AWS

Graviton Processor
https://aws.amazon.com/ec2/graviton/

EC2

Instance types: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html
Price Calcualtor: https://aws.amazon.com/ec2/pricing/on-demand/
ESB Volume Types: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html

Mount a new disk to instance

https://docs.aws.amazon.com/ebs/latest/userguide/ebs-using-volumes.html

```
# To see device attached
lsblk
# If output shows data, that means no FS on the disk
sudo file -s /dev/nvme1n1
# See devices with FS
sudo lsblk -f
# TO format the disk with XFS FS
sudo mkfs -t xfs /dev/nvme1n1
sudo mkdir /mysql-data
sudo mount /dev/nvme1n1 /mysql-data
sudo cp /etc/fstab /etc/fstab.orig
sudo blkid
# Add this entry in /etc/fstab file
UUID=6a9cc469-cee4-4f78-a0cb-7f5cda41f052  /mysql-data  xfs  defaults,nofail
0  2

# Unmount the device
sudo umount /mysql-data
# Try to mount with fstab file entry to see if it shall mount during machine
start or not
sudo mount -a
```

Increasing the disk size (partition of EC2 instance)

After you modify volume from AWS console, then you have to run below commands in this order.

```
# First grow partition size on the disk
sudo lsblk
```

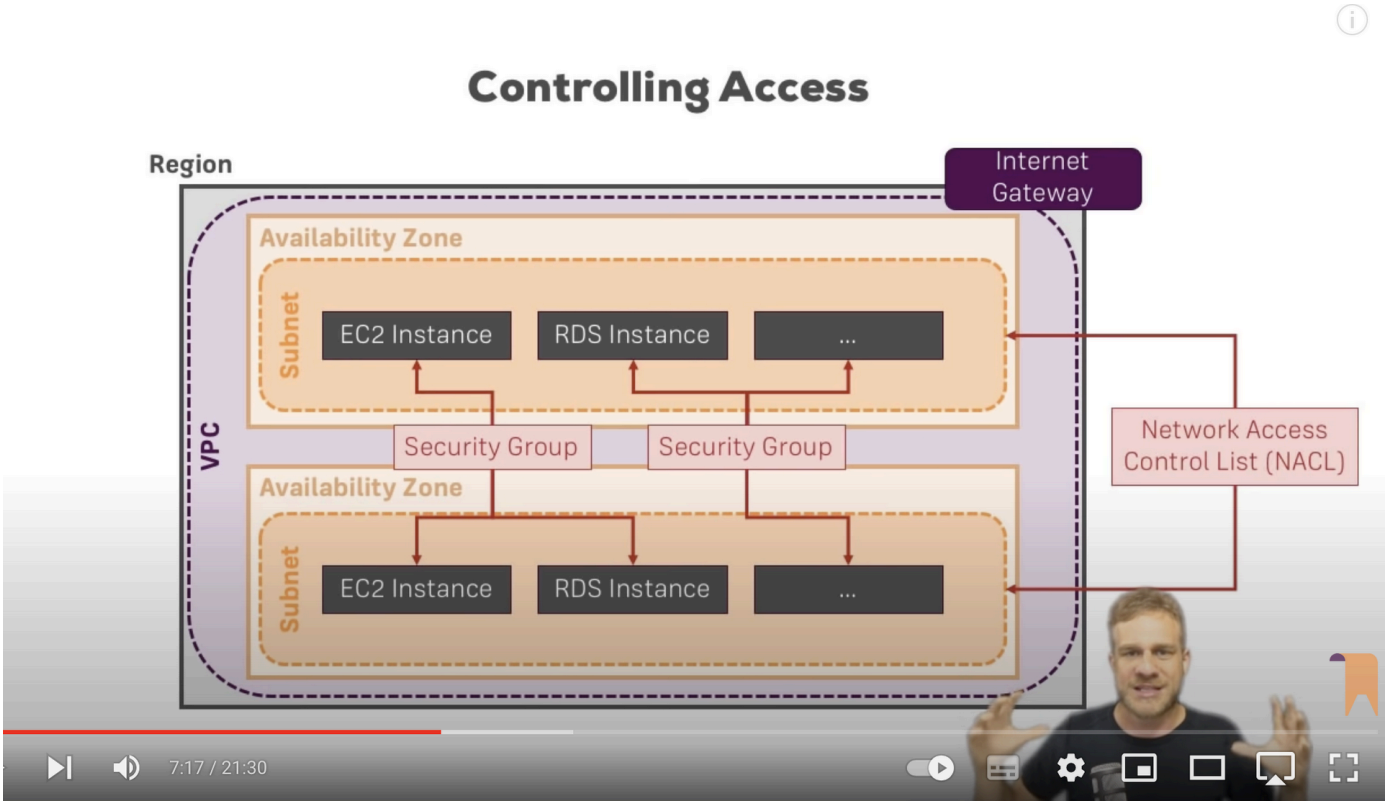
```
sudo growpart /dev/nvme1n1 1
# Then grow the filesystem on parttion to fill the partition
sudo resize2fs /dev/nvme1n1
```

```
sudo lsblk
sudo growpart /dev/xvda 1
sudo resize2fs /dev/xvda1
```

How to tune your EC2 instance

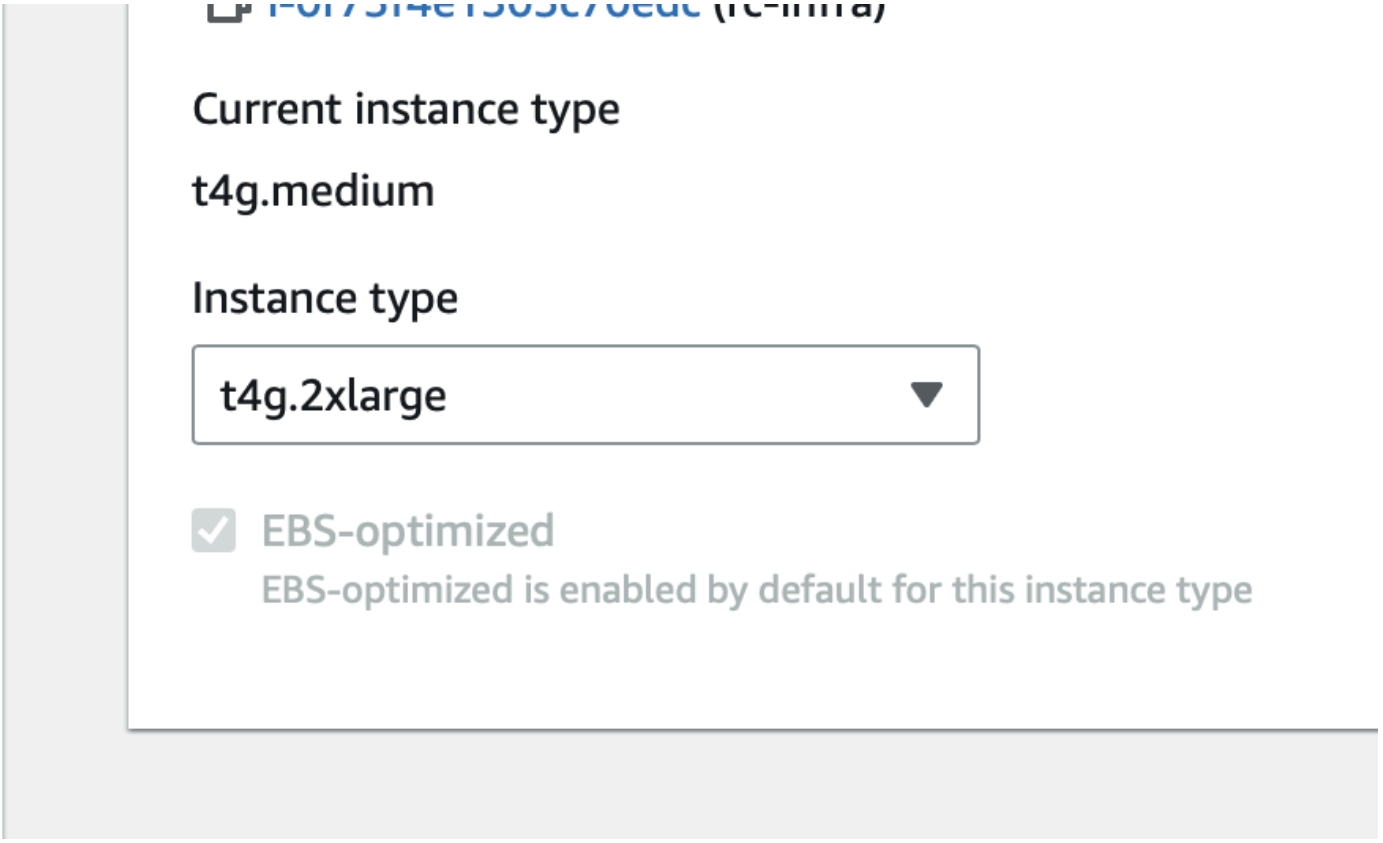
https://www.slideshare.net/brendangregg/how-netflix-tunes-ec2-instances-for-performance

https://www.slideshare.net/brendangregg/how-netflix-tunes-ec2-instances-for-performance



RC-Infra

Temp elevation



Setting up AWS CLI

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html

https://docs.aws.amazon.com/singlesignon/latest/userguide/howtogetcredentials.html

https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp_use-resources.html#using-temp-creds-sdk-cli

Created Permissions using Inline Policy and chose STS (Security Token Srevice) service to grant read permission to a group. Added my user to that group.

For HB Stage, this is my setup on my local

https://us-east-1.console.aws.amazon.com/iamv2/home#/users/details/akhilesh?section=security_credentials

```
akhilesht@akhilesh:~/work/keys # cat ~/.aws/config
[default]
region = ap-south-1
output = json
akhilesht@akhilesh:~/work/keys # cat ~/.aws/credentials
[default]
aws_access_key_id =
aws_secret_access_key =
```

Setting up AWS CLI

To generate the credentials while using MFA with user, use below command:

This is how we can get the token:
https://repost.aws/knowledge-center/authenticate-mfa-cli

```
aws sts get-session-token --serial-number <mfa/arn> --token-code 240996

aws sts get-session-token --serial-number <mfa/arn> --token-code 240996
```

It shall give use credentials back:

```
{
    "Credentials": {
        "AccessKeyId": "adfa",
        "SecretAccessKey": "adfa",
        "SessionToken": "",
        "Expiration": "2023-04-21T21:59:45+00:00"
    }
}
```

Then you can export these as variables:

```
export AWS_ACCESS_KEY_ID=example-access-key-as-in-previous-output
export AWS_SECRET_ACCESS_KEY=example-secret-access-key-as-in-previous-output
export AWS_SESSION_TOKEN=example-session-token-as-in-previous-output
```

Now you can actually use this function in your rc script: here

Lambda Functions

To increase the timeout from lambda function

Here is what we did.

This is the code

```
function doWork() {
    return new Promise((resolve, reject) => {
        function getRespone() {
            return {"offline": false, "new_endpoint": "https://<endpoint>"};
        }

        setTimeout(() => {
            resolve(getRespone());
```

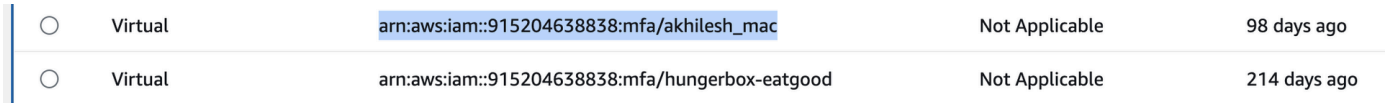
```
        }, 9000);
    });
}

export const handler = async(event) => {
    return await doWork();
};
```

Then on lambda level - in general configuraiton section - we increased the timout value from default 3 seconds to 10 seconds.

Then on API gateway layer as well - which was using this lambda function - increased timeout there and deployed that as well.

To upload ECR image from aws cli to ecr



○	Virtual	arn:aws:iam::915204638838:mfa/akhilesh_mac	Not Applicable	98 days ago
○	Virtual	arn:aws:iam::915204638838:mfa/hungerbox-eatgood	Not Applicable	214 days ago

```
# First get access keys etc from cli
aws sts get-session-token  --serial-number  '<mfa/arn>' --token-code 301581
```

Then I simply ran these commands as shown in the pic.

Push commands for hb-django-scheduler

macOS / Linux | Windows

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see Getting started with Amazon ECR.

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry authentication.

1. Retrieve an authentication token and authenticate your Docker client to your registry.
   Use the AWS CLI:

   ```
   aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 915204638838.dkr.ecr.ap-south-1.amazonaws.com
   ```

   Note: if you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.
2. Build your Docker image using the following command. For information on building a Docker file from scratch, see the instructions here. You can skip this step if your image has already been built:

   ```
   docker build -t hb-django-scheduler .
   ```
3. After the build is completed, tag your image so you can push the image to this repository:

   ```
   docker tag hb-django-scheduler:latest 915204638838.dkr.ecr.ap-south-1.amazonaws.com/hb-django-scheduler:latest
   ```
4. Run the following command to push this image to your newly created AWS repository:

   ```
   docker push 915204638838.dkr.ecr.ap-south-1.amazonaws.com/hb-django-scheduler:latest
   ```

Close

Set up hb-infra vpc

Idea behind this VPC is that this shall be very closely protected vpc which shall host our important main infra services. Mainly private subnets only. No outside traffic allowed. Only traffic shall be allowed from other VPCs that too in a very controlled manner.

- first we created hb-infra vpc (quota had to be increased, since default quota is 5)
  - Enabled ipv6 here (because I want to use egress only nat gateway for internet access for this VPC. for downloading softwares etc on instances running in this VPC)
- then we created a subnet in the vpc (ap1, 1a region)
  - Enabled ipv6 again
- Then I created egress only nat gateway and associated it with above created private (not public) subnet.

- Then I launched an instance in this VPC. Had added user data to this so that I can login into the machine from aws console.

```
#!/bin/bash

# Set the password for the Ubuntu user
passwd -d ubuntu

# Generate a strong password
password=appasadfaf

# Set the password for the Ubuntu user
echo "ubuntu:$password" | chpasswd
```

- Then I paired the VPC with other VPCs.
- Added CIDR block (of prod vpc) on route table of hb-infra vpc to allow traffic
- Added CIDR block (of hb-infra vpc) on route table of prod vpc
- Then Fianlly I tuned the network acl of subnet created in hb-infra VPC. Read this for thoroguh understanding of network acl rules
- This post here helped me in finding the right ephimeral ports for ubuntu instance.
    - From above post: You almost certainly don't need to use network ACLs unless you have a particularly complex network configuration. The inbound rules in the security group are usually sufficient to control access to an instance. Inbound security group rules deny by default, and unlike Network ACLs, which are stateless packet filters, security groups are stateful, TCP session-aware.

Some notes about Network ACL

- NACLs are stateless, which means that information about previously sent or received traffic is not saved. If, for example, you create a NACL rule to allow specific inbound traffic to a subnet, responses to that traffic are not automatically allowed. This is in contrast to how security groups work. Security groups are stateful, which means that information about previously sent or received traffic is saved. If, for example, a security group allows inbound traffic to an EC2 instance, responses are automatically allowed regardless of outbound security group rules.
- A network ACL has inbound rules and outbound rules. Each rule can either allow or deny traffic. Each rule has a number from 1 to 32766. We evaluate the rules in order, starting with the lowest numbered rule, when deciding whether allow or deny traffic. If the traffic matches a rule, the rule is applied and we do not evaluate any additional rules. We recommend that you start by creating rules in increments (for example, increments of 10 or 100) so that you can insert new rules later on, if needed.
- Network ACLs do not filter traffic destined to and from the following:
  Amazon Domain Name Services (DNS)
  Amazon Dynamic Host Configuration Protocol (DHCP)

Amazon EC2 instance metadata
Amazon ECS task metadata endpoints
License activation for Windows instances
Amazon Time Sync Service
Reserved IP addresses used by the default VPC router

Mounting disk

See `ELK stack in rc env` Note in this notebook.

To format a disk that I had mistakenly formatted as ext4

- `umount /data`

S3 Getting bucket data to local system (backup/copy s3 local folder)

Configure aws cli with right credentials
`aws configure`

Then simply sync the bucket to local folder

`aws s3 sync s3://<bucketname> .`

This should bring all the data to this folder.

Copy local folder to s3 bucket

`aws s3 cp --recursive <local fodler> s3://<bucketname>/<prefix>`