# Kubernetes
# Manifeste et services

# Objectifs

- Comprendre la notion d'objet et de manifeste au sain de K8S
- Notion d’exposition de services dans k8s

# Notions d’objects

- *Les objets Kubernetes sont des entités persistantes dans le système Kubernetes. Kubernetes utilise ces entités pour représenter l'état de votre cluster. Plus précisément, ils peuvent décrire*
  - *Quelles applications conteneurisées sont en cours d'exécution (et sur quels nœuds).*
  - *Les ressources disponibles pour ces applications*
  - *Les politiques relatives au comportement de ces applications, telles que les politiques de redémarrage, les mises à niveau et la tolérance aux pannes.*
- Un objet Kubernetes est un «enregistrement d'intention », il s’efforce constamment de garantir son existence.

# Spécification et statut de l'objet

- Pour les objets qui ont une **spec**, vous devez la définir lorsque vous créez l'objet, en fournissant **une description des caractéristiques** que vous voulez que la ressource ait : son état **souhaité**.
- L'état décrit **l'état actuel de l'objet**, fourni et mis à jour par le système Kubernetes et ses composants. Le plan de contrôle de Kubernetes gère continuellement et activement l'état réel de chaque objet pour qu'il corresponde à l'état souhaité que vous avez fourni.
- **Par exemple :**
- un Deployment est un objet qui peut représenter une application exécutée sur votre cluster.
- Vous pouvez définir la spécification de déploiement pour indiquer que **vous souhaitez que trois répliques de l'application soient exécutées.** Le système Kubernetes lit la spécification de déploiement et démarre trois instances de l'application souhaitée, en mettant à jour le statut pour qu'il corresponde à votre spécification. Si l'une de ces instances devait échouer (changement de statut), le système Kubernetes répond à la différence entre la spécification et le statut en effectuant une correction - dans ce cas, en démarrant une instance de remplacement.

# Discription des objets à partir des  manifeste

- Champs requis  :
- Dans le fichier .yaml de l'objet Kubernetes que vous souhaitez créer, vous devez définir les valeurs des champs suivants :
  - **apiVersion** - La version de l'API Kubernetes que vous utilisez pour créer cet objet.
  - **kind** - Le type d'objet que vous souhaitez créer.
  - **metadata** - Données qui permettent d'identifier l'objet de manière unique, y compris une chaîne de nom, un UID et un espace de nom facultatif.
  - **spec** - L'état que vous souhaitez pour l'objet.

# Manifeste

- C’est un fichier texte au format YAML (*.yml)
- Un fichier de manifeste contient des objets qui décrivent des ressources par intention
- Entête :
  - `apiVersion` : API et version de l’API à appeler. Souvent seulement `v1`
  - `kind` : Type de d’objet (Pod, service, deployment, namespace, etc.)
  - `metadata` :
    - **`name`** : chaine pour identifier l’objet à créer (obligatoire)
    - `UID` : identifiant unique
    - `namespace` : nom de l’espace où réside l’objet
  - `spec` : spécification de l’objet à créer

https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

# Manifeste

- Par exemple, voici le fichier de configuration d'un Pod qui a deux étiquettes environment : production et app : nginx

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    environment: production
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

# Manifeste Kubernetes – Labels – Déclaration

- Les labels permettent **d'identifier des groupes de ressources**
- Toujours dans l'entête et dans le champ « `metadata` » :
  - `labels` : peut contenir un objet de type « clef/valeur »
- Exemple :

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    environment: production
    app: nginx
[...]
```

JSON

```json
{
   "apiVersion": "v1",
   "kind": "Pod",
   "metadata": {
      "name": "label-demo",
      "labels": {
         "environment": "production",
         "app": "nginx"
      }
   }
}
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

# Manifeste Kubernetes – Labels – Déclaration

- Autres exemples de labels :
  - version : 1.0, 1.1, etc.
  - tier : presentation, service, frontend, backend, bd, etc.
  - environnement / environment : dev, unit, fonct, accept, it, production
  - Etc.

# Manifeste Kubernetes – Annotations

- Les annotations permettent de passer **des informations supplémentaires arbitraires non identifiantes aux objets**. Des clients tels des outilss et bibliothèques peuvent récupérer ces métadonnées. (exemple : fournisseur cloud, proxy, etc.)
- Toujours dans l'entête et dans le champ « `metadata` » :
  - `annotations` : peut contenir un objet de type « clef/valeur »
- Exemple :

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: annotations-demo
  annotations:
    imageregistry: "https://hub.docker.com/"
[...]
```

JSON

```json
{
   "apiVersion": "v1",
   "kind": "Pod",
   "metadata": {
      "name": "annotations-demo",
      "annotations": {
         "imageregistry": "https://hub.docker.com/"
      }
   }
}
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/

# Manifeste Kubernetes – Labels – Sélecteur

- Deux modes de sélection :
  - Basé sur les (in)égalités :
    - Opérateurs : = (et ==) ou !=
      - Avec kubectl, vous pouvez utiliser `-l` suivi d'une requêtes : `'environment=production,tier!=frontend'`

Exemple : kubectl get pods -l environment=production,tier=frontend

      - Dans un fichier YAML/JSON, ajoutez une propriété :
        - « `selector` » pour les ressources de type service
        - « `selector` » suivi de « `matchLabels` » pour les ressources de type Job, Deployment, ReplicaSet, DaemonSet

```yaml
selector:
  matchLabels:
    component: redis
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

# Manifeste Kubernetes – Labels – Sélecteur

- Deux modes de sélection :
  - Basé sur les ensembles :
    - Opérateurs : `in`, `notin`, `exists`
      - Avec kubectl, toujours avec `-l`, suivi de `'environment,environment notin (dev,qa)'`

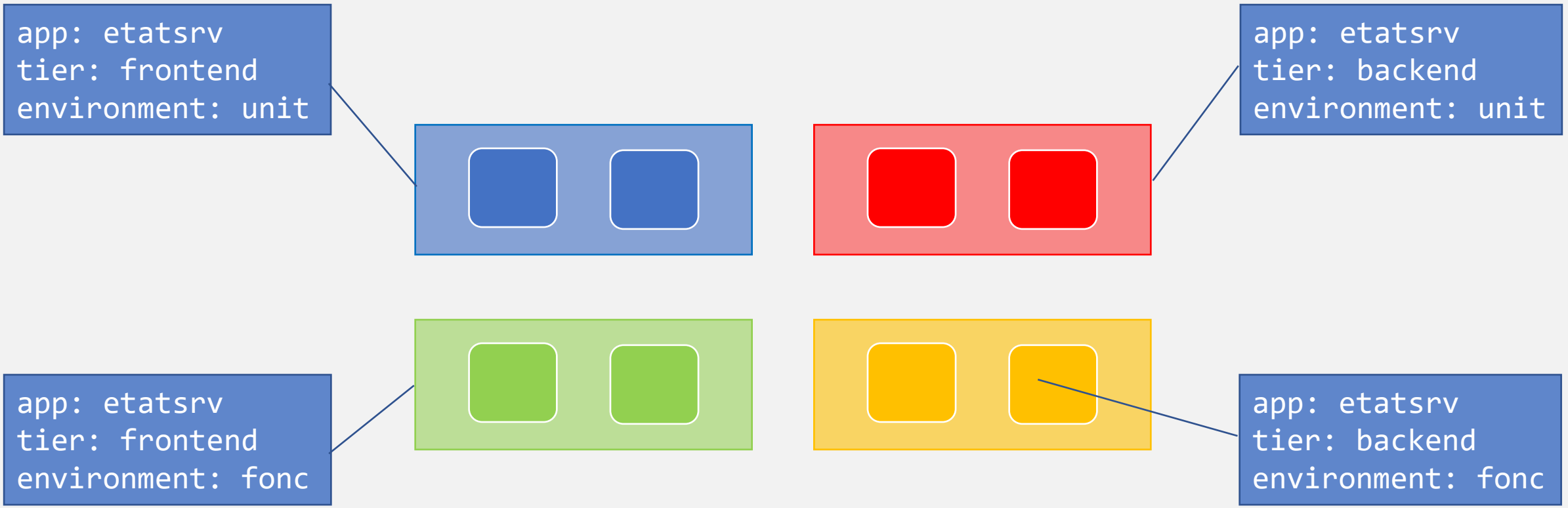
```
kubectl get pods -l 'environment,environment notin (frontend)'
```

      - Dans un fichier YAML/JSON, ajoutez une propriété « `matchExpressions` » dans la propriété « `selector` »

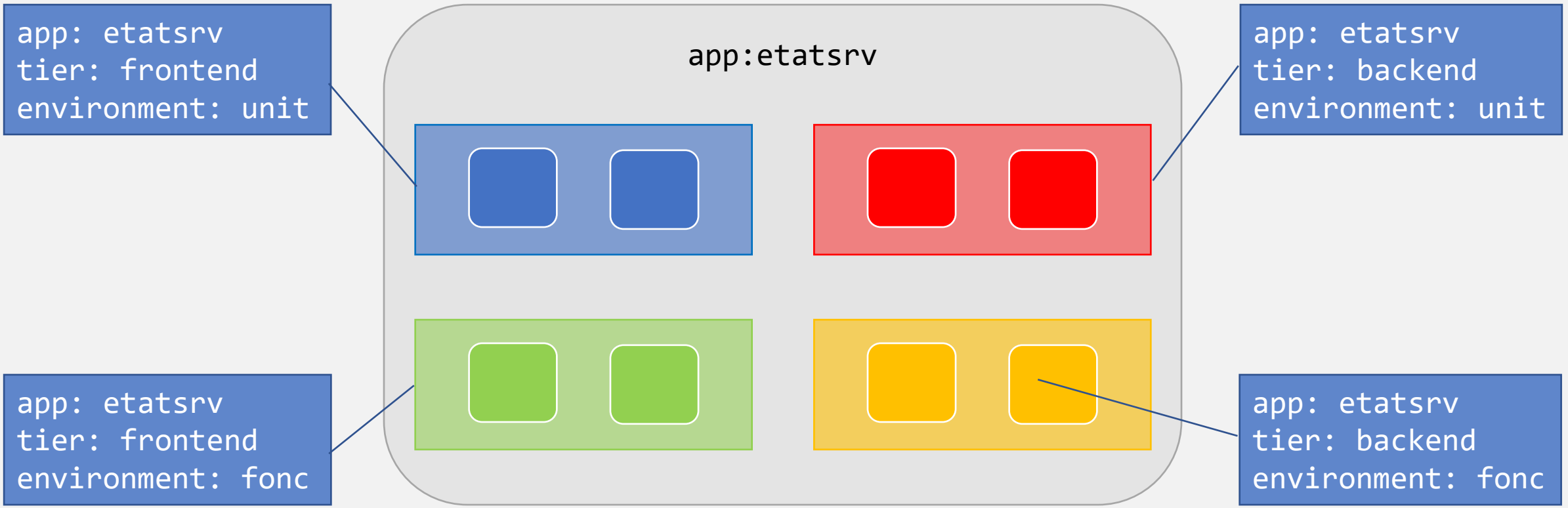
```yaml
selector:
  matchExpressions:
    - {key: tier, operator: In, values: [cache]}
    - {key: environment, operator: NotIn, values: [dev]}
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

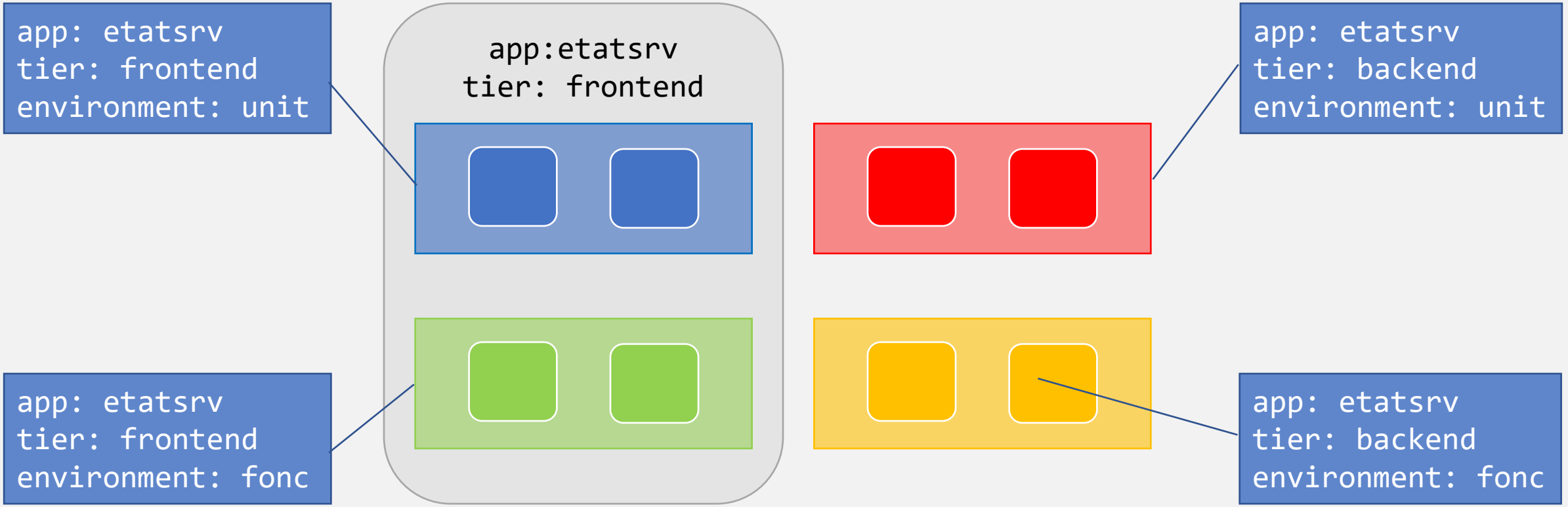
# Manisfeste Kubernetes – Labels – Sélecteur Hypothèse



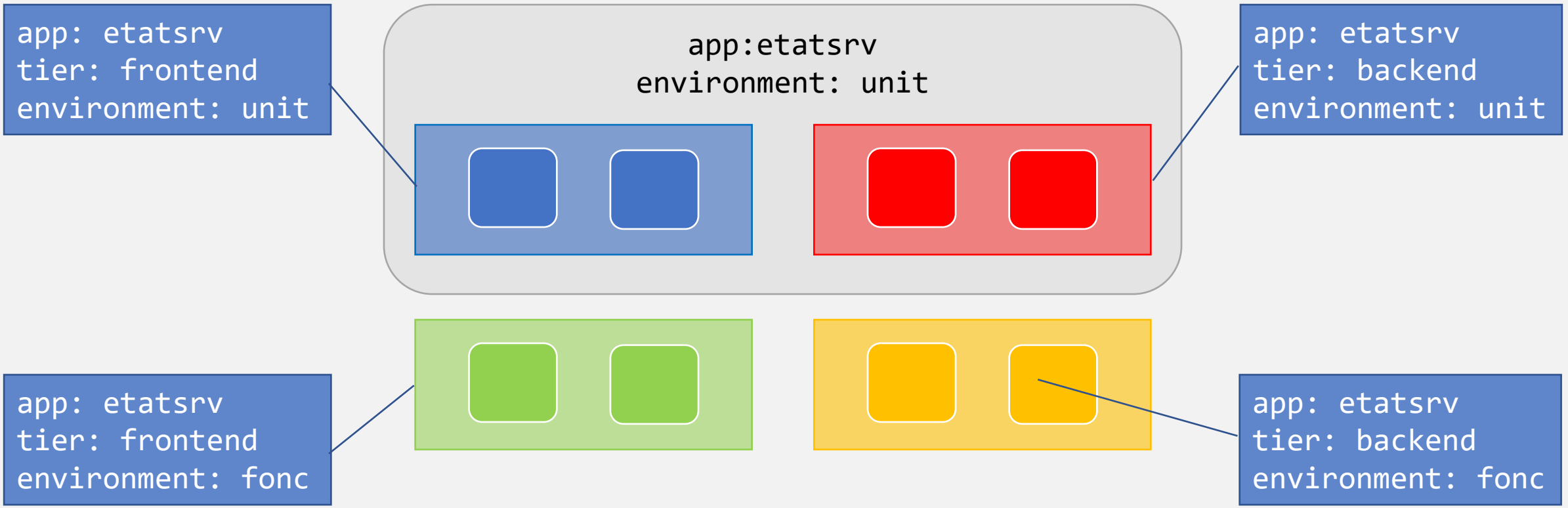
# Manisfeste Kubernetes – Labels – Sélecteur Exemple 1



# Manisfeste Kubernetes – Labels – Sélecteur Exemple 2



# Manisfeste Kubernetes – Labels – Sélecteur Exemple 3



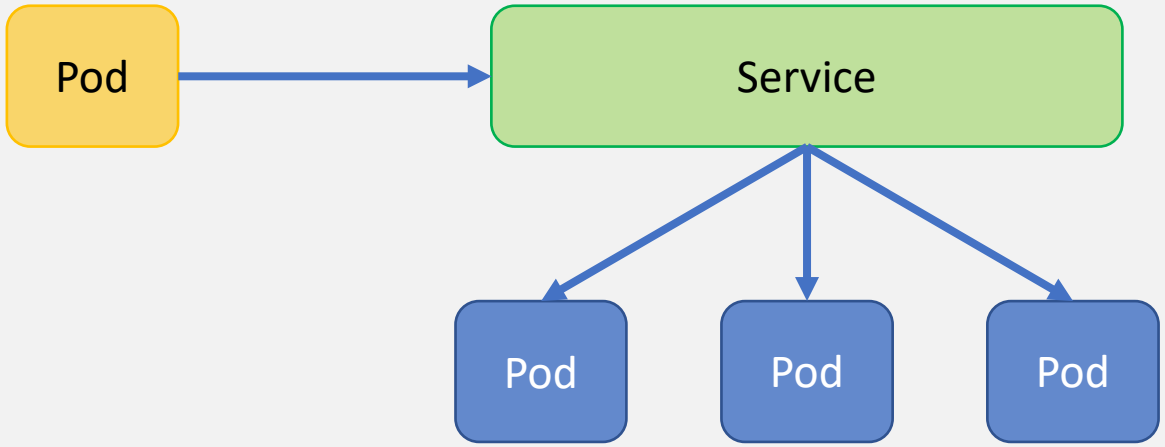
# Manifeste Kubernetes – Labels – Déclaration

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: jpduches

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: jpduches
  labels:
    app: nginx
    user: jpduches
    env: dev
```

```yaml
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      user: jpduches
      env: dev
  template:
    metadata:
      labels:
        app: nginx
        user: jpduches
        env: dev
    spec:
      containers:
      - name: nginx
        image: nginxdemos/hello
        ports:
        - containerPort: 80
```

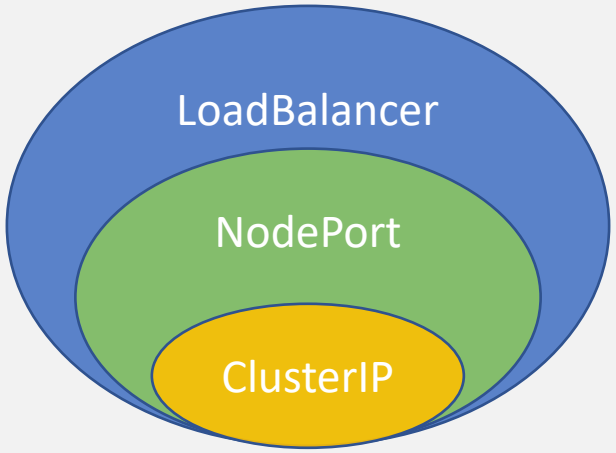
# Services

- Comme pour docker, vous pouvez exposer directement un port d'un Pod sur un nœud
  - Pas pratique si vous avez plusieurs réplicas
  - Utilisation des services
- Les services sont enregistrés par leur nom dans le serveur DNS interne (nom du service = nom d'hote, namespace = domaine)



# Services

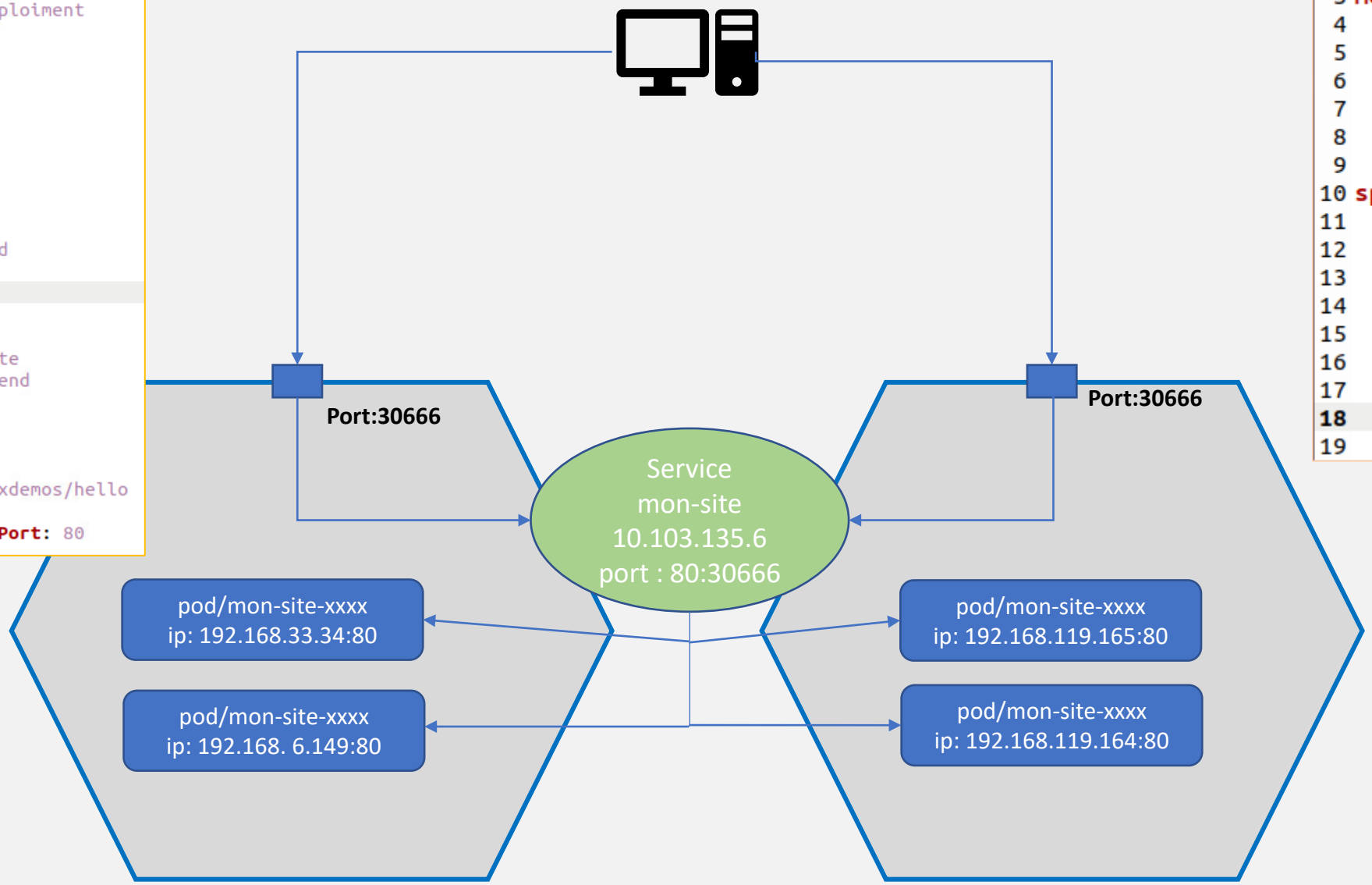
- 3 modes d’exposition :
  - ClusterIP : le service est accessible à l’intérieur de la grappe (cluster) par les pods
  - NodePort : le service est exposé sur l’ensemble des nœuds de calcul. Par défaut les ports disponibles sont à prendre dans 30000 à 32767
  - LoadBalancer : balancer de charge externe sous la responsabilité des équipes TI ou des fournisseurs de services/cloud




https://kubernetes.io/docs/concepts/services-networking/service/

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-site-deploiment
  namespace: jpd
  labels:
    app: mon-site
    tier: frontend
    env : unit
spec:
  replicas: 10
  selector:
    matchLabels:
      app: mon-site
      tier: frontend
      env : unit
  template:
    metadata:
      labels:
        app: mon-site
        tier: frontend
        env : unit
    spec:
      containers:
      - name: nginx
        image: nginxdemos/hello
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mon-site
  namespace: jpd
  labels:
    app: mon-site
    tier: frontend
    env: unit
spec:
  selector:
    app: mon-site
    tier: frontend
    env: unit
  ports:
    - protocol: TCP
      port: 80
      nodePort: 30666
  type: NodePort
```




Node1 : ivk8sc02w01
IP:10.100.2.91

Node2 : ivk8sc02w02
IP:10.100.2.92

NodePort: Expose le service sur l'IP de chaque nœud sur un port statique (le NodePort). Un service ClusterIP, vers lequel le service NodePort est automatiquement créé. Vous pourrez contacter le service NodePort, depuis l'extérieur du cluster, en demandant <NodeIP>: <NodePort>

Lors de l'utilisation du type NodePort vous devez utiliser la plage **30000 à 32767**.

```
jpduches@VM-DevOpsJPD:~/k8s/exer15part3$ kubectl get all -o wide
NAME                                      READY   STATUS    RESTARTS   AGE   IP               NODE          NOMINATED NODE
pod/mon-site-deploiment-78769cc56f-6fxqk  1/1     Running   0          71m   192.168.33.34    ivk8sc02w09   <none>
pod/mon-site-deploiment-78769cc56f-c57hc  1/1     Running   0          71m   192.168.24.49    ivk8sc02w04   <none>
pod/mon-site-deploiment-78769cc56f-dck2m  1/1     Running   0          71m   192.168.6.149    ivk8sc02w01   <none>
pod/mon-site-deploiment-78769cc56f-fbbfm  1/1     Running   0          71m   192.168.102.13   ivk8sc02w03   <none>
pod/mon-site-deploiment-78769cc56f-gzx6m  1/1     Running   0          71m   192.168.6.150    ivk8sc02w01   <none>
pod/mon-site-deploiment-78769cc56f-m6dqm  1/1     Running   0          71m   192.168.119.165  ivk8sc02w02   <none>
pod/mon-site-deploiment-78769cc56f-ptnv9  1/1     Running   0          71m   192.168.33.33    ivk8sc02w09   <none>
pod/mon-site-deploiment-78769cc56f-txjlm  1/1     Running   0          71m   192.168.24.47    ivk8sc02w04   <none>
pod/mon-site-deploiment-78769cc56f-vlpxt  1/1     Running   0          71m   192.168.119.164  ivk8sc02w02   <none>
pod/mon-site-deploiment-78769cc56f-vsp79  1/1     Running   0          71m   192.168.24.48    ivk8sc02w04   <none>
pod/monbash                               1/1     Running   0          97m   192.168.119.227  ivk8sc02w05   <none>

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE   SELECTOR
service/mon-site     NodePort    10.103.135.6    <none>        80:30666/TCP     71m   app=mon-site,env=unit,tier=frontend

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES              SELECTOR
deployment.apps/mon-site-deploiment     10/10   10           10          71m   nginx        nginxdemos/hello    app=mon-site,en
```