In-Class Assessment 4

Name: Yin Yuang
Student ID: 202383930027
Class: Software Engineering Class 1

Introduction to Cloud Computing
(Autumn,2025)

Nanjing University of Information Science and Technology, China
Waterford Institute

1. What problem does serverless computing aim to solve compared to traditional microservice deployment on Kubernetes?

Serverless computing aims to simplify infrastructure management by abstracting away server provisioning, scaling, and maintenance. Unlike traditional microservices deployment on Kubernetes, where developers need to manage pods, services, and scaling, serverless environments automatically scale and handle requests based on actual demand.

Example where serverless is better: For event-driven workloads like image processing, where functions are triggered by incoming events, serverless automatically handles scale, making it cost-efficient as you only pay for the resources you consume.

Example where serverless may not be suitable: Long-running applications with high CPU or memory requirements, such as stateful applications, might not be ideal for serverless, as the environment may introduce cold-start latency and resource limitations.

2. What are the advantages of using a service mesh (like Istio) for managing microservices communication instead of relying only on Kubernetes networking?

A service mesh, such as Istio, provides enhanced capabilities for managing microservices communication beyond Kubernetes networking:

- **Traffic management:** Fine-grained control over traffic routing, retries, timeouts, and circuit breaking.
- **Security:** Provides strong security mechanisms such as mutual TLS for secure service-to-service communication.
- **Observability:** Integrated metrics, logging, and tracing to monitor and trace requests between microservices.
- **Resilience:** Improved fault tolerance with features like retries and timeouts to ensure high availability.

3. Explain what a sidecar proxy (such as Envoy in Istio) does. Why is it needed in a service mesh?

A sidecar proxy, like Envoy, is deployed alongside each microservice instance (as a separate container in the same pod) to intercept and manage communication to and from the service. It provides:

- **Traffic Management:** Routing, load balancing, and retries.
- **Security:** Encryption of traffic via mutual TLS.
- **Observability:** Collection of metrics, logs, and traces.

In a service mesh, sidecar proxies are needed to decouple the communication logic from the application, enabling the management of traffic policies, security, and observability without modifying the application code.

4. What kind of traffic management features does Istio provide? Give two examples of how they can be useful in production systems.

Istio provides several traffic management features, including:

- **Routing Rules:** Fine-grained control over how requests are routed to microservices, such as A/B testing or canary deployments.
- **Retries and Timeouts:** Automatic retries for failed requests and configurable timeouts to ensure reliability.

Example in production systems: 1. Canary deployments can route a small percentage of traffic to a new version of a service to test it before a full rollout. 2. Automatic retries and timeouts ensure that services can handle transient failures gracefully without affecting the end user.

5. Explain how Knative Serving enables autoscaling for an application. What triggers scaling up and scaling down?

Knative Serving enables autoscaling by dynamically adjusting the number of instances of an application based on incoming traffic. It can scale down to zero when there are no incoming requests, saving resources.

Scaling Up: Knative scales up the application when there is an increase in incoming traffic, triggering the creation of additional pods.

Scaling Down: Knative scales down the application to zero when traffic decreases, terminating idle instances and freeing resources.

6. What is the role of Knative Eventing, and how does it support event-driven architectures?

Knative Eventing is a component of Knative that enables event-driven architecture by allowing services to produce, consume, and respond to events in a decoupled way. It facilitates the processing of events from various sources (e.g., HTTP, CloudEvents, Kafka).

It supports event-driven architectures by providing a uniform and scalable way to manage the flow of events between services, enabling automatic scaling of consumers based on event load.

7. How does Knative leverage Kubernetes primitives to provide a serverless experience? Discuss which components of Kubernetes (e.g., Deployments, Services, Horizontal Pod Autoscaler) are abstracted away and how this abstraction benefits developers.

Knative abstracts away several Kubernetes primitives to provide a simplified serverless experience. It automates the creation and scaling of services without requiring the user to manage:

- **Deployments:** Knative abstracts the creation and scaling of deployments, allowing users to focus on application logic rather than infrastructure.

- **Services:** Knative provides a high-level abstraction for managing services, handling routing and scaling automatically.
- **Horizontal Pod Autoscaler:** Knative automatically scales the number of pods based on incoming traffic, without the need to configure HPA manually.

This abstraction allows developers to focus on writing code without worrying about infrastructure and scaling details.

8. In KServe, what is the main function of an InferenceService, and how does it simplify deploying ML models?

In KServe, an **InferenceService** is the primary abstraction for deploying machine learning models. It simplifies deployment by abstracting the complex process of creating Kubernetes resources such as Pods, Deployments, and Services.

Main Functions:

- Define the model's runtime environment (e.g., framework, version).
- Automatically handles model loading, scaling, and monitoring.

This abstraction makes it easier for data scientists to deploy models without having to manually configure Kubernetes resources.

9. In a production ML workflow using KServe, describe how data moves from an incoming HTTP request to a model prediction response. Which layers (Knative, Istio, KServe, Kubernetes) handle which responsibilities, and where could latency bottlenecks occur?

In a production ML workflow using KServe:

- **HTTP Request:** The incoming request is routed to the appropriate KServe InferenceService via an Istio Ingress Gateway.
- **Knative Serving:** Knative scales the service based on incoming traffic, ensuring that the appropriate number of instances are running.
- **KServe:** Handles the model inference request and returns the prediction.
- **Kubernetes:** Manages the infrastructure and scaling of resources.

Latency Bottlenecks: - **Model Loading:** Initializing a large model can introduce latency. - **Network Latency:** Communication between services and clusters may introduce delays.

10. How can Istio's traffic routing capabilities (e.g., weighted routing, retries, circuit breaking) be used to support canary deployments or A/B testing in Knative or KServe environments? Discuss the pros and cons compared to manual rollout strategies.

Istio's traffic routing features can support canary deployments or A/B testing by directing a percentage of traffic to different versions of a service:

- **Weighted Routing:** Traffic can be split between multiple versions of a service based on predefined weights.
- **Retries and Circuit Breaking:** Automatic retries and circuit breaking help ensure resilience during testing.

Pros of Istio: - Automated, fine-grained control over traffic distribution. - Reduced risk during rollouts through gradual traffic shifting.

Cons of Manual Rollout: - Requires manual intervention and careful monitoring. - Risk of downtime or inconsistent behavior during the transition.