Chapitre 3
Le shell et les commandes GNU

1. Le shell bash

1.1 Rôle du shell

Même si toutes les distributions proposent des interfaces utilisateur graphiques, un informaticien professionnel travaillant sur un système Linux doit connaître le fonctionnement de l'interpréteur de commandes (shell) et des principales commandes en mode caractère. D'une part, les systèmes serveurs sont généralement installés sans interface graphique, d'autre part il est indispensable de pouvoir gérer les scripts d'exploitation et d'administration écrits en langage shell et combinant des commandes en mode caractère.

L'interpréteur de commandes permet d'exécuter des instructions saisies au clavier ou lues dans un fichier script. Cet interpréteur est le plus souvent un programme de type shell. Le terme shell (coquille), d'origine Unix, est employé en référence au terme **kernel** (noyau) : le shell est une interface « autour » du noyau Linux, fonctionnant en mode caractère.

Il existe plusieurs programmes de type shell, chacun disposant de spécificités propres. Le **Bourne Shell**, du nom de son créateur **Steve Bourne**, est le shell le plus ancien, écrit pour Unix. Le shell a ensuite été normalisé dans le cadre des normes POSIX.

Le shell de référence de la plupart des distributions Linux est le bash (*Bourne Again Shell*), mais il en existe de nombreux autres, dont :

- `sh` : Bourne Shell
- `ksh` : Korn Shell
- `csh` : C Shell
- `zsh` : Z Shell
- `ash` : A Shell
- `dash` : Debian Almquist Shell.

Remarque

Le fichier `/etc/shells` fournit la liste des shells installés sur le système.

1.2 Bash : le shell Linux par défaut

Le shell bash est un dérivé du Bourne Shell. Il est conforme aux normes POSIX mais il ajoute de nombreuses extensions qui lui sont spécifiques.

Remarque

Dans les distributions Debian récentes, le shell par défaut est le dash, une variante très proche du shell bash.

1.2.1 Un shell puissant et libre

Le bash, sous licence open source GNU, est fourni par défaut avec toutes les distributions Linux. Il existe même en version macOS et Windows (via la fonctionnalité Sous-système Windows pour Linux).

Le shell fonctionne en mode ligne de commande. Quand il est lancé depuis un terminal, il s'initialise à partir de différents fichiers, affiche un message de prompt au début d'une ligne d'invite de commande et se place en mode lecture du clavier. Quand la ligne de commande est saisie et validée par la touche [Entrée], le shell interprète son contenu et l'exécute. Une fois l'exécution terminée, le shell affiche à nouveau le prompt et se remet en attente d'une nouvelle ligne.

La séquence de touches [Ctrl] D termine l'exécution du shell. La commande `exit` provoque également la terminaison du shell.

Remarque

Linux, comme Unix, distingue les minuscules des majuscules, dans les commandes, leurs options et arguments, ainsi que dans les noms de fichiers et de répertoires.

1.2.2 L'invite de commandes

Le shell attend des saisies au clavier sur une ligne appelée l'invite de commandes. La chaîne de caractères affichée au début de cette ligne s'appelle le **prompt**.

Le prompt est configurable (par la variable d'environnement `PS1`), son contenu par défaut est variable suivant les distributions. Il affiche en général le nom du compte utilisateur, le répertoire courant et un caractère `$` (compte non-administrateur) ou # (compte administrateur).

Exemple

Prompt par défaut de l'utilisateur `pba` sur le système `srvrh` (distribution RHEL 9) :

```
[pba@srvrh ~]$
```

Prompt par défaut de l'utilisateur `root` (administrateur) sur le système `srvdeb` (distribution Debian 12) :

```
root@srvdeb:~#
```

1.3 Utiliser le shell

1.3.1 La saisie sur la ligne de commande

Sur la ligne de commande, on peut déplacer le curseur avec les touches [Flèche à droite] et [Flèche à gauche], et effacer des caractères avec les touches [Retour arrière] ou [Suppr]. pour déclencher l'exécution, il faut appuyer sur la touche [Entrée].

Les raccourcis-clavier suivants peuvent être utilisés :

- **[Ctrl] A** : aller au début de la ligne.
- **[Ctrl] E** : aller en fin de ligne.
- **[Ctrl] L** : effacer le contenu de l'écran, et afficher l'invite en haut de celui-ci.
- **[Ctrl] U** : effacer la ligne jusqu'au début.
- **[Ctrl] K** : effacer la ligne jusqu'à la fin.

Exemples

Commande d'affichage de la date.

```
$ date
lun. 15 mai 2023 09:40:20 CEST
```

Commande d'affichage du chemin d'accès du répertoire courant.

```
$ pwd
/home/pba
```

1.3.2 Syntaxe générale des commandes

La plupart des commandes fournies avec les distributions Linux sont d'origine Unix, mais ont été réécrites dans le cadre du projet open source GNU. Leur syntaxe peut varier d'une version à l'autre.

Les commandes GNU/Linux ont en général la syntaxe suivante :

```
Commande [options] [arguments]
```

Une commande peut n'avoir ni option, ni argument. Les options sont le plus souvent identifiées par un caractère précédé d'un tiret : `-l`, `-p`, `-s`, etc. Si la commande accepte plusieurs options, on peut les spécifier les unes après les autres en les séparant par des espaces : `-l -r -t`, ou les grouper derrière un seul tiret : `-lrt`. L'ordre des options n'a pas d'importance, les deux syntaxes précédentes produisent le même résultat.

Remarque

Certaines options attendent un argument, par exemple un nom de fichier. Dans ce cas, on les séparera des autres : `-lrt -f monfichier` ou on les placera en dernière position : `-lrtf monfichier`.

Les arguments sont des chaînes de caractères séparées par un caractère espace ou tabulation. Si un argument doit contenir un espace, il faut l'encadrer par des guillemets simple `''` ou doubles `""`.

1.3.3 Exemple de commande : cal

La commande `cal` admet plusieurs options et arguments. Appelée seule, elle affiche le calendrier du mois en cours.

Exemple

```
$ cal
      mai 2023
lu ma me je ve sa di
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

La commande admet deux arguments optionnels. Si un seul est précisé, il s'agit de l'année, et l'intégralité du calendrier de cette année est affichée. Si deux arguments sont précisés, le premier est le mois, le second l'année.

Exemple

```
$ cal 12 1975
    décembre 1975
lu ma me je ve sa di
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

La commande prend quelques options, variables selon la version installée.

L'option `-m` (*monday*) affiche les jours de la semaine en commençant par lundi.

L'option `-s` (*sunday*) affiche les jours de la semaine en commençant par dimanche.

Exemple

```
$ cal -s 12 1975
    décembre 1975
di lu ma me je ve sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

L'option `-m3` permet d'afficher le mois précédent et le mois suivant le mois spécifié ou, sans argument, le mois courant.

Exemple

```
$ cal -m3 12 1975
cal -m3 12 1975
    novembre 1975         décembre 1975         janvier 1976
lu ma me je ve sa di  lu ma me je ve sa di  lu ma me je ve sa di
                1  2   1  2  3  4  5  6  7            1  2  3  4
 3  4  5  6  7  8  9   8  9 10 11 12 13 14   5  6  7  8  9 10 11
10 11 12 13 14 15 16  15 16 17 18 19 20 21  12 13 14 15 16 17 18
17 18 19 20 21 22 23  22 23 24 25 26 27 28  19 20 21 22 23 24 25
24 25 26 27 28 29 30  29 30 31              26 27 28 29 30 31
```

Avec une distribution de type Debian, on peut utiliser la commande similaire `ncal` qui possède une syntaxe un peu différente (la commande `cal` exécutant en fait `ncal`).

Exemple

Pour obtenir le même résultat qu'avec l'exemple précédent :

```
$ ncal -b -M -3 12 1975
   Novembre 1975         Décembre 1975          Janvier 1976
lu ma me je ve sa di  lu ma me je ve sa di  lu ma me je ve sa di
                1  2   1  2  3  4  5  6  7            1  2  3  4
 3  4  5  6  7  8  9   8  9 10 11 12 13 14   5  6  7  8  9 10 11
10 11 12 13 14 15 16  15 16 17 18 19 20 21  12 13 14 15 16 17 18
17 18 19 20 21 22 23  22 23 24 25 26 27 28  19 20 21 22 23 24 25
24 25 26 27 28 29 30  29 30 31              26 27 28 29 30 31
```

1.3.4 Enchaîner les commandes

On peut indiquer plusieurs commandes sur une même ligne de commande, en les séparant par un point-virgule. Elles seront exécutées successivement.

Exemple

```
$ date; pwd; cal -m
lun. 15 mai 2023 11:30:26 CEST
/home/pba
      mai 2023
lu ma me je ve sa di
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

1.3.5 Afficher du texte

La commande `echo` affiche les arguments qu'on lui passe, séparés par un espace et suivis d'un saut de ligne.

Exemple

```
$ echo Bonjour les amis
Bonjour les amis
```

Les arguments peuvent contenir des caractères spéciaux, issus du langage C, à condition de spécifier l'option `-e`. Les plus utilisés sont les suivants :

Séquence	Action
`\n`	Saut de ligne
`\t`	Tabulation horizontale

Séquence	Action
\c	Pas de saut de ligne après l'afffichage des arguments
\b	Retour d'un caractère en arrière
\\	Afficher l'antislash (barre oblique inverse)
\nnn	Afficher le caractère de code octal nnn

Exemple

```
$ echo -e "Salut.\tJe m'appelle XXX\b\b\bPersonne\n"
Salut. Je m'appelle Personne
```

Remarque

Cette commande est surtout utilisée dans les scripts, pour afficher des commentaires, des instructions utilisateurs, des messages d'erreur, etc.

1.3.6 Commandes internes et externes

Il existe deux types de commandes :

– Les **commandes externes** correspondent à des programmes exécutables stockés dans des fichiers. À l'exécution de la commande, le shell détermine l'emplacement du fichier correspondant et, s'il le trouve, lance son exécution en lui passant options et arguments éventuels.

– Les **commandes internes** sont internes au shell et exécutées directement par celui-ci. Le code exécutable de ces commandes fait partie intégrante de celui du shell (primitive du shell). Les commandes `cd` ou `pwd` en sont deux exemples.

Pour distinguer une commande interne d'une commande externe, on peut utiliser la commande interne `type`.

Exemple

```
$ type date
date est haché (/bin/date)
```

La commande `date` est une commande externe, mais son chemin est mémorisé par le shell (hashage).

```
$ type pwd
pwd est une primitive du shell
```

1.3.7 Séquences de contrôle

`[Ctrl] C` : provoque l'interruption de la commande en cours.

`[Ctrl] D` : en début de ligne, termine la saisie si la commande est en lecture du clavier, ou termine le shell courant s'il est en attente du clavier.

Exemple

Sans argument, la commande `sort` trie les lignes saisies au clavier. Pour arrêter la saisie, il faut taper `[Ctrl] D`, en début de ligne :

```
$ sort
bbbbbbbbbbb
aaaaaaa
zzzzzzz
eeeeeeee
[Ctrl]D
aaaaaaa
bbbbbbbbbbb
eeeeeeee
zzzzzzz
```

1.4 Historique des commandes

Le shell conserve dans un fichier du répertoire de connexion de chaque compte utilisateur un historique des lignes de commande, nommé `.bash_history`. Il est possible d'y naviguer avec les touches [Flèche en haut] et [Flèche en bas] sachant que la flèche du haut permet de remonter dans l'historique. On peut modifier ou non la ligne de commande affichée, et demander son exécution avec la touche [Entrée] (quelle que soit la position du curseur sur la ligne de commande).

La commande `history` affiche les dernières lignes de commande saisies.

Exemple

```
$ history
...
 1000  date
 1001  pwd
 1002  uname -a
 1003  ls
 1004  fc -l -5
 1005  history
```

La commande `fc -l` affiche les quinze dernières lignes de commande, en les numérotant. On peut indiquer le nombre de lignes de commande à afficher en le passant en option.

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 :

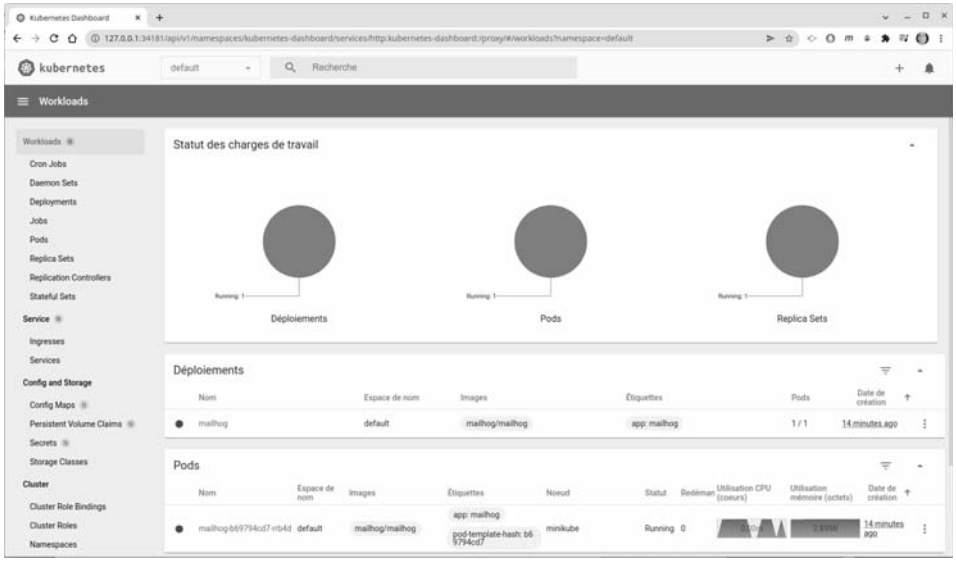
```
    Using image k8s.gcr.io/metrics-server/metrics-server:v0.4.2
    The 'metrics-server' addon is enabled
```

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
```



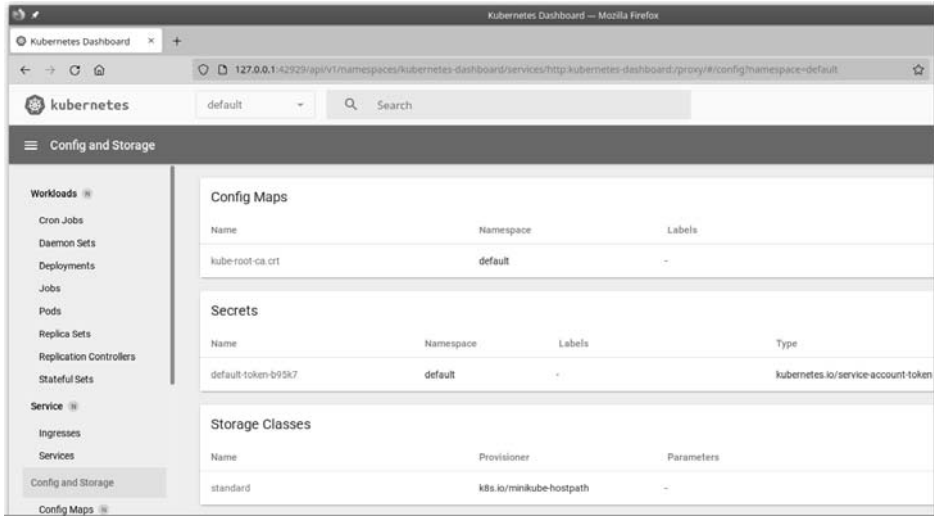
Page d'accueil du tableau de bord de Kubernetes

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**.

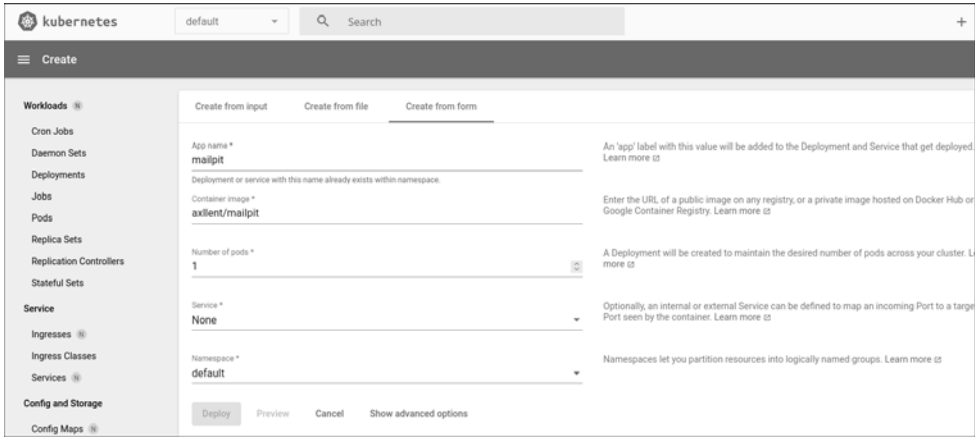
Sur l'écran qui apparaît, cliquez sur le bouton **+** en haut à droite pour créer un nouvel objet. Sur cet écran, cliquez sur le lien **Create from form** puis entrez les informations suivantes :

– 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.



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