Conteneurs et Orchestrateurs

C. Sarralié (DPT Loiret)
Y. Stroppa (CNRS)

SPGoO

Sommaire

- Introduction CNCF
- Inventaire des solutions de conteneurisation
  - LXC, **Docker, Podman,** Cri-o, Singularity et autres
  - Concepts et comparaisons
- Orchestrateurs
  - Principes et concepts
  - **Swarm, K8S**, OpenShift
- Applications stateless/stateful : problématique
- Persistances NFS -- Ceph -- GlusterFS
- Point de vue du développeur : déploiement/continuité
  - CI/CD
- Point de vue de l'administrateur/supervision/monitoring
  - Optimisation
  - Surveillance et sécurité
- Conclusion

Bibliographie

- The Kubernetes Bible
  - Nassim Kebbani, Piotr Tylenda, Russ McKendrick
    - packt
- Kubernetes Patterns
  -
- Mastering Kubernetes
  - Ggi Sayfan Packt
- Orchestration de conteneurs
  - Docker - Swarm - K8s
    - François-Emmanuel GOFFINET
- Mastering Linux Security and Hardening
  - Donald A. Tevault
    - Packt
-

Bibliographie

- E-handbook
  - Guide de l'administration stockage et réseau sous Kubernetes
    - LeMagIT Pro+
- Docker
  - Pratique des architectures à base de conteneurs
  - Pierre-Yves Cloux Thomas Garlot Johann Kohler
    - Dunod

Sigles et acronymes

- CNCF : Cloud Native Computing Foundation
- CNI : Networking for Linux containers
- OCI : Open Container Initiative
- CRI : Container Runtime Interface

- DevOps :
- CI/CD : Continuous Integration / Continuous Delivery

- SDLC : Software Development Life Cycle

Introduction

- Objectif de cette présentation
  - Vulgariser les concepts des conteneurs
  - Expliquer les principes des orchestrateurs selon des points de vues complémentaires
    - Les développeurs
    - Les administrateurs
  - Mettre en avant le rapprochement nécessaire
    - --> DEVOPS et DEVSECOPS

Quelques cas et besoins

- RunMyCode/ExecAndShare (Orléans)
- ChickenPods (Lille)
- Plateformes de traitements ML/IA
  - Linguistiques, autres (SPGoO)

RunMyCode -- 2012 ExecAndShare

- Mise en place d'une solution de généralisation d'un accès SaaS
- Besoin de mettre à disposition via des sites web (Site compagnon) des besoins de calculs
- Développement sous J2EE
- Accès au CCIN2P3 grille de calculs
  - AFS - SGE - IRODS
  - DTM (Distributed Task Manager)

ChickenPods -- 2016-2017

- Développement d'application avec réseaux de neurones
- Serious Games
- Contexte de développement : nodeJS, web socket, multi-users
- Encapsulation dans une image docker
- Déploiement sur un orchestrateur (K8S)
  - Scalabilité horizontale
  - Monitoring -- prometheus
- Spring Boop : web service
- Nginx : reverse proxy

Plateformes IAML et autres

- Au sein du LLL
  - développement de plateformes de traitements de données
    - Classiques aux plus évoluées
      - AlimCorp -- DeepCorp
      - DeepBDD -- Anglais contemporain
    - FRAPeOR
      - orienté dans l'apprentissage des langues
      - avatar conversationnel

Plusieurs freins :
LLL n'est pas PP pas les mêmes moyens
Orléans n'est pas Lille :
Mais l'envie et les besoins restent les mêmes

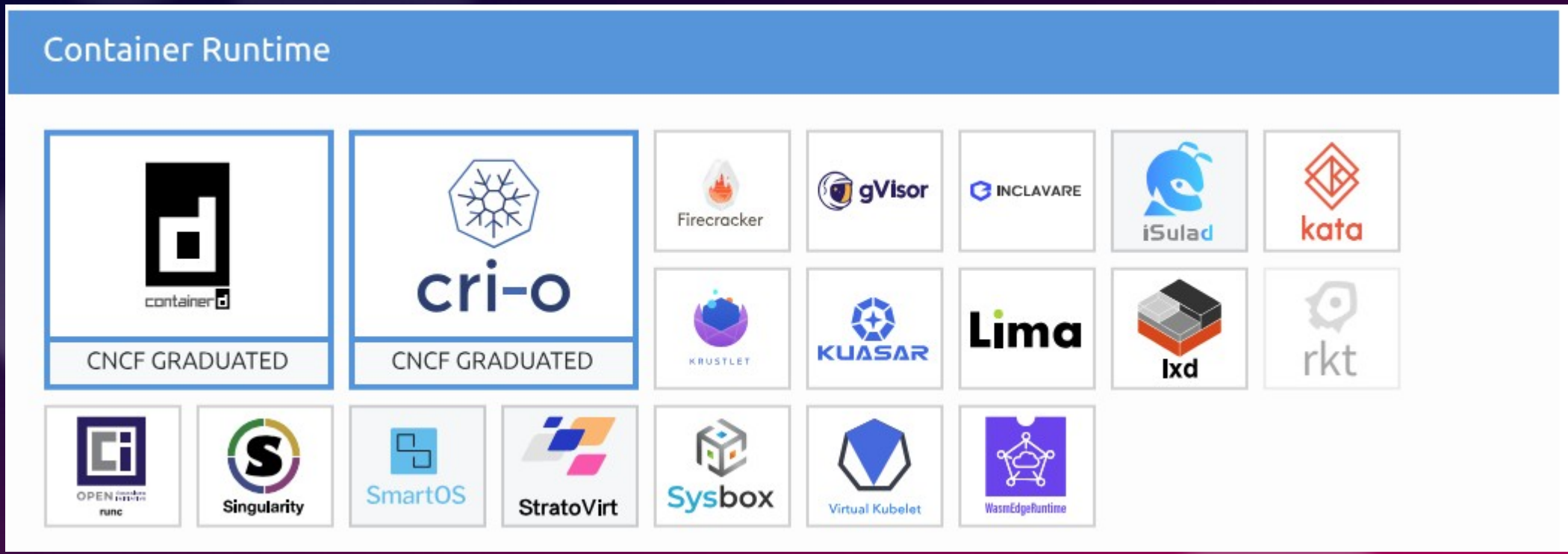
Introduction CNCF : Cloud Native Computing Foundation

- https://landscape.cncf.io/
- (pour voir l'état des projets en cours sur ce type de technologie)
- Application definition & image build
- Scheduling & Orchestration
- Service Proxy
- APi Gateway
- Coordination & service discovery

Containers

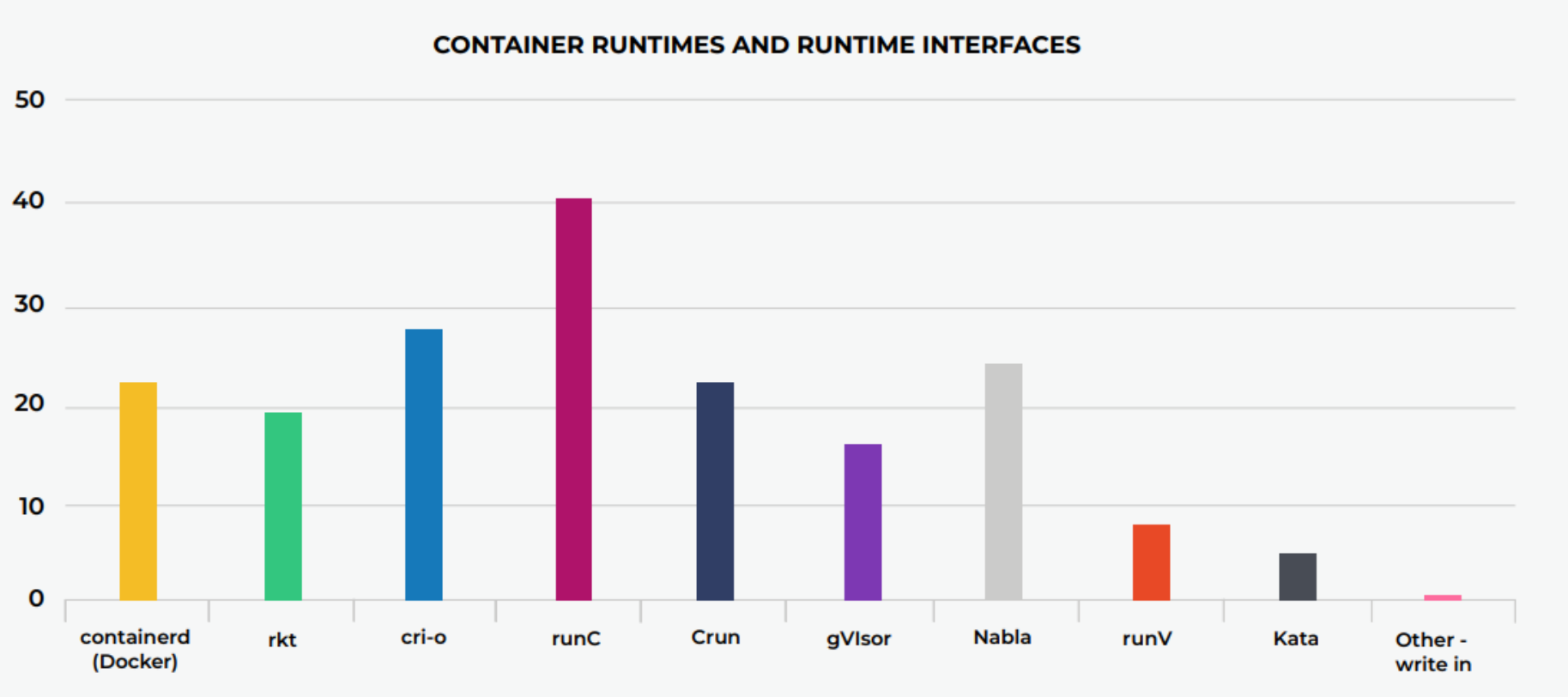
- Rappels
- Fonctionnement et principe
- Approche docker
- Approche Podman
- Approche Cri-o
- Approche Singularity
- Comparaisons et sécurité

Solutions possibles

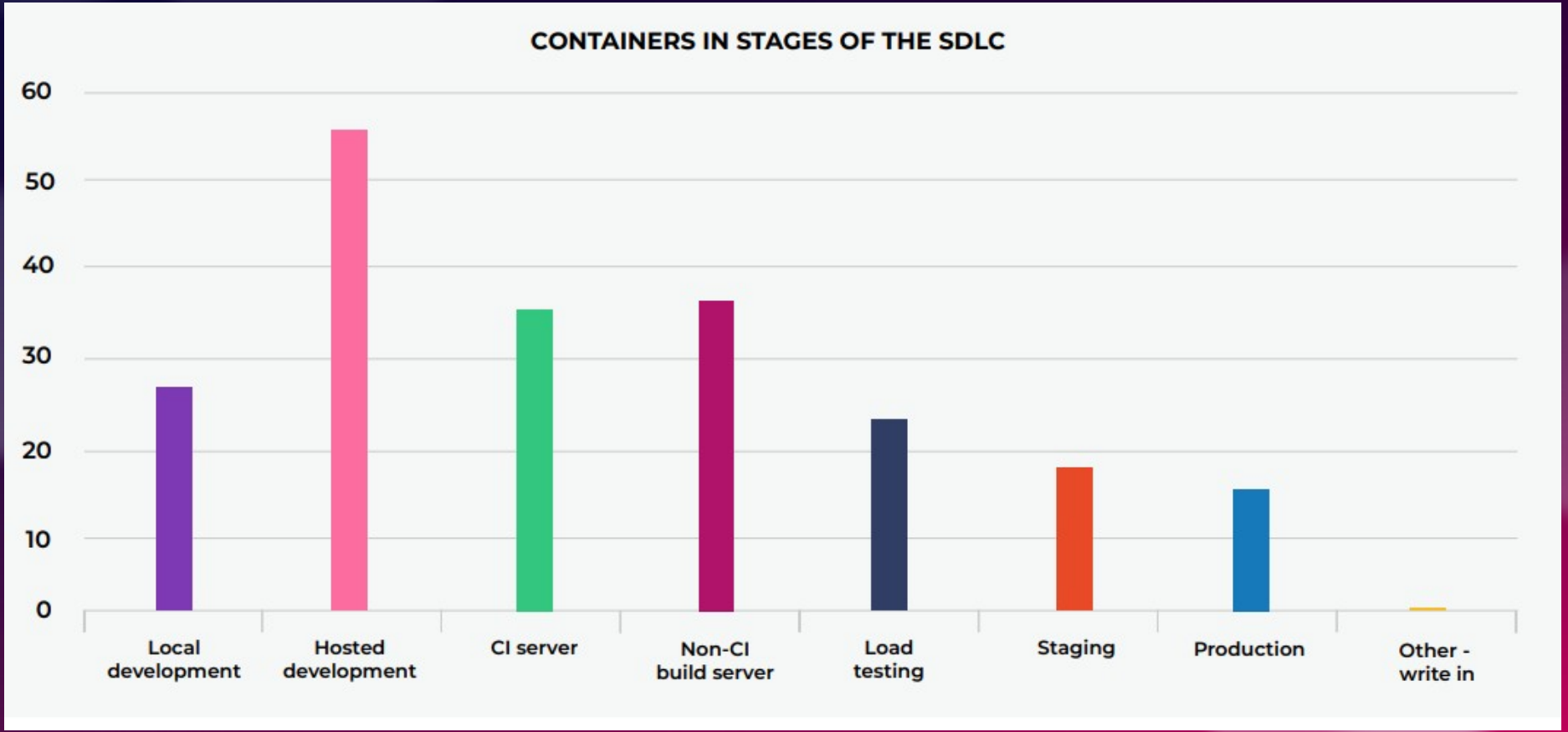


Rappel : les conteneurs runtime

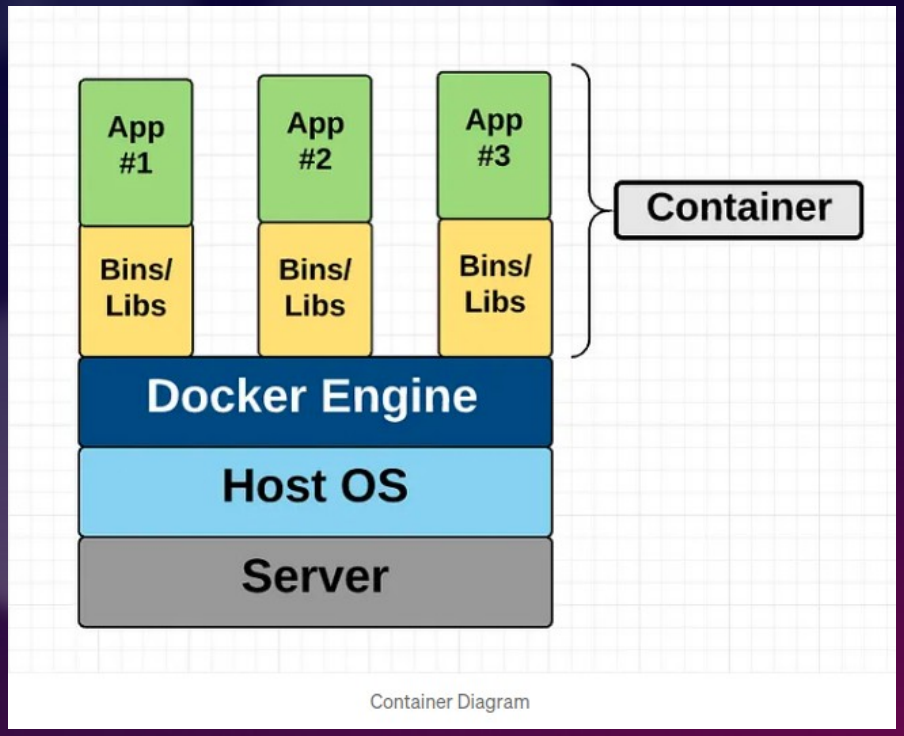
LXC, Docker , Podman, rkt, singularity, crio



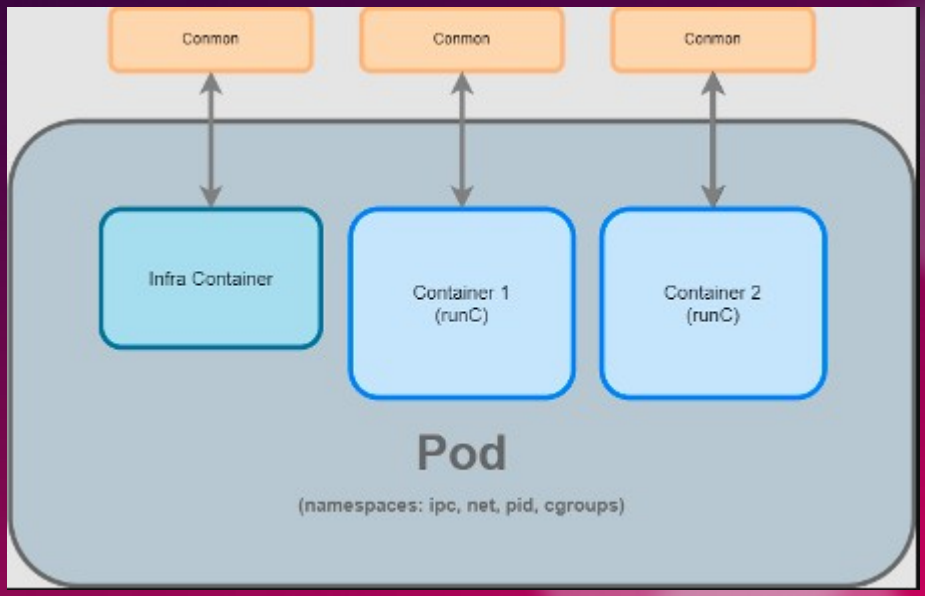
Container dans le cycle de vie



Rappel Conteneur



Container Diagram



Containers

- Objectifs et concepts
  - Analogie avec OOP
    - Image ~ classes ~ moule de fabrication
    - Conteneur ~ objet ~ Instanciation
  - Notion Extend ~ héritage
    - Conception d'une image à partir d'une autre
      - FROM
    - override (Surcharge)
      - Concept UFS

Container

- Différence en tre VM et Conteneur
- Conteneur -->
  - application
  - lib/dependencies
  - paramétrage

Approche LXC

- Le système est beaucoup plus présent que dans les solutions docker ;
-

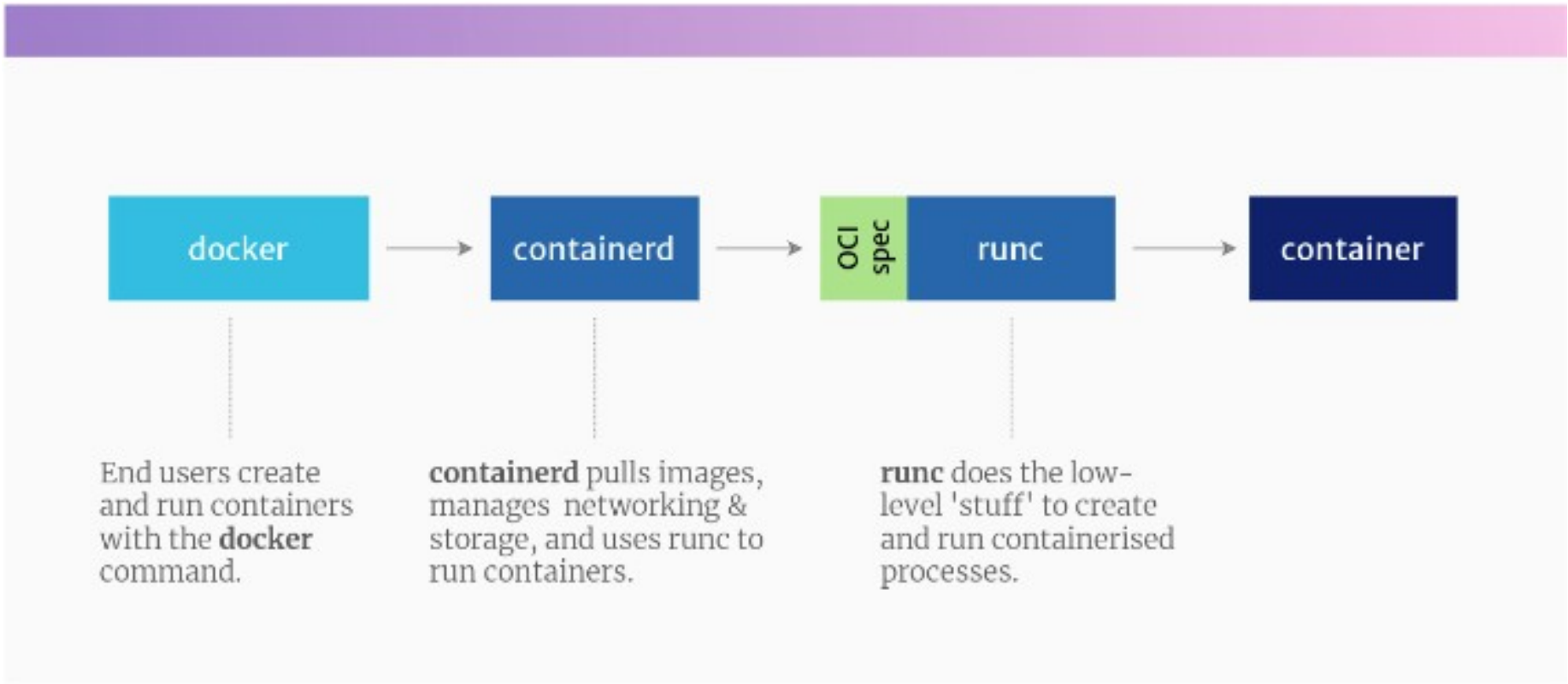
Approche Docker

- Démarrer en 2013 PyCon --> Santa Clara
- 2015-2016
  - rejoint le CNCF
  - standardisation OCI (Open Containers Initiative)
  - docker devient une plateforme
    - Docker Compose
    - Docker Machine
    - Docker Swarm
    - Kitematic
    - Docker cloud
    - Docker DATACENTER
    - etc
  - Docker INC --> offre commerciale

Docker : installation

- Solution en ligne
  - Play-With-Docker --> https://labs.play-with-docker.com/
  - Killercoda --> https://killercoda.com/
- Installation locale : desktop
  - Windows
  - Mac OS
  - Linux

Comment se lance le conteneur



The projects involved in running a container with Docker
Source: Tutorial Works

Quelques illustrations simples

- docker ps
- docker inspect
- docker logs
- docker run -d -p 82:80 httpd
- docker run -d -p 27087:27017 mongod
- Avec persistance
  - docker run -d -p 27087:27017 -v /home/ystroppa/mongodb:/db/data mongod
  - docker run -d -p 27087:27017 -v /home/ystroppa/mongodb:/db/data mongod --auth

Création d'image

- docker build
  - Création du dockerFile
- Création d'image multi stages

- Création d'image multi config
  - Fichier de configuration

Construction d'une image en multi-stages

```
FROM maven:3.6.3-jdk-11-openj9 as builder
COPY src /usr/src/app/src
COPY pom.xml /usr/src/app
RUN mvn -f /usr/src/app/pom.xml clean package

FROM openjdk:11.0.11-jre-slim
COPY --from=builder /usr/src/app/target/webservice-0.0.1-SNAPSHOT.jar /usr/app/webservice-0.0.1-SNAPSHOT.jar
RUN ls -la /usr/app
EXPOSE 9995
ENTRYPOINT ["java","-jar","/usr/app/webservice-0.0.1-SNAPSHOT.jar"]
```

Container Multi-Architecture

Multi-arch Docker image --> liste d'images avec binaires et librairies compilées pour de multi-architectures (ARM, x86, RISC-V,...)

- Performance and Cost Optimization
- Cross Platform Developement
- IoT Devices

Bénéfices d'utiliser des images multi-architectures

- capacité à exécuter une image docker sur plusieurs archi
- capable de choisir son eco-friendly CPU architecture
- moindre effort dans la migration d'une architecture vers une autre
- meilleure performance et cout en utilisant ARM64
- capacité à utiliser plusieurs cores par CPU en utilisant ARM64

Comment construire des images de conteneurs en multi-architectures

- Méthode traditionnelle
- en utilisant docker buildx
- en utilisant buildah

Container Multi-Architecture / manuelle

Dockerfile
FROM nginx
RUN echo"Hello But3" > /usr/share/nginx/html/index.html

sur une machine amd64
docker build -t ystroppa2/custom-nginx:v1-amd64
docker push ystroppa2/custom-nginx:v1-amd64

sur une machine arm64
docker build -t ystroppa2/custom-nginx:v1-arm64
docker push ystroppa2/custom-nginx:v1-arm64

Création du fichier manifest
docker manifest create ystroppa2/custom-nginx:v1 \
   ystroppa2/custom-nginx:v1-amd64 \
   ystroppa2/custom-nginx:v1-arm64
docker manifest push ystroppa2/custom-nginx:v1

Container Multi-Architecture / buildx

Dockerfile
FROM nginx
RUN echo"Hello But3" > /usr/share/nginx/html/index.html

docker buildx build --push --platform linux/amd64,
linux/arm64 -t ystroppa2/custom2-nginx:v1

Attention à la version de docker avec l'option platform

Avec Buildah /pour Podman

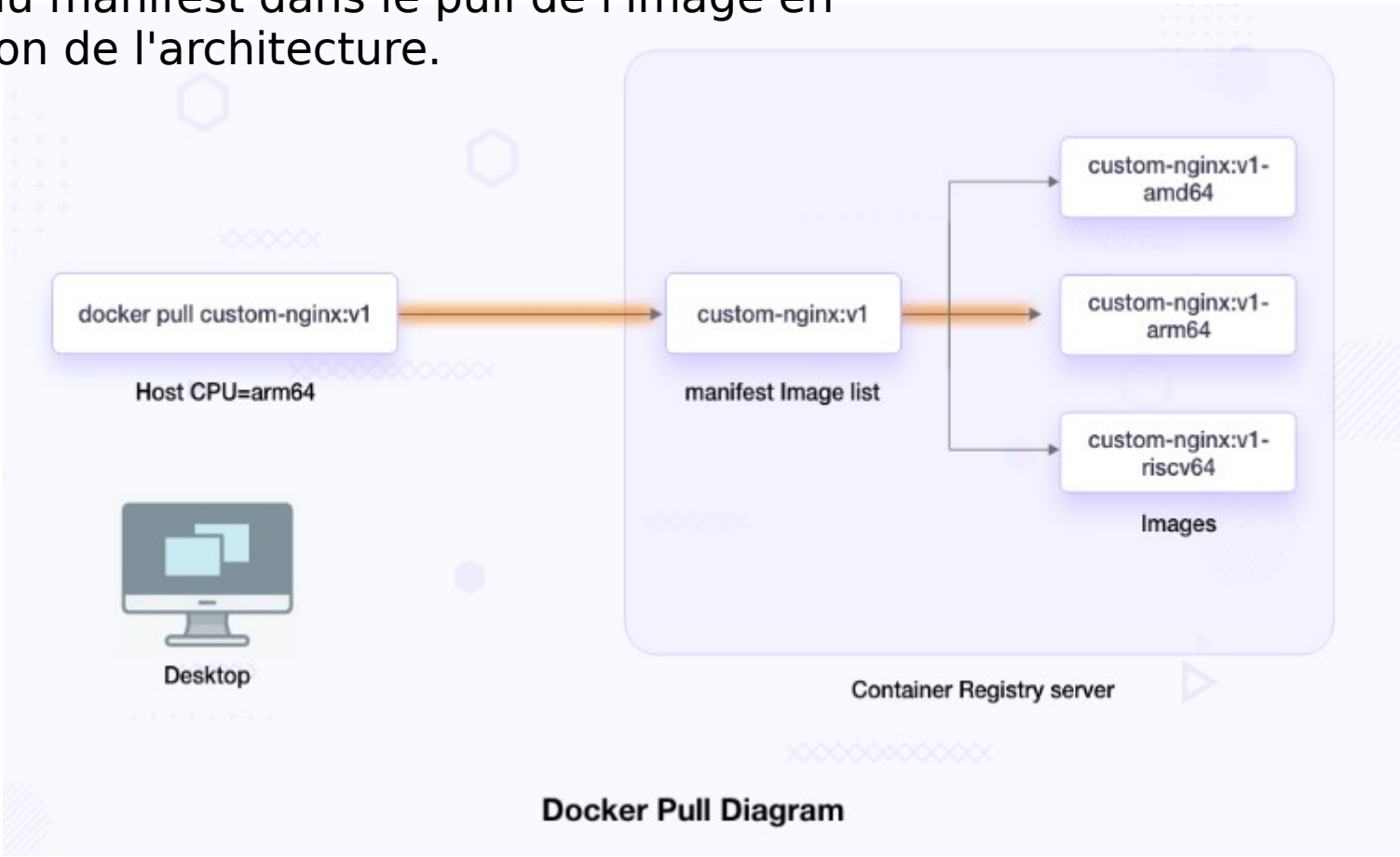
Dockerfile
FROM nginx
RUN echo"Hello But3" > /usr/share/nginx/html/index.html

Au préalable installer
sudo apt install qemu-user-static

buildah manifest create multiarch-test
buildah bud --tag ystroppa2/cut-nginx:v1-amd --manifest multiarch-test --arch amd64 .
buildah bud --tag ystroppa2/cut-nginx:v1-arm --manifest multiarch-test --arch arm64 .
buildah manifest
buildah manifest inspect multiarch-test

Container Multi-Architecture

Rôle du manifest dans le pull de l'image en fonction de l'architecture.



Docker Pull Diagram

Sécurité

- Installation de scout pour vérifier la vulnérabilité des images
- (https://docs.docker.com/scout/install/ attention il faut être connecté sur se compte de docker)
- Identification des CVEs (commun Vulnerability and exposure)
- https://cevdetails.com; https://cve.org
- Exemples d'utilisations :
  - docker scout cves alpine
  - docker scout cves httpd
  - docker scout recommandations httpd
- (informations renvoyées par ces commandes :
  - **C : Critical, H : High,** M : Medium, L : Low)

Approche podman

- Podman s'exécute en tant qu'utilisateur non privilégié et sans daemon central.
- Podman est sorti en 2018, concentrés au début sur la compatibilité avec la ligne de commandes (CLI) de Docker . N'a pas inclut tout de suite la prise en charge de l'API REST .
- Un projet communautaire Podman Compose a vu le jour. Podman compose traite la spécification compose et la traduit en commandes CLI Podman.
- Podman s'oriente plus vers Kube pour son interfaçage

Podman : exemple de commande

- podman run -dt --rm docker.io/httpd
  - activation un processus conmon

```
ystroppa@vmteo:/etc/containers$ podman ps
CONTAINER ID  IMAGE                           COMMAND         CREATED        STATUS
PORTS                   NAMES
f86aba57ad8a  docker.io/library/httpd:latest  httpd-foreground  13 seconds ago  Up 13 seconds ago
0.0.0.0:37785->80/tcp  bold_visvesvaraya
```

```
ystroppa@vmteo:/etc/containers$ podman port -l
80/tcp -> 0.0.0.0:37785
```

```
ystroppa@vmteo:/etc/containers$ curl http://localhost:37785
<html><body><h1>It works!</h1></body></html>
```

Comparaisons

- Docker
  - Service
  - root
  - network
-
- Podman
  - pas de service
  - pas root
  - pas de network

Fonctionnement

Architecture
Monolithique - Micro services

- Explication sur la comparaison des deux approches
- Notion de micro-service important avant d'expliquer les orchestrateurs
  - architecture très populaire
  - modularité
  - bien adaptée à K8S

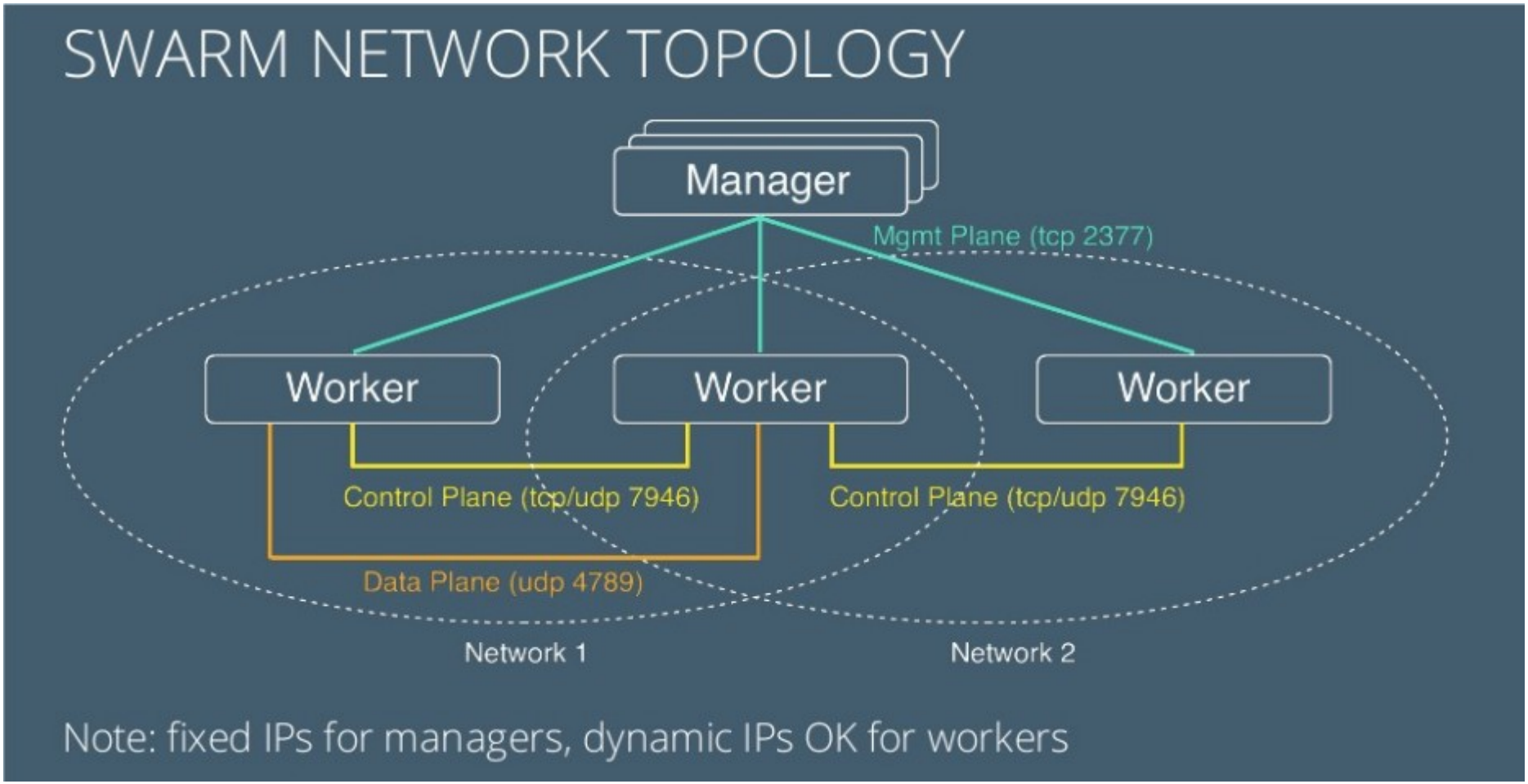
Orchestrateurs

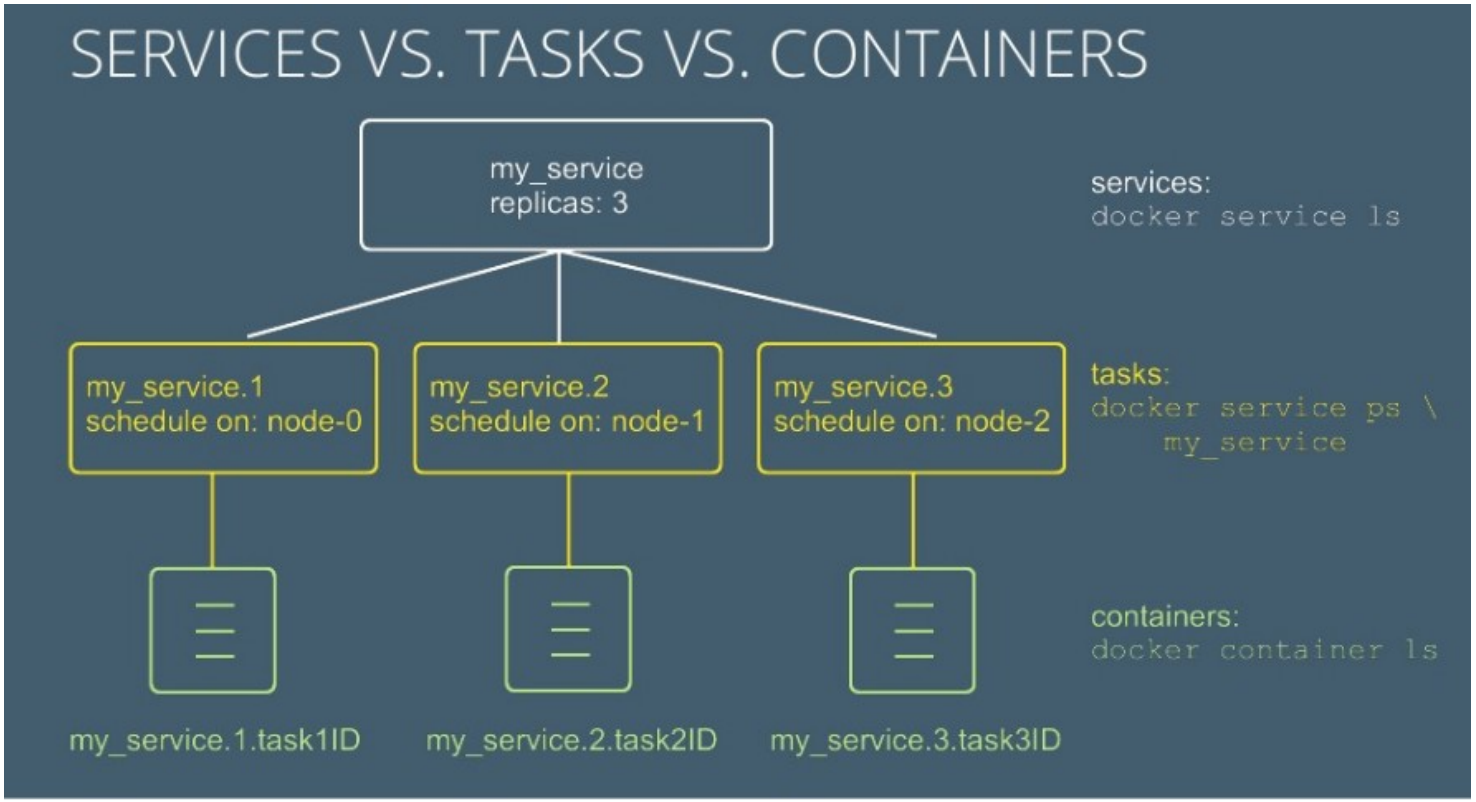
- SWARM
- Kubernetes
  - 2014 Google --> Kubernetes
  - 2015 --> CNCF

SWARM

Cluster natif et outils d'orchestration pour les conteneurs Docker. Partie intégrante de l'éco-système de Docker et apporte au moteur Docker la coordination de multiples instances de docker sur plusieurs machines. Comprend des nodes Manager et Worker. Permet de faire fonctionner des conteneurs de haute disponibilité sans la complexité de Kubernetes.

Vous pouvez utiliser Docker Cli, Docker-compose et docker API pour gérer les conteneurs et déployer des applications sur ce cluster.


SWARM NETWORK TOPOLOGY
Manager
Mgmt Plane (tcp 2377)
Worker
Worker
Worker
Control Plane (tcp/udp 7946)
Control Plane (tcp/udp 7946)
Data Plane (udp 4789)
Network 1
Network 2
Note: fixed IPs for managers, dynamic IPs OK for workers


SERVICES VS. TASKS VS. CONTAINERS
my_service
replicas: 3
services:
docker service ls
my_service.1
schedule on: node-0
my_service.2
schedule on: node-1
my_service.3
schedule on: node-2
tasks:
docker service ps \
my_service
containers:
docker container ls
my_service.1.task1ID
my_service.2.task2ID
my_service.3.task3ID

Orchestrateur : Cluster Swarm

notion de services

docker service create nginx

```
$ docker service create --name my_web \
        --replicas 3 \
        --publish published=8080,target=80 \
        nginx

$ curl localhost:8080

$ docker service create \
  --mode global \
  --publish mode=host,target=80,published=8080 \
  --name=nginx \
  nginx:latest
```

Orchestrateur : Cluster Swarm

Notion de stack : un ensemble de conteneurs que l'on va déployer en même temps sur un même worker.

docker service create --name srv_nginx nginx
-->Attention d'accès via IP
-->curl http://172.....

docker service create --publish 82:80 --replicas 2 --name serv_nginx_repl nginx
-->Accès sur les managers via http://localhost:82

On peut augmenter le nb de replicas
docker service scale serv_nginx_repl=4

Installation d'un petit utilitaire de visualisation
docker service create --name=viz --publish=8080:8080/tcp --constraint 'node.role==manager' --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock dockersamples/visualizer

Installation nouveau service :
docker service create --publish 8081:80 --replicas 3 --name srv_hello tutum/hello-world
--> modification du nombre de réplicas
docker service update srv_hello --replicas=5

Faire un curl sur le port 8081, et on doit constater le load-balancing de l'accès au service proposé par Swarm.

Exemple d'exécution d'une stack

```
version: '3.1'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

docker stack deploy -c docker-compose.yaml stack1

Accès via le port 8000 à la configuration de wordpress.

Orchestrateur : Cluster Swarm

Commandes très simples :

docker swarm init

docker node list ou ls

Avantages :

facile, service discovery, qui assigne automatiquement un nom DNS pour chaque service de SWARM, simplifie les communications inter-services, étend les fonctionnalités de load-balacing. Offre également la distributions des tasks à travers les workers nodes.

Inconvénients :

n'a pas de système de provisionnement de stockage partagé n'a pas le même niveau d'offre que Kubernetes. On peut utiliser GlusterFS

Swarm - secret

-->Créer un fichier mypasswd dans lequel on pousse le password

docker secret create password ./mypasswd

-->Détruire le fichier

docker secret inspect password

--> Préparer le fichier yaml pour de démarrage du service

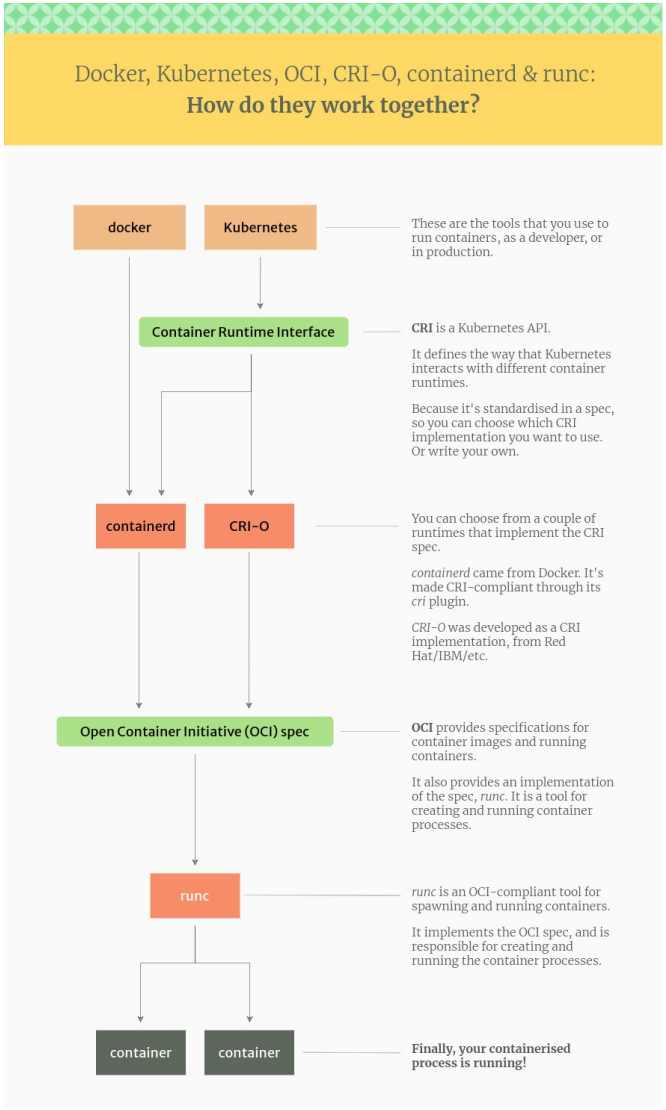
```
version: "3.3"

services:
  db:
    image: mysql:5.7
    volumes:
      - /var/lib/mysql
    secrets:
      - password
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD_FILE: /run/secrets/password

secrets:
  password:
    external: true
```

docker stack deploy -c docker-compose.yaml stack1

Docker, Kubernetes, OCI, CRI-O, containerd & runc



Notion de Microservice

- Architecture populaire
  - Modularité bien adapté aux orchestrateurs

Orchestrateur : Kubernetes

Invention de google, maintenu par le Cloud Native Computing foundation CNCF. Contrairement à SWARM, Kubernetes est mieux équipé pour gérer des charges de travail complexes et une architecture avec des APIs.

Induit plusieurs Nodes, partitionnés entre Worker et Manager Nodes. il introduit plusieurs focntionnalités telles que automates scaling, load balancing, service discovery. Comme SWARM il n'a pas de stockage partagé intégré.

Avantage de Kubernetes

automatiquement scaling et load balancing, il est plus fort que Swarm dans cette gestion. Il a une architecture robuste et sécurisée plus sophistiquée qui nécessitent des protocoles de sécurité rigoureux. On peut implémenter des RBAC, des network policies ...

Inconvénients :

Plus complexe dans l'installation et nécessite une bonne connaissance de ces structures et des lignes de commandes.

Kubernetes services dispo sur le cloud

Cluster en ligne :

- Google Kubernetes Engine (GKE)
  - client gcloud
- Amazon Elastic kubernetes Service (EKS)
  - voir également ECS
- Azure Kubernetes Service (AKS)
- IBM Cloud Kubernetes Service
- DigitalOcean Kubernetes

Clusterless Container service

- AWS fargat
- Azure Container Instances
- Google Cloud Run

Kubernetes services dispo sur le cloud

Kubernetes installers

- kops
- kubespray
- kubeadm
- Rancher Kubernetes engine
- Puppet Kubernetes Module

Multicloud Kubernetes clusters

- VMWare tanzu
- OpenShift
- Anthos
- version de base

Orchestrateur : Kubernetes

Plusieurs solutions possibles pour installer ce type de solution :

- k0s CNCF
- k3s Rancher
- K8s CNCF
- MiniKube
- MicroK8s

Pour K8S plusieurs configurations possibles avec podman, cri-o ou docker.

Rappel infrastructure Kubernetes

1.2.1 Orchestration efficace des conteneurs

Automatise le déploiement, la mise à l'échelle et la gestion des applications conteneurisées.

Optimise l'utilisation des ressources pour améliorer les performances et réaliser des économies.

Permet l'auto-réparation en remplaçant automatiquement les conteneurs ou nœuds défaillants.

1.2.2 Évolutivité et équilibrage de charge

Prend en charge la mise à l'échelle automatisée pour gérer différents niveaux de demande.

Fournit un équilibrage de charge intégré pour une répartition uniforme du trafic.

1.2.3 Stratégies de déploiement flexibles

Propose diverses stratégies de déploiement pour des mises à jour et des restaurations transparentes.

Permet aux versions Canary de tester les modifications avec un sous-ensemble d'utilisateurs.

1.2.4 Découverte de services et équilibrage de charge

Facilite la découverte de services internes basés sur DNS pour une communication transparente.

Équilibre automatiquement le trafic pour maintenir des performances constantes.

Rappel infrastructure Kubernetes

1.2.5 Gestion déclarative des configurations

Utilise des fichiers YAML déclaratifs pour définir l'état d'application souhaité.

Réduit la dérive de configuration et garantit des déploiements cohérents.

1.2.6 Communauté et écosystème forts

Possède une communauté dynamique et active pour le soutien et la collaboration.

Offre un riche écosystème d'outils, d'extensions et d'intégrations.

1.2.7 Prise en charge multi-environnements

Fonctionne dans divers environnements, y compris sur site et dans le cloud.

Permet un déploiement et une gestion cohérents des applications.

1.2.8 Extensibilité et personnalisation

Hautement extensible avec des API et des plugins pour des fonctionnalités personnalisées.

Permet d'adapter Kubernetes aux besoins organisationnels spécifiques.

Les commandes pour la solution de base

les outils :

- kubeadm
- kubelet
- kubectl

Figure 1: Kubernetes architecture

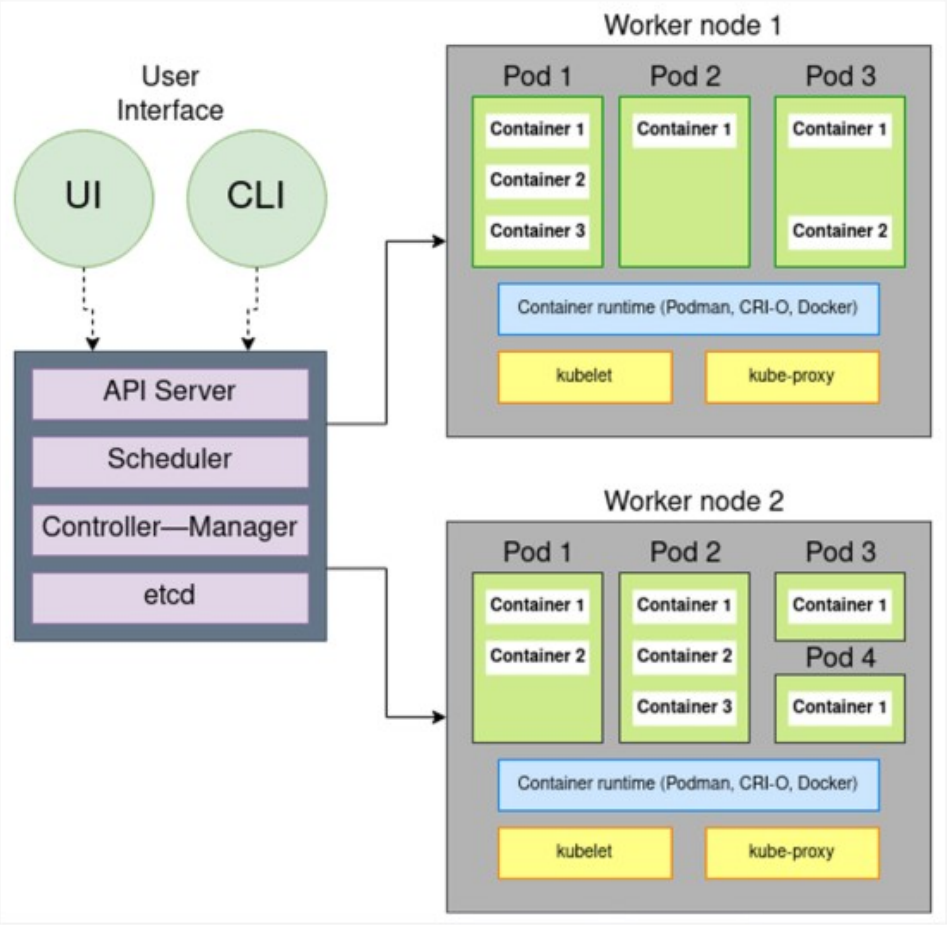


Image source: Nived Velayudhan, CC BY-SA 4.0

Kubernetes :

kube-apiserver : frontal du control-plan --> APi request

etcd : (key-value store) database où K8S stocke toutes ses informations quels noeuds, quelles ressources ...

kube-scheduler : tâche qui décide où sera exécuté le prochain pod.

kube-controller-manager : responsable du contrôle d'exécution des ressources --> telles que les déploiements

cloud-controller-manager : gère les ressources telles que load-balancers et les disk volumes.

codeDNS : DNS-based Kubernetes service discovery.

Kubernetes : Concepts

namespaces

pod

deploy

service

endpoints

secret

configmap

persistent volume

persistent claim volume

Kubernetes démarche

https://medium.com/@bijit211987/kubernetes-ci-cd-pipelines-best-practices-and-tools-ca2158939299

K8S est un des leaders dans ce contexte CI/CD, mais utiliser des commandes kubectl manuellement peut être cause d'erreurs. en utilisant un pipeline CI/CD avec Kubernetes on va automatiser le déploiement des applications dans ce type de contexte.

7 bonnes pratiques pour CI/CD et Kubernetes

- use GitOps
- Scan your container images (Snyk, docker scan command)
- Use Helm to manage Deployments
- Ensure There's Rollbak Mechanism
- Use Immutable Image Tags : évitez les my-app:latest ???
- Follow Kubernetes security Best Practices : secret, RBAC ....
- Try pull-based CI/CD workflows (automatisation)

Kubernetes : déploiement

Helm : voir https://helm.sh/

installation ./get_helm.sh

Système d'installation clés en main à partir de dépôts sous Kubernetes

Fonctionne avec une notion de charts : ensemble de paramétrage spécifique pour le déploiement sous kubernetes.

Ajout de dépôt dans votre environnement :

helm repo add stable https://charts.helm.sh/stable

Exemple : helm install mysqlys bitnami/mysql

à exécuter sur une machine qui accéde au cluster

Kubernetes : déploiement

Helm :

Quelques commandes pratiques :

helm list

helm show chart bitnami/mysql

helm uninstall [nom]

helm install --dry-run --debug nginx-test bitnami/nginx

Attention à l'installation de service nécessitant de la persistance comme mysql, wordpress . Helm n'approvisionne pas votre système c'est à vous de mettre en place les PV qui vont bien pour permettre ensuite aux CLAIMS de les accrocher.

Kubernetes : persistance PV et PVC Applications statefull

Comment conserver une persistance quand un pod est supprimé ou replacé sur un autre worker sachant que l'on ne maîtrise pas où sont exécutés les pods. donc il faut que l'on ait un système de volume qui s'associe au pod.

Plusieurs solutions : local, NFS, CEPH, GlusterFS, EKS,AWS S3 .....

Premier concept PV :

est une façon pour définir un stockage de données tels qu'une Storage Class ou storage implementations.

est une ressource object de Kubernetes (kubectl get pv)

créer un PV est équivalent à créer un objet ressource de stockage avec un plugin spécifique en fonction de la nature de cette ressource. Pour utiliser cette ressource, on doit le demander à l'aide d'un PVC. Un PVC est une requête de stockage, laquelle est utilisée pour monter le PV dans le Pod.

L'administrateur peut mapper différentes classes sur différents niveaux de services et différentes règles.

Kubernetes : persistance PV et PVC Applications statefull

Un PV est décrit à partir d'un fichier Yaml qui permet de décrire la configuration et le plugin utilisé

```
apiVersion : v1
kind : PersistentVolume
metadata :
   name : pv005
spec :
   capacity :
      storage : 5Gi
   volumeMode : Filesystem
   accessMode :
     - ReadWriteOnce
   persistantVolumeReclaimPolicy : Recycle
   storageClassName : slow
   mountOptions :
      - hard
      - nfsvers : 4.1
   nfs:
     path : /tmp
     server : 172.17.0.2
```

Kubernetes : persistance PV et PVC Applications statefull

Les PVC ?

c'est une requête/déclaration de l'usage d'un espace de stockage. Lequel sera monté dans le pod. Niveau d'abstraction, les devs ne sont pas obligés de savoir comment et sur quoi va reposer leur espace de stockage.

Exemple de fichier yaml de déclaration d'un PVC

```
apiVersion : v1
kind : PersistantVolumeClaim
metadata :
   name : pvc004
spec :
   storageClassName : manual
   accessModes :
     - ReadWriteOnce
   ressources :
     requests :
       storage : 3Gi
```

Kubernetes : persistance PV et PVC Applications statefull

Lifecycle de PV et PVC

Provisionning : création du PV

Binding : assignation PV à PVC

Using : Pods utilisent le PV à travers le PVC

Reclaimaing : il est gardé pour une nouvelle utilisation ou destruction

Un volume peut être :

- Available
- Bound
- Released
- failed

Provisioning :

- Static : l'administrateur créer les PVs et lors des demandes ils sont associés aux PVC en fonction des classes éventuellement.
- dynamic : si il n'y a pas de PV, le cluster peut allouer directement des PVs pour répondre aux besoins des PVC.

Notion de MESH

Istio, Linkerd, Hashicorp Consul, Kong KUMA, Google Anthos, VMWARE Tanzu

Fonctions clés d'un service MESH : networking, security and observability pour des micro services

Centralized Security :

la mise en place de la sécurité peut-être faite à partir du controlPlan en central. plutôt qu'à partir de chaque service. Un service MESH peut renforcer la sécurité à travers les frontières des différents clusters.

Service MESH fournit authentification,autorisation et le contrôle d'accès aux applications.

- Authentification : mTLS implémentations , JWT
- Autorisation : règles --> allow, deny ou actions personnalisées
- Access Control : règles RBAC

MESH

Advanced Networking and Resilience Testing

Offre un contrôle sur les flux entre service .Devops engineers peuvent splitter le trafic entre service ou les router.

Unified Observability

Metrics

Distributed tracing

Access logs

ISTIO produit le plus populaire.

Rappel sur les protocoles

TLS :

le serveur dispose d'un certificat et d'une paire de clés publique/privée alors que le client n'en dispose pas.

1. le client se connecte au serveur
2. le serveur présente son certificat TLS
3. le client vérifie le certificat du serveur
4. le client et le serveur échangent des nformtioans via une connexion TLS cryptée

Rappel sur les protocoles

mTLS

Dans mTLS , le client et le serveur disposent tous deux d'un certificat et les 2 parties s'authentifient à l'aide de leur paire de clés publiques/privées. Le protocole comporte des étapes supplémentaires.

1. le client se connecte au serveur
2. le serveur présente son certificat TLS
3. le client vérifie le certificat du serveur
4. Le client présente son certificat TLS
5. Le serveur vérifie le certificat du client
6. le serveur accorde l'accès
7. Le client et le serveur échangent des informations via une connexion TLS cryptée.

mTLS

Pourquoi utiliser mTLS ?

mTLS permet de garantir que le trafic est sécurisé et fiable dans les deux sens entre client et serveur.

mTLS prévient divers types d'attaques notamment :

- Attaques sur le chemin
- Attaques par usurpation d'identité
- Remplissage d'info d'identification
- Attaque par force brute
- Attaques par fishing
- Demandes d'API malveillantes