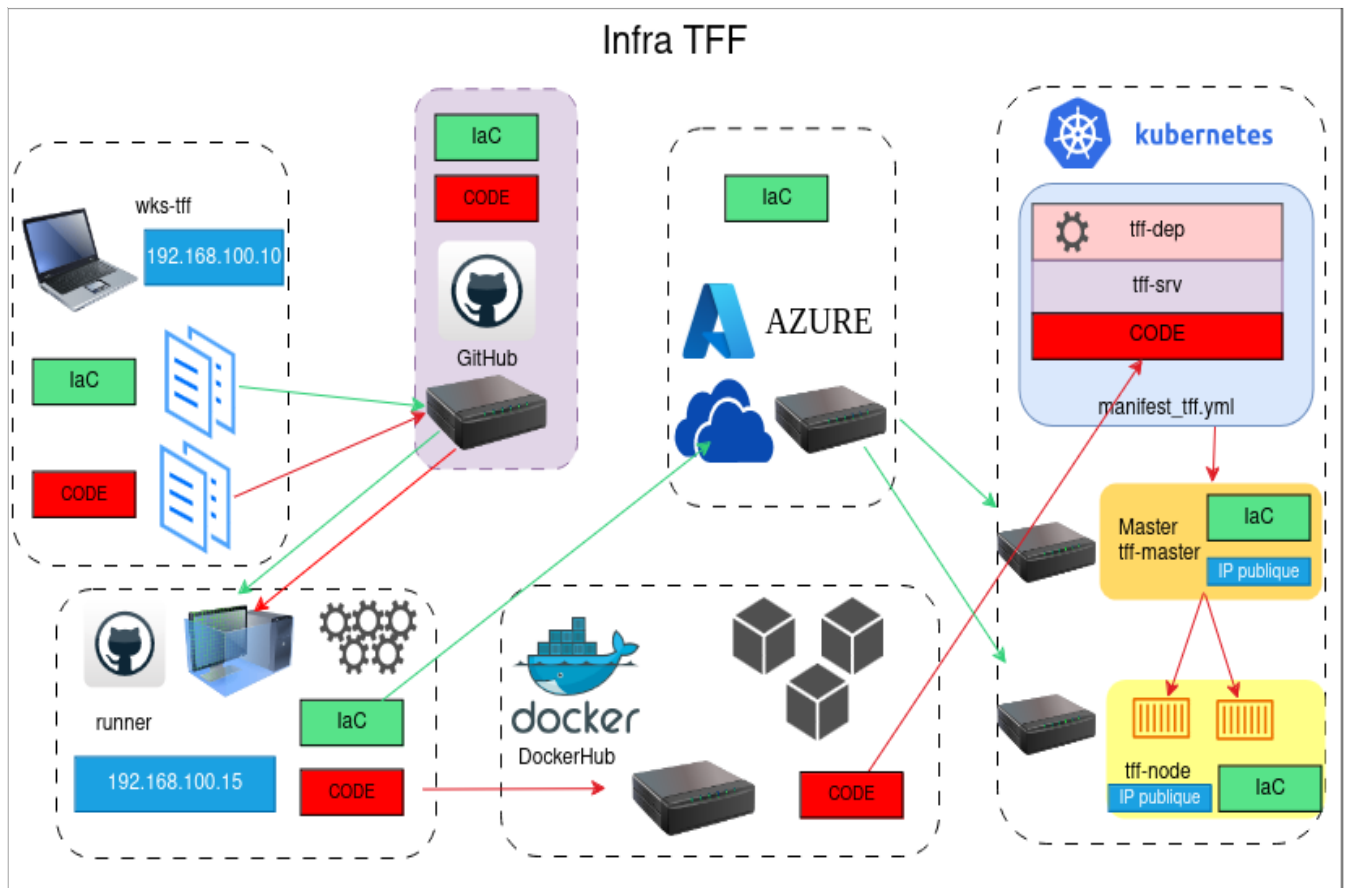




# Automatisation du Déploiement d'Infrastructure et d'Application Web sur Azure via un Pipeline CI/CD avec GitHub Actions, Terraform, Ansible, Docker, et Kubernetes.

Par Pierre Magain

Le 29 Août 2024



Voici le schéma complet de l'infrastructure et des déploiements s'y déroulant. Le but de ce document est de décrire les ressources, les scripts et les déploiements.

## 1. WKS-TFF

Type de ressource : VM

OS : Linux Mint

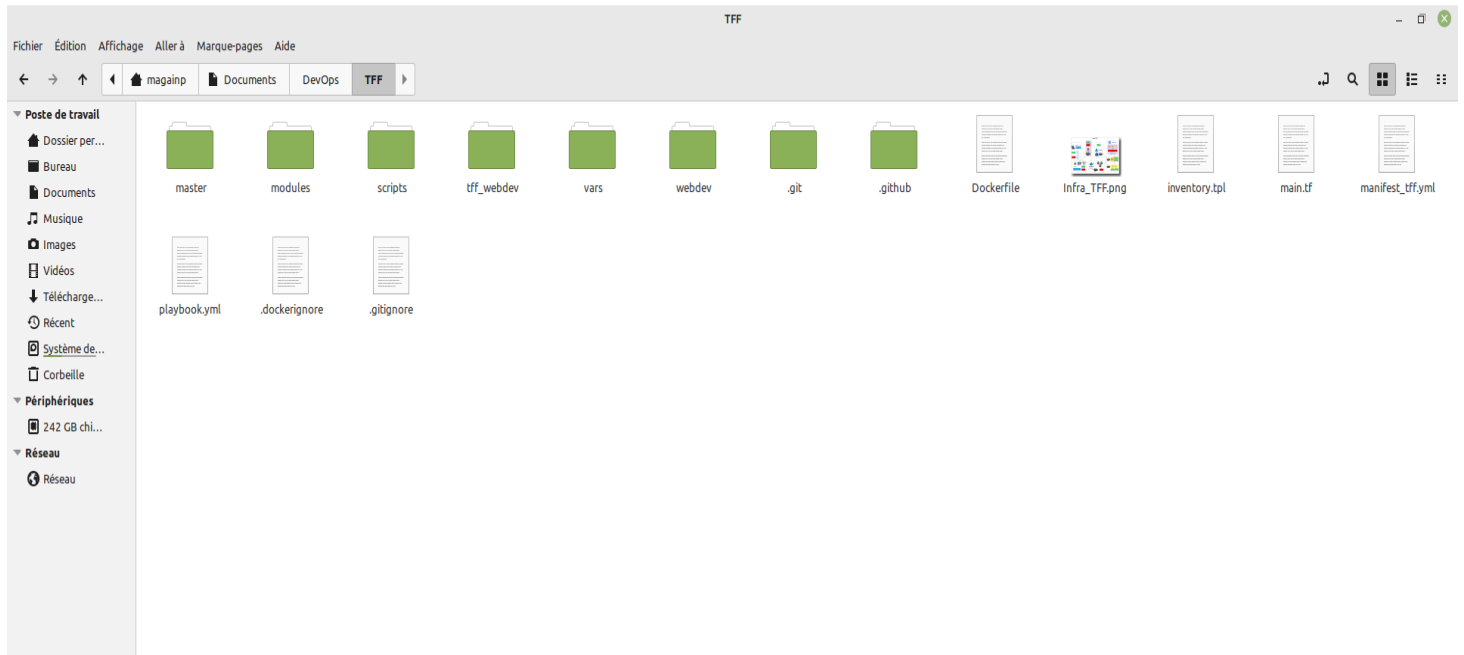
Processeur : 2

RAM : 4 Giga

Adresse IP : 192.168.100.10

La machine virtuelle wks-tff n'est autre que la VM contenant le projet à la base, c'est elle qui enverra le repository .git local comprenant tous les codes du projet sur le repository GitHub distant.

Voici d'ailleurs une image du dossier contenant le repository local :



On distingue plusieurs sous dossiers, notamment celui des modules terraform que l'on utilisera dans le pipeline du déploiement de l'infrastructure. On y distingue également le sous dossier tff\_webdev comprenant les fichiers de notre site web qui sera déployé. On y distingue bien évidemment le .git et très important le .github qui est le sous dossier comprenant les 3 workflows qui s'exécuteront une fois le git push sur le repository distant effectué. On peut y voir le manifest\_tff.yml qui comme son nom l'indique est le manifest que l'on exécutera sur le cluster microk8s une fois celui-ci installé, il y'a également le main.tf qui a l'instar du playbook.yml pour Ansible est le fichier racine c'est a dire que c'est le fichier qui appellera les modules Terraform pour configurer l'infrastructure sur le cloud Azure et comme mentionné le playbook.yml qui lui installera les rôles Ansible qui serviront à mettre en place notre cluster microk8s.

## **2. RUNNER (auto-hébergé)**

Type de ressource : VM

OS : Ubuntu

Processeur : 4

RAM : 8 Giga

Adresse IP : 192.168.100.15

## 2.1. Installation du Runner auto hébergé

General

Access

Collaborators

Code and automation

Branches

Tags

Rules

Actions

General

Runners

Webhooks

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets and variables

Integrations

GitHub Apps

Email notifications

### Runners / Add new self-hosted runner · PierreMagain/TFF

Adding a self-hosted runner requires that you download, configure, and execute the GitHub Actions Runner. By downloading and configuring the GitHub Actions Runner, you agree to the [GitHub Terms of Service](#) or [GitHub Corporate Terms of Service](#), as applicable.

Runner image

macOS

Linux

Windows

Architecture

x64

Download

```
# Create a folder
$ mkdir actions-runner && cd actions-runner
# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.319.1.tar.gz -L https://github.com/actions/runner/releases/download/v2.319.1/actions-runner-linux-x64-2.319.1.tar.gz
# Optional: Validate the hash
$ echo "3f6efb7488a183e291fc2c62876e14c9ee732864173734facc85a1bf1744464 actions-runner-linux-x64-2.319.1.tar.gz" | shasum -a 256 -c
# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.319.1.tar.gz
```

Configure

```
# Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/PierreMagain/TFF --token BIN3I7ESENASUS73PHFPRGDGZWM40
# Last step, run it!
$ ./run.sh
```

Using your self-hosted runner

```
# Use this YAML in your workflow file for each job
runs-on: self-hosted
```

For additional details about configuring, running, or shutting down the runner, please check out our [product docs](#).

## Runners

[New self-hosted runner](#)

Host your own runners and customize the environment used to run jobs in your GitHub Actions workflows. [Learn more about self-hosted runners.](#)

Runners	Status
<div>github-runner self-hosted Linux X64</div>	● Offline ...

Une fois bien installé, ne pas oublier les secrets utilisés par les workflows: dans ce projet, nous n'en utiliserons que 3.

General

Access

Collaborators

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Codespaces

Pages

Security

Code security and analysis

Deploy keys

Secrets and variables

Actions

Codespaces

Dependabot

Integrations

GitHub Apps

Email notifications

## Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for **non-sensitive** data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Secrets

Variables

Repository secrets

New repository secret

Name	Last updated
DOCKER_REGISTRY_PASSWORD	5 days ago
DOCKER_REGISTRY_USER	5 days ago
PAT	3 days ago

## 2.2. Installation de Docker, Terraform, Ansible et les roles Ansible

Étant donné que notre runner auto-hébergé va manipuler plusieurs technologies, il faut bien les installer en amont. Le titre de ce chapitre est assez complet, pour que tout marche , le Runner doit posséder docker, Terraform, Ansible et les rôles Ansible que le playbook appellera.

```

Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
magainp@runner:~$ ll
total 92
drwxr-x--- 13 magainp magainp 4096 Aug 26 09:00 ./
drwxr-xr-x  3 root    root    4096 Aug  1 08:46 ../
drwxr-xr-x  6 magainp magainp 4096 Aug 23 08:33 actions-runner/
drwxrwxr-x  4 magainp magainp 4096 Aug 22 14:56 .ansible/
drwxrwxr-x  3 magainp magainp 4096 Aug 22 13:49 Ansible/
drwxrwxr-x  5 magainp magainp 4096 Aug 26 14:54 .azure/
-rw-----  1 magainp magainp 16071 Aug 26 15:45 .bash_history
-rw-r--r--  1 magainp magainp  220 Mar 31 08:41 .bash_logout
-rw-r--r--  1 magainp magainp 3919 Aug 22 14:00 .bashrc
drwx-----  2 magainp magainp 4096 Aug  1 08:47 .cache/
-rwxr-xr-x  1 magainp magainp  529 Aug 26 09:00 create_backend.sh*
drwxrwxr-x  3 magainp magainp 4096 Aug 26 15:08 .docker/
drwx-----  2 magainp magainp 4096 Aug 22 13:56 .gnupg/
-rw-----  1 magainp magainp  20 Aug  1 08:49 .lessht
drwxrwxr-x  3 magainp magainp 4096 Aug  1 08:47 .local/
drwxrwxr-x  2 magainp magainp 4096 Aug 26 09:11 master/
-rw-r--r--  1 magainp magainp  807 Mar 31 08:41 .profile
drwx-----  2 magainp magainp 4096 Aug 26 14:57 .ssh/
-rw-r--r--  1 magainp magainp   0 Aug  1 08:49 .sudo_as_admin_successful
drwxr-xr-x  2 magainp magainp 4096 Aug 24 16:05 .terraform.d/
-rw-rw-r--  1 magainp magainp  181 Aug 22 13:56 .wget-hsts
magainp@runner:~$

```

Attention, il ne faut pas oublier d'éditer le chemin où Ansible trouvera ses rôles:

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
magainp@runner:~$ cat /etc/ansible/ansible.cfg
[defaults]
roles_path = /home/magainp/Ansible/roles
magainp@runner:~$ ll Ansible/roles/
total 32
drwxrwxr-x 7 magainp magainp 4096 Aug 22 14:19 ./
drwxrwxr-x 3 magainp magainp 4096 Aug 22 13:49 ../
drwxrwxr-x 4 magainp magainp 4096 Aug 22 13:49 microk8s_generate_token/
drwxrwxr-x 5 magainp magainp 4096 Aug 22 13:49 microk8s_install/
drwxrwxr-x 4 magainp magainp 4096 Aug 22 13:49 microk8s_join_node/
drwxrwxr-x 8 magainp magainp 4096 Aug 22 13:49 os_common/
-rw-rw-r-- 1 magainp magainp 160 Aug 22 13:49 README.md
drwxrwxr-x 8 magainp magainp 4096 Aug 22 13:49 skeleton/
magainp@runner:~$
```

### 2.3. Script `create_backend.sh`

Étant donné que nous utiliserons un runner auto-hébergé, l'exécution des tâches du pipeline se feront dans des dossiers temporaires et de ce fait, terraform ne saura pas sauvegarder l'infrastructure qu'il vient de réaliser comme normalement il le ferait sur une machine normale. Ce problème vient du fait que dans ce type de process, terraform ne sauvegarde rien dans son '*terraform.tfstate*' (qui est son backend), il faut donc en amont du pipeline créer un backend à distance. Voici un script qui crée ce backend sur Azure :

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
magainp@runner:~$ cat create_backend.sh
#!/bin/bash

# Variables
RESOURCE_GROUP="my-terraform-state"
LOCATION="westeurope"
STORAGE_ACCOUNT="backendentfstatetff"
CONTAINER_NAME="tfstatestorage"
SKU="Standard_LRS"

# Créer un groupe de ressources
az group create --name $RESOURCE_GROUP --location $LOCATION

# Créer un compte de stockage
az storage account create --name $STORAGE_ACCOUNT --resource-group $RESOURCE_GROUP --location $LOCATION --sku $SKU

# Créer un conteneur de stockage
az storage container create --name $CONTAINER_NAME --account-name $STORAGE_ACCOUNT
magainp@runner:~$
```

Et bien évidemment l'exécuter !

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
magainp@runner:~$ ./create_backend.sh
{
  "id": "/subscriptions/d59a343a-4f98-492f-974a-7aab5defa573/resourceGroups/my-terraform-state",
  "location": "westeurope",
  "managedBy": null,
  "name": "my-terraform-state",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
{
  "accessTier": "Hot",
  "accountMigrationInProgress": null,
  "allowBlobPublicAccess": false,
  "allowCrossTenantReplication": false,
  "allowSharedKeyAccess": null,
  "allowedCopyScope": null,
  "azureFilesIdentityBasedAuthentication": null,
  "blobRestoreStatus": null,
  "creationTime": "2024-08-27T08:53:52.546729+00:00",
  "customDomain": null,
  "defaultToOAuthAuthentication": null,
  "dnsEndpointType": null,
  "enableExtendedGroups": null,
  "enableHttpsTrafficOnly": true,
  "enableNfsV3": null,
  "encryption": {
    "encryptionIdentity": null,
    "keySource": "Microsoft.Storage",
    "keyVaultProperties": null,
    "requireInfrastructureEncryption": null,
  },
  "services": {

```

On peut ensuite bien voir le backend à distance sur le portail Azure :

Accueil > Toutes les ressources

Répertoire par défaut (magainppngmail.onmicrosoft.com)

+ Créer Gérer la vue Actualiser Exporter au format CSV Ouvrir une requête Attribuer des étiquettes Supprimer

Filter un champ... Abonnement égal à tout Groupe de ressources égal à tout Type égal à tout Emplacement égal à tout Ajouter un filtre

0 Ressources non sécurisées 0 Recommandations 1 Changed resources

Aucun regroupement Vue liste

Nom ↑	Type ↑	Groupe de ressources ↑	Emplacement ↑	Abonnement ↑
backendstateff	Compte de stockage	my-terraform-state	West Europe	Azure subscription 1

Une fois tout en ordre, il ne faut plus que mettre le runner en ligne pour qu'il reçoive les commandes des pipelines :

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
magainp@runner:~$ cd actions-runner/
magainp@runner:actions-runner$ ./run.sh
✓ Connected to GitHub
Current runner version: '2.319.1'
2024-08-27 09:16:08Z: Listening for Jobs
2024-08-27 09:16:12Z: Running job: terraform
```

## 3. Les Pipelines CI/CD via GitHub Actions

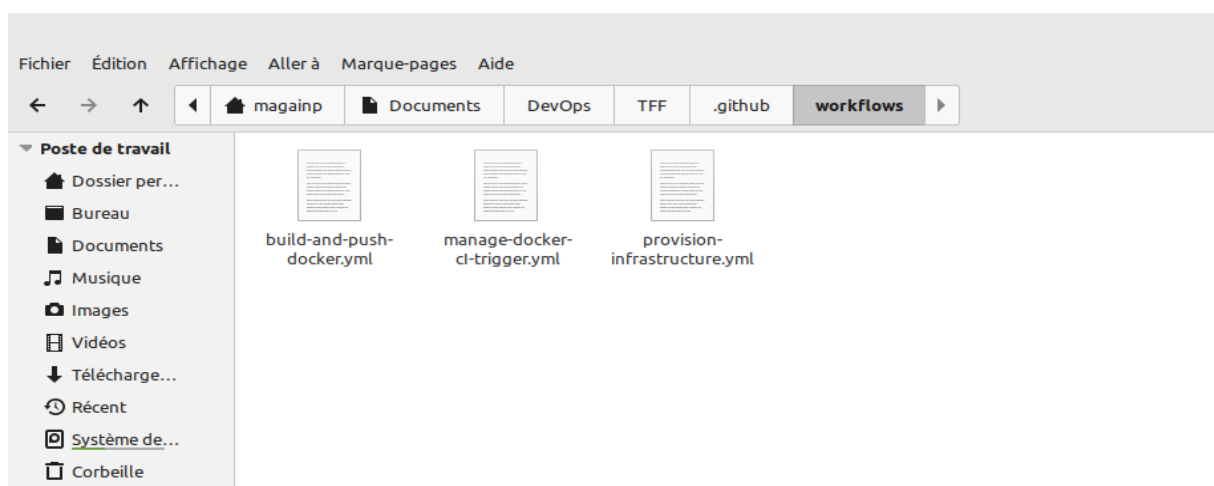
### 3.1. Lancement

Une fois tout ceci fait, un triplet de git sur wks-tff :

- git add .
- git commit -m «ce que vous voulez»
- git push

```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
magainp@wks-tff:~$ git add .
magainp@wks-tff:~$ git commit -m "Demo"
[master d8e882d] Demo
32 files changed, 612 insertions(+), 3186 deletions(-)
rewrite tff/webdev/contact.html (74%)
rewrite tff/webdev/css/contact.css (83%)
rewrite tff/webdev/css/global.css (64%)
delete mode 100644 tff/webdev/css/sandbox-js.css
rewrite tff/webdev/css/sandbox.css (78%)
delete mode 100644 tff/webdev/documents/moncv.pdf
rewrite tff/webdev/form-processing.html (60%)
delete mode 100644 tff/webdev/img/slider1.jpg
delete mode 100644 tff/webdev/img/slider2.jpg
delete mode 100644 tff/webdev/img/slider3.jpg
rewrite tff/webdev/index.html (84%)
delete mode 100644 tff/webdev/js/index.js
delete mode 100644 tff/webdev/js/main.js
delete mode 100644 tff/webdev/js/sandbox.js
delete mode 100644 tff/webdev/sandbox-js.html
rewrite tff/webdev/sandbox.html (91%)
rewrite webdev/webdev/contact.html (74%)
rewrite webdev/webdev/css/contact.css (83%)
rewrite webdev/webdev/css/global.css (64%)
delete mode 100644 webdev/webdev/css/sandbox-js.css
rewrite webdev/webdev/css/sandbox.css (78%)
delete mode 100644 webdev/webdev/documents/moncv.pdf
rewrite webdev/webdev/form-processing.html (60%)
delete mode 100644 webdev/webdev/img/slider1.jpg
delete mode 100644 webdev/webdev/img/slider2.jpg
delete mode 100644 webdev/webdev/img/slider3.jpg
rewrite webdev/webdev/index.html (84%)
delete mode 100644 webdev/webdev/js/index.js
delete mode 100644 webdev/webdev/js/main.js
delete mode 100644 webdev/webdev/js/sandbox.js
delete mode 100644 webdev/webdev/sandbox-js.html
rewrite webdev/webdev/sandbox.html (91%)
magainp@wks-tff:~$ git push
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 2 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 1.85 KiB | 946.00 KiB/s, done.
Total 14 (delta 10), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (10/10), completed with 8 local objects.
To github.com:~/.github: PierreMagain/TFF.git
   fc9e116..d8e882d master -> master
magainp@wks-tff:~$
```

C'est là que entrent en jeu les 3 workflows, ils ont un ordre déterminé par des dépendances au début de chacun d'eux et ils sont situés dans le dossier .github :





### 3.2. Premier Workflow : Provision Infrastructure

C'est le workflow le plus simple à comprendre : dès qu'il y a un push sur la branche master , il exécute le main.tf du repository.

Voici le code du workflow provision-infrastructure.yml :

```
! provision-infrastructure.yml x
.github > workflows > ! provision-infrastructure.yml
1  name: Provision Infrastructure
2
3  on:
4    push:
5      branches:
6        - master
7
8  jobs:
9    terraform:
10     runs-on: self-hosted
11
12     steps:
13       - name: Checkout code
14         uses: actions/checkout@v2
15
16       - name: Terraform get
17         run: terraform get
18
19       - name: Terraform Init
20         run: terraform init
21
22       - name: Terraform Plan
23         run: terraform plan
24
25       - name: Terraform Apply
26         run: terraform apply -auto-approve -lock=false
27
```

Et voici le main.tf exécuté :

main.tf - TFF - Visual Studio Code

```
1 provider "azurerm" {
2   features {}
3
4   subscription_id = "d59a343a-4f98-492f-974a-7aab5defa573"
5 }
6
7 terraform {
8   backend "azurerm" {
9     resource_group_name = "my-terraform-state"
10    storage_account_name = "backndtsttsttff"
11    container_name       = "tftstteststorage"
12    key                  = "terraform.tfstate"
13  }
14 }
15
16 module "resource_group" {
17   source          = "../modules/azure_resource_group_name"
18   resource_group_name = "ClusterK8S"
19   location         = "Germany West Central"
20 }
21
22 module "network" {
23   source          = "../modules/azure_resource_network"
24   resource_group_name = module.resource_group.resource_group_name
25   location         = module.resource_group.resource_group_location
26   address_space    = ["192.168.0.0/16"]
27   subnet_prefix    = ["192.168.0.0/24"]
28 }
29
30 module "vm1" {
31   source          = "../modules/azure_resource_vm_ubuntu"
32   resource_group_name = module.resource_group.resource_group_name
33   location         = module.resource_group.resource_group_location
34   vnet_name        = module.network.vnet_name
35   subnet_name       = module.network.subnet_name
36   subnet_id         = module.network.subnet_id
37   vm_name           = "tff-master"
38   vm_size           = "Standard_B2s"
39 }
40
41 module "vm2" {
42   source          = "../modules/azure_resource_vm_ubuntu"
43   resource_group_name = module.resource_group.resource_group_name
44   location         = module.resource_group.resource_group_location
45   vnet_name        = module.network.vnet_name
46   subnet_name       = module.network.subnet_name
47   subnet_id         = module.network.subnet_id
48   vm_name           = "tff-node"
```

Ln 90, Col 16 Spaces: 2 UTF-8 LF Terraform

main.tf - TFF - Visual Studio Code

```
41 module "vm2" {
42   subnet_id        = module.network.subnet_id
43   vm_name           = "tff-node"
44   vm_size           = "Standard_B2s"
45 }
46
47 resource "null_resource" "update_known_hosts" {
48
49   depends_on = [
50     module.vm1,
51     module.vm2
52   ]
53
54   provisioner "local-exec" {
55     command = <<EOT
56     sleep 30
57     export VM1_IP=$(module.vm1.public_ip_address)
58     export VM1_NAME=$(module.vm1.vm_name)
59     export VM2_IP=$(module.vm2.public_ip_address)
60     $(path.module)/scripts/update_hosts.sh
61     $(path.module)/scripts/add_known_hosts.sh
62     EOT
63   }
64 }
65
66 resource "local_file" "ansible_inventory" {
67   depends_on = [null_resource.update_known_hosts]
68
69   content = templatefile("${path.module}/inventory.tpl", {
70     vm1_name = module.vm1.vm_name,
71     vm1_ip   = module.vm1.public_ip_address,
72     vm1_user = module.vm1.admin_username,
73     vm2_name = module.vm2.vm_name,
74     vm2_ip   = module.vm2.public_ip_address,
75     vm2_user = module.vm2.admin_username
76   })
77   filename = "${path.module}/inventory.ini"
78 }
79
80 resource "null_resource" "ansible_provision" {
81   depends_on = [local_file.ansible_inventory]
82
83   provisioner "local-exec" {
84     command = "ansible-playbook -i $(path.module)/inventory.ini $(path.module)/playbook.yml"
85   }
86 }
87
88
```

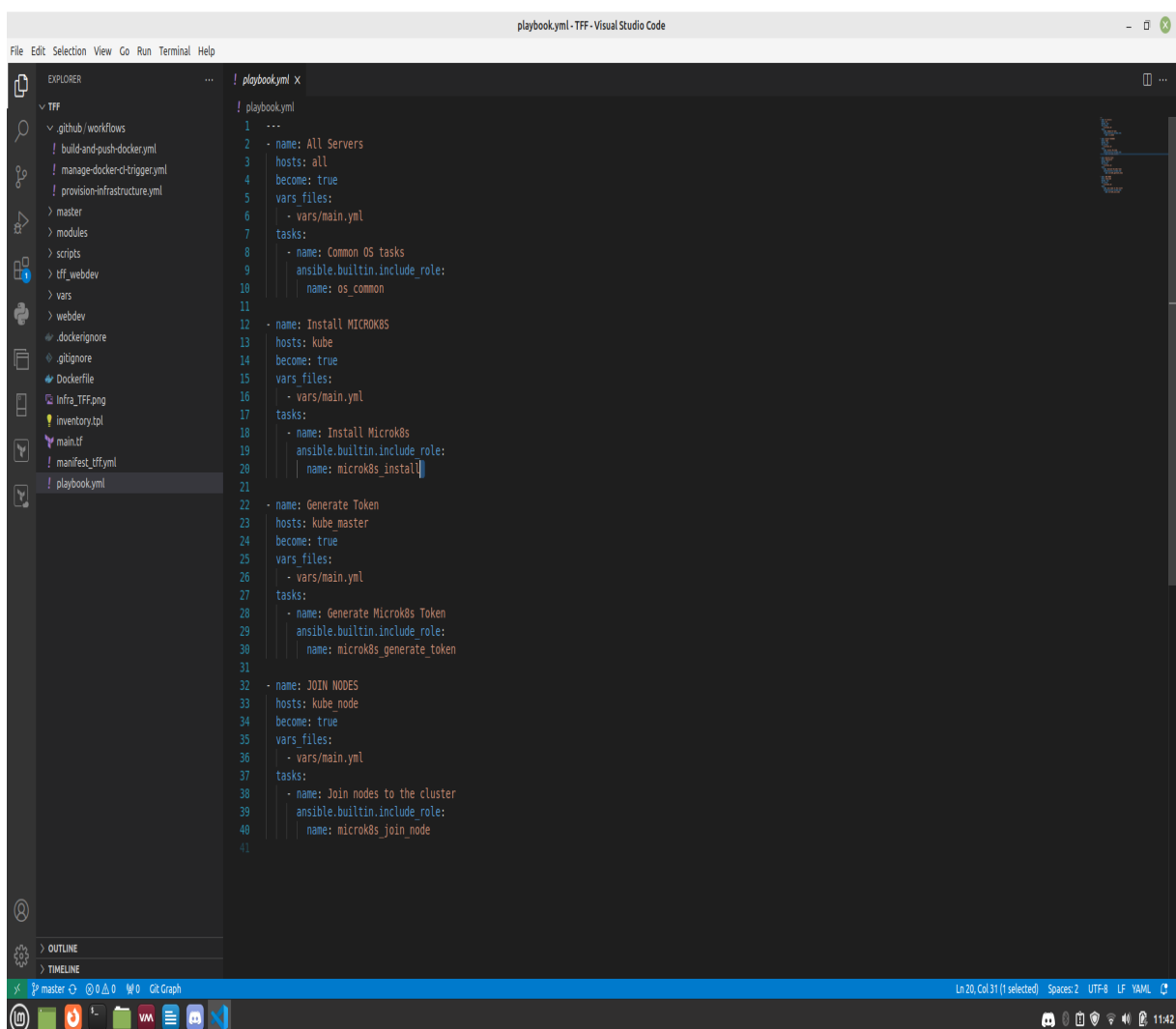
Ln 90, Col 16 Spaces: 2 UTF-8 LF Terraform

Sans rentrer dans les détails , voici ce que fait le main.tf appliquer par Terraform :

- il crée un groupe de ressources
- crée un réseau virtuel et un sous-réseau associé
- crée deux VM ubuntu dans le groupe de ressource et le réseaux/sous-réseaux créé
- valide les fingerprint des connexions ssh et ajoute dans le '/etc/hosts' du runner l'adresse Ip publique de la VM master
- crée le fichier inventory.ini grâce a un fichier template inventory.tpl en remplaçant les endroits des variables par les valeurs des VM créés
- applique les rôle ansible en executant la commande :

```
'ansible-playbook -i ${path.module}/inventory.ini ${path.module}/playbook.yml'
```

Le playbook.yml qui lui installe les os\_common et un cluster microk8s :



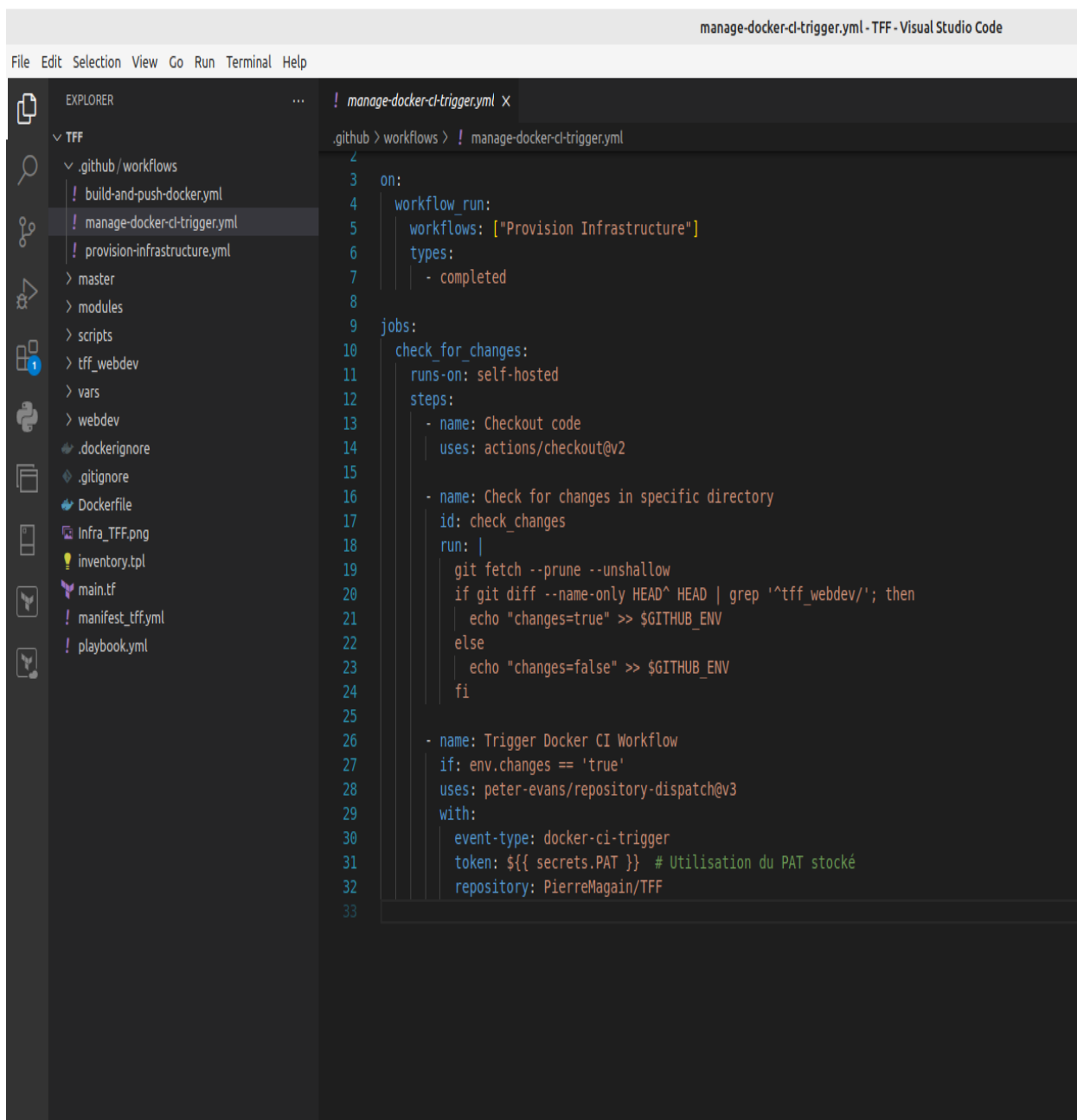
```
1 ---
2 - name: All Servers
3   hosts: all
4   become: true
5   vars_files:
6     - vars/main.yml
7   tasks:
8     - name: Common OS tasks
9       ansible.builtin.include_role:
10         name: os_common
11
12 - name: Install MICROK8S
13   hosts: kube
14   become: true
15   vars_files:
16     - vars/main.yml
17   tasks:
18     - name: Install Microk8s
19       ansible.builtin.include_role:
20         name: microk8s_install
21
22 - name: Generate Token
23   hosts: kube_master
24   become: true
25   vars_files:
26     - vars/main.yml
27   tasks:
28     - name: Generate Microk8s Token
29       ansible.builtin.include_role:
30         name: microk8s_generate_token
31
32 - name: JOIN NODES
33   hosts: kube_node
34   become: true
35   vars_files:
36     - vars/main.yml
37   tasks:
38     - name: Join nodes to the cluster
39       ansible.builtin.include_role:
40         name: microk8s_join_node
41
```

### 3.3. Deuxième Workflow : Manage Docker CI Trigger

Ce workflow est assez particulier car il n'exécute rien en tant que tel, il s'assure que le premier workflow Provision Infrastructure s'est bien terminé et il regarde s'il y a eu un changement dans le sous-dossier tff\_webdev.

Uniquement si ces 2 conditions sont réunies, il déclenche le troisième workflow : Docker CI

voici le workflow manage-docker-ci-trigger.yml :



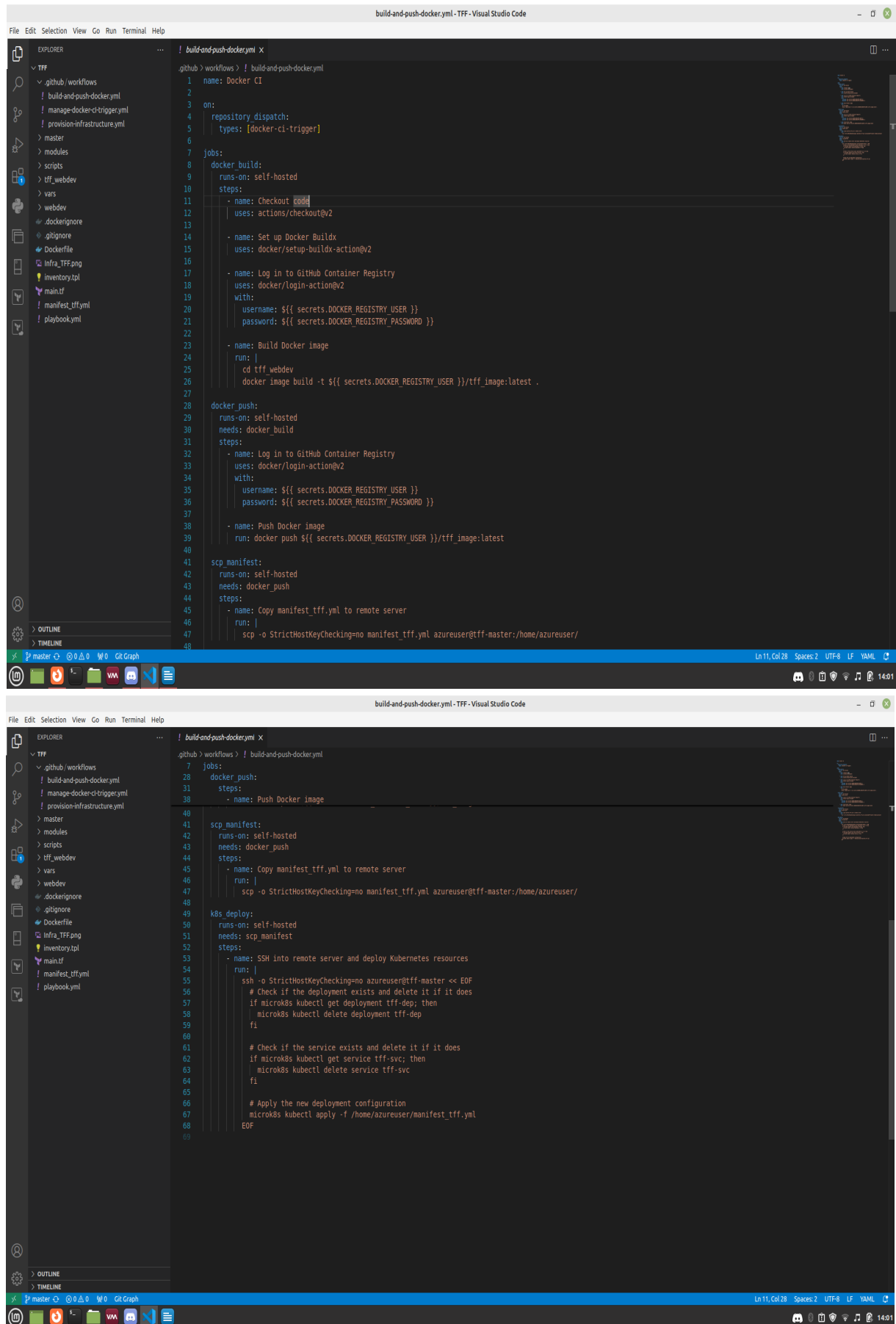
The screenshot shows the Visual Studio Code interface with the file `manage-docker-ci-trigger.yml` open. The Explorer sidebar on the left shows the project structure under `TFF`, including `.github/workflows` and `tff_webdev`. The main editor displays the YAML content of the workflow file.

```
1  on:
2
3  workflow_run:
4    workflows: ["Provision Infrastructure"]
5    types:
6      - completed
7
8  jobs:
9    check_for_changes:
10     runs-on: self-hosted
11     steps:
12       - name: Checkout code
13         uses: actions/checkout@v2
14
15       - name: Check for changes in specific directory
16         id: check_changes
17         run: |
18           git fetch --prune --unshallow
19           if git diff --name-only HEAD^ HEAD | grep '^tff_webdev/'; then
20             echo "changes=true" >> $GITHUB_ENV
21           else
22             echo "changes=false" >> $GITHUB_ENV
23           fi
24
25       - name: Trigger Docker CI Workflow
26         if: env.changes == 'true'
27         uses: peter-evans/repository-dispatch@v3
28         with:
29           event-type: docker-ci-trigger
30           token: ${ secrets.PAT } # Utilisation du PAT stocké
31           repository: PierreMagain/TFF
32
33
```

### 3.4. Troisième Workflow : Docker CI

C'est le workflow qui déploie le site web, il exécute 4 tâches.

Tout d'abord voici le code du workflow build-and-push-docker.yml :



```
1 name: Docker CI
2
3 on:
4   repository dispatch:
5     types: [docker-ci-trigger]
6
7 jobs:
8   docker_build:
9     runs-on: self-hosted
10    steps:
11      - name: Checkout code
12        uses: actions/checkout@v2
13
14      - name: Set up Docker Buildx
15        uses: docker/setup-buildx-action@v2
16
17      - name: Log in to GitHub Container Registry
18        uses: docker/login-action@v2
19        with:
20          username: ${ secrets.DOCKER_REGISTRY_USER }
21          password: ${ secrets.DOCKER_REGISTRY_PASSWORD }
22
23      - name: Build Docker image
24        run: |
25          cd tff_webdev
26          docker image build -t ${ secrets.DOCKER_REGISTRY_USER }/tff_image:latest .
27
28   docker_push:
29     runs-on: self-hosted
30     needs: docker_build
31     steps:
32       - name: Log in to GitHub Container Registry
33         uses: docker/login-action@v2
34         with:
35           username: ${ secrets.DOCKER_REGISTRY_USER }
36           password: ${ secrets.DOCKER_REGISTRY_PASSWORD }
37
38       - name: Push Docker image
39         run: docker push ${ secrets.DOCKER_REGISTRY_USER }/tff_image:latest
40
41   scp_manifest:
42     runs-on: self-hosted
43     needs: docker_push
44     steps:
45       - name: Copy manifest_tff.yml to remote server
46         run: |
47           scp -o StrictHostKeyChecking=no manifest_tff.yml azureuser@tff-master:/home/azureuser/
48
49   k8s_deploy:
50     runs-on: self-hosted
51     needs: scp_manifest
52     steps:
53       - name: SSH into remote server and deploy Kubernetes resources
54         run: |
55           ssh -o StrictHostKeyChecking=no azureuser@tff-master << EOF
56           # Check if the deployment exists and delete it if it does
57           if microk8s kubectl get deployment tff-dep; then
58             microk8s kubectl delete deployment tff-dep
59           fi
60
61           # Check if the service exists and delete it if it does
62           if microk8s kubectl get service tff-svc; then
63             microk8s kubectl delete service tff-svc
64           fi
65
66           # Apply the new deployment configuration
67           microk8s kubectl apply -f /home/azureuser/manifest_tff.yml
68           EOF
```

Comme on peut le voir, ce workflow comprends 4 étapes :

The screenshot shows a GitHub Actions workflow run for the repository 'PierreMagain / TFF'. The workflow is named 'docker-ci-trigger' and is currently in a 'Queued' state. The workflow file is 'build-and-push-docker.yml' and it is triggered by a repository dispatch. The workflow consists of four steps: 'docker\_build', 'docker\_push', 'scp\_manifest', and 'k8s\_deploy'. The 'docker\_build' step is currently running and has a duration of 14s. Below the workflow steps, there are two annotations: one warning about the use of deprecated Node.js version 12 and another warning about the use of deprecated Node.js version 10. The interface also includes a sidebar with links to 'Summary', 'Jobs', 'Run details', 'Usage', and 'Workflow file'.

Résumé des 4 étapes de ce workflow :

- docker\_build : construit l'image docker en local grâce au Dockerfile suivant :

```
FROM httpd:2.4.59-bookworm
EXPOSE 80
WORKDIR /usr/local/apache2/htdocs
COPY . .
```

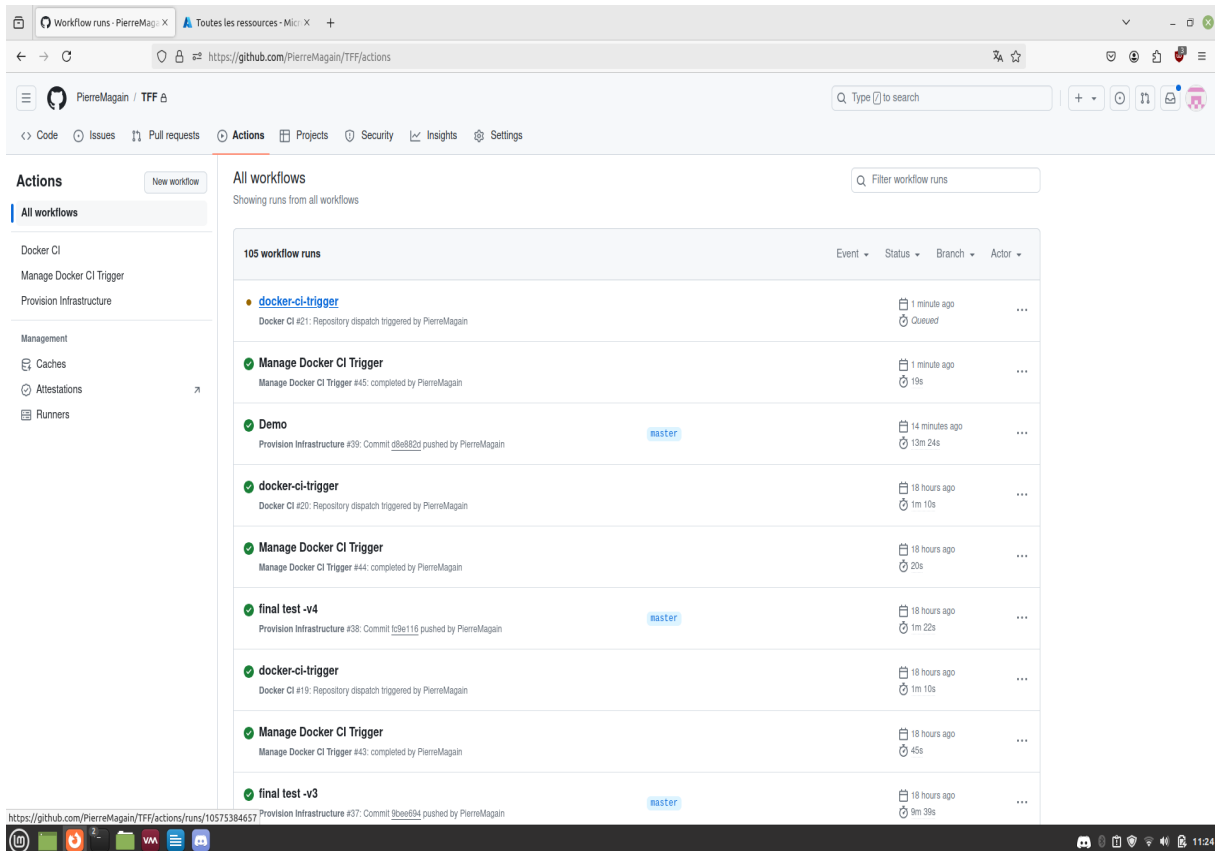
- docker\_push : pousse l'image construite sur DockerHub grâce au secrets des GitHub Actions

- scp\_manifest : envoie le manifest\_tff via une commande scp sur le dossier personnel du node master

- applique ce manifest via la commande :

```
microk8s kubectl apply -f /home/azureuser/manifest_tff.yml
```

## 3.5. Enchainement des Workflow



The screenshot displays the GitHub Actions interface for the repository 'PierreMagain/TFF'. The left sidebar shows the 'Actions' tab with a list of workflow runs. The main area, titled 'All workflows', shows a list of 105 workflow runs. The table below summarizes the visible runs:

Workflow Name	Event	Status	Branch	Actor	Time
docker-ci-trigger	Docker CI #21: Repository dispatch triggered by PierreMagain	Completed			1 minute ago
Manage Docker CI Trigger	Manage Docker CI Trigger #45: completed by PierreMagain	Completed			1 minute ago
Demo	Provision Infrastructure #39: Commit d9e882d pushed by PierreMagain	Completed	master		14 minutes ago
docker-ci-trigger	Docker CI #20: Repository dispatch triggered by PierreMagain	Completed			16 hours ago
Manage Docker CI Trigger	Manage Docker CI Trigger #44: completed by PierreMagain	Completed			16 hours ago
final test -v4	Provision Infrastructure #38: Commit b9e1116 pushed by PierreMagain	Completed	master		16 hours ago
docker-ci-trigger	Docker CI #19: Repository dispatch triggered by PierreMagain	Completed			16 hours ago
Manage Docker CI Trigger	Manage Docker CI Trigger #43: completed by PierreMagain	Completed			16 hours ago
final test -v3	Provision Infrastructure #37: Commit 9bee694 pushed by PierreMagain	Completed	master		16 hours ago

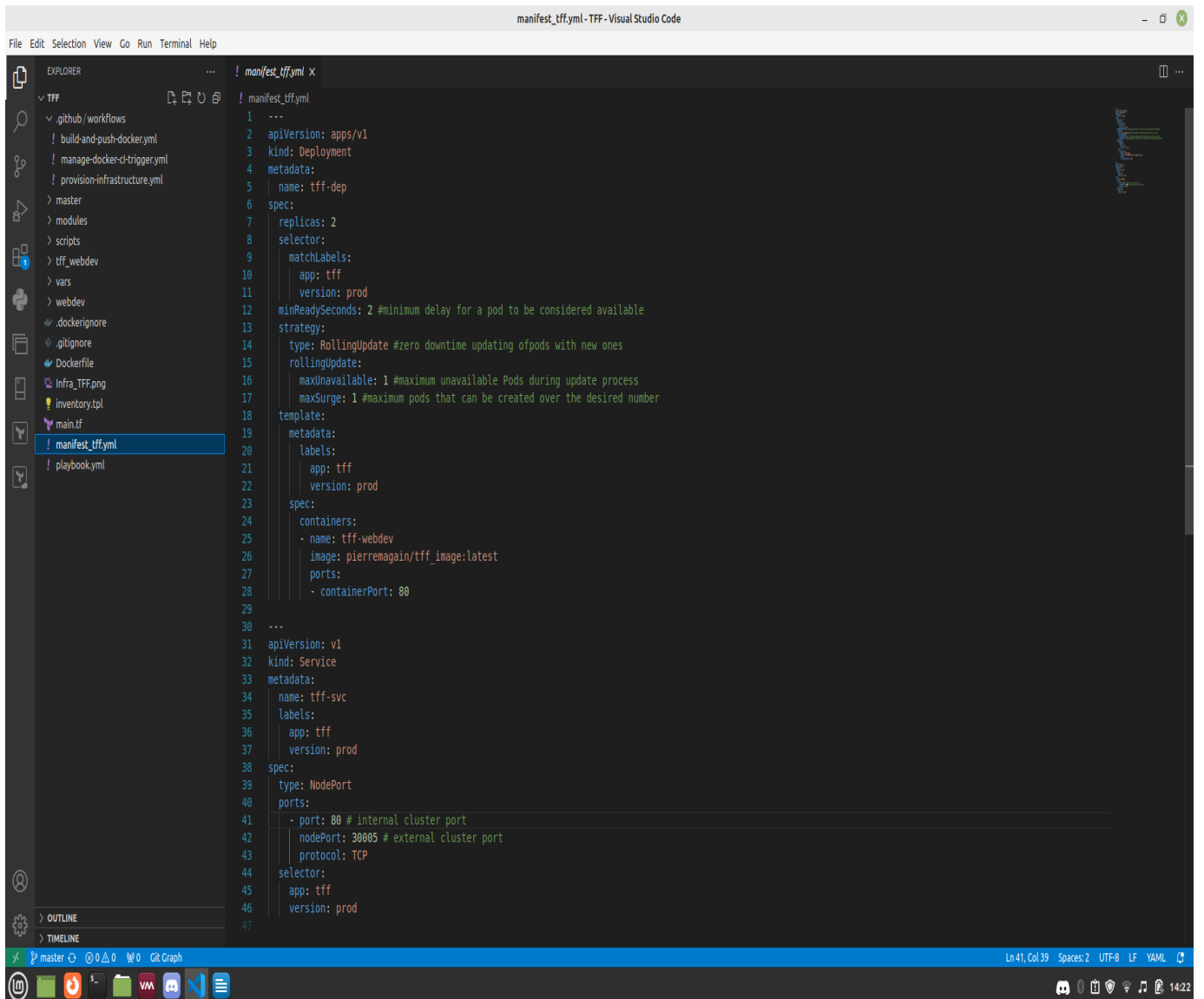
## 4. Déploiement réussi

Une fois le pipeline terminé avec succès, il suffit de se connecter en http sur l'adresse ip publique de la VM master sur le port mentionné par le manifest\_tff.yml.

Celui ci crée un deployment nommé tff-dep qui comprends 2 répliquas, qui sont des pods contenant chacun un conteneur utilisant l'image de notre site web que l'on a envoyé sur le DockerHub

Le manifest crée également tff-svc qui est un service NodePort qui expose le port 30005. D'ailleurs dans les groupes de sécurité de chaque VM , j'ai ouvert le port 30005 en toute connaissance de cause, bien évidemment

## manifest\_tff.yml que voici :



```
1 ---
2 apiVersion: apps/v1
3 kind: Deployment
4 metadata:
5   name: tff-dep
6 spec:
7   replicas: 2
8   selector:
9     matchLabels:
10       app: tff
11       version: prod
12   minReadySeconds: 2 #minimum delay for a pod to be considered available
13   strategy:
14     type: RollingUpdate #zero downtime updating of pods with new ones
15     rollingUpdate:
16       maxUnavailable: 1 #maximum unavailable Pods during update process
17       maxSurge: 1 #maximum pods that can be created over the desired number
18   template:
19     metadata:
20       labels:
21         app: tff
22         version: prod
23     spec:
24       containers:
25       - name: tff-webdev
26         image: pierremagain/tff_image:latest
27         ports:
28         - containerPort: 80
29 ---
30
31 apiVersion: v1
32 kind: Service
33 metadata:
34   name: tff-svc
35 labels:
36   app: tff
37   version: prod
38 spec:
39   type: NodePort
40   ports:
41     - port: 80 # internal cluster port
42       nodePort: 30085 # external cluster port
43       protocol: TCP
44   selector:
45     app: tff
46     version: prod
47
```



Et donc : `http://<adresse_ip_public_vm_master>:30005`



docker-ci-trigger - PierreMa X Toutes les ressources - Micr X J'apprends à coder un site web X +

4.184.159.117:30005

HTML CSS

Accueil Sandbox Contact

## Bienvenue sur le site d'apprentissage du codage web



Ceci est mon premier paragraphe. Son contenu "coulera" automatiquement de ligne en ligne jusqu'à la balise fermante

Mon deuxième paragraphe. La balise d'ouverture du paragraphe a automatiquement ajouté un "retour" à la ligne

### Une liste ordonnée

50. Bruxelles
49. Paris
48. Berlin

### Une liste non ordonnée

- Lundi
- Mardi
- Mercredi

### Exemple de fichier audio

11:25