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 aims to solve the infrastructure management complexity and resource costs associated with traditional microservice deployment on Kubernetes. It abstracts server management, offering pay-per-use pricing, automatic scaling, and high availability, allowing developers to focus on code rather than operations. For example, serverless is clearly better for event-driven, short-lived tasks like image processing or API gateways, as it handles traffic spikes automatically with lower costs. However, it may not be suitable for long-running, compute-intensive applications such as batch processing or real-time video streaming, due to runtime limits and cold start latency, whereas Kubernetes provides more consistent performance and control

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 offers advantages over basic Kubernetes networking by providing advanced traffic management (e.g., canary deployments and fault injection), observability (e.g., metrics and tracing), enhanced security (e.g., mTLS encryption), and resilience features (e.g., retries and circuit breaking). These capabilities decouple communication logic from application code, simplifying governance in complex microservices environments. For instance, Istio automatically secures inter-service communications via sidecar proxies, whereas Kubernetes only offers basic load balancing and network policies

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

A sidecar proxy (e.g., Envoy in Istio) is a separate process deployed alongside a microservice to intercept and manage all inbound/outbound traffic, offering features like load balancing, service discovery, TLS termination, and metric collection. It is needed in a service mesh to decouple network functions (e.g., security and observability) from business logic, enabling consistent policy enforcement without code changes. For example, Istio uses Envoy sidecars to implement mTLS and traffic routing rules

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

Istio provides traffic management features such as weighted routing, retries, timeouts, circuit breaking, and fault injection. For production systems, canary deployments (via weighted routing) allow gradual traffic shifts to new versions, reducing rollout risks, while fault injection helps test resilience by simulating delays to verify timeout handling

.

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

Knative Serving enables autoscaling by abstracting Kubernetes' Horizontal Pod Autoscaler (HPA), triggering scale-up based on HTTP request concurrency or custom metrics (e.g., queue depth), and scale-down to zero during inactivity to reduce resources. For instance, when request rates exceed a threshold, Pod instances increase rapidly; with no traffic, it scales to zero for cost efficiency(Note: Search results lack direct details on Knative, but this is inferred from serverless principles).

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

Knative Eventing manages event production and consumption, linking sources (e.g., Kafka or HTTP requests) to serverless functions to support event-driven architectures. It uses brokers and triggers for event routing, allowing functions to respond to asynchronous events (e.g., file uploads). For example, when a new file arrives in cloud storage, Knative Eventing can trigger a processing function without requiring always-on services

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 leverages Kubernetes primitives by abstracting components like Deployments, Services, and HPA through custom resources (e.g., Service and Route), offering a simpler API. This abstraction benefits developers by eliminating the need to manage Pod scaling or load balancing directly, accelerating deployment and reducing operational overhead. For instance, Knative automates rollouts and networking, minimizing YAML configuration

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

The InferenceService in KServe serves as a scalable API endpoint for ML models, automating versioning, scaling, and traffic routing. It simplifies deployment by enabling declarative setup for A/B testing and blue-green deployments without manual Kubernetes management. For example, developers define the model path and resources, and KServe handles load balancing for inference requests

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, an HTTP request is routed by Istio to KServe's InferenceService, with Knative managing autoscaling and Kubernetes orchestrating Pods. Responsibilities include: Istio for traffic and security, Knative for lifecycle management, KServe for model serving, and Kubernetes for resources. Latency bottlenecks may occur due to cold starts (after scale-to-zero), model loading times, or network hops (e.g., sidecar proxies)

10. How can Istio鈥檚 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.

In Knative/KServe environments, Istio supports canary deployments or A/B testing via weighted routing (e.g., directing 10% of traffic to a new model version). Pros include precise traffic control, automated rollback, and real-time metrics, reducing risks; cons involve added configuration complexity and reliance on service mesh expertise. Manual strategies (e.g., editing Kubernetes Deployments) are simpler but lack fine-grained control and automation.