



Kubernetes Manifest / Réseau

Objectifs

- Comprendre la notion de manifeste
- Notion d'exposition de services dans k8s

Manifeste Kubernetes

- C'est un fichier texte au format YAML (*.yaml)
- 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 déploiement (Pod, service, deployment, namespace, etc.)
 - `metadata` :
 - `name` : chaîne pour identifier l'objet à créer
 - `UID` : identifiant unique
 - `namespace` : nom de l'espace où réside l'objet
 - `spec` : spécification de l'objet à créer

Manifeste Kubernetes – Annotations

- Les annotations permettent de passer **des informations supplémentaires à certains outils et bibliothèques** (exemple : fournisseur cloud, proxy, etc.)
- Toujours dans l'entête et dans le champ « metadata » :
 - annotations : peut contenir un objet de type « clef/valeur »
- Exemple :

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

YAML

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

JSON

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 :

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

YAML

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

JSON

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 – Labels – Sélecteur

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

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

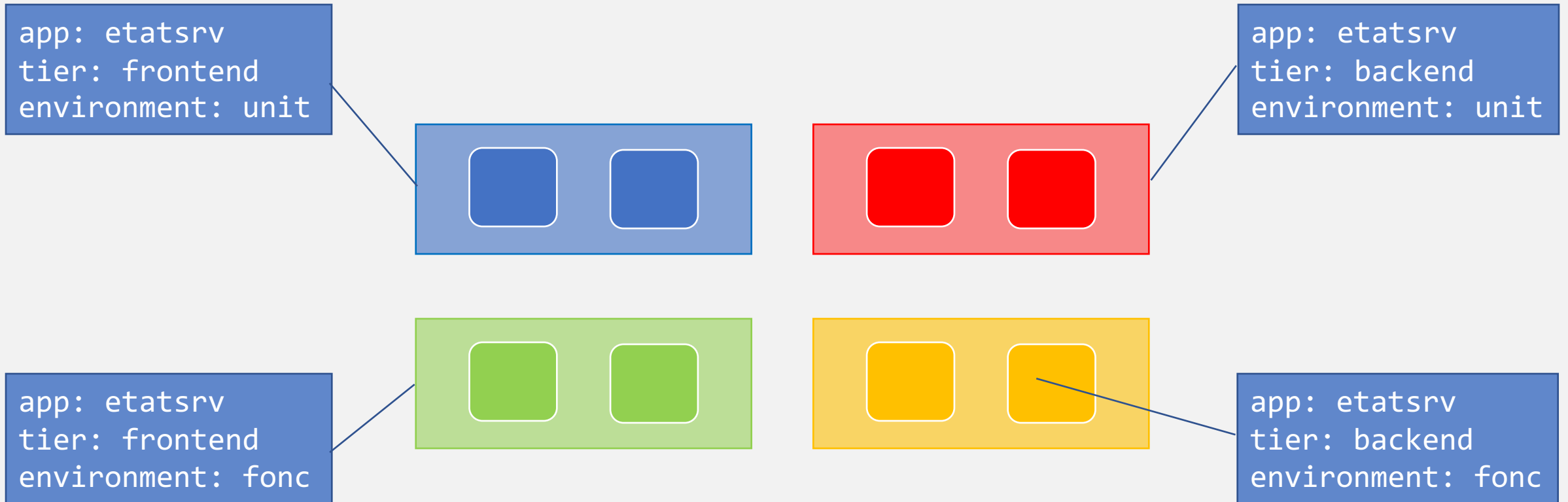
Manifeste Kubernetes – Labels – Sélecteur

- Deux modes de sélection :
 - Basé sur les ensembles :
 - Opérateurs : `in`, `notin`, `exists`
 - Avec `kubectl`, toujours avec `-l`, suivi de `'environment,environment notin (dev,qa)'`
 - Dans un fichier YAML/JSON, ajoutez une propriété « `matchExpressions` » dans la propriété « `selector` »

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

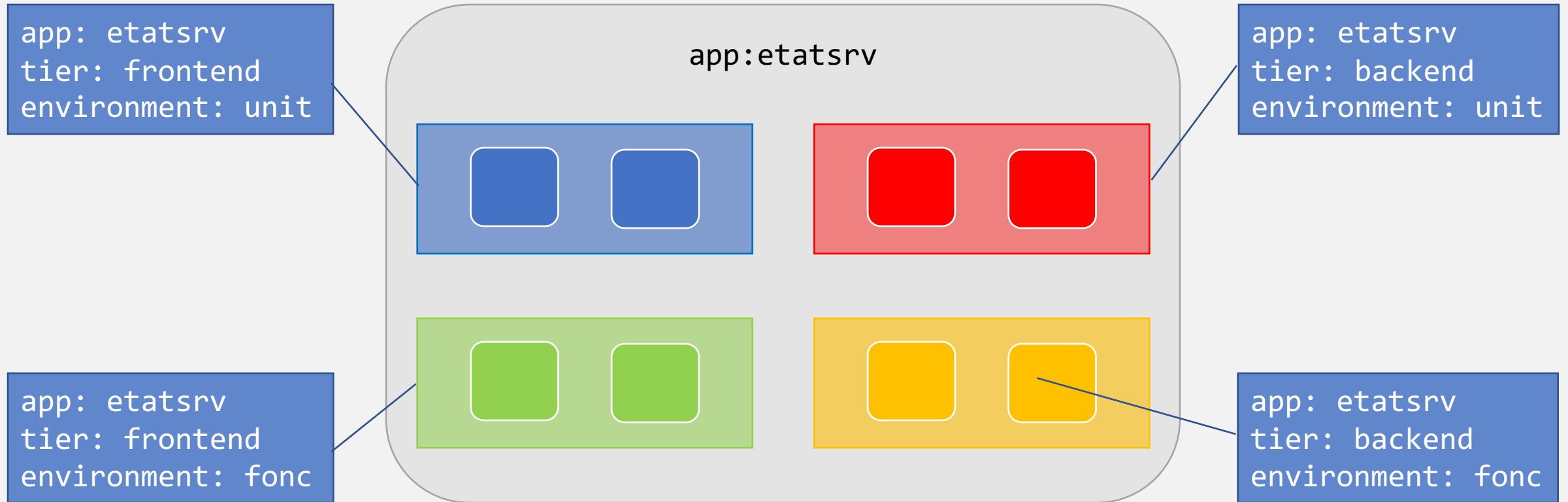

Manifeste Kubernetes – Labels – Sélecteur

Hypothèse



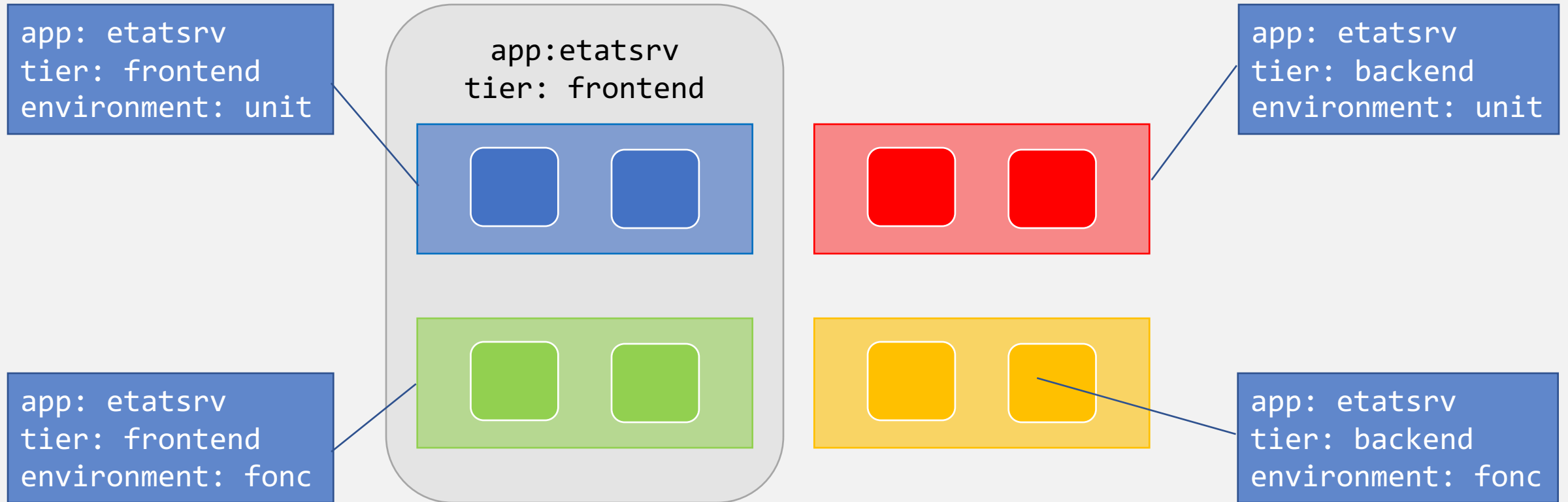
Manifeste Kubernetes – Labels – Sélecteur

Exemple 1



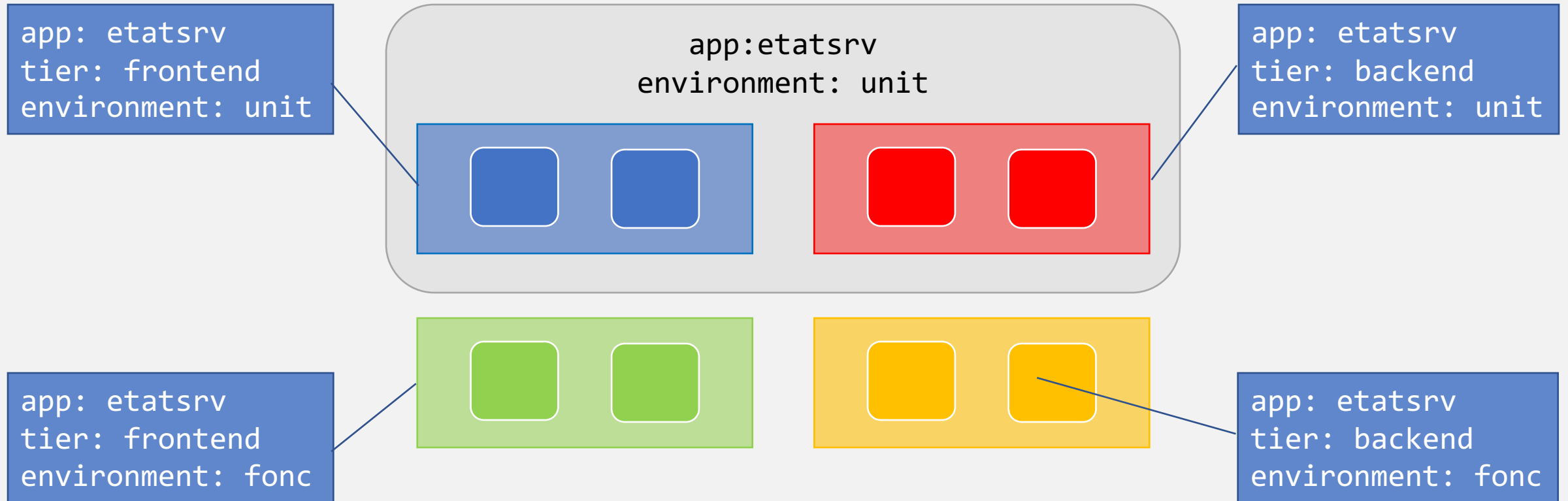
Manifeste Kubernetes – Labels – Sélecteur

Exemple 2



Manifeste Kubernetes – Labels – Sélecteur

Exemple 3



Manifeste Kubernetes – Labels – Déclaration

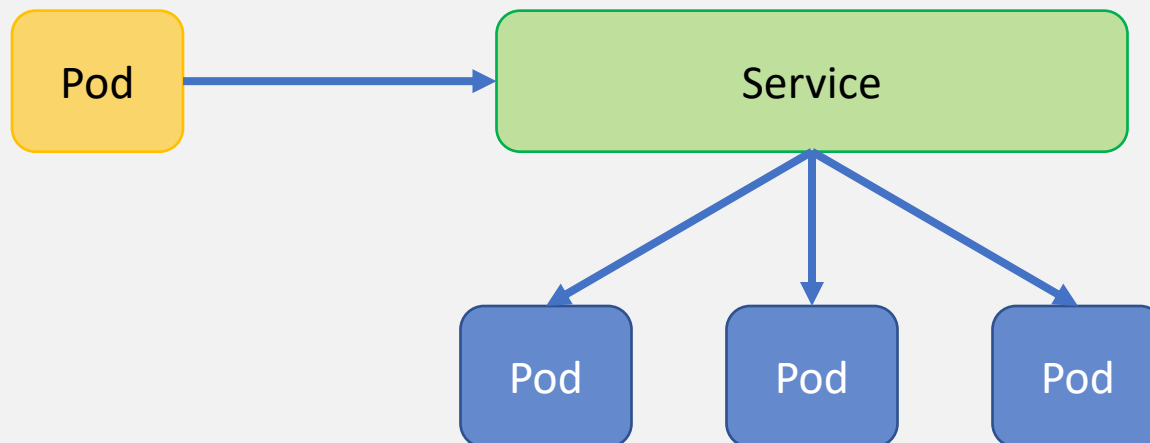
```
apiVersion: v1
kind: Namespace
metadata:
  name: pfleon
```

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

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

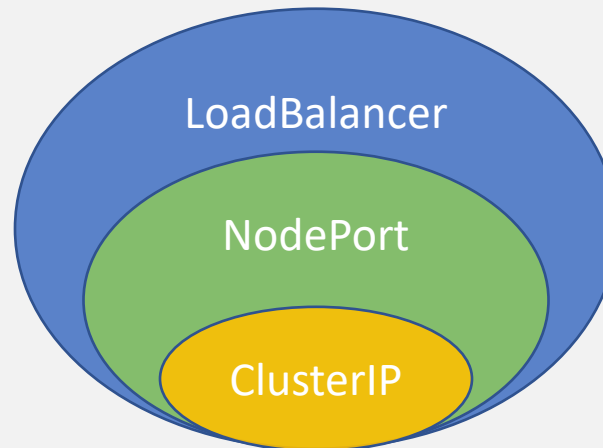
Services

- 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'hôte, 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



Services - Déclaration

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: pfleon
  labels:
    app: nginx
    user: pfleon
    env: dev
spec:
  selector:
    app: nginx
    user: pfleon
    env: dev
  ports:
    - protocol: TCP
      port: 80
      nodePort: 30100
  type: NodePort
```