

Anthos in Action

Curated by Antonio Gulli, Michael Madison, and Scott Surovich





Anthos in Action


Curated by
Antonio Gulli, Michael Madison, and Scott Surovich

©2021 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

© Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.

 Manning Publications Co.
20 Baldwin Road Technical
PO Box 761
Shelter Island, NY 11964

ISBN: 9781633439320

contents

preface *vii*

acknowledgements *viii*

- Chapter 1 Overview 1**
by Aparna Sinha
- Chapter 2 Cloud is a new computing stack 8**
by Phil Taylor
- Chapter 3 Anthos, the one single pane of glass 20**
by Melika Golkaram
- Chapter 4 Anthos Service Mesh:
Security and observability at scale 40**
by Onofrio Petragallo
- Chapter 5 Operations management in Anthos 64**
by Jason Quek

preface

Whether you are a software developer, a businessperson interested in technology, a technology researcher, or simply a person who wants to know more about where the world of computers and people is going, it is important to know more about Anthos. This mini-book is an extended abstract of the full edition which will be published soon.

More than a decade ago, the idea that “software is eating the world” gained common currency. Even then, it was clear that computer code had reached a level of high sophistication and complexity, along with a lower level of computing cost. As a result, a large number of industrial and consumer processes and experiences could be created as software. We had seen it happen to music, video, architectural design, and much more.

Since then, software sophistication has continued to rise and enabling costs to fall, particularly with the popularization of public cloud computing. Software has indeed replaced more once-analog processes, increasing their market availability and improving their performance. Additionally, near real-time data gathering and data analysis have become a critical part of computing. The number of software developers around the world has grown to tens of millions of people. Demand for industry-changing software has increased too; it’s common now for organizations to aspire to be “software first,” so that they might create and innovate products and services faster than they ever could in the analog world.

Anthos was conceived of in 2018 with the goal of broadening and accelerating these positive trends. If data is critical in planning, why shouldn’t a company be able to access its data at all times, wherever it is held? If computing takes place both on corporate premises and in one or more clouds, shouldn’t all that computing be put to the greatest possible use? If software developers have ambitions to create more useful innovations, shouldn’t they be able to deploy, manage, and change them where and how they want? What would it look like if we could read and respond to the world in the most effective way possible?

These are real issues that can also be stated in contemporary business and technical terms. Whether people need to simply build a new mobile application, or take on the complexity of a hybrid cloud back-end services on-premise connected to a new cloud-based SaaS application with real-time event processing and continuous machine-learning inference processing at scale, the productivity of development teams is a top priority for any organization. At the same time, many development tools and closed platforms, manual processes and legacy APIs have come to increase complexity and hinder progress. Working at Google Cloud, we heard our customers tell us how challenging it is to run a computing stack on-premise and in multiple cloud environments, and how easier things could be running on the cloud and leveraging open and scalable APIs, while paying only for what is consumed. Anthos is the solution to these business problems.

Early on, we spent a lot of time on a single principle: Let's abstract the less-productive majority of the effort in running computing infrastructure operations and remove habitual constraints that are not really core to building and running software services. In this more self-sustaining environment, developers have more time and energy to write application code and ensure their system behaves well in production. Operational people are able to concentrate better on key areas like cost, efficiency, and security.

Anthos had the benefit of Google's own success in both Site Reliability Engineering and Continuous Integration/Continuous Development toolchains at scale, affording best practice observability and service commitments, which has been crucial to its success in creating so many successful software products at scale. These are methods applicable now for many companies, particularly as we address their existing technology and data practices in a world where computing is increasingly foundational.

This began the creation of a customer-focused, run-anywhere distributed development platform, one able to automate utilization to reduce expensive dependencies and autonomously seek to work in a pre-declared desired production state. The automation that software seeks in so many other areas was brought to software development and operations itself, enabling development teams to move faster, working on the quick deployment on snippets of code that contribute to a totality, with better agility and product quality. No need for lengthy deployment, complicated upgrades and manual patching, tools integrations, and manual steps to keep the system operational.

In the process of building and deploying Anthos, it has become increasingly clear that the division among computing on-premises, in the cloud, and now at a compute-rich edge of mobile devices and sensors no longer makes sense. The different stacks, different APIs for infrastructure provisioning, and different operational and security models are relics, often sustained more by vendors than by market need. It is the customer's data, and it is the developer's code—shouldn't they be able to get it where and how they want?

Software development is a team sport, and a complex one with many moving parts that all need to work well together. At the center of scaling software there are APIs, the ability for various components to set the way they interact. If done right, depen-

dencies are harmonized and present minimal friction. It was not initially clear, however, which APIs can be trusted to help scale, automate, and secure apps. Security is frequently based on an on-premise networking model of perimeter firewalls and compliance guardrails that no longer make sense in a world of pervasive computing. Working from home, for example, requires access to corporate and production networks, or differential application access. Customers, likewise, want to access business software in all sorts of ways. Add to that integration of identity systems, compliance blueprints, audit logging, and encryption, and it becomes really challenging to ensure uniformity of software workloads in a secure and safe environment.

From our earliest prototypes, Anthos has received positive feedback for both vision and the way we have iterated to meet developer needs. Usable products gain momentum from pent-up demand; the solutions often seem obvious in retrospect, usually because they respect the task people are trying to accomplish. Engineers have understood the merits of Anthos and have successfully deployed it with better results than they did with their previous stack.

The ease of use in Anthos represents a lot of hard work, and deep attention to details reducing the steps required to run workloads safely and rapidly. Software service calls to APIs that happen in minutes, not hours or days, simply make more sense. We've addressed on-prem and cloud-based security challenges, built policy guardrails that customers tell us work for them, and improved automation in every release.

You will see the breadth and depth of the platform in this book. Our teams, along with our many contributing partners and beta customers, have thought carefully about the design options, in the process finding path breaking concepts for things like service mesh-based security, GitOps declarative compliance with config manager, and scale to zero capabilities in order to optimize costs.

I have no doubt there will be more to come, based on both the momentum Anthos is seeing and the overall behavior of the computing industry. The acceleration of a software-based focus in every organization means that every day there are more developers who want to write code with the greatest possible impact, drawing the most from their individual and team creativity.

Away from its positive impact for our many customers, the success of Anthos is a reminder of the true power of the computing revolution. When Gordon Moore noted in 1965 that the transistors on a semiconductor appeared to double roughly every two years, he prophesied a revolution that has touched our lives in countless ways. Although people have debated whether Moore's Law, as it pertains to chips, is dead, there is no doubt that the living implications of Moore's Law have continued apace. Moore's Law is also an observation that computing wants to be smaller, cheaper, and available in more places; it's startling that an implication in the age of mainframes has proven true in the progression to mini- and microcomputers, to PCs, smartphones, and now sensors connected to clouds, the basis of ambient computing.

Anthos, a unifying and automating creation, sustains and broadens this deep historic trend. It is not the end stage, however. The journey is only getting started; the

wave of application modernization is accelerating with the rise of technology innovation in every industry and aspect of our lives. We are proud to be a part of this positive trend, which can bring new resources and understanding to billions of people. Our success in this journey remains our key principle of focus on our customers: With Anthos, we help developers write better software faster, automating everything we can from initial code to production, enabling them to find further insight and new creation.

—Eric Brewer,
VP Infrastructure,
and Google Fellow

acknowledgements

Anthos in action would not be possible without the work of countless Fellow Travelers (https://en.wikipedia.org/wiki/Fellow_traveller).

The Curators would like to thank the Authors for their contribution. In alphabetic order: Ameer Abbas, Amita Kapoor, Aparna Sinha, Eric Brewer, Giovanni Galloro, Jarosław Gajewski, Jason Quek, Konrad Cłapa, Kyle Bassett, Melika Golkaram, Michael Madison, Onofrio Petragallo, Patricia Florissi', Phil Taylor, Scott Surovich, Some Authors have been selected for the book's preview published at Google Next 21, while others' contributions will be made available for the book's full edition publication.

THE AUTHORS AND THE CURATORS

The Authors and Curators would like to thank reviewers for their thoughtful input, discussion and for review. In alphabetical order thanks to Ady Degany, Alex Mattson, Alon Pildus, Amina Mansur, Amr Abdelrazik, Anil Dhawan, Ankur Jain, Anna Berenberg, Antoine Larmanjat, Barbara Stanley, Ben Good, Brian Grant, Brian Kaufman, Chen Goldberg, Christopher Bussler, Eric Johnson, Eyal Manor, Gabriele Di Piazza, Harish Yakumar, Issy Ben-Shaul, Jamie Duncan, Jason Polites, Jeff Reed, Jennifer Lin, Jerome Simms, John Abel, Jonathan Donaldson, Kavitha Radhakrishnan, Kevin Shatzkamer, Laura Cellerini, Leonid Vasetsky, Louis Ryan, Maluin Patel, Manu Batra, Marco Ferrari, Marcus Johansson, Massimo Mascaro, Maulin Patel, Micah Baker, Michael Abd-El-Malek, Michelle Au, Miguel de Luna, Mike Ensor, Nina Kozinska, Nima Badiy, Norman Johnson, Purvi Desai, Quan To, Raghu Nandan, Rich Rose, Richard Seroter, Ron Avnur, Scott Penberthy, Steren Giannini, Tariq Islam, Tim Hockin, Tony Savor, Vanna Stano, Vinay Anand, Will Grannis, Yoav Reich, Zach Casper, Zach Seils.

This book would not have been possible without a massive collaboration among Curators, Authors, Reviewers, Editors, and Marketing. We are particularly thankful to Arun Ananthampalayam, JP Schaengold, Maria Bledsoe, Richard Seroter, and Yash Kamath from Google, and Doug Rudder, Aleksandar Dragosavljević, Gloria Lukos from Manning. Thanks for your continuous support and inspiration.

All the Authors and Curators' royalties will be donated to Charities.

1 *Overview*

by Aparna Sinha

This chapter covers

- Anatomy of a modern application
- Accelerating software development with Anthos
- Standardizing operations at scale with Anthos
- Origins at Google
- How to read this book

1.1 *Anatomy of a modern application*

Software has been eating the world for a while. As consumers, we are used to applications that make it faster, smarter, and more efficient for us to do things like calling a cab or depositing a paycheck. Increasingly, our health, education, entertainment, social life, and employment are all enhanced by modern software applications. At the other end of those applications is a chain of enterprises, large and small, that deliver these improved experiences, services, and products. Modern applications are deployed not just in the hands of consumers, but also at points along this enterprise supply chain. Major transactional systems in many traditional industries such as retail, media, financial services, education, logistics, and more are gradually being replaced by modern micro-services that auto-update frequently, scale efficiently, and incorporate more real-time intelligence. New digital-first startups are using this

opportunity to disrupt traditional business models, while enterprise incumbents are rushing to modernize their systems so they can compete and avoid disruption.

This begs the question: What is a modern application? When you think of software that has improved your life, perhaps you think of applications that are interactive, fast (low latency), connected, intelligent, context aware, reliable, secure, and easy to use on any device. As technology advances, the capabilities of modern applications, such as the level of security, reliability, awareness, and intelligence advance as well. For example, new development frameworks, such as React and Angular, have greatly enhanced the level interactivity of applications, new runtimes like node.js have increased functionality. Modern applications have the property of constantly getting better through frequent updates. On the backend *these applications often comprise a number of services that are all continuously improving*. This modularity is attained by breaking the older “monolith” pattern for writing applications, where all the various functions were tightly coupled to each other.

Applications written as a set of modular or micro-services, have several benefits: constituent services can be evolved independently or replaced with other more scalable or otherwise superior services over time. Also, the modern microservices pattern is better at separating concerns and setting contracts between services making it easier to inspect and fix issues. This approach to writing, updating, and deploying applications as “micro-services” that can be used together but also updated, scaled, and debugged independently is at the heart of modern software development. In this book we refer to this pattern as “modern” or “cloud native” application development. The term cloud native applies here because the microservices pattern is well suited to run on distributed infrastructure or cloud. Microservices can be rolled out incrementally, scaled, revised, replaced, scheduled, rescheduled, and bin packed tightly on distributed servers creating a highly efficient, scalable, reliable system that is responsive and frequently updated.

Modern applications can be written “greenfield”, from scratch, or refactored from existing “brownfield” applications by following a set of architectural and operational principles. *The end goal of application modernization is typically revenue acceleration, and often this involves teams outside IT, in line-of-business (LoB) units.* IT organizations at most traditional enterprises have historically focused on reducing costs and optimizing operations, while cost reduction and optimized operations can be by-products of application modernization, they are not the most important benefit. Of course, the modernization process itself requires up-front investment. Anthos is Google Cloud’s platform for application modernization in hybrid and multi-cloud environments. It provides the approach and technical foundation needed to attain high ROI application modernization. An IT strategy that emphasizes modularity through APIs, micro-services, and cloud portability combined with a developer platform that automates reuse, experiments, and cost-efficient scaling along with secure, reliable operations are the basic critical prerequisites for successful application modernization.

The first part of Anthos is a modern developer experience that accelerates line-of-business application development. It is optimized for refactoring brownfield apps and

writing microservices and API based applications. It offers unified local, on-prem, and cloud development with event-driven automation from source to production. It gives developers the ability to write code rapidly using modern languages and frameworks with local emulation and test, integrated CI/CD, and supports rapid iteration, experimentation, and advanced roll-out strategies. The Anthos developer experience emphasizes cloud APIs, containers and functions but is customizable by enterprise platform teams. A key goal of the Anthos developer experience is for teams to release code multiple times a day, thereby enhancing both velocity and reliability. Anthos features built-in velocity and ROI metrics to help development teams measure and optimize their performance. Data-driven benchmarks are augmented with pre-packaged best practice blueprints that can be deployed by teams to achieve the next level of performance.

The second part of Anthos is an operator experience for central IT. Anthos shines as the uniquely scalable, streamlined way to run operations across multiple clouds. This is enabled by the remarkable foundation of technology invented and honed at Google over a period of 20 years, for running services with extremely high reliability on relatively low-cost infrastructure. This is achieved through standardization of the infrastructure using a layer of abstraction comprising Kubernetes, Istio, Knative, and several other building blocks along with Anthos-specific extensions and integrations for automated configuration, security, and operations. The operator experience on Anthos offers advanced security and policy controls, automated declarative configuration, highly scalable service visualization and operations, and automated resource and cost management. It features extensive automation, measurement, and fault avoidance capabilities for high availability, secure service management across cloud, on-prem, edge, virtualized, and bare metal infrastructure.

Enterprise and small companies alike find that multi-cloud and edge is their new reality, either organically or through acquisitions. Regulations in many countries require proven ability to migrate applications between clouds and demonstrate failure tolerance with support for sovereignty. Non-regulated companies find multi-cloud necessary for providing developers' choice and access to innovative services. Opportunities for running services and providing greater intelligence at the edge add further surfaces to the infrastructure footprint. Some IT organizations roll their own cross-cloud platform integrations, but this job gets harder every day. It is extremely difficult to build a cross-cloud platform in a scalable, maintainable way, but more importantly, it detracts from precious developer time for product innovation.

Anthos provides a solution rooted in years of time-tested experience and technical innovation at Google in SW development and SRE operations, augmented with Google Cloud's experience managing infrastructure for modern applications across millions of enterprise customers. Anthos is unique in serving the needs of LoB developers and central IT together and with advanced capabilities in both domains. Consistency of developer and operator experience across environments enables enterprises to obtain maximum ROI from application modernization with Anthos.

1.1.1 **Accelerating Software Development**

Software product innovation and new customer experiences are the engine of new revenue generation in the digital economy. But the innovation process is such that only a few ideas lead to successful new products, most fail and disappear. As every industry transitions to being software driven, new product innovation becomes dependent on having a highly agile and productive software development process. Developers are the new kingmakers. Without an agile, efficient development process and platform, companies can fail to innovate or innovate at very high costs and even negative ROI. An extensive DevOps Research Assessment¹ study (DORA) surveyed more than 30,000 IT professionals over several years across a variety of IT functions. It has shown that excellence in software development is a hallmark of business success. This is not surprising given the importance of modern applications in fueling the economy.

DORA quantifies these benefits, showing that “elite” or the highest performing software teams are 2X more effective in attaining revenue and business goals than Low performing teams. The distinguishing characteristic of elite teams is the practice of releasing software frequently. DORA finds that four key metrics provide an accurate measurement of software development excellence. These are:

- Deployment frequency
- Lead time for changes
- Change fail rate
- Time to restore service

High-performance teams release software frequently, for example, several times a day. In comparison, low performers release less than once a month. The study also found that teams that release frequently have a lower software defect ratio and recover from errors more rapidly than others. As a result, in addition to being more innovative and modern, their software is more reliable and secure. Year-over-year DORA results also show that an increasing number of enterprises are investing in the tools and practices that enable elite performance.

Why do teams with higher development velocity have better business results? In general, higher velocity means that developers are able to experiment more, test more, and so they come up with a better answer in the same amount of time. But there is another reason. Teams with higher velocity have usually made writing and deploying code an automated, low effort process. This has the side effect of enabling more people to become developers, especially those who are more entrenched in the business vs. the tooling. As a result, high velocity developer teams have more line-of-business thinking and a greater understanding of end-user needs on the development team. The combination of rapid experimentation and focus on users yields better business results. Anthos is the common substrate layer that runs across clouds to provide a common developer experience for accelerating application delivery.

¹ <https://www.devops-research.com/research.html>

1.1.2 Standardizing operations at scale

Developers may be the new kingmakers, but operations is the team that runs the kingdom day in and day out. Operations includes teams that provision, upgrade, manage, troubleshoot, and scale all aspects of services, infrastructure, and cloud. Typically networking, compute, storage, security, identity, asset management, billing, and reliability engineering is part of the operations team of an enterprise. Traditional IT teams have anywhere from 15-30% of their staff in IT operations. This team is not always visibly engaged in new product introductions with the line of business, but often lays the groundwork, selecting clouds, publishing service catalogs, and qualifying services for use by the business. Failing to invest in operations automation often means that operations become the bottleneck and a source of fixed cost.

On the flip side, modernizing operations has a tremendous positive effect on velocity. Modern application development teams are typically supported by a very lean operations team, where 80%+ of staff is employed in software development vs. operations. Such a developer-centric ratio is only achieved through modern infrastructure with scaled, automated operations. This means operations are extremely streamlined and leverage extensive automation to bring new services online quickly. Perhaps the greatest value of Anthos is in automating operations at scale consistently across environments. The scale and consistency of Anthos is enabled by a unique open cloud approach that has its origins in Google's own infrastructure underpinning.

1.2 Origins in Google

Google's software development process has been optimized and fine-tuned over many years to maximize developer productivity and innovation. This attracts the best software developers in the world and leads to a virtuous cycle of innovation in software and software development and delivery practices. The Anthos development stack has evolved from these foundations and is built on core open-source technology that Google introduced to the industry.

At the heart of Anthos is Kubernetes, the extensive orchestration and automation model for *managing infrastructure through the container abstraction* layer. The layer above Kubernetes is grounded in Google's Site Reliability Engineering or operations practices, which standardize the control, security, and management of services at scale. This layer of *service management* is rooted in Google's Istio-based Cloud Service Mesh. Enterprise *policy and configuration* automation is built in at this layer using Anthos Configuration Management to provide automation and security at scale. This platform can run on multiple clouds and abstracts the disparate networking, storage, and compute layers underneath (see figure 1.1).

Above this Anthos stack is Developer experience and DevOps tooling, including a deployment environment that uses Knative and integrated CICD with Tekton.

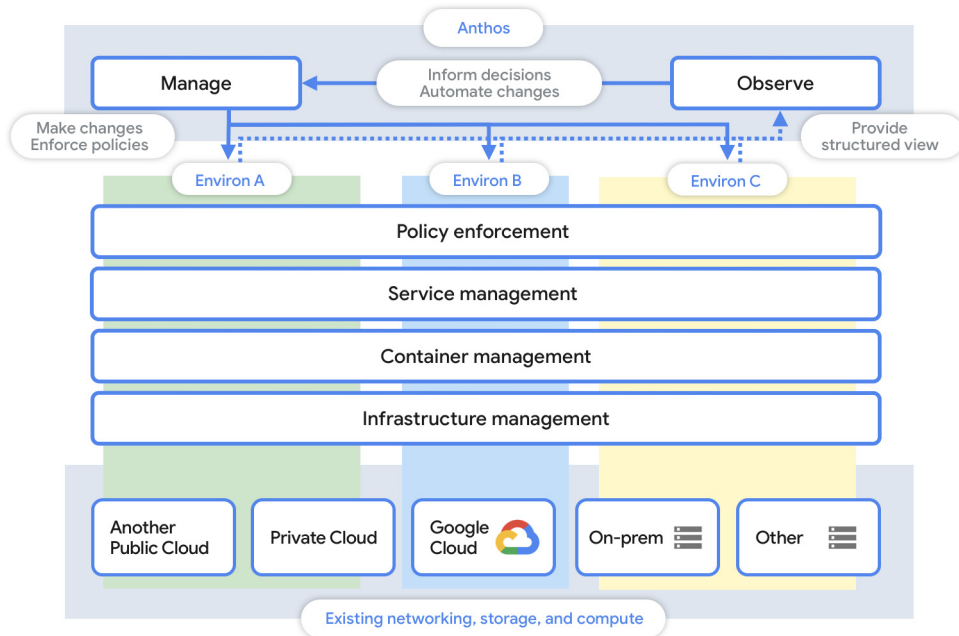


Figure 1.1 Anthos architecture overview

1.3 How to read this book

This book will take you through the anatomy of Anthos, the platform, the development environment, the elements of automation and scaling and the connection to patterns adapted from Google to attain excellence in modern software development in any industry. Equally importantly, each chapter has hands-on exercises to implement the techniques and practical examples on how to use the platform.

We begin with an overview of microservices and the shift to writing and running software in the cloud. We then introduce Anthos from a user's perspective and its major elements: Kubernetes, the service mesh, security, and operational management and bring it all together. We then introduce Anthos as it runs on prem and in multiple clouds and at the edge, showing the real power of a consistent platform across clouds, always with an eye to the applications that run on this platform.

The next section focuses on the operator experience with DevOps, networking, and policy and walks through the benefits and transformation possible for each role and person in an enterprise as it modernizes. The final section focuses on various types of workloads run on the platform with an emphasis on data, ML, security, and an ecosystem of marketplace applications that run anywhere on Anthos. We will also talk in detail about the innovations the platform provides in migrating and modernizing existing applications. We conclude with a look at the roadmap and the upcoming capabilities from our customers and field.

Summary

- Modern software applications provide a host of business benefits and are driving transformation in many industries.
- The backend for these applications is typically based on the cloud-native, micro-services architectural pattern which allows for great scalability, modularity, and a host of operational and DevOps benefits that are well suited to run on distributed infrastructure.
- Anthos, which originated in Google Cloud, is a platform for hosting cloud-native applications providing both a develop and an operational benefit.

Cloud is a new computing stack

by Phil Taylor

This chapter covers

- Defining digital velocity and the enterprise dilemma
- Understanding traditional models for application development and delivery
- Disrupting application delivery and the birth of cloud
- Learning about microservices and containers
- Discussing software-defined everything and DevOps
- Explaining why cloud is the modern computing stack

2.1 Introduction

Many technology executives and engineers talk about cloud as a new destination. In reality, it's a new state of mind. Cloud is best defined as a set of modern patterns and practices that abstract away the underlying infrastructure. On top of a new technology stack lies the evolution of new architectures for distributed systems that allow scale without requiring the developer to manage the underlying infrastructure and its design. This greatly lowers the barrier to building highly scalable, secure, and distributed applications.

Most small companies or startups are already running on the cloud and realize these advantages. However, enterprise computing platforms and applications are often more complex in comparison. This means enterprises need to support their data centers and branch facilities for many years to come. Their move to the cloud must be more purposeful and deliberate.

In the past, the upfront labor and cost of developing a new application represented the majority of a company's investment. There are three primary factors that influenced these costs:

- 1 The capital required to acquire the hardware and build the software for your new product.
- 2 The elapsed time to acquire the infrastructure and properly deploy it into a data center.
- 3 Quickly scaling infrastructure for anticipated and unanticipated usage spikes.

Early cloud computing capabilities solved these problems by allowing consumers to rent hardware in an Infrastructure as a Service model (IaaS). Under this model, a new product team could deploy the needed infrastructure in minutes or hours and start developing their applications almost immediately. The per-minute charging models for cloud infrastructure helped keep the upfront costs under control as well.

During the lifecycle of any successful product, there are multiple scaling events and higher reliability needs that require you to expand your infrastructure to meet new demands. Public cloud providers make it easy for you to scale up or scale down your networking, compute, and storage infrastructure to accommodate these events.

While it's more difficult than ever for enterprise companies to keep up with the pace of change, it's also easier than ever before to launch a startup and compete in their space. This creates a tremendous need for innovation within enterprise application design, compute models, and delivery lifecycle.

In 2011, Marc Andreessen published an essay in *The Wall Street Journal* titled "Software is eating the world" (republished by his VC firm at <https://a16z.com/2011/08/20/why-software-is-eating-the-world/>). Most of his peer investors and business executives around the world probably thought Marc was nuts. After all, it had been 10 years since the last .com bubble had burst, and they all thought it was coming again. The pace of change wasn't what it was today, and technology visionary Marc was right again. Having invented the modern web browser and the company Netscape, he had seen this type of change before.

The changes Marc was talking about are only now being applied by large enterprise companies at scale due to new capabilities like Google Cloud Platform (GCP) and Anthos. The pace of change is faster than ever. This term has become known as digital velocity.

2.2 *Digital velocity and the enterprise dilemma*

At the turn of the 20th century, business growth was slow, taking on average 90 years to reach 1 billion dollars (US) in annual revenues (see <https://www.inc.com/laura-montini/infographic/how-long-it-takes-to-get-to-a-1-billion-valuation.html>). This allowed companies who were first or early to their markets to effectively monopolize their industry. Innovation in manufacturing reduced this average to about 25 years in the years following World War II. Fast forward to the late 1990s, and things really started heating up as companies leveraged technology to accelerate their business models. Most of the acceleration was in companies with no physical presence (true web companies), and most full-stack companies failed in this era with the exception of a few shining stars like Google, Amazon, and Netflix. Today, innovations in software development and infrastructure operations have led to an extraordinary amount of acceleration. These advances in digital velocity have disrupted entire markets in a relatively short amount of time. Think of what Netflix did for streaming, Amazon for retail, AirBnB for short term rentals, and Uber for taxis.

In 2021 and beyond, there is not a market that is safe from disruption due to advances in digital velocity because of new software development patterns and more efficient infrastructure resources. Every company is finding they are rapidly becoming software companies. In the next section, you'll begin to analyze these software development patterns.

2.3 *Traditional models for application development and delivery*

Historically, enterprise applications were most often written using monolithic software development patterns. To help deal with monoliths at scale, many developers moved from functional programming in languages like COBOL to Object Oriented Programming (OOP) in modern languages like C++ or Java and leveraged modular design patterns. This allowed developers to break up code bases into libraries that could be shared and reused across entire application portfolios. For example, other attempts at efficiency involved using stored procedures in relational databases to process data more rapidly. These innovations helped reduce toil, but they only rarely allowed enterprises to scale innovative ideas across more than a few teams. Instead, enterprises more often became quagmire in dealing with the legacy problems introduced by these new development paradigms.

When the IT industry began, code would be written and remain relatively static for months or even years. Most businesses did not invest in rewriting or re-platforming code. As these new OOP-based applications grew organically, supporting them became increasingly complex. Today, these messy areas of the solution or “hacks” as they are popularly called, are formally referred to as technical debt. Technical debt is recognized as a first-class factor that can be used to gauge the digital velocity of an organization or specific project. Too much technical debt results in a reduced velocity over time, whereas not enough means you're not moving fast enough in most cases to keep up with business needs.

Enterprise applications were built using three popular design patterns (1) [Client-Server](#), (2) [n-tier web architecture](#), and (3) [Service Oriented Architecture](#).

2.3.1 Advantages and pitfalls of client/server architecture

The client-server architecture (see figure 2.1) was simple and easy to build. Several popular programming languages and runtime environments helped build client-server applications faster. These Rapid Application Development (RAD) platforms like Visual Basic or Delphi were easy to use, and many developers were able to get results rapidly.

The pitfalls of this application architecture included duplicated logic (business logic was duplicated in the client and server code bases), having to install the application on end-user machines, updating the application on those machines, and keeping the server and its clients in sync. All the aforementioned problems led to usability issues for the users, higher sustainability costs, and longer lead times to get new features to market.

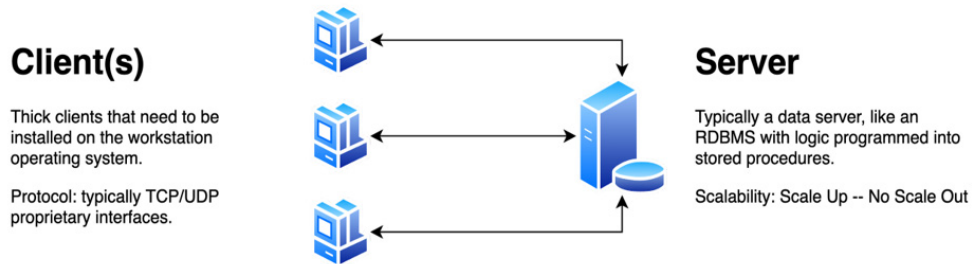


Figure 2.1 Client server architecture.

2.3.2 Advantages and pitfalls of web architecture

A more complex architecture started to evolve with the invention of the modern web browser. This architecture is similar to the old [dumb terminal](#) days when an application was centrally deployed and managed. This new n-tier architecture (see figure 2.2) was based on the idea that we should be generally separating our applications into three tiers: presentation, application or business logic, and data.

Although this architecture and deployment model fixed some pitfalls from client-server (installing and updating the application on clients), it still suffered from similar issues at scale. Most of these applications still suffer from duplicated logic (data validation, business logic) that lead to usability issues, higher sustainability costs, and longer lead times to get new features to market.

2.3.3 Service-oriented architecture

Both client-server and n-tier architecture had another common pitfall: application crashes on the server affected every module or service and, possibly, a large majority of users. [Service-oriented architecture \(SOA\)](#), described in figure 2.3, helps resolve

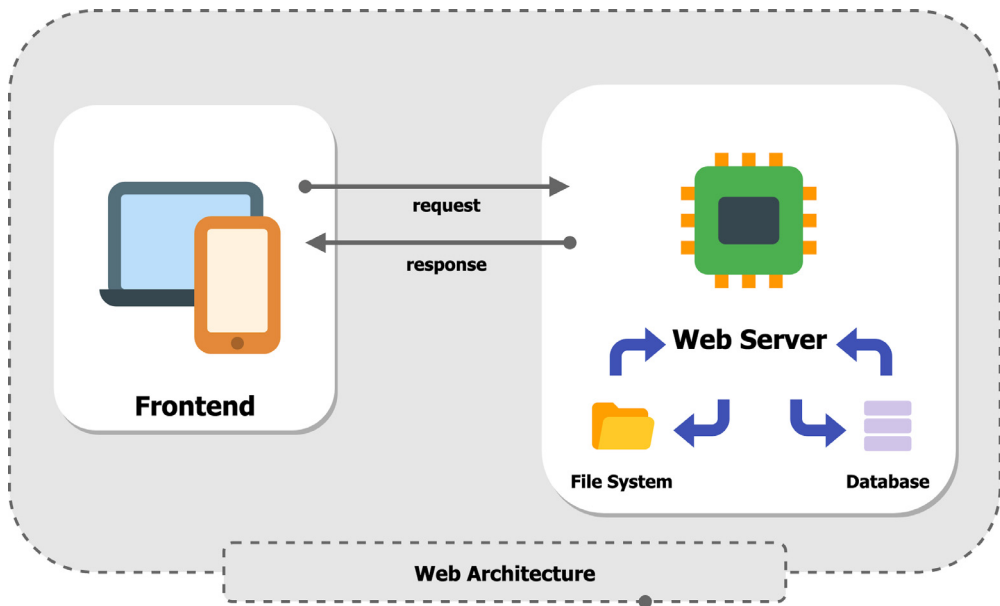


Figure 2.2 N-Tier (web) application architecture.

some challenges with monolithic architectures and runtime support. The SOA model splits application business logic into well-known [domains](#) and standardizes on protocols for communication between other tiers and security. This pattern provided lots of operational and development advantages, including better scalability and resilience. While the API revolution still remains valid, there were a number of pitfalls with adopting SOA, including cost and complexity of the underlying frameworks required by commercial products which were hard to install and maintain leading to a higher Total Cost of Ownership (TCO). However, its biggest pitfall was the learning curve to adopt the complex pattern and protocols. These limitations and high costs have resulted in a decline in popularity in recent years and a rise in the [microservices](#) pattern. We will cover microservices in more detail later in the book, chapter 21, Application Modernization.

During this evolution, data centers were also undergoing change and disruption.

2.4 *Disrupting application delivery and the birth of cloud*

As application development methods evolved, teams still struggled to execute successful builds of their source code and deploy them into the existing infrastructure. Processes were manual and error prone. Developers, operations teams, and managers all on a late-night conference call to deploy and debug an application were not uncommon. To help address this, [Continuous Integration \(CI\)](#) and [Continuous Delivery \(CD\)](#) was developed. Continuous Integration is the process of merging all developer

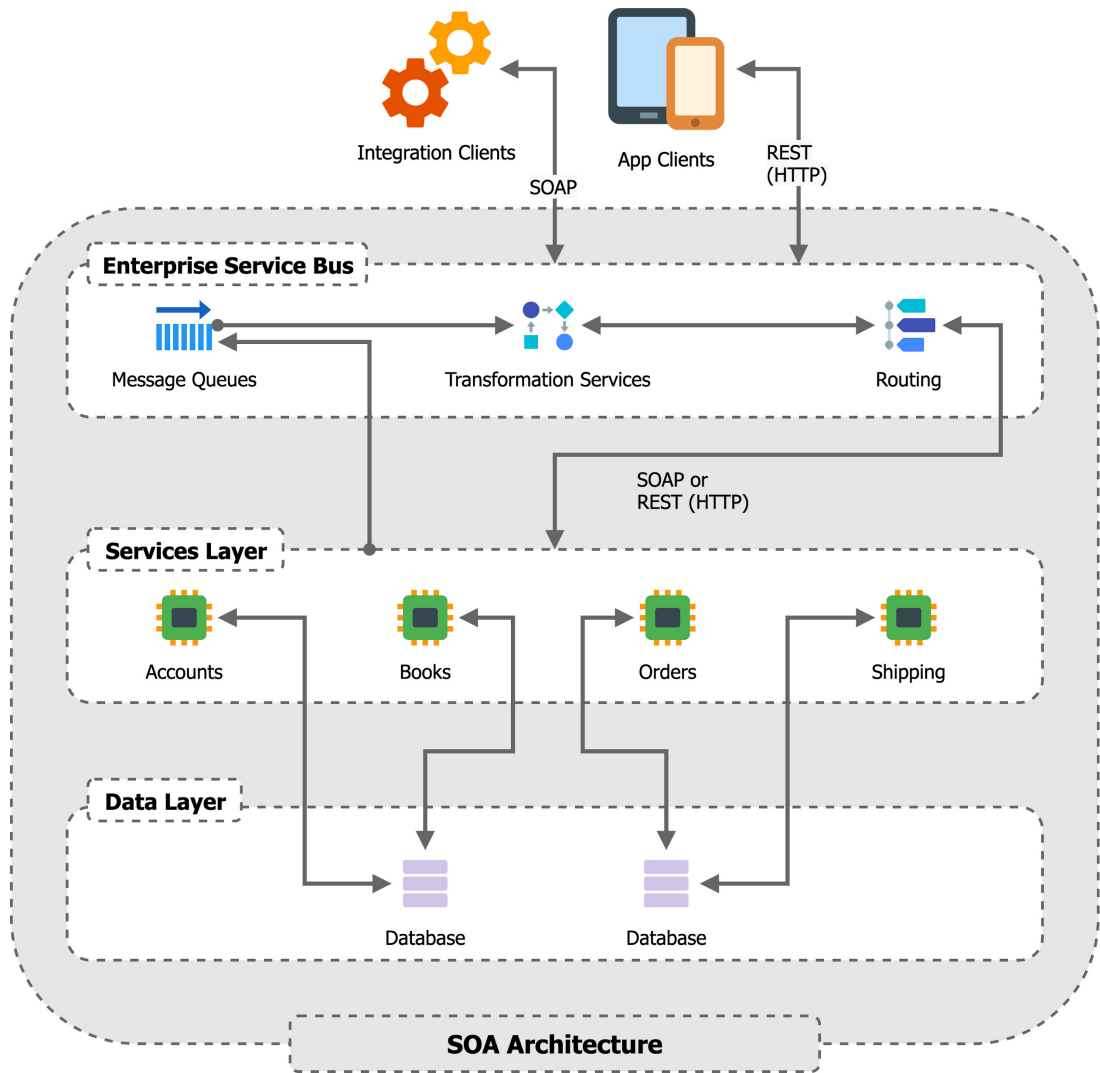


Figure 2.3 SOA architecture.

changes once or multiple times throughout the day to validate build integrity. Continuous Delivery is the process of validating and packaging deployment artifacts for release at any point in time. The goal of CI/CD is to create a reliable build and packaging process for repeatable success. Even with a powerful framework like CI/CD, realizing its efficiencies was difficult because the frameworks used to develop software were designed for applications that were designed for months or years and ran largely unchanged for long periods of time. One of the most popular of these project management processes is called Waterfall.

2.4.1 **Disrupting how software is made**

Waterfall is a project management process that is designed to manage a project based on milestones, timelines, and resources. With Waterfall projects, the team does a lot of up-front work to think through all the requirements and design the solution. This often leads to problems later in the product cycle because the business requirements have changed, and the design doesn't reflect the new business requirements. This inevitably leads to missed deadlines and quality issues with the finished product.

Beginning in 2001 with the release of *The Agile Manifesto* (<https://agilemanifesto.org/>), development teams began to evaluate their project management processes. A leading process that provides more efficient software development, Agile teams begin a project by thinking through major requirements and rough out a high-level design rather than a full detailed design.

The team then works in iterations to deliver working software with more realistic designs at each iteration. Scrum is a well-known framework for developing and delivering complex products using an agile mindset. Lots of teams prefer the [Scrum](#) methodology to run their Agile projects and two-week iterations known as sprints. Once the teams have moved to Agile, it is imperative that they implement CI/CD as they are now expected to build, test, and ship the system at the end of each iteration.

2.4.2 **Development innovation at Google**

At Google we leverage a number of patterns to create acceleration like [monorepo](#) and [trunk-based development](#) that are described both in the popular book *Software Engineering At Google*. Many of the patterns in use at Google to create acceleration have been adopted across the industry.

2.4.3 **Application development throughout the industry**

In 2017, according to an [article published by InfoQ](#), the Facebook team was performing 50,000 builds a day just for its Android app. The statistics we just mentioned only refer to their CI process, which is responsible for building the source code and running local testing to validate the build. It doesn't include the CD workflows, which would have massive infrastructure requirements to deploy 50,000 versions of the application on physical or virtual devices and run additional testing.

These infrastructure challenges aren't unique to Facebook. Years earlier, Google and Amazon ran into similar challenges building and operating their core infrastructure and applications. These challenges directly led to the cloud as we know it today. The invention of cloud platforms were largely driven by internal IT groups' inability to respond to business needs on an acceptable timeline. Shadow IT, as it is commonly known, is the practice of leveraging a third-party IT provider to deliver the business needs outside of internal IT groups. A lot of early cloud customers have started in a shadow IT mode before fully adopting new cloud platforms as a standard for infrastructure and application delivery. In 2006, Amazon launched Amazon Elastic Compute Cloud (EC2). Heroku (2007), GCP (2008), Azure (2010), and Cloud Foundry

(2011) rapidly followed EC2. Both Heroku and Cloud Foundry were cloud-based Platform as a Service (PaaS) solutions that focused on the idea of building [12 Factor Apps](#). Although all of these cloud-based platforms were great at creating incremental improvements, enterprise customers struggled to create complete transformation. The public cloud needed more maturity and the early PaaS platforms like [Google App Engine](#) were narrow in scope, having limited support for languages and advanced language capabilities.

Although the aforementioned platforms all had their constraints, this was the start of an era that would disrupt the data center and a number of business markets. It was the first time in history that you didn't need to build the software, acquire the hardware, and build the data center to launch your new startup. You simply had to rent the infrastructure from the cloud or PaaS provider, build, and deploy your app!

This new cohort of startups is often referred to as “Digital Natives”—companies born in the cloud. Their innovations continue to drive how applications are created.

2.4.4 Contract-first development, SOA, and the evolution to microservices

Around 2005, architects and developers started talking about the idea of [Test-Driven Development \(TDD\)](#). It was said that by following the principles of TDD, your code would be more testable and ultimately more reliable. The concept of TDD is to think through a coding problem and write the testing plan and actual tests prior to writing the code itself. Many developers found success in this practice which has evolved into the concept of [Design by contract](#). If you're writing a web service exposing an API, then you should define the contract and data model first. This forces developers to think through the input/output boundaries and build a well-encapsulated service. It also gives the consumer teams of your API a contract to start using in their own development process. Design by contract leverages several of the concepts from SOA. With SOA, developers build all the subsystems using a web service pattern with messaging protocols between services. As newer ideas were applied to this concept, new technologies began to emerge.

2.5 Microservices and containers

While enterprises were adopting SOA to build complex backend subsystems, tech startups continued to innovate. Microservices in many ways are an evolution of SOA. They share the same core idea to build all subsystems as web services that communicate with each other as illustrated in figure 2.4.

Microservices differ in that they favor smart endpoints with dumb protocols, as opposed to Enterprise Service Bus (ESB) and Simple Object Access Protocol (SOAP) protocols that are popular with service-oriented architecture (SOA). The function of the ESB is to centralize service communication, message transformation, and routing. Whereas microservices leverage Publish/Subscribe (Pub/Sub) message buses and favor decentralized communication between services.

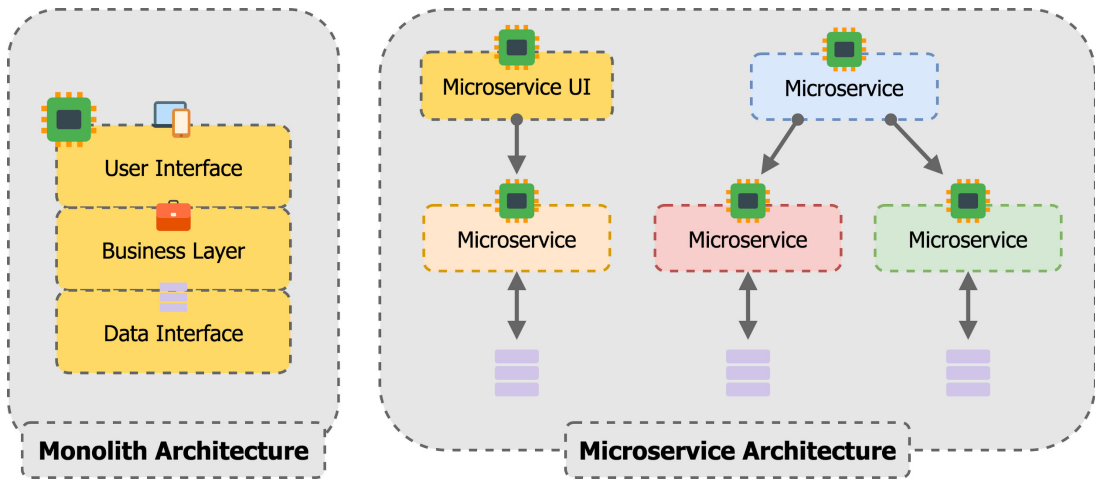


Figure 2.4 Microservice architecture.

Instead, teams build [REST](#) web services using HTTP + JSON for transport and payload protocols. Here are some other key differences:

- Microservices organize the design around business capabilities (Domain Driven Design). This allows us to write software with clear boundaries focused on solving a single business problem or a set of related problems within one microservice.
- The team chooses the programming language, database, or other tech stack architecture to meet the requirements. This allows teams to choose technologies that lend well to solving the domain problem the microservice is responsible for providing.
- “Shared nothing” approach, each service has an isolated datastore. This approach allows teams to integrate at the API layer, encapsulating all functionality within their service and having clear boundaries to any dependent services.
- All communication happens through a single contract, typically exposed by a RESTful API. All communication between services (internal consumers or external consumers) access the same API interface.
- Smart endpoints, dumb pipes—simple protocols are leveraged, and the services know how to handle the inputs and where to send the outputs.
- Message queues are leveraged for asynchronous processing and throttling at scale. It’s the service’s responsibility to implement a message stack to support this capability. Not all services will need it.
- Automated testing is much easier with these cloud native design patterns. We are able to create ephemeral environments dedicated to testing a specific component or feature, before releasing our software into an integrated environment for end-to-end testing. In the past, most teams struggled to write automated tests that would accurately validate their software and not require a

lot of change between deployments. The **test pyramid** is a metaphor that tells us to group tests into buckets of different granularity (UI tests, service tests, unit tests). Most monolithic application development teams focused on building UI tests to validate the application. Since the UI group is known to require more integration, this meant that these tests generally had to be updated regularly. This kept them from achieving advanced patterns like **canary** or **blue/green** deployments. A canary is where we would slowly roll out the tested software to a small subset of users to collect live feedback before rolling out to all users. Whereas a blue/green allows us to maintain two copies, a green or production environment and a blue or staging environment allowing us to roll back to a previous state quickly if anything goes wrong. Since most of these apps are built from scratch, the teams are able to properly apply the test pyramid, as well as advanced end-to-end deployment patterns like a canary or blue/green deployment. This leads to complete autonomous deployments and unmatched development velocity. Another big challenge for monolithic teams was data rehydration: the ability to seed the software platform with the data required for automated testing. The challenge of data rehydration is no longer an issue because these teams generally build up their required data sets as part of their automated testing code bases. This allows them to meet the unique requirements for each use case in the test plan and avoid compliance problems created by copying live data to test systems.

Microservice architectures can be fast, efficient, and powerful. But to find widespread adoption, they needed infrastructure innovation as well. This innovation happened when Linux Containers came to the market.

2.5.1 Containers enable microservices

Containers have been around in some fashion since the days of Solaris UNIX. The problem with containers historically is that they were hard to build, and it took a series of Command Line Interface (CLI) calls to build one. In 2010, Docker revolutionized this process when they released their own open-source container engine.

This new engine allowed us to document the container requirements in a source file and greatly reduced the number of CLI calls we have to make to successfully build a container. The new Dockerfiles, another Docker innovation, gave developers a way to document the requirements of a container using an Infrastructure as Code (IaC) approach instead of documenting CLI calls. By packing apps or microservices in containers, developers have a quick way to build, test, and deploy their applications to a target environment. This allows developers to build software, build the target environment and then deploy and test the entire application stack as a single unit of work.

Individual containers are a first step, but full applications need to orchestrate multiple containers and instantiate them on an underlying infrastructure, making this more complicated.

Kubernetes solves most of the infrastructure and deployment problems (configuration, service discovery, secrets, scheduling, etc.) and provides a common orchestration interface for application infrastructure. As teams scale their solutions, they will generally require capabilities that go beyond the basics that Kubernetes provides. Teams that are running large solutions will generally need more advanced capabilities and leverage other ecosystem services like Istio for service mesh or Apigee for API gateway traffic.

2.6 **Software defined everything and DevOps**

As you may be starting to realize, the key to velocity is through software development and independent deployments that allows teams to work in parallel with limited coordination. It's not enough to just build your application with software. The cloud has shown us that true velocity happens when the whole solution can be defined as code. This concept has been defined by the industry as DevOps which encompasses a lot more than just the technology pieces. We are not going to unpack all the DevOps concepts in this book and will focus only on the technical aspects. If you would like to learn more about DevOps, we recommend you read *The DevOps Handbook*.

To fully automate the build, testing and deployment of an application, the team will need to become experts on CI/CD concepts, along with mastering **Infrastructure as Code (IaC)**. The example in figure 2.5 shows a visualization of a CI/CD pipeline for supporting a Kubernetes application deployment.

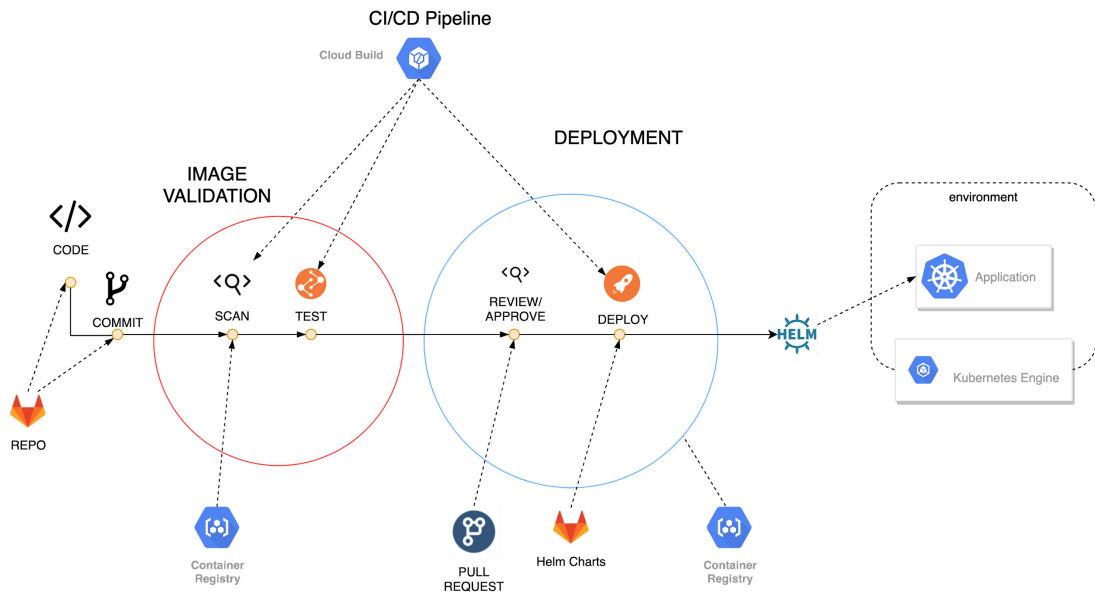


Figure 2.5 CI/CD pipeline for the Kubernetes application.

Teams that tried to deploy containers in the early days had mixed results as they needed to solve a lot of the core platform concerns required to serve these applications at scale. There is enough work to do in order to build, test, and deploy the application successfully. This is the reason that Google created Kubernetes and subsequently Anthos. Anthos greatly simplifies the process of creating and managing the Kubernetes clusters and the auxiliary components and processes needed to support multiple clusters and hybrid cluster architectures. Anthos will help your teams deploy the core Kubernetes distribution, uniformly apply security policies to clusters, provide capabilities to enhance your service mesh to deal with hybrid cloud or hybrid cluster routing, and more easily apply upgrades or other operational tasks. With Anthos developers can “write once, run everywhere”, including across multiple clouds, the edge, and on-prem environments.

2.7 Cloud is the modern computing stack

In this chapter, you have come to understand the speed at which modern businesses are rising and how technology and software patterns have changed to enable this acceleration. A decade ago, it would have been unthinkable to have a new business enter the taxi market and in a few short years disrupt the entire industry. Today this is a regular occurrence due to the rapid growth of new businesses fueled by digital velocity software architectures and universal access thanks to the internet.

The concepts making this possible can be boiled down to distributed highly scalable architectures like microservices and the use of DevOps patterns like SRE to deliver these solutions autonomously creating rapid iterative change. This rapid pace of innovation began with cloud native startups developing new technology. In a few short years, they’re now being applied to enterprise workloads and existing applications successfully as well.

Summary

- Business are leveraging the public cloud to meet the demands and agility their businesses require.
- Microservices are becoming more popular in software designs, allowing for parallel development to meet the software requirements without the typical drawbacks of a monolithic architecture.
- Good CI/CD capable of autonomous deployments leveraging IaC and automated testing is critical to success.
- Teams are leveraging containers and Kubernetes for the packaging and operation of these applications.
- Google Anthos is filling the void to make Kubernetes a reliable, scalable platform for the enterprise.

Anthos, the one single pane of glass

by Melika Golkaram

This chapter covers

- Understanding the advantages of a single pane of glass and its components
- Learning how different personas can use and benefit from components
- Getting hands-on experience configuring the UI and attaching a cluster to the Anthos UI

We live in a world where application performance is critical for success. To better serve their end-users, many organizations have pushed to distribute their workloads out of centralized data centers. Whether to be closer to their users, to enhance disaster recovery, or to leverage the benefits of cloud computing, this distribution has placed additional pressure on the tooling used to manage and support this strategy. The tools that have flourished under this new paradigm are those that have matured and become more sophisticated and scalable.

There is no one-size-fits-all tool. Likewise, there is no one person who can manage the infrastructure of even a small organization. All applications require tools to

manage CI/CD, monitoring, logging, orchestration, deployments, storage, authentication/authorization, and more. In addition to the scalability and sophistication mentioned here, most of the tools in this space offer an informative and user-friendly graphical user interface (GUI). Having an easily understood GUI can help people use the tool more effectively because it lowers the bar for learning the software and increases the amount of pertinent output the user receives.

Anthos itself has the capacity to support hundreds of applications and thousands of services, so a high-quality GUI and user experience is required to leverage the ecosystem to its full potential. To this end, Google Cloud Platform has developed a rich set of dashboards and integrated tools within the Google Cloud Console to help you monitor, troubleshoot, and interact with your deployed Anthos clusters, regardless of their location or infrastructure provider. This single pane of glass allows administrators, operations professionals, developers, and business owners to view the status of their clusters and application workloads, all while leveraging the capabilities of Google Cloud's Identity and Access Management (IAM) framework and any additional security provided on each cluster.

The Anthos single pane of glass is not the first product to attempt to centralize the visibility and operations of a fleet of clusters, but it is the one offering support for a large variety of environments. In order to fully understand the benefits of the Anthos GUI, we are going to look at some of the other options available to aggregate and standardize the interactions with multiple Kubernetes clusters. However, we first need to have a standard set of terminology and a brief overview of several DevOps principles.

In this chapter, we take a journey through the offerings of the Anthos GUI, its “single pane of glass”.

3.1 Terminology

A single pane of glass offers three main pillars of characteristics that are shared across all operators, industries, and operations scales. These three pillars are:

- *Centralization:* As the name suggests, a single pane of glass should provide a central UI for resources, no matter where they run and to whom they are provided. The former aspect relates to which infrastructure and cloud provider the clusters are operating on and the latter relates to inherently multi-tenant services where one operator centrally manages multiple clients' clusters and workloads. With the benefits of a central dashboard, admins will be able to get a high-level view of resources and drill down to areas of interest without switching the view.

However, a central environment might cause some concern in areas of privacy and security. Not every administrator is required to connect to all clusters and neither all admins should have access to the UI. A central environment should come with its own safeguards to avoid any operational compromise with industry standards.

- *Consistency:* Let's go back to the scenario of an operator running clusters and customers in multi-cloud or hybrid architectures. A majority of infrastructure

providers, whether they offer proprietary services or run on open source, attempt to offer a solid interface for their users. However, they use different terminology and have inconsistent views on the priorities. Finally, depending on their UI philosophy and approach, they design the view and navigation differently. Remember, for a cloud provider, cluster and container management is only part of the bigger suite of services and a member of a pre-designed dashboard. While this might be a positive element in single operating environments (you can learn to navigate outside of Kubernetes dashboard into the rest of cloud services dashboard with minimum switching), it becomes a headache in multi-environment services and those who focus mainly on Kubernetes.

- *Ease of use:* Part of the appeal of a single pane of glass in operation is how data coming from different sources are aggregated, normalized, and visualized. This brings a lot of simplicity in drilling down into performance management and triage, especially if it combines a graphical interface with it.

A graphical UI has always been an important part of any online application. First, at some point in an app management cycle, there is a team who doesn't have either the skills or the interest of working with remote calls. This team expects a robust, easy to navigate, and a highly device agnostic UI for their day-to-day responsibility.

Second, regardless of the skillsets, an aggregated dashboard has so much more to offer in one concentrated view than calling service providers and perhaps clusters individually, given that the UI provides lots of data fields with the right installation and readability.

When working in an IT division or department of an organization, roles can be split in any number of ways. We are going to consider a specific set of users, or personas, that are indicative of common roles and responsibilities within an organization. However, any given group may combine some of these roles together or split them out further.

3.1.1 *Personas*

We start where most companies start: with the *infrastructure administrator*. This is the individual who is primarily responsible for the day-to-day maintenance of existing hardware, and the deployment of new hardware and core services. In the context of Anthos, this person will usually be responsible for the network backbone, as well as the physical servers the on-premise clusters run on. In general, this persona has the least interaction with the day-to-day operations and monitoring of the clusters but will be involved in discussions around new compute or network requirements and will be in the notification chain for outages.

A large portion of the tooling provided as part of Anthos is intended to make the day-to-day operations of clusters and workloads as streamlined as possible—this task typically falls to the Operations Team, headed by the *operations administrator*. For Anthos operations, this persona will typically be handling core monitoring and logging, as well as ensuring the health of the internal platform built on top of an organization's infrastructure and cloud environments, rather than dealing with the physical

hardware itself. While Google advocates for the wider adoption of DevOps or Site Reliability Engineering (SRE) practices², even in a company that has fully embraced the DevOps/SRE ways, ownership and monitoring of applications are never fully decentralized. This persona will be involved in managing the health and utilization of the platform as a whole. They may drill into a specific application from time to time, but more often they act as a resource for the developers (see the following), rather than handling any issues directly.

As distributed operations and deployments expand, some traditional security practices such as strict release cycles, scheduled, manual regression tests, or tight lock-down of production environments have become unwieldy, slow, and frustrating. Because of this, the *security administrator* is moving to perform more monitoring and enforcement-by-default for the deployed applications and clusters. While a security team may be responsible for both physical and digital security, this book is primarily focused on the digital side. Anthos provides robust tooling, such as Anthos Configuration Management³, and the Anthos dashboards to enable the security administrator to enforce, monitor, and audit clusters and workloads that have been deployed.

Finally, the *developers* of an organization have important roles to play in deploying and managing the application their teams are directly responsible for. As we said before, we strongly encourage that developers are given the tools and responsibility to operate and maintain their own applications, while removing the need for them to manage their infrastructure directly. For the purposes of this chapter, we consider the Developer persona to encompass all technical personnel associated with a particular application, including application developers, test and quality assurance, and DevOps/SRE experts.

3.1.2 DevOps/Site Reliability Engineering concepts

DevOps is a method of approaching development to encourage faster delivery while improving reliability of the finished product. Site Reliability Engineering (SRE) is Google's way of implementing DevOps in the particular Google style, moving as much as possible to automation and reducing the manual work that needs to be done. An exhaustive review of the principles and practices for DevOps/SRE is far too broad for this book, let alone this chapter. However, several concepts from those principles have an outsized presence in the Anthos GUI. Specifically, we will be exploring the concepts related to automatically monitoring a workload, including how to analyze the metrics that the Anthos system captures:

- A *Service Level Indicator (SLI)* is a specific measure of how a workload is performing. Many types of SLIs can be created, either generic or specific to the workload, but the four key focus areas are LETS or Latency, Errors, Traffic, and Saturation. For Anthos, this can track the resource requests for a pod, or the time it takes to process a specific inbound HTTP request.

² Books can be found at <https://sre.google/books/>.

³ See chapter 14, Anthos configuration management.

- With the SLIs, we can generate *Service Level Objectives (SLO)*. These are achievable targets we should expect to satisfy regularly, expressed as a measurement of the SLI. For example, we may specify that the 95th percentile latency for a service must be under 200ms, or that the general error rate for an application must be less than one per minute.
- Usually, a more generous target than the SLO, a *Service Level Agreement (SLA)* is a minimum target that must be met, or the consequence specified in the SLA will be invoked. For example, this may be a requirement that the platform is available 99.99% of the time, or that the 95th percentile latency of a service must never exceed 2 seconds.

For a more in-depth examination of DevOps, please see the Google Cloud DevOps page at <https://cloud.google.com/devops>. SRE is used across all of Google, and more information can be found at <https://sre.google/>.

3.1.3 Other terminology

In addition to these concepts, we have a few miscellaneous terms that we will use throughout this chapter. For Anthos, we have two types of clusters that are visible in the GUI: *Attached* and *Anthos-managed*. Anthos-managed clusters are clusters that are deployed via Anthos utilities and are a flavor of Google Kubernetes Engine (GKE) in GCP, on-premise, or in another cloud. Attached clusters are those that are compatible with the Connect software but are not actual GKE clusters. These can include the cloud-native Kubernetes flavors or other third-party versions of Kubernetes. There are some features that are not officially supported on Attached clusters; these features will be called out when they are covered in the chapter.

3.2 Non-Anthos visibility and interaction

Anthos is not the first solution to expose information about a Kubernetes cluster through a more easily digested form than the built-in APIs. While many developers and operators have used the command-line interface (CLI), `kubectl`, to interact with a cluster, the information presented can be very technical and does not usually help surface potential issues in a friendly way. Extensions to Kubernetes, such as Istio or Anthos Configuration Management, typically come with their own CLIs as well (`istioctl` and `nomos`, for example). Cross-referencing information between all the disparate tools can be a substantial exercise, even for the most experienced developer or operator.

3.3 Kubernetes Dashboard

One of the first tools developed to solve this particular issue was the Kubernetes Dashboard⁴. While this utility is not deployed by default on new Kubernetes clusters, it is trivial to deploy to the cluster and begin utilizing the information it provides. In addition to providing a holistic view of most of the components of a Kubernetes cluster,

⁴ See <https://github.com/kubernetes/dashboard>.

the Dashboard also provides users with a GUI interface to deploy new workloads into the cluster. This makes the dashboard a convenient and quick way to view the status and interact with a new cluster.

However, it only works on one cluster. While you can certainly deploy the Kubernetes Dashboard to each of your clusters, they will remain independent of each other and have no cross-connection. In addition, because the dashboard is located on the cluster itself, accessing it remotely requires a similar level of effort to using the CLI tool, requiring services, load balancing, and ingress rules to properly route and validate incoming traffic. While the dashboard can be powerful for proof-of-concept or small developer clusters, multi-user clusters need a more powerful tool.

3.3.1 Provider-specific UIs

Kubernetes was released from the beginning as an open-source project. While based on internal Google tools, the structure of Kubernetes allowed vendors and other cloud providers to easily create their own customized versions of Kubernetes, either to simplify deployment or management on their particular platform(s), or to add additional features. Many of these adaptations have customized UIs for either deployment or management operations.

For cloud providers in particular, much of the existing user interfaces for their other products already existed and followed a particular style. Each provider developed a different UI for their particular version of Kubernetes. While a portion of these UIs dealt with provisioning and maintaining the infrastructure of a cluster, some of the UI was dedicated to cluster operations and manipulation. However, each UI was implemented differently, and cannot manage clusters other than the native Kubernetes flavor for that particular cloud provider.

3.3.2 Bespoke software

Some companies have decided to push the boundaries themselves and develop their own custom software and UIs to visualize and manage their Kubernetes installations and operations. While always an option due to the open standards of the Kubernetes APIs, any bespoke development brings all the associated challenges that come with maintaining any custom operations software: maintaining the software for new versions, bug fixing, handling OS and package upgrades, etc. . . . For the highest degree of customization, nothing beats bespoke software, but the cost vs. benefit calculation does not work out for most companies.

3.4 The Anthos UI

Each of the previous solutions has a fundamental flaw that prevents most companies from fully leveraging it. The Kubernetes Dashboard has no multi-cluster capability and does not handle remote access easily. The provider-specific UIs work well for their flavor but cannot handle clusters that are not on their network or running their version of Kubernetes. And bespoke software comes with a high cost of development and maintenance. This is where the Anthos multi-cluster single pane of glass comes into

play. This single pane of glass is an extension of, and embedded in, Google Cloud Platform's already-extensive Cloud Console that allows users to view, monitor, and manage their entire cloud infrastructure and workloads.

The solution Google has developed for multi-cluster visibility in Anthos depends on a new concept called Fleets (formerly referred to as Environs) (continue reading the next section), the Connect framework and the Anthos dashboard. The Anthos dashboard is an enhancement of the existing GKE dashboard that Google has provided for several years for its in-cloud GKE clusters. The Connect framework is new with Anthos and simplifies the communication process between Google Cloud and clusters located anywhere in the world. Fleets are methods of aggregating clusters together in order to simplify common work between them. Let's take a moment to discuss a bit more about fleets.

3.4.1 **Fleets**

Fleets are a Google Cloud concept for logically organizing clusters and other resources, letting you use and manage multi-cluster capabilities and apply consistent policies across your systems. Think of them as a grouping mechanism that applies several security and operation boundaries to resources within a single project⁵. They help administrators build a one-to-many relationship between a Fleet and its component clusters and resources to reduce the configuration burden of individual security and access rules. The clusters in a Fleet also exist in a higher trust relationship with each other by belonging to the same Fleet. This means that it is easier to manage traffic into and between the clusters and join their service meshes together.

An Anthos cluster will belong to one and only one Fleet and cannot join another without leaving the first. Unfortunately, this can present a small problem in complex service communications. For example, if we have an API service and a data processing service that need to run in distinct Fleets for security reasons, but both need to talk to a bespoke permissioning service. The permissioning service can be placed in one of the two Fleets, but whichever service does not share its Fleets will need to talk to the permissioning service using outside-the-cluster networking. However, this rule for Fleets prevents users from accidentally merging clusters that must remain separate, as allowing the common service to exist in both Fleets simultaneously would open up additional attack vectors.

When multiple clusters are in the same Fleets, many types of resources must have unique names, or they will be treated as the same resource. This obviously includes the clusters themselves, but also namespaces, services, and identities. Anthos refers to this as "sameness". Sameness forces consistent ownership across all clusters within an Fleets, and namespaces that are defined on one cluster, but not on another, will be reserved implicitly.

⁵ A Google Cloud Platform project is a set of configuration settings that define how your app interacts with Google services and what resources it uses.

An important consideration when designing the architecture of your services is that the namespace and services sameness also includes a unique internal identity across all environments in one Fleet. This means by implicitly configuring a mesh policy that allows simple access between *service A* and *service B* will apply this policy across all pods in all environments and clusters that these services can be found.

Finally, Anthos allows all services to utilize a common identity when accessing external resources such as Google Cloud services, object stores, and so on. This common identity makes it possible to give the services within a Fleet access to an external resource once, rather than cluster-by-cluster. While this can be overridden and multiple identities defined, if resources are not architected carefully and configured properly, negative outcomes can occur.

3.5 **Connect, how does it work?**

Now that we have discussed Fleets, we need to examine how the individual clusters communicate with Google Cloud. Any cluster that is part of Anthos, whether Attached⁶ or Anthos-managed, has Connect deployed to the cluster as part of the installation or registration process. This deployment establishes a persistent connection from the cluster outbound to Google Cloud that accepts traffic from the cloud to permit cloud-side operations secure access to the cluster. Because the initial connection is outbound, it does not rely on a fully routable connection from the cloud to the cluster. This greatly reduces the security considerations and does not require the cluster to be discoverable on the public internet.

Once the persistent connection is established, Anthos can proxy requests made by Google Cloud services or users using the Google Cloud UI to the cluster whether it is located within Google Cloud, in another cloud provider, at the edge, or on-premise. These requests use the user's or the service's credentials, maintaining the security on the cluster and allowing the existing role-based access controls (RBAC)⁷ rules to span both direct connectivity as well as connections through the proxy. A request using the Anthos UI may look like figure 3.1.

While the tunnel from the Connect agent to Google Cloud is persistent, each stage of each request is authenticated using various mechanisms to validate the identity of the requestor and that that particular layer is allowed to make the request. Skipping layers is not permitted and will be rejected by the next layer receiving the invalid request. An overview of the request-response authentication is seen in figure 3.2.

Even if a user has the appropriate RBAC permissions to gain access to a given cluster, they must still be allowed to view the Google Cloud project the cluster is attached to in order to use the Connect functionality. This uses the standard IAM processes for

⁶ Attaching clusters lets you view your existing Kubernetes clusters in the Google Cloud Console along with your Anthos clusters and enable a subset of Anthos features on them, including configuration with Anthos Config Management. More details can be found at <https://cloud.google.com/anthos/docs/setup/attached-clusters#prerequisites>.

⁷ Role-based access control (RBAC) is a set of permissions and an authorization component that allows and denies admin or compute objects access to a set of requesting resources.

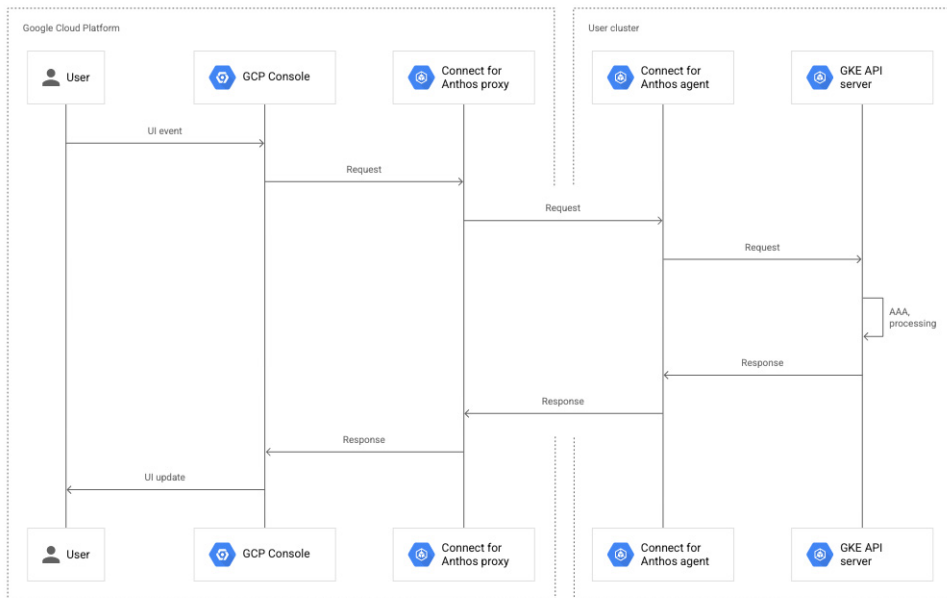


Figure 3.1 Flow of request and response from Google Cloud to cluster and back.

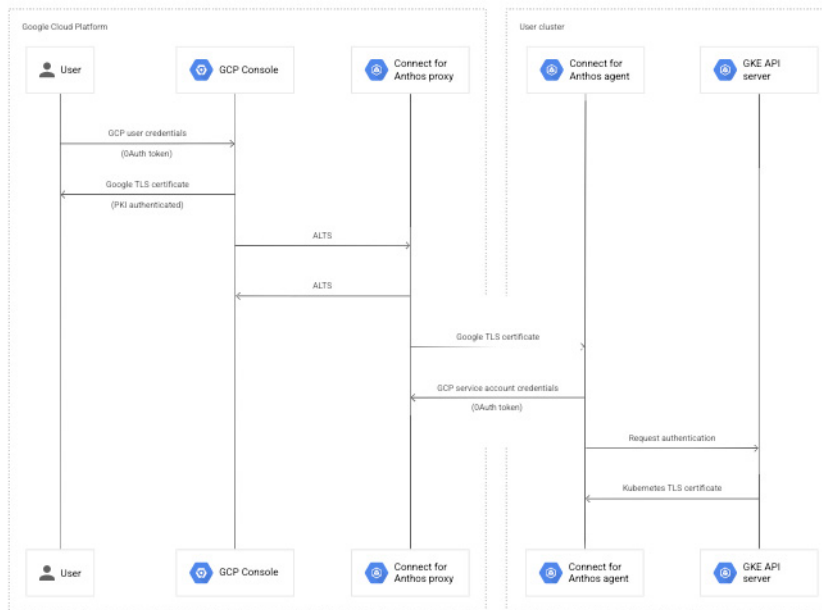


Figure 3.2 Request validation steps from Google Cloud to cluster.

a given project, but having the separate permission allows the Security team to grant a user access to a cluster through a direct connection (or some other tunnel), but not allow them remote access via Google Cloud.

Connect is compliant with Google's Access Transparency⁸, which provides transparency to the customer in two main areas:

- *Access approval*: Customers can authorize Google support staff to work on certain parts of their services. Customers can view the reasons why a Google employee might need that access.
- *Activity visibility*: Customers can import access logs into their project Cloud logging to have visibility into Google employees' actions and location and can query the logs in real-time, if necessary.

3.5.1 Installation and registration

In order to leverage the Connect functionality, we obviously need to install the Connect agent on our cluster. We also need to inform Google about our cluster and determine which project, and therefore which Fleets, the cluster belongs to. Fortunately, Google has provided a streamlined utility for performing this task via the `gcloud` command line tool⁹. This process will utilize either Workload Identity or a Google Cloud Service Account to enroll the cluster with the project's Connect pool and install and start the Connect agent on the cluster.

While these steps have enrolled the cluster with Google and enabled most Anthos features, you still need to authenticate with the cluster from the Google Console in order to view and interact with the cluster from Google Cloud. Connect allows authentication via Cloud Identity (when using the Connect Gateway¹⁰), bearer token, or OIDC, if enabled on the cluster. The easiest, and recommended, method is to use Cloud Identity, but this requires the activation and configuration of the Connect Gateway for the cluster. For more information on Connect Gateway, please see chapter 5, "Operations management with Anthos".

3.6 The Anthos Cloud UI

Now that we've done the plumbing, we can actually walk through and show off the UI. Google provides the Anthos UI via the Cloud Console, at the project level. Because the Anthos UI is only visible at the project level, only clusters registered to that project's Fleets are visible. We will discuss other tools later in this chapter to view clusters in multiple fleets simultaneously. The Anthos UI menu contains multiple sub-pages,

⁸ Access Transparency enabled services let customers control access to organization's data by Google personnel. It also provides logs that capture the actions Google personnel take when accessing customer's content.

⁹ See <https://cloud.google.com/anthos/multicloud-management/connect/registering-a-cluster>.

¹⁰ Google Cloud Identity and Access Management (IAM), lets you give more granular access to specific Google Cloud resources and prevents unwanted access to other resources.

each providing a focus on a distinct aspect of cluster management. At the time of writing, these sections are the Dashboard, Service Mesh, Config Management, Clusters, Features, Migrate to Containers, and Security. Let's look at each of these pages:

3.6.1 The Anthos Dashboard

The default page for the Anthos menu, and the central hub for the UI, is the Dashboard. The Dashboard is intended to give admins a wide-angle view of the clusters in the current Fleet, while making it easy to drill down into details for the specific components. To start, go to the hamburger menu on the top left corner of the console. Find “Anthos” from the menu and click on it to enter the all-Anthos features page. Figure 3.3 shows how to navigate to Anthos dashboard.

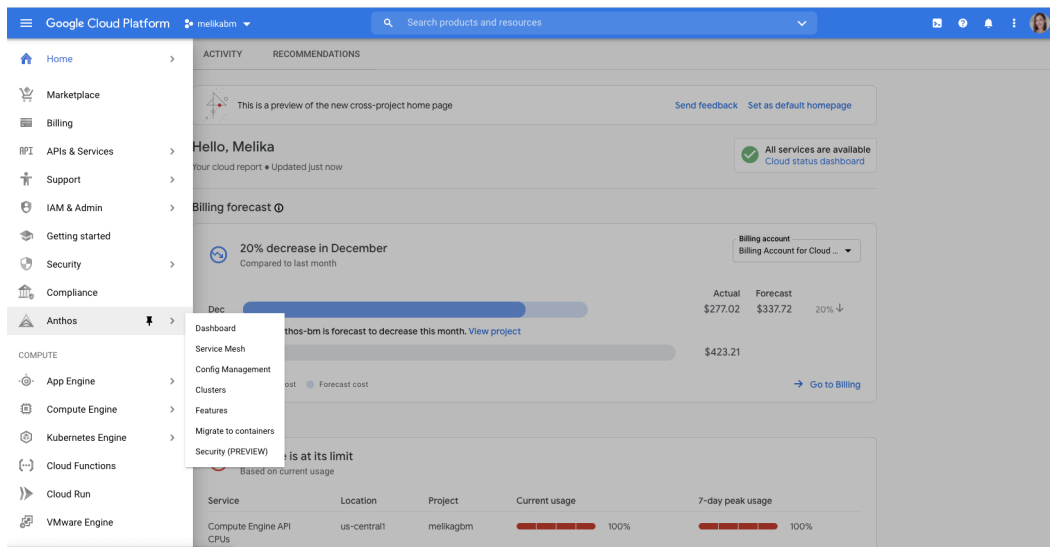


Figure 3.3 Navigation to Anthos Dashboard.

Figure 3.4 shows an example of the Anthos Dashboard view.

While this example shows the current Anthos project cost, the Dashboard still leverages Google's IAM and that information will only appear if the viewing user has the appropriate billing-related permissions. The remaining sections highlight critical errors or other user-involved issues for that particular aspect of Anthos. Following those links takes you to the appropriate sub-page explained in the following sections.

3.6.2 Service Mesh

The Service Mesh page shows all services registered in any of the clusters in the current Fleet. The initial list shows the names, namespaces, and clusters of each service, as well as basic metrics such as error rate and latency at predefined thresholds. This

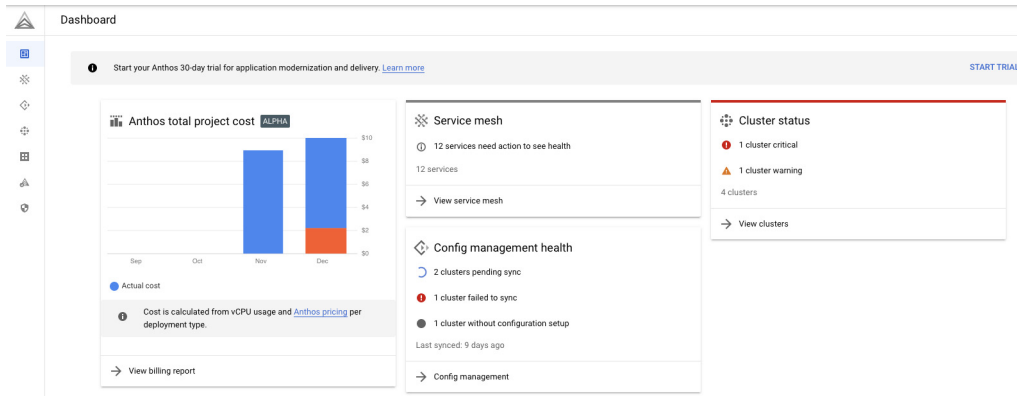


Figure 3.4 Example of an Anthos Dashboard.

list can also be filtered by namespace, cluster name, request per second, error rate, latency, request size, and resource usage to allow admins to easily drill down for specific tasks. Figure 3.5 shows the Service Mesh screen filtered for services in the default namespace.

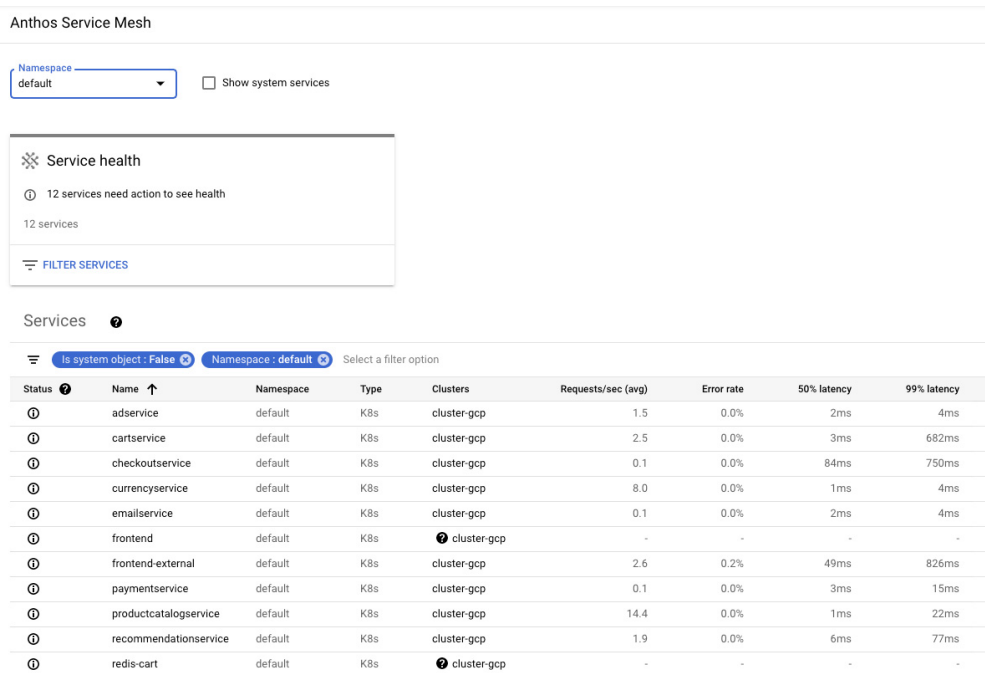


Figure 3.5 Service Mesh UI with filters.

3.6.3 Config Management

Anthos Configuration Management, explored in depth in chapter 14, is Anthos' method of automatically adding and maintaining objects on a Kubernetes cluster. These objects can include most common Kubernetes core objects (such as Pods, Services, ServiceAccounts, etc.) as well as custom entities such as policies and cloud-configuration objects. As shown in figure 3.6, this tab displays the list of all clusters in the current Fleet, their sync status, and which revision is currently enforced on the cluster. The table also shows whether Policy Controller¹¹ has been enabled for the cluster.

Anthos Config Management ⚙️ CONFIGURE				
Clusters for "melikabm"				
Filter table				
Cluster name ↑	Config sync status	Revision	Policy controller status	
<input type="radio"/> azure-cluster	Unreachable			
<input type="radio"/> cluster-1	Pending	master/47b472d55320c37fb8c064571c617669febd06f5	Disabled	
<input type="radio"/> cluster-gcp	Pending	master/47b472d55320c37fb8c064571c617669febd06f5	Disabled	
<input type="radio"/> externalazure	Not installed		Disabled	

Figure 3.6 Clusters in Config Management view.

Selecting a specific cluster opens up the Config Management cluster detail as shown in figure 3.7. This detailed view gives further information about the configuration settings, including the location of the repo used, the cycle for syncing, and the version of ACM running on the cluster.

3.6.4 Clusters

The Clusters menu lists all clusters in the current Fleet, along with the location, type, labels, and any warnings associated with each cluster as shown in figure 3.8. By selecting a cluster in the list, a more detailed view of the cluster, with the current Kubernetes version, the CPU and memory available, and the features enabled, will be displayed in the right sidebar as shown in figure 3.9. Below the sidebar information, a “Manage Features” button will take you to the Features tab for that cluster. In figure 3.8, clusters are created on the project:

- GKE (cluster-gcp)
- Baremetal (cluster-1)
- Azure AKS (azure-cluster and externalazure)

¹¹ Policy Controller is part of Anthos Configuration Management, allowing administrators to define customized rules to place guard-rails for security, resource management, or operational reasons. Policy Controller is explored in greater depth in chapter 14 with Anthos Configuration Management.

[←](#) Clusters
 [UPDATE CLUSTER](#)

[cluster-1](#)
[DETAILS](#)

Anthos config management

ACM

Version	1.5.2
---------	-------

Config sync

Cluster name	cluster-1
Status	Pending
Source format	
Sync repo	ssh://melikag@google.com@source.developers.google.com:2022/p/melikabm/r/config-repo
Revision	master/47b472d55320c37fb8c064571c617669febd06f5
Policy directory	.
Sync wait ?	
Secret type	ssh
Git proxy	

Policy controller

Status	Disabled
Audit Interval	
Template Library Installed	Disabled
Exemptable Namespaces	
Referential Rules Enabled	Disabled

Figure 3.7 Cluster detail in Configuration Management view.

Clusters BETA
[CREATE CLUSTER](#)
[REGISTER EXISTING CLUSTER](#)

Status

❗ 1 cluster critical
⚠ 1 cluster warning
 4 clusters total

Anthos managed clusters

Filter table

Name ↑	Location	Type	Labels
❗ azure-cluster	registered	Unknown	
✅ cluster-1	registered	Anthos	
✅ cluster-gcp	us-central1-c	GKE	
⚠ externalazure	registered	Unknown	

Figure 3.8 List view in the Clusters menu.

The screenshot shows the 'Clusters' menu in the Google Cloud console. The top navigation bar includes 'Clusters BETA', 'CREATE CLUSTER', and 'REGISTER EXISTING CLUSTER'. The main content area is divided into three sections:

- Status:** A summary box showing '1 cluster critical', '1 cluster warning', and '4 clusters total'.
- Anthos managed clusters:** A table listing clusters with columns for Name, Location, Type, and Labels. The table includes entries for 'azure-cluster', 'cluster-1' (highlighted), 'cluster-gcp', and 'externalazure'.
- Details:** A section for 'cluster-1' showing metadata: Type (Anthos), Master version (v1.18.6-gke.6600), Location (registered), Cluster Size (5), Total cores (40 CPU), and Total memory (157.81 GB).
- Cluster features:** A list of features with their status: Feature Authorizer (Enabled), Binary Authorization (Anthos Feature), Cloud Run (Anthos Feature), Config Management (Enabled), Ingress (Available), and Service Mesh (Anthos Feature). A 'Manage features' link is at the bottom.

Figure 3.9 Cluster detail view in the Clusters menu.

3.6.5 Features

The Anthos service encompasses several features, including:

- Configuration Management (chapter 14)
- Ingress (chapter 4)
- Binary Authorization (chapters 15 and 17)
- Cloud Run for Anthos (chapter 12)
- Service Mesh (chapter 4)

The Features menu provides an easy way to enable and disable specific services for the entire Fleet. Figure 3.10 shows the list of existing features for every cluster.

The screenshot shows the 'Features' menu in the Google Cloud console. The top navigation bar includes 'Features ALPHA' and a 'REFRESH' button. The main content area is titled 'Add and manage the Anthos features available in your fleet.' and contains a table of features:

Feature name	Status	Actions
Cloud Run for Anthos	Available	ENABLE
Config Management	Enabled	DETAILS
Identity Service	Available	ENABLE
Ingress	Available	ENABLE
Service Mesh	Available	ENABLE
Binary Authorization	Anthos Feature	View documentation
Feature Authorizer	Enabled	DETAILS

Figure 3.10 Feature menu.

An admin also has the ability to disable/enable most of these features from the interface (some features are integral components of Anthos and cannot be disabled). It's worth

noting that if enablement is not possible fully through the visual interface, the Console generates the right commands for the admin to seamlessly enter them into their CLI.

3.6.6 Migrate to Containers

One of the major benefits to Anthos is the automatable migration of Windows and Linux VMs to containers and their deployment onto a compatible Anthos cluster. Previously, this has primarily been done via CLI and initiated from the source cluster, but this menu now provides a convenient, centralized process for shifting VMs to containers and into a different deployment scheme. This menu contains tabs for viewing and managing your migrations, sources, and processing clusters. For more information on the process of migrating your existing VMs to containers, see chapter 21, “Migrate for Anthos”.

3.6.7 Security

The security menu is where we find multiple tools related to viewing, enabling, and auditing the security posture of the clusters in the current Fleet. Figure 3.11 shows the basic view when the Security menu is first selected.

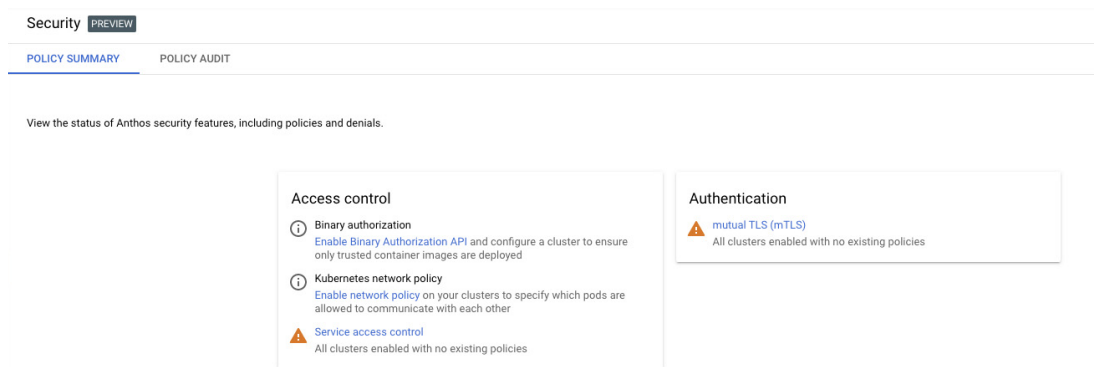


Figure 3.11 Security menu.

As you can see, we do not currently have Binary Authorization¹² enabled, but Anthos provides a shortcut here to quickly turn it on. Once we do, we are presented with the configuration page for Binary Authorization enabling us to view and edit the policy, if needed (figure 3.12).

3.7 Monitoring and logging

The Anthos menu in the GCP Console is only part of the solution, however. Google also provides the Operations suite, including Cloud Monitoring and Cloud Logging, in order to help manage the operations of applications and infrastructure. Anthos

¹² Binary Authorization is explored in further detail in chapter 15, “Anthos integration with CI/CD”.

Binary Authorization	
POLICY	ATTESTORS
Binary Authorization lets you restrict which images can be deployed to a Kubernetes Engine cluster by making sure they pass through the appropriate checkpoints in your deployment workflow. Learn more	
Policy deployment rules	EDIT POLICY
Project default rule	Allow all images
Cluster-specific rules	us-central1-c.cluster-gcp Allow all images Dry run mode: Disabled
Dry run mode	Not enabled
Images exempt from policy	Google-provided system images VIEW DETAILS gcr.io/google_containers/* gcr.io/google-containers/* k8s.gcr.io/* gke.gcr.io/* gcr.io/stackdriver-agents/*

Figure 3.12 Binary Authorization policy details.

simplifies the logging of application data and metrics to the Operations suite as part of the default deployment. This can make it simple to add SLOs and SLAs based on these metrics¹³. In addition, several pages within the Anthos menu include shortcuts and buttons that trigger wizards to create SLOs in a guided fashion.

3.8 *GKE Dashboard*

Google has provided the GKE Dashboard for several years to assist with viewing and managing your clusters for GKE in GCP. With the release of Anthos, the GKE Dashboard has been expanded to be able to view the details for Kubernetes clusters attached via GKE Connect. While the Anthos menu is focused on the clusters at a high-level and on the Anthos-specific features, such as the Service Mesh and Configuration Management, the GKE Dashboard allows an admin to drill down to specific workloads and services. The next section is a simple tutorial to register an Azure AKS cluster into Anthos dashboard.

3.9 *Connecting to a remote cluster*

In this example, a cluster is already created in the Azure Kubernetes engine (AKS). Google supports below cluster types to be registered remotely, referred to as Attached Clusters:

- Amazon Elastic Kubernetes Service (Amazon EKS) on Kubernetes versions 1.18, 1.19, 1.20
- Microsoft Azure Kubernetes Service (Microsoft AKS) on Kubernetes versions 1.18, 1.19, 1.20

¹³ See chapter 5 for details on SLIs, SLOs, and SLAs with Anthos.

- Red Hat OpenShift Kubernetes Engine (OKE) version 4.6, 4.7
- Red Hat OpenShift Container Platform (OCP) version 4.6, 4.7
- Rancher Kubernetes Engine (RKE) version 1.2.4, 1.2.5, 1.2.6
- KIND version 0.10

Step 1: Use the following command to register your cluster:

```
gcloud container hub memberships register MEMBERSHIP_NAME \
  --context=KUBECONFIG_CONTEXT \
  --kubeconfig=KUBECONFIG_PATH14 \
  --service-account-key-file=SERVICE_ACCOUNT_KEY_PATH
service account key that you need to create beforehand with the role: --
  role="roles/gkehub.connect"
```

This command also stores the service account key by creating a secret named “creds-gcp” in the “gke-connect” namespace.

It takes a few minutes, and your cluster appears on the GCP console as displayed in figure 3.13.

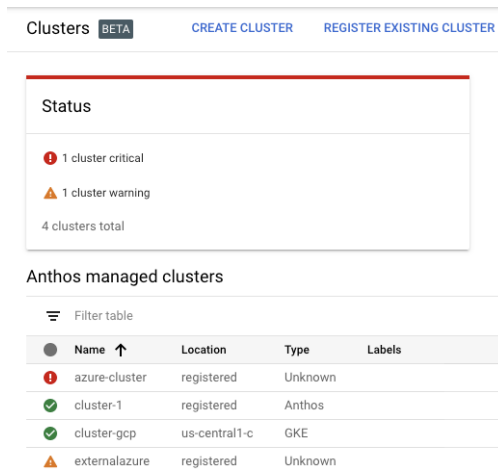


Figure 3.13 Registered cluster view.

Step 2: Authenticate to the registered cluster. As you can see, there is a warning sign next to the recently created cluster (externalazure). That is normal and a reminder to sign into the cluster to be able to perform more operations on the cluster. Figure 3.14 shows the view of the registration status of the cluster.

By clicking on the login menu, you can see what login options are available:

- Use your Google identity to log in
- Token

¹⁴ The local file path where your kubeconfig containing an entry for the cluster being registered is stored. This defaults to \$KUBECONFIG if that environment variable is set; otherwise, this defaults to \$HOME/.kube/config.

Kubernetes clusters								
+ CREATE + DEPLOY REFRESH DELETE REGISTER								
Filter table								
<input type="checkbox"/>	Name ↑	Location	Type	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
<input type="checkbox"/>	azure-cluster	registered	Unknown	unknown	unknown	unknown	Unreachable Agent	—
<input type="checkbox"/>	cluster-1	registered	Anthos	5	40	157.81 GB		—
<input type="checkbox"/>	cluster-gcp	us-central1-c	GKE	4	16	64 GB		asmv:1-7-3-asm-6
<input type="checkbox"/>	externalazure	registered	Unknown	unknown	unknown	unknown	Log in to cluster	—

Figure 3.14 View of the registration status of the cluster.

- Basic authentication
- Authenticate with Identity Provider configured for the cluster

Let's go ahead and authenticate with a token. In order to do that you need to have a KSA with the right permissions¹⁵:

```
KSA_NAME=[KSA_NAME]
kubectl create serviceaccount ${KSA_NAME}
kubectl create clusterrolebinding [VIEW_BINDING_NAME] \
--clusterrole view --serviceaccount default:${KSA_NAME}
kubectl create clusterrolebinding [CLOUD_CONSOLE_READER_BINDING_NAME] \
--clusterrole cloud-console-reader --serviceaccount default:${KSA_NAME}
```

Once you have created the KSA, acquire the KSA's bearer token:

```
SECRET_NAME=$(kubectl get serviceaccount [KSA_NAME] -o
jsonpath='{$.secrets[0].name}')
kubectl get secret ${SECRET_NAME} -o jsonpath='{$.data.token}' | base64 --
decode
```

Once, you have pasted the token in the login, you immediately get the same view in your AKS cluster (externalazure) that you would see in other cluster types. Figure 3.15 provides that view.

Figure 3.16 shows the nodes and their health status through the dashboard.

Several other types of Kubernetes clusters that are not managed by GCP can be attached to Anthos this way. It gives operations simplicity, consistency and access security to administrators from a single platform.

¹⁵ All accounts logging in to a cluster need to hold at least the following Kubernetes RBAC roles in the cluster:

- *View*: Kubernetes primitive role that allows read-only access to see most objects in a namespace. It does not allow viewing roles or role bindings.
- *Cloud-console-reader*: Users who want to view your cluster's resources in the console need to have the relevant permissions to do so. You define this set of permissions by creating a ClusterRole RBAC resource, cloud-console-reader, in the cluster. cloud-console-reader grants its users the get, list, and watch permissions on the cluster's nodes, persistent volumes, and storage classes, which allow them to see details about these resources.

Kubernetes clusters									
CREATE DEPLOY REFRESH DELETE REGISTER									
Filter table									
<input type="checkbox"/>	<input type="checkbox"/>	Name ↑	Location	Type	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
<input type="checkbox"/>	<input checked="" type="checkbox"/>	azure-cluster	registered	Unknown	unknown	unknown	unknown	Unreachable Agent	—
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cluster-1	registered	Anthos	5	40	157.81 GB		—
<input type="checkbox"/>	<input checked="" type="checkbox"/>	cluster-gcp	us-central1-c	GKE	4	16	64 GB		asmv:1-7-3-asm-6
<input type="checkbox"/>	<input checked="" type="checkbox"/>	externalazure	registered	External	3	6	21.92 GB		—

Figure 3.15 Anthos attached cluster authenticated.

Kubernetes cluster details							
DEPLOY LOGOUT DISCONNECT REFRESH							
externalazure							
DETAILS STORAGE NODES							
Nodes							
Filter nodes							
Name ↑	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested	Storage allocatable
aks-agentpool-17822882-vmss000000	Ready	845 mCPU	1.9 CPU	1.13 GB	4.8 GB	0 B	0 B
aks-agentpool-17822882-vmss000001	Ready	749 mCPU	1.9 CPU	524.29 MB	4.8 GB	0 B	0 B
aks-agentpool-17822882-vmss000002	Ready	685 mCPU	1.9 CPU	638.58 MB	4.8 GB	0 B	0 B

Figure 3.16 Node view on attached cluster.

Summary

- Providing a single pane of glass to hybrid and multi-cloud Kubernetes for any organization who uses microservices is a stepping stone to a successful and global operation.
- One of the biggest benefits to a single pane of glass is that admins can use the same interface to configure service level objectives and alerts to reassure service guarantees.
- Anthos UI provides some major advantages including:
 - Central operation of services and resources
 - Consistent operation experience across multiple service providers
 - Effortless navigation and easy staff training
 - Providing a window to any organizational persona
- Anthos UI provides multiple usages including cluster management, service operation, and observability using a unified interface.
- An admin can deploy applications directly from Anthos UI to any registered cluster in a few easy steps.

Anthos Service Mesh: Security and observability at scale

by Onofrio Petragallo

This chapter covers

- Discussing sidecar proxy and proxyless architectures
- Introducing the main features of Istio
- Learning about security and observability with Istio
- What is Anthos Service Mesh?
- Working through practical examples with code

One of the key components to being cloud native is to break up your application into microservices. This means that an application that may have run on a single server, now has multiple services, backed by multiple pods, that are separate components. As applications scale out their services, it becomes difficult to troubleshoot issues that you may encounter with the application. With this added complexity, we needed a tool that helped organize, secure, and add resilience to the expanding complexities that microservices introduced. Another important problem that a service mesh could fix is the fact that enterprises often have a huge number of microservices and aren't always able to control, manage, and observe them.

In this chapter, we will discuss Anthos Service Mesh (ASM), and the features that ASM inherits from Istio¹⁶, a popular open-source framework for creating, managing, and implementing a service mesh.

The implementation of a service mesh not only facilitates communication between microservices using a dedicated communication management control plane, it also includes tools to observe communication between services, aka observability, enhance security, control application traffic flow, and simulate faults in an application.

Anthos Service Mesh is a Google-managed service that allows enterprise management of all the service meshes present in a hybrid cloud or multi cloud architecture from a single point, providing complete and in-depth visibility of all microservices. The visualization of the topography of the service mesh and the complete integration with cloud monitoring, provides users the tools to identify failing workloads or other problems, making problem resolution faster.

The security features made available by Anthos Service Mesh allow you to manage the authentication, authorization, and encryption of communications through mTLS¹⁷ mutual authentication to secure and ensure trust in both directions for the communication between microservices; mTLS ensures a high level of security, minimizing the related risks.

Finally, the traffic management features of Istio provide users the tools to manipulate traffic using request routing, fault injection, request timeouts, circuit breaking, and mirroring.

As you can see, Istio includes a number of features that are complex and may be difficult to troubleshoot. There are few offerings in the market today that include support for Istio, leaving you to support your service mesh on your own. Google has addressed this by including ASM in Anthos, providing a single support point for your Kubernetes clusters and Istio.

Before talking about the Anthos Service Mesh features and Istio in detail, let's start by explaining what a service mesh actually is.

Technical requirements

The hands-on portion of this chapter will require you to have access to a Kubernetes cluster running on GCP with the following deployment pattern:

- A GKE cluster with at least 3 nodes with 4 CPUs and 16GB of RAM

4.1 What is a service mesh?

To understand how a service mesh works, and why it's becoming a standard tool in the microservices toolbox, you need to understand what a service mesh provides.

The main advantages of adopting a service mesh are the ability to:

- 1 Observe and monitor all communications between the individual microservices
- 2 Secure connections between the available microservices

¹⁶ See <https://istio.io/>.

¹⁷ See https://en.wikipedia.org/wiki/Mutual_authentication.

- 3 Deliver resilient services (Distributed services) through multi-cluster and multi-cloud architecture patterns
- 4 Advanced traffic management through A/B testing, traffic splitting, and canary rollouts

A service mesh is an infrastructure layer in a microservices architecture that controls the communication between services. It is possible to create a mesh of services within a microservices architecture that not only runs in a Kubernetes cluster, but it is also possible to create a single service mesh that spans multiple clusters or even non-microservice services running on virtual machines.

A service mesh manages all the ingress, or inbound traffic, and egress, or outbound traffic for each microservice. Traffic management is a complex topic, and something that most users do not want to deal with. To remove this burden, Istio doesn't require the developer to make any changes in their application logic—instead, the service mesh handles all of this by using a sidecar proxy approach or a proxyless approach.

The sidecar proxy is one of the main components of a service mesh that manages the ingress and egress traffic for each microservice, abstracting itself from the application logic of the microservices. Because all the traffic flows through the sidecar proxy, it can monitor all the traffic to send metrics and logs to the centralized control plane.

While the sidecar proxy is the most common approach to create a service mesh, it's not the only approach that is available. There are three approaches to create a service mesh:

- *Sidecar Proxy*: In a microservices architecture where a proxy is connected to each microservice, with the same life cycle as the microservice itself, but executes as a separate process (figure 4.1).

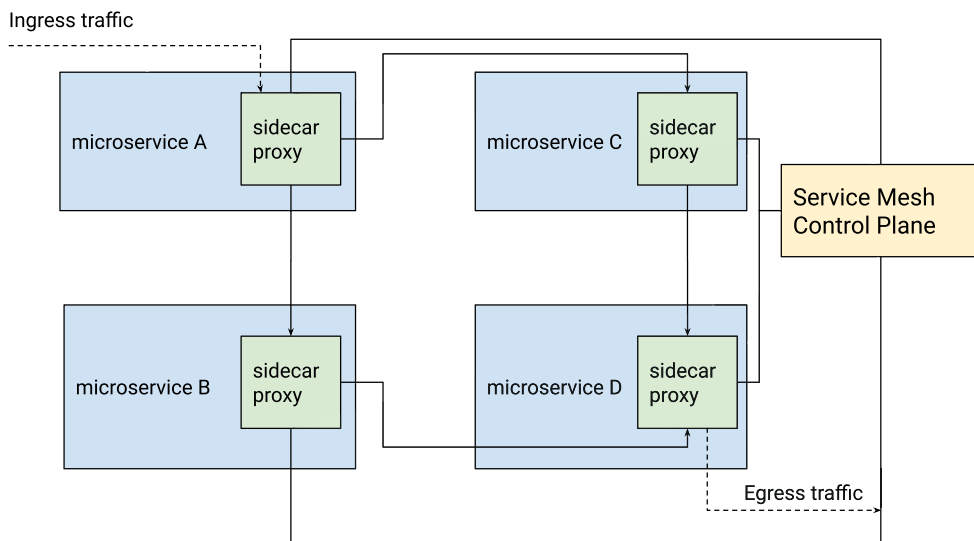


Figure 4.1 Sidecar proxy architecture.

- *Proxyless*: In a microservices architecture where the microservice can send the telemetry directly to the control plane by using gRPC, a remote procedure call (RPC) system developed by Google.
- *Proxy inside a VM*: An L7 proxy runs inside VMs as a process or an agent, which can be added to the service mesh as if it were a sidecar proxy.

After seeing what a service mesh is and what the approaches are to create a service mesh, let's review how Anthos Service Mesh uses each of them.

To monitor in real time the telemetry of all inbound and outbound communications between the microservices of the various service mesh networks, ASM uses two approaches:

- Anthos Service Mesh can use the sidecar proxy approach, using Envoy¹⁸ proxies, an open-source service proxy attached on each pod to get the real time telemetry (figure 4.2).

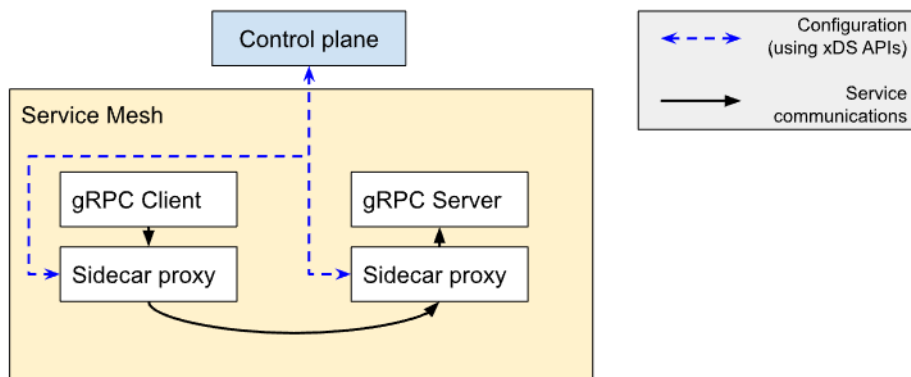


Figure 4.2 Sidecar proxy approach.

- Anthos Service Mesh can use a proxyless approach using Google Cloud Traffic Director¹⁹ that is able to use gRPC, with xDS API²⁰, the same technology used by Envoy to communicate with the control plane (figure 4.3).

As shown in figure 4.3, the proxyless approach removes the Istio sidecar from your deployments. By removing the sidecar, we are removing an extra hop in the network traffic to the service, leading to a reduction in network latency, enhancing the service overall response time.

A single service mesh can have services that both use the standard Istio sidecar and the proxyless approach. This allows you to use the correct approach for different

¹⁸ See <https://www.envoyproxy.io/>.

¹⁹ See <https://cloud.google.com/traffic-director>.

²⁰ See <https://github.com/envoyproxy/data-plane-api>.

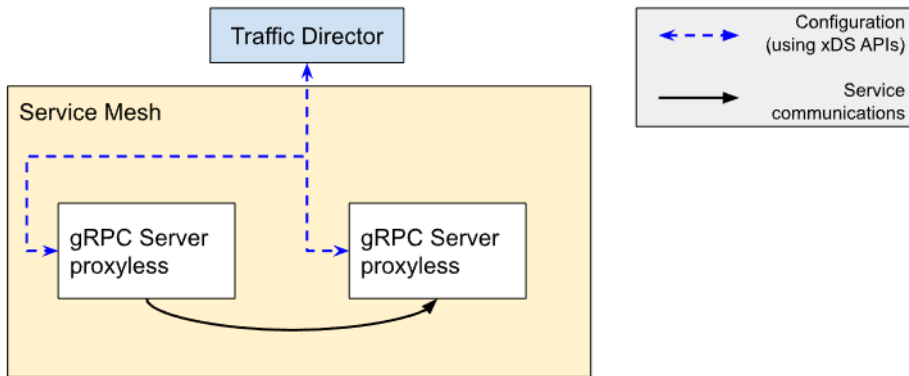


Figure 4.3 Proxyless approach.

applications, including gRPC using a proxyless approach, sidecars for services that do not use gRPC, and sidecars for services that use gRPC.

As discussed in previous chapters, Anthos is a complete platform from Google to build applications running on hybrid or multi-cloud platforms. Anthos Service Mesh is the main component that provides service management to developers and cluster administrators.

In the next section, we will examine the features of Istio, which is the basis for the Anthos Service Mesh.

4.2 *An introduction to Istio*

Istio is an open platform for providing a uniform way to integrate microservices, manage traffic flow across microservices, enforce policies, and aggregate telemetry data. Istio's control plane provides an abstraction layer over the underlying cluster management platform, such as Kubernetes.

The main features²¹ of Istio are:

- Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic
- Fine-grained control of traffic behavior with robust routing rules, retries, failover, and fault injection
- A pluggable policy layer and configuration API supporting access controls, rate limits, and quotas
- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress
- Secure service-to-service communication in a cluster with strong identity-based authentication and authorization

²¹ See <https://istio.io/latest/docs/concepts/what-is-istio/#why-use-istio>.

Given its open-source nature, Istio is extensible and usable in various environments; for example, you can execute it on-prem, in the cloud, inside a VM or with microservices, and more allowing you to customize the service mesh to your security and monitoring requirements.

To understand Istio, you need to understand the underlying architecture of the system. In the next section, we will explain the components of Istio and the features they provide.

4.2.1 Istio architecture

Istio's flexible architecture allows you to implement a service mesh from scratch. You do not have to have Istio installed before developers start using the cluster, a service mesh can be deployed before or after the developers have implemented deployed their services. Remember, Istio uses the sidecar proxy injection to intercept the ingress and egress traffic from inside and outside the network of microservices, so there are no dependencies on the developers (figure 4.4).

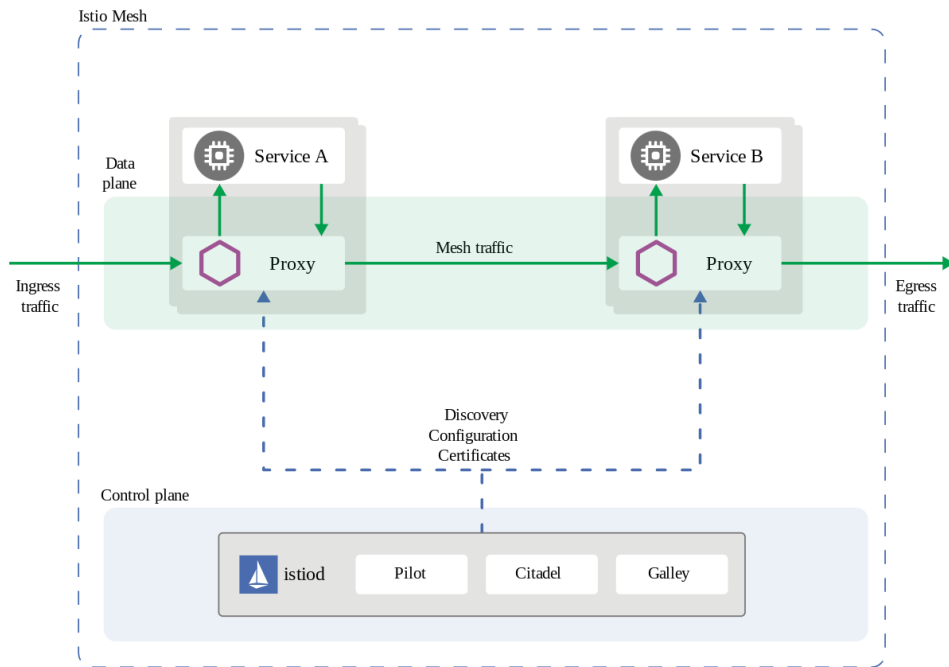


Figure 4.4 Detailed Istio architecture.

Starting with Istio version 1.5, the main components of Istio's architecture are the sidecar proxy and Istiod, which contains three sub-components: *Pilot*, *Citadel*, and *Galley*.

Istiod provides service discovery, configuration, and certificate management. It can provide high-level routing rules that control traffic behavior into a specific configura-

tion for the Envoy proxy, injecting them into the sidecars at runtime. Istiod also acts as a Certificate Authority (CA) that generates certificates to allow secure mTLS communication in the data plane.

Istiod contains three processes to provide its services:

- *Pilot*: Responsible for the lifecycle of Envoy sidecar proxy instances deployed across the service mesh. Pilot abstracts platform-specific service discovery mechanisms and is conformant with any sidecar that can consume Envoy API.
- *Galley*: Interprets the YAML files for Kubernetes and transforms them into a format that Istio understands. Galley makes it possible for Istio to work with other environments than Kubernetes since it translates various configuration data into the common format that Istio understands.
- *Citadel*: Enables robust service-to-service and end-user authentication with built-in identity and credential management.

So far, we have covered Istio at a high level, but now let's go through the features of Istio in detail. We can divide the features into three main categories: traffic management, security, and observability.

4.2.2 Istio traffic management²²

Istio provides powerful traffic management that allows users to control ingress and egress traffic. This control is not limited to simply routing traffic to a specific service, it also provides the ability to split traffic between different versions and simulate failures and timeouts in applications. Traffic management consists of the following features, as shown in table 4.1.

Table 4.1 Istio Traffic Management features

Feature	Description
Ingress	Controls the ingress traffic for the service mesh, to expose a service outside the service mesh over TLS or mTLS using the Istio gateway. In the chapter “Hybrid Applications in Anthos: The Edge and Beyond”, we will deep dive into Ingress for Anthos.
Egress	Controls the egress traffic from the service mesh, route traffic to an external system, perform TLS for outbound traffic, configure the outbound gateway to use an HTTPS proxy.
Request routing and traffic splitting	Dynamically route the traffic to multiple versions of the microservice or migrate traffic from one version to another version gradually and in a controlled way.
Fault Injection	Provides configurable HTTP delays and fault injection, allowing developers to discover problems before they would occur in production.

Traffic management is a powerful feature of Istio, allowing developers to completely control traffic, down to the level where a single user could be directed to a new ver-

²² See <https://istio.io/latest/docs/tasks/traffic-management/>.

sion of an application, while all other requests are directed to the current version. The fault injection feature provides developers the ability to cause a delay between services, simulating an HTTP delay or faults to verify how the application will react to unexpected issues. All these features can be used by users without any code changes to their application, providing a big advantage over the old “legacy” development days.

Security is everyone’s job, but not everyone has a background in security to create the code to enhance application security. Just like the traffic management features, Istio provides extra security features, all without developers needing to create any code.

In the next section, we will explain the security features included with Istio and ASM.

4.2.3 Istio security²³

Because services in the mesh need to communicate with each other over a network connection, you need to consider additional security to defend against various attacks, including man-in-the-middle and unknown service communication. Istio includes components to enhance your application security, ranging from an included certificate authority, peer authentication, and authorization.

The first component that Istio provides to increase your application security is the handling of certificate management, including an Istio’s certificate authority (CA) with an existing root certificate. In cryptography, a certificate authority²⁴ or certification authority is an entity that issues digital certificates. A digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others to rely upon signatures or on assertions made about the private key that corresponds to the certified public key—proving the identity of the certificate owner. A CA acts as a trusted third party—trusted both by the owner of the certificate and by the party relying upon the certificate.

Certificates follow a format known as the X.509²⁵ or EMV standard. In an organization, the root certificate signatory may want to remain responsible for signing all certificates that are issued for all entities in the organization. It is also possible that whoever has the responsibility of signing the root certificate wants to delegate the responsibility of signing the certificates of a subordinate entity; in this case we refer to the subordinate entity as a “delegate CA”.

Istio’s certificate authority can be configured in multiple ways, by default, the CA generates a self-signed root certificate and key, which is used to sign the certificates for microservices. Istio’s CA can also sign certificates using an administrator-specified certificate and key, and with an administrator-specified root certificate. The last configuration is most common in enterprise environments, where the CA is configured with an existing root certificate or delegated CA signing certificate and key.

²³ See <https://istio.io/latest/docs/tasks/security/>

²⁴ See https://en.wikipedia.org/wiki/Certificate_authority.

²⁵ See <https://en.wikipedia.org/wiki/X.509>.

The CA in Istio is used to securely provision strong identities to every workload in the mesh. Certificates are issued using the X.509 certificates, which is a standard that defines the format of public key certificates. X.509 certificates are used in many Internet protocols, including TLS/SSL, which is the basis for HTTPS, the secure protocol for browsing the web.

Istio agents, running alongside each Envoy proxy, work together with the Istio CA component of Istiod to automate key and certificate rotation at scale. Because the rotation and distribution of certificates is automated, once configured, there is little overhead for operators or users of the cluster. This is a powerful feature of Istio to secure communications between services.

Certificates are the building block for additional security in our service mesh. In the next section we will discuss authentication and mutual TLS encryption—a security layer that relies on the issued certificates to secure the mesh workloads.

4.2.4 Istio authentication

Istio uses peer authentication for service-to-service authentication and to verify the client initiating the connection. Istio also makes mutual TLS (mTLS) available as a full-stack solution for transport authentication, which can be enabled without requiring changes to any application code. Peer authentication provides:

- Each service with a strong identity that represents its role to enable interoperability inside the clusters.
- Protection of all communication between services.
- A key management system to automate the generation, distribution, and rotation of keys and certificates.

Istio allows request-level authentication with JSON Web Token (JWT) validation with many authentication providers (e.g., Google Auth²⁶).

4.2.5 Istio authorization

Istio provides a mechanism for operators to define authorization policies to control access to the service mesh, namespace, and workloads within the service mesh (figure 4.5). Traffic can be restricted by type, such as TCP or HTTP, and the identity of the requestor.

The advantages that authorization provides are:

- Authorization between workloads and from user to workload
- Flexible semantics in which operators can define custom conditions on Istio attributes and use the DENY and ALLOW actions to tune the policies to suit their needs
- High performance because Istio authorization is applied natively on the Envoy proxy

²⁶ See <https://developers.google.com/identity>.

- Flexibility supports gRPC, HTTP, HTTPS, and HTTP2 natively as well as all regular TCP protocols
- Distributed in that each Envoy proxy runs its own authorization engine that authorizes each request to be executed

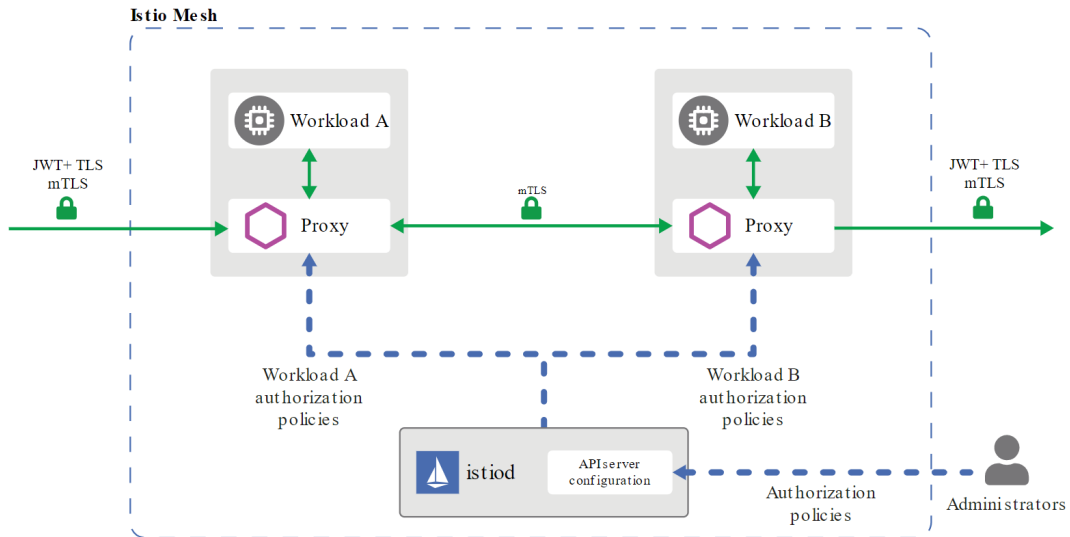


Figure 4.5 Istio Authorization architecture.

At the beginning of the chapter, we mentioned that one advantage of a service mesh was the ability to organize microservices. As your number of services grows, so does the complexity and the only way to maintain health and to troubleshoot issues in services is to have a powerful set of tools that allow you to look deep into the mesh activities. In the next section, we will go over the tools that you can use with Istio to view the activities in the mesh.

4.2.6 Istio observability²⁷

To offer a view into the service mesh, Istio has multiple add-on components that provide distributed tracing, metrics and logging, and a dashboard.

Istio offers a few options that provide a mesh with distributed tracing. Distributed tracing allows you to track the user through all the services and understand request latency, serialization, and parallelism. Istio can be configured to send distributed metrics to different systems including Jaeger²⁸, Zipkin²⁹, or Lightstep³⁰.

²⁷ See <https://istio.io/latest/docs/tasks/observability/>.

²⁸ See <https://www.jaegertracing.io/>.

²⁹ See <https://zipkin.io/>.

³⁰ See <https://lightstep.com/>.

Jaeger is a distributed tracing system released as open source by Uber Technologies. It is used for monitoring and troubleshooting microservices-based distributed systems, including those features: distributed context propagation, distributed transaction monitoring, root cause analysis, service dependency analysis, and performance optimization.

Zipkin and Lightstep are other distributed tracing systems. They help gather timing data needed to troubleshoot latency problems in service architectures. Features include both the collection and lookup of this data.

Metrics and Logs collect all the metrics and logs from Envoy proxy and TCP session. You can customize the metrics using Istio Metrics making available all the data via Kiali³¹, Prometheus³², or Grafana³³.

Kiali is a management console for Istio-based service meshes (figure 4.6).

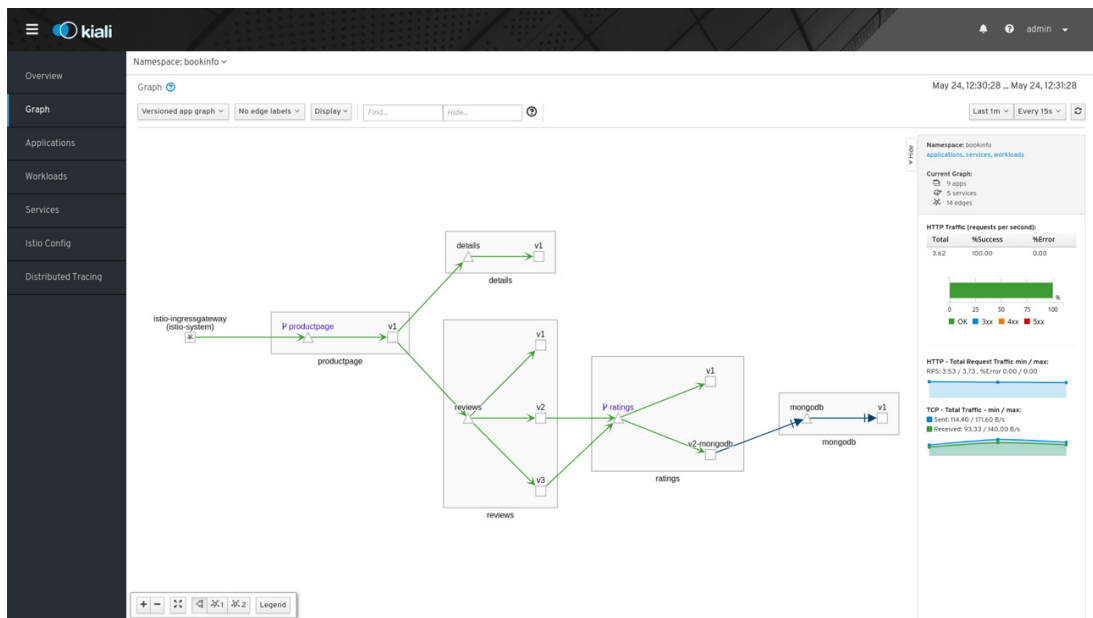


Figure 4.6 Kiali Console UI.

It provides dashboards and observability and lets you operate your mesh with robust configuration and validation capabilities. It shows the structure of your service mesh by inferring traffic topology and displays the health of your mesh. Kiali provides detailed metrics, powerful validation, Grafana access, and strong integration for dis-

³¹ See <https://kiali.io/>.

³² See <https://prometheus.io/>.

³³ See <https://grafana.com/>.

tributed tracing with Jaeger. Kiali allows you to use Kubernetes JWT tokens to provide native RBAC permission. The JWT presented by the user allows access to all namespaces they have access to in the cluster while denying all users that do not have permissions to the namespaces—all without any configuration by cluster admins.

Prometheus is an open-source systems monitoring and alerting that has those principal features: a multi-dimensional data model with time series data identified by metric name and key/value pairs, and a flexible query language to leverage this dimensionality named PromQL. The time series collection happens via a pull model over HTTP and pushing time series is supported via an intermediary gateway.

Grafana is open-source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored. It provides you with tools to turn your time-series database (TSDB), data into beautiful graphs and visualizations. Grafana can connect with Prometheus and Kiali.

Now that we have taken a look at Istio, let's see what features and advantages of Istio are when it is used by Anthos Service Mesh, managed by Google Cloud.

4.3 What is Anthos Service Mesh?

Anthos Service Mesh has a suite of features and tools that help you observe and manage secure, reliable services in a unified way. With Anthos Service Mesh, you get an Anthos tested and supported distribution of Istio, managed by Google, letting you create and deploy a service mesh on GKE on Google Cloud and other platforms with full Google support.

The use of Istio features, in Anthos Service Mesh, varies according to the architecture you want to design and implement: *full cloud*, *multi cloud*, *hybrid cloud*, or *edge*. Each implementation will have different available features; therefore, it is necessary to check the availability of the supported features³⁴ for the various scenarios.

Before installing Anthos Service Mesh, always check the documentation and choose the most suitable and updated configuration profile. Configuration profiles are YAML files that are used by the IstioOperator API, defining and configuring the features that are installed with Anthos Service Mesh.

At time of writing, you can install ASM in different scenarios:

- Anthos cluster (GKE) on Google Cloud in a single project
- Anthos cluster (GKE) on Google Cloud between different projects
- Anthos cluster (GKE) on VMware
- Anthos cluster (GKE) on bare metal
- Anthos cluster (GKE) on AWS
- Attached cluster Amazon Elastic Kubernetes Service (Amazon EKS)
- Attached cluster Microsoft Azure Kubernetes Service (Microsoft AKS)

³⁴ See <https://cloud.google.com/service-mesh/docs/supported-features>.

4.3.1 Installing ASM

ASM installs differently on GKE clusters on GCP and on-premise. You can view the most current installation procedures on the ASM site at <https://cloud.google.com/service-mesh/docs/scripted-install/asm-onboarding>. Explaining each option of the installation is beyond the scope of a single chapter, but the steps to deploy ASM on a GKE cluster with all components for testing only requires a few steps. The steps below will install ASM 1.9 in a cluster:

- 1 Downloading the ASM installation scripts

```
curl https://storage.googleapis.com/csm-artifacts/asm/install_asm_1.9 >
install_asm
```

- 2 Make the script executable

```
chmod +x install_asm
```

- 3 Install ASM using the install_asm script

```
./install_asm --project_id PROJECT_ID --cluster_name CLUSTER_NAME --
cluster_location CLUSTER_LOCATION --mode install --output_dir
./asm-downloads --enable_all
```

After Istio is deployed into the Kubernetes cluster, it is possible to start configuring and using it right away. One of the first things to do is to define which approach we want to follow to make proxies communicate within the service mesh.

In the next section, we will define how Istio will handle the sidecar proxy injection.

4.3.2 Sidecar proxy injection

Activating Anthos Service Mesh features is an easy, transparent process, thanks to the possibility of injecting a sidecar proxy next to each workload or microservice.

You can inject a sidecar proxy manually by updating your Pods' Kubernetes manifest or you can use automatic sidecar injection. By default, sidecar auto-injection is disabled for all namespaces. To enable auto-injection for a single namespace, you can execute the following command:

```
kubectl label namespace NAMESPACE istio.io/rev=asm-managed --overwrite
```

where NAMESPACE is the name of the namespace for your application's services and rev=asm-managed is the release channel³⁵.

All channels are based on a generally available (GA) release (although individual features may not always be GA, as marked). New Anthos Service Mesh versions are first released to the Rapid channel, and over time are promoted to the Regular and Stable channel. This allows you to select a channel that meets your business, stability, and functionality needs.

³⁵ See https://cloud.google.com/service-mesh/docs/release-channels-managed-service-mesh#available_release_channels.

After you execute the command, because sidecars are injected when pods are created, you must restart any running pods for the change to take effect. When Kubernetes invokes the webhook, the `admissionregistration.k8s.io/v1beta1#MutatingWebhookConfiguration` configuration is applied. The default configuration injects the sidecar into pods in any namespace with the `istio-injection=enabled` label. The `istio-sidecar-injector` configuration map specifies the configuration for the injected sidecar.

The way you restart pods depends very much on the way they were created such as:

- 1 If you used a deployment, you should update or recreate the deployment first, which will restart all Pods, adding the sidecar proxies:

```
kubectl rollout restart deployment -n YOUR_NAMESPACE
```

- 2 If you didn't use a deployment, you should delete the pods, and they will be automatically recreated with sidecars:

```
kubectl delete pod -n YOUR_NAMESPACE --all
```

- 3 Check that all the pods in the namespace have sidecars injected:

```
kubectl get pod -n YOUR_NAMESPACE
```

- 4 In the following example output from the previous command, you will notice that the `READY` column indicates there are two containers for each of your workloads: the *primary container* and the container for the *sidecar proxy*:

NAME	READY	STATUS	RESTARTS	AGE
YOUR_WORKLOAD	2/2	Running	0	20s

We have seen how to install Anthos Service Mesh, using the approach with a sidecar proxy and how important it is to choose the right profile. Now let's see what the other features of Anthos Service Mesh are and what the advantages are of using them.

4.3.3 Uniform observability

One of the most important and useful features of the Anthos Service Mesh is observability. Implementing a service mesh through the proxy architecture and taking advantage of the Google Cloud Monitoring services ensures an in-depth visibility of what is happening among the various microservices that are present in the mesh.

Through the proxy, each microservice can send telemetry automatically, without the developers adding any code in their application. All traffic is intercepted by proxies and the telemetry data is sent to Anthos Service Mesh. Each proxy sends the data to Google Cloud Monitoring and Google Cloud Logging without any extra development using the APIs that Google makes available.

As we have seen in chapter 3, the Anthos Service Mesh control plane provides two main dashboards: table view and topology view.

In the “table view” you have a complete view and intuitive look at all the services that are deployed in the cluster. You can see all the metrics and you can add SLI and SLO to better monitor your services.

In the “topology” view, service meshes are represented as a graphical map. All the services such as workloads, pod, systems services, and relative owners are connected as a network of nodes. This enables a comprehensive look at the overall performance of the entire service mesh and an inside look into each node with detailed information.

4.3.4 Operational agility

If observability is one of the most “visible” features to manage a service mesh, then the management of traffic “to / from” within a microservices architecture is another fundamental asset to manage operations easily.

Since Anthos Service Mesh is based on Istio, it inherits most³⁶ of the traffic and network management features that Istio provides, so let’s look at what these features are.

4.3.5 Request routing and traffic splitting

Istio provides the ability to redirect traffic to multiple versions of the same microservice deployed in the cluster; the ability to safely (under possible quota) control part of the traffic from an old version of the microservice to a newly installed version. Both of these options allow you to be agile in the deployment of new features or to fix bugs that may impact the business.

For example, let’s imagine we need to urgently fix a microservice. After the deployment of the new version, we can redirect a small part of the incoming traffic, verify that the fix works correctly and then completely redirect the traffic to the new version without any downtime. If the fix is carried out only for a specific case, it is possible to keep both versions of the microservice active, redirecting traffic to both versions based on pre-established rules, without necessarily having to delete the old version that is working well for most cases.

Through these traffic management features, Anthos Service Mesh can manage A/B testing³⁷, allowing you to direct a particular percentage of traffic to a new version of a service. This is a good idea when you want to introduce a new version of a service by first testing it using a small percentage of user traffic, and if all goes well, increase the percentage while simultaneously phasing out the old version.

Thanks to these functionalities, it is possible to implement canary deployments or progressive rollout strategies, directly testing the new versions of the services that are released. If the new version of the service experiences no issues with a small percentage of the traffic directed toward it, you could move all traffic to the new version, discarding the old one.

In figure 4.7 a canary deployment with traffic splitting is used to redirect the 5% of traffic to test the new release of the Service A.

In figure 4.8, a canary deployment with traffic splitting is used to redirect the iPhone’s traffic to test the new release of the Service A.

³⁶ See <https://cloud.google.com/service-mesh/docs/supported-features#networking>.

³⁷ See https://en.wikipedia.org/wiki/A/B_testing.

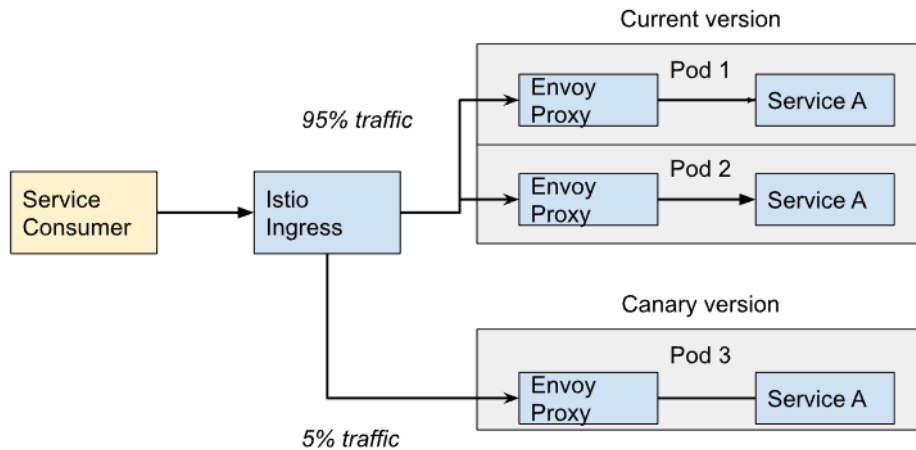


Figure 4.7 Istio canary deployment feature based on traffic.

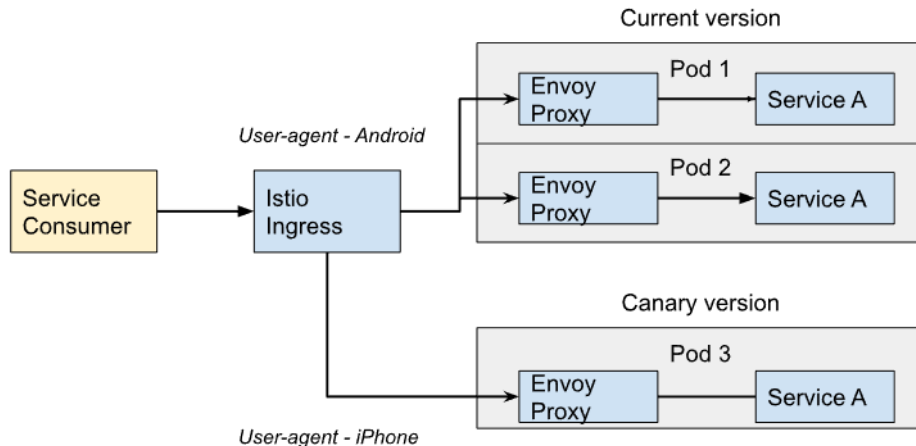


Figure 4.8 Istio canary deployment feature based on user-agent.

The operations departments that manage the production environment take advantage of these features as they release plans for fixes and new versions of microservices. Each of these scenarios can be executed on the fly without disrupting the end users.

4.3.6 Circuit breaking³⁸

Microservices architectures were born to be scalable, agile, and resilient, but it is not always easy to design and implement these architectures to manage high workloads or manage integration with external services with consequent possible downtime or timeout.

³⁸ See <https://istio.io/latest/docs/tasks/traffic-management/circuit-breaking/>.

As already mentioned, the service mesh is independent of the application code and the programming language used; consequently, this facilitates the adoption of functions dedicated to the management of the office.

The functionality that allows you to manage timeouts, failures, and loads on the architecture is called circuit breaking functionality. When you design a microservices architecture, you must always put yourself in the condition of being able to manage faults correctly. These faults may have been caused not only by bugs in the application code, but also from external factors. In the event of a fault or failure to reach the SLA (on availability and/or performance of a given service), this feature automatically allows you to redirect traffic to another microservice or external service to limit down-time or to limit the loss of functionality by the final user.

Let's see an example. In figure 4.9, the service consumer is invoking Istio Ingress to call Service A that is distributed in two pods. Let's assume that Service A inside Pod 1 has a load problem and becomes unreachable: thanks to the circuit breaking feature, Istio will close the connection of the proxy to Service A inside Pod 1, redirecting all traffic to the proxy inside the Pod 2, until Service A in Pod 1 works properly again.

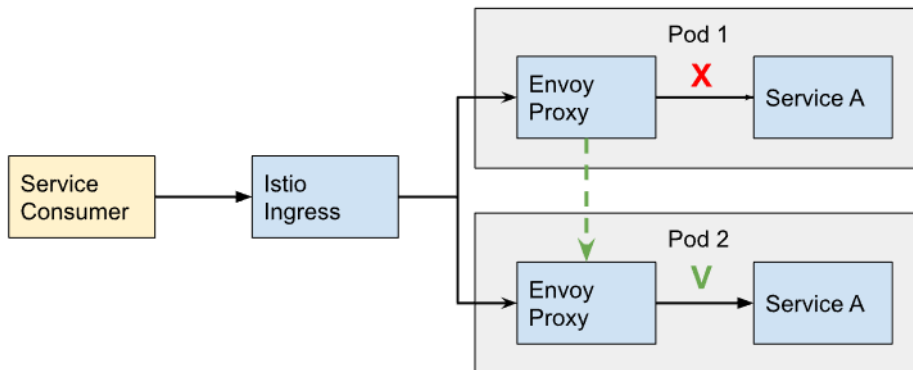


Figure 4.9 Istio circuit-breaking feature example.

4.3.7 Egress control

Usually when it comes to service mesh, a very important feature is the control of the ingress gateway to ensure absolute security to the network (figure 4.10).

However, if we find ourselves in situations for which the regulations (for example PCI³⁹) or the customer's requirements require us to also control the outgoing traffic from the service mesh, then thanks to Istio and thanks to the egress gateway control it is possible to cover the security required.

With Anthos Service Mesh, it is possible to configure the routing of the traffic from the service mesh to the external services (HTTP or HTTPS), using a dedicated egress

³⁹ See https://www.pcisecuritystandards.org/pai_security/.

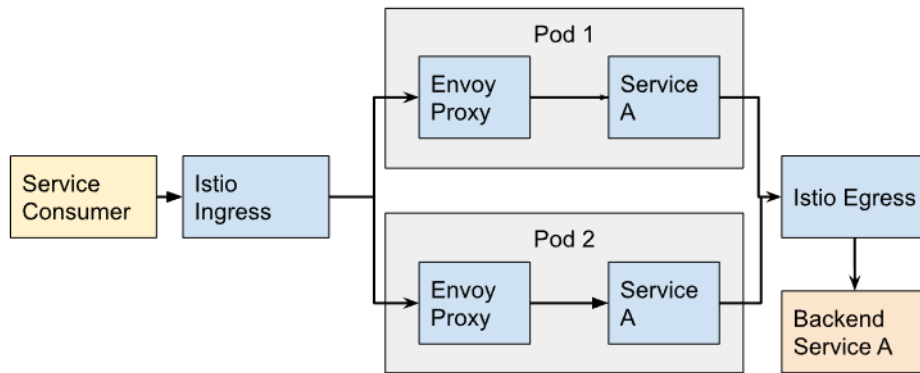


Figure 4.10 Istio egress control example.

gateway or an external HTTPS proxy if necessary; to perform TLS origination (SDS or File mount) from the service mesh for the connection on those external services.

4.4 Policy-driven security⁴⁰

Since the adoption of containers, shared services, and distributed architectures, it has become more difficult to mitigate insider threats and minimize and limit data breaches, ensuring that communications between workloads remain encrypted, authenticated, and authorized. Anthos Service Mesh security features help mitigate these threats, configuring context-sensitive service levels or context-sensitive networks.

Before offerings like Istio, the requirements to secure an application was the responsibility of the application developer, many of the tasks were complex and time consuming and included:

- Encrypting the communications between the application and other services required certificate management and maintenance
- Creating modules that were language specific to authenticate access based on open identity standards like JSON web tokens (JWTs)
- Implementing a complex authorization system to limit the permissions allowed using the assertions on the presented JWT

Instead of a developer taking time to create and manage this security, they can leverage the features of Istio, which addresses each of these without any additional code required.

With Anthos Service Mesh it is possible to adopt a defense in-depth posture consistent with Zero Trust security principles. It allows you to reach this position via declarative criteria and without modifying any application code. The principal security features in ASM are the managed private certificate authority, identity-aware access

⁴⁰ See <https://cloud.google.com/service-mesh/docs/security-overview>.

controls, request claims-aware access control policies, user authentication with identity-aware proxy, and access logging and monitoring.

The first feature, the managed private certificate authority (Mesh CA) includes a Google-managed multi-regional private certificate authority, for issuing certificates for mTLS. The Mesh CA is a highly reliable and scalable service, managed by Google, that is optimized for dynamically scaled workloads on a cloud platform. Mesh CA lets you rely on a single root of trust across Anthos clusters. When using Mesh CA, you can rely on workload identity pools to provide coarse-grained isolation. By default, authentication fails if the client and the server are not in the same workload identity pool.

The next feature, the identity-aware access control (firewall) policies, allows you to configure network security policies that are based on the mTLS identity versus the IP address of the peer. This lets you create policies that are independent of the network location of the workload. Only communications across clusters in the same Google Cloud project are currently supported.

The third feature, request claims-aware access control (firewall) policies, allow you to grant access to services based on request claims in the JWT header of HTTP or gRPC requests. Anthos Service Mesh lets you assert that a JWT is signed by a trusted entity. This means that you can configure policies that allow access from certain clients only if a request claim exists or matches a specified value.

The fourth feature, user authentication with Identity-Aware Proxy, provides authentication of users that are accessing any service exposed on an Anthos Service Mesh ingress gateway by using Identity-Aware Proxy (IAP). IAP can authenticate users that log in from a browser, integrate with custom identity providers, and issue a short-lived JWT token or an RCToken that can then be used to grant access at the Ingress Gateway or a downstream service (by using a sidecar).

The final features, access logging and monitoring, ensure that access logs and metrics are available in Google Cloud's operations suite, and provide an integrated dashboard to understand access patterns for a service or workload based on this data. You can also choose to configure a private destination. Anthos Service Mesh lets you reduce noise in access logs by only logging successful accesses once in a configurable time window. Requests that are denied by a security policy or result in an error are always logged. This lets you significantly reduce the costs associated with ingesting, storing, and processing logs, without the loss of key security signals.

4.5 Conclusion

In this chapter, we have seen what a service mesh is, what are the advantages of implementing it, and how Anthos Service Mesh exploits the potential of Istio to manage the entire service mesh.

Thanks to Anthos Service Mesh, the developers are more agile in implementing microservices architectures and don't need to worry about implementing monitoring probes within the application code, while taking advantage of sidecar proxy and proxyless approaches.

The operations structures are able to monitor everything that happens within the service mesh in real time, guaranteeing the required service levels. The traffic splitting and rolling release features allow you to efficiently release new versions of the services, ensuring that everything works correctly. Thanks to the security features, the service mesh is protected from risks that can come from outside or inside the network, implementing effective authentication and authorization policies.

4.5.1 Examples and case studies

Using the knowledge from the chapter, address each of the requirements in the case study found below.

EVERMORE INDUSTRIES

You have been asked by Evermore Industries to enable ASM on a GKE cluster running in GCP. The cluster will be used for initial service mesh testing and should be installed with all features to allow the developers to test any feature. They have also asked you to deploy an Online Boutique application to prove that the service mesh is up and running as expected.

Because they are new to Istio and the advantages of using a service mesh, they do not have any special requirements outside of deploying ASM and the Boutique demo application. The only additional requirement is to provide proof that the Boutique application is running in the mesh as expected from the GCP console.

The next section contains the solution to address Evermore's requirements. You can follow along with the solution, or if you are comfortable, configure your cluster to address the requirements and use the solution to verify your results.

EVERMORE INDUSTRIES SOLUTION—INSTALLING ASM

To install ASM, you can download the ASM installation script to deploy ASM with all components installed. You can follow the steps below to install ASM with all components on your GKE cluster running in GCP.

- 1 Download the ASM installation script:

```
curl https://storage.googleapis.com/csm-artifacts/asm/install_asm_1.9 >
install_asm
```

- 2 Make the installer executable:

```
chmod +x install_asm
```

- 3 You will need the following information from your project and GKE cluster to execute the installation script: Project ID, GKE cluster name, and the GKE cluster location.
- 4 Execute the installation script with the information from your cluster to install ASM:

```
./install_asm --project_id gke-test1-123456 --cluster_name gke-dev-001
--cluster_location us-central1-c --mode install --output_dir
./asm-downloads --enable_all
```

- 5 The installation will take a few minutes. Once the installation is complete, you will see a message similar to the one shown here:

```
install_asm: Successfully installed ASM.
```

- 6 Verify that the istio-system namespace has healthy pods that have been started successfully:

```
kubectl get pods -n istio-system
```

- 7 This should show you that there are four running pods, (2) istio-ingressgateway pods and (2) istiod pods, an example output is shown Here:

NAME	READY	STATUS
istio-ingressgateway-68fb877774-9tm8j	1/1	Running
istio-ingressgateway-68fb877774-qf5dp	1/1	Running
istiod-asm-192-1-78fb6c7f98-n4xpp	1/1	Running
istiod-asm-192-1-78fb6c7f98-sgttk	1/1	Running

- 8 Now that Istio has been deployed, you need to create a namespace and enable Istio for sidecar injection.

EVERMORE INDUSTRIES SOLUTION—ENABLING SIDECAR INJECTION

To enable the namespace for sidecar injection, follow the steps:

- 1 Create a namespace that will be used to deploy the Boutique application. In our example, we will use a namespace called demo:

```
kubectl create namespace demo
```

- 2 Next, we need to label the namespace with the correct label to enable sidecar injection. Starting with Istio 1.7, the label used to enable sidecar injection changed from a generic istio-injection, to using the value of the control plane version.

To find the control plane version we will use in the label, retrieve the labels in the istio-system namespace:

```
kubectl -n istio-system get pods -l app=istiod --show-labels
```

- 3 This will return the labels of the istiod pods, which will contain the value we need. The output will look similar to the following example. (Note: The label value we will need has been highlighted in bold):

NAME	READY	STATUS	RESTARTS	AGE
istiod-asm-192-1-78fb6c7f98-n4xpp	1/1	Running	0	44m
app=istiod,install.operator.istio.io/owning-resource=unknown, istio.io/rev=asm-192-1 ,istio=istiod,operator.istio.io/component=Pilot,pod-template-hash=78fb6c7f98,sidecar.istio.io/inject=false				
istiod-asm-192-1-78fb6c7f98-sgttk	1/1	Running	0	44m
app=istiod,install.operator.istio.io/owning-resource=unknown, istio.io/rev=asm-192-				

```
1,istio=istiod,operator.istio.io/component=Pilot,pod-template-hash=78fb6c7f98,sidecar.istio.io/inject=false
```

- 4 Using the istio/io value, label the demo namespace to enable sidecar injection:

```
kubectl label namespace demo istio.io/rev=asm-192-1
```

EVERMORE INDUSTRIES SOLUTION—INSTALLING THE BOUTIQUE APPLICATION

Now that ASM has been installed and we have created a new namespace with the correct label to enable sidecar injection, we can deploy the Boutique application. Follow the steps to deploy the Google Boutique demo:

- 1 Download the Boutique demo application from the GIT repository. The following command will download the GIT repo into a directory called online-boutique:

```
kpt pkg get https://github.com/GoogleCloudPlatform/microservices-demo.git/release online-boutique
```

- 2 Deploy the application using the files in the online-boutique directory:

```
kubectl apply -n demo -f online-boutique
```

- 3 This will install a number of deployments and services into the demo namespace. It will take a few minutes for the pods to start up. You can watch the namespace or list the pods in the namespace to verify that each pod enters a running state, and that each pod shows two containers. (Remember from the chapter that each pod will have a sidecar injected for the Istio proxy.)

Once all pods are running, the demo namespace output should look similar to the following output:

NAME	READY	STATUS
adservice-6b74979749-2qd77	2/2	Running
cartservice-6fc79c6d86-tvncv	2/2	Running
checkoutservice-7c95787547-8dmzw	2/2	Running
currencyservice-67674dbdf7-hkw78	2/2	Running
emailservice-799966ff9f-qcb6s	2/2	Running
frontend-597d957cdf-dmdwr	2/2	Running
loadgenerator-88f7dbff5-cn78t	2/2	Running
paymentservice-5bdd645d9f-4w9f9	2/2	Running
productcatalogservice-7ffbf4fbf5-j98sq	2/2	Running
recommendationservice-599dfdc445-gpmww	2/2	Running
redis-cart-57bd646894-tdxwb	2/2	Running
shippingservice-5f4d856dc-cwtcl	2/2	Running

- 4 As part of the deployment, a load-balancer service was created that allows connectivity to the Boutique application from the Internet. To find the IP address that has been assigned, get the service called frontend-external in the demo namespace:

```
kubectl get services frontend-external -n demo
```

- 5 This will output the service details which will contain the external address that you can use to verify the application is working.

- 6 Use the address from the output in step 3 to connect to the application from a web browser. Once you connect, you should see the main Boutique page:

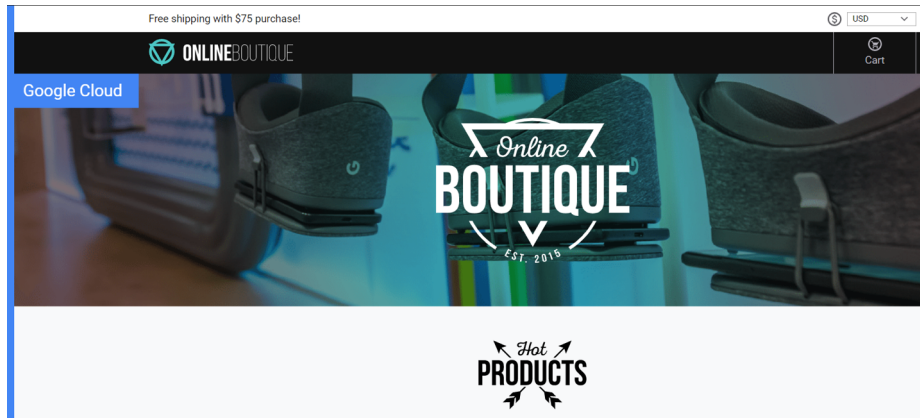


Figure 4.11 Online Boutique web application

EVERMORE INDUSTRIES SOLUTION—OBSERVING SERVICES USING THE GCP CONSOLE

The final requirement from Evermore is to prove that the Boutique application is running in the mesh, using the GCP console. To prove this, you can use the Anthos > Service Mesh from the GCP console.

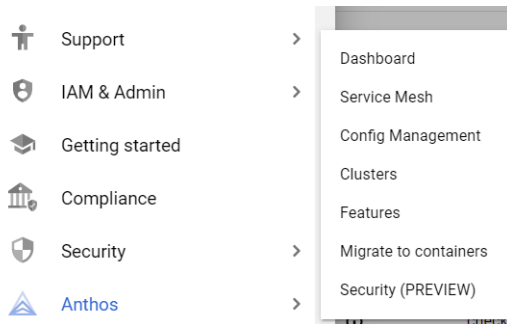


Figure 4.12 Google Cloud Console—Anthos menu.

This will provide a list of all services in the service mesh. The default table view in figure 4.13 will show each service and metrics including requests, latency, and failures.

You can change the view from the table view to a topology view by clicking the topology button in the upper right-hand corner of the console. This will provide a graphical topology layout of the mesh services (figure 4.14).

Both of these views will only show services in the service mesh. Because we see the expected services for the Boutique application, this proves that the deployment is successfully working inside of the mesh. This completes the exercise. In this exercise you deployed ASM in a GKE cluster, created a new namespace that was enabled for sidecar injection, and deployed a test application into the service mesh.

Services ?

Filter

Select a filter option

Status ?	Name ↑	Namespace	Types	Clusters	Requests/sec (avg)
i	adservice	demo	K8s	gke-dev-001	0.9
i	cartservice	demo	K8s	gke-dev-001	1.5

Figure 4.13 Anthos Service Mesh - list of services.

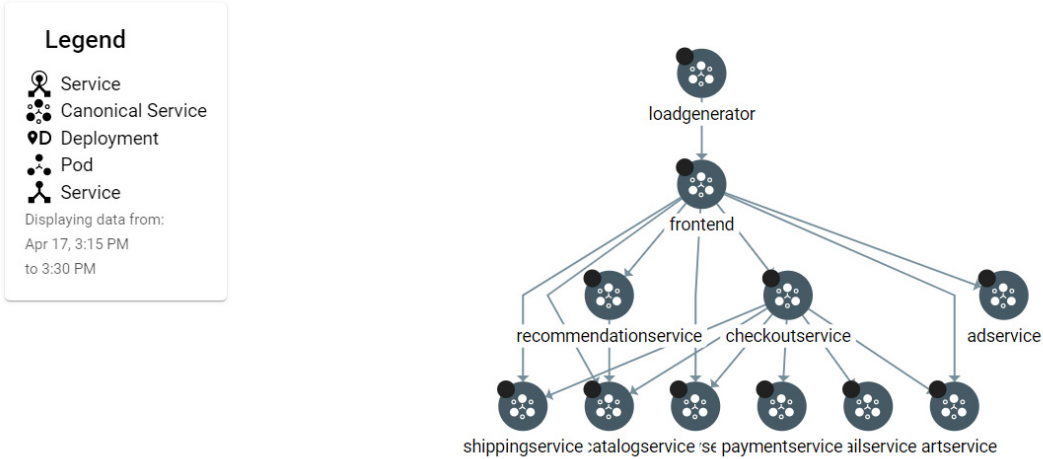


Figure 4.14 Anthos Service Mesh - topology view.

Summary

In this chapter you got to learn that a service mesh based on Anthos Service Mesh helps organizations run microservices at scale by providing:

- A more flexible release process with canary deployments and A/B testing
- Availability and resilience with circuit breakers, traffic splitting and fault injection
- In-depth visibility, monitoring, security, and control of the entire service mesh and between multiple environments

5 *Operations management in Anthos*

by Jason Quek

This chapter covers

- Using the unified Google cloud interface to manage Kubernetes clusters
- Managing Anthos clusters
- Logging and monitoring
- Understanding Anthos deployment patterns

Operations is the act of ensuring your clusters are functioning, active, secure, and able to serve the application to the users. To that end, there is one prevailing school of thought that has gained adoption and momentum in the cloud era, an operations practice known as DevOps.

The simplest definition of DevOps is the combination of developers and IT operations. DevOps aims to address two major points, the first is to enable continuous delivery through automated testing, frequent releases and management of the entire infrastructure as code. The second, an often-overlooked part of DevOps, is IT operations, which includes tasks like logging, monitoring, and using those indicators to scale and manage the system.

Google developed an approach before the development of DevOps called Site Reliability Engineering (SRE). This approach addresses the requirements that enable DevOps, which automates and codifies all tasks in operating the infrastructure to enhance reliability in the systems, and if something goes wrong, it can be repaired automatically through code. An SRE team is responsible for not only keeping the environment stable, but it should also be the team that is responsible for new operational features and improvements to the infrastructure.

Both schools of thought have different responsibilities assigned to different teams; however, they have the same goal, which is to be able to implement changes rapidly and efficiently, automate where possible and continuously monitor and enhance the system. This commonality underlies the “desire” from the engineering and operations team to break silos and take common responsibility for the system.

Either approach will deliver many of the same advantages, but they solve problems in different ways. For example, in a DevOps approach, there may be a dedicated operations team that takes care of the operations management aspect of the infrastructure, handing issues off to another development team to resolve. This differs from the SRE approach, where operations are driven by the development team and approached from a software engineering point of view, allowing for a single SRE team to address issues within their own team.

Anthos provides a path to build a strong DevOps or SRE culture by usage of the tools provided in the framework. This chapter will show product owners, developers and infrastructure teams that through using Anthos, they are able to build a DevOps/SRE culture that will help to reduce silos in their company, build reliable systems, and enhance development efficiency.

In the next section, we will explain the tools that Anthos includes, starting with the Unified User Interface through Google Cloud Console, then centralized logging and monitoring, and environs, which are all key concepts that will provide the building blocks to enable an operations practice.

5.1 Unified user interface from Google Cloud Console

With everything-as-code these days, there is a certain pride that a software engineer takes in doing everything from the command line, or, as code. However, when a production issue occurs affecting real-life services, an intuitive and assistive user interface can help an engineer identify the issue quickly.

This is where Google’s unified user interface comes in handy, as seen in figure 5.1.

These tools often allow you to view multiple items, like Kubernetes clusters, in a single view. Having this view available to an administrator gives an oversight of all the resources available, without having to log into three separate clusters as seen from figure 5.1. This view also shows where they are located, which are their providers as well and actions required to manage the clusters.

Kubernetes clusters CREATE CLUSTER DEPLOY REGISTER CLUSTER REFRESH DELETE SHOW INFO PANEL

Filter by label or name

Name	Location	Cluster type	Cluster size	Total cores	Total memory	Notifications	Labels	
<input type="checkbox"/> finland-gke-cluster	hamina	GKE from avalonx-dev	4	10 CPU	39.29 GB		location: hamina provider: gcp	<input type="button" value="Log out"/> <input type="button" value="Delete"/>
<input type="checkbox"/> gke-bf08bb38-cluster	paris	Anthos	3	6 CPU	12.22 GB		location: paris provider: aws	<input type="button" value="Log out"/> <input type="button" value="Delete"/>
<input type="checkbox"/> stockholm-user-cluster-1	stockholm	Anthos	3	12 CPU	25.08 GB	Upgrade available	location: stockholm provider: onprem	<input type="button" value="Log out"/> <input type="button" value="Delete"/>

Figure 5.1 Multiple clusters registered to Google Cloud Console.

To access this view requires that the user is already logged into Google Cloud console, which is secured by Google Cloud Identity, providing an additional layer of security to build defense in-depth against malicious actors.

Having access to this type of view fulfills one of the DevOps principles, leveraging tooling to provide observability into the system. For more details, please refer to chapter 3, “Anthos, the one single glass-of-pane UX”.

To have a single pane of glass view, you will need to register your clusters with your GCP project. In the next section, we will cover how to register a cluster that is running Anthos on any of the major Cloud Service Providers or on-prem.

5.1.1 Registering clusters to Google Cloud Console

The component responsible for connecting clusters to Google Cloud Console is called Connect, which is often deployed as one of the last steps after a cluster is created. If an Anthos on GKE/AWS/Azure/on-prem is deployed, the Connect agent is automatically deployed at cluster creation time. However, if the cluster is not deployed by Anthos, such as EKS, AKS, OpenShift, Rancher clusters, the agent will have to be deployed separately, because Anthos is not involved in the installation process. This will be covered later in this chapter on Anthos attached clusters.

Because Anthos is built with the best practices from Kubernetes, the Connect Agent is represented as a Kubernetes Deployment, with the image provided by Google as part of the Anthos framework. This means the agent can also be seen from the Google Cloud Console, and can be managed like any other Kubernetes object, as shown in figure 5.2.

The Connect agent acts as a conduit for Google Cloud Console to issue commands to the clusters where it has been deployed on and to report vCPU usage for licensing. This brings up one important point, which is that the clusters need to be able to reach Google Cloud APIs (egress), however, the clusters do not need to be reachable by Google Cloud APIs (ingress).

So, how does Google Cloud Console issue Kubernetes API commands, such as listing pods to display on the Google Cloud Console? The answer is through the Connect Agent, which establishes a persistent TLS 1.2 connection to GCP to wait for requests, which eliminates the need of having an inbound firewall rule to the user cluster.

For those who are unfamiliar with TLS (Transport Layer Security), it is a cryptographic protocol designed to provide privacy and data integrity between the sender

✓ gke-connect-agent-20200515-02-00

Overview Details Revision history Events YAML

Cluster	finland-gke-cluster
Namespace	gke-connect
Labels	app : gke-connect-agent hub.gke.io/project : anthos-sandbox-256114 version : 20200515-02-00
Replicas	1 updated, 1 ready, 1 available, 0 unavailable
Pod specification	Revision 1, containers: gke-connect-agent-20200515-02-00, volumes: creds-gcp, http-proxy

Active revisions

Revision	Name	Status	Summary	Created on	Pods running/Pods total
1	gke-connect-agent-20200515-02-00-5865ddc9f5	✓ OK	gke-connect-agent-20200515-02-00: gcr.io/gkeconnect/gkeconnect-gce:20200515-02-00	25 May 2020, 01:32:41	1/1

Managed pods

Revision	Name	Status	Restarts	Created on
1	gke-connect-agent-20200515-02-00-5865ddc9f5-jpgfr	✓ Running	0	28 May 2020, 21:33:34

Figure 5.2 Connect agent deployed on GKE cluster.

and receiver. It utilizes symmetric encryption based on a shared secret to ensure that the message is private. Messages are signed with a public key to ensure its authenticity, and also include a message integrity check to ensure that messages are complete. In short, the communication channel to the Connect agent over the internet is as secure as internet bank transfers. The full communication flow can be seen in chapter 3, “Anthos, one single glass-of-pane UX”.

One important point to note is that outbound TLS encrypted connection over the internet is used for Anthos deployments to communicate with Google Cloud as shown in figure 5.3. This simplifies things, as no inbound firewalls rules have to be added to Anthos deployments, only outbound traffic toward Google Cloud, without any virtual private networks (VPN) requiring set up.

One great thing about Kubernetes is the standardization, which means this agent will be able to issue Kubernetes API commands on clusters created by Google or *any* provider that provides a compliant Kubernetes distribution as defined by the Cloud Native Com.

A common issue in large enterprises is that IT security would want to know what is the Connect agent sending to Google Cloud APIs, and this is a tricky issue to overcome due to the perceived worry that if Google shares the keys to decrypt the traffic, it is effectively overriding the security put into place. More details of what information is actually sent from the Connect agent to Google Cloud can be found in this white paper by Google (<https://cloud.google.com/files/security-features-for-connect-for-anthos.pdf>). Google has also stated unequivocally that no customer data is sent via the Connect Agent, and that it is only to provide functionality to communicate with the Kubernetes API and also provide licensing metrics for Anthos billing.

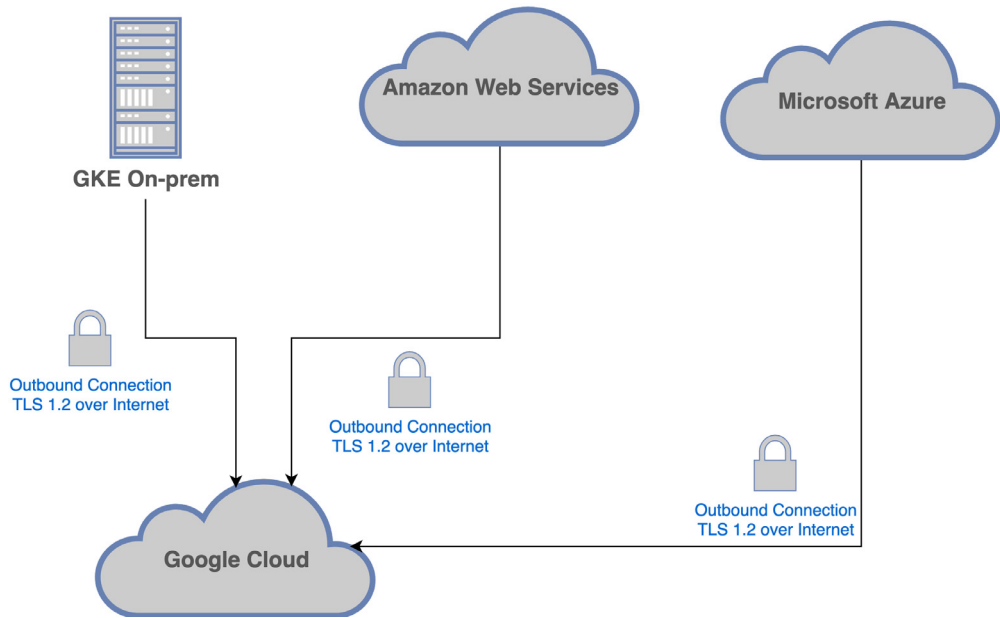


Figure 5.3 Outbound connection to Google Cloud from Anthos deployments.

5.1.2 Authentication

Authentication to Google Kubernetes Engine should utilize Identity and Access Management roles to govern access to the GKE clusters. The following section pertains to GKE On-prem, GKE on AWS, GKE on Azure, and Anthos-attached clusters.

To access Anthos clusters, users with access to the project will always have to provide either a Kubernetes Service Account (KSA) token, basic authentication or authenticate against an identity provider configured for the cluster.

Using a Kubernetes Service Account token would be the easiest to set up, but requires token rotation, and a secure way to distribute tokens regularly to the users who need access to the clusters. Using basic authentication would be the least secure due to having password management requirements, but it is still supported as an authentication method if an identity provider is not available. If you need to use basic authentication, one tip would be to implement a password rotation strategy in the event of password leaks.

The recommended practice would be to set up OpenID Connect (OIDC) with Google Cloud Identity so that users can leverage their existing security set up to manage access to their clusters as well. As of September 2020, OIDC is supported on GKE on-prem clusters from the command line (not from the console). So, a solid KSA token rotation and distribution strategy is highly recommended, which can be as simple as utilizing Google Secret Manager, where permissions to retrieve the token can be controlled via IAM permissions, and the token can be updated every seven days using Cloud Scheduler.

Once OIDC with Google Cloud Identity has been set up, users are able to authenticate to the user clusters using the gcloud CLI or from the Google Cloud Console (figure 5.4).

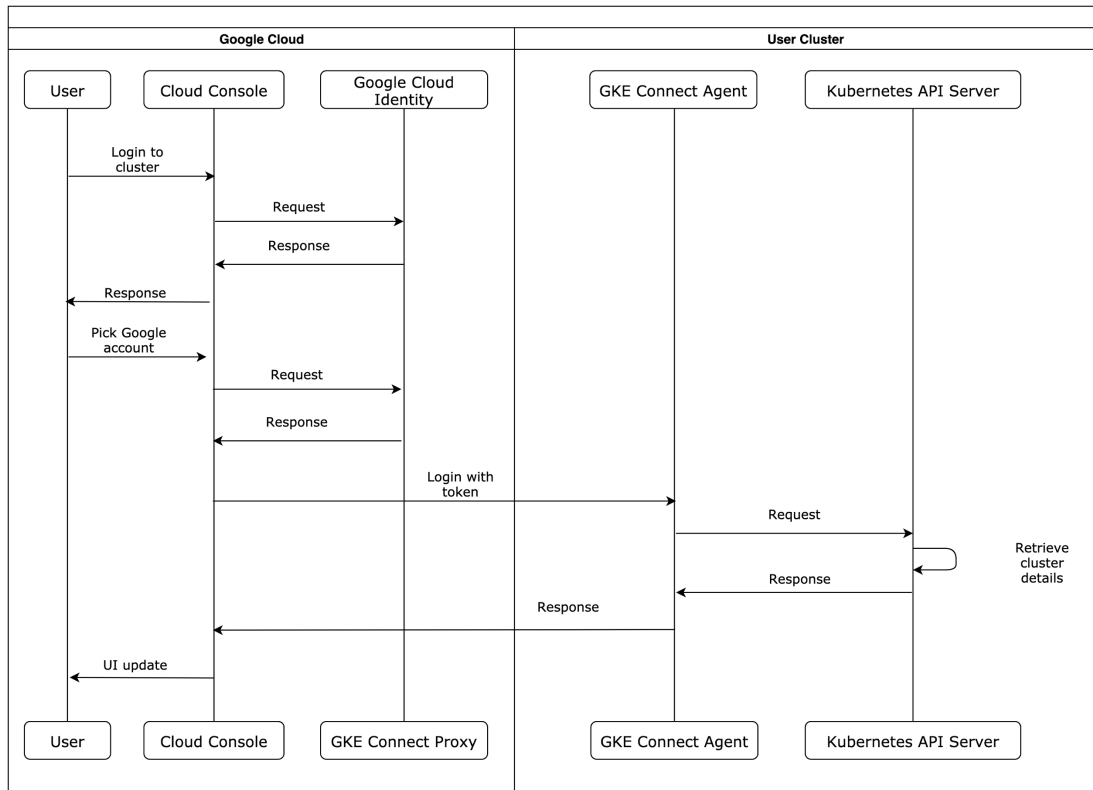


Figure 5.4 Authentication Flow for OIDC with Google Cloud Identity.

In figure 5.4, we show the identity flow using OIDC with Google Cloud Identity. With Anthos Identity Service, other providers who follow the OIDC, and Lightweight Directory Access Protocol (LDAP) protocols can provide identity. This allows a seamless user administration with technologies such as MS Active Directory or an LDAP server, and follows the principle of having one single source of truth of identity.

5.1.3 Cluster management

After registering the clusters and authenticating, users will be able to see pods, services, ConfigMaps and persistent volumes that are normally available from GKE native clusters. In this section, the cluster management options available via the Google Cloud Console will be covered. However, to build a good SRE practice, cluster man-

agement should be automated and scripted, but it is nice to be able to modify these from a user-friendly interface.

Administrators who have experience in Google Kubernetes Engine on GCP know how easy it is to connect to the cluster from Google Cloud Console. They just navigate to the cluster list as seen in figure 5.5 and click on the Connect button.

Name ^	Location	Cluster size	Total cores	Total memory	Notifications	Labels
✓ anthos-primary-cluster	europe-north1-a	4	10 vCPUs	37.50 GB	⚠ Low resource requests	mesh_id : proj-710282601588 Connect

Figure 5.5 Cluster list in GCP Console.

Once the Connect button is clicked, a popup window, as shown in figure 5.6, will provide the command to run in a Google Cloud Shell to connect to the selected cluster.

Connect to the cluster

You can connect to your cluster via command-line or using a dashboard.

Command-line access

Configure `kubectl` command-line access by running the following command:

```
$ gcloud container clusters get-credentials primary-cluster --region europe-west1 --project msp-cc-prod
```

[Run in Cloud Shell](#)

Cloud Console dashboard

You can view the workloads running in your cluster in the Cloud Console [Workloads dashboard](#).

[Open Workloads dashboard](#)

OK

Figure 5.6 One of the best features in GKE, generating `kubectl` credentials via `gcloud`.

For on-prem and other cloud clusters, the Connect Gateway functionality further in the chapter allows operations administrators to still manage their clusters remotely but through a different command.

Google Cloud Console provides a user-friendly interface to edit and apply yaml deployments as shown in figure 5.7. Through this interface, administrators are able to modify Kubernetes configurations without having to go through the `kubectl` command line, which can save some time in emergency situations. These actions on Google Cloud Console translate to Kubernetes API calls, or `kubectl` edit commands, and are

issued via the Connect agent to the Anthos clusters. Of course, this should only be used in triage or dev situations, not necessarily in production, but it shows the future possibilities of opening up access to the Connect Agent from the local command line.

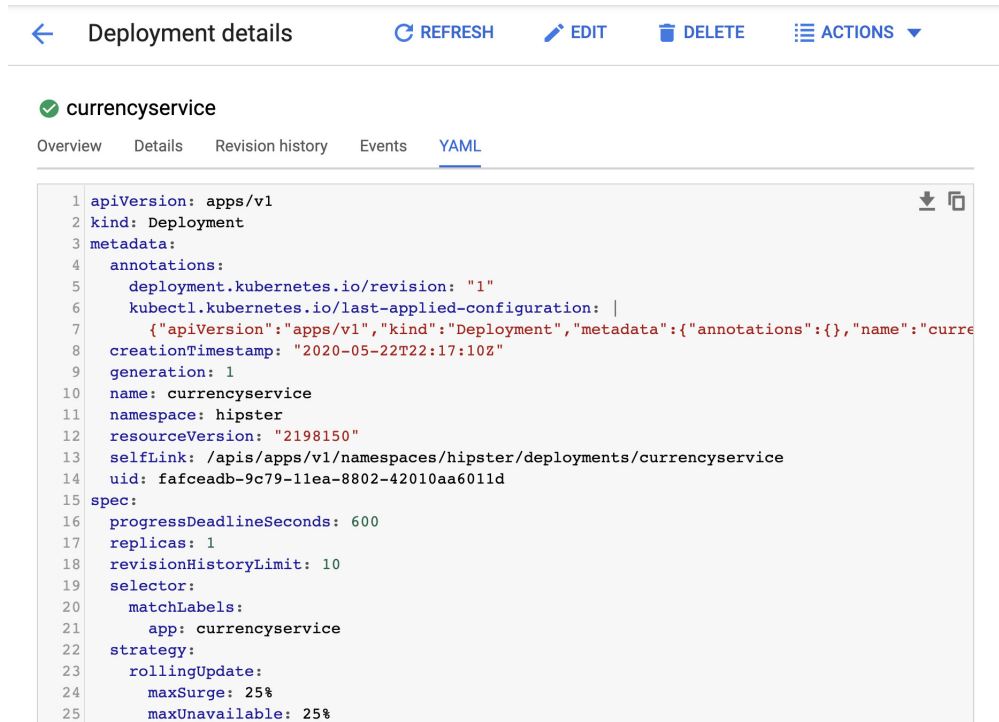


Figure 5.7 Editing a YAML definition from the Google Cloud Console.

Google Cloud Console also provides useful information about the underlying Docker, kubelet, and memory pressure for the nodes, as seen from figure 5.8. This provides administrators a quick root cause analysis if there is a fault with one of the nodes and they are able to cordon and drain the node.

When listing the workloads in Google Cloud Console, a user can see deployments across all clusters and filter them by clusters they are in. This gives the user an overview of what services are running across all clusters and indicates if there are any issues with any services and scaling limits. One of the most common issues is the pods unable to be provisioned due to lack of CPU or memory, and this is clearly visible as a bright red error message in the console as seen from figure 5.9.

Viewing a cluster interactively with tools is beneficial for real-time views of object states, node statuses, and more. While this can be helpful in the right scenario (e.g., when diagnosing a previously unknown issue which impacts production where an user-friendly interface reduces the need to remember commands in a high stress situ-

← Node details REFRESH EDIT CORDON			
Huge pages 1Gi	0 B	0 B	0 B
Huge pages 2Mi	0 B	0 B	0 B
Memory	7.84 GB	5.92 GB	774.85 MB
Pods	110	110	0
Storage	0 B	0 B	0 B

Conditions			
Condition ^	Message	Last heartbeat	Last transition
✔ CorruptDockerOverlay2: False	docker overlay2 is functioning properly	14 Jul 2020, 20:10:39	28 May 2020, 22:36:26
✔ DiskPressure: False	kubelet has no disk pressure	14 Jul 2020, 20:10:57	28 May 2020, 22:36:22
✔ FrequentContainerdRestart: False	containerd is functioning properly	14 Jul 2020, 20:10:39	28 May 2020, 22:36:26
✔ FrequentDockerRestart: False	docker is functioning properly	14 Jul 2020, 20:10:39	28 May 2020, 22:36:26
✔ FrequentKubeletRestart: False	kubelet is functioning properly	14 Jul 2020, 20:10:39	28 May 2020, 22:36:26
✔ FrequentUnregisterNetDevice: False	node is functioning properly	14 Jul 2020, 20:10:39	28 May 2020, 22:36:26
✔ KernelDeadlock: False	kernel has no deadlock	14 Jul 2020, 20:10:39	28 May 2020, 22:36:26
✔ MemoryPressure: False	kubelet has sufficient memory available	14 Jul 2020, 20:10:57	28 May 2020, 22:36:22
✔ NetworkUnavailable: False	RouteController created a route	28 May 2020, 22:36:33	28 May 2020, 22:36:33
✔ PIDPressure: False	kubelet has sufficient PID available	14 Jul 2020, 20:10:57	28 May 2020, 22:36:22
✔ ReadOnlyFilesystem: False	Filesystem is not read-only	14 Jul 2020, 20:10:39	28 May 2020, 22:36:26
✔ Ready: True	kubelet is posting ready status. AppArmor enabled	14 Jul 2020, 20:10:57	28 May 2020, 22:36:22

Figure 5.8 Node information from Google Cloud Console.


 Pod errors: Unscheduleable SHOW DETAILS

Figure 5.9 Unscheduleable pods example.

ation), you will find yourself looking at logs and creating monitoring events more often than real-time views. In the next section, we will detail the logging and monitoring features that Anthos includes.

5.2 *Logging and monitoring*

In Kubernetes, there are different kinds of logs which are useful for administrators to investigate when managing the cluster. These are the system logs which Kubernetes system services such as kube-apiserver, etcd, kube-scheduler and kube-controller-manager log to. Clusters also have application logs, which contain log details for all the workloads running on the Kubernetes cluster. These logs can be accessed through the Connect Agent, which would communicate with the Kubernetes API and essentially issue a `kubectl logs` command.

These logs are not stored in the cloud, but retrieved on demand from the Kubernetes API, which translates to an increased retrieval latency but is at times necessary in case of IT security requests. These are both for application logs and for system logs as seen in figure 5.10.

✓ **adservice-84449b8756-n54d5**

DETAILS EVENTS **LOGS** YAML

☰ Filter logs ?

Container	Timestamp ↓	Message
istio-proxy	14 Jul 2020, 20:42:23	[2020-07-14T18:38:12.810Z] "-" 0 - "-" 1428 3415 240859 - "-" "" "" "173.194.222.95:443" outbound 443 *.googleapis.com 10.10.4.11:38954 173.194.222.95:443 10.10.4.11:38952 cloudtrace.googleapis.com -
istio-proxy	14 Jul 2020, 20:40:08	[Envoy (Epoch 1)] [2020-07-14 18:40:08.150][38][warning][filter] [src/istio/mixerclient/report_batch.cc:110] Mixer Report failed with: UNAVAILABLE:upstream connect error or disconnect/reset before headers. reset reason: connection failure
istio-proxy	14 Jul 2020, 20:29:37	[Envoy (Epoch 1)] [2020-07-14 18:29:37.449][31][warning][config] [bazel-out/k8-opt/bin/external/envoy/source/common/config/_virtual_includes/grpc_stream_lib/common/config/grpc_stream.h:91] gRPC config stream closed: 13,
istio-proxy	14 Jul 2020, 20:13:23	[2020-07-14T18:13:20.731Z] "GET /computeMetadata/v1/instance/service-accounts/default/token HTTP/1.1" 200 - "-" 0 210 26 25 "-" "Google-HTTP-Java-Client/1.24.1 (gzip)" "f8ef3aea-0512-43e3-9c79-5148e0802f7c" "169.254.169.254" "169.254.169.254:80" PassthroughCluster - 169.254.169.254:80 10.10.4.11:55300 - -
istio-proxy	14 Jul 2020, 20:09:07	[Envoy (Epoch 1)] [2020-07-14 18:09:07.886][38][warning][filter] [src/istio/mixerclient/report_batch.cc:110] Mixer Report failed with: UNAVAILABLE:upstream connect error or disconnect/reset before headers. reset reason: connection failure
istio-proxy	14 Jul 2020, 20:06:03	[2020-07-14T17:34:10.309Z] "-" 0 - "-" 948462 7830 1904930 - "-" "" "" "173.194.73.95:443" outbound 443 *.googleapis.com 10.10.4.11:49044 173.194.73.95:443 10.10.4.11:49042 monitoring.googleapis.com -

Figure 5.10 Container logs.

Logs are primarily about errors, warnings that output to the standard output stream during the execution of any Kubernetes pod. These logs are written to the node itself and if the Google Cloud's operations suite (formerly Stackdriver) agent is enabled on the GKE cluster, the logs are aggregated and forwarded to the Cloud Logging API and written into the cloud.

Metrics are observations about a service, such as memory consumption, requests per second. These observations are saved in a historical trend which can be used to scale services, or to identify possible issues in implementation. Given that each service can potentially have tens of observations occurring every second or minute depending on the business requirements, managing this data in a usable manner is non-trivial and a few solutions have been proposed in the following paragraph involving Google's Cloud Logging and Monitoring service. Partner technology such as Elastic Stack, Prometheus, Grafana or Splunk can also be used to make sense of the metrics (e.g., <https://cloud.google.com/architecture/logging-anthos-with-splunk-connect> and <https://cloud.google.com/architecture/partners/monitoring-gke-on-prem-with-the-elastic-stack>).

5.2.1 Logging and monitoring GKE on-prem

Administrators can choose between a few different options for observability when installing GKE on-prem clusters.

The first option is to use Google's native Cloud Logging and Cloud Monitoring solutions. Cloud Logging and Monitoring handles infrastructure and cloud services,

as well as Kubernetes Logging and Monitoring. All logged data can be displayed in hierarchical levels according to the Kubernetes object types. By default, GKE logging only collects logs and metrics from the kube-system, gke-system, gke-connect, istio-system, and config-management system namespaces, which are used to track cluster health and are sent to Cloud Logging and Monitoring in Google Cloud. This is a fully managed service and includes dashboarding and alerting capabilities to be able to build a useful monitoring control panel. Cloud Logging and Monitoring is often used to monitor Google Cloud resources and issue alerts on certain logged events, and it also serves as a single pane of glass for monitoring service health.

This is the recommended option in the event the organization is open to using and learning a new logging and monitoring stack and wants a low cost and fully managed option.

There may be certain organizations that want to disable Cloud Logging and Monitoring due to internal decisions. While this can be disabled, the Google Support SLA will be voided, and Google support will only be able to help as a best-effort when resolving GKE On-prem operation issues.

The second option is to use Prometheus, AlertManager, and Grafana, a popular open-source collection of projects, used to collect application and system level logs, and provide alerting and dashboarding capabilities. Prometheus and Grafana are deployed as Kubernetes Monitoring Add-on workloads, and as such, will benefit from the scalability and reliability of running on Kubernetes. Support from Google is limited when using this solution, limited to basic operations and basic installation/configuration. For more information on Prometheus and AlertManager, please visit <https://prometheus.io>, and for Grafana please visit <https://grafana.com>.

This setup can be used across any Kubernetes setup, and there are also many Grafana dashboards prebuilt which can be used to monitor Kubernetes cluster health. One downside is that administrators will now have to manage Prometheus and ensure its health and manage its storage of historical metrics, as it is running as any other application workload. Other tools such as Thanos can be used to query, aggregate, downsample, and manage multiple Prometheus sources, as well as store historical metrics in object storage such as Google Cloud Storage or any S3 compatible object stores.

For more information on Thanos, please visit <https://thanos.io/>.

This option is easy for organizations which have built logging and monitoring using open-source technologies and have deployed this stack before. This also improves portability and reduces vendor lock-in due to the open-source technologies used.

The third option is to use validated solutions such as Elastic Stack, Splunk, or Datadog, to consume logs and metrics from Anthos clusters and make them available to the operations team.

This is an attractive option if these current logging methods are already in place and rely on partners to manage the logging and monitoring systems uptime. Organizations who choose this option often already purchased this stack and are used for their overall organizations with many heterogeneous systems.

A fourth option is also a tiered telemetry approach, which is recommended for organizations embarking on a hybrid journey with Anthos.

The first reason is that platform and system data from Anthos Clusters are always tightly coupled with Cloud Monitoring and Logging so administrators would have to learn Cloud Monitoring and Logging to get the most up-to-date logs and metrics anyways as well as it does not have any additional costs and is part of the Anthos suite.

The second reason is that building a hybrid environment often requires migrating applications to the hybrid environment, with developers who are used to working with these partner solutions and have built debugging and operating models around that stack, making it a supported option that reduces the operational friction to moving workloads to a hybrid environment.

The third reason is to build the ability to balance points of failure among different providers and having a backup option.

5.3 Service mesh logging

Anthos Service Mesh is an optional component but included in the Anthos platform which is explained in depth in chapter 4. It is an extended and supported version of open-source Istio, included with Anthos and supported by Google. Part of what Google extended is to upload telemetry data from sidecar proxies injected with your pods directly to Cloud Monitoring API and Cloud Logging API on Google Cloud. These metrics are then used to visualize preconfigured dashboards in the Google Cloud Console. For more details, please refer to chapter 3, “Anthos, the one single glass-of-pane UX”.

Storing these metrics on Google Cloud also allows the user to have historical information on latency, errors, and traffic between their microservices, so that post-mortem can be conducted on any issues.

These metrics can be further used to drive your Service Level Indicators, pod scaling strategy, and identify services for optimization.

5.4 Using Service Level Indicators and Agreements

Anthos Service Mesh Service Level Indicator (SLI), Service Level Objectives (SLO) and Service Level Agreement (SLA) are features which can be used to build an SRE practice where Anthos is deployed. These concepts were introduced in chapter 4; consider these concepts when designing operations management procedures in Anthos.

With Service Level Indicators, there are two indicators which are available to measure service levels, latency, and availability. Latency is how long the service takes to respond, while availability is how often the service responds. When this is designed from a DevOps view, administrators have to take into consideration Anthos upgrade and scaling needs and plan accordingly so it does not affect these indicators.

For Service Level Objectives, think from the angle of the worst-case scenario, and not the best-case scenario, making that decision as data driven as possible. For example, if the latency is unrealistic and it does not have an impact on the user experience,

there will be no way to even release the service. Find the highest latency which is acceptable according to the user experience and then work on reducing that based on business needs. Educate your business stakeholders that 100% availability is very expensive to attain and that a practical tradeoff often has to be agreed. This is an important concept mentioned as well in the SRE book by Google, to strive to make a service reliable enough but no more than it has to be. You can find more information on the SRE book by Google at <https://landing.google.com/sre/sre-book/chapters/embracing-risk/>.

Understanding the procedures and risks of Anthos upgrades, rollbacks, and security updates is essential input to determining if a Service Level Objective is realistic or not.

A compliance period should also be defined for the Service Level Objective to be measured on. The set SLO can be any period of measurement: a day, week, or a month. This allows for the teams responsible for the service to decide when it is time to rollback, make a hotfix, or slow down development to prioritize fixing bugs.

The SLI and SLO also empowers product owners to propose Service Level Agreements with users which require it and offer a realistic latency and availability agreement.

5.5 *Anthos command line management*

These are various command line tools that deal with cluster creation, scaling and upgrading of Anthos versions, such as *gkectl*, *gkeadmin*, and *anthos-gke*. This chapter is not meant to replace the documentation on Google Cloud, but it summarizes the actions and some of the gotchas to look out for.

REMINDER Admin clusters are deployed purely to monitor and administer user clusters, think of them as the invisible control plane analogous to GKE and do not deploy services which can affect it there.

TIP You can use a kubeconfig manager like ktx from <https://github.com/heptiolabs/ktx>, which allows administrators to switch between admin and user cluster contexts easily.

In the next section, the segments will be broken up into GKE on-prem and GKE on AWS as the tools and installation process differ.

5.5.1 *Using CLI tools for GKE on-prem*

GKE on-prem installation uses the APIs from VMware⁴¹ to build an admin workstation, admin cluster nodes, and user cluster nodes programmatically. Persistent volumes are powered from individual VMWare's Datastore or vSAN, and networking is provided by either distributed or standard vSphere switches. These act like the IaaS components provided by Google Cloud when building a GKE cluster, thus the name GKE on-prem. The concept of having an Admin Cluster with User Clusters and Node Pools mirror with GKE best practices.

⁴¹ In addition to VMWare, it is possible to use Anthos on Bare Metal. That is the topic discussed in chapter 22.

The current installation process is to download a tool named *gkeadm*, which creates an admin workstation. It is from this admin workstation the admin cluster and user clusters are installed from. While there are versions of *gkeadm* available for Windows, Linux, and Mac OS, this section will only explain an abbreviated process for Linux.

- 1 The first step is to download the tool from the cloud storage bucket:

```
gsutil cp gs://gke-on-prem-release-public/gkeadm/<anthos
version>/linux/gkeadm ./chmod +x gkeadm
```

- 2 Next, create a pre-populated config file:

```
./gkeadm create config
```

- 3 Fill in the vCenter credentials, GCP whitelisted service account key path. (After purchasing Anthos, customers will be asked to provide a service account which Google will whitelist to be able to download images and other proprietary tools), vCenter Certificate Authority certs path.

- 4 The vCenter Certificate Authority certs can be downloaded by:

```
curl -k "https://[SERVER_ADDRESS]/certs/download.zip" > download.zip
```

- 5 After unzipping the download.zip file, the relevant certs can be found in the certs/lin folder. The file with .0 suffix is the root certificate. Rename it to vcenter.crt and use it in the reference from the installation config file.

The vCenter and BigIP F5 credentials are saved in plain text in the config file when creating new user clusters or on installation. One way to secure the F5 credentials is through using a wrapper around Google Cloud Secrets Manager and *gcloud*.

To create a password secured by Google Secret Manager:

```
echo "vcenterp455w0rd" | gcloud secrets create vcenterpass --data-file=- --
replication-policy=user-managed --locations=us-east1
```

To retrieve a password secured by Google Secret Manager:

```
gcloud secrets versions access latest --secret="vcenterpass"
```

This secret is now protected via Google IAM policies and a wrapper script can be written to retrieve the secret, replace the placeholder in the config file, apply, and then delete the file.

The process to create Anthos cluster components is quickly evolving, and it's not uncommon for a newer version to have some changes to the config file. You can follow this link for latest release procedures at <https://cloud.google.com/anthos/clusters/docs/on-prem/1.9/how-to/create-admin-workstation>.

5.5.2 Cluster management: Creating a new user cluster

The *gkectl* command is used for this operation.

As a rule of thumb, admins should constrain the setups so that there is a ratio of one admin cluster to ten user clusters. User clusters should have a minimum of three

nodes, with a maximum of 100 nodes. As previously mentioned, newer releases may increase these numbers. When a new Anthos release is published, you can check the new limits in the “Quotas and Limits” section of the respective release.

The general advice is to leave some space for at least one cluster, which can be created in your on-prem environment. This would give the operations team space to recreate clusters and move pods over when upgrading or triage.

Keep good documentation like which IP addresses have been already assigned for other user clusters, so that non-overlapping IPs can be determined easily. Take into consideration that user clusters can be resized to 100 nodes, so reserve up to 100 IP addresses per range to keep that possibility.

Source control your configuration files, but do not commit the vsphere username/passwords. Committing such sensitive information into repositories will open security risks because anyone with access to the repository will be able to get those login details. Tools like ytt can be used to template configuration yamls, code reviews, and repository scanners should be used to prevent such mistakes from taking place (e.g., <https://github.com/UKHomeOffice/repo-security-scanner>).

Node pools can also be created with different machine shapes, so size them correctly to accommodate your workloads. This also gives granular control over which machine types to scale and save costs. For production workloads, use three replicas for the user cluster master nodes for high availability, but for dev, one should be fine.

Validate the configuration file to make sure the file is valid. The checks are both syntactic and programmatic, such as checking for IP range clashes and IP availability using the `gkectl check-config` command:

```
gkectl check-config --kubeconfig [ADMIN_CLUSTER_KUBECONFIG] --config
[CONFIG_FILE]
```

After the first few validations, most time-consuming validations can be skipped by passing the `--fast` flag.

Next, the seesaw load balancer should be created if the bundled load balancer is chosen. If you do not create the Seesaw node(s) before attempting a cluster build that has been configured with the integrated load-balancer option, you will receive an error during the cluster pre-check. To create the Seesaw node(s) use the `gkectl create loadbalancer` command:

```
gkectl create loadbalancer --kubeconfig [ADMIN_CLUSTER_KUBECONFIG] --config
[CONFIG_FILE]
```

After creation of a new user cluster do remember that for the bundled lb seesaw version, the user will then be able to create the user cluster:

```
gkectl create cluster --kubeconfig [ADMIN_CLUSTER_KUBECONFIG] --config
[CONFIG_FILE]
```

You can also add the `--skip-validation-all` flag if the config file has already been validated.

The whole user cluster process can take 20-30 minutes, depending on the hardware, which consists of starting up new VMWare virtual machines with the master and worker node images and joining them into a cluster. The administrator is also able to see the nodes being created from the VMWare vCenter console.

High availability setup

High availability is necessary for Anthos deployments in production environments. This is important because failures can occur at different parts of the stack, ranging from networking, to hardware, to the virtualization layer.

High availability for admin clusters uses the vSphere High Availability in a vSphere Cluster setup to protect GKE on-prem clusters from going down in the event of a host failure. This ensures that admin cluster nodes are distributed among different physical nodes in a vSphere cluster, so that in the event of a physical node failure, the admin cluster will still be available.

To enable HA user control planes, simply specify `usercluster.master.replicas: 3` in the GKE on-prem configuration file. This will create three user cluster masters for each user cluster, consuming three times the resources, but providing a high availability Kubernetes setup.

5.5.3 Cluster management: Scaling

Administrators can use the `gkectl` cli to scale up or down nodes as seen here. They could change the config file to set the number of expected replicas and execute the following command to update the node pool.

```
gkectl update cluster --kubeconfig [USER_CLUSTER_KUBECONFIG] --config  
[CONFIG_FILE]
```

5.5.4 Cluster management: Upgrading Anthos

Like any upgrade process, there can be failures during the process. A lot of effort has been put into making the upgrade process robust, including the addition of pre-checks before executing the upgrade to catch potential issues before they occur. Each product team at Google works closely when an upgrade is being developed to avoid any potential incompatibilities between components like Kubernetes, ACM, and ASM.

Anthos versions are quite fast-moving due to industry demand for new features and this means that upgrading Anthos is a very common activity. Upgrading Anthos can also mean upgrading to a new version of Kubernetes, which also impacts Anthos Service Mesh due to Istio dependency on Kubernetes. This means the upgrade chain is complex, which is why the recommendation in the beginning is to keep some spare hardware resources that can be used to create new versions of Anthos clusters and then move workloads to the new cluster before tearing down the older version cluster. This process reduces the risk associated with upgrades by providing an easy rollback path in case of a failed upgrade. In this type of upgrade path, there should be a load-

balancer in front of the microservices running in the old cluster to be upgraded which can direct traffic from the old cluster to the new cluster, because they will exist at the same time.

However, if this is not an option, administrators are able to upgrade Anthos clusters in place.

Firstly, consult the upgrade paths. From GKE on-prem 1.3.2 onwards, administrators can upgrade directly to any version in the same minor release; otherwise, sequential upgrades are required. From version 1.7, administrators can keep their admin cluster on an older version, while only upgrading the admin workstation and the user cluster. As a best practice, administrators should still schedule the admin cluster upgrades to keep up to date.

Next, download the gkeadm tool, which has to be the same as the target version of your upgrade, and run gkeadm to upgrade the admin workstation and gkectl to upgrade your user cluster, and finally the admin cluster.

When upgrading in place, a new node is created with the image of the latest version and workloads are drained from the older version and shifted over to the latest version, one node after the other. This means administrators should plan for additional resources in their physical hosts to accommodate at least one user node for upgrade purposes. The full flow can be seen from figure 5.11.

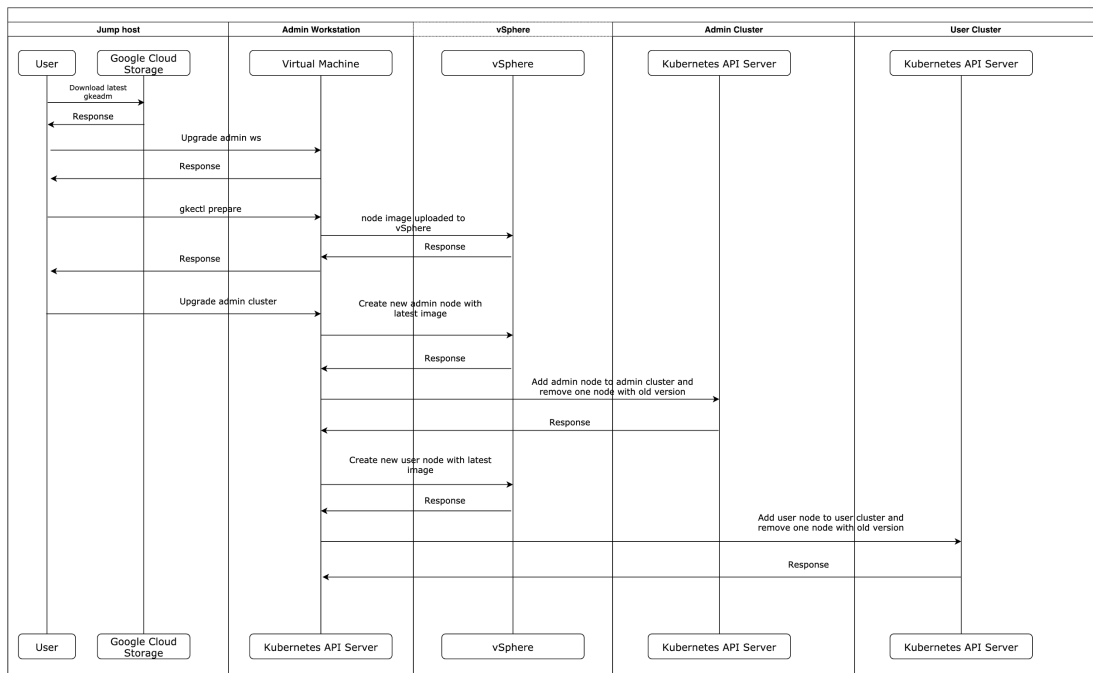


Figure 5.11 Upgrading flow.

For a detailed list of commands please consult the upgrading documentation at https://cloud.google.com/anthos/gke/docs/on-prem/how-to/upgrading#upgrading_your_admin_workstation.

5.5.5 Cluster management: Backing up clusters

Anthos admin clusters can be backed up by following the steps found at <https://cloud.google.com/anthos/clusters/docs/on-prem/1.8/how-to/back-up-and-restore-an-admin-cluster-with-gkectl>.

This is recommended to be done as part of a production Anthos environment setup to regularly schedule backups and to do on demand backups when upgrading Anthos versions.

Anthos user clusters etcd can be backed up by running a backup script which you can read more about backing up a cluster on the Anthos documentation page at <https://cloud.google.com/anthos/gke/docs/on-prem/how-to/backing-up>.

Do note that this only backs up the etcd of the clusters, which means the Kubernetes configuration. Google also states this to be a last resort. Backup for GKE promises to make this simpler, and we look forward to similar functionality for Anthos clusters soon. (<https://cloud.google.com/blog/products/storage-data-transfer/google-cloud-launches-backups-for-gke>).

Any application specific data, such as persistent volumes are not backed up by this process. Those should be backed up regularly to another storage device using a number of different tools like Velero.

You should treat your cluster backups the same as any data that is backed up from a server. The recommendation is to practice restoring an admin and user cluster from backup, along with application specific data to gain confidence in the backup and recovery process.

Targeted in the roadmap for H2 2021, Google will release a feature named Anthos Enterprise Data Protection functionality to back up cluster-wide config such as custom resource definitions, namespace-wide configuration, and application data from Google Cloud Console into a cloud storage bucket and be able to restore using the backup as well.

5.6 GKE on AWS

GKE on AWS uses AWS EC2 instances and other components to build GKE clusters, which means these are not EKS clusters. If a user logs into the AWS console, they will be able to see the admin cluster and user clusters nodes only as individual AWS EC2 instances. It is important to differentiate this between managing EKS clusters from Anthos because the responsibilities assigned to the different cloud providers according to each cluster type is different.

GKE on AWS installation is done via the *anthos-gke* cli, which can be run on Linux or MacOS as of the time of writing. The next release of GKE on AWS set for Q4 2021

will use the gcloud cli and follow the same architecture as GKE on Azure in this section with a single multi-cloud API for managing cluster life cycle operations. This will further simplify the installation process and remove the need for a bastion host and management server mentioned in the steps below.

The installation process is to first get an AWS KMS key, then use anthos-gke which in turn uses Terraform to generate Terraform code. Terraform is an Infrastructure as Code open-source tool to define a target state of a computing environment by Hashicorp. Terraform code is declarative and utilizes Terraform providers which are often contributed by cloud providers such as Google, AWS, and Microsoft to map their cloud provisioning APIs to Terraform code. The resulting Terraform code describes how the GKE on AWS infrastructure will look like. It has components that are analogous to GKE on-prem, such as a LoadBalancer, EC2 Virtual Machines, but leverage the Terraform AWS Provider work to instantiate the infrastructure on AWS.

You can learn more about Terraform at <https://www.terraform.io/>.

The architecture of GKE on AWS can be seen on figure 5.12 is from the Google Cloud documentation at <https://cloud.google.com/anthos/gke/docs/aws/concepts/architecture>.

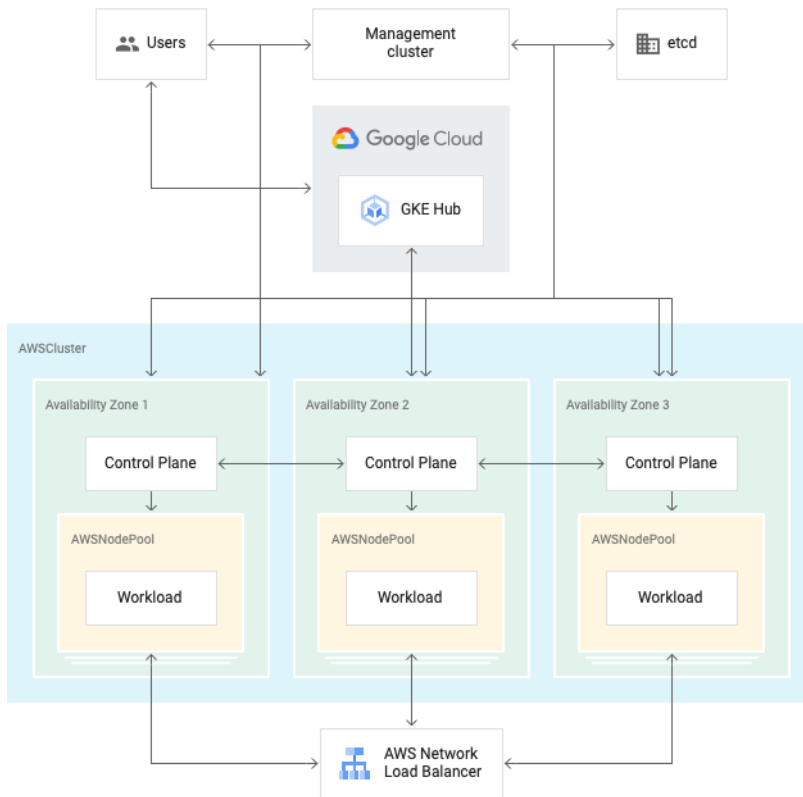


Figure 5.12 GKE on AWS architecture.

The use of node pools is similar to GKE, with the ability in a cluster to have different machine sizes.

NOTE To do any GKE on AWS operations management, the administrator will have to log into the bastion host which is part of the management service.

5.6.1 Connecting to the management service

When doing any management operations, the administrator needs to connect to the bastion host deployed during the initial installation of the management service. This script is named `bastion-tunnel.sh`, which is generated from Terraform during the management service installation.

5.6.2 Cluster management: Creating a new user cluster

Use the `bastion-tunnel` script to connect to the management service.

After connecting to the bastion host, the administrator will use Terraform to generate a manifest configuring an example cluster in a yaml file:

```
terraform output cluster_example > cluster-0.yaml
```

In this yaml file the administrator will then change the `AWSCluster` and `AWSNodePool` specifications. Be sure to save the cluster file to a code repository, this will be reused for scaling the user cluster.

Custom Resources are extensions of Kubernetes to add additional functionality, such as in the case of provisioning AWS EC2 instances. AWS Clusters and objects are represented as yaml referencing the `AWSCluster` and `AWSNodePool` Custom Resources in the management service cluster, which interpret this yaml and adjust resources in AWS accordingly.

To read more about Custom Resources please refer to <https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/>.

5.6.3 Cluster management: Scaling

You may experience a situation where a cluster requires additional compute power, and you need to scale the cluster out. Luckily, an Anthos node pool has an option to scale a cluster, including a minimum and maximum node count. If you created a cluster with the same count in both the minimum and maximum nodes, you could change it at a later date to grow your cluster. To scale a cluster for GKE on AWS, you simply require the administrator to modify the YAML file by updating the `minNodeCount` while creating the user cluster and applying it to the management service:

```
apiVersion: multicloud.cluster.gke.io/v1
kind: AWSNodePool
metadata:
  name: cluster-0-pool-0
spec:
  clusterName: cluster-0
  version: 1.20.10-gke.600
  minNodeCount: 3
  maxNodeCount: 10
```

5.6.4 Cluster management: Upgrading

Upgrading GKE on AWS is done in two steps, with the management service handled first, and then the user clusters.

To upgrade a GKE on AWS management service, the administrator has to do this from the directory with the GKE on AWS configuration.

The user has to first download the latest version of the anthos-gke binary. Next the user will have to modify the anthos-gke.yaml file to the target version:

```
apiVersion: multicloud.cluster.gke.io/v1
kind: AWSManagementService
metadata:
  name: management
spec:
  version: <target_version>
```

Finally, to validate and apply the version changes by running:

```
anthos-gke aws management init
anthos-gke aws management apply
```

The management service will be down, so no changes to user clusters can be applied, but user clusters continue to run their workloads.

To upgrade the user cluster, the administrator switches context in the management service from the GKE on AWS directory using the following command:

```
anthos-gke aws management get-credentials
```

Then upgrading the version of the user cluster is as simple as using:

```
kubectl edit awscluster <cluster_name>
```

And then editing the yaml to point to the right GKE version:

```
apiVersion: multicloud.cluster.gke.io/v1
kind: AWSCluster
metadata:
  name: cluster-0
spec:
  region: us-east-1
  controlPlane:
    version: <gke_version>
```

On submission of this change, the CRD starts to go through the nodes in the control plane one by one and start upgrading them to the latest GKE on AWS version. This upgrade process causes a downtime of the control plane, which means the cluster may be unable to report the status of the different node pools until it is completed.

Finally, the last step is to upgrade the actual Node Pool. The same procedure applies, and the administrator simply edits the yaml to the version required and applies the yaml to the management service:

```
apiVersion: multicloud.cluster.gke.io/v1
kind: AWSNodePool
metadata:
```

```

name: cluster-0-pool-0
spec:
  clusterName: cluster-0
  region: us-east-1
  version: <gke-version>

```

5.7 Anthos attached clusters

Anthos attached clusters are managed conformant Kubernetes, which are provisioned and managed by Elastic Kubernetes Service (EKS) by AWS, Azure Kubernetes Service (AKS) or any conformant Kubernetes cluster. In this case, the scaling and provisioning of the clusters are done from the respective clouds. However, these clusters can still be attached and managed by Anthos by registering them to Google Cloud through deployment of the Connect agent as seen from figure 5.13. Google Kubernetes Engine is also handled in the same way and can be attached from another project into the Anthos project.

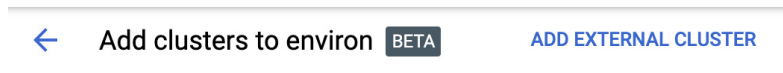


Figure 5.13 Adding an external cluster (bring your own Kubernetes).

- 1 The administrator has to generate a kubeconfig to the EKS or AKS cluster, and then provide that kubeconfig in a generated cluster registration command in gcloud. Please consult documentation from AWS and Azure on how to generate a kubeconfig file for the EKS or AKS clusters. The administrator is also able to generate one manually using the template below and providing the necessary certificate, server info and service account token:

```

apiVersion: v1
kind: Config
users:
- name: svcs-acct-dply
  user:
    token: <replace this with token info>
clusters:
- cluster:
    certificate-authority-data: <replace this with certificate-
      authority-data info>
    server: <replace this with server info>
    name: self-hosted-cluster
contexts:
- context:
    cluster: self-hosted-cluster
    user: svcs-acct-dply
    name: svcs-acct-context
current-context: svcs-acct-context

```

- 2 The administrator has to create a Google service account and a service account key to provide for the registration as seen in figure 5.14:

← Register a Kubernetes cluster

Install [Connect](#) into your cluster and register your cluster to Google Cloud Platform. GKE Connect works behind firewalls and can traverse NAT's to establish an encrypted connection to Google Cloud Platform. [Learn more.](#)

Cluster name
eks-cluster

Configure GCP labels for the cluster.

Key *	Value
location	stockholm
provider	aws

+ ADD LABEL

CHANGE CLUSTER NAME

CANCEL

To register the cluster, run the following command:

```
$ gcloud container hub memberships register eks-cluster \
  --context=[CLUSTER_CONTEXT] \
  --service-account-key-file=[LOCAL_KEY_PATH] \
  --kubeconfig=[KUBECONFIG_PATH] \
  --project=anthos-sandbox-256114
```



Waiting for eks-cluster GKE Connect connection with Google



Waiting for eks-cluster Membership to be created

Figure 5.14 Generating registration command to the external cluster.

- The administrator will provide these two into the generated registration command, and after the connect agent has been deployed into the external cluster, it will be visible in the Google Cloud Console.

5.8 *Anthos on Bare Metal*

Operating and managing Anthos on Bare Metal often requires additional skill sets in the OS configuration space as it is based on installing Anthos on RHEL, Ubuntu, or CentOS. For the detailed steps in installation and upgrading of Anthos on Bare Metal, please consult the chapter 22 “Anthos on Bare Metal”.

Anthos on Bare Metal is somehow similar to Anthos on VMWare, but with more flexibility in its deployment models and no dependency on VMWare.

A few key decisions must be reached when designing the operations management procedures for Anthos on Bare Metal.

First is capacity planning and resource estimation for running Anthos on Bare Metal. Unlike the rest of the set ups, where new nodes need to be provisioned using either public cloud resources or a pool of VMWare resources, new Bare Metal nodes have to be provisioned. This requires additional capacity requirements if there is a zero-downtime requirement during upgrades of the nodes as there is always a risk of nodes failing upgrades and causing a decrease in capacity.

Second is automating as much as possible the prerequisite installation of the nodes. Many companies also require a golden image of a base operating system which has to be vetted by a security team and continuously updated with security patches and latest versions. This should be built into the Anthos on Bare Metal provisioning process to be able to verify compatibility with Anthos installation. One option is to set up PXE boot servers and have newly provisioned Bare Metal servers point to the PXE boot servers to install the operating system of bare metal nodes to the right configuration.

Third is determining the different deployments to run Anthos on Bare Metal, in standalone, multi-cluster, or hybrid cluster deployments. Flexibility also means complexity and having to build different operational models for the different deployments. The Anthos Bare Metal goes more into detail about the differences, but this chapter highlights the different operational considerations when choosing the different deployment models.

- *Standalone cluster deployment:* This deployment model has the admin and user clusters in the same cluster. In such a configuration, workloads run in the same nodes that have ssh credentials and Google service account keys are stored. This configuration is well suited for edge deployment and as such operational models should introduce ssh credential and service account key generation for each new standalone cluster provisioned and deployed, and a plan to decommission those credentials when a cluster is compromised or lost. There is a minimum requirement of five nodes for a high availability setup.
- *Multi-cluster deployment:* This deployment model has an admin cluster and one or more user clusters, similar to Anthos on VMWare. This has many benefits, such as admin—user isolation for the clusters, multi-tenanted setups (e.g., each team can have their own cluster) and a centralized plan for upgrades. The downside is the increased footprint in node requirements, and a minimum of eight nodes for a high availability setup. This would take more effort when setting up for multiple edge locations and is more for a datacenter setup.
- *Hybrid cluster deployment:* This deployment model allows for running of user workloads on the admin clusters, and managing other user clusters. This reduces the footprint required for multi-cluster deployment to five nodes for a high availability setup but has the same security concern of running user workloads on nodes which may contain sensitive data from the stand-alone cluster deployment. This gives the flexibility to tier workloads by security levels and introduce user clusters for workloads which require higher security.

5.9 Connect Gateway

Registering Anthos clusters allows the user to interact with them through the UI, but administrators often have a toolbox of scripts which they use to work with clusters through the *kubectl* command line. With GKE On-prem/on AWS/on Azure, these clusters often can only be accessed via either the admin workstation or a bastion host. GKE users on the other hand can use *gcloud* and generate kubeconfig details to *kubectl* to their clusters on their local machines. With Connect Gateway, this problem is solved.

Administrators are also able to connect to any Anthos registered cluster and generate kubeconfig that enables the user to use *kubectl* to those clusters via the Connect agent.

With this feature, administrators will not be required to utilize jump hosts to deploy to the GKE on X clusters, but instead run a *gcloud* command to generate a kubeconfig to connect via *kubectl*.

The setup requires an impersonation policy which allows the Connect agent service account to impersonate an user account to issue commands on their behalf. An example of the yaml which creates the ClusterRole and ClusterRoleBinding for impersonation can be seen here:

```
# [USER_ACCOUNT] is an email, either USER_EMAIL_ADDRESS or
  GCPSA_EMAIL_ADDRESS
$ USER_ACCOUNT=foo@example.com
$ cat <<EOF > /tmp/impersonate.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: gateway-impersonate
rules:
- apiGroups:
  - ""
  resourceNames:
  - ${USER_ACCOUNT}
  resources:
  - users
  verbs:
  - impersonate
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: gateway-impersonate
roleRef:
  kind: ClusterRole
  name: gateway-impersonate
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: connect-agent-sa
  namespace: gke-connect
EOF
```

After the impersonation policy has been set up, the administrator has to run the command in figure 5.16 below to generate a kubeconfig seen in figure 5.15.

```
gcloud alpha container hub memberships get-credentials stockholm-gke-cluster
```

Figure 5.15 Command to get credentials to GKE on-prem cluster.

```
- cluster:
  server: https://connectgateway.googleapis.com/v1alpha1/projects/74668819743/memberships/stockholm-gke-cluster
  name: connectgateway_anthos-sandbox-256114_stockholm-gke-cluster
```

Figure 5.16 kubeconfig generated via gcloud.

With this kubeconfig in place, administrators are able to manage GKE on-prem workloads even from their local machine, while being secured by their Google Cloud Identity. This also opens up the possibility for building pipelines to deploy to the different Anthos clusters.

5.10 Anthos on Azure

Anthos GKE clusters can be installed on Azure with an architecture consisting of a Multi-Cloud API hosted on GCP that provides life cycle management capabilities to GKE Clusters in Azure as seen in figure 5.17. Azure GKE Clusters are also accessed via the Connect Gateway mentioned in this chapter. Anthos on Azure uses Azure native technologies like the Azure Load Balancer, Azure AD, and Azure Virtual Machines, but relies on Anthos via the Multi-Cloud API to manage GKE cluster life cycle operations. This creates an uniform way to deploy applications across the three major public clouds and on premise.

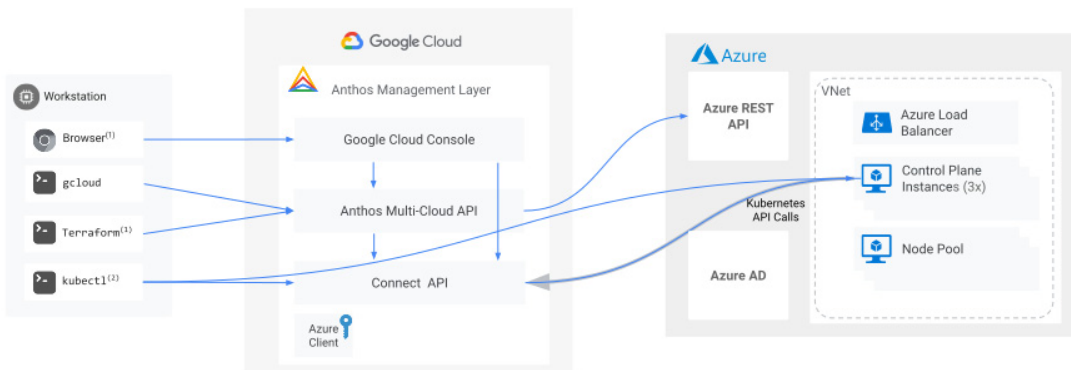


Figure 5.17 Anthos on Azure architecture.

As a prerequisite, the administrator has to install the gcloud CLI.

The administrator has to have the following Azure built-in roles:

- 1 Application Administrator
- 2 User Access Administrator
- 3 Contributor

The next steps would be to create an Azure Active Directory application, a virtual network, a resource group for the clusters, and grant the necessary permissions to the Azure Active Directory application. Detailed prerequisite information can be found in the [public documentation](#).

5.10.1 Cluster management: Creation

To create a new user cluster, the administrator will have to first set up an azure client with ssh key pair:

```
export SUBSCRIPTION_ID=$(az account show --query "id" --output tsv)
export TENANT_ID=$(az account list \
  --query "[?id=='${SUBSCRIPTION_ID}'].tenantId" --output tsv)
export APPLICATION_ID=$(az ad app list --all \
  --query "[?displayName=='APPLICATION_NAME'].appId" --output tsv)

gcloud alpha container azure clients create CLIENT_NAME \
  --location=GOOGLE_CLOUD_LOCATION \
  --tenant-id="${TENANT_ID}" \
  --application-id="${APPLICATION_ID}"

CERT=$(gcloud alpha container azure clients get-public-cert --
  location=GOOGLE_CLOUD_LOCATION \
  CLIENT_NAME)

az ad app credential reset --id "${APPLICATION_ID}" --cert "${CERT}" --append

ssh-keygen -m PEM -t rsa -b 4096 -f KEY_PATH

SSH_PUBLIC_KEY=$(cat KEY_PATH.pub)

ssh-keygen -m PEM -t rsa -b 4096 -f ~/.ssh/anthos-multicloud-key
SSH_PUBLIC_KEY=$(cat ~/.ssh/anthos-multicloud-key.pub)
```

Next the administrator will need to assign Azure resource groups, VNet and subnet IDs to environment variables, add IAM permissions and run the gcloud command to create the Anthos on Azure cluster:

```
CLUSTER_RG_ID=$(az group show --resource-group=CLUSTER_RESOURCE_GROUP_NAME \
  --query "id" -otsv)
VNET_ID=$(az network vnet show --resource-group=VNET_RESOURCE_GROUP_NAME \
  --name=VNET_NAME --query "id" -otsv)
SUBNET_ID=$(az network vnet subnet show \
  --resource-group=VNET_RESOURCE_GROUP_NAME --vnet-name=VNET_NAME \
  --name default --query "id" -otsv)
```

```
PROJECT_ID=$(gcloud config get-value project)
gcloud projects add-iam-policy-binding "$PROJECT_ID" \
  --member="serviceAccount:$PROJECT_ID.svc.id.goog[gke-system/gke-multicloud-
    agent]" \
  --role="roles/gkehub.connect"

gcloud alpha container azure clusters create CLUSTER_NAME \
  --location GOOGLE_CLOUD_LOCATION \
  --client CLIENT_NAME \
  --azure-region AZURE_REGION \
  --pod-address-cidr-blocks POD_CIDR \
  --service-address-cidr-blocks SERVICE_CIDR \
  --vm-size VM_SIZE \
  --cluster-version 1.19.10-gke.1000 \
  --ssh-public-key "$SSH_PUBLIC_KEY" \
  --resource-group-id "$CLUSTER_RG_ID" \
  --vnet-id "$VNET_ID" \
  --subnet-id "$SUBNET_ID"
```

This cluster should then be available on the administrator's GKE console.

Finally add a node pool to be able to deploy workloads to the cluster:

```
SUBNET_ID=$(az network vnet subnet show \
  --resource-group=VNET_RESOURCE_GROUP_NAME --vnet-name=VNET_NAME \
  --name default --query "id" -otsv)
SSH_PUBLIC_KEY=$(cat KEY_PATH.pub)

gcloud alpha container azure node-pools create NODE_POOL_NAME \
  --cluster=CLUSTER_NAME \
  --location GOOGLE_CLOUD_LOCATION \
  --node-version=1.19.10-gke.1000 \
  --vm-size=VM_SIZE \
  --max-pods-per-node=110 \
  --min-nodes=MIN_NODES \
  --max-nodes=MAX_NODES \
  --ssh-public-key="${SSH_PUBLIC_KEY}" \
  --subnet-id="${SUBNET_ID}"
```

5.10.2 Cluster management: Deletion

To delete a cluster, administrators will have to first delete all the node pools which belong to a cluster, before deleting the cluster:

```
gcloud alpha container azure node-pools delete NODE_POOL_NAME \
  --cluster CLUSTER_NAME \
  --location GOOGLE_CLOUD_LOCATION

gcloud alpha container azure clusters delete CLUSTER_NAME \
  --location GOOGLE_CLOUD_LOCATION
```

With an autoscaler ready to go with Anthos on Azure, it is easy for the administrator to control costs and manage minimum resource requirements for each cluster. It is recommended to have a security device like Hashicorp Vault to store the ssh keys for retrieval and rotation.

Summary

The best way to learn is by trying and the best advice is to try building clusters on the various providers to understand the optimizations available, and the actions that an administrator would need to do in their day-to-day life managing operations in Anthos. This is key to build a continuous improvement process as new features in Anthos are released to make kubernetes cluster management always easier and faster.

After reading this chapter, the reader should be able to:

- Use Google Cloud Console to operate and manage workloads of the various Anthos cluster types
- Understand the various logging and monitoring options available with their Anthos deployments and the criteria to consider
- Understand how to operate and manage various types of Anthos deployments through the command line, and how and what kind of communication happens between Google Cloud and the deployments
- Upgrade, scale, and design operations management procedures in Anthos across an hybrid environment

In the next chapter, the features described in the previous few chapters will be discussed as an overview of the Anthos product, roadmap, and strategy.