LOG3000 – Processus du génie logiciel

Hiver 2021

TP No.5

Groupe 2 - Équipe no3

1960266 – Yanis Toubal

1947497 – Yuhan Li

Soumis à : Isabella Vieira Ferreira

Mardi 27 avril 2021

Q1.

Cas 1. Stratégie Rolling Update

Cette stratégie consiste à remplacer progressivement les instances de la version précédente d'une application par des instances de la nouvelle version. Ce type de déploiement attend généralement que les nouveaux *pods* soient prêts via un contrôle de disponibilités avant de *scale down* les anciens composants. [1]

Le service Amazon ECS (Elastic Container Service) est un bon exemple de déploiement *rolling update*. Dans un déploiement dans Amazon ECS, les conteneurs exécutant des versions précédentes de l'application sont remplacés un par un par des conteneurs exécutant de nouvelles versions de l'application. Cela est fait lorsqu'un nouveau déploiement de service est démarrée, car le planificateur de service d'Amazon ECS va remplacer les tâches en cours d'exécution par de nouvelles tâches. [2]

Cas 2. Stratégie Blue-Green

Une stratégie de déploiement Blue-Green consiste à exécuter deux versions d'une application en même temps et de déplacer le trafic de la version en production (version verte) vers la version plus récente (version bleue). Ce déplacement est fait au niveau de l'équilibreur de charge lorsque la nouvelle version a été testée et validée pour toutes les exigences. Il est possible d'utiliser une stratégie Rolling ou de changer de service lors d'une route. [1]

Un exemple d'utilisation du déploiement Blue-Green serait pour les sites web basés sur Drupal (cadriciel *open source* pour la gestion de contenu web). En effet, lorsque nous recevons un trafic élevé à la suite de la construction du site, il vaut mieux utiliser la stratégie Blue-Green plutôt que de passer par le processus conventionnel de développement/test/*staging*/production. Cette stratégie permet de réduire significativement les *downtime* et offre des capacités de restauration faciles. [3]

Cas 3. Stratégie Canari

La stratégie Canari consiste à déployer des versions vers un sous-ensemble d'utilisateurs ou de serveurs. L'idée est de déployer d'abord le changement sur un petit sous-ensemble de serveurs, de le tester, puis de le déployer sur le reste des serveurs. Ainsi, il s'agit de déplacer progressivement le trafic de production de l'ancienne version vers la nouvelle version. Par exemple, 85% des demandes vont à l'ancienne version, puis 15% à la nouvelle. Cela permet de ne pas impacter les autres parties du serveur lorsqu'un déploiement a échoué. [1]

Cette stratégie est surtout utilisée lorsqu'il manque des tests, lorsqu'ils ne sont pas fiables, ou lorsqu'il y a de l'incertitude quant à la stabilité de la nouvelle version. Prenons l'exemple d'un environnement possédant 4 serveurs web. Plutôt que de déployer sur toutes les cibles de déploiement de l'environnement, la stratégie Canari va procéder avec une étape de vérification. En effet, on va d'abord déployer sur un, deux ou 3 serveurs, tester ce déploiement, puis finalement déployer sur les serveurs restants. Cette stratégie est lente, mais elle est très utile quand il est important de s'assurer du bon fonctionnement de la nouvelle version sans beaucoup impacter les utilisateurs. [4]

Q2. Quelle est la relation entre OpenShift et Kubernetes?

Kubernetes est un cadriciel *open-source* qui fournit des services d'orchestration de conteneurs. L'objectif ultime est d'aider à automatiser de nombreux processus impliqués dans le déploiement, la mise à l'échelle, la gestion et les diverses autres opérations liées aux applications conteneurisées. [5]

OpenShift, lui, est considéré à la fois comme un logiciel de conteneurisation ainsi qu'une plateforme de service PaaS (Platform-as-a-service). Ainsi, pour réaliser le déploiement de projet dans des conteneurs, OpenShift utilise des services Kubernetes pour ces conteneurs. En d'autres mots, à l'intérieur de OpenShift, Kubernetes gère les applications conteneurisées et fournit des mécanismes de déploiement, de mise à l'échelle et de maintenance. [6]

Q3. Dans vos propres mots, qu'est-ce que les Kubernetes et pourquoi est-il utile pour DevOps?

Kubernetes est une plateforme portable, extensible et *open-source* qui sert à gérer les charges de travail (« *workloads* ») et les services qui sont sous forme de conteneurs. Cette plateforme favorise la configuration ainsi que l'automatisation. [7]

Le rôle de Kubernetes dans un processus DevOps se situe au niveau de la gestion des conteneurs. En effet, Kubernetes permet d'automatiser des aspects tels que le déploiement, le « *scaling* » ainsi que la résilience des conteneurs, ce qui peut amener un grand gain de temps et d'argent pour une organisation. Aussi, Kubernetes optimise le passage entre les environnements. D'autres avantages d'utiliser Kubernetes est que cette plateforme permet de déployer n'importe où et sans temps d'arrêt, de définir l'infrastructure et la configuration « as code » ce qui offre beaucoup de flexibilité. Elle peut également rouler sur le cloud et « *on-premise* », et est très flexible et permet de faire de nombreuses actions sur les conteneurs tout en minimisant l'impact. [8]

Q4. Expliquez la différence entre la mise à l'échelle horizontale et la mise à l'échelle verticale.

La différence entre une mise à l'échelle horizontale et une mise à l'échelle verticale est qu'à la verticale on ajoute davantage de puissance à une machine (un conteneur dans le cas de OpenShift) existante alors qu'à l'horizontale on ajoute davantage de machines (de «container» dans le cas de OpenShift). En comparant ces 2 approches, on peut voir que la mise à l'échelle horizontale offre beaucoup plus de flexibilité et est donc la solution de choix pour la plupart des situations. Par contre, pour que cette dernière soit fonctionnelle, il faut s'assurer que l'application du côté client ne dépend pas de l'état du serveur (« *stateless* ») puisque le serveur peut changer. Dans le cas contraire, une mise à l'échelle verticale est donc préférable. Il existe une fonctionnalité de « *autoscaling* » sur OpenShift qui permet d'ajuster les ressources selon le trafic en utilisant les pods. [9]

Q5. Qu'est-ce qu'un pod? Expliquez comment ils fonctionnent.

Les pods sont les objets déployables les plus petits et les plus élémentaires qu'on peut créer dans Kubernetes. Un pod peut être vu comme une instance d'un ou de plusieurs conteneurs. En effet, lorsqu'un pod exécute plusieurs conteneurs, les conteneurs sont gérés comme une seule entité et partagent les ressources du pod. Toutefois, l'exécution de plusieurs conteneurs dans un seul pod est un cas d'utilisation avancé. Plus encore, un pod contient un réseau et un espace de stockage qui sont partagés par ses conteneurs. [10]

OpenShift utilise le concept de pods de Kubernetes. Il est important de noter que OpenShift traite les pods comme étant immuable c'est-à-dire que lorsqu'un changement dans un pod est nécessaire, le pod sera détruit et un autre sera créé avec les changements. Aussi, les pods sont considérés comme étant temporaires et sont majoritairement contrôlés par le système et non pas par l'utilisateur. [11]

Q6. Qu'est-ce que "l'auto-guérison d'application"?

OpenShift permet la résilience des « *containers* » avec des mécanismes d'auto-guérison. Cela signifie que s'il y a un problème avec une des applications, la plateforme va entamer des actions réparatives tels que redémarrer le conteneur ou encore rediriger le trafic vers une autre application. Un exemple serait que si une application cesse de fonctionner OpenShift va redémarrer le « *container* » afin de réinitialiser l'état de l'application. Les conditions qui permettent à OpenShift de savoir si une application est fonctionnelle ou non peuvent être choisies par le créateur du « *container* » et il est donc de sa responsabilité de bien définir ses conditions. [12]

Q7. Quel est le but du routage?

Le but du routage est d'exposer un service afin qu'il soit accessible par Internet. On permet donc à des acteurs externes d'accéder aux « *clusters* » de l'application. Cela est fait en attribuant au service un nom d'hôte qui peut être atteint de l'extérieur. [13]

Q9. Maintenant, vous devez implémenter le déploiement bleu-vert avec le code source disponible sur

Avant de commencer, nous avons « *fork* » le répertoire [14] contenant le code « *blue-green-openshift* ». Ensuite, nous avons créé la branche « *green* » dans laquelle nous avons modifié la couleur du cercle en vert. Le répertoire que nous avons utilisé est celui de Yanis [15] (newidentity1).

D'abord, comme on peut le voir à la Figure 1 à la page suivante, nous nous sommes connectés au compte *developer* avec le mot de passe *developer.* Puis, sur la même figure, il est possible de voir que nous avons créé le projet *bluegreen*.

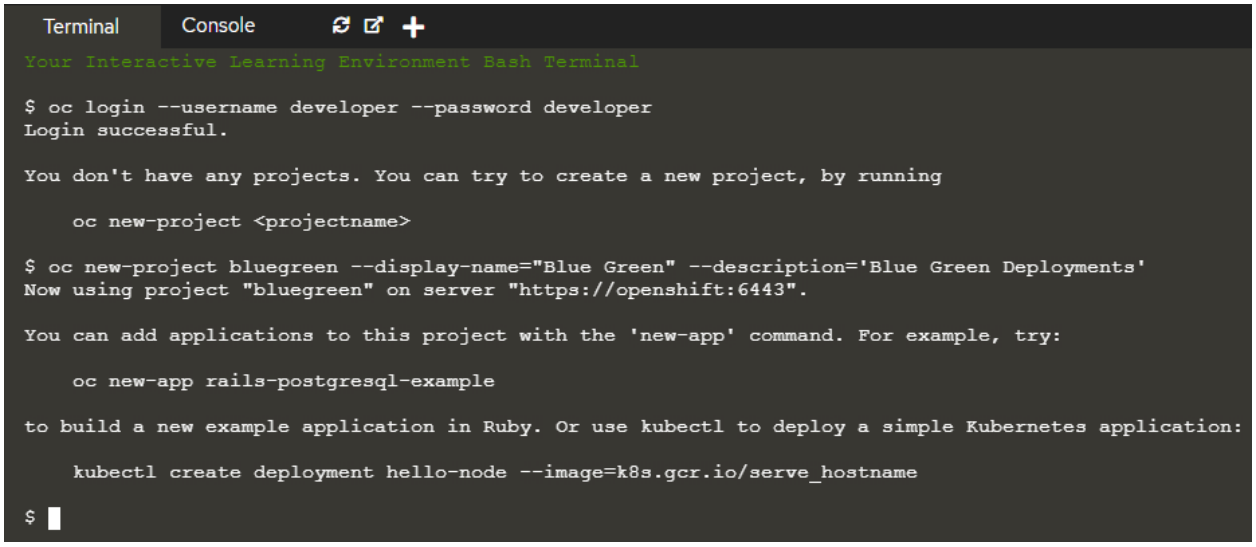

Figure 1. Connexion au compte « *developer* » et création du projet « *bluegreen* »

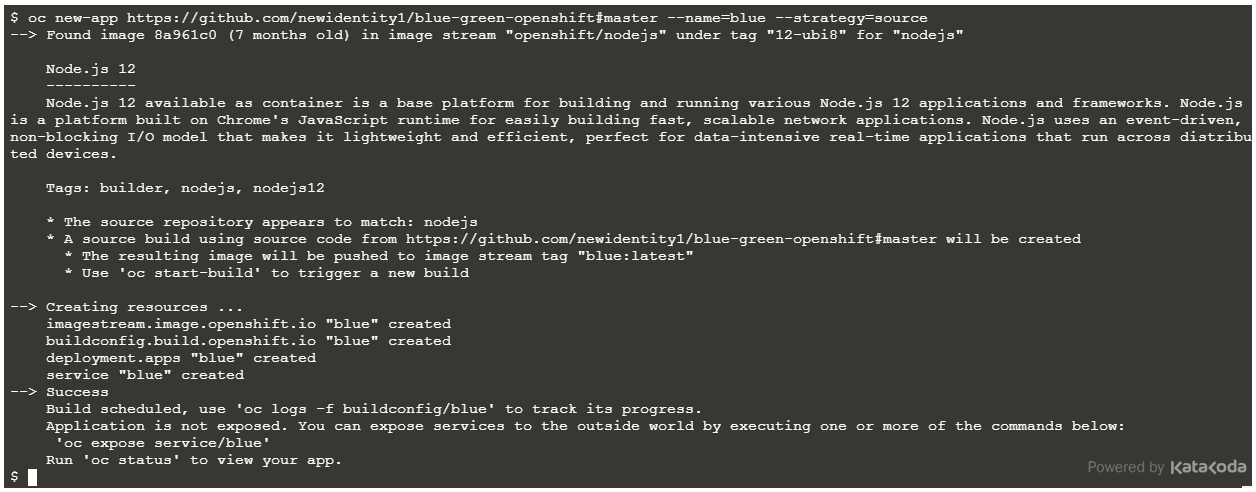

Figure 2. Création de l'application « *blue* » à partir de la branche master

Nous avons créé une nouvelle application avec la commande « *oc new-app* » à partir de la branche « *master* » du répertoire Github de Yanis. Le paramètre *--name* indique le nom que nous souhaitons donner à l'application et, comme on peut le voir sur la Figure 2 ci-dessus, elle s'intitule « *blue* ». Puis, afin de rendre cette nouvelle application visible aux personnes externes, nous avons effectué la commande à la Figure 3 ci-dessous. En gros, cette commande va créer une route publique qui va exécuter le code de notre application « *blue* ».

Figure 3. Création de la route pour l'application « *blue* »

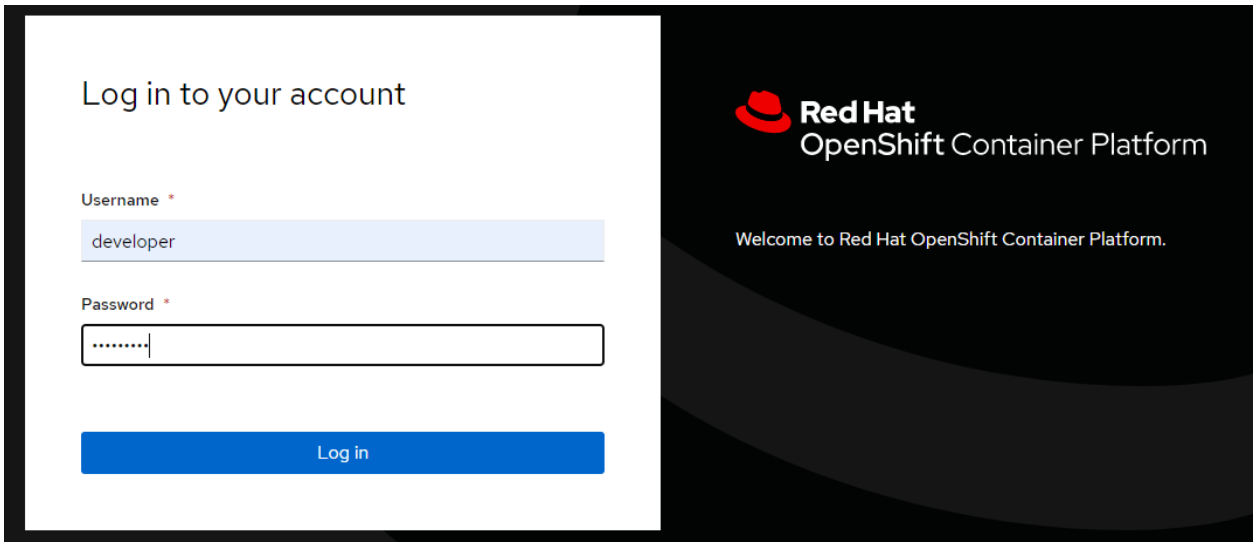

Figure 4. Connexion sur la plateforme «OpenShift Container» avec le compte «developer»

Ensuite, pour confirmer et mieux visualiser les commandes que nous faisons, nous nous sommes connectés manuellement sur la plateforme de conteneur OpenShift. Comme on peut le voir à la Figure 4, nous avons utilisé le même identifiant que celui avec lequel nous avons créé notre projet *bluegreen*.

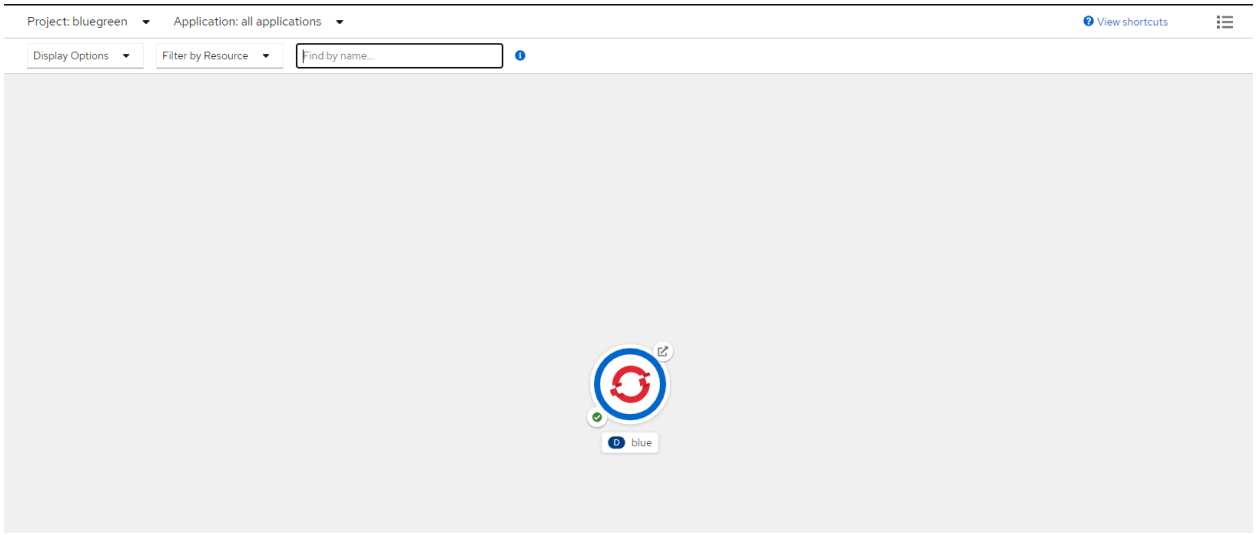

Figure 5. Représentation visuelle de l'application « *blue* »

On peut observer que l'application que nous avons créé dans la Figure 2 est bel est bien présente. On peut également voir que la route créée dans la Figure 3 est présente grâce à l'icône en haut à droite sur le cercle de « *blue* ».

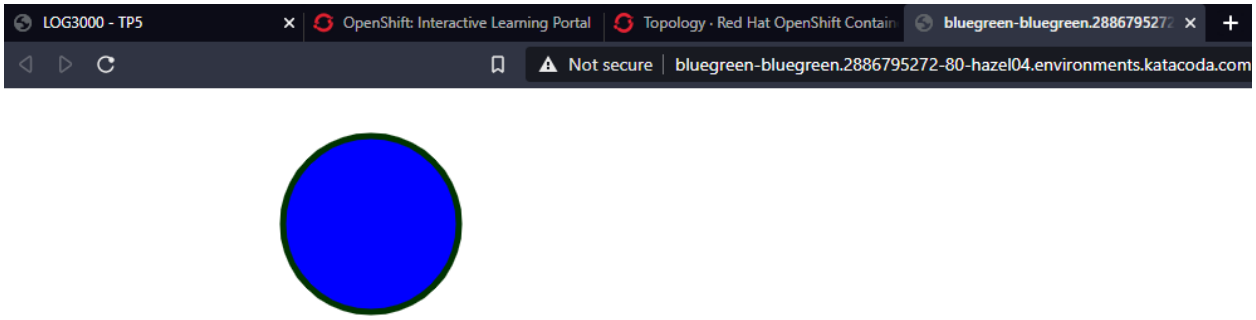

Figure 6. Affichage de l'application «blue» à travers la route

En visitant la route, on peut observer sur la Figure 6 ci-dessus qu'on obtient bel est bien le code du répertoire « *master* » ; soit un cercle bleu.

```
$ oc new-app https://github.com/newidentity1/blue-green-openshift#green --name=green
--> Found image 8a961c0 (7 months old) in image stream "openshift/nodejs" under tag "12-ubi8" for "nodejs"

    Node.js 12
    ----------
    Node.js 12 available as container is a base platform for building and running various Node.js 12 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 distribu
ted devices.

    Tags: builder, nodejs, nodejs12

    * The source repository appears to match: nodejs
    * A source build using source code from https://github.com/newidentity1/blue-green-openshift#green will be created
      * The resulting image will be pushed to image stream tag "green:latest"
      * Use 'oc start-build' to trigger a new build

--> Creating resources ...
    imagestream.image.openshift.io "green" created
    buildconfig.build.openshift.io "green" created
    deployment.apps "green" created
    service "green" created
--> Success
    Build scheduled, use 'oc logs -f buildconfig/green' 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:
     'oc expose service/green'
    Run 'oc status' to view your app.
$
```

Powered by Katacoda

Figure 7. Création de l'application « *green* » à partir de la branche green

Ensuite, de la même façon que pour notre application « *blue* », nous avons créé une nouvelle application avec la commande « *oc new-app* » à partir de la branche « green » du répertoire Github de Yanis. Cette application a le nom de « *green* » comme on peut le voir sur la Figure 7.

Sur la Figure 8 à la page suivante, on peut voir qu'il y a désormais deux applications présentes dans notre projet « *bluegreen* ». Une contenant le cercle bleu (« *blue* ») et l'autre contenant le cercle vert (« *green* »).

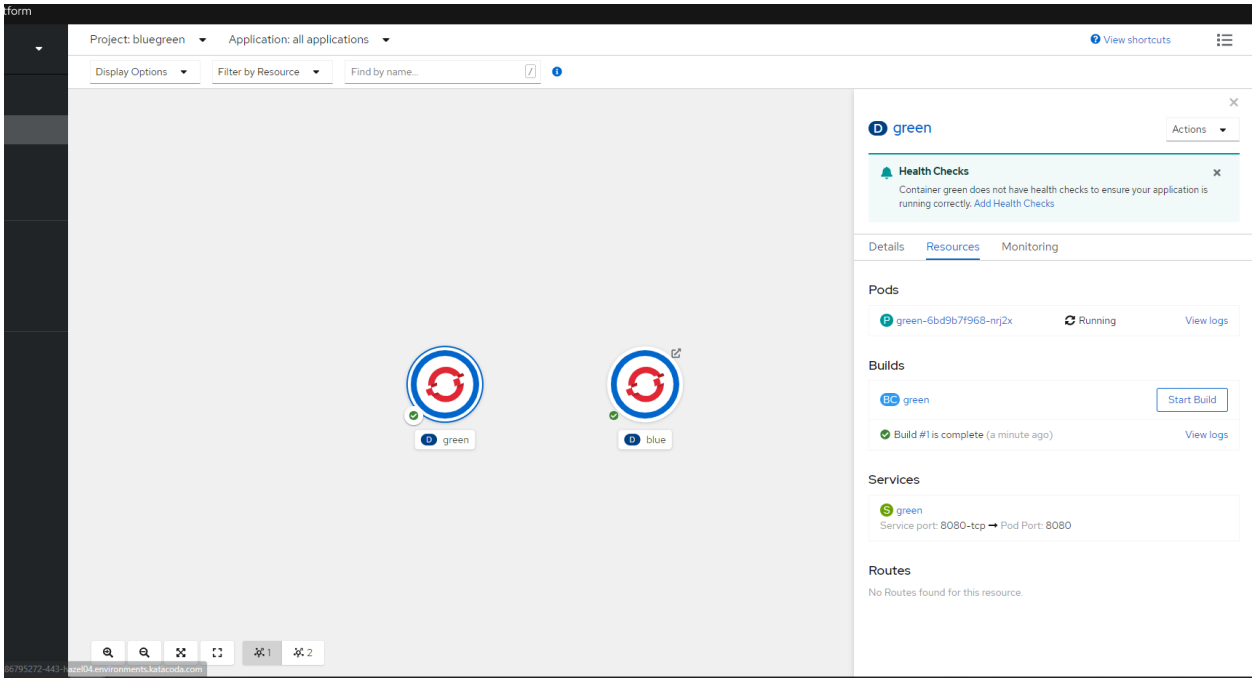

Figure 8. Représentation visuelle des applications « *blue* » et « *green* » avec la route pointant sur « *blue* »

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

Figure 9. Commande qui alterne le code à exécuter pour la route (de « *blue* » à « *green* »)

La commande ci-dessus permet de modifier la route du projet (en format yaml) en remplaçant «name: blue» par «name: green». Cela va faire en sorte de faire pointer la route du projet de l'application « *blue* » vers l'application « *green* ».

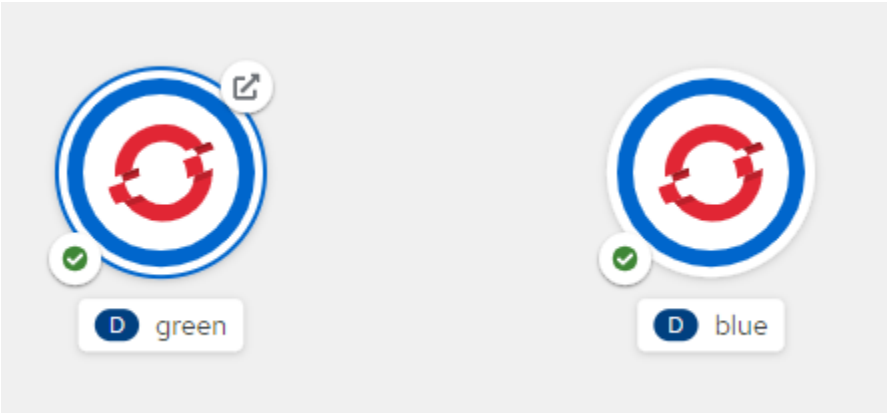

Figure 10. Représentation visuelle des applications « *blue* » et « *green* » avec la route pointant sur « *green* »

On peut maintenant voir sur la Figure 10 que la route pointe maintenant sur l'application « *green* ».

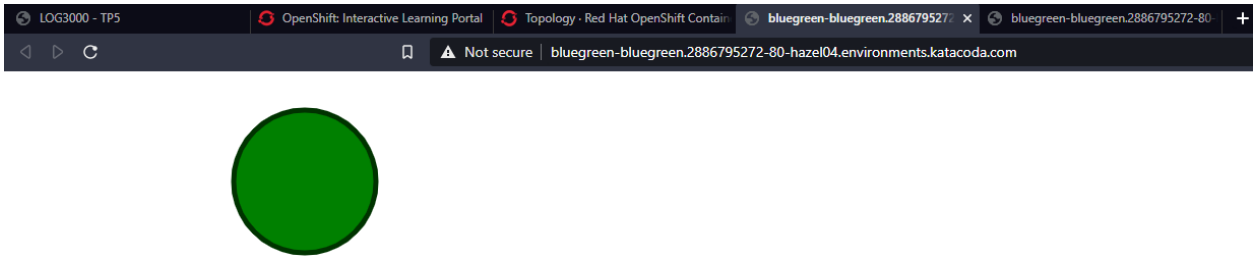

Figure 11. Affichage qui confirme visuellement le changement au service « *green* »

La route affiche maintenant le contenu de l'application « *green* », c'est-à-dire le code de la branche « *green* » du répertoire.

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

Figure 12. Commande qui alterne le code à exécuter pour la route (de « *green* » à « *blue »*)

La commande sur la Figure 12 ci-dessus permet de modifier la route du projet (en format yaml) en remplaçant «name: green» par «name: blue». Cela va faire en sorte de faire pointer la route de l'application « *green* » vers l'application *«blue»*.

Puis, on peut maintenant voir sur la Figure 13 que la route pointe effectivement à nouveau sur l'application « *blue* ».

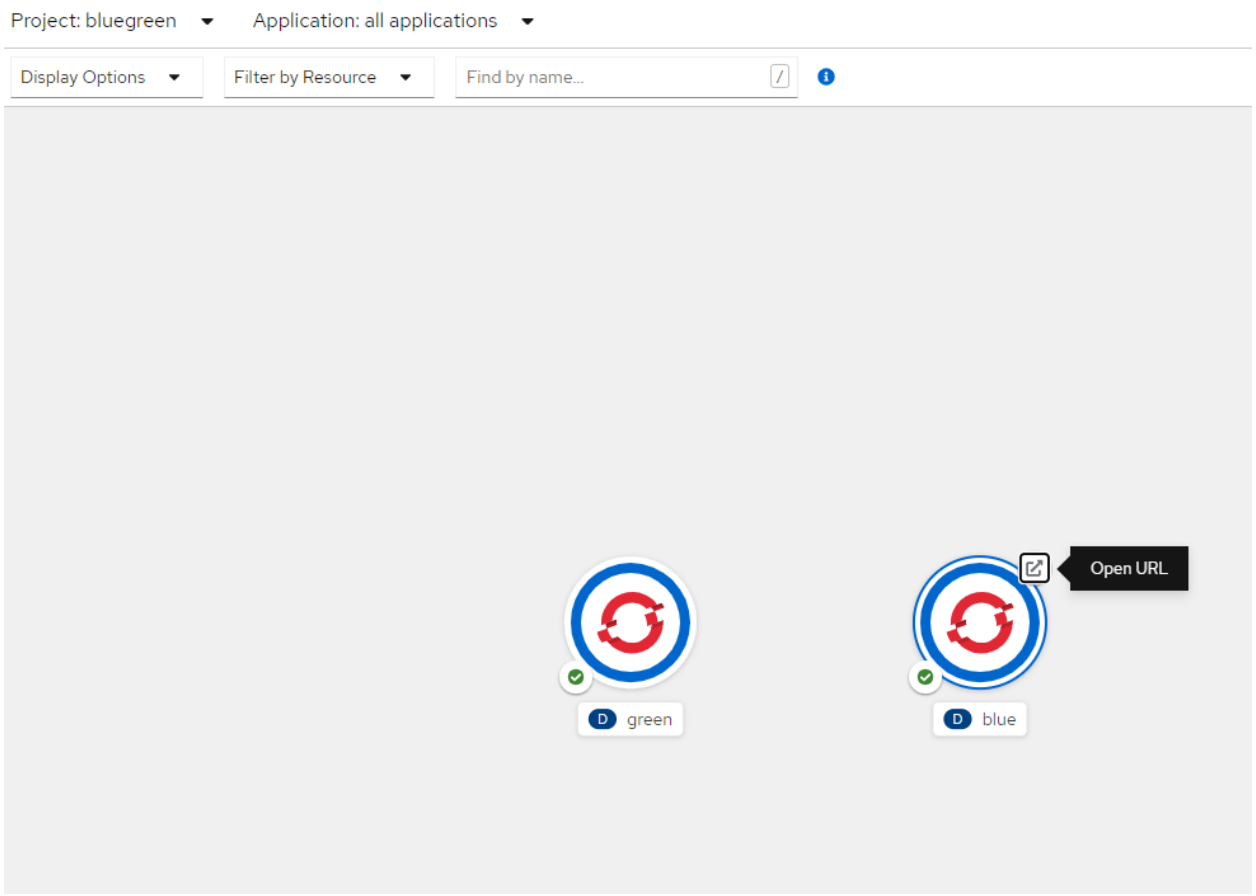

Figure 13. Confirmation visuelle que l'URL reprend bel et bien désormais le code de « *blue* »

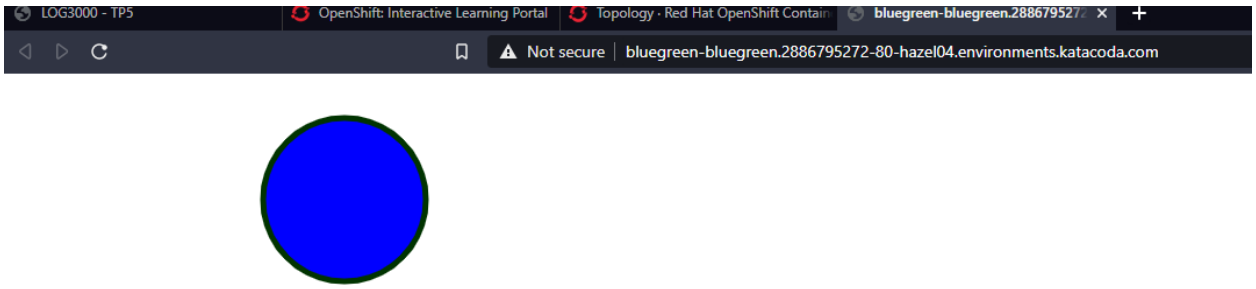

Figure 14. Affichage qui confirme visuellement le changement au service « *blue* »

Le route affiche de nouveau le contenu de l'application «*blue*»!

Question de rétroaction

Nous avons chacun passé environ 5 heures sur ce travail pratique, ce qui revient à un total de 10 heures-personnes. Nous sommes satisfaits de cette charge de travail et trouvons que c'est tout à fait adéquat.

Références

[1] https://thenewstack.io/deployment-strategies/

[2] https://docs.aws.amazon.com/AmazonECS/latest/developerguide/deployment-type-ecs.html

[3] https://opensenselabs.com/blog/articles/blue-green-deployment-drupal

[4] https://octopus.com/docs/deployments/patterns/canary-deployments#:~:text=Canary%20deployments%20are%20a%20pattern,the%20rest%20of%20the%20servers.

[5] https://www.simplilearn.com/kubernetes-vs-openshift-article

[6] https://docs.openshift.com/enterprise/3.0/architecture/infrastructure_components/kubernetes_infrastructure.html

[7] https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[8] https://rancher.com/blog/2020/create-kubernetes-devops-pipeline

[9] https://developers.redhat.com/blog/2019/11/11/autoscaling-red-hat-fuse-applications-with-openshift/#:~:text=There%20are%20two%20types%20of,running%20application's%20or%20container's%20runtime.

[10] https://kubernetes.io/docs/concepts/workloads/pods/

[11] https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/pods_and_services.html

[12] https://developers.redhat.com/blog/2017/10/17/container-images-openshift-part-4-cloud-readiness/

[13] https://docs.openshift.com/enterprise/3.0/architecture/core_concepts/routes.html

[14] https://github.com/isabellavieira/blue-green-openshift.git

[15] https://github.com/newidentity1/blue-green-openshift