KaozhengPro

IT認證考試題庫

專業平臺

考證寶提供最新考古題與模擬試題
協助您高效通過認證考試

www.kaozhengpro.com

Exam : **CNPA**

Title : Certified Cloud Native Platform Engineering Associate

Version : DEMO

1.What is the goal of automating processes in platform teams?
A. Reducing time spent on repetitive tasks.
B. Focusing on manual processes.
C. Increasing the number of tasks completed.
D. Ensuring high-quality coding standards.
Answer: A
Explanation:
Comprehensive and Detailed Explanation at least 150 to 200 words:
In platform engineering, automation's primary goal is to eliminate manual, repetitive toil by codifying repeatable workflows and guardrails so teams can focus on higher-value work. Authoritative Cloud Native Platform Engineering guidance emphasizes that platforms should provide consistent, reliable, and secure self-service capabilities—achieved by automating provisioning, configuration, policy enforcement, and delivery pipelines. This directly reduces cognitive load and handoffs, shortens lead time for changes, decreases error rates, and improves overall reliability. While automation often improves code quality indirectly (e.g., through automated testing, linting, and policy-as-code), the central, explicitly stated aim is to remove repetitive manual work and standardize operations, not to simply "do more tasks" or prioritize manual intervention. Therefore, option A most accurately captures the intent.
Options B and C misframe the objective: platform engineering seeks fewer manual steps and better outcomes, not just higher task counts.
Option D is a beneficial consequence but not the core purpose. By systematizing common paths ("golden paths") and embedding security and compliance controls into automated workflows, platforms deliver predictable, compliant environments at scale while freeing engineers to focus on product value.
Reference:
— CNCF Platforms Whitepaper (Platform Engineering)
— CNCF Platform Engineering Maturity Model
— Cloud Native Platform Engineering Study Guide

2.Which of the following strategies should a team prioritize to enhance platform efficiency?
A. Encourage teams to handle all platform tools independently without guidance.
B. Implement manual updates for all cluster configurations.
C. Automate the version bump process (or cluster updates).
D. Conduct weekly meetings to discuss every minor update.
Answer: C
Explanation:
Comprehensive and Detailed Explanation at least 150 to 200 words:
Enhancing platform efficiency requires reducing operational friction and ensuring that updates, patches, and upgrades happen consistently without introducing unnecessary manual effort or delays. According to Cloud Native Platform Engineering practices, automation of the version bump process—whether for libraries, services, or cluster configurations—is a critical strategy for improving both reliability and security. By automating cluster updates, teams can minimize human error, enforce standardized practices, and ensure systems remain aligned with compliance and security benchmarks.
Option A, where each team independently manages platform tools, increases fragmentation and cognitive load, ultimately reducing efficiency.
Option B, relying on manual updates, is both error-prone and unsustainable at scale, particularly in

environments with multiple clusters or microservices.
Option D, holding frequent meetings to discuss minor updates, wastes engineering cycles without delivering the tangible improvements that automation can achieve.
Automating updates is a direct application of Infrastructure as Code and GitOps principles, enabling declarative management, reproducibility, and consistent rollout strategies. Additionally, automation supports zero-downtime upgrades, aligns with cloud native resilience patterns, and improves developer experience by abstracting away operational complexity. Thus, option C represents the most effective strategy for enhancing platform efficiency.
Reference:
— CNCF Platforms Whitepaper (Platform Engineering)
— CNCF GitOps Principles for Platforms
— Cloud Native Platform Engineering Study Guide

3.In a multi-cluster Kubernetes setup, which approach effectively manages the deployment of multiple interdependent applications together as a unit?
A. Employing a declarative application deployment definition.
B. Creating separate Git repositories per application.
C. Direct deployments from CI/CD with Git configuration.
D. Using Helm for application packaging with manual deployments.
Answer: A
Explanation:
In multi-cluster Kubernetes environments, the challenge lies in consistently deploying interdependent applications across clusters while ensuring reliability and repeatability. The Cloud Native Platform Engineering guidance stresses the importance of a declarative approach to define applications as code, which enables teams to describe the entire application system—including dependencies, configuration, and policies—in a single manifest. This ensures that applications are treated as a cohesive unit rather than isolated workloads.
Option A is correct because declarative application deployment definitions (often managed through GitOps practices) allow for consistent and automated reconciliation of desired state versus actual state across multiple clusters. This approach supports scalability, disaster recovery, and compliance by ensuring identical deployments across environments.
Option B (separate repos per application) increases fragmentation and does not inherently manage interdependencies.
Option C (direct deployments from CI/CD) bypasses the GitOps model, which reduces auditability and consistency.
Option D (Helm with manual deployments) partially addresses packaging but lacks the automation and governance needed in a multi-cluster setup.
Reference:
— CNCF GitOps Principles for Platforms
— CNCF Platforms Whitepaper
— Cloud Native Platform Engineering Study Guide

4.In the context of platform engineering and the effective delivery of platform software, which of the following statements describes the role of CI/CD pipelines in relation to Software Bill of Materials

(SBOM) and security scanning?
A. SBOM generation and security scanning are particularly valuable for application software. While platform software may have different security considerations, these practices are highly beneficial within CI/CD pipelines for applications.
B. CI/CD pipelines should integrate SBOM generation and security scanning as automated steps within the build and test phases to ensure early detection of vulnerabilities and maintain a clear inventory of components.
C. CI/CD pipelines are designed to accelerate the delivery of platform software, and adding SBOM generation and security scanning would slow down the process, so these activities are better suited for periodic audits conducted outside of the pipeline.
D. CI/CD pipelines are primarily for automating deployments; SBOM generation and security scanning are separate, manual processes performed after deployment.
Answer: B
Explanation:
Modern platform engineering requires security and compliance to be integral parts of the delivery process, not afterthoughts. CI/CD pipelines are the foundation for delivering platform software rapidly and reliably, and integrating SBOM generation and automated vulnerability scanning directly within pipelines ensures that risks are identified early in the lifecycle.
Option B is correct because it reflects recommended practices from cloud native platform engineering standards: SBOMs provide a transparent inventory of all software components, including dependencies, which is crucial for vulnerability management, license compliance, and supply chain security. By automating these steps in CI/CD, teams can maintain both velocity and security without manual overhead.
Option A downplays the relevance of SBOMs for platform software, which is inaccurate because platform components (like Kubernetes operators, ingress controllers, or logging agents) are equally susceptible to vulnerabilities.
Option C dismisses automation in favor of periodic audits, which contradicts the shift-left security principle.
Option D misunderstands CI/CD's purpose: security must be integrated, not separated.
Reference:
— CNCF Supply Chain Security Whitepaper
— CNCF Platforms Whitepaper
— Cloud Native Platform Engineering Study Guide

5.A developer is struggling to access the necessary services on a cloud native platform due to complex Kubernetes configurations.
What approach can best simplify their access to platform capabilities?
A. Increase the number of required configurations to enhance security.
B. Implement a web portal that abstracts the Kubernetes complexities.
C. Limit user access to only a few services.
D. Provide detailed documentation on Kubernetes configurations.
Answer: B
Explanation:
One of the primary objectives of internal developer platforms (IDPs) is to improve developer experience

by reducing cognitive load. Complex Kubernetes configurations often overwhelm developers who simply want to consume services and deploy code without worrying about infrastructure intricacies.
Option B is correct because implementing a self-service web portal (or developer portal) abstracts away Kubernetes complexities, providing developers with easy access to platform services through standardized workflows, templates, and golden paths. This aligns with platform engineering principles: empowering developers with self-service capabilities while maintaining governance, security, and compliance.
Option A increases burden unnecessarily and negatively impacts productivity.
Option C limits access to services, reducing flexibility and developer autonomy, which goes against the core goal of IDPs.
Option D, while helpful for education, does not remove complexity—it only shifts the responsibility back to the developer. By leveraging portals, APIs, and automation, platform teams allow developers to focus on building business value instead of managing infrastructure details.
Reference:
— CNCF Platforms Whitepaper
— Team Topologies and Platform Engineering Practices
— Cloud Native Platform Engineering Study Guide