Cred - <https://637423281818.signin.aws.amazon.com/console>

Pavani

Bapuji@1964

Access key - AKIAZI2LDM2NOF5DQ6M3

Secret key - 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

AWS - [https://us-east-2.console.aws.amazon.com/console/home?region=us-east-2#](https://us-east-2.console.aws.amazon.com/console/home?region=us-east-2)

Un [pavanibillapati112@gmail.com](mailto:pavanibillapati112@gmail.com)

Password – Bapuji@196

<https://826808407770.signin.aws.amazon.com/console>

ITVCloud9User

Bapuji@1964

Unix/ubuntu username: pavani

Password: pavani

With power shell, we can connect to remote device (can use ubuntu) without putty

Install ubuntu on power shell

wsl -l –-online 🡪 shows all available linux distributions

wsl –-install -d Ubuntu<version>

Once installed restart n login into ubuntu -> wsl -d ubuntu<version num>

Python would be installed by default

Exit() -> to exit from ubuntu

Install venv n pip(sometimes we need to update the list prior to installations 🡪 sudo apt update)

sudo apt-get install python3-venv

sudo apt-get install python3-pip

To validate or create venv installation 🡪 python3 -m venv <virtual env name>

To enter to virtual 🡪 source venv/bin/activate

wsl -l -v 🡪 to see all the virtual machines

if we have only 1 virtual machine js by wsl command we can directly connect to that virtual machine

uname -a 🡪 to check the environment

ls -ltr 🡪 to check the files

cd 🡪 brings to home directory of root user

**Setup AWS CLI to interact with aws from local machine**

Pip install awscli – install & then relaunch terminal

aws –-version – to check version

**Create an iam user**

Login to aws, search for iam -> users -> create user(provide a user name & click next)

In set permissions – choose attach policies manually – can add required permissions since its root user giving administrator access & next

In review & create – click create user

A screenshot of a computer

Description automatically generated

Select the user here & once it’s opened, ‘ll able to create access key save or download access key & secret key, are used for logging to aws cli

Access key 🡪 AKIA4BAM7C3NACIKHCVI

Secret key 🡪 H2ykSaFPIdNMP7jasSmLoXefaWFYfgolQAcx8ziK

Go to power shell

* aws configure -> creates default profile(or)
* aws configure –-profile <iam username> 🡪 prompts for access & secret keys

This creates .aws folder in root folder with credentials & config files

* Credentials file contains access key & secret given above
* aws s3 ls – to check buckets in default profile
* aws s3 ls –profile <iam username> 🡪 to check buckets in specific user’s profile

create a folder & setup a virtual environment – creates default python packages

can be checked using find <virtual env name>

or ls -ltr <virtual env name>

activate the environment & install boto3 🡪 used to write python based logic to interact with aws services.

However it’s not advised to use instead we can use IDE for interacting with aws

pip install boto3

to check boto3 installation

python3

>> import boto3

>>s3\_client = boto3.client(‘s3’)

>> s3\_client.list\_buckets()

**Setting up jupyter to interact with AWS**

pip install jupyterlab

jupyter lab – to launch jupyter

with boto3 – we can access aws from jupyter

how is it authenticated to login ?? it happens internally by checking credentials file

**Cloud9**

**Setup**

1. Create IAM group ITVcloud9 & add admin policy to it
2. Cerate an iam user ITVcloud9User & add this user to group ITVcloud9
3. Login as iam user of ITVcloud9
4. Go to cloud9 console
5. Create cloud9 instance using linux2
6. Explore aws services

Iam user – create access key & under manage console -> create password

Access key - AKIA4BAM7C3NIYLFQMG2

Secret key - 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

<https://826808407770.signin.aws.amazon.com/console>

ITVCloud9User

Bapuji@1964

Log into this

Highly recommended to not create cloud9 using root user but use as IAM user

Once logged in as IAM user, search for cloud9 & create environment with default settings

Launch it

This is the IDE we use to create codes on AWS

* Git, docker httpd are already installed by default in cloud9

To check

Git

Sudo systemctl status docker

Sudo systemctl status httpd

Docker is not only installed but also actively running by default

Docker ps 🡪 to see all containers running

Docker ps -a 🡪 to see all containers

Docker images 🡪 to see docker images (like snapshots of a Docker container, containing everything needed to run an application, including the code, system tools, libraries, and settings)

Docker rm <container name> 🡪 to delete docker container

Docker rmi <image name> 🡪 to delete docker image

Docker run <file>.py 🡪 to run a python file

Aws s3 ls 🡪 to see all buckets/check aws services accessibility from cli

* Aws cli is also available in cloud9
* By default cloud9 inherits all permissions from the user
* Many programming languages are also configured like python java(JRE &JDK)

**Cloud9 & EC2**

A computer screen shot of a computer

Description automatically generated

**Accessing web applications**

We can directly access web application using ports from IDE

Default port is 8080

This can run by cloud9 -> preview -> preview running application

If application is running on a different url, can be accessed from preview ->configure custom url

Another way, using public DNS

For that check httpd status & activate it

sudo systemctl status httpd – check status– activate

Telnet – is used to validate whether we r able to listen to web service or not

sudo yum install telnet -y

telnet localhost 80 – to check whether we r able to listen to port 80 locally

to be able to access run cloud9 services using public dns (backed ec2 instance) externally, edit inbound rules & add type ->http & source -> my IP

**Assigning Static IP**

Every time an ec2 instance is resumed, it’s public IPV4 & DNS address changes, to avoid this we need to make it static by assigning elastic IP

under network & security -> Allocate elastic IP -> give a name & hit allocate

once IP address is create click on associate IP address -> select the EC2 instance to which u want to allocate -> click associate

elastic ips incur additional costs but is advisable to use so that ip addresses doesn’t change

check if u r able to access applications using static IP address(make sure httpd is active n running 🡪 sudo systemctl start httpd)

**Changing Permissions using IAM policies**

Login as root user into aws account

Go to IAM -> user -> select the user ->add permissions -> attach policies directly-> attach

deattach policies in previous page from above

Adminstrator access – full access

S3 full access – busket full access

Based on role, policies are given to give only required permissions to aws

**Increase size of EBS volume**

Ec2 instance associated to cloud9 instances are created with 10GB storage by default

To check in cloud 9 env, df -h check for /dev/…. size

If we want to change it,

Select the ec2 instance associated -> below go to storage scroll down to volume -> volume id -> volume id -> modify(can change volume type & size)

* To reflect the changes reboot the ec2 instance

Increasing volume doesn’t cause issues however decreasing may sometimes

**Opening ports for cloud9**

We create web apps using cloud9 & want to connect to it we need info like DNS alias & port num

Apache is by default installed up & running in cloud9 🡪 status check – sudo systemctl status apache2

Check the port using 🡪 telenet port 80

Open with public dns only after adding port number

**Setup jupyter lab on cloud9**

Check python availability -> python

Create a folder -> change to it – mkdir <name>, cd <name>

Create virtual environment 🡪 python3 -m venv <name>

Activate 🡪 source <name>/bin/activate

Install jupyter lab 🡪 pip install jupyter lab

pip install 'urllib3<2'

Start jupyter lab webserver 🡪 jupyter lab –ip 0.0.0.0 –gives u an url followed by port number

Open another console tab n check the port 🡪 telnet localhost 8888

Chek if the port number is added under security groups -> inbound rules, If not add it to access

Use public DNS ip of EC2 instance to access jupyter notebook not the hyperlink from AWS cli

Use token from aws cli’s jupyter notebook hyperlink for password or token to login to jupyter

**Connect to aws using SSH from powershell**

make sure to have added ssh 22 port in security group inbound rules - to check if all users would be able to access connecting thru port 22 🡪 telnet ec2-34-232-103-245.compute-1.amazonaws.com 22

ssh-keygen

provide path /home/pavani/.ssh/itvc9demo

click enter when prompted for passphrase

this would generate two keys public & secret keys

cat <public key file path> - displays content of it, copy it

go to cloud9 terminal open ~/.ssh/authorized keys & paste the copied content in the last line, save & exit

ssh -i ~/ssh/itvc9demo [ec2-user@ec2-34-232-103-245.compute-1.amazonaws.com](mailto:ec2-user@ec2-34-232-103-245.compute-1.amazonaws.com)

**Create s3 Bucket**

Let us create an s3 Bucket using AWS Web Console. It will be used to store GitHub Activity Data.

* Go to AWS Web Console and go to s3.
* Create a new bucket by name **itv-github**.
* We can also create folders for landing and raw zones for our data.
* The **landing zone** will be used to ingest data from external sources(data lake)
* We will store data in the landing zone using JSON. Typically data in the landing zone will be deleted. It will act as a scratch pad and we can delete data that is older than 30 days or as per the SLAs.
* The **raw zone** will be used to store data from sources following our data lake standards. In our case we will use parquet as target file format and partition all the data on a daily basis.
* We will have the data in the raw zone up to 7 to 10 years in most of the cases as part of data lakes.

**[Instructions] Create IAM Group and User**

Let us create an IAM Group as well as User to have access to GitHub Activity data and associated jobs. Typically, this step is taken care of by the AWS Admin or DevOps team.

* Go to IAM Console and add a group by the name **ITVGitHubGroup**.
* Go to IAM Console and add the user by name **ITVGitHubUser**. Let us add the user to the group **ITVGitHubGroup**. We will give both AWS web console as well as programmatic access to the user.
* We will only use this user using programmatic access. Make sure to download credentials so that we can configure AWS CLI on our desktop or laptop later.
* All the permissions that are attached to the group via policies will be inherited by all the users in the group.
* Typically we will have multiple users associated with a project and hence we create a group and add all the other users associated with the project.
* We might even have different groups with different permissions in a large scale project. For example, developers, testers, business analysts, devops engineers or administrators etc.

In latest aws versions after user is created -> create access key & enable console access

Console sign-in URL

https://826808407770.signin.aws.amazon.com/console

User name

itvgithubuser

Console password

Bapuji@1964

**[Instructions and Code] Create and Attach Custom Policy**

Let us create a custom policy and provide required permissions on the s3 bucket created earlier.

* We grant permissions to roles or to users via groups by attaching policies with them.
* We can either use AWS Predefined Policies or create custom ones.
* In this case we will create a custom policy by name **ITVGitHubS3FullPolicy** by adding a custom definition.
* Search for aws documentation for read & write access to policy

1. {
2. "Version": "2012-10-17",
3. "Statement": [
4. {
5. "Sid": "ListObjectsInBucket",
6. "Effect": "Allow",
7. "Action": [
8. "s3:ListBucket"
9. ],
10. "Resource": [
11. "arn:aws:s3:::itv-github"
12. ]
13. },
14. {
15. "Sid": "AllObjectActions",
16. "Effect": "Allow",
17. "Action": "s3:\*Object",
18. "Resource": [
19. "arn:aws:s3:::itv-github/\*"
20. ]
21. }
22. ]
23. }

Let us attach policy to users via group.

* Go to the **ITVGitHubGroup**and attach the policy **ITVGitHubS3FullPolicy**.
* All the users which are already added as well as users that will be added in future

Go to iam-> policies -> create policy -> json -> <add policy>(search for aws documentation s3 read & write access, copy & paste -> update the correct bucket name)& hit next -> provide a policy name & hit create policy

Go to user groups-> select group -> select permissions -> add permissions -> Attach policies-> search for policy & add -> click attach policies

**Setup docker for running jupyter lab on it**

1. Check docker if installed

Df -h

1. Pull jupyter docker image

docker pull jupyter/datascience-notebook

*If encountering no space issue, trying increasing the backed EC2 instance & reboot it*

1. Run jupyter lab container

docker run -p 8888:8888 jupyter/datascience-notebook

1. Open jupyter notebook

**Configure and Validate AWS CLI**

* 1. Open terminal from jupyter lab
  2. Check if aws installed by js entering aws
  3. If not installed, update earlier packages by

sudo apt-get update

* 1. Install aws cli

sudo pip3 install awscli

* 1. Verify

aws --version

Let us configure AWS CLI to access the services and resources related to our GitHub Activity Project.

* We can use aws configure command to configure the credentials. We will configure AWS CLI for GitHub Activity project under **itvgithub** profile.
* Run the below command copy paste Access Key as well as Secret Key when prompted. Optionally you can also configure the default region.

aws configure --profile itvgithubpavani

* The credentials will be saved under **.aws/credentials** in our home directory.
* Here is the command to list files under our s3 bucket using the profile.

1. # aws s3 ls <bucket\_name>
2. aws s3 ls s3://itv-github --profile itvgithub

**Getting Started with AWS S3**

Let us get an overview of s3.

* S3 stands for Simple Storage Service.
* It is a low-cost cloud-based permanent storage that is accessible from anywhere based upon the permissions.

**Setup Data Set locally to upload into AWS s3**

Make sure to set up a data set for you to learn about s3.

* I am using **/Users/itversity/Research/data** as the main folder for all my data sets on my Mac. You can choose the location of your choice.
* You can clone the repository from [here](https://github.com/dgadiraju/retail_db.git) into the folder you have decided upon.
* It will create a folder called retail\_db and will contain 6 subfolders.
* Make sure to review the subfolders to confirm data is downloaded.

**Adding AWS s3 Buckets and Objects**

Let us get into the details related to s3 Buckets, Folders, and Objects.

* Create an s3 bucket using AWS Web Console by using your initials and then retail. In my case, it is **dg-retail**. We cannot use underscores and other special characters than hyphen while creating buckets.
* S3 Bucket name has to be unique globally and it is owned by the account that created it.
* Upload all the 6 folders from the retail\_db data set which you might have set up earlier. Using AWS Web Console, you can upload only one file at a time.
* S3 uses object storage (not file storage).

**Version Control in AWS S3**

Let us understand the details related to the versioning of the S3 Bucket. We will also configure the basic life cycle.

* Login to AWS Web Console and go to S3 Management Console.
* Go to the bucket (**dg-retail** in my case).
* Click on properties and review Bucket Versioning. By default, it is suspended.
* You can click on edit and enable versioning.

Why Versioning of the S3 Bucket?

* We might delete the objects or update the objects in S3 accidentally. Versioning will facilitate us to recover the objects using the prior version.
* Due to a bug in the code or mistake, objects might get corrupted. The corrupted files or objects can be overwritten with the prior versions of the files or objects.

Once versioning is enabled we can go to Management and add Life cycle rules. We will add a basic life cycle rule to delete previous versions older than 3 days.

* Click on Management then click on **Create Lifecycle Rule**.
* Name: **Archive Old Retail Files**
* We can filter data by Prefix or Tag or both. In our case, we will filter by Prefix - enter **retail\_db**. The prefix is nothing but the beginning string of the Object Key.
* Under **Lifecycle rule actions**, choose **Permanently delete previous versions of objects**.
* Enter **3** for the **Number of days after objects become previous versions**.
* Click on **Create rule** to create the rule.

We will not be able to validate immediately. You can reload files and then validate after 3 days.

Enable toggle button show versions to see previous versions within subfolders or folders

**AWS S3 Cross-Region Replication for fault tolerance**

Let us go through the details about the **Cross-Region Replication** of s3 Bucket or objects within s3 Bucket.

* In some extreme cases, our S3 might not be accessible within a specific region due to unforeseen circumstances which might impact data centers in AWS Region or AZ within AWS Region.
* By enabling **Cross-Region Replication** we can have a copy of the s3 bucket or objects within the bucket in some other Region.

Let’s enable **Cross-Region Replication** of our bucket **dg-retail**.

* Login to AWS Web Console and go to S3 Management Console.
* Create another bucket in another region by name **dg-retail-copy**
* Create a role by the name **AWSS3FullAccessRole** with **AmazonS3FullAccess** policy.
* Go to the bucket (**dg-retail** in my case).
* Click on Management and go to **Replication rules**.
* Click on **Create replication rule**.
* Replication rule name: **Retail replication**
* Status: **Enabled**
* Choose a Rule Scope: Choose to **Limit the scope of this rule using one or more filters**
* Prefix: **retail\_db** (all objects under retail\_db will be replicated)
* Enter bucket name: **dg-retail-copy**
* Make sure to enable versioning on the destination bucket.
* Make sure to choose **AWSS3FullAccessRole** and click on **Save**.

Note: Only new files added with Prefix defined in the filter will be replicated. Existing files will not be replicated.

After creating replication rule & adding a replication bucket detail doesn’t reflect the copies in destination bucket. Changes made after replication rule creation would only be updated. Deleted files in source bucket would be reflected to replication.

**Overview of AWS S3 Storage Classes or Storage Tiers**

Let us review different [storage classes](https://aws.amazon.com/s3/storage-classes/?sc_icampaign=Adoption_Campaign_m6y20_console_storage_s3_storage-classes&sc_ichannel=ha&sc_icontent=awssm-4885_console-s3_all_users&sc_ioutcome=CSI_Digital_Marketing&sc_iplace=console-s3&trk=ha_a134p000003yIfKAAU~ha_awssm-4885_console-s3_all_users&trkCampaign=CSI_Storage_S3_Storage_Classes) in s3. You can review the storage classes as well as the Performance Chart to get a detailed understanding of Storage Classes.

* We can change the storage class while uploading files either by using s3 Management Console or AWS CLI Commands.
* We can edit the storage class at the level of an object or folder. It is available under **Actions**.
* By default the storage class is **Standard**. You can change to any other storage class. You should be familiar with which type of storage class should be used as per your requirements.
* We can configure the storage class as part of the **Replication Rule**. It is useful when you are using replication for low-cost backup.
  + Click on editfor replication rule.
  + At the destination, change the storage class.
  + Delete on the source bucket **dg-retail** and upload the files again.
  + Refresh **dg-retail-copy** and validate the storage class.
* We can also configure the storage class as part of the Life Cycle. Update lifecycle rule **Archive Old Retail Files** as follows.
  + Change the storage class of noncurrent versions to Glacier with 0 days.
  + Delete noncurrent versions after 3 days.
  + View all the versions by clicking on **List Versions**.
  + It might not be affected immediately.

**Overview of Glacier in AWS s3**

Let us get the details about low-cost storage called Glacier within s3.

* The glacier is a low-cost tier within s3.
* We can use Glacier either to manage older versions or replicas for backup.
* Here are the most common ways in which we can set storage class as Glacier.
  + Edit the object or folder to use Glacier.
  + Configure Glacier as part of lifecycle management to move older versions to Glacier.
  + Configure Glacier as part of defining the Cross-Region Replication rule.

**Managing AWS S3 buckets and objects using AWS CLI**

Let us understand how we can manage s3 using AWS CLI.

* We have a subcommand called s3 under aws main command.
* You can get help on the **aws s3** command using the below command

aws s3 help

* Here are some of the important commands under s3.
  + Listing objects and folders - ls
  + Copying files - cp
  + We can use cp to copy the files from the local file system to s3, s3 to the local file system as well as s3 to s3.
  + Moving objects or folders - mv
  + Deleting objects or folders - rm
  + Creating bucket - mb
  + Removing bucket - rb
* You can get help on any subcommand using the below format.

1. aws s3 <subcommand> help
2. aws s3 ls help # Example

aws s3 ls s3://<bucketname> –-summarize -–recursive –-human-readable

**Managing Objects in AWS S3 using AWS CLI - Lab**

You can perform the below tasks to make sure that you are comfortable using CLI to manage objects in s3.

* List the folders in the **dg-retail** bucket created earlier. It is recommended to list the objects recursively to review all the objects.

ITVCloud9User:~/environment $ aws s3 ls s3://pb-retail --recursive

* Delete the folders in the **retail\_db** main folder from the bucket created earlier.

ITVCloud9User:~/environment $ aws s3 rm s3://pb-retail/retail-db --recursive

* Go to AWS Web Console and confirm that folders and objects in the **retail\_db** folder within the**dg-retail** bucket are deleted.
* Copy all the folders along with files in the retail\_db folder from the local file system to s3. Make sure to copy these folders along with the files in them.
  + departments
  + categories
  + products
  + orders
  + order\_items
  + customers

aws s3 cp C:\Users\pavan\OneDrive\Desktop\prep\AWS\retail\_db-master s3://pb-retail/retail-db

copy individual folders or else all files would be copied under retail-db without respective folders

* Make sure to validate using both CLI as well as AWS Web Console.
* Hint: You can delete as well as copy all the folders and files using one command in recursive mode.

aws s3 cp C:\Users\pavan\OneDrive\Desktop\prep\AWS\retail\_db-master s3://pb-retail/retail-db –-recursive - - exclude “\*.sql” - - exclude “README.md”

**Creating IAM Users**

Let us get into the details related to IAM Users.

* Login as a root or user with Administrator policy
* Go to IAM Service
* Create the user and review the options

**Logging into AWS Management Console using IAM User**

Here are the instructions to login into Management Console using IAM User.

* A link will be generated for each AWS account with an AWS account id.
* We can use that link to launch the login page with the Account id pre-populated.
* Enter the IAM username and password shared with you. It will ask to reset the password if **Users must create a new password at next sign-in** is selected while creating the account.
* Make sure to reset the password adhering to the password policy.

**Validate Programmatic Access to IAM User**

Let us go ahead and validate programmatic access by configuring AWS CLI using the access key and secret key generated for our new account.

* You need to make sure that AWS CLI is set up on your system.
* Launch the terminal or shell and run the**aws configure** command to configure with new keys.
* You can create a sample bucket and validate to confirm that your credentials are configured to the user in the right account.
* Here are the steps I have followed.
  + Configure credentials using profile by name **itvadmin**.

1. aws configure --profile itvadmin

* Create an s3 bucket with some unique name.
* Run the following command to list the newly created bucket.

1. aws s3 ls --profile itvadmin

**IAM Identity-based Policies**

Let us understand [IAM Identity-based Policies](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html#policies_id-based). We will be focusing on Predefined policies for now.

* Permissions are assigned to policies using JSON Syntax.
* Policies are typically attached to either group or role. It is also possible to attach policies directly to users.
* Users, groups, and roles are also known as **IAM identities**.
* We can attach more than one policy to an IAM identity.
* We have predefined policies which we can leverage and also we can define custom policies.
* Let us review the following policies to understand how permissions on services are typically defined.
  + [AmazonS3FullAccess](https://console.aws.amazon.com/iam/home?region=us-west-1#/policies/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonS3FullAccess)
  + [AmazonS3ReadOnlyAccess](https://console.aws.amazon.com/iam/home?region=us-west-1#/policies/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonS3ReadOnlyAccess)
* Here are the key terms which you should be familiar with related to policies.
  + Effect - This is where you typically define Allow or Deny
  + Action - This is where you typically define the type of actions that can be performed
  + Resource - We can control the permissions over the resources that are related to the Effect.
* For example:
  + Effect - s3 (service)
  + Action - Get, List, Put, Delete, etc.
  + Resource - Buckets or objects within the bucket we have created.
* Let us perform a few tasks related to Identity-based Policies.
  + Create a new user **itvsupport1** with only programmatic access.
  + Configure AWS CLI with profile **itvsupport1**.
  + Attach [AmazonS3ReadOnlyAccess](https://console.aws.amazon.com/iam/home?region=us-west-1#/policies/arn%3Aaws%3Aiam%3A%3Aaws%3Apolicy%2FAmazonS3ReadOnlyAccess) to **itvsupport1**.
  + Make sure a bucket is created using AWS Web Console by logging in as admin or root user. I will be creating a bucket by the name **dg-retail**. If you already have such a bucket, you can directly copy files into S3.
  + Try running this command to copy files into the bucket.

1. aws s3 cp ~/Research/data/retail\_db s3://dg-retail1/retail\_db/ \
2. --recursive \
3. --exclude "\*.sql" \
4. --exclude "README.md" \
5. --profile itvsupport1

**Managing IAM Groups**

Let us go through the details related to IAM Groups.

* We typically have users in IAM Groups.
* Permissions are typically defined using Policies and policies are assigned to Groups.
* All the users in the group will inherit all the permissions associated with the group.

Let us perform these tasks to make sure that we are comfortable in dealing with groups.

* Let us create two groups.
  + itvadmin
  + itvsupport
* Add user itvadmin to group itvadmin
* Add user itvsupport1 to group itvsupport
* Detach **AdministratorAccess** policy from user itvadmin and attach it to group itvadmin
* Detach **AmazonS3ReadOnlyAccess** policy from user itvsupport1 and attach it to group itvsupport
* Run the below command as an itvadmin user to confirm that the user account has all the admin permissions that were assigned earlier directly.

1. aws s3 rm s3://dg-retail1/retail\_db/ --recursive --profile itvadmin
3. aws s3 cp ~/Research/data/retail\_db s3://dg-retail1/retail\_db/ \
4. --recursive \
5. --exclude "\*.sql" \
6. --exclude "README.md" \
7. --profile itvadmin

* Run the below commands as itvsupport1 to confirm that the user account has only read-only access on s3.

1. # Both these commands will fail.
2. aws s3 rm s3://dg-retail1/retail\_db/ --recursive --profile itvsupport1
4. aws s3 cp ~/Research/data/retail\_db s3://dg-retail1/retail\_db/ \
5. --recursive \
6. --exclude "\*.sql" \
7. --exclude "README.md" \
8. --profile itvsupport1
10. # This command should work as the user have s3 read-only access
11. aws s3 ls dg-retail1 --recursive --profile itvsupport1

**Managing IAM Roles**

Roles are used to assigning the permissions on one service to another.

* We typically create roles and attach policies with them.
* All the permissions associated with the roles via policies will be inherited by the service when we attach a role to it.

Let us perform these tasks to understand how the roles are defined and used.

* Create a role by name itvsupport and attach the **AmazonS3ReadOnlyAccess** to it. Make sure to choose the service as EC2 as we want to use this role to attach to the EC2 instance.
* Launch EC2 instance using this role. We will be using the Amazon Linux image as it will come with AWS CLI already setup.
* We don’t need to configure AWS CLI as the permissions are assigned via role to this EC2 instance.
* If you provision, EC2 instance with other operating systems than Amazon Linux, then you need to install AWS CLI first.
* Login to the EC2 instance and run these commands.

1. # This command will fail.
2. aws s3 rm s3://dg-retail1/retail\_db/ --recursive
4. # This command should work as the role has s3 read-only access
5. aws s3 ls dg-retail1 --recursive

To connect to an instance, select that instance click on connect from top right corner -> SSH client, copy the command update the correct path of .pem file & run it from powershell to connect to EC2 instance from powershell

To check the role associated to ec2 isntance scrolldown & select security, under IAM Role – ‘ll be able to see the role associated to the instance.

To change the role associated to instance select the ec2 instance click on actions & select security -> modify IAM & change it.

**Overview of Custom Policies**

We can provide finer granular permissions on any AWS Service or component using Custom Policies. Let us do a lab to understand how we can use custom policies to grant specific permissions to a user on a specific bucket.

* Go to Policies and click on **Create Policy**. We will be using **ITVSupportS3RetailDBAll** as the name for the policy.
* We can enter the service and use the wizard to create the policy or we can import from managed policies and improvise on top of it.
* Let us import **AmazonS3ReadOnlyAccess** and then customize it. We will give the get, put and delete permissions on the retail\_db folder in the dg-retail1 bucket along with list all buckets permission.

1. {
2. "Version": "2012-10-17",
3. "Statement": [
4. {
5. "Effect": "Allow",
6. "Action": [
7. "s3:\*"
8. ],
9. "Resource": "arn:aws:s3:::dg-retail1/retail\_db\*"
10. },
11. {
12. "Effect": "Allow",
13. "Action": [
14. "s3:List\*"
15. ],
16. "Resource": "\*"
17. }
18. ]
19. }

* Once the policy is created, detach the existing policy from the itvsupport group and attach the new policy.
* You can validate by running the below commands. We should be able to delete and copy the files again using **the itvsupport1** profile as **itvsupport1** is part of the **itvsupport** group.

1. aws s3 rm s3://dg-retail1/retail\_db/ --recursive --profile itvsupport1
3. aws s3 cp ~/Research/data/retail\_db s3://dg-retail1/retail\_db/ \
4. --recursive \
5. --exclude "\*.sql" \
6. --exclude "README.md" \
7. --profile itvsupport1
9. aws s3 ls dg-retail1 --recursive --profile itvsupport1
11. # This command should fail
12. aws s3 mb s3://dg-retail2 --profile itvsupport1

**Managing IAM using AWS CLI**

As we have understood how to manage IAM using Web Console, let us get into the details about managing IAM using CLI.

* We should be able to manage IAM components using the**aws iam** command.

1. aws iam list-users --profile itvadmin
2. aws iam list-groups --profile itvadmin
3. aws iam list-roles --profile itvadmin
5. # Lists all AWS Managed Policies as well as custom policies
6. aws iam list-policies --profile itvadmin
8. # List only custom policies
9. aws iam list-policies --scope Local --profile itvadmin

* Let us create a user **itvsupport2** and assign it to the group **itvsupport**.

1. aws iam create-user --user-name itvsupport2 --profile itvadmin
2. aws iam list-users --profile itvadmin
4. aws iam add-user-to-group \
5. --group-name itvsupport \
6. --user-name itvsupport2 \
7. --profile itvadmin
9. aws iam list-groups-for-user \
10. --user-name itvsupport2 \
11. --profile itvadmin

* Let us remove the user from the group and then delete. We cannot delete the user when he is part of the group.

1. aws iam remove-user-from-group \
2. --group-name itvsupport \
3. --user-name itvsupport2 \
4. --profile itvadmin
5. aws iam delete-user \
6. --user-name itvsupport2 \
7. --profile itvadmin
9. aws iam list-users --profile itvadmin

**Getting Started with EC2**

Let us get started with AWS EC2.

* EC2 stands for Elastic Cloud Compute
* It is nothing but a virtual machine provisioned from AWS Data Center for you. The Virtual Machines are provisioned from availability zones within a region.
* EC2 instances are provisioned using the instance type. An instance type is nothing but a combination of pre-configured CPU, Memory, and Instance Store.
* On top of CPU, Memory, and Instance Store, we also need to choose Operating System, at least one EBS for the root file system, VPC for networking, security groups for firewall security, and EC2 Key Pair to login into the system.
* EC2 instances can be in a running or stopped state. You will be charged only when the instance is up and running (except for storage).
* You can attach additional EBS Volumes to add more storage and you can create snapshots for the volumes periodically for backups.

**Create EC2 Key Pair**

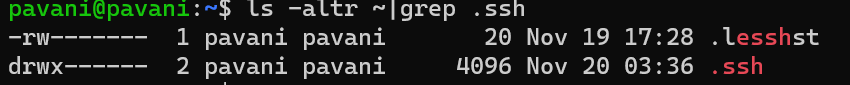
Let us understand how to create EC2 Key Pair so that we can connect to the EC2 instance from external clients such as our PC or Mac.

* We can either create a key pair or use an existing one while creating an EC2 instance.
* It is not good practice to create key pairs while creating EC2 instances.

Here are the instructions to create a key pair.

* Login into AWS Web Console.
* Search for EC2 and go to EC2 Console.
* On the left sidebar, you have EC2 Key Pair under Security and Networking. Click on EC2 Key Pair and then click on “Create Key Pair”.
* Make sure to download the pem file and save it to the standard location on our PC or Mac. You need this later to connect to the EC2 instance.

Let us understand what happens when we create a Key Pair.

* A random private key and the associated public key will be created.
* Private Key files will be provided to you for download.
* Public Key will be managed somewhere inside AWS repositories.
* When you launch an ec2 instance with a specific key pair, the public key will be added to authorized\_keys on the server under standard user.
* Ssh Directory should contain read, write & append permissions as below
* ls -altr ~|grep .ssh – to check .ssh directory permissions
* 
* Move the downloaded key pair file to .ssh
* Check permissions of .pem file

A screen shot of a computer

Description automatically generated

Modify permissions so that it has only read write permissions to owner.

**Launch EC2 Instance**

Let us understand how we can launch an EC2 instance using AWS Web Console. Follow the video to understand how to create an EC2 instance using Ubuntu 18.04.

* You can leverage free tier instances to learn how to create instances.
* Go to AWS Management Console and then to EC2 Console.
* Click on Launch Instance and follow the steps as demonstrated.

**Connecting to EC2 Instance**

Let us understand how to connect to EC2 Instance using SSH from Mac or Linux. We can use the Mac or Linux terminal to connect to remote servers via SSH.

Here is the command for reference.

1. ssh -i ~/.ssh/itvaws.pem \
2. ubuntu@ec2-34-232-68-20.compute-1.amazonaws.com

A computer screen with white text

Description automatically generated

**Security Groups Basics**

Let us understand the basics of security groups. AWS EC2 Security Groups facilitate us to define Firewall rules to block unintended traffic into EC2 instances from external systems.

* We need to choose an existing security group or create a new one while launching an ec2 instance.
* By default, SSH Daemon will be running on Linux-based ec2 instances on AWS using port 22.
* The default rule as part of most of the security groups is to open port 22 so that we can connect from remote machines using SSH.
* We will be able to login using SSH only.

Let us perform these tasks to understand the relevance of security groups.

* Make sure you can connect to the instance that is launched earlier using SSH.
* Go to the security group associated with the ec2 instance and delete the rule related to port 22.
* Try connecting to the ec2 instance using SSH. It will be struck.
* Go back and open port 22 for everyone and validate by using SSH.
* Install Apache on the Ubuntu machine. Login and run these commands to install and start the Apache Web server.

1. sudo apt update
2. sudo apt install -y apache2
3. sudo systemctl status apache2
4. # Hit Ctrl+C to come out of the command output
6. # Confirm that the apache server is running on port 80 using telnet
7. telnet localhost 80
8. telnet ec2-34-232-68-20.compute-1.amazonaws.com 80

* Go to the browser and try to access Apache Web Server using http://<public\_ip>. Here is the link for your reference - [http://ec2-34-232-68-20.compute-1.amazonaws.com](http://ec2-34-232-68-20.compute-1.amazonaws.com/)
* The link will not work. Now go to the security group and the new rule for port number 80 using **My IP**.
* Now the link should work without any issues. It will launch the default Apache Web Server Page for Ubuntu.

**Public and Private IP Addresses**

Let us understand the concepts behind public and private IP addresses associated with AWS EC2 Instances.

* Typically 2 IP Addresses and corresponding DNS aliases will be attached to each AWS EC2 Instance.
* Public DNS aliases start with **ec2** and Private DNS aliases start with **ip**.
* Public DNS alias or underlying Public IP address can be used to access EC2 instances or services running on it via the internet from outside of AWS.
* Private DNS alias or underlying Private IP address can be used for internal communication between EC2 instances within AWS VPC.
* Connect to 1 ec2 instance using public ip

ssh -i ~/.ssh/iamkey.pem [ec2-user@ec2-34-228-61-233.compute-1.amazonaws.com](mailto:ec2-user@ec2-34-228-61-233.compute-1.amazonaws.com)

* connect to other ec2 instance from this ec2 using former’s private ip address

telnet ip-172-31-24-77.ec2.internal 22

* By default, the Public DNS alias or Public IP address is ephemeral. It means if you stop and start an EC2 instance, most likely the Public DNS alias and Public IP address will change.

**EC2 Life Cycle**

Let us go through the life cycle of EC2 Instance. EC2 instance will be in one of these states as long as it is not terminated.

* Running
* Stopped
* Restarting

If you stop the EC2 instance, the public IP might be reset as it is ephemeral by default. You need to lease elastic IP and assign it to the EC2 instance so that public IP does not change.

* In case of restarting an ec2 instance, the public ip address wnt be deassociated but remains the same.

**Allocating and Assigning Elastic IP Addresses**

Let us understand how we can configure a static elastic IP address for EC2 Instance.

* Go to AWS Web Console and then go to EC2 Console.
* Go to **Elastic IPs** under **Network & Security**.
* Click on **Allocate Elastic IP address** to lease the IP address so that we can associate with the ec2 instance created earlier.
* Select newly allocated elastic IP address, go to **Actions** and click on **Associate Elastic IP address**.
* Select the EC2 instance in the drop-down related to Instance.
* Now validate by running SSH using newly associated Public DNS Alias and make sure that you can connect to EC2 instance.

You can also try accessing Apache Web Server that is supported earlier via **http**.

* There would be nominal costs associated with elastic ip addresses

**Managing EC2 Using AWS CLI**

Let us understand how we can manage EC2 Instances using AWS CLI.

* We should be able to take care of all the tasks related to managing EC2 Instances via AWS CLI. Here are some of the examples.
  + Create Key Pair or use existing Key Pair.
  + Create EC2 Instances
  + Attach Security Groups to EC2 Instances
  + Manage Security Group Rules
  + Stop, Restart as well as Terminate EC2 Instance.
  + Allocate and Attach Elastic IP.
  + Describe an instance to get all the metadata associated with it.
* You can get help on **aws ec2** by using the following command.

1. aws ec2 help

Let us warm up by performing some basic tasks against EC2 Instances using AWS CLI. We will be using the **itvadmin** profile as the user account have Administrator Access on AWS Account.

* Describe instances and get the instance ids.

1. aws ec2 describe-instances \
2. --profile itvadmin \
3. --region us-west-1
5. aws ec2 describe-instances \
6. --profile itvadmin \
7. --region us-west-1 | \
8. grep -i instanceid
10. # You can use one of the instance ids and get instance status
11. aws ec2 describe-instance-status \
12. --instance-id i-07c085b765f162233 \
13. --profile itvadmin \
14. --region us-west-1

* Stop the instance and validate whether the instance is stopped or not.

1. aws ec2 stop-instances \
2. --instance-id i-07c085b765f162233 \
3. --profile itvadmin \
4. --region us-west-1
6. aws ec2 describe-instance-status \
7. --instance-id i-07c085b765f162233 \
8. --profile itvadmin \
9. --region us-west-1

* Start the instance and validate whether the instance is started or not.

1. aws ec2 start-instances \
2. --instance-id i-07c085b765f162233 i-00f80143dc2e77b85 \
3. --profile itvadmin \
4. --region us-west-1
6. aws ec2 describe-instance-status \
7. --instance-id i-07c085b765f162233 i-00f80143dc2e77b85 \
8. --profile itvadmin \
9. --region us-west-1

* List Elastic IPs that are allocated so far.

1. aws ec2 describe-addresses \
2. --profile itvadmin \
3. --region us-west-1

**Upgrade or Downgrade EC2 Instances**

Let us understand how to upgrade or downgrade EC2 Instance using AWS Management Console.

* Increasing Memory or CPU or both is called Upgrading the EC2 Instance.
* Decreasing Memory or CPU or both is called Downgrading the EC2 Instance.
* Here we are talking about vertical scaling. Upgrading or Downgrading the same instance with a different configuration is vertical scaling.
* Adding more servers or removing some of the servers from a cluster of servers is called horizontal scaling. We typically talk about horizontal scaling with services like EMR, ECS, EKS, etc.

Let us go through the steps to upgrade as well as downgrade the EC2 Instance.

* Login into AWS Management Console
* Go to EC2 Console and choose the instance to which you want to upgrade.
* Go to **Instance State** and stop the instance. You can also use CLI to stop the instance.
* Go to **Actions**, then choose **Instance Settings** and then click on **Change Instance Type**.
* As of now, the EC2 Instance is of type **t2.micro**. Let us upgrade it to **t2.medium**.
* Start the server and login via SSH to confirm that you can log in to the server.
* Once validated, you can shutdown, downgrade to **t2.micro** and start the server.
* You can then log in once again to validate.
* We can run the below commands to confirm the memory and CPU configuration from within the server.

1. free -h
2. lscpu

* You can also run these commands to change the instance type using the command line.

1. aws ec2 stop-instances \
2. --instance-id i-07c085b765f162233 \
3. --profile itvadmin \
4. --region us-west-1
6. aws ec2 modify-instance-attribute \
7. --instance-id i-07c085b765f162233 \
8. --instance-type t2.micro \
9. --profile itvadmin \
10. --region us-west-1
12. aws ec2 describe-instances \
13. --instance-id i-07c085b765f162233 \
14. --profile itvadmin \
15. --region us-west-1
17. aws ec2 start-instances \
18. --instance-id i-07c085b765f162233 \
19. --profile itvadmin \
20. --region us-west-1

**Understanding EC2 Metadata**

Let us understand details about metadata related to EC2 Instance.

* All the actions that are performed while creating the instance are considered metadata.
  + Instance Type (Memory, CPU, and Instance Store).
  + Security Group
  + EC2 Key Pair
  + Private and Public IP Addresses as well as DNS Aliases
  + VPC and Subnet
* We can get all the metadata in the form of JSON by running the describe-instances command.
* Let us generate the JSON and review it to understand the structure of the metadata.

1. aws ec2 describe-instances \
2. --profile itvadmin \
3. --region us-west-1 > instances.json

**Querying on EC2 Metadata**

Let us understand how we can get specific attributes from metadata while describing instances using aws ec2 command.

* We can use **--query** to project the metadata related to specific attributes.
* As metadata is represented as JSON, we have to specify the attribute using the fully qualified JSON Path.
* Get only instance ids of all the instances.

1. aws ec2 describe-instances \
2. --query 'Reservations[\*].Instances[\*].{Instance:InstanceId,Status:State.Name}' \
3. --output json \
4. --profile itvadmin \
5. --region us-west-1

**Filtering on EC2 Metadata**

Let us understand how we can filter the data while describing instances using aws ec2 command.

* Get all the ec2 instances which are of type t2.micro

1. aws ec2 describe-instances \
2. --filters Name=instance-type,Values=t2.micro \
3. --output json \
4. --profile itvadmin \
5. --region us-west-1

* Get only instance id, type and status of t2.micro instances.

1. aws ec2 describe-instances \
2. --filters Name=instance-type,Values=t2.micro \
3. --query 'Reservations[\*].Instances[\*].{Instance:InstanceId,InstanceType:InstanceType,Status:State.Name}' \
4. --output json \
5. --profile itvadmin \
6. --region us-west-1

* Get only instance id, type and status of t2.micro instances in stopped state.

1. aws ec2 describe-instances \
2. --filters Name=instance-type,Values=t2.micro Name=instance-state-name,Values=stopped \
3. --query 'Reservations[\*].Instances[\*].{Instance:InstanceId,InstanceType:InstanceType,Status:State.Name}' \
4. --output json \
5. --profile itvadmin \
6. --region us-west-1

**Using Bootstrapping Scripts**

Let us understand how to take care of installing additional libraries or softwares as EC2 instances are created and started for the first time.

* Let’s terminate the existing **ec2demo** instance and create a new one with bootstrap script.
* Launch EC2 instance using Ubuntu 18.04
* Install Apache Web Server
* Install Python3 Pip
* Install AWS CLI using pip
* We can use this [document](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html) as reference to prepare the ec2 instance as it is launched.

1. #!/bin/bash
2. apt update -y
3. apt install apache2 -y
4. apt install python3-pip -y
5. python3 -m pip install awscli

* Create instance using aws cli

1. aws ec2 run-instances \
2. --image-id ami-013f17f36f8b1fefb \
3. --count 1 \
4. --instance-type t2.micro \
5. --key-name keyname \
6. --security-group-ids sg-ID \
7. --user-data file://ec2\_user\_data.sh

Bootstrap is nothing but installing additional softwares also to be installed automatically during creation of ec2 instance

During creation of ec2 instance, in advanced details -> user data include the commands which has to be executed after instance is created with #!/bin/bash in 1st line so that it has to interpret the following lines as commands

**Create an AMI**

Let us understand how we can create AMI for existing EC2 Instance’s Volume. In our case, the EC2 instance is **ec2demo** and we will name the image as **ec2demoimage**.

* Go to EC2 Dashboard and then to Instances.
* Select the instance for which you want to create AMI.
* Go to Storage and click on the Volume.
* Create Snapshot for the Volume selected.
* Go to Actions and click on Create Image
* Creating AMI image from CLI

1. aws ec2 create-image \
2. --instance-id i-ID \
3. --name webAppAMI \
4. --description "A sample AMI with preinstalled apache web server"

AMI – is nothing but the operating system we choose while creating ec2 instance. We can choose Ami such that there are few applications along with it(already available or can be customised)

This is becos u dnt have to install additional softwares

 create an AMI from an instance

1. Right-click the instance you want to use as the basis for your AMI, and choose **Create Image** from the context menu.

**Create Image** context menu

1. In the **Create Image** dialog box, type a unique name and description, and then choose **Create Image**. By default, Amazon EC2 shuts down the instance, takes snapshots of any attached volumes, creates and registers the AMI, and then reboots the instance. Choose **No reboot**if you don't want your instance to be shut down.
2. **Create Image** dialog box

It may take a few minutes for the AMI to be created. After it is created, it will appear in the **AMIs** view in AWS Explorer. To display this view, double-click the **Amazon EC2 | AMIs** node in AWS Explorer. To see your AMIs, from the **Viewing** drop-down list, choose **Owned By Me**. You may need to choose **Refresh** to see your AMI. When the AMI first appears, it may be in a pending state, but after a few moments, it transitions to an available state.

**Validate AMI - Lab**

Once the image is created, follow these steps to validate.

* Go to Images and create an instance using it.
* We have to go through the standard steps.
  + Choosing Instance Type
  + Configure Instance Details
  + Add Storage
  + Add or Choose Security Group
  + Launch with a new or existing Key Pair.
* Once the instance is launched, we will name it as **ec2demo1**.
* Wait until the instance is started and then connect via SSH. Once you connect via SSH, make sure to validate that we have Apache 2 as well as AWS CLI already setup and also Apache 2 is started on port 80.
* We can also use command line to create the instance using our AMI

1. aws ec2 run-instances \
2. --image-id ami-ID \
3. --count 1 \
4. --instance-type t2.micro \
5. --key-name keyname \
6. --security-group-ids sg-ID

**Lambda Functions**

**Hello World using AWS Lambda**

Let us start with Hello World using AWS Lambda. We will use Python3 as a runtime environment.

* Go to AWS Console and use the Lambda template.
* Deploy and Test and make sure it is successful.
* Here are the default names that are used.
  + Program Name: **lambda\_function**
  + Function Name: **lambda\_handler**
* We can change the names of the file, function etc. However, we need to ensure that Handler details as part of Runtime Settings are updated.
* Let us rename the file name to **main.py** and validate. We will revert the name back to **lambda\_function.py**.

**Setup Project for local development**

Here are the steps using which we can set up a project. We will use this project to download the GitHub Activity files in incremental fashion.

* Create folder for the project - **ghactivity-downloader**
* Create virtual environment for this project - **ghad-venv**

1. python3 -m venv ghad-venv
2. source ghad-venv/bin/activate

* We will install boto3 in the default location within the virtual environment. We need not include it as part of the bundle that will be deployed as a lambda function.

1. pip install boto3 //required to upload to s3 buckets
2. pip install requests //to download apis from gharchive website

* If we r doing this in lambda boto3 installation is not required as aws python3 environment includes it by default.
* We need to maintain two folders one for development in local system & once for deployment in aws lambda
* We will install requests as part of **lambdalib** folder. It will be included as part of the bundle that will be deployed as a lambda function.

1. mkdir ghalib
2. pip install requests -t ghalib

* We can create **lambda\_function.py** with the default code we got when we get started.

import json

def lambda\_handler(event, context):

# TODO implement

return {

'statusCode': 200,

'body': json.dumps('Hello from Lambda using gha downloader!')

}

* We will create an additional file **lambda\_validate.py** and have this piece of code so that we can run locally.

from lambda\_function import lambda\_handler

res = lambda\_handler(None, None)

print(res)

* We can run **lambda\_validate** to validate locally.

python lambda\_validate.py

Compress-Archive -Path "lambda\_function.py" -DestinationPath "ghactivity-downloader.zip" - to create a zip file

Compress-Archive -Path "lambda\_function.py" -Update -DestinationPath "ghactivity-downloader.zip" - To update the zip file

**Develop download functionality using requests**

Let us develop the download functionality using requests and validate locally. We will understand how to integrate 3rd party libraries as part of the next topic using requests as an example.

* Develop the base functionality to download the zip file using requests library. I have created a new program called download.py for this.

1. import requests
3. def download\_file(file):
4. res = requests.get(f'https://data.gharchive.org/{file}')
5. return res

* Refactor the code as part of lambda\_function.py to invoke the new function and also to capture the response.

1. import json
2. from download import download\_file
4. def lambda\_handler(event, context):
5. download\_res = download\_file('2021-01-29-0.json.gz')
6. return {
7. 'statusCode': download\_res.status\_code,
8. 'body': json.dumps('Download status code')
9. }

**Using 3rd party libraries in AWS Lambda**

Let us understand how to include 3rd party libraries as part of the zip file that will be deployed as lambda function.

* You need to ensure that all the 3rd party libraries which are supposed to be deployed along with lambda functions are downloaded to a folder. In our case it is ghalib.
* We need to go to the folder to build the zip file. Make sure the zip file is created in the base directory of the project and update the zip file with source code.

While creating zip folder make sure only contents of virtual environment folder is copied but not folder itself – it causes folder structure issues

**compress-Archive -Path C:\Users\pavan\OneDrive\Desktop\prep\AWS\ghactivity-downloader/ghalib/\* -DestinationPath ../downloader.zip**

**compress-Archive -Path "download.py","lambda\_function.py" -Update -DestinationPath downloader.zip**

1. rm ghactivity-downloader.zip
2. cd ghalib
3. zip -r ../ghactivity-downloader.zip .

or

Compress-Archive -Path "C:\path\to\your\directory" -DestinationPath "C:\path\to\archive\ghactivity-downloader.zip"

1. cd ..
2. zip -g ghactivity-downloader.zip lambda\_function.py download.py

* We can upload the zip file to AWS Lambda console and validate successfully. Make sure to increase memory size to 512 MB as demonstrated(can be done by gng to configuration tab -> edit & increase the memory size)

**Validating s3 access for local development**

Let us validate s3 access for local development. We need to have the appropriate credentials to access s3 bucket from the local development environment.

* Import required libraries.
* Configure environment variable **AWS\_PROFILE** to appropriate value.
* Develop the code to upload the contents of the zip file from GitHub archive to s3. We will create a new file called **upload.py**.

1. import os
2. import boto3
3. import requests
5. os.environ.setdefault('AWS\_DEFAULT', 'itvgithub')
7. s3\_client = boto3.client('s3')
9. file = '2021-01-29-0.json.gz'
10. res = requests.get(f'https://data.gharchive.org/{file}')
12. upload\_res = s3\_client.put\_object(
13. Bucket='itv-github',
14. Key='2021-01-29-0.json.gz',
15. Body=res.content
16. )
18. print(upload\_res)

**Develop upload functionality to s3**

Let us understand how to upload the GitHub activity file to s3.

* We need to ensure that the role using which lambda function is being executed has permission on the target bucket.
* Let us create a sandbox folder under **itv-github** bucket.
* We need to develop a new function as part of a new program which takes the response object and uploads it as an object in s3.

import boto3

def get\_client():

return boto3.client('s3')

def upload\_s3(body, bucket, file):

s3\_client = get\_client()

res = s3\_client.put\_object(

Bucket=bucket,

Key=file,

Body=body

)

return res

* We also need to update the logic in the lambda handler to call the function which uploads the response as an object in s3.

import os

from download import download\_file

from upload import upload\_s3

def lambda\_handler(event, context):

file = '2021-01-29-2.json.gz'

download\_res = download\_file(file)

bucket = os.environ.get('BUCKET\_NAME')

environ = os.environ.get('ENVIRON')

if environ == 'DEV': // For this also, we need to set environment variable as environ with value as DEV(If condition adds aws profile only when code is run from local environment)

print(f'Running in {environ} environment')

os.environ.setdefault('AWS\_PROFILE', 'itvgithub')

upload\_res = upload\_s3(

download\_res.content,

bucket,

file

)

return upload\_res

We need to add environment variable BUCKET\_NAME <ur bucket name>. Can be done by clicking on Run -> Edit configurations -> move to the program from where u r accessing environment variables -> click on edit near environment variables dialog box -> in the popup click on + symbol & add BUCKET\_NAME, <ur bucket name> in Name & value resp. -> Ok -> Apply -> Ok

* We can rebuild the zip file, upload and run.

1. cd ghalib
2. zip -r ../ghactivity-downloader.zip .
3. cd ..
4. zip -g ghactivity-downloader.zip download.py lambda\_function.py

we need to add environment variable BUCKET\_NAME <ur bucket name> here as well under configuration -> environmental variables & add variables

To check if the role used to create lambda function has enough permissions or not, click configuration -> permissions & click on role – opens the role in IAM, here u can add additional policies

In my case added itvgithubs3fullaccess

& rerun the code -> this ll upload file to bucket

**Validating using AWS Lambda Console**

Let us validate using AWS Lambda Console by uploading the zip file.

* Upload the zip file using AWS Lambda Console.
* Add required environment variables.

Need to add FILE\_PREFIX <sandbox> to environment variables whether it’s local or in Lamdba before running

* The test will fail as there are no required permissions granted to Lambda function.

1. import os
2. from download import download\_file
3. from upload import upload\_s3
5. def lambda\_handler(event, context):
6. file = '2021-01-29-2.json.gz'
7. download\_res = download\_file(file)
8. bucket = os.environ.get('BUCKET\_NAME')
9. file\_prefix = os.environ.get('FILE\_PREFIX')
10. environ = os.environ.get('ENVIRON')
11. if environ == 'DEV':
12. print(f'Running in {environ} environment')
13. os.environ.setdefault('AWS\_PROFILE', 'itvgithub')
14. upload\_res = upload\_s3(
15. download\_res.content,
16. bucket,
17. f'{file\_prefix}/{file}'
18. )
19. return upload\_res

This is to upload files to specific folder which has been created as file\_prefix

**Run using AWS Lambda Console**

We need to grant required permissions on s3 bucket to the role associated with Lambda Function. We can take care of it using the policy which has permissions on s3 bucket.

* Identify the role associated with the lambda function by going to the permissions tab.
* Go to the role and attach the policy to the role associated with Lambda Function.
* We will also copy the files to the sandbox folder within the s3 bucket. Hence, let us create the folder, refactor the code and then deploy.
* Once deployed we can test using AWS Lambda Console and then validate the logs as well as the target folder within s3.

**Validating files incrementally**

Let us implement the logic to validate files incrementally. We will use the current day’s beginning hour as baseline.

* We can convert the date and hour part of the file to timestamp.
* Once we get the timestamp we should be able to use **timedelta** to add 1 hour in each iteration.
* Using the new date and time, we should be able to generate the next file. We can check whether the file is already available or not.

from datetime import datetime as dt

from datetime import timedelta as td

import requests

next\_file = '2021-01-30-0.json.gz'

while True:

res = requests.get(f'https://data.gharchive.org/{next\_file}')

if res.status\_code != 200:

break

print(f'The status code for {next\_file} is {res.status\_code}')

dt\_part = next\_file.split('.')[0]

next\_file = f"{dt.strftime(dt.strptime(dt\_part, '%Y-%M-%d-%H') + td(hours=1), '%Y-%M-%d-%-H')}.json.gz"

**Reading and Writing Bookmark using s3**

Let us go through s3 APIs to read and write bookmark details using s3. For now we will maintain the last copied file as a bookmark.

* Writing content to s3. We will use **s3://itv-github/sandbox/bookmark**.

import boto3, os

from botocore.errorfactory import ClientError

os.environ.setdefault('AWS\_PROFILE', 'itvgithub')

s3\_client = boto3.client(‘s3’)

bookmark\_contents = '2021-01-30-0.json.gz'

s3\_client.put\_object(

Bucket='itv-github',

Key='sandbox/bookmark',

Body=bookmark\_contents.encode('utf-8')

)

* Reading content from s3. If the bookmark is not there we need to catch the exception and use the baseline date as a bookmark.

import boto3, os

from botocore.errorfactory import ClientError

os.environ.setdefault('AWS\_PROFILE', 'itvgithub')

s3\_client = boto3.client(‘s3’)

bookmark\_contents = '2021-01-30-0.json.gz'

try:

bookmark\_file = s3\_client.get\_object(

Bucket='itv-github',

Key='sandbox/bookmark'

)

prev\_file = bookmark\_file['Body'].read().decode('utf-8')

except ClientError as e:

if e.response['Error']['Code'] == 'NoSuchKey':

# Catch exception

# prev\_file = baseline\_file

else:

raise

**Maintaining Bookmark using s3**

We need to maintain the bookmark for the files copied so that when we schedule the job to incrementally pull the next one, in case if the file is available.

from datetime import datetime as dt

from datetime import timedelta as td

import requests, boto3, os

from botocore.errorfactory import ClientError

# Make sure to change the date to one or two day older than based on the date you are running.

baseline\_file = '2023-11-27-0.json.gz'

os.environ.setdefault('AWS\_PROFILE', 'itvgithub')

s3\_client = boto3.client('s3')

while True:

try:

bookmark\_file = s3\_client.get\_object(

Bucket='itv-github',

Key='sandbox/bookmark'

)

prev\_file = bookmark\_file['Body'].read().decode('utf-8')

except ClientError as e:

if e.response['Error']['Code'] == 'NoSuchKey':

prev\_file = baseline\_file

else:

raise

dt\_part = prev\_file.split('.')[0]

next\_file = f"{dt.strftime(dt.strptime(dt\_part, '%Y-%M-%d-%H') + td(hours=1), '%Y-%M-%d-%-H')}.json.gz"

res = requests.get(f'https://data.gharchive.org/{next\_file}')

print(f'https://data.gharchive.org/{next\_file}')

if res.status\_code != 200:

break

print(f'The status code for {next\_file} is {res.status\_code}')

bookmark\_contents = next\_file

s3\_client.put\_object(

Bucket='itv-github',

Key='sandbox/bookmark',

Body=bookmark\_contents.encode('utf-8')

)

**ghactivity-downloader Lambda Function**

refer **ghactivity-downloader-src** in aws folder

Here are the instructions you need to follow to go through the source code.

1. Create a folder by name **ghactivity-downloader** - mkdir ghactivity-downloader
2. Go to the folder **ghactivity-downloader** - cd ghactivity-downloader
3. Download the zip file from the downloadable resources and move the zip file to **ghactivity-downloader**
4. Unzip the zip file **ghactivity-downloader-src.zip** - On Mac or Linux you can use the command unzip ghactivity-downloader-src.zip
5. Create virtual environment using Python 3 - python3 -m venv ghad-venv
6. Activate virtual environment - source ghad-venv/bin/activate
7. Install required dependencies in the virtual environment (boto3, requests)
8. pip install requests
9. pip install boto3

8. Create a folder by name **ghalib** using mkdir ghalib. This will be used for dependencies that are supposed to be shipped along with the source code for our lambda function.

9. Install dependencies that need to be shipped to deploy as a lambda function

pip install requests -t ghalib

10. Build the zip file with the source code and dependencies such as requests using the following commands.

1. cd ghalib
2. zip -r ghactivity-downloader.zip .
3. cd ..
4. zip -r ghactivity-downloader.zip \*.py
5. Now zip file is ready for deployment as a lambda function.

**Schedule Lambda Function using AWS Event Bridge**

Amazon Event Bridge is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. Event Bridge delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services and routes that data to targets such as AWS Lambda. You can set up routing rules to determine where to send your data to build application architectures that react in real time to all of your data sources. Event Bridge enables you to build event-driven architectures that are loosely coupled and distributed.

We can schedule the job using Event Bridge so that files are downloaded and uploaded to s3 as soon as files are available in gharchive.

* Go to AWS Console and search for Event Bride.
* Create rules and schedule every 15 minutes.
* Clean up the bookmark and the files added beyond baseline
* Validate whether the files are downloaded or not.

After the validation is done make sure the event bridge rule is disabled and deleted so that you do not end up paying unnecessarily.

**Setup Virtual Environment and Install Pyspark**

Here are the commands used to set up a virtual environment and install Pyspark.

1. python3 -m venv deod-venv
2. source deod-venv/bin/activate
3. pip install pyspark

**Getting Started with Pycharm**

Getting Started

Here is how we can get started with local development of data engineering pipelines using Spark.

* Create Virtual Environment - python3.7 -m venv itvg-venv
* Activate virtual environment - source itvg-venv/bin/activate
* Install PySpark for local development - pip install pyspark==2.4.\*
* Open using PyCharm and make sure appropriate virtual environment is used from the virtual environment which we have setup.
* Create a program called as **app.py** and enter this code.

from pyspark.sql import SparkSession

spark = SparkSession. \

builder. \

master('local'). \

appName('GitHub Activity - Getting Started'). \

getOrCreate()

spark.sql('SELECT current\_date').show()

**Passing Runtime arguments**

argv - takes 1st argument the name of the program file we are using to run it

we can pass additional arguments while running the program from command line like

python app.py <arguments> or if running from PyCharm we can pass these arguments by Run -> Edit configuration -> Script(path of the python program), script parameters <add these runtime parameters here>, Working directory (path) click apply & ok

import sys

print(type(sys.argv))

print(sys.argv)

In realtime, there would be hundreds of runtime arguments, all of these cant be passed by runtime.

For this we have something environmental variables

**Passing environment variables**

*Program*

#to access environment variables we use ‘os’ library & use os.environ.get

import os

Print(f‘hello world from {os.environ.get(‘key’)}’)

**Through Command line**

export key = value in command prompt

or set key = value in pychram terminal

& run <program>.py

**Thru pycharm**

Run-> edit configuration -> environment variables -> add here

Save & run the program