Module 1

1. Choose an appropriate automation tool for the build phase and justify its fit.
2. Match DevOps practices with corresponding Agile values.
3. Demonstrate the integration flow from code commit to deployment using DevOps stages.
4. Illustrate typical transitions observed in teams shifting from Agile to DevOps.
5. Examine the shift in developer responsibilities with DevOps adoption.
6. Apply DevOps principles to reduce lead time in a release cycle scenario.
7. Construct a deployment workflow incorporating continuous testing and integration.
8. Justify the role of cloud-native practices within DevOps environments.
9. Analyze the coordination between DevOps engineers and QA in a microservices project.
10. Outline core values linked to DevOps philosophy.
11. List stages in a standard DevOps pipeline.
12. State primary goals achieved through DevOps implementation.
13. Identify tools commonly used in continuous monitoring.
14. Classify deployment models aligned with DevOps in cloud environments.
15. Contrast DevOps lifecycle with traditional release cycles across infrastructure and testing phases.
16. Design a secure and scalable DevOps pipeline integrating containerization and orchestration tools.
17. Synthesize strategies to align DevOps with environmental sustainability in IT operations.
18. Recommend role-specific responsibilities within a multidisciplinary DevOps implementation team.
19. Evaluate performance enhancements in a system after introducing DevOps practices.
20. Create a modular DevOps framework for continuous delivery in a banking application.
21. Formulate an ecosystem incorporating tools, practices, and policies for enterprise DevOps transformation.
22. Define DevOps. How does it differ from traditional software development?
23. What are the key objectives and benefits of adopting DevOps practices?
24. Explain the relationship between Agile and DevOps.
25. List and briefly describe the key stages of the DevOps lifecycle.
26. What challenges can organizations face while implementing DevOps?

Module 2

1. Define actions performed in version control branching.
2. Name two benefits achieved through automated configuration scripts.
3. List major stages involved in continuous integration pipelines.
4. Recognize suitable file types managed by version control tools.
5. Classify container orchestration platforms based on feature sets.
6. Identify matching tools for each DevOps phase in a pipeline.
7. Suggest an open-source tool for log aggregation during monitoring.
8. Describe the steps in establishing a version control workflow in a team.
9. Analyze the transition from traditional deployment to continuous methods.
10. Design a toolchain that connects integration, testing, and deployment.
11. Evaluate configuration drift in cloud-managed environments.
12. Illustrate stages of container lifecycle and isolation benefits.
13. Choose tools for implementing configuration management in hybrid infra.
14. Map each monitoring layer with appropriate feedback metrics.
15. Build a version control structure for a distributed DevOps team with branching logic.
16. Critique pipeline failures in CI/CD systems and trace root causes with examples.
17. Compare stateless vs stateful containerized deployments in production systems.
18. Design a continuous testing strategy using multiple testing frameworks.
19. Evaluate resource consumption patterns from continuous monitoring dashboards.
20. Compose a tool pipeline integrating SCM, CI, CT, CD, containerization, and monitoring.
21. Recommend policies to secure configuration files and prevent unauthorized modifications.

Module 3

1. List two characteristics of distributed version control systems.
2. State two industry use cases where Git enhances code management.
3. Identify two key differences between Git and CVS.
4. Match Git commands with their respective functions in a repository workflow.
5. Classify commands under local and remote operations in Git.
6. Select branching models suited for collaborative development.
7. Point out one feature that distinguishes Mercurial from Git.
8. Describe Git's role in modern software development pipelines.
9. Compare CVCS and DVCS in terms of scalability and collaboration.
10. Demonstrate the usage of Git commands to clone, commit, push, and pull from a repository.
11. Analyze a use case involving Git branching strategy in a feature-release cycle.
12. Organize a table comparing Git workflows like Git Flow, GitHub Flow, and Forking Workflow.
13. Illustrate the sequence of Git operations involved in working with remote repositories.
14. Distinguish Git from Mercurial based on branching, ease of use, and platform integration.
15. Construct a branching and merging strategy using Git for a multi-developer project.
16. Evaluate the pros and cons of CVCS and DVCS architectures for enterprise applications.
17. Recommend a suitable Git workflow for a fast-paced startup environment with justification.
18. Create a comparison report between Git, Mercurial, and CVS based on key DevOps requirements.
19. Create a quick reference guide for beginners to effectively use Git repositories and manage branches.
20. Interpret error messages from Git CLI and propose resolution strategies for each.
21. Formulate a migration plan to switch from CVS to Git in a legacy enterprise system.

Module 4

1. List core activities performed during a Jenkins build job.
2. State key benefits achieved through automation in CI/CD pipelines.
3. Identify two CI tools other than Jenkins used in industry environments.
4. Match CI/CD stages with corresponding DevOps goals.
5. Select configuration options used in Jenkins job scheduling.
6. Classify build failure types observed in Jenkins console output.
7. Choose steps involved in integrating Git repositories with Jenkins.
8. Explain stages of a Jenkins-managed CI/CD pipeline using a real-world use case.
9. Analyze the impact of continuous integration on software release frequency.
10. Illustrate Jenkins architecture including master-agent model and plugin system.
11. Compare scheduling strategies for running CI jobs during development and release cycles.
12. Construct a Jenkins job configuration involving Git-based trigger and post-build steps.
13. Evaluate the reliability of Jenkins builds through reporting plugins and test logs.
14. Distinguish continuous deployment from continuous delivery based on risk management.
15. Build a CI/CD pipeline using Jenkins for a multi-module application including GitHub integration.
16. Evaluate Jenkins capabilities in reducing integration errors during parallel development.
17. Compose a plan to automate test and deployment jobs using Jenkins pipeline DSL and cron triggers.
18. Compare Jenkins with other CI tools in terms of plugin ecosystem, scalability, and security.
19. Formulate a troubleshooting guide for failed builds in a CI/CD pipeline involving Git and Jenkins.
20. Justify the role of CI/CD in increasing developer productivity and product stability.
21. Design a Jenkinsfile script supporting branching strategies and conditional build triggers.

Module 5

1. Select configuration parameters used during container creation.
2. Compare containers and virtual machines based on resource usage.
3. Construct a Dockerfile for deploying a multi-tier web application using custom base images.
4. Analyze resource isolation in containers using Linux namespaces and cgroups.
5. Match Dockerfile commands with corresponding container build actions.
6. Demonstrate the steps for creating a custom container using a Dockerfile.
7. Identify file types produced during Docker image build processes.
8. Evaluate Docker's role in simplifying software delivery across different environments.
9. Recommend a container management solution for automating lifecycle operations across OS variants.
10. Outline tasks involved in deploying a Java web application using Docker.
11. Identify two benefits that containerization offers in DevOps practices.
12. List components involved in the container lifecycle.
13. Describe the process of container instantiation using Docker CLI.
14. Construct a Dockerfile for deploying a multi-tier web application using custom base images.
15. Distinguish features of Docker images and containers with practical scenarios.
16. Compare Docker with legacy deployment methods using parameters like overhead, portability, and setup.
17. Analyze performance differences observed between containerized and virtualized systems.
18. Choose appropriate Docker instructions to optimize image layering.
19. Name two Docker platforms used in modern infrastructure.
20. Design a scalable container-based architecture for microservice-based application deployment.
21. Develop a stepwise deployment plan for running Java services inside Docker containers.
22. Examine roles of container registries in continuous deployment pipelines.

Module 6

1. List two phases involved in the agile testing life cycle.
2. Identify two tools commonly used for software configuration management.
3. Mention commands used to execute a test case in Selenium.
4. Match provisioning tools (Chef, Ansible, Puppet) with their configuration styles.
5. Select appropriate testing approach for a continuously integrated web application.
6. Point out provisioning script components in a Chef recipe.
7. Classify configuration management tools based on agent/agentless operation.
8. Explain how continuous testing integrates with DevOps feedback loops.
9. Analyze the execution flow of Selenium in automating UI testing for a login page.
10. Illustrate the structure and features of a Puppet manifest with an example.
11. Evaluate differences in configuration approaches of Ansible and Chef with real cases.
12. Demonstrate test automation strategy for a continuously deployed e-commerce platform.
13. Compare manual testing life cycle with continuous testing practices in agile delivery.
14. Construct a use case showcasing end-to-end provisioning using Ansible.
15. Design a test automation framework using Selenium for a multi-form web application.
16. Evaluate the effectiveness of configuration management tools in reducing deployment risks.
17. Create a testing life cycle model incorporating automation tools for continuous feedback.
18. Analyze the provisioning pipeline using Chef in a scalable multi-node deployment environment.
19. Recommend a toolchain combining testing, provisioning, and configuration management in DevOps.
20. Build a configuration-as-code setup using Ansible to provision a LAMP stack for production deployment.
21. Justify the transition from traditional configuration scripting to tool-based provisioning models.