Cloud computing

Martial Luyts

Catholic University of Leuven, Belgium

martial.luyts@kuleuven.be

KU LEUVEN

LEUVEN STATISTICS
RESEARCH CENTRE

Contents

Part 0:

Introduction & outline of this class

0.1 Introduction

- Imagine you are a data scientist working on a large machine learning model.
- You set it up (locally or containerized), and train it.
- **Problem:** Computation time!

- **Solution:** Use cloud computing services for speeding up your work.

Example:

- Training a large model on **Amazon Web Services (AWS)**, i.e., a cloud service provider, can reduce time from 10 hours to only 30 minutes

0.2 Outline of the class

- This presentation is specifically designed for data science practitioners, who want to learn more about the basic building blocks of cloud computing, i.e., one of the key building blocks of MLOps
- Prior knowledge of Python is required.
- By the end of this class, we will:
  - Explore what cloud computing is
  - Compare major **cloud platforms**

- Discuss different **cloud service models**
- Explore different **types of cloud services** for data science
- Step-by-step AWS account setup and management
- A walkthrough of setting up a data science project on AWS

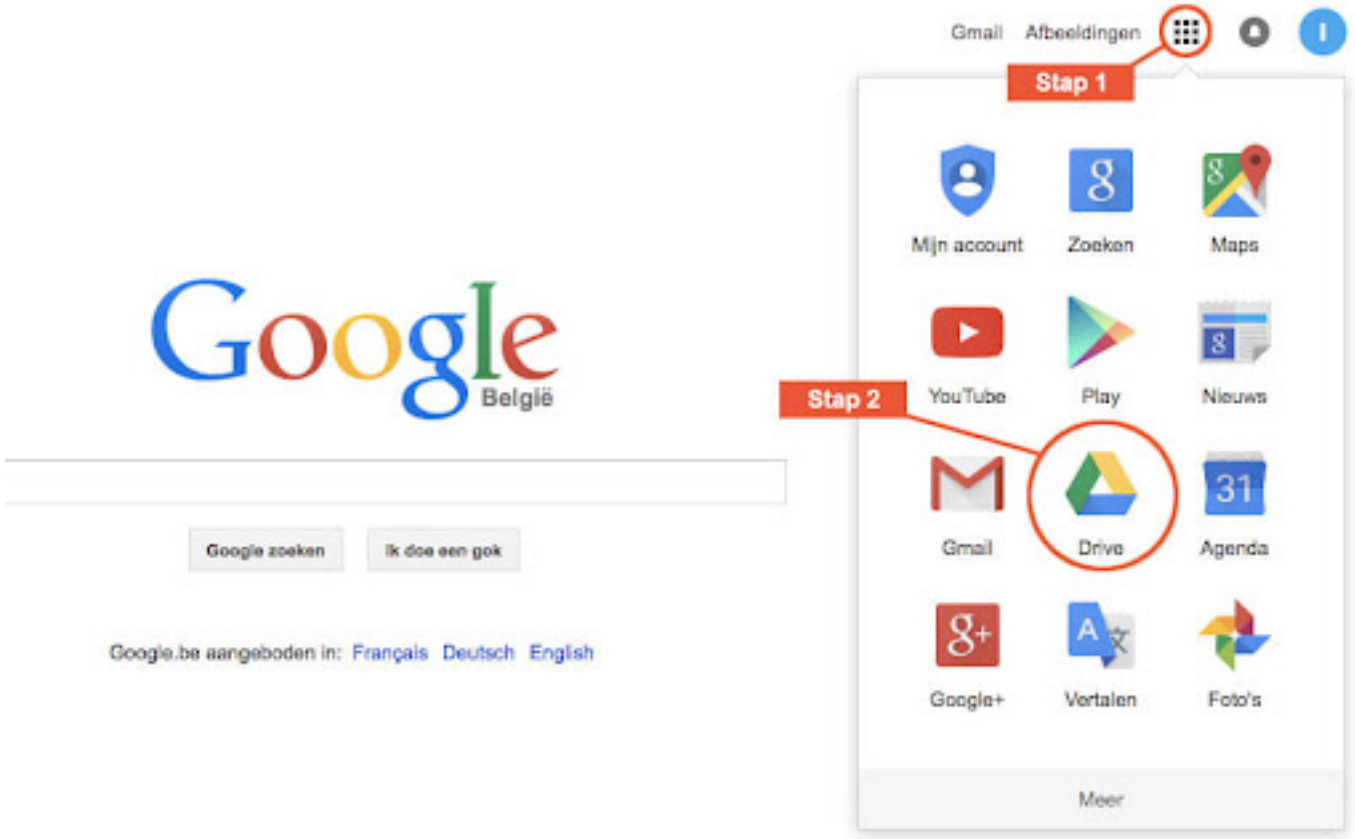
Part 1:

Cloud computing overview

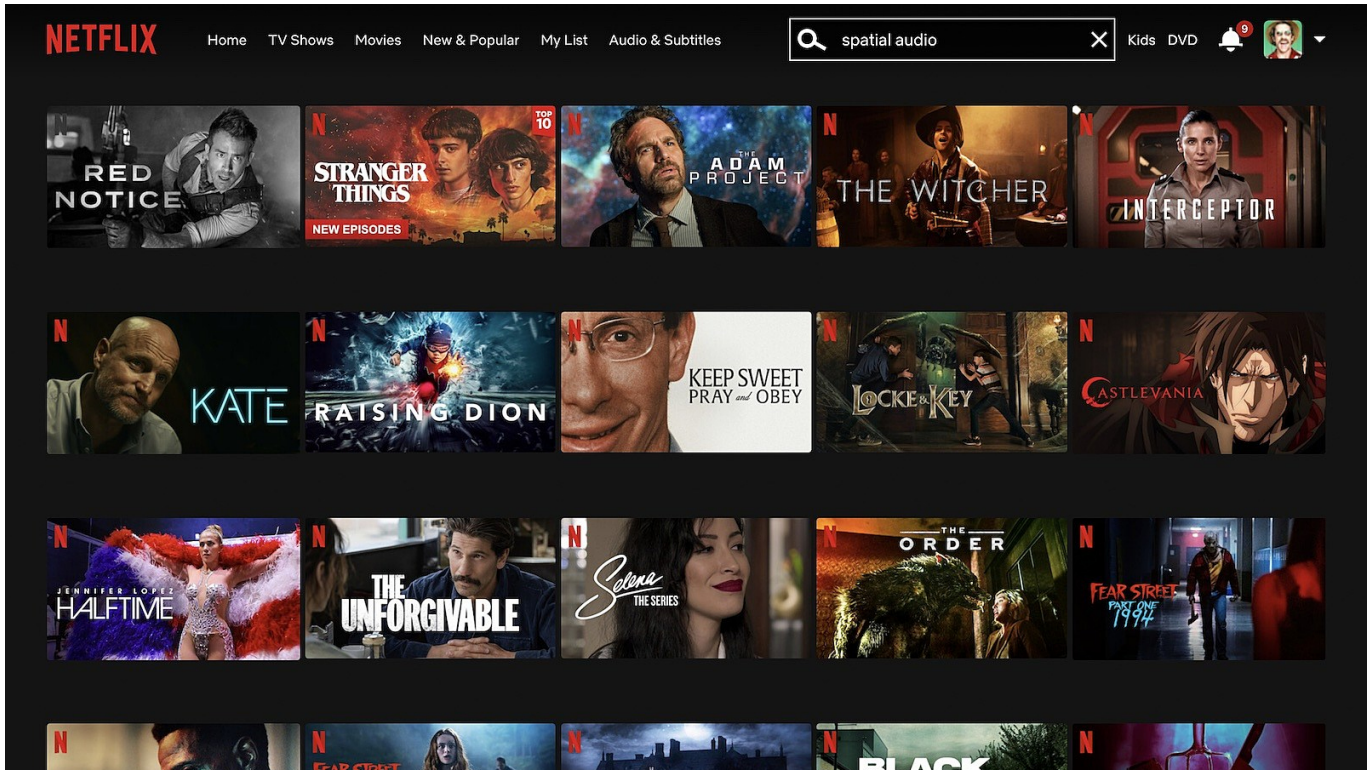
1.1 What is Cloud Computing?

- **On-demand access** to **computing resources** via the internet
- It delivers IT resources s.a. compute, storage, databases, analytics, and more
- Cloud computing is foundational to modern data science

- **Examples:**
  - **Google Drive** (Cloud storage)



- **Netflix** (Cloud-hosted content delivery)

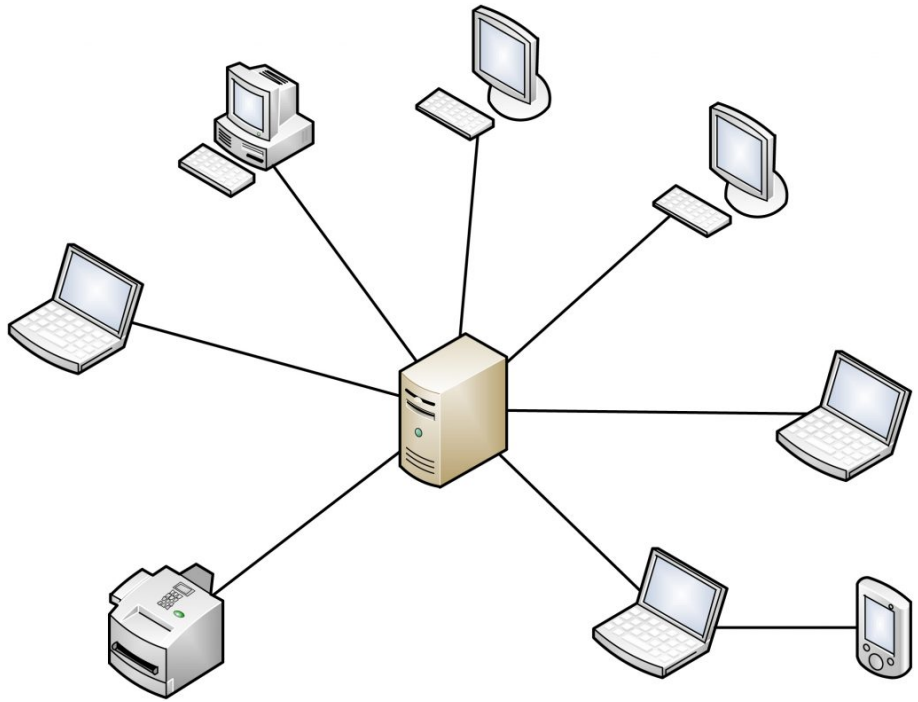


- **Question:** But how did it started?

- **Answer:**
  - *In the beginning (1980s and 1990s)*, many companies relied on an **on-premises data center**, i.e., a group of servers that were privately owned and controlled by them.

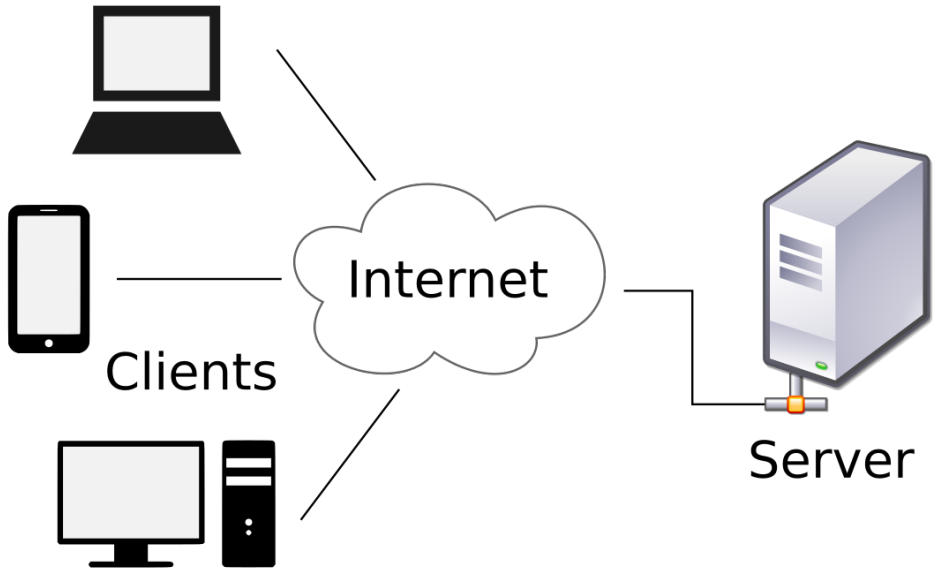
  - The management and maintenance of these servers was handled by the firm itself.

- Companies used **client-server computing**, meaning that their apps were deployed on servers (their on-premise data center) in the same building as the users



- **Drawback:** You had to physically attend the site to access the server

- Due to the rise of internet, the switch was made *in 1990s and 2000s* to **internet and web-based hosting**, i.e., elimination of the need for proximity to the servers, but still own maintenance.

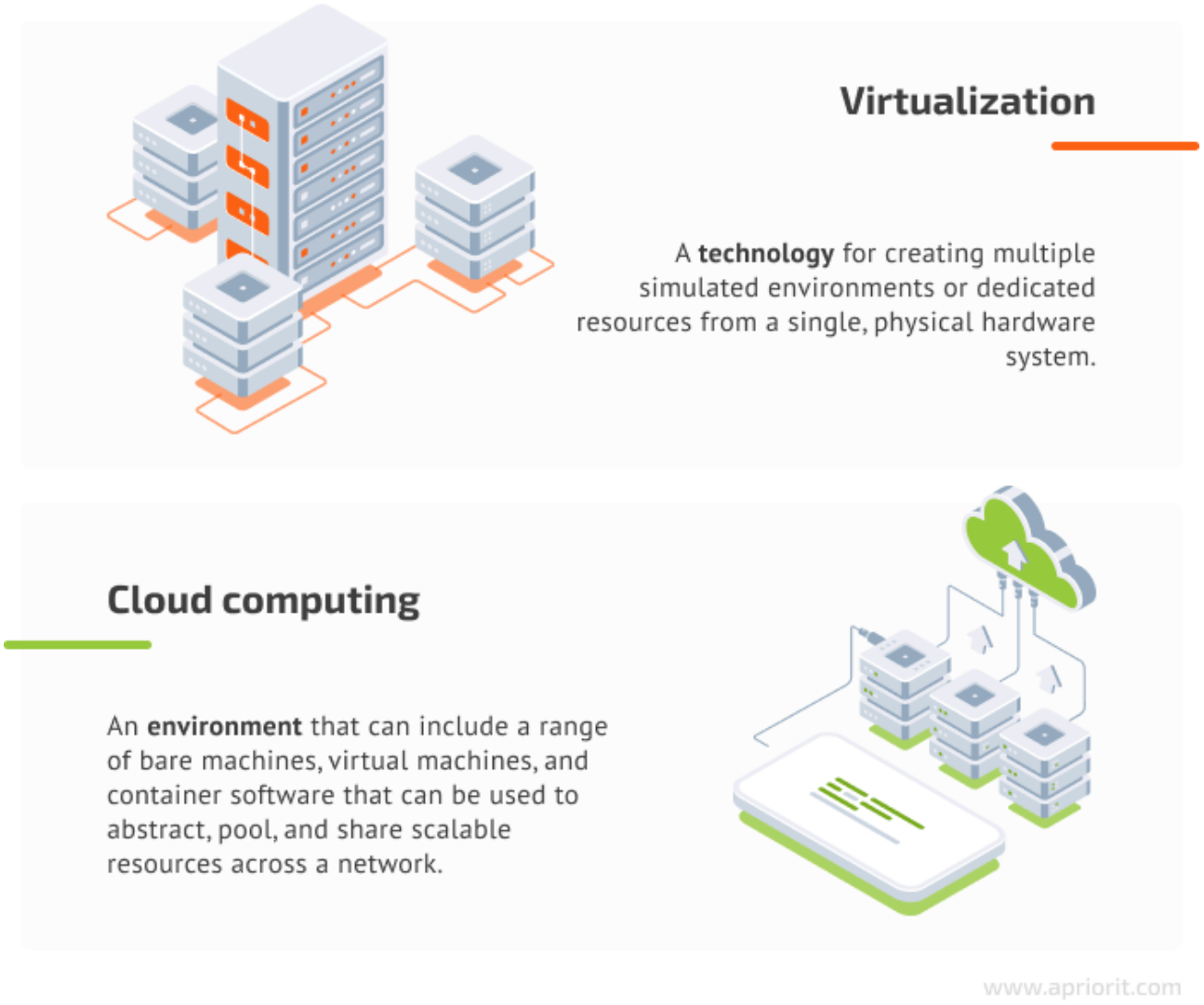


- The access of computing resources and applications could be done **over the internet**, using *browsers and standard protocols*.

- *In 2000s and 2010s*, the area of **virtualization and cloud computing** was introduced.
  - Data centers were build by big companies like Amazon, Microsoft and Google.

  - Other companies can access these with **virtualization technology and cloud computing platforms** to seperate their IT infrastructure from the underlying hardware and deploy it on these shared, scalable, and pay-per-use resources

- **Advantage:** Focus on their core business instead of IT management.



1.2 Benefits of cloud computing

Several benefits of cloud computing include:

- **Scalability:** Increase computation power when needed
- **Flexibility:** Run notebooks, models, pipelines from anywhere
- **Cost efficiency:** Pay for what you use

1.3 Use Cases in Data Science

- Storing and querying large datasets (e.g., S3, BigQuery)
- Machine learning training at scale (e.g., SageMaker, Vertex AI)
- Data engineering and ETL pipelines (e.g., AWS Glue, Dataflow)
- Model deployment and monitoring
- Real-time analytics (e.g., Kinesis, Pub/Sub)

Part 2:

Cloud Service Providers, Platforms & Services

2.1 Cloud Service Providers

- **Cloud services providers** offer on-demand computing resources s.a. servers, storage, databases, and AI tools, *over the internet*
- They help individuals, teams, and companies build, run, and scale applications without owning physical hardware
- Providers differ in what they specialize in:
  - Some offer broad infrastructure (**IaaS**; see Section 2.2)
  - Others focus on developer tools (**PaaS**; see Section 2.2)
  - Or deliver ready-to-use software (**SaaS**; see Section 2.2)

- Choosing the right provider depends on your needs and comfort, such as:
  - Budget
  - Data science workflow
  - Integration with existing tools
  - Performance and scalability
- In what follows, we will discuss some main providers, and what makes them different

1. Amazon Web Services (AWS)

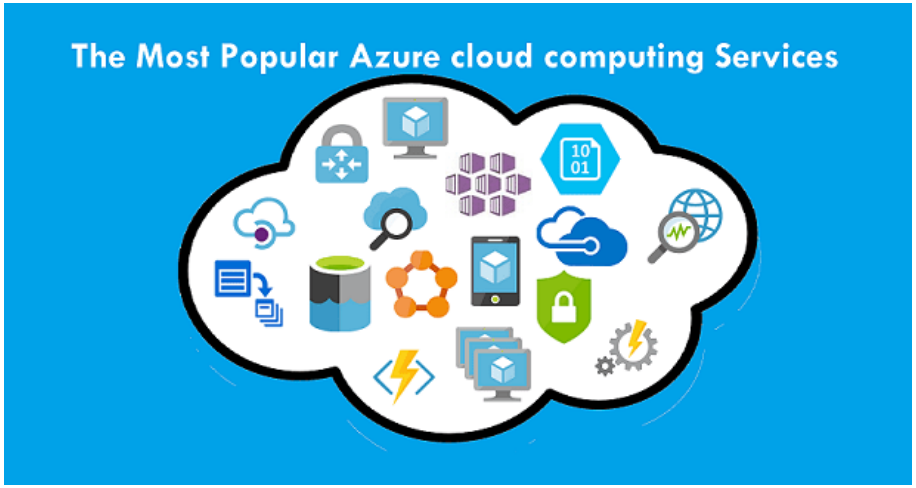
- Largest cloud provider
- Launched in 2006 by Amazon
- Amazon's flagship cloud computing service, generating over 80 billion dollar in revenue in 2022

- Excellent support for AI and ML
- Broadest range of services (200+)
- **Popular services:** Elastic Compute Cloud (EC2), Simple Storage Service (S3), Lambda, SageMaker

2. Microsoft Azure

- Launched in 2010 by Microsoft
- Seamless integration with Microsoft tools (Windows, Office 365, ...)
- Shines in **hybrid cloud setups**, i.e., combining on-premise data centers with cloud environments

- Data science friendly (supports Jupyter notebooks, containers, Kubernetes, etc.)
- Trusted by governments and highly regulated industries, due to its enterprise-grade security and compliance
- **Popular services:** Azure Virtual Machines, Azure Functions, Azure Machine Learning



3. Google Cloud Platform (GCP)

- Launched in 2011 by Google
- Best in-class for AI, data analytics, and Kubernetes

- Simple, developer friendly user experience
- Build on Google's global infrastructure, which is highly secure and reliable.
- **Popular services:** Compute Engine, Cloud Storage, Cloud Functions, Vertex AI

Overview of the most popular services between them:

4. Other notable providers

While AWS, Microsoft Azure and GCP are the most used and known cloud service providers, there are also other notable providers:

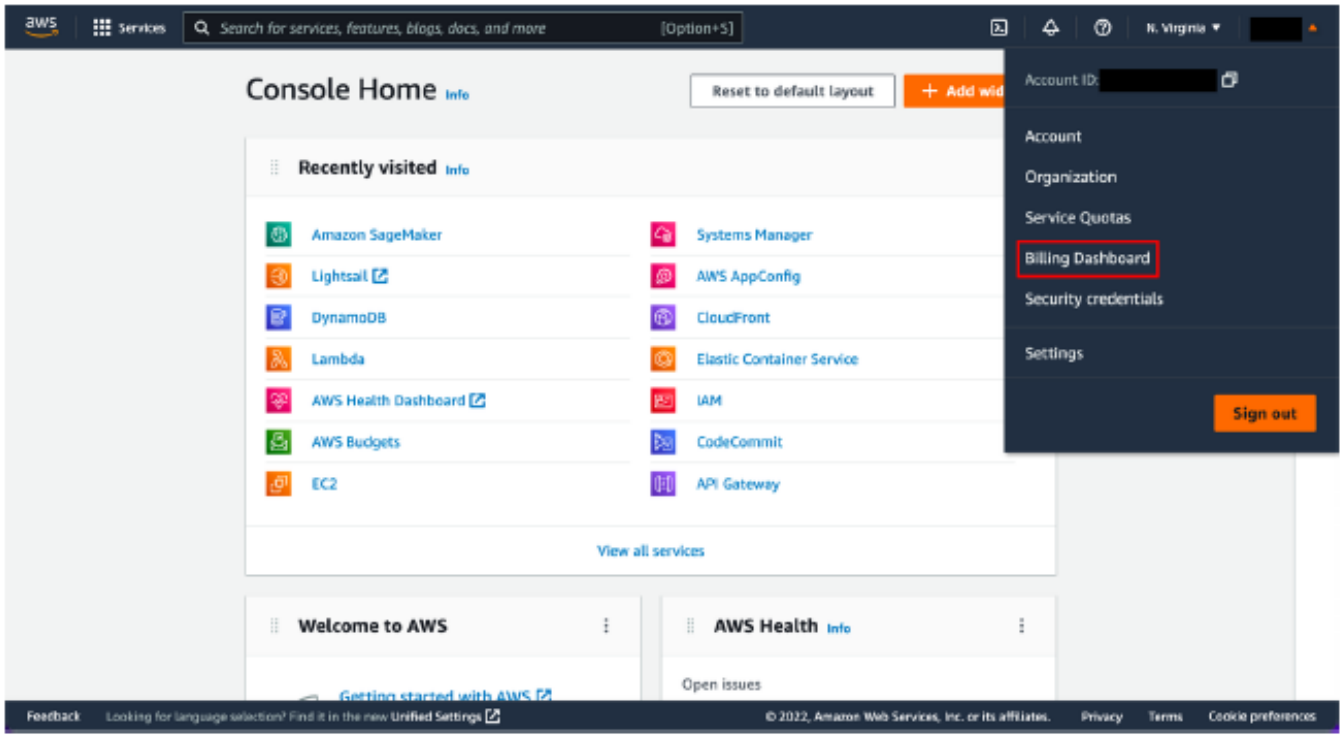
- **IBM cloud:**
  - Strong in AI and hybrid cloud
  - Focused on regulated industries
- **Oracle cloud:**
  - Optimized for databases and Oracle-based enterprise software

- **Alibaba cloud:**
  - Leader in China and Asia
  - Strong presence in e-commerce and global markets
- **DigitalOcean:**
  - Developer-friendly, simple and cost-effective
  - Ideal for small teams and startups
- **Heroku:**
  - Simple platform-as-a-service (PaaS; see later)
  - Quick deployments, great for prototypes

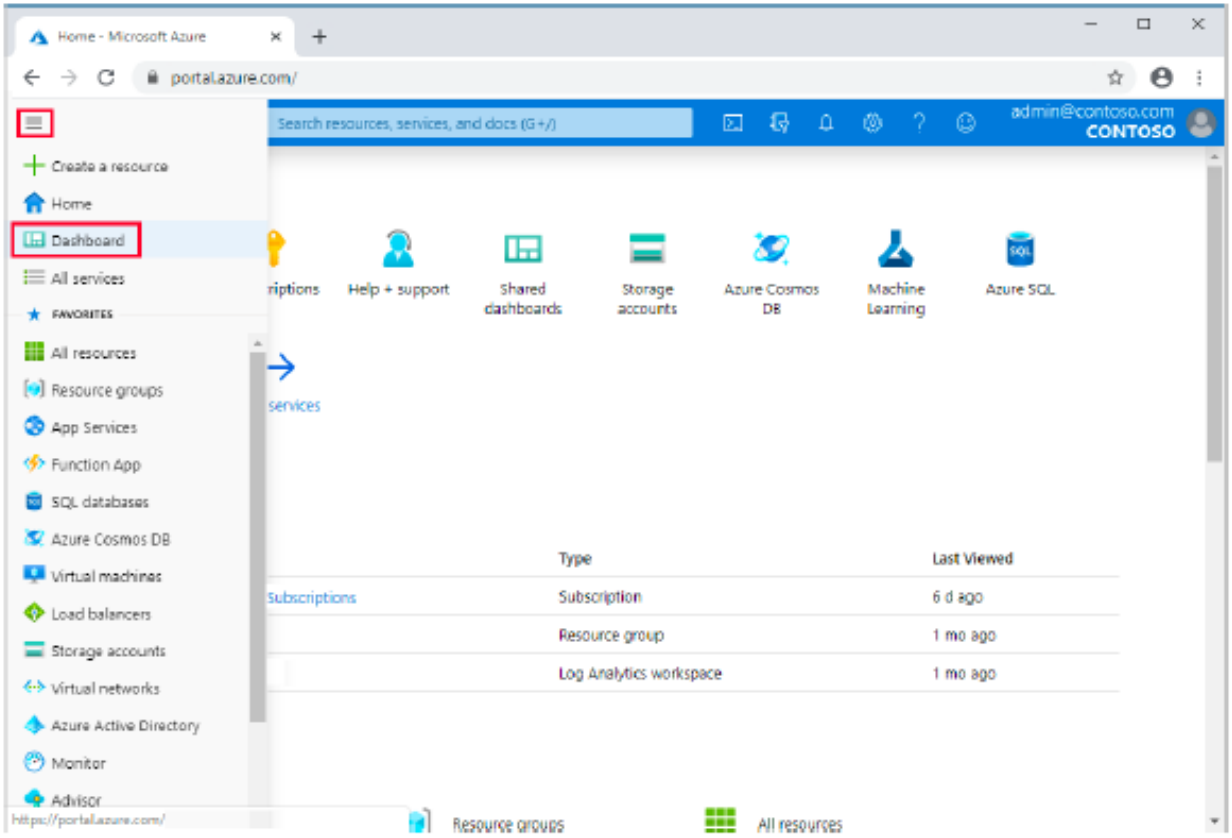
One of the primary characteristics of cloud service provider is the ability to provision virtualized infrastructure resources using a **self-service management tool**.

Example: AWS offers such tools in the form of its

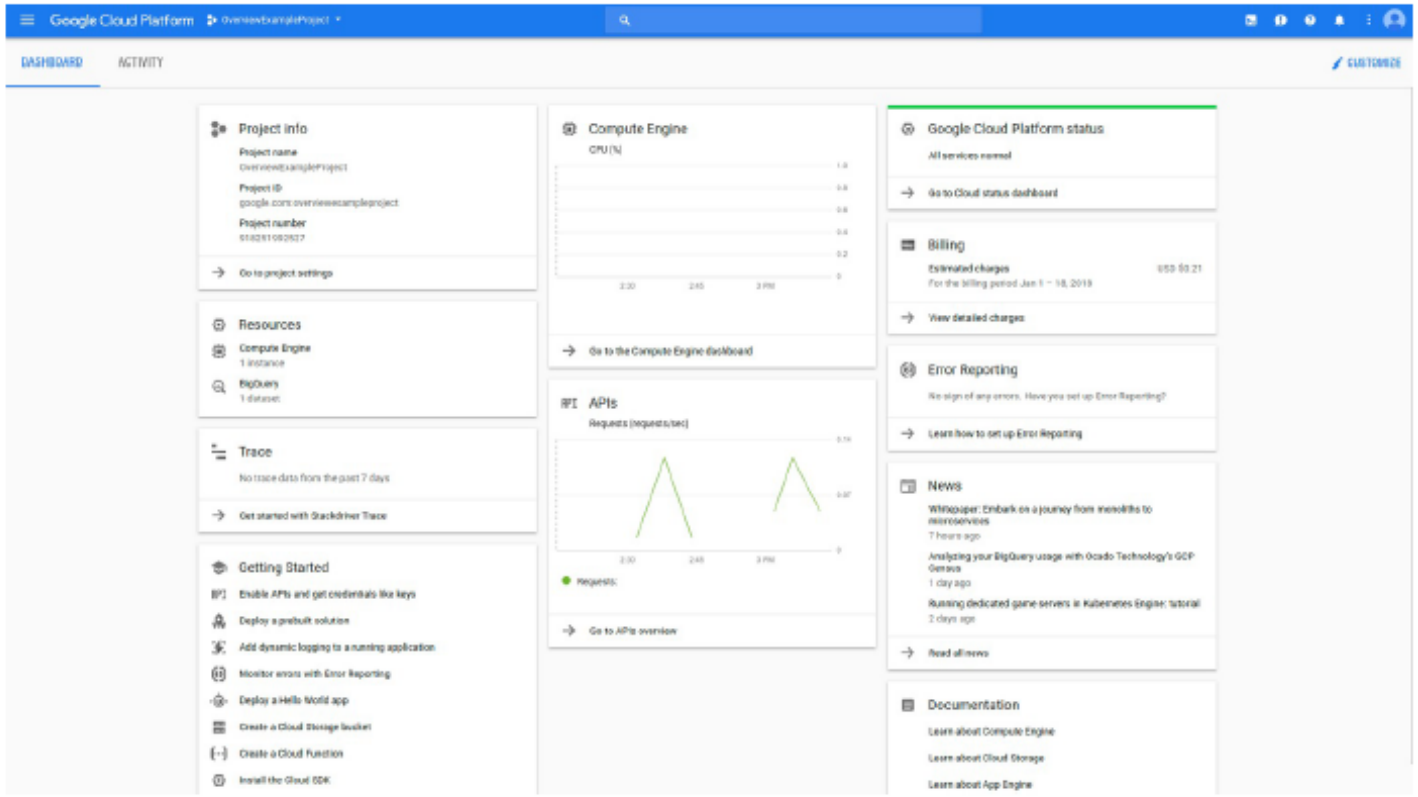
- **Management Console** (accessible via a web browser)
- **Command-line interface (CLI)**
- **Software Development Kit (SDK)**


Services
Search for services, features, blogs, docs, and more
[Option+S]
N. Virginia
Console Home Info
Reset to default layout
Account ID:
Account
Organization
Service Quotas
Billing Dashboard
Security credentials
Settings
Sign out
Recently visited Info
Amazon SageMaker
Lightsail
DynamoDB
Lambda
AWS Health Dashboard
AWS Budgets
EC2
Systems Manager
AWS AppConfig
CloudFront
Elastic Container Service
IAM
CodeCommit
API Gateway
View all services
Welcome to AWS
AWS Health Info
Open issues
Feedback
Looking for language selection? Find it in the new Unified Settings
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Microsoft
Azure


Home - Microsoft Azure
portal.azure.com/
Search resources, services, and docs (G+/)
admin@contoso.com
CONTOSO
Create a resource
Home
Dashboard
All services
FAVORITES
All resources
Resource groups
App Services
Function App
SQL databases
Azure Cosmos DB
Virtual machines
Load balancers
Storage accounts
Virtual networks
Azure Active Directory
Monitor
Advisor
Help + support
Shared dashboards
Storage accounts
Azure Cosmos DB
Machine Learning
Azure SQL
services
Type
Last Viewed
Subscriptions
Subscription
6 d ago
Resource group
1 mo ago
Log Analytics workspace
1 mo ago
Resource groups
All resources


Google Cloud Platform

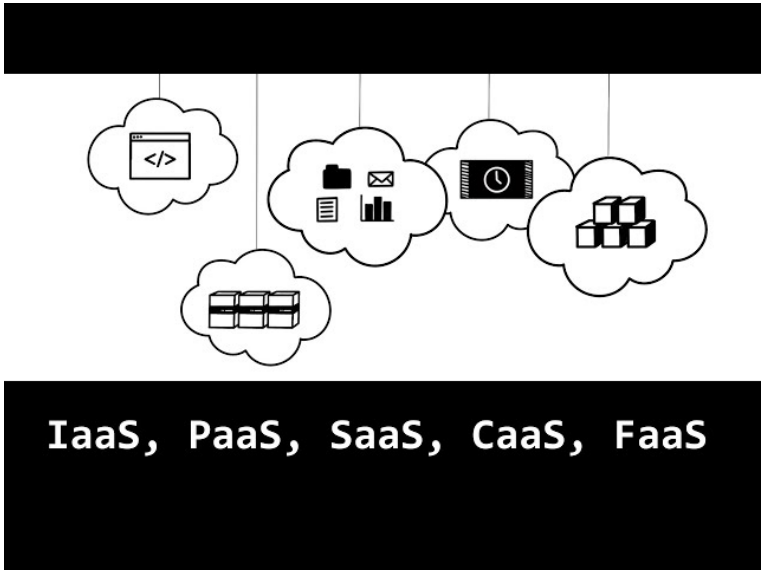

Google Cloud Platform
DASHBOARD
ACTIVITY
CUSTOMIZE
Project info
Project name
Project ID
Project number
Go to project settings
Resources
Compute Engine
1 instance
BigQuery
1 dataset
Trace
No trace data from the past 7 days
Get started with Stackdriver Trace
Getting Started
Enable APIs and get credentials like keys
Deploy a prebuilt solution
Add dynamic logging to a running application
Monitor errors with Error Reporting
Deploy a Hello World app
Create a Cloud Storage bucket
Create a Cloud Function
Install the Cloud SDK
Compute Engine
CPU (%)
Go to the Compute Engine dashboard
APIs
Requests (requests/sec)
Requests
Go to APIs overview
Google Cloud Platform status
All services normal
Go to Cloud status dashboard
Billing
Estimated charges
View detailed charges
Error Reporting
Learn how to set up Error Reporting
News
Whitepaper: Embark on a journey from monoliths to microservices
7 hours ago
Analyzing your BigQuery usage with Ocado Technology's GCP Genava
1 day ago
Running dedicated game servers in Kubernetes Engine: tutorial
2 days ago
Read all news
Documentation
Learn about Compute Engine
Learn about Cloud Storage
Learn about App Engine

2.2 Cloud Service models

Cloud providers deliver their services in different ways by means of **cloud service models**, which defines the level of control and responsibility users have for each service.

These include:

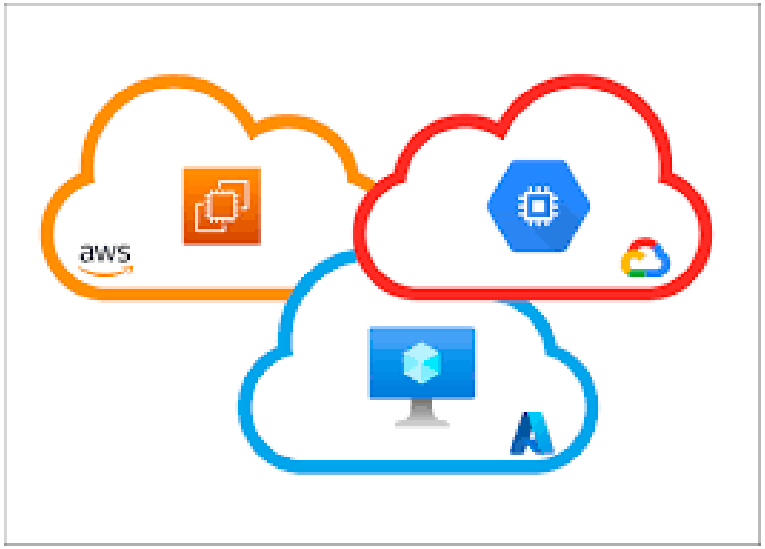
- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)
- Function as a Service (FaaS)
- Container as a Service (CaaS)



In what follows, we will discuss them in detail!

1. Infrastructure as a Service (IaaS)

- Provides **virtualized computing resources** over the internet
- User manages operation system, runtime, applications
- **Services:** Amazon EC2, Google Compute Engine, Azure Virtual Machines



- **Example:**
  You are a data scientist and want to train a ML model on a huge dataset. Your laptop is too slow or doesn't have enough memory.

  With **IaaS**, you can:

  - **Rent a powerful virtual machine** on AWS (e.g., with GPUs and lots of RAM)
  - **Install your tools** (e.g., Python, Jupyter and scikit-learn)
  - **Upload your data** and train your model
  - **Shut it down** when you are done!

  In other words, you didn't buy any hardware, but just borrowed it for a few hours.

- **Advantages:**
  - Full control over the environment
  - Flexibility to install any software
- **Disadvantages:**
  - More management overhead (patching, security)

2. Platform as a Service (PaaS)

- Provides a **managed environment** for application development and deployment
- User focuses on code and logic
- With **PaaS**, the cloud provider gives you not just the infrastructure (like with IaaS; e.g., the raw kitchen equipment), but also the tools and setup (ingredients and tools) to build and run your app (recipe) easily, so you don't need to worry about installing software, setting up servers, or managing updates (cooking your recipe from scratch)
- **Services:** AWS Elastic Beanstalk, Google App Engine, Azure App Services

- **Example:**
  You want to build a web app that shows the predictions of your ML model.

  With **PaaS**, you can:

  - Use a service like **Google App Engine**
  - Upload your code and model
  - The platform takes care of:
    - Running your app
    - Scaling it when many people use it
    - Keeping everything updated and secure

  So instead of managing servers, you focus only on writing your app and using your model!

- **Advantages:**
  - Fast deployment
  - No infrastructure management
- **Disadvantages:**
  - Limited flexibility compared to IaaS

3. Software as a Service (SaaS)

- Fully **managed software products** accessed via the internet
- It's like going to a restaurant. The food (software) is ready, you just order and enjoy, meaning no cooking, no dishes, and no cleanup.
- **Services:** Google Workspace, Tableau Online, Salesforce

- **Example:**
  You want a dataset and train a model, but don't want to write code or install anything.

  With **SaaS**, you can:

  - Use **Google Colab**, **DataRobot**, or **BigML**
  - Upload your dataset
  - Use point-and-click tools or simple notebooks
  - Get your results and download your model or graphs.

  You didn't write server code, set up Python, or manage any infrastructure. The software handled everything!

- **Advantages:**
  - No setup or maintenance
  - Accessible from anywhere
- **Disadvantages:**
  - Minimal customization

4. Function as a Service (FaaS)

- **Event-driven compute model**
- Code runs in response to events without server provisioning
- It's like a light switch with motion detection. The light (your function) turns on only when someone walks in. When no one's there, it is completely off, saving energy (and money).
- **Services:** AWS Lambda, Google Cloud Functions, Azure Functions

- **Example:**
  You have a trained ML model, and want to use it to make predictions when someone uploads new data.

  With **FaaS**, you can:

  - Write a small function like `predict(input_data)`
  - Deploy it using **AWS Lambda** or **Google Cloud functions**
  - Whenever someone sends new data, the cloud runs the function and returns the prediction
  - When there is no request, nothing is running, and you pay nothing

  You don't set up or manage any server, just the function!

- **Advantages:**
  - Cost-efficient for intermittent workloads
  - Highly scalable automatically
- **Disadvantages:**
  - Cold start issues
  - Limited runtime duration

5. Container as a Service (CaaS)

- Focused on **containerized applications**
- Combines benefits of IaaS and PaaS
- You package your code, dependencies, and environment into a container, and a cloud service runs and manages those containers for you.
- **Services:** AWS Elastic Container Service (ECS), Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS)

- **Example:**
  You have built a data science model that depends on specific Python libraries and a certain version of pandas and scikit-learn.

  With **CaaS**, you can:

  - Put your code, model and environment into a Docker container
  - Deploy it using a CaaS platform like **AWS Fargate**, **Google Cloud Run**, or **Azure Container Instances**
  - The platform runs your container whenever needed. Scalable and without managing servers.

  You control exactly what's inside the container, but the cloud handles the running and scaling it!

- **Advantages:**
  - Flexible, portable, scalable applications
  - Easy migration between platforms
- **Disadvantages:**
  - Requires container orchestration knowledge

2.3 Cloud Services

Within a cloud service model, Cloud providers often offer a range of cloud services where the user can use from.

These can be grouped into different categories, *depending on its specific function*:

- **Compute services**
- **Storage services**
- **Databases services**

- **AI/Machine Learning (ML) services**
- **Analytics/Big Data services**
- **DevOps/Continuous Integration (CI)-Continuous Delivery/Deployment (CD)**
- **Networking and Security services**

In what follows, we will now discuss the most known services, for the top 3 cloud providers, i.e., AWS, Microsoft Azure and GCP!

1. Compute services

Compute services provide the processing power to run your *applications, scripts, models, and servers*.

- **Virtual Machines:**
  → EC2 (IaaS), Azure Virtual Machine (IaaS), Compute Engine (IaaS)
- **Batch computing:** Run large-scale, parallel jobs
  → AWS Batch (PaaS/CaaS), Azure Batch (PaaS), Batch (PaaS/CaaS)
- **Serverless functions:**
  → Lambda (FaaS), Azure Functions (FaaS), Cloud Functions (FaaS)
- **Container Services:**
  → Fargate (CaaS), Azure Container Instance (CaaS), Cloud Run (CaaS)

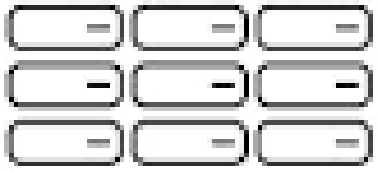
2. Storage services

Storage services let you store and retrieve files, datasets, and backups.

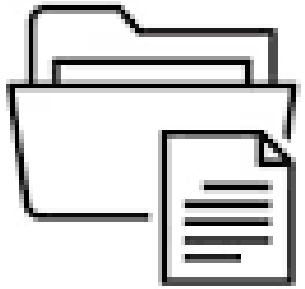
- **Object storage:** Ideal for unstructured data like images, videos
  → Simple Storage Service (S3) (IaaS/PaaS), Azure Blob Storage (IaaS/PaaS), Google Cloud Storage (IaaS/PaaS)

- **Block storage:** Like a virtual hard drive, used with VMs
  → Elastic Block Store (EBS) (IaaS), Azure Disk Storage (IaaS), Persistent Disks (IaaS)

- **File storage:** Shared file system
  → Elastic File System (EFS) (PaaS), Azure Files (PaaS), Filestore (PaaS)



Block storage
Data stored in fixed-size 'blocks' in a rigid arrangement—ideal for enterprise databases



File storage
Data stored as 'files' in hierarchically nested 'folders'—ideal for active documents

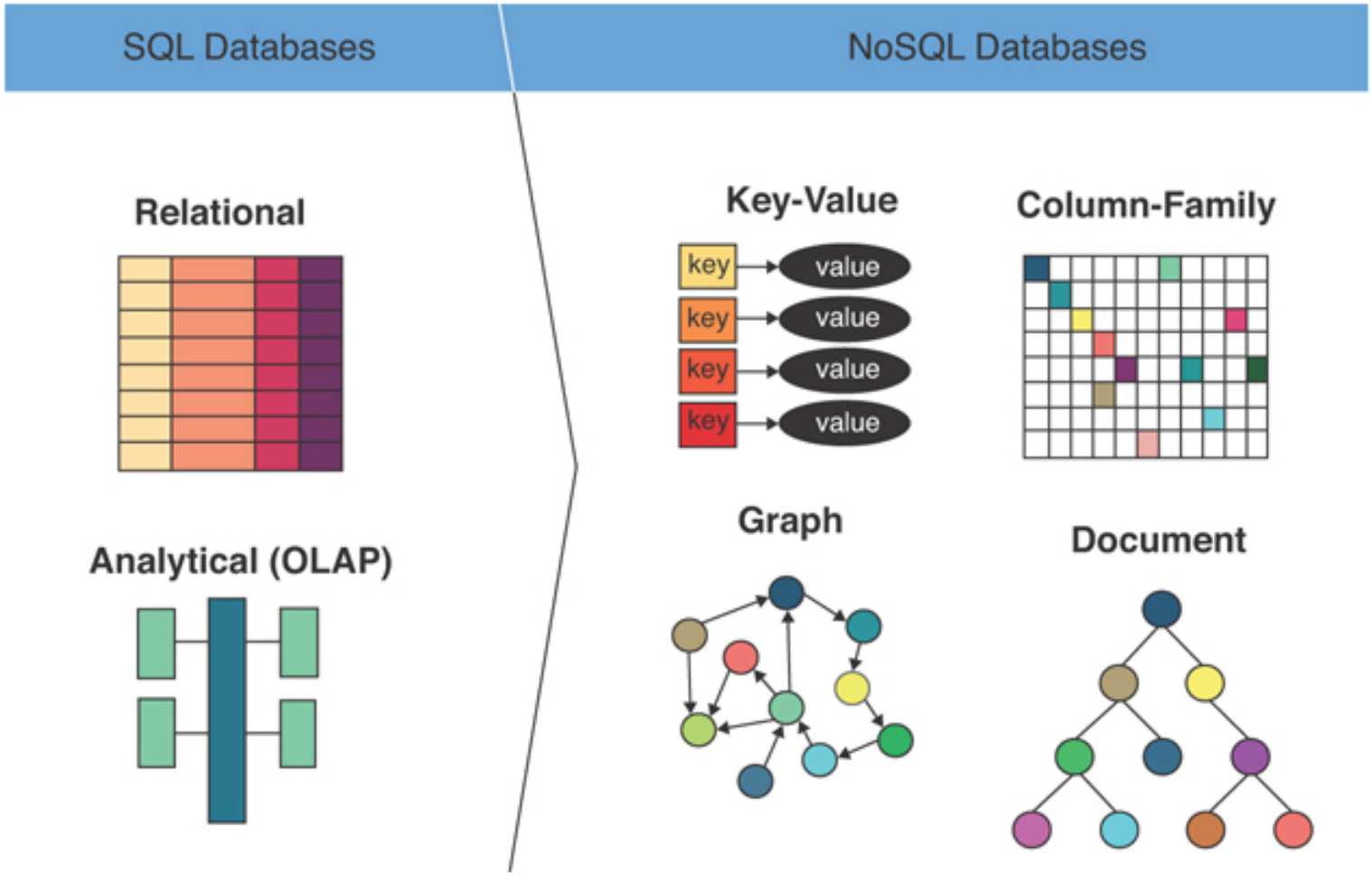
Object storage
Data stored as 'objects' in scalable 'buckets'—ideal for unstructured big data, analytics and archiving

3. Database services

Managed databases eliminate the need to maintain database servers

- **Relational (SQL):**
  → Relational Database Service (RDS) (PaaS), Azure SQL Database (PaaS), Cloud SQL (PaaS)

- **NoSQL:**
  → DynamoDB (PaaS), Cosmos DB (PaaS), Firestore (PaaS), MogoDB Atlas (SaaS/PaaS)


SQL Databases
NoSQL Databases
Relational
Analytical (OLAP)
Key-Value
key
value
key
value
key
value
key
value
Column-Family
Graph
Document

4. AI/ML services

Ready-made or customizable tools for building, training, and deploying ML models

- **ML platforms:**
  → SageMaker (PaaS), Azure Machine Learning (PaaS), Vertex AI (PaaS), Inference Endpoints (Provider is Hugging Face; PaaS/SaaS)

- **AutoML:** Train models without writing code
  → SageMaker Autopilot (PaaS), Azure AutoML (PaaS), AutoML (PaaS)

- **AI API's:** Pre-trained models for computer vision, NLP, etc.
  → Comprehend (SaaS/PaaS), Cognitive Services (SaaS), Cloud Vision, NLP API (SaaS), Claude (Provider Anthropic; SaaS)

5. Analytics/Big Data services

Services that **analyze large datasets** quickly and efficiently

- **Data Warehouses:** Columnar SQL
  → Redshift (PaaS), Synapse Analytics (PaaS), BigQuery (PaaS)

- **ETL tools:** Moving and transforming data
  → Glue (PaaS), Dat Factory (PaaS), Cloud Data Fusion (PaaS)

- **BI tools:** Dashboarding and visual analytics.
  → QuickSight (SaaS), Power BI (SaaS), Looker (SaaS)

6. DevOps/CI-CD

Tools to **automate deployment, testing, and versioning**

- **CI/CD Pipelines:** Tools that automate the process of building, testing, and deploying code
  → CodePipeline (PaaS), Azure Pipelines (PaaS), Cloud Build (PaaS)

- **Container registries:** Services for storing and managing Docker container images
  → Elastic Container Registry (ECR) (PaaS), Azure Container Registry (ACR) (PaaS), Artifact Registry (PaaS), Docker Hub (Provider Docker; SaaS)

7. Networking and security

Ensure **connectivity, access control, and protection of resources**

- **Virtual Private Clouds (VPCs):** Isolated networks in the cloud to organize and secure resources
  → Amazon VPC (IaaS), Azure Virtual Network (VNET) (IaaS), Virtual Private Cloud (VPC) (IaaS)

- **IAM (Identity and Access Management):** Define who can access what, and under what conditions
  → AWS IAM (IaaS/PaaS), Azure Active Directory (AAD) (SaaS/PaaS), IAM (Cloud Identity) (IaaS/PaaS)

- **Encryption and Compliance:** Services to encrypt data at rest and in transit.
  → AWS Key Management Service (KMS) (PaaS), Azure Key Vault (PaaS), Cloud Key Management Service (KMS) (PaaS)

Part 3:

Amazon Web Services

3.1 Amazon Web Services Free Tier

- **AWS Free Tier** is Amazon's Web Service's introductory offering that lets you use many of its cloud services *for free*, within specified usage limits.
- It is specifically designed to help you explore and learn AWS without incurring charges.
- **Question:** What types of free offers are available in AWS Free Tier?

- **Answer:**
  - **Always free:** Available to all AWS users indefinitely, not just new accounts

Examples:

- **AWS Lambda:** 1 million requests/month (see Section 4.6)
- **Amazon DynamoDB:** 5 GB Storage

- **12-month Free Tier:** For new AWS accounts, valid for the first 12 months after signup

  Examples:

  - **Amazon S3:** 5 GB of standard storage
  - **Amazon EC2:** 750 hours/month of t2.micro or t3.micro instance
  - **Amazon RDS:** 750 hours/month of db.t2.micro (MySQL, PostgreSQL, etc.)

- **Short-term trials:** Some services offer free trials for a limited period once activated

  Examples:

  - Free trials for some AI services

- More details can be found here: https://aws.amazon.com/free/

- The free tier is an excellent way to get familiar with AWS.

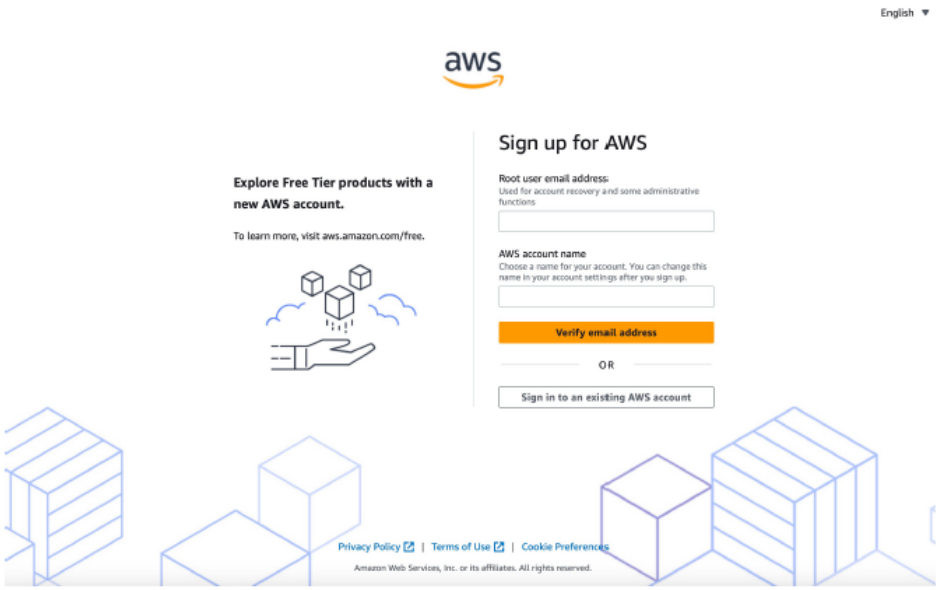
- **Important notes:**
  - To avoid of being charged, you must stay within usage limits
  - **Question:** How can you track this?
  - **Solution:** Use the **AWS Billing** Dashboard and set up **billing alerts** (see Section 3.4)
    - The **billing alert** is a notification that AWS sends (via mail or SMS) *when your usage costs exceed a specified amount*
    - You must enable billing alerts manually (*they are not on by default*)
    - It does not stop or suspend your usage, it only warns you!
  - Free tier resets monthly.

3.2 Create an AWS account

To use Amazon Free Tier, we first need to **create an AWS account**.

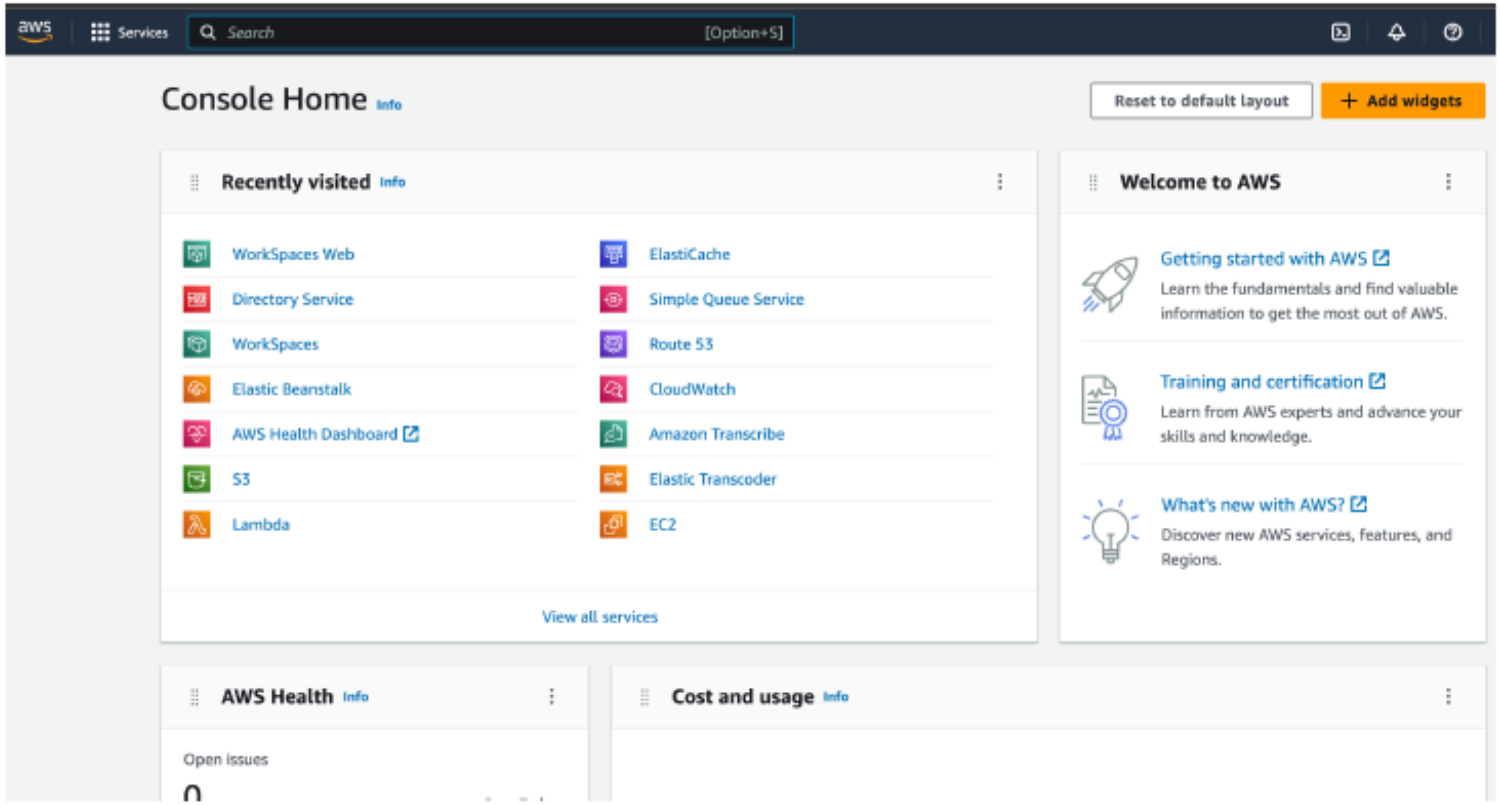
Steps:

- Go to aws.amazon.com and create a new account



→ Direct link: https://signin.aws.amazon.com/signup?request_type=register

- After setting up your account, you get access to the console:



3.3 Security setup (IAM)

1. Secure your root account with MFA

After signing in to the AWS console for the first time, your first priority is to secure your root account with MFA (Multi-Factor Authentication).

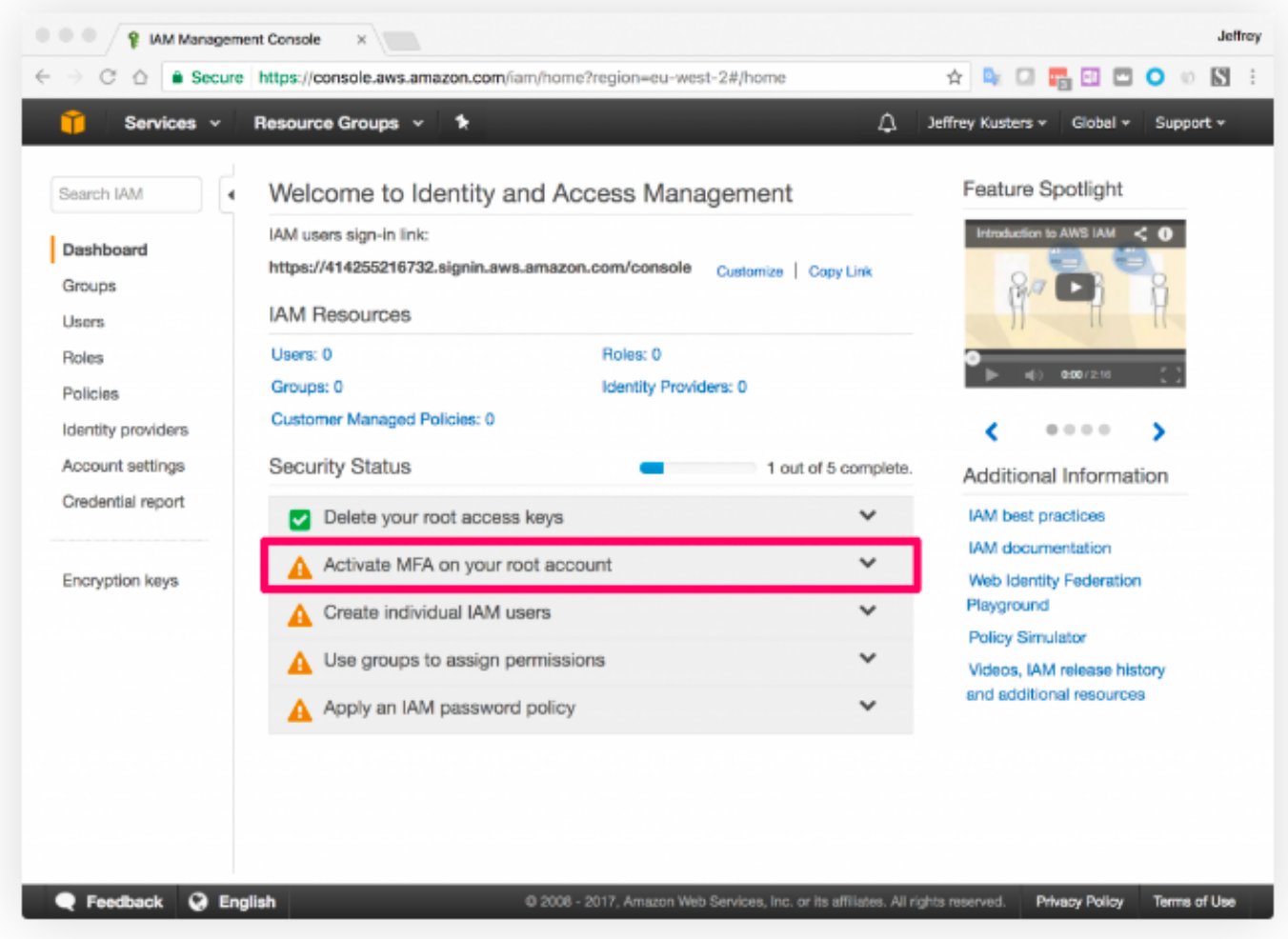
Question: Why is this needed?

Answer: Your creditcard is linked to your AWS root account. So, secure it!

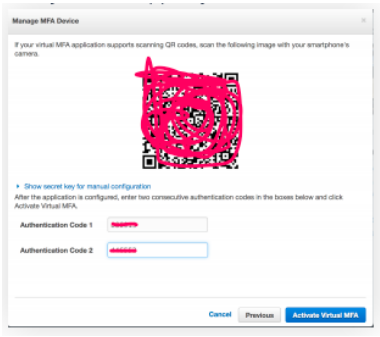
Steps:

- In the **AWS Console**, go to the **IAM (Identity and Access Management) service**.
  - **IAM service** = An authentication and authorization service that enables you to:
    - Decide who or what can access the AWS services in your account (authentication)
    - What these entities are permitted to do in your account (authorization)

- You will see a 5-step wizard with step 2 being **'Activate MFA on your root account'**:



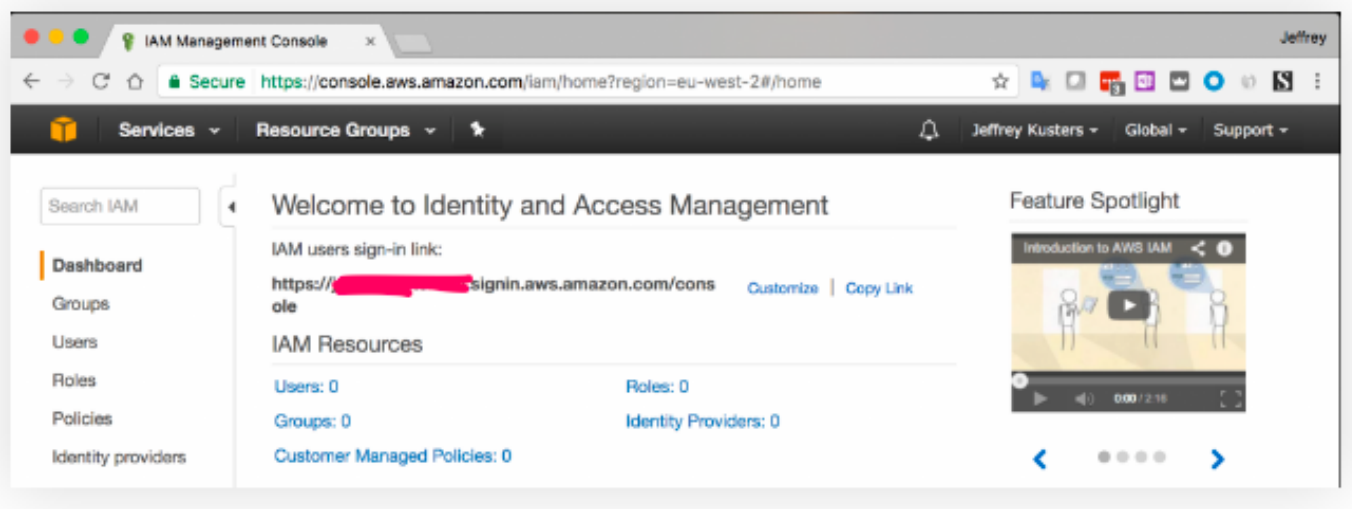
- Expand step 2 and click **Manage MFA**
- Select **virtual MFA device**
- You will get a popup message about downloading a compatible MFA app for your smartphone, e.g., Google Authenticator
- Link your MFA app to your AWS root account by entering two consecutive codes:



- Refresh your browser and you will get a checkmark at step number 2

2. Create a user-friendly IAM sign-in link

In the top of the IAM screen, you will notice a sign-in link starting with a number. This is your **AWS account number**. You can create a user-friendly sign-in link by clicking **Customize**. This will create a new DNS namespace so it has to be a unique name:



Remark: The **sign-in link** is a special URL that your IAM users can browse to access your account.

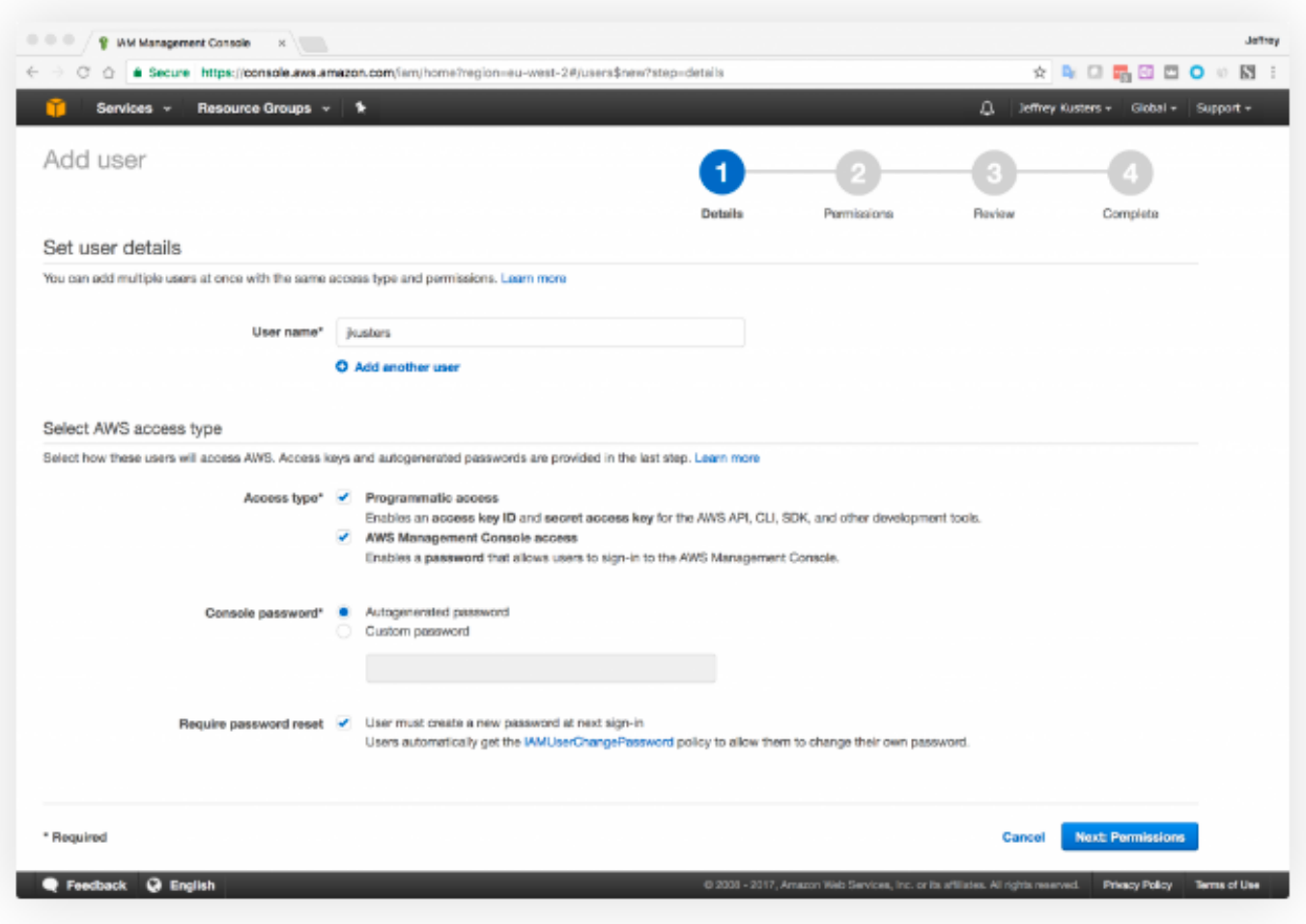
3. Create a new IAM user

As a security best practice, you should limit the use of your AWS root account.

In step 3 of the 5 step security wizard, you can **create additional user accounts (IAM users)** to perform daily tasks.

Steps:

- Expand step 3, click **'Manage users'** and select **'Add user'**.
- You can choose to allow **'Programatic access'** and/or access via the **'AWS Management Console'**.

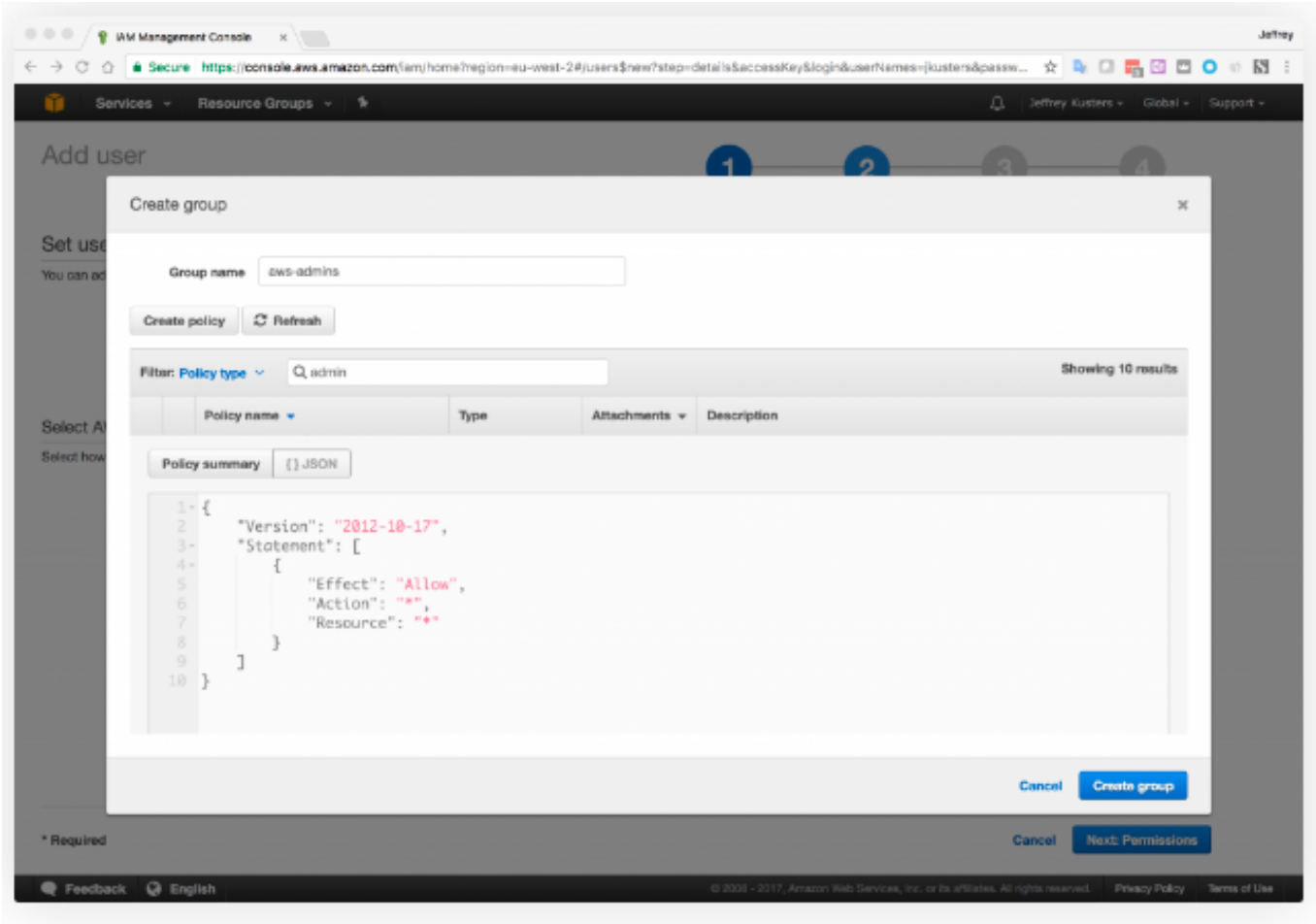

IAM Management Console
Jeffrey
Secure https://console.aws.amazon.com/iam/home?region=eu-west-2#/users$new?step=details
Services
Resource Groups
Jeffrey Kusters
Global
Support
Add user
1
Details
2
Permissions
3
Review
4
Complete
Set user details
You can add multiple users at once with the same access type and permissions. Learn more
User name*
jkusters
Add another user
Select AWS access type
Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more
Access type*
Programmatic access
Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.
AWS Management Console access
Enables a password that allows users to sign-in to the AWS Management Console.
Console password*
Autogenerated password
Custom password
Require password reset
User must create a new password at next sign-in
Users automatically get the IAMUserChangePassword policy to allow them to change their own password.
* Required
Cancel
Next: Permissions
Feedback
English
© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

4. Create a group and assign permissions

The next step is to **assign permissions**. This is done by placing the new IAM user in a group.

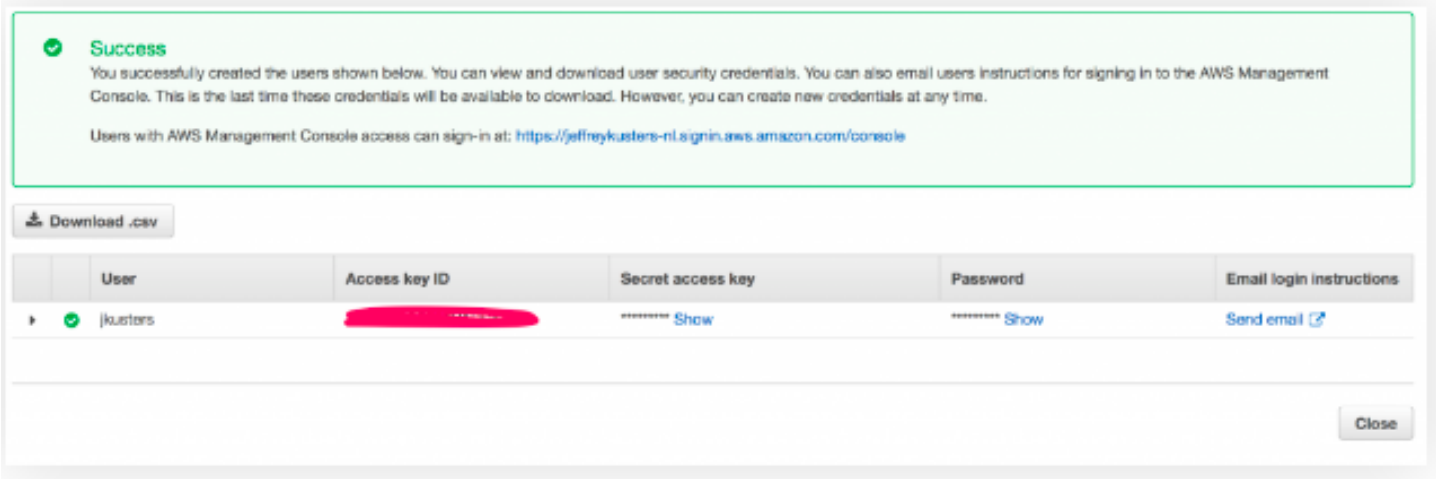
Steps:

- Create a group called **'aws-admins'**
- To set full access admin permissions for this group, assign the **'Administrator Access'** policy. You can expand each policy and, either view a plain English summary or look directly at the JSON code


IAM Management Console
Jeffrey
Secure https://console.aws.amazon.com/iam/home?region=eu-west-2#/users$new?step=details&accessKey&login&userNames=jkusters&passw...
Services
Resource Groups
Jeffrey Kusters
Global
Support
Add user
Create group
Group name
aws-admins
Create policy
Refresh
Filter: Policy type
admin
Showing 10 results
Policy name
Type
Attachments
Description
Policy summary
{} JSON
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "*",
"Resource": "*"
}
]
}
Cancel
Create group
* Required
Cancel
Next: Permissions
Feedback
English
Privacy Policy
Terms of Use

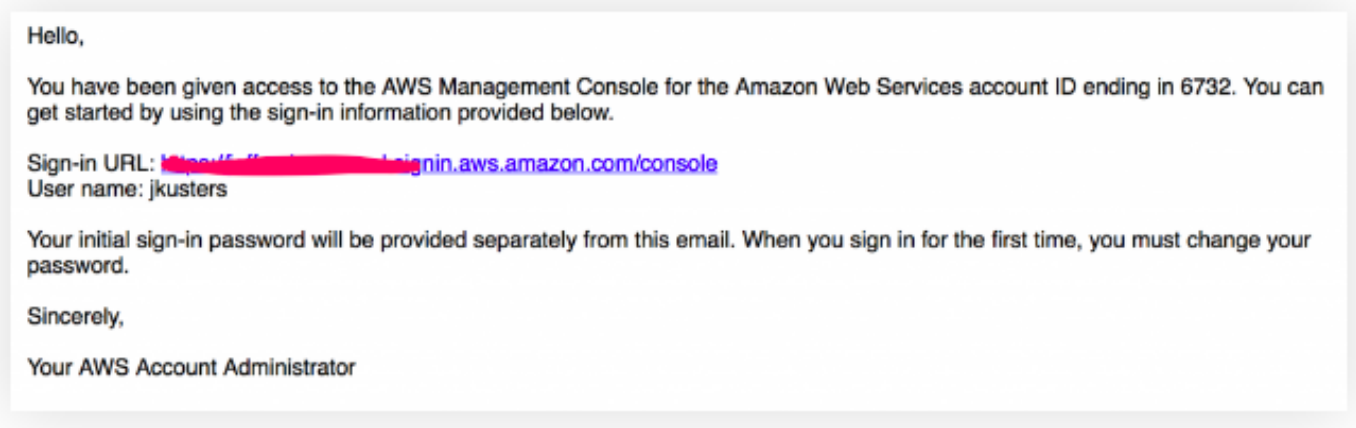
- Assign the policy to the group and proceed.

- You can review a summary and click on create user.



Suggestion: Download the CSV file with the security information and store it somewhere in a safe place.

- For here onwards, different things can be done:
  - Use the **'Access key ID'** together with the **'Secret access key'** to programmatically access your AWS account.
  - Use the username and password combination to login to the **AWS Management console**
  - Choose to email instructions to a specified email address of the newly created user



Hello,

You have been given access to the AWS Management Console for the Amazon Web Services account ID ending in 6732. You can get started by using the sign-in information provided below.

Sign-in URL: [redacted]signin.aws.amazon.com/console
User name: jkusters

Your initial sign-in password will be provided separately from this email. When you sign in for the first time, you must change your password.

Sincerely,

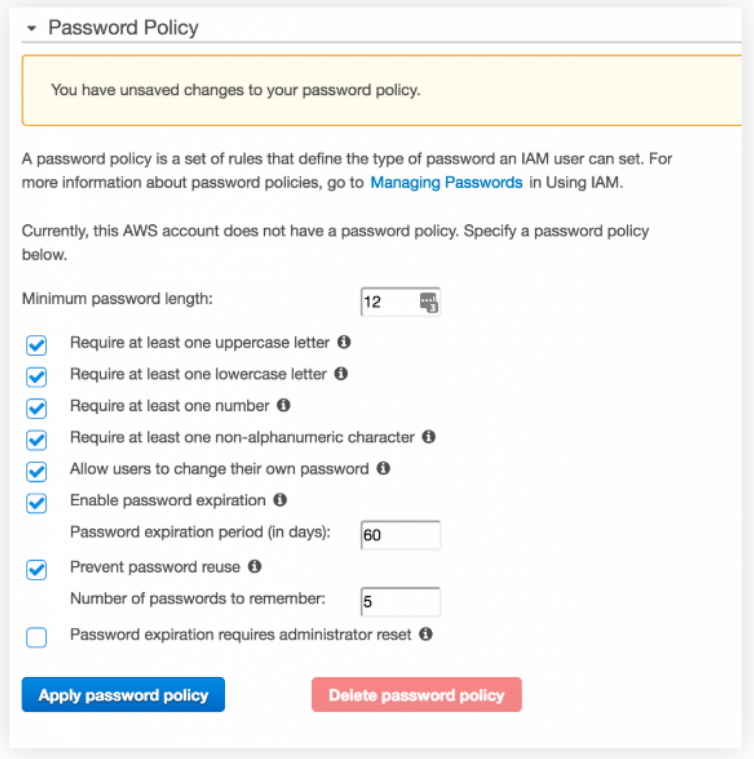
Your AWS Account Administrator

- This ticks off steps 3 and 4 of the security wizard!

5. Create a password policy

A last step in the **security setup** (step 5) is to create a password policy.

Step: Click **Manage password policy** and specify a policy that meets your requirements:



3.4 Setting up billing alarms

A next step is to set up your **billing alerts**.

You can monitor your estimated AWS charges by using **Amazon CloudWatch**. When you enable the monitoring of estimated charges for your AWS account, the estimated charges are calculated and sent several times daily to **CloudWatch** as metric data.

1. Enable billing alerts

- Access the **Billing and Cost Management console** at:

  https://console.aws.amazon.com/costmanagement

- In the navigation pane, choose **Billing Preferences**.
- By **Alert preferences** choose **Edit**.
- Choose **Receive CloudWatch Billing Alerts**.
- Choose **Save preferences**.

2. Create a billing alarm using the CloudWatch console

Important: Before you create a billing alarm, you must set your **Region** to **US East (N. Virginia)**. Billing metric data is stored in this Region and represents worldwide charges.

- Access the **CloudWatch console** at:

  https://console.aws.amazon.com/cloudwatch

- In the navigation pane, choose **Alarms**, and then choose **All alarms**.

- Choose **Create alarm**.

- Choose **Select metric**. In **AWS Namespaces**, choose **Billing**, and then choose **Total Estimated Charge**.

- Select the box for the **EstimatedCharges metric**, and then choose **Select metric**.
- For **Statistic**, choose **Maximum**.
- For **Period**, choose **6 hours**.
- For **Threshold type**, choose **Static**.
- For **Whenever EstimatedCharges is ...**, choose **Greater**.
- For **than ...**, define the value that you want to cause your alarm to trigger. For example, 1 USD.

- Choose **Additional Configuration** and do the following:
  - For **Datapoints to alarm**, specify **1 out of 1**.
  - For **Missing data treatment**, choose **Treat missing data as missing**.
- Choose **Next**.
- Under **Notification**, ensure that **In alarm** is selected. Then specify an Amazon SNS topic to be notified when your alarm is in the ALARM state. The Amazon SNS topic can include your email address so that you receive email when the billing amount crosses the threshold that you specified.You can select an existing Amazon SNS topic, create a new Amazon SNS topic, or use a topic ARN to notify other account. If you want your alarm to send multiple notifications for the same alarm state or for different alarm states, choose **Add notification**.

- Choose **Next**.
- Under **Name and description**, enter a name for your alarm. The name must contain only UTF-8 characters, and can't contain ASCII control characters.
- Choose **Next**.
- Under **Preview and create**, make sure that your configuration is correct, and then choose **Create alarm**.

3. Deleting a billing alarm

- Access the **CloudWatch console** at:

  https://console.aws.amazon.com/cloudwatch

- If necessary, change the **Region** to **US East (N. Virginia)**. Billing metric data is stored in this Region and reflects worldwide charges.

- In the navigation pane, choose **Alarms, All alarms**.

- Select the check box next to the alarm and choose **Actions, Delete**.

- When prompted for confirmation, choose **Yes, Delete**.

3.5 Accessibility of AWS Free Tier

Until now, by setting up your AWS account and settings, the **management console of AWS** was used, accessible via the web browser.

In practice, after setting up your account, there exist different ways to access the AWS services:

- **Management console** (most easy and used one)
- **Command-line interface (CLI)**
- **Software Development Kit (SDK)** (for programmers)

In what follows, we will explain them in detail.

1. Management console

- The **AWS Management console** is the **web-based graphical user interface (GUI)** for managing AWS services.

- Best for beginners and manual setup.

- **Examples:**
  - Uploading files to S3 via drag-and-drop
    → www.youtube.com/watch?v=GzgAS5FnHH4
  - Clicking through SageMaker to start a Jupyter notebook
    → www.youtube.com/watch?v=J5I7P593beg
  - Creating a Lambda function via wizard
    → www.youtube.com/watch?v=5hF2M6GLxFQ

2. Command-line interface (CLI)

- The **AWS CLI** is a **text-based tool** for managing AWS services via terminal/command prompt.
- You type commands like `aws s3 cp` or `aws ec2 run-instances`
- Fast for batch tasks
- **Example:**

```
aws s3 cp myfile.csv s3://my-bucket/
```

→ www.youtube.com/watch?v=FLIp6BLtwjk

- **To use the AWS CLI**, you first need to install it on your local machine, and *depends in your operating system*.
  - **Windows:**

    ```
    msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi
    ```

  - **Mac:**

    ```
    curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o
    "AWSCLIV2.pkg"
    sudo installer -pkg AWSCLIV2.pkg -target /
    ```

- Once installed, **configure the AWS CLI** with your AWS credentials, allowing you to access your AWS resources and services.

```
aws configure
AWS Access Key ID [None]:  SKDELSOEJALZEK2
AWS Secret Access Key [None]:  wJqlrXUtndlncjJZdkjn
Default region name [None]:  us-west-2
Default output format [None]:  json
```

- **To create a S3 Bucket** with CLI:

```
aws s3api create-bucket \
    --bucket mdaclassdemo \
    --region eu-central-1 \
    --create-bucket-configuration
LocationConstraint=eu-central-1
```

  - **Remark:** A **S3 Bucket** is a **storage container** in **Amazon S3**, where you can store files, data, or entire datasets in the cloud (**objects**), similar to a folder or drive on your computer.

3. Software Development Kits (SDKs)

- **SDKs** are **programming language libraries** for interacting with AWS services directly in your code (Python, R, Java, etc.).
- Best for automation with applications and custom workflows in data science
- Popular among data scientists, since it is *fully programmable* and *integratable into code*
  - `Boto3` allows you to write Python code that interacts with AWS services

- Prior to this, setup is needed (IAM roles, AWS credentials configured)
- **Example (boto3 in Python):**

```
import boto3
s3 = boto3.client('s3',
     AWS_ACCESS_KEY_ID = 'YOUR_ACCESS_KEY_ID',
     AWS_SECRET_ACCESS_KEY = 'YOUR_SECRET_ACCESS_KEY',
     REGION_NAME='eu-central-1',
)
```

```
s3.creat_bucket(
    Bucket=my-bucket,
    CreateBucketConfiguration={'LocationConstraint' :
'eu-central-1'}
)
s3.upload_file('train.csv, 'my-bucket', 'train.csv'
```

→ www.youtube.com/watch?v=QSDIKakB8qs

Remarks:

- Never hard-code your credentials in Python scripts that are shared or production code.

  Instead, consider the use of **environment variables**, e.g.,

  Terminal:

```
export AWS_ACCESS_KEY_ID = 'YOUR_ACCESS_KEY_ID'
export AWS_SECRET_ACCESS_KEY = 'YOUR_SECRET_ACCESS_KEY'
AWS_DEFAULT_REGION = 'eu-central-1'
```

  Python:

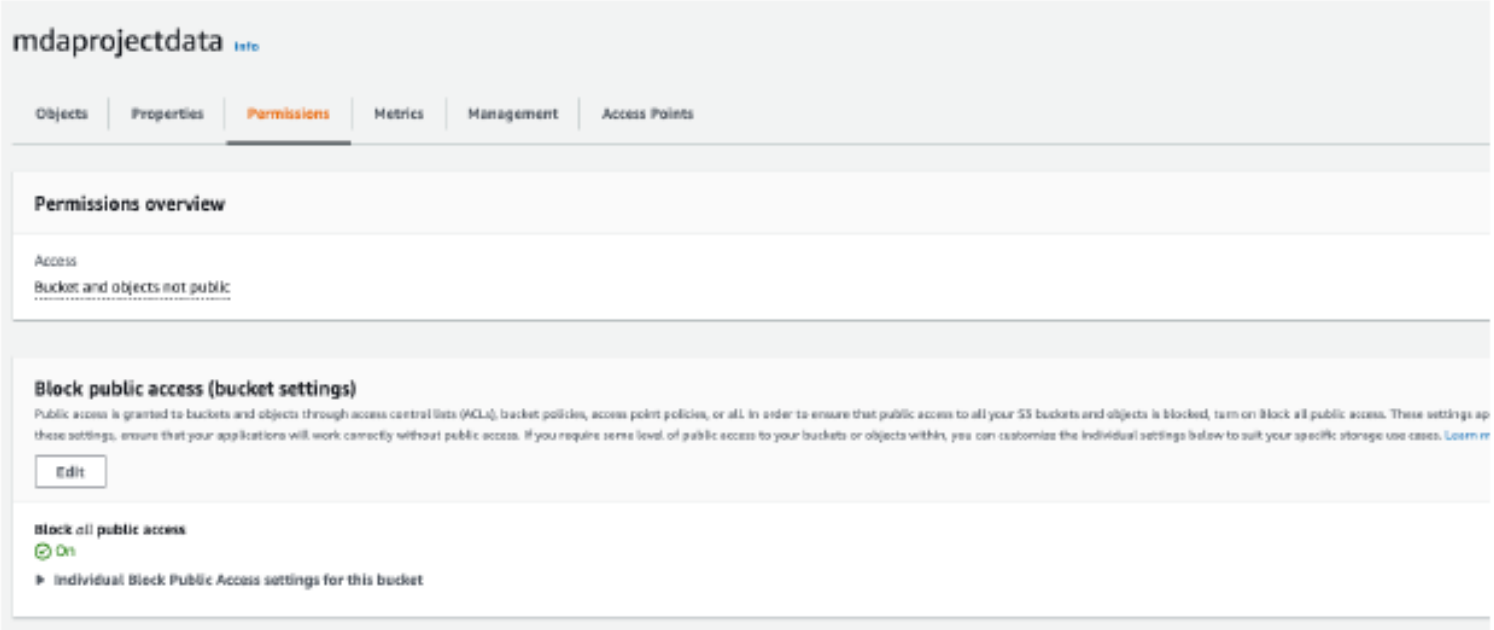
```
import boto3
s3 = boto3.client('s3')
```

- By default, S3 buckets and its objects are NOT publicly available.

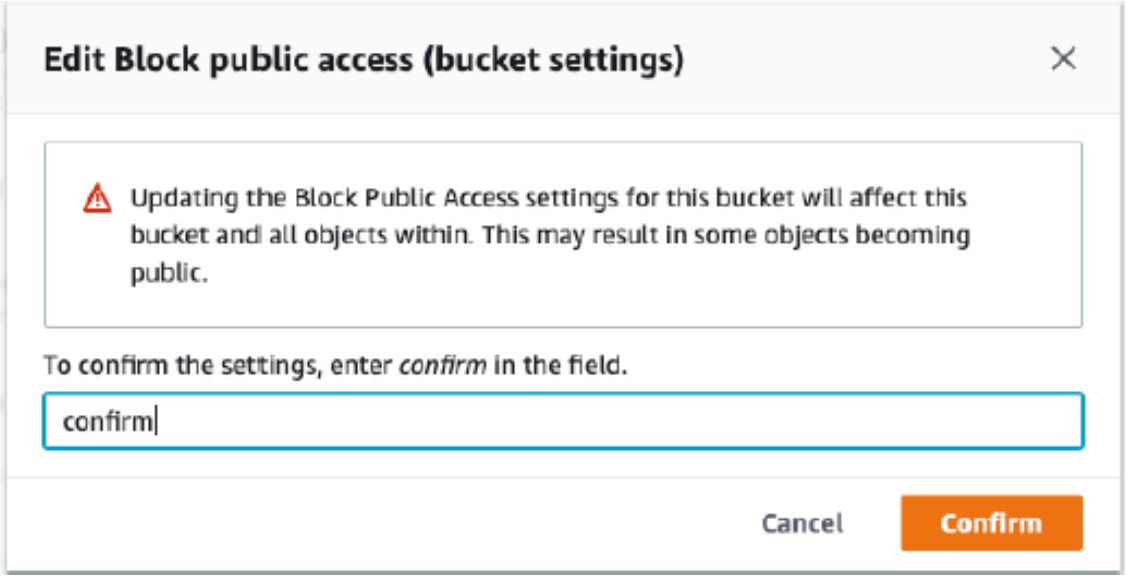
  Question: How to grant access and make it publicly available?

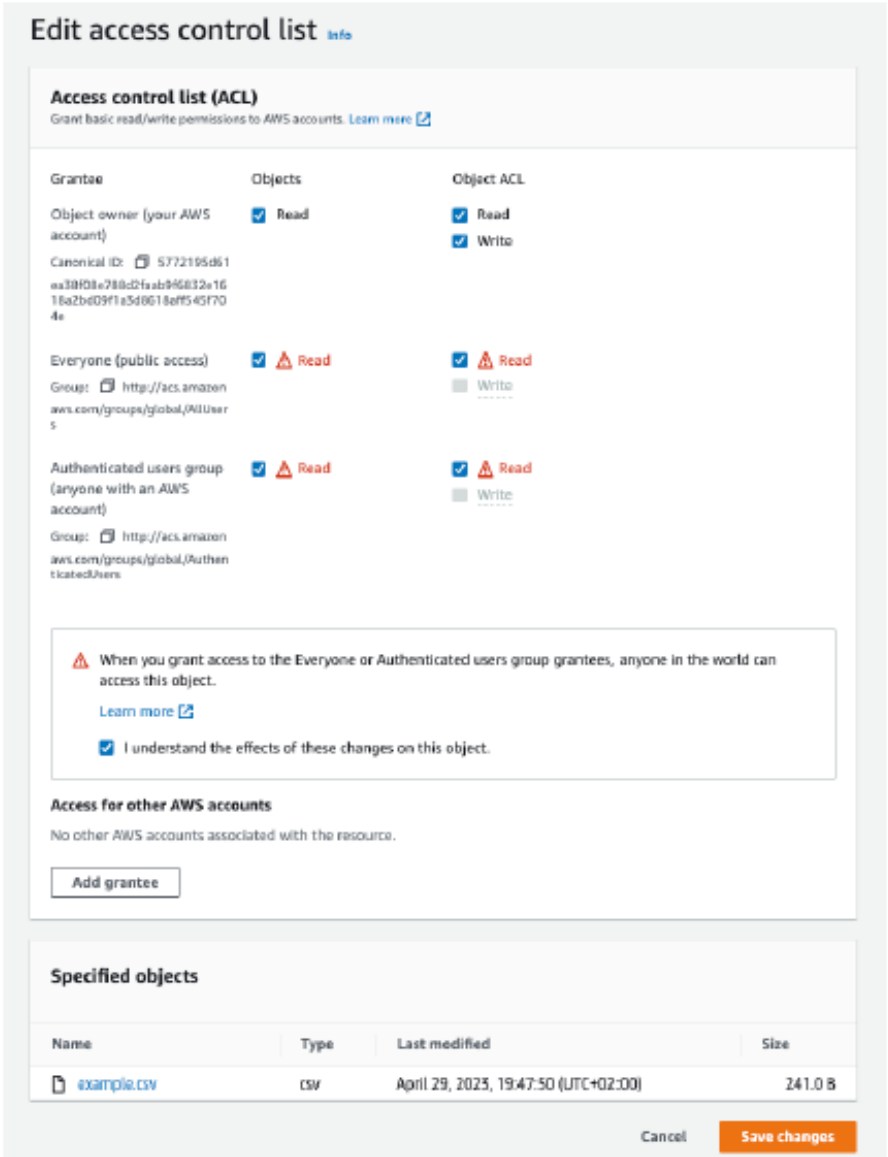
  Answer:

  - If block public access is activated for all buckets within the account, the message "Bucket and objects not public" is shown in the management console.



- By clicking on **'Edit'**, you can edit the bucket and objects settings




Edit access control list Info
Access control list (ACL)
Grant basic read/write permissions to AWS accounts. Learn more
Grantee
Objects
Object ACL
Object owner (your AWS account)
Canonical ID: 5722195d61ea38f08e788d2faab9f6832e1618a2bd09f1a5d8618eff545f704e
Read
Read
Write
Everyone (public access)
Group: http://acs.amazonaws.com/groups/global/AllUsers
Read
Read
Write
Authenticated users group (anyone with an AWS account)
Group: http://acs.amazonaws.com/groups/global/AuthenticatedUsers
Read
Read
Write
When you grant access to the Everyone or Authenticated users group grantees, anyone in the world can access this object.
Learn more
I understand the effects of these changes on this object.
Access for other AWS accounts
No other AWS accounts associated with the resource.
Add grantee
Specified objects
Name
Type
Last modified
Size
example.csv
csv
April 29, 2023, 19:47:50 (UTC+02:00)
241.0 B
Cancel
Save changes

- After AWS configuration and boto3 installation (`pip install boto3`), S3 objects can directly be accessed via Pandas:

  Route 1:

```
import pandas as pd
pd.read_csv('https://mdaprojectdata.s3.eu-central-1.
     amazonaws.com/example.csv')
```

  Route 2:

```
pd.read_csv('s3://mdaprojectdata/example.csv')
```

Part 4:

End-to-end data science project with AWS

4.1 Objective of the project

Use Python and scikit-learn to train a logistic model predicting Titanic survivors, *using AWS Free Tier services* to handle

- **storage** → S3,
- **compute** → SageMaker Studio or EC2,
- **deployment** → Lambda + API Gateway.

We will examine how to perform these steps with **management console**, **CLI** and **SDKs**!

4.2 Create an S3 bucket

1. Management console:

- Go to the **S3 service**
- Click **"Create bucket"**.
  - Bucket name: titanic-data-console
  - Region: us-east-1
- Click **"Create bucket"** at the bottom.

2. CLI:

```
aws s3api creat-bucket \
    --bucket titanic-data-cli \
    --region eu-east-1 \
    --create-bucket-configuration LocationConstraint=us-east-1
```

3. SDKs:

```
import boto3

s3 = boto3.client('s3', region_name='us-east-1')

bucket_name = 'titanic-data-bucket-sdk'

s3.create_bucket(
    Bucket=bucket_name,
    CreateBucketConfiguration={'LocationConstraint':'us-east-1'}
)
```

4.3 Upload the Titanic Dataset

1. Management console:

- Go into your new bucket: titanic-data-console.
- Click **"Upload"** → **Add files**:
  - Upload train.csv and test.csv from **Kaggle** (kaggle.com/competitions/titanic/data)
- Leave permissions as default → Click **"Upload"**.

2. CLI:

```
aws s3 cp train.csv s3://titanic-data-cli/train.csv
```

3. SDKs:

```
s3.upload_file('train.csv', bucket_name, 'train.csv')
```

4.4 Set up SageMaker notebook (JupyterLab)

1. Management console:

- Go to the **SageMaker service**.
- Select **SageMaker Studio** (if this is the first time, you will need to create a user and domain).
  - Accept defaults or create a new IAM role.
  - Launch **Studio** (JupyterLab interface opens in-browser).
- Inside **JupyterLab**:
  - Click **"File"** $\rightarrow$ **"New"** $\rightarrow$ **"Notebook (Python 3)"**.

2. CLI:

While the CLI is limited compared to the Console for managing SageMaker Studio, you can create a Notebook Instance:

- Create an IAM role with **SageMaker permissions**
- Create a notebook instance:

```
aws sagemaker create-notebook-instance \
    --notebook-instance-name TitanicNotebookCLI \
    --instance-type ml.t2.medium \
    --role-arn
arn:aws:iam::123456789012:role/SageMakerExecutionRole
    --region us-east-1
```

- Starting the notebook instance:

```
aws sagemaker start-notebook-instance \
    --notebook-instance-name TitanicNotebookCLI
```

- Then open the **Jupyter environment from the AWS Console** $\rightarrow$ **SageMaker** $\rightarrow$ **Notebook Instances**.

3. SDKs:

Creating a full SageMaker Studio instance is more complex via SDK (and usually done via Console), but we can automate traditional notebook instances:

```
sagemaker = boto3.client('sagemaker')

sagemaker.create_notebook_instance(
    NotebookInstanceName='TitanicNotebookSDK',
    InstanceType='ml.t2.medium',
    RoleArn='your-iam-role-arn',
    SubnetId='your-subnet-id',
    SecurityGroupIds=['your-security-group-id']
)
```

4.4 Train model in Notebook

1. Management console:

In the **new notebook**, run:

```
import pandas as pd

df = pd.read_csv('s3://titanic-data-console/train.csv')

df['Age'].fillna(df['Age'].mean(), inplace=True)

df = pd.get_dummies(df.drop(columns=['Name', 'Cabin', 'Ticket',
'PassengerId']), drop_first=True)
```

```
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

X = df.drop('Survived', axis=1)

y = df['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
model = LogisticRegression(max_iter=500)

model.fit(X_train, y_train)

print("Accuracy:", accuracy_score(y_test, model.predict(X_test)))
```

2. CLI & 3. SDKs:

Use the same python script inside the Jupyter notebook you just launched in both cases.

4.5 Save and upload the model

1. Management console:

- In a **new notebook cell**:

```
import joblib

joblib.dump(model, 'model.pkl')
```

Remark: `joblib` is a Python library used to **serialize (save, e.g., `model.pkl`)** and **deserialize (load)** Python objects (here, our training ML model)

  - **Serialization**: Converting a Python object (like a model) into a binary format that can be saved and uploaded to the cloud

  - **Deserialization**: Reading that binary file and reconstructing the original model object in memory.

- Then in the **JupyterLab file browser**:
  - Right-click model.pkl $\rightarrow$ **Download** (or **note location**).
  - Go back to S3 $\rightarrow$ Bucket: titanic-data-console
  - Click **"Upload"** $\rightarrow$ Add model.pkl.

2. CLI:

After saving model.pkl in your notebook:

```
aws s3 cp model.pkl s3://titanic-data-cli/model.pkl
```

3. SDKs:

```
import joblib

joblib.dump(model, 'model.pkl')

s3.upload_file('model.pkl', bucket_name, 'model.pkl')
```

4.6 Deploy your model

The next task is to

- Create a **[1] Lambda function** that loads the model from S3
- Trigger it via HTTP request using **[2] API Gateway**
- Input: JSON format (data points); Output: Prediction (`survived:yes/no`)

But before reporting all (coding) steps, an explanation will be given about the principles and usage of **[1] AWS Lambda** and **[2] API Gateway**!

[1] AWS Lambda:

- Introduced in 2014 by Amazon, and extended its functionalities over time
- **AWS Lambda** lets you run code (e.g., Python, Java, Node.js) without provisioning servers. You upload your function code, and AWS runs it in response to events.
- At its core, **Lambda functions** consists of two parts:
  - The **function** to call
  - **Additional layers** that can be used to add other functionalities, s.a. additional Python packages

- There are three cost drivers for Lambda functions:
  - Number of executions per month
  - Memory allocated for a function
  - Execution time in miliseconds
    → https://s3.amazonaws.com/lambda-tools/pricing-calculator.html
- **Reminder:** AWS Free tier offers 1 million requests/month for free, and 400000 **GB-seconds** of compute time per month!
  - **GB-seconds** = number of seconds your function runs for, multiplied by the amount of RAM memory consumed.

- The file uploaded in AWS needs to have a **"handler" function**
- It is the entry point AWS Lambda uses to execute your code.
- When you configure a Lambda function, you specify the handler name, typically in the format:

  `filename.function_name`

  Example in CLI:

  `--handler lambda_file.lambda_handler`

  - `lambda_file.py` is the file that contains your code
  - `lambda_handler` is the function within that file to be called when the Lambda is triggered

- AWS looks inside that file and executes the specified function when the event occurs

- This function takes two arguments:

```
def lambda_handler(event, context):
    ...
```

  - **event**: contains the input data (e.g., JSON body of an HTTP request)
  - **context**: contains metadata about the invocation, function name, timeout, etc.

- The handler must return a response that AWS Lambda understands.

1. Management console:

- Go to the **AWS Lambda** $\rightarrow$ Click **Create Function**
- Choos **"Author from scratch"**
  - Name: `predictionTitanic`
  - Runtime: Python 3.x
  - Permissions: Create a new role with basic Lambda permissions
- Click **Create Function**

- In the inline editor, **upload or past your handler code**:

```
import json
import boto3
import joblib
import pandas as pd
import s3fs

s3=boto3.client('s3')
bucket='titanic-data-bucket'
key='model.pkl'
model_file='/tmp/model.pkl'
s3.download_file(bucket, key, model_file)
model=joblib.load(model_file)
```

```
def lambda_handler(event,context):
    input_data=pd.DataFrame([event['features']])
    prediction=model.predict(input_data)[0]
    return {
        'statusCode':200,
        'body':json.dumps({'prediction':int(prediction)})
    }
```

- Click **Deploy**

2. CLI:

- Package your Lambda code in a ZIP file:

```
zip function.zip lambda_file.py
```

- Create the Lambda function:

```
aws lambda create-function \
    --function-name predictTitanic \
    --runtime python3.9 \
    --handler lambda_file.lambda_handler \
    --role arn:aws:iam::ACCOUNT_ID:role/LambdaExecutionRole \
    --zip-file fileb://function.zip
```

3. SDKs:

```
import boto3

client=boto3.client('lambda')

with open('function.zip', 'rb') as f:

    zipped_code=f.read()
```

```
client.create_function(
    FunctionName='PredictTitanic',
    RunTime='python3.9',
    Role='arn:aws:iam::ACCOUNT_ID:role/LambdaExecturionRole',
    Code=dict(ZipFile=zipped_code),
    Timeout=300
)
```

Remark: Lambda requires permission to read from S3. Therefore, attach the following policy:

```
{

    "Effect":"Allow",

    "Action":["s3:GetObject"],

    "Resource":"arn:aws:s3:::titanic-data-bucket",

}
```

[2] API Getaway:

- **API Getaway** is like a "front door" for your model. It lets anyone (a browser, Postman or your app) call your deployed model using a simple HTTP request.
- In our Titanic project, for example, we are building a ML model that predicts if someone would survive the Titanic.
- You want users to (1) send passenger data and (2) get back a prediction.

- We don't want users to interact directly with Lambda or S3, but instead, just want an API like:

  ```
  POST https://api.com/predict

       ...
  ```

  and receives the result

- **Question:** How does it work?

- **Answer:** Let's say you want to send in your data in JSON format:

```
{
    "features":[22, 1, 7.25, 1, 0, 0, 1, 0, 0]
}
```

API Getaway does this:

- Receives the HTTP `POST` request on `/predict`
- Forward the request to your Lambda function
- Lambda runs the model on the data and returns:

```
{ "prediction":  0 }
```

- API Gateway sends the response back to the user.

Advantages:

- **User-friendly interface** for your model
- **Security:** You can require API keys or tokens
- **Scalability:** It can handle thousands of requests
- **Monitoring:** View metric like request count or latency

1. Management console:

- Go to the **API Getaway** $\rightarrow$ Create **REST API**
- Choose **"REST API"** for detailed setup
- Name: `titanicAPI`
- Create a resource `/predict` and a `POST` method
- Link it to the Lambda function
- Deploy the API to a new stage (e.g., `prod`)

- After deployment, **API Getaway** generates a public endpoint that looks like this:

```
curl -X POST
https://<api-id>.execute-api.<region>.amazonaws.com/<stage>
/<resource> \
```

Our example:

```
curl -X POST
https://abc123xyz.execute-api.us-east-1.amazonaws.com/prod
/predict \
```

- Testing can now be done in your Terminal:

```
curl -X POST https://abc123xyz.execute-api.us-east-1.
amazonaws.com/prod/predict \
    -H "Content-Type:  application/json" \
    -d '{ "features":[22, 1, 7.25, 1, 0, 0, 1, 0, 0] }'
```

2. CLI:

- Create **REST API**:

```
aws apigateway create-rest-api --name 'TitanicAPI'
```

- After retrieving the restApiId, get the root resource ID:

```
aws apigateway get-resources --rest-api-id $restApiId
```

- Create /predict resource:

```
aws apigateway create-resource \
    --rest-api-id $restApiId \
    --parent-id $rootResourceId \
    --path-part predict
```

- Store the resource ID of /predict:

```
predictResourceId=xyz
```

- Create POST method:

```
aws apigateway put-method \
    --rest-api-id $restApiId \
    --resource-id $predictResourceId \
    --http-method POST \
    --authorization-type "NONE"
```

- Integrate with Lambda:

```
aws apigateway put-integration \
    --rest-api-id $restApiId \
    --resource-id $predictResourceId \
    --http-method POST \
    --type AWS_PROXY \
    --integration-http-method POST \
    --uri arn:aws:apigateway:us-east-1:lambda:path/
2015-03-31/functions/arn:aws:lambda:us-east-1:ACCOUNT_ID:
function:predictTitanic/invocations
```

- Deploy the API:

```
aws apigateway create-deployment \
    --rest-api-id $restApiId \
    --stage-name prod
```

3. SDKs:

- Create **REST API**:

```
apigw = boto3.client('apigateway')
api = apigw.create_rest_api(name='TitanicAPI')
rest_api_id = api['id']
```

- Get the **Root Resource ID**:

```
resources = apigw.get_resources(restApiId=rest_api_id)
root_id = resources['items'][0]['id']
```

- Create /predict Resource:

```
predict_resource = apigw.create_resource(
    restApiId=rest_api_id,
    parentId=root_id,
    pathPart='predict'
)
predict_id = predict_resource['id']
```

- Define POST Method:

```
apigw.put_method(
    restApiId=rest_api_id,
    resourceId=predict_id,
    httpMethod='POST',
    authorizationType='NONE'
)
```

- Integrate Lambda with API Gateway:

```
lambda_arn =
'arn:aws:lambda:<region>:<account-id>:function:predictTitanic'
```

```
uri = f'arn:aws:apigateway:<region>:lambda:path/
2015-03-31/functions/{lambda_arn}/invocations'

apigw.put_integration(
    restApiId=rest_api_id,
    resourceId=predict_id,
    httpMethod='POST',
    type='AWS_PROXY',
    integrationHttpMethod='POST',
    uri=uri
)
```

- Deploy the API:

```
deployment = apigw.create_deployment(
    restApiId=rest_api_id,
    resourceId=predict_id,
    stageName='prod'
)
```