affinité qui fournissent des fonctionnalités avancées pour limiter le placement de pods sur des nœuds spécifiques, l'emplacement des données, les interférences entre charges de travail et les dates limites.

### ETCD

Il s'agit d'une base de données clé-valeur distribuée, résistante aux pannes et utilisée comme mémoire de sauvegarde pour toutes les données du cluster.

C'est dans cette base de données que sont stockées les informations d'identification requises pour authentifier les requêtes que vous envoyez à l'API.

Elle permet donc de stocker de manière fiable les données de configuration du cluster, représentant l'état du cluster à n'importe quel instant.

Dans un environnement de production, il est conseillé d'avoir un cluster ETCD pour assurer la haute disponibilité.

### kube-controller-manager

Le Controller Manager gère l'orchestration du cluster. Il se réfère au planificateur pour s'assurer qu'un nombre suffisant de pods est exécuté. En cas de défaillance d'un nœud, et pour garder toujours le nombre de répliques voulu, ilinstanciera de nouveaux pods sur les nœuds restants.

## Nœud Kubernetes

Un nœud est une machine de travail dans Kubernetes (machine virtuelle ou physique). Les nœuds forment une plate-forme de déploiement unique pour les ressources Kubernetes du cluster.

Un cluster Kubernetes requiert au moins un nœud de calcul, mais en général il en contient un grand nombre. Chaque nœud contient les services nécessaires à l'exécution de pods et est géré par les composants du maître. Les services sur un nœud incluent container runtime, Kubelet et kube-proxy.

### Container runtime

Le service container runtime, « environnement d'exécution de conteneurs » en français, est le logiciel responsable de l'exécution des conteneurs. Dans la plupart des cas, il s'agit de Docker, mais Kubernetes supporte également d'autres environnements d'exécution de conteneurs : containerd, cri-o, rktlet ainsi que toute implémentation de Kubernetes CRI (*Container Runtime Interface*).

Dans le monde des conteneurs, la runtime est responsable de la création et du fonctionnement du conteneur. Il y a quelques années, on ne parlait pas d'environnement d'exécution de conteneurs, et le seul outil connu pour lancer et administrer des conteneurs était Docker. Puis est arrivé rkt. L'OCI, pour Open Container Initiative, a ensuite été créée et le standard runC est apparu.

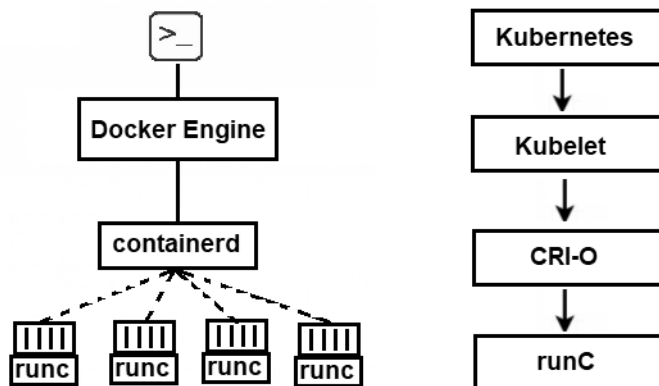
La commande suivante montre que runC est la runtime utilisée par les dernières versions de Docker et qu'il est possible d'avoir, a priori, plusieurs runtimes :

```
$ docker --version
Docker version 20.10.0, build 7287ab3
```

```
$ docker info | grep -i runtime
Runtimes: io.containerd.runtime.v1.linux runc io.containerd.runc.v2
Default Runtime: runc
```

Le rôle de l'OCI est de créer un standard en ce qui concerne la manière de démarrer un conteneur. Ce standard est nommé runC. Lorsque des versions modernes de Docker sont installées, containerd est installé avec lui et CRI communique directement avec containerd.

**Figure 4-2**  
Environnement d'exécution  
de conteneurs



Docker, CRI-O et containerd peuvent tous servir Kubernetes pour le lancement et la maintenance des pods. Ces trois environnements dépendent de runC au niveau le plus bas pour gérer l'exécution des conteneurs.

## Kubelet

Il s'agit d'un agent qui s'exécute sur chaque nœud du cluster et communique au maître l'état du nœud. L'agent Kubelet est également présent sur les maîtres puisqu'il est responsable de la gestion des pods. C'est le service qui interagit avec l'API et applique la configuration des ressources stockées dans l'ETCD sur le nœud. Il prend en charge le démarrage, l'arrêt, et la maintenance des conteneurs d'applications qui sont organisés en pods.

## kube-proxy

Le service kube-proxy s'exécute sur chaque nœud du cluster et fournit l'équilibrage de charge. Il est utilisé pour atteindre les services. Il maintient les règles réseau sur les nœuds en permettant une communication réseau vers les pods depuis des sessions réseau à l'intérieur ou à l'extérieur du cluster.

## Stockage persistant

Un utilisateur peut demander des ressources de stockage, sans nécessairement connaître les détails de l'infrastructure de stockage sous-jacente. Le sous-système PersistentVolume (PV) fournit une API pour les utilisateurs et les administrateurs avec une abstraction des détails liés à ce stockage.

Les PV ont un cycle de vie indépendant de tout pod individuel qui utilise le PV et ils peuvent donc avoir une durée de vie supérieure à celle d'un pod.

Un PersistentVolumeClaim (PVC) est une demande de stockage par un utilisateur. Comme les pods consomment des ressources de nœud, les PVC consomment des ressources PV. Et comme les pods peuvent demander des ressources CPU et mémoire, les PVC peuvent demander une taille et des modes d'accès spécifiques (lecture/écriture, lecture seule).



Les types `PersistentVolume` sont implémentés en tant que plug-ins. Kubernetes prend en charge plusieurs plug-ins, qu'il s'agisse de FC (Fibre Channel), NFS, iSCSI, CephFS et Glusterfs ou un système de stockage spécifique au fournisseur de cloud comme AWS Elastic Block Store (EBS), Disque Azure ou Disque persistant GCE (Google Compute Engine).

## Registre de conteneurs

Les images Docker sont généralement stockées dans un registre d'images pouvant être accessible au public (registre public) ou, au contraire, dont l'accès est limité à un petit groupe d'utilisateurs (registre privé). Actuellement, Docker Hub est le registre de conteneurs public par défaut et le plus largement utilisé.

Les registres privés fournissent un référentiel pour les images personnalisées et couramment utilisées pour une organisation comme Google Cloud Registry (GCR), Amazon Elastic Container Registry (ECR) et Azure Container Registry (ACR).

## Fonctionnalités de Kubernetes

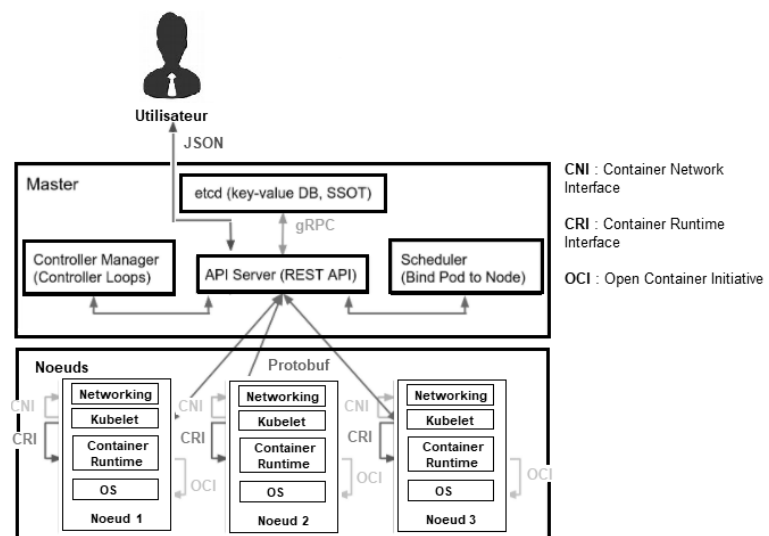
Kubernetes est conçu pour prendre en charge les applications cloud natives modulaires et sa plate-forme est aussi modulaire et flexible.

Il intègre des plug-ins, des modules complémentaires, des services et des interfaces pour étendre les fonctionnalités de base de la plate-forme.

Les extensions sont définies comme des composants qui s'intègrent de manière transparente avec le reste de l'environnement, offrant des fonctionnalités de type natif et étendant les commandes disponibles pour les administrateurs de cluster.

**Figure 4-3**

Schéma général de l'architecture de Kubernetes



## Interfaces Kubernetes

Les plug-ins d'interface étendent les fonctionnalités de Kubernetes ainsi que sa prise en charge de matériel nouveau et personnalisé.

### Plug-ins réseau pour Kubernetes

Il existe deux types de mise en place réseau :

- L'utilisation du réseau k8s proposé par défaut : consiste à créer sur chaque hôte un bridge virtuel avec une plage d'IP, puis à ajouter sur chaque hôte les routes vers les autres hôtes.
- L'utilisation de CNI (*Container Network Interface*) et de ses plug-ins, solution la plus fréquente. CNI est un ensemble de spécifications et bibliothèques (en Go) ayant pour but de faciliter l'intégration de plug-ins réseau. Les trois solutions les plus utilisées sont Calico, Flannel et WeaveNet.

### Runtimes de conteneurs améliorés

Le container runtime est au cœur de chaque environnement Kubernetes. C'est le composant de l'architecture qui organise les ressources matérielles, exécute et arrête les conteneurs, et s'assure que ces derniers reçoivent les ressources dont ils ont besoin pour fonctionner de manière optimale.

L'interface Container Runtime ou les plug-ins CRI sont conçus pour permettre à la nouvelle API CR d'être pleinement utilisée.

Les environnements d'exécution comme Docker peuvent être rendus plus flexibles avec le bon plug-in. Naturellement, les plug-ins CRI offrent un avantage majeur : ils vous permettent d'exécuter différents environnements d'exécution de conteneurs sans avoir à recompiler.

### Plug-ins de volume avec CSI

Kubernetes s'est toujours appuyé sur un système de plug-ins de volume pour gérer les blocs de stockage, mais l'approche n'était pas assez ouverte pour permettre à des outils de gestion tiers de fonctionner sans problème.

CSI (*Container Storage Interface*) est considéré comme la solution palliant cet inconvénient, offrant des volumes CSI et un provisionnement dynamique des blocs de stockage en tant que fonctions.

La principale différence entre les plug-ins CSI et les plug-ins de volume Kubernetes de base est que les plug-ins CSI n'ont pas besoin d'être compilés et livrés avec les binaires Kubernetes de base.

## Résolution DNS

Le service DNS de Kubernetes permet aux pods de communiquer entre eux en utilisant leur nom ou leur FQDN (*Fully Qualified Domain Name*) au lieu de leur IP locale.

La résolution DNS est configurée dans le cluster Kubernetes via CoreDNS. Le Kubelet configure le fichier `/etc/resolv.conf` de chaque pod pour utiliser le pod `coredns` comme serveur de noms.

## Déploiement d'une application

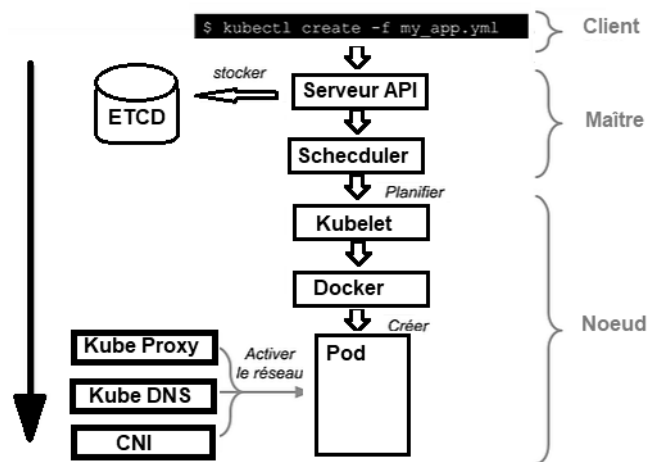
Quand vous déployez une application conteneurisée sur un cluster Kubernetes, vous utilisez la commande `kubectl` pour envoyer la description de votre application et sa configuration à l'API du maître.

L'API stocke cette configuration dans l'ETCD, et le planificateur assigne vos pods d'application aux nœuds.

Kubelet reçoit alors la description de ces pods programmés sur les nœuds et le container runtime les instancie.

L'interface réseau de conteneurs (CNI), le service DNS et proxy de Kubernetes assurent alors que les pods créés ont un accès réseau et peuvent communiquer avec les pods du nœud et à travers le cluster.

**Figure 4-4**  
Déploiement d'une application

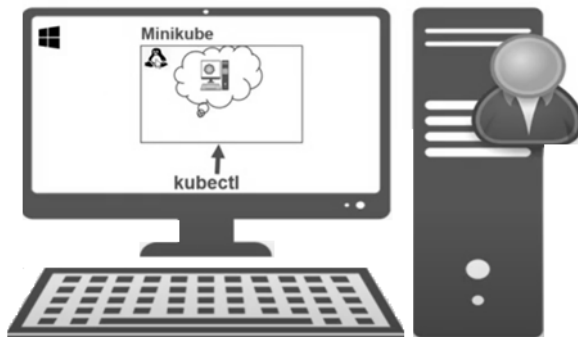


## Minikube

Minikube est une distribution allégée de Kubernetes qui assure une performance maximale pour utiliser les fonctions de Kubernetes tout en profitant d'une charge de travail minimale.

**Figure 4-5**

Minikube



Lancez le cluster sur Minikube :

```
C:\Users\Elyes>minikube start
```

Résultat :

```
* minikube v1.23.2 sur Microsoft Windows 10 Pro 10.0.19043 Build 19043
* Utilisation du pilote vmware basé sur le profil existant
* Démarrage du nœud de plan de contrôle minikube dans le cluster minikube
* Mise à jour du VM vmware en marche "minikube" ...
* Préparation de Kubernetes v1.22.2 sur Docker 20.10.8...
* Vérification des composants Kubernetes...
  - Utilisation de l'image gcr.io/k8s-minikube/storage-provisioner:v5
* Modules activés: storage-provisioner, default-storageclass
* Terminé ! kubectl est maintenant configuré pour utiliser "minikube" cluster et
espace de noms "default" par défaut.
```

Pour afficher plus d'informations sur le cluster, utilisez la commande :

```
C:\Users\Elyes>kubectl cluster-info
```

Résultat :

```
Kubernetes control plane is running at https://20.128.0.0:8443
CoreDNS is running at https://20.128.0.0:8443/api/v1/namespaces/kube-system/
services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

### Rappels

Le maître héberge le plan de contrôle Kubernetes : un ensemble de services qui administrent et orchestrent l'ensemble du cluster. Ces services se trouvent sous la forme de pods dans l'espace de nommage `kube-system`.

Le service DNS de Kubernetes permet aux pods de communiquer entre eux en utilisant leur nom ou leur FQDN au lieu de leur IP locale.

Kubernetes planifie un pod et un service DNS sur le cluster et configure les Kubelets pour indiquer à chaque conteneur d'utiliser l'adresse IP du service DNS pour résoudre les noms DNS.

Pour afficher des informations sur la configuration du cluster, utilisez la commande :

```
C:\Users\Elyes>kubect1 config view
```

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority: C:\Users\Elyes\.minikube\ca.crt
  extensions:
  - extension:
    last-update: Fri, 08 Oct 2021 18:48:52 CET
    provider: minikube.sigs.k8s.io
    version: v1.23.2
    name: cluster_info
  server: https://20.128.0.0:8443
  name: minikube
contexts:
- context:
  cluster: minikube
  extensions:
  - extension:
    last-update: Fri, 08 Oct 2021 18:48:52 CET
    provider: minikube.sigs.k8s.io
    version: v1.23.2
    name: context_info
  namespace: default
  user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: C:\Users\Elyes\.minikube\profiles\minikube\client.crt
    client-key: C:\Users\Elyes\.minikube\profiles\minikube\client.key
```

Pour afficher les nœuds appartenant au cluster, tapez la commande :

```
C:\Users\Elyes>kubect1 get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
minikube	Ready	control-plane,master	20h	v1.22.2

Affichez les pods en cours d'exécution dans l'espace de nommage par défaut :

```
C:\Users\Elyes>kubect1 get pods
No resources found in default namespace.
```

Affichez les pods en cours d'exécution dans l'espace de nommage kube-system :

```
C:\Users\Elyes>kubect1 get pods --namespace kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
coredns-78fcd69978-nq6wh	1/1	Running	1 (20h ago)	20h
etcd-minikube	1/1	Running	1 (20h ago)	20h
kube-apiserver-minikube	1/1	Running	1 (20h ago)	20h
kube-controller-manager-minikube	1/1	Running	1 (20h ago)	20h
kube-proxy-zfwvk	1/1	Running	1 (20h ago)	20h
kube-scheduler-minikube	1/1	Running	1 (20h ago)	20h
storage-provisioner	1/1	Running	3 (11m ago)	20h

Vous pouvez créer un objet dans Kubernetes à l'aide d'une méthode déclarative ou impérative.

- Lors de l'utilisation de commandes impératives, un utilisateur travaille directement sur des objets vivant dans un cluster. L'utilisateur fournit des opérations à la commande `kubect1` (`create`, `delete`, `replace`, etc.) sous forme d'arguments.
- Lors de l'utilisation de la configuration d'objet déclarative, un utilisateur opère sur les fichiers de configuration d'objet stockés localement, mais il ne définit pas les opérations à effectuer sur les fichiers. Les opérations de création, de mise à jour et de suppression sont automatiquement détectées.

## Créer un pod avec la méthode impérative

Créez un pod avec la méthode impérative :

```
kubect1 run premier-pod --image=nginx --restart=Never
```

```
pod/premier-pod created
```

Affichez les pods en cours d'exécution :

```
C:\Users\Elyes>kubect1 get pods
```

NAME	READY	STATUS	RESTARTS	AGE
premier-pod	1/1	Running	0	2m25s

Affichez les logs liés au premier pod :

```
kubectl logs premier-pod
```

```
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform
configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/
default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/
default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2021/10/08 18:00:15 [notice] 1#1: using the "epoll" event method
2021/10/08 18:00:15 [notice] 1#1: nginx/1.21.3
2021/10/08 18:00:15 [notice] 1#1: built by gcc 8.3.0 (Debian 8.3.0-6)
2021/10/08 18:00:15 [notice] 1#1: OS: Linux 4.19.202
2021/10/08 18:00:15 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2021/10/08 18:00:15 [notice] 1#1: start worker processes
```

Effectuez un mappage de port pour atteindre le pod à partir de la machine hôte :

```
kubectl port-forward pods/premier-pod 8080:80
```

```
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
Handling connection for 8080
Handling connection for 8080
```

Figure 4-6

① 127.0.0.1:8080

## Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](https://nginx.org).  
Commercial support is available at [nginx.com](https://nginx.com).

*Thank you for using nginx.*

Vous pouvez lancer des commandes à l'intérieur du pod :

```
kubectl exec -it premier-pod -- bash
```

```
root@premier-pod:/# hostname -i
172.17.0.3
root@premier-pod:/# service nginx status
[ ok ] nginx is running.
```

Pour supprimer le pod, utilisez la commande suivante :

```
kubectl delete pod premier-pod
```

```
pod "premier-pod" deleted
```

## Créer un pod avec la méthode déclarative

Vérifiez qu'aucun pod n'est en cours d'exécution :

```
C:\Users\Elyes>kubectl get pods
```

```
No resources found in default namespace.
```

Pour déclarer le pod, créez un fichier `nginxpod.yml` sur le Bureau contenant les lignes suivantes :

```
apiVersion: v1

kind: Pod

metadata:
  name: premier-pod
  labels:
    app: elyesapp
    type: frontend

spec:
  containers:
    - name: nginxpod
      image: nginx
      resources:
        requests:
          memory: "128Mi"
          cpu: "250m"
        limits:
          memory: "256Mi"
          cpu: "500m"
```

Créez le pod avec cette commande :

```
kubectl apply -f Desktop\nginxpod.yml
```

Vérifiez que le pod est en cours d'exécution :

```
C:\Users\Elyes>kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
premier-pod	1/1	Running	0	6s



## Kubernetes Dashboard

Le tableau de bord (*dashboard*) est une interface web pour Kubernetes. Vous pouvez l'utiliser pour déployer des applications conteneurisées dans un cluster Kubernetes, dépanner votre application conteneurisée et gérer les ressources du cluster.

Activez les plug-ins nécessaires :

```
C:\Users\Elyes>minikube addons enable dashboard
```

- Utilisation de l'image kubernetesui/dashboard:v2.3.1
- Utilisation de l'image kubernetesui/metrics-scraper:v1.0.7

\* Certaines fonctionnalités du tableau de bord nécessitent le module metrics-server. Pour activer toutes les fonctionnalités, veuillez exécuter :

```
minikube addons enable metrics-server
```

\* Le module 'dashboard' est activé

Lancez le dashboard avec cette commande :

```
C:\Users\Elyes>minikube dashboard
```

\* Vérification de l'état du tableau de bord...

\* Lancement du proxy...

\* Vérification de l'état du proxy...

\* Ouverture de <http://127.0.0.1:61293/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/> dans votre navigateur par défaut...

Une nouvelle fenêtre s'ouvre alors dans votre navigateur :

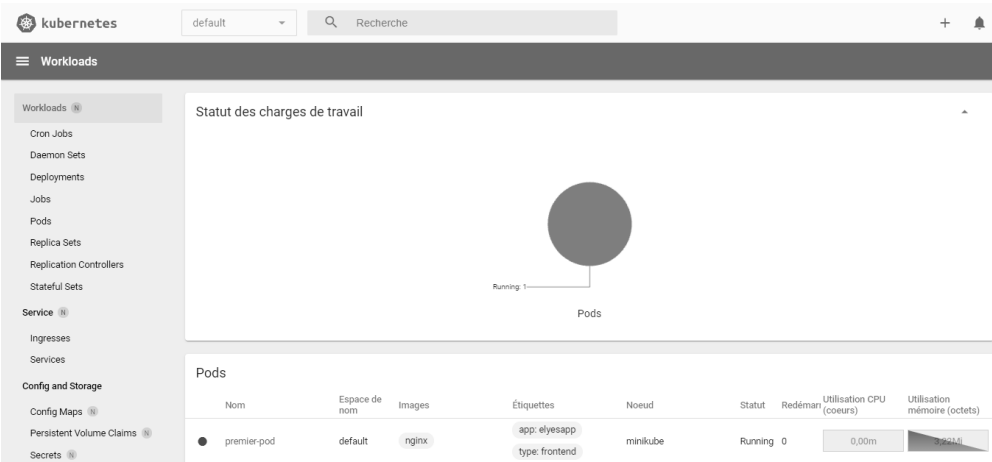


Figure 4–7 Dashboard de Kubernetes

Vous pouvez utiliser le tableau de bord pour obtenir une vue d'ensemble des applications en cours d'exécution dans votre cluster, créer ou modifier des ressources Kubernetes individuelles.

Par exemple, vous pouvez redimensionner un déploiement, lancer une mise à jour progressive, recréer un pod ou déployer de nouvelles applications à l'aide d'un assistant de déploiement.

Le tableau de bord fournit également des informations sur l'état des ressources Kubernetes de votre cluster et sur les erreurs éventuelles.

Supprimez le pod `premier-pod` en utilisant le dashboard (voir figure 4-8).

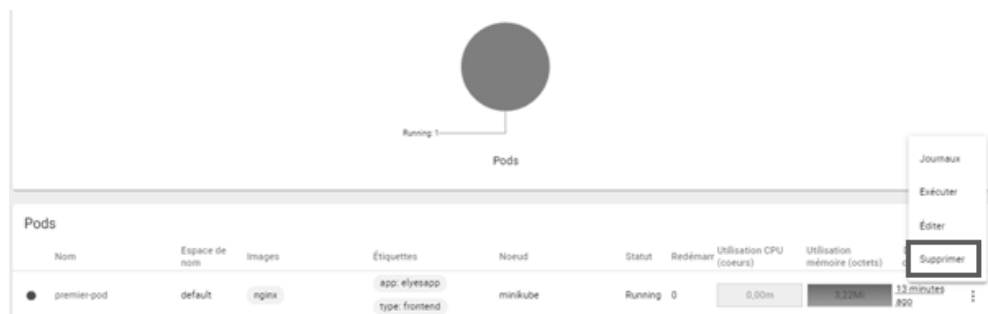
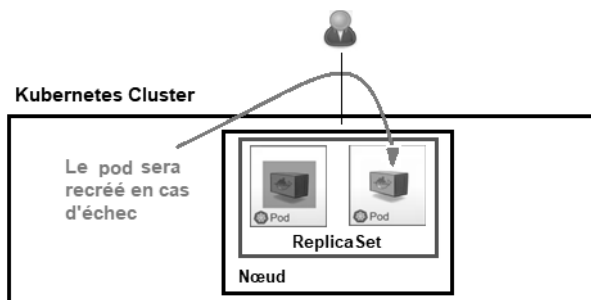


Figure 4-8

## ReplicaSets

Un ReplicaSet (« ensemble de répliques » en français) a pour but de maintenir stable un ensemble de pods à un moment donné. Cet objet est souvent utilisé pour garantir la disponibilité d'un certain nombre identique de pods.

Figure 4-9  
ReplicaSet

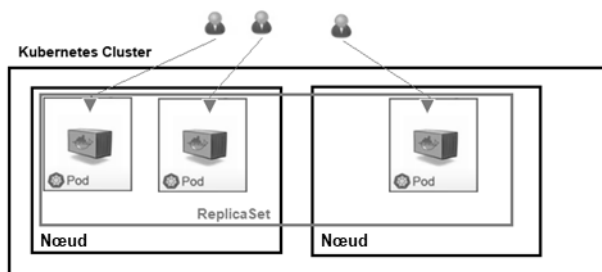


Un ReplicaSet est défini avec des champs, incluant un sélecteur qui spécifie comment identifier les pods qu'il va posséder, un nombre de répliques indiquant le nombre de pods qu'il doit maintenir et un modèle de pod spécifiant les données des nouveaux pods que le ReplicaSet va créer jusqu'au nombre de répliques demandé.

Il va atteindre son objectif en créant et supprimant des pods pour atteindre le nombre de répliques désirés.

**Figure 4-10**

Garantir la disponibilité d'un certain nombre identique de pods



Créez un fichier `nginxpod_replicaset.yml` sur le Bureau contenant les lignes suivantes :

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
spec:
  template:
    metadata:
      name: premiere-api
      labels:
        app: elyesapp
        type: frontend
    spec:
      containers:
        - name: nginxpod
          image: nginx
  replicas: 4
  selector:
    matchLabels:
      app: elyesapp
      type: frontend
```

Créez le ReplicaSet avec cette commande :

```
kubectl apply -f Desktop\nginxpod_replicaset.yml
```

```
replicaset.apps/myapp-replicaset created
```

Affichez les ReplicaSets :

```
C:\Users\Elyes>kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
myapp-replicaset	4	4	4	17s

```
C:\Users\Elyes>kubect1 get pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-replicaset-crvhz	1/1	Running	0	6m39s
myapp-replicaset-grr7l	1/1	Running	0	6m39s
myapp-replicaset-p288p	1/1	Running	0	3m7s
myapp-replicaset-xd5sz	1/1	Running	0	29s

Vérifiez que 4 pods sont bien en cours d'exécution et affichez plus d'informations sur le ReplicaSet :

```
C:\Users\Elyes>kubect1 describe rs myapp-replicaset
```

```
Name:          myapp-replicaset
Namespace:     default
Selector:      app=elyesapp,type=frontend
Labels:        <none>
Annotations:   <none>
Replicas:      4 current / 4 desired
Pods Status:   4 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=elyesapp
           type=frontend
  Containers:
    nginxpod:
      Image:          nginx
      Port:           <none>
      Host Port:      <none>
      Environment:    <none>
      Mounts:         <none>
      Volumes:        <none>
Events:
  Type     Reason             Age    From                      Message
  ----     -
  Normal   SuccessfulCreate   7m54s  replicaset-controller     Created pod: myapp-replicaset-grr7l
  Normal   SuccessfulCreate   7m54s  replicaset-controller     Created pod: myapp-replicaset-crvhz
  Normal   SuccessfulCreate   7m54s  replicaset-controller     Created pod: myapp-replicaset-mnkvh
  Normal   SuccessfulCreate   4m22s  replicaset-controller     Created pod: myapp-replicaset-p288p
  Normal   SuccessfulCreate   104s   replicaset-controller     Created pod: myapp-replicaset-xd5sz
```

Essayez à présent de supprimer un pod :

```
C:\Users\Elyes>kubect1 delete pod myapp-replicaset-crvhz
```

```
pod "myapp-replicaset-crvhz" deleted
```

Vérifiez que le pod a été remplacé par un autre pour avoir toujours 4 pods en cours d'exécution :

```
C:\Users\Elyes>kubect1 get pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-replicaset-grr7l	1/1	Running	0	8m49s
myapp-replicaset-p288p	1/1	Running	0	5m17s
myapp-replicaset-wswjj	1/1	Running	0	4s
myapp-replicaset-xd5sz	1/1	Running	0	2m39s

Vous pouvez mettre à l'échelle votre ReplicaSet avec cette commande :

```
kubect1 scale --replicas=2 replicaset myapp-replicaset
```

```
replicaset.apps/myapp-replicaset scaled
```

```
C:\Users\Elyes>kubect1 get pods
```

NAME	READY	STATUS	RESTARTS	AGE
myapp-replicaset-grr7l	1/1	Running	0	10m
myapp-replicaset-p288p	1/1	Running	0	7m9s

Supprimez le ReplicaSet avant de continuer :

```
kubect1 delete rs myapp-replicaset
```

```
replicaset.apps "myapp-replicaset" deleted
```

Vérifiez que tous les pods ont été supprimés :

```
C:\Users\Elyes>kubect1 get pods
```

```
No resources found in default namespace.
```

Un ReplicaSet garantit qu'un nombre spécifié de réplicas de pod est exécuté à un moment donné. Un déploiement est un concept de plus haut niveau qui gère les ReplicaSets et fournit des mises à jour déclaratives aux pods ainsi que de nombreuses autres fonctionnalités utiles.

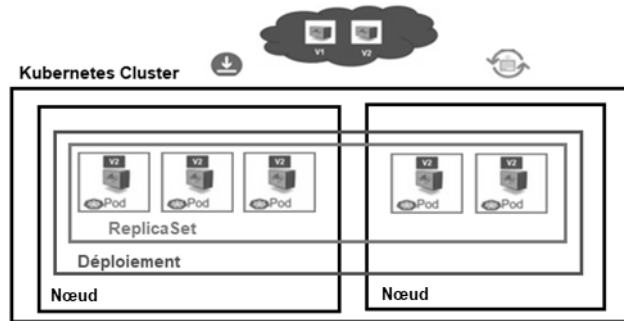
Par conséquent, il est recommandé d'utiliser des déploiements au lieu d'utiliser directement des ReplicaSets, sauf si vous avez besoin d'une orchestration personnalisée des mises à jour ou si vous n'avez pas besoin de mises à jour.

## Déploiements

Un déploiement Kubernetes est un objet qui fournit des mises à jour déclaratives pour des applications.

Il permet de décrire le cycle de vie d'une application, en spécifiant par exemple les images à utiliser, le nombre de pods à exécuter et la façon dont ils doivent être mis à jour.

**Figure 4-11**  
Déploiement



Créez votre premier déploiement en utilisant le dashboard de Kubernetes. L'exemple suivant permet de créer un déploiement de 4 réplicas à partir de l'image `httpd` :

```
apiVersion: apps/v1

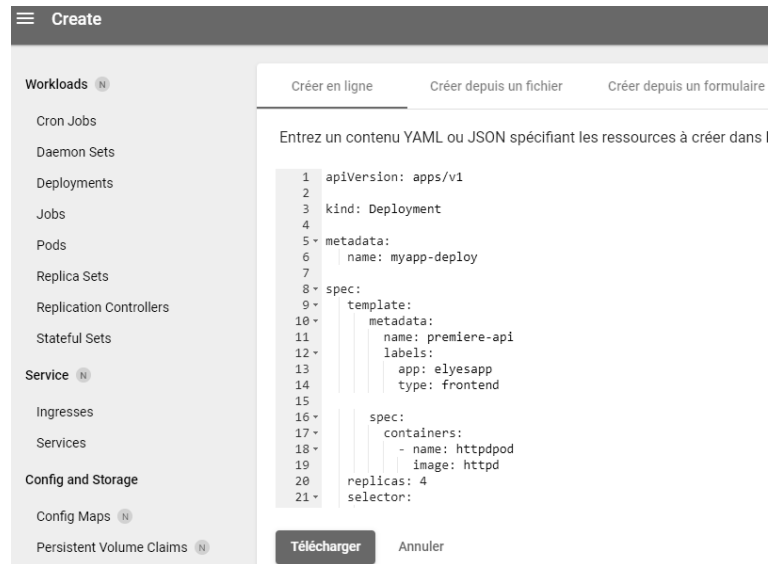
kind: Deployment

metadata:
  name: myapp-deploy

spec:
  template:
    metadata:
      name: premiere-api
      labels:
        app: elyesapp
        type: frontend

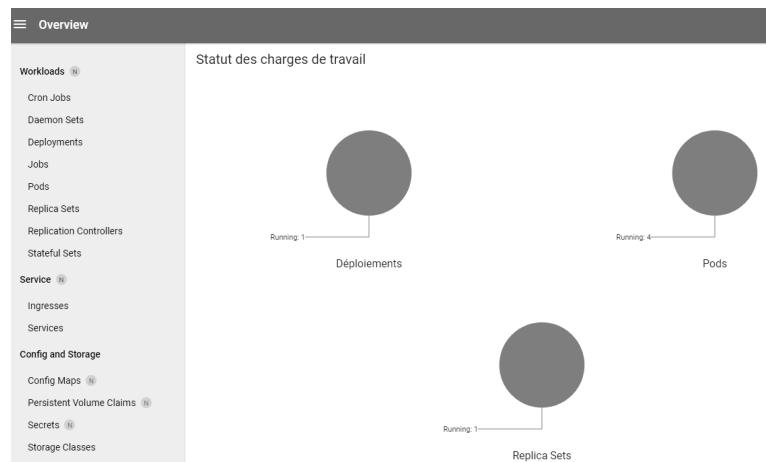
    spec:
      containers:
        - name: httpdpod
          image: httpd
  replicas: 4
  selector:
    matchLabels:
      app: elyesapp
      type: frontend
```

**Figure 4–12**  
Créer un déploiement en utilisant  
le dashboard de Kubernetes



Vous voyez ensuite l'état de votre déploiement :

**Figure 4–13**



Vous pouvez ajouter un mappage de port pour accéder à votre déploiement :

```
kubectl port-forward deployment/myapp-deploy 8080:80
```

```
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

Figure 4-14



## It works!

```
C:\Users\Elyes>kubect1 get deployments
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
myapp-deploy	4/4	4	4	3m49s

Pour afficher plus de détails, utilisez la commande suivante :

```
C:\Users\Elyes>kubect1 describe deployment myapp-deploy
```

```
Name: myapp-deploy
Namespace: default
CreationTimestamp: Fri, 08 Oct 2021 19:51:33 +0100
Labels: <none>
Annotations: deployment.kubernetes.io/revision: 1
Selector: app=elyesapp,type=frontend
Replicas: 4 desired | 4 updated | 4 total | 4 available | 0 unavailable
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels: app=elyesapp
         type=frontend
  Containers:
    httpdpod:
      Image: httpd
      Port: <none>
      Host Port: <none>
      Environment: <none>
      Mounts: <none>
      Volumes: <none>
Conditions:
  Type           Status  Reason
  ----           -
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet: myapp-deploy-685b45c844 (4/4 replicas created)
Events:
  Type    Reason             Age    From                      Message
  ----    -
  Normal  ScalingReplicaSet  4m32s  deployment-controller     Scaled up replica set
myapp-deploy-685b45c844 to 4
```



Et pour supprimer le déploiement, utilisez cette commande :

```
C:\Users\Elyes>kubectl delete deployment myapp-deploy
```

```
deployment.apps "myapp-deploy" deleted
```

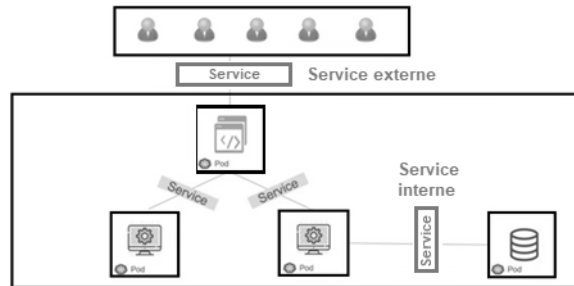
## Services

Un service est une manière abstraite d'exposer une application s'exécutant sur un ensemble de pods en tant que service réseau.

Avec Kubernetes, vous n'avez pas besoin de modifier votre application pour utiliser un mécanisme de découverte de services inconnus. Kubernetes attribue aux pods leurs propres adresses IP et un nom DNS unique pour un ensemble de pods, et peut équilibrer la charge entre eux.

**Figure 4-15**

Service

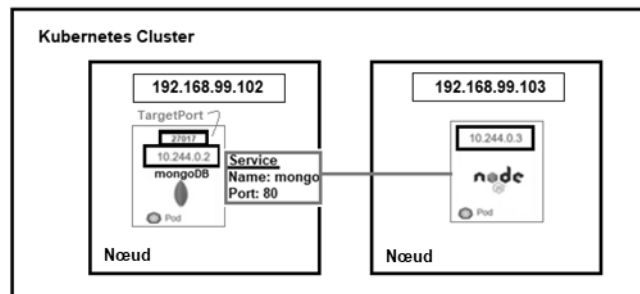


Il existe quatre types de services pour une utilisation particulière :

- ClusterIP : il s'agit du type par défaut. Il expose le service sur une adresse IP interne du cluster. De ce fait, le service n'est accessible que depuis l'intérieur du cluster.

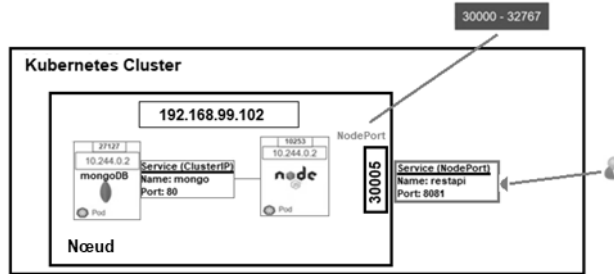
**Figure 4-16**

Service ClusterIP

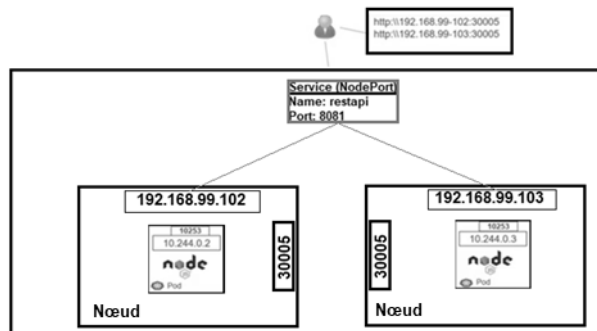


- NodePort : ce type expose le service vers l'extérieur du cluster à l'aide du NAT (la plage de ports autorisés est comprise entre 30000 et 32767).

**Figure 4–17**  
Service NodePort

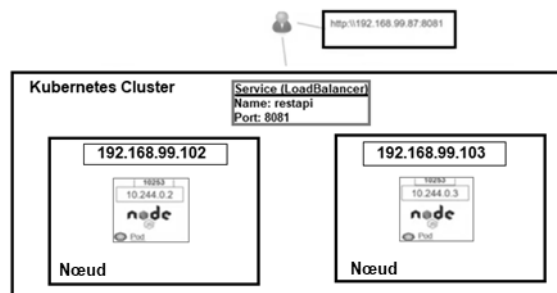


**Figure 4–18**  
Accès au service à l'aide de  
l'adresse du nœud



- **LoadBalancer** : ce type utilise l'équilibreur de charge des fournisseurs de cloud. Ainsi, les services NodePort et ClusterIP sont créés automatiquement et sont acheminés par l'équilibreur de charge externe.

**Figure 4–19**  
Service LoadBalancer



- **ExternalName** : ce service effectue une simple redirection CNAME (par exemple, rediriger le trafic vers le nom de domaine `e1yes.tn`).

# Application

Pour ce cas pratique, nous allons créer un service interne redis en nous basant sur l’image redis (voir figure 4-20).

Figure 4-20

Create

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Service

Ingresses

Services

Config and Storage

Config Maps

Persistent Volume Claims

Secrets

Storage Classes

Créer en ligne

Créer depuis un fichier

Créer depuis un formulaire

Nom de l'application \*

redis

5 / 24

Image du conteneur \*

redis

Nombre de pods \*

1

Service \*

Internal

Port \*

6379

Port cible \*

6379

Protocole \*

TCP

Port

Port cible

Protocole \*

TCP

Déployer

Annuler

Afficher les options avancées

Créez ensuite un service externe en utilisant l’image « elyesntc/nodejs-redis-app » comme suit :

Figure 4-21

Create

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Service

Ingresses

Services

Config and Storage

Config Maps

Persistent Volume Claims

Secrets

Storage Classes

Créer en ligne

Créer depuis un fichier

Créer depuis un formulaire

Nom de l'application \*

nodejs-redis-ap

15 / 24

Image du conteneur \*

elyesntc/nodejs-redis-app

Nombre de pods \*

1

Service \*

External

Port \*

80

Port cible \*

8080

Protocole \*

TCP

Port

Port cible

Protocole \*

TCP

Déployer

Annuler

Afficher les options avancées



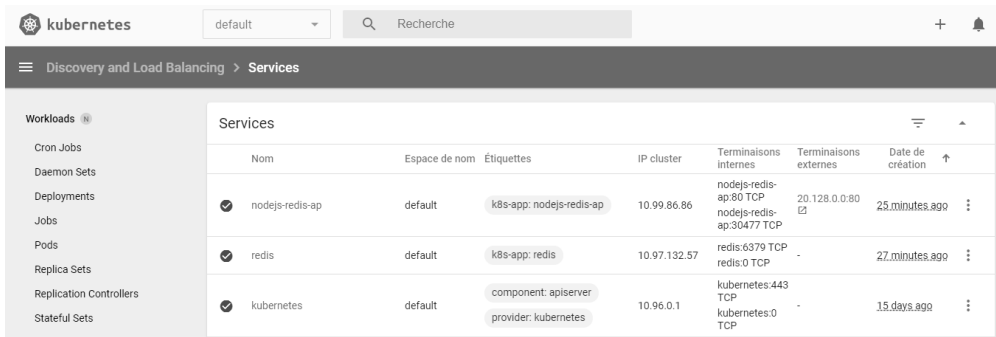
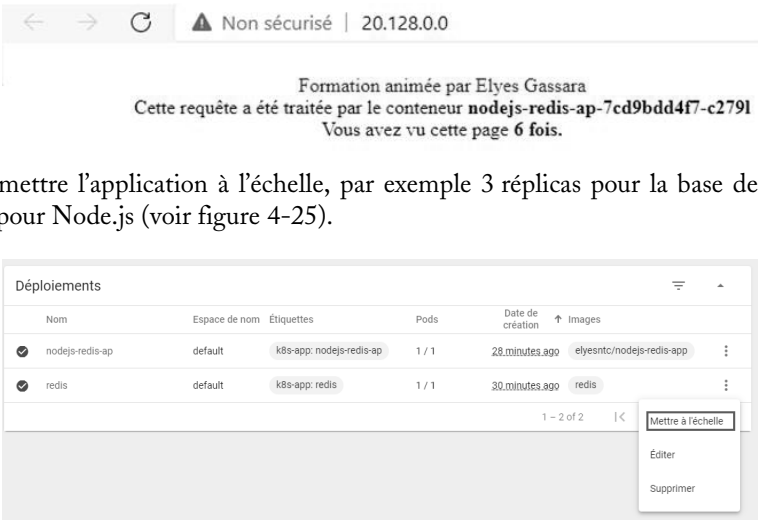


Figure 4-23 Vérifier l'état des services sur le dashboard

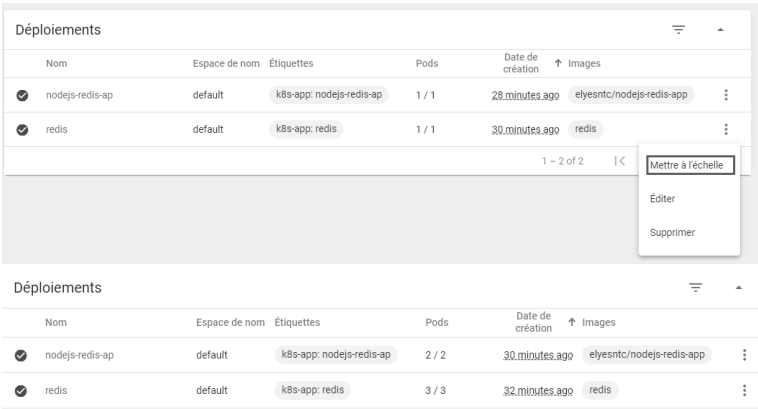
Accédez désormais à la page via cette adresse :

Figure 4-24



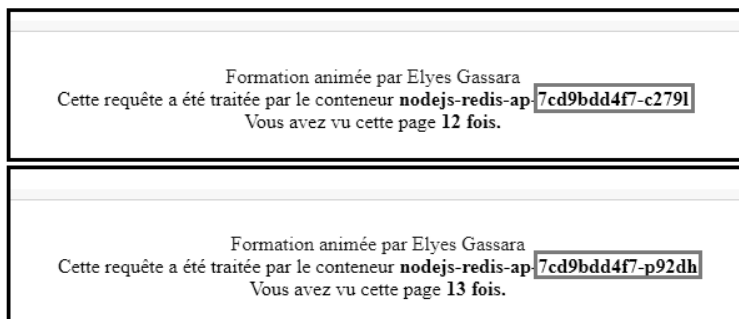
Vous allez à présent mettre l'application à l'échelle, par exemple 3 réplicas pour la base de données et 2 réplicas pour Node.js (voir figure 4-25).

Figure 4-25



Vérifiez que la répartition de charge fonctionne bien :

Figure 4-26



Affichez les services en utilisant cette commande :

```
C:\Users\Elyes>kubect1 get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	22h
nodejs-redis-app	LoadBalancer	10.99.86.86	20.128.0.0	80:30477/TCP	5m59s
redis	ClusterIP	10.97.132.57	<none>	6379/TCP	7m27s

Et saisissez cette commande pour obtenir plus de détails :

```
C:\Users\Elyes>kubect1 describe service nodejs-redis-app
```

```
Name: nodejs-redis-app
Namespace: default
Labels: k8s-app=nodejs-redis-app
Annotations: <none>
Selector: k8s-app=nodejs-redis-app
Type: LoadBalancer
IP: 10.99.86.86
External IPs: 20.128.0.0
Port: tcp-80-8080-11fx1 80/TCP
TargetPort: 8080/TCP
NodePort: tcp-80-8080-11fs1 30477/TCP
Endpoints: 172.17.0.6:8080,172.17.0.9:8080
Session Affinity: None
External Traffic Policy: Cluster
Events:
  Type    Reason      Age   From              Message
  ----    -
  Normal  ExternalIP  21m   service-controller Count: 0 -> 1
  Normal  ExternalIP  17m   service-controller Added: 20.128.0.0
```

Supprimez les services et les déploiements avant de continuer.

## Rolling Update

Nous allons ici voir comment mettre à jour un déploiement. Commençons par créer un service pour l’image « elyesntc/http-server:v1 » avec un nombre de réplicas égal à 10 (voir figure 4-27).

Figure 4–27

Créer en ligne

Créer depuis un fichier

Créer depuis un formulaire

Nom de l'application \*

http-server

11 / 24

Image du conteneur \*

elyesntc/http-server:v1

Nombre de pods \*

10

Service \*

External

Port \*

80

Port cible \*

8080

Protocole \*

TCP

Port

Port cible

Protocole \*

TCP

Déployer

Annuler

Afficher les options avancées

Test du serveur :

Figure 4–28

20.128.0.0

Formation présentée par Elyes Gassara

Application web version 1

Conteneur : http-server-6db688f44-sp6bh

Migrez maintenant vers la version 2 de l’application en éditant le déploiement (voir figure 4-29).

Figure 4-29



Il est possible de suivre la progression de la mise à jour sur le dashboard de Kubernetes (voir figure 4-30).

Figure 4-30

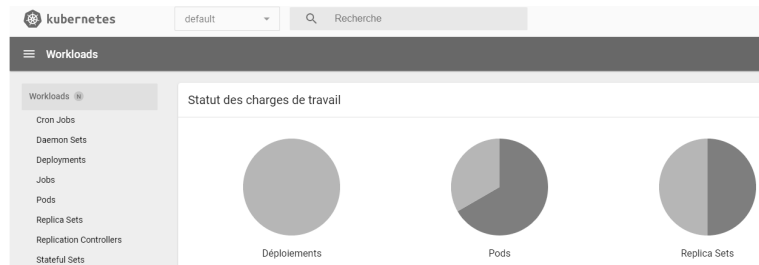


Figure 4-31

Formation présentée par Elyes Gassara  
 Application web version 2  
 Conteneur : http-server-7458977d87-grb69

Pour afficher les révisions et les configurations de déploiement précédentes, saisissez cette commande :

```
kubectl rollout history deployment/http-server
```

```

deployment.apps/http-server
REVISION CHANGE-CAUSE
1      <none>
2      <none>
  
```



Si vous voulez retourner vers la version 1, tapez :

```
kubectl rollout undo deployment/http-server --to-revision=1
```

```
deployment.apps/http-server rolled back
```

Testez alors votre rollback :

Figure 4-32

Formation présentée par Elyes Gassara  
Application web version 1  
Conteneur : **http-server-6db688f44-mzth9**

## Job

Un job crée un ou plusieurs pods et continuera à réessayer l'exécution des pods jusqu'à ce qu'un nombre spécifié d'entre eux se termine avec succès.

Lorsque le nombre spécifié d'achèvements réussis est atteint, le job ou la tâche (c'est-à-dire le travail) est terminé.

La suppression d'un job entraîne le nettoyage des pods qu'il a créés. La suspension d'une tâche supprime les pods actifs jusqu'à la reprise de la tâche.

## CronJob

Un CronJob crée des jobs selon un calendrier répétitif. Un objet CronJob est comme une ligne d'un fichier `crontab` (table `cron`). Il exécute un travail périodiquement selon un calendrier donné, écrit au format Cron.

## DaemonSet

Kubernetes s'assure que votre application dispose de ressources suffisantes, qu'elle s'exécute de manière fiable et qu'elle maintient une haute disponibilité tout au long de son cycle de vie. L'emplacement de l'application dans le cluster n'est pas une priorité.

Un DaemonSet vous permet de surmonter les limitations de planification de Kubernetes et de vous assurer qu'une application spécifique est déployée sur tous les nœuds du cluster.

Vous pouvez déclarer l'état souhaité, indiquant qu'un pod spécifique doit être présent sur chaque nœud. Si un nœud observé n'a pas de pod correspondant, le contrôleur DaemonSet va en créer un automatiquement.

## ConfigMap

Les ConfigMaps vous permettent de découpler les artefacts de configuration du contenu de l'image. Cela permet à Kubernetes de rendre votre application conteneurisée portable sans

que vous ayez à vous soucier des configurations. Les utilisateurs et les composants système peuvent stocker les données de configuration dans ConfigMap.

ConfigMap est quelque peu similaire aux secrets, mais il permet de travailler avec des chaînes qui ne contiennent pas d'informations sensibles.

## Secrets

Les secrets Kubernetes vous permettent de stocker et de gérer des informations sensibles, telles que les mots de passe, les jetons OAuth et les clés ssh. Stocker des informations confidentielles dans un secret est plus sûr et plus flexible que de les placer textuellement dans une définition de pod ou dans une image de conteneur (The Linux Foundation, 2021).

## Conclusion

Dans ce chapitre, nous avons abordé tous les aspects majeurs de Kubernetes, rendant ainsi son utilisation très simple, intéressante et puissante.

### Validation des acquis

#### Questions :

- ❶ Pourquoi utiliser Kubernetes ?
- ❷ Quel est le lien entre Kubernetes et Docker ?
- ❸ Quelles sont les caractéristiques de Kubernetes ?
- ❹ Qu'est-ce que Minikube ?
- ❺ À quoi sert Kubectl ?
- ❻ Quel est le rôle du kube-scheduler ?
- ❼ Pourquoi utiliser les espaces de noms dans Kubernetes ?
- ❽ Quels sont les différents services disponibles dans Kubernetes ?
- ❾ Pourquoi un équilibreur de charge est-il nécessaire ?
- ❿ Quels sont les outils utilisés pour la surveillance des conteneurs ?
- ⓫ Qu'est-ce que l'orchestration de conteneurs ?

**Réponses :****① Pourquoi utiliser Kubernetes ?**

- Kubernetes peut fonctionner sur des serveurs physiques sur site et sur les clouds publics Google, Azure, AWS, etc.
- Il permet aux applications d'être lancées et mises à jour sans aucun temps d'arrêt.
- Il vous permet de vous assurer que les applications conteneurisées s'exécutent où et quand vous le souhaitez, et vous aide à trouver les ressources et les outils que vous souhaitez utiliser.

**② Quel est le lien entre Kubernetes et Docker ?**

Kubernetes permet de lier et d'orchestrer plusieurs conteneurs, qui fonctionnent sur plusieurs hôtes créés à l'aide de Docker.

**③ Quelles sont les caractéristiques de Kubernetes ?**

- planification, mises à jour et retours en arrière automatisés ;
- mise à l'échelle horizontale et équilibrage de charge ;
- cohérence de l'environnement pour le développement, les tests et la production ;
- chaque composant peut agir comme une unité distincte ;
- plus grande densité d'utilisation des ressources ;
- fonctionnalités adaptées aux entreprises avec une gestion centrée sur les applications ;
- infrastructure auto-extensible.

**④ Qu'est-ce que Minikube ?**

Minikube est un logiciel qui aide l'utilisateur à exécuter Kubernetes. Il s'exécute sur un nœud unique qui se trouve dans une machine virtuelle sur votre ordinateur. Cet outil est également utilisé par les programmeurs qui développent une application à l'aide de Kubernetes.

**⑤ À quoi sert Kubectl ?**

Kubectl est un outil permettant de contrôler les clusters Kubernetes. En fait, « `ctl` » signifie « contrôle ». Il s'agit d'une interface en ligne de commande qui vous permet de transmettre des commandes au cluster et de gérer les composants Kubernetes.

**⑥ Quel est le rôle du kube-scheduler ?**

`kube-scheduler` est l'ordonnanceur par défaut de Kubernetes. Il affecte les nœuds aux pods nouvellement créés.

**⑦ Pourquoi utiliser les espaces de noms dans Kubernetes ?**

Les espaces de noms dans Kubernetes sont utilisés pour diviser les ressources du cluster entre les utilisateurs. Il aide l'environnement où plusieurs utilisateurs se répartissent les projets ou les équipes et fournit une étendue de ressources.

**⑧ Quels sont les différents services disponibles dans Kubernetes ?**

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName : il s'agit d'un cas spécial de service qui n'a pas de sélecteurs et utilise des noms DNS à la place.

**9 Pourquoi un équilibreur de charge est-il nécessaire ?**

Un équilibreur de charge est nécessaire parce qu'il offre un moyen standard de répartir le trafic réseau entre les différents services qui fonctionnent en arrière-plan.

**10 Quels sont les outils utilisés pour la surveillance des conteneurs ?**

- Heapster
- cAdvisor
- Prometheus
- InfluxDB
- Grafana

**11 Qu'est-ce que l'orchestration de conteneurs ?**

L'orchestration désigne l'intégration de plusieurs services ce qui leur permet d'automatiser des processus ou de synchroniser des informations en temps voulu.

Elle permet à tous les services de chaque conteneur de travailler de manière transparente pour atteindre un objectif unique.

L'orchestration de conteneurs est donc nécessaire pour permettre aux services de communiquer entre eux et de travailler ensemble pour répondre aux besoins du serveur.

# Kubernetes en production

---

Jusqu'à présent, nous avons utilisé Minikube pour déployer des applications sous un cluster Kubernetes. Comme nous l'avons vu, Minikube a été développé pour permettre aux utilisateurs d'exécuter Kubernetes localement et il s'adresse en premier lieu aux petits projets privés de développeurs, qui peuvent ainsi déployer leurs créations très simplement.

Aucun serveur ni cloud ne sont alors nécessaires car le cluster Kubernetes s'exécute uniquement sur l'hôte local. Cependant, en production, Kubernetes vous permet de tirer parti de vos infrastructures, qu'elles soient sur site (*on-premises*), hybrides ou en cloud public.

La majorité des organisations qui envisagent de déployer Kubernetes disposent déjà d'une infrastructure cloud ou sont en train de procéder à une migration vers le cloud.

Pour fonctionner, Kubernetes utilise plusieurs modules complémentaires qui fournissent des services pour divers domaines de fonctionnalités, notamment la mise en réseau, la sécurité, la surveillance et la journalisation. C'est pourquoi un outil de gestion de la configuration solide et polyvalent est nécessaire dans votre boîte à outils (Aly Saleh, 2021).

Les outils suivants constituent des choix judicieux :

- les outils de gestion de la configuration ordinaires, tels que Ansible, Chef et Puppet ;
- les outils spécifiques à Kubernetes, tels que Helm et Kustomize ;
- Terraform.

Dans ce chapitre, nous allons voir comment déployer notre application dans un environnement Kubernetes hébergé à l'aide du service Azure Kubernetes Service (AKS).

## Création d'un cluster Kubernetes sous AKS

Vous pouvez exécuter un cluster Kubernetes géré dans Microsoft Azure en utilisant AKS. Cela peut être une bonne option si vous voulez accéder à votre cluster à partir de plusieurs machines ou si vous avez un abonnement MSDN avec des crédits Azure.

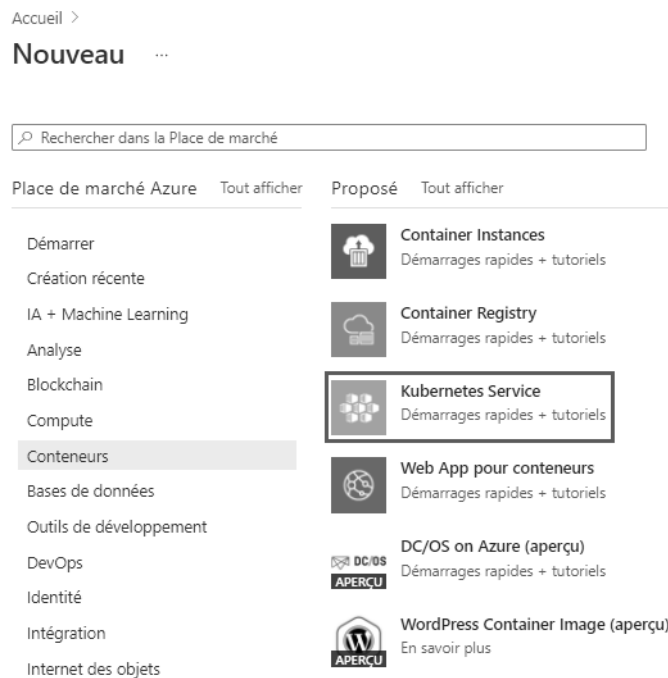
Vous pouvez exécuter un cluster minimal à un seul nœud, qui ne coûtera pas très cher, mais gardez à l'esprit qu'il n'y a aucun moyen d'arrêter le cluster et que vous paierez pour son utilisation 24h/24, 7 j/7, jusqu'à ce que vous le supprimiez (Stoneman, 2021).

AKS vous permet d'effectuer de nombreuses opérations de maintenance courantes sans mettre votre application hors connexion. Il permet aussi de déployer et de gérer des applications conteneurisées sans expertise d'orchestration de conteneurs. Ces opérations incluent le provisionnement bien sûr, la mise à niveau et la scalabilité des ressources à la demande.

Comme nous l'avons fait au chapitre 2, accédez au portail Azure via l'adresse <https://portal.azure.com/>. Cliquez sur *Créer une ressource* et choisissez la catégorie *Conteneurs*.

Vérifiez que vous avez la possibilité de créer une instance de conteneurs, un registre de conteneurs ainsi que le service Kubernetes (voir figure 5-1).

Figure 5-1



Pour la suite, nous allons choisir le service Kubernetes. Utilisez votre abonnement Azure pour créer un groupe de ressources afin d'organiser et gérer toutes les ressources liées au cluster nommé « elyes-cluster » et créé dans la région « France-Centre » (voir figure 5-2).

Vous pouvez répartir les nœuds de ce pool de nœuds sur plusieurs emplacements physiques, ce qui assure une haute disponibilité et protège donc les applications contre les défaillances du centre de données. Dans cette démonstration, nous n'utiliserons pas cette fonctionnalité.

Figure 5-2

Détails du projet

Sélectionnez un abonnement pour gérer les coûts et les ressources déployées. Utilisez les groupes de ressources comme des dossiers pour organiser et gérer toutes vos ressources.

Abonnement \* ⓘ Azure for Students

Groupe de ressources \* ⓘ (Nouveau) k8s-groupe  
Créer nouveau

Détails du cluster

Nom du cluster Kubernetes \* ⓘ elyes-cluster

Région \* ⓘ (Europe) France-Centre

Zones de disponibilité ⓘ Aucune

Version de Kubernetes \* ⓘ 1.18.14 (par défaut)

Pool de nœuds principal

Nombre et taille des nœuds dans le pool de nœuds principal de votre cluster. Pour les charges de travail de production, il est recommandé d'avoir au moins 3 nœuds à des fins de résilience. Pour les charges de travail de développement ou de test, un seul nœud est nécessaire. Si vous souhaitez ajouter des pools de nœuds supplémentaires ou afficher d'autres options de configuration pour ce pool de nœuds, accédez à l'onglet « Pools de nœuds » ci-dessus. Vous pourrez ajouter des pools de nœuds une fois votre cluster créé. En savoir plus sur les pools de nœuds dans Azure Kubernetes Service

Taille de nœud \* ⓘ Standard DS2 v2  
Changer la taille

Nombre de nœuds \* ⓘ 3

Vérifier + créer < Précédent Suivant : Pools de nœuds >

Sélectionnez la version de Kubernetes à utiliser pour ce cluster. Ensuite, il faut spécifier le nombre et la taille des nœuds dans le pool de nœuds principal de votre cluster.

Cliquez sur *Changer la taille* pour choisir la taille d'une machine virtuelle. Vous verrez plusieurs choix incluant le nombre de processeurs virtuels, la quantité de RAM et une estimation sur le coût par mois.

Vous devez choisir un modèle avec au moins deux processeurs virtuels et 4 Go de mémoire (voir figure 5-3).

Pour cet exemple, nous n'allons choisir qu'un seul nœud (voir figure 5-4). Cependant, en production, il est recommandé d'avoir au moins trois nœuds à des fins de résilience.

Vous pouvez également ajouter des pools de nœuds facultatifs pour gérer diverses charges de travail (voir figure 5-5).

L'activation de nœuds virtuels vous permet de déployer ou d'envoyer en rafale des conteneurs sur des nœuds soutenus par des instances de conteneurs Azure Serverless (Microsoft, 2021).

Figure 5-3

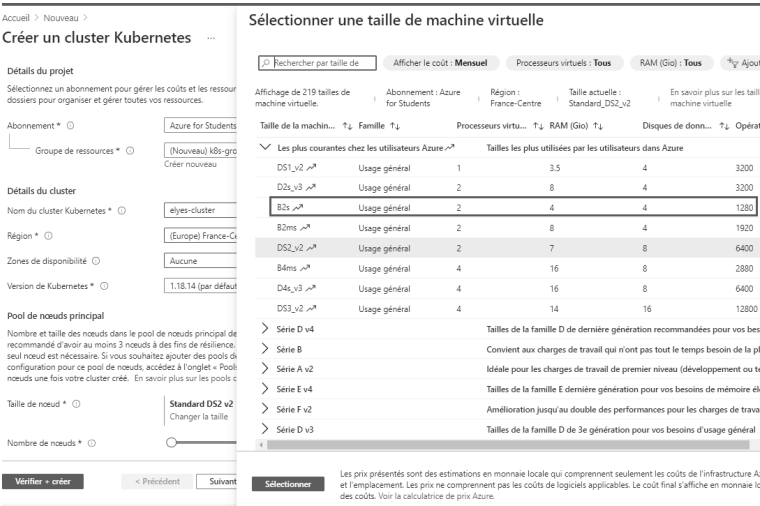


Figure 5-4

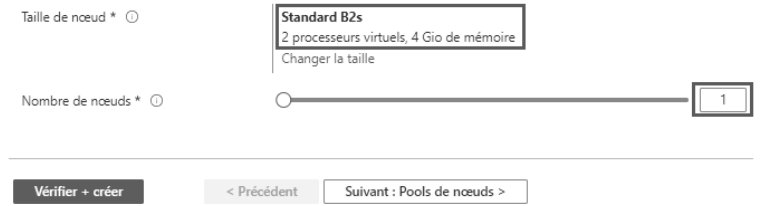
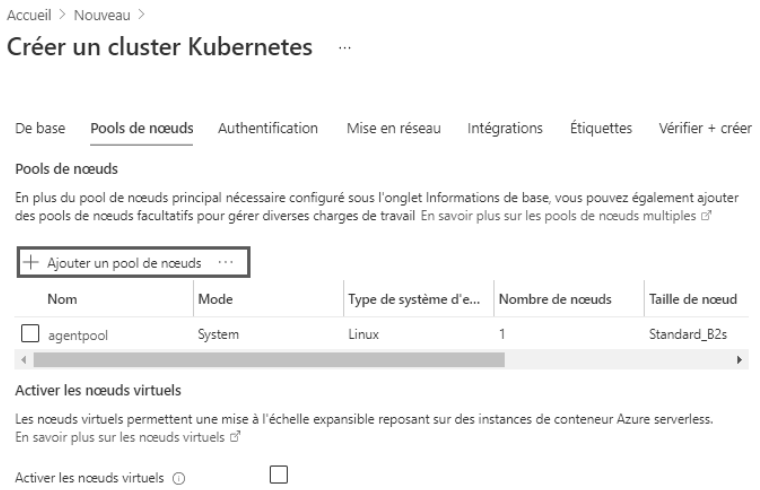


Figure 5-5





Vous pouvez activer un groupe de machines virtuelles identiques, ce qui est nécessaire dans des scénarios comme la mise à l'échelle automatique, les pools multinœuds et la prise en charge de Windows (voir figure 5-6).

**Figure 5-6**

#### Activer les groupes de machines virtuelles identiques

L'activation de groupes de machines virtuelles identiques crée un cluster qui utilise des groupes de machines virtuelles identiques au lieu de machines virtuelles individuelles pour les nœuds de cluster. Les groupes de machines virtuelles identiques sont nécessaires dans des scénarios comme la mise à l'échelle automatique, les pools multinœuds et la prise en charge de Windows. En savoir plus sur les groupes de machines virtuelles identiques dans AKS [☞](#)

Activer les groupes de machines  
virtuelles identiques ⓘ

[Vérifier + créer](#)[< Précédent](#)[Suivant : Authentification >](#)

Cliquez sur *Ajouter un pool de nœuds*. Vous pouvez ensuite choisir le mode *Système* ou *Utilisateur* pour ce pool de nœuds (voir figure 5-7) ainsi que le type de système d'exploitation des machines virtuelles (*Windows* ou *Linux*).

**Figure 5-7**

Microsoft Azure (Préversion)

Rechercher dans les ressources, services et documents (G+/I)

Accueil > Créer une ressource > Créer un cluster Kubernetes >

### Ajouter un pool de nœuds ...

Nom du pool de nœuds \* ⓘ

Mode \* ⓘ

☒ Utilisateur

☐ Système

Type de système d'exploitation \* ⓘ

☒ Linux

☐ Windows

Zones de disponibilité ⓘ

Aucune ▾

Taille de nœud \* ⓘ

Choisir une taille

Nombre de nœuds \* ⓘ

3

Nb max. de pods par nœud \* ⓘ

110 ▾

10 - 250

Ajouter

Annuler

Passons maintenant à l'étape d'authentification qui est utilisée par Azure Kubernetes Service pour gérer les ressources cloud attachées au cluster.

AKS nécessite une identité pour créer des ressources supplémentaires telles que des équilibreurs de charge et des disques managés dans Azure.

Vous pouvez choisir soit une identité gérée qui sera automatiquement créée pour vous par AKS, soit un principal du service qui doit être renouvelé pour que le cluster continue de fonctionner (voir figure 5-8).

Figure 5-8

Accueil > Nouveau >

## Créer un cluster Kubernetes ...

De base Pools de nœuds **Authentification** Mise en réseau Intégrations Étiquettes Vérifier + créer

**Infrastructure de cluster**  
L'authentification d'infrastructure de cluster spécifiée est utilisée par Azure Kubernetes Service pour gérer les ressources cloud attachées au cluster. Il peut s'agir d'un principal de service ☐ ou d'une identité managée affectée par le système ☑.

Méthode d'authentification ☐ Principal du service ☒ Identité managée affectée par le système

Activez le contrôle d'accès en fonction du rôle (RBAC) qui permet de gérer avec précision les ressources de cluster (voir figure 5-9).

Il s'agit d'une méthode de régulation de l'accès aux ordinateurs et aux ressources réseau basée sur les rôles des utilisateurs individuels au sein d'une entreprise.

Nous pouvons utiliser le contrôle d'accès basé sur les rôles sur toutes les ressources Kubernetes supportant les accès CRUD (*Create, Read, Update, Delete*).

Des autorisations sont combinées si plusieurs rôles sont attribués à un utilisateur. Par ailleurs, les autorisations peuvent être limitées à un seul espace de noms ou accordées à l'ensemble du cluster.

Figure 5-9

**Autorisation et authentification Kubernetes**  
L'authentification et l'autorisation sont utilisées par le cluster Kubernetes pour contrôler l'accès utilisateur au cluster ainsi que les actions possibles de l'utilisateur une fois qu'il est authentifié. En savoir plus sur l'authentification Kubernetes ☑

Contrôle d'accès en fonction du rôle (RBAC) ☒ Activé ☐ Désactivé

Azure Active Directory géré par AKS ☐ Activé ☒ Désactivé

**Chiffrement de disque de système d'exploitation du pool de nœuds**  
Par défaut, tous les disques sont chiffrés au repos dans AKS avec des clés gérées par Microsoft. Pour mieux contrôler le chiffrement, vous pouvez fournir vos propres clés à l'aide d'un jeu de chiffrement de disque issu d'Azure Key Vault. Le jeu de chiffrement de disque est utilisé pour chiffrer les disques de système d'exploitation de tous les pools de nœuds du cluster. En savoir plus ☑

Type de chiffrement (Par défaut) Chiffrement au repos avec une clé gérée par la plateforme ▼

Vérifier + créer < Précédent Suivant : Mise en réseau >

De même, il est possible de configurer le contrôle d'accès en fonction du rôle (RBAC) à l'aide de l'appartenance au groupe Azure Active Directory.

Le service d'identité d'entreprise Azure Active Directory (Azure AD) fournit l'authentification unique et l'authentification multifacteur pour vous aider à protéger vos utilisateurs contre 99,9 % des cyber-attaques (Microsoft, 2021).

Concernant les paramètres réseau du cluster, vous pouvez activer le routage des applications HTTP et configurer votre réseau à l'aide des options *Kubenet* ou *Azure CNI*.

Comme indiqué, « Le plug-in de réseau "kubenet" crée un réseau virtuel en se basant sur des valeurs par défaut » (voir figure 5-10). Si vous voulez personnaliser les adresses, choisissez plutôt le plug-in de réseau Azure CNI.

Figure 5-10

Accueil > Nouveau >

### Créer un cluster Kubernetes ...

De base Pools de nœuds Authentification **Mise en réseau** Intégrations Étiquettes Vérifier + créer

Vous pouvez changer les paramètres réseau de votre cluster, notamment en activant le routage des applications HTTP et en configurant votre réseau à l'aide des options « Kubenet » ou « Azure CNI » :

- Le plug-in de réseau « **kubenet** » crée un réseau virtuel pour votre cluster à l'aide des valeurs par défaut.
- Le plug-in de réseau « **Azure CNI** » permet aux clusters d'utiliser un réseau virtuel nouveau ou existant avec des adresses personnalisables. Les pods d'application sont connectés directement au réseau virtuel, ce qui permet une intégration native aux fonctionnalités de réseau virtuel.

En savoir plus sur les réseaux dans Azure Kubernetes Service

Configuration réseau ⓘ

☒ Kubenet  
☐ Azure CNI

Préfixe du nom DNS \* ⓘ

L'équilibreur de charge Azure Load Balancer route et équilibre le trafic vers votre cluster Kubernetes.

Une solution de routage des applications HTTP permet de faciliter l'accès aux applications déployées en créant des noms DNS accessibles à tous pour les points de terminaison d'application (voir figure 5-11).

Figure 5-11

### Routage du trafic

Équilibreur de charge ⓘ Standard

Activer le routage d'application HTTP ⓘ ☐

### Sécurité

Activer le cluster privé ⓘ ☐

Définir des plages d'adresses IP autorisées ⓘ ☐

Stratégie réseau ⓘ

☒ Aucun  
☐ Calico  
☐ Azure

❗ La stratégie réseau Azure n'est pas compatible avec le réseau kubenet.

Vérifier + créer < Précédent Suivant : Intégrations >

Pour des raisons de sécurité, vous pouvez configurer votre cluster comme un cluster privé. Ce dernier utilise alors une adresse IP interne pour garantir que le trafic réseau entre le serveur d'API et les pools de nœuds reste sur un réseau privé uniquement. Il est également possible de sécuriser l'accès au serveur d'API à l'aide de plages d'adresses IP autorisées.

Si vous voulez définir des règles pour le trafic entrant et sortant entre les pods d'un cluster, et donc améliorer la sécurité de votre cluster en limitant l'accès à certains pods, il est possible d'utiliser une stratégie réseau.

Dans l'onglet *Intégrations*, vous pouvez connecter votre cluster à un registre de conteneurs Azure (ACR, Azure Container Registry) pour activer des déploiements sans interruption à partir d'un registre d'images privé (voir figure 5-12). C'est un service de registre de conteneurs Docker géré et utilisé pour stocker des images de conteneurs Docker privées.

Figure 5-12

Accueil > Nouveau >

### Créer un cluster Kubernetes ...

De base Pools de nœuds Authentification Mise en réseau Intégrations Étiquettes Vérifier + créer

Connectez votre cluster AKS avec des services supplémentaires.

**Azure Container Registry**  
Connectez votre cluster à un registre de conteneurs Azure pour activer des déploiements sans interruption à partir d'un registre d'images privé. Vous pouvez créer un registre ou en choisir un que vous avez déjà.  
[En savoir plus sur Azure Container Registry](#)

Registre de conteneurs  Créer

Avec Azure Monitor, vous pouvez activer Container Insights pour obtenir des données plus complètes sur les performances générales et l'intégrité de votre cluster en plus des métriques de mémoire et de processeur incluses dans AKS par défaut.

Azure Policy permet d'appliquer des mises en œuvre et protections à grande échelle à vos clusters d'une manière centralisée et cohérente.

Figure 5-13

**Azure Monitor**  
En plus des métriques de mémoire et de processeur incluses dans AKS par défaut, vous pouvez activer Container Insights pour avoir des données plus complètes sur les performances générales et l'intégrité de votre cluster. La facturation est basée sur les paramètres d'ingestion des données et de conservation.  
[En savoir plus sur les performances et la supervision de l'intégrité des conteneurs](#)  
[En savoir plus sur les tarifs](#)

Supervision de conteneur ☒ Activé ☐ Désactivé

Espace de travail Log Analytics ☐  Créer

**Azure Policy**  
Appliquez à grande échelle des exécutions et des protections pour les clusters AKS de manière centralisée et cohérente dans Azure Policy. [En savoir plus sur Azure Policy pour AKS](#)

Azure Policy ☐ Activé ☒ Désactivé

Vérifier + créer < Précédent Suivant : Étiquettes >

Il ne reste qu'à ajouter des étiquettes qui sont des paires nom/valeur (voir figure 5-14). Elles permettent de catégoriser les ressources et de voir une facturation centralisée en appliquant la même étiquette à plusieurs ressources et groupes de ressources.

Figure 5-14

Accueil > Nouveau >

Créer un cluster Kubernetes ...

De base Pools de nœuds Authentification Mise en réseau Intégrations Étiquettes Vérifier + créer

Les étiquettes sont des paires nom/valeur qui vous permettent de catégoriser les ressources et de voir une facturation centralisée en appliquant la même étiquette à plusieurs ressources et groupes de ressources.  
En savoir plus sur les étiquettes ⓘ

Notez que si vous créez des étiquettes, puis que vous changez les paramètres de ressource sous d'autres onglets, vos étiquettes sont automatiquement mises à jour.

Nom ○	Valeur ○
Département	TI

L'étape de validation est importante. AKS vérifie les informations que vous avez saisies et valide ou non la création du cluster (voir figure 5-15).

La validation peut échouer si vous oubliez de renseigner des informations obligatoires ou si celles-ci ne sont pas valides.

Figure 5-15

Accueil > Nouveau >

Créer un cluster Kubernetes ...

✓ Validation réussie

De base Pools de nœuds Authentification Mise en réseau Intégrations Étiquettes Vérifier + créer

De base

Abonnement	Azure for Students
Groupe de ressources	(nouveau) k8s-groupe
Région	France-Centre
Nom du cluster Kubernetes	elyes-cluster
Version de Kubernetes	1.18.14

Pools de nœuds

Pools de nœuds	1
Activer les nœuds virtuels	Désactivé
Activer les groupes de machines virtuelles identiques	Activé

Authentification

Méthode d'authentification	Identité managée affectée par le système
Contrôle d'accès en fonction du rôle (RBAC)	Activé
Azure Active Directory géré par AKS	Désactivé
Type de chiffrement	(Par défaut) Chiffrement au repos avec une clé gérée par la plateforme

Créer

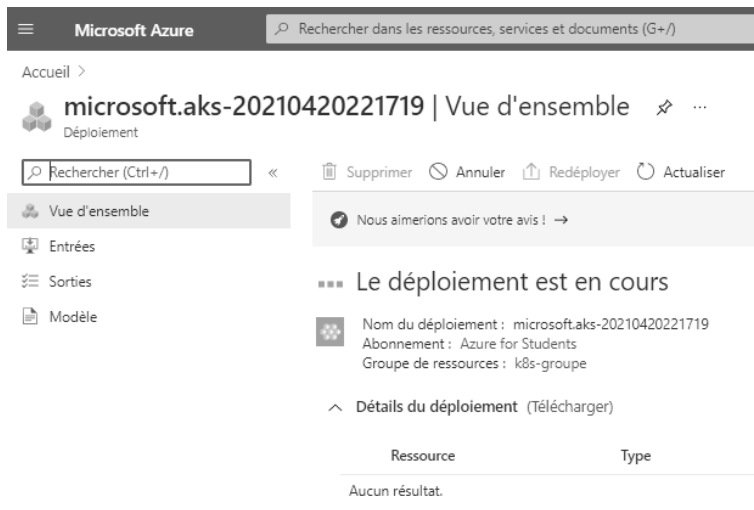
< Précédent

Suivant >

Télécharger un modèle pour automation

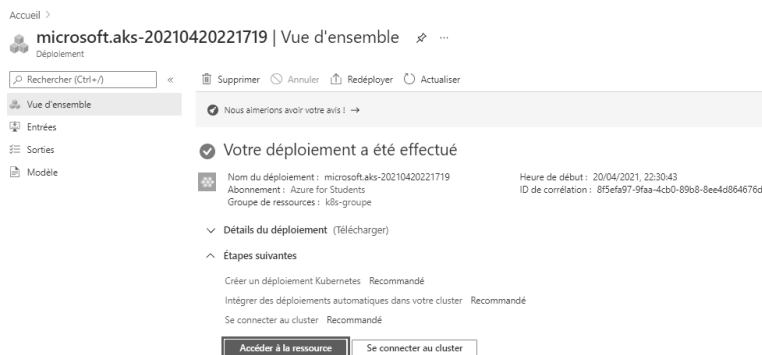
Si tout fonctionne correctement, une étape de déploiement prend quelques minutes : le temps du provisionnement des machines virtuelles et la configuration de votre cluster Kubernetes (voir figure 5-16).

Figure 5-16



À la fin de la configuration, un récapitulatif du déploiement apparaît (voir figure 5-17). Vous pouvez désormais accéder à cette nouvelle ressource.

Figure 5-17



Une vue d'ensemble présentant le statut, l'emplacement et les propriétés du cluster est affichée (voir figure 5-18).

Le menu de gauche permet d'afficher des informations, de naviguer dans les ressources Kubernetes et de modifier des paramètres.

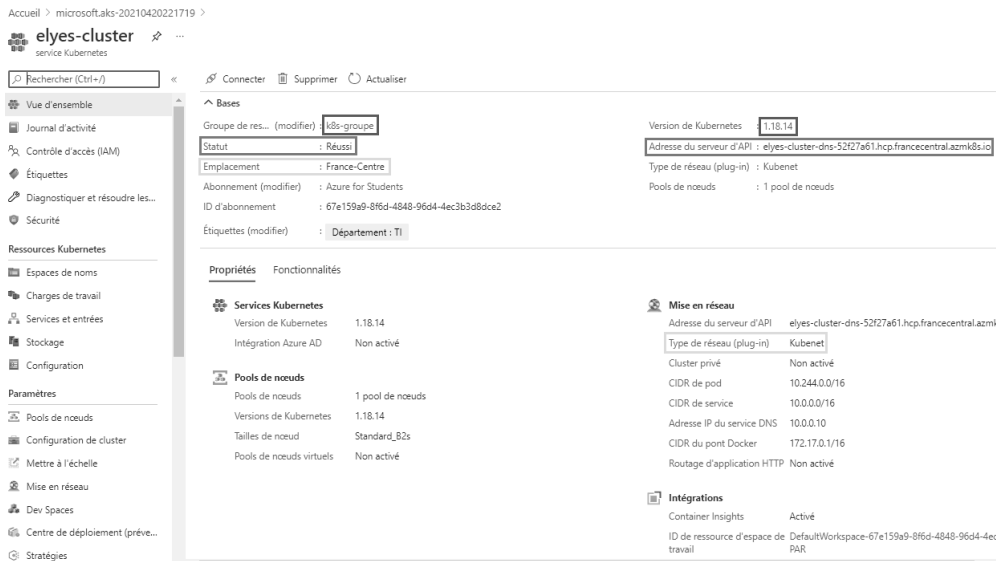
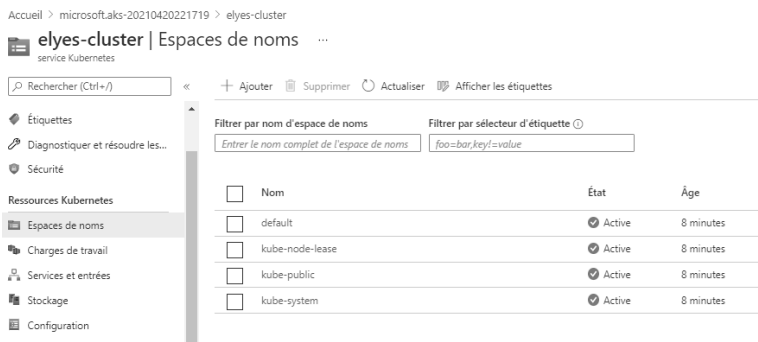


Figure 5–18

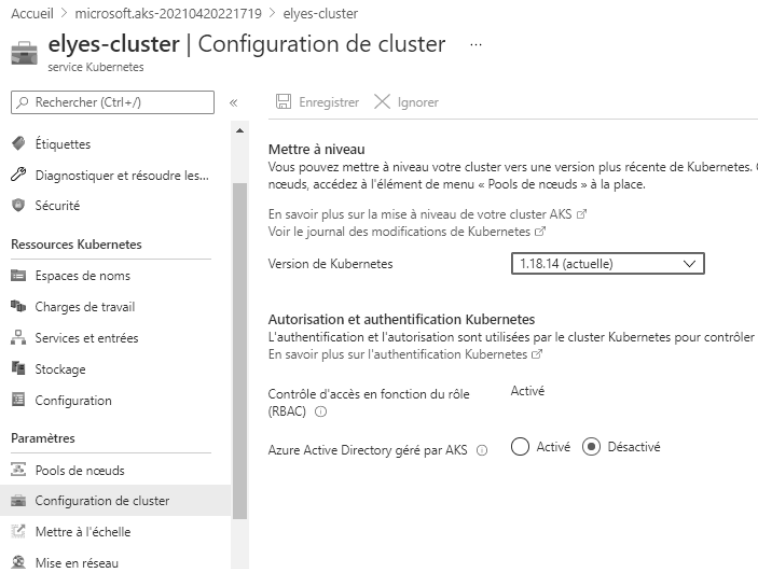
Voyons par exemple ce qu'il en est des espaces de noms :

Figure 5–19



Il est aussi possible de mettre à niveau le cluster Kubernetes. Pour cela, cliquez sur *Paramètres*>*Configuration de cluster* (voir figure 5-20).

Figure 5–20

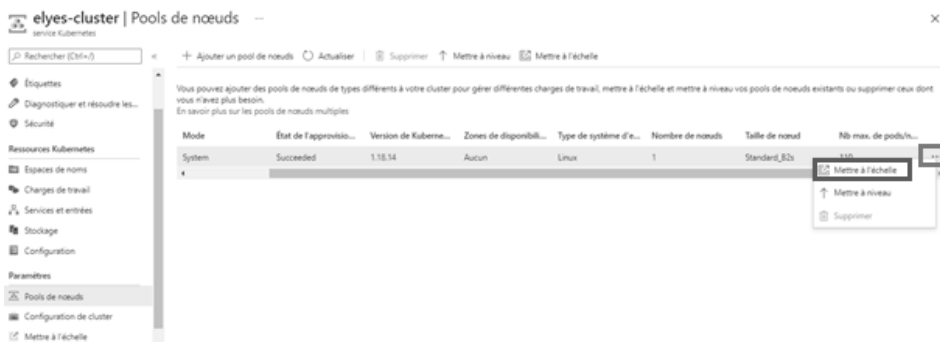


## Mise à l'échelle du cluster Kubernetes sous AKS

Kubernetes fournit une série de fonctionnalités pour garantir que vos clusters ont la bonne dimension pour pouvoir gérer n'importe quel type de charge.

Dans le menu *Paramètres* > *Pools de nœuds*, déroulez les options de votre pool de nœuds et choisissez *Mettre à l'échelle* comme indiqué sur la figure 5-21.

Figure 5–21



Kubernetes fournit plusieurs couches de fonctionnalités de mise à l'échelle automatique : mise à l'échelle basée sur le pod et mise à l'échelle basée sur des nœuds.

Il met automatiquement à l'échelle votre cluster dès que vous en avez besoin et redescend à sa taille nominale lorsque la charge est plus faible. Vous pouvez définir une plage pour le nombre de nœuds.



**Figure 5–22**  
Mettre à l'échelle un cluster  
Kubernetes

**Mettre à l'échelle** ✕

Méthode de mise à l'échelle ⓘ  
☐ Manuel ☒ Mise à l'échelle automatique

Plage du nombre de nœuds \* ⓘ  
 11  75

x Standard B2s (2 processeurs virtuels, 4 Gio de mémoire)

⚠ Le nombre maximal de nœuds que vous avez sélectionné est supérieur au quota restant pour cet abonnement. La mise à l'échelle automatique peut échouer au-dessus de 2 nœuds en raison de restrictions de quota (2 cœurs restants).

Nombre actuel de nœuds  
1

Nb max. de pods par nœud  
110

## Déploiement d'une application sous AKS

Il ne reste plus qu'à déployer l'application sur le cluster. Pour cela, nous pouvons soit installer Azure CLI (l'interface de ligne de commande Azure), soit utiliser Azure Cloud Shell.

Le Cloud Shell permet d'accéder à une ligne de commande basée sur un navigateur, conçue avec les tâches de gestion Azure.

Nous allons continuer sur ce cloud Shell directement sur le site [shell.azure.com](https://shell.azure.com) qui demande de créer un compte de stockage puisque Azure Cloud Shell nécessite qu'un partage de fichiers Azure conserve les fichiers. Cliquez donc sur le bouton *Créer un stockage*, comme indiqué (voir figure 5-23).

**Figure 5–23**

Vous n'avez aucun stockage monté

Azure Cloud Shell nécessite qu'un partage de fichiers Azure conserve les fichiers. En savoir plus  
 Ceci crée un compte de stockage pour vous, dont le coût mensuel est peu élevé. Voir les prix

\* Abonnement  
 Azure for Students [Afficher les paramètres avancés](#)

Utilisez les commandes `az account list -o table` et `az aks list -o table` pour obtenir les informations concernant votre compte Azure et votre cluster Kubernetes telles que le nom du groupe de ressources et le nom du cluster (voir figure 5-24).

Figure 5-24

```
Welcome to Azure Cloud Shell

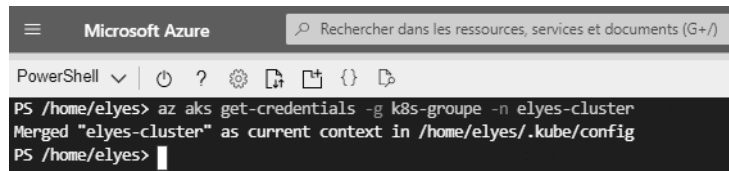
Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

NOTE: Scripts installed with 'Install-Script' can be run from the shell

VERBOSE: Authenticating to Azure ...
VERBOSE: Building your Azure drive ...
PS /home/elyes> az account list -o table
-----
Name                        CloudName      SubscriptionId      State      IsDefault
-----
Azure for Students          AzureCloud      67e159a9-8f6d-4848-96d4-4ec3b3d8dce2      Enabled    True
PS /home/elyes> az aks list -o table
-----
Name                        Location      ResourceGroup      KubernetesVersion      ProvisioningState      Fqdn
-----
elyes-cluster               francecentral k8s-groupe          1.18.14                 Succeeded              elyes-cluster-dns-52f27a61.hcp.francecentral.azurek8s.io
```

Ces informations seront ensuite utilisées pour récupérer les paramètres d'accès à notre cluster. La commande à utiliser est `az aks get-credentials -g k8s-groupe -n elyes-cluster`, comme indiqué sur la figure 5-25.

Figure 5-25



```
Microsoft Azure
Rechercher dans les ressources, services et documents (G+ /)

PowerShell
PS /home/elyes> az aks get-credentials -g k8s-groupe -n elyes-cluster
Merged "elyes-cluster" as current context in /home/elyes/.kube/config
PS /home/elyes>
```

Vous pouvez visualiser le contenu du fichier `config` à l'aide de la commande `more` et vérifier que le contexte du cluster indique que les informations de configuration de l'administrateur ont été appliquées (voir figure 5-26).

Figure 5-26

```
PS /home/elyes> az aks get-credentials -g k8s-groupe -n elyes-cluster
Merged "elyes-cluster" as current context in /home/elyes/.kube/config
PS /home/elyes> more /home/elyes/.kube/config
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUJzQ0FURS0tLS0tCk1JSUJ2RENDQXR0ZD0tLS0tL0t0ZDZlZCtmdqaX
  ME1qQXN1E14TmPsYUdBOH1NRV4TURReUE1ESXhNkVST1ZvdwEVEVMTUFR0EXVUBeE1DWFJFd2dnSm1NQTBHQ1NxR1NDYVJNEUUVQVFPVQU
  YkQ1NVh1YUdh1pkT11Ia2VzdjJ2YU80G4HydW9FbzZ1cE1VC3VQompFYXkQT3Zxa3VuQ09uRTFSVGJBL1ZHRXpqdmU5U1R1V3RU0FwSExSuk
  MS95S5k6wK25NClQ2RFVFE4DjXN2kwc3R3Ynlpd1pLZmlhaUEpLL2VWdDhMMkg1RVhXTG1uaTVLQ2EramhTU0xRWmJWlnEYNDhEa1oKcGV253JKeG
  a1VMWGg2ZF3JHMHZYTfjendK1MhY5S205RXBNUFR1MhxE51o5V65VUFRIUjpfTfB2OVZXBhMvdV1a29nOm5IZzjD0XbJb2yb2pYbVbG6BfdY2
  wEN1K0V4bk0YzXhBa11Zlmlh1VQCbD0tdm1xd09IR1dManNEV205Q1ZwaFFsTApsSUSNAhNMVU1Yak0V1ZZbFdTU0hM3F0DdxVhQK3SM1
  c0Vyb0x2Y2k2K0sxTHgwdTFmV2qY1ZNUJjZDNLdG8KU1tMv01pUnA35FMMV0tsRUZ5d1U1R25dmdRTDVB1e1ha1RXUUh1eU1SURBUUFcbz
  UWw0D10SDZ3bzA5XVYMGF4Cj9GNi9wVDJ03Njd08RMUpLb1p3aHZjTKFRRUxOUFEZ2dJQkFDK2C3K3Z34aXR1UGFY2R5R25R0
  WDRVQpRknpMQ11LSzR1R3FjY3F3cz1h2GwrbEdhK3h0VGZjcnRzT1UxS3V0bDhmb2Z1TmJ0SmowalUyQ1ENNQwXNpC120RkdPhk9DZERiR0
  QUHTEpZ8FVYbUlpM01MR4dSU81odGU0dJmN6SKc1FCTXRtcXpiMnJTL2NrZ1pkbTJzjnm03SQxdERs0Wq8ovSD0TRjhmVndjev1EBw5NG
  UEpMFEVz3VhT3E0BhpzWpYpSK9TMkL4lkF3ZmkzeXVY1AwYjFvU0kT1DqVjFUTG1nWFFbFNZ5k0Z0QozZjP5cmJ0ZG1MHjRUS1Vc4b1dW
  QUyzDURPjKx0e1d29PHtDYS2U5Y3Hge6TGKoraUHM1pMnFQDqyAhVWzZjVWpZj1S5E91Swt5m9K2w1d0RS10x0UdaafZfb3dEmN
  kS91QhK0Mx0bWRYcnp35MwQTFVMHF1Zk9wSFhtC10tLS0tRUS1ENFUIRJRk1DQVRFLS0tLS0k
  server: https://elyes-cluster-dns-52f27a61.hcp.francecentral.azurek8s.io:443
  name: elyes-cluster
contexts:
- context:
  cluster: elyes-cluster
  user: clusterUser_k8s-groupe_elyes-cluster
  name: elyes-cluster
current-context: elyes-cluster
kind: Config
preferences: {}
users:
- name: clusterUser_k8s-groupe_elyes-cluster
  user:
  client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUJzQ0FURS0tLS0tCk1JSUJ2RENDQXR0ZD0tLS0tL0t0ZDZlZCtmdqaX
  qQXN1E14TmPsYUdBOH1NRV4TURReUE1ESXhNkVST1ZvdwEVEVMTUFR0EXVUBeE1DWFJFd2dnSm1NQTBHQ1NxR1NDYVJNEUUVQVFPVQU
  YkQ1NVh1YUdh1pkT11Ia2VzdjJ2YU80G4HydW9FbzZ1cE1VC3VQompFYXkQT3Zxa3VuQ09uRTFSVGJBL1ZHRXpqdmU5U1R1V3RU0FwSExSuk
  MS95S5k6wK25NClQ2RFVFE4DjXN2kwc3R3Ynlpd1pLZmlhaUEpLL2VWdDhMMkg1RVhXTG1uaTVLQ2EramhTU0xRWmJWlnEYNDhEa1oKcGV253JKeG
  a1VMWGg2ZF3JHMHZYTfjendK1MhY5S205RXBNUFR1MhxE51o5V65VUFRIUjpfTfB2OVZXBhMvdV1a29nOm5IZzjD0XbJb2yb2pYbVbG6BfdY2
  wEN1K0V4bk0YzXhBa11Zlmlh1VQCbD0tdm1xd09IR1dManNEV205Q1ZwaFFsTApsSUSNAhNMVU1Yak0V1ZZbFdTU0hM3F0DdxVhQK3SM1
  c0Vyb0x2Y2k2K0sxTHgwdTFmV2qY1ZNUJjZDNLdG8KU1tMv01pUnA35FMMV0tsRUZ5d1U1R25dmdRTDVB1e1ha1RXUUh1eU1SURBUUFcbz
  UWw0D10SDZ3bzA5XVYMGF4Cj9GNi9wVDJ03Njd08RMUpLb1p3aHZjTKFRRUxOUFEZ2dJQkFDK2C3K3Z34aXR1UGFY2R5R25R0
  WDRVQpRknpMQ11LSzR1R3FjY3F3cz1h2GwrbEdhK3h0VGZjcnRzT1UxS3V0bDhmb2Z1TmJ0SmowalUyQ1ENNQwXNpC120RkdPhk9DZERiR0
  QUHTEpZ8FVYbUlpM01MR4dSU81odGU0dJmN6SKc1FCTXRtcXpiMnJTL2NrZ1pkbTJzjnm03SQxdERs0Wq8ovSD0TRjhmVndjev1EBw5NG
  UEpMFEVz3VhT3E0BhpzWpYpSK9TMkL4lkF3ZmkzeXVY1AwYjFvU0kT1DqVjFUTG1nWFFbFNZ5k0Z0QozZjP5cmJ0ZG1MHjRUS1Vc4b1dW
  QUyzDURPjKx0e1d29PHtDYS2U5Y3Hge6TGKoraUHM1pMnFQDqyAhVWzZjVWpZj1S5E91Swt5m9K2w1d0RS10x0UdaafZfb3dEmN
  kS91QhK0Mx0bWRYcnp35MwQTFVMHF1Zk9wSFhtC10tLS0tRUS1ENFUIRJRk1DQVRFLS0tLS0k
```

Le cluster est correctement configuré et en cours d'exécution. Vous pouvez y accéder en utilisant la commande `kubectl` comme nous l'avons fait précédemment avec Minikube.

Listez alors les nœuds et les pods disponibles, comme sur la figure 5-27.

Figure 5-27

```

Microsoft Azure
Rechercher dans les ressources, services et documents (G+/I)

PowerShell
PS /home/elyes> kubectl get nodes
NAME                                STATUS    ROLES    AGE    VERSION
aks-agentpool-37136749-vmss000000  Ready    agent    14m    v1.18.14

PS /home/elyes> kubectl get pods
No resources found in default namespace.

PS /home/elyes> kubectl get pods -A
NAMESPACE   NAME                                READY    STATUS    RESTARTS   AGE
kube-system  coredns-748cdb7bf4-dwhtt            1/1      Running    0           16m
kube-system  coredns-748cdb7bf4-g2vrf            1/1      Running    0           14m
kube-system  coredns-autoscaler-868b684fd4-cmrsv 1/1      Running    0           16m
kube-system  kube-proxy-5tfjd                    1/1      Running    0           14m
kube-system  metrics-server-58fdc875d5-lnhhs     1/1      Running    0           16m
kube-system  omsagent-69fr9                      1/1      Running    0           14m
kube-system  omsagent-rs-64bd7d87cd-lxzz4        1/1      Running    0           16m
kube-system  tunnelfront-66bbfc4c68-zgz82        1/1      Running    0           16m

```

Déployons à présent notre application Node.js qui stocke le nombre de visites dans une base redis. Le contenu du fichier décrivant le déploiement et le service redis est le suivant :

Figure 5-28

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  labels:
    application: node-redis
    component: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      application: node-redis
      component: redis
  template:
    metadata:
      labels:
        application: node-redis
        component: redis
    spec:
      containers:
        - image: redis
          name: redis
          ports:
            - containerPort: 6379
              name: http
              protocol: TCP

```

```

apiVersion: v1
kind: Service
metadata:
  name: redis
  labels:
    application: node-redis
    component: redis
spec:
  ports:
    - port: 6379
      protocol: TCP
      targetPort: 6379
  selector:
    application: node-redis
    component: redis

```

Et voici le fichier décrivant le déploiement et le service web :

Figure 5-29

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: webservice
  labels:
    application: node-redis
    component: webservice
spec:
  replicas: 2
  selector:
    matchLabels:
      application: node-redis
      component: webservice
  template:
    metadata:
      labels:
        application: node-redis
        component: webservice
    spec:
      containers:
        - name: webservice
          image: elyesntc/nodejs-redis-app
          ports:
            - name: webservice
              containerPort: 8080
              protocol: TCP
  ---
apiVersion: v1
kind: Service
metadata:
  name: webservice
  labels:
    application: node-redis
    component: webservice
spec:
  type: LoadBalancer
  selector:
    application: node-redis
    component: webservice
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
      nodePort: 30000
  
```

Nous allons charger ces deux fichiers à partir de la machine locale dans le répertoire de travail en les glissant directement sur l'interface du cloud Shell (voir figure 5-30).

Figure 5-30



Appliquons alors le déploiement des services avec la commande `kubectl apply` en indiquant à chaque fois le fichier correspondant (voir figure 5-31).

Figure 5-31

```

PowerShell
PS /home/elyes> kubectl apply -f redis.yaml
deployment.apps/redis created
service/redis created
PS /home/elyes> kubectl apply -f nodejs.yaml
deployment.apps/webservice created
service/webservice created
PS /home/elyes>
  
```

Vérifiez que tous les objets (pods, Deployment, ReplicaSet) ont été déployés correctement.

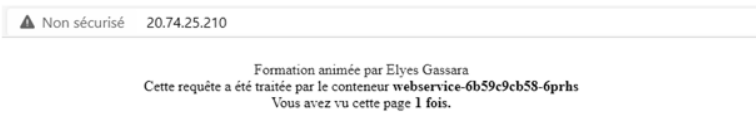
Notez l'adresse IP externe qui a été attribuée au service `webservice` de type `LoadBalancer` par le fournisseur de cloud.

Figure 5–32

```
PowerShell
PS /home/elyes> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
redis-5f6b884cf8-88cb2             1/1     Running   0           54s
webservice-6b59c9cb58-6prhs        1/1     Running   0           48s
webservice-6b59c9cb58-d5l86        1/1     Running   0           48s
PS /home/elyes> kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
redis     1/1     1            1           57s
webservice 2/2     2            2           51s
PS /home/elyes> kubectl get rs
NAME                                DESIRED   CURRENT   READY   AGE
redis-5f6b884cf8                    1         1         1       61s
webservice-6b59c9cb58                2         2         2       55s
PS /home/elyes> kubectl get svc
NAME      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes ClusterIP   10.0.0.1     <none>        443/TCP          20m
redis     ClusterIP   10.0.7.27    <none>        6379/TCP         68s
webservice LoadBalancer 10.0.174.16  20.74.25.210  80:30000/TCP     62s
```

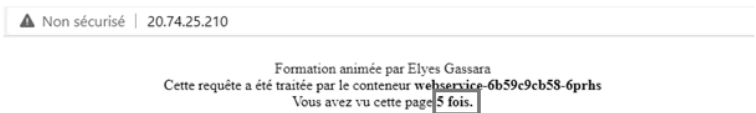
Utilisez alors cette adresse pour accéder au serveur web :

Figure 5–33



Actualisez la page et vérifiez la bonne connexion avec la base de données :

Figure 5–34



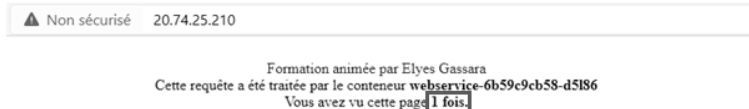
Nous allons maintenant supprimer le pod exécutant la base `redis` et vérifier qu’il est bien automatiquement relancé (voir figure 5-35).

Figure 5–35

```
PowerShell
PS /home/elyes> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
redis-5f6b884cf8-88cb2             1/1     Running   0           3m50s
web-service-6b59c9cb58-6prhs        1/1     Running   0           3m44s
web-service-6b59c9cb58-d5l86        1/1     Running   0           3m44s
PS /home/elyes> kubectl delete pod redis-5f6b884cf8-88cb2
pod "redis-5f6b884cf8-88cb2" deleted
PS /home/elyes> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
redis-5f6b884cf8-7qv49             1/1     Running   0           25s
web-service-6b59c9cb58-6prhs        1/1     Running   1           4m29s
web-service-6b59c9cb58-d5l86        1/1     Running   1           4m29s
```

Le pod a été relancé, mais le nombre de visites stocké dans l’ancienne base a été perdu.

Figure 5–36



Pour pallier ce problème, nous allons supprimer ces déploiements (voir figure 5-37) pour ajouter à la base un stockage persistant qui gardera les données même en cas de perte du conteneur.

Figure 5–37

```
PowerShell
PS /home/elyes> kubectl delete -f nodejs.yml
deployment.apps "webservice" deleted
service "webservice" deleted
PS /home/elyes> kubectl delete -f redis.yml
deployment.apps "redis" deleted
service "redis" deleted
```

Afin d'utiliser un stockage persistant dans le pod `redis`, nous devons créer un `PersistentVolumeClaim`, dans lequel nous spécifions la taille minimale du volume et son mode d'accès.

Kubernetes trouvera alors le `PersistentVolume` approprié et liera le volume à cet objet.

Les ressources `PersistentVolume` permettent de gérer l'espace de stockage durable dans un cluster et peuvent être provisionnées de manière dynamique via des ressources `PersistentVolumeClaims` ou être créées explicitement par un administrateur de cluster.

Figure 5–38

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
  labels:
    deployment: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      pod: redis
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
  template:
    metadata:
      labels:
        pod: redis
    spec:
      containers:
        - image: redis
          name: redis
          volumeMounts:
            - name: redis-persistent-storage
              mountPath: /data
          ports:
            - containerPort: 6379
              name: http
              protocol: TCP
      volumes:
        - name: redis-persistent-storage
          persistentVolumeClaim:
            claimName: redis-pv-claim
---

apiVersion: v1
kind: Service
metadata:
  name: redis
  labels:
    service: redis
spec:
  ports:
    - port: 6379
      protocol: TCP
      targetPort: 6379
  selector:
    pod: redis
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: redis-pv-claim
  labels:
    pod: redis
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Une fois qu'un volume a été lié avec succès, vous pouvez l'utiliser comme l'un des volumes à l'intérieur du pod.

Contrairement aux PersistentVolumes qui ne sont liés à aucun espace de noms, les PersistentVolumeClaims ne peuvent être créés que dans un espace de noms spécifique.

Appliquons alors ces changements en déployant à nouveau les fichiers. Notez la nouvelle adresse IP attribuée au serveur web.

Figure 5–39

```
PS /home/elyes> kubectl apply -f nodejs.yml
deployment.apps/webservice created
service/webservice created
PS /home/elyes> kubectl apply -f redis.yml
deployment.apps/redis created
service/redis created
persistentvolumeclaim/redis-pv-claim created
PS /home/elyes> kubectl get services
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP     10.0.0.1      <none>         443/TCP          27m
redis         ClusterIP     10.0.24.10    <none>         6379/TCP          17s
webservice    LoadBalancer 10.0.229.247  51.138.222.165 80:30000/TCP     22s
```

Figure 5–40



Incrémentez le nombre de visites pour pouvoir vérifier la persistance des données après le relancement du pod redis que nous allons supprimer.

Figure 5–41



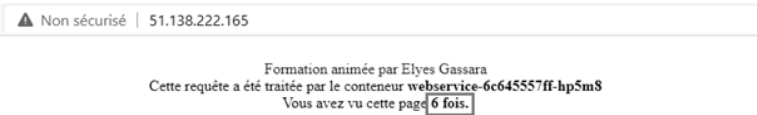
Supprimez alors le pod redis et notez le nombre de redémarrage des pods webservice. Cela est dû à l'erreur de connexion vers la base à la suite de sa suppression.

Figure 5–42

```
PS /home/elyes> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
redis-596bbff745-hbfgm  1/1     Running   0           2m39s
webservice-6c645557ff-9mqgv  1/1     Running   3           2m44s
webservice-6c645557ff-hp5m8  1/1     Running   3           2m44s
PS /home/elyes> kubectl delete pod redis-596bbff745-hbfgm
pod "redis-596bbff745-hbfgm" deleted
PS /home/elyes> kubectl get pods
NAME          READY   STATUS             RESTARTS   AGE
redis-596bbff745-bbrbb  1/1     Running            0           21s
webservice-6c645557ff-9mqgv  0/1     CrashLoopBackOff   3           3m21s
webservice-6c645557ff-hp5m8  0/1     CrashLoopBackOff   3           3m21s
PS /home/elyes> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
redis-596bbff745-bbrbb  1/1     Running   0           104s
webservice-6c645557ff-9mqgv  1/1     Running   4           4m44s
webservice-6c645557ff-hp5m8  1/1     Running   4           4m44s
```

Maintenant, actualisez la page et vérifiez que cette fois le nombre de visites n’a pas été perdu (voir figure 5-43).

Figure 5-43



Au moment de la rédaction de ce livre, le module complémentaire de tableau de bord AKS (le dashboard) est défini pour la dépréciation.

Le tableau de bord Kubernetes est activé par défaut pour les clusters exécutant une version Kubernetes inférieure à la version 1.18 et le module complémentaire de tableau de bord sera désactivé par défaut pour tous les nouveaux clusters créés sur Kubernetes 1.18 ou sur une version ultérieure.

Utilisez plutôt la vue des ressources Kubernetes sur le portail Azure (voir figure 5-44). Vous pouvez y visualiser des informations sur le cluster comme le pourcentage d'utilisation de l'unité centrale ou de la mémoire des nœuds.

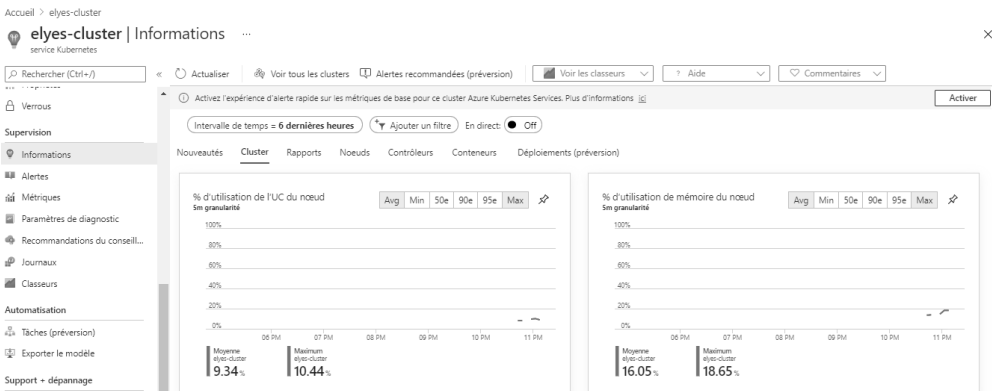


Figure 5-44

Le menu *Ressources Kubernetes>Stockage* affiche les informations relatives aux volumes persistants et aux classes de stockage (voir figure 5-45).

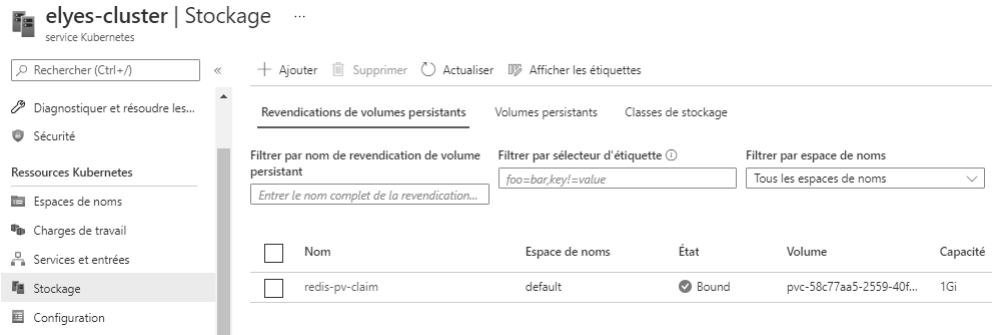


Figure 5-45



De même, il est possible d’afficher les charges de travail comme les déploiements, les pods et les jeux de réplicas (voir figure 5-46).

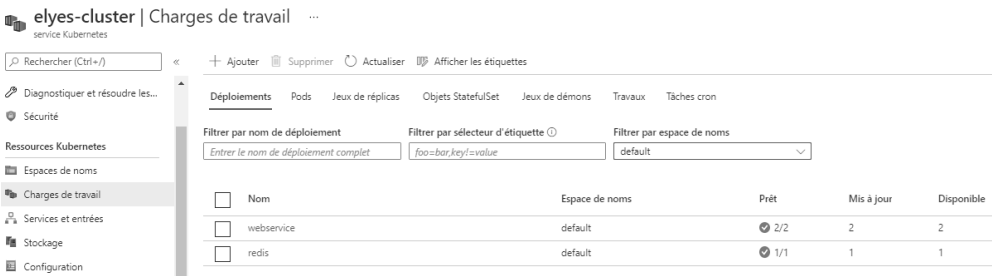


Figure 5-46

La figure 5-47 présente une vue d’ensemble sur le service `webservice`. Notez l’adresse IP externe et les ports utilisés.



Figure 5-47

Vous pouvez aussi afficher les pods liés à ce service (voir figure 5-48). Comme vous pouvez le constater, tout ce que vous pouvez faire via le dashboard de Kubernetes, vous pouvez le faire directement sur le portail Azure.

Affichez maintenant les informations concernant le service base de données (voir figure 5-49).

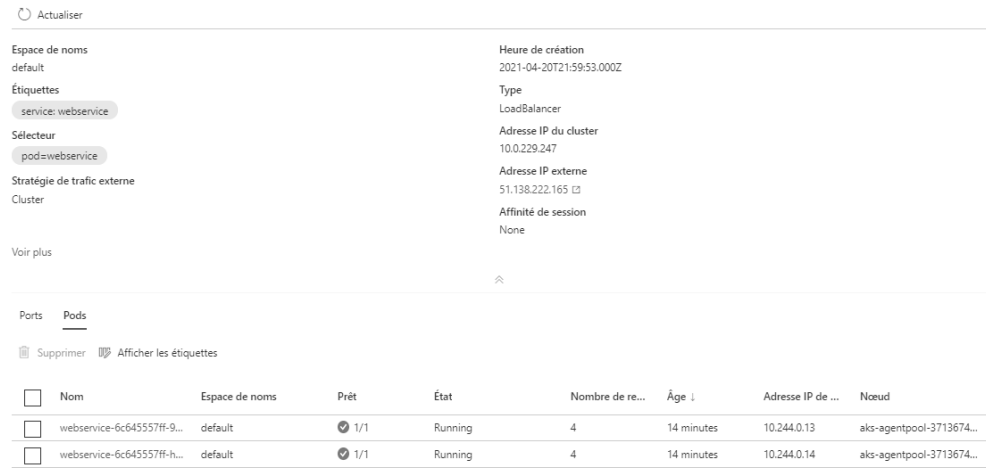


Figure 5-48

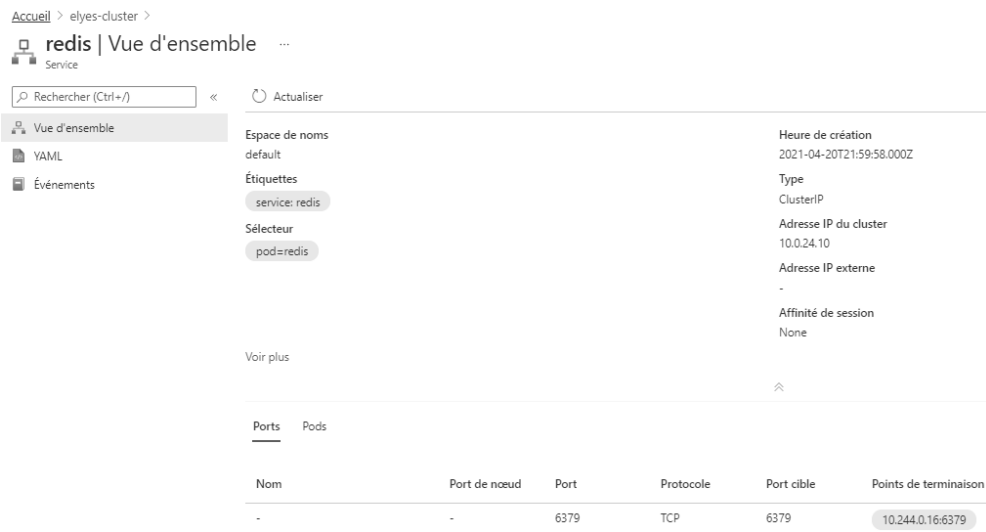


Figure 5-49

Comme vous le constatez sur la figure 5-50, vous pouvez aussi accéder directement à ces services via le menu *Ressources Kubernetes>Services et entrées*.

N'hésitez pas à explorer davantage le portail Azure, il regorge de fonctionnalités intéressantes.

Si vous voulez supprimer le cluster, utilisez cette commande en mentionnant bien sûr le nom du groupe de ressources et le nom du cluster :

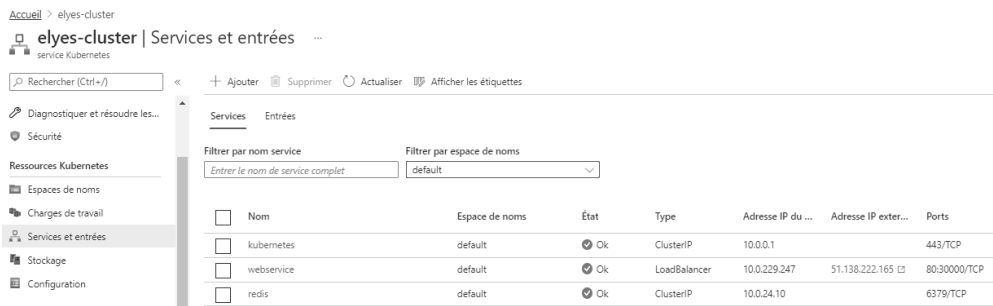
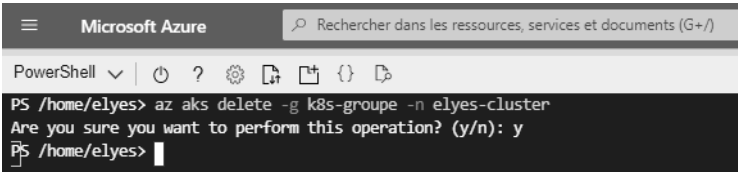


Figure 5-50

Figure 5-51



# Conclusion

Dans ce chapitre, nous avons abordé les ressources nécessaires pour déployer une plate-forme Kubernetes gérée sur Microsoft Azure. Grâce à son offre de Kubernetes en tant que Service, Microsoft a choisi de tout encapsuler et de proposer un portail qui permet d’afficher toutes les propriétés liées à un cluster AKS.

Le but est donc de simplifier le déploiement d’un cluster Kubernetes en déléguant la complexité et la surcharge opérationnelle de la gestion de ce cluster au cloud de Microsoft.

## Validation des acquis

### Questions :

- 1 Service Azure Kubernetes (AKS) : qu’est-ce que c’est et pourquoi l’utilisons-nous ?
- 2 Est-il possible de répartir un cluster AKS sur plusieurs régions ?
- 3 Est-il possible de répartir un cluster AKS sur plusieurs zones de disponibilité ?
- 4 Est-il possible d’avoir différentes tailles de machines virtuelles dans un seul cluster ?
- 5 Est-il possible d’exécuter des conteneurs Windows Server sur AKS ?
- 6 En quoi consiste le nœud virtuel Azure Kubernetes Service (AKS) ?

**Réponses :****① Service Azure Kubernetes (AKS) : qu'est-ce que c'est et pourquoi l'utilisons-nous ?**

AKS est un service d'orchestration de conteneurs open source entièrement géré, qui peut être utilisé pour déployer, mettre à l'échelle et gérer des conteneurs Docker et des applications basées sur des conteneurs dans un environnement de cluster. Il est devenu disponible en juin 2018 sur le cloud public Microsoft Azure.

Nous l'utilisons car il présente plusieurs avantages :

- utilisation efficace des ressources ;
- développement plus rapide des applications ;
- sécurité et conformité ;
- développement et intégration plus rapides.

**② Est-il possible de répartir un cluster AKS sur plusieurs régions ?**

Non. Les clusters AKS sont des ressources régionales et ne peuvent pas s'étendre sur plusieurs régions.

**③ Est-il possible de répartir un cluster AKS sur plusieurs zones de disponibilité ?**

Oui. Vous pouvez déployer un cluster AKS sur une ou plusieurs zones de disponibilité dans les régions qui les prennent en charge.

**④ Est-il possible d'avoir différentes tailles de machines virtuelles dans un seul cluster ?**

Oui. En créant plusieurs pools de nœuds, vous pouvez utiliser différentes tailles de machine virtuelle dans votre cluster AKS.

**⑤ Est-il possible d'exécuter des conteneurs Windows Server sur AKS ?**

Oui. Vous devez dans ce cas créer un pool de nœuds qui exécute Windows Server en tant que système d'exploitation invité.

**⑥ En quoi consiste le nœud virtuel Azure Kubernetes Service (AKS) ?**

Il permet d'approvisionner de façon élastique des pods supplémentaires à l'intérieur d'Azure Container Instances sans avoir à gérer des ressources de calcul supplémentaires.

# 6

## Développement et conteneurisation d'une application web moderne

---

Jusqu'à présent, nous avons effectué des travaux pratiques mettant en évidence les avantages de la conteneurisation avec Docker et nous avons vu l'importance de l'orchestration des conteneurs dans une architecture en microservices.

Dans ce chapitre, nous allons développer puis embarquer une application web moderne dans des conteneurs Docker et orchestrer ses services avec Kubernetes. Vous aurez ainsi un exemple d'application possible dans le monde professionnel.

Dans ce chapitre, nous allons développer et conteneuriser une application qui se décompose en une partie frontend codée avec Angular, une partie backend développée avec Spring Boot et une base de données MySQL.

### Étape 1 : développer et tester l'application en local avec un IDE

Pour fonctionner, une application ou un site web a très souvent besoin d'une partie backend (aussi back-end ou encore back) et une partie frontend (ou front-end ou encore front).

Le frontend concerne l'interface du site web et regroupe deux aspects spécifiques, à savoir le design et le langage de l'interface utilisateur. La partie backend va construire, développer et mettre en interaction trois piliers essentiels au fonctionnement de l'application :

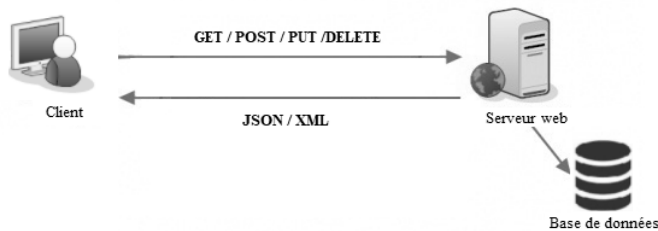
- le serveur d'hébergement ;
- l'application web ;
- la base de données.

On dit qu'un site web est dynamique s'il est construit à l'aide d'un backend. En effet, un site statique n'a pas de base de données, c'est juste une interface.

Un développeur *full stack*, aussi appelé « développeur à tout faire », est un codeur capable de réaliser la programmation d'un site ou d'une application web à la fois en frontend et backend.

Le plus souvent les développeurs utilisent des API REST pour créer des services web, souvent appelés « services web RESTful ». API REST est un style architectural qui permet aux logiciels de communiquer entre eux sur un réseau ou sur un même appareil.

**Figure 6-1**  
Services web RESTful

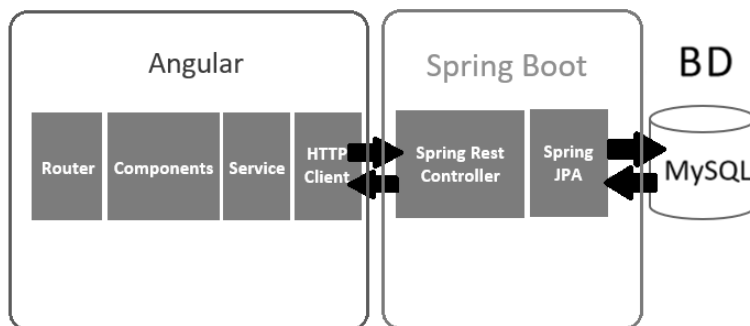


REST utilise des méthodes HTTP pour récupérer et publier des données entre un périphérique client et un serveur. Ces données peuvent être au format JSON, XML ou HTML.

Notre application est un exemple typique permettant de réaliser des opérations de base (créer, lire, modifier et supprimer) sur des étudiants enregistrés dans une base de données à travers des API REST.

Le développement a été fait sur une machine Windows 10 Professionnel à l'aide du logiciel IntelliJ IDEA, qui est un environnement de développement intégré de technologie Java. L'enregistrement a été effectué sur une base de données MySQL administrable à travers le panneau de contrôle du logiciel XAMPP.

**Figure 6-2**  
Technologies de développement  
de l'application web



Angular est un framework très populaire et puissant dans le monde du JavaScript. Développé par les équipes de Google, il apporte efficacité, rapidité et organisation pour un meilleur partage de votre travail.

C'est un framework complet proposant au développeur tous les outils nécessaires au développement d'applications web, mobiles et desktop.

L'avantage d'un framework professionnel est qu'il permet à plusieurs développeurs de travailler sur le même projet, sans se perdre dans l'organisation du code source.

Si vous êtes amené à travailler sur une application développée autour de l'architecture en microservices en Java, vous aurez tôt ou tard affaire à Spring Boot.

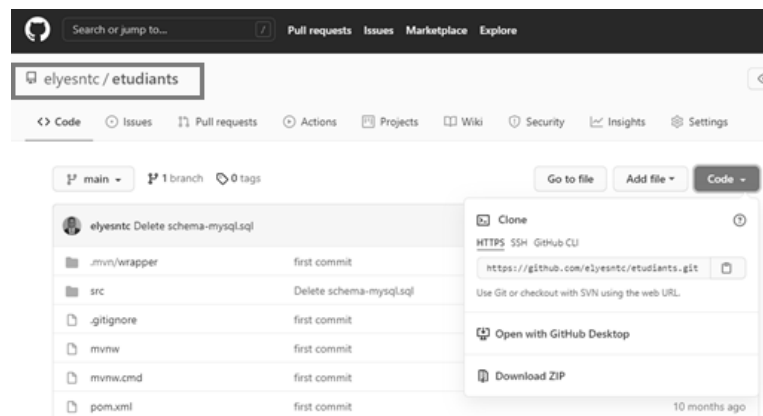
Spring Boot est un framework qui facilite le développement d'applications fondées sur Spring en offrant des outils permettant d'obtenir une application packagée en jar, totalement autonome.

Une application Spring Boot permet de développer plus rapidement en se focalisant essentiellement sur le code métier. Spring Boot propose différentes annotations afin d'appliquer rapidement des rôles et des comportements à des classes de votre application.

## Partie backend

Vous pouvez télécharger le code de l'application Spring Boot à partir du dépôt Git suivant :

**Figure 6-3**  
Référentiel Git pour  
le projet Spring Boot



Il s'agit d'un projet Maven. Maven nous a libéré des IDE, nous pouvons utiliser celui que nous voulons et en changer à tout moment, car aucune de nos tâches stratégiques n'y est liée.

Dans cette démonstration, j'utilise IntelliJ IDEA. Ce dernier est développé et maintenu par JetBrains et disponible en édition Community et Ultimate. Cet IDE riche en fonctionnalités permet un développement rapide et aide à améliorer la qualité du code.

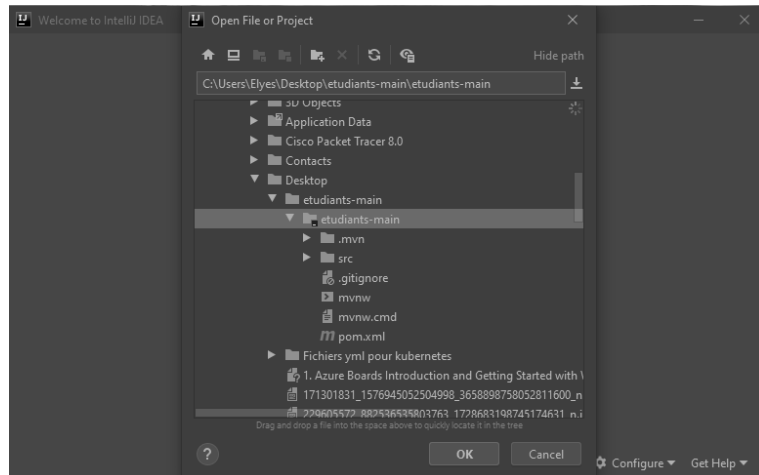
Cliquez sur *Open* (voir figure 6-4) et sélectionnez le projet que vous avez téléchargé.

Figure 6-4



Validez en cliquant sur le bouton *OK* (voir figure 6-5).

Figure 6-5

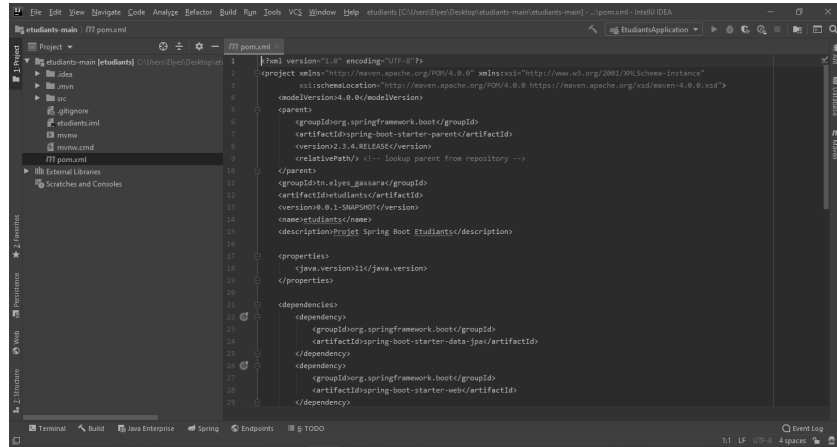


Maven est un outil permettant d'automatiser la gestion de projets Java. Il offre entre autres les fonctionnalités suivantes :

- compilation et déploiement des applications Java (.jar, .war) ;
- gestion des bibliothèques requises par l'application ;
- exécution des tests unitaires ;
- génération des documentations du projet (site web, PDF, Latex) ;
- intégration dans différents IDE.



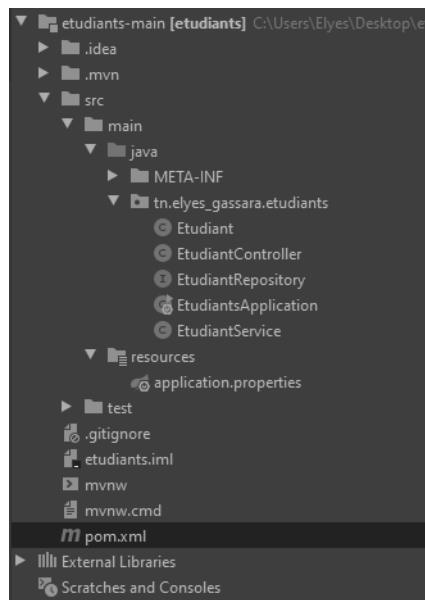
**Figure 6-6**  
Ouverture du projet Maven



Un projet Maven suit le modèle objet-projet (appelé « POM » pour *Project Object Model*). Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus encore.

Le POM prend la forme d'un fichier XML (pom.xml) qui est placé dans le répertoire de base de votre projet, à côté de deux sous-répertoires : `src` pour tout le code source et `target` pour les éléments générés.

**Figure 6-7**  
Architecture d'un projet Maven



La première partie du POM permet d'identifier le projet lui-même :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://
maven.apache.org/xsd/maven-4.0.0.xsd">
```

Version du modèle de projet

```
<modelVersion>4.0.0</modelVersion>
```

Identifiant de groupe

```
<groupId>tn.elyes_gassara</groupId>
```

Nom unique du projet au sein du groupe qui le développe

```
<artifactId>etudiants</artifactId>
```

Version du projet

```
<version>0.0.1-SNAPSHOT</version>
<name>etudiants</name>
<description>Projet Spring Boot Etudiants</description>
```

Nous devons le configurer pour en faire un projet Spring Boot. Ici, nous ajoutons un parent à notre projet Maven. Il est utilisé pour déclarer que notre projet est un enfant de ce projet parent.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.4.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Après cela, nous indiquons la version Java pour le projet.

```
<properties>
    <java.version>11</java.version>
</properties>
```

Bien sûr, pour utiliser les dépendances, elles doivent d'abord être insérées dans le fichier `pom.xml` :

Cette dépendance obtiendra les données Spring, Hibernate, HikariCP et toutes les dépendances liées aux bases de données.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Quand vous développez une application web, vous avez besoin d'une dépendance spring-boot-starter-web.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Dépendance Maven du pilote MySQL JDBC.

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Lombok est une API dont le but est de générer à la compilation du code Java à notre place.

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

Spring-boot-maven-plugin est un plug-in qui fournit des bibliothèques nécessaires pour que notre projet soit capable de s'exécuter directement sans déploiement sur le serveur web. Il aide à créer un exécutable sous la forme d'un fichier .jar.

```
<plugins>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
  </plugin>
</plugins>
```

Le fichier `pom.xml` contient donc les informations nécessaires à Maven pour traiter le projet (nom du projet, numéro de version, dépendances vers d'autres projets, bibliothèques nécessaires à la compilation, noms des contributeurs, etc.). Passons maintenant à la déclaration des entités et des méthodes.

Notre application consiste à lire les données d'une table de base de données MySQL nommée `etudiant`. Un étudiant possède les attributs `identifiant`, `prénom` et `nom` de famille ainsi que d'autres informations comme `département` et `adresse` de messagerie électronique.

Dans le projet Spring Boot, l'entité correspondante est déclarée dans le fichier `Etudiant.java` :

```
package tn.elyes_gassara.etudiants;

import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Data
@NoArgsConstructor
@Entity
public class Etudiant {

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Integer id;
    private String prenom;
    private String nom;
    private String departement;
    private String email;
}
```

Chaque entité doit avoir au moins deux annotations définies : `@Entity` et `@Id`.

- L'annotation `@Entity` spécifie que la classe est une entité et est mappée à une table de base de données.
- L'annotation `@Id` spécifie la clé primaire d'une entité et `@GeneratedValue` fournit la spécification des stratégies de génération pour les valeurs des clés primaires.

Pour pouvoir communiquer avec la base de données, nous devons également compléter les paramètres de connexion dans le fichier `application.properties` du dossier `resources`.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/etudiants
spring.datasource.username=elyes
spring.datasource.password=elyes
server.port=9001
```

Notez que l'application a besoin d'accéder à une base de données nommée `etudiants`. L'accès doit être autorisé pour l'utilisateur `elyes` avec son mot de passe.

Les applications Spring Boot incluent généralement des serveurs intégrés tels que Tomcat ou Jetty. Le port d'écoute par défaut pour ces serveurs intégrés est 8080. J'ai modifié ce port en 9001 dans ce fichier.

Nous configurerons la base de données plus tard avec ces paramètres à l'aide du logiciel XAMPP.

Il est important d'initialiser la table `etudiant`. Le fichier `src/resources/schema-mysql.sql` contient le script d'initialisation suivant :

```
CREATE TABLE IF NOT EXISTS etudiant (  
    id INT(10) PRIMARY KEY AUTO_INCREMENT,  
    prenom VARCHAR(100) NOT NULL,  
    nom VARCHAR(100) NOT NULL,  
    departement VARCHAR(100) NOT NULL,  
    email VARCHAR(100) NOT NULL  
);
```

Pour faciliter la communication avec la base de données, nous utilisons également un référentiel JPA. Pour cela, nous créons simplement une nouvelle interface Java dans le fichier `EtudiantRepository.java`.

```
package tn.elyes_gassara.etudiants;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface EtudiantRepository extends JpaRepository<Etudiant, Integer> {  
}
```

Nous créons un contrôleur approprié pour l'édition des enregistrements dans la base de données par des requêtes URL dans le fichier `EtudiantController.java` :

```
package tn.elyes_gassara.etudiants;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.*;  
  
import java.util.List;  
  
@RestController  
@CrossOrigin  
public class EtudiantController {  
    @Autowired  
    private EtudiantService service;
```

GetMapping est une spécialisation de l'annotation RequestMapping qui peut être utilisée pour mapper uniquement les demandes Get.

```
@GetMapping("/Etudiants")
public List<Etudiant> getEtudiants(){
    return service.getEtudiants();
}
```

PostMapping pour mapper uniquement les demandes POST.

```
@PostMapping("/Etudiants/ajouter")
public void ajouterEtudiant(@RequestBody Etudiant etudiant){
    service.ajouterEtudiant(etudiant);
}
```

PutMapping est une annotation pour mapper les requêtes http PUT sur des méthodes de gestionnaire spécifiques.

```
@PutMapping("/Etudiants/{id}/editer")
public void modifierEtudiant(@PathVariable("id") Integer id,@RequestBody Etudiant
etudiant){
    service.modifierEtudiant(etudiant);
}
```

DeleteMapping est une annotation permettant de mapper les requêtes http DELETE sur des méthodes de gestionnaire spécifiques.

```
@DeleteMapping("/Etudiants/{id}/supprimer")
public void supprimerEtudiant(@PathVariable("id") Integer id){
    service.supprimerEtudiant(id);
}
}
```

L'annotation Spring @Service est utilisée avec les classes qui fournissent certaines fonctionnalités métier.

Le contexte Spring détecte automatiquement ces classes lorsque la configuration basée sur des annotations et l'analyse des chemins de classe sont utilisées.

Le contenu de notre fichier EtudiantService.java est le suivant :

```
package tn.elyes_gassara.etudiants;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.util.List;

@Service
public class EtudiantService {

    @Autowired
    private EtudiantRepository repository;
    public List<Etudiant> getEtudiants(){
        return repository.findAll();
    }
    public void ajouterEtudiant(Etudiant etudiant){
        repository.save(etudiant);
    }
    public void modifierEtudiant(Etudiant etudiant){
        repository.save(etudiant);
    }
    public void supprimerEtudiant( Integer id){
        repository.deleteById(id);
    }
}
```

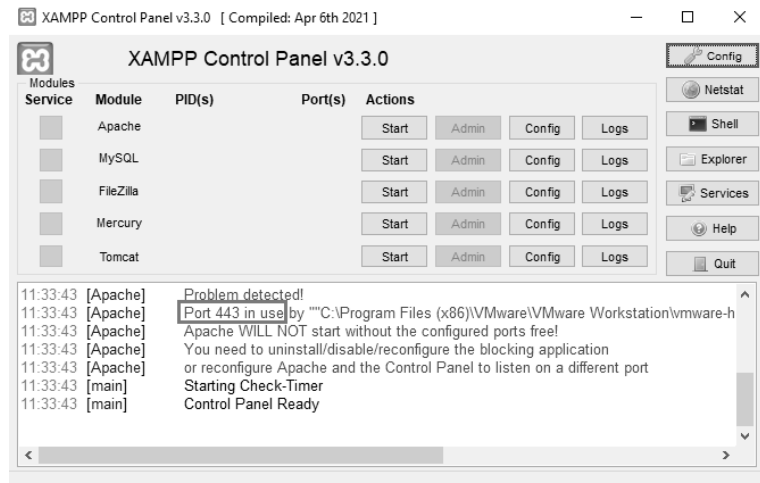
Avant de compiler et d'exécuter ce code, il faut créer la base de données `etudiants` ainsi que l'utilisateur `elyes` et accorder l'accès à cette base pour cet utilisateur. Ainsi, le code Java aura accès à cette base et l'insertion des étudiants sera désormais possible.

Pour créer la base de données, vous pouvez utiliser XAMPP qui est un ensemble de logiciels permettant de faire tourner des sites web en local.

Téléchargez et installez cette application et exécutez le XAMPP Control Panel pour avoir accès à l'interface du logiciel.

Les logiciels qui composent XAMPP sont appelés des « modules ». Pour notre application, nous avons besoin des modules Apache et MySQL.

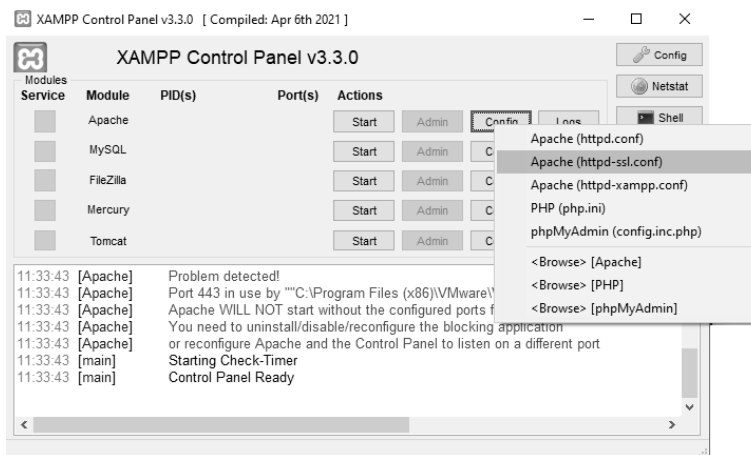
**Figure 6-8**  
Panneau de contrôle  
du logiciel XAMPP



Notez que les ports d'écoute d'Apache sont déjà utilisés dans ma machine par d'autres logiciels. Je vais donc modifier ces numéros de ports par défaut.

Cliquez sur le bouton *Config* correspondant au service Apache et choisissez successivement les fichiers `httpd-ssl.conf` et `httpd.conf`, comme indiqué sur la figure 6-9.

Figure 6-9



Je vais modifier à titre d'exemple le port HTTPS de 443 à 4433 (voir figure 6-10) et le port httpd de 80 à 8000 (voir figure 6-11).

Figure 6-10

```
#
# When we also provide SSL we have to listen to the
# standard HTTP port (see above) and to the HTTPS port
#
Listen 4433
```

Figure 6-11

```
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 8000
```

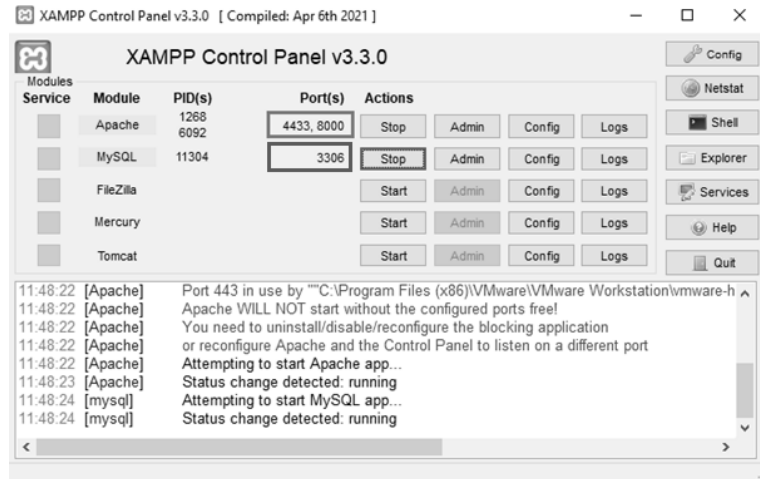
Vous pouvez chercher le terme « Listen » dans chacun des deux fichiers et changer le port correspondant.

Démarrez maintenant les services MySQL et Apache et vérifiez que les ports d'écoute sont bien modifiés. Pour MySQL, notez que le port d'écoute par défaut est 3306.

Notre application a besoin d'une base de données nommée `etudiants` dans laquelle nous allons stocker des étudiants.



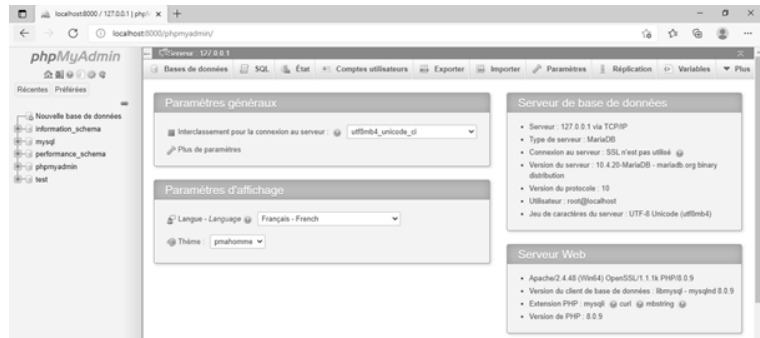
Figure 6-12



Commençons donc par créer cette base de données en accédant à la page d'administration de MySQL. Pour cela, il suffit de cliquer sur le bouton *Admin* et d'ajouter dans l'URL le nouveau port de notre serveur web : `http://localhost:8000/phpmyadmin`.

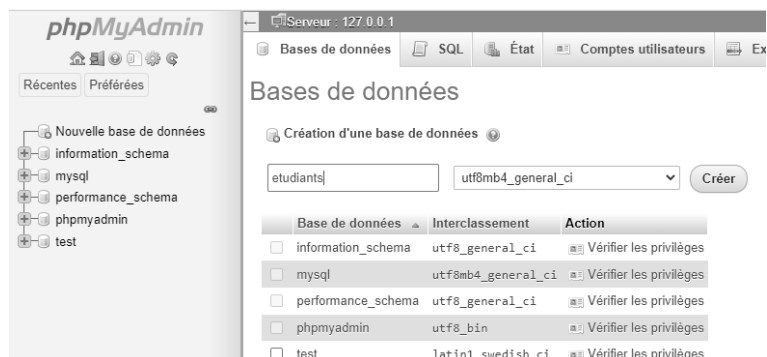
Figure 6-13

Page d'administration de MySQL



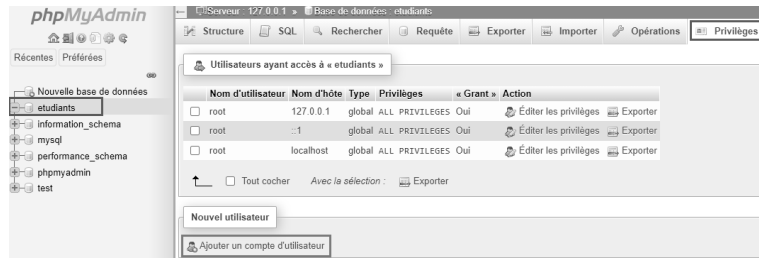
Cliquez sur l'onglet *Bases de données* et saisissez le nom de la base *etudiants*. Cliquez sur le bouton *Créer* pour valider (voir figure 6-14).

Figure 6-14



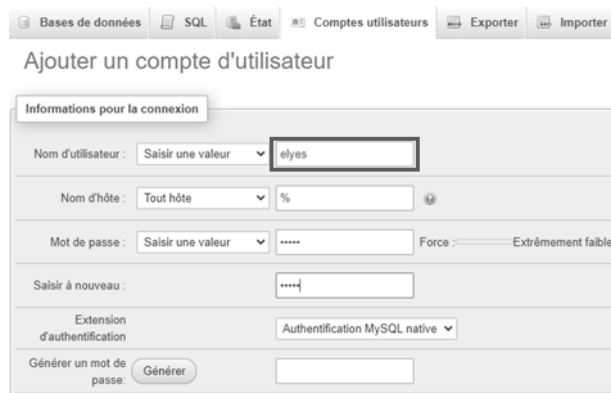
Sélectionnez la base étudiants dans la liste de gauche et cliquez sur l'onglet *Privèlèges* (voir figure 6-15).

Figure 6-15



Vous pouvez donc ajouter un compte pour l'utilisateur elyes et spécifier son mot de passe, comme sur la figure 6-16.

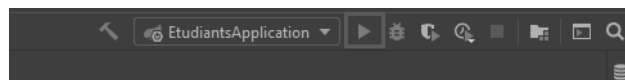
Figure 6-16



N'oubliez pas de cliquer sur le bouton *Exécuter* tout en bas de la page. L'accès à la base est désormais autorisé pour l'utilisateur elyes.

À ce niveau, nous pouvons tester notre code Java. Cliquez sur le bouton *Exécuter*, représenté par une flèche (voir figure 6-17).

Figure 6-17



Si tout va bien, le serveur sera lancé sur le port http 9001 comme mentionné dans le fichier `application.properties` (figure 6-18).

Pour tester, saisissez l'URL `http://localhost:9001/Etudiants` et notez que la base est encore vide (figure 6-19).

Figure 6–18

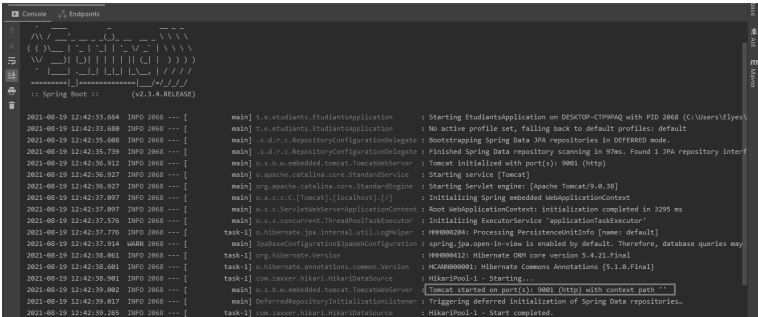
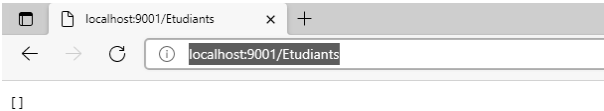


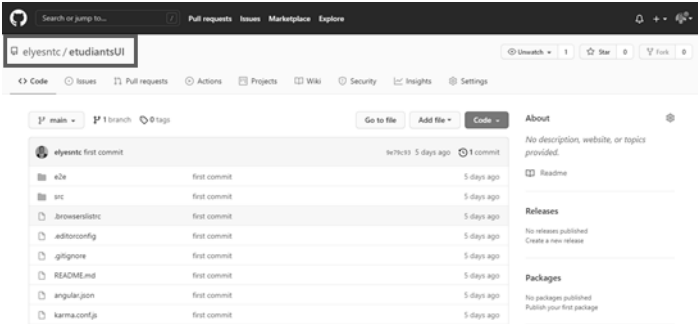
Figure 6–19



## Partie frontend

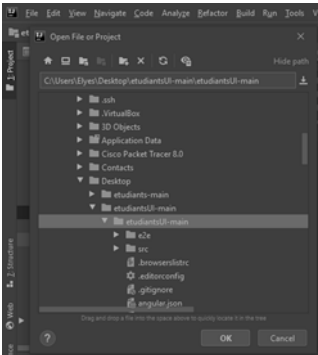
Pour ajouter des étudiants, il faut passer par la partie frontend codée avec Angular. Vous pouvez télécharger le code à partir du dépôt Git suivant :

Figure 6–20



Ouvrez alors ce projet avec IntelliJ IDEA dans une nouvelle fenêtre :

Figure 6–21

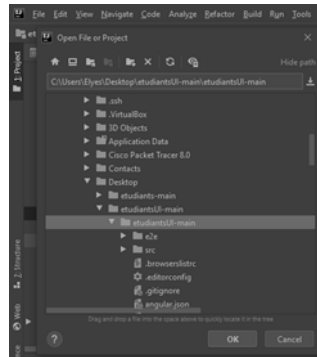


Angular est un framework permettant de créer la partie frontend des applications web en utilisant les langages HTML et JavaScript ou TypeScript compilé en JavaScript. Une application Angular se compose d'un à plusieurs modules dont chaque module peut inclure :

- des composants web : la partie visible de l'application web (IHM) ;
- des services pour la logique applicative ;
- les directives : un composant peut utiliser des directives ;
- les pipes : utilisés pour formater l'affichage des données dans les composants.

**Figure 6–22**

Architecture d'un projet Angular



Un module est un bloc de code conçu pour effectuer une seule tâche. Ce bloc peut contenir les éléments suivants : Component, Service, Directive ou Pipe.

Angular possède son propre système de modularité appelé « modules angulaires » ou « NgModules ». Chaque application Angular possède au moins une classe de module angulaire : le module classique appelé AppModule.

Un module est une classe avec un décorateur @NgModule.

Vous trouverez les propriétés suivantes dans le fichier `src/app/app.module.ts` :

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    EtudiantComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    NgbModule,
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Les propriétés les plus importantes sont :

- les déclarations : la classe représentant le module ;
- les exports : pour exporter les classes utilisables dans d'autres modules ;
- les imports : pour importer d'autres modules ;
- les providers : pour déclarer les fabriques de services ;
- bootstrap : pour déclarer le composant racine du module. Seul le module racine doit définir cette propriété.

Le fichier `package.json` situé à la racine de l'application définit les bibliothèques qui seront installées dans `node_modules` lors de l'exécution de la commande `npm install`.

La figure 6-23 présente un extrait de notre fichier `package.json`.

Figure 6-23

```
1 {
2   "name": "etudiants-ui", Nom du projet
3   "version": "0.0.0", Version du projet
4   "scripts": {
5     "ng": "ng",
6     "start": "ng serve",
7     "build": "ng build",
8     "test": "ng test",
9     "lint": "ng lint",
10    "e2e": "ng e2e"
11  },
12  "private": true, Le projet est privé
13  "dependencies": {
14    "@angular/animations": "~10.2.0",
15    "@angular/common": "~10.2.0",
16    "@angular/compiler": "~10.2.0",
17    "@angular/core": "~10.2.0",
18    "@angular/forms": "~10.2.0",
19    "@angular/localize": "~10.2.0",
20    "@angular/platform-browser": "~10.2.0",
21    "@angular/platform-browser-dynamic": "~10.2.0",
22    "@angular/router": "~10.2.0",
23    "@ng-bootstrap/ng-bootstrap": "^7.0.0",
24    "angular-material": "^1.2.1",
25    "bootstrap": "^4.5.3",
26    "font-awesome": "^4.7.0",
```

Cette section décrit les scripts  
Node que vous pouvez exécuter  
dans votre application.

La liste des  
packages installés  
en tant que  
dépendances pour  
ce projet est requise  
lors de l'exécution.

Le fichier `app.component.html` présent dans le dossier `src/app` est le rendu visuel de la première page de notre projet. Si vous changez un élément HTML, le changement sera automatiquement et immédiatement affiché sur le navigateur à l'adresse `localhost:4200`, qui est l'adresse par défaut de tous les projets créés par Angular CLI.

Le fichier `app.component.ts` contient la définition du composant. La propriété `title` sera consommée et évaluée par le template (`app.component.html`). Si vous changez sa valeur, elle sera modifiée immédiatement dans le navigateur.

Nous avons défini la propriété `selector` du composant à `app-root`. Voici le contenu de ce fichier dans notre application :

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'EtudiantsUI';
}
```

Le fichier `index.html` est la page principale chargée par Angular. Dans cette page, nous choisissons le composant à appeler parmi plusieurs autres.

On fait appel à `<app-root>` qui est le sélecteur de notre composant. Donc notre page principale, `index.html` va chercher le composant ayant ce nom et elle le charge.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>EtudiantsUI</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <p><div style="text-align: center;"><b>Formation animée par : Elyes Gassara</b></div>
  <app-root></app-root>
</body>
</html>
```

Nous devons aller à l'API Spring Boot pour vérifier le point de terminaison du contrôleur et pouvoir effectuer les quatre opérations de base sur des données. En anglais, ces opérations sont désignées à l'aide de l'acronyme CRUD (*Create, Read, Update, Delete*) :

- créer : ajouter, insérer ;
- lire : extraire, lister, consulter, interroger, rechercher ;
- mettre à jour : modifier, éditer ;
- supprimer : effacer.

Pour comprendre le principe, examinons le code permettant d'effectuer la mise à jour des étudiants. Dans le fichier `EtudiantController.java`, étudions la méthode `modifierEtudiant` à titre d'exemple :

```
@PostMapping("/Etudiants/{id}/editer")
public void modifierEtudiant(@PathVariable("id") Integer id,@RequestBody Etudiant
etudiant){
    service.modifierEtudiant(etudiant);
}
```

L'URL de demande d'édition serait donc `http://localhost:9001/Etudiants/{id}/editer`.

Revenons maintenant à Angular. Tout d'abord, nous ajoutons l'événement au bouton *Enregistrer* dans le mode d'édition. Ensuite, nous écrivons la méthode.

Placez l'événement `click` sur le bouton *Enregistrer* dans le fichier `etudiant.component.html`.

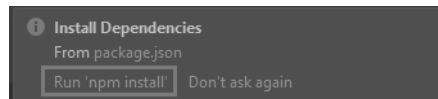
```
<div class="border text-center">
  <button data-dismiss="modal" class="btn btn-info"
(click)="onSave()">Enregistrer</button>
</div>
```

Écrivez à présent la méthode `onSave()` dans le fichier `etudiant.component.ts` comme indiqué ci-dessous :

```
onSave() {
    const editURL = 'http://localhost:9001/Etudiants/' + this.editForm.value.id + '/'
    + 'editer';
    console.log(this.editForm.value);
    this.httpClient.put(editURL, this.editForm.value)
    .subscribe((results) => {
        this.ngOnInit();
        this.modalService.dismissAll();
    });
}
```

Après l'ouverture du projet, vous devez installer les dépendances. Une petite fenêtre en bas, représentée dans la figure 6-24, permet de le faire.

Figure 6-24



Cette étape va prendre quelques minutes (voir figure 6-25). Attendez jusqu'à la fin.

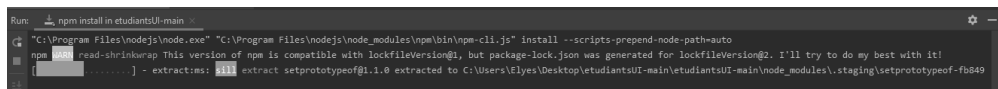
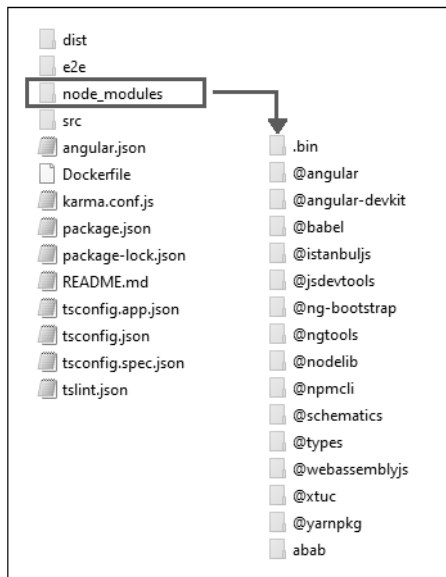


Figure 6-25

Lorsque vous exécutez cette commande (`npm install`), les bibliothèques définies dans le fichier `package.json` seront installées dans le dossier `node_modules`, détaillé dans la figure 6-26.

Figure 6-26



Lancez ensuite l'exécution de ce projet.

Cliquez alors sur le lien en bleu qui s'affiche en bas pour lancer le site (voir figure 6-27).

Figure 6-27

```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help etudiantsUI-main [C:\Users\Elyes\Desktop\etudiantsUI-main]
Project etudiantsUI-main
Run: Angular CLI Server
"C:\Program Files\nodejs\node.exe" "C:\Program Files\nodejs\node_modules\npm\bin\npm-cli.js" run start --scripts-prepare
> etudiants-ui@0.0.0 start C:\Users\Elyes\Desktop\etudiantsUI-main\etudiantsUI-main
> ng serve

Compiling @angular/core : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/forms : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
Compiling @angular/router : es2015 as esm2015
Compiling @angular/common/http : es2015 as esm2015
Compiling @ng-bootstrap/ng-bootstrap : es2015 as esm2015

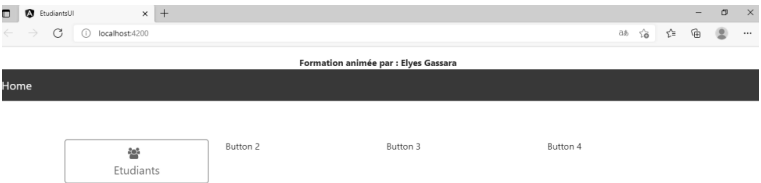
chunk {main} main.js, main.js.map (main) 54.4 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 150 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 1.66 MB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.65 MB [initial] [rendered]
Date: 2021-08-19T12:14:15.158Z - Hash: 565234a28f051b8fe0da - Time: 20562ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
: Compiled successfully.

```

Il s'agit d'une page d'accueil contenant un bouton permettant d'accéder à une page grâce à laquelle il est possible d'ajouter et d'éditer les étudiants enregistrés dans la base de données.



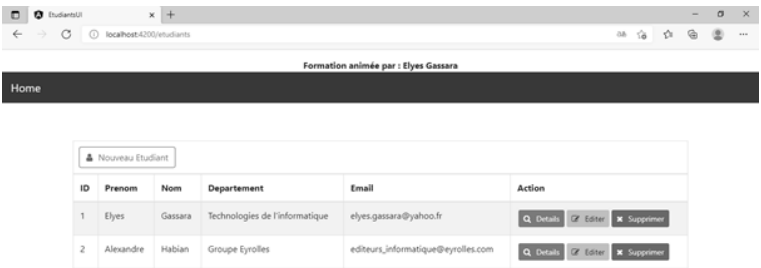
**Figure 6–28**  
Page d'accueil de l'application  
Angular



C'est un exemple simple avec une seule table dans la base mais le principe sera le même quelle que soit la taille de votre site.

Cliquez sur le bouton *Nouveau Etudiant* et ajoutez des étudiants (voir figure 6–29).

**Figure 6–29**



Vous pouvez afficher les détails d'un étudiant, l'éditer ou le supprimer (voir figure 6–30).

**Figure 6–30**



## Étape 2 : embarquer l'application dans des conteneurs Docker

Afin d'adapter notre application au nouveau contexte de conteneurisation, nous devons apporter quelques modifications au fichier `application.properties`.

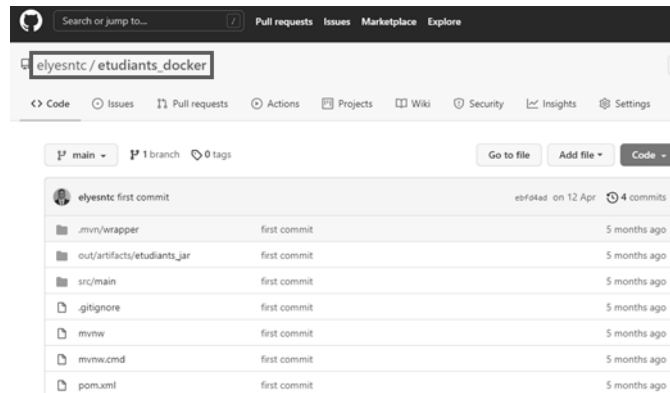
```
spring.datasource.url=jdbc:mysql://mysql-db/etudiants
spring.datasource.username=root
spring.datasource.password=elyes
spring.datasource.platform=mysql
spring.datasource.initialization-mode=always
```

Notez que dans ce cas, je vais me connecter à la base en utilisant l'utilisateur `root`. L'accès à la base se fait par le nom du service `mysql-db`, conformément à la logique Docker qui attribue une adresse IP à chaque service et s'occupe de toute traduction de nom de service en adresse IP.

Il faut donc respecter ce nommage lors de la création du service MySQL dans le fichier `docker-compose.yml`.

Le projet Git contenant le code avec cette modification est le suivant :

Figure 6-31



L'application a été développée sous Windows. Pour mettre en évidence l'indépendance des projets Maven, je vais la tester sur un système Linux Ubuntu.

Dans le dossier racine contenant le projet Spring Boot, j'ai lancé un terminal et exécuté cette commande :

```
elyes@ubuntu:~/Avec docker/etudiants-main$ mvn clean package -Dmaven.test.skip=true
```

J'ai ignoré l'étape de test car l'accès à la base de données n'est pas possible avant de lancer un conteneur de base de données.

Le fichier `.jar` sera alors généré dans le répertoire `target` sous le nom `etudiants-0.0.1-SNAPSHOT.jar` comme indiqué ci-après :

```
[INFO] -----< tn.elyes_gassara:etudiants >-----
[INFO] Building etudiants 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ etudiants ---
[INFO] Deleting /home/elyes/Avec docker/etudiants-main/target
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ etudiants ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ etudiants ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 5 source files to /home/elyes/Avec docker/etudiants-main/target/classes
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ etudiants ---
[INFO] Not copying test resources
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ etudiants ---
[INFO] Not compiling test sources
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ etudiants ---
[INFO] Tests are skipped.
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ etudiants ---
[INFO] Building jar: /home/elyes/Avec docker/etudiants-main/target/etudiants-0.0.1-SNAPSHOT.jar
[INFO] --- spring-boot-maven-plugin:2.3.4.RELEASE:repackage (repackage) @ etudiants ---
[INFO] Replacing main artifact with repackaged archive
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 40.956 s
[INFO] Finished at: 2021-10-10T03:17:24-07:00
[INFO] -----
```

Mettons alors ce fichier JAR dans un répertoire contenant le fichier Dockerfile suivant :

```
FROM openjdk:11
COPY etudiants-0.0.1-SNAPSHOT.jar /usr/src/etudiants.jar
WORKDIR /usr/src
CMD ["java","-jar","etudiants.jar"]
```

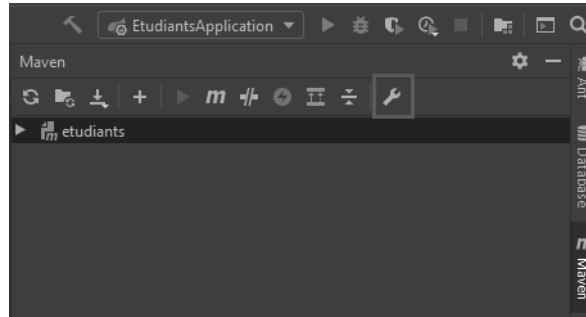
Dans l'exemple ci-dessus, nous utilisons l'image `openjdk:11` comme image de base puisque nous avons développé notre application avec Java 11.

Pour créer l'image :

```
elyes@ubuntu:~/Avec docker$ docker build -t elyesntc/springboot:latest
```

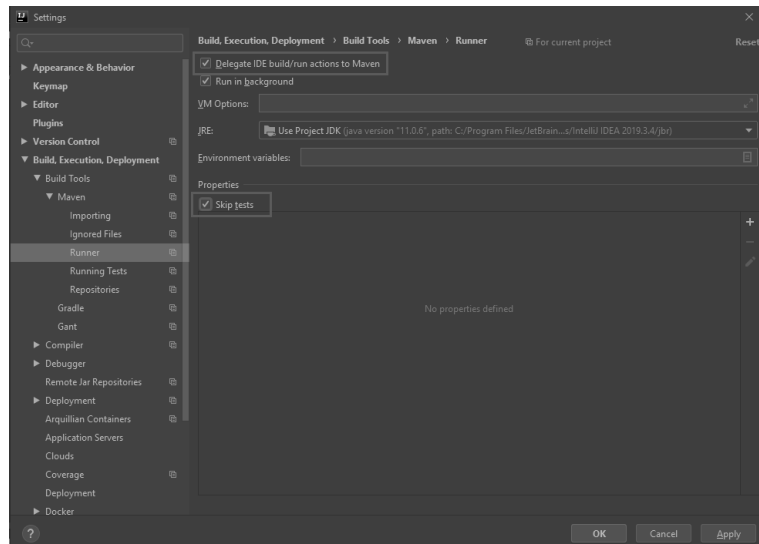
Si vous voulez compiler votre projet Maven directement avec IntelliJ IDEA sous Windows et non pas sous Linux, accédez aux paramètres Maven :

Figure 6–32



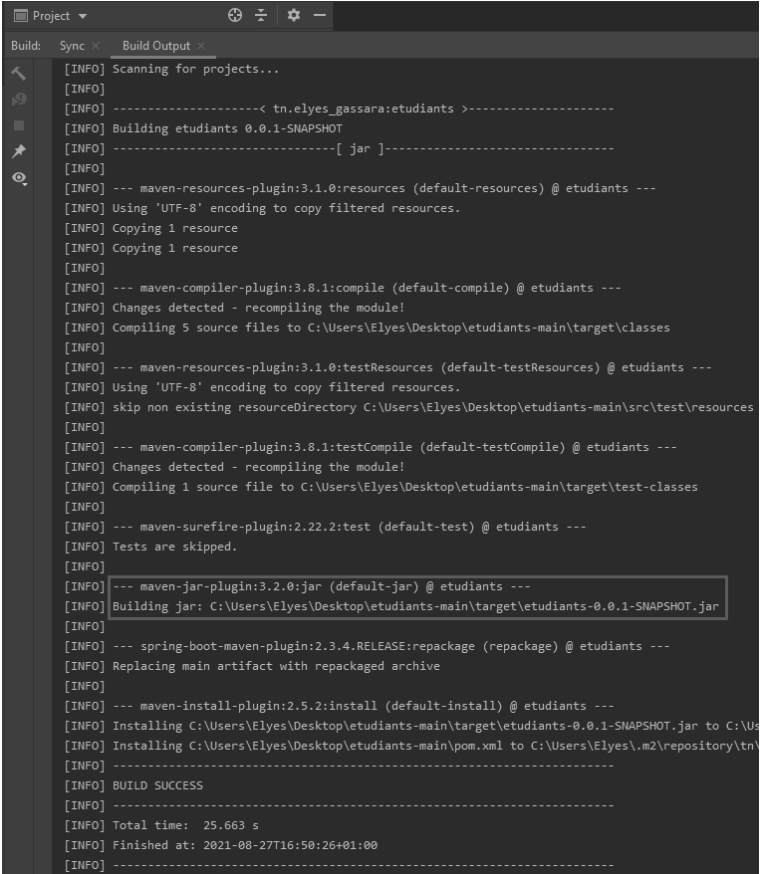
Dans le menu *Maven*, choisissez *Runner* puis cochez la case *Déléguer les actions de construction/exécution de l'IDE à Maven* (*Delegate IDE build/run actions to Maven*) et la case *Sauter les tests* (*Skip tests*), comme indiqué sur la figure 6-33. Appliquez les changements et validez en cliquant sur *OK*.

Figure 6–33



Compilez le projet.

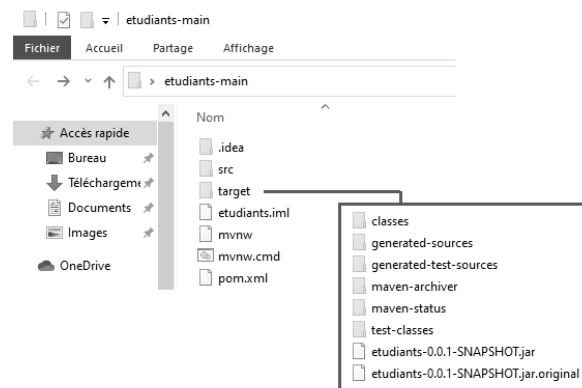
Figure 6–34



```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< tn.elyes_gassara:etudiants >-----
[INFO] Building etudiants 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ etudiants ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ etudiants ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 5 source files to C:\Users\Elyes\Desktop\etudiants-main\target\classes
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ etudiants ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\Elyes\Desktop\etudiants-main\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ etudiants ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\Elyes\Desktop\etudiants-main\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ etudiants ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ etudiants ---
[INFO] Building jar: C:\Users\Elyes\Desktop\etudiants-main\target\etudiants-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.3.4.RELEASE:repackage (repackage) @ etudiants ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ etudiants ---
[INFO] Installing C:\Users\Elyes\Desktop\etudiants-main\target\etudiants-0.0.1-SNAPSHOT.jar to C:\Users\Elyes\
[INFO] Installing C:\Users\Elyes\Desktop\etudiants-main\pom.xml to C:\Users\Elyes\.m2\repository\tn
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 25.663 s
[INFO] Finished at: 2021-08-27T16:50:26+01:00
[INFO]
[INFO] -----
```

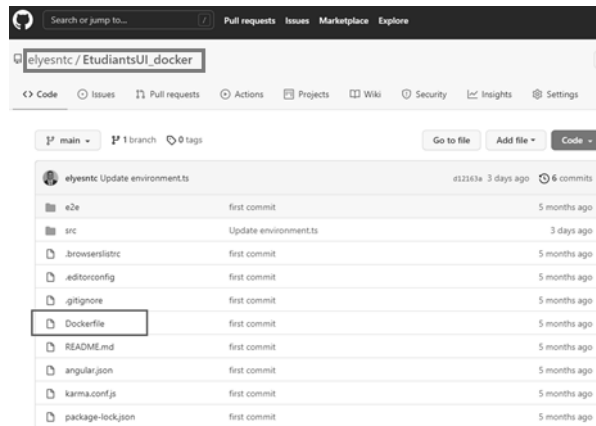
Le fichier JAR est créé dans le répertoire target.

Figure 6–35



Il faut maintenant créer la deuxième image contenant l'application Angular. Le projet ainsi que le fichier Dockerfile existe dans le projet Git suivant :

Figure 6–36



Pour autoriser l'accès à notre application Angular, qui s'exécutera dans un conteneur Docker à partir de l'extérieur, il faut apporter la modification suivante dans le fichier `package.json` :

```
"name": "etudiants-ui",
"version": "0.0.0",
"scripts": {
  "ng": "ng",
  "start": "ng serve --host=0.0.0.0",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e"
},
```

Le contenu du fichier Dockerfile est le suivant :

```
FROM node:latest as node
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 4200
CMD ["npm", "start", "--host=0.0.0.0"]
```

Et la commande permettant la génération de l'image :

```
elyes@ubuntu:~/Avec docker/EtudiantsUI_docker-main$ docker build -t elyesntc/
angular:latest .
```

Les images Docker sont maintenant prêtes. Il ne reste qu'à lancer les trois services MySQL, Spring Boot et Angular avec le fichier `docker-compose.yml` suivant :

```
version: "3"
services:
  frontend:
    image: elyesntc/angular:latest
    depends_on:
      - backend
    networks:
      - frontend-network
    ports:
      - "4200:4200"

  backend:
    image: elyesntc/springboot:latest
    ports:
      - "9001:8080"
    networks:
      - backend-network
      - frontend-network
    depends_on:
      - mysqladb

  mysqladb:
    image: mysql
    networks:
      - backend-network
    environment:
      - MYSQL_ROOT_PASSWORD=elyes
      - MYSQL_DATABASE=etudiants

networks:
  frontend-network:
  backend-network:
```

Pour l'application, utilisez la commande `docker-compose up -d` :

```
elyes@ubuntu:~/Avec docker$ docker-compose up -d
```

```
Creating network "avedocker_backend-network" with the default driver
Creating network "avedocker_frontend-network" with the default driver
Creating avedocker_mysqladb_1 ... done
Creating avedocker_backend_1 ... done
Creating avedocker_frontend_1 ... done
```

Vérifions les conteneurs en cours d'exécution :

```
elyes@ubuntu:~/Avec docker$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
b4cc12544886	elyesntc/angular:latest	"docker-entrypoin..."	5 minutes ago
Up 5 minutes	0.0.0.0:4200->4200/tcp	avedocker_frontend_1	
06519055539c	elyesntc/springboot:latest	"java -jar etudiants..."	5 minutes ago
Up About a minute	0.0.0.0:9001->8080/tcp	avedocker_backend_1	
08b62e14a2b6	mysql	"docker-entrypoin..."	5 minutes ago
Up 5 minutes	3306/tcp, 33060/tcp	avedocker_mysqlldb_1	

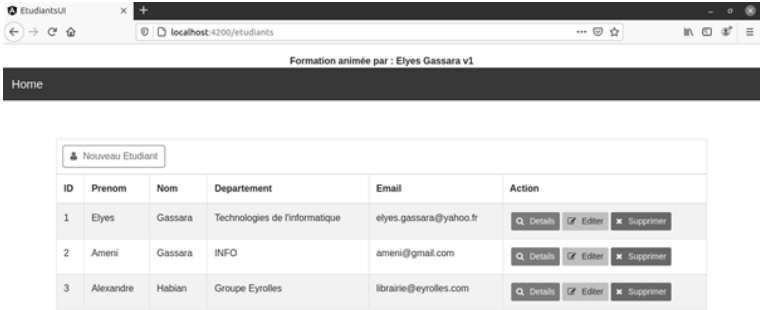
Les trois conteneurs sont bien en cours d'exécution et les services sont en écoute sur les ports définis. Accédons à présent à notre page d'accueil :

Figure 6-37



Il faut ajouter des étudiants pour tester la communication avec la base de données :

Figure 6-38



Vous pouvez lancer un Shell à l'intérieur du conteneur de la base de données et l'interroger directement :

```
elyes@ubuntu:~/Avec docker$ docker exec -it avecdocker_mysqlldb_1 /bin/bash

root@08b62e14a2b6:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 8.0.26 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.
```





```
2021-10-10 10:29:08.136 INFO 1 --- [          main]
org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/
9.0.38]
2021-10-10 10:29:08.258 INFO 1 --- [          main] o.a.c.c.C.[Tomcat].[localhost].[/]
: Initializing Spring embedded WebApplicationContext
2021-10-10 10:29:08.258 INFO 1 --- [          main]
w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization
completed in 2301 ms
```

Et pour visualiser les journaux de la partie frontend :

```
elyes@ubuntu:~/Avec docker$ docker logs avecdocker_frontend_1
```

```
> etudiants-ui@0.0.0 start
> ng serve --host=0.0.0.0
```

WARNING: This is a simple server for use in testing or debugging Angular applications locally. It hasn't been reviewed for security issues.

Pour arrêter et supprimer les conteneurs ainsi que les volumes, utilisez cette commande :

```
elyes@ubuntu:~/Avec docker$ docker-compose down --volume
```

```
Stopping avecdocker_frontend_1 ... done
Stopping avecdocker_backend_1 ... done
Stopping avecdocker_mysqldb_1 ... done
Removing avecdocker_frontend_1 ... done
Removing avecdocker_backend_1 ... done
Removing avecdocker_mysqldb_1 ... done
Removing network avecdocker_backend-network
Removing network avecdocker_frontend-network
```

## Étape 3 : déployer l'application sur un serveur de production avec Kubernetes

Vous pouvez directement tester l'application en utilisant les images Docker que j'ai déposées sur Docker Hub, mais je vous recommande de créer et d'utiliser vos propres images pour bien maîtriser cette technologie.

Via le tableau de bord de Minikube, commençons par créer le service MySQL et exposer le port 3306 (voir figure 6-39).

Il est important de garder le nom de l'application tel qu'il est codé dans le backend, soit `mysqldb`. Toute modification de ce nom empêcherait la connexion à la base de données.

Figure 6-39

**kubernetes** default Recherche

**Create**

**Workloads** N

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

**Service** N

- Ingresses
- Services

**Config and Storage**

Créer en ligne   Créé depuis un fichier   Créé depuis un formulaire

Nom de l'application \*  
mysql 7 / 24

Image du conteneur \*  
mysql

Nombre de pods \*  
1

Service \*  
Internal

Port \*  
3306

Port cible \*  
3306

Protocole \*  
TCP

N'oubliez pas d'afficher les options avancées pour ajouter les variables d'environnement contenant le nom de la base de données et le mot de passe root :

Figure 6-40

**Variables d'environnement**


Nom	Valeur	
MYSQL_DATABASE	etudiants	
MYSQL_ROOT_PASSWORD	elyes	
Nom	Valeur	

**Déployer**   Annuler   Afficher les options avancées

Nous allons maintenant ajouter le service backend. Afin de suivre la démarche précédente, le port 9001 sera exposé alors que le backend écoute sur le port par défaut 8080, comme indiqué sur la figure 6-41.

Vous pouvez éditer le service backend pour ajouter manuellement une terminaison externe (adresse IP externe) car ces terminaisons ne sont pas disponibles automatiquement par Minikube et elles sont généralement fournies par le fournisseur de service cloud lorsqu'on choisit le service LoadBalancer.

Figure 6–41

kubernetes

default

Recherche

Create

Workloads N

Cron Jobs  
Daemon Sets  
Deployments  
Jobs  
Pods  
Replica Sets  
Replication Controllers  
Stateful Sets

Service N

Ingresses  
Services

Config and Storage

Créer en ligneCréer depuis un fichierCréer depuis un formulaire

Nom de l'application \*

backend

7 / 24

Image du conteneur \*

elyesntc/springboot:latest

Nombre de pods \*

1

Service \*

External

Port \*

9001

Port cible \*

8080

Protocole \*

TCP

Figure 6–42

Éditer une ressource

YAMLJSON

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

'f:type': {}

spec:

ports:

- name: tcp-9001-8080-bl6vr

protocol: TCP

port: 9001

targetPort: 8080

nodePort: 32512

selector:

k8s-app: backend

clusterIP: 10.104.163.68

clusterIPs:

- 10.104.163.68

type: LoadBalancer

externalIPs:

- 20.128.0.0

sessionAffinity: None

externalTrafficPolicy: Cluster

status:

loadBalancer: {}

Mettre à jourAnnuler

Remarque

L'adresse IP que nous avons indiquée est celle de la machine virtuelle exécutant Kubernetes. Vous pouvez trouver cette adresse en exécutant la commande `minikube ssh` puis `ip a s` :

Figure 6-43

```

C:\Windows\system32\cmd.exe - minikube ssh
Microsoft Windows [version 10.0.19042.1165]
(c) Microsoft Corporation. Tous droits réservés.

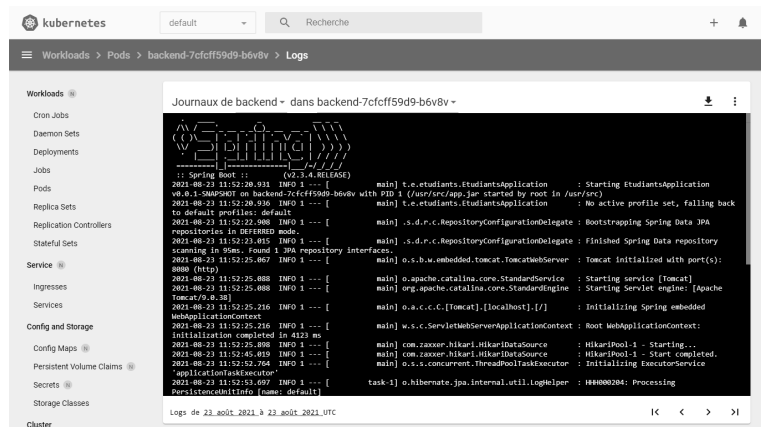
C:\Users\Elyes>minikube ssh

$ ip a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0c:29:38:14:1e brd ff:ff:ff:ff:ff:ff
    inet 20.128.0.0/8 brd 20.255.255.255 scope global dynamic eth0
        valid_lft 1741sec preferred_lft 1741sec

```

Vous pouvez voir les journaux du pod exécutant le code Java :

Figure 6-44



Avant de poursuivre, une bonne pratique consiste à vérifier le fonctionnement de la partie backend :

Figure 6-45



La connexion à la base de données s'est effectuée avec succès mais nous n'avons pas encore ajouté des étudiants.

Il ne reste qu'à s'occuper du service frontend. Pour adapter notre application Angular avec le contexte actuel, c'est-à-dire dans lequel le backend ne s'exécute plus dans la machine hôte d'adresse localhost, une petite rectification consiste à modifier les fichiers `src/environments/environment.prod.ts` et `src/environments/environment.ts` pour indiquer l'adresse IP sur laquelle seront envoyées les requêtes :

```
export const environment = {  
  production: true,  
  apiEndPoint: "20.128.0.0"  
};
```

Lorsque vous générez ou exécutez votre projet dans un environnement de développement, le fichier `environment.ts` sera utilisé. Alors que si vous exécutez votre projet en mode production, le fichier `environment.prod.ts` sera utilisé.

Notez que l'adresse IP sur laquelle sont exposées les API (l'adresse IP de la machine exécutant le backend) sera conservée dans une variable `apiEndPoint`. Déclarez alors cette variable dans le fichier `etudiant.component.ts` et effectuez les modifications dans les méthodes en remplaçant le terme `localhost` par `this.apiEndPoint`.

```
export class EtudiantComponent implements OnInit {  
  
  etudiants: Etudiant[];  
  closeResult: string;  
  editForm: FormGroup;  
  private deleteId: number;  
  apiEndPoint:string="";  
  
  constructor(  
    private httpClient: HttpClient,  
    private modalService: NgbModal,  
    private fb: FormBuilder  
  ) {  
    this.apiEndPoint = environment.apiEndPoint;  
  }  
  
  onSave() {  
    const editURL = 'http://' + this.apiEndPoint + ':9001/Etudiants/' +  
    this.editForm.value.id + '/editor';  
    console.log(this.editForm.value);  
    this.httpClient.put(editURL, this.editForm.value)  
      .subscribe((results) => {  
        this.ngOnInit();  
        this.modalService.dismissAll();  
      });  
  }  
}
```

Il faut alors reconstruire l'image. Ici j'ai ajouté le tag `v2` pour cette version (voir figure 6-46).

Figure 6-46

The screenshot shows the 'Create' page in the Kubernetes dashboard. On the left, a sidebar lists various resources: Workloads (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets), Service (Ingresses, Services), and Config and Storage. The 'Services' option is selected. The main area has three tabs: 'Créer en ligne' (selected), 'Créer depuis un fichier', and 'Créer depuis un formulaire'. The form fields are as follows: 'Nom de l'application \*' is 'frontend'; 'Image du conteneur \*' is 'elyesntc/angular.v2'; 'Nombre de pods \*' is '1'; 'Service \*' is 'External'; 'Port \*' is '4200'; 'Port cible \*' is '4200'; and 'Protocole \*' is 'TCP'.

Le port 4200 sera exposé. Vous pouvez aussi spécifier l'adresse IP externe pour ce service :

Figure 6-47

Éditer une ressource

```
YAML  JSON
34 | | | | 'f:k8s-app': {}
35 | | | | 'f:sessionAffinity': {}
36 | | | | 'f:type': {}
37 | spec:
38 |   ports:
39 |     - name: tcp-4200-4200-z4xlr
40 |       protocol: TCP
41 |       port: 4200
42 |       targetPort: 4200
43 |       nodePort: 31507
44 |   selector:
45 |     k8s-app: frontend
46 |   clusterIP: 10.106.38.143
47 |   clusterIPs:
48 |     - 10.106.38.143
49 |   externalIPs:
50 |     - 20.128.0.0
51 |   type: LoadBalancer
52 |   sessionAffinity: None
53 |   externalTrafficPolicy: Cluster
54 | status:
```

Mettre à jour Annuler

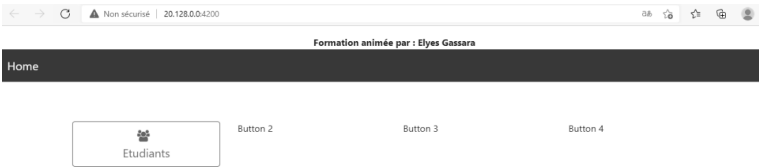
Voici un aperçu des services :

Figure 6–48

Services						
Nom	Espace de nom	Étiquettes	IP cluster	Terminaisons internes	Terminaisons externes	Date de création
✔ frontend	default	k8s-app: frontend	10.106.38.143	frontend:4200 TCP frontend:31507 TCP	20.128.0.0-4200	5 minutes ago
✔ backend	default	k8s-app: backend	10.104.163.68	backend:9001 TCP backend:32512 TCP	20.128.0.0-9001	20 minutes ago
✔ mysqlpdb	default	k8s-app: mysqlpdb	10.98.199.249	mysqlpdb:3306 TCP mysqlpdb:0 TCP	-	23 minutes ago
✔ kubernetes	default	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	5 months ago

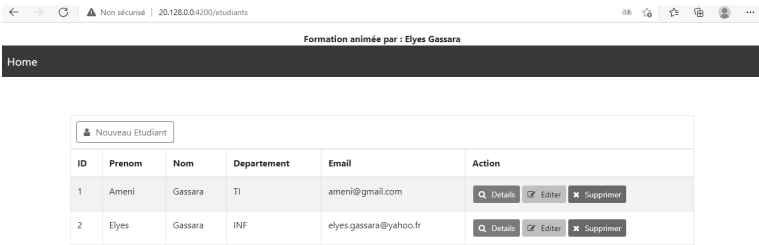
Il ne reste qu'à tester l'application :

Figure 6–49



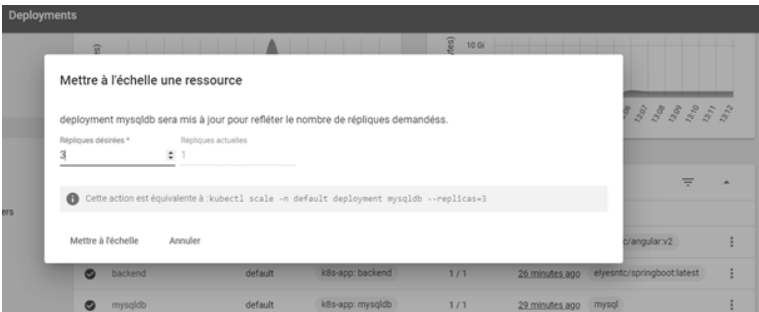
Ajoutez des étudiants :

Figure 6–50



Nous pouvons mettre à jour nos déploiements, par exemple trois répliques pour la base de données :

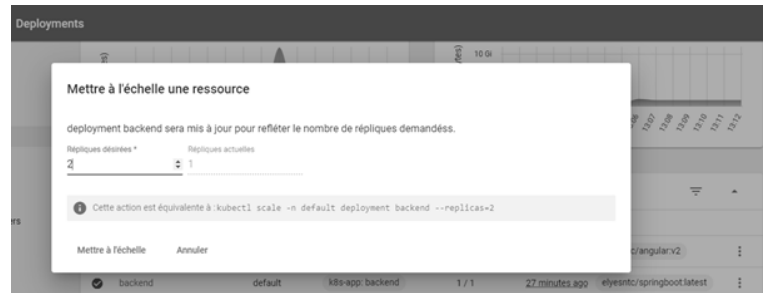
Figure 6–51





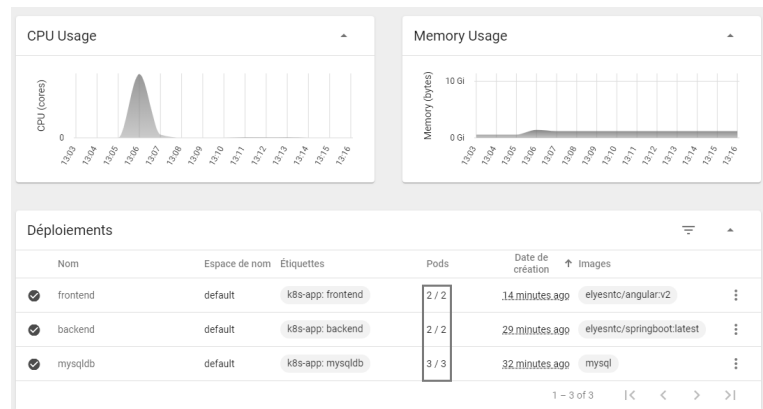
Deux répliques pour les services frontend et backend :

Figure 6-52



Voici un aperçu des déploiements :

Figure 6-53



Vous pouvez aussi vérifier l'état des pods :

Figure 6-54

```
C:\Users\Elyes>kubectl get pods
NAME                                READY   STATUS
backend-7cfcff59d9-b6v8v            1/1     Running
backend-7cfcff59d9-gdr99            1/1     Running
frontend-765b649f65-7rghm          1/1     Running
frontend-765b649f65-pnqft          1/1     Running
mysqlpdb-8469cdc9f8-p8jxn          1/1     Running
mysqlpdb-8469cdc9f8-pwvff          1/1     Running
mysqlpdb-8469cdc9f8-rx5qz          1/1     Running
```

Tous les pods s'exécutent sous Kubernetes sur un seul nœud, qui lui-même s'exécute sur notre machine hôte. Ne spécifiez donc pas un nombre important de répliques car elles consomment énormément de ressources CPU et RAM.

Sur un cluster, Kubernetes est capable à tout moment de stopper un pod et de le relancer sur un autre nœud pour des raisons de répartition de charge ou de crash de pod. Il est donc primordial de pouvoir conserver les données de la base de données.

## Conclusion

Une architecture en microservices se compose de services individuels qui possèdent chacun un rôle spécifique. L'un des nombreux avantages de cette architecture est qu'elle permet la suppression et l'ajout d'un service ou de le modifier tout en perturbant peu l'ensemble de l'application.

Cette application web est extensible, vous pouvez coder les autres boutons et ajouter d'autres entités ce qui étale la base de données. Le principe reste le même si vous décidez d'utiliser d'autres technologies frontend ou backend.

### Validation des acquis

#### Questions :

- 1 Pourquoi passer d'une architecture monolithique à une architecture en microservices ?
- 2 Quelles différences y a-t-il entre backend, frontend et full stack ?
- 3 Quels sont les avantages de construire un backend REST avec Spring Boot ?
- 4 Comment développer une application avec Spring Boot à l'aide d'annotations ?
- 5 Pourquoi utiliser un framework de développement frontend et pas simplement du JavaScript ?

#### Réponses :

- 1 **Pourquoi passer d'une architecture monolithique à une architecture en microservices ?**

Le principal problème des applications monolithiques est qu'elles centralisent tous les besoins et qu'elles sont réalisées dans une seule technologie. Au fil du temps, ces applications deviennent de plus en plus complexes et il est alors difficile de garder l'architecture modulaire prévue à l'origine.

Ces inconvénients nous ont conduits à penser autrement, à exploser le bloc applicatif en services indépendants, écrits dans différents langages et maintenus par des équipes différentes. L'application devient alors une suite de services. Ces services autonomes sont amenés à communiquer entre eux par le biais d'un protocole simple tel que HTTP et une API REST. La complexité de l'application est alors décomposée en un ensemble de modules facile à maintenir (Dubreuil, 2014).

- 2 **Quelles différences y a-t-il entre backend, frontend et full stack ?**

Chacun de ces concepts englobe un groupe de plusieurs langages de programmation. Les développeurs frontend et backend sont deux profils qui doivent travailler en étroite collaboration, laquelle est indispensable pour mener au mieux un projet web.

Lorsque l'on parle de frontend, il s'agit des éléments du site que l'on voit à l'écran et avec lesquels on peut interagir. Ces éléments sont composés de HTML, CSS et de JavaScript, contrôlés par le navigateur web de l'utilisateur.

Backend en revanche signifie « en arrière-plan » et correspond à toute la partie que ne voit pas visuellement le client. Cette partie, c'est ce qui permet au site de fonctionner et d'interpréter toutes les actions de l'utilisateur sur l'interface.

Voici des exemples de langages de programmation que l'on peut qualifier de backend :

- PHP ;
- Ruby ;
- Python ;
- Java.

Enfin, un développeur full stack est un développeur qui maîtrise les deux facettes frontend et backend.

### ③ Quels sont les avantages de construire un backend REST avec Spring Boot ?

Spring Boot nous apporte toute la puissance du framework Spring bien connu des développeurs Java, ainsi qu'un Tomcat embarqué et une documentation de référence expliquant clairement comment déployer une telle application dans le cloud.

Spring Boot est très simple à utiliser. Tout ce que vous devez avoir est un JDK et Maven ou Gradle. Une base de données peut également être utile pour stocker des données.

Dans Spring Boot, tout est configuré automatiquement. Nous avons juste besoin d'utiliser une configuration appropriée pour utiliser une fonctionnalité particulière. Spring Boot est très utile si nous voulons développer une API REST.

### ④ Comment développer une application avec Spring Boot à l'aide d'annotations ?

Le langage de programmation Java a pris en charge les annotations à partir de Java 5.0. Les principaux frameworks Java ont rapidement adopté les annotations. Les annotations Java permettent de simplifier grandement les fichiers de configuration de Spring.

Avant les annotations, le comportement de Spring était largement contrôlé via la configuration XML.

Spring Boot est un framework basé sur des microservices, ce qui rend une application prête pour la production en très peu de temps. Il est en fait basé sur toutes les fonctionnalités par défaut du Spring.

Exemples d'annotations de base du framework Spring :

- `@Required` : indique que le bean affecté doit être rempli au moment de la configuration avec la propriété requise. Sinon, une exception de type `BeanInitializationException` est levée.
- `@Autowired` : permet d'activer l'injection automatique de dépendance.

Exemples d'annotations de stéréotype Spring :

- `@Component` : marque une classe Java comme étant un bean.
- `@Controller` : marque une classe en tant que gestionnaire de requêtes web. Elle est souvent utilisée pour servir des pages web. Elle est principalement utilisée avec `@RequestMapping` annotation.

- `@Service` : utilisée au niveau de la classe, elle indique au Spring que la classe contient la logique métier.
- `@Repository` : utilisée sur les classes Java qui accèdent directement à la base de données.

Exemples d'annotations Spring Boot :

- `@EnableAutoConfiguration` : indique à Spring Boot de commencer à ajouter des beans en fonction des paramètres de chemin de classe, d'autres beans et de divers paramètres de propriété.
- `@SpringBootApplication` : combinaison de trois annotations `@EnableAutoConfiguration`, `@ComponentScan` et `@Configuration`.

Exemples d'annotations Spring MVC et REST :

- `@RequestMapping` : utilisée pour mapper les requêtes web.
- `@GetMapping` : gère la méthode GET.
- `@PostMapping` : gère la méthode POST.
- `@PutMapping` : mappe la méthode HTTP PUT.
- `@DeleteMapping` : gère la méthode HTTP DELETE.

### 5 Pourquoi utiliser un framework de développement frontend et pas simplement du JavaScript ?

Revenons d'abord sur quelques définitions :

- Une bibliothèque est une collection de fonctionnalités écrites dans un langage pour permettre la réutilisation de code déjà écrit par d'autres développeurs.
- Un framework est une collection de plusieurs bibliothèques, une architecture minimale de projet et un ensemble de scripts simplifiant le développement.

Les frameworks frontaux apportent donc une base pour construire tout en permettant la flexibilité avec le design final. Ils permettent d'accélérer le flux de travail et d'augmenter la productivité sans sacrifier la qualité ou la fonctionnalité.

Exemples de frameworks frontaux :

- React : bibliothèque JavaScript déclarative, efficace et flexible pour la création d'interfaces utilisateur. Ce framework permet de créer des interfaces utilisateur complexes à partir de petits morceaux de code isolés appelés « composants ». React est donc idéal pour des composants individuels, mais pour construire une nouvelle application, Angular sera beaucoup plus adapté.
- Angular : framework open source écrit en JavaScript qui permet la création d'applications web. Il est basé sur une architecture du type MVC et permet donc de séparer les données, le visuel et les actions pour une meilleure gestion des responsabilités. Il s'agit d'un type d'architecture qui a largement fait ses preuves et qui permet une forte maintenabilité et une amélioration du travail collaboratif.
- Vue : cadre progressif pour la création d'interfaces utilisateur. Contrairement à d'autres frameworks monolithiques, Vue est conçu dès le départ pour être progressivement adoptable.

# À vous de jouer

---

Comme nous l'avons vu dans cet ouvrage, Docker offre un meilleur moyen de créer et de distribuer vos applications.

Il propose également son propre outil d'orchestration, Docker Swarm, qui est généralement meilleur en termes de mise à l'échelle à cause de la complexité de Kubernetes qui entraîne une certaine lourdeur. Cependant, les mises à l'échelle automatiques sont plus efficaces avec Kubernetes grâce à ce système complexe. Par ailleurs, Kubernetes présente l'avantage considérable de pouvoir surveiller l'état des conteneurs à tout moment et de compenser directement une panne. Il veille à ce que tout fonctionne comme souhaité et est capable de fournir un nœud de remplacement en cas de défaillance, ce qui fournit une haute disponibilité.

La mise en place d'une infrastructure et de clusters Kubernetes fiables et de qualité ne se limite pas au provisionnement d'un cluster et au déploiement d'applications sur celui-ci. C'est un parcours continu qui combine la planification, la conception ainsi que la mise en œuvre, le CI/CD de l'infrastructure et des services et les opérations et la maintenance.

Concernant la sécurité, les pipelines de construction peuvent être compromis, les images de conteneurs peuvent être modifiées, les conteneurs peuvent exécuter des logiciels vulnérables en tant qu'utilisateurs privilégiés, et les attaquants ayant accès à l'API Kubernetes pourraient même prendre le contrôle de votre cluster. Vous ne saurez pas que votre application est sûre à 100 % et vous ne pourrez pas confirmer qu'aucune faille de sécurité ne s'est produite pendant son fonctionnement. Pour atteindre cet objectif, il faut appliquer une sécurité approfondie à l'ensemble de votre chaîne logistique logicielle. Je n'ai pas abordé cette partie sur la sécurité au sein de Kubernetes dans ce livre.

Je ne peux pas non plus aborder toutes les plates-formes d'infrastructure de cloud public. J'ai donc décidé d'opter pour un choix courant pour le déploiement de Kubernetes, en utilisant Azure. Vous pouvez toujours utiliser un autre fournisseur de cloud, un cloud privé, ou même une installation sur site. La plupart des concepts et des pratiques abordés dans ce livre seront toujours applicables.



# Références

---

*Kubernetes in Production Best Practices* de Aly Saleh et Murat Karslioglu, Packt Publishing Ltd, 2021.

*Control startup and shutdown order in Compose*, Docker Inc. 2013-2021. Disponible sur Docker docs à l'adresse : <https://docs.docker.com/compose/startup-order/>.

*Build and Ship any Application Anywhere*, Docker Inc. 2021. Disponible sur Docker Hub à l'adresse : <https://hub.docker.com/>.

*De l'application monolithique aux architectures microservices* de Julien Dubreuil, 2014. Disponible à l'adresse : <https://juliendubreuil.fr/blog/developpement/de-application-monolithique-aux-architectures-microservices-ou-orientees-composants/>.

*Créer un cluster Kubernetes*, Microsoft, 2021. Disponible sur Microsoft Azure (préversion) à l'adresse : <https://preview.portal.azure.com/?azure-portal=true#create/microsoft.aks>.

*Protégez votre entreprise avec une plateforme d'identités universelle*, Microsoft, 2021. Disponible sur Microsoft Azure à l'adresse : <https://azure.microsoft.com/fr-fr/services/active-directory/>.

*Windows et conteneurs*, Microsoft, 2021. Disponible à l'adresse : <https://docs.microsoft.com/fr-fr/virtualization/windowscontainers/about/>.

*Zones géographiques Azure*, Microsoft, 2021. Disponible sur Microsoft Azure à l'adresse : <https://azure.microsoft.com/fr-fr/global-infrastructure/geographies/#choose-your-region>.

*Docker : outil et sujet d'enseignement en D.U.T.* de Jean-Marc Pouchoulon (s.d.), *Resinfo*, 2016.

*Learn Kubernetes in a Month of Lunches* d'Elton Stoneman, Manning Publications Co, 2021.

*L'API Kubernetes* The Linux Foundation, 2020. Disponible à l'adresse : <https://kubernetes.io/docs/concepts/overview/kubernetes-api/#:~:text=The%20Kubernetes%20API%20lets%20you%20query%20and%20manipulate,as%20kubeadm,%20which%20in%20turn%20use%20the%20API.>

*Secrets* The Linux Foundation, 2021. Disponible à l'adresse : <https://kubernetes.io/docs/concepts/configuration/secret/>.

*Qu'est-ce qu'un hyperviseur (hypervisor) ?*, VMware Inc., 2021. Disponible à l'adresse : <https://www.vmware.com/fr/topics/glossary/content/hypervisor.html>.





# Index

---

## A

- ACI
  - Azure Container Instances 49
- ADD 47
- AKS
  - Azure Kubernetes Service 128, 130, 131
- Angular 149
- Ansible 123
- API 130
- API REST 148
- AWS
  - Amazon Web Services 121
- AWS (Amazon Web Services) 9
- Azure AD
  - Azure Active Directory 129
- Azure Monitor 130
- Azure Policy 130

## B

- backend 147
- Bitbucket 9

## C

- Calico 96
- Chef 123
- CLI
  - Interface de ligne de commande 36, 39
- ClusterIP 111, 121
- CMD 46
- ConfigMap 119
- containerd 93, 94
- COPY 47
- CoreDNS 97
- CRI (Container Runtime Interface) 93
- cri-o 93
- CronJob 119
- crontab 119
- CRUD (Create, Read, Update, Delete) 128, 164
- CSI (Container Storage Interface) 96
- CSS 184

## D

- DaemonSet 119
- dashboard 103
- déploiement 107
- DevOps 2
- Docker
  - conteneur 15, 18
  - installation 11
- Docker Hub 16, 19, 95
- docker service 72
- Docker Stack 81
- Docker Swarm 90
- docker-ce 12
- Dockerfile 42
- DockerHub 9
- driver
  - bridge 22
  - host 25
  - macvlan 25
  - none 25
  - overlay 25

## E

- ENTRYPOINT 46
- ETCD 92
- ExternalName 112, 121

## F

- Flannel 96
- frontend 147
- full stack 148, 184

## G

- GitHub 3, 9

## H

- Helm 123

## I

- IDE
  - Environnement de développement intégré 149

IntelliJ IDEA 148

## J

JAR

Java Archive 169

JavaScript 184

JDK 185

job 119

## K

k8s 1, 96

kube-apiserver 92

kube-controller-manager 93

kubect1 97

Kubelet 94

kube-proxy 94

Kubernetes 89, 95

dashboard 103

service DNS 97, 99

kube-scheduler 92

Kustomize 123

## L

LABEL 47

LoadBalancer 112, 121, 122

## M

MAINTAINER 47

Maven 149

Minikube 98, 120, 121

MVC

Modèle-vue-contrôleur 186

## N

NAT (Network Address Translation) 111

Nginx 18

NodePort 111

nœud 69

nslookup 28

## O

OAuth 120

OCI (Open Container Initiative) 93

openjdk 169

## P

POM 151

Puppet 123

PV

Volume persistent 94

PWD (Play With Docker) 30

## R

RBAC 128

React 186

ReplicaSet 104, 107

réseau

bridge 27

rkt 93

rktlet 93

runC 93

## S

service 111

Spring Boot 149

ssh 120

stockage persistent 94

Swarm 69

cluster 70

## T

Terraform 123

## V

Vue 186

## W

WeaveNet 96

worker 69

## X

XAMPP 148