Advanced DevOps – Security, Automation & Multi-Cloud CI/CD Using GitHub Actions And Azure DevOps

🔷 What is GitHub Actions?

- **GitHub Actions** is a **CI/CD (Continuous Integration and Continuous Deployment) automation tool** that allows developers to automate workflows directly in their **GitHub repositories**.
- It enables tasks such as **building, testing, deploying applications, running security scans, and managing infrastructure**—all triggered by events like a code push or a pull request.

Key Features of GitHub Actions

✅ **Event-Driven Workflows** – Automatically run actions when code is pushed, PRs are created, or issues are opened.
✅ **YAML-based Configuration** – Define automation workflows using simple YAML files.
✅ **Multi-Cloud Support** – Deploy applications to **AWS, Azure, GCP, Kubernetes, or on-prem servers**.
✅ **Parallel and Sequential Jobs** – Run jobs in parallel for faster execution or sequentially for dependencies.
✅ **Security & Compliance** – Integrate security scanning tools like **Trivy, Snyk, or SonarQube** in CI/CD.
✅ **Self-Hosted Runners** – Run workflows on your own **custom VMs or Kubernetes clusters**.

How GitHub Actions Work

A GitHub Actions workflow consists of:
1️⃣ **Events:**
Events trigger workflows. Examples:
🔷 **push** → Runs when code is pushed.
🔷 **pull_request** → Runs on PR creation.
🔷 **schedule** → Runs on a **CRON schedule**.
🔷 **workflow_dispatch** → Manual trigger.

2️⃣ **Jobs:**
A job is a set of steps **executed on a runner**. Jobs can run **in parallel** or **depend on each other**.

3️⃣ **Steps:**
A step is **a single task inside a job** (e.g., checkout code, build, test, deploy).

4️⃣ **Actions:**
Reusable units in workflows (e.g., **checkout code, set up dependencies, run tests**).

Step 1: Multi-Cloud CI/CD with GitHub Actions + Azure DevOps

- Many organizations use **multi-cloud** (Azure, AWS, GCP) for reliability and flexibility.
- Integrating **GitHub Actions** with **Azure DevOps** allows teams to build, test, and deploy applications across different cloud providers efficiently.

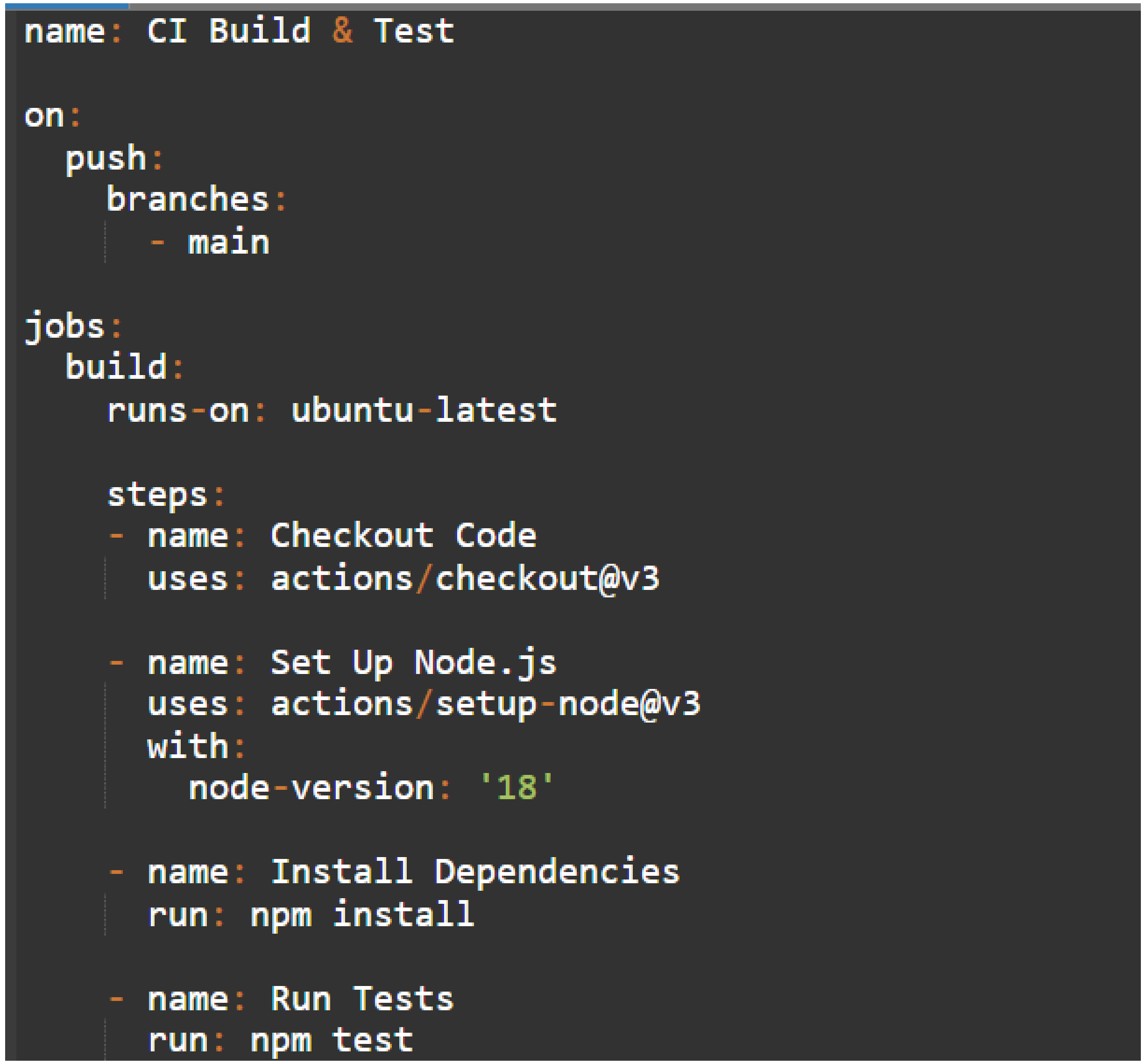
🔧 Implementation Steps:

✅ Step 1.1: Create a GitHub Repository

1. Go to GitHub → Click **New Repository**
2. Name it (e.g., multi-cloud-cicd) → Set it to **Public/Private**
3. Initialize with a **README** and clone it locally.

✅ Step 1.2: Create a GitHub Actions Workflow for CI

1. In your GitHub repo, go to **Actions** → Click **New Workflow**
2. Click **Set up a Workflow yourself**
3. Add the following YAML file (.github/workflows/ci.yml):
4. Commit and push the file → The workflow will **automatically trigger** on a push to main.



```yaml
name: CI Build & Test

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Code
      uses: actions/checkout@v3

    - name: Set Up Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '18'

    - name: Install Dependencies
      run: npm install

    - name: Run Tests
      run: npm test
```

✅ **Step 1.3: Integrate GitHub Actions with Azure DevOps for Deployment**

1. In **Azure DevOps**, go to **Project Settings** → **Service Connections**
2. Click **New Service Connection** → Select **GitHub**
3. Authenticate and **authorize access** to your repository.
4. In **Azure Pipelines**, create a new **release pipeline** and select **GitHub** as the source.
5. Deploy the app using Azure DevOps release pipelines.

Step 2: Container Security with Trivy & Microsoft Defender

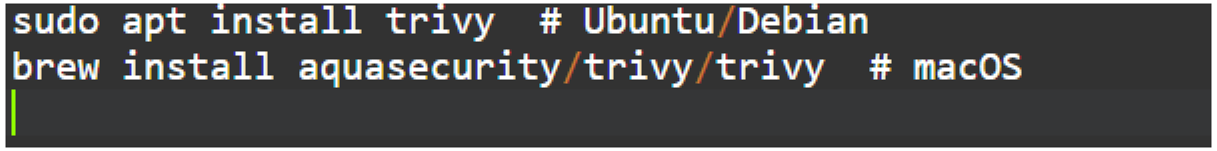
Containers may contain vulnerabilities in base images or dependencies. **Trivy (by Aqua Security)** scans Docker images for vulnerabilities, and **Microsoft Defender for Cloud** provides runtime protection.

🔧 **Implementation Steps:**

✅ **Step 2.1: Install Trivy**

Run the following command based on your OS:

For Linux/macOS:



```
sudo apt install trivy  # Ubuntu/Debian
brew install aquasecurity/trivy/trivy  # macOS
```

✅ Step 2.2: Scan a Docker Image for Vulnerabilities

```
trivy image nginx:latest
```

✅ Step 2.3: Integrate Trivy into Azure DevOps Pipeline

1. In **Azure DevOps**, go to **Pipelines → Edit your YAML pipeline**.
2. Add the following **Trivy scanning stage** to scan Docker images.

```
stages:
- stage: SecurityScan
  displayName: Security Scanning
  jobs:
    - job: ScanImage
      displayName: Scan Docker Image with Trivy
      steps:
      - script: |
          sudo apt install -y trivy
          trivy image my-container-registry.azurecr.io/myapp:latest
        displayName: 'Run Trivy Scan'
```

✅ **Step 2.4: Enable Microsoft Defender for Cloud**

1. Go to **Azure Portal** → **Microsoft Defender for Cloud**.
2. Enable **Defender for Kubernetes** & **Defender for Container Registries**.
3. This will provide **real-time alerts** if security risks are detected in your containerized workloads.

Step 3: Automate Release Approvals with Azure DevOps Gates

Approval Gates prevent risky deployments by ensuring security, compliance, and manual approvals before releasing to production.

🔧 Implementation Steps:

✅ **Step 3.1: Enable Approval Gates in Azure DevOps**

1. Go to **Azure DevOps** → **Pipelines** → **Releases**
2. Select your release pipeline → Click on the **Stage (e.g., Production)**
3. Click on **Pre-deployment conditions** → **Enable Gates**

✅ Step 3.2: Add Conditions (Approval Policies)

1. Click **+ Add Gate** → Choose from:
    1. **Azure Monitor Alert** (Blocks release if an alert is triggered).
    2. **Work Item Query** (Ensures pending issues are resolved).
    3. **Security Scan Results** (Waits for vulnerability scan results).
2. Configure conditions → Click **Save**.

✅ Step 3.3: Require Manual Approvals for High-Risk Deployments

1. In **Pre-deployment conditions**, enable **Manual Approval**.
2. Assign **Approvers (Team Leads, Security Officers, or Compliance Teams)**.
3. Before a deployment, an **approver must review & approve** the release.

Secure DevOps Pipeline: A Security & Compliance Project using Azure DevOps

Project Overview

- This project aims to **secure an end-to-end DevOps pipeline** by integrating security and compliance measures into the **CI/CD process** using **Azure DevOps, Trivy, Microsoft Defender, and Approval Gates**.

✅ Key Features:

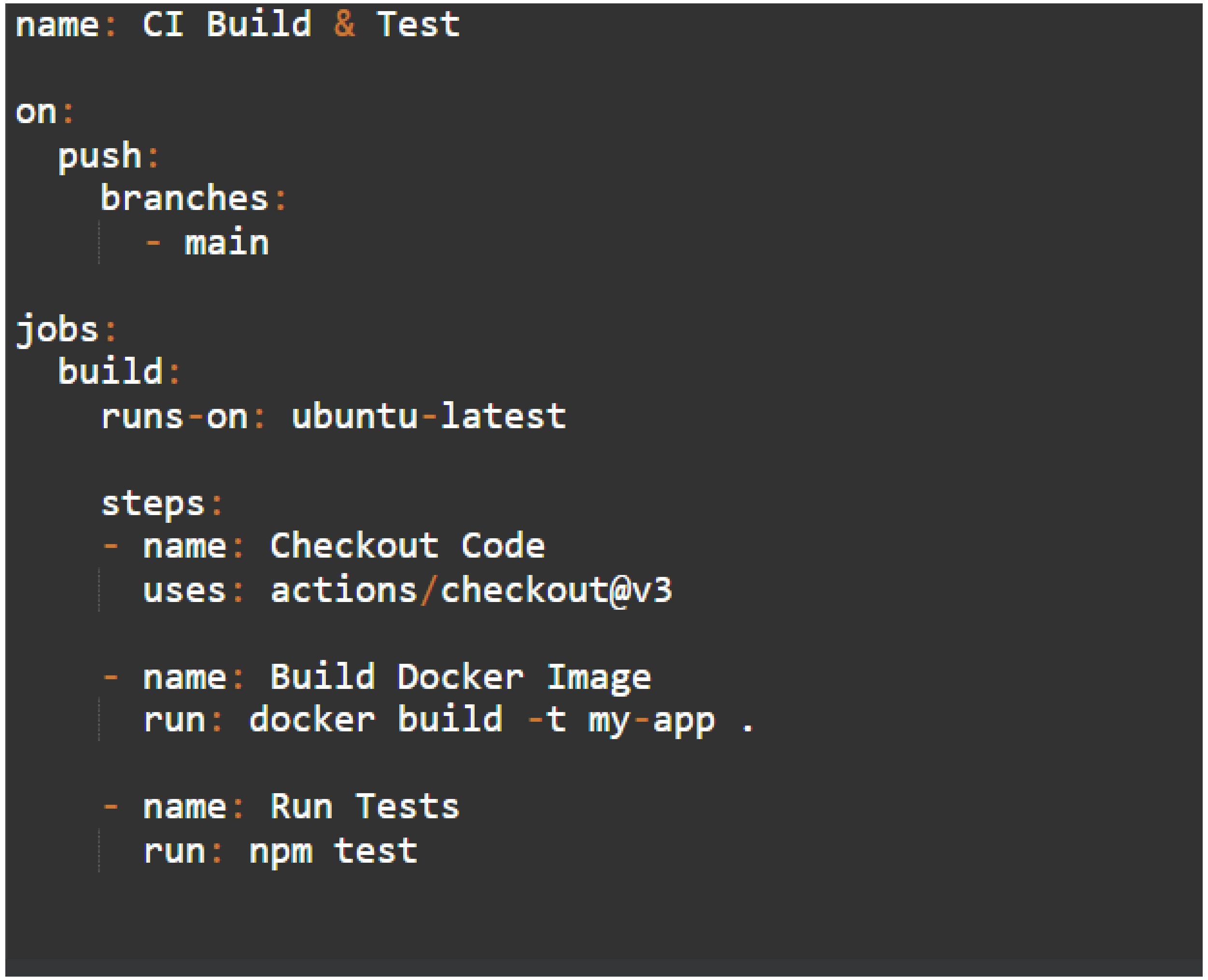
- **Multi-Cloud CI/CD Integration**: GitHub Actions + Azure DevOps for deployment flexibility.
- **Container Security**: Scanning Docker images with **Trivy** & enabling **Microsoft Defender for Cloud**.
- **Automated Compliance & Approvals**: Implementing **Azure DevOps Gates** for security validation.
- **Role-Based Access Control (RBAC)**: Restricting permissions to enforce least privilege access.
- **Secrets Management**: Using **Azure Key Vault** to store credentials securely.

Project Architecture

🔷 **Step 1:** Code is pushed to **GitHub** → Triggers a **GitHub Actions Workflow** for build & test.
🔷 **Step 2:** Build artifacts are sent to **Azure DevOps** → Security scans are triggered using **Trivy**.
🔷 **Step 3:** Deployment approval is **validated using Azure DevOps Gates** (Security, Manual).
🔷 **Step 4:** If approved, the release is deployed to **Azure Kubernetes Service (AKS)**.
🔷 **Step 5: Microsoft Defender for Cloud** continuously monitors security & compliance.

🔷 Step 1: Set Up GitHub Actions for CI

1. **Create a GitHub repository** → Push your application code.
2. **Create a GitHub Actions Workflow** (.github/workflows/ci.yml):
3. **Commit and push** → The workflow triggers on every push.



```yaml
name: CI Build & Test

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Code
      uses: actions/checkout@v3

    - name: Build Docker Image
      run: docker build -t my-app .

    - name: Run Tests
      run: npm test
```

🔷 Step 2: Scan Docker Images with Trivy in Azure DevOps

1.Install Trivy:

```
sudo apt install trivy  # Ubuntu/Debian
```

2.Run a security scan:

```
trivy image my-container-registry.azurecr.io/my-app:latest
```

3.Integrate Trivy in Azure DevOps Pipeline (azure-pipelines.yml):

```
stages:
- stage: SecurityScan
  jobs:
    - job: ScanImage
      steps:
      - script: |
          sudo apt install -y trivy
          trivy image my-container-registry.azurecr.io/my-app:latest
        displayName: 'Run Trivy Scan'
```

🔷 Step 3: Secure Secrets with Azure Key Vault

1. **Create an Azure Key Vault** in the Azure Portal.
2. **Store secrets** (e.g., database password, API keys).
3. **Integrate Key Vault with Azure DevOps**:

```yaml
- task: AzureKeyVault@2
  inputs:
    azureSubscription: 'My Azure Subscription'
    KeyVaultName: 'my-keyvault'
    SecretsFilter: '*'
    RunAsPreJob: true
```

🔷 Step 4: Implement Deployment Gates in Azure DevOps

1. **Go to Azure DevOps → Pipelines → Releases**.
2. Click on the **Stage (e.g., Production)** → **Enable Pre-deployment Gates**.
3. **Add security conditions**:
    1. **Azure Monitor Alert** – Blocks if security threats are detected.
    2. **Trivy Scan Results** – Stops deployment if vulnerabilities exist.
    3. **Manual Approvals** – Requires review before production deployment.

🔷 Step 5: Enable Microsoft Defender for Cloud

1. Go to **Azure Portal → Microsoft Defender for Cloud**.
2. Enable:
   1. **Defender for Containers** (scans images in Azure Container Registry).
   2. **Defender for Kubernetes** (real-time threat detection).
3. **Configure Defender to send alerts** to your DevOps pipeline for immediate action.