***Building Software for the cloud:***

Storing Data in the cloud:

When storing data in the cloud, there are many factors to consider, like for example, how much data do you need to store, the type of data, the geographic region you are storing the data at, if the data is only reading or also writing, or how often the data changes in time.

For each case, you will be choosing one of the data storage technologies. You can choose traditional storing tec like **Block storage**, or newer tecs like **Object storage** or **Blob storage.**

When we create a VM running in the cloud, it has a local disk attached to it, this is what we would call **Block Storage**. This works the same way as if we had a physical disk, the VM’s OS creates a files system on top of the block storage, Our **Block Storage** can be **Persistent** (Used for instances that are long lived and need to keep data across reboots and upgrades) or **Ephemeral** (Used for instances that are only temporary and only need to keep local data while they’re running, meaning Temporary files).

If what we need is to share data across devices, the cloud providers can provide with a system to do it and have all your files in the same files management system with a NTF protocol for example.

If what we need is to store application’s data, we might want to look for an **Object storage**, also known as **Blob storage**. This kind of storage lets you place and retrieve objects in a storage bucket. Here there is no file management system, what you store inside this storage systems are objects, and objects have a different name from one another, so whenever you want to call an object, you can do it by using the object’s name. To interact with an object, inside of an object storage you need to use an API.

Also, we said before that the cloud providers also offer **databases as a service**. These databases can be either **SQL** (Relational) or **NoSQL**.

Factors that affect the storage class that we will use:

Throughput: The amount of data that you can read and write in a given amount of time.

Input/Output Per Second (IOPS): Measures how many reads or writes you can do in one second, no matter how much data you’re accessing.

Latency: The amount of time it takes to complete a read or write operation.

Hot Data: Accessed frequently and stored in hot storage. **SSP**

Cold Data: Accessed infrequently and stored in cold storage. **SHD**

Load Balancing:

There are several Load Balancing techniques like **Round Robin DNS** (Domain Name System): This is a broadly used method, this method iterates on a list of servers, and gives a task per server, if the number of tasks is bigger than the len of the server list, it iterates again till there are no more tasks.

The DNS is the protocol in charge of transforming a URL into an IP address, returning the numbers corresponding the IP address corresponding to the URL. In the simplest configuration the URLs always get translated into the same IP addresses, but if we configure our **DNS** to use **Round Robin,** it will give each client asking for the translation a group of IP addresses in a different order. The clients will then pick one of the IP addresses to try to reach the service and if an attempt fails, the client will jump to another address on the list. You can’t choose which IP is being choose, or even change that if the server gets overloaded

Other method it to use a dedicated load balancer which is a machine that functions as a proxy between the client and the server. It receives the requests and bases on the rules we applied, it directs them to the right Backend server.

**Content Delivery Networks (CDNs)**: Make up a network of physical hosts that are geographically located as close to the users as possible. What they do is that they cache an information that has been already asked for to the server, as close to the user as possible, this way, if someone else in the area tries to get this same information, only the first person in ask for the info has to wait, the for the rest of them, the latency is much less.

Change Management:

To make sure that our changes are not going to become bugs, we must apply Unit Tests and Integration Tests. Usually, what we do is that we create different environments, to test, develop, and then we a=have the one that we use to work, we call it production enviroment. What you want to do is to get all the changes we want to make to a Test enviroment, then get the ones that won’t crash our system and send them to the development enviroment and see how they interact with each other, and only when we are sure that those are the changes that we want to ultimately push, we push them to the production enviroment.

**A/B testing:** Some requests are served using one set of code and configuration (A) and other requests are served using a different set of code configuration(B).

***Managing and Alerting:***

Getting started with Monitoring:

The key to ensure our service runs correctly in the cloud, is by using the correct **Monitoring and Alerting rules.** We monitor them by doing requests and getting responses (404 for example), also we can monitor other factors, like how much memory or disk is being used.

Getting Alerts:

Since we can’t be always in front of the computer testing the state of these, we must create scripts that can make those tests for us often, raise alerts and solve the problem if possible. We can use **cron** in Linux for example, to run a script automatically. If we an error, the most common approach is to send an email to yourself with the details of the problem, so you have an idea of what is happening whit the service and fix it when possible.

We can classify the alerts we receive in two types, those that need our immediate attention, and those who need our attention soon, we call the first kind **pages**. We can receive this info through automated phone calls, SMS, or automated emails.

Service-Level Objectives:

These are Pre-established performance goals for a specific service, is more important for example a bank account transfer than a videogame, and maybe, the cache that we use to make the application run is no longer present, so the application is running, but is running slow.

When a service as a SLO that means that the service promises an estimated number in percent of the time that the service will be down, this is measured in nines as it follows: 99.0 (two nines), 99.999 (five nines), this represents the amount of time int a year that the service can be down, and still be considered as functional.

A **Server-Level Agreement (SLA)** is a commitment between a provider an a client.

Basic Monitoring in GCP:

The tool we are going to use in this course is called **Stackdriver**.

***Troubleshooting and Debugging:***

What do if we are not Physically there:

When you are troubleshooting a VM in the cloud there is a bunch of things that you can’t do, like walking there and check what the problem is, instead there are many other things that are easier in VMs than they are to ‘‘physical’’ computer, for example adding more memory, or moving the machine to a different data center.

In this case the better option for us is to have a version control system attached to a system, so you can go back to a state where the system was functional, and then take a snapshoot of the disk at the moment that the problem occurs and then put it in a different healthy machine and see how it behaves in order to discover its problem.

Another important feature to look into are the logs, we can ask to the provider for the logs and then get valuable information from them.

Identifying where the failure comes from:

First, we must know if the error is coming from our side of the network or from the provider itself. A very handy way to know this is by looking into the geographic region of the machines that are down, if the problems is the provider, this can be a clue to investigate. In this case the next step to do is to contact the provider. If that’s not the case, then the failure is within your system.

If you’re having problem with resource usage, you can try use the same system in a different machine that has more resources and see how it behaves. If the problem is inside of the code itself, we can do **Rollbacks**.

**Containers:** Packaged applications that are shipped together with their libraries and dependencies.

Recovering from failures:

When we want our service to run without any sudden error occurring, or, if we want to recover fast from them, we must have **backups,** in this case having backups doesn’t only mean to have a backup for the version control system, but also, a **network backup** or **hardware backup**.

If you operate a service that stores any kind of data, it’s critical that you implement automatic backups, and that you periodically check that those backups are working correctly by performing restores.