# Jeu des technologies de cloud computing : Kubernetes contre Cloud Foundry

Par Max Shapiro

Date de publication : 26 mai 2019

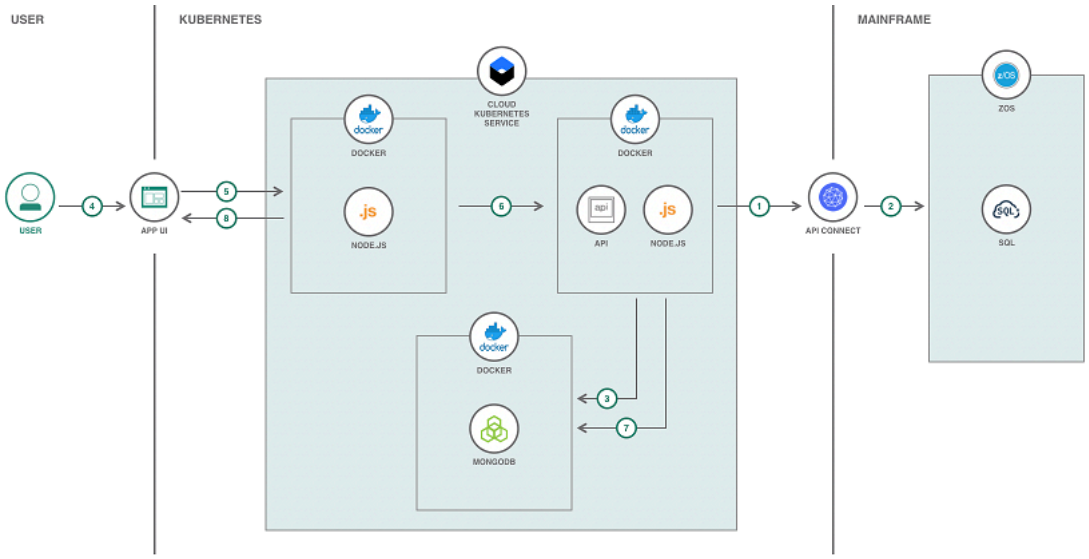
TOUT PUBLIC

Kubernetes et Cloud Foundry sont deux technologies qui vous permettent de déployer et d'exécuter des applications sur le cloud. Dans ce blog, je parle de mes expériences dans l'utilisation des deux technologies sur IBM Cloud™ avec le même modèle de code, « Créer un système de dossiers de santé avec la technologie moderne de cloud computing et le code de l'ancien mainframe ».

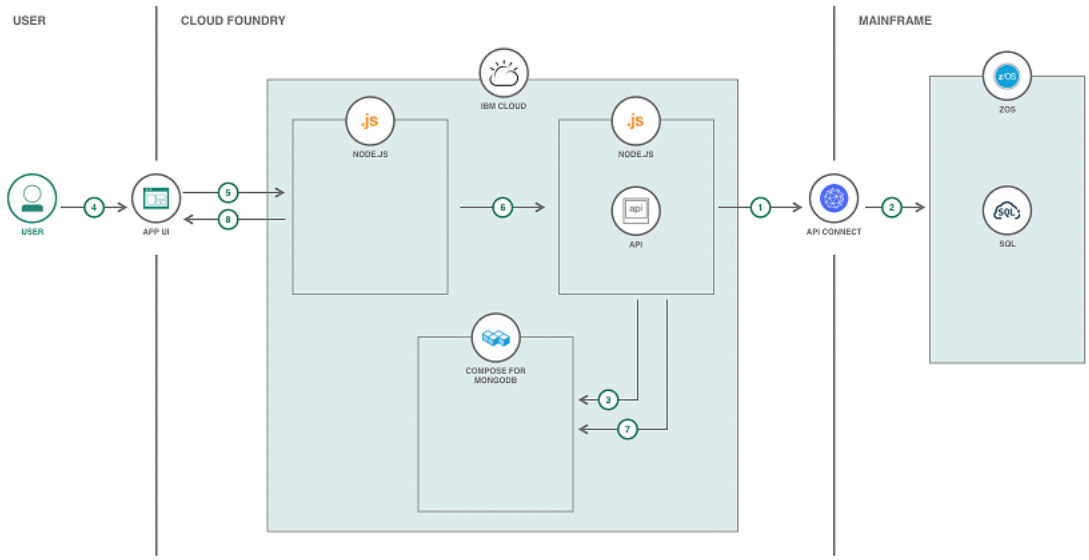
Un espace de discussion vous est proposé sur le forum. **Commentez**

# I - Architecture

## I-A - Kubernetes




## I-B - Cloud Foundry




## I-C - Comparaison

Dans le **modèle de code**, les deux architectures ont suivi les étapes suivantes :

1. L'API de service de données agit comme un pipeline de données et est déclenchée pour mettre à jour le lac de données avec les données mises à jour des dossiers de santé en appelant les API de connexion API associées à l'unité centrale zOS.
2. Les API API Connect traitent les données pertinentes des dossiers de santé à partir de l'entrepôt de données de l'ordinateur central zOS et envoient les données via le pipeline de données.
3. Le pipeline de données du Service des données traite les données de l'entrepôt de données de l'ordinateur central zOS et met à jour le lac de données MongoDB.

4 L'utilisateur interagit avec l'interface utilisateur pour visualiser et analyser les analyses.
5 La fonctionnalité de l'interface utilisateur avec laquelle l'utilisateur interagit est gérée par Node.js. Node.js est l'endroit où les appels API sont initialisés.
6 Les appels API sont traités dans le service de données Node.js et sont traités en conséquence.
7 Les données sont recueillies à partir du lac de données MongoDB à partir d'appels API.
8 Les réponses des appels API sont traitées en conséquence par l'interface utilisateur de l'application.

### I-D - Score

Puisque les deux suivent le même modèle d'architecture, il n'y a pas de favori et les deux obtiennent un point.

- Kubernetes : 1
- Cloud Foundry : 1

## II - Déploiement

### II-A - Kubernetes

**Le service IBM Cloud Kubernetes** utilise des conteneurs pour exécuter l'application. Cela signifie que j'avais besoin de conteneuriser le modèle de code ; pour ce faire, j'ai utilisé **Docker**. J'ai dû trouver des conteneurs compatibles pour les différentes parties du modèle de code. Cela comprenait deux conteneurs Node.js : un pour l'interface utilisateur frontale et l'autre pour le service de données et les API. De plus, j'ai également inclus un conteneur MongoDB pour la base de données des lacs. Une fois que tous les conteneurs ont été configurés et fonctionnent correctement sur ma machine locale, j'ai poussé les conteneurs vers **Docker Hub**.

Les conteneurs étant prêts à être déployés sur le cloud, j'ai ensuite dû approvisionner un cluster sur Kubernetes. Le déploiement de ce modèle de code sur IBM Cloud peut être réalisé en utilisant le cluster Lite ou Standard. Une fois le cluster créé, j'ai dû appliquer les fichiers YAML qui configuraient et déployaient chaque conteneur depuis Docker Hub. Pour le cluster standard, des fichiers YAML supplémentaires ont été appliqués pour configurer l'entrée des deux conteneurs Node.js.

Une fois tous les conteneurs déployés, j'ai pu exécuter avec succès et interagir avec le modèle de code sur IBM Cloud.

### II-B - Cloud Foundry

Cloud Foundry sur IBM Cloud inclut un **SDK pour Node.js** pour exécuter les applications Node.js sur Cloud Foundry. Pour ce modèle de code, deux instances du SDK devaient être provisionnées : une pour l'interface utilisateur frontale et l'autre pour le service de données et les API. De plus, j'ai fourni une instance de **Compose pour MongoDB** pour la base de données des lacs.

Une fois les trois instances exécutées, j'ai dû configurer un fichier YAML manifeste pour les deux parties Node.js du modèle de code. Une fois le fichier YAML configuré, je l'ai poussé vers IBM Cloud pour déployer le code de ma machine locale vers le cloud.

Une fois que le code a été déployé avec succès, j'ai pu exécuter avec succès et interagir avec le modèle de code sur IBM Cloud.

### II-C - Comparaison

Le déploiement vers Kubernetes a pris plus d'étapes et m'a demandé d'avoir un compte Docker Hub, mais à la fin, toutes les parties du modèle de code étaient exécutées dans le même cluster sur IBM Cloud. Cela signifie que toutes les parties du modèle de code sont accessibles à partir de la même URL de base.

Même s'il y avait moins d'étapes à déployer dans Cloud Foundry, les différentes parties du modèle de code ont été déployées séparément et ont donc dû être accessibles à partir de différentes URL.

Les conteneurs pour Kubernetes nécessitent une image Docker pour créer le conteneur. Par conséquent, la langue et la version correctes devaient être spécifiées. De plus, les commandes d'installation et d'initialisation devaient être spécifiées pour que le conteneur puisse être compilé et exécuté. Par ailleurs, Cloud Foundry n'avait pas besoin de tout cela. La seule chose qu'il fallait savoir, c'était la langue, qui était nécessaire lors de l'approvisionnement de l'application. Tout le reste a été détecté automatiquement.

Contrairement au déploiement vers Cloud Foundry, le déploiement vers Kubernetes sur IBM Cloud a une option gratuite. L'utilisation de l'option gratuite signifie que vous ne pouvez pas configurer l'entrée et que vous devez donc accéder au modèle de code par l'adresse IP et les ports. De plus, il utilise http plutôt que https.

### II-D - Score

En raison de la simplicité de déploiement, je donnerais ici l'avantage à Cloud Foundry.

- Kubernetes : 1
- Cloud Foundry : 2

## III - Mise à jour

### III-A - Kubernetes

Avec le modèle de code fonctionnant sur Kubernetes sur IBM Cloud, je pouvais alors me concentrer sur les mises à jour. Après que ces mises à jour ont été exécutées avec succès sur ma machine locale, j'ai pu déployer le code mis à jour sur le cloud. Ce processus fonctionne de la même manière que le processus de mise en place. Tout d'abord, les conteneurs mis à jour doivent être poussés vers Docker Hub. Ensuite, les fichiers YAML qui ont été utilisés pour déployer les conteneurs associés aux conteneurs mis à jour doivent être supprimés et réappliqués. Ce que cela signifie, c'est que si j'ai mis à jour le code qui était dans un seul des conteneurs Node.js, j'aurais seulement besoin de redéployer ce conteneur.

### III-B - Cloud Foundry

Avec le modèle de code s'exécutant sur Cloud Foundry sur IBM Cloud, je pouvais alors me concentrer sur les mises à jour. Une fois ces mises à jour exécutées avec succès sur ma machine locale, j'ai pu déployer le code mis à jour sur le cloud. Ce processus fonctionne de la même manière que la mise en place. Le même fichier YAML manifeste est poussé vers IBM Cloud pour redéployer le code de la machine locale vers le cloud.

### III-C - Comparaison

Comme pour le déploiement, il est beaucoup plus facile de mettre à jour sur Cloud Foundry que sur Kubernetes. Il ne nécessite qu'une seule commande, alors que Kubernetes nécessite la suppression et le redéploiement de chaque conteneur à mettre à jour. J'ai également remarqué que parfois le conteneur/service n'effacerait pas complètement sur Kubernetes et m'obligeait à l'effacer plusieurs fois.

Une différence clé entre les deux est la portabilité et le versioning des applications vers le cloud. Docker permet de créer facilement différentes versions d'un conteneur. Pour déployer différentes versions d'une application ou même plusieurs instances d'une seule version, Kubernetes rend ce processus simple. Pour un cluster différent provisionné, les mêmes fichiers YAML peuvent être utilisés. Cependant, si une version différente d'un conteneur est utilisée, l'image dans le fichier doit être changée à l'endroit où elle se trouve dans Docker Hub. En fait, vous n'avez même pas besoin du code de votre machine locale pour déployer, vous avez juste besoin des fichiers YAML nécessaires. D'autre part, l'utilisation de Cloud Foundry est plus compliquée. Pour chaque application supplémentaire que vous souhaitez

déployer, les composants nécessaires doivent être provisionnés sur IBM Cloud, ou tout autre cloud vraiment. De plus, le code doit se trouver sur votre machine locale. Pour le versioning, une option que vous pouvez utiliser est de créer différentes branches sur GitHub qui sont associées à une version et de pousser le code pour la branche que vous voulez déployer.

### III-D - Score

En raison de la différence de portabilité et de versionnage, je donne ici l'avantage à Kubernetes.

- Kubernetes : 2
- Cloud Foundry : 2

## IV - Débogage

### IV-A - Kubernetes

Ce n'est pas parce qu'une application s'exécute avec succès en local qu'elle s'exécutera également avec succès sur le cloud. J'ai appris cela en déployant le modèle de code sur le nuage. Le débogage de ce modèle de code localement via le navigateur et les logs via la console du terminal était assez similaire au fonctionnement de Kubernetes sur le débogage d'IBM Cloud. Le débogage du navigateur fonctionne de la même manière que l'exécution locale, cependant les logs peuvent être trouvés en exécutant kubectl logs <podname>. Si vous êtes intéressé par la lecture d'un journal particulier, le nom du pod est le pod où se trouve le journal du conteneur. Malheureusement, le débogage nécessite parfois de pousser plusieurs conteneurs mis à jour vers le cloud, ce qui ralentit le processus.

### IV-B - Cloud Foundry

Le processus de débogage sur Cloud Foundry est similaire à celui de Kubernetes, où vous pouvez utiliser le navigateur pour déboguer et vérifier les logs. Les logs peuvent aussi être exécutés en exécutant une commande : ibmcloud cf logs <appname>, où appname est le nom de l'application pour laquelle vous souhaitez lire les logs.

### IV-C - Comparaison

J'ai trouvé la lecture des journaux plus facile à suivre sur Kubernetes. Dans Cloud Foundry, les logs enveloppaient ce qui était affiché dans la console dans les logs de CF, ce qui peut être déroutant à comprendre au début.

Parfois, lorsque je déboguais, j'avais besoin de redéployer le modèle de code plusieurs fois afin de trouver le bogue. Bien que cela ait ralenti le processus de débogage pour Kubernetes et Cloud Foundry, c'était plus important lorsque Kubernetes était utilisé.

### IV-D - Score

Je préfère le débogage sur Kubernetes que sur Cloud Foundry et par conséquent, Kubernetes obtient l'avantage.

- Kubernetes : 3
- Cloud Foundry : 2

# V - Manipulation d'intrants importants

## V-A - Kubernetes

Initialement, lorsque je travaillais sur ce modèle de code, je travaillais avec un petit ensemble de données (< 1MB) à envoyer à l'API Node.js chargée de remplir la base de données MongoDB. Quand j'ai décidé d'utiliser un ensemble de données plus important (~ 6MB), j'ai commencé à rencontrer des problèmes. Tout d'abord, lors de l'exécution locale, j'ai dû augmenter la limite de l'analyseur de corps dans l'application Node.js. Après avoir corrigé cela et avoir réussi à remplir la base de données localement, je l'ai essayée sur Kubernetes. Malheureusement, j'obtenais l'erreur suivante en essayant d'envoyer l'ensemble de données plus grand :

```
<head><title>413 Request Entity Too Large</title></head>
<body bgcolor="white">
<center><h1>413 Request Entity Too Large</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

J'ai trouvé que j'avais besoin de définir le fichier ingress.bluemix.net/client-max-body-size dans le fichier ingress qui était associé au service conteneur avec les API.

## V-B - Cloud Foundry

Heureusement, Cloud Foundry a implémenté la mise à l'échelle automatique, donc l'envoi d'un ensemble de données plus important à mon API n'était pas un problème et fonctionnait sans problème.

## V-C - Comparaison

La fonction de mise à l'échelle automatique de Cloud Foundry fonctionne très bien pour ce modèle de code, car la mise à l'échelle n'a pas à s'inquiéter puisqu'elle se fait automatiquement. Avec Kubernetes, une limite sur la taille des données doit être fixée.

## V-D - Score

La fonction de mise à l'échelle automatique de Cloud Foundry donne l'avantage à Cloud Foundry.

- Kubernetes: 3
- Cloud Foundry: 3

# VI - Fonctionnement

## VI-A - Kubernetes

La façon dont j'ai configuré ce modèle de code pour Kubernetes signifiait que toutes les parties du modèle de code fonctionnaient dans le même cluster Kubernetes. Cela signifie que toutes les parties du modèle de code sont accessibles à partir de la même adresse URL/IP de base. Si des fichiers d'entrée sont utilisés, vous avez également la possibilité de contrôler ce qui est exposé à l'accès et comment il l'est. Par exemple, avec ce modèle de code, j'ai exposé l'interface utilisateur sur https://some-url et les API sur https://api.some-url.

## VI-B - Cloud Foundry

La façon dont j'ai configuré ce modèle de code en utilisant Cloud Foundry signifiait que chaque partie fonctionnait séparément sur le nuage. Malheureusement, avec la façon dont ce modèle de code est structuré, il ne peut pas exécuter tout l'ensemble dans une seule application. Cela signifie que le modèle de code est exposé sur des URL différentes pour chaque partie.

## VI-C - Comparaison

Comme je l'ai mentionné dans la section déploiement, contrairement au déploiement vers Cloud Foundry, le déploiement vers Kubernetes sur IBM Cloud a une option gratuite. L'utilisation de l'option gratuite signifie que vous ne pouvez pas configurer l'entrée et que vous devez donc accéder au modèle de code via l'adresse IP et les ports. De plus, il utilise http plutôt que https. Cloud Foundry utilise automatiquement https.

Comme les parties du modèle de code sont dispersées sur Cloud Foundry sur IBM Cloud, j'ai remarqué que l'exécution de ce modèle de code était plus lente que sur Kubernetes.

## VI-D - Score

Je donne ici l'avantage à Kubernetes en raison de la vitesse à laquelle l'application a pu fonctionner sur Kubernetes par rapport à Cloud Foundry.

- Kubernetes : 4
- Cloud Foundry : 3

## VII - Conclusion

Kubernetes bat Cloud Foundry 4:3 sur le score final. Avec un score aussi proche, cela montre que Kubernetes et Cloud Foundry ont des avantages à utiliser l'un ou l'autre. Lorsqu'il s'agit de décider lequel utiliser pour votre application, il est important de considérer quels aspects sont prioritaires pendant le cycle de vie de l'application. Pour cette application, si je devais en choisir une, je choisirais Kubernetes. Compte tenu de la possibilité qu'une grande quantité de données soit traitée par l'application, la vitesse du système et l'expérience de l'utilisateur devraient être la priorité.

## VIII - Prochaines étapes

Cela dépend de vos préférences personnelles selon que vous décidez d'utiliser Kubernetes ou Cloud Foundry. Mais si vous n'êtes toujours pas sûr ou si vous voulez voir quels autres modèles de code ou tutoriels nous avons, consultez **Cloud Foundry sur IBM Developer** ou **Kubernetes sur IBM Developer**. Encore une fois, **regardez le modèle de code** que j'ai écrit, qui a inspiré ce billet de blog.

## IX - Remerciements Developpez.com

Developpez.com remercie IBM pour l'autorisation de publication de ce tutoriel. Tous nos remerciements aussi à **Laethy** pour la mise au gabarit et **Claude Leloup** pour la relecture orthographique.