Task-1: Exploring AWS cloudshell

* Open cloud shell
* aws --version
* aws s3 ls
* From actions split into columns
* Download and upload file
* cat filename.py.
* python filename.py → returns bucket name.
* copying file from cloud shell to bucket
  → aws s3 cp filename.py s3://bucketname

Task-2: Creating an AWS cloud9 instance.

* search cloud9 instance.
* select → Create environment
* Enter name, EC2 instance, additional instance.
* In network settings choose ssh.
* Create & Open

Task 3: Exploring the AWS cloud9 IDE.

* Observe IDE user interface.
* aws s3 ls
* aws s3 cp s3://bucket name / filename.py
  → list of buckets is shown
* Open the file and run
* sudo pip3 install boto3
* run again
* create new HTML file
* choose region for AWS explorer
* upload html file

Task-1 : Connecting to the AWS cloud9 IDE and configuring environment

* Search cloud9
* cloud9 instance → open
* sudo pip install boto3
* Download files
  wget https://
* unzip code.zip
* aws --version

Task-2: Creating S3 bucket

* aws s3 api create bucket --bucket bucket-name --region.
* S3 → access & permissions.
* Deselect block all public access
  - Block public access to buckets & objects granted through new access control files
  - Block public access to buckets & objects granted through any access control files
  - Block public and cross count access to bucket & objects.
  - save.

Task-3: Setting bucket policy

* aws cloud9
* instance
* new file → filename.json → save
* Paste the given code into json file
* Replace bucket name & ip address → save
* in python directory open permissions
* Replace bucket name → save
* cd python3
  python3 permissions.py

Task-4: Uploading objects to bucket to create website.

* aws s3 cp .. /resources/website s3://bucket name/.....

Task-5: Testing access to website

* load website
* bucket name → objects
* index.html → copy object url
* paste in browser
* Try to access from outside network.
* click login to check if it is working.

Task-6: Analyzing code

* cloud 9
* resources → website
* index.html
* open js files and analyse code.

Task 1: Preparing lab.

* Cloud9 → instances → open
  wget https://..
* unzip code.zip
* chmod +x ./resources/setup.sh && ./resources/setup.sh
* aws --version.
* pip show boto3

Task-2: Creating a DynamoDB tables

* Open dashboard AWS Cloud9
* search Dynamo DB
* from menu click tables
* Back to Cloud9
* Python → Create table.py
* replace place holder with given data → save
* Table = DDB.create_table(**params)
* cd python.3
* python3 create_table.py
* aws dynamodb list_tables --region
* verify tables

Task3: Working with Dynamo DB data

* resources → not.on.existing-product.json
* insert given code
* Explore table items it should be reflected
* replace product name → save
* new item will be added
* replace product id → save

Task-4: Adding and modifying a simple item by using the SDK

* python3 directory → conditional.put.py.
* replace place holder → save
* python 3 conditional-put.py
* modify product id and run again in terminal

* change product name and run again.

Task-5: Adding multiple items.

* Item explorer → run
* Delete all items
* resources → test.json
* python3 → test-batch-put.py
  → replace all place holder → save
* run the file → python3 test-batch-put.py
* check item explorer
* with table.batch_writer() as batch: replace & run again
* modify python3 → batch-put.py & run
* check item explorer

Task-6: Querrying the table

* python3 → get-all-item.py → replace all placeholders → save → run
* get-one-item.py → replace placeholders → save → run

Task-7: Adding a global secondary index to the table

* add-gsi.py → replace place holder → save → run
* wait for status to become active.
* with-filter.py → replace place holder → save → run.

Task 1 : preparing

* Cloud 9 instance → Open
* wget https://
* unzip code.zip
* chmod +x resources/setup.sh && resources/setup.sh
* enter ip address
* aws --version.
* pip show boto3
* load website
* S3 console → Objects → index.html → copy & paste url.
* View JS console

Task 2: Creating first API endpoint

* Python 3 → create_products_api.py → replace placeholder → save → run
* search API gateway
* Product API
* choose GET → study data flow → test

Task-3 : Creating second API endpoint

* python3 → create_an_offer_api → replace placeholder → save → run
* /product/{}/an_offer choose GET → Test

Task 4 : Creating third API endpoint

* create_products_api.py & create_report_api.py
* analyse both & replace placeholders → save → run
* Resource / create_report → Post → test

Task 5 : Deploying the API.

* select root → Deploy API → Stage: Prod → deploy → copy url

Task 6 : updating the website to use API's

* resources / website / config.js
* replace with given code → save
* python 3 → update_config.py → replace placeholder → save → run

* open website url
* view javascript console.

Task 1 : prepare

* cloud 9 : instance → open
* wget https://
* unzip code.zip
* chmod +x resources/script.sh && resources/script.sh
* enter ip address
* pip show boto3
* S3 → index.html → copy url → run
* DynamoDB → Tables → choose table → explore table items
* API Gateway → test for each method
* API Gateway → stages → prod → copy url
* resources/website/config.js → replace null with copied url → save
* python3 → update-config.py → replace placeholder → save → run
* load website again

Task 2 : Creating a lamba function

* python3 → get-all-product-code.py → replace placeholder → save → run
* modify setting in code & run now revert back comment the last line
* save
* IAM → Roles → Lamba access to DynamoDB
* copy Role ARN
* Python3 → get-all-products-wrapper.py → replace placeholders → save
* verify python3 directory
* zip get-all-product-code.zip
  get-all-products-cgi.py
  aws s3 ls
  aws s3 cp get-all-products-code.zip s3://bucketname
* Run get-all-products.wrapper.py.
* Browse to lamba → Test → save → test.
* Test → Configure test Event → create new event → Enter given details → save → test.

Task 3: configuring REST API to revoke lambda function.

* API Gateway → Product API → /product/GET → Test
* Integration request → given necessary details → save → Test again.
* Re enable CORS on/products → make given changes to allow access → save
* test again
* /on-offer/GET → intepretation request → give necessary details → save.
* Enable CORS → make given changes to allow access control → save test
* /products/on-offer/GET → add mapping template code → save → test.
* select root → display API → prod → deploy

Task 4: creating lamba function for report request in function

* run create_report_code.py → connect last line → save
* python3 → create-report.wrapper.py → replace placeholder → save
* zip create_code.zip

  Create-report.code.zip
* aws s3 cp create_report_code.zip s3://bucket name
* run create-report.wrapper.py
* Test create-report.wrapper.py → save → test

Task 5: configure REST API to invoke lamba function to handle report

* create-report/POST → test → Integration request → give necessary details → save, select root deploy API

Task 6: Testing the integration

* load website
* Browse Pastries → view all
* modify price of one product → refresh page & changes are reflected.

Task 1: preparing

* cloud9 instance → open
* wget https://..
* unzip code.zip
* chmod +x /resources/setup.sh && /resources/setup.sh
* aws --version
* pip show boto3

Task 2: Analyze existing app instructions

* EC2 instances → appserver Node → copy IPv4 → paste in browser
* choose list of suppliers → Add new suppliers → submit.

Task 3: migrating the application to Docker container

* mkdir container cd containers
* mkdir node.app cd node.app
* mv - environment/resources/..
  - environment/container/node-app.
* create Docker file → cd ../environment/container/node-app touch Dockerfile
* Paste given code into Docker file → save
* docker build --tage node-app
* docker images
* docker run -d --name node.app -p 3000:3000 code-app
* docker container ls
* url https://localhost:3000
* cloud9 instance → security → edit inbound rules make changes → save
* EC2 instance → copy IPv4 add of aws cloud9 instance
* docker PS
* docker exec -ti container id
* whoami → su node → env → exit → exit
* docker ps
* docker stop node-app.1 && docker rm node.app.1
* EC2 instance → copy IPv4 add of mysql server Node
* docker run -d --name node-app-1 -p 3000:3000 APPDB_HOST="ip add"

Task 4: migrate mySQL database
* Create new file → paste give code
* cd to change directory
* enter password.
* Open my-sql.sql make give changes → save
* mkdir mysql → cd mysql
* touch dockerfile
* mv ~/my.sql . sql
* Open empty docker file paste code
* docker rm -f $(docker images ls -s -q)
* sudo docker image -f && sudo docker container f - f
* docker build -- tag mysql-server
* docker images

Task 5: Testing mySQL container
* docker inspect network.
* run docker with new ip address
* docker ps → test application

Task 6: Adding the Docker image to ECR
* copy Acc num
* Authorize Aws cloud 9 Docker client
* aws ecr create-repository --name node-app
* Docker tag using registry id
* Docker image
* Push Docker image to ELR repository