SECTION 1

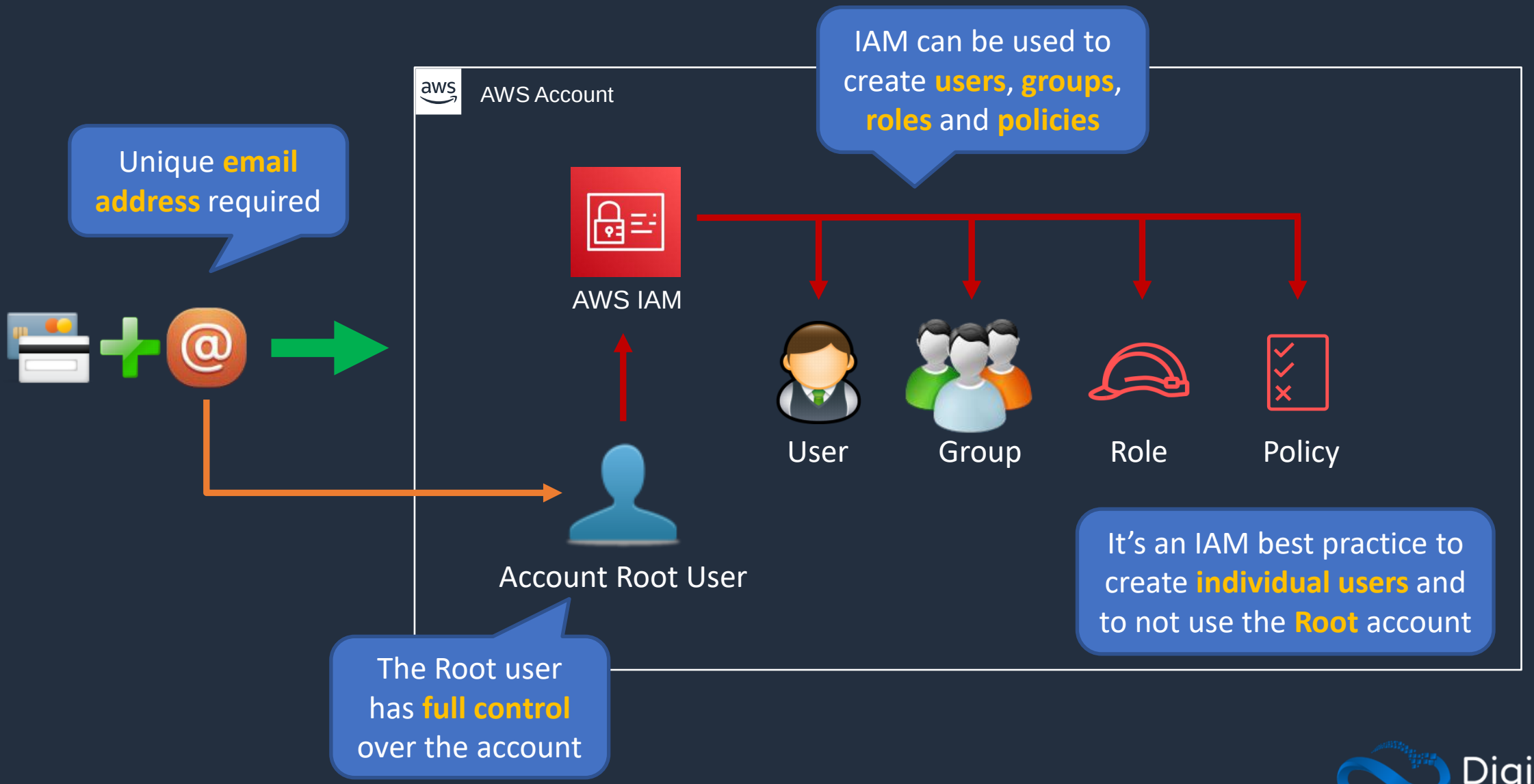
Python for the AWS Cloud

Python for the AWS Cloud

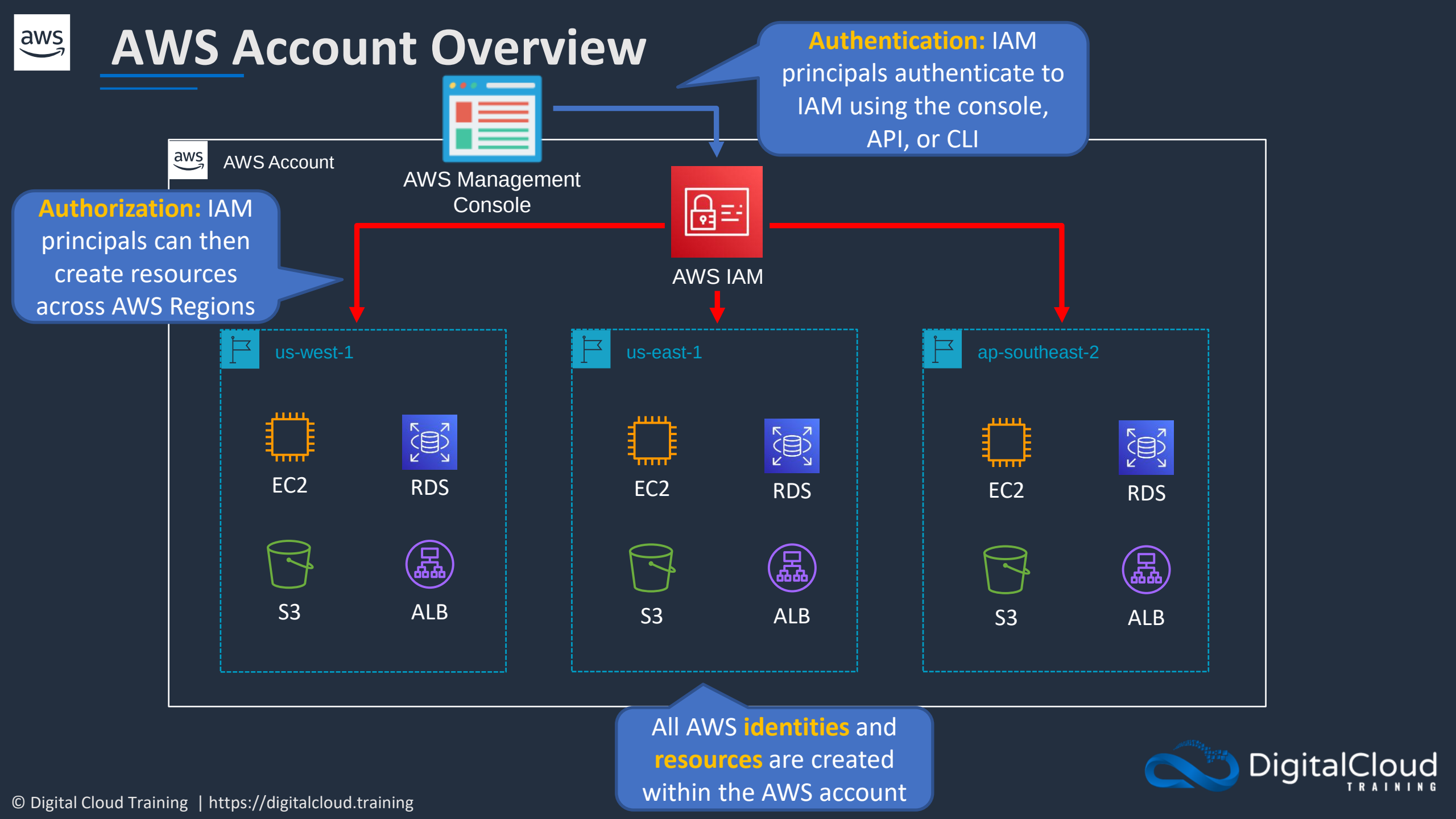
DigitalCloud TRAINING

AWS Account Overview

AWS Account Overview



AWS Account Overview



Create your AWS Free Tier Account

What you need…

Credit card for setting up the account and paying any bills

Unique email address for this account

Check if you can use a **dynamic alias** with an existing email address

john@gmail.com

john+ACCOUNT-ALIAS-1@gmail.com

john+ACCOUNT-ALIAS-2@gmail.com

AWS account name / alias

Phone to receive an **SMS** verification code

IDEs

IDE - Integrated Development Environment

What is an IDE?

- A software app that provides a set of tools for software development
  - A code editor
  - Debugger
  - Compiler
  - And other features that facilitate the development process

Local IDEs

- Installed on a local machine
- Requires manual setup and configuration
- Limited to local system resources
- Hardware and OS dependent

PyCharm

Cloud-based IDEs

- Accessible from any device with internet
- Scales with project needs (CPU, memory, storage)
- Collaborative features for real-time editing and sharing
- Pre-configured with essential tools
- Integrated with AWS services for seamless deployment

AWS Cloud9

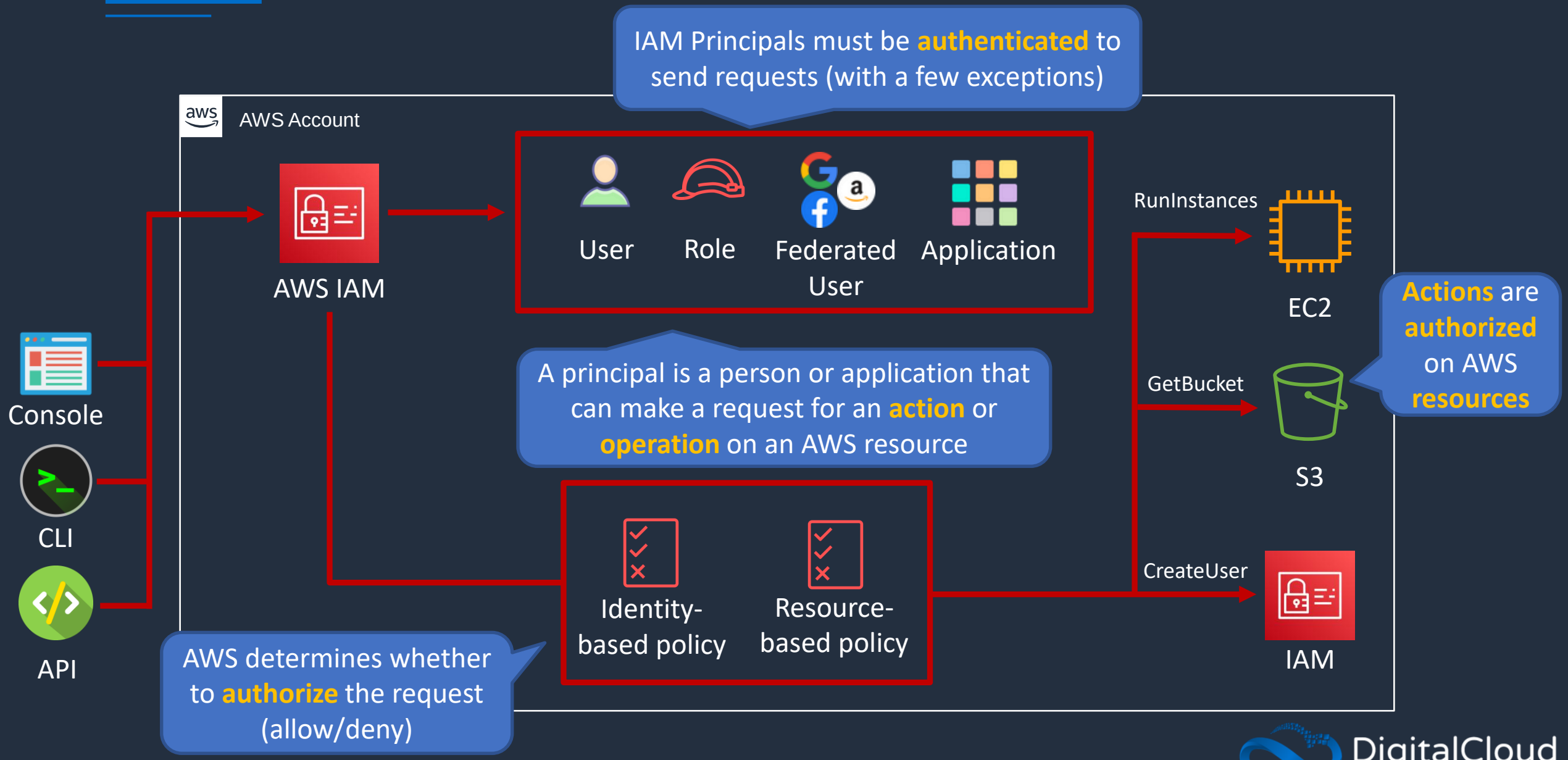
DigitalCloud TRAINING

SECTION 2

AWS Fundamentals [OPTIONAL]

AWS IAM Overview

AWS Identity and Access Management (IAM)

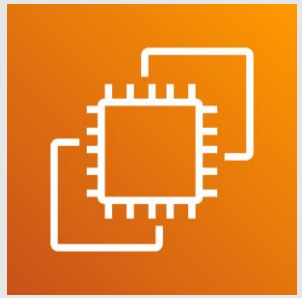


Setup Individual User Account

Root User vs IAM User

User	Login Details	Permissions
 Root User	 Email address	 Full - Unrestricted
 IAM User	Friendly name: **John** + AWS account ID or Alias	 IAM Permissions Policy

Amazon Elastic Compute Cloud (EC2)

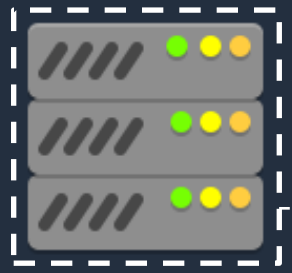


Amazon EC2

EC2 instances run Windows, Linux, or MacOS

An **EC2 instance** is a virtual server

EC2 hosts are **managed by AWS**

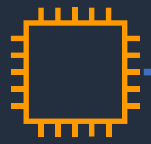


EC2 Host Server

A selection of **instance types** come with varying combinations of CPU, memory, storage and networking

DigitalCloud TRAINING

Launching an EC2 Instance



Select an **instance type**

EC2 Instance

A **snapshot** is a point-in-time backup of an instance

An **AMI** defines the configuration of the instance

Amazon Machine Image (AMI)

EBS Snapshot

Linux

Microsoft Windows

You can customize your instance and create a **custom AMI**

Customized AMI

Family	Type	vCPUs	Memory (GiB)
General purpose	t2.micro	1	1
Compute optimized	c5n.large	2	5.25
Memory optimized	r5ad.large	2	16
Storage optimized	d2.xlarge	4	30.5
GPU instances	g2.2xlarge	8	15

The **instance type** defines the hardware profile (and cost)

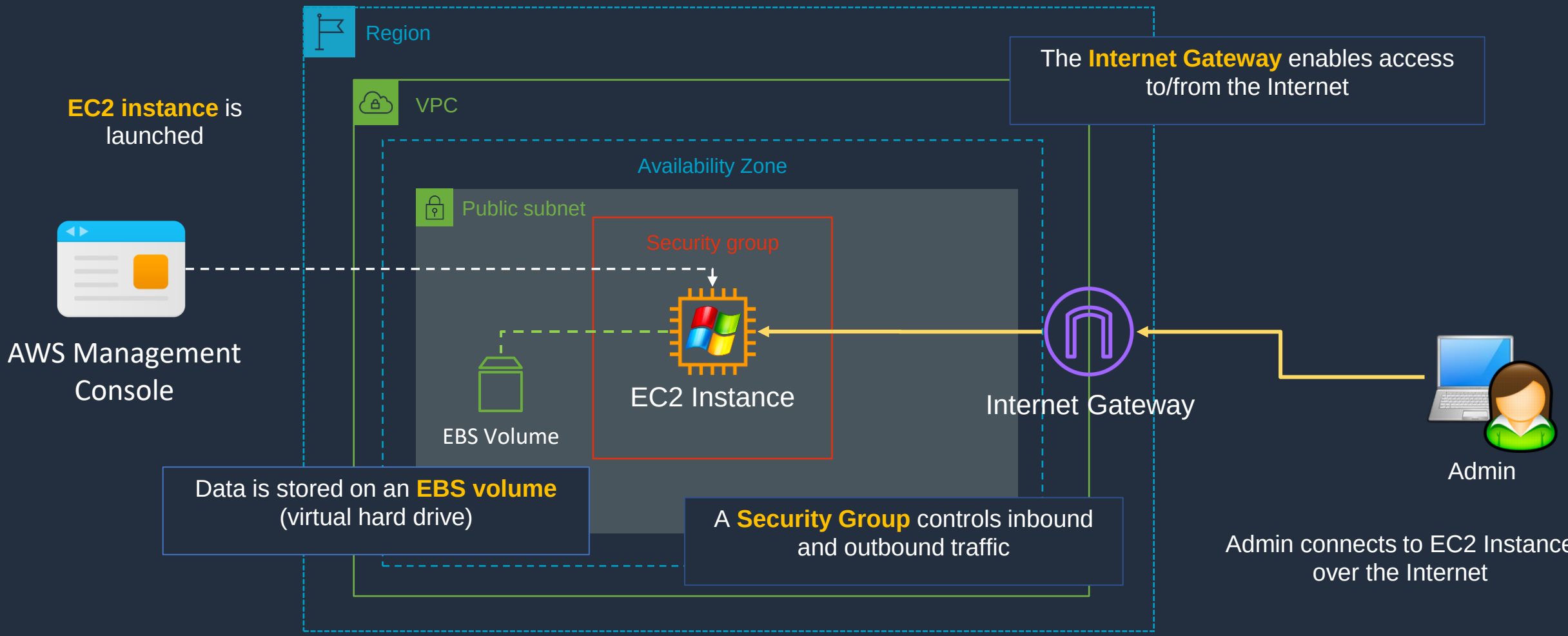
DigitalCloud TRAINING

Benefits of Amazon EC2

- **Elastic computing** – easily launch hundreds to thousands of EC2 instances within minutes
- **Complete control** – you control the EC2 instances with full root/administrative access
- **Flexible** – Choice of instance types, operating systems, and software packages
- **Reliable** – EC2 offers very high levels of availability and instances can be rapidly commissioned and replaced
- **Secure** – Fully integrated with Amazon VPC and security features
- **Inexpensive** – Low cost, pay for what you use

Launch EC2 Instances (Windows + Linux)

Amazon EC2 Instance in a Public Subnet



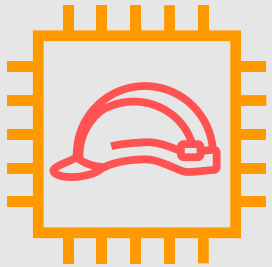
EC2 Instance Connect and SSH

EC2 Instance Connectivity Options

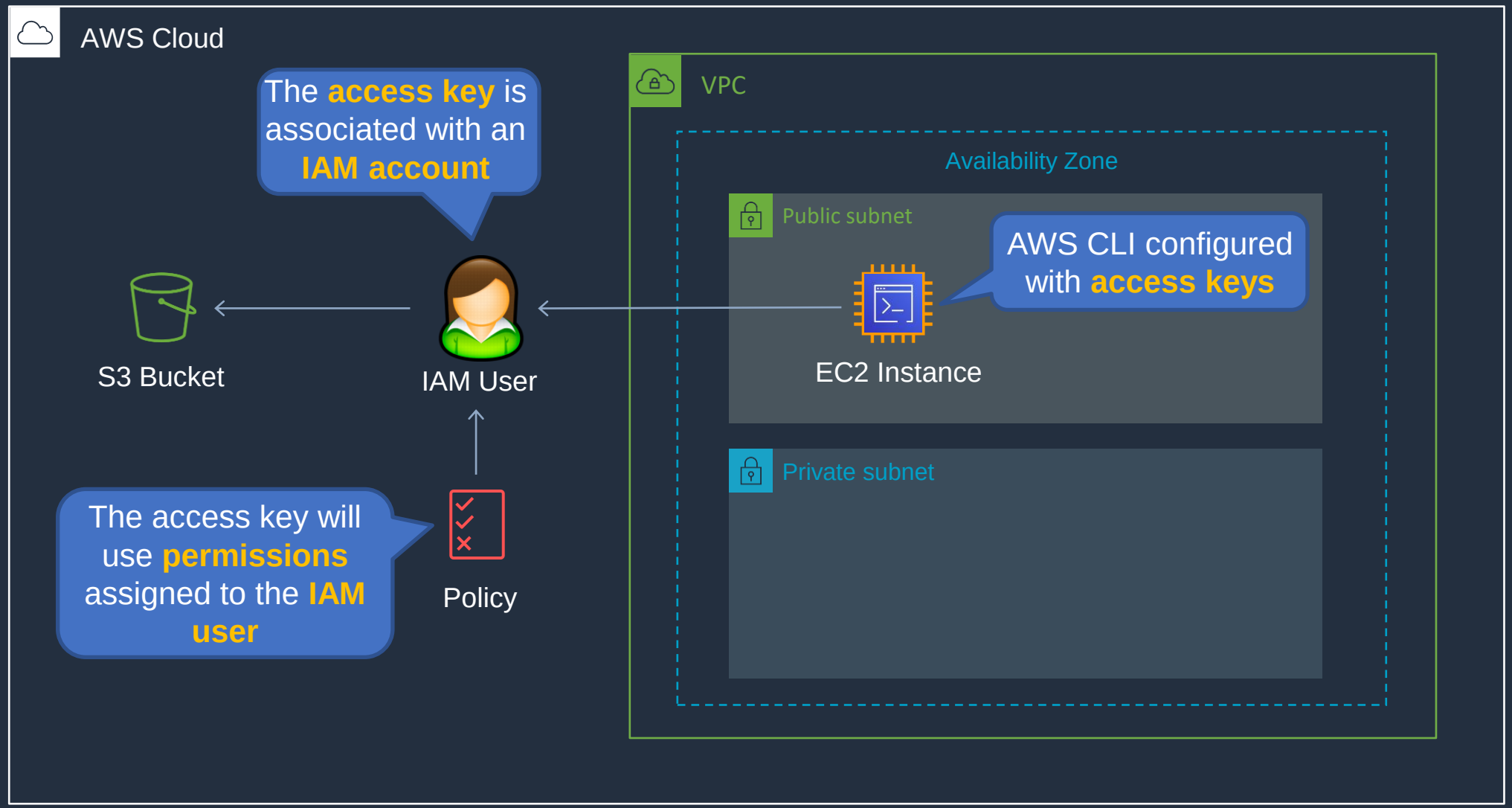
SSH/Putty	RDP	EC2 Instance Connect	Session Manager
All operating systems	Windows only	All operating systems	All operating systems
Terminal / CLI	Desktop / visual	Terminal / CLI / browser	Terminal / CLI
SSH daemon	Remote Desktop Service	SSH daemon	SSM agent
Uses instance key pair	Requires RDP client	Temporary key pair	IAM access control
Port 22 must be open	Port 3389 must be open	Port 22 must be open	No ports must be open
Anyone with key pair can access instance	Need user name and password to login	IAM access control via policy	IAM access control via policy

RDP to Windows Instance

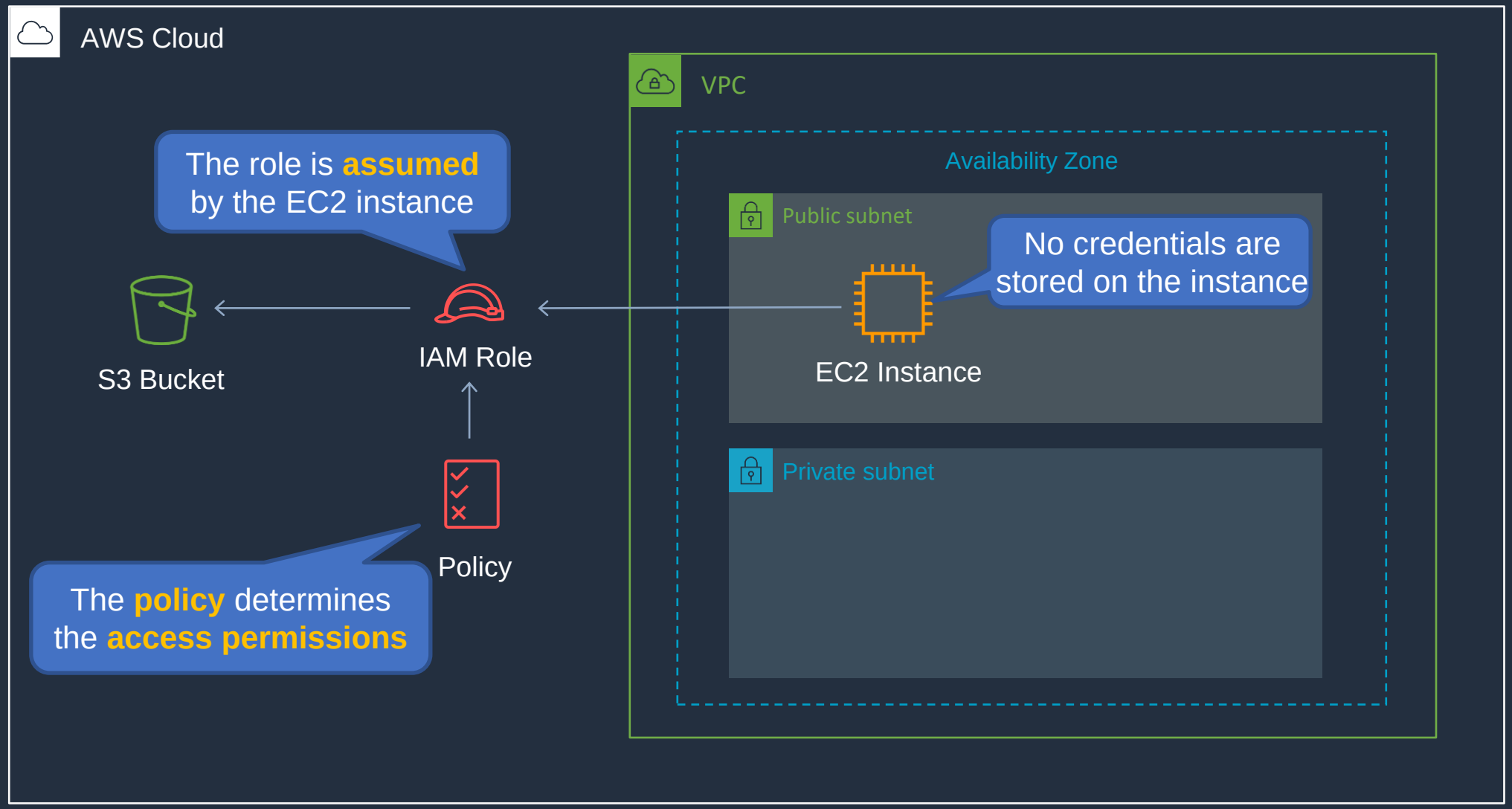
Accessing Services – Access Keys and IAM Roles



Access Keys



Amazon EC2 Instance Profiles (IAM Roles for EC2)



Access Keys and IAM Roles

Amazon VPC Overview

Amazon Virtual Private Cloud (VPC)

A **VPC** is a logically isolated portion of the AWS cloud within a region

Subnets are created within **AZs**

The **VPC router** takes care of routing within the VPC and outside of the VPC

You can launch **EC2 instances** into your VPC subnets

An **Internet Gateway** is attached to a VPC and used to connect to the Internet

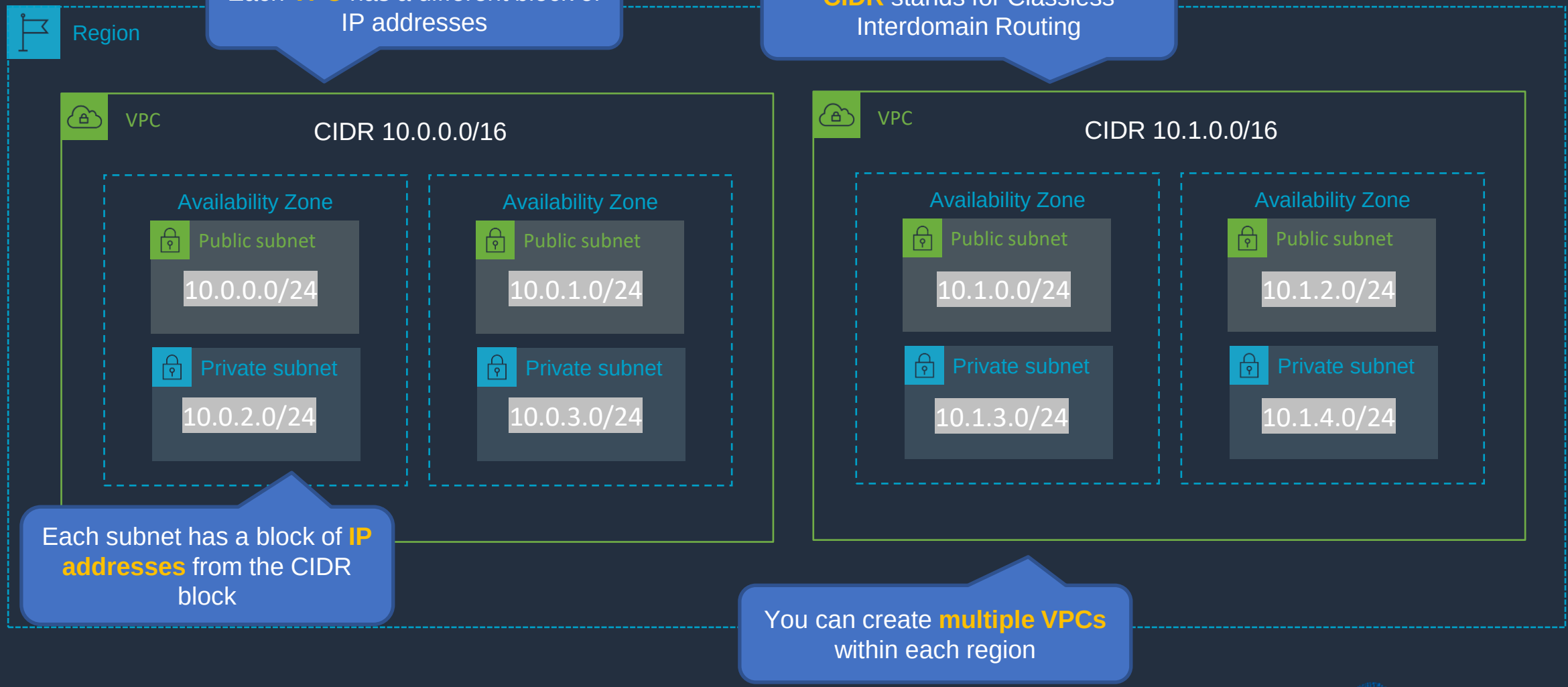
Main Route Table

Destination	Target
10.0.0.0/16	Local
0.0.0.0/0	igw-id

The **route table** is used to configure the VPC router

DigitalCloud TRAINING

Amazon VPC



DigitalCloud TRAINING

Amazon VPC Components

VPC Component	What it is
Virtual Private Cloud (VPC)	A logically isolated virtual network in the AWS cloud
Subnet	A segment of a VPC's IP address range where you can place groups of isolated resources
Internet Gateway/Egress-only Internet Gateway	The Amazon VPC side of a connection to the public Internet for IPv4/IPv6
Router	Routers interconnect subnets and direct traffic between Internet gateways, virtual private gateways, NAT gateways, and subnets
Peering Connection	Direct connection between two VPCs
VPC Endpoints	Private connection to public AWS services
NAT Instance	Enables Internet access for EC2 instances in private subnets managed by you)
NAT Gateway	Enables Internet access for EC2 instances in private subnets (managed by AWS)
Virtual Private Gateway	The Amazon VPC side of a Virtual Private Network (VPN) connection
Customer Gateway	Customer side of a VPN connection
AWS Direct Connect	High speed, high bandwidth, private network connection from customer to aws
Security Group	Instance-level firewall
Network ACL	Subnet-level firewall

Amazon VPC Core Knowledge

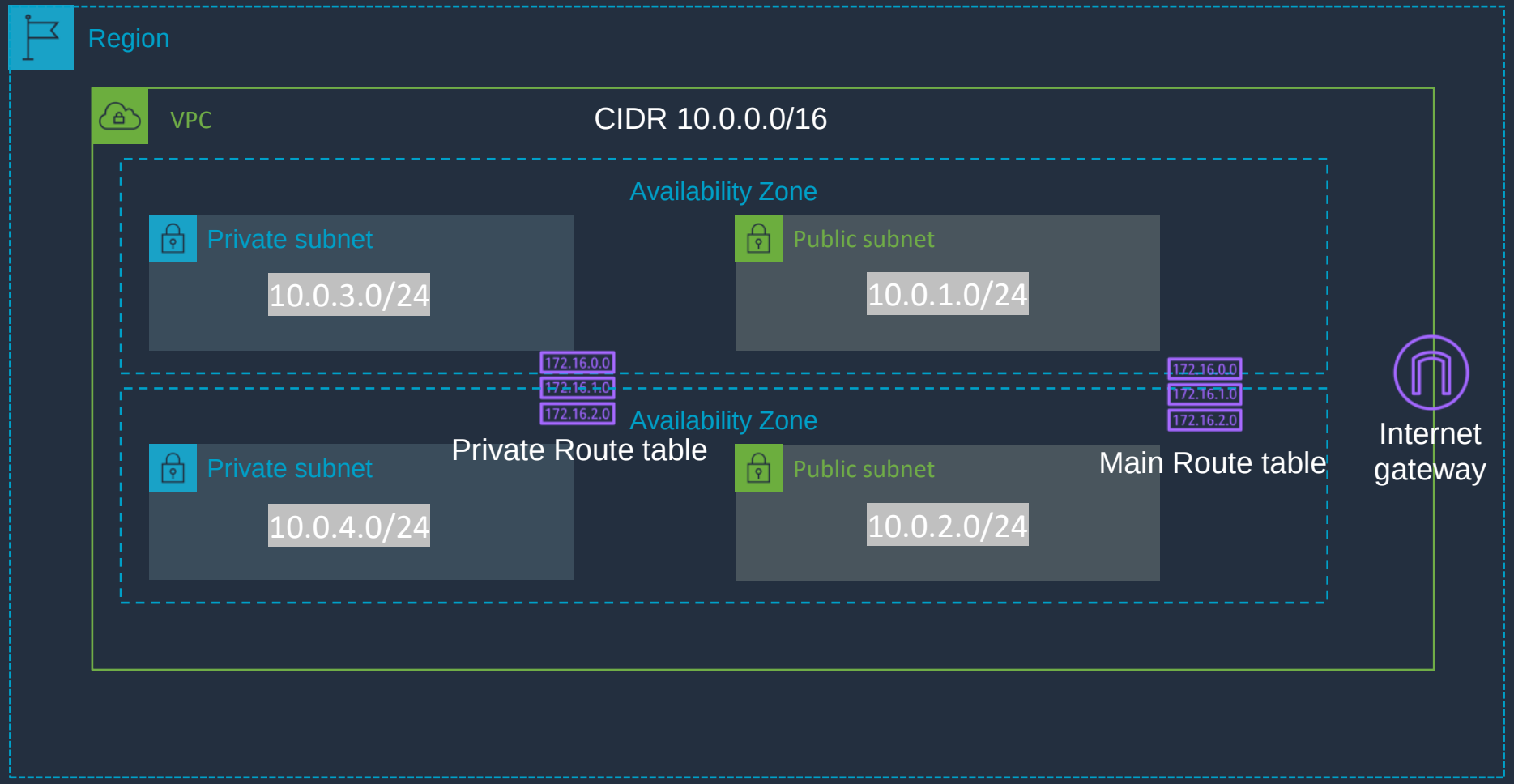
- A virtual private cloud (VPC) is a virtual network dedicated to your AWS account
- Analogous to having your own data center inside AWS
- It is logically isolated from other virtual networks in the AWS Cloud
- Provides complete control over the virtual networking environment including selection of IP ranges, creation of subnets, and configuration of route tables and gateways
- You can launch your AWS resources, such as Amazon EC2 instances, into your VPC

Amazon VPC Core Knowledge

- When you create a VPC, you must specify a range of IPv4 addresses for the VPC in the form of a Classless Inter-Domain Routing (CIDR) block; for example, 10.0.0.0/16
- A VPC spans all the Availability Zones in the region
- You have full control over who has access to the AWS resources inside your VPC
- By default you can create up to 5 VPCs per region
- A default VPC is created in each region with a subnet in each AZ

Create a Custom VPC

Create a Custom VPC



Main Route Table

Destination	Target
10.0.0.0/16	Local
0.0.0.0/0	igw-id

Private Route Table

Destination	Target
10.0.0.0/16	Local

DigitalCloud TRAINING

Amazon S3 Overview

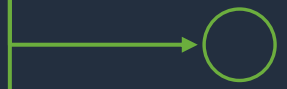
Amazon Simple Storage Service (S3)

S3 Bucket

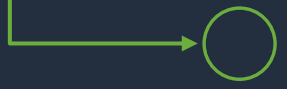
A **bucket** is a container for objects

Object

An **object** is a file you upload

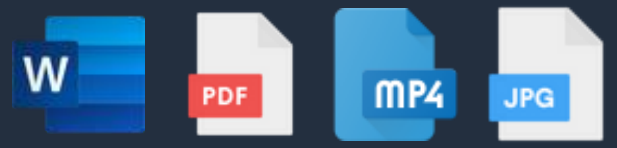


Object



Object

You can store millions of **objects** in a **bucket**



Accessing objects in a bucket:

`https://bucket.s3.aws-region.amazonaws.com/key`

`https://s3.aws-region.amazonaws.com/bucket/key`

The **HTTP protocol** is used with a **REST API** (e.g. GET, PUT, POST, SELECT, DELETE)

DigitalCloud TRAINING

Amazon Simple Storage Service (S3)

- You can store any type of file in S3
- Files can be anywhere from 0 bytes to 5 TB
- There is unlimited storage available
- S3 is a universal namespace so **bucket names** must be **unique globally**
- However, you create your buckets within a **REGION**
- It is a best practice to create buckets in regions that are physically closest to your users to reduce latency
- There is no hierarchy for objects within a bucket
- Delivers strong read-after-write consistency

Buckets, Folders, and Objects

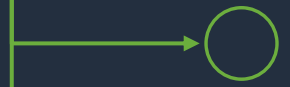
https://mybucket.s3-website-us-west-1.amazonaws.com

mybucket

The object name is the **Key** the data is the **Value**

https://mybucket.s3.us-west-1.amazonaws.com/Orders.xlsx

Orders.xlsx



https://mybucket.s3.us-west-1.amazonaws.com/Project+Alpha/Sales+Proposal.docx

Sales Proposal.docx

https://mybucket.s3.us-west-1.amazonaws.com/Project+Alpha/Project+Brief.docx

Project Brief.docx

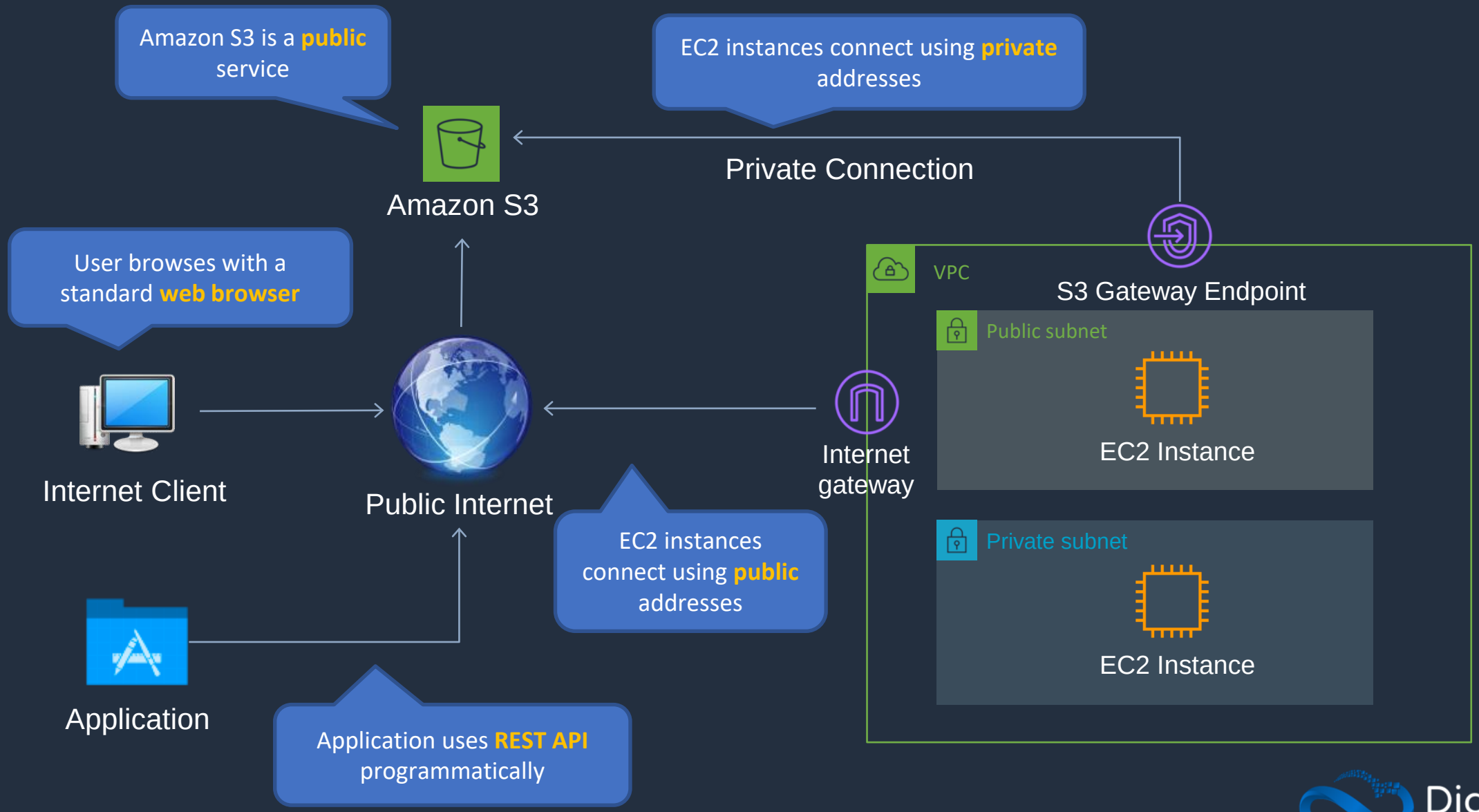
A **Folder** is a shared **prefix** for grouping objects

DigitalCloud TRAINING

Buckets, Folders, and Objects

- Folders **can** be created within folders
- Buckets **cannot** be created within other buckets
- An objects consists of:
  - Key (the name of the object)
  - Version ID
  - Value (actual data)
  - Metadata
  - Subresources
  - Access control information

Accessing Amazon S3



Amazon S3 Buckets and Objects

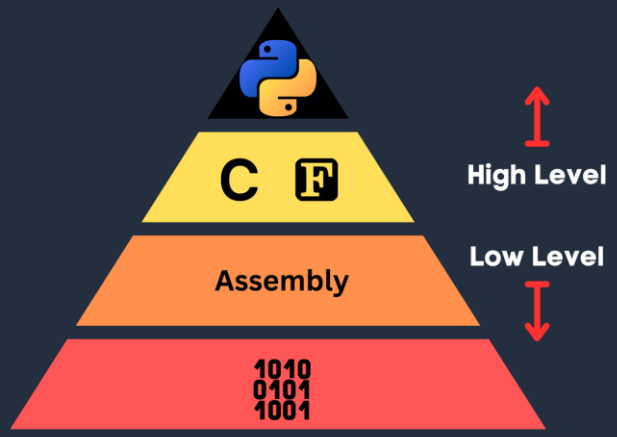
DigitalCloud TRAINING

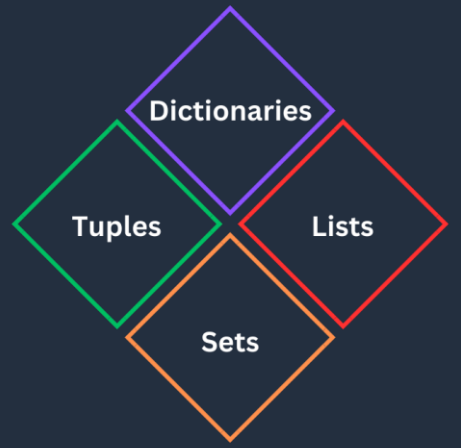
SECTION 3

Python Fundamentals

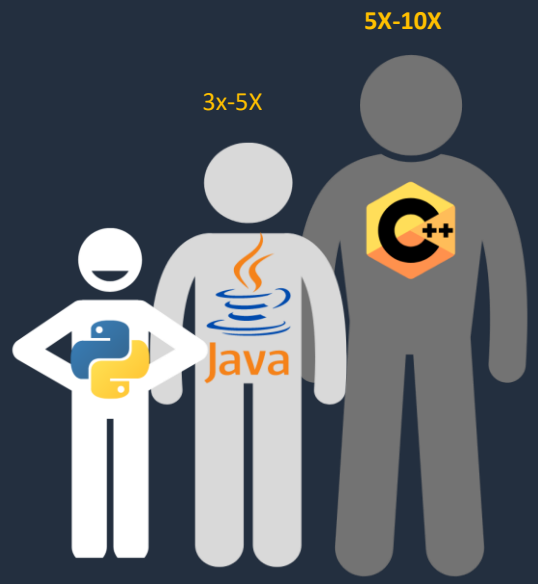
Python Overview

Python Overview





Example use cases



DigitalCloud TRAINING

Commenting Code

Commenting Code

Comment for your future self

Comment for others

```python
# This function calculates the average of a list of numbers
def calculate_average(numbers):
    # Ensure the list is not empty
    if len(numbers) == 0:
        return 0
    # Calculate the sum of the numbers in the list
    sum = 0
    for number in numbers:
        sum += number

    average = sum / len(numbers)
    return average
```

DigitalCloud TRAINING

Print Function

Print Function

- Print outputs data to the console
- Useful for:

Variables

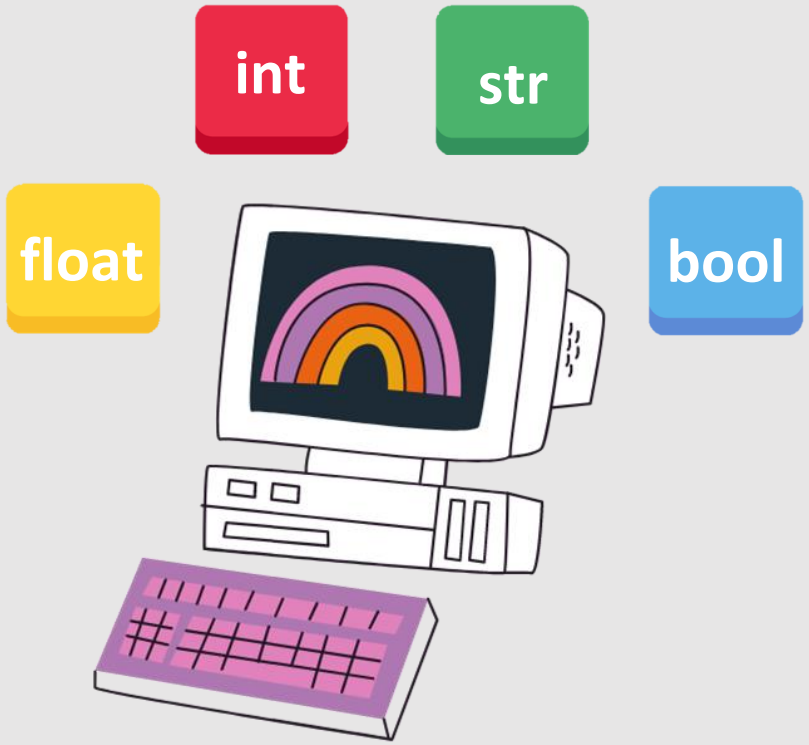
Variables

Containers

Transfer of information

Code Maintenance

Data Types



Data Types

Integer

Represent whole numbers.

```
# Integer (int)
my_integer = 39
```

String

Strings are sequences of character data.

```
# String (str)
my_string = 't2.micro'
```

Floating-Point Number

Float numbers are specified with a decimal point.

```
# Floating-point number (float)
my_float = 75.412
```

Boolean

One of two values, True or False.

```
# Boolean (bool)
is_on = True
```

DigitalCloud TRAINING

Operators

Operators

Assignment

Operators used to assign values to variables.

```
8  number = 10
```

Arithmetic

Can be used to concatenate strings.

```
26 greeting = "Hello, "
27 name = "World!"
28 message = greeting + name
```

Comparison

Compare values and return if the comparison is true or false.

```
44 number_1 = 121
45 number_2 = 39
46
47 if number_1 == number_2:
48     print(f"{number_1} is equal to {number_2}.")
```

Logical

Can be used to combine conditional statements.

```
41 number = 15
42
43 if number > 10 and number < 20:
44     print(f"{number} is between 10 and 20.")
```

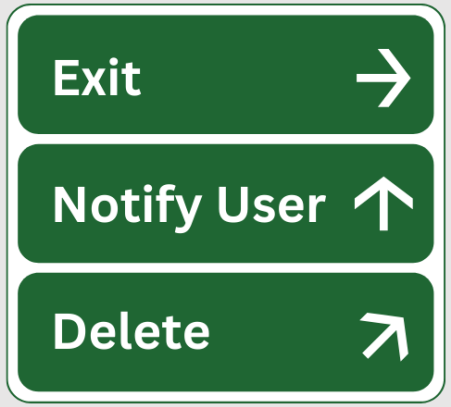
Membership

Can be used to test whether a value is a member of a sequence, like strings, lists, or tuples.

```
51 sentence = "Welcome to the world of Python programming."
52 word = "Python"
53
54 if word in sentence:
55     print(f"The word '{word}' is in the sentence.")
```

DigitalCloud TRAINING

Conditional Statements



Conditional Statements

- A piece of code that executes instructions based on a particular state
- Useful for:
  - Controlling the flow of a program
  - Executing different code based on different conditions

- In a Python program, the if statement is how you perform this sort of decision-making.

```
1  weather = 'cold'
2
3  if weather == 'cold':
4      drink = "a hot coffee"
5  else:
6      drink = "an iced coffee"
7
8  print("Hi, I'd like " + drink + " please.")
```

Functions

Functions

What is a Function?

- A block of organized and reusable code that performs a specific task
- Functions help break a program into smaller, more manageable pieces
- Code reuse and modularity
- Essential for code that is:
  - Efficient
  - Maintainable
  - Scalable
- Built-in & user-defined functions
- Think of functions as factories

```python
"""This script demonstrates the use of Python functions.
The create_shoe function takes a list of materials as input and
determines the type of shoe created based on those materials.
"""
def create_shoe(materials_list):
    shoe_type = ''

    if 'leather' in materials_list and 'rubber' in materials_list:
        shoe_type = 'boots'
    elif 'mesh' in materials_list and 'rubber' in materials_list:
        shoe_type = 'sneakers'
    else:
        shoe_type = 'unknown'

    return {'type': shoe_type, 'materials': materials_list}
```

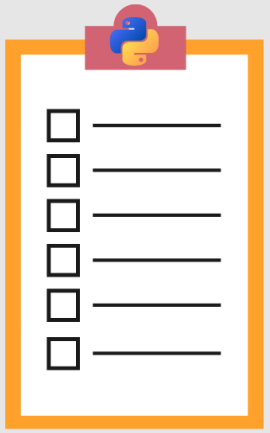
Inputs

Function

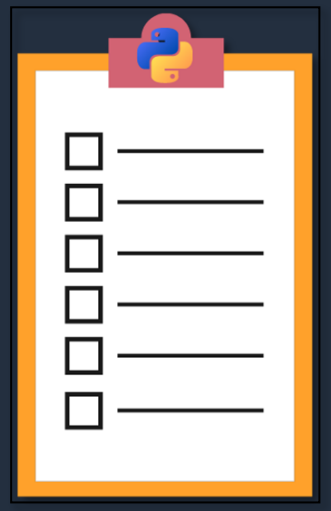
Output

DigitalCloud TRAINING

Lists



Lists



What are lists and why use them?

- An ordered collection of items
- Especially useful when you need to work with a collection of related items
- Containers that help you store and manage multiple pieces of data that can be of different data types
- Think of a list as a grocery shopping list

```
7  shopping_list = [
8      'eggs', 'flour', 'spinach', 'onion', 'mushrooms,' 'tomatoes'
9      ]
```

How to work with lists:

- Creating a list
- Accessing list elements
- Modifying list elements

DigitalCloud TRAINING

Dictionaries

Dictionaries

What are dictionaries and why use them?

- An unordered collection of key-value pairs
- Think of a python dictionary as a real-world dictionary
- Each key is unique

```
1  baked_dish_fillings = {
2      "quiche": "eggs, cream, cheese, spinach, onions, and mushrooms",
3      "empanadas": "ground beef, vegetables, and spices",
4      "chicken pot pie": "chicken, vegetables, and gravy"
5  }
```

How to work with dictionaries:

- Creating a dictionary
- Accessing dictionary values
- Modifying dictionary values
- Adding and removing key-value pairs

- Dictionary methods
  - keys()
  - values()
  - items()

Loops

Loops

Definite iteration - The number of repetitions is specified explicitly in advance


```
1  ingredients = ['spinach', 'cheese', 'onions', 'mushrooms']
2
3  # Use a for loop to iterate through the list of ingredients
4  for ingredient in ingredients:
5      print(f"Adding {ingredient} to the recipe.")
```

Indefinite iteration - The number of repetitions isn't specified explicitly in advance.


```
7  desired_cookies = 12
8  baked_cookies_count = 0
9
10 # Use a while loop to bake cookies until we reach the desired number
11 while baked_cookies_count < desired_cookies:
12     print("Baking a cookie...")
13     baked_cookies_count += 1
14
15 print(f"We have baked {baked_cookies_count} cookies.")
```

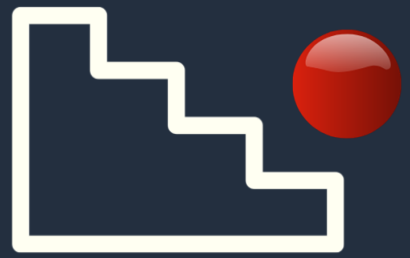
DigitalCloud TRAINING

Debugging

Debugging

Debugging is **the practice of identifying and fixing errors**

- Inspecting code
- Print statements
- Creating and viewing logs
- Debugging in IDE's
  - Setting breakpoints
  - Stepping through code



SECTION 4

AWS Services Using Python

AWS Boto3 SDK

AWS Boto3 SDK

An SDK (Software Development Kit) is a set of preconfigured tools and libraries

Boto3 allows us to write Python code that interacts with AWS services

- Boto3 contains easy-to-use APIs to many AWS services, including:

automation

S3

EC2

DynamoDB

CloudWatch

upload_file()

create_fleet()

put_item()

filter_log_events()

DigitalCloud TRAINING

AWS Boto3 SDK Cont.

Resource

- Higher-level, object-oriented API
- Represents AWS service objects (e.g., S3 bucket, EC2 instance)
- Allows actions directly on objects
- Use when you prefer a more Pythonic and concise interface

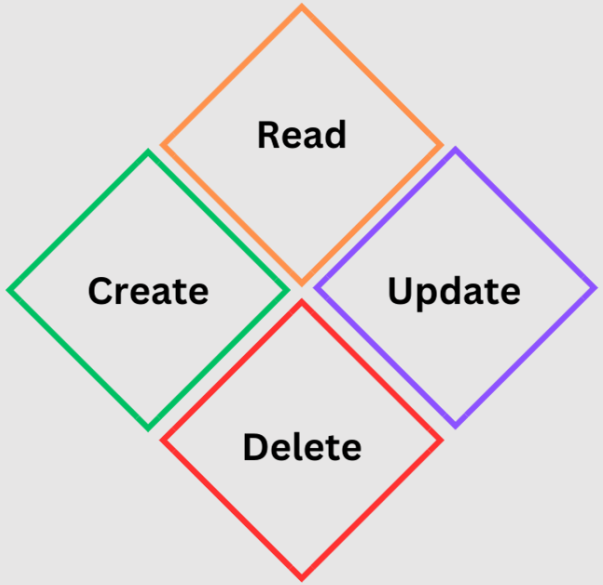
```python
import boto3

instance_id = 'your-instance-id'
ec2 = boto3.resource('ec2')
instance = ec2.Instance(instance_id)

instance.start()
```

Client

- Use when you need fine-grained control and access to all available operations
- Lower-level, service-oriented API
- Provides access to all AWS service actions

```python
import boto3

instance_id = 'your-instance-id'
client = boto3.client('ec2')
response = client.describe_instances(InstanceIds=[instance_id])

tags = response['Reservations'][0]['Instances'][0]['Tags']

for tag in tags:
    print(f"{tag['Key']}: {tag['Value']}")
```

S3: CRUD Operations and Boto3

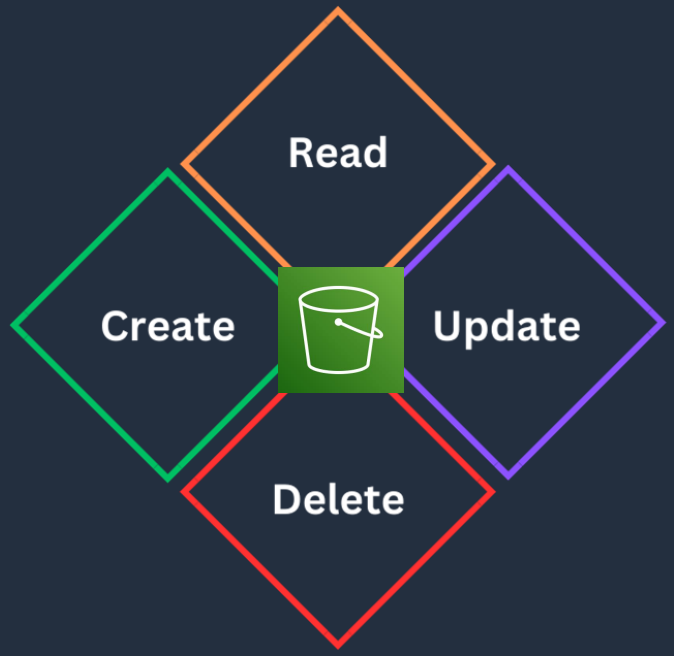


S3: CRUD Operations and Boto3

What is **CRUD**?

- Four basic operations that can be performed on data in a database or a data service like S3

```
1 obj = s3.Object('dct-crud-1', 'file_1.txt')
2 body = obj.get()['Body'].read()
```

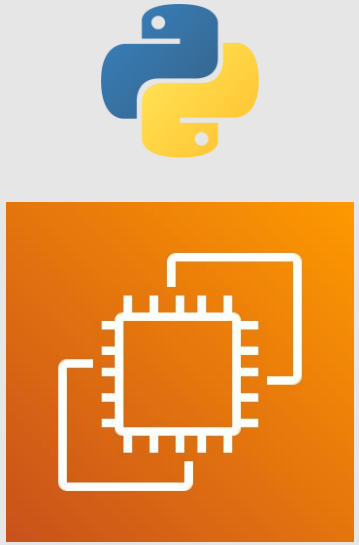



```
1 s3.create_bucket(
2     Bucket='dct-crud-1',
3     CreateBucketConfiguration={
4         'LocationConstraint': 'us-west-2'
5     }
6 )
```

```
1 s3.Object('dct-crud-1', 'file_1.txt').put(
2     Body=open('file_2.txt', 'rb')
3     )
```

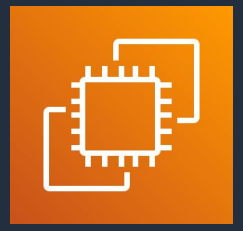
```
1 bucket = s3.Bucket('dct-crud-1')
2 bucket.delete()
```

EC2: Basics, Launching, and Management



EC2: Basics and Boto3

EC2 (Elastic Compute Cloud)



- A core component of the AWS platform
- Resizable, scalable compute capacity in the cloud
- Simplifies the set up and scaling of virtual servers
- Works in conjunction with Amazon VPC to provide security and robust networking functionality

Using Boto3 with EC2

- Automates and streamline tasks and operations
- You can launch and manage EC2 instances using Python code
- Resource and Client
  - Creates one or more new instances
  - Flexible because of many parameters
- **run_instances()** Client Method
- **create_instances()** Resource Method

EC2: Launching Instances

- If you don’t specify a subnet ID, a default subnet from default VPC will be chosen
- If you don’t specify a private IPV4 address, AWS will choose one from the IPv4 range of your subnet
- If you don’t specify a security group ID, AWS uses the default security group
- You can create a launch template, which is a resource that contains the parameters to launch an instance
- An instance is ready for you to use when it’s in the running state

```
new_instance = ec2.create_instances(
    ImageId='ami-0889a44b331db0194',
    InstanceType='t2.micro',
    MinCount=1,
    MaxCount=1,
    Placement={
        'AvailabilityZone': 'us-east-1e'
    },
    SecurityGroupIds=[
        'sg-076869e06e0541d6b',
    ],
    KeyName='DCT-3',
    SubnetId='subnet-08a488423a06d2980',
    UserData="#!/bin/bash\necho 'Hello, World!' > /home/ec2-user/hello.txt\n",
    TagSpecifications=[
        {
            'ResourceType': 'instance',
            'Tags': [
                {
                    'Key': 'Name',
                    'Value': 'My Linux Instance'
                }
            ]
        }
    ]
)
```

EC2: Managing Instances

- Using a resource, reference the instance by passing the instance ID
- Call operations on an instance using an action
  - attach_volume
  - create_tags
  - delete_tags
  - monitor
  - reboot
  - reload
  - start
  - stop
  - terminate

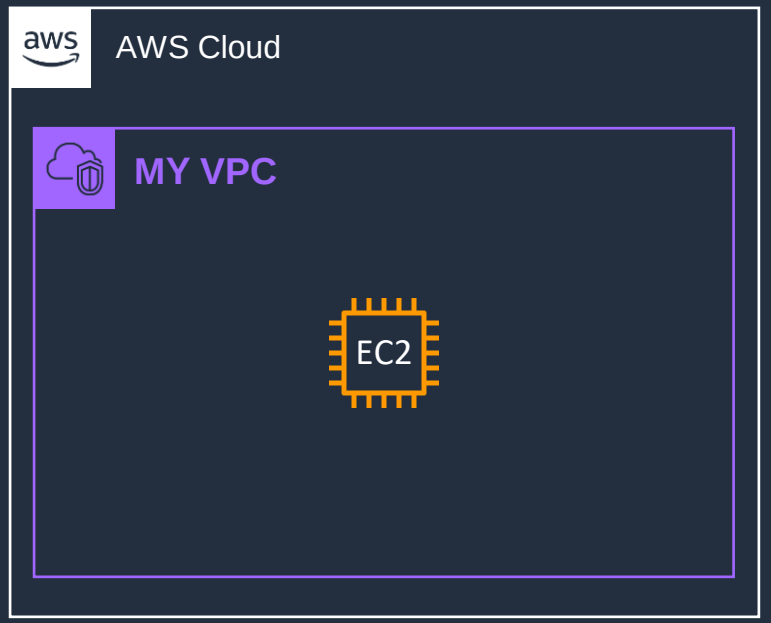
```
# Stop an instance
ec2.Instance(instance_id).stop()

# Start an instance
ec2.Instance(instance_id).start()

# Terminate an instance
ec2.Instance('i-09ecbf06fc2fa097e').terminate()
```

VPC: Basics, Creation, and Management

VPC: Introduction and Basics

VPC (Virtual Private Cloud)

- Scalable and highly configurable AWS service
- Provides networking functionality
- A customizable logically isolated section of the AWS Cloud
- Including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways
- Into which you can launch AWS resources

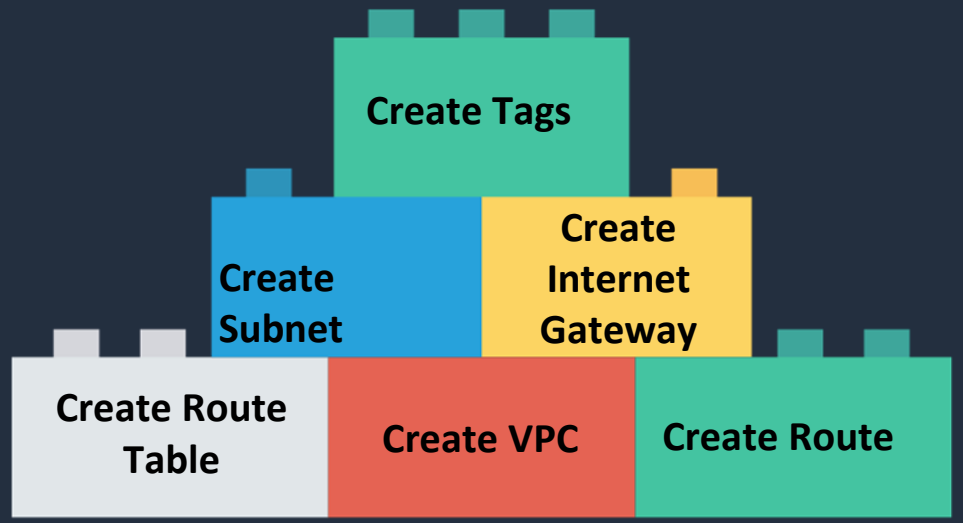


Boto3

- VPC and EC2 Relation - logical grouping from an API design perspective
- EC2 Resource or client

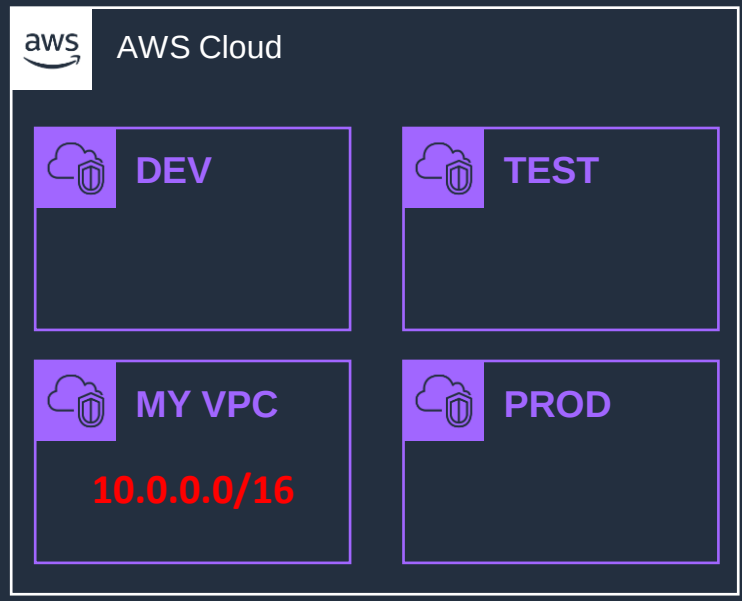
VPC: Creation and Management

- Previous examples had their own client or resource
- VPC is created with EC2 client or resource
- Methods used with VPCs include:



- You can create multiple VPCs
  - Helpful for managing separate environments
- Creating and tagging a VPC

```
create_vpc(CidrBlock='10.0.0.0/16')
create_tags(**kwargs)
```



DigitalCloud TRAINING

RDS: Basics, Aurora Serverless, DB Management

Amazon RDS

RDS - Allows you to run isolated database environments in the cloud

Supported Databases

MySQL

PostgreSQL

Microsoft SQL Server

ORACLE

MariaDB

Amazon Aurora

Security and Access

RDS provides multiple layers of security, including:

- network isolation
- encryption at rest and in transit
- Fine-grained access control using (IAM)

Managed Service

Amazon RDS takes care of the tedious database administration tasks such as:

- Backups
- Software Patching
- Monitoring
- Scaling

Use Cases

E-commerce platforms

Customer relationship management apps

DigitalCloud TRAINING

RDS: Amazon Aurora Serverless

An on-demand version of Amazon Aurora with auto-scaling capabilities.
Managed via RDS which handles provisioning, patching, backup, restore, and other tasks

Removes complexity of provisioning and managing database capacity

Seamlessly scale compute and memory capacity

Aurora Serverless

Highly available
Built on distributed, fault-tolerant, self-healing Aurora storage with 6-way replication to protect against data loss.

Cost-effective
Pay only for the resources you consume

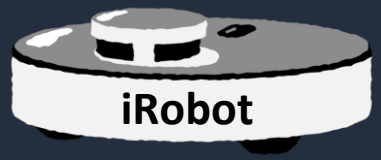
- Aurora Serverless only works with PostgreSQL and MySQL

MySQL

PostgreSQL

- The required computing capacity is configured via Aurora Capacity Units (ACUs), which are a combination of virtual CPUs and memory
- 1 ACU has approximately 2 GiB of memory

Use Case



DigitalCloud TRAINING

RDS: Aurora Serverless V1 and V2

Aurora Serverless V1

- Suitable for apps without frequent traffic
- Can cut capacity to zero ACUs during idle periods
- Scale up latency (cold start)
- Scales by doubling ACUs (1, 2, 4, 8, 16 … 256)
- Best for test and dev environments
- Console Query Editor
- About $0.06 an hour per ACU

Aurora Serverless
V1

Aurora Serverless V2

- Supports all features of Aurora
- Suitable for production environments
- Instant scaling to hundreds-of-thousands of transactions per second
- Scales in increments of 0.5 ACUs
- Cannot cut capacity to zero
- Does not support the query editor
- About $0.12 an hour per ACU

Aurora Serverless
V2

Boto3

- RDS Client
- create_db_cluster()
- describe_db_clusters()

```python
response = rds.create_db_cluster(
    Engine='aurora-mysql',
    EngineVersion='5.7.mysql_aurora.2.08.3',
    DBClusterIdentifier=db_cluster_id,
    MasterUsername=username,
    MasterUserPassword=password,
    DatabaseName='rds_hol_db',
    DBSubnetGroupName=db_subnet_group,
    EngineMode='serverless',
    EnableHttpEndpoint=True,
    ScalingConfiguration={
        'MinCapacity': 1, # Minimum ACU
        'MaxCapacity': 8, # Maximum ACU
        'AutoPause': True,
        'SecondsUntilAutoPause': 300
    }
)
```

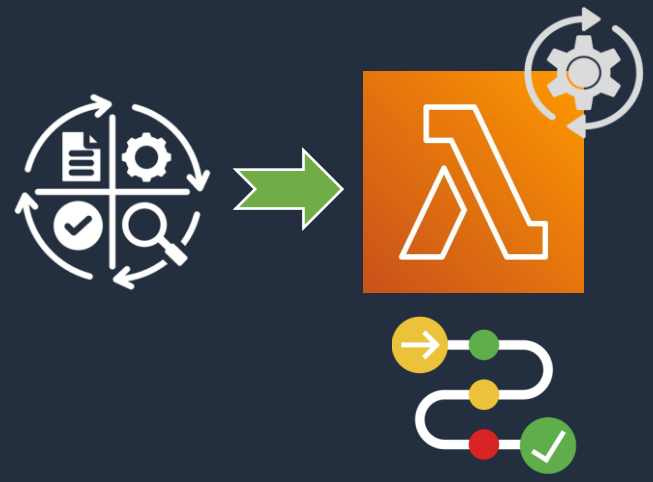
SECTION 5

Automating AWS Tasks With Lambda Functions

AWS Lambda

Amazon Lambda Functions

A serverless compute service that automatically runs your code in response to events and manages the underlying compute resources for you.



Scales in response to each trigger

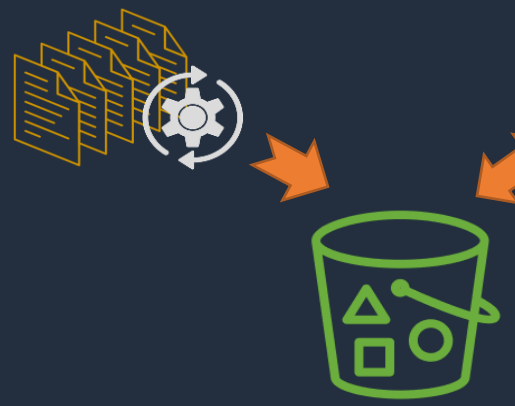
Highly available infrastructure

Only pay for the compute time and number of requests that you consume

Supported Languages

Common Use Cases

Real-time file processing



Resize images

Parse/filter

DigitalCloud TRAINING

Lambda & Python

Why Lambda is important

- More efficient code execution
- Lambda's serverless nature lets developers focus more on their code and less on infrastructure management
- Quicker deployment times

Why Python?

- Readability
- Extensive libraries
- Python simplicity + power of Lambda
  - Robust, scalable serverless applications

Pandas

Pillow

Example of services used with Lambda in this course

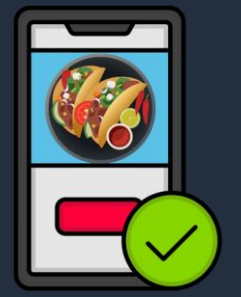
Food Delivery App Example

Pricing

- Compute and request charges
- Free Tier Usage includes
  - 400,000 GB-seconds
  - 1 million requests per month
- x86 based processor price: $0.0000166667 USD per GB-second
- Requests: $0.20 USD per 1M

Lambda

- 3 million requests per month
- 120 milliseconds per request
- Configured with 1.5 GB of memory, on an x86 based processor



Compute Charges:

3 million requests * 120 milliseconds = 360,0000 seconds

360,000 seconds * 1.5 GB = 540,000 GB-s

540,000 GB-s - 400,000 GB-s (Free Tier) = 140,000 GB-s

140,000 GB-s * $0.0000166667 = **$2.33**

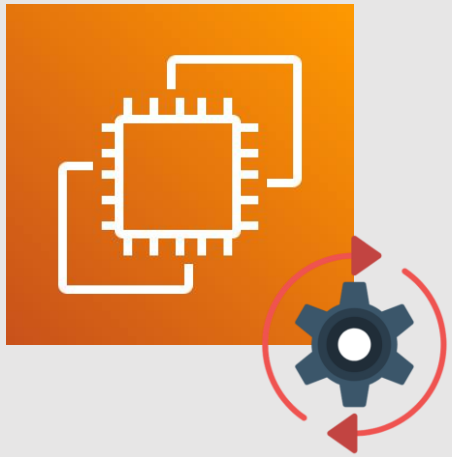
Request Charges:

3M requests – 1M (free tier) = 2M

2M x $0.2/M = $0.40

Total cost to run Lambda: $2.73 USD per month

Automating EC2 with Lambda

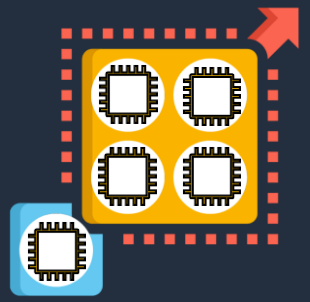


Automating EC2 with Lambda

Why Automate EC2 with Lambda?

Efficiency
- Instance scheduling
- Auto-scaling
- System events responses

Scalability

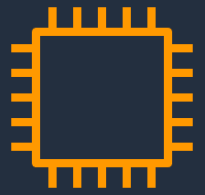


Error Reduction

Cost-Effectiveness

How Automate EC2 with Lambda?

EC2 instances can trigger events
- Instance launching
- Terminating
- State changes



Lambda can respond to these events

- Stops an EC2 instance when it's been idle for too long

- Create backups at regular intervals to prevent any data loss

DigitalCloud TRAINING

Automating EC2 Backups with Lambda

Scenario

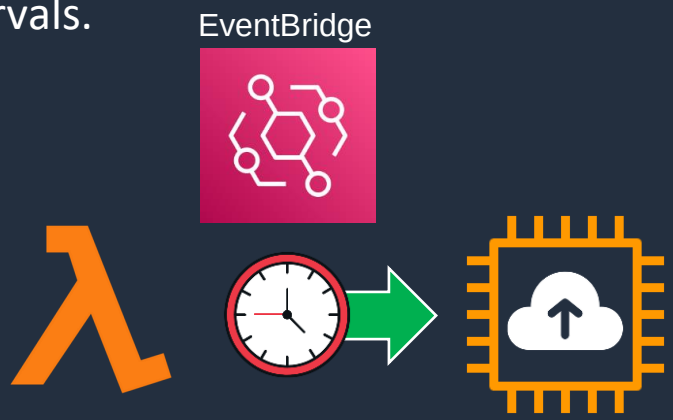
Your EC2 instance contains crucial data that would be very costly to lose.

Solution

Best practice: Create backups at regular intervals to prevent any data loss in case of instance failures or errors.

Setting the schedule

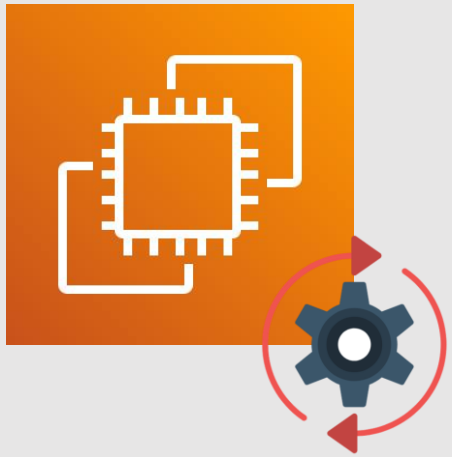
Use EventBridge Scheduler to trigger the lambda function at regular intervals.




```python
def lambda_handler(event, context):
    ec2 = boto3.client('ec2')
    response = ec2.create_snapshot(
        Description='My EC2 snapshot',
        VolumeId='Your-Volume-ID',
        TagSpecifications=[
            {
                'ResourceType': 'snapshot',
                'Tags': [
                    {
                        'Key': 'Name',
                        'Value': f'My EC2 snapshot'
                    },
                ]
            },
        ]
    )
```


DigitalCloud TRAINING

Automating EC2 with Lambda HOL



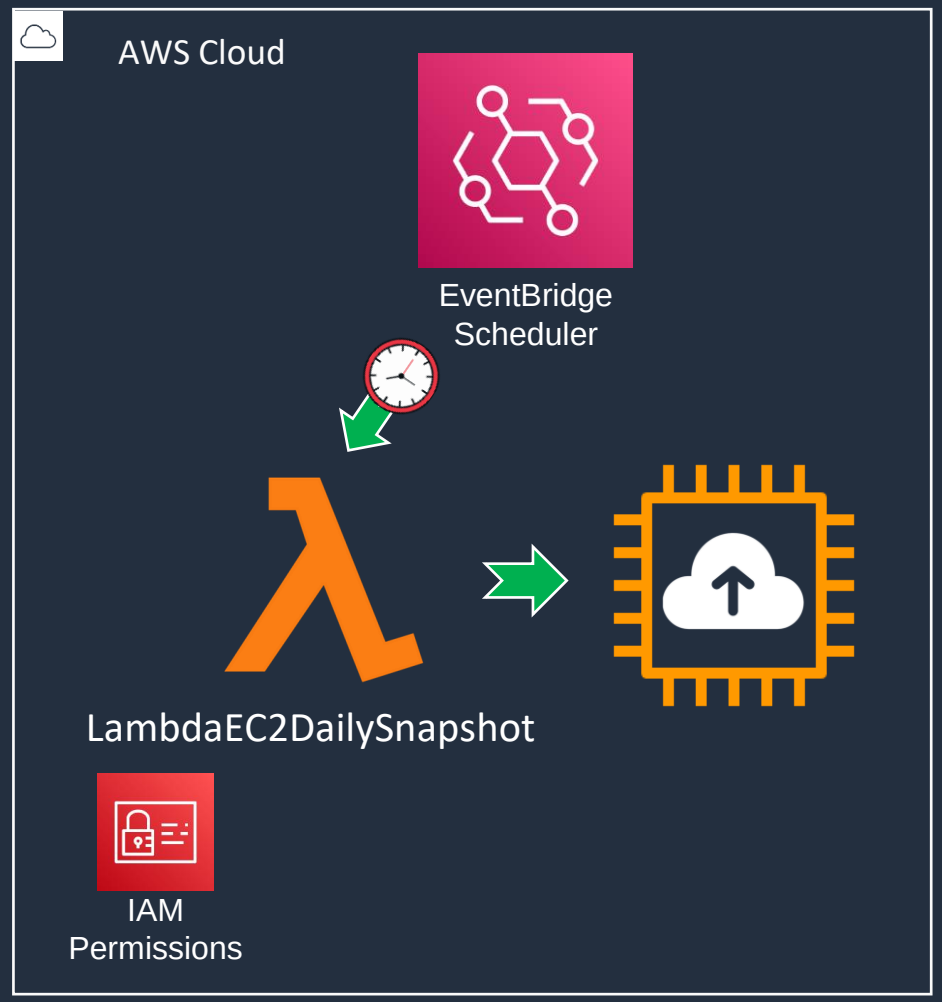
DigitalCloud TRAINING

Automating EC2 Backups with Lambda HOL

Task:
Create a lambda function which creates a snapshot of our EC2 instance at regular intervals.

Prerequisites:

- Create ‘LambdaEC2DailySnapshot’ lambda function
- Download lambda function into your Cloud9 environment
- Create or have an EC2 instance ready to use



Automating S3 with Lambda

Automating S3 with Lambda: Use Cases

Why Automate S3?

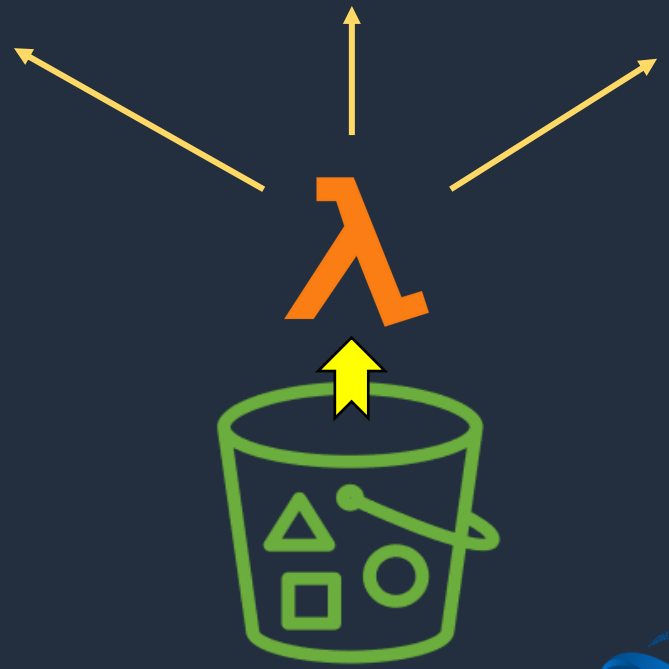
Efficiency

Real-World Applications

File Transformation

Log Analysis

Archiving Specific Data



Common Use Cases

Processing

Alerts

Clean-up

Automating S3: Key Concepts

Event-Driven Architecture and Programming

- In Python, we often use event-driven programming to respond to real-time events
- AWS expands this concept with an event-driven architecture
- Python plays a key role here, as it's one of the primary languages used for writing Lambda functions.
- Events in S3 can trigger responses in Lambda

S3 Events and Event Notifications

- S3 supports a variety of events
- We can set S3 to send an event notification whenever these events occur, which can trigger a Lambda function.

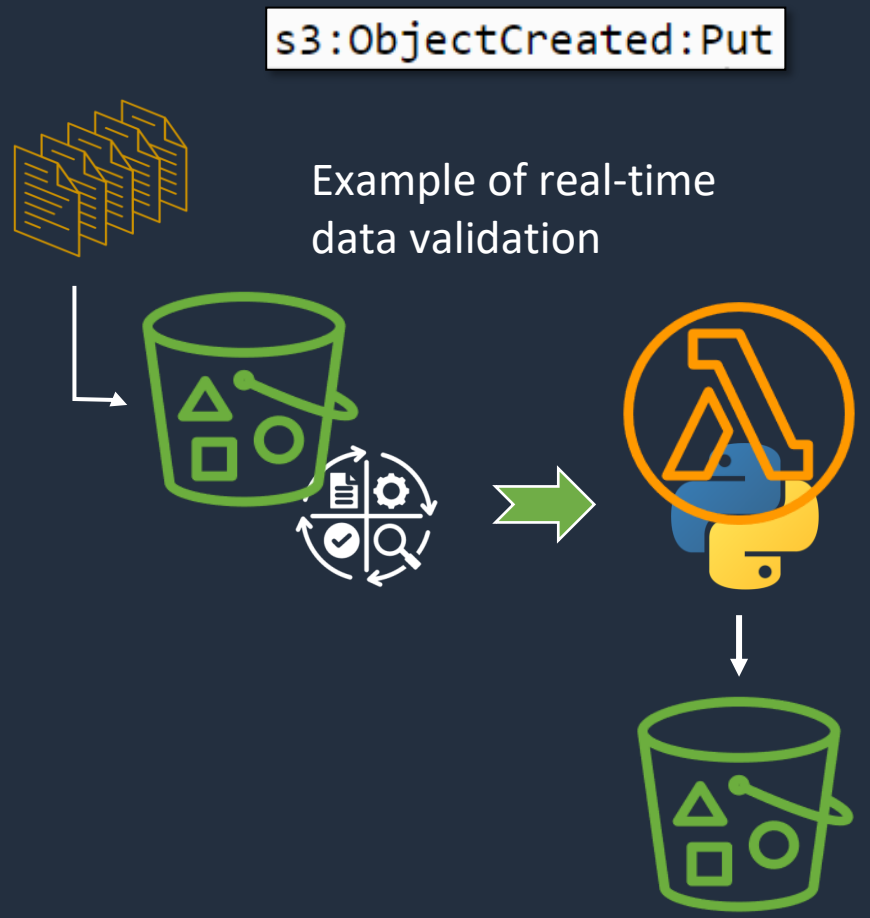
`s3:ObjectRemoved:*`

- Delete
- DeleteMarkerCreated

`s3:ObjectCreated:*`

- Put
- Post
- Copy

Lambda and S3 Interaction



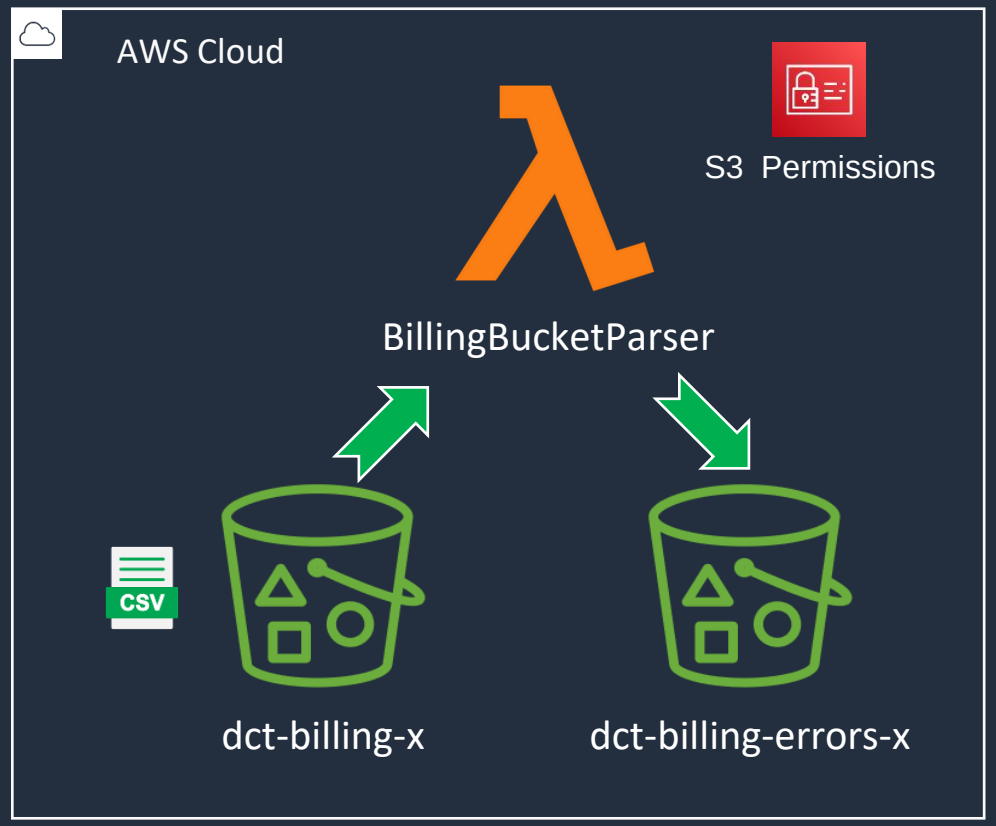
Automating S3 with Lambda: Real-time Data Validation HOL

Real-time Data Validation HOL: Automating S3 with Lambda

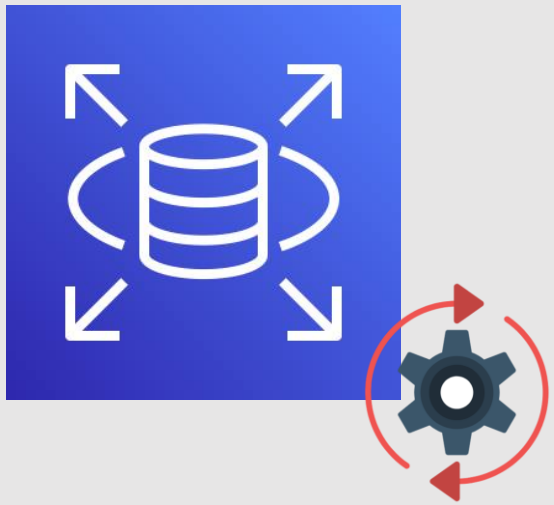
Overview:
You will set up a Lambda function which will be triggered whenever a new CSV file is uploaded to an S3 bucket. It will then validate the file and if any discrepancies are found, it will move the file to an 'error' bucket.

Prerequisites:
1. Boilerplate (default) Lambda function 'BillingBucketParser' downloaded in Cloud9 environment.
2. Two S3 buckets with default settings.
   a) 'dct-billing-x' for uploading csv files.
   b) 'dct-billing-errors-x' for error files.
3. Download the provided csv test files.

Tasks:
1. Add necessary permissions for the Lambda role.
2. Write the Lambda Python code.
3. Set up the S3 trigger.



Automating RDS with Lambda



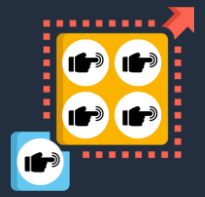
Automating RDS with Lambda

RDS & Lambda - A Powerful Combination

Automate DB Tasks

- Scheduled SQL queries
- Backups and snapshots
- Data archiving and purging activities based on business rules

Scalability

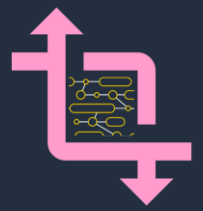


- Lambda and RDS can scale independently
- Lambda can handle more requests without overloading the RDS instance
- More granular and cost-effective scaling

Event-Driven DB Operations

- Sending alerts based on specific DB conditions
- Custom failover logic for high availability scenarios
- Auto-scaling instances based on usage patterns

Data Alteration



- Lambda can help maintain data cleanliness and integrity
- Pre-processing or transforming data before it's inserted into RDS
- Post-processing data after it's retrieved from RDS

Microservices & Serverless Apps

- Lambda can handle business logic of various microservices, working with RDS to:
- Implement efficient decoupling of compute and storage services
- Respond to HTTP requests through Amazon API Gateway and perform CRUD operations on RDS

Automating RDS with Lambda

Data Transformation

- Example: Normalizing currencies in financial data before storing in RDS
- AWS Lambda can perform these transformations as part of the ETL (Extract, Transform, Load) process

Automating ETL Processes:

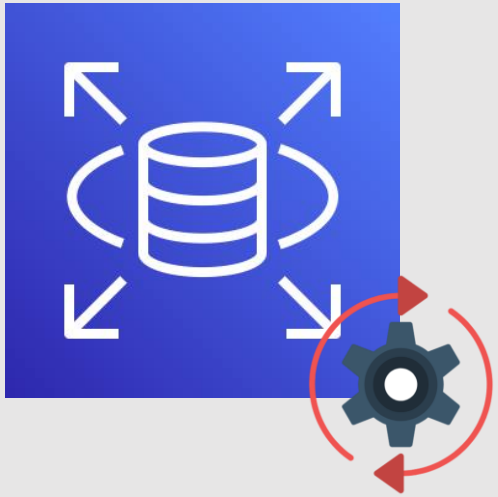
- Lambda can automate this ETL process
- Upon the arrival of new data Lambda can read, transform and load the data into an RDS instance

```
id,company_name,country,city,product_line,item,bill_date,currency,bill_amount
1,The Beef Baron,US,San Antonio,Meat,Wagyu Beef,2023-06-01,USD,5100.00
2,Longhorn Delicacies,US,Dallas,Meat,USDA Prime Ribeye,2023-06-02,USD,6800.00
3,Bison Bites,US,Houston,Meat,Bison Burgers,2023-06-03,USD,6300.00
4,The Windy City Butcher,US,Chicago,Meat,Chicago-style Hot Dogs,2023-06-04,USD,5600.00
5,East Coast Charcuterie,US,New York,Meat,Prosciutto,2023-06-05,USD,7900.00
6,Boreal Butchers,CA,Vancouver,Meat,Canadian Bacon,2023-06-06,CAD,15500.00
7,Trillium Trimmings,CA,Toronto,Meat,Smoked Duck Breast,2023-06-07,CAD,16200.00
8,The Montreal Meaterie,CA,Montreal,Meat,Montreal Smoked Meat,2023-06-08,CAD,17500.00
9,Carnes de la Capital,MX,Mexico City,Meat,Texas-Style Brisket,2023-06-09,MXN,185000.00
```

id	company_name	item	currency	bill_amount	bill_amount_usd
1	The Beef Baron	Wagyu Beef	USD	5100.00	5100.00
2	Longhorn Delicacies	USDA Prime Ribeye	USD	6800.00	6800.00
3	Boreal Butchers	Canadian Bacon	CAD	15500.00	12245.00
4	Carnes de la Capital	Texas-Style Brisket	MXN	185000.00	9250.00

DigitalCloud TRAINING

Automating RDS with Lambda HOL



Automating RDS with Lambda HOL

Overview:

Configure a Lambda function that's triggered whenever a new billing CSV file is uploaded to an S3 bucket. The function will read the file, convert any billing amount not in USD into USD, and insert the records into an Aurora Serverless V1 database.

Prerequisites:

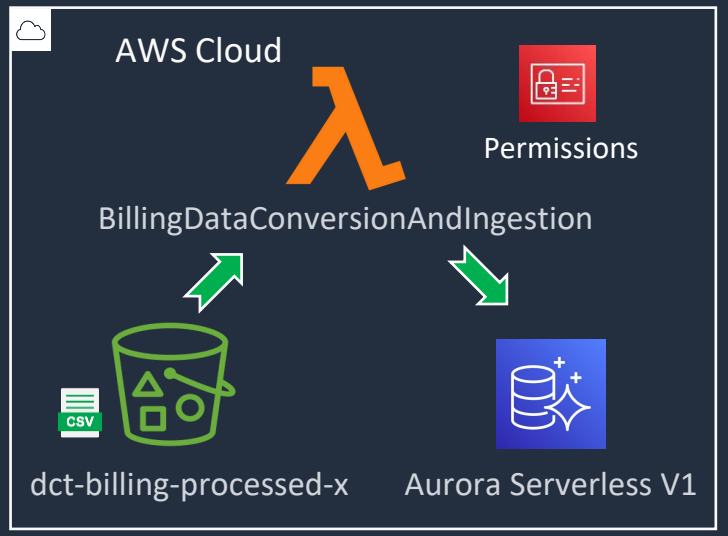
1. Create an Aurora Serverless V1 database cluster
   a) Create the DB cluster we generated in the "RDS Hands-on Lab" from the "AWS Services using Python" section of this course
2. Boilerplate (default) Python Lambda function 'BillingDataConversionAndIngestion' downloaded into Cloud9 environment.
3. An S3 bucket with default settings for uploading CSV files: 'dct-billing-processed-x'.
4. Download the provided CSV test files.

Tasks:

1.Write and test the Lambda Python code in Cloud9 IDE.
2.Add necessary permissions to the Lambda execution role.
   1. Access to S3 to read the input CSV file.
   2. Access to RDS to write data to Aurora Serverless.
   3. Access to Secrets Manager to retrieve database credentials.
3.Set up the S3 bucket trigger to invoke the Lambda function whenever a new CSV file is uploaded.
4.Test the Lambda function by uploading a CSV file to the 'dct-billing-processed-x' S3 bucket and verify the data insertion in Aurora Serverless database.

```
id,company_name,country,city,product_line,item,bill_date,currency,bill_amount
1,The Beef Baron,US,San Antonio,Meat,Wagyu Beef,2023-06-01,USD,5100.00
2,Longhorn Delicacies,US,Dallas,Meat,USDA Prime Ribeye,2023-06-02,USD,6800.00
3,Bison Bites,US,Houston,Meat,Bison Burgers,2023-06-03,USD,6300.00
4,The Windy City Butcher,US,Chicago,Meat,Chicago-style Hot Dogs,2023-06-04,USD,5600.00
5,East Coast Charcuterie,US,New York,Meat,Prosciutto,2023-06-05,USD,7900.00
6,Boreal Butchers,CA,Vancouver,Meat,Canadian Bacon,2023-06-06,CAD,15500.00
7,Trillium Trimmings,CA,Toronto,Meat,Smoked Duck Breast,2023-06-07,CAD,16200.00
8,The Montreal Meaterie,CA,Montreal,Meat,Montreal Smoked Meat,2023-06-08,CAD,17500.00
9,Carnes de la Capital,MX,Mexico City,Meat,Texas-Style Brisket,2023-06-09,MXN,185000.00
```

CSV Billing File



Automating VPC Operations with Lambda

Automating VPC Operations with Lambda

Why combine Lambda with VPC Operations?

Event-Driven Network Operations

- Update network configurations in response to changes in other AWS resources
- Automatically react to changes in network traffic or utilization

Improving Security

- Automate the enforcement of security rules and standards
- Respond to potential security threats or unusual network activities

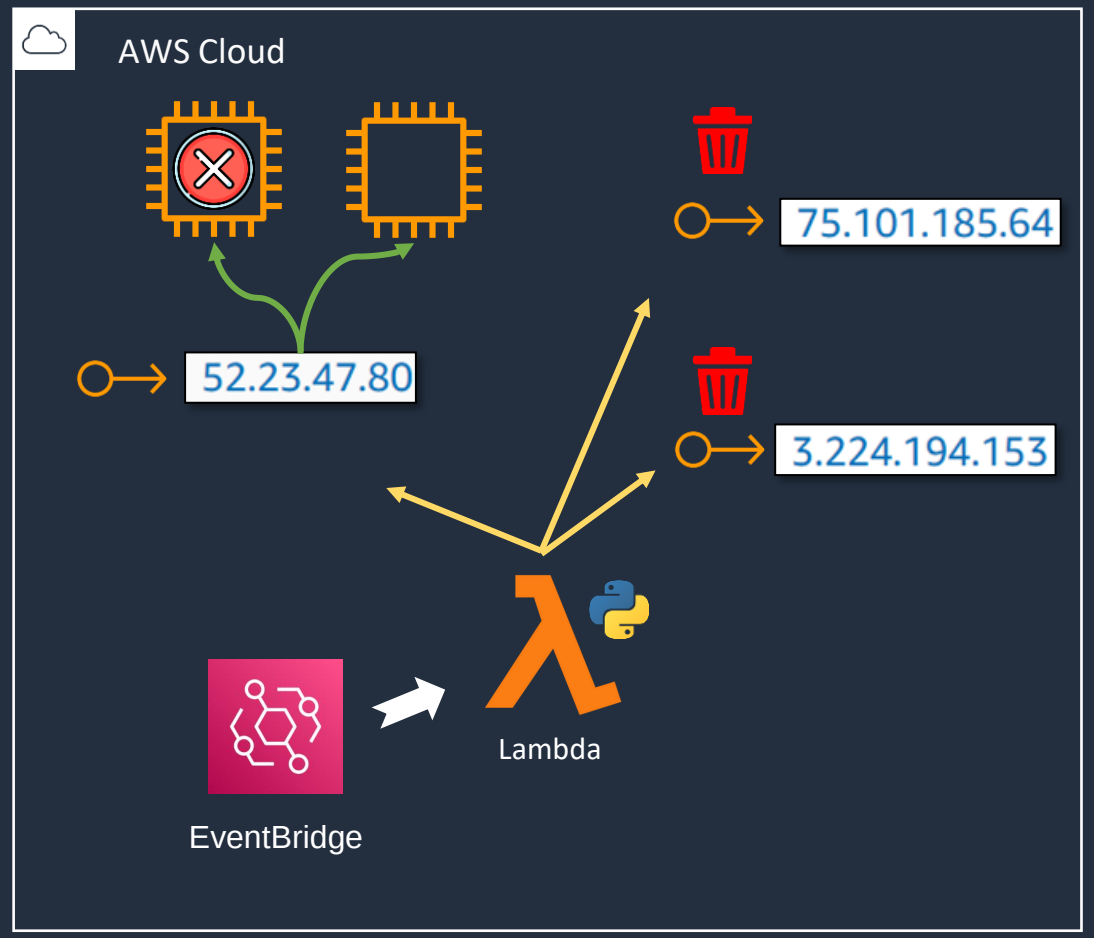
Automate Network Tasks

- Automatically adjust network Access Control Lists (ACLs)
- Create and delete subnets based on demand or schedules
- Configuring security group rules to adapt to changes in your environment
- Ensure optimal usage of our resources

Automating Elastic IP Management with AWS Lambda

Elastic IPs (EIPs)

- Static, public IPv4 addresses that you can allocate to your account
- To use an Elastic IP address, you first allocate one to your account, and then associate it with a network interface or an instance
- You can use an EIP to mask the failure of an instance or software by rapidly remapping the address to another instance in your account
- While EIPs are free when they are associated with an instance, there is a small hourly charge when an EIP is not being used



Example Scenario:

- You have several Elastic IPs that are not currently associated with an instance and result in unnecessary costs
- A Lambda function
  - checking for any unassociated Elastic Ips
  - Releasing them to save costs
- Triggered by EventBridge every day
- This demonstrates how we can automate and streamline VPC operations and tasks using Lambda

Applying Lambda & VPC: Real-World Scenarios

Real Use-Cases

Scalability & Efficiency

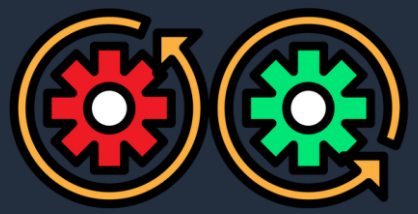
Lambda can be set to automatically adjust the number of instances within a VPC based on the current load. This ensures you're using resources optimally.

- Peak usage times: launch additional instances
- Off-peak time: terminate unnecessary instances
- Keeping costs in check

Security

Lambda can be used to automate security checks, such as verifying that all Security Groups in a VPC follow organizational guidelines.
If a security group is found not in compliance, Lambda can automatically send notifications or even adjust the rules to match the standards.

DevOps



Teams can use Lambda to automatically assign or unassign Elastic IPs to instances when they are launched or terminated, ensuring optimal resource utilization and cost-efficiency.

Summary

- How Lambda can automate VPC operations
  - Automate tasks
  - Increase efficiency
  - Improve security
- We dove into a practical example of managing Elastic IPs automatically
- The combination of Lambda and VPC opens a host of possibilities for automating tasks within your environment and allowing your teams to focus on what matters most.

DigitalCloud TRAINING

Automating VPC with Lambda HOL

DigitalCloud TRAINING

Automating VPC Operations with Lambda HOL

Scenario:

Your company regularly uses Elastic IPs for EC2 instances but at times some of these IPs are left unassociated and result in unnecessary costs. Your task is to automate the process of checking for these unassociated Elastic IPs and releasing them.

Overview:

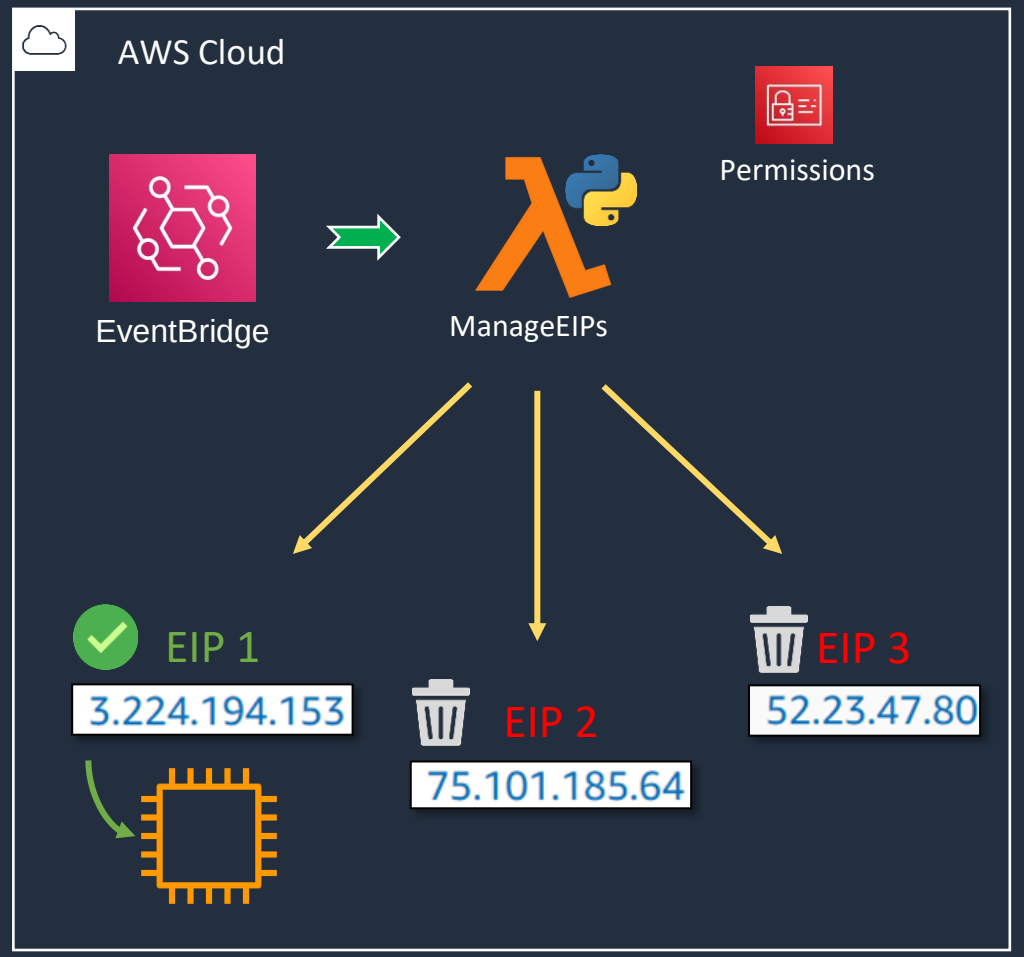
We'll create a Lambda function that is triggered daily by EventBridge. The Lambda function will check for any unassociated Elastic IP addresses in our VPC and **release them**. This way, we ensure we don't incur unnecessary costs from unused Elastic IPs.

Prerequisites:

1. Boilerplate (default) Python Lambda function 'ManageEIPs' downloaded into Cloud9 environment.

Tasks:

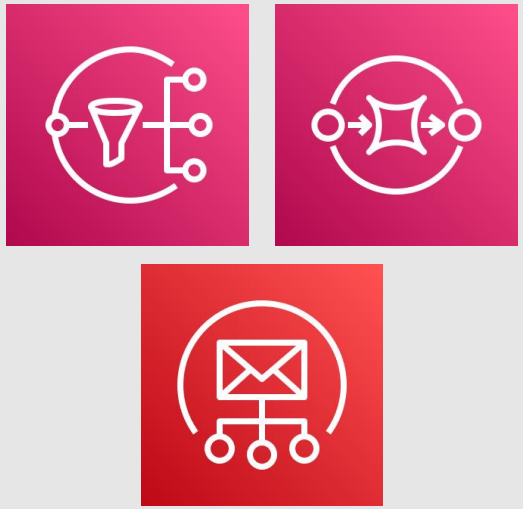
1. Create three EIPs and one EC2 instance. Associate one of the EIPs with the EC2 instance
2. Write and test the Lambda Python code in Cloud9 IDE.
3. Upload the local Cloud9 'ManageEIP' version of our function to Lambda
4. Add the necessary EC2 permissions to the Lambda execution role.
5. Set up the EventBridge trigger and test it



SECTION 6

Advanced AWS and Python Topics

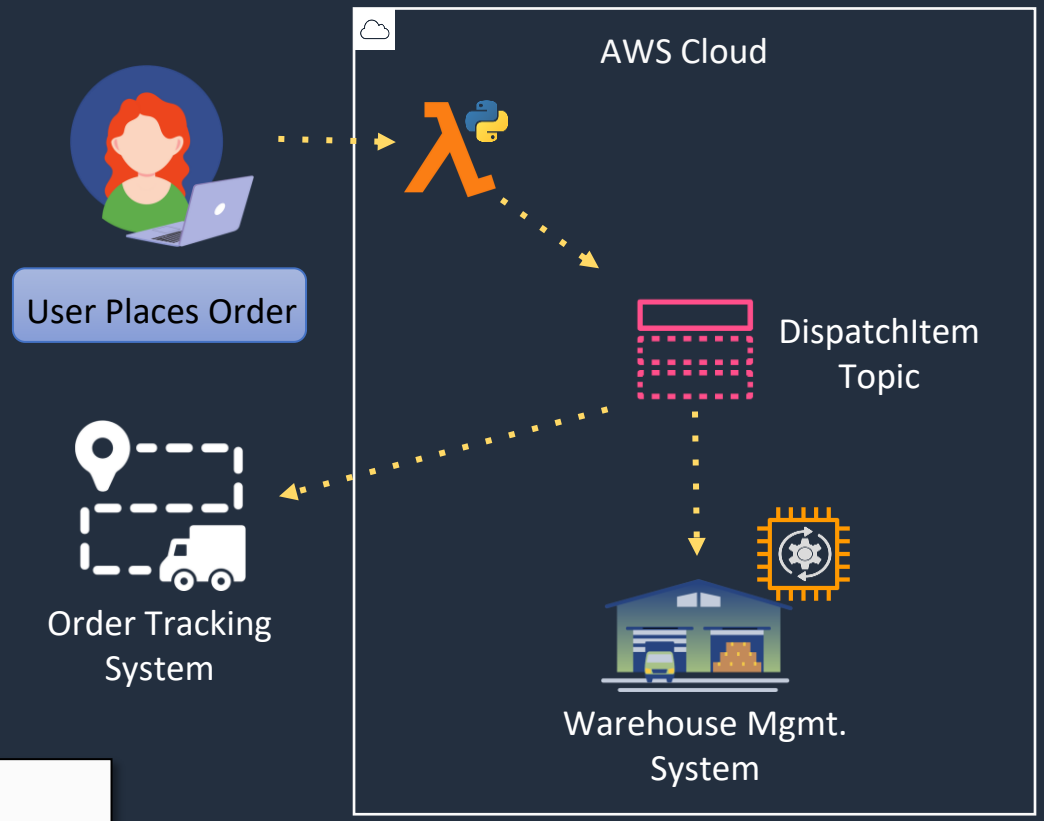
Understanding and Using SNS, SQS, and SES



Amazon Simple Notification Service (SNS)

What is SNS?

- A fully managed messaging service for communication
  - Application-to-application (A2A)
  - Application-to-person (A2P)
- Supports the publishing of messages to multiple subscribing endpoints or clients
- SNS Topic
  - A logical access point and communication channel



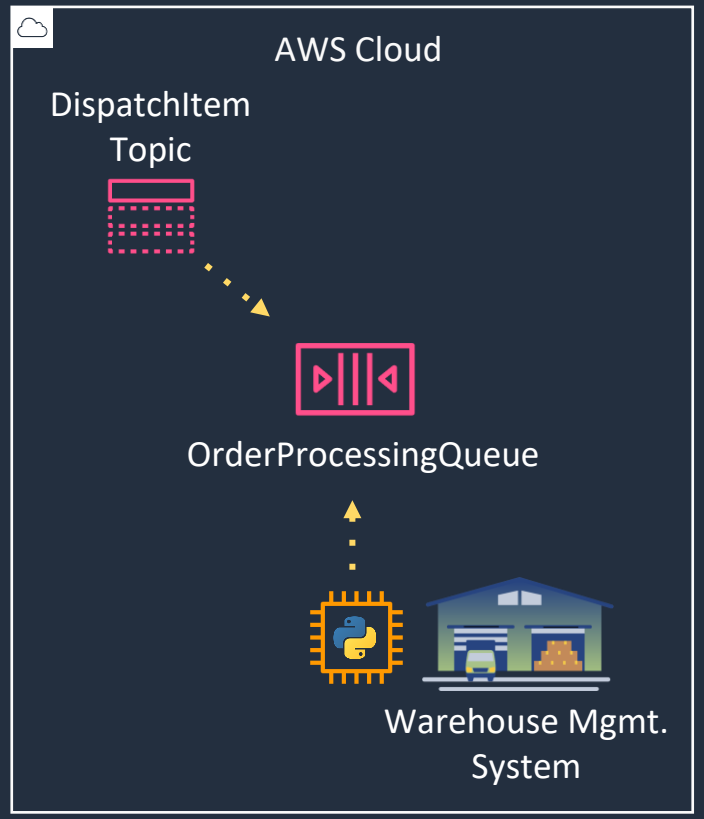

```
sns = boto3.client('sns')
response = sns.publish(
    TopicArn='arn:aws:sns:us-east-1:123456789012:DispatchItem',
    Message="Order ID: 12345, Item ID: ABC123 needs to be dispatched",
    Subject="New Order 12345"
)
```

Amazon Simple Queue Service (SQS)

What is SQS?

- A fully managed message queuing service that enables you to decouple and scale
  - Microservices
  - Distributed systems
  - Serverless applications
- Eliminates the complexity and overhead associated with managing and operating message-oriented middleware
- Stores messages on multiple servers for redundancy and to ensure message durability

```python
sqs = boto3.client('sqs')

while True:
    response = sqs.receive_message(
        QueueUrl='https://sqs.us-east-1.amazonaws.com/123456789012/OrderProcessingQueue',
        MaxNumberOfMessages=1,
        WaitTimeSeconds=20
    )

    if 'Messages' in response:
        message = response['Messages'][0]
        receipt_handle = message['ReceiptHandle']

        # Process the message here...

        sqs.delete_message(
            QueueUrl='https://sqs.us-east-1.amazonaws.com/123456789012/OrderProcessingQueue',
            ReceiptHandle=receipt_handle
        )
```

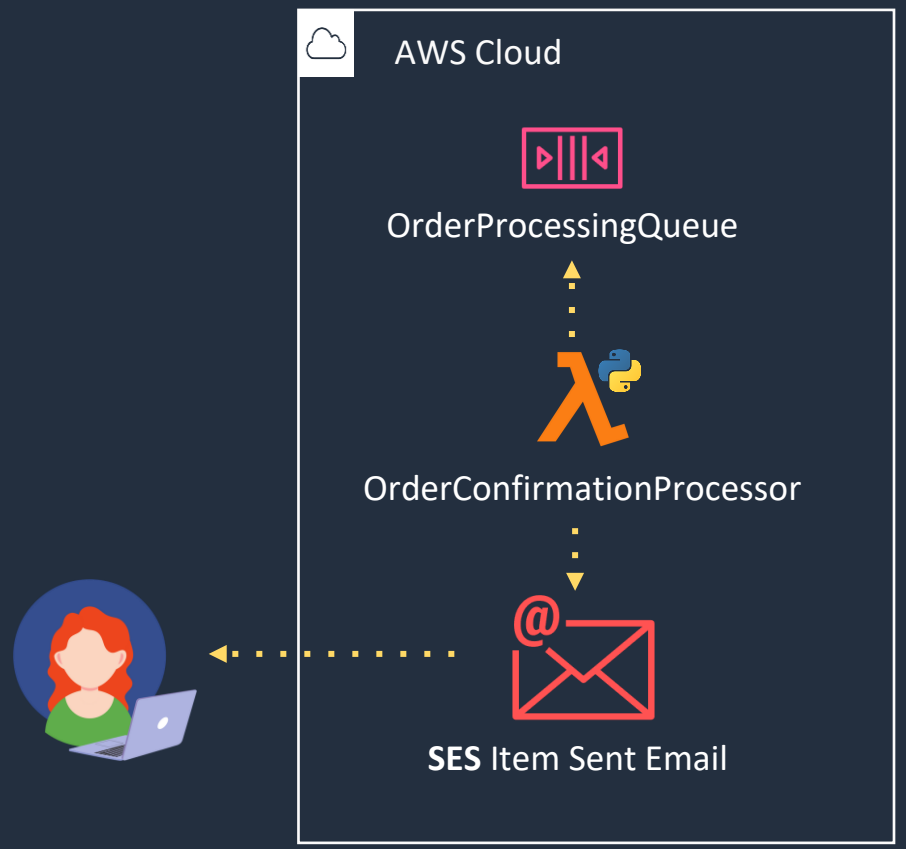


Amazon Simple Email Service (SES)

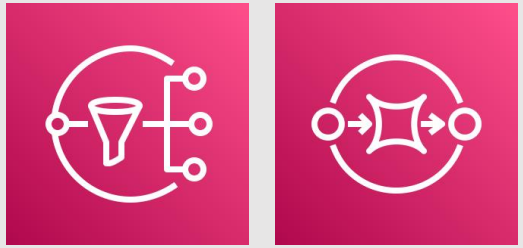
What is SES?

- An emailing service designed to send
  - Marketing
  - Notifications
  - Transactional emails
- Allows us to easily automate email notifications

```python
ses.send_email(
    Source='no-reply@ecommerce-example.com',
    Destination={
            'ToAddresses': [order_details['customer_email']]
        },
        Message={
            'Subject': {
                'Data': email_subject,
                'Charset': 'UTF-8'
            },
            'Body': {
                'Text': {
                    'Data': email_body,
                    'Charset': 'UTF-8'
                }
            }
        }
)
```



SNS and SQS HOL



SNS and SQS Hands-On Lesson

Scenario:

Your company has a system for validating CSV billing files stored in an S3 bucket for North American customers. In order to get real-time international tax data, you need to make a 3rd party API call. Occasionally, errors might occur during calls to a 3rd party service, which will prevent the billing file from being processed correctly. Your task is to automate the process of handling these errors, notifying the relevant employees, and ensuring error-related information is stored for further investigation.

Overview:

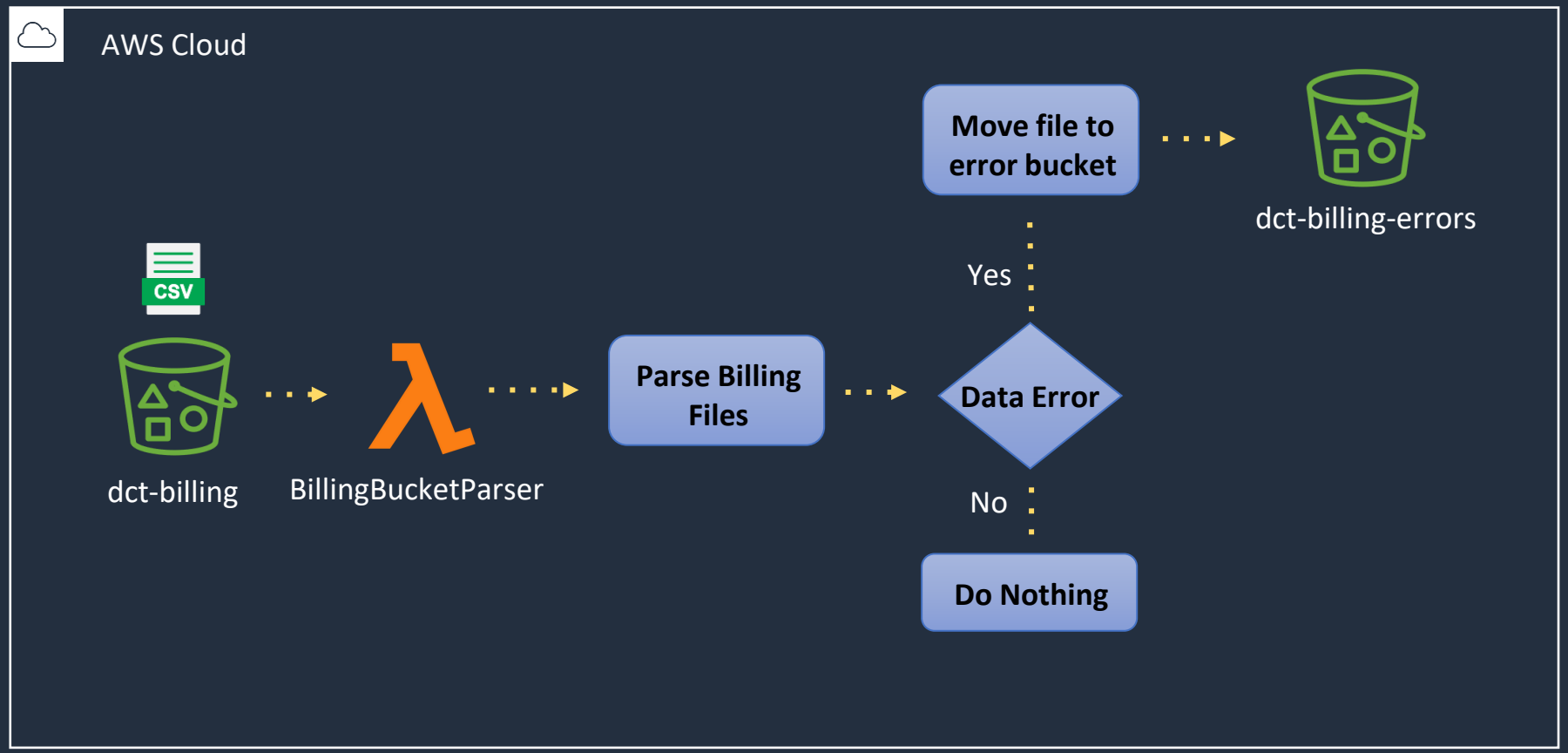
We'll modify the finalized version of the "Automating S3 with Lambda: Real-time Data Validation HOL" by creating an additional lambda 'RetryBillingParser' function, an SNS topic, and an SQS queue. If there is an error in the 'BillingBucketParser' function's mock API call to a 3rd party service 'BillingBucketParser' publishes to the SNS topic. An email is sent will be sent out and a subscribed SQS queue triggers the 'RetryBillingParser' function. This function re-attempts the data validation.

Prerequisites:

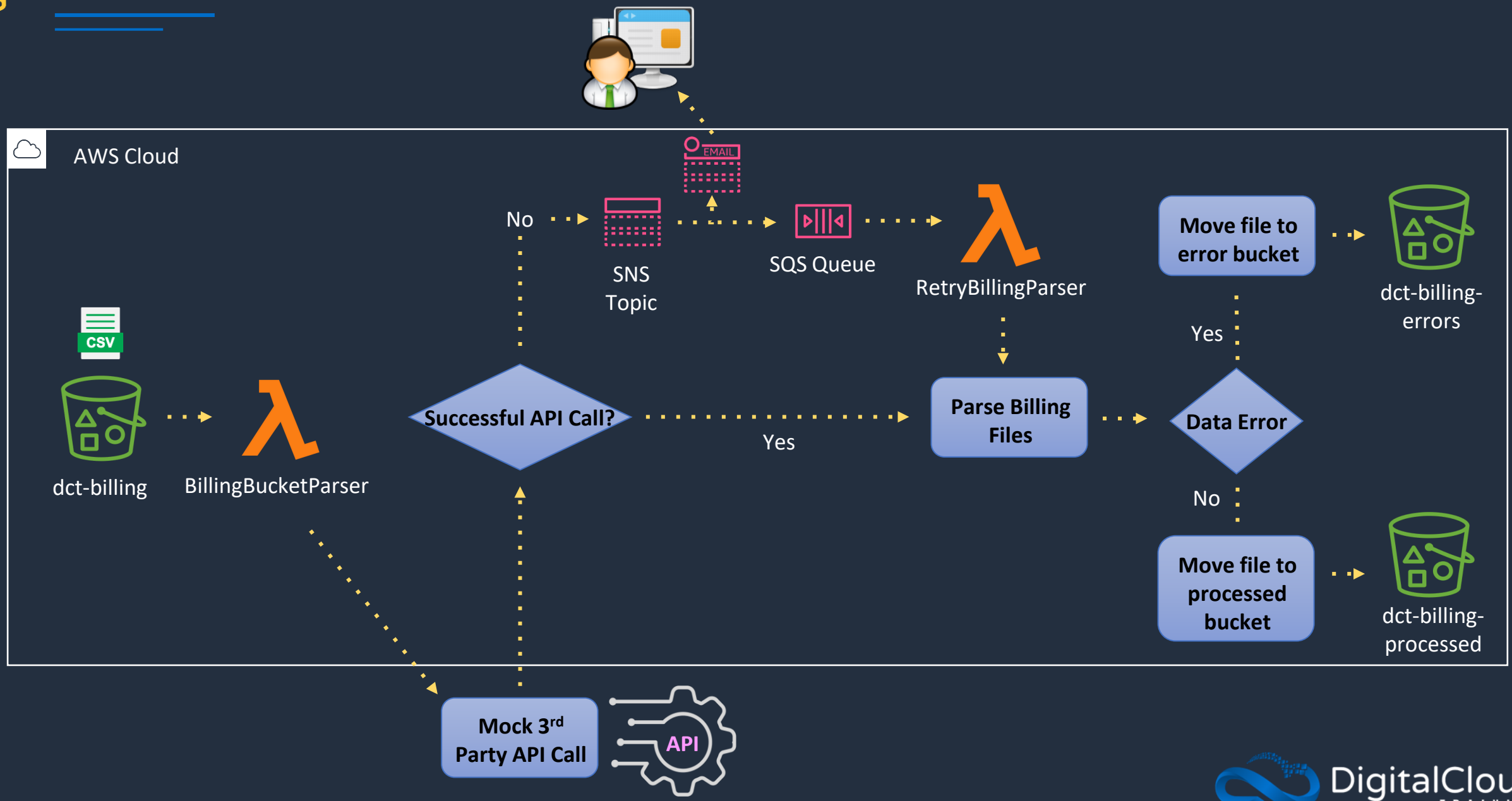
- Completed Automating S3 with Lambda: Real-time Data Validation HOL
- Boilerplate Python Lambda functions 'RetryBillingParser'
- S3 bucket 'dct-billing-processed'

SNS and SQS Hands-On Lesson

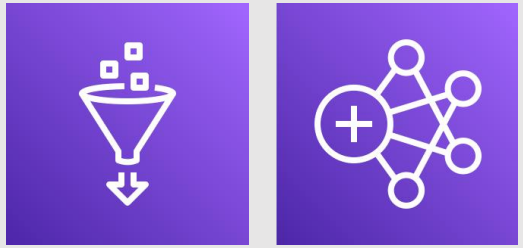
Existing S3 Real-time Data Validation HOL



SNS and SQS Hands-On Lesson



Understanding & Using Glue and EMR with Python



DigitalCloud TRAINING

Using AWS Glue

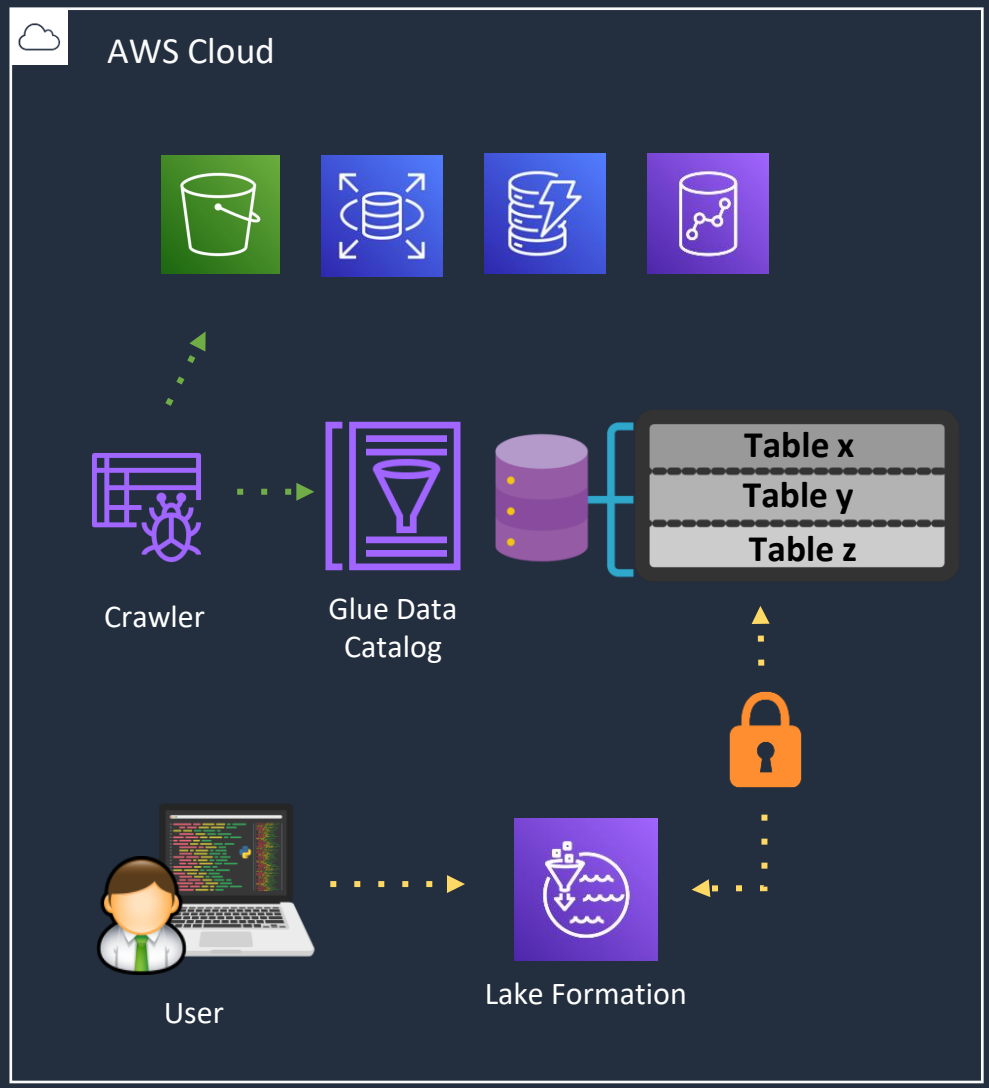
What is AWS Glue?

- Glue is a fully managed ETL (Extract, Transform, Load) service that simplifies the process of discovering and cataloging your data across various data stores
- At the heart of Glue is the Data Catalog, a centralized metadata repository

- Glue Crawlers connect to your source data, extract metadata and create table definitions in the Data Catalog
- Securing access to your data is crucial. Lake Formation simplifies the process of setting up, securing, and managing data
- Lake Formation ensures that only those users and services (such as AWS EMR) with granted permissions can access the data in your Data Catalog

Scenario:

- Consider a company that holds vast amounts of data across several S3 buckets. By using Glue Crawlers, they can automate the process of extracting, transforming, and loading this data for analytics
- With Lake Formation, they can secure their Data Catalog, making sure only authorized users have access to specific data



AWS Elastic MapReduce (EMR)

What is EMR?

A big data platform that enables businesses to process large datasets quickly and cost-effectively using popular frameworks such as Apache Hadoop and Apache Spark.

Some essential features of EMR include:

- Scalability
- Flexibility
- Security

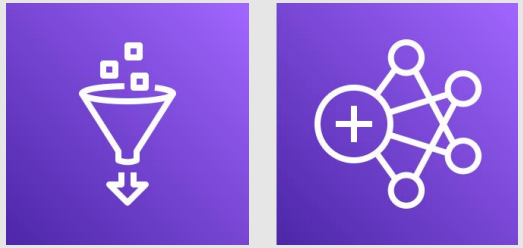
Apache Spark is an open-source, distributed computing system used for big data processing and analytics.

- PySpark is the Python library for Spark which allows for high-throughput data processing
- PySpark and EMR form a powerful toolset for big data analytics
- Use cases include log analysis, data transformations, financial analysis, machine learning, etc....
- We can use Boto3 to create an EMR cluster, add steps (tasks) to the cluster, check the status of the steps, and terminate clusters

EMR Steps


```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Create a SparkSession with Hive support enabled
spark = (SparkSession.builder.appName('data_processing')
        .enableHiveSupport().getOrCreate())

# Set the current Hive database
spark.sql("use `movie_db`")

# Read data from the 'movies' table
movies_df = spark.table('movies')

# Read 'ratings' table and filter out where 'rating' is null
ratings_df = (spark.table('ratings')
              .filter(col('rating').isNotNull()))

# Join the two DataFrames
joined_df = (movies_df.join(ratings_df,
                movies_df.movie_id == ratings_df.movie_id)
                .drop(ratings_df.movie_id))

# Calculate the average rating for each movie
average_rating_df = (joined_df.groupBy('movie_id', 'movie_name')
                     .agg(avg('rating').alias('average_rating')))

# Write the result DataFrame to an S3 bucket in CSV format
(average_rating_df
    .write.option('header', 'true')
    .csv('s3://movie-db-data-lake/reports/average-ratings'))
```

Glue and EMR Hands-On Lesson



Glue and EMR Hands-On Lesson

Scenario:

Your company processes a multitude of transaction files daily which detail billing, products sold, and their quantities. This data is stored in different CSV files within S3. There's a need to consolidate this data, analyze it, and derive insights like the gross profit. You are tasked with automating the data processing and analytics pipeline using AWS Glue and EMR.

Overview:

1. Store the raw CSV data in S3 buckets
2. Use Glue Crawlers to catalog this data
3. Spin up an EMR cluster, which we'll use to run a PySpark script
   a) This script will read the data from the Glue catalog and determine the gross profit for each product sold
   b) Store gross profit results back into S3 for reporting

Prerequisites:

Download the provided files:

- emr_software_settings.json
- billing_data_dairy_07_2023.csv
- units_sold_07_2023.csv
- production_costs_07_2023.csv

S3 buckets:

- dct-billing-processed
- dct-billing-data-lake-x
  - units-sold/
  - production-costs/
  - reports/
  - pyspark_script/

```
company_id,item_sold,units_sold
1,Artisan Cheese,900
2,Grass-Fed Butter,1050
3,Gourmet Yogurt,1200
4,Blue Cheese,800
```

units_sold_07_2023.csv

```
item,cost_per_unit_usd
Artisan Cheese,2.5
Grass-Fed Butter,2.2
Gourmet Yogurt,2.0
Blue Cheese,3.0
```

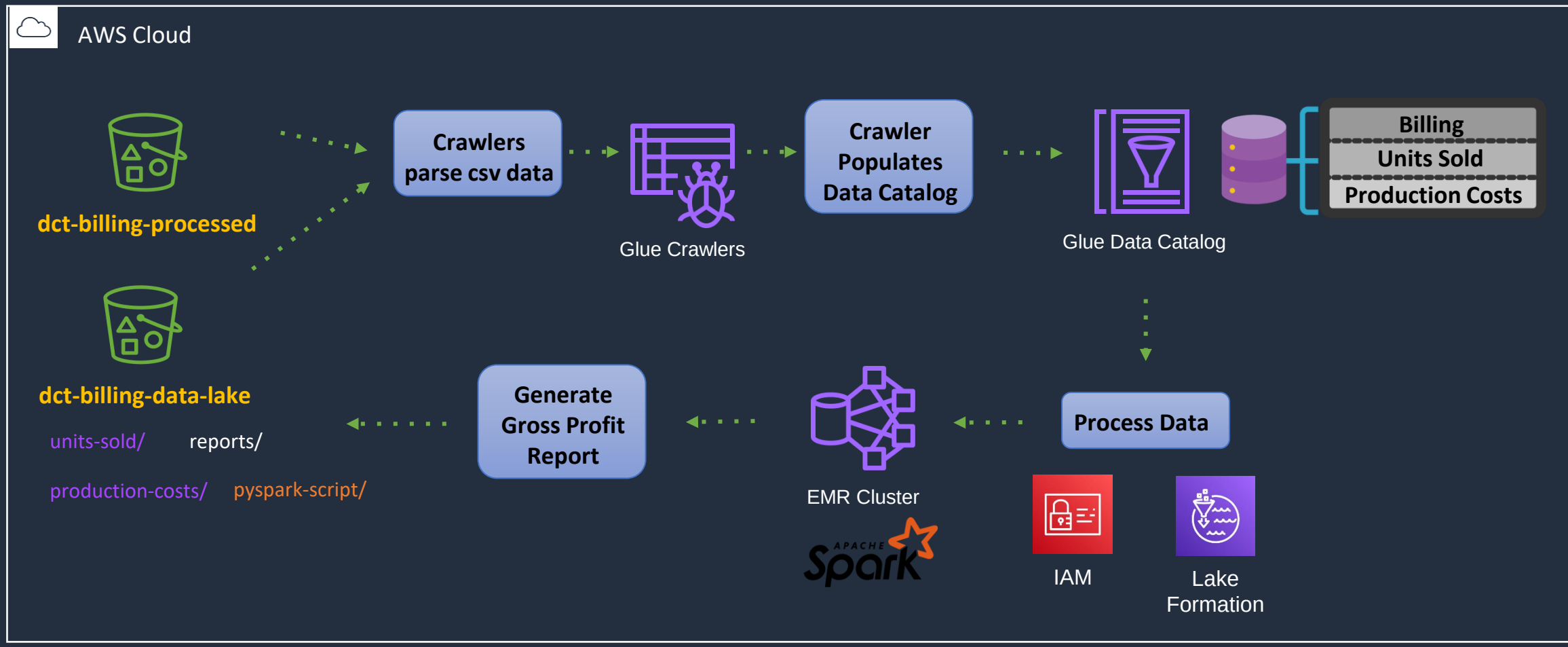
production_cost_07_2023.csv

```
id,company_name,country,city,product_line,item,bill_date,currency,bill_amount
1,Lone Star Lactose,US,San Antonio,Dairy,Artisan Cheese,2023-07-01,USD,3525.00
2,Bluebonnet Butter,US,Dallas,Dairy,Grass-Fed Butter,2023-07-02,USD,3685.00
3,Houston Creamery,US,Houston,Dairy,Gourmet Yogurt,2023-07-03,USD,3850.00
4,Chicago Cheesemonger,US,Chicago,Dairy,Blue Cheese,2023-07-04,USD,3950.00
5,Manhattan Milk,US,New York,Dairy,Organic Milk,2023-07-05,USD,4050.00
```

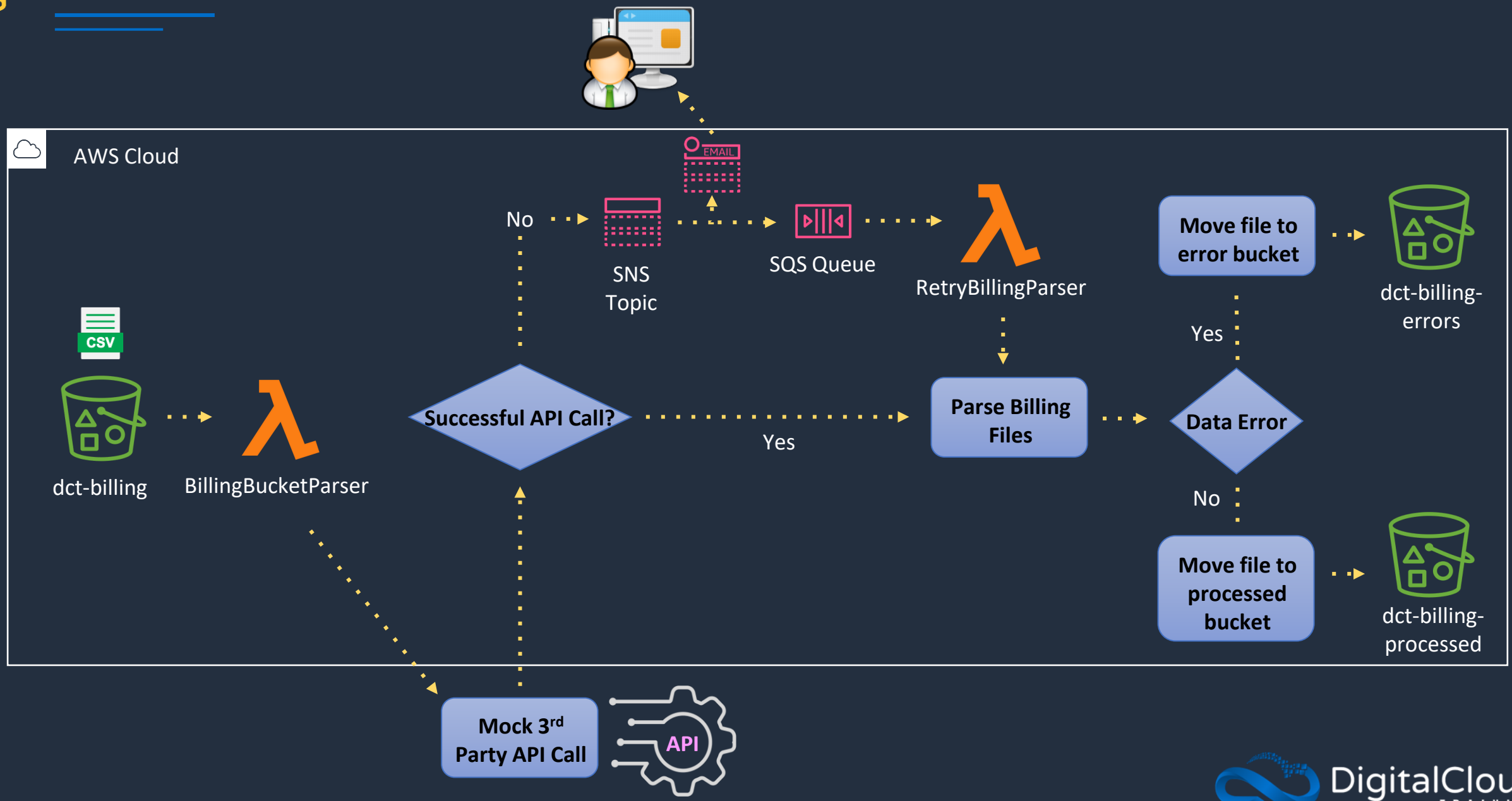
billing_data_dairy_07_2023.csv

DigitalCloud TRAINING

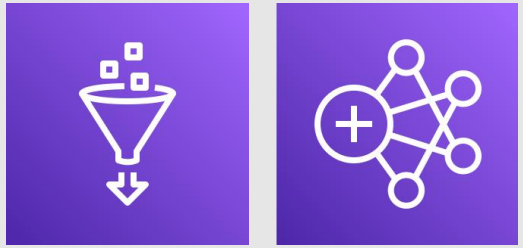
Glue and EMR Hands-On Lesson



SNS and SQS Hands-On Lesson



Understanding & Using Glue and EMR with Python



DigitalCloud TRAINING

Using AWS Glue

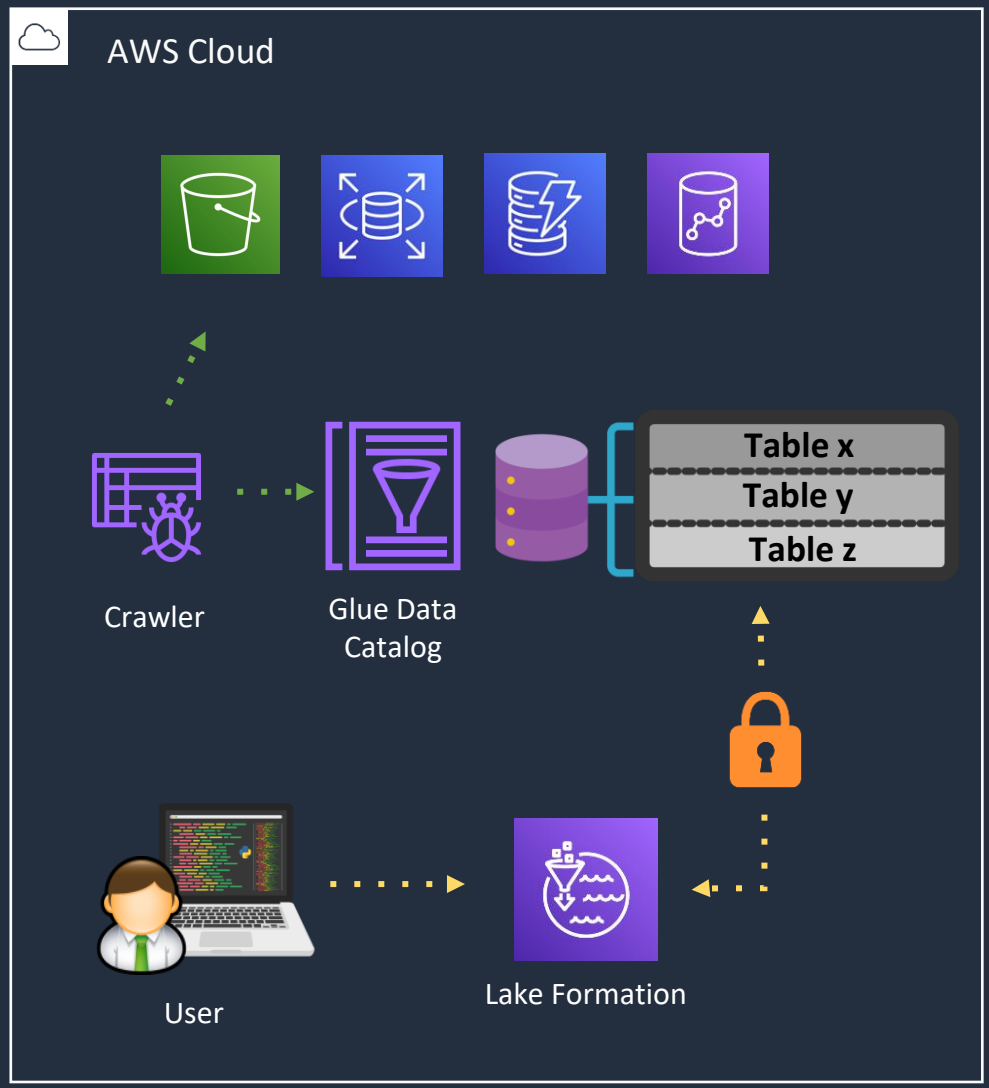
What is AWS Glue?

- Glue is a fully managed ETL (Extract, Transform, Load) service that simplifies the process of discovering and cataloging your data across various data stores
- At the heart of Glue is the Data Catalog, a centralized metadata repository

- Glue Crawlers connect to your source data, extract metadata and create table definitions in the Data Catalog
- Securing access to your data is crucial. Lake Formation simplifies the process of setting up, securing, and managing data
- Lake Formation ensures that only those users and services (such as AWS EMR) with granted permissions can access the data in your Data Catalog

Scenario:

- Consider a company that holds vast amounts of data across several S3 buckets. By using Glue Crawlers, they can automate the process of extracting, transforming, and loading this data for analytics
- With Lake Formation, they can secure their Data Catalog, making sure only authorized users have access to specific data



AWS Elastic MapReduce (EMR)

What is EMR?

A big data platform that enables businesses to process large datasets quickly and cost-effectively using popular frameworks such as Apache Hadoop and Apache Spark.

Some essential features of EMR include:

- Scalability
- Flexibility
- Security

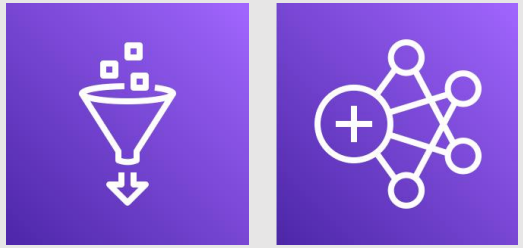
Apache Spark is an open-source, distributed computing system used for big data processing and analytics.

- PySpark is the Python library for Spark which allows for high-throughput data processing
- PySpark and EMR form a powerful toolset for big data analytics
- Use cases include log analysis, data transformations, financial analysis, machine learning, etc....
- We can use Boto3 to create an EMR cluster, add steps (tasks) to the cluster, check the status of the steps, and terminate clusters

EMR Steps


```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

# Create a SparkSession with Hive support enabled
spark = (SparkSession.builder.appName('data_processing')
        .enableHiveSupport().getOrCreate())

# Set the current Hive database
spark.sql("use `movie_db`")

# Read data from the 'movies' table
movies_df = spark.table('movies')

# Read 'ratings' table and filter out where 'rating' is null
ratings_df = (spark.table('ratings')
              .filter(col('rating').isNotNull()))

# Join the two DataFrames
joined_df = (movies_df.join(ratings_df,
                movies_df.movie_id == ratings_df.movie_id)
                .drop(ratings_df.movie_id))

# Calculate the average rating for each movie
average_rating_df = (joined_df.groupBy('movie_id', 'movie_name')
                     .agg(avg('rating').alias('average_rating')))

# Write the result DataFrame to an S3 bucket in CSV format
(average_rating_df
    .write.option('header', 'true')
    .csv('s3://movie-db-data-lake/reports/average-ratings'))
```

Glue and EMR Hands-On Lesson



Glue and EMR Hands-On Lesson

Scenario:

Your company processes a multitude of transaction files daily which detail billing, products sold, and their quantities. This data is stored in different CSV files within S3. There's a need to consolidate this data, analyze it, and derive insights like the gross profit. You are tasked with automating the data processing and analytics pipeline using AWS Glue and EMR.

Overview:

1. Store the raw CSV data in S3 buckets
2. Use Glue Crawlers to catalog this data
3. Spin up an EMR cluster, which we'll use to run a PySpark script
   a) This script will read the data from the Glue catalog and determine the gross profit for each product sold
   b) Store gross profit results back into S3 for reporting

Prerequisites:

Download the provided files:

- emr_software_settings.json
- billing_data_dairy_07_2023.csv
- units_sold_07_2023.csv
- production_costs_07_2023.csv

S3 buckets:

- dct-billing-processed
- dct-billing-data-lake-x
  - units-sold/
  - production-costs/
  - reports/
  - pyspark_script/

```
company_id,item_sold,units_sold
1,Artisan Cheese,900
2,Grass-Fed Butter,1050
3,Gourmet Yogurt,1200
4,Blue Cheese,800
```

units_sold_07_2023.csv

```
item,cost_per_unit_usd
Artisan Cheese,2.5
Grass-Fed Butter,2.2
Gourmet Yogurt,2.0
Blue Cheese,3.0
```

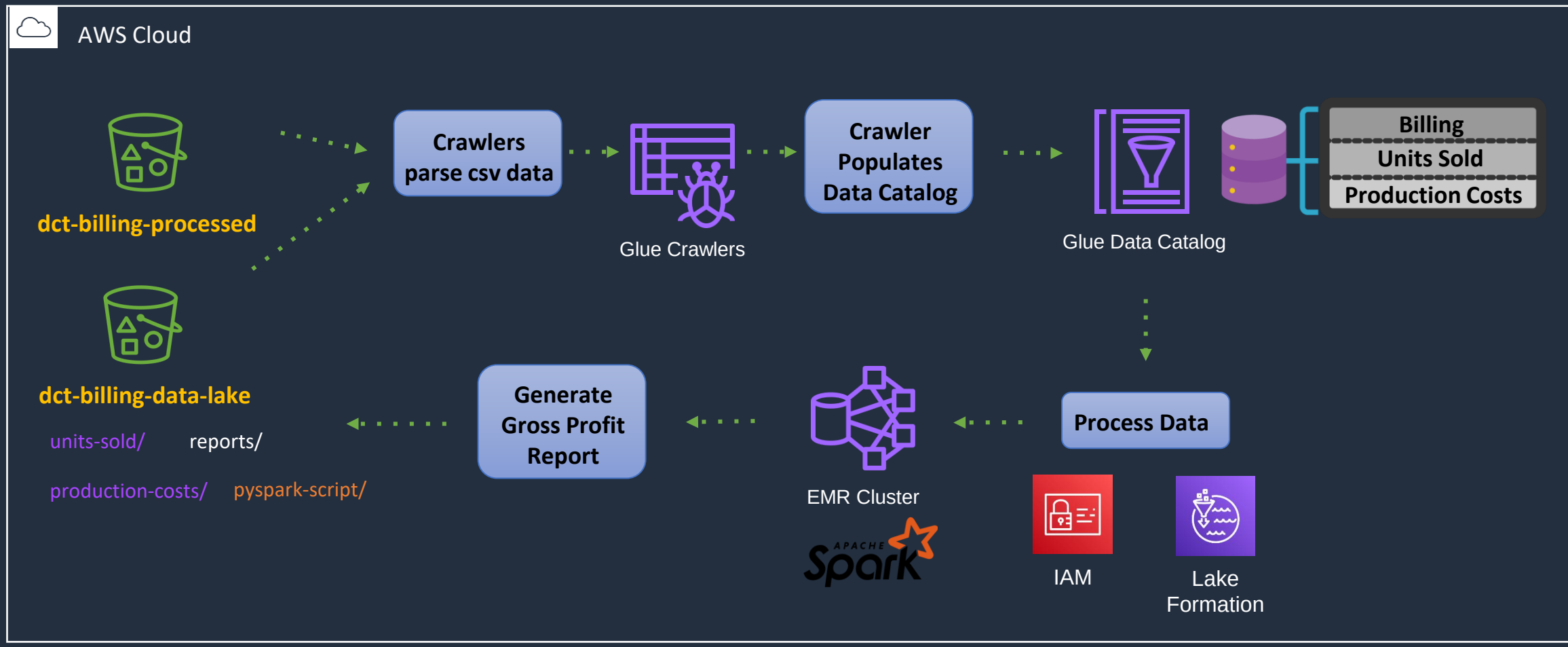
production_cost_07_2023.csv

```
id,company_name,country,city,product_line,item,bill_date,currency,bill_amount
1,Lone Star Lactose,US,San Antonio,Dairy,Artisan Cheese,2023-07-01,USD,3525.00
2,Bluebonnet Butter,US,Dallas,Dairy,Grass-Fed Butter,2023-07-02,USD,3685.00
3,Houston Creamery,US,Houston,Dairy,Gourmet Yogurt,2023-07-03,USD,3850.00
4,Chicago Cheesemonger,US,Chicago,Dairy,Blue Cheese,2023-07-04,USD,3950.00
5,Manhattan Milk,US,New York,Dairy,Organic Milk,2023-07-05,USD,4050.00
```

billing_data_dairy_07_2023.csv

DigitalCloud TRAINING

Glue and EMR Hands-On Lesson



Managing and Automating AWS Security with Python

DigitalCloud TRAINING

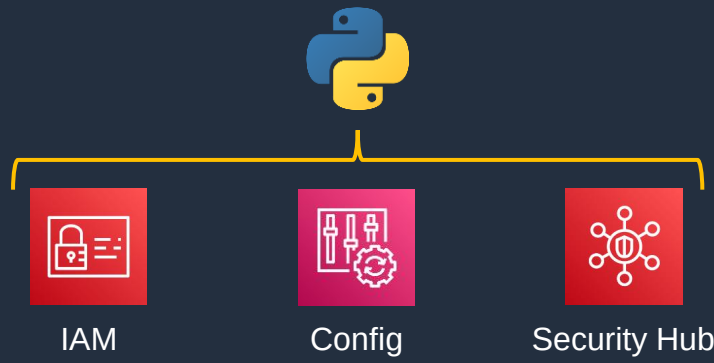
Managing Security in AWS with Python

Importance of AWS Security

Security in the cloud is a shared responsibility between the user and AWS. This distinction necessitates effective security management strategies.

Python and AWS Security

With the Boto3 SDK, Python can interact with AWS services programmatically, providing endless possibilities for enhancing security management tasks



Automating Security Tasks

Automation ensures that security checks and responses happen in real-time

Security Hub - Aggregates and prioritizes your security alerts or findings, from services such as:

- GuardDuty
- Inspector
- Macie

- You can automate the processing of security alerts and findings and respond to the potential security threats raised by these services

Managing Security with Lambda and Python

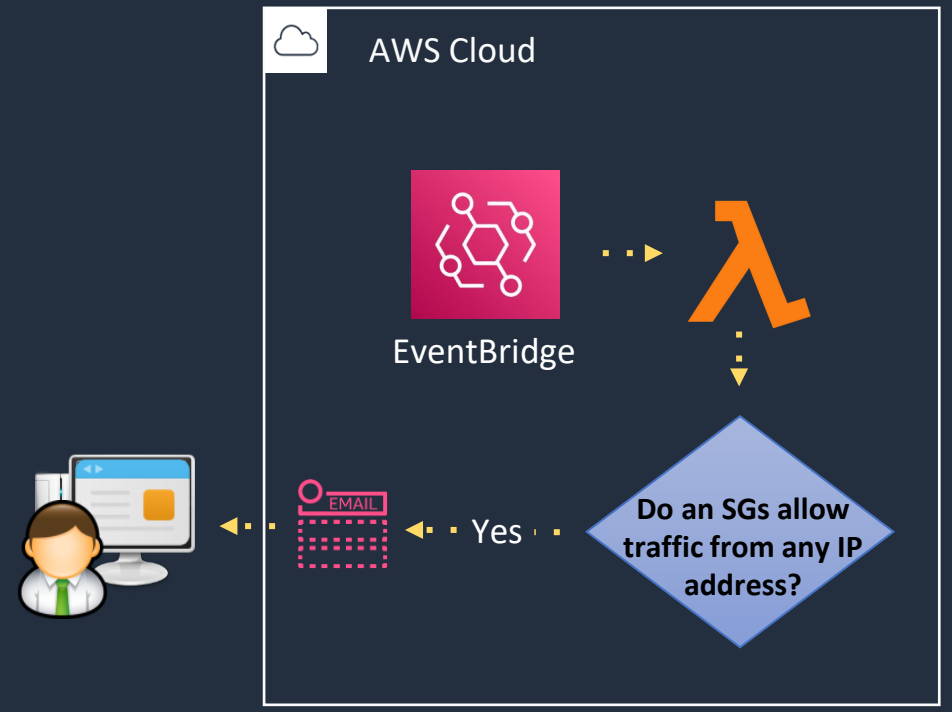
Python, coupled with AWS Lambda, provides a powerful platform for automating security tasks.

For example, if Security Hub detects an unauthorized access attempt, a Lambda function could automatically strengthen security group rules, revoke IAM credentials, or isolate affected resources.

```python
iam = boto3.client('iam')

def lambda_handler(event, context):
    finding = event['detail']['findings'][0]

    finding_type = finding['Title']
    severity = finding['ProductFields']['aws/securityhub/SeverityLabel']

    if 'UnauthorizedAccess' in finding_type and severity == 'HIGH':
        user_name = finding['Resources'][0]['Id']

        response = iam.delete_access_key(UserName=user_name,
                                         AccessKeyId='AKIAIOSFODNN7EXAMPLE')
```

Automated Security Audit System

Performs automatic audits of your AWS resources for potential security issues. Such as Security Group rules that allow traffic from any IP address (0.0.0.0/0).



Automated Security Auditing HOL

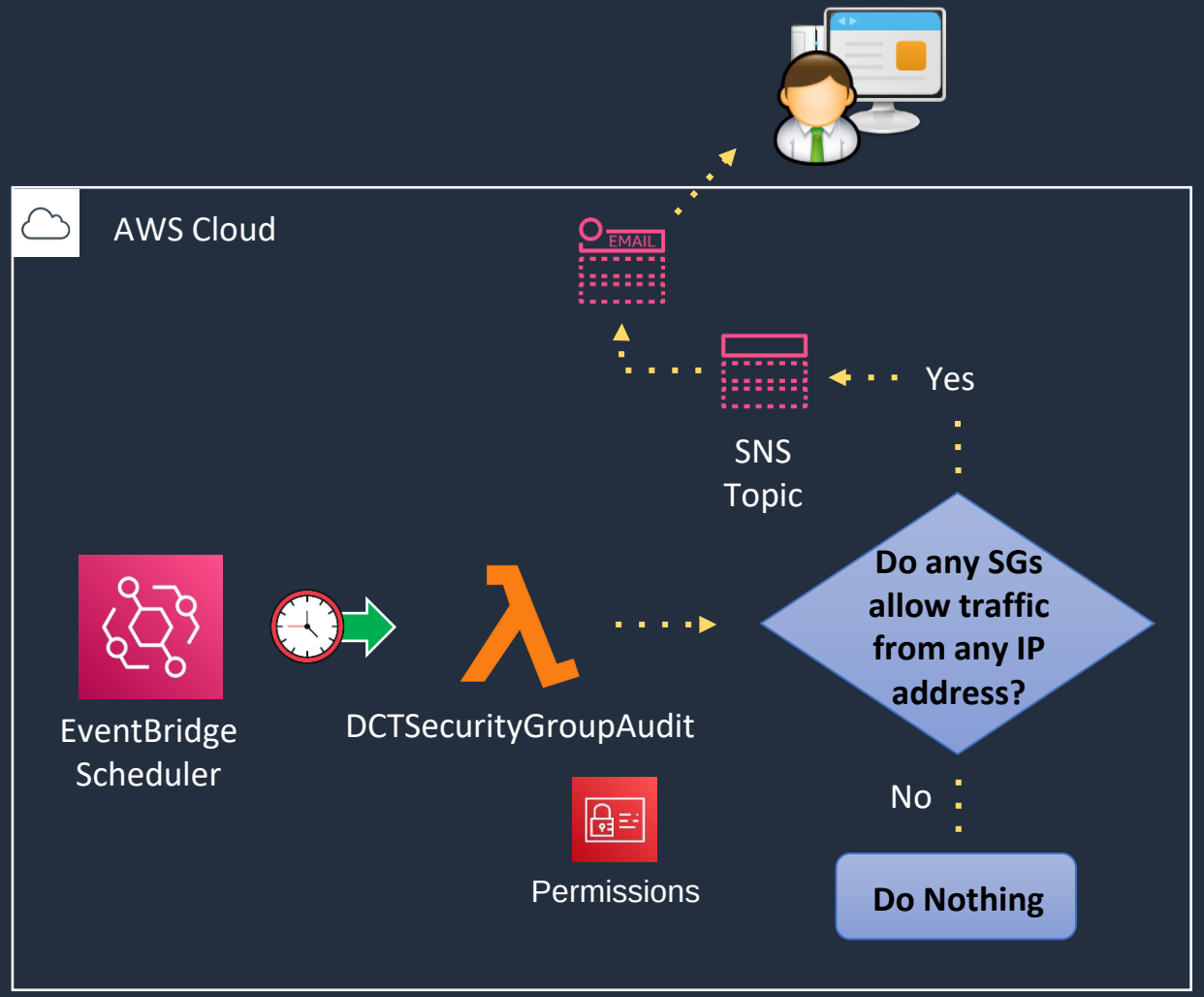
Automated Security Auditing Hands-On Lesson

Scenario:
Your company needs to create a system to audit AWS security groups for potential security issues, such as rules that allow traffic from any IP address (0.0.0.0/0).

Your task is to create a system that checks security groups automatically, compiles a report of its findings, and sends this report via email.

Prerequisites:

- Create Boilerplate (Default) Python Lambda function 'DCTSecurityGroupAudit'
- Download 'DCTSecurityGroupAudit' into Cloud9



SECTION 7

Leveraging Tools for Python and AWS Development

Pair Programming with GitHub Copilot

DigitalCloud TRAINING

Pair Programming with GitHub Copilot

Task:

Create a local python script with the help of Copilot that processes ‘Units Sold’ files and pushes them to S3

Prerequisites:

1. Download the provided files:
    1. units_sold_gh_copilot1.csv
    2. units_sold_gh_copilot2.csv

1. GitHub account

1. GitHub Copilot subscription (free trial)

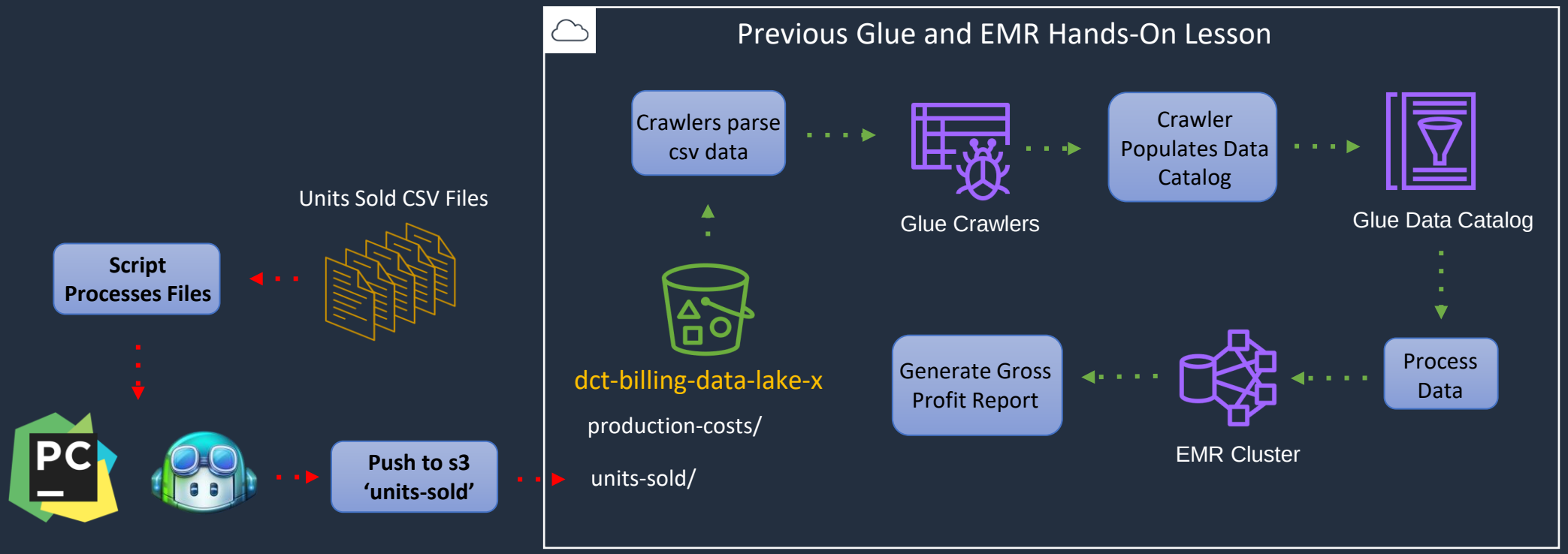
1. IDE which supports Copilot

1. Configure the AWS CLI:
    a) Check the “AWS CLI Setup and Configuration” lecture from Section 1
    b) https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

6. ‘dct-billing-data-lake-x’ bucket

6. ‘units-sold’ subfolder

Pair Programming with GitHub Copilot



Python for the AWS Cloud

DigitalCloud TRAINING