Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules

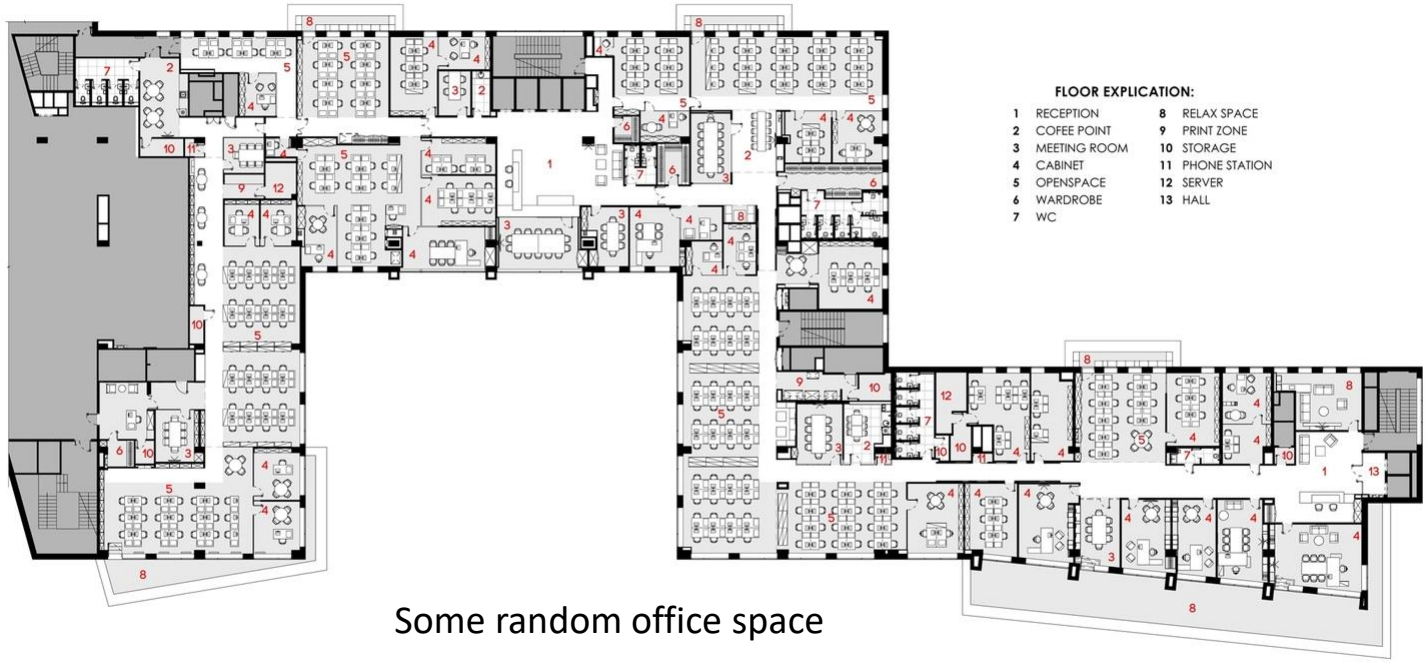
Introduction à Kubernetes

Benjamin Guillon

Principes de base, architecture, composants, ressources …

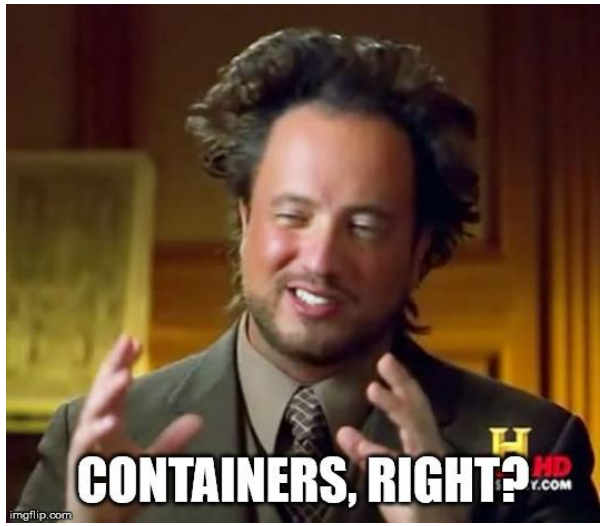
Le plan

- **La base: les conteneurs**
- **Kubernetes**
  - →Principes, architecture, composants
  - →Ressources
  - →Intégration
- **Pour aller plus loin …**



Some random office space

Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules



Partie I - Conteneurs

Faire du neuf avec du vieux

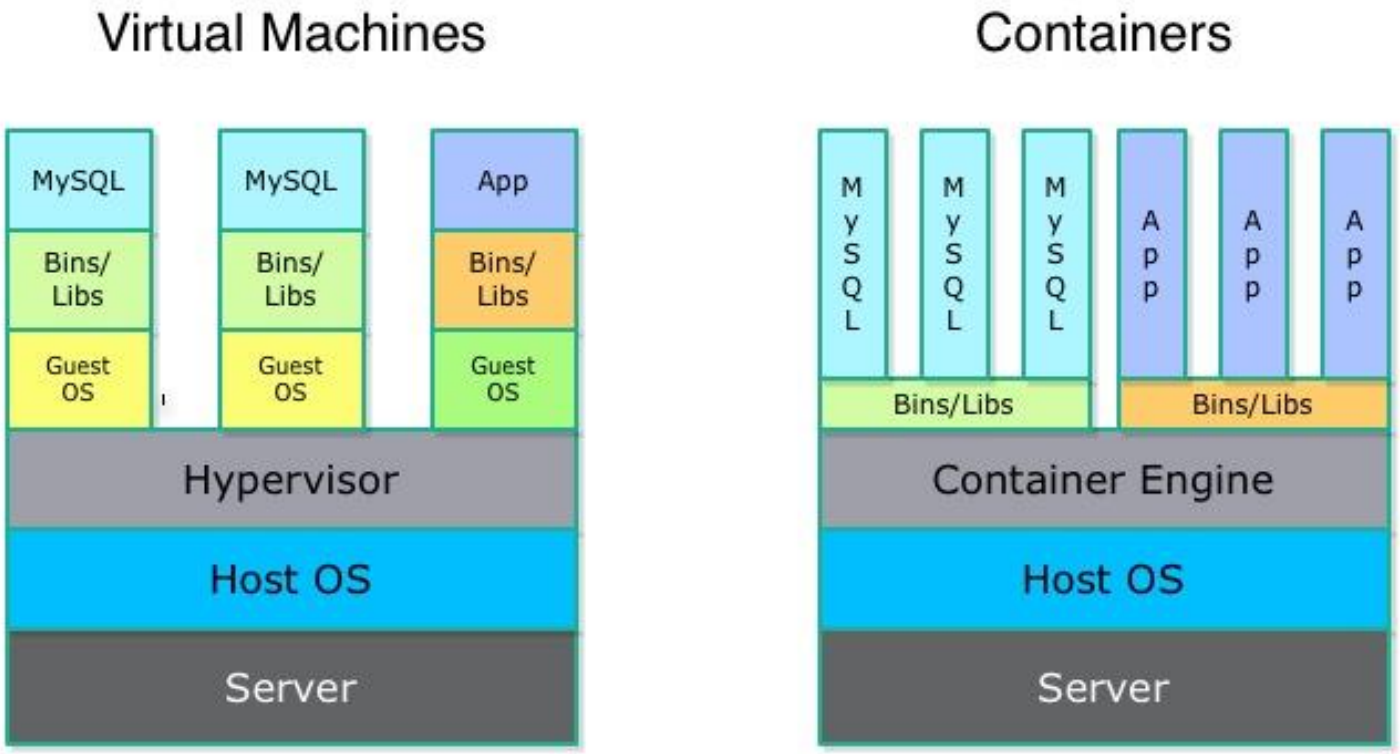
Virtualisation légère?

Ce n’est pas vraiment de la virtualisation …

On n’émule pas de matériel

- On n’a pas de système d’exploitation invité
- On utilise directement le kernel de l’hôte

C’est juste un processus ☺

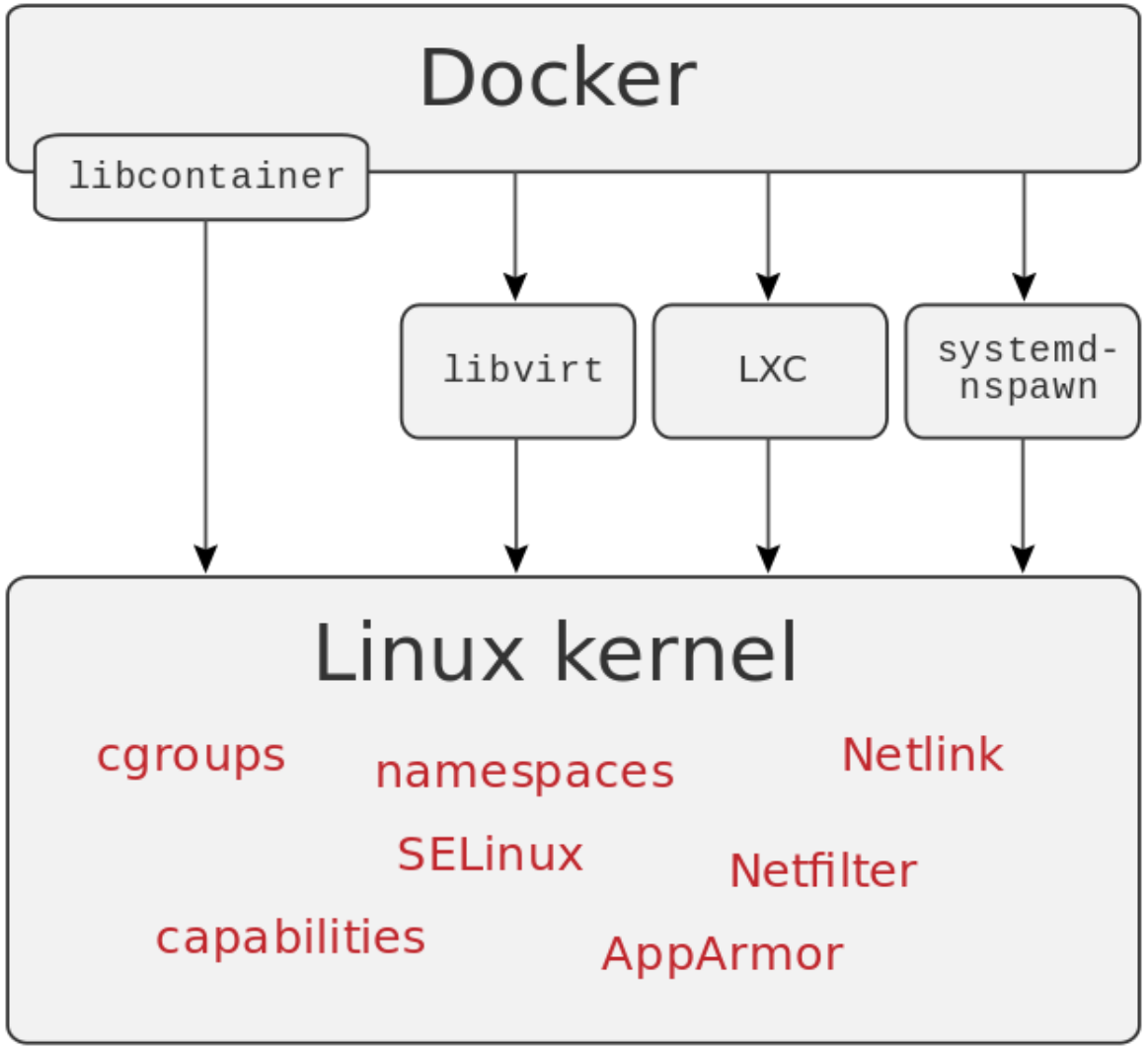


Quelques fonctionnalités présentes dans le kernel

- Cgroups: qu'est-ce que mon processus peut consommer ?
- Namespaces: qu'est-ce que mon processus peut voir et donc faire?

Et tout un tas d'autres choses:

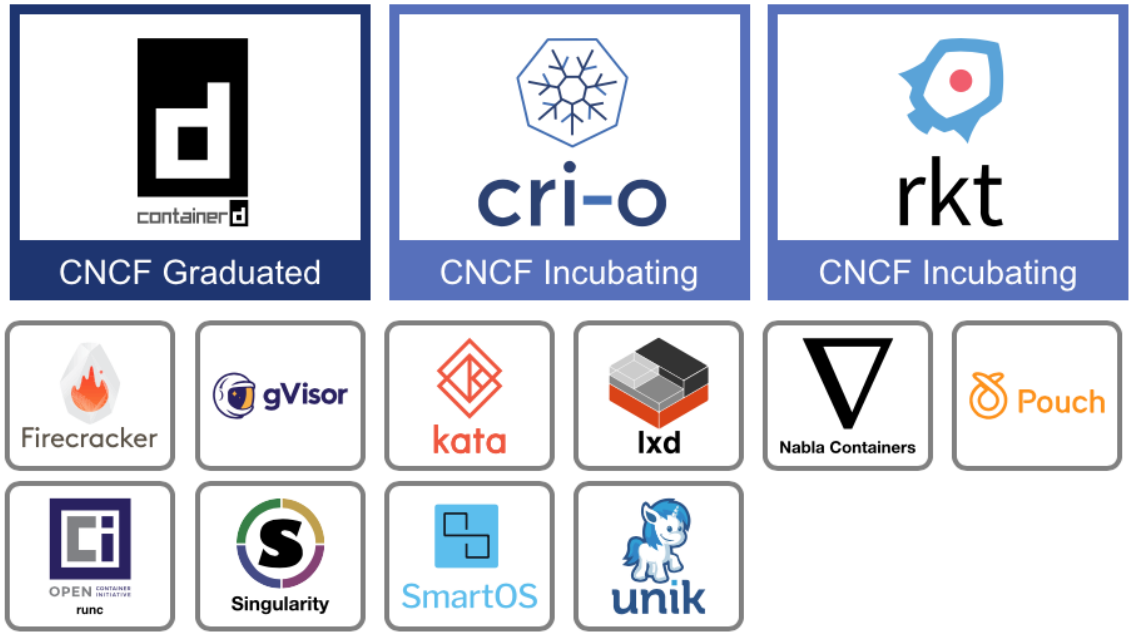
- Capabilities: gestion fine des permissions
- SELinux/AppArmor: tout contrôler
- Netlink/Netfilter: communication breakdown



Quel container runtime utiliser ?

Il en existe quand même pas mal …

© Sascha Grunert

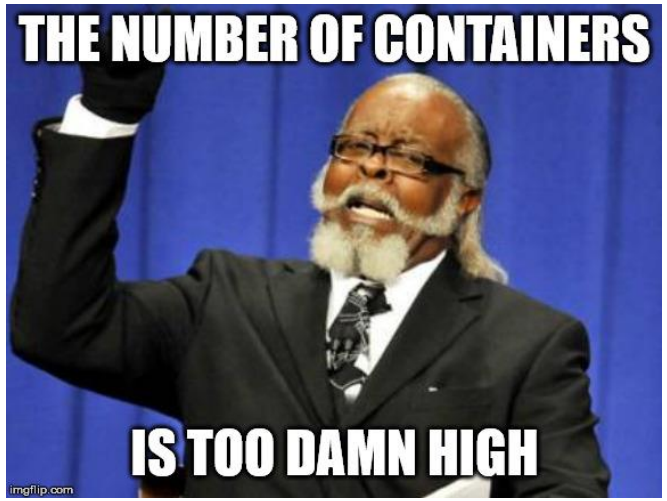


En général: Docker, qui repose sur containerd.

A minima, il faut que le runtime soit OCI compliant.

OPEN CONTAINER INITIATIVE

Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules



Partie II – Kubernetes

Les principes de base

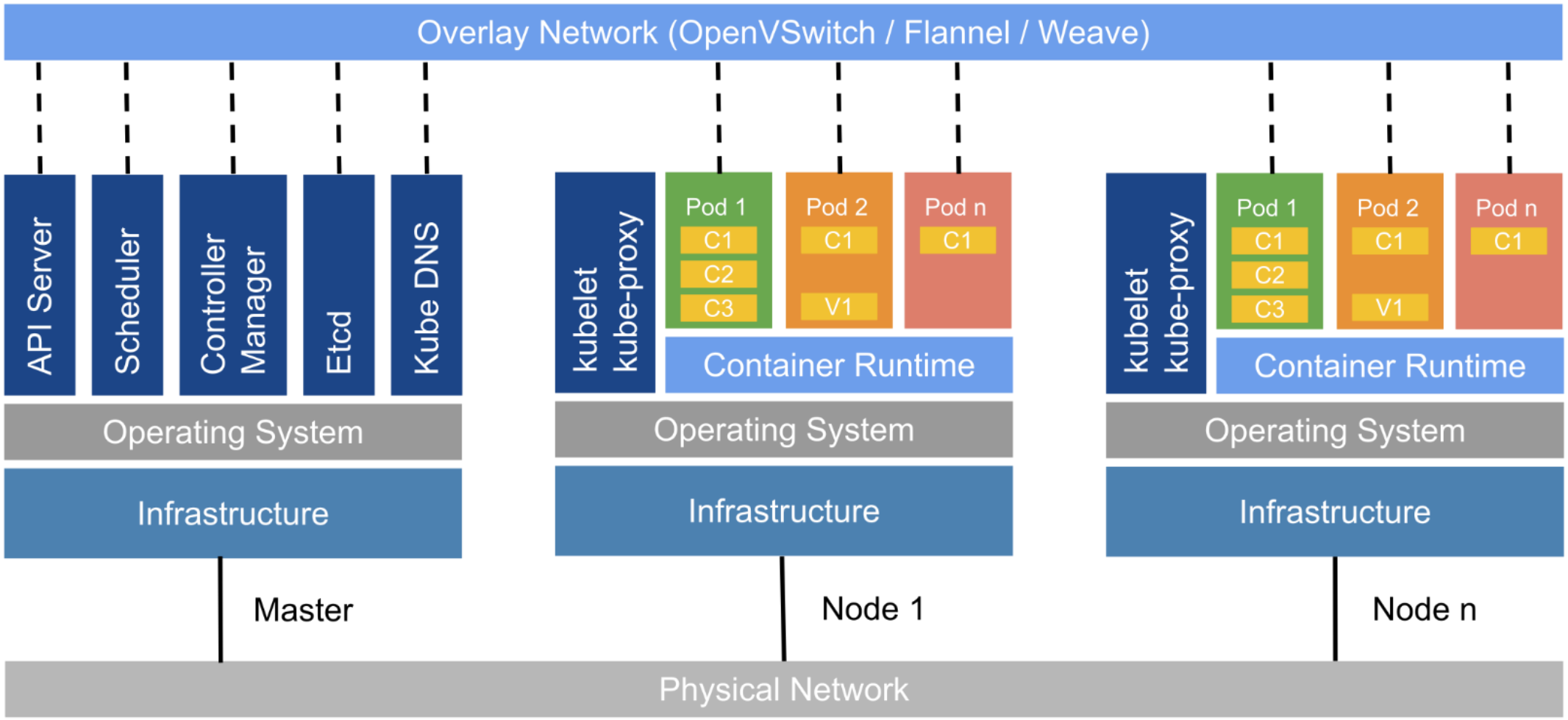
Y a-t-il un pilote dans l'avion?

- **Créé par Google et rendu public en 2015**
  - →Issu de Borg, qui gère les datacenters de Google dès 2004
  - →Partenariat avec la Linux Foundation
    - La Cloud Native Computing Foundation est née

- **Automatisation du cycle de vie de conteneurs**
  - →Déploiement
  - →Configuration
  - →Dimensionnement

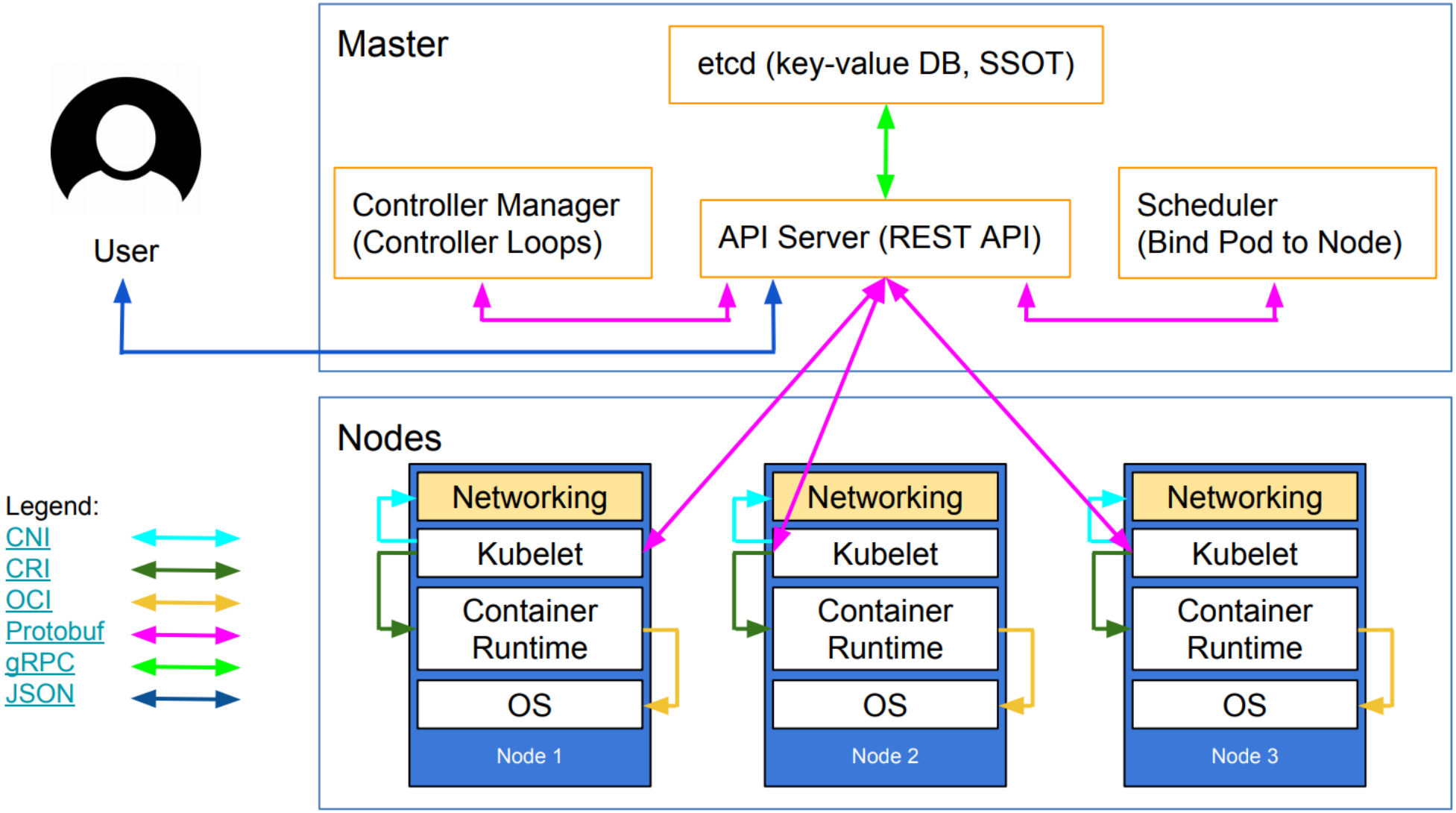
- **Framework extensible**
  - →Pourrait permettre de gérer autre chose que des conteneurs ?

Architecture d'une plateforme K8s



© Imesh Gunaratne

« Piece of cake! » said no one, ever.



Plein de solutions …

Local, une seule machine

Minikube

« On Premise »

heptio

Dans ton « Cloud »

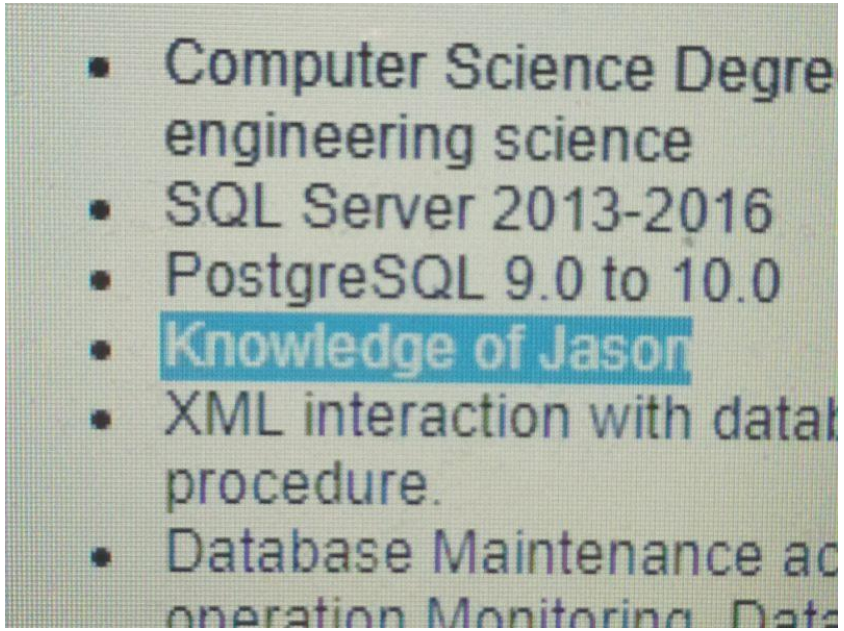
Google
Amazon
Microsoft
IBM
Redhat
VMWare
HP
…

« Managed » ou pas …

CLOUD NATIVE COMPUTING FOUNDATION

Si vous aimez le yaml vous allez être servis…

```
apiVersion: "v1"
kind: "PersistentVolumeClaim"
metadata:
  name: "prometheus-data"
spec:
  accessModes:
    - "ReadWriteOnce"
  resources:
    requests:
      storage: 5Gi
  storageClassName: regular
```



… également disponible en JSON.

Tous les « objets » manipulés sont définis dans les APIs.

On peut aller lire leurs définitions :
https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/

Ou pas …

```
$ kubectl explain pod.spec
KIND:     Pod
VERSION:  v1

RESOURCE: spec <Object>

DESCRIPTION:
     Specification of the desired behavior of the pod. More info:
     https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-status

     PodSpec is a description of a pod.

FIELDS:
   activeDeadlineSeconds        <integer>
     Optional duration in seconds the pod may be active on the node relative to
     StartTime before the system will actively try to mark it failed and kill
     associated containers. Value must be a positive integer.

   affinity     <Object>
     If specified, the pod's scheduling constraints  […]
```

Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules

Partie III – Kubernetes

Ressources

Plusieurs façons de créer des ressources:

```
$ kubectl run web --image=nginx
```

```
$ kubectl create deployment web2 --image=nginx
```

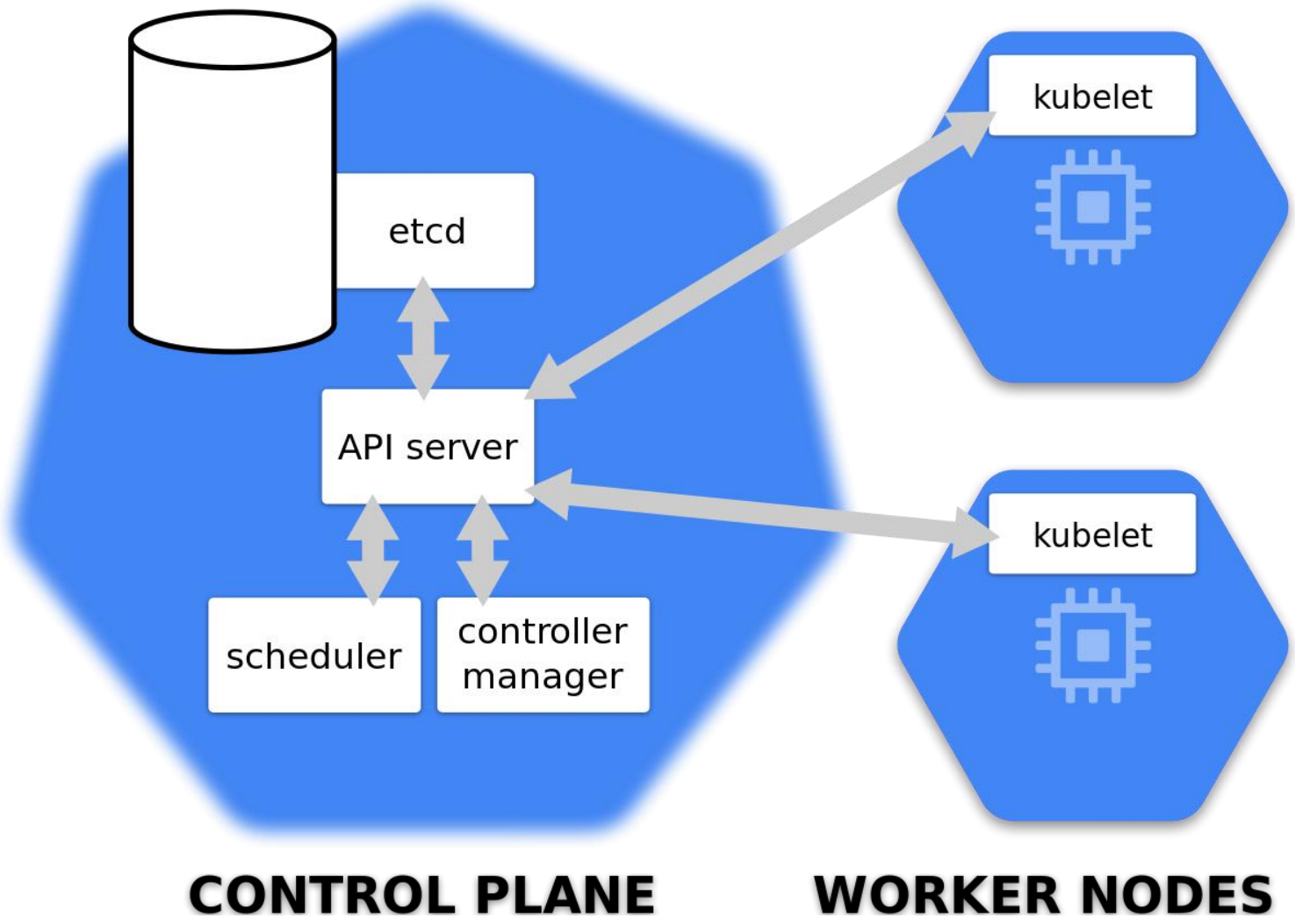
```
$ cat web3.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: web3
  name: web3
spec:
  selector:
    matchLabels:
      app: web3
  template:
    metadata:
      labels:
        app: web3
    spec:
      containers:
      - image: nginx
        name: web3
$ kubectl create -f web3.yaml
```

```
$ kubectl get deploy
NAME   READY  UP-TO-DATE  AVAILABLE  AGE
web    1/1    1           1          2m
web2   1/1    1           1          2m
web3   1/1    1           1          2m
```

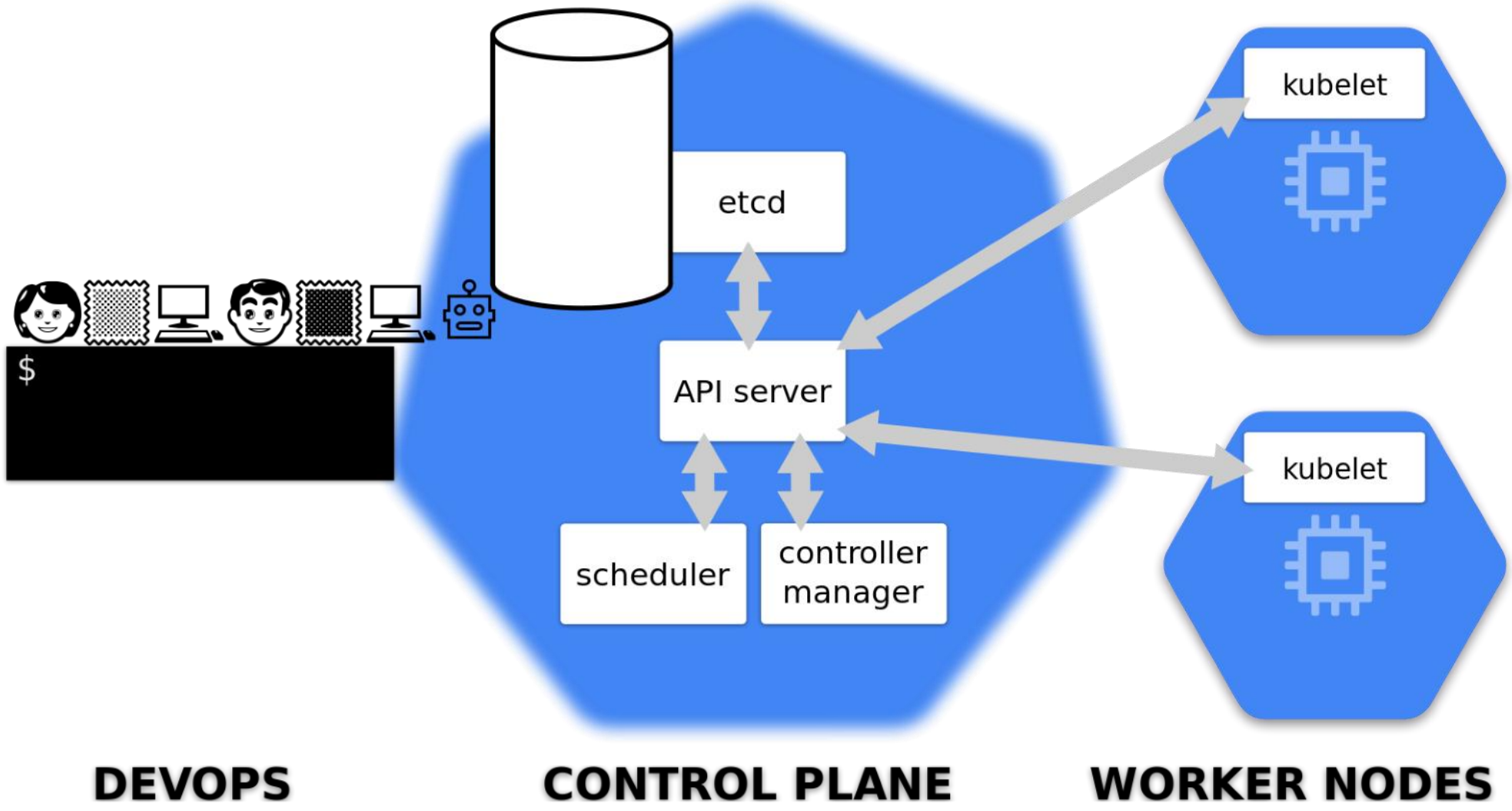
Quelques différences:

- Labels associés (ie. « run » vs « app »)
- Noms de ressources
- Valeurs par défaut

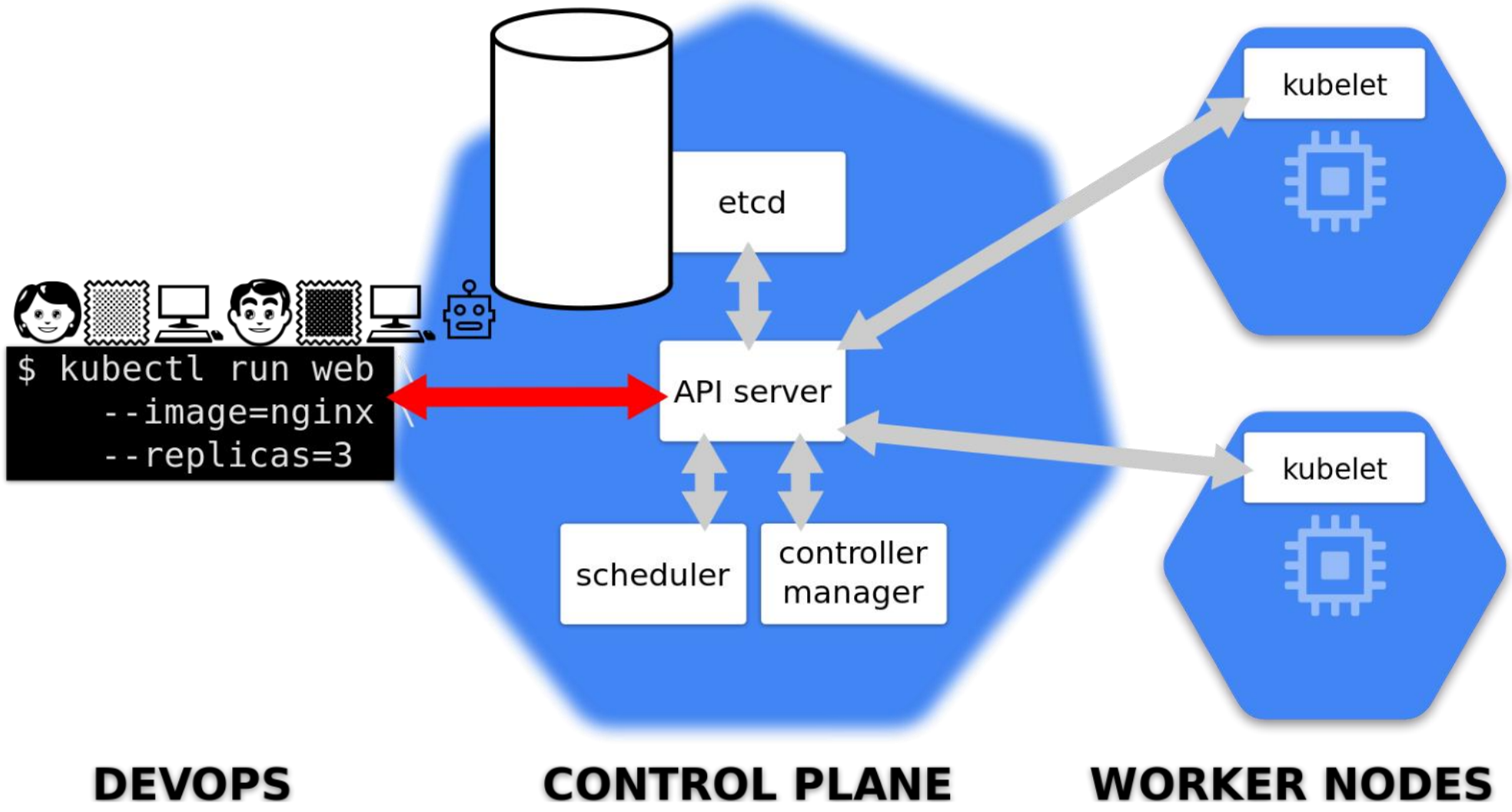
kubectl run web --image=nginx --replicas=3



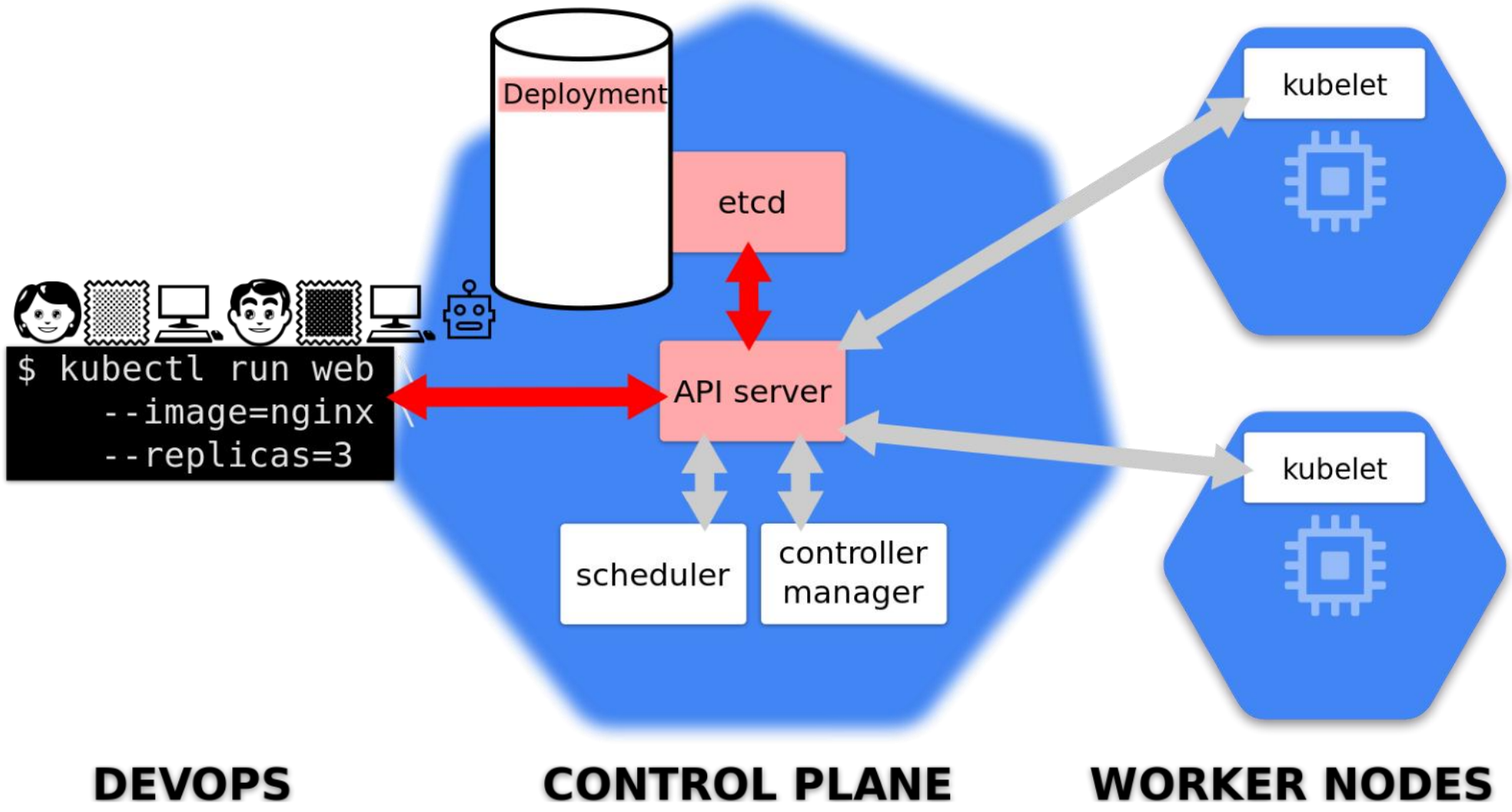
kubectl run web --image=nginx --replicas=3



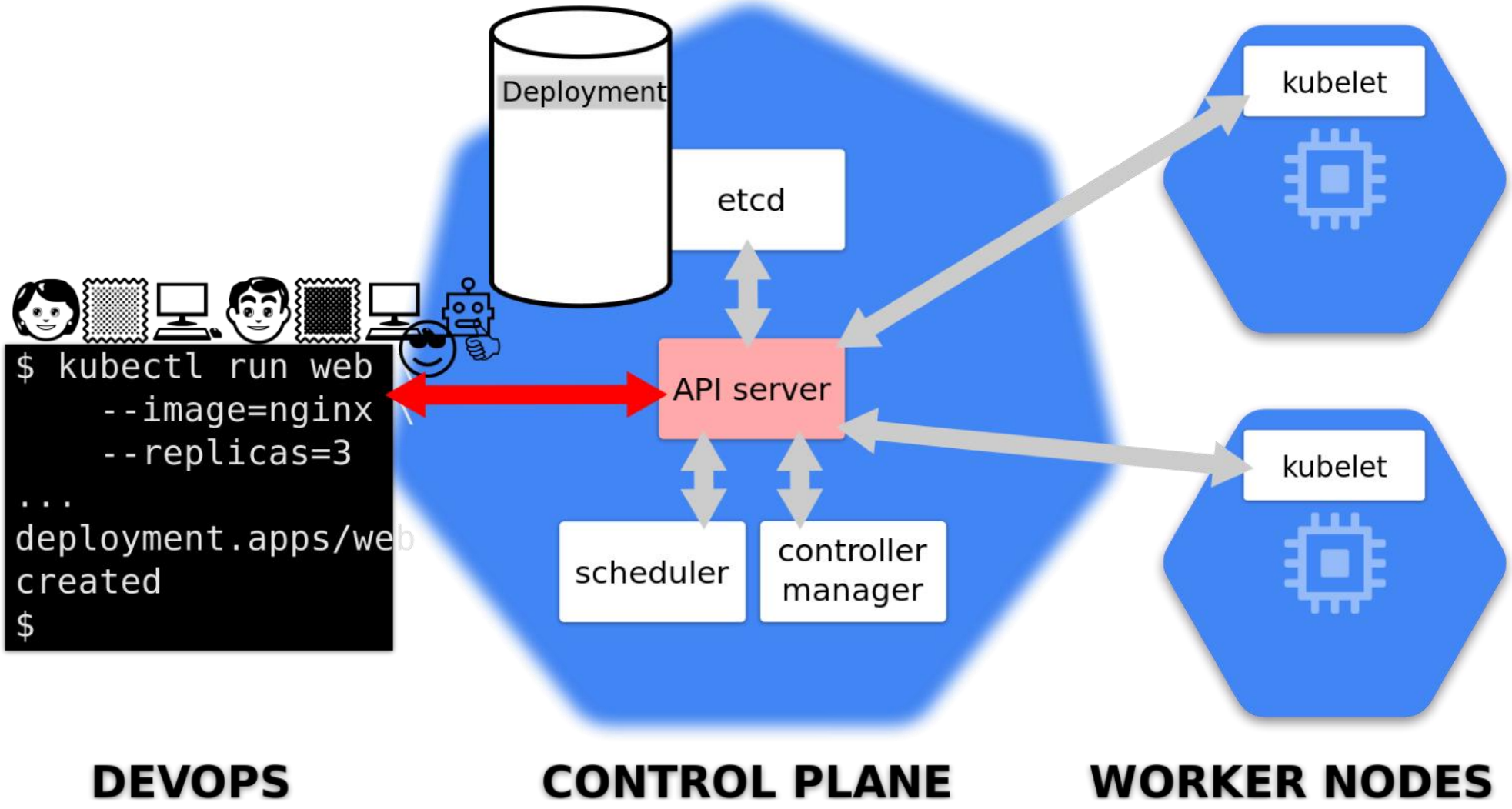
kubectl run web --image=nginx --replicas=3



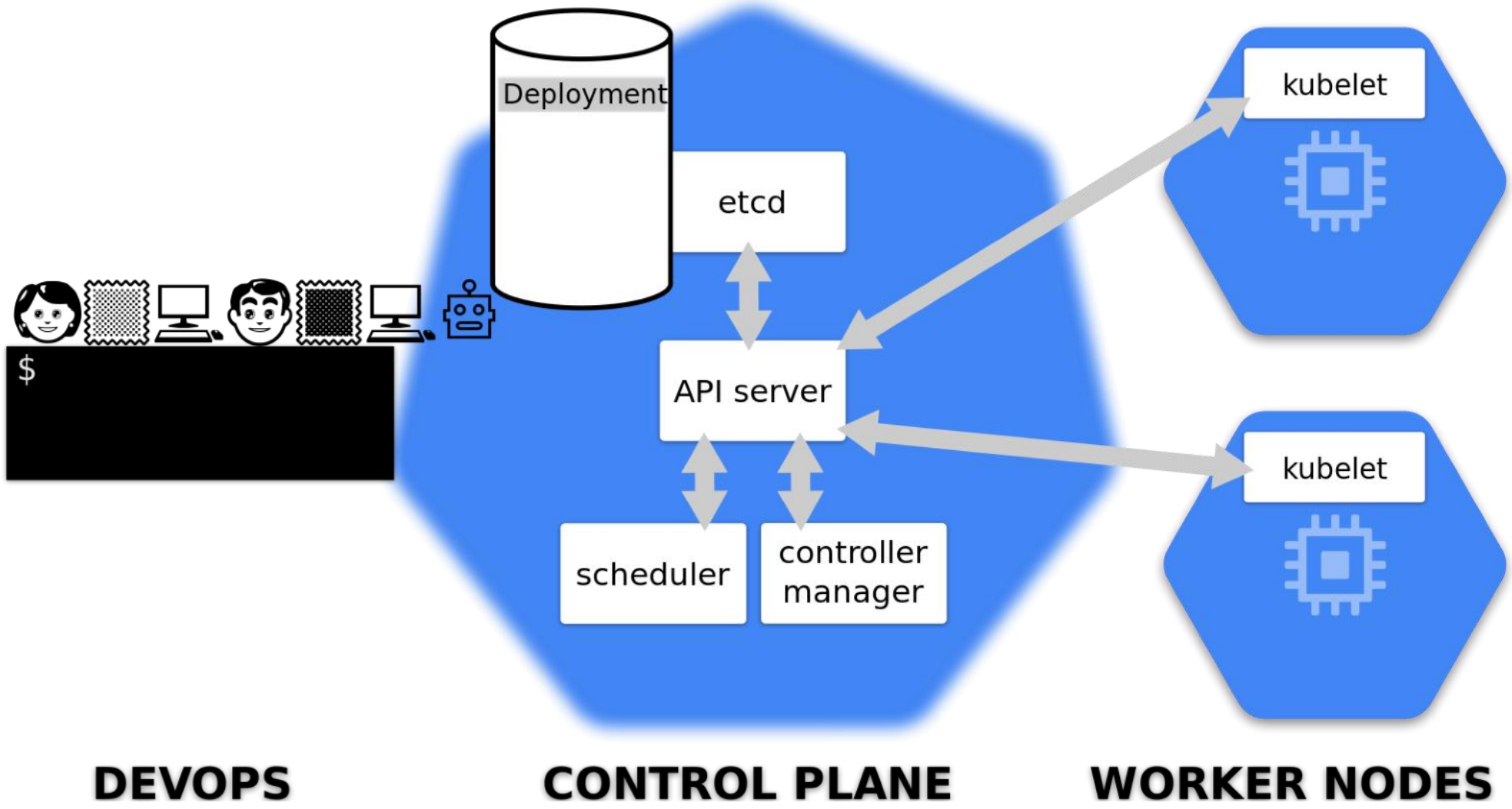
kubectl run web --image=nginx --replicas=3



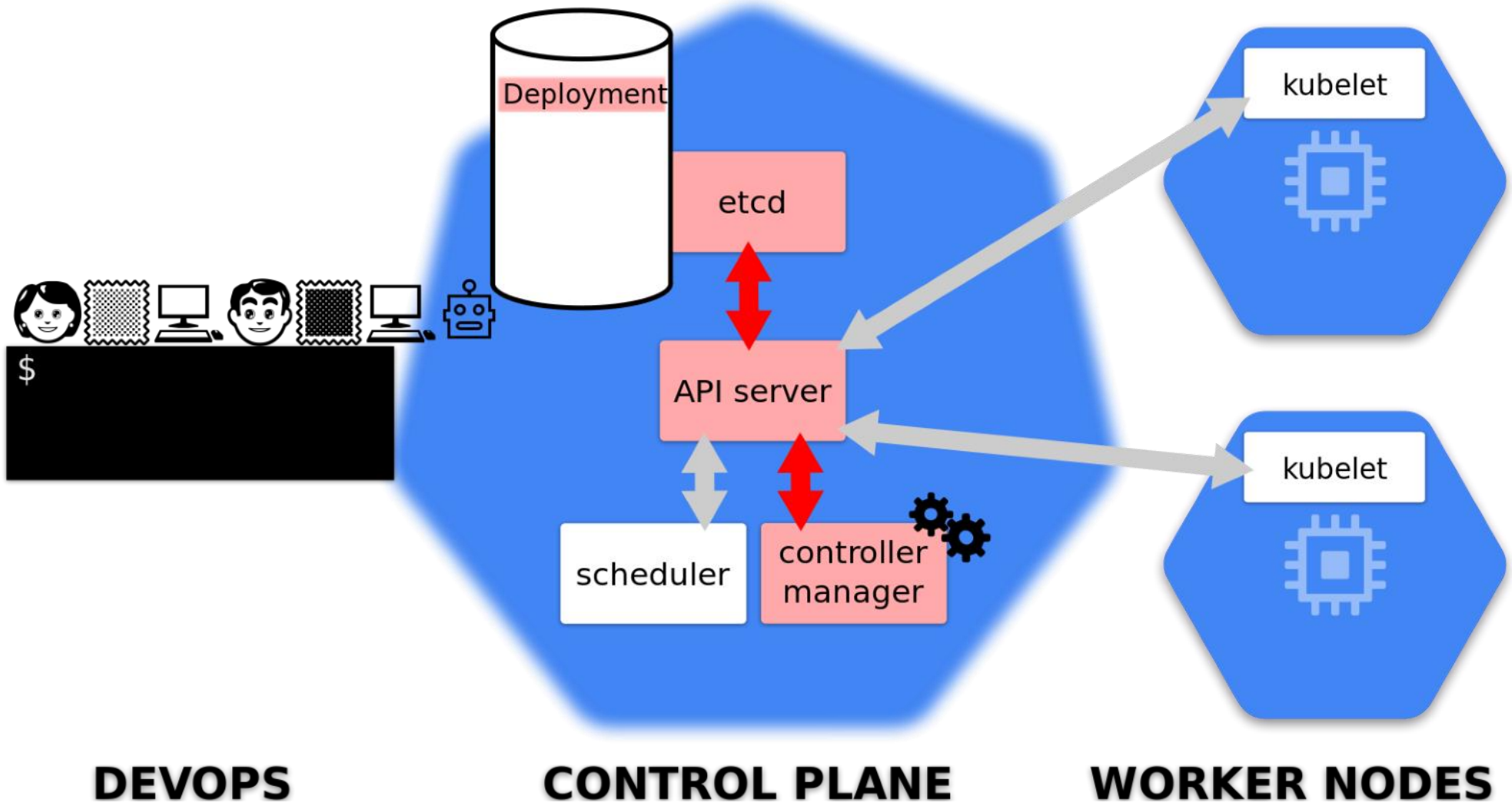
kubectl run web --image=nginx --replicas=3



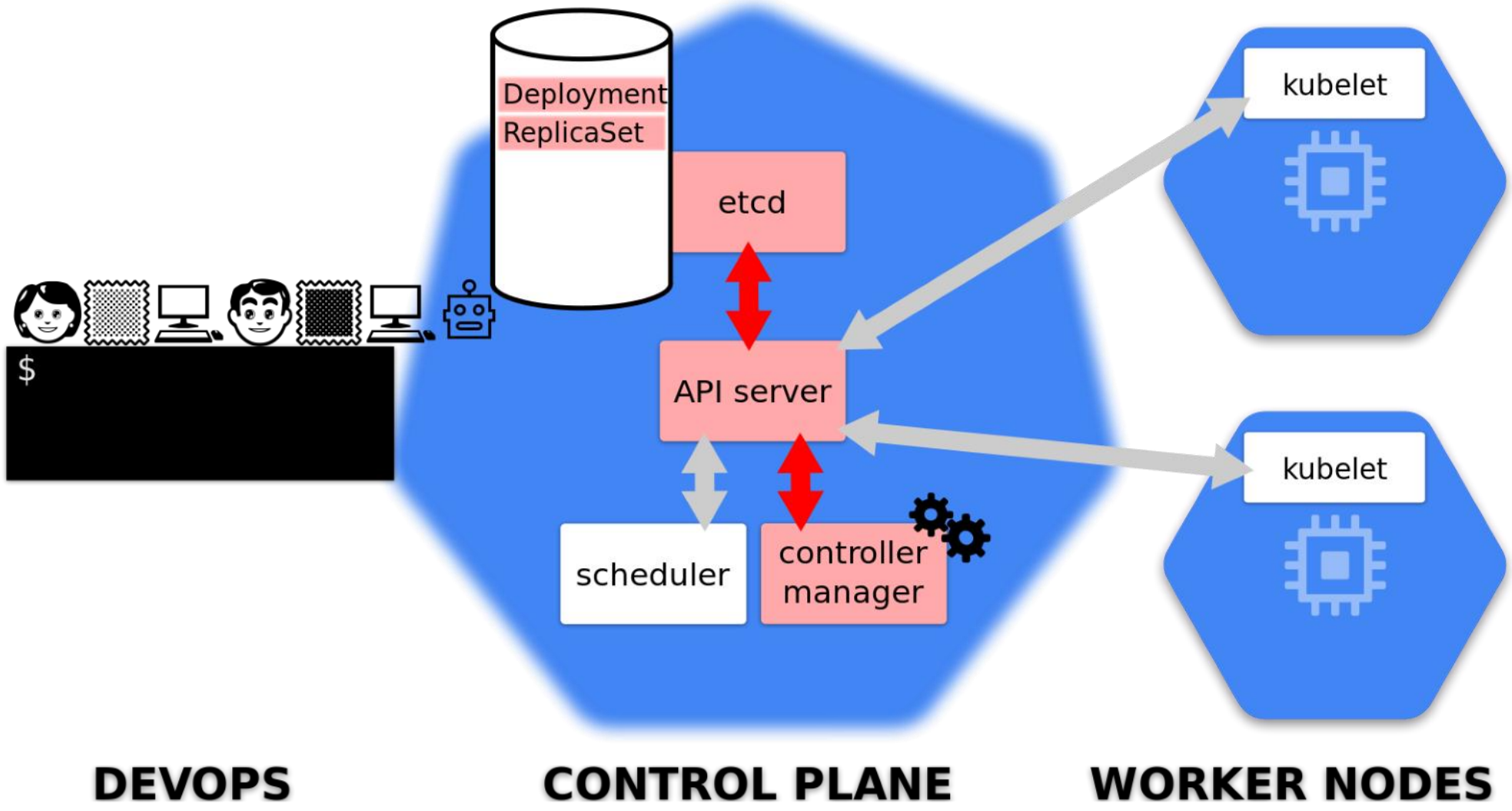
kubectl run web --image=nginx --replicas=3



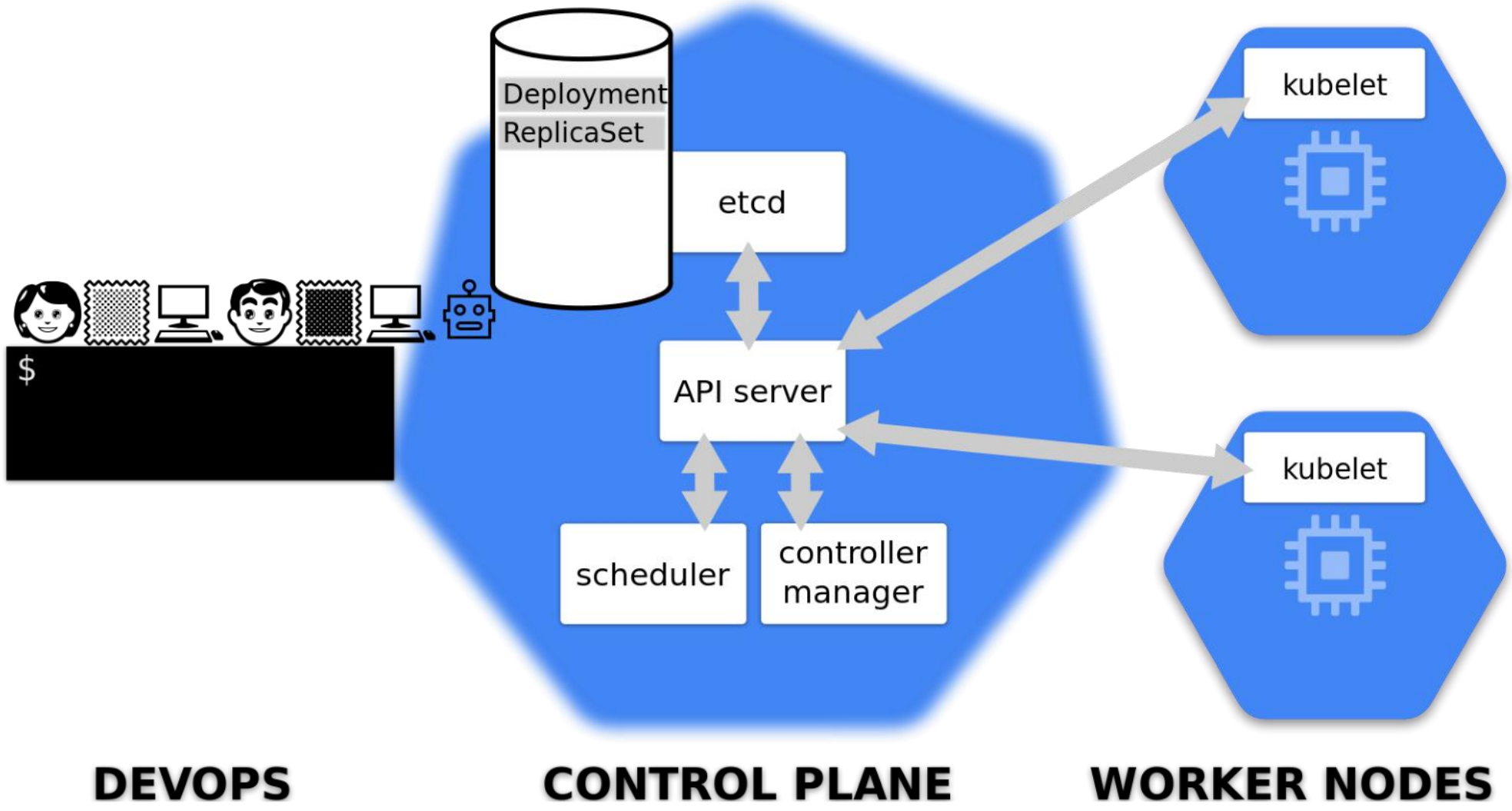
kubectl run web --image=nginx --replicas=3



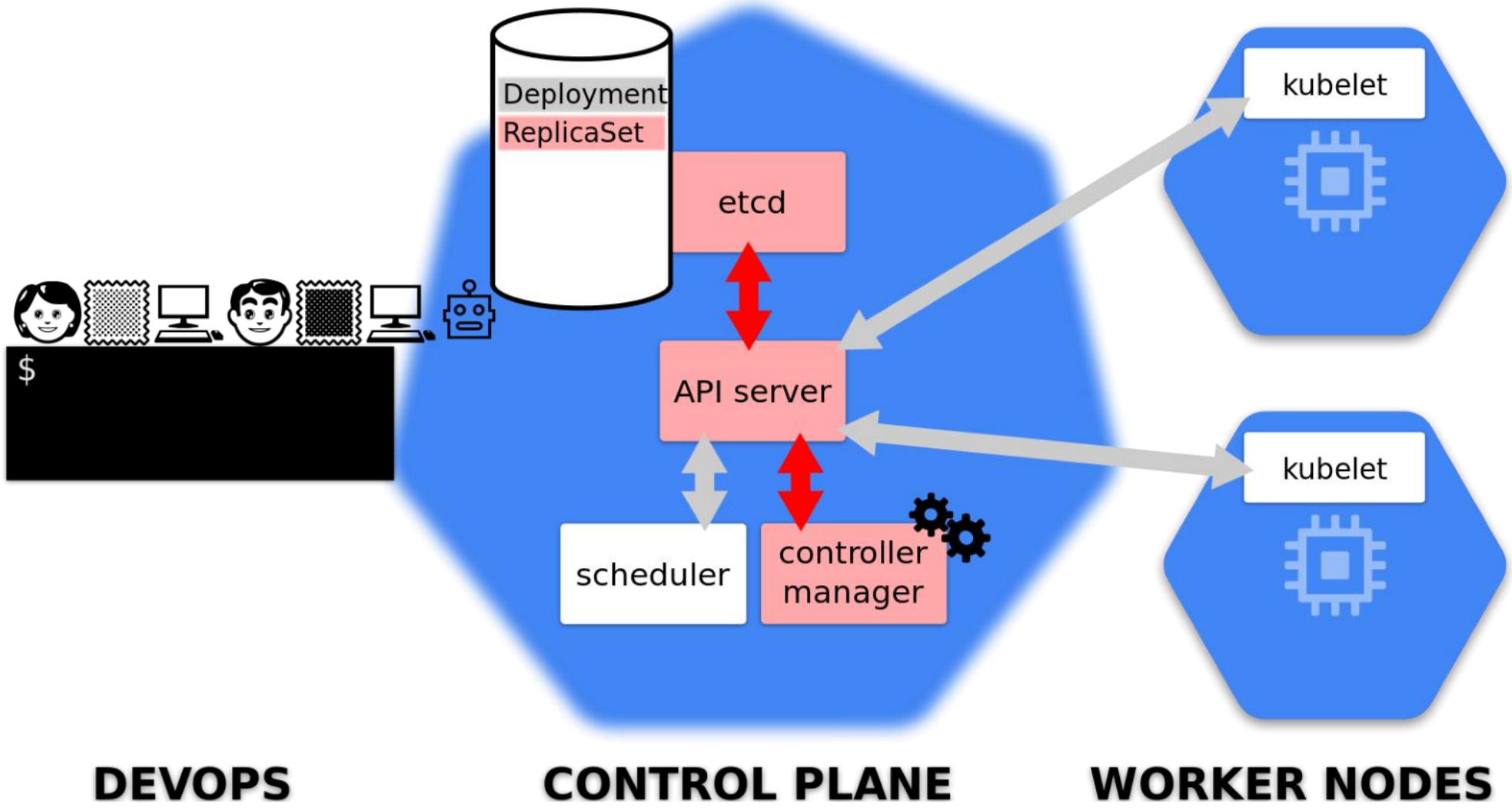
kubectl run web --image=nginx --replicas=3



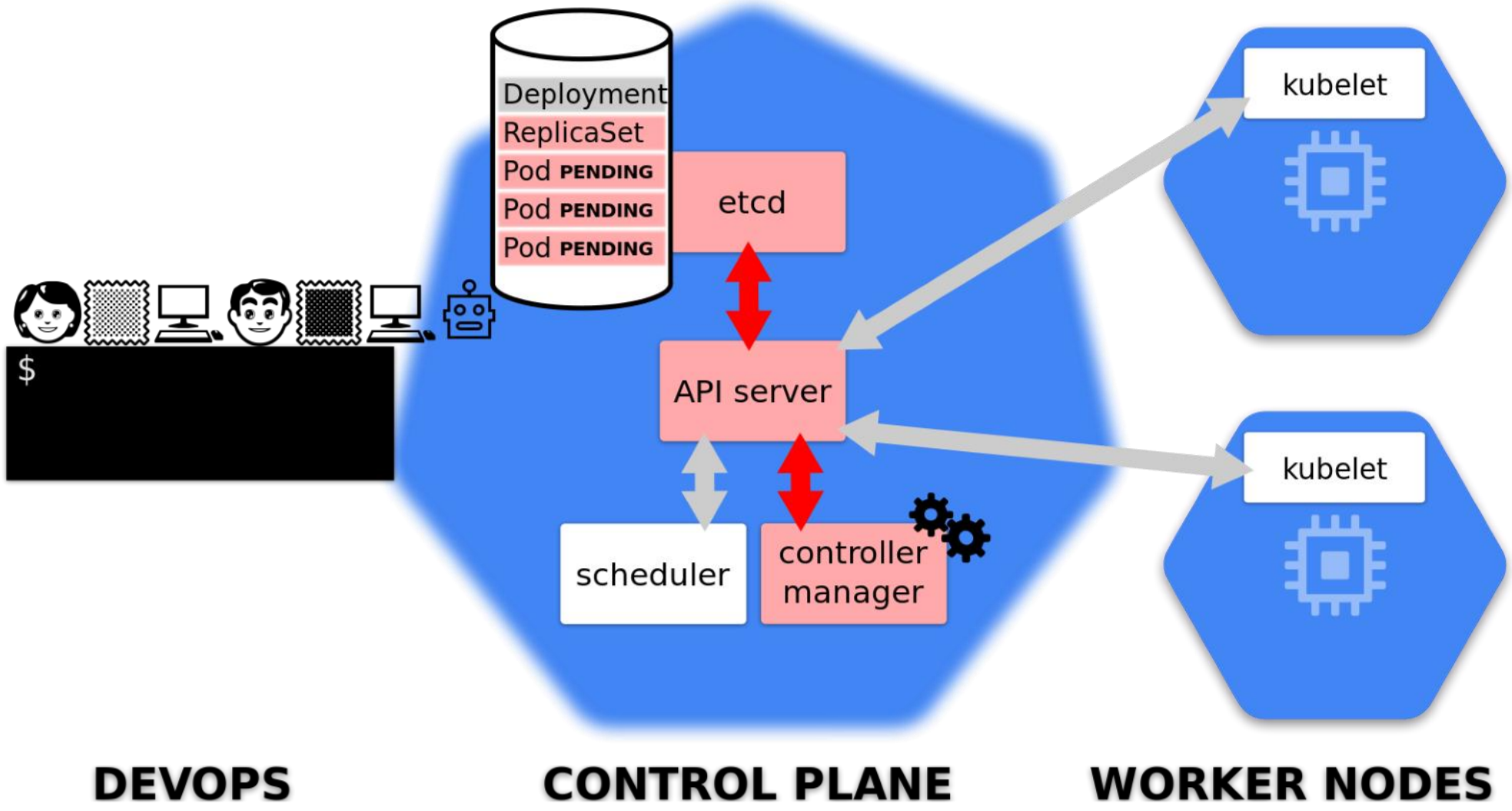
kubectl run web --image=nginx --replicas=3



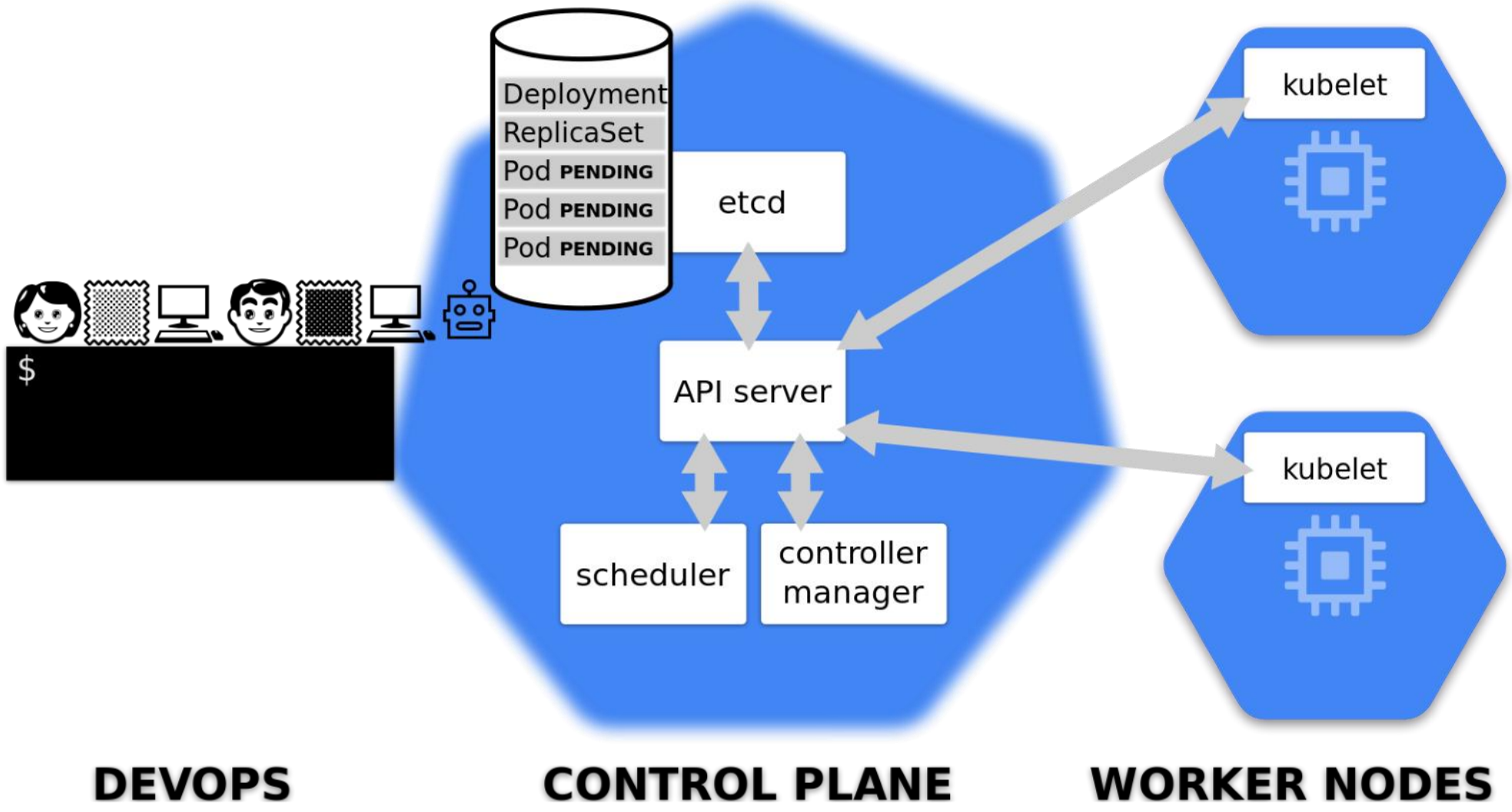
kubectl run web --image=nginx --replicas=3



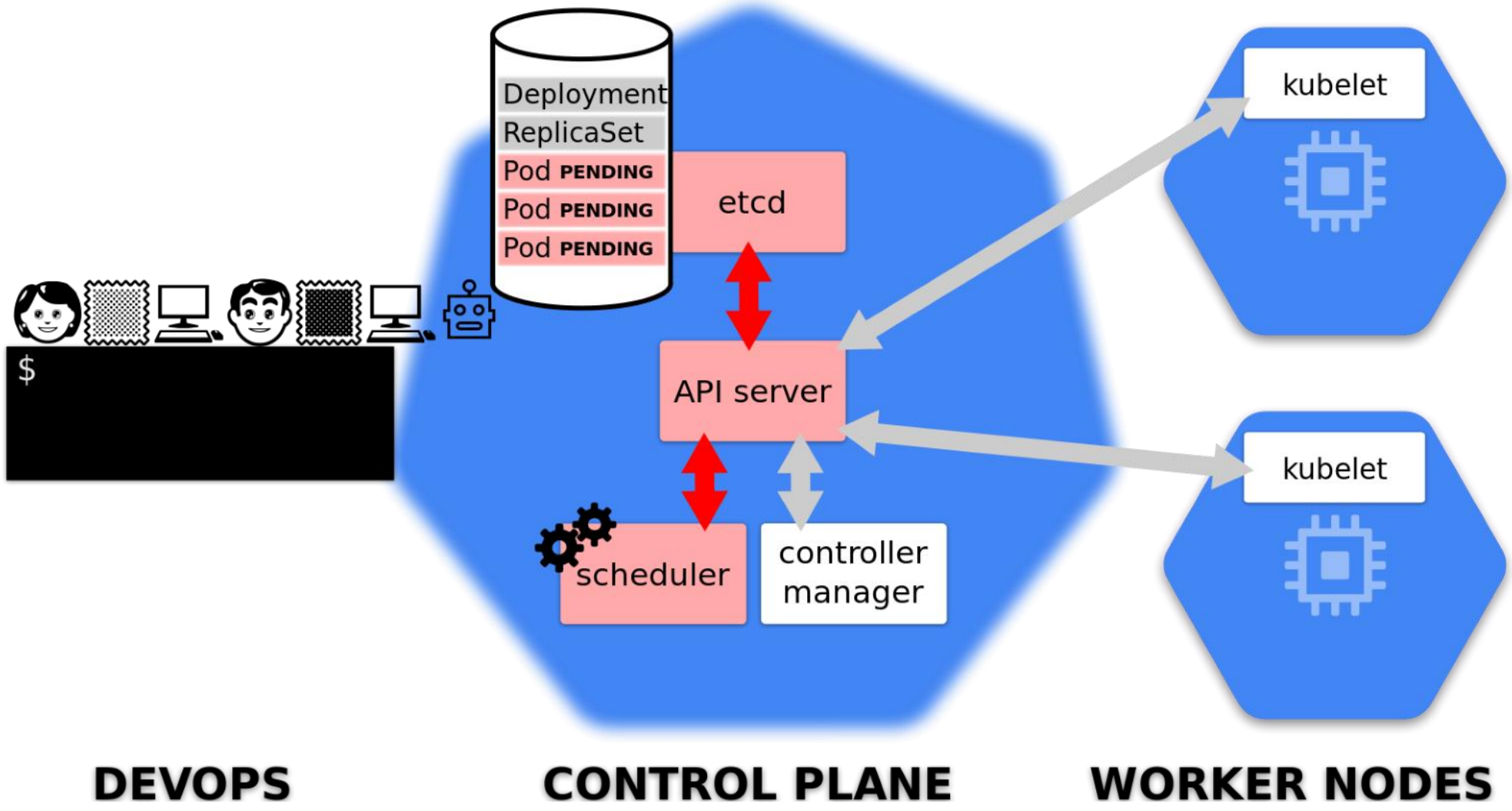
kubectl run web --image=nginx --replicas=3



kubectl run web --image=nginx --replicas=3

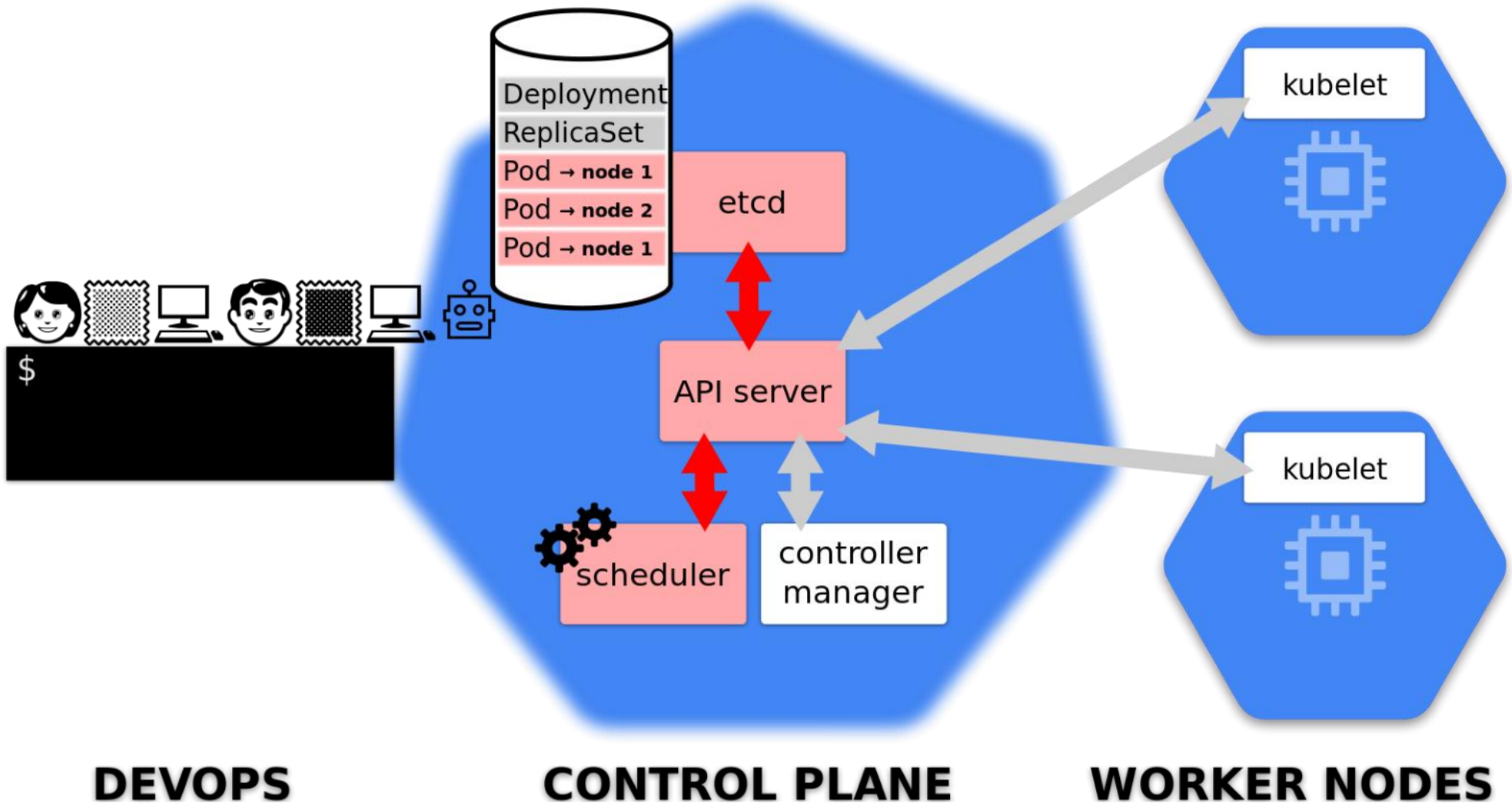


kubectl run web --image=nginx --replicas=3



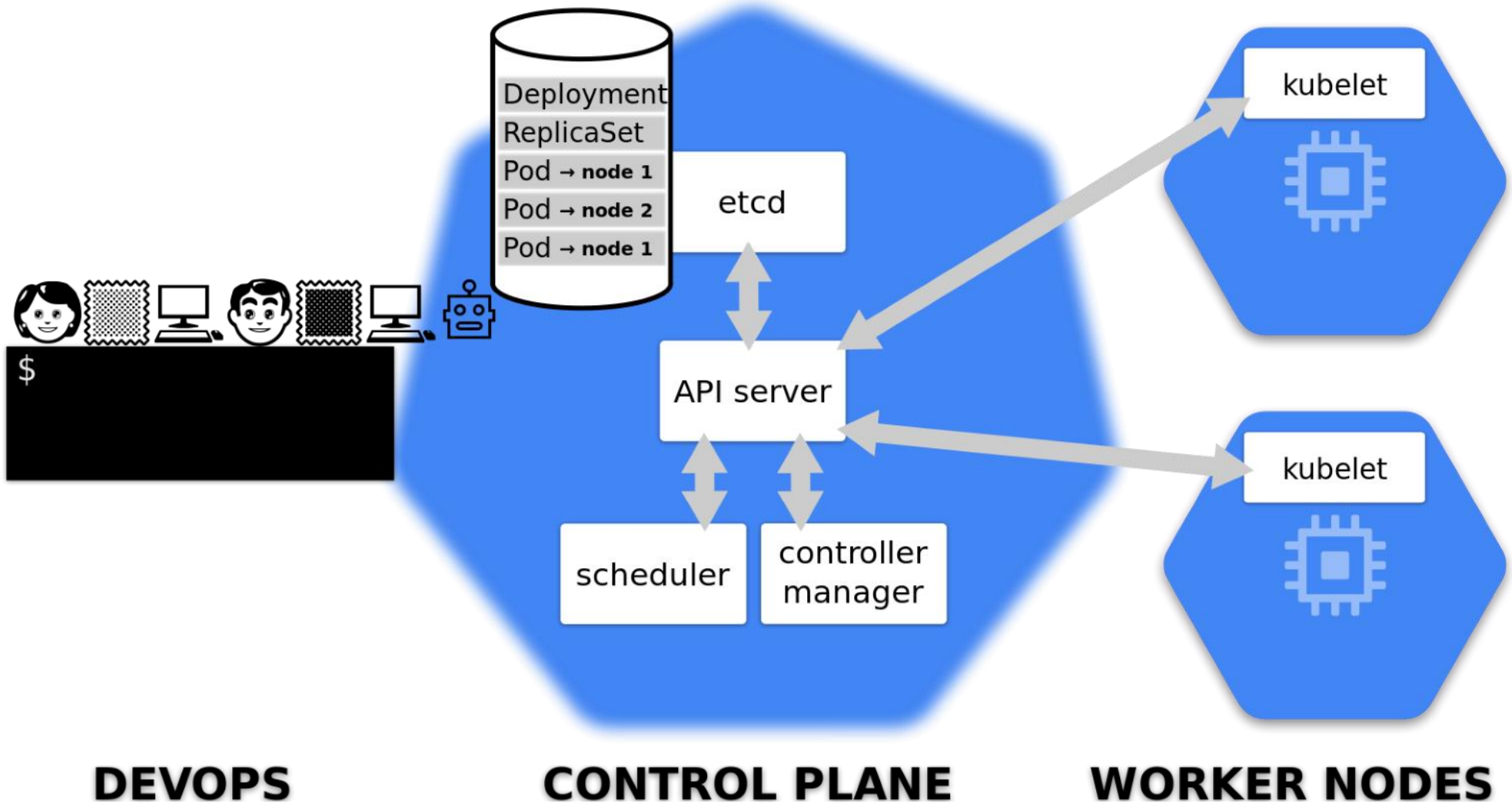
©Jérôme Petazzoni

kubectl run web --image=nginx --replicas=3

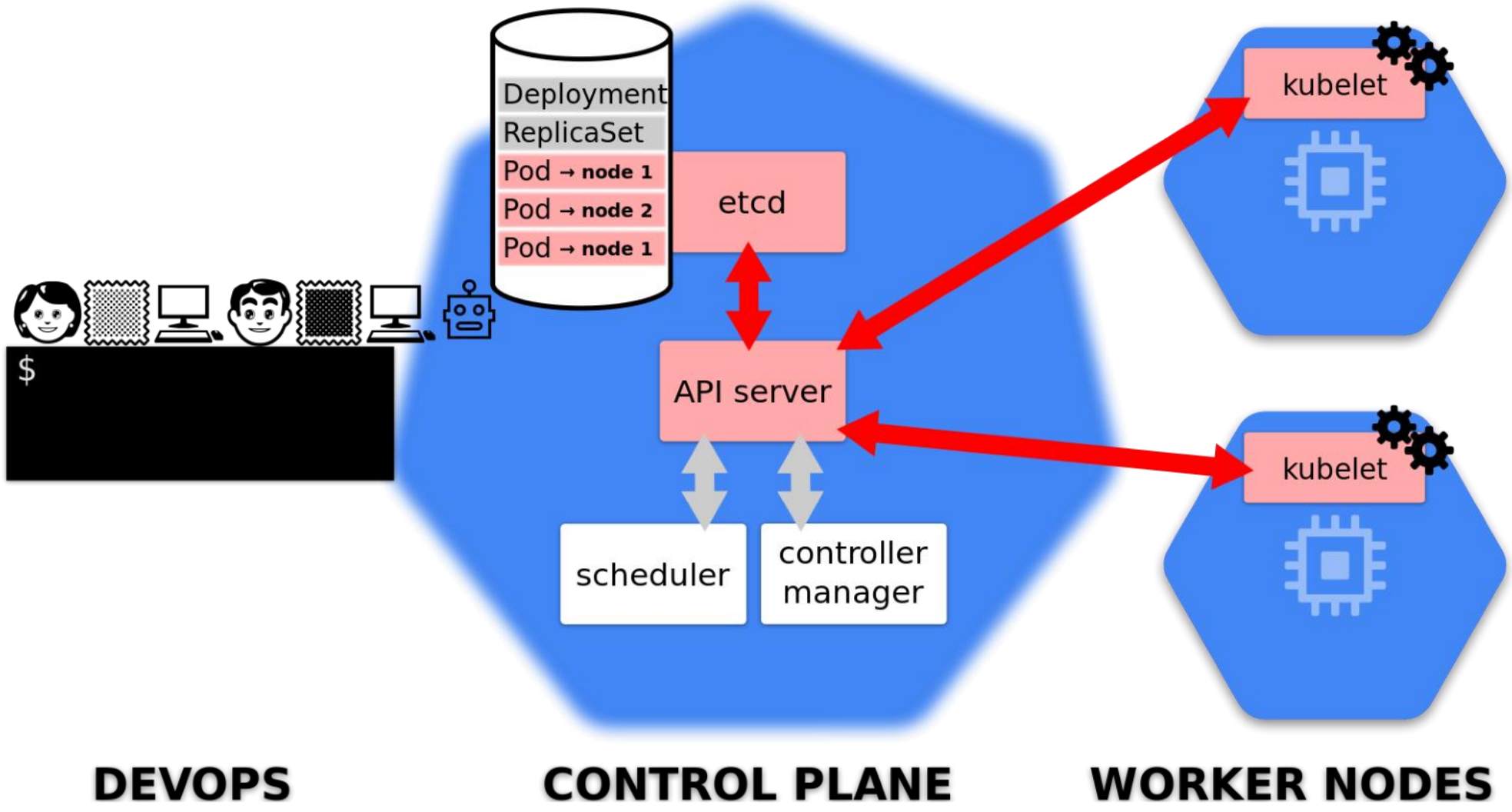


©Jérôme Petazzoni

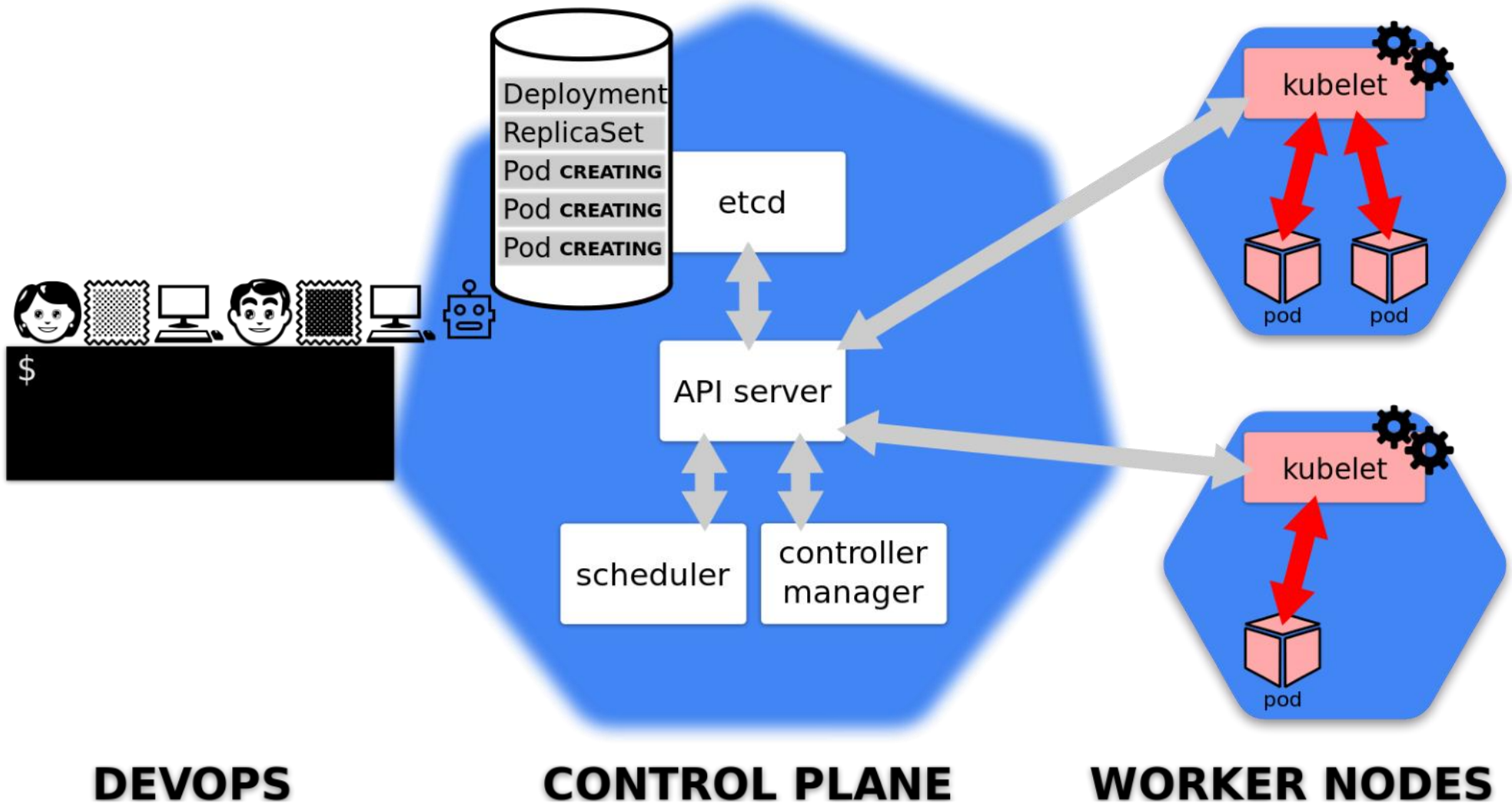
kubectl run web --image=nginx --replicas=3



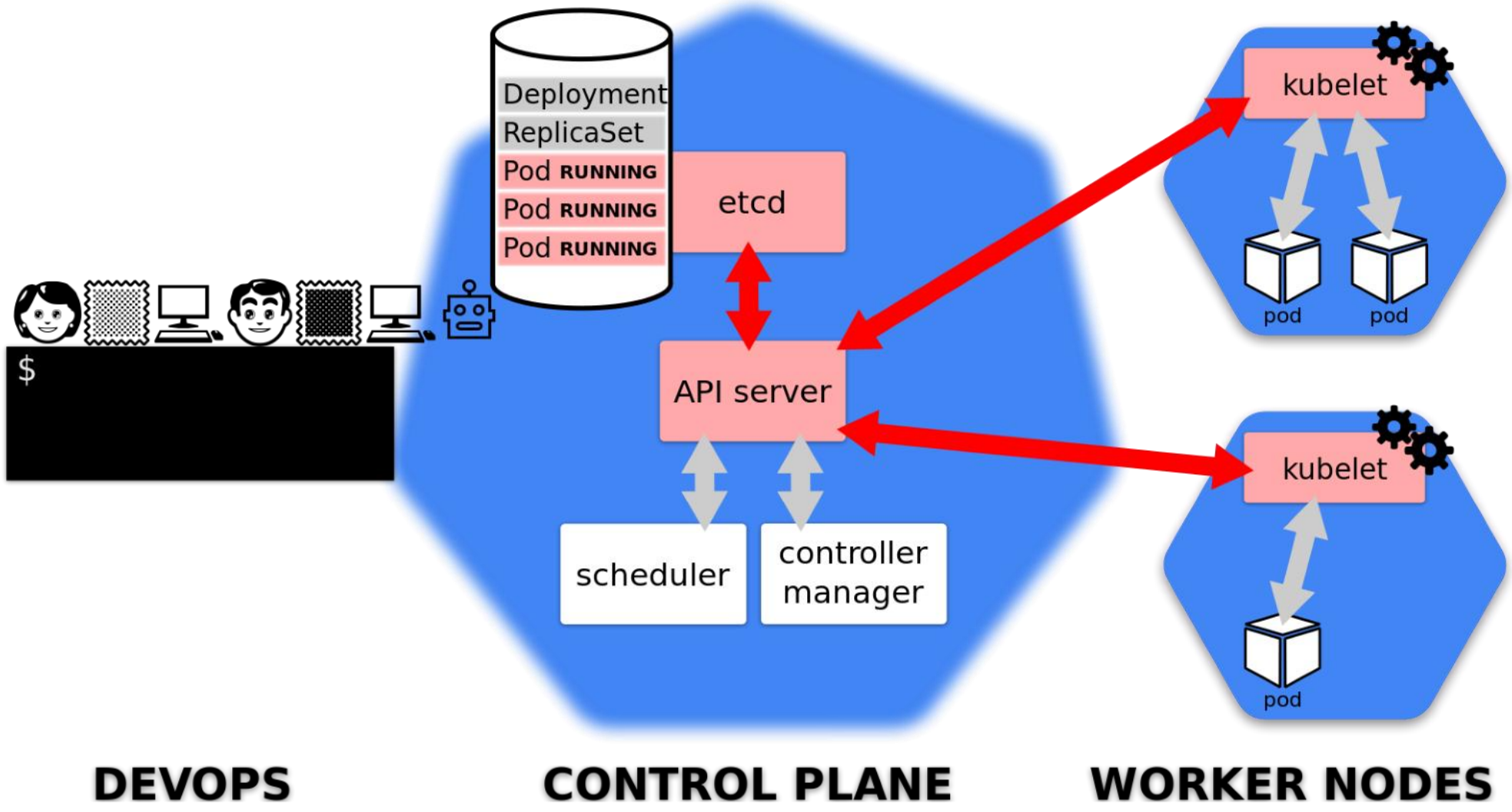
kubectl run web --image=nginx --replicas=3



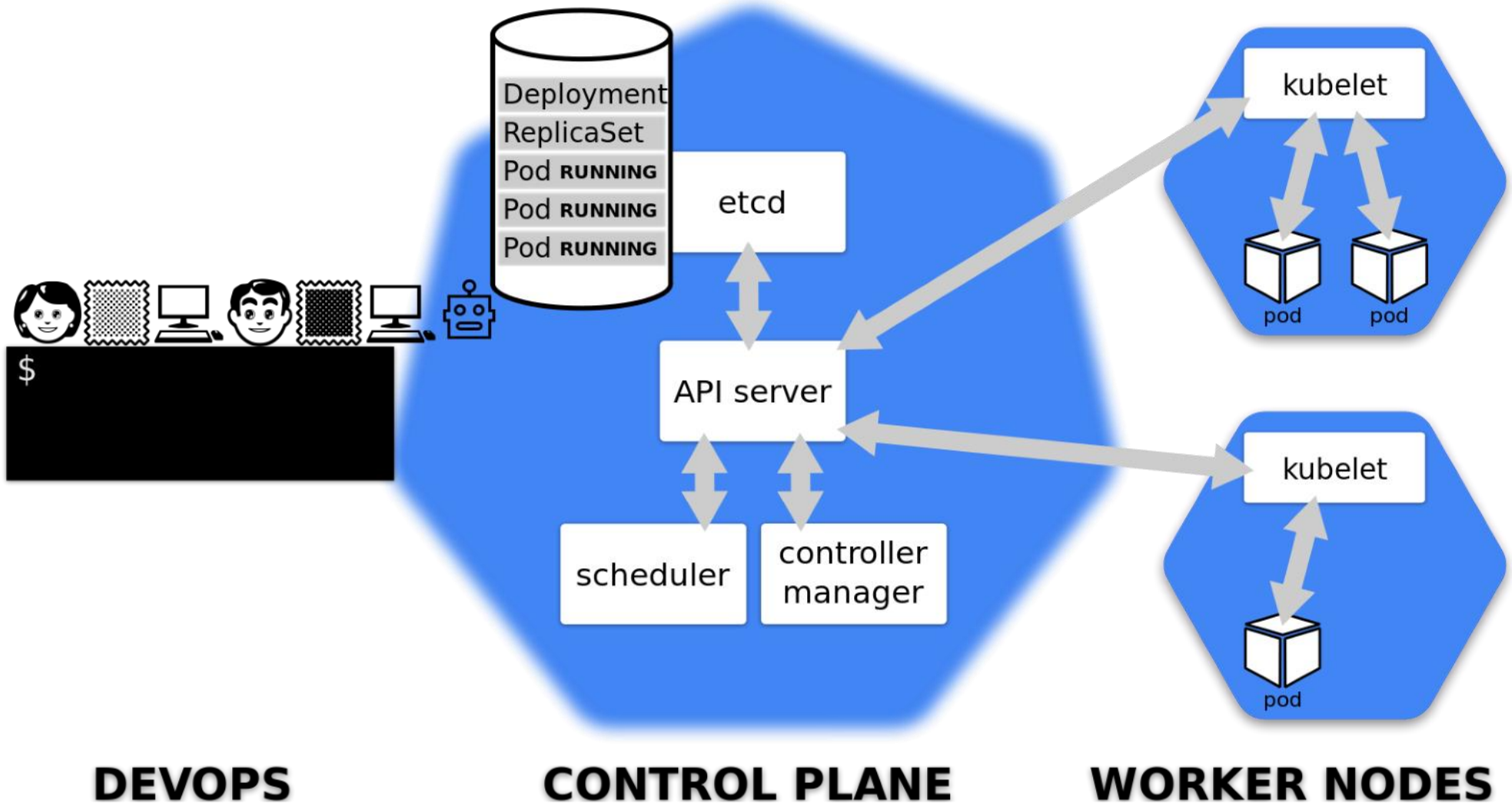
kubectl run web --image=nginx --replicas=3



kubectl run web --image=nginx --replicas=3



kubectl run web --image=nginx --replicas=3



Notion de « replicas » dans un deployment: ordonne « n » copies du pod

Manuellement:

```
$ kubectl scale --replicas=3 deploy/web
deployment.extensions/web scaled

$ kubectl get deploy -l run=web
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
web    3/3     3            3           3d5h

$ kubectl get pods -l run=web
NAME                   READY   STATUS    RESTARTS   AGE
web-59765d756f-chfpq   1/1     Running   0          29s
web-59765d756f-tw2c4   1/1     Running   0          3d5h
web-59765d756f-x6tgk   1/1     Running   0          29s
```

Ou automatiquement avec un HorizontalPodAutoscaler par exemple.

- En fonction de l'utilisation des ressources allouées (CPU, RAM …)

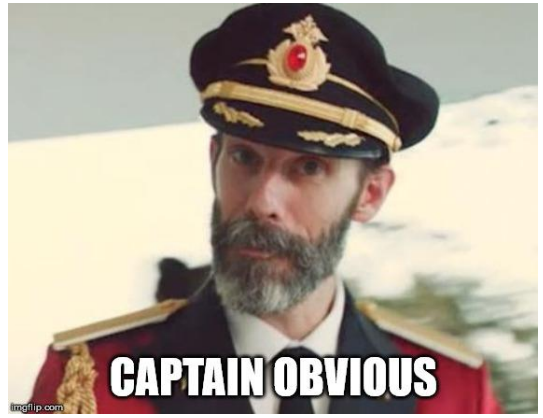
Un déploiement va créer un replicaSet par « version »

- Modifier le déploiement crée une nouvelle version
- Retour arrière en cas de problème

```
$ kubectl patch deploy/web \
   -p '{"spec":{"template":{"spec":{"containers":[{"name":"web","image":"nginx:latest"}]}}}}'
deployment.extensions/web patched

$ kubectl get deploy -l run=web
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
web    3/3     1            3           3d5h

$ kubectl get rs –l run=web
NAME                 DESIRED   CURRENT   READY   AGE
web-59765d756f       0         0         0       3d6h
web-5d6f76596        3         3         3       18s

$ kubectl get pods –l run=web
NAME                        READY   STATUS    RESTARTS   AGE
web-5d6f76596-7zvzz         1/1     Running   0          21s
web-5d6f76596-gkjpd         1/1     Running   0          17s
web-5d6f76596-lr5kv         1/1     Running   0          27s
```

Le deployment définit la stratégie de déploiement

```
$ k get deploy/web -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: web
  name: web
spec:
  […]
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template: […]
```



RollingUpdate

- On attend que n+1 soit opérationnel avant de tuer n

Recreate

- On tue n avant de créer n+1

Comment « exposer » son conteneur au monde qui l'entoure ? Et à internet?

```
$ kubectl expose deployment web --port 8443
service/web exposed

$ kubectl get svc
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
web             ClusterIP   10.97.246.71    <none>        8443/TCP         3s
```

Un service rend notre pod accessible:

- Via un nom « statique », unique mais local (cluster local DNS)
- Quel que soit le nombre de replicas ou leur état

C'est un point d'accès (ie. « endpoint ») de niveau 4

```
$ kubectl get endpoints
NAME   ENDPOINTS         AGE
web    10.1.0.85:8443    34s
```

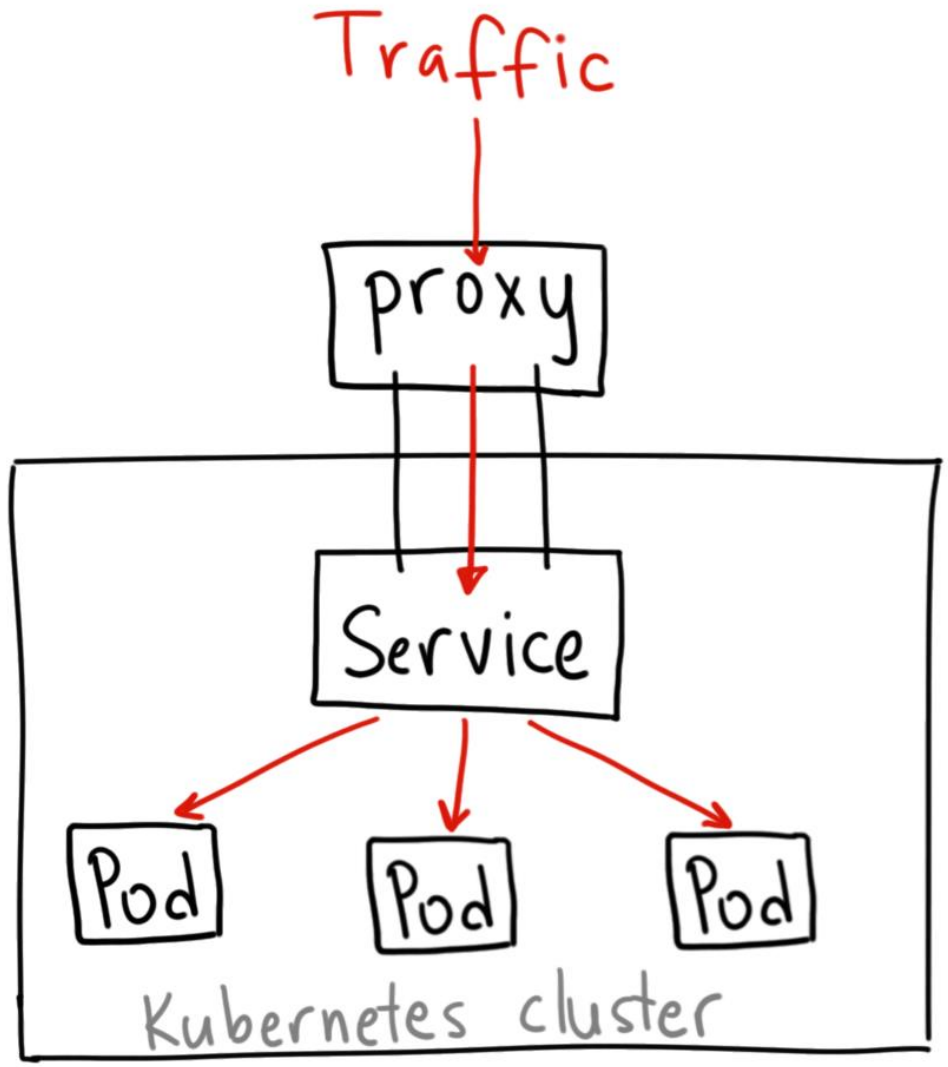
ClusterIP = adresse IP privée

- C’est bien pour communiquer à l’intérieur du cluster
- Mais comment accéder à notre pod en dehors du réseau local?

KubeProxy est capable de mettre en place quelques règles iptables pour ça (SNAT FTW!)

```
$ kubectl proxy --port=8080
```

- Bien pour débuguer
- Il faut être cluster admin !!



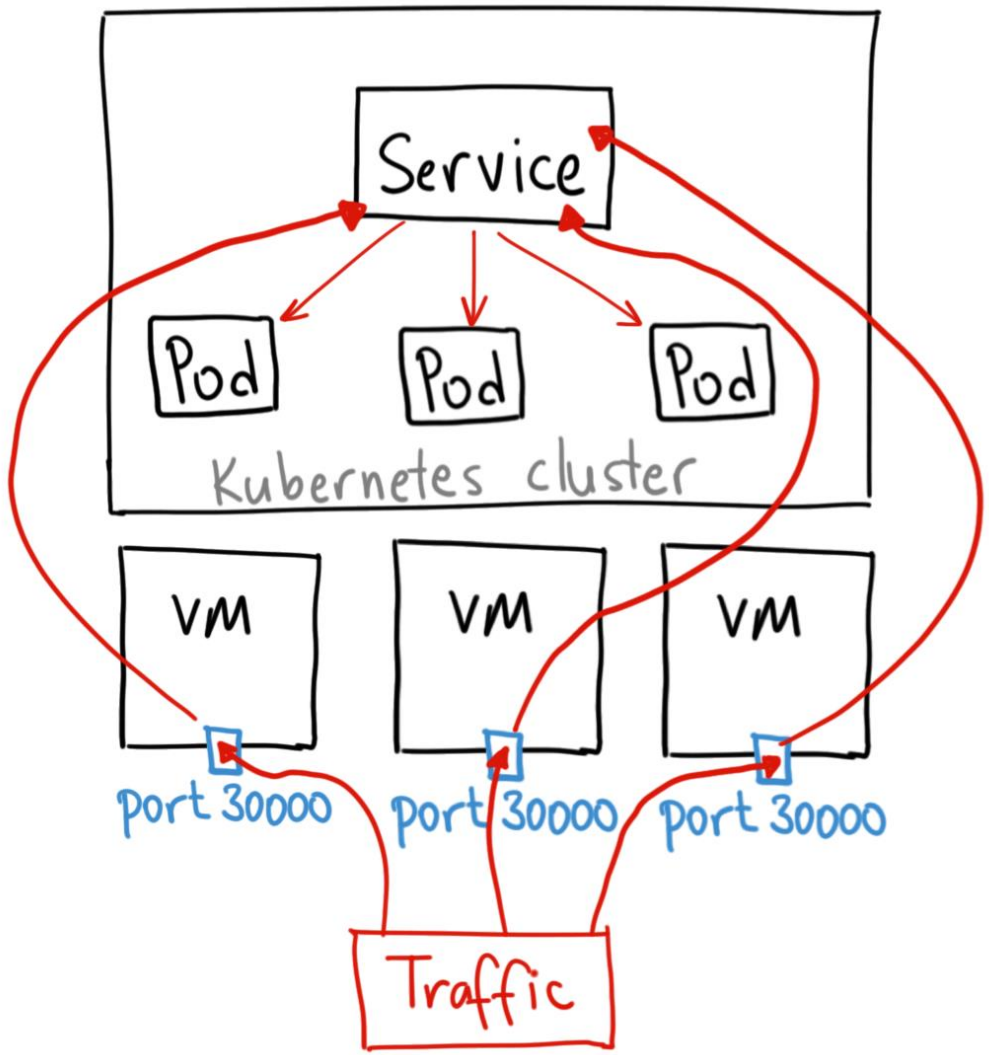
© Ahmet Alp Balkan

Du NAT à grand échelle

- Un port ouvert sur chaque nœud du cluster
- Forward des trames depuis n'importe quel nœud vers le service

C'est simple et ça marche

- Mais ça reste du NAT
- Il faut utiliser un port > 32000
- Un service = un port

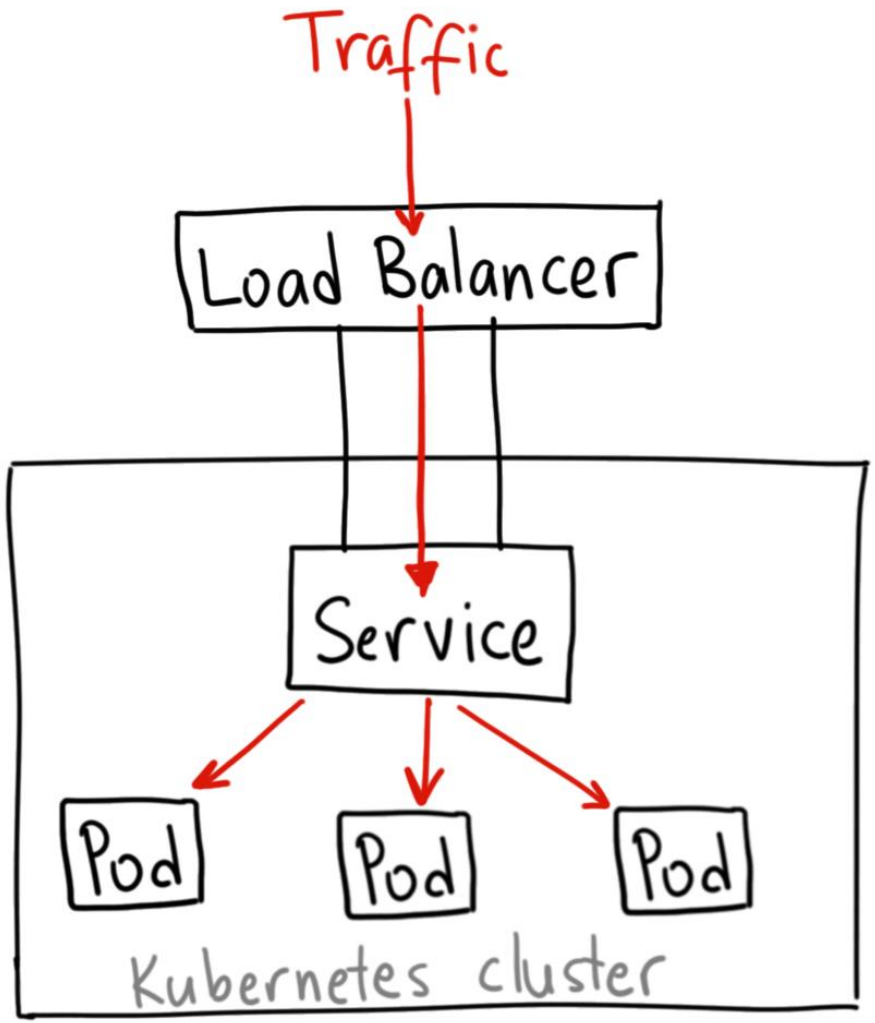


Le plus logique et efficace

- Comme une VIP classique
- Supporte tous les protocoles, tous les ports
- Aussi performant que l'implémentation utilisée
- Nécessite un service de loadbalancer (LBaaS)

Mais aussi le plus couteux:

- Un service = une adresse IP
- LBaaS = $$$ (GKE, AKS, EKS …)



© Ahmet Alp Balkan

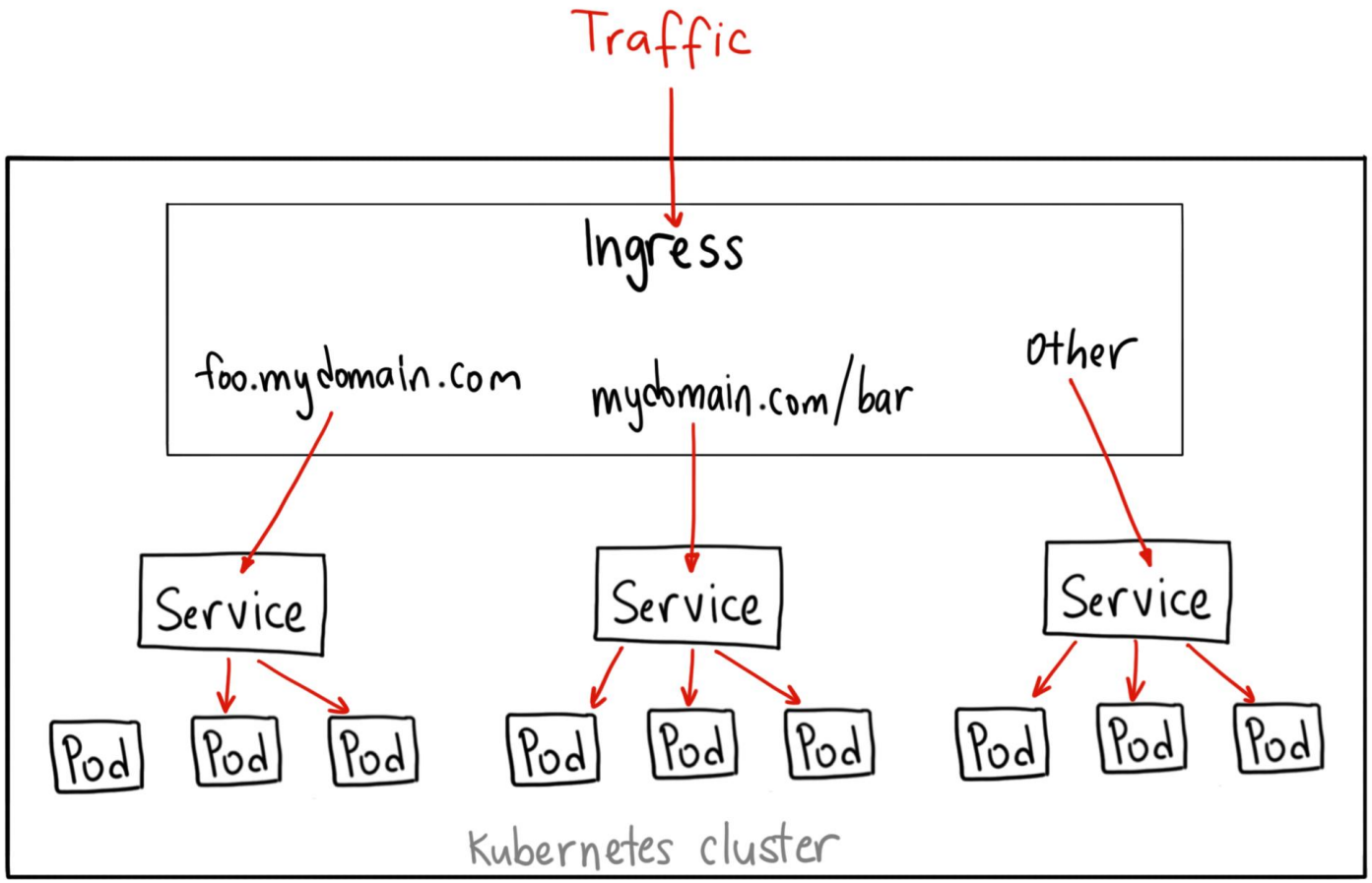
Le plus « élégant »

- Optimisé pour HTTP/HTTPS
- Peut faire plein de choses avec le FQDN
- Gère les certificats (plugin)

Mais ne convient pas à tous les usages …

- Je suis un hacker, je veux faire de l'UDP sur un port chelou !

Attention, Ingress n'est pas un type de service ;)



© Ahmet Alp Balkan

Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules

Partie IV - Kubernetes

Intégration

Conteneur = éphémère

- Où mettre ses données?

Dans un volume !

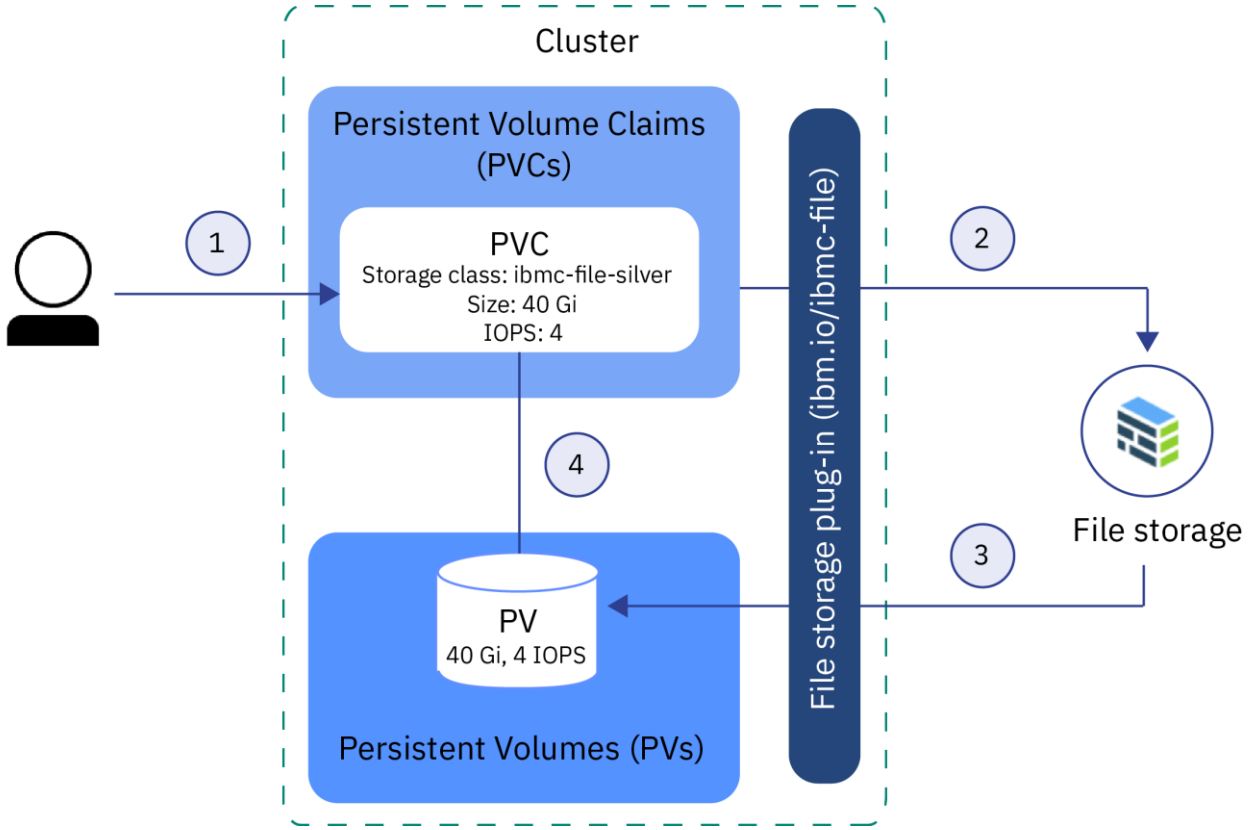
- Non, c'est local au nœud :/

Système de stockage externe au cluster

- PersistentVolumeClaim: j'ai besoin de tant
- PersistentVolume: je peux stocker tant

PV + PVC = stockage pour le pod

- Et on monte tout ça sur le conteneur



Les images de conteneurs doivent être le plus générique possible

- Que faire des éléments de configuration ?
- Que faire des secrets (mots de passe, clés, certificats, etc) ?

ConfigMap pour le premier et Secret pour le second.

- Stocké dans etcd (et chiffré pour le Secret)
- Fichier ou chaine de caractères
- Monté dans un conteneur, comme un PV
- Associé à un namespace (droits d’accès)

L’agent Kubelet récupère plein d’informations sur les conteneurs

- Les journeaux d’évènements (stdout/stderr)
- Les métriques (CPU, RAM, IOPS, NET, etc..)

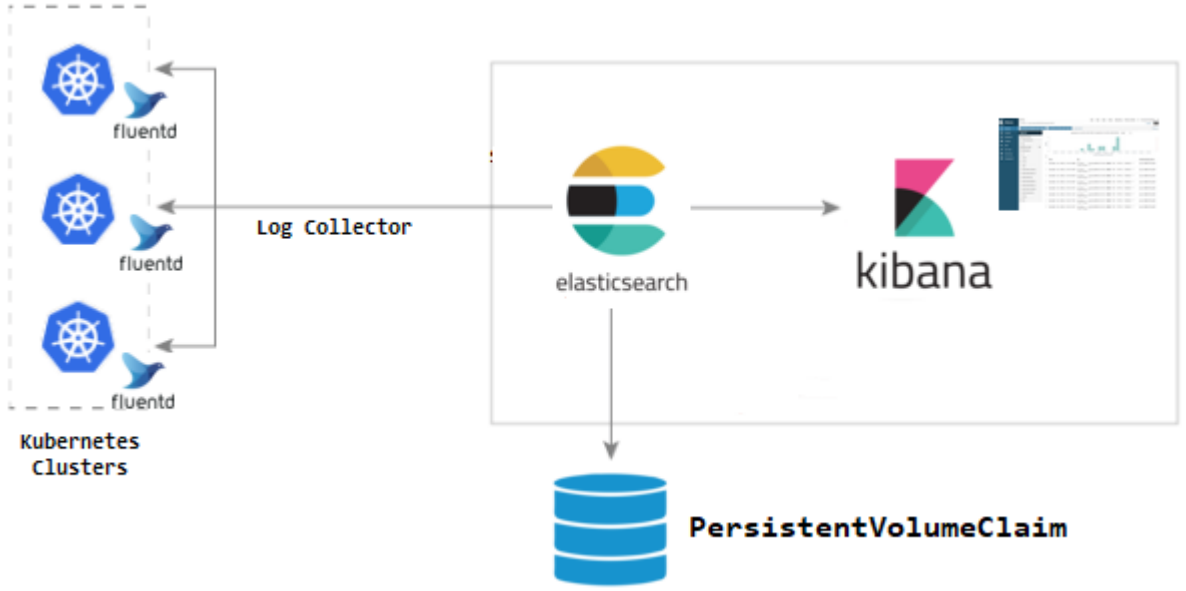
```
$ kubectl logs -f deploy/web
```

```
$ kubectl logs -f pod/web-59765d756f-chfpq
```

C’est bien mais ça scale pas très bien tout ça

- Centralisation des logs
- Centralisation des métriques

Pour visualisation, alertes, autoscaling …



Et la sécurité ?

Plusieurs niveaux de sécurité:

- Utilisateur (AAI):
  - Role Based Access Control (RBAC)
  - IdentityProvider
  - ServiceAccount
- Conteneur:
  - NetworkPolicy (iptables, netfilter …)
  - PodSecurityPolicy (capabilities, SELinux, AppArmor …)
- Images:
  - Private registries
  - Image signature

Centre de Calcul de l'Institut National de Physique Nucléaire et de Physique des Particules

Partie V – Pour aller plus loin…

Au boulot!

Le site officiel https://kubernetes.io est la référence incontournable.

Pour commencer: Minikube, un cluster « single node all-in-one »

- Sur son laptop (Linux, MacOS, Win10: pas d'excuse)
- Sur une machine physique ou virtuelle
- Dans le cloud (GKE, AKS, EKS …)

Sinon, une bonne ressource: https://container.training par Jérôme Petazzoni

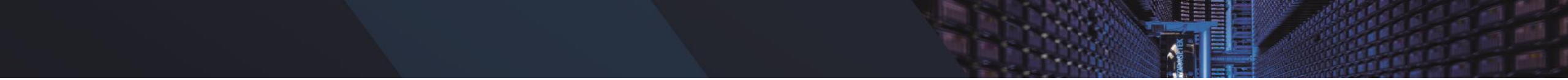
- Docker, Compose
- Docker Swarm
- Kubernetes -> 1500 slides ! (dont je me suis pas mal servi pour cette présentation)

Plein d’autres choses à voir

- Advanced deployment schemes
- Templates (ie. Helm charts)
- Sidecar containers
- CI/CD
- Ingress/Egress
- Storage classes
- Network policies
- Custom resource definitions
- Operator framework
- Service meshes
- …

Certains de ces éléments seront abordés aujourd’hui.

Stay tuned!



The end