

SIDECAR PROXY IN KUBERNETES

PRESENTED BY: DIYA ANNA SUNIL

INTRODUCTION

KUBERNETES

Kubernetes is a container orchestration platform that automates deployment, scaling and management of containerized applications.

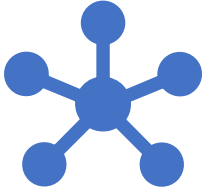
It is widely used to manage microservices at scale.

WHAT IS A SIDECAR PROXY?

The Sidecar pattern is a design pattern for containerized applications.

A sidecar proxy is a helper container running alongside the main application container in the same Pod.

It extends functionality like networking, logging, and security without modifying the main application.



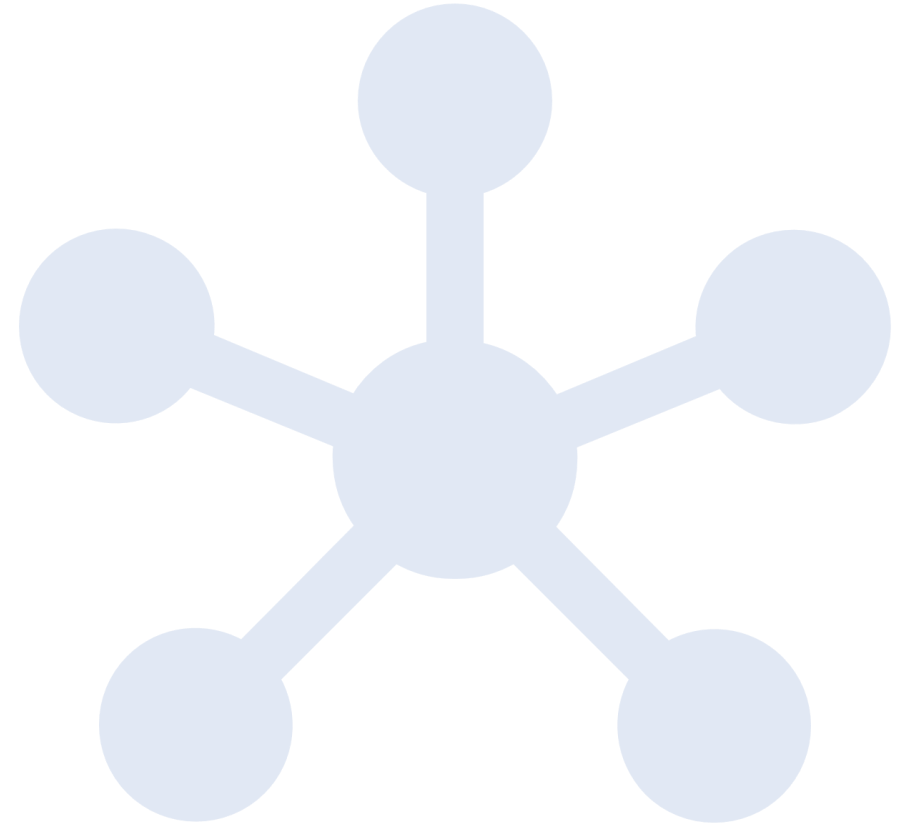
WHY USE A SIDECAR PROXY?

CHALLENGES IN DISTRIBUTED SYSTEMS

- Service discovery and load balancing.
- Observability for monitoring and debugging.
- Security through encrypted communication (mTLS).
- Traffic routing and management.

SIDECAR PROXY AS A SOLUTION

- Simplifies service-to-service communication.
- Offloads operational concerns from the application code.



SIDECAR PROXY ARCHITECTURE

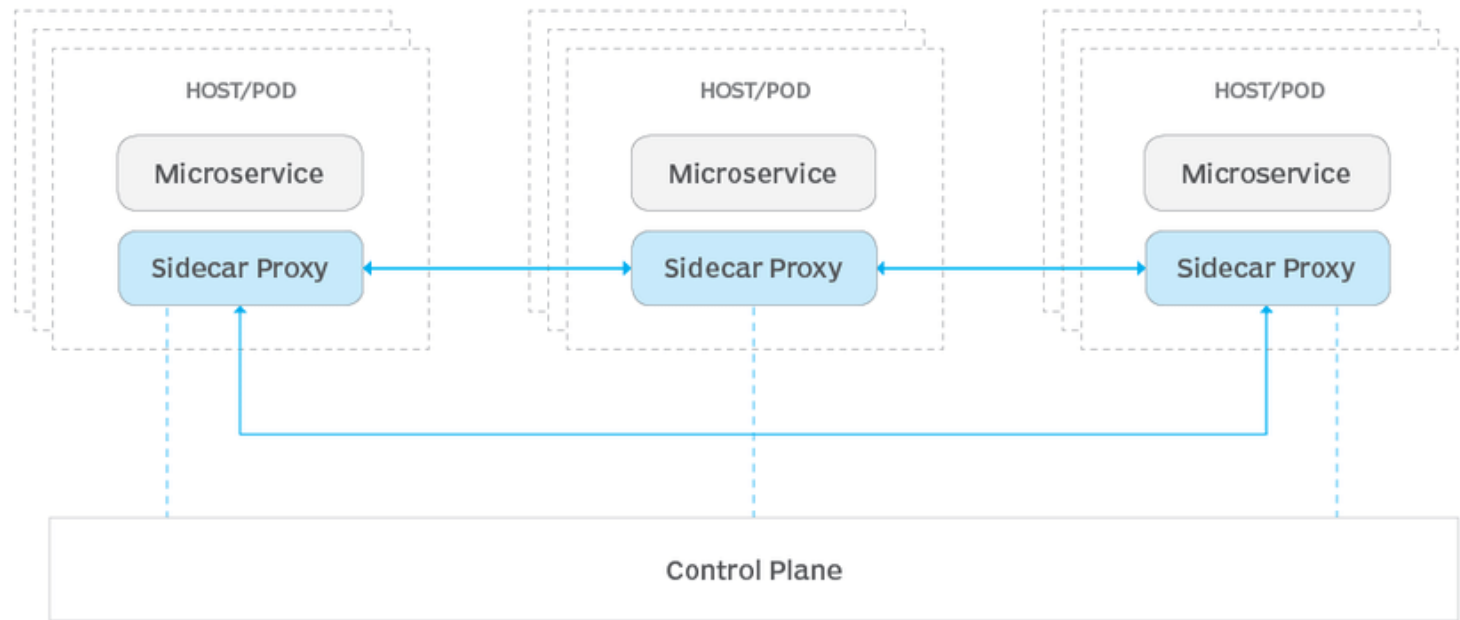
HOW IT WORKS IN KUBERNETES

- The sidecar proxy is deployed within the same Pod as the main application container.
- Both containers share networking and can communicate over localhost.

TYPICAL SETUP

- Main Application Container: Handles business logic.
- Sidecar Container: Manages networking, security, and observability.

ARCHITECTURAL SETUP



USE CASES

1.SERVICE MESH INTEGRATION

Tools like Istio and Linkerd rely on sidecars (e.g., Envoy) for managing service-to-service communication.

2.OBSERVABILITY

Collect logs and metrics for performance monitoring (e.g., Prometheus, Grafana).

3.SECURITY

Implement mutual TLS (mTLS) for encrypted communication

4.TRAFFIC MANAGEMENT

Enable retries, rate limiting, and intelligent routing.

SIDECAR PROXY IN PRACTICE

- CODE DEMO



ENGINEERING BENEFITS

1.DECOUPLING BUSINESS LOGIC

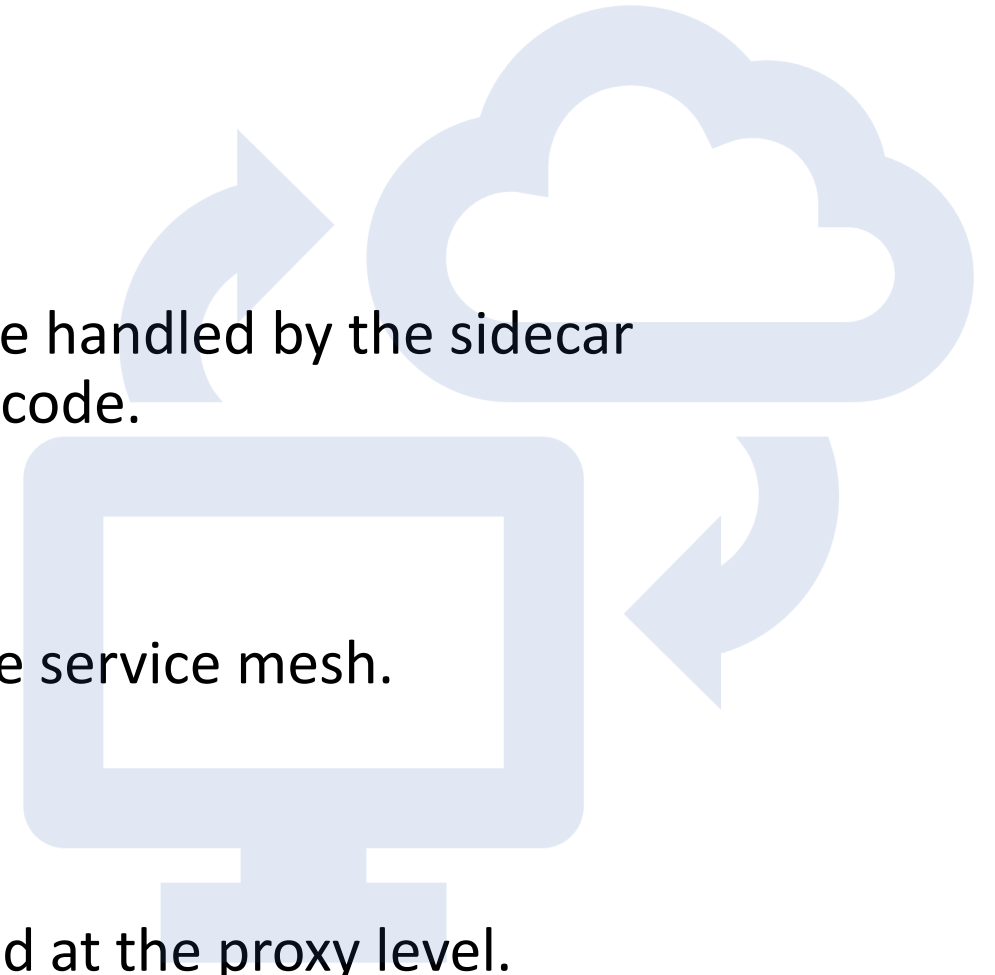
Operational features (e.g., monitoring, security) are handled by the sidecar proxy, allowing developers to focus on application code.

2.SCALABILITY

Centralized control over distributed services via the service mesh.

3.RESILENCY

Fault tolerance and retry mechanisms implemented at the proxy level.





CHALLENGES

1.Resource Overhead

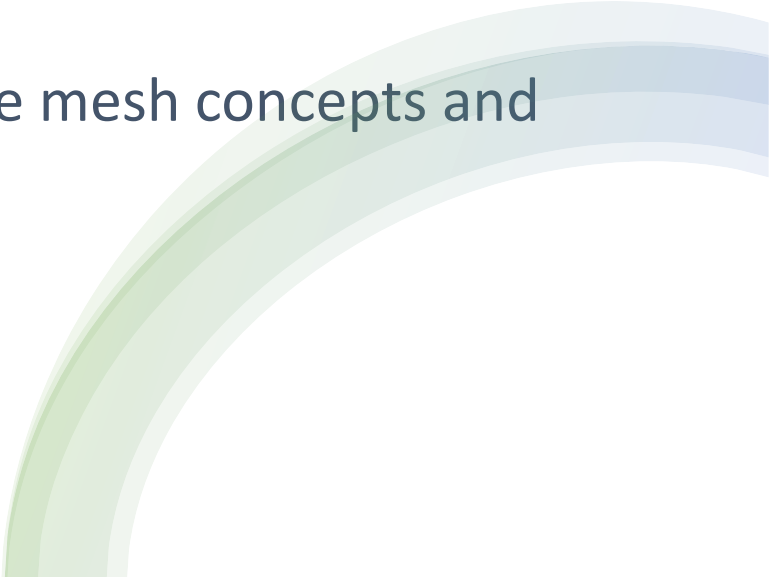
Sidecar proxies consume CPU and memory resources.

2.Complexity

Managing sidecars for multiple Pods can add operational overhead.

3.Learning Curve

Engineers must understand service mesh concepts and tools.



FUTURE TRENDS

eBPF-based Solutions

- Lightweight networking and observability without sidecars.

Advances in Service Mesh Technologies

- Improved performance and ease of use in tools like Istio and Linkerd

CONCLUSION

- Sidecar proxies solve critical engineering challenges in distributed systems.
- They enable enhanced functionality such as service discovery, security, and observability.
- Essential for building scalable, resilient microservice-based architectures.

The background features a series of concentric, semi-transparent circles that create a tunnel-like effect. The color gradient transitions from a light blue on the left to a light green on the right. The text "THANKYOU" is centered within the circles.

THANKYOU