Phases of DevOps Lifecycle Basics

Total points 18/20

MCQs
10 one mark questions and 5 two mark questions for a total of 20 marks. Duration of the quiz 30 minutes

The respondent's email (**divya29112004@gmail.com**) was recorded on submission of this form.

0 of 0 points

Regd. No. *

22B91A6253

Name *

U DIVYA

Branch *

- (●) CSE
- ( ) IT
- ( ) AIML
- ( ) CSBS

Assessment 18 of 20 points

✓ In the Operate phase, which practice best demonstrates a DevOps mindset focused on reliability and shared ownership? *2/2

- ○ A. Keeping operations responsibilities isolated to a dedicated ops team
- ○ B. Relying solely on reactive incident handling without defined service-level objectives
- ◉ C. Implementing Site Reliability Engineering practices with shared on-call rotations ✓
- ○ D. Allowing developers to deploy but forbidding them from accessing production metrics

✓ How does the continuous feedback loop between Monitor and Plan phases help refine the DevOps lifecycle over time? *1/1

- ◉ A. By feeding production insights into planning to reprioritize work, improve reliability, and refine assumptions ✓
- ○ B. By using monitoring data solely to prove that service-level objectives are always met
- ○ C. By using monitoring only to identify which team member caused an incident
- ○ D. By treating monitoring results as operational noise separate from product decisions

✓ In the Deploy phase, how can a DevOps team best ensure that infrastructure changes are consistent, auditable, and repeatable across environments? *1/1

- A. Applying configuration changes manually on each server as needed
- B. Relying on operations staff to remember deployment steps from documentation
- C. Using infrastructure as code with version control, automated validation, and immutable artifacts ✓
- D. Cloning production servers manually to create new environments when required

✗ Which feedback mechanism best connects the Operate and Plan phases to continuously refine reliability strategies across the DevOps lifecycle? *···/2

- A. Collecting incident reports but not systematically analyzing them
- B. Annual reliability reviews that produce a static runbook for the next year
- C. Tracking only infrastructure cost metrics and optimizing for the lowest spend
- D. Maintaining a live error-budget dashboard integrated with planning tools, influencing prioritization of reliability work vs. new features ✗

No correct answers

✓ In the Code phase, a team wants to reduce integration issues and improve collaboration across multiple microservices. Which practice most strongly supports a DevOps-first approach to coding? *1/1

A.

- (●) Adopting trunk-based development with frequent small merges and automated checks ✓
- ( ) Separating ops-related code into a different repository to avoid coupling
- ( ) Mandating that all developers commit directly to the main branch without any checks
- ( ) Maintaining long-lived feature branches that are merged only at the end of a release cycle

✓ In the Monitor phase, which approach most effectively supports continuous improvement across the DevOps lifecycle? *1/1

- (●) A. Implementing centralized observability with metrics, logs, and traces tied to business outcomes ✓
- ( ) B. Monitoring solely infrastructure metrics without including user experience signals
- ( ) C. Relying on basic server uptime checks without application-level monitoring
- ( ) D. Collecting logs and metrics only when a major incident occurs

✓ In the Test phase, what is the most DevOps-aligned way to ensure that tests effectively protect production reliability without blocking fast delivery? *1/1

- ○ A. Postponing most testing to the staging environment to keep earlier stages fast
- ○ B. Running a large, manual regression test only before major releases
- ◉ C. Automating a layered test strategy with fast checks early and targeted slower tests later in the pipeline ✓
- ○ D. Relying solely on unit tests and skipping integration and end-to-end tests

✓ In the Build phase, a DevOps team is experiencing inconsistent builds across developer machines and the CI server. Which strategy best addresses this issue in a mature DevOps lifecycle? *1/1

- ○ A. Allowing each microservice team to choose its own unique build tools with no constraints
- ○ B. Running builds only on the CI server and never locally to avoid discrepancies
- ○ C. Relying on developers to manually configure their environments to match the CI server
- ◉ D. Using containerized build environments defined as code and reused across all stages ✓

✓ During the Code, Build, and Test phases, a team notices that deployment frequency is high but change failure rate is also increasing. Which adjustment across the DevOps lifecycle is most appropriate? *1/1

- A. Removing automated tests that take longer than a few seconds to speed up the pipeline
- B. Slowing down deployments indefinitely to reduce risk
- C. Adding targeted automated tests and improving observability around high-risk areas before release ✓
- D. Delegating all deployment decisions to a central change advisory board

✓ Which practice most effectively closes the loop between Monitor and Test phases to prevent regression of resolved incidents? *2/2

- A. Automatically generating or updating regression tests from incident postmortems and linking them to monitoring alerts ✓
- B. Logging incidents in a ticketing system without further action
- C. Running only synthetic uptime checks without validating business flows
- D. Relying on tribal knowledge so engineers remember to add tests after major outages

✓ During the Test phase, which approach best supports continuous, production-like validation that directly informs the Deploy and Operate phases? *2/2

- ○ A. Running nightly end-to-end tests against a shared staging environment that loosely resembles production
- ○ B. Using contract tests only at the service boundary and ignoring cross-service workflows
- ◉ C. Implementing environment-agnostic, declarative test suites plus synthetic and chaos tests executed automatically in pre-production and selectively in production ✓
- ○ D. Relying solely on unit tests with very high code coverage

✓ A team wants to better connect the Plan, Code, and Operate phases by defining meaningful metrics. Which set of metrics best supports a holistic DevOps lifecycle? *1/1

- ○ A. Focusing solely on hardware utilization and infrastructure cost metrics
- ○ B. Only tracking lines of code written and number of commits per developer
- ○ C. Measuring only the number of features delivered per quarter
- ◉ D. Using a combination of flow metrics (like deployment frequency), reliability metrics (like error rates), and customer-centric metrics (like task success) ✓

✓ During the Release phase, a team wants to minimize risk while deploying frequently. Which release strategy best aligns with advanced DevOps practices? *2/2

- (●) A. Using blue-green or canary releases combined with feature flags ✓
- ( ) B. Performing big-bang releases with many features bundled together
- ( ) C. Allowing teams to deploy directly to production without any release governance
- ( ) D. Freezing the production environment for long periods before each release

✓ During the Code phase, which strategy most effectively prepares artifacts for later high-frequency, low-risk deployments in the Release and Deploy phases? *1/1

- ( ) A. Deferring non-functional requirements like performance and security to a hardening sprint
- (●) B. Embedding feature flags and trunk-based development with strict automated quality gates ✓
- ( ) C. Requiring manual exploratory testing as the primary quality control measure
- ( ) D. Mandating long-lived feature branches that are merged only before major releases

✓ During the Plan phase of the DevOps lifecycle, a team is deciding on work items for an upcoming quarter. Which practice best aligns with a high-maturity DevOps approach to planning? *1/1

- (●) Using a dynamic, prioritized product backlog with frequent feedback-driven refinement ✓
- ( ) Locking down a fixed scope for the whole quarter and disallowing changes
- ( ) Creating a detailed Gantt chart with all dependencies fixed upfront
- ( ) Letting each developer choose tasks independently without shared prioritization