**1. Serverless vs Traditional Microservices on Kubernetes**

Problem serverless aims to solve:

Traditional microservices require you to manage pod scaling, load balancing, and cluster resource usage, which can be complex and resource-intensive. Serverless abstracts this away: developers only focus on code, and the platform handles scaling, scheduling, and lifecycle management automatically.

Example where serverless is clearly better:

Event-driven webhooks: A service that responds to occasional webhook events. Serverless scales up only when requests arrive, saving resources during idle periods.

Example where serverless may not be ideal:

High-throughput, low-latency services (like a real-time trading engine). Serverless cold-start delays and stateless limitations may hurt performance.

**2. Advantages of using a service mesh like Istio**

Fine-grained traffic control: Routing, retries, circuit-breaking, and fault injection without modifying application code.

Security: Automatic mTLS between services.

Observability: Metrics, tracing, and logging for all microservice traffic.

Policy enforcement: Quotas, rate-limiting, and access control.

Compared to plain Kubernetes networking, Istio gives application-level visibility and control instead of just L3/L4 connectivity.

**3. Sidecar proxy (Envoy)**

Role:

Intercepts all inbound and outbound traffic for a microservice.

Handles retries, routing, telemetry, and security on behalf of the service.

Why needed:

Services remain agnostic of networking concerns.

Provides consistent, centralized traffic management, observability, and security policies without modifying service code.

**4. Istio traffic management features**

Examples:

Weighted routing: Gradually shift traffic between versions for canary deployments.

Useful to reduce risk when deploying a new microservice version.

Circuit breaking & retries: Automatically prevent cascading failures in production.

Useful in high-load scenarios to keep services resilient.

**5. Knative Serving autoscaling**

How it works:

Monitors metrics such as request concurrency or CPU usage.

Automatically scales pods up when demand increases, and scales down to zero when idle.

Triggers:

Scale up: Increased incoming requests.

Scale down: No requests within a configured time window.

**6. Role of Knative Eventing**

Provides event routing infrastructure for serverless workloads.

Supports pub/sub patterns: services can subscribe to events without knowing the producer.

Enables event-driven architectures, where functions run in response to events like messages, HTTP calls, or cloud notifications.

**7. Knative leveraging Kubernetes primitives**

Abstracted Kubernetes components:

Deployments: Knative automatically creates/manages them.

Services: Knative manages network routing.

Horizontal Pod Autoscaler: Abstracted via Knative autoscaling policies.

Benefit:

Developers focus on writing code, not cluster management.

Enables autoscaling to zero, versioning, and traffic splitting without manual HPA or service changes.

**8. KServe InferenceService**

Main function:

Encapsulates all resources needed to serve an ML model: predictor, transformer, explainer, etc.

Benefit:

Simplifies deployment: a single CRD defines a fully functional inference endpoint.

Automatically integrates autoscaling, networking, and routing.

**9. Data flow in production ML with KServe**

Flow:

HTTP request arrives → Knative Serving ingress.

Knative routes request → Istio handles routing, retries, mTLS, traffic splitting.

KServe InferenceService → Picks predictor pod for the model.

Kubernetes schedules pods, manages container resources.

Prediction response returns through same layers.

Latency bottlenecks:

Knative cold-start (if pods scaled to zero)

Istio sidecar processing (Envoy overhead)

Model loading/compute time (heavy ML models)

**10. Istio traffic routing for canary or A/B testing**

How to use:

Weighted routing: Send 10% of traffic to new version.

Retries/circuit breaking: Handle failures in the canary version.

Pros compared to manual rollout:

Safe gradual rollout, immediate rollback possible

Fine-grained control over traffic

Metrics-driven decisions

Cons:

Adds operational complexity (Istio config management)

Slight network latency due to sidecar proxy

Requires monitoring infrastructure to act on metrics