## Reference from Cloud Architect Learning Path

* 1.Tour\_GC
* 2. GC\_Fundamentals\_Core\_Infrastructure
* 3.Create and Manage Cloud Resources
* 4.Perform Foundational Infrastructure Tasks in GC
* 5.Essential GC Infrastructure Foundation
* 6.Essential GC Infrastructure Core Services
* 7.Elastic GC Infrastructure Scaling and Automation
* 8.Reliable GC Infrastructure Design and Process
* 9.Set Up and Configure a Cloud Environment in GC
* 10.Automating Infrastructure on GC with Terraform
* 11.Getting Started with Google Kubernetes Engine

## 12. Preparing for the Associate Cloud Engineer Exam

## 1 About the Associate Cloud Engineer Certification

* Devising a study strategy, part 1 2 minutes - https://youtu.be/XKSkyxlMtoA
* Devising a study strategy, part 2 4 minutes - https://youtu.be/Mh5chpRDBt4
* Devising a study strategy, part 3 3 minutes - https://youtu.be/BiMqvmEP2Dw
* Are you ready? 1 minute - https://youtu.be/jdzRARGbzmk
* Taking the exam 2 minutes- https://youtu.be/g3O9TSY9IHw
* Resource links for Module 1

## 2 Setting up a Cloud Solutions Environment

* Setting up a Cloud Solutions Environment
* About cloud projects and accounts, part 1 7 minutes - https://youtu.be/m3LnPjku8UQ
* About cloud projects and accounts, part 2 2 minutes - https://youtu.be/3ri8gxv1T8s
* About cloud projects and accounts, part 3 1 minute - https://youtu.be/-nwcs2\_pbJM
* Billing management overview 1 minute - https://youtu.be/J2W\_0lUX-SU
* Command-line interface 1 minute - https://youtu.be/hCjFh53lyVo
* Tour Qwiklabs and the Google Cloud Platform [ACE] 30 minutes
* Resource links for Module 2

## 3 Planning and Configuring a Cloud Solution

* Budgeting and planning with the Pricing Calculator 1 minute - https://youtu.be/VE5a6EB4GNE
* Pricing Calculator demo 1 minute - https://youtu.be/VE5a6EB4GNE
* Planning and configuring compute resources 1 minute - https://youtu.be/C53xyHjhTpM
* Planning and configuring data storage 7 minutes - https://youtu.be/PrfZcdXMmfc
* Planning and configuring network resources 3 minutes - https://youtu.be/w5iX9sKgHYU
* Setting Up Network and HTTP Load Balancers [ACE] 40 minutes
  + Overview
    - learn the differences between a network load balancer and a HTTP load balancer, and how to set them up for your applications running on Google Compute Engine virtual machines.
    - There are several ways you can load balance in Google Cloud Platform. This lab takes you through the setup of the following load balancers:
      * L4 Network Load Balancer
      * L7 HTTP(s) Load Balancer
  + What you'll do
    - Setup a network load balancer.
    - Setup a HTTP(s) load balancer.
    - experience learning the differences between network load balancers and HTTP load balancers.
  + Set the default region and zone for all resources
    - gcloud config set compute/zone us-central1-a
    - gcloud config set compute/region us-central1
  + Create multiple web server instances
    - To simulate serving from a cluster of machines, create a simple cluster of Nginx web servers to serve static content using Instance Templates and Managed Instance Groups. Instance Templates define the look of every virtual machine in the cluster (disk, CPUs, memory, etc). Managed Instance Groups instantiate a number of virtual machine instances using the Instance Template.
    - To create the Nginx web server clusters, create the following:
      * A startup script to be used by every virtual machine instance to setup Nginx server upon startup
      * An instance template to use the startup script
      * A target pool
      * A managed instance group using the instance template
    - create a startup script to be used by every vm instance. it sets up the Nginx server upon startup:
      * cat << EOF > startup.sh
      * #! /bin/bash
      * apt-get update
      * apt-get install -y nginx
      * service nginx start
      * sed -i -- 's/nginx/Google Cloud Platform - '"\$HOSTNAME"'/' /var/www/html/index.nginx-debian.html
      * EOF
    - Create an instance template, which uses the startup script:
      * gcloud compute instance-templates create nginx-template \
      * --metadata-from-file startup-script=startup.sh
    - Create a target pool. A target pool allows a single access point to all the instances in a group and is necessary for load balancing in the future steps.
      * gcloud compute target-pools create nginx-pool
    - Create a managed instance group using the instance template:
      * gcloud compute instance-groups managed create nginx-group \
      * --base-instance-name nginx \
      * --size 2 \
      * --template nginx-template \
      * --target-pool nginx-pool
    - This creates 2 VM instances with names that are prefixed with nginx-.
    - List the compute engine instances and you should see all of the instances created:
      * gcloud compute instances list
    - Now configure a firewall so that you can connect to the machines on port 80 via the EXTERNAL\_IP addresses:
      * gcloud compute firewall-rules create www-firewall --allow tcp:80
    - access via http://EXTERNAL\_IP/
  + Create a Network Load Balancer
    - Network load balancing allows you to balance the load of your systems based on incoming IP protocol data, such as address, port, and protocol type. You also get some options that are not available, with HTTP(S) load balancing. For example, you can load balance additional TCP/UDP-based protocols such as SMTP traffic. And if your application is interested in TCP-connection-related characteristics, network load balancing allows your app to inspect the packets, where HTTP(S) load balancing does not.
    - Create an L4 network load balancer targeting your instance group:
      * gcloud compute forwarding-rules create nginx-lb \
      * --region us-central1 \
      * --ports=80 \
      * --target-pool nginx-pool
    - List all Google Compute Engine forwarding rules in your project.
      * gcloud compute forwarding-rules list
    - access the load balancer from the browser <http://IP_ADDRESS/>
  + Create a HTTP(s) Load Balancer
    - HTTP(S) load balancing provides global load balancing for HTTP(S) requests destined for your instances. You can configure URL rules that route some URLs to one set of instances and route other URLs to other instances. Requests are always routed to the instance group that is closest to the user, provided that group has enough capacity and is appropriate for the request. If the closest group does not have enough capacity, the request is sent to the closest group that does have capacity.
    - First, create a health check. Health checks verify that the instance is responding to HTTP or HTTPS traffic:
      * gcloud compute http-health-checks create http-basic-check
    - Define an HTTP service and map a port name to the relevant port for the instance group. Now the load balancing service can forward traffic to the named port:
      * gcloud compute instance-groups managed \
      * set-named-ports nginx-group \
      * --named-ports http:80
    - Create a backend service:
      * gcloud compute backend-services create nginx-backend \
      * --protocol HTTP --http-health-checks http-basic-check --global
    - Add the instance group into the backend service:
      * gcloud compute backend-services add-backend nginx-backend \
      * --instance-group nginx-group \
      * --instance-group-zone us-central1-a \
      * --global
    - Create a default URL map that directs all incoming requests to all your instances:
      * gcloud compute url-maps create web-map \
      * --default-service nginx-backend
    - Create a target HTTP proxy to route requests to your URL map:
      * gcloud compute target-http-proxies create http-lb-proxy \
      * --url-map web-map
    - Create a global forwarding rule to handle and route incoming requests. A forwarding rule sends traffic to a specific target HTTP or HTTPS proxy depending on the IP address, IP protocol, and port specified. The global forwarding rule does not support multiple ports.
      * gcloud compute forwarding-rules create http-content-rule \
      * --global \
      * --target-http-proxy http-lb-proxy \
      * --ports 80
    - After creating the global forwarding rule.
      * gcloud compute forwarding-rules list
    - Take note of the http-content-rule IP\_ADDRESS for the forwarding rule.
    - Access http://IP\_ADDRESS/.
  + Network Load Balancing is a regional, non-proxied load balancer.
    - True
* Resource links for Module 3

## 4 Deploying and Implementing a Cloud Solution

* Deploying and implementing compute resources 3 minutes - https://youtu.be/q5TbzN5LEDU
* Deploying and implementing Kubernetes resources 4 minutes - Deploying and Implementing a Cloud Solution
* Deploying App Engine and Cloud Function resources 3 minutes - https://youtu.be/NW4vAJ9qTVE
* Deploying and implementing data solutions 2 minutes - https://youtu.be/kaiee-4xnw0
* Deploying and implementing networking resources 2 minutes - https://youtu.be/aYiDnV5\_vFo
* Creating an auto mode VPC network with a subnet demo 1 minute - https://youtu.be/6DII1-gIPD4
* Creating a custom VPC network with a subnet demo 1 minute - https://youtu.be/QhgqL3hCjVw
* Deploying a solution with Cloud Launcher 2 minutes - https://youtu.be/JUii-IqSGc4
* Deploying an application using Deployment Manager 1 minute - https://youtu.be/\_qPIhmbwebs
* Deployment Manager - Full Production [ACE] 1 hour 30 minutes
  + Overview
    - you will launch a service using an infrastructure orchestration tool called Deployment Manager and monitor the service using Cloud Monitoring. In Cloud Monitoring, you will set up basic black box monitoring with a Cloud Monitoring dashboard and establish an Uptime Check (alert notification) to trigger incident response.
    - More specifically, you will:
      * Install and configure an advanced deployment using Deployment Manager sample templates.
      * Enable Cloud monitoring.
      * Configure Cloud Monitoring Uptime Checks and notifications.
      * Configure a Cloud Monitoring dashboard with two charts, one showing CPU usage and the other ingress traffic.
      * Perform a load test and simulate a service outage.
    - Cloud Monitoring provides visibility into the performance, uptime, and overall health of cloud-powered applications. Cloud Monitoring collects metrics, events, and metadata from Google Cloud Platform, Amazon Web Services, hosted uptime probes, application instrumentation, and a variety of common application components including Cassandra, Nginx, Apache Web Server, Elasticsearch, and many others. Cloud Monitoring ingests that data and generates insights via dashboards, charts, and alerts. Cloud Monitoring alerting helps you collaborate by integrating with Slack, PagerDuty, HipChat, Campfire, and more.
  + Objectives
    - Launch a cloud service from a collection of templates.
    - Configure basic black box monitoring of an application.
    - Create an uptime check to recognize a loss of service.
    - Establish an alerting policy to trigger incident response procedures.
    - Create and configure a dashboard with dynamically updated charts.
    - Test the monitoring and alerting regimen by applying a load to the service.
    - Test the monitoring and alerting regimen by simulating a service outage.
  + Create a virtual environment
    - Execute the following command to download and update the packages list.
      * sudo apt-get update
    - Python virtual environments are used to isolate package installation from the system.
      * sudo apt-get install virtualenv
      * If prompted [Y/n], press Y and then Enter.
      * virtualenv -p python3 venv
    - Activate the virtual environment.
      * source venv/bin/activate
    - If error , try this commnt
      * sudo apt-get remove python3-virtualenv
  + Clone the Deployment Manager Sample Templates
    - Google provides a robust set of sample Deployment Manager templates that you can learn from and build upon.
    - To clone the repository, a directory to hold the Deployment Manager sample templates.
      * mkdir ~/dmsamples
      * cd ~/dmsamples
      * git clone https://github.com/GoogleCloudPlatform/deploymentmanager-samples.git
    - Not all of the subdirectories are independent projects. For example, the directory named common contains templates that are used by several of the other projects. If you are studying independently later, use the README files as a guides.
    - The nodejs directory contains everything you'll need to build this lab. Note that there is a nodejs directory and a nodejs\_l7directory. Use nodejs.
    - Navigate to and list the version2 examples:
      * cd nodejs/python
    - The main deployment manager configuration file is nodejs.yaml. It makes use of templates to generate infrastructure. The rest of the files are templates. Templates use variables defined in the nodejs.yaml configuration file to produce customized results.
      * Diagram

        Description automatically generated
    - frontend.py
      * frontend.py includes frontend.py.schema, which creates an instance template based on container\_instance\_template.py. This template is used to create a managed instance group and an autoscaler. The template also creates a network load balancer that has a forwarding rule with a single public IP address. It will also create:
        + A target pool that refers to the managed instance group.
        + A health check attached to the target pool.
    - nodejs.py
      * nodejs.py includes nodejs.py.schema, which brings the frontend and backend templates together.
        + Note that the frontend is frontend.py.
        + The backend is /common/python/container\_vm.py.
        + This is a VM running a Docker container with MySQL, so it doesn't require a custom template.
    - Other files
      * /common/python/container\_instance\_template.py
      * /common/python/container\_vm.py
      * /common/python/container\_helper.py
  + Customize the Deployment
    - Now that you've downloaded and reviewed the nodejs Deployment Manager template, let's start customizing the deployment.
    - Specify the zone
      * The nodejs.yaml file requires a zone, and you'll now add one to the file.
    - Open nodejs.yaml in nano so you can edit the zone value:
      * nano nodejs.yaml
    - Replace ZONE\_TO\_RUN with a zone name that's near you, then exit nano and save the file.
    - This example shows ZONE\_TO\_RUN set to us-east1-d
      * resources:
      * - name: nodejs
      * type: nodejs.py
      * properties:
      * zone: us-east1-d
    - Modify the maximum number of instances in the instance group
      * nano nodejs.py
    - Current scaling limit is 20 (refer maxSize). Modify the maxSize and set it to 4:
      * {
      * 'name': frontend,
      * 'type': 'frontend.py',
      * 'properties': {
      * 'zone': context.properties['zone'],
      * 'dockerImage': 'gcr.io/qwiklabs-resources/nodejsservice',
      * 'port': application\_port,
      * # Define the variables that are exposed to container as env variables.
      * 'dockerEnv': {
      * 'SEVEN\_SERVICE\_MYSQL\_PORT': mysql\_port,
      * 'SEVEN\_SERVICE\_PROXY\_HOST': '$(ref.' + backend
      * + '.networkInterfaces[0].networkIP)'
      * },
      * # If left out will default to 1
      * 'size': 2,
      * # If left out will default to 1
      * 'maxSize': 4
      * }
      * },
  + Run the Application
    - Now you'll use Deployment Manager to deploy the application and make it operational. This builds the infrastructure, but it won't allow traffic. After Deployment Manager sets up the infrastructure, you can apply service labels.
  + Deploy the application
    - Enter this command to name the application advanced-configuration and pass Deployment Manager the configuration file (nodejs.yaml).
      * gcloud deployment-manager deployments create advanced-configuration --config nodejs.yaml
    - Confirm the limit for the maximum number of instances > Instance groups > Click > Details
      * Graphical user interface, text, application, email

        Description automatically generated
  + Find the global load balancer forwarding rule IP address
    - Enter the following command in the Cloud Command Line to find your forwarding IP address.
      * gcloud compute forwarding-rules list
    - Try http://<your IP address>:8080
    - Try http://<your forwarding IP address>:8080/?msg=<enter\_a\_message>
    - Try http://35.196.56.153:8080/?msg=my dog has spots
    - Try http://<your IP address>:8080.
  + Create a Monitoring workspace
    - You will now setup a Monitoring workspace that's tied to your Qwiklabs GCP Project. The following steps create a new account that has a free trial of Monitoring.
    - In the Google Cloud Platform Console, click on Navigation menu > Monitoring.
    - When the Monitoring dashboard opens, your workspace is ready.
      * Graphical user interface

        Description automatically generated
  + Configure an uptime check and alert policy in Cloud Monitoring
    - Now that Cloud Monitoring is running, you'll set up alerts and a dashboard.
    - Configure an uptime check
      * Uptime checks, and then click + Create Uptime Check.
      * Set the following fields:
        + Title: Check1, then click Next.
        + Protocol: TCP
        + Resource Type: URL
        + Hostname: <your forwarding address>
        + Path: 8080
        + Check Frequency: 1 min
    - Click on Next to leave the other details to default and click Test to verify that your uptime check can connect to the resource.
    - When you see a green check mark everything can connect. Click Create.
    - The uptime check you configured takes a while for it to become active. Continue with the lab, you'll check for results later. While you wait, create an alerting policy for a different resource.
  + Configure an alerting policy and notification
    - Click on Alerting > + Create Policy. Click on Add Condition.
    - For Find resource type and metric, select VM Instance.
    - In the Metric field, select the CPU usage or CPU utilization.
    - Under Configuration, for Condition, select is above.
    - Specify the threshold value and for how long the metric must cross this set value before the alert is triggered. For example, for THRESHOLD, type 20 and set FOR to 1 minute.
    - Click Add. Click on Next. Notification Channels, then click on Manage Notification Channels.
    - Add Email. Refresh Notification Channels & select Email
    - Click on Save.
  + Configure a Dashboard with a Couple of Useful Charts
    - Configure a dashboard
    - On the Cloud Monitoring window or tab, click on Dashboards. Click +Create Dashboard.
    - For the New Dashboard Name, type DMDash.
    - Click Line Chart.
    - Configure the chart as follows:
      * Property Value
      * Title Sample
      * Resource type VM Instance
      * Metric Type CPU Usage
    - Click on Add Chart to add another chart to the dashboard.
    - Click Line Chart.
    - Configure the chart as follows:
      * Property Value
      * Title Sample2
      * Resource type VM Instance
      * Metric Type Sent Bytes
    - The Completed Dashboard should appear similar to the screenshot below:
  + Create a test VM with ApacheBench
    - Now that you've configured monitoring for traffic in a specified region, see if it works. You'll install and use ApacheBench to apply 3 levels of load to the service and then view the Cloud Monitoring Dashboard you've set up.
    - In the Cloud Console, click Compute Engine > VM instances. > Click Create.
    - Install ApacheBench
      * sudo apt-get update
      * sudo apt-get -y install apache2-utils
    - Apply and monitor load
      * Now you'll use ApacheBench to apply load to the service. Watch the DMDash dashboard in Cloud Monitoring to monitor the CPU usage and the Network Inbound Traffic. You'll also be able to track the number of instances in Cloud Monitoring by mousing over the lines, or by viewing the instances in the Cloud Console (Navigation menu > Compute Engine > VM).
    - SSH window, enter this command for ApacheBench to apply load to the service. Replace your forwarding IP for Your\_IP. Run the following command two or three times to create traffic.
      * ab -n 1000 -c 100 http://<Your\_IP>:8080/
      * ab -n 5000 -c 100 http://<Your\_IP>:8080/
      * ab -n 10000 -c 100 http://<Your\_IP>:8080/
    - Now see what happens when you lower the CPU usage per instance.
      * In the cloud console, click Navigation menu > Compute Engine > Instance groups.
      * Click on the name of your instance group, then Edit Group.
      * Click on pencil icon of CPU utilization. Change the Target CPU utilization to 20 then click Done.
      * Click Save.
    - The target CPU usage is the total value of the CPU usage for all VMs in the instance group. This controls when autoscaling occurs. In production you would usually have this set to at least 60%. For this exercise you will set it temporarily to 20% to make it quicker to examine autoscaling.
    - Run this command two or three times to create traffic:
      * ab -n 10000 -c 100 http://<Your\_IP>:8080/
    - Expected behavior: The load consumed more than 20% of the cumulative CPU in the group, triggering autoscaling. A new instance was started.
    - Now see what happens when you turn autoscaling off.
      * Go to Compute Engine > Instance groups.
      * Click the name of your instance group, then Edit Group.
      * Change Autoscaling mode to Don't autoscale.
      * Click Save.
    - Wait a few minutes, then run this command two or three times to create traffic:
      * ab -n 10000 -c 100 http://<Your\_IP>:8080/
    - Expected behavior: With autoscaling off, no new instances are created, cumulative CPU usage increases.
  + Simulate a Service Outage
    - To simulate an outage, remove the firewall.
    - Click Navigation menu > VPC Networks > Firewall.
      * Check the box next to tcp:8080 protocols / ports, then click Delete at the top of the page. Click Delete again to confirm.
    - You will receive a notification email in 15 to 30 minutes.
  + Cloud Monitoring collects metrics, events, and metadata from Google Cloud Platform, Amazon Web Services, hosted uptime probes, application instrumentation, and a variety of common application components including Cassandra, Nginx, Apache Web Server, Elasticsearch, and many others.
    - True
* Resource links for Module 4

## 5 Ensuring the Successful Operation of a Cloud Solution

* Managing Compute Engine resources 3 minutes - https://youtu.be/VJu6gUBZMwg
* Managing Kubernetes Engine resources 1 minute - https://youtu.be/OwMwQA4\_vgE
* Google Kubernetes Engine: Qwik Start [ACE] 30 minutes
* Managing App Engine resources 2 minutes - https://youtu.be/H2NYvm0Y8FU
* Managing data solutions 2 minutes - https://youtu.be/WCOmIhNU9r4
* Managing networking resources 1 minute - https://youtu.be/xZ5ueS3AXsM
* Expanding a subnet IP demo 1 minute - https://youtu.be/YawjKJHYKqo
* Monitoring and logging 2 minutes - https://youtu.be/bOosLsAdDPM
* Resource links for Module 5

## 6 Configuring Access and Security

* Managing Identity and Access Management 1 minute - https://youtu.be/lRiO8vOtFc8
* Managing IAM members demo 1 minute - https://youtu.be/lRiO8vOtFc8
* Defining custom IAM roles 1 minute - https://youtu.be/MkwhSJSlVMg
* Creating custom IAM roles demo 3 minutes - https://youtu.be/rJqblanalVU
* Managing service accounts 3 minutes - https://youtu.be/P-5d7kMgDZQ
* Viewing audit logs for projects and services 1 minute - https://youtu.be/JtfJI8ZJaFQ
* Site Reliability Troubleshooting with Cloud Monitoring APM [ACE] 1 hour 34 minutes
  + Overview
    - The objective of this lab is to familiarize yourself with the specific capabilities of Cloud Monitoring to monitor GKE cluster infrastructure, Istio, and applications deployed on this infrastructure.
    - What you'll do
      * Create a GKE cluster
      * Deploy a microservices application to it
      * Define latency and error SLIs and SLOs for it
      * Configure Cloud Monitoring to monitor your SLIs
      * Deploy a breaking change to the application and use Cloud Monitoring to troubleshoot and resolve the issues that result
      * Validate that your resolution addresses the SLO violation
    - What you'll learn
      * How to deploy a microservices application on an existing GKE cluster
      * How to select appropriate SLIs/SLOs for an application
      * How to implement SLIs using Cloud Monitoring features
      * How to use Cloud Trace, Profiler, and Debugger to identify software issues
  + Infrastructure setup
    - Set the zone in gcloud:
      * gcloud config set compute/zone us-west1-b
      * export PROJECT\_ID=$(gcloud info --format='value(config.project)')
      * Verify that the cluster named shop-cluster has been created:
    - Create a Monitoring workspace
      * In the Google Cloud Platform Console, click on Navigation menu > Monitoring.
      * Wait for your workspace to be provisioned.
      * When the Monitoring dashboard opens, your workspace is ready.
        + Graphical user interface

          Description automatically generated
    - Once your cluster has RUNNING status, get the cluster credentials:
      * gcloud container clusters get-credentials shop-cluster --zone us-west1-b
  + Deploy application
    - deploy a microservices application called Hipster Shop to your cluster to create an actual workload you can monitor.
      * git clone https://github.com/GoogleCloudPlatform/training-data-analyst
    - Create a soft link to your working directory:
      * ln -s ~/training-data-analyst/blogs/microservices-demo-1 ~/microservices-demo-1
    - Download and install skaffold:
      * curl -Lo skaffold https://storage.googleapis.com/skaffold/releases/latest/skaffold-linux-amd64 && chmod +x skaffold && sudo mv skaffold /usr/local/bin
    - Install the app using skaffold:
      * cd microservices-demo-1
      * skaffold run
    - Get the external IP of the application:
      * export EXTERNAL\_IP=$(kubectl get service frontend-external | awk 'BEGIN { cnt=0; } { cnt+=1; if (cnt > 1) print $4; }')
    - Finally, confirm that the app is up and running:
      * curl -o /dev/null -s -w "%{http\_code}\n" http://$EXTERNAL\_IP
    - Note: You may need to run this command a second time if you get a 500 error.
    - Download the source and put the code in the Cloud Source Repo:
      * ./setup\_csr.sh
    - Now that the application has been deployed, set up monitoring for the application.
      * <https://github.com/GoogleCloudPlatform/microservices-demo>
      * https://skaffold.dev/
  + Develop Sample SLOs and SLIs
    - Before implementing any monitoring, review the introduction to the chapter on Service Level Objectives from the SRE Book: https://sre.google/books/
    - It's impossible to manage a service correctly, let alone well, without understanding which behaviors really matter for that service and how to measure and evaluate those behaviors. To this end, we would like to define and deliver a given level of service to our users, whether they use an internal API or a public product.
    - We use intuition, experience and an understanding of what users want to define service level indicators (SLIs), objectives (SLOs) and agreements (SLAs). These measurements describe basic properties of metrics that matter, what values we want those metrics to have and how we'll react if we can't provide the expected service. Ultimately, choosing appropriate metrics helps to drive the right action if something goes wrong and also gives an SRE team confidence that a service is healthy.
    - An SLI is a service level indicator: A carefully defined quantitative measure of some aspect of the level of service that is provided.
    - Most services consider
    - request latency: how long it takes to return a response to a request as a key SLI. Other common SLIs include the error rate, often expressed as a fraction of all requests received and system throughput, typically measured in requests per second. Another kind of SLI important to SREs is availability or the fraction of the time that a service is usable. It is often defined in terms of the fraction of well-formed requests that succeed.
    - Durability: the likelihood that data will be retained over a long period of time is equally important for data storage systems. The measurements are often aggregated: i.e., raw data is collected over a measurement window and then turned into a rate, average, or percentile.
    - Now that you have established a basic understanding, define the SLIs and SLOs for your application. Given that the application itself serves end user ecommerce traffic, it's going to be very important that user experience remains constant and that performance is good. You will monitor SLIs for request latency, error rate, throughput and availability.
    - Application Architecture
      * Diagram

        Description automatically generated
      * It's impossible to develop SLIs without understanding how the application is built. Details are in the original repository, but for this lab, it suffices to understand that:
      * Users access the application through the Frontend.
      * Purchases are handled by CheckoutService.
      * CheckoutService depends on CurrencyService to handle conversions.
      * Other services such as RecommendationService, ProductCatalogService and Adservice are used to provide the frontend with content needed to render the page.
    - Service Level Indicators and Objectives
      * The following SLIs and SLOs are selected based on the end-user experience and the theoretical impact to users and business objectives.

|  |  |  |  |
| --- | --- | --- | --- |
| **SLI** | **Metric** | **Description** | **SLO** |
| Request latency | Front end latency | Measures how long a user is waiting for the page to load. A high latency typically correlates to a negative user experience | 99% of requests from the previous 60 minute period are services in under 3 seconds |
| Error rate | Front end error rate | Measures the error rate experienced by users. A high error rate likely indicates an issue. | 0 Errors in the previous 60 minute period |
| Error rate | Checkout error rate | Measures the error rate experienced by other services calling the checkout service. A high error rate likely indicates an issue. | 0 Errors in the previous 60 minute period |
| Error rate | Currency Service error rate | Measures the error rate experienced by other services calling the currency service. A high error rate likely indicates an issue. | 0 Errors in the previous 60 minute period |
| Availability | Front end success rate | Measures the rate of successful requests as a way to determine the availability of the service. A low success rate likely indicates that users are having a poor experience. | 99% of requests are successful over the previous 60 minute period |

* + Configure Latency SLI
    - Now that you have SLOs and SLIs defined, you can implement cloud monitoring. The metrics you are interested in are already being collected. You will create alerting policies for each of your SLOs.
    - Front End Latency
      * In the Monitoring tab, click Alerting and select Create Policy.
      * Click Add Condition and specify the metric and condition that will be used to trigger the Alerting Policy. The condition will let you know when you're experiencing performance issues that are impacting user experience.
      * As described in the Service Level Indicators and Objectives table above, you will use the 99th percentile front end latency as the SLI.
      * Add the following into the Find resource type and metric field then select the following from the dropdown menu:
        + custom.googleapis.com/opencensus/grpc.io/client/roundtrip\_latency
      * In the Resource Type, select the Global option.
      * Click into the Filter field and select the opencensus\_task item. Click on the first default Value, then click Apply. Next, set the Aggregator to 99th percentile.
        + Graphical user interface, text

          Description automatically generated
      * Next, in the Configuration area, set the options as follows:
        + Condition triggers if Any time series violates
        + Condition: is above
        + Threshold: 500
        + For: Most recent value
        + Table

          Description automatically generated
      * Click Add.
      * Click Next. Skip the Who should be notified? step and click Next.
      * Now, name the alerting policy as Latency Policy in the Alert name field.
      * Click Save. You've configured Cloud Monitoring to monitor your frontend latency SLI!
  + Configure Availability SLI
    - Next, configure Cloud Monitoring to monitor service availability by creating another Alerting Policy.
    - Front End Availability
      * Start by monitoring the error rate for the front end service, since that’s where user experience is going to be most directly impacted. As discussed above, you’re going to consider any failures observed to be an SLO violation. Create an alerting policy that will trigger an incident if any failures are observed.
      * An easy way to trigger on a particular failure is to use log-based metrics.
      * In the Google Cloud Platform Console, click Navigation menu > Logging.
      * Configure the filter as follows:
        + In Resource type select Kubernetes Container.
        + In the filter bar add: labels."k8s-pod/app"="currencyservice"
        + In Severity select ERROR.
        + Graphical user interface, text, application

          Description automatically generated
      * Above Query results section, click Create metric.
      * Name the metric Error\_Rate\_SLI and click Create Metric to save the log based metric:
        + Graphical user interface, text, application, email

          Description automatically generated
      * You now see the metric listed on the Logs-based metrics page. Click Logs-based Metrics. To create an alert for this metric, click the 3 dots at the end of the row and select Create Alert from Metric.
        + Graphical user interface, text, application, email

          Description automatically generated
      * Notice the resource type and metric have already been filled in.
      * NOTE: If the Resource type does not appear itself, in that case select Kubernetes Container as Resource type.
      * Name the condition "Error Rate SLI".
      * Click the Show Advanced Options link and set the following:
        + Aligner: rate
      * In Configuration, use 0.5 as your Threshold for 1 minutes.
      * Then Save the condition.
      * Click Next. Skip the Who should be notified? step and click Next.
      * In the subsequent screen, name your new policy "Error Rate SLI", and Save it.
      * As expected, there are no failures, and your application is currently meeting its availability SLO!
  + Deploy new release
    - Now that you have configured SLI monitoring, you're ready to measure the impact of application changes on user experience. See what happens when you deploy a new release of the application.
    - Next you'll modify the Kubernetes manifests for the services which have new releases and then run skaffold to deploy the application again.
    - Find the microservices-demo-1 folder and open the kubernetes-manifests folder within it.
    - Update the kubernetes manifests to pull the new images by:
    - Replacing the accl-demo image tag with rel013019
    - Adding: imagePullPolicy: Always
    - in the following files:
      * kubernetes\_manifests/recommendationservice.yaml
      * kubernetes\_manifests/currencyservice.yaml
      * kubernetes\_manifests/frontend.yaml
    - As an example, here's the original version of the recommendationservice.yaml file:
      * Graphical user interface

        Description automatically generated with low confidence
    - And here it is after making the update:
      * Text

        Description automatically generated with low confidence
    - Once you've updated and closed each of the files, you're ready to deploy the new version!
    - Deploy New Version
      * skaffold run
  + Send some data
    - Now that the application is running, go look at what you have deployed.
    - In the Console, navigate to Kubernetes Engine > Services & Ingress. Look for the frontend-external service and click on the Endpoint URL.
    - Once on the Hipster Shop website, click on a Buy and/or Add to Cart for a couple of items to send some traffic.
  + Latency SLO Violation - Find the Problem
    - In this exercise you will use Cloud Application Performance Management tools (APM) to identify and resolve an issue causing poor application latency.
    - First see if everything is still OK with the application after deploying the new version.
    - Go to Monitoring Overview page. A Latency Policy incident appears shortly if it hasn't already, please wait a few minutes to see it show up.
    - Graphical user interface, text, application

      Description automatically generated
    - To learn more about what's going on, click Alerting in the left menu, and then click the alert in the Incidents section. You may also click See all incidents and then select the autorefresh arrows so you will always be looking at the latest information. You may need to click on the Acknowledge Incident to see that the alert happened. The best way to analyze latency issues is by using Trace. In the Google Cloud Platform Console, click Trace.
    - You're now in the Console. The initial overview is useful, but you need to get to the next level of detail. Open the Trace List page.
    - Click Auto Reload. Notice the scatter plot at the top of the page and that, around the time of the alert, there are a large number of outlier requests.
    - Wait a minute or two, to gather data, then click on one of the outlier traces to see the specifics about what is going on.
      * Graphical user interface, application, table

        Description automatically generated
    - Notice the Span name (which represents the service or function that is being called) is either /cart/ or /cart/checkout/.
    - To help understand how this trace compares with similar ones prior to the issue, look at the Recv./cart Summary in the lower left for all the cart operations and look for similar traces.
    - Set the time period to 1 hour so that it includes traces that occurred before the issue.
    - Click on an example trace from before the issue occurred
    - Notice that in this similar trace ListRecommendations was only called once. However, after the most recent deploy, ListRecommendations is being called many times per request, causing significant additional latency.
    - You can conclude that the issue with these outliers is caused by multiple calls to ListRecommendations.
  + Deploy Change to Address Latency
    - In order to address the latency issue that the last release created, you need to roll out another version that fixes the broken code. You will next modify the Kubernetes manifests for the services that contained the broken code.
    - To deploy a fix return to Cloud Shell and open the Source Code Editor. You'll be modifying the following files:
      * kubernetes\_manifests/recommendationservice.yaml
      * kubernetes\_manifests/frontend.yaml
    - Modify the image tag rel013019 with rel013019fix so the image should look like this:
      * containers:
      * - name: server
      * image: gcr.io/accl-19-dev/frontend:rel013019fix
      * imagePullPolicy: Always
      * Graphical user interface, text, application

        Description automatically generated
    - Save the files.
    - Return to the Cloud Shell prompt and redeploy the images with the fixes in them:
      * skaffold run
    - Validate Fix
      * Now that you've rolled back the breaking change, verify that your application is back to a healthy state.
      * Return to Monitoring and click Metrics Explorer.
      * In the metric search field, enter roundtrip\_latency and select custom.googleapis.com/opencensus/grpc.io/client/roundtrip\_latency.
        + Graphical user interface, text, application, email

          Description automatically generated
      * Select Global as a resource type.
      * NOTE: Please make sure that the Aggregator field is set to 99th percentile.
      * Change the chart type to Line. You should see a chart that shows an immediate decrease in latency (and if you don't, wait a minute).
        + Graphical user interface, chart, line chart

          Description automatically generated
      * Now, see if the incidents are resolved. Return to the Monitoring Overview.
      * You should notice two things - you no longer have an incident and there are events letting you know that the incident has been resolved. Again, if you don't see an Incident resolved message, wait a couple of minutes.
      * Your monitoring was able to correctly identify a change that caused user experience (as measured by latency) to degrade, you were able to identify the root cause and you've rolled back the breaking change! In the next section, see how Monitoring can help you resolve an issue with availability.
  + Error Rate SLO Violation - Find the Problem
    - In this exercise you will use Cloud Monitoring Application Performance Management tools (APM) to troubleshoot an issue causing ERRORs in your application violating your error budget.
    - Click Alerting in Monitoring.
    - Look for a Error Rate SLI incident and click the incidents to learn more about what's going on. Incidents can take several minutes to be confirmed and listed as an incident. If an incident has not yet arrived, you can skip the Incident step below. Select Navigation > Error Reporting in the GCP console.
    - You can see that the pod is logging significantly more errors than it was previously.
    - Acknowledged the incident so that no further notification escalation takes place.
    - For an alert like this there are many places to start, but the easiest is Error Reporting.
    - Notice the Open Error Reporting incidents with a recent spike in occurrences. Click on the Error: Conversion is Zero to learn more about the error in question.
      * A picture containing chart

        Description automatically generated
    - Look at the Stack Trace sample on the right. Here you can see what specific calls were related to the error. f8bc991519b95fa2.png
    - Click on the lowest call showing here: /usr/src/app/server.js:131
    - This will load you into Debugger. On the top bar under Application dropdown select currencyservice.
      * Graphical user interface, application, Teams

        Description automatically generated
    - Next, select the source code that is running from Cloud Source Repositories.
      * Graphical user interface, application

        Description automatically generated
    - Select your source with:
    - Repository: apm-qwiklabs-demo
    - Tagged version or branch: APM-Troubleshooting-Demo-2
    - Then click Select Source.
    - In the left hand menu browse to /src/currencyservice/server.js.
      * Graphical user interface, text, application, Word

        Description automatically generated
    - Scroll down to around line 155 which is the function where the exception was thrown. You can see the logline Conversion is Zero that was referenced in error reporting.
      * Graphical user interface, text, application

        Description automatically generated
    - From the above code snippet you see this error is logged when result.units < 0. To troubleshoot this issue you'll use Snapshots to inspect the variables as the application progresses.
    - Make sure you have selected Snapshot in the top right:
      * Graphical user interface, text, application, email

        Description automatically generated
    - Then click on the line number (155) you want to snapshot, then select Create snapshot:
      * Graphical user interface, text, application, email

        Description automatically generated
    - For this exercise take snapshots at line 155, 141 and 149. Add additional snapshot points wherever you feel appropriate. The system will take a variable snapshot the next time that code is executed. While the application is waiting for the code to be executed next you can see a "Waiting for snapshot to hit...." notice.
    - When the snapshot is complete the right hand pane will display the variables for that given snapshot.
      * Graphical user interface, text, application

        Description automatically generated
    - Notice the Variable and Call Stack information. This information can provide extremely deep understanding of the path you code is taking including the variables and structures that exist as it takes that path, all without restarting the application or changing any code.
    - Click result to inspect all 3 snapshots finishing on line 155. Remember the error is triggered when result.units is NOT > 0. Inspecting the variables you can see that result.units = NaN (meaning ‘not a number'). This is the issue that is causing the error.
      * Graphical user interface, application

        Description automatically generated
    - At this point you can conclude that the error is caused by a bug in the convert (or child) functions which sets the result.units to 0 resulting in a 0.00 price tag for the item being converted. Your troubleshooting along with the snapshot information and logs is a solid diagnosis of the issue.
    - So what is the bug that has caused this problem? From the code, result.units is set by the line 144 from euros, which was set in line 136 by operating on from.units
      * Graphical user interface

        Description automatically generated with low confidence
    - Inspecting the snapshots euros.units is also NaN, however, from.units is a valid number. Thus the issue happened when converting from.units to euros.
      * Table

        Description automatically generated with medium confidence
    - You can conclude that the root cause is a bug in how from.units is converted into euros.units on line 137 and 8 is passed into Data[] which is actually a key value mapping of currency units (like EUR) into exchange rates. The corrected line 137 would use from.to\_currency (aka USD) instead of from.units (aka 8).
      * Table

        Description automatically generated
    - At this point you have determined the cause of the bug and can make the appropriate change. Based on the timing of the alert, this could have been caused by the latest deployment.
    - See if the previous branch Master had this code error on line 137.
    - Go back to the Console and inspect the code using Cloud Source Repositories (in the console menu under CI/CD).
    - Open the apm-qwiklabs-demo repository and select the master branch.
    - Browse on the left hand side to src > currencyservice > server.js. Notice on line 137 the proper dividend: data[from.currency\_code] is used.
      * A picture containing text

        Description automatically generated
    - You have confirmation this bug was introduced in the latest push. To mitigate this problem you need to roll back to the previous version.
  + Deploy Change to Address Error Rate
    - In order to fix this issue, you'll need to deploy a fix to your application. To do that, you'll need to modify the Kubernetes manifest for the service that contained the broken code.
    - Deploy Fix
      * Return to Cloud Shell and in the Source Code Editor open the currencyservice.yaml file in the kubernetes\_manifests folder.
        + Graphical user interface, text, application

          Description automatically generated
      * Replace the image tag rel013019 with rel013019fix so the image should look like this:
        + containers:
        + - name: server
        + image: gcr.io/accl-19-dev/frontend:rel013019fix
        + imagePullPolicy: Always
      * Close the file to save it and return to the Cloud Shell prompt.
      * Redeploy the image with the fix in it:
        + skaffold run
    - Validate Fix
      * Now that you've rolled back the breaking change, verify that the application is back to a healthy state.
      * As before, start by verifying that your incident is resolved. Go to the Alerting in the Monitoring UI and verify that the error rate incident is resolved.
      * Next, return to Error Reporting. Open the error previously observed and verify that it is no longer occurring (the timeline should show no further occurrences since the last deployment):
        + Chart

          Description automatically generated
      * Congratulations - your monitoring was able to correctly identify a change that caused user experience (as measured by application errors) to degrade, you were able to identify the root cause and you've rolled back the breaking change!
      * Move on to the next section to learn about how you can optimize resource utilization using Cloud Monitoring.
  + Application optimization with Cloud Monitoring APM
    - In this exercise use Cloud Monitoring Application Performance Management tools (APM) to identify opportunities for improvement that will help your application run faster and use less compute resources.
    - In this scenario, the Director of Cloud Operations is disappointed with the recent rise in compute costs. Specifically, the currencyservice service is using more CPU than expected based on the usage of the system.
    - Your team has been tasked with finding optimization opportunities. APM tools will be used to analyze the service and ensure your team's efforts are focused on the right areas of the application.
    - From the Console, open Profiler from the left hand menu.
    - Change the Timespan in the upper right to 30 minutes. If there is no data, wait a minute of 2 for the data to populate.
    - NOTE: Profiler takes a random sample of calls to build an aggregate call stack. If you don't see the data you expect, it's because not enough time has elapsed during this lab and you completed it faster than expected. Feel free to use the screenshots below during the exercise.
    - In the filter options select the frontend service, the CPU time Profile type.
      * Graphical user interface

        Description automatically generated
    - Profiler takes random sample profiles of the system and combines the data to show you what functions are using the most resources. The flame graph below shows the function calls grouped by their use of the resource (in this case CPU) where the X-axis is the amount of CPU and the Y-axis shows parent child relationships.
      * A screenshot of a computer

        Description automatically generated
    - In this case the majority of the CPU is used by the ServeHTTP call on the left hand side. Click on this call to drill into the cause.
      * Timeline

        Description automatically generated
    - The expanded view shows almost half of this is caused by viewCartHandler, which in turn is mostly caused by getRecommendations.
    - The opportunity here is in the getRecommendations and in turn getProduct. Thinking back to your earlier exercise, remember that the recommendation service and getProduct were being called often in a loop due to an error in retry logic. The resolution for that issue will likely decrease compute cost by as much as 20%.
* Resource links for Module 6
* The path to certification 4 minutes - https://youtu.be/ijMU-lTkb8I