

Kubernetes

Un aperçu

C'est quoi ?

- Un orchestrateur. Celui qui, à ce jour, semble avoir gagné la bataille.
- Développé par Google à la base, en 2014, et rapidement passé en open source (été 2015).

« Objets » Kubernetes

Concepts (« Objets »)

- Pods
- Services
- Volumes
- Namespaces
- Secrets
- Deployments

Pods

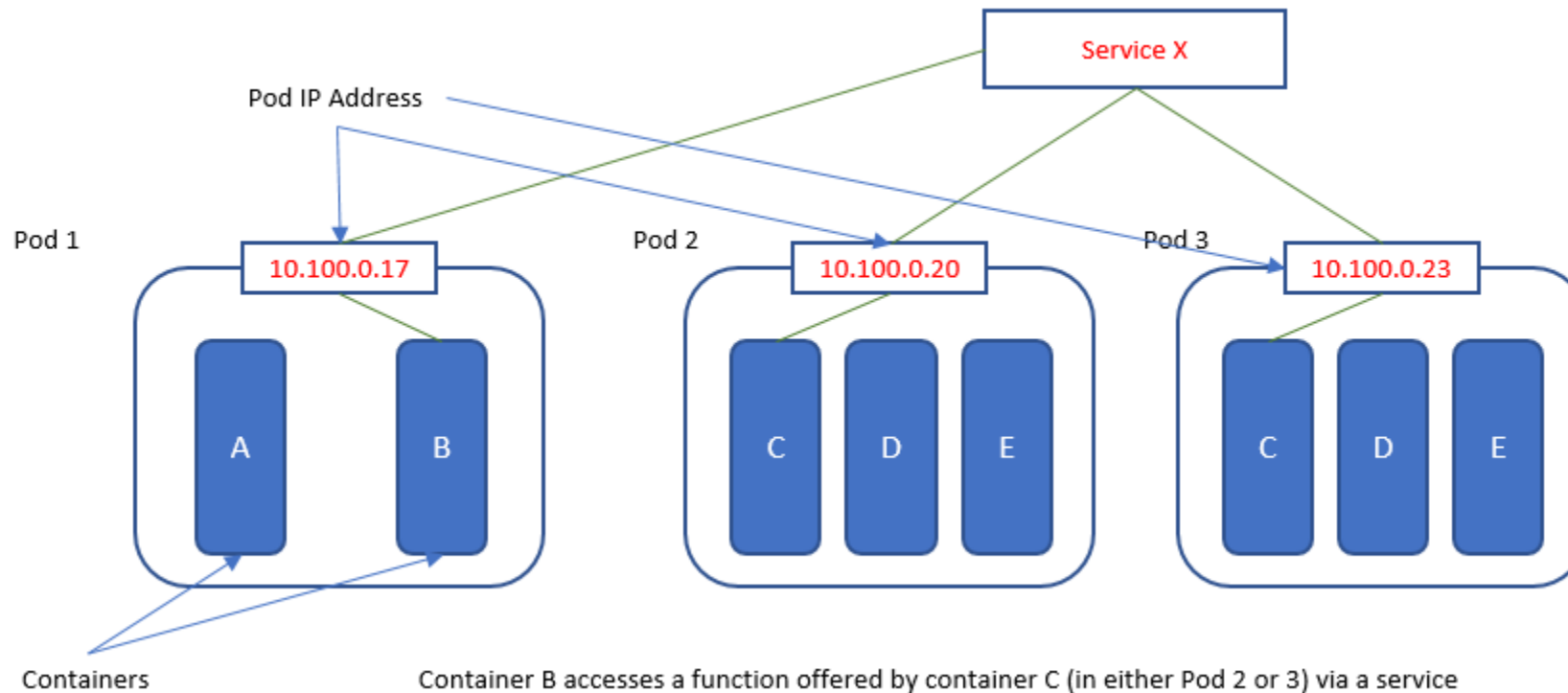
- Une abstraction de haut niveau groupant des composants conteneurisés.
- En clair, un ensemble d'un ou plusieurs conteneurs qui seront localisés sur une machine hôte (un nœud), et qui peuvent partager des ressources.
- Chaque pod a une adresse IP unique dans le *cluster*. À l'intérieur d'un pod, tous les conteneurs peuvent discuter via *localhost*.

Pods, suite

- Pour discuter entre pods, on ne doit jamais utiliser l'adresse IP du pod, celle-ci pouvant changer à tout moment. On doit utiliser des références à des *services*.
- Un pod peut définir un volume, comme un répertoire local ou un disque réseau, et le mettre à disposition des conteneurs du pod.
- On gère les pods à la main, via l'API Kubernetes, ou on délègue ladite gestion à un *contrôleur*.

Services

- Un service Kubernetes est un ensemble de pods qui travaillent ensemble.



Services, suite

- L'ensemble des pods qui constituent un service sont définis via des « labels selectors ». Pour simplifier, on donne un même label à un ensemble d'objets.
- Kubernetes fournit un service de découverte et de routage, en assignant une adresse IP stable et un nom de domaine à un service.
- Par défaut, un service est exposé à l'intérieur du cluster, mais il peut aussi l'être à l'extérieur (pour qu'un client puisse s'y connecter 😊)

Volumes

- Par défaut, le stockage dans les conteneurs est éphémère
- Les volumes Kubernetes offrent un stockage persistant, qui existe durant toute la vie du pod.
- Ce stockage peut être partagé par les conteneurs du pod. Les points de montage d'un même volume peuvent différer d'un conteneur à l'autre.

Namespaces

- Les ressources gérées peuvent être partitionnées dans des ensembles disjoints, appelés des « namespaces » (espaces de nommage).
- Cela permet de partager un même cluster entre plusieurs entités, ou entre plusieurs environnements (production, développement, tests, ...).

Secrets

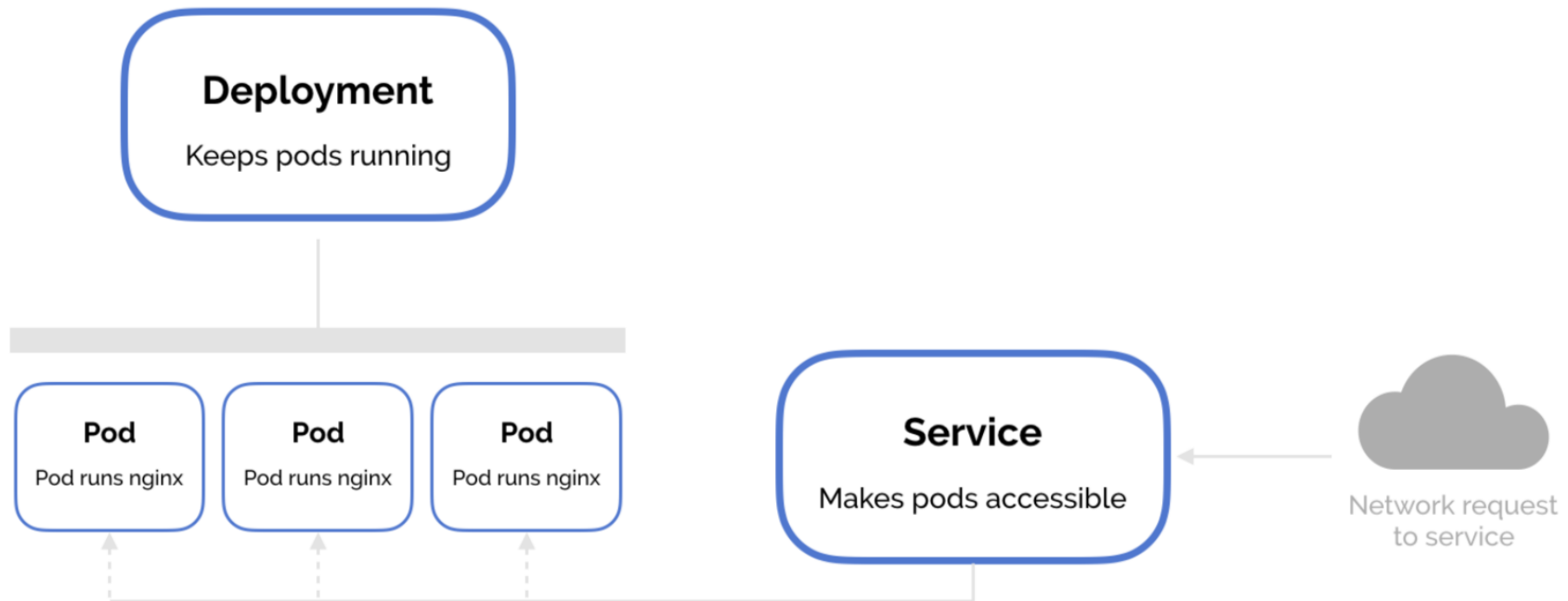
- Où stocker les secrets (mots de passe, clefs SSH, ...) est toujours un problème délicat.
- Kubernetes propose un mécanisme pour les gérer, appelé « secrets » (oui, bon, pas original, mais au moins on comprend 😊).
- Un secret est envoyé à un pod si et seulement si celui-ci le requière.
- Les secrets sont passés via des variables d'environnement ou via le filesystem.

Deployments

- Les déploiements sont constitués d'ensembles de *réplicas* qui contiennent des pods identiques.
- Un ensemble de réplica (« replica set ») contient n pods identiques ($n \geq 1$).
- Les déploiements (la taille du replica set) peuvent être ajustés manuellement ou automatiquement (en fonction de la charge CPU ou mémoire ou ...). Kubernetes va créer ou détruire des pods pour atteindre le nombre de réplicas demandé.

Deployment vs Service

- C'est quoi la différence entre un déploiement et un service ?



Gestion des objets Kubernetes

Labels and selectors

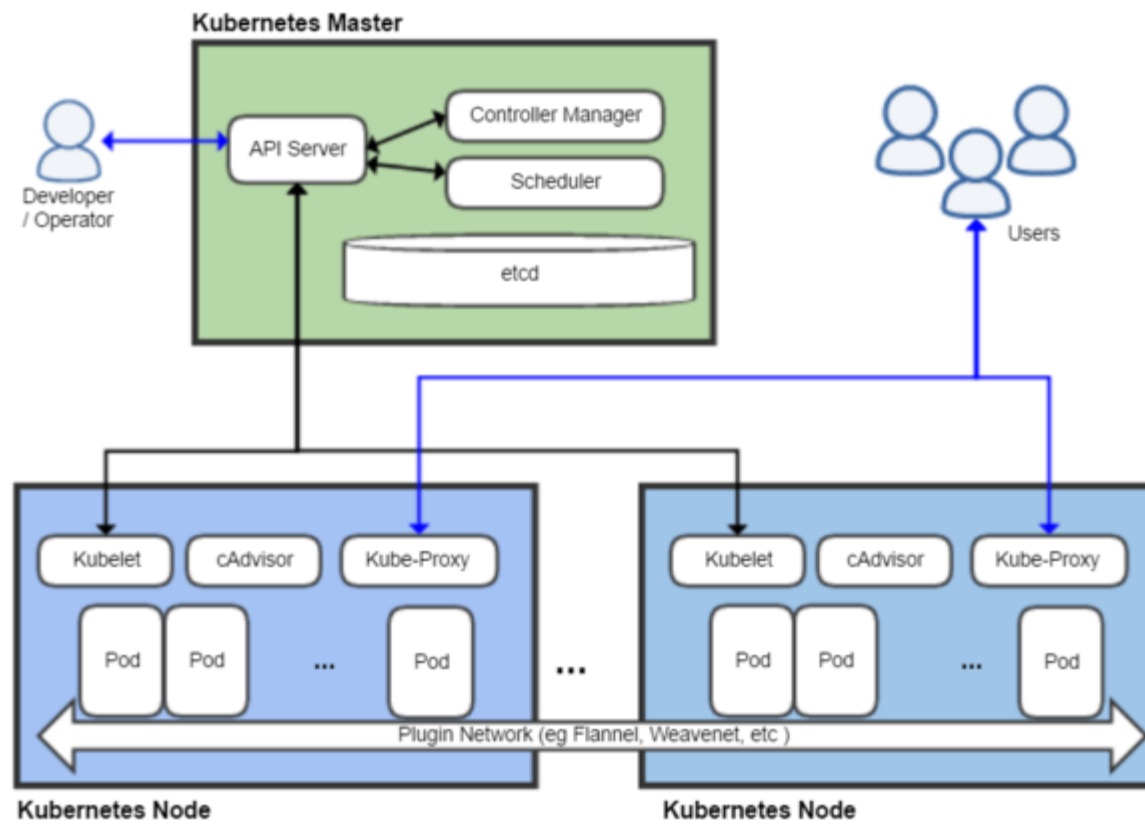
- On peut attacher des labels à tout objet Kubernetes, comme les pods ou les nœuds.
- Les « label selectors » sont des requêtes sur ces labels, qui retournent des ensembles d'objets correspondants.
- Les labels peuvent être attribués et changés tout au long de la vie des objets Kubernetes.

`tier=back-end AND release_track=canary`

Field selectors

- Comme avec les labels, il est possible de définir des filtres basés sur la valeurs de champs des objets.
- Par exemple, tous les objets Kubernetes ont les deux champs suivants :
 - `metadata.name`
 - `metadata.namespace`

Architecture



Minikube

Pourquoi minikube ?

- « a tool that runs a single-node Kubernetes cluster in a virtual machine on your personal computer »
- Parce qu'installer Kubernetes pas en version « minikube », c'est complexe. Cela requière plusieurs machines (virtuelles ou physiques), de bonnes connaissances réseau et, en gros, de bien savoir ce que l'on fait.
- Notre but c'est d'apprendre et d'utiliser, pas d'installer et maintenir 😊

Installation de minikube

- C'est par là :
- <https://kubernetes.io/docs/tasks/tools/install-minikube/>

(En cas de difficultés, s'assurer que le user est bien dans le groupe libvirt (sudo adduser <user> libvirt), et redémarrer le service libvirtd)

On joue

Squash sur Kube !

On a besoin de quoi ? (1/n)

- Un *Deployment* qui exécute la partie PostgreSQL. Ce Deployment contiendra :
 - Un *ReplicaSet* avec une seule instance car on ne souhaite avoir qu'un seul *pod* avec l'image PostgreSQL officielle
 - Un Service de type *ClusterIP* pour lui permettre de communiquer avec le Squash mais pas avec le monde extérieur
 - Un *PersistentVolumeClaim* et son *PersistentVolume* associé pour stocker les données de la base
 - Des *Secrets* pour permet de configurer les mots de passes de PostgreSQL et de l'utilisateur Squash.

On a besoin de quoi ? (2/n)

- Un *Deployment* qui exécute la partie Squash. Ce Deployment contiendra :
 - Un *ReplicaSet* avec une seule instance car on ne fait tourner qu'un seul pod avec l'image Squash
 - Un Service de type *NodePort* (ou mieux si vous avez des *LoadBalancers* ou des *Ingress* à dispo.) qui permettra d'exposer le port interne au cluster Kubernetes vers le monde extérieur
 - Et aussi le Secret défini précédemment pour accéder à la BDD.

On a besoin de quoi ? (/n)

- De rien d'autre 😊

C'est à vous !

- Objectif :
 - Faire ce qui est décrit précédemment, avec minikube et les images officielles de postgres et de squash.
- Moyens :
 - Un ordinateur, avec minikube, et une connexion internet.
- Attendus :
 - Se connecter au service, ça marche.

Hein ?

Je plaisante.

- 😊
- Voir ça posé dans une liste comme ça, ça peut faire peur.
- Alors vous allez me dire : « c'est plus facile de lancer ton DockerCompose ».
- Mais au final, tout ce qui est décrit peut être concaténé dans un seul et même fichier YAML et le résultat est aussi simple dans l'absolu.

On fait ensemble 😊

Les fichiers « manifests »

- Des fichiers YAML, pour définir tous nos objets Kubernetes.
- Toujours le même format, toujours la même structure de base.

```
apiVersion: versioned l'API (v1, ...)
kind: NomDeLaRessource
metadata: # une partie documentaire (nom, labels, ...)
  name: ...
spec:
  ...
```

- Avec le nom qu'on veut. On peut même regrouper plusieurs définitions dans un même fichier YAML.

Les secrets

- Ils sont encodés en BASE64 (pour éviter les conflits de caractères interdits dans les manifests YAML).

```
apiVersion: v1
kind: Secret
metadata:
  name: postgres-credentials
type: Opaque
data:
  user: ZGphbmdv
  password: MWEyNmQxZzI2ZDFnZXNiP2U3ZGVzYj9lN2Q=
```


Les secrets, suite

- Comment je génère les valeurs ?

```
echo -n 'admin' | base64 -w 0
```

```
YWRtaW4=
```

```
echo -n '1f2d1e2e67df' | base64 -w 0
```

```
MWYyZDF1MmU2N2Rm
```

- Par là pour plus de détail :

<https://kubernetes.io/docs/concepts/configuration/secret/>

Les secrets, fin

- On ne sauvegarde pas les secrets dans un système de gestion de code source (SVN, git, ...). Jamais.
- Alors, comment on fait ?
- On ne versionne pas, ou on utilise un générateur, si on a une version récente de Kubernetes (v1.14 ou +).
- Par là pour plus de détails :
<https://kubernetes.io/docs/tasks/manage-kubernetes-objects/kustomization/>

Les volumes

- <https://kubernetes.io/docs/concepts/storage/volumes/>
 - <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>
-
- (Oui, parce que, rappelons-nous, les volumes Kubernetes, par défaut, ne sont « persistants » que pour la durée de vie du pod...)

Les volumes persistants

- Kubernetes est basé sur l'idée d'un couplage faible. La manière dont le volume est mis à disposition par la plate-forme est distinguée de la manière dont il est utilisé par les pods.
- C'est parce que la manière dont le volume est mis à disposition dépend généralement de la plate-forme qui héberge le cluster (*i.e.*, la définition d'un volume sur un cloud publique est probablement différente de la définition d'un volume sur une infrastructure en interne).
- De ce fait, Kubernetes met à disposition deux types de ressources, les `PersistentVolume` et les `PersistentVolumeClaim`.

Les volumes persistants, suite

- Une ressource de type `PersistentVolume` (PV) est un bout de stockage sur le cluster, de la même manière qu'un nœud est soit un serveur physique soit une VM. Elle a un cycle de vie qui diffère de celui d'un pod, et elle précise la manière dont le stockage est implémenté (NFS, un moyen propre à mon cloud public, ...).
- En d'autres termes, le PV dit, « Je vais créer un bloc de 100GB de mémoire sur le cluster, basé sur NFS. Je n'ai aucune idée de qui va l'utiliser, mais qui voudra pourra. »

Les volumes persistants, suite

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: postgres-pv
  labels:
    type: local
spec:
  capacity:
    storage: 2Gi
  storageClassName: standard
  accessModes:
    - ReadWriteMany
  hostPath:
    path: /data/postgres-pv
```

La partie « metadata », c'est de la documentation

« storage », c'est la capacité à créer

« accessModes », c'est comment un pod pourra accéder au volume

« hostPath », indique un volume sur le filesystem de l'hôte. Parfait pour minikube, très probablement pas pour une prod...

Les volumes persistants, fin

- Pour créer le volume :

```
$ kubectl apply -f volume.yaml
```

- Pour obtenir le détail des informations du volume :

```
$ kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON	AGE
postgres-pv	2Gi	RWO	Retain	Bound	default/postgres-pvc	standard		3m

- Par là pour plus de détail :

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

Les « PersistentVolumeClaims »

- Une ressource de type PersistentVolumeClaim (PVC) est une demande d'accès à un volume persistant par un utilisateur. Elle permet à l'utilisateur de consommer une ressource abstraite de stockage sur le cluster
- Les demandes peuvent préciser une taille ou un mode d'accès.
- En d'autres termes, la ressource PVC dit « Je veux 2GB de stockage sur le réseau. Je ne sais pas où il est, ni comment il a été créé, mais je suppose qu'il existe et je veux y accéder ».
- De la sorte, la définition du volume et son utilisation sont séparés.

PersistentVolumeClaim, suite

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: postgres-pvc
  labels:
    type: local
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
  volumeName: postgres-pv
```

PersistentVolumeClaim, fin

- Pour créer la demande d'accès à un volume :
`$ kubectl apply -f postgres/volume_claim.yaml`
- Pour voir notre demande d'accès :
`$ kubectl get pvc`

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
postgres-pvc	Bound	postgres-pv	2Gi	RWO	standard	3m

- Par là pour plus de details :

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims>

Deployment

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres-container
  template:
    metadata:
      labels:
        app: postgres-container
        tier: backend
    spec:
      containers:
        - name: postgres-container
          image: postgres:9.6.6
          env:
            - name: POSTGRES_USER
              valueFrom:
                secretKeyRef:
```

```
secretKeyRef:
```

```
  name: postgres-credentials
  key: user
```

```
- name: POSTGRES_PASSWORD
  valueFrom:
    secretKeyRef:
      name: postgres-credentials
      key: password
```

```
- name: POSTGRES_DB
  value: kubernetes_django
```

```
ports:
```

```
- containerPort: 5432
```

```
volumeMounts:
```


```
- name: postgres-volume-mount
  mountPath: /var/lib/postgresql/data
```

```
volumes:
```


```
- name: postgres-volume-mount
  persistentVolumeClaim:
    claimName: postgres-pvc
```

Service

```
kind: Service
apiVersion: v1
metadata:
  name: postgres-service
spec:
  selector:
    app: postgres-container
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
```



« port », c'est le port exposé par le service (dans le cluster, par défaut)



« targetPort », c'est le port exposé par le ou les pods du service (ceux qui matchent le label selector)

Service, suite

- Par défaut, un service est exposé à l'intérieur du conteneur.
- Ouais, mais ça veut dire quoi ? 😊
- Qu'on peut y accéder ... via son nom, si on est dans le même « namespace ».
- Et, donc, dans l'exemple précédent, on peut y accéder via « postgres-service ». Sur le ou les ports exposés.

Service, suite

- Bon, d'accord.
- Mais, si je suis dans un autre namespace, comment je fais ?
 - Un service « toto » dans le namespace « foo » est accessible via le nom de domaine « toto.foo ». L'entrée DNS « toto.foo » sera résolue en l'adresse IP de cluster pour le service concerné (générée automatiquement ou définie via `spec:clusterIP:`).

Service, suite

- Bon, pff.
- Et si cela ne me convient pas ?
 - Les services sont aussi exposés via des variables d'environnements, dans les conteneurs : {SVCNAME}_SERVICE_HOST et {SVCNAME}_SERVICE_PORT.
 - Par exemple, pour un service « redis-master » qui expose le port 6376 et qui s'est vu allouer l'IP de cluster 10.0.0.11, on a :
 - REDIS_MASTER_SERVICE_HOST=10.0.0.11
 - REDIS_MASTER_SERVICE_PORT=6376
 - (Le nom de service est mis en majuscules et les tirets sont remplacés par '_')

Services, fin

- Ouais, mais si je veux accéder à mon service sous un autre nom, hein, comment je fais, vu que c'est mon problème, là, en fait ?

```
kind: Service
apiVersion: v1
metadata:
  name: service-y
  namespace: namespace-a
spec:
  type: ExternalName
  externalName: service-x.namespace-b.svc.cluster.local
  ports:
    - port: 80
```


Service, fin alternative

- Ou, si je ne veux pas définir un autre service, ou si j'ai un conflit de noms entre deux services, je peux fixer l'adresse cluster IP du service :

```
kind: Service
apiVersion: v1
metadata:
  name: mon-service
spec:
  clusterIP: « x.y.z.y »
  selector:
    foo: bar
  ports:
    - protocol: TCP
      port: 12345
```

Et pour Squash ?

- On utilise les mêmes secrets, donc rien à faire.
- On n'utilise pas de volumes (on pourrait, si on veut conserver les logs de Squash, m'enfin, pas d'intérêt).
- On va donc avoir un déploiement, et un service.
- Le déploiement va beaucoup ressembler à celui de PostgreSQL (sauf les volumes, et les noms des variables d'environnement diffèreront).
- Le service sera de type NodePort, pour qu'on puisse y accéder de l'extérieur du cluster.

Dashboard ... en texte 😊

```
C:\Users\Martin\Kubernetes>kubectl get svc
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes           ClusterIP     10.96.0.1       <none>           443/TCP          7h31m
postgres             ExternalName   <none>          postgres-service.default.svc.cluster.local  5432/TCP         20m
postgres-service     ClusterIP     10.96.253.138   <none>           5432/TCP         22m

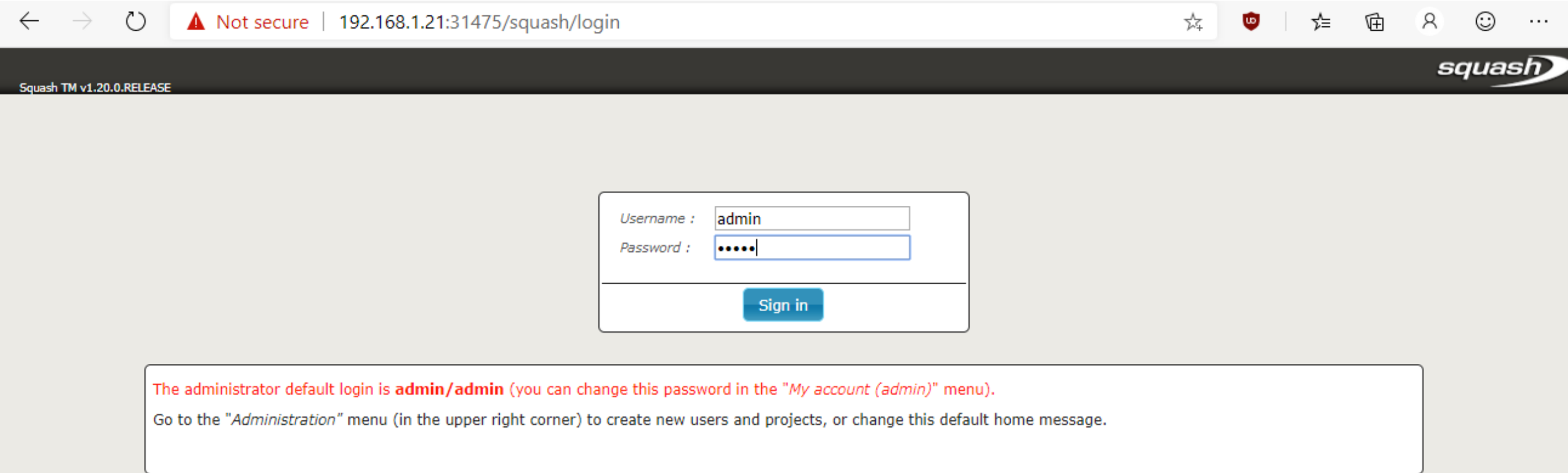
C:\Users\Martin\Kubernetes>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
postgres-7c97cd8fc4-rth5l          1/1     Running   0          23m
squashtm-c755cf7ff-k66f2           1/1     Running   5          12m

C:\Users\Martin\Kubernetes>kubectl get pv
NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                    STORAGECLASS   REASON   AGE
postgres-pv   2Gi       RWX            Retain           Bound    default/postgres-pvc     standard                        25m

C:\Users\Martin\Kubernetes>kubectl get pvc
NAME          STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   AGE
postgres-pvc  Bound    postgres-pv      2Gi       RWX            standard       24m

C:\Users\Martin\Kubernetes>kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
postgres      1/1     1            1           23m
squashtm      1/1     1            1           13m
```

Squash sur minikube sur Windows 😊



The screenshot shows a web browser window with the address bar displaying "192.168.1.21:31475/squash/login". The page has a dark header with the "squash" logo on the right and "Squash TM v1.20.0.RELEASE" on the left. The main content area is light gray and contains a login form. The form has two input fields: "Username :" with the value "admin" and "Password :" with masked characters "•••••". Below the fields is a blue "Sign in" button. A message box at the bottom of the form contains the following text: "The administrator default login is **admin/admin** (you can change this password in the *"My account (admin)"* menu). Go to the *"Administration"* menu (in the upper right corner) to create new users and projects, or change this default home message.

Username : admin

Password : •••••

Sign in

The administrator default login is **admin/admin** (you can change this password in the *"My account (admin)"* menu).
Go to the *"Administration"* menu (in the upper right corner) to create new users and projects, or change this default home message.

- Pour l'IP : kubectl cluster-info, et kubectl get services pour le port.