FORMATION DOCKER

Docker sur AWS

Parties 6 à 9 : Amazon ECR, Amazon ECS,
écosystème AWS et concepts avancés

06

Du registre local à Amazon ECR

Registre privé, dépôts ECR, authentification, versioning, scan des images et CI/CD

Rappel : héberger son propre registre

```
# Lancer un registre local
docker run -d -p 5000:5000 --name registry registry:2

# Taguer puis pousser une image
docker tag mon-app:1.0 localhost:5000/mon-app:1.0
docker push localhost:5000/mon-app:1.0

# Récupérer l'image depuis un autre hôte
docker pull localhost:5000/mon-app:1.0
```

Ce qui reste à votre charge

- Stockage et sauvegarde des images
- Certificat TLS et authentification
- Haute disponibilité du registre
- Nettoyage des anciennes images

Amazon ECR : principes

Registre managé

Un registre par compte et par région, hébergé par AWS. Compatible Docker et OCI.

Dépôts privés

Accès contrôlé par IAM et par une politique attachée au dépôt.

Dépôts publics

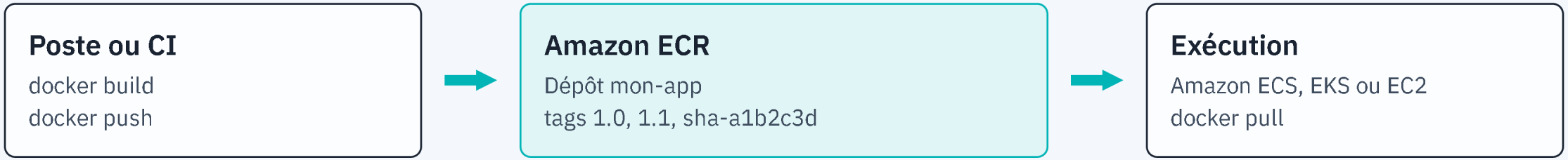
ECR Public, sous public.ecr.aws. Lecture anonyme, publication authentifiée.

Adresse d'un dépôt privé

```
123456789012.dkr.ecr.eu-west-3.amazonaws.com/mon-app
```

Images chiffrées au repos (AES-256 ou clé KMS). Facturation au volume stocké et au transfert sortant.

Circulation d'une image avec ECR



Les droits IAM s'appliquent dans les deux sens : la CI reçoit le droit de pousser, l'environnement d'exécution reçoit le droit de tirer.

Authentification, push et pull

```
# 1. Créer le dépôt
aws ecr create-repository --repository-name mon-app --region eu-west-3

# 2. Authentifier Docker (jeton valable 12 heures)
aws ecr get-login-password --region eu-west-3 \
  | docker login --username AWS --password-stdin 123456789012.dkr.ecr.eu-west-3.amazonaws.com

# 3. Taguer, pousser, récupérer
docker tag mon-app:1.0 123456789012.dkr.ecr.eu-west-3.amazonaws.com/mon-app:1.0
docker push 123456789012.dkr.ecr.eu-west-3.amazonaws.com/mon-app:1.0
docker pull 123456789012.dkr.ecr.eu-west-3.amazonaws.com/mon-app:1.0
```

Tags et versioning

Conventions de tag

- Version sémantique : **1.4.2**
- Traçabilité : SHA du commit, **sha-a1b2c3d**
- Pas de **latest** en production
- Référence fixe par digest : **mon-app@sha256:...**

Réglages du dépôt

- Tags immuables : un tag poussé ne peut plus être écrasé
- Règles de cycle de vie : suppression automatique des images non taguées ou trop anciennes

```
aws ecr put-image-tag-mutability --repository-name mon-app --image-tag-mutability IMMUTABLE
```

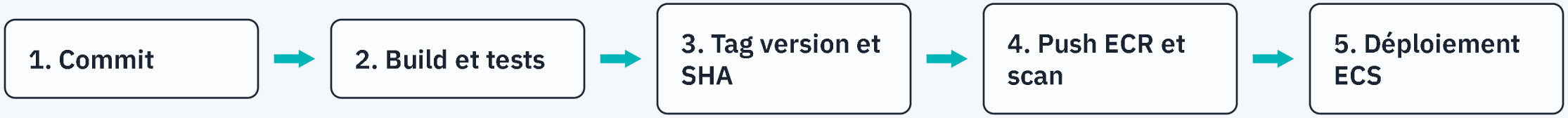
Scan de vulnérabilités

Mode	Moteur	Déclenchement	Couverture
Basique	AWS (natif)	Au push ou à la demande	Paquets du système
Amélioré	Amazon Inspector	Continu, à chaque nouvelle CVE	Système et dépendances (npm, pip, Maven...)

```
# Activer le scan au push sur tous les dépôts du registre
aws ecr put-registry-scanning-configuration --scan-type BASIC \
  --rules '[{"scanFrequency":"SCAN_ON_PUSH","repositoryFilters":[{"filter":"*","filterType":"WILDCARD"}]}]'

# Consulter les résultats
aws ecr describe-image-scan-findings --repository-name mon-app --image-id imageTag=1.4.2
```

ECR dans un pipeline CI/CD



Authentification de la CI

Rôle IAM assumé via OIDC (GitHub Actions, GitLab CI). Aucune clé d'accès stockée dans le pipeline.

Contrôle avant déploiement

Le pipeline s'arrête si le scan remonte des vulnérabilités critiques. Le déploiement référence le tag SHA.

07

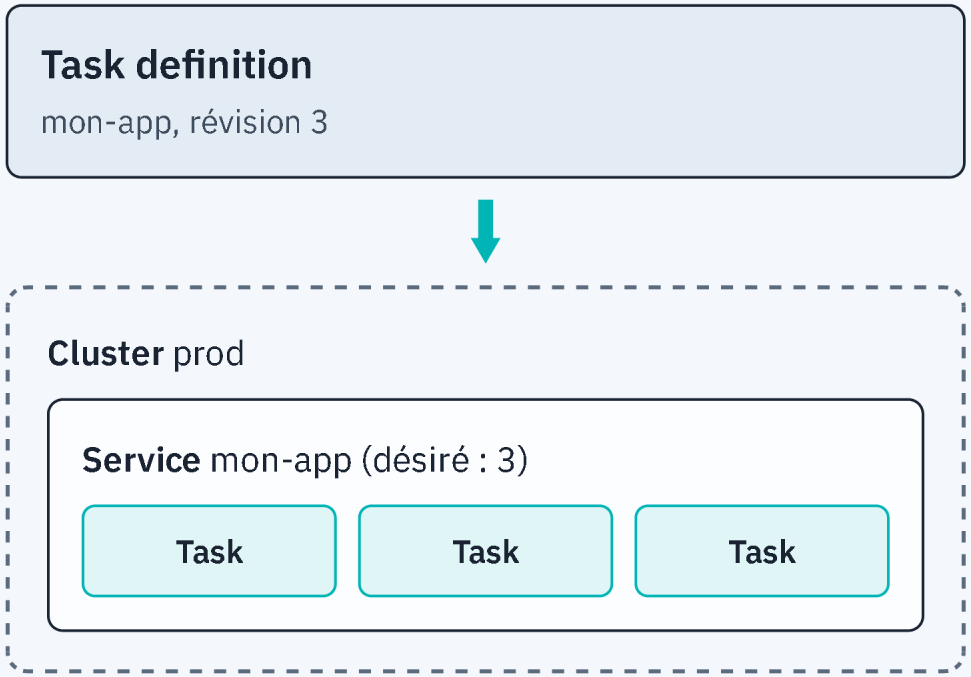
Déployer des conteneurs avec Amazon ECS

Clusters, task definitions, services, EC2 et Fargate, logs et configuration

Du docker run au service managé

	docker run	Docker Compose	Amazon ECS
Périmètre	Un conteneur	Plusieurs conteneurs, un hôte	Plusieurs services, plusieurs hôtes
Redémarrage	--restart	restart:	Le service maintient le nombre de tâches
Montée en charge	Manuelle	replicas sur un hôte	Auto Scaling des tâches
Exposition	-p	ports:	ALB, VPC, groupes de sécurité
Description	La commande	docker-compose.yml	Task definition et service

Les concepts ECS



Cluster : regroupement logique de la capacité de calcul, EC2 ou Fargate.

Task definition : modèle versionné qui décrit les conteneurs, l'image, le CPU, la mémoire et le réseau.

Task : instance en cours d'exécution d'une task definition.

Service : maintient N tâches actives, remplace celles qui échouent et les inscrit auprès d'un load balancer.

Exemple de task definition

```
{
  "family": "mon-app",
  "requiresCompatibilities": ["FARGATE"],
  "networkMode": "awsvpc",
  "cpu": "512",
  "memory": "1024",
  "executionRoleArn": "arn:aws:iam::<compte>:role/ecsTaskExecutionRole",
  "containerDefinitions": [{
    "name": "web",
    "image": "<compte>.dkr.ecr.eu-west-3.amazonaws.com/mon-app:1.4.2",
    "portMappings": [{ "containerPort": 8080 }],
    "essential": true
  }]
}
```

Chaque enregistrement crée une révision : mon-app:1, mon-app:2…

Le service pointe vers une révision précise.

En mode awsvpc, chaque tâche reçoit sa propre interface réseau.

Modèles de lancement : EC2 ou Fargate

EC2

Les tâches tournent sur des instances EC2 rattachées au cluster.

- Choix du type d'instance (GPU, ARM, disque local)
- Correctifs et dimensionnement des instances à votre charge
- Facturation à l'instance, avantageuse à forte densité

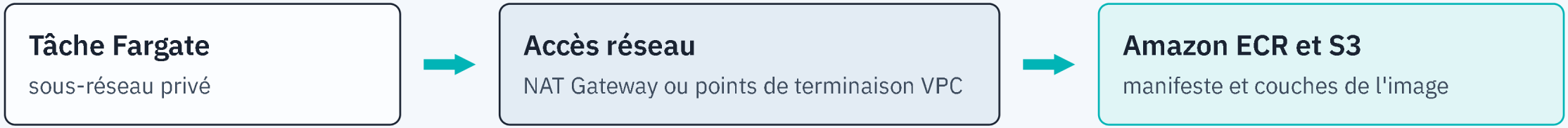
Fargate (serverless)

AWS fournit la capacité de calcul pour chaque tâche.

- Aucun serveur à gérer
- Isolation par tâche, réseau awsvpc obligatoire
- Facturation au vCPU et à la mémoire, à la seconde (1 minute minimum)
- Fargate Spot pour les charges interruptibles

Les capacity providers permettent de combiner les deux modèles dans un même cluster.

Récupérer les images depuis ECR



Deux rôles IAM distincts

Rôle d'exécution : utilisé par ECS pour tirer l'image et écrire les logs (AmazonECSTaskExecutionRolePolicy). La lecture des secrets demande des droits en plus.

Rôle de tâche : utilisé par le code de l'application pour appeler S3, SQS, DynamoDB...

Points de terminaison VPC

ecr.api et ecr.dkr (interface), plus la passerelle S3 car les couches des images sont stockées dans S3. Ajouter logs pour CloudWatch.

Logs CloudWatch et ressources

```
"logConfiguration": {
  "logDriver": "awslogs",
  "options": {
    "awslogs-group": "/ecs/mon-app",
    "awslogs-region": "eu-west-3",
    "awslogs-stream-prefix": "web"
  }
}
```

Ressources allouées

Niveau tâche : cpu et memory, obligatoires sur Fargate avec des combinaisons imposées (0,5 vCPU : 1 à 4 Go).

Niveau conteneur : memory (limite stricte), memoryReservation (réservation souple), cpu.

Unité CPU : 1024 = 1 vCPU.

Variables d'environnement et secrets

```
"environment": [
  { "name": "APP_ENV", "value": "production" }
],
"secrets": [
  { "name": "DB_PASSWORD",
    "valueFrom": "arn:aws:secretsmanager:eu-west-3:<compte>:secret:prod/db-AbCdEf" },
  { "name": "API_KEY",
    "valueFrom": "arn:aws:ssm:eu-west-3:<compte>:parameter/prod/api-key" }
]
```

secrets : valeurs lues dans Secrets Manager ou SSM Parameter Store au démarrage de la tâche, via le rôle d'exécution.

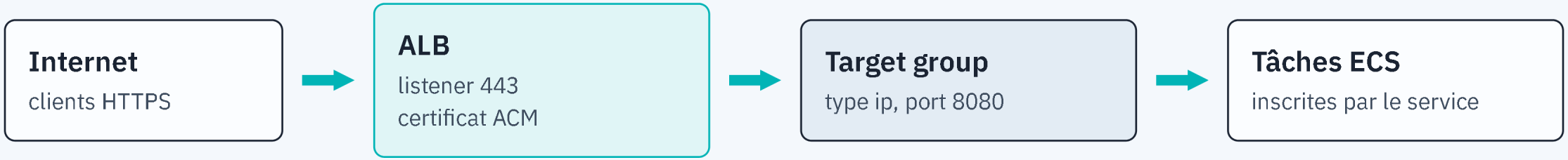
environmentFiles : fichier .env stocké dans S3. Toute nouvelle valeur nécessite un redéploiement des tâches.

08

Intégration de l'écosystème AWS

ALB, RDS, ElastiCache et passage d'un fichier Compose à une architecture ECS

Exposer un service ECS avec un ALB



Règles du listener : routage par chemin (/api/*) ou par nom d'hôte vers des target groups différents.

Côté service : le bloc loadBalancers (targetGroupArn, containerName, containerPort) inscrit et retire les tâches à chaque déploiement.

Sur Fargate : target group de type ip, imposé par le mode réseau awsvpc.

Health checks et déploiements

Target group

Valeurs par défaut

- Chemin : / (à remplacer par /health)
- Intervalle : 30 s
- Seuil sain : 5, seuil non sain : 2
- Code attendu : 200

Service ECS

- healthCheckGracePeriodSeconds : délai de démarrage avant prise en compte des checks
- Rolling update : minimumHealthyPercent et maximumPercent
- Circuit breaker avec retour arrière automatique

Groupe de sécurité des tâches : le port applicatif n'est ouvert qu'au groupe de sécurité de l'ALB.

Persistance : RDS à la place d'un conteneur base

Conteneur postgres (Compose)

- Volume local à sauvegarder soi-même
- Un seul hôte, pas de bascule
- Mises à jour de version manuelles

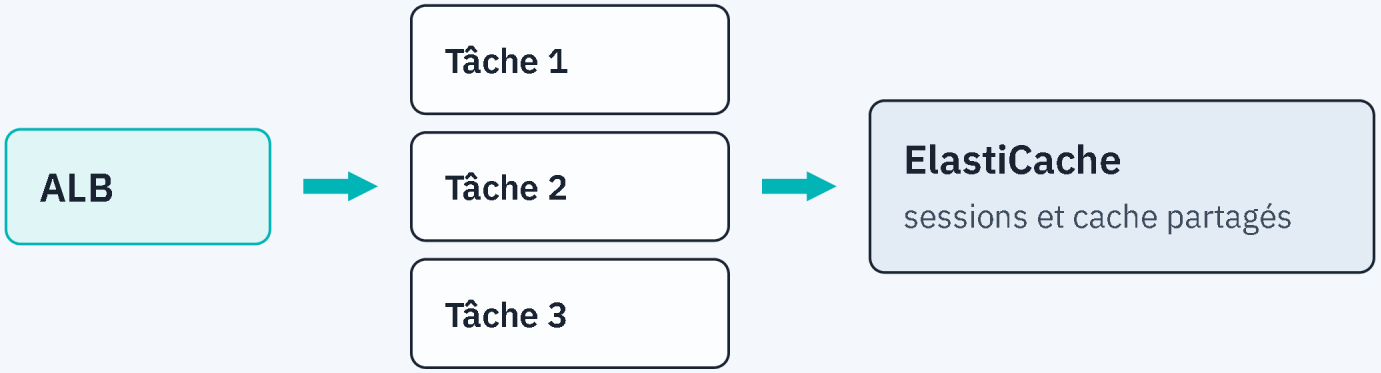
Amazon RDS

- Sauvegardes automatiques, restauration à un instant donné
- Multi-AZ avec bascule automatique
- Correctifs gérés, réplicas en lecture

Connexion depuis ECS

Endpoint RDS en variable d'environnement, mot de passe dans Secrets Manager, groupe de sécurité RDS ouvert sur le port 5432 au seul groupe des tâches.

Cache et sessions avec ElastiCache



Usages

Sessions partagées dès qu'il y a plusieurs tâches derrière l'ALB, cache applicatif.

Options

Redis OSS ou Valkey, réplicas Multi-AZ, chiffrement TLS et au repos, mode serverless.

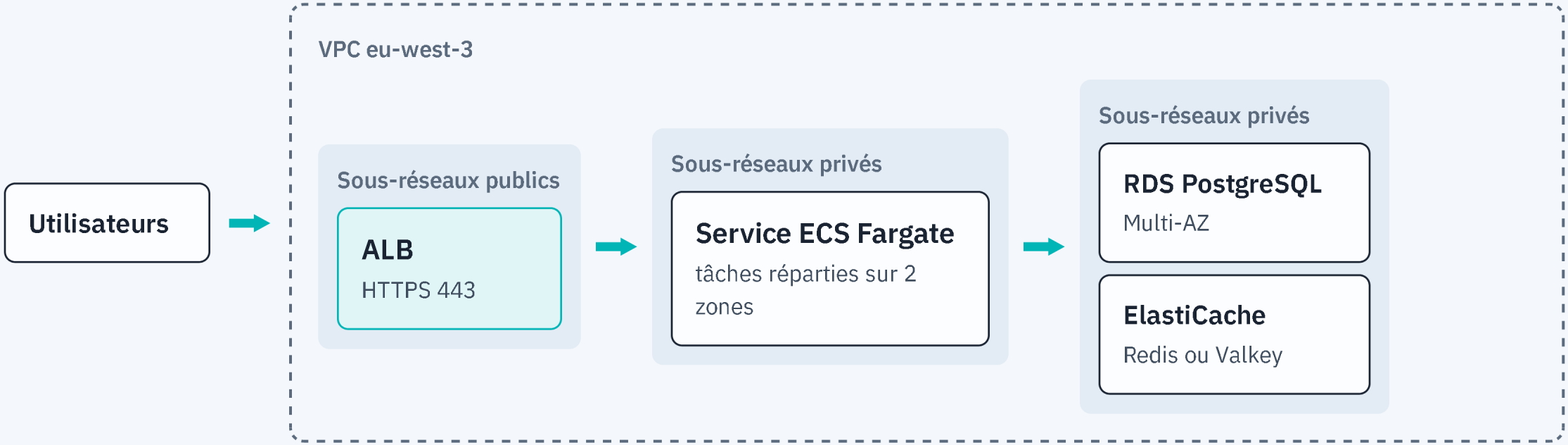
Connexion

Endpoint du cluster en variable d'environnement, port 6379 ouvert aux seules tâches.

Du docker-compose.yml à ECS

Docker Compose	Équivalent AWS
services.web (image, build)	Image sur ECR, task definition et service ECS
ports: "80:8080"	Listener ALB et target group vers le port 8080
environment, env_file	environment, secrets, environmentFiles
services.db (postgres)	Amazon RDS
services.redis	Amazon ElastiCache
volumes	Amazon EFS monté dans la tâche, ou RDS et S3
depends_on, healthcheck	dependsOn et healthCheck du conteneur, checks ALB
deploy.replicas	desiredCount du service et Auto Scaling

Architecture cible



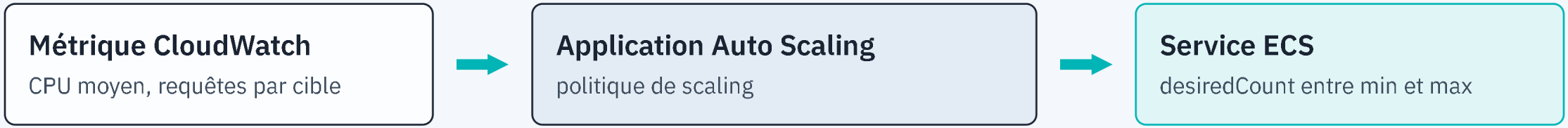
Services utilisés par les tâches : Amazon ECR : images | CloudWatch Logs : journaux | Secrets Manager : identifiants

09

Concepts avancés

Mise à l'échelle, stratégies de déploiement, communication entre services, sécurité, observabilité, coûts et infrastructure as code

Auto Scaling des services ECS



Suivi de cible

Maintient une métrique à une valeur : CPU à 60 %, nombre de requêtes ALB par tâche.

Par paliers

Ajoute ou retire N tâches selon des seuils d'alarme CloudWatch.

Planifié

Ajuste min et max à heure fixe, par exemple à zéro la nuit en recette.

Mettre en place le suivi de cible

```
aws application-autoscaling \
  register-scalable-target \
  --service-namespace ecs \
  --resource-id service/prod/mon-app \
  --scalable-dimension ecs:service:DesiredCount \
  --min-capacity 2 --max-capacity 10

aws application-autoscaling put-scaling-policy \
  --service-namespace ecs \
  --resource-id service/prod/mon-app \
  --scalable-dimension ecs:service:DesiredCount \
  --policy-name cpu-60 \
  --policy-type TargetTrackingScaling \
  --target-tracking-scaling-policy-configuration \
    file://cpu60.json
```

cpu60.json

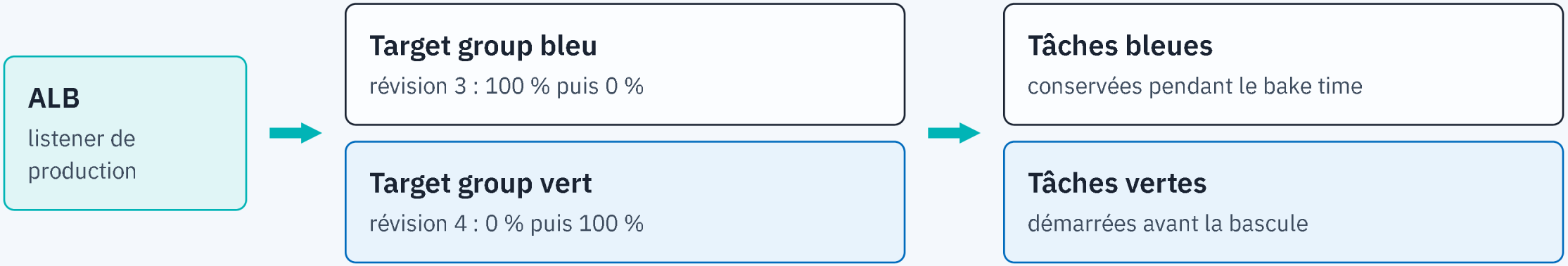
```
{
  "TargetValue": 60.0,
  "PredefinedMetricSpecification": {
    "PredefinedMetricType":
      "ECSServiceAverageCPUUtilization"
  },
  "ScaleInCooldown": 120
}
```

Stratégies de déploiement

Stratégie	Bascule du trafic	Ressources	Usage type
Rolling update	Tâches remplacées par lots	Jusqu'à maximumPercent	Cas général
Blue/green	100 % en une étape après validation	Doublées	Changement majeur, retour arrière rapide
Linéaire	Paliers égaux, ex. 10 % toutes les 5 min	Doublées	Validation progressive
Canary	Petite part (5 à 10 %), puis le reste	Doublées	Test sur un échantillon réel

Blue/green, linéaire et canary sont intégrés à ECS depuis 2025, sans CodeDeploy. Ils s'appuient sur l'ALB ou Service Connect, des hooks de validation et des alarmes CloudWatch pour le retour arrière.

Blue/green avec l'ALB



1. Les tâches vertes démarrent et passent les health checks.
2. Un listener de test permet de valider la version verte avant la bascule.
3. Le trafic bascule, la version bleue reste disponible pendant le bake time.
4. Une alarme CloudWatch déclenche le retour immédiat vers le bleu.

Communication entre services



Service Connect

Nom court, proxy géré par ECS, retries, timeouts et métriques par service.

Service discovery

Enregistrement DNS des tâches dans Cloud Map, sans proxy.

ALB interne

Point d'entrée stable, routage HTTP par chemin ou par hôte.

Modèles de capacité

	EC2 (Auto Scaling group)	ECS Managed Instances	Fargate
Gestion des instances	Vous : AMI, correctifs, scaling	AWS : provisionnement, correctifs, remplacement	Aucune instance visible
Choix du matériel	Tous les types EC2	Types EC2 au choix, GPU et ARM inclus	vCPU et mémoire
Densité	Plusieurs tâches par instance	Plusieurs tâches par instance	Une tâche isolée
Facturation	Instance EC2	Instance EC2 et frais de gestion	vCPU et mémoire, à la seconde

Managed Instances, lancé en 2025, se situe entre EC2 et Fargate. Vérifier sa disponibilité dans la région visée.

Sécurité des tâches

Identité

Un rôle de tâche par service, limité aux actions et ressources nécessaires.

Réseau

Tâches en sous-réseaux privés, points de terminaison VPC, groupes de sécurité chaînés.

Entrée

HTTPS avec un certificat ACM, AWS WAF devant l'ALB.

Conteneur

Utilisateur non root, système de fichiers en lecture seule (prévoir un volume pour /tmp), capacités Linux retirées.

```
"user": "1000",
"readonlyRootFilesystem": true,
"linuxParameters": { "capabilities": { "drop": ["ALL"] } }
```

Déboguer avec ECS Exec

```
# Activer sur le service (nouvelles tâches nécessaires)
aws ecs update-service --cluster prod --service mon-app \
  --enable-execute-command --force-new-deployment

# Ouvrir un shell dans un conteneur
aws ecs execute-command --cluster prod --task <task-id> \
  --container web --interactive --command "/bin/sh"
```

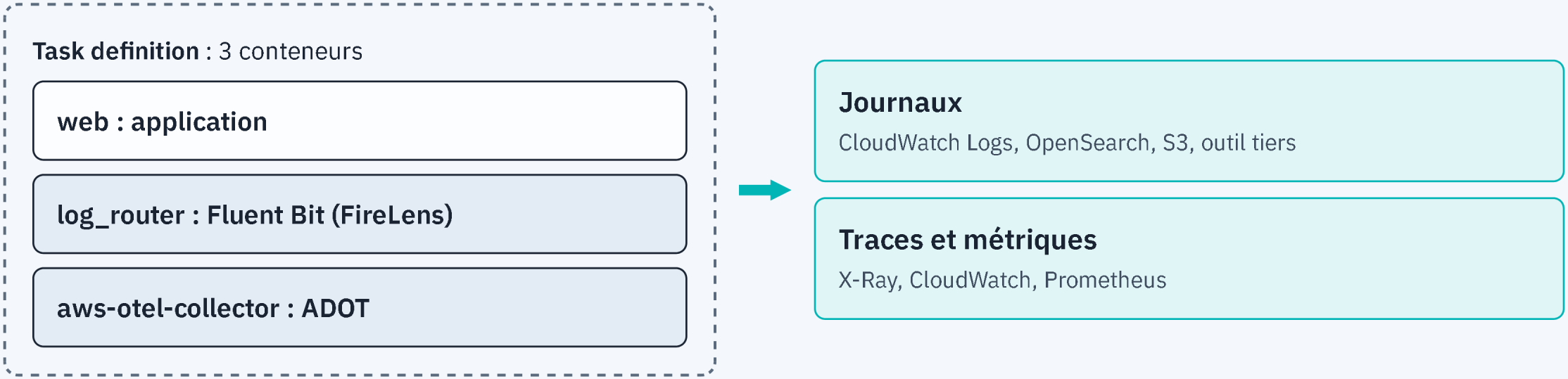
Prérequis

Plugin Session Manager sur le poste. Rôle de tâche avec ssmmessages:CreateControlChannel, CreateDataChannel, OpenControlChannel et OpenDataChannel.

Traçabilité

Sessions journalisables dans CloudWatch Logs ou S3. Accès contrôlé par IAM sur ecs:ExecuteCommand.

Observabilité



Container Insights : métriques CPU, mémoire et réseau par cluster, service et tâche.

```
aws ecs update-cluster-settings --cluster prod --settings name=containerInsights,value=enhanced
```

Maîtrise des coûts

Fargate Spot

Jusqu'à 70 % moins cher. Interruption possible avec un préavis de 2 minutes (SIGTERM).

Graviton (ARM64)

Environ 20 % moins cher par vCPU sur Fargate. Nécessite une image multi-architecture.

Savings Plans

Les Compute Savings Plans couvrent Fargate et EC2, sur 1 ou 3 ans.

Dimensionnement

Ajuster CPU et mémoire avec Container Insights. Arrêter les environnements hors production la nuit.

```
# Image multi-architecture
docker buildx build --platform linux/amd64,linux/arm64 -t $REGISTRY/mon-app:1.2.0 --push .
# 2 tâches garanties en Fargate, le reste à 75 % en Spot
--capacity-provider-strategy capacityProvider=FARGATE,base=2,weight=1 capacityProvider=FARGATE_SPOT,weight=3
```

Infrastructure as code

```
import * as ecsPatterns from 'aws-cdk-lib/aws-ecs-patterns';

new ecsPatterns.ApplicationLoadBalancedFargateService(this, 'App', {
  cluster,
  cpu: 256,
  memoryLimitMiB: 512,
  desiredCount: 2,
  taskImageOptions: {
    image: ecs.ContainerImage.fromEcrRepository(repo, '1.1.0'),
    containerPort: 8080,
  },
});
```

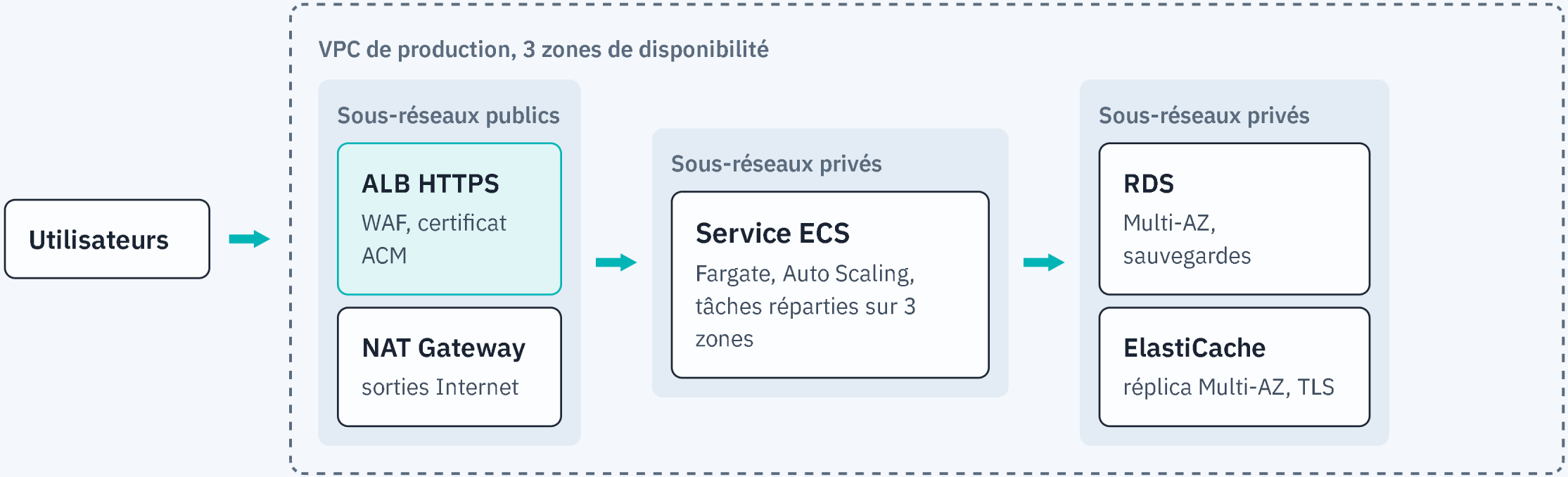
CloudFormation : natif AWS, YAML ou JSON.

AWS CDK : TypeScript ou Python, constructions de haut niveau (ALB et service en un bloc).

Terraform : multi-cloud, état à stocker et verrouiller.

CI : actions amazon-ecs-render-task-definition puis amazon-ecs-deploy-task-definition.

Architecture de production



ECR et scan amélioré | Secrets Manager | Container Insights | Points de terminaison VPC

Synthèse

06

Amazon ECR

Stocker, versionner et scanner les images, puis les distribuer depuis la CI.

07

Amazon ECS

Exécuter les conteneurs avec des task definitions et des services, sur EC2 ou Fargate.

08

Écosystème AWS

ALB en entrée, RDS et ElastiCache pour les données, configuration externalisée.

09

Concepts avancés

Auto Scaling, déploiements progressifs, sécurité, observabilité et maîtrise des coûts.