GCP Cloud Architecture

Noah Gift, Andrew Nguyen, Alfredo Deza and Michael Vierling

GCP Cloud Architecture

Noah Gift, Andrew Nguyen, Alfredo Deza and Michael Vierling

This book is for sale at http://leanpub.com/gcpcloudarchitecture

This version was published on 2020-08-21

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2020 Noah Gift, Andrew Nguyen, Alfredo Deza and Michael Vierling

Contents

Chapter 1: (Google) Cloud Overview

Andrew Nguyen

This chapter covers high level concepts of cloud computing and then dives into setting up Google Cloud Platform. If you're already familiar with cloud computing and how to access GCP via the command line (gcloud) or via the web UI, you can skip right to the chapters of interest.

If this is your first time with GCP but you have experience with AWS or Azure, you can skip to the second half of this chapter and dive right into the technical details.

Cloud Concepts

At some level, the cloud is nothing new. Before there were PC's, people had to perform computing on mainframes via client terminals. The PC revolution enabled local computing to be done independent of large mainframes. As the amount of data continues to grow, it is becoming harder and harder to handle all of the data management, engineering, and analysis on local workstations. Naturally, we are migrating back to a model similar to that of the mainframe.

However, there are clearly major differences – namely, it is now possible to connect to these centralized systems from anywhere on earth. Whether you are working in your company office, your living room couch, or a coffee shop, the cloud had completely changed computing.

There are many benefits to cloud computing but, as with any technology solution, there are always trade-offs. Chapter 2 spends a little time providing you with a high level framework for evaluating your business and technical requirements. Once you have determined your requirements, one of your first considerations should be whether or not to even use GCP or cloud computing.

Most of the costs and benefits of cloud computing can be grouped as follows:

- Cost Saving
- Security / Compliance
- Data Management
- Scalability

Cost saving is one of the most commonly referenced benefits when considering a cloud solution. In many cases, this is very true. For example, as an early stage startup, you know you need some servers to handle your backend machine learning application. However, you don't know how much capacity you have yet and you do not have a dedicated operations team. This is a perfect example of leveraging the cloud. You can scale as necessary and pay-as-you-go without needing to invest heavily upfront in hardware that you need to maintain.

That said, at some point, the balance may shift and a cloud-based solution might end up being more expensive. As a startup, this is a great problem to have! Determining this inflection point is complicated and will ultimately require input from teams throughout your organization (e.g., IT, data engineering, data science, devops, human resources, etc.) and is beyond the scope of this book. The simple point is keep this in mind – an on-premise solution or hybrid cloud solution may end up being something to consider.

Another major benefit of a cloud-based solution is the centralization of security and compliance considerations. Imagine now that you are a digital health company and you are in charge of the data science team. Your CEO hates the cloud so everyone on your team maintains a copy of the data on their laptops. It is Monday morning and one of your data scientists come into your office with a somber look and shares that his company laptop was stolen out of his car sometime over the weekend. Now, you need to do a complete inventory of what data may have been on the laptop at the time it was stolen (which is actually impossible to determine because there are no access logs when employees copy data from the file server. As a result, you decide to flag all of your data as compromised and now must handle breach notifications per HIPAA (Health Information Portability and Accountability Act) requirements.

Now, let us imagine if you had built the majority of your infrastructure within GCP. Your data are stored in a database hosted within Google Compute Engine (GCE) and you use some combination of AutoML, BigQuery, and Dataflow for all of your data science. Consequently, none of the data are actually stored locally. When your data scientist informs you that his laptop was stolen, all you need to do is remove access for any keys stored on the laptop.

Of course, like anything else, the trade-off for a fully cloud-based approach is that your data science team must always be connected in order to do their work. This means they would be unable to do any work while on a flight (without wifi) or any other situation where their internet connection is unavailable or unstable.

Along the lines of security and compliance, your data management strategy could also be dramatically simplified when moving everything to the cloud. You would not need to worry about your team members having stale copies of the data. If you need to update the data or correct it, you can simply update the version

stored in the cloud and everyone would have access to the latest version.

Lastly, one of the most beneficial aspects of a cloud-based approach is the scalability. It is difficult to plan for various load scenarios, especially if your project is new. You may not yet fully understand the demand for various aspects of your system. By building it within Google Cloud Platform, you can scale up and down as you need. Additionally, with API access to GCP itself, you can often trigger scaling automatically based on predetermined thresholds. This allows you to focus on the product itself, knowing that you can scale if and when you need to.

GCP Concepts

At a very high level, GCP is like any other cloud provider (with the big three being Amazon Web Services[1], Microsoft Azure[2], and Google Cloud Platform[3]).

Each provides various products and services that can be generally grouped into:

- Compute (virtual machines, containers)
- Storage (object stores, databases)
- Serverless (moving away from virtual machines)
- Machine learning (including automation of machine learning)
- Big data (visualization, analytics, map reduce, etc.)
- Networking (virtual private cloud, load balancing, etc.)

This book provides you with a hands-on introduction to each of the above *except* for networking.

Chapter 2: Getting Started

Create account

First, you need a valid Google account that can access Google Cloud Platform. Standard Gmail accounts work; accounts from your employer or school may or may not work, depending on the permissions from your IT department.

[1] https://aws.amazon.com/
[2] https://azure.microsoft.com/
[3] https://cloud.google.com/

You will find out right away when you attempt to access the Google Cloud Platform Console[4].

If this is your first time accessing GCP, you will be pleasantly greeted with an offer to activate a free trial that comes with $300 in credits to explore GCP products!

Once you have created your account and activated your billing account, you will need to create a project. You can name this project anything you want; you can also change the *project ID*. The *project ID* is important because it is the primary way the command line utilities and API's access your project.

Take a moment to explore the various products within GCP. This book will focus on the following groups of products given the focus on data engineering and data science:

- Compute
- Storage
- Big Data
- Artificial Intelligence

API's

GCP provides several different ways to interact with the various products – you can use the web interface, command line utilities, or direct API access. This lets you control how you want to interact with the various products in a way that fits best with your workflow.

The web interface provides a great starting point when you are exploring a new product, or if you need to quickly spin up a simple, one off deployment of some sort. However, it is best practice to automate as much as your deployment as possible. This ensures that you can reproduce your environment without a bunch of manual steps that you may forget.

In order to use the API's, you must first enable them for your project. Instructions can be found within GCP's documentation[5].

Googles API's are accessible from a variety of languages including Java, Javascript/Node.js, Python, among others[6].

while API's provide a powerful mechanism for interacting with GCP and managing a project, this book focuses on the use of command line utilities and the web UI.

[4] https://console.cloud.google.com
[5] https://cloud.google.com/apis/docs/getting-started
[6] https://developers.google.com/api-client-library/

Google Cloud SDK and gcloud

Overall, Google refers to the suite of mechanisms for interacting with GCP the *Cloud SDK*. Within this, there is the *gcloud* command-line tool, the client libraries that provide access to the API's in a variety of languages, and also product-specific command line tools (e.g., *gsutil* for interacting with Cloud Storage).

This section walks you through the basics of installing and configuring your command line utilities.

Installation

Mac / Linux

To install the latest version in Mac or Linux:

1. Download and install

Run:

Download and install

```
curl https://sdk.cloud.google.com | bash
```

You will be prompted for an installation directory (defaulting to your home directory), and an option to opt-in for some usage metrics. Once the installation starts, you'll also be prompted to update your shell's rc file.

2. Restart shell (to load changes)

Run:

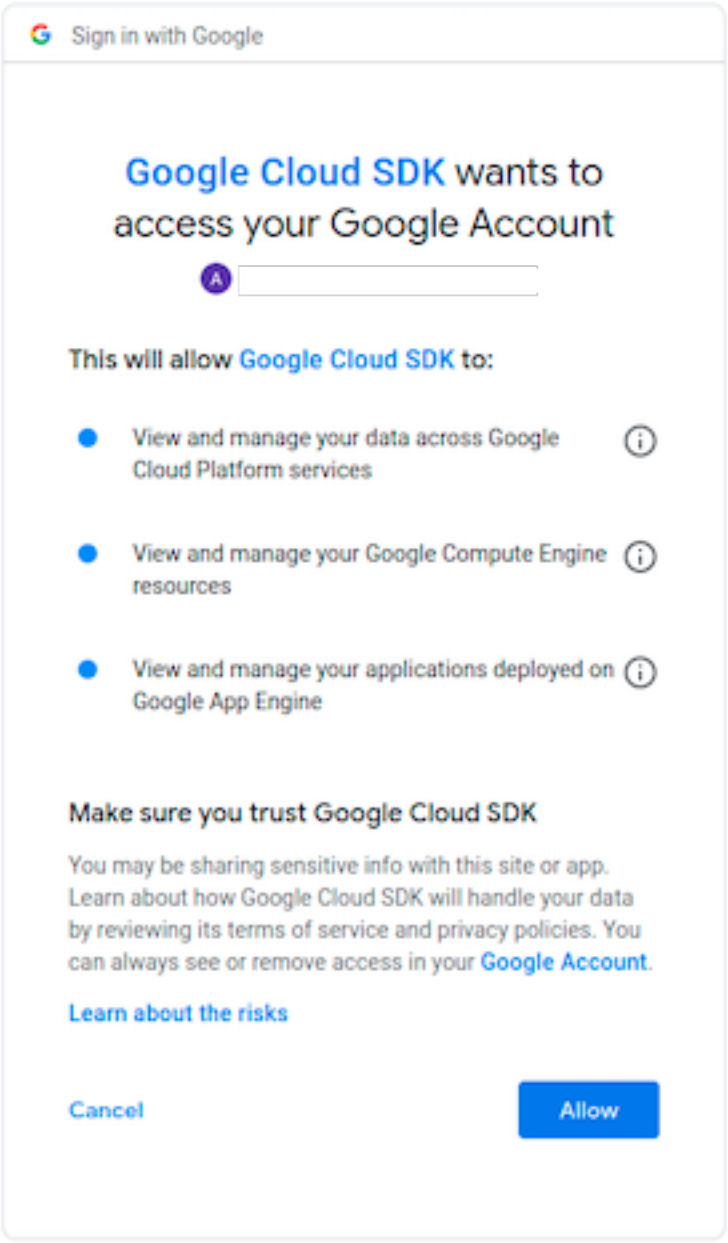
```
exec -l $SHELL
```

3. Initialize the gcloud environment

Run:

```
gcloud init
```

You will see a prompt for logging in to your Google account. This should open a web browser automatically; if not, you can copy/paste the provided link into your web browser. Once you login to your Google account, you should see an OAuth authentication screen similar to the following:



OAuth screen

Once you accept, you will be taken to a GCP documentation page.

4. Select project and zone

Your browser will be forwarded to the documentation page but there are still a few things for us to finish within the terminal window.

First, you will be prompted to select a project. This will link your command line instance to a project within GCP. If you have used GCP before, you will see a list of all projects associated with your account. If this is the first time you have used GCP, you should have created an account and project earlier in this chapter.

Once you have selected your project, you will need to select a default zone. Most people select a zone that is geographically nearby and you can always change this later.

Congratulations, you are all setup!

Windows

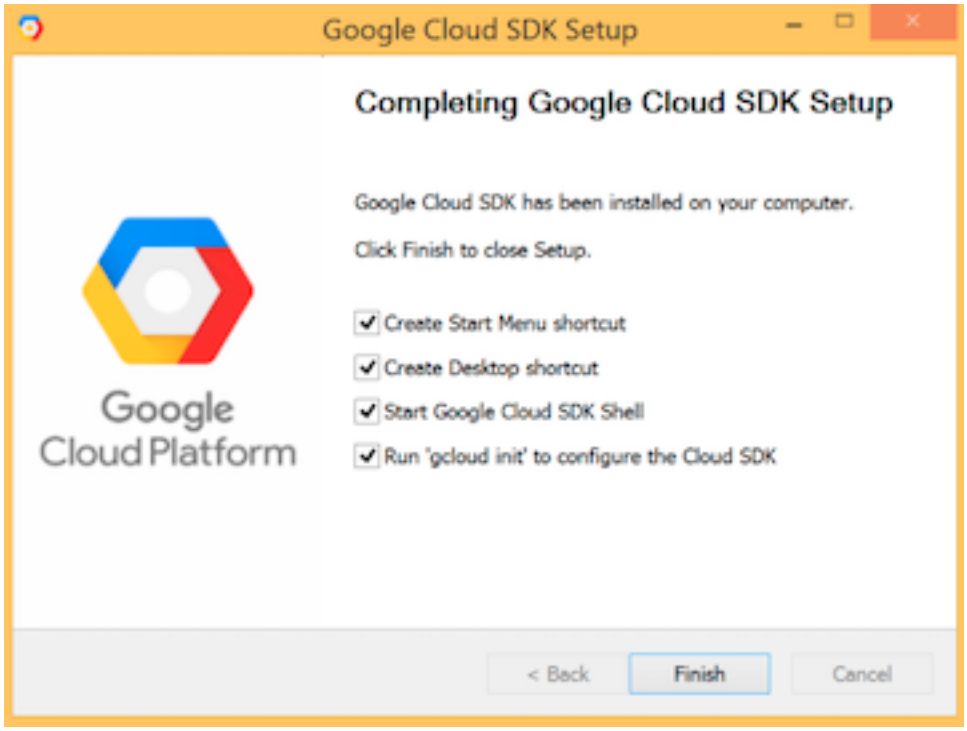
1. Download and install

First, download the installer from https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe

This will install a bundled Python if you do not already have it installed. Supported versions are 3.5 to 3.7, and 2.7.9 or higher (for Python 2.x).

2. Initialize the gcloud environment

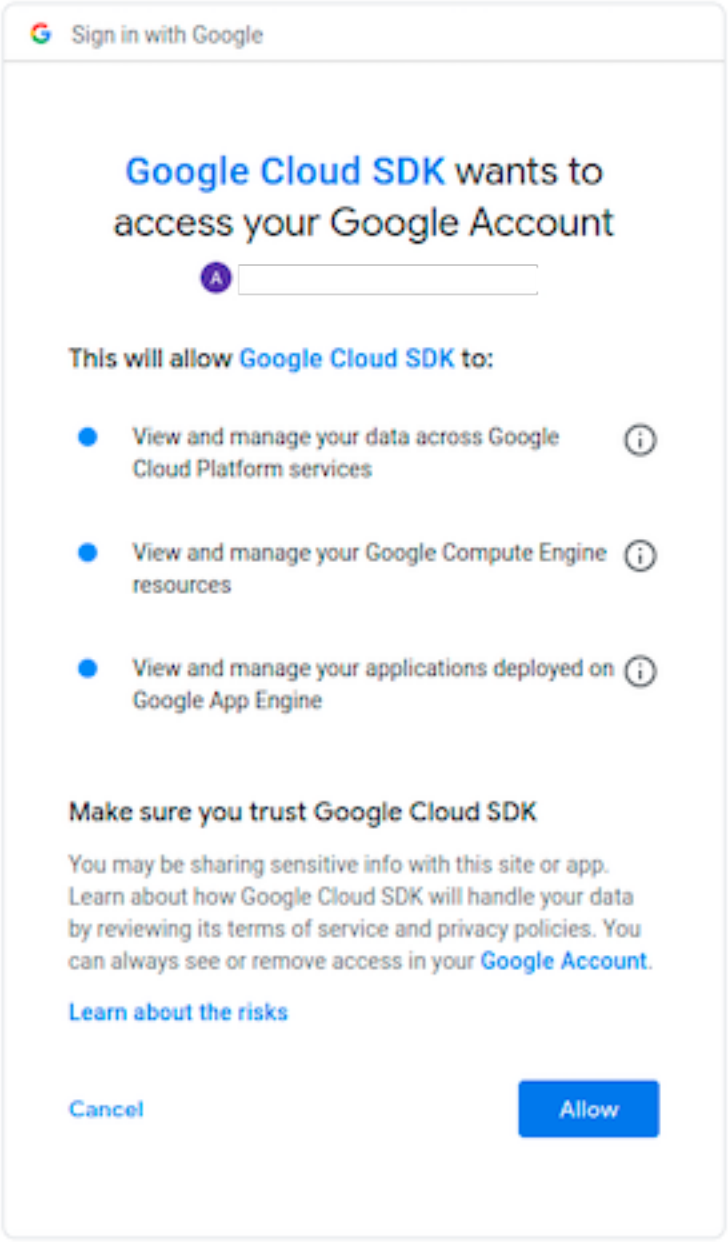
Once the installer has finished, you should see the following screen:



Windows post-install screen

Be sure to check "Start Google Cloud SDK Shell" and "Run 'gcloud init' to configure the Cloud SDK. The other options are personal preference.

You will see a prompt for logging in to your Google account. This should open a web browser automatically; if not, you can copy/paste the provided link into your web browser. Once you login to your Google account, you should see an OAuth authentication screen similar to the following:



OAuth screen

Once you accept, you will be taken to a GCP documentation page.

3. Select project and zone

Your browser will be forwarded to the documentation page but there are still a few things for us to finish within the terminal window.

First, you will be prompted to select a project. This will link your command line instance to a project within GCP. If you have used GCP before, you will see a list of all projects associated with your account. If this is

the first time you have used GCP, you should have created an account and project earlier in this chapter.

Once you have selected your project, you will need to select a default zone. Most people select a zone that is geographically nearby and you can always change this later.

Congratulations, you are all setup!

Basic CLI Usage

The gcloud CLI follows a similar approach as other CLI tools with commands and subcommands. However, gcloud refers to these as *groups* and *commands* with the following format:

```
gcloud <group> <command>
```

For the most part, there is a group for each GCP product (e.g., `app` for App Engine, `compute` for Compute Engine, `sql` for Cloud SQL, etc.). To get a list of all of them, run:

```
gcloud
```

This will print a list of command groups and a short description of each.

```
Command name argument expected.

Available command groups for gcloud:

  AI and Machine Learning
      ai-platform             Manage AI Platform jobs and models.
      ml                      Use Google Cloud machine learning capabilities.
      ml-engine               Manage AI Platform jobs and models.

  API Platform and Ecosystems
      endpoints               Create, enable and manage API services.
      recommender             Manage Cloud recommendations and recommendation
                              rules.
      service-management      Create, enable and manage API services.
      services                List, enable and disable APIs and services.

  Anthos CLI
      anthos                  Anthos command Group.
```

Screenshot of gcloud command groups

The section below covers the management of multiple configurations so that you can work on multiple different projects or with different accounts. However, there are a set of global flags that apply to every command in every group such as `--account` and `--project` that allow you to explicitly specify the appropriate account/project ID regardless of your active configuration.

Managing multiple configurations

The gcloud CLI tools are a great way to interact with GCP and provide a simple interface for including in scripts and other build tools. As you continue to work with GCP, you will inevitably start working on multiple projects.

gcloud provides a simple mechanism for working on several different projects at the same time via the `gcloud config` command.

To see all active configurations:

```
gclod config configurations list
```

You should see at least one configuration (likely named "default") along with the associated account email address, project ID, and zone/region.

To create a new configuration:

```
gcloud config configurations create <name>
```

This will only create a new configuration for you. To link it with a particular account and project, you will need to login:

```
gcloud auth login
```

However, unlike the initial authentication, the above command will not prompt you for a project ID. You can verify this by running `gcloud config configurations list` again – you will see that your new configuration is associated with an account but does not have a project ID, zone, or region set.

To set a project ID, run:

```
gcloud config set project <project ID>
```

If you list the configurations again, you will see that your active configuration is now set to the specified project ID.

This is a powerful feature that allows you to maintain multiple configurations for different projects and Google accounts (e.g., you might have a personal configuration associated with your Gmail account and another configuration associated with a work or school account/project).

Chapter 2: Translating Business Requirements to Technical Requirements

Andrew Nguyen

It is easy to get bogged down in all of the technical details of a project. This is especially true if you have a technical background and enjoy coding, working with databases, or creating machine learning models. However, one of the most important aspects of being an architect is the ability to balance the technical details with what we are really trying to accomplish – the business requirements.

This chapter focuses on how to translate what our stakeholders want or need (e.g., "I want a cloud-based machine learning system that can automatically identify seizures from EEG signals") into what our software engineers, data engineers, and data scientists need to know to actually build it.

When your stakeholders have technical backgrounds or have experience working with technical teams, this process may be relatively trivial; you can simply focus on how to map their request to various aspects of the Google Cloud Platform. In fields such as healthcare, this process can be much more cumbersome and challenging. Often, our stakeholders know what they want and feel that they are communicating clearly. However, they are using different vocabulary so it is up to us to understand their request and translate it.

This chapter provides a set of considerations when working with clients and stakeholders, then translates these business requirements into specific technical requirements. Details will be discussed in the context of the Google Cloud Platform but the general approach and overarching concepts can be applied to any project built on any platform.

Determining Business Requirements

This is one of the most important parts when architecting a solution since it sets the stage for the entire project. If we do not fully understand the business requirements, we run the risk of putting together a solution that ends up being overbudget, suffers from feature creep, or fails to meet the needs of our client or stakeholders.

The technical details are usually where we want to spend our time as architects. We want to focus on how many Compute Engine instances we need, how best to construct the Dataflow pipeline, or whether we should use the Kubernetes Engine. As exciting as these details are, we should spend the bulk of our time up front, simply talking to the client and understanding what it is they want and, more importantly, need.

This section steps through key aspects of understanding your client's business requirement. The section after takes the business requirements and starts the process of translating them into specific technical requirements.

What's The Goal?

Before you start focusing on the gritty details of the project (which we'll discuss in plenty of detail throughout this section), make you you understand what the overarching goal is – what is it that we (the collective we, anyone and everyone associated with the project) are truly trying to accomplish? Some say it doesn't matter, that our job is simply to architect and build the tools that the client is asking for.

It's hard to argue with this perspective. We are being brought onboard the project as a cloud solutions architect. Our job is to design and implement the cloud-based computing platform "to spec." So, let's do as the client asks and build what they ask us to build. There is a good chance you will get away with this strategy. After all, if (when) the solution fails to meet the client's expectation, we can point to the requirements or specification document and prove that we built exactly what was asked.

This is how most IT departments operate and likely one of the reasons IT departments also have the reputations they do. If we work to understand what the client is really trying to build, we become part of the team. We are then in a position where we can start to identify conflicting requirements, or when the client is asking for A but we know they really want B. This translates into success because we end up building what the client *wants* not just what they ask for.

So, how do we make sure we understand the goal?

The first step is to make sure we baseline the conversation with the client; we want them to know what we are asking about the big picture. It is most helpful if you have some understanding of the domain. This way, you can ask questions using a language they understand well. If you are unfamiliar with the domain, try to do some background reading. This can be difficult in complex fields such as medicine or healthcare but you do not need to be an expert. You just need to know enough to use the correct vocabulary and to understand what the client is asking for.

Once you've baselined the conversation, the next step is to be *naturally curious*. Ask questions to make you understand the basics: what, why, when, where, and how.

What?

This is the most obvious. What is it that the client is trying to achieve? What is *their* product to *their customers*? Say that your client comes to you asking for a "cloud-based machine learning pipeline that will classify images." Because they are familiar with the Google Cloud Platform (after having attended a 1-hour webinar), they go on to tell you that they want to "build this with AutoML Vision and expose the model via the AutoML API."

This may sound like most of your job is done and you can start to spec out the details of how all the pieces will be connected together. After all, they clearly know what they are talking about and we can build exactly what they asked for!

However, if we talk to the client a bit, we learn that they want this AutoML Vision model to be called from a mobile app running on both iOS and Android. After asking another follow up question, we also learn that there are no plans to calling this API from any other source (e.g., web-based application). Given this new information, we are wondering if something like AutoML Vision Edge would be more appropriate; after all, it is designed to run AutoML Vision models directly in iOS and Android.

There is no way that we can walk you through every possible scenario. The key thing to remember is think of the conversation with the client as peeling an onion one layer at a time – you want to keep probing and asking questions until you get to the core, a true understanding of what your client is trying to accomplish.

Why

Similar to digging deeper and working to truly understand what the client is asking for, we also want to know the whys behind the project. Why is the client looking to build this in the cloud? Why is the client considering streaming vs. batch processing? Why is the client even embarking on this project?

Sometimes, it does not matter why – we will still architect and build the solution exactly the same regardless. However, oftentimes, understanding the why behind the client's project will help us select the best tools for the job. In the example above, the client may tell us they want to use the AutoML Vision API. Perhaps we have not considered asking if there would be other uses for the API but we asked the client why they thought the AutoML Vision API would be

best. During this conversation, they might respond with, "So that we can call it from a mobile app!" Again, this immediately clues us into the mobile-only perspective which points us to Firebase and the AutoML Vision Edge components of GCP.

When

Generally, this question is a reminder to start determining what the requirements are with respect to all things *time*. We can start with very high level aspects such as when will the project start? How long do we have to complete each phase of the project? In other words, we need to remember the project management basics.

As we start to get into the specific details of the project, we want to get a sense of the timing of all the components, mainly, from the perspective of the client's customer. We can easily ask the client what they *think* they want. However, it ultimately comes down to the end user of the system and if we can provide the response time that they expect.

Within GCP, there are many products that have similar or overlapping functionality. For example, there is Dataproc and Dataflow, both data processing tools. There is also Firestore, Datastore, or Cloud SQL (just to name a few), all some sort of database. One of the things that separates one GCP product from the next is the underlying timing that is provided or guaranteed.

For example, the GCP documentation highlights that one of the key differences between Firestore and Datastore is the addition of "real-time updates" to Firestore. Another example can be found when considering deploying containers via Kubernetes Engine (GKE) or Compute Engine (GCE). The former will be much faster but requires that several compute engine instances running all the time. Deploying a container directly to GCE could be cheaper but will also take much more time as we wait for a new instance to spin up.

All this discussion about timing means that we need to spend a little time talking about "real-time" and what that even means. If you ask an embedded firmware engineer what real-time is, they may respond "microseconds." For example, in a servo control system, each iteration of the control loop might need to complete in 125 μsec. In an electronic health record, an extra millisecond will likely go completely unnoticed. On the other extreme, a large biopharma company had used the term "real-time report" – when asked, the speaker indicated that the reports were generated once every 3 months. This was in contrast to how long it used to take them to generate a report, upwards of a year.

Sometimes, when a client says they want something in "real-time," they may also be referring to the idea of *stream* vs. *batch* processing. That is, they want

data to be processed as it comes in and not just once a day or once a month. It might be that the data is to be processed immediately but the exact amount of time allotted to the processing could be on the order of hours. To many, this is not a real-time system despite others referring to it as such.

This is really to drive home the point that you want to understand the specific details of what the project requires and to stay away from buzz phrases such as *real-time* or *near real-time*.

Where

This is critically important question especially as companies look to multiple cloud providers, have partial on-premise solutions, or have their data spread around various corners of the internet. Determining where the data are stored, where they are to be processed, where the results of the processing will be stored, and where it will ultimately be *used* are all important aspects when asking, "Where?"

The answer will impact decisions on which products to use but will also affect other operational aspects, such as networking configurations, access control, and container hosting, among others.

At a minimum, you will want to understand where "things" are coming from, where they are going to be processed, and where they go next. *Things* can refer to almost anything:

- Data (e.g., where are the source data located? What kind of databases? Are there any access control issues?)
- Users (e.g., if we are creating a web app for end users to input data, where are they located? Is this for an intranet? Will there be VPN? Can anyone access it from anywhere? Will they be located in a country with locked down firewalls?)
- Other software systems (e.g., will we need to interface with a particular API? How is access control managed? What data formats are supported?)

How

Here, we are referring to how the client wants the problem to be solved. In many cases, they will not have an answer; this is why you were brought on board! However, in other cases, the client may have some idea how they want the solution to look. Once given an answer, your immediate follow up should be "why?" The client may not realize that they haven't communicated a key requirement to you because they went straight to specifying that they want to use Kubernetes Engine instead of Compute Engine. If you ask why, they might explain that they have a significant number of Kubernetes configurations already available from another project.

Data Sources and Sinks

Presumably, the systems you will be architecting will have a significant data component. As you step through the questions above to get a better understanding of the project, it is important to consider the data that is coming into the system (sources) and the data that leaves the system (sinks).

Considerations for both sources and sinks are very similar. Mainly, we want to ask the questions "what do we need to do to read the data from the source?" and "what do we need to do to write data to the sink?" If you can walk through the process and trace the flow of data, you will be in good shape. For example, say we want to connect to a source PostgreSQL database and bring the data into BigQuery.

First, you need to decide how you are going to access the data from GCP. You could spin up an instance within Compute Engine to run the job. Or, perhaps you will want to run a Dataflow job instead. One of the first considerations is if there are any access control considerations on your data source. For example, if your data source is using IP filtering, a GCE instance may be easier since you can assign it an IP address and add it to the data source's whitelist. On the other hand, a Dataflow job is easier to scale if your source can handle a large number of concurrent reads; however, there is not straightforward way to whitelist all of the Dataflow worker IP addresses.

Second, you need to decide how you need to format your data in the context of BigQuery. What kind of database is it (e.g., row or column-oriented? document-oriented?) and what sort of transformations are necessary. We will cover the details of transforming the data when discussing *data engineering*. Again, the same considerations apply regarding security, networking, etc. As you work through Chapter 9 (Storage), pay particular attention to the limitations and nuances behind each of the types of databases and storage available to you.

Data Engineering

We can break things down into two subcategories, *syntactic* and *semantic* considerations. The first looks at things like the format of the data (e.g., JSON vs. XML) or whether you are moving data between a relational database and a column-oriented one. Generally, syntactic considerations are usually looking at the structure of the data, not the underlying meaning. When considering the underlying meaning of the data, we call this *semantics* (vs. syntax). Examples of semantic considerations include merging first and last name fields to create a full name field, or removing the date of birth and replacing with age. These all require some *understanding* of the data.

Generally speaking, syntactic considerations require solid technical skills and knowledge whereas semantic considerations also require domain-specific knowledge.

Extract, Transform, Load (ETL)

If you have been working with databases and ever moved data into or out of a database, you have essentially performed and ETL job. That is, you *extracted* the data from a source, *transformed* (or translated) the data some how, and *loaded* it into a target database. In the context of ETL, the discussion in a previous section about data sources and sinks correlate to the extract and load phases, respectively.

Oftentimes, the transformation process is simply the absolute minimum manipulation necessary to move between databases such as when moving from one relational database to another. When moving between fundamentally different types of databases, however, the ETL process can be a bit more involved. The specifics are beyond the scope of this book but requires a combination of expertise in the databases themselves as well as some understanding of the underlying data. Consequently, teams working on large ETL projects typically consist of database administrators, domain experts, and data engineers.

Machine Learning / Artificial Intelligence

These days, when someone speaks of data engineering, it is almost always in the context of machine learning or data science. The main goal here is to reduce the burden on the data science team so that they can focus on analyzing the data and extracting useful and actionable information. There is an oft quoted mantra in data science – 80% of the work is in cleaning and processing the data, 20% in the actual analysis. While the origins are suspect (and actual numbers are as low as 60% for cleaning and processing data), it captures an important point. Data scientists spend a significant amount of time gathering, processing, and cleaning data, oftentimes referred to as *data wrangling*.

Sometimes, data wrangling is a one-off exercise, something that is performed once during some exploratory data analysis and never needs to be performed again. However, most of the time, the wrangling is something that will need to be performed repeatedly over the lifetime of a project. One of the most common reasons is integrating new or updated data. For example, in a large-scale natural language processing project involving data from patients with Parkinson's Disease, new data is extracted from the source every 3-6 months. After each data extraction, the data need to be processed and cleaned, and then validated, prior to integrating these new data into the overall analysis.

This is a great example of the need to build a reliable data engineering pipeline to handle the cleaning and processing. One great tool within GCP for building these pipelines is Dataflow[7] which uses the Apache Beam[8] library for implementing both stream and batch data processing jobs. With small datasets, these jobs can be run locally on a single workstation; yet, they can also be scaled up to run on hundreds or thousands of worker nodes within Compute Engine. The beauty is that all of the scaling and job coordination is handled by Beam and Dataflow. On top of this, the Beam programming model has *runners* that allow Beam jobs to be executed on existing infrastructure such as Hadoop and Spark.

Integrations

It is challenging enough to manage large software projects; it is significantly more difficult when managing projects that have multiple integrations with external software and systems. As early as possible in a project, you will want to identify *all* integrations. Your client may say, "Oh, we don't need to worry about that. The integration should be very straightforward. They have a clear and well-documented API." While this is great news, should also press your client to facilitate the necessary introductions immediately. Integrations are often the most challenging because they are the hardest to test. We can write good unit tests for our code. How do we write an effective test that includes a third-party's software system? Sure, we can create stubs and simulate some of the external traffic. This assumes that the specifications for the other system are well-documented AND that the other system is actually built to spec.

As with data sources and sinks, there are many nuances to consider and it simply is not feasible for us to list out every possibility. What is most important is that you dig deeply to fully understand what the requirements are when integrating a third party system. Additionally, test early and test often. You will want to identify if there is an integration problem as early as possible in the project. The fix may require equal effort on both your end as well as the third party's. Since you do not control their priorities and you will likely need to involve your client in the conversation (after all, they are the third party's actual customer), it will be a slow and painful process. You do not want to leave this for a couple weeks before the project deadline because "it should have just worked."

[7] https://cloud.google.com/dataflow
[8] https://beam.apache.org/

Cost

Estimating the cost of a project is extremely difficult. There is a delicate balance between "underpromising and overdelivering" but that is much easier said than done. Sure, we can overestimate the cost of a project to ensure that we come in under-budget. However, providing too high of an initial estimate is likely to send your client running to another architect. Underestimating the cost might help you win the project but your client will be very disappointed when you come in with a final budget twice the original estimate.

There is an old adage: *good, cheap, and fast; pick two*. That is, if your client is looking for good and fast, the project will likely be very expensive. If they are trying to cut cost, then they will need to sacrifice the number of features or the overall timeline. As an architect, part of your job is to juggle these three things while simultaneously managing the client's expectations.

Once you have sense of what the client is trying to accomplish (i.e., what's the goal?) and have identified the bulk of the requirements, sit down and put together a basic cost estimate. How many people would be needed to accomplish the project in the given timeline? What is the anticipated platform cost (GCP's calculator[9] is a great tool for this)? Of course, you will need to continue to refine this as you learn about "that other" integration and "oh, one more" feature. However, the sooner you can (accurately) set the client's expectation of the overall cost, the better.

Timeline

Timelines and cost are intertwined and cannot be separated. How long a project takes is a function of what needs to be done and the amount of resources dedicated to the project, the latter of which is somewhat interchangeable with cost. Developing a timeline with the client is a bit of a see-saw process – you will provide a timeline estimate which will impact the overall cost and/or scope of the project; then you change the scope or the cost and re-estimate the timeline. This process will go round and round until the project scope, timeline, and overall contract have been finalized and agreed upon by all involved.

During this process, it is important to highlight (for yourself and for your client) the parts you control versus the parts outside of your control (e.g., a third party's integration task). This will help avoid miscommunication down the line. As with cost, estimating timelines also involves a delicate balance.

[9] https://cloud.google.com/products/calculator

Provide too long of an estimate upfront and you might appear incompetent; providing too short of an estimate will result in an awkward conversation when it is clear you are going to miss your original deadline.

Adaptability

Another key consideration for any project is whether you are building something as a one-off solution to solve an immediate problem as quickly as possible, or if you are building something that can support a variety of future use cases. This has direct implications in the overall timeline and cost of the project. The cheapest option in the short-term is to narrow the scope as much as possible and address immediate needs. After all, the project might get cancelled or might completely change directions 12 months from now. On the flip side, perhaps another week's worth of work would enable the system to be quickly adapted to a variety of use cases that your client is already considering.

Success

Given the need to balance cost, timelines, and (frequently changing) requirements, it is easy to get lost in the weeds and forget what constitutes "success" for the project. As we alluded to when earlier in this chapter, success is defined relative to the goals of the project. As a result, it is important to check in with your client and make sure that the project is on track and that expectations are being met. That said, it is also important to recognize and respond to issues of feature creep.

In any case, it is important to define specific metrics or *key performance indicators* that are measurable. It might be tempting to create a metric of "data processing engine successfully processes data in real-time." The challenge with this is the client can keep changing their definition of real-time and you may never succeed. This is not to say that your client is attempting to take advantage of you. Over the course of the project, the team may better understand the overall system and redefine "real-time" accordingly.

One way to improve this might be "data processing engine successfully processes data within 2 minutes." This is much better and clearly defines the overall timeline. However, it is unclear how much data is to be processed. To further refine this, we may say, "… successfully processes up to 5GB of data within 2 minutes." This sets much clearer boundaries and will make it very easy for us to create a set of automated tests that will ensure this requirement is met, even as the project evolves.

Managing Technical Requirements

The key to managing technical requirements is balancing the discussion above about the business requirements against what is actually possible given the technology stack – in our case, what is it that the Google Cloud Platform is actually capable of?

So, in addition to understanding the business needs of the project, you will need a deep understanding of your technology stack. The remainder of this book provides you with a broad survey of the Google Cloud Platform from the perspective of an architect.

Prioritizing the Requirements

When considering the prioritization of requirements, you will want to consider it from both the business and technical perspectives. From a business perspective, you want to use a framework similar to the MoSCoW method:

- Must Have

  This includes those features or requirements that are non-negotiable. They are absolutely critical to the success of the project and it would not be able to deliver the product (this may be a legal requirement, a core function of the project, or a safety issue).
- Should Have

  This includes things that are important to the overall success of the project but the project will still function without it. It is important to note those items that may diminish or decrease the functionality in some way, but the end solution is still viable (possibly with the use of a work around).
- Could Have

  These are desirable and may enhance the functionality of the solution. However, they are not as important as *Should Have* items. Incorporating these items should only be done if the project is ahead of schedule or under budget.
- Won't Have

  These are items that are being left out of the current phase of the project. It is important to communicate to the client that you are not removing these features or requirements completely, you are simply pushing them into a future project. Items that fall into this category are typically those that will push the project beyond the allocated timeline or budget. They may also include things that are nice to have but otherwise do not have any real impact on the overall functionality (e.g., cosmetic changes, convenience features).

Chapter 3: Architectural Decision Making

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Abstractions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Chapter 4: Compute Options for App Development on the Google Cloud Platform

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

GCE

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Google App Engine and Cloud Build Continuous Delivery

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Cloud Build Continuous Delivery

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

References

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Chapter 5: Storage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

The Four Fallacies

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

The CAP theorem

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Cloud Storage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Buckets

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Downloading objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Composite Objects

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Resumable uploads

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

SQL

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

NoSQL

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Bigtable

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Memorystore

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Cloud SQL

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Cloud Spanner

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Chapter 6: Containers and Orchestration

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Intro to Containers and Kubernetes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Installing and Testing Docker

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Deploying a Container within Docker

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Deploying a Container within Compute Engine

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Google Container Registry

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Google Kubernetes Engine

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Some Basic Terminology

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Setting Up a Cluster

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Deploying to Kubernetes

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Chapter 7: Serverless

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Types of Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

HTTP Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Background Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

A basic function

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Enable the cloud functions API

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Advanced setup

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Create a Github webhook

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.

Use the Secrets API

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/gcpcloudarchitecture.