

Chapitre 3

Premiers pas avec Docker

1. Hello World, Docker

1.1 Démarrage d'un conteneur simple

Le sujet de l'installation de Docker étant désormais traité, il est temps de commencer enfin à explorer l'outil lui-même. Comme toute technologie informatique qui se respecte, Docker dispose d'un exemple "Hello World", soit dans notre cas un conteneur qui ne fait rien à part afficher un message de bienvenue. L'intérêt de ce genre d'approche est qu'il permet de valider que l'installation du produit s'est bien passée, de vérifier que l'ensemble de la chaîne logicielle fonctionne et de mettre le pied à l'étrier pour l'utilisateur débutant.

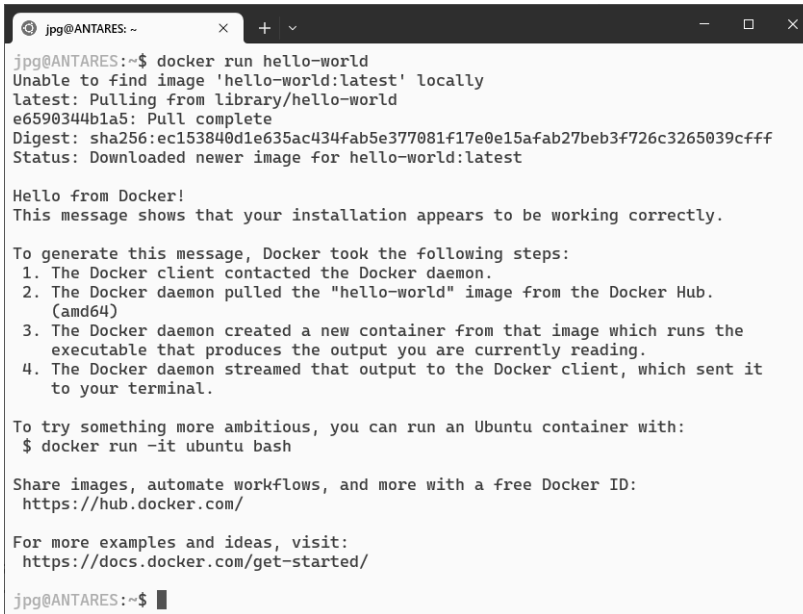
Démarrer un conteneur exemple minimaliste

```
docker run hello-world
```

■ Remarque

Dorénavant et pour la suite du livre, nous considérons que la gestion des droits est traitée. Ainsi, s'il est nécessaire de préfixer les commandes par le mot-clé `sudo` sous Linux, nous ne le montrerons plus dans les exemples. L'obtention d'un message comportant le texte "permission denied" ou une expression approchante doit donner le réflexe de rajouter le mot-clé si nécessaire ou de traiter le problème de sécurité. Dans le même souci de simplification, les exemples sont montrés sur Linux.

Sauf problème d'installation de Docker ou d'accès à Internet, l'affichage devrait être le suivant :



```
jpg@ANTARES: ~  
jpg@ANTARES:~$ docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: Pulling from library/hello-world  
e6590344b1a5: Pull complete  
Digest: sha256:ec153840d1e635ac434fab5e377081f17e0e15afab27beb3f726c3265039cfff  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
   (amd64)  
3. The Docker daemon created a new container from that image which runs the  
   executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
   to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/  
jpg@ANTARES:~$
```

1.2 Détails des opérations effectuées

Bien que l'exécution de la commande ci-dessus soit très rapide, il s'est passé un certain nombre d'opérations pour arriver à ce résultat. C'est d'ailleurs la raison d'être de Docker que de rendre disponible en une seule ligne de commande un enchaînement d'opérations dont nous allons voir en les détaillant ci-dessous qu'elles sont très variées.

1.2.1 Récupération d'une image

Tout d'abord, Docker a besoin de lire l'image nommée `hello-world` qui lui a été passée en paramètre de la commande `run`.

Comme nous l'avions rapidement abordé dans l'introduction, Docker dispose d'un dépôt d'images auxquelles il peut accéder par Internet. Ce dépôt, accessible par l'URL <https://registry.hub.docker.com>, regroupe toutes les images "officielles" fournies par Docker. Comme il s'agit du registre d'images Docker utilisé par défaut, il est souvent désigné simplement comme "le registre", mais nous verrons un peu plus loin qu'il est possible de créer d'autres registres.

■ Remarque

Le registre par défaut est évidemment public, de façon que la commande ci-dessus soit la plus simple possible et n'ait pas à être alourdie par des informations d'authentification. D'autres registres peuvent être privés et réservés à telle ou telle organisation, voire hybrider ce mode d'accessibilité avec certaines ressources publiques et d'autres protégées.

Constatant que l'image n'était pas déjà présente sur la machine hôte, Docker l'a téléchargée. Il est toutefois possible de récupérer l'image au préalable.

Télécharger une image sans la lancer

■ `docker pull hello-world`

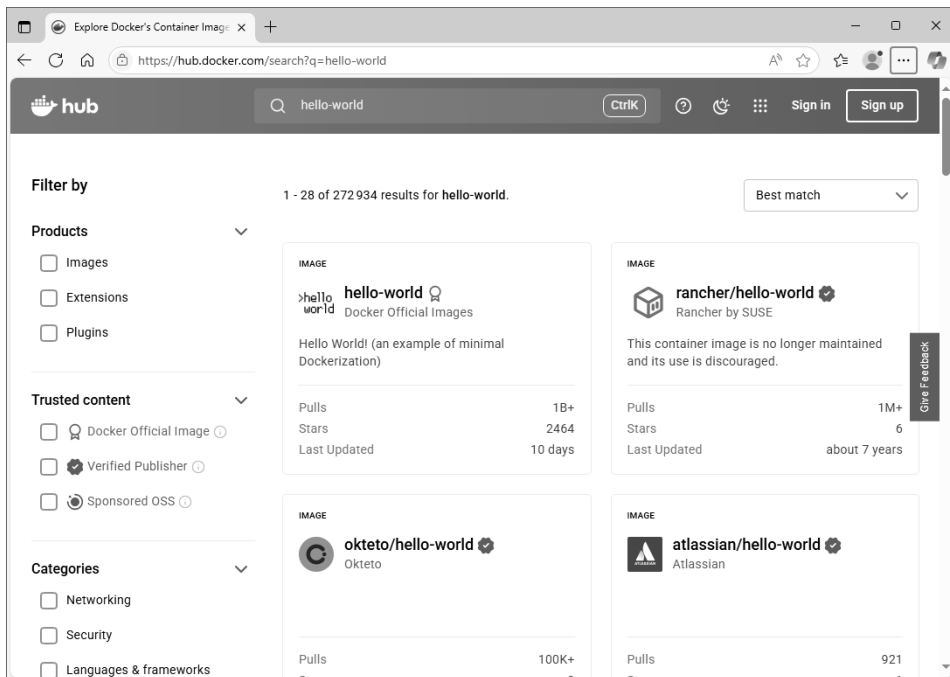
Dans le cas de l'image `hello-world`, qui pèse moins d'1 ko, la différence de temps est infime, mais pour des images plus volumineuses, il peut être particulièrement utile de séparer le téléchargement et le lancement. Dans tous les cas, l'image ne sera chargée qu'une seule fois. En effet, Docker gère un cache des images sur la machine, et le prochain démarrage d'une instance de conteneur sur la même image n'aboutira donc pas nécessairement au rechargement de celle-ci, sauf bien sûr si une instruction est fournie pour que Docker aille vérifier que la nouvelle version n'a pas changé.

■ Remarque

Une bonne pratique est de gérer des versions immuables des images Docker, c'est-à-dire que si une image a été publiée avec une version donnée et que des modifications ont été réalisées, il faut que la version augmente, de façon à signifier aux consommateurs cette modification. Dans le cas contraire, en ce qui concerne Docker, le client considérera qu'il possède déjà la version dans son cache et n'ira pas vérifier si la version sur le registre est différente, à moins qu'on lui demande explicitement de la recharger.

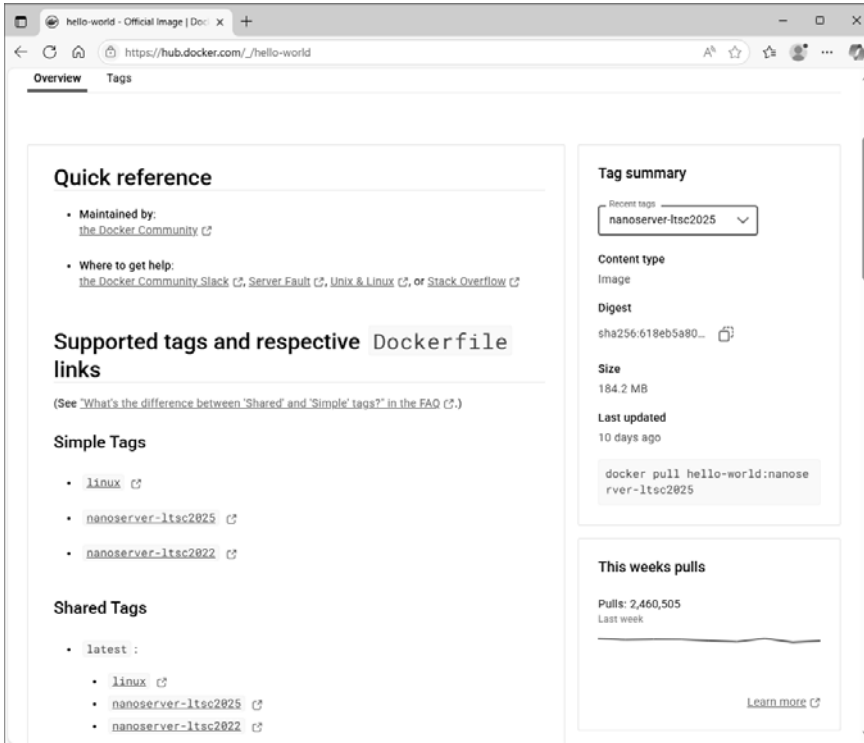
1.2.2 Identité de l'image

Une recherche sur le registre Docker permet de localiser la définition de l'image :



Dès l'affichage lié à la recherche, on constate que les images peuvent supporter plusieurs architectures informatiques et qu'`hello-world` en supporte justement un grand nombre. Il se trouve que cette image est disponible sur des plateformes Linux et Windows, mais aussi sur des systèmes MIPS 64, ARM, PowerPC, et même pour des mainframes IBM Z. Dans la ligne de commande lancée plus haut, la plateforme n'a pas eu besoin d'être précisée, car le démon Docker est capable, de lui-même, de négocier la bonne image avec le registre, et une erreur est remontée si l'image ne supporte pas la plateforme installée.

Un clic sur la première entrée de la liste ci-dessus, qui correspond à l'image cherchée, montre plus de détails sur cette image. Le premier onglet d'information, nommé **Overview**, expose en particulier ce qui s'appelle les tags, ou étiquettes en français.



Bien qu'il y ait nécessairement un lien entre les différentes images et les différentes plateformes, une image étiquetée Linux étant prévue pour une plateforme Linux, il s'agit bien de deux notions séparées. Par exemple, il est habituel que des images pensées pour la plateforme Linux soient pour certaines basées sur une version de Linux très complète comme une Ubuntu et pour certaines sur une image réduite comme une Alpine. La plateforme est alors toujours Linux, mais les étiquettes permettront de faire la différence entre les deux images.

Lorsque l'étiquette pointe sur une seule image et que seul le choix de la plateforme sous-jacente reste à réaliser par le moteur Docker lors du téléchargement, on parle d'étiquette simple (**Simple Tags** dans l'interface en anglais), mais pour complexifier la compréhension – bien que cela simplifie au final les manipulations –, il existe des étiquettes partagées (**Shared Tags**) dont l'usage fait que le moteur Docker choisit non seulement la plateforme d'exécution, mais aussi négocie l'image même qui va être déployée. Dans l'écran plus haut, on voit que l'étiquette `latest` est dite partagée, car son usage fera que le moteur Docker choisit entre deux images, à savoir l'image prévue pour Linux et l'image prévue pour Windows Nano Server. Bien sûr, il se trouve que la première fonctionnera sur des plateformes Linux et la seconde sur des plateformes Windows, mais les deux notions restent bien différentes.

Comme expliqué plus haut, on peut avoir des images Linux très différentes en nombre de fonctionnalités (et donc en taille) et il en va de même pour Windows où, au lieu d'un Nano Server, une image pourrait se baser sur une version Windows Server complète, avec une image bien plus lourde en conséquence.

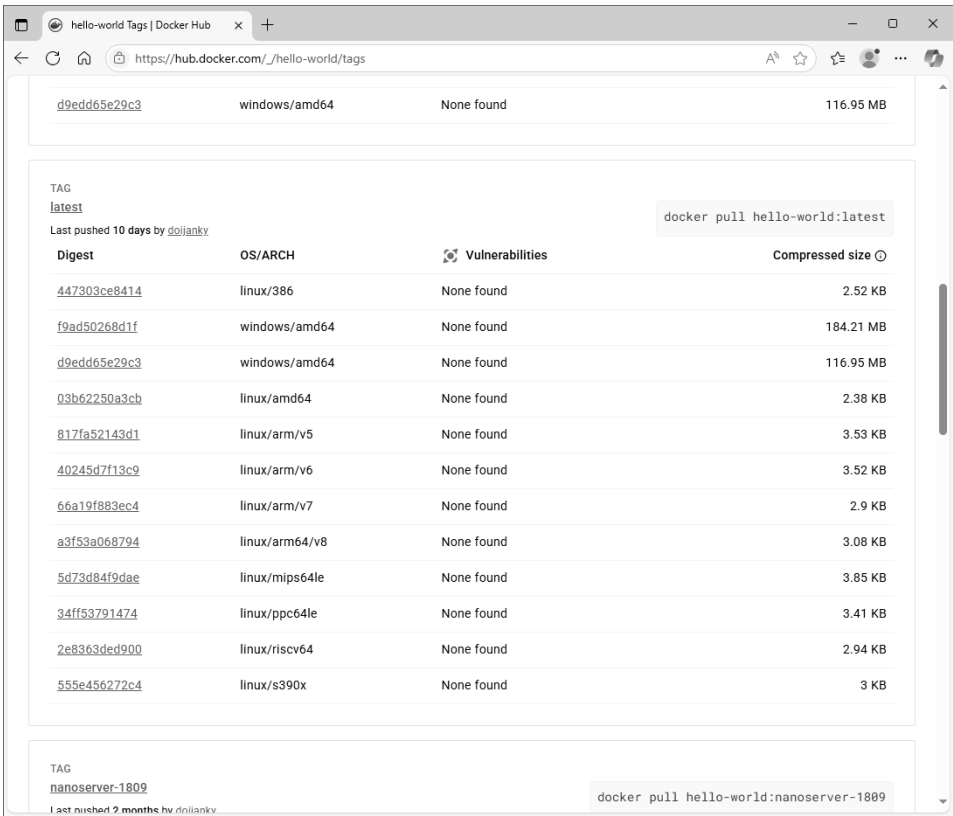
Dans notre cas, la commande ayant été lancée sans préciser d'étiquette, c'est l'étiquette `latest` qui est appelée par défaut (comportement standard de Docker dans son échange avec les registres). Cette étiquette donne le choix entre une image `nano-server-ltsc2022` ou une image `nanoserver-ltsc2025`, qui supportent la plateforme Windows dans ses deux déclinaisons Long Term Support, et une image `linux`, qui supporte toutes les autres plateformes.

■ Remarque

Les images étant versionnées, comme expliqué plus haut, nous reviendrons sur cette étiquette `latest` bien particulière, qui permet également de choisir une version de référence (et non la plus récente, comme la traduction française de `latest` pourrait le faire croire).

1.2.3 Taille des images

En accédant à l'onglet **Tags** de l'interface, on obtient une description plus précise des étiquettes, qui devrait permettre d'aider à la compréhension de cette notion qui n'est pas évidente en première utilisation :



On voit tout d'abord qu'il existe en fait une dizaine d'images simples, pour autant d'architectures supportées, et avec des tailles extrêmement différentes. Celles qui supportent la plateforme `windows/amd64` pèsent plus de 100 Mo. Les autres images de la liste supportent huit plateformes matérielles et pèsent quelques Ko chacune, avec des variations très légères.

Remarque

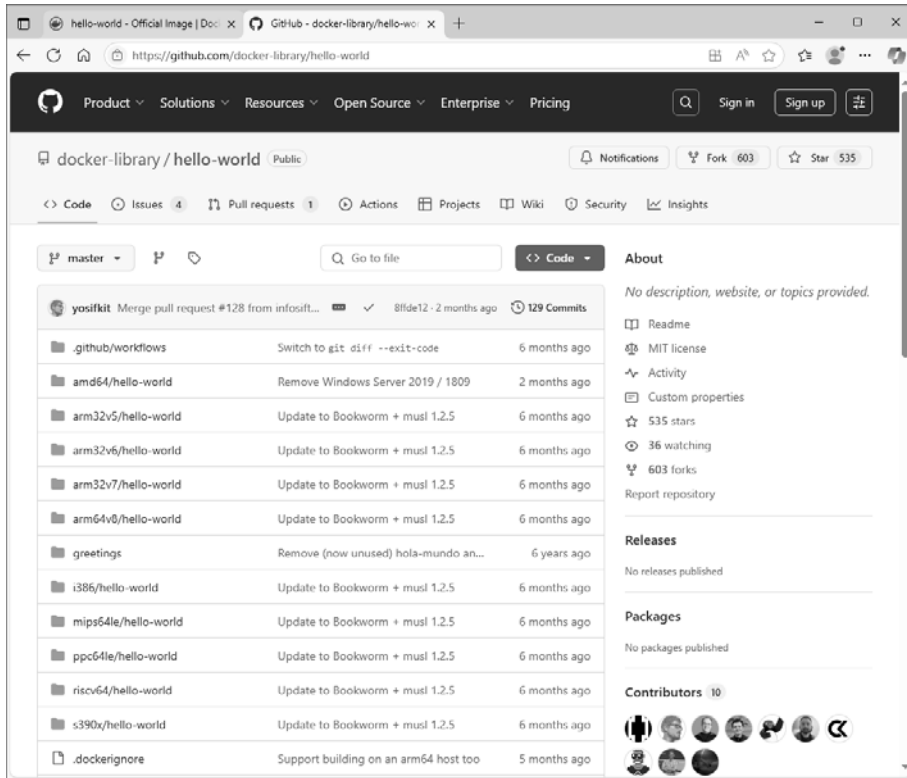
Cette énorme différence de poids s'explique par deux raisons. La première est que Docker est une technologie pensée pour Linux et à laquelle Windows s'adapte sans avoir les caractéristiques prévues spécifiquement pour dans son architecture. Une image Windows, au lieu de s'appuyer simplement sur le système sous-jacent, doit embarquer un mini-système Windows pour être réellement autonome. Dans ces conditions, passer sous la barre des 100 Mo était déjà un exploit (réalisé uniquement avec les anciennes versions, les nouvelles dépassant cette valeur comme on peut le voir) et rapproche ces images des poids standards d'images Docker pour Linux avec des serveurs complexes, ou embarquant des systèmes plus complets comme une image Ubuntu. Il y a toutefois une seconde part d'explication à la différence de taille : pour cette image particulière, les concepteurs sont partis de zéro et n'incluent dans l'image qu'un exécutable compilé nativement dans chacun des systèmes ciblés, ce qui fait naturellement que le poids est réduit à l'extrême, seul le binaire exécutable étant alors nécessaire. Nous verrons dans la section suivante comment ceci est mis en œuvre.

L'interface plus haut est également l'occasion de revenir sur la notion d'étiquette partagée et de montrer justement l'étiquette `latest`, tout en haut de la liste, et qui couvre quant à elle dix plateformes, à savoir les huit de l'image `linux` et les deux de la plateforme supplémentaire `nanoserver`. On comprend ainsi mieux l'effet "chapeau" de ces étiquettes partagées, qui permettent à l'utilisateur de ne pas se poser de questions et de demander à Docker de se débrouiller pour récupérer une image qui fonctionne sur son système et sa plateforme. Dans le cas de `latest`, le principe est aussi de laisser Docker s'occuper de choisir la bonne version, le cas échéant. Il se trouve que pour l'image `hello-world`, il n'y a qu'une version, donc ceci n'est pas visible. Mais pour des images de serveurs logiciels, nous reviendrons plus loin sur l'association entre `latest` et la dernière version stable de l'application, qui est une convention dans l'écosystème Docker.

On remarque aussi que les tailles sont exactement les mêmes entre les différentes plateformes des étiquettes simples et de l'étiquette partagée, les étiquettes n'étant en fait que des pointeurs sur les images au sens binaire, qui possèdent un contenu et donc un poids associé.

1.2.4 Anatomie de l'image Linux

L'onglet **Description** de l'interface Docker Hub donne une autre information intéressante pour comprendre le fonctionnement de cette image `hello-world`, à savoir le lien vers le dépôt GitHub du code source de génération de cette image, en l'occurrence <https://github.com/docker-library/hello-world> :



Comme indiqué plus haut, nous allons revenir dans cette section sur la raison du poids extrêmement réduit de l'image `hello-world` dans son étiquette `linux`. Contrairement à de nombreuses images Docker qui contiennent non seulement une application serveur mais aussi les dépendances nécessaires en termes de système Linux, cette image part d'une base complètement vide, et n'ajoute qu'un binaire exécutable sans aucune autre dépendance sur la plateforme ciblée.

Chapitre 3

Tableau de bord et ligne de commande

1. Objectifs du chapitre et prérequis

Ce chapitre fera tout d'abord un petit historique sur l'origine du projet et sur les notions autour des conteneurs.

La suite sera consacrée à une introduction sur l'utilisation de Kubernetes et la découverte de quelques éléments :

- le tableau de bord (dashboard) Kubernetes ;
- la commande `kubectl` ;
- le moteur Docker de Minikube ;
- la consultation de quelques objets de base (pods et nodes).

À la fin de ce chapitre, vous disposerez des éléments suivants :

- un cluster Kubernetes fonctionnel ;
- un contexte `kubectl` cohérent.

Afin de suivre l'ensemble des exercices, il est indispensable de disposer d'une connexion Internet haut débit (ADSL ou supérieure).

2. Préambule

2.1 Origine du nom et du logo

Kubernetes est un mot d'origine grec (κυβερνήτης) qui signifie timonier (ou pilote). En navigation, c'est la personne qui est en charge de tenir la barre du gouvernail.

Chez Google, le nom interne de ce projet était Project Seven, en référence à un personnage de l'univers Star Trek (Seven of Nine ou de son vrai nom Annika Hansen). Les sept rayons du logo en forme de barre de gouvernail sont un clin d'œil à cet ancien nom.

2.2 Pourquoi utiliser Kubernetes ?

La sortie de Docker ces dernières années a constitué une rupture importante dans la manière de gérer les applications au sein d'un système d'information. Il devenait ainsi possible de réaliser des opérations auparavant compliquées de façon plus rapide et simple, comme par exemple :

- démarrer des environnements de tests à la demande ;
- s'affranchir des problèmes de dépendances de librairies ;
- uniformiser les livrables de tous les environnements ;
- isoler de manière très poussée les ressources sur un même serveur.

Comme tout nouvel outil, certaines questions n'avaient pas forcément de solution simple et introduisaient de nouveaux problèmes. Parmi ces problèmes, on retrouve les points suivants :

- persistance des données ;
- surveillance des applications dans les conteneurs ;
- mise à jour automatique des applications ;
- scalabilité du moteur Docker ;

- scalabilité de la charge applicative ;
- publication des environnements sur l'extérieur.

C'est entre autres pour ces raisons que Kubernetes a été introduit en 2014 afin de répondre aux problèmes de passage à l'échelle de Docker.

2.3 Origine de Kubernetes

Les premières versions de Kubernetes proviennent de Google. La société derrière le moteur de recherche travaille depuis plus de dix ans sur les notions de conteneurisation. Ces travaux ont amené Google à reverser à la communauté de très nombreuses contributions sur le noyau Linux (Google fait partie des cinq plus gros contributeurs).

Une partie de ces travaux a d'ailleurs aidé à l'émergence de Docker, comme par exemple les Cgroups. Ce mécanisme – à l'origine développé par des ingénieurs de chez Google – permet de mettre en place une limitation de la consommation CPU et/ou mémoire et également de l'isolation de process.

Chez Google, ces ancêtres des conteneurs contemporains étaient pilotés en interne par un produit maison : Borg. Ce produit est toujours en activité chez Google et gère quotidiennement des milliers de machines dans le monde.

À noter qu'une partie des développeurs de Kubernetes sont d'anciens contributeurs sur le projet Borg. Ces derniers ont repris une grande partie des principes qui étaient présents dans Borg pour les appliquer dans Kubernetes.

2.4 Fondation CNCF

Le code du produit a été donné par Google à la Linux Foundation, ce qui a servi à la création de la CNCF (*Cloud Native Computing Foundation*). Cette organisation est derrière la promotion des produits open source dans le cloud comme Prometheus, OpenTracing ou Containerd.

L'idée de Kubernetes est de garder ce qui fonctionne bien dans Docker (le principe des conteneurs Dockers) et d'étendre le reste avec une couche d'API ainsi qu'un mécanisme d'extension et d'orchestration.

2.5 Les orchestrateurs du marché

La société à l'origine de Docker a développé son propre moteur d'orchestration : Docker Swarm. Ce livre n'en parlera pas, mais ce produit répond lui aussi à certains problèmes évoqués plus haut. À noter que les dernières évolutions de cet outil permettent de communiquer avec Kubernetes.

En plus de Docker Swarm, il existe d'autres orchestrateurs comme par exemple Nomad ou Mesos qui utilisent des mécanismes totalement différents de ceux de Kubernetes.

■ Remarque

L'intégration Docker Shim qui permet de faire dialoguer Kubernetes avec Docker est difficile à maintenir alors qu'il est possible de communiquer directement avec le moteur de conteneur sous-jacent (Containerd). La dépréciation vient directement de cette raison.

Il existe également des surcouches à Kubernetes apportant des fonctionnalités ou un support commercial. On retrouve notamment les produits suivants :

- Openshift chez RedHat ;
- Tectonic chez CoreOS ;
- Rancher chez Rancher Labs.

Ces produits sont tous basés sur une version de Kubernetes sur lequel l'éditeur va apporter les services suivants :

- préconfigurer des services prêts à l'emploi ;
- figer un ensemble de versions afin d'offrir un support longue durée.

Heureusement, ces différences sont généralement marginales. Elles ne devraient pas vous poser de problème vis-à-vis des instructions qui se trouvent dans ce livre.

3. Le tableau de bord de Kubernetes (dashboard)

3.1 Présentation

Le dashboard de Kubernetes est un moyen de consulter l'état des différents éléments d'un cluster. Pour l'utilisateur débutant sur cette technologie, il s'agit d'une bonne porte d'entrée puisque cette application permet de consulter graphiquement les différents constituants d'un cluster Kubernetes.

Néanmoins, si ce dernier peut être toléré sur une machine Minikube, il n'est pas forcément souhaitable de le déployer sur les plateformes de production pour des raisons de sécurité.

3.2 Tableau de bord Kubernetes sur service managé

Dans le cas où vous feriez appel à un service managé, il sera peut-être nécessaire de procéder à l'installation du dashboard avant de continuer.

Consultez la section Déploiement du tableau de bord Kubernetes dans les annexes afin de réaliser l'installation de ce dernier.

3.3 Déploiement du dashboard sur Minikube

Sous Minikube, l'activation du dashboard se fait à l'aide de l'instruction `minikube` suivie des options suivantes :

- l'option `addons` suivie du mot-clé `enable` ;
- le nom du plugin à activer (ici `dashboard`).

Ci-dessous la commande à lancer :

```
■ $ minikube addons enable dashboard
```

Ci-après le résultat de cette commande :

```
■ Using image kubernetesui/dashboard:v2.3.1
■ Using image kubernetesui/metrics-scraper:v1.0.7
💡 Some dashboard features require the metrics-server addon. To
enable all features please run:
    minikube addons enable metrics-server

★ The 'dashboard' addon is enabled
```

Comme indiqué par la commande précédente, activez le serveur de métriques (metrics-server) pour bénéficier de l'ensemble des fonctionnalités du tableau de bord :

```
■ $ minikube addons enable metrics-server
```

Ci-dessous les messages renvoyés par cette opération :

```
■ Using image k8s.gcr.io/metrics-server/metrics-server:v0.4.2
★ The 'metrics-server' addon is enabled
```

3.4 Accès au dashboard sur Minikube

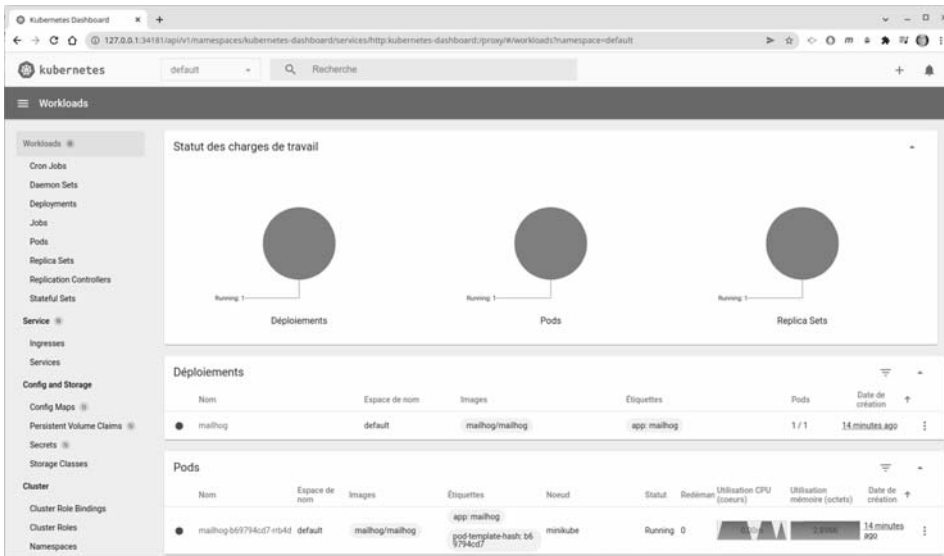
L'accès au tableau de bord peut se faire avec la commande `minikube` suivie de l'option `dashboard`.

Ci-dessous la commande correspondante :

```
■ $ minikube dashboard
```

Tableau de bord et ligne de commande _____ 91

Chapitre 3



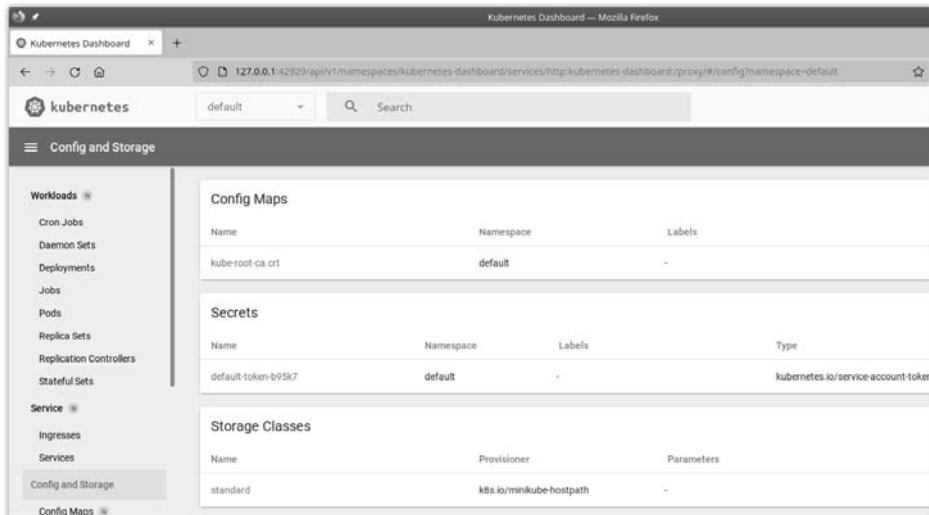
Page d'accueil du tableau de bord de Kubernetes

La commande s'occupera alors de publier le dashboard dans un tunnel sécurisé et de lancer le navigateur par défaut.

3.5 Structure du tableau de bord

La partie de gauche contient des liens vers les éléments suivants :

- les éléments déployés (**Workloads**) ;
- les entrées de service interne et externe (**Services, Ingresses**) ;
- les éléments de configuration (**Config and Storage**) ;
- les éléments du cluster (**Namespaces, Nodes, Persistent Volumes**, etc.) ;
- les définitions d'objets personnalisés (**Custom Resource Definitions**).



Vue d'ensemble de l'écran **Config and Storage**

Remarque

À noter la présence du secret `default-token`. Combiné avec le service `kubernetes` de l'espace de noms par défaut, ceux-ci permettront de référencer en interne le point d'entrée de l'API de Kubernetes avec un compte de service par défaut.

3.6 Création d'un déploiement

3.6.1 Un petit mot sur Mailpit

Depuis l'interface de déploiement, il est possible de déployer une première application. Pour les besoins du test, vous déploierez l'application Mailpit. Cette application embarque un serveur de mail SMTP. Une interface web est également présente et permet de consulter les mails reçus au travers du serveur SMTP.

Cette application sert à simuler l'envoi de mails. Du fait de sa simplicité et de sa légèreté, cette application fera une excellente candidate pour aborder les déploiements sous Kubernetes.

Sur Docker Hub, l'image de Mailpit porte le nom de `axllent/mailpit`.

■ Remarque

Mailpit trouve son inspiration originale dans un produit similaire : MailHog. Malheureusement, ce dernier n'est plus maintenu : il est préférable de ne plus y faire appel. Pour en apprendre davantage sur les avantages de Mailpit par rapport à MailHog, n'hésitez pas à consulter la page GitHub du projet : <https://github.com/axllent/mailpit>

3.6.2 Lancement du déploiement

Le dashboard permet le déploiement d'applications. Vous devez pour cela suivre le lien **Deployments** sous le titre **Workloads**.

Sur l'écran qui apparaît, cliquez sur le bouton **+** en haut à droite pour créer un nouvel objet. Sur cet écran, cliquez sur le lien **Create from form** puis entrez les informations suivantes :

- le nom du déploiement (**App name**) : mailpit ;
- le nom de l'image (**Container image**) : axllent/mailpit.

Les autres champs seront laissés aux valeurs par défaut.

Vous devez ensuite cliquer sur le bouton **Deploy** pour lancer le déploiement de l'application Mailpit.

The screenshot shows the 'Create from form' tab in the Kubernetes dashboard. The left sidebar lists 'Workloads' with sub-items: Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, and Stateful Sets. Under 'Service', there are Ingresses, Ingress Classes, and Services. Under 'Config and Storage', there are Config Maps. The main form area has three tabs: 'Create from input', 'Create from file', and 'Create from form' (selected). The form fields are: 'App name' (mailpit), 'Container image' (axllent/mailpit), 'Number of pods' (1), 'Service' (None), and 'Namespace' (default). Each field has a small 'Learn more' link. At the bottom, there are buttons for 'Deploy', 'Preview', 'Cancel', and 'Show advanced options'.

L'assistant de création d'applications dans Kubernetes avec les champs préremplis pour la création de l'application Mailpit