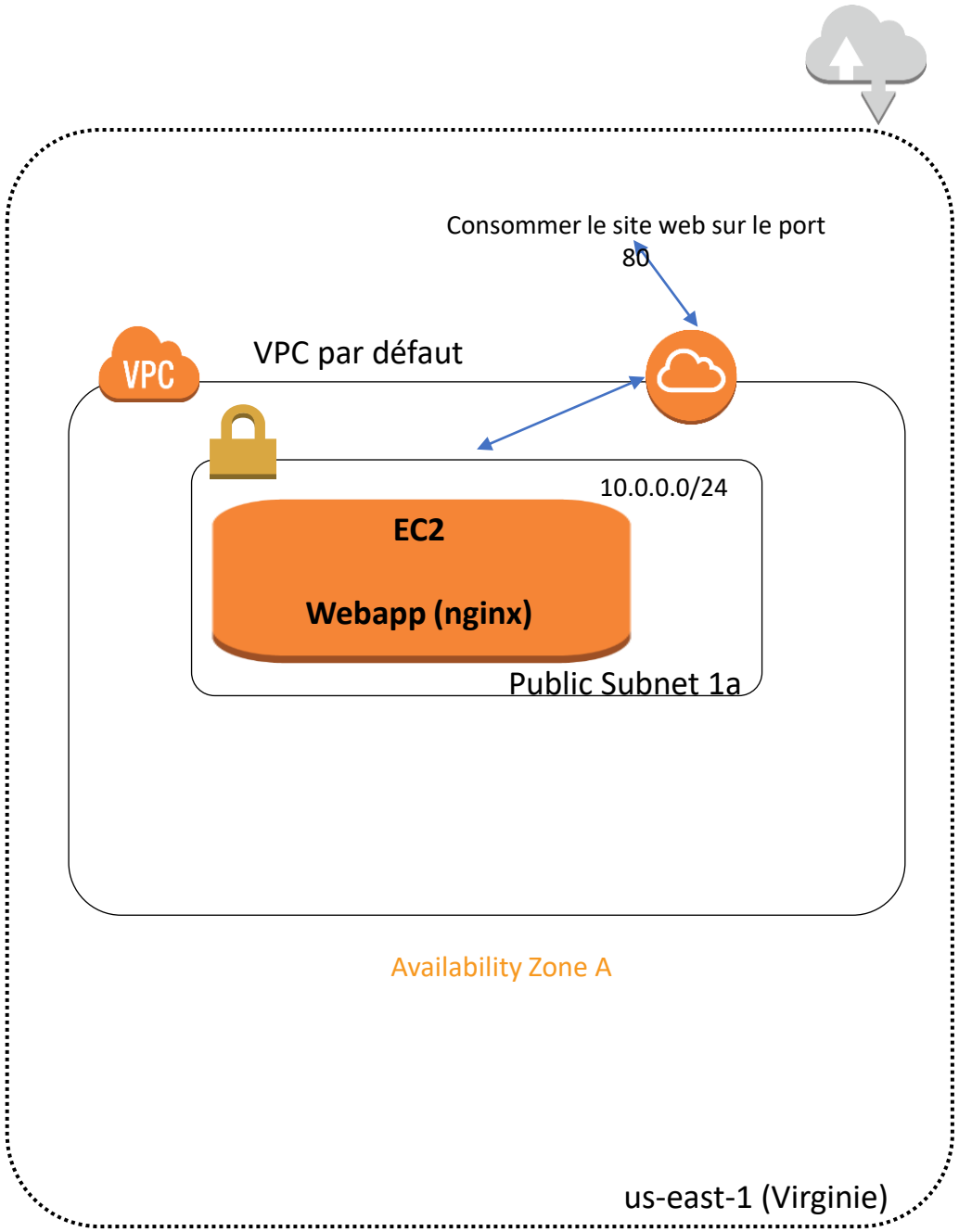
TP-MINI PROJET AWS-DEVOPS

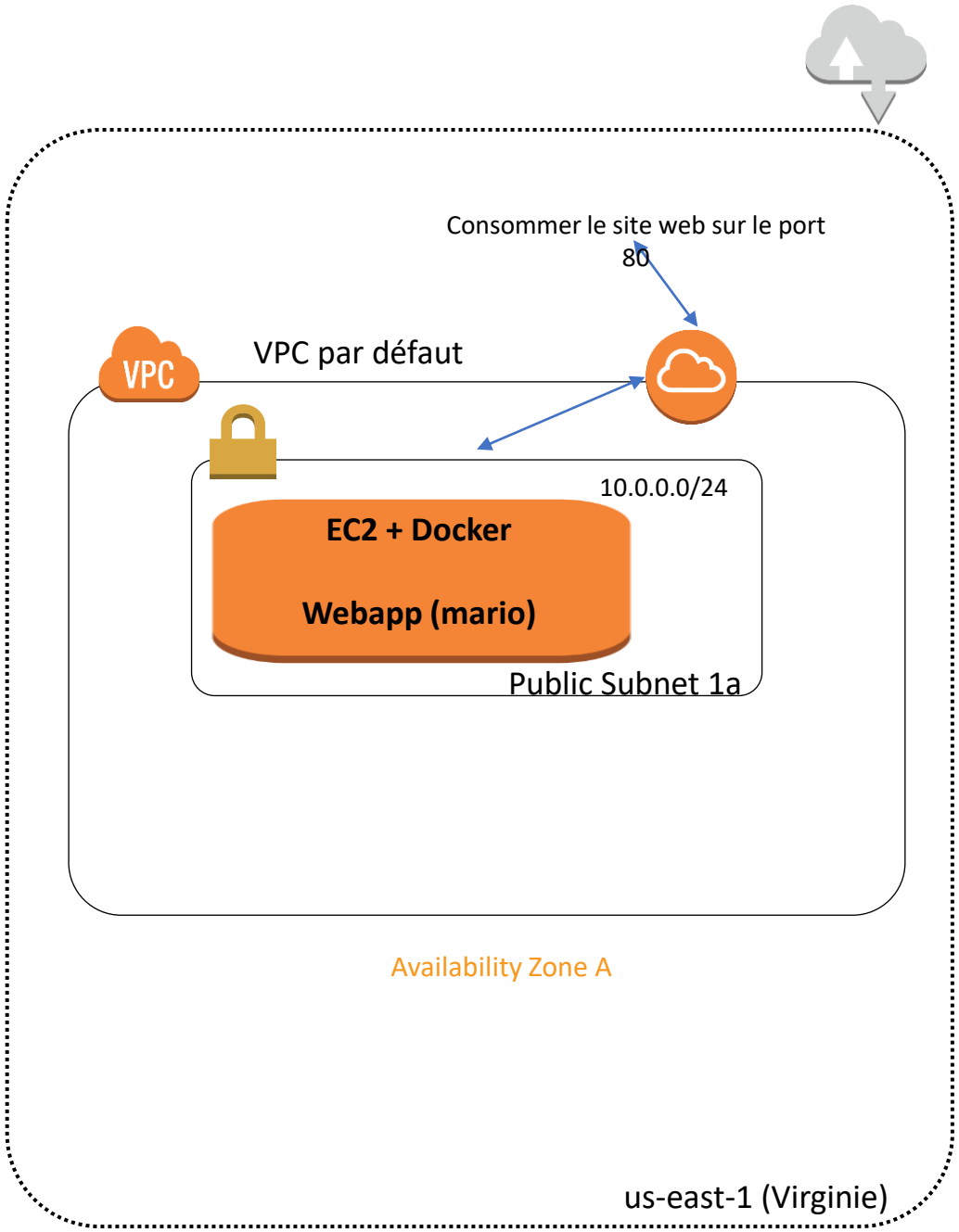
EC2



Enoncé TP1: Script de démarrage + serveur Web

Créer une instance Ec2 à partir du VPC par défaut dans laquelle vous déploierez automatiquement à l'initialisation de la machine, un site web qui devra être consommé directement après le démarrage de la machine à travers son port 80.
Utilisez le serveur web nginx
Ec2: t2.micro, 8Gb
OS: Ubuntu.

NB: Le code applicatif du site web à héberger sur cette machine se trouve sur le repo suivant :
https://github.com/daviddias/static-webpage-example.git

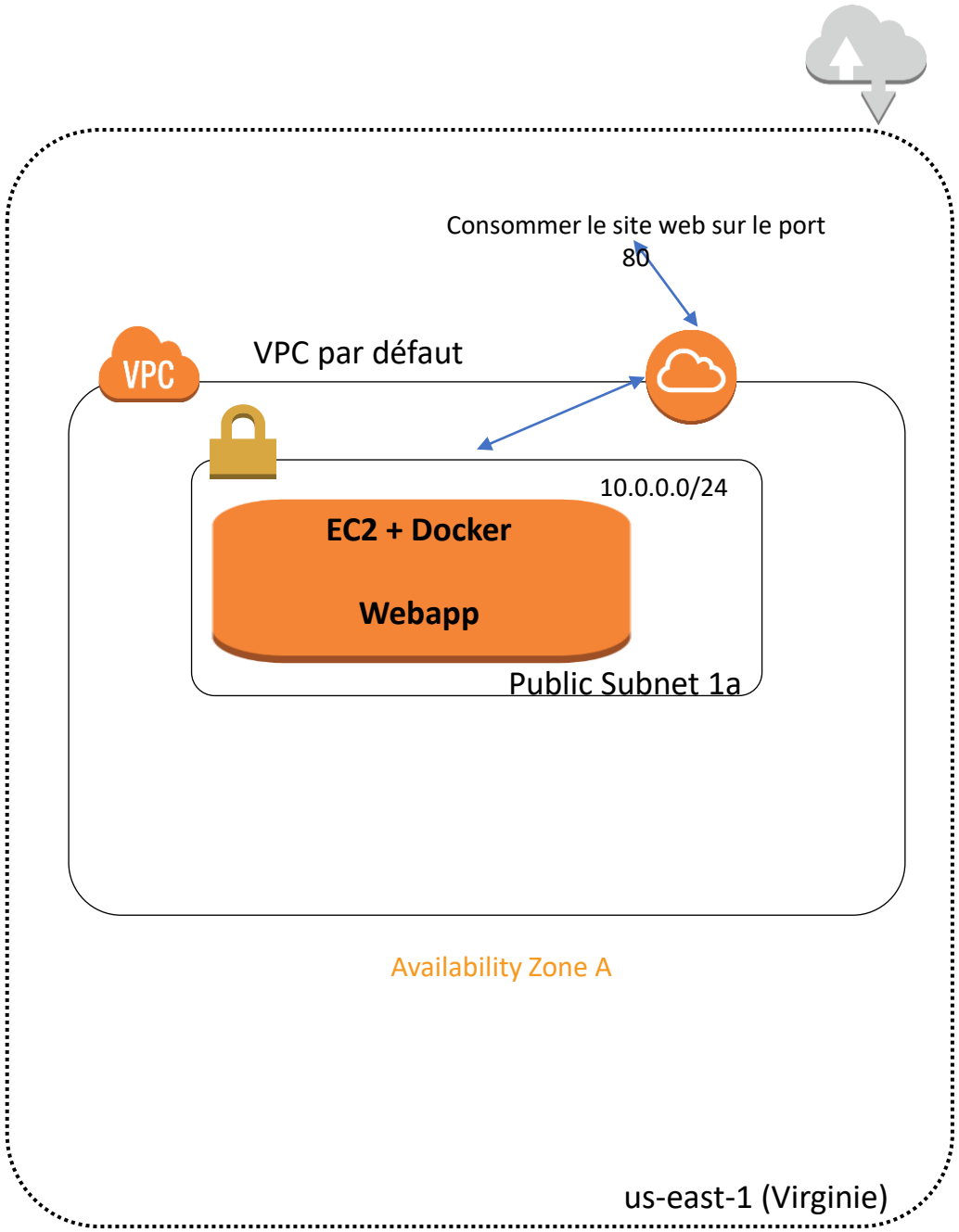


Enoncé TP2: Script de démarrage + Docker

Créer une instance Ec2 à partir du VPC par défaut dans laquelle vous déploierez automatiquement à l'initialisation de la machine une application web (mario) à l'aide de container docker. Cette application web devra être consommé directement après le démarrage de la machine à travers son port 80.

Ec2: t2.micro, 8Gb
OS: Ubuntu.
Image docker : pengbai/docker-supermario
Port exposé dans le container : 8080

NB: pensez à installer docker sur votre machine et donner les droits à l'utilisateur ubuntu afin qu'il puisse exécuter les commandes docker



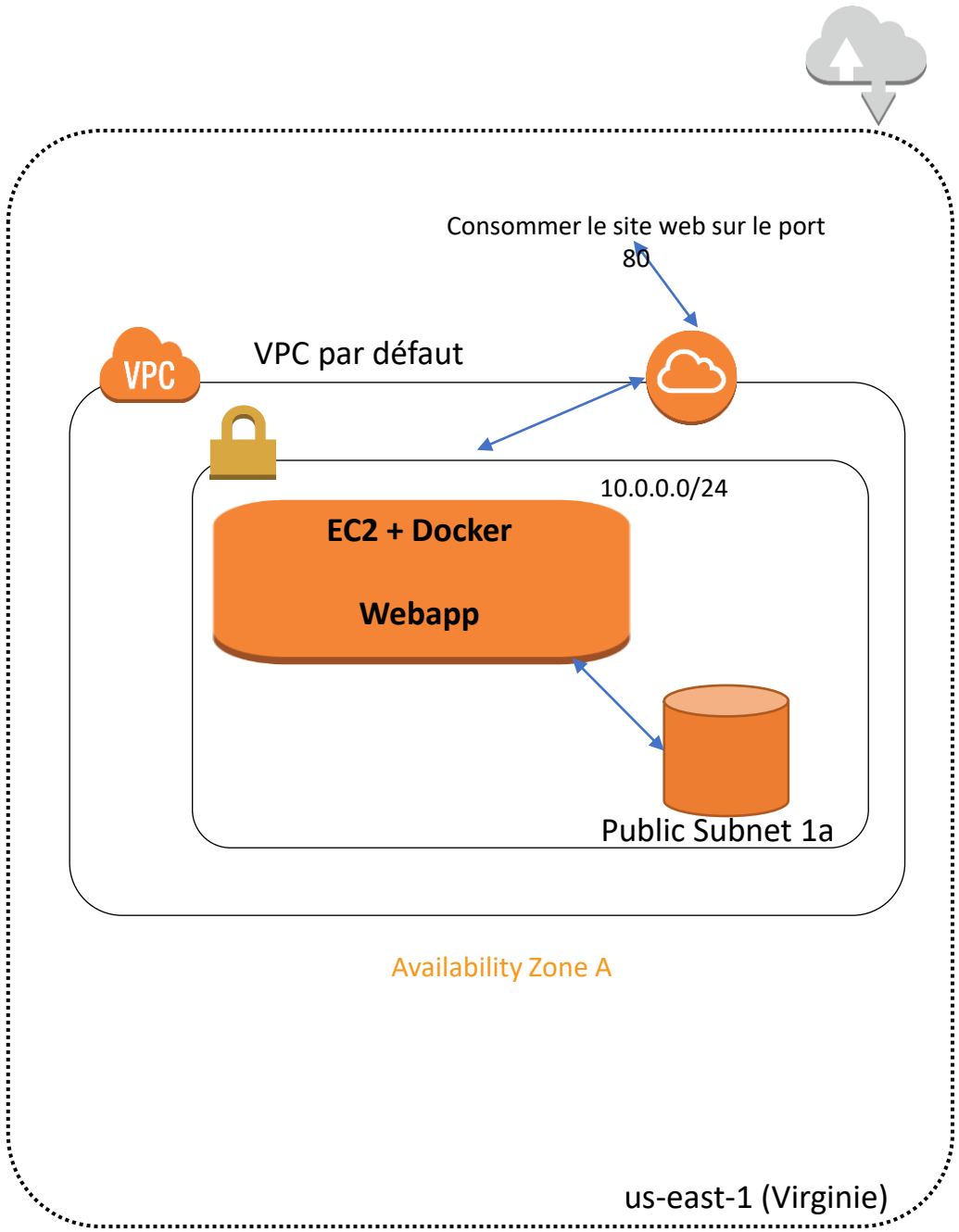
Enoncé TP3: Script de démarrage + Docker + connexion SSH

Créer une instance Ec2 à partir du VPC par défaut dans laquelle vous déploierez automatiquement à l'initialisation de la machine une application web (Intranet) à l'aide de container docker. Cette application web devra être consommé directement après le démarrage de la machine à travers son port 80.

Au démarrage de la machine, vérifiez que le site web est bien joignable au travers du port 80.
Ensuite connectez vous en SSH à la machine et récupérez le contenu du fichier */opt/releases.txt* du container que vous saisirez dans le chat une fois terminé.

Ec2: t2.micro, 8Gb
OS: Ubuntu.
Image docker : sadofrazer/ic-webapp:1.0
Port exposé dans le container : 8080

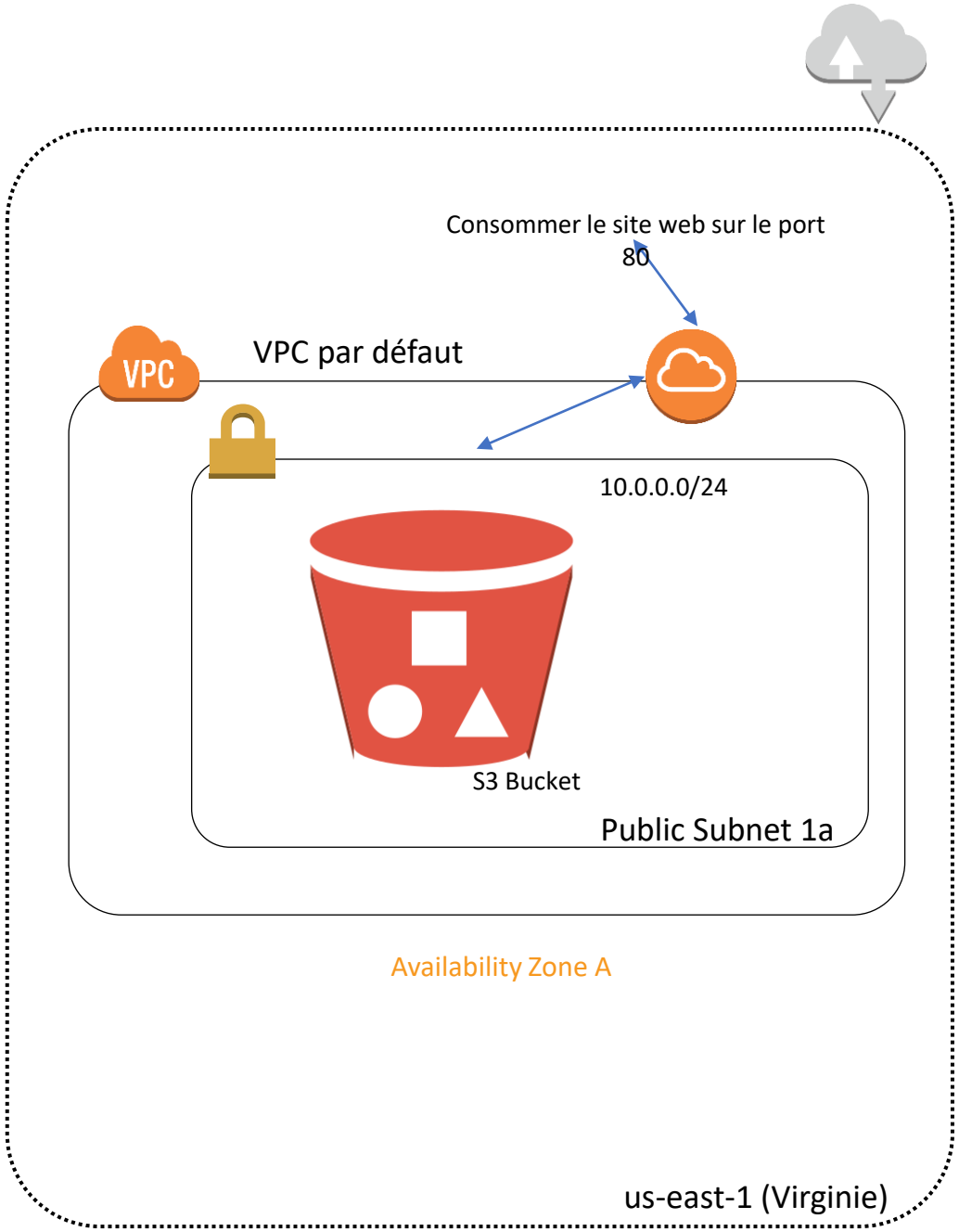
STOCKAGE



Enoncé TP4: EBS + Snapshot

En utilisant la même infrastructure qu'au TP précédent, ajoutez un volume EBS de 1Gb à votre instance EC2.
Puis connectez vous en SSH à cet instance et réalisez les taches suivantes :

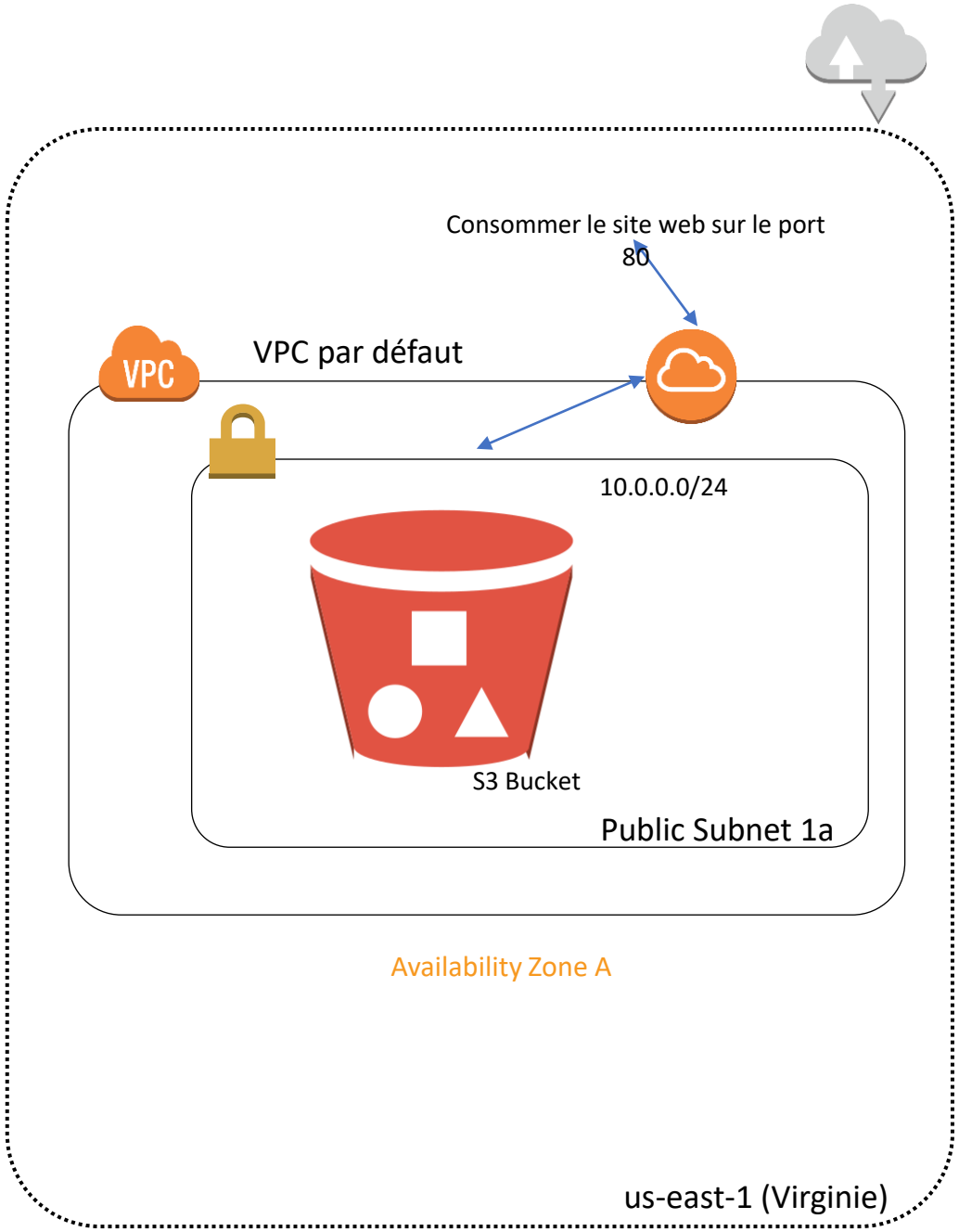
1) Vérifiez que le nouveau disque ait bien été ajouté en listant les disques au travers de la commandes (*ls /dev*)
2) Formatez le disque à l'aide de cette commande (*sudo mkfs.ext3 /dev/xvdf*). Pensez à vérifiez et modifiez si nécessaire la référence du nouveau volume xvd(?).
3) Créez un répertoire /mnt/data-store/
4) Monter le nouveau volume à ce repertoire grâce à la commande (*sudo mount /dev/sdf /mnt/data-store*)
5) Vérifiez que le volume est bien monté *(df –h)*
6) Récupérez à nouveau le contenu du fichier /opt/releases.txt du précédent container et stockez le dans /mnt/data-store/releases-backup.txt
7) Après avoir sauvegardé le contenu du fichier de votre container, faites un snapshot ou instantané de votre nouveau volume
8) Supprimez ce volume une fois le snapshot crée et simulez une panne en redémarrant votre machine (EC2)
9) Une fois le serveur redémarré, servez vous du snapshot précédemment crée pour récupérer le fichier releases-backup.txt



Enoncé TP5: S3 Bucket; partage de fichiers

Créez un bucket S3 et mettez y une image de votre choix que vous rendrez publique afin de pouvoir la partager au travers de votre S3

Ensuite copiez l'url permettant de télécharger votre image et partagez la dans e chat

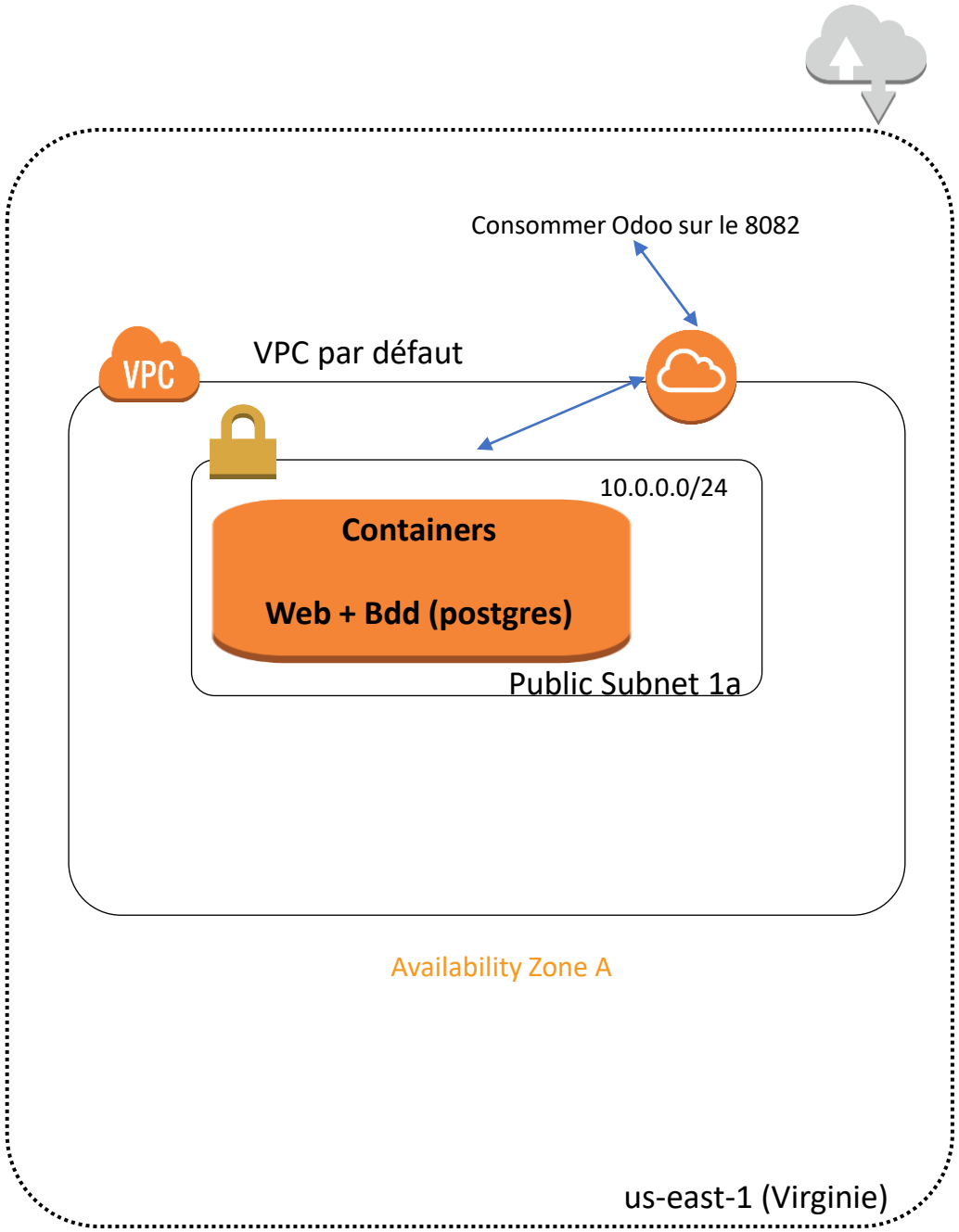


Enoncé TP6: S3 Bucket; Hébergement de site web statique

Créez un bucket S3 et mettez les fichiers d'un site web de votre choix (site web contenant un fichier index.html)
Si vous n'avez pas de site web à votre disposition, utilisez les fichiers se trouvant dans le repo suivant : https://github.com/diranetafen/static-website-example.git

Ensuite réalisez les configurations nécessaire afin d'héberger ce site web au travers de votre S3 (page web atteignable au travers de l'url du S3)

RESEAU



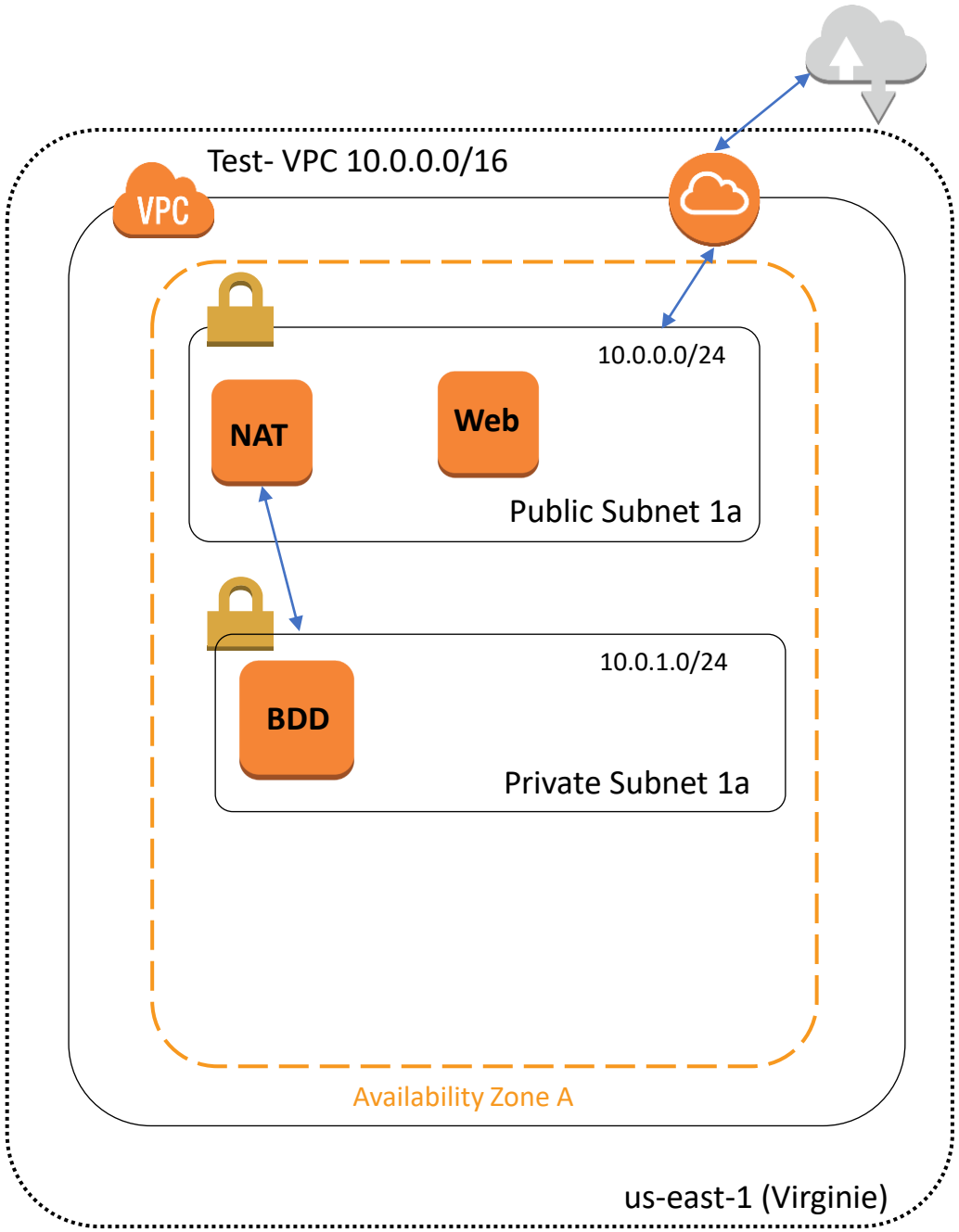
Enoncé TP7: Rappel Docker, Déploiement Odoo

Créer une instance Ec2 à partir du VPC par défaut dans laquelle vous déploierez l'application Odoo à l'aide de Docker.

Donc dans cette instance on devrait avoir 02 containers : un pour la base de données et un autre pour l'application web pointant sur cette BDD. Vous devrez également créer un réseau dans docker appelé odoo (subnet: 192.168.40.0/24) qui devra contenir ces deux containers

Les containers devront être crées automatiquement à l'aide de script Bash lors de la création de l'instance EC2

NB: Vous servir de la documentation sur le déploiement de odoo à l'aide de container : https://hub.docker.com/_/odoo



<u>Enoncé TP8</u>: VPC

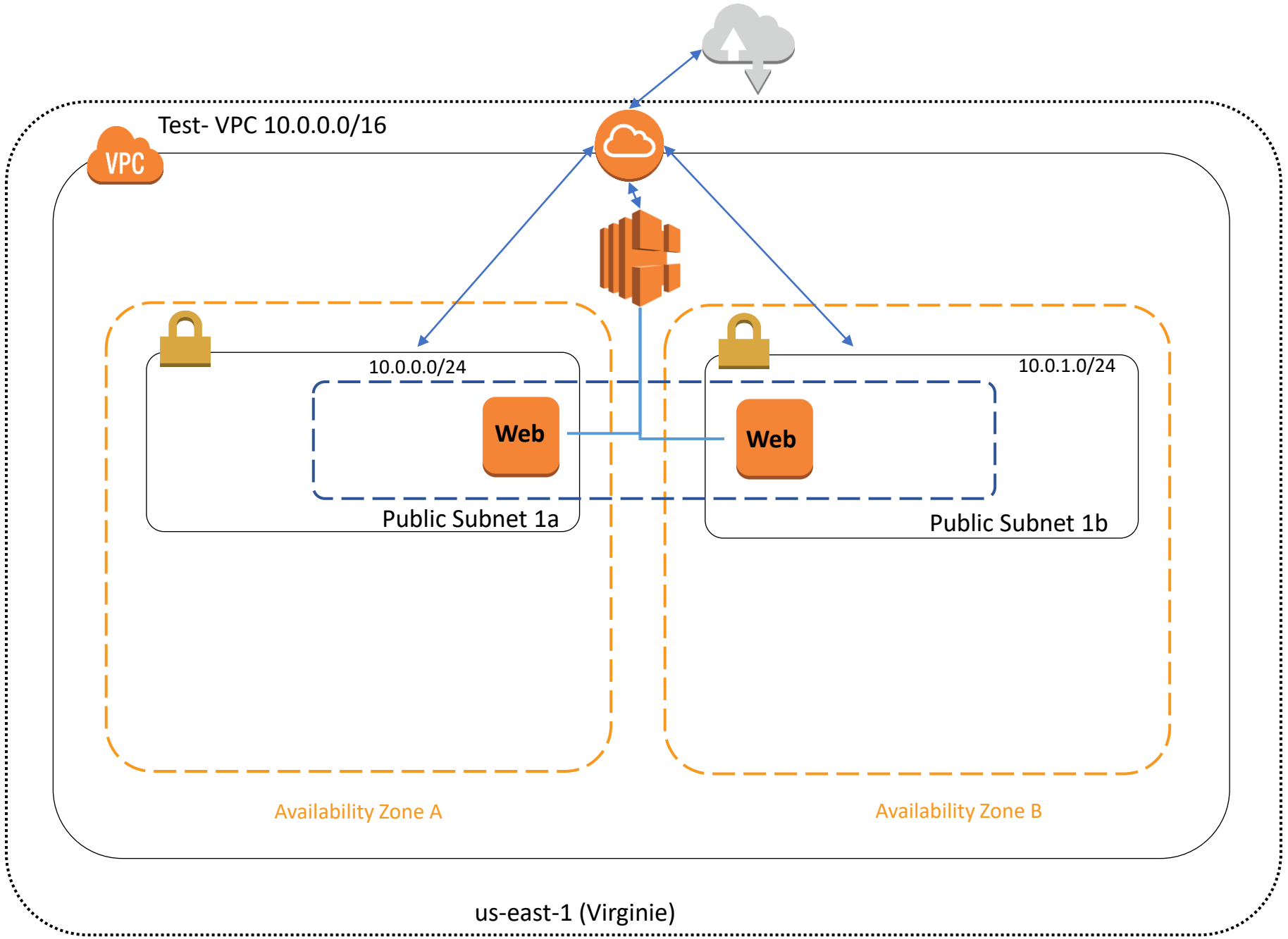
1) Créer un VPC avec des sous-réseaux respectant l'architecture ci-contre.

2) Déployer à l'aide d'un script Bash que vous mettrez dans le user data lors de la création de votre ec2 un container docker de postgres qui sera utilisée comme BDD dans le réseau privé

3) Déployer le container Odoo à l'aide également d'un script bash dans l'EC2 Web qui sera dans le réseau public et consommable par nos clients à travers le port 8082; le container Odoo devra utiliser la base de données postgres sur l'ec2 BDD pour fonctionner

NB: Vous servir de la documentation sur le déploiement de Odoo à l'aide de container : https://hub.docker.com/_/odoo

RESEAU + ELB



Enoncé TP9:

1) Créer un VPC avec des sous-réseaux respectant l'architecture ci-contre.

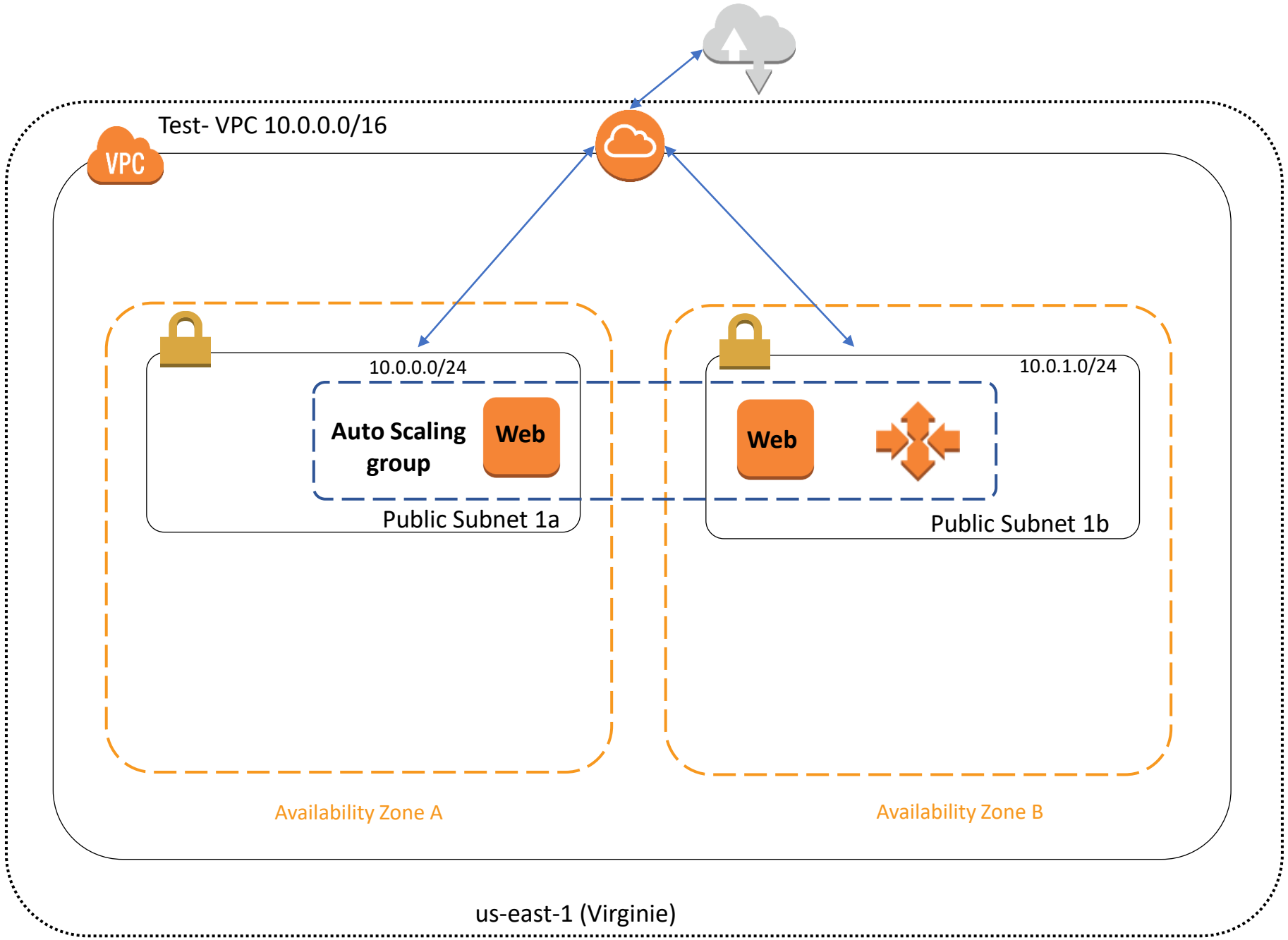
2) Créer une instance EC2 qui hébergera un site web à l'aide de nginx; installez nginx, créez et déplacez dans le répertoire qui va bien un fichier index.html qui contiendra (Bonjour de la part de <votre nom>)

3) Créer une deuxième instance EC2 qui hébergera un site web à l'aide de nginx; installez nginx, créez et déplacez dans le répertoire qui va bien un fichier index.html qui contiendra (Bonjour de la part de AJC)

4) Créez un ELB qui devra permettre de fédérer ces deux instances à travers une seule adresse dns.

5) Récupérez l'adresse dns de votre ELB et testez là en actualisant plusieurs fois la page web (Que se passe t-il)
6) Une fois terminé, partagez votre adresse dns (ELB) dans le chat.

RESEAU + ASG



Enoncé TP10:

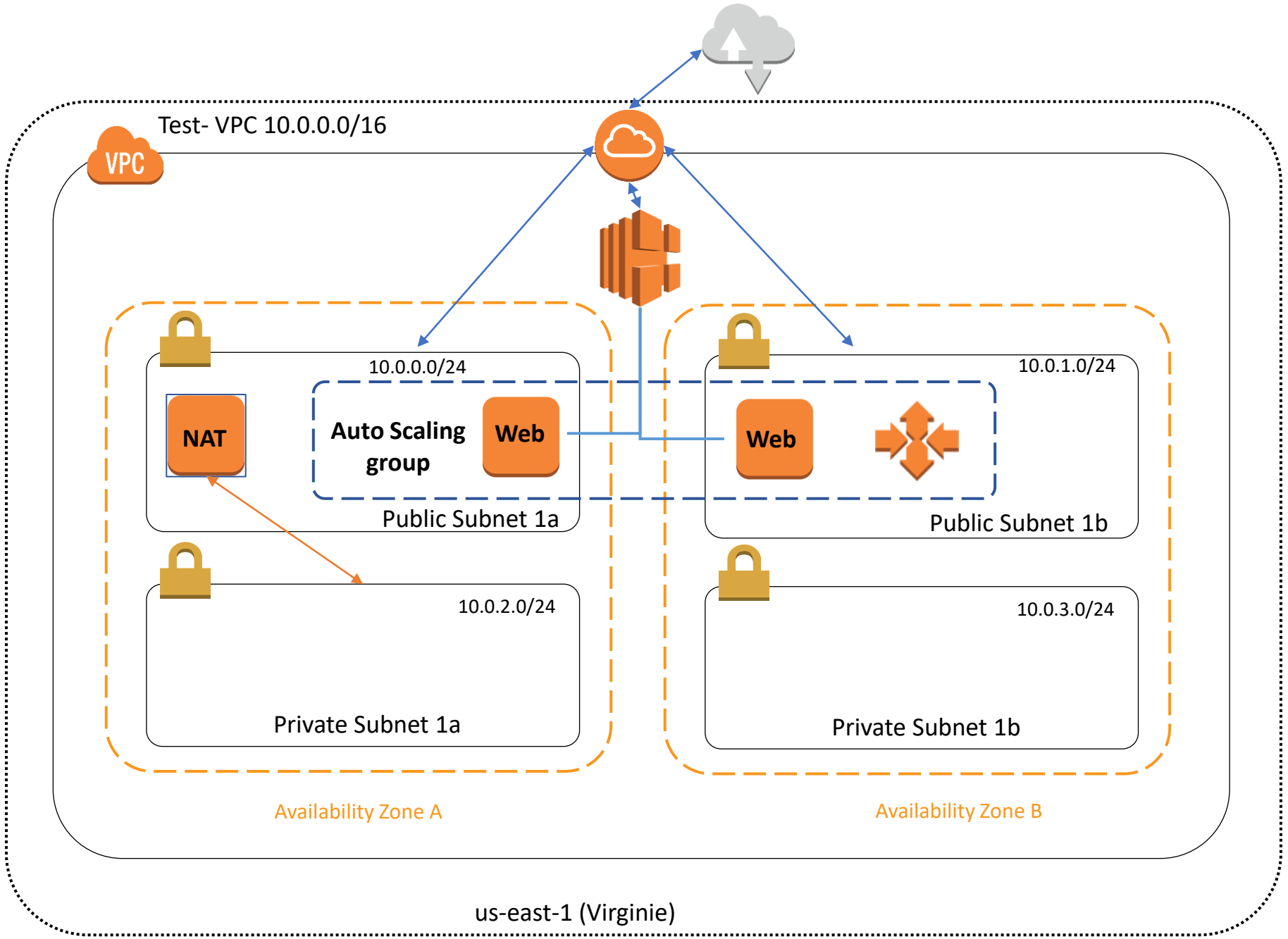
1) Créer un VPC avec des sous-réseaux respectant l'architecture ci-contre.

2) Déployer le site web contenu dans le repo ci-dessus à l'aide d'un script dans une EC2

3) Une fois le serveur démarré, testez le bon fonctionnement de votre site
4) Si OK, créez une image de votre Ec2 fonctionnelle
5) A l'aide de cette image, créez une configuration de lancement que vous utiliserez plus tard pour créer un ASG
6) Creez un ASG à partir de cette configuration de lancement qui déploiera un minimum de 02 instances, maximum de 04. Les instances doivent pouvoir être crées par l'ASG dans les deux zones de disponibilités spécifiées. La politique de suivi de cible de l'ASG doit être liée au CPU pour une capacité d'utilisation moyenne de 70%.

NB: Lien utile https://github.com/daviddias/static-webpage-example.git

RESEAU + ASG +ELB



Enoncé TP11:

1) Créer un VPC avec des sous-réseaux respectant l'architecture ci-contre.

2) Déployer votre site web à l'aide d'ASG et de l'ELB. Le code pour le site se trouve sur le repo ci-dessous. Les instances doivent pouvoir être crées par l'ASG dans les deux zones de disponibilités spécifiées. La politique de suivi de cible de l'ASG doit être liée au CPU pour une capacité d'utilisation moyenne de 70%.

NB: Lien utile https://github.com/daviddias/static-webpage-example.git

Rt1 : local + IGW

Public Subnet 1b
Public Subnet 1a

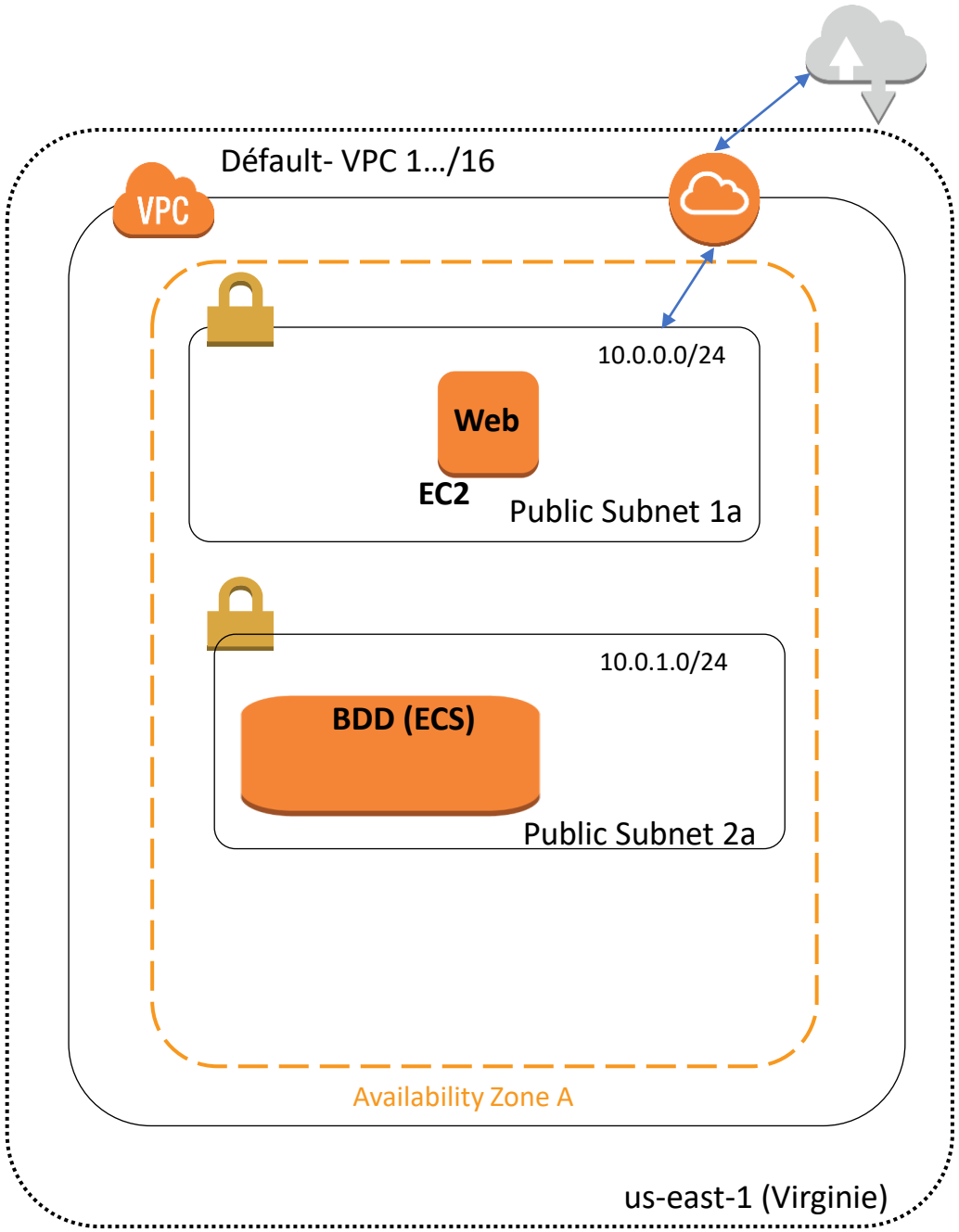
Rt2 : local + NAT

Private Subnet 1a

Rt3 : local

Private Subnet 1b

ECS



Enoncé TP12:

1) Créer un VPC avec des sous-réseaux respectant l'architecture ci-contre.

2) Déployer à l'aide d'un du service ECS la base de données mysql pour notre application Wordpress (votre container ECS devra être crée dans le cluster existant crée à cet effet.

1) Déployer le container wordpress à l'aide d'un script Bash dans l'EC2 Web qui sera dans le réseau public et consommable par nos clients à travers le port 8082; le container wordpress devra utiliser la base de données Mysql crée à partir de l'ECS

NB: Vous servir de la documentation sur le déploiement de wordpress à l'aide de container : https://hub.docker.com/_/wordpress

SCM / CODECOMMIT

Enoncé TP13 : CodeCommit

Le but de ce TP est de cloner le repo suivant : https://github.com/sadofrazer/youtube-lab-example.git et de le pusher dans un nouveau repo sur le cloud AWS que vous aurez créer au préalable.

1) Créer votre nouveau repo dans AWS à l'aide de CodeCommit.

2) Créer dans IAM vos identifiants de connexion à votre repo CodeCommit qui pour permettront de réaliser les push et les pull.

3) Pusher le repo git ci-dessus préalablement cloné sur votre machine locale vers votre repo git

4) Aller sur CodeCommit et se rassurer que votre repo contient les nouveaux fichiers pushés.

CODEBUILD

Enoncé TP14 : CodeBuild

Ce TP doit être réalisé à la suite du TP1,, l'objectif ici étant de récupérer le code se trouvant dans votre repo CodeCommit précédemment créé afin de le builder, le tester et si Ok pusher l'image buildée vers votre docker Hub.

1) Créer votre fichier buildspec.yaml qui contiendra les étapes et les actions à réaliser dans votre Build. Ce fichier devra être stocké à la racine de votre repo CodeCommit

2) Créer votre projet CodeBuild avec les paramètres et variables d'environnement qui vont bien afin que votre Buildspec soit correctement exécuté.

3) Lancer manuellement votre Buildspec et vérifiez que toutes les taches s'exécutent avec succès, vérifier également le contenu de votre Docker Hub si l'image testée et crée a bien été pushée à travers CodeBuild

4) Créer un projet CodePipeline pour automatiser ces taches de telles sorte qu'à chaque fois qu'une modification est faite dans votre repo CodeCommit, alors le CodeBuild respectif se lance et réalise vos différentes taches.