



# Déploiement de VM Automatisées


24.10.2023

---

Victor Marechal  
Prodigy Software



Révision	État	Date	Rédacteurs
0.1	En cours ▾	24.10.2023	Victor Marechal
	Pas com... ▾		



<b>Contexte du projet</b>	<b>4</b>
<b>Modalités pédagogiques</b>	<b>4</b>
<b>Contexte de l'exercice</b>	<b>4</b>
<b>Qu'est qu'un template?</b>	<b>5</b>
<b>Principales étapes pour réaliser un template sur Azure</b>	<b>5</b>
<b>Qu'est ce qu'un déploiement?</b>	<b>7</b>
<b>Principales étapes pour réaliser un déploiement sur Azure</b>	<b>8</b>
<b>Qu'est ce qu'une routine?</b>	<b>10</b>
<b>Principales étapes pour réaliser une routine de mise à jour</b>	<b>10</b>



## Contexte du projet

Défi : Prodigy Software connaît une croissance rapide et déploie ses solutions logicielles dans de plus en plus d'environnements clients. Pour répondre à cette demande croissante, l'entreprise souhaite automatiser le déploiement de ses solutions sur des machines virtuelles dans le cloud.

Rôle des Administrateurs Cloud :

Les administrateurs Cloud de Prodigy Software ont pour mission de concevoir et de gérer des templates de machines virtuelles optimisés pour le déploiement de leurs logiciels. Ils doivent également mettre en place un processus de déploiement automatisé de ces ressources.

Responsabilités clés :

Création de Templates de Machines Virtuelles : Les administrateurs Cloud doivent collaborer avec les équipes de développement pour créer des templates de machines virtuelles qui répondent aux besoins spécifiques de chaque produit de Prodigy Software. Ces templates doivent être optimisés en termes de performances et de sécurité.

Automatisation du Déploiement : Ils doivent mettre en place des pipelines d'automatisation du déploiement en utilisant des outils tels que Terraform, Ansible ou d'autres solutions similaires. L'objectif est de permettre un déploiement rapide et reproductible des machines virtuelles.

## Modalités pédagogiques

Brief "ECF" (évaluation individuelle)

- Création de templates pour serveur Jenkins et Gitlab. (Avec Packer ou ARM).
- Création de déploiement automatique. (Terraform ou Ansible).

Création d'une routine de mise à jour des templates.

## Contexte de l'exercice

Une souscription Azure, un resource group, un vnet, un subnet, et un nsg ont été précédemment créés. Tout le code du rendu est disponible sur [github](https://github.com). La documentation a été faite à l'aide de terraform-docs et chatGPT. Afin de ne pas me répéter, j'ai choisi de



présenter la documentation orientée sur Gitlab, la configuration de Jenkins étant semblable. Le code pour Jenkins est néanmoins présent sur le repo [github](#).

## Qu'est qu'un template?

Un template est un modèle prédéfini de configuration utilisé pour automatiser et standardiser le déploiement et la gestion d'infrastructures et d'applications dans un environnement cloud. Ces modèles sont généralement utilisés pour créer des configurations reproductibles, évolutives et cohérentes, tout en facilitant l'automatisation des tâches liées au déploiement et à la gestion des services cloud.

Principaux avantages des templates :

- Automatisation du déploiement
- Consistance et reproductibilité
- Évolutivité et flexibilité
- Gestion efficace des configurations
- Réduction des coûts et du temps de déploiement
- Documentation et traçabilité améliorées

Les fournisseurs de services cloud comme Azure, AWS et Google Cloud proposent des services de déploiement basés sur des templates, ce qui permet aux utilisateurs de créer, de gérer et de déployer facilement des ressources cloud à l'aide de modèles prédéfinis.

## Principales étapes pour réaliser un template sur Azure

Ici, j'ai choisi d'utiliser packer pour réaliser le template d'un serveur Gitlab compatible azure.

En premier lieu nous avons besoin de créer une Azure Compute Gallery pour y stocker nos images. On peut la déployer avec la commande Az cli suivante:

```
az sig create --resource-group myGalleryRG --gallery-name myGallery
```

Ensuite on doit créer une Image definition avec la commande:

```
az sig image-definition create \
```

```
--resource-group myGalleryRG \ --gallery-name myGallery \
--gallery-image-definition myImageDefinition \
--publisher myPublisher \
--offer myOffer \
--sku mySKU \
--os-type Linux \
--os-state specialized
```






Nous pouvons désormais passer aux templates Packer. Pour l'exercice j'ai créé un fichier [gitlab.pkr.hcl](#) qui contient la définition du template, et un fichier [variables.auto.pkr.hcl](#) (note: le .auto indique a packer d'utiliser ce fichier de variables sans devoir le spécifier dans la commande) qui contient les définitions de variables et certaines valeurs par défaut. Toutes les variables ne sont pas assignées afin de garantir la modularité et la sécurité du code lorsque nous voudront le versionner. Toutes les ressources sont taggées afin de faciliter leur gestion ultérieure. Le nom de l'image que l'on veut créer doit correspondre à la définition d'image créée précédemment. Pour build le template et le stocker dans notre gallery on utilisera la commande qui suit:

```
packer build -var="subscription_id=SUBSCRIPTION" \
-var="resource_group=RESOURCE_GROUP" \
-var="gallery_name=GALLERY" \
-var="root_password=PASSWORD" WORKDIR
```

On peut vérifier la création de nos ressources dans Azure Portal en recherchant les ressources taggées avec [brief : deployment-de-vm-automatisees]

## Resources with tag brief : deployment-de-vm-automatisees


Simplonformations.co

 Manage view  Refresh  Export to CSV  Open query |  Assign tags

Filter for any field...




Subscription equals all

Type equals all

 Add filter

 0 Recommendations

 0 Unsecure resources

<input type="checkbox"/> Name ↑↓	Type ↑↓
<input type="checkbox"/>  1.0.0 (t0t0r_acg_001/gitlab/1.0.0)	VM image version
<input type="checkbox"/>  gitlab (t0t0r_acg_001/gitlab)	VM image definition
<input type="checkbox"/>  gitlab	Image

## Qu'est ce qu'un déploiement?

Le déploiement avec Infrastructure as Code (IaC) implique l'utilisation de fichiers de configuration pour décrire et gérer l'infrastructure cloud, y compris les ressources matérielles et logicielles, de manière automatisée. Dans le contexte de déploiement sur Azure, cela se traduit souvent par l'utilisation d'outils tels que Terraform, Azure Resource Manager (ARM) Templates, ou des outils spécifiques à Azure comme Azure CLI ou Azure PowerShell.

Le déploiement avec Infrastructure as Code (IaC) présente de nombreux avantages significatifs pour les organisations et les équipes, notamment :

- Reproductibilité et cohérence
- Automatisation et efficacité
- Gestion simplifiée des changements
- Documentation et traçabilité améliorées
- Détection rapide des erreurs
- Environnements de test et de développement identiques

Dans l'ensemble, l'utilisation de l'Infrastructure as Code facilite la gestion et l'entretien des infrastructures, en réduisant les coûts opérationnels, en accélérant les déploiements et en améliorant la fiabilité globale de l'infrastructure.

## Principales étapes pour réaliser un déploiement sur Azure

Pour réaliser le déploiement j'ai choisi d'utiliser terraform. J'ai créé un module afin de garder le code DRY car les deux infras à déployer sont quasi identiques (une VM, une NIC et une IP publique).

La documentation du module est présente à la racine du module ou [ici](#).

nous pouvons maintenant appeler le module dans le main.tf du dossier terraform/gitlab et renseigner les variables nécessaires dans le fichier basic\_infra.auto.tfvars.

On valide la configuration:

```
terraform validate
```

On génère le tfplan avec la commande suivante:

```
terraform plan -out tfplan
```

Et on peut appliquer la configuration avec:





```
terraform apply tfplan
```

Une fois la configuration appliquée on peut une nouvelle fois vérifier la création de nos ressources dans Azure Portal en recherchant les ressources taggées avec [brief : deploiement-de-vm-automatisees]



## Resources with tag brief : deployment-de-vm-automatisees


Simploninformations.co

 Manage view  Refresh  Export to CSV  Open query |  Assign tags

Filter for any field...

Subscription equals all








Type equals all

 Add filter

 0 Recommendations

 0 Unsecure resources

Show recommendations in the advisor blade

<input type="checkbox"/> Name ↑↓	Type ↑↓
<input type="checkbox"/>  1.0.0 (t0t0r_acg_001/gitlab/1.0.0)	VM image version
<input type="checkbox"/>  gitlab (t0t0r_acg_001/gitlab)	VM image definition
<input type="checkbox"/>  gitlab	Image
<input type="checkbox"/>  gitlab-nic-001	Network Interface
<input type="checkbox"/>  gitlab-pubip-001	Public IP address
<input type="checkbox"/>  gitlab-vm-001	Virtual machine
<input type="checkbox"/>  gitlab-vm-001_disk1_db819ca0410c4b849582820a6657974b	Disk

Et on peut se rendre sur l'adresse IP de la VM afin de vérifier le bon fonctionnement du serveur Gitlab. Les credentials sont root / mot de passe renseigné lors du build de l'image.



### GitLab Community Edition

Nom d'utilisateur ou adresse de courriel principale

Mot de passe

[Mot de passe oublié ?](#)

☐ Se souvenir de moi

Connexion

Vous n'avez pas encore de compte ? [Inscrivez-vous maintenant](#)

## Qu'est ce qu'une routine?

Une routine est une séquence d'instructions ou d'actions spécifiques qui sont exécutées régulièrement ou de manière systématique pour accomplir une tâche particulière. Cette séquence d'actions est souvent planifiée et répétée dans un processus spécifique, visant à automatiser ou à standardiser certaines opérations pour garantir leur cohérence et leur fiabilité.

Dans ce contexte spécifique, la routine de mise à jour est une série d'étapes planifiées et automatisées exécutées à intervalles réguliers ou en réponse à certains événements spécifiques, garantissant ainsi que l'image est constamment mise à jour et prête à être déployée dans un environnement de production.

## Principales étapes pour réaliser une routine de mise à jour

Pour réaliser la routine j'ai fait le choix d'utiliser un workflow Github Actions.

Tout d'abord, nous avons besoin de créer un custom role Azure pour permettre à notre worker d'accéder et d'écrire dans notre Compute Gallery.

J'ai utiliser le fichier suivant:

```
{
  "properties": {
    "roleName": "Image Creation Role",
    "IsCustom": true,
    "description": "Azure Image Builder access to create resources for the image build",
    "assignableScopes": [
      "/subscriptions/{subscriptionID}/resourceGroups/{rgName}"
    ],
    "permissions": [
      {
        "actions": [
          "Microsoft.Compute/galleries/read",
          "Microsoft.Compute/galleries/images/read",
          "Microsoft.Compute/galleries/images/versions/read",
          "Microsoft.Compute/galleries/images/versions/write",
          "Microsoft.Compute/images/write",
```

```

        "Microsoft.Compute/images/read",
        "Microsoft.Compute/images/delete"
    ],
    "notActions": [],
    "dataActions": [],
    "notDataActions": []
  }
}
}
}

```

On upload ce fichier sur le Portal ou via az cli afin de créer notre custom role "Image Creation Role".

Ensuite nous avons besoin de créer un service principal et de récupérer ses credentials. On exécute la commande suivante:

```

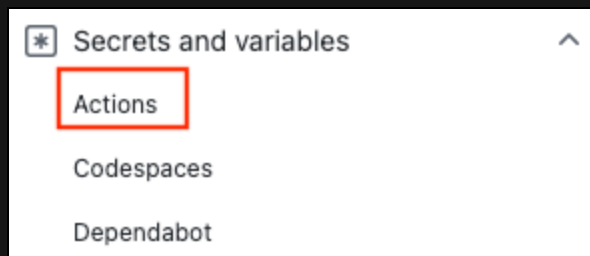
az ad sp create-for-rbac --name "SERVICE_NAME" --role contributor --scopes /subscriptions/<subscription-id>/resourceGroups/<group-name> --sdk-auth

```

et on récupère le json contenant les credentials en output de la commande.

Nous allons ensuite copier ces credentials dans un secret Github Actions du repository.

1. In [GitHub](#), go to your repository.
2. Go to Settings in the navigation menu.
3. Select Security > Secrets and variables > Actions.



4. Select New repository secret.
5. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name AZURE\_CREDENTIALS.
6. Select Add secret.

Nous pouvons maintenant passer au fichier de workflow. J'ai créé un workflow [gitlab\\_updater.yaml](#).

Voici les caractéristiques de ce workflow:

- Le workflow run grâce à un scheduler tous les dimanches à 0h30 (volontairement commenté dans le code pour économiser des ressources) mais peut aussi être démarré manuellement grâce à 'workflow\_dispatch'
- Il s'exécute sur une machine ubuntu:22.04 à l'instar de notre VM Gitlab
- Étapes:
  1. Checkout le repository et se place à la racine
  2. Log in dans Azure
  3. Install Packer
  4. Run 'packer init'
  5. Run 'packer validate'
  6. Récupère le nom de la dernière version de l'image et incrémente le numéro de patch
  7. Update les packages de l'image et la stocke dans la Compute Gallery avec le numéro de version crafté à l'étape précédente

On peut surveiller dans github l'état du run

Update Custom Gitlab VM Image #13

Summary

Jobs

- build-image

Run details

Usage

Workflow file

build-image  
succeeded now in 15m 0s

Search logs

Set up job 7s

Checkout 1s

Log in with Azure 17s

Setup `packer` 2s

Run `packer init` 0s

Run `packer validate` 0s

Find latest version 3s

```
1 ▶ Run version=$(az sig image-version list --gallery-name t0t0r_acg_001 \
19 Latest image version is 1.0.1
20 Nouvelle version : 1.0.2
```




Update existing image 14m 23s

```
1 ▶ Run packer build -force -var="subscription_id=*" -var="resource_group=t0t0r-rg-001" -var="gallery_name=t0t0r_acg_001" -
var="version=1.0.2" .
8 azure-arm.update_image: output will be in this color.
9
10 ==> azure-arm.update_image: Running builder ...
11   azure-arm.update_image: Creating Azure Resource Manager (ARM) client ...
12 ==> azure-arm.update_image: the managed image named gitlab already exists, but deleting it due to -force flag
13 ==> azure-arm.update_image: Getting source image id for the deployment ...
14 ==> azure-arm.update_image: Using existing resource group ...
15 ==> azure-arm.update_image: -> ResourceGroupName : 't0t0r-rg-001'
```

Une fois le run fini on peut une nouvelle fois vérifier la création de nos ressources dans Azure Portal en recherchant les ressources taggées avec [brief : deployment-de-vm-automatisees]


## Resources with tag brief : deploiement-de-vm-automatisees

Simploninformations.co

 Manage view  Refresh  Export to CSV  Open query |  Assign tags

Filter for any field...













Subscription equals **all**

Type equals **all** 

 Add filter

 **0** Recommendations

 **0** Unsecure resources

<input type="checkbox"/> Name 	Type 
<input type="checkbox"/>  1.0.0 (t0t0r_acg_001/gitlab/1.0.0)	VM image version
<input type="checkbox"/>  1.0.1 (t0t0r_acg_001/gitlab/1.0.1)	VM image version
<input type="checkbox"/>  1.0.2 (t0t0r_acg_001/gitlab/1.0.2)	VM image version
<input type="checkbox"/>  1.0.3 (t0t0r_acg_001/gitlab/1.0.3)	VM image version
<input type="checkbox"/>  gitlab (t0t0r_acg_001/gitlab)	VM image definition
<input type="checkbox"/>  gitlab	Image
<input type="checkbox"/>  gitlab-nic-001	Network Interface
<input type="checkbox"/>  gitlab-pubip-001	Public IP address
<input type="checkbox"/>  gitlab-vm-001	Virtual machine
<input type="checkbox"/>  gitlab-vm-001_disk1_db819ca0410c4b849582820a6657974b	Disk