Linux Foundation CNPA Practice Questions

Shared by Miranda on 18-08-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

In a cloud native environment, which approach is effective for managing resources to ensure a balance between defined states and dynamic adjustments?

Options:

A- Imperative Resource Management
B- Manual Resource Tracking
C- Declarative Resource Management
D- Static Resource Allocation

Answer:

C

Explanation:

Declarative resource management is a core principle in Kubernetes and cloud native platforms. Option C is correct because declarative systems define the desired state of resources (e.g., YAML manifests for Deployments, Services, or ConfigMaps), and controllers reconcile the actual state to match the desired state. This provides consistency, automation, and resilience, while also allowing dynamic adjustments like scaling.

Option A (imperative management) requires step-by-step commands, which are error-prone and not scalable. Option B (manual tracking) adds overhead and risk of drift. Option D (static allocation) wastes resources and does not adapt to changing workloads.

Declarative management enables GitOps workflows, automated scaling, and consistent application of policies. This approach aligns with platform engineering principles by combining automation with governance, enabling efficiency and reliability at scale.

--- CNCF GitOps Principles

--- Kubernetes Design Principles

--- Cloud Native Platform Engineering Study Guide

Question 2

Question Type: MultipleChoice

As a platform engineer, how do you automate application deployments across multiple Kubernetes clusters using GitOps, Helm, and Crossplane, ensuring a consistent application state?

Options:

A- Employ a GitOps controller to synchronize Git-stored Helm and Crossplane configurations.
B- Use Helm and Crossplane, with manual GUI-based configuration updates.
C- Integrate Helm and Crossplane into a GitOps-enabled CI/CD pipeline.
D- Leverage Git for configuration storage, with manual application of Helm and Crossplane.

Answer:

A

Explanation:

The most effective way to achieve consistent, automated deployments across multiple Kubernetes clusters is to combine GitOps controllers (e.g., Argo CD, Flux) with declarative configurations managed by Helm and Crossplane. Option A is correct because the GitOps controller continuously reconciles the desired state stored in Git---Helm charts for applications and Crossplane manifests for infrastructure---ensuring consistency across clusters.

Option B and D rely on manual updates, which are error-prone and not scalable. Option C mischaracterizes GitOps by suggesting push-based pipelines rather than the core GitOps model of pull-based reconciliation.

This combination leverages Helm for application packaging, Crossplane for cloud infrastructure provisioning, and GitOps for declarative, version-controlled delivery. It ensures applications remain in sync with Git, providing auditability, automation, and resilience in multi-cluster environments.

--- CNCF GitOps Principles

--- CNCF Platforms Whitepaper

--- Cloud Native Platform Engineering Study Guide

Question 3

Question Type: MultipleChoice

A platform engineering team needs to provide comprehensive cost visibility for Kubernetes workloads to optimize infrastructure utilization. Which tool is recommended to achieve this goal?

Options:

A- Application performance monitoring tools with limited resource cost tracking.
B- OpenCost for real-time, granular Kubernetes cost allocation and analysis.
C- Kubernetes resource usage metrics paired with cloud provider billing data.
D- Cloud provider cost estimation tools with basic Kubernetes integration.

Answer:

B

Explanation:

OpenCost is the CNCF-supported open-source project designed specifically for Kubernetes cost visibility and optimization. Option B is correct because OpenCost provides granular, real-time allocation of Kubernetes costs across namespaces, workloads, and teams. This allows organizations to understand true cost drivers and optimize resource utilization effectively.

Option A (APM tools) may track performance but usually lack deep integration with Kubernetes cost allocation. Option C provides partial visibility but requires complex manual correlation of resource usage with billing data. Option D (cloud provider estimators) typically offer limited or high-level insights and do not map costs down to Kubernetes workloads.

By adopting OpenCost, platform teams can align financial accountability with engineering usage, a practice known as FinOps. This supports sustainable scaling, cost efficiency, and transparency---critical aspects of measuring platform success.

--- CNCF OpenCost Project

--- CNCF Platforms Whitepaper

--- Cloud Native Platform Engineering Study Guide

Question 4

Question Type: MultipleChoice

A company is implementing a service mesh for secure service-to-service communication in their cloud native environment. What is the main benefit of using mutual TLS (mTLS) within this context?

Options:

A- Allows services to authenticate each other and secure data in transit.
B- Allows services to bypass security checks for better performance.
C- Enables logging of all service communications for audit purposes.
D- Simplifies the deployment of microservices by automatically scaling them.

Answer:

A

Explanation:

Mutual TLS (mTLS) is a core feature of service meshes, such as Istio or Linkerd, that enhances security in cloud native environments by ensuring that both communicating services authenticate each other and that the communication channel is encrypted. Option A is correct because mTLS delivers two critical benefits: authentication (verifying the identity of both client and server services) and encryption (protecting data in transit from interception or tampering).

Option B is incorrect because mTLS does not bypass security---it enforces it. Option C is partly true in that service meshes often support observability and logging, but that is not the primary purpose of mTLS. Option D relates to scaling, which is outside the scope of mTLS.

In platform engineering, mTLS is a fundamental security mechanism that provides zero-trust networking between microservices, ensuring secure communication without requiring application-level changes. It strengthens compliance with security and data protection requirements, which are crucial in regulated industries.

--- CNCF Service Mesh Whitepaper

--- CNCF Platforms Whitepaper

--- Cloud Native Platform Engineering Study Guide

Question 5

Question Type: MultipleChoice

A software development team is struggling to adopt a new cloud native platform efficiently. How

can a centralized developer portal, such as Backstage, help improve their adoption process?

Options:

A- Provides a single access point for all platform services and documentation.
B- Provides tutorials on unrelated programming languages.
C- Offers a place for developers to share their personal projects and code snippets.
D- Limits access to platform tools to only senior developers.

Answer:

A

Explanation:

Developer portals like Backstage act as the single entry point for platform services, APIs, golden paths, and documentation. Option A is correct because centralizing access greatly reduces the friction developers face when trying to adopt a new platform. Instead of searching across fragmented systems or learning low-level Kubernetes details, developers can find everything in one place, including templates, service catalogs, automated workflows, and governance policies.

Option B is irrelevant to platform adoption. Option C may foster community sharing but does not directly address adoption challenges. Option D contradicts platform engineering principles, which emphasize democratizing access and self-service rather than restricting tools to senior developers.

By providing a unified experience, portals improve discoverability, consistency, and self-service. They reduce cognitive load and support the platform engineering principle of improving developer experience, making adoption of new platforms smoother and more efficient.

--- CNCF Platforms Whitepaper

--- CNCF Platform Engineering Maturity Model

--- Cloud Native Platform Engineering Study Guide

Question 6

Question Type: MultipleChoice

A platform team is deciding whether to invest engineering time into automating cluster autoscaling. Which of the following best justifies making this automation a priority?

Options:

A- Cluster autoscaling is a repetitive task that increases toil when done manually.
B- Manual upgrade tasks help platform teams stay familiar with system internals.
C- Most engineers prefer doing upgrade tasks manually and prefer to review each one.
D- Automation tools are better than manual processes, regardless of context.

Answer:

A

Explanation:

Automation in platform engineering is primarily about reducing repetitive manual work, or toil, which consumes engineering capacity and increases the risk of human error. Option A is correct because cluster autoscaling---adjusting resources to meet workload demand---is a repetitive, ongoing task that is better handled through automation. Automating this process ensures scalability, efficiency, and reliability while freeing platform teams to focus on higher-value work.

Option B may provide learning opportunities but is not a sustainable justification. Option C is subjective and inefficient, while Option D is overly broad---automation should be applied thoughtfully to tasks that bring measurable benefits.

Automating autoscaling aligns with cloud native best practices, ensuring workloads can respond elastically to demand changes while maintaining cost efficiency. This reduces manual overhead, improves resiliency, and supports the developer experience by ensuring resource availability.

--- CNCF Platforms Whitepaper

--- SRE Principles on Eliminating Toil

--- Cloud Native Platform Engineering Study Guide

Question 7

Question Type: MultipleChoice

Which of the following would be considered an advantage of using abstract APIs when offering cloud service provisioning and management as platform services?

Options:

A- Abstractions enforce explicit platform team approval before any cloud resource is deployed.
B- Abstractions curate cloud services with built-in guardrails for development teams.
C- Abstractions allow customization of cloud services and resources without guardrails.
D- Development teams can arbitrarily deploy cloud services via abstractions.

Answer:

B

Explanation:

Abstract APIs are an essential component of platform engineering, providing a simplified interface for developers to consume infrastructure and cloud services without deep knowledge of provider-specific details. Option B is correct because abstractions allow platform teams to curate services with built-in guardrails, ensuring compliance, security, and operational standards are enforced automatically. Developers get the benefit of self-service and flexibility while the platform team ensures governance.

Option A would slow down the process, defeating the purpose of abstraction. Option C removes guardrails, which risks security and compliance violations. Option D allows uncontrolled deployments, which can create chaos and undermine platform governance.

Abstract APIs strike the balance between developer experience and organizational control. They provide golden paths and opinionated defaults while maintaining the flexibility needed for developer productivity. This approach ensures efficient service provisioning at scale with reduced cognitive load on developers.

--- CNCF Platforms Whitepaper

--- CNCF Platform Engineering Maturity Model

--- Cloud Native Platform Engineering Study Guide

Question 8

Question Type: MultipleChoice

In a GitOps workflow, what is a secure and efficient method for managing secrets within a Git repository?

Options:

A- Use environment variables to manage secrets outside the repository.
B- Use a secrets management tool and store references in the repository.
C- Encrypt secrets and store them directly in the repository.
D- Store secrets in plain text within the repository.

Answer:

B

Explanation:

The secure and efficient way to handle secrets in a GitOps workflow is to use a dedicated secrets management tool (e.g., HashiCorp Vault, Sealed Secrets, or External Secrets Operator) and store only references or encrypted placeholders in the Git repository. Option B is correct because Git should remain the source of truth for configuration, but sensitive values should be abstracted or encrypted to maintain security.

Option A (environment variables) can supplement secret management but lacks versioning and auditability when used alone. Option C (encrypting secrets in Git) can work with tools like Mozilla SOPS, but it still requires external key management, making Option B a more complete and secure approach. Option D (plain text secrets) is highly insecure and should never be used.

By integrating secrets managers into GitOps workflows, teams achieve both security and automation, ensuring secrets are delivered securely during reconciliation without exposing sensitive data in Git.

--- CNCF GitOps Principles

--- CNCF Supply Chain Security Whitepaper

--- Cloud Native Platform Engineering Study Guide