

Formation AWS DevOps

Ihab ABADI / UTOPIOS

SOMMAIRE

1. Introduction DevOps
 1. Pourquoi le devops
 2. Culture devops
 3. Pratique devops
 4. Outils devops
2. Outils AWS
3. Infrastructure en Code
 1. AWS CloudFormation
 2. AWS CDK
4. AWS Intégration et livraison continue.
 1. AWS CodeCommit
 2. AWS CodeBuild
 3. AWS CodeDeploy
 4. AWS CodePipeline
5. Architecture Microservice
6. Utilisation des Conteneurs avec Docker
7. Containerisation avec AWS

SOMMAIRE

1. Architecture Microservice
2. Utilisation des Conteneurs avec Docker
3. Containerisation avec AWS
 1. AWS Container Registry
 2. AWS Container service EC2
 3. AWS Container service FARGATE
 4. AWS EKS
4. Application sans serveur
 1. AWS Fonction Lambda
 2. AWS SAM
5. DevSecOps
 1. Amazon GuardDuty
 2. Amazon Inspector

Pour quoi Nous avons besoin du devops?

30x

Déploiements
plus fréquents

200x

Délais plus courts

60x

Moins d'échecs

168x

Récupération plus
rapide

C'est quoi le devops ?

- Une culture et une philosophie
- Des bonnes pratiques
- Des outils

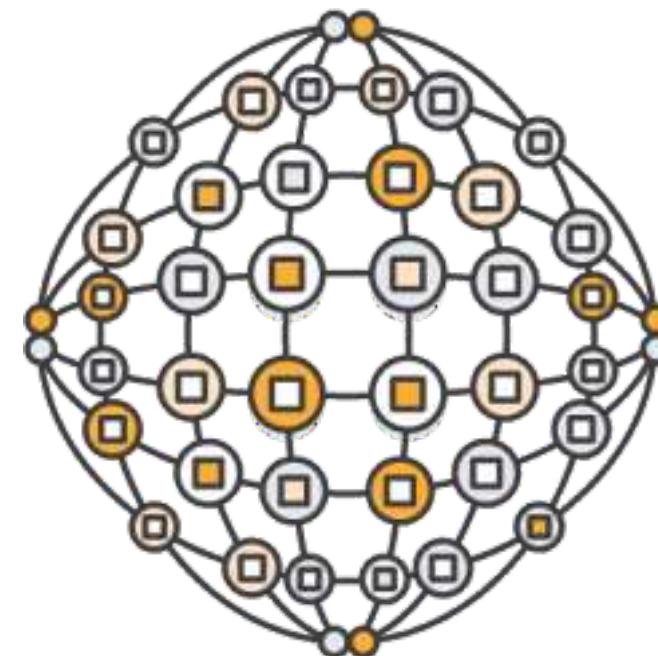
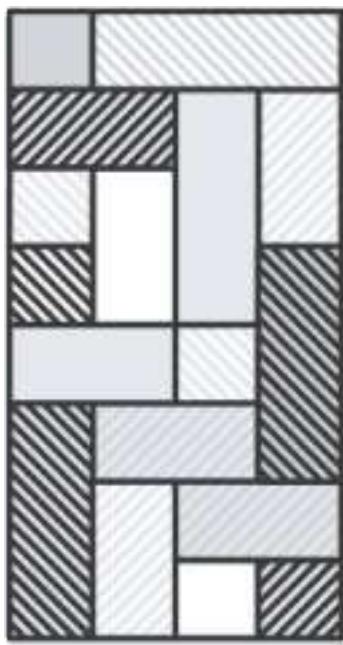
Culture DevOps

- Dev & Ops ensemble
- Responsabilité partagée
- Visibilité et communication



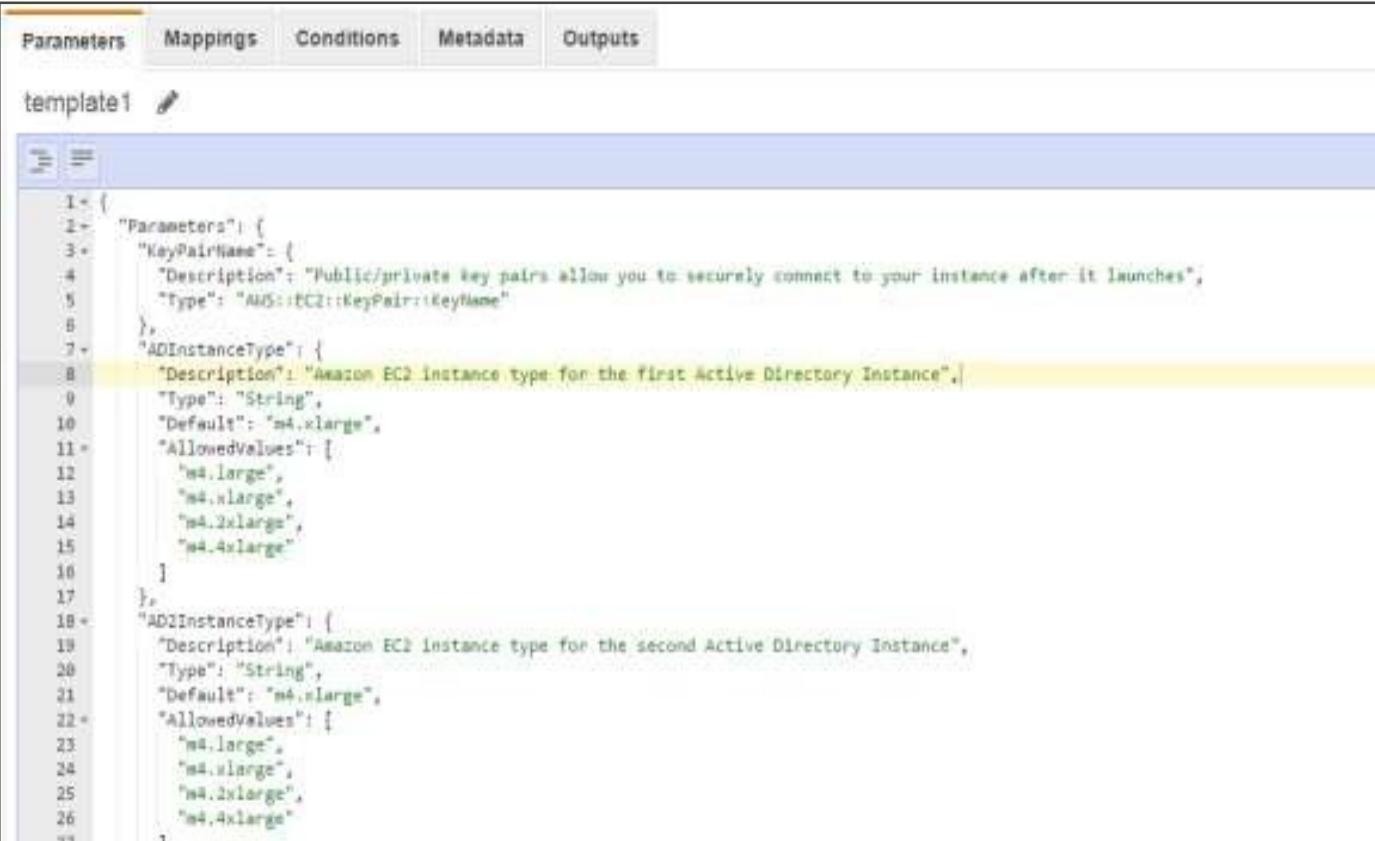
Pratiques DevOps

- Architecture microservice
 - Le passage d'architecture monolithique à des architectures en plusieurs petits services



Pratiques DevOps

- Infrastructure as Code
 - Modélisation de notre Infrastructure par le code

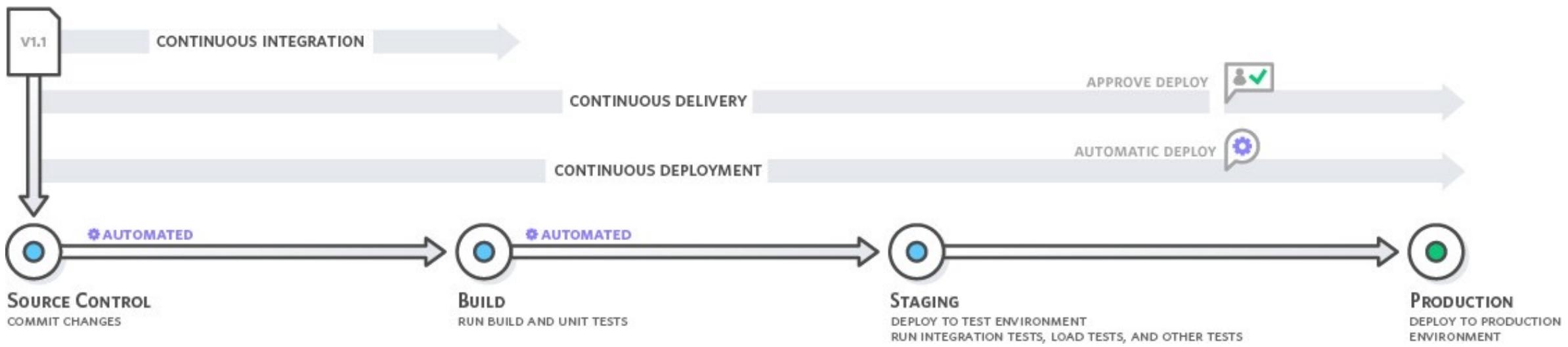


The screenshot shows the AWS CloudFormation 'template1' editor interface. The top navigation bar includes 'Parameters', 'Mappings', 'Conditions', 'Metadata', and 'Outputs'. The main area displays a JSON template with the following content:

```
1+ {
2+   "Parameters": {
3+     "KeyPairName": {
4+       "Description": "Public/private key pairs allow you to securely connect to your instance after it launches",
5+       "Type": "AWS::EC2::KeyPair::KeyName"
6+     },
7+     "AD1InstanceType": {
8+       "Description": "Amazon EC2 instance type for the first Active Directory Instance",
9+       "Type": "String",
10+      "Default": "m4.xlarge",
11+      "AllowedValues": [
12+        "m4.large",
13+        "m4.xlarge",
14+        "m4.2xlarge",
15+        "m4.4xlarge"
16+      ]
17+    },
18+    "AD2InstanceType": {
19+      "Description": "Amazon EC2 instance type for the second Active Directory Instance",
20+      "Type": "String",
21+      "Default": "m4.xlarge",
22+      "AllowedValues": [
23+        "m4.large",
24+        "m4.xlarge",
25+        "m4.2xlarge",
26+        "m4.4xlarge"
27+      ]
28+    }
29+  }
30+}
```

Pratiques DevOps

- Intégration continue
- Déploiement et livraison continue



Pratiques DevOps

- Monitoring et Logging
 - Suivie et analyse des métriques des logs
 - Suivie en temps réel les performances de notre infrastructure et notre application



Bénéfices DevOps



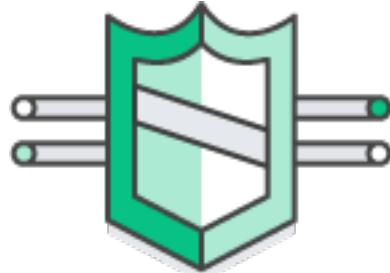
Improved Collaboration



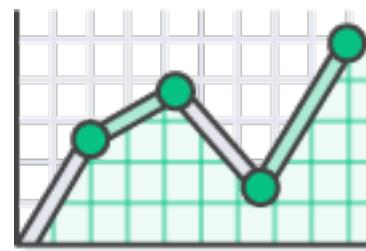
Rapid Delivery



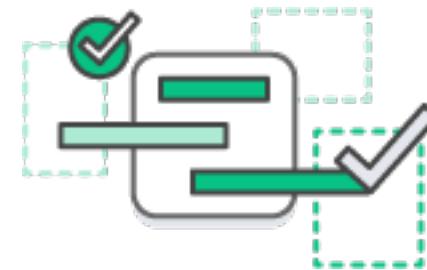
Reliability



Security



Scale

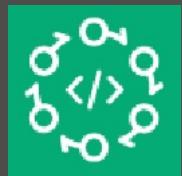


Speed

Aws DevOps outils



AWS CodeCommit



AWS CodeStar



AWS CodeBuild



AWS CodeDeploy



AWS CodePipeline



AWS CloudFormation



AWS OpsWorks



AWS Config



Amazon CloudWatch



AWS CloudTrail

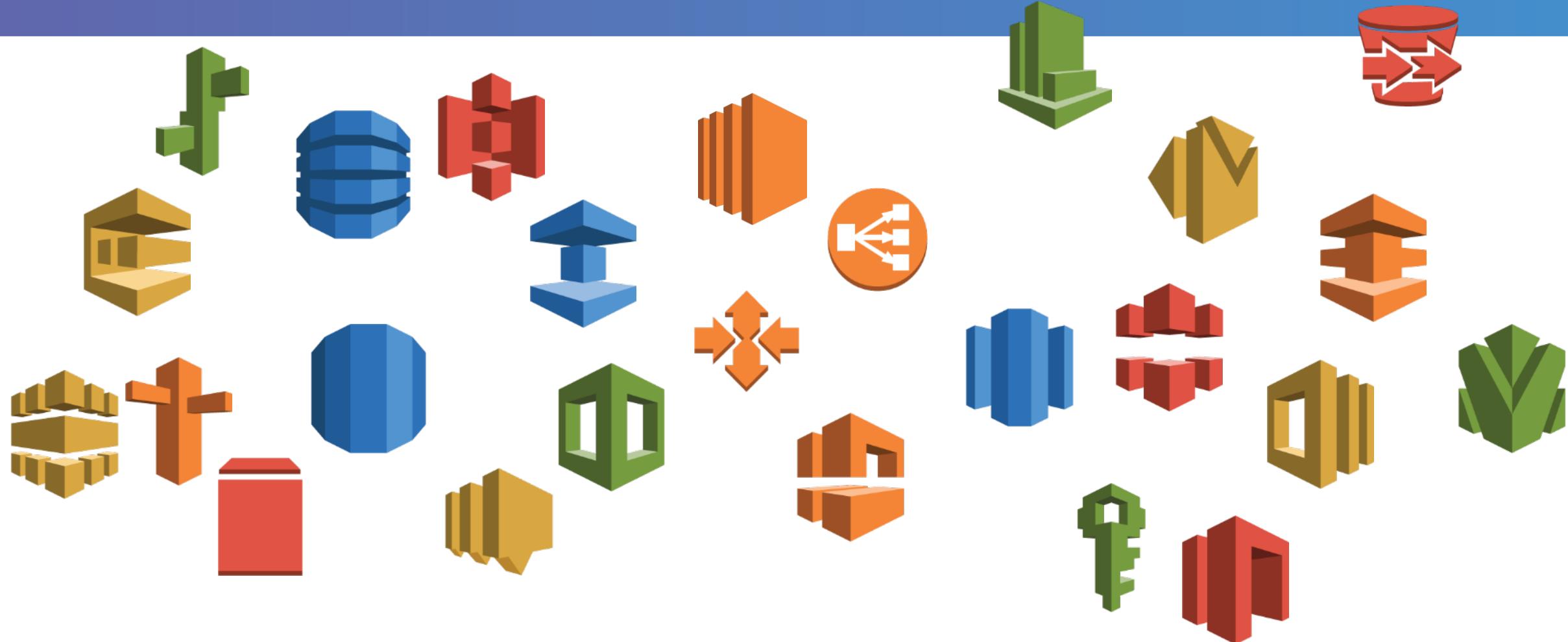


AWS Elastic Beanstalk

Aws command Line interface

- Outil managé pour la gestion des service aws en ligne de commande

Aws CLI un seul outil



Installation

Windows:

32-bit MSI: <http://s3.amazonaws.com/aws-cli/AWSCLI32.msi>

64-bit MSI: <http://s3.amazonaws.com/aws-cli/AWSCLI64.msi>

Bundled Installer: <http://aws-cli.s3.amazonaws.com/awscli-bundle.zip>

Configuration

IAM Role	Environment	Config File: <code>~/.aws/config</code>
<i>Automatic</i>	<code>AWS_ACCESS_KEY_ID</code> <code>AWS_SECRET_ACCESS_KEY</code>	<code>aws_access_key_id</code> <code>aws_secret_access_key</code>

Configuration

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [json]:
```

Configuration

```
[default]
aws_access_key_id = EXAMPLE
aws_secret_access_key = EXAMPLEKEY
region = us-west-2
output = json
```

Aws CloudFormation

- Créez et provisionnez des déploiements d'infrastructure AWS de manière prévisible et répétée
- Les ressources sont écrites dans des fichiers texte au format JSON ou YAML. On peut utiliser Cloud9 ou n'importe quel ide
- Outil de versioning pour le template
- Aide à créer les services spécifiés de manière sûre et reproductible

CloudFormation – composants et technologies

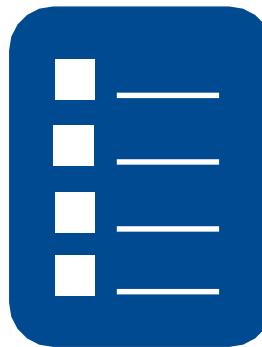
Template



JSON ou yaml
formatted file

Parameter definition
Resource creation
Configuration actions

CloudFormation

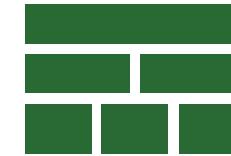


Framework

Stack creation
Stack updates

Error detection and rollback

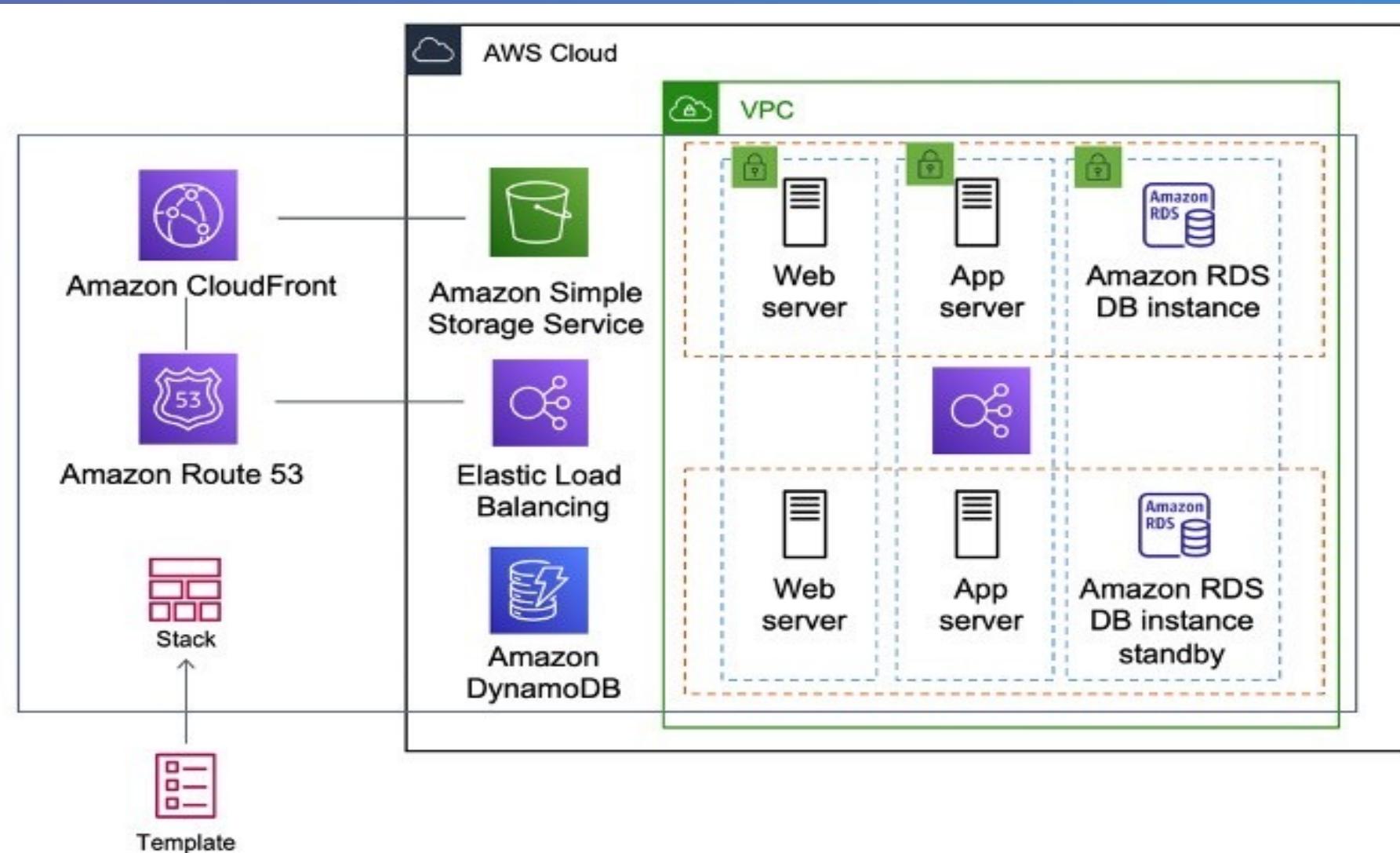
Stack



Configured AWS resources

Comprehensive service support
Service event aware
Customizable

Aws Cloud Formation



Aws Cloudformation Anatomie d'un template

```
{  
    "AWSTemplateFormatVersion" : "version date",  
    "Description" : "JSON string",  
    "Metadata" : {  
        template metadata  
    },  
    "Parameters" : {  
        set of parameters  
    },  
    "Mappings" : {  
        set of mappings  
    },  
    "Conditions" : {  
        set of conditions  
    },  
    "Transform" : {  
        set of transforms  
    },  
    "Resources" : {  
        set of resources  
    },  
    "Outputs" : {  
        set of outputs  
    }  
}
```

```
AWSTemplateFormatVersion: "version date"  
  
Description:  
String  
  
Metadata:  
template metadata  
  
Parameters:  
set of parameters  
  
Mappings:  
set of mappings  
  
Conditions:  
set of conditions  
  
Transform:  
set of transforms  
  
Resources:  
set of resources  
  
Outputs:  
set of outputs
```

Aws Cloudformation Anatomie d'un template

Parameters:

```
"myInstanceType" : {  
    "Type" : "String",  
    "Default" : "t2.large",  
    "AllowedValues" : ["t2.micro", "t2.small",  
    "t2.medium", "t2.large"],  
    "Description" : "Instance type for instances created,  
    must be in the t2 family."  
}
```

Mappings:

```
"AWSInstanceType2Virt": {  
    "t2.micro": {"Virt": "HVM"},  
    "t2.small": {"Virt": "HVM"},  
    "t2.medium": {"Virt": "HVM"},  
    "t2.large": {"Virt": "HVM"},  
}
```

Mappings:

```
"AWSRegionVirt2AMI": {  
    "us-east-1": {  
        "PVM": "ami-50842d38",  
        "HVM": "ami-08842d60"  
    },  
    "us-west-2": {  
        "PVM": "ami-af86c69f",  
        "HVM": "ami-8786c6b7"  
    },  
    "us-west-1": {  
        "PVM": "ami-c7a8a182",  
        "HVM": "ami-cfa8a18a"  
    }  
}
```

Aws CloudFormation – Ressources

- Amazon EC2
- Amazon EC2 Container Service
- Amazon EC2 Simple Systems Manager (**New**)
- AWS Lambda (including event sources)
- Auto Scaling (including Spot Fleet)

- Amazon VPC
- Elastic Load Balancing
- Amazon Route 53
- Amazon CloudFront
- AWS WAF (**New**)

- Amazon RDS
- Amazon Redshift
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon RDS (including Aurora)
- Amazon S3

- Amazon IAM (including managed policies)
- Amazon SimpleAD / MicrosoftAD (**New**)

- Amazon Kinesis
- Amazon SNS
- Amazon SQS

- AWS CloudTrail
- Amazon CloudWatch
- AWS Config (**New**)
- AWS Key Management Service (**New**)

- AWS Data Pipeline
- AWS Elastic Beanstalk
- AWS OpsWorks
- AWS CodeDeploy
- AWS CodePipeline (**New**)

- Amazon Workspaces

AWS CloudFormation versionning

Pour suivre le changement dans le code

Qu'est-ce qui change ?

Qui a fait ce changement ?

Quand a-t-il été fabriqué ?

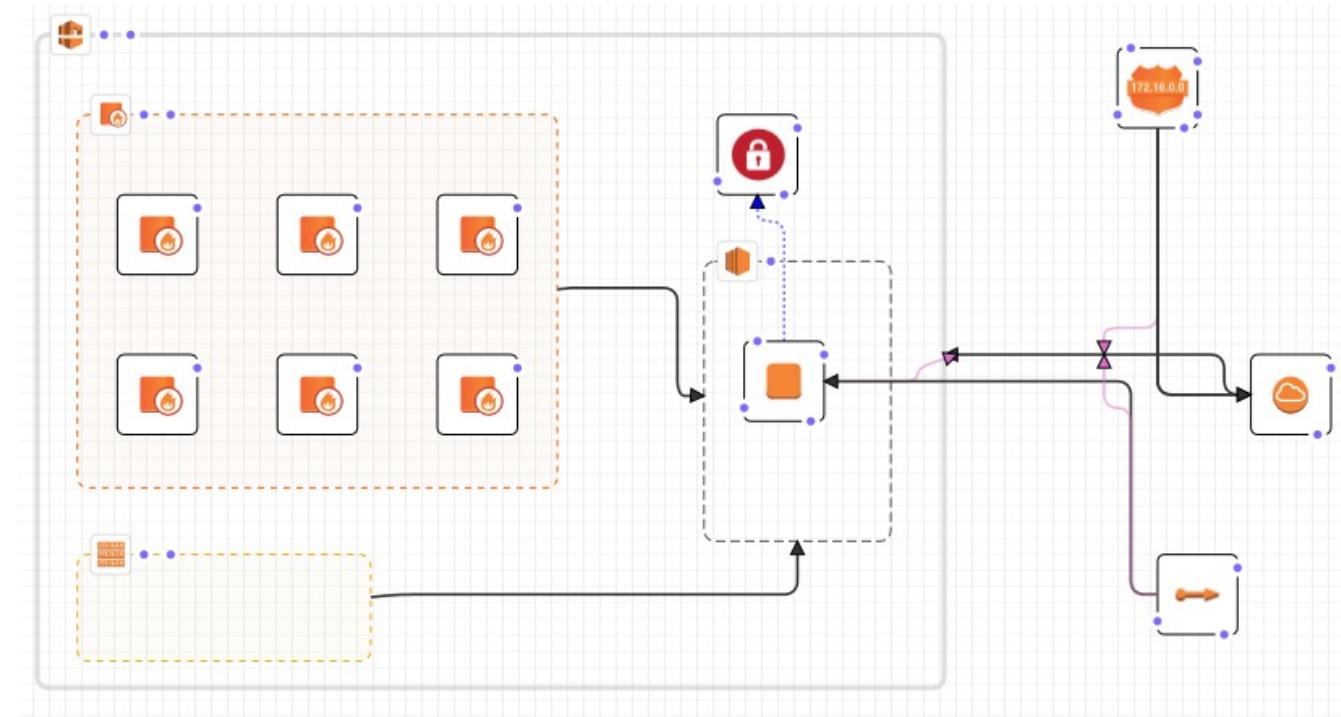
Pourquoi a-t-il été créé ? (

AWS CloudFormation Testing

- Validate via API/CLI
 - \$ aws cloudformation validate-template – confirm CF syntax
- Utiliser une pipeline d'intgration continue pour tester le code

Aws CloudFormation Designer

- Visualisation du template des ressources
- Modification du template avec drag and drop



AWS CloudFormation – Déploiement et mise à jour

- En utilisant le portail

Ou

- aws cloudformation create-stack --stack-name myteststack --template-body file:///home//local//test//sampletemplate.json

-- parameters

ParameterKey=string,ParameterValue=string

Aws CloudFormation

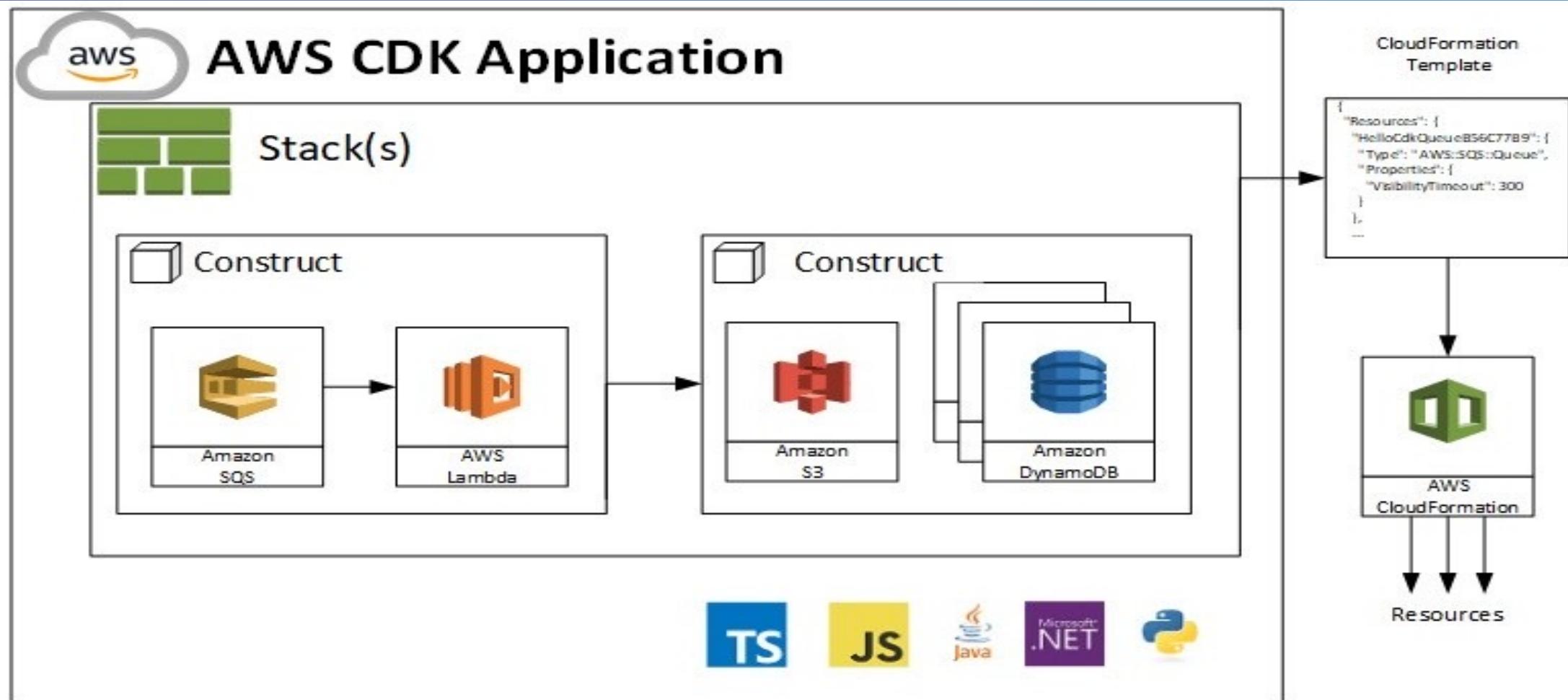
- Démo

AWS CDK

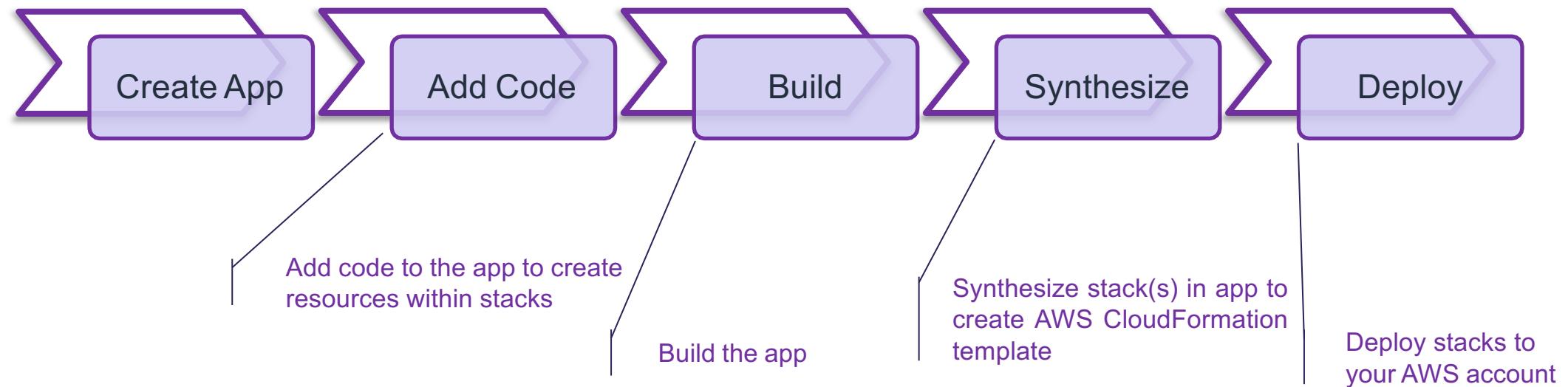


- AWS CDK nous permet de créer nos ressources à l'aide de notre langage de programmation
- AWS CDK supports TypeScript, JavaScript, Python, Java, C#/.Net, and (in developer preview) Go
- AWS CDK permet de définir les ressources en tant qu'objets réutilisables
- Les objets forment une stack dans notre application cdk

AWS CDK



AWS CDK Workflow



AWS CDK Toolkit

List des commandes AWS CDK

- `cdk lists (ls)` – Lists the stacks in the app
- `cdk synthesize (synth)` – Synthesizes and prints the CloudFormation template for the specified stack(s)
- `cdk bootstrap` – Deploys the CDK Toolkit stack
- `cdk deploy` - Deploys the specified stack(s)
- `cdk destroy` - Destroys the specified stack(s)
- `cdk diff` – Compares the stack with deployed or local CloudFormation template
- `cdk metadata` – Displays metadata about the specified stack
- `cdk init` – Creates a new CDK project in the current directory from a specified template
- `cdk context` - Manages cached context values
- `cdk docs (doc)` – Opens the CDK API reference in your browser
- `cdk doctor` - Checks your CDK project for potential problems

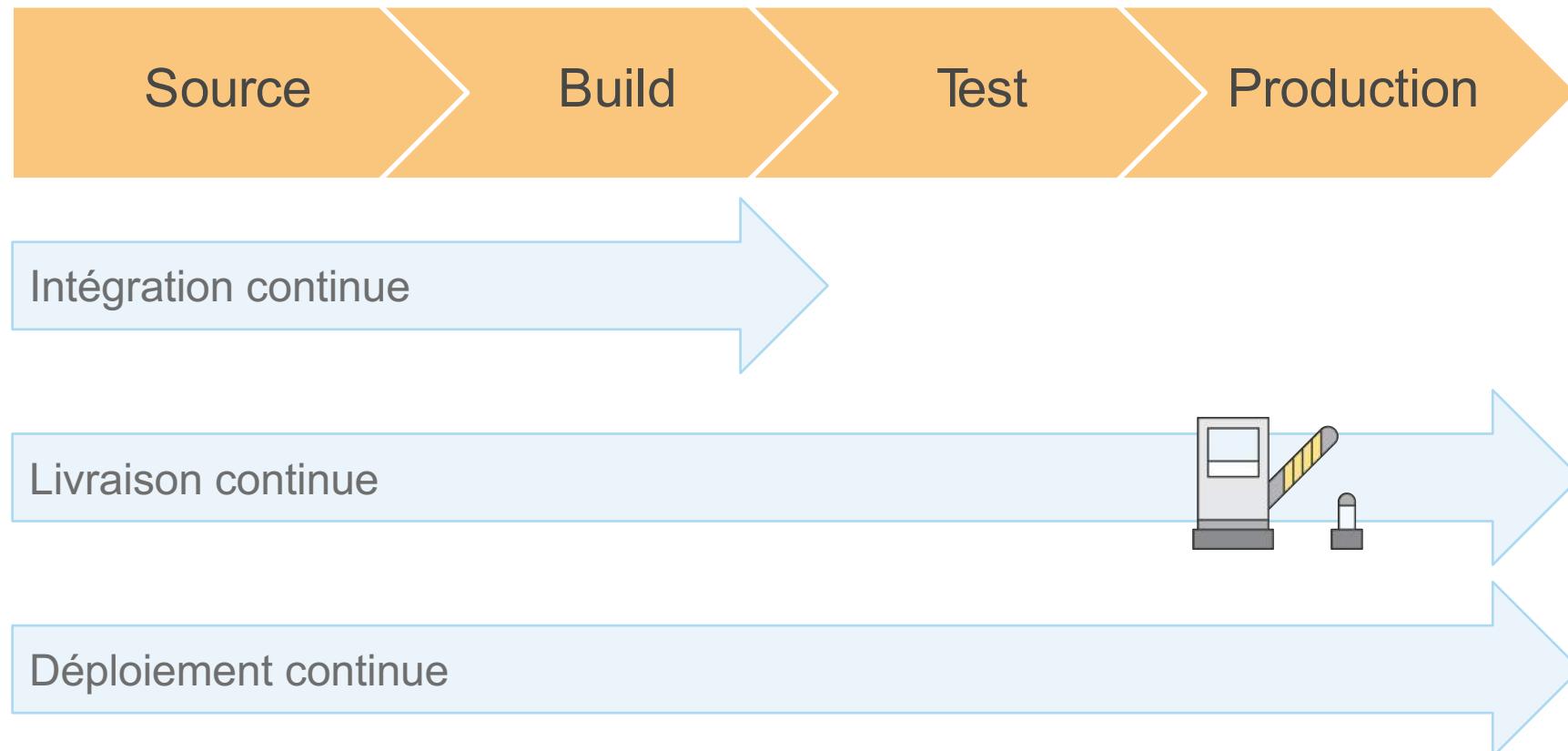
Aws CDK

- Démo

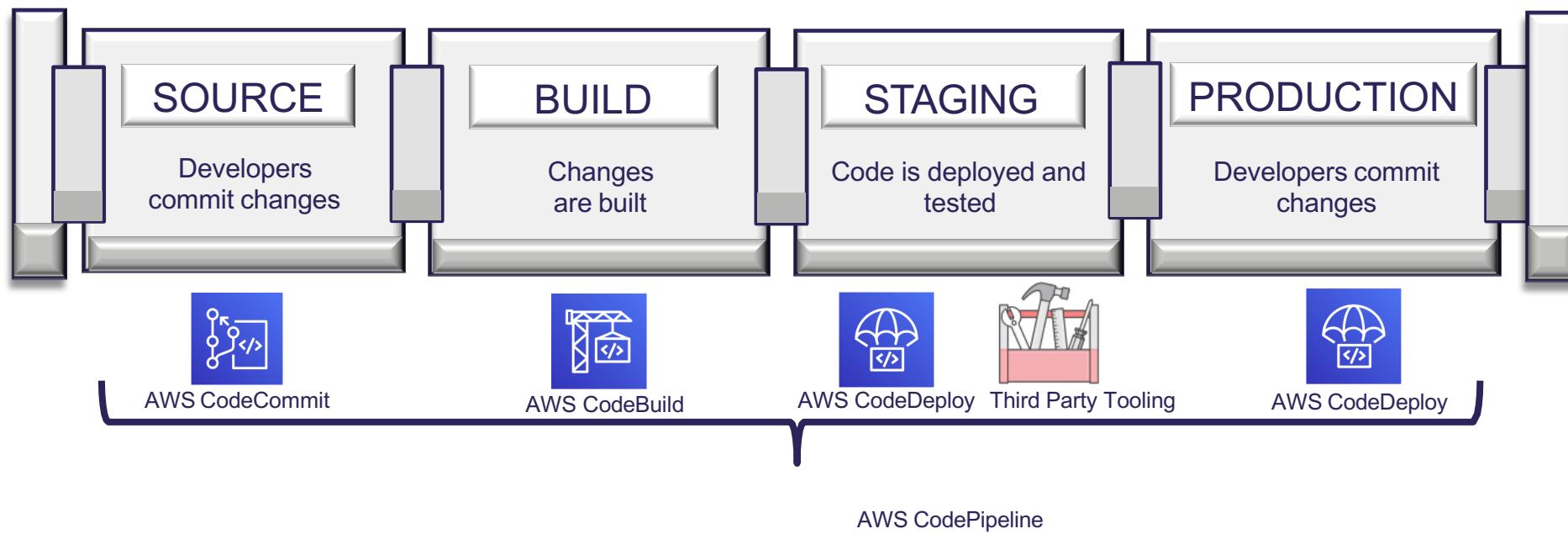
TP Infrastructure en Code

- Créer un serveur web sur une ressource de type instance ec2

Aws Intégration et livraison continue



Aws Intégration et livraison continue



Aws CodeCommit

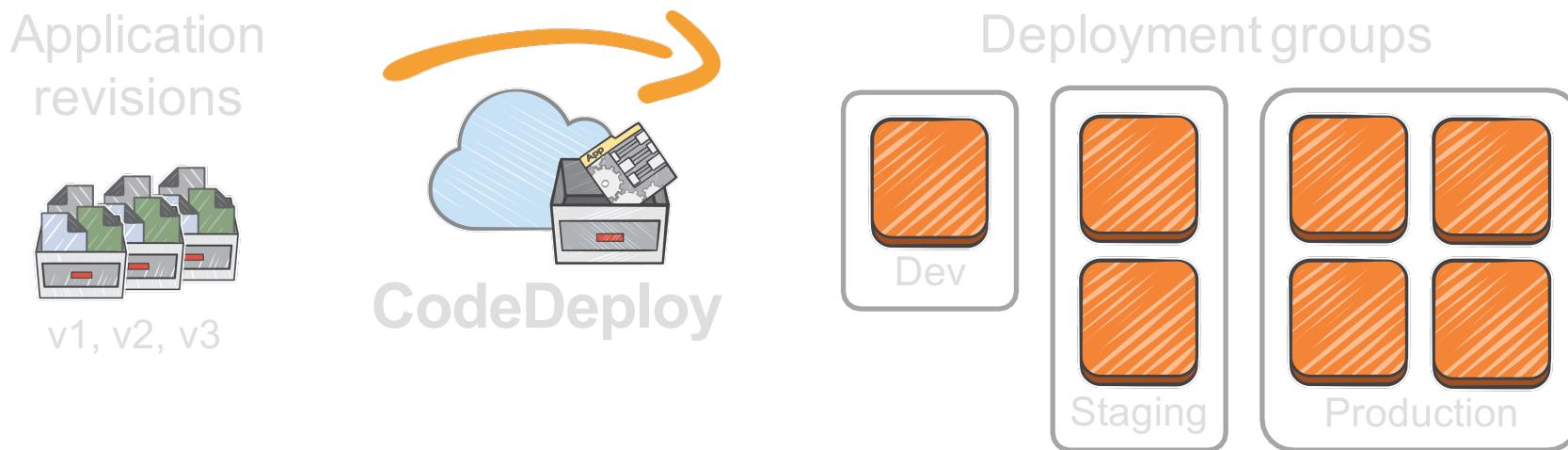


- CodeCommit offre une redondance sur plusieurs zones.
- CodeCommit permet un cryptage des données.
- CodeCommit intègre les IAM.
- CodeCommit permet une utilisation de repos illimité.

AWS CodeBuild

- Aws CodeBuild permet d'automatiser le build de nos applications.
- Entièrement géré : CodeBuild élimine la nécessité de configurer, corriger, mettre à jour et gérer vos serveurs d'automatisation.
- Utilisable à la demande : CodeBuild s'adapte à la demande selon vos besoins de génération. Vous ne payez que pour le nombre de minutes de génération que vous consommez.
- Prêt à l'emploi : CodeBuild fournit des environnements de génération préconfigurés pour les langages de programmation les plus courants. Il vous suffit de pointer sur votre script de génération pour démarrer votre première génération.
- Intégré : CodeBuild s'intègre simplement avec tous les services AWS.

AWS CODEDEPLOY



- Scale le déploiement sur plusieurs instances
- Déploiement sans temps d'arrêt
- Centralise le déploiement et le monitoring

Step 1: Package your application (with an AppSpec file)

```
version: 0.0
os: linux
files:
  - source: chef/
    destination: /etc/chef/codedeploy
  - source: target/hello.war
    destination: /var/lib/tomcat6/webapps
hooks:
  ApplicationStop:
    - location: deploy_hooks/stop-tomcat.sh
  BeforeInstall:
    - location: deploy_hooks/install-chef.sh
  AfterInstall:
    - location: deploy_hooks/librarian-install.sh
  ApplicationStart:
    - location: deploy_hooks/chef-solo.sh
  ValidateService:
    - location: deploy_hooks/verify_service.sh
```

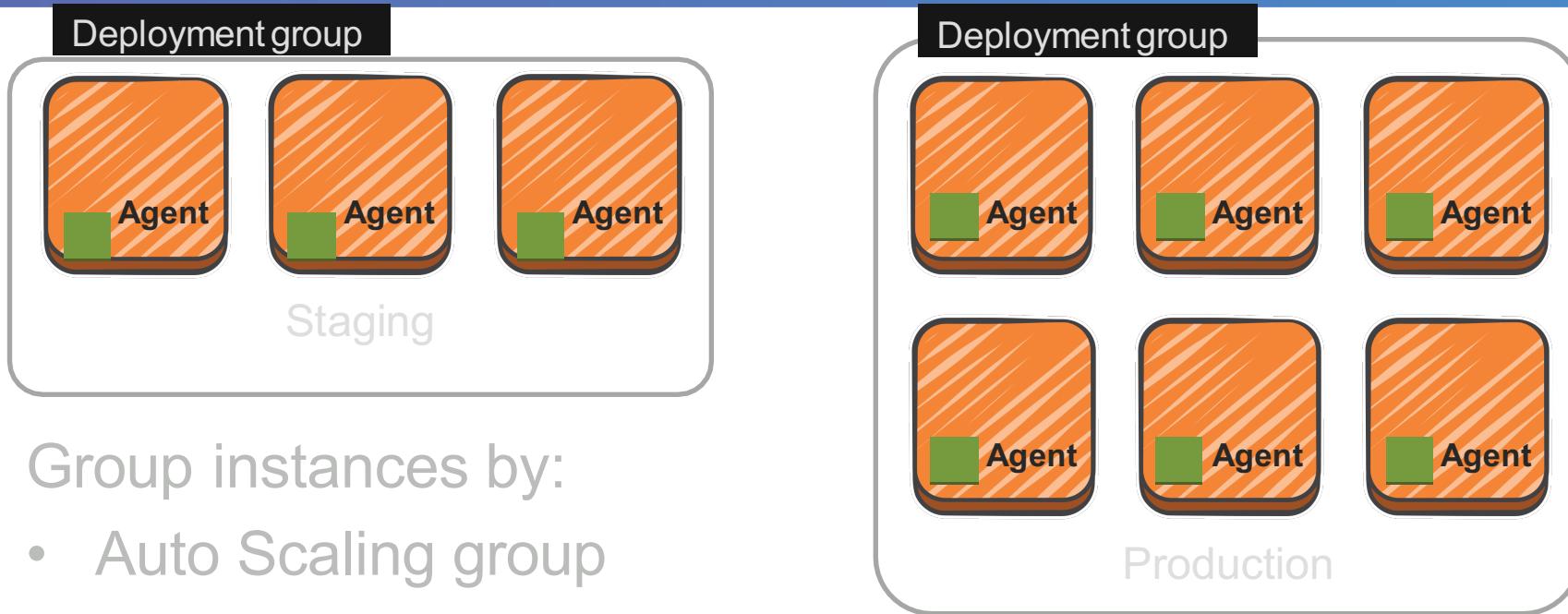
Step 1: Package your application (with an AppSpec file)

```
version: 0.0
os: linux
files:
  - source: chef/
    destination: /etc/chef/codedeploy
  - source: target/hello.war
    destination: /var/lib/tomcat6/webapps
hooks:
  ApplicationStop:
    - location: deploy_hooks/stop-tomcat.sh
  BeforeInstall:
    - location: deploy_hooks/install-chef.sh
  AfterInstall:
    - location: deploy_hooks/librarian-install.sh
  ApplicationStart:
    - location: deploy_hooks/chef-solo.sh
  ValidateService:
    - location: deploy_hooks/verify_service.sh
```

Step 1: Package your application (with an AppSpec file)

```
version: 0.0
os: linux
files:
  - source: chef/
    destination: /etc/chef/codedeploy
  - source: target/hello.war
    destination: /var/lib/tomcat6/webapps
hooks:
  ApplicationStop:
    - location: deploy_hooks/stop-tomcat.sh
  BeforeInstall:
    - location: deploy_hooks/install-chef.sh
  AfterInstall:
    - location: deploy_hooks/librarian-install.sh
  ApplicationStart:
    - location: deploy_hooks/chef-solo.sh
  ValidateService:
    - location: deploy_hooks/verify_service.sh
```

Step 2: Set up your target environments



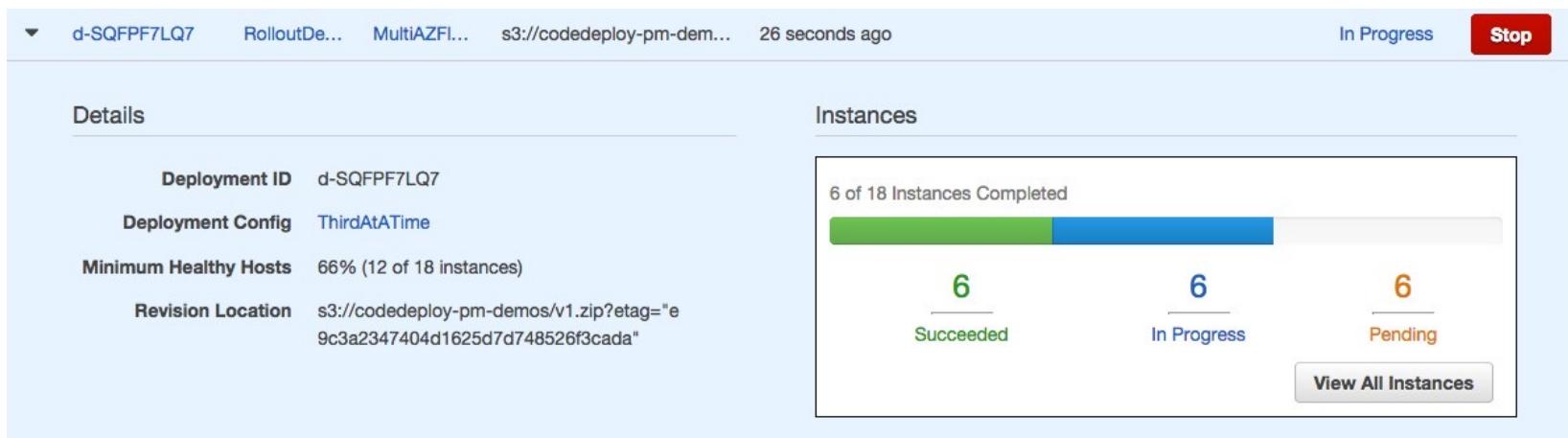
Group instances by:

- Auto Scaling group
- Amazon EC2 tag
- On-premises tag

Step 3: Deploy!

AWS CLI & SDKs
AWS Console
CI / CD Partners
GitHub

```
aws deploy create-deployment \
--application-name MyApp \
--deployment-group-name TargetGroup \
--s3-location bucket=MyBucket,key=MyApp.zip
```



AWS CodePipeline



TP Intégration et déploiement continu

- Déployer une application web en utilisant la pipeline aws sur des instance EC2

Rappel Pattern MicroService et pattern associé

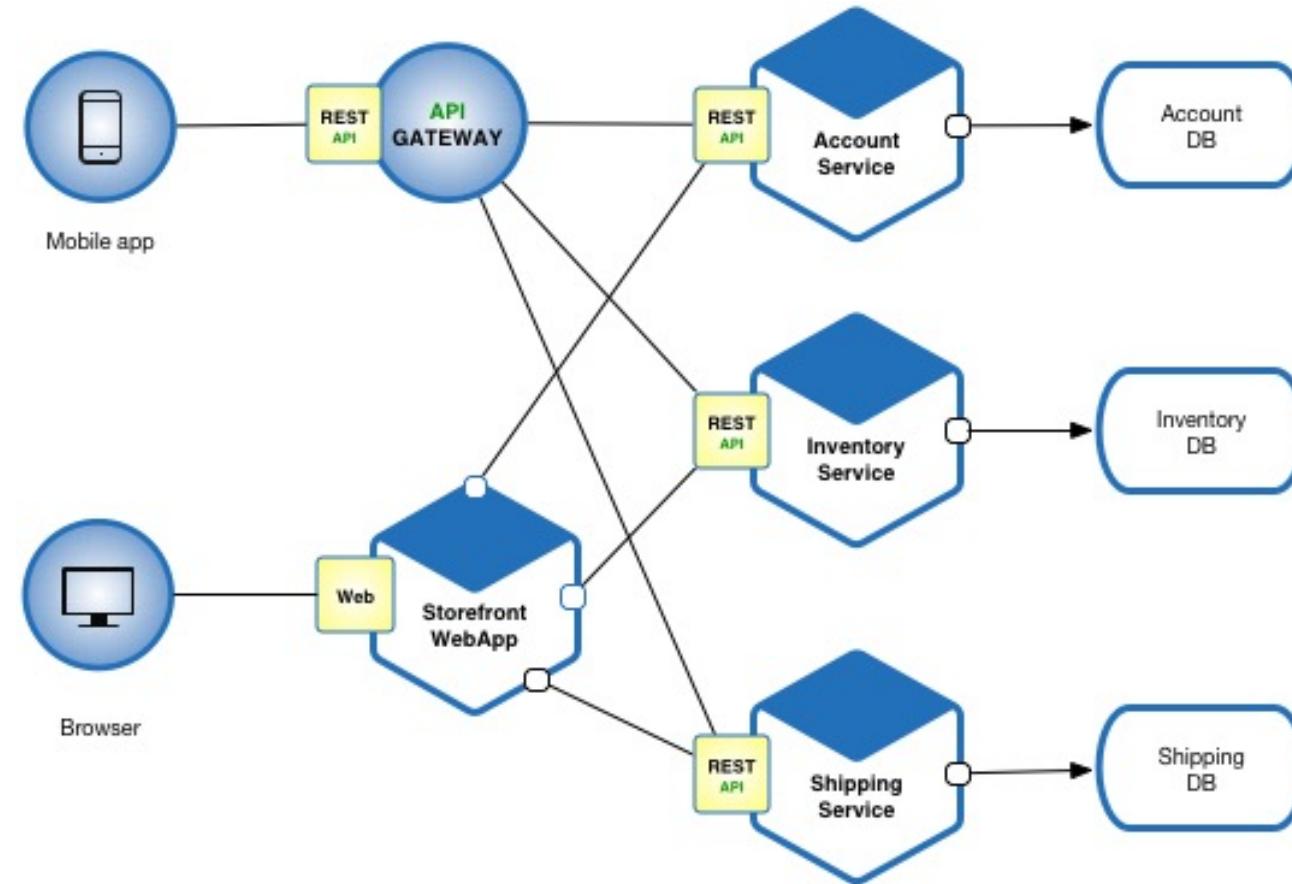
- Pourquoi choisir un Pattern MicroService ?
- Dans un contexte de réalisation d'une application qui :
 - Utilise plusieurs clients(mobile, Desktop, web).
 - Expose une API Public
 - Interagit avec d'autre application par système de messagerie
 - Exécute différent logiques métier et interagit avec plusieurs source de données.
 - Gère plusieurs protocole de communication.
- Avec comme condition :
- Possibilité de travailler à plusieurs équipes
- Facilement maintenable
- Facilement scalable
- Facilement testable.

Rappel Pattern MicroService et pattern associé

- Solution :
 - Une approche de développement par services fortement découplés
 - Chaque service déployable indépendamment des autres
 - Chaque service totalement autonome
 - Chaque service développé indépendamment des autres

Rappel Pattern MicroService et pattern associé

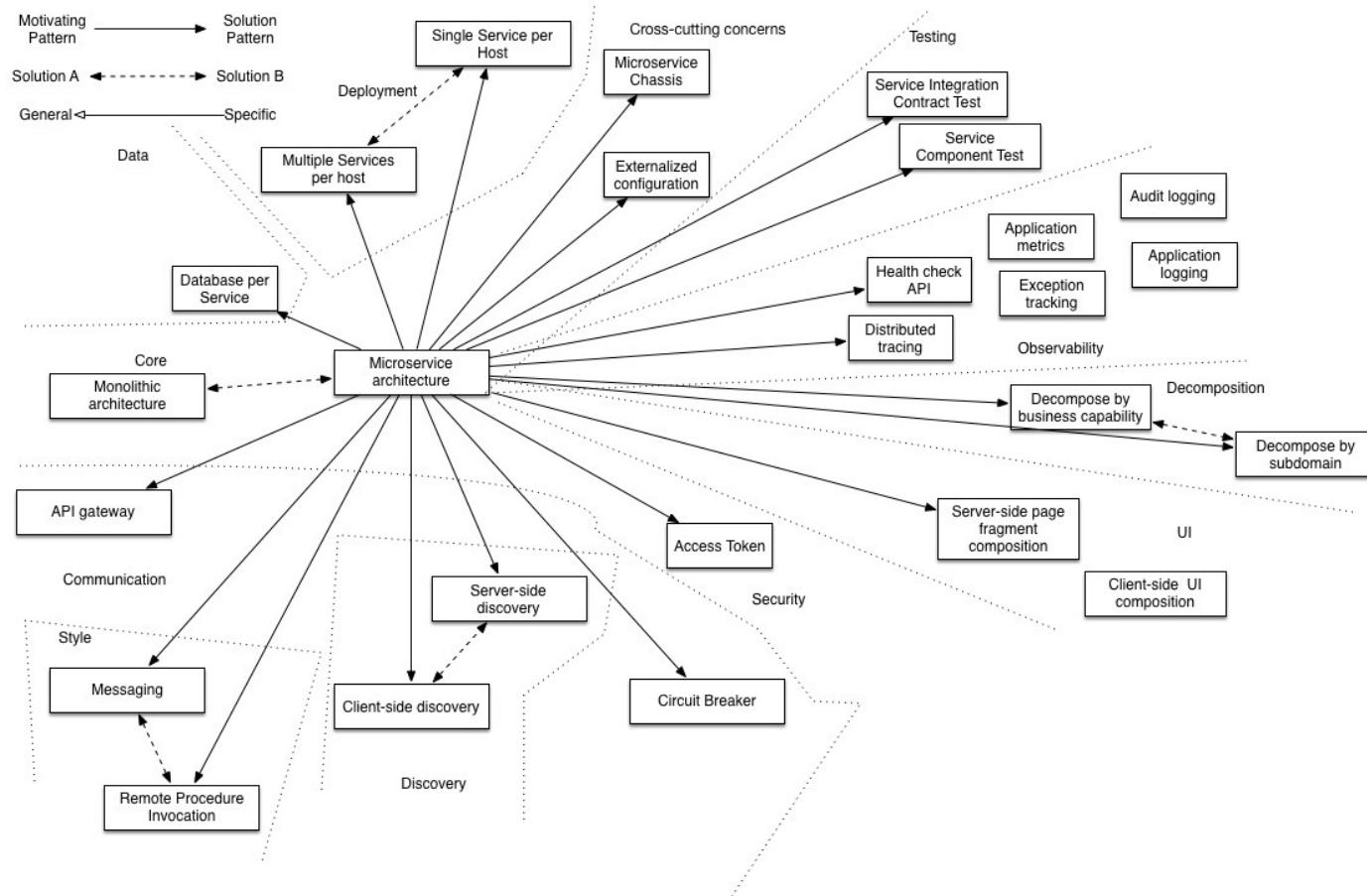
- Exemple site Ecommerce



Rappel Pattern MicroService et pattern associé

- Problématiques liées à l'utilisation des mircoservices
- Problématiques de développement
- Problématiques de déploiement
- Problématiques de sécurités

Rappel Pattern MicroService et pattern associé



Décomposition des services

- Pour profiter des avantages d'une architecture microservice il faut réussir à découper notre application en service assez petit pour être facilement testable, développé par une petite équipe et également facilement déployable.
- Chaque service doit apporter une plus value sans dépendre des autres services.
- Pour réussir notre décomposition, on peut utiliser :
- Décomposition par Capacité métier
- Décomposition par sous domaine

Décomposition par capacité métier

- Décomposition par capacité Métier est un sous pattern de la modélisation d'architecture d'entreprise.
- Capacité métier est tout élément qui apporte de la valeur ajoutée à notre application
- Le but est de décomposer l'application en service qui apporte de la valeur.
- Exemple E-Commerce :
- Products management
- Cart management
- Shipping Management
- Order Management
- Payment management
- Shipping Management
- Notifications (Email/SMS)Management

Décomposition par DDD

- Ce modèle modélise les microservices autour du DDD (Domain-Driven Design).
- DDD est une approche de développement logiciel qui repose sur les principes et les idées de l'analyse et de la conception orientées objet.
- Dans DDD, un modèle de domaine utilise les connaissances sur un sous domaine pour résoudre le problème sur un domaine.
- Dans DDD, tout le monde utilise le même vocabulaire dans les équipes.
- Ceci est connu sous le nom de «langage omniprésent». en termes DDD.
- Exemple Ecommerce

Communication Entre Microservice

- Pour faire communiquer deux microservices, on peut utiliser :
- Communication en http avec des API Rest
- Communication avec un protocole gRpc
- Communication avec broker de messages

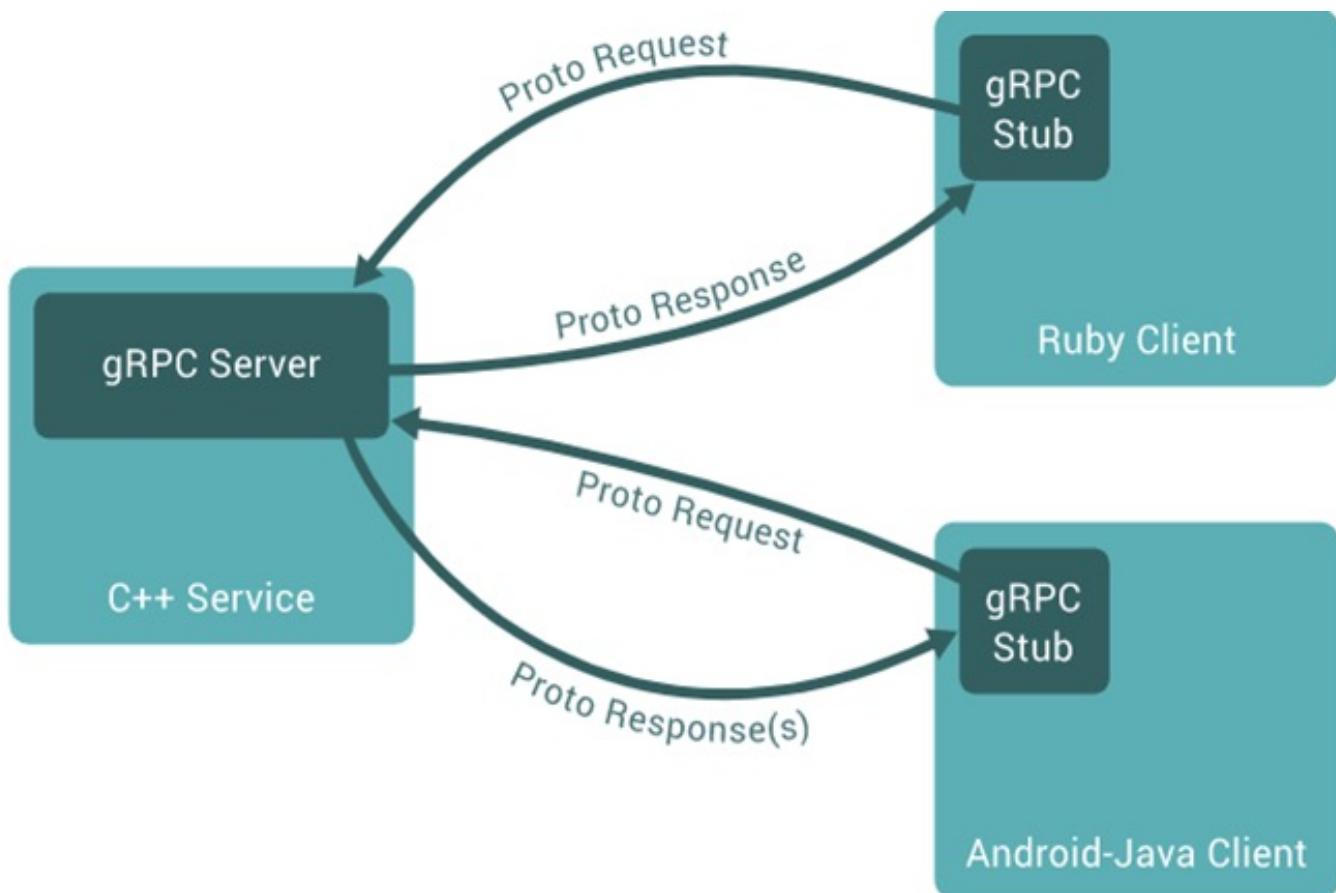
Communication Entre Microservice API REST

- Dans le cadre d'une application microservices, nos services sont en interaction en eux.
- On peut utiliser une communication en REST basée sur principe d'une requête, réponse
- Utilisation d'une communication en REST impose que les deux services soit actifs
- Utilisation d'une communication en REST impose une communication unidirectionnelle

Communication Entre Microservice gRPC

- gRPC est un framework de communication openSource développé par google
- gRPC offre des performances très élevées
- gRPC utilise HTTP/2 pour transporter des messages binaires
- gRPC utilise un mécanisme de contrat de service défini à l'aide d'un Protocol Buffers
- Utilisation du framework gRPC permet l'utilisation de client et server issues de plusieurs technologies

Communication Entre Microservice gRPC



Communication Entre Microservice gRPC

- Protocol buffers est un mécanisme open source développé par Google qui permet la sérialisation et la structuration des données
- Protocol buffers permet de structurer les données sous forme de messages
- Chaque message représente une petite unité logique

Communication Entre Microservice gRPC

- 1- Définition du protoBuf
- 2- Compilation du ProtoBuf
- 3- Génération des services
- 4- Implémentation soit client ou serveur

Communication Entre Microservice Broker message

- La communication par Broker de message est asynchrone
- La communication par Broker de message est très fiable et garantie une stabilité dans l'utilisation
- Le choix du Broker se fait en fonction de plusieurs critères
 - Scalabilité du courtier - Le nombre de messages envoyés par seconde dans le système.
 - Persistance des données - La possibilité de récupérer des messages.
 - Capacité du consommateur - Indique si le courtier est capable de gérer des consommateurs un à un et / ou un à plusieurs.

Communication Entre Microservice Broker message

- RabbitMQ
- Redis
- kafka
- Démo kafka

Gestion de base de données en Microservice

- Dans toute application, il faut gérer une persistance dans une base de données
- Pour une architecture Microservice, la gestion des données peut se faire de plusieurs façon toute en respectant les paradigme du pattern

Gestion de base de données en Microservice

- Première solution :
- Chaque service gère sa propre persistance, accessible uniquement en API, à travers :
- Tables privées uniquement par service
- Schéma par service
- Base de données par service

Gestion de base de données en Microservice

- L'utilisation d'une base de données par service présente les avantages suivants :
 - Aide à garantir que les services sont faiblement couplés. Les modifications apportées à la base de données d'un service n'ont aucun impact sur les autres services.
 - Chaque service peut utiliser le type de base de données le mieux adapté à ses besoins. Par exemple, un service qui effectue des recherches de texte pourrait utiliser ElasticSearch. Un service qui manipule un graphe social pourrait utiliser Neo4j.
- L'utilisation d'une base de données par service présente les inconvénients suivants :
 - La mise en œuvre de transactions couvrant plusieurs services n'est pas simple.
 - La mise en œuvre de requêtes qui joignent des données qui se trouvent désormais dans plusieurs bases de données est un défi.
 - Complexité de la gestion de plusieurs bases de données SQL et NoSQL

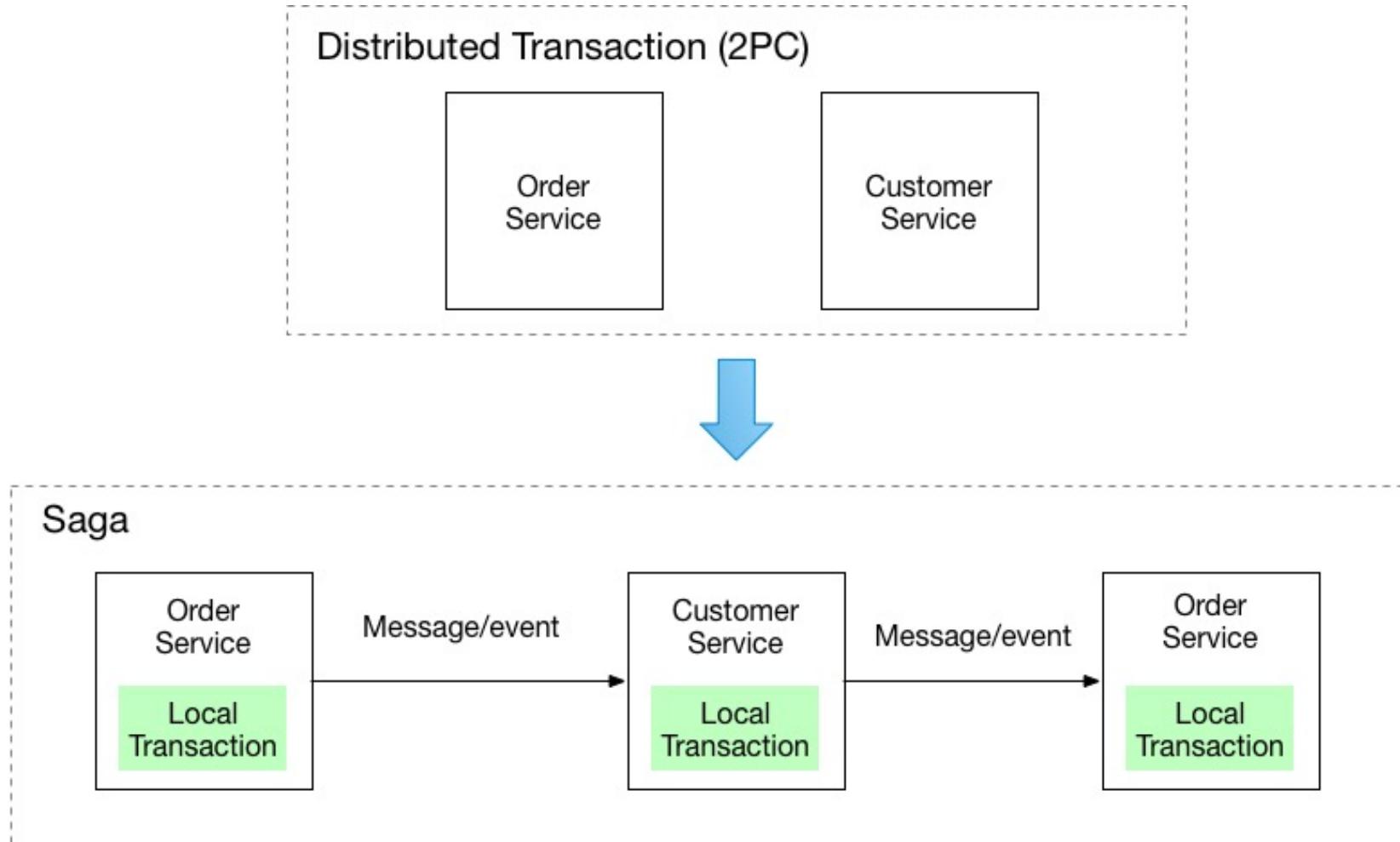
Gestion de base de données en Microservice

- Deuxième solution :
- Avoir des données partagées entre les services
- Les avantages de ce modèle sont :
 - Un développeur utilise des transactions et procédures simples pour assurer la cohérence des données
 - Une seule base de données est plus simple à exploiter
- Les inconvénients de ce modèle sont :
 - Couplage du développement - un développeur travaillant sur le service de commande devra coordonner les changements de schéma avec les développeurs d'autres services qui accèdent aux mêmes tables. Ce couplage et cette coordination supplémentaire ralentiront le développement.
 - Couplage d'exécution - parce que tous les services accèdent à la même base de données, ils peuvent potentiellement interférer les uns avec les autres. Par exemple, si une longue transaction CustomerService détient un verrou sur la table ORDER, le OrderService sera bloqué.
 - Une seule base de données peut ne pas répondre aux exigences de stockage de données et d'accès de tous les services.

Microservice SAGA

- Dans le cadre d'une base de données par service, pour résoudre la problématique des transactions cross services, on peut utiliser le pattern SAGA
- Une Saga est une séquence de transactions locales.
- Chaque transaction locale procède à la mise à jour de la base de données et publie un message pour déclencher la prochaine transaction locale
- Si une transaction locale échoue, on exécute des transactions de compensation qui annulent les transactions locales
Implement each business transaction that spans multiple services is a saga. A saga is a sequence of local transactions. Each local transaction updates the database and publishes a message or event to trigger the next local transaction in the saga. If a local transaction fails because it violates a business rule then the saga executes a series of compensating transactions that undo the changes that were made by the preceding local transactions. précédentes

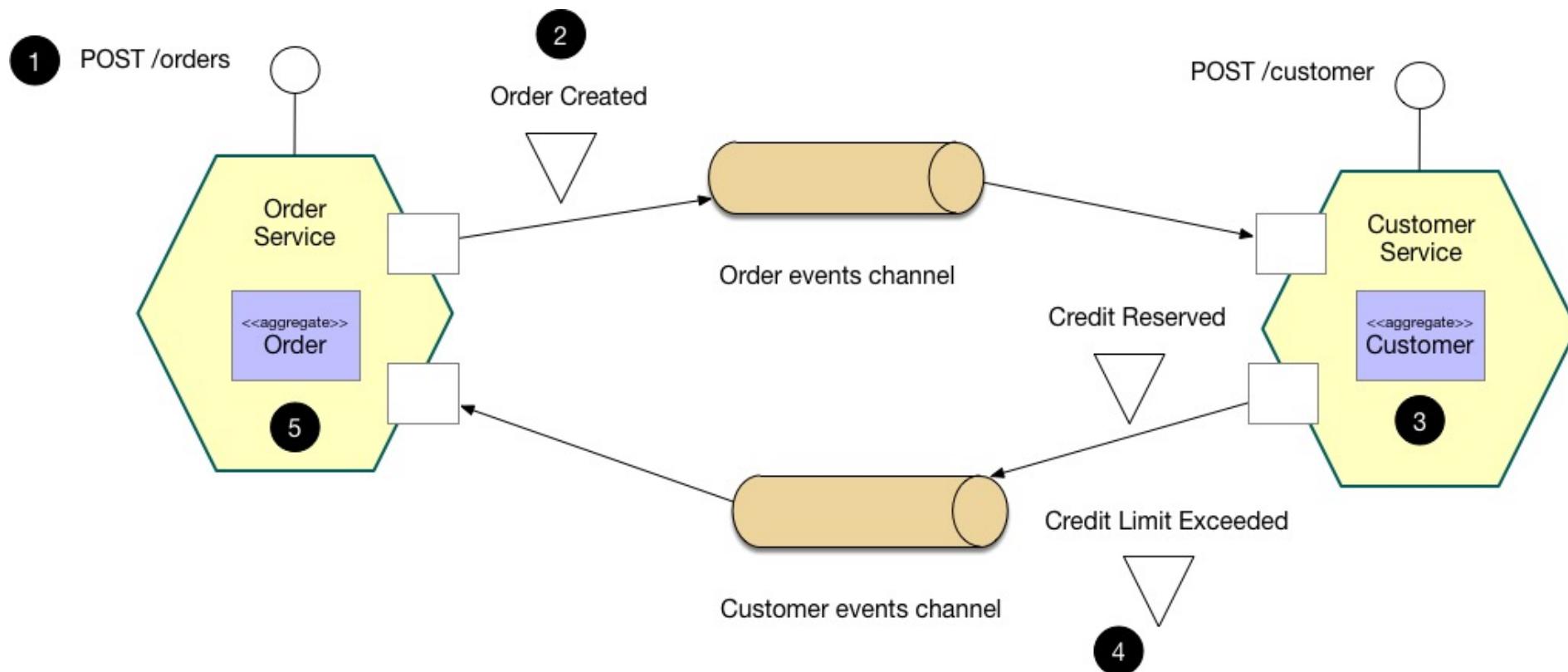
Microservice SAGA



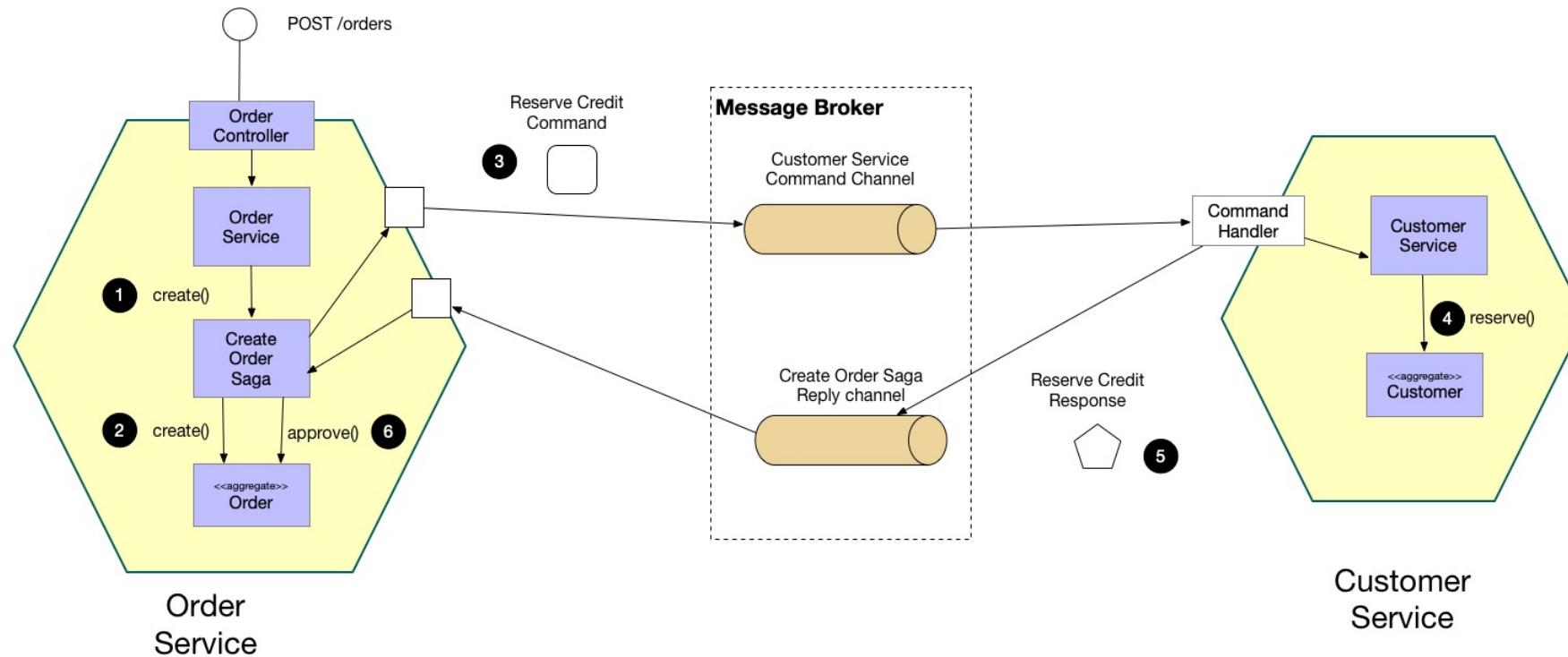
Microservice SAGA

- On peut implémenter le pattern SAGA Soit de manière :
 - Chorégraphique
 - Par orchestration

Microservice SAGA Chorégraphique



Microservice SAGA Orchestration

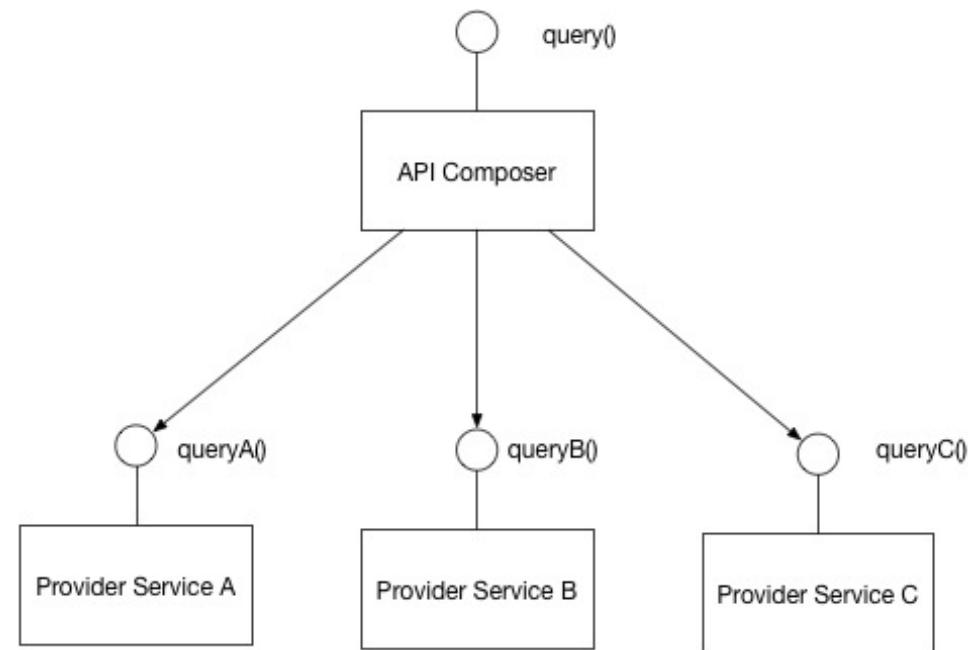


Requête cross services

- Dans une architecture microservices, avec une base de données par service (par exemple), il devient compliquer de faire des jointures sur des tables relationnelles (par exemple).
- Pour répondre à cette problématique, on trouve une multitude de patterns par exemple:
- API Composition
- Command Query Responsibility Segregation (CQRS)

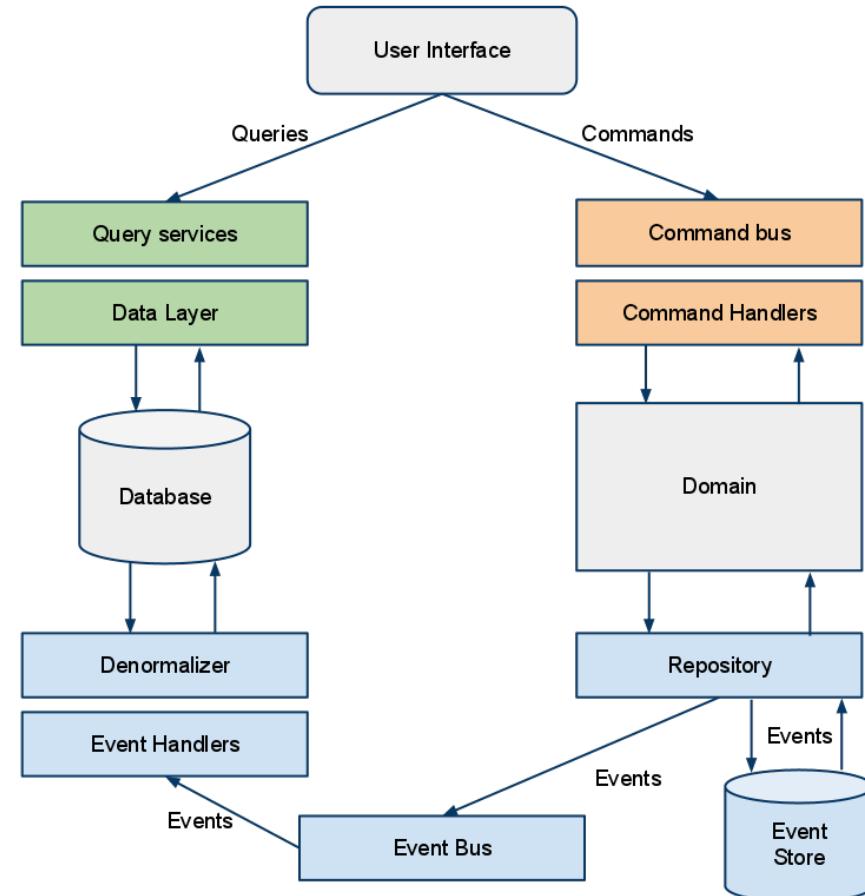
Requête cross services (API Composition)

- API Composition est un pattern qui consiste, dans le cadre d'une requête sur plusieurs services, à utiliser un service intermédiaire qui appelle chaque services et procède à une jointure.



Requête cross services (CQRS)

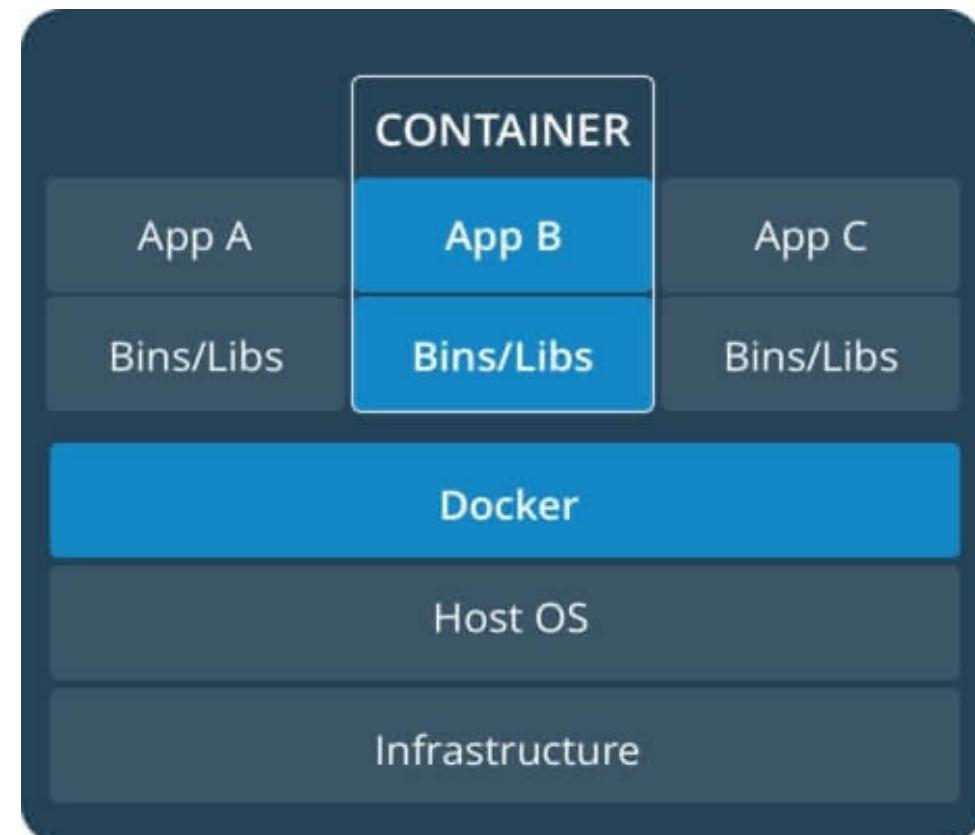
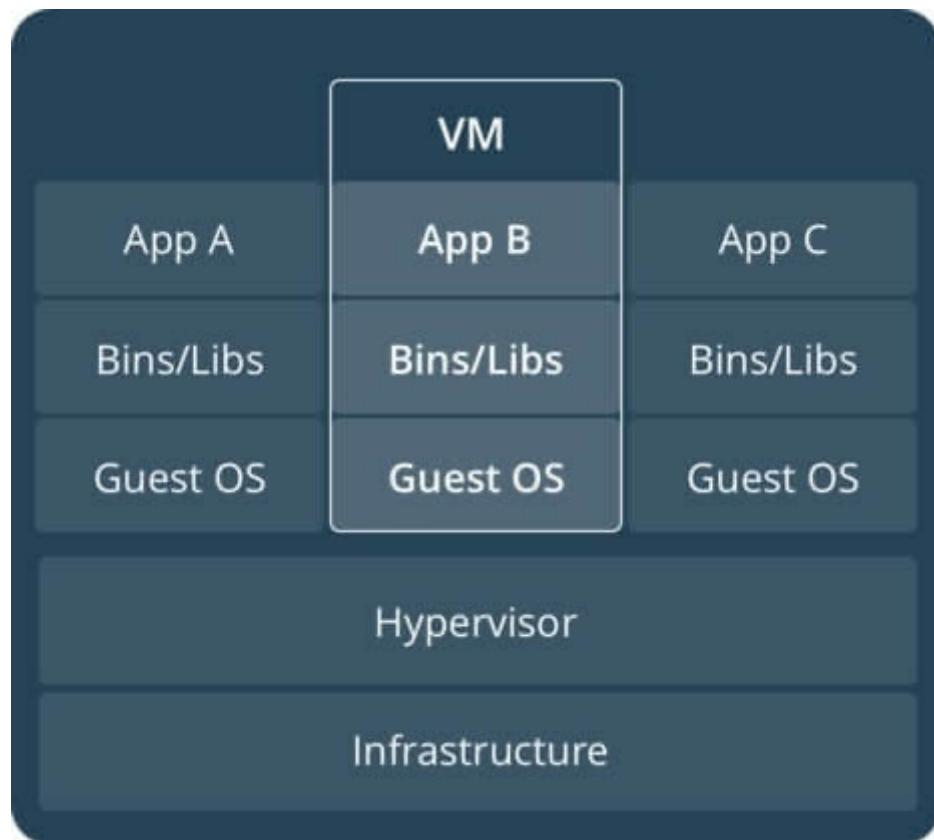
- CQRS consiste à avoir un microservice dédié aux requêtes associé à une vue de nos données (en mode lecture seul)
- La vue est régulièrement mise à jour à partir d'évènement qui écoute des commandes dans les autres microservices



DOCKER – Isolation

- **Isolation en VM:**
 - Se fait au niveau matérielle
 - Accès virtuel aux ressources via l'hôte à l'aide de l'hyperviseur
- **Isolation Dans la conteneurisation:**
 - Se fait au niveau du système d'exploitation
 - Exécution native sur linux et partage du noyau hôte avec les conteneurs
- **Conclusion**
 - La virtualisation est une technologie avec beaucoup d'avantages mais avec certain inconvénients bloquant, d'où la nécessité de pousser vers une nouvelle technologie qui répond à ces inconvénients

DOCKER – Isolation

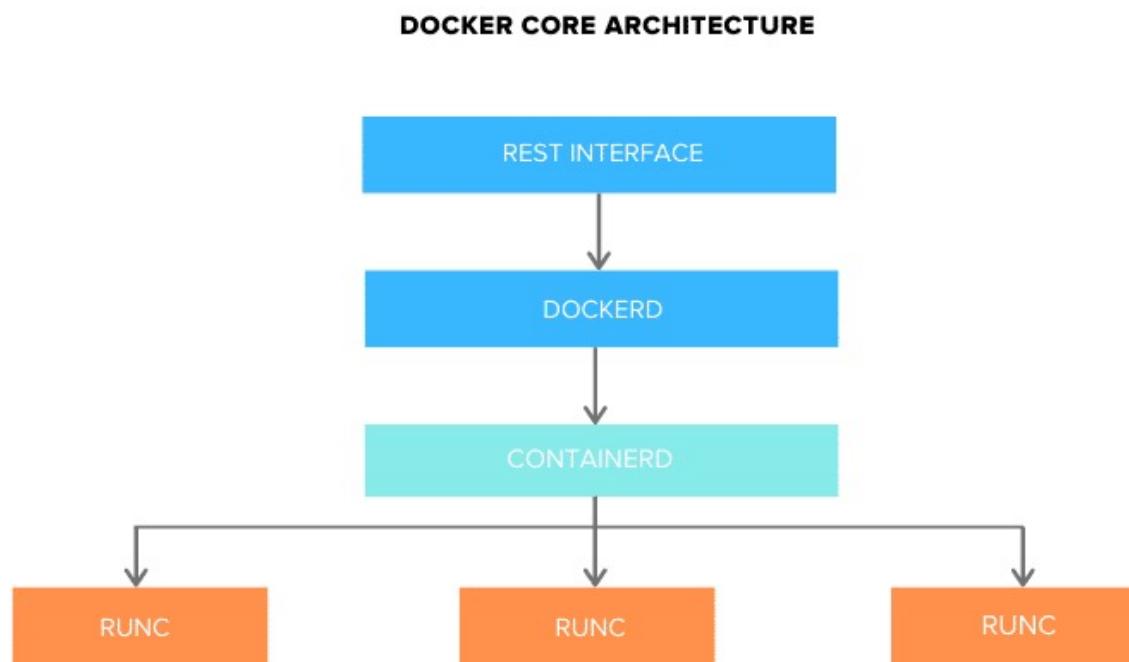


DOCKER – Avantages

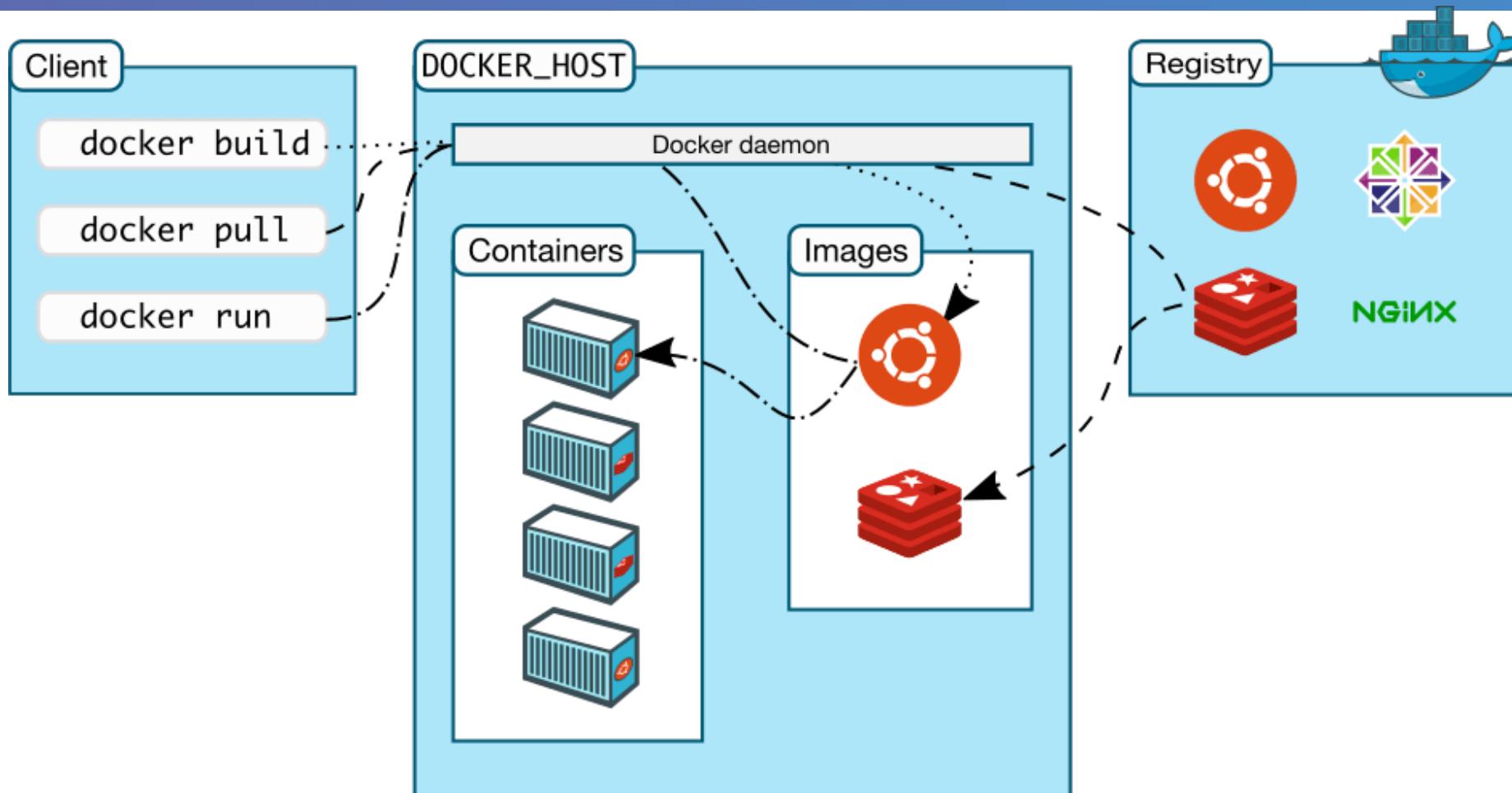
- **Avantages :**
 - Flexibilité quelque soit la complexité de l'application
 - Légereté grâce au partage du noyau du hôte
 - Scalabilité ou Extensibilité
 - ...

DOCKER – définition

- Docker est une technologie qui permet la conteneurisation sur Linux
- Docker démocratise l'utilisation des conteneur
- Docker est composé de :
- Docker engine
- Docker-containerd
- Docker-runc



DOCKER – Fonctionnement



DOCKER – Image

- Sur Docker, un conteneur est lancé à partir d'une image
- Une image est un package qui inclut les fonctionnalités nécessaires à l'exécution de notre processus
- Le contenu de l'image est :
 - Le code
 - Les variables d'environnement
 - Les fichiers de configuration
 - Les bibliothèques
- On peut créer une image de plusieurs façons
 - A partir d'un conteneur
 - A partir d'un fichier DockerFile
- Docker fournit un CLI pour gérer les images sur notre machine

DOCKER – Commande pour Image

- Pour lister les images sur la machines : docker images
- Pour rechercher une image sur le hub docker : docker search
- Pour télécharger une image à partir du hub : docker pull
- Pour exécuter une image : docker run
 - Chaque image a un nom unique
 - Chaque image peut avoir une description
 - Chaque image a plusieurs tags
- Pour supprimer une image : docker rmi nom_image

DOCKER – Conteneur

- Un conteneur est la représentation en mémoire lors de l'exécution d'une image
- Pour créer un conteneur, on utilise la commande docker run [OPTIONS] <nom_image>
- Lors de la création d'un conteneur, on peut spécifier une ou plusieurs options
- Quelques options :
- -t : Allouer un pseudo TTY
- -i : Garder un STDIN ouvert
- -d: Exécuter le conteneur en arrière-plan
- -p :exposer un ou plusieurs ports
- --name : donner un nom au container

DOCKER – Conteneur

- Pour afficher la liste des conteneurs docker container ls ou docker ps
- On peut utiliser l'option -a pour afficher la totalité des conteneurs quelque soit le statut
- Chaque conteneur a :
 - Container ID : id du conteneur
 - IMAGE : l'image utilisée pour créer le conteneur
 - COMMAND : dernière commande lancée lors de l'exécution de l'image
 - CREATED : date de création du conteneur
 - STATUS : statut du conteneur
 - PORTS : les ports utilisés par le conteneur
 - NAME : nom du conteneur

DOCKER – Conteneur

- Un conteneur passe par plusieurs états
 - created : conteneur créé mais non démarré (cet état est possible avec la commande docker create)
 - restarting : conteneur en cours de redémarrage
 - running : conteneur en cours d'exécution
 - paused : conteneur stoppé manuellement (cet état est possible avec la commande docker pause)
 - exited : conteneur qui a été exécuté puis terminé
 - dead : conteneur que le service docker n'a pas réussi à arrêter correctement (généralement en raison d'un périphérique occupé ou d'une ressource utilisée par le conteneur)

DOCKER – Conteneur

- Pour supprimer un conteneur :
- Docker rm <container_id> ou <container_name>
- Pour exécuter une commande dans un conteneur
- Docker exec <OPTIONS> <container_id ou container_name> command
- Pour convertir un conteneur en Image
- Docker commit <container_id ou container_name> <image_name>

DOCKER – Volumes

- Les données dans un conteneur sont éphémères.
- Pour sauvegarder les données d'un conteneur, on peut le convertir à chaque fois en image (mauvaise pratique)
- La deuxième solution, consiste à utiliser les volumes

DOCKER – Volumes (Système de fichier docker)

- Docker utilise un système de fichier en calques
- Chaque image se compose de pile de calques en lecture seule
- A la création d'un conteneur un calque est ajouté dans le haut de pile en lecture-écriture
- Lors d'une modification de fichier, Docker crée une copie depuis les couches en lecture seule vers le layer en lecture-écriture.
- Lors d'une création de fichier, Docker crée le fichier que sur le layer en lecture-écriture, et ne touche pas au layer en lecture seule.
- Lors d'une suppression de fichier, Docker supprime le fichier que sur le layer en lecture-écriture, et s'il est déjà existant dans le layer en lecture seule alors il le garde.
- Les données dans le layer de base sont en lecture seule, elles sont donc protégées et intactes par toutes modifications, seul le layer en lecture-écriture est impacté lors de modifications de données.
- Lorsqu'un conteneur est supprimé, le layer en lecture-écriture est supprimé avec. Cela signifie que toutes les modifications apportées après le lancement du conteneur auront disparus avec.

DOCKER – Volumes (Volume non lié)

- Docker permet la création de volume en dehors de tout conteneur
- Pour créer un volume, on utilise la commande
- Docker volume create <nom_volume>
- Pour lister des volumes
- Docker volume ls
- Pour supprimer un volume
- Docker volume rm <nom_volume>
- Pour créer un conteneur avec volume, on utilise l'option v
- Docker run -v <nom_volume>:<dossier_conteneur>

DOCKER – Volumes (Volume lié)

- On peut également monter un volume local sur un conteneur
- Docker run -v <volume_local>:<volume_conteneur>

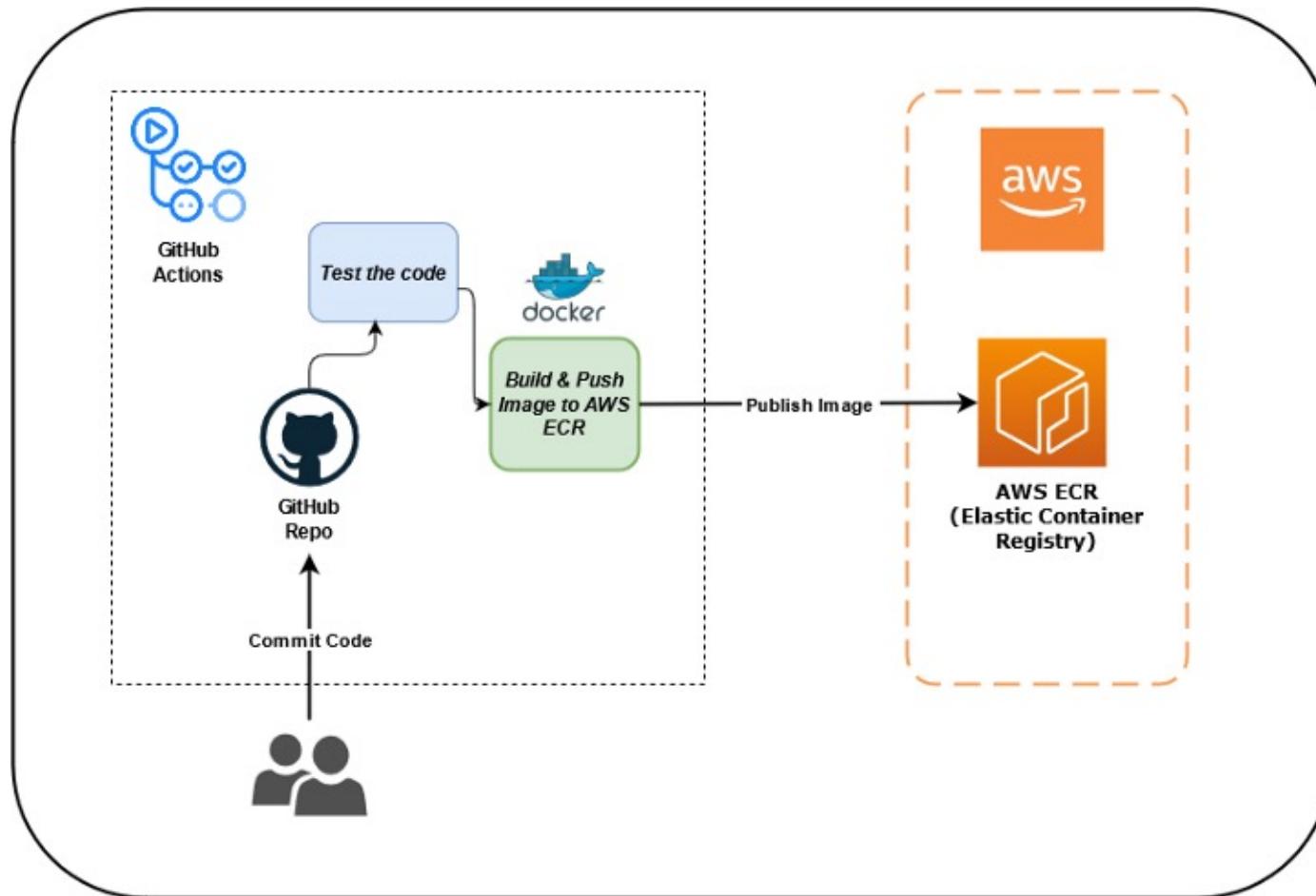
AWS Container registry

- Une fois les dockerfile créés, il faut mettre à disposition les images.
- On peut partager les images dans dockerhub.
- On peut partager les images sur un serveur docker privé.
- AWS propose un service qui permet de créer un serveur d'image privé (AWS container registry).

AWS Container registry

- Les registres privés Amazon ECR hébergent les images de conteneur dans une architecture hautement disponible et évolutive.
- Amazon ECR est un registre privé pour gérer des référentiels d'images privés composés d'images et d'artefacts Docker et Open Container Initiative (OCI).
- Chaque compte AWS est fourni avec un registre Amazon ECR privé par défaut.
- AWS ECR propose un mécanisme de chiffrage des images.
- AWS ECR s'intègre avec AWS IAM

AWS Container registry



AWS Container Registry - Exercice

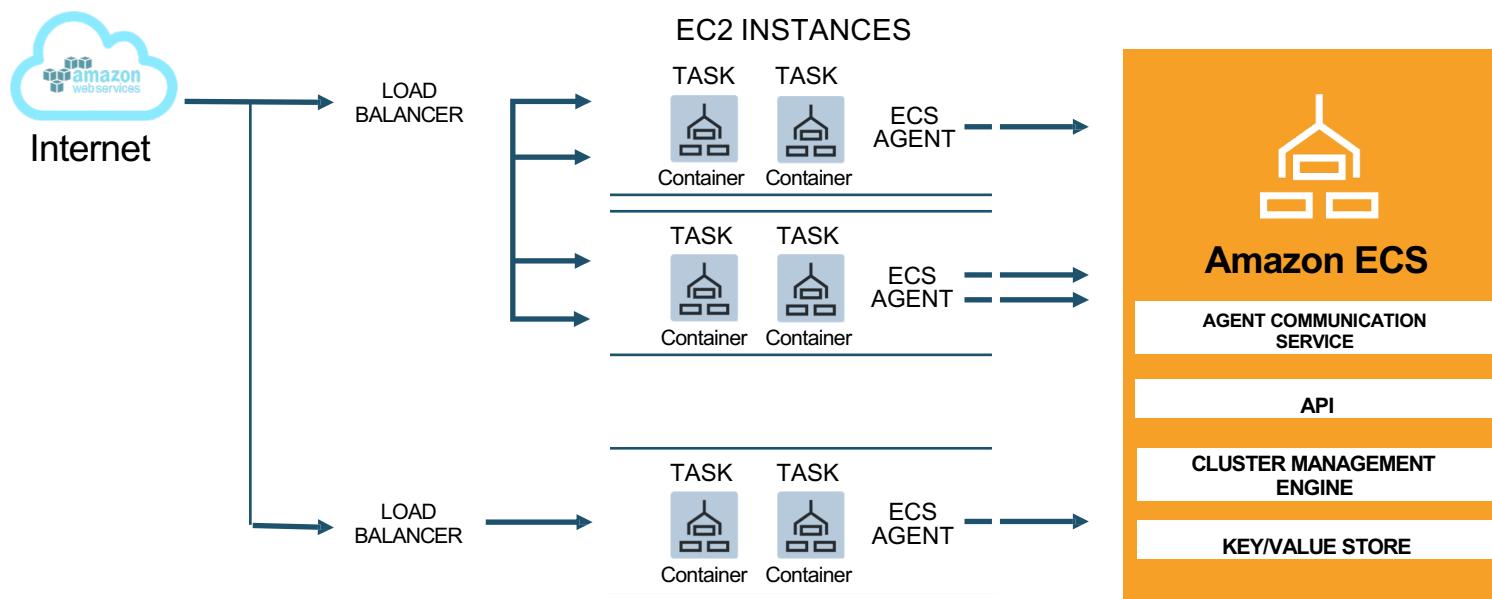
- Partager les images de l'application vote sur un un container registry aws.

AWS Container service

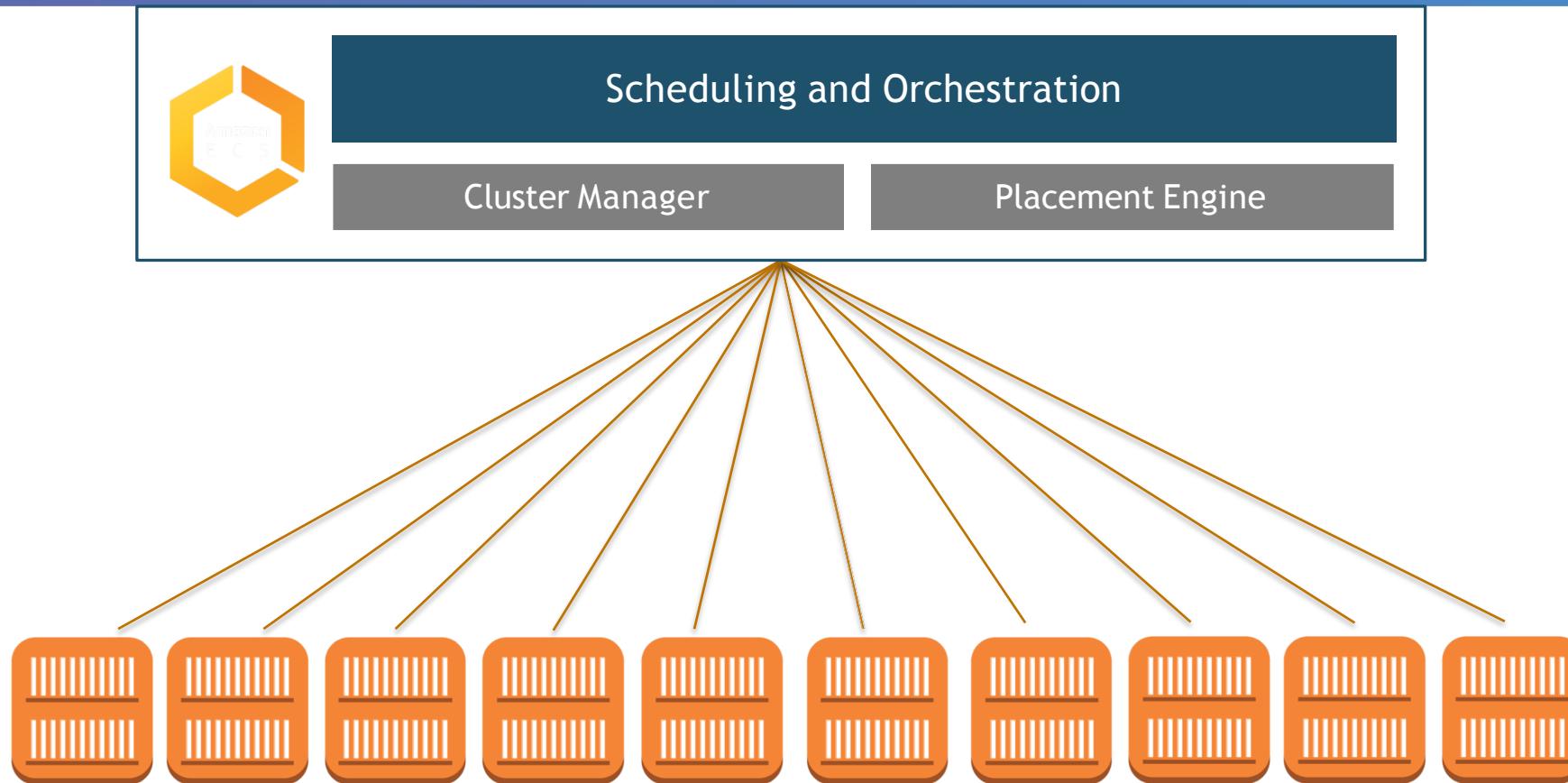
- Aws propose un service managé pour démarrer les conteneurs.



AWS Container Service



Aws container service



AWS Container service

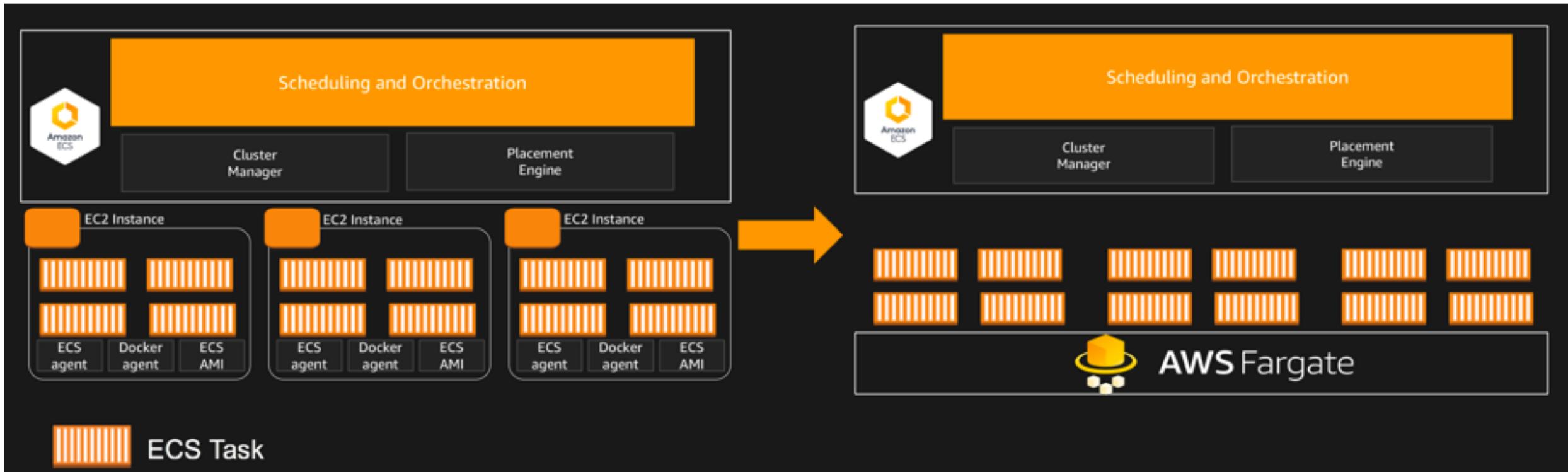
- Le déploiement des conteneur en tant que service sur des instances EC2 nécessite :
 - Gestion de l'os
 - Gestion de Docker agent
 - Gestion de ECS Agent

AWS ECS - FARGATE

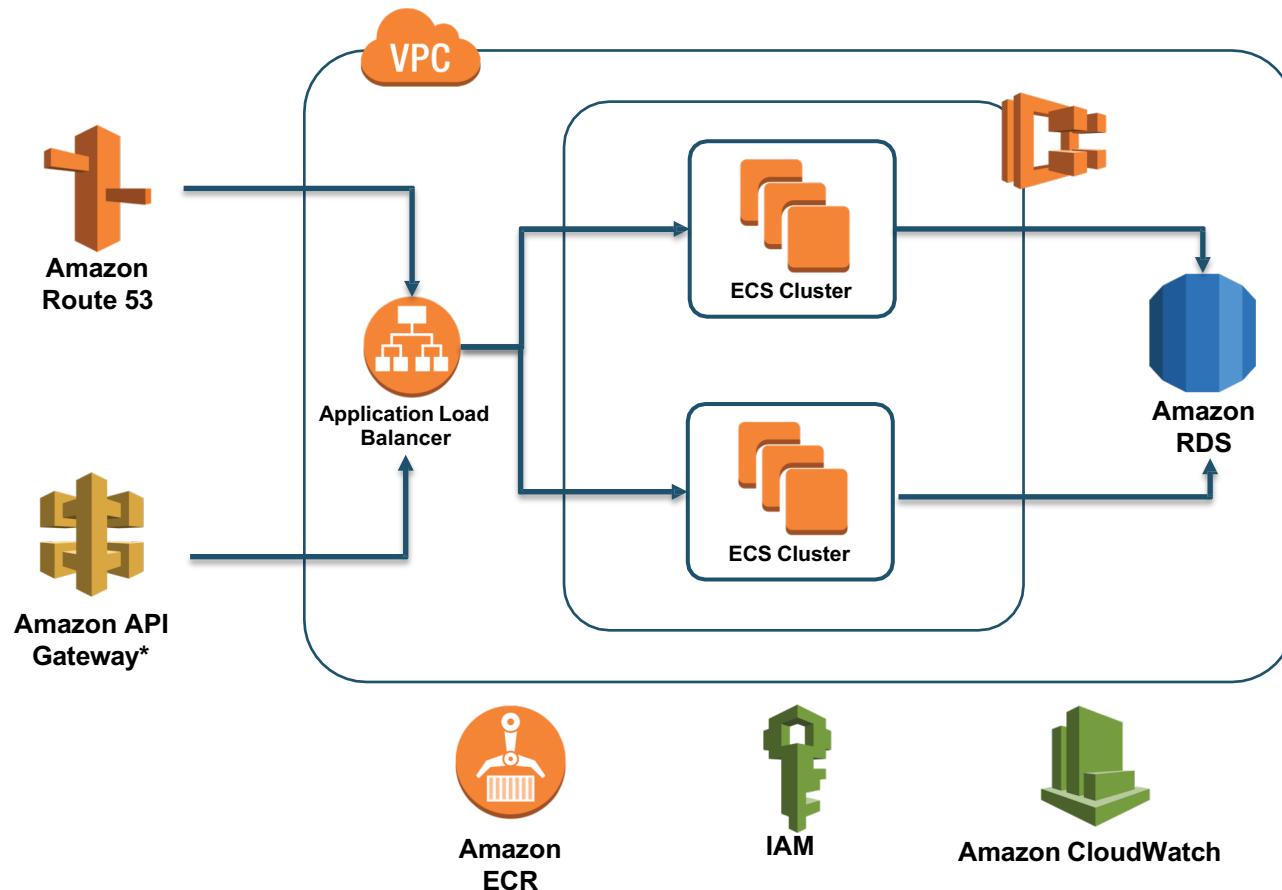
- Un service managé par AWS
 - Aucune instance EC2 à gérer.
 - Pas de besoin de gérer la scalabilité.
- Un service élastique.
- Un service totalement intégré dans l'écosystème aws
 - VPC networking
 - Elastic load Balancing



ECS ec2 -> ECS FARGATE



AWS ECS - Microservice



ECS TP

- Déployer l'application vote dans un cluster ECS.

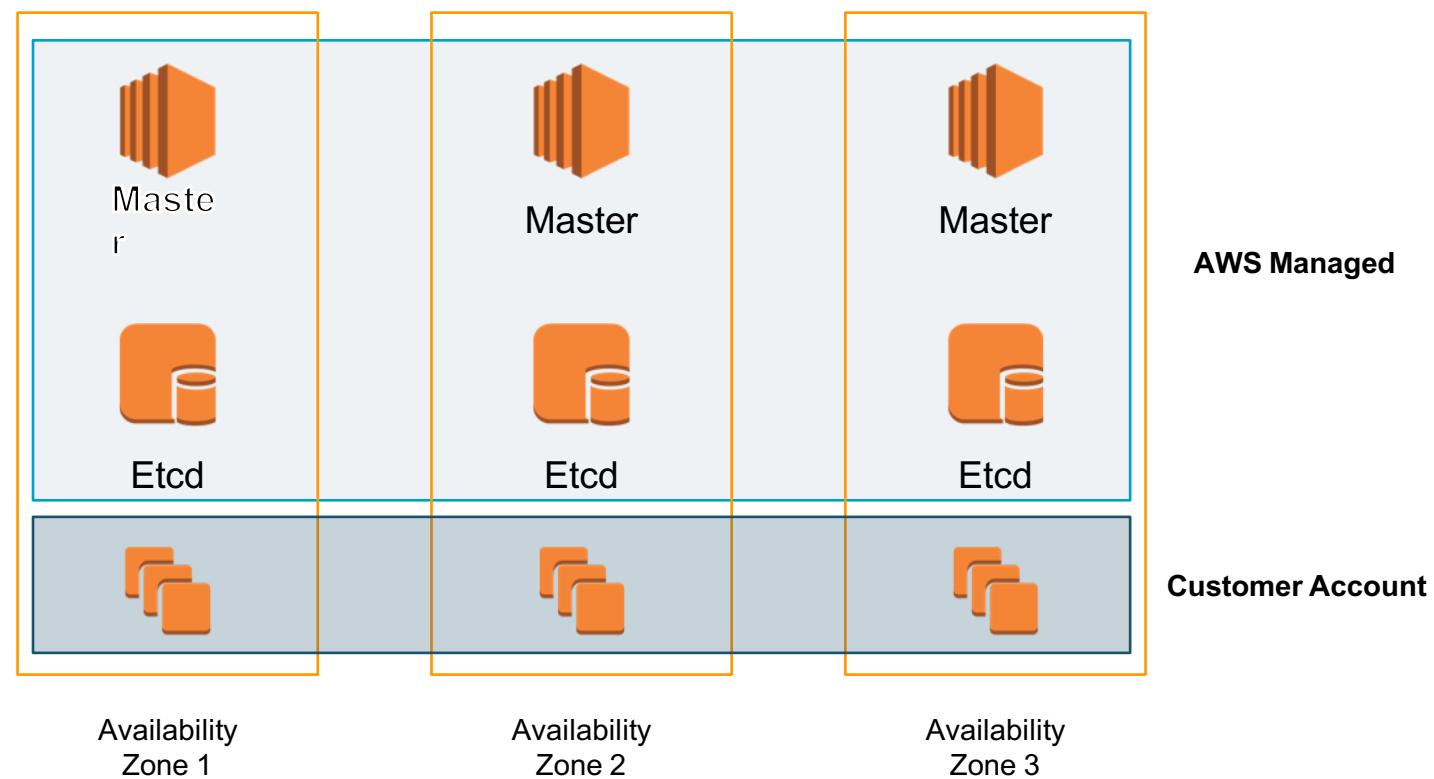
AWS EKS

- EKS est la version managée de Kubernetes pour l'orchestration des conteneurs.

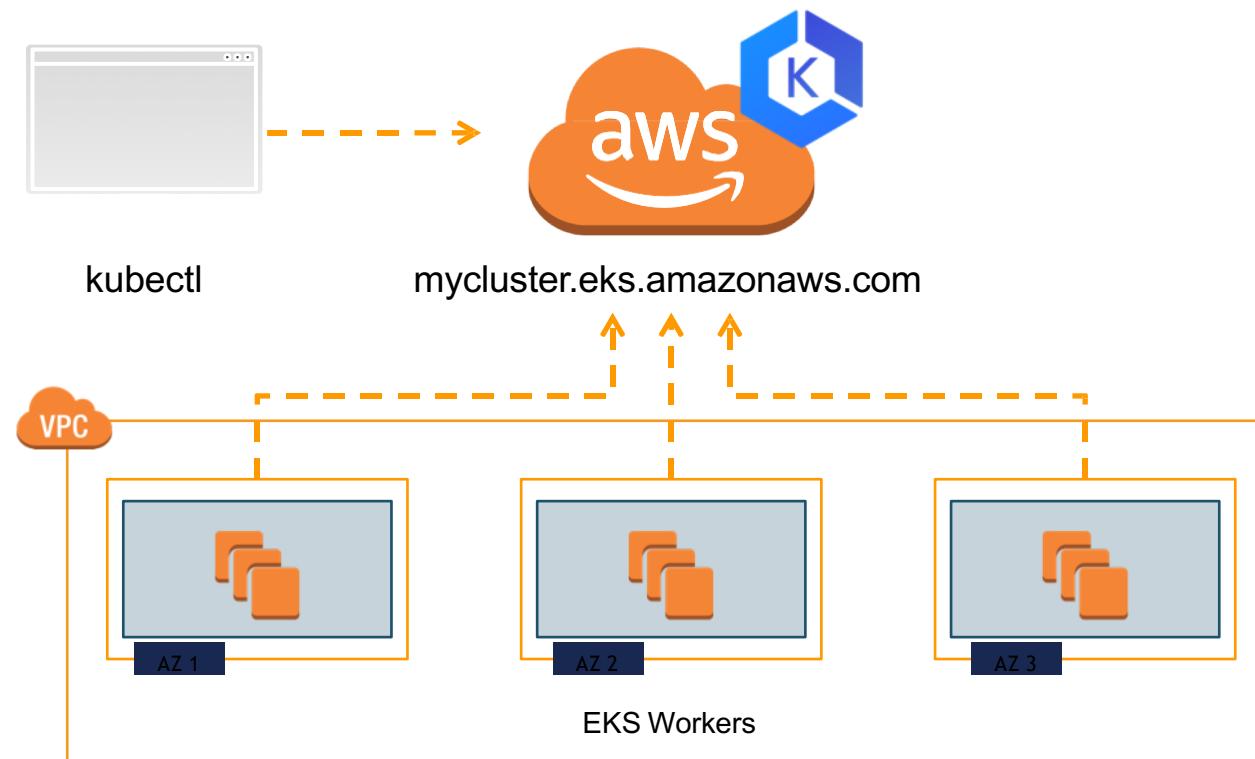


Amazon EKS

AWS EKS architecture



AWS EKS architecture



AWS EKS - IAM

EKS permet d'accorder des permissions aux pods pour accéder aux autres services de AWS à l'aide des rôles iam.

Pour ça il faut avoir :

Un DaemonSet avec un service kube2iam.

En utilisant des annotations dans les déploiements

AWS EKS - IAM

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      annotations:
        iam.amazonaws.com/role:
          arn:aws:iam:123567989012:role/nginx-role
    labels:
      app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.9.1
          ports:
            - containerPort: 80
```

AWS EKS – Ressources AWS

- EKS permet de déployer les ressources aws avec les templates cloudFormation à l'aide Kubernetes.

```
apiVersion:cloudformation.linki.space/v1alpha1
kind: Stack
metadata:
  name: my-bucket

spec:
  template: |
    ---
    AWSTemplateFormatVersion: '2010-09-09'
    Resources:
      S3Bucket:
        Type::AWS::S3::Bucket
        Properties:
          BucketName: my-bucket
```

AWS – Application serverless



Aucun serveur à manager



Sclabilité automatique en fonction de l'usage

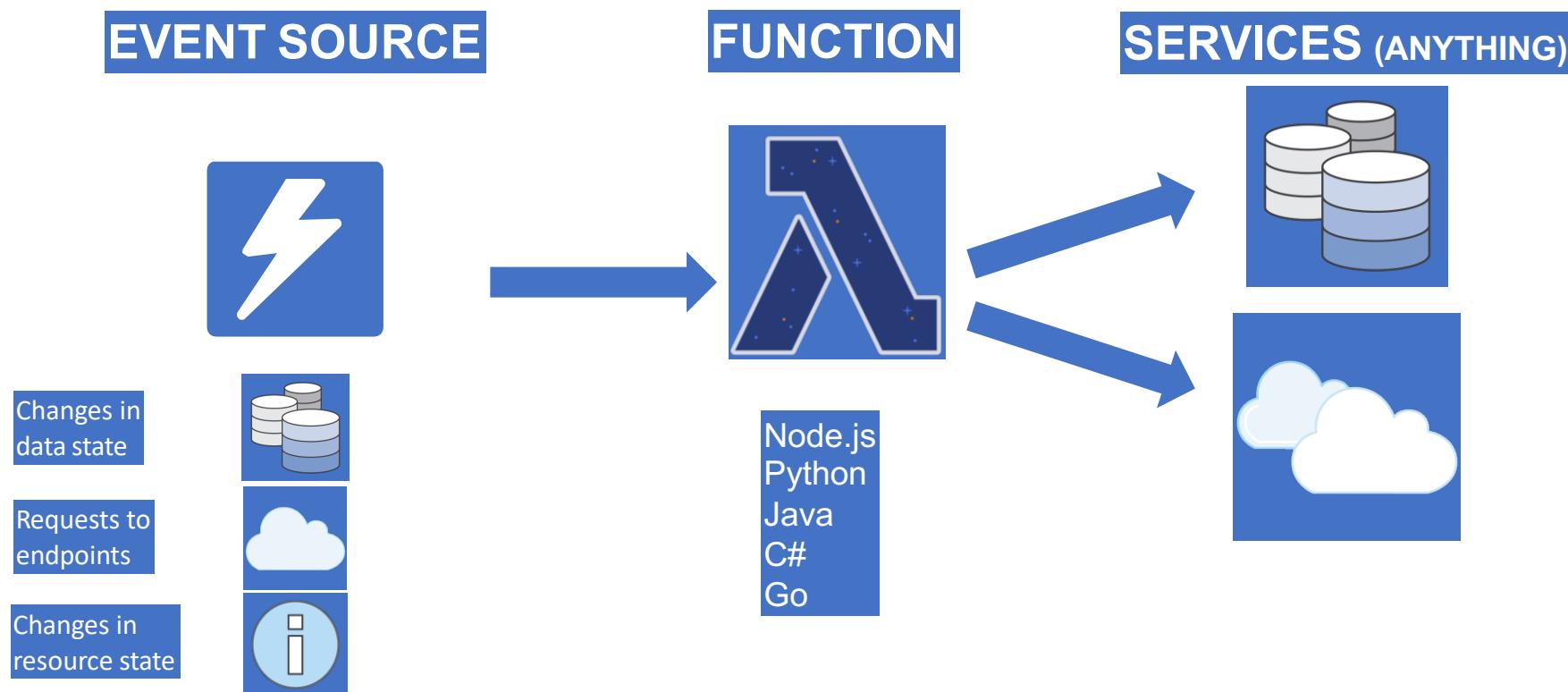


Paiement à l'utilisation



Haute disponibilité

AWS – Fonction Lambda



AWS – Fonction Lambda

Handler() function

Function to be executed upon invocation

Event object

Data sent during Lambda Function Invocation

Context object

Methods available to interact with runtime information (request ID, log group, etc.)

```
public String handleRequest(Book book, Context context) {  
    saveBook(book);  
  
    return book.getName() + " saved!";  
}
```

AWS – Fonction Lambda



Utilisation direct du code

- Node.js, Java, Python, C#, Go
- Bring your own libraries
(even native ones)



Simple resource model

- Select power rating from 128 MB to 3 GB
- CPU and network allocated proportionately



Flexible

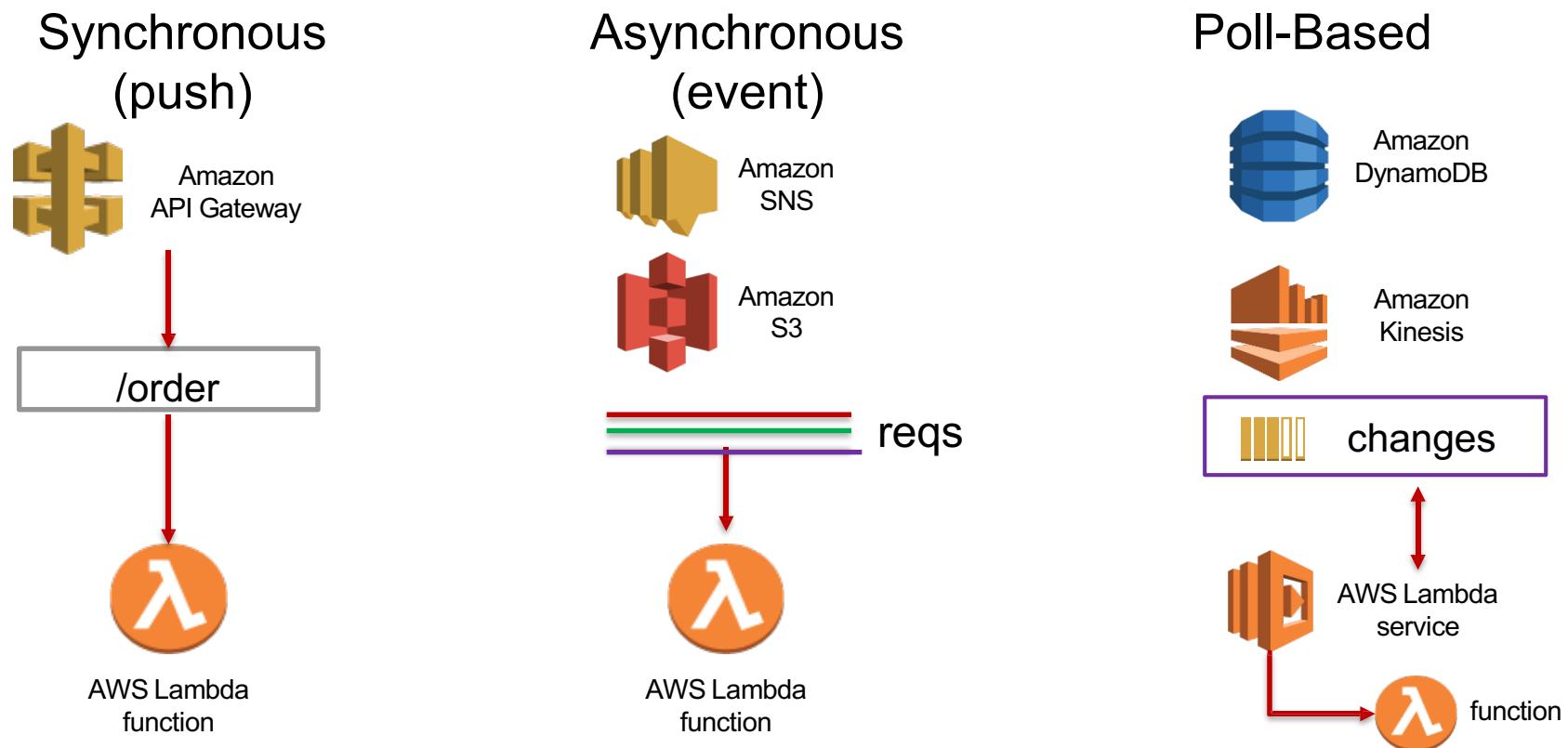
- Synchronous or asynchronous
- Integrated with other AWS services



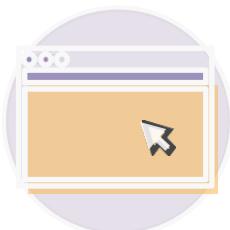
Flexible authorization

- Securely grant access to resources and VPCs
- Fine-grained control for invoking your functions

AWS – Fonction Lambda

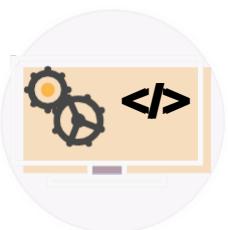


AWS Fonction Lambda – Cas d'utilisation



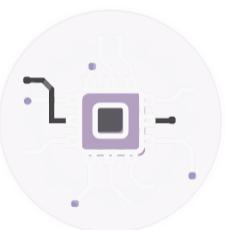
Web Applications

- Static websites
- Complex web apps
- Packages for Flask and Express



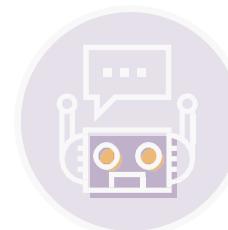
Backends

- Apps & services
- Mobile
- IoT



Data Processing

- Real time
- MapReduce
- Batch



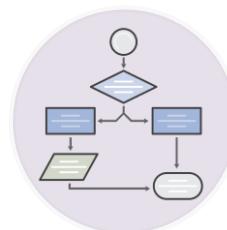
Chatbots

- Powering chatbot logic



Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit



IT Automation

- Policy engines
- Extending AWS services
- Infrastructure management

AWS – Serverless Application Model (SAM)

- Serverless est une extension de AWS CloudFormation
- Nouvelles ressources : functions, APIs, and tables
- SAM est open source

AWS – SAM Template

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetHtmlFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri:s3://sam-demo-bucket/todo_list.zip
      Handler:index.gethtml
      Runtime:nodejs6.10
    Policies: AmazonDynamoDBReadOnlyAccess
    Events:
      GetHtml:
        Type: Api
        Properties:
          Path: /{proxy+}
          Method: ANY
  ListTable:
    Type: AWS::Serverless::SimpleTable
```

AWS SAM Event source

S3

SNS

Kinesis | DynamoDB

Api

Schedule

CloudWatchEvent

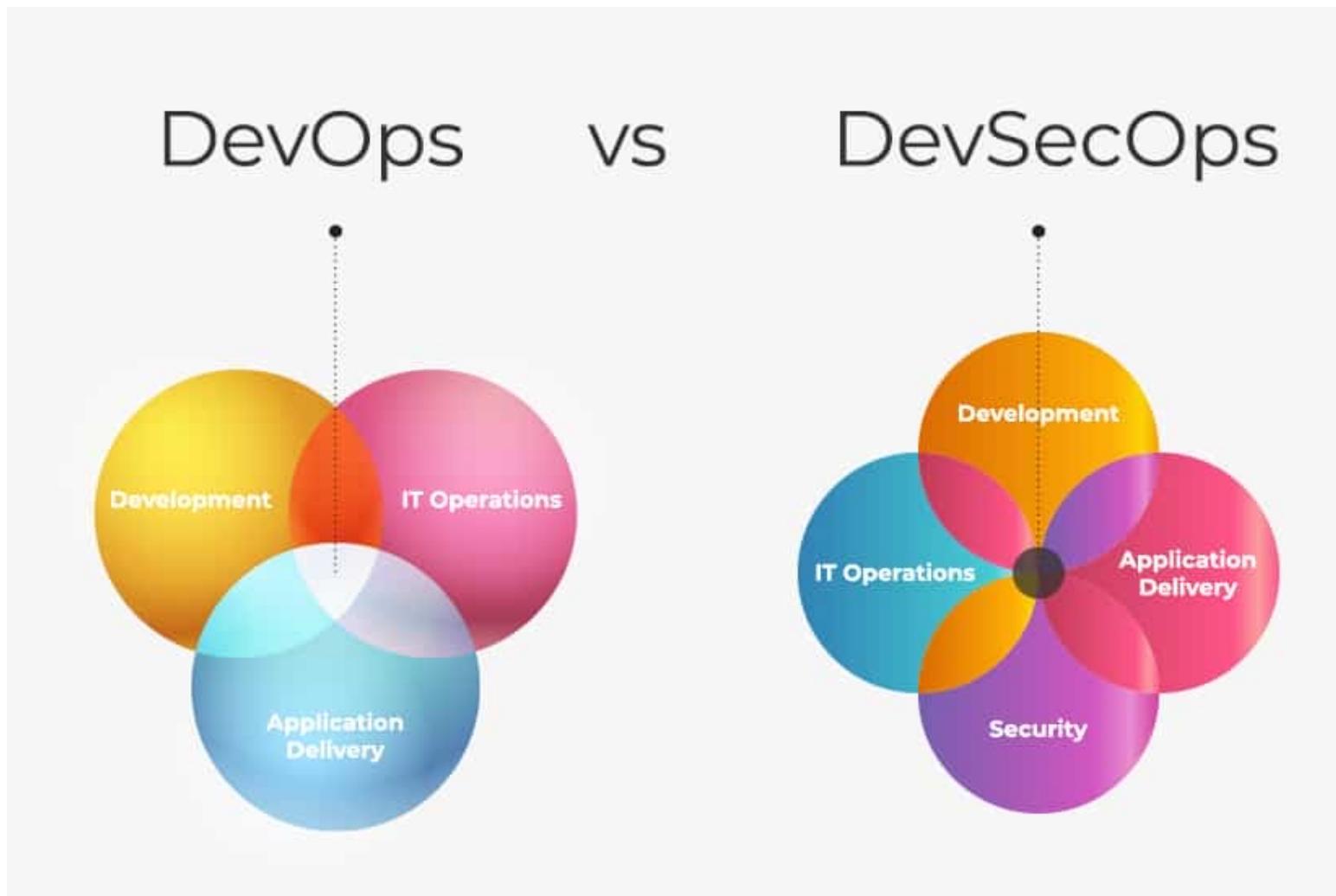
IoTRule

AlexaSkill

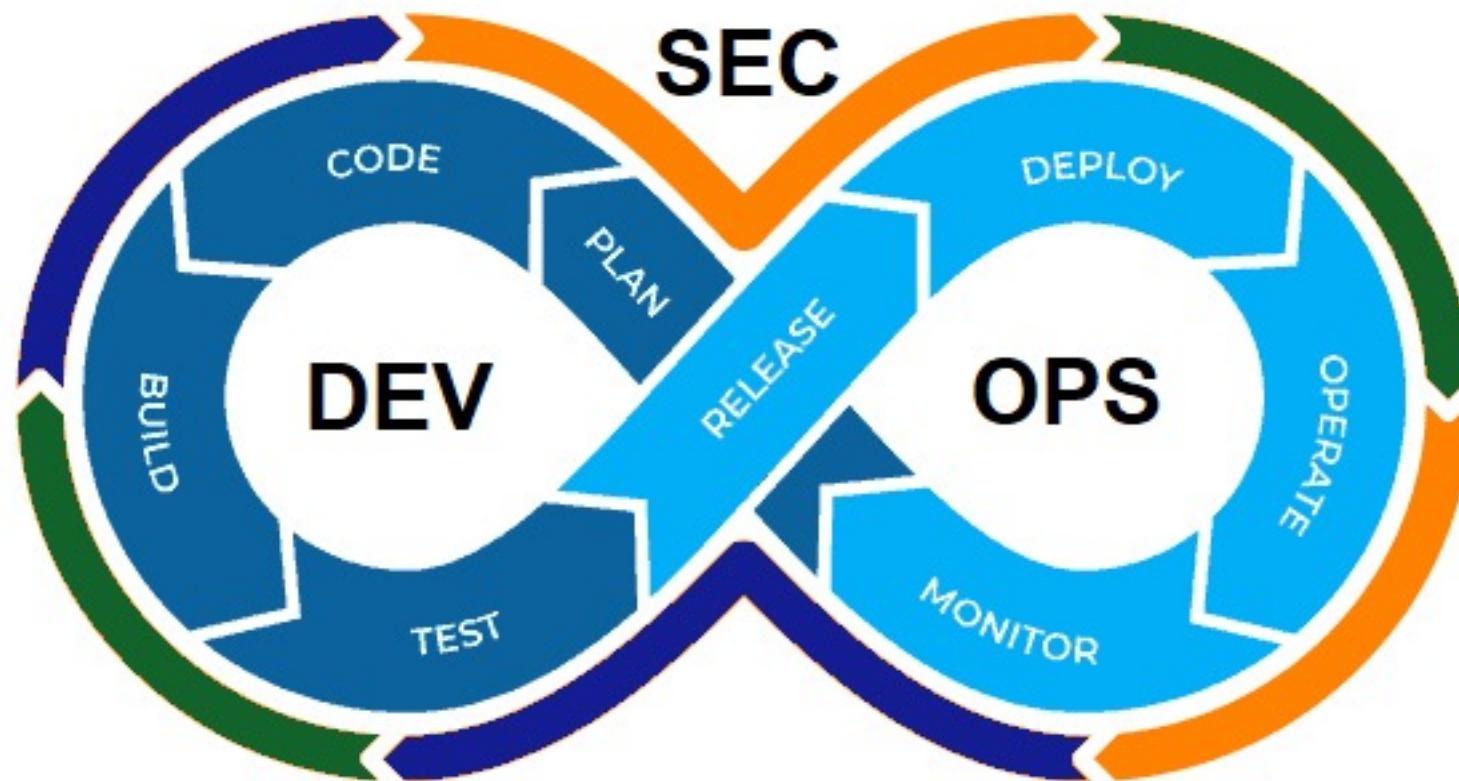
AWS - TP

- Déployer notre application de vote à l'aide de SAM

Introduction DevSecOps



Introduction DevSecOps



PipeLine DevSecOps

