Task 2

2.1 Steps to create VM in google cloud platform

First step was we logged in to our Google cloud console and create a new project with any name we want. (As a matter of chance we were able to create two different projects).

To create an instance from a public image we have three different options.

1. Create using Google Cloud Console.
2. Create using gcloud command line tool.
3. Create using Compute Engine APIs (REST and client APIs)

But in this report, we will only focus on the first option. There are a few steps to create a VM using google cloud consol. (we listed the steps as an instruction that will be used by others).

1. In the GCP Console, go to the VM Instances page.

[GO TO THE VM INSTANCES PAGE](https://console.cloud.google.com/compute/instances)

1. Select your project and click **Continue**.
2. Click the **Create instance** button.
3. Specify a **Name** for your instance.
4. Optionally, change the **Zone** for this instance.
5. Select a **Machine type** for your instance.
6. In the **Boot disk** section, click **Change** to configure your boot disk.  
   Create a boot disk no larger than 2 TB to account for the limitations of MBR partitions.
7. In the **OS images** tab, choose an image.
8. Click **Select**.
9. To add secondary non-root disks to your VM instance:
   1. Click on the **Management, disks, networking, SSH keys**.
   2. Select the **Disks** tab.
   3. Under **Additional disks** click **Add item**.
   4. Specify a disk **Name**, **Mode**, and set the **When deleting instance** option.
   5. Add additional disks as needed.
10. Click the **Create** button to create and start the instance.

2.2 APIs for remote creation of VMs

There are several ways to create an instance remotely.

But, to summaries the available options we can classify them as follows.

1. Using **REST** API and.
   * In this API, we will construct a POST request to the instances URI with the same request body. We can add up to 15 secondary non-boot disks at the time we create a VM instance by using the **initializeParam** property for each additional disk. Create additional disks with a public or a private image. To add blank disks, we do not need to specify an image source. Optionally, we can include the **diskSizeGb** and **diskType** properties.
2. Using client libraries.

Client libraries provide better language integration, improved security, and support for making calls that require user authorization.

The available client libraries are categorized as

* 1. community libraries
     1. libcloud
     2. jclouds
     3. fog.io
  2. Google cloud client library
     1. [Google Cloud Node.js Client Library](https://cloud.google.com/compute/docs/api/libraries#google_cloud_nodejs_client_library)
  3. Google API client libraries
     1. [Google APIs Java Client Library](https://cloud.google.com/compute/docs/api/libraries#google_apis_java_client_library)
     2. Google APIs JavaScript Client Library
     3. Google APIs Ruby Client Library
     4. Google APIs Node.js Client Library
     5. Google API Objective C Client Library
     6. Google API Objective C Client Library
     7. Google API PHP Client Library
     8. Google API Python Client Library

From thus listed client libraries we have picked the java client library and tested it in creating a new instance and listing the available instances.

2.3 Accessing the VM from our own laptop:

In this section, we will explain how we were able to access the VM from our own laptop.

The first step was to define the firewall rule that will be used by TCP/UDP connections. This can be done in several ways, but we did it by using gcloud tool.

gcloud compute firewall-rules create default-allow-http-8080\

--network default \

--action allow \

--direction ingress \

--rules tcp:8080 \

--source-ranges 0.0.0.0/0 \

--priority 1000 \

And for the UDP

gcloud compute firewall-rules create allow-udp\

--network default \

--action allow \

--direction ingress \

--rules udp:8181 \

--source-ranges 0.0.0.0/0 \

--priority 1000 \

After defining these rules what we did was, we created 4 different java classes, two for TCP and the rest for UDP communications. Then we run the server codes on the VM after installing the jdk on the specified instances. From our laptop we run the client codes by providing the external IP of the VM on the specified port. (8080 for tcp client and 8181 for the udp).