



**POLYTECHNIQUE
MONTRÉAL**

UNIVERSITÉ
D'INGÉNIERIE

Polytechnique Montréal

Département de génie informatique et génie logiciel

**Cours LOG3000
Processus du génie logiciel**

**A2022 - Travail Pratique 5 - DÉPLOIEMENT BLEU-VERT DANS
OPENSIFT - Groupe 01**

**Rendu par :
Aghilès Gasselin 2013772
Maximiliano Falicoff 2013658
Mohammed Fenjiro 1901744**

**Soumis à
Varun Shiri**

**Date:
Pour le 30 novembre 2022**

5) QUESTIONS

Q1:

Cas 1: Stratégie Rolling Update

Cette stratégie consiste à progressivement changer la version d'une application par une autre.

Un exemple utilisant le rolling update est le service ECS de AWS (Elastic Container Service). Les instances de l'ancienne explications seront une à la fois remplacées par la nouvelle version.

Cas 2: Stratégie Blue Green

La stratégie Blue Green consiste à avoir deux applications, une avec les nouvelles fonctionnalités. Le déplacement du Blue (ancien) vers le Green (nouveau) permet d'être sûr que les fonctionnalités de la version verte marchent bien avant de donner accès au grand public.

Une compagnie qui utilise cette stratégie sont les sites utilisant le cadriciel Drupal. Lors d'un déploiement, la base de données est copiée pour l'instance green. Cela permet de minimiser le downtime qui aurait pu avoir lieu dans une stratégie classique car la base de donnée doit être rétablie à partir d'une sauvegarde.

Cas 3: Stratégie Canari

La stratégie consiste à déployer la nouvelle version de l'application à un petit groupe d'utilisateurs afin de pouvoir tester toutes les fonctionnalités d'un point de vue du client. Par la suite, on va déplacer le reste des utilisateurs vers la nouvelle version après le fix des problèmes s'il y en a.

Par exemple Google Chrome possède une branche canary qui est déployé à chaque nuit (nightly) que les utilisateurs peuvent utiliser à leur discrétion. Le principe reste le même, un sous section d'utilisateurs pourra tester les nouvelle fonctionnalités et par la suite le reste des utilisateurs aura autant accès à ces fonctionnalités lors du release final.

Q2: Kubernetes est un système open-source permettant d'automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées.

OpenShift: est une plateforme de conteneurs Kubernetes pour les entreprises. Elle fonctionne comme un PAAS (platform as a service) pour héberger des systèmes qui utilisent des clusters kubertes pour leurs déploiements.

Q3: Kubernetes permet de gérer le déploiement d'une application avec une configuration qui peut être utilisée par la suite pour automatiser le déploiement.

Pour le gestion des conteneurs, l'écriture d'un fichier de configuration permet de facilement déployer l'application, tout en spécifiant comment l'application doit se mettre à l'échelle en fonction du trafic, voire elle peut même réaliser cela automatiquement. Le fait que ca suit le principe Infrastructure as Code permet de configurer l'environnement de production d'une

telle façon à ce que l'environnement soit le même à chaque fois et donc l'application peut rouler n'importe où (cloud ou local) sur n'importe quelle machine.

Q4: Une mise à l'échelle est le processus d'ajuster la puissance d'une machine individuelle ou ajouter carrément de nouvelles machines au cluster. Généralement cela peut être soit configuré manuellement dans la configuration de déploiement ou peut être géré automatiquement par la plateforme qui héberge le cluster. En fonction du trafic, si les ressources de la machine commencent à être remplies, soit une nouvelle machine sera ajoutée au cluster pour recevoir une partie du trafic, ou la machine est ajoutée pour en supporter plus.

Cette mise à l'échelle peut permettre d'éviter que le système plante car il reçoit trop de trafic qui pourrait alors consommer toutes les ressources et donc ralentir le système.

Q5: Un pod est un ensemble d'un ou plusieurs conteneurs. Les conteneurs à l'intérieur du pod partagent les ressources allouées au pod. C'est un meilleur moyen d'optimiser l'utilisation des ressources. Les conteneurs partagent aussi le même réseau local dans le pod ce qui est très pratique pour faire parler des conteneurs entre eux.

Q6: L'auto guérison d'application est une combinaison de capacités effectuelles par l'application Kubernetes. L'auto guérison peut permettre de redémarrer les conteneurs ayant planté, remplacer les conteneurs qui ont besoin d'une mise à jour, désactiver les conteneurs qui ne répondent pas et ne remplissent pas les tests de status. et empêcher les conteneurs d'apparaître aux utilisateurs tant qu'ils ne sont pas prêts (opérationnels).

Q7: Le but du routage est de pouvoir rediriger les utilisateurs vers tel ou tel conteneur de façon simple lors d'une mise à jour, une maintenance ou lors de problèmes sur les conteneurs par exemple.

Q8:

Etape 1 : `oc new-app https://github.com/Moa1er/blue-green-openshift#master --name=blue --strategy=source`

Cette étape consiste à forker le github "<https://github.com/Moa1er/blue-green-openshift>" depuis le master et créer l'application kubernetes.

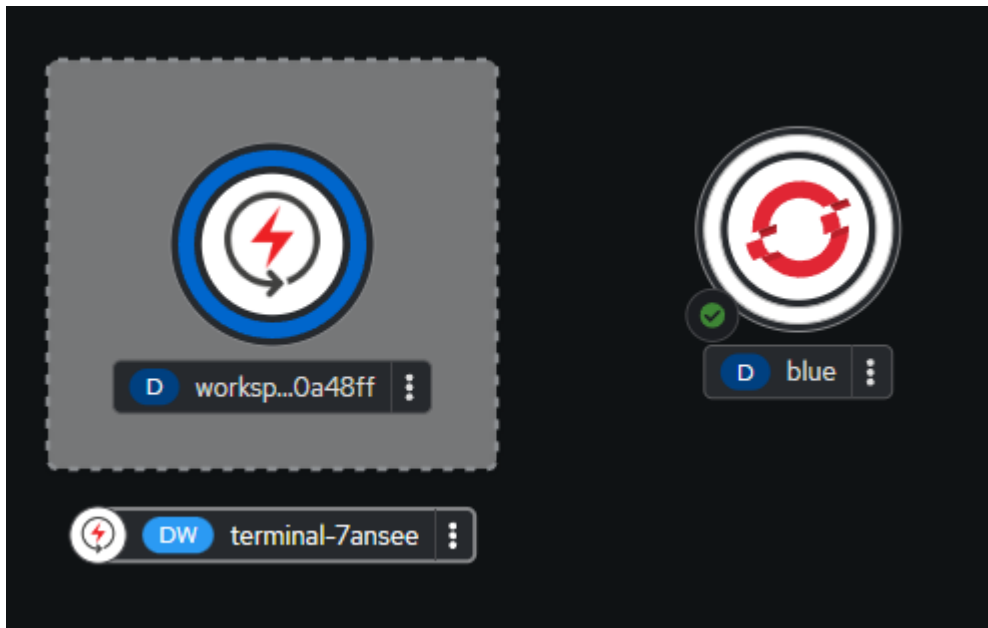
```
bash-4.4 ~ $ oc new-app https://github.com/mfalicoff/blue-green-openshift#master --name=blue --strategy=source
--> Found image df080e2 (8 months old) in image stream "openshift/nodejs" under tag "16-ubi8" for "nodejs"

Node.js 16
-----
Node.js 16 available as container is a base platform for building and running various Node.js 16 applications and frameworks. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Tags: builder, nodejs, nodejs16

* The source repository appears to match: nodejs
* A source build using source code from https://github.com/mfalicoff/blue-green-openshift#master will be created
* The resulting image will be pushed to image stream tag "blue:latest"
* Use 'oc start-build' to trigger a new build

--> Creating resources ...
imagestream.image.openshift.io "blue" created
buildconfig.build.openshift.io "blue" created
deployment.apps "blue" created
service "blue" created
--> Success
Build scheduled, use 'oc logs -f buildconfig/blue' to track its progress.
Application is not exposed. You can expose services to the outside world by executing one or more of the commands below:
```



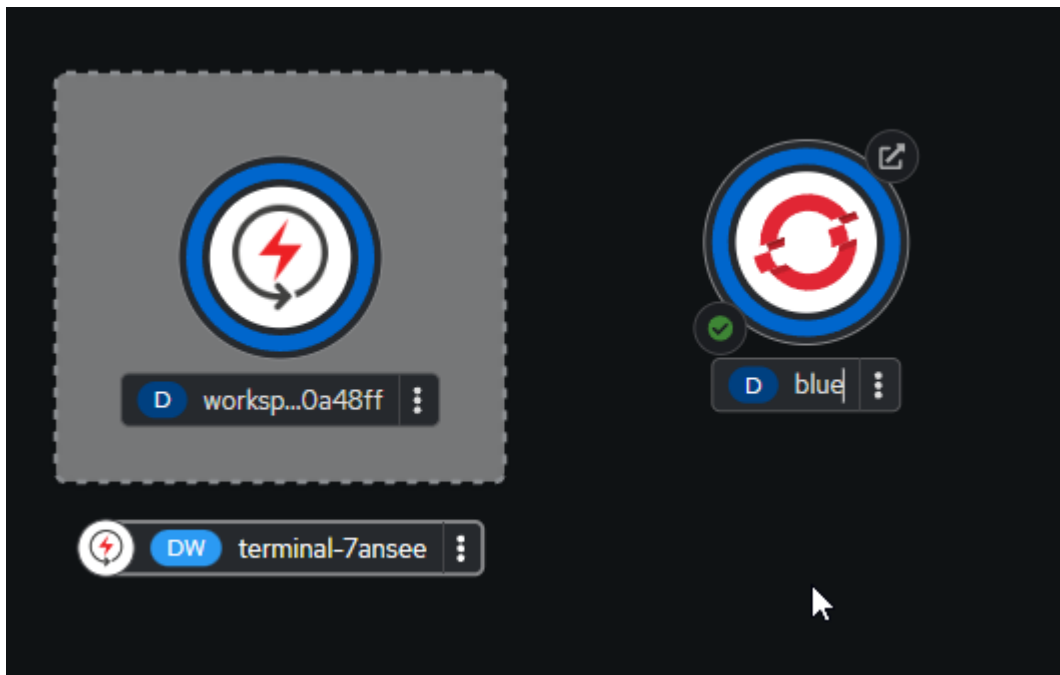
Etape 2 : `oc expose service blue --name=bluegreen`

En réponse à cette commande nous obtenons le résultat suivant :

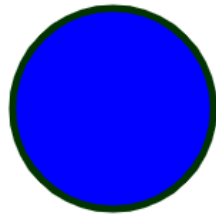
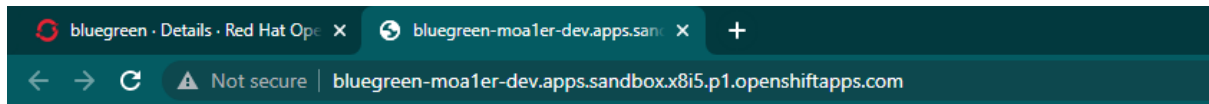
route.route.openshift.io/bluegreen exposed

Cette commande permet de créer une adresse pour notre pod afin de se connecter au conteneur. En l'occurrence, pour le pod "blue", cela va créer un service avec un lien et port public dont le nom du service sera "bluegreen";

On peut voir sur le screenshot ci- dessous qu'une petite icône s'est rajoutée en haut à droite qui envoie vers le lien public du service.



Si l'on va sur le lien public nous avons le résultat suivant :



Etape 3: `oc new-app https://github.com/Moa1er/blue-green-openshift#green --name=green`
 On fait la même chose que l'étape 1 mais depuis la branche "green" et auquel on donne le nom "green";

On obtient alors ce layout suivant :



Etape 4 : `oc get route/bluegreen -o yaml | sed -e 's/name: blue$/name: green/' | oc replace -f -`

Cette étape n'est pas nécessaire mais permet de changer le service opérationnel de notre application vers le service "green";

À l'inverse on peut faire la commande `"oc get route/bluegreen -o yaml | sed -e 's/name: green$/name: blue/' | oc replace -f -"` qui va changer le service opérationnel de notre application vers le service "blue";

```
bash-4.4 ~ $ oc get route/bluegreen -o yaml | sed -e 's/name: blue$/name: green/' | oc replace -f -
route.route.openshift.io/bluegreen replaced
```

Cette commande est décomposée de la façon suivante :

oc get route/bluegreen -o yaml permet d'avoir la configuration des routes dans un format yaml pour les commandes suivantes

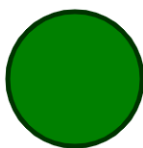
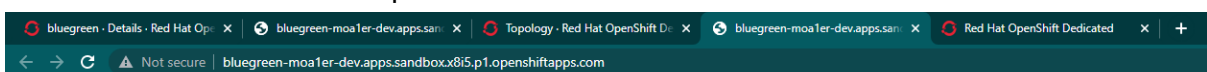
sed -e 's/name: blue\$/name: green/' modifie le fichier de config yaml pour remplacer le nom actuellement "blue" en "green"

oc replace -f - remplace la vieille configuration par la nouvelle ce qui résulte en un changement de route.

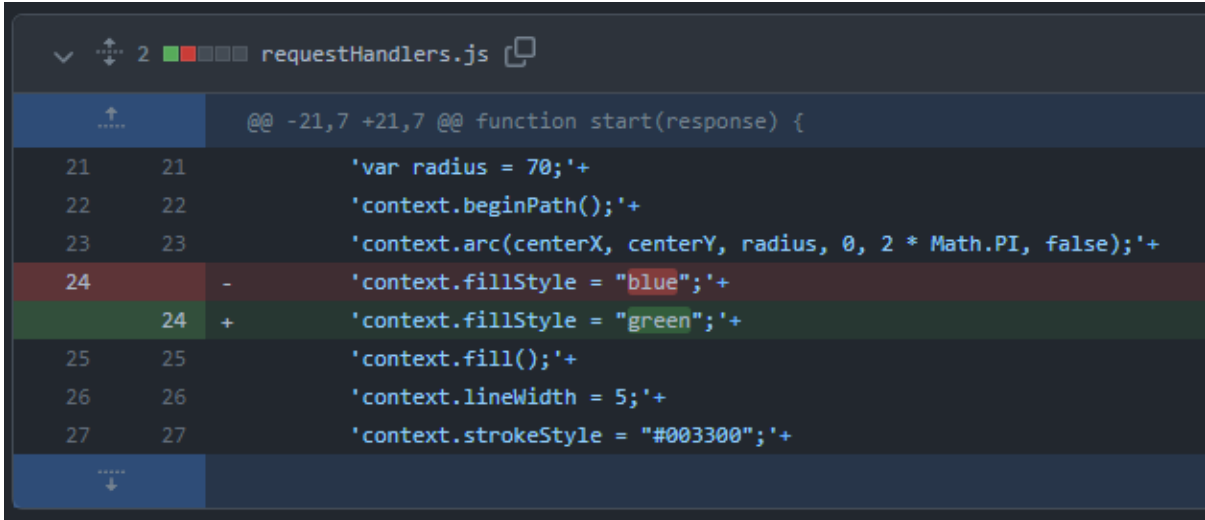
À la suite de cette commande on peut observer le résultat suivant (l'icône pour le lien sortant du pod est passé de notre pod "blue" à notre pod "green").



Si l'on va sur le nouveau lien public nous avons le résultat suivant :



Les changements à faire dans le code pour obtenir le cercle vert sur notre branche “green” ont été les suivants :



```
requestHandlers.js

@@ -21,7 +21,7 @@ function start(response) {
21      21      'var radius = 70;'+
22      22      'context.beginPath();'+
23      23      'context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);'+
24      -      'context.fillStyle = "blue";'+
24      +      'context.fillStyle = "green";'+
25      25      'context.fill();'+
26      26      'context.lineWidth = 5;'+
27      27      'context.strokeStyle = "#003300";'+
```

5.1 Question de rétroaction

1. Ce laboratoire nous a pris environ 4h par personnes à faire. Ce laboratoire présente une bonne charge de travail et l'effort est à un bon niveau. Le plus long à été de comprendre comment marchait openShift avec RedHat. Le tp est très éducatif.