|  |  |  |
| --- | --- | --- |
| 1. | cat /home/ec2\_user/ .ssh/authorized\_keys | Public key in linux |
| 2. | ssh-keygen –y-q-f privatekey.pem | Create public key from private key |
| 3. | ssh –i key.pem ec2-user@<public\_ip> -vv | Connecting to ec2 mc |
| 4 | ssh-keygen  id\_rsa  id\_rsa.pub | Generate new keypair |
| 5 | scp –i key.pem –v file.txt ec2\_user@<IP>:/home/ec2-user | Copy file from local to remote |
| 6 | ssh key.pem ec2\_user@<IP> ‘df –h’ | Run command on server and come back to local |
| 7 | printenv | Print env varialbles |
| 8 | export | Set env variables |
| 9 | https://bellard.org/jslinux/ | Linux practicing site |
| 10 | curl https://filesamples.com/samples/document/txt/sample3.txt --output | useful when we want a web file to be saved on computer |
| 11 | wget https://filesamples.com/samples/document/txt/sample3.txt -o | Wget is the non-interactive downloader which is used to download files from the server/remote path. |
| 12 | cat sample3.txt | wc -l | See no of lines inside file |
| 13 | cat number.txt | sort | Sort line |
| 14 | cat number.txt | sort -n | Smallest to highest no |
| 15 | cat number.txt | sort -rn | Highest to smallest no |
| 16 | cat number.txt | sort –rn | unique | To give only unique no |
| 17 | Cat alfa.txt | sort | ascending order |
| 18 | Cat alfa.txt | sort -r | Descending order |
| 19 | cat sample3.txt | grep "ipsum" | (pattern matching)  Print all lines having a specific keyword |
| 20 | cat sample3.txt | grep "ipsum" | wc -l | Count all lines having a specific keyword |
| 21 | echo "www:google:com" > test.txt sed 's/:/./g' test.txt > newtest.txt | Stream Editor |
| 22 | sed 's/:/./g' test.txt -i | # To edit the file with inplace changes |
| 23 | echo "this is aws" >> test.txt echo "this is aws" >> test.txt sed '3 s/aws/cli/' test.txt -i | # Replacing string on a specific line number |
| 24 | sed '$d' test.txt -i | Delete last line |
| 25 | sed '2d' test.txt -i | Delete 2nd line |
| 26 | grep –r ‘aws’ . | Search all files and folders in pwd |
|  | grep –r ‘ipsum . |  |
|  | awk ‘{print}’ /etc/passwd | Print all content |
|  | awk -f “:” ‘{print $1}’ /etc/passwd | To print only first column |
|  | awk -f “:” ‘{print $1, $6}’ /etc/passwd | To print only first and 6th column |
|  | awk -f “:” ‘{print $1, $NF}’ /etc/passwd | To print only first and last column |
|  | awk ' NR < 6 ' /etc/passwd | To print first five lines |
|  | awk ' NR==5,NR==10 ' /etc/passwd | print lines from 5 to 10 |
|  | awk -F":" 'NR==5{ print $1 }' /etc/passwd | print only 5th line |
|  | awk -F":" 'NR>0 && NR<6{ print $1 }' /etc/passwd | print range of lines(0 to 5 lines) |
|  | awk '{print NR,$0}' /etc/passwd | # Display Line Number |
|  | awk -F"," '{ print $1 }' company.csv | sort -n | uniq | Find the uniq years in the csv file |
|  | head -10 | See first 10 lines |
|  | tail -10 | See last 10 lines |
|  | find /var -name "\*.conf" | Find  command searches directories and sub-directories for files matching the criteria |
|  | -type d : Only match directories. -type f : Only match files. -empty : Only find empty files |  |
|  | sudo du -a /var | sort -n -r | head -n 10 | **Find the Largest Top 10 Files and Directories On a Linux** |
|  | sudo du -h /var | sort -rh | head -10 |  |
|  | **du command -h** : Display sizes in human readable format (e.g., 1K, 234M, 2G). **sort command -r** : Reverse the result of comparisons **head command -10 OR -n 10** : It shows the first 10 lines. |  |
|  | # stat of file stat /tmp/file1.txt # Find files older than 30 days find /opt/backup -type f -mtime +30 | **Find Files Older Than 30 Days** |
|  | # Create empty csv file touch test{1..5}.csv # Write some content in above create file for f in test1.csv test2.csv test3.csv test4.csv test5.csv; do echo -e "one,two,three,four,five" >> $f; done # Find all csv files and zip into a single file find . -name '\*.csv' -print | zip csvZipFile.zip -@ | **Find All Csv files and create a zip** |
|  | unzip csvZipfile.csv | **Unzip file** |
|  | cat /proc/cpuinfo  lscpu | **Cpu information** |
|  | free -m | **Ram free in mb** |
|  | Free -g | **Ram free in gb** |
|  | uptime | **When you start server** |
|  | top | **See application and process running,can see PID process id** |
|  | ps ux | **Process status which command running under user** |
|  | Kill <pid> | **Stop process** |
|  | ps -elf | **Shows all processes running** |
|  | Pidof bash |  |
|  | sleep 5 | **Sleep cli for 5 sec** |
|  | traceroute | **Can see hub** |
|  | nslookup www.google.com | **Can see server information** |
|  | netstat -nltp | **Can see which port is used by httpd** |
|  | AWS cli approach  1.windows CLI  2.AWS ec2 linux  3.Cloud shell will run shell script |  |
|  | aws ec2 authorize-security-group-ingress --protocol tcp --port 80 --cidr 10.0.0.0/24 --group-id sg-0a97252c1661bf218 | Add a single inbound rule using aws cli command: |
|  | for i in $(cat inbound\_rules.csv); do echo "This is i: $i"; done | echo the content of the csv file using shell for loop command |
|  | cat ./inbound\_rules.csv | awk -F, '{ print $1 }' cat ./inbound\_rules.csv | awk -F, '{ print $2 } | Get the individual value using awk utility |
|  | #!/bin/bash SECURITY\_GROUP\_NAME=$1 INPUT\_FILE\_NAME=$2 REGION\_NAME=$3 SECURITY\_GROUP\_ID=$(aws ec2 describe-security-groups --region $REGION\_NAME --query"SecurityGroups[?GroupName=='$SECURITY\_GROUP\_NAME'].[GroupId]" --output text) echo "SECURITY\_GROUP\_ID value is $SECURITY\_GROUP\_ID --region $REGION\_NAME" if [ $SECURITY\_GROUP\_ID != "" ]; then if [ -f $INPUT\_FILE\_NAME ]; then for i in $(cat $INPUT\_FILE\_NAME);  do INBOUND\_IP=$(echo $i | awk -F, '{ print $1}') INBOUND\_PORT=$(echo $i | awk -F, '{ print $2}') echo "Inbound ip is $INBOUND\_IP" echo "Inbound port is $INBOUND\_PORT" aws ec2 authorize-security-group-ingress --region $REGION\_NAME -- protocol tcp --port $INBOUND\_PORT --cidr $INBOUND\_IP --group-id $SECURITY\_GROUP\_ID done else echo "File $INPUT\_FILE\_NAME does not exists" fi else echo "$SECURITY\_GROUP\_ID is blank, cannot execute" fi | Aading sg entries |
|  | PYTHON |  |
|  | first = 'AWS' last = 'Devops' full\_name = first + ' ' + last print(full\_name) | Concatenation (combining strings) |
|  | age = 21 if age < 4: print("Assigning value of ticket\_price variable as 0") ticket\_price = 0 elif age < 18: print("Assigning value of ticket\_price variable as 10") ticket\_price = 10 else: print("Assigning value of ticket\_price variable as 15") ticket\_price = 15 print("ticket\_price final value is",ticket\_price) |  |
|  | bikes = ['Apache', 'Suzuki', 'Pulsar'] print(bikes) type(bikes) #Get the first item in a list first\_bike = bikes[0] second\_bike = bikes[1] bikes[3] # Traceback (most recent call last): # File "<stdin>", line 1, in <module> #IndexError: list index out of range bikes[1] = 'bmw' last\_bike = bikes[-1] # total number of items in list len(bikes) # bikes = ['Apache', 'Suzuki', 'Pulsar']  # Iterate through every item in the list for bike in bikes: print("This is bike:",bike) |  |
|  | bikes = [] len(bikes) bikes.append('Apache') bikes.append('Suzuki') bikes.append('Pulsar') bikes.append(9) len(bikes) type(bikes[3]) | Adding items to a list |
|  | bikes.remove('Apache') print(bikes) len(bikes) | Removing items to a list with item name |
|  | bikes.pop(1) print(bikes) len(bikes) | Removing items to a list with item index number |
|  | aws\_topics = ['ec2', 's3', 'rds', 'lambda'] aws\_topics[<start\_index>:<nth\_index - 1>] first\_two = aws\_topics[0:2] first\_two = aws\_topics[:2] print(first\_two) aws\_topics\_copy = aws\_topics[:] >>> aws\_topics[:] ['ec2', 's3', 'rds', 'lambda'] >>> aws\_topics[0:1]  ['ec2'] >>> aws\_topics[0:2] ['ec2', 's3'] >>> aws\_topics[:2] ['ec2', 's3'] >>> aws\_topics[1:3] ['s3', 'rds'] | Slicing a list |
|  | t = ('a', 'b', 'c', 'd', 'e') | Tuples are immutable |
|  | >>> t = tuple() >>> print(t) >>> t = ('a', 'b', 'c', 'd', 'e') >>> print(t[0]) >>> print(t[1:3]) |  |
|  | s3buckets = {'name': 'mys3bucket', 'numOfObj': 10} type(s3buckets) print("The S3 Bucket name is " + s3buckets['name']) print("The numOfObj in S3 Bucket " + s3buckets['numOfObj']) print("The numOfObj in S3 Bucket " + str(s3buckets['numOfObj'])) print(len(s3buckets)) | Dictionaries in Python |
|  | s3buckets['size'] = 0 len(s3buckets) print(s3bucket) | Adding a new **key: value** pair |
|  | removed\_value = s3buckets.pop('size') print(s3buckets) | Removing a new **key: value** pair |
|  | >>> d = {'name':'mys3bucket', 'numOfObj':10, 'totalSize':200} >>> type(d) >>> list\_d = d.items() >>> type(list\_d) >>> print(list\_d) | Dictionaries have a method called items that returns a list of tuples, where each tuple is a key-value pair |
|  | # Function Definition def greet\_user(username): print("Hello, " + username + "!") # Function call greet\_user("AWS") greet\_user("Python") greet\_user("DevOps") | Function example |
|  | # Function Defination def add\_numbers(x, y): print("Inside add\_numbers function") return x + y sum1 = add\_numbers(3, 5) sum2 = add\_numbers(10, 15) print("Value of sum1",sum1) print("Value of sum2",sum2) |  |
|  | import math math.factorial(4) |  |
|  | >>> import math >>> print(math) <module 'math' (built-in)> >>> print(math.pi) 3.14159265359 |  |
|  | import json x = '{ "name":"John", "age":30, "city":"New York"}' print("Type of x is ",type(x)) y = json.loads(x) print("Type of y is ",type(y)) print(y["age"] | **json.loads()** |
|  | import json a = {"GroupName": "default","GroupId": "sg-32ef414c"} print("Type of a is ",type(a)) b = json.dumps(a) print("Type of b is ",type(b)) print (b) | **json.dumps()** |
|  | import sys # total arguments print(type(sys.argv)) n = len(sys.argv) print("Total arguments passed:", n) # Arguments passed print("\nName of Python script:", sys.argv[0]) # print("\nArguments passed:", end = " ") print("\nArguments passed:", sys.argv ) for i in range(1, n): print(sys.argv[i]) # print(sys.argv[i], end = " ") # Addition of numbers sum\_val = 0 # Using argparse module for i in range(1, n): sum\_val = sum\_val + int(sys.argv[i]) print("\n\nResult:", sum\_val) | It is a list of command line arguments. **len(sys.argv)** provides the number of command line arguments. **sys.argv[0]** is the name of the current Python script. |
|  | **BOTO 3** |  |
|  | python3 -c "import boto3" sudo python3 -m pip install boto3 | **pip** is the package installer for Python |
|  | #!/usr/bin/env python import boto3 #Create a client object connection with ec2 service ec2 = boto3.client('ec2',region\_name='us-west-2') # Execute a function call to describe instances present in aws account ec2\_dict=ec2.describe\_instances() print("ec2\_dict type is",type(ec2\_dict)) print("ec2\_dict is",ec2\_dict) | **Python boto3 List EC2 Instances** |
|  | JSONlint- format your jason |  |
| 1 | import boto3 ec2 = boto3.client('ec2',region\_name='us-west-2') ec2\_dict=ec2.describe\_instances() print("ec2\_dict type is",type(ec2\_dict)) print("ec2\_dict is",ec2\_dict) reservations\_list=ec2\_dict['Reservations'] print(reservations\_list) print(type(reservations\_list)) print(len(reservations\_list)) # print("reservations\_list is",reservations\_list, type(reservations\_list),len(reservations\_list)) print("-----------------------------------------") InstanceIdsList=[] for instances in reservations\_list:  # print("instances",instances,type(instances))  print("instance is of type",type(instances))  instance\_id=instances['Instances'][0]['InstanceId']  instance\_state=instances['Instances'][0]['State']['Name']  print("instance\_id is",instance\_id)  print("instance\_state is",instance\_state) if instance\_state == 'stopped':  print(instance\_id ,"will be started")  InstanceIdsList.append(instance\_id) # Check whether list is empty if not InstanceIdsList:  print("InstanceIdsList is empty, cannot perform start operation") else: print("Starting all the instances with instance ids: ",InstanceIdsList) ec2.start\_instances(InstanceIds=InstanceIdsList) | Get list of EC2 instances that are in 'stopped' state and start them |
|  | import boto3 ec2 = boto3.client('ec2',region\_name='us-west-2') ec2\_dict=ec2.describe\_instances() print("ec2\_dict type is",type(ec2\_dict)) print("ec2\_dict is",ec2\_dict) reservations\_list=ec2\_dict['Reservations'] print(reservations\_list) print(type(reservations\_list)) print(len(reservations\_list)) # print("reservations\_list is",reservations\_list, type(reservations\_list),len(reservations\_list)) print("-----------------------------------------") InstanceIdsList=[] for instances in reservations\_list:  # print("instances",instances,type(instances))  print("instance is of type",type(instances))  instance\_id=instances['Instances'][0]['InstanceId']  instance\_state=instances['Instances'][0]['State']['Name']  print("instance\_id is",instance\_id)  print("instance\_state is",instance\_state) if instance\_state == '**runnig**':  print(instance\_id ,"will be **stopped**")  InstanceIdsList.append(instance\_id) # Check whether list is empty if not InstanceIdsList:  print("InstanceIdsList is empty, cannot perform start operation") else: print("Starting all the instances with instance ids: ",InstanceIdsList) ec2.**stop**\_instances(InstanceIds=InstanceIdsList) | Get list of EC2 instances that are in 'stopped' state and start them |
|  | # This function is written to get list of EC2 instances using filter by tags as "env": "dev" and that are in 'stopped' state and start them.  # Here, first find all the instance id as per Tags and once a Python List of Instance ids is created, this can be passed to ec2.start\_instances(InstanceIds=InstanceIdsList)  import boto3  ec2 = boto3.client('ec2',region\_name="ap-south-1")  ec2\_dict=ec2.describe\_instances()  print("ec2\_dict type is",type(ec2\_dict))  print("ec2\_dict is",ec2\_dict)  reservations\_list=ec2\_dict['Reservations']  print(reservations\_list)  print(type(reservations\_list))  print(len(reservations\_list))  # print("reservations\_list is",reservations\_list, type(reservations\_list),len(reservations\_list))  print("-----------------------------------------")  InstanceIdsList=[]  for instances in reservations\_list:  # print("instances",instances,type(instances))  print("instance is of type",type(instances))  instance\_id=instances['Instances'][0]['InstanceId']  instance\_state=instances['Instances'][0]['State']['Name']  if not instances['Instances'][0]['Tags']:  print("Tags Key is not available for this instance")  else:  tags\_list=instances['Instances'][0]['Tags']  print("tags\_list is - ", tags\_list)  print("instance\_id is",instance\_id)  print("instance\_state is",instance\_state)  if not tags\_list:  print("Tags not available for this instance")  else:  for tags in tags\_list:  print(type(tags))  if tags['Key'] == 'env' and tags['Value'] == 'dev':  if instance\_state == 'stopped':  print(instance\_id ,"will be started")  InstanceIdsList.append(instance\_id)  if not InstanceIdsList:  print("InstanceIdsList is empty, cannot perform start operation")  else:  print("Starting all the instances with instance ids: ",InstanceIdsList)  ec2.start\_instances(InstanceIds=InstanceIdsList) | **Start ec2 having tag** |
|  | import boto3 #Create a client object connection with ec2 service s3 = boto3.client('s3') # Execute a function call to get list of S3 buckets in aws account bucket\_dict=s3.list\_buckets() print("Type of bucket\_dict is",type(bucket\_dict)) bucket\_list=bucket\_dict['Buckets'] print("Type of bucket\_list is",type(bucket\_list)) for bucket\_info in bucket\_list:  print("Type of bucket\_info is",type(bucket\_info))  print("Bucket Name is ",bucket\_info['Name'])  print("length of bucket\_list is",len(bucket\_list)) | **Python boto3 List S3 Buckets Information** |
|  | import os,json,boto3 s3 = boto3.client("s3") sns = boto3.client("sns") def lambda\_handler(event,context):  print(event)  source\_bucket = event['Records'][0]['s3']['bucket']['name']  aws\_region = event['Records'][0]['awsRegion']  key\_val = event['Records'][0]['s3']['object']['key']  size\_val = event['Records'][0]['s3']['object']['size']/1024  ipAddress = event['Records'][0]['requestParameters']['sourceIPAddress']  event\_time = event['Records'][0]['eventTime']  message = "Hi, \n The Event time is : " + event\_time + "Hi, \nYou are receiving this email because you are subscribed to this S3event. \nThe Source Bucket is : " + source\_bucket + "\nThe AWS Region is :" + aws\_region + "\nThe Uploaded Filename is : " + key\_val + " having Size : " + str(size\_val) + " KB" + "\nThe Object is upload from IP Address: " + ipAddress  #Below are the variables for copy\_object function parameters #Provide below the target bucket name where your object needs to be copied backupBucket = os.environ['BACKUP\_BUCKET\_NAME'] snsArn = os.environ['SNS\_TOPIC\_ARN'] emailSubject = "S3EventTrigger-Notification" copy\_source = {'Bucket' : source\_bucket, 'Key' : key\_val} sns\_response = sns.publish(TopicArn=snsArn,Message=message,Subject=emailSubject) print(sns\_response) try: print("Copying the object from source to destination") s3.copy\_object(Bucket=backupBucket,Key=key\_val, CopySource=copy\_source) except Exception as e: print(e) print("Error getting object") raise e | Lambda SNS backup folder |
|  | { "Version": "2012-10-17", "Statement": [ { "Sid": "SnsPolicy", "Effect": "Allow", "Action": "sns:Publish", "Resource": "**arn:aws:sns:us-east-1:ACCOUNT\_ID:SNSDemoTopic**" }, { "Sid": "S3Access", "Action": "s3:\*", "Effect": "Allow", "Resource": "**arn:aws:s3:::\*, arn:aws:s3:::\*, arn:aws:s3:::\*,** **arn:aws:s3:::\***"  #"Resource": "**arn:aws:s3:::s3-bucket”,** "**arn:aws:s3:::s3-bucket/\*”,** "**arn:aws:s3:::s3-bucket-backup”,** “**arn:aws:s3:::s3-bucket-backup/\*** " } ] } | Add the below inline policy and add to lambda role, below policy should have only Publish access to specific topic and S3 Permissions |
|  | https://docs.aws.amazon.com/AmazonS3/.. |  |
|  |  |  |
|  |  |  |
|  | GIT |  |
|  | git --version mkdir git-practical cd git-practical git init ls -altR # List all the files recursively git config --list |  |
|  | git config --local user.name "TestUser" git config --local user.email "testuser@example.com" |  |
|  | git config --global user.email "testuser@example.com" git config --global user.name "TestUser" | Global user |
|  | git config user.name git config user.email | To get the specific config value that is currently set |
|  | ls -al echo 'This is Repo Created for Devops Demo' >> file.txt git status | Working stage |
|  | git add file.txt git status | Staging state |
|  | git commit -m "changes made in the particular file" git log | commit |
|  | git diff --staged | compares your staged changes to your last commit: |
|  | git diff | make some changes in the file that is already staged, below command will show the differences |
|  | git log git log --oneline | View Commit History and view commit details in one line |
|  | git log -p -2 | To view the only last two entries |
|  | git log --stat | common options for git log |
|  | echo "this is testing file" >> newfile.txt git add newfile.txt git commit -m '1st commit' git log -p -2 git commit --amend -m "an updated commit message" git log -p -2 | To change the commit message of an existing commit |
|  | echo "adding some content to unstage changes" >> newfile.txt git add \* git reset newfile.txt | Unstaging a Staged File |
|  | ssh-keygen -t rsa -C "<GITHUB\_EMAIL\_ID>@gmail.com" | Generating an SSH Key, Use the github sign-in email address in the below command. |
|  | Add the Public Key file content into your Github Account Settings under: Settings > SSH and GPG keys. > New SSH key > Paste the Public Key Content > Save |  |
|  | ssh -T [git@github.com](mailto:git@github.com) or  ssh -i <PRIVATE\_KEY\_PATH> -T git@github.com | Verify SSH authentication |
|  | git remote add origin git@github.com:<GITHUB\_USERNAME>/git-practical.git | Using git remote add command allows us to associate a remote repository |
|  | git remote -v | -v parameter (verbose) will display the full URL of the remote repository for each remote name listed. |
|  | git add <FILENAME> git status git commit -m "Message for the commit" | To have multiple commits created, add or modify some files -> commit and push it to GitHub |
|  | git pull origin master | f there are any changes made in the Remote Repository, to have those changes present in Local Repository, use below command: |
|  | git push origin master | Push changes in GitHub |
|  | git show e4b71efa7f76c0fc0875e0562d5fb6d7dadbff9c:newfile.txt | To get or display the content of the file as per particular commit. |
|  | git remote -v origin git@hostname:USERNAME/REPOSITORY.git (fetch) origin git@hostname:USERNAME/REPOSITORY.git (push)  #list your existing remotes in order to get the name of the remote you want to change | Switching remote URLs from SSH to HTTPS |
|  | git remote set-url origin https://hostname/USERNAME/REPOSITORY.git | Change your remote's URL from SSH to HTTPS with the git remote set-url command. |
|  | git clone git@github.com:<GITHUB\_USERNAME>/<REPO\_NAME>.git git remote -v | Use below command to clone the Repository with SSH URL. |
|  | git remote set-url origin git@github.com:USERNAME/REPOSITORY.git | Change your remote's URL from HTTPS to SSH with the git remote set-url command. |
|  | git push -u remote-name branch-name git push remote-name branch-name | Send Changes to Remote |
|  | git pull remote-name branch-name | Receive Changes from Remote |
|  | git clone <GITHUB\_URL> | Clone a new Empty repository. |
|  | echo "Hello World-1a" >> file1.txt echo "Hello World-1b" >> file1.txt git add file1.txt git commit -m "added hello world-1 to file1.txt" echo "Hello World - 2nd commit" >> file1.txt git add file1.txt git commit -m "added 2nd hello world to file1.txt"  or  git commit -a -m "added 2nd hello world to file1.txt"  git log --all --decorate --graph |  |
|  | alias mygraph="git log --all --decorate --graph" | if you dont want to type the above command everytime , create a alias for above command |
|  | alias ls -ltr .git/refs git branch | To see all the alias created in linux |
|  | git checkout main git branch | Checkout to your current branch |
|  | git branch # Create a new branch git branch feature\_1 git checkout feature\_1 OR git checkout -b feature\_1 | Create a new branch using below command: |
|  | git checkout main cat .git/HEAD git checkout feature\_1 cat .git/HEAD git branch mygraph | Since there are multiple branches now, i.e main and feature branches, git knows about the current branch using HEAD |
|  | git show <commit-id> | To View all changes using commit-id |
|  | git checkout main cat file1.txt git merge feature\_1 mygraph cat file1.txt |  |
|  | git reset --merge ORIG\_HEAD mygraph cat file1.txt git merge feature\_1 mygraph | undo a recent merge |
|  | git branch --merged | verify the branches that are merged |
|  | git branch -d feature\_1 | deleting the branch |
|  |  | 3way merge |
|  |  | Pull request  reviewers |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |