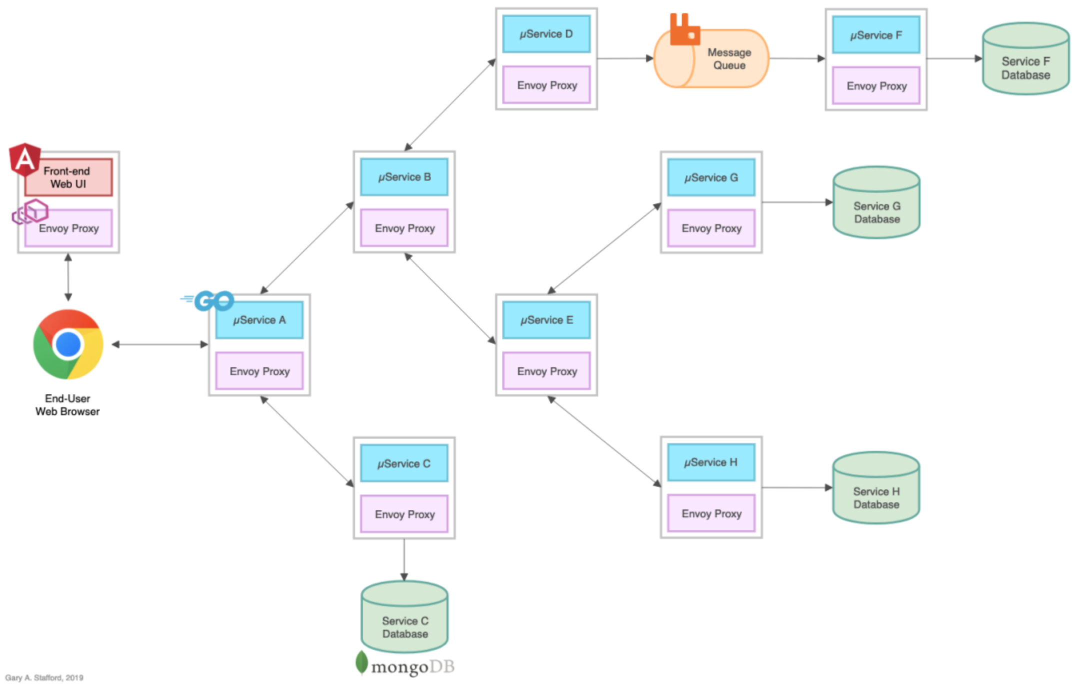
Kubernetes-based Microservice Observability with Istio Service Mesh: Part 1

In this two-part (https://programmaticponderings.com/2019/03/21/kubernetes-based-microservice-observability-with-istio-service-mesh-part-2/) post, we will explore the set of observability tools which are part of the Istio Service Mesh. These tools include Jaeger, Kiali, Prometheus, and Grafana. To assist in our exploration, we will deploy a Go-based, microservices reference platform to Google Kubernetes Engine, on the Google Cloud Platform.



(https://programmaticponderings.files.wordpress.com/2019/03/golang-service-diagram-with-proxy-v2.png)

What is Observability?

Similar to blockchain, serverless, AI and ML, chatbots, cybersecurity, and service meshes, Observability is a hot buzz word in the IT industry right now. According to Wikipedia, observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs. Logs, metrics, and traces are often known as the three pillars of observability. These are the external outputs of the system, which we may observe.

The O'Reilly book, Distributed Systems Observability (https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/), by Cindy Sridharan, does an excellent job of detailing 'The Three Pillars of Observability', in Chapter 4 (https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/ch04.html). I recommend reading this free online excerpt, before continuing. A second great resource for information on observability is honeycomb.io (https://www.honeycomb.io/), a developer of observability tools for production systems, led by well-known industry thought-leader, Charity Majors (https://twitter.com/mipsytipsy). The honeycomb.io site includes articles, blog posts, whitepapers, and podcasts on observability.

As modern distributed systems grow ever more complex, the ability to observe those systems demands equally modern tooling that was designed with this level of complexity in mind. Traditional logging and monitoring systems often struggle with today's hybrid and multi-cloud, polyglot language-based, event-driven, container-based and serverless, infinitely-scalable, ephemeral-compute platforms.

Tools like Istio Service Mesh (https://istio.io/) attempt to solve the observability challenge by offering native integrations with several best-of-breed, open-source telemetry tools. Istio's integrations include Jaeger (https://www.jaegertracing.io/) for distributed tracing, Kiali (https://www.kiali.io/) for Istio service mesh-based microservice visualization, and Prometheus (https://prometheus.io/) and Grafana (https://grafana.com/) for metric collection, monitoring, and alerting. Combined with cloud platform-native monitoring and logging services, such as Stackdriver (https://cloud.google.com/monitoring/) for Google Kubernetes Engine (https://cloud.google.com/kubernetes-engine/) (GKE) on Google Cloud Platform (GCP), we have a complete observability platform for modern, distributed applications.

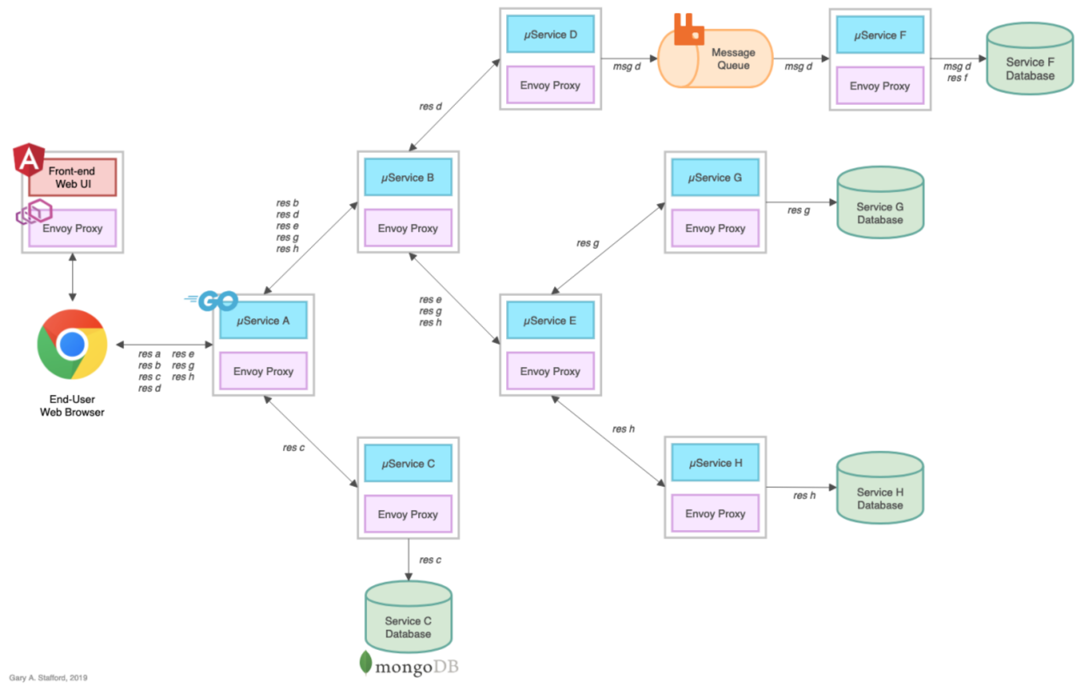
A Reference Microservices Platform

To demonstrate the observability tools integrated with the latest version of Istio Service Mesh, we will deploy a reference microservices platform, written in Go, to GKE on GCP. I developed the reference platform to demonstrate concepts such as API management, Service Meshes, Observability, DevOps, and Chaos Engineering (https://principlesofchaos.org/). The platform is comprised of (14) components, including (8) Go-based (https://golang.org/) microservices, labeled generically as Service A – Service H, (1) Angular 7, TypeScript-based (https://en.wikipedia.org/wiki/TypeScript) front-end, (4) MongoDB databases, and (1) RabbitMQ queue for event queue-based communications. The platform and all its source code is free and open source.

The reference platform is designed to generate HTTP-based service-to-service, TCP-based service-to-database (MongoDB), and TCP-based service-to-queue-to-service (RabbitMQ) IPC (inter-process communication). Service A calls Service B and Service C, Service B calls Service D and Service E, Service D produces a message on a RabbitMQ queue that Service F consumes and writes to MongoDB, and so on. These distributed communications can be observed using Istio's observability tools when the system is deployed to a Kubernetes cluster running the Istio service mesh.

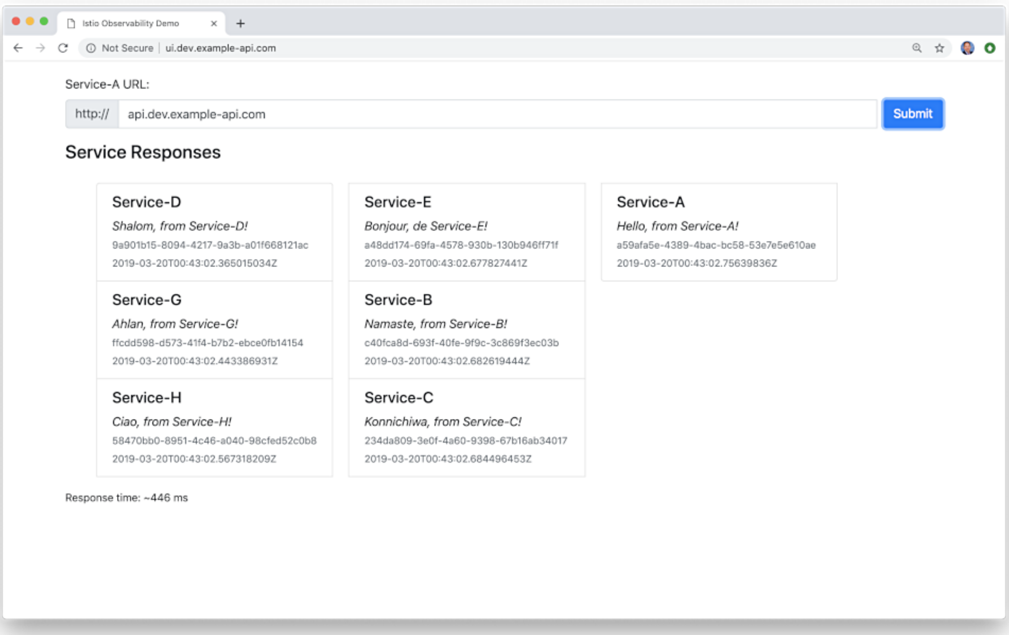
Service Responses

On the reference platform, each upstream service responds to requests from downstream services by returning a small informational JSON payload (termed a *greeting* in the source code).



(https://programmaticponderings.files.wordpress.com/2019/03/golang-service-diagram-with-proxy-v2-res.png)

The responses are aggregated across the service call chain, resulting in an array of service responses being returned to the edge service and on to the Angular-based UI, running in the end user's web browser. The response aggregation feature is simply used to confirm that the service-to-service communications, Istio components, and the telemetry tools are working properly.



Service-A URL:

http:// api.dev.example-api.com

Service Responses

Service-D
Shalom, from Service-D!
9a901b15-8094-4217-9a3b-a01f668121ac
2019-03-20T00:43:02.365015034Z

Service-E
Bonjour, de Service-E!
a48dd174-69fa-4578-930b-130b946ff71f
2019-03-20T00:43:02.677827441Z

Service-A
Hello, from Service-A!
a59afa5e-4389-4bac-bc58-53e7e5e610ae
2019-03-20T00:43:02.75639836Z

Service-G
Ahlan, from Service-G!
ffcdd598-d573-41f4-b7b2-ebce0fb14154
2019-03-20T00:43:02.443386931Z

Service-B
Namaste, from Service-B!
c40fca8d-693f-40fe-9f9c-3c869f3ec03b
2019-03-20T00:43:02.682619444Z

Service-H
Ciao, from Service-H!
58470bb0-8951-4c46-a040-98cfed52c0b8
2019-03-20T00:43:02.567318209Z

Service-C
Konnichiwa, from Service-C!
234da809-3e0f-4a60-9398-67b16ab34017
2019-03-20T00:43:02.684496453Z

Response time: ~446 ms

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-19_at_8_43_10_pm.png)

Each Go microservice contains a `/ping` and `/health` endpoint. The `/health` endpoint can be used to configure Kubernetes Liveness and Readiness Probes. Additionally, the edge service, Service A, is configured for Cross-Origin Resource Sharing (https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS) (CORS) using the `access-control-allow-origin: *` response header. CORS allows the Angular UI, running in end user's web browser, to call the Service A `/ping` endpoint, which resides in a different subdomain from UI. Shown below is the Go source code for Service A (https://github.com/garystafford/golang-srv-demo/blob/master/service-a/main.go).

```
// author: Gary A. Stafford
// site: https://programmaticponderings.com
// license: MIT License
// purpose: Service A

package main

import (
	"encoding/json"
	"github.com/banzaicloud/logrus-runtime-formatter"
	"github.com/google/uuid"
	"github.com/gorilla/mux"
	"github.com/prometheus/client_golang/prometheus/promhttp"
	"github.com/rs/cors"
	log "github.com/sirupsen/logrus"
	"io/ioutil"
	"net/http"
	"os"
	"strconv"
	"time"
)

type Greeting struct {
	ID          string    `json:"id,omitempty"`
	ServiceName string    `json:"service,omitempty"`
	Message     string    `json:"message,omitempty"`
	CreatedAt   time.Time `json:"created,omitempty"`
}

var greetings []Greeting

func PingHandler(w http.ResponseWriter, r *http.Request) {
	w.Header().Set("Content-Type", "application/json; charset=utf-8")

	log.Debug(r)

	greetings = nil

	CallNextServiceWithTrace("http://service-b/api/ping", w, r)
	CallNextServiceWithTrace("http://service-c/api/ping", w, r)

	tmpGreeting := Greeting{
		ID:         uuid.New().String(),
		ServiceName: "Service-A",
		Message:     "Hello, from Service-A!",
		CreatedAt:   time.Now().Local(),
	}

	greetings = append(greetings, tmpGreeting)

	err := json.NewEncoder(w).Encode(greetings)
	if err != nil {
		log.Error(err)
	}
}

func HealthCheckHandler(w http.ResponseWriter, r *http.Request) {
	w.Header().Set("Content-Type", "application/json; charset=utf-8")
	_, err := w.Write([]byte("{\"alive\": true}"))
	if err != nil {
		log.Error(err)
	}
}

func ResponseStatusHandler(w http.ResponseWriter, r *http.Request) {
	params := mux.Vars(r)
	statusCode, err := strconv.Atoi(params["code"])
	if err != nil {
		log.Error(err)
	}
	w.WriteHeader(statusCode)

}

func CallNextServiceWithTrace(url string, w http.ResponseWriter, r *http.Request) {
	var tmpGreetings []Greeting

	w.Header().Set("Content-Type", "application/json; charset=utf-8")

	req, err := http.NewRequest("GET", url, nil)
	if err != nil {
		log.Error(err)
	}

	// Headers must be passed for Jaeger Distributed Tracing
	headers := []string{
		"x-request-id",
		"x-b3-traceid",
		"x-b3-spanid",
		"x-b3-parentspanid",
		"x-b3-sampled",
		"x-b3-flags",
		"x-ot-span-context",
	}

	for _, header := range headers {
		if r.Header.Get(header) != "" {
			req.Header.Add(header, r.Header.Get(header))
		}
	}

	log.Info(req)

	client := &http.Client{}
	response, err := client.Do(req)

	if err != nil {
		log.Error(err)
	}

	defer response.Body.Close()

	body, err := ioutil.ReadAll(response.Body)
	if err != nil {
		log.Error(err)
	}

	err = json.Unmarshal(body, &tmpGreetings)
	if err != nil {
		log.Error(err)
	}

	for _, r := range tmpGreetings {
		greetings = append(greetings, r)
	}
}

func getEnv(key, fallback string) string {
	if value, ok := os.LookupEnv(key); ok {
		return value
	}
	return fallback
```

```go
}

func init() {
	formatter := runtime.Formatter{ChildFormatter: &log.JSONFormatter{}}
	formatter.Line = true
	log.SetFormatter(&formatter)
	log.SetOutput(os.Stdout)
	level, err := log.ParseLevel(getEnv("LOG_LEVEL", "info"))
	if err != nil {
		log.Error(err)
	}
	log.SetLevel(level)
}

func main() {
	c := cors.New(cors.Options{
		AllowedOrigins:   []string{"*"},
		AllowCredentials: true,
		AllowedMethods:   []string{"GET", "POST", "PATCH", "PUT", "DELETE", "OPTIONS"
	})
	router := mux.NewRouter()
	api := router.PathPrefix("/api").Subrouter()
	api.HandleFunc("/ping", PingHandler).Methods("GET", "OPTIONS")
	api.HandleFunc("/health", HealthCheckHandler).Methods("GET", "OPTIONS")
	api.HandleFunc("/status/{code}", ResponseStatusHandler).Methods("GET", "OPTIONS")
	api.Handle("/metrics", promhttp.Handler())
	handler := c.Handler(router)
	log.Fatal(http.ListenAndServe(":80", handler))
}
```

view raw main.go hosted with ❤ by **GitHub**
For this demonstration, the MongoDB databases will be hosted, external to the services on GCP, on MongoDB Atlas (https://www.mongodb.com/cloud/atlas), a MongoDB-as-a-Service, cloud-based platform. Similarly, the RabbitMQ queues will be hosted on CloudAMQP (https://www.cloudamqp.com/), a RabbitMQ-as-a-Service, cloud-based platform. I have used both of these SaaS providers in several previous posts. Using external services will help us understand how Istio and its observability tools collect telemetry for communications between the Kubernetes cluster and external systems.

Shown below is the Go source code for Service F (https://github.com/garystafford/golang-srv-demo/blob/master/service-f/main.go), This service consumers messages from the RabbitMQ queue, placed there by Service D (https://github.com/garystafford/golang-srv-demo/blob/master/service-D/main.go), and writes the messages to MongoDB.

```go
// author: Gary A. Stafford
// site: https://programmaticponderings.com
// license: MIT License
// purpose: Service F
```

```go

package main

import (
	"bytes"
	"context"
	"encoding/json"
	"github.com/banzaicloud/logrus-runtime-formatter"
	"github.com/google/uuid"
	"github.com/gorilla/mux"
	log "github.com/sirupsen/logrus"
	"github.com/streadway/amqp"
	"go.mongodb.org/mongo-driver/mongo"
	"go.mongodb.org/mongo-driver/mongo/options"
	"net/http"
	"os"
	"time"
)

type Greeting struct {
	ID          string    `json:"id,omitempty"`
	ServiceName string    `json:"service,omitempty"`
	Message     string    `json:"message,omitempty"`
	CreatedAt   time.Time `json:"created,omitempty"`
}

var greetings []Greeting

func PingHandler(w http.ResponseWriter, r *http.Request) {
	w.Header().Set("Content-Type", "application/json; charset=utf-8")

	greetings = nil

	tmpGreeting := Greeting{
		ID:          uuid.New().String(),
		ServiceName: "Service-F",
		Message:     "Hola, from Service-F!",
		CreatedAt:   time.Now().Local(),
	}

	greetings = append(greetings, tmpGreeting)

	CallMongoDB(tmpGreeting)

	err := json.NewEncoder(w).Encode(greetings)
	if err != nil {
		log.Error(err)
	}
}

func HealthCheckHandler(w http.ResponseWriter, r *http.Request) {
	w.Header().Set("Content-Type", "application/json; charset=utf-8")
	_, err := w.Write([]byte("{\"alive\": true}"))
	if err != nil {
		log.Error(err)
	}
}

func CallMongoDB(greeting Greeting) {
	log.Info(greeting)
	ctx, _ := context.WithTimeout(context.Background(), 10*time.Second)
	client, err := mongo.Connect(ctx, options.Client().ApplyURI(os.Getenv("MONGO_CONN
	if err != nil {
		log.Error(err)
	}

	defer client.Disconnect(nil)

	collection := client.Database("service-f").Collection("messages")
	ctx, _ = context.WithTimeout(context.Background(), 5*time.Second)

	_, err = collection.InsertOne(ctx, greeting)
	if err != nil {
		log.Error(err)
	}
}

func GetMessages() {
	conn, err := amqp.Dial(os.Getenv("RABBITMQ_CONN"))
	if err != nil {
		log.Error(err)
	}
	defer conn.Close()

	ch, err := conn.Channel()
	if err != nil {
		log.Error(err)
	}
	defer ch.Close()

	q, err := ch.QueueDeclare(
		"service-d",
		false,
		false,
		false,
		false,
		nil,
	)
	if err != nil {
		log.Error(err)
	}

	msgs, err := ch.Consume(
		q.Name,
		"service-f",
		true,
		false,
		false,
		false,
		nil,
	)
	if err != nil {
		log.Error(err)
	}

	forever := make(chan bool)

	go func() {
		for delivery := range msgs {
			log.Debug(delivery)
			CallMongoDB(deserialize(delivery.Body))
		}
	}()

	<-forever
}

func deserialize(b []byte) (t Greeting) {
	log.Debug(b)
	var tmpGreeting Greeting
	buf := bytes.NewBuffer(b)
	decoder := json.NewDecoder(buf)
	err := decoder.Decode(&tmpGreeting)
	if err != nil {
		log.Error(err)
	}
	return tmpGreeting
}

func getEnv(key, fallback string) string {
	if value, ok := os.LookupEnv(key); ok {
		return value
	}
	return fallback
}

func init() {
	formatter := runtime.Formatter{ChildFormatter: &log.JSONFormatter{}}
	formatter.Line = true
	log.SetFormatter(&formatter)
	log.SetOutput(os.Stdout)
	level, err := log.ParseLevel(getEnv("LOG_LEVEL", "info"))
	if err != nil {
		log.Error(err)
	}
	log.SetLevel(level)
}

func main() {
	go GetMessages()

	router := mux.NewRouter()
	api := router.PathPrefix("/api").Subrouter()
	api.HandleFunc("/ping", PingHandler).Methods("GET")
	api.HandleFunc("/health", HealthCheckHandler).Methods("GET")
	log.Fatal(http.ListenAndServe(":80", router))
}
```

view raw main.go hosted with ❤ by **GitHub**

Source Code

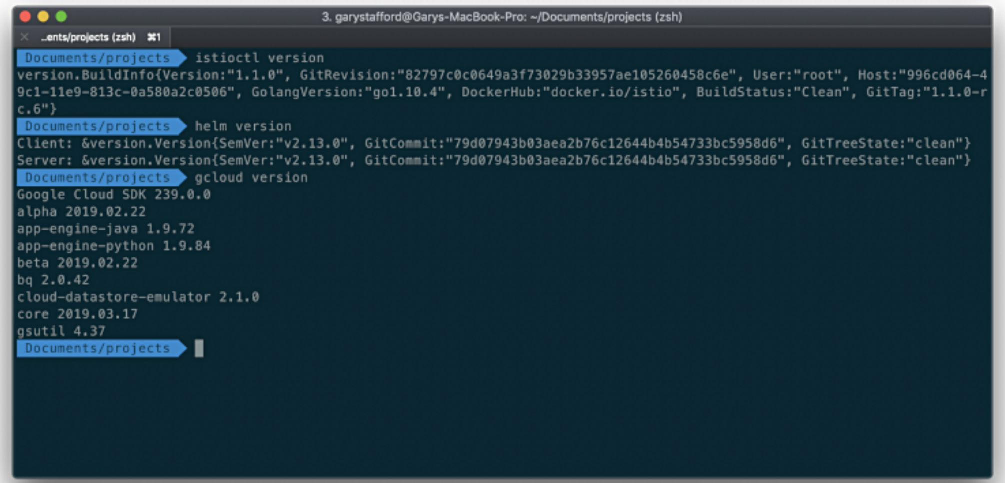
All source code for this post is available on GitHub in two projects. The Go-based microservices source code, all Kubernetes resources, and all deployment scripts are located in the k8s-istio-observe-backend (https://github.com/garystafford/k8s-istio-observe-backend) project repository. The Angular UI TypeScript-based (https://en.wikipedia.org/wiki/TypeScript) source code is located in the k8s-istio-observe-frontend (https://github.com/garystafford/k8s-istio-observe-frontend) project repository. You should not need to clone the Angular UI project for this demonstration.

```
git clone --branch master --single-branch --depth 1 --no-tags \
  https://github.com/garystafford/k8s-istio-observe-backend.git
```

Docker images referenced in the Kubernetes `Deployment` resource files, for the Go services and UI, are all available on Docker Hub (https://hub.docker.com/u/garystafford/). The Go microservice Docker images were built using the official Golang Alpine (https://hub.docker.com/_/golang) base image on DockerHub, containing Go version 1.12.0. Using the Alpine image to compile the Go source code ensures the containers will be as small as possible and contain a minimal attack surface.

System Requirements

To follow along with the post, you will need the latest version of `gcloud` CLI (min. ver. 239.0.0), part of the Google Cloud SDK, Helm (https://helm.sh/), and the just releases Istio 1.1.0 (https://github.com/istio/istio/releases/tag/1.1.0), installed and configured locally or on your build machine.


3. garystafford@Garys-MacBook-Pro: ~/Documents/projects (zsh)
…ents/projects (zsh) ⌘1

```
Documents/projects  istioctl version
version.BuildInfo{Version:"1.1.0", GitRevision:"82797c0c0649a3f73029b33957ae105260458c6e", User:"root", Host:"996cd064-4
9c1-11e9-813c-0a580a2c0506", GolangVersion:"go1.10.4", DockerHub:"docker.io/istio", BuildStatus:"Clean", GitTag:"1.1.0-r
c.6"}
Documents/projects  helm version
Client: &version.Version{SemVer:"v2.13.0", GitCommit:"79d07943b03aea2b76c12644b4b54733bc5958d6", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.13.0", GitCommit:"79d07943b03aea2b76c12644b4b54733bc5958d6", GitTreeState:"clean"}
Documents/projects  gcloud version
Google Cloud SDK 239.0.0
alpha 2019.02.22
app-engine-java 1.9.72
app-engine-python 1.9.84
beta 2019.02.22
bq 2.0.42
cloud-datastore-emulator 2.1.0
core 2019.03.17
gsutil 4.37
Documents/projects 
```

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-19_at_9_23_17_pm.png)

Set-up and Installation

To deploy the microservices platform to GKE, we will proceed in the following order.

1. Create the MongoDB Atlas database cluster;
2. Create the CloudAMQP RabbitMQ cluster;
3. Modify the Kubernetes resources and scripts for your own environments;
4. Create the GKE cluster on GCP;
5. Deploy Istio 1.1.0 to the GKE cluster, using Helm;
6. Create DNS records for the platform's exposed resources;
7. Deploy the Go-based microservices, Angular UI, and associated resources to GKE;
8. Test and troubleshoot the platform;

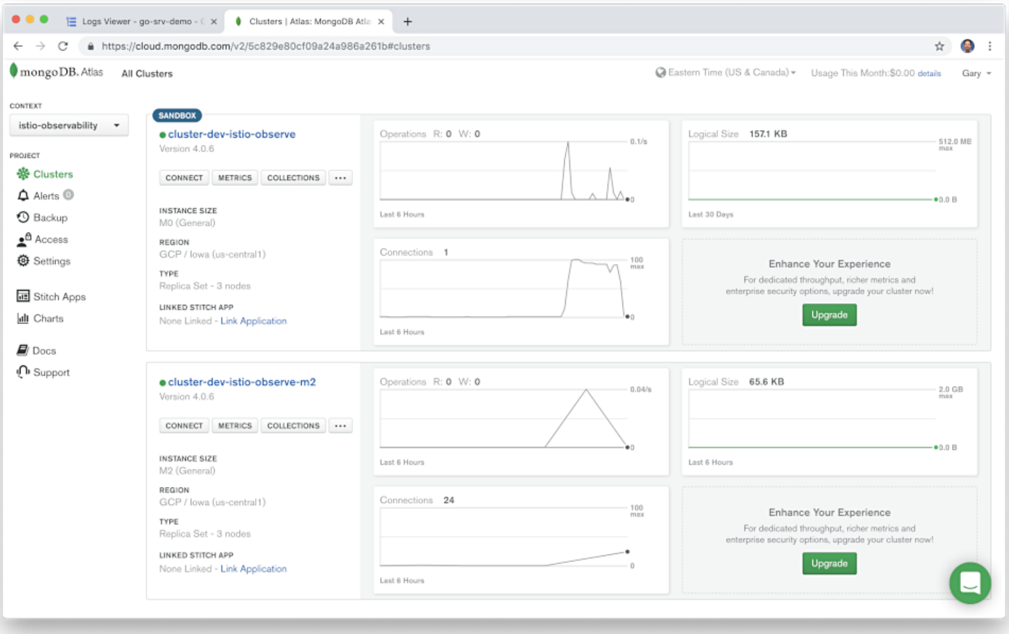
9. Observe the results in part two!

MongoDB Atlas Cluster

MongoDB Atlas (https://www.mongodb.com/cloud/atlas) is a fully-managed MongoDB-as-a-Service, available on AWS, Azure, and GCP. Atlas, a mature SaaS product, offers high-availability, guaranteed uptime SLAs, elastic scalability, cross-region replication, enterprise-grade security, LDAP integration, a BI Connector, and much more.

MongoDB Atlas currently offers four pricing plans (https://www.mongodb.com/cloud/atlas/pricing), Free, Basic, Pro, and Enterprise. Plans range from the smallest, M0-sized MongoDB cluster, with shared RAM and 512 MB storage, up to the massive M400 MongoDB cluster, with 488 GB of RAM and 3 TB of storage.

For this post, I have created an M2-sized MongoDB cluster in GCP's us-central1 (Iowa) region, with a single user database account for this demo. The account will be used to connect from four of the eight microservices, running on GKE.



https://cloud.mongodb.com/v2/5c829e80cf09a24a986a261b#clusters

All Clusters

Eastern Time (US & Canada) Usage This Month:$0.00 details Gary

CONTEXT istio-observability

SANDBOX

cluster-dev-istio-observe
Version 4.0.6

INSTANCE SIZE
M0 (General)

REGION
GCP / Iowa (us-central1)

TYPE
Replica Set - 3 nodes

LINKED STITCH APP
None Linked - Link Application

Operations R: 0 W: 0 — Last 6 Hours
Logical Size 157.1 KB — 512.0 MB max — Last 30 Days
Connections 1 — 100 max — Last 6 Hours

cluster-dev-istio-observe-m2
Version 4.0.6

INSTANCE SIZE
M2 (General)

REGION
GCP / Iowa (us-central1)

TYPE
Replica Set - 3 nodes

LINKED STITCH APP
None Linked - Link Application

Operations R: 0 W: 0 — Last 6 Hours
Logical Size 65.6 KB — 2.0 GB max — Last 6 Hours
Connections 24 — 100 max — Last 6 Hours

Enhance Your Experience
For dedicated throughput, richer metrics and enterprise security options, upgrade your cluster now!

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_7_48_00_pm.png)

Originally, I started with an M0-sized cluster, but the compute resources were insufficient to support the volume of calls from the Go-based microservices. I suggest at least an M2-sized cluster or larger.

CloudAMQP RabbitMQ Cluster

CloudAMQP (https://www.cloudamqp.com/) provides full-managed RabbitMQ clusters on all major cloud and application platforms. RabbitMQ will support a decoupled, eventually consistent, message-based architecture for a portion of our Go-based microservices. For this post, I have created a RabbitMQ cluster in GCP's us-central1 (Iowa) region, the same as our GKE cluster and MongoDB Atlas cluster. I chose a minimally-configured free version of RabbitMQ. CloudAMQP also offers robust, multi-node RabbitMQ clusters for Production use.

Modify Configurations

There are a few configuration settings you will need to change in the GitHub project's Kubernetes resource files and Bash deployment scripts.

Istio ServiceEntry for MongoDB Atlas

Modify the Istio `ServiceEntry`, external-mesh-mongodb-atlas.yaml (https://github.com/garystafford/golang-srv-demo/blob/master/resources/other/external-mesh-mongodb-atlas.yaml) file, adding you MongoDB Atlas host address. This file allows egress traffic from four of the microservices on GKE to the external MongoDB Atlas cluster.

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: mongodb-atlas-external-mesh
spec:
  hosts:
  - {{ your_host_goes_here }}
  ports:
  - name: mongo
    number: 27017
    protocol: MONGO
  location: MESH_EXTERNAL
  resolution: NONE
```

Istio ServiceEntry for CloudAMQP RabbitMQ

Modify the Istio `ServiceEntry`, external-mesh-cloudamqp.yaml (https://github.com/garystafford/golang-srv-demo/blob/master/resources/other/external-mesh-cloudamqp.yaml) file, adding you CloudAMQP host address. This file allows egress traffic from two of the microservices to the CloudAMQP cluster.

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: cloudamqp-external-mesh
spec:
  hosts:
  - {{ your_host_goes_here }}
  ports:
  - name: rabbitmq
    number: 5672
    protocol: TCP
  location: MESH_EXTERNAL
  resolution: NONE
```

Istio Gateway and VirtualService Resources

There are numerous strategies you may use to route traffic into the GKE cluster, via Istio. I am using a single domain for the post, `example-api.com`, and four subdomains. One set of subdomains is for the Angular UI, in the `dev` Namespace (`ui.dev.example-api.com`) and the `test` Namespace (`ui.test.example-api.com`). The other set of subdomains is for the edge API microservice, Service A, which the UI calls (`api.dev.example-api.com` and `api.test.example-api.com`). Traffic is routed to specific Kubernetes `Service` resources, based on the URL.

According to Istio (https://istio.io/docs/reference/config/istio.networking.v1alpha3/#Gateway), the `Gateway` describes a load balancer operating at the edge of the mesh, receiving incoming or outgoing HTTP/TCP connections. Modify the Istio ingress `Gateway`, inserting your own domains or subdomains in the `hosts` section. These are the hosts on port 80 that will be allowed into the mesh.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: demo-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - ui.dev.example-api.com
    - ui.test.example-api.com
    - api.dev.example-api.com
    - api.test.example-api.com
```

According to Istio (https://istio.io/docs/reference/config/istio.networking.v1alpha3/#VirtualService), a `VirtualService` defines a set of traffic routing rules to apply when a host is addressed. A `VirtualService` is bound to a `Gateway` to control the forwarding of traffic arriving at a particular host and port. Modify the project's four Istio `VirtualServices`, inserting your own domains or subdomains. Here is an example of one of the four `VirtualServices`, in the istio-gateway.yaml (https://github.com/garystafford/golang-srv-demo/blob/master/resources/other/istio-gateway.yaml) file.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: angular-ui-dev
spec:
  hosts:
  - ui.dev.example-api.com
  gateways:
  - demo-gateway
  http:
  - match:
    - uri:
        prefix: /
    route:
    - destination:
        port:
          number: 80
        host: angular-ui.dev.svc.cluster.local
```

Kubernetes Secret

The project contains a Kubernetes `Secret`, go-srv-demo.yaml (https://github.com/garystafford/golang-srv-demo/blob/master/resources/secrets/go-srv-demo.yaml), with two values. One is for the MongoDB Atlas connection string and one is for the CloudAMQP connections string. Remember Kubernetes `Secret` values need to be `base64` encoded.

```
apiVersion: v1
kind: Secret
metadata:
  name: go-srv-config
type: Opaque
data:
  mongodb.conn: {{ your_base64_encoded_secret }}
  rabbitmq.conn: {{ your_base64_encoded_secret }}
```

On Linux and Mac, you can use the `base64` program to encode the connection strings.

```
> echo -n "mongodb+srv://username:password@atlas-cluster.gcp.mongodb.net/t
bW9uZ29kYitzcnY6Ly91c2VybmFtZTpwYXNzd29yZEBhdGxhcy1jbHVzdGVyLmdjcC5tb25nb2

> echo -n "amqp://username:password@rmq.cloudamqp.com/cluster" | base64
YW1xcDovL3VzZXJuYW1lOnBhc3N3b3JkQHJtcS5jbG91ZGFtcXAuY29tL2NsdXN0ZXI=
```

Bash Scripts Variables

The bash script, part3_create_gke_cluster.sh, (https://github.com/garystafford/golang-srv-demo/blob/master/part3_create_gke_cluster.sh)contains a series of environment variables. At a minimum, you will need to change the PROJECT variable in all scripts to match your GCP project name.

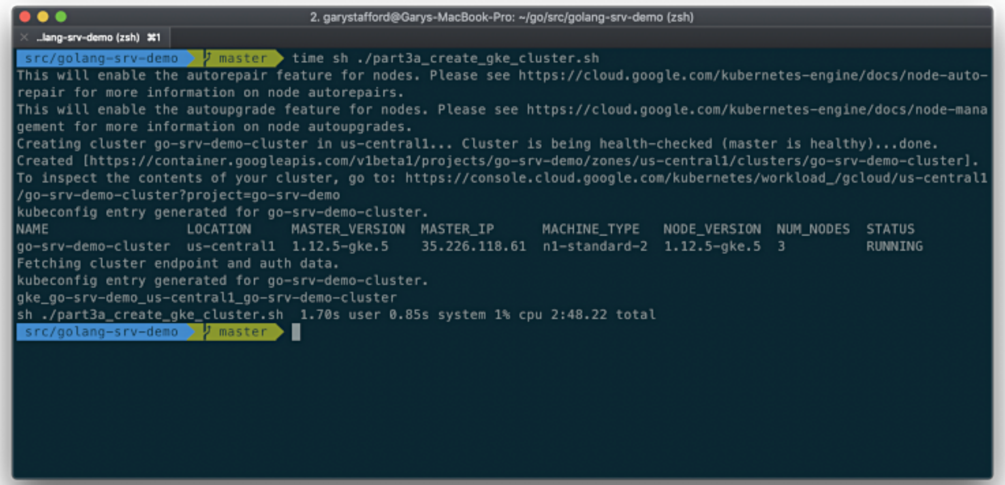
```
# Constants - CHANGE ME!
readonly PROJECT='{{ your_gcp_project_goes_here }}'
readonly CLUSTER='go-srv-demo-cluster'
readonly REGION='us-central1'
readonly MASTER_AUTH_NETS='72.231.208.0/24'
readonly GKE_VERSION='1.12.5-gke.5'
readonly MACHINE_TYPE='n1-standard-2'
```

The bash script, part4_install_istio.sh (https://github.com/garystafford/golang-srv-demo/blob/master/part4_install_istio.sh), includes the ISTIO_HOME variable. The value should correspond to your local path to Istio 1.1.0. On my local Mac, this value is shown below.

```
readonly ISTIO_HOME='/Applications/istio-1.1.0'
```

Deploy GKE Cluster

Next, deploy the GKE cluster using the included bash script, part3_create_gke_cluster.sh (https://github.com/garystafford/golang-srv-demo/blob/master/part3_create_gke_cluster.sh). This will create a Regional, multi-zone, 3-node GKE cluster, using the latest version of GKE at the time of this post, 1.12.5-gke.5. The cluster will be deployed to the same region as the MongoDB Atlas and CloudAMQP clusters, GCP's us-central1 (Iowa) region. Planning where your Cloud resources will reside, for both SaaS providers and primary Cloud providers can be critical to minimizing latency for network I/O intensive applications.

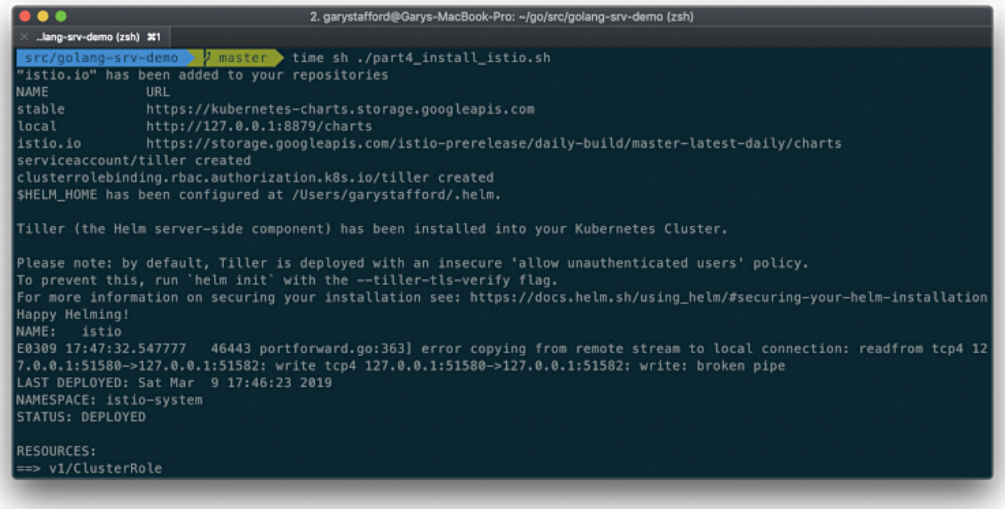


```
2. garystafford@Garys-MacBook-Pro: ~/go/src/golang-srv-demo (zsh)
..lang-srv-demo (zsh) ⌘1
src/golang-srv-demo  master  time sh ./part3a_create_gke_cluster.sh
This will enable the autorepair feature for nodes. Please see https://cloud.google.com/kubernetes-engine/docs/node-auto-
repair for more information on node autorepairs.
This will enable the autoupgrade feature for nodes. Please see https://cloud.google.com/kubernetes-engine/docs/node-mana
gement for more information on node autoupgrades.
Creating cluster go-srv-demo-cluster in us-central1... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1beta1/projects/go-srv-demo/zones/us-central1/clusters/go-srv-demo-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-central1
/go-srv-demo-cluster?project=go-srv-demo
kubeconfig entry generated for go-srv-demo-cluster.
NAME                 LOCATION     MASTER_VERSION  MASTER_IP      MACHINE_TYPE   NODE_VERSION  NUM_NODES  STATUS
go-srv-demo-cluster  us-central1  1.12.5-gke.5    35.226.118.61  n1-standard-2  1.12.5-gke.5  3          RUNNING
Fetching cluster endpoint and auth data.
kubeconfig entry generated for go-srv-demo-cluster.
gke_go-srv-demo_us-central1_go-srv-demo-cluster
sh ./part3a_create_gke_cluster.sh  1.70s user 0.85s system 1% cpu 2:48.22 total
src/golang-srv-demo  master
```

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_5_44_33_pm.png)

Deploy Istio using Helm

With the GKE cluster and associated infrastructure in place, deploy Istio. For this post, I have chosen to install Istio using Helm (https://istio.io/docs/setup/kubernetes/helm-install/), as recommended my Istio. To deploy Istio using Helm, use the included bash script, part4_install_istio.sh (https://github.com/garystafford/golang-srv-demo/blob/master/part4_install_istio.sh).



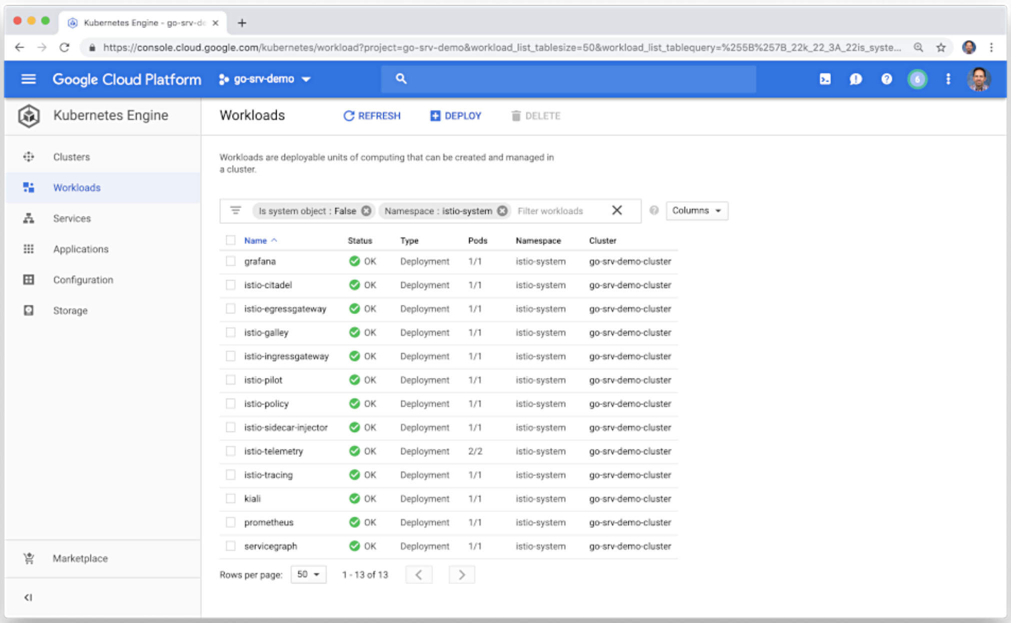
```
2. garystafford@Garys-MacBook-Pro: ~/go/src/golang-srv-demo (zsh)
..lang-srv-demo (zsh) ⌘1
src/golang-srv-demo  master  time sh ./part4_install_istio.sh
"istio.io" has been added to your repositories
NAME            URL
stable          https://kubernetes-charts.storage.googleapis.com
local           http://127.0.0.1:8879/charts
istio.io        https://storage.googleapis.com/istio-prerelease/daily-build/master-latest-daily/charts
serviceaccount/tiller created
clusterrolebinding.rbac.authorization.k8s.io/tiller created
$HELM_HOME has been configured at /Users/garystafford/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-your-helm-installation
Happy Helming!
NAME:   istio
E0309 17:47:32.547777   46443 portforward.go:363] error copying from remote stream to local connection: readfrom tcp4 12
7.0.0.1:51580->127.0.0.1:51582: write tcp4 127.0.0.1:51580->127.0.0.1:51582: write: broken pipe
LAST DEPLOYED: Sat Mar  9 17:46:23 2019
NAMESPACE: istio-system
STATUS: DEPLOYED

RESOURCES:
==> v1/ClusterRole
```

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_5_47_57_pm.png)

The script installs Istio, using the Helm Chart in the local Istio 1.1.0 `install/kubernetes/helm/istio` directory, which you installed as a requirement for this demonstration. The Istio install script overrides several default values in the Istio Helm Chart (https://github.com/istio/istio/tree/master/install/kubernetes/helm/istio) using the `--set`, flag. The list of available configuration values is detailed in the Istio Chart's GitHub project (https://github.com/istio/istio/tree/master/install/kubernetes/helm/istio#configuration). The options enable Istio's observability features, which we will explore in part two. Features include Kiali, Grafana, Prometheus, and Jaeger.

```
helm install ${ISTIO_HOME}/install/kubernetes/helm/istio-init \
  --name istio-init \
  --namespace istio-system

helm install ${ISTIO_HOME}/install/kubernetes/helm/istio \
  --name istio \
  --namespace istio-system \
  --set prometheus.enabled=true \
  --set grafana.enabled=true \
  --set kiali.enabled=true \
  --set tracing.enabled=true

kubectl apply --namespace istio-system \
  -f ./resources/secrets/kiali.yaml
```

Below, we see the Istio-related Workloads running on the cluster, including the observability tools.


Kubernetes Engine - go-srv-d
https://console.cloud.google.com/kubernetes/workload?project=go-srv-demo&workload_list_tablesize=50&workload_list_tablequery=%255B%257B_22k_22_3A_22is_syste...
Google Cloud Platform
go-srv-demo
Kubernetes Engine
Workloads
REFRESH
DEPLOY
DELETE
Clusters
Workloads
Services
Applications
Configuration
Storage
Marketplace

Workloads are deployable units of computing that can be created and managed in a cluster.

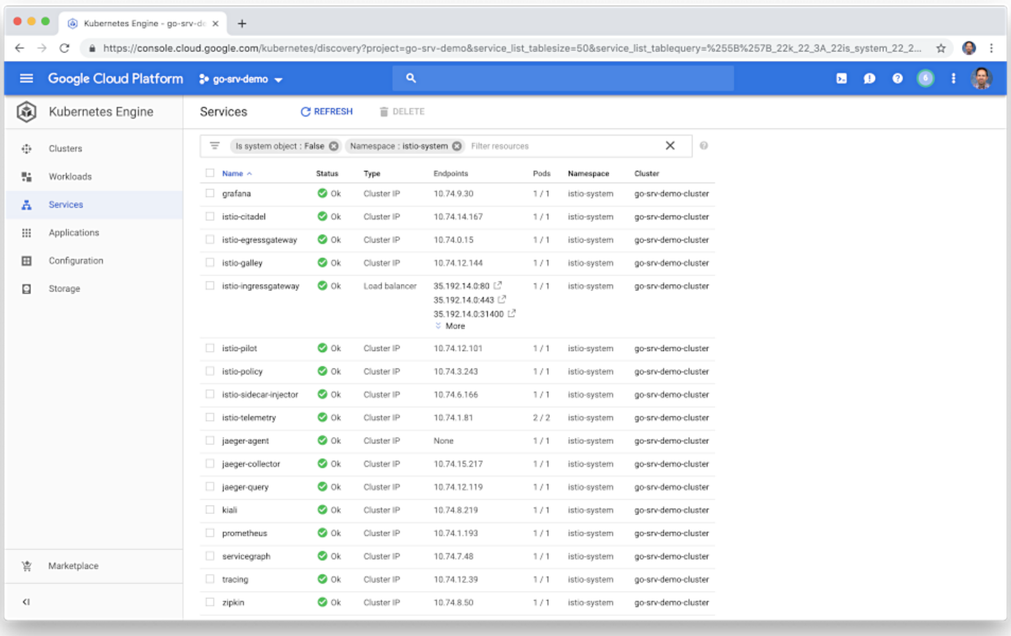
Is system object : False | Namespace : istio-system | Filter workloads | Columns

Name	Status	Type	Pods	Namespace	Cluster
grafana	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-citadel	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-egressgateway	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-galley	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-ingressgateway	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-pilot	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-policy	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-sidecar-injector	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
istio-telemetry	✓ OK	Deployment	2/2	istio-system	go-srv-demo-cluster
istio-tracing	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
kiali	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
prometheus	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster
servicegraph	✓ OK	Deployment	1/1	istio-system	go-srv-demo-cluster

Rows per page: 50 1 - 13 of 13

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_5_58_35_pm.png)

Below, we see the corresponding Istio-related `Service` resources running on the cluster.



Services

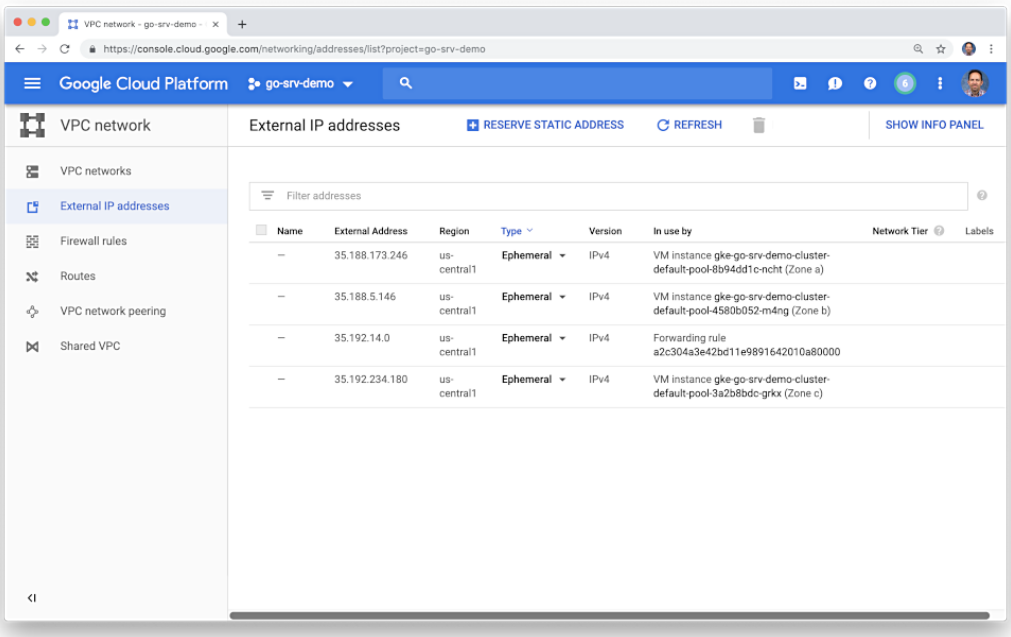
Name	Status	Type	Endpoints	Pods	Namespace	Cluster
grafana	✓ Ok	Cluster IP	10.74.9.30	1 / 1	istio-system	go-srv-demo-cluster
istio-citadel	✓ Ok	Cluster IP	10.74.14.167	1 / 1	istio-system	go-srv-demo-cluster
istio-egressgateway	✓ Ok	Cluster IP	10.74.0.15	1 / 1	istio-system	go-srv-demo-cluster
istio-galley	✓ Ok	Cluster IP	10.74.12.144	1 / 1	istio-system	go-srv-demo-cluster
istio-ingressgateway	✓ Ok	Load balancer	35.192.14.0:80 35.192.14.0:443 35.192.14.0:31400 More	1 / 1	istio-system	go-srv-demo-cluster
istio-pilot	✓ Ok	Cluster IP	10.74.12.101	1 / 1	istio-system	go-srv-demo-cluster
istio-policy	✓ Ok	Cluster IP	10.74.3.243	1 / 1	istio-system	go-srv-demo-cluster
istio-sidecar-injector	✓ Ok	Cluster IP	10.74.6.166	1 / 1	istio-system	go-srv-demo-cluster
istio-telemetry	✓ Ok	Cluster IP	10.74.1.81	2 / 2	istio-system	go-srv-demo-cluster
jaeger-agent	✓ Ok	Cluster IP	None	1 / 1	istio-system	go-srv-demo-cluster
jaeger-collector	✓ Ok	Cluster IP	10.74.15.217	1 / 1	istio-system	go-srv-demo-cluster
jaeger-query	✓ Ok	Cluster IP	10.74.12.119	1 / 1	istio-system	go-srv-demo-cluster
kiali	✓ Ok	Cluster IP	10.74.8.219	1 / 1	istio-system	go-srv-demo-cluster
prometheus	✓ Ok	Cluster IP	10.74.1.193	1 / 1	istio-system	go-srv-demo-cluster
servicegraph	✓ Ok	Cluster IP	10.74.7.48	1 / 1	istio-system	go-srv-demo-cluster
tracing	✓ Ok	Cluster IP	10.74.12.39	1 / 1	istio-system	go-srv-demo-cluster
zipkin	✓ Ok	Cluster IP	10.74.8.50	1 / 1	istio-system	go-srv-demo-cluster

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_5_59_14_pm.png)

Modify DNS Records

Instead of using IP addresses to route traffic the GKE cluster and its applications, we will use DNS. As explained earlier, I have chosen a single domain for the post, `example-api.com`, and four subdomains. One set of subdomains is for the Angular UI, in the `dev` Namespace and the `test` Namespace. The other set of subdomains is for the edge microservice, Service A, which the API calls. Traffic is routed to specific Kubernetes `Service` resources, based on the URL.

Deploying the GKE cluster and Istio triggers the creation of a Google Load Balancer (https://cloud.google.com/load-balancing/), four IP addresses, and all required firewall rules. One of the four IP addresses, the one shown below, associated with the Forwarding rule, will be associated with the front-end of the load balancer.

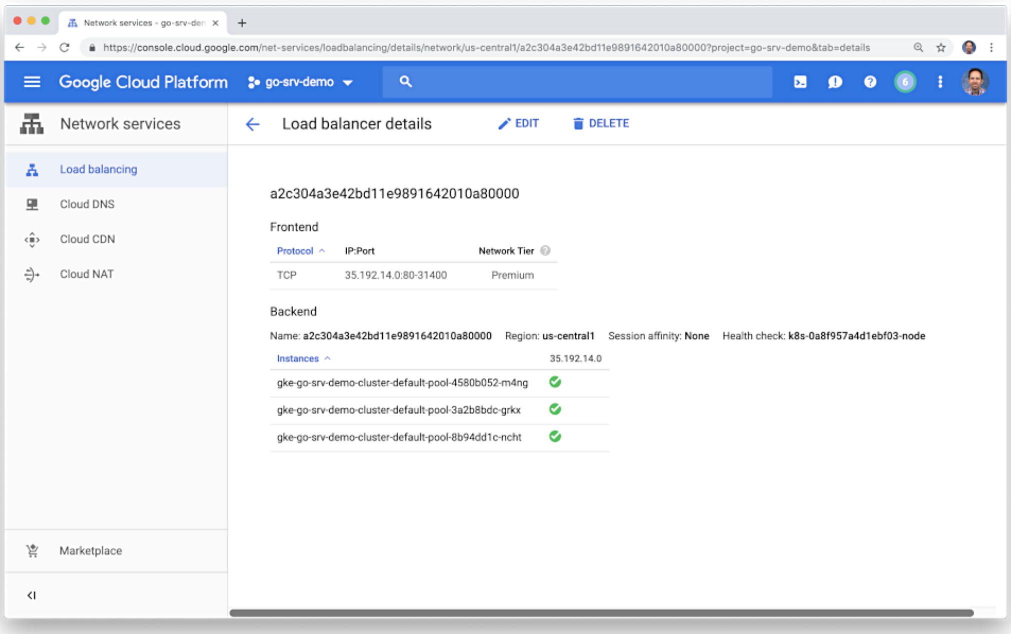


External IP addresses

Name	External Address	Region	Type	Version	In use by	Network Tier	Labels
–	35.188.173.246	us-central1	Ephemeral	IPv4	VM instance gke-go-srv-demo-cluster-default-pool-8b94dd1c-ncht (Zone a)		
–	35.188.5.146	us-central1	Ephemeral	IPv4	VM instance gke-go-srv-demo-cluster-default-pool-4580b052-m4ng (Zone b)		
–	35.192.14.0	us-central1	Ephemeral	IPv4	Forwarding rule a2c304a3e42bd11e9891642010a80000		
–	35.192.234.180	us-central1	Ephemeral	IPv4	VM instance gke-go-srv-demo-cluster-default-pool-3a2b8bdc-grkx (Zone c)		

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_5_49_37_pm.png)

Below, we see the new load balancer, with the front-end IP address and the backend VM pool of three GKE cluster's worker nodes. Each node is assigned one of the IP addresses, as shown above.


Network services - go-srv-dem
https://console.cloud.google.com/net-services/loadbalancing/details/network/us-central1/a2c304a3e42bd11e9891642010a80000?project=go-srv-demo&tab=details
Google Cloud Platform
go-srv-demo
Network services
Load balancing
Cloud DNS
Cloud CDN
Cloud NAT
Marketplace
EDIT
DELETE

Load balancer details

a2c304a3e42bd11e9891642010a80000

Frontend

Protocol	IP:Port	Network Tier
TCP	35.192.14.0:80-31400	Premium

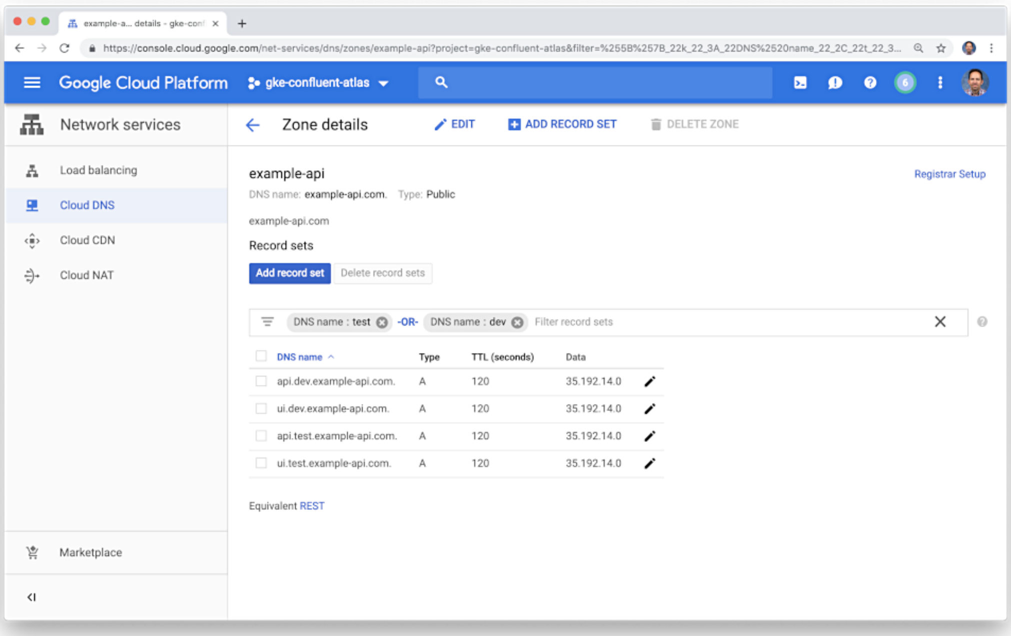
Backend

Name: a2c304a3e42bd11e9891642010a80000 Region: us-central1 Session affinity: None Health check: k8s-0a8f957a4d1ebf03-node

Instances	35.192.14.0
gke-go-srv-demo-cluster-default-pool-4580b052-m4ng	✓
gke-go-srv-demo-cluster-default-pool-3a2b8bdc-grkx	✓
gke-go-srv-demo-cluster-default-pool-8b94dd1c-ncht	✓

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_5_57_20_pm.png)

As shown below, using Google Cloud DNS (https://cloud.google.com/dns/), I have created the four subdomains and assigned the IP address of the load balancer's front-end to all four subdomains. Ingress traffic to these addresses will be routed through the Istio ingress `Gateway` and the four Istio `VirtualServices`, to the appropriate Kubernetes `Service` resources. Use your choice of DNS management tools to create the four A Type DNS records (https://en.wikipedia.org/wiki/List_of_DNS_record_types).



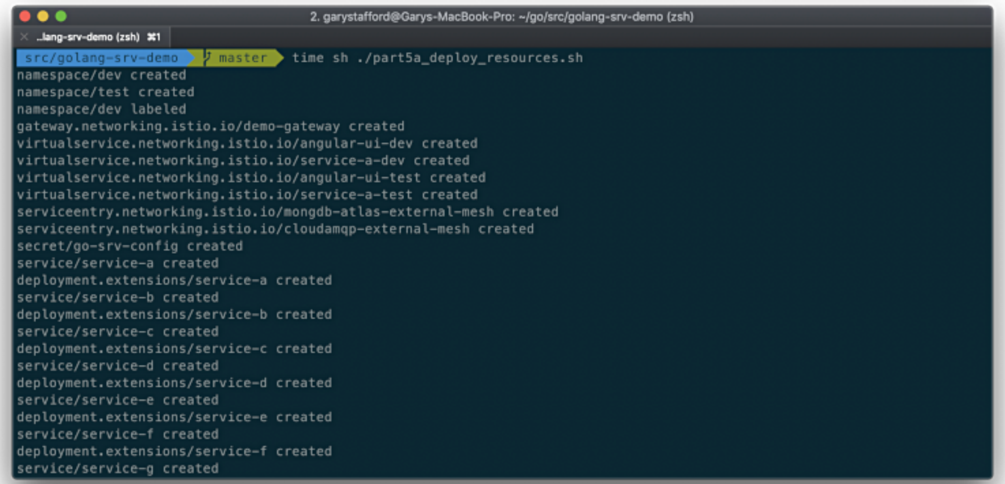
example-api
DNS name: example-api.com. Type: Public
example-api.com
Record sets

DNS name	Type	TTL (seconds)	Data
api.dev.example-api.com.	A	120	35.192.14.0
ui.dev.example-api.com.	A	120	35.192.14.0
api.test.example-api.com.	A	120	35.192.14.0
ui.test.example-api.com.	A	120	35.192.14.0

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_5_56_29_pm.png)

Deploy the Reference Platform

Next, deploy the eight Go-based microservices, the Angular UI, and the associated Kubernetes and Istio resources to the GKE cluster. To deploy the platform, use the included bash deploy script, part5a_deploy_resources.sh (https://github.com/garystafford/golang-srv-demo/blob/master/part5a_deploy_resources.sh). If anything fails and you want to remove the existing resources and re-deploy, without destroying the GKE cluster or Istio, you can use the part5b_delete_resources.sh (https://github.com/garystafford/golang-srv-demo/blob/master/part5b_delete_resources.sh) delete script.

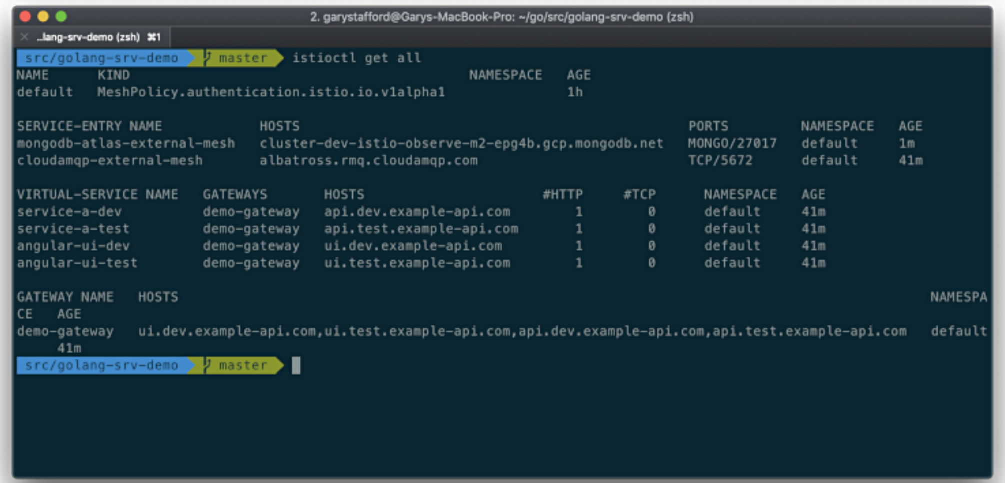

2. garystafford@Garys-MacBook-Pro: ~/go/src/golang-srv-demo (zsh)
..lang-srv-demo (zsh) ⌘1

```
src/golang-srv-demo  master  time sh ./part5a_deploy_resources.sh
namespace/dev created
namespace/test created
namespace/dev labeled
gateway.networking.istio.io/demo-gateway created
virtualservice.networking.istio.io/angular-ui-dev created
virtualservice.networking.istio.io/service-a-dev created
virtualservice.networking.istio.io/angular-ui-test created
virtualservice.networking.istio.io/service-a-test created
serviceentry.networking.istio.io/mongdb-atlas-external-mesh created
serviceentry.networking.istio.io/cloudamqp-external-mesh created
secret/go-srv-config created
service/service-a created
deployment.extensions/service-a created
service/service-b created
deployment.extensions/service-b created
service/service-c created
deployment.extensions/service-c created
service/service-d created
deployment.extensions/service-d created
service/service-e created
deployment.extensions/service-e created
service/service-f created
deployment.extensions/service-f created
service/service-g created
```

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_6_01_29_pm.png)

The deploy script deploys all the resources two Kubernetes Namespaces, `dev` and `test`. This will allow us to see how we can differentiate between Namespaces when using the observability tools.

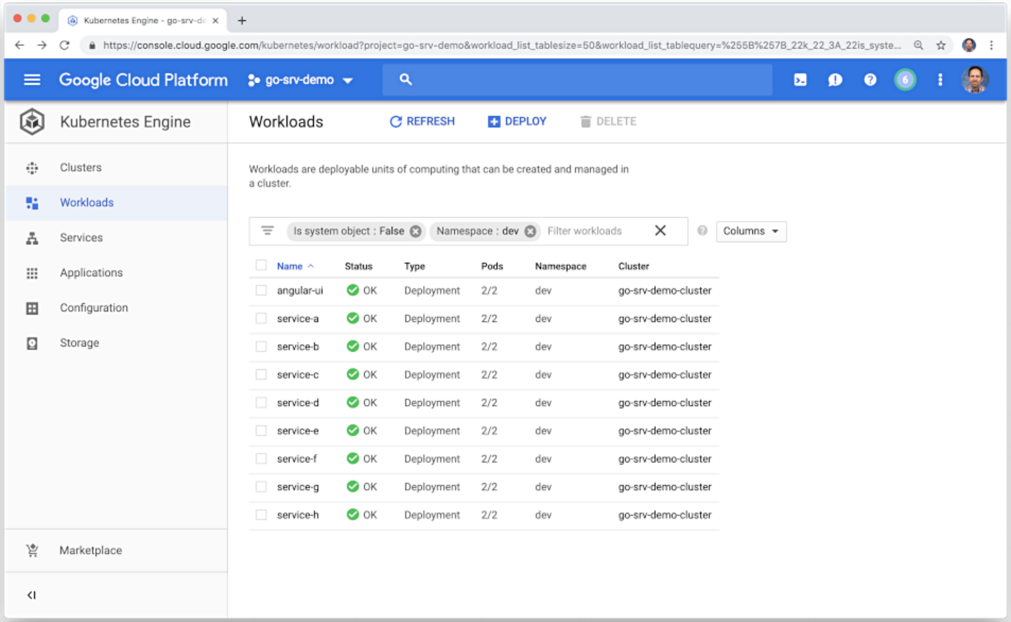
Below, we see the Istio-related resources, which we just deployed. They include the Istio `Gateway`, four Istio `VirtualService`, and two Istio `ServiceEntry` resources.



```
src/golang-srv-demo  master  istioctl get all
NAME      KIND                                        NAMESPACE   AGE
default   MeshPolicy.authentication.istio.io.v1alpha1             1h

SERVICE-ENTRY NAME           HOSTS                                              PORTS         NAMESPACE   AGE
mongodb-atlas-external-mesh  cluster-dev-istio-observe-m2-epg4b.gcp.mongodb.net MONGO/27017   default     1m
cloudamqp-external-mesh      albatross.rmq.cloudamqp.com                        TCP/5672      default     41m

VIRTUAL-SERVICE NAME   GATEWAYS       HOSTS                        #HTTP     #TCP      NAMESPACE   AGE
service-a-dev          demo-gateway   api.dev.example-api.com          1        0      default     41m
service-a-test         demo-gateway   api.test.example-api.com         1        0      default     41m
angular-ui-dev         demo-gateway   ui.dev.example-api.com           1        0      default     41m
angular-ui-test        demo-gateway   ui.test.example-api.com          1        0      default     41m

GATEWAY NAME   HOSTS                                                                                    NAMESPA
CE   AGE
demo-gateway   ui.dev.example-api.com,ui.test.example-api.com,api.dev.example-api.com,api.test.example-api.com   default
     41m
src/golang-srv-demo  master
```

[(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-10_at_10_48_49_pm.png)](https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-10_at_10_48_49_pm.png)

Below, we see the platform's Workloads (Kubernetes `Deployment` resources), running on the cluster. Here we see two Pods for each Workload, a total of 18 Pods, running in the `dev` Namespace. Each Pod contains both the deployed microservice or UI component, as well as a copy of Istio's [Envoy Proxy (https://istio.io/docs/concepts/what-is-istio/#envoy)](https://istio.io/docs/concepts/what-is-istio/#envoy).

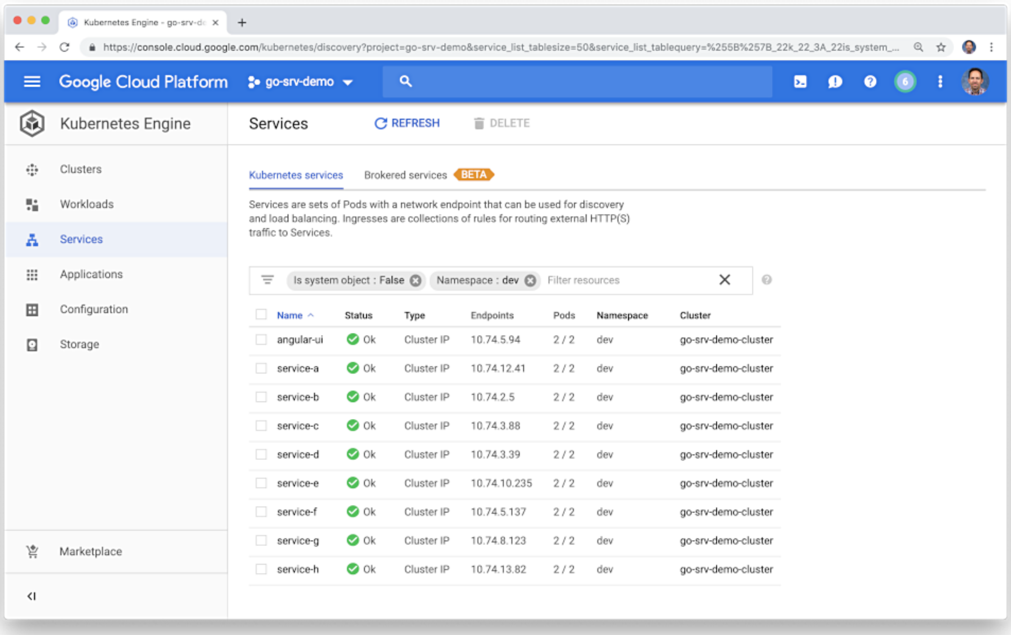


Workloads are deployable units of computing that can be created and managed in a cluster.

Name	Status	Type	Pods	Namespace	Cluster
angular-ui	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-a	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-b	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-c	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-d	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-e	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-f	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-g	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster
service-h	✓ OK	Deployment	2/2	dev	go-srv-demo-cluster

[(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_6_12_59_pm.png)](https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_6_12_59_pm.png)

Below, we see the corresponding Kubernetes `Service` resources running in the `dev` Namespace.

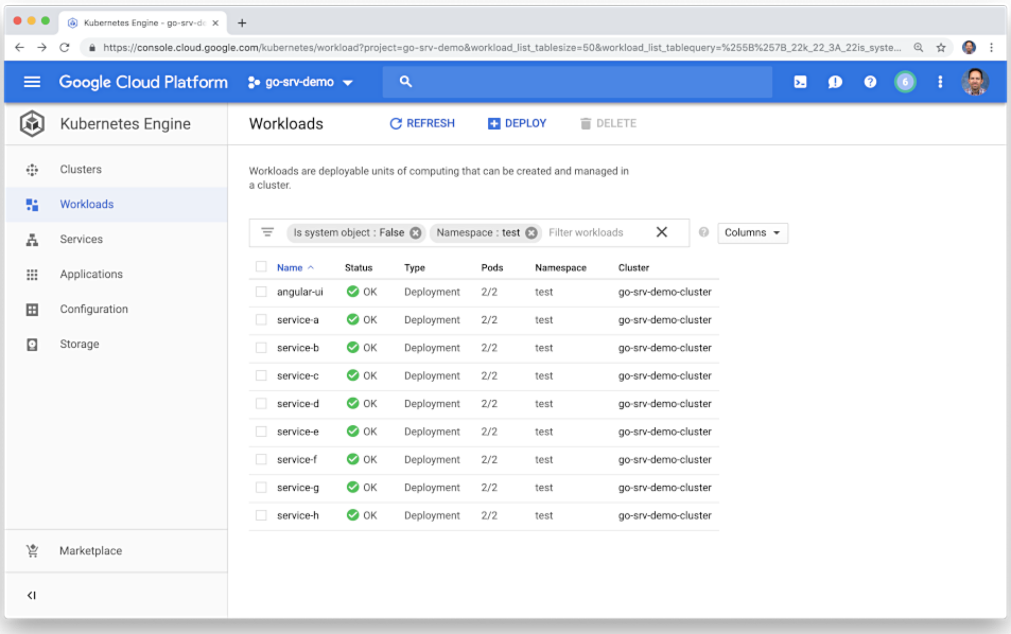


Services are sets of Pods with a network endpoint that can be used for discovery and load balancing. Ingresses are collections of rules for routing external HTTP(S) traffic to Services.

Name	Status	Type	Endpoints	Pods	Namespace	Cluster
angular-ui	✓ Ok	Cluster IP	10.74.5.94	2 / 2	dev	go-srv-demo-cluster
service-a	✓ Ok	Cluster IP	10.74.12.41	2 / 2	dev	go-srv-demo-cluster
service-b	✓ Ok	Cluster IP	10.74.2.5	2 / 2	dev	go-srv-demo-cluster
service-c	✓ Ok	Cluster IP	10.74.3.88	2 / 2	dev	go-srv-demo-cluster
service-d	✓ Ok	Cluster IP	10.74.3.39	2 / 2	dev	go-srv-demo-cluster
service-e	✓ Ok	Cluster IP	10.74.10.235	2 / 2	dev	go-srv-demo-cluster
service-f	✓ Ok	Cluster IP	10.74.5.137	2 / 2	dev	go-srv-demo-cluster
service-g	✓ Ok	Cluster IP	10.74.8.123	2 / 2	dev	go-srv-demo-cluster
service-h	✓ Ok	Cluster IP	10.74.13.82	2 / 2	dev	go-srv-demo-cluster

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_6_03_02_pm.png)

Below, a similar view of the `Deployment` resources running in the `test` Namespace. Again, we have two Pods for each deployment with each Pod contains both the deployed microservice or UI component, as well as a copy of Istio's Envoy Proxy (https://istio.io/docs/concepts/what-is-istio/#envoy).



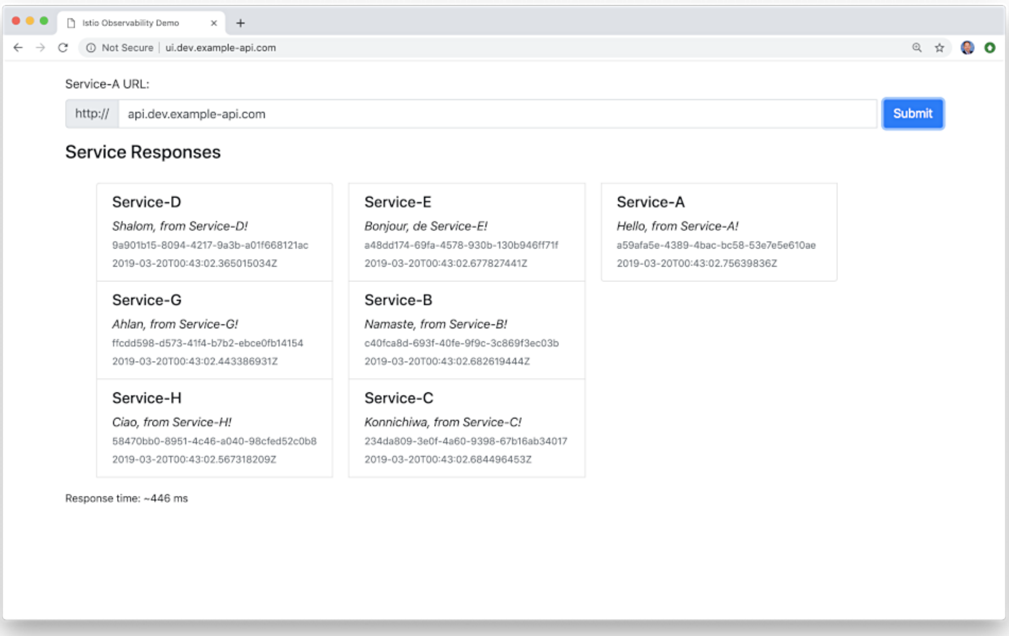
Workloads are deployable units of computing that can be created and managed in a cluster.

Name	Status	Type	Pods	Namespace	Cluster
angular-ui	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-a	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-b	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-c	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-d	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-e	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-f	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-g	✓ OK	Deployment	2/2	test	go-srv-demo-cluster
service-h	✓ OK	Deployment	2/2	test	go-srv-demo-cluster

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_6_13_16_pm.png)

Test the Platform

We do want to ensure the platform's eight Go-based microservices and Angular UI are working properly, communicating with each other, and communicating with the external MongoDB Atlas and CloudAMQP RabbitMQ clusters. The easiest way to test the cluster is by viewing the Angular UI in a web browser.



Service-A URL:

http:// api.dev.example-api.com

Service Responses

Service-D
Shalom, from Service-D!
9a901b15-8094-4217-9a3b-a01f668121ac
2019-03-20T00:43:02.365015034Z

Service-E
Bonjour, de Service-E!
a48dd174-69fa-4578-930b-130b946ff71f
2019-03-20T00:43:02.677827441Z

Service-A
Hello, from Service-A!
a59afa5e-4389-4bac-bc58-53e7e5e610ae
2019-03-20T00:43:02.75639836Z

Service-G
Ahlan, from Service-G!
ffcdd598-d573-41f4-b7b2-ebce0fb14154
2019-03-20T00:43:02.443386931Z

Service-B
Namaste, from Service-B!
c40fca8d-693f-40fe-9f9c-3c869f3ec03b
2019-03-20T00:43:02.682619444Z

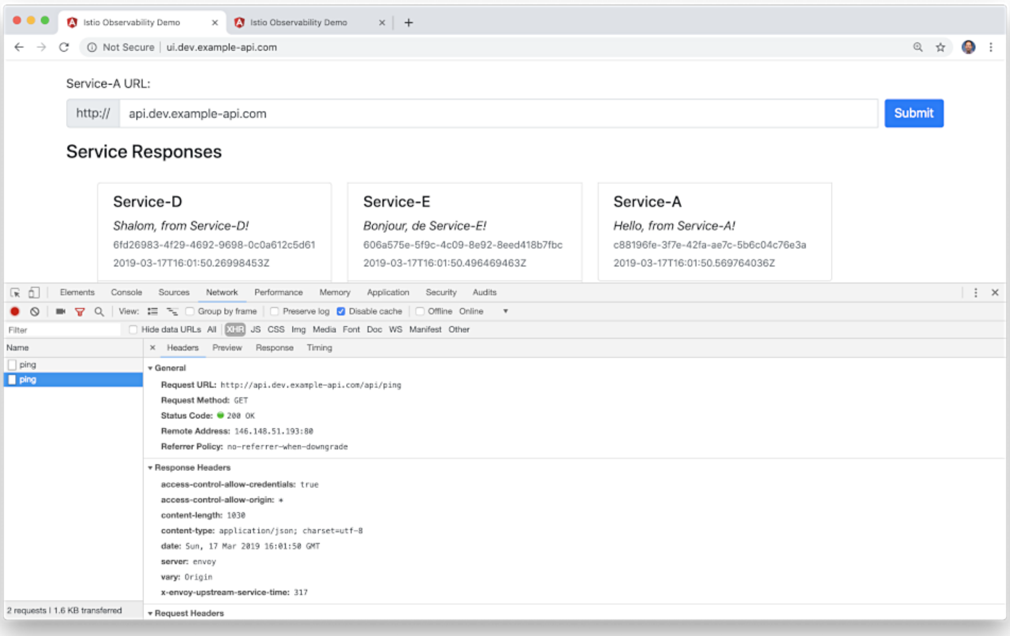
Service-H
Ciao, from Service-H!
58470bb0-8951-4c46-a040-98cfed52c0b8
2019-03-20T00:43:02.567318209Z

Service-C
Konnichiwa, from Service-C!
234da809-3e0f-4a60-9398-67b16ab34017
2019-03-20T00:43:02.684496453Z

Response time: ~446 ms

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-19_at_8_43_10_pm.png)

The UI requires you to input the host domain of the Service A, the API's edge service. Since you cannot use my subdomain, and the JavaScript code is running locally to your web browser, this option allows you to provide your own host domain. This is the same domain or domains you inserted into the two Istio `VirtualService` for the UI. This domain route your API calls to either the FQDN (fully qualified domain name) of the Service A Kubernetes Service running in the `dev` namespace, `service-a.dev.svc.cluster.local`, or the `test` Namespace, `service-a.test.svc.cluster.local`.


Istio Observability Demo
Istio Observability Demo
Not Secure | ui.dev.example-api.com
Submit
Elements Console Sources Network Performance Memory Application Security Audits
View: Group by frame Preserve log Disable cache Offline Online
Filter Hide data URLs All XHR JS CSS Img Media Font Doc WS Manifest Other
Name
Headers Preview Response Timing

Service-A URL:

http:// api.dev.example-api.com

Service Responses

Service-D
Shalom, from Service-D!
6fd26983-4f29-4692-9698-0c0a612c5d61
2019-03-17T16:01:50.26998453Z

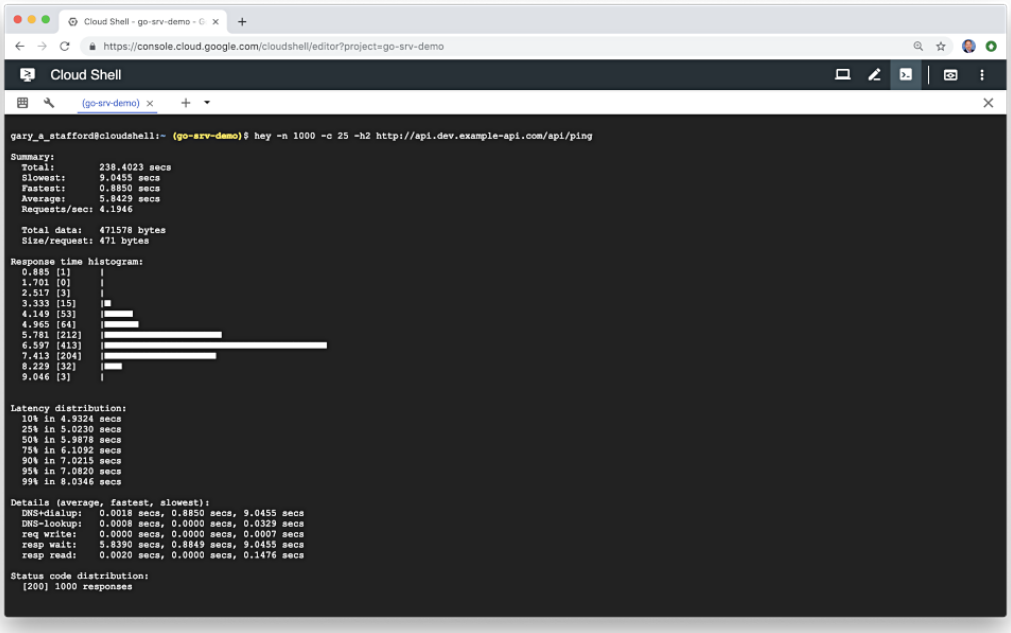
Service-E
Bonjour, de Service-E!
606a575e-5f9c-4c09-8e92-8eed418b7fbc
2019-03-17T16:01:50.496469463Z

Service-A
Hello, from Service-A!
c88196fe-3f7e-42fa-ae7c-5b6c04c76e3a
2019-03-17T16:01:50.569764036Z

```
ping
ping

▾ General
  Request URL: http://api.dev.example-api.com/api/ping
  Request Method: GET
  Status Code: ● 200 OK
  Remote Address: 146.148.51.193:80
  Referrer Policy: no-referrer-when-downgrade
▾ Response Headers
  access-control-allow-credentials: true
  access-control-allow-origin: *
  content-length: 1030
  content-type: application/json; charset=utf-8
  date: Sun, 17 Mar 2019 16:01:50 GMT
  server: envoy
  vary: Origin
  x-envoy-upstream-service-time: 317
▾ Request Headers

2 requests | 1.6 KB transferred
```

[(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-17_at_12_02_22_pm.png)](https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-17_at_12_02_22_pm.png)

You can also use performance testing tools to load-test the platform. Many issues will not show up until the platform is under load. I recently starting using [hey (https://github.com/rakyll/hey)](https://github.com/rakyll/hey), a modern load generator tool, as a replacement for Apache Bench (`ab`), Unlike `ab`, `hey` supports HTTP/2 endpoints, which is required to test the platform on GKE with Istio. Below, I am running `hey` directly from [Google Cloud Shell (https://cloud.google.com/shell/)](https://cloud.google.com/shell/). The tool is simulating 25 concurrent users, generating a total of 1,000 HTTP/2-based GET requests to Service A.


Cloud Shell - go-srv-demo - G
https://console.cloud.google.com/cloudshell/editor?project=go-srv-demo
Cloud Shell
(go-srv-demo)

```
gary_a_stafford@cloudshell:~ (go-srv-demo)$ hey -n 1000 -c 25 -h2 http://api.dev.example-api.com/api/ping

Summary:
  Total:        238.4023 secs
  Slowest:      9.0455 secs
  Fastest:      0.8850 secs
  Average:      5.8429 secs
  Requests/sec: 4.1946

  Total data:   471578 bytes
  Size/request: 471 bytes

Response time histogram:
  0.885 [1]     |
  1.701 [0]     |
  2.517 [3]     |
  3.333 [15]    |■
  4.149 [53]    |■■■■■
  4.965 [64]    |■■■■■■
  5.781 [212]   |■■■■■■■■■■■■■■■■■■■■■
  6.597 [413]   |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  7.413 [204]   |■■■■■■■■■■■■■■■■■■■■
  8.229 [32]    |■■■
  9.046 [3]     |


Latency distribution:
  10% in 4.9324 secs
  25% in 5.0230 secs
  50% in 5.9878 secs
  75% in 6.1092 secs
  90% in 7.0215 secs
  95% in 7.0820 secs
  99% in 8.0346 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0018 secs, 0.8850 secs, 9.0455 secs
  DNS-lookup:   0.0008 secs, 0.0000 secs, 0.0329 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0007 secs
  resp wait:    5.8390 secs, 0.8849 secs, 9.0455 secs
  resp read:    0.0020 secs, 0.0000 secs, 0.1476 secs

Status code distribution:
  [200] 1000 responses
```

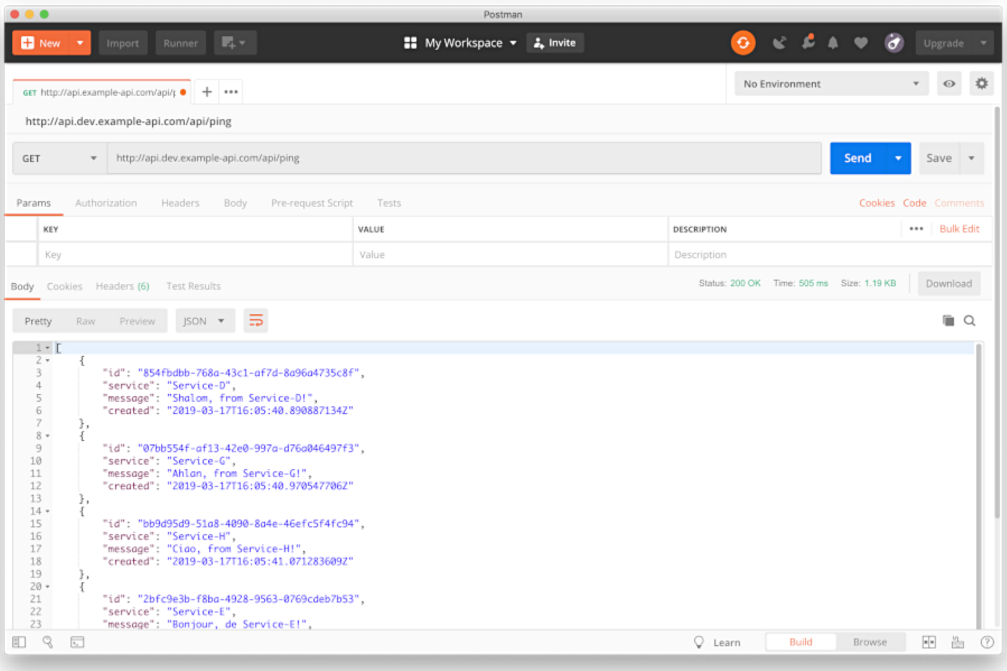
[(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-19_at_8_53_47_pm.png)](https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-19_at_8_53_47_pm.png)

Troubleshooting

If for some reason the UI fails to display, or the call from the UI to the API fails, and assuming all Kubernetes and Istio resources are running on the GKE cluster (all green), the most common explanation is usually a misconfiguration of the following resources:

1. Your four Cloud DNS records are not correct. They are not pointing to the load balancer's front-end IP address;
2. You did not configure the four Kubernetes `VirtualService` resources with the correct subdomains;
3. The GKE-based microservices cannot reach the external MongoDB Atlas and CloudAMQP RabbitMQ clusters. Likely, the Kubernetes `Secret` is constructed incorrectly, or the two `ServiceEntry` resources contain the wrong host information for those external clusters;

I suggest starting the troubleshooting by calling Service A, the API's edge service, directly, using cURL or Postman. You should see a JSON response payload, similar to the following. This suggests the issue is with the UI, not the API.



http://api.dev.example-api.com/api/ping

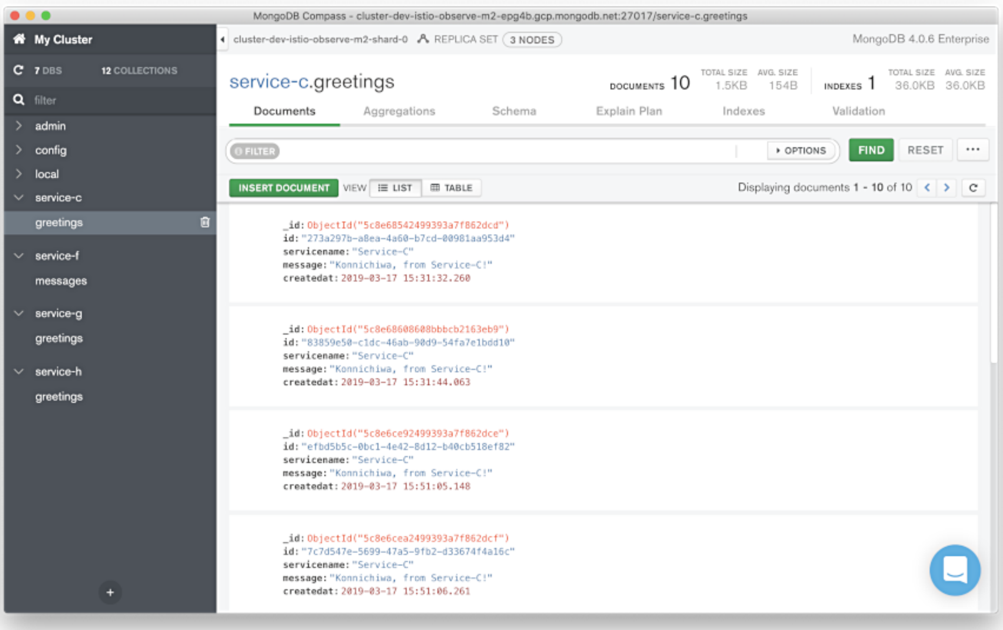
GET http://api.dev.example-api.com/api/ping

Status: 200 OK Time: 505 ms Size: 1.19 KB

```
[
    {
        "id": "854fbdbb-768a-43c1-af7d-8a96a4735c8f",
        "service": "Service-D",
        "message": "Shalom, from Service-D!",
        "created": "2019-03-17T16:05:40.890887134Z"
    },
    {
        "id": "07bb554f-af13-42e0-997a-d76a046497f3",
        "service": "Service-G",
        "message": "Ahlan, from Service-G!",
        "created": "2019-03-17T16:05:40.970547706Z"
    },
    {
        "id": "bb9d95d9-51a8-4090-8a4e-46efc5f4fc94",
        "service": "Service-H",
        "message": "Ciao, from Service-H!",
        "created": "2019-03-17T16:05:41.071283609Z"
    },
    {
        "id": "2bfc9e3b-f8ba-4928-9563-0769cdeb7b53",
        "service": "Service-E",
        "message": "Bonjour, de Service-E!",
```

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-17_at_12_06_27_pm.png)

Next, confirm that the four MongoDB databases were created for Service D, Service, F, Service, G, and Service H. Also, confirm that new documents are being written to the database's collections.


MongoDB Compass - cluster-dev-istio-observe-m2-epg4b.gcp.mongodb.net:27017/service-c.greetings

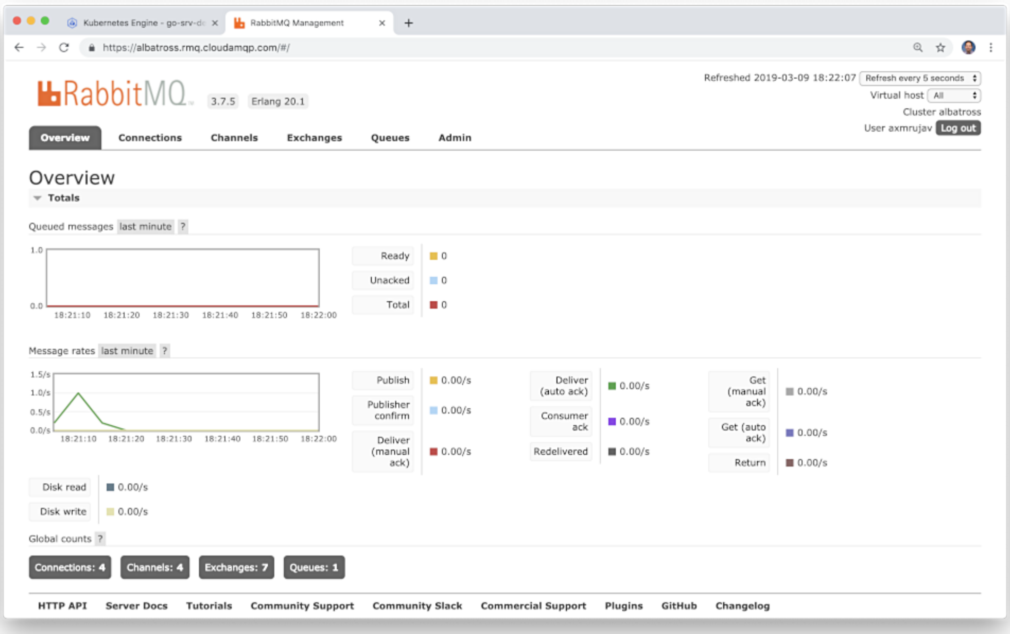
service-c.greetings — DOCUMENTS 10, TOTAL SIZE 1.5KB, AVG. SIZE 154B; INDEXES 1, TOTAL SIZE 36.0KB, AVG. SIZE 36.0KB

Displaying documents 1 - 10 of 10

```
_id: ObjectId("5c8e68542499393a7f862dcd")
id: "273a297b-a8ea-4a60-b7cd-00981aa953d4"
servicename: "Service-C"
message: "Konnichiwa, from Service-C!"
createdat: 2019-03-17 15:31:32.260

_id: ObjectId("5c8e68608608bbbcb2163eb9")
id: "83859e50-c1dc-46ab-90d9-54fa7e1bdd10"
servicename: "Service-C"
message: "Konnichiwa, from Service-C!"
createdat: 2019-03-17 15:31:44.063

_id: ObjectId("5c8e6ce92499393a7f862dce")
id: "efbd5b5c-0bc1-4e42-8d12-b40cb518ef82"
servicename: "Service-C"
message: "Konnichiwa, from Service-C!"
createdat: 2019-03-17 15:51:05.148

_id: ObjectId("5c8e6cea2499393a7f862dcf")
id: "7c7d547e-5699-47a5-9fb2-d33674f4a16c"
servicename: "Service-C"
message: "Konnichiwa, from Service-C!"
createdat: 2019-03-17 15:51:06.261
```

[(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-17_at_11_55_19_am.png)](https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-17_at_11_55_19_am.png)

Next, confirm new the RabbitMQ queue was created, using the CloudAMQP RabbitMQ Management Console. Service D produces messages, which Service F consumes from the queue.


Refreshed 2019-03-09 18:22:07 | Refresh every 5 seconds | Virtual host: All | Cluster albatross | User axmrujav

Overview — Totals

Queued messages (last minute): Ready 0, Unacked 0, Total 0

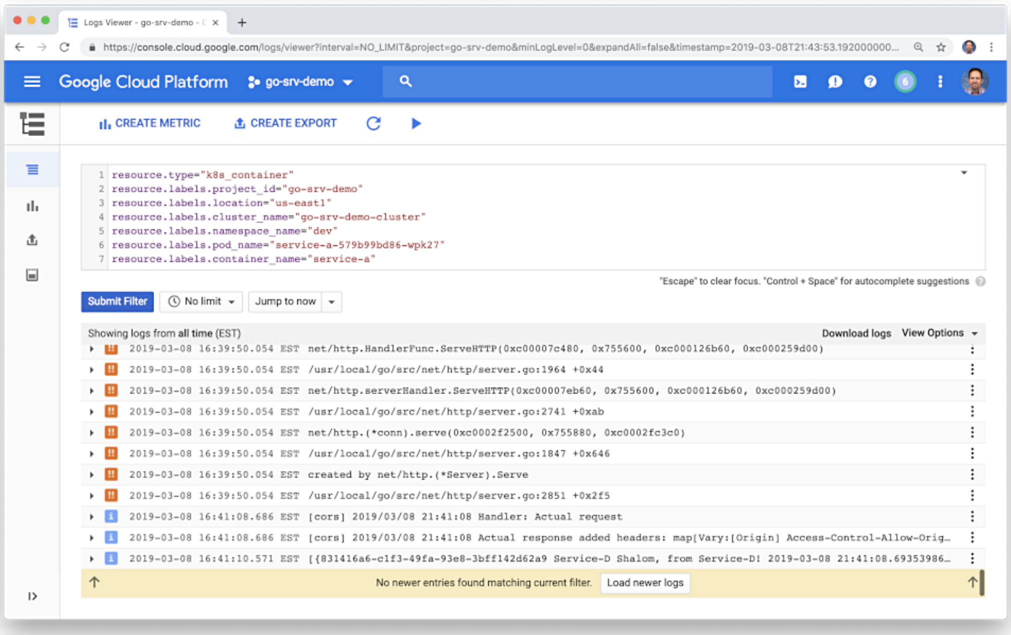
Message rates (last minute): Publish 0.00/s, Publisher confirm 0.00/s, Deliver (manual ack) 0.00/s, Deliver (auto ack) 0.00/s, Consumer ack 0.00/s, Redelivered 0.00/s, Get (manual ack) 0.00/s, Get (auto ack) 0.00/s, Return 0.00/s

Disk read 0.00/s, Disk write 0.00/s

Global counts: Connections: 4, Channels: 4, Exchanges: 7, Queues: 1

[(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_6_22_08_pm.png)](https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_6_22_08_pm.png)

Lastly, review the Stackdriver logs to see if there are any obvious errors.



```
resource.type="k8s_container"
resource.labels.project_id="go-srv-demo"
resource.labels.location="us-east1"
resource.labels.cluster_name="go-srv-demo-cluster"
resource.labels.namespace_name="dev"
resource.labels.pod_name="service-a-579b99bd86-wpk27"
resource.labels.container_name="service-a"
```

"Escape" to clear focus. "Control + Space" for autocomplete suggestions

Showing logs from all time (EST)

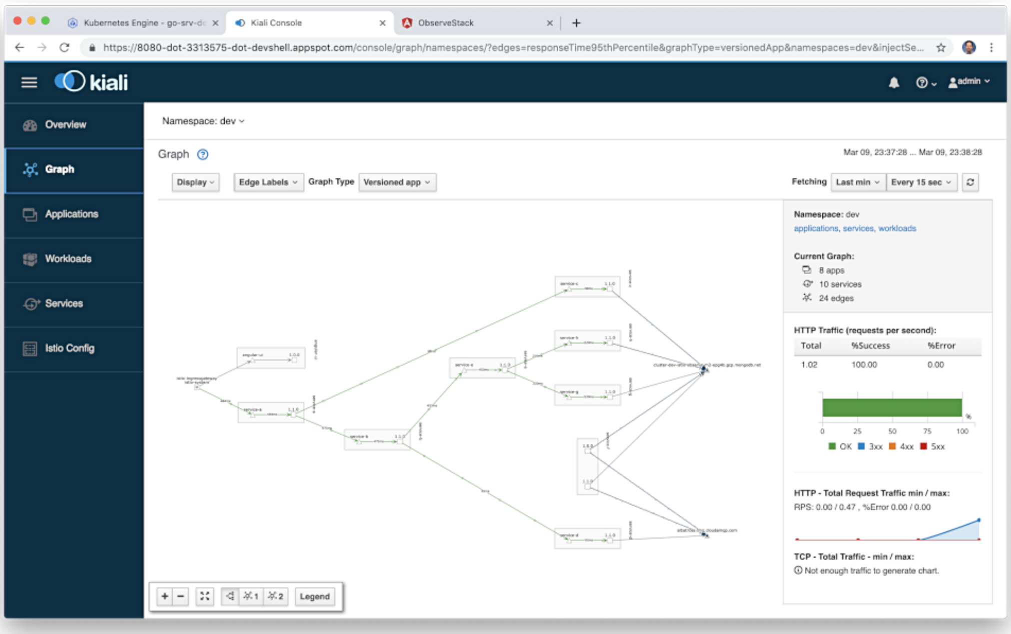
```
2019-03-08 16:39:50.054 EST net/http.HandlerFunc.ServeHTTP(0xc00007c480, 0x755600, 0xc000126b60, 0xc000259d00)
2019-03-08 16:39:50.054 EST /usr/local/go/src/net/http/server.go:1964 +0x44
2019-03-08 16:39:50.054 EST net/http.serverHandler.ServeHTTP(0xc00007eb60, 0x755600, 0xc000126b60, 0xc000259d00)
2019-03-08 16:39:50.054 EST /usr/local/go/src/net/http/server.go:2741 +0xab
2019-03-08 16:39:50.054 EST net/http.(*conn).serve(0xc0002f2500, 0x755880, 0xc0002fc3c0)
2019-03-08 16:39:50.054 EST /usr/local/go/src/net/http/server.go:1847 +0x646
2019-03-08 16:39:50.054 EST created by net/http.(*Server).Serve
2019-03-08 16:39:50.054 EST /usr/local/go/src/net/http/server.go:2851 +0x2f5
2019-03-08 16:41:08.686 EST [cors] 2019/03/08 21:41:08 Handler: Actual request
2019-03-08 16:41:08.686 EST [cors] 2019/03/08 21:41:08 Actual response added headers: map[Vary:[Origin] Access-Control-Allow-Orig…
2019-03-08 16:41:10.571 EST [{831416a6-c1f3-49fa-93e8-3bff142d62a9 Service-D Shalom, from Service-D! 2019-03-08 21:41:08.69353986…
```

No newer entries found matching current filter.

(https://programmaticponderings.files.wordpress.com/2019/03/screen-shot-2019-03-08-at-4_44_03-pm.png)

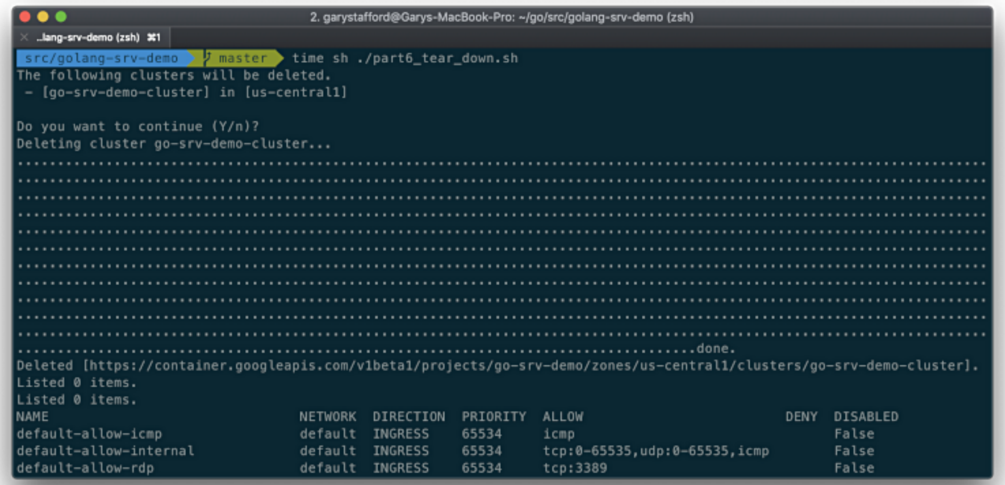
Part Two

In part two (https://programmaticponderings.com/2019/03/21/kubernetes-based-microservice-observability-with-istio-service-mesh-part-2/) of this post, we will explore each observability tool, and see how they can help us manage our GKE cluster and the reference platform running in the cluster.


kiali
Overview
Graph
Applications
Workloads
Services
Istio Config
Namespace: dev
Graph
Display
Edge Labels
Graph Type
Versioned app
Fetching
Last min
Every 15 sec
Mar 09, 23:37:28 ... Mar 09, 23:38:28
Namespace: dev
applications, services, workloads
Current Graph:
8 apps
10 services
24 edges
HTTP Traffic (requests per second):
Total
%Success
%Error
1.02
100.00
0.00
OK
3xx
4xx
5xx
HTTP - Total Request Traffic min / max:
RPS: 0.00 / 0.47 , %Error 0.00 / 0.00
TCP - Total Traffic - min / max:
Not enough traffic to generate chart.
Legend

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-09_at_11_38_34_pm.png)

Since the cluster only takes minutes to fully create and deploy resources to, if you want to tear down the GKE cluster, run the part6_tear_down.sh (https://github.com/garystafford/golang-srv-demo/blob/master/part6_tear_down.sh) script.


2. garystafford@Garys-MacBook-Pro: ~/go/src/golang-srv-demo (zsh)
..lang-srv-demo (zsh) ⌘1

```
src/golang-srv-demo  master  time sh ./part6_tear_down.sh
The following clusters will be deleted.
 - [go-srv-demo-cluster] in [us-central1]

Do you want to continue (Y/n)?
Deleting cluster go-srv-demo-cluster...
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
......................................................................done.
Deleted [https://container.googleapis.com/v1beta1/projects/go-srv-demo/zones/us-central1/clusters/go-srv-demo-cluster].
Listed 0 items.
Listed 0 items.
NAME                          NETWORK  DIRECTION  PRIORITY  ALLOW                            DENY  DISABLED
default-allow-icmp            default  INGRESS    65534     icmp                                   False
default-allow-internal        default  INGRESS    65534     tcp:0-65535,udp:0-65535,icmp           False
default-allow-rdp             default  INGRESS    65534     tcp:3389                               False
```

(https://programmaticponderings.files.wordpress.com/2019/03/screen_shot_2019-03-10_at_10_58_55_pm.png)

All opinions expressed in this post are my own and not necessarily the views of my current or past employers or their clients.

GCP , GKE , Go , Go-lang , Google Cloud Platform , Grafana , Istio , Jaeger , Kiali , Kubernetes , Logging , Metrics , Microservices , Observability , Prometheus , Telemetry , Zipkin

This entry was posted on March 10, 2019, 11:01 pm and is filed under Build Automation, Client-Side Development, Cloud, DevOps, GCP, Go, JavaScript, Kubernetes, Software Development. You can follow any responses to this entry through RSS 2.0. You can leave a response, or trackback from your own site.

COMMENTS (0) TRACKBACKS (3)

1. Kubernetes-based Microservice Observability with Istio Service Mesh: Part 2 | Programmatic Ponderings
2. Azure Kubernetes Service (AKS) Observability with Istio | Programmatic Ponderings
3. Istio Observability with Go, gRPC, and Protocol Buffers-based Microservices | Programmatic Ponderings