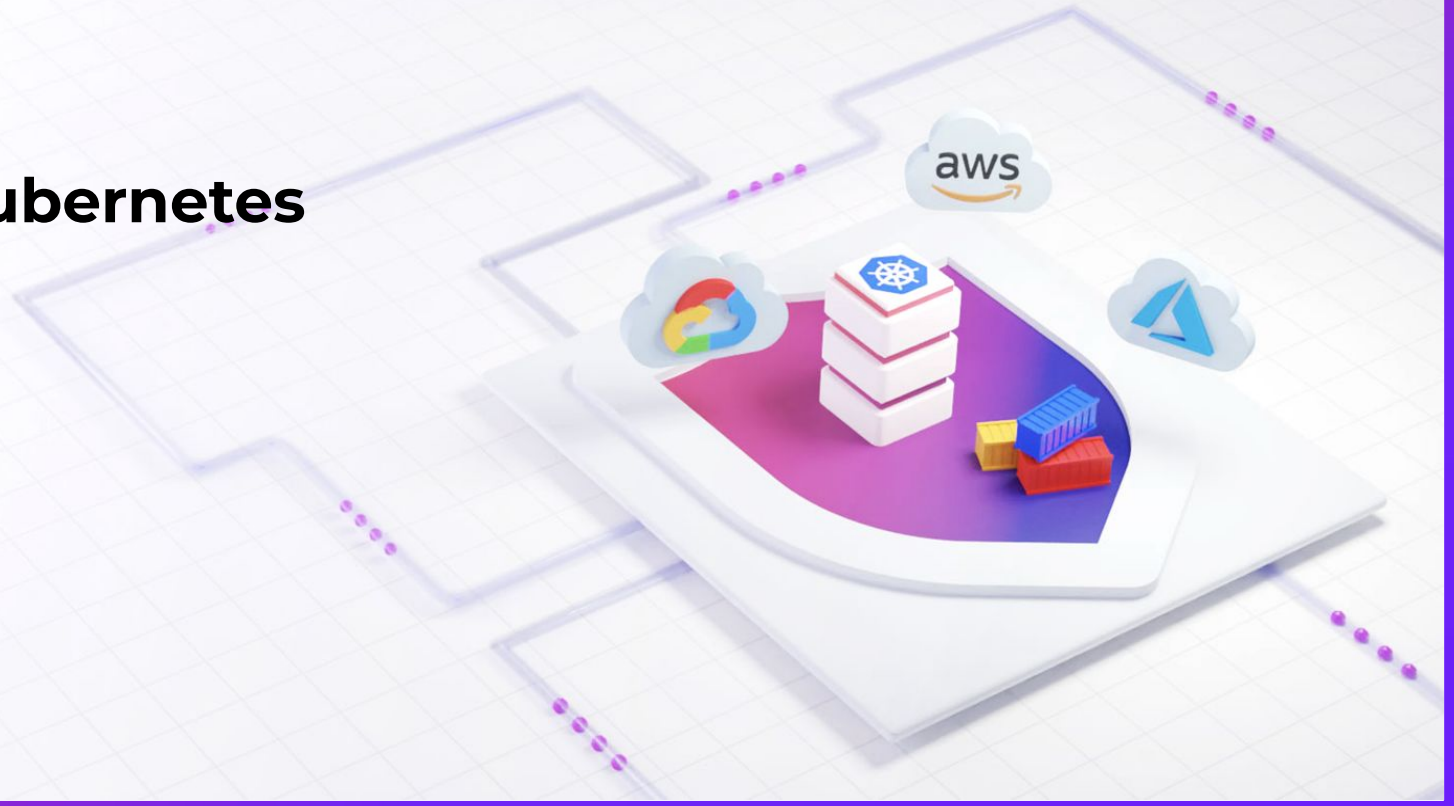
Dojo

Docker & Kubernetes



10/04/24

À PROPOS

Qui sont les profs ?

Fabio Benoit

💼Solution Architect

📩fabio.benoit@theodo.com

Ses Stats

💼 6 ans de conseil, 2 ans de SA

🏢 Oracle, Navigacom, Theodo

🏗️ Projets : UGC

Antoine ‘CHAP’ Chapusot

💼Solution Architect / Lead SRE

📩antoine.chapusot@theodo.com

🏢linkedin.com/in/antoinechapusot/

🐙github.com/ChapChap

Ses Stats

💼 10 ans d’XP SysAdmin/SRE

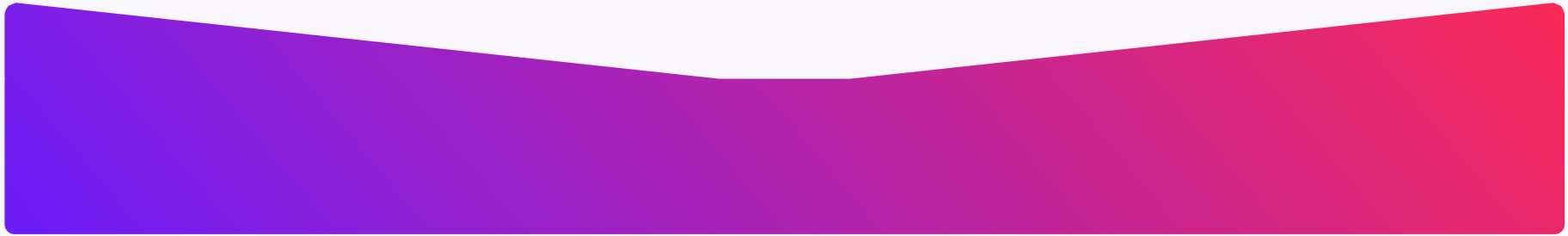
🏢 AccorHotel, BNP, Theodo

🏗️ Projets : PassCulture, MeilleursAgents, Cerballiance

SETUP

WiFi

TheodoGroup-Guest



Agenda - Overview

1. Tour de table
2. Docker
3. Architecture Kube/EKS
4. Déployer avec Helm
5. Observabilité
6. Sécurisation
7. Mise à jour
8. LoadTest & fine tuning

Objectifs - Semaine 1

Docker

- ❏ Je connais les best practices sur la conteneurisation appli
- ❏ Je sais conteneuriser une application
- ❏ Je sais conteneuriser en multi stage
- ❏ Je sais conteneuriser pour différentes infrastructures
- ❏ Je sais déployer une stack docker compose
- ❏ Je sais afficher le SBOM d'une image
- ❏ Je sais afficher l'historique d'une image
- ❏ Je sais mettre en place du hot reloading

(Optionnel) EKS

- ❏ Je comprends l'architecture d'EKS
- ❏ Je sais déployer une ECR
- ❏ Je sais déployer un LB depuis le cluster EKS
- ❏ Je sais déployer un EKS
- ❏ Je sais déployer un bastion SSM
- ❏ Je sais déployer un Cloudfront branché à EKS
- ❏ Je sais configurer cloudtrail pour EKS

Kubernetes

- ❏ Je sais ce qu'est un pod
- ❏ Je sais ce qu'est un déploiement vs statefulset vs daemonset
- ❏ Je comprends les indication min/max Surge
- ❏ Je sais ce qu'est une configmap
- ❏ Je sais ce qu'est une application stateless
- ❏ Je comprends le fonctionnement d'external-secret
- ❏ Je comprends le fonctionnement du control plane
- ❏ Je comprends le fonctionnement du scheduler
- ❏ Je comprends le templating helm
- ❏ Je comprends le patching kustomize
- ❏ Je connais le toolkit minimal d'admin k8s
- ❏ Je sais rolling update un déploiement / sts / daemonset
- ❏ Je sais importer des secrets depuis un référentiel externe
- ❏ Je sais utiliser kubectl explain
- ❏ Je sais update la valeur d'un secret
- ❏ Je sais update une configmap
- ❏ Je sais déployer une application avec Helm

Objectifs - Semaine 2

Observabilité

- ❏ Je sais quelles métriques sont importantes à observer
- ❏ Je sais consulter les métriques machines de mon EKS
- ❏ Je sais consulter les logs de mon EKS
- ❏ Je sais consulter les events kube
- ❏ Je sais consulter les consommations des pods

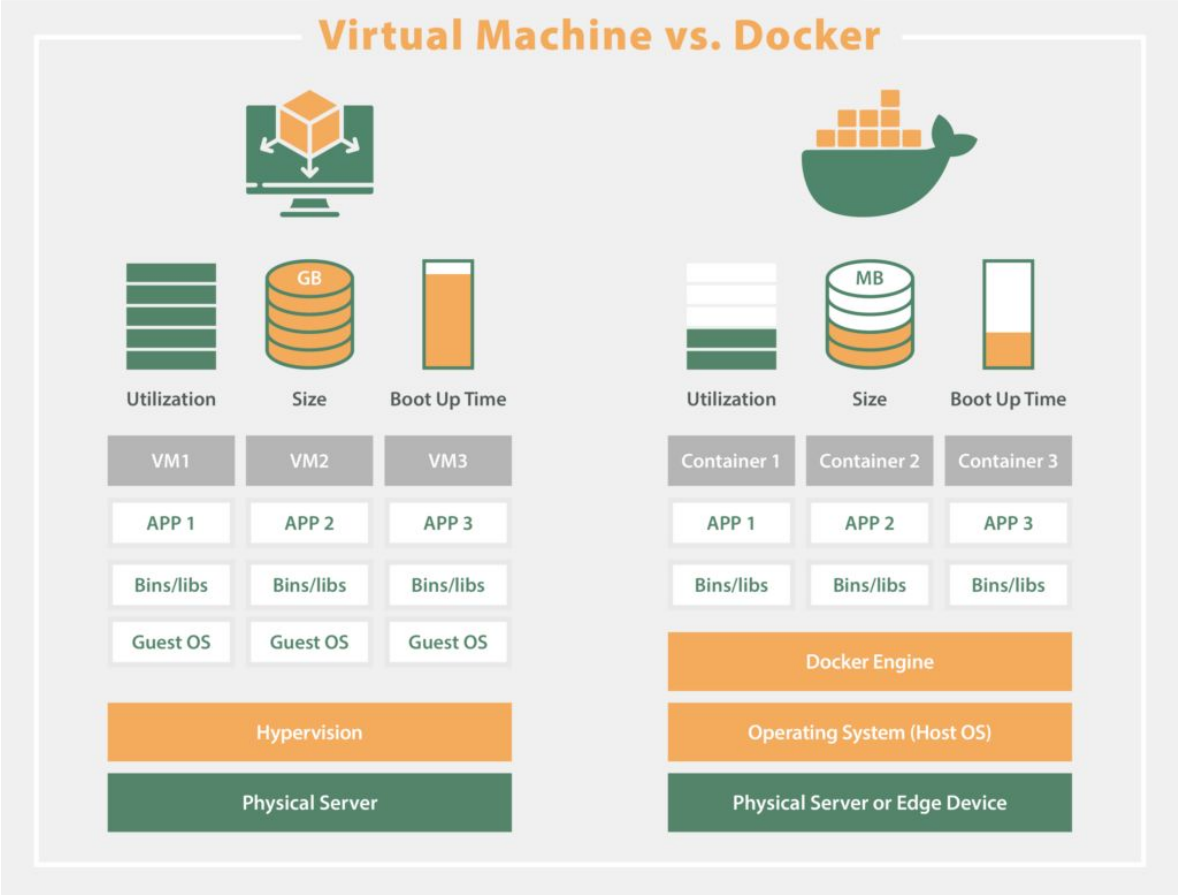
(Optionnel) Manager un cluster EKS

- ❏ Je comprends le fonctionnement de Karpenter
- ❏ Je comprends comment initialiser des droits IAM sur un cluster EKS
- ❏ Je sais mettre en place des droits RBAC limités
- ❏ Je sais sizer mes nodepools
- ❏ Je sais upgrade un cluster EKS
- ❏ Je sais identifier les incompatibilités des resource APIs avec la nouvelle version de mon cluster

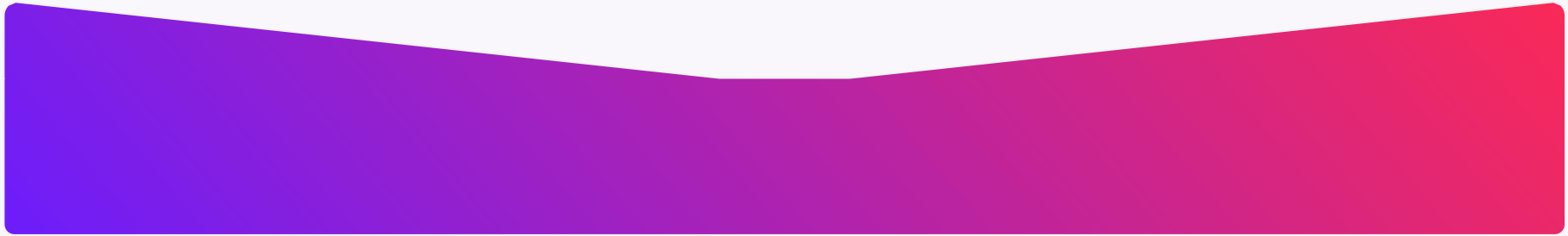
Lancement en production

- ❏ Je sais comment fine tuner un déploiement pour une application (req/lim)
- ❏ Je sais load stress une application
- ❏ Je sais déployer en mode GitOps
- ❏ Je sais faire scale mon application
- ❏ Je sais mettre en place des PDB
- ❏ Je sais isoler mes namespaces niveau réseau
- ❏ Je sais mettre en place des spread constraints
- ❏ Je sais mettre en place des affinity

Quick quiz VM vs Conteneur

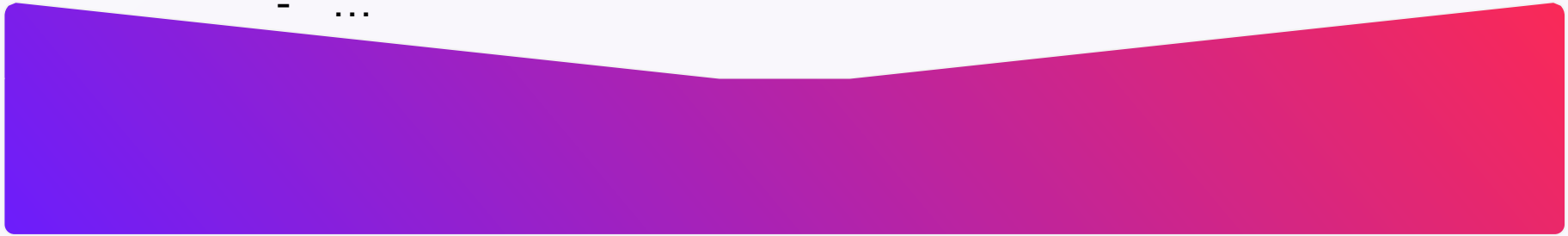

Virtual Machine vs. Docker
GB
MB
Utilization
Size
Boot Up Time
Utilization
Size
Boot Up Time
VM1
VM2
VM3
Container 1
Container 2
Container 3
APP 1
APP 2
APP 3
APP 1
APP 2
APP 3
Bins/libs
Bins/libs
Bins/libs
Bins/libs
Bins/libs
Bins/libs
Guest OS
Guest OS
Guest OS
Docker Engine
Hypervision
Operating System (Host OS)
Physical Server
Physical Server or Edge Device

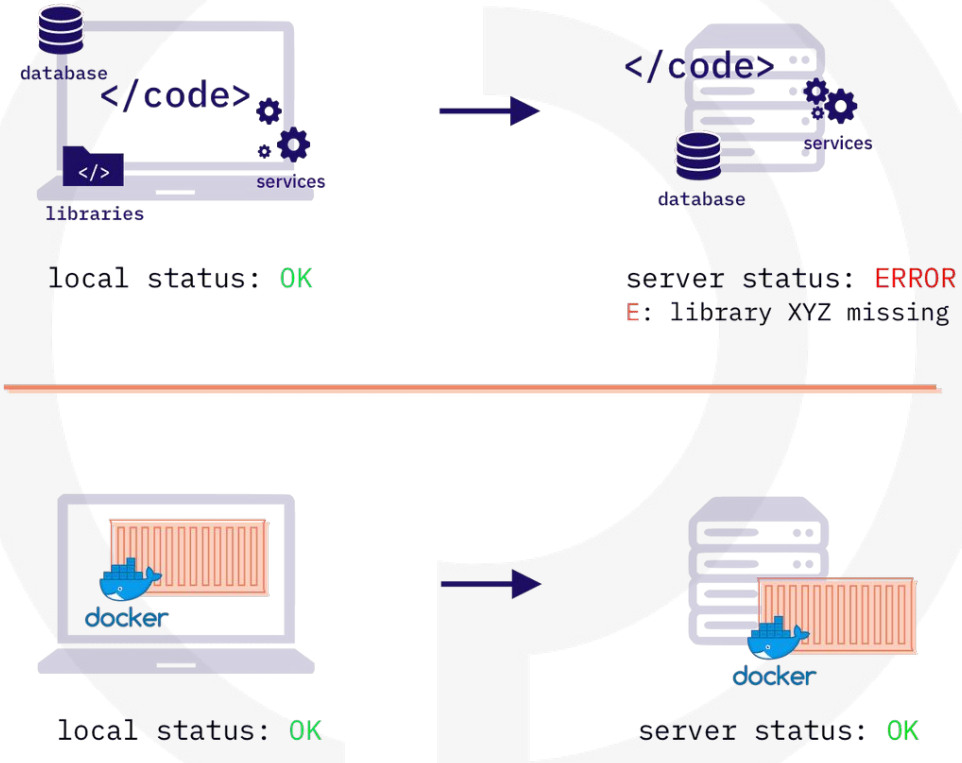
“Ça marche chez moi”



“Ça marche chez moi”

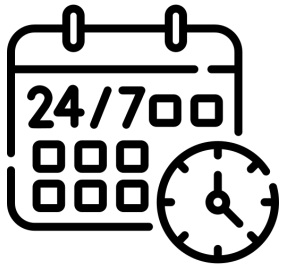
- dépendances
- OS
- versions
- ports
- secrets.
- …



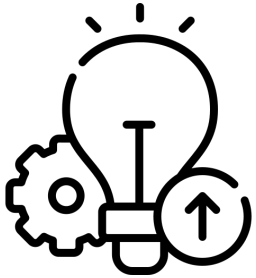

database
</code>
services
</>
libraries
local status: OK
</code>
services
database
server status: ERROR
E: library XYZ missing
docker
local status: OK
docker
server status: OK

Les 2 enjeux auxquels doivent répondre les stacks techniques des entreprises

Disponible
pour les utilisateurs
finaux



Modifiable par les
développeurs



Disponible pour les utilisateurs finaux

Résiliente - Recovery automatique
Redondance - Services redondés géographiquement (SPOFs)
Sécurisée - Surface d'attaque minimisée et connue
Performante - Rapide quelle que soit la charge
Stable - Pas d'incidents lors des mises en production

Modifiable par les développeurs

Environnement local performant et proche des environnements réels
Tests automatisés
Déploiements automatisés
Environnements de test stables

Cahier des charges d’une application

ETQApplication je dois :

- Tourner en local sur les postes des développeurs
- Être facilement et rapidement déployable
- Tourner sur les environnements réels
- Être identique (ISO) entre les environnements de test et de production
- Être redondée géographiquement
- M’adapter au trafic entrant pour garantir un certain niveau de performance
- Me réparer si la machine qui m’héberge tombe
- Générer moins de coûts que ce que je rapporte à mon entreprise

CONTENEURS & IMAGES

Static website

NGINX + modsecurity + openssl + bootstrap

User DB

PostgreSQL + PLV8 + V8

Analytics DB

Hadoop + Hive + Thrift + OpenJDK

Queue

Redis + Redis Sentinel

Background workers

Python 3.5 + celery + pyredis + libcurl
+ ffmpeg + libopencsv + nodejs + phantomjs

Web frontend

Ruby + Rails + SASS + Unicorn

API endpoint

Python 2.7 + flask + pyredis + celery
+ psycop + postgresql-client

Development VM

Production cloud

Disaster recovery

QA server

Contributor's laptop

Customer appliances

Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	Customer appliances	QA server	Disaster recovery	API endpoint	Contributor's laptop	Production cloud

Une problématique rencontrée dans l'industrie

Des biens variés

peluches, tonneaux, violons, ordinateurs, lampes, céréales...

Des infrastructures variées

bateaux, camions, grues, warehouse...

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Format standard

pour envoyer des biens


VLTU1 3 4 5 6 7
2 G 1
30.480 KG
3.700 KG
8.360 LB
26.690 KG
58.840 LB
76.4 CU.M
2.700 CU.FT
VLTU 12 34 56 7
22 G1

Standard d'API

pour livrer du code

Static website							
Web frontend							
Background workers							
User DB							
Analytics DB							
Queue							
	Development VM	Customer appliances	QA server	Disaster recovery	API endpoint	Contributor's laptop	Production cloud

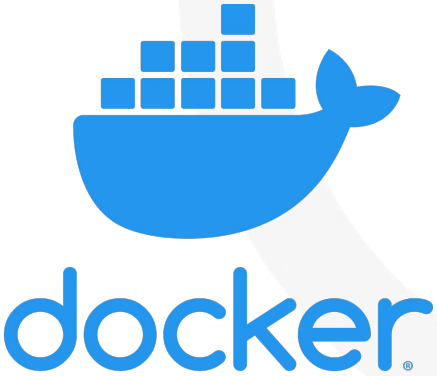
Container images

un standard d'API - les conteneurs

OPEN CONTAINER INITIATIVE

Container runtimes

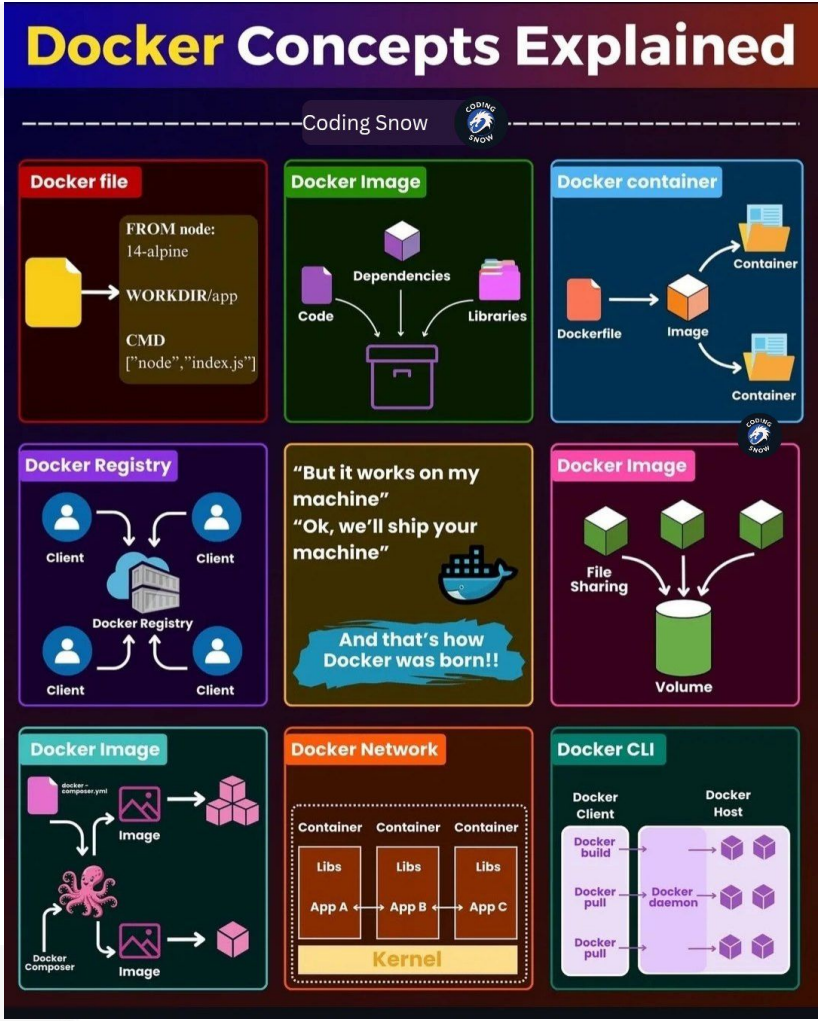
faire tourner des conteneurs dans des environnements isolés



podman

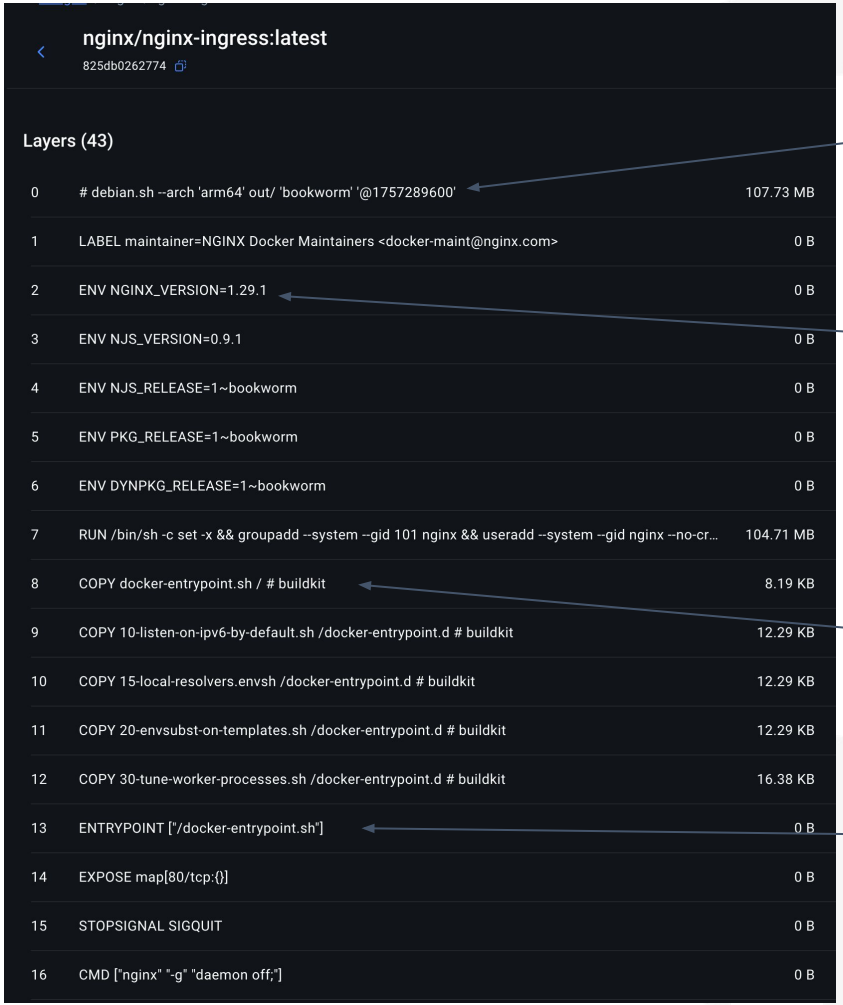
Les features qu'apportent la conteneurisation

1. Un fichier Dockerfile qui est la recette de création de l'image
2. Une image légère qui embarque code, librairies et dépendances
3. La possibilité d'instancier plusieurs fois la même image
4. Une registry qui permet le partage d'images
5. Une gestion des volumes détachées pour la persistance
6. Composer une stack (3-tier par exemple) en empilant les conteneurs
7. Une virtualisation réseau permettant la communication inter-conteneurs
8. Une CLI pour construire, exécuter et publier ses images



Conteneurs

Anatomie d’une image

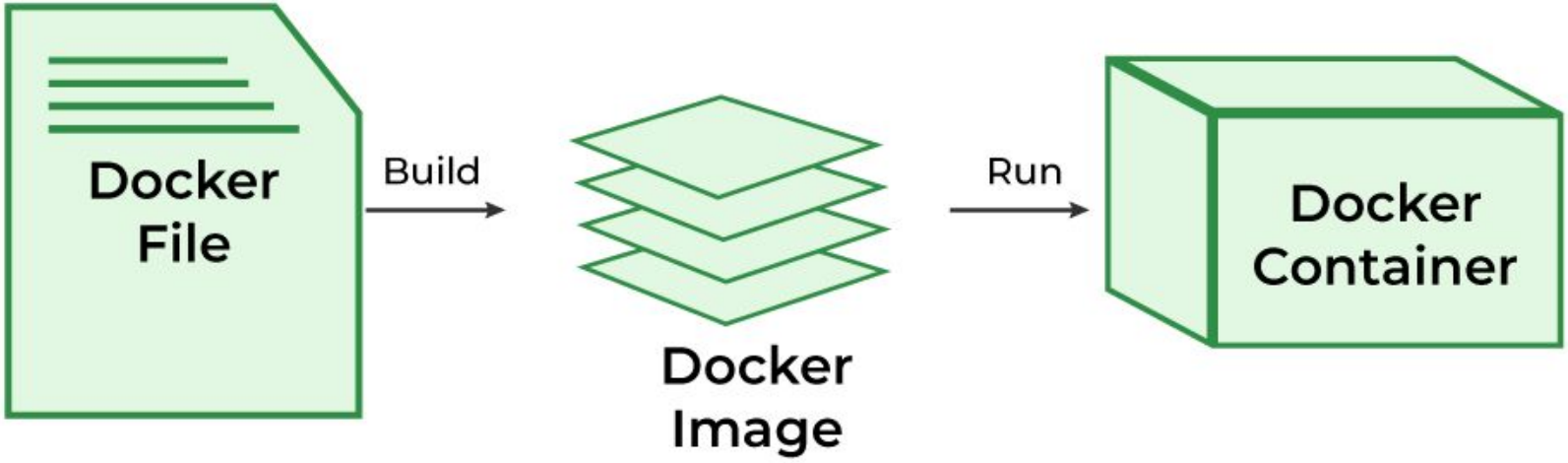


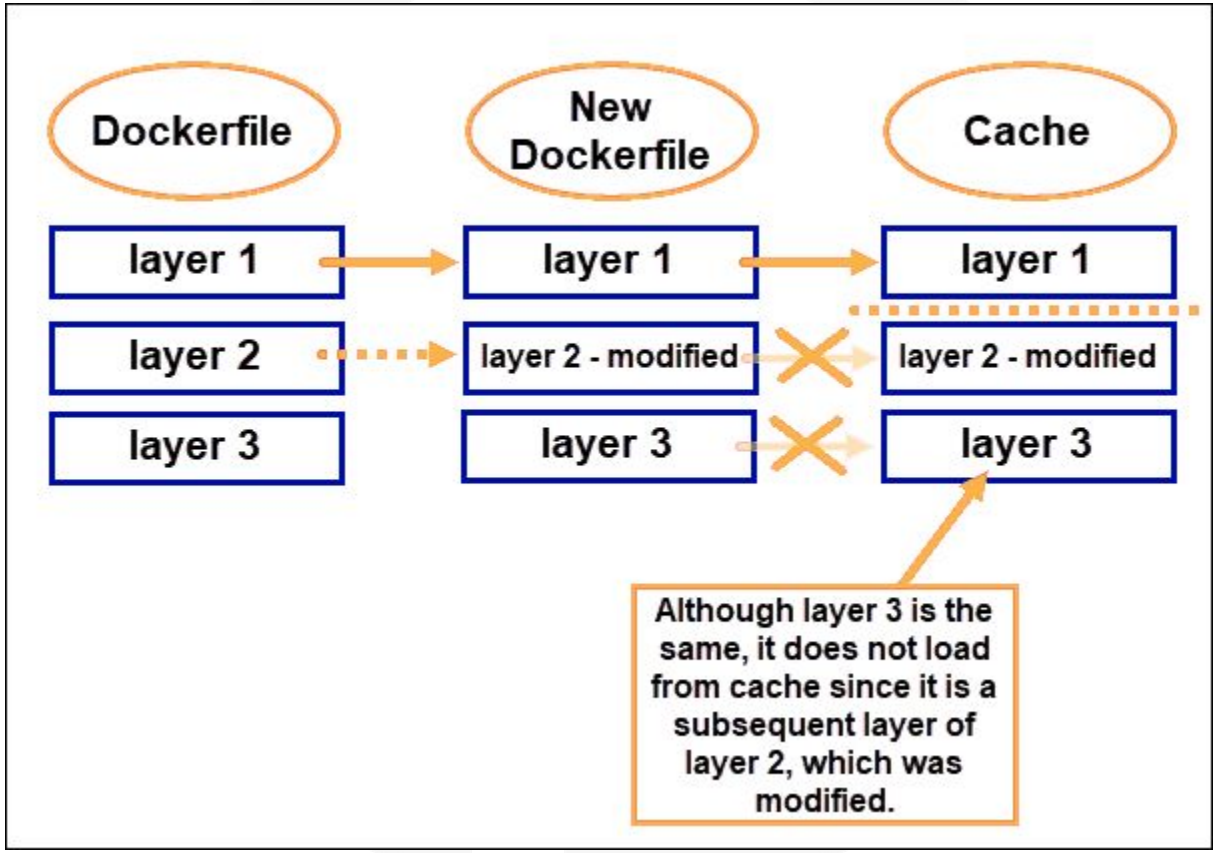
Base Image

Variables

Files added

Entry point (run command)


Docker
File
Build
Docker
Image
Run
Docker
Container


Dockerfile
New Dockerfile
Cache
layer 1
layer 2
layer 3
layer 1
layer 2 - modified
layer 3
layer 1
layer 2 - modified
layer 3
Although layer 3 is the same, it does not load from cache since it is a subsequent layer of layer 2, which was modified.

Dockerfiles

recette d’une image pour conteneur

```
# Start from an image that already contains Python 3.8.
FROM python:3.8

# Install dependencies; be explicit about versions.
RUN pip install flask==1.1

# Copy the application's source code into the image.
COPY src/ /code

# Add useful information for operators.
EXPOSE 80/tcp

# Specify how to start the application.
ENTRYPOINT python /code/main.py
```

Dockerfiles

Un bon Dockerfile

```dockerfile
# syntax=docker/dockerfile:1

# Stage 1 : build
FROM golang:1.23-alpine AS builder
# Dossier de travail
WORKDIR /src
# Copie du code source
COPY . .
# Compile le binaire
RUN go build -o hello .

# Stage 2 : runtime
FROM alpine:3.20 AS runtime
# Copie uniquement le binaire depuis le builder
COPY --from=builder /src/hello /usr/local/bin/hello
# Commande par défaut
ENTRYPOINT ["hello"]
```

Bonnes pratiques

- *multi-stage*
- *deps verrouillées*
- *user non-root*
- *bases minimales*
- *SBOM*
- *scans*

Pièges

- *Tag latest*
- *UID/GID*
- *permissions volumes*
- *secrets dans l'image*

Conteneurs

Du build local à la registry

```
# version sémantique que tu veux publier
VERSION=1.0.0

# ton registry Harbor (exemple : harbor.monentreprise.com)
REGISTRY=harbor.monentreprise.com

# ton projet Harbor (namespace)
PROJECT=demo

# nom de l'image
IMAGE=hello-multi

docker build -t ${REGISTRY}/${PROJECT}/${IMAGE}:${VERSION} .

docker inspect --format='{{index .RepoDigests 0}}'
${REGISTRY}/${PROJECT}/${IMAGE}:${VERSION}
harbor.monentreprise.com/demo/hello-multi:1.0.0@sha256:abcd1234ef567890 ...
```

Faire tourner un conteneur

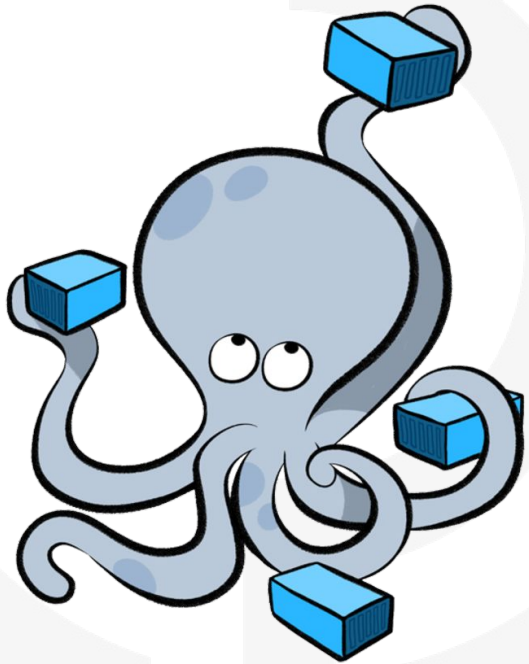
```
# Nginx qui écoute sur 8080 dans le conteneur → mappé sur 8080 local.
docker run --rm -d -p 8080:8080  nginx:alpine

# monter un répertoire local (./html) dans /usr/share/nginx/html du conteneur
docker run --rm -d -p 8080:8080 -v $(pwd)/html:/usr/share/nginx/html:ro \
  nginx:alpine

# une appli qui lit APP_ENV et SECRET_KEY
docker run --rm -it \
  -e APP_ENV=production \
  myapp:1.0.0
```

Faire tourner plusieurs conteneurs

Docker compose



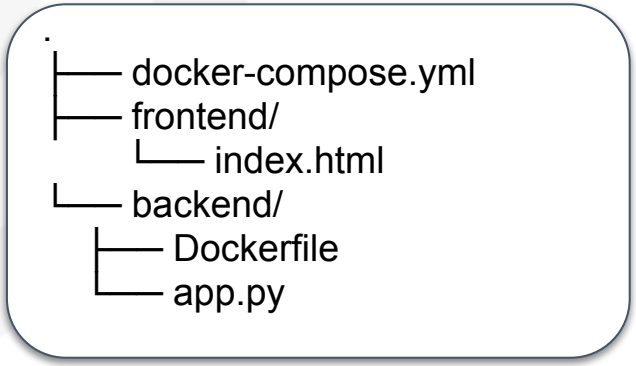
```yaml
version: "3.9"

services:
  frontend:
    image: nginx:alpine
    container_name: frontend
    ports:
      - "8080:80"
    volumes:
 # On monte un dossier local "frontend" contenant index.html, etc.
      - ./frontend:/usr/share/nginx/html:ro
    depends_on:
      - backend

  backend:
    build: ./backend   # un Dockerfile Flask dans ./backend
    container_name: backend
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/appdb
    volumes:
      # On monte le code source local pour développement
      - ./backend:/app
    ports:
      - "5000:5000"
    depends_on:
      - db

  db:
    image: postgres:16-alpine
    container_name: db
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_DB: appdb
    volumes:
      # Persistance des données PostgreSQL
      - db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

volumes:
  db_data:
```



Destination la Production

En dev

👉 *Objectif : outils + hot reload + debug*
👉 *Caractéristiques : tooling complet, volumes montés, root autorisé, taille moins critique.*

- *Monte le code source local avec -v $(pwd):/app*
- *Peut être root (pratique pour debug)*
- *Beaucoup plus lourd (pleins d'outils)*

En prod

👉 *Objectif : minimale, performante, non-root*
👉 *Caractéristiques : multi-stage, pas d'outils superflus, user dédié.*

- *Basée sur distroless (mince, pas de shell)*
- *Pas de root*
- *Pas d'outils inutiles → surface d'attaque réduite*

```
# On construit l'image pour toutes les architectures
docker buildx build \
  --platform linux/amd64,linux/arm64 \
  -t myregistry.com/project/myapp:1.0.0 \
  --push \
  .

# On vérifie le digest
docker buildx imagetools inspect
myregistry.com/project/myapp:1.0.0
```

Cahier des charges d’une application

ETQApplication je dois :

- **Tourner en local sur les postes des développeurs**
- **Être facilement et rapidement déployable**
- **Tourner sur les environnements réels**
- **Être identique entre les environnements de test et de production**
- Être redondée géographiquement
- M’adapter au traffic entrant pour garantir un certain niveau de performance
- Me réparer si la machine qui m’héberge tombe
- Générer moins de coûts que ce que je rapporte à mon entreprise

Le dojo

Time to practice!

https://github.com/padok-team/dojo-kubernetes-eks/tree/main/Docker

Kubernetes

Faire tourner des conteneurs à l'échelle

c'est compliqué...

Gestion du lifecycle

self-healing, rolling-updates

=

compliqué

Résilience

plusieurs instances

=

load balancing + déploiements multi-instances

=

compliqué

Scalabilité

nombre d’instances variable

=

load balancing + gestion de l’ajout/destruction

=

compliqué

Mars 2014 chez Google

Craig McLuckie, Joe Beda et Brendan Burns

Kubernetes

= timonier en 🇬🇷

Qu’est-ce que k8s ?

Un orchestrateur de conteneurs

Une plateforme extensible

Une abstraction entre dev et ops

Une API

Qu’est-ce que fait k8s ?

Déploie des applications conteneurisées sur un grand nombre de noeuds

Automatise les rolling updates

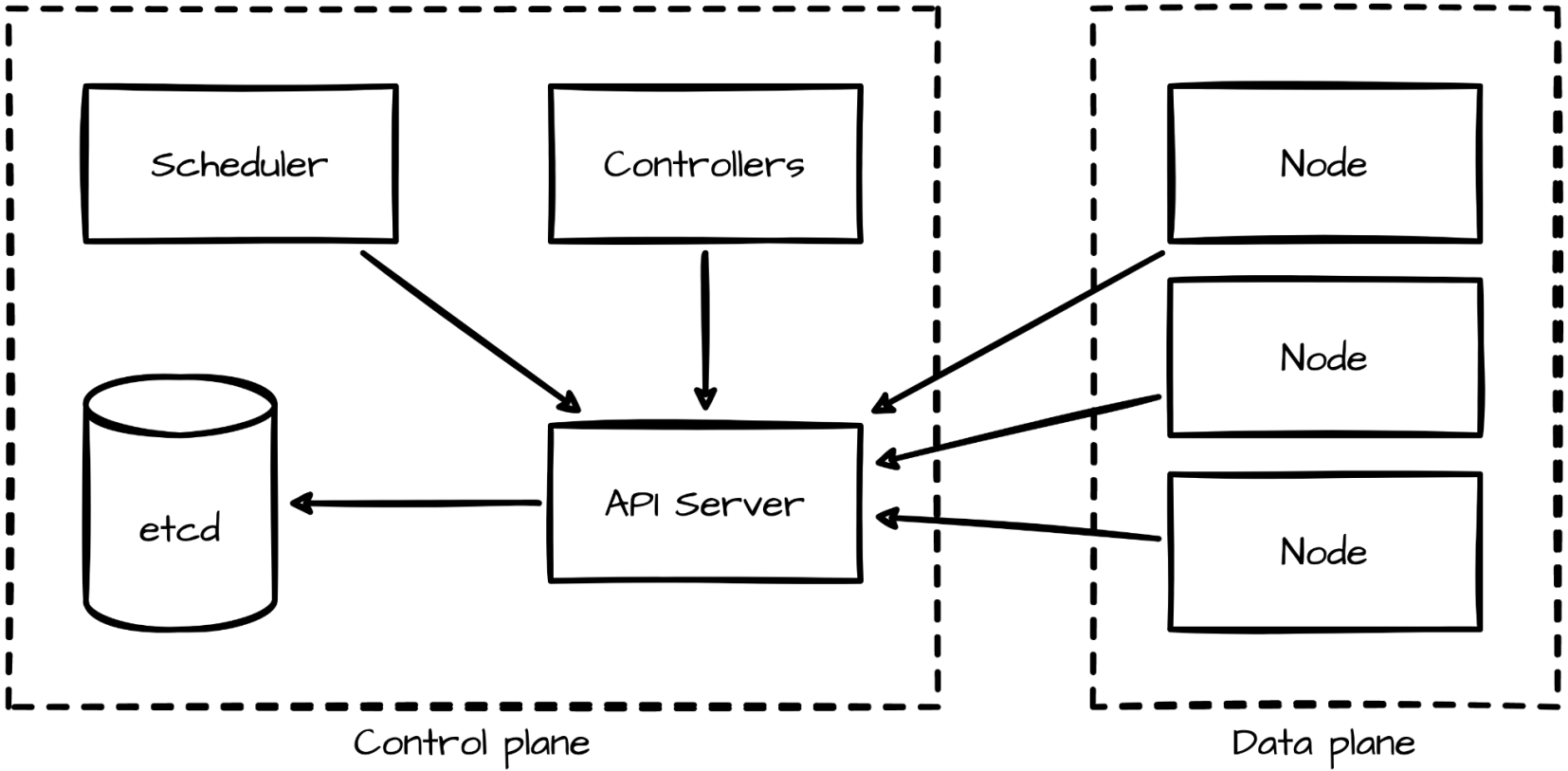
Détecte les incidents et les répare

Scale automatiquement en fonction de métriques sur-mesure

Load balance les conteneurs

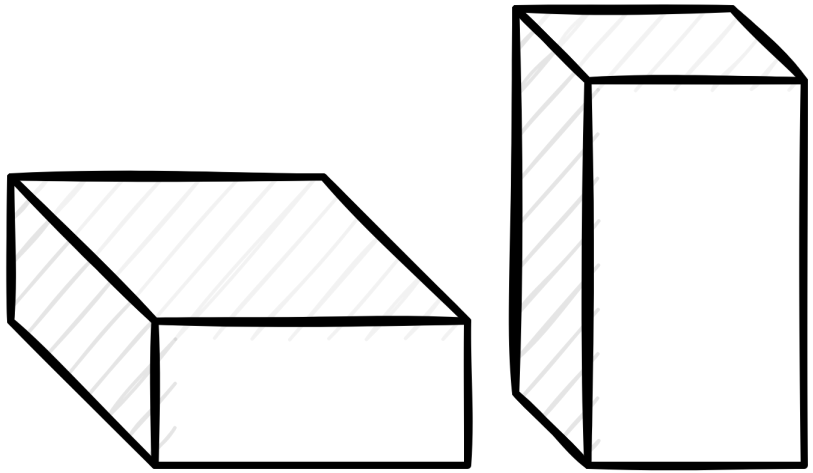
Qu’est-ce qu’un noeud ?

VM connectée au cluster Kubernetes avec des attributs (CPU, mémoire, zone géographique)


Scheduler
Controllers
Node
Node
Node
etcd
API Server
Control plane
Data plane

Kubernetes resources

pods + deployments + ...



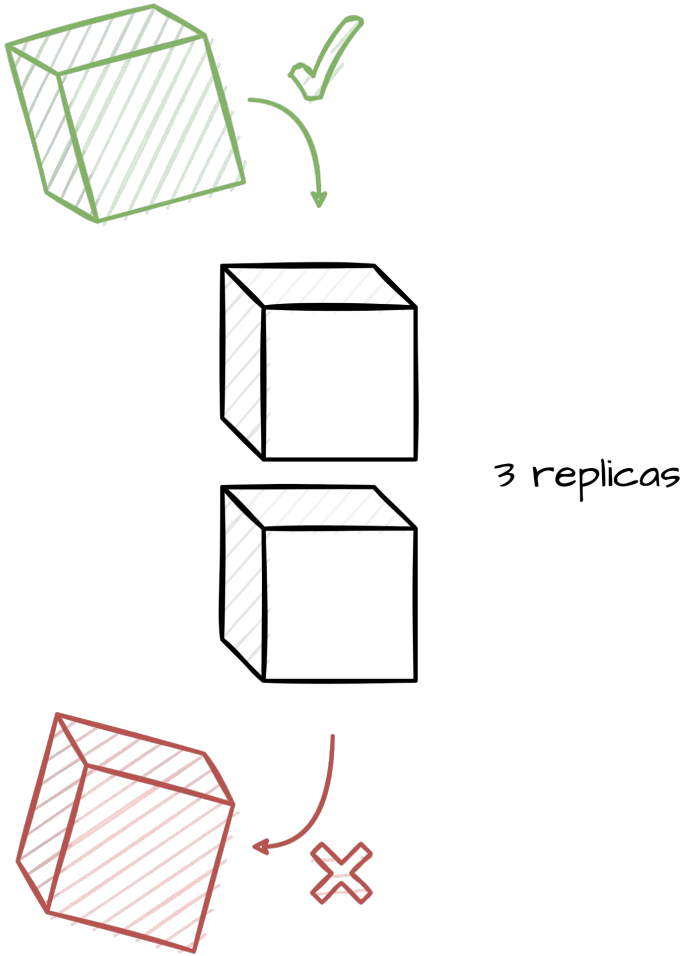
Pods

basic unit for running containers

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  labels:
    foo: bar
spec:
  containers:
    - name: my-container
      image: myapp:v1.0.0
      command: ['/bin/my-app']
      args: ['--migrate-db', '--db-host=12.34.56.78']
```

ReplicaSets

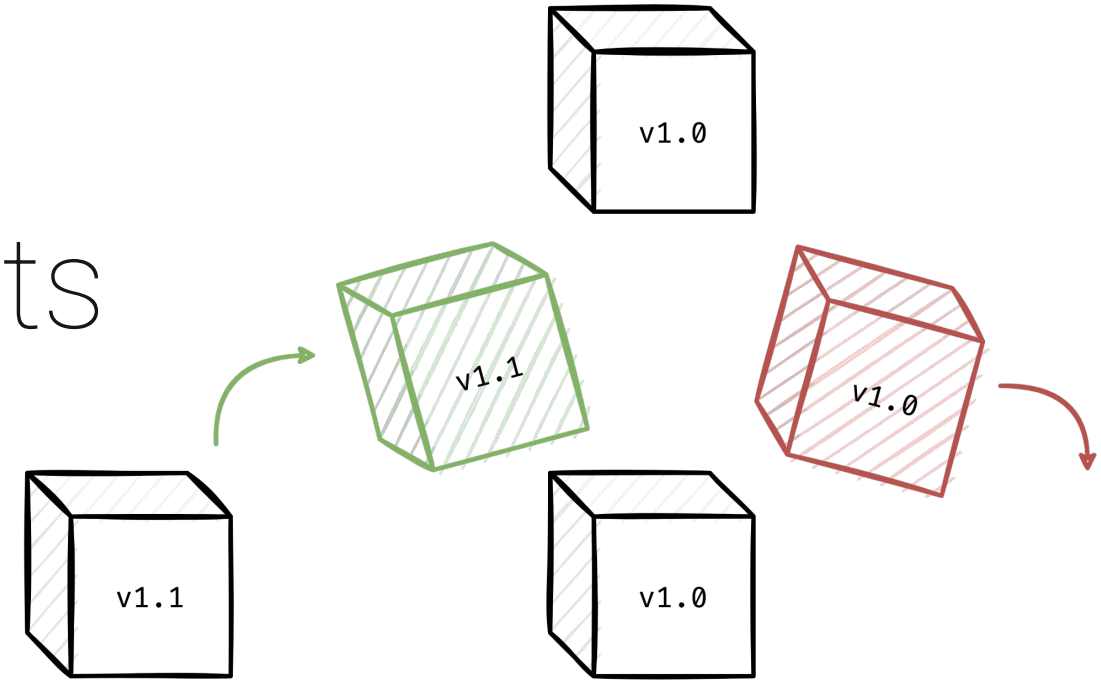
identical Pods



```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      foo: bar
  template:
    metadata:
      labels:
        foo: bar
    spec:
      containers:
        - name: my-container
          image: myapp:v1.0.0
          ports:
            - containerPort: 3000
```

Deployments

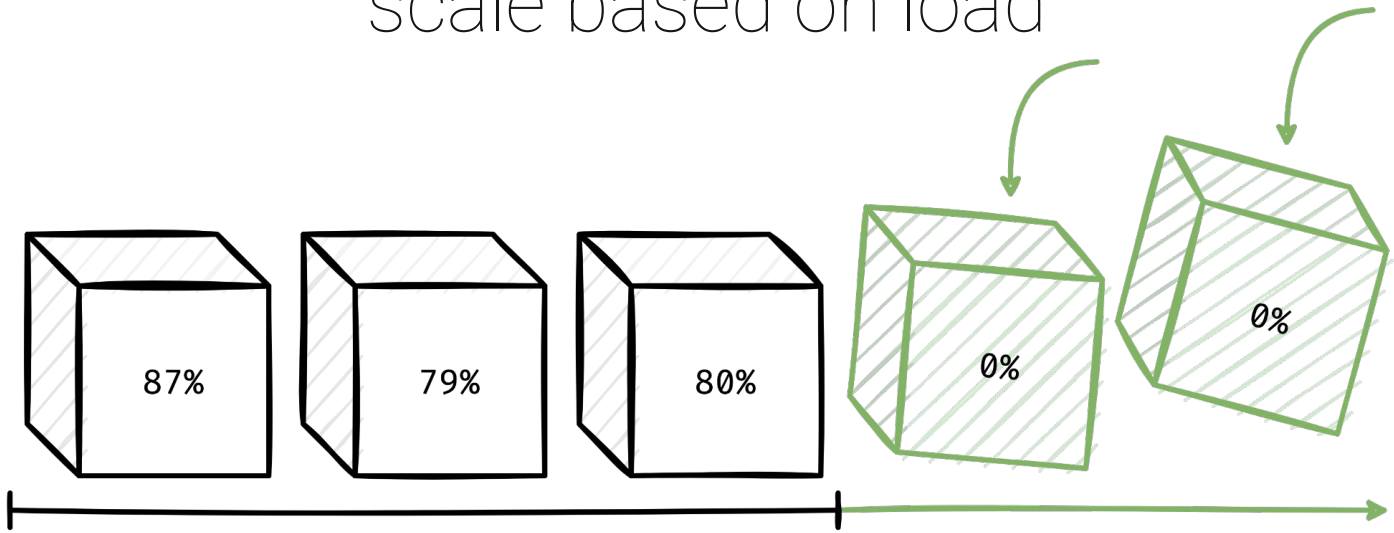
no downtime



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      foo: bar
  template:
    metadata:
      labels:
        foo: bar
    spec:
      containers:
        - name: my-container
          image: myapp:v1.0.0
          ports:
            - containerPort: 3000
```

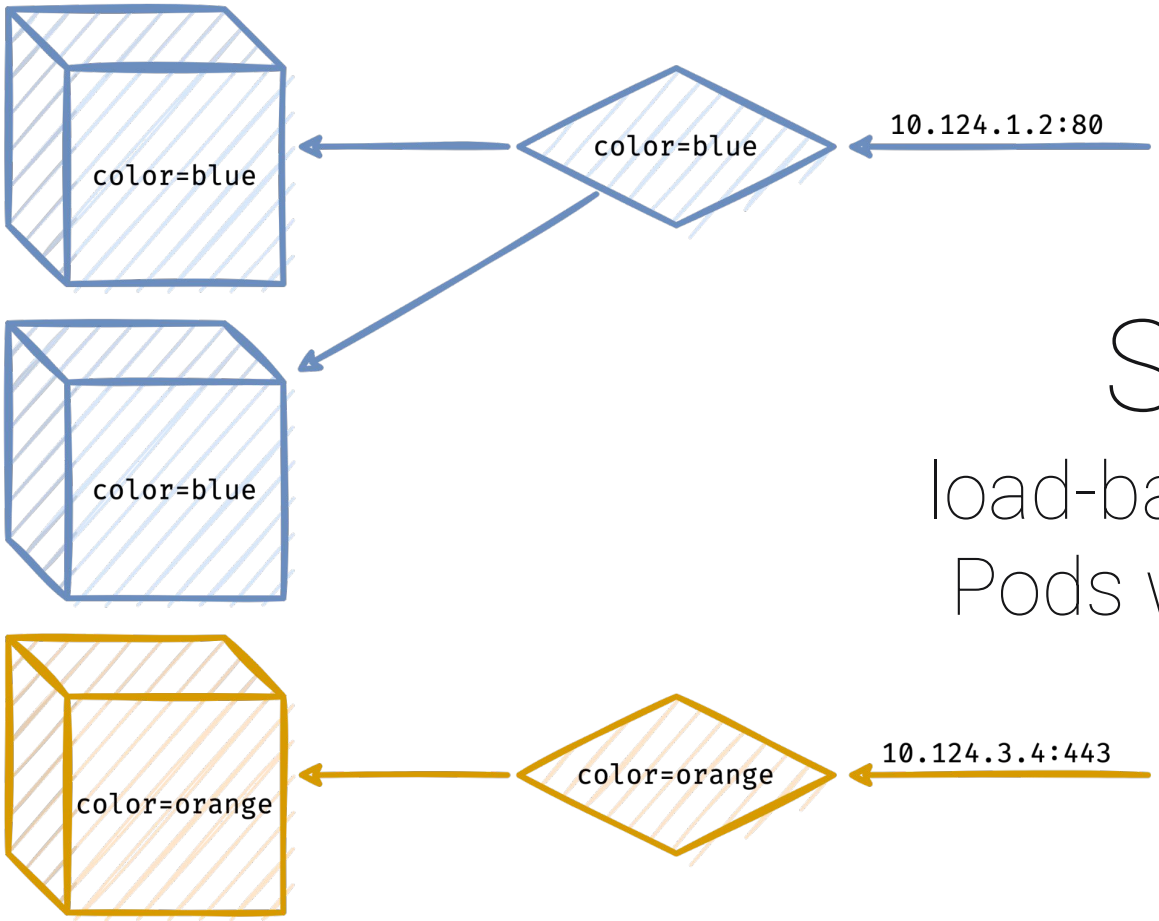
HorizontalPodAutoscalers

scale based on load



target: 60%
average: 82% → 49.2%

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 60
```



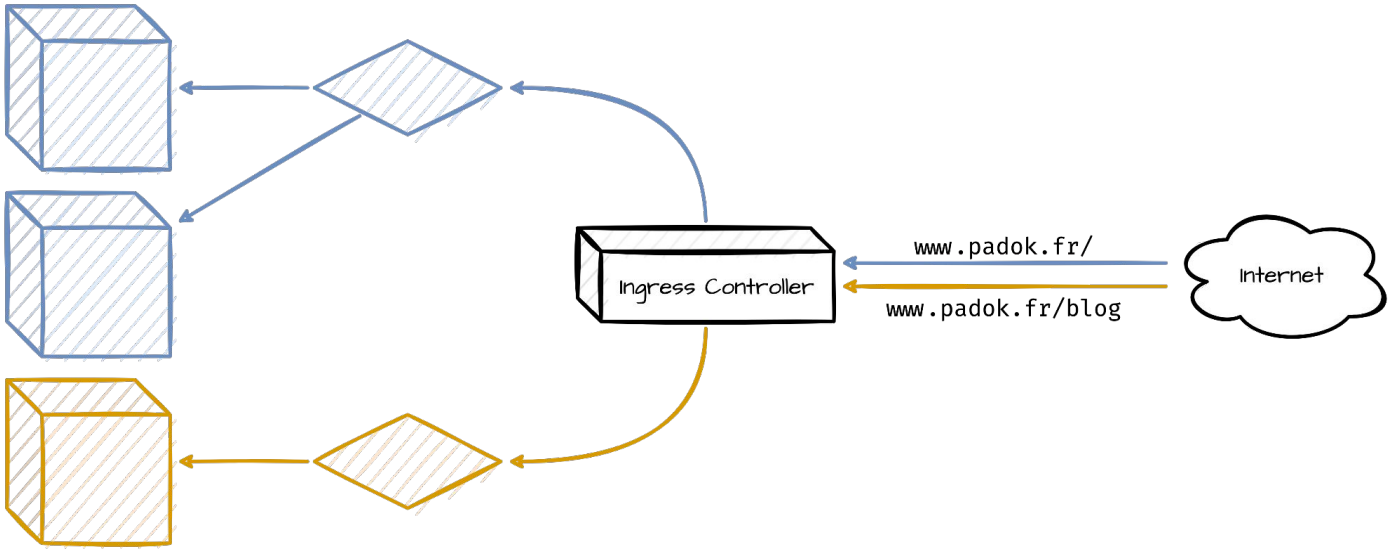
Services

load-balancing between Pods with same labels

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    foo: bar
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Ingresses

route requests that come from the outside



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: www.padok.fr
      http:
        paths:
          - path: /blog
            pathType: Prefix
            backend:
              service:
                name: my-service
                port:
                  number: 80
```

Cahier des charges d’une application

ETQApplication je dois :

- **Tourner en local sur les postes des développeurs**
- **Être facilement et rapidement déployable**
- **Tourner sur les environnements réels**
- **Être identique entre les environnements de test et de production**
- **Être redondée géographiquement**
- **M’adapter au traffic entrant pour garantir un certain niveau de performance**
- **Me réparer si la machine qui m’héberge tombe**
- **Générer moins de coûts que ce que je rapporte à mon entreprise**

Best Practices

Disponibilité

Définir des probes correctes /healthz

Définir des requests / limits

Sécurité

Pas de mots de passe en clair

Isoler le trafic entre namespaces

Moindre privilèges pour les conteneurs

Scalabilité

Activer des HPA

Déployer des applications stateless

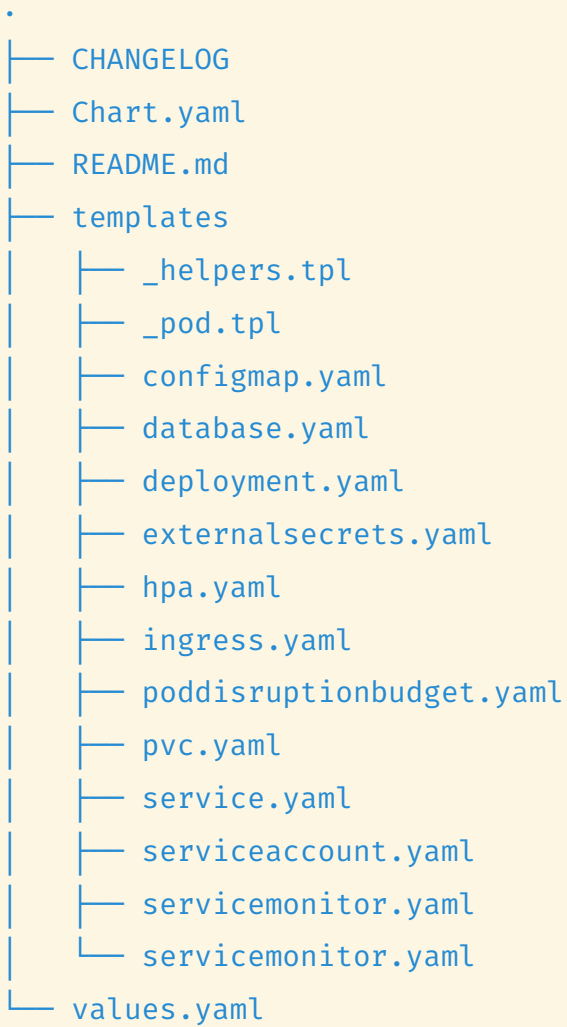
Maintenabilité

Séparer code et config

Exposer des métriques

Helm

Deploy all these YAML files


.
├── CHANGELOG
├── Chart.yaml
├── README.md
├── templates
│   ├── _helpers.tpl
│   ├── _pod.tpl
│   ├── configmap.yaml
│   ├── database.yaml
│   ├── deployment.yaml
│   ├── externalsecrets.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── poddisruptionbudget.yaml
│   ├── pvc.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   ├── servicemonitor.yaml
│   └── servicemonitor.yaml
└── values.yaml

```
helm install nginx oci://ghcr.io/nginx/charts/nginx-ingress --version 2.3.0
```

```
22:41 ❯ helm list -A
NAME                NAMESPACE         REVISION  UPDATED                                  STATUS    CHART                     APP VERSION
argocd              argocd            1         2025-07-08 16:52:22.397491628 +0200 CEST deployed  argocd-8.0.10
cilium              kube-system       2         2025-07-11 15:01:27.224542 +0200 CEST    deployed  cilium-1.16.7             1.16.7
coredns             kube-system       1         2025-07-08 10:25:01.773316593 +0000 UTC  deployed  coredns-1.39.1            1.12.0
external-secrets    external-secrets  1         2025-07-08 15:54:44.97293825 +0200 CEST  deployed  external-secrets-0.12.1
metrics-server      kube-system       1         2025-07-08 10:25:01.740777111 +0000 UTC  deployed  metrics-server-3.12.2     0.7.2
osc-bsu-csi-driver  kube-system       1         2025-07-08 10:25:20.302795975 +0000 UTC  deployed  osc-bsu-csi-driver-1.7.0  v1.5.0
```

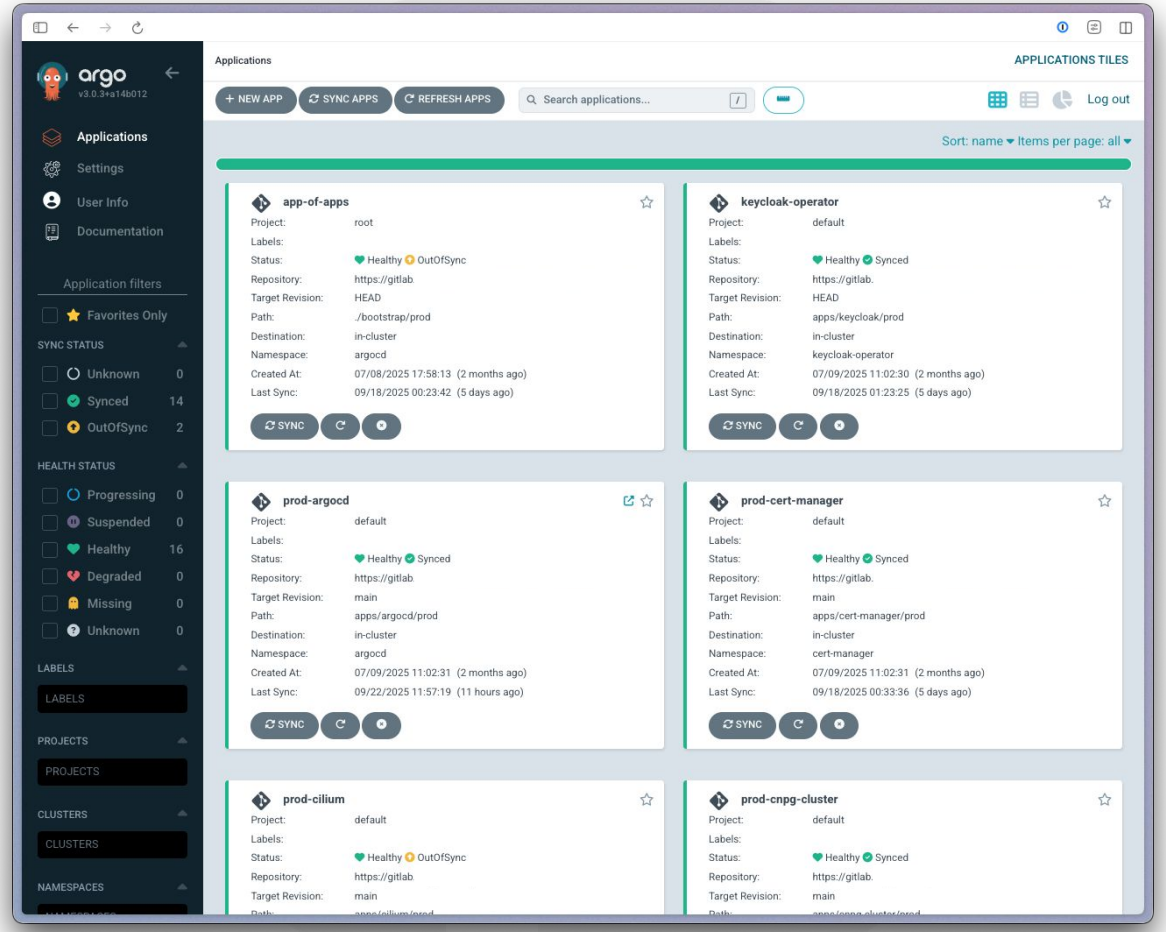
Qu’est-ce qu’on a ?

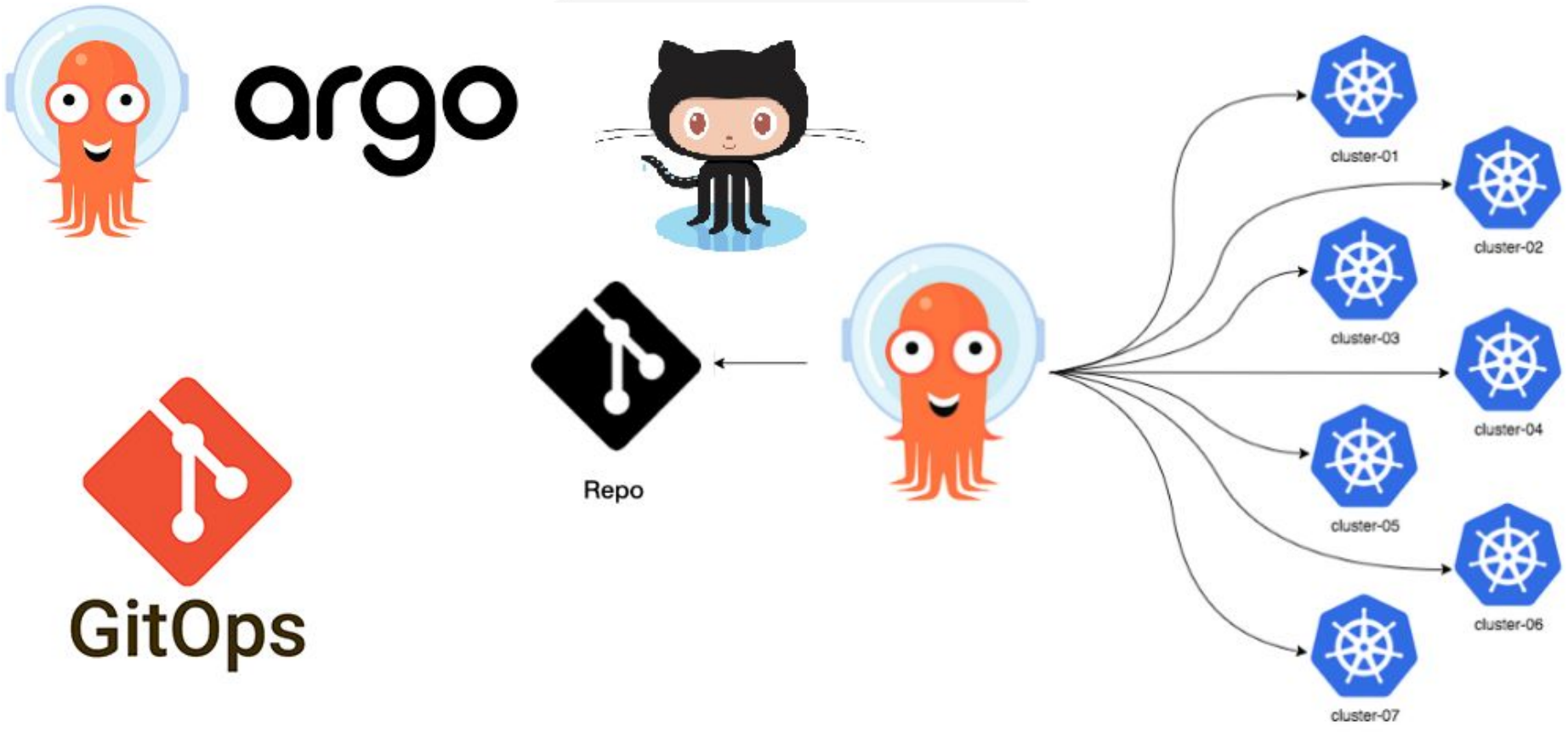
1. Un (ou plusieurs) conteneur(s) qui package mon application pour la faire tourner ailleurs que sur ma machine
2. Une plateforme extensible qui orchestre les conteneurs de mon application
3. Une solution qui package les objets supplémentaires à mon application

Qui déploie ?

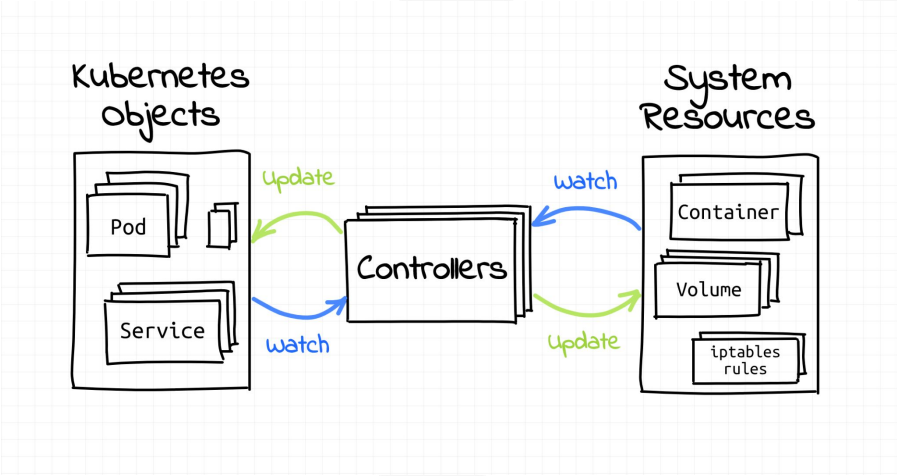
ArgoCD !

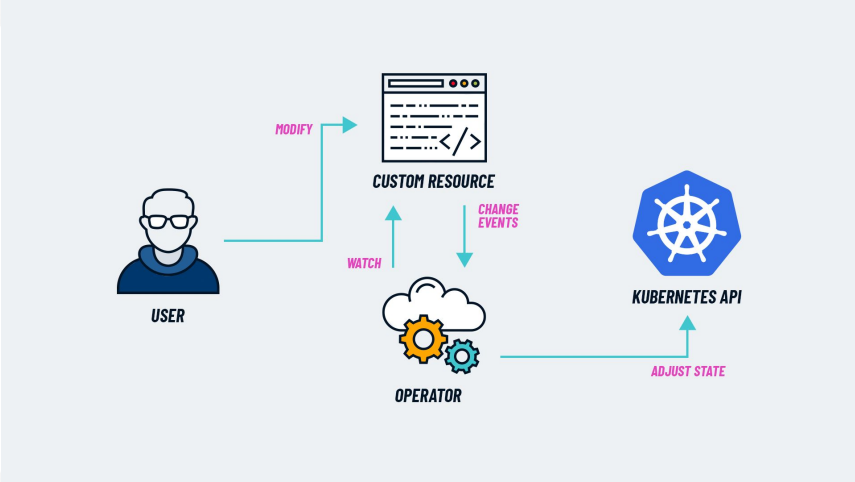
Le GitOps


argo
v3.0.3+a14b012
Applications
Settings
User Info
Documentation
Application filters
Favorites Only
SYNC STATUS
Unknown 0
Synced 14
OutOfSync 2
HEALTH STATUS
Progressing 0
Suspended 0
Healthy 16
Degraded 0
Missing 0
Unknown 0
LABELS
LABELS
PROJECTS
PROJECTS
CLUSTERS
CLUSTERS
NAMESPACES
Applications
APPLICATIONS TILES
+ NEW APP
SYNC APPS
REFRESH APPS
Search applications...
Log out
Sort: name
Items per page: all
app-of-apps
Project: root
Labels:
Status: Healthy OutOfSync
Repository: https://gitlab
Target Revision: HEAD
Path: ./bootstrap/prod
Destination: in-cluster
Namespace: argocd
Created At: 07/08/2025 17:58:13 (2 months ago)
Last Sync: 09/18/2025 00:23:42 (5 days ago)
SYNC
keycloak-operator
Project: default
Labels:
Status: Healthy Synced
Repository: https://gitlab.
Target Revision: HEAD
Path: apps/keycloak/prod
Destination: in-cluster
Namespace: keycloak-operator
Created At: 07/09/2025 11:02:30 (2 months ago)
Last Sync: 09/18/2025 01:23:25 (5 days ago)
SYNC
prod-argocd
Project: default
Labels:
Status: Healthy Synced
Repository: https://gitlab
Target Revision: main
Path: apps/argocd/prod
Destination: in-cluster
Namespace: argocd
Created At: 07/09/2025 11:02:31 (2 months ago)
Last Sync: 09/22/2025 11:57:19 (11 hours ago)
SYNC
prod-cert-manager
Project: default
Labels:
Status: Healthy Synced
Repository: https://gitlab.
Target Revision: main
Path: apps/cert-manager/prod
Destination: in-cluster
Namespace: cert-manager
Created At: 07/09/2025 11:02:31 (2 months ago)
Last Sync: 09/18/2025 00:33:36 (5 days ago)
SYNC
prod-cilium
Project: default
Labels:
Status: Healthy OutOfSync
Repository: https://gitlab
Target Revision: main
prod-cnpg-cluster
Project: default
Labels:
Status: Healthy Synced
Repository: https://gitlab.
Target Revision: main


argo
Repo
GitOps
cluster-01
cluster-02
cluster-03
cluster-04
cluster-05
cluster-06
cluster-07

Kubernetes - Controllers / Operators

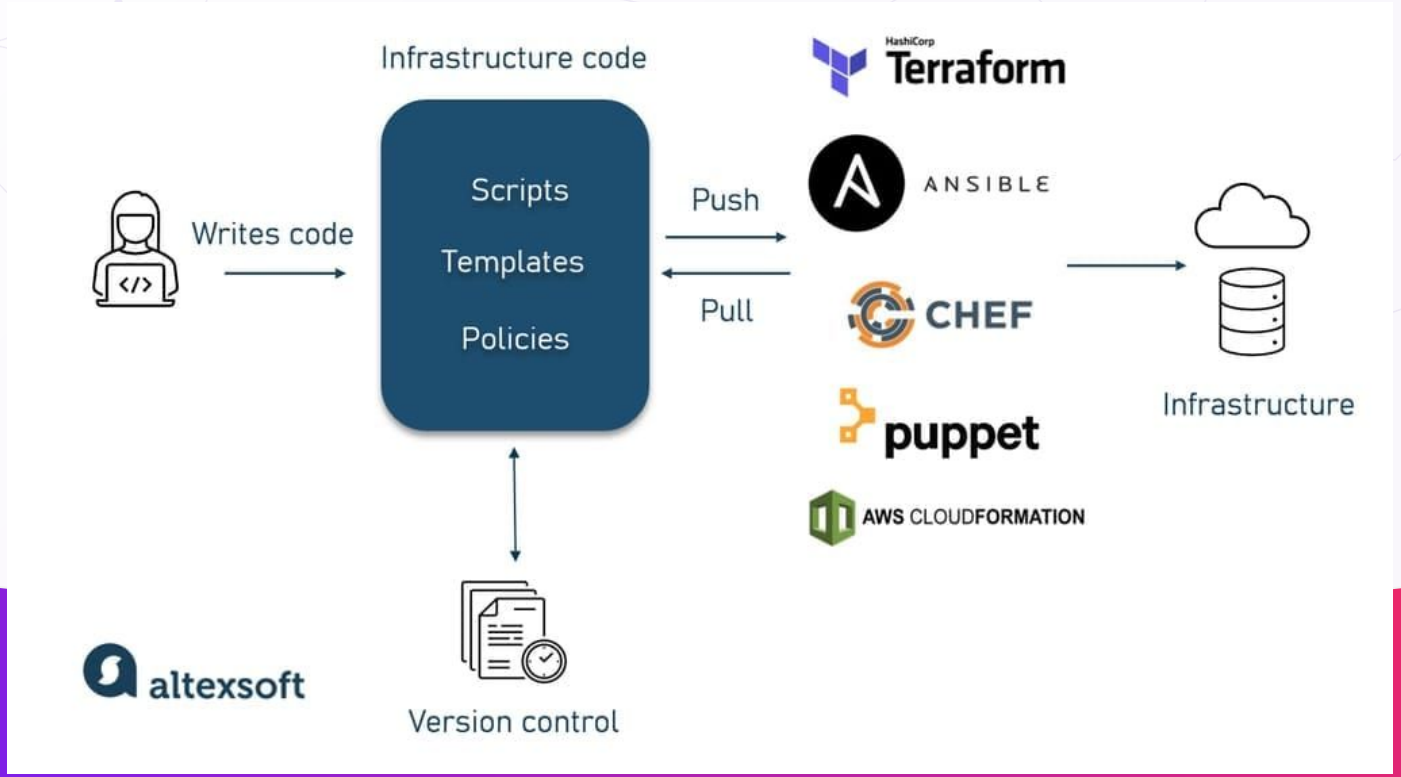




Le dojo

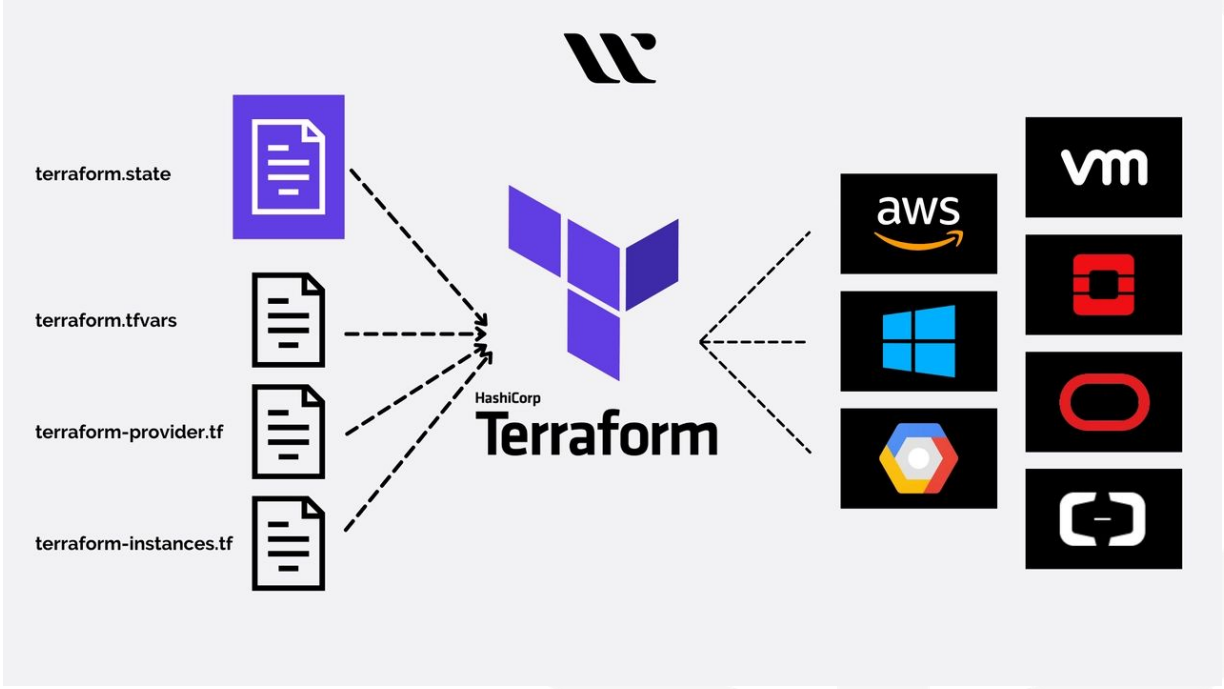
EKS
et son déploiement IaC via terraform

Terraform - l'Infrastructure as Code



CULTURE & FONDATIONS DEVOPS/SRE

l'Infrastructure as Code - Terraform

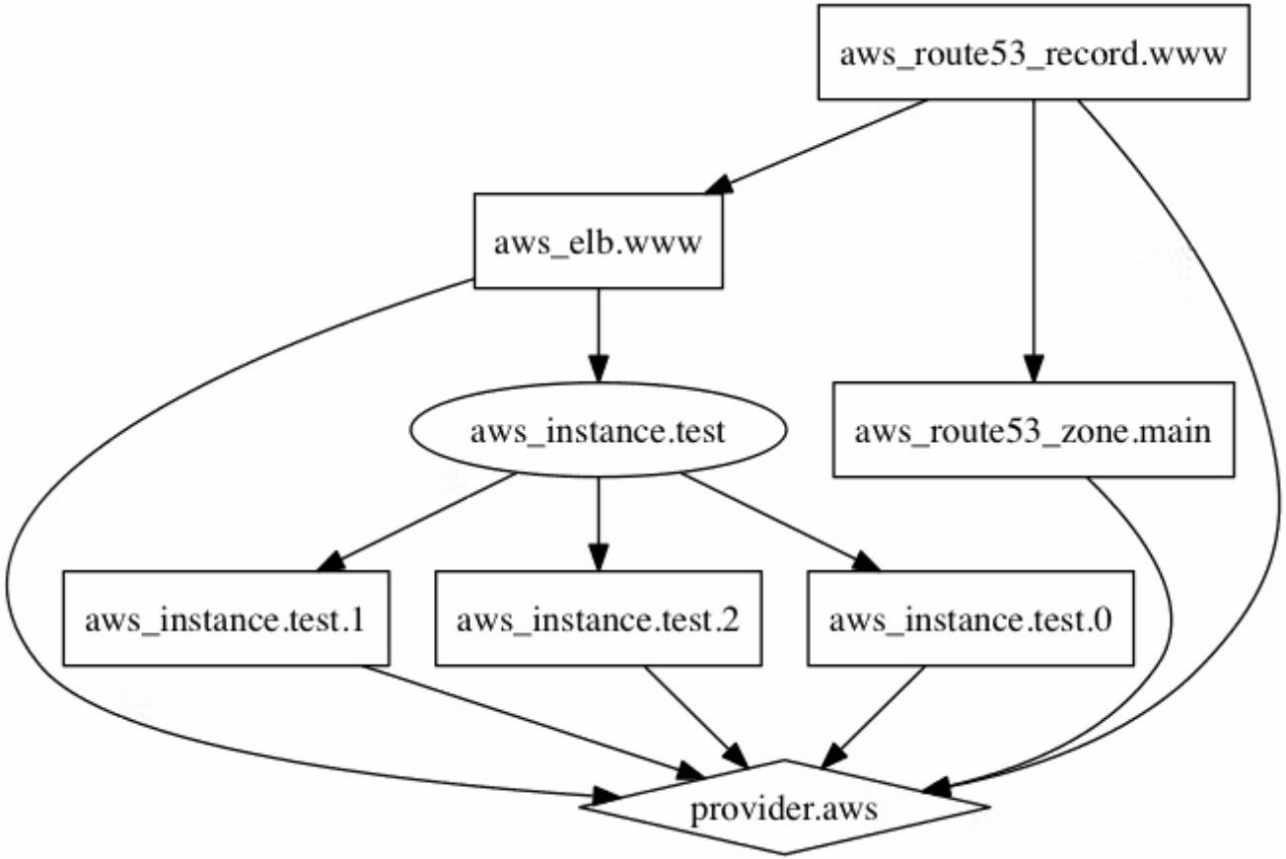


✅ Écrit en Go

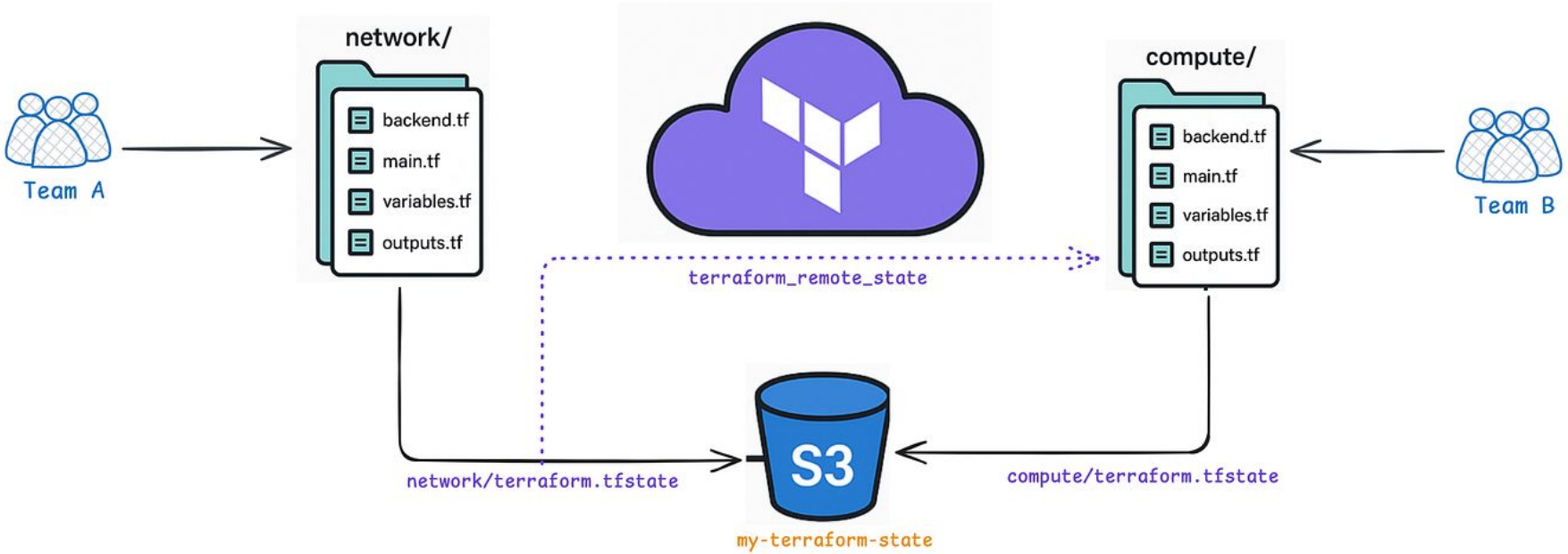
✅ Évolutif

✅ Gère les dépendances

l'Infrastructure as Code - Terraform



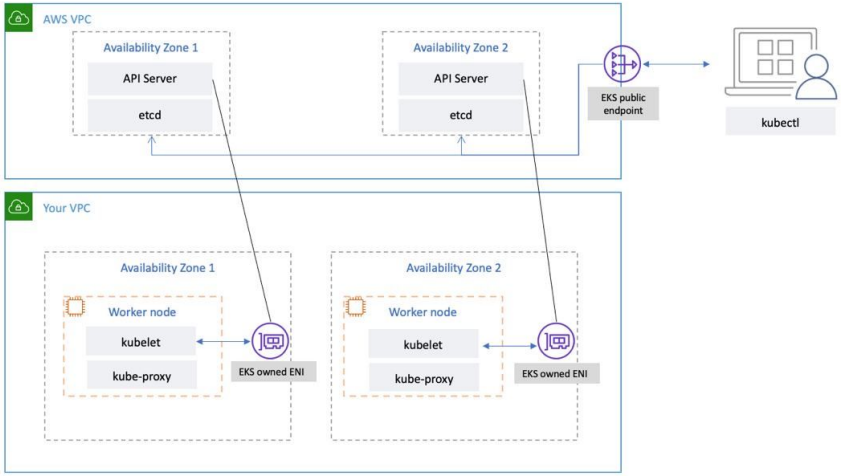
l'Infrastructure as Code - Terraform



EKS - l'implémentation AWS de Kubernetes

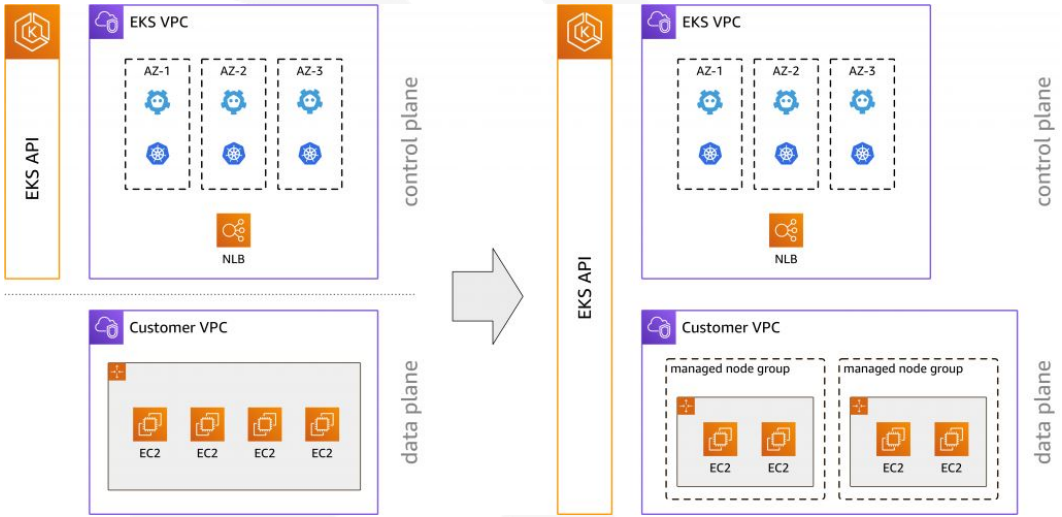
- **Plan de contrôle managé** (HA multi-AZ, patchs/MAJ par AWS) ; vous gérez surtout les **nœuds** et les **add-ons**. Les mises à niveau se font **control plane → nœuds** (rolling).

- **EKS Add-ons**: VPC CNI, CoreDNS, kube-proxy, EBS/EFS/FSx CSI… gérés via EKS (console/CLI/eksctl/Terraform).



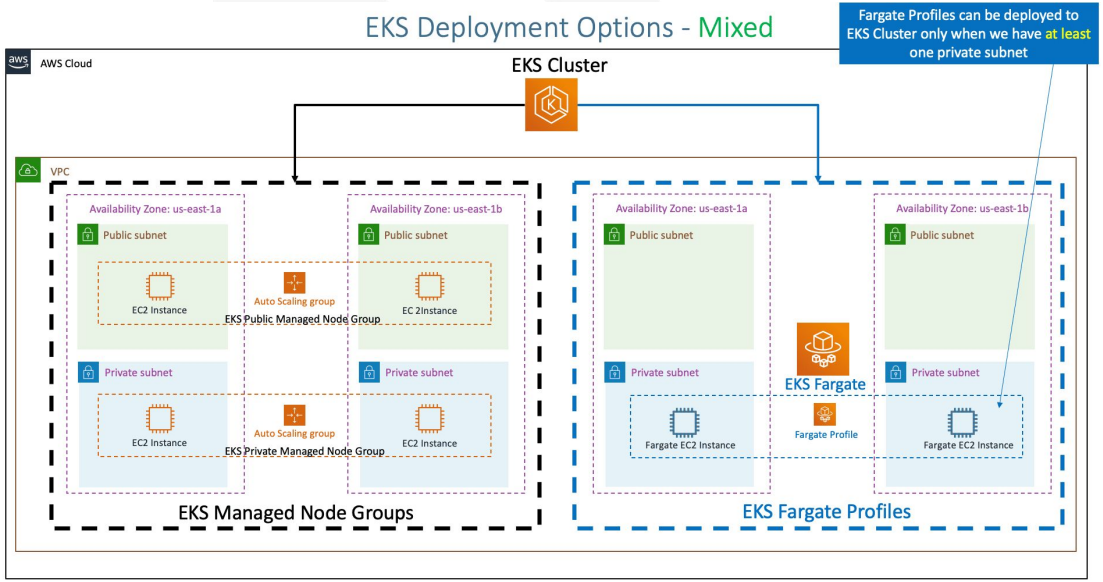
EKS - l'implémentation AWS de Kubernetes

- **Managed Node Groups (MNG)** : AWS gère le cycle de vie (création, update, drain/cordon, remplacement AMI). Pas de coût EKS supplémentaire au-delà des ressources EC2/EBS/etc.



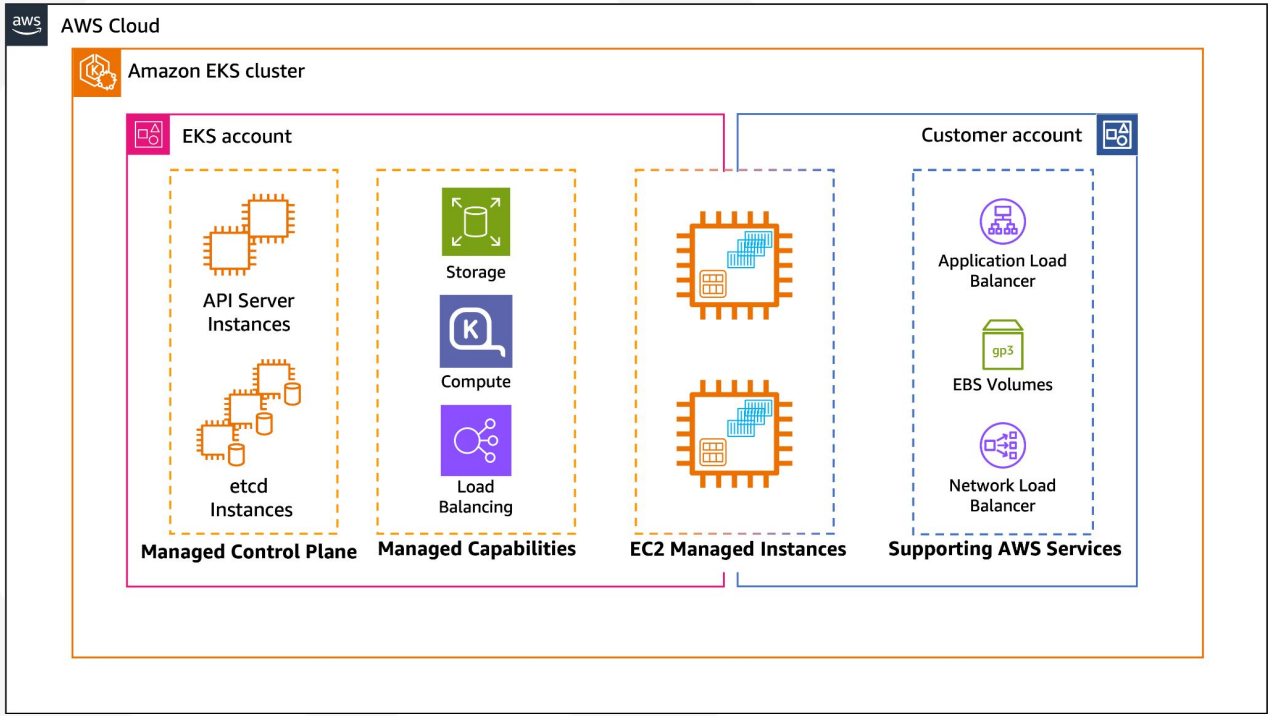
EKS - l'implémentation AWS de Kubernetes

- **Fargate** : pods "serverless" (pas de nœuds visibles) avec limites notables : **pas de DaemonSet**, pas de conteneurs **privilégiés**, pas de HostNetwork/HostPort, pas d'EBS. Idéal pour des workloads stateless simples.



EKS - l'implémentation AWS de Kubernetes

- **EKS Auto Mode** (nouveau) : EKS **provisionne et gère** automatiquement compute/réseau/stockage autour de vos pods ; peut cohabiter avec des MNG traditionnels.



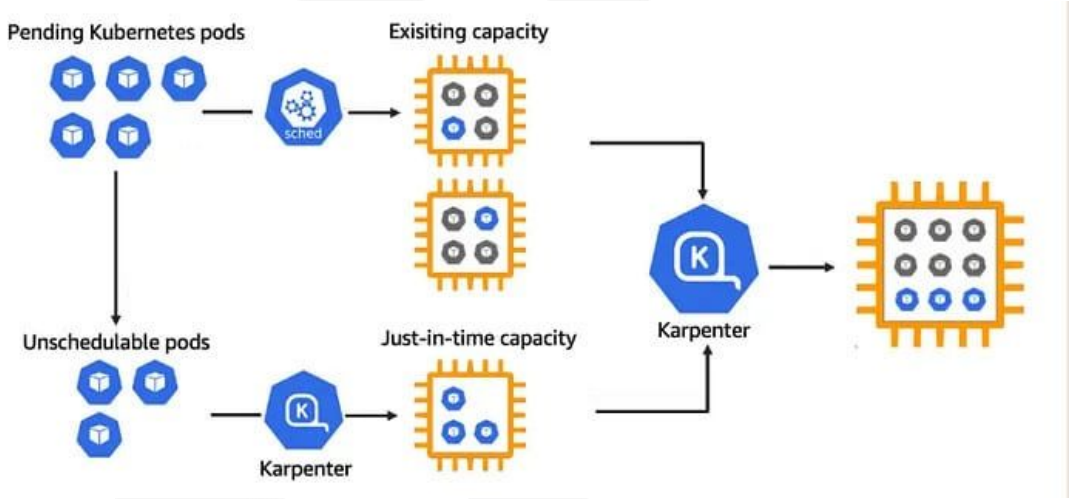
EKS - l'implémentation AWS de Kubernetes

Features	EKS Auto-mode	EKS Fargate
Core Add-ons	Managed by AWS	Managed by User
Worker Nodes	Managed Instances	Serverless (One Pod per VM)
DaemonSet Support	Supported	Not Supported
Customization	High	Limited
Auto Resource Optimization	Yes	No
Pricing Model	EC2-based billing (per instance-hour) : scalable with Spot/Savings Plans	Per vCPU-second and GB-second: no idle cost, great for ephemeral workloads

EKS - l'implémentation AWS de Kubernetes

Autoscaling des nœuds

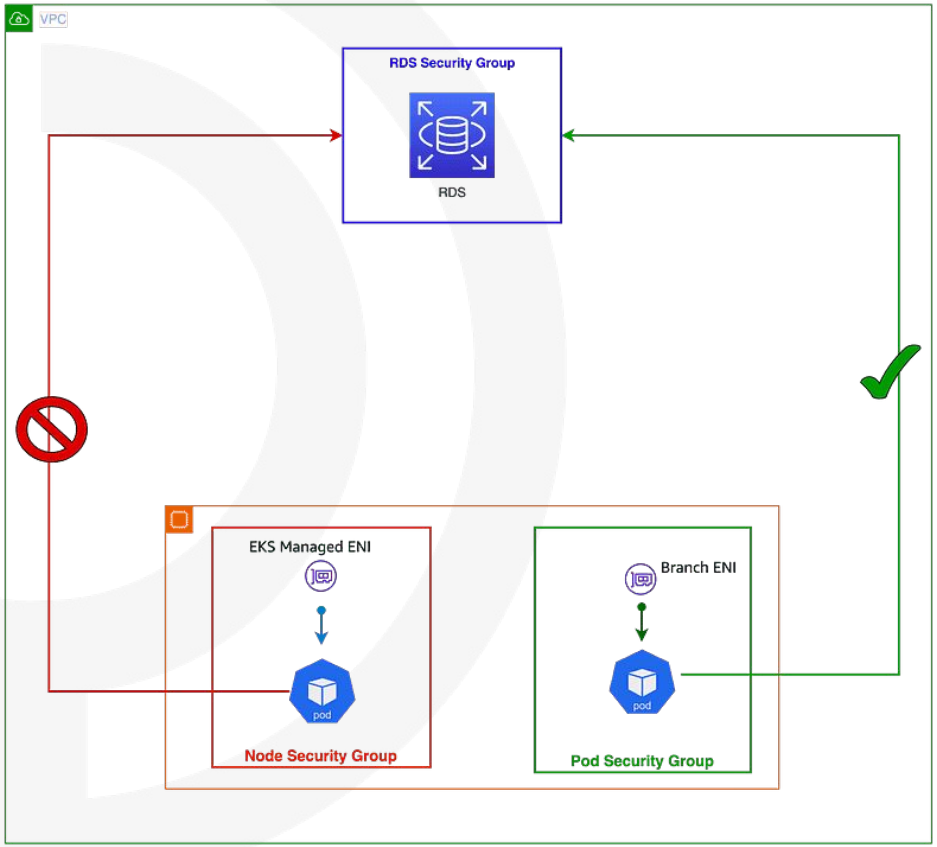
- Historique : **Cluster Autoscaler** (CA) sur des groupes (ASG/MNG).
- Moderne : **Karpenter** → provisionne des nœuds "à la demande" par **NodePool** + **EC2NodeClass**, gère consolidation/coût/Spot et scale très vite. (Le modèle v1 s'appuie sur NodePool/NodeClass.)



EKS - l'implémentation AWS de Kubernetes

Réseau de pods (CNI) & sécurité

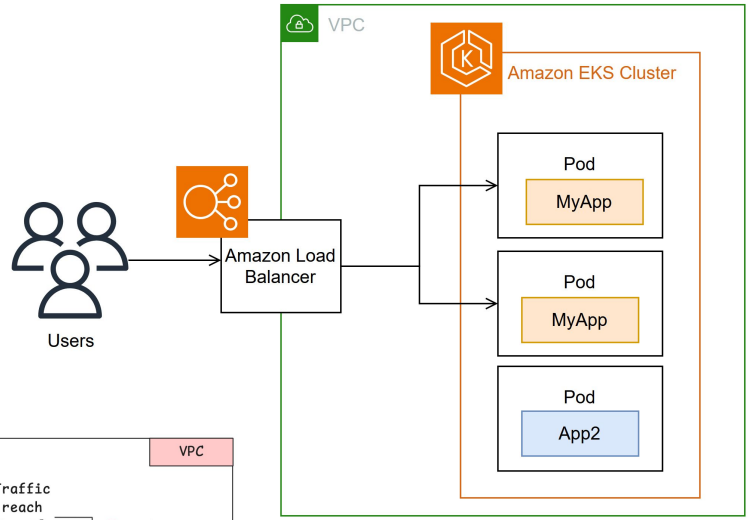
- **Amazon VPC CNI** (daemonset aws-node) : donne aux pods des IPs **VPC natives** (ENI/Prefix delegation) ; c'est l'add-on réseau par défaut d'EKS.
- **Security Groups for Pods (SGP)** : possibilité d'assigner **un SG par pod** (ENI trunk) au lieu d'un SG partagé par nœud → micro-segmentation L3/L4.

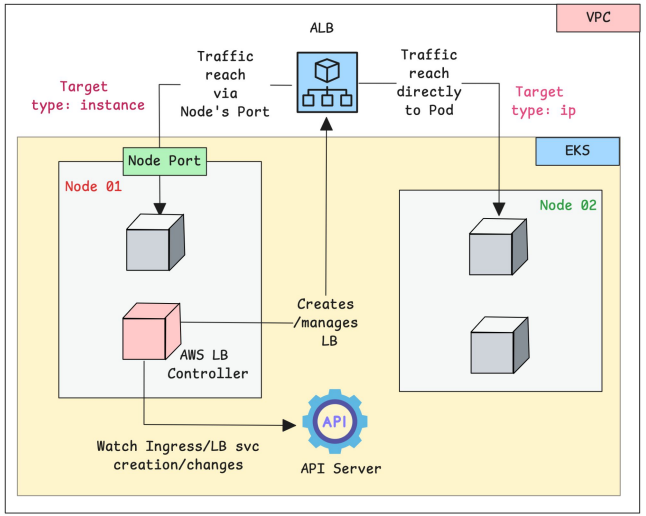


EKS - l'implémentation AWS de Kubernetes

Exposition & Load Balancing

- **AWS Load Balancer Controller (LBC)** :
  - **Ingress → ALB** (L7 HTTP/HTTPS, IngressClass alb).
  - **Service type=LoadBalancer → NLB** (L4).





EKS - l’implémentation AWS de Kubernetes

Stockage

- **EBS CSI Driver** pour volumes bloc (PV/PVC) – recommandé comme **EKS add-on** ; **non dispo sur Fargate** pour le montage.
- **EFS/FSx CSI** pour partage de fichiers et besoins spécialisés.

EKS - l'implémentation AWS de Kubernetes

IAM ↔ Kubernetes (authN/Z) & identités de pods

- **Accès humain/automate au cluster** :
  - Historiquement : ConfigMap **aws-auth** qui fait le pont **IAM → groupes RBAC**. Le créateur du cluster reçoit system:masters.
  - Recommandé aujourd'hui : **EKS Access Entries** (API EKS) + **EKS Access Policies** pour gérer les droits sans éditer aws-auth.
- **Accès des pods aux services AWS** :
  - Hier : **IRSA** (OIDC + trust policy IAM).
  - Aujourd'hui : **EKS Pod Identity** (agent EKS côté nœud, config via API EKS) → plus simple que l'IRSA ; **non supporté sur Fargate/Windows**.

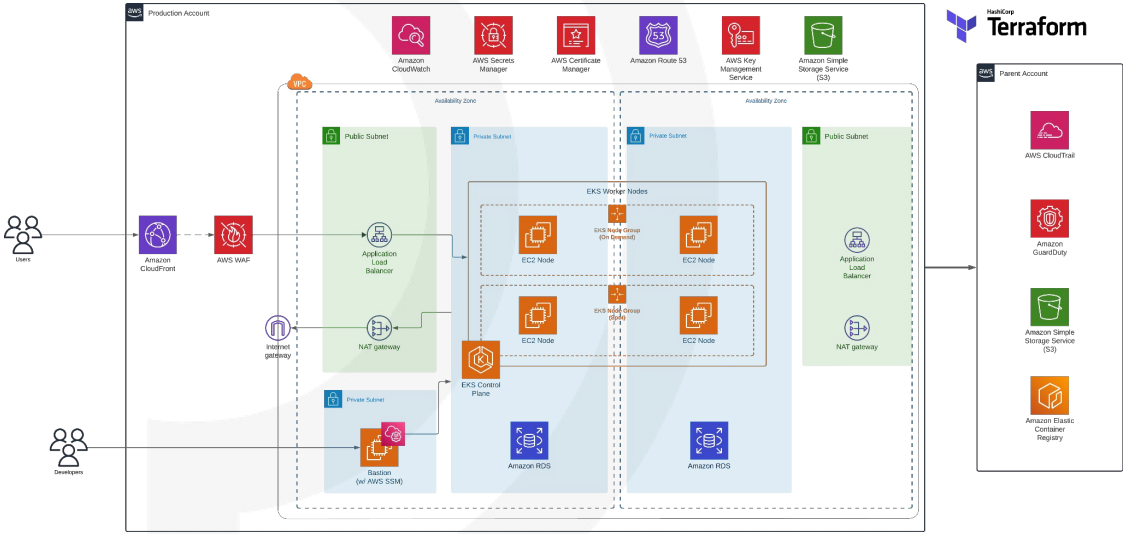
EKS - l’implémentation AWS de Kubernetes

Sécurité réseau du cluster

- **Endpoint API** : public (+ CIDR restreints) et/ou **privé** (VPC only). Pour des environnements sensibles, privilégier **private endpoint** et restreindre le public.

- **Security Groups** : EKS crée un **cluster security group** et recommande de l’associer aux node groups ; vérifiez les règles egress/outbound nécessaires.

- Mettre en place un **bastion SSM** pour accéder au control plane afin d’y accéder avec une **identité AWS**



Merci !

FEEDBACK

Aide-nous à nous améliorer !

https://forms.gle/FJkwGnUX2ZWcZJTq7