PeerIsland DevOps Assignment — Full concrete files

This document contains a ready-to-copy set of files for the PeerIsland DevOps assignment. Each file is presented with its path and full contents — copy each file into your repo under the same path.

Repo layout

```
peerisland-devops-assignment/
├─ infra/
│  ├─ provider.tf
│  ├─ main.tf
│  ├─ variables.tf
│  ├─ outputs.tf
│  └─ terraform.tfvars.example
├─ app/
│  ├─ service-a/
│  │  ├─ app.py
│  │  ├─ requirements.txt
│  │  └─ Dockerfile
│  └─ k8s/
│     └─ base/
│        ├─ deployment.yaml
│        ├─ service.yaml
│        └─ hpa.yaml
├─ charts/
│  └─ prometheus-values.yaml
├─ .github/
│  └─ workflows/
│     ├─ terraform.yml
│     └─ ci-cd.yml
├─ grafana/
│  └─ dashboards/
│     └─ app-dashboard.json
├─ scripts/
│  └─ prometheus-report.py
└─ README.md
```

`infra/provider.tf`

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">=3.0"
    }
  }
  required_version = ">= 1.3.0"
}

provider "azurerm" {
  features {}
}
```

`infra/variables.tf`

```
variable "azure_subscription_id" {
  description = "Azure subscription id"
  type        = string
}
variable "azure_tenant_id" {
  description = "Azure tenant id"
  type        = string
}
variable "location" {
  description = "Azure region"
  type        = string
  default     = "eastus"
}
variable "rg_name" {
  description = "Resource group name"
  type        = string
  default     = "peerisland-rg"
}
variable "node_count" {
  description = "AKS initial node count"
  type        = number
  default     = 2
}
variable "acr_name" {
  description = "ACR name (must be globally unique)"
  type        = string
}
variable "aks_name" {
  description = "AKS cluster name"
  type        = string
  default     = "peerisland-aks"
}
```

`infra/main.tf`

```
resource "azurerm_resource_group" "rg" {
  name     = var.rg_name
  location = var.location
}

resource "azurerm_virtual_network" "vnet" {
  name                = "${var.rg_name}-vnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

resource "azurerm_subnet" "aks_subnet" {
  name                 = "aks-subnet"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}

resource "azurerm_container_registry" "acr" {
  name                = var.acr_name
  resource_group_name = azurerm_resource_group.rg.name
  location            = azurerm_resource_group.rg.location
  sku                 = "Standard"
  admin_enabled       = false
}

resource "azurerm_kubernetes_cluster" "aks" {
  name                = var.aks_name
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  dns_prefix          = "${var.rg_name}-aks"

  default_node_pool {
    name            = "agentpool"
    node_count      = var.node_count
    vm_size          = "Standard_D2s_v3"
    vnet_subnet_id   = azurerm_subnet.aks_subnet.id
    enable_auto_scaling = true
    min_count = 1
    max_count = 3
  }

  identity {
    type = "SystemAssigned"
  }

  role_based_access_control { enabled = true }

  network_profile {
    network_plugin    = "azure"
    load_balancer_sku = "standard"
  }
}
```

`infra/outputs.tf`

```
output "aks_cluster_name" {
  value = azurerm_kubernetes_cluster.aks.name
}

output "aks_kube_config" {
  value     = azurerm_kubernetes_cluster.aks.kube_config.0.raw_kube_config
  sensitive = true
}

output "acr_login_server" {
  value = azurerm_container_registry.acr.login_server
}
```

`infra/terraform.tfvars.example`

```
azure_subscription_id = "00000000-0000-0000-0000-000000000000"
azure_tenant_id       = "00000000-0000-0000-0000-000000000000"
location              = "eastus"
rg_name               = "peerisland-rg"
acr_name              = "peerislandacr12345"
```

```
aks_name                = "peerisland-aks"
node_count              = 2
```

Application

`app/service-a/app.py`

```
from flask import Flask, jsonify, request
import os
app = Flask(__name__)

@app.route('/')
def hello():
    return jsonify({
        "message": "hello from service-a",
        "pod": os.getenv('HOSTNAME')
    })

@app.route('/health')
def health():
    return jsonify({"status": "ok"})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

`app/service-a/requirements.txt`

```
Flask==2.1.3
requests==2.31.0
```

`app/service-a/Dockerfile`

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py ./
```

```
EXPOSE 80
CMD ["python", "app.py"]
```

Kubernetes manifests

`app/k8s/base/deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: service-a
  labels:
    app: service-a
spec:
  replicas: 2
  selector:
    matchLabels:
      app: service-a
  template:
    metadata:
      labels:
        app: service-a
    spec:
      containers:
      - name: service-a
        image: REPLACE_WITH_ACR_LOGIN/service-a:latest
        ports:
        - containerPort: 80
        readinessProbe:
          httpGet:
            path: /health
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 10
        resources:
          requests:
            cpu: "100m"
            memory: "128Mi"
          limits:
            cpu: "500m"
            memory: "512Mi"
```

`app/k8s/base/service.yaml`

```yaml
apiVersion: v1
kind: Service
metadata:
  name: service-a
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: service-a
```

`app/k8s/base/hpa.yaml`

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: service-a-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: service-a
  minReplicas: 2
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 60
```

Prometheus & Grafana Helm values

`charts/prometheus-values.yaml`

```yaml
# Minimal kube-prometheus-stack values for production minded defaults.
prometheus:
  prometheusSpec:
    retention: 14d

grafana:
  enabled: true
  adminUser: admin
  # Do NOT store plaintext passwords in values; prefer External Secrets/Key
Vault.
  adminPassword: "replace-with-secure-password-or-use-keyvault"
  ingress:
    enabled: false
    # configure ingress host if required

kubeStateMetrics:
  enabled: true
nodeExporter:
  enabled: true
```

GitHub Actions workflows

`.github/workflows/terraform.yml`

```
name: Terraform CI
on:
  push:
    paths:
      - 'infra/**'

permissions:
  contents: read
  id-token: write

jobs:
  terraform:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v2

      - name: Terraform Init
        working-directory: ./infra
        run: terraform init -input=false
      - name: Terraform Format
        working-directory: ./infra
        run: terraform fmt -check

      - name: Run Checkov (IaC security scan)
        uses: bridgecrewio/checkov-action@master
        with:
          directory: infra

      - name: Terraform Plan
        working-directory: ./infra
        run: terraform plan -out=tfplan -input=false

      - name: Upload tfplan
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: tfplan
          path: infra/tfplan

      - name: Terraform Apply
        if: github.ref == 'refs/heads/main'
        working-directory: ./infra
        env:
          ARM_SUBSCRIPTION_ID: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
          ARM_CLIENT_ID: ${{ secrets.AZURE_CLIENT_ID }}
          ARM_CLIENT_SECRET: ${{ secrets.AZURE_CLIENT_SECRET }}
          ARM_TENANT_ID: ${{ secrets.AZURE_TENANT_ID }}
        run: terraform apply -auto-approve tfplan
```

`.github/workflows/ci-cd.yml`

```
name: CI/CD
on:
  push:
    branches: [ main ]
    paths:
      - 'app/**'
      - 'charts/**'

permissions:
  contents: read
  id-token: write

```

```
jobs:
  build-push-deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Login to Azure
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - name: Set variables
        run: |
          echo "ACR=${{ secrets.ACR_NAME }}.azurecr.io" >> $GITHUB_ENV
          echo "IMAGE_NAME=$ACR/service-a:${{ github.sha }}" >> $GITHUB_ENV

      - name: Build Docker image
        run: |
          docker build -t $IMAGE_NAME ./app/service-a

      - name: ACR Login
        run: az acr login --name ${{ secrets.ACR_NAME }}

      - name: Push Docker image
        run: |
          docker push $IMAGE_NAME

      - name: Get AKS credentials
        run: |
          az aks get-credentials --resource-group ${{ secrets.AKS_RG }} --name $
{{ secrets.AKS_NAME }} --overwrite-existing

      - name: Deploy manifests
        run: |
          # replace placeholder image in manifest then apply
          sed -i "s|REPLACE_WITH_ACR_LOGIN/service-a:latest|$
{{ env.IMAGE_NAME }}|g" app/k8s/base/deployment.yaml
          kubectl apply -f app/k8s/base -n default

      - name: Verify rollout
        run: kubectl rollout status deployment/service-a -n default --
timeout=120s
```

Grafana dashboard (example)

`grafana/dashboards/app-dashboard.json`

```json
{ "dashboard": { "id": null, "title": "Service-A Overview", "panels": [] } }
```

> Note: This is a placeholder. Export your dashboard JSON from Grafana (after installing kube-prometheus-stack) and put it here.

Monitoring report script

`scripts/prometheus-report.py`

```python
#!/usr/bin/env python3
"""
Simple Prometheus query exporter for daily CPU/memory/time-series for service-a.
Run inside cluster or where Prometheus is accessible.
"""
import os
import csv
import requests
import datetime

PROM_URL = os.environ.get('PROM_URL', 'http://prometheus-
operated.monitoring.svc:9090')
OUTPUT_DIR = os.environ.get('OUTPUT_DIR', '/tmp')

def query_range(query, start, end, step='300'):
    url = f"{PROM_URL}/api/v1/query_range"
    params = {'query': query, 'start': start, 'end': end, 'step': step}
    r = requests.get(url, params=params, timeout=30)
    r.raise_for_status()
    return r.json()

def write_csv(name, timeseries):
    fname = os.path.join(OUTPUT_DIR, name)
    with open(fname, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(['timestamp', 'value'])
        for t,v in timeseries:
            ts = datetime.datetime.utcfromtimestamp(float(t)).isoformat()
            writer.writerow([ts, v])
    print('Wrote', fname)
```

```
if __name__ == '__main__':
    now = datetime.datetime.utcnow()
    end = now.isoformat() + 'Z'
    start = (now - datetime.timedelta(days=1)).isoformat() + 'Z'

    queries = {
        'cpu': 'rate(container_cpu_usage_seconds_total{pod=~"service-a.*"}
[5m])',
        'memory': 'container_memory_working_set_bytes{pod=~"service-a.*"}',
        'http_errors': 'sum(rate(http_requests_total{job=~"service-
a",status=~"5.."}[5m]))'
    }

    for name, q in queries.items():
        try:
            res = query_range(q, start, end)
            # pick first result series (may need to adapt if multiple pods)
            if res['data']['result']:
                values = res['data']['result'][0]['values']
                write_csv(f'service-a-{name}-{now.strftime("%Y%m%d")}.csv',
values)
            else:
                print('No data for', name)
        except Exception as e:
            print('Query failed for', name, e)
```

README.md

```
# PeerIsland DevOps Assignment

This repository contains a complete, ready-to-copy solution for provisioning an
AKS-based environment on Azure using Terraform, deploying a small microservice,
installing Prometheus+Grafana, running IaC security scans with Checkov, and a
CI/CD pipeline using GitHub Actions.

## What is included
- `infra/` — Terraform code (RG, VNet, Subnet, ACR, AKS)
- `app/service-a` — sample Flask microservice and Dockerfile
- `app/k8s/base` — Kubernetes manifests (Deployment, Service, HPA)
- `charts/prometheus-values.yaml` — Helm values for kube-prometheus-stack
- `grafana/dashboards` — placeholder for dashboard JSONs
- `scripts/prometheus-report.py` — simple Prometheus daily exporter
- `.github/workflows` — Terraform and CI/CD workflows (Checkov integrated)
```

~~~markdown
## Quickstart (high level)
1. Create an Azure service principal and set GitHub secrets: `AZURE_CREDENTIALS`
(JSON), `AZURE_SUBSCRIPTION_ID`, `AZURE_TENANT_ID`, `ACR_NAME`, `AKS_NAME`,
`AKS_RG`.
2. Copy `infra/terraform.tfvars.example` to `infra/terraform.tfvars` and update
values.
3. Run locally (optional):
   ```bash
   cd infra
   terraform init
   terraform apply -auto-approve
   ```
4. Build & push image (CI does this automatically):
   - Ensure AKS can pull from ACR: `az aks update -n <aks> -g <rg> --attach-acr
<acr>`
5. Install monitoring:
   ```bash
   helm repo add prometheus-community https://prometheus-community.github.io/
helm-charts
   helm repo update
   helm upgrade --install prometheus prometheus-community/kube-prometheus-stack
-n monitoring --create-namespace -f charts/prometheus-values.yaml
   ```
6. Deploy app (CI will apply manifests), or run:
   ```bash
   sed -i "s|REPLACE_WITH_ACR_LOGIN/service-a:latest|<ACR_LOGIN>/service-
a:<TAG>|g" app/k8s/base/deployment.yaml
   kubectl apply -f app/k8s/base
   ```

## Notes & security
- Do not commit secrets. Use Azure Key Vault + External Secrets Operator to sync
secrets into the cluster.
- Checkov is configured in CI to scan Terraform and block runs on failures
(adjust severity thresholds as needed).
- Use managed identity and role assignments to allow AKS to pull images from ACR
(preferred over admin user).

## Evidence to collect for submission
- Terraform apply logs (or CI artifacts)
- Checkov scan output (CI logs)
- `kubectl get pods/services` showing app running
- Grafana dashboard screenshots (CPU/memory/latency)
- Daily report CSVs produced by `prometheus-report.py`
~~~

Final notes

- The files here are ready to copy/paste. Replace placeholders (e.g., `REPLACE_WITH_ACR_LOGIN`, GitHub Secrets) with your real values before running.
- If you'd like, I can also produce a downloadable `.zip` of these files or generate a single branch-ready tree of files you can import directly into GitHub. Tell me which format you prefer.