Lab 05 - Cloud Computing

Objectives

- Learning about cloud providers
- Using the **gcloud** CLI tool for Google Cloud interactions
- Creating and using a Reverse SSH Tunnel

Contents

Tasks

- 01. [15p] Initial setup
- 02. [50p] The compute engine
- 03. [30p] Reverse SSH
- 04. [5p] Terminating instances
- 05. [10p] Feedback

Proof of Work

Cloud computing has gained tremendous popularity in recent years. In fact, most companies opt to use hosting solutions such as AWS or Microsoft Azure in stead of dedicating resources for building and maintaining their own datacenters. Today, we're going to take an introductory look at what Google Cloud has to offer. That being said, we're not going to take what some may call the regular approach by using the web interface for managing VM instances. In stead, we are going to be using the **gcloud** Command Line Interface (CLI). This tool is much more powerful and has the added benefit of allowing you to integrate it in your **bash** scripts. Yes, sooner or later it may come to that…

Tasks

01. [15p] Initial setup

[5p] Google Cloud account

While there are many cloud providers (e.g.: AWS, Microsoft Azure, DigitalOcean, etc.), today we are going to use Google Cloud. By signing up here [https://gcp.secure.force.com/GCPEDU?cid=7pKiq86GWNFwQ%2BCpTFn9pLUtjIfhChsYLm9v2VmPRz1Zuc9546iEOA1e97Br2Did/] with your university email

(`${YOUR_ID}@stud.acs.upb.ro` most likely), you will get $50 in credits to play around with their infrastructure up until the 10th of April. In case you aren't forwarding your UPB emails to/from your main account via IMAP/POP [https://support.google.com/mail/answer/7104828?hl=en], you can access them in outlook [https://outlook.office365.com/].

[5p] Google Cloud SDK

Normally, cloud provides give you the option to access a web dashboard [https://console.cloud.google.com/]. Using this is fine for starting up one virtual machine, or checking your billing settings. However, if you want to do some automation work, you will want to utilize the gcloud SDK [https://cloud.google.com/sdk/docs/install]. By following the instructions here, you should be able to install the **gcloud** CLI application. If the steps for Ubuntu don't work (very likely on WSL), try the more generic approach for Linux [https://cloud.google.com/sdk/docs/install#linux] (i.e.: download a *.tar.gz* and run the install script). Remember: Ubuntu is a Linux distribution! Finally, you will be asked to run:

```
# create default configuration for usage with gcloud
$ gcloud init
```

In addition to logging in with your Google account, you will also be asked to create a new project. Because of how Google Cloud is designed, this project must have a *globally unique* ID. So don't go for something obvious like *"lab-ii"*, but in stead derive something from your university account name. That should be unique enough.

[5p] Billing

CLI tools like **ip** and **gcloud** work based on a more modern paradigm. If older programs mainly use flags (i.e.: `--this, -t`) to specify what functionality should be invoked at runtime, these use commands and subcommands [https://clig.dev/#subcommands] for a more intuitive classification. For example:

```
# <tool_name> <subject> <action>
$ gcloud projects list
PROJECT_ID      NAME            PROJECT_NUMBER
lab-radumantu   lab-radumantu   1234567890
```

If you're unsure what a command does, add `--help` anywhere to get a manual page. Otherwise, check the web reference [https://cloud.google.com/sdk/gcloud/reference].

If you're still using **zsh**, first of all congrats. Second, try tab completion to get a list of possible commands based on what you've already written:

```
$ gcloud compute networks <TAB>
create                  get-effective-firewalls  subnets
delete                  list                     update
describe                peerings                 vpc-access
```

Note that sometimes, the feature that you want to access is in fact hidden behind the **alpha** or **beta** commands. These commands indicate that the development version of regular commands should be used in stead.

```
# try to list your configured billing accounts
$ gcloud billing accounts list
ERROR: (gcloud.billing) Invalid choice: 'accounts'.
This command is available in one or more alternate release tracks.  Try:
  gcloud alpha billing accounts
  gcloud beta billing accounts
```

```
# now try the same thing with the alpha variant of the command
$ gcloud alpha billing accounts list
ACCOUNT_ID            NAME                          OPEN   MASTER_ACCOUNT_ID
XXXXXX-XXXXXX-XXXXXX  Billing Account for Education  True
```

Now knowing your *project ID* and your *billing account ID*, it's time to link the two. This way, when you request resource allocation from the **gcloud compute** engine, Google will know to use the $50 credit account (charges are usually made at the end of the month). While there's also an alpha version [https://cloud.google.com/sdk/gcloud/reference/alpha/billing/accounts/projects/link] of the following command, we'll be using the *recommended* **beta** variant:

```
# link billing account to gcloud project
$ gcloud beta billing projects link ${PROJECT_ID} --billing-account ${BILLING_ACC_ID}
```

Before we proceed to actually starting instances in different locations, here's one final config that we should do.

```
# set default project id
$ gcloud config set core/project ${PROJECT_ID}

# check that new value was saved
$ gcloud config list
[core]
...
project = ${PROJECT_ID}
```

Setting the *core/project* propriety is optional. However, not setting it would have meant that every time you invoked a **gcloud compute** command, you would have been required to also pass a `--project=${PROJECT_ID}` flag. Which is annoying…

02. [50p] The compute engine

In this exercise, we will instantiate a virtual machine using the **gcloud compute** engine. This may not be as straightforward as you expect. The reason for this is that there are many aspects to consider. For example, in what datacenter do we want our instance to reside. Do we want a public IP address assigned to it? Looking at the gcloud compute instances create [https://cloud.google.com/sdk/gcloud/reference/compute/instances/create] command, it may first appear intimidating. Let's take it step-by-step and discover what we need to create a VM. With each step, make sure to write down the parameters that you'll need later.

[5p] Task A - Enabling the compute service

Google Cloud offers a number of services, none of which are enabled by default. One of them is the **compute** service (i.e.: `compute.googleapis.com`) that lets us create VM instances. When running a command that requires a certain service that was not previously enabled, you get a prompt asking you if you want to enable it then and there. In this case, we'll do it manually. Note that this may take a bit of time, up to a couple of minutes.

```
# get full list of available services
$ gcloud services list --available

# enable the compute service
$ gcloud services enable compute.googleapis.com
```

[5p] Task B - Configuring a SSH public key

Once your VM is up and running (at *Task G*), you will want to be able to SSH into it. For this to happen, you will have to configure a public SSH key. This key will be automatically copied into *~/.ssh/authorized_keys* at creation. If you *still* don't have a SSH keypair generated, now's a good a time as any (see ssh-keygen [https://www.ssh.com/academy/ssh/keygen]).

This is one annoying aspect of Google Cloud that I wish they'd change. You can't choose your default username for the VMs you create. In stead, they derive a username from your account's gmail address. After you run the next command, look for the **username** variable in the output and write it down.

```
# upload your public SSH key to gcloud
# --ttl 0 means that the key does not have an expiration date
$ gcloud compute os-login ssh-keys add --ttl 0 --key-file ~/.ssh/id_rsa.pub
```

[5p] Task C - Selecting a base image

First things first. What OS do we want to run on our machine? A Windows server [https://www.microsoft.com/en-us/windows-server]? Maybe CentOS [https://www.centos.org/]? Nay – let's look for something familiar, namely Ubuntu. Make sure to take note of the *PROJECT* and *FAMILY* columns:

```
# list available base VM images
$ gcloud compute images list
```

[5p] Task D - Selecting a region

All cloud providers worth their salt will offer you a number of physical locations (datacenters) where to deploy your instance. Locality is very important when offering web services. Normally, this is a difficult task. Can you imagine YouTube running on a single server somewhere in the US and you accessing it from SEA? Many people use Content Delivery Networks (CDN) [https://www.cloudflare.com/learning/cdn/what-is-a-cdn/] for this task. Even DigitalOcean, a rather important cloud provider uses CloudFlare as a proxy for their HTTP servers.

CDNs offer many advantages. The reason why DitigalOcean is using CloudFlare despite having the resources themselves is Distributed Denial of Service (DDoS) protection [https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/]. This protection however, comes at a cost that is not necessarily monetary in nature. CDNs usually have access to the private communication between you and said HTTP server, even if you are using HTTPS [https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/]. Why? Because they need to perform deep packet introspection in order to classify malicious traffic as such.

You can read up on Google's regions and zones [https://cloud.google.com/compute/docs/regions-zones]. When working with your own funds and not with free tier accounts [https://cloud.google.com/free/docs/gcp-free-tier] or education credits, you will want to consult their regional pricing model [https://cloud.google.com/compute/all-pricing]. Usually, US-based datacenters are much cheaper.

```
# show available regions
$ gcloud compute regions list

# show zones in selected region
$ gcloud compute zones list --filter ${REGION}
```

[5p] Task E - Selecting a machine type

When selecting the number of Virtual CPUs (vCPU) and RAM for your VM, you will have to choose from a list of presets. These presets may vary depending on the region.

It is not unusual for cloud providers to limit the number of vCPUs that you may reserve, especially for personal accounts (i.e.: not organizations). AWS for example automatically imposed a 128 vCPU limit (across all VMs registered under an account) some time ago, for default users. This parameter was automatically set at account creation. So people who created AWS accounts a while back may have a 1024 vCPUs limit instead. The reason for this is to limit the losses that they may incur from a bad actor that registers a credit card with say, $5 and no intention to actually pay for their resource usage when charged. In AWS's case, this limit can be increased by contacting support. Since RAM is an inexpensive resource in comparison, it's not usually a limiting factor.

```
# show available flavors for your selected zone
$ gcloud compute machine-types list --zones "${YOUR_ZONE}"
```

[5p] Task F - Selecting a storage option

Storage options refer to HDDs/SSDs and can be separated from the instance you create. Meaning that you can delete the instance and still keep your storage image intact, in case you may want to mount it to another instance in the future (think moving a USB stick between PCs). For this exercise, we won't dive into this specific use-case and delete the virtual storage device together with the instance (at the end of the lab). A 10G disk should be *more* than sufficient. Again, the storage device selection is specific to each zone.

```
# select storage type for selected zone
$ gcloud compute disk-types list --zones "${YOUR_ZONE}"
```

[5p] Task G - Creating the instance

Finally, it's time to start up our VM. In addition to all the parameters that you've selected until now, you will also have to give the instance a name.

```
# create the instance
$ gcloud compute instances create ${INSTANCE_NAME} \
        --image-family   ${IMAGE_FAMILY}           \
        --image-project  ${IMAGE_PROJECT}          \
        --zone           ${ZONE}                   \
        --machine-type   ${FLAVOR}                 \
        --boot-disk-type ${DISK_TYPE}              \
        --boot-disk-size ${DISK_SIZE}              \
        --metadata enable-oslogin=TRUE

# show active instances
$ gcloud compute instances list
```

[5p] Task H - Connecting to your instance

Using the username that was generated for you (see Task B) and the Public (External) IP address of your VM, connect to it via SSH. The Ubuntu image comes with a pre-configured SSH server. But remember that after you created the instance, it still requires ~1m to boot and start up the services.

```
# connect to your instance
$ ssh ${USERNAME}@${PUBLIC_IP}
```

In case you lost your private SSH key in the meantime (how?), you can also connect using **gcloud compute ssh**, which will generate a Google Cloud-specific SSH key for you. The normal method above is preferable to this.

[10p] Task I - Configuring the firewall

Google Cloud implements a software firewall with a few default rules that allow, for example, incoming SSH connections. If you want to run a server on your VM and contact it from a public network, you will need to add a new rule, whitelisting it.

Without getting into too much detail regarding networking stuff, IP addresses represent a network interface on your machine. In contrast to an IP address, a port does not have a hardware correspondent. It's a 16-bit number that represents an endpoint on your machine. This endpoint can be used by *a single process at a time* to either listen for incoming connections or establish outgoing connections. Some port numbers have consecrated meanings (e.g.: 22 - SSH, 80 - HTTP, 443 - HTTPS) and are reserved for servers. Why servers and not clients? Because the clients initiate connections and need to know where to find the servers. The servers don't care what port the clients are running on; anything will do. Port numbers are split into three categories:

- **0 - 1023** : system / well-known ports – don't fuck with these!
- **1024 - 49151** : reserved ports – usually used for servers of any kind
- **49152 - 65535** : ephemeral ports – used for dynamic connections

Next, we want to start a **netcat** server on the gcloud instance and send a text message from our localhost. The **netcat** server will run on port 8989, so we want to whitelist it. The **tcp** part in the commands below refers to the TCP protocol [https://en.wikipedia.org/wiki/Transmission_Control_Protocol]. You don't need to understand what protocols are. These will be covered in the 3rd year Local Networks [https://ocw.cs.pub.ro/courses/rl] course. For the sake of this task, mentioning it is unavoidable. If you want to know more, ask your assistant.

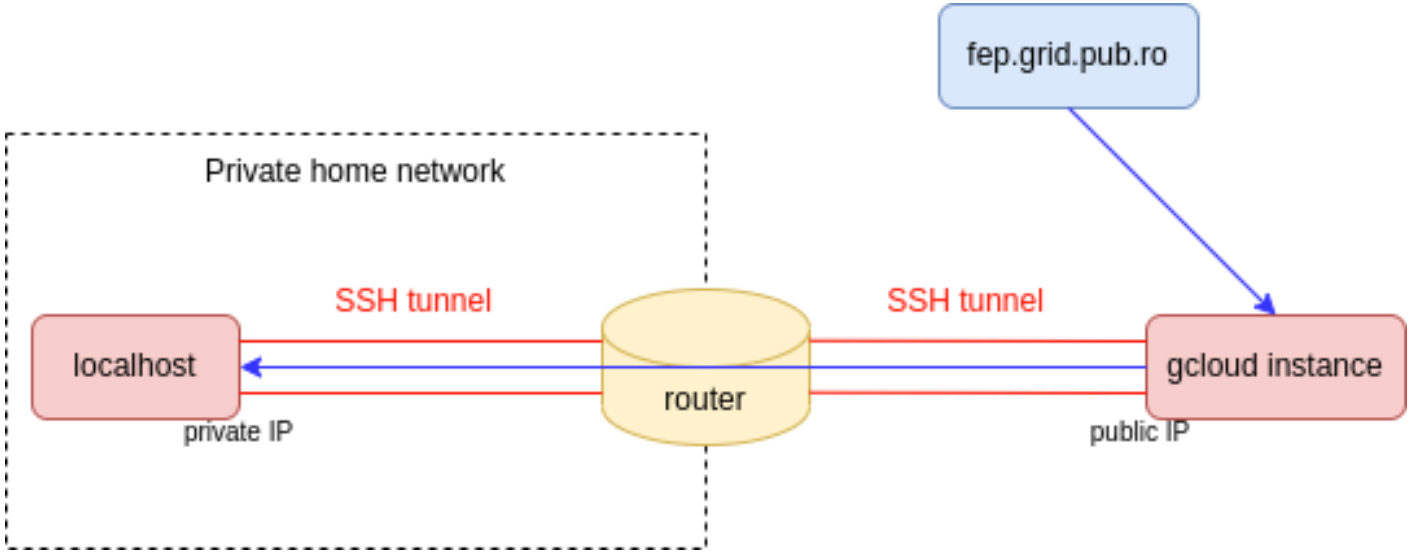
In the following commands *[localhost]* means that the command should be executed on your computer, and *[gcloud]* means that the command should be executed on the VM, over SSH.

```
# display current firewall rules
[localhost]$ gcloud compute firewall-rules list

# add a new firewall rule
[localhost]$ gcloud compute firewall-rules create ${SOME_RULE_NAME} --allow tcp:8989

# start a netcat server in listening mode (-l)
[gcloud]$ nc -l 8989

# send some data to the server
[localhost]$ echo "Hello world!" | nc ${PUBLIC_IP} 8989
```

These firewall rules are not specific to your VM, but to your account! They apply to all your VM instances.

Linux has a built-in firewall system called **iptables**. If you want to create instance-specific firewall rules, you can use this. We won't be covering **iptables** in this lab, but it's nice to know that it's there.

03. [30p] Reverse SSH



Loosely speaking, there are two types of IP addresses:

- **Public**: what your Google Cloud instance has, and what allows you to contact it from anywhere over the Internet.
- **Private**: what your router allocates to your devices (laptop, phone, etc.) in your local network.

Public addresses can be uniquely identified in the Internet. Private addresses can't. Why not, you ask? Well… can you find the IP address assigned to your machine's network interface? Hint: **ip addr show**. That *exact* IP address is shared by millions of other devices, all across the world, in their respective local networks. When you initiate connections from your private network, the router performs a process called Network Address Translation (NAT) [https://avinetworks.com/glossary/network-address-translation/] which uses a single Public IP to represent multiple Private IPs. Unless you have administrative access to said router (too add some custom configurations), clients outside your network will be unable to initiate contact with individual machines inside your network. And most times you don't…

In this exercise, we will set up what is called a reverse SSH tunnel. This tunnel is a persistent two-way communication channel between your computer and the gcloud instance. While this connection is initiated by you (from your private network, to a public server), someone on the other side can piggy back on this channel to initiate connections with you, *in your private network*. The reason for this is that it doesn't target your IP, specifically, in its request. In stead, it sends its request into the gcloud endpoint of the channel and it just so happens that the other endpoint is located on your machine.

To be more specific, the scenario is as follows:

1. You create a SSH channel from your machine to the google cloud instance
2. Then, you connect to fep.grid.pub.ro. SSH-ing from fep back to your computer should be impossible. Imagine your fep instance being you on vacation, trying to access your home computer.
3. To sidestep this problem, you will SSH from fep to your Google Cloud instance, and from Google Cloud to your localhost via the SSH channel.

In the following commands *[localhost]* means that the command should be executed on your computer, *[gcloud]* on the Google Cloud VM, and *[fep]* on fep.grid.pub.ro.

If you are going to SSH from fep.grid.pub.ro to your gcloud instance, you will need to:

1. create a SSH public keypair on your fep account (if not already there).
2. configure the fep public key on your google cloud instance.

If you don't do this, access from fep to your gcloud instance will be denied.

Also, if you want to SSH from gcloud to your localhost via the SSH tunnel, note that you must have a SSH server installed and running.

```
# create a public keypair on fep, if you don't already have one

# create the .ssh directory (if not already there)
[gcloud]$ mkdir -p ~/.ssh

# print out your fep.grid.pub.ro public key and copy it
[fep]$ cat ~/.ssh/id_rsa.pub

# configure your fep.grid.pub.ro public key on gcloud instance
[gcloud]$ vim ~/.ssh/authorized_keys

# install a ssh server on your computer
[localhost]$ sudo apt install openssh-server

# check if the ssh server is running; if not, start it
[localhost]$ systemctl status sshd
[localhost]$ sudo systemctl start sshd
```

And now for the main part:

```
# create a reverse ssh tunnel from your computer to the cloud instance
[localhost]$ ssh -T -N -R 43210:localhost:22 ${GCLOUD_USERNAME}@${GCLOUD_IP}

# show tcp listeners (bound ports, processes, etc.)
[gcloud]$ netstat -tlpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:43210         0.0.0.0:*               LISTEN      1931/sshd: ........
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      448/systemd-resolve
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      893/sshd: /usr/sbin

# test that the reverse ssh tunnel works
[gcloud]$ ssh ${LOCALHOST_USERNAME}@localhost -p 43210

# connect from fep.grid.pub.ro to your localhost via gcloud instance
[fep]$ ssh -J ${GCLOUD_USERNAME}@${GCLOUD_IP} ${LOCALHOST_USERNAME}@localhost -p 43210
```

Let's take a look at some of the arguments used in these commands:

- `ssh -T -N -R 43210:localhost:22 …`
  - `-T`: do not start a shell on the remote computer (we're not using the connection for that, remember?)
  - `-N`: do not execute one-shot commands either (combined with `-T`, this makes **ssh** do nothing)

    - `-R 43210:localhost:22`: when logged in on the gcloud instance, writing data to port 43210 with itself (i.e.: localhost) intended as the destination, will ensure that the data actually ends up on whoever initiated the tunnel, on port 22 (the SSH server port).
- `netstat -tlpn`
    - `t`: filter for TCP protocol (SSH actually runs *over* TCP)
    - `l`: show ports where processes are listening for new connections
    - `p`: show the PID and name of the program that is listening
    - `n`: use numeric IP addresses in the output
- `ssh -J ${GCLOUD_USERNAME}@${GCLOUD_IP} ${LOCALHOST_USERNAME}@localhost -p 43210`
    - `-J ${GCLOUD_USERNAME}@${GCLOUD_IP}`: create a SSH connection with google cloud as an intermediary hop; it acts like you'd SSH to google cloud, and there you would write another SSH command to your final destination.
    - `${LOCALHOST_USERNAME}@localhost`: *"localhost"* here is relative to the jump point (i.e.: google cloud instance), not to fep.
    - `-p 43210`: in stead of using the default SSH server port 22, pretend that it's in fact running on 43210.

Regarding the output from **netstat**:

- `127.0.0.1:43210`: on port 43210 there is a process listening for connections towards IP *127.0.0.1*. This is synonymous with *localhost*, denoting a loopback interface. In other words, this IP always refers to the current machine.
- `127.0.0.53:53`: this IP also corresponds to a loopback interface. The process listening on port 53 for connections to *127.0.0.53* is a DNS [https://www.cloudflare.com/learning/dns/what-is-dns/] caching server that comes pre-installed on Ubuntu. This is not relevant for this exercise.
- `0.0.0.0:22`: on port 22, there is a SSH server called **sshd**, listening for incoming connection requests. The *0.0.0.0* IP signifies a catch-all. In other words, it doesn't matter if the request's destination IP is that of your Ethernet interface, or your WiFi interface, or your loopback interface (*127.0.0.**). This server will take on all of them.

Reiterating over the last command, since it might still be a bit unclear:

- You are on fep.grid.pub.ro and you SSH to the Google Cloud instance (the `-J` part)
- Once on the Google Cloud instance, it may appear that you're trying to do something utterly insane: you are trying to log in via SSH with your personal computer's username, on the very same Google Cloud instance (that's your localhost at that moment in time), using a SSH server that doesn't run on port 22 (like a normal SSH server) but on port 43210…
- When you initiate that connection you realize that localhost:43210 is in fact a… *wormhole*. localhost is not in fact the Google Cloud instance, but your personal computer. Port 43210 is not in fact port 43210, but port 22. But you already know that, didn't you?

04. [5p] Terminating instances

Since we're done with the VM, we might as well terminate it. Unless we want it to keep running and eating up our funds, needlessly.

```
# terminate instance and delete attached disks
$ gcloud compute instances delete ${INSTANCE_NAME} \
```

```
        --zone ${ZONE}                          \
        --delete-disks all
```

05. [10p] Feedback

Please take a minute to fill in the feedback form [https://forms.office.com/Pages/ResponsePage.aspx?id=usiMLdqNNEOeXPrCCS6brJoxNMaLqNZHpd8YaA7IhDNUNVVLQ0lQV0tJRzBaRjhOQzdNOVhYWklBVC4u] for this lab.