Boto3 S3 Learning Document

Objective

This document captures my learnings from implementing a Python script using Boto3 to fetch VPC and EC2 details in AWS's us-east-1 region , saving them in separate JSON files (**vpc_details_us-east-1.json and ec2_details_us-east-1.json).**

Key Learnings

1. **Boto3 SDK Overview**:
    - **What is Boto3?** AWS's Python SDK for interacting with services like EC2 and VPC.
    - **Authentication**: Relies on ~/.aws/credentials, environment variables, or IAM roles. Ensured credentials were set up.
2. **Components**:

- **Client**: Direct API calls for fine-grained control.
- **Resource**: Object-oriented abstraction for simpler coding.
- **Paginator**: Manages large API responses.
- **Waiter**: Polls for resource states.

🔷 S3 Client

The **S3 client** provides low-level access to all S3 operations. It's the most direct way to interact with S3, supporting methods like upload_file, download_file, list_objects_v2, etc

⇒ Use when you need full control and want to work with the exact AWS API operations.

🔷 S3 Paginators

Paginators help you handle large sets of results that are returned in pages (e.g., when listing thousands of objects in a bucket). They abstract the logic needed to paginate through responses.

⇒ Use when dealing with long lists of items (like list_objects_v2) that exceed response limits.

🔷 S3 Waiters

Waiters poll S3 until a resource reaches a desired state (e.g., waiting until a bucket exists after creation).

⇒ Use when you want to ensure a resource is ready before proceeding with the next action.

🔷 S3 Resources

Resources are higher-level abstractions over the client. The boto3.resource('s3') interface is more Pythonic and object-oriented, offering convenience methods like Bucket() or Object() .

⇒ Use when you prefer a cleaner, object-based way to interact with S3.