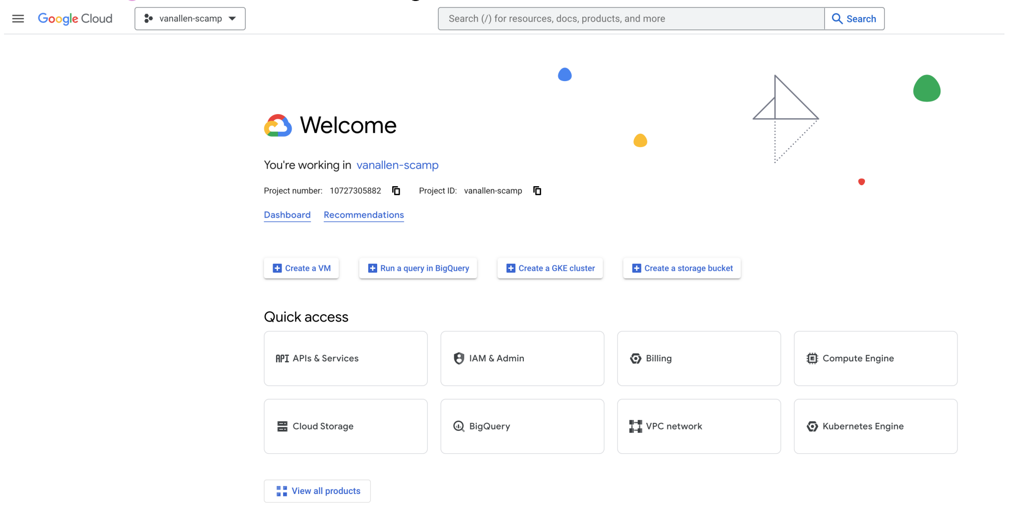
Introduction to GCP VMs and using Terra notebook environments

Recently in the lab, our two main avenues for performing primarily notebook-based analyses with high compute resources have either stopped being maintained (broad server) or come with a lot of headaches (Terra).

IMO, a promising solution to the above is to strip away the Terra UI and notebook management service, only using the underlying GCP VMs and disks to utilize jupyter notebooks.

Creating a virtual machine and persistent disk

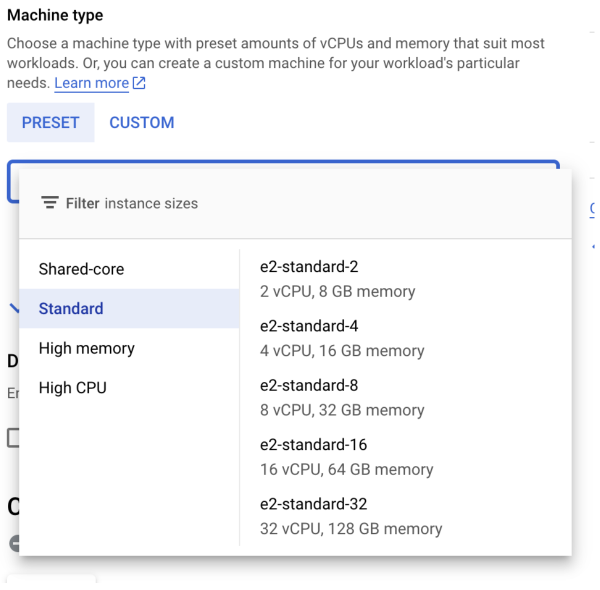
1. Creating a VM instance and attaching a persistent disk through the GCP UI
    1. Ask Brendan and Erin to create a google cloud project for you (e.g. `vanallen-scamp` is mine). You will receive an email to accept the project invitation where you will click the link provided. Then, the project should show up when you navigate to the Google Cloud Consoleand log in.



You're working in vanallen-scamp

Project number: 10727305882 Project ID: vanallen-scamp

    2. Navigate to `Compute Engine` -> `VM instances` tab. Select `Create an instance`.
        1. General lab guidelines for VM naming can be found in our GCP Handbook - Non-Terra
            1. Here, I named mine `scamp-cpu-16` encoding that the VM is using CPUs not GPUs and has 16 GB of memory.
        2. Set region to `us-central1 (Iowa)`
        3. I've been keeping most options as the default. I've been choosing from the Standard machine types.

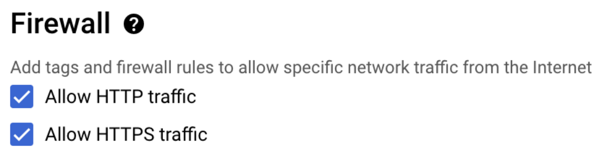


Choose a machine type with preset amounts of vCPUs and memory that suit most workloads. Or, you can create a custom machine for your workload's particular needs.

e2-standard-2
2 vCPU, 8 GB memory
e2-standard-4
4 vCPU, 16 GB memory
e2-standard-8
8 vCPU, 32 GB memory
e2-standard-16
16 vCPU, 64 GB memory
e2-standard-32
32 vCPU, 128 GB memory

        4. Modify the firewall rules.

        1. Allow both HTTP and HTTPS traffic

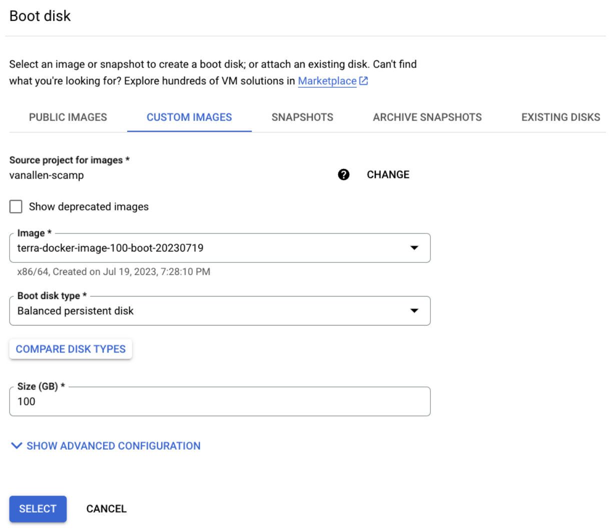


Firewall

Add tags and firewall rules to allow specific network traffic from the Internet

- ☑ Allow HTTP traffic
- ☑ Allow HTTPS traffic

    5. Ask Sabrina to [give you access to her project]() and change the boot disk to be from the Custom Image named `terra-docker-image-100-boot-20230720`. This boot disk already has Docker installed and the following three Terra notebook environments are cached:
        - **R/Bioconductor**: us.gcr.io/broad-dsp-gcr-public/terra-jupyter-bioconductor:2.1.11
        - **Python**: us.gcr.io/broad-dsp-gcr-public/terra-jupyter-python:1.0.15
        - **Default**: us.gcr.io/broad-dsp-gcr-public/terra-jupyter-gatk:2.2.14
          These images are using R version 4.3.0 and Python version 3.7.12.



Boot disk

Select an image or snapshot to create a boot disk; or attach an existing disk. Can't find what you're looking for? Explore hundreds of VM solutions in Marketplace

Source project for images *
vanallen-scamp

☐ Show deprecated images

Image *
terra-docker-image-100-boot-20230719
x86/64, Created on Jul 19, 2023, 7:28:10 PM

Boot disk type *
Balanced persistent disk

Size (GB) *
100

    6. If you don't already have a persistent disk created, you can create and attach a disk at this time in the `Advanced options` section. These operate the same as Terra PDs, where if you delete the VM the persistent disk will remain.
        1. Here I've named mine `scamp-singlecell` to indicate which project's data will be stored here.
        2. Consider creating a [snapshot schedule]() at this time for automatic data back up.
    7. In the `Advanced options` -> `Networking` -> `Network inferfaces` section, click on the drop down arrow. In the `External IPv4 address` section, choose the option to "reserve static external IP address". Note down the IP address, it will be used for navigating to your jupyter notebook in the browser (e.g., [http://33.245.66.245:8080]())
        - Note: Each static IP address costs money per hour. Navigate to `VPC network` -> `IP addresses` and release the static addresses not in use.

2. SSH into the VM from your local terminal
    1. Set your default google cloud project to be the one Brendan and Erin assigned to you using the following command:

```
gcloud config set project {project-id}

#example
gcloud config set project vanallen-scamp
```

    2. SSH into the VM instance using the following command:

```
gcloud compute ssh --zone "us-central1-a" "{instance-name}" --project "{project-id}"

#example
gcloud compute ssh --zone "us-central1-a" "scamp-cpu-16" --project "vanallen-scamp"
```

3. Format the attached persistent disk
   You only have to do this once to a disk. It will wipe your disk clean if you do it again once you have data stored on it
    1. Find the name of the persistent disk you created using the following command:

```
ls /dev/disk/by-id/
```

Note down the name of the disk.

2. Format the persistent disk using the following command:

```
sudo mkfs.ext4 -m 0 -E lazy_itable_init=0,lazy_journal_init=0,discard /dev/disk/by-
id/{persistent-disk-name}

#example
sudo mkfs.ext4 -m 0 -E lazy_itable_init=0,lazy_journal_init=0,discard /dev/disk/by-id/scsi-
0Google_PersistentDisk_scamp-singlecell
```

4. Mount the formatted and attached disk
   1. Create a folder to mount the disk to (specific naming up to you)

```
sudo mkdir /mnt/disks
sudo mkdir /mnt/disks/{folder-name}

#example
sudo mkdir /mnt/disks
sudo mkdir /mnt/disks/scamp-singlecell
```

   2. Mount disk to folder location

```
sudo mount -o discard,defaults /dev/disk/by-id/{persistent-disk-name} /mnt/disks/{folder-name}

#example
sudo mount -o discard,defaults /dev/disk/by-id/scsi-0Google_PersistentDisk_scamp-singlecell
/mnt/disks/scamp-singlecell
```

   3. Set read and write permissions for the disk

```
sudo chmod a+w /mnt/disks/{folder-name}

#example
sudo chmod a+w /mnt/disks/scamp-singlecell
```

Running a Terra notebook environment in a GCP VM

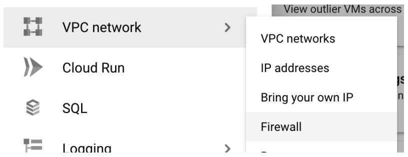
The ready-to-go Terra notebook environments have a lot of upsides. They already have google cloud, git, python, R, pip, conda, jupyter, FISS, and several major python/R packages installed (depending on which environment you choose). The notebook environments also interface with the boot and persistent disk very well. For example, packages that *come with the environment* (ones you did not manually install) are stored on the boot disk. By default, boot disks are deleted when the VM is deleted. This is nice because if you switch to a *new* Terra environment (R -> Python, for example), you wont have all the clutter of the previous environment installed.

However, any user-installed packages or tools will be stored on the persistent disk along with any notebooks and data generated. The persistent disk is **not** deleted when the VM is deleted. Persistent disks can be detached and reattached to any VM instance you create, allowing you to retain *your* packages, tools, data, notebooks, etc in any compute environment.

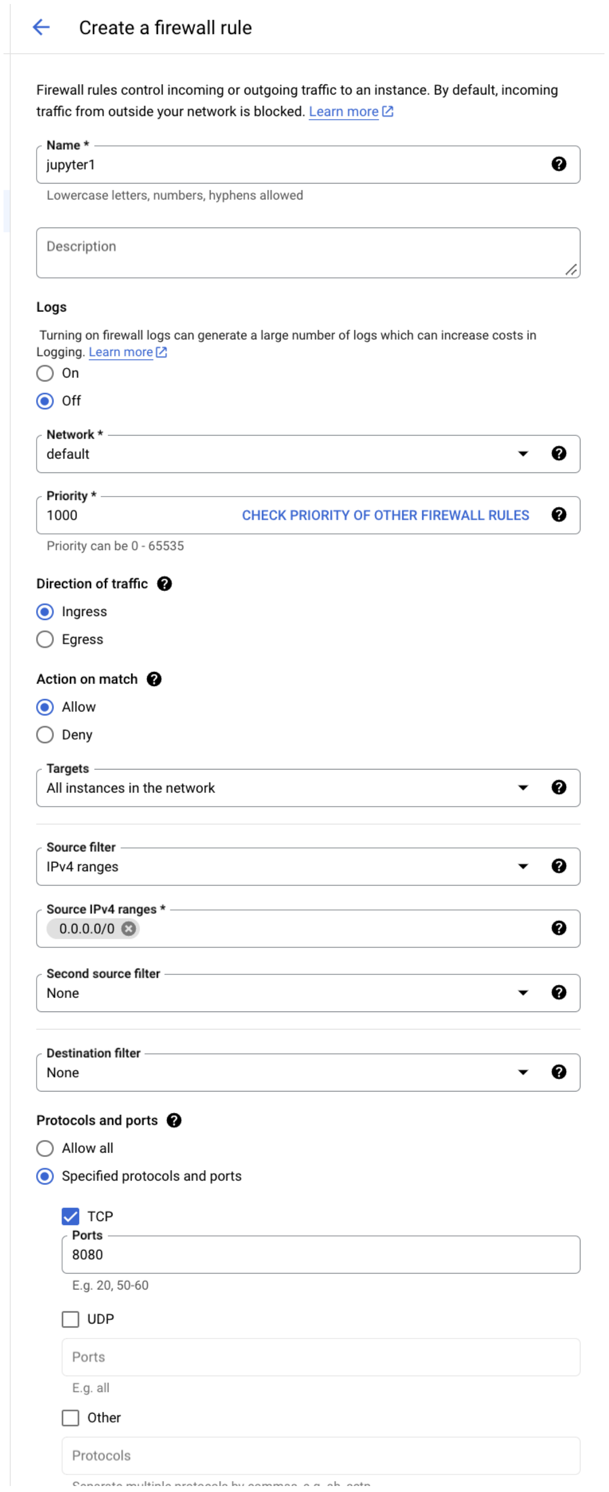
Terra has some good documentation on boot and persistent disks here and here

In this tutorial, I show how you can use the Terra notebook environments in a GCP VM. This allows us to use these nifty environments without having to interface with the Terra UI.

1. Create a firewall rule allowing a specific port number.
    - This will be relevant to running the jupyter notebook in the browser. Navigate to VPC network -> Firewall -> Create Firewall Rule



      Set Targets to "All instances in the network". Set source IPv4 ranges to "0.0.0.0/0". Select TCP Ports and enter "8080". Create the firewall rule.



Create a firewall rule

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. Learn more

Name *: jupyter1
Lowercase letters, numbers, hyphens allowed

Description

Logs
Turning on firewall logs can generate a large number of logs which can increase costs in Logging. Learn more
On [✓ Off]

Network *: default

Priority *: 1000
CHECK PRIORITY OF OTHER FIREWALL RULES
Priority can be 0 - 65535

Direction of traffic
[✓ Ingress] Egress

Action on match
[✓ Allow] Deny

Targets: All instances in the network

Source filter: IPv4 ranges

Source IPv4 ranges *: 0.0.0.0/0

Second source filter: None

Destination filter: None

Protocols and ports
Allow all [✓ Specified protocols and ports]

[✓ TCP]
Ports: 8080
E.g. 20, 50-60

☐ UDP
Ports
E.g. all

☐ Other
Protocols

2. Set persistent disk permissions so that docker can read/write to it.
    - We also need to set the appropriate permissions for our persistent disk prior to running the Terra docker so that when we enter the docker and mount our persistent disk the docker user can read/write to it. The idea is more fully explored in this stackoverflow post.

```
sudo chown -R 1000:100 /mnt/disks/{folder-name}

#example
sudo chown -R 1000:100 /mnt/disks/scamp-singlecell
```

3. Run the Terra docker of choice.
   - Example command below using one of the cached Terra docker images.

   ```
   sudo docker run -e R_LIBS='/home/jupyter/packages' --rm -it -u jupyter -p 8080:8080 -v
   /mnt/disks/{folder-name}:/home/jupyter --entrypoint /bin/bash {terra-docker-image-path}

   #example
   sudo docker run -e R_LIBS='/home/jupyter/packages' --rm -it -u jupyter -p 8080:8080 -v
   /mnt/disks/scamp-singlecell:/home/jupyter --entrypoint /bin/bash us.gcr.io/broad-dsp-gcr-
   public/terra-jupyter-bioconductor:2.1.11
   ```

   - To explain some of this command, we are specifying that we want to interactively run the docker container as the non-root `jupyter` user (this is how its done in Terra notebooks). We specify to place user-installed R packages into the `/home/jupyter/packages` location. We perform port mapping `8080:8080` so that we can access the services running inside the docker (when we connect to a jupyter notebook via the browser). So far, only the `8080` port has worked, not sure why. We are mounting our persistent disk to the `/home/jupyer` location inside of the docker. This means when you are inside of the docker, ONLY the things saved in the `/home/jupyter` path will be saved to the persistent disk. Everything else will not be saved. When you exit the docker, navigate to `/mnt/disks/{folder-name}` to access what you put in `/home/jupyter` when you were inside of the docker.
   - Terra docker images not cached on the boot disk (those cached listed here) **can still be used here**, but the `docker run` command will take significantly longer. This is because it is pulling the docker from scratch. If you don't plan on using any of the cached images, I would recommend clearing the cache using the following command:

   ```
   docker system prune
   ```

   I recommend this because the Terra docker images are huge, and if you will be utilizing other ones, the boot disk may quickly run out of space. Consider creating your own book disk image with the Terra docker images you will be using.
4. Set up gcloud authentication.
   - With newer versions of gcloud, it's no longer possible to authenticate on a machine that doesn't have a web browser (like the GCP VM). The new Instructions below for authenticating a machine without a web browser is from the google cloud SDK documentation You *should* only have to do this once, because the credentials are stored on the persistent disk and could be used with any VM. *I think?*
     1. Once inside Terra docker, run the following command in the GCP VM:

     ```
     gcloud auth login --no-browser
     ```

     2. Copy the long command that begins with `gcloud auth login --remote-bootstrap="`
     3. Paste and run this command on the command line of a different terminal (e.g. your local terminal) that has gcloud installed
     4. Copy the long URL output from the alternative terminal and paste it back to the first machine terminal. Press enter. You should now be authenticated.
5. Transfer Terra persistent disk data to GCP persistent disk
   - Before performing these steps, I would recommend doing your best to clean up your Terra PD, deleting what you no longer need. Similar to cleaning before you move apartments, there is no need to bring junk to a new place!
   1. Make a complete copy of your Terra PD to a google bucket. The google bucket folder (here, `notebook-cache-{date}`) should *not* already exist in your bucket for everything to copy in the right directory structure. This folder will be automatically created by the `gsutil cp` process. Navigate to the **Terra notebook associated terminal** and run the below command.

```
gsutil -m cp -r -L "terraPD_to_gbucket_{date}.log" /home/jupyter gs://{google-bucket}/notebook-
cache-{date}

#example
e.g. gsutil -m cp -r -L "terraPD_to_gbucket_20230724.log" /home/jupyter gs://fc-3a463b92-98d9-
47e3-9d16-4ba001069ee9/notebook-cache-20230724
```

- This will not copy over conda environments unless you had the conda config set to save in a `/home/jupyter` directory. (sorry) Consider exporting environments to .yml files for easy rebuild on GCP VM.
- [Optional] Check if any files were not successfully transferred by inspecting `filtered_output.csv`

```
grep -v "OK" terraPD_to_gbucket_{date}.log > filtered_output.csv
```

2. Copy google bucket folder contents to GCP PD
   1. Run Terra docker of choice, if not already in the docker.
   2. Copy Terra PD contents from google bucket to GCP PD

```
gsutil -m cp -r -L "gbucket_to_gcpPD_{date}.log" gs://{google-bucket}/notebook-cache-{date}/*
/home/jupyter/

#example
gsutil -m cp -r -L "gbucket_to_gcpPD_20230724.log" gs://fc-3a463b92-98d9-47e3-9d16-
4ba001069ee9/notebook-cache-20230724/* /home/jupyter/
```

6. [Option one] Jupyter notebook
   1. Create the jupyter notebook configuration and password.
      - Loosely following this tutorial.

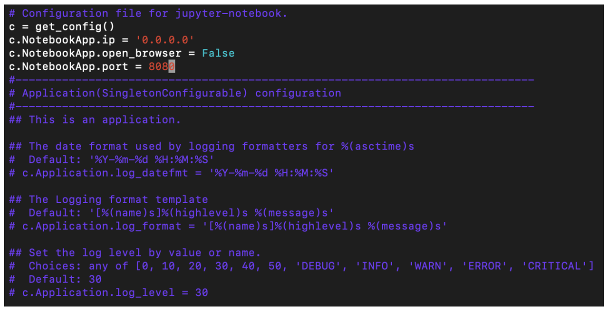
```
jupyter notebook --generate-config
# set password
jupyter notebook password
```

   2. Edit jupyter notebook configuration to let us run the notebook in our browser.
      - Use whatever text editor you want. Example below.

```
vim /home/jupyter/.jupyter/jupyter_notebook_config.py
```

      - Paste the following lines to the top of the file and save.

```
c = get_config()
c.NotebookApp.ip = '0.0.0.0'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 8080
```



```
# Configuration file for jupyter-notebook.
c = get_config()
c.NotebookApp.ip = '0.0.0.0'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 8080
#------------------------------------------------------------------------------
# Application(SingletonConfigurable) configuration
#------------------------------------------------------------------------------
## This is an application.

## The date format used by logging formatters for %(asctime)s
#  Default: '%Y-%m-%d %H:%M:%S'
# c.Application.log_datefmt = '%Y-%m-%d %H:%M:%S'

## The Logging format template
#  Default: '[%(name)s]%(highlevel)s %(message)s'
# c.Application.log_format = '[%(name)s]%(highlevel)s %(message)s'

## Set the log level by value or name.
#  Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
#  Default: 30
# c.Application.log_level = 30
```

   3. Run jupyter notebook.

   - Below command.

     ```
     jupyter notebook --no-browser --port=8080
     ```

   4. Navigate to the jupyter notebook in your browser of choice.
      - The address you are going to navigate to will be the following, replacing `external_ip_address` with yours from step 2. e.g. http://external_ip_address:8080/notebooks

7. [Option two, Sabrina preferred] Jupyter lab
   1. Create the jupyter lab configuration and password.
      - See below.

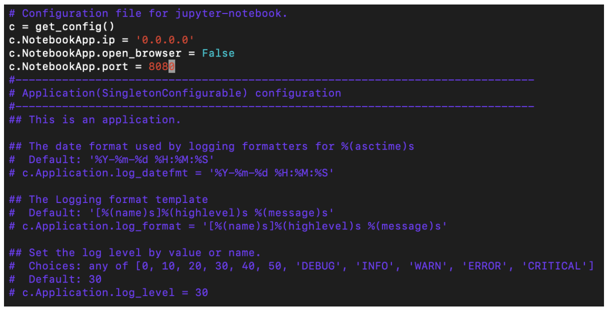
        ```
        jupyter-lab --generate-config
        # set password
        jupyter-lab password
        ```

   2. Edit jupyter lab configuration to let us run the notebook in our browser.
      - Use whatever text editor you want. Example below.

        ```
        vim /home/jupyter/.jupyter/jupyter_lab_config.py
        ```

      - Paste the following lines to the top of the file and save.

        ```
        c = get_config()
        c.NotebookApp.ip = '0.0.0.0'
        c.NotebookApp.open_browser = False
        c.NotebookApp.port = 8080
        ```



```
# Configuration file for jupyter-notebook.
c = get_config()
c.NotebookApp.ip = '0.0.0.0'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 8080
#------------------------------------------------------------------------------
# Application(SingletonConfigurable) configuration
#------------------------------------------------------------------------------
## This is an application.

## The date format used by logging formatters for %(asctime)s
#  Default: '%Y-%m-%d %H:%M:%S'
# c.Application.log_datefmt = '%Y-%m-%d %H:%M:%S'

## The Logging format template
#  Default: '[%(name)s]%(highlevel)s %(message)s'
# c.Application.log_format = '[%(name)s]%(highlevel)s %(message)s'

## Set the log level by value or name.
#  Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
#  Default: 30
# c.Application.log_level = 30
```

   3. Run jupyter lab.
      - Below command.

        ```
        jupyter-lab --no-browser --port=8080
        ```

   4. Navigate to jupyter lab in your browser of choice.
      - The address you are going to navigate to will be the following, replacing `external_ip_address` with yours. e.g. http://external_ip_address:8080/

I stopped my VM and now resumed it. What all do I have to do to get jupyter up and running again?

1. SSH into VM from local terminal
2. Mount persistent disk to VM
3. Run Terra docker of choice
4. Run jupyter notebook or jupyter lab

   1. e.g. `jupyter notebook --no-browser --port=8080` or `jupyter-lab --no-browser --port=8080`

I needed to change my VM. What all do I have to do to get jupyter up and running again?

1. Set external IP to static or select a static IP address you have already created in the GCP UI and note it down.
2. SSH into VM from local terminal
3. Mount persistent disk to VM
4. Allow docker user to read/write to PD
5. Run Terra docker of choice
6. Run jupyter notebook or jupyter lab
   1. e.g. `jupyter notebook --no-browser --port=8080` or `jupyter-lab --no-browser --port=8080`

Supplementary information

Conda environment and kernels

By default, conda environments are placed in `/opt/conda/envs`. As I mentioned earlier, only files in `/home/jupyter` will be saved to the persistent disk, so by default the created conda environments **would be lost** if you created a new VM.

To keep your conda environments, edit the conda configuration to save the environments to a location that is on the persistent disk.

```
conda config --append envs_dirs /home/jupyter/envs
```

We can also use different conda environments within a Jupyter notebook using kernels. S/O Erica Pimenta for providing some insight into this. The main idea is that if you want to use a conda environment with a jupyter notebook, you need to have `ipykernel` installed in the environment. If you want the R kernel, you would need to have `r-irkernel` installed in the environment. For example, the below command would create an environment and associated python kernel for the `scanpy_env`. You could then use the `scanpy_env` python kernel and associated packages in a jupyter notebook.

```
conda create --name scanpy_env scanpy ipykernel
```

Keep process on VM running when you shut computer/ lose wifi/ close terminal

To keep a process in the VM (e.g. a notebook session) running when you shut your computer/lose wifi connection/close terminal/etc, you need to use the `screen` function.

```
# SSH into the VM if not already
gcloud compute ssh --zone "us-central1-a" "{instance-name}" --project "{project-id}"

# start screen
screen

# start process that you want to keep running regardless of connection
# example, a jupyter notebook
sudo docker run -e R_LIBS='/home/jupyter/packages' --rm -it -u jupyter -p 8080:8080 -v
/mnt/disks/{folder-name}:/home/jupyter --entrypoint /bin/bash {terra-docker-image-path}

jupyter-lab --no-browser --port=8080

# disconnect from the screen and you should still be able to access notebook in browser
press CTRL + A
press CTRL + D

## other useful functions with screens
```

```
# list screens
screen -ls
# connect back to a screen
screen -r {screen-name}
# end all detatched screens
screen -ls | grep Detached | cut -d. -f1 | awk '{print $1}' | xargs kill
```

Persistent disk snapshot schedule

Google cloud has a feature to regularly and automatically back up your persistent disks. You can create the snapshot schedule when you create the disk, or afterwards.

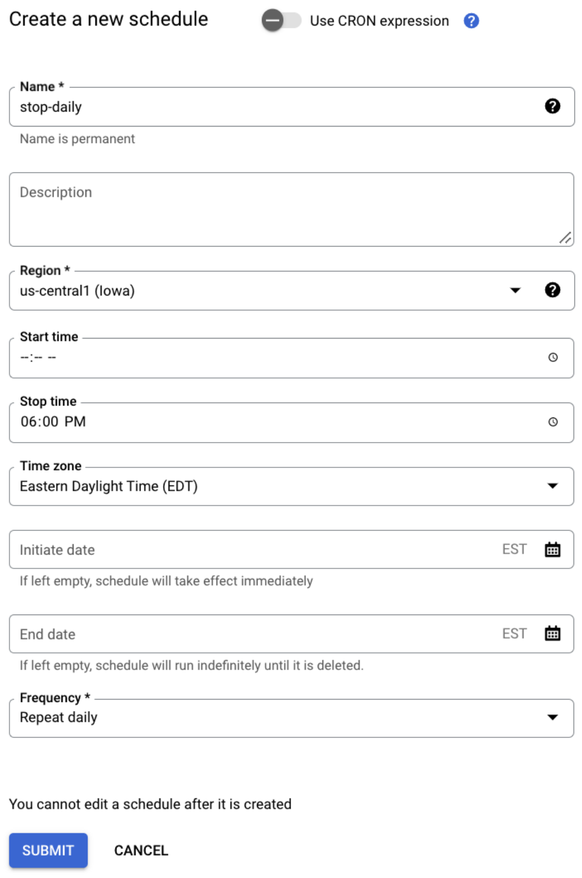
At the moment, I don't think we have any guidelines here. I think it's best practice to select to autodelete snapshots after a certain amount of time to minimize storage costs. Also, perhaps a weekly snapshot is a good combination of frequency and storage and/or copy costs?

Instance schedule

Terra had a nice feature of auto-pausing your VM when you weren't using it for >30 minutes. This feature allows us to reduce costs of VMs we forgot to shut off.

This auto-pausing feature isn't the default GCP VM behavior, and I haven't yet found Terra's documentation on how they do this.

However, google cloud does have what's called "Instance schedules". Instance schedules allow you to program a start and/or stop time to a VM. Here, I am creating a schedule to stop my instance at 6PM daily in case I forget to shut it off myself.


Create a new schedule

Use CRON expression

Name * stop-daily

Name is permanent

Description

Region * us-central1 (Iowa)

Start time --:-- --

Stop time 06:00 PM

Time zone Eastern Daylight Time (EDT)

Initiate date EST

If left empty, schedule will take effect immediately

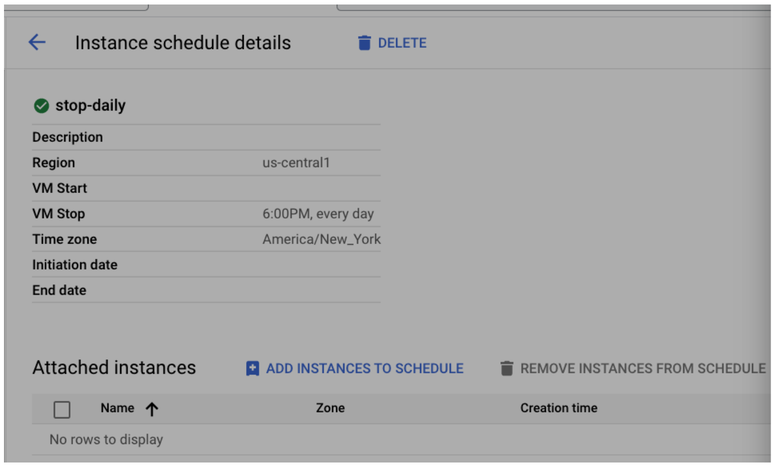
End date EST

If left empty, schedule will run indefinitely until it is deleted.

Frequency * Repeat daily

You cannot edit a schedule after it is created

After creating the schedule, you can attach your VM to it.

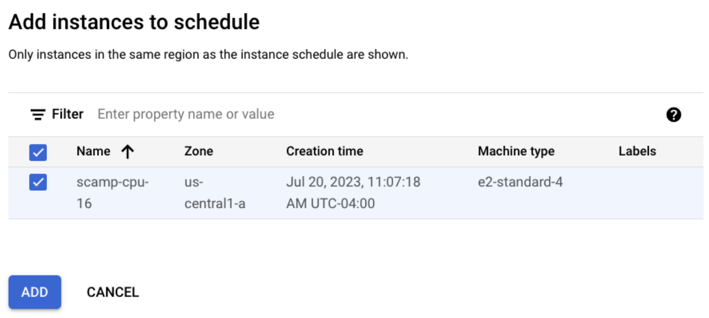

Instance schedule details
DELETE

stop-daily

Description	
Region	us-central1
VM Start	
VM Stop	6:00PM, every day
Time zone	America/New_York
Initiation date	
End date	

Attached instances
ADD INSTANCES TO SCHEDULE
REMOVE INSTANCES FROM SCHEDULE

	Name ↑	Zone	Creation time
No rows to display			


Add instances to schedule

Only instances in the same region as the instance schedule are shown.

Filter Enter property name or value

✓	Name ↑	Zone	Creation time	Machine type	Labels
✓	scamp-cpu-16	us-central1-a	Jul 20, 2023, 11:07:18 AM UTC-04:00	e2-standard-4	

ADD CANCEL

If you are getting a permissions error, you may need to edit the IAM permissions. Talk to Sabrina

How the boot disk image used in this tutorial was created

Steps to how I created the boot disk image `terra-docker-image-100-boot-20230720`

1. Install docker.
   - The Terra notebook environments are docker images. Therefore, in order to utilize these environments in our VM instances, we first have to install docker. I'm following this tutorial which assumes a Debian Linux distribution, which is what GCP uses.

   ```
   sudo apt update
   sudo apt install --yes apt-transport-https ca-certificates curl gnupg2 software-properties-common
   curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
   sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release
   -cs) stable"
   sudo apt update
   sudo apt install --yes docker-ce
   ```

2. Pull the following Terra dockers
   1. R/Bioconductor: us.gcr.io/broad-dsp-gcr-public/terra-jupyter-bioconductor:2.1.11
   2. Python: us.gcr.io/broad-dsp-gcr-public/terra-jupyter-python:1.0.15
   3. Default: us.gcr.io/broad-dsp-gcr-public/terra-jupyter-gatk:2.2.14

Erica has several workspaces. I think she could have one PD per workspace, and mount just mount the workspace PD she is interested in using that time/day/whatever. If she wanted to do things on each PD at once, she could create two VMs, ssh into both, mount each PD, and all should be good.

Sudo access on docker

- To be able to use the `sudo` command, you have to enter the docker as the root user. Generally wouldn't recommend accessing the docker as the root user because of file/folder permissions weirdness later.

  ```
  sudo docker run -e R_LIBS='/home/jupyter/packages' --rm -it -u root -p 8080:8080 -v
  /mnt/disks/{folder-name}:/home/jupyter --entrypoint /bin/bash {terra-docker-image-path}

  #example
  sudo docker run -e R_LIBS='/home/jupyter/packages' --rm -it -u root -p 8080:8080 -v /mnt/disks/scamp-
  singlecell:/home/jupyter --entrypoint /bin/bash us.gcr.io/broad-dsp-gcr-public/terra-jupyter-
  bioconductor:2.1.11
  ```

Known differences

- cant access PROJECT, WORKSPACE, etc environment variables