

# AWS

Christophe Dufour



# Les fondamentaux AWS - Partie I

Régions, IAM et EC2

# Régions

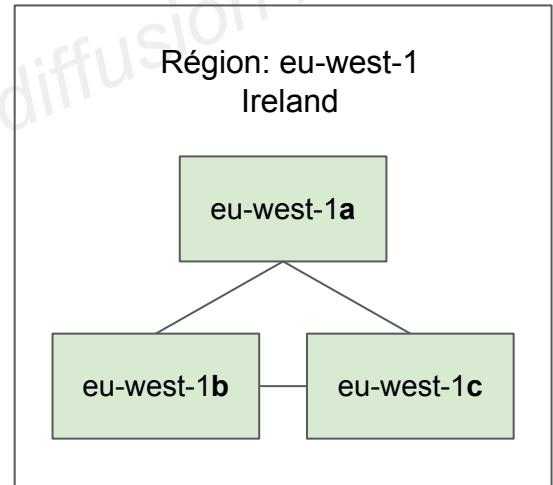
- AWS dispose de régions partout dans le monde
- Noms des régions: eu-central-1, eu-south-2, etc.
- Une région est un cluster de data centers
- La plupart des services sont attachés à une région



US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Osaka)	ap-northeast-3
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Milan)	eu-south-1
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1

# Zones de disponibilités (AZ)

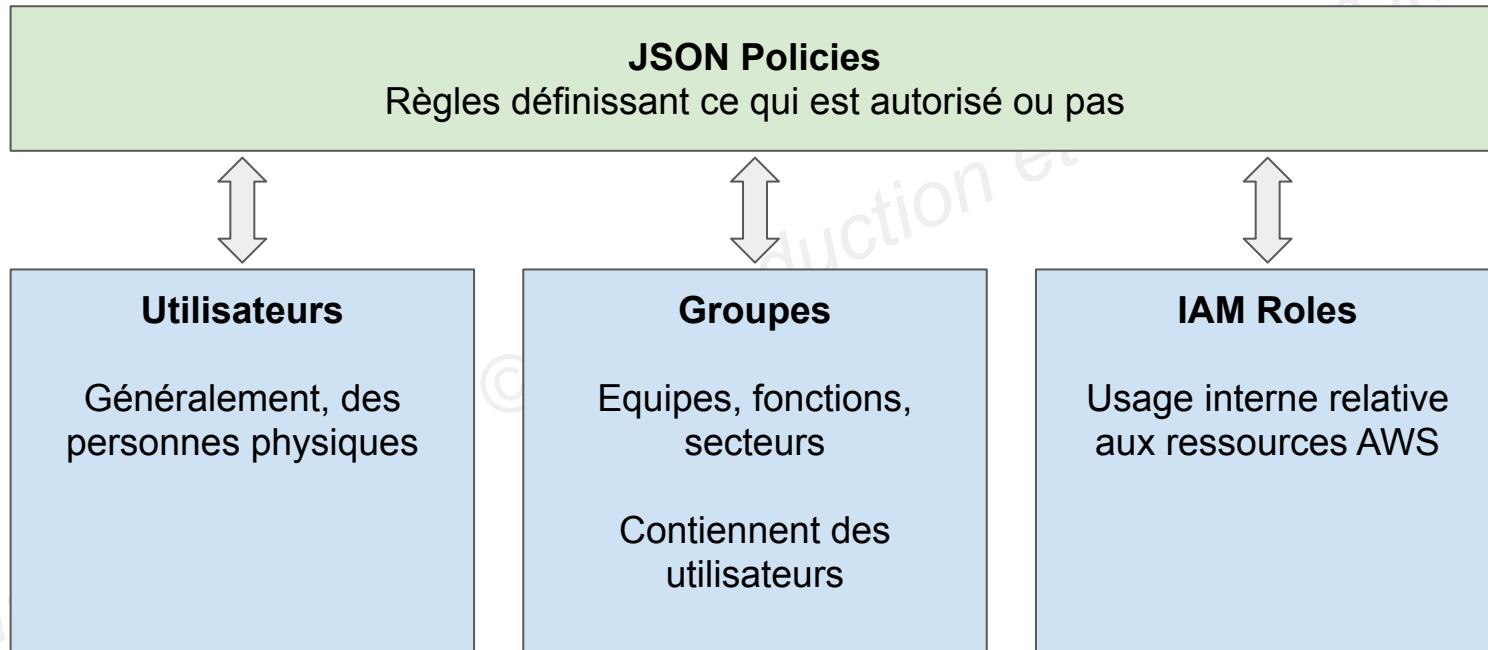
- Chaque région a plusieurs zones de disponibilités (généralement 3, 2 au minimum, 6 au plus)
- Chaque AZ représente un ou plusieurs data centers disposant d'une infrastructure électrique, réseau et d'une connectivité
- AZ physiquement séparées par mesure de sécurité en cas de désastre
- Bande passante très élevée, réseau à latence extrêmement faible



# IAM Introduction

- IAM (Identity and Access Management)
- Service central dans AWS
- Regroupe les éléments suivants auxquelles un ensemble de permissions peuvent être attachées:
  - Utilisateurs
  - Groupes
  - Rôles
- Le compte **Racine (Root)** ne devrait jamais être utilisé (et partagé)
- Les utilisateurs doivent être créés avec les permissions adéquates
- Les stratégies d'accès (policies) sont au format JSON

# IAM Introduction



# IAM Introduction

- IAM a une portée globale (pas restreint à une région)
- Les permissions sont définies par des **Policies** (JSON)
- Un MFA (Accès multi-facteurs) peut être activé
- IAM fournit des règles prédéfinies (“managed policies”)
- Bonne pratique: limiter les permissions données aux utilisateurs à leurs stricts besoins métiers (principe du “least privilege”)

# IAM Federation

- Généralement, les grandes entreprises intègrent à IAM leurs propre base utilisateurs
- Un utilisateur peut ainsi s'identifier à AWS avec ses identifiants entreprise
- La fédération d'identité utilise le standard SAML (Active Directory)

# IAM à retenir

- **Un utilisateur IAM par personne physique**
- **Un rôle IAM par application**
- Les identifiants IAM doivent jamais être partagés
- Les identifiants IAM ne doivent jamais figurés dans le code
- Les identifiants IAM se doivent jamais faire l'objet d'un “commit”
- Ne jamais utiliser le compte Root sauf pour le paramétrage initial
- Ne jamais utiliser les identifiants du compte Root

# Atelier - TP

- Nous allons
  - localiser le service IAM
  - créer un utilisateur (**individual user** vs principal user)
  - créer un groupe
  - lier un utilisateur à un groupe
  - attacher des règles (policies) à un groupe
  - tester

# EC2, c'est quoi ?

- EC2 est l'un des services les plus populaires d'AWS
- En résumé, il permet de:
  - louer des machines virtuelles (EC2)
  - stocker des données sur des disques virtuels (EBS)
  - distribuer la charge sur plusieurs machines (ELB)
  - redimensionner (scale) les services en utilisant un auto-scaling group (ASG)
- Connaître EC2 est indispensable pour comprendre le fonctionnement du Cloud

# Atelier: Démarrer une instance EC2 Linux

- Nous allons lancer notre premier serveur virtuel en utilisant la console AWS
- Nous aborderons un ensemble de paramètres
- Nous apprendrons à démarrer, arrêter, supprimer notre instance

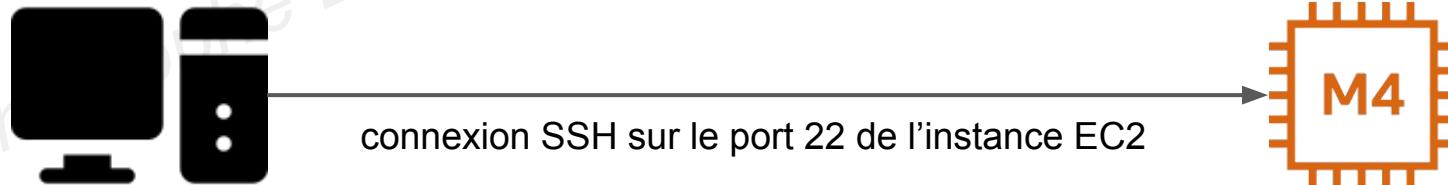
# SSH : tableau récapitulatif

	SSH	Putty	EC2 Instance Connect
Mac	✓		✓
Linux	✓		✓
Win < 10		✓	✓
Win $\geq 10$	✓		✓

# Se connecter en SSH à votre instance EC2

## Linux / Mac OS X

- SSH est outil/protocole incontournable. Il permet de contrôler une machine distante en ligne de commande
- Nous verrons comment nous connecter à une instance EC2 sous Linux/Mac
- Nous verrons comment configurer OpenSSH `~/.ssh/config` pour faciliter la connexion à nos instances EC2



# EC2 Instance Connect

- Permet de se connecter à l'instance directement depuis la navigateur web
- Pas besoin de générer et télécharger un **key file** de connexion
- “En coulisse”, une clé temporaire est chargée dans l'instance EC2 par AWS
- Pas besoin de s'assurer que le port 22 est ouvert
- Ne fonctionne en standard qu'avec l'image Amazon Linux 2

# Introduction aux groupes de sécurité (SG)

- Les **Security Groups** sont un aspect fondamental de la sécurité réseau dans AWS
- Ils déterminent le trafic autorisé en entrée/sortie sur les machines EC2
- C'est la compétence la plus importante à acquérir afin de solutionner un problème réseau
- Nous verrons comment les utiliser afin d'autoriser des transmissions de données en entrée (**inbound**) et en sortie (**outbound**) sur des ports précis

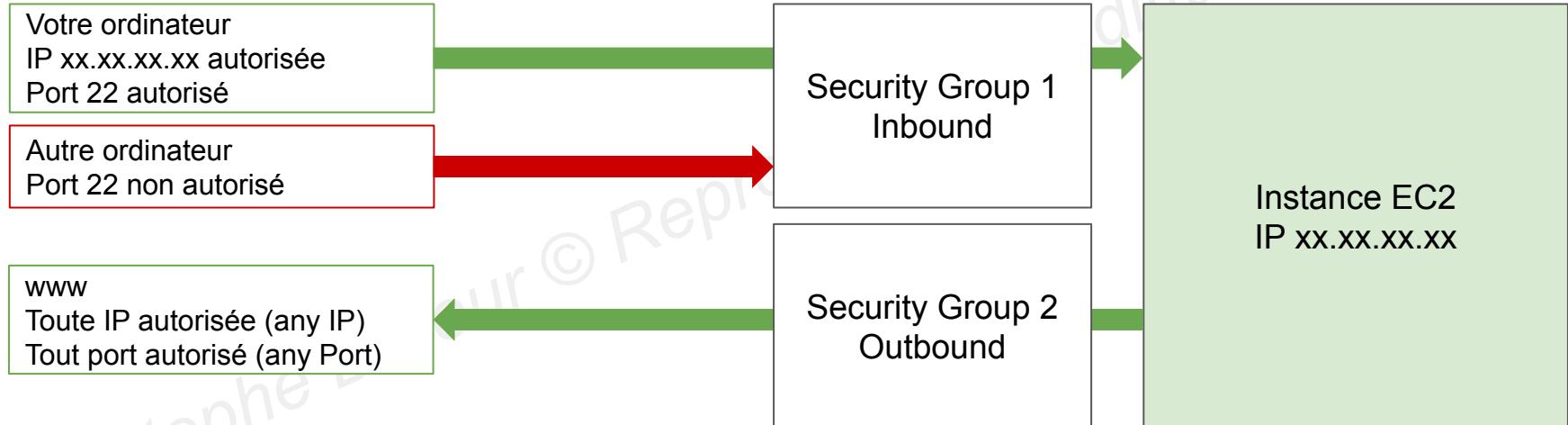


# Groupes de sécurité

- Les **Security Groups** agissent comme des pare-feu sur les instances EC2
- Ils régulent:
  - l'accès aux ports
  - Les plages IP autorisées - IPv4 et IPv6
  - le trafic interne en entrée (depuis une autre instance)
  - le trafic interne en sortie (vers une autre instance)

Inbound rules	Outbound rules	Tags	
Type	Protocol	Port range	Source
HTTP	TCP	80	0.0.0.0/0
HTTP	TCP	80	::/0
SSH	TCP	22	0.0.0.0/0

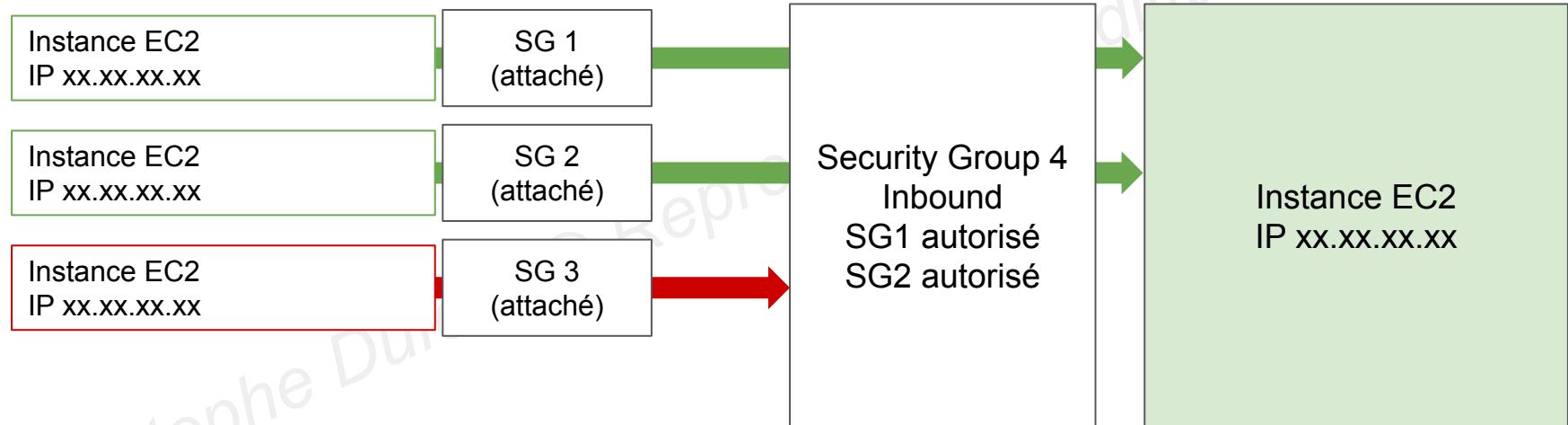
# Groupes de sécurité - schéma



# Groupes de sécurité - bon à savoir

- Peuvent être attachés à plusieurs instances
- Vérrouillés sur une combinaison région/vpc
- Vivent “en dehors” de l’instance EC2. Si un trafic est bloqué, l’instance ne le voit pas
- Bonne pratique: maintenir un SG par accès SSH
- Si votre application n’est pas accessible (time out), c’est probablement un problème de groupe de sécurité
- Si votre application retourne une erreur “connexion refusée”, ou bien elle n’est pas lancée ou bien elle rencontre un problème applicatif
- Tout le trafic en entrée est **bloqué** par défaut
- Tout le trafic en sortie en **autorisé** par défaut

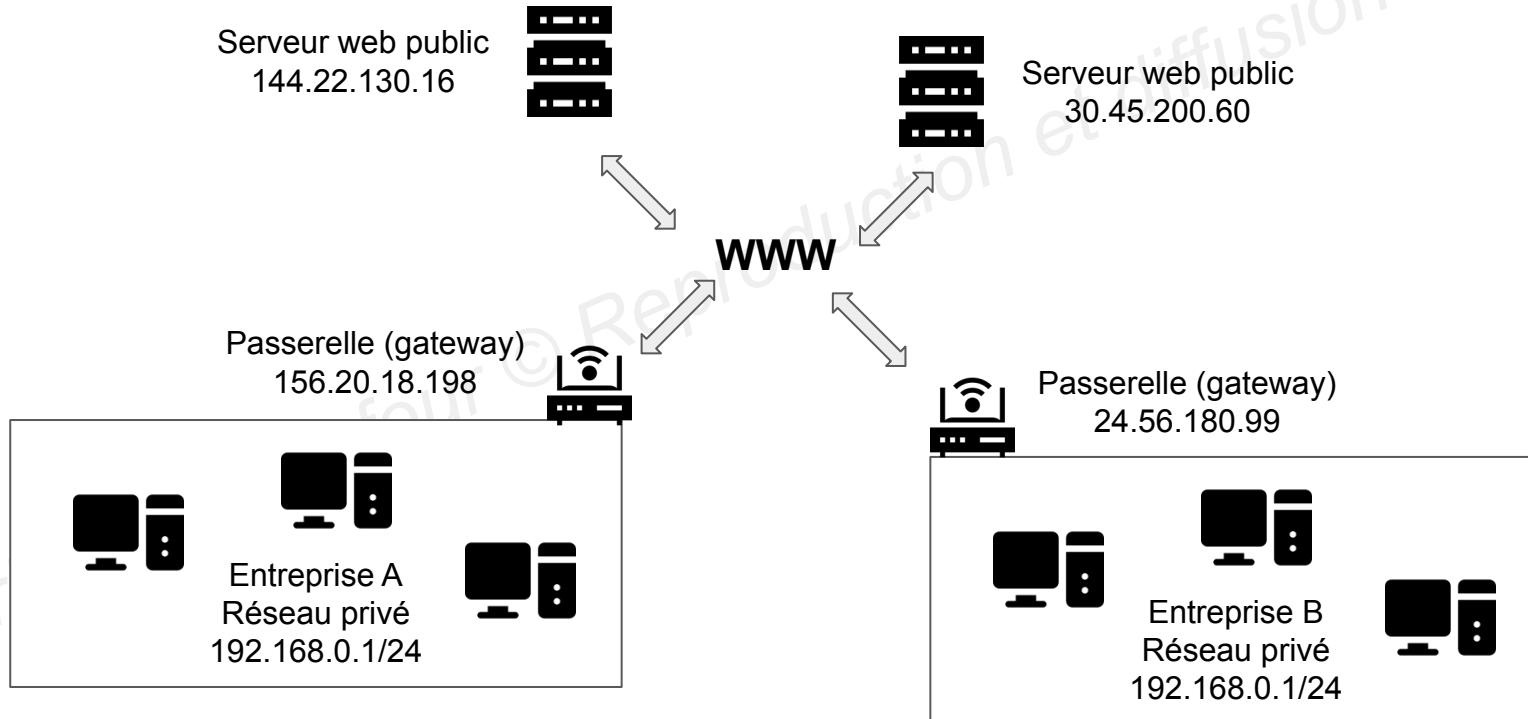
# Référencement d'autres SG - schéma



# IP privée vs publique

- 2 types d'adresse IP
  - IPv4: 192.168.0.29
  - IPv6: fe80::a00:27ff:fe8d:dac8
- Dans ce cours, nous utiliserons uniquement des adresses IPv4
- IPv4 reste le format le plus couramment utilisé.
- IPv6 est plus récent, utile pour les nouveaux besoins d'adressage (Internet of Things - IoT)
- IPv4 offre potentiellement plus de 4 milliards d'adresses IP publiques différentes
  - $[0-255].[0-255].[0-255].[0-255] = 256^4$

# IP privée vs publique - exemple



# IP privée vs publique - différences majeures

- IP publique
  - La machine est identifiable sur internet (www)
  - Doit être unique (deux machines ne peuvent pas avoir la même IP publique)
  - Facilement géo-localisable
- IP privée
  - La machine est identifiable uniquement dans le cadre d'un réseau privé
  - L'adresse doit être unique sur un seul et même réseau privé
  - Deux réseaux privés distincts (entreprises) peuvent avoir les mêmes IPs
  - Les machines se connectant au web utilisent un routeur/passerelle (proxy)
  - Seules des plages spécifiques (192.xx, etc.) peuvent être utilisées en tant qu'IPs privées

# IP élastique (Elastic IP)

- Une instance EC2 se voit assignée une nouvelle adresse IP publique à chaque démarrage (attribution dynamique)
- Une IP élastique correspond à une IP fixe (statique)
- Une IP élastique est une IP publique qu'on détient aussi longtemps qu'elle n'est pas supprimée, c'est une ressource AWS
- Elle peut être attachée à différentes instances mais à une seule à la fois

# IP élastique (Elastic IP) - suite

- Peut masquer l'échec d'une instance/application en étant rapidement et automatiquement assignée à une autre instance
- Par défaut, un maximum de 5 IPs élastiques peut être détenu par compte (il est possible de faire une demande à AWS pour en obtenir davantage)
- De manière générale, il vaut mieux éviter son utilisation:
  - elle reflète souvent une architecture/configuration fébrile
  - une adresse IP publique aléatoire associer à un nom DNS est souvent préférable
  - l'utilisation d'un Load Balancer peut être aussi une solution à privilégier

# IP privée vs publique - Atelier

- Par défaut, une instance EC2 est fournie avec:
  - une adresse IP privée pour des considérations réseau internes à AWS
  - une adresse IP publique aléatoire pour le web
- Une connexion SSH à une instance EC2
  - ne peut pas se faire un utilisant l'IP privée de l'instance (réseaux différents)
  - peut uniquement se faire par utilisation de l'IP publique de l'instance
- Si l'instance est redémarrée, son IP publique change

# Lancer un serveur Apache dans une instance EC2

- Utilisons notre instance EC2
- Nous allons installer un serveur http apache afin d'afficher une page web
- Nous allons créer une page index.html affichant le hostname de notre machine

# EC2 User Data

- Il est possible d'exécuter un script - **EC2 User Data** - au démarrage initial d'une instance
- Ce script est exécuté **seulement une fois, au premier démarrage**
- S'utilise pour automatiser des tâches de d'initialisation telles que:
  - Installation de mises à jour
  - Installation de programmes
  - Téléchargement de fichiers
  - Exécution d'autres scripts, etc.
- EC2 User Data est exécuté en tant qu'utilisateur root

# EC2 User Data - Atelier

- Nous souhaitons qu'un serveur apache soit installé sur notre instance EC2, afin d'afficher une simple page web
- Nous allons écrire un script user-data
- Ce script sera exécuté au premier démarrage

# Types de lancement d'instance EC2 (Launch Types)

- **On Demand Instances**: courte charge de travail, prix prévisible
- **Reserved**: (1 an minimum)
  - **Reserved Instances**: longue charge de travail
  - **Convertible Reserved Instances**: longue charge de travail avec flexibilité
  - **Scheduled Reserved Instances**: par exemple, chaque lundi, de 5 à 7
- **Spot instances**: courte charge de travail, instances “jetables” et bon marché
- **Dedicated Instances**: matériel réservé (pas de partage)
- **Dedicated Hosts**: réservation d'un serveur physique complet, contrôle sur le placement des instances

# EC2 On Demand

- Facturation de ce qui est consommé (par seconde, après la première minute)
- Coût le plus élevé mais pas de paiement en avance ou récurrent
- Pas d'engagement à long terme
- Recommandé pour des charges de travail à court terme et ininterrompue où le comportement de l'application est difficilement anticipable

# EC2 Reserved Instances

- Jusqu'à 75% d'économies par rapport à On Demand
- Paiement en avance avec un engagement à long terme
- La période de réservation peut aller de 1 à 3 ans
- On réserve un type d'instance spécifique (ex: T2)
- Recommandé pour des applications de nature plutôt stable (base de données par exemple)
- Convertible Reserved Instances
  - Type d'instance EC2 modifiable (exemple: T2 -> M4)
  - Jusqu'à 54% d'économies
- Scheduled Reserved Instances
  - lancement durant une plage chronologique déterminée
  - adapté à un usage fractionnaire (jour/semaine/mois)

# EC2 Spot Instances

- Jusqu'à 90% d'économies par rapport à On Demand
- Instances qu'on peut se permettre de "perdre" à tout moment
- Tarif le plus attrayant dans AWS
- Utile pour des charges de travail peu sensibles à l'échec
  - Batchs
  - Analyse de données
  - Traitement d'image
  - ...
- Pas adapté à des travaux critiques ou des bases de données
- Combinaison intéressante: Reserved Instances pour la charge de base constante + On-Demand et Spot pour les pics

# EC2 Dedicated Hosts

- Serveur EC2 physique dédié
- Contrôle total du placement des instances EC2
- Visibilité sur les couches bas niveau, sur les aspects matériels
- Période de réservation: 3 ans
- Plus coûteux
- Utile, par exemple, pour:
  - des programmes ayant un modèle de licence compliqué
  - des entreprises/organisation ayant des besoins pointus en termes de réglementation et de conformité

# EC2 Dedicated Instances

- Instances exécutées sur du matériel dédié
- Matériel pouvant être partagé par des instances du même compte
- Pas de contrôle sur le placement des instances

Characteristic	Dedicated Instances	Dedicated Hosts
Enables the use of dedicated physical servers	x	x
Per instance billing (subject to a \$2 per region fee)	x	
Per host billing	x	
Visibility of sockets, cores, host ID	x	
Affinity between a host and instance	x	
Targeted instance placement		x
Automatic instance placement	x	x
Add capacity using an allocation request		x

# Quel hébergement me convient le mieux ?

- **On demand:** je prends une chambre quand je veux, j'y reste autant que je veux, je paie le prix fort
- **Reserved:** je planifie ma réservation, plus la réservation est longue plus les économies sont importantes
- **Spot instances:** l'hôtel permet la location de chambres vides et le plus offrant du moment les conserve. Je peux perdre ma chambre à tout instant
- **Dedicated Hosts:** je réserve la totalité du bâtiment



# Comparaison de prix - exemple

Price Type	m4.large - us-east-1	Price (per hour)
On-demand		\$0.10
Spot Instance (Spot Price)		\$0.032 - \$0.045 (up to 90% off)
Spot Block (1 to 6 hours)		~ Spot Price
Reserved Instance (12 months) – no upfront		\$0.062
Reserved Instance (12 months) – all upfront		\$0.058
Reserved Instance (36 months) – no upfront		\$0.043
Reserved <b>Convertible</b> Instance (12 months) – no upfront		\$0.071
Reserved <b>Dedicated</b> Instance (12 months) – all upfront		\$0.064
Reserved <b>Scheduled</b> Instance (recurring schedule on 12 months term)		\$0.090 – \$0.095 (5%-10% off)
Dedicated Host		On-demand price
Dedicated Host Reservation		Up to 70% off

# Tarification EC2

- La tarification EC2 varie selon plusieurs critères
  - région
  - type d'instance
  - launch type
  - Système d'exploitation (Linux, Windows, etc.)
- Tarification à la seconde, après un délai de soixante secondes
- La tarification peut également être impactée par d'autres facteurs:
  - stockage, transfert de données, IP fixe, load balancing
- On ne paye rien pour une instance arrêtée

# Tarification EC2 - exemple

# Amazon Machine Images (AMI)

- Comme nous l'avons vu, AWS fournit des images prédéfinies telles que:
  - Ubuntu
  - Fedora
  - RedHat
  - Windows
  - Etc.
- Par la suite, ces images peuvent être personnalisées
  - Au premier démarrage de l'instance (EC2 User Data)
  - Durant toute la durée de vie de l'instance
- AWS permet de créer ses propres AMI
- Les AMIs peuvent être construites pour des machines Linux ou Windows

# Pourquoi utiliser une AMI personnalisée

- Utiliser une AMI personnalisée peut fournir les avantages suivants:
  - Logiciels, dépendances préinstallés
  - Démarrage plus rapide (plus besoin d'un long script User Data)
  - Configuration pour du monitoring
  - Aspects sécuritaires
  - Contrôle de la maintenance et des mises à jour des AMIs dans le temps
  - Intégration d'Active Directory
  - Installation d'application rapide
  - Optimisation des logiciels exécutés
- Les AMIs dépendent des régions AWS

# Instances EC2 - vue d'ensemble

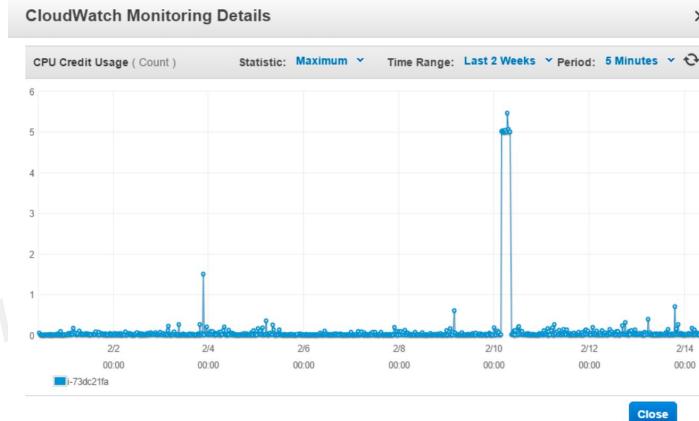
- Les instances ont 5 propriétés distinctes
  - la RAM (type, quantité, génération)
  - le CPU (type, fréquence, génération, nombre de coeurs)
  - Les entrées/sorties (I/O, performances disque, EBS, optimisations)
  - Le réseau (bande passante, latency)
  - Le GPU (Graphical Processing Unit)
- Vaste choix: <https://aws.amazon.com/ec2/instance-types/>
- <https://ec2instances.info/> récapitulent les types d'instance
- R/C/P/G/H/I/F/Z/CR sont spécialisées en RAM, CPU, I/O, Network, GPU
- Les instances M sont équilibrées
- Les instances T2/T3 sont “burstable”

# Instances à capacité extensible (burstable, T2)

- AWS propose le concept d'instance “burstable” (Machines T2)
- En substance, cela signifie que l'instance offre des performances CPU accrues
- Quand la machine doit faire face à un traitement inattendu (pic de charge par exemple), elle peut étendre sa capacité de calcul, son CPU peut devenir très performant
- Quand la machine étend sa capacité (bursts), elle a recourt à des “burst credits”
- Quand tous les crédits sont épuisés, le CPU devient “faible”
- Quand la machine cesse d'étendre sa capacité, les crédits se reconstituent

# Instances à capacité extensible (burstable, T2)

- Ce type d'instance est capable de gérer un trafic inattendu et fournir la garantie de le faire correctement
- Si l'instance est constamment en manque de crédits, il est nécessaire de s'orienter vers un autre type d'instance (non-burstable)



# Crédits CPU

Instance type	CPU credits earned per hour	Maximum earned credits that can be accrued*	vCPUs	Baseline utilization per vCPU
<b>T2</b>				
t2.nano	3	72	1	5%
t2.micro	6	144	1	10%
t2.small	12	288	1	20%
t2.medium	24	576	2	20%**
t2.large	36	864	2	30%**
t2.xlarge	54	1296	4	22.5%**
t2.2xlarge	81.6	1958.4	8	17%**
<b>T3</b>				
t3.nano	6	144	2	5%**
t3.micro	12	288	2	10%**
t3.small	24	576	2	20%**
t3.medium	24	576	2	20%**
t3.large	36	864	2	30%**
t3.xlarge	96	2304	4	40%**
t3.2xlarge	192	4608	8	40%**

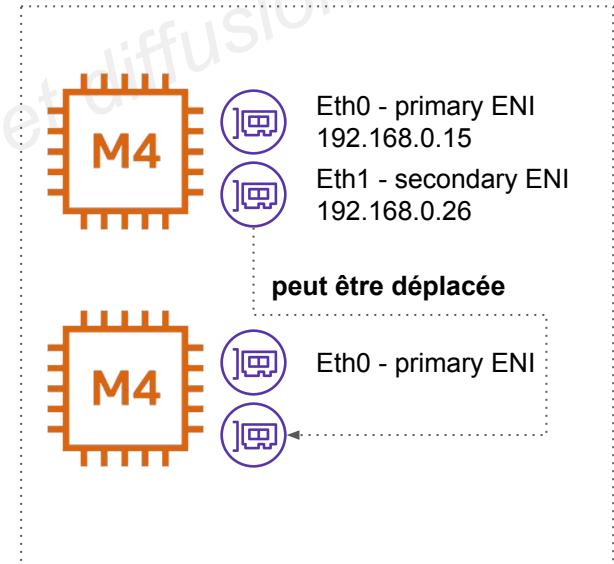
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html>

# T2 sans limite (unlimited)

- Nov 2017: il est possible d'avoir un crédit burst illimité
- On paye un surcoût, lorsque les crédits sont épuisées, les performances ne sont pas amoindries
- C'est une relativement nouvelle fonctionnalité, il s'agit de surveiller correctement l'instance afin de ne pas être surpris par des coûts importants

# Elastic Network Interfaces (ENI)

- Composant logique dans un VPC. Il représente une **carte réseau virtuelle**
- L'ENI peut avoir les propriétés suivantes:
  - une IPv4 privée primaire, une ou plusieurs IPv4 secondaires
  - une élastique IPv4 par IPv4 privée
  - une IPv4 publique
  - un ou plusieurs groupes de sécurité
  - une adresse MAC
- On peut créer des ENI séparément et les attacher à la volée à des instances
- Liée à une zone de disponibilité (AZ) précise



# EC2 - Checklist

- Savoir comment se connecter en SSH à une instance EC2
- Savoir utiliser correctement les groupes de sécurité
- Connaître les différences essentielles entre IP privée, IP publique et IP élastique
- Savoir utiliser le User Data pour personnaliser une instance au premier démarrage
- Savoir qu'on peut construire une AMI personnalisée pour enrichir l'OS servant de base à une instance
- Les instances EC2 sont facturées à la seconde, on les crée et les détruit avec une grande facilité. Bienvenue dans le Cloud !

# Les fondamentaux AWS - Partie II

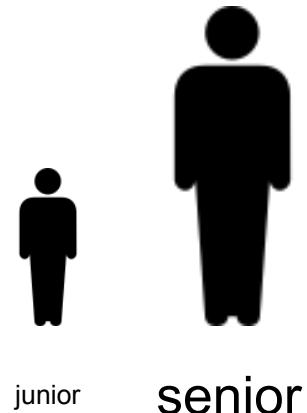
Load Balancing, Auto Scaling Groups et volumes EBS

# Scalabilité et haute disponibilité

- La scalabilité signifie qu'une application / un système peut gérer des charges plus importantes en s'adaptant
- Deux types de scalabilité:
  - verticale
  - horizontale (élasticité)
- La scalabilité est liée à la notion de haute disponibilité mais s'en distingue
- Pour bien comprendre ces notions, prenons l'exemple d'un centre d'appels...

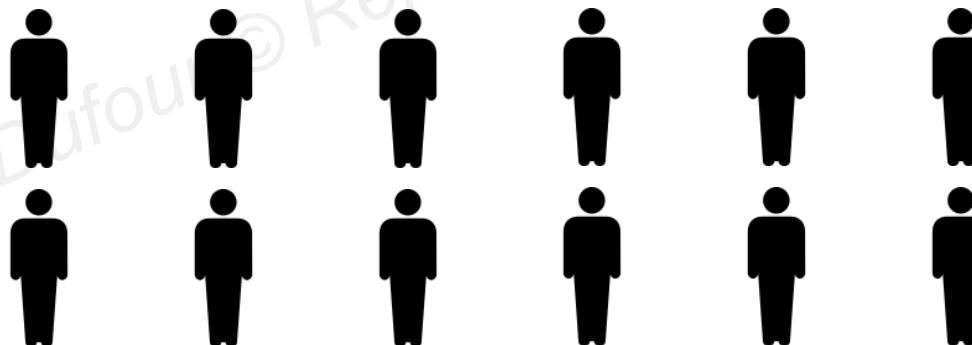
# Scalabilité verticale

- La scalabilité verticale signifie que l'instance grandit en taille/capacité
- Par exemple, une application s'exécute sur une t2.micro
- L'exécuter sur une t2.large correspond un “vertical scaling”
- La scalabilité verticale est très commune pour des systèmes non distribués telle qu'une base de données
- RDS, ElastiCache sont des services Amazon qui peuvent “scaler” verticalement
- Il y a généralement une limite à cette verticalité (limite matérielle)



# Scalabilité horizontale

- La scalabilité horizontale signifie augmenter le nombre d'instances / systèmes pour l'application
- Implique des systèmes distribués
- Très commun pour les applications web et les applications modernes
- Le cloud permet de recourir facilement à cette scalabilité



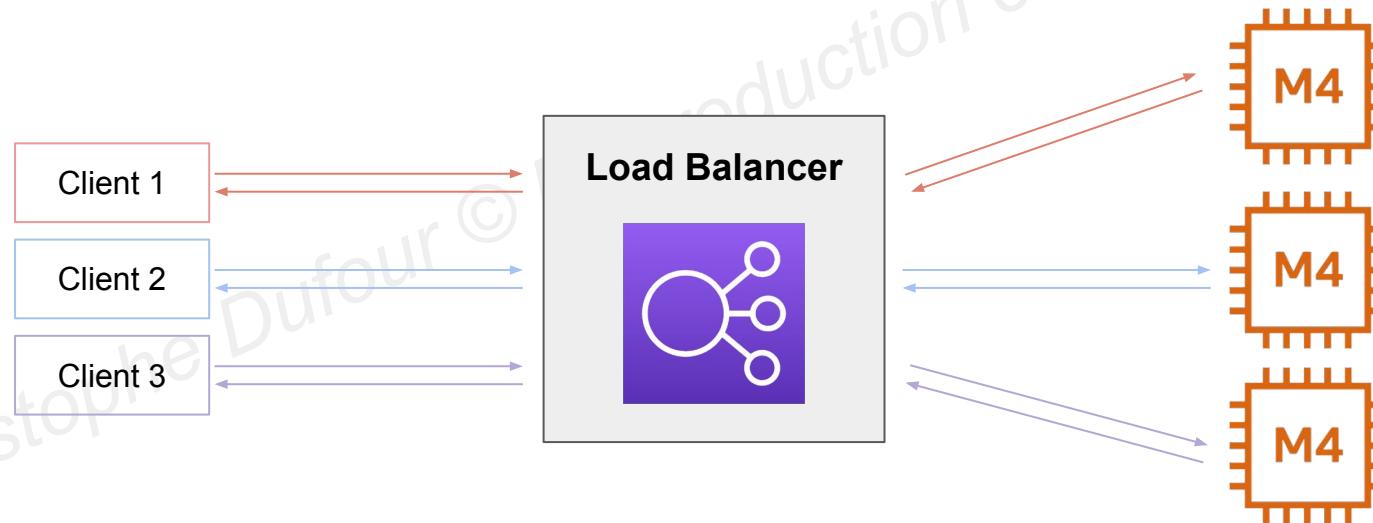
# Haute disponibilité

- La haute disponibilité va généralement de pair avec la scalabilité horizontale
- La haute disponibilité signifie que l'application/système est exécutée dans au moins deux data centers différents (AZ)
- Le but de la haute disponibilité est de survivre à la perte d'un data center
- Elle peut être passive (pour RDS Multi AZ par exemple)
- Elle peut être active (pour la scalabilité horizontale par exemple)



# Qu'est-ce que le load balancing ?

- Les load balancers sont des serveurs qui, en amont, relayent le trafic internet vers de multiples serveurs (instances EC2) en aval



# Pourquoi utiliser un load balancer ?

- Répartir la charge sur de multiples instances
- Exposer un seul point d'accès (DNS) à l'application
- Gérer les instances en échec de manière transparente
- Effectuer des “health check” régulier sur les instances
- Fournir une terminaison SSL (https) aux applications web
- Renforcer la persistence de session (cookies)
- Haute disponibilité multi zones
- Séparer le trafic public du trafic privé

# Pourquoi utiliser un load balancer EC2 ?

- Un ELB (EC2 Load Balancer) est un load balancer géré par AWS
  - AWS garantie son bon fonctionnement
  - AWS s'occupe des mises à jour, de la maintenance, de la haute disponibilité
  - AWS se limite à fournir quelques possibilités de paramétrage
- Il est moins coûteux de mettre en place son propre LB mais beaucoup plus d'effort à fournir
- Un ELB est intégré aux autres services AWS

# Health Checks

- Les health checks sont cruciaux pour les load balancers
- Ils permettent aux LB de savoir si les instances vers lesquelles ils dirigent le trafic sont disponibles et peuvent donc leur répondre
- Le health check est fait sur un port et sur une route (/health par exemple)
- Si la réponse n'est pas 200 (OK), l'instance est considérée **unhealthy**



# Types de load balancers dans AWS

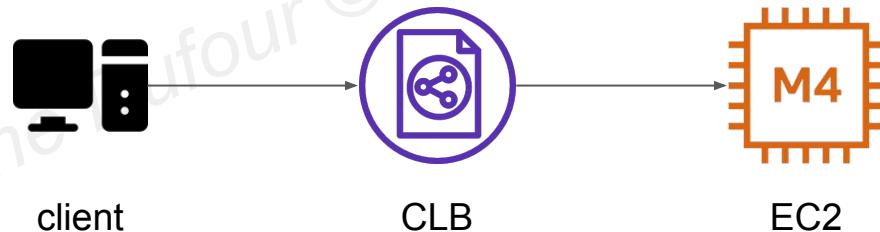
- AWS gère **3 types de LB**
- Classic Load Balancer (v1 - ancienne génération) - 2009
  - HTTP, HTTPS, TCP
- Application Load Balancer (v2 - nouvelle génération) - 2016
  - HTTP, HTTPS, WebSocket
- Network Load Balancer (v2 - nouvelle génération) - 2017
  - TCP, TLS, UDP
- De façon générale, il est recommandé d'utiliser la nouvelle génération de load balancers qui offrent plus de fonctionnalités
- Il est possible de paramétrier des ELB **internes** (privés) et **externes** (publics)

# Load Balancer - bon à savoir

- Les LBs peuvent “scaler” mais pas instantanément. Un demande peut être adressée à AWS
- Dépannage
  - les erreurs 4xx sont des erreurs côté client (requête sur une ressource inexistante)
  - les erreurs 5xx sont des erreurs côté serveur (l’application rencontre un problème)
  - L’erreur 503 renvoyé par le LB signifie ou bien qu’il est surchargé ou bien qu’il n’a pas pu atteindre la cible (instance) vers laquelle il doit diriger le trafic
- Surveillance
  - Les logs d’accès d’un ELB contiennent toutes les requêtes
  - CloudWatch Metrics fournit des statistiques aggrégées (ex: nombre de connections)

# Classic Load Balancer (v1)

- Supporte TCP (couche 4), HTTP et HTTPS (couche 7)
- Health checks basés sur TCP ou HTTP
- Hostname fixe xxx.region.elb.amazonaws.com



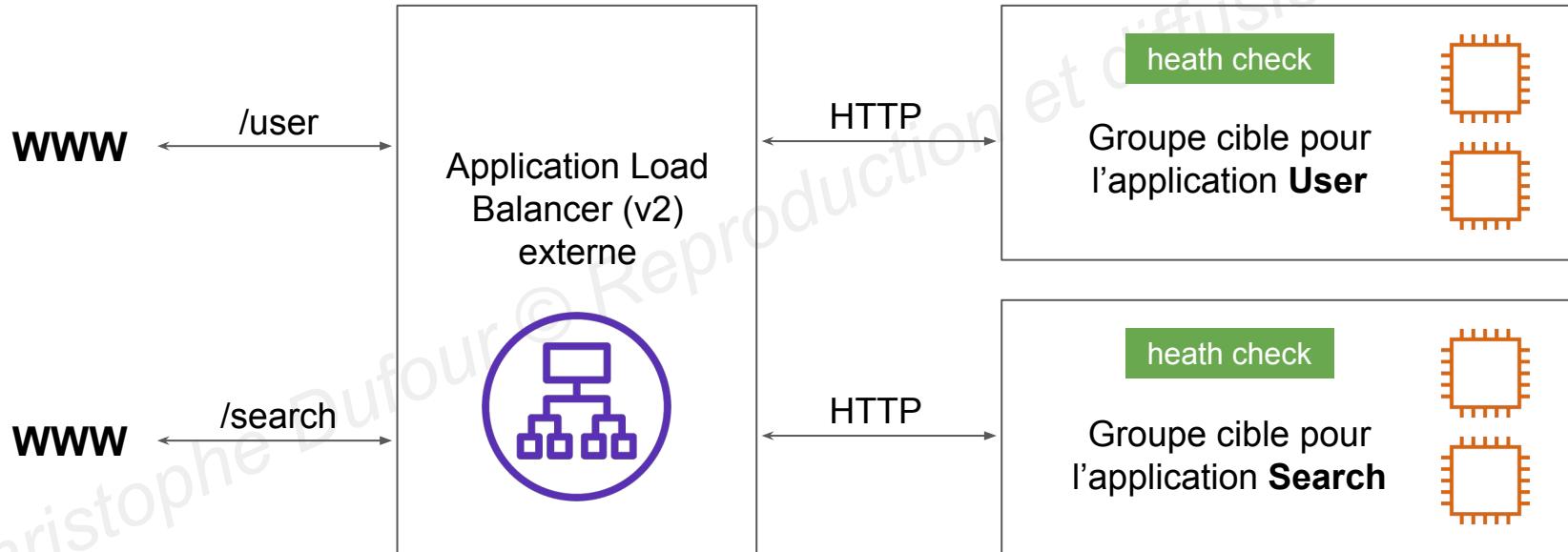
# Application Load Balancer (v2)

- Un ALB est de niveau 7 (HTTP)
- Distribution de la charge vers de multiples applications HTTP réparties sur différentes machines (groupes cibles - target groups)
- Distribution de la charge vers de multiples applications sur la même machine (ex: conteneurs)
- Supporte HTTP/2 et WebSocket
- Supporte les redirections (ex: HTTP vers HTTPS)

# Application Load Balancer (v2)

- Tables de routage vers différents groupes cibles
  - Routage basé sur le chemin url (example.com/users, example.com/posts)
  - Routage basé sur le hostname (one.example.com, other.example.com)
  - Routage basé sur la chaîne de recherche (QueryString) ou les entêtes (example.com?id=1)
- Les ALB sont bien adaptés aux micro-services et aux applications basées sur des conteneurs (ex: Docker et Amazon ECS)
- L'ALB offre une fonctionnalité de mapping de port pour rediriger le trafic sur un port dynamique dans ECS
- Par comparaison, il faudrait plusieurs CLB par application

# ALB - trafic basé sur HTTP

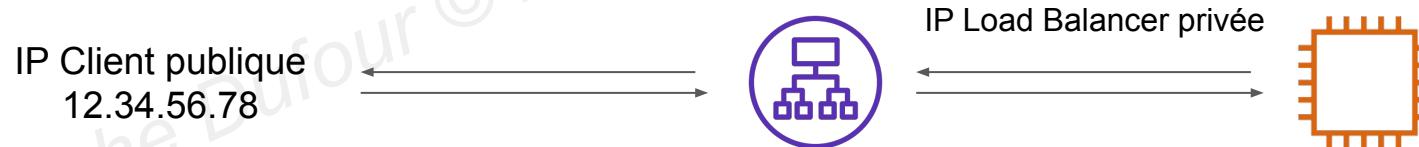


# Application Load Balancer (v2) - Groupes cibles

- Les groupes cibles (target groups) peuvent être:
  - des instances EC2 (gérées par Auto Scaling Group)
  - des tâches ECS (gérées par ECS)
  - des fonctions Lambda
  - des adresses IP (devant être privées)
- Un ALB peut diriger le trafic sur différents groupes cibles
- Les Health Checks se font au niveau du groupe cible

# Application Load Balancer (v2) - bon à savoir

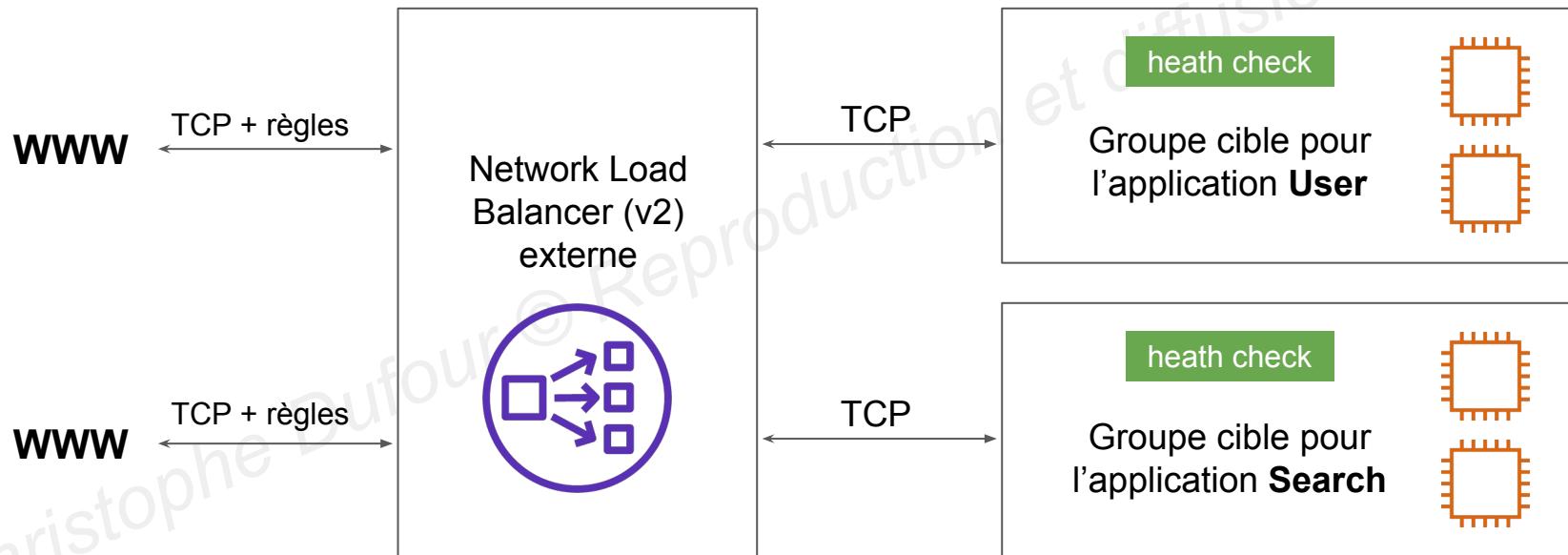
- Hostname fixe (xxx.region.elb.amazonaws.com)
- Les serveurs applicatifs ne voient pas directement l'IP du client
  - La véritable IP du client est insérée dans l'entête HTTP X-Forwarded-For
  - On peut aussi obtenir le port (X-Forwarded-Port) et le protocole (X-Forwarded-Proto)



# Network Load Balancer (v2)

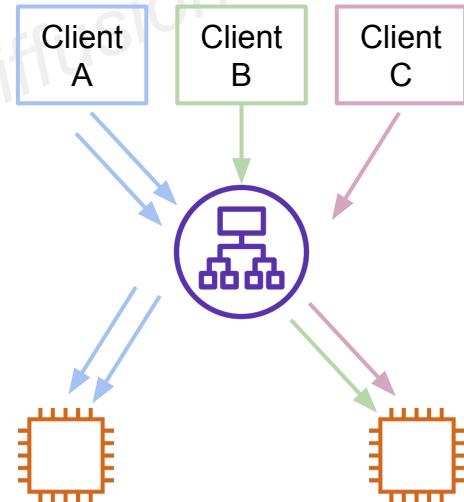
- Un NLB (couche 4) permet:
  - la redirection du trafic TCP et UDP vers des instances
  - la gestion des millions de requêtes par seconde
  - une latence de ~ 100ms (contre 400ms pour un ALB)
- Un NLB dispose d'une IP statique par AZ et support l'attribution d'une IP élastique
- Performances très élevées (trafic TCP, UDP)
- Non inclu dans le AWS free tier

# NLB - trafic basé sur TCP



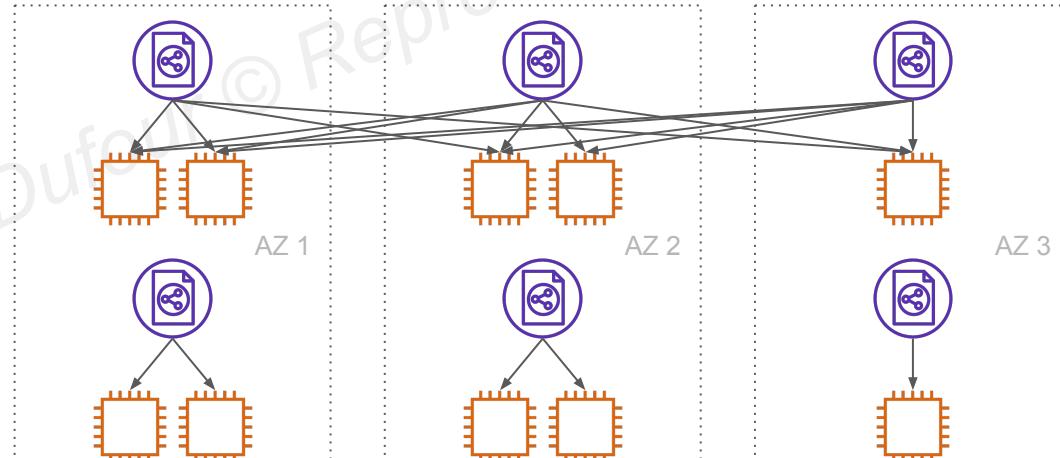
# Persistence (stickiness) des LB

- Il est possible de mettre en place un persistence afin que le même client soit toujours redirigé vers la même instance située en aval du LB
- Réalisable avec les CLB et les ALB
- Nous avons le contrôle sur la date d'expiration du “cookie” utilisé par assurer cette persistence
- Use case: s’assurer que l’utilisateur ne perd pas sa session
- Activer la persistence peut générer un déséquilibre dans la répartition de la charge entre instances



# Répartition de charge sur plusieurs zones

- Avec le Cross Zone Load Balancing, chaque LB distribue son trafic de façon homogène sur toutes les instances enregistrées dans toutes les AZ
- Sans cela, chaque LB distribue les requêtes uniquement sur les instances de son AZ



# Répartition de charge sur plusieurs zones

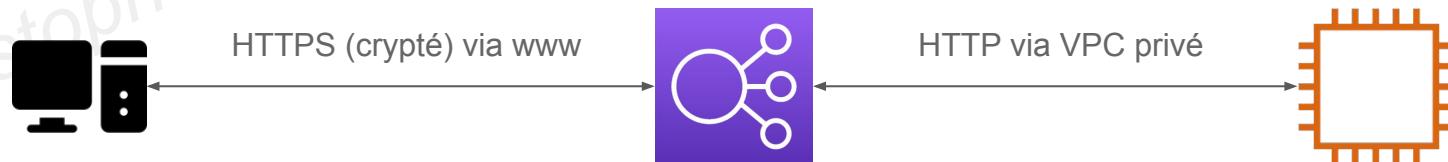
- Classic Load Balancer
  - Désactive par défaut
  - Pas de facturation pour du trafic entre AZ si activé
- Application Load Balancer
  - Toujours activé (ne peut pas être désactive)
  - Pas de facturation pour du trafic entre AZ
- Network Load Balancer
  - Désactivé par défaut
  - Facturation pour du trafic entre AZ si activé

# SSL/TLS - les bases

- Un certificat SSL permet d'encrypter à la volée (in-flight encryption) le trafic entre les clients et le load balancer
- SSL: Secure Sockets Layer, utilisé par les connexions encryptées
- TLS: Transport Layer Security, version plus récente
- On continue d'employer le terme de “certificat SSL” bien qu'il faudrait plutôt parler de certificat TLS
- Des certificats publics sont délivrés par des Certificate Authorities (CA)
  - Comodo, Symantec, GoDaddy, GlobalSign, DigiCert, LetsEncrypt, etc.
- Les certificats SSL ont une date d'expiration et doivent être renouvelés

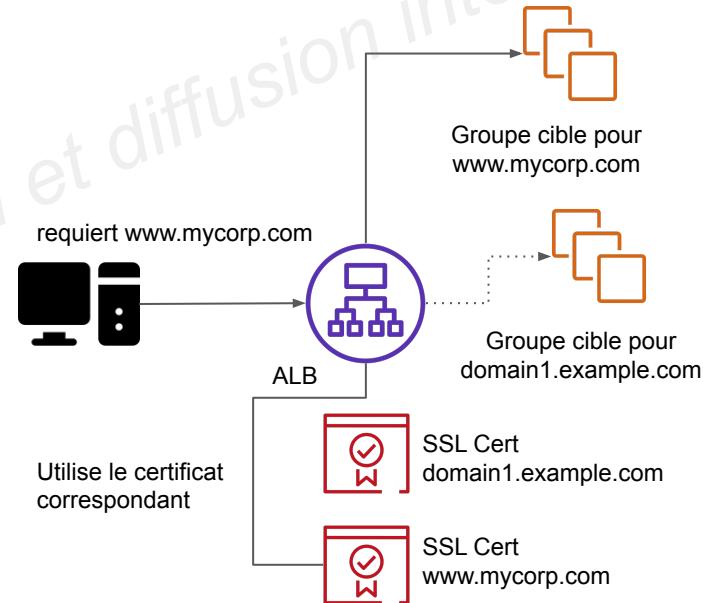
# Load Balancer - Certificat SSL

- Le LB utilise un certificat de norme x.509 (certificat SSL/TLS)
- On peut gérer les certificats via ACM (AWS Certificate Manager)
- On peut créer/charger ses propres certificats
- HTTPS listener
  - nécessité de spécifier un certificat par défaut
  - possibilité d'ajouter une liste de certificats pour supporter de multiples domaines
  - les clients peuvent utiliser **SNI** (Server Name Indication) pour spécifier le hostname à joindre
  - possibilité de spécifier des règles de sécurité pour le support d'anciennes versions SSL/TLS



# SSL - Server Name Indication (SNI)

- SNI résout le problème du chargement de multiples certificats SSL sur un seul serveur web (afin de servir plusieurs applications web)
- C'est un protocole récent obligeant le client à **préciser le hostname** du serveur ciblé dans le “handshake” SSL initial
- Le serveur cherchera ensuite le certificat correspondant ou bien retournera le certificat par défaut
- Valables uniquement pour les ALB et NLB, CloudFront, ne fonctionne pas avec CLB

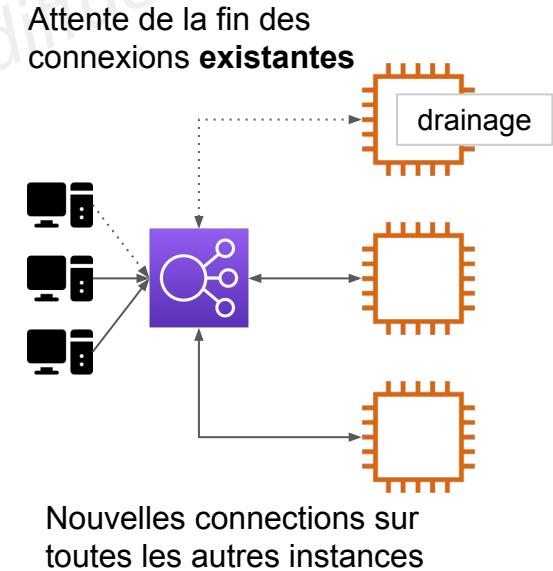


# Elastic Load Balancers - Certificats SSL

- Classic Load Balancer (v1)
  - Supporte un seul certificat
  - L'utilisation de plusieurs certificats nécessitent l'utilisation de plusieurs CLB pour plusieurs hostname
- Application Load Balancer (v2)
  - Supporte plusieurs "listeners" avec plusieurs certificats
  - Utilise SNI à cet effet
- Network Load Balancer (v2)
  - Supporte plusieurs "listeners" avec plusieurs certificats
  - Utilise SNI à cet effet

# ELB - drainage de connexion

- Lexique
  - CLB: drainage de connexion
  - Groupe cible: délai d'annulation d'enregistrement
- Spécifie la durée maximale durant laquelle les requêtes actives portant sur une instance en cours de retrait (annulées, défectueuse) seront traitées
- Cesse d'envoyer de nouvelles requêtes sur une instance en cours de retrait
- Entre 1 à 3600 secondes, 300 par défaut
- Peut être désactivé (valeur à 0)
- Paramétrier avec une faible valeur si les requêtes sont brèves

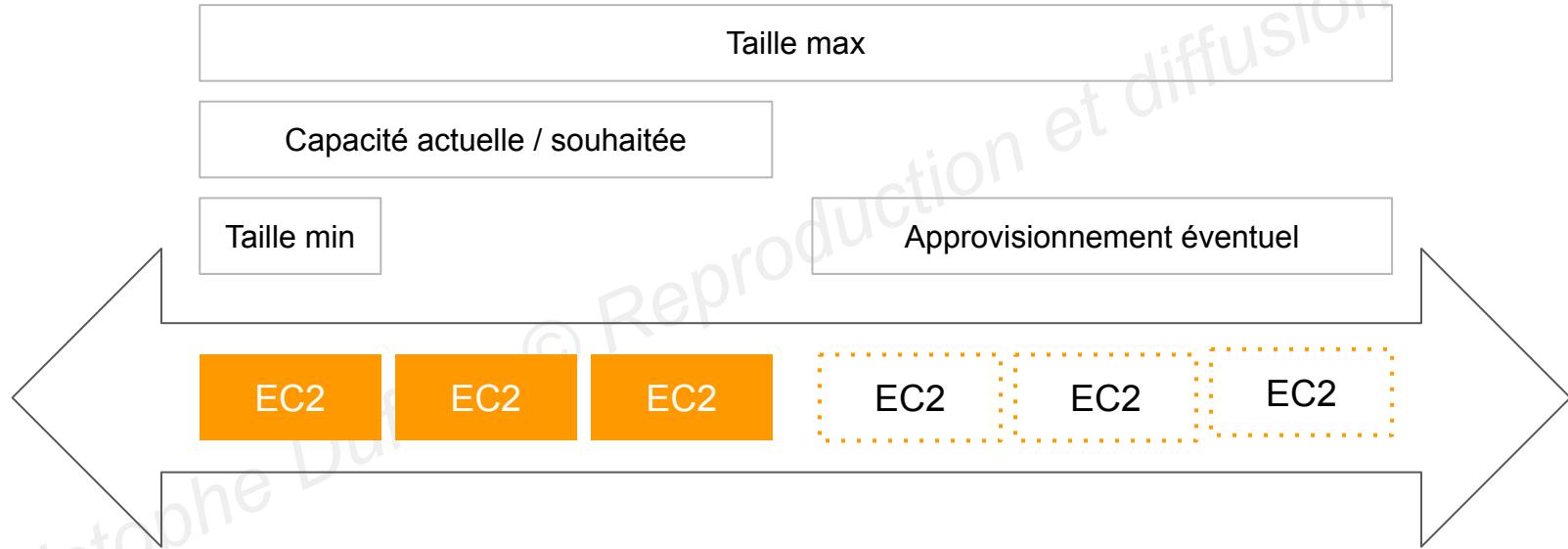


# Auto Scaling Group

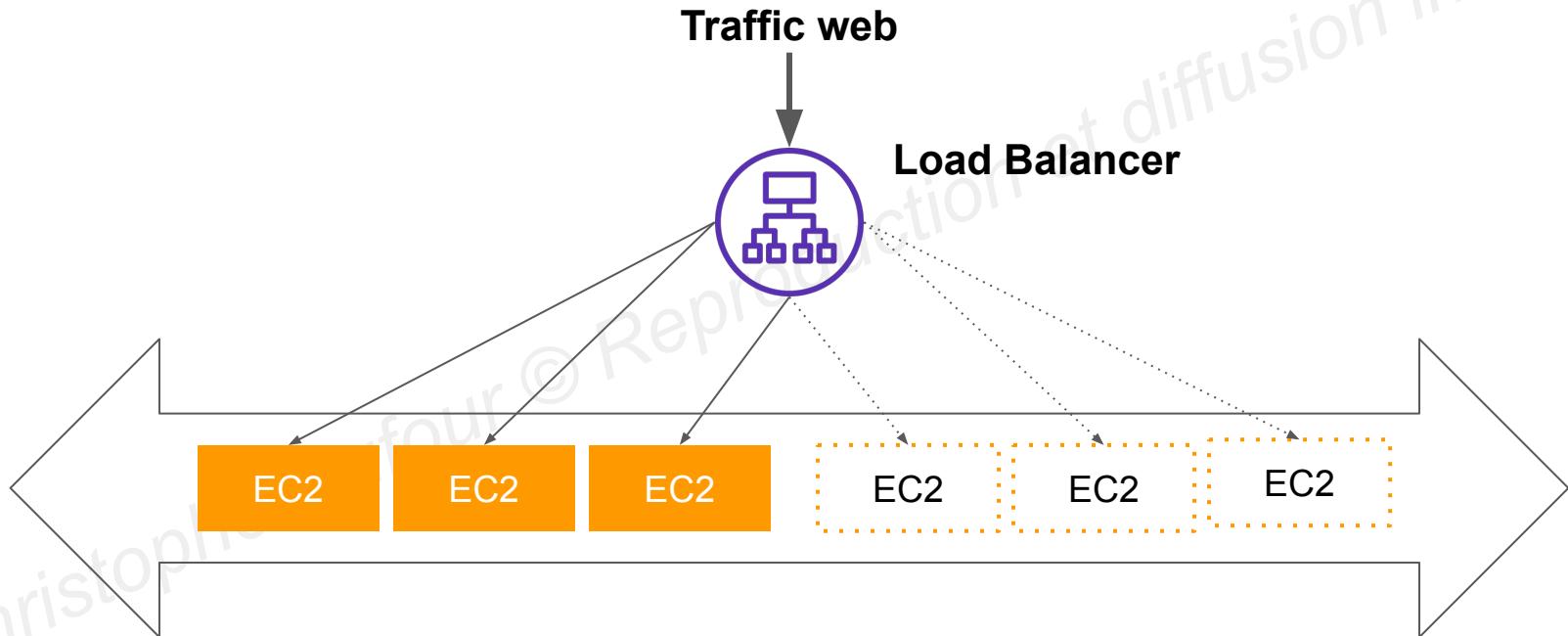


- Concrètement, les charges pesant sur une application (web) peuvent varier
- Le cloud permet de créer/détruire des serveurs très rapidement
- Objectifs d'un Auto Scaling Group (ASG):
  - Augmenter le nombre d'instances EC2 pour s'adapter à la montée en charge
  - Diminuer le nombre d'instances EC2 pour s'adapter à la descente en charge
  - Vérifier que nous disposons d'un minimum et d'un maximum de machines en cours d'exécution
  - Enregistrer automatiquement les nouvelles instances auprès d'un load balancer

# Auto Scaling Group



# Auto Scaling Group avec Load Balancer



# Propriétés d'un Auto Scaling Group

- Configuration de lancement (launch configuration)
  - AMI + type d'instance
  - EC2 User Data
  - Volumes EBS
  - Groupes de sécurité
  - Paire de clés SSH
- Taille min/max/intiale
- Information quant au réseau/sous-réseau
- Information quant au load balancer
- Stratégies de dimensionnement (scaling policies)



Alarme



# Alarmes d'un auto-dimensionnement

- Il est possible de dimensionner un ASG sur la base d'alarmes CloudWatch
- Une alarme surveille une métrique (ex: moyenne CPU)
- Les métriques sont calculées pour l'ensemble des instances du ASG
- En s'appuyant sur l'alarme, on peut créer des:
  - stratégies de dimensionnement à la hausse (scale-out policies)
  - stratégies de dimensionnement à la baisse (scale-in policies)

# Nouvelles règles d'auto-dimensionnement

- Il est maintenant possible de définir de “meilleurs” régimes d’auto-dimensionnement directement gérées par EC2
  - Cibler l’usage moyen CPU
  - Nombre de requêtes par instance ELB
  - Trafic réseau moyen en entrée
  - Trafic réseau moyen en sortie
- Ces règles sont plus faciles et plus cohérentes dans leur mise en place

# Auto-dimensionnement - Métriques personnalisées

- Il est possible de dimensionner à partir d'une métrique personnalisée (ex: nombre d'utilisateurs connectés)
- Recette:
  - a. Envoyer un métrique personnalisée depuis l'application EC2 vers CloudWatch (PutMetric API)
  - b. Créer une alarme CloudWatch pour réagir aux valeurs basses/hautes
  - c. Utiliser l'alarme CloudWatch comme règle de dimensionnement pour l'ASG

# Auto-dimensionnement - à retenir

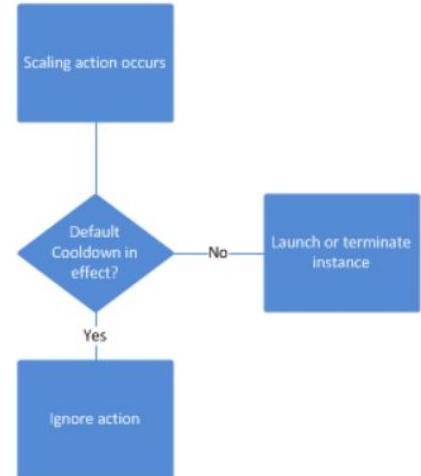
- Les stratégies de dimensionnement peuvent s'appuyer sur le CPU, réseau, etc. mais également sur des métriques personnalisées ou sur un créneau chronologique
- Les ASG utilisent des configurations ou modèles (nouveau) de lancement
- Pour mettre à jour un ASG, il faut fournir un nouveau modèle de lancement
- Les rôles IAM attachés à un ASG seront assignés aux instances EC2
- ASG est gratuit. Les ressources sous-jacentes sont payantes.
- Gérer des instances via un ASG implique que si elles sont détruites par quelque raison que ce soit, l'ASG créera de **nouvelles instances de remplacement**
- Un ASG peut détruire des instances désignées défectueuses par un LB

# Auto-dimensionnement - stratégies

- Target Tracking Scaling
  - Très simple à mettre en place
  - Exemple: je souhaite que l'utilisation moyenne du CPU ne dépasse pas 40%
- Simple/Step Scaling
  - Quand une alarme CloudWatch is déclenchée (ex: CPU > 70%), ajoute 2 unités
  - Quand une alarme CloudWatch is déclenchée (ex: CPU < 30%), supprime 1 unité
- Scheduled Actions
  - Anticiper un dimensionnement sur la base de motifs d'utilisation connus
  - Ex: augmenter la capacité minimale à 10 unités tous les vendredis à 17h

# ASG - Scaling Cooldowns

- Une stabilisation progressive vous permet d'empêcher que votre groupe Auto Scaling lance ou résilie des instances supplémentaires avant que les effets des activités précédentes ne soient visibles.
- Outre la spécification du temps de stabilisation par défaut pour le groupe Auto Scaling, vous pouvez créer des temps de stabilisation qui s'applique à une stratégie de dimensionnement simple spécifique.
- Un temps de stabilisation propre au dimensionnement remplace celui par défaut.
- Un temps de stabilisation spécifique à une mise à l'échelle est souvent utilisé avec une stratégie de diminution en charge. Cette stratégie met fin aux instances, Amazon EC2 Auto Scaling a donc besoin de moins de temps pour déterminer s'il doit mettre fin à des instances supplémentaires. La résiliation d'instances doit être une opération beaucoup plus rapide que le lancement d'instances. Le temps de stabilisation par défaut de 300 secondes est donc trop long. Dans ce cas, un temps de stabilisation spécifique à la mise à l'échelle avec une valeur inférieure de 180 secondes pour votre stratégie de diminution en charge peut vous aider à réduire les coûts en permettant au groupe de diminuer plus rapidement.



<https://docs.aws.amazon.com/autoscaling/ec2/userguide/Cooldown.html>

# Stockage EC2

# Qu'est-ce qu'un volume EBS ?

- Une machine EC2 perd son volume racine (disque principal) quand elle est manuellement détruite
- Des destructions inattendues peuvent parfois se produire (AWS le notifie)
- Parfois, il est nécessaire de stocker les données des instances ailleurs
- Un **volume ELB (Elastic Block Store)** est un **disque réseau** qu'on peut attacher à une instance
- Un ELB permet de persister des données



Amazon EBS

# Volume EBS

- **Disque réseau (pas un disque physique)**
  - utilise le réseau pour communiquer avec l'instance, ce qui implique un peu de latence
  - peut être détaché d'une instance EC2 et attaché à une autre rapidement
- **Il est verrouillé sur une zone de disponibilité**
  - Un volume EBS dans eu-west-2a ne peut pas être attaché à eu-west-2b
  - Pour déplacer un volume d'une zone à l'autre, un snapshot est nécessaire
- **Il est approvisionné en capacité (taille en GB, opérations I/O; etc.)**
  - facturation pour la capacité approvisionnée
  - la capacité peut être modifiée dans le temps

# Types de volume EBS

- 4 types
  - **GP2 (SSD)**: usage généraliste, compromis prix/performances, grande variété d'usages
  - **IO1 (SSD)**: disque hautes performances, usage critique, travaux nécessitant faible latence ou haut débit
  - **ST1 (HD)**: volume HDD faible coût. Conçu pour des accès fréquents, travaux intensifs
  - **SC1 (HDD)**: Le coût le plus faible. Conçu pour des accès moins fréquents.
- Les volumes EBS sont configurables en taille, débit, IOPS
- Seuls GP2 et IOI peuvent être utilisés comme disque de démarrage

# Volume EBS - GP2

- Recommandé pour la plupart des usages
  - Disque de démarrage système
  - Bureaux virtuels
  - Applications interactives à faible latence
  - Environnement de développement et de tests
- 1 GB à 16 TB
- Les petits disques gp2 peuvent intensifier leurs IOPS à 3000
- IOS max: 16000
- 3 IOPS par GB signifie que le maximum est atteint à 5,334GB

# Volume EBS - IO1

- Recommandé pour des usages critiques nécessitant des performances accrues ou plus de 16000 IOPS
- Intenses travaux en base de données, telles que:
  - MongoDB, Cassandra, MySQL, PostgreSQL, Oracle, Microsoft SQL Server
- 4 GB - 16 TB
- Les IOPS sont approvisionnées (PIOPS). Min: 100, Max: 64000 (instance Nitro), 32000 (les autres)
- Le ratio maximal d'IOPS approvisionnés est de 50:1

# Volume EBS - ST1

- Usages
  - Streaming nécessitant un débit élevé et constant à faible prix
  - Big data, entrepôts de données, traitement de log
  - Apache Kafka
- Ne peut pas servir de disque de démarrage
- 500 GB - 16 TB
- IOPS max: 500
- Débit max: 500 MB/s - peut monter en puissance (burst)

# Volume EBS - SC1

- Débit orienté stockage pour de grandes quantités de données à faible fréquence
- A privilégier pour les cas de figure où le prix le plus faible est recherché en priorité
- Ne peut pas servir de dique de démarrage
- 500GB - 16TB
- IOPS max: 250
- Débit max: 250 MB/s - peut monter en puissance (burst)

# Types de volume EBS - résumé

- gp2: généraliste (peu coûteux)
  - 3 IOPS / GB, 100 IOPS min, augmente jusqu'à 3000 IOPS, 16000 IOPS max
  - 1 GB - 16 TB ; +1TB = +3000 IOPS
- io1: IOPS approvisionnées (coûteux)
  - 100 IOPS min, 64000 IOPS max (Nitro) ou 32000 (autres)
  - 4 GB - 16 TB ; taille du disque et IOPS sont indépendants
- st1: HDD à débit optimisé
  - 500 - 16 TB, 500 MB/s en débit
- sc1: HDD froid, données accédées peu fréquemment
  - 500 - 16 GB, 250 MB/s en débit

# EBS vs stockage d'instance

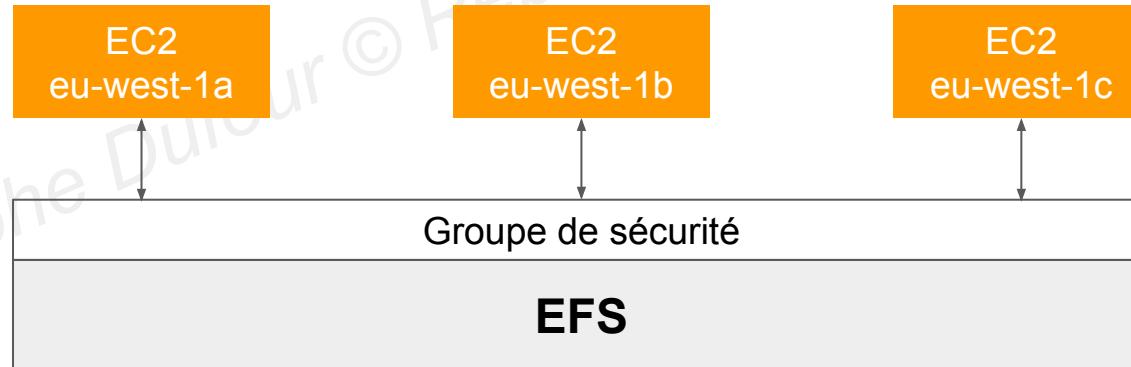
- Certaines instances n'ont pas de volume racine EBS
- Elles ont à la place un “Instance Store” qui est stockage temporaire
- Le stockage d'instance est physiquement attaché à la machine
- Avantages:
  - Performances I/O supérieures
  - Bon pour mémoire tampon, cache, contenu temporaire
  - Les données survivent au redémarrages
- Inconvénients:
  - Les données sont perdues à l'arrêt ou la destruction
  - Ne peut pas être redimensionné
  - Les sauvegardes sont de la responsabilité de l'utilisateur

# Stockage d'instance EC2

- Disque physique attaché au serveur physique où se situe l'instance EC2
- IOPS très élevées
- Taille jusqu'à 7,5 TB
- Ne peut pas croître en taille
- Risque de perte de données en cas de panne matérielle

# EFS - Elastic File System

- NFS (Network File System, système de fichiers réseau) géré pouvant être monté sur plusieurs instances EC2
- EFS fonctionne en multi-AZ
- Haute disponibilité, dimensionnable, onéreux (3x gp2), facturation à l'usage



# EFS - Elastic File System

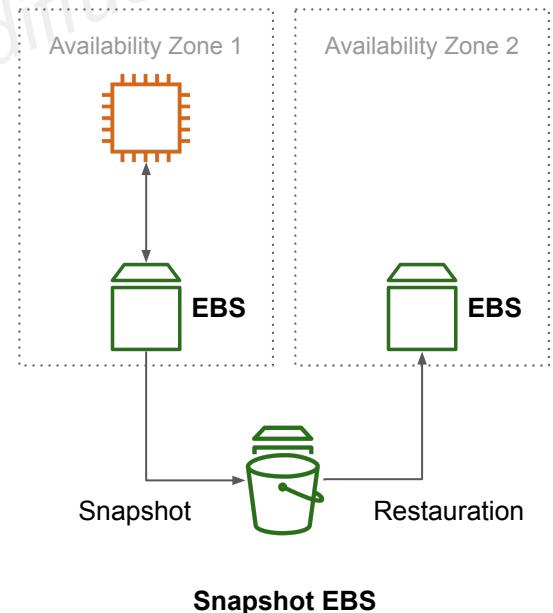
- Cas d'utilisation: gestion de contenu, service web, partage de données, Wordpress
- Utilise le protocole NFSv4.1
- Utilise un groupe de sécurité pour le contrôle d'accès
- **Compatible avec les images AMI basées sur Linux (pas Windows)**
- Chiffrement de données au repos via KMS
- Système de fichier POSIX disposant d'une API standardisée
- Dimensionnement automatique, facturation à l'usage, pas de plannification

# EFS - Performance et classes de stockage

- EFS Scale
  - Milliers de clients NFS, +10GB de débit
  - Grossit à l'échelle du Petabyte automatiquement
- Performance mode (paramètrer au moment de la création)
  - Usage généraliste (par défaut), cas d'utilisation avec sensibilité à la latence (serveurs web, CMS, etc...)
  - I/O max, latency, débit, traitement parallèles optimisés (big data, traitement media)
- Storage Tiers (fonctionnalité des gestion de cycle de vie)
  - Standard: fichiers accédés fréquemment
  - Infrequent access (EFS-IA): coût de récupération, la solution meilleure marché pour le stockage

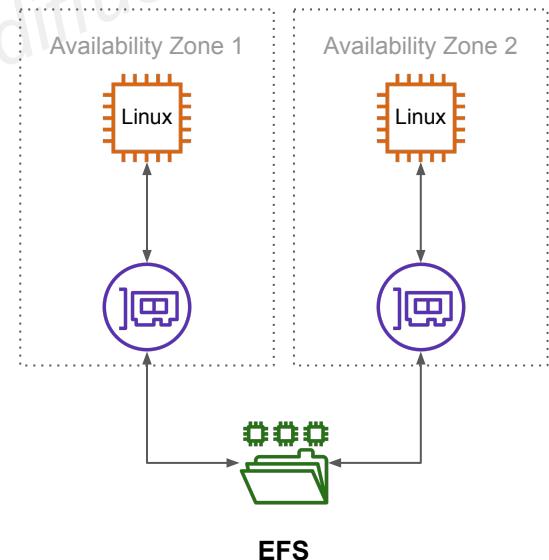
# EBS vs EFS - Elastic Block Storage

- Disque EBS
  - ne peut être attaché qu'à une seul instance à la fois
  - vérrouillé sur une une AZ
  - gp2: IO grandit si la taille du disque grandit
  - io1: IO grandit indépendamment de la taille disque
- Pour migrer un disque EBS entre deux AZ
  - Faire un snapshot
  - Restaurer le snapshot dans l'autre AZ
  - Les sauvegardes consomment des IO ; mieux vaut les éviter en période de forte activité de l'application
- Un disque EBS racine d'une instance est détruit par défaut (désactivable) quand l'instance est détruite



# EBS vs EFS - Elastic File System

- Montable sur des centaines d'instances en multi AZ
  - Partage de fichiers d'un site web (Wordpress)
  - Seulement sur les instances Linux (POSIX)
  - Prix plus élevé que EBS
  - Peut se servir de EFS-IA pour faire des économies
- 
- Rappel: EFS vs EBS vs Instance Store



# Les fondamentaux AWS - Partie III

RDS, Aurora et ElastiCache



# AWS RDS - vue d'ensemble

- RDS: Relational Database Service
- Service géré de base de données relationnelles utilisant le langage SQL pour les requêtes
- Permet de créer dans le cloud des bases de données maintenues par AWS
  - Posgres
  - MySQL
  - MariaDB
  - Oracle
  - Microsoft SQL Server
  - Aurora (propriété d'AWS)

# Avantages par rapport à un déploiement EC2

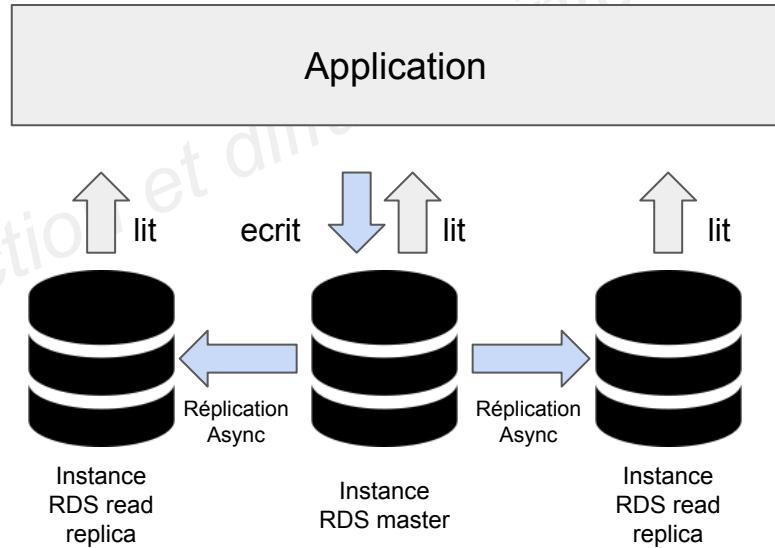
- RDS est un service géré
  - Approvisionnement automatique, correctif OS
  - Sauvegardes, restauration à un timestamp précis
  - Monitoring
  - Read replicas pour améliorer les performances
  - Multi AZ (Disaster Recovery)
  - Capacités de scaling (vertical et horizontal)
  - Stockage sur EBS (gp2 ou io 1)
- MAIS pas de connection via SSH

# Sauvegardes RDS

- Automatiquement activées dans RDS
- Sauvegardes automatiques
  - Sauvegarde complète journalière
  - Logs de transaction sauvegardés toutes les 5 min
    - Possibilité de restaurer à un tout moment dans le temps ( $> 5$  min)
  - Rétention de 7 jours (augmentable à 35)
- DB Snapshots
  - Déclenchés manuellement par l'utilisateur
  - Durée de rétention au choix

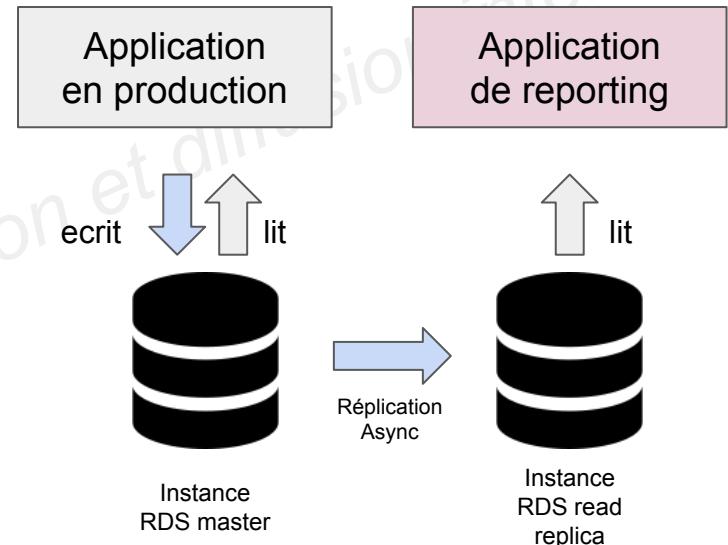
# Replicas de lecture RDS

- Jusqu'à 5 Read Replicas
- Dans une seule AZ, multi AZ ou multi régions
- La réPLICATION est ASYNC
- Les réPLICAS peuvent recevoir leur propre DB (master)
- Les applications doivent mettre à jour leur chaîne de connexion pour profiter des replicas de lecture



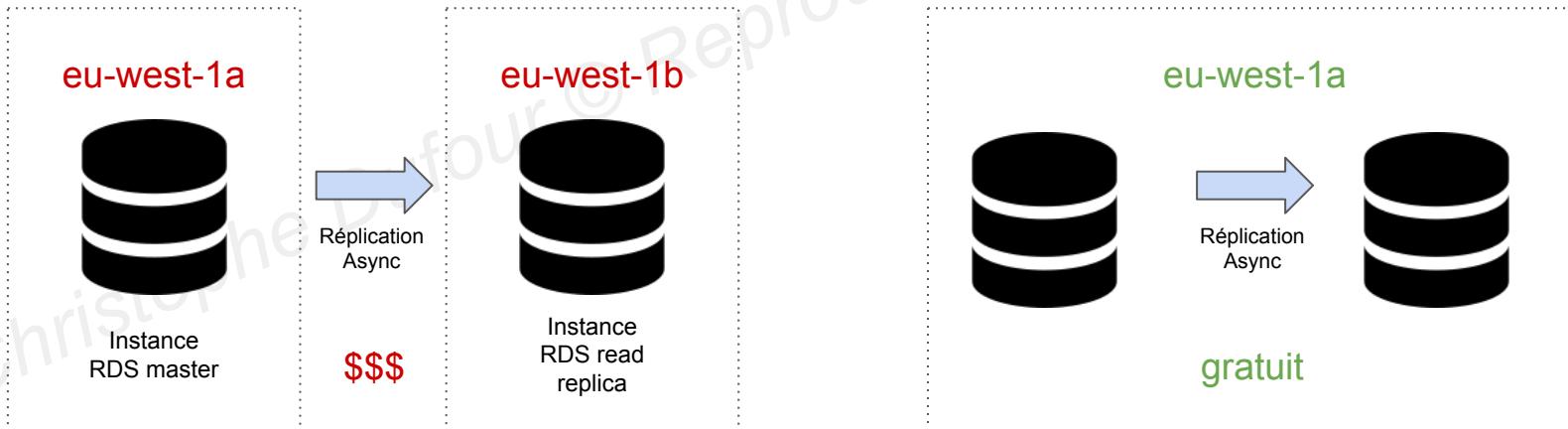
# Replicas de lecture RDS - Cas d'utilisation

- Nous avons une base de données en production recevant une charge normale
- Nous voulons faire du reporting sur cette base
- Création d'un Read Replica pour cette nouvelle charge de travail
- L'application en production n'est pas affectée
- Les Read Replicas sont utilisés pour tout type d'opération SELECT (Et non INSERT, UPDATE, DELETE)



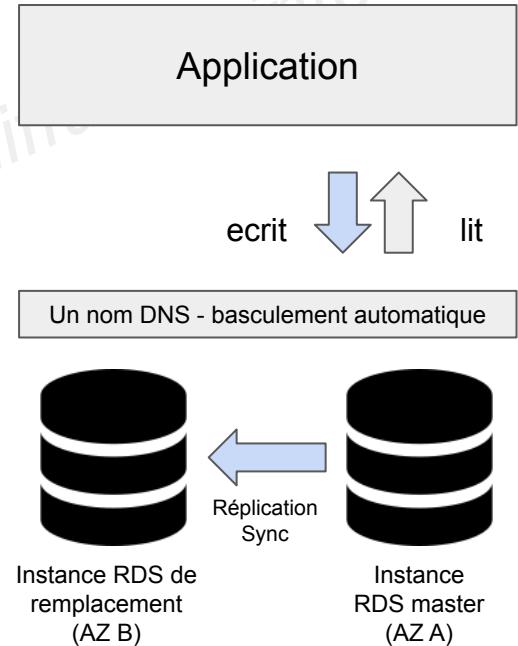
# Replicas de lecture RDS - Coûts réseau

- Dans AWS, il y a des coûts réseau dès lors que des données vont d'une AZ à une autre
- Pour reduire ces coûts, on peut avoir des Read Replicas dans la même AZ



# RDS Multi AZ (Disaster Recovery)

- RéPLICATION SYNC
- Un nom DNS, basculement automatique vers une instance de remplacement
- Augmente la disponibilité
- Basculement en cas de perte d'une AZ, d'un réseau, d'une instance ou d'un échec de stockage
- Pas d'intervention manuelle dans les applications
- Ne s'utilise pas pour le dimensionnement



# Sécurité RDS - Chiffrement des données

- Chiffrement au repos (at rest)
  - possibilité de chiffrer le master et les read replicas avec AWS KMS - AES-256 encryption
  - le chiffrement doit être défini au lancement
  - Si le master n'est pas chiffré, les read replicas ne peuvent pas être chiffrés
  - Transparent Data Encryption (TDE) disponible pour Oracle et SQL Server
- Chiffrement à la volée
  - Certificat SSL pour chiffrer les données vers RDS
  - Fournir les options SSL du certificat de confiance lors de la connection à la db
  - Pour forcer SSL
    - Postgres: rds.forceçssl=1 dans la console AWS RDS (Parameter Groups)
    - MySQL: GRANT USAGE ON \*.\* TO 'mysqluser'@'%' REQUIRE SSL;

# RDS - opérations de chiffrement

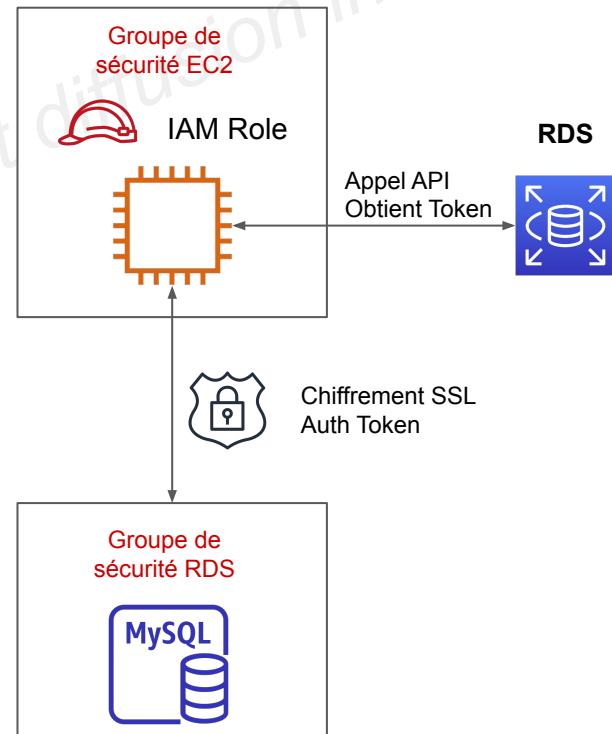
- Chiffrer des sauvegardes RDS
  - Les snapshots de bases RDS non chiffrées sont non chiffrés
  - Les snapshots de bases RDS chiffrées sont chiffrés
  - Possibilité de copier un snapshot dans une version chiffrée
- Pour chiffrer une base RDS non chiffrée
  - Faire un snapshot d'une base non chiffrée
  - Copier le snapshot et activer le chiffrement
  - Restaurer la base de données depuis le snapshot chiffré
  - Migrer les applications vers la nouvelle base de données et supprimer l'ancienne

# Sécurité RDS - Réseau et IAM

- Sécurité réseau
  - Les bases de données sont généralement déployées dans un sous-réseau privé
  - La sécurité RDS s'appuie sur les groupes de sécurité (même concept que pour EC2), ils contrôlent quel IP / groupe de sécurité peut communiquer avec RDS
- Gestion d'accès
  - Les règles IAM aident à contrôler qui peut gérer AWS RDS (via RDS API)
  - Les traditionnels nom d'utilisateur / mot de passe peuvent être utilisés pour se connecter
  - L'authentification IAM peut être utilisée pour se connecter à RDS MySQL et PostgreSQL

# RDS - Authentification IAM

- Fonctionne avec MySQL et PostgreSQL
- Pas besoin de mot de passe, juste un token d'identification obtenu via un appel à l'API IAM et RDS
- Le token d'authentification a une durée de vie de 15 minutes
- Avantages
  - Les entrées/sorties réseau peuvent être chiffrées via SSL
  - Gestion centralisée au niveau d'IAM plutôt que de la base
  - Permet d'exploiter les rôles IAM et les profiles d'instance EC2 pour une meilleure intégration



# Sécurité RDS - résumé

- Chiffrement au repos
  - effectué uniquement à la création de l'instance de base de données
  - sinon: db non chiffrée => snapshot => copie encryptée => création à partir de la copie
- Responsabilité utilisateur
  - vérification des IP/ports/règles en entrée des SG du SG de la base
  - création d'utilisateurs, au niveau de la base ou bien via IAM
  - création d'une base avec ou sans un accès public
  - vérification que les parameter groups ou la base est configurée pour n'accepter que des connections SSL
- Responsabilité AWS
  - Accès SSH
  - Correctifs appliqués à la base et à l'instance
  - Surveillance de l'instance sous-jacente

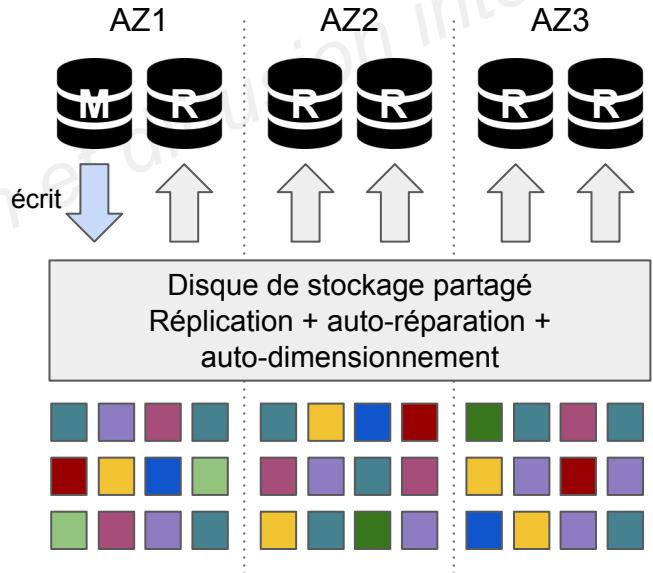
# Amazon Aurora



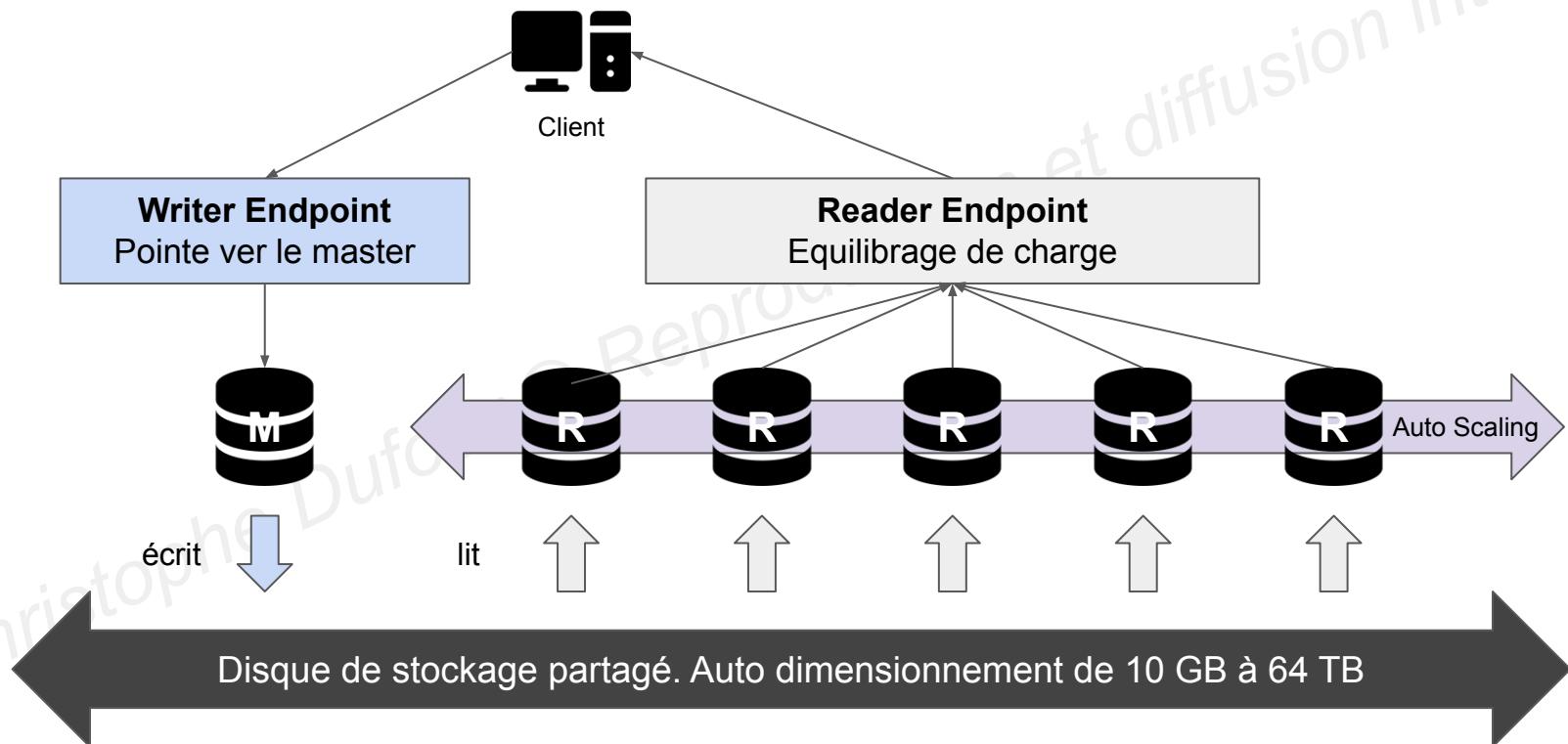
- Aurora est une technologie propriétaire développée par Amazon
- Postgres et MySQL sont tous deux supportés par Aurora (même drivers)
- Aurora est optimisé pour le cloud AWS et annonce des performances 5 fois supérieures à RDS MySQL 3 fois supérieures à RDS Postgres
- Le stockage augmente automatiquement jusqu'à 64TB par paliers de 10 GB
- Aurora peut avoir 15 replicas (contre 5 pour RDS) et la procédure de réPLICATION est plus rapide (latence sous les 10 ms)
- Le basculement (failover) est instantané, il est nativement en haute dispo.
- Aurora est 20% plus coûteux que RDS mais plus efficace

# Aurora - haute disponibilité

- 6 copies des données à travers 3 AZ
  - 4 copies sur 6 pour l'écriture
  - 2 copies sur 6 pour la lecture
  - auto réparation avec réPLICATION peer-to-peer
  - le stockage est segmenté à travers des centaines de volumes
- Une instance Aurora (master) reçoit l'écriture
- Basculement automatisé pour le master en moins de 30 s.
- Master + jusqu'à 15 Read Replicas
- Support pour le réPLICATIONS Multi régions



# Aurora DB Cluster



# Fonctionnalités d'Aurora

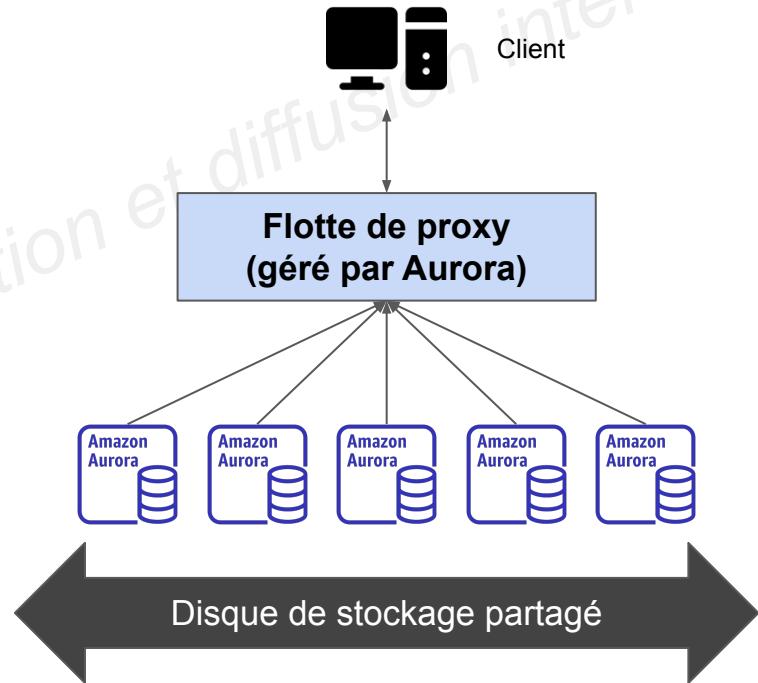
- Basculement automatique
- Sauvegarde et restauration
- Isolement et sécurité
- Conformité aux standards “industriels”
- Dimensionnement automatique
- Correctifs automatisés sans temps d'arrêt
- Monitoring avancé
- Routine de maintenance
- Retour en arrière: restauration de données à n'importe quel moment sans utilisation de sauvegardes

# Aurora - sécurité

- Similaire à RDS
- Chiffrement au repos utilisant KMS
- Sauvegardes automatisés, snapshots et replicas chiffrés
- Chiffrement à la volée via SSL
- Possible authentification par token IAM (méthode identique à RDS)
- Utilisateur à la charge de protéger son instance avec les groupes de sécurité
- Pas de SSH

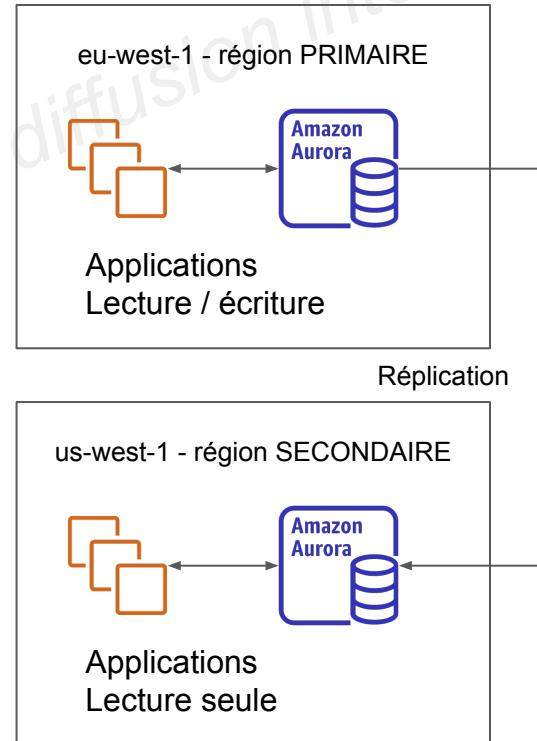
# Aurora Serverless

- Instanciation de base de données automatisée et auto-dimensionnement basé sur l'utilisation
- Bonne solution pour des charges de travail peu fréquentes, intermittentes ou imprévisibles
- Pas de planification en capacité requise
- Facturation à la seconde, tarification peut être plus efficace



# Global Aurora

- Réplicas de lecture Multi Region
  - Utile pour la récupération après désastre
  - Simple à mettre en place
- Aurora Global Database (recommandé)
  - 1 région primaire (écriture/lecture)
  - Jusqu'à 5 régions secondaires (lecture seule), temps de réplication < 1 s.
  - Jusqu'à 16 réplicas de lecture par région secondaire
  - Aide à réduire la latence
  - Promouvoir une autre région a une durée maximale d'interruption (RTO) < 1 min.



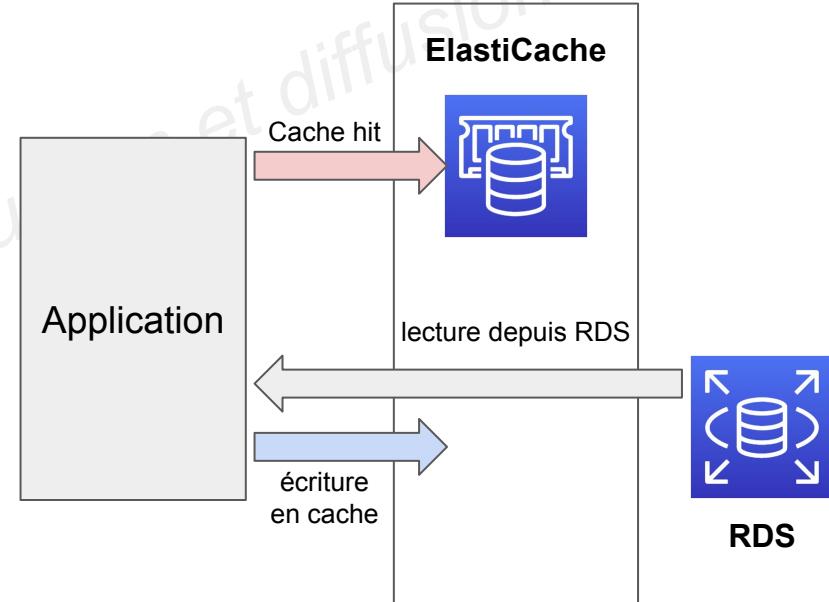


# Amazon ElastiCache

- De la même manière que RDS permet d'obtenir des bases de données relationnelles gérées, ElastiCache permet d'obtenir Redis ou Memcached gérés
- Les caches sont des base de données en mémoire offrant de hautes performances et temps de latence très faible
- Permet de soulager les bases de données des charges de lecture intensives
- Aide à rendre l'application “stateless”
- AWS gère maintenance/correctifs/optimisations/configuration/surveillance/sauvegarde/restauration

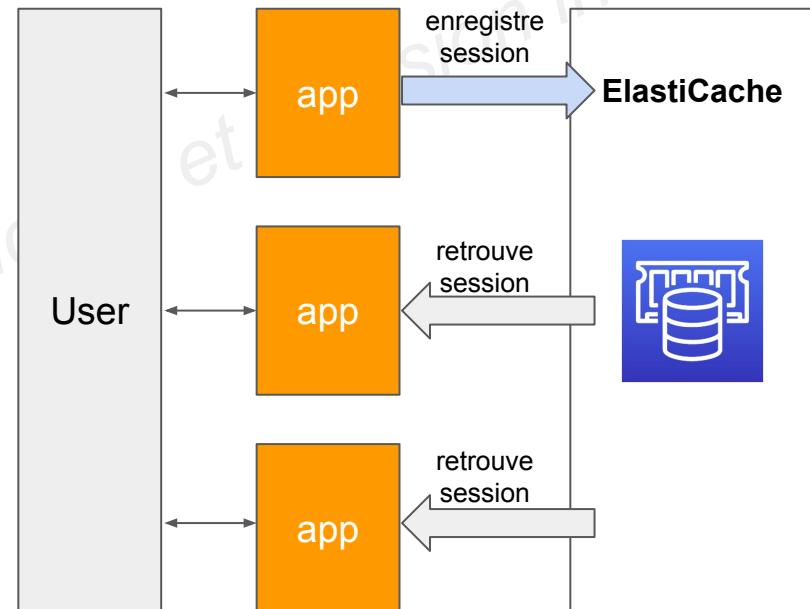
# Amazon ElastiCache - exemple d'architecture

- Exemple DB Cache
- L'application interroge ElastiCache, si les données n'y sont pas, elles sont récupérées dans RDS puis stockées dans ElastiCache
- Permet de soulager RDS
- Le cache doit obéir à une stratégie d'invalidation



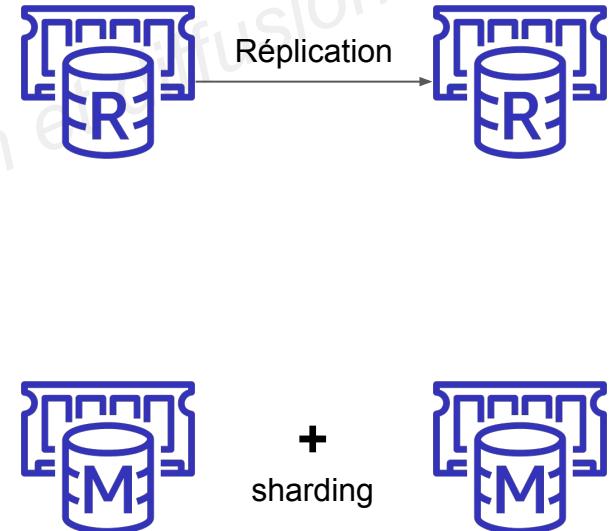
# Amazon ElastiCache - exemple d'architecture

- Exemple: session utilisateur
- L'utilisateur se connecte à l'application
- L'application écrit dans ElastiCache les données de session
- L'utilisateur effectue une requête sur une autre instance de l'application
- L'instance retrouve les données de session



# ElastiCache - Redis vs Memcached

- REDIS
  - Multi AZ avec basculement automatique
  - Réplicas de lecture pour dimensionnement et haute disponibilité
  - Durabilité des données avec persistence AOF
  - Fonctionnalités de sauvegarde et restauration
- MEMCACHED
  - Partitionnement des données sur de multiples noeuds (sharding)
  - Pas de persistance
  - Pas de sauvegarde ni restauration
  - Architecture multi-thread

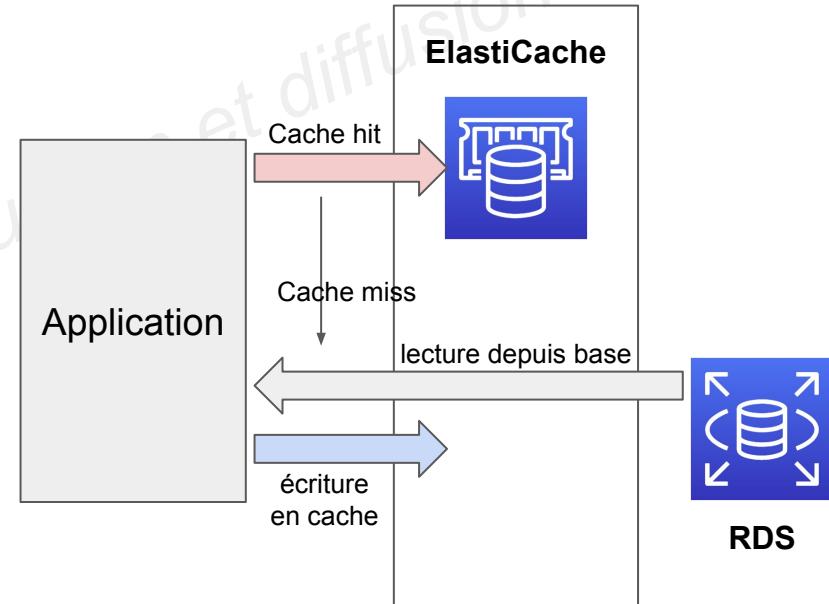


# Mise en cache - considérations

- Il est-il fiable de mettre des données en cache ?
  - Les données peuvent ne pas être à jour, être hétérogènes
- Est-ce efficace pour les données ?
  - Exemples: données changement rarement, quelques clés sont fréquemment demandées
  - Contre-exemples: données changeant rapidement, nombreuses clés fréquemment requises
- Les données sont-elles bien structurer pour de la mise en cache ?
  - Exemple: mise en cache clé/valeur ou mise en cache d'aggrégations de résultats
- Quel modèle de mise en cache est le plus adéquat ?
- Lire: <https://aws.amazon.com/fr/caching/best-practices/>

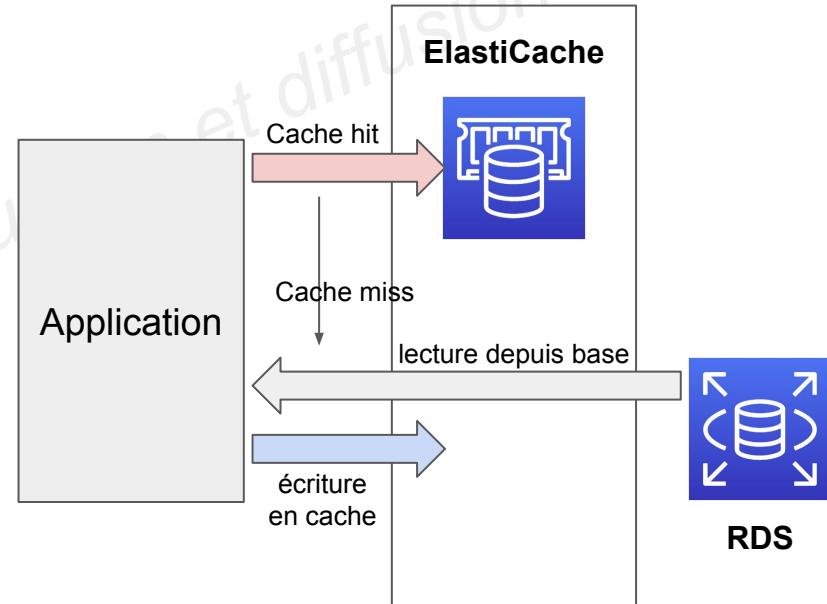
# Lazy Loading / Cache-Aside / Lazy Population

- Avantages:
  - Seules des données demandées sont mises en cache (le cache n'est pas rempli avec des données inutilisées)
  - L'échec d'un noeud n'est pas fatal (cela augmente uniquement la latence de réactivation)
- Inconvénients
  - Pénalité d'absence en cache (Cache miss) conduisant à des requêtes additionnelles
  - Données périmées: les données peuvent être plus récentes en base que dans le cache



# Lazy Loading / Cache-Aside / Lazy Population

- Avantages:
  - Seules des données demandées sont mises en cache (le cache n'est pas rempli avec des données inutilisées)
  - L'échec d'un noeud n'est pas fatal (cela augmente uniquement la latence de réactivation)
- Inconvénients
  - Pénalité d'absence en cache (Cache miss) conduisant à des requêtes additionnelles
  - Données périmées: les données peuvent être plus récentes en base que dans le cache



# Lazy Loading / Cache-Aside / Lazy Population

## Pseudocode Python

```
def get_user(user_id):
    # Check the cache
    record = cache.get(user_id)

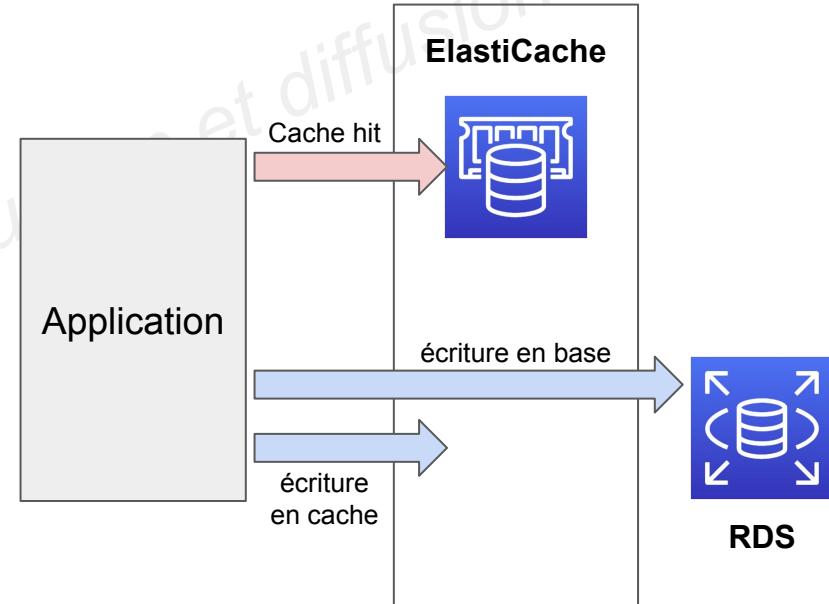
    if record is None:
        # Run a DB query
        record = db.query("select * from users where id = ?", user_id)
        # Populate the cache
        cache.set(user_id, record)
    return record

else:
    return record

# App code
user = get_user(10)
```

# Write Through

- Consiste à ajouter au / modifier le cache quand la base est mise à jour
- Avantages:
  - Les données en cache ne sont jamais périmées, **lectures rapides**
  - **Pénalité en écriture** vs pénalité en lecture (chaque écriture requiert deux appels)
- Inconvénients
  - Données absentes jusqu'à ce qu'elles soient ajoutées/modifiées en base.  
Atténuable par implémentation d'une stratégie de Lazy Loading également
  - “Cache churn”: de nombreuses données ne seront jamais lues



# Write-Through

## Pseudocode Python

```
def save_user(user_id, values):
    # Save to DB
    record = db.query("update users ... where id = ?", user_id, values)

    # Push into cache
    cache.set(user_id, record)

    return record

# App code
user = save_user(10, {"name": "Paolo Dybala"})
```

# Evictions du cache et Time-to-live (TTL)

- Les évictions du cache peuvent se produire de trois manières:
  - Suppression explicite d'un élément du cache
  - Elément évincé car la mémoire est saturée et qu'il n'est pas été récemment utilisé (LRU)
  - Un TTL a été spécifié sur l'élément
- TTL est utilisé pour tout type de données
  - Classements
  - Commentaires
  - FLux d'activité
- TTL paramétrable de quelques secondes à plusieurs heures/jours
- Si trop d'évictions de produisent en raison d'une saturation mémoire, il faut songer à redimensionner

# Derniers mots de sagesse

- Lazy Loading / Cache aside est facile à implémenter et fonctionne pour de nombreuses situations
- Write-Through est généralement associée au Lazy Loading ciblant les requêtes ou travaux tirant parti de cette optimisation
- Utiliser le TTL est souvent une bonne idée, sauf quand le Write-Through est utilisé. Choisir des valeurs en rapport avec la nature de l'application
- Ne mettre en cache que les données naturellement candidates (profiles utilisateurs, blogs, etc...)
- Citation: “*Il y a deux choses difficiles en informatique: l'invalidation du cache et le nommage*”

# Route 53

# Amazon S3 Introduction

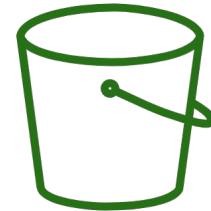
# S3 - Introduction



- Simple Storage Service
- Amazon S3 est l'un des piliers d'AWS
- Se présente comme un stockage à dimensionnement infini
- Très populaire
- De nombreux sites internet utilisent S3 comme colonne vertébrale
- De nombreux services AWS utilisent S3 en interne

# S3 - Buckets

- Amazon S3 permet de stocker des objets (fichiers) dans des “buckets” (dossiers)
- Un bucket doit avoir un **nom global unique**
- Un bucket est défini au niveau de la région
- Conventions de nommage
  - pas de majuscule
  - pas d'underscore
  - longueur de 3 à 63
  - pas d'IP
  - doit commencer avec une lettre ou un chiffre



# S3 - Objets

- Les objets (fichiers) ont une Clé
- La **clé** est le chemin **COMPLET**:
  - s3://my-bucket/my\_file.txt
  - s3::://my-bucket/my\_folder1/another\_folder/my\_file.txt
- Clé = **préfixe** + **nom de l'objet**
  - s3://my-bucket/my\_folder/another\_folder/my\_file.txt
- Pas de concept de “dossier” dans les buckets bien que l’UI semble indiquer le contraire
- Il n'y a que des clés (longs noms avec slashes '/')



# S3 - Objets - suite

- La valeur d'un objet correspond au corps de la requête
  - Taille max: 5TB
  - Pour un envoi de plus de 5GB, il faut utiliser le "multi-part"
- Méta-données: liste de paires clé/valeur
- Tags: paire clé/valeur unicode, jusqu'à 10, utile pour la sécurité, cycle de vie
- ID de version, si le versioning est activé



# S3 - gestion de versions

- Il est possible de gérer des versions de fichier dans S3
- Fonctionnalité activable au **niveau du bucket**
- L'écrasement d'une même clé provoquera une incrémentation de l'id de version (1,2,3,...)
- C'est une bonne pratique d'activer la gestion de version
  - protection contre les suppressions involontaires (possibilité de restaurer)
  - facile retour à une version antérieure
- Notes:
  - Un fichier non versionné avant l'activation de la gestion de version aura comme id de version “null”
  - Suspendre la gestion de version ne supprime pas les versions antérieures

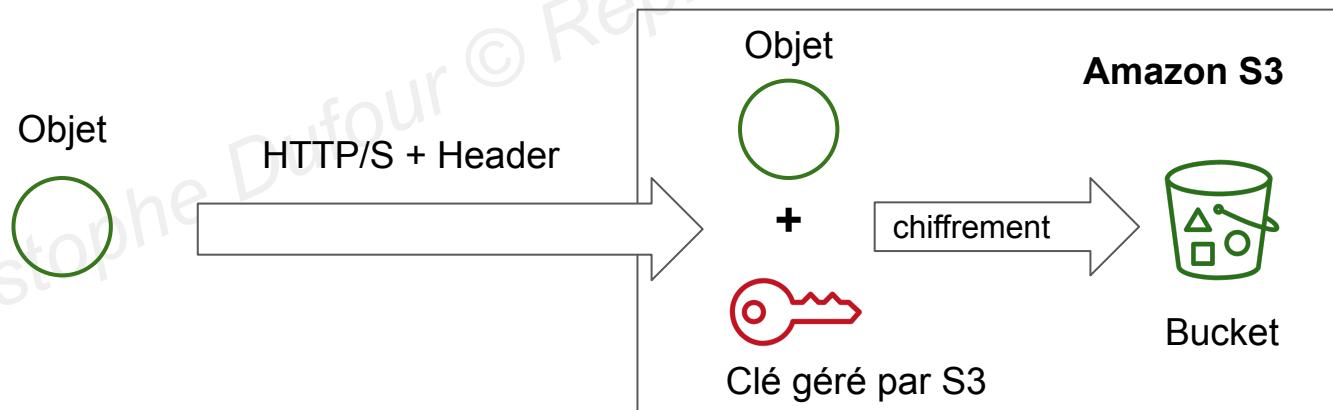


# Chiffrement des objets S3

- 4 façons de chiffrer les objets S3
  - SSE-S3: chiffre les objets S3 avec des clés gérées par AWS
  - SSE-KMS: se sert de AWS Key Management Service pour gérer les clés de chiffrement
  - SSE-C: permet à l'utilisateur de gérer ses propres clés
  - Client Side Encryption
- Il est important de comprendre quelle méthode est la plus adaptée à telle ou telle situation (notamment lors de l'exam de certification)

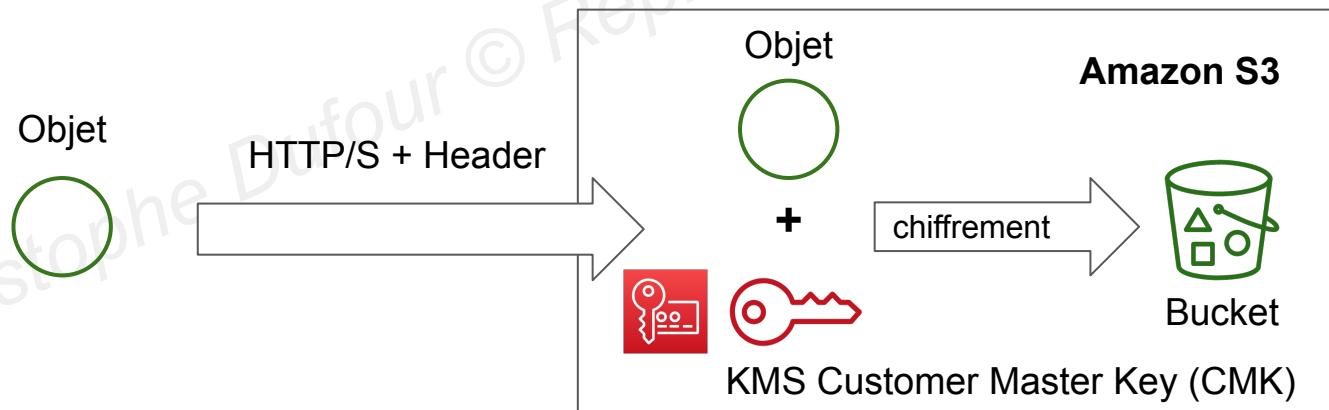
# SSE-S3

- SSE-S3: chiffrement utilisant des clés gérées par Amazon S3
- Les objets sont chiffrés côté serveur
- Chiffrement AES-256
- Entête http à utiliser: “x-amz-server-side-encryption”:”AES256”



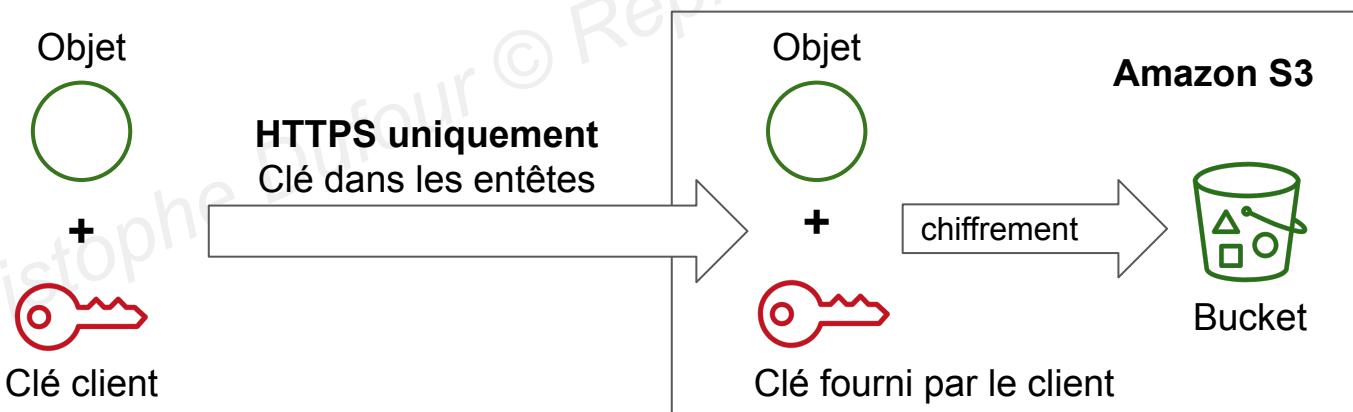
# SSE-KMS

- SSE-KMS: chiffrement utilisant des clés gérées par KMS
- Avantages KMS: plus de contrôle utilisateur + fonctionnalités d'audit
- Objet chiffré côté serveur
- Entête http à utiliser: “x-amz-server-side-encryption”:”aws:kms”



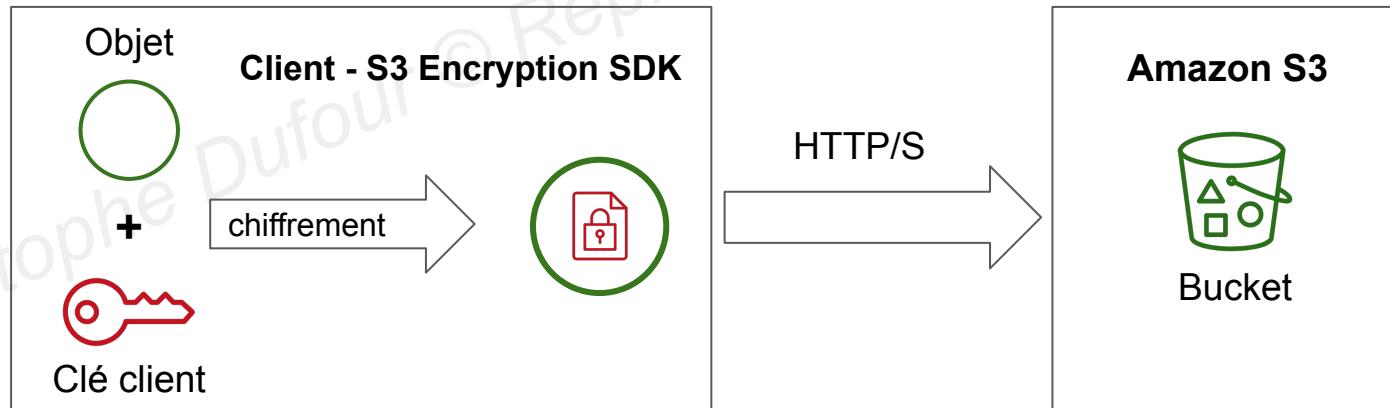
# SSE-C

- SSE-KMS: chiffrement côté serveur utilisant des clés gérées par l'utilisateur en-dehors d'AWS
- Amazon S3 ne stocke pas la clé de chiffrement fourni par l'utilisateur
- **HTTPS doit être utilisé**
- La clé de chiffrement doit être incluse dans les entêtes http, pour chaque requête



# Client Side Encryption

- Le client doit chiffrer lui-même les données avant l'envoi à S3
- Le client doit déchiffrer lui-même les données reçues par S3
- Il existe des outils de chiffrement côté client tels que Amazon S3 Encryption Client
- Le client gère intégralement les clés et le cycle de chiffrement





# Chiffrement en transit (SSL/TLS)

- Amazon S3 expose
  - des points finaux (endpoints) HTTP, non chiffrés
  - des points finaux HTTPS, chiffrés à la volée
- L'utilisateur est libre d'utiliser les endpoints de son choix mais HTTPS est recommandé
- De nombreux clients utilisent les endpoints HTTPS par défaut
- HTTPS est obligatoire pour SSE-C
- Le chiffrement à la volée est également appelé SSL/TLS

# Sécurité S3

- Basée sur l'utilisateur
  - Stratégies IAM: quels appels API sont autorisés pour un utilisateur IAM précis
- Basée sur la ressource
  - Bucket policies: règles de bucket réalisables dans la console, transversales aux comptes
  - Object Access Control List (ACL): plus précis
  - Bucket Access Control List (ACL): moins commun
- Note: un utilisateur IAM principal peut accéder à un objet S3 si
  - les permissions d'utilisateur IAM l'autorisent OU que les règles de ressource l'autorisent
  - ET qu'il n'y ait aucun REFUS (DENY) explicite en place

# Règles des buckets S3

- Règles JSON
  - Resources: buckets et objets
  - Actions: ensemble d'API pour autoriser ou refuser
  - Effect: Allow/Deny
  - Principal: le compte ou l'utilisateur auquel appliquée les règles
- Utiliser les règles de bucket pour:
  - autoriser un accès public
  - forcer des objets à être chiffré au chargement
  - autoriser l'accès à un autre compte (cross account)

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PublicRead",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket/*"  
            ]  
        }  
    ]  
}
```

# Blocage de l'accès public à un bucket

- Par défaut, le blocage d'accès public est activé pour:
  - nouveaux buckets
  - nouveaux points d'accès
  - nouveaux objets
- Les utilisateurs peuvent débloquer l'accès public pour chacun de ces éléments
- Ces paramètres ont été créés pour prévenir les fuites de données entreprise
- Si l'on a la certitude que le bucket ne devra jamais être public, ces paramètres ne doivent pas être modifiés
- Peut être défini au niveau du compte

# Sécurité S3 - suite

- Réseau
  - support de points finaux VPC (pour instances dans un VPC, sans internet)
- Loggin et audit
  - les logs d'accès S3 peuvent être stockés dans un autre bucket
  - les appels API peuvent être loggués dans AWS CloudTrail
- Sécurité utilisateur
  - MFA Delete: l'authentification multi-factures peut-être requise pour les buckets versionnés à supprimer
  - URLs pré-signées: URLs valides pour une période de temps limité

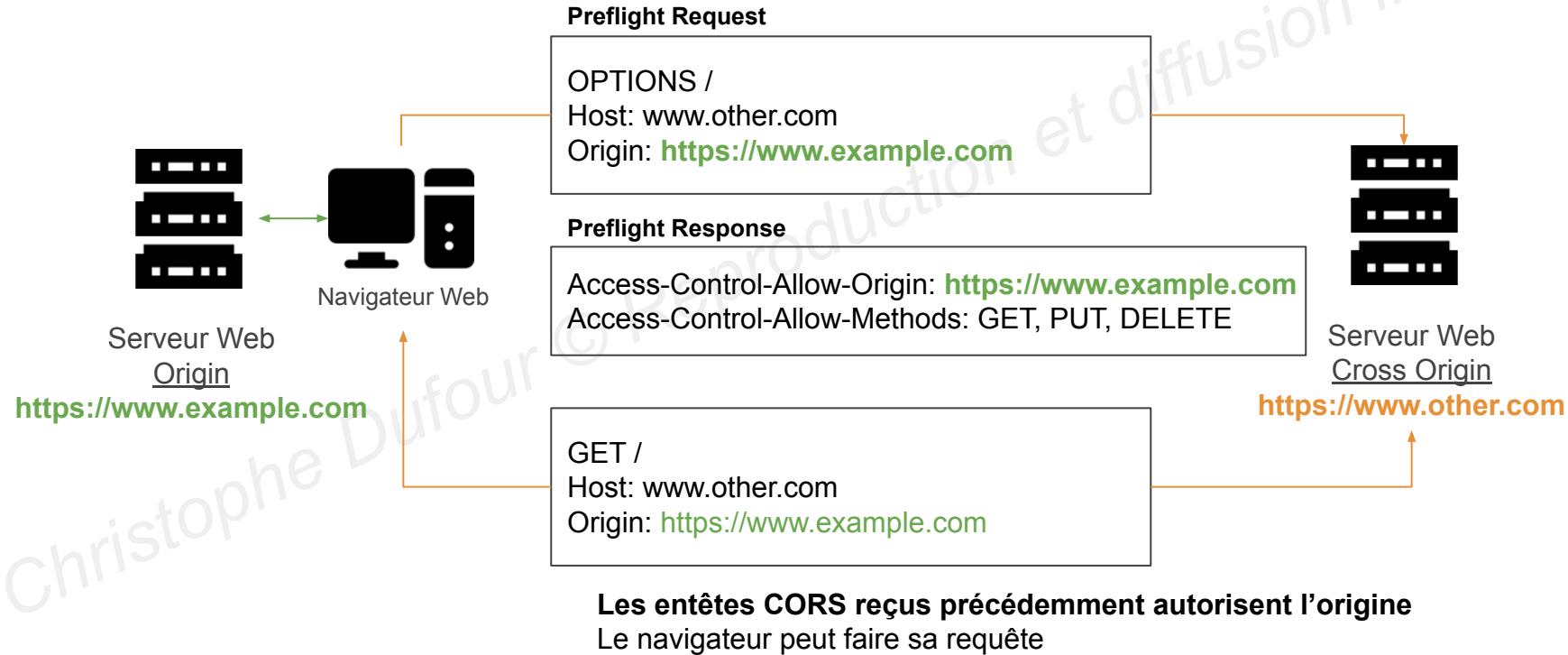
# Sites internet S3

- S3 peut héberger des sites web statiques et les rendre publiquement accessibles
- L'URL du site sera:
  - <bucket-name>.s3-website-<AWS-region>.amazonaws.com
- En cas d'erreur 403 (Forbidden), vérifier que les règles de bucket acceptent l'accès public

# CORS expliqué

- Cross-Origin Resource Sharing
- Une origine: schème (protocole) + host (domaine) + port
- Un navigateur web permet des requêtes sur d'autres origines que celle visitée
- Même origine
  - <http://example.com/app1>
  - <http://example.com/app2>
- Origines différentes
  - <http://example.com>
  - <http://other.example.com>
- Un navigateur refusera certaines réponses, si elles ne contiennent les entêtes CORS (ex: Access-Control-Allow-Origin)

# CORS - schéma



# S3 CORS

- Si un client fait une requête cross-origin sur un bucket S3, il faut activer les entêtes CORS appropriés
- Question fréquente à l'examen de certification
- Possibilité d'autoriser une origine précise ou toutes (\*)



# S3 - Modèle de consistence

- **Read after write consistency** pour des nouveau objets PUTS
  - Dès qu'un nouvel objet est créé, on peut le retrouve (PUT 200 => GET 200)
  - Toujours vrai, sauf si un GET est été effectué peu avant (GET 404 => PUT 200 => GET 404). Eventually consistent
- **Eventual consistency** pour les modifications et suppresions d'objets existants
  - La lecture d'un objet immédiatement après modification peut renvoyer une ancienne version (PUT 200 => PUT 200 => GET 200 (peut-être ancienne version))
  - Un objet supprimer peut être encore accessible un bref instant (DELETE 200 => GET 200)
- Note: aucune possibilité de réclamer une “strong consistency”

# Développer sur AWS

CLI, SDK and stratégies IAM

# Développer sur AWS - introduction

- Effectuer des tâches sur AWS peut se faire de plusieurs manières, en utilisant:
  - AWS CLI sur une machine locale
  - AWS CLI sur une machine EC2
  - AWS SDK sur une machine locale
  - AWS SDK sur une machine EC2
  - AWS Instance Metadata service pour EC2
- Dans cette section, nous apprendrons
  - comment réaliser toutes ces opérations
  - la manière la plus sécurisée, en accord avec les bonnes pratiques

# Paramètrage de AWS CLI v2 sur Linux

- Paramètrons correctement AWS CLI sur Linux

# Problème à l'installation du CLI

- En cas d'erreur `aws: command not found` après installation d'AWS CLI
  - le chemin de l'exécutable `aws` n'a pas été trouvé par le système, il faut l'ajouter dans la variable d'environnement `PATH`
- `PATH` permet à l'OS de savoir où sont situés certains exécutables sans avoir à saisir leur chemin absolu dans l'invite de commandes (terminal)

# Configuration d'AWS CLI

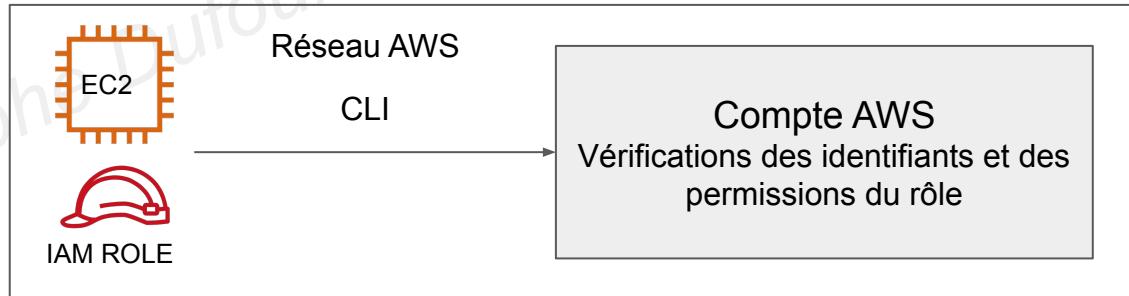
- Configurons correctement le CLI
- Nous apprendrons à obtenir des identifiants d'accès (access credentials) et à les protéger
- Ne surtout pas partager vos:
  - AWS Access Key
  - Secret Key

# AWS CLI sur EC2 - mauvaise approche

- Nous pourrions exécuter “aws configure” sur EC2 comme nous venons de le faire (et cela fonctionnerait)
  - Mais...cela est très DANGEREUX en termes de sécurité
- Ne JAMAIS mettre ses identifiant dans une instance EC2
- Vos identifiants **personnels** sont **personnels** et n'appartiennent qu'à votre ordinateur **personnel**
- Si l'instance EC2 est compromise, votre compte le sera aussi
- Si l'instance EC2 est partagée, d'autres personnes peuvent effectuer des actions AWS sous vos identifants
- Dans le cas d'EC2, il existe une meilleure façon de procéder: AWS IAM Roles

# AWS CLI sur EC2 - la bonne approche

- Des rôles IAM peuvent être attachés à une instance EC2
- Les rôles IAM peuvent être accompagnés d'une stratégie définissant précisément ce que l'instance EC2 est autoriser à faire
- Les instances EC2 peuvent ensuite utiliser ces profils automatiquement sans autre configuration additionnelle
- C'est la meilleure pratique et AWS la recommande à 100%



# AWS CLI Dry Runs

- Parfois, nous voulons seulement savoir si nous disposons des permissions, sans nécessairement exécuter de commandes réelles
- Certaines commandes AWS CLI (comme celles du service EC2) peuvent être onéreuses si elles réussissent
- Certaines commandes AWS CLI (pas toutes) disposent d'une option `--dry-run` pour simuler l'appel API
- Nous pouvons le comparer à un "tir à blanc", sans conséquence
- Faisons le test !

# AWS CLI STS Decode Errors

- Quand nous exécutons un appel API et qu'il échoie, nous pouvons obtenir un long message d'erreur
- Ce message peut être décodé en utilisant la commande STS
  - **sts decode-authorization-message**
- Faisons le test !

# AWS EC2 Instance Metadata

- AWS ECS Instance Medata est puissant mais assez méconnu des développeurs
- Cette fonctionnalité permet aux instances EC2 dans “savoir sur elles-mêmes”  
**sans avoir à utiliser de rôle IAM à ce effet**
- URL: <http://169.254.169.254/latest/meta-data>
- On peut retrouver le rôle IAM à partir des méta-données mais pas la stratégie IAM
- Metadata: informations sur l'instance EC2
- Userdata: script exécutable au premier démarrage de l'instance
- Faisons le test !

# AWS CLI et MFA

- Pour utiliser une authentification multi-facteurs avec le CLI, il faut créer une session temporaire
- Pour ce faire, exécuter l'appel API **STS GetSessionToken**
- **aws sts get-session-token** --serial-number arn-of-the-mfa-device --token-code code-from-tolen --duration-seconds 3600

```
{  
    "Credentials": {  
        "SecretAccessKey": "secret-access-key",  
        "SessionToken": "temporary-session-token",  
        "Expiration": "expiration-date-time",  
        "AccessKeyId": "access-key-id"  
    }  
}
```

# AWS SDK - vue d'ensemble

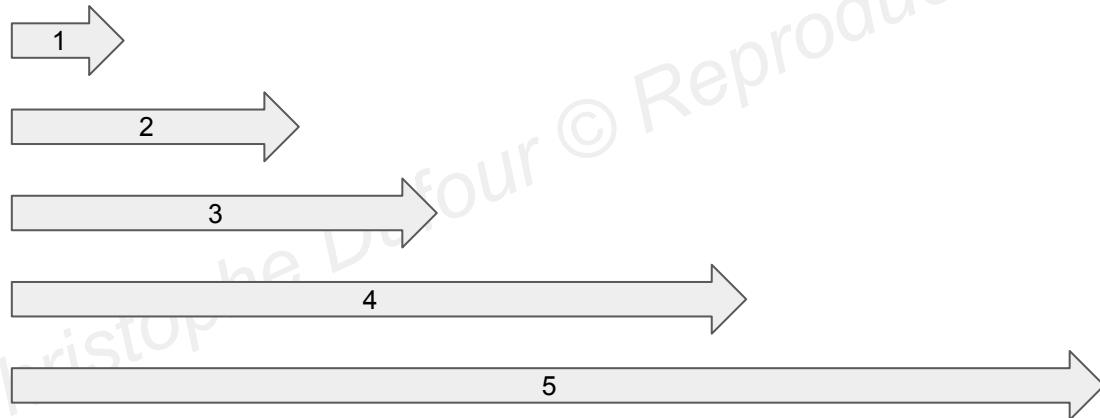
- Parfois, il est utile/nécessaire d'effectuer des actions AWS directement depuis le code applicatif (sans passer par le CLI)
- Dans ce cas de figure, on peut recourir au SDK (Software Development Kit)
- Les SDKs sont liés à des environnement de développement/langages
  - Java
  - .NET
  - Nodejs
  - PHP
  - Python (boto3)
  - Go
  - Ruby
  - C++

# AWS SDK - vue d'ensemble

- Il faut passer par AWS SDK quand le code doit utiliser des services AWS tels que DynamoDB
- Fait intéressant: AWS CLI repose sur le SDK Python (boto3)
- Pour l'examen de certification, il est nécessaire de savoir quand il est opportun d'utiliser un SDK
- Bon à savoir: si aucune région n'est spécifiée ou configurée, us-east-1 est choisie par défaut

# Exponential Backoff

- En cas de **ThrottlingException** intermittentes, utiliser exponential backoff
- Mécanisme de réitération d'appels inclu dans l'API SDK
- Doit être implémenter par le développeur si l'API est utilisée telle quelle



# AWS CLI - Chaîne de fournisseurs d'identifiants

Le CLI recherche les identifiants dans cet ordre

1. **Options de ligne de commande** --region, --output, et --profile
2. **Variables d'environnement** AWS\_ACCESS\_KEY\_ID,  
AWS\_SECRET\_ACCESS\_KEY et AWS\_SESSION\_TOKEN
3. **Fichier d'identifiants CLI** --aws configure ~/.aws/credentials sur Linux/Mac  
& C:\Users\user\.aws\credentials sur Windows
4. **Fichier de configuration CLI** --aws configure ~/.aws/config sur Linux/Mac  
& c:\Users\user\.aws\config sur Windows
5. **Identifiants de conteneur** pour les tâches ECS
6. **Identifiants de profil d'instance** pour les profils d'instance EC2

# AWS SDK - Chaîne de fournisseurs d'identifiants par défaut

Le SDK Java (par exemple) cherchera les identifiants dans cet ordre

1. **Variables d'environnement** AWS\_ACCESS\_KEY\_ID et AWS\_SECRET\_ACCESS\_KEY
2. **Propriétés système Java** -- aws/accessKeyId et aws.secretKey
3. **Le fichier de profils d'identifiants par défaut** -- ex: ~/.aws/credentials, partagé par de nombreux SDK
4. **Identifiants Amazon ECS container** pour les conteneurs ECS
5. **Identifiants de profil d'instance** pour les instances EC2

# Scénario d'utilisation d'identifiants AWS

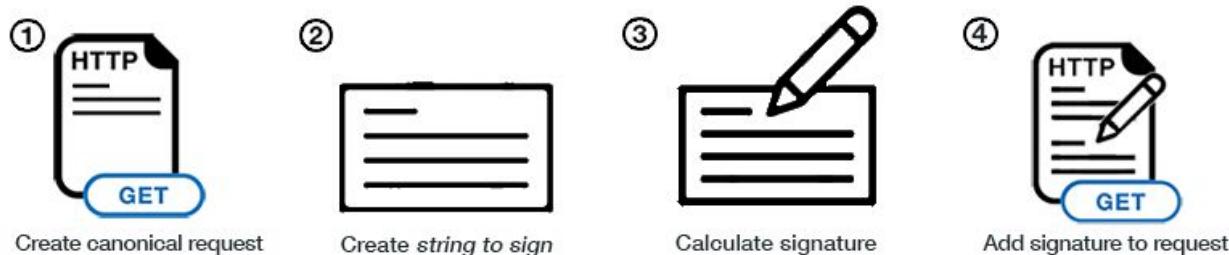
- Une application déployée sur une instance EC2 utilise des variables d'environnement avec des identifiants provenant d'un utilisateur IAM pour faire des appels à l'API S3
- L'utilisateur IAM a des permissions S3FullAccess
- L'application utilise uniquement un bucket S3, donc selon les meilleures pratiques:
  - Un rôle IAM et un profil d'instance EC2 a été créé pour l'instance EC2
  - Le rôle a été assigné avec le minimum de permissions pour accéder à ce bucket S3 précis
- Le profil d'instance a été assigné à l'instance EC2 mais l'application continue d'avoir accès à tous les buckets S3. Pourquoi ?
  - => le chaîne d'identifiants donne priorité aux variables d'environnement

# Identifiants AWS - meilleures pratiques

- Avant tout, ne JAMAIS JAMAIS inclire les identifiants dans le code
- La meilleure pratique consister à obtenir les identifiants par héritage de la chaîne d'identifiants
- A l'intérieur d'AWS, utiliser des rôles IAM
  - Rôles d'instance EC2 pour les instances EC2
  - Rôles ECS pour les tâches ECS
  - Rôles Lambda pour les fonctions Lambda
- A l'extérieur d'AWS, utiliser des variables d'environnement / profils nommés

# Signer des requêtes API AWS

- Quand on appelle l'API HTTP AWS, on signe les requêtes pour être identifié par Amazon, grâce aux identifiants AWS (access key et secret key)
- Note: certaines requêtes vers S3 n'ont pas besoin d'être signées
- Lorsqu'on utilise le SDK ou le CLI, les requêtes HTTP sont automatiquement signées
- Une requête HTTP AWS doit être signée via AWS Signature v4 (SigV4)



# Exemple d'une requête SigV4

- Entête HTTP

```
Authorization: AWS4-HMAC-SHA256  
Credential=AKIAIOSFODNN7EXAMPLE/20130524/us-east-1/s3/aws4_request,  
SignedHeaders=host;range;x-amz-date,  
Signature=fe5f80f77d5fa3beca038a248ff027d0445342fe2855ddc963176630326f1024
```

- Query String (ex: URLs pré-signée S3)

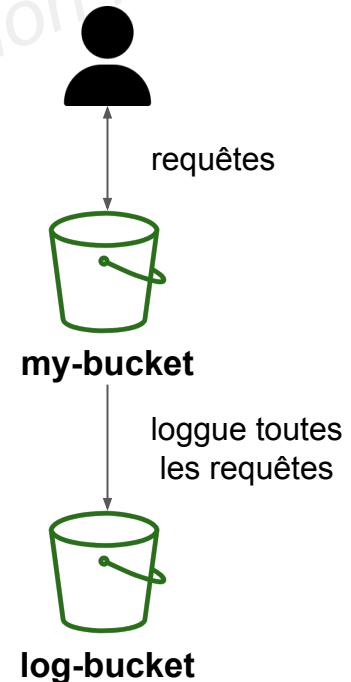
```
https://s3.amazonaws.com/examplebucket/test.txt  
?X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=<your-access-key-id>/20130721/us-east-1/s3/aws4_request  
&X-Amz-Date=20130721T201207Z  
&X-Amz-Expires=86400  
&X-Amz-SignedHeaders=host  
&X-Amz-Signature=<signature-value>
```

# S3 - suppression par MFA

- MFA force l'utilisateur à obtenir un code supplémentaire depuis un appareil (généralement un téléphone mobile)
- Pour utiliser MFA-Delete, il faut activer la gestion de version sur le bucket S3
- Il faudra un MFA pour
  - supprimer définitivement une version d'un objet
  - suspendre le versionning d'un bucket
- Pas besoin de MFA pour
  - activer la gestion de version
  - lister les versions supprimées
- Seul le propriétaire du bucket (compte racine) peut activer/désactiver le MFA-Delete
- Actuellement le MFA-Delete peut uniquement être activé via le CLI

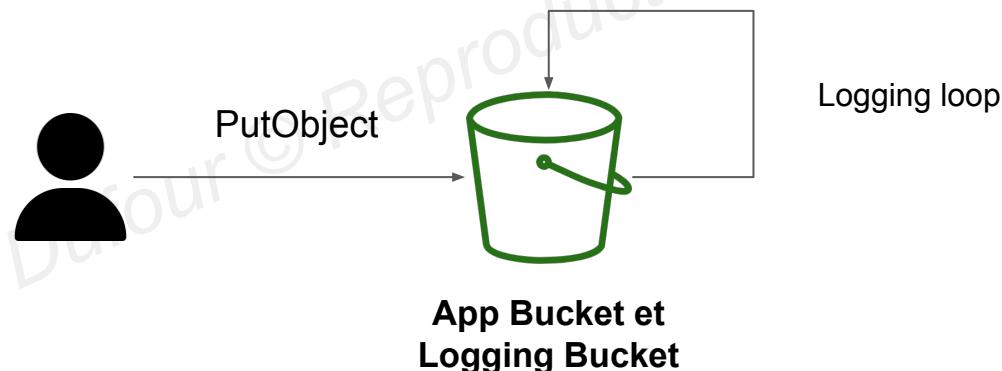
# Logs d'accès S3

- Pour des raisons d'audit, il peut être avantageux de logguer tous accès aux buckets S3
- Tout requête vers S3, depuis n'importe quel compte, autorisée ou refusée sera logguée dans un autre bucket
- Ces données peuvent être traitées
  - par des outils d'analyze...
  - ...par Amazon Athena (comme on le verra plus tard)
- Voir le format du log:  
<https://docs.aws.amazon.com/AmazonS3/latest/userguide/LogFormat.html>



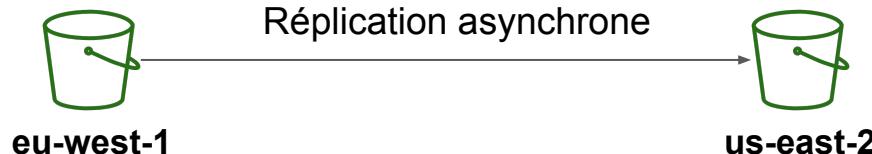
# Logs d'accès S3 : mise en garde

- Ne pas paramétrer le bucket de logging en tant que bucket surveillé
- Cela créera un boucle de logging et la taille du bucket grandira de façon exponentielle



# RéPLICATION S3 (CRR et SRR)

- Nécessité d'activer la gestion de version pour la source et la destination
- CRR: Cross Region Replication
- SRR: Same Region Replication
- Les buckets peuvent être dans des comptes différents
- La copie est asynchrone
- Les bonnes permissions IAM doivent être données à S3
- Cas d'utilisation CRR: conformité, diminution de latence
- Cas d'utilisation SRR: agrégation de log, réPLICATION en directe entre comptes de test et de production



# RéPLICATION S3 - notes

- Après activation, seuls les nouveaux objets sont répliqués (pas rétroactif)
- Pour les opérations de suppression
  - Si supprimé sans ID de version, marqueur “delete” ajouté, pas répliqué
  - Si supprimé avec ID de version, objet source supprimé, pas répliqué
- Il n'y a pas de “chaînage” de la réPLICATION
  - si bucket 1 est répliqué en bucket 2 qui est lui-même répliqué en bucket 3, les objets créés en bucket 1 ne sont pas répliqués enbucket 3

# S3 - URLs pré-signées

- Possibilité de générer des URLs pré-signées par CLI ou SDK
  - pour download: facile, on peut utiliser le CLI
  - pour upload: plus difficile, il faut utiliser le SDK
- Validité de 3600 secondes par défaut, l'échéance peut être modifiée avec --expires-in[TIME\_BY\_SECONDS]
- L'utilisateur à qui est fourni l'URL pré-signée hérite des permissions de la personne ayant généré l'URL pour GET/PUT
- Exemples:
  - Autoriser uniquement des utilisateurs loggués à télécharger une vidéo dans votre bucket S3
  - Autorier une liste variable d'utilisateurs à télécharger des fichiers en générant des URLs dynamiques
  - Autoriser temporairement un utilisateur à uploader un fichier dans un emplacement précis de notre bucket

# Les classes de stockage S3 (Storage Classes)

- Amazon S3 Standard - General Purpose
- Amazon S3 Standard - Infrequent Access (IA)
- Amazon S3 One Zone - Infrequent Access
- Amazon S3 Intelligent Tiering
- Amazon Glacier
- Amazon Glacier Deep Archive

# S3 Standard - General Purpose

- Haute durabilité (99.99999999%) des objets à travers plusieurs AZ
- Si l'on stocke 10000000 d'objets dans Amazon S3, on peut s'attendre en moyenne à perdre un seul d'entre eux tous les 10000 ans
- 99.99% de disponibilité sur un année donnée
- Résiste à deux défaillances concurrentes
- Cas d'utilisaton: Big Data, applications mobile et de jeu, distribution de contenu

# S3 Standard - Infrequent Access (IA)

- Adapté aux données moins fréquemment accédées mais nécessitant un accès rapide quand il le faut
- Haute durabilité des objets à travers plusieurs AZs
- 99.9% de disponibilité
- Coût plus faible que Amazon S3 Standard
- Résiste à deux défaillances concurrentes
- Cas d'utilisation: dépôt de données pour récupération, sauvegardes

# S3 One Zone - Infrequent Access (IA)

- Pareil que IA mais données stockées dans une seul AZ
- Haute durabilité des objets dans une seule AZ, données perdues si l'AZ est détruite
- 99.5% de disponibilité
- Faible latency et performances haut débit
- Supporte SSL pour les données en transit et chiffrement au repos
- Coût inférieur à IA d'environ 20%
- Cas d'utilisation: stockage de sauvegardes secondaires ou de données pouvant être récrées

# S3 Intelligent Tiering

- Latence et performances haut débit comparables à celles de S3 Standard
- Léger coût mensuel de monitoring et “auto-tiering”
- Déplace automatiquement les objets entre des “access tiers” sur la base de modèles d'utilisation
- Haute durabilité à travers plusieurs AZ
- Résistant aux événements impactant une AZ entière
- 99.9% de disponibilité pour une année donnée

# Amazon Glacier

- Stockage d'objets à bas coût destinés à l'archivage / sauvergarde
- Données stockés à long terme (dizaines d'années)
- Alternative au stockage sur site
- Durabilité moyenne annuelle de 99.99999999%
- Coût de stockage mensuel (\$0.004 / GB) + coût de récupération
- Chaque élément dans Glacier est appelée "**Archive**" (jusqu'à 40TB)
- Les archives sont stockées dans des "**Vaults**" (coffres)

# Amazon Glacier et Glacier Deep Archive

- Amazon Glacier
  - 3 options de récupération
    - Expedited (1 à 5 minutes)
    - Standard (3 à 5 heures)
    - Bulk (5 à 12 heures)
  - durée minimale de stockage: 90 jours
- Amazon Deep Glacier
  - pour du stockage à long terme
  - moins coûteux
  - options de récupération: Standard (12 heures), Bulk (48 heures)
  - durée minimale de stockage: 180 jours

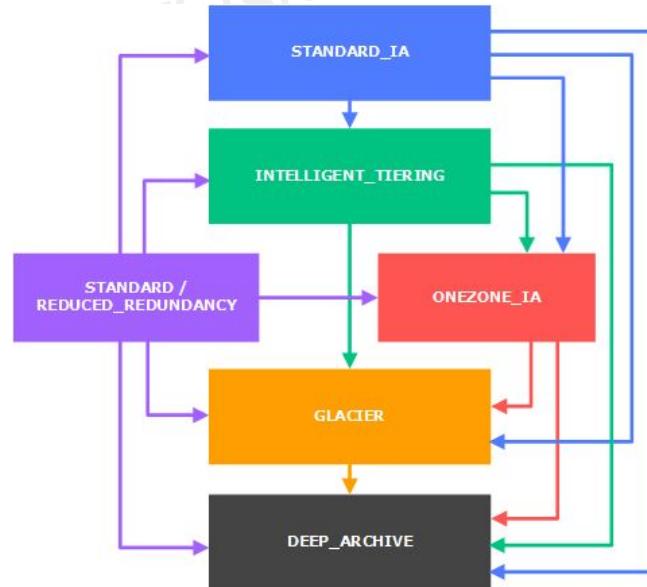
# Comparaison des classes de stockage S3

	S3 Standard	Hiérarchisation intelligente S3*	S3 Standard - IA	S3 One Zone-IA†	S3 Glacier	S3 Glacier Deep Archive
Conçu pour la durabilité	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)	99,999999999 % (11 9s)
Conçu pour la disponibilité	99,99 %	99,9 %	99,9 %	99,5 %	99,99 %	99,99 %
Disponibilité SLA	99,9 %	99 %	99 %	99 %	99,9 %	99,9 %
Zones de disponibilité	≥3	≥3	≥3	1	≥3	≥3
Frais de capacité minimale par objet	N/A	N/A	128 Ko	128 Ko	40 Ko	40 Ko
Frais minimum de durée de stockage	N/A	30 jours	30 jours	30 jours	90 jours	180 jours
Frais d'extraction	N/A	N/A	par Go extrait	par Go extrait	par Go extrait	par Go extrait
Latence du premier octet	millisecondes	millisecondes	millisecondes	millisecondes	sélectionnez minutes ou heures	sélectionnez heures

<https://aws.amazon.com/fr/s3/storage-classes/>

# S3 - Déplacer les objets entre les classes de stockage

- Il est possible de déplacer des objets d'une classe de stockage à une autre
- Pour des objets peu fréquemment demandés, on peut les déplacer dans STANDARD\_IA
- Pour des archives dont on n'a pas besoin en temps réel, on peut les déplacer dans GLACIER ou DEEP\_ARCHIVE
- Ce déplacement peut être automatisé par une **configuration de cycle de vie**



# S3 - règles de cycle de vie

- **Transition actions:** détermine quand déplacer les objets
  - vers Standard IA 60 jours après création
  - vers Glacier après 6 mois
- **Expiration actions:** détermine quand supprimer les objets
  - les fichiers de log après 1 an
  - peut s'utiliser pour supprimer d'anciennes versions (si la gestion est activée)
  - peut s'utiliser pour supprimer des uploads multi-part incomplets
- Règles pouvant s'appliquer à un préfixe (ex: s3://mybucket/mp3/\*)
- Règles pouvant s'appliquer pour les objets “taggués” (ex: Country:France)

# S3 - règles de cycle de vie - scénario 1

- Votre application EC2 génère des miniatures de photos uploadées sur S3. Ces miniatures peuvent être aisément regénérées et n'ont besoin d'être conservées que 45 jours. On doit pouvoir retrouver les images source immédiatement durant ces 45 jours, ensuite on peut attendre jusqu'à 6 heures. Comment faire ?
  - Les images sources: classe STANDARD, avec une configuration de cycle de vie pour les placer dans GLACIER après 45 jours
  - Miniatures: classe ONEZONE\_IA, avec une configuration de cycle de vie pour les supprimer (expiration) après 45 jours

# S3 - règles de cycle de vie - scénario 2

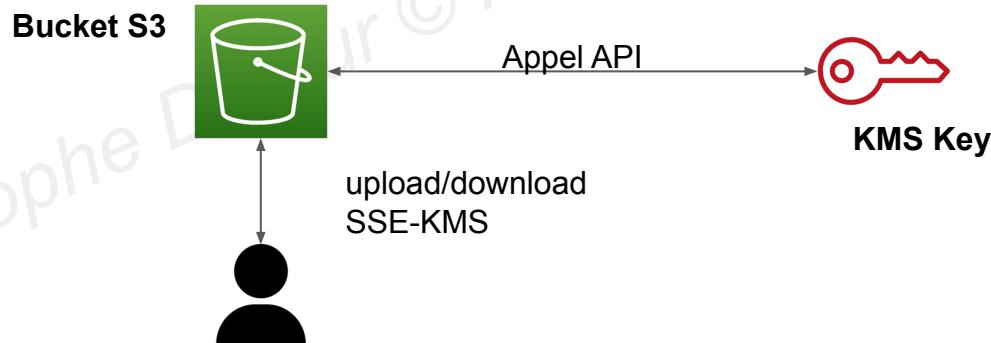
- Notre société veut qu'on soit capable de retrouver immédiatement les objets S3 supprimés, pendant 15 jours. Passé ce délai, et durant une période pouvant aller jusqu'à 365 jours, on peut attendre jusqu'à 48 heures pour les récupérer.
  - Il faut activer la gestion de version afin que les objets "supprimés" soient en fait "marqués" comme tels (delete marker) et donc récupérables
  - Les versions non courantes peuvent être déplacées vers S3\_IA
  - Elles seront ensuite déplacées vers DEEP\_ARCHIVE

# S3 - performances de base

- Amazon S3 dimensionne automatiquement à des volumes requête éléves, latence 100-220ms
- Nombre de requêtes par seconde et par prefixe dans un bucket:
  - 3500 PUT/COPY/POST/DELETE
  - 5500 GET/HEAD
- Pas de limite au nombre de prefixes dans un bucket
- Example (chemin vers l'objet      => prefixe)
  - bucket/folder1/sub1/file      => /folder/sub1/
  - bucket/folder1/sub2/file      => /folder/sub2/
  - bucket/1/file                  => /1/
  - bucket/2/file                  => /2/
- 22000 requêtes en GET/HEAD par seconde, distribuées sur les 4 prefixes

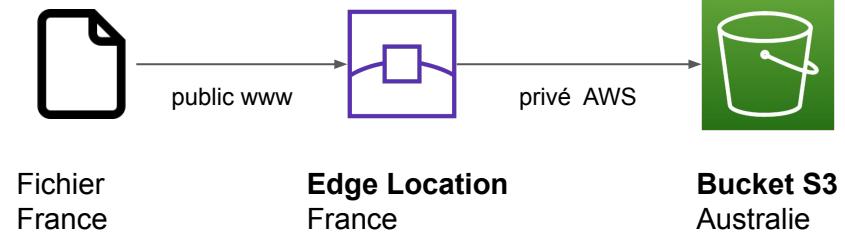
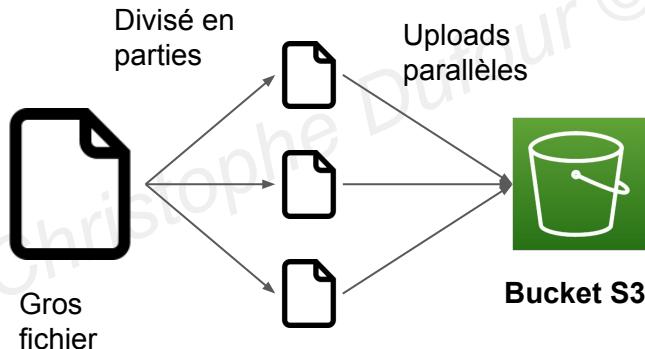
# S3 - limitation KMS

- L'utilisation de SSE-KMS peut impacter les limites
- Un upload de fichier, génère un appel API KMS **GenerateDataKey**
- Un téléchargement génère un appel API KMS **Decrypt**
- Quotas KMS: 5500, 10000, 30000 req/s, selon la région
- Actuellement, on peut pas demander une augmentation des quotas pour KMS



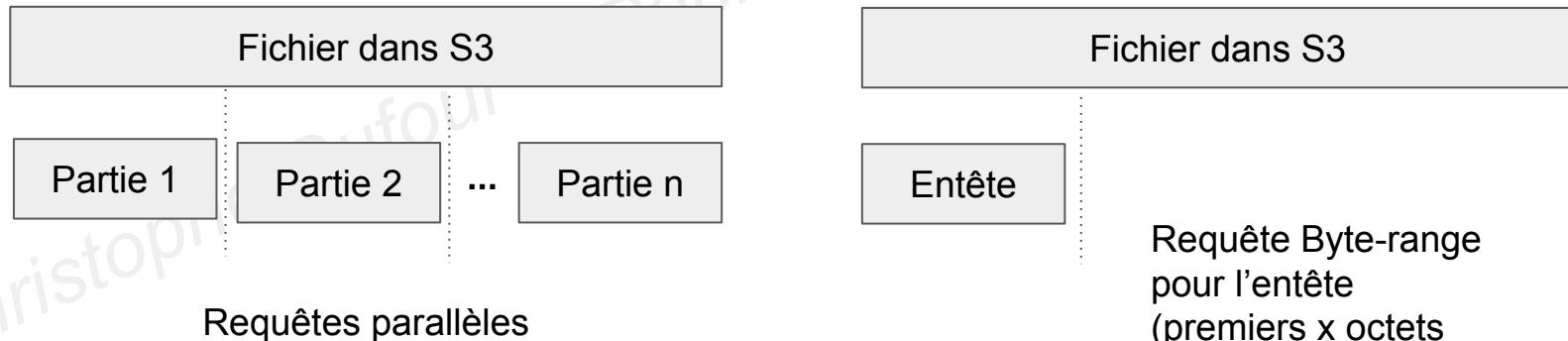
# S3 - performance

- **Multi-Part upload**
  - recommandé pour fichiers > 100MB, obligatoire > 5 GB
  - aide à paralléliser les uploads (gain en vitesse)
- **S3 Transfer Acceleration (upload uniquement)**
  - augmente la vitesse de transfert via une “edge location”
  - compatible avec l'upload multi-part



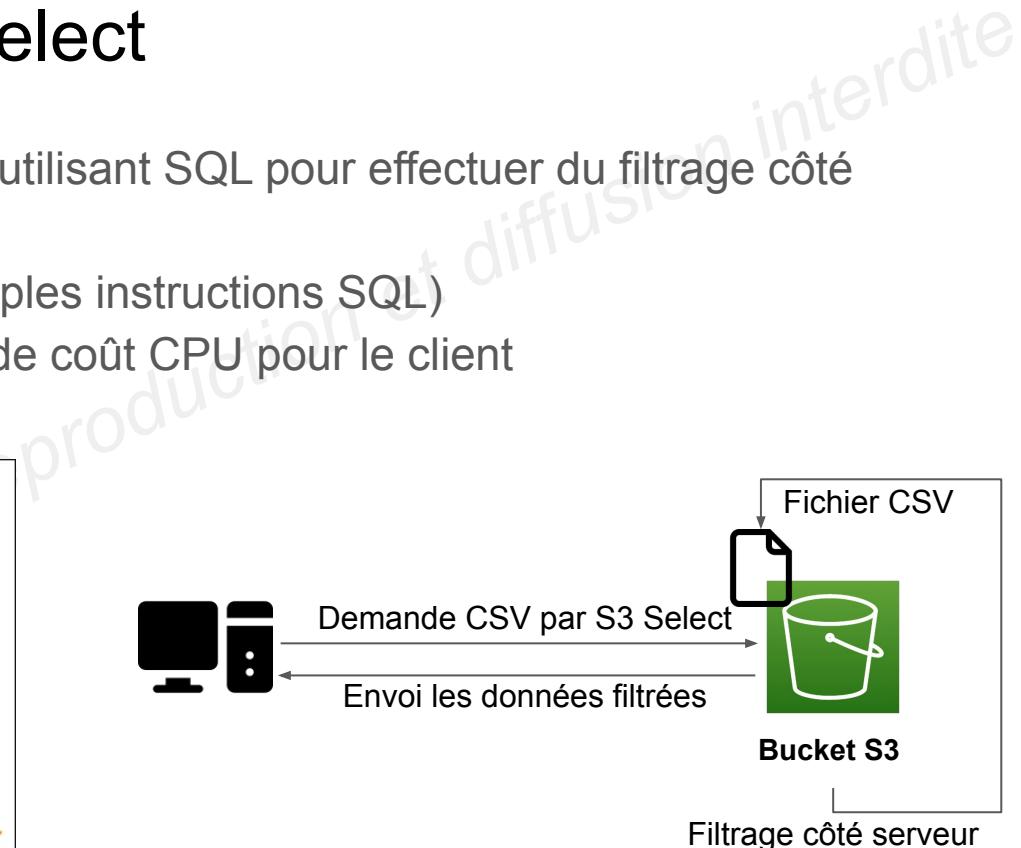
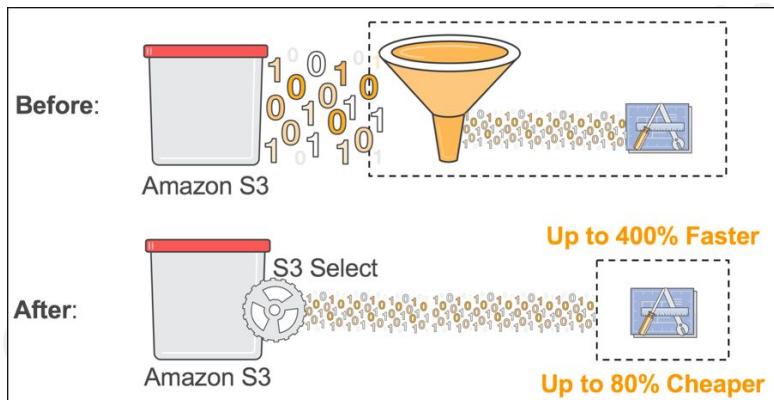
# S3 - performance - S3 Byte-Range Fetches

- Parallélise les requêtes en GET en requérant une plage octets précise
- Meilleure résistance aux défaillances
- Peut être utilisé pour accélérer les téléchargements
- Peut être utilisé pour retrouver des données partielles (ex: entête d'un fichier)



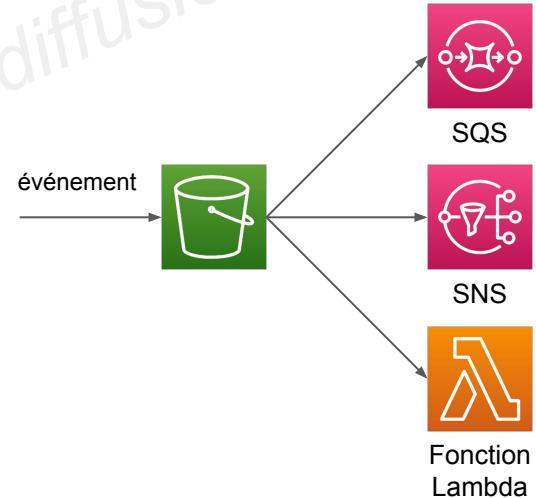
# S3 - Select et Glacier Select

- Récupère moins de données en utilisant SQL pour effectuer du filtrage côté serveur
- Filtrage par ligne et colonne (simples instructions SQL)
- Moins de charge réseau, moins de coût CPU pour le client



# S3 - Notifications d'événement

- S3:ObjectCreate, S3:ObjectRemoved, S3:ObjectRestore, S3:Replication,...
- Possible filtrage par nom (\*.jpg)
- Cas d'utilisation: générer des miniatures d'image uploadées sur S3
- On peut créer autant d'événement S3 que souhaité
- Délai d'envoi: généralement quelques secondes, parfois quelques minutes
- Si deux écritures sont faites en même temps sur le même objet non versionné, il est possible qu'une seule notification soit envoyée
- Pour s'assurer que toute écriture s'accompagne d'un envoi de notification, on peut activer la gestion de version sur le bucket



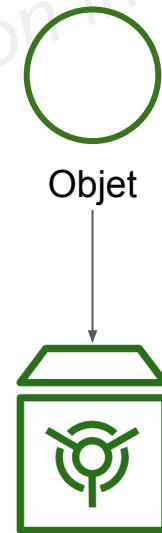


# AWS Athena

- Service **serverless** pour de l'**analyse directe d'objets S3**
- Utilise le langage SQL pour interroger les fichiers
- Dispose d'un driver JDBC/ODBC
- Facturation par requête et volume de données scannées
- Supporte les formats CSV, JSON, ORC, Avro et Parquet
- Cas d'utilisation: analyse, reporting, logs de flux VPC, logs d'ELB, logs CloudTrail, etc...
- Astuce examen: analyse directe sur S3 => Athena

# S3 - Object Lock et Glacier Vault Lock

- **S3 Object Lock**
  - Modèle WORM (Write Once Read Many)
  - Bloque la suppression d'une version d'objet pour une période déterminée
- **Glacier Vault Lock**
  - Modèle WORM
  - Bloque la stratégie pour les futures mises à jour (ne peut plus être modifiée)
  - Utile pour la conformité et la rétention de données



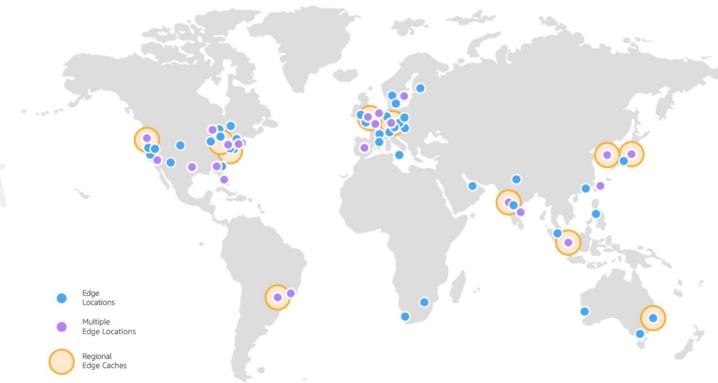
**Vault Lock**  
L'objet ne peut pas être supprimé

# CloudFront



# AWS CloudFront

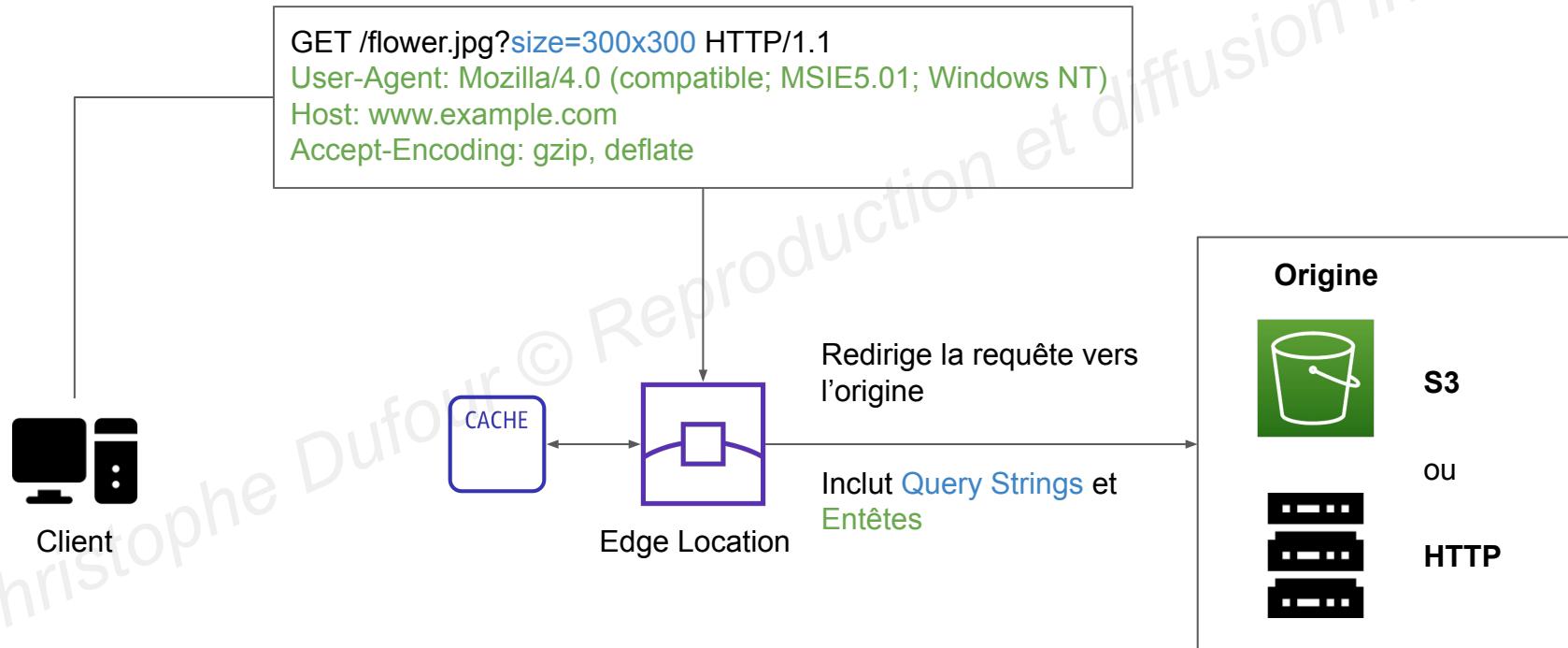
- Content Delivery Network (CDN)
- Améliore les performances en lecture, le contenu est mis en cache “at the edge”
- 216 points de présence globale (edge locations)
- Protection contre DDoS, intégration avec Shield et AWS WAF
- Peut exposer de points d'accès externes HTTPS et communiquer avec des points d'accès internes HTTPS



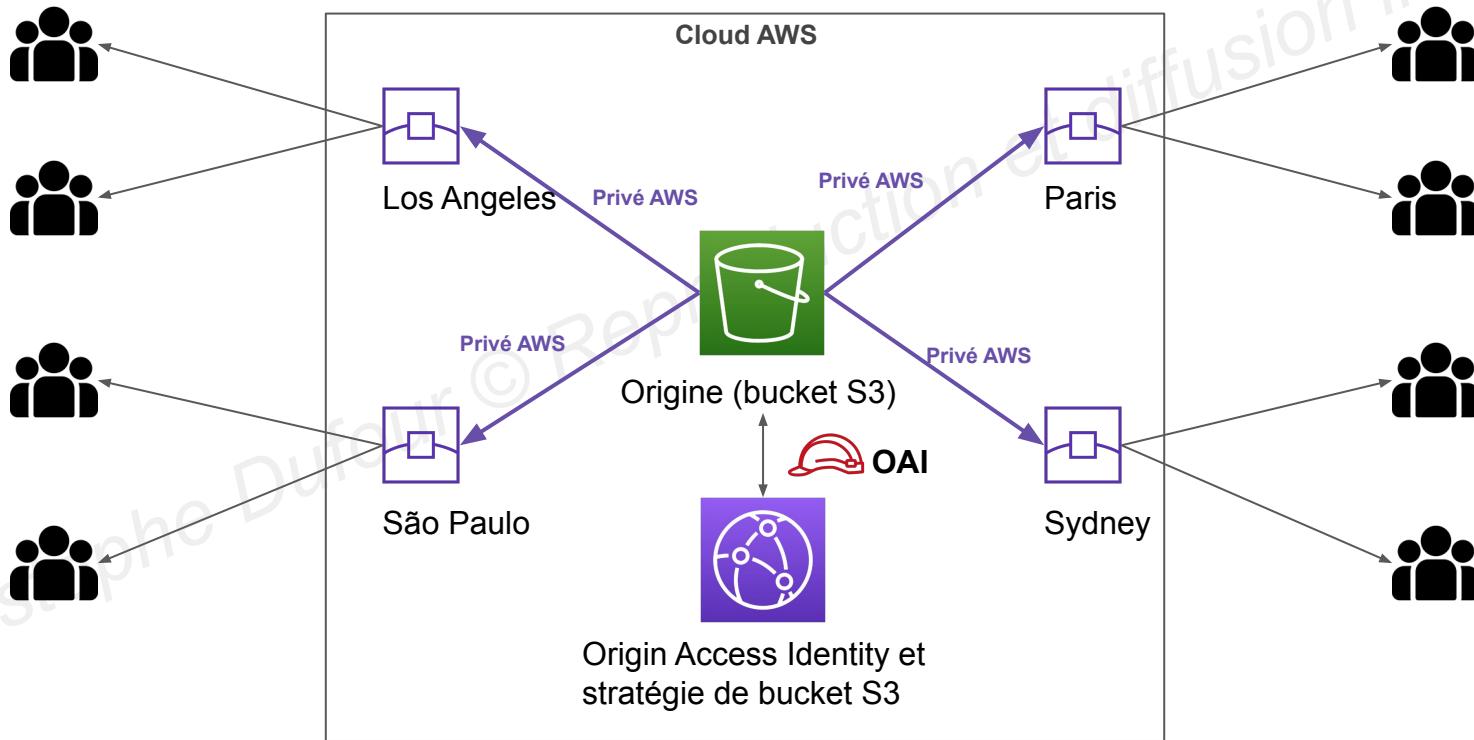
# CloudFront - Origines

- **Bucket S3**
  - Distribution de fichiers et leur mise en cache “at the edge”
  - Renfort de la sécurité grâce à CloudFront **Origin Access Identity (OAI)**
  - CloudFront peut être utilisé comme une entrée (pour upload dans S3)
- **Custom Origin (HTTP)**
  - Application Load Balancer
  - Instance EC2
  - Website S3 (le bucket doit d'abord être paramétré comme site statique)
  - N'importe quel backend HTTP

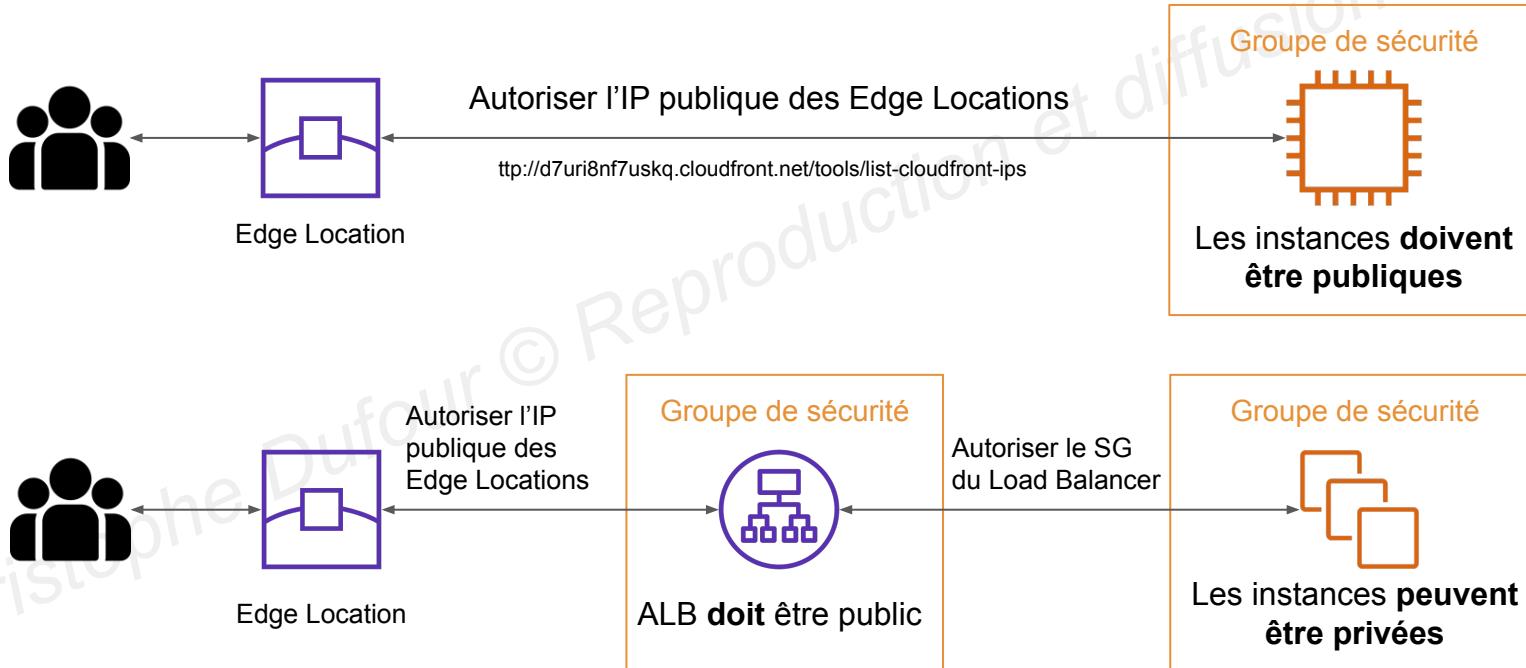
# CloudFront - Schéma



# CloudFront - S3 comme origine



# CloudFront - ALB ou EC2 comme origine



# CloudFront - Geo Restriction

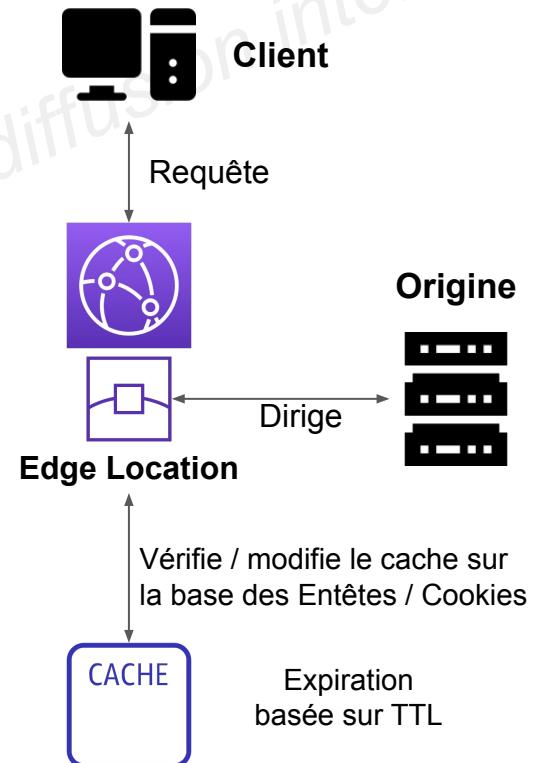
- Restrictions d'accès basés sur l'emplacement géographique du client
  - **Whitelist:** peuvent accéder aux contenus les utilisateurs dont le pays figure dans la liste des pays autorisés
  - **Blacklist:** accès bloqué pour les utilisateurs dont le pays figure dans la liste des pays bannis
- Le "pays" est déterminé par l'utilisation d'une base données tiers d'adresses IP géolocalisées
- Cas d'utilisation: lois copyright pour certains contenus

# CloudFront vs S3 CRR

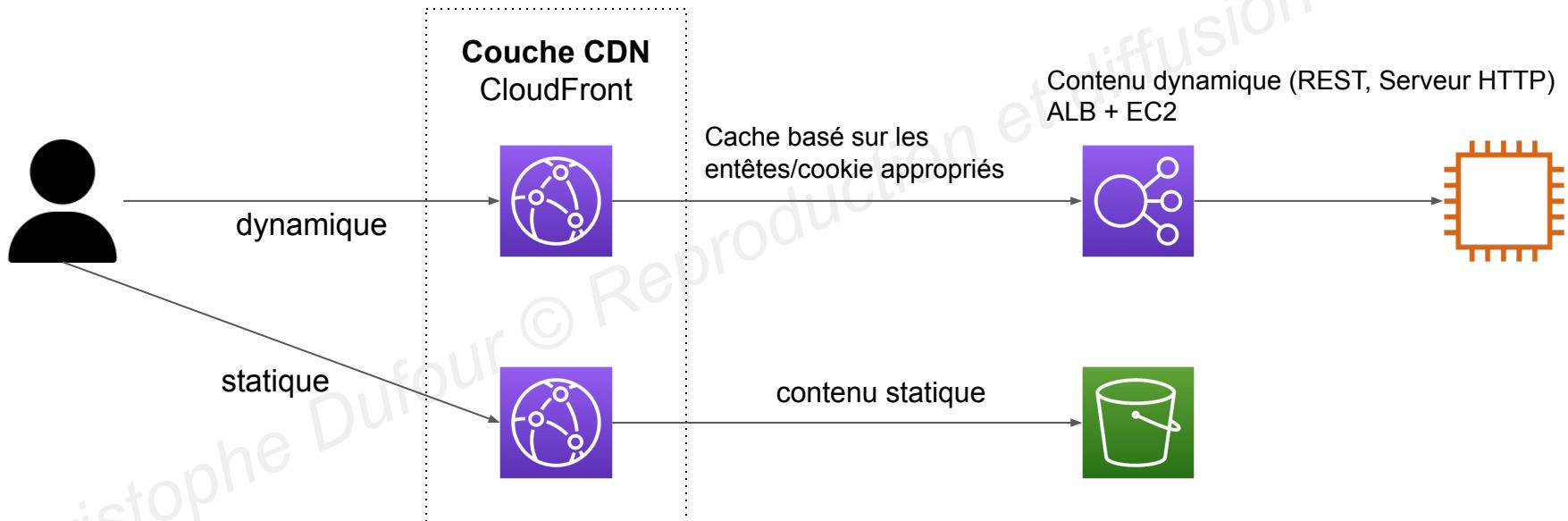
- CloudFront
  - Réseau global à multiples points d'accès (edges)
  - Fichiers mise en cache pendant un TTL (ex: un jour)
  - **Idéal pour du contenu statique devant être disponible partout**
- S3 Cross Region Replication
  - Doit être paramétré pour chaque région devant recevoir la réPLICATION
  - Fichiers quasiment mis à jour en temps réel
  - Lecture seule
  - **Parfait pour du contenu dynamique devant être disponible à faible latence dans quelques régions**

# CloudFront - Mise en cache

- Cache basé sur
  - Entête
  - Cookie de session
  - Paramètres de Query String
- Le cache est présent à chaque **Edge Location**
- Maximiser le taux de “cache hit” pour réduire le nombre de requêtes vers l'origine
- Contrôle du TTL (0 sec. à 1 an), peut être défini par l'origine grâce aux entêtes Cache-Control, Expires, etc.
- Invalidation possible via l'API **CreateInvalidation**

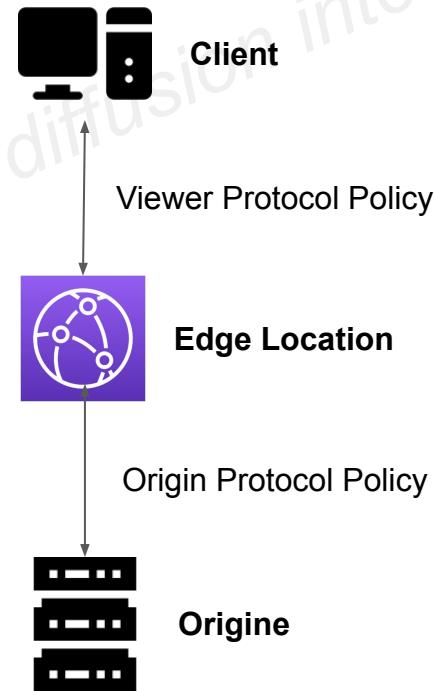


# CloudFront - Maximisation des “cache hits” par séparation des distributions



# CloudFront et HTTPS

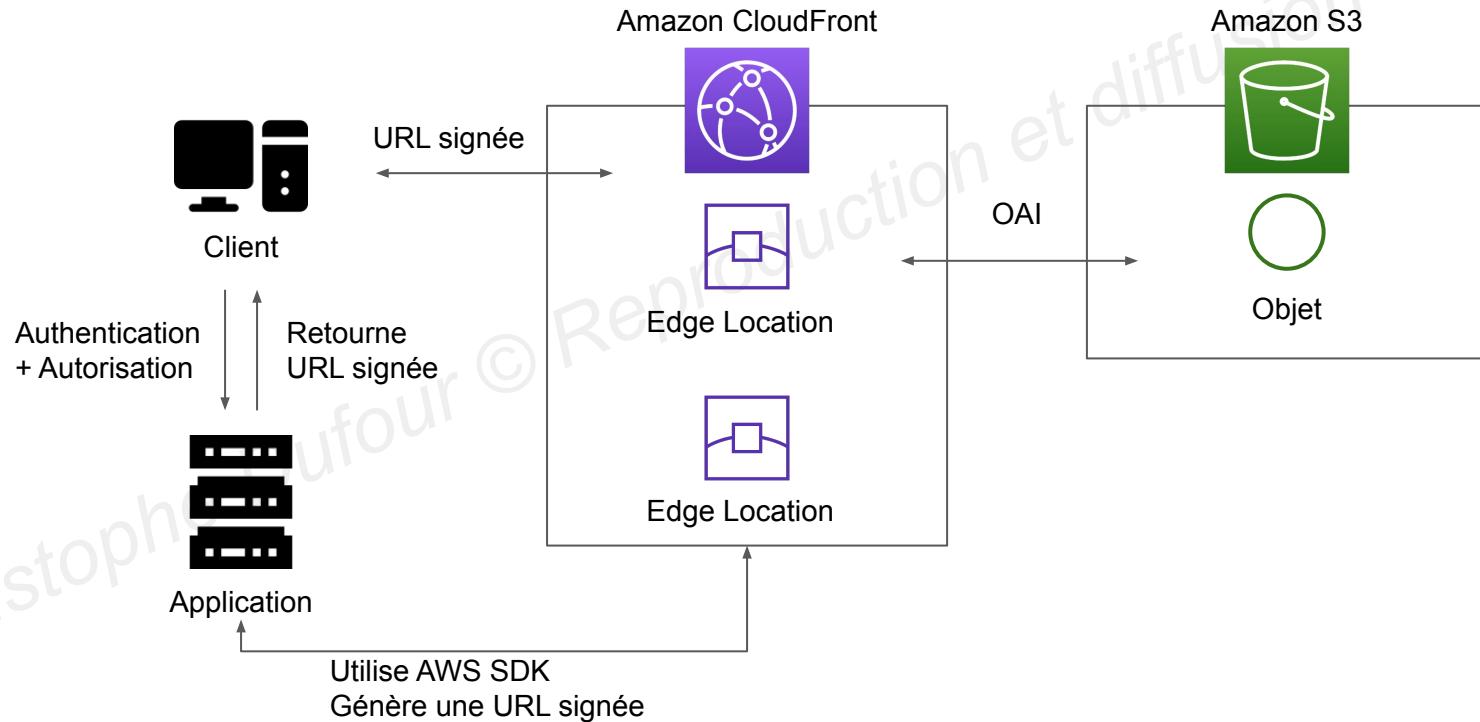
- **Viewer Protocol Policy**
  - Rediriger HTTP vers HTTPS
  - Utiliser HTTPS uniquement
- **Origin Protocol Policy (HTTP ou S3)**
  - HTTPS uniquement
  - Match Viewer (http => http & https => https)
- Note: les sites S3 ne supportent pas HTTPS



# CloudFront - URL signée / Cookie signé

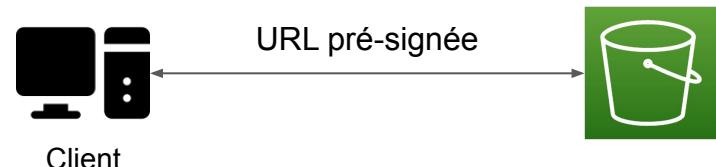
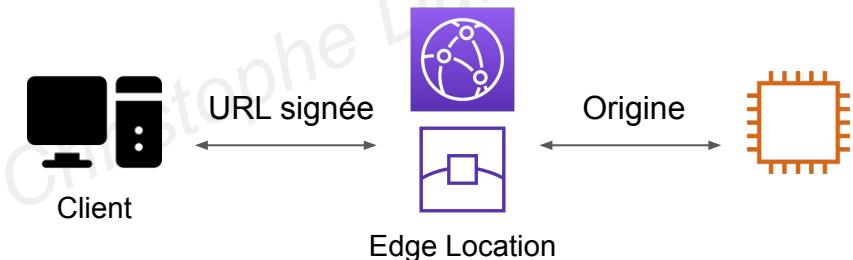
- On souhaite distribuer du contenu partagé payant à des membres premium partout dans le monde
- Pour restreindre l'accès, on peut créer un CloudFront Signed URL / Cookie
- Durée de validité conseillée ?
  - Contenu partagé (film, musique): minutes/heures
  - Contenu privé (pour l'utilisateur): années
- URL signée: accès à de fichiers individuels (une URL signée pour 1 fichier)
- Cookie signé: accès à de multiples fichiers (un cookie signé pour n fichier)

# URL signée CloudFront - Schéma



# URL signée CloudFront vs URL pré-signée S3

- **URL signée CloudFront**
  - Autorise l'accès à un chemin, peu importe l'origine
  - Seul le root peut gérer
  - Peu être filtrée par IP, chemin, date, expiration
  - Peut bénéficier du cache
- **URL pré-signée S3**
  - Effectue la requête en tant qu'utilisateur ayant pré-signé l'URL
  - Utilise la clé IAM de l'utilisateur IAM ayant signé
  - Durée de vie limitée



# AWS ECS - les bases

Conteneurs Docker dans AWS



# ECS - intro

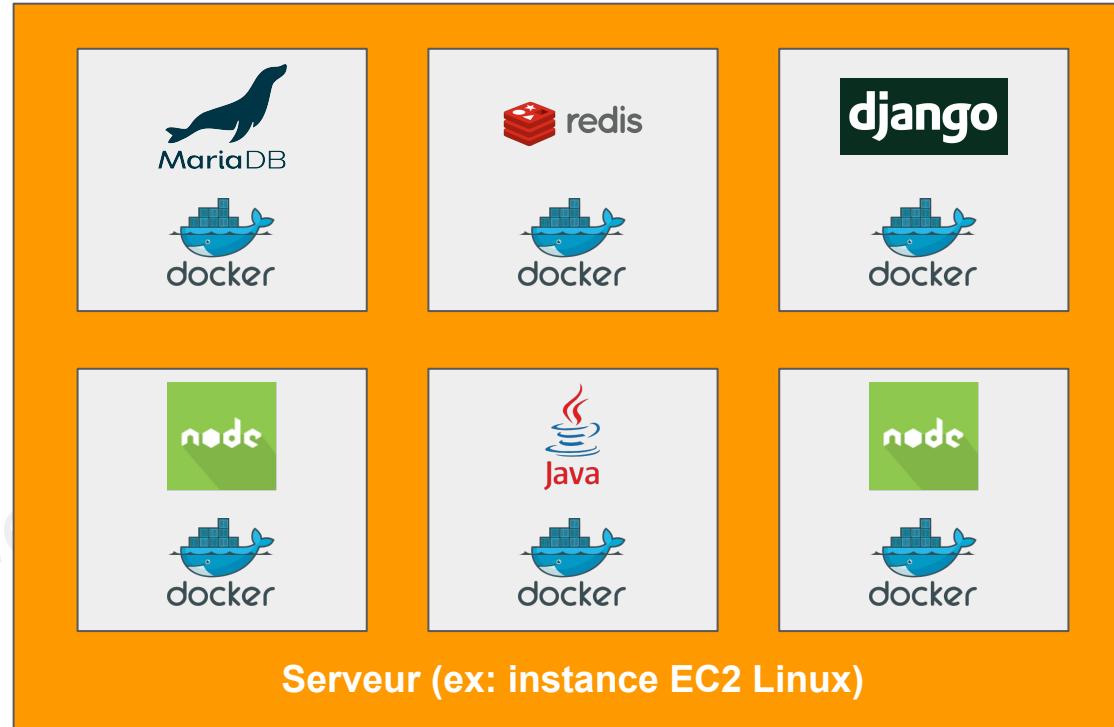
- Intro à Docker
- ECS
  - cluster
  - services
  - tasks
  - tasks definition
- ECR
- Fargate
- Astuces pour l'examen



# C'est quoi Docker ?

- Docker est une plateforme de développement logiciel destinée au déploiement d'applications
- Les applications sont packagées dans des **conteneurs** pouvant être exécutés sur n'importe quel OS
- Les applications fonctionnent de façon identique
  - peu importe la machine
  - pas de problèmes de compatibilité, de dépendances
  - comportement prévisible
  - allège la charge administrative
  - plus facile à maintenir et déployer
  - virtuellement, tous les OS, langages, applicatifs sont “conteneurisables”

# Docker sur OS

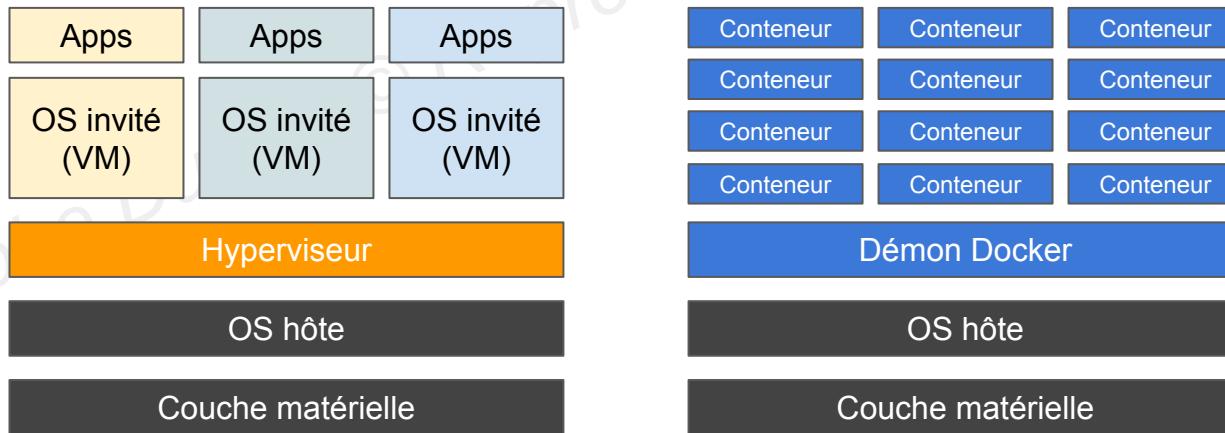


# Images Docker

- Une image est un fichier read-only (immutable) à partir duquel des instances (= conteneurs) sont générées
- Un conteneur reproduit exactement les caractéristiques de l'image sur laquelle il se base
- Docker permet de créer ses propres images (Dockerfile)
- Un grand nombre d'images déjà faites par d'autres sont disponibles
- Les images sont stockées
  - localement
  - dans un dépôt (repository) distant, public ou privé.
    - Docker Hub
    - Amazon ECR (Elastic Container Registry)

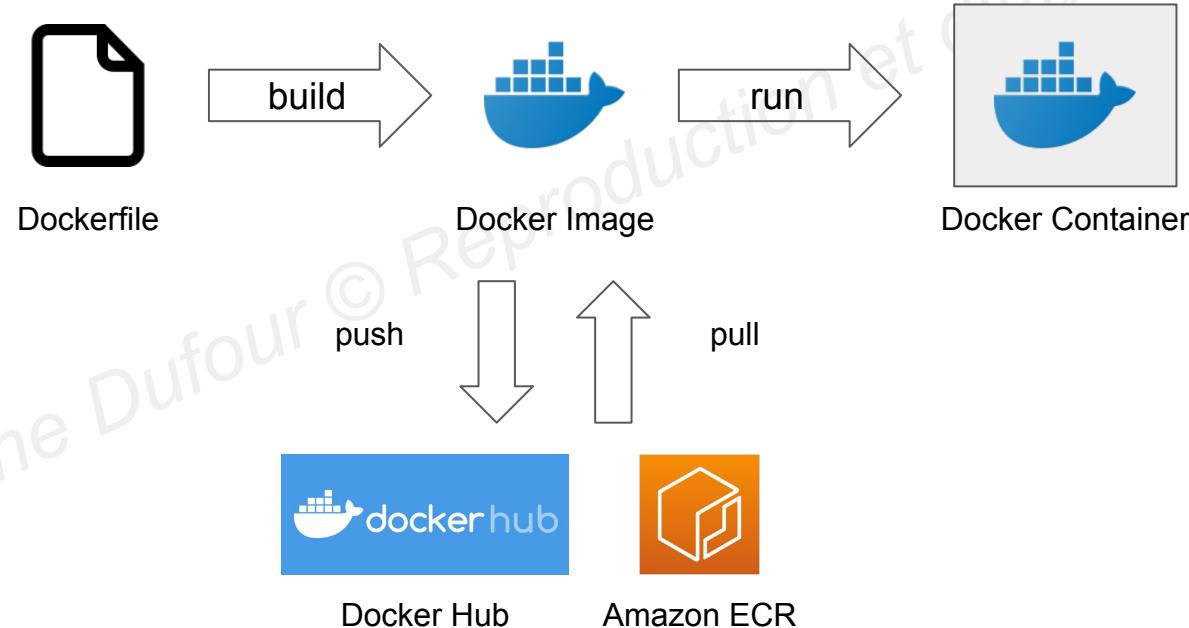
# Docker vs machines virtuelles

- Docker est également une technologie de virtualisation mais se différenciant des hyperviseurs tels que VirtualBox ou VMWare
- Les ressources sont partagées avec l'hôte, qui peut héberger bien plus de conteneurs que de machines virtuelles complètes



# Démarrer avec Docker

- Télécharger Docker: <https://www.docker.com/get-started>

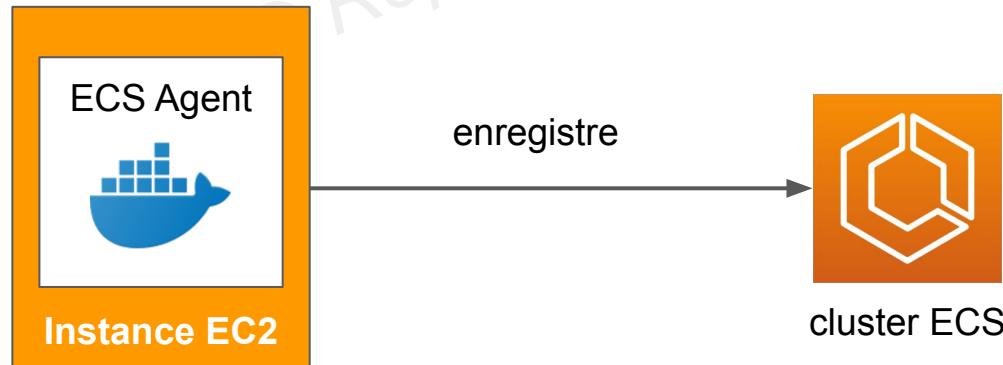


# Gestion des conteneurs Docker sur AWS

- Nécessité d'une plateforme de gestion (dev/prod)
- Amazon offre 3 possibilités en termes de gestion de conteneurs
  - **ECS**: plateforme propriétaire Amazon
  - **Fargate**: plateforme propriétaire “serverless” Amazon
  - **EKS**: plateforme kubernetes (open source) gérée par Amazon

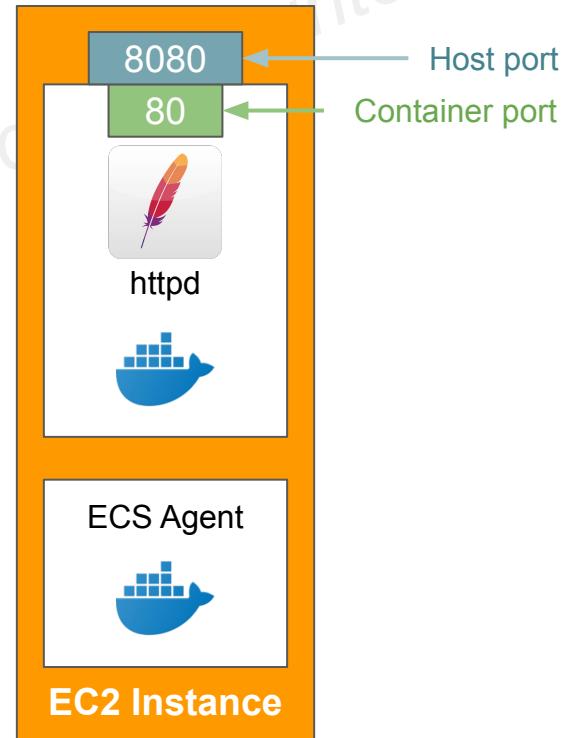
# ECS Clusters

- Regroupement logique d'instances EC2
- Les instances EC2 exécutent l'ECS agent (Docker container)
- Les agents ECS enregistrent l'instance dans le cluster ECS
- Les instances EC2 sont basées sur une AMI spécifique à ECS



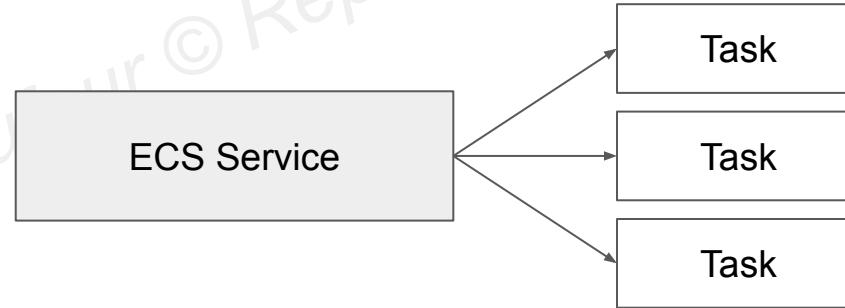
# ECS Task Definitions

- Métadonnées au **format JSON** indiquant à ECS comment instancier un conteneur
- Contient des informations importantes telles que:
  - nom de l'image (exemple: httpd:2.4)
  - correspondance de port entre l'hôte et le container
  - mémoire et le CPU requis
  - variables d'environnement
  - configuration réseau
  - rôle IAM
  - configuration de logging (ex: CloudWatch)

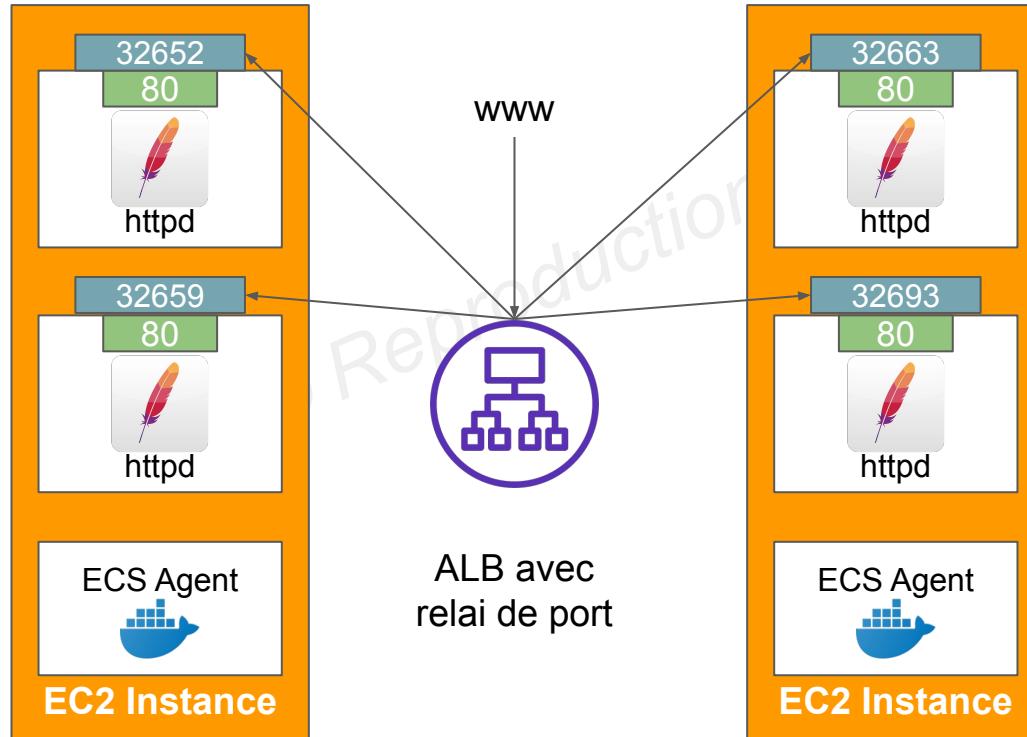


# ECS Service

- Permet de définir combien de tâches doivent être exécutées et comment
- Vérifie que le nombre de tâches souhaitées est respecté à travers notre “flotte” d’instances EC2
- Peut être relié à un Load Balancer si besoin



# ECS Service avec Load Balancer





# ECR (Elastic Container Service)

- ECR est un dépôt privé d'images Docker
- Accès contrôlé via IAM
- Commande de login AWS CLI v1 (exam)
  - \$(aws ecr get-login --no-include-email --region eu-central-3)
- Commande de login AWS CLI v2
  - aws ecr get-login-password --region eu-central-3 | docker login --username AWS --password-stdin 1234567890.dkr.ecr.eu-central-3.amazonaws.com
- Docker Push & Pull
  - docker push 1234567890.dkr.ecr.eu-central-3.amazonaws.com/demo:latest
  - docker pull 1234567890.dkr.ecr.eu-central-3.amazonaws.com/demo:latest



# Fargate

- La création d'un cluster ECS implique de créer nos instances EC2
- Si nous devons dimensionner, il nous faut ajouter des instances EC2
- Des tâches d'administrations sont donc requises...
- A la différence d'ECS, Fargate est “serverless”
  - pas d'approvisionnement d'instances EC2
  - seules des Task Definitions sont à spécifier
  - pour dimensionner, il suffit d'augmenter le nombre de tâches

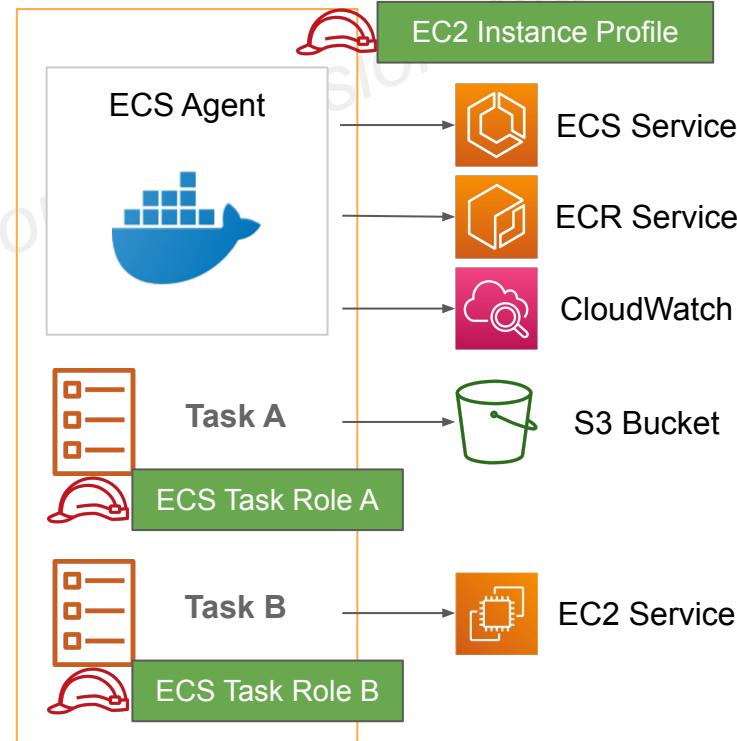
# ECS - Rôles IAM

- **EC2 Instance Profile**

- utilisé par l'agent ECS
- effectue des appels API vers ECS service
- envoie les logs du conteneur à CloudWatch Logs
- récupère l'image Docker depuis ECR

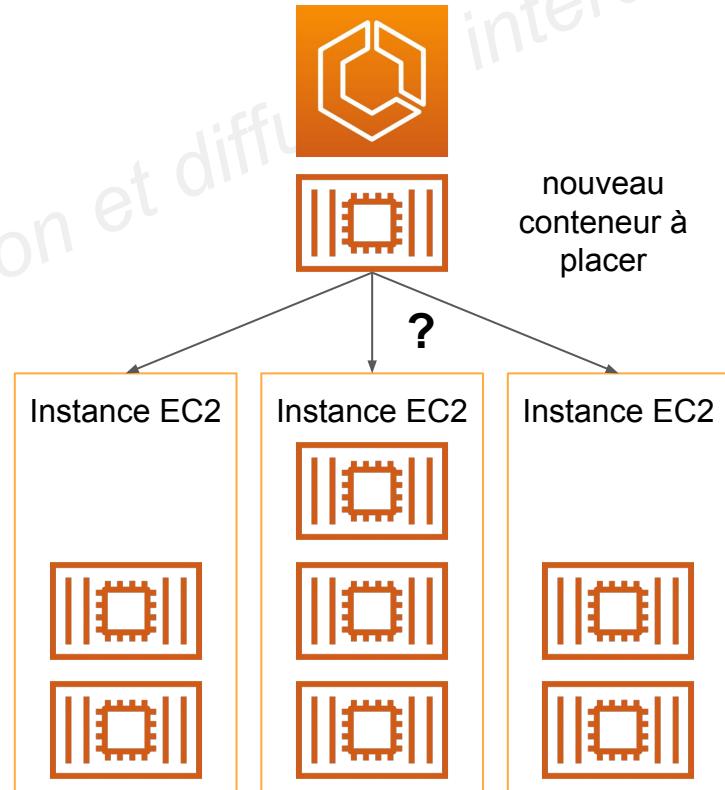
- **ECS Task Role**

- autorise chaque tâche à suivre un rôle spécifique
- utilise différents rôles pour les différents services ECS lancés
- le Task Role est défini dans le **task definition**



# ECS - placement de tâche

- Quand une tâche de type EC2 est lancée, ECS doit déterminer où la placer, ainsi que les contraintes de CPU, RAM et ports disponibles
- Quan un service dimensionne à la baisse, ECS doit détermoner quelles tâches terminer
- Cette problématique peut être abordée via une stratégie de placement de tâche et les contraintes de placement de tâche
- N.b: ceci ne concerne pas Fargate



# ECS - processus de placement de tâche

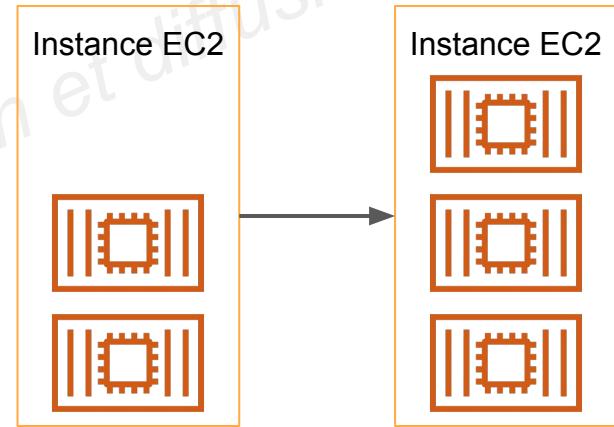
- Quand Amazon ECS place des tâches, il suit le processus suivant pour sa sélection d'instances
  1. identification des instances satisfaisant les pré-requis CPU, mémoire et port dans la définition de tâche
  2. identification des instances satisfaisant les contraintes de placement
  3. identification des instances satisfaisant les stratégies de placement
  4. sélection des instances pour le placement

# ECS - stratégies de placement de tâche

- **Binpack**

- place les tâches sur la base de la quantité minimale CPU/mémoire disponible dans l'instance
- réduit le nombre d'instances utilisées (économies)

```
"placementStrategy": [  
    {  
        "field": "memory",  
        "type": "binpack"  
    }  
]
```



# ECS - stratégies de placement de tâche

- **Random**
  - place les tâches de façon aléatoire

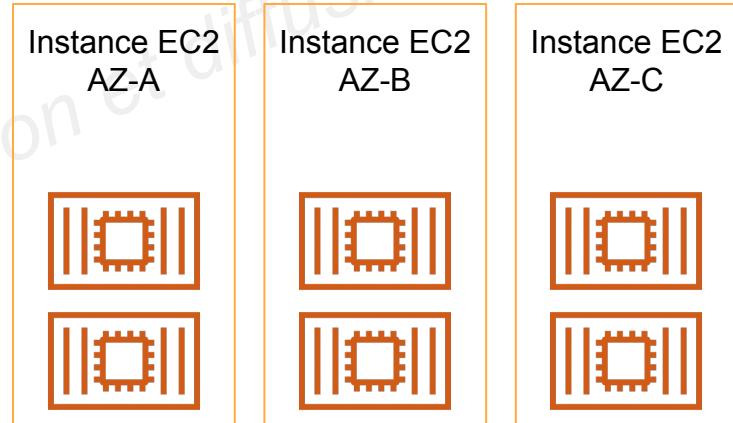
```
"placementStrategy": [  
    {  
        "type": "random"  
    }  
]
```

# ECS - stratégies de placement de tâche

- **Spread**

- place les tâches équitablement sur la base de la valeur spécifiée
- ex: instanceId,  
attribute:ecs.availability-zone

```
"placementStrategy": [  
    {  
        "field": "attribute:ecs.availability-zone",  
        "type": "spread"  
    }  
]
```



# ECS - stratégies de placement de tâche

- Les stratégies peuvent être combinées

```
"placementStrategy": [  
    {  
        "field": "attribute:ecs.availability-zone",  
        "type": "spread"  
    },  
    {  
        "field": "instanceId,  
        "type": "spread"  
    }  
]
```

```
"placementStrategy": [  
    {  
        "field": "attribute:ecs.availability-zone",  
        "type": "spread"  
    },  
    {  
        "field": "memory,  
        "type": "binpack"  
    }  
]
```

# ECS - contraintes de placement de tâche

- **distinctDistance**: place chaque tâche dans une instance différente

```
"placementConstraints": [{  
    "type": "distinctInstance"  
}]
```

- **memberOf**: place la tâche dans les instances satisfaisant une expression Cluster Query Language (avancé)

```
"placementConstraints": [{  
    "expression": "attribute:ecs.instance-type =~ t2.*",  
    "type": "memberOf"  
}]
```

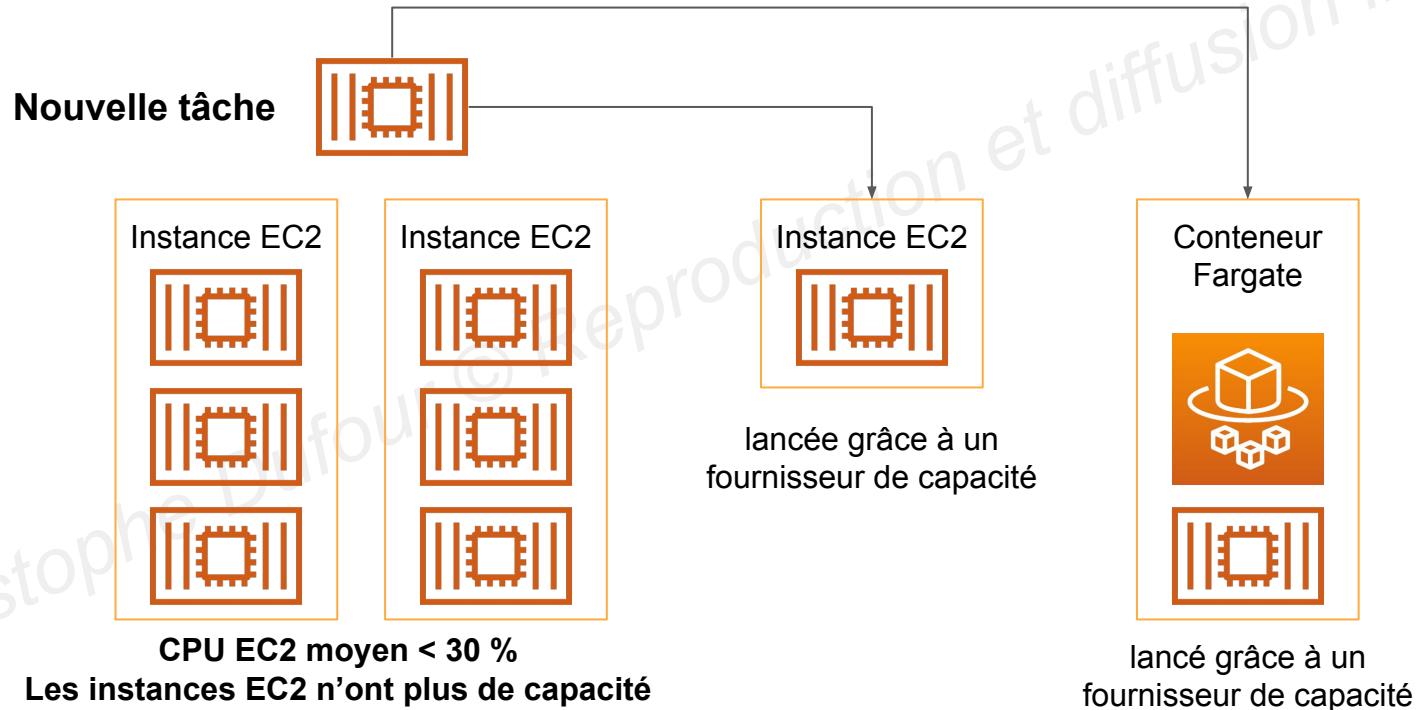
# ECS - Auto-dimensionnement du service

- CPU et RAM sont surveillés par CloudWatch au niveau du service ECS
- **Target Tracking**: cible une métrique spécifique CloudWatch
- **Step Scaling**: dimensionne sur la base d'alarmes CloudWatch
- **Scheduled Scaling**: dimensionne sur la base de changements prévisibles
- ECS Service Scaling (task level) **!≠** EC2 Auto Scaling (instance level)
- L'auto-dimensionnement de Fargate est bien plus simple à paramétrier (serverless)

# ECS - Cluster Capacity Provider

- Un fournisseur de capacité est utilisé conjointement au cluster afin de déterminer l'infrastructure à exécuter
  - pour ECS/EC2, il faut associer un fournisseur de capacité et un ASG
  - pour ECS/Fargate, les fournisseurs de capacité FARGATE et FARGATE\_SPOT sont automatiquement ajoutés
- Quand on exécute une tâche ou un service, on définit une stratégie de fournisseur de capacité afin de déterminer lequel utiliser en priorité
- Ceci permet au fournisseur de capacité d'approvisionner automatiquement l'infrastructure

# ECS - Cluster Capacity Provider



# ECS - résumé + astuces exam

- ECS est utilisé pour exécuter des conteneurs Docker de 3 façons:
  - ECS “classique”: approvisionnement d’instances EC2 dans lesquelles les conteneurs seront lancés
  - Fargate: ECS serverless, pas d’approvisionnement d’instances EC2
  - EKS: clusters K8S gérés par AWS

# ECS classique

- Les instances EC2 doivent être créées
- Il faut configurer le fichier **/etc/ecs/ecs.config** avec le nom du cluster
- Les instances EC2 doivent exécuter un agent ECS
- Les instances EC2 peuvent exécuter plusieurs conteneurs du même type
  - il ne faut pas spécifier le port du hôte (seul le port du conteneur)
  - il est recommandé d'utiliser un ALB avec mapping de port dynamique
  - le SG EC2 doit autoriser le trafic provenant du ALB
- Les tâches ECS peuvent avoir des rôles IAM afin d'utiliser des services AWS
- Les groupes de sécurité agissent au niveau de l'instance pas de la tâche

# ECR pour le stockage des images Docker

- ECR est étroitement intégré avec IAM
- Commande de login AWS CLI v1 (exam)
  - \$(aws ecr get-login --no-include-email --region eu-central-3)
- Commande de login AWS CLI v2 (pipe character)
  - aws ecr get-login-password --region eu-central-3 | docker login --username AWS --password-stdin 1234567890.dkr.ecr.eu-central-3.amazonaws.com
- Docker Push & Pull
  - docker push 1234567890.dkr.ecr.eu-central-3.amazonaws.com/demo:latest
  - docker pull 1234567890.dkr.ecr.eu-central-3.amazonaws.com/demo:latest
- Dans le cas où une instance ne parvient pas à récupérer une image Docker, vérifier IAM

# Fargate

- Fargate est serverless (pas d'instances EC2 à gérer)
- AWS approvisionne les conteneurs pour nous et leur assigne un ENI (Elastic Network Interface)
- Les conteneurs Fargate sont approvisionnés selon les spécification conteneur (CPU/RAM)
- Les tâches Fargate peuvent avoir des rôles IAM afin d'utiliser des services AWS

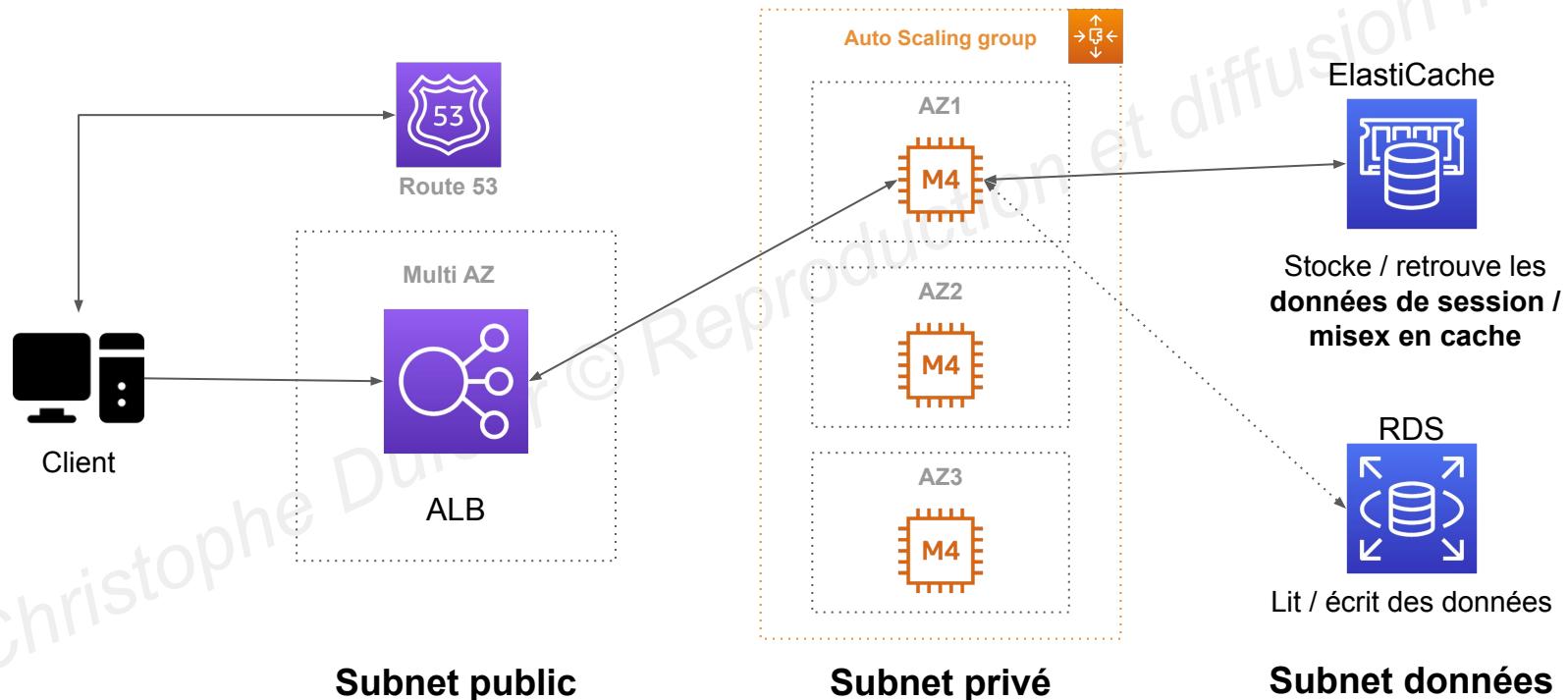
# ECS - autres

- ECS est intégré à CloudWatch Logs
  - il faut paramétrier le logging au niveau de la définition de tâche
  - chaque conteneur aura un flux de log (stream) différent
  - l'Instance Profile EC2 doit avoir les droits IAM appropriés
- Utiliser des rôles IAM pour les tâches
- Stratégies de placement de tâche: binpack, random, spread
- Service Auto Scaling: target tracking, step scaling, scheduled scaling
- Cluster Auto Scaling: Capacity Providers

# AWS Elastic Beanstalk

Déployer des applications dans AWS de façon sécurisée et prévisible

# Architecture classique: application web 3 tiers



# Problèmes des développeurs sur AWS

- Gérer l'infrastructure
  - Déployer le code
  - Configurer base de données, load balancers, etc
  - Dimensionnement (scaling)
- 
- La plupart des applications web ont la même architecture (ALB + ASG)
  - Les développeurs veulent que leur code fonctionne au plus vite !
  - Et si possible, de manière stable, dans différents environnements

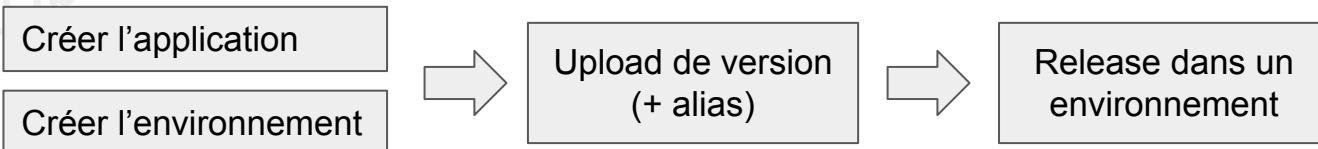
# Elastic Beanstalk - vue d'ensemble



- EB est service focalisée sur la vision développeur en vue d'un déploiement d'application sur AWS
- EB utilise en arrière-plan les services qu'on a rencontrés précédemment: EC2, ASG, ELB, RDS, etc...
- Mais dans un seul cadre cohérent
- Nous conservons le contrôle complet de la configuration
- EB est gratuit en lui-même, sont payantes les ressources sous-jacentes

# Elastic Beanstalk

- EB a 3 composants principaux
  - Application
  - Application version: chaque déploiement reçoit un identifiant de version
  - Environment name (dev, test, prod, etc.): nom au choix
- Les versions d'application sont déployées dans des environnements
- Une version d'application peut être affectée (promoted) à un environnement cible
- Fonctionnalité de rollback à une version antérieure
- Contrôle total sur le cycle de vie des environnements

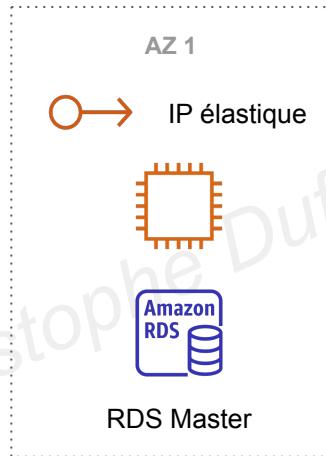


# Elastic Beanstalk

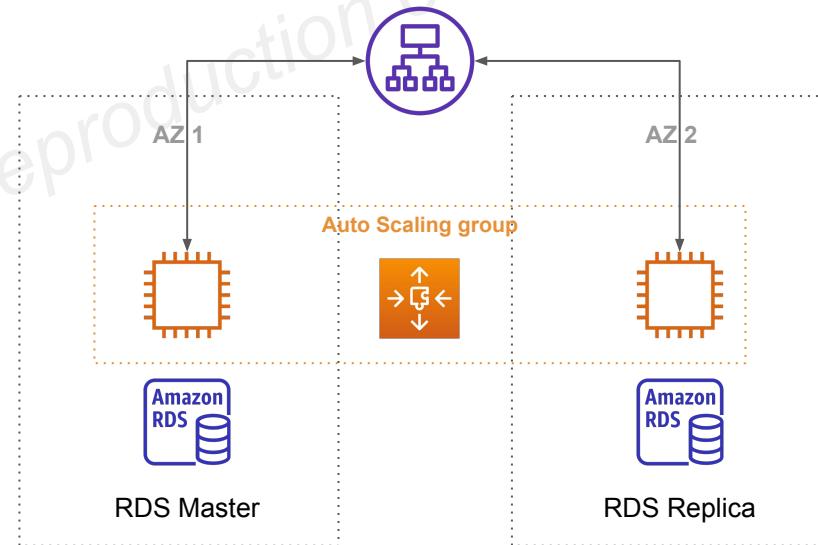
- Support de nombreux langages
  - Go
  - Java
  - .NET
  - Nodejs
  - PHP
  - Python
  - Ruby
  - Packer Builder
- Single Container Docker
- Multicontainer Docker
- Preconfigured Docker
- Possibilité de créer sa propre plateforme (avancé)

# Elastic Beanstalk - modes de déploiement

**Une seule instance**  
Oriентé dev



**Haute disponibilité avec LB**  
Orienté prod

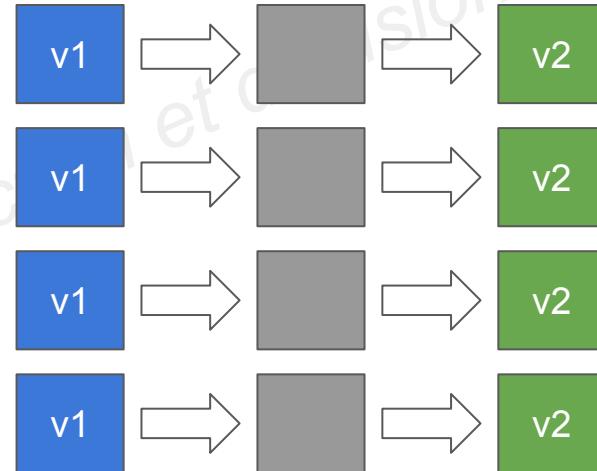


# Elastic Beanstalk - options de mise à jour

- **All at once**: le plus rapide mais les instances sont indisponibles pendant un quelques instants (downtime)
- **Rolling**: met à jour quelques instances en même temps (bucket) puis passe aux autres
- **Rolling with additional batches**: comme rolling mais crée de nouvelles instances (la version précédente reste disponible de temps de la mise à jour)
- **Immutable**: crée de nouvelles instances dans un nouvel ASG, déploie la nouvelle version dans ces instances puis échange toutes les instances quand tout est sain (healthy)

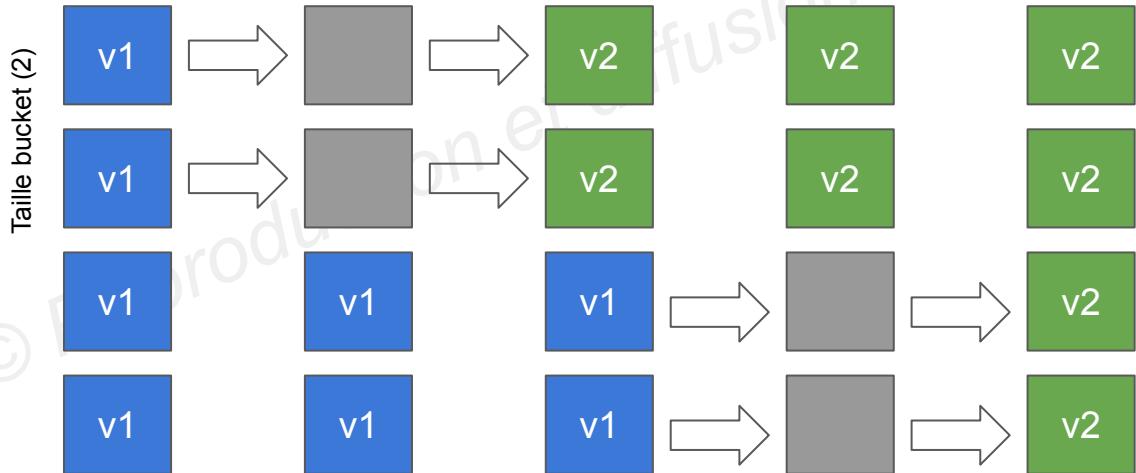
# Elastic Beanstalk - déploiement - All at once

- Le plus rapide
- Temps d'indisponibilité
- Bon pour des itérations rapides en environnement de dev
- Pas de coût additionnel



# Elastic Beanstalk - déploiement - Rolling

- L'application tourne à régime moindre
- Taille du bucket paramétrable
- Deux versions en simultané
- Déploiement assez long
- Pas de coût additionnel



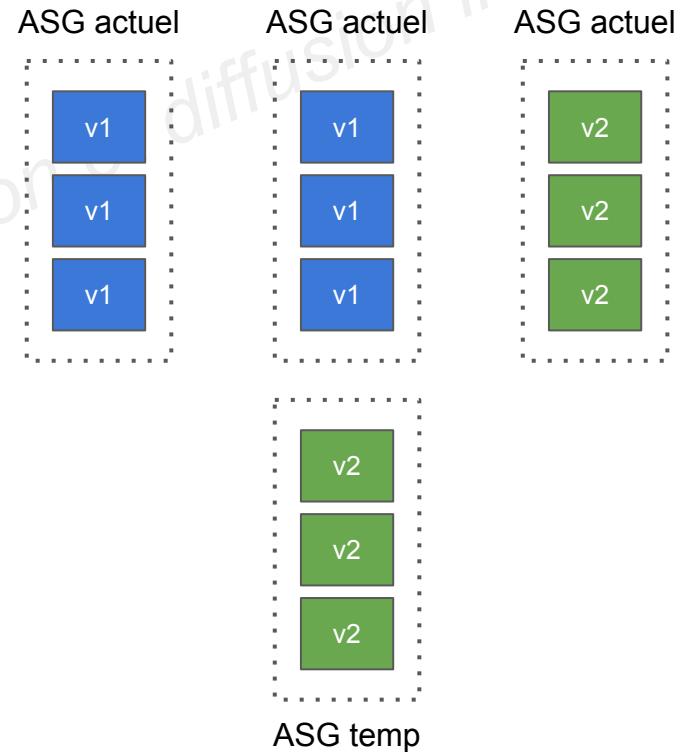
# Elastic Beanstalk - déploiement - Rolling with additional batches

- L'application tourne à régime normal
- Deux versions en simultané
- Long déploiement
- Petit coût additionnel
- Bon pour prod



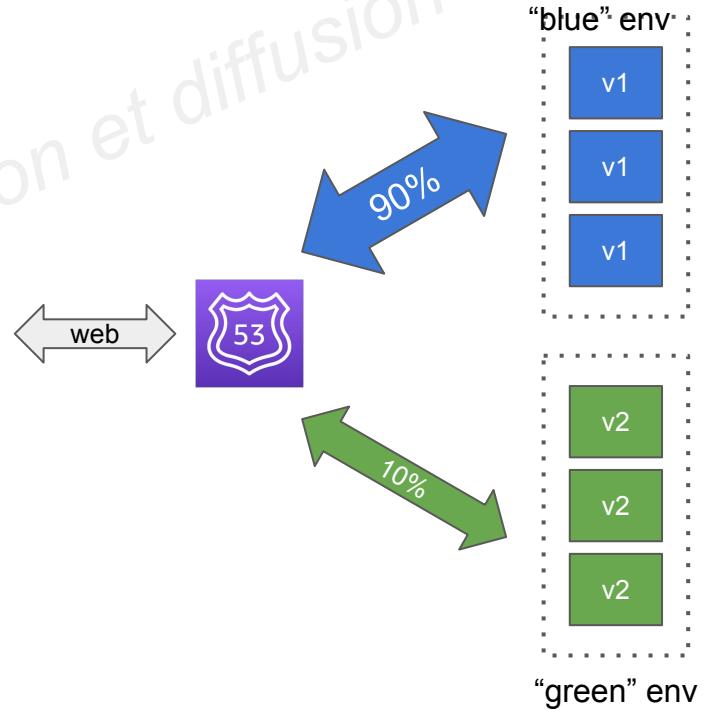
# Elastic Beanstalk - déploiement - Immutable

- Aucune interruption
- Nouveau code déployé dans de nouvelles instances à l'intérieur dans ASG temporaire
- Long déploiement
- Rapide rollback en cas d'échec (supprime le nouvel ASG)
- Bon pour prod
- Coût plus élevé (capacité doublée)



# Elastic Beanstalk - déploiement - Blue / Green

- Pas une fonctionnalité à proprement parler, juste une “technique”
- Aucune interruption
- Créer un nouvel environnement, y déployer le nouveau code
- Le nouvel env. peut être validé indépendamment et faire l'objet d'un rollback en cas de problème
- Route 53 peut être paramétré pour rediriger des portions du traffic sur le nouvel environnement
- Dans EB utiliser “swap URLs”



# Elastic Beanstalk - déploiement - résumé

Deployment methods		<a href="https://docs.aws.amazon.com/fr_fr/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html">https://docs.aws.amazon.com/fr_fr/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html</a>				
Method	Impact of failed deployment	Deploy time	Zero downtime	No DNS change	Rollback process	Code deployed to
All at once	Downtime	⊕	X	✓	Manual redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	⊕ ⊕ †	✓	✓	Manual redeploy	Existing instances
Rolling with an additional batch	Minimal if first batch fails; otherwise, similar to Rolling	⊕ ⊕ ⊕ †	✓	✓	Manual redeploy	New and existing instances
Immutable	Minimal	⊕ ⊕ ⊕ ⊖	✓	✓	Terminate new instances	New instances
Traffic splitting	Percentage of client traffic routed to new version temporarily impacted	⊕ ⊕ ⊕ ⊖ ††	✓	✓	Reroute traffic and terminate new instances	New instances
Blue/green	Minimal	⊕ ⊕ ⊕ ⊖	✓	X	Swap URL	New instances

# Elastic Beanstalk - CLI

- Il existe un CLI spécifique pour EB
- Commandes de base:
  - eb create
  - eb status
  - eb health
  - eb events
  - eb logs
  - eb open
  - eb deploy
  - eb config
  - eb terminate
- Utile pour automatiser des pipelines de déploiement

# Elastic Beanstalk - processus de déploiement

- Décrire les dépendances
  - requirements.txt pour Python, composer.json pour PHP, package.json pour Nodejs
- Packaging du code dans un fichier zip
- Console: upload du fichier (création d'une nouvelle version) et déploiement
- CLI: création d'une nouvelle version et déploiement
- EB déploie le fichier zip dans chaque instance EC2, résout les dépendances et démarre l'application

# Elastic Beanstalk - Politique de cycle de vie

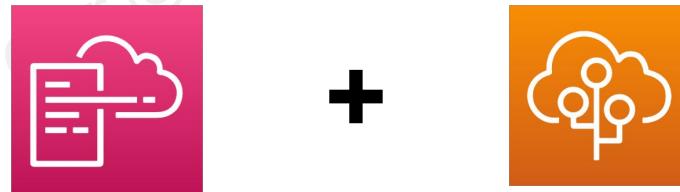
- EB peut stocker au plus 1000 versions d'une application
- Suppression d'anciennes version pour permettre un redéploiement
- Suppression automatisable par une **lifecycle policy**
  - basée sur le temps
  - basée sur l'espace
- Les version en cours d'utilisation ne seront pas supprimées
- La suppression du bucket source en S3 peut être protégée

# Elastic Beanstalk - Extensions

- Le code doit être déployé vers EB sous forme d'un fichier zip
- Tous les paramètres UI peuvent être configurés via des fichiers
- Pré-requis
  - fichiers à placer dans un dossier .ebextensions à la racine du code source
  - format YAML/JSON
  - extensions **.config** (ex: logging.config)
  - possibilité de modifier des paramètres par défaut, via option\_settings
  - possibilité d'ajouter des ressources telles que RDS, ElastiCache, DynamoDB, etc.
- Les ressources gérées par .ebextensions sont supprimées avec l'environnement

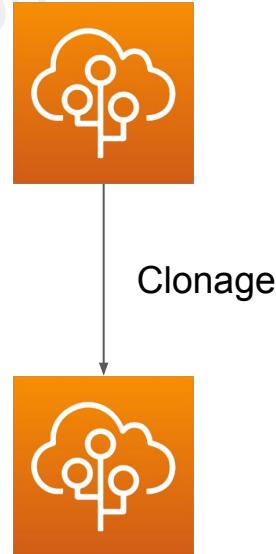
# Elastic Beanstalk - en coulisse

- En coulisse, EB s'appuie sur CloudFormation
- CloudFormation est utilisé pour approvisionner d'autres services AWS
- Cas d'utilisation: définir des ressources CloudFormation dans .ebextensions afin de fournir ElastiCache, un bucket S3, etc.



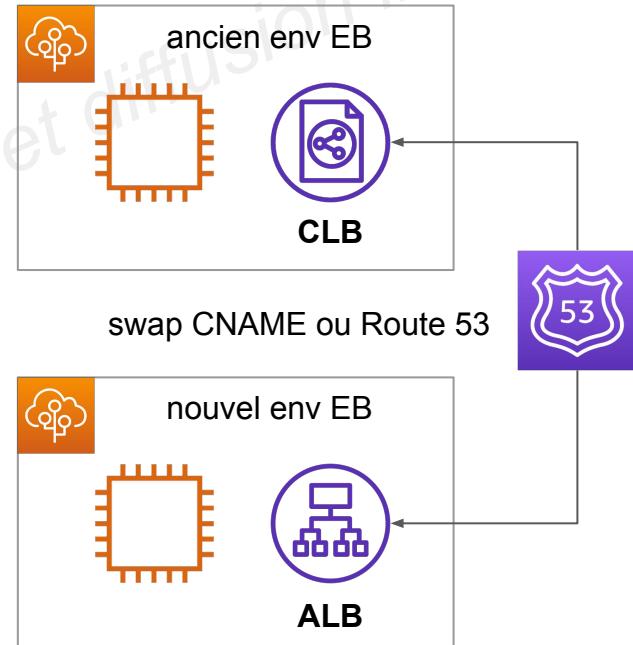
# Elastic Beanstalk - clonage

- EB permet de cloner un environnement avec une réplique exacte de sa configuration
- Utile pour déployer un environnement “test” de l’application
- Toutes les ressources et configuration sont préservées
  - Type de Load Balancer et configuration
  - Type de base de données RDS (mais les données ne sont pas clonées)
  - Variables d’environnement
- Les paramètres d’un environnement cloné peuvent être modifiées indépendamment de l’original



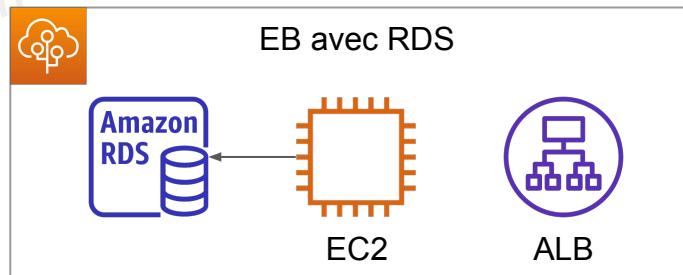
# Elastic Beanstalk - migrer un Load Balancer

- Il n'est pas possible de changer le type de **ELB** (juste la configuration) après une création d'environnement
- Pour migrer
  1. créer un nouvel env avec la même configuration, sauf pour le LB (qui ne peut pas être cloné)
  2. déployer l'application dans le nouvel env
  3. faire un swap CNAME ou mise à jour dans Route 53



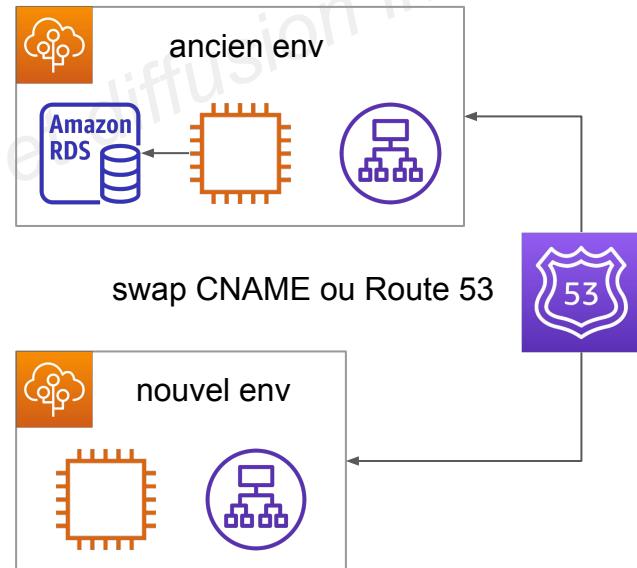
# Elastic Beanstalk - RDS

- RDS peut être approvisionné avec EB
- Bon pour dev/test
- Pas bon pour prod car le cycle de vie de la base de données est lié à celui de l'environnement Beanstalk
- La meilleure approche pour prod est de créer une base RDS séparée et de fournir la chaîne de connexion à l'application EB



# Elastic Beanstalk - migrer/découpler RDS

- Créer un snapshot de la base RDS
- Dans la console RDS, protéger la base de données contre la suppression
- Créer un nouvel environnement EB, sans RDS intégré ; faire pointer l'app vers l'instance RDS
- Faire un swap CNAME (bleu/green) ou mise à jour Route 53
- Supprimer l'ancien env
- Supprimer la pile (stack) CloudFormation (DELETE\_FAILED state)



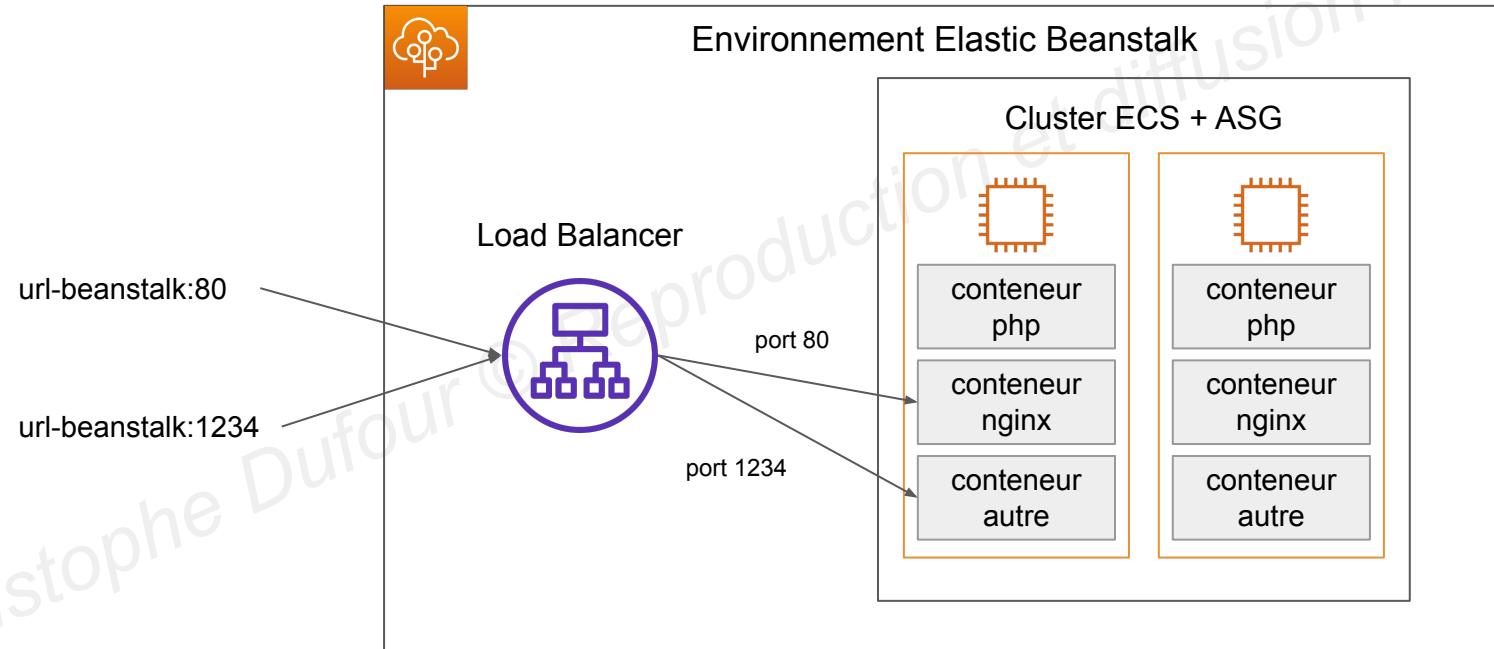
# Elastic Beanstalk - conteneur Docker unique

- Exécuter l'application en tant que **Single Docker Container**
- Fournir au choix
  - Dockerfile: EB construira et exécutera le conteneur Docker
  - Dockerrun.aws.json (v1): décrit où l'image Docker déjà constituée se situe:
    - Image
    - Ports
    - Volumes
    - Logging
    - Etc.
- Single Docker Container de Beanstalk **n'utilise pas ECS**

# Elastic Beanstalk - conteneur Docker multiple

- Exécuter l'application en tant que **Multiple Docker Container**
- Cela produit les ressources suivantes:
  - ECS Cluster
  - EC2 instances, configurées pour utiliser le cluster ECS
  - Load Balancer (en mode haute disponibilité)
  - Task definitions et exécution
- Requiert un fichier **Dockerrun.aws.json (v2)** à la racine du code
- Dockerrun.aws.json est utilisé pour générer l'**ECS task definition**
- Nos images Docker doivent pré-construire et stocker dans ECR par exemple

# Elastic Beanstalk - Multi Docker ECS

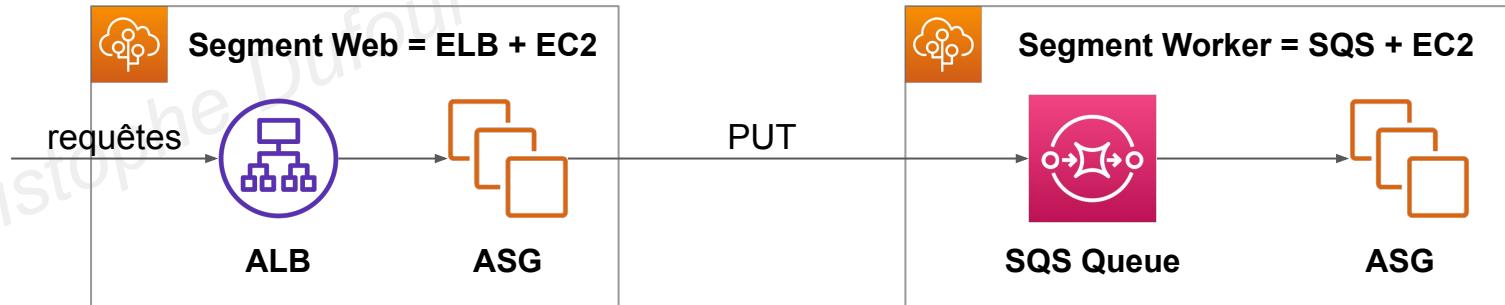


# Elastic Beanstalk - HTTPS

- Beanstalk avec HTTPS
  - charger le certificat SSL au niveau du LB
  - peut se faire depuis la console (EB console, LB config)
  - peut se faire par le code (.ebextensions/securelistener-alb.config)
  - le certif SSL peut être approvisionné via ACM ou CLI
  - un SG doit être configuré pour accepter les requêtes sur le port 443
- Beanstalk redirect HTTP to HTTPS
  - configurer les instances pour la redirection:  
<https://github.com/awsdocs/elastic-beanstalk-samples/tree/master/configuration-files/aws-provided/security-configuration/https-redirect>
  - OU: configurer ALB (uniquement) avec une règle
  - vérifier que les health checks ne sont pas redirigés (afin de toujours renvoyer 200 OK)

# Web Server vs Worker Environment

- Si votre app exécute de longues tâches, placer les dans un **worker environment** dédié
- Il est commun de découper une application en deux segments
- Ex: traiter une vidéo, générer un fichier zip, etc.
- Possibilité de définir des tâches périodiques dans un fichier **cron.yaml**



# Elastic Beanstalk - Plateforme personnalisée

- Sujet de niveau avancé
- Permet de définir ex nihilo
  - système d'exploitation
  - logiciels additionnels
  - scripts que EB exécutent sur ces plateformes
- Cas d'utilisation: app incompatible avec EB et n'utilisant pas Docker
- Pour créer sa propre plateforme
  - définir une AMI via le fichier **Platform.yaml**
  - construire la plateforme via **Packer** software (**outil open source pour créer des AMIs**)
- Custom Platform vs Custom Image (AMI)
  - Custom Image: se branche à une plateforme EB **existante** (Python, Java, etc.)
  - Custom Platform: crée une **nouvelle plateforme** EB

# AWS CICD

CodeCommit, CodePipeline, CodeBuild, CodeDeploy

# AWS CloudFormation

Gérer votre infrastructure comme du code

# Infrastructure as a Code (IaaC)

- Jusqu'à présent nous avons effectué beaucoup de tâches manuelles
- Tout ce travail manuel est très long à reproduire
  - dans une autre région
  - dans un autre compte AWS
  - dans le même région, après suppression
- Ne serait-ce pas merveilleux si la totalité de l'infrastrucurtre était du...code ?
- Ce code executé pourrait créer/modifier/supprimer notre infrastructure



# CloudFormation - intro

- CF propose une façon déclarative de constuire une infrastructure AWS, incluant tout type de ressource (la plupart sont supportées)
- Par exemple, un template CF permet de déclarer
  - je veux un Security Group
  - je veux 2 instances EC2 utilisant ce SG
  - je veux une Ip élastique pour ces instances
  - je veux un bucket S3
  - je veux un Load Balancer devant ces instances
- CF crée pour nous toutes ces ressources dans l'**ordre approprié** en suivant **précisant la configuration** indiquée

# CloudFormation - avantages (1/2)

- Infrastructure as code
  - pas de ressource manuellement créée, parfait pour le contrôle
  - code versionnable
  - changements visibles dans le code
- Coût
  - chaque ressource dans la pile (stack) est taguée avec un identifiant permettant d'évaluer facilement le coût de la pile
  - on peut estimer le coûts des ressources par le template CloudFormation
  - stratégie économique: en Dev, on peut automatiser la suppression d'un template à 17h et le recréer à 8h

# CloudFormation - avantages (2/2)

- Productivité
  - capacité à détruire et recréer une infrastructure dans le cloud, à la demande
  - production automatique de diagrammes des templates
  - programmation déclarative (pas besoin de déterminer l'ordre et l'orchestration)
- Séparation des problèmes: différentes piles
  - piles VPC
  - piles Réseaux
  - piles applicatives
- Pas de réinvention de la roue
  - utilisation de templates existants le web
  - utilisation de la documentation

# CloudFormation - comment ça marche ?

- Les templates doivent être uploadés dans S3 et référencés dans CloudFormation
- Pour mettre un jour un template, nous ne pouvons pas éditer la version précédente, il faut uploader la nouvelle version dans AWS
- Les piles sont identifiées par un nom
- Supprimer une pile supprime chaque artefact (ressource) individuel ayant été créé par CloudFormation

# CloudFormation - déploiement de template

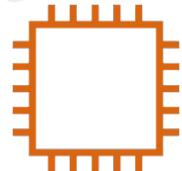
- Façon manuelle
  - éditer les templates dans le CloudFormation Designer
  - utiliser la console pour saisir les paramètres, etc.
- Façon automatisée (recommandée)
  - éditer les les templates dans un fichier YAML
  - utiliser AWS CLI pour déployer les templates

# CloudFormation - composants de base

- Templates components
  1. **Resources (obligatoire): ressources AWS déclarées dans le template**
  2. Parameters: les entrées dynamiques du template
  3. Mappings: les variables statiques du template
  4. Outputs: références aux ressources créées
  5. Conditionals: liste de conditions à respecter pour la création de ressources
  6. Metadata
- Templates helpers
  1. References
  2. Functions

# CloudFormation - atelier

- Nous allons créer une simple instance EC2
- Nous allons créer une IP élastique et l'attacher à l'instance
- Nous attacherons deux Security Groups à l'instance



# CloudFormation - le format YAML en bref

- CF permet l'utilisation des langages YAML et JSON
- JSON n'est pas très adapté (lisible) pour CF
- YAML est plus avantageux
- Principes de base
  - paires clé/valeur
  - objets imbriqués
  - support des tableaux
  - chaînes multi-lignes
  - commentaire possibles
  - structure par indentation

# CloudFormation - parameters

- les Parameters permettent de fournir des entrées au template CF
- Il est important de les connaître si
  - on souhaite les templates
  - certaines entrées ne peuvent pas être connues à l'avance
- les Parameters sont extrêmement puissants, ils offrent un excellent contrôle et peuvent prévenir des erreurs grâce aux types

# CloudFormation - quand utiliser un paramètre ?

- Il faut se poser la question
  - cette configuration de ressource est-elle susceptible de changer dans le futur ?
  - si oui, l'emploi d'un paramètre s'impose
- Pas besoin de re-uploader un template pour changer son contenu ;-)

```
Parameters:
```

```
  SecurityGroupDescription:
```

```
    Description: Security Group Description
```

```
    Type: String
```

# CloudFormation - Parameters Settings

Les parameters peuvent être contrôlés par tous ces éléments:

- Type
  - String
  - Number
  - CommaDelimitedList
  - List<Type>
  - AWS Parameter (aide à détecter les valeurs incorrectes)
- Description
- Constraints
  - ConstraintDescription (String)
  - Min/MaxLength
  - Min/MaxValue
  - Defaults
  - AllowedValues (array)
  - AllowedPattern (regexp)
  - NoEcho (Boolean)

# CloudFormation - référencer un paramètre

- Les Parameters peuvent être utilisés n'importe où dans un template
- La fonction `Fn::Ref` permet de référencer des paramètres
- Dans YAML cette fonction a pour raccourci `!Ref`
- La fonction peut aussi référence d'autres éléments dans le template

```
DbSubnet1:  
  Type: AWS::EC2::Subnet  
  Properties:  
    VpcId: !Ref MyVPC
```

# CloudFormation - Pseudo Parameters

- AWS offre des pseudo paramètres pour chaque template CloudFormation
- Ils peuvent être utilisés n'importe quand et sont activés par défaut

Pseudo paramètre	Exemple de valeur renournée
AWS::AccountId	123456789
AWS::NotificationARNs	[arn:aws:sns:us-east-1:123456789:MyTopic]
AWS::NoValue	
AWS::Region	us-east-2
AWS::StackName	MyStack

# CloudFormation - Mappings

- Les Mappings sont des variables fixes dans les template CloudFormation
- Très pratiques pour différencier les environnements (dev/prod), les régions, les types d'AMI, etc.
- Toutes les valeurs sont “codées en dur” dans les templates

```
Mappings:  
  Mapping01:  
    Key01:  
      Name: Value01  
    Key02:  
      Name: Value02  
    Key03:  
      Name: Value03
```

```
RegionMap:  
  us-east-1:  
    "32": "ami-6411e20d"  
    "64": "ami-7a11e213"  
  us-west-1:  
    "32": "ami-c9c7978c"  
    "64": "ami-cfc7978a"  
  eu-east-1:  
    "32": "ami-37c2f643"  
    "64": "ami-31c2f645"
```

# CloudFormation - mappings vs parameters

- Les Mappings sont un bon choix quand les valeurs sont connues à l'avance ou peuvent être déduite de variables telles que
  - région
  - az
  - compte aws
  - env (dev/prod)
  - etc.
- Les Mappings offrent un contrôle plus sécurité sur le template
- Utiliser les Parameters quand les valeurs sont spécifiques à l'utilisateur

# CloudFormation - accès aux valeurs des Mappings

- La fonction `Fn::FindInMap` retourne une valeur nommée à partir d'une clé
- `!FindInMap [ MapName, TopLevelKey, SecondLevelKey ]`

```
Resources:  
  myEC2Instance:  
    Type: "AWS::EC2::Instance"  
    Properties:  
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]  
      InstanceType: m1.small
```

# CloudFormation - Outputs

- La section des Outputs déclare des valeurs de sorties optionnelles pouvant être importées dans d'autres piles (il faut d'abord les exporter)
- Les outputs sont visibles dans la console AWS ou via AWS CLI
- Très utiles, par exemple, si l'on définit un réseau CF et qu'on souhaite exporter les variables telles que le VPC ID et les Subnet IDs
- C'est la meilleure façon de faire des piles collaboratives, en laissant à chacun le soin de gérer sa propre partie de la pile
- On ne peut pas supprimer une pile CF si ses outputs sont référencés par une autre pile CF

# CloudFormation - exemple

- Création d'un groupe de sécurité SSH
- Création d'un output référençant ce SG

```
Outputs:
```

```
StackSSHSecurityGroup:
```

```
Description: Le groupe de sécurité SSH de mon entreprise
```

```
Value: !Ref MyCompanyWideSSHSecurityGroup
```

```
Export:
```

```
Name: SSHSecurityGroup
```

# CloudFormation - Cross Stack Reference

- Nous pouvons créer un second template faisant usage du SG
- Pour ce faire, nous utilisons la fonction `Fn::ImportValue`
- On ne peut pas supprimer la pile référencée tant que les références ne sont pas elles-mêmes supprimées

```
Resources:  
  MySecureInstance:  
    Type: AWS::EC2::Instance  
    Properties:  
      AvailabilityZone: us-east-1a  
      ImageId: ami-a4c7edb2  
      InstanceType: t2.micro  
      SecurityGroups:  
        - !ImportValue SSHSecurityGroup
```

# CloudFormation - Conditions

- Les conditions permettent la création de ressources sur la base de conditions
- Elles peuvent par exemple porter sur
  - environnement (dev/test/prod)
  - région AWS
  - toute valeur de paramètre
- Chaque condition peut référencer une autre condition, valeur de paramètre ou mapping

# CloudFormation - définir une condition

- L'id logique est à choisir par l'utilisateur, c'est le nom de la condition
- Les fonctions “intrinsic” (logiques) peuvent être
  - Fn::And
  - Fn::Equals
  - Fn::If
  - Fn::Not
  - Fn::Or

Conditions:

```
CreateProdResources: !Equals [ !Ref EnvType, prod ]
```

# CloudFormation - Fn::FindInMap

- Les conditions peuvent s'appliquer à une ressource/output/etc.

```
Resources:  
  MountPoint:  
    Type: "AWS::EC2::VolumeAttachment"  
    Condition: CreateProdResources
```

# CloudFormation - Intrinsic Functions à connaître

- Fn::Ref
- Fn::GetAtt
- Fn::FindInMap
- Fn::ImportValue
- Fn::Join
- Fn::Sub
- Conditions Functions (Fn::If, Fn::Not, Fn::Equals, etc.)

# CloudFormation - Fn::Ref

- La fonction **Fn::Ref** permet de référence
  - des Parameters => retourne la valeur du paramètre
  - des Resources => retourne l'ID de la ressource sous-jacente (ex: EC2 ID)
- Raccourci YAML: **!Ref**

```
DBSubnet1:  
  Type: AWS::EC2::Subnet  
  Properties:  
    VpcId: !Ref MyVPC
```

# CloudFormation - Fn::GetAtt

- Des attributs sont attachés à toute ressource créée
- Pour connaître les attributs d'une ressource, le mieux est de consulter la documentation
- Exemple: l'AZ d'une machine EC2

```
Resources:  
  EC2Instance:  
    Type: AWS::EC2::Instance  
    Properties:  
      ImageId: ami-1234567  
      InstanceType: t2.micro
```

```
NewVolume:  
  Type: AWS::EC2::Volume  
  Condition: CreateProdResources  
  Properties:  
    Size: 100  
    AvailabilityZone:  
      !GetAttr EC2Instance.AvailabilityZone
```

# CloudFormation - Fn::FindInMap

- La fonction `Fn::FindInMap` retourne une valeur nommée à partir d'une clé
- `!FindInMap [ MapName, TopLevelKey, SecondLevelKey ]`

```
Resources:  
  myEC2Instance:  
    Type: "AWS::EC2::Instance"  
    Properties:  
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]  
      InstanceType: m1.small
```

# CloudFormation - Fn::ImportValue

- La fonction `Fn::ImportValue` importe des valeurs exportées par d'autres templates
- Raccourci YAML: `!ImportValue`

```
Resources:  
  MySecureInstance:  
    Type: AWS::EC2::Instance  
    Properties:  
      AvailabilityZone: us-east-1a  
      ImageId: ami-a4c7edb2  
      InstanceType: t2.micro  
      SecurityGroups:  
        - !ImportValue SSHSecurityGroup
```

# CloudFormation - Fn::Join

- Joint des valeurs avec un délimiteur

```
!Join [ delimiter, [ liste de valeurs séparées par une virgule ]]
```

- Retourne "a:b:c"

```
!Join [ ":", [a,b,c]]
```

# CloudFormation - Fn::Sub

- Fn::Sub ou le raccourci !Sub permet de remplacer des variables dans un texte. C'est une fonction très utilisée pour personnaliser un template
- On peut par exemple combiner Fn::Sub avec des References ou des Pseudo variables AWS
- String doit contenir \${VariableName}

## !Sub

- String
- { Var1Name: Var1Value, Var2Name: Var2Value }

# CloudFormation - Conditions

- L'id logique est à choisir par l'utilisateur, c'est le nom de la condition
- Les fonctions “intrinsic” (logiques) peuvent être
  - Fn::And
  - Fn::Equals
  - Fn::If
  - Fn::Not
  - Fn::Or

Conditions:

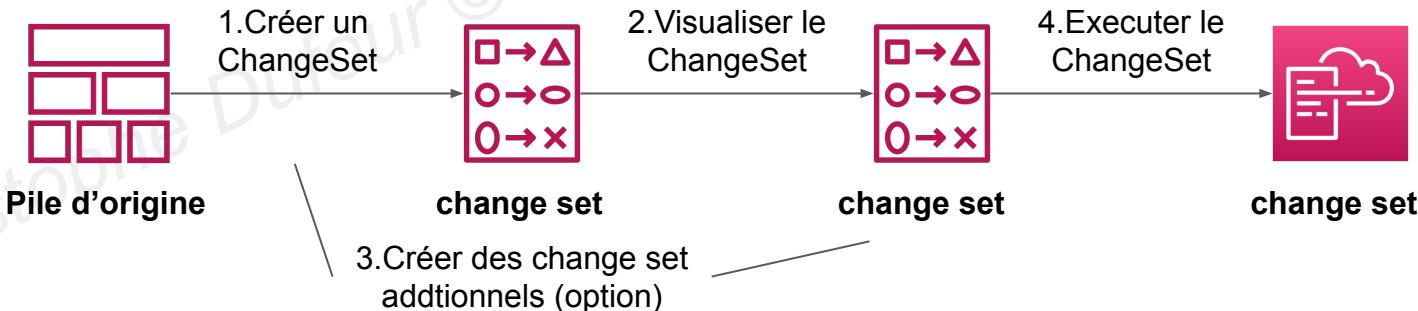
```
CreateProdResources: !Equals [ !Ref EnvType, prod ]
```

# CloudFormation - Rollbacks

- Echec à la création de la pile
  - toutes les ressources sont détruites. On peut inspecter les logs
  - possibilité de désactiver le rollback et de réparer
- Echec à la modification de la pile
  - la pile revient immédiatement à la précédente version en état de fonctionnement
  - possibilité de regarder les logs pour analyser les message d'erreur

# CloudFormation - ChangeSets

- Echec à la création de la pile
  - toutes les ressources sont détruites. On peut inspecter les logs
  - possibilité de désactiver le rollback et de réparer
- Echec à la modification de la pile
  - la pile revient immédiatement à la précédente version en état de fonctionnement
  - possibilité de regarder les logs pour analyser les messages d'erreur

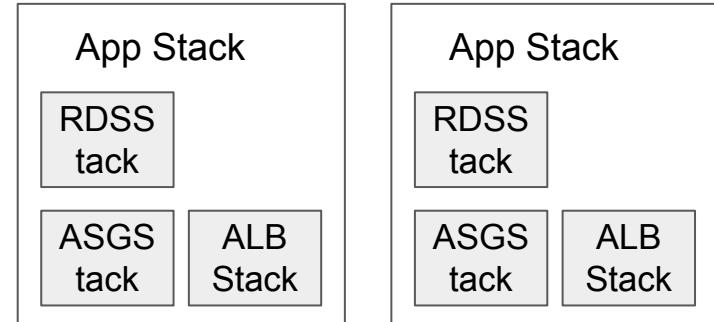
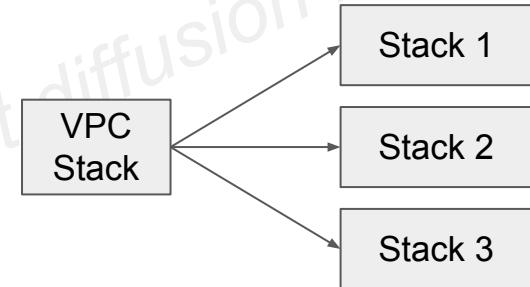


# CloudFormation - Piles imbriquées (nested stacks)

- Piles faisant partie d'autres piles
- Permettent d'insérer des motifs/composant récurrent dans des piles à part et les appeler dans d'autre piles
- Exemple
  - configuration réutilisable de Load Balancer
  - groupe de sécurité réutilisable
- Considérées comme une bonne pratique
- Pour mettre à jour une pile imbriquée, toujours mettre à jour le parent (root stack)

# CloudFormation - Cross vs Nested Stacks

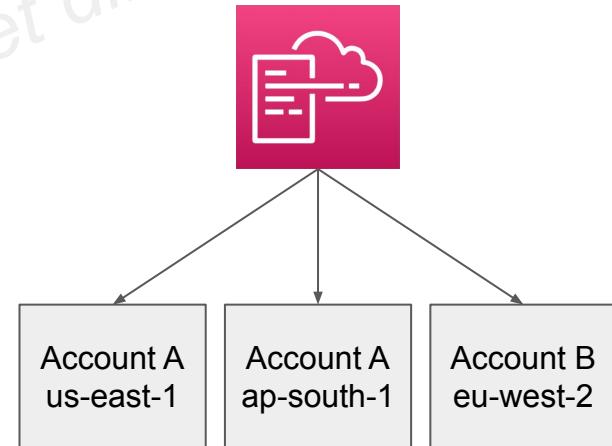
- Cross Stacks
  - utile quand les piles ont des cycles de vie différents
  - utiliser l'export d'Outputs et Fn::ImportValue
  - utile quand il est nécessaire d'exporter des valeurs vers de nombreuses piles (VPC id, etc.)
- Nested Stacks
  - utile quand les composants doivent être réutilisés
  - ex: réutiliser une configuration de Load Balancer



# CloudFormation - StackSets

- Créer, modifier ou supprimer des piles à travers de **multiples comptes et régions** en une seule opération
- Il faut un compte administrateur pour créer des StackSets
- Les comptes autorisés peuvent créer/modifier/supprimer des instances de pile depuis les StackSets
- Quand un StackSet est modifié, toutes les instances de stack associées sont mise à jour à travers tous les comptes et toutes les régions

**StackSet** CloudFormation  
Compte admin



# AWS Monitoring & Audit

CloudWatch, X-Ray et CloudTrail

# Pourquoi le monitoring est-il important ?

- Nous savons déployer nos applications
  - de façon sécurisée
  - de façon automatique
  - via l'Infrastructure as Code
  - via les services AWS appropriés
- Nos utilisateurs se moquent de quelle façon nous avons déployé
- Nos utilisateurs veulent que l'application fonctionne
  - la latence augmentera-t-elle dans le temps ?
  - quid des pannes ?
  - devoir contacter le service IT n'est pas souhaitable
  - dépannage et solution
- Monitoring interne
  - peut-on prévenir les problèmes ?
  - coût et performance
  - tendances
  - connaître et améliorer

# Monitoring dans AWS

- AWS CloudWatch
  - **Metrics**: collecte et surveille les métriques clés
  - **Logs**: collecte, surveille, analyse et stocke des fichiers des logs
  - **Events**: émet des notifications quand certains événements se produisent
  - **Alarms**: réagit en temps réel à des métriques/événements
- AWS X-Ray
  - dépanne les applications en cas d'erreurs et de problèmes de performance
  - traçage distribué de micro-services
- AWS CloudTrail
  - monitoring interne des appels API effectués
  - audite les modifications de ressources AWS faites par nos utilisateurs



# AWS CloudWatch Metrics

- CloudWatch fournit de métriques pour tous les services AWS
- **Métrique:** variable à surveiller (utilisation CPU, Traffic réseau, etc.)
- Une métrique appartient à un **espace de nom**
- **Dimension:** attribut d'une métrique (instance id, environment, etc...)
- Jusqu'à 10 dimensions par métrique
- Les métriques ont un **horodatage**
- On peut créer des tableaux de bord de métriques

# AWS CloudWatch monitoring EC2

- Les instances EC2 ont des métriques toutes les cinq minutes
- Avec un monitoring détaillé (payant) l'intervalle passe à une minute
- Utiliser le monitoring détaillé pour un déclenchement plus rapide de l'ASG
- Le Free Tier permet d'obtenir un monitoring détaillé de 10 métriques
- N.b: l'utilisation RAM n'est par défaut pas prise en compte (elle doit être communiquée depuis l'instance en tant que métrique personnalisée)

# AWS CloudWatch Alarms



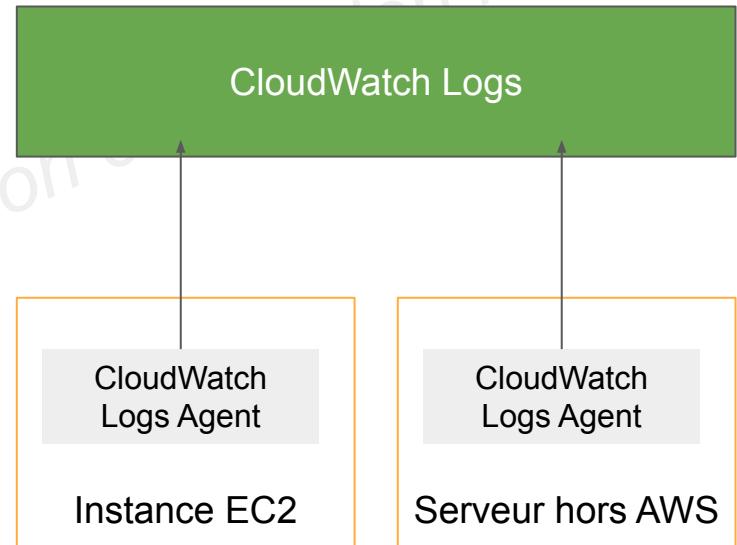
- Les alarmes sont utilisées pour déclencher des notifications relatives à n'importe quelle métrique
- Elles se marient avec l'Auto Scaling, les actions EC2, les notifications SNS
- Diverses options (sampling, %, max, mon, etc.)
- Status d'alarme
  - OK
  - INSUFFICIENT\_DATA
  - ALARM
- Périodicité
  - durée en secondes d'évaluation de la métrique
  - métrique personnalisée haute résolution: 10 ou 30 sec sont les seules valeurs possibles

# AWS CloudWatch Logs

- CloudWatch Logs peut utiliser des filtres
- Architecture du stockage des logs
  - Log groups: nom arbitraire, représente généralement une application
  - Log stream: instances d'une application, fichiers de log, conteneurs
- On peut définir une politique d'expiration (jamais, 30 jours, etc.)
- AWS CLI permet de tronquer (tail) les logs CloudWatch
- S'assurer d'avoir les droits IAM requis pour envoyer les logs à CloudWatch
- Logs chiffrables via KMS au niveau du groupe

# CloudWatch Logs pour EC2

- Par défaut, aucun log provenant d'une machine EC2 ne va vers CloudWatch
- Il est nécessaire d'exécuter un agent CloudWatch dans la machine EC2
- S'assurer d'avoir les droits IAM
- L'agent CloudWatch peut également être utilisé hors AWS



# CloudWatch Logs - agent et agent unifié

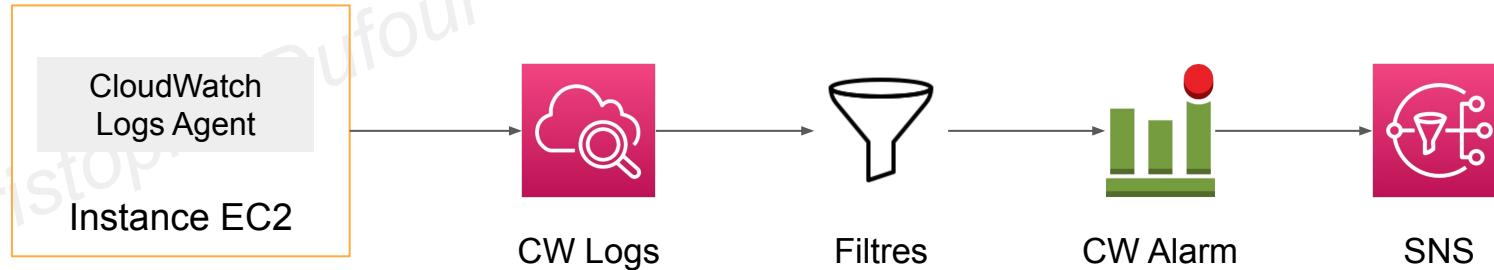
- Pour serveurs virtuels (instances EC2, on-premise, etc.)
- **CloudWatch Logs Agent**
  - ancienne version de l'agent
  - peut seulement envoyer des données vers CloudWatch Logs
- **CloudWatch Unified Agent**
  - collecte des métriques systèmes supplémentaires telles que la RAM, le processus, etc.
  - collecte les logs à envoyer à CloudWatch Logs
  - configuration centralisée via SSM Parameter Store

# CloudWatch Unified Agent - métriques

- Collectées directement depuis un serveur Linux / instance EC2
- **CPU** (active, guest, idle, system, user, steal)
- **Disk metrics** (free, used, total), Disk IO (writes, reads, bytes, iops)
- **RAM** (free, inactive, used, total, cached)
- **Netstat** (nombre de connexions TCP et UDP, paquets réseaux, octets)
- **Processus** (total, dead, bloqued, idle, running, sleep)
- **Swap Space** (free, used, used %)

# CloudWatch Logs - filtres de métriques

- CloudWatch Logs propose des filtres
  - exemple1: recherche d'une IP spécifique dans un log
  - exemple2: compte les nombre d'occurrences du mot "ERROR"
  - les filtres peuvent être utilisés par déclencher des alarmes
- Les filtres s'agissent pas de façon rétroactive. Ils produisent des données pour des événements ayant lieu après leur création



# CloudWatch Events

- Programmation: Cron jobs
- Event Pattern: règles d'événement auxquelles réagir
  - ex: changement d'état de CodePipeline
- Intégrés aux fonctions Lambda, messages SQS/SNS/Kinesis
- Un CW Event génère un petit document JSON relatif à l'événement



# Amazon EventBridge

- EventBridge représente l'évolution de CloudWatch Events
- **Default event bus:** utilisé par les services AWS
- **Partner event bus:** reçoit des événements depuis des services SaaS ou des applications (Zendesk, DataLog, Segment, Auth0, etc.)
- **Custom event buses:** pour nos propres applications
- Les canaux (buses) d'événement sont accessibles à d'autres comptes AWS
- **Règles:** comment traiter les événements (similaires aux CW Events)

# Amazon EventBridge Schema Registry

- EventBridge peut analyser les événements de notre canal (bus) et en inférer le schéma
- Le **Schema Registry** permet de générer du code pour notre application qui sera en avance comment les données sont structurées...
- Le schéma peut être versionné

The screenshot shows the AWS Schema Registry interface. At the top, it displays the schema ARN: `aws.codepipeline@CodePipelineActionExecutionStateChange`. Below this, the "Schema details" section provides the following information:

Schema name	Last modified	Schema ARN
<code>aws.codepipeline@CodePipelineActionExecutionStateChange</code>	Dec 1, 2019, 12:11 AM GMT	-
Description	Schema for event type <code>CodePipelineActionExecutionStateChange</code> , published by AWS service <code>aws.codepipeline</code>	Schema registry <code>aws.events</code> Number of versions 1 Schema type OpenAPI 3.0

Below the details, there is a section titled "Version 1 Created on Dec 1, 2019, 12:11 AM GMT". It includes a "Action" dropdown menu and a "Download code bindings" button. The "code bindings" section displays the following JSON code:

```
1 {
2   "openapi": "3.0.0",
3   "info": {
4     "version": "1.0.0",
5     "title": "CodePipelineActionExecutionStateChange"
6   },
7   "paths": {},
8   "components": {
9     "schemas": {
10       "AWSEvent": {
```

# EventBridge vs CloudWatch Events

- EventBridge se base sur CW Events et l'étend
- Il utilise la même API et la même infrastructure sous-jacente
- Peut être étendu par l'ajout des canaux d'événements pour nos applications propres ou pour des SaaS tierces
- Fonctionnalité de Schema Registry
- A terme, CW Events devrait être remplacé par EventBridge



# AWS X-Ray

- Façon traditionnelle de déboger en prod
  - tester localement
  - ajouter des instructions de log à différents endroits
  - redéployer
- Les formats de log changent d'une application à l'autres
- Le débogage des services distribués est réputé plus difficile que celui des services monolithiques
- Pas de vue uniforme sur la totalité de l'architecture
- AWS X-Ray tente d'apporter une solution

# AWS X-Ray - Analyses visuelles



# AWS X-Ray - avantages

- Dépanne les problèmes de performance
- Expose les dépendances d'une architecture micro-service
- Repère les problèmes de service
- Analyse le comportment des requêtes
- Recherche erreurs et exceptions
- Permet de répondre aux questions telles que
  - rencontre-ton un problème de latence ?
  - où se trouve le point d'engorgement ?
- Identifie les utilisateurs impactés

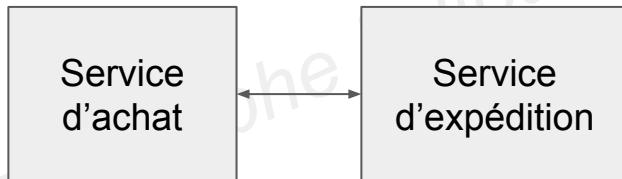
# AWS Intégration & Messaging

SQS, SNS et Kinesis

# Intégration & Messaging - intro

- Il arrive fréquemment que des applications doivent communiquer
- Cette communication peut s'exprimer à travers deux modèles
  - Communication synchrone
  - Communication asynchrone / événementielle

**Communication synchrone  
(app to app)**



**Communication asynchrone  
(app to queue to app)**



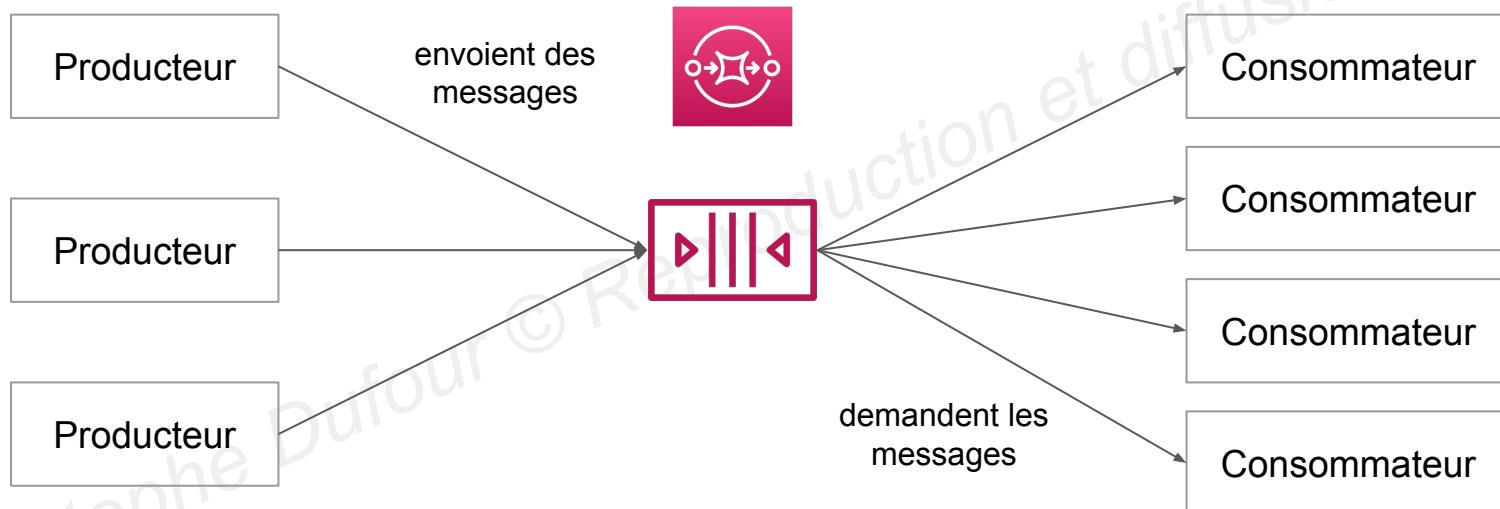
# Intégration & Messaging - intro

- Rester synchrone entre applications peut être problématique, notamment en cas de pics soudains de trafic
- Que faire si 1000 vidéos doivent être encodées au lieu de 10 (charge ordinaire) ?
- Dans ce cas, il vaut mieux **découpler** les applications
  - par SQS: modèle queue
  - par SNS: modèle pub/sub
  - par Kinesis: modèle flux (stream) temps réel
- Ces services peuvent être redimensionnés indépendamment des applications

# Intégration & Messaging - intro

- Rester synchrone entre applications peut être problématique, notamment en cas de pics soudains de trafic
- Que faire si 1000 vidéos doivent être encodées au lieu de 10 (charge ordinaire) ?
- Dans ce cas, il vaut mieux **découpler** les applications
  - par SQS: modèle queue
  - par SNS: modèle pub/sub
  - par Kinesis: modèle flux (stream) temps réel
- Ces services peuvent être redimensionnés indépendamment des applications

# Intégration & Messaging - SQS



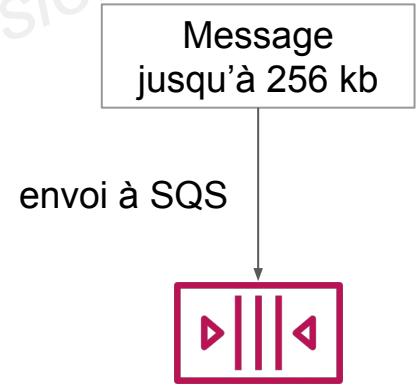


# SQS - file standard

- Offre la plus ancienne (plus de 10 ans)
- Service entièrement géré, utilisé pour découpler les applications
- Attributs
  - débit illimité, nombre de messages illimité dans les files
  - durée de rétention par défaut: 4 jours, max: 14
  - latence faible (< 10 ms à la publication et à la réception)
- Peut comporter des doublons
- Peut avoir des messages désordonnés (best effort ordering)

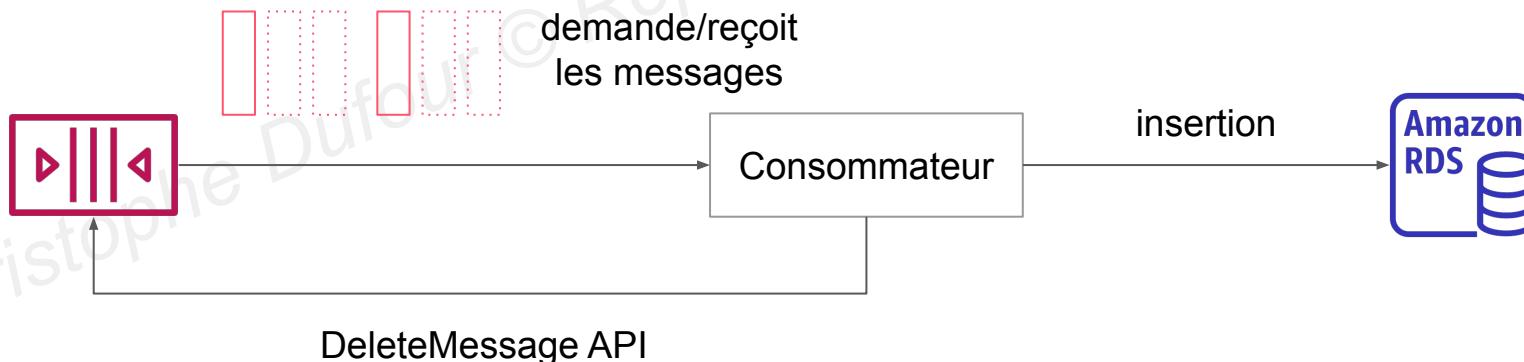
# SQS - produire des messages

- Production par le SDK (SendMessage API)
- Le message est persisté dans SQS jusqu'à ce qu'on consommateur le supprime
- Rétention du message: 4 jours par défaut, 14 jours au plus
- Example: envoie d'une commande devant être traitée
  - id de la commande
  - id du client
  - n'importe quelle donnée
- SQS Standard: débit illimité



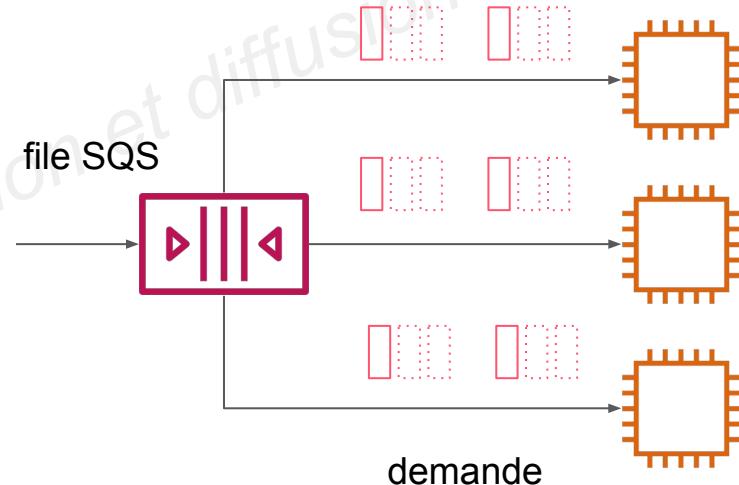
# SQS - consommer les messages

- Consommateurs: instances EC2, serveurs, fonctions Lambda, etc.
- Interrogent SQS (poll). Reçoivent jusqu'à 10 messages à la fois
- Traitent les messages. Ex: insérer le message dans une base RDS
- Supprimer les messages via l'API DeleteMessage

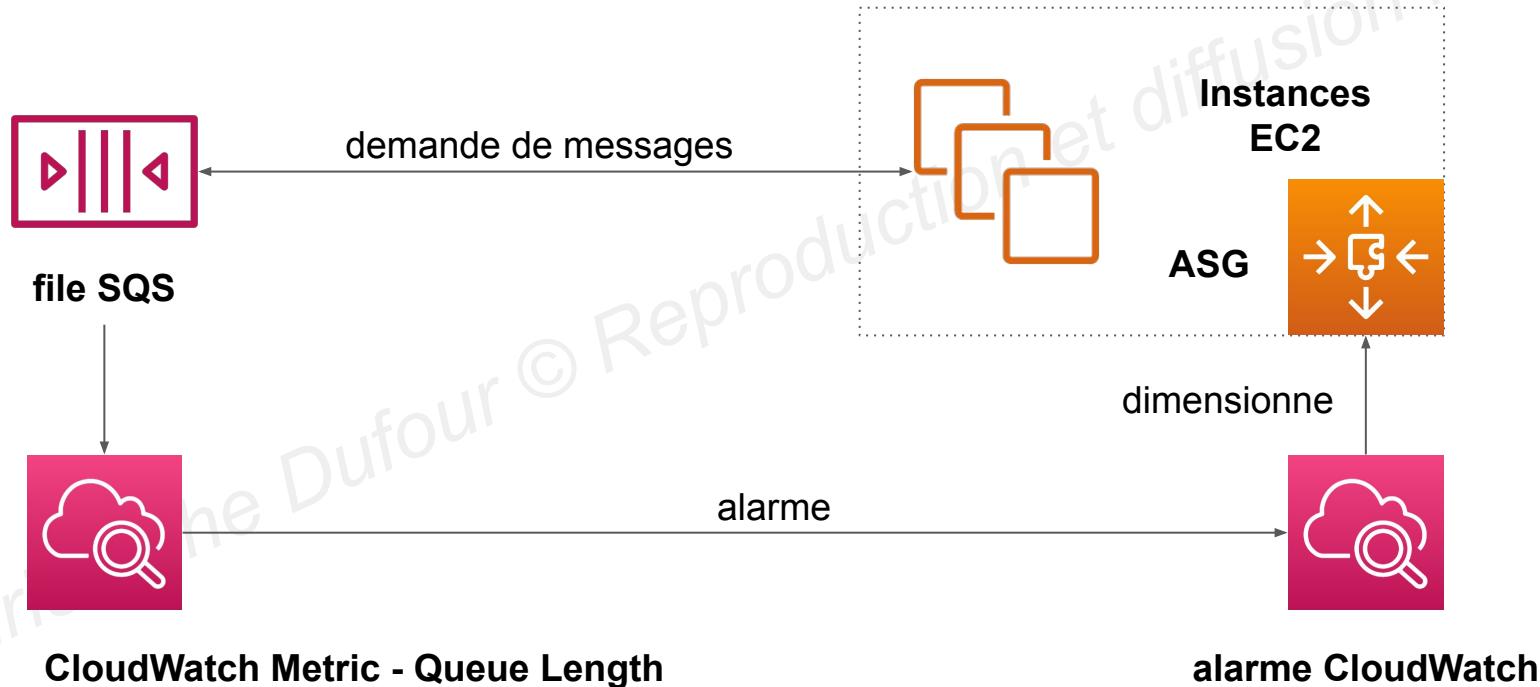


# SQS - Consommateurs: instances EC2

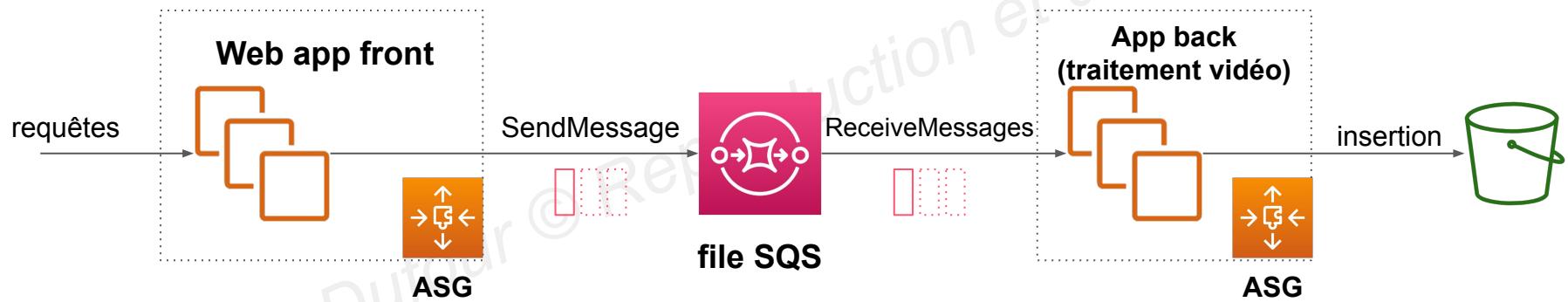
- Les consommateurs reçoivent et traitent les messages en parallèle
- Au moins une réception
- Best-effort message ordering
- Les consommateurs suppriment les messages après traitement
- Nous pouvons dimensionner les consommateurs horizontalement afin d'augmenter le débit



# SQS - Auto Scaling Group



# SQS - découplage entre applications

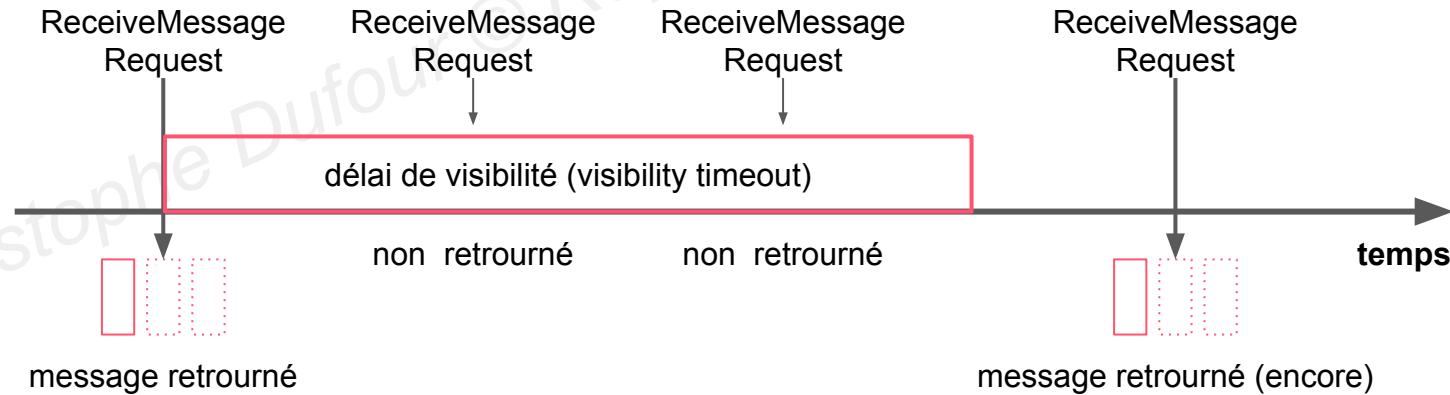


# SQS - sécurité

- Chiffrement
  - en transit via l'API HTTPS
  - au repos via KMS
  - côté client si le client souhaite chiffrer/déchiffrer lui-même
- Contrôles d'accès: stratégies IAM pour réguler l'accès à l'API SQS
- SQS Access Policies (similaires aux S3 Bucket policies)
  - utiles pour les accès cross-account aux files SQS
  - utiles pour autoriser d'autres services (SNS, S3,...) à écrire dans une file SQS

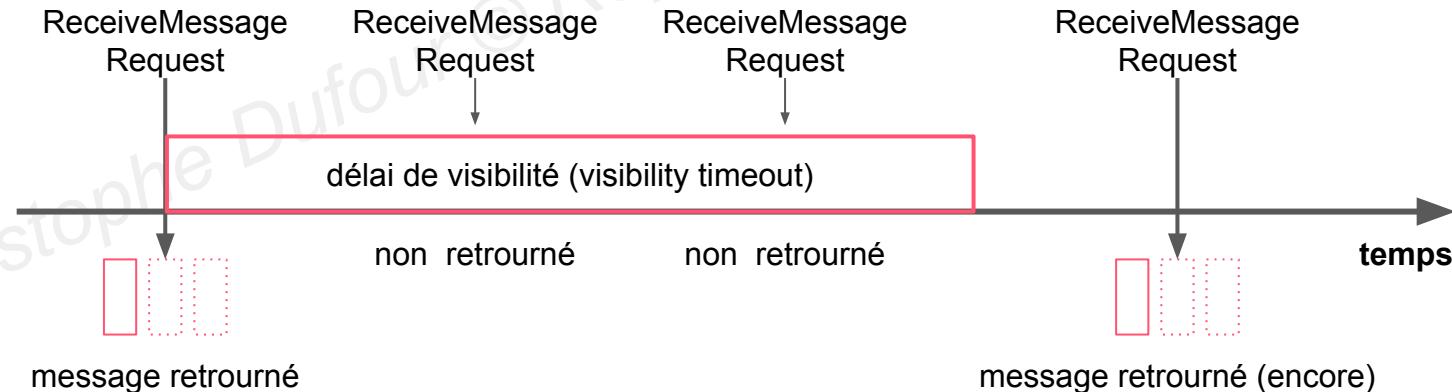
# SQS - délai de visibilité du message

- Après qu'un message est pris en compte par un consommateur, il devient **invisible** aux autres consommateurs
- Par défaut, ce temps d'invisibilité est de **30 secondes**
- Cela signifie que le message à 30 sec. pour être traité
- Une fois passé ce délai, le message redevient visible dans le file SQS



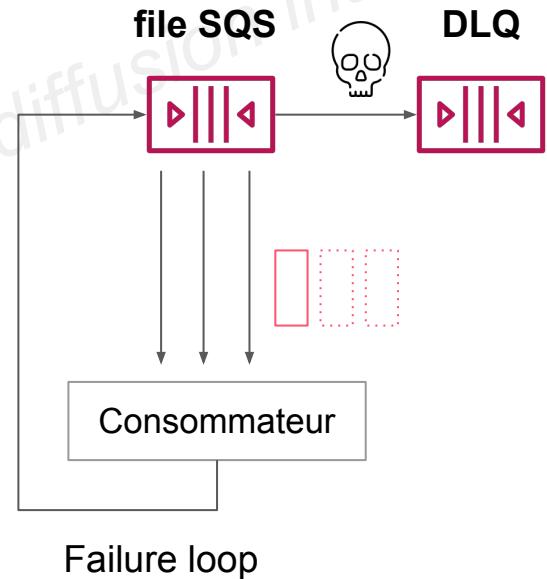
# SQS - délai de visibilité du message

- Si un message n'est pas traité durant ce délai, il sera traité une deuxième fois
- Un consommateur peut l'appeler l'API **ChangeMessageVisibility** pour obtenir plus de temps
- Si le délai est long (heures) et que le conso. échoue, le (re)traitement prendra du temps
- Si le délai est trop court (secondes), on peut obtenir des doublons



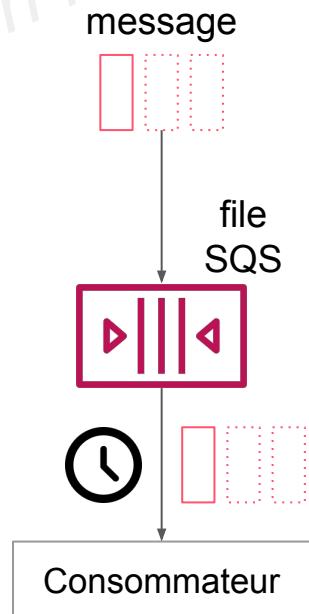
# SQS - Dead Letter Queue

- Si un consommateur échoue à traiter le message durant le délai de visibilité, le message retourne dans la file
- Il est possible de définir un seuil de nombre de fois qu'un message doit retourner dans la file
- Quand le seuil **MaximumReceives** est dépassé, le message va dans une dead letter queue (DLQ)
- Utilisé pour le débogage
- S'assurer de traiter les messages dans la DLQ avant leur expiration
  - il est bon de paramétriser à 14 jours le délai de rétention DLQ



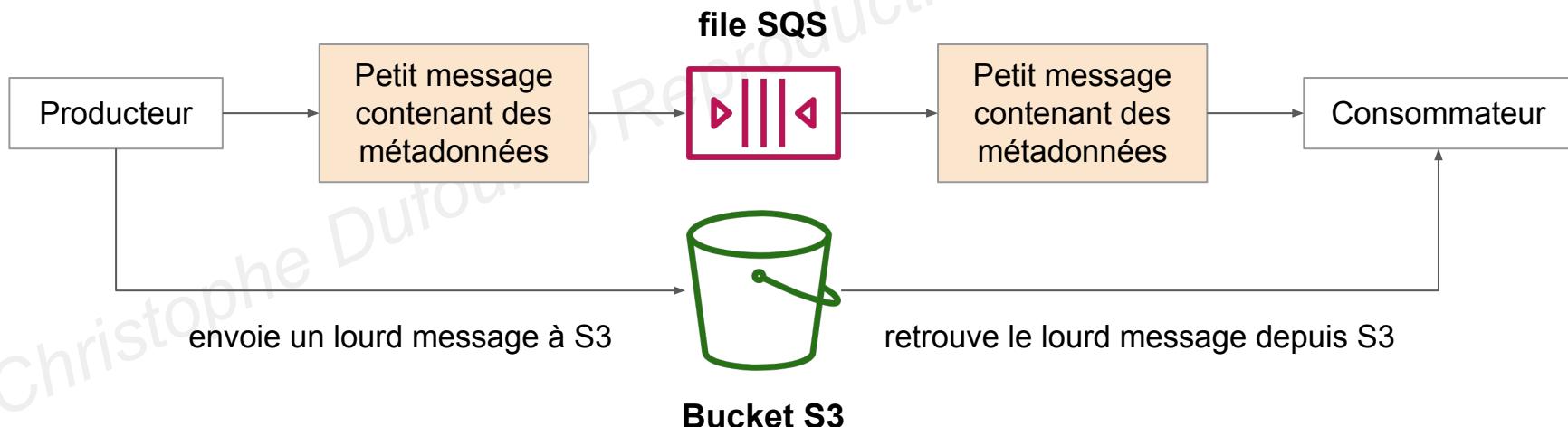
# SQS - Long Polling

- Quand un consommateur interroge la file, si elle est vide il peut opter pour “attendre”
- C'est ce qu'on appelle le Long Polling
- Le Long Polling diminue le nombre d'appels API vers SQS et fait gagner l'application en efficacité
- Le temps d'attente peuvent varier entre 1 et 20 secondes (préférable)
- Le Long Polling est préférable au Short Polling
- Activable au niveau de la file ou bien par l'API WaitTimeSeconds



# SQS - Extended Client

- La taille d'un message est limitée à 256 kb. Comment envoyer des messages volumineux (ex: 1GB) ?
- Solution: le SQS Extended Client (bibliothèque Java)



# SQS - fonctions d'API à connaître

- **CreateQueue (MessageRetentionPeriod)**, **DeleteQueue**
- **PurgeQueue**: supprime tous les messages de la file
- **SendMessage (DelaySeconds)**, **ReceiveMessage**, **DeleteMessage**
- **ReceiveMessageWaitTimeSeconds**: Long Polling
- **ChangeMessageVisibility**: change la durée du délai de visibilité
- Scripter les appels API pour **SendMessage**, **DeleteMessage**,  
**ChangeMesageVisibility** permet de réduire les coûts

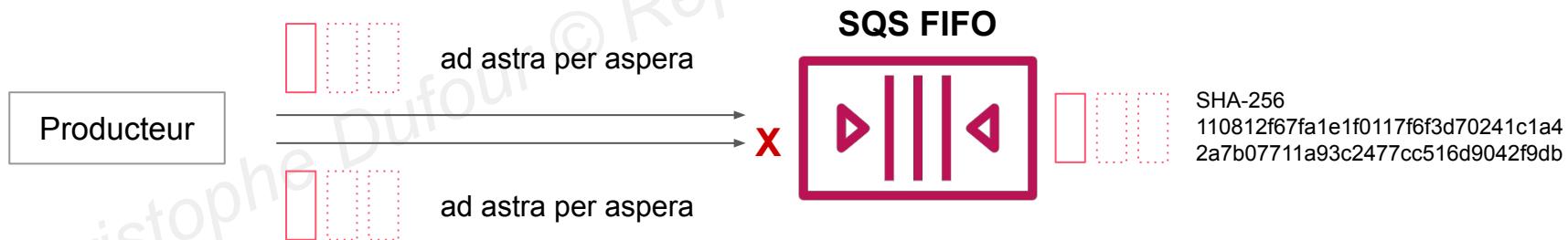
# SQS - file FIFO

- FIFO = First In First Out (ordonnancement des messages de la file)
- Débit limité à 300 msg/s sans traitement par lots (batching), 3000 avec
- Exactement un envoi de message (suppression des doublons)
- Messages traités dans l'ordre par le consommateur



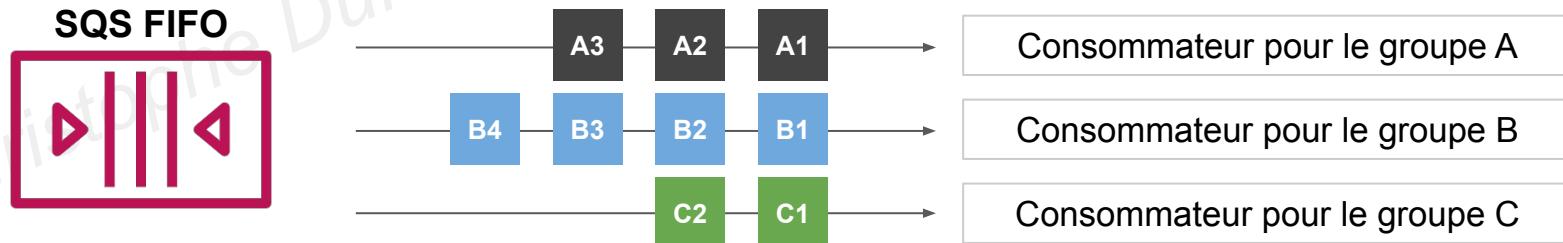
# SQS FIFO - Deduplication

- Intervalle de deduplication: 5 minutes
- Deux méthodes
  - sur la base du contenu: hash SHA-256 du corps du message
  - fournir un explicite Message Deduplication ID



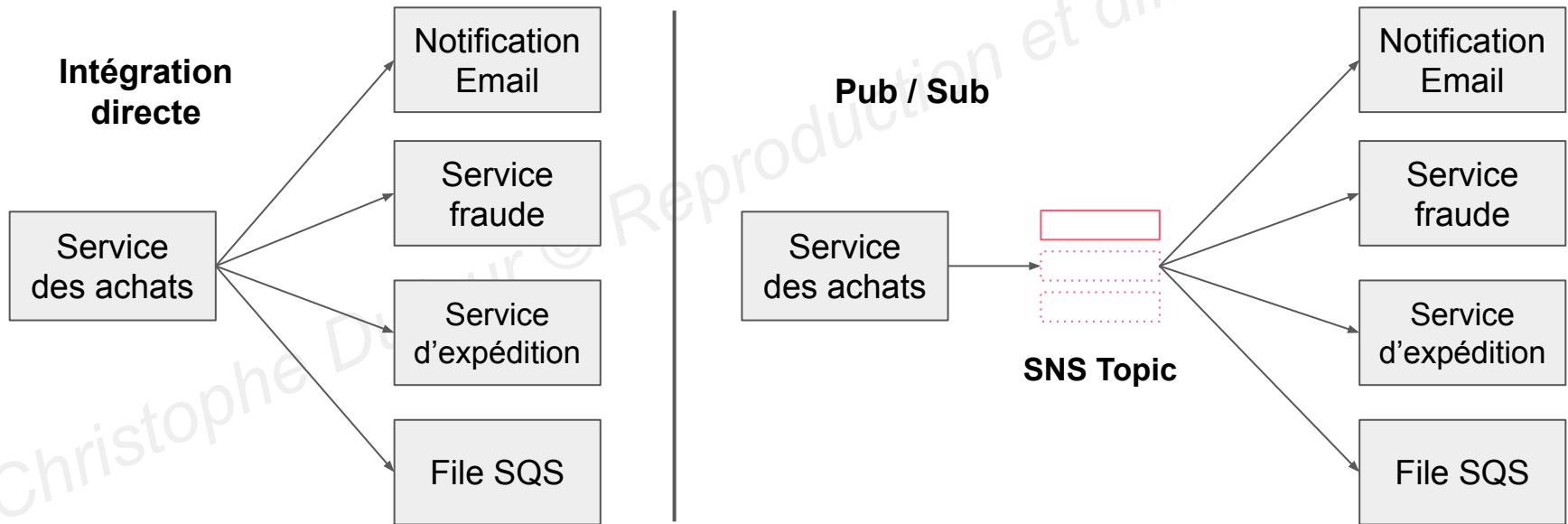
# SQS FIFO - Message grouping

- Spécifier un **MessageGroupId** dans une file SQS FIFO permet d'avoir un seul consommateur avec tous les messages ordonnés
- Pour obtenir un ordonnancement au niveau d'un sous-ensemble de messages, spécifier différentes valeurs pour le **MessageGroupId**
  - les messages partageant un MessageGroupId commun seront ordonnés dans le groupe
  - chaque Group ID peut avoir plusieurs consommateurs (traitement parallèle)
  - l'ordonnancement à travers (cross) les groupes n'est pas garanti



# SNS

- Et si l'on souhaitait envoyer un message à plusieurs receveurs ?





# SNS

- L'émetteur d'un événement envoie les messages uniquement à un topic SNS
- Nous pouvons ajouter autant de souscripteurs que souhaité au topic SNS
- Chaque souscripteur recevra tous les messages
- Jusqu'à 10 000 000 de subscriptions par topic
- Limite de 100 000 topics
- Les souscripteurs peuvent être
  - SQS
  - HTTP/HTTPS
  - Lambda
  - Messages SMS
  - Notifications mobile

# SNS - intégration avec de nombreux services AWS

- De nombreux services AWS peuvent envoyer directement des données à SNS
  - CloudWatch (alarmes)
  - notifications d'ASG
  - S3 pour les événements de bucket
  - CloudFormation, au changement d'état
  - etc.

# SNS - publier

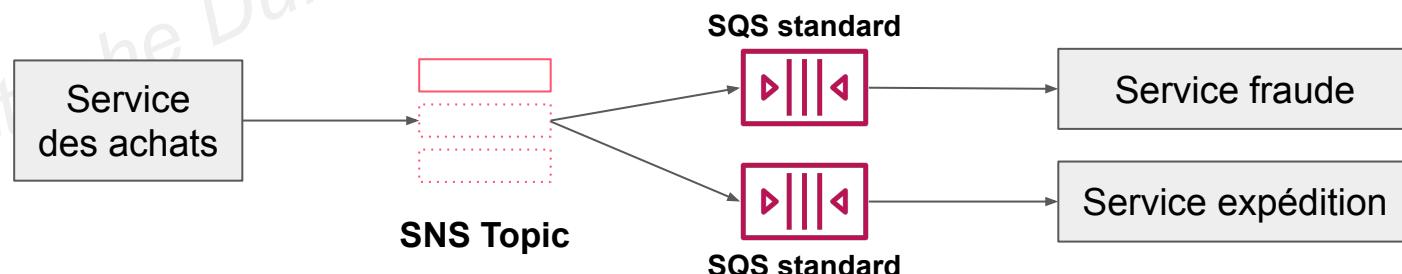
- Topic Publish (via SDK)
  - créer un topic
  - créer une souscription (ou plusieurs)
  - publier vers le topic
- Direct Publish (pour les SDK des apps mobile)
  - créer une plateforme application
  - créer un platform endpoint
  - publier vers le platform endpoint
  - fonctionne avec Google GCM, Apple APNS, Amazon ADM, ...

# SNS - sécurité

- Chiffrement
  - en transit via l'API HTTPS
  - au repos via KMS
  - côté client si le client souhaite chiffrer/déchiffrer lui-même
- Contrôles d'accès: stratégies IAM pour réguler l'accès à l'API SQS
- SNS Access Policies (similaires aux S3 Bucket policies)
  - utiles pour les accès cross-account aux topics SNS
  - utiles pour autoriser d'autres services (S3,...) à écrire dans un topic SNS

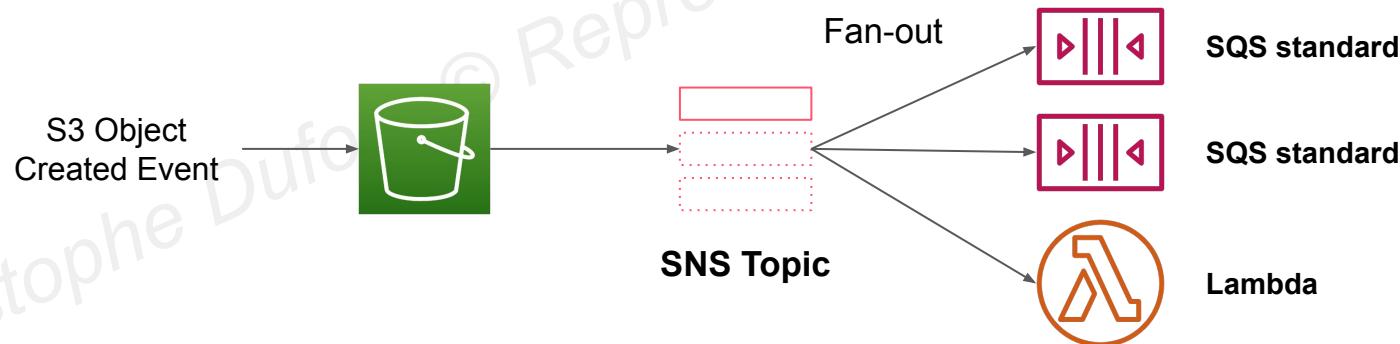
# SNS + SQS: Fan Out pattern

- Push une première fois dans SNS, le message est reçu par toutes les files SQS (souscripteurs)
- Totalement découplé, pas de perte de données
- SQS permet de persister les données, différer et retenter le traitement
- Possibilité d'ajouter davantage de souscripteurs SQS dans le temps
- S'assurer que la politique d'accès SQS autorise SNS à écrire
- **SNS ne peut pas envoyer des messages à SQS FIFO** (AWS limitation)



# SNS - événements S3 vers plusieurs files

- Pour une même combinaison de **type d'événement** (ex: création d'objet) et de **préfixe** (ex: images/) une seule S3 Event Rule est possible
- Pour envoyer le même événement S3 à plusieurs files, utiliser fan-out

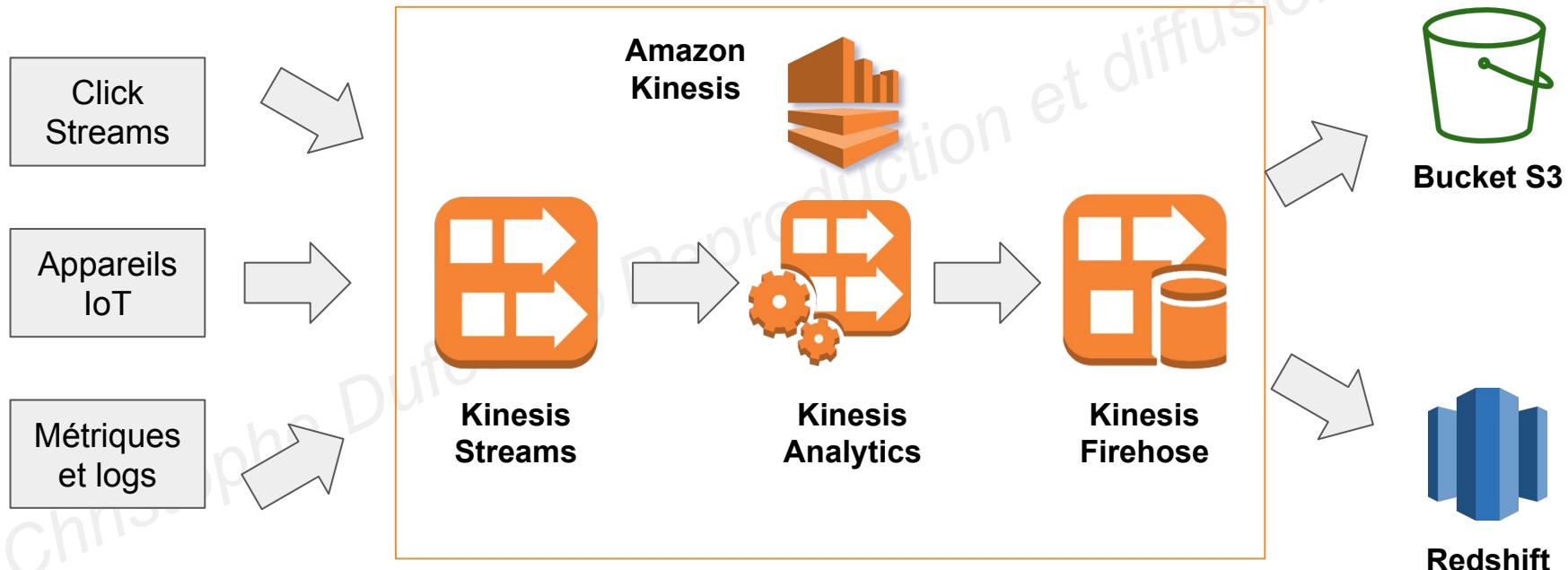




# Kinesis - vue d'ensemble

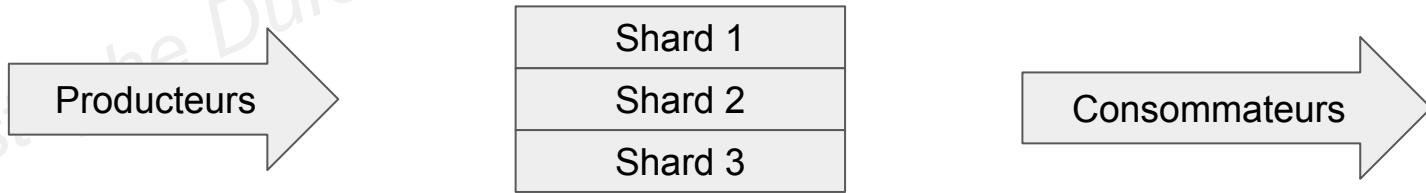
- **Kinesis** est une alternative, gérée, à Apache Kafka
- Excellent pour les logs d'application, métriques, IoT, clickstreams
- Excellent pour le big data en temps réel
- Excellent pour les frameworks de traitement de streaming (Spark, NiFi, etc.)
- Les données sont automatiquement répliquées dans 3 AZ
- **Kinesis Streams:** récupération de flux faible latence avec dimensionnement
- **Kinesis Analytics:** analyses en temps réel sur des flux avec SQL
- **Kinesis Firehose:** charge des flux dans S3, Redshift, ElasticSearch...

# Kinesis - schéma



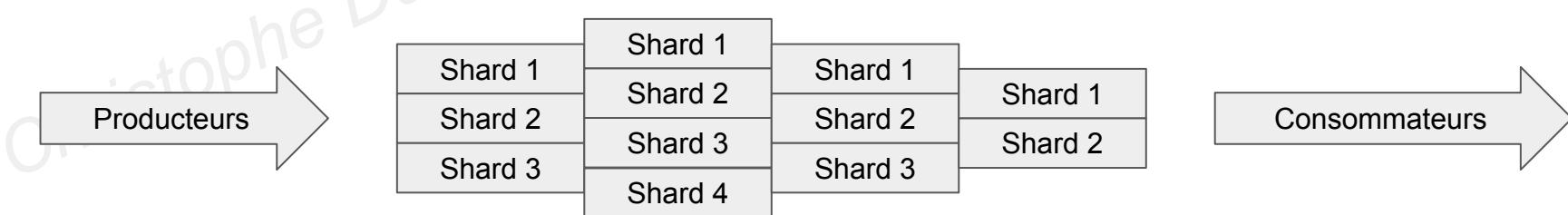
# Kinesis Streams - vue d'ensemble

- Les flux sont divisés en partitions/shards ordonnées
- Rétention de données: 1 jour par défaut, 7 max
- Plusieurs applications peuvent consommer le même flux
- Traitement temps réel avec dimensionnement du débit
- Une fois les données insérées dans Kinesis, elles ne peuvent pas être supprimées (immutability)



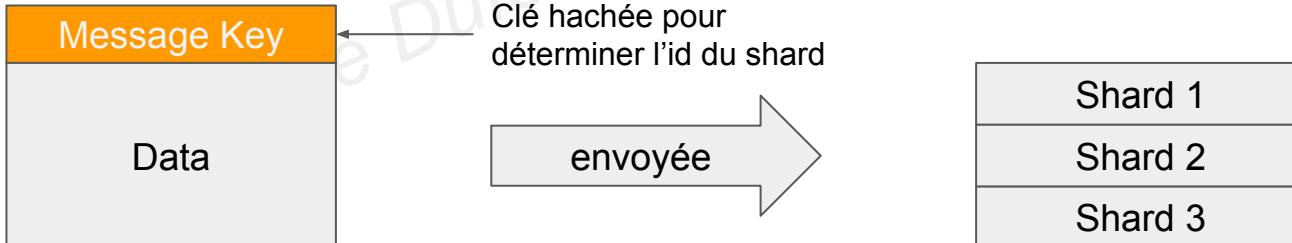
# Kinesis Streams - Shards

- Un flux est fait de plusieurs shards
- 1 MB/s ou 1000 msg/s en écriture par shard
- 2 MB/s en lecture par shard
- Facturation par shard approvisionné, nombre de shards illimité
- Traitement par lots disponibles
- Le nombre de shards peut évoluer (reshard/merge)
- **Les enregistrements sont ordonnés dans les shards**



# Kinesis API - Put records

- PutRecord API + Clé de partition hachée
- Même clé, même partition
- Les messages envoyés obtiennent un index de séquence
- Utiliser le batching avec PutRecords pour réduire les coûts et augmenter le débit
- **ProvisionedThroughputExceeded** en cas de dépassement des limites
- Utilisation avec CLI et SDK

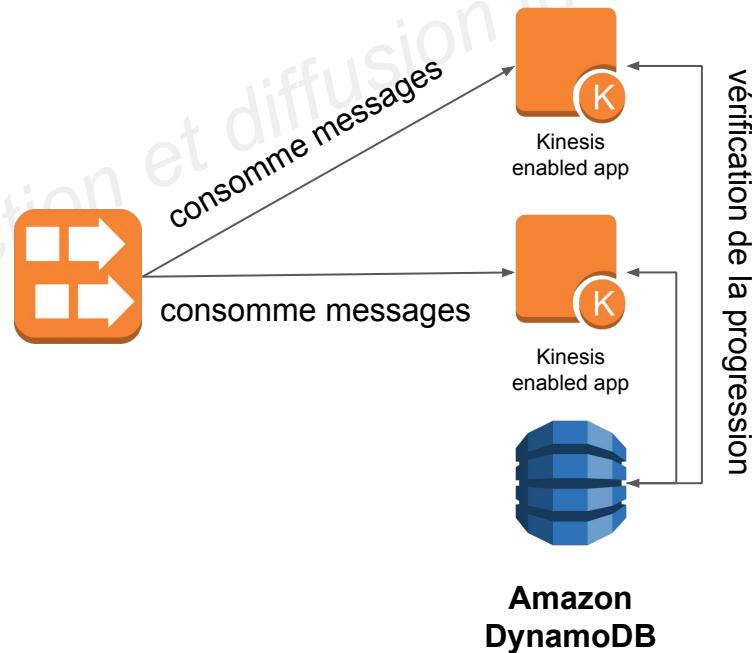


# Kinesis API - Exceptions

- ProvisionedThroughputExceeded Exception
  - se produit quand trop de données sont envoyées
  - s'assurer de ne pas avoir de "hot shard" (mauvaise clé de partition, trop de données y sont envoyées)
- Solution
  - retenter avec backoff
  - augmenter le nombre de shards (dimensionnement)
  - vérifier que la clé de partition est bonne

# Kinesis API - Consommateurs

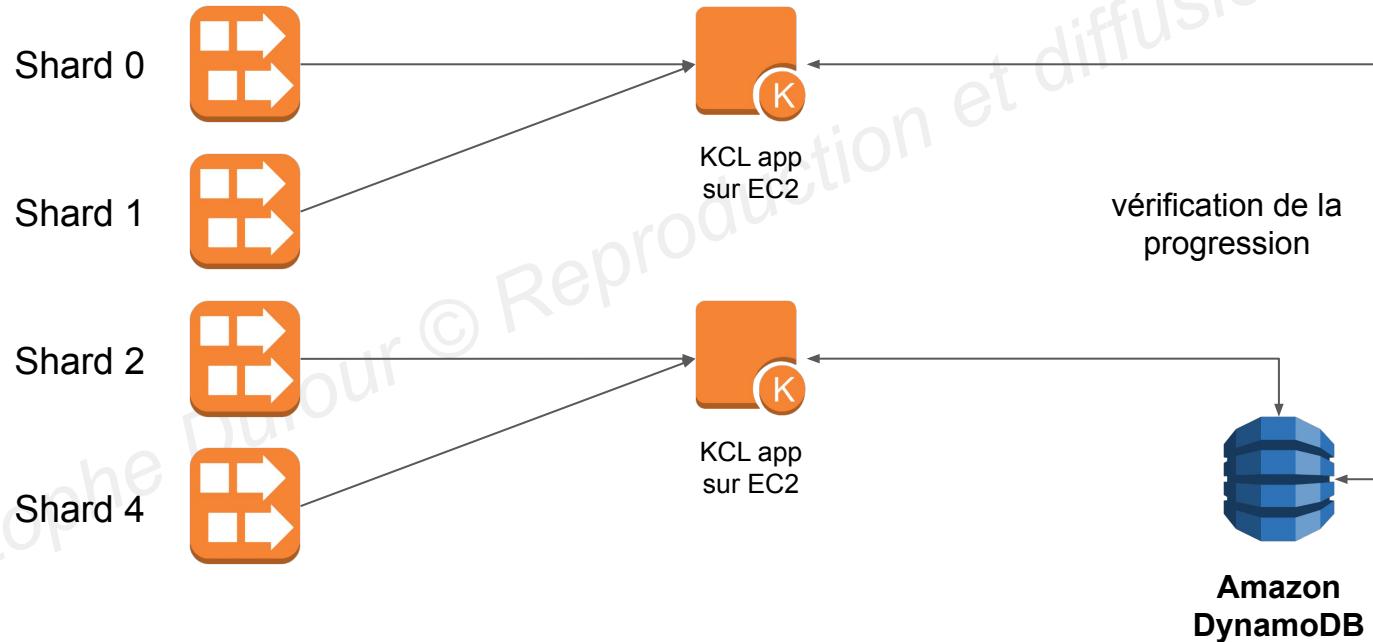
- Consommateur “classique”: CLI, SDK, etc
- Kinesis Client Library (KCL) en Java, Nodejs, Python, Ruby, .Net
  - KCL utilise DynamoDB pour marquer l'offset
  - KCL utilise DynamoDB pour suivre d'autres workers et partager le travail entre plusieurs shards



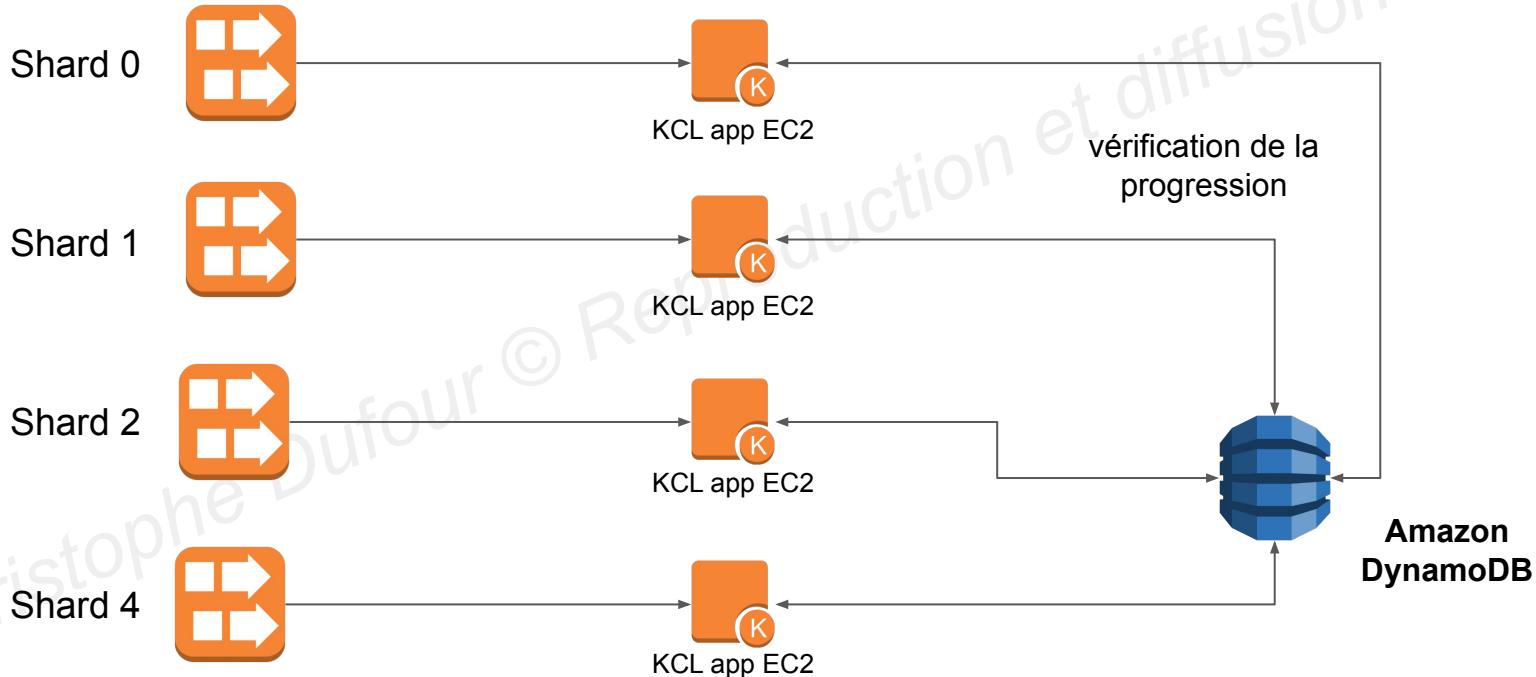
# Kinesis - KCL

- Kinesis Client Library (KCL) est une bibliothèque Java permettant de lire des enregistrements à partir de flux Kinesis dans des applications distribuées partageant la charge de lecture
- Règle: chaque shard doit être lu par une seule instance KCL
- 4 shards = 4 instances KCL, 6 shards = 6 instances, etc.
- La progression du traitement est vérifiée dans DynamoDB (requiert accès IAM)
- KCL peut d'exécuter dans EC3, EB, Application sur site
- **Enregistrements lues dans l'ordre au niveau du shard**

# KCL - exemple: 4 shards



# KCL - exemple: 4 shards, scaling KCL app



# Kinesis - Sécurité

- Contrôle d'accès via stratégies IAM
- Chiffrement en transit via points de terminaison HTTPS
- Chiffrement au repos avec KMS
- Possible chiffrement/déchiffrement côté client (plus difficile)
- Points de terminaison VPC disponibles pour permettre à Kinesis d'accéder à un VPC



# Kinesis Analytics

- Analyse de flux en temps réel avec SQL
- Kinesis Analytics
  - Auto Scaling
  - pas besoin d'approvisionner de serveur (service géré)
  - temps réel
- Facturation à la consommation
- Possibilité de créer des flux à partir de requêtes en temps-réel



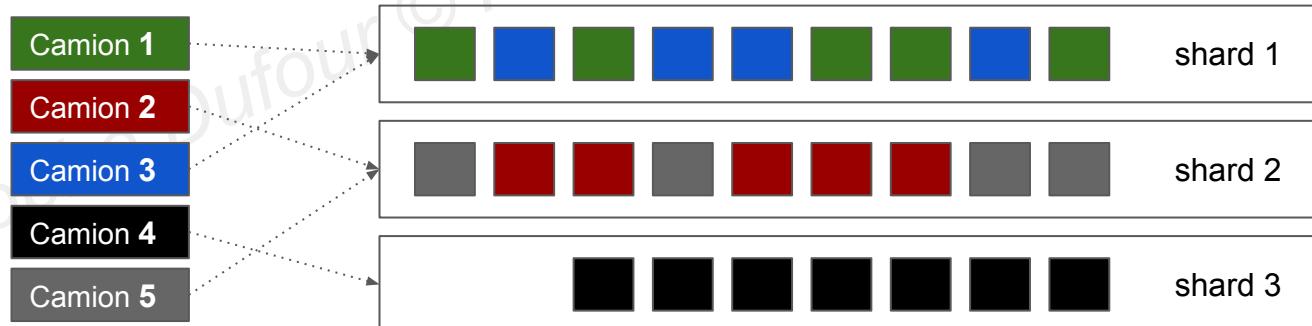
# Kinesis Firehose

- Service complètement géré, pas d'administration
- Presque temps réel (60 secondes de latence)
- Charge les données dans Redshift, S3, ElasticSearch, Splunk
- Dimensionnement auto
- Supporte de nombreux formats de donnée (conversion facturée)
- Facturation liée au volume de données chargées



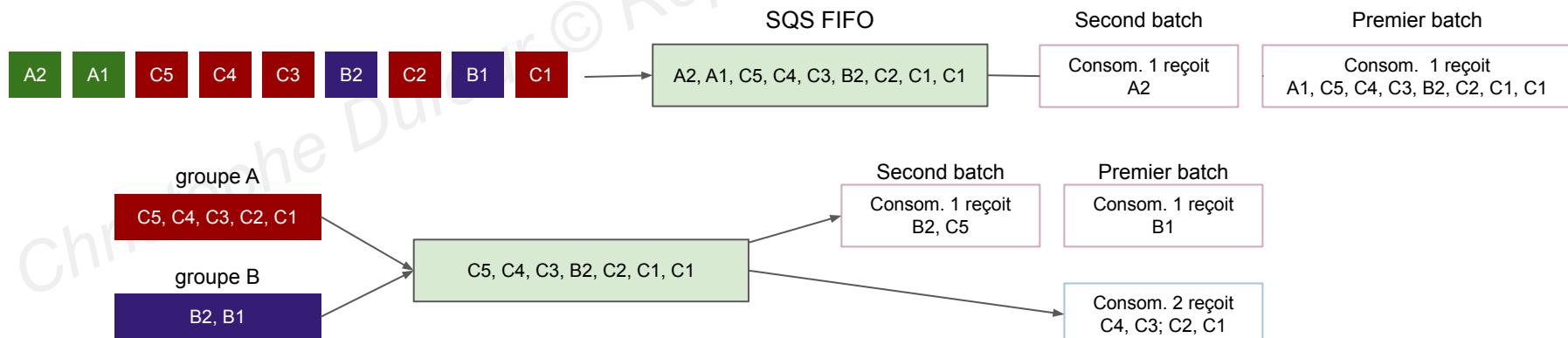
# Kinesis - ordonner les données

- Imaginer 100 camions sur la route enoyant leurs coords. GPS à AWS
- On veut consommer les données dans l'ordre pour chaque camion afin de suivre les déplacements précisément
- Comment envoyer les données à Kinesis ?
- Réponse: clé de partition => l'ID du camion (même clé, même partition)



# SQS - ordonner les données

- Pour SQS standard, il n'y pas d'ordonnancement
- Pour SQS FIFO, sans utilisation de Group ID, les messages sont consommés dans leur ordre d'émission, **avec un seul consommateur**
- Pour dimensionner le nombre de consommateurs avec les messages “groupés” quand ils sont liés, utiliser un Group ID (similaire à la clé de partition dans Kinesis)



# Kinesis vs SQS ordering

- **Imaginons que nous ayons 100 camions, 5 shards Kinesis, 1 SQS FIFO**
- Kinesis Data Streams
  - en moyenne, nous aurons 20 camions par shard
  - les données seront ordonnées par shard
  - nombre max. de consommateurs en parallèle: 5
  - nous recevrons jusqu'à 5mb/s de données
- SQS FIFO
  - nous n'avons qu'une seule file SQS FIFO
  - nous aurons 100 id de groupe
  - nb max de consom. en parallèle : 100 (puisque 100 Groupd ID)
  - nous recevrons jusqu'au 300 msg/s (ou 3000 avec le batching)

# SQS vs SNS vs Kinesis

SQS	SNS	Kinesis
consommateur “Pull” (demande) les données	envoi (push) des données aux souscripteurs (Pub/Sub)	consommateur “Pull” les données
données supprimées après traitement	jusqu'à 10 000 000 de souscripteurs	autant de consommateurs que souhaité
autant de consommateurs que souhaités	données non persistées (perdues si non livrées)	possibilité de “rejouer” les données
pas besoin d'approv. de débit	pas besoin d'approv. de débit	besoin d'approv. de débit
pas de garantie d'ordre (sauf FIFO)	jusqu'à 100 000 topics	ordre au niveau du shard
capacité de durée par message individuel	SQS pour fan-out pattern	données expirent après x jours

# AWS Lambda

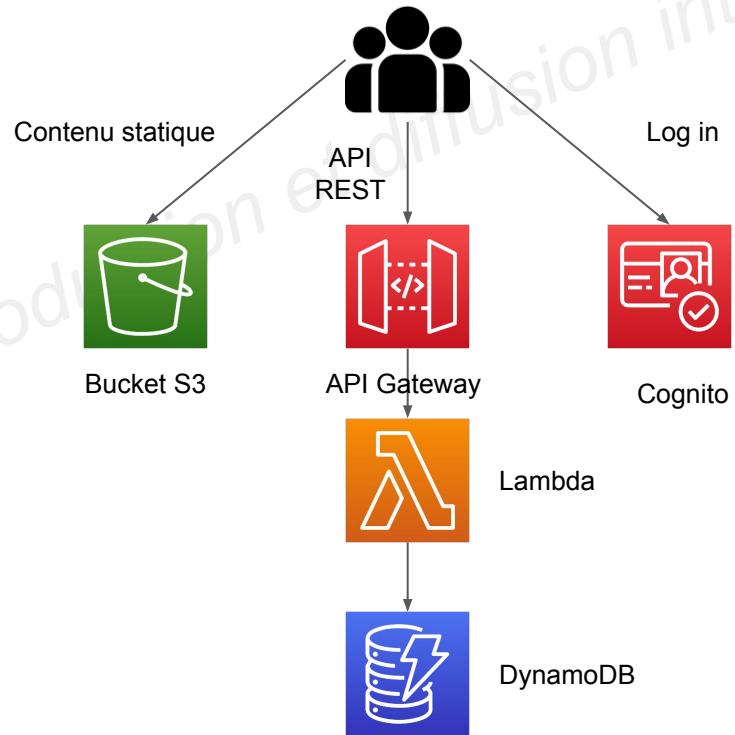
Le monde sans serveur (serverless) !

# Serverless - de quoi parle-t-on ?

- Nouveau paradigme dans lequel les développeurs n'a pas à se soucier de la gestion de serveurs
- Concentration sur le code
- Déploiement de... fonctions !
- A l'origine du Serverless, les FaaS (Function as a Service)
- Evidemment, les serveurs continuent d'exister mais “cachés”, nous n'avons pas à les gérer, les approvisionner, etc.

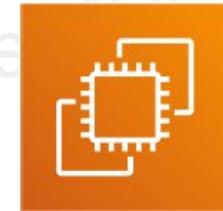
# AWS Serverless

- AWS Lambda
- DynamoDB
- AWS Cognito
- AWS API Gateway
- Amazon S3
- AWS SNS et SQS
- AWS Kinesis Data Firehose
- Aurora Serverless
- Step Functions
- Fargate



# AWS Lambda - pourquoi ?

- Serveurs virtuels dans le Cloud
- Limités en RAM et CPU
- En permanent cours d'exécution (selon le launch type)
- Dimensionnement manuel parfois



Amazon EC2

- 
- Fonctions virtuelles - **pas de serveur à gérer**
  - Limitées en temps - **exécutions brèves**
  - Exécutées à la demande
  - Dimensionnement automatique



Amazon Lamda

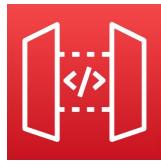
# AWS Lambda - avantages

- Tarification facile
  - par requête et temps d'exécution
  - free tier de 1 000 000 de requêtes et 400 000 GBs de temps d'exec.
- Intégré avec toute une suite de services AWS
- Intégré avec de nombreux langages de programmation
- Monitoring facile via AWS CloudWatch
- Obtention facile de ressources allouées aux fonctions (jusqu'à 3GB de ram)
- L'augmentation de ram améliore aussi les paramètres CPU et réseau

# AWS Lambda - langages supportés

- Nodejs
- Python
- Java (compatible Java 8)
- C# (.NET Core)
- C#/Powershell
- Golang
- Ruby
- Custom Runtime API (supporté par la communauté, ex: Rust)
- *Note: pas de Docker dans Lambda (cf ECS/Fargate)*

# AWS Lambda - intégrations principales



API Gateway



DynamoDB



S3



CloudFront



Cognito



Kinesis



SQS



SNS

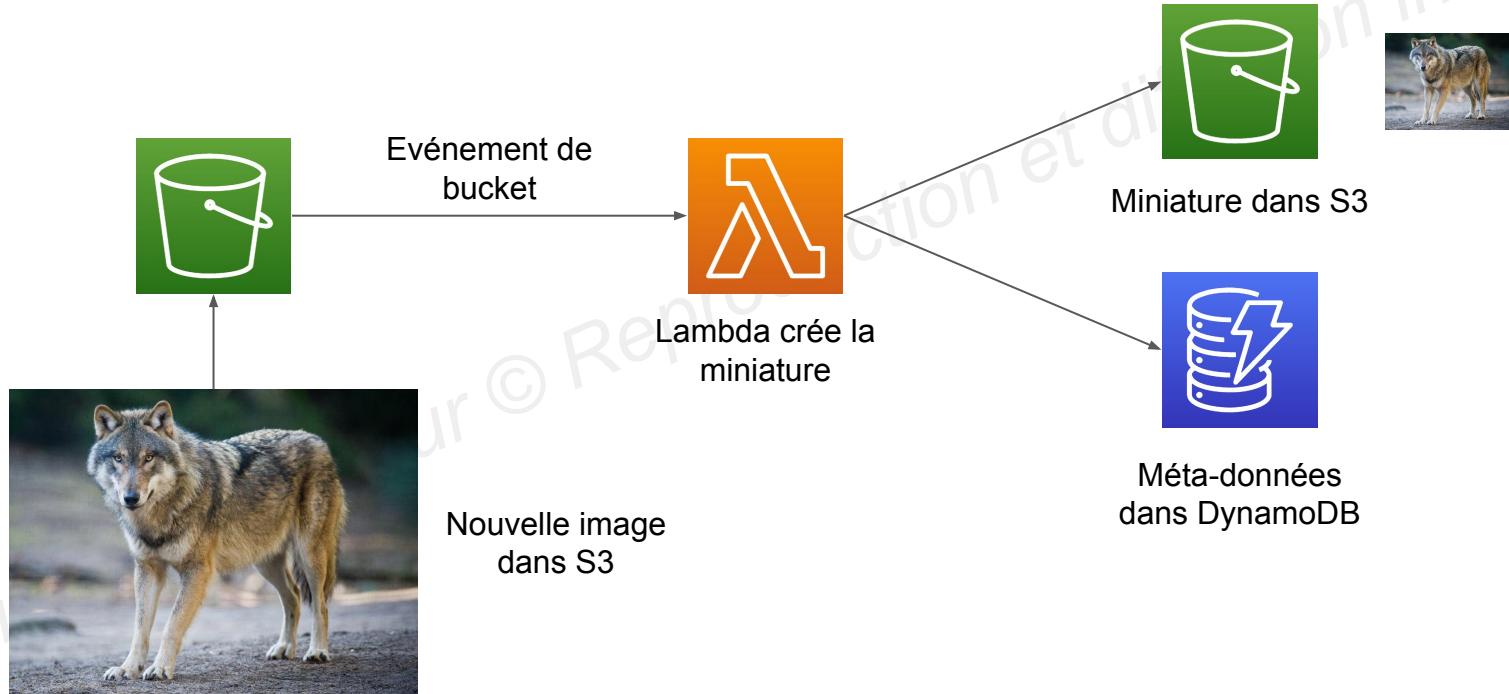


CloudWatch  
Logs



CloudWatch  
EventBridge

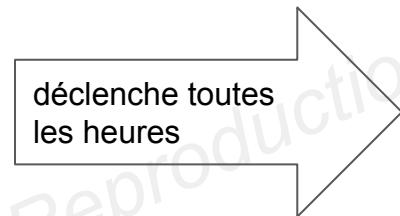
# Lambda - exemple: création de miniature



# Lambda - exemple: tâche CRON



CloudWatch  
EventBridge



Lambda exécute  
une tâche

# AWS Lambda - tarification

## Tarification AWS Lambda

Région : Europe (Paris) ▾

### Prix

Demandes	0,20 USD par million de demandes
Durée	0,0000166667 USD for pour chaque Go-seconde

Le tarif pour la **durée** est fonction de la quantité de mémoire que vous attribuez à votre fonction. Vous pouvez allouer n'importe quelle quantité de mémoire entre 128 Mo et 10 240 Mo à votre fonction, et ceci par incrément de 1 Mo. Le tableau ci-dessous contient quelques exemples de tarifs par tranche de 1 ms associés à différentes tailles de mémoire.

Mémoire (Mo) (avec une plus grande mémoire)	Tarif par 1 ms
128	0,000000021 USD
512	0,000000083 USD
1 024	0,000000167 USD
1 536	0,000000250 USD
2 048	0,000000333 USD
3 072	0,000000500 USD

<https://aws.amazon.com/fr/lambda/pricing>

Si vous avez attribué 128 Mo de mémoire à votre fonction et que vous l'avez exécutée 30 millions de fois en un mois (pour une durée de 200 ms par exécution), vos frais sont calculés comme suit :

### Frais de calcul mensuels

Le tarif de calcul mensuel revient à 0,00001667 USD par Go-s et le niveau gratuit offre 400 000 Go-s.

Taux de calcul (en secondes) = 30 M \* (0,2 s) = 6 000 000 secondes

Taux de calcul total (en Go-s) = 6 000 000 \* 128Mo/1 024 = 750 000 Go-s

Taux de calcul total – Taux de calcul offert = Taux de calcul facturable par mois (en secondes)

750 000 Go-s – 400 000 Go-s offerts = 350 000 Go-s

**Frais de calcul mensuels = 350 000 \* 0,00001667 USD = 5,83 USD**

### Frais de requêtes mensuels

Le tarif de requête mensuel est de 0,20 USD par million de requêtes et le niveau gratuit offre un million de requêtes par mois.

Nombre total de requêtes – Nombre de requêtes offertes = Nombre de requêtes facturables par mois

30 M de requêtes – 1 M de requêtes offertes = 29 M de requêtes facturables par mois

**Frais de requêtes mensuels = 29 M \* 0,2 USD/M = 5,80 USD**

### Frais de calcul totaux

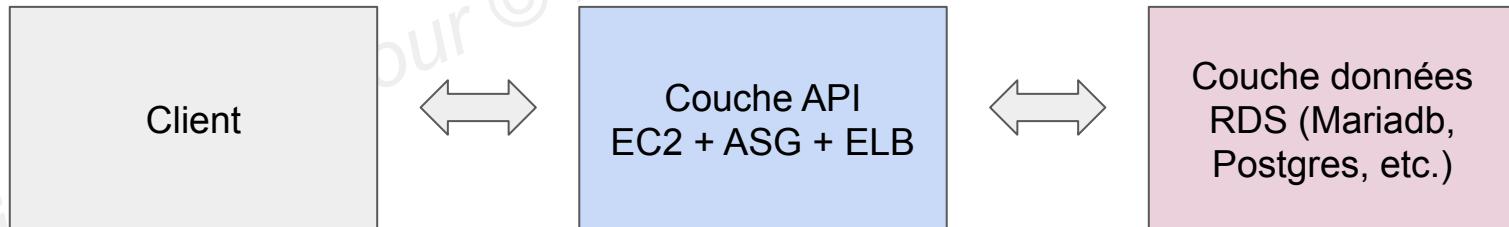
**Frais totaux = Frais de calcul + Frais de requêtes = 5,83 USD + 5,80 USD = 11,63 USD par mois**

# DynamoDB

Base de données NoSQL Serverless

# Architecture traditionnelle

- Les applications traditionnelles utilisent des bases de données relationnelles
- Ces bases de données peuvent être pilotées par le langage SQL
- Forts pré-requis concernant la modélisation des données
- Possibilité de jointure, agrégation, calcul sur les données
- Dimensionnement généralement vertical (plus de cpu/ram/io)



# Bases de données NoSQL

- Non relationnelles et sont distribuées
- Exemples: MongoDB, Apache CouchDB, DynamoDB, etc.
- Pas de schéma strict
- Pas de jointures
- Données fournies en un seule ligne (record/doc)
- Pas d'agrégation telle que “SUM”
- Dimensionnement horizontal



# DynamoDB

- Service entièrement géré avec haute disponibilité et réPLICATION 3 AZ
- Bdd non relationnelle distribuée
- Dimensionnement pour des charges de travail gigantesques
- Millions de requêtes/s, milliards de lignes, centaines de TB de stockage
- Rapide et stable en performances (faible latence en lecture)
- Intégré avec IAM pour sécurité, autorisation et administration
- Permet la programmation orientée événement avec DynamoDB Streams
- Faible coût
- Capacités d'auto-dimensionnement

# DynamoDB - les bases

- DynamoDB est composé de **tables**
- Chaque table a une **clé primaire** (déterminée à la création)
- Chaque table peut avoir un nombre infini d'éléments (~ lignes)
- Chaque élément a des **attributs** (ajout/modification dynamique)
- Taille max d'un élément: 400KB
- Types de données supportés
  - Scalaire: String, Number, Binary, Boolean, Null
  - Document: List, Map
  - Set: String Set, Number St, Binary Set

# DynamoDB - clé primaire

- **Option 1: Clé de partition uniquement**
- La clé de partition doit être unique pour chaque élément (pas deux éléments avec la même clé)
- La clé de partition permet la distribution de données
- Exemple:
  - user\_id pour une table users
  - city\_name pour une table cities

Clé de partition (unique)	city_name	country	major	Attributs
	Paris	France	Jacques Chirac	
	Milan	Italie	Silvio Berlusconi	
	Turin	Italie	Alex Del Piero	

# DynamoDB - clé primaire

- **Option 2: Clé de partition + Clé de tri**
- La combinaison doit être unique
- Données groupées par clé de partition
- Sort Key == range Key
- Exemple: table users-games
  - partition key: user\_id
  - sort key: game\_id

Clé primaire		
Clé de partition	Clé de tri	Attributs
user_id	game_id	result
chris85	400	win
chris85	410	win
gab23	400	lose

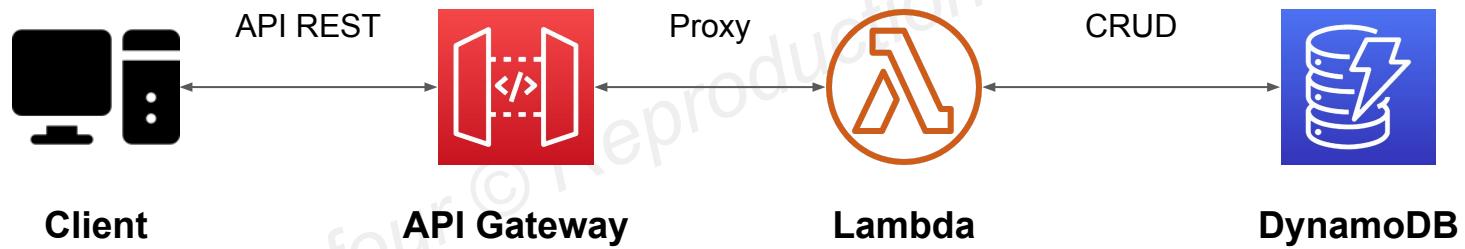
# DynamoDB - débits approvisionnés

- Une table doit être approvisionnée en unités de lecture/écriture
- **Read Capacity Units (RCU)**: débit en lecture
- **Write Capacity Units (WCU)**: débit en écriture
- Option: paramètrer un auto scaling des débits pour s'adapter à la demande
- Les limites peuvent temporairement dépassées via le “burst credit”
- Si le “burst credit” est épuisé, une “ProvisionedThroughputException” est levée. Il est alors conseillé d'utiliser l’ “exponential back-off retry”

# API Gateway

Construire, déployer et gérer ses APIs

# Exemple: construire une API serverless





# AWS API Gateway

- Lambda + API Gateway = aucune infrastructure à gérer
- Support de websocket
- Gestion de version (API v1, v2, etc.)
- Gestion d'environnement (dev, test, prod, etc.)
- Gestion de la sécurité (Authentification et autorisation)
- Création de clés API
- Gestion de l'engorgement des requêtes (request throttling)
- Swagger / Open API pour import rapide d'APIs
- Transformation et validation des requêtes et réponses
- Génération de spécifications SDK et API
- Mise en cache des réponses

# Amazon Cognito



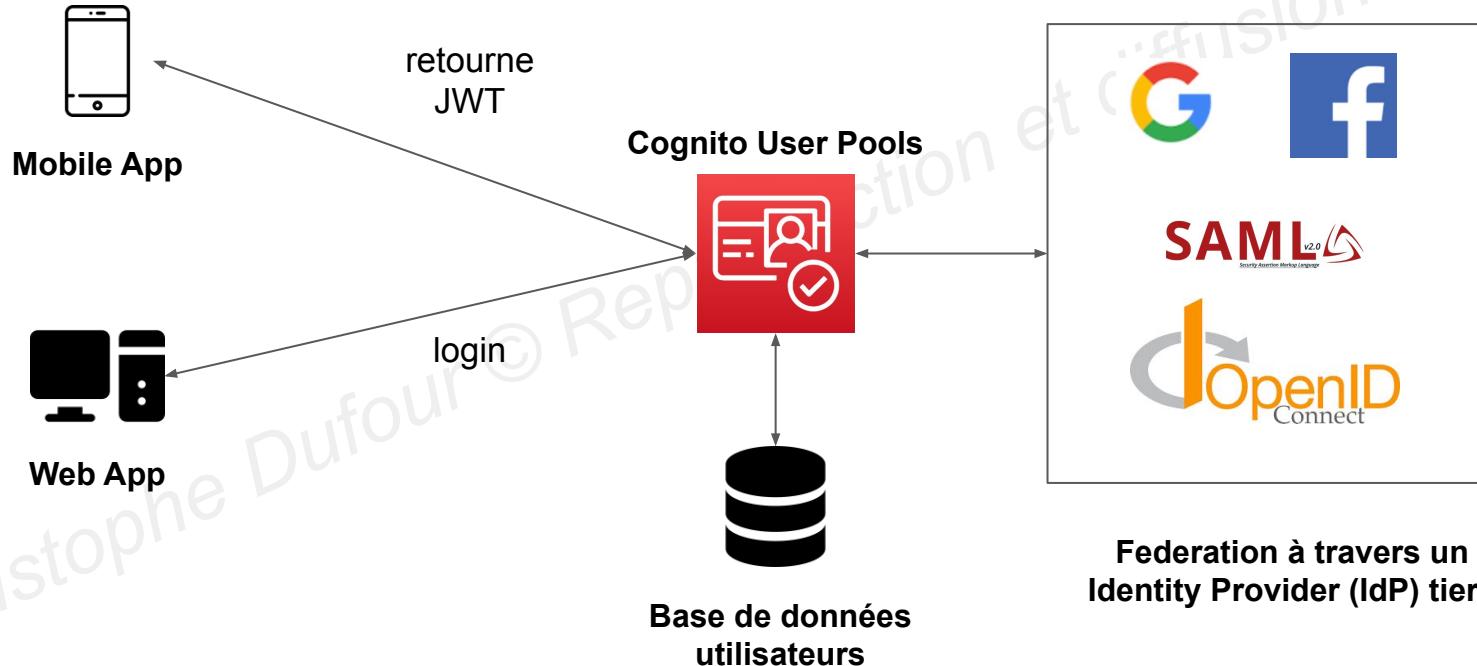
# Amazon Cognito

- Donne une identité à des utilisateurs externes à AWS pour intéragir avec l'application
- **Cognito User Pool**
  - fonctionnalité d'identification
  - intégration avec API Gateway et Application Load Balancer
- **Cognito Identity Pool (Federated Identity)**
  - fournit des identifiants AWS à nos utilisateurs pour leur permettre d'accéder directement à des ressources AWS
  - intégration avec Cognito Users Pool en tant que fournisseur d'identité
- **Cognito Sync**
  - synchronise les données depuis un appareil vers Cognito
  - déprécié et remplacé par AppSync

# Cognito User Pools (CUP)

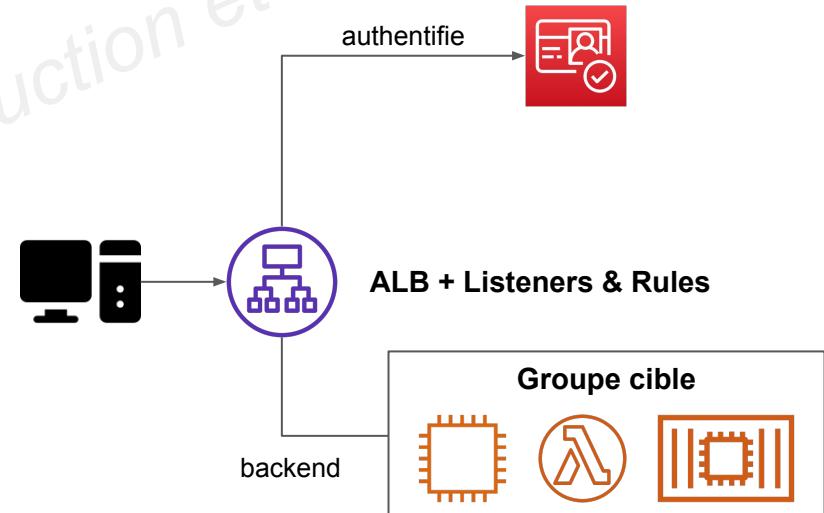
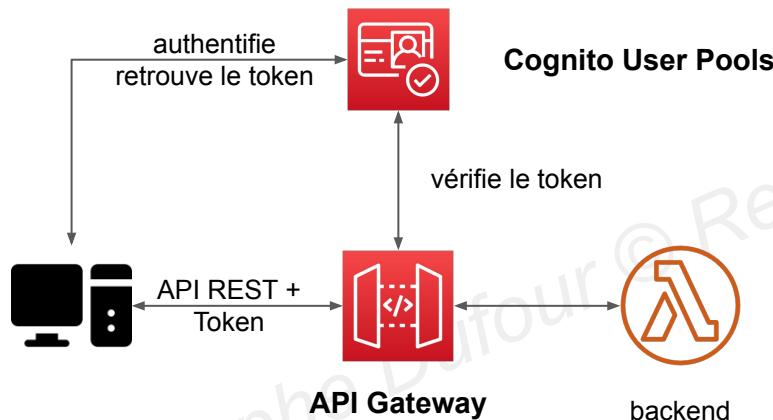
- Base de données serverless d'utilisateurs pour applications web et mobile
- Simple login: username (ou email) / password
- Password reset
- Vérification email et numéro de téléphone
- MFA
- Federated Identites: utilisateurs facebook, Google, SAML, etc.
- Blocage d'utilisateurs si identifiants compromis ailleurs
- Le login renvoie un JSON Web Token (JWT)

# Cognito User Pools (CUP) - schéma



# Cognito User Pools (CUP) - intégration

- CUP s'intègre avec API Gateway et Application Load Balancer

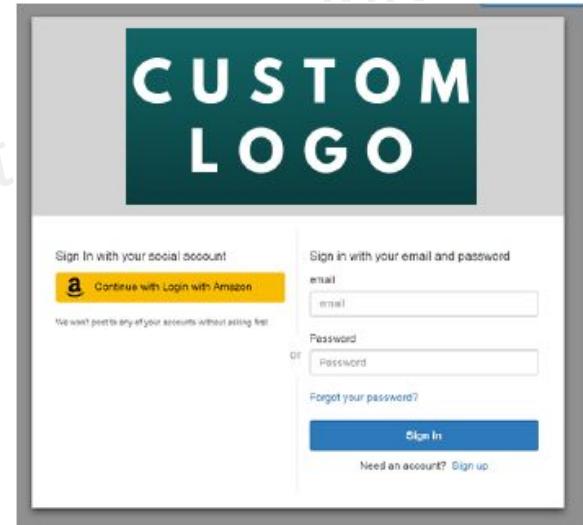


# Cognito Identity Pools - déclencheurs Lambda

Flux de groupe d'utilisateurs	Opération	Description
Flux d'authentification personnalisé	<b>Définition de la stimulation d'authentification</b>	Détermine la question de sécurité (stimulation) dans un flux d'authentification personnalisé
	<b>Création d'une stimulation d'authentification</b>	Crée une question de sécurité (stimulation) dans un flux d'authentification personnalisé
	<b>Vérification de la réponse à la stimulation d'authentification</b>	Détermine si une réponse est correcte dans un flux personnalisé
Événements d'authentification	<b>Déclencheur Lambda avant authentification</b>	Validation personnalisée pour accepter ou refuser la demande de connexion
	<b>Déclencheur Lambda après authentification</b>	Journalisation des événements pour l'analyse personnalisée
	<b>Déclencheur Lambda avant génération de jeton</b>	Demandes d'augmentation ou de suppression de jeton
Inscription	<b>Déclencheur Lambda avant inscription</b>	Validation personnalisée pour accepter ou refuser la demande d'inscription
	<b>Déclencheur Lambda après confirmation</b>	Messages de bienvenue ou journalisation des événements pour l'analyse personnalisée
	<b>Déclencheur Lambda de migration d'utilisateur</b>	Migration d'un utilisateur à partir d'un annuaire d'utilisateurs existants vers des groupes d'utilisateurs.
Messages	<b>Déclencheur Lambda de message personnalisé</b>	Personnalisation et localisation de messages avancées
Création de jeton	<b>Déclencheur Lambda avant génération de jeton</b>	Ajout ou suppression d'attributs dans des jetons d'ID
Fournisseurs tiers d'e-mails et de SMS	<b>Déclencheurs Lambda d'expéditeur personnalisés</b>	Utiliser un fournisseur tiers pour envoyer des SMS et des e-mails

# Cognito User Pools - UI hébergée d'authentification

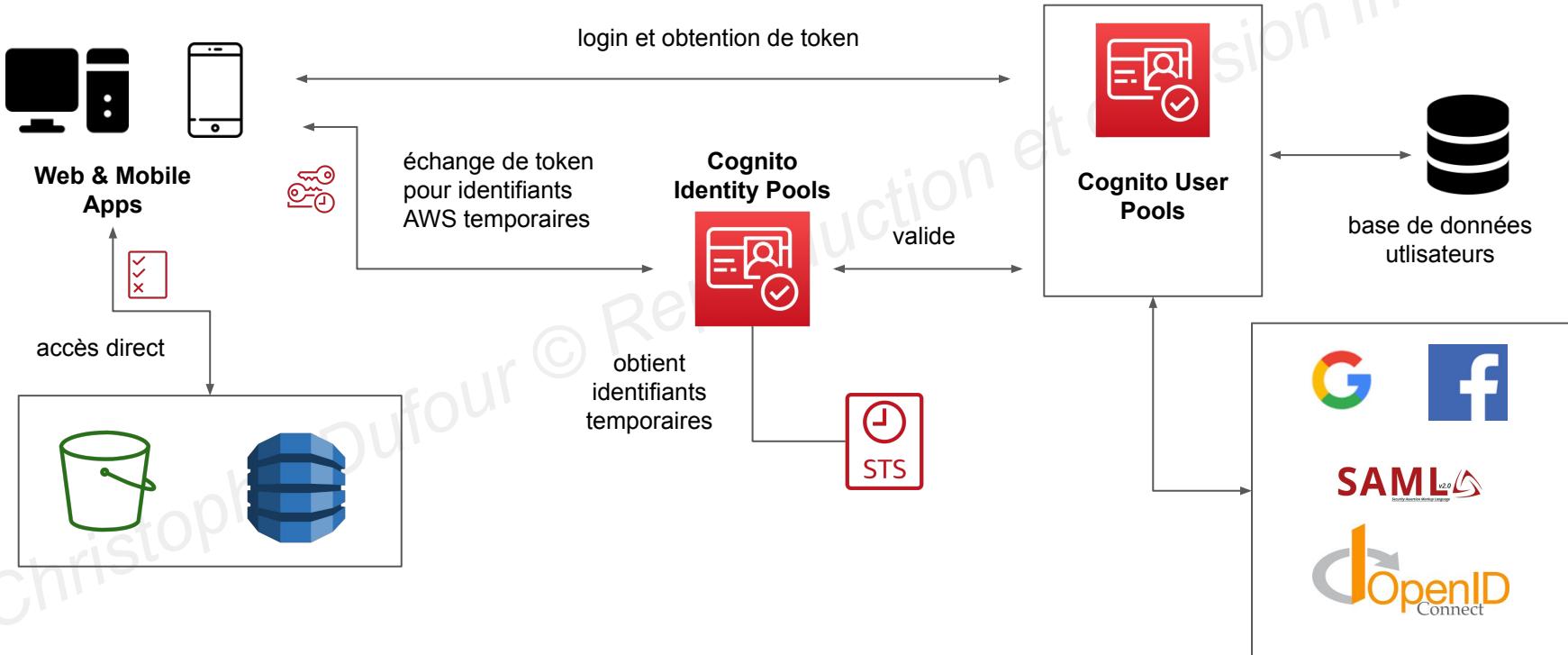
- Cognito dispose d'une interface utilisateur d'authentification hébergée qu'on peut ajouter à notre application pour gérer la création de compte / connexion au compte
- Ce dispositif offre une intégration avec les réseaux sociaux, OIDC ou SAML
- Personnalisable avec logo et CSS



# Cognito Identity Pools (Federated Identities)

- Permet d'obtenir des identifiants AWS temporaires pour nos utilisateurs
- Votre identity pool peut inclure
  - fournisseurs publics (login avec Amazon, Facebook, Google, Apple)
  - utilisateurs d'un Amazon Cognito user pool
  - fournisseurs OpenID et SAML
  - Developer Authenticated Identities (serveur de login personnalisé)
  - accès utilisateur invité (unauthenticated guest)
- Les utilisateurs peuvent alors accéder à AWS directement ou via API Gateway
  - Les règles IAM appliquées aux identifiants sont définies dans Cognito
  - Ces règles sont paramétrables finement sur la base du user\_id

# Cognito Identity Pools - schéma avec CUP



# Cognito Identity Pools - Rôles IAM

- Rôles IAM par défaut pour les utilisateurs authentifiés et invités
- Définissent les règles à suivre pour chaque utilisateur sur la base de son ID
- Possibilité de segmenter les accès avec des **policy variables**
- Les identifiants IAM sont obtenus par Cognito Identity Pools via STS

# Cognito Identity Pools - exemple utilisateur guest

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::mybucket/assets/my_picture.jpg"  
            ]  
        }  
    ]  
}
```

# Cognito Identity Pools - policy variable dans S3

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::mybucket"],  
            "Condition": {"StringLike": {"s3:prefix": ["${cognito-identity.amazonaws.com:sub}/*"]}}  
        },  
        {  
            "Action": [  
                "s3:GetObject",  
                "s3:PutObject"  
            ],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::mybucket/${cognito-identity.amazonaws.com:sub}/*"]  
        }  
    ]  
}
```

# Cognito Identity Pools - DynamoDB

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "dynamodb:GetItem", "dynamodb:BatchGetItem", "dynamodb:Query",  
                "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem",  
                "dynamodb:BatchWriteItem"  
            ],  
            "Resource": [  
                "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"  
            ],  
            "Condition": {  
                "ForAllValues:StringEquals": {  
                    "dynamodb:LeadingKeys": [  
                        "${cognito-identity.amazonaws.com:sub}"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

# Cognito User Pools vs Identity Pools

- **Cognito User Pools**
  - base de données utilisateurs pour apps web et mobile
  - permet des “federate logins” via Google, facebook, SAML, etc.
  - UI de login hébergée personnalisable
  - peut déclencher des Lambdas
- **Cognito Identity Pools**
  - obtient des identifiants AWS pour vos utilisateurs
  - permet des “federate logins” via Google, facebook, SAML, etc.
  - utilisateurs peuvent être invités (guets)
  - utilisateurs reliés (mapped) à des rôles/stratégies IAM, utilisation de variables
- **CUP + CIP = gestion user/password + accès aux services AWS**