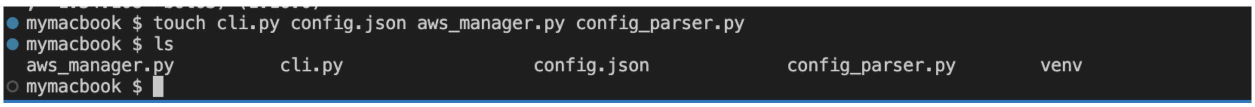
Nandita - Solace - IAC - Documentation

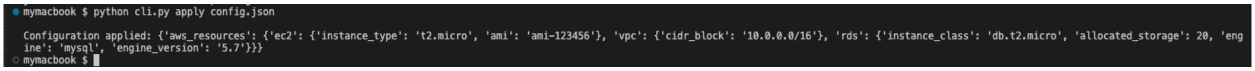
Setup:
-Initiated a virtual python venv
-Installing required libraries like click to apply cli commands
-Created a basic project structure



```
● mymacbook $ touch cli.py config.json aws_manager.py config_parser.py
● mymacbook $ ls
  aws_manager.py          cli.py                  config.json             config_parser.py        venv
○ mymacbook $ 
```

-I am implementing CLI using cli.py file with required imports like json, click
-Copied boiler plate code in my 3 files **cli.py, config_parser.py and aws_manager.py** and **config.json** (I have left placeholders as required)

-Tested basic JSON parsing and successful



```
● mymacbook $ python cli.py apply config.json

  Configuration applied: {'aws_resources': {'ec2': {'instance_type': 't2.micro', 'ami': 'ami-123456'}, 'vpc': {'cidr_block': '10.0.0.0/16'}, 'rds': {'instance_class': 'db.t2.micro', 'allocated_storage': 20, 'eng
  ine': 'mysql', 'engine_version': '5.7'}}}
○ mymacbook $ 
```

This output indicates that my CLI is currently able to read the configuration from the JSON file and pass it through my system.

Assumptions:

- User has a working AWS account
- “aws configure” is configured successfully with key and secret
- Installed required python requirements.txt
- Not a production environment
- No state file as we are not performing destroy of AWS resources in the assignment
- Since this is not prod, we are only doing error handling rather implementing logging
- Since this is not prod, we are not considering security best practices
- Folder structures and file names evolves through the assignment development

Developing
In the nandita_aws_manager.py file, I would like to separate all my aws related function definitions. For example **defining** vpc, subnets, sg, ec2, rds and also a definition to apply our configurations.
The above will keep our cli.py main file neat and clear. As we are just importing the function definitions from nandita_aws_manager.py file

My approach to develop nandita_aws_manager.py file to apply configurations:

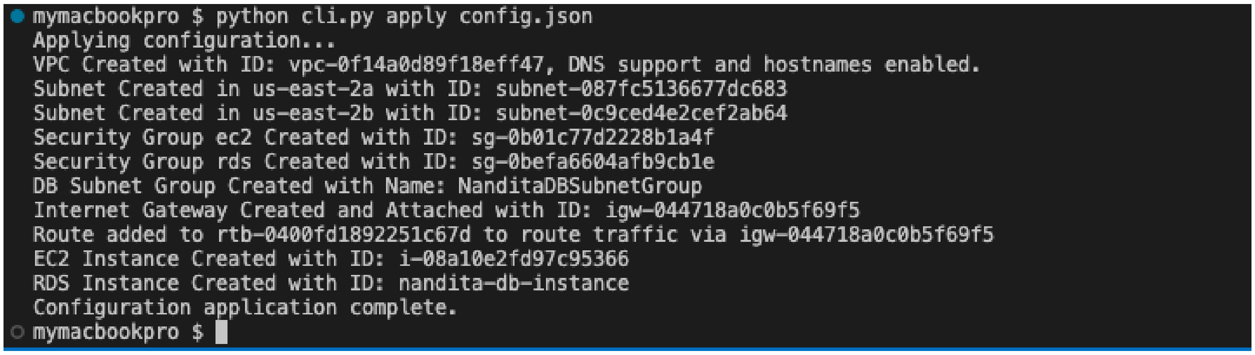
As mentioned earlier, write functions that reflects the sequence of operations that generally would happen in an AWS environment

- A function for creating VPC
- A function for creating subnet
- An individual function for rest of the networking elements like an Internet Gateway, Route Table Updates, Security Groups, DB Subnet Group
- Functions for EC2 and RDS creation

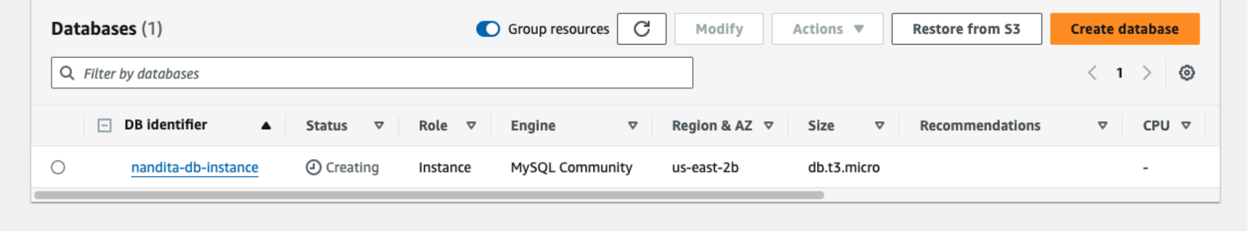
- Finally, my favorite function, apply_configuration to APPLY !

My approach to develop functions in nandita_aws_manager.py:

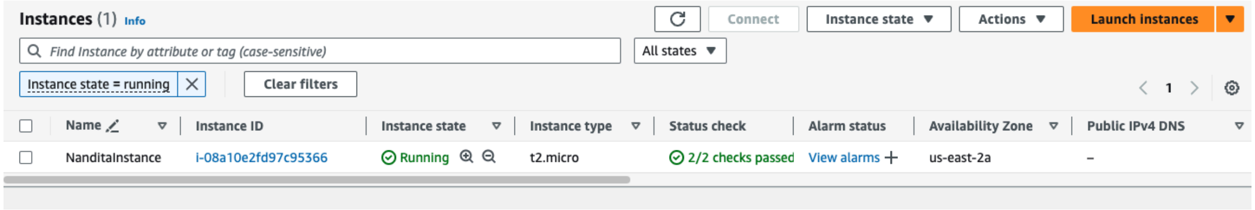
- Use individual functions for each resource specification for better code readability
- **Error handling**: I have added error handling for most error prone functions with the free tier, such as VPC creation function, db subnet group function and rds creation function, (will do more in phase 2)



```
mymacbookpro $ python cli.py apply config.json
Applying configuration...
VPC Created with ID: vpc-0f14a0d89f18eff47, DNS support and hostnames enabled.
Subnet Created in us-east-2a with ID: subnet-087fc5136677dc683
Subnet Created in us-east-2b with ID: subnet-0c9ced4e2cef2ab64
Security Group ec2 Created with ID: sg-0b01c77d2228b1a4f
Security Group rds Created with ID: sg-0befa6604afb9cb1e
DB Subnet Group Created with Name: NanditaDBSubnetGroup
Internet Gateway Created and Attached with ID: igw-044718a0c0b5f69f5
Route added to rtb-0400fd1892251c67d to route traffic via igw-044718a0c0b5f69f5
EC2 Instance Created with ID: i-08a10e2fd97c95366
RDS Instance Created with ID: nandita-db-instance
Configuration application complete.
mymacbookpro $
```



	DB identifier	Status	Role	Engine	Region & AZ	Size	Recommendations	CPU
○	nandita-db-instance	Creating	Instance	MySQL Community	us-east-2b	db.t3.micro		-



□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
□	NanditaInstance	i-08a10e2fd97c95366	Running	t2.micro	2/2 checks passed	View alarms +	us-east-2a	–

User can do modifications using config.json and apply configurations using cli.py file
Command: "python cli.py apply configuration"

Phase 1: We now have,

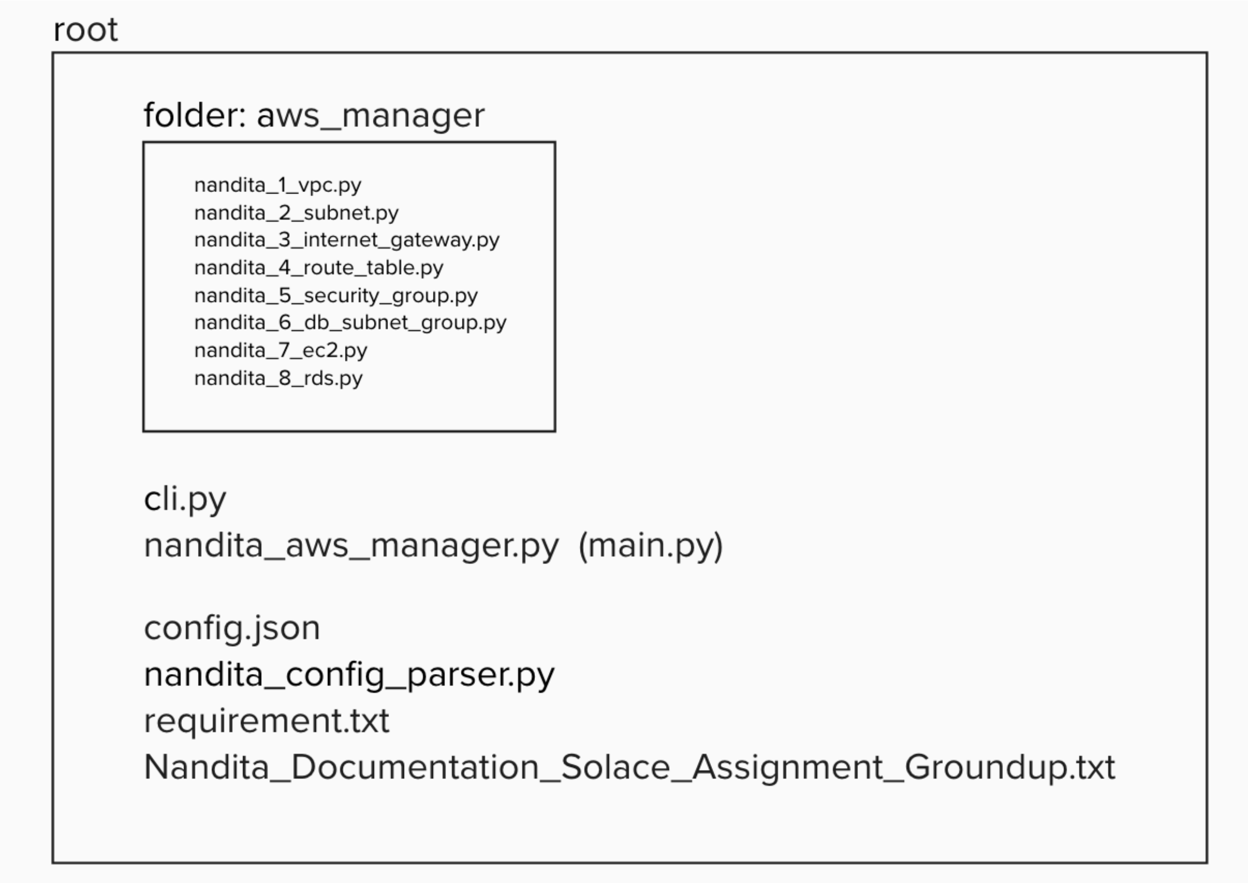
- config.json, where a user can specify resource specifications
- nandita_config_parser.py, a file that parses above json specifications
- nandita_aws_manager.py, a py file that interacts with aws by written functions
- cli.py that applies configuration and creates resources in aws

Phase 2: Since we have our cli.py successfully applying configurations, here I have focused on adding error messages to the rest of the functions. We can use the retry wrapper, but I think it would be an overkill. Focusing on clear error messages without adding retry logic will simplify the implementation and keep the error handling straightforward.

Phase 3:

- Code base reorganization by creating aws_manager folder to save modular functions
- Developing modular functions and importing required function to nandita_aws_manager.py
- Adding the draft documentation doc to github

Final Folder Structure:



root

folder: aws_manager

nandita_1_vpc.py
nandita_2_subnet.py
nandita_3_internet_gateway.py
nandita_4_route_table.py
nandita_5_security_group.py
nandita_6_db_subnet_group.py
nandita_7_ec2.py
nandita_8_rds.py

cli.py
nandita_aws_manager.py (main.py)

config.json
nandita_config_parser.py
requirement.txt
Nandita_Documentation_Solace_Assignment_Groundup.txt

Dev best practices followed in this application:

- **Function Naming**
  All my function names are descriptive and follow the Python naming convention of using lowercase with words separated by underscores (snake_case). This makes the code more readable and easier to understand.
- **Error handling**
  I have implemented error handling using try-except blocks in several functions, such as create_vpc, create_db_subnet_group, and create_rds_instance. This is a good practice as it helps catch and handle exceptions gracefully, preventing the program from crashing unexpectedly.
- **Modular code**
  Each function has a specific responsibility, making the code more modular and easier to

maintain. For example, create_vpc, create_subnets, create_security_group, and create_ec2_instance each handle a different aspect of the AWS infrastructure setup

- **Meaningful print statements**
  The print statements provide useful information about the actions being performed and the resources being created.
- **Consistent formatting**
  I have made the code follow a consistent formatting style, making it easier to read and understand. This includes proper indentation, spacing, and line breaks.
- **Descriptive variable names**
  The variable names used in the code are descriptive and self-explanatory
- **Handling the configuration**
  Our code handles configuration data from config.json, which is a good practice for separating configuration from the application logic. Also one of the specifications from our assignment objectives

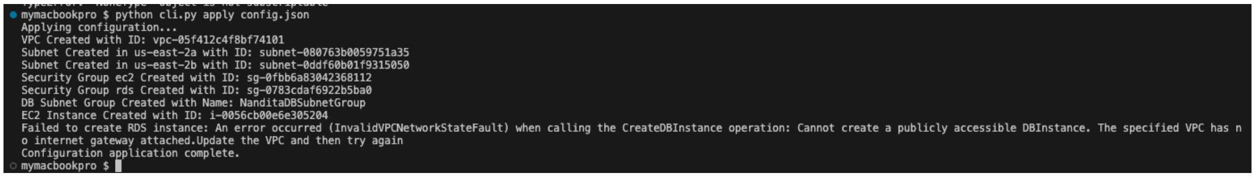
Challenges faced during app development:

Error 1: Errors on maximum vpc limits - reached

Error 2: Subnet related errors
botocore.errorfactory.DBSubnetGroupDoesNotCoverEnoughAZs: An error occurred (DBSubnetGroupDoesNotCoverEnoughAZs) when calling the CreateDBSubnetGroup operation: The DB subnet group doesn't meet Availability Zone (AZ) coverage requirement. Current AZ coverage: us-east-2b. Add subnets to cover at least 2 AZs.

Fixed by changing the code to add subnets to cover at least 2 AZs

Error 3:

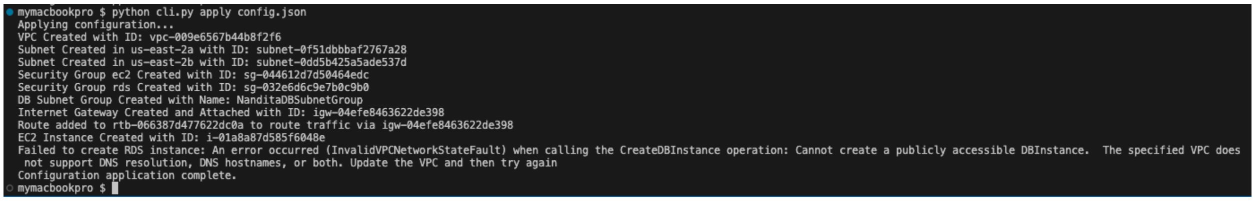


```
mymacbookpro $ python cli.py apply config.json
Applying configuration...
VPC Created with ID: vpc-05f412c4f8bf74101
Subnet Created in us-east-2a with ID: subnet-080763b0059751a35
Subnet Created in us-east-2b with ID: subnet-0ddf60b01f9315050
Security Group ec2 Created with ID: sg-0fbb6a83042368112
Security Group rds Created with ID: sg-0783cdaf6922b5ba0
DB Subnet Group Created with Name: NanditaDBSubnetGroup
EC2 Instance Created with ID: i-0056cb00e6e305204
Failed to create RDS instance: An error occurred (InvalidVPCNetworkStateFault) when calling the CreateDBInstance operation: Cannot create a publicly accessible DBInstance. The specified VPC has n
o internet gateway attached.Update the VPC and then try again
Configuration application complete.
mymacbookpro $ 
```

“Failed to create RDS instance: An error occurred (InvalidVPCNetworkStateFault) when calling the CreateDBInstance operation: Cannot create a publicly accessible DBInstance. The specified VPC has no internet gateway attached.Update the VPC and then try again
Configuration application complete.”

I have figured out this is because “rds instance could not be created because the VPC in which it is supposed to be deployed does not have an Internet Gateway (igw) attached”, so writing required functions to create them.

Error 4:



```
mymacbookpro $ python cli.py apply config.json
Applying configuration...
VPC Created with ID: vpc-009e6567b44b8f2f6
Subnet Created in us-east-2a with ID: subnet-0f51dbbbaf2767a28
Subnet Created in us-east-2b with ID: subnet-0dd5b425a5ade537d
Security Group ec2 Created with ID: sg-044612d7d50464edc
Security Group rds Created with ID: sg-032e6d6c9e7b0c9b0
DB Subnet Group Created with Name: NanditaDBSubnetGroup
Internet Gateway Created and Attached with ID: igw-04efe8463622de398
Route added to rtb-066387d477622dc0a to route traffic via igw-04efe8463622de398
EC2 Instance Created with ID: i-01a8a87d585f6048e
Failed to create RDS instance: An error occurred (InvalidVPCNetworkStateFault) when calling the CreateDBInstance operation: Cannot create a publicly accessible DBInstance.  The specified VPC does
 not support DNS resolution, DNS hostnames, or both. Update the VPC and then try again
Configuration application complete.
mymacbookpro $ 
```

"Failed to create RDS instance: An error occurred (InvalidVPCNetworkStateFault) when calling the CreateDBInstance operation: Cannot create a publicly accessible DBInstance. The specified VPC does not support DNS resolution, DNS hostnames, or both. Update the VPC and then try again
"

Resolved by adding,
EnableDnsHostnames={'Value': True}
EnableDnsSupport={'Value': True}

In the script while creating vpc (create_vpc function)

Docs and resources used:
https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
https://github.com/aws-samples
https://stackoverflow.com/
https://forums.aws.amazon.com/
And got few boiler plate function definitions, validations from various online tools
For example,
What are the important elements in boto3 client resource configurations?
What are the important lists of resource keywords while scripting with boto3 to build aws vpc?
A boilerplate code to create 2 subnets in a AZ and rdb subnet group using boto3 python?
What are common AWS identifiers?

My future considerations would be:

- Writing delete/destroy functions for rds, ec2 followed by vpc, db sg
- Create a state.json variable file to manager identifier and use it for destroy (similar to state.tf in terraform)
- For prod, improve Error Handling (I would do it by adding more robust error handling and logging)
- Improving security considerations, to manage AWS sensitive info efficiently (Right now, everything is stored in file. Therefore, I would secure the access to this file appropriately)