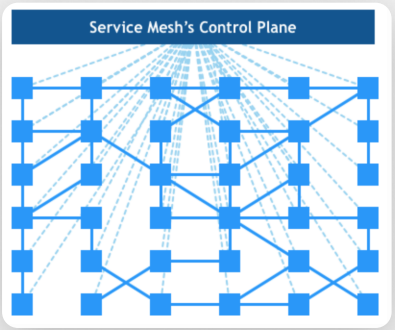
microstack

Service Mesh and Zero-Trust Architecture

“Decompose the monolith” is the new mantra of application modernization.

Containers are the preferred solution for delivering microservices, but they come with the cost of more security challenges than the Monolith system. The Monolith involves systems running in dedicated hardware, overprovisioning systems, and a handful of network connections (web tier, middle tier, and backend). Naturally, this produces several questions:

- **Who can access my internal services?**
- **Are all calls to my internal services secured enough?**
- **Can somebody from outside call my service?**
- **Can my internal service control rates and limits?**
- **Do I need to monitor all services and performance individually?**



In modern architecture, network diagrams are no longer layered. In the era of microservices, it’s more of a mesh that’s dynamic in nature.

Despite this, network infrastructure is still one flat box. It’s hard to penetrate a network, but once you enter, it’s a flat world in the mesh of microservices. This means that a network rail guard at the edge isn’t enough to limit your services' boundaries and accessibility. A network intrusion by any compromised services can bring the entire island down.

Fortunately, several solutions (like Envoy proxy in Istio, or Finagle & Jetty in Linkerd) provide solutions to the above questions. Service Mesh provides a single pane of glass for all management and observability. The beauty is that it’s language independent and comes as an operator sidecar stack rather than libraries baked into the code.

What Is a Service Mesh?

The end goal of a Service Mesh is the ability to talk to another service securely in a controlled fashion that supports transparency and observability. It’s a concept that facilitates the basic principles of microservices:

- **Smart endpoints & dumb pipes**
- **Decentralized service discovery**
- **Decentralized governance with centralized policy management**
- **Smart, policy-driven load balancing**
- **Design for failure**
- **Circuit breaker patterns**
- **Centralized exit and entry (inbound endpoint and outbound endpoints) with managed**
- **policies**
- **Rate limiting**
- **End-to-end encryption**
- **Visualize and monitor your mesh**
- **Zero trust security**
- **Fine-grained advanced canary deployment**
- **Fault injection**
- **The last service mesh doesn’t have any knowledge about the business**

We've heard clients ask why Kubernetes itself doesn't come with a service mesh. The answer is that Kubernetes promotes microservices architecture using the service construct. Services are wrapped in containers (usually docker containers) and further repackaged to the pod with the container, volume, and network namespace. Lastly, a clustered pod (a.k.a. deployment) is exposed with a well-defined API using basic L4 load balancing services. ServiceMesh solves the higher-level problems such as using L7 load balancing with the visibility and security construct.

Another common question we hear has to do with the role of the Network Policy in Service Mesh. For your convenience, we've broken down the differences between the Kubernetes Network Policy and Service Mesh below:

	Istio Policy	Network Policy
Layer	**L7**	**L4**
Implementation	**User Space**	**Kernal**
Enforcement Point	**Pod**	**Node**

A recent report from CNCF shows that Istio is by far the best choice and covers most of the requirements of a service mesh. At this point in time, all the capabilities of Istio have been outlined. Next, let's see how Istio can help to solve high-security requirements like PCI standards.

PCI Requirements Using a Service Mesh

For PCI DSS compliance, there are six major requirements that companies must follow:

- **Build and Maintain a Secure Network and Systems**
- **Protect Cardholder Data**
- **Maintain a Vulnerability Management Program**
- **Implement Strong Access Control Measures**
- **Regularly Monitor and Test Networks**
- **Maintain an Information Security Policy**

Let's review each of these in more detail.

Build and Maintain a Secure Network and Systems

As mentioned earlier, maintaining a secure network in a microservice mesh can't be achieved with network firewalls alone. Microservices in general are mortal; they could auto-scale up or down based on load. IP addresses assigned to a pod could get reallocated to other pods once the current pod gets evicted.

The point is in the world of microservices “IP-IP” firewall is a NO-GO. It needs to move to policy-based “application-to-application” firewall with application identification. Again, service meshes come to the rescue. Meshes provide a unique identity to each microservice for authentication, and based on the policy, they authorize microservice-to-microservice communication.

And in terms of maintenance, a service mesh provides centralized management of all policies using YAML with Kubernetes-supported declarative pipelines.

Protect Cardholder Data

The biggest change in breaking up a monolithic application to a microservices-based application is changing the communication mechanism. The application must evolve from a language-based method call to an inter-process network communication protocol, such as HTTP, TCP/UDP. This is why encrypting the inter-process network communication is so important. Service meshes provide out-of-the-box features to configure mutual TLS globally on a k8s cluster and provide data on transit, encrypted by default.

Maintain a Vulnerability Management Program

While Service Mesh doesn’t directly do vulnerability management, it can be easily achieved by container scanning software like Clair, SNYK, etc.

Implement Strong Access Control Measures

Microservices have specific security needs. mTLS defends against man-in-the-middle attacks and implements traffic encryption. Fine-grained access policies also uniquely audit every action to ensure maximum security.

Regularly Monitor and Test Networks

One of the biggest challenges of microservices is to capture metrics and data from thousands of microservices. This data is essential for tracing bad microservices and finger-pointing a problematic service out of the thousands of microservices currently running. Improperly-designed microservices often cause chain failures of multiple microservices and it make it even harder to locate the root cause of failures.

Service Mesh observability tools like Grafana, Prometheus, or Kiali provide a single log connection and visibility of the entire mesh. Tools like Jaeger, Zipkin, or Opentracing help distributed tracing backed by the centralized log collections from other tools like Prometheus Elasticsearch and Stackdriver. These tools coordinate to create a simple, effective system of network monitoring.

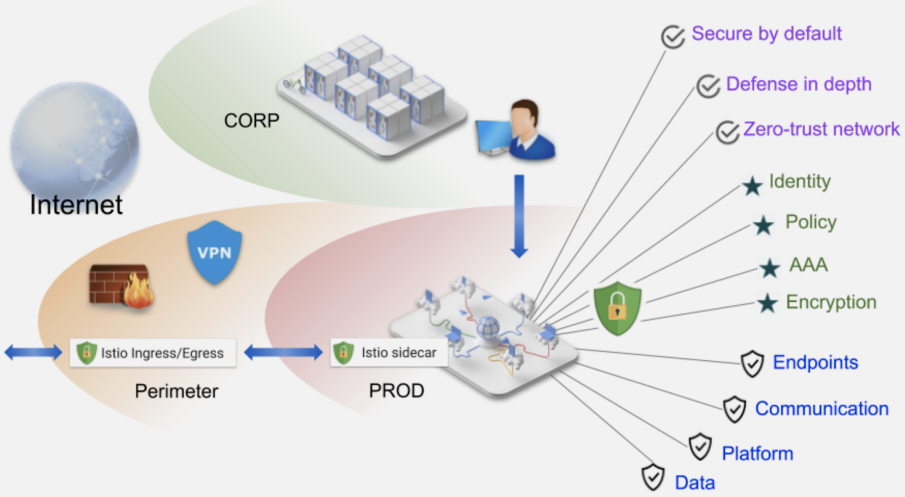
Maintain an Information Security Policy

The world of microservices is impossible to manage without having networking and security as declarative policies, backed by immutable security. This extends to network security policies as well. All policies in the Service Mesh are declarative policies using YAML.

Service Mesh and Zero-Trust Architecture

Finally, we come to Zero trust security. It's not just some buzzword; it's a security requirement of microservices. Zero trust security in Kubernetes Microservices can be broken down as the following concepts:

- **Don't assume that someone else will protect your service.**
- **By default, everything is untrusted.**
- **No trusted users.**
- **No trusted applications.**
- **No trusted packets.**
- **Verify before trust.**
- **Log & inspect all traffic.**
- **Embedded, default security for every pod.**
- **Context-aware access control.**
- **The firewall at the edge.**
- **In-Depth Defense: Add existing security systems to create a layered architecture.**



Source: https://istio.io/latest/docs/concepts/security/overview.svg

At its core, the Service Mesh security stack provides identity, policy, TLS encryption, and AAA (authentication, authorization, and audits). Taken together, this stack protects the application, data, and network, building a zero trust architecture all within Kubernetes.

Learn More About the Benefits of Service Mesh Security

While the above rundowns offer a good starting point for microservices security, the concept runs deeper. For microservices, a Service Mesh can be added as an "onion" layer security model to create a truly zero trust architecture in your enterprise. For more details, check out https://istio.io/ as well as the tutorial.

At Microstack, we specialize in best practices and solutions developed specifically for the Kubernetes environment. Contact us to learn more about our in-depth Kubernetes security solutions.

microstack