RAPPORT TD3 - How to Deploy Your Apps

Compte AWS: 681479472828

Région principale: us-east-2

Introduction

Ce TD avait pour objectif de déployer une même application Node.js avec plusieurs paradigmes d'infrastructure et d'orchestration afin de comprendre les compromis techniques entre configuration de serveurs, image immuable, orchestration conteneurisée et serverless.

L'attendu principal n'était pas de produire un simple "step by step", mais d'expliquer les choix techniques, de démontrer une méthode d'ingénierie cohérente et de valider chaque étape avec des preuves observables.

Le travail a été mené comme un mini projet d'exploitation cloud: hypothèse, exécution, observation, correction de la cause racine, puis validation finale par tests HTTP et états de ressources.

Démarche Méthodologique

La démarche suivie repose sur un principe de réduction du risque. En environnement cloud, les erreurs proviennent souvent de dépendances extérieures au code applicatif, notamment IAM, versions de providers, temporalité des ressources managées et choix de type d'instance.

Le projet a donc été découpé en blocs indépendants avec validation explicite de fin de bloc: Ansible, Packer + ASG/ALB, EKS + ECR + Kubernetes, puis Lambda + API Gateway. Cette stratégie a permis de diagnostiquer rapidement les échecs sans mélanger les causes.

Choix Techniques Et Raisons

La structure du dépôt sépare les modules OpenTofu réutilisables (`scripts/tofu/modules`) des compositions d'environnement (`scripts/tofu/live`). Cette séparation améliore la maintenabilité et réduit le risque de régression lors des itérations.

Sur Ansible, l'inventaire dynamique AWS a été retenu plutôt qu'un inventaire statique, car les IP publiques changent fréquemment. Le découpage en rôles rend le déploiement plus lisible et idempotent.

Sur Packer + OpenTofu, une stratégie immuable a été privilégiée: runtime et application sont préparés dans l'AMI, puis l'ASG lance des instances homogènes derrière ALB.

Sur EKS + ECR, la chaîne image -> registre -> cluster -> service a été validée pour reproduire un flux proche production. Enfin, la partie Lambda + API Gateway a servi de comparaison avec une approche serverless sans gestion directe de serveurs.

Extraits De Code Et Explications

L'inventaire dynamique Ansible regroupe automatiquement les hôtes via les tags AWS, ce qui évite de maintenir des adresses IP en dur.

scripts/ansible/inventory.aws_ec2.yml

```
plugin: amazon.aws.aws_ec2
regions:
  - us-east-2
hostvars_prefix: aws_
keyed_groups:
  - key: aws_tags.Ansible
leading_separator: ''
```

Le template Nginx intègre un fallback robuste de résolution d'hôte, ce qui évite les échecs quand certaines variables d'inventaire sont absentes.

scripts/ansible/roles/nginx/templates/nginx.conf.j2

```
upstream backend {
    {% for host in groups['sample_app_instances'] %}
    server {{ hostvars[host].public_dns_name | default(hostvars[host].ansible_host | default(host)) }}:80
    {% endfor %}
}
```

Le build Packer prépare le runtime et PM2 au niveau image. Les instances lancées par ASG deviennent immédiatement exploitables.

scripts/packer/sample-app.pkr.hcl (extrait)

```
provisioner "shell" {
  inline = [
    "set -euxo pipefail",
    "sudo dnf -y update",
    "sudo rm -f /etc/yum.repos.d/nodesource-nsolid.repo",
    "curl -fsSL https://rpm.nodesource.com/setup_21.x | sudo bash -",
    "sudo dnf -y install nodejs",
    "sudo npm install -g pm2",
    "sudo systemctl enable pm2-app-user"
  ]
}
```

Dans OpenTofu, l'ASG utilise une stratégie de rafraîchissement progressif. C'est ce qui garantit des mises à jour d'instances sans interruption massive.

scripts/tofu/modules/asg/main.tf (extrait)

```
dynamic "instance_refresh" {
  for_each = var.instance_refresh == null ? [] : [1]
  content {
    strategy = "Rolling"
    preferences {
      min_healthy_percentage = var.instance_refresh.min_healthy_percentage
      max_healthy_percentage = var.instance_refresh.max_healthy_percentage
      auto_rollback          = var.instance_refresh.auto_rollback
    }
```

```
  }
}
```

Le node group EKS est paramétré avec `instance_types = [var.instance_type]`. Cette ligne a permis de corriger le blocage initial (t2.micro -> t3.micro).

scripts/tofu/modules/eks-cluster/main.tf (extrait)

```
resource "aws_eks_node_group" "nodes" {
  cluster_name    = aws_eks_cluster.cluster.name
  node_group_name = var.name
  instance_types  = [var.instance_type]
}
```

Le déploiement Kubernetes est configuré en `RollingUpdate` avec `maxUnavailable: 0`, ce qui protège la disponibilité pendant les rollouts.

scripts/kubernetes/sample-app-deployment.yml (extrait)

```
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 3
    maxUnavailable: 0
```

La fonction Lambda est packagée avec hash de contenu, ce qui impose des déploiements reproductibles et traçables.

scripts/tofu/modules/lambda/main.tf (extrait)

```
resource "aws_lambda_function" "function" {
  function_name = var.name
  role          = aws_iam_role.lambda.arn
  filename         = data.archive_file.source_code.output_path
  source_code_hash = data.archive_file.source_code.output_base64sha256
  runtime = var.runtime
  handler = var.handler
}
```

Validation Avec Preuves Intégrées

Les preuves ci-dessous ont été générées pendant l'exécution réelle et confirment l'état des ressources ainsi que les réponses applicatives. Chaque preuve comporte un titre, un emplacement d'annexe, la commande exécutée et l'output observé.

Annexe A - Identité AWS Et Ansible

Titre	Horodatage des exécutions
Emplacement	Annexe A1
Input	`date -Is`
Output	Horodatage de collecte des preuves

```
2026-02-18T01:03:29-04:00
```

Titre	Identité AWS active
Emplacement	Annexe A2
Input	`aws sts get-caller-identity`
Output	Compte et ARN de l'utilisateur actif

```
"Account": "681479472828"
"Arn": "arn:aws:iam::681479472828:user/yoan.roul"
```

Titre	Inventaire dynamique Ansible
Emplacement	Annexe A3
Input	`ansible-inventory -i inventory.aws_ec2.yml --graph`
Output	Groupes nginx_instances et sample_app_instances résolus

```
|--@nginx_instances:
|  |--ec2-18-190-217-81.us-east-2.compute.amazonaws.com
|--@sample_app_instances:
|  |--ec2-18-216-25-69.us-east-2.compute.amazonaws.com
|  |--ec2-18-216-69-20.us-east-2.compute.amazonaws.com
|  |--ec2-3-21-158-243.us-east-2.compute.amazonaws.com
```

Titre	État des instances EC2
Emplacement	Annexe A4
Input	`aws ec2 describe-instances --query ... --output table`
Output	Instances présentes avec tags cohérents

```
i-096a0e97f1629d194  nginx_instances-0       18.190.217.81
i-01cc189ddd430a104  sample_app_instances-0  18.216.25.69
i-0cb8f2ee9c5bfaedf  sample_app_instances-1  18.216.69.20
i-08c74287d64eb4076  sample_app_instances-2  3.21.158.243
```

Titre Vérification HTTP via Nginx

Emplacement Annexe A5

Input `curl -i http://<NGINX_IP>`

Output Réponse applicative attendue

```
HTTP/1.1 200 OK
Server: nginx/1.24.0

DevOps Base!
```

Annexe B - Packer + ASG/ALB

Titre AMI la plus récente créée par Packer

Emplacement Annexe B1

Input `aws ec2 describe-images --owners self --filters Name=name,Values=packer-sample-app-*`

Output Identifiant AMI et date de création

```
"AMI": "ami-04b3d8f1e3fd99a15"
"Name": "packer-sample-app-20260218043457"
"Created": "2026-02-18T04:36:33.000Z"
```

Titre Sorties OpenTofu ASG/ALB

Emplacement Annexe B2

Input `tofu output -json (asg-sample)`

Output DNS ALB actif

```
"alb_dns_name": "sample-app-alb-1357069232.us-east-2.elb.amazonaws.com"
```

Titre Vérification HTTP via ALB

Emplacement Annexe B3

Input `curl -i http://<ALB_DNS>`

Output Réponse applicative exposée via load balancer

```
HTTP/1.1 200 OK

DevOps Base!
```

Annexe C - EKS + ECR + Kubernetes

Titre Sorties OpenTofu du cluster EKS
Emplacement Annexe C1
Input `tofu output -json (eks-sample)`
Output Nom et endpoint du cluster

```
"cluster_name": "eks-sample"
"cluster_endpoint": "https://E12F5C179F6A1D2D39B35D73E51A7C02.gr7.us-east-2.eks.amazonaws.com"
```

Titre État du cluster EKS
Emplacement Annexe C2
Input `aws eks describe-cluster --name eks-sample --region us-east-2`
Output Cluster disponible

```
"name": "eks-sample"
"version": "1.29"
"status": "ACTIVE"
```

Titre État du node group EKS
Emplacement Annexe C3
Input `aws eks describe-nodegroup --cluster-name eks-sample --nodegroup-name eks-sample --region us-east-2`
Output Node group actif en t3.micro

```
"status": "ACTIVE"
"instanceTypes": ["t3.micro"]
```

Titre Sortie OpenTofu du registre ECR
Emplacement Annexe C4
Input `tofu output -json (ecr-sample)`
Output URL du registre privé

```
"registry_url": "681479472828.dkr.ecr.us-east-2.amazonaws.com/sample-app"
```

Titre	Propriétés du repository ECR
Emplacement	Annexe C5
Input	`aws ecr describe-repositories --region us-east-2`
Output	Repository immuable

```
"repositoryName": "sample-app"
"imageTagMutability": "IMMUTABLE"
```

Titre	Service Kubernetes exposé
Emplacement	Annexe C6
Input	`kubectl get svc sample-app-loadbalancer`
Output	LoadBalancer avec EXTERNAL-IP

```
NAME                     TYPE           EXTERNAL-IP
sample-app-loadbalancer  LoadBalancer   ac181c1d4da5d4d6d94cbde4933a3543-1771289685.us-east-2.e
```

Titre	Vérification HTTP Kubernetes
Emplacement	Annexe C7
Input	`curl -i http://<EXTERNAL_IP_K8S>`
Output	Réponse applicative via service EKS

```
HTTP/1.1 200 OK

DevOps Base!
```

Annexe D - Lambda + API Gateway

Titre	Sortie OpenTofu API Gateway
Emplacement	Annexe D1
Input	`tofu output -json (lambda-sample)`
Output	Endpoint API exposé

```
"api_endpoint": "https://19si4gvgg0.execute-api.us-east-2.amazonaws.com"
```

Titre Vérification HTTP API Gateway

Emplacement Annexe D2

Input `curl -i https://<API_ENDPOINT>/`

Output Réponse fonctionnelle serverless

```
HTTP/2 200

Hello, World!
```

Titre Configuration de la fonction Lambda

Emplacement Annexe D3

Input `aws lambda get-function --function-name lambda-sample --region us-east-2`

Output État opérationnel de la fonction

```
"FunctionName": "lambda-sample"
"Runtime": "nodejs20.x"
"State": "Active"
"LastUpdateStatus": "Successful"
```

Difficultés Rencontrées Et Résolution

La difficulté la plus coûteuse en temps a été la création du node group EKS. Le premier déploiement échouait en `CREATE_FAILED` avec `t2.micro`. Le passage à `t3.micro` a résolu le blocage.

Un second incident critique concernait l'ASG et le rôle lié au service Auto Scaling. Tant que `AWSServiceRoleForAutoScaling` n'était pas disponible, le déploiement ne pouvait pas converger. La vérification puis la création explicite du rôle ont débloqué la situation.

Sur Ansible, la clé SSH et la résolution de variables d'inventaire ont été les deux causes racines principales. Les corrections apportées ont stabilisé les playbooks et permis la configuration distante complète.

Utilisation De L'IA Et Positionnement D'Équipe

L'IA a été utilisée comme accélérateur de diagnostic lorsque plusieurs couches échouaient en parallèle. Elle a surtout servi à structurer les hypothèses et à prioriser les actions correctives.

Le travail intellectuel principal est resté porté par l'équipe. Chaque correction a été validée par exécution réelle, lecture d'outputs AWS/OpenTofu et test HTTP final. L'IA a donc réduit le temps de recherche sans remplacer la compréhension technique.

Bilan Pédagogique Et Perspectives

Ce TD a permis de comparer concrètement les différences entre configuration mutable, image immuable, orchestration conteneurisée et serverless. Le principal apprentissage est que la réussite cloud dépend autant des

choix d’architecture que de la maîtrise opérationnelle (IAM, versions, santé et temporalité des ressources).

Les perspectives logiques sont l’ajout d’une CI/CD complète, d’une observabilité centralisée et d’un durcissement IAM plus fin pour passer d’un laboratoire validé à un socle réellement exploitable.

Conclusion

Le TD3 est terminé et validé fonctionnellement sur les quatre volets demandés. Le rapport présente les choix techniques, la méthode d’analyse, les difficultés réellement rencontrées et les preuves d’exécution associées.