KCNA - Cloud Native Architecture

Cloud Native Architecture (16%)

Download PDF Version

This domain covers cloud native principles, design patterns, and the CNCF ecosystem.

What is Cloud Native?

Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds.

CNCF Definition

> "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach."

Cloud Native Principles

The Twelve-Factor App

Factor	Description
1. Codebase	One codebase tracked in revision control
2. Dependencies	Explicitly declare and isolate dependencies
3. Config	Store config in the environment
4. Backing Services	Treat backing services as attached resources
5. Build, Release, Run	Strictly separate build and run stages
6. Processes	Execute the app as stateless processes
7. Port Binding	Export services via port binding
8. Concurrency	Scale out via the process model
9. Disposability	Maximize robustness with fast startup and graceful shutdown
10. Dev/Prod Parity	Keep development, staging, and production as similar as possible
11. Logs	Treat logs as event streams
12. Admin Processes	Run admin/management tasks as one-off processes

Microservices Architecture

Monolith vs Microservices

Aspect	Monolith	Microservices
Deployment	Single unit	Independent services
Scaling	Scale entire app	Scale individual services
Technology	Single stack	Polyglot
Team Structure	Large teams	Small, autonomous teams
Failure Impact	Entire app affected	Isolated failures

Microservices Characteristics

- **Single Responsibility**: Each service does one thing well
- **Independently Deployable**: Services can be deployed without affecting others
- **Decentralized Data**: Each service manages its own data
- **Smart Endpoints, Dumb Pipes**: Logic in services, simple communication
- **Design for Failure**: Services handle failures gracefully

CNCF Landscape

Project Maturity Levels

Level	Description	Examples
Graduated	Production-ready, widely adopted	Kubernetes, Prometheus, Envoy
Incubating	Growing adoption, maturing	Argo, Cilium, Flux
Sandbox	Early stage, experimental	New projects

Key CNCF Projects by Category

Container Runtime

- **containerd** - Industry-standard container runtime

- **CRI-O** - Lightweight runtime for Kubernetes

Orchestration

- **Kubernetes** - Container orchestration platform

Service Mesh

- **Istio** - Connect, secure, control, and observe services
- **Linkerd** - Ultralight service mesh
- **Envoy** - Edge and service proxy

Observability

- **Prometheus** - Monitoring and alerting
- **Grafana** - Visualization and dashboards
- **Jaeger** - Distributed tracing
- **OpenTelemetry** - Observability framework

CI/CD

- **Argo** - GitOps continuous delivery
- **Flux** - GitOps toolkit
- **Tekton** - Cloud-native CI/CD

Networking

- **Cilium** - eBPF-based networking
- **Calico** - Network policy engine
- **CoreDNS** - DNS server

Storage

- **Rook** - Storage orchestration
- **Longhorn** - Distributed block storage

Security

- **Falco** - Runtime security
- **OPA** - Policy engine
- **cert-manager** - Certificate management

Serverless

What is Serverless?

Serverless computing allows you to build and run applications without managing servers. The cloud provider automatically provisions, scales, and manages the infrastructure.

Serverless Platforms

- **AWS Lambda**
- **Google Cloud Functions**
- **Azure Functions**
- **Knative** (Kubernetes-native)

Function as a Service (FaaS)

```
 Event → Function → Response

Examples:
- HTTP request triggers function
- Message queue triggers function
- Scheduled event triggers function
```

Service Mesh

What is a Service Mesh?

A dedicated infrastructure layer for handling service-to-service communication, providing:

- **Traffic Management**: Load balancing, routing
- **Security**: mTLS, authentication
- **Observability**: Metrics, tracing, logging

Service Mesh Architecture

```
 ┌─────────────────────────────────────────┐
│              Control Plane               │
│  (Configuration, Policy, Certificates)   │
└─────────────────────────────────────────┘
                    │
    ┌───────────────┼───────────────┐
    │               │               │
┌───▼───┐       ┌───▼───┐       ┌───▼───┐
│ Proxy │       │ Proxy │       │ Proxy │
├───────┤       ├───────┤       ├───────┤
│Service│◄─────►│Service│◄─────►│Service│
│   A   │       │   B   │       │   C   │
└───────┘       └───────┘       └───────┘
```

Sidecar Pattern

A sidecar container runs alongside the main application container:

```
 apiVersion: v1
kind: Pod
metadata:
  name: app-with-sidecar
spec:
  containers:
  - name: app
    image: my-app:v1
  - name: sidecar-proxy
    image: envoy:v1.20
```

Cloud Native Design Patterns

Circuit Breaker

Prevents cascading failures by stopping requests to failing services.

Retry with Backoff

Automatically retries failed requests with increasing delays.

Bulkhead

Isolates components to prevent failures from spreading.

Sidecar

Deploys helper components alongside the main application.

Ambassador

Creates helper services that send network requests on behalf of a consumer.

Key Concepts to Remember

1. **Cloud native is about how applications are built**, not where they run
2. **Microservices enable independent deployment** and scaling
3. **CNCF projects have maturity levels**: Graduated, Incubating, Sandbox
4. **Service meshes handle cross-cutting concerns** like security and observability
5. **Serverless abstracts infrastructure** management completely

Practice Questions

1. What are the key characteristics of cloud native applications?
2. Name three graduated CNCF projects.
3. What is the difference between a monolith and microservices?
4. What does a service mesh provide?
5. What is the sidecar pattern?

← Previous: Container Orchestration | Back to KCNA Overview | Next: Cloud Native Observability →