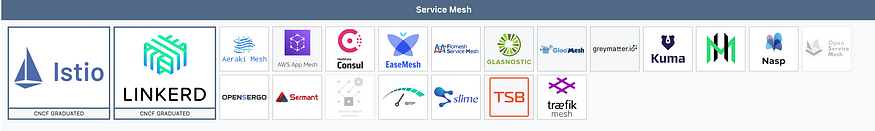
In software development, managing applications has become increasingly complex. With the rise of containers and Kubernetes, developers have been faced with the challenge of [handling network traffic](https://istio.io/latest/docs/concepts/traffic-management/), [security](https://istio.io/latest/docs/concepts/security/), and [observability](https://istio.io/latest/docs/concepts/observability/), [extensibility](https://istio.io/latest/docs/concepts/wasm/). This is where Istio comes in.

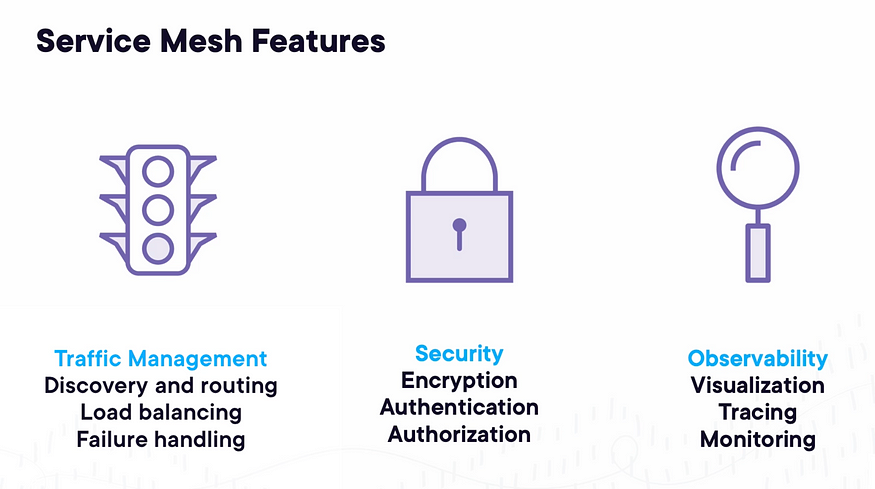
[Istio](https://github.com/istio/istio) is a service mesh, there are many other service mesh, for example, [Linkerd](https://github.com/linkerd/linkerd2), and [Kuma](https://github.com/kumahq/kuma) are service mesh implementations.



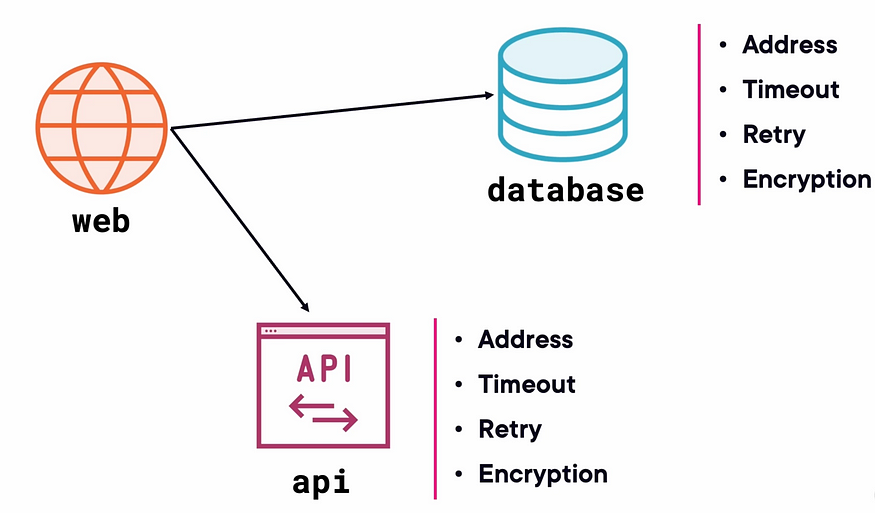
**So, what is a service mesh?**

A service mesh connects all the components of your application and lets you simplify your app development and configuration. It takes away concerns like managing network traffic and security, and dealing with temporary faults, and lets you control all that in a central place in the mesh. Think of it as the communication layer between the software components in a distributed system, made into its own entity.

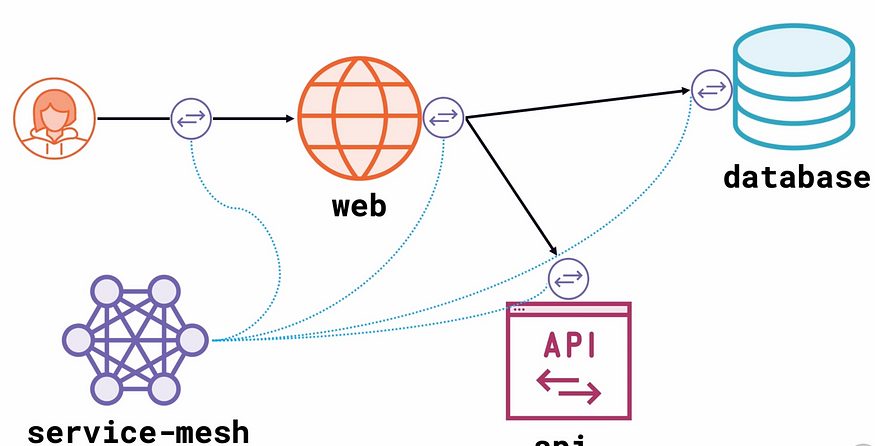
This communication layer becomes a separate entity that you can control independently of all the other components. This means you can change how those components talk to each other without changing the configuration of your apps or doing a new release of your software. In Kubernetes, you don’t need to do a new deployment.



Let’s take a look at a simple distributed system. You have a web application that talks to a database and an API. Traditionally, the web application would need to know a lot of details about these services, such as their addresses, how to handle connection failures, and whether to encrypt the communication. All these concerns would be managed within the web application itself.



With service mesh, in this case, Istio, you can offload these concerns to the istio. The service mesh injects a network proxy into each component, which takes control of all the network traffic. This allows you to configure the communication between components in the mesh, including timeouts, retries, encryption, and authentication.



Once you have Istio in place, you can control the communication between components in three main areas: traffic management, security, and observability. Istio can discover service endpoints, take control of routing, provide efficient load balancing, handle failures, and provide advanced security features such as mutual TLS, authentication, and authorization. It also provides observability features such as service graphs and distributed tracing.

Istio is composed of these components:

1. Envoy — Sidecar proxies per microservice to handle ingress/egress traffic between services in the cluster and from a service to external services. The proxies form a *secure microservice mesh* providing a rich set of functions like discovery, rich layer-7 routing, circuit breakers, policy enforcement, and telemetry recording/reporting functions.

* Note: The service mesh is not an overlay network. It simplifies and enhances how microservices in an application talk to each other over the network provided by the underlying platform.

2. Istiod — The Istio control plane. It provides service discovery, configuration, and certificate management. It consists of the following sub-components:

* Pilot — Responsible for configuring the proxies at runtime.
* Citadel — Responsible for certificate issuance and rotation.
* Galley — Responsible for validating, ingesting, aggregating, transforming, and distributing config within Istio.

3. Operator — The component provides user-friendly options to operate the Istio service mesh.

1. Simplifying Microservices Communication: In a microservices architecture, applications are composed of many small, independent services that communicate with each other over a network. A service mesh provides a centralized way to manage this communication, making it easier for developers to understand and control how services interact.
2. Handling Network Traffic: A service mesh can handle tasks like load balancing, routing, and traffic shaping, which can be complex and time-consuming for developers to manage manually. This allows developers to focus on building and improving their applications, rather than worrying about the underlying infrastructure.
3. Implementing Security Policies: A service mesh can enforce security policies such as encryption, authentication, and authorization, ensuring that only authorized services can communicate with each other. This helps developers build more secure applications without having to implement these features themselves.
4. Monitoring and Observability: A service mesh can provide detailed insights into how services are performing, including metrics like latency, error rates, and throughput. This can help developers identify and troubleshoot issues more quickly, improving the overall reliability and performance of their applications.

Inconclusion, Istio is a powerful tool that simplifies application management by providing a centralized control plane for managing network traffic, security, and fault tolerance in distributed systems. It allows you to focus on developing your applications without worrying about the underlying infrastructure.