Summary on Performance Testing

Module Summary

PW SKILLS

1. Load Testing Tools (JMeter, Gatling, Locust, etc)

- **JMeter:** An open-source load testing tool for measuring the performance of web applications, APIs, and services. It simulates multiple users, supports various protocols (HTTP, FTP, etc.), and provides comprehensive reports.
- **Gatling:** A powerful tool for load and stress testing, especially for web applications. It's written in Scala and offers high scalability and detailed metrics with a user-friendly web interface.
- **Locust:** A Python-based load testing tool that allows you to define user behavior in code. It's ideal for testing how many concurrent users a system can handle by simulating thousands of users.

2. Stress Testing and Soak Testing

- **Stress Testing:**

Tests the system beyond its normal capacity to identify breaking points or system limits. The goal is to see how the system behaves under extreme conditions, such as high traffic or resource exhaustion.

- **Soak Testing:**

Tests system performance over an extended period to check its stability and reliability. It focuses on detecting memory leaks, performance degradation, or other long-term issues that may arise after prolonged use.

3. Performance Testing in CI/CD Pipelines

Integrating performance testing into CI/CD pipelines helps catch performance bottlenecks early. This process ensures that new code changes do not degrade system performance. It usually involves:

- Running automated performance tests (e.g., load or stress tests) after every build or deployment.
- Setting up threshold limits so that the pipeline fails if the performance degrades beyond acceptable levels.
- Using tools like JMeter, Gatling, or Locust integrated with CI tools like Jenkins, GitLab CI, or CircleCI.

4. Performance Testing Best Practices

- **Start Small:** Begin with baseline tests and gradually increase the load to simulate real-world scenarios.
- **Use Realistic Data:** Ensure test data resembles production data to simulate real conditions.
- **Test Early and Regularly:** Integrate performance testing into development and CI/CD cycles to catch issues early.
- **Monitor System Resources:** Keep an eye on CPU, memory, disk I/O, and network usage during tests to identify bottlenecks.
- **Simulate Peak Load and Stress Conditions:** Include stress tests to check system resilience under peak loads.
- **Analyze Results Thoroughly:** Investigate response times, error rates, throughput, and resource consumption to understand performance behavior.