Maintenance and Monitoring

Philipp Maier
Course Developer, Google Cloud

In this final module of this course, we cover application maintenance and monitoring.

Learning objectives

- Manage new service versions using rolling updates, blue/green deployments, and canary releases.
- Forecast, monitor and optimize service cost using the Google Cloud pricing calculator and billing reports, and by analyzing billing data.
- Observe whether your services are meeting their SLOs using Cloud Monitoring and Dashboards.
- Use Uptime Checks to determine service availability.
- Respond to service outages using Cloud Monitoring Alerts.

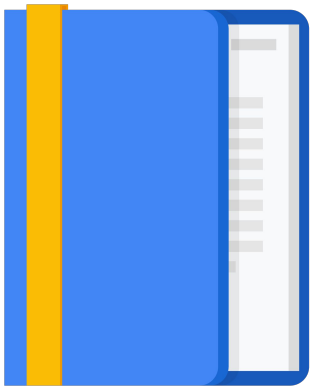
Maintenance is primarily concerned with how updates are made to running applications, the different strategies available, and how different deployment platforms support them. For monitoring, I discuss this vital area for cloud-native applications from two perspectives:

1. First, I will talk about the cost perspective to make sure that resources are being best provisioned against demand. After all, why should you pay for resources that you don't need?
2. Second, I will discuss how to implement monitoring and observability to determine and alert on the health of services and applications using Cloud monitoring and dashboards.

This will also allow us to define uptime checks and use Cloud Monitoring alerts to identify service outages. Let's get started!

Agenda

- Managing Versions
- Cost Planning
- Monitoring Dashboards



Let’s begin by taking a look at version management.

In a microservice architecture, be careful not to break clients when services are updated

- Include version in URI:
  - If you deploy a breaking change, you need to change the version.
- Need to deploy new versions with zero downtime.
- Need to effectively test versions prior to going live.

A key benefit of a microservice architecture is the ability to independently deploy microservices. This means that the service API has to be protected. Versioning is required, and when new versions are deployed, care must be taken to ensure backward compatibility with the previous version. Some simple design rules can help, such as indicating the version in the URI and making sure you change the version when you make a backwardly incompatible change. Deploying new versions of software always carries risk. We want to make sure we test new versions effectively before going live, and when ready to deploy a new version, we do so with zero downtime.

Let me discuss some strategies that can help achieve these objectives.

Rolling updates allow you to deploy new versions with no downtime

- Typically, you have multiple instances of a service behind a load balancer.
- Update each instance one at a time.
- Rolling updates work when it is ok to have 2 different versions running simultaneously during the update.

- Rolling updates are a feature of instance groups; just change the instance template.
- Rolling updates are the default in Kubernetes; just change the Docker image.
- Completely automated in App Engine.

Rolling updates allow you to deploy new versions with no downtime. The typical configuration is to have multiple instances of a service behind a load balancer. A rolling update will then update one instance at a time. This strategy works fine if the API is not changed or is backward compatible, or if it is ok to have two versions of the same service running during the update.

- If you are using instance groups, rolling updates are a built-in feature. You just define the rolling update strategy when you perform the update.
- For Kubernetes, rolling updates are there by default; you just need to specify the replacement Docker image.
- Finally, for App Engine, rolling updates are completely automated.

Use a blue/green deployment when you don't want multiple versions of a service running simultaneously

- The blue deployment is the current version.
- Create an entirely new environment (the green).
- Once the green deployment is tested, migrate client requests to it.
- If failures occur, switch it back.

- In Compute Engine, you can use DNS to migrate requests from one load balancer to another.
- In Kubernetes, configure your service to route to the new pods using labels.
  - Simple configuration change
- In App Engine, use the Traffic Splitting feature.

Use a blue/green deployment when you don't want multiple versions of a service to run simultaneously.

Blue/green deployments use two full deployment environments. The blue deployment is running the current deployed production software, while the green deployment environment is available for deploying updated versions of the software.

When you want to test a new software version, you deploy it to the green environment. Once testing is complete, the workload is shifted from the current (blue) to the new (green) environment. This strategy mitigates the risk of a bad deployment by allowing the switch back to a previous deployment if something goes wrong.

For Compute Engine, you can use DNS to migrate requests, while in Kubernetes you can configure your service to route to new pods using labels, which is just a simple configuration change. App Engine allows you to split traffic, which you explored in the previous lab of this course.

Canary releases can be used prior to a rolling update to reduce the risk

- The current service version continues to run.
- Deploy an instance of the new version and give it a portion of requests.
- Monitor for errors.

- In Compute Engine, you can create a new instance group and add it as an additional backend in in your load balancer.
- In Kubernetes, create a new pod with the same labels as the existing pods; the service will automatically route a portion of requests to it.
- In App Engine, use the Traffic Splitting feature.

Now, you can use canary releases prior to a rolling update to reduce risk. With a canary release, you make a new deployment with the current deployment still running. Then you send a small percentage of traffic to the new deployment and monitor it.

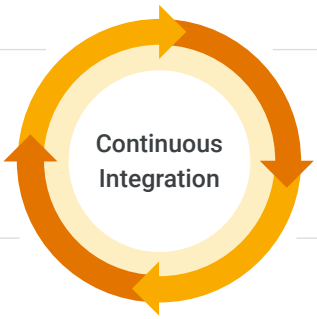
Once you have confidence in your new deployment, you can route more traffic to the new deployment until 100% is routed this way.

- In Compute Engine, you can create a new instance group and add it to the load balancer as an additional backend.
- In Kubernetes, you can create a new pod with the same labels as the existing pods. The service will automatically divert a portion of the requests to the new pod.
- In App Engine, you can again use the traffic splitting feature to drive a portion of traffic to the new version.

Cost Planning

Cost planning is an important phase in your design that starts with capacity planning.

Capacity planning is a continuous, iterative cycle



Forecast
Estimate capacity needed
Monitor Repeat

Allocate
Determine resources required to meet forecasted capacity

Deploy
Monitor to see how accurate your forecasts were

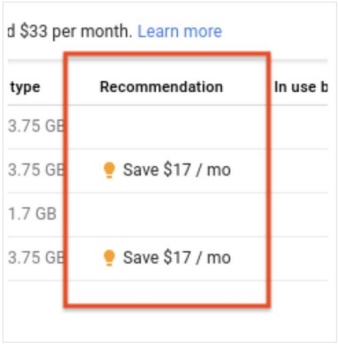
Approve
Cost estimation versus risks and rewards

I recommend that you treat capacity planning not as a one off task, but as a continuous, iterative cycle, as illustrated on this slide.

Start with a forecast that estimates the capacity needed. Monitor and review this forecast. Then allocate by determining the resources required to meet the forecasted capacity. This allows you to estimate costs and balance them against risks and rewards. Once the design and cost is approved, deploy your design and monitor it to see how accurate your forecasts were. This feeds into the next forecast as the process repeats.

Optimizing cost of compute

- Start with small VMs, and test to see whether they work.
- Consider more small machines with auto scaling turned on.
- Consider committed use discounts.
- Consider at least some Spot VMs:
  - 60-91% discounts
  - Use a managed instance group to recreate VMs when they are preempted.
- Google Cloud rightsizing recommendations will alert you when VMs are underutilized.



A good starting point for anybody working on cost optimization is to become familiar with the VM instance pricing. It is often beneficial to start with a couple of small machines that can scale out through auto scaling as demand grows.

To optimize the cost of your virtual machines, consider using committed use discounts, as these can be significant. Also,if your applications are fault tolerant and can withstand possible instance preemptions, then Spot instances can reduce your Compute Engine costs by up to 91%.

Compute Engine provides sizing recommendations for your VM instances, as shown on the right. This is a really useful feature that can help you select the right size of VM for your workloads and optimize costs.

For more details on Spot VMs see the Google Cloud documentation.

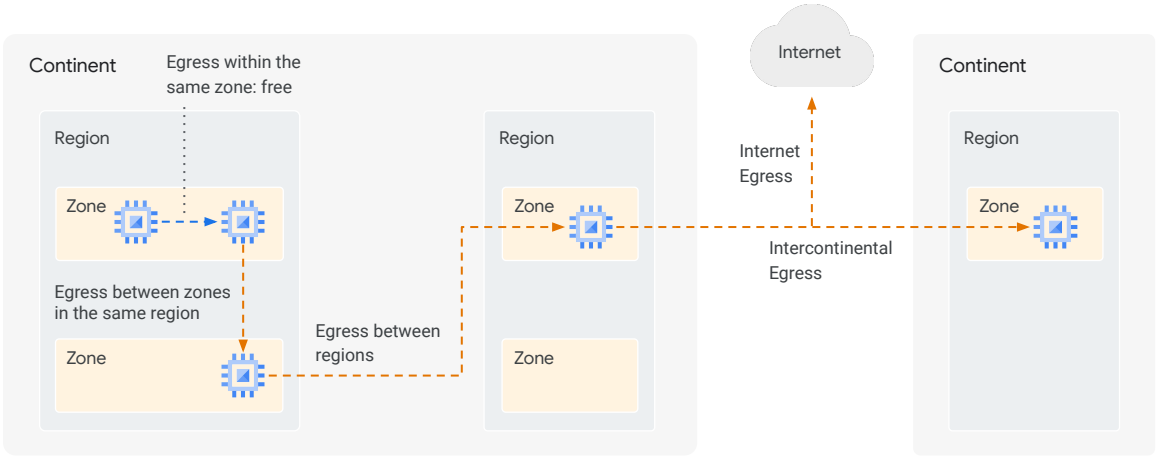
Optimizing disk cost

- Don't over-allocate disk space.
- Determine what performance characteristics your applications require:
  - I/O Pattern: small reads and writes or large reads and writes
  - Configure your instances to optimize storage performance.
- Depending on I/O requirements, consider Standard over SSD disks.

Monthly capacity	Standard PD	SSD PD
10 GB	$0.40	$1.70
1 TB	$40	$170
16 TB	$655.36	$5,570.56

A common mistake is to over-allocate disk space. This is not cost-efficient, but selecting a disk is not just about size. It is important to determine the performance characteristics your applications display: the I/O patterns, do you have large reads, small writes, vice versa, mainly only read data? This type of information will help you select the correct type of disk. As the table shows, SSD persistent disks are significantly more expensive than standard persistent disks. Understanding your I/O patterns can help provide significant savings.

To optimize network costs, keep machines close to your data

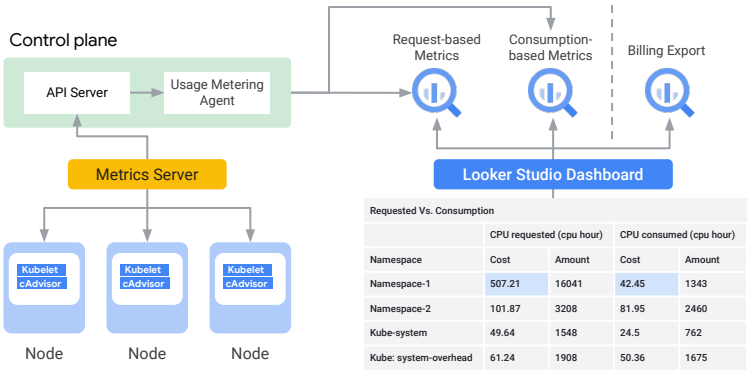


To optimize network costs, it is best practice to keep machines as close as possible to the data they need to access. This graphic shows the different types of egress: within the same zone, between zones in the same region, intercontinental egress, and internet egress. It is important to be aware of the egress charges. These are not all straightforward. Egress in the same zone is free. Egress to a different Google Cloud service within the same region using an external IP address or an internal IP address is free, except for some services such as Memorystore for Redis. Egress between zones in the same region is charged and all internet egress is charged.

One way to optimize your network costs is to keep your machines close to your data.

GKE usage metering can prevent over-provisioning Kubernetes clusters

Compares requested resources with consumed resources.



Another way to optimize cost is to leverage GKE usage metering, which can prevent over-provisioning your Kubernetes clusters.

With GKE usage metering, an agent collects consumption metrics in addition to resource requests by polling PodMetrics objects from the metrics server. The resource request records and resource consumption records are exported to two separate tables in a BigQuery dataset that you specify. Comparing requested with consumed resources makes it easy to spot waste and take corrective measures.

This graphic shows a typical configuration where BigQuery is used for request-based metrics collected from the usage metering agent and, together with data obtained from billing export, it is analyzed in a Looker Studio dashboard.

Compare the costs of different storage alternatives before deciding which one to use

Choose a storage service that meets your capacity requirements at a reasonable cost:

- Storing 1GB in Firestore is free.
- Storing 1GB in Bigtable would be around $500/month.

Earlier in this course, we talked about all of the different storage services. It's important to compare costs of the different options as well as their characteristics.

In other words, your storage and database service choice can make a significant difference on your bill.

Consider alternative services to save cost rather than allocating more resources

- CDN
- Caching
- Messaging
- Queueing
- Etc.

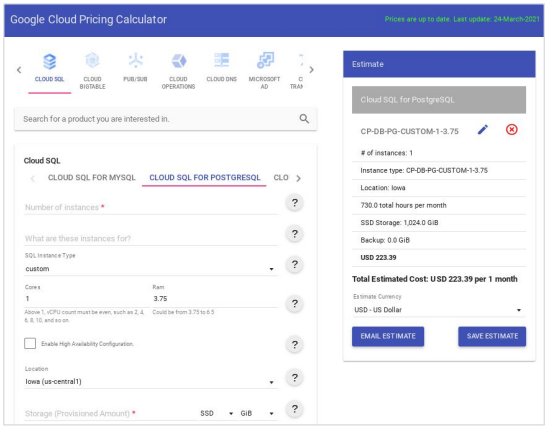
Your architectural design can also help optimize your costs.

For example, if you use Cloud CDN for static content or Memorystore as a cache, you can save instead of allocating more resources. Similarly, instead of using a datastore between two applications, consider message/queuing with Pub/Sub to decouple communication between services and reduce storage needs.

Use the Google Cloud Pricing Calculator to estimate costs

- Base your cost estimates on your forecasting and capacity planning.
- Compare the costs of different compute and storage services.

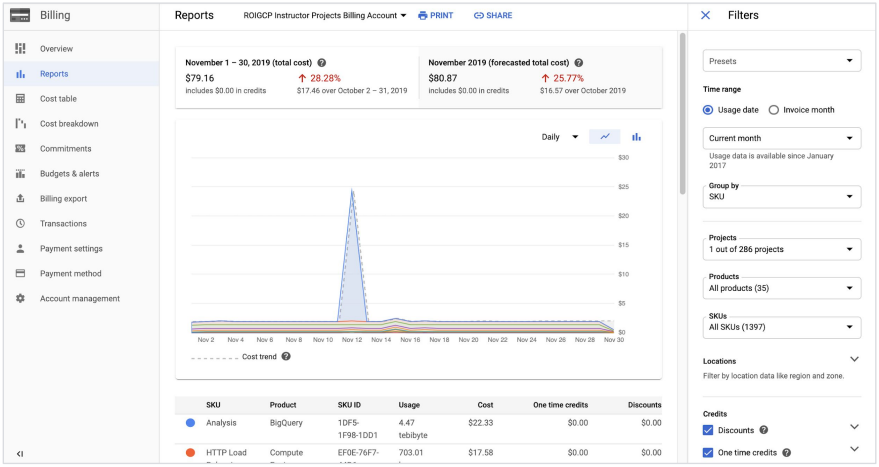
https://cloud.google.com/products/calculator



The pricing calculator should be your go-to resource for estimating costs. Your estimates should be based on your forecasting and capacity planning. The tool is great for comparing costs between compute and storage services, and you will use it in the upcoming design activity.

[Google Cloud Pricing Calculator: https://cloud.google.com/products/calculator]

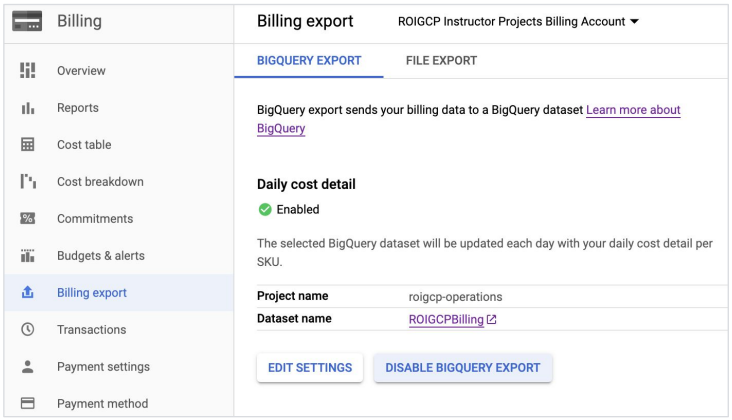
Billing reports provide detailed cost breakdowns



To monitor the cost of your existing service, leverage the Cloud Billing Reports page as shown here. This report shows the changes in cost compared to the previous month, and you can use the filters to search for particular projects, products, and regions, as shown on the right.

The sizing recommendations for your Compute Engine instances will also be in this report.

For advanced cost analysis, export billing data to BigQuery

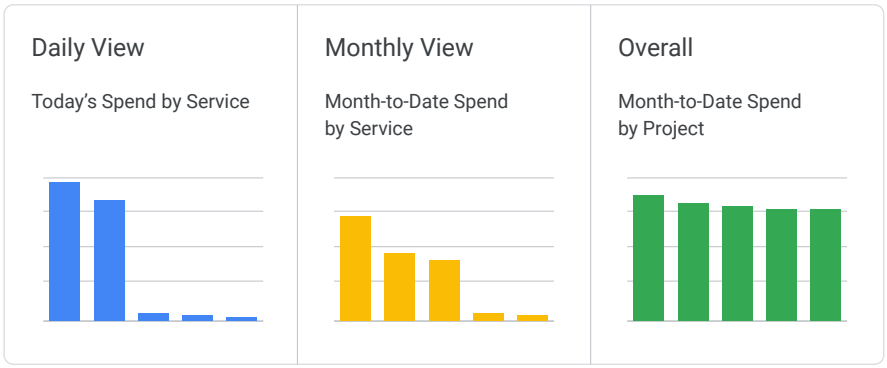


For advanced cost analysis I recommend exporting your billing data to BigQuery, as shown in this screenshot. You can then analyze the billing data to identify large expenses and optimize your Google Cloud spend.

For example, let's assume you label VM instances that are spread across different regions. Maybe these instances are sending most of their traffic to a different continent, which could incur higher costs. In that case, you might consider relocating some of those instances or using a caching service like Cloud CDN to cache content closer to your users, which reduces your networking spend.

Visualize spend with Looker Studio

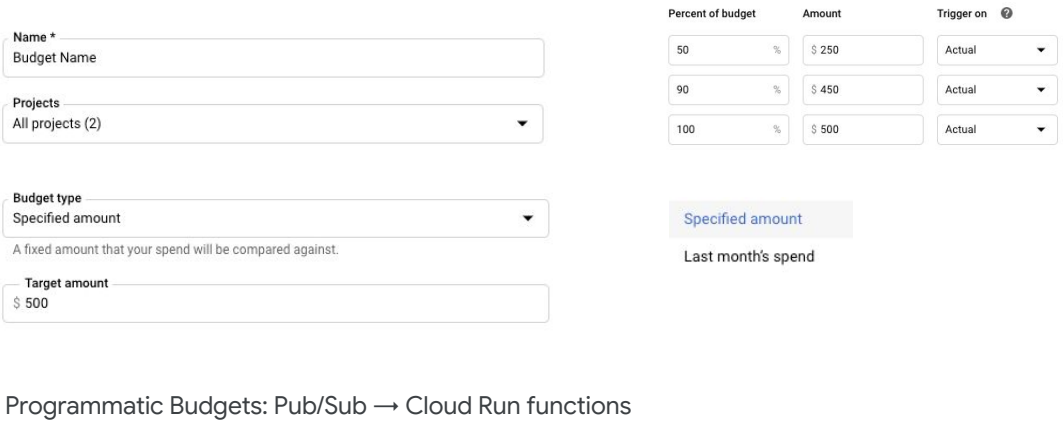
Billing Dashboard



You can even visualize spend over time with Looker Studio, which turns your data into informative dashboards and reports that are easy to read, easy to share, and fully customizable.

The service data is displayed in a daily and monthly view, providing at-a-glance summaries that can also be drilled down to provide greater insights.

Set budgets and alerts to keep your team aware of how much they are spending



Programmatic Budgets: Pub/Sub → Cloud Run functions

To help with project planning and controlling costs, you can set a budget. Setting a budget lets you track how your spend is growing towards that amount.

1. Set a budget name and specify which project this budget applies to.
2. Set the budget at a specific amount or match it to the previous month's spend.
3. Set the budget alerts. These alerts send emails to Billing Admins after spend exceeds a percentage of the budget or a specified amount.

In our case, it would send an email when spending reaches 50%, 90%, and 100% of the budget amount. You can even choose to send an alert when the spend is forecasted to exceed the percent of the budget amount by the end of the budget period.

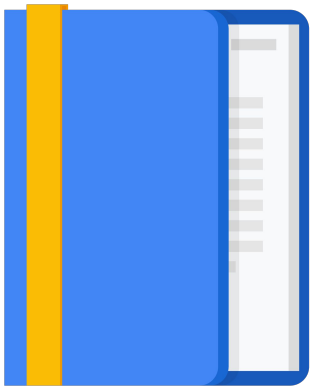
In addition to receiving an email, you can use Pub/Sub notifications to programmatically receive spend updates about this budget. You could even create a Cloud Run function that listens to a Pub/Sub topic to automate cost management.

Agenda

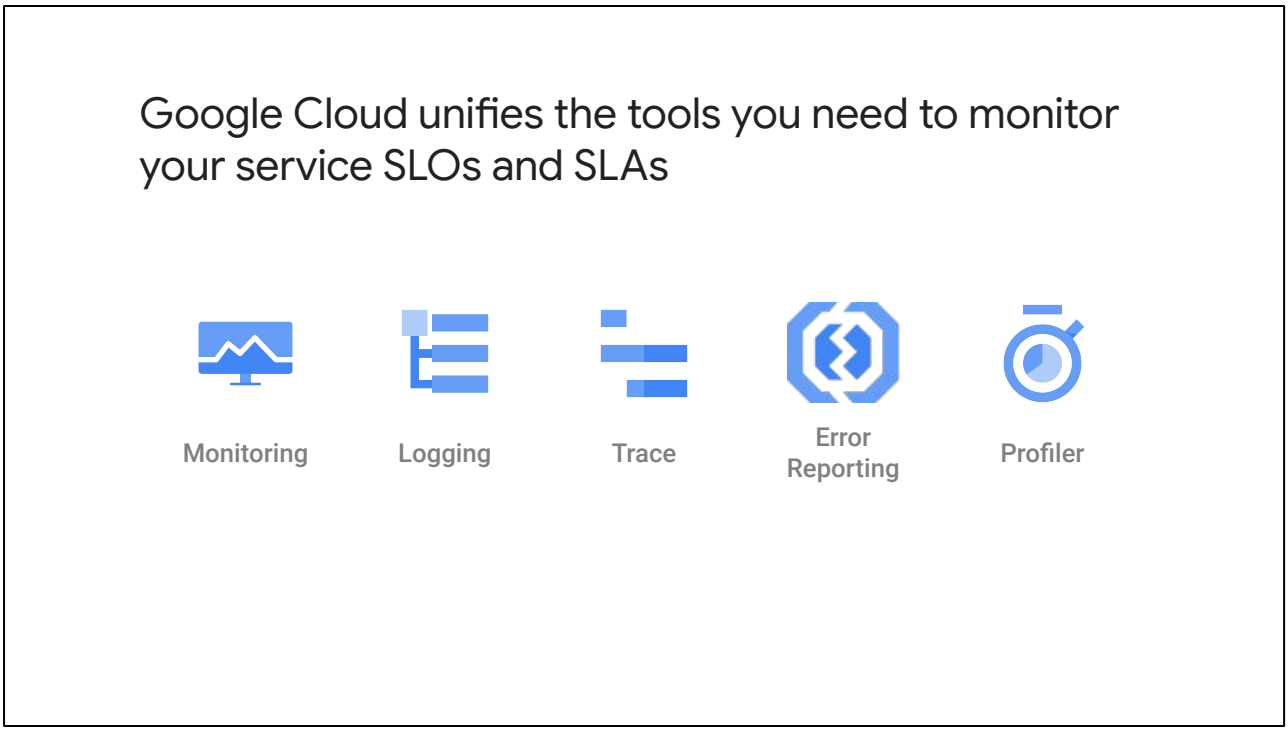
Managing Versions

Cost Planning

Monitoring Dashboards



Let’s get into monitoring and visualizing information with dashboards.

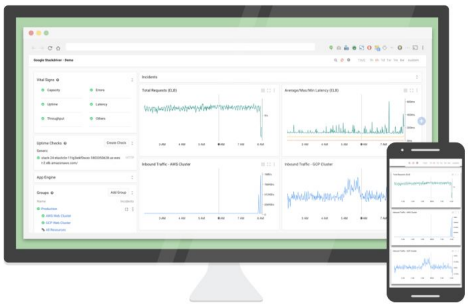


Google Cloud unifies the tools you need to monitor your service SLOs and SLAs in real time.

These tools include Monitoring, Logging, Trace, Error Reporting, and Profiler. All of these enable you to gain the insights you need to achieve your SLOs and determine the root cause in those rare cases that you do not achieve your SLOs.

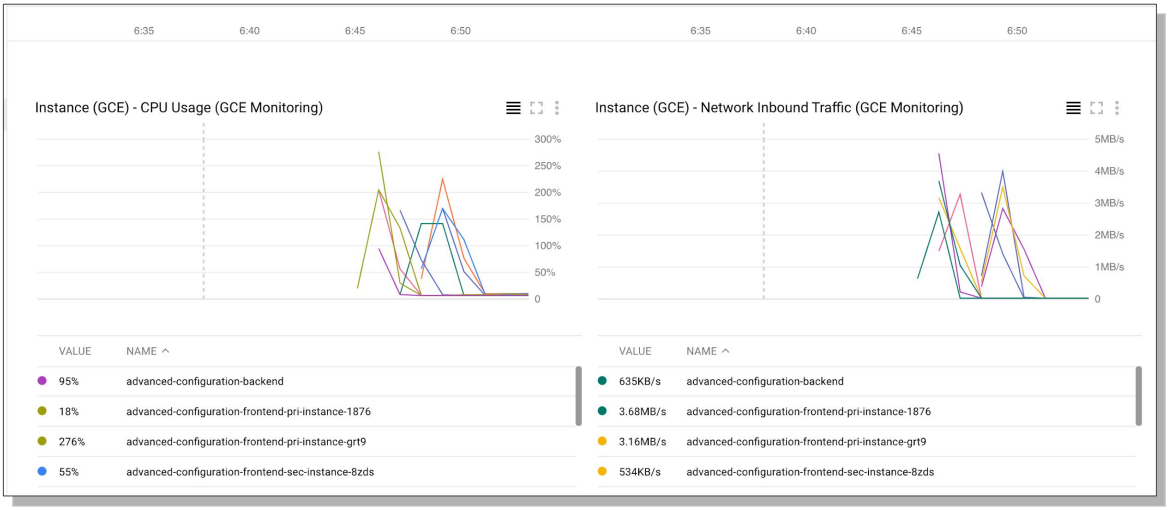
Monitoring dashboards monitor your services

- Monitor the things you pay for:
  - CPU use
  - Storage capacity
  - Reads and writes
  - Network egress
  - Etc.
- Monitor your SLIs to determine whether you are meeting your SLOs.



Dashboards are one way for you to view and analyze metric data that is important to you. This includes your SLIs to ensure that you are meeting your SLAS. The Monitoring page of the Cloud Console automatically provides predefined dashboards for the resources and services that you use. It is important that you monitor the things you pay for to determine trends, bottlenecks, and potential cost savings.

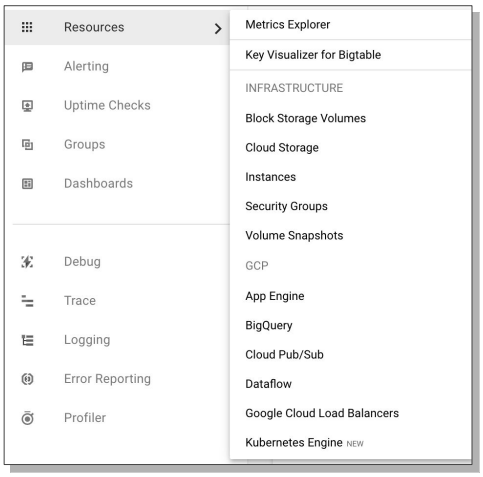
Example charts in a Monitoring dashboard



Here is an example of some charts in a Monitoring dashboard. On the left you can see the CPU usage for different Compute Engine instances, and on the right is the ingress traffic for those instances.

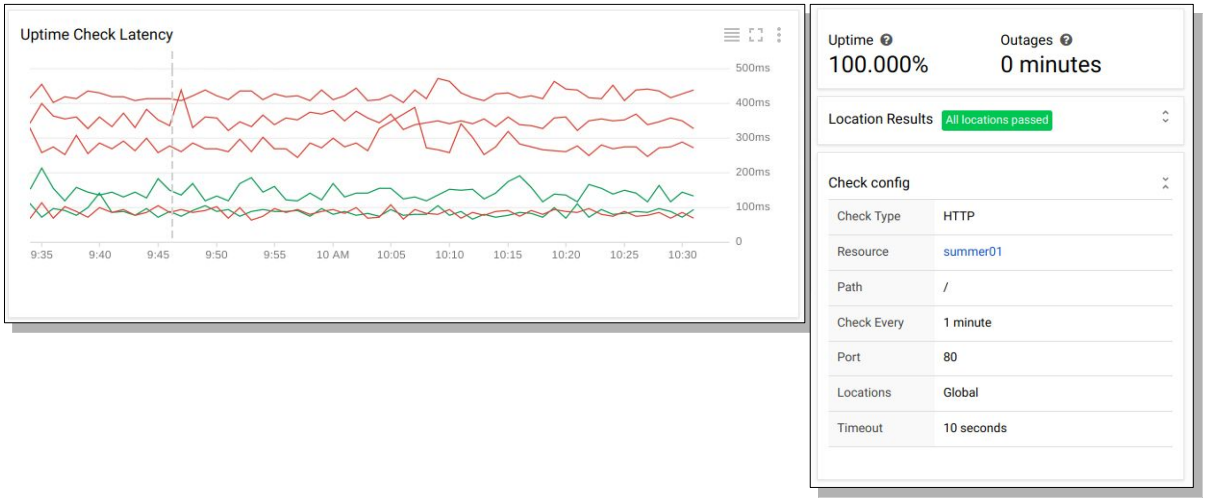
Charts like these provide valuable insights into usage patterns.

To help you get started, Cloud Monitoring creates default dashboards for your project resources



To help you get started, Cloud Monitoring creates default dashboards for your project resources, as shown in this screenshot. You can also create custom dashboards, which you can explore in the upcoming lab.

Create uptime checks to monitor availability and latency

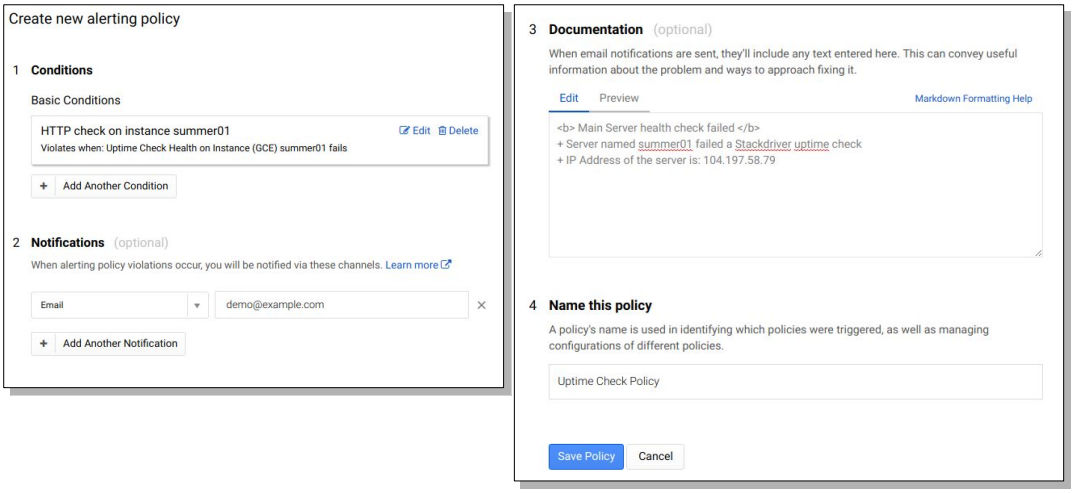


Now, it's a good idea to monitor latency, because it can quickly highlight when problems are about to occur. As shown on this slide, you can easily create uptime checks to monitor the availability and latency of your services. So far there is a 100% uptime with no outages.

Latency is actually one of the four golden signals called out in Google's site reliability engineering, or SRE, book. SRE is a discipline that applies aspects of software engineering to operations whose goals are to create ultra-scalable and highly reliable software systems. This discipline has enabled Google to build, deploy, monitor, and maintain some of the largest software systems in the world.

I've linked the SRE book in the slides of this module [https://landing.google.com/sre/books/].

Create alerts when your service fails to meet your SLOs



Your SLO will be more strict than your SLA, so it is important to be alerted when you are not meeting an SLO because its an early warning that the SLA is under threat.

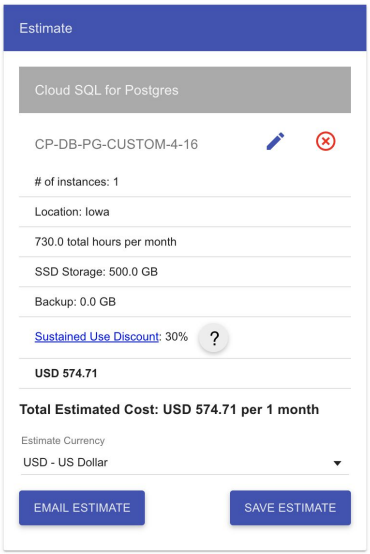
Here is an example of what creating an alerting policy looks like. On the left, you can see an HTTP check condition on the summer01 instance. This will send an email that is customized with the content of the documentation section on the right.

Activity 13: Cost estimating and planning

Refer to your Design and Process Workbook.

- Use the price calculator to create an initial estimate for deploying your case study application.

In this design activity, use Google Cloud's pricing calculator to create an initial estimate for deploying your case study application.



The pricing calculator gives you a form for each service, which you fill out to estimate the cost of using that service. For example, in this screenshot I calculated the cost of one custom SQL instance with 4 cores, 16 GB of RAM, and 500 GB of SSD storage. This could represent the orders database of my online travel application.

Some of these estimates aren't easy to generate because you might not know how much data your storage and database services need and how much compute your deployment platforms require. However, it can be more challenging to estimate things like network egress or the number of reads and writes. Start with a rough estimate and refine it as your capacity plans improve.

Refer to activity 13 in your workbook for similar cost estimates for your case study.

Review Activity 13: Cost estimating and planning

- Use the price calculator to create an initial estimate for deploying your case study application.

In this activity, you were asked to use the Google Cloud pricing calculator to estimate the cost of your case study application.

Service name	Google Cloud Resource	Monthly cost
Orders	Cloud SQL	$1264.44
Inventory	Firestore	$ 215.41
Inventory	Cloud Storage	$1801.00
Analytics	BigQuery	$ 214.72

Here's a rough estimate for the database applications of my online travel portal, ClickTravel.

I adjusted my orders database to include a failover replica for high availability and came up with some high-level estimates for my other services. My inventory service uses Cloud Storage to store JSON data stored in text files. Because this is my most expensive service, I might want to reconsider the storage class or configure object lifecycle management.

Again, this is just an example, and your costs would depend on your case study.

45 minutes

Lab

Monitoring Applications in Google Cloud

Objectives

- Examine the Cloud Logs.
- View Profiler Information.
- Explore Cloud Trace.
- Monitor Resources using Dashboards.
- Create Uptime Checks and Alerts.

We started this course with a discussion on defining SLOs and SLIs for your services. This helps with the detailed design and architecture and helps developers know when they are done implementing a service.

However, the SLIs and SLOs aren't very useful if you don't monitor your applications to see whether you are meeting them. That's where the monitoring tools come in. In this lab you will see how to use some of these tools.

Specifically, you will examine logs, view Profiler information, explore tracing, monitor your resources using Dashboards, and create Uptime Checks and Alerts.

Lab review

Monitoring Applications in Google Cloud

In this lab, you saw how to monitor your applications using built-in Google Cloud tools. First, you deployed an application to App Engine and examined Cloud logs. Then, you viewed Profiler information and explore Cloud Trace. Last but not least, you monitored your application with dashboards and created uptime checks and alerts.

You can stay for a lab walkthrough, but remember that Google Cloud's user interface can change, so your environment might look slightly different.

Review

Maintenance and Monitoring

In this module you learned about managing new versions of your microservices using rolling updates, canary deployments, and blue/green deployments. It's important when deploying microservices that you deploy new versions with no downtime, but also that the new versions don't break the clients that use your services.

You also learned about cost planning and optimization, and you estimated the cost of running your case study application.

You finished the module by learning how to leverage the monitoring tools provided by Google Cloud. These tools can be invaluable for managing your services and monitoring your SLIs and SLOs.

Review

Reliable Cloud Infrastructure: Design and Process

Thank you for taking the “Reliable Cloud Infrastructure: Design and Process” course! We hope you have a better understanding of how to design applications and services that make best use of the platform services provided by Google Cloud.

We also hope that the design activities and labs made you feel more comfortable with design and process in Google Cloud.

Now it’s your turn. Go ahead and apply what you have learned by designing your own applications, deployments, and monitoring.

See you next time!