3. Build and release pipelines
  └ **3.2 Design and implement pipelines**
    └ **3.2.1 Automation tools: GitHub Actions, Azure Pipelines**

---

1. What are the primary differences between GitHub Actions and Azure Pipelines?
2. How do you create and structure a workflow in GitHub Actions?
3. How do you define and configure pipelines using YAML in Azure Pipelines?
4. What triggers can start workflows in GitHub Actions and Azure Pipelines?
5. How do you use runners (GitHub) and agents (Azure) in automation pipelines?
6. How do you securely manage secrets and variables in both tools?
7. How do you implement environment deployments and approvals?
8. How do you integrate testing and code quality gates in automation pipelines?
9. What are best practices for job dependencies and parallel execution?
10. How do you monitor and troubleshoot workflow and pipeline executions?

---

1. What are the primary differences between GitHub Actions and Azure Pipelines?

- *GitHub Actions* is natively integrated with GitHub, focused on repository-centric automation using workflow files in the repo.
- *Azure Pipelines* is part of *Azure DevOps*, supports multi-repo, project-level pipelines, offers both YAML and classic UI, and is more feature-rich for complex enterprise scenarios.

---

2. How do you create and structure a workflow in GitHub Actions?

Create a YAML file in .github/workflows/. Define name, on (trigger), jobs, and each job's runs-on and steps. Each step specifies actions or shell commands.

---

3. How do you define and configure pipelines using YAML in Azure Pipelines?

Create azure-pipelines.yml at the repo root. Define trigger, pool, variables, stages, jobs, and steps. YAML is used to declaratively describe the pipeline structure and logic.

---

4. What triggers can start workflows in GitHub Actions and Azure Pipelines?

- *GitHub Actions:* push, pull_request, schedule, workflow_dispatch (manual), environment events.
- *Azure Pipelines:* trigger (CI on commit), pr, schedule, pipeline (other pipelines), and manual runs.

---

5. How do you use runners (GitHub) and agents (Azure) in automation pipelines?

- *GitHub Actions* uses hosted or self-hosted <u>runners</u> defined in runs-on.
- *Azure Pipelines* uses Microsoft-hosted or self-hosted <u>agents</u> set in the pool definition.

---

6. How do you securely manage secrets and variables in both tools?

- *GitHub Actions*: Store secrets via *Repository Settings > Secrets* and reference as ${{ secrets.NAME }}.
- *Azure Pipelines*: Store secrets in Variable Groups and reference as $(NAME) in YAML.

---

7. How do you implement environment deployments and approvals?

- *GitHub Actions*: Define environments in workflows, set protection rules and required reviewers.
- *Azure Pipelines*: Use environments with approvals and checks; configure in the Environments tab and reference in YAML with environment.

8. How do you integrate testing and code quality gates in automation pipelines?

- *GitHub Actions*: Add test jobs/steps using testing actions; configure status checks to enforce passing tests before merging.
- *Azure Pipelines*:
    - Add test tasks,
    - publish test results,
    - set Quality Gates with policies or extensions (e.g., SonarQube).

---

9. What are best practices for job dependencies and parallel execution?

- *GitHub Actions*: Define job dependencies with needs: and run jobs in parallel by default unless dependencies are set.
- *Azure Pipelines*: Use `dependsOn` for controlling order; jobs and stages run in parallel unless dependencies are specified.

---

10. How do you monitor and troubleshoot workflow and pipeline executions?

- *GitHub Actions*:
    - Review logs in the Actions tab
    - Inspect individual job/step output
    - Use `run: echo` for debugging.
- *Azure Pipelines*:
    - Use the Runs/Logs UI
    - Download logs
    - Enable diagnostics
    - Check task-level error output for troubleshooting