


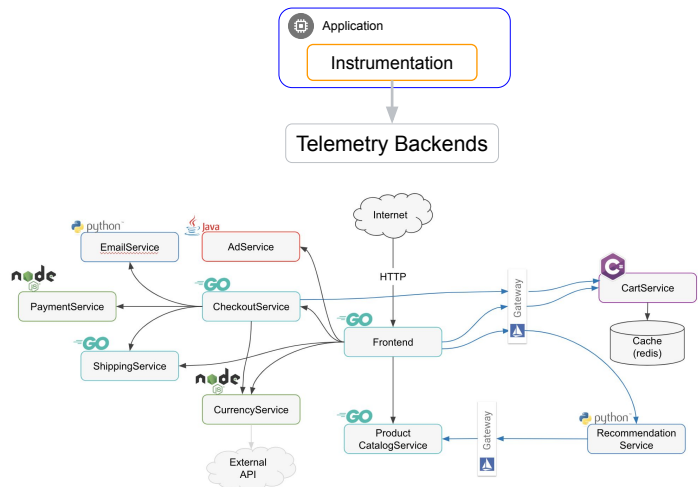
Today's agenda



- 01 Introduction to Anthos Service Mesh
- 02 Lab: Anthos Service Mesh Walkthrough
- 03 Lab review
- 04 Architecture
- 05 Installation
- 06 Life of a request in the mesh
- 07 [Mesh telemetry and instrumentation](#)
- 08 Anthos Service Mesh dashboards
- 09 Anthos Service Mesh pricing and support
- 10 Lab: Observing Anthos Services

Distributed service telemetry

- The challenge of collecting measurements on the use and performance of distributed services
- Traditionally supplied via the application instrumentation
- Difficult to manage in polyglot environments

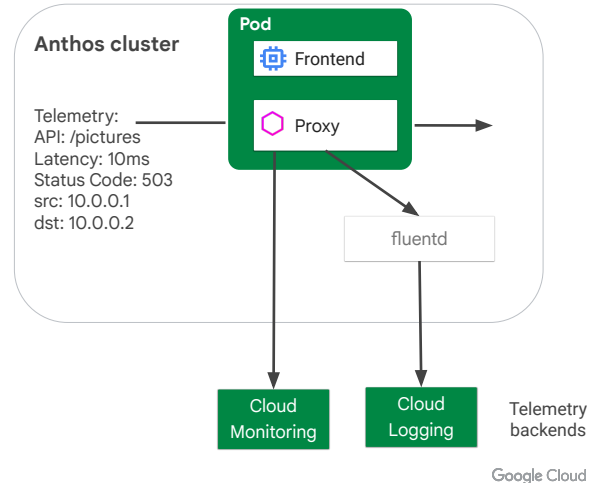


Traditionally, programmers have implemented instrumentation in the form of code instructions that monitor specific components in a system in order to monitor or measure the level of a product's performance, to diagnose errors, and to write trace information.

As containers are becoming the compute unit of choice, environments becomes more polyglot as more and more diverse programming languages are introduced, making this effort very hard to maintain and track, and it hinders the philosophy of separation of duties between operators and developers.

Telemetry approach with distributed proxies

- Envoy proxies collect metrics, traces, and logs, and send them to Google Cloud's operations suite and to Prometheus.
- Logs are exported to the node, and a fluent-bit daemonset forwards them to Cloud Logging.
- Standard and custom metrics collected in the Anthos Service Mesh by the Envoy sidecar proxies include:
 - Proxy metrics
 - Service metrics
 - Control plane metrics



To obtain telemetry data, Anthos Service Mesh relies on the Envoy sidecar proxies that you inject in your workload pods. The proxies intercept all inbound and outbound HTTP traffic to the workloads and report the data to Anthos Service Mesh. They report data by calling the Cloud Monitoring, Cloud Logging, and Cloud Tracing APIs, as well as the Prometheus service in the cluster. With this system, service developers don't have to instrument their code to collect telemetry data.

Logs are exported to the node and a fluentd daemonset that comes with the Anthos cluster forwards the logs to Cloud Logging.

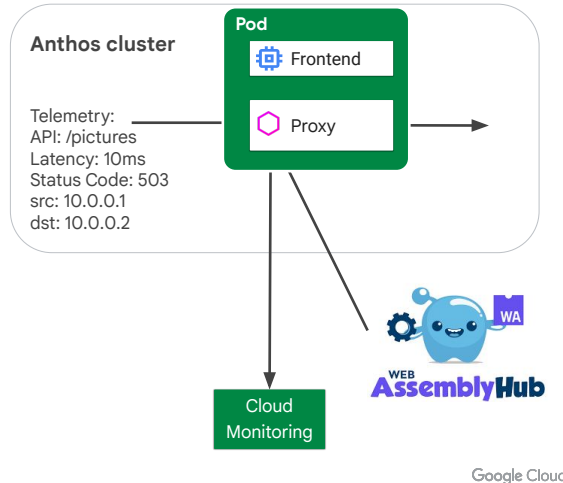
Standard and custom metrics collected in the Anthos Service Mesh by the Envoy sidecar proxies include proxy, service, and control plane metrics.

You can configure custom metrics with Envoy Filters, an Istio CRD that gets applied directly to the Envoy Proxy. However, Envoy Filters are not supported in ASM 1.11 and, therefore, won't be covered in this course.

Extend service mesh functionality with Wasm

WebAssembly (Wasm) enables you to deploy third-party plugins directly on the Envoy proxy and provides the following benefits:

- Multi-language plugin support
- Plugin development agility
- Plugin reliability and isolation
- Plugin security
- Easy plugin management with WebAssembly Hub



As of this writing (August 2023), Wasm is in Alpha.

You might be wondering, how does Envoy know how to communicate with Google Cloud's Operations suite APIs?

Anthos Service Mesh installs a WebAssembly plugin in the Envoy proxy that contains the functionality to call the Cloud Monitoring and Cloud Tracing APIs.

WebAssembly (or `wasm` for short) is a portable bytecode format for executing code written in multiple languages at near-native speed built into Google's high performance V8 engine. After receiving a W3C recommendation in Dec 2019 to run natively in all major browsers, it is the fourth standard language (following HTML, CSS, and JavaScript).

Wasm enables you to deploy third-party telemetry filters that can redirect requests to your observability and policy backends.

Benefits of Wasm include:

- Multi-language plugin support, such as C++, Rust, and AssemblyScript with more to come. Also, [over 30 programming languages can be compiled](#) to WebAssembly, allowing developers from all backgrounds to write Envoy extensions in their language of choice.
- Plugin development agility: extensions can be delivered and reloaded at runtime using the Istio control plane. This enables a fast develop → test → release cycle for extensions without requiring Envoy rollouts.

- Reliability and isolation: extensions are deployed inside a sandbox with resource constraints, which means they can now crash, or leak memory, without bringing the whole Envoy process down. CPU and memory usage can also be constrained.
- Security: the sandbox has a clearly defined API for communicating with Envoy, so extensions only have access to, and can modify, a limited number of properties of a connection or request. Furthermore, because Envoy mediates this interaction, it can hide or sanitize sensitive information from the extension (e.g., “Authorization” and “Cookie” HTTP headers, or the client’s IP address).
- Easily create and deploy modules to your service mesh, and publish and share them to WebAssembly Hub as a repository. (Similar to how Helm charts are deployed to the stable.)

Like Docker, the [WebAssembly Hub](https://webassemblyhub.com/) stores and distributes Wasm extensions as OCI images. This makes pushing, pulling, and running extensions as easy as Docker containers. Wasm extension images are versioned and cryptographically secure, making it safe to run extensions locally the same way you would in production. This allows you to build and push, as well as trust the source when they pull down and deploy images.

--- For instructors/students reference ---

<https://www.youtube.com/watch?v=-XPTGXEpUp8>

<https://istio.io/blog/2020/wasm-announce/>

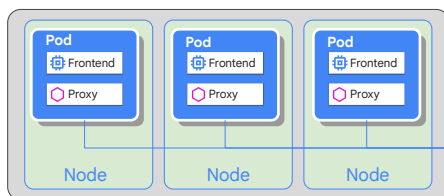
<https://istio.io/docs/concepts/wasm/>

Tutorial:

- Build: https://docs.solo.io/web-assembly-hub/latest/tutorial_code/build_tutorials/building_assemblyscript_filters/
- Deploy: https://docs.solo.io/web-assembly-hub/latest/tutorial_code/deploy_tutorials/deploying_with_istio/
- Other: https://docs.solo.io/web-assembly-hub/latest/tutorial_code/

Enabling trace data collection

- Anthos Service Mesh can collect trace data at the mesh level.
- Trace data can be forwarded to different backends.
- Tracing in general is disabled by default but is easy to enable.



Anthos Cluster

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: istio-asm-managed
  namespace: istio-system
data:
  mesh: |-
    defaultConfig:
      tracing:
        Stackdriver: {}
```

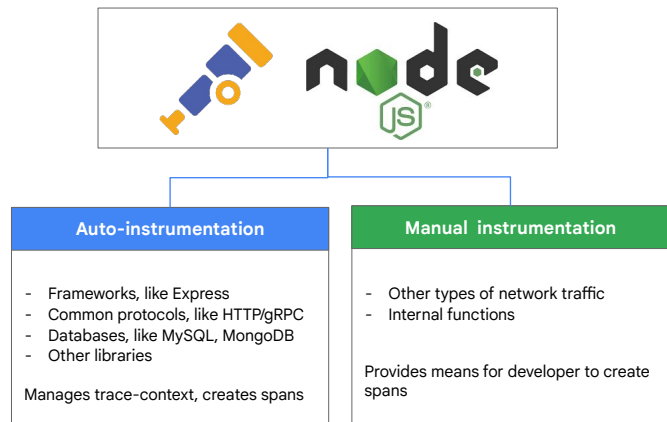
```
kubectl apply -f configmap.yaml
```

Google Cloud

While logging and monitoring are enabled by default, you have to enable tracing yourself. To do so, configure an overlay file for In-cluster ASM or apply a Kubernetes ConfigMap with tracing set to stackdriver as a backend for Managed ASM.

Tracing instrumentation

- Systems must be instrumented and send traces to Cloud Trace.
- Libraries provide some auto-instrumentation.
- Developers must implement additional instrumentation.



Google Cloud

When using a Service Mesh, some individual traces are collected automatically, so that you can see the service topography or understand the latency per service.

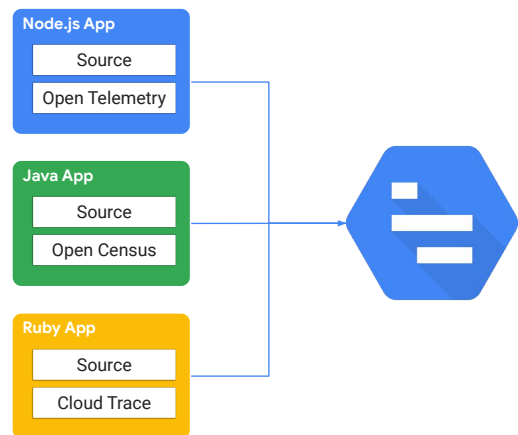
However, if you want to understand an end-to-end trace that takes places across multiple microservices, you need to instrument your application.

In the instrumentation process, you use libraries in your code to add unique headers to your requests, so that you can provide a trace of the request.

There are some frameworks, protocols, and databases that provide automatic instrumentation. However, most of the time, you will need to provide additional instrumentation.

Libraries and backends for tracing instrumentation

- Use Open Telemetry, Open Census, or Cloud Trace library for instrumentation.
- Use Cloud Trace as the backend for visualizing and reporting.
- Works with Compute Engine, Kubernetes Engine, etc.



Google Cloud

To instrument your applications, there are a variety of options. Tracing implementation is evolving rapidly, so the details may change. Across all languages, Google is transitioning to make use of OpenTelemetry libraries.

However, at the moment, when it comes to library support by language, use:

- OpenTelemetry for Node.js, Go.
- OpenCensus for Python, Java, Go, PHP.
- And Cloud Trace for Ruby, Node.js, C# (both ASP.NET core and regular ASP.NET).

Once traces are collected by your library of choice, you can visualize requests in the Google Cloud Console using the Cloud Trace dashboards.