The Service Mesh Era: Architecting, Securing and Managing Microservices with Istio

Google Cloud Whitepaper

Google Cloud

Google Cloud

Contents

Google Cloud

Contents

How we got here

Chances are, like many forward-thinking Google Cloud customers, you believe that building applications inside containers in a microservices architecture is the future of software development. We do too: single-purpose applications running inside containers combine to form the bulk of Google's services, and it's this model that enables Google developers to create massively distributed systems that power some of the world's most trafficked and reliable applications.

For enterprises too, moving away from proprietary, monolithic application architectures to containers and microservices is one of the best ways to improve developer efficiency and achieve separation of concerns within an IT organization. However, this model brings its own sets of issues. Container networking, in particular, has long been a problem in the enterprise. How can containerized microservices talk to one another in an efficient fashion? How can developers and operators see into those communications, to understand performance characteristics and troubleshoot issues? How do you ensure that that communication is authorized and secure? And perhaps most of all, how do you get that networking functionality in a microservices way, i.e., without building it directly into the application like you would with a software monolith?

Like other hyperscale organizations, we built ourselves an internal service mesh to handle interprocess communication and coordination, and it works very well. But if you've solved the problem in a way that only you can use it, you haven't really solved the problem. In 2016, we joined forces with IBM and Lyft, the creator of the Envoy proxy, to create a service mesh that would be useful to enterprise organizations who were starting to build out containerized, microservices environments. The result was Istio, which provides the key functionality you need for effective microservice communication: traffic management, telemetry, and service discovery and authentication. As an open-source project, it's free to use and to contribute to, and is supported by a vibrant community.

This eBook groups a collection of blog posts written in early 2019 by Google network and security experts. Here, you'll find in-depth explanations of what Istio is, tactical advice on how to deploy it and use it in your environment, and a great customer example. This is foundational information for anyone who wants to gain a deep understanding of service mesh and how it is architected. It's also great background reading for anyone interested in learning more about Anthos, Google Cloud's hybrid and multi-cloud platform, and the new Anthos Service Mesh, a fully managed Istio-based offering for Anthos customers. We hope you enjoy this eBook, and look forward to talking with you about how service mesh can help take your application modernization efforts to the next level.

- Jennifer Lin, Director of Product Management, Google Cloud

Welcome to the service mesh era

By Megan O’Keefe, Developer Programs Engineer

Adopting a microservices architecture brings a host of benefits, including increased autonomy, flexibility, and modularity. But the process of decoupling a single-tier monolithic application into smaller services introduces new obstacles: How do you know what's running? How do you roll out new versions of your services? How do you secure and monitor all those containers?

To address these challenges, you can use a service mesh: software that helps you orchestrate, secure, and collect telemetry across distributed applications. A service mesh transparently oversees and monitors all traffic for your application, typically through a set of network proxies that sit alongside each microservice. Adopting a service mesh allows you to decouple your application from the network, and in turn, allows your operations and development teams to work independently.

Alongside IBM, Lyft, and others, Google launched Istio in 2016 as an open-source service mesh solution. Built on the high-performance Envoy proxy, Istio provides a configurable overlay on your microservices running in Kubernetes. It supports end-to-end encryption between services, granular traffic and authorization policies, and unified metrics—all without any changes to your application code.

Istio’s architecture is based on trusted service mesh software used internally at Google for years. And much in the same way we brought Kubernetes into the world, we wanted to make this exciting technology available to as many users as possible. To that end, in December, 2018, we announced the beta availability of Istio on GKE, an important milestone in our quest to deliver a managed, mature service mesh that you can deploy with one click. You have also heard from us about our vision for Anthos Service Mesh—a service mesh that spans both the Cloud and on-prem and Anthos, our open platform that lets you run your applications, without modifications, on your existing on-prem hardware or in the cloud.

With all that’s happening in the service mesh world, we thought we'd take a step back and dive deep into how you can use Istio right now, in production. This eBook is designed to be a practical resource on Istio and service mesh. In it, we will cover all kinds of user perspectives, from developers and cluster operators to security administrators and SREs. Through real use cases, we will shed light on the "what" and "how" of service mesh—but most importantly, how Istio can help you deliver immediate business value to your customers.

To start, let's explore why Istio matters in the context of other ongoing shifts in the cloud-native ecosystem: towards abstraction from infrastructure, towards automation, and towards a hybrid cloud environment.

Automate everything

The world of modern software moves quickly. Increasingly, organizations are looking for ways to automate the development process from source code to release, in order to address business demands and increase velocity in a competitive landscape. Continuous delivery is a pipeline-based approach for automating application deployments, and represents a key pillar in DevOps best practices.

Istio's declarative, CRD-based configuration model integrates seamlessly with continuous delivery systems, allowing you to incorporate Istio resources into your deployment pipelines. For example, you can configure your pipeline to automatically deploy Istio VirtualServices to manage traffic for a canary deployment. Doing so lets you leverage Istio's powerful features—from granular traffic management to in-flight chaos testing—with zero manual intervention. With its declarative configuration model, Istio can also work with modern GitOps workflows, where source control serves as the central source of truth for your infrastructure and application configuration.

Serverless, with Istio

Serverless computing, meanwhile, transforms source code into running workloads that execute only when called. Adopting a serverless pattern can help organizations reduce infrastructure costs, while allowing developers to focus on writing features and delivering business value.

Serverless platforms work well because they decouple code and infrastructure. But most of the time, organizations aren't only running serverless workloads— they also have stateful applications, including microservices apps on Kubernetes infrastructure. To address this, several open-source, Kubernetes-based serverless platforms have emerged in the open-source community. These platforms allow Kubernetes users to deploy both serverless functions and traditional Kubernetes applications onto the same cluster.

Last year, we released Knative, a new project that provides a common set of building blocks for running serverless applications on Kubernetes. Knative includes components for serving requests, handling event triggers, and building containerized functions from source code. Knative Serving is built on Istio, and brings Istio's telemetry aggregation and security-by-default to serverless functions.

Knative aims to become the standard across Kubernetes-based serverless platforms. Further, the ability to treat serverless functions as services in the same way you treat traditional containers will help provide much-needed uniformity between the serverless and Kubernetes worlds. This standardization will allow you to use the same Istio traffic rules, authorization policies, and metrics pipelines across all your workloads.

Build once, run anywhere

As Kubernetes matures, users are increasingly adopting more complex cluster configurations. Today, you might have several clusters, not one. And those clusters might span hybrid environments, whether in the public cloud, in multiple clouds, or on-prem. You might also have microservices that have to talk to single-tier applications running in virtual machines, or service endpoints to manage and secure, or functions to spin up across clusters.

Driven by the need for lower latency, security, and cost savings, the era of multi-cloud is upon us, introducing the need for tools that span both cloud and on-prem environments.

Released with 1.0, Istio Multicluster is a feature that allows you to manage a cross-cluster service mesh using a single Istio control plane, so you can take advantage of Istio's features even with a complex, multicluster mesh topology. With Istio Multicluster, you can use the same security roles across clusters, aggregate metrics, and route traffic to a new version of an application. The multicluster story gets easier in 1.1, as the new Galley component helps synchronize service registries between clusters.

Anthos is another example of the push towards interoperable environments, combining solutions including Google Kubernetes Engine, GKE On-Prem, and Istio, towards the ultimate goal of creating a seamless Kubernetes experience across environments.

What's next?

This eBook will cover Istio's key features: traffic management, authentication, security, observability, IT administration, and infrastructure environments. Whether you're just getting started with Istio, or working to move Istio into your production environment, we hope it will have something relevant and actionable for you.

We're excited to have you along for the ride on our service mesh journey!

Advanced applicaton deployments and traffic management with Istio on GKE

By Megan O'Keefe, Developer Programs Engineer

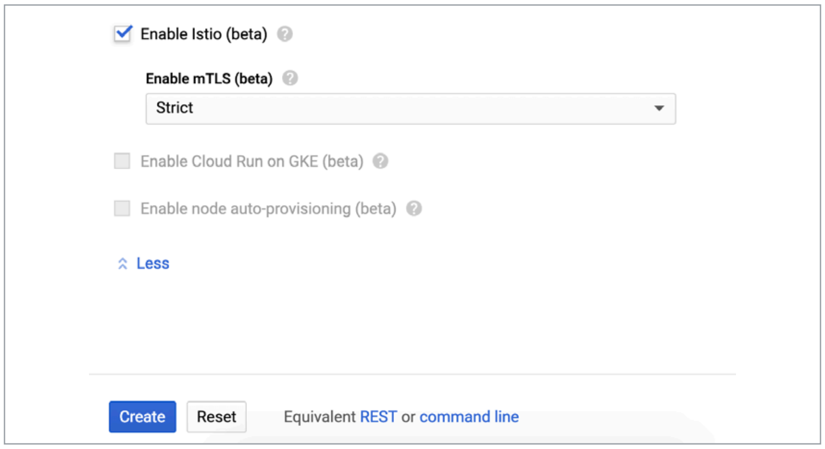
In the first section we explored the benefits of using a service mesh, and placed Istio in context with other developments in the cloud-native ecosystem. In this section, we'll dive into the "what" and "how" of installing and using Istio with a real application. Our goal is to demonstrate how Istio can help your organization decrease complexity, increase automation, and ease the burden of application management on your operations and development teams.

Install with ease; update automatically

When done right, a service mesh should feel like magic: a platform layer that "just works," freeing up your organization to use its features to secure, connect, and observe traffic between your services. So if Istio is a platform layer, why doesn't it come preinstalled with Kubernetes? If Istio is middleware, why are we asking developers to install it?

At Google, we are working on simplifying adoption by providing a one-click method of installing Istio on Kubernetes. Istio on GKE, the first managed offering of its kind, is an add-on for Google Kubernetes Engine (GKE) that installs and upgrades Istio's components for you—no YAML required. With Istio on GKE, you can create a cluster with Istio pre-installed, or add Istio to an existing cluster.

Installing Istio on GKE is easy, and can be done either through the Cloud Console or the command line. The add-on supports mutual TLS, meaning that with a single check-box, you can enforce end-to-end encryption for your service mesh.



Enable Istio (beta) [✓]

Enable mTLS (beta): Strict

Enable Cloud Run on GKE (beta) [ ]

Enable node auto-provisioning (beta) [ ]

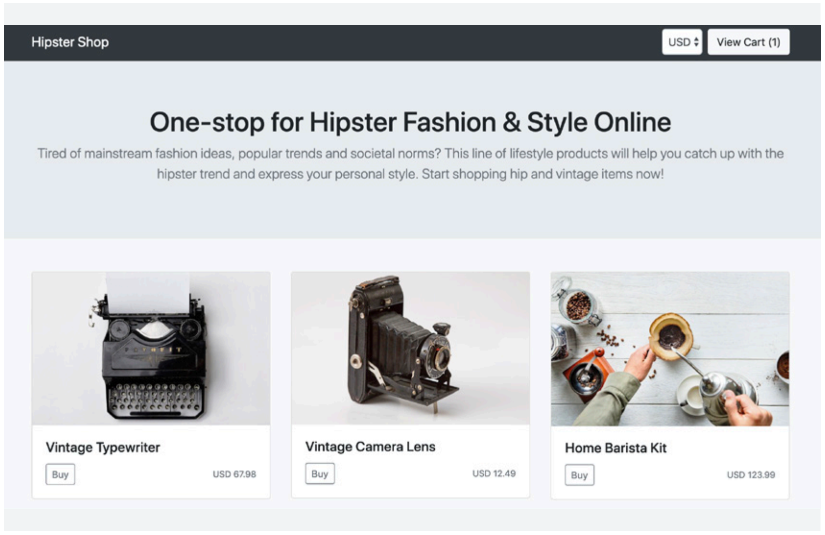
The Companion Demo

This eBook features many examples and real-life scenarios to better illustrate what Istio on GKE can do. To expand on the examples here, we have created a tutorial in GitHub. It includes step-by-step instructions, screenshots, and even YAML for a variety of tasks. Follow along as you go through the text here, or save it for later, but we hope you'll check it out.

ACCESS THE TUTORIAL HERE

Once enabled, Istio on GKE provisions the Istio control plane for you, and enables Stackdriver integrations. You get to choose into which namespaces, if any, the Istio sidecar proxy is injected.

Now that we have Istio installed on a GKE cluster, let's explore how to use it with a real application. For this example, we'll use the Hipster Shop demo, a microservices-based web application.



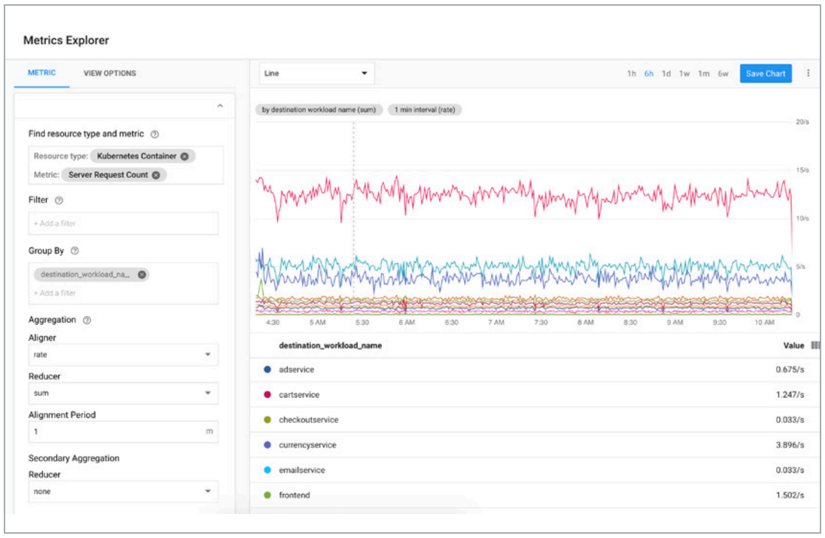
Hipster Shop
USD
View Cart (1)
One-stop for Hipster Fashion & Style Online
Tired of mainstream fashion ideas, popular trends and societal norms? This line of lifestyle products will help you catch up with the hipster trend and express your personal style. Start shopping hip and vintage items now!
Vintage Typewriter
Buy
USD 67.98
Vintage Camera Lens
Buy
USD 12.49
Home Barista Kit
Buy
USD 123.99

While this sample app has multiple components, we'll focus on Product Catalog, which serves the list of products above.

Zero effort Stackdriver: Monitoring, logging, and tracing

When you use Istio on GKE, the Stackdriver Monitoring API is provisioned automatically, along with an Istio adapter that forwards service mesh metrics to Stackdriver. This means that you have access to Istio metrics right away, alongside hundreds of existing GCP and GKE metrics.

Stackdriver includes a feature called the Metrics Explorer, which allows you to use filters and aggregations together with Stackdriver's built-in metrics to gain new insights into the behavior of your services. The example below shows an Istio metric (requests per second) grouped across each microservice in our sample application.

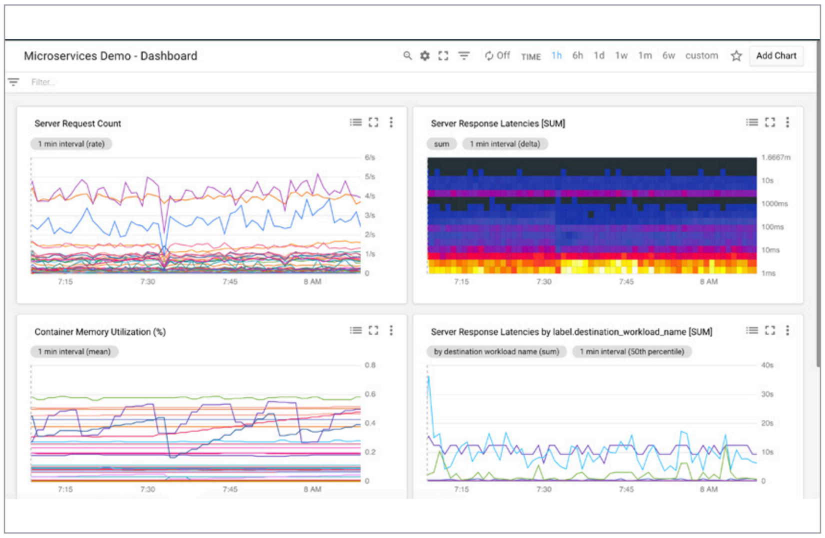


Find resource type and metric
Resource type: Kubernetes Container
Metric: Server Request Count
Filter
+ Add a filter
Group By
destination_workload_na...
+ Add a filter
Aggregation
Aligner: rate
Reducer: sum
Alignment Period: 1 m
Secondary Aggregation
Reducer: none

	destination_workload_name	Value
●	adservice	0.675/s
●	cartservice	1.247/s
●	checkoutservice	0.033/s
●	currencyservice	3.896/s
●	emailservice	0.033/s
●	frontend	1.502/s

Stackdriver's Metrics Explorer

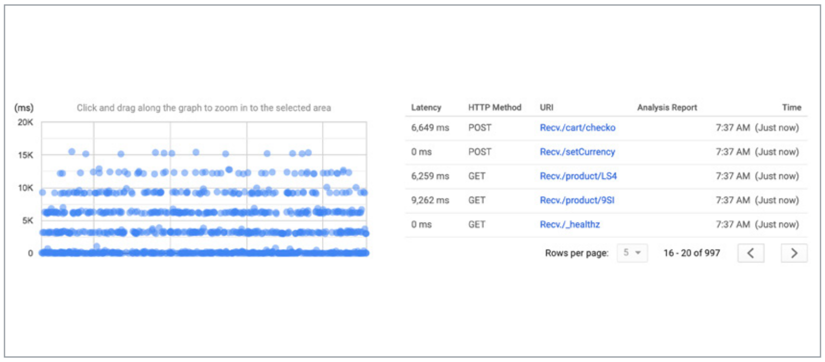
You can add any Metrics Explorer chart to a new or existing Stackdriver Dashboard. Using Dashboards, you can also combine Istio metrics with your application metrics, giving you a more complete view into the status of your application.


Microservices Demo - Dashboard
Off
TIME
1h
6h
1d
1w
1m
6w
custom
Add Chart
Filter...
Server Request Count
1 min interval (rate)
Server Response Latencies [SUM]
sum
1 min interval (delta)
Container Memory Utilization (%)
1 min interval (mean)
Server Response Latencies by label.destination_workload_name [SUM]
by destination workload name (sum)
1 min interval (50th percentile)
7:15
7:30
7:45
8 AM

Stackdriver Dashboard sample

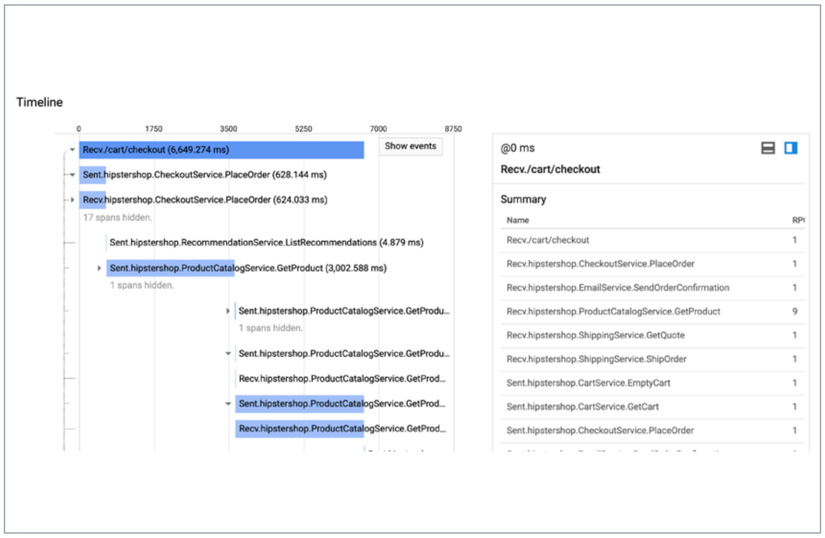
You can also use Stackdriver Monitoring to set SLOs using Istio metrics—for example, latency, or non-200 response codes. Then, you can set Stackdriver Policies against those SLOs to alert you when a policy reaches a failing threshold. In this way, Istio on GKE sets up your organization with SRE best practices, out of the box.

Istio on GKE also makes tracing easy. With tracing, you can better understand how quickly your application is handling incoming requests, and identify performance bottlenecks. When Stackdriver Trace is enabled and you've instrumented tracing in your application, Istio automatically collects end-to-end latency data and displays it in real-time to the GCP Console.


(ms)
Click and drag along the graph to zoom in to the selected area
20K
15K
10K
5K
0

Latency	HTTP Method	URI	Analysis Report	Time
6,649 ms	POST	Recv./cart/checko		7:37 AM (Just now)
0 ms	POST	Recv./setCurrency		7:37 AM (Just now)
6,259 ms	GET	Recv./product/LS4		7:37 AM (Just now)
9,262 ms	GET	Recv./product/9SI		7:37 AM (Just now)
0 ms	GET	Recv./_healthz		7:37 AM (Just now)

Rows per page: 5 16 - 20 of 997



Timeline

Recv./cart/checkout (6,649.274 ms)
Sent.hipstershop.CheckoutService.PlaceOrder (628.144 ms)
Recv.hipstershop.CheckoutService.PlaceOrder (624.033 ms)
17 spans hidden.
Sent.hipstershop.RecommendationService.ListRecommendations (4.879 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (3,002.588 ms)
1 spans hidden.
Sent.hipstershop.ProductCatalogService.GetProdu…
1 spans hidden.
Sent.hipstershop.ProductCatalogService.GetProdu…
Recv.hipstershop.ProductCatalogService.GetProd…
Sent.hipstershop.ProductCatalogService.GetProd…
Recv.hipstershop.ProductCatalogService.GetProd…

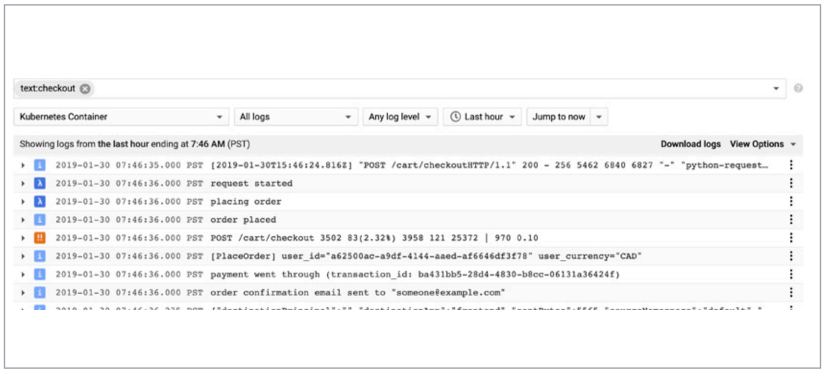
@0 ms
Recv./cart/checkout

Summary

Name	RP
Recv./cart/checkout	1
Recv.hipstershop.CheckoutService.PlaceOrder	1
Recv.hipstershop.EmailService.SendOrderConfirmation	1
Recv.hipstershop.ProductCatalogService.GetProduct	9
Recv.hipstershop.ShippingService.GetQuote	1
Recv.hipstershop.ShippingService.ShipOrder	1
Sent.hipstershop.CartService.EmptyCart	1
Sent.hipstershop.CartService.GetCart	1
Sent.hipstershop.CheckoutService.PlaceOrder	1

Stackdriver Dashboard sample

On the logging front, Stackdriver also creates a number of logs-based metrics, which let you extract latency information from log entries, or record the number of log entries that contain a particular message. You can also develop custom metrics to keep track of logs that are particularly important to your organization.



text:checkout

Kubernetes Container | All logs | Any log level | Last hour | Jump to now

Showing logs from the last hour ending at 7:46 AM (PST)

```
2019-01-30 07:46:35.000 PST [2019-01-30T15:46:24.816Z] "POST /cart/checkoutHTTP/1.1" 200 - 256 5462 6840 6827 "-" "python-request…
2019-01-30 07:46:36.000 PST request started
2019-01-30 07:46:36.000 PST placing order
2019-01-30 07:46:36.000 PST order placed
2019-01-30 07:46:36.000 PST POST /cart/checkout 3502 83(2.32%) 3958 121 25372 | 970 0.10
2019-01-30 07:46:36.000 PST [PlaceOrder] user_id="a62500ac-a9df-4144-aaed-af6646df3f78" user_currency="CAD"
2019-01-30 07:46:36.000 PST payment went through (transaction_id: ba431bb5-28d4-4830-b8cc-06131a36424f)
2019-01-30 07:46:36.000 PST order confirmation email sent to "someone@example.com"
```

Stackdriver Logs Viewer

Then, using the Logs Viewer, you can export the logs to Google Cloud data solutions, including Cloud Storage and BigQuery, for storage and further analysis.

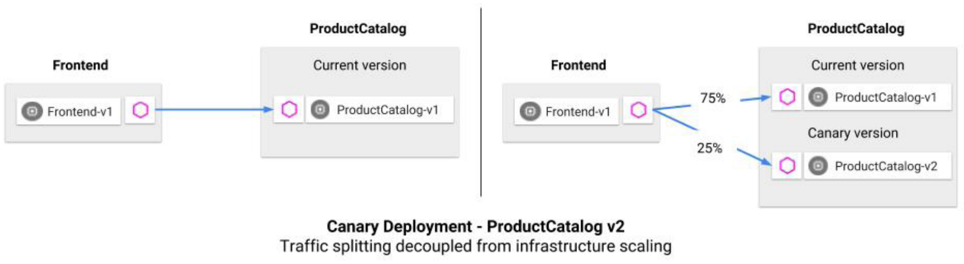
Traffic management and visualization

In addition to providing visibility into your service mesh, Istio supports fine-grained, rule-based traffic management. These features give you control over how traffic and API calls flow between your services.

As the first section in this book explains, adopting a service mesh lets you decouple your applications from the network. And unlike Kubernetes services, where load balancing is tethered to the number of running pods, Istio allows you to decouple traffic flow from infrastructure scaling through granular percentage-based routing.

Let's run through a traffic routing example, using a canary deployment.

A **canary deployment** routes a small percentage of traffic to a new version of a microservice, then allows you to gradually roll it out to the whole user base, while phasing out and retiring the old version. If something goes wrong during this process, traffic can be switched back to the old version.



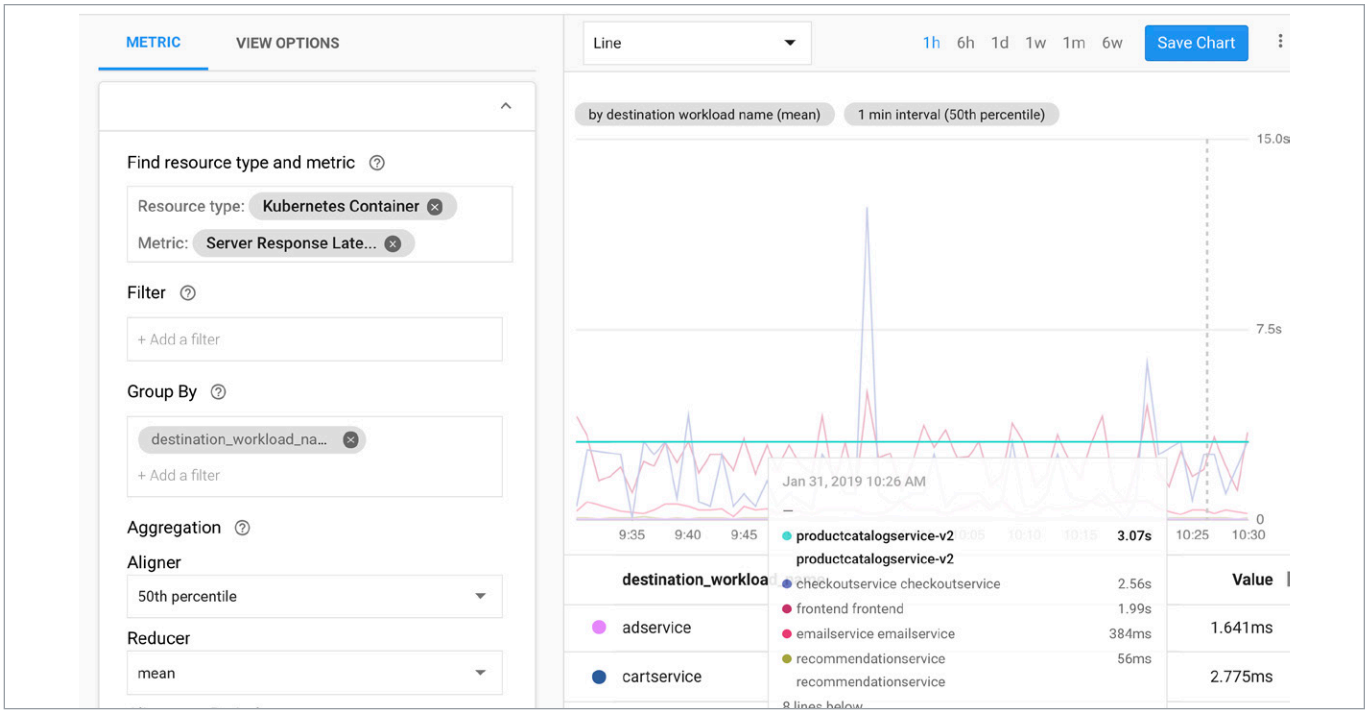
Canary deployment example

In this example, we create a new version of the ProductCatalog microservice. The new version ("v2") is deployed to Kubernetes alongside the working ("v1") deployment.

Then, we create an Istio VirtualService (traffic rule) that sends 25% of ProductCatalog traffic to v2. We can deploy this rule to the Kubernetes cluster, alongside our application. With this policy, no matter how much production traffic goes to ProductCatalog—and how many pods scale up as a result—Istio ensures that the right percentage of traffic goes to the specified version of ProductCatalog.

We'll also use another feature of Istio and Envoy: for demo purposes, we inject a three-second latency into all ProductCatalog v2 requests.

Once the canary version is deployed to GKE, we can open Metrics Explorer to see how ProductCatalog v2 is performing. Notice that we are looking at the Istio Server Response Latency metric, and we have grouped by "destination workload name"—this tells us the time it takes for each service to respond to requests.



Find resource type and metric

Resource type: Kubernetes Container

Metric: Server Response Late...

Filter

Group By

destination_workload_na...

Aggregation

Aligner: 50th percentile

Reducer: mean

Jan 31, 2019 10:26 AM

● productcatalogservice-v2 productcatalogservice-v2	3.07s
● checkoutservice checkoutservice	2.56s
● frontend frontend	1.99s
● emailservice emailservice	384ms
● recommendationservice recommendationservice	56ms

8 lines below

destination_workloa	Value
● adservice	1.641ms
● cartservice	2.775ms

Istio Server Response Latency metric example

Here, we can see ProductCatalog v2's injected three-second latency. From here, it's easy to roll back from v2 to v1. We can do this by updating the Istio VirtualService to return 100% of traffic to v1, then deleting the v2 Kubernetes deployment.

Although this example demonstrates a manual canary deployment, often you'll want to automate the process of promoting a canary deployment: increasing traffic percentages, and scaling down the old version. Open-source tools like Flagger can help automate percentage-based traffic shifting for Istio.

Istio supports many other traffic management rules beyond traffic splitting, including content-based routing, timeout and retries, circuit breaking, and traffic mirroring for testing in production. Like in this canary example, these rules can be defined with the same declarative Istio building blocks.

Want more?

To learn more about Istio, Stackdriver, and traffic management, see:

- Drilling down into Stackdriver Service Monitoring (GCP blog)
- Incremental Istio Part 1, Traffic Management (Istio blog)

Securing your environment with Istio

By Samrat Ray, Senior Product Manager

As we've discussed, we're seeing accelerated adoption of containers, Kubernetes, and microservices, driven by the desire to increase developer productivity, deployment velocity, and operational scalability. The adoption of these technologies and paradigms results in a dynamic production environment where containerized workloads are deployed to a pool of hosts (or VMs) and are typically ephemeral. Further, there is a significant increase in the network surface area within the perimeter as microservices are exposed as network endpoints. Lastly, IP-based network flow logs are no longer sufficient to demonstrate compliance to internal and external stakeholders. Thus, there is a need to reconsider the traditional approach to network security for such environments.

One of Istio's more important value propositions, then, is how it can effectively increase the security of your modern production environments, without sacrificing developer productivity.

Given the proliferation of threats within the production network and the increased points of privileged access, it is increasingly necessary to adopt a zero-trust network security approach for microservices architectures. This approach requires that all accesses are strongly authenticated, authorized based on context, logged, and monitored… and the controls must be optimized for dynamic production environments.

Istio on GKE helps with these security goals in a few ways.

It provides defense in depth; it layers on top of your existing layer 3 network security controls to provide an independent layer of network security. It provides the foundation for implementing zero-trust network security, where trust and access are determined by strongly authenticated peer identities and additional context of the request rather than by presence inside the same network perimeter. It enables you to demonstrate compliance using access logs that capture service identities and layer 7 attributes than just 5-tuple information. Finally, it lets you configure this security by default—you don't need to change your application code or infrastructure to turn it on.

The best way to demonstrate the value of the Istio security layer is to show it in action. Specifically, let's look at how Istio on GKE can help you adopt a zero-trust security approach through authentication—who a service or user is, and whether we can trust that they are who they say they are—and authorization—what specific permissions this user or service has. Together, these protect your environment from security threats like access using stolen credentials and replay attacks, and keep your sensitive data safe.

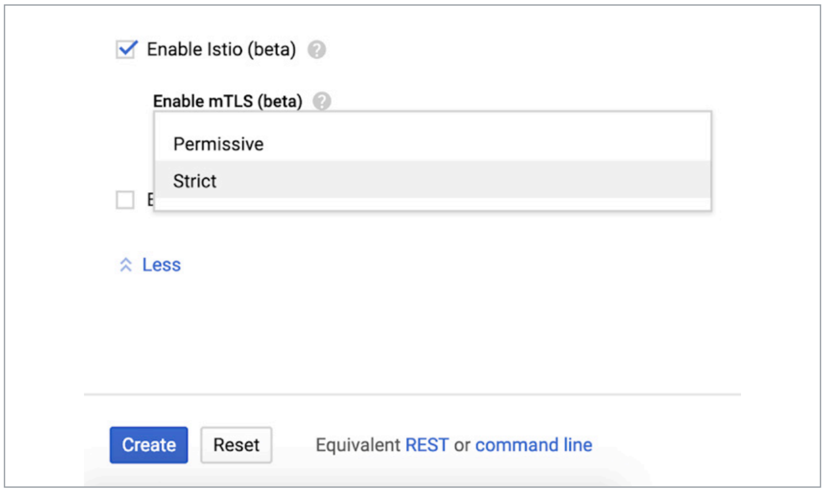
Authenticaton with mutual TLS

One of the anti-patterns with microservices authentication is to rely on a bearer token, e.g. a JWT, to authenticate a peer. Bearer tokens can be stolen—from the source, destination or through man-in-the middle attacks—and replayed, enabling lateral movement of threats and privilege escalation.

An approach to mitigate this risk is to ensure that peers are only authenticated using non-portable identities. Mutual TLS authentication (mTLS) ensures that peer identities are bound to the TLS channel and cannot be replayed. It also ensures that all communication is encrypted in transit, and mitigates the risk of man-in-the middle attacks and replay attacks by the destination service. While mutual TLS helps strongly identify the network peer, end user identities (or identity of origin) can still be propagated using bearer tokens like JWT.

While mTLS is an important security tool, it's often difficult and time consuming to manage. To start, you have to create, distribute, and rotate keys and certificates to a large number of services. You then need to ensure you are properly implementing mTLS on all of your clients and servers. And when you adopt a microservices architecture, a manual approach is hard to scale to an increasing number of services. Finally, traditional X.509 certificates used for TLS identify domains and are not optimized for authenticating workloads.

Istio on GKE supports mTLS and can help ease many of these challenges. It automates key and certificate management, including generation, distribution, and rotation, and its certificates identify the workload using a Service Identity (vs. the host or domain). Istio uses the Envoy sidecar proxy to enforce mTLS and requires no code changes to implement. The approach of using Service Identities enables workload portability across clusters and clouds with no changes to the access control policies. You can easily enable Istio mTLS on GKE today, by choosing an mTLS option from a simple dropdown menu.



Enabling Istio on GKE from a dropdown menu

Permissive mode is the default. It allows services in your mesh to accept both mTLS authenticated and non-mTLS traffic. In this mode, existing clients that are not enabled for mTLS can continue accessing the service while mTLS is incrementally rolled out across your environment. Istio clients can be configured to enable mTLS by changing the destination rule. The objective should be to lock down a port to only mTLS enabled clients over time using the strict mTLS mode.

When you select strict mTLS mode, Istio on GKE enforces mTLS for all accesses to services; all calls are encrypted and authenticated based on the certificate-based identity. While this is an ideal end state, you need to ensure that all clients to the service are mTLS enabled, otherwise you may break your existing application.

Many organizations choose to first enable permissive mTLS for the entire namespace, and then transition to strict mode on a service-by-service or even port-by-port basis. You can also override client-side defaults with destination-specific rules. This is one of the major benefits of Istio—it lets you incrementally adopt mTLS, or turn it on and off for your whole mesh. This incremental adoption model lets you implement the security features of mTLS without breaking existing applications.

To enable mTLS incrementally you first need a Policy for inbound traffic, and a DestinationRule for outbound. Enabling mTLS for all services in a namespace is a very similar process. Just set up another policy and DestinationRule, this time for the full namespace, then execute it.

With mTLS enabled, you now have a strong authenticated peer identity that can be used for access control (authorization). You can also rely on additional context such as the end user (also known as origin) identity for granting access. Istio can validate JSON web tokens so that you can safely build authorization policies that rely on authenticated claims in the token—such as the end user identity—in addition to authenticated channel attributes.

Authorization tools to protect your data

Another key component to building a zero-trust network security posture is to ensure that access to sensitive data is only granted to authorized clients and users.

Istio Authorization—which is built on Kubernetes role-based access control (RBAC)—provides access control for the services in your mesh based on multiple attributes in the request context. With Istio authorization, you can constrain who can access a service endpoint based on the certificate-based identity of the peer, as well as claims in a JWT. Further, Istio authorization is a layer 7 policy and be used to grant specific permissions based on the URL.

At its most basic, Istio RBAC maps subjects to roles. An Istio authorization policy involves groups of permissions for accessing services (the ServiceRole specification), and then determining which users, groups, and services gets those specific access permissions (ServiceRoleBinding). A ServiceRole contains a list of permissions, while a ServiceRoleBinding assigns a specific ServiceRole to a list of subjects.

When you're configuring Istio authorization policies, you can specify a wide range of different groups of permissions, and grant access to them at the level that makes sense, down to the user level.

Network access logging with service level information

With these features set up, you can also address an increasingly important aspect of security: demonstrating to both internal and external stakeholders that all services and accesses are in compliance with required network security policies. Istio on GKE's robust logging and metrics collection features can help provide this.

We hope this tour of Istio's security features demonstrated how Istio makes it easier for you to implement and manage a comprehensive microservices security strategy that makes sense for your organization.

Want more?

- Istio security overview
- Mutual TLS Deep Dive
- Using Istio to Improve End-to-End Security
- Micro-Segmentation with Istio Authorization

Using Istio and Stackdriver to build an SRE service

By Sandeep Parikh, Cloud Native Advocate

Now that we've talked about the benefits of using a service mesh, using Istio for application deployments and traffic management, and how Istio helps you achieve your security goals, we're going to dig further into monitoring, tracing, and service-level objectives. This chapter will demonstrate how you can use Istio to level up your own Site Reliability Engineering (SRE) practices for workloads running in Kubernetes.

The pillars of SRE

At Google, we think SRE is so important, we wrote a book on it. Let's quickly review what the term really means to us. The goal of SRE is to improve service reliability and performance and, in turn, the end-user experience. Conceptually, that means proactively managing and incorporating three main components: service level objectives (SLOs), service level agreements (SLAs), and service level indicators (SLIs). We can summarize these as follows:

- SLOs: targets you set for overall service health
- SLAs: promises you make about your service's health (so, they often include specific SLOs)
- SLIs: metrics that you use to define the SLO targets

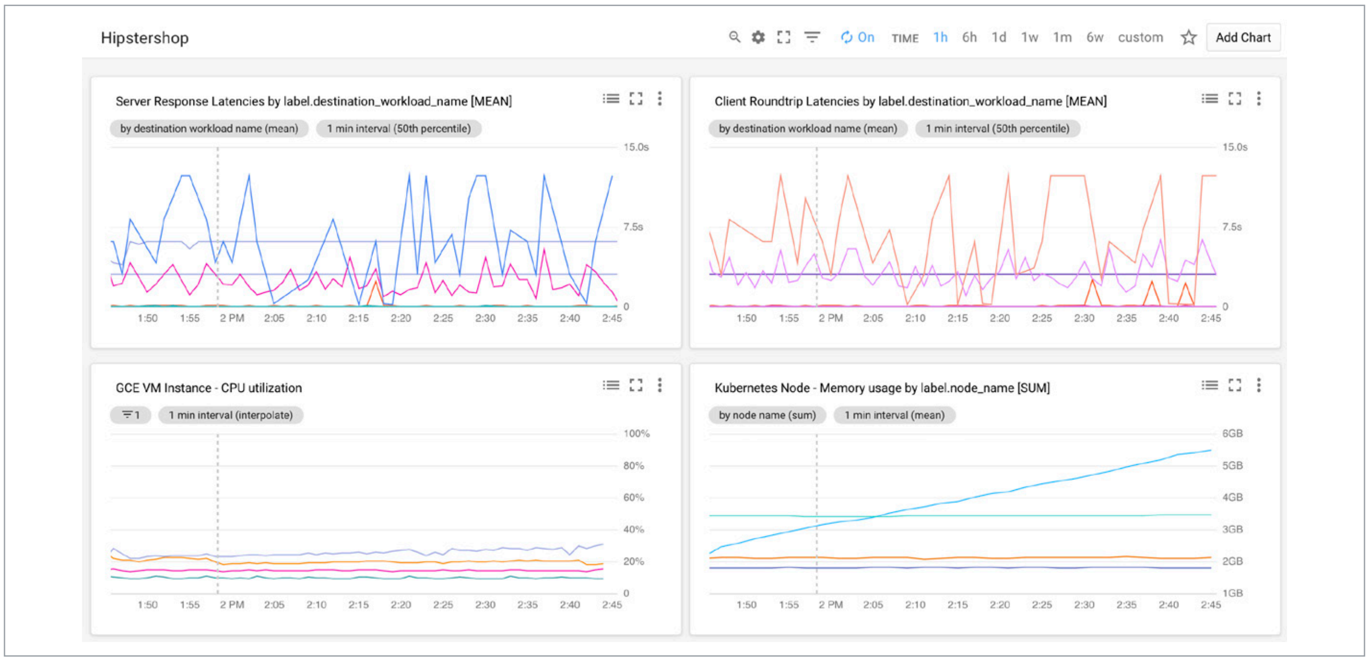
How do we take these ideas from conceptual to practical? To provide guarantees about your service (SLAs), you need to set targets (SLOs) that incorporate several key service metrics (SLIs). That's where Istio and Stackdriver come in.

Surfacing application metrics with Stackdriver Monitoring

In the “Advanced application deployments and traffic management” section, we talked about how Google Kubernetes Engine (GKE), Istio, and Stackdriver are integrated right out of the box. This means that Stackdriver Monitoring gives you the ability to monitor a dozen Istio-specific metrics without any special configuration or setup. These include metrics for bytes sent and received, request counts, and roundtrip latencies, for both clients and servers.

Once you create a Stackdriver Workspace, you can immediately head to the Metrics Explorer and start visualizing those metrics from Istio. Without any manual instrumentation, Istio provides a significant amount of telemetry information for your workloads—enough to begin thinking about which of those metrics (Istio-provided or GCP-provided) could make for useful SLIs.

Which SLIs make the most sense will depend on your application and deployments. For Istio-enabled workloads we usually recommend creating Dashboards containing metrics that will reflect your end users’ experience, but these typically include service request counts and service request/response latency broken out by Kubernetes Namespaces and/or Pods. For completeness, your Dashboards may also include other important monitoring metrics like GKE node availability along with CPU or RAM usage. The example Dashboard below provides a combined overview of cluster and service health


Hipstershop
On
TIME
1h
6h
1d
1w
1m
6w
custom
Add Chart
Server Response Latencies by label.destination_workload_name [MEAN]
by destination workload name (mean)
1 min interval (50th percentile)
15.0s
7.5s
0
1:50
1:55
2 PM
2:05
2:10
2:15
2:20
2:25
2:30
2:35
2:40
2:45
Client Roundtrip Latencies by label.destination_workload_name [MEAN]
by destination workload name (mean)
1 min interval (50th percentile)
15.0s
7.5s
0
GCE VM Instance - CPU utilization
1
1 min interval (interpolate)
100%
80%
60%
40%
20%
0
Kubernetes Node - Memory usage by label.node_name [SUM]
by node name (sum)
1 min interval (mean)
6GB
5GB
4GB
3GB
2GB
1GB

Stackdriver Dashboard sample

After identifying the appropriate SLIs for your deployment, the next step is to create alerting policies that notify you or your team about any problems in your deployment. Alerting policies in Stackdriver are driven by metrics-based conditions that you define as part of the policy. In addition, you can combine multiple metrics-based conditions to trigger alerts when any or all of the conditions are met.

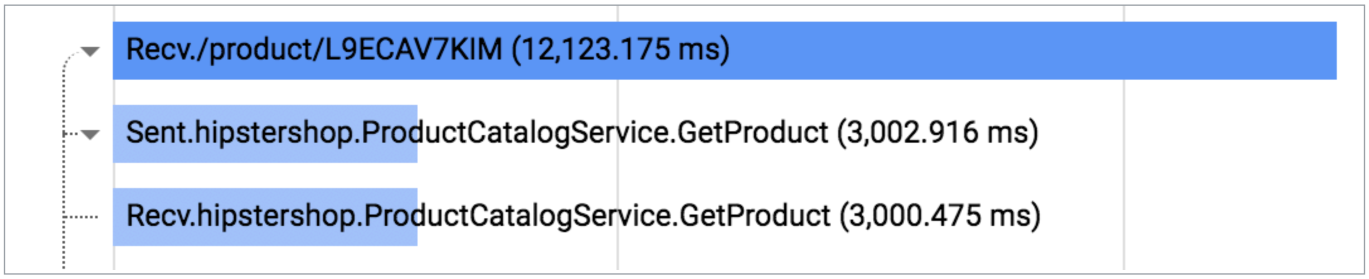
With a working metrics dashboard and alerting policies in place, you're now at a point where you can keep track of the health of each of your services. But what happens when you see an alert? What if it turns out that one of your services has a server response latency that's much higher than expected—and that it's happening on a pretty regular basis? The good news is that now you know there's a problem; but now the challenge is tracking it down.

Digging into requests using Stackdriver Trace

So far we've been talking about monitoring, but Istio's telemetry support also includes the ability to capture distributed tracing spans directly from individual services.

Distributed tracing allows you to track the progression of a single user-driven request, and follow along as it is handled by other services in your deployment.

Once the Stackdriver Trace API is enabled in your GCP project, Istio's telemetry capture components start sending trace data to Stackdriver, where you can view it in the trace viewer. Without instrumenting any of your services or workloads, Istio captures basic span information, like HTTP requests or RPCs.


Recv./product/L9ECAV7KIM (12,123.175 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (3,002.916 ms)
Recv.hipstershop.ProductCatalogService.GetProduct (3,000.475 ms)

Detail of Stackdriver Trace viweer

This is a good start, but to truly diagnose our example (higher than expected server response latency) we'll need more than just the time it takes to execute a single service call. To get that next level of information, you need to instrument your individual services so that Istio (and by extension, Stackdriver) can show you the complete code path taken by the service called. Using OpenCensus tracing libraries, you can add tracing statements to your application code. We recommend instrumenting tracing for critical code paths that could affect latency, for example, calls to databases, caches, or internal/external services. The following is a Python example of tracing within a Flask application:

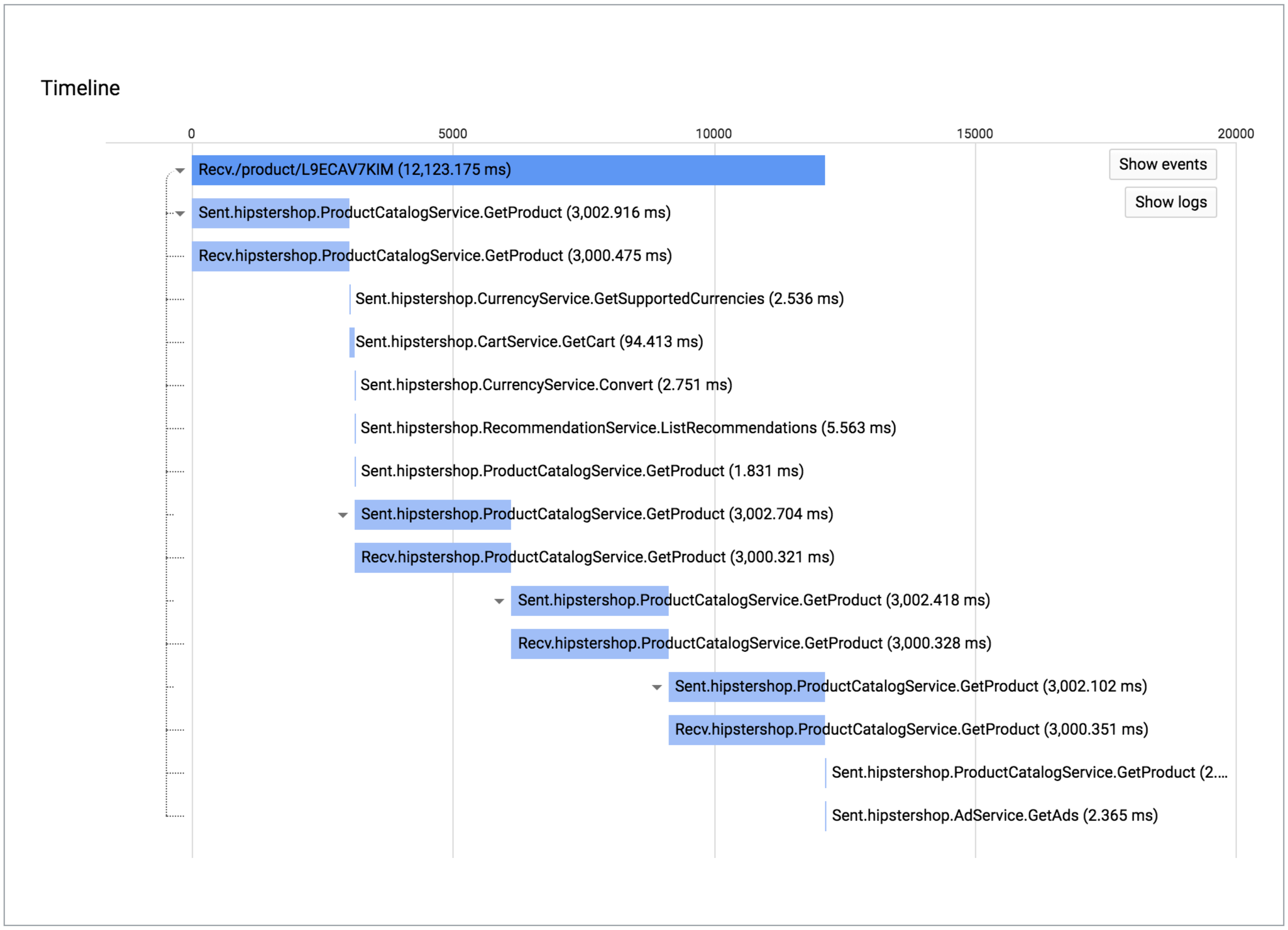
```
@app.route('/index.html', methods=['GET'])
def index():
    tracer = app.config['TRACER']
    tracer.start_span(name='index')

    # Add up to 1 sec delay, weighted toward zero
    time.sleep(random.random() ** 2)
    result = "Tracing requests"

    tracer.end_span()
    return result
```

Python example of tracing within a Flask application

We instrumented our sample microservices demo using OpenCensus libraries. Once you've deployed that app and the built-in load generator has had a chance to generate some requests, you can head over to Stackdriver Trace to examine one of the higher latency service calls.



Recv./product/L9ECAV7KIM (12,123.175 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (3,002.916 ms)
Recv.hipstershop.ProductCatalogService.GetProduct (3,000.475 ms)
Sent.hipstershop.CurrencyService.GetSupportedCurrencies (2.536 ms)
Sent.hipstershop.CartService.GetCart (94.413 ms)
Sent.hipstershop.CurrencyService.Convert (2.751 ms)
Sent.hipstershop.RecommendationService.ListRecommendations (5.563 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (1.831 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (3,002.704 ms)
Recv.hipstershop.ProductCatalogService.GetProduct (3,000.321 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (3,002.418 ms)
Recv.hipstershop.ProductCatalogService.GetProduct (3,000.328 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (3,002.102 ms)
Recv.hipstershop.ProductCatalogService.GetProduct (3,000.351 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (2....
Sent.hipstershop.AdService.GetAds (2.365 ms)

Stackdriver Trace viewer

As you can see in the diagram above, Stackdriver Trace lets you examine the complete code path and determine the root of the high latency call.

Examining application output using Stackdriver Logging

The final telemetry component that Istio provides is the ability to direct logs to Stackdriver Logging. By themselves, logs are useful for examining application status or debugging individual functions and processes. And with Istio's telemetry components sending metrics, trace data, and logging output to Stackdriver, you can tie all of your application's events together. Istio's Stackdriver integration allows you to quickly navigate between monitoring dashboards, request traces, and application logs. Taken together, this information gives you a more complete picture of what your app is doing at all times, which is especially useful when an incident or policy violation occurs.

Stackdriver Logging's integration comes full circle with Stackdriver Monitoring by giving you the ability to create metrics based on structured log messages. That means you can create specific log-based metrics, then add them to your monitoring dashboards right alongside your other application monitoring metrics. And Stackdriver Logging also provides additional integrations with other parts of Google Cloud—specifically, the ability to automatically export logs to Cloud Storage or BigQuery for retention and follow-on ad-hoc analysis, respectively. Stackdriver Logging also supports integration with Cloud Pub/Sub where each output log entry is exported as an individual pub/sub message, which can then be analyzed in real-time using Cloud Dataflow or Cloud Dataproc.

Coming soon: SLOs and service monitoring using Stackdriver

So far we've reviewed the various mechanisms Stackdriver provides to assess your application's SLIs; Stackdriver now also provides native support for setting SLOs against your specific service metrics. That means you will be able to set specific SLO targets for the metrics you care about, and Stackdriver will automatically generate SLI graphs, and track your target compliance over time. If any part of your workload violates your SLOs, you are immediately alerted to take action.

SRE isn’t about tools; it’s a lifestyle

Think of SRE as a set of practices, and not as a specific set of tools or processes. It’s a principled approach to managing software reliability and availability, through the constant awareness of key metrics (SLIs) and how those metrics are measured against your own targets (SLOs)—which you might use to provide guarantees to your customers (via SLAs). When you combine the power of Istio and Stackdriver and apply it to your own Kubernetes-based workloads, you end up with an in-depth view of your services and the ability to diagnose and debug problems before they become outages.

As you can see, Istio provides a number of telemetry features for your deployments. And when combined with deep Stackdriver integration, you can develop and implement your own SRE practices.

driver and apply it to your own Kubernetes-based workloads, you end up with an in-depth view of your services and the ability to diagnose and debug problems before they become outages.

As you can see, Istio provides a number of telemetry features for your deployments. And when combined with deep Stackdriver integration, you can develop and implement your own SRE practices.

Want more?

We haven’t even begun to scratch the surface on defining SRE and these terms, for more information, check out:

- SRE Fundamentals: SLIs, SLAs, and SLOs
- SLOs, SLIs, SLAs, oh my - CRE life lessons

Then for more on the specific topics we discussed here, please see:

- Istio and Stackdriver tutorial
- Advanced application deployments and traffic management with Istio on GKE
- SRE fundamentals: SLIs, SLAs, and SLOs
- Drilling down into Stackdriver Service Monitoring

Istio's role in the future of hybrid cloud

By Megan O'Keefe, Developer Programs Engineer, Google Cloud

Up to now, we've mostly discussed topics related to Istio on GKE in a single environment. In this section, we'll go up a level and investigate using Istio across environments, and how Istio can help you unlock the power of hybrid cloud.

Why hybrid?

Hybrid cloud can take on many forms. Typically, hybrid cloud refers to operating across public cloud and private (on-premises) cloud, and multi-cloud means operating across multiple public cloud platforms.

Adopting a hybrid- or multi-cloud architecture could provide a ton of benefits for your organization. For instance, using multiple cloud providers helps you avoid vendor lock-in, and allows you to choose the best cloud services for your goals. Using both cloud and on-premises environments allows you to simultaneously enjoy the benefits of the cloud (flexibility, scalability, reduced costs) and on-prem (security, lower latency, hardware re-use). And if you're looking to move to the cloud for the first time, adopting a hybrid setup lets you do so at your own pace, in the way that works best for your business.

Based on our experience at Google, and what we hear from our customers, we believe that adopting a hybrid service mesh is key to simplifying application management, security, and reliability across cloud and on-prem environments—no matter if your applications run in containers, or in virtual machines. Let's talk about how to use Istio to bring that hybrid service mesh into reality.

Hybrid Istio: a mesh across environments

One key feature of Istio is that it provides a services **abstraction** for your workloads (Pods, Jobs, VM-based applications). When you move to a hybrid topology, this services abstraction becomes even more crucial, because now you have not just one, but many environments to worry about.

When you adopt Istio, you get all the management benefits for your microservices on one Kubernetes cluster—visibility, granular traffic policies, unified telemetry, and security. But when you adopt Istio across multiple environments, you are effectively giving your applications new superpowers. Because Istio is not just a services abstraction on Kubernetes. Istio is also a way to **standardize networking** across your environments. It's a way to

centralize API management and decouple JWT validation from your code. It's a fast-track to a **secure, zero-trust network** across cloud providers.

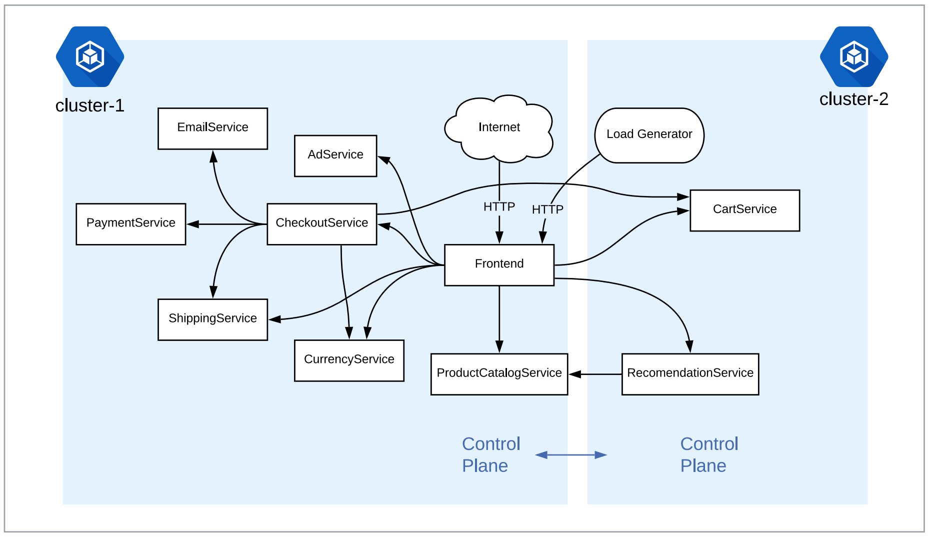
So how does all this magic happen? Hybrid Istio refers to a set of sidecar Istio proxies (Envoys) that sit next to all your services across your environments—every VM, every container—and know how to talk to each other across boundaries. These Envoy sidecars might be managed by one central Istio control plane, or by multiple control planes running in each environment.

Let's dive into some examples.

Multicluster Istio, one control plane

One way to enable hybrid Istio is by configuring a remote Kubernetes cluster that "calls home" to a centrally-running Istio control plane. This setup is useful if you have multiple GKE clusters in the same GCP project, but Kubernetes pods in both clusters need to talk to each other. Use cases for this include: production and test clusters through which you canary new features, standby clusters ready to handle failover, or redundant clusters across zones or regions.

The diagram below illustrates a scenario where we spin up two GKE clusters in the same project, but across two different zones (us-central and us-east). We install the Istio **control plane** on one cluster, and Istio's **remote** components (including the sidecar proxy injector) on the other cluster. From there, we can deploy a sample application spanning both Kubernetes clusters.



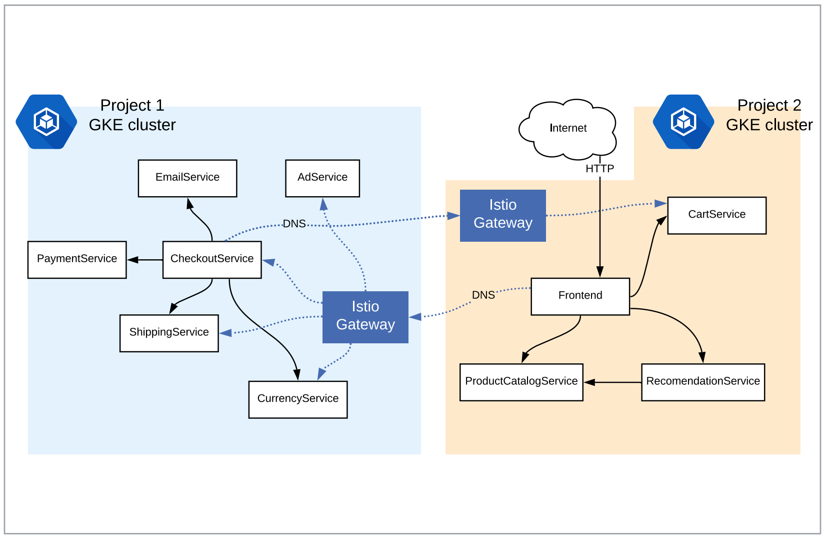
The exciting thing about this single control plane approach is that we didn't have to change anything about how our microservices talk to each other. For instance, the Frontend can still call CartService with a local Kubernetes DNS name `(cartservice:port)`. This DNS resolution works because GKE pods in the same GCP project belong to the same virtual network, thus allowing direct pod-to-pod communication across clusters.

Multicluster Istio, two control planes

Now that we have seen a basic multi-cluster Istio example, let's take it a step further.

Say you're running applications on-prem and in the cloud, or across cloud platforms. For Istio to span these different environments, pods inside both clusters must be able to cross network boundaries.

The following diagram uses two Istio control planes—one per cluster—to form a single, two-headed logical service mesh. Rather than having the sidecar proxies talk directly to each other, traffic moves across clusters using Istio's Ingress Gateways. An Istio Gateway is just another Envoy proxy, but it's specifically dedicated for traffic in and out of a single-cluster Istio mesh.



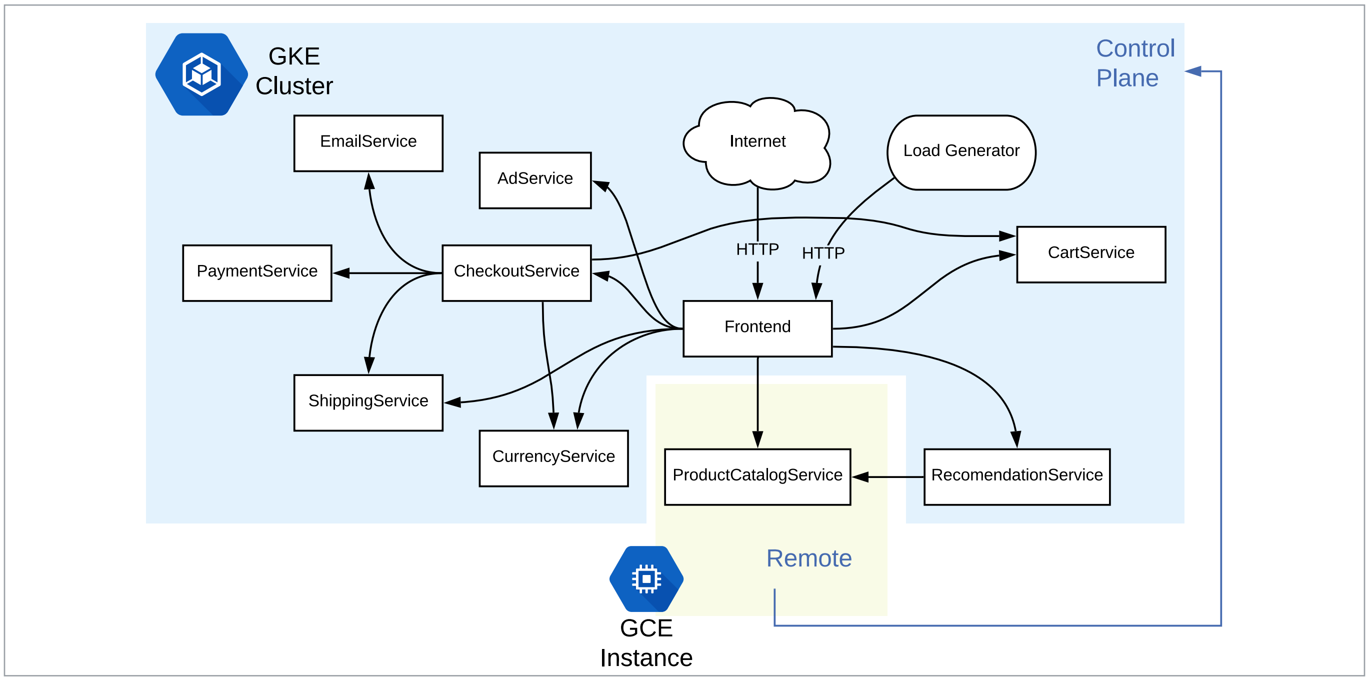
For this setup to work across a network partition, each Istio control plane has a special domain name server (**DNS**) configuration. In this dual-control-plane topology, Istio installs a secondary DNS server (CoreDNS) which resolves domain names for services outside of the local cluster. For those outside services, traffic moves between the Istio Ingress Gateways, then onwards to the relevant service.

Unlike the first demo, this dual control-plane Istio setup **does not require a flat network** between clusters. This means you can have overlapping GKE pod CIDRs between your clusters. All that this setup requires is that the Istio Gateways are exposed to the Internet. In this way, the services inside each cluster can stay safe in their own respective environments.

In the demo for this topology, we show how this installation works, then how to configure the microservices running across both clusters to talk to each other. We do this through the Istio ServiceEntry resource. For instance, we deploy a service entry for the Frontend (cluster 2) into cluster 1. In this way, cluster 1 knows about services running in cluster 2.

Adding a virtual machine to the Istio mesh

Many organizations use **virtual machines** (VMs) to run their applications, instead of (or in addition to) containers. If you're using VMs, you can still enjoy the benefits of an Istio mesh. The diagram below illustrates the topology when you integrate a Google Compute Engine instance with Istio running on GKE. We deploy the same application as before. But this time, one service (ProductCatalog) runs on an external VM, outside of the Kubernetes cluster.



This GCE VM runs a minimal set of Istio components to be able to communicate with the central Istio Control Plane. We then deploy an Istio ServiceEntry object to the GKE cluster, which logically adds the external ProductCatalog service to the mesh.

This Istio configuration model is useful because now, all the other microservices can reference ProductCatalog as if it were running internal to the Kubernetes cluster. From here, you could even add Istio policies and rules for ProductCatalog as if it were running in Kubernetes; for instance, you could enable mutual TLS for all inbound traffic to the VM.

Note that while this scenario uses a Google Cloud VM for demo purposes, you could run this same example on bare metal, or with an on-prem VM. In this way, you can bring Istio's modern, cloud-native principles to virtual machines running anywhere.

Building the hybrid future

We hope that one or more of these hybrid Istio demos resonates with the way your organization runs applications today. But we also understand that adopting a service mesh like Istio means taking on complexity and installation overhead, in addition to any complexity associated with moving to microservices and Kubernetes. In that case, adopting a hybrid service mesh is even more complex, because you're dealing with different environments, each with their own technical specifications.

Here at Google Cloud, we are dedicated to helping you simplify your day-to-day cloud operations with a consistent, modern, cross-platform setup. It's why we created Istio on GKE, which provides a one-click install of Istio on Google Kubernetes Engine (GKE). And it's the driving force behind our work on Anthos, which helps your organization **move to (and across) the cloud**—at your own pace, and in the way that works best for you. Anthos relies on an **open cloud stack**—Kubernetes and Istio—to emphasize portability.

Istio in practice: How Auto Trader UK uses Istio and Google Kubernetes Engine to drive change

By Eimear Hennessy, Head of Corporate UKI, Google Cloud

Brand-new or second-hand? Diesel or electric? Convertible or SUV? Buying a car means choosing from a plethora of options, and that can be hard for some people to navigate. As a result, retailers are constantly rethinking their technology offerings—which means digital transformation must move just as fast.

As the UK's largest digital automotive marketplace and the country's 16th largest website, Auto Trader UK prides itself on how simple it is for its consumers and retailers alike to buy and sell cars on their platform. To do it, they rely on Istio on GKE. Istio has helped to enable visibility, increase agility and effectively secure their production environment, without sacrificing developer productivity.

"Over the last 14 months we have worked directly with Google's Kubernetes product managers with ongoing access to the Google Cloud Istio teams."

Russell Warman, Head of Infrastructure, Auto Trader UK

Improving security and agility with GKE and Istio

Since 2013, Auto Trader has been a completely digital business, and they are now the UK's market leader, with 55 million cross-platform visits every month and an audience four times larger than their nearest competitor. In total, they offer 300 applications including valuation tools, detailed reviews of dealerships and new cars, and integrations with car finance and insurance partners.

Looking ahead

Since adopting Kubernetes and Istio, Auto Trader has seen significant gains in efficiency. For example, they are 75 percent more efficient in terms of their compute resources, without impacting performance. Auto Trader has also lowered their monthly bill and can now predict future spending more accurately. Istio, meanwhile, has helped them improve security and visibility, with no extra developer effort or training needed.

Auto Trader is now planning to complete its migration to the public cloud. With about a third of its services already running in production on GCP, they plan to migrate their remaining workloads over the next year to ensure everything is built, managed and monitored in the same way.

Auto Trader is certainly in the driver's seat when it comes to their Istio journey.

To find out more about the other benefits of migrating to GCP, both from an operational and development perspective, including improved security, see the Auto Trader UK case study.

"From a business perspective, migrating to Google Cloud Platform means we can get ideas up and running quickly, enabling us to build brilliant new products, helping us to continue to lead in this space."

Russell Warman, Head of Infrastructure, Auto Trader UK

Appendix of resources

Chapter 1: Welcome to the service mesh era

General information...

Kubernetes users, get ready for the next chapter in microservices management

An update on container support on Google Cloud Platform

Accelerate your app delivery with Kibernetes and Istio on GKE

Research: What sets top performing DevOps teams apart

CNCF Cloud Native Interactive landscape

Knative Serving: Kubernetes-based, scale-to-zero, request-driven compute

The future is multi-cloud: Here’s how to get ready

Chapter 2: Advanced application deployments and traffic management with Istio on GKE

General Information...

Drilling down into Stackdriver Service Monitoring

Learn more about DevOps and SRE

How to...

Create an Istio on GKE cluster

Add Istio on GKE to an existing cluster

Installing Istio on GKE

What's installed

Choose a security option

Choose namespaces where sidecar is injected

More on the Envoy sidecar proxy

Stackdriver Support

Enabling Stackdriver monitoring

Istio adapters for interfacing with infrastructure backends

Istio metrics

GKE metrics

Creating a chart for a Stackdriver Dashboard

Create a new alerting policy

Stackdriver logs-based metrics

Creating custom metrics

Exporting with the Logs Viewer

Using exported logs (including in Cloud Storage and BigQuery)

Performing a rolling update with Kubernetes

Traffic management with Istio on GKE: Decoupling traffic flow from infrastructure scaling

Traffic splitting: Canary deployments with Istio on GKE

Istio-supported traffic management rules: canary deployments, content-based routing, timeout and retries, circuit breaking, traffic mirroring

Chapter 3: Securing your environment with Istio

How to...

Chapter 4: Using Istio and Stackdriver to build an SRE service

General information…

Google’s Site Reliability Engineering Book

How to…

Istio-specific metrics

A list of GCP-provided metrics

Creating a Stackdriver Workspace

Creating Stackdriver charts and dashboards

Introduction to alerting

Alerting policies in-depth

Tracing

Enabling the Stackdriver Trace API

Capturing span information

Using OpenCensus trace libraries

Setting up Stackdriver Trace for Python

Analyzing a pub/sub message in real-time using

Chapter 5: Istio's role in the future of hybrid cloud

General information...

Mesh expansion: integrate VMs and bare metal hosts into an Istio mesh deployed on Kubernetes

Accelerate your app delivery with Kubernetes and Istio on GKE

Kubernetes users, get ready for the next chapter in microservices management

How to...

Kubernetes concepts: Pods, Jobs

Using Istio in multi-cluster deployments

Warm standby clusters for recovery to GCP

Redundant clusters across zones or regions

Using multiple Istio control planes

Networking within a Kubernetes cluster

Enabling mutual TLS