Orchestration

Avantages des containers
- entité légère (configuration et déploiement)
- permet des déploiements rapides et adaptatifs

Un système d'informations peut être constitué de centaines de containers qui s'exécutent simultanément
- impossibilité d'administrer cela manuellement
  - notamment les mises à jour
  - et la tolérance aux fautes ?

L'**orchestration** est essentielle
- différentes solutions logicielles (docker compose)
- un standard : Kubernetes

Kubernetes

Permet d'automatiser le déploiement et l'administration des applications structurées en containers
- open source par Google
- standard de fait, comme docker pour les containers
- par défaut, des containers sans état

2 éléments principaux de configuration
- la composition de l'architecture (matérielle ?) sur laquelle s'exécutent les applications
- la description des applications et comment y accéder

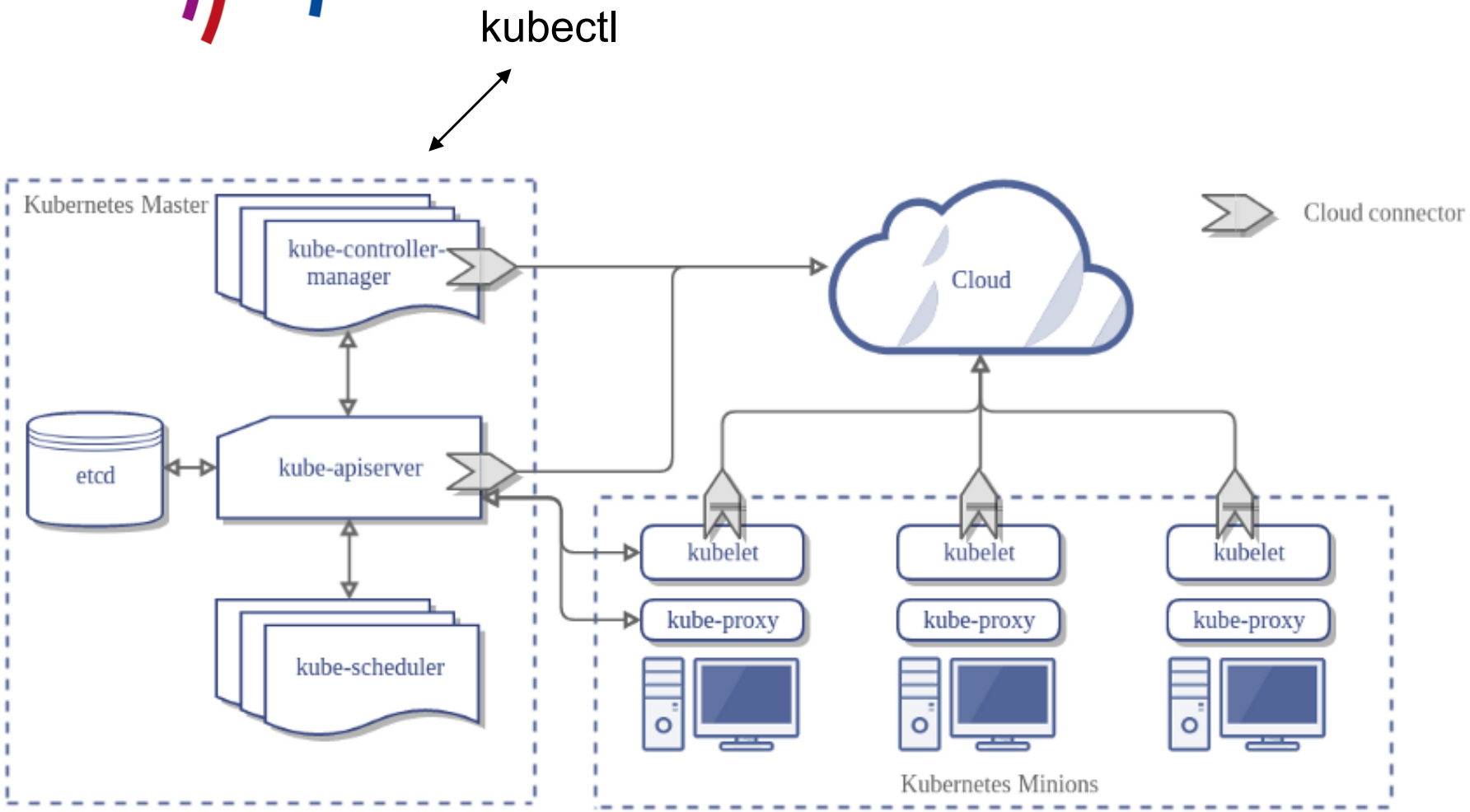
Composition de l'architecture matérielle
- control plane : exécute les services permettant à K8S de fonctionner
- noeuds (minions) : exécute les applications

Description des applications
- en terme de haut niveau (containers et liaisons)
- de niveau de service voulu (réplication)
- et d'accès (machine, port, routage)

Le but de K8S est de toujours garder les applications dans **l'état voulu décrit dans la configuration**
- nombre d'instances de containers
- ports de communication

et de router les requêtes vers les containers qui peuvent y répondre

Architecture globale



kubernetes

L'architecture de Kubernetes

Les composants maitres
- contrôlent l'état des noeuds : actif, chargé, vide, etc.
- prennent les décisions notamment l'ordonnancement
- les principaux
  - kube-apiserver : front-end d'accès au cluster kubernetes
  - etcd : base (clé,valeur) contenant les informations du cluster
  - kube-scheduler : surveille les demandes de pods et affecte un noeud à ceux qui n'en ont pas

Les composants noeud
- kubelet : sur chaque noeud, on leur envoie des pods et s'assure qu'ils tournent
  - s'appuient sur un *container runtime* (typiquement Docker)

Tous ces éléments communiquent par REST (web service)
- ils peuvent être largement distribués

L’architecture de Kubernetes (2)

Controller(s) : surveille et agit sur l’état du cluster pour maintenir l’état spécifié
- node controller : surveille l’état des noeuds
- replication controller : surveille que le bon nombre de pods s’exécute
- endpoints controller : en charge des endpoints (pods+services)
- cloud-controller-manager : fait l’interface avec le service de cloud sous-jacent

Éléments de Kubernetes

Pod : groupe de containers

créés pour exécuter des containers ensemble

partagent le namespace et les volumes car sont hébergés sur une même machine virtuelle

correspondent à un(e) (morceau d') application

avec un (éventuel) tag de version ou autre

https://kubernetes.io/docs/concepts/workloads/pods/

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Description des applications

Déploiement
- décrit comment une application doit s'exécuter
  - ensemble des containers/pods la composant et comment les démarrer
  - nombre de copies de chaque container/pod
- K8S s'assure que le déploiement voulu est toujours respecté

Si réplication de pods demandée, K8S crée un contrôleur replicaset en charge de s'assurer qu'il y en a toujours assez
- en charge d'un ensemble de pods tous identiques

Voir les déploiements en cours dans un cluster K8S

```
kubectl get deployments
kubetctl get replicasets
```

Exemple

https://kubernetes.io/fr/docs/concepts/workloads/controllers/deployment/

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Accès aux applications

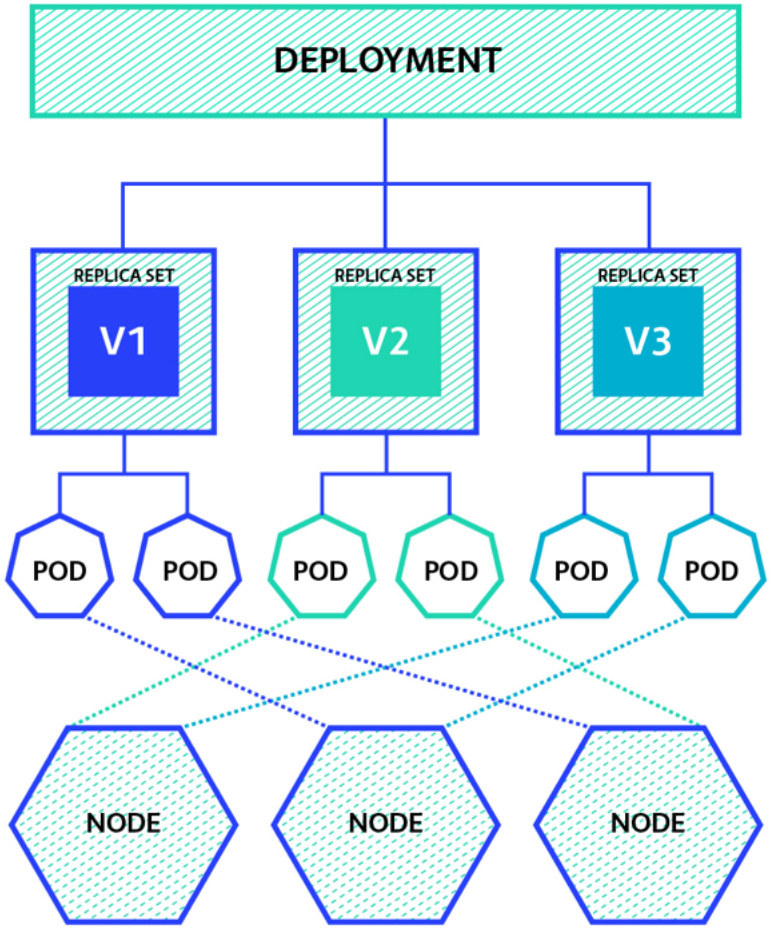
Application (~=pods) =
- ensemble de containers en cours d'exécution
- ressources partagées entre les containers (stockage)
- communication entre les containers (ports)

Comment interagir avec l'application ?

Les services
- permet de spécifier les entrées (port) de l'application
  - K8S se charge de router les requêtes vers un déploiement (pod) capable d'y répondre
- mapping au niveau de l'application entre le port exporté et ceux des containers du pod

Co-existence de versions

Kubernetes permet de faire exécuter simultanément des versions différentes du même service



Durée de vie d'un pod variable
- très court : micro services
- long : application persistente

Utilisation

Cluster Kubernetes utilisable avec Docker Desktop
  facile et « rapide » pour démarrer et prototyper
Sur les ensipcs, minikube permet de lancer un cluster Kubernetes local

Outil `kubectl` en ligne de commande pour interagir avec le master Kubernetes

Définition de configuration
  en passant des ordres et arguments par la ligne de commande
  par des fichiers YAML/JSON
    de nombreux exemples dans la documentation

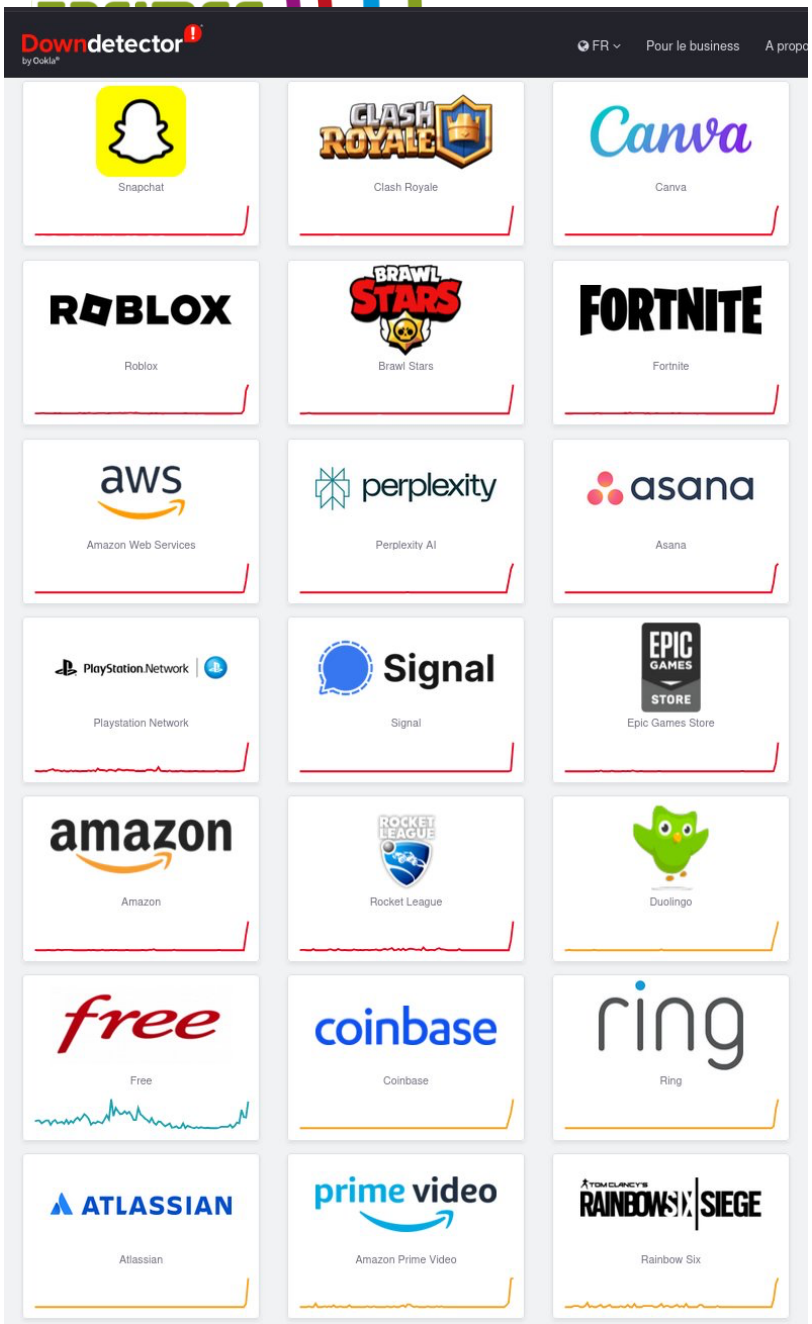
```
kubectl apply -f config.{json, yaml}
```

Cloud et Kubernetes

Création et administration d'un cluster Kubernetes
- pile logicielle complexe
- environnement très évolutif
- de très nombreux outils pour aider le développeur/opérateur
- IaC (Infrastructure as Code)

Utilisation d'un cluster clé en main (facturation à la ressource)
- Google Kubernetes Engine (GKE)
  - par les créateurs de Kubernetes, la référence
  - cluster autoscaling
- Amazon Elastic Kubernetes Service (EKS)
  - sur plusieurs zones de disponibilité
  - peut utiliser les instances spot ($$$)
- Azure Kubernetes Service
  - support de windows

Panne AWS us-east-1



Breakdown of **AWS outage** in simple words

1. Sunday night, a DNS problem hit **AWS** - DynamoDB endpoint lost

2. This meant services couldn't find DynamoDB (a database that stores tons of data).

3. **AWS** fixed the DNS issue in about 3 hours.

4. But then EC2 (the system that creates virtual servers) broke because it needs DynamoDB to work.

5. Then the system that checks if network load balancers are healthy also failed.

6. This crashed Lambda, CloudWatch, SQS, and 75+ other services - everything that needed network connectivity.

7. This created a chain reaction - servers couldn't talk to each other, new servers couldn't start, everything got stuck

8. AWS had to intentionally slow down EC2 launches and Lambda functions to prevent total collapse.

9. Recovery took 15+ hours as they fixed each broken service while clearing massive backlogs of stuck requests.

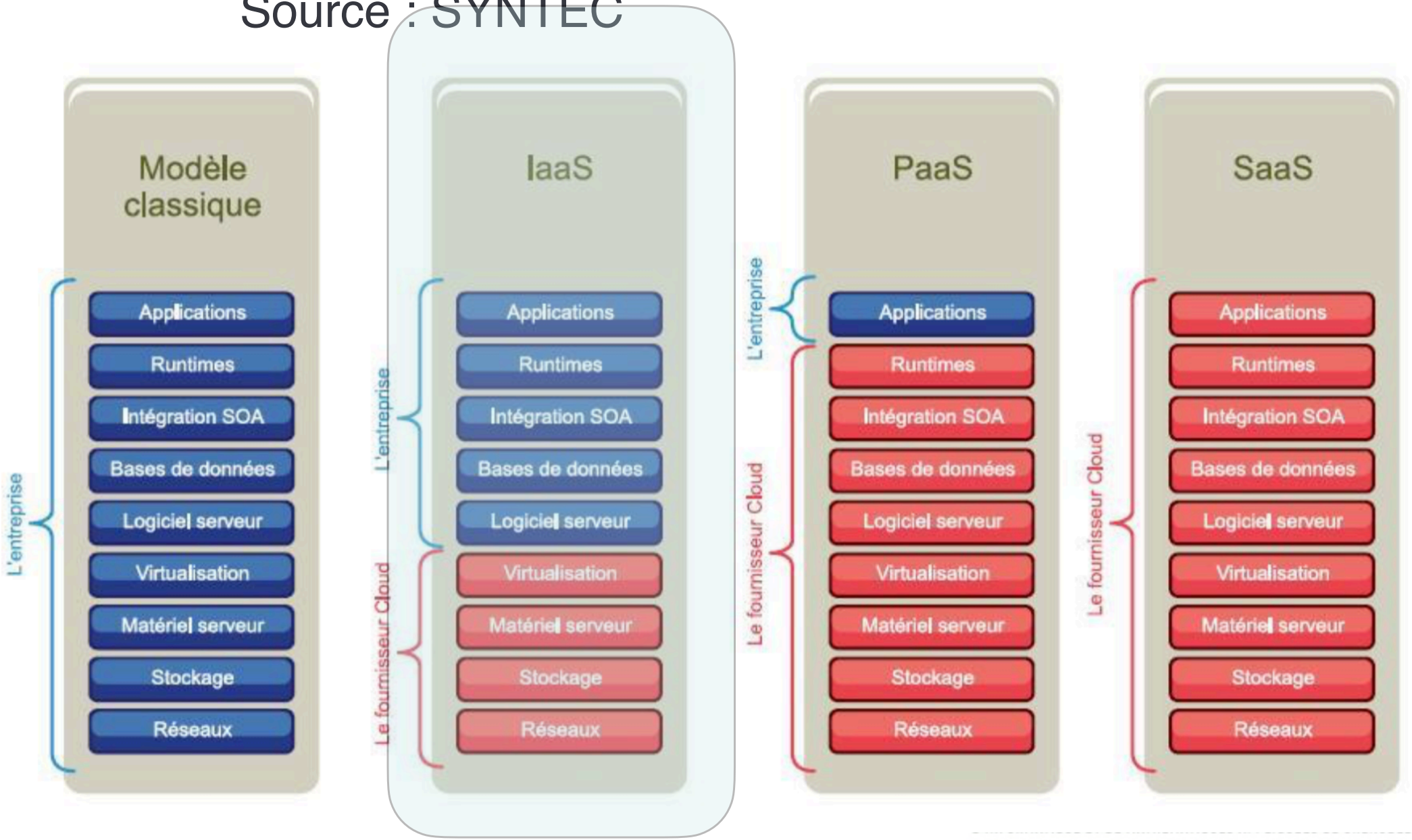
This outage impacted: Snapchat, Roblox, Fortnite, McDonald's app, Ring doorbells, banks, and 1,000+ more websites.

This all happened in one AWS region (us-east-1).

This is why multi-region architecture isn't optional anymore.

Vision schématique

Source : SYNTEC

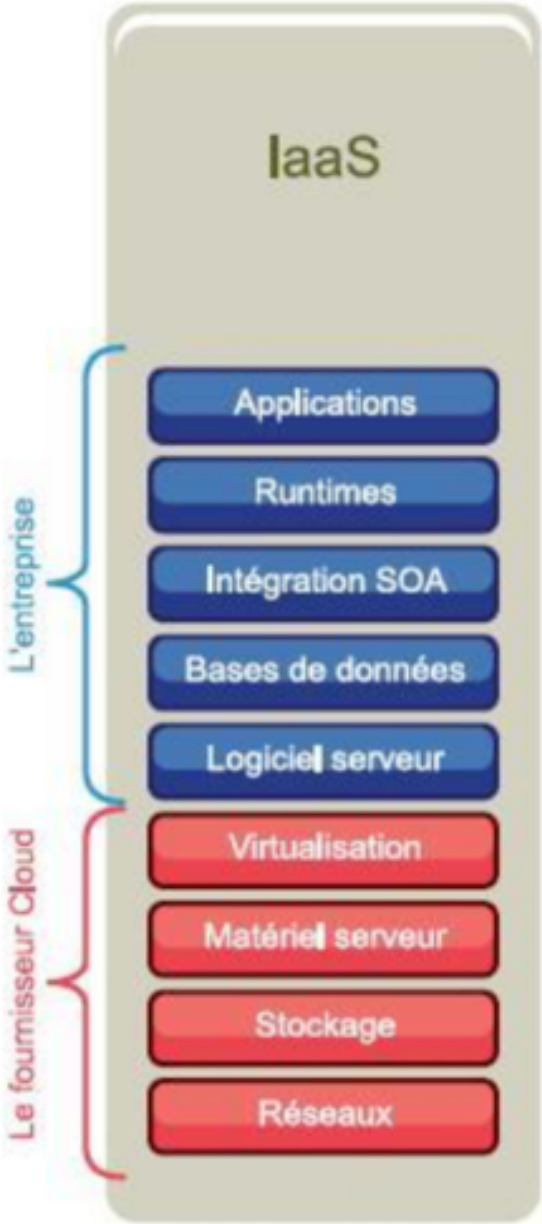


Infrastructure as a Service

- Historiquement le premier type fourni
  - d'abord pour le HPC, calcul puis stockage
  - ISP avec hébergement, milieu des 90s
- Service fourni : stockage, couche de virtualisation, matériel serveur, réseau
- permet de ne pas avoir pas à acheter et gérer de matériels
  - la notion de ressource virtuelle est centrale
    - permet le partage au niveau du datacenter

IaaS : implémentation

- Pas de standards dans le déploiement
  - vision « bas niveau » des ressources
- Notion de machines
  - virtuelles ou physiques
- Isolation (théorique?) entre les clients
  - disque
  - réseau
- Solution pour cloud privé : OpenStack, Xen, …



IaaS : services de base

- machines virtuelles (ou physiques)
- espace de stockage
  - vue fichier ou clé,valeur
  - stockage longue durée
  - (R)DBMS
- Services réseau de base : DNS, IP

Stockage en bloc

- Abstraction fournie : bloc (ensemble de secteurs sur un disque traditionnel)
  - abstraction de très bas niveau (utilisée pour construire un système de fichiers)
- Caractéristiques
  - peut être accédé à distance
    - mais dans le même datacenter
  - une machine virtuelle à la fois!
  - prise d'images possible
- Utilisé pour construire un FS ou une base de données
- Exemples : AWS elastic bloc store, google persistent disk

Stockage en fichier

- Systèmes de fichiers à distance (comme à l'école…)
  - Généralement restreint à la même région
- Partageable entre machines virtuelles
- Système de verrouillage intégré ou externe
- Différentes variantes proposées
  - cohérence et tolérance aux fautes
  - optimisé pour certains usages (HPC, big data (hadoop))

- Exemples
  - NFS, CIFS
  - Lustre, HDFS
  - Google filestore, AWS elastic file system

Stockage par objets

- on dit aussi par blob (binary objet)
- taille quelconque, accès par clé
  - pas de hiérarchie entre les objets (≠FS)
  - des méta-données supplémentaires utilisables
- Caractéristiques
  - hautement scalable (taille, nombre, accès)
  - simplicité
  - génériques
  - fiabilité++ efficacité++
  - peu cher (optimisable côté fournisseur) : stockage + transfert

Stockage par objets (2)

- Accessible sous la forme de services webs
- Opérations : CRUD
  - Create, Read, Update, Delete
    - Update = réécriture totale, atomique
  - copie, lecture partielle, effacement multiples
- API AWS S3 est devenue un standard de fait
  - mais des détails d'implantation peuvent différer

- Accès concurrents possibles, pas de système de verrouillage fourni

IaaS : avantages

- Évolutivité
  - matérielle
  - logicielle (de base)
- Scalabilité
  - peut même être automatique

IaaS : inconvénients

- Single Point of Failure
  - Service Level Agreement
- Vendor lock-in
- Confidentialité des données et traitements
  - architecture partagée entre différents clients
  - législation
    - du site d'hébergement
    - des pays traversés
- Anticipation des coûts
  - intéressant seulement si infrastructure peu exploitée ?

IaaS : critères de choix

- localisation des ressources
- taux de disponibilité
- dimensionnement des serveurs
- qualité des liens réseau
- support système associé

- et le coût, bien évidemment