

Lab 4: Synthèse, déployer une infrastructure de A à Z

Cloud Computing

Université Claude Bernard Lyon 1 May 28, 2026

Assignment Overview

Document Contents:

1. Objectif	2
2. Pré-requis	2
2.1. Conseils	2
3. Contexte	2
4. Cahier des charges	3
4.1. Réseau	3
4.2. Filtrage	3
4.3. Compute	3
4.4. Stockage	3
4.5. Conventions	3
4.6. Livrables	3
5. Étapes suggérées	5
5.1. Étape 1: Préparer le squelette du projet .	5
5.2. Étape 2: Déclarer les variables	5
5.3. Étape 3: Construire le réseau	5
5.4. Étape 4: Écrire les security groups	5
5.5. Étape 5: Déployer les instances EC2	5
5.6. Étape 6: Créer le bucket et son contenu .	6
5.7. Étape 7: Définir les sorties	6
5.8. Étape 8: Déployer et vérifier	6
5.9. Étape 9: Nettoyage	6
6. Grille de validation	7
7. Questions	7
8. Aller plus loin	7

Assignment Details:

- **Course:** Cloud Computing
- **Instructor:** Hugo Blanc
- **Software:** Terraform, AWS, LocalStack
- **Duration:** ~ 2 heures
- **Lab Number:** Lab 4

Objectif

Vous avez vu les fondamentaux (TP 1), le déploiement d'une instance EC2 dans un VPC (TP 2), et la construction d'une topologie réseau complète avec routage et filtrage (TP 3). Ce TP est différent : **aucun fichier Terraform ne vous est fourni**. Vous allez écrire vous-mêmes l'intégralité de la configuration pour répondre à un cahier des charges. L'objectif est de consolider les notions précédentes en passant du mode "lecture" au mode "production".

Pré-requis

- Avoir Terraform ($\geq 1.5.0$);
- avoir `tflocal` (`pip3 install terraform-local`);
- avoir LocalStack qui tourne;
- avoir la CLI AWS (voir la documentation officielle).

Conseils

Je vous recommande de faire les alias suivants:

```
alias tf=tflocal
alias aws="aws --endpoint-url=http://localhost:4566"
```

Également, pour simplifier l'usage de la CLI AWS, je vous recommande de définir des faux credentials pour éviter les erreurs:

```
export AWS_ACCESS_KEY_ID=test
export AWS_SECRET_ACCESS_KEY=test
export AWS_DEFAULT_REGION=eu-west-1
```

Contexte

LocalStack émule des services AWS en local sur la machine, sur le port 4566. La commande `tflocal` est un wrapper autour de Terraform qui permet de rediriger les appels d'API vers LocalStack.

Vous travaillez dans une équipe d'ingénieurs cloud. On vous demande de provisionner la base d'une plateforme web composée d'une partie publique (frontend) et d'une partie privée (backend), avec un espace de stockage objet pour les fichiers statiques. Le tout doit être reproductible, paramétrable, et propre à votre identifiant étudiant pour éviter les collisions de noms.

Vous travaillerez dans le dossier `04-synthese/` qui est **vide à dessein**. Tous les fichiers que vous y créez doivent l'être de vos mains.

Cahier des charges

L'infrastructure attendue est la suivante :

Réseau

- Un **VPC** dans le bloc CIDR 10.42.0.0/16, avec DNS activé.
- **Deux subnets publics** : 10.42.1.0/24 en eu-west-1a, 10.42.2.0/24 en eu-west-1b. Attribution automatique d'IP publique à la création d'instance.
- **Deux subnets privés** : 10.42.10.0/24 en eu-west-1a, 10.42.20.0/24 en eu-west-1b.
- Une **Internet Gateway** attachée au VPC.
- Une **NAT Gateway** placée dans le subnet public de eu-west-1a, avec sa propre **Elastic IP**.
- Une **table de routage publique** (route par défaut vers l'IGW) associée aux deux subnets publics.
- Une **table de routage privée** (route par défaut vers la NAT Gateway) associée aux deux subnets privés.

Filtrage

- Un security group web-sg qui autorise en entrée le port 80 et 443 depuis Internet, et tout en sortie.
- Un security group app-sg qui autorise en entrée le port 8080 **uniquement depuis web-sg**, et tout en sortie.

Compute

- Une instance EC2 web dans un subnet public, associée au security group web-sg.
- Une instance EC2 app dans un subnet privé, associée au security group app-sg.
- L'instance web doit être lancée avec un user_data shell minimal (par exemple echo "hello from <student_name>" > /tmp/hello.txt).

Limite LocalStack

LocalStack en édition communautaire émule l'API EC2 mais ne lance pas de vraie machine virtuelle. Le user_data est stocké comme attribut de l'instance, mais **n'est pas exécuté**. La validation se fait en lisant l'attribut userData via la CLI, pas en se connectant à l'instance.

Stockage

- Un bucket S3 nommé <student_name>-tp04-assets.
- Le **versioning** du bucket doit être activé.
- Un objet manifest.json créé par Terraform à la racine du bucket. Son contenu doit être un JSON contenant au moins student_name et vpc_cidr.

Conventions

- Toutes les ressources sont préfixées par la variable student_name.
- Toutes les ressources portent les tags communs : Lab = "04", Student = <student_name>, ManagedBy = "terraform".
- Le nom étudiant est en minuscules, alphanumérique avec tirets, 2 à 21 caractères, commence par une lettre.

Livrables

Dans le dossier 04-synthese/, vous devez produire au minimum :

- versions.tf : déclaration du provider AWS et de la version de Terraform.

- `variables.tf` : variables d'entrée (au moins `student_name`, `vpc_cidr`, les CIDR de subnets).
- `main.tf` : l'ensemble des ressources.
- `outputs.tf` : les sorties listées ci-dessous.

Sorties attendues :

```
vpc_id
public_subnet_ids
private_subnet_ids
web_instance_public_ip
app_instance_private_ip
web_security_group_id
app_security_group_id
bucket_name
bucket_versioning_status
```

Étapes suggérées

L'ordre suivant est une suggestion. Rien ne vous empêche de procéder différemment, mais cet ordre minimise les erreurs de dépendance.

Étape 1: Préparer le squelette du projet

Créez les quatre fichiers `.tf` vides. Définissez dans `versions.tf` le provider `hashicorp/aws` (version `~> 5.0`) et la contrainte Terraform `>= 1.5.0`. Initialisez le projet avec `tflocal init`.

Question

Pourquoi vaut-il mieux figer la version du provider plutôt que d'écrire `version = ">= 0"` ? Que se passe-t-il si AWS publie une version `6.0` qui introduit des changements incompatibles ?

Étape 2: Déclarer les variables

Déclarez vos variables dans `variables.tf`. Pour `student_name`, ajoutez un bloc `validation` qui rejette les noms ne respectant pas la convention.

Question

Pourquoi vouloir des validations sur des variables d'entrée plutôt que de faire confiance à l'utilisateur ? À quel moment du cycle Terraform ces validations sont-elles évaluées ?

Étape 3: Construire le réseau

Créez la VPC, les subnets, l'Internet Gateway, l'Elastic IP, la NAT Gateway et les deux tables de routage. Pensez à associer chaque subnet à la bonne table.

Question

Sans `depends_on`, dans quel ordre Terraform créera-t-il la VPC, l'IGW, l'Elastic IP et la NAT Gateway ? Justifiez à partir des références entre ressources.

Étape 4: Écrire les security groups

Créez `web-sg` et `app-sg`. Pour la règle d'entrée de `app-sg`, référez le security group source par son ID Terraform (pas par une plage CIDR).

Question

Quel est l'effet concret de référencer le security group source par son ID plutôt que par le CIDR du subnet public ? Donnez un scénario où cela change le comportement.

Étape 5: Déployer les instances EC2

Créez les deux instances `web` et `app`. Vous pouvez utiliser n'importe quel ami arbitraire (LocalStack ne valide pas l'AMI). Attachez chaque instance au bon subnet et au bon security group.

Pour le `user_data` de `web`, un script shell minimal suffit :

```
#!/bin/bash
echo "hello from <student_name>" > /tmp/hello.txt
```

Encodez-le avec `base64encode()` si nécessaire.

Question

Le `user_data` est-il exécuté à chaque `tflocal apply` ou seulement à la création de l'instance ? Que se passerait-il si vous modifiez le contenu du script après création ?

Étape 6: Créer le bucket et son contenu

Créez le bucket S3, activez le versioning via la ressource dédiée (`aws_s3_bucket_versioning`), et déposez un objet `manifest.json`. Vous pouvez générer son contenu avec la fonction `jsonencode()` de Terraform.

Question

Pourquoi le versioning S3 est-il géré par une **ressource séparée** (`aws_s3_bucket_versioning`) plutôt que par un argument du bucket lui-même ? Quel autre service AWS suit ce même pattern ?

Étape 7: Définir les sorties

Renseignez `outputs.tf` avec toutes les sorties listées dans le cahier des charges. Pour `bucket_versioning_status`, vous devrez lire l'attribut approprié de la ressource de versioning.

Étape 8: Déployer et vérifier

```
tflocal plan -var="student_name=your-name"
tflocal apply -var="student_name=your-name"
```

Vérifiez ensuite avec la CLI AWS :

```
aws ec2 describe-vpcs
aws ec2 describe-instances
aws ec2 describe-security-groups
aws ec2 describe-nat-gateways
aws ec2 describe-instance-attribute \
  --instance-id <web-instance-id> --attribute userData \
  --query 'UserData.Value' --output text | base64 -d
aws s3api list-objects --bucket your-name-tp04-assets
aws s3api get-bucket-versioning --bucket your-name-tp04-assets
aws s3api get-object --bucket your-name-tp04-assets --key
manifest.json /tmp/manifest.json && cat /tmp/manifest.json
```

Question

Comparez les sorties Terraform à ce que vous voyez via la CLI AWS. Les IDs correspondent-ils ? Si une divergence apparaît, d'où peut-elle venir ?

Étape 9: Nettoyage

```
tflocal destroy -var="student_name=your-name"
```

Grille de validation

Votre rendu sera considéré comme correct si :

1. `tflocal validate` passe sans erreur.
2. `tflocal plan` n'affiche aucune ressource à modifier après un `apply` (idempotence).
3. Toutes les sorties listées sont définies et non vides.
4. Le bucket S3 a bien le versioning à l'état Enabled.
5. L'objet `manifest.json` existe et contient `student_name` et `vpc_cidr`.
6. Le subnet privé n'a pas de route directe vers l'IGW.
7. Le security group `app-sg` référence `web-sg` par ID, pas par CIDR.

Questions

1. **Quel est l'intérêt d'une variable `student_name` et pourquoi est-elle injectée dans la majorité des noms et tags ?** Que se passerait-il si deux étudiants déployaient simultanément sans cette variable dans un compte AWS partagé ?
2. **Vous avez vu trois manières d'exprimer une dépendance entre ressources : implicite par référence (`aws_subnet_public.id`), explicite par `depends_on`, et via un module.** Donnez un exemple où chacune est préférable.
3. **Dans ce TP, vous créez la NAT Gateway avant l'EIP grâce à `depends_on` ?** Pourquoi le contraire serait-il logique en théorie, et pourquoi en pratique l'EIP doit exister avant la NAT ?
4. **Quelles ressources de ce TP ne seraient pas créables sur LocalStack en édition communautaire ?** Argumentez à partir de la documentation LocalStack et de votre expérience du TP.
5. **Comment garantiriez-vous qu'un second étudiant qui clone votre dépôt obtient exactement la même version du provider AWS ?** Quel fichier joue ce rôle, et faut-il le commiter ?
6. **Le fichier `terraform.tfstate` contient les valeurs des sorties et l'état complet du déploiement, y compris parfois des secrets.** Quel risque cela pose-t-il, et quelles solutions existent pour ne pas l'avoir en clair sur disque ?
7. **Reprenez votre `main.tf` et identifiez les opportunités de factorisation (utilisation de `count`, `for_each`, `locals`).** Lesquelles amélioreraient la maintenabilité, lesquelles seraient prématurées ?

Aller plus loin

- Remplacez les deux subnets publics par un `for_each` indexé par zone de disponibilité.
- Ajoutez une NACL sur les subnets privés qui n'autorise en entrée que le port 8080 depuis le CIDR des subnets publics.
- Ajoutez un second bucket S3 `<student_name>-tp04-logs` avec une lifecycle policy qui expire les objets après 7 jours.
- Factorisez le réseau (VPC, subnets, routes, NAT) dans un module local `modules/network/`.
- Déplacez vos commandes de validation dans un script shell `validate.sh` que vous pourrez rejouer après chaque modification.