3. Build and release pipelines

└ **3.1 Package management and testing strategy**

└ **3.1.2 Testing Strategy: Unit, Integration, Load**

---

1. What are the differences between unit, integration, and load tests?
2. How do you structure pipelines to run unit, integration, and load tests efficiently?
3. How do you configure test tasks in Azure Pipelines or GitHub Actions?
4. How are test results collected and visualized in Azure DevOps?
5. What are best practices for running unit tests in CI/CD pipelines?
6. How do you isolate and manage dependencies for integration tests?
7. How do you implement load testing in a pipeline?
8. What tools are commonly used for each test type in Azure DevOps?
9. How do you handle test failures in pipelines?
10. How do you ensure code coverage thresholds are enforced during testing?

---

1. What are the differences between unit, integration, and load tests?

- Unit tests validate isolated code units and run quickly without external dependencies.
- Integration tests verify interactions between multiple components or services.
- Load tests simulate real-world traffic or usage to assess performance and stability under stress.

---

2. How do you structure pipelines to run unit, integration, and load tests efficiently?

- Run unit tests early and in parallel on every commit.
- Integration tests run after builds or deployments to test environments.
- Load tests typically run in dedicated stages after integration tests or before production deployments to avoid resource conflicts.

---

3. How do you configure test tasks in Azure Pipelines or GitHub Actions?

Use built-in tasks (e.g., `VsTest`, `DotNetCoreCLI@2`, `PublishTestResults@2`) in *Azure Pipelines*, or actions like `actions/setup-dotnet`, `actions/upload-artifact` in *GitHub Actions*. Specify test commands and result formats in YAML steps.

---

4. How are test results collected and visualized in Azure DevOps?

Publish test results using the `PublishTestResults` task. Results appear in the *Azure Pipelines* Test tab with detailed logs, failure summaries, and trends across runs.

---

5. What are best practices for running unit tests in CI/CD pipelines?

Run all unit tests on every push and pull request. Ensure tests are isolated and fast. Fail the build if any unit test fails. Use code coverage tools to track and enforce minimum coverage levels.

---

6. How do you isolate and manage dependencies for integration tests?

- Use dedicated test environments, containers, or service virtualization to replicate production.
- Mock or stub external systems as needed.
- Clean up data after each test run to avoid cross-test interference.

7. How do you implement load testing in a pipeline?
Integrate load testing tools (e.g., *Azure Load Testing*, *JMeter*, k6*)* into dedicated pipeline stages. Configure thresholds for performance metrics, and fail the pipeline if critical metrics (latency, errors) exceed limits.

---

8. What tools are commonly used for each test type in Azure DevOps?

- Unit: *MSTest*, *NUnit*, *xUnit*, *Jest*, *pytest*
- Integration: *Postman*, *REST Assured*, *SpecFlow*, *TestContainers*
- Load: *Azure Load Testing*, *Apache JMeter*, *k6*, *Artillery*

---

9. How do you handle test failures in pipelines?
Fail the pipeline stage when critical tests fail.

- Configure alerts and notifications.
- Optionally, collect logs and artifacts for debugging.
- Triage failures quickly to maintain build quality.

---

11. **How do you ensure code coverage thresholds are enforced during testing?**
    - Use code coverage tools (e.g., *Coverlet*, *JaCoCo*, *Istanbul*) in the pipeline.
    - Set minimum coverage thresholds in pipeline definitions; fail the build if coverage falls below.