



Inception-of-Things (IoT)

Résumé: Ce document est un sujet d'Administration Système.

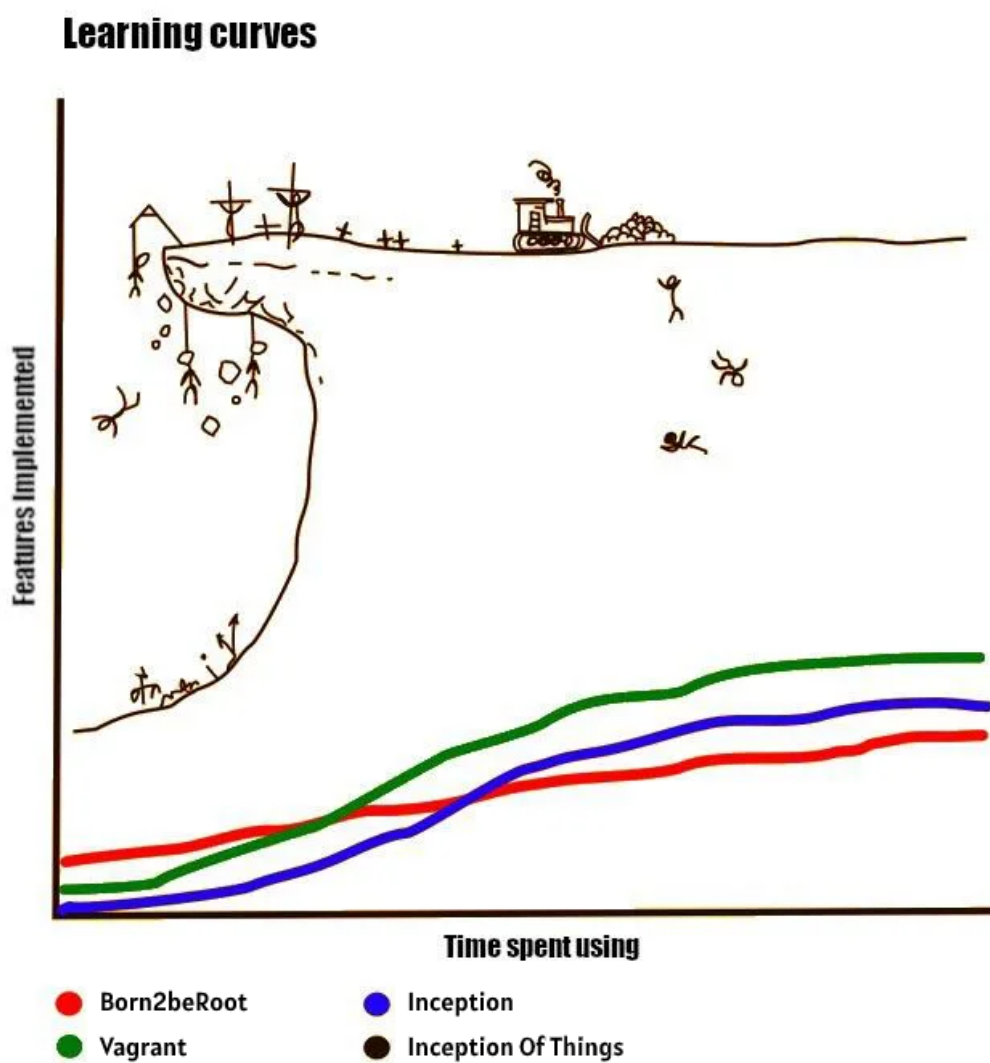
Version: 1

Table des matières

I	Préambule	2
II	Introduction	3
III	Consignes générales	4
IV	Partie obligatoire	5
IV.1	Partie 1 : K3s avec Vagrant	6
IV.2	Partie 2 : K3s avec trois applications simples	9
IV.3	Partie 3 : K3d avec Argo CD	12
V	Partie Bonus	16
VI	Rendu et peer-évaluation	17

Chapitre I

Préambule



Chapitre II

Introduction

Ce projet a pour but d'approfondir vos connaissances en vous faisant utiliser K3d ainsi que K3s avec Vagrant.

Vous apprendrez à mettre en place une machine virtuelle personnelle avec Vagrant et CentOS. Ensuite, à utiliser K3s et son Ingress. Enfin, vous utiliserez K3d qui vous simplifiera la vie.

Vous aurez ainsi un début d'expérience dans l'utilisation de Kubernetes.



Ce projet est une introduction basique à Kubernetes. En effet, cet outil est trop complexe à appréhender en un seul sujet.

Chapitre III

Consignes générales

- L'intégralité de ce projet est à réaliser dans une **machine virtuelle**.
- Vous devez rendre tous les fichiers nécessaires à la configuration de votre projet dans des dossiers à la racine de votre dépôt (cf. Rendu et peer-évaluation pour plus d'informations). Pour les trois sous-parties de la partie obligatoire, dans les dossiers : p1, p2 et p3. Pour la partie bonus, dans le dossier : bonus.
- Ce sujet requiert de mettre en pratique des notions que, selon votre parcours, vous n'avez possiblement pas encore abordées. Nous vous conseillons donc de ne pas avoir peur de lire beaucoup de documentation sur l'utilisation de K8s avec K3s, ainsi que sur K3d.



Vous pouvez utiliser tout les outils que vous souhaitez pour la mise en place de votre **machine virtuelle** hôte ainsi que pour le provider utilisé dans Vagrant.

Chapitre IV

Partie obligatoire

Ce projet consiste à vous faire mettre en place plusieurs environnements en suivant des règles spécifiques.

Ce projet est découpé en trois parties à faire dans l'ordre indiqué :

- Partie 1 : K3s avec Vagrant
- Partie 2 : K3s avec trois applications simples
- Partie 3 : K3d avec Argo CD

IV.1 Partie 1 : K3s avec Vagrant

Dans cette première partie, vous devez mettre en place **2 machines**.

Vous allez rédiger votre premier fichier **Vagrantfile** avec pour système d'exploitation la **dernière version stable** de **CentOS**. Il est **FORTEMENT** conseillé d'allouer le strict minimum en matière de ressources, à savoir : 1 CPU, 512 Mo de RAM (ou 1024). Les machines doivent être lancées avec **Vagrant**.

Voici les spécifications attendues :

- Avoir le login d'une personne de votre groupe comme nom de machine. Pour la première machine, le hostname sera suivi de la lettre majuscule S (comme *Server*). Pour la seconde, de SW (comme *ServerWorker*).
- Avoir une IP dédiée sur l'interface eth1. Pour la première machine (*Server*), l'IP sera 192.168.42.110. Pour la seconde (*ServerWorker*), l'IP sera 192.168.42.111.
- Pouvoir se connecter en SSH sur les deux machines sans mot de passe.



Vous mettrez en place votre fichier Vagrantfile en suivant des pratiques modernes.

Vous devez installer K3s sur chaque machine :

- Dans la première (*Server*), en mode contrôleur.
- Dans la seconde (*ServerWorker*), en mode agent.



Vous allez devoir utiliser **kubectl** (et donc l'installer).

Voici un petit **exemple** d'un fichier Vagrantfile :

```
$> cat Vagrantfile
Vagrant.configure(2) do |config|
  [...]
  config.vm.box = REDACTED
  config.vm.box_url = REDACTED

  config.vm.define "wils" do |control|
    control.vm.hostname = "wils"
    control.vm.network REDACTED, ip: "192.168.42.110"
    control.vm.provider REDACTED do |v|
      v.customize ["modifyvm", :id, "--name", "wils"]
      [...]
    end
    config.vm.provision :shell, :inline => SHELL
    [...]
    SHELL
    control.vm.provision "shell", path: REDACTED
  end
  config.vm.define "wilsW" do |control|
    control.vm.hostname = "wilsW"
    control.vm.network REDACTED, ip: "192.168.42.111"
    control.vm.provider REDACTED do |v|
      v.customize ["modifyvm", :id, "--name", "wilsW"]
      [...]
    end
    config.vm.provision "shell", inline: <<-SHELL
    [...]
    SHELL
    control.vm.provision "shell", path: REDACTED
  end
end
end
```


Inception-of-Things (IoT)

Voici un exemple lorsqu'on lance les machines virtuelles :

```
→ p1 vagrant up
Bringing machine 'wils' up with 'virtualbox' provider...
Bringing machine 'wilSW' up with 'virtualbox' provider...
[...]
→ p1 vagrant ssh wils          → p1 vagrant ssh wilSW
[vagrant@wils ~]$             [vagrant@wilSW ~]$
```

Voici un exemple lorsque la configuration est incomplète :

```
[vagrant@wils ~]$ k get nodes -o wide
NAME    STATUS    ROLES    AGE    VERSION    INTERNAL-IP    EXTERNAL-IP    OS-IMAGE    KERNEL-VERSION    CONTAINER-RUNTIME
wils    Ready    control-plane,master    4m37s    v1.21.4+k3s1    192.168.42.110    <none>    CentOS Linux 8    4.18.0-240.1.1.el8_3.x86_64    containerd://1.4.9-k3s1
[vagrant@wils ~]$ ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.42.110  netmask 255.255.255.0  broadcast 192.168.42.255
    inet6 fe80::a00:27ff:fe79:56d8  prefixlen 64  scopeid 0x20<link>
    ether 08:00:27:79:56:d8  txqueuelen 1000  (Ethernet)
    RX packets 10  bytes 2427 (2.3 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 28  bytes 3702 (3.6 KiB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

Voici un exemple lorsque les machines sont correctement configurées :

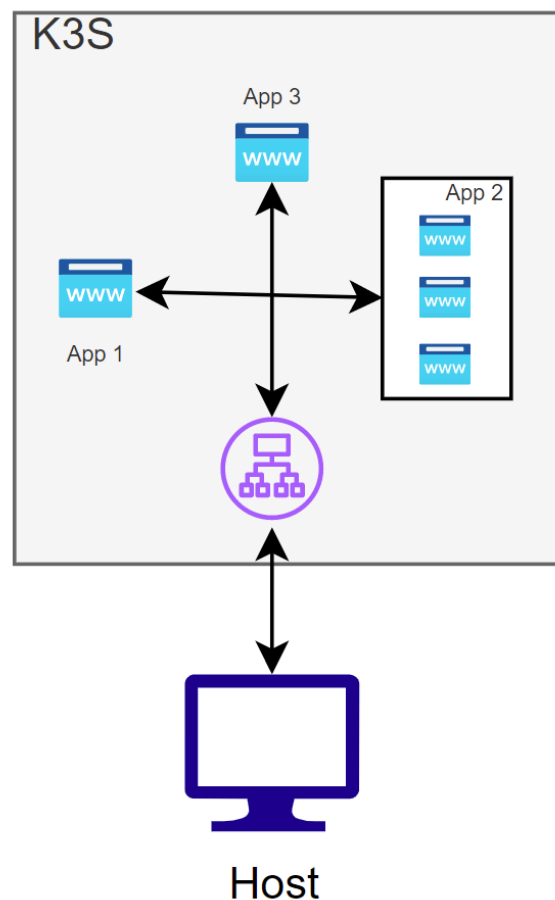
```
[vagrant@wils ~]$ k get nodes -o wide
NAME    STATUS    ROLES    AGE    VERSION    INTERNAL-IP    EXTERNAL-IP    OS-IMAGE    KERNEL-VERSION    CONTAINER-RUNTIME
wils    Ready    control-plane,master    16m    v1.21.4+k3s1    192.168.42.110    <none>    CentOS Linux 8    4.18.0-240.1.1.el8_3.x86_64    containerd://1.4.9-k3s1
wilsw   Ready    <none>    78s    v1.21.4+k3s1    192.168.42.111    <none>    CentOS Linux 8    4.18.0-240.1.1.el8_3.x86_64    containerd://1.4.9-k3s1
[vagrant@wils ~]$
[vagrant@wilSW ~]$ ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.42.111  netmask 255.255.255.0  broadcast 192.168.42.255
    inet6 fe80::a00:27ff:fea8:bc4  prefixlen 64  scopeid 0x20<link>
    ether 08:00:27:a8:bc:b4  txqueuelen 1000  (Ethernet)
    RX packets 446  bytes 322199 (314.6 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 472  bytes 101181 (98.8 KiB)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
[vagrant@wilSW ~]$
```

IV.2 Partie 2 : K3s avec trois applications simples

Désormais, vous comprenez les bases de K3s. Il est temps d'aller plus loin ! Pour réaliser cette partie, vous avez besoin d'une seule machine virtuelle dans laquelle sont installés CentOS (**dernière version stable**) et K3s en mode serveur.

Vous allez mettre en place 3 applications web de votre choix qui tourneront dans votre instance K3s. Il faudra pouvoir accéder à celles-ci en fonction de l'HOST choisi en faisant une requête vers l'adresse IP 192.168.42.110. Le nom de cette machine sera encore une fois un de vos logins suivi de S (par exemple *wilS* si votre login est *wil*).

Voici un petit exemple sous forme de schéma :



Lorsqu'un client entre l'IP 192.168.42.110 dans son navigateur web avec l'HOST *app1.com*, le serveur doit afficher l'app1. S'il utilise l'HOST *app2.com*, le serveur doit afficher l'app2. Sinon, l'app3 sera choisie par défaut.



Comme vous pouvez le constater, il y a 3 répliques pour l'application numéro 2. Il faudra donc adapter votre configuration pour les créer.

Inception-of-Things (IoT)

Dans un premier temps, voici un résultat attendu lorsque la machine virtuelle n'est pas configurée :

```
[vagrant@wils ~]$ k get nodes -o wide
NAME      STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE      KERNEL-VERSION   CONTAINER-RUNTIME
wils      Ready     control-plane,master   14m   v1.21.4+k3s1   192.168.42.110   <none>        CentOS Linux 8   4.18.0-240.1.1.el8_3.x86_64   containerd://1.4.9-k3s1

[vagrant@wils ~]$ k get all -n kube-system
NAME                                     READY   STATUS    RESTARTS   AGE
pod/metrics-server-86cbb8457f-69zx4     0/1     ContainerCreating   0          14m
pod/local-path-provisioner-5ff76fc89d-p7g5b  0/1     ContainerCreating   0          14m
pod/coredns-7448499f4d-jwlpt            0/1     ContainerCreating   0          14m
pod/helm-install-traefik-crd-wkn88       0/1     ContainerCreating   0          14m
pod/helm-install-traefik-82sqz          0/1     ContainerCreating   0          14m

NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/kube-dns    ClusterIP     10.43.0.10   <none>        53/UDP,53/TCP,9153/TCP   14m
service/metrics-server ClusterIP     10.43.89.169 <none>        443/TCP           14m

NAME                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/local-path-provisioner  0/1     1             0          14m
deployment.apps/coredns                  0/1     1             0          14m
deployment.apps/metrics-server           0/1     1             0          14m

NAME                DESIRED   CURRENT   READY   AGE
replicaset.apps/metrics-server-86cbb8457f  1         1         0       14m
replicaset.apps/local-path-provisioner-5ff76fc89d  1         1         0       14m
replicaset.apps/coredns-7448499f4d          1         1         0       14m

NAME                COMPLETIONS   DURATION   AGE
job.batch/helm-install-traefik  0/1           14m       14m
job.batch/helm-install-traefik-crd  0/1           14m       14m
[vagrant@wils ~]$
```

Inception-of-Things (IoT)

Voici un résultat attendu lorsque la machine virtuelle est correctement configurée :

```
[vagrant@wils de]$ k get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/app-two-6bc974bc98-qtjj7       1/1     Running   0           15m
pod/app-one-6fd76fc6f9-9h64n       1/1     Running   0           15m
pod/app-three-688f68bdcc-sm9rt     1/1     Running   0           15m
pod/app-two-6bc974bc98-nzwth       1/1     Running   0           15m
pod/app-two-6bc974bc98-qhp6p       1/1     Running   0           15m

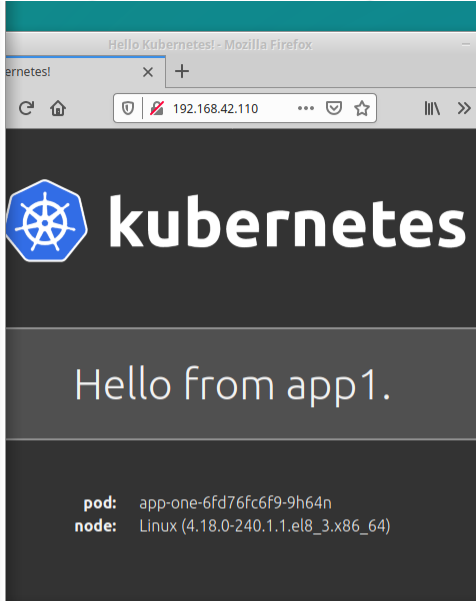
NAME                                TYPE               CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                 ClusterIP          10.43.0.1    <none>         443/TCP    16m
service/app-three                  ClusterIP          10.43.229.156 <none>         80/TCP     15m
service/app-two                    ClusterIP          10.43.193.160 <none>         80/TCP     5m2s
service/app-one                    ClusterIP          10.43.171.213 <none>         80/TCP     4m45s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/app-two             3/3     3             3           15m
deployment.apps/app-three          1/1     1             1           15m
deployment.apps/app-one            1/1     1             1           15m

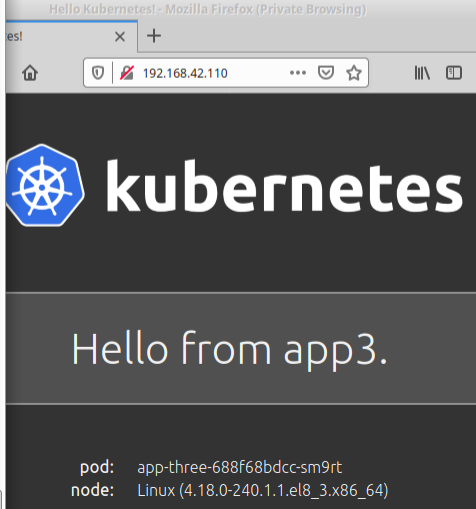
NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/app-one-6fd76fc6f9 1          1         1       15m
replicaset.apps/app-three-688f68bdcc 1          1         1       15m
replicaset.apps/app-two-6bc974bc98   3          3         3       15m
[vagrant@wils de]$ curl -H "Host:app2.com" 192.168.42.110
<!DOCTYPE html>
<html>
<head>
<title>Hello Kubernetes!</title>
<link rel="stylesheet" type="text/css" href="/css/main.css">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Ubuntu:300" >
</head>
<body>

<div class="main">

<div class="content">
<div id="message">
Hello from app2.
</div>
<div id="info">
<table>
<tr>
<th>pod:</th>
<td>app-two-6bc974bc98-qtjj7</td>
</tr>
<tr>
<th>node:</th>
<td>Linux (4.18.0-240.1.1.el8_3.x86_64)</td>
</tr>
</table>
</div>
</div>
</div>
</body>
</html>[vagrant@wils de]$
```



Screenshot of a web browser (Mozilla Firefox) displaying the Kubernetes logo and the text "Hello from app1.". Below the text, it shows the pod name "app-one-6fd76fc6f9-9h64n" and the node "Linux (4.18.0-240.1.1.el8_3.x86_64)".



Another screenshot of a web browser (Mozilla Firefox) displaying the Kubernetes logo and the text "Hello from app3.". Below the text, it shows the pod name "app-three-688f68bdcc-sm9rt" and the node "Linux (4.18.0-240.1.1.el8_3.x86_64)".



L'Ingress n'est volontairement pas affiché ici. Vous devrez l'afficher pendant votre évaluation.

IV.3 Partie 3 : K3d avec Argo CD

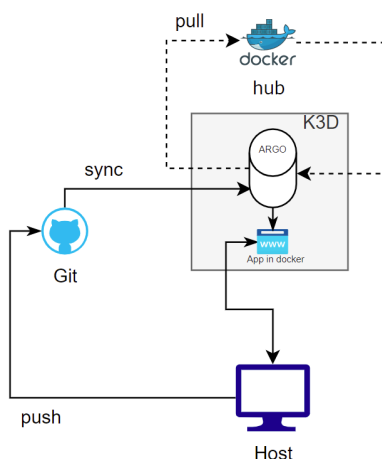
Vous maîtrisez maintenant parfaitement K3s dans sa version minimaliste ! Vous allez pouvoir mettre en place tout ce que vous venez de voir (et bien plus !) cette fois sans Vagrant. Pour cela, installez K3d sur votre machine virtuelle.



Vous aurez besoin de Docker, ceci afin que K3d puisse fonctionner, et probablement aussi d'autres logiciels. Il vous faut donc écrire un script qui installera tout ceci durant votre évaluation.

Pour commencer, il faut bien comprendre la différence entre K3s et K3d.

Une fois votre configuration fonctionnelle, vous allez set up votre première **intégration continue** ! Pour ce faire, vous devez mettre en place une petite infrastructure en suivant la logique du schéma suivant :



Vous devrez créer deux **namespaces** :

- Le premier pour la mise en place du logiciel Argo CD.
- Le second, qui sera appelé *dev*, pour contenir une application. Celle-ci sera automatiquement déployée avec Argo CD en utilisant votre dépôt Github en ligne.



Oui, vous allez devoir créer un dépôt public sur Github où vous ajouterez vos fichiers de configuration. Vous êtes libres de l'organiser comme bon vous semble. La seule obligation est que le login d'un(e) des membres du groupe doit être visible dans le nom de votre dépôt.

L'application qui sera déployée doit être disponible sous **deux versions différentes** (lisez sur les tags si vous ne connaissez pas).

Vous avez le choix entre :

- Utiliser l'application toute prête que Wil a rendu disponible sur Dockerhub.
- Ou coder et utiliser votre propre application. Créez un dépôt Dockerhub public afin d'y push, et donc de rendre disponible, une image de l'application. Ses versions seront taguées ainsi : **v1** et **v2**.



Le dépôt de l'application de Wil sur Dockerhub est :
<https://hub.docker.com/r/wil42/playground>.
L'application utilise le port 8888.
Les deux versions sont visibles dans la rubrique *TAG*.



Si vous choisissez de faire votre propre application, elle doit être disponible grâce à une image Docker publique sur Dockerhub. De plus, les deux versions de votre application devront présenter un minimum de différences.

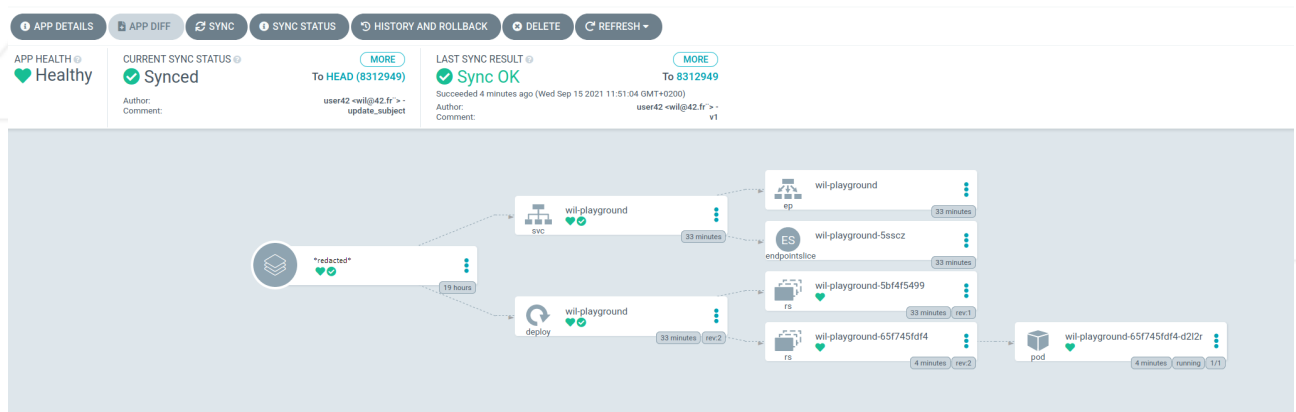
Vous devez pouvoir changer de version de l'application depuis votre dépôt Github public, puis vérifier qu'elle a bien été mise à jour en conséquence.

Voici un exemple de résultat attendu avec les deux **namespaces** et le *POD* du namespace *dev* :

```
$> k get ns
NAME          STATUS  AGE
[.]
argocd        Active  19h
dev           Active  19h
$> k get pods -n dev
NAME                                READY  STATUS   RESTARTS  AGE
wil-playground-65f745fdf4-d2l2r  1/1    Running  0          8m9s
$>
```

Inception-of-Things (IoT)

Voici un exemple du lancement d'Argo CD configuré :



On peut vérifier ici que l'application utilise la version souhaitée (dans ce cas, la **v1**) :

```
$> cat deployment.yaml | grep v1
- image: wil42/playground:v1
$> curl http://localhost:8888/
{"status":"ok", "message": "v1"}
```

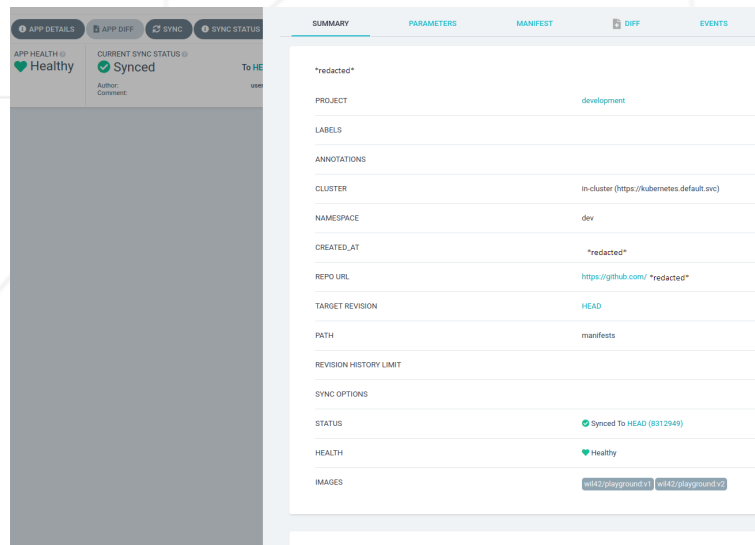
Voici un exemple d'Argo CD avec l'application **v1** utilisant Github :

Ci-dessous, on met à jour le dépôt Github en changeant la version de l'application :

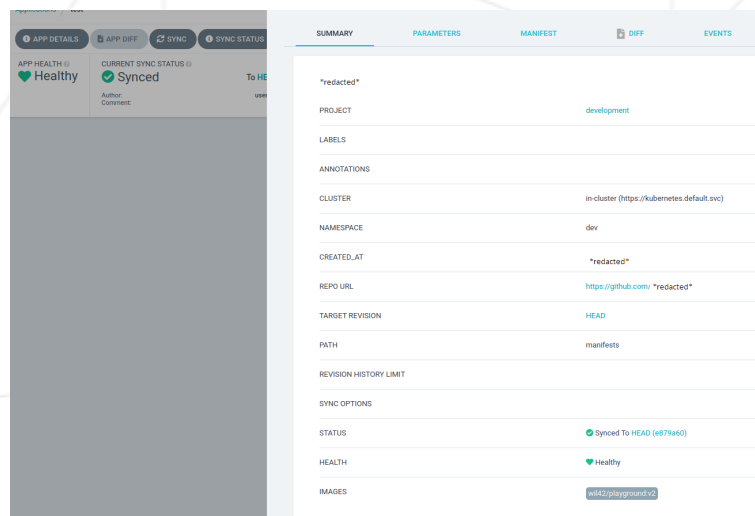
```
$> sed -i 's/wil42/playground\:v1/wil42/playground\:v2/g' deployment.yaml
$> git add+commit+push
[.]
a773f39..999b9fe master -> master
$> cat deployment.yaml | grep v2
- image: wil42/playground:v2
```

Inception-of-Things (IoT)

On peut voir dans Argo CD la synchronisation de l'application :



On remarque que l'application est bien à jour :



On vérifie que la nouvelle version est bien disponible :

```
$> curl http://localhost:8888/  
{ "status": "ok", "message": "v2" }
```



Durant la soutenance, vous serez amené(e)s à refaire cette opération avec l'application que vous aurez choisie : celle de Wil ou la vôtre.

Chapitre V

Partie Bonus

Nous souhaitons que le bonus vous soit utile avant tout : ajoutez **Gitlab** dans votre lab de la troisième partie.



Attention, ce bonus est complexe. La dernière version de Gitlab disponible sur le site officiel est attendue.

Vous avez le droit d'utiliser tout ce qui vous sera nécessaire pour parvenir à ce résultat. Par exemple, **helm** pourrait être utile.

- Votre instance Gitlab devra obligatoirement tourner localement.
- Pour que Gitlab puisse marcher avec votre cluster, il faudra bien entendu le configurer en conséquence.
- Vous devez avoir un **namespace** dédié nommé *gitlab*.
- Tout ce qui était fonctionnel dans la partie 3 du projet doit l'être également avec votre Gitlab local.

Rendez ce bonus à la racine de votre dépôt dans un nouveau dossier nommé **bonus**. Il contiendra tout ce dont vous avez besoin pour que l'intégralité de votre cluster puisse fonctionner.



Les bonus ne seront évalués que si la partie obligatoire est **PARFAITE**. Par parfaite, nous entendons complète et sans aucun dysfonctionnement. Si vous n'avez pas réussi **TOUS** les points de la partie obligatoire, votre partie bonus ne sera pas prise en compte.

Chapitre VI

Rendu et peer-évaluation

Rendez votre travail dans votre dépôt Git comme d'habitude. Seul le travail présent dans votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

En résumé :

- Vous devez rendre à la racine de votre dépôt la partie obligatoire dans trois dossiers : p1, p2 et p3.
- Optionnel : La partie bonus dans un dossier à la racine de votre dépôt : bonus.

Voici un exemple de structure attendue dans votre rendu :

```
$> find -maxdepth 2 -ls
424242  4 drwxr-xr-x  6 wandre wil42    4096 sept. 17 23:42 .
424242  4 drwxr-xr-x  3 wandre wil42    4096 sept. 17 23:42 ./p1
424242  4 -rw-r--r--  1 wandre wil42    XXXX sept. 17 23:42 ./p1/Vagrantfile
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./p1/scripts
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./p1/confs
424242  4 drwxr-xr-x  3 wandre wil42    4096 sept. 17 23:42 ./p2
424242  4 -rw-r--r--  1 wandre wil42    XXXX sept. 17 23:42 ./p2/Vagrantfile
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./p2/scripts
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./p1/confs
424242  4 drwxr-xr-x  3 wandre wil42    4096 sept. 17 23:42 ./p3
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./p3/scripts
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./p3/confs
424242  4 drwxr-xr-x  3 wandre wil42    4096 sept. 17 23:42 ./bonus
424242  4 -rw-r--r--  1 wandre wil42    XXXX sept. 17 23:42 ./bonus/Vagrantfile
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./bonus/scripts
424242  4 drwxr-xr-x  2 wandre wil42    4096 sept. 17 23:42 ./bonus/confs
```



Vous pouvez ajouter les scripts dont vous aurez besoin dans un dossier scripts et les fichiers de configuration dans un dossier confs.



L'évaluation se déroulera sur l'ordinateur du groupe évalué.