# Cloud Computing

**Winter Term 2024/2025**

*Practical Assignment No. 2*

**Due: 09.12.2024 11:55 pm**

The primary goal of this assignment is to gain insight into a cloud computing platform from a provider perspective. Building upon gained experience from practical assignment no. 1, you will set up your own OpenStack cloud computing platform, configure the network, perform functionality checks, and deploy an exemplary data processing application on OpenStack VMs.

# Prerequisites

This assignment consists of several steps. Work with checkpoints. If you reach an operational status, wrap everything up to a script and verify that the outcome is reproducible. Move on to the next step after that. It will prevent you from getting lost around all the required tools and systems.
Being stuck at some point, a good strategy is to clean everything, run the deployment until a known and working checkpoint and start again from there.
It is also important to shut down your gc VMs when you do not need them. Again, checkpoints will help you to get to your previous working state.
You need a machine (your local laptop or another gc VM) from which you run the deployment scripts. It must not be one of the machines from "1. Virtual Machines"! We refer to it as "local machine".

## Linux OS

Linux is required. If you use Mac OS or Windows privately, work in a virtual machine (either locally, such as VirtualBox, or use one of the Cloud platforms). This task was tested with Ubuntu. Therefore, we recommend using it.

## Google Cloud

Refresh your knowledge about the google cloud platform from assignments no. 1.

# 1. Virtual Machines

Write a well-commented Shell script that prepares a virtual environment on the gc platform. It will be used to deploy OpenStack. Note: We recommend using the gcloud commands as in the previous assignment. However, if you would like to work

with Ansible or Terraform for this task, this is also fine – just note that you will then most likely receive less support from fellow students or us in the ISIS forum.

- You can first manually experiment with the **`gcloud`** commands and make notes, before finally combining the commands into a script.
- Each command in the scripts should be commented, make sure the scripts are clean and not cluttered with unnecessary text.
- After writing the script, test it: delete all running disks, images, VMs, networks and firewall rules, run the script, and check that the result is satisfactory.
- Do **not** include any private information such as your access key or secret key in your submission (if necessary, replace them with dummy strings).
- All requirements listed below should be covered by at least one command.
- The key from practical assignment no. 1 can be used. We therefore skip the key creation here.

**Requirements:**

1. Create two additional VPC networks "cc-network1" and "cc-network2" with subnet-mode "custom".
2. For each created network, create a respective subnet "cc-subnet1" and "cc-subnet2" and assign different IP ranges to them. Furthermore, "cc-subnet1" needs a secondary range (check the --secondary-range parameter).
3. Read up about nested virtualization on the gc platform: https://cloud.google.com/compute/docs/instances/enable-nested-virtualization-vm-instances?hl=en
4. Create a disk based on the "ubuntu-2204-lts" image family and set the size to at least 100GB.
5. Use the disk to create a custom image and include the required license for nested virtualization.
6. Start 3 gc VMs that allow nested virtualization:
   - Name them "controller", "compute1", "compute2" (Hint: Start with 1 gc VM and verify everything below before extending it to 3).
   - The image should be the previously created custom image that supports nested virtualization.
   - Set the tag "cc" for each gc VM.
   - Choose machine type "n2-standard-2"
   - The gc VMs must have 2 NICs (check the --network-interface parameter). The first NIC should be connected to cc-subnet1, the second NIC should be connected to cc-subnet2.
7. Create a firewall rule that allows all TCP, ICMP and UDP traffic for the IP ranges of cc-subnet1 and cc-subnet2. Restrict it to gc VMs that have the "cc"-tag.
8. You can read up on all required ports of OpenStack: https://docs.openstack.org/install-guide/firewalls-default-ports.html
   You must open them to be accessible from any external IP address. For simplicity, you can create a firewall rule, that allows all TCP and ICMP traffic from external IPs for cc-network1 (Caution: This is OK for this

assignment, in production you need to be careful opening up your environment to the internet.). Restrict it to gc VMs that have the "cc"-tag.

**Verify:**

- These verifications are not required to be part of the submission but help to verify that the environment is ready for OpenStack deployment.
- Check nested virtualization support. Each gc VM should have VMX enabled. Execute: `grep -cw vmx /proc/cpuinfo`
  The response indicates the number of CPU cores that are available for nested virtualization. A non-zero return value means that nested virtualization is available. If you get 0, you need to revisit step 3.
- Each instance should have 2 NICs and 2 internal IP addresses. Check via gc dashboard, gc command line or run `ifconfig` or `ip link show` within the gc VMs.
- ssh to all gc VMs must work:
  `ssh -i /path/to/id_rsa <user>@<public_ip>`
  If it is not working, check you firewall rules (especially step 8).
- Connect via ssh to the gc VMs and try to ping both internal IP addresses of the other 2 gc VMs. If it does not work, check firewall rules (step 7).
- Connect via ssh to instances and test TCP traffic to internal IP addresses of the other 2 instances: `nc -z -v <internal ip adress> 22`
  It must be successful. If it does not work, check firewall rules (step 7).

**Outputs:**

- Script **spinup-gcp-for-openstack.sh**
  - Containing all commands for preparing and starting your gc VMs, including comments explaining what the commands do.
- Alternatively, if you use Ansible or Terraform, submit an artifact with the same name as the script.

# 2. OpenStack Setup

We will use Ansible and kolla-ansible to deploy OpenStack on the 3 previously created virtual machines (gc VMs). Revisit the lecture slides on infrastructure as code. It will help you to understand Ansible and kolla-ansible.
The steps below are mostly from the official kolla-ansible guide (https://docs.openstack.org/kolla-ansible/latest/user/quickstart.html) but include some additional comments to catch frequent pitfalls.
The output of each relevant step is piped into an output log file via `tee`. You will need to submit these output files.

**General requirements:**

1. You must have a non-root user with sudo rights on your local machine. It will make your life easier if a password is not requested when running commands with sudo.
2. Check the kolla-ansible quick start guide. Use the provided requirements.txt files to install ansible, kolla-ansible and the openstack cli.

We recommend using a virtual environment for this. We tested it with python 3.9.

3. It is important that the versions from the provided requirements.txt are used. Otherwise, your openstack deployment will probably fail. However, feel free to experiment with other versions.

**Prepare kolla-ansible:**

1. Download the provided kolla-ansible archive from ISIS and unpack into your working directory.
2. Open the files "multinode" and "globals.yml". Replace the placeholders "[******<>******]" with respective values.
   Example:
   `kolla_internal_vip_address: "10.34.122.100"`

3. The interfaces in the globals.yml are default for gc VMs and should be correct. However, you can verify this by checking the interfaces on the "controller VM" (the gc VM you named "controller").

**Install OpenStack:**

We prepared a script `deploy-openstack.sh`. However, it is recommended to execute the commands manually one after another and verify that they are executed without errors. Otherwise debugging will be very hard.

1. `cd` into your openstack setup working directory.
2. Run `ansible -m ping all -i ./multinode` to verify that ansible is setup correctly and the gc VMs are reachable. In case of failure:
   - Check correct installation of ansible
   - Revisit **Prepare kolla-ansible** and check if the placeholders were replaced correctly.
   - Check again if you can connect via ssh to the gc VMs.
3. Run the commands from `deploy-openstack.sh`.
   - You need to adjust the fix-hosts-file.yml to comply with the IP ranges that you set up for your ""cc-subnet1" and ""cc-subnet2".
   - In some rare cases the kola-ansible "deploy" command fails due to runtime conditions (especially when pulling docker images). You can rerun all ansible and kolla-ansible scripts (idempotence). Verify that the occurred error is reproducible before searching for a solution.
   - All steps on the gc VMs must pass. If not, try to resolve problems. Googling the error usually helps. Ask in the ISIS forum if you are stuck.
   - `tee` removes colours from the ansible output. For initial testing and possible debugging, you can remove the piping into `tee`. Add it again when generating logs for final submission.
   - Do not be afraid to tear everything down and redeploy it. Remember to work with checkpoints.
4. If everything is successfully executed, check the public IP address of the "controller VM" (the gc VM you named "controller") via your browser. It

should display the OpenStack login landing page. If not, check your firewall rules.

Congratulations, you are now a cloud provider and are hosting your own OpenStack cloud computing platform on the google cloud computing platform.

**Outputs:**

- Files **pre-bootstrap.log**, **bootstrap-servers.log**, and **deploy.log**

# 3. Configure OpenStack

Execute the steps and verify that everything is working. Write a well-commented Shell script **prepare-openstack.sh** for steps 7-11. Add at least one comment for each of the steps.

1. Activate the previously configured python3 virtual environment on your local machine (or any other machine acting as your controller server for ansible) to use the OpenStack CLI.
2. The kolla-ansible post-deploy command should have generated an admin openrc.sh file. Open it and get the admin login credentials.
3. Login via the OpenStack dashboard. You can verify that all necessary resources are created in OpenStack.
4. Run `source admin-openrc.sh` to set environment variables that are used for authentication.
5. Test if the openstack CLI is working: `openstack service list.` It must work. If not, check the firewall rules.
6. Run the scripts `import-images.sh` and `create-ext-network.sh`
    - Depending on the network connection of your local machine, the image upload might take a while.
7. Create a new security group "open-all" (security groups are used to define firewall rules in OpenStack).
8. Add rules to the "open-all" OpenStack security group. Allow all TCP, UDP and ICMP ingress and egress traffic (Since it is a firewall behind a firewall, we consider it to not be too much of a threat. However, keep in mind that you need to set such rules cautiously when working on production systems).
9. Create another key-pair and import it to OpenStack. Copy the private key to your controller VM or generate it there (home or .ssh directory). Verify its permissions to be 400.
10. Start a VM instance on OpenStack (we refer to it as "OpenStack VM"). Make sure to use the ubuntu image, the medium flavor, admin-net, the

"open-all" security group, and your imported public key. The OpenStack VM must reach state "RUNNING". Check it via cli or Dashboard.
11. Assign a floating IP to the OpenStack VM.
12. From your controller VM, try to ping the floating IP of your OpenStack VM. It is not working. Think about why and answer question 2.
13. Copy the `iptables-magic.sh` script to your controller VM. Run it with **sudo**. Look into it and answer question 3.
14. From your controller VM, try to ping the floating IP of your OpenStack VM. It should work.
15. From your controller VM, try to connect to the OpenStack VM via ssh. If you followed step 9, the key should be located on the controller VM. The username is "ubuntu".
16. From the OpenStack VM, try to ping an external IP address, e.g. 8.8.8.8. It should work.
17. From the OpenStack VM, execute
`wget 169.254.169.254/openstack/2018-08-27/network_data.json`
You will need to submit the **network_data.json** file.

**Questions:**

1. Give a short overall explanation in your own words of what you did in this assignment (max. 200 words). (5 Points)
2. Initially, the OpenStack VM was not reachable from the controller VM (step 12). Why? (2 Points)
3. Look into the iptables-magic.sh script. What is happening there? Describe every command with 1-2 sentences. (5 Points)

**Outputs:**

- One shell script **prepare-openstack.sh**
  - Containing all commands from step 7-11 with respective comments.
- The **network_data.json** file from step 17
- Text with answers to the questions.
  - Please submit .txt or .pdf files via ISIS or use the submission text field. Other formats will not be accepted.

# 4. Execute Data Processing Application

One advantage of cloud computing environments is the dynamic allocation of resources, depending on needs. In the following, we will deploy an example Apache Spark application in our cloud environment and collect the results. In a subsequent assignment, we will get back to this topic again and use even more sophisticated technologies, e.g., for dynamic scaling of such applications.

Create 3 Openstack VMs, with specifications as outlined before. You are free to reuse the already created VM from the previous task. On each Openstack VM, install Java 8 as well as Spark 3.4. Follow the documentation of Spark for a correct configuration of Spark on each node. Then, form a spark compute cluster, where one Openstack VM hosts the Spark master and the other two Openstack VMs each host a spark worker. Once the cluster has been formed, use the JavaSparkPi job which is packaged as an

example with Spark, and execute it with input parameter ("slices") of 500. Eventually, collect the logs of the workers and the master (the location depends on your chosen installation path).

**Question:** Look at the logs and the example application. What happens here? Describe what you see. How is the program structured and how are tasks distributed? (10 Points)

**Outputs:**

- 3 Spark log files (2x worker, 1x master)
- Text with answer to the question.
    - Please submit .txt or .pdf files via ISIS or use the submission text field. Other formats will not be accepted.

# Submission Deliverables

Submit your solution on the ISIS platform as individual files. Please submit ONLY the necessary files and use exactly the given file names!
Submit the text with answers to the questions directly in the submission text field or use .txt or .pdf files.

Expected submission files:

- spinup-gcp-for-openstack.sh
    - **15 points**
- `pre-bootstrap.log, bootstrap-servers.log, and deploy.log`
    - **10 points (2, 3, 5)**
- `prepare-openstack.sh`
    - **7 points**
- network_data.json
    - **5 points**
- Answers to questions of task 3
    - **12 points**
- 3 Spark log files
    - **9 points**
- Answers to questions of task 4
    - **10 points**

*Total points: 68*

**Final Warning: Please make sure your VMs are shut down when you are not using them! Make good use of your credits :-)**