1. What problem does serverless computing aim to solve compared to traditional microservice deployment on Kubernetes? Give one example where serverless is clearly better, and one where it may not be.

Serverless computing primarily aims to abstract away the infrastructure management, allowing developers to focus on business logic without worrying about scaling or provisioning servers. Unlike Kubernetes-based microservices, where each microservice needs to be deployed and scaled individually, serverless platforms automatically manage resource scaling based on demand.

Example where serverless is better:

For a simple API that experiences sudden spikes in traffic, such as a holiday-themed event or a flash sale, serverless computing can automatically scale up and down without manual intervention.

Example where serverless may not be ideal:

For long-running services that require persistent connections or stateful operations (e.g., real-time data processing), serverless may introduce overhead due to cold starts or lack of control over resource management.

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 like Istio provides a dedicated layer for managing microservices communication, offering features like traffic management, security, and observability. It abstracts the complexity of networking away from the individual microservices, allowing developers to focus on business logic.

Advantages over Kubernetes networking:

Traffic Control: Advanced routing, load balancing, and retries without modifying the service code.

Security: Enforces mTLS for secure communication between services.

Observability: Provides in-depth metrics, logging, and tracing for microservices communication.

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

A sidecar proxy is deployed alongside each service in the form of a small container that handles networking concerns (e.g., service discovery, retries, traffic management, monitoring). It acts as a mediator between the service and the network.

Need in a service mesh:

The sidecar proxy abstracts the network logic, enabling the service to focus solely on its business logic. This simplifies the microservice architecture and allows for centralized management of traffic, security, and observability.

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

Istio offers several traffic management features such as:

Weighted Routing: Directs a percentage of traffic to different versions of a service, useful for canary deployments.

Circuit Breaking: Prevents cascading failures by limiting traffic to a failing service, ensuring stability.

Production Usefulness:

Canary Deployments: Weighted routing allows gradual rollout of a new version of a service with minimal risk.

Reliability: Circuit breaking can enhance service resilience by stopping faulty services from impacting the rest of the system.

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

Knative Serving enables autoscaling based on the incoming traffic. It uses horizontal scaling to scale up the application when requests come in, and scales down to zero when traffic is low or non-existent.

Triggers:

Scaling Up: New requests trigger the scaling of pods to handle the load.

Scaling Down: When there are no incoming requests, Knative will scale the application down to zero pods.

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

Knative Eventing provides a framework for building event-driven architectures. It decouples the event producers (e.g., services that emit events) from consumers (e.g., services that process events).

Support for Event-driven Architectures:

It allows services to trigger events in response to external actions (e.g., HTTP requests, Kafka messages), ensuring loose coupling and scalability.

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 uses Kubernetes resources such as Deployments, Services, and Horizontal Pod Autoscaler to abstract the complexity of serverless operations. It allows developers to focus on code while Kubernetes takes care of scaling, routing, and infrastructure management.

Abstracted Components:

Deployments and Services are abstracted into Knative services that automatically manage scaling.

The Horizontal Pod Autoscaler is leveraged to scale the application up or down based on traffic.

Benefits: This abstraction simplifies developer operations and enables a serverless experience without the need for complex Kubernetes configurations.

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

An InferenceService in KServe is a resource that simplifies deploying machine learning models by encapsulating model deployment, serving, and management into a single abstraction.

Simplification: It handles versioning, scaling, and routing of predictions automatically, providing a standardized interface for serving models.

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:

Data flow: An incoming HTTP request triggers KServe to forward the data to the model for inference.

Responsibilities:

Knative handles autoscaling of the model service based on traffic.

Istio manages traffic routing, retries, and security.

KServe handles the model lifecycle and inference.

Kubernetes manages the underlying infrastructure.

Latency Bottlenecks:

The network latency between services (Istio and KServe).

Model inference latency in KServe.

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, such as weighted routing, allow traffic to be distributed between different versions of a service, enabling canary deployments or A/B testing.

Pros:

Gradual Rollouts: Test new versions with a small fraction of traffic before full deployment.

Reduced Risk: Canary deployments minimize the impact of bugs in new versions.

Cons:

More complex to manage compared to manual rollout strategies.

Requires continuous monitoring to ensure smooth transitions.