AWS_CloudOps_Early_Study_Guide

Sunday, February 15, 2026 11:52 PM

AWS CloudOps Early Study Guide

What is an API?

An API (Application Programming Interface) is a set of rules that allows two software programs to communicate with each other. It acts as an intermediary, delivering a request from one application to another and then returning the response.

Restaurant Analogy

- The Customer (You): The user or application that wants something (like a weather update).
- The Kitchen (Server): The system that contains the data or performs the work.
- The Menu (Documentation): Lists what you are allowed to request and required inputs.
- The Waiter (API): Takes your request to the system and returns the response.

What is Boto3?

Boto3 is the AWS SDK for Python. It allows Python code to interact with AWS services using API calls over HTTPS.

Client vs Resource

client(): Low-level interface mapping directly to AWS APIs. resource(): Higher-level, object-oriented abstraction.

```
import boto3

client = boto3.client('s3')
response = client.list_buckets()

for b in response["Buckets"]:
    print(b["Name"])
```

Amazon S3 Core Concepts

- Bucket: Top-level container for storage.

- Object: File stored inside a bucket.
- Key: The object's name/path.
- Region: Geographic location of the bucket.
- IAM: Controls access permissions.

IAM (Identity and Access Management)

IAM controls who can access AWS services and what actions they can perform. Common S3 permissions include: - s3:ListAllMyBuckets - s3:CreateBucket - s3:PutObject

Imperative vs Declarative

Imperative (Boto3): Step-by-step instructions executed immediately. Declarative (Terraform): Define desired state; tool ensures environment matches.

2-Week Practice Roadmap

- Week 1: S3 operations (list, create, upload, delete).
- Week 1: IAM permission testing and debugging.
- Week 2: Lambda integration with S3.
- Week 2: Deploy with SAM and connect API Gateway.

AWS CloudOps Study Guide – Version 2

What is an API?

An API (Application Programming Interface) is a set of rules that allows two software programs to communicate with each other. It acts as an intermediary, delivering a request from one application to another and then returning the response.

Restaurant Analogy

- Customer: The user or application making a request.
- Kitchen: The system performing the work.
- Menu: Documentation explaining allowed requests.
- Waiter: The API carrying requests and returning responses.

Boto3 and S3 Basics

Boto3 is the AWS SDK for Python. It allows Python applications to interact with AWS services via API calls. S3 is object storage built around buckets and objects.

```
import boto3

client = boto3.client('s3')
response = client.list_buckets()

for b in response["Buckets"]:
    print(b["Name"])
```

Lambda → IAM Role → Secrets Manager → RDS → SQL

Architecture Flow: Client/API ↓ Lambda Function ↓ IAM Role ↓ Secrets Manager (Retrieve DB Password) ↓ RDS MySQL Database ↓ SQL Execution (SELECT, INSERT, etc.) Key Concepts: • Lambda assumes an IAM Role. • IAM Role grants permission to read secrets. • Secrets Manager securely stores database credentials. • pymysql connects Lambda to RDS. • SQL queries execute via cursor.execute().

Section 2 – DynamoDB (NoSQL)

DynamoDB is a fully managed NoSQL database service. It uses partition keys (and optional sort keys) instead of traditional relational schemas.

- Partition Key: Primary identifier for items.
- Scan: Reads entire table (expensive).
- Query: Reads by partition key (efficient).
- Schemaless: Attributes can vary by item.
- Fully managed and serverless.

```
import boto3

dynamodb = boto3.client("dynamodb")

dynamodb.put_item(
    TableName="ZipcodeWeather",
    Item={
        "Zipcode": {"S": "90210"},
        "TemperatureF": {"N": "62"}
    }
)
```

DynamoDB vs RDS: RDS → Relational, SQL, fixed schema, joins supported. DynamoDB → NoSQL, key-value/document, horizontally scalable, no joins.

AWS CloudOps Study Guide – Version 3

DynamoDB Consistency Model

DynamoDB supports two read consistency models: Eventually Consistent (default): • May not reflect the most recent write immediately. • Higher read throughput. • Typically consistent within milliseconds to a second. Strongly Consistent: • Guarantees the latest write is returned. • Lower throughput than eventual consistency. • Used when applications require strict correctness. DynamoDB also supports ACID transactions for coordinated writes across multiple items.

Global Secondary Index (GSI) vs Local Secondary Index (LSI)

Primary keys control how data is partitioned. GSI (Global Secondary Index): • Different partition key than main table. • Allows flexible querying on alternate attributes. • Separate throughput capacity. LSI (Local Secondary Index): • Same partition key, different sort key. • Must be created at table creation. • Shares table throughput. Indexes allow querying without scanning the entire table.

Amazon SNS (Simple Notification Service)

SNS is a fully managed publish/subscribe messaging service. Pattern: One-to-Many Publisher → SNS Topic → Multiple Subscribers Subscribers can include: • Lambda functions • SQS queues • HTTP endpoints • Email/SMS SNS decouples microservices and enables fan-out messaging.

Amazon SQS (Simple Queue Service)

SQS is a fully managed message queue service. Pattern: One-to-One or Many-to-One Producer → SQS Queue → Consumer (Lambda, EC2, etc.) Key Concepts: • Messages stored durably until processed. • Visibility timeout prevents duplicate processing. • Supports Standard (at-least-once) and FIFO (exactly-once ordering).

```
import boto3

sqs = boto3.client("sqs", region_name="us-east-1")

response = sqs.send_message(
    QueueUrl="https://sqs.us-east-1.amazonaws.com/123456789/example-queue",
    MessageBody="Example message here"
)
```

Lambda Triggered by SQS

Flow: Producer ↓ SQS Queue ↓ Lambda (Event Source Mapping) ↓ CloudWatch Logs Lambda polls SQS automatically and processes messages in batches. If processing fails, message becomes visible again after visibility timeout.

CloudWatch Monitoring & Logs

CloudWatch provides logging and monitoring for AWS resources. Lambda automatically writes logs to: /aws/lambda/ Key Features: • Log groups and streams • Metrics and alarms • Log Insights (query logs) • Dashboards CloudWatch helps debug: • Errors • Timeouts • Message processing failures • Performance issues

Event-Driven Architecture Summary

Modern AWS serverless architecture commonly looks like: API Gateway ↓ Lambda ↓ DynamoDB / RDS OR Producer → SNS → Multiple Services Producer → SQS → Lambda Worker Key Design Goals: • Decoupling services • Scalability • Fault tolerance • Managed infrastructure • Observability (CloudWatch) Understanding these patterns is foundational for CloudOps and Cloud Engineering roles.

AWS CloudOps Study Guide – Version 4

Chapter 4 – AWS is an API Platform (GUI vs Code)

AWS is fundamentally an API-driven platform. Every action performed inside AWS — whether through the Console (GUI), CLI, Boto3, Terraform, CloudFormation, or CDK — ultimately results in an API call to AWS services. When you click "Launch EC2" in the AWS Console, the browser sends an HTTPS request to the EC2 API endpoint. The Console is simply a visual interface that makes those API calls for you. This means: • The GUI does not directly create servers. • The GUI calls the same API that code would call. • The AWS CLI calls the same API. • Boto3 calls the same API. • Terraform calls the same API. They are just different interfaces to the same backend system.

Why Corporations Use Code Instead of Clicking

- Clicking is not repeatable.
- Clicking is not version-controlled.
- Clicking does not scale well.
- Clicking causes configuration drift.
- Clicking cannot easily integrate into CI/CD pipelines.

In production environments, infrastructure is usually defined as code (Infrastructure as Code – IaC). Example production flow: Developer commits Terraform code → CI pipeline runs → Terraform calls AWS APIs → Infrastructure is created or updated. No one logs into the console to manually click resources in mature environments. AWS is not just a cloud provider — it is a collection of APIs that allow programmatic control of infrastructure.

Even Cleaner Mental Model

AD → Controls access to:

- File servers
- Domain resources
- Printers
- Windows login

IAM → Controls access to:

- S3 buckets
- EC2 instances
- DynamoDB tables
- Lambda functions

Same concept.
Different environment.

Key Difference Conceptually

JSON = Data
Python = Logic + Data
JSON says:

> "Here is information."

Python says:

> "Here is what to do with that information."

How They Work Together in APIs

When you call an API:

1. You send JSON
2. The server (written in Python, Go, Java, etc.) reads the JSON
3. The server runs logic
4. The server sends JSON back

JSON is the message.
Python is the brain.

SDK (Software Development Kit)

An SDK is:

> A library that helps you use an API easily from a specific programming language.

It wraps the API.

Think of It Like This

API = The official rulebook
JSON = The language of the messages
SDK = A translator that makes it easier for your language to follow the rulebook