Kubernetes Certified Network Administrator (KCNA) Part 2 Practice Test (Sample)

Study Guide

Everything you need from our exam experts!

Table of Contents

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- **Practice answering questions under realistic conditions,**
- **Improve accuracy and speed,**
- **Review explanations to strengthen weak areas, and**
- **Approach the exam with greater confidence.**

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here’s how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don’t just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There’s no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

Questions

1. Which technology is incompatible with the Kubernetes Container Runtime Interface (CRI)?

A. VirtualBox

B. CRI-O

C. containerd

D. runc

2. Which kubectl command runs a command in an existing container?

A. kubectl attach

B. kubectl logs

C. kubectl exec

D. kubectl run

3. In Kubernetes, which object represents the endpoints backing a Service?

A. Pod

B. Endpoints

C. Service

D. Node

4. What Kubernetes resource provides a stable entry point and can load balance across a set of pods?

A. Deployment

B. Service

C. Ingress

D. StatefulSet

5. Which open-source storage platform provides integrated block, file, and object storage for distributed environments?

A. GlusterFS

B. OpenEBS

C. Ceph

D. Lustre

6. Which tool is specifically designed for monitoring and managing costs in a Kubernetes environment, providing insights into spending and resource optimisation?

A. Prometheus

B. Grafana

C. Kubecost

D. Kubernetes Dashboard

7. Docker multistage build technique reduces final image size.

A. Convert to multi-arch

B. Use smaller base images

C. Use multiple build stages to discard build artifacts and dependencies

D. Squash layers after build

8. Which two resources are commonly used together to expose applications to external traffic, including type options like ClusterIP, NodePort, and LoadBalancer?

A. Services and Ingress

B. Pods and Services

C. ConfigMaps and Secrets

D. Deployments and Daemons

9. What term describes the end-to-end journey of a request across services in a distributed system?

A. Logs

B. Metrics

C. Traces

D. Path tracking

10. Which tool is specifically designed as a comprehensive service mesh to control, secure, and observe microservices, including advanced traffic management features?

A. Linkerd

B. Consul

C. Istio

D. AWS App Mesh

Answers

1. A
2. C
3. B
4. B
5. C
6. C
7. C
8. A
9. C
10. C

Explanations

1. Which technology is incompatible with the Kubernetes Container Runtime Interface (CRI)?

A. VirtualBox

B. CRI-O

C. containerd

D. runc

Kubernetes' CRI defines a standard API that a container runtime must implement so kubelet can manage pods and containers. VirtualBox is a hypervisor for running entire virtual machines, not a container runtime, and it doesn't provide the CRI API. Because of that, it can't serve as the CRI backend for Kubernetes. In contrast, CRI-O and containerd are CRI-compliant runtimes designed to plug into Kubernetes via CRI, handling container lifecycle. Runc is a lightweight OCI runtime that runs the actual container processes and is used by the CRI implementations like containerd and CRI-O under the hood. So VirtualBox can't be used as the CRI backend, while the others fit into the CRI model in their respective roles.

2. Which kubectl command runs a command in an existing container?

A. kubectl attach

B. kubectl logs

C. kubectl exec

D. kubectl run

When you want to run a command inside a running container, you need a tool that executes inside the container's process space without creating something new. That's what kubectl exec does: it runs the specified command in a pod's existing container. You can make it interactive with -it, for example kubectl exec -it my-pod -- /bin/bash to get a shell, or run a single command like kubectl exec my-pod -- ls / to check a directory. If the pod has multiple containers, you can pick one with -c container-name. Attaching to a running container via kubectl attach doesn't start a new command; it connects your terminal to the main process's standard input/output streams. Viewing logs is done with kubectl logs, which prints container output but doesn't execute a new command. Running a new pod or container is handled by kubectl run, which creates a pod, not command execution inside an existing container.

3. In Kubernetes, which object represents the endpoints backing a Service?

A. Pod

B. Endpoints

C. Service

D. Node

The mapping from a Service to the actual back-end pods is held in an Endpoints object. When a Service selects pods by labels, Kubernetes populates Endpoints with the IP addresses and ports of those pods. This Endpoints data is what the cluster's proxy uses to route traffic to the correct pods. A Pod is the workload itself, not the representation of the back-end addresses for a Service. A Node is simply the host where pods run, not the set of addresses backing a Service. The Service provides a stable front end, while Endpoints lists the actual Pod endpoints behind it. (Note: EndpointSlices serve the same purpose in newer designs for scalability, but Endpoints is the traditional backing store.)

4. What Kubernetes resource provides a stable entry point and can load balance across a set of pods?

A. Deployment

B. Service

C. Ingress

D. StatefulSet

A Kubernetes Service provides a stable entry point and load-balances traffic across a set of pods. It exposes a fixed DNS name and IP, decoupled from the lifecycle of any individual Pod, and targets pods via a label selector. When requests reach the Service, kube-proxy distributes them among healthy backend pods, giving you simple, ongoing load balancing. The Service can be configured as ClusterIP (default, internal), NodePort (exposes on each node), or LoadBalancer (provisions an external load balancer in many cloud environments), aligning exposure with your needs. Ingress sits above Services for HTTP routing and usually relies on a Service as the backend, rather than providing the stable entry point on its own. StatefulSet focuses on stable Pod identities and storage rather than a generic load-balanced endpoint for a pod pool. Deployment manages pod lifecycles but does not by itself offer a stable entry point or load balancing across pods.

5. Which open-source storage platform provides integrated block, file, and object storage for distributed environments?

A. GlusterFS

B. OpenEBS

C. Ceph

D. Lustre

This question tests a storage platform that provides unified access to block, file, and object storage across a distributed cluster. Ceph delivers this by design: its core RADOS distributed object store underpins all formats, while additional interfaces cover the different storage needs. Block storage is exposed through RADOS Block Device (RBD), ideal for VM and container workloads. File storage comes from CephFS, a scalable POSIX-compatible filesystem. Object storage is provided by the RADOS Gateway (RGW), offering S3 and Swift-compatible interfaces. Together, these capabilities let a single Ceph cluster serve multiple storage workloads in distributed environments. GlusterFS is primarily a distributed file system, not a built-in block and object storage suite. OpenEBS focuses on Kubernetes-native block storage rather than file or object storage, and Lustre is a high-performance parallel file system for HPC environments, not a unified block/file/object platform.

6. Which tool is specifically designed for monitoring and managing costs in a Kubernetes environment, providing insights into spending and resource optimisation?

A. Prometheus

B. Grafana

C. Kubecost

D. Kubernetes Dashboard

Cost visibility and optimization in Kubernetes is about tying resource usage to actual spend and making informed decisions to reduce waste. Kubecost is built for this purpose: it provides cost allocation and optimization insights across your cluster, broken down by namespace, deployment, label, service, or workload, and it can show you spend across cloud providers and clusters. Kubecost pulls together cloud billing data with Kubernetes usage to give real-time and historical views of cost, plus budgets, alerts, and forecasting. It helps you spot waste—like over-provisioned or idle resources—and offers practical optimization guidance, such as right-sizing resources or adjusting schedules. While Prometheus collects metrics, Grafana visualizes them, and the Kubernetes Dashboard offers cluster management UI, none of these focus on cost accounting and optimization. Kubecost specifically targets that cost perspective, making it the best fit for monitoring and managing Kubernetes expenses.

7. Docker multistage build technique reduces final image size.

A. Convert to multi-arch

B. Use smaller base images

C. Use multiple build stages to discard build artifacts and dependencies

D. Squash layers after build

Using multiple build stages lets you separate the environment used to build the application from the environment that runs it. In the first stage you install compilers, build tools, and other dependencies needed to compile or assemble the app. In a later stage you start from a minimal runtime base and copy only the final, built artifacts into it. This means all the build tools, caches, and intermediate files stay out of the final image, so the runtime image is much smaller. For example, you build with a full-featured image in the first stage, then use a slim or specialized runtime image in the final stage and transfer only what's needed (the compiled binary, libraries it truly depends on, and the application files). This approach directly targets reducing the final image size by excluding unnecessary build-time content. Other options touch related ideas but don't achieve the same guaranteed reduction. Converting to multi-arch is about supporting multiple CPU architectures, not about shrinking the image. Using smaller base images helps, but if you still carry over build tools and caches, the final size won't shrink as much. Squashing layers can reduce the number of layers or, in some cases, the apparent size, but it doesn't reliably remove build-time artifacts and can complicate caching and reproducibility.

8. Which two resources are commonly used together to expose applications to external traffic, including type options like ClusterIP, NodePort, and LoadBalancer?

A. Services and Ingress

B. Pods and Services

C. ConfigMaps and Secrets

D. Deployments and Daemons

The key idea is how Kubernetes exposes applications to outside traffic. A Service provides a stable network endpoint for a set of pods, and it can be configured with types like ClusterIP (internal), NodePort (opens a port on each node), or LoadBalancer (integrates with a cloud LB to expose it externally). An Ingress sits in front of one or more Services and defines rules for routing external HTTP(S) requests to the appropriate Service backend, often with features like host-based or path-based routing and TLS termination. In practice, you typically pair a Service with an Ingress: the Service handles the actual pods behind a stable address, and the Ingress provides flexible, externally accessible routing to those Services. An Ingress Controller is needed to implement the Ingress rules. Other resources like ConfigMaps/Secrets or workload controllers (Deployments/Daemons) aren't used primarily for exposing external traffic, so they don't fit this pairing.

9. What term describes the end-to-end journey of a request across services in a distributed system?

A. Logs

B. Metrics

C. Traces

D. Path tracking

Tracing captures the end-to-end journey of a request as it moves through multiple services. It attaches a unique trace identifier to the request and propagates it across all services, with each service recording a span that marks its work, start and end times, and metadata. The collection of these spans forms a trace, revealing the full path the request took and where time is spent, so you can understand dependencies, latency hotspots, and failures in a distributed system. Logs are individual events from a single service, not the whole path; metrics are aggregated numbers describing performance at a higher level; path tracking isn't the standard term used for this end-to-end view.

10. Which tool is specifically designed as a comprehensive service mesh to control, secure, and observe microservices, including advanced traffic management features?

A. Linkerd

B. Consul

C. Istio

D. AWS App Mesh

A comprehensive service mesh that controls, secures, and observes microservices relies on a platform that ties together secure communication, policy enforcement, and rich traffic management through sidecar proxies. Istio fits this description best. It couples Envoy proxies with a robust control plane, enabling end-to-end mutual TLS, strong identity management, and centralized policy enforcement across services. On top of security, it offers advanced traffic management capabilities such as fine-grained routing (including canary and weighted deployments), fault injection, retries, timeouts, and circuit breakers, which are essential for resilient microservice architectures. Beyond traffic control, Istio provides deep observability with metrics, traces, and logs, giving you visibility into service behavior and performance. This visibility, combined with security and policy features, makes it a powerful, feature-rich choice for managing microservices in complex environments. While other options like Linkerd emphasize simplicity and performance, Consul focuses more on service discovery and broader mesh features, and AWS App Mesh is a managed provider-specific option, Istio remains the most widely recognized for its comprehensive, feature-rich approach to service mesh functionality.

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://kcnapt2.examzify.com

We wish you the very best on your exam journey. You've got this!