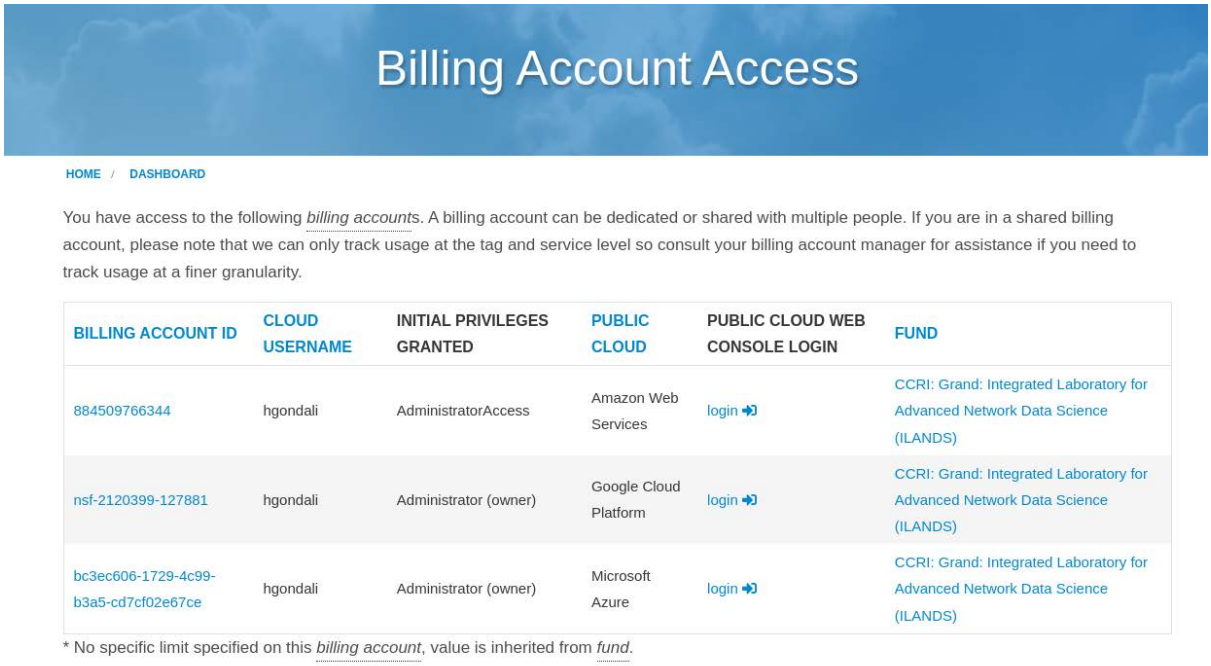
Geoping Project: Tools and Technologies

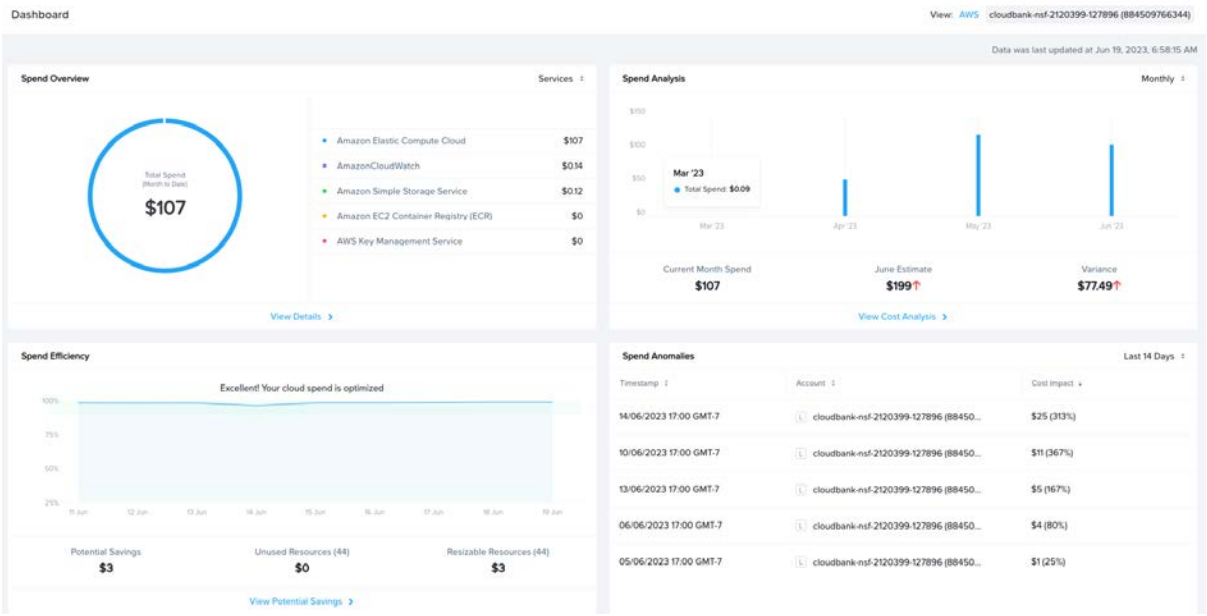
Harsh Gondaliya
May 2023

Cloudbank and Nutanix Beam
(https://www.cloudbank.org/)

- Cloudbank portal is serving as an identity provider and we are able to gain access to the three cloud platforms (AWS, GCP, Azure) using it.
- Login to Cloudbank by clicking on “LOG IN WITH CILOGON”. If asked, select UC San Diego as the institute name and then you will be redirected to the UC San Diego SSO page (one which we generally use to login to all UCSD portals). Enter your UCSD SSO credentials to log in.
- On the Cloudbank homepage, click on the “Dashboard” drop-down menu and then click on “Access Cloudbank Billing Accounts”. This will lead you to a page similar to the screenshot shared below where you will have links to log in to three Cloud Platforms (Amazon Web Services, Google Cloud Platform, and Azure). You can click on the link and it will direct you to the respective cloud platform’s dashboard.



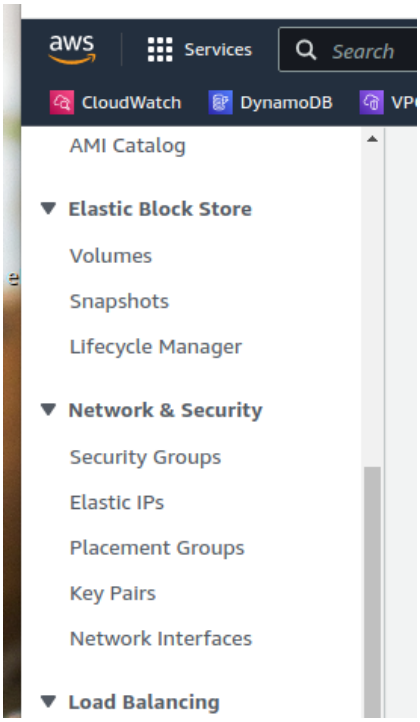
- Cloud resources and services used after logging in are directly charged to our grant.
- On the Cloudbank homepage, click on the “Dashboard” drop-down menu and then click on “Monitor & Optimize Your Usage”. This will direct you to a Nutanix Beam portal where you can see overall costs incurred while using different cloud services and resources across three cloud platforms (AWS, Azure, GCP).
  Note: It takes 2 days for the costs incurred on AWS to become visible on Nutanix. For GCP and Azure, costs are immediately reflected.



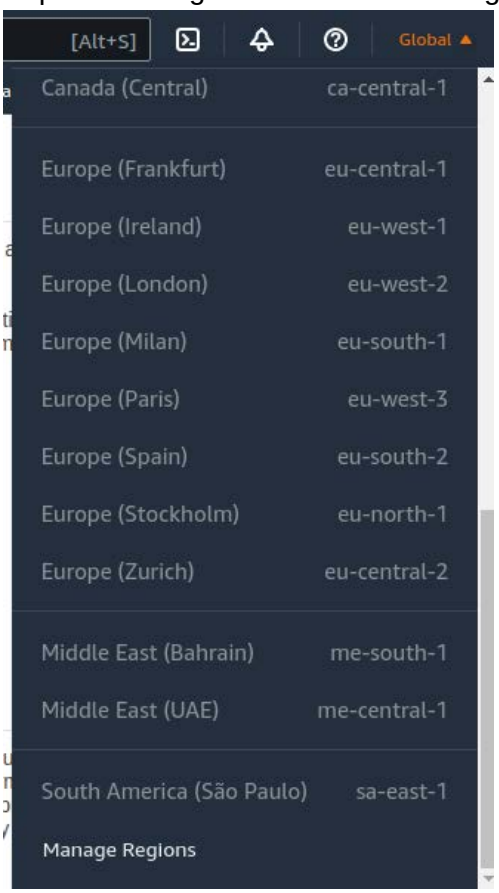
- You can click on a given cloud service in the “Spend Overview” section to get a detailed view of the costs incurred. On all the pages, you can change the view granularity from “monthly” to “daily” to get better insight into the costs incurred.
- To grant access to a new user on the Cloudbank portal or perform any such administrative tasks, the PI or Co-PI needs to refer to this user guide https://www.cloudbank.org/training/managing-cloud-funds-and-billing-accounts and do the needful. We can also contact Cloudbank Support (help@cloudbank.org) to help perform the same operations.
- **Cloudbank Support:**
  help@cloudbank.org
  ssmallen@sdsc.edu (Shava Smallen: Please contact her only if it's urgent or you have not received a response from the above contact in 3-4 business days)

Amazon Web Services (includes cloud-init user data script):

1. **Setting security group:** In the left pane of the EC2 dashboard, click on “Security Groups”. There is a security group named geoping created in all regions for the purpose of this project.
    a. In all regions, there is no restriction on outbound connections from the EC2 instances.
    b. In all regions except us-east-2 (ohio), only SSH inbound connection (TCP port 22) is allowed.
    c. In the us-east-2 region, inbound connections on SSH port (TCP port 22) and TCP Ports 4505 and 4506 are allowed. The latter two ports are used by the Saltstack master to accept inbound connections from minions.

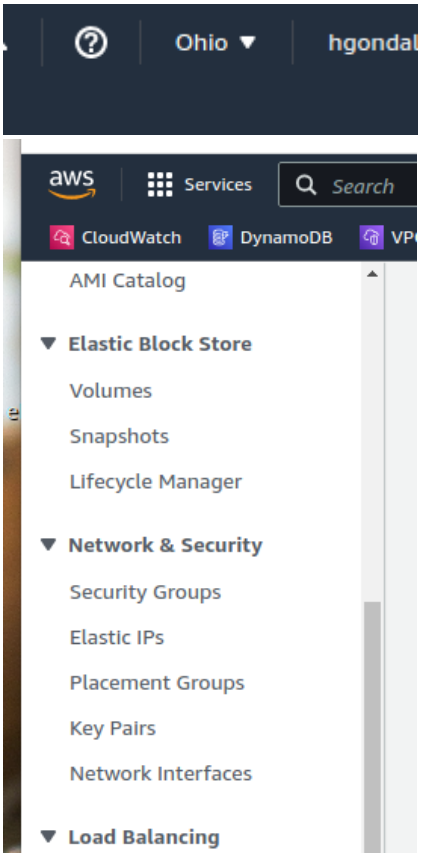


2. **Activating different regions:** By default, certain AWS regions will be disabled. To enable all the AWS regions, click on the “Manage Regions” option in the regions drop-down menu available at the top of the AWS console and you will find the required settings to enable/disable regions.

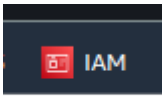


3. **Setting SSH public-private key pair:** In order to have a single public-private key pair that we can use to access EC2 instances across all the regions, follow the steps

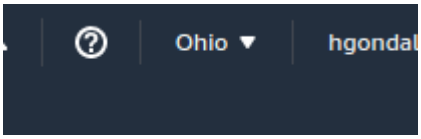
mentioned on page https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
It creates an RSA public-private key pair locally and then uses AWS CLI to import this key pair to all the AWS regions.
The geoping-private/set-public-key.sh script in GitHub repo is doing the same operation for us, but it needs AWS CLI installed with the required config and generation of public-private key pair locally before executing the script (as documented in comments in the script).



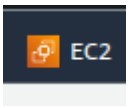
4. **Setting up AWS CLI** - Follow the steps mentioned on https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html to install AWS CLI. Then, it needs to be configured with your credentials to be able to perform the required operations. There are multiple options to configure AWS CLI with your credentials https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html
I picked up the unsafe way of creating a user by going into the “IAM” section of AWS and then using that user’s access_key_id and access_secret_key to configure AWS CLI. This is unsafe as suggested in the above link and thus you can explore other recommended approaches shared on the above link to configure AWS CLI.



5. **Creating an EC2 Instance:**
    a. Select the region in which you want to create an EC2 instance from the top menu in the AWS console

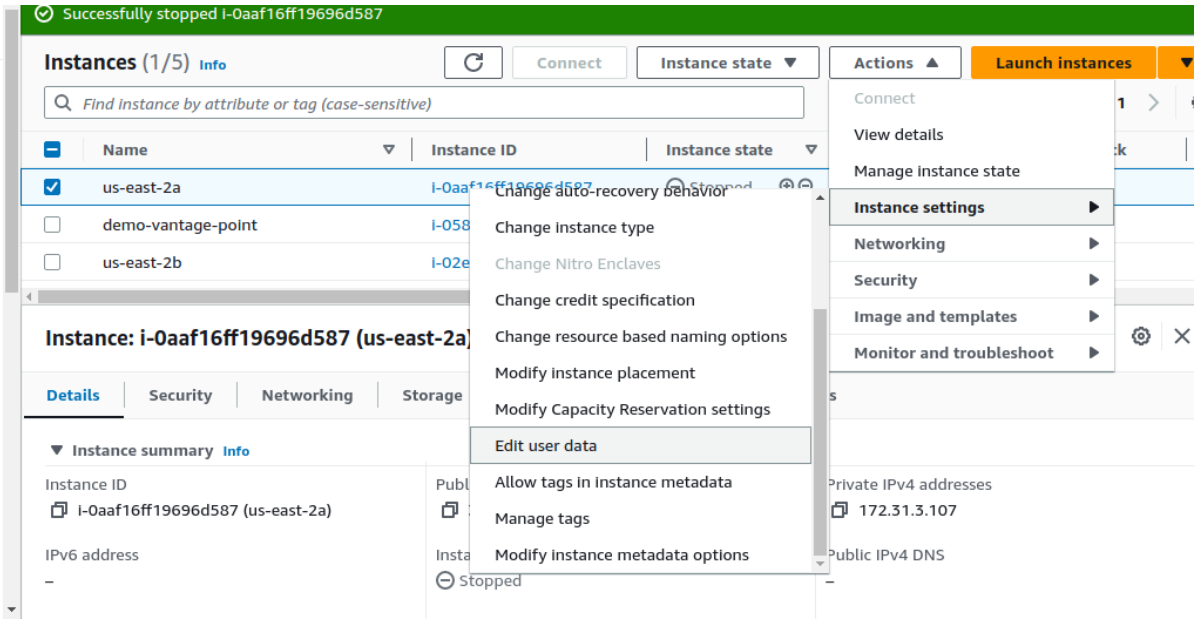


b. Click on the “EC2” button to get the EC2 dashboard and click on the “Instances” option.



c. Click on “Launch Instances”
d. Give a name to your EC2 instance. This is just a name that appears on the EC2 dashboard for your EC2 instance
e. Select the Amazon Machine Image (based on the operating system and apps that you want for your EC2 instance). We are using “Ubuntu Server 22.04 LTS (HVM), SSD Volume Type” image. **It's advisable to use the same OS in other cloud platforms too** so that we have the same environment in all our cloud vantage points and we need not make many changes to our startup.sh and geoping-pipeline.py scripts.
f. Select the instance type (based on the hardware resources you need)
g. Select the public-private key pair that you want to use for this instance. Generally, we are using geoping-allregion.pem for all regions.
h. In “Network Settings” click on “Select and Existing Security Group” and select the “geoping” security group that you have created for your use case. Also, select the “Edit” option in network settings to choose a specific subnet (i.e. availability zone in a region) where you want to place your EC2 instance. If a subnet is not selected then it randomly places your EC2 instance in one of the availability zones within a region.
i. Configure the amount of disk storage you want for an instance. This can be increased in the future.
j. Click on “Advanced Details”, and in the “User Data” section copy and paste the contents of the geoping-private/aws-ec2-cloud-init-user-data.txt file
    i. This file is a MIME-type file that is telling the EC2 instance to execute the user script always on reboot
    ii. The user script is intentionally named “001.txt” so that alphabetically it is the first script that is executed.
    iii. When we edit user data script in the future, cloud-init creates a new file for it and thus the old as well as new script exists. And the working of cloud-init is such that it executes all the user scripts one by one on boot.
    iv. Thus, we are intentionally naming our user data script as 001.txt so that it is the first one to be executed and this script first deletes all the other user scripts present in the directory and then starts its operations.
    v. The script first sets the hostname of the EC2 instance because the same hostname is taken up by Netbird as the name of the peer and by saltstack as the name of the minion.

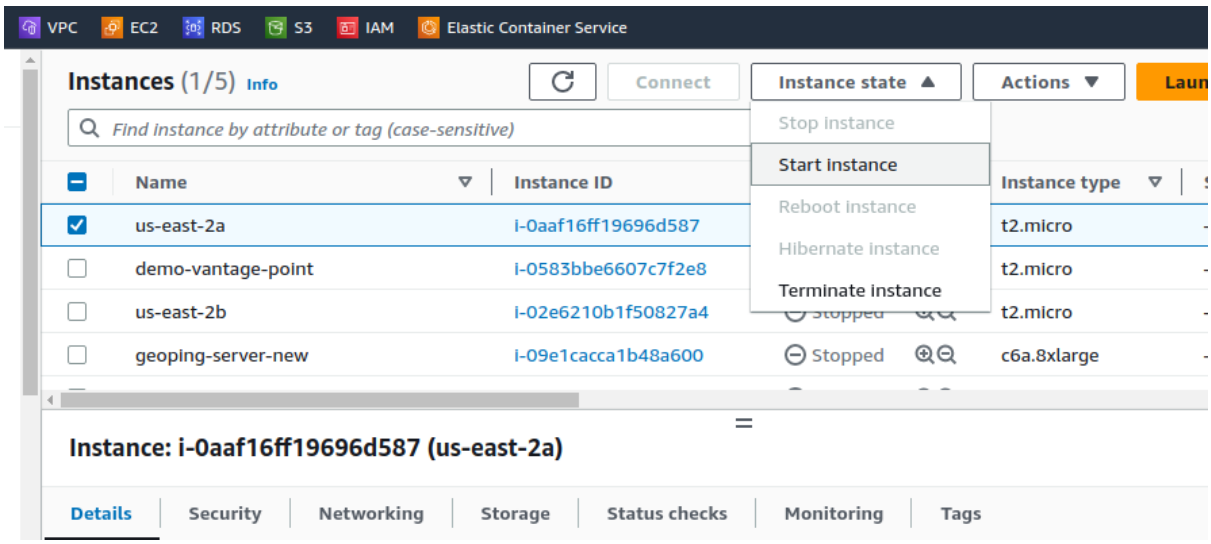
vi. To edit the user-data script for an already created instance, select the instance from the dashboard, click on “Actions”, go to “Instance Settings” and click on “Edit User Data”. Again please be careful of keeping the script name as “001.txt” and keeping the initial lines that delete all the other user data scripts



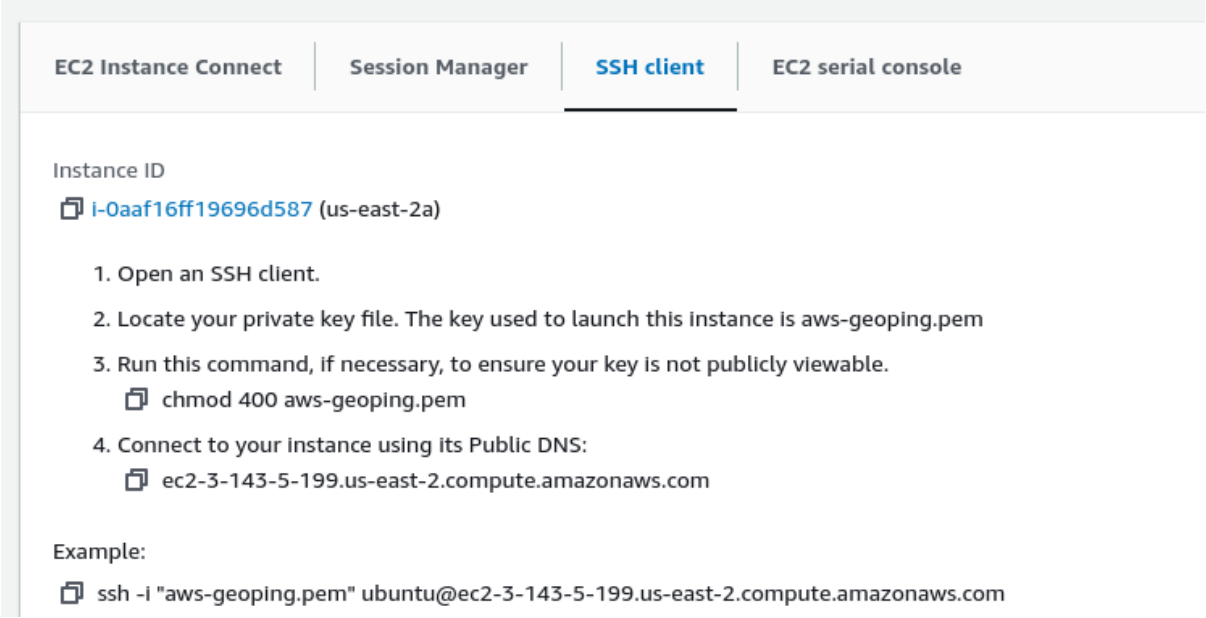
k. Click on “Launch Instance” and the instance would be created and start running.

l. From the EC2 dashboard, you can select an instance and start/stop it as required. You can also terminate an instance (i.e. delete an instance).

m. When you SSH to an EC2 instance, logs of the cloud-init (user data script that runs on boot) is available at `/var/log/cloud-init-output.log` and all the user scripts that are executed on boot are available at `/var/lib/cloud/instance/scripts`. All the scripts are available in the above scripts directory are executed on the boot one after another. Thus, you would only want one required script to be present there.

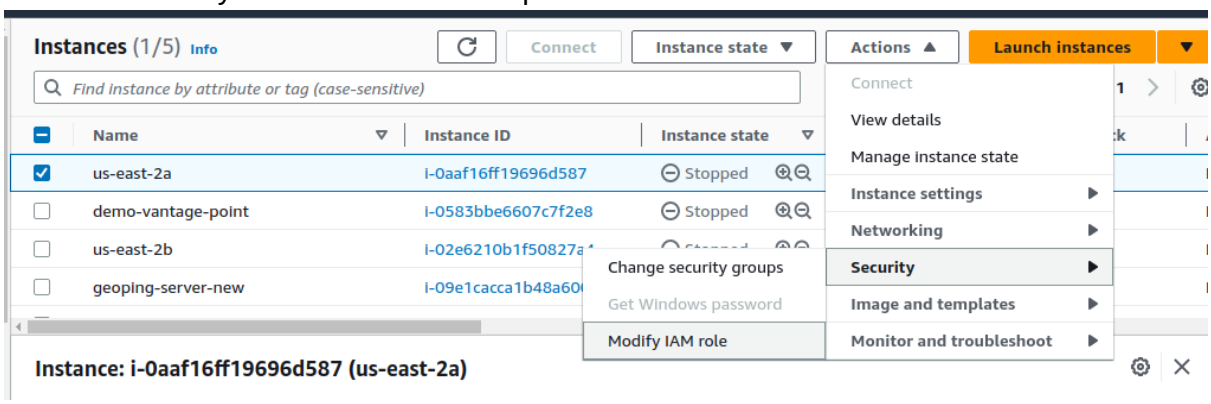


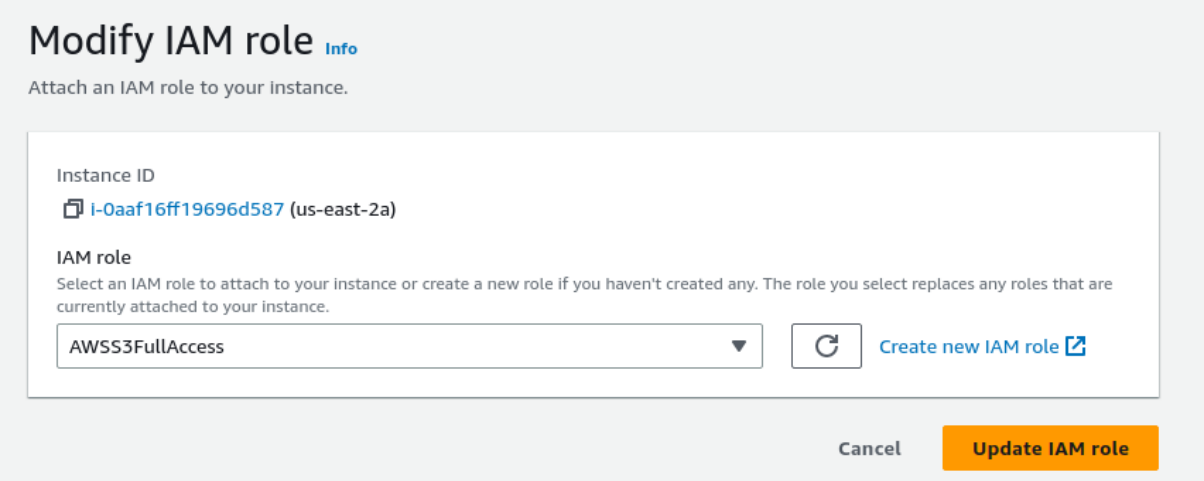
6. **Connecting to an instance:** Once an instance has been started and is running, click on the “Instance ID” shown in the above pic and go to the “Connect” tab. It will have details about how to SSH to a given EC2 instance. The public IP address and the hostname of the EC2 instance keep changing on each reboot and hence you need to manually go to this tab and look for the latest command to ssh to this instance.



You need to change to the directory where your geoping-allregion.pem file is located and then from there execute the above ssh command.
Please note that geoping-allregion.pem (available in geoping-private/ssh-key) will work on all EC2 instances in all regions.
Thus, if you see a different key like aws-geoping.pem in the above pic then that’s just an additional key that is working for that given EC2 instance.

7. **Accessing S3 bucket:** To allow an EC2 instance to perform all operations on S3, you need to assign a role to that instance having the policy “AWSS3FullAccess”. You can select the instance, open the “Actions” dropdown menu, select the “Security” option, and then select “Modify IAM Role”.
If no such role already exists then you need to create such a role by going into IAM. Basically, a role having an “AWSS3FullAccess” policy needs to be created and given to an EC2 instance. Then, using AWS CLI or AWS CLI python client on that EC2 instance you can do read/write operations to the S3 bucket.





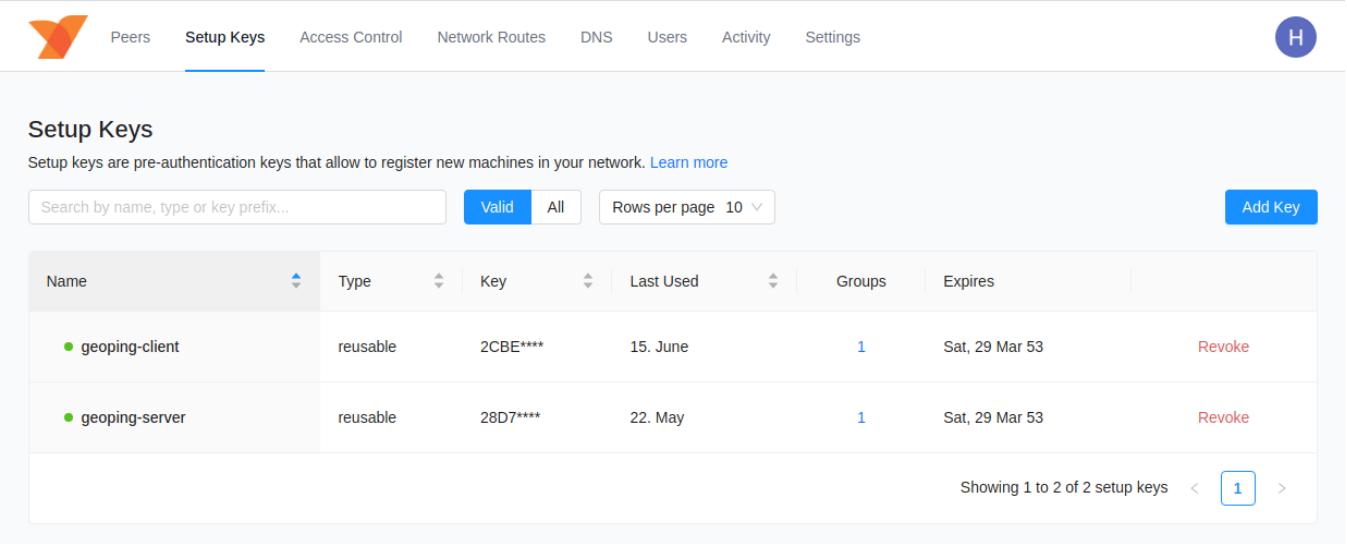
In the us-east-2 region (the same region where our geoping server is there), we have created a bucket named geoloc-pinger.2023-05-05.21-35-00 and resultant measurement result files collected from all 85 vantage points are stored in that bucket. It's advisable to move all data from the geoping server to the S3 bucket to save disk space on the server. **Also, please note that we have allocated 30 GB of disk storage to our server and for that, we are being charged $2.85 per day.**

8. **Sending updates/configuration changes to all the EC2 instances:** If you need to configure any updates or install something on all the EC2 instances then its advisable to follow this workflow:
    a. Push changes to the geoping/startup.sh (**public** version of geoping GitHub repo) that performs necessary installation or configuration. You can learn from the way the existing script is written. Have print statements as necessary for logging.
    b. Next start a single EC2 instance. Each EC2 instance is configured to download the geoping/startup.sh file and execute it. Hence, wait for some seconds after starting the instance and check /var/log/cloud-init.log to see the logs and check whether desired installation and configuration are successful.
    c. Start and second instance to double check whether desired changes are taking place using geoping/startup.sh script.
    d. Once you are confident you can start all EC2 instances one by one and wait for some time. After that time, all the new configurations and installations that you included in geoping/startup.sh would be reflected in all the EC2 instances.

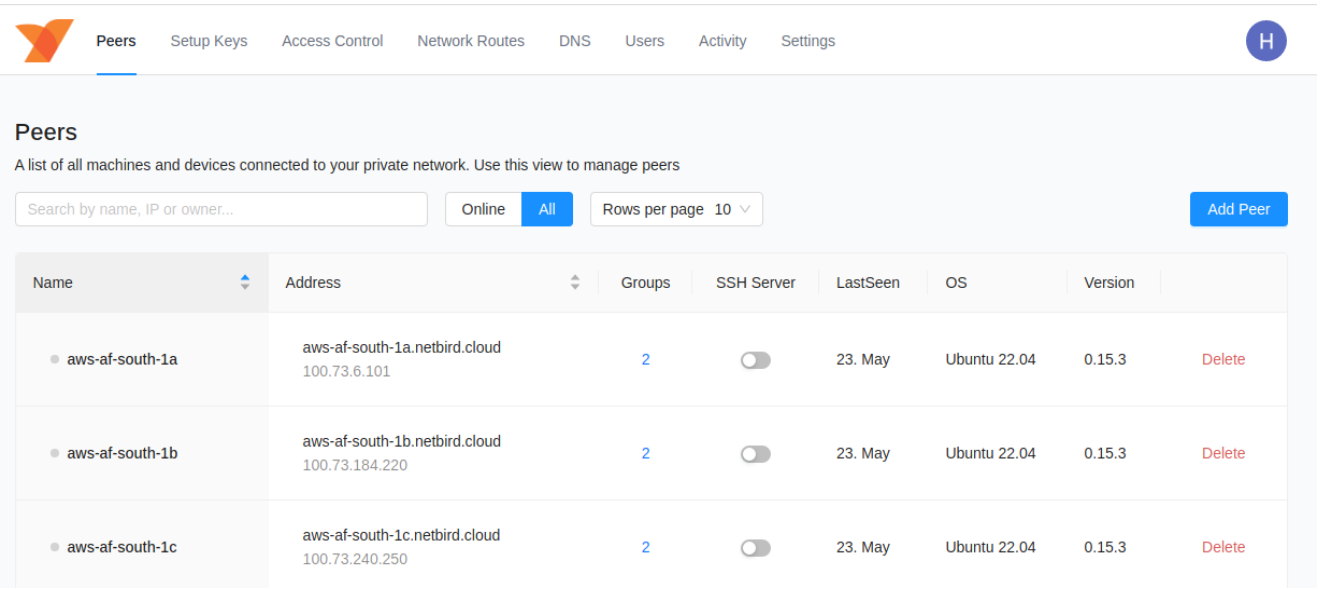
Netbird

- Visit https://app.netbird.io/peers and Sign In using Google. Currently, Shivani's and Harsh's UCSD email are added to the Netbird account that has access to all the AWS vantage points.
  Go to the "Setup Keys" tab and there you will find an option to "Add Key". While adding keys, it will ask you to set the expiration time and also the access group to which a peer who joins using this key must be added.
- Access Groups abstraction is used to manage access control. You can go to the "Access Control" tab and set a policy of which two groups must be able to communicate with one another. If you want members of the same group (for ex, geoping-client) to be able to communicate with one another then you need to set "geoping-client" <—> "geoping-client". Also, please note that once the setup key

expires, the existing nodes added using that key are not affected and you can create a new setup key to add new members to the same existing group.



- You can refer Netbird documentation to learn about its installation steps and steps to add new peers to your network https://docs.netbird.io/



- On the peers tab of Netbird, you will be able to see all the online and offline peers. You will also be able to remove peers from your virtual network.
- Netbird Repo: https://github.com/netbirdio/netbird
- Use Netbird Slack Channel to get any assistance and help: https://join.slack.com/t/netbirdio/shared_invite/zt-vrahf41g-ik1v7fV8du6t0RwxSrJ96A
- Current setup-key for geoping-clients group - 2CBEBAE1-B99A-4800-94D7-FEBFD5BF254A

  Geoping-server: 3E659727-45AD-43F4-B6FA-8A3F6C7B90B1

Generally, community members on Slack actively respond to queries. Based on the last conversation with *Mikhail (Misha) Bragin* (Owner - Netbird) over Slack these were the updates:

1. They have some plans of introducing some pricing for the managed version of Netbird. The self-hosted version (where we run and deploy Netbird code on our servers) will remain free.
2. For our purpose, they can allow us to use the managed version of Netbird for free (when they introduce pricing) if we allow them to put our name on their webpage.
3. As per Misha, as the number of peers in our network increases, there is a chance of a large amount of traffic reaching to each of our peers. At that point, we might face performance problems. Hence, Misha suggested doing a pilot project together where all clients could directly talk to a single server instead of using a relay server that is used currently. In the future if such problems come then need to contact Misha for this and explore about the pilot project.

SaltStack

- You can follow the Salt installation guide to install the salt-master service on the the geoping server instance and salt-minion service on the rest of the geoping vantage points. https://docs.saltproject.io/en/latest/contents.html
- You can follow the configuration guide available at the above link to set different configuration parameters for the salt master and salt minon.
    - Salt master config parameters can be set by either modifying the/etc/salt/master file or by manually creating .conf files in /etc/salt/master.d/ directory.
    - Similarly, Salt minion config parameters can be set by either modifying /etc/salt/minion file or by manually creating .conf files in /etc/salt/minion.d/ directory. Currently we are creating the following .conf files in minion:
        - network.conf contains `ipv6:false`
        - minion.conf contains `id: <hostname of instance>`
        - master.conf contains `master: <ip address of master>`
- geoping-private/salt-config directory contains the necessary salt-master and salt-minion configuration files that are being used currently.
- When a new minion attempts to join a cluster, it sends its key to the master and then the master needs to accept the minion's key to allow it into the cluster.
  `sudo salt-key`
  command will show you list of accepted, rejected and pending keys.
  `sudo salt-key -A` will accept all the pending keys.
  Please note that all saltstack commands need to be executed as sudo.
- The salt master's config is set such that the /srv/salt directory serves as a file server. Thus, any file that you need to send to minions needs to be present in that directory and then `salt://<name of file or path to file within /srv/salt/ directory>` is provided to python API and salt commands to specify the file to be sent to the minions.
- All the files pushed from minions to masters are saved in `/var/cache/salt/master/minions/<minion-name>/files` directory. Generally, the hostname of the minion's machine is picked up as the minion's name.
- .bashrc of the geoping-server is set to restart salt-master service on boot. After every config change on the minion, salt-minion service needs to be restarted using the command
  `sudo systemctl restart salt-minion.service`

- Salt Python API https://docs.saltproject.io/en/latest/ref/clients/index.html is used to write geoping pipeline code as a Python file. Here, we club multiple Linux commands into a single command using ';' or '&&' operator and then publish this command to the minions.
  When ';' is used to merge Linux commands then all the commands are executed one after another although one or more commands in between may fail.
  When "&&" is used to merge Linux commands then commands are executed one after another and execution stops on encountering a failure. Thus, commands executed after failed command are not executed.
- After starting all the required EC2 instances, and appropriately setting the c"IP_ADDRESS_LIST_FILENAME" (file containing a list of IP addresses to whom you want to launch ping experiments), execute the below command:
  `sudo ./geoping-pipeline.py - -regex=aws-us-east-2*`
  The regular expression determines the targeted minions on which the command is to be executed.
- The geoping pipeline output will give limited visibility while debugging. While debugging you will need to have multiple echo statements and trace all sub-commands one by one.
- After the geoping-pipeline.py script is run successfully, all aggregated results would be available in a directory inside /home/ubuntu/aggregate-results
- Please note that if you want to specify a file name in a command that is published to a minion then please specify its absolute path and not relative path. You can notice this in the IP address list file passed to scamper command in geoping-pipeline.py
- If you want to change the master, edit the content of geoping/startup.sh (**public version of geoping NOT geoping-private**) and add following lines at the end.

```
# change conf file at minion to point to new master
sudo echo -e "master: <ip address of new master>" | sudo tee
/etc/salt/minion.d/master.conf

# delete the old master public key at minion
sudo rm /etc/salt/pki/minion/minion_master.pub

# restart the minion service
sudo systemctl restart salt-minion.service
```

  Now start EC2 instances one by one and notice that the updated startup.sh file is downloaded and the new master now gets key requests from different minions to join its cluster.

Scamper

- https://www.caida.org/catalog/software/scamper/ We are using the scamper tool developed by Mathew to launch ping experiments to all the router IP addresses. Specifically, sc_pinger util within scamper is used for our purpose.
- Following are the steps to lunch ping experiments for "ipaddr.txt" file using scamper and sc_pinger
  - Start scamper on your desired port (ex, 5001) in the background as a daemon
    `sudo scamper -P 5001 -p 1500 -D`

    - Launch sc_pinger while providing ipaddr.txt file as input and specifying the port on which scamper daemon is running
      `sudo sc_pinger -a ip-list.txt -o ping-output.warts -p 5001 >ping.log`
    - Compress the generated warts file
      `bzip2 -9 -f <warts file name>`
- Scamper can be installed using Ubuntu PPA https://launchpad.net/~matthewluckie/+archive/ubuntu/scamper or via binary installation as mentioned on https://www.caida.org/catalog/software/scamper/
  We are using the Ubuntu PPA version and the startup.sh script is configured such that all the vantage points on boot, will check whether a new scamper and scamper-utils version is available. If its available then the scamper and scamper-utils version will be updated.

Addresses and determined lat/long of different AWS Vantage Points
https://docs.google.com/spreadsheets/d/1C2bwEBB7zQaqzdkdVVzG-Yn8xzfyp8BoqTYkJDqUxHc/edit?usp=sharing

Running the hoiho Algorithm

- After we have obtained all the warts files, we need to do the following steps before running the hoiho algorithm.

1. Create aws-vploc.txt file
   https://github.com/CAIDA/geoping-private/blob/main/mathew-files/aws-vploc.txt
   This file contains the vantage point names and its lat/lon coordinates.
2. Download the routers file
   https://www.caida.org/~mjl/tmp/harsh-20230518-routers.txt.bz2
3. Merge all the warts files (from measured data) into a single *minrtt.txt file containing all the RTT tuples

   ```
   perl harsh-build-rtt.pl harsh-20230518-routers.txt.bz2 /path/to/aws*.geoloc-pinger.*.warts.bz2 >harsh-20230518-minrtt.txt
   ```

4. Edit the geoping-private/prune-rtt.py file parameters to point to the *.txt file you want to prune and the different config that you want to use to split the file and distribute workload among cores. Then, execute the script and it will generate a *minrtt-pruned.txt file.
5. Run hoiho using the downloaded routers file and the pruned-minrtt file (all the files needed for this command are in geoping-private/mathew-files directory)

```
sc_hoiho -O json -O learngeo -d best-regex -g geocodes.txt -g aws-vploc.txt -R harsh-20230518-minrtt-pruned.txt public_suffix_list.dat harsh-20230518-routers.txt >harsh-20230518-geo.json
```

   To run hoiho for a given domain

```
sc_hoiho -D acsdata.net.nz -O learngeo -d routers -g geocodes.txt -g aws-vploc.txt -R harsh-20230518-minrtt.txt public_suffix_list.dat harsh-20230518-routers.txt
```

6. geoping-private/results directory contains hoiho results for AWS vp data before and after pruning. There was a difference in results for three domains. The diff files in that directory shows differences in results before and after pruning for those domains.

Shivani's Next Steps:

- **Milestone 1:** We want to create one VM/instance in each of the Google Cloud Platform and Microsoft Azure geographical regions and add it to our **existing** salt cluster.
    - Please note that the geoping-server in AWS us-east-2 region is going to act as the overall geoping project server. Thus, this server will execute the geoping-pipeline.py script and trigger the launching of experiments on the GCP or Azure nodes and the results will come back to the geoping server in AWS.
    - All the vantage points regardless of their cloud platform or region are brought to the same virtual network using Netbird. Once they are on the same network, we have a SINGLE salt cluster having one salt-master and all other nodes acting as salt-minions.
    - Each node (regardless of cloud platform) on boot will download and execute geoping/startup.sh script. We need to be aware of this while pushing changes to this file on GitHub.
- **A key goal here is to maintain uniformity.** If we are able to do the environment setup for a VM in GCP and Azure in exactly the same way as it is in a VM in AWS then all the current scripts (startup.sh and geoping-pipeline.py) must work without any major changes.
    - Try using the same OS as the one used in AWS EC2 instances. This will ensure that Linux commands that are used in startup.sh script will work without any new problems in other cloud platforms.
    - Follow the same naming convention to give hostnames. For example, gcp-<name of region, etc> or mz-<name of region>
      So that its easy while executing geoping-pipeline code
      For example, sudo ./geoping-pipeline.py - -regex=gcp* must be able to launch an experiment on all the GCP vantage points.
    - Try exploring if the same aws-ec2-cloud-init-user-data.txt MIME type file can be used to make a VM in GCP or Azure to execute a set of instructions every time it boots. The cloud-init framework used in above MIME type file is cross-platform framework and hence we expect it to work across platforms.
    - <not too important> Can we use the same ssh public-private key pair to access VMs in other cloud platforms? Or have a single public-private key pair per cloud platform?
    - Please set the security policy as it is in “us-east-1” region of AWS.

Here onwards, you can discuss with mjl and tweak milestones.

Milestone 2: Come up with lat/long for each vantage point in GCP and Azure. This might need tracing addresses of different data centers as it was done for AWS.

Milestone 3: Use the current prune-rtt-file.py to prune the RTT tuples file and generate hoiho results. Check results before and after pruning.
Results obtained using data from each cloud platform and by merging all cloud platforms data could be compared.

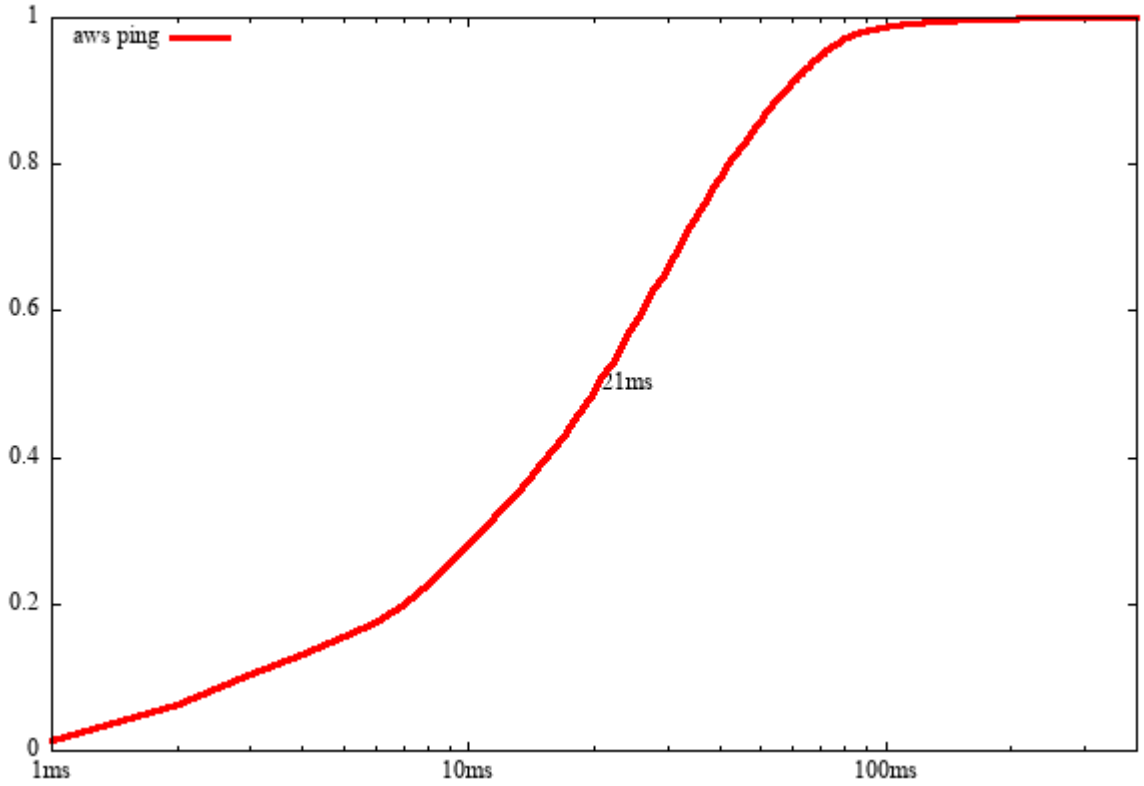
AWS Results:

The plot for minimum rtt was generated for the pruned min-rtt file by using the dump-minrtt-dist-diff.pl and plot-minrtt-dist-diff.pl file.

The difference before and after pruning is dumped in the results folder. The difference was majorly in domains akamai, aorta.

Commands used:

- perl dump-minrtt-dist-diff.pl --mode rtt-union aws-minrtt-prunned.txt -> This command generates the cdf for 400 values which can be copied onto a text file
- perl plot-minrtt-dist-diff.pl --mode rtt-union --nolabels aws.png aws-cdf.txt -> This command generates the plot of minimum



Setting up Terraform for GCP Instances:

Link: https://github.com/CAIDA/geoping-private/tree/main/gcp_vm-in-few-steps

Step1: Install Terraform

Step2: Configure google account to get the credentials.json file which is used by terraform script.

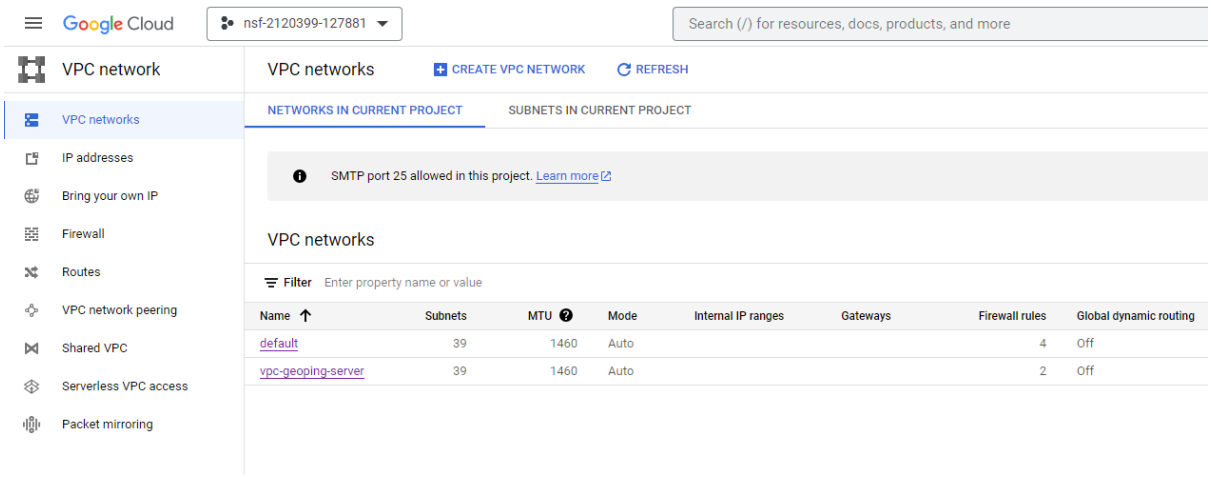
Step3: The terraform commands and it's functionality is provided in the usage.md file in the repository above.

Step4: Ensure the gcp-geoping-allregions ssh key file is in the same directory as terraform.

Step5: The VM instance creation script is mainly contained in the create-instance.tf. To edit or view the value of variables, check terraform.tfvars.

GCP Instances Description:

VPC network in GCP works across all regions. Created a VPC network called vpc-geoping-server for this project.



Firewall rules are added at the VPC network level to allow all outbound connections and allow only TCP 22 (SSH) inbound connections.

☐	geoping-server-egress	Egress	Apply to all	IP ranges: 0.(	all	Allow	1000	vpc-geoping-server	Off
☐	vpc-allow-ssh	Ingress	Apply to all	IP ranges: 0.(	tcp:22	Allow	65534	vpc-geoping-server	Off

Subnets are automatically created based on regions and IP addresses to these instances are automatically assigned on creation.
To SSH in GCP instances, ensure you have the private key file and use the command : ssh -i gcp-geoping-allregions ***shiva***@<IP Addr>
All the VM instances are of machine type **e2-micro,** Ubuntu, 22.04 LTS Pro Server, amd64.
The cloud init start-up script that downloads the startup.sh script from geoprivate repository for setting up scamper and netbird is part of the custom metadata.

Custom metadata

Key	Value
startup-script	Content-Type: multipart/mixed; boundary="//"

All these configurations are taken care of by the terraform script.
There are 109 instances across all regions and availability zones of GCP.
Once all the instances are up and running, connect to the master node on AWS and accept all the Salt-keys requests sent by these GCP nodes. They would be of the form gcp-*

Garnering Latitude and Longitude of GCP:

The latitude and longitude of the GCP data centers from collected from various internet resources and documented at
https://docs.google.com/spreadsheets/d/1VZ2ZC2ikU3NihOTMDeCIGqIc6EAcFomiTSqBf6nrO9Y/edit#gid=0
This document contains google map pinpoints of all the locations of data centres and the final lat and long coordinates for running hoiho algorithm. All the collected data is aggregated in gcp.vploc.txt file.

Launching experiments on all GCP nodes:

To launch experiments on all the GCP nodes using the itdk file, run the command from geoprivate folder:
sudo ./geoping-pipeline.py –regex=”gcp-*”

Pushing results on S3:

- Create an S3 bucket on the same region as that of the master AWS node
- Fetch the AWS secret access key and access key ID after creating an IAM role for AWS CLI
- Configure the AWS CLI on the master node by using command aws configure and provide the keys generated in the previous step.
- Push all the ping results to s3 using the following command:
  sudo aws s3 sync <local folder name> s3://<bucket name>

- To aggregate all the GCP ping results into one text file, use the following command

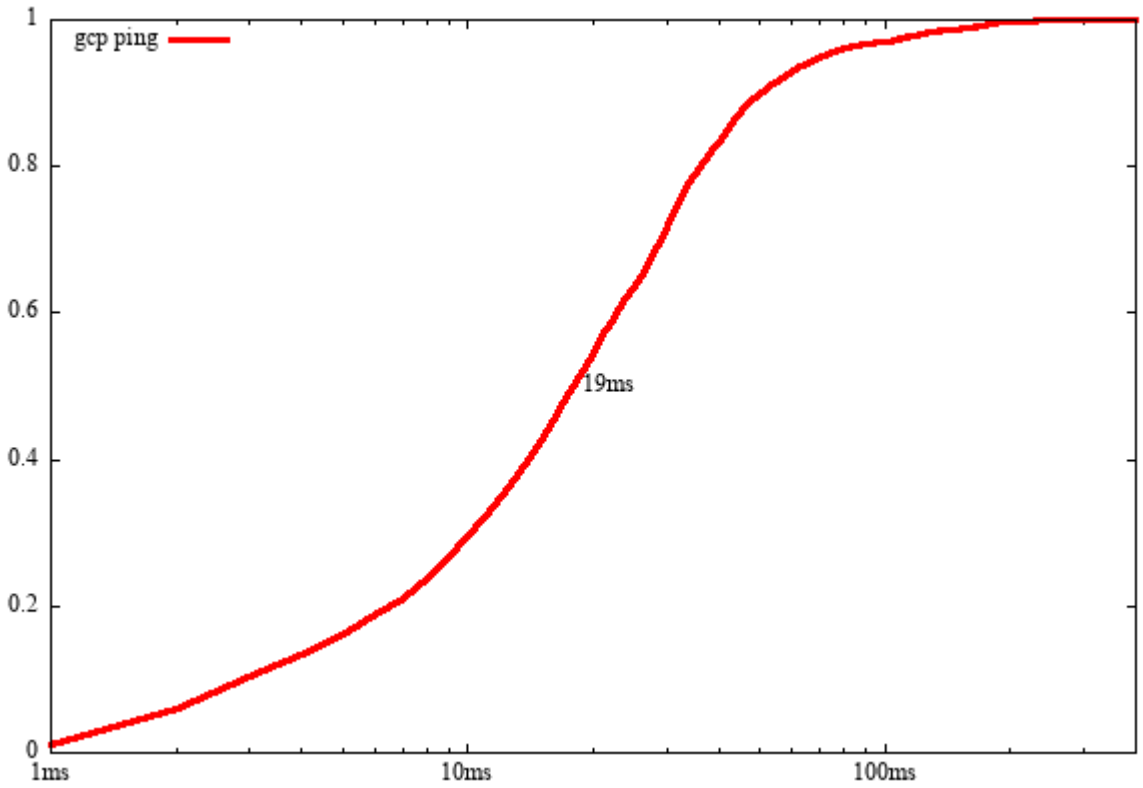
perl gcp-build-rtt.pl harsh-20230518-routers.txt.bz2 /home/ubuntu/aggregate-results/gcp/gcp*.geoloc-pinger.*.warts.bz2 >gcp-minrtt.txt

- To prune this text file, update the name of text file in prune-rtt-file.py and the gcp-vploc.txt file and run the following command:
  python3 prune-rtt-file.py
- Once the pruned min-rtt file is generated, we can run hoiho on the txt file using the following command:
  sc_hoiho -O json -O learngeo -d best-regex -g geocodes.txt -g gcp-vploc.txt -R gcp-minrtt-prunned.txt public_suffix_list.dat harsh-20230518-routers.txt >gcp-geo.json
- More information about hoiho commands and parameters can be found at https://www.caida.org/catalog/software/scamper/man/sc_hoiho.1.pdf

GCP results:
The plot for minimum rtt was generated for the pruned min-rtt file by using the dump-minrtt-dist-diff.pl and plot-minrtt-dist-diff.pl file.
The difference before and after pruning is dumped in results folder in file called gcp.diff. The difference was majorly in domains akamai, aorta, akis.net apa.net, as54104.net and blokowe.pl etc

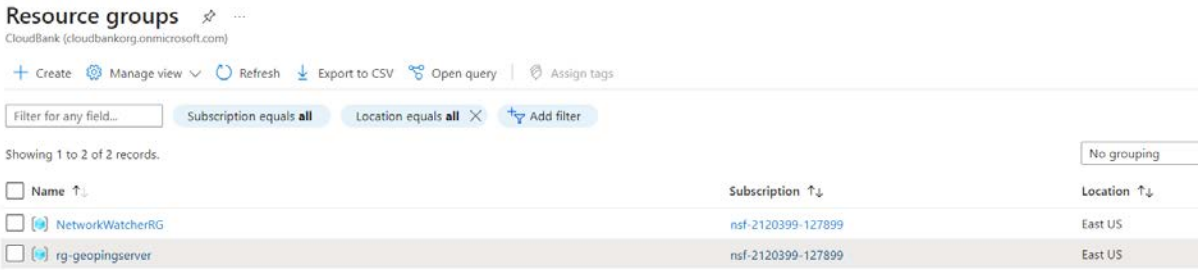
Commands used:

- perl dump-minrtt-dist-diff.pl --mode rtt-union gcp-minrtt-prunned.txt -> This command generates the cdf for 400 values which can be copied onto a text file
- perl plot-minrtt-dist-diff.pl --mode rtt-union --nolabels gcp.png gcp-cdf.txt -> This command generates the plot of minimum rtt.

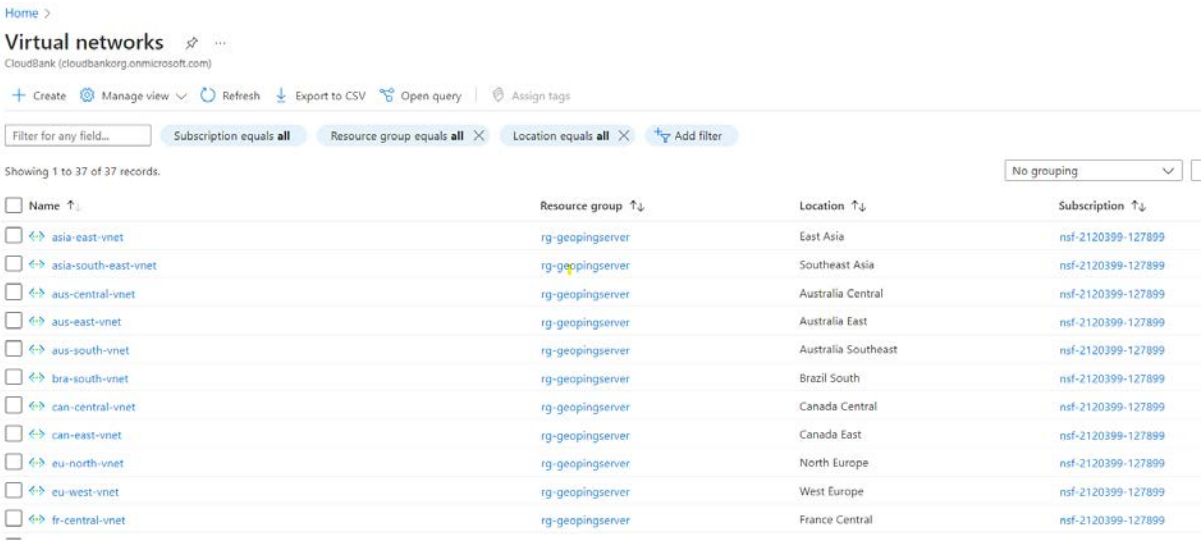


Setting up Azure Machines:

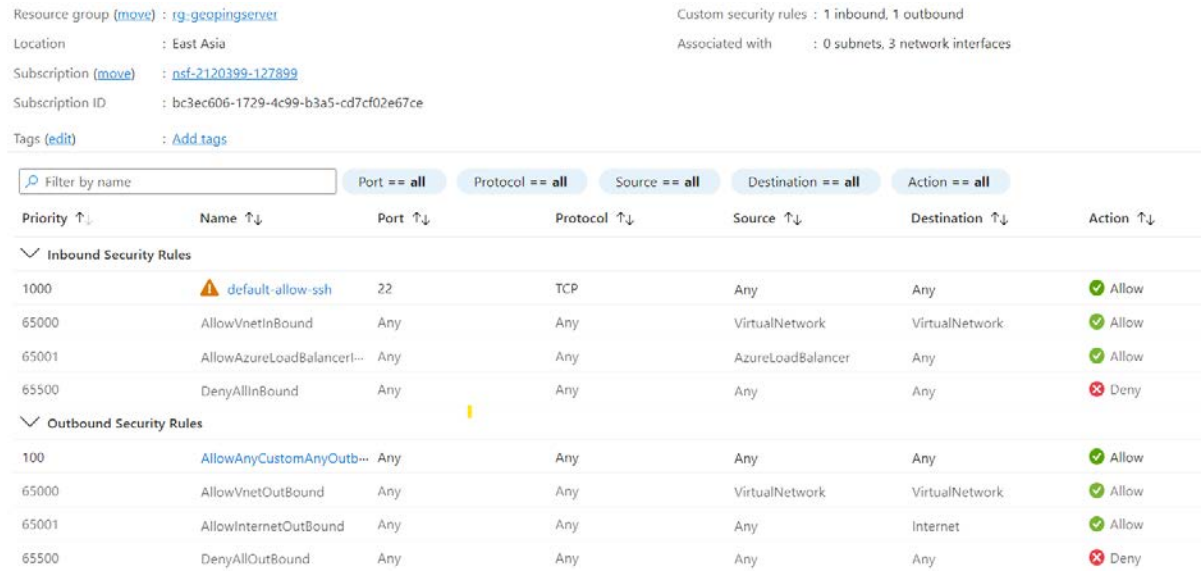
Step1: Created a resource group for this project called rg-geopingserver



Step2: Created virtual networks for each region



Step3: Created a network security group for each region which allows all outbound connections and allows only ssh tcp 22 incoming connections.



Step4: All the instances created are of machine type Standard B1s and operating system is linux. The IP addresses were automatically created and assigned to these instances.
Step5: There are 88 instances across all regions and availability zones of Azure.
Step6: Once all the instances are up and running, connect to the master node on AWS and accept all the Salt-keys requests sent by these Azure nodes. They would be of the form mz-*

Garnering Latitude and Longitude of Azure:

The latitude and longitude of the Azuredata centers from collected from various internet resources and documented at
https://docs.google.com/spreadsheets/d/1hZcBiy5maGi0Q_K2IHuQWZ0gCr4OuWBM75pej1j_SJs/edit#gid=0

This document contains google map pinpoints of all the locations of data centres and the final lat and long coordinates for running hoiho algorithm. All the collected data is aggregated in azure.vploc.txt file.

Launching experiments on all Azure nodes:

To launch experiments on all the GCP nodes using the itdk file, run the command from geo private folder:
sudo ./geoping-pipeline.py –regex=”mz-*”

Pushing results on S3:

- Create an S3 bucket on the same region as that of the master AWS node
- Fetch the AWS secret access key and access key ID after creating an IAM role for AWS CLI
- Configure the AWS CLI on the master node by using command aws configure and provide the keys generated in the previous step.
- Push all the ping results to s3 using the following command:
  sudo aws s3 sync <local folder name> s3://<bucket name>

- To aggregate all the GCP ping results into one text file, use the following command

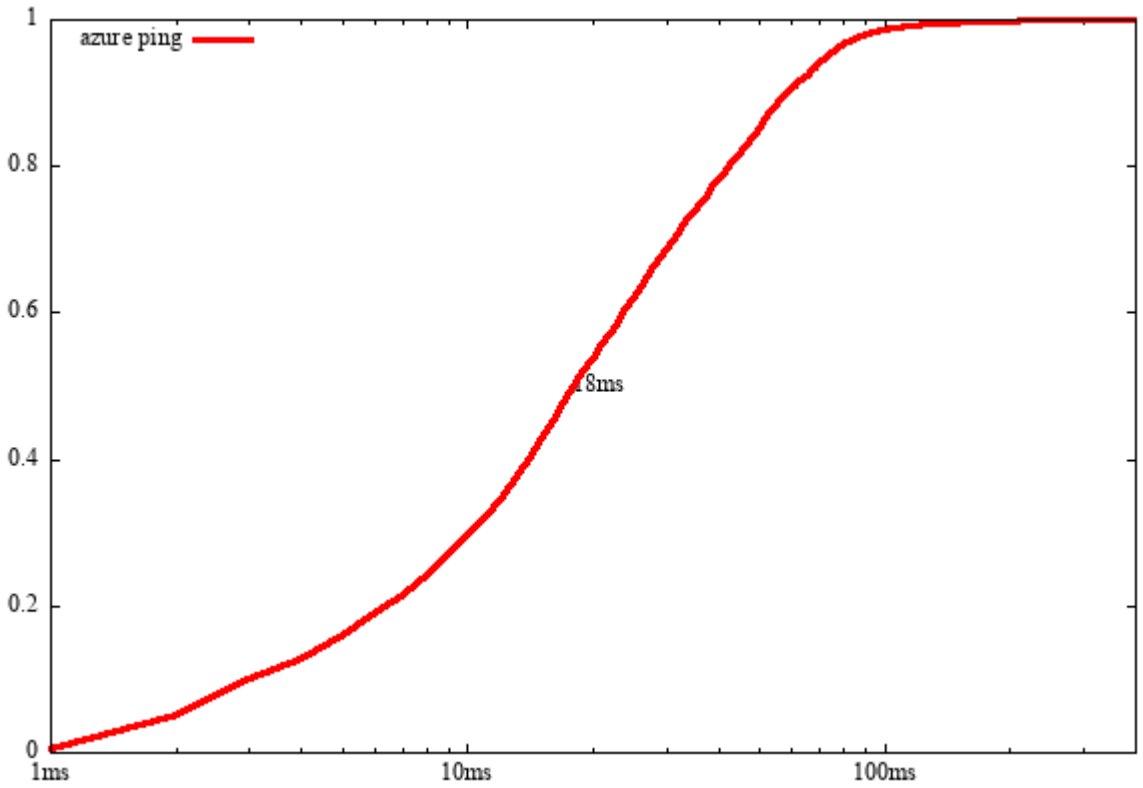
perl mz-build-rtt.pl harsh-20230518-routers.txt.bz2 /home/ubuntu/aggregate-results/mz/mz*.geoloc-pinger.*.warts.bz2 >mz-minrtt.txt

- To prune this text file, update the name of text file in prune-rtt-file.py and the azure-vploc.txt file and run the following command:
  python3 prune-rtt-file.py
- Once the pruned min-rtt file is generated, we can run hoiho on the txt file using the following command:
  sc_hoiho -O json -O learngeo -d best-regex -g geocodes.txt -g azure-vploc.txt -R mz-minrtt-prunned.txt public_suffix_list.dat harsh-20230518-routers.txt >mz-geo.json
- More information about hoiho commands and parameters can be found at https://www.caida.org/catalog/software/scamper/man/sc_hoiho.1.pdf

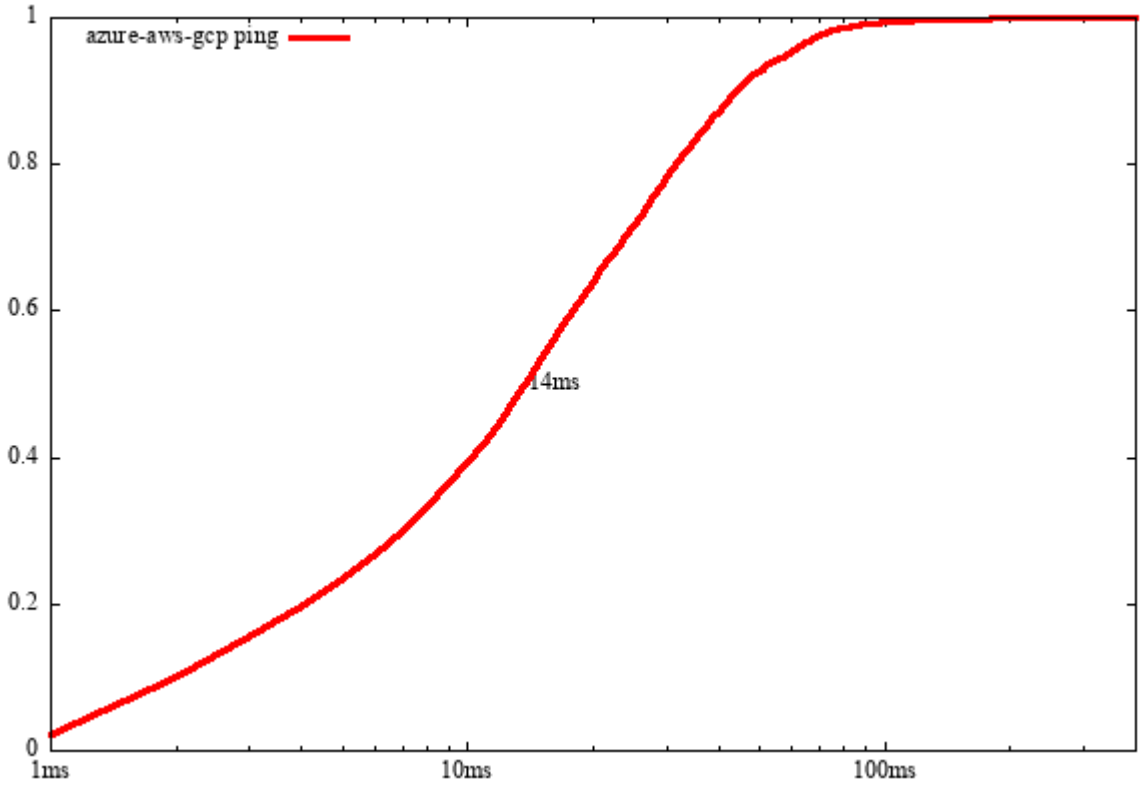
Azure results:
The plot for minimum rtt was generated for the pruned min-rtt file by using the dump-minrtt-dist-diff.pl and plot-minrtt-dist-diff.pl file.
The difference before and after pruning is dumped in results folder in file called mz.diff. The difference was majorly in domains 360.net, 2day.kz, 4830.org, ac-creteil.fr, acesso10.net.br, adamant.net, ahrt.hu, akamai.com, onecommunications.net etc

Commands used:

- perl dump-minrtt-dist-diff.pl --mode rtt-union mz-minrtt-prunned.txt -> This command generates the cdf for 400 values which can be copied onto a text file
- perl plot-minrtt-dist-diff.pl --mode rtt-union --nolabels mz.png mz-cdf.txt -> This command generates the plot of minimum rtt.



Combined Results of AWS, GCP and Azure
Combined all the min-rtt-prunned text files from all 3 cloud providers using the cat command.
Modified the dump-minrtt-dist-diff.pl and plot-minrtt-dist-diff.pl file to generate the graph.


azure-aws-gcp ping
14ms
1
0.8
0.6
0.4
0.2
0
1ms
10ms
100ms