MLOps Assignment - Task 2 Answers

# 1. Explain the difference between Continuous Integration, Continuous Delivery, and Continuous Deployment. How do GitHub Actions support these practices?

Continuous Integration (CI) is the practice of frequently merging code changes into a shared repository. With each integration, automated builds and tests are executed to catch issues early.  
  
Continuous Delivery (CD) builds on CI by automatically preparing applications for release. Code that passes all tests is packaged and ready for deployment; therefore keeping software deployable at all times  
  
Continuous Deployment goes a step further by automatically releasing every change that passes the pipeline to production without manual intervention. Any change that passes the entire CI/CD pipeline is deployed to production automatically.  
  
GitHub Actions supports these practices using YAML-defined workflows triggered by GitHub events. It enables automation of testing, packaging, and deployment steps.

name: Build and Test  
on:  
 push:  
 branches:  
 - main  
jobs:  
 test:  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
 - name: Set up Python  
 uses: actions/setup-python@v4  
 with:  
 python-version: '3.9'  
 - name: Install dependencies  
 run: pip install -r requirements.txt  
 - name: Run tests  
 run: pytest tests/

# 2. How can GitHub Actions be integrated with cloud services like Azure for end-to-end MLOps workflows? Provide an example scenario.

GitHub Actions can integrate with Azure via Azure CLI or dedicated GitHub Actions like 'azure/login' and 'azure/cli'. This supports end-to-end MLOps including model training, deployment, and monitoring. It allows automation of training, model registration, and deployment to services like Azure Kubernetes Service or Azure ML.  
  
Scenario:  
1. Code is pushed to GitHub.  
2. GitHub Actions sets up the Python environment.  
3. Authenticates with Azure.  
4. Submits a job to Azure ML.  
5. Registers and deploys the trained model.

name: Train and Deploy Model  
on:  
 push:  
 branches: [ main ]  
jobs:  
 train-and-deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
 - name: Set up Python  
 uses: actions/setup-python@v4  
 with:  
 python-version: '3.9'  
 - name: Install dependencies  
 run: |  
 pip install azureml-sdk  
 pip install -r requirements.txt  
 - name: Azure Login  
 uses: azure/login@v1  
 with:  
 creds: ${{ secrets.AZURE\_CREDENTIALS }}  
 - name: Submit training job  
 run: |  
 az ml job create --file ml/train-job.yml --resource-group ${{ secrets.AZURE\_RG }} --workspace-name ${{ secrets.AZURE\_WS }}  
 - name: Register model  
 run: |  
 az ml model register --name my-model --path outputs/model.pkl --resource-group ${{ secrets.AZURE\_RG }} --workspace-name ${{ secrets.AZURE\_WS }}  
 - name: Deploy model to AKS  
 run: |  
 az ml online-endpoint create --name ml-endpoint --file ml/deploy.yml --resource-group ${{ secrets.AZURE\_RG }} --workspace-name ${{ secrets.AZURE\_WS }}

## 3. Compare GitHub Actions with another CI/CD tool (e.g., Jenkins, GitLab CI/CD). What are the advantages and disadvantages of using GitHub Actions for MLOps?

GitHub Actions is tightly integrated with GitHub, simple to set up, and requires no external infrastructure. It supports a large marketplace of actions. But there are several alternatives which achieving similar outcomes can have striking differences in design philosophy, approach and scope.

Below is a breakdown of the design philosophies, usage approach, capabilities, and limitations of five major CI/CD and infrastructure tools.

### GitHub Actions

Event-driven automation tightly integrated with GitHub. Emphasizes simplicity, developer productivity, and YAML-defined workflows.

### GitLab CI/CD

Comprehensive DevOps solution offering version control, CI/CD, and security scanning. Provides an all-in-one experience within the GitLab platform.

### Jenkins

Extensible automation server. Highly flexible with a plugin-based architecture and custom Groovy-based scripting.

### Puppet (Bolt)

Declarative configuration management platform that ensures system state. Bolt adds task-based orchestration without agents.

### Azure DevOps

Enterprise-focused modular suite for planning, building, testing, and deploying applications with deep Azure integration.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tool** | **Hosting** | **Config Language** | **Extensibility** | **Primary Use** | **Limitations** |
| **GitHub Actions** | Cloud / Self-hosted | YAML | Marketplace Actions | GitHub-native CI/CD | Limited control, GitHub dependency |
| **GitLab CI/CD** | Cloud / Self-hosted | YAML | Built-in Templates | Integrated DevOps | Tightly coupled to GitLab |
| **Jenkins** | Self-hosted | Groovy DSL | 1,800+ Plugins | Highly customized CI/CD | Requires manual setup and maintenance |
| **Puppet (Bolt)** | Self-hosted | Puppet DSL / YAML | Forge Modules | Infra config management | Not a CI/CD orchestrator |
| **Azure DevOps** | Cloud / Self-hosted | YAML / GUI | Azure Extensions | Enterprise DevOps | Complex for small teams |

## CI/CD Code Snippets for reference and comparison

### GitHub Actions Example:

# Triggers pipeline on every push to 'main' branch  
name: CI Pipeline  
on:  
 push:  
 branches: [ main ]  
  
jobs:  
 build:  
 runs-on: ubuntu-latest # Runs job on Ubuntu VM  
 steps:  
 - uses: actions/checkout@v3 # Clones repo  
 - uses: actions/setup-python@v4  
 with:  
 python-version: '3.9' # Sets up Python  
 - run: pip install -r requirements.txt # Installs dependencies  
 - run: pytest # Runs test suite

### GitLab CI/CD Example:

# Defines build and test stages  
stages:  
 - build  
 - test  
  
build:  
 stage: build  
 script:  
 - pip install -r requirements.txt # Install packages  
  
test:  
 stage: test  
 script:  
 - pytest # Run tests

### Jenkins Example:

pipeline {  
 agent any  
  
 stages {  
 stage('Build') {  
 steps {  
 sh 'pip install -r requirements.txt' # Install Python packages  
 }  
 }  
 stage('Test') {  
 steps {  
 sh 'pytest' # Execute tests  
 }  
 }  
 }  
}

### Puppet Bolt Example:

# Run command across servers  
bolt command run 'uptime' --targets all  
  
# Apply Puppet configuration  
bolt apply site.pp --targets all # site.pp declares infra state

### Azure DevOps Example:

trigger:  
 branches:  
 include:  
 - main  
  
pool:  
 vmImage: 'ubuntu-latest' # Hosted runner  
  
steps:  
- task: UsePythonVersion@0  
 inputs:  
 versionSpec: '3.9'  
  
- script: |  
 pip install -r requirements.txt  
 pytest # Run tests  
 displayName: 'Install and Test'

### 4. How has your understanding of MLOps changed after learning about GitHub Actions? In what scenarios might you choose not to use GitHub Actions for your MLOps workflow?

Learning GitHub Actions has highlighted how automation and reproducibility are essential to MLOps. It's clear how pipelines can automate model evaluation, enforce standards, and trigger deployments seamlessly.

Before working with GitHub Actions, I understood MLOps as a broad concept involving model development, experimentation, and deployment, but not how directly CI/CD principles could be applied.  
  
After learning GitHub Actions, I now see MLOps as an automated and reproducible extension of DevOps, enabling full-lifecycle model delivery — from data ingestion to retraining, deployment, and monitoring — all integrated into code repositories.  
  
GitHub Actions taught me how to automate everything:  
- Versioned workflows triggered by GitHub events  
- Reproducible environments via setup actions or Docker  
- Integration with cloud platforms like Azure for scalable ML operations  
  
It also helped me appreciate how MLOps improves collaboration by giving data scientists and ML engineers shared tooling and continuous feedback loops.

### However, GitHub Actions may not be ideal in the following scenarios:

* **Heavy computation or large datasets:** Runners have resource/time limits; alternatives like Kubeflow or Azure ML Pipelines are more suitable.
* **Strict governance or regulatory environments:** GitHub Actions lacks on-prem controls and enterprise security features out of the box.
* **Multi-cloud or Git-agnostic workflows:** GitHub Actions is GitHub-centric; not ideal for Bitbucket, GitLab, or hybrid setups.
* **Advanced pipeline scheduling or queuing:** Purpose-built orchestrators provide better control over job execution, retries, and state management.

In summary, GitHub Actions is a powerful entry point into MLOps — especially for GitHub-first teams or rapid iteration cycles — but more complex workflows may require complementing it with dedicated MLOps platforms or orchestrators.