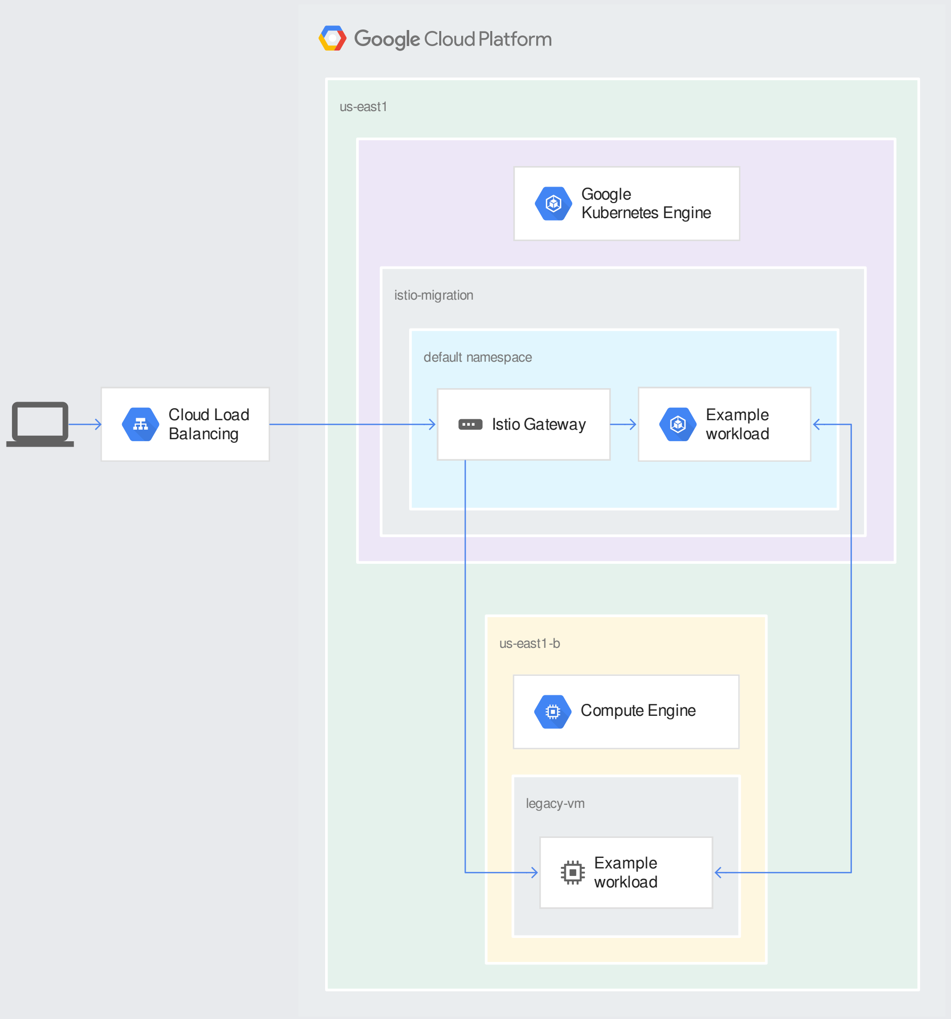
This tutorial shows how to initialize and configure a service mesh (https://istio.io/docs/concepts/what-is-istio/) to support a feature-by-feature migration from an on-premises (legacy) data center to Google Cloud (/). The tutorial and its accompanying conceptual article (/solutions/supporting-your-migration-with-istio-mesh-expansion-concept) is intended for sysadmins, developers, and engineers who want to use a service mesh that dynamically routes traffic either to the legacy environment or to Google Cloud.

A service mesh can greatly reduce the complexity of both the migration exercise and the refactoring effort because it decouples the network functions from the service functions. It also reduces the network operational complexity because it provides load balancing, traffic management, monitoring, and observability.

The following diagram shows how you can use a service mesh to route traffic either to microservices running in the legacy environment or to Google Cloud:



In this tutorial, you use the following software:

- Ubuntu Server (https://www.ubuntu.com/server) and Container-Optimized OS (/container-optimized-os/docs/): Operating systems used in this tutorial
- Docker Community Edition (https://www.docker.com/docker-community): Platform to run containerized workloads
- Docker Compose (https://docs.docker.com/compose/): A tool for defining and running Docker apps
- Helm (https://helm.sh/): A tool to install and manage Kubernetes app

- Istio (https://istio.io/): An open source service mesh
- Kiali (https://www.kiali.io/): A tool to visualize Istio service meshes
- Envoy (https://www.envoyproxy.io/): sidecar proxy used when joining Istio service mesh

Note: this tutorial uses the default Virtual Private Cloud (/vpc/) (VPC). If you prefer to use a dedicated VPC, modify the commands accordingly.

- Initialize an environment simulating the on-premises data center.
- Deploy example workloads to the on-premises data center.
- Test the workloads running on the on-premises data center.
- Configure the destination environment on Google Cloud.
- Migrate the workload from the on-premises data center to the destination environment.
- Test the workloads running in the destination environment.
- Retire the on-premises data center.

This tutorial uses billable components of Google Cloud, including:

- Compute Engine (/compute/)
- Google Kubernetes Engine (GKE) (/kubernetes-engine/)
- Persistent Disk (/persistent-disk/)
- Networking (/vpc/)

Use the Pricing Calculator (/products/calculator/#id=0ab80d88-26ac-4575-b80d-52b96689480c) to generate a cost estimate based on your projected usage.

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

   ★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboard)

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).

4. Enable the Compute Engine and GKE APIs.

   Enable the APIs (https://console.cloud.google.com/flows/enableapi?apiid=compute.googleapis.com,container.googleapis.com&redirect=https://console.clo

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Cleaning up (#clean-up).

You perform most of the steps for this tutorial in Cloud Shell (/shell/).

1. Open Cloud Shell:

   OPEN Cloud Shell (https://console.cloud.google.com/?cloudshell=true)

★ **Note:** You need about 200 MB of free space to complete this tutorial. You can check how much free space you have by running the `df -h` command in Cloud Shell.

2. Change the working directory to the `$HOME` directory:

3. Clone the Git repository, which contains the scripts and the manifest files to deploy and configure the demo app:

4. Set the default region and zone:

5. Initialize an environment variable that stores the Istio version identifier:

6. Download and extract Istio:

★ **Note:** In the next step, you set and use the `$ISTIO_PATH` and `$HELM_PATH` environment variables to avoid interfering with any existing installations of Istio and Helm.

7. Initialize environment variables that store the path where you extracted Istio, the Helm version identifier, and the path where you extract Helm:

8. Download and extract Helm:

In this tutorial, you use the Bookinfo (https://istio.io/docs/examples/bookinfo/) app, which is a 4-tier, polyglot (https://wikipedia.org/wiki/Polyglot_(computing)) microservices app that shows information about books. This app is designed to run on Kubernetes, but you first deploy it on a Compute Engine instance using Docker and Docker Compose. With Docker Compose, you describe multi-container apps using YAML (https://yaml.org/) descriptors (https://docs.docker.com/compose/compose-file). You can then start the app by executing a single command.

Although this example workload is already containerized, this approach also applies to non-containerized services. In such cases, you can add a "modernization phase" where you containerize services (/containers/) that you intend to migrate.

The Bookinfo app has four microservice components:

- `productpage`: Calls the `details`, `ratings`, and `reviews` microservices to populate the book information page
- `details`: Serves information about books
- `reviews`: Contains book reviews
- `ratings`: Returns book ranking information to accompany a book review

Note: To demonstrate Istio and its features, the authors and maintainers of the Bookinfo app have implemented multiple versions of some of these components. In this tutorial, you deploy just one version of each component.

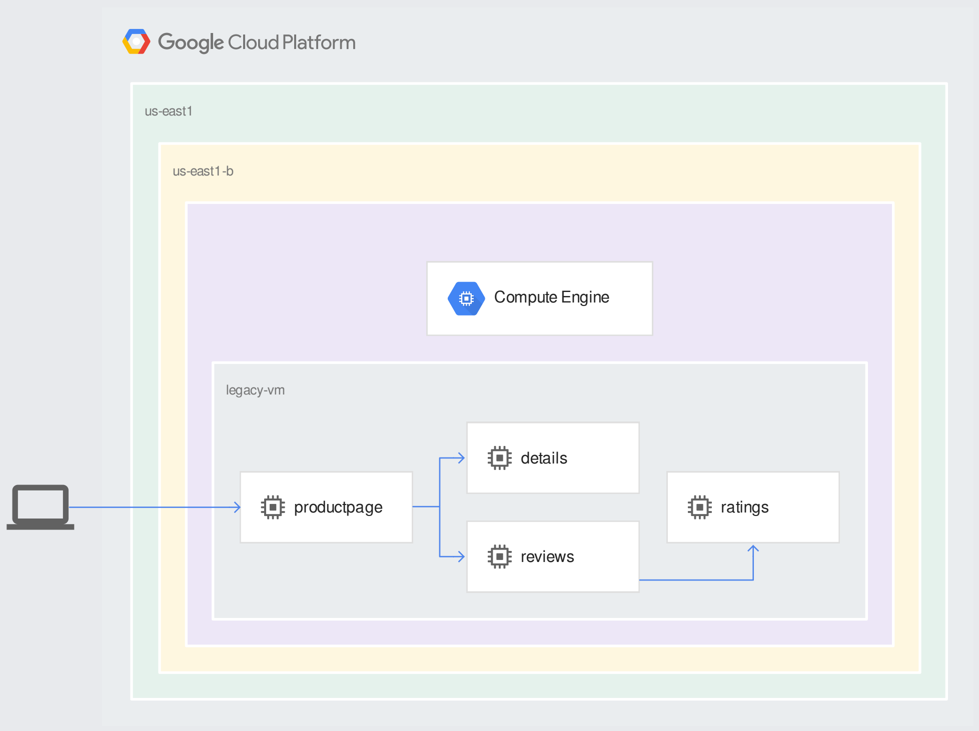
The first step is to configure the environments that you need for this tutorial:

- An environment simulating the (legacy) on-premises data center
- An environment simulating the migration destination

This tutorial is intended to help you migrate from a non-Google Cloud environment (such as on-premises or another cloud provider) to Google Cloud. Such migrations have a layer of network complexity because you have to set up a secure communication channel (/hybrid-connectivity/) between the non-Google Cloud environment and the Google Cloud one.

For this tutorial, both environments run in Google Cloud. This simplifies the setup process by requiring only a single bootstrapping phase.

In this section, you configure a Google Cloud environment to emulate a separate non-Google Cloud environment by initializing Compute Engine (/compute/) instances and deploying the workloads to migrate. The following diagram shows the target architecture of the legacy environment:



Note: In this tutorial, clients connect directly to a single Compute Engine instance to reduce complexity. In a real production environment, this direct connection is unlikely: running multiple instances of a workload requires a load-balancing layer.

You create firewall rules to allow external access to the microservices and to the database.

- In Cloud Shell, create the firewall rules that are required for inter-node communications:

In this tutorial, you create a service account to manage Compute Engine instances. It's a best practice to limit the service account to just the roles and access permissions that are required in order to run the app. For this tutorial, the only role required for the service account is the Compute Viewer role (`roles/compute.viewer`). This role provides read-only access to Compute Engine resources.

1. In Cloud Shell, initialize an environment variable that stores the service account name:

2. Create a service account:

3. Initialize an environment variable that stores the full service account email address:

4. Bind the `compute.viewer` role to the service account:

The next step is to create and configure a Compute Engine instance to host the workloads to migrate.

1. In Cloud Shell, initialize and export a variable with the name of the Compute Engine instance:

2. Create a Compute Engine instance:

   The `n1-standard-1` machine type specified in this command is the smallest that lets you run the example workload without impacting performance. When this command is done, the console shows details about the new instance:

The startup script configures the Compute Engine instance by:

- Installing Docker
- Installing Docker Compose
- Installing Dnsmasq (https://wikipedia.org/wiki/Dnsmasq)

In this tutorial, you deploy the Istio Bookinfo App (https://istio.io/docs/examples/bookinfo/) as the workload to migrate.

1. In Cloud Shell, copy Docker Compose descriptors to the Compute Engine instance:

2. Wait for Docker Compose installation to complete:

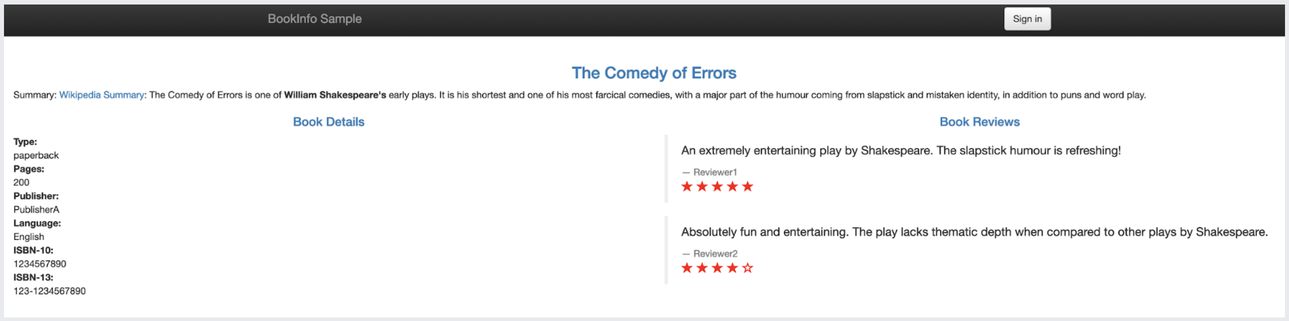
3. Start the Bookinfo app with Docker Compose:

You've finished configuring the example workload, so now you can test it.

1. In Cloud Shell, find the external IP address of the Compute Engine instance that's running the example workload:

2. Open a browser and go to the following URL, where `[EXTERNAL_IP]` is the IP address from the previous step:

   A page is displayed with details about books and relevant ratings:



The Comedy of Errors

Summary: Wikipedia Summary: The Comedy of Errors is one of **William Shakespeare's** early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

Book Details

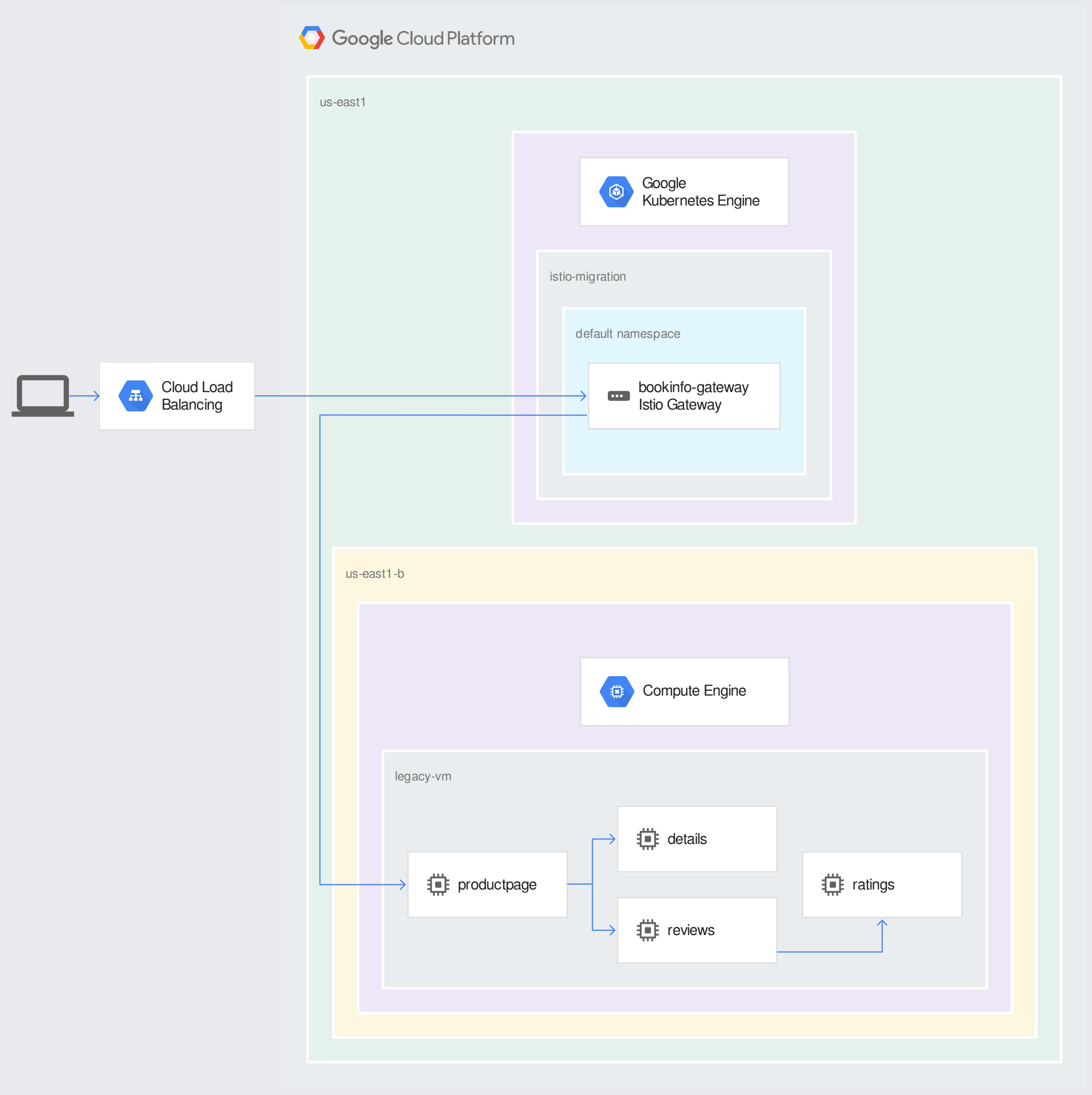
Type:
paperback
Pages:
200
Publisher:
PublisherA
Language:
English
ISBN-10:
1234567890
ISBN-13:
123-1234567890

Book Reviews

An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!
— Reviewer1
★★★★★

Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare.
— Reviewer2
★★★★☆

In this section, you configure the destination environment in Google Cloud by initializing a GKE cluster and exposing the legacy service using Istio. The following diagram shows the target architecture of destination runtime environment:



In this tutorial, you create a service account to manage Compute Engine instances in the GKE cluster. The GKE cluster nodes use this service account instead of the default service account, because you want to provide fewer permissions than what are granted to the default service account.

The roles required for the service account are `monitoring.viewer`, `monitoring.metricWriter`, and `logging.logWriter`, as described in Hardening your cluster's security (/kubernetes-engine/docs/how-to/hardening-your-cluster#use_least_privilege_sa).

1. In Cloud Shell, initialize an environment variable that stores the service account name:

2. Create a service account:

3. Initialize an environment variable that stores the service account's email account name:

4. Grant the `monitoring.viewer`, `monitoring.metricWriter`, and `logging.logWriter` roles to the service account:

In this section, you launch the GKE cluster, install Istio, expose Istio services, and finish the cluster configuration. The first step is to create and launch the GKE cluster.

1. In Cloud Shell, initialize and export an environment variable that stores the GKE cluster name:

2. Create a regional GKE cluster (/kubernetes-engine/docs/concepts/regional-clusters) with one node pool (/kubernetes-engine/docs/concepts/node-pools) and a single node in each zone:

   This command creates a GKE cluster named `istio-migration`. The execution of this command might take up to five minutes. When the command is done, the console shows details about the newly created cluster:

Next, you use Helm to install Istio in the cluster.

1. In Cloud Shell, initialize and export an environment variable that stores the Istio namespace (https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/) name:

2. Create the Istio namespace:

3. Create a Kubernetes service account (https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/) for Tiller (https://v2.helm.sh/docs/install/#installing-tiller), the server portion of Helm:

4. Install Tiller in your cluster:

5. Install the `istio-init` chart to bootstrap all Istio's custom resource definitions (https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/):

   When this command is done, the console shows a summary of the new objects that were installed in the GKE cluster:

6. Verify that the Istio CRDs were committed to the Kubernetes `api-server`:

The expected output is `53`.

7. Install the Istio chart:

   Execution of this command might take up to two minutes. When the command is done, the console shows a summary of the new objects that were installed in the GKE cluster:

   A list of the deployed resources follows this summary.

Because Compute Engine instances that you want to add to the service mesh must have access to the Istio control plane services (https://istio.io/docs/ops/architecture/) (Pilot, Mixer, Citadel), you must expose these services through the `istio-ingressgateway` and `istio-ilbgateway` services. You also expose the Kubernetes DNS server using an internal load balancer (/load-balancing/docs/internal/) so the server can be queried to resolve names for services running in the cluster. You also expose Kiali to visualize the service mesh.

1. In Cloud Shell, create the internal load balancer to expose the Kubernetes DNS server:

2. Wait for the service object named `kube-dns-ilb` to be assigned an external IP address:

   The output displays an IP address for `EXTERNAL-IP`:

   To stop the command, press `Control+C`.

3. Wait for the service object named `istio-ingressgateway` to be assigned an external IP address:

The output displays an IP address for `EXTERNAL-IP`:

To stop the command, press `Control+C`.

4. Wait for the service object named `istio-ilbgateway` to be assigned an external IP address:

   The output displays an IP address for `EXTERNAL-IP`:

   To stop the command, press `Control+C`.

5. Expose Kiali with a Gateway (https://istio.io/docs/reference/config/networking/gateway/) and a Virtual Service (https://istio.io/docs/reference/config/networking/virtual-service/):

Warning: The Kiali instance you're exposing uses weak credentials over an unsecure channel (plain HTTP). For production use, we recommend that you use stronger credentials over a secure channel, like HTTPS.

In this section, you configure Istio mesh expansion (https://istio.io/docs/examples/mesh-expansion/) to allow Compute Engine instances to join the service mesh. The first task is to generate the configuration files to deploy in each Compute Engine instance you want to join the mesh.

1. In Cloud Shell, initialize and export an environment variable that stores the default namespace name:

2. Extract the keys for the service account used by Istio:

3. Initialize and export an environment variable that stores the options for the configuration generation script:

4. Run the cluster environment configuration generation script:

5. Run the DNS configuration generation script:

Next, you configure the Compute Engine instance that you want to join the mesh.

1. Unset the `GCP_OPTS` variable to let the initialization script pick the defaults:

2. Initialize and export an environment variable that stores the path for the setup script:

3. Prepare the Istio version descriptor for the Compute Engine instance initialization:

4. Run the cluster environment configuration generation script:

5. Configure the local ports that will use Envoy sidecar for inbound services:

6. Configure the namespace of the cluster:

7. Restart Istio:

8. Check that Istio has been started correctly:

   The output shows that the `dnsmasq`, `istio`, `istio-auth-node-agent`, and `systemd-resolved` services are loaded, active, and running:

You then add services running on the Compute Engine instance in the Istio service mesh.

1. In Cloud Shell, create the Kubernetes services without selectors (https://kubernetes.io/docs/concepts/services-networking/service/#services-without-selectors) to expose the services running in the Compute Engine instance:

★ **Note:** Because you're creating Kubernetes services without selectors, the corresponding Endpoints (https://kubernetes.io/docs/concepts/services-networking/service/#defining-a-service) aren't created now. They're created when you register services that are running in the Compute Engine instance in the mesh.

★ **Note:** Because the Bookinfo app expects that each microservice is serving content from port `9080`, you create the Kubernetes Service objects in advance to forward traffic to the correct ports that you're going to register in the Istio service mesh.

2. Initialize an environment variable that stores the IP address of the Compute Engine instance where the example workload is running:

3. Register the services to the mesh:

Finally, you configure the VirtualServices and the corresponding routing rules for the services in the mesh that are currently running on the Compute Engine instance. You also expose the `productpage` service through an Istio ingress gateway.

1. In Cloud Shell, deploy a Gateway object to expose services:

2. Deploy the VirtualService that routes traffic from the Gateway object to the `productpage` instance running in Compute Engine:

3. Create the ServiceEntries (https://istio.io/docs/reference/config/networking/service-entry/) to enable service discovery for the services running in the Compute Engine instances:

4. Wait for the Service object named `istio-ingressgateway` to be assigned an external IP address:

In the output, you should see an IP address for `EXTERNAL-IP`:

5. Confirm that a gateway object named `bookinfo-gateway` has been created:

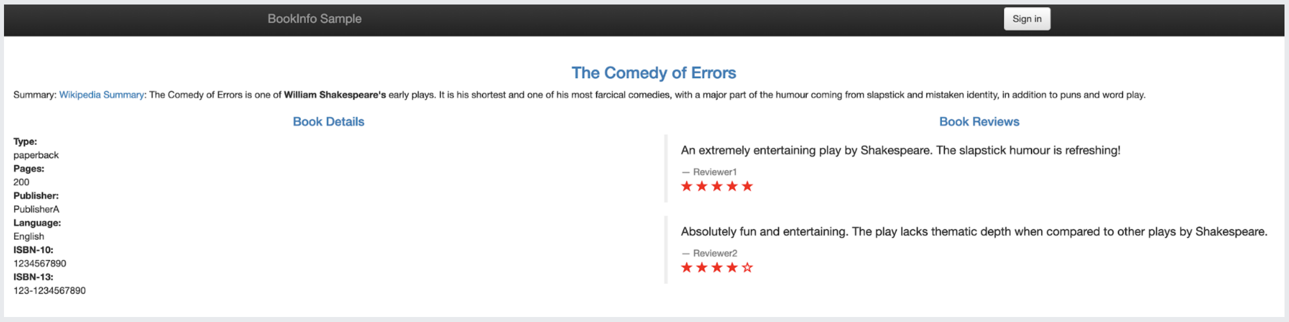
   In the output, you should see the `bookinfo-gateway` in the list of gateway objects:

Now that you've finished exposing the example workload running in the Compute Engine instance using Istio, you can test it:

1. In Cloud Shell, find the external IP address of the Istio ingress gateway:

2. Open a browser and go to the following URL, where `[EXTERNAL_IP]` is the IP address from the previous step:

   A page is displayed with information about books and relevant ratings:



The Comedy of Errors

Summary: Wikipedia Summary: The Comedy of Errors is one of **William Shakespeare's** early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

Book Details

Type:
paperback
Pages:
200
Publisher:
PublisherA
Language:
English
ISBN-10:
1234567890
ISBN-13:
123-1234567890

Book Reviews

An extremely entertaining play by Shakespeare. The slapstick humour is refreshing!
— Reviewer1
★★★★★

Absolutely fun and entertaining. The play lacks thematic depth when compared to other plays by Shakespeare.
— Reviewer2
★★★★☆

Note: The "legacy entry-point" (direct connection to the Compute Engine instance) is still available at this point. In a real migration, we recommend that you gradually redirect traffic to the target environment by updating the configuration of the load balancing layer.

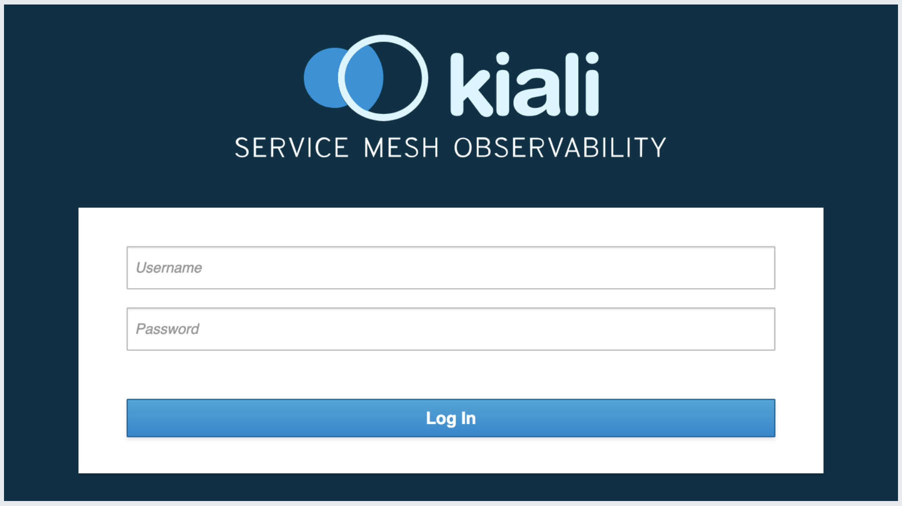
In this section, you use Kiali to see a visual representation of the service mesh.

1. In Cloud Shell, find the external IP address of the Istio ingress gateway:

2. Open a browser and go to the following URL, where `[EXTERNAL_IP]` is the IP address from the previous step:

3. At the Kiali login screen, log in with the following credentials:

   - Username: `admin`
   - Password: `admin`

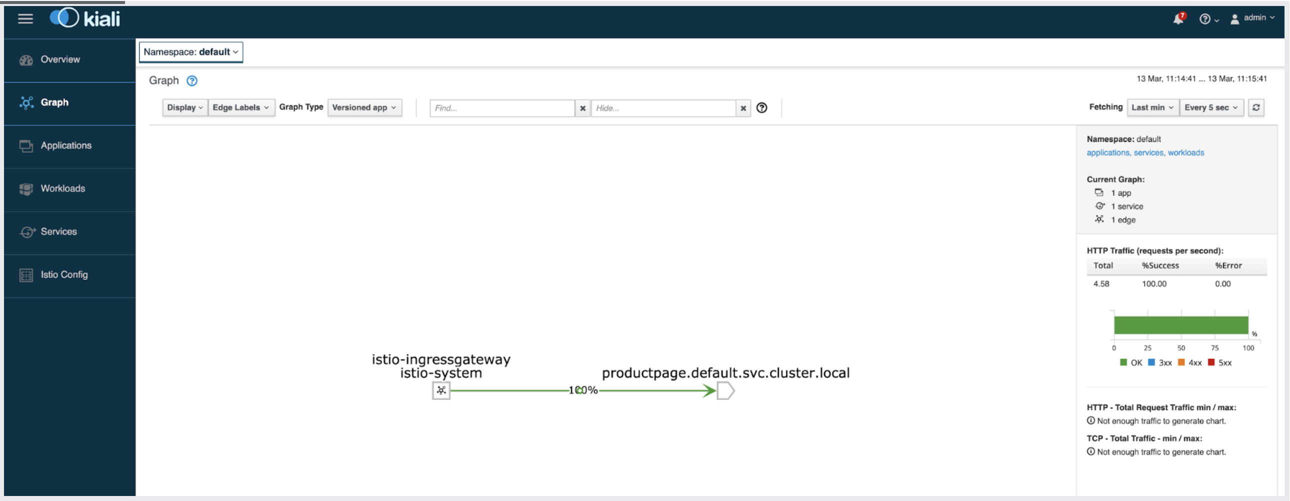
   

4. Run a request multiple times for the main page of the example workload:

   With this command, you generate traffic to the Bookinfo app. The expected output is a list of the HTTP return codes of each request (`200 OK` in this case):

   In the Kiali service dashboard, you should see a diagram of the current mesh, with all the traffic routed to services running in Compute Engine. All the traffic is routed from `istio-ingressgateway` to the Kubernetes Service `productpage.default.svc.cluster.local`, pointing to the `productpage` microservice running on the Compute Engine instance. You don't see the other microservices in the graph (`details`, `reviews`, `ratings`) because Docker Compose is handling the routing locally on the Compute Engine instance.

   If you don't see the diagram, refresh the Kiali dashboard page.


kiali
admin
Overview
Graph
Applications
Workloads
Services
Istio Config
Namespace: default
Graph
13 Mar, 11:14:41 ... 13 Mar, 11:15:41
Display
Edge Labels
Graph Type
Versioned app
Find...
Hide...
Fetching
Last min
Every 5 sec
istio-ingressgateway
istio-system
100%
productpage.default.svc.cluster.local
Namespace: default
applications, services, workloads
Current Graph:
1 app
1 service
1 edge
HTTP Traffic (requests per second):
0
25
50
75
100
%
OK
3xx
4xx
5xx
HTTP - Total Request Traffic min / max:
Not enough traffic to generate chart.
TCP - Total Traffic - min / max:
Not enough traffic to generate chart.

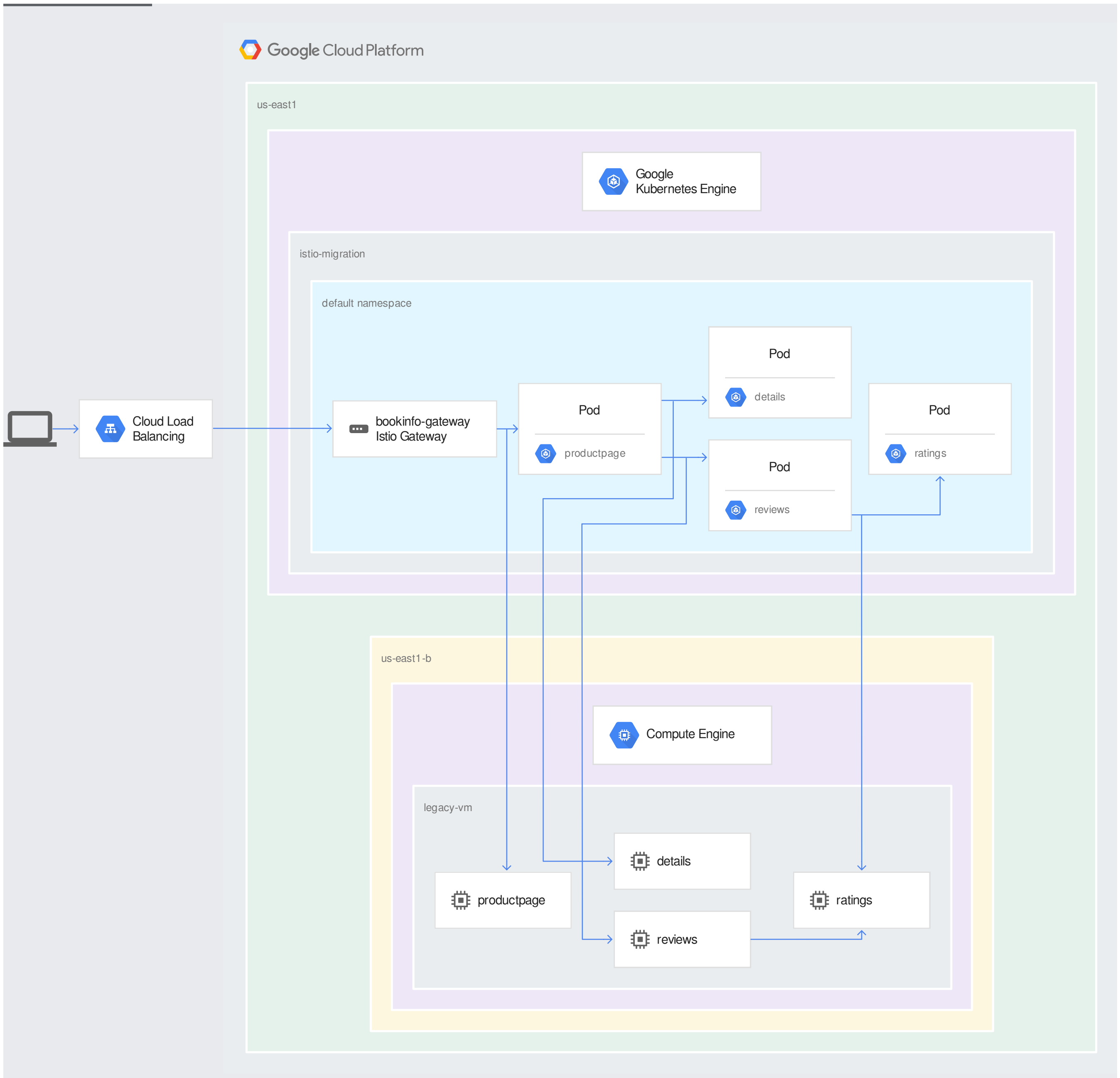
Total	%Success	%Error
4.58	100.00	0.00

★ **Note:** The traffic generation command is still running.

In this section, you migrate the components of the example workload from the Compute Engine instance to the GKE cluster. For each service of the example workload, you:

1. Deploy a Pod (https://kubernetes.io/docs/concepts/workloads/pods/pod/) that runs the service in the GKE cluster.
2. Configure rules to split traffic (https://istio.io/docs/concepts/traffic-management/#routing-rules) between the service that's running in the GKE cluster and the one that's running in the Compute Engine instance.
3. Gradually migrate traffic (https://istio.io/docs/tasks/traffic-management/traffic-shifting/) from the service that's running in Compute Engine instance to the GKE cluster.
4. Stop the service that's running in the Compute Engine instance.

The following diagram shows the target architecture of the system for this section:



Each service needs an Envoy (https://www.envoyproxy.io/) sidecar proxy running to join Istio service mesh. You can enable automatic sidecar injection (/istio/docs/istio-on-gke/installing#enabling_sidecar_injection) to avoid manual edits to your pods' configuration.

1. In Cloud Shell, stop the traffic generation command by pressing `Control+C`.

2. Enable the Istio-managed automatic sidecar injection in the Kubernetes namespace where you will deploy service instances:

In this section, you deploy instances in the GKE cluster and route part of the traffic to those instances.

1. In Cloud Shell, delete the ServiceEntries before deploying other services in the mesh. Deleting these entries prevents routing requests to instances that aren't yet ready:

2. Deploy pods and Kubernetes Services for the `productpage`, `details`, `reviews`, and `ratings` microservices in the cluster:

3. Update the VirtualServices configuration to split incoming traffic between the instances running in the Compute Engine machine and the ones running in the GKE cluster:

4. Create the ServiceEntries to enable service discovery for the services running in the Compute Engine instances:

Note: To avoid downtime, always deploy Istio Gateways and VirtualServices *after* the workload that will serve content for such objects.

The next task is to validate your hybrid deployment by inspecting traffic directed to all microservice instances in your two environments (Compute Engine and GKE).

You use Kiali to see a visual representation of the service mesh:

1. In Cloud Shell, find the external IP address of the Istio ingress gateway:

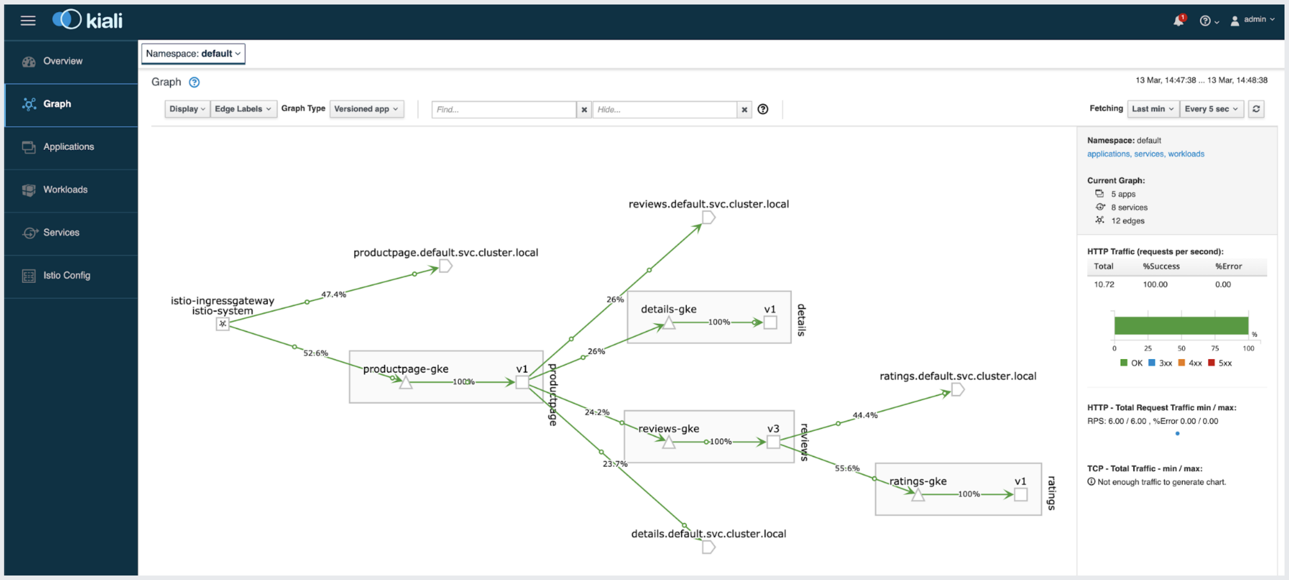
2. Open a browser and go to the following URL, where `[EXTERNAL_IP]` is the IP address from the previous step:

3. At the Kiali login screen, log in with the following credentials, if necessary:
   - Username: `admin`
   - Password: `admin`
4. Run a request multiple times for the main page of the example workload:

With this command, you generate traffic to the Bookinfo app. The expected output is a list of the HTTP return codes of each request (`200 OK` in this case):

In the Kiali service dashboard, you should see that traffic is being almost equally split between microservice instances running in Compute Engine and the ones running in GKE (with the `-gke` suffix in the first part of the service identifier). This is because you configured the routes in the VirtualService to have the same weight.

If you don't see the diagram, refresh the Kiali dashboard page.



Namespace: default

Graph Type: Versioned app

13 Mar, 14:47:38 ... 13 Mar, 14:48:38

Fetching: Last min, Every 5 sec

Namespace: default
applications, services, workloads

Current Graph:
5 apps
8 services
12 edges

HTTP Traffic (requests per second):

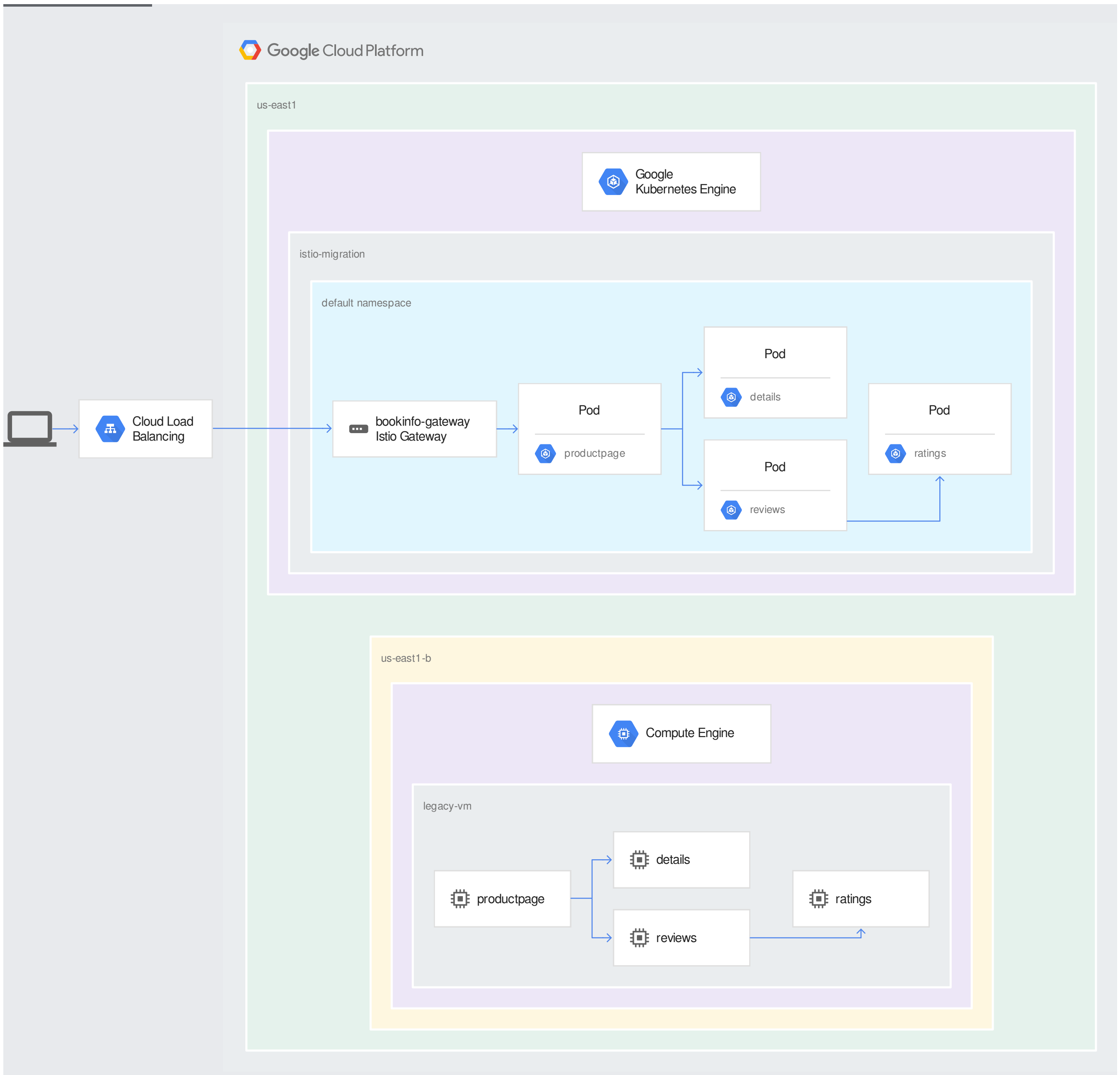
Total	%Success	%Error
10.72	100.00	0.00

HTTP - Total Request Traffic min / max:
RPS: 6.00 / 6.00 , %Error 0.00 / 0.00

TCP - Total Traffic - min / max:
Not enough traffic to generate chart.

When you're confident of the deployment in the GKE cluster, you can update Services and VirtualServices to route traffic to the cluster only.

The following diagram shows the target architecture of the system for this section:



1. In Cloud Shell, stop the traffic generation command by pressing `Control+C`.
2. Update Services and VirtualServices to route traffic to microservice instances in the GKE cluster:

To validate your deployment, you inspect traffic directed to all microservice instances in your two environments (Compute Engine and GKE).

You use Kiali to see a visual representation of the service mesh:

1. In Cloud Shell, find the external IP address of the Istio ingress gateway:

2. Open a browser and go to the following URL, where `[EXTERNAL_IP]` is the IP address from the previous step:

   ```
   http://[EXTERNAL_IP]:15029/kiali/console/graph/namespaces/?
   edges=requestsPercentOfTotal&graphType=versionedApp&namespaces=default&injectServiceNodes=true&duration=60&pi=5000&layout
   =dagre
   ```

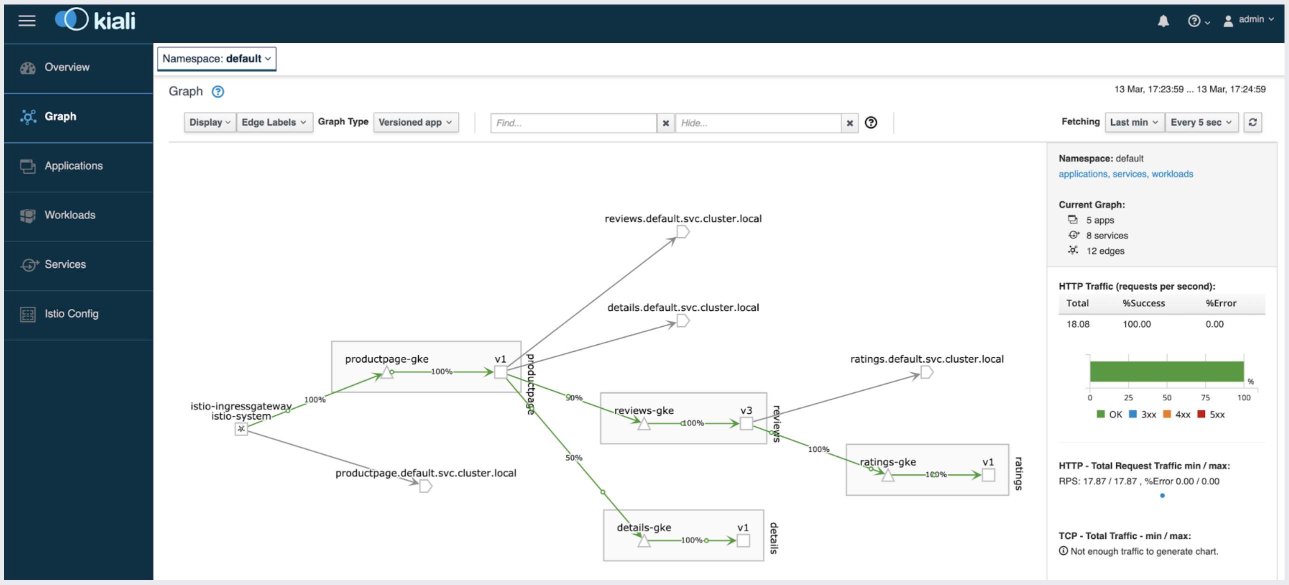
3. At the Kiali login screen, log in with the following credentials, if necessary:
   - Username: `admin`
   - Password: `admin`

4. Run a request for the main page of the example workload multiple times:

   With this command, you generate traffic to the Bookinfo app. The expected output is a list of the HTTP return codes of each request (`200 OK` in this case):

   In the Kiali service dashboard, you should see that traffic is being routed to microservice instances running in the GKE cluster (with the `-gke` suffix in the first part of the service identifier), while no traffic is routed to the instances running in Compute Engine. If you don't see the diagram, refresh the Kiali dashboard page. While you deployed two instances of each microservice (one running in the Compute Engine instance, the other running in the GKE cluster), you configured the VirtualServices to route traffic to only the microservices instances running in the GKE cluster.

   If you don't see the diagram, refresh the Kiali dashboard page.



Namespace: default
applications, services, workloads

Current Graph:
5 apps
8 services
12 edges

HTTP Traffic (requests per second):

Total	%Success	%Error
18.08	100.00	0.00

HTTP - Total Request Traffic min / max:
RPS: 17.87 / 17.87 , %Error 0.00 / 0.00

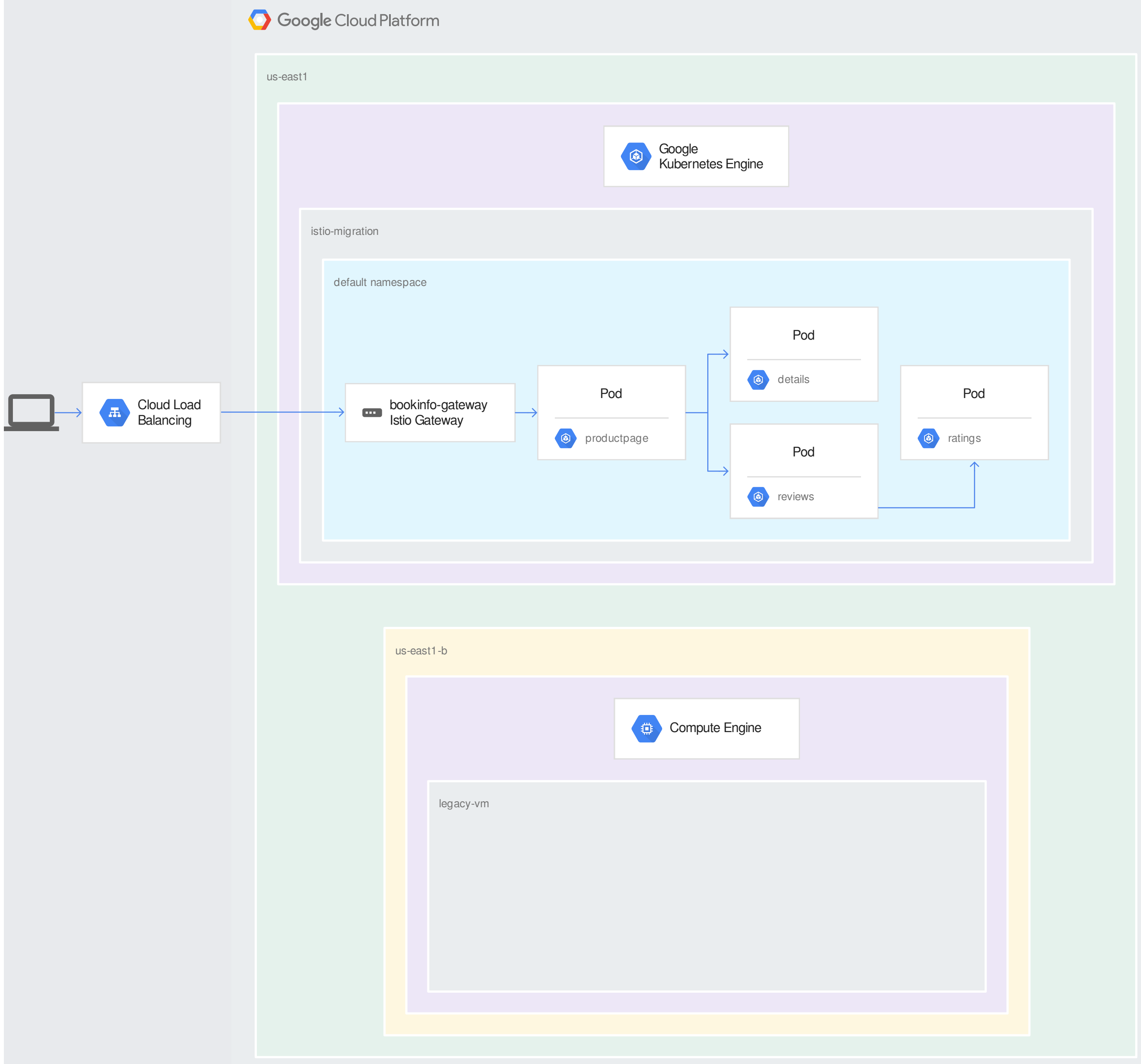
TCP - Total Traffic - min / max:
Not enough traffic to generate chart.

★ **Note:** The traffic generation command is still running.

Because all the traffic is routed to the GKE cluster, you can now delete the ServiceEntries for the microservices running in Compute Engine and stop Docker Compose.

Warning: During a real migration, keep the legacy data center ready for your rollback strategy. We recommend that you start retiring the legacy data center only when you're sure that the new solution is working as expected and that all backup and fault-tolerance mechanisms are in place.

The following diagram shows the target architecture of the system for this section:



1. In Cloud Shell, stop the traffic generation command by pressing `Control+C`.
2. Delete the ServiceEntries before deploying other services in the mesh:

3. Stop the example workload running with Docker Compose:

In this section, you validate your deployment by inspecting traffic directed to all microservice instances in your two environments (Compute Engine and GKE).

You use Kiali to see a visual representation of the service mesh:

1. In Cloud Shell, find the external IP address of the Istio ingress gateway:

2. Open a browser and go to the following URL, where `[EXTERNAL_IP]` is the IP address from the previous step:

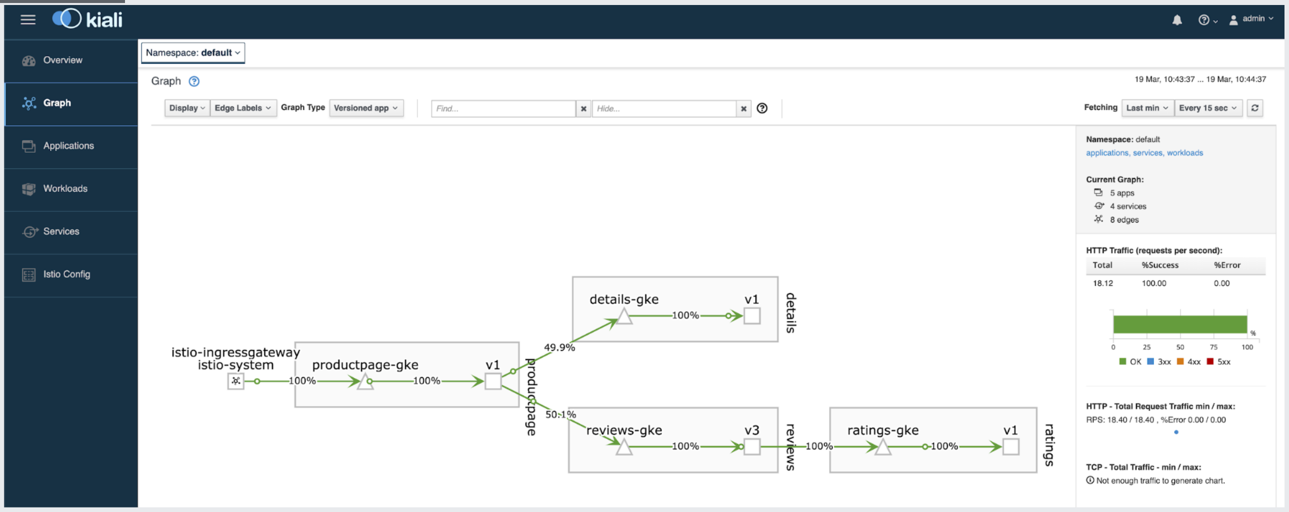
3. At the Kiali login screen, log in with the following credentials, if necessary:
   - Username: `admin`
   - Password: `admin`

4. Run a request multiple times for the main page of the example workload:

   With this command, you generate traffic to the Bookinfo app. The expected output is a list of the HTTP return codes of each request (`200 OK` in this case):

   In the Kiali service dashboard, you should see only services pointing to instances in the GKE cluster (with the `-gke` suffix in the first part of the service identifier), while services pointing to instances running in Compute Engine aren't part of the mesh anymore, since you deleted the related ServiceEntries.

   If you don't see the diagram, refresh the Kiali dashboard page:



Namespace: default
applications, services, workloads

Current Graph:
5 apps
4 services
8 edges

HTTP Traffic (requests per second):

Total	%Success	%Error
18.12	100.00	0.00

HTTP - Total Request Traffic min / max:
RPS: 18.40 / 18.40 , %Error 0.00 / 0.00

TCP - Total Traffic - min / max:
Not enough traffic to generate chart.

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

   Go to the Manage resources page (https://console.cloud.google.com/iam-admin/projects)

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

- Read about Google Kubernetes Engine (/kubernetes-engine/).
- Read about Istio (https://istio.io/).
- Try out other Google Cloud features for yourself. Have a look at our tutorials (/docs/tutorials).