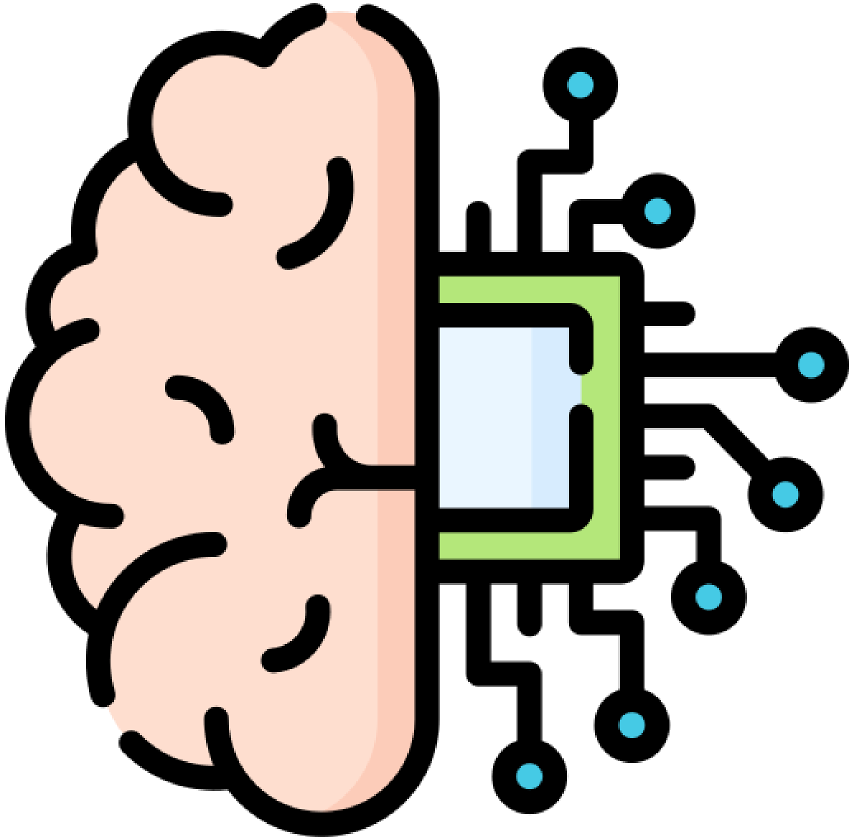
4-weeks Bootcamp

Introducing

Amazon Machine Learning



Instructor : THET SU WIN

On Boarding

Course Schedules

Details Schedules

No	Week	Course			
		Course Titles	Theory	Practical Labs	Discussion
1	Week 1 02-aug-2025	Intro to Cloud Computing	AWS services overview for ML (S3, EC2, IAM) ML lifecycle on AWS	AWS Free Tier setup (IAM, EC2, S3) Hands-on: launching EC2 , Setup IAM and creating S3 bucket, and usage of boto3	Details in Lecture Slides and Assignments
2	Week 2	Containerization & Serverless ML Pipelines	Intro to Docker + AWS ECR (Elastic Container Registry) Serverless Concepts	Build a simple ML inference container using Lambda, API gateway	Details in Lecture Slides and Assignments
3	Week 3	Big Data Processing with PySpark & EMR	What is Big Data? PySpark concepts and EMR overview	Local Spark Setup and data preprocessing using pyspark	Details in Lecture Slides and Assignments
4	Week 4	Model Development with SageMaker	Amazon SageMaker intro Built-in algorithms & Estimators & Inferencing	Tabular model training with built-in XGBoost , Creating the inferencing pipeline	Details in Lecture Slides and Assignments
5	Optional Lecture	Creating the RAG Application using AWS Services Creating the Orchestration pipelines of model training & Inferencing using AWS Stepfunction			

Declaimer : This 4-week bootcamp will not cover to understand 100% workflow of machine learning using AWS, but will have a sense of which AWS services are using and how to use it as a Data Scientist or Machine Learning Engineer.

Weekly Course Schedules

Overall Schedules

09/06	09/13	09/20	09/27	10/04
On Boarding	Week 1	Week 2	Week 3	Week 4 & Closing

Course References

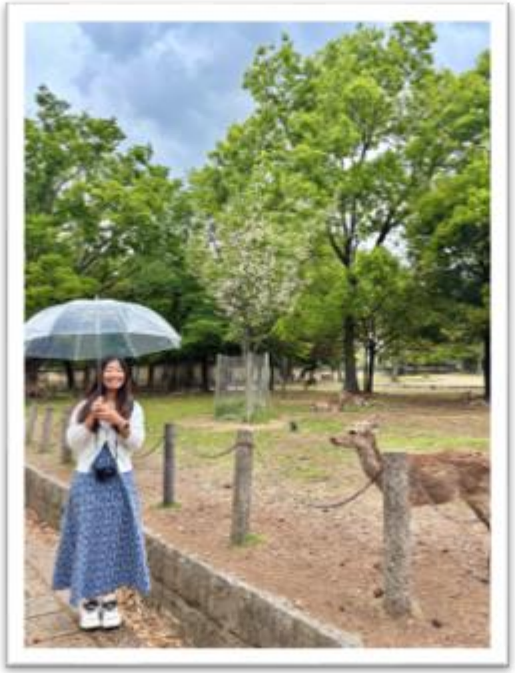
Udemy: AWS Machine Learning Specialty by Frak Kane and Stephane Maarek
AWS Documentation
Detailed citations and links will be provided in the lecture slides for each module.

Weekly Schedules

Sunday	Happy Weekend !!! (Nothing to do on this day)
Monday	New theory content and hands-on materials are released at 9:00PM (MMT)
Monday – Wednesday	study the theory and complete the case study assignment.
Thursday	Submit the case study for instructor feedback.
Friday	Instructor prepares for feedback
Saturday	Includes hands-on, discussion, feedback, and Q&A.

On Boarding

GET TO KNOW ME 👋



Hi, I'm Thet Su Win (Your Instructor for this bootcamp)

- **Data Scientist** at a cybersecurity company in Singapore
- **Master's Student** at School of Computing (AI Specialization), National University of Singapore
- **Instructor** at DATAVERSITY Myanmar, where I taught Python, AI, and Data Science
- Got AWS Machine Learning Specialty Certificate from AWS in 2023
- Over **5 years of experience** in AI & Data Science, with **7+ years** in the software industry
- Occasionally share content on **Medium** about AI and tech
- Passionate about **teaching**, especially to kids, and enjoy **traveling, reading**, and watching dramas
- **Fun Fact:** I left SageMaker running for 2 days and burned around **SGD 8,000**. (And yes, *I still somehow kept my job.)*

aws certified Machine Learning SPECIALTY

Connect with me

in https://www.linkedin.com/in/thet-su-win-169221172/

M Medium https://medium.com/@thetsuwin.tsw6

I'M HERE TO GUIDE, BUT ALSO TO LEARN WITH YOU — LET'S GROW TOGETHER AND MAKE THE MOST OF THESE 4 WEEKS!!

On Boarding

What You’ll need for this Bootcamp

1. AWS Account (Required)

- Please sign up at aws.amazon.com if you don’t have one
- Free Tier is enough for most of our hands-on exercises
- Make sure you can access **SageMaker, S3, Lambda, and IAM**

2. Basic Tools

- A laptop or desktop with **internet access**
- A modern browser (Chrome, Firefox, or Edge recommended)
- A GitHub account (optional, but useful for code sharing)
- Visual Studio Code or any code editor (optional for advanced practice)

3. Optional, but Helpful

- AWS CLI installed
- Python 3.x installed locally
- An IDE with terminal access (like VS Code or JupyterLab)

Things to Be Aware of

1. AWS is not 100% FREE

- While AWS **Free Tier** covers many services (S3, SageMaker Studio, Lambda), some actions (e.g., large training jobs, real-time endpoints) can incur **charges**
- We'll **stay within the Free Tier** in this bootcamp — but always check your usage on the **Billing Dashboard**

2. Don't share your AWS credentials

- Never expose or hardcode your **Access Keys or Secrets** in code
- If using notebooks, **use IAM roles or environment variables** to authenticate

3. Clean Up Resources After Use

- Some services like **SageMaker Notebook Instances** and **EC2** keep running and **incur cost** even when idle
- Always **stop or delete** your resources after each session
- I'll remind you at the end of every hands-on lab

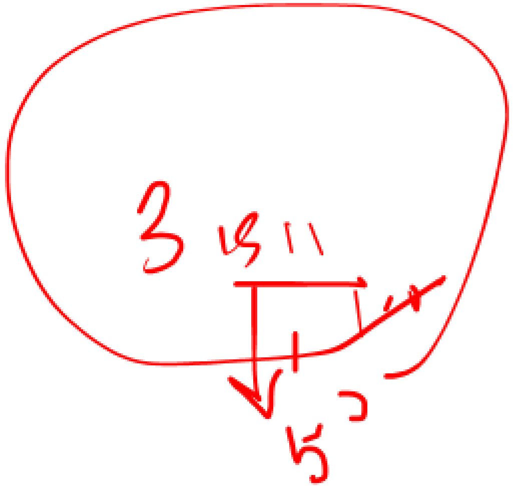
4. Region Matters

- Always use the same **AWS Region** (e.g., us-east-1) to avoid confusion
- Services and pricing vary by region — we'll stick to one region throughout

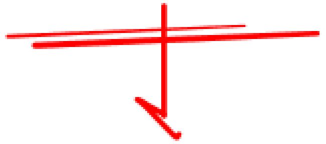
5. Use Bootcamp Resources ONLY for Learning

- Please don't use the AWS account for unrelated production tasks or heavy workloads
- This is for educational purposes only

I trust you'll use this access responsibly so everyone can benefit — let's make this smooth for all of us 🙌



AWS ML



Cloud

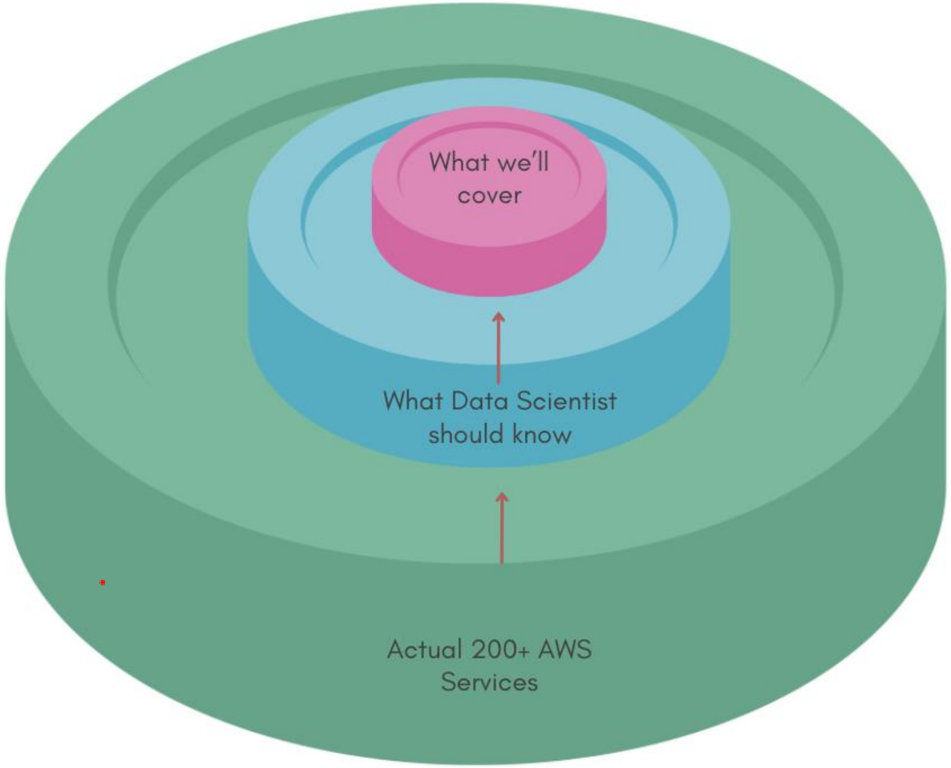
Let’s Dive In! 🚀

Ready to explore AWS Machine Learning?

What is Amazon Web Services (AWS) ?

AWS offers over **200+ services** across:

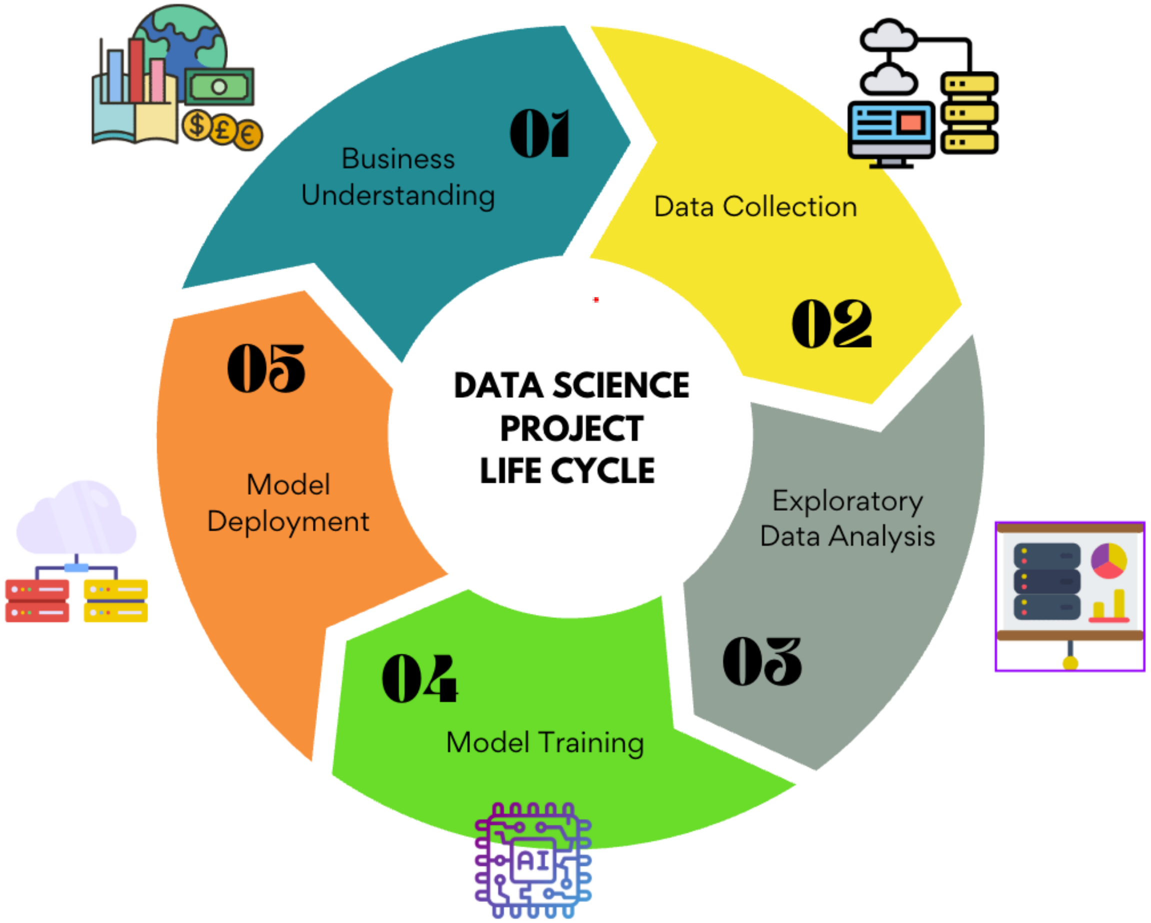
- Compute
- Storage
- Machine Learning
- Analytics
- Security
- IoT and more...



How many AWS services can you name off the top of your head?

(Drop it in the chat or shout it out!)

Data Science/Machine Learning Project Lifecycle



1. Business Understanding: Define the client's problem, background, use cases, and target users.

2. Data Collection: Gather data (in-house or through crawling), clean, sample, and possibly manually label it.

3. Exploratory Data Analysis (EDA): Analyze data quality, identify noise/outliers, and ensure suitability for training ("Garbage in, Garbage out").

4. Model Training: Select algorithms, train the model, and refine performance through **hyperparameter tuning**. If accuracy is low, revisit earlier stages for data or requirement issues.

5. Model Deployment: Release the finalized model to production (on-premise or cloud) and consider incremental training for new data.

https://medium.com/@thetsuwin.tsw6/data-science-machine-learning-project-life-cycle-8ec9e17d9d07

Data Science/Machine Learning Project Lifecycle *in AWS Environment*

	Life Cycle	AWS Services
1	Data Collection	Amazon S3 Amazon GLUE Amazon Kinesis Amazon IoT core
2	Data Processing	Amazon SageMaker Amazon GLUE Amazon Athena Amazon EMR Amazon RedShift
3	Exploratory Data Analysis	Amazon SageMaker Amazon Athena Amazon EMR amazon QuickSight
4	Model Training & Model Evaluation	Amazon SageMaker Amazon EC2 Amazon EMR Amazon EKS
5	Model Deployment	Amazon SageMaker Amazon API Gateway AWS Lambda Amazon EKS Amazon StepFunction

In this course, we'll cover

- **Amazon S3**
- **Amazon SageMaker**
- **Amazon EC2**
- **Amazon EMR (partially)**
- **Amazon Lambda**
- **Amazon API Gateway**
- **Amazon StepFunction**

Due to some cost limitation.

Don't worry if you didn't know the services, you *will* by the end of this bootcamp. For now, just start thinking like a Data Scientist using AWS.

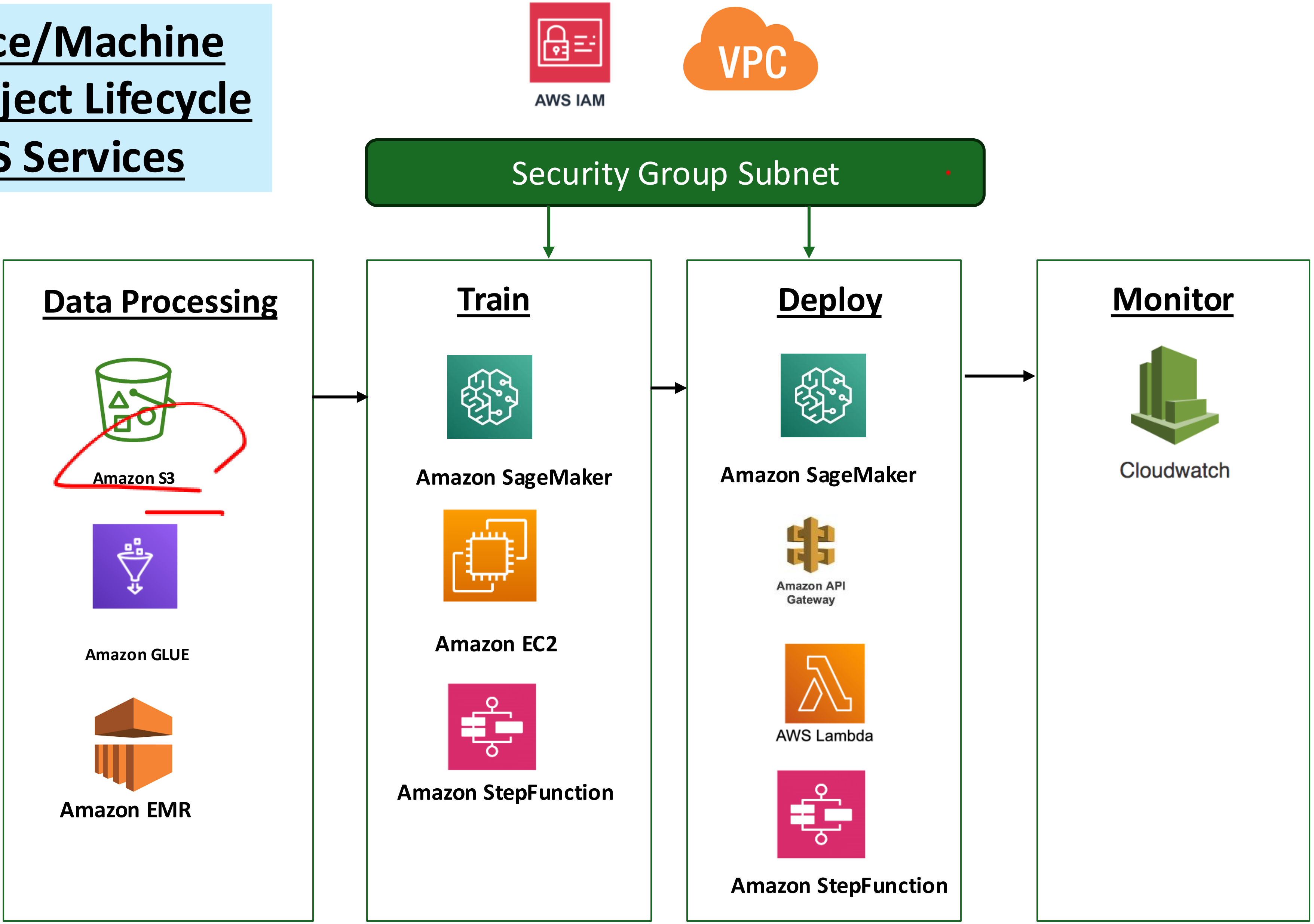
On Boarding

Additionally, you should know

Additional AWS Services

1	Security and Access Management	AWS IAM ✓ AWS KMS
2	Networking & Infrastructure	VPC ✓ Security group Instance
3	Workflow and Automation	✓ Cloudwatch Amazon EventBridge ✓ Codepipeline
4	Other Useful Tools	AWS ECR ✓

Data Science/Machine Learning Project Lifecycle with AWS Services



AWS IAM

VPC

Security Group Subnet

Data Processing

- Amazon S3
- Amazon GLUE
- Amazon EMR

Train

- Amazon SageMaker
- Amazon EC2
- Amazon StepFunction

Deploy

- Amazon SageMaker
- Amazon API Gateway
- AWS Lambda
- Amazon StepFunction

Monitor

- Cloudwatch

WEEK 1 : Intro to Cloud Computing

WEEK 1

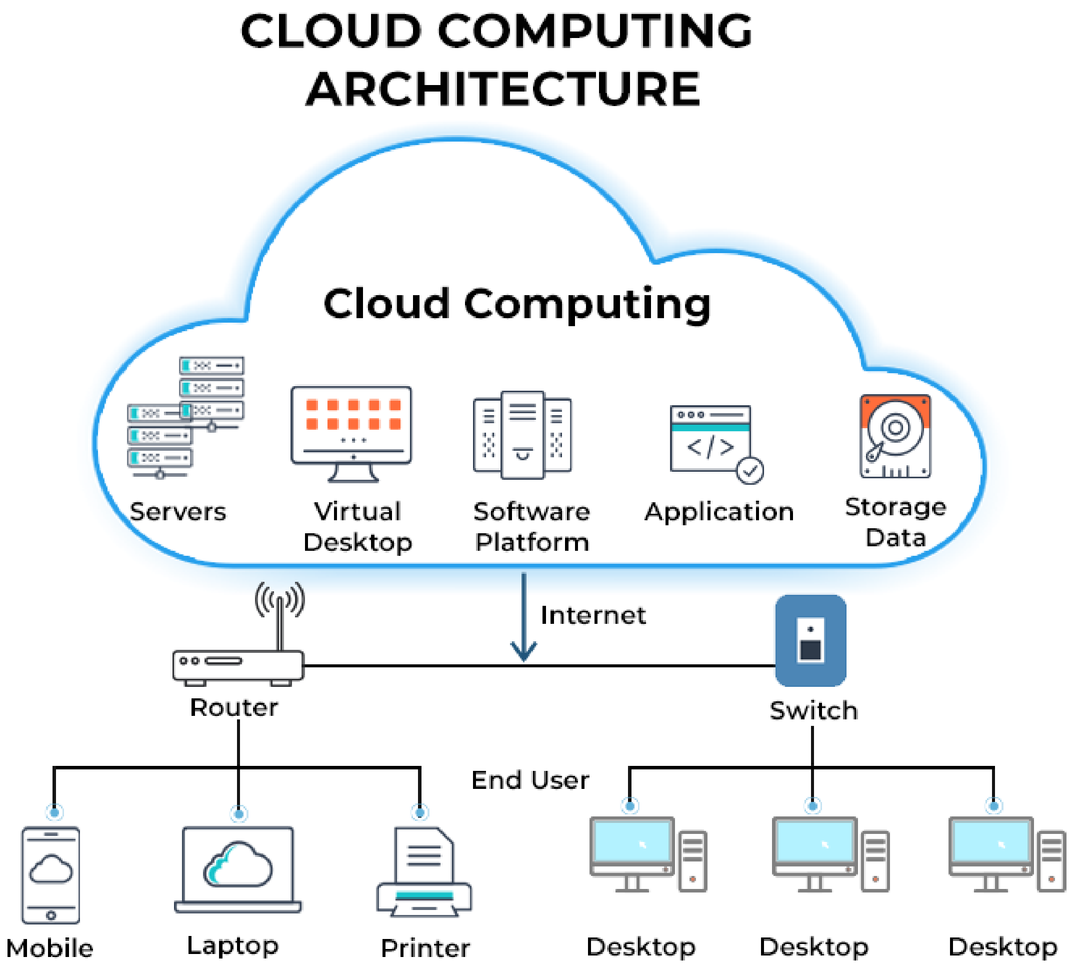
Course Schedules

Details Schedules

No	Week	Course			
		Course Titles	Theory	Practical Labs	Discussion
1	Week 1 06-Sep-2025	Intro to Cloud Computing	AWS services overview for ML (S3, EC2, IAM) ML lifecycle on AWS	AWS Free Tier setup (IAM, EC2, S3) Hands-on: launching EC2 Jupyter setup on EC2	
2	Week 2	Containerization & Serverless ML Pipelines	Intro to Docker + AWS ECR (Elastic Container Registry) Serverless Concepts	Build a simple ML inference container using Lambda, API gateway	
3	Week 3	Big Data Processing with PySpark & EMR	What is Big Data? PySpark concepts and EMR overview	Local Spark Setup and data preprocessing using pyspark	
4	Week 4	Model Development with SageMaker	Amazon SageMaker intro Built-in algorithms & Estimators	Tabular model training with built-in XGBoost Model tuning and logging	

WEEK 1

Why Cloud Computing ?



Resource Limitation

1. If we try to train a large deep learning model on a personal laptop, what computational bottlenecks will we face?

Cost

2. Suppose training is not frequent; maybe once a month. From an economic standpoint, is it efficient to invest in expensive hardware, or is a rental model better?

Scalability

3. Now imagine an application experiences a sudden 100x increase in users. How can we design infrastructure to elastically adjust to demand instead of collapsing?

Compliance

4. In certain industries, data must remain in specific countries due to regulatory requirements. How can infrastructure adapt to such constraints?

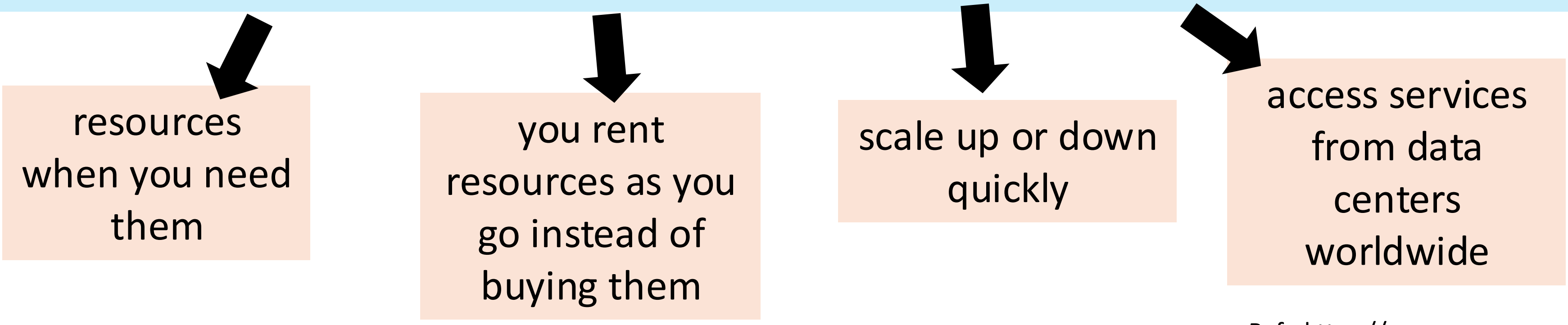
These four problems ; compute bottlenecks, cost, scaling, compliance — are exactly what cloud computing was invented to solve

1. What is Cloud Computing ?

“”Cloud computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centres and servers, you can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS).”” ***FROM AWS***

IN SHORT

- *“Cloud computing = on-demand delivery of IT resources via the internet with pay-as-you-go pricing.”*
- Keywords => **on-demand**, **pay-as-you-go**, **scalable**, **global**.



resources when you need them

you rent resources as you go instead of buying them

scale up or down quickly

access services from data centers worldwide

Ref : https://aws.amazon.com/what-is-cloud-computing/

Types of Cloud Computing

The three main types of cloud computing -

On Premises

You own and manage everything: servers, storage, networking, OS, applications

IaaS (Infrastructure as a Service)

You rent raw infrastructure (compute, storage, networking). You install OS, frameworks, apps.

PaaS (Platform as a service)

You rent a managed environment — provider handles OS, runtime, scaling. You just deploy code or models.

SaaS (Software as a Service)

You use the software directly via the internet. No need to manage infra, runtime, or code.

Every day life examples -

Cooking at home from scratch

- You buy your own stove, fridge, ingredients.
- You cook, serve, and clean everything.
- *Full control, full responsibility.*

Renting an empty kitchen

- The landlord gives you the kitchen space (electricity, water)
- You bring your own oven recipes and cook your own meal.
- *You control the cooking, but someone else owns the building*

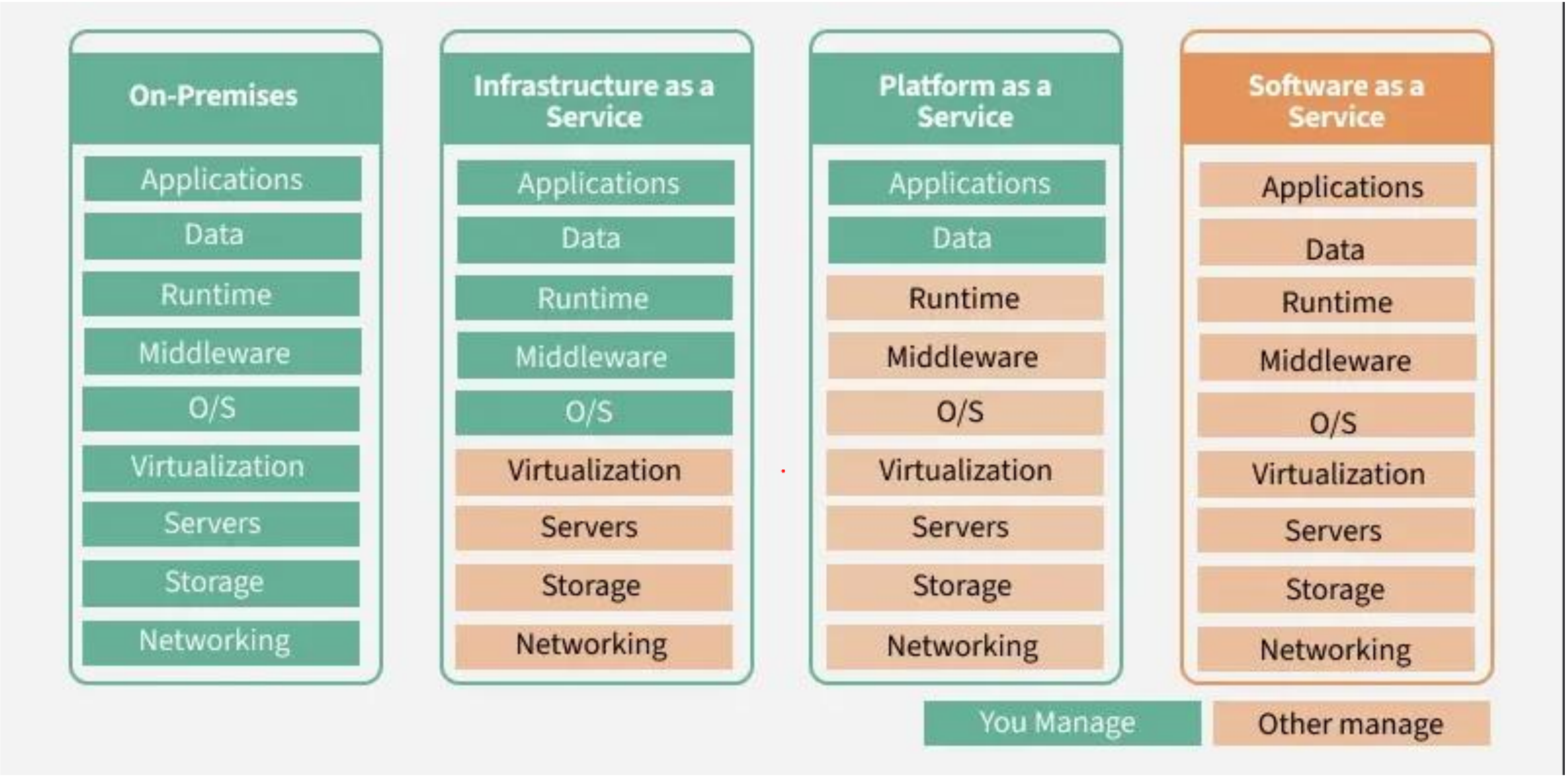
Using a ready-to-use kitchen (with tools & oven)

- The kitchen comes with oven, mixer, utensils.
- You just bring ingredients and bake your cake.
- *You focus on the recipe, not setting up the kitchen.*

Ordering food via GrabFood / Foodpanda

- You just order, food arrives ready to eat.
- No cooking, no cleaning.
- *Everything is handled for you. you only consume the service.*

Each type of cloud computing provides different levels of control, flexibility, and management so that you can select the right set of services for your needs.



On-Premises	Infrastructure as a Service	Platform as a Service	Software as a Service
🟩 Applications	🟩 Applications	🟩 Applications	🟧 Applications
🟩 Data	🟩 Data	🟩 Data	🟧 Data
🟩 Runtime	🟩 Runtime	🟧 Runtime	🟧 Runtime
🟩 Middleware	🟩 Middleware	🟧 Middleware	🟧 Middleware
🟩 O/S	🟩 O/S	🟧 O/S	🟧 O/S
🟩 Virtualization	🟧 Virtualization	🟧 Virtualization	🟧 Virtualization
🟩 Servers	🟧 Servers	🟧 Servers	🟧 Servers
🟩 Storage	🟧 Storage	🟧 Storage	🟧 Storage
🟩 Networking	🟧 Networking	🟧 Networking	🟧 Networking

🟩 You Manage 🟧 Other manage

Ref : https://www.geeksforgeeks.org/software-engineering/difference-between-iaas-paas-and-saas/

Now that you get the food analogy… what do you think the equivalent examples would be for **Machine Learning**?

Machine Learning Scenario

On Premises

IaaS (Infrastructure as a Service)

PaaS (Platform as a service)

SaaS (Software as a Service)

- *You buy your own GPU servers, install TensorFlow, set up networking, manage everything yourself.*
- *You rent AWS EC2 GPU instances and use S3 for storage. You install ML libraries yourself and run training.*
- *You use AWS SageMaker — AWS gives you Jupyter notebooks, pre-installed ML frameworks, and managed training jobs. You focus only on the model and data.*
- *You call AWS Rekognition API for image classification, or AWS Translate for language translation. You don't train anything; you just consume the ML service.*

No	Types of Cloud Computing	AWS Services
1	**On Premises**	-
2	**IaaS**	EC2 (Compute), S3 (Storage), IAM (Security)
3	**PaaS**	SageMaker (end to end ML platform), AWS Glue (ETL Preprocesing)
4	**SaaS**	Rekoginition (image/video analysis), Comprehend (NLP), Translate, Polly (tts)

DISCUSSION 1: Cloud Strategy

You are building a real-time fraud detection system for a mobile payment app serving **500,000 monthly users** across Southeast Asia. The app requires **low-latency predictions (<300ms)** and needs to comply with **local data residency laws** (e.g., in Singapore and Indonesia).

Would you choose a **cloud-based ML pipeline** or an **on-premise setup**? Justify your decision with respect to **latency, security, DevOps complexity, cost, and data compliance**.

My Response (Cloud is Preferable)

Latency

- **<300ms prediction latency** is realistic using AWS services like **Lambda**, **SageMaker endpoints**, or even **Edge deployment (SageMaker Neo)**.
- AWS supports regional endpoint deployment (e.g., Singapore, Jakarta) to minimize network roundtrip delays.
- On-prem latency could be slightly lower, but managing low-latency infrastructure in multiple countries is operationally painful.

Security & Data Compliance

- Local data residency laws can be met using **AWS region-specific S3 buckets**, **VPC controls**, and **KMS** for encryption.
- IAM roles and logging help maintain strong access control and auditing.
- On-prem would offer tighter control but would **increase complexity and staffing needs**, especially across multiple jurisdictions.

DevOps & Maintenance

- In the cloud, I can automate model deployment and versioning with **CI/CD pipelines (e.g., CodePipeline + Lambda + S3)**.
- Scaling to 500K users becomes trivial with **autoscaling EC2 or Lambda**, which would be much harder on-prem without large DevOps investment.

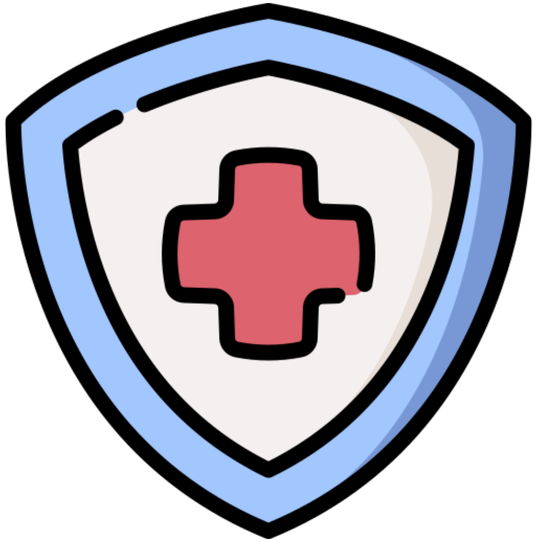
Cost

- **Cloud is more cost-efficient** for variable workloads. I can use **spot instances for training** and **serverless inference** to reduce idle cost.
- On-prem would require purchasing hardware upfront for peak capacity — expensive and inefficient for a mid-scale use case.

A **cloud-first strategy with regional AWS configurations** gives the best balance of **performance, security, and compliance** for this specific use case. On-prem might be justified for *highly classified or ultra-low latency* systems, but not here.

DISCUSSION 2: Cloud Strategy

You’re working for a **national healthcare agency** building a long-term ML system for **medical image classification (e.g., tumor detection)**. The system processes **sensitive MRI/CT scan data** from multiple hospitals, must operate in a **secure and air-gapped environment**, and is expected to run continuously for **at least 5 year**.



Due to regulatory policies, **patient data cannot leave the physical infrastructure**. Would you choose a **cloud-based setup** or an **on-premise architecture**? Justify your choice.

My Response (On Prem is Preferable)

Data Privacy & Compliance

- Health data is **extremely sensitive**. The scenario explicitly states that **data cannot leave the infrastructure**, which makes most public cloud options (even regionally hosted ones) **non-compliant**.
- AWS and other cloud providers offer HIPAA-compliant services, but **regulatory approval** for handling medical images often requires **physical air-gapped systems** in national infrastructure.

Security Control

- On-prem allows **total control over network traffic**, **access permissions**, and **hardware-based encryption**.
- There's no reliance on third-party APIs or internet access; important for **zero-trust or classified healthcare systems**.

Cost Predictability

- For a **5-year deployment**, on-prem hardware amortized over time could be **cheaper** than cloud usage (especially for **GPU-heavy inference or training workloads** like medical imaging).
- Cloud's pay-as-you-go becomes **less attractive** when usage is **steady and high-volume** for years.

Integration with Hospital Infrastructure

- Hospitals may already use **local PACS servers**, proprietary image formats, and specific network configurations that integrate better with an on-prem setup.
- On-prem simplifies integration and avoids costly data transfer or translation layers required in cloud.

Risk Tolerance

- In critical systems (like national health), **dependency on external vendors** for model hosting, service uptime, or API changes may be unacceptable.
- On-prem gives full-stack control and avoids vendor lock-in.

For sensitive, regulated, long-term medical ML systems, on-premise deployment offers compliance, security, control, and cost stability that cloud infrastructure cannot match.

DISCUSSION 3: Cloud Strategy

Scenario-based Question:

You’re building an ML pipeline to **detect phishing websites** by taking screenshots of URLs and comparing them to known company logos and UI layouts (e.g., fake login pages).
Each day, the system ingests **thousands of URLs**, renders them (headless browser), takes a screenshot, and passes it through a **CV model (e.g., ResNet or ViT)** to assess visual similarity.

You’re deciding whether to run this **on-prem** or in the **cloud**.

My Response (On Prem is Preferable)

Scalability

- Screenshot rendering + image inference is compute-intensive.
- Cloud (e.g., AWS Lambda + ECS + SageMaker) scales up quickly for bursty, parallel inference.

URL Access and Threat Intel

- Since you're scraping **live URLs**, running in the cloud may avoid local network exposure.
- You can route traffic through **VPC NAT gateways**, restrict domains, and avoid hitting the internal corp network.

Pay-per-user Cost Model

If you're only running inference during peak attack windows or using event-driven triggers (e.g., AWS EventBridge), cloud billing stays low.

My Response (On Prem is Preferable)

WEEK 1 : AWS Service Overview for ML (IAM, S3, EC2)

AWS IAM

→ AWS

Learning Objectives

- Understand **IAM core concepts** (Users, Roles, Policies, Permissions)
- Learn **principle of least privilege** in an ML context
- Explore **how IAM affects ML lifecycle stages** (e.g., access to S3, EC2, SageMaker)
- Hands-on: Create users, roles, groups and policies
- Discussion:
  - Design a **secure IAM strategy for a production ML pipeline**
  - Analyze real-world IAM misconfigurations and **simulate attacks**

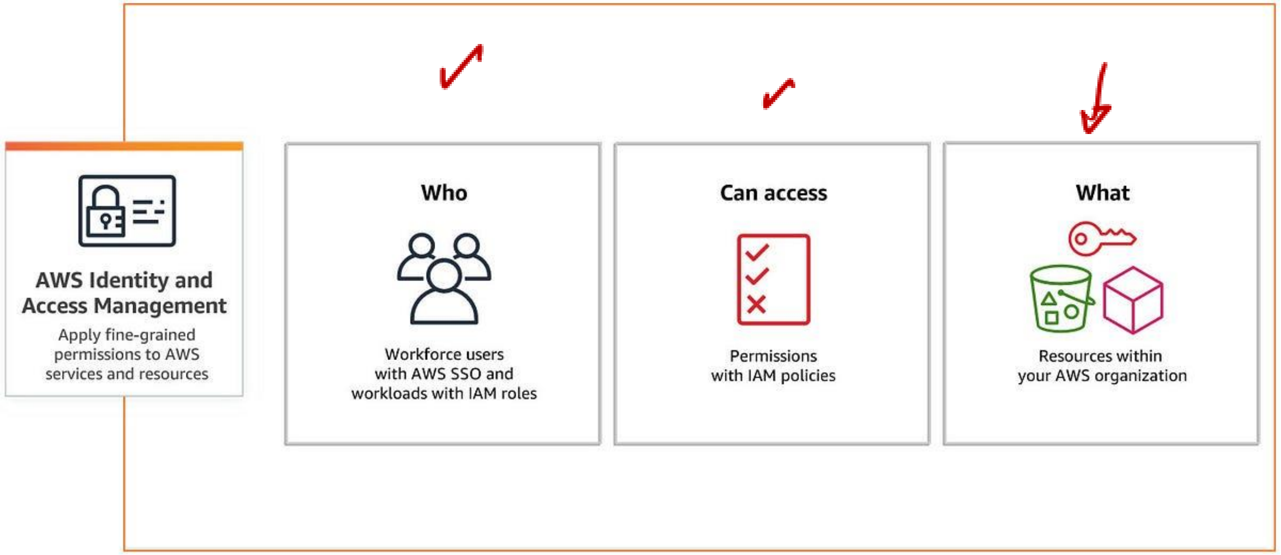
1. AWS IAM

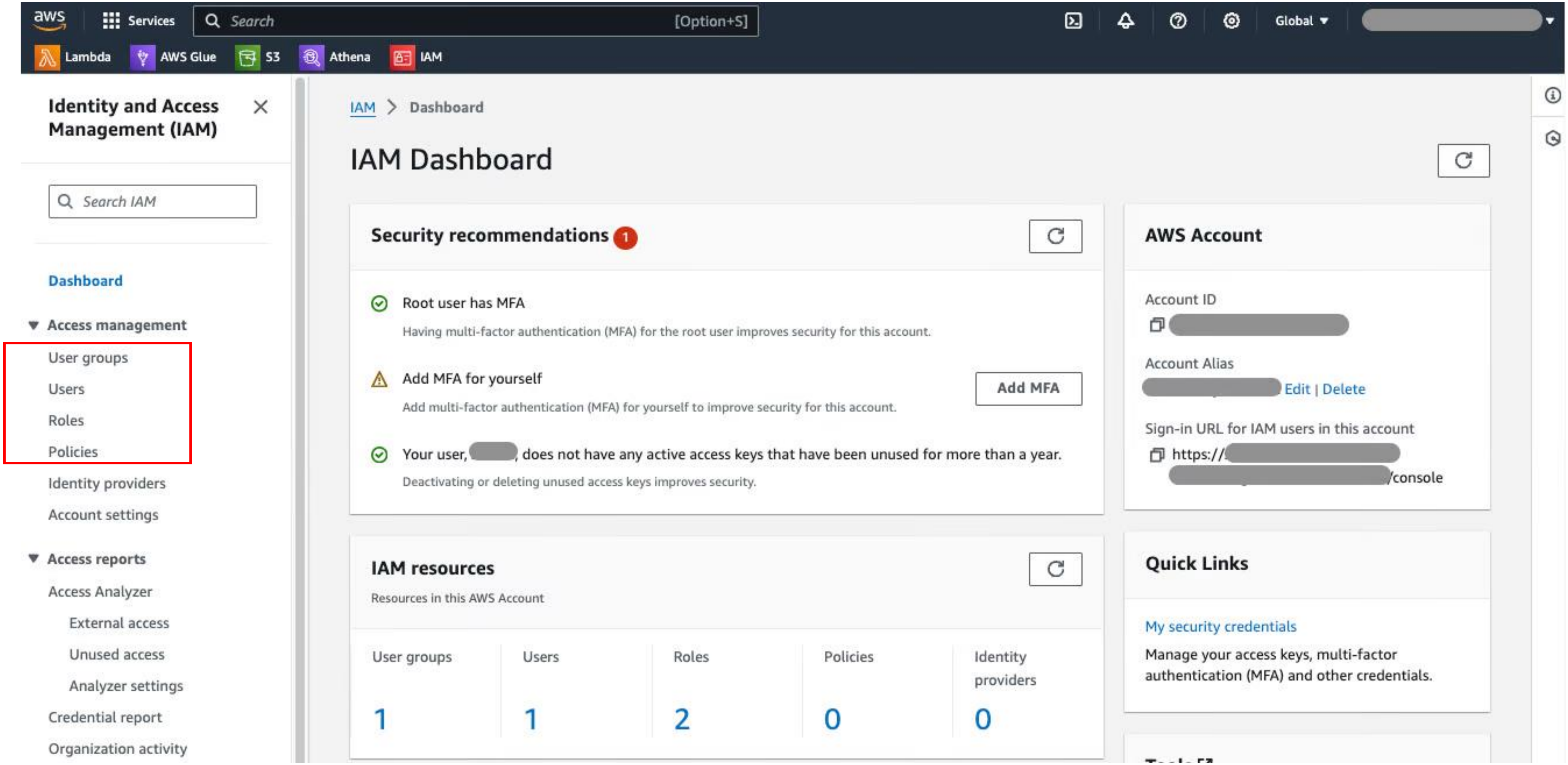
IAM stands for Identity and Access Management

It was released in 2011 and is a tool for managing ***who can access your AWS resources and what they can do with them***.

Think in terms of:

- **Identity**: *Who* is making the request? (User, Role, Service)
- **Access**: *What* are they trying to do? (s3:GetObject, ec2:StartInstances, etc.)
- **Resource**: *Which* object? (arn:aws:s3:::your-bucket/data.csv)


AWS Identity and Access Management
Apply fine-grained permissions to AWS services and resources
Who
Workforce users with AWS SSO and workloads with IAM roles
Can access
Permissions with IAM policies
What
Resources within your AWS organization



IAM Dashboard

Security recommendations 1

Root user has MFA

Having multi-factor authentication (MFA) for the root user improves security for this account.

Add MFA for yourself

Add multi-factor authentication (MFA) for yourself to improve security for this account.

Your user, [redacted], does not have any active access keys that have been unused for more than a year.

Deactivating or deleting unused access keys improves security.

IAM resources

Resources in this AWS Account

User groups	Users	Roles	Policies	Identity providers
1	1	2	0	0

AWS Account

Account ID

[redacted]

Account Alias

[redacted] Edit | Delete

Sign-in URL for IAM users in this account

https://[redacted]/console

Quick Links

My security credentials

Manage your access keys, multi-factor authentication (MFA) and other credentials.

AWS IAM

These are the four essential components of IAM:

1

User

Users are individuals or applications that need access to your AWS resources. Each user receives unique credentials, such as passwords and access keys.

SSO

Group

Groups are collections of users. Instead of assigning permissions to each user individually, you can put users in groups and assign permissions to the group. This makes it easier to manage permissions for multiple users at once.

Ec2, lambda

Role

Roles are meant to be assumed by anyone who needs them instead of being associated with a single person. For example, an EC2 instance can “assume a role” to access S3 buckets without needing permanent credentials. Roles use temporary security credentials that automatically expire, which is great in terms of security.

S3 Full

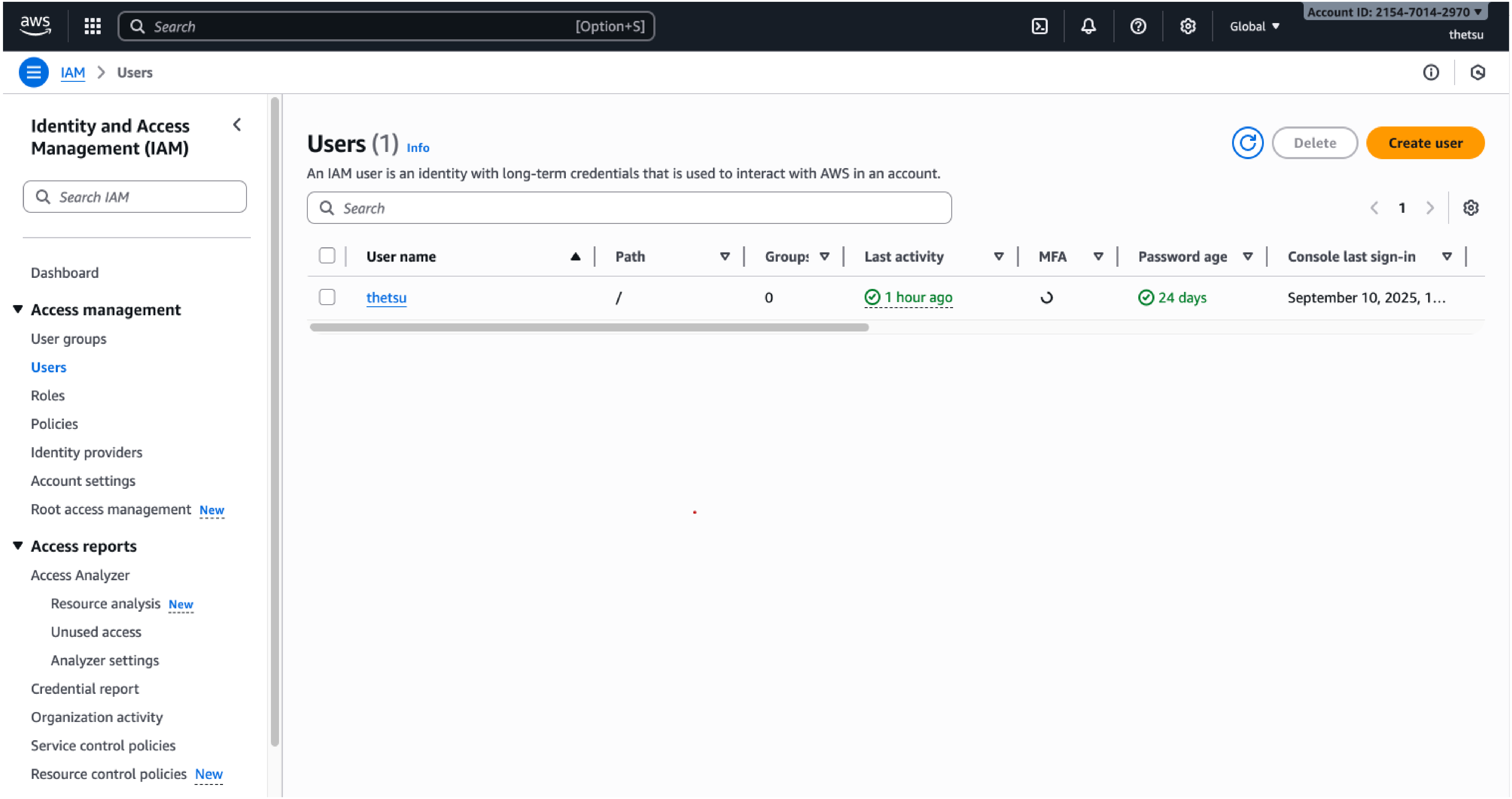
↓ Deny

Policy

Policies are JSON documents that define permissions. They specify what actions are allowed or denied on which resources. Policies can be attached to users, groups, or roles.

User

Users are individuals or applications that need access to your AWS resources.

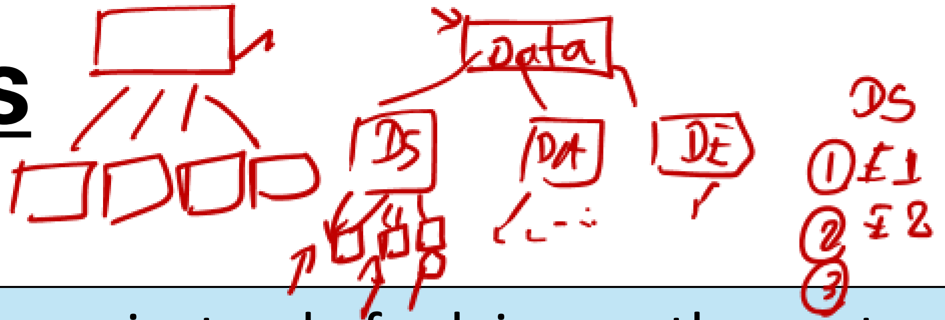


Users (1) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

□	User name ▲	Path	Groups	Last activity	MFA	Password age	Console last sign-in
□	thetsu	/	0	✓ 1 hour ago		✓ 24 days	September 10, 2025, 1...

IAM User Best Practices



1. Don’t use root as a user — Create the IAM user instead of relying on the root user account
2. Create individual IAM users — No shared accounts ;each person gets their own IAM user.
3. Assign Policies to groups, not directly to all IAM users
4. Enable MFA (Multi-Factor Authentication) for all IAM users.
5. Rotate credentials regularly.
6. Give IAM users console or programmatic access only if required
7. Remove inactive users, review permission to make sure they align with current job needs.

IAM – Lab 1

Create the IAM User

Group

Similar to user (an identity with permission)

Does not have credentials (password or keys)

Assumable, temporarily, by anyone who needs it.

https://youtu.be/hAk-7ImN6iM?si=PauhosND60Lu6wxT

IAM ~~User~~ Group Best Practices

1. Assign Policies to groups, not directly to all IAM users
2. Follow least-privilege principal at group level.
3. Rotate credentials regularly.
4. Remove inactive groups, review permission to make sure they align with current job needs.

IAM – Lab 2

Create the IAM Group

✓

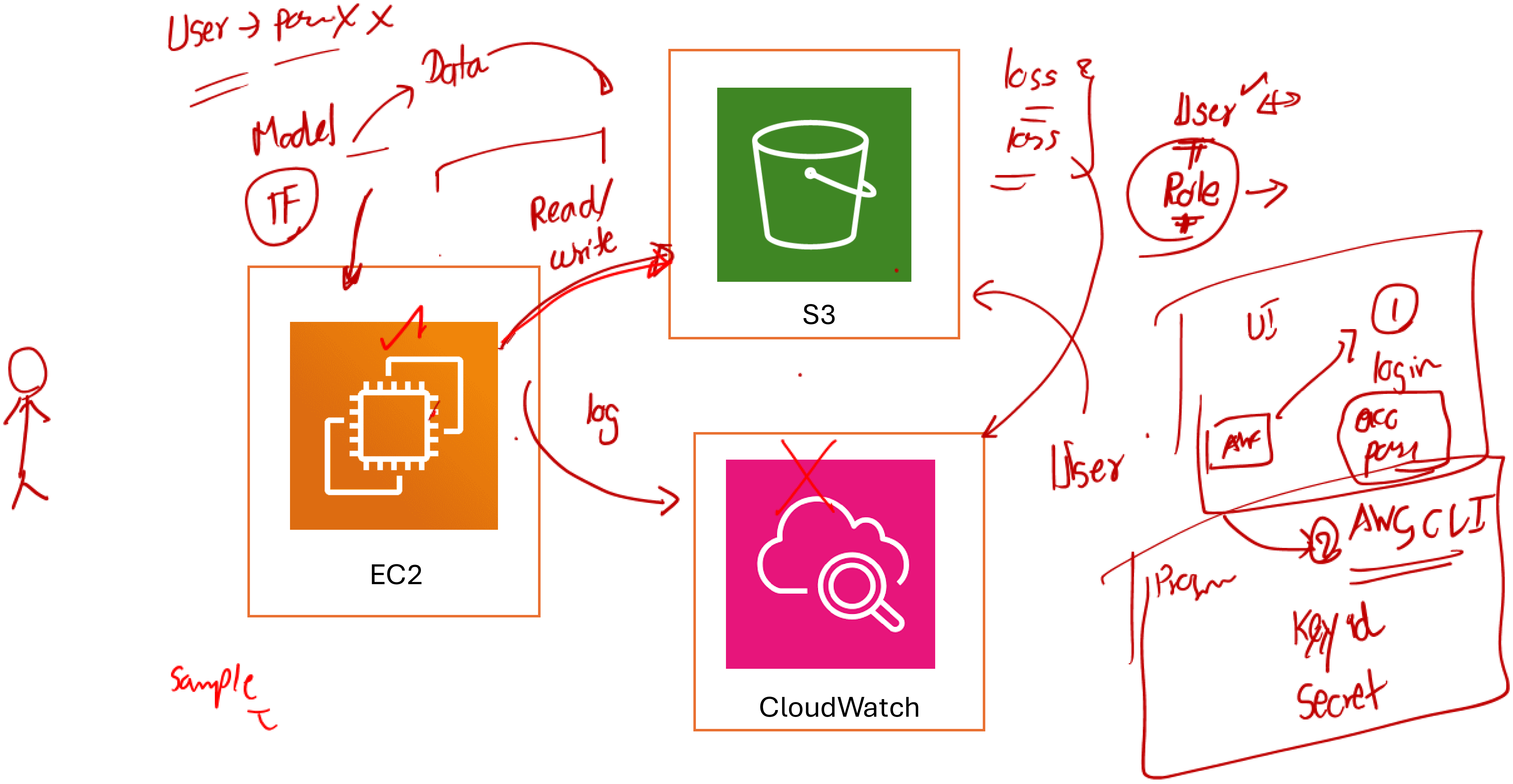
Role

Similar to user (an identity with permission)

Does not have credentials (password or keys)

Assumable, temporarily, by anyone who needs it.

https://youtu.be/hAk-7ImN6iM?si=PauhosND60Lu6wxT



EC2

S3

CloudWatch

Option 1

Sklearn

Python → boto3 → AWS Services

Read S3 EC2

1. Create an IAM user for the application with appropriate permissions
2. Hard-coded user credentials in the application or on the EC2 instance's file system and retrieve them at runtime.

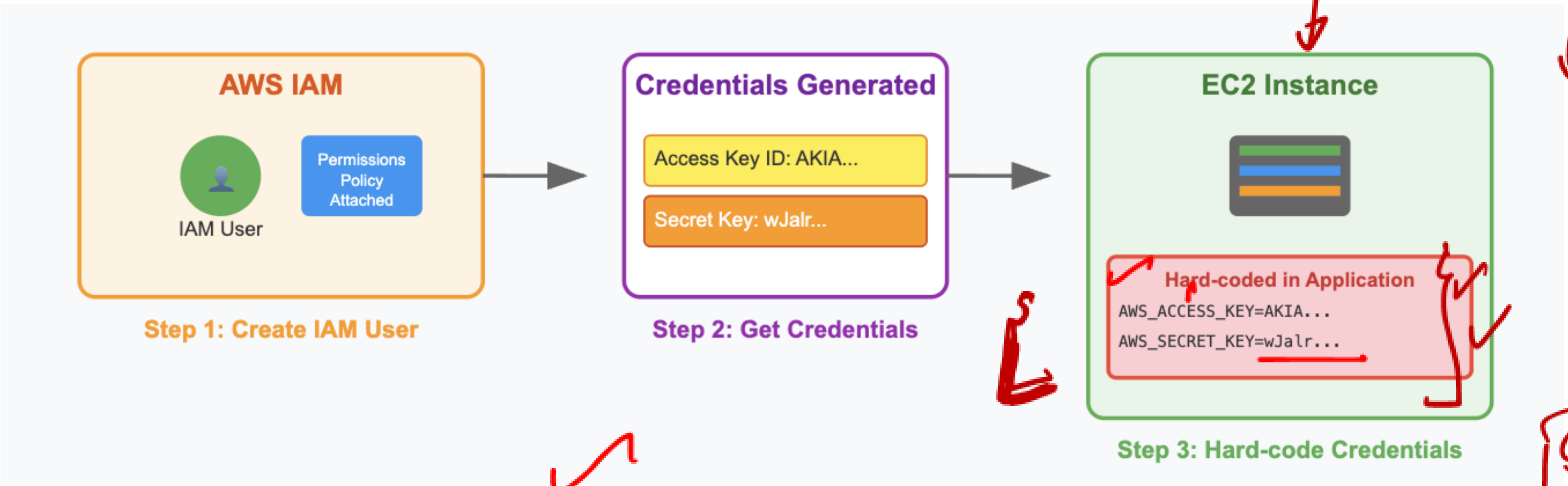
.py

model-train

prep

boto3

S3



AWS IAM

IAM User

Permissions Policy Attached

Step 1: Create IAM User

Credentials Generated

Access Key ID: AKIA...

Secret Key: wJalr...

Step 2: Get Credentials

EC2 Instance

Hard-coded in Application

```
AWS_ACCESS_KEY=AKIA...
AWS_SECRET_KEY=wJalr...
```

Step 3: Hard-code Credentials

boto3

AWS ✓

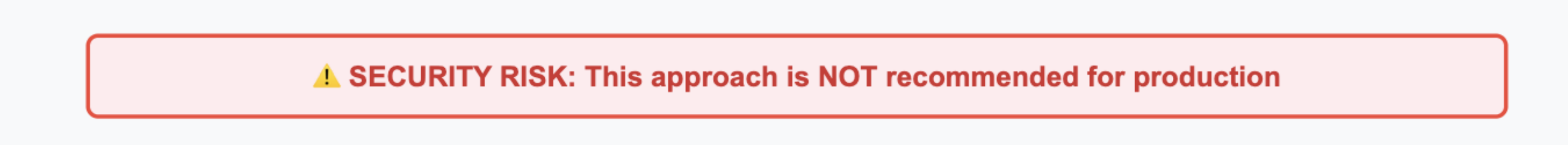
EC2 → AWS CLI → S3

Local → AWS CLI → S3

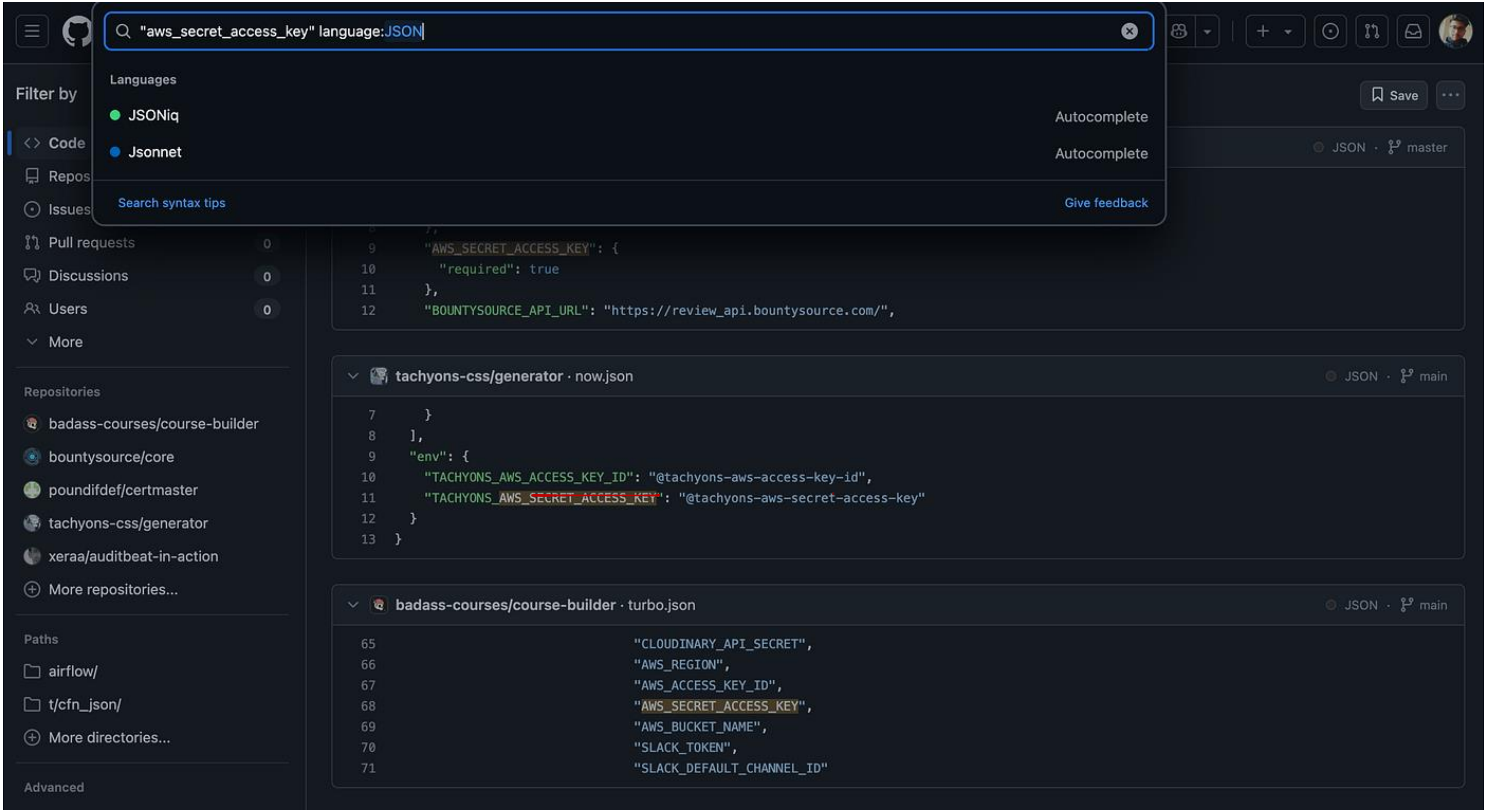
Really ???

Imagine you create an IAM user called ml-app-user with S3 full access. Then you copy its Access Key + Secret Key into your code, or you drop them into /home/ec2-user/credentials.txt. Now your app can run… but what’s the risk?

1. If code is pushed to GitHub → **credentials leaked**.
2. If EC2 instance is compromised → attacker **steals the keys**.
3. Hard-coded keys are **long-lived** → attacker has permanent access.
4. Difficult to **rotate/revoke** credentials without breaking the app.



⚠️ **SECURITY RISK: This approach is NOT recommended for production**



```
"aws_secret_access_key" language:JSON
```

```
    },
    "AWS_SECRET_ACCESS_KEY": {
      "required": true
    },
    "BOUNTYSOURCE_API_URL": "https://review_api.bountysource.com/",
```

tachyons-css/generator · now.json

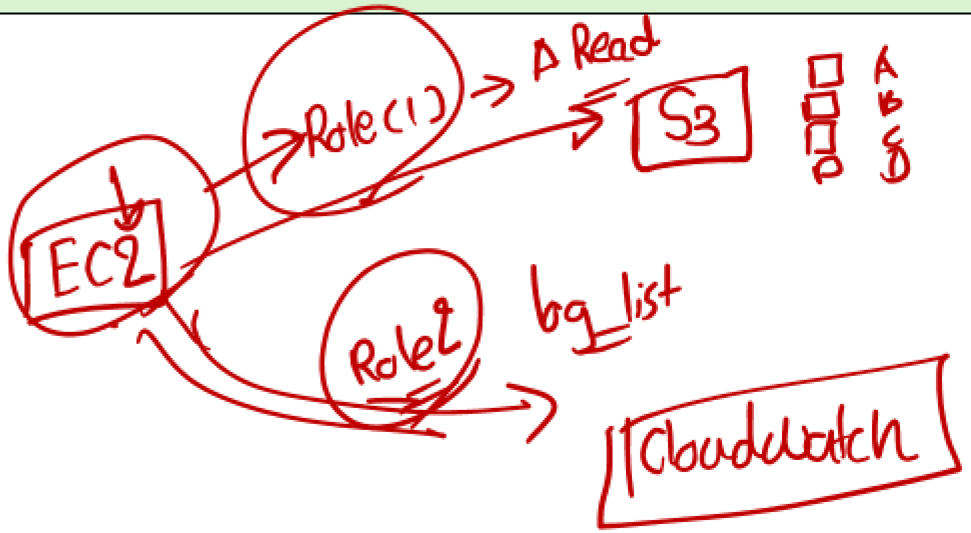
```
      }
    ],
    "env": {
      "TACHYONS_AWS_ACCESS_KEY_ID": "@tachyons-aws-access-key-id",
      "TACHYONS_AWS_SECRET_ACCESS_KEY": "@tachyons-aws-secret-access-key"
    }
  }
```

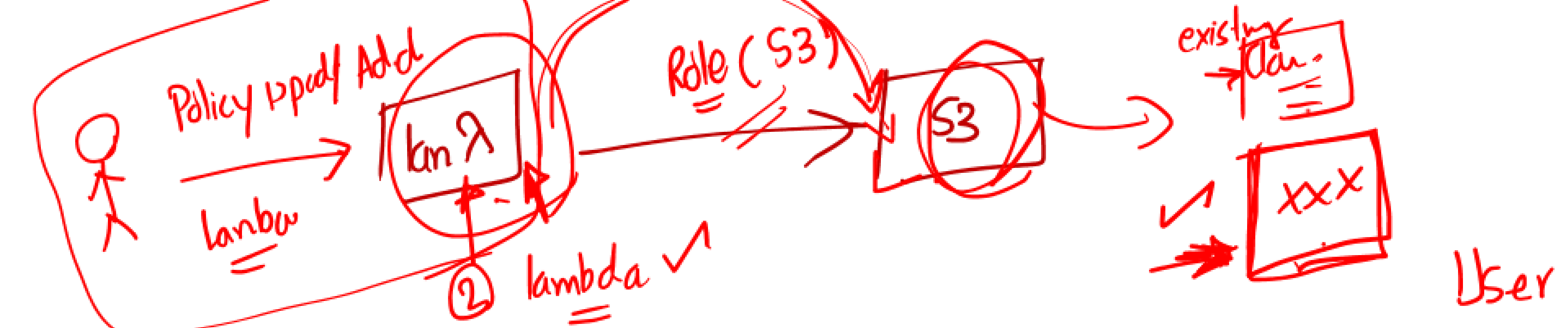
badass-courses/course-builder · turbo.json

```
                                "CLOUDINARY_API_SECRET",
                                "AWS_REGION",
                                "AWS_ACCESS_KEY_ID",
                                "AWS_SECRET_ACCESS_KEY",
                                "AWS_BUCKET_NAME",
                                "SLACK_TOKEN",
                                "SLACK_DEFAULT_CHANNEL_ID"
```

Option 2

1. Create an IAM role for the application with appropriate permissions
2. When creating the EC2 instance, assign it this role
3. No credentials required





IAM – Lab 3

EC2 → Role

land.

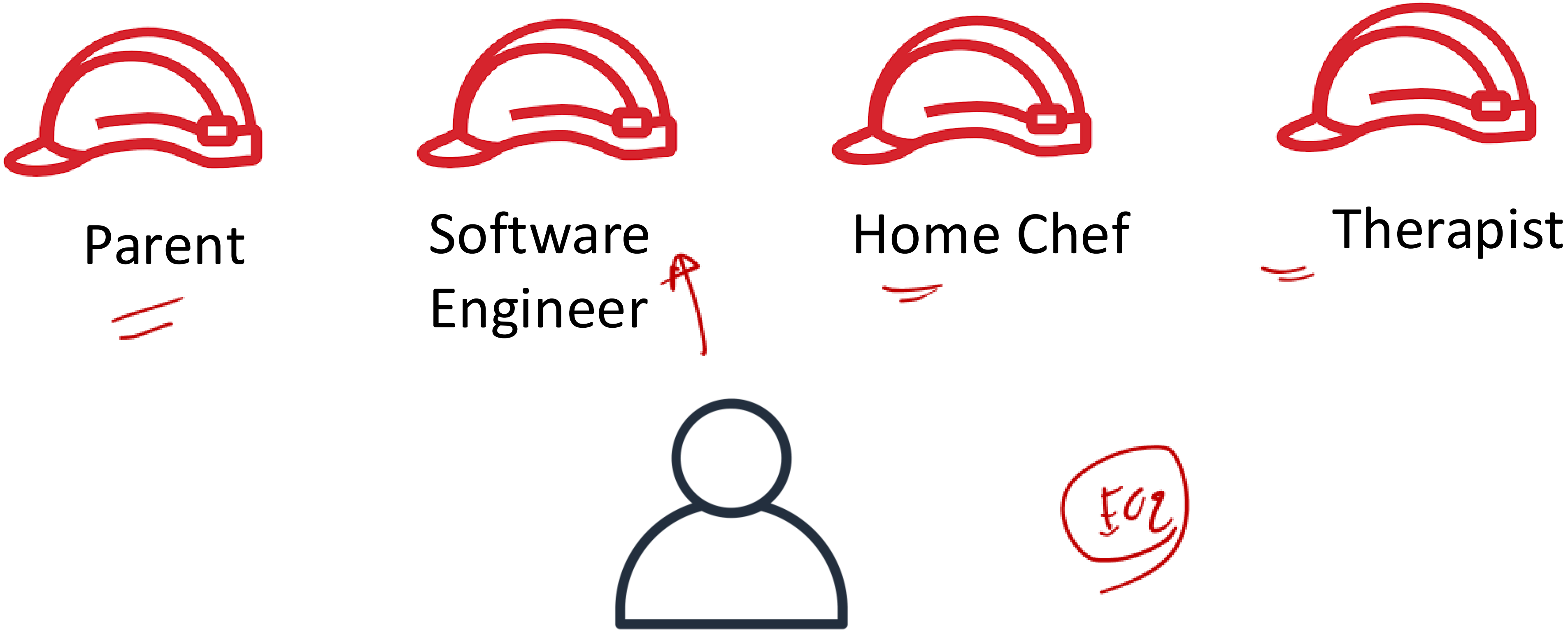
Create the IAM Role

- Lambda Functions Setup to access S3

.py →

Real Life Analogy

Roles: hats you put on temporarily depending on context



Parent

Software Engineer

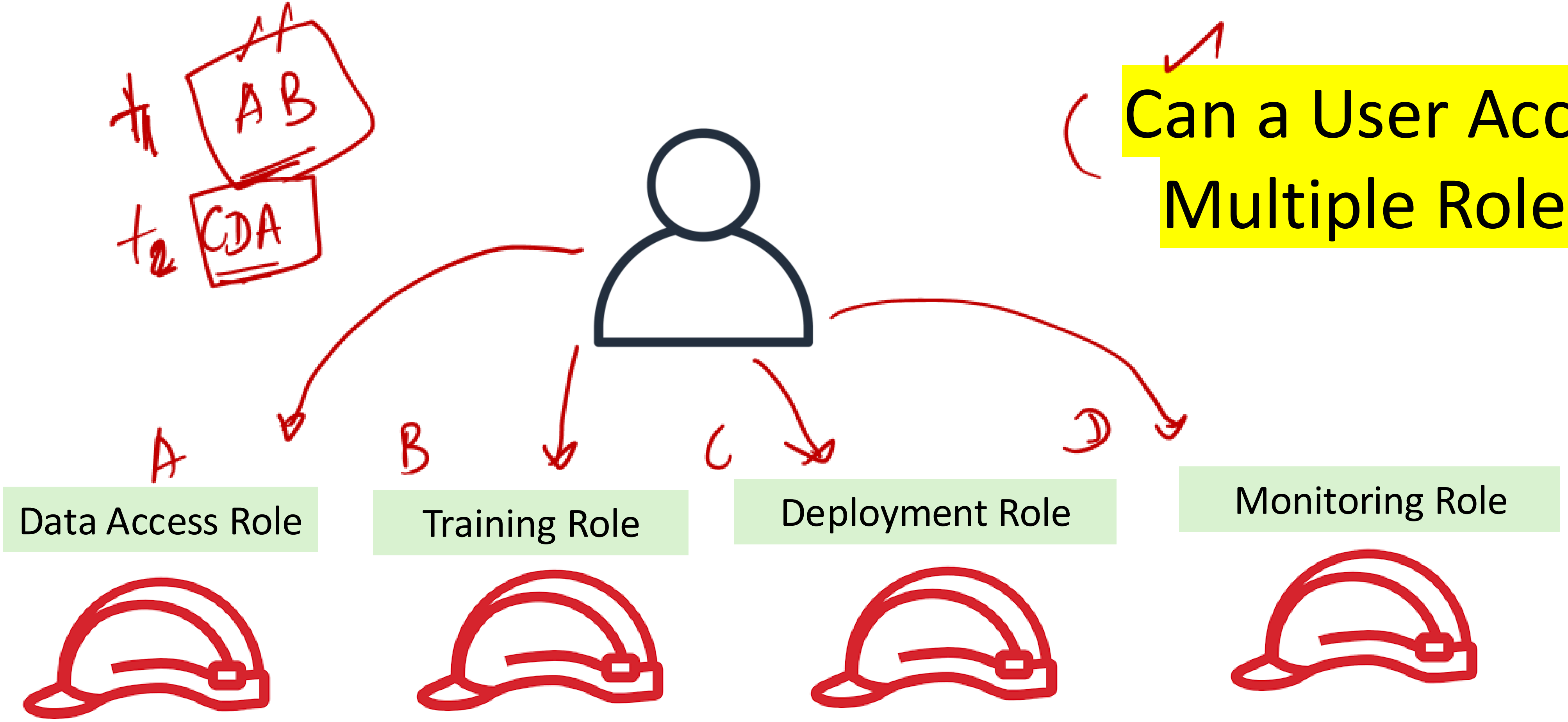
Home Chef

Therapist

Foz

identity (you, always the same person)

Can a User Access Multiple Roles?



Data Access Role

read/write datasets in **S3**

Training Role

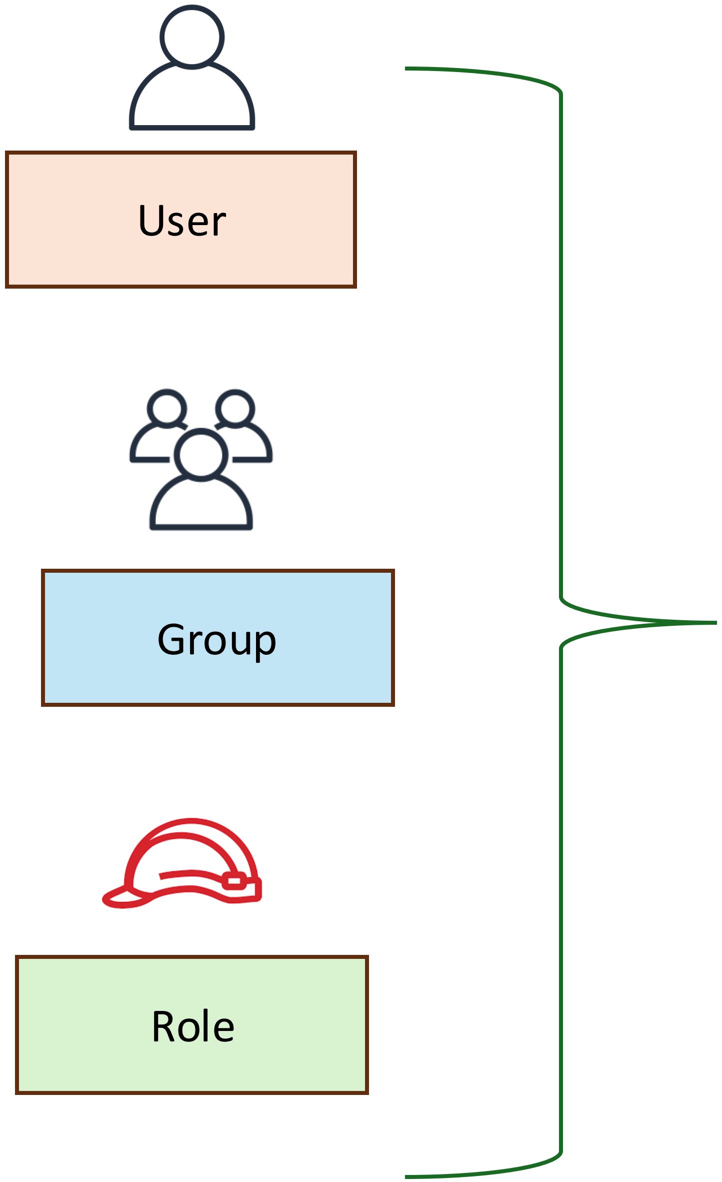
Used by **SageMaker** training jobs to spin up compute and access training data.

Deployment Role

Used by **SageMaker Endpoints / EC2 / ECS** to serve trained ML models.

Monitoring Role

Allows CloudWatch/SageMaker Model Monitor to log metrics and detect drift



User

Group

Role

Identities (the ‘Who’)

There is nothing we can do

Policies

Who can do what to which resources and when

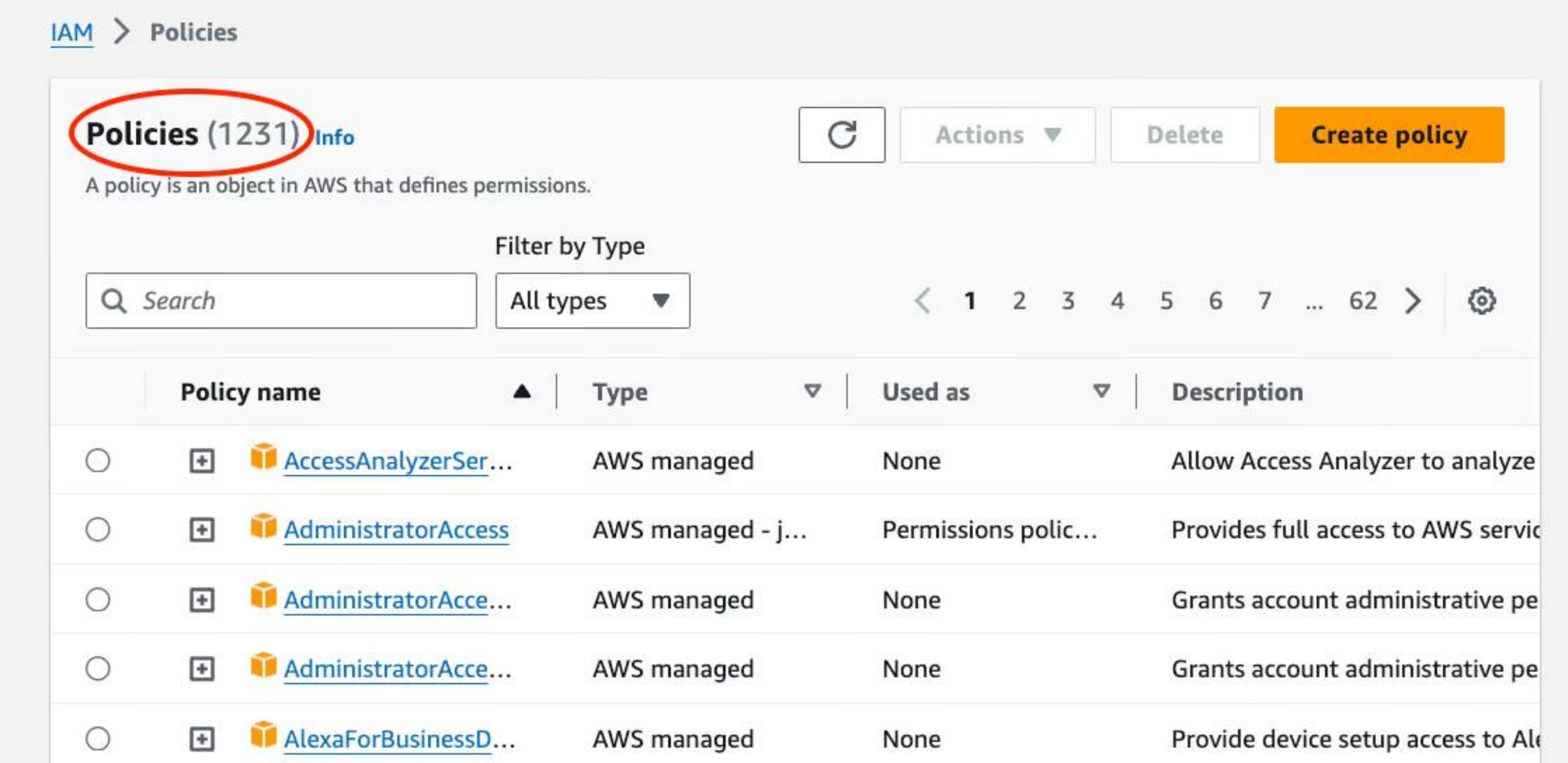
Policy 1

“Allow a Lambda function to access a Dynamo table”

Policy 2

“Allow a user to start and stop EC2 instances”

https://youtu.be/hAk-7ImN6iM?si=PauhosND60Lu6wxT



A policy is an object in AWS that defines permissions.

	Policy name ▲	Type ▽	Used as ▽	Description
○	⊞ AccessAnalyzerSer…	AWS managed	None	Allow Access Analyzer to analyze
○	⊞ AdministratorAccess	AWS managed - j…	Permissions polic…	Provides full access to AWS servic
○	⊞ AdministratorAcce…	AWS managed	None	Grants account administrative pe
○	⊞ AdministratorAcce…	AWS managed	None	Grants account administrative pe
○	⊞ AlexaForBusinessD…	AWS managed	None	Provide device setup access to Ale

IAM Policy

As an example, here’s the JSON document for the AWS-managed policy named “AmazonS3FullAccess”:

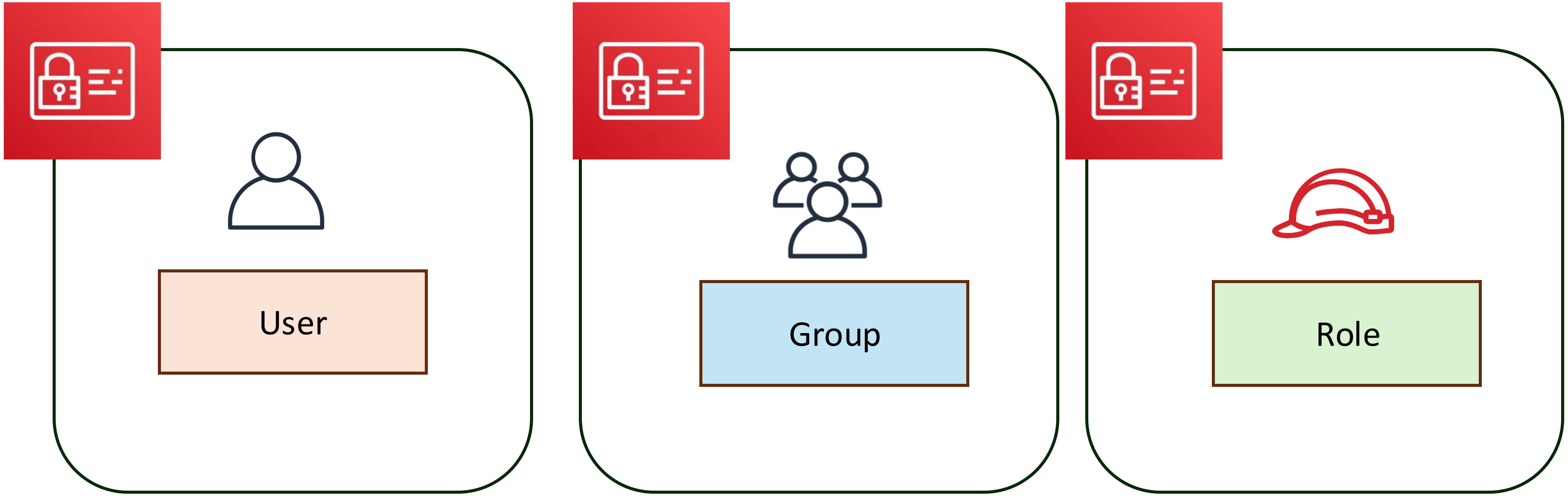
Note : this is not ideal in real-word scenario since the policy is too permissive.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "s3:*",
         "Resource": "*"
      }
   ]
}
```

- The Version element specifies the policy's language version.
- The Statement is where the “meat” of the policy is. A single policy can have multiple statements. Each statement includes a few essential fields:
  - **Effect**: This field specifies whether the statement allows or denies access. It can be either **"Allow" or "Deny."**
  - **Action**: This defines what actions are allowed or denied. Actions are usually the operations you can perform on a resource. AWS has specific actions for each service, so you must specify them correctly.
  - **Resource**: This specifies the AWS resources the actions apply to. Resources are identified using Amazon Resource Names (ARNs), which uniquely identify a resource.

https://www.datacamp.com/tutorial/aws-identity-and-access-management-iam-guide

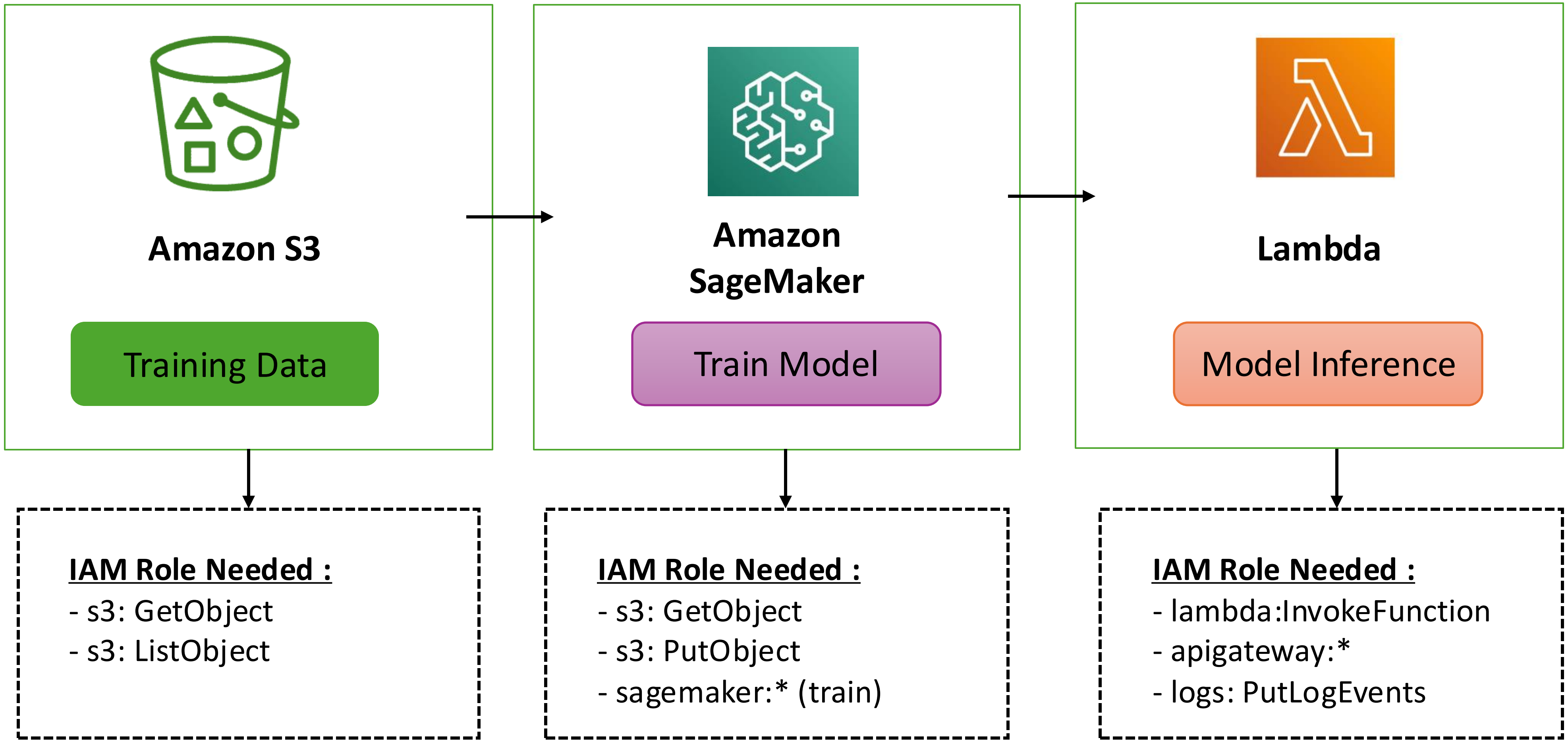
There is nothing we can do



Policies

Who can do what to which resources and when

Attach the policies to the identities



IAM Role Needed :
- s3: GetObject
- s3: ListObject

IAM Role Needed :
- s3: GetObject
- s3: PutObject
- sagemaker:* (train)

IAM Role Needed :
- lambda:InvokeFunction
- apigateway:*
- logs: PutLogEvents

Every arrow above requires **explicit IAM permissions**. If even one is missing or misconfigured, your ML pipeline breaks silently or fails mid-way.

→ S3FullAccess

→ Read

✓

IAM – Lab 4

✗

Create the IAM Policy

- Create the IAM policy to see the S3Bucket

IAM – Lab 5

Create the IAM Policy

- S3 Read-only Access from Ec2

IAM S3, EC2

100 /

25 25 25 25

100

cost -> 100

Week 1 – Part 2

S3 (Simple Storage Service)

Where data is kept, until someone with the right key asks for it.

20250919 -> hr=00

01

02

20

21

21

23

Telco ->

A ->

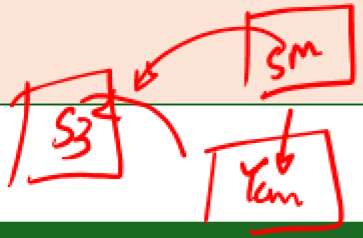
HTTP -> PUT

1. AWS S3 (Simple Storage Service)

Learning Objectives

- **Understand S3 basics**: Buckets, objects, and versioning
- **Data Partitioning** ✓
- **Learn how S3 integrates with the ML lifecycle**:
  - Data ingestion → Training → Inference → Logging
- **Explore S3 access control**: Bucket policies vs. IAM policies
- **Hands-on**: Upload datasets to S3 and assign correct bucket policies
- **Discussion**: Design a secure and scalable S3 storage structure for an ML project

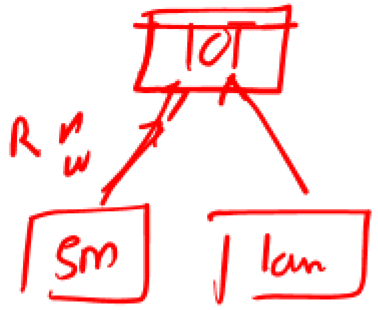
Think of S3 as the **warehouse** of your ML system.
It stores the raw material (data), the finished goods (models), and the operational logs (for QA).
Your job? Make sure it's **organized, secure, and scalable** — or everything else falls apart



	Stage	Usage Example
1	Data Collection	Store raw CSV, JSON, images, video, logs
2	Preprocessing	Upload clean feature sets, transformed files
3	Training	Models pull training data directly from S3 (via SageMaker or EC2)
4	Model Output	Store .pt, .pkl, or model.joblib after training
5	Serving Input	Upload new data for predictions
6	Predictions Log	Store model predictions or inference logs for audit/debug

Amazon S3 - Overview

- Amazon S3 allows people to store objects (files) in “buckets” (directories)
- Buckets must have a globally unique name
- Objects (files) have a Key. The key is the FULL path:
  - <my_bucket>/my_file.txt
  - <my_bucket>/my_folder1/another_folder/my_file.txt
- This will be interesting when we look at partitioning
- Max object size is 5TB
- Object Tags (key / value pair – up to 10) – Mini metadata (key/value) per object useful for security / lifecycle



AWS doesn't just charge for storage. You’re charged for data transfer and *how often* you read/write too. Efficient formats can literally save hundreds of dollars on big jobs

REF : Slides from AWS Machine Learning Course

10GB → SM, EC2   → ▤ → SM, EC2

Common File Formats Used in ML with S3

F1 | F2 | F3

	Format	Usage Example
1	csv	Tabular datasets, fast for small-medium jobs
2	Parquet	Columnar format; efficient for large data (via Spark/SageMaker)
3	JSON	Good for logs or structured text data
4	Images	JPEG, PNG — used in CV tasks (e.g., stored in s3://bucket/images/)
5	Models Files	.pt, .pkl. .joblib, .h5, etc – saved/loaded during training

- S3 wt charges per GB stored and per GB retrieved
- CSV may be "easy", but it’s expensive for big data ML
- Use **Parquet** for large tabular data — better compression & cheaper compute (especially with PySpark/SageMaker)
- For images, **always compress** (e.g., JPEG over PNG or BMP)
- Split large datasets into **smaller chunks** to parallelize reading in training jobs (SageMaker/EC2)

Credit →

Augh 08

July

August →

df=pd.

[. filter

Amazon S3 Data Partitioning

date/ hr=00

08

01

23

Partitioning = Splitting large datasets into smaller, logically organized chunks.

Benefits	Explanation
Faster training	Load only relevant data(e.g one region/date)
Lower cost ✓	Minimize I/O and compute time
Parallelism ✓	Train or process data in parallel across multiple workers
Easier filtering ✓	Only read partions than match condition (e.g date=2024-07-04)

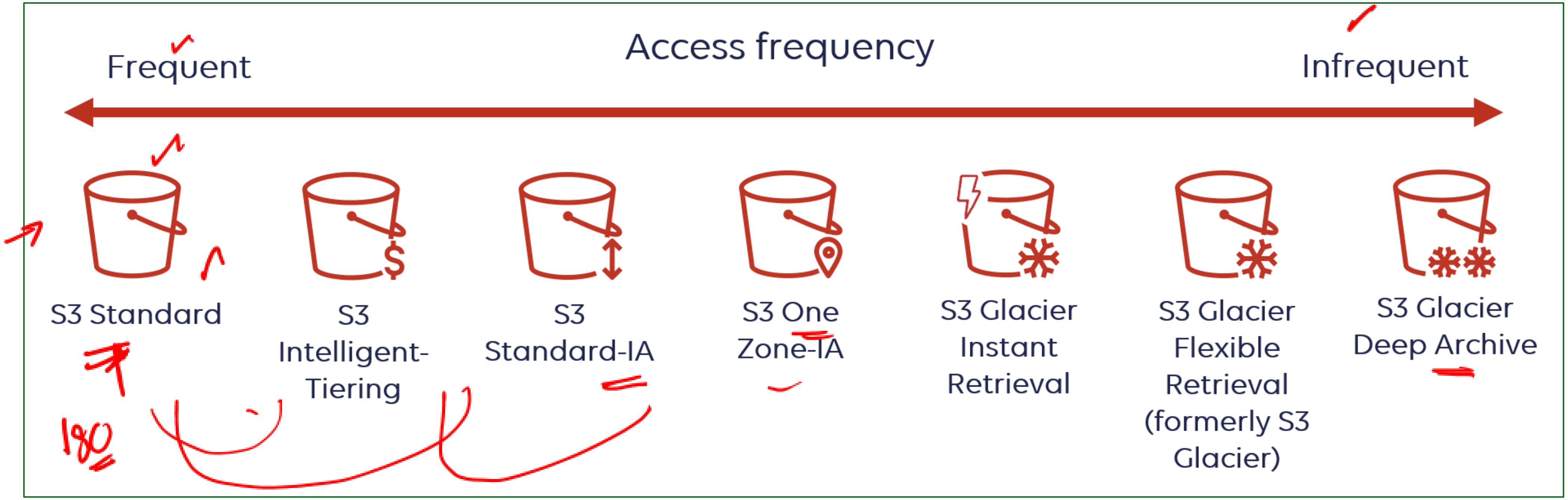
2015 ... 2015

₹

S3

Sep → predict → report

S3 Storage Classes



Access frequency

Frequent

Infrequent

S3 Standard

S3 Intelligent-Tiering

S3 Standard-IA

S3 One Zone-IA

S3 Glacier Instant Retrieval

S3 Glacier Flexible Retrieval (formerly S3 Glacier)

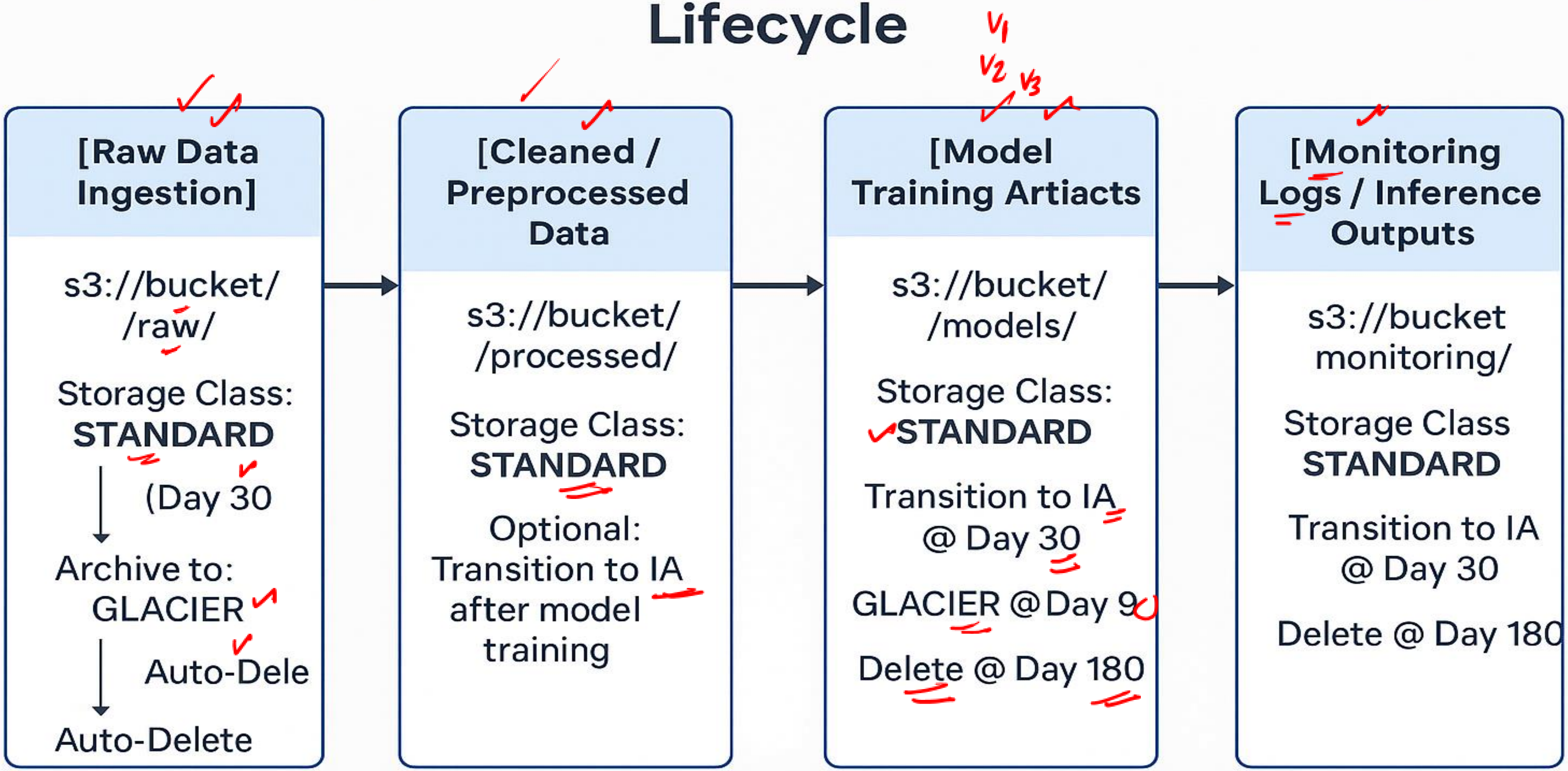
S3 Glacier Deep Archive

₹

180

Can move between classes manually or using S3 Lifecycle configurations

S3 Lifecycle Policies in the Machine Learning Lifecycle



V1
V2 V3

[Raw Data Ingestion]

s3://bucket/
/raw/

Storage Class:
STANDARD

(Day 30

Archive to:
GLACIER

Auto-Dele

Auto-Delete

[Cleaned / Preprocessed Data

s3://bucket/
/processed/

Storage Class:
STANDARD

Optional:
Transition to IA
after model
training

[Model Training Artiacts

s3://bucket/
/models/

Storage Class:
STANDARD

Transition to IA
@ Day 30

GLACIER @Day 90

Delete @ Day 180

[Monitoring Logs / Inference Outputs

s3://bucket
monitoring/

Storage Class
STANDARD

Transition to IA
@ Day 30

Delete @ Day 180

T/H → location

10K

2025

Discussion Question

"You are building a fraud detection model using 2 years of credit card transaction logs (≈5TB). The data includes

- *timestamp,*
- *location,*
- *merchant type,* ✓
- *transaction amount, and* ✓
- *customer ID.*

How would you design your S3 data partitioning strategy to reduce cost, improve training speed, and support model retraining over time?"

Sample Answer

S3 Data Partitioning Strategy for Fraud Detection

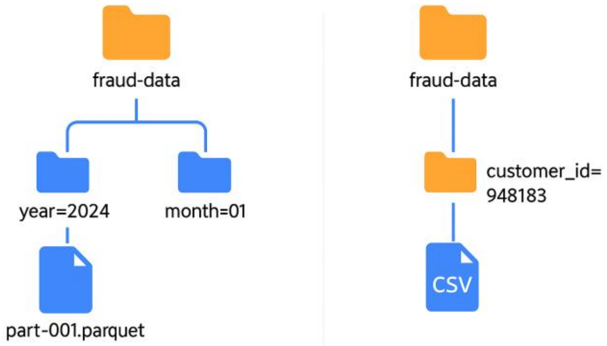
```
s3://fraud-detection-bucket/
— raw-data/
|   — year=2023/month=01/day=01/
|   — year=2023/month=01/day=02/
|   — year=2024/month=12/day=31/
— processed-data/
|   — year=2023/month=01/
|   — year=2024/month=12/
— model-artifacts/
    — training-runs/2024-01-15/
    — production-models/v1.2/
```

1. Cost Optimization:

- **Lifecycle policies**: Move data older than 30 days to IA, 90+ days to Glacier
- **File format**: Use Parquet with Snappy compression (60-80% size reduction vs JSON)
- **File sizing**: Target 128-512MB files to optimize for both S3 costs and query performance
- **Delete incomplete multipart uploads** after 7 days

2. Training Speed Improvements:

- **Date partitioning** enables efficient incremental training (load only recent months)
- **Parquet columnar format** allows reading only needed columns (timestamp, amount, merchant_type)



Good Partitioning **Bad Partitioning**

Week 1 – Part 3

Amazon EC2

3.1 AWS EC2

Learning Objectives

- **Why do machine learning engineer need EC2?**
- **Understand EC2 basics**
- **EC2 pricing models**
- **Lab 1 : Ec2 cost calculation**
- **Lab 2 : Create the Ec2 Instance**

EC2

Amazon Elastic Compute Cloud (EC2) is AWS's core computing service that provides **resizable compute capacity in the cloud**. Think of it as **renting computers on-demand** instead of buying and maintaining your own servers.

Simple Analogy

Imagine you need a computer for a project:

✓

Traditional Way

Buy a physical computer, set it up, maintain it, and it sits idle when not in use

✓

EC2 way

Rent exactly the computer power you need, when you need it, and pay only for what you use

Why do ML Engineer need EC2 ?

Challenges

- Training a deep learning model = Lots of computing power
- Processing big datasets = More memory than your laptop has
- Model inference at scale = Need multiple servers
- Peak demand vs. normal usage = Expensive to own enough hardware

✓ How Ec2 Solve this problem

8GB 8GB 16GB

- **Scalable:** Need more power? Launch bigger instances
- **Flexible:** Different workloads need different hardware
- **Cost-effective:** Pay only for what you use, when you use it
- **Fast:** New "computer" ready in minutes, not weeks
- **Global:** Run anywhere in the world

Scenario 1 : Data Scientist Development

Problem: "My laptop takes 6 hours to train this model"
EC2 Solution: Launch GPU instance, train in 30 minutes, shut down
Cost: $3 instead of buying $3,000 GPU

Scenario 2 : Batch Data Processing

Problem: "I need to process 100GB of customer data monthly"
EC2 Solution: Launch powerful instance for 2 hours monthly
Cost: $5/month instead of $5,000 server

Scenario 3 : Model Serving

Problem: "My ML API needs to handle 1000 requests/second"
EC2 Solution: Auto-scale from 1 to 10 instances based on traffic
Cost: Pay for actual usage, not peak capacity 24/7

Scenario 4 : Experimentation

Problem: "I want to test 10 different algorithms"
EC2 Solution: Launch 10 instances simultaneously, run all tests
Time: 1 hour instead of 10 hours sequential

Core EC2 Concepts

Instance = Virtual Computer

An EC2 instance is a virtual computer running in AWS data centers. Each instance has:

- CPU: Processing power for computations
- Memory (RAM): For holding data while processing
- Storage: Disk space for your data and applications
- Network: Internet connection and bandwidth

Instance Types = Different "Computer Specs"

Just like buying a laptop, you choose specs based on your needs:

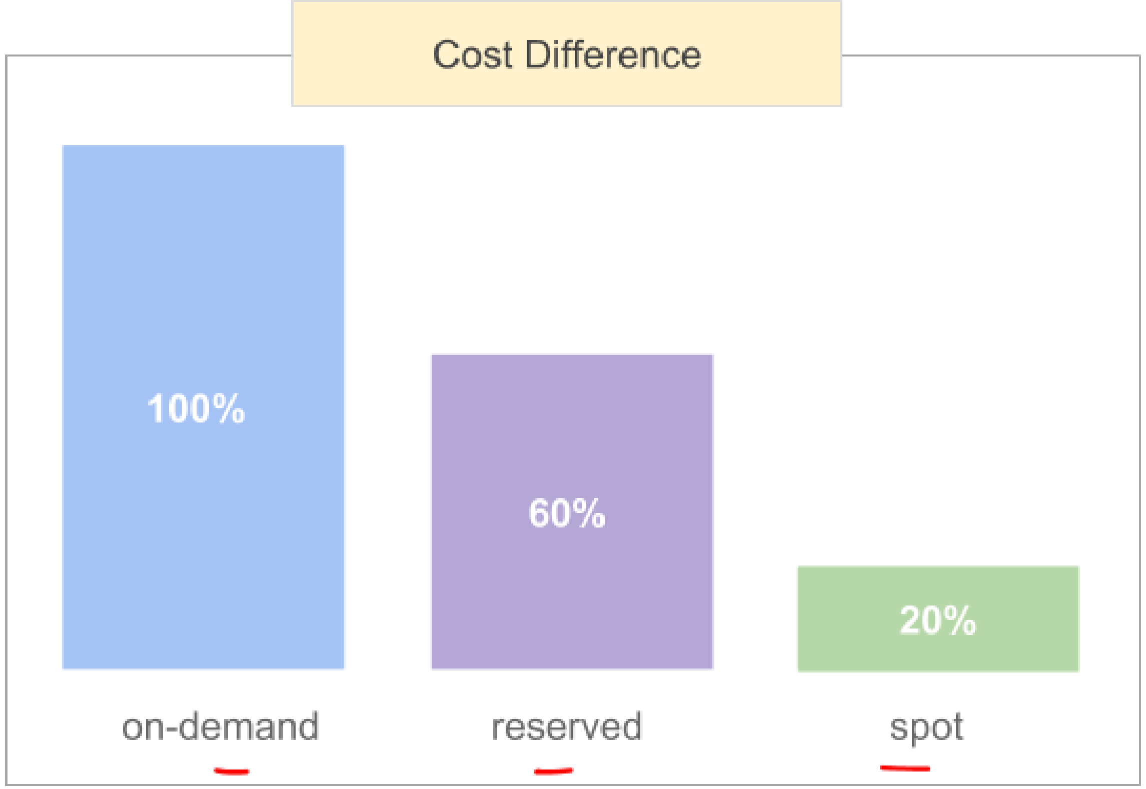
1. General Purpose (t3, m5): Balanced CPU/memory - good for development
2. Compute Optimized (c5, c6): High CPU - great for data preprocessing
3. Memory Optimized (r5, r6): Lots of RAM - perfect for big datasets
4. GPU Instances (p3, g4): Graphics cards - essential for deep learning

Regions and Availability Zones

- Region: Geographic area (e.g., US East, Europe, Asia)
- Availability Zone: Data center within a region
- Why it matters: Choose regions close to your users for better performance

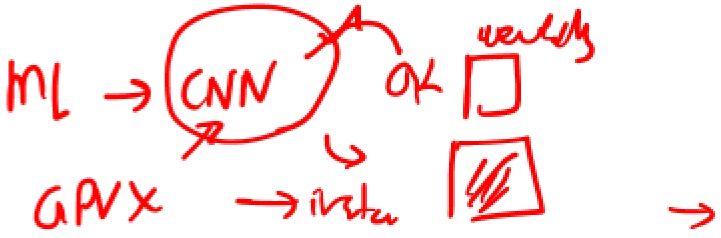
EC2 Pricing Models

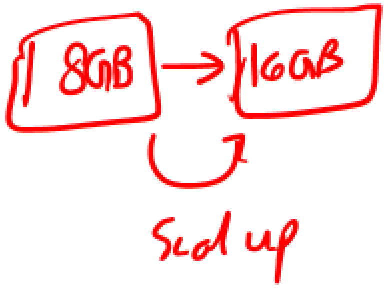
1. On-Demand (Pay-as-you-go)
2. Spot Instances (Up to 90% cheaper)
3. Reserved Instances (1–3-year commitment)



https://blog.boltops.com/2018/07/13/on-demand-vs-reserved-vs-spot-aws-ec2-pricing-comparison/

EC2 Pricing Models



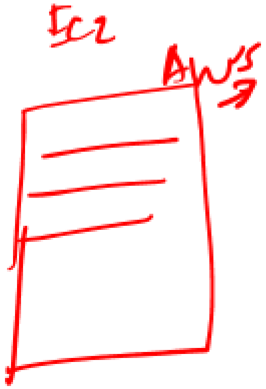


1. On-Demand (Pay-as-you-go)

- **What:** Pay by the hour/second with no commitment
- **Best for:** Development, testing, unpredictable workloads
- **Example price :** $0.50/hour for a GPU instance
- **Use case example:**
  - A data scientist launches a GPU instance to **experiment with a new model architecture (e.g. trying out a novel CNN design)** for a few hours.
  - A researcher runs **ad-hoc EDA (Exploratory Data Analysis)** on a newly collected dataset, unsure how long it'll take or how large the data will grow.

Pros	Cons
No commitment: Perfect for quick **experiments, EDA, PoC**	Most **expensive** per hour
Full availability whenever needed	Can **waste budget** if you forget to shut down
Easy to scale up/down during hacky exploration	Not ideal for long-running training

EC2 Pricing Models



2. Spot Instances (Up to 90% cheaper)

- **What:** Bid on unused EC2 capacity
- **Best for:** Training jobs that can handle interruptions
- **Example:** Same GPU for $0.15/hour (but might be terminated)
- **Use case Example** :
  - A team uses dozens of Spot instances to **parallelize hyperparameter tuning for a BERT fine-tuning job**, with checkpoints saved to S3 so progress isn't lost if interrupted.
  - A student uses Spot instances to **train multiple deep reinforcement learning agents overnight**, restarting jobs if they get interrupted.

Pros	Cons
Massively cheaper (up to 90%) → great for large training jobs	Can be terminated anytime, breaking long runs
Can run many experiments in parallel cheaply	Requires checkpointing + fault-tolerant code
Ideal for distributed hyperparameter tuning / AutoML	Unpredictable availability may block experiments

EC2 Pricing Models

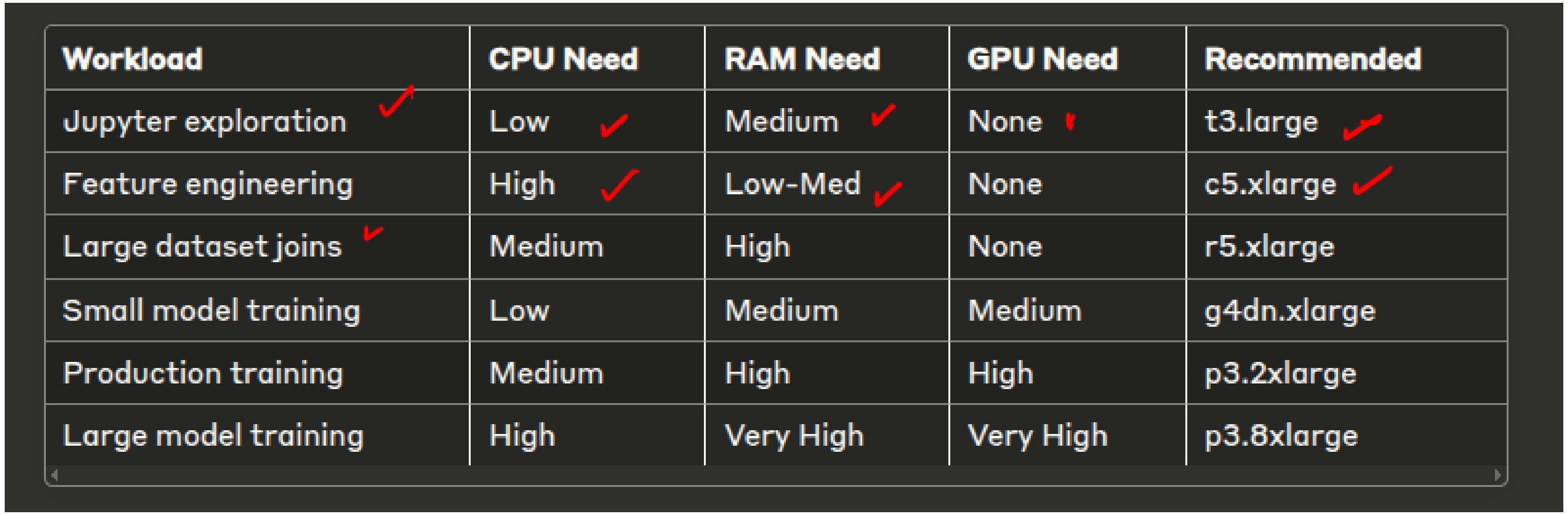
3. Reserved Instances (1–3-year commitment)

- **What:** Pay upfront for significant discounts
- **Best for:** Production workloads with predictable usage
- **Example:** Same GPU for $0.30/hour with 1-year commitment
- **Use case Example** : ✓ load → 10m
  - A company runs **a production ML inference API serving customer predictions (e.g. fraud detection or recommendation system)** 24/7 and knows the workload won't shrink.
  - An ML ops team **hosts their model monitoring and feature store infrastructure** on reserved instances to ensure capacity and stable cost over the long term.

Pros	Cons
Cost-efficient for always-on production workloads	Locked-in commitment (1–3 years)
Guaranteed capacity → stable inference latency	- Not flexible if instance types or architectures change
Good for serving models, monitoring, feature stores	Risk of underutilization if workload shrinks

Specific Instance and Tech Spec

- **t2/t3/t4g** - General Purpose (cheap experiments)
- **c5/c6** - Compute Optimized (data preprocessing)
- **r5/r6** - Memory Optimized (big data in-memory)
- **p3/p4/g4/g5** - GPU Instances (deep learning training)



Workload	CPU Need	RAM Need	GPU Need	Recommended
Jupyter exploration ✓	Low ✓	Medium ✓	None ✓	t3.large ✓
Feature engineering	High ✓	Low-Med ✓	None	c5.xlarge ✓
Large dataset joins ✓	Medium	High	None	r5.xlarge
Small model training	Low	Medium	Medium	g4dn.xlarge
Production training	Medium	High	High	p3.2xlarge
Large model training	High	Very High	Very High	p3.8xlarge

General Purpose	**t3.medium** - $0.0416/hour	2 vCPUs (1 physical core), 4GB RAM Baseline: 20% CPU, can burst to 100% Use case: Small experiments, Jupyter notebooks **When to choose:** Development, testing, cost-sensitive workloads **Avoid for:** Sustained CPU-intensive tasks
	t3.large - $0.0832/hour	2 vCPUs (1 physical core), 8GB RAM Same burst model, better for memory-hungry notebooks **Sweet spot for:** Data exploration, small model training
Compute Optimized	**c5.large** - $0.085/hour	2 vCPUs (1 physical core), 4GB RAM 3.0 GHz Intel Xeon Platinum Use case: CPU-bound preprocessing, feature engineering **Performance:** ~25% faster than t3.large for sustained loads
	c5.xlarge - $0.17/hour	4 vCPUs (2 physical cores), 8GB RAM **Best for:** Parallel data processing, scikit-learn jobs **Parallel efficiency:** Linear scaling up to physical core count
	c5.4xlarge - $0.68/hour	16 vCPUs (8 physical cores), 32GB RAM **When to choose:** Large-scale feature engineering, distributed training coordination

Memory Optimized	**r5.large** - $0.126/hour	2 vCPUs, **16GB RAM** Use case: In-memory data processing, large pandas DataFrames **Memory-to-vCPU ratio:** 8:1 (vs 4:1 for general purpose)
	r5.xlarge - $0.252/hour	4 vCPUs, **32GB RAM** **Sweet spot for:** Medium datasets (5-20GB), joins, aggregations
	r5.4xlarge - $1.008/hour	16 vCPUs, **128GB RAM** **When to choose:** Your dataset + intermediate results > 50GB
	g4dn.xlarge - $0.526/hour	4 vCPUs, 16GB RAM, **1x NVIDIA T4 (16GB GPU memory)** **Best for:** Small-medium models, inference, prototyping **GPU specs:** 2,560 CUDA cores, 320 Tensor cores **Model capacity:** BERT-base, ResNet-50, small transformers
	p3.2xlarge - $3.06/hour	8 vCPUs, 61GB RAM, **1x NVIDIA V100 (16GB GPU memory)** **Performance:** 3-4x faster training than T4 **Best for:** Production model training, large CNNs **GPU specs:** 5,120 CUDA cores, 640 Tensor cores
	p3.8xlarge - $12.24/hour	32 vCPUs, 244GB RAM, **4x NVIDIA V100 (64GB total GPU memory)** **Best for:** Large language models, distributed training **Multi-GPU efficiency:** 85-95% scaling for well-optimized models

EC2 – Lab 1

Let’s create the new EC2 Instance.

EC2 – Activity 2

Detail cost calculation.

Cost Optimization Framework for EC2

Hourly Cost = (Instance Rate) + (EBS Rate × GB) + (Data Transfer)
Daily Cost = Hourly Cost × Hours per Day
Monthly Cost = Daily Cost × Days per Month
Annual Cost = Monthly Cost × 12

Spot Savings = On-Demand Cost × (1 - Spot Discount) × Interruption Factor
Reserved Savings = On-Demand Cost × (1 - Reserved Discount)

Example 1: Model Training Comparison

Scenario: Training a computer vision model for 8 hours

Option A: p3.2xlarge On-Demand

- Instance cost: $3.06/hour × 8 hours = $24.48
- EBS cost: 100GB gp3 × 8 hours = $0.0133 × 8 = $0.11
- Data transfer: 50GB from S3 = $0.90
- Total: $25.49

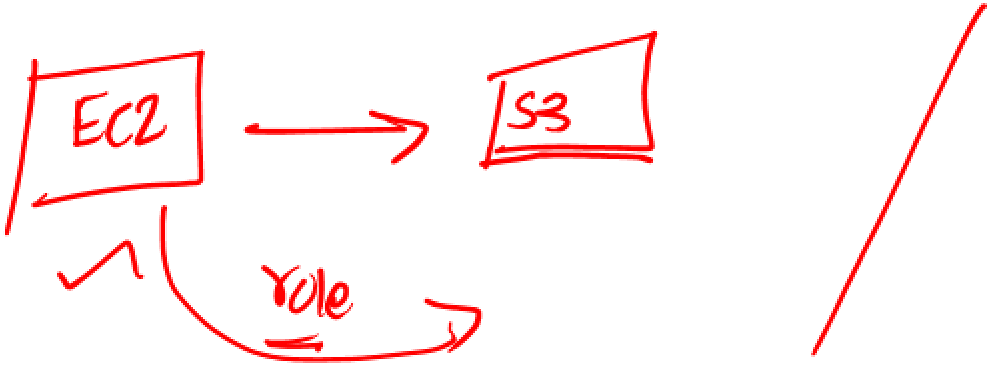
Example 1: Model Training Comparison

Example 2: Data Preprocessing Pipeline

Scenario: Processing 500GB dataset daily for 1 month

Option A: c5.4xlarge On-Demand (2 hours daily)

- Instance: $0.68/hour × 2 hours × 30 days = $40.80
- EBS: 1TB gp3 storage × 30 days = $0.08/GB × 1000GB × 30 = $2,400/month ÷ 30 = $80
- Monthly total: $120.80



EC2 – Activity 3

Access S3 bucket from Ec2 instance

EC2 – Activity 4

Access S3 bucket from Ec2 instance via instance profile

Access S3 bucket from Ec2 instance via instance profile

Step1 : Create the Policy and Create the role

- **Create role for EC2**
- IAM → **Roles** → **Create role** → Trusted entity: **AWS service**, Use case: **EC2**
- Attach the **minimum** policy you need (e.g., AmazonS3ReadOnlyAccess)

Name it (e.g., EC2-S3-ReadOnly) → **Create role**

(This also creates the instance profile with the same name.)

Access S3 bucket from Ec2 instance via instance profile

Step 2: (Run on your laptop not inside EC2)

```
> aws iam create-instance-profile --instance-profile-name EC2-S3-Access-Profile
```

```
> aws iam add-role-to-instance-profile --instance-profile-name EC2-S3-Access-Profile --role-name <your_created_role>
```

Step 3: (Update your ec2 instance)

- Go to your ec2 instance > Action > Security Group > Update IAM

Step 4: (Verify your Ec2 can access the s3 bucket via following commands)

```
> aws s3 ls s3://sagemaker-ap-southeast-1-215470142970/fashion-mnist/train/
```

⏳ You have 15 minutes to complete this task. All resources will be cleaned up after class.

AWS CLI

Command Line Interface

AWS CLI = Command Line Interface for AWS

- **Faster than clicking** through AWS Console
- **Scriptable and automatable** for repetitive tasks
- **Essential for production** ML workflows
- **Works great in Jupyter notebooks** and scripts

When You'll Use It:

- Upload/download datasets to/from S3
- Launch EC2 instances quickly
- Check costs and resource usage
- Automate ML pipeline tasks

① Install AWS CLI

② Configure Your Credentials:

```
> aws configure
```

- **Access Key ID:** Your AWS access key
- **Secret Access Key:** Your secret key
- **Default region:** us-east-1 (or your preferred region)
- **Default output format:** json (recommended)

If you're on EC2 with an IAM role, you might not need to configure credentials!

Essential S3 Commands

- **List your bucket**
  - aws s3 ls
- **List Contents of a Bucket**
  - aws s3 ls s3://your-bucket-name/
  - aws s3 ls s3://your-bucket-name/folder/ --recursive
- **Upload Files to S3:**

  s

  src dest
  - aws s3 cp my_dataset.csv s3://your-bucket-name/data/
  - aws s3 cp ./my_folder/ s3://your-bucket-name/data/ --recursive

Essential S3 Commands

- **Download files from S3**
  - src dest
  - aws s3 cp s3://your-bucket-name/data/dataset.csv ./
  - aws s3 cp s3://your-bucket-name/data/ ./local_data/ --recursive
  - aws s3 sync s3://your-bucket-name/data/ ./local_data (only newer files)
- **Sync Folders (Very Useful!):**
  - aws s3 sync ./my_project/ s3://your-bucket-name/projects/
  - aws s3 sync s3://your-bucket-name/results/ ./results/

Essential Ec2 Commands

- **Start/Stop Instances**

  -
    ```
    aws ec2 stop-instances --instance-ids i-1234567890abcdef0
    ```

  -
    ```
    aws ec2 start-instances --instance-ids i-1234567890abcdef0
    ```

  -
    ```
    aws ec2 describe-instance-status --instance-ids i-1234567890abcdef0
    ```
- Launch New Instances

  -
    ```
    aws ec2 run-instances \
     --image-id ami-0abcdef1234567890 \
     --count 1 \
     --instance-type t2.micro \
     --key-name your-key-pair \
     --security-group-ids sg-903004f8
    ```

AWS CLI in Python (Not Recommended)

```python
import subprocess
import json

# Run AWS CLI command from Python
def run_aws_command(command):
    result = subprocess.run(command, shell=True, capture_output=True, text=True)
    return json.loads(result.stdout) if result.stdout else None

# Example: List S3 buckets
buckets = run_aws_command("aws s3api list-buckets")
print(f"You have {len(buckets['Buckets'])} S3 buckets")

# Example: Upload file
upload_result = run_aws_command("aws s3 cp data.csv s3://my-bucket/")
```

Comln CLI → AWS CLI

Python → Boto3

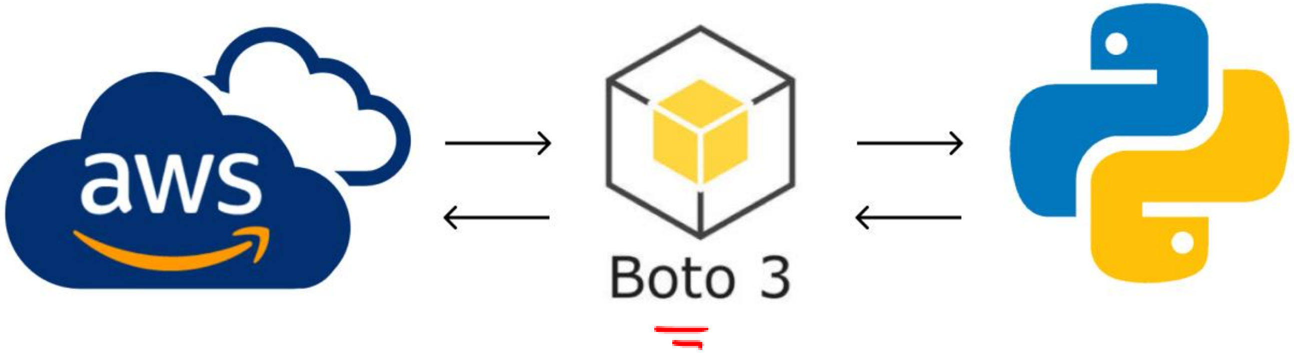
Using Boto3 (Better)

```
import boto3

# Create S3 client
s3 = boto3.client('s3')

# List buckets
buckets = s3.list_buckets()
print(f"You have {len(buckets['Buckets'])} buckets")

# Upload file
s3.upload_file('local_file.csv', 'my-bucket', 'remote_file.csv')
```

AWS CLI

Remember: AWS CLI is your friend for automation and speed. Start with these basics and add more commands as you need them!

Week 1 – Part 3



Boto3

What is boto3?

- **boto3** is the **official Python SDK (Software Development Kit) for AWS**.
- It allows Python developers to:
  - **Create**, **manage**, and **interact with AWS services** (like S3, EC2, SageMaker, IAM, etc.)
  - Automate cloud operations directly from Python scripts, notebooks, or apps

```
# AWS CLI (good for scripts)
aws s3 cp data.csv s3://bucket/

# Boto3 (better for Python programs)
import boto3
s3.upload_file('data.csv', 'bucket', 'data.csv')
```

AWS Service	Use Case	Example
S3	Stores and retrieve the data	Upload training CSVs, download model outputs
SageMaker	Model training & deployment	Launch XGBoost jobs, deploy real-time endpoints
EC2	Manage compute resources	Starts and Stops VMs
IAM	Cloud Security	Create roles for model execution
CloudWatch	Monitor logs and metrics	Track model traning logs in real time

Boto3

1. Boto3 needs AWS credentials

- **Problem**: If you don't configure credentials, boto3 won't know how to access AWS, and you'll get errors like ***botocore.exceptions.NoCredentialsError***.
- How to fix:

1. Use aws configure in CLI to set up your credentials locally

default

```
aws configure
# Enter: AWS Access Key, Secret Key, Region, Output format
```

Or set **environment variables** in your terminal:

```
export AWS_ACCESS_KEY_ID=YOUR_KEY
export AWS_SECRET_ACCESS_KEY=YOUR_SECRET
export AWS_DEFAULT_REGION=ap-southeast-1
```

Never push .aws/credentials to GitHub.

Boto3

2. Every AWS action may incur cost

- **Problem**: Spinning up SageMaker endpoints or large EC2 instances can cost money **by the hour** — and the billing **keeps running** unless you manually stop or delete them.
- How to fix:

1. **Always terminate endpoints** after inference using:

```
sm = boto3.client("sagemaker")
sm.delete_endpoint(EndpointName="your-endpoint")
```

Most people forget to delete SageMaker endpoints and get surprise

3. You must specify the correct AWS region

Problem: If you create a bucket in one region and call boto3 from another, you'll get cryptic 403/404 errors.

How to Fix

Or set **environment variables** in your terminal:

```
session = boto3.Session(region_name="ap-southeast-1")
s3 = session.client("s3")
```

Cross-region mistakes are one of the most frustrating bugs for beginners to debug.

Boto3 Security Best Practice

1. Don’t hardcode AWS keys in Notebooks

- Keys can leak if you upload your notebook to GitHub, email it, or even leave it in shared folders.

- Best practice:

- Use environment variables ✓
- Or use IAM roles ✓

2. **Use IAM Roles when possible**

- Especially useful in:
  - SageMaker Notebooks
  - EC2 instances
- These services can inherit IAM roles without needing manual key setup

3. **Follow the “Least Privilege” principle**

- Don’t give your users or notebooks full admin access
- Instead, create scoped IAM policies:
  - S3: Allow read/write to only specific buckets or prefixes
  - SageMaker: Allow only CreateTrainingJob, not full control over all services
- Example IAM policy snippet for a safe S3 + SageMaker combo:

```
{
 "Effect": "Allow",
 "Action": [
  "s3:GetObject",
  "s3:PutObject",
  "sagemaker:CreateTrainingJob",
  "sagemaker:InvokeEndpoint"
 ],
 "Resource": "*"
}
```

data/train/

Boto3 – Activity1

Checkout the jupyter notebook

→ small

✓

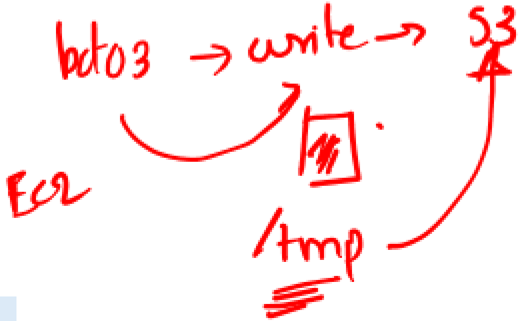
AWS Lambda

Serverless Computing Platform

Lambda

lamder_handler

bto → wrt → S3



- **No servers to manage** - AWS handles everything
- **Event-driven** - runs code in response to triggers
- **Pay per execution** - only pay when your code runs
- **Auto-scaling** - handles 1 request or 1000 simultaneously

Traditional	Lambda
EC2 instance running 24/7	Function runs on-demand
$50/month even if unused	$0.20 per 1M requests
Manual Scaling	Automatic Scaling
Server Maintenance	Zero Maintenance

Lambda Limits (Important!):

- **Timeout:** 15 minutes max
- **Memory:** 128MB to 10GB
- **Storage:** 512MB in /tmp only
- **Best for:** Inference, preprocessing, NOT training

AWS Lambda : Activity 1

Your first ML Lambda Function

AWS Lambda : Activity 2

S3 Triggered Data Preprocessing Lambda

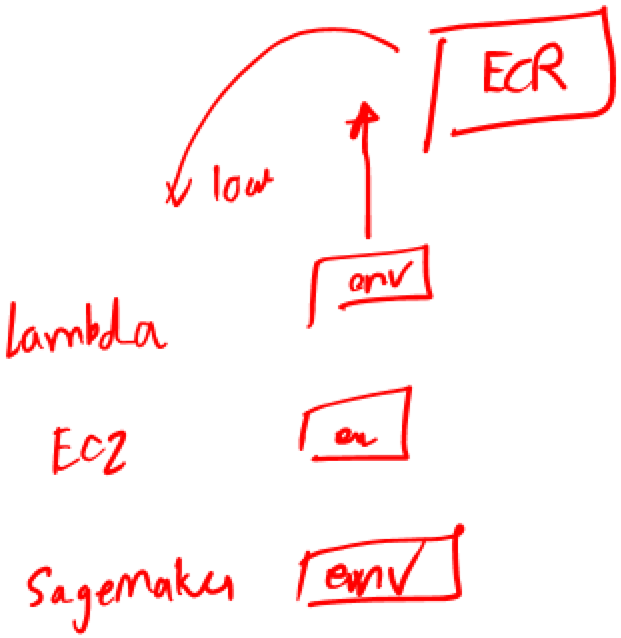
AWS Lambda : Activity 3

Real Time ML API with Lambda

Docker

AWS ECR

Elastic Container Registry



ECR

AWS

→ → AWS →

Sc → Scratch

- **Docker image storage** in the cloud
- **Version control** for your ML applications
- **Integration** with AWS services (Lambda, ECS, EKS)
- **Security** with IAM permissions and image scanning

Problem: "My ML model works on my laptop but not in production"

Solution: Container + ECR = Same environment everywhere

Traditional Deployment
- Work on my machine ✓
- Dependency conflicts ✓
- Environment Setup complexity
- Version management chaos

ECR Deployment ✓
- Identical environment
- all dependencies packaged
- Deploy with one command
- Tagged, version image

Perfect ML Use Cases:

- **Custom ML environments** with specific Python versions
- **Model serving** with complex dependencies
- **Reproducible training** environments
- **Team collaboration** with shared environments
- **Production deployment** with Lambda containers

Prerequisites

Docker Basics

What is Docker ?

Docker = Package your application + all dependencies into a container

Key Docker Concepts :

Dockerfile → Instructions to build image
Image → Template/snapshot of your application
Container → Running instance of an image
Registry → Storage for images (ECR is AWS's registry)

Basic Docker Commands:

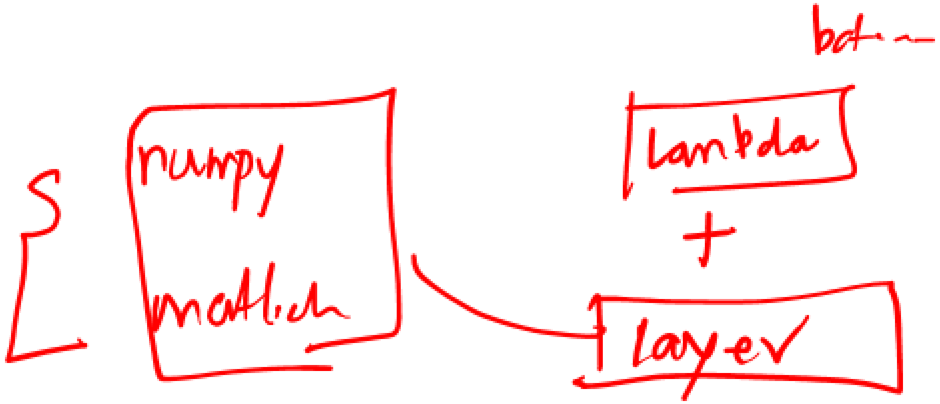
```
docker build -t my-app .          # Build image from Dockerfile
docker run my-app                 # Run container from image
docker images                     # List local images
docker ps                         # List running containers
docker tag image-name:tag new-name:tag  # Tag image
docker push repository-url        # Push to registry
```

Install Docker (if not installed) :

- https://docs.docker.com/engine/install/

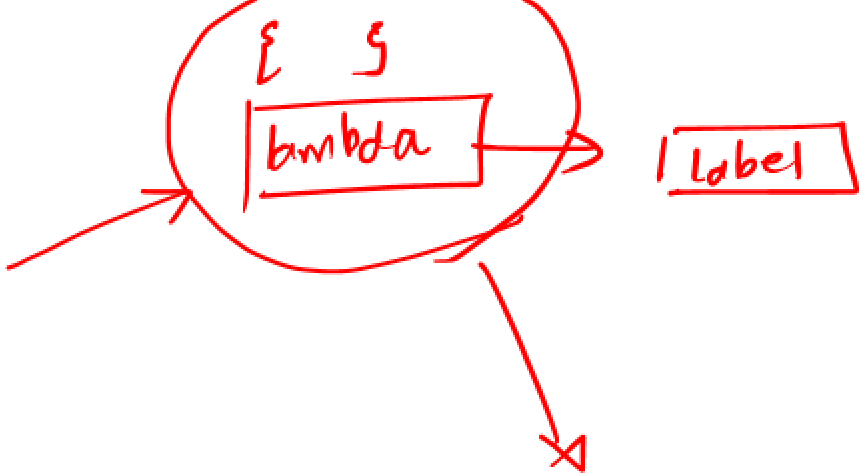
AWS ECR: Activity 1

First ML Container + ECR



AWS Lambda: Activity 2

Use the external library as image and attached to Lambda as Layer



Sagemaker

AWS API Gateway

The first door for your ML Services

What is API Gateway ?

1M → 0.0024

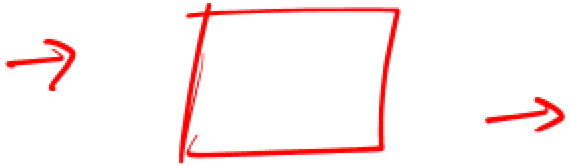
API Gateway = The front door for your ML Services

Key Docker Concepts :

- REST and HTTP APIs for your Lambda functions
- WebSocket APIs for real-time ML inference
- Authentication & Authorization (API keys, Cognito, IAM)
- Rate limiting and throttling for production safety
- Request/Response transformation for data preprocessing

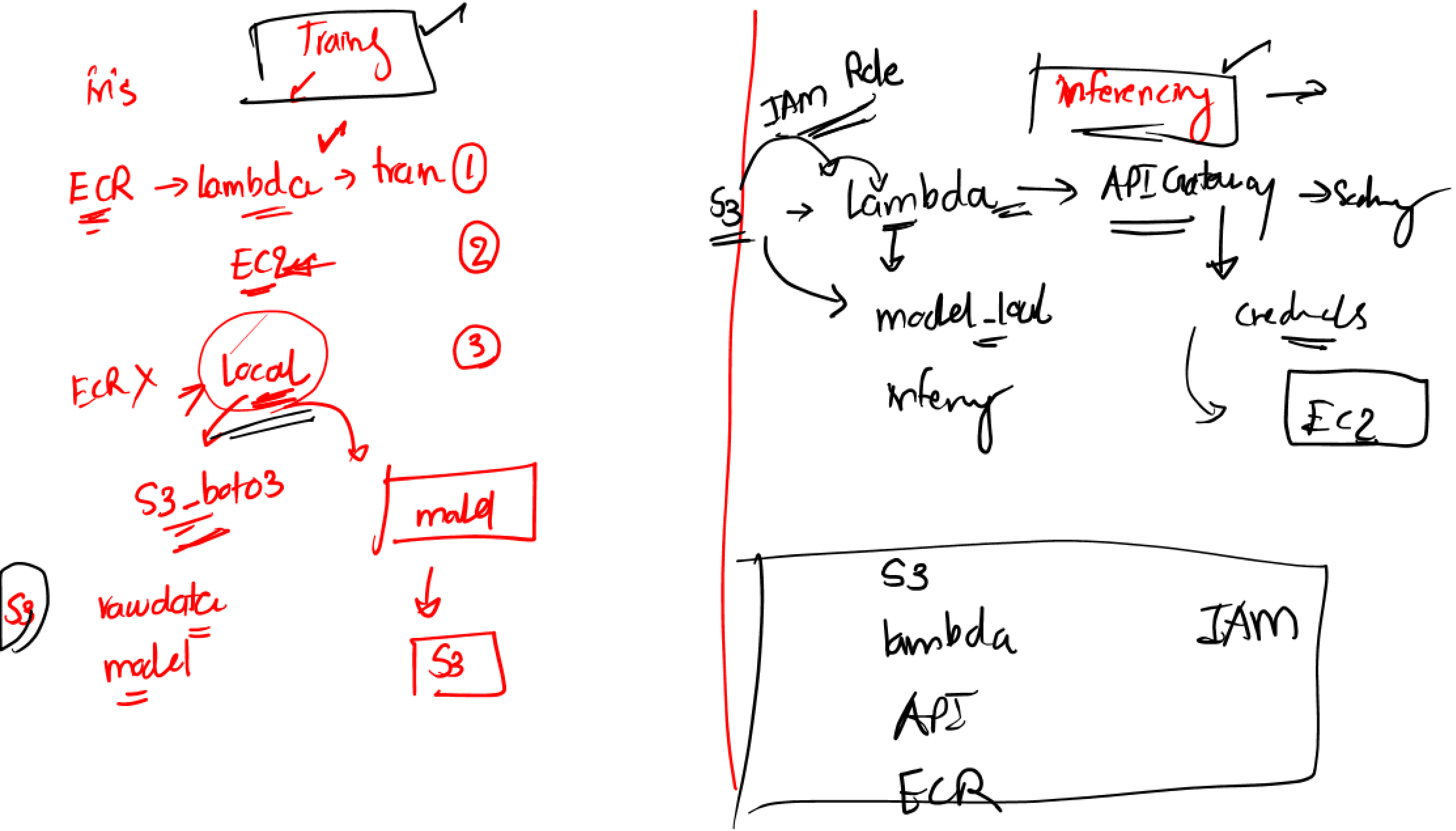
Use Cases

- Model serving APIs with authentication
- Batch processing endpoints with file upload
- Real-time inference with WebSocket streaming
- A/B testing with traffic splitting
- Multi-model serving with routing



API Gateway: Activity 1

Connect Lambda ML Function to API Gateway


Traing
iris
ECR → lambda → train ①
EC2 ②
local ③
ECR X
S3_boto3
model
S3
rawdata
model
S3
IAM Role
Inferencing
S3 → lambda → API Gateway → Scaling
model_load
infering
credntls
EC2
S3
lambda
API
ECR
IAM