13 – CI/CD & Cloud Execution

🎯 Objective

Learn how to automate JMeter tests so they run automatically whenever developers push code. This is called **"Shift-Left Performance Testing"**.

1️⃣ Why Automate Performance Tests?

If you only run load tests once before a major release, it's too late. Fixing performance issues at the end is expensive.
The Goal: Run a small "Performance Smoke Test" on every commit.

- If `Login API` becomes 500ms slower today, we fail the build immediately.

2️⃣ The CI/CD Pipeline Workflow

Regardless of the tool (Jenkins, Azure DevOps, GitHub Actions), the logic is the same:

1. **Checkout Code:** Get the `.jmx` scripts from Git.
2. **Setup Environment:** Install Java and JMeter (or use a Docker image).
3. **Execute Test:** Run the CLI command (`jmeter -n -t ...`).
4. **Check Thresholds:** Did the test pass? (e.g., Error % < 1).
5. **Publish Report:** Save the HTML dashboard as a build artifact.

3️⃣ Example: Azure DevOps Pipeline (YAML)

Since your student is learning Azure, this is the most relevant example.

```
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'
```

```
steps:
# 1. Install Java
- task: JavaToolInstaller@0
  inputs:
    versionSpec: '11'
    jdkArchitectureOption: 'x64'
    jdkSourceOption: 'PreInstalled'

# 2. Install JMeter (using a simple script)
- script: |
    wget [https://archive.apache.org/dist/jmeter/binaries/apache-jmeter-5
    tar -xf apache-jmeter-5.6.3.tgz
  displayName: 'Install JMeter'

# 3. Run JMeter Test
- script: |
    ./apache-jmeter-5.6.3/bin/jmeter -n -t tests/login_test.jmx -l result
  displayName: 'Run Load Test'

# 4. Publish Test Results (HTML Report)
- task: PublishPipelineArtifact@1
  inputs:
    targetPath: 'report_html'
    artifact: 'JMeterReport'
```

4️⃣ Failing the Build (Performance Assertions)

Running the test is not enough. You must **fail** the pipeline if performance degrades.
You can use the **Taurus** wrapper or a simple Python script to parse the JTL.

Simpler JMeter Native Way:
Use a **JSR223 Assertion** in your JMX plan:

```
if (prev.getTime() > 2000) {
    AssertionResult.setFailure(true);
    AssertionResult.setFailureMessage("Response time exceeded 2s limit!")
}
```

If this fails, JMeter exits with a non-zero code, and Azure DevOps marks the build as ❌ **Failed**.

5️⃣ Jenkins Integration

For Jenkins, there is a dedicated **"Performance Plugin"**.

- **Step 1:** Run JMeter shell command.
- **Step 2:** Add "Publish Performance Test Result Report" post-build action.
- **Step 3:** Configure "Constraint":
- *If Error % > 5%, mark build as Unstable.*
- *If Average Response Time > 1000ms, mark build as Failed.*

6️⃣ Azure Load Testing (The Cloud Native Way)

Instead of installing JMeter on the build agent (as shown in section 3), you can use the **Azure Load Testing Task**.

```
- task: AzureLoadTest@1
  inputs:
    azureSubscription: 'MySubscription'
    loadTestConfigFile: 'config.yaml'
    resourceGroup: 'MyResourceGroup'
    loadTestResource: 'MyLoadTestResource'
```

Benefits:

- No need to install Java/JMeter on the agent.
- Scales to thousands of users instantly.
- Results appear directly in the Azure Portal.

7️⃣ Interview Question

"How do you ensure performance doesn't degrade over time?"

Answer:
"I integrate JMeter into our CI/CD pipeline (e.g., Azure DevOps). On every nightly build, we run a baseline load test. I configured a **quality gate** that fails the pipeline if the 90th percentile response time exceeds 2 seconds or if the error rate is non-zero."

🎯 Mini Exercise

You might not have a full DevOps setup, but you can simulate the logic:

1. Create a batch file (Windows) or shell script (Linux) named `run_test.sh`.
2. Add the JMeter CLI command inside it.
3. Add logic to check if `result.jtl` exists.
4. Run this script.

- *Congratulations, you just wrote the "Step" for a pipeline!*