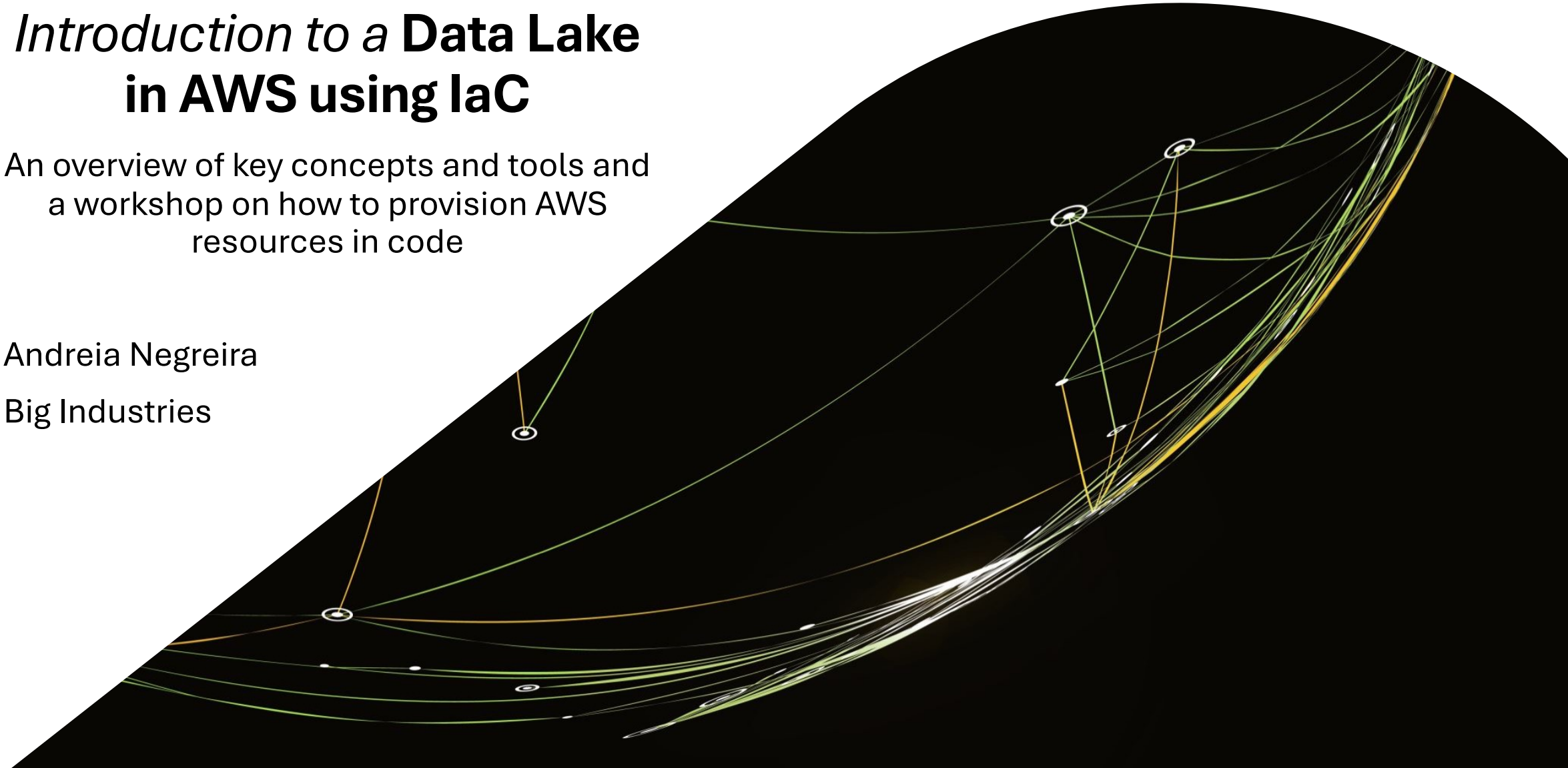
Introduction to a **Data Lake in AWS using IaC**

An overview of key concepts and tools and a workshop on how to provision AWS resources in code

Andreia Negreira

Big Industries



Hi, my name is Andreia

I am a 30 years old mom from Brazil

I live in Antwerp with my 1 year old baby, my husband and our cat

I have a bachelor and a master in the Environmental Engineering field – I worked in the researching field back in Brazil

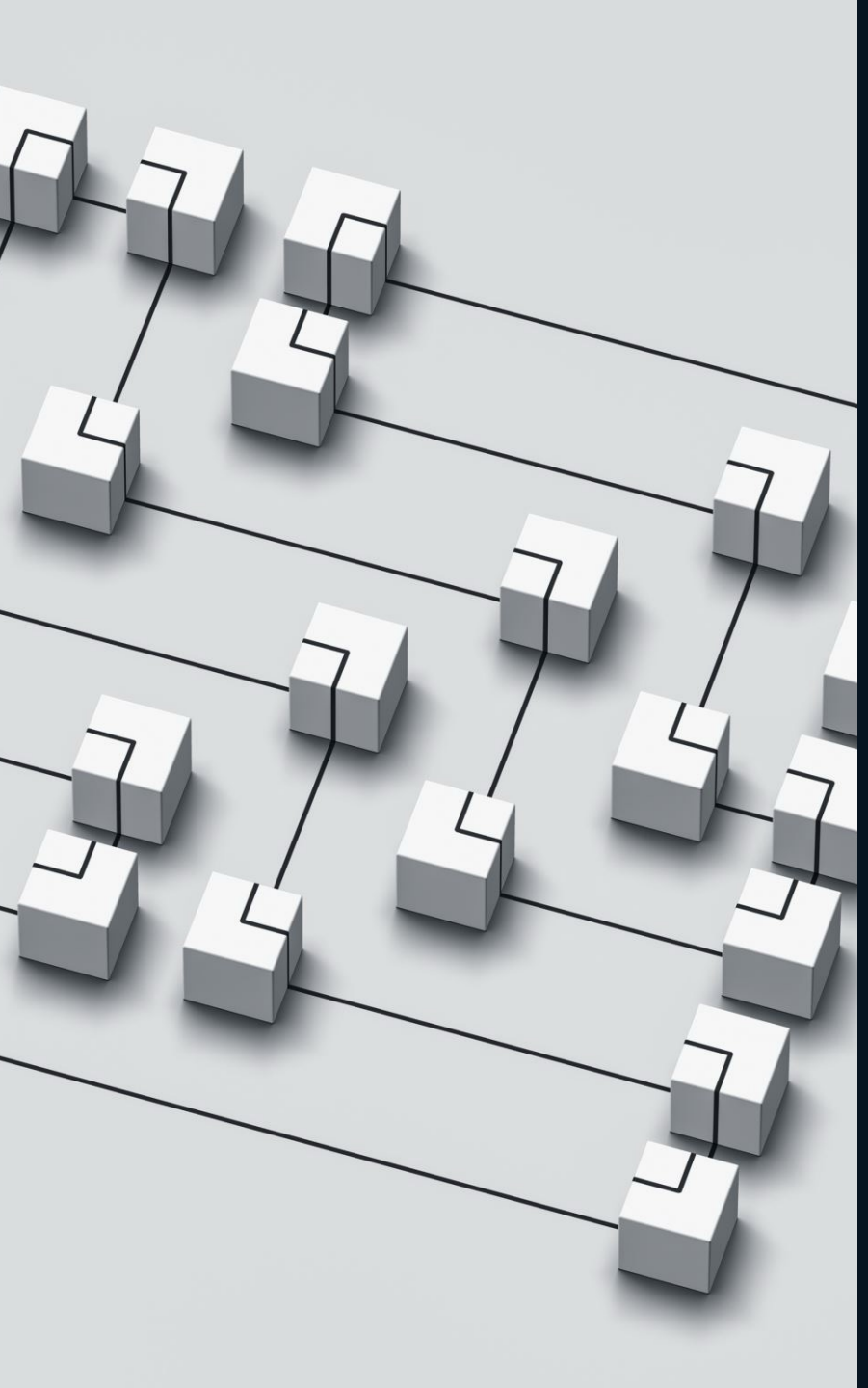
I came to Belgium in Dec 2020

I was first introduced to Python around October 2022

After becoming a mom, I followed a course to learn python on youtube around February 2023

I joined BeCode in June 2023 (ARAI-5, Data Engineering path)

I started my internship in January 2024 and it will last until July – so I'm currently a Data Engineer intern!



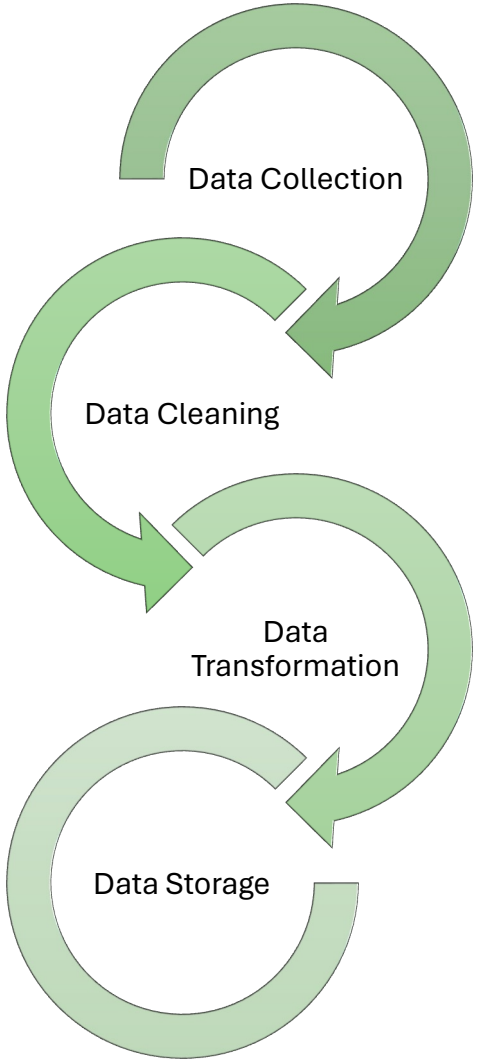
Data Processing

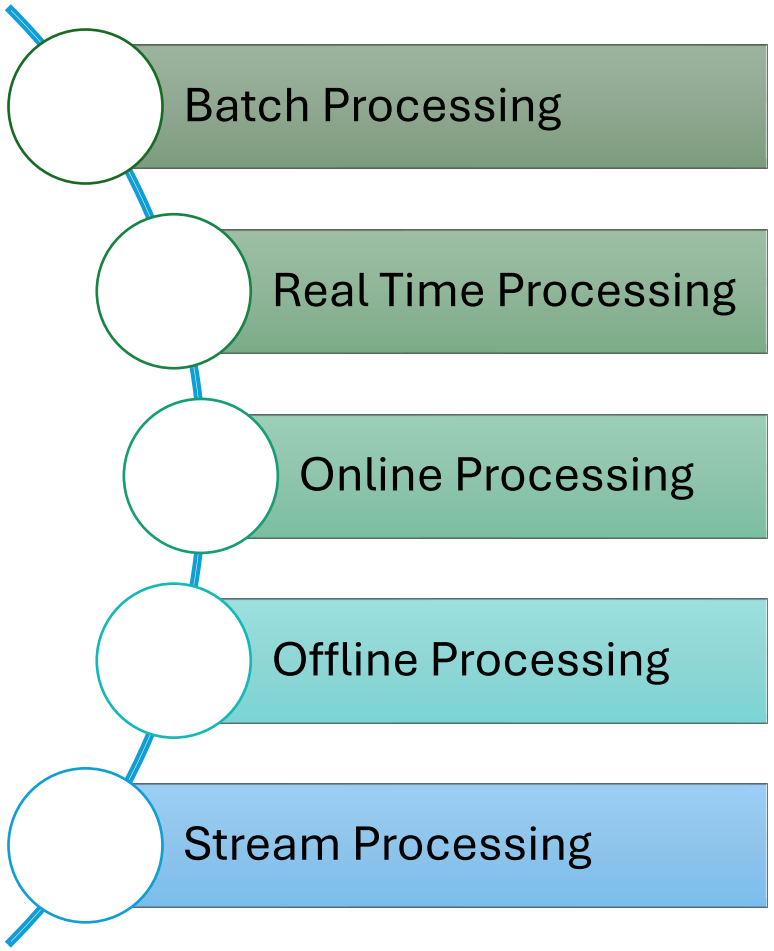
→ The process of transforming *raw* or *unprocessed* data into a **clean and readable format** is called ***data processing***.

Data is transformed, what does that mean? → Process of applying multiple data operations, like removing null data, sorting it, filtering it, applying a dataframe, etc., in order to make the raw data more readable.

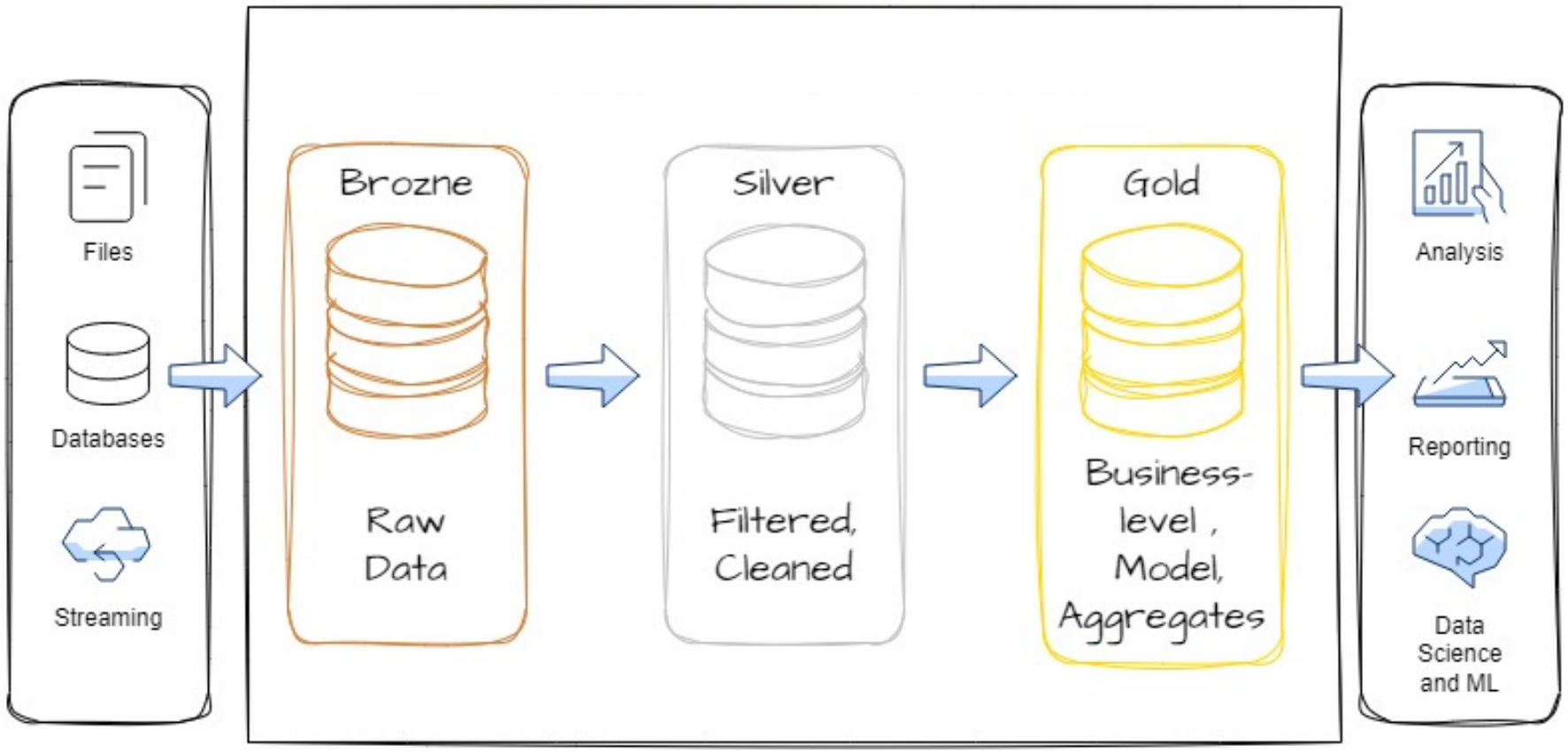
Data processing is done by either a **Data Engineer** or a **Data Scientist**!

Data Processing: Cycle and Types





Data Lake with medallion architecture



Introduction to AWS Cloud Service

- On-demand cloud computing solution;
- More than 200+ services;
- Third most used cloud provider in Belgium;
- Popular in the finance, healthcare and technological sectors.
- Key benefits: scalability, flexibility, cost-efficiency and security.

AWS Glue Overview

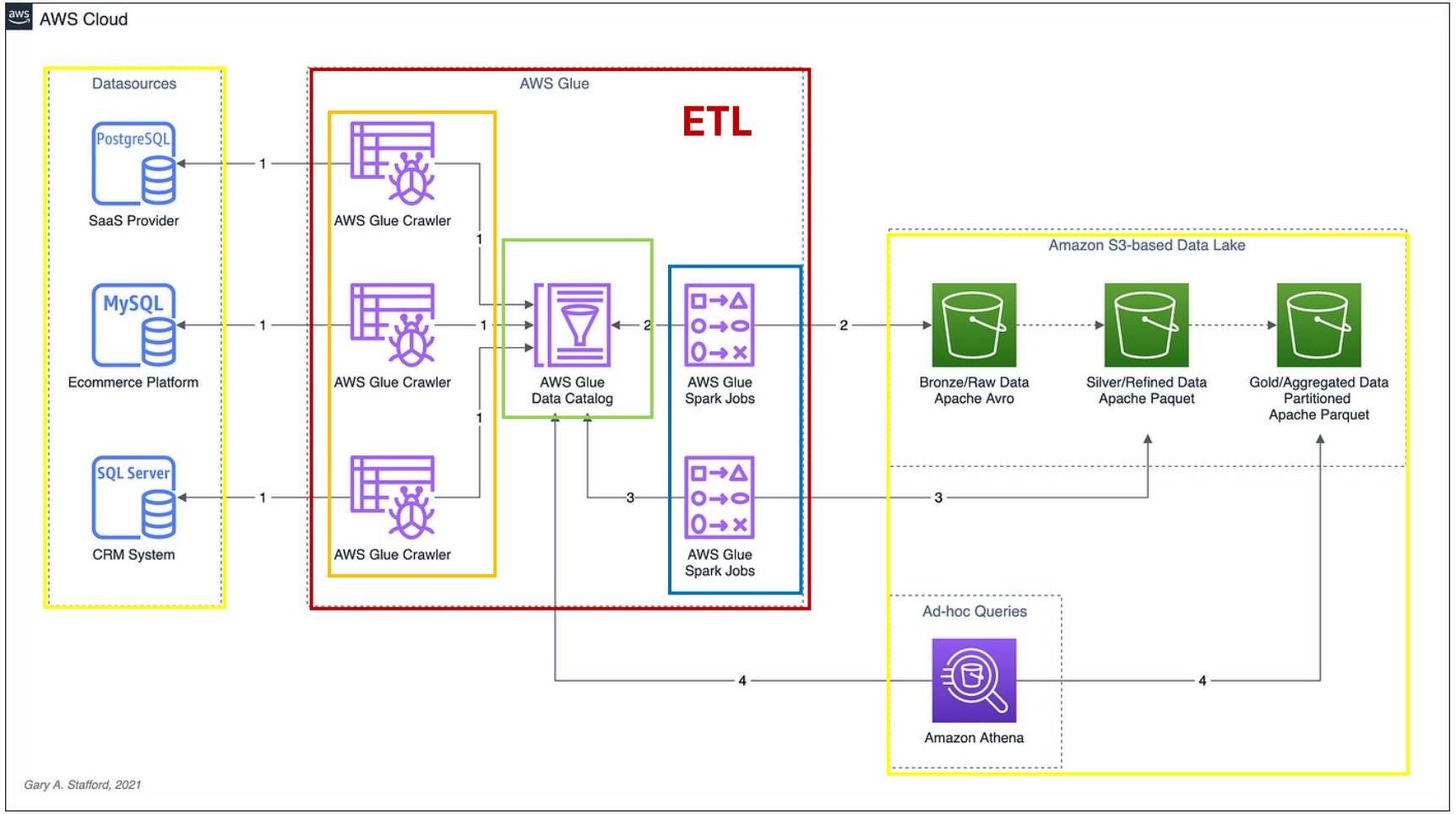
On a high level:

AWS Glue is a **serverless** data integration service that makes it easy to:

- Discover,
- Prepare,
- Integrate,
- Move data and
- Apply ETL and ELT frameworks for:
- Analytics,
- Machine learning,
- Application development, etc.

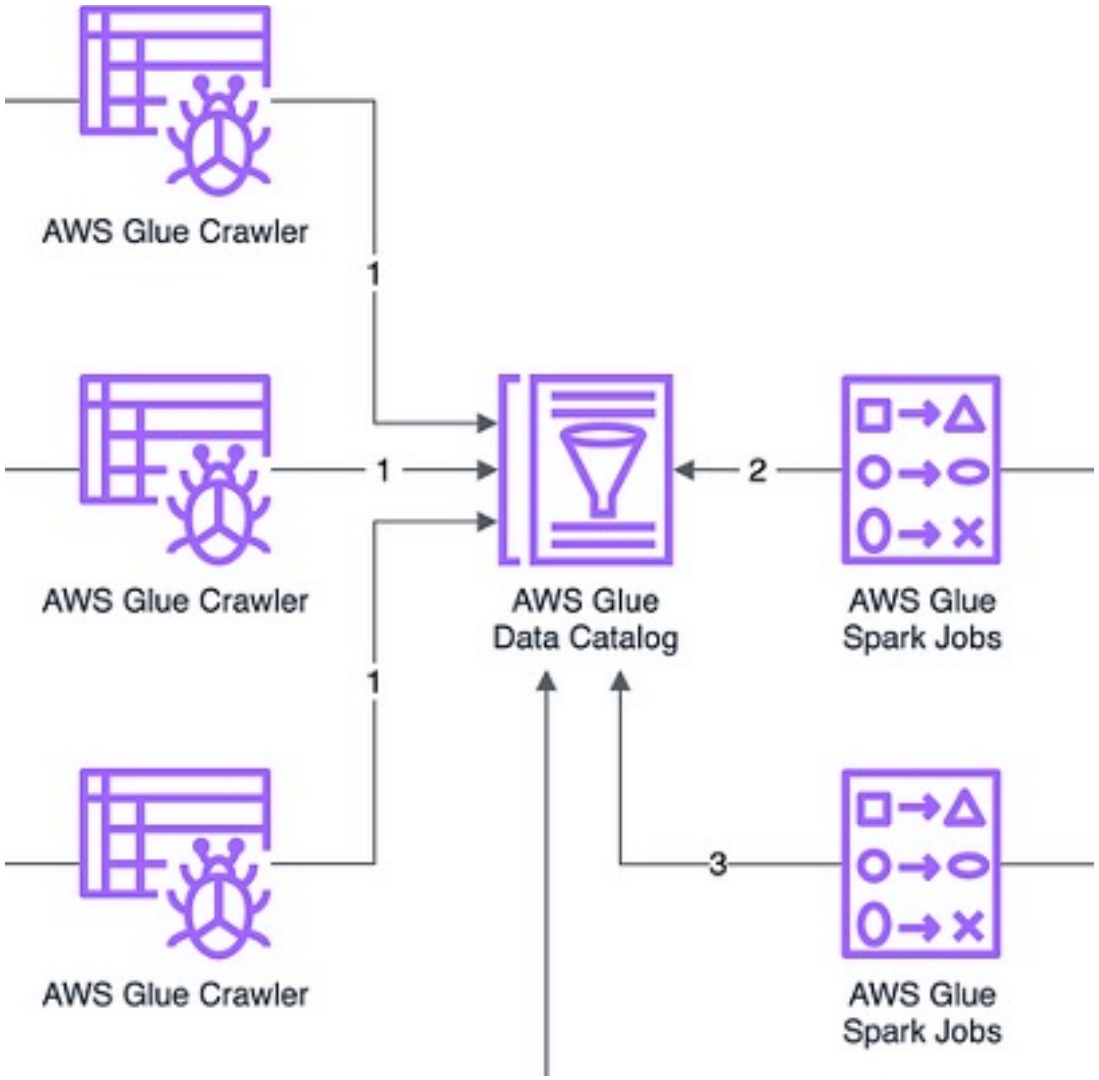
It provides a scalable, flexible, and cost-effective way to manage data across various data stores.

AWS Glue Architecture Overview



AWS Glue Architecture Overview

- **AWS Glue Crawler:** Crawlers automatically scan your data stored in various sources, such as S3 buckets, and infer a schema to it.
- **AWS Glue Data Catalog:** is where you can see and manage your metadata database.
- **AWS Glue Jobs:** are scripts that you create to perform ETL (Extract, Transform, Load) operations on your data. These jobs can be written in Python or Scala and run in parallel to process large datasets efficiently.



AWS Glue Jobs Primitives

Python shell

Spark

Ray (released in Nov 2022)

Ray vs Spark job – when to use one or another?

Ray:

What is Ray?: Ray is a framework for building and running distributed Python applications.

How it Works: It helps you run Python tasks in parallel across many computers.

Use Cases: Great for machine learning, data processing, and other Python tasks that need to run quickly and efficiently.

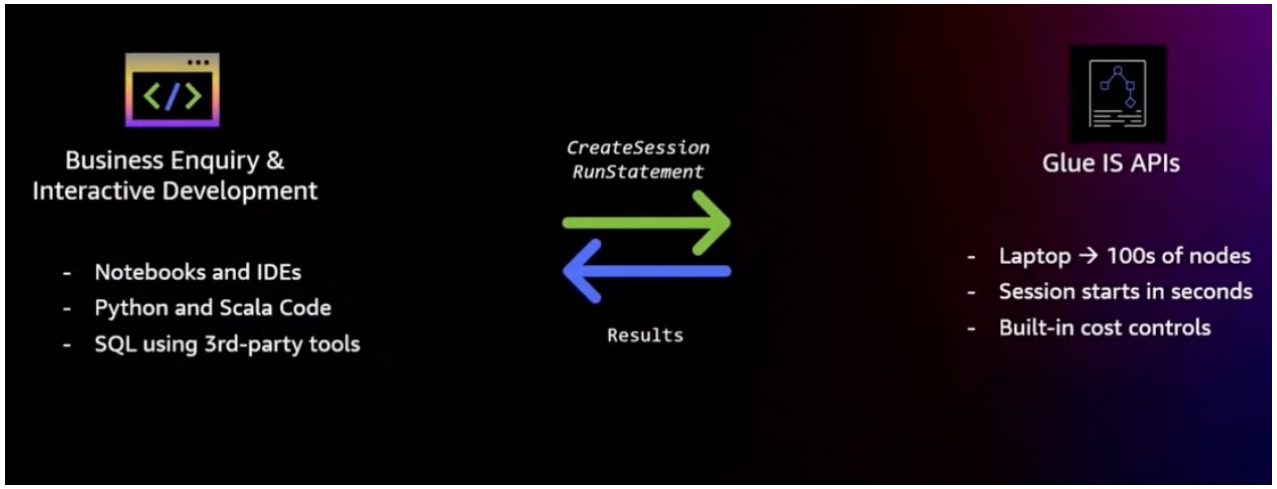
Spark:

•**What is Spark?:** Apache Spark is a framework for big data processing.

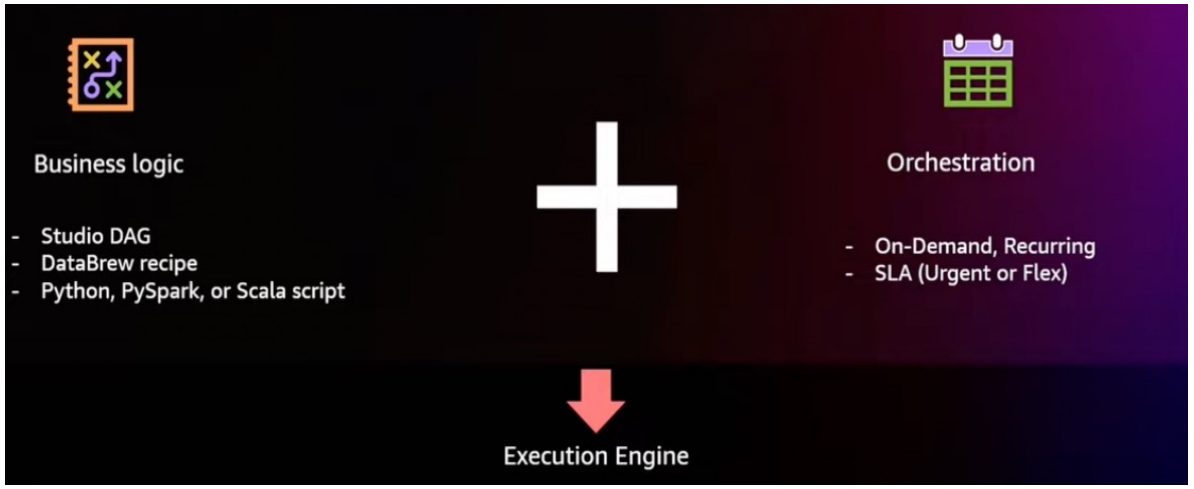
•**How it Works:** It allows you to process large amounts of data across many computers, using languages like Python, Scala, or Java.

•**Use Cases:** Ideal for complex data processing, ETL jobs (Extract, Transform, Load), and big data analytics.

AWS Glue Primitives Details



Python shell



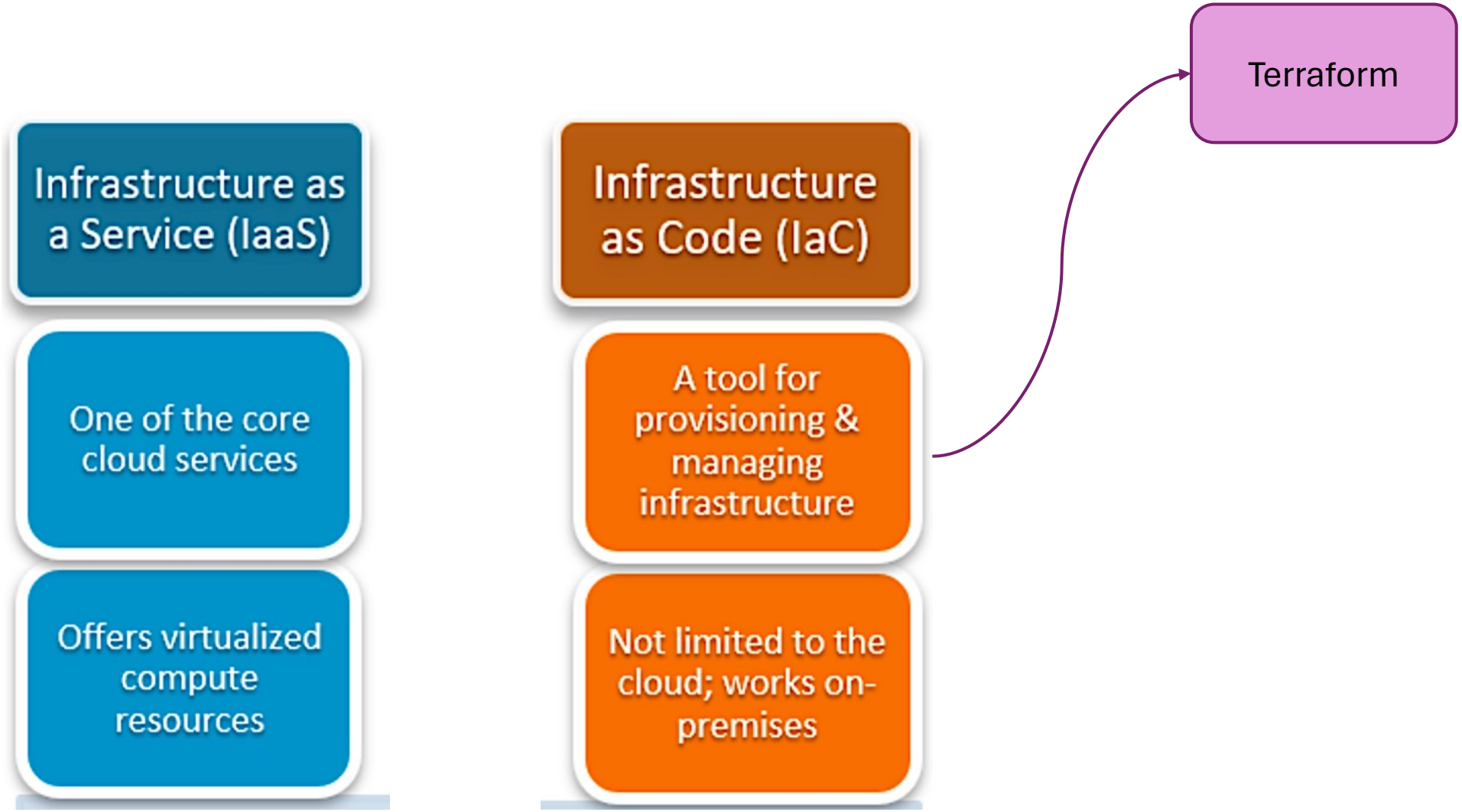
Spark and Ray Jobs

IaC vs IaaS

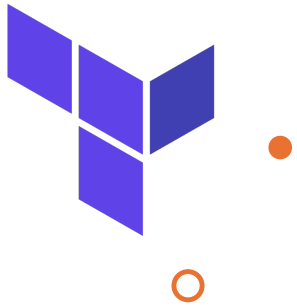
- How do I provision my resources on the cloud in order to process my data?

- You have only 2 options:

IaC vs IaaS


Infrastructure as a Service (IaaS)
One of the core cloud services
Offers virtualized compute resources
Infrastructure as Code (IaC)
A tool for provisioning & managing infrastructure
Not limited to the cloud; works on-premises
Terraform

Terraform



Infrastructure automation to provision and manage resources in any cloud or data center.

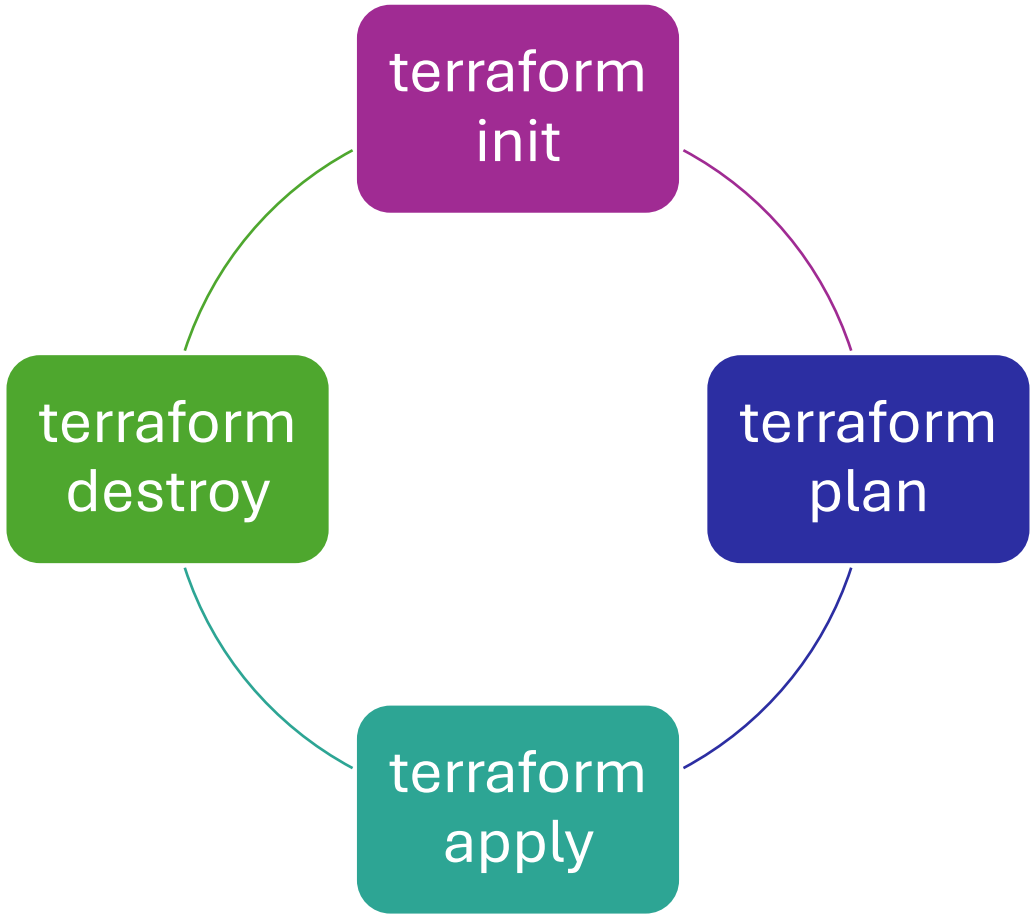
Some Benefits of using it:

- **Automation:** Reduces manual intervention;
- **Consistency:** Ensures uniformity across environments;
- **Version Control:** Infrastructure changes can be versioned;
- **Open Source**: freely available for use and contributions from the community.

Declarative structure example:

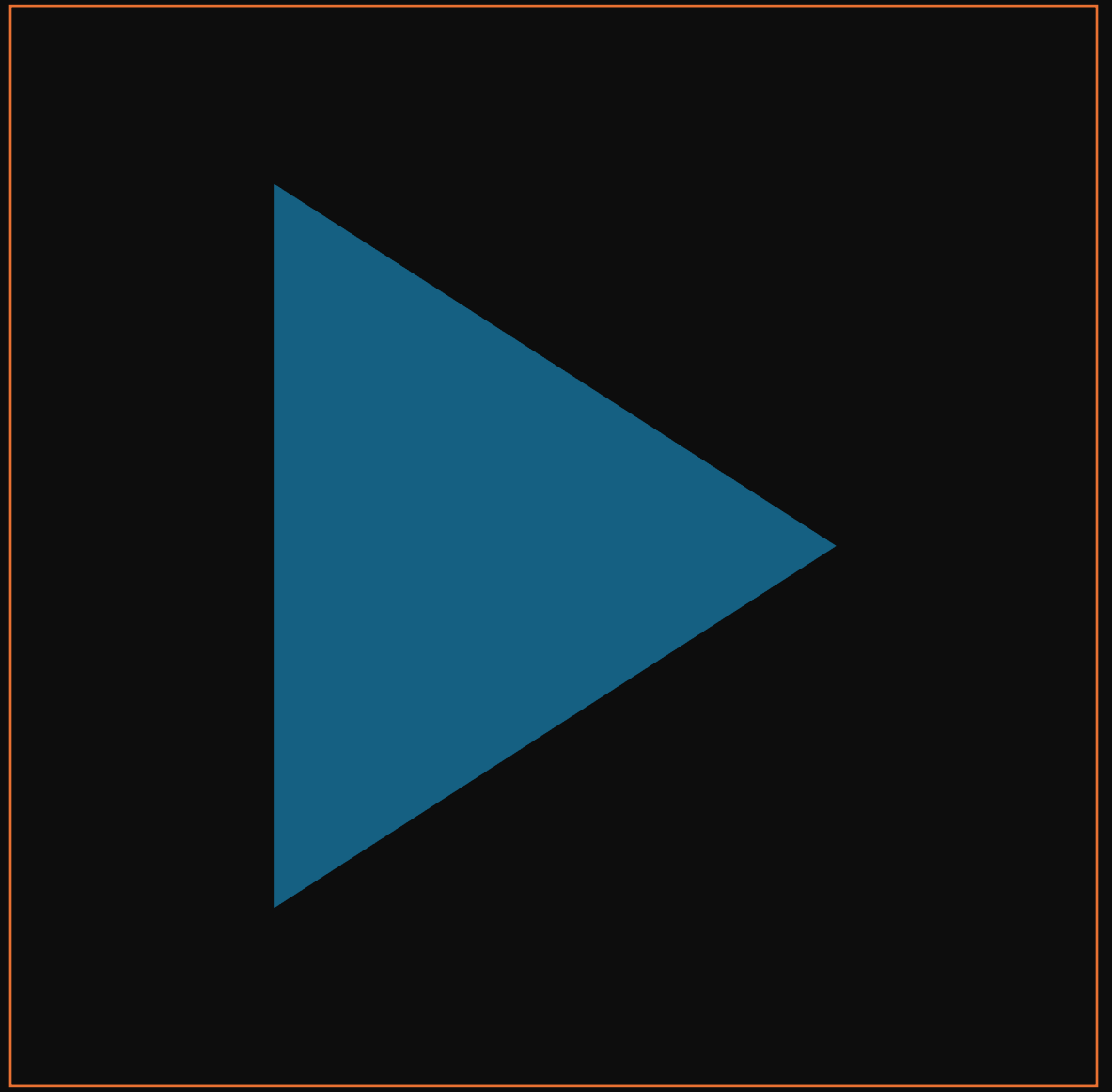
```
resource “<provider>”_“<resource_type>” “name” {
        config options…
        key = “value”
        key2 = “value”
}
```

Some important commands



Workshop Time

Hands-on Session



Workshop Overview

Main goal of this workshop: use AWS glue for Ray to train your immo eliza model by provisioning your resources with Terraform!

So today we will:

- Provision an S3 bucket for your project
- Upload your script and your data to your bucket
- Create and configure an AWS Glue for Ray job using HSC language
- Execute a Ray script using AWS Glue environment.

Step 1: Setting up the environment

Tools Required:

✓ AWS CLI

✓ Terraform CLI

✓ AWS Management Console

✓ Code editor

→ Requirements met with the preparations!

Step 2: Directory Structure

.

|-- **terraform/**

| |-- main.tf

|-- **script/**

| `-- model.py

|-- **data/**

| `-- immo_data.csv

|-- .gitignore

`-- ReadMe.md

- Create 3 folders:

1. terraform/
2. script/
3. data/

- Insert in script/ your .py file and in data/ your .csv file.

Step 3: Provisioning your bucket

```
.
|-- terraform/
|   |-- main.tf
|-- script/
|   `-- model.py
|-- data/
|   `-- immo_data.csv
|-- .gitignore
`-- ReadMe.md
```

- Inside terraform/ create a file called main.tf
- Let's provision your first bucket and upload your files to different folders.
- Your bucket structure should be like this:

```
your_bucket_name/
|-- script/
|   `-- your_model.py
|-- data/
|   `-- your_data.csv
`-- model/
    `-- your_final_model.pickle
```

Step 3: Provisioning your bucket

```hcl
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region = "eu-west-1"
}
resource "aws_s3_bucket" "local-bucket-name" {
  bucket = "your_bucket_name"
  force_destroy = true
}
resource "aws_s3_object" "folder-plus-object-upload" {
  bucket = aws_s3_bucket.local-bucket-name.id
  key    = "data/your_data.csv"     # this should be the path expected on your bucket!
  source = "../data/your_data.csv"  # relative path to your local file
  etag = filemd5("../data/your_data.csv") # relative path to your local file
}
```

In your main.tf:

1. Declare your provider section.
2. Declare your bucket.
3. Upload your files to your bucket. Adjust the folder names and relative paths accordingly.

your_bucket_name/

|-- **script/**

| `-- your_model.py

|-- **data/**

| `-- your_data.csv

`-- **model/**

`-- your_final_model.pickle

Step 4: Updating the .main block in your script

- Now that you have your bucket, your folders and files in your bucket, you're ready to update the main section in your script .py
- Change the value of the variables accordingly.

```
def main():
    input_bucket_name = "your-bucket-name"        # your bucket name
    input_file_key = "your_data.csv"              # path in your bucket to your .csv file
    output_bucket_name = "your-bucket"            # your bucket name
    output_file_key = "model/your_model.pickle"  # path to your model output. Change after model/

    df_future = read_csv_from_s3.remote(input_bucket_name, input_file_key)
    df = ray.get(df_future)
    cleaned_df = cleaning(df)
    trained_model = model(cleaned_df)
    ray.get(write_model_to_s3.remote(output_bucket_name, output_file_key, trained_model))
```

Step 5: Creating an IAM Role for your glue job

```
resource "aws_iam_role" "glue_role" {
  name = "glue-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Service = "glue.amazonaws.com"
        }
        Action = "sts:AssumeRole"
      },
    ]
  })
}

resource "aws_iam_role_policy_attachment" "glue_policy_attach" {
  role       = aws_iam_role.glue_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole"
}
```

Still on main.tf

- It’s necessary to provision a permission role to your resource to it interact with other resources (in our case, s3 bucket) and perform operations.
- The whole permission allows a full interaction with s3.

Step 6: Provisioning your glue for ray job

```
resource "aws_glue_job" "demo_ray_job" {
  name         = "your-job-name"
  role_arn     = aws_iam_role.glue_role.arn # your glue role local name
  glue_version = "4.0"
  description  = "foo"
  worker_type  = "Z.2X" # only machine available for ray jobs
  number_of_workers = 5 # minumum of 2. Stick with a low number to avoid unexpected costs

  command {
    name            = "glueray"
    python_version  = "3.9"
    runtime         = "Ray2.4"
    script_location = "s3://${aws_s3_bucket.local-bucket-name.bucket}/your_script.py"
    # update bucket name and script path in your bucket
  }

  default_arguments = {
    "--pip-install" = "pandas==1.3.3,numpy==1.21.2,scikit-learn==0.24.2,xgboost==1.4.2,boto3==1.18.12"
    # this is how you install extra libraries in ray
    # in case you need an extra one, you should pass the name and version here
  }
}
```

Still on main.tf

- This resource is composed of 3 sections:

1. General parameters section.
2. Command section: here you declare the type of the job and pass the path to your script.
3. Default arguments section: extra parameters that are important for your job, such as installing extra libraries.

Step 7: Running Terraform

Yay! We're almost there! Now you have all your resources and you're ready to provision them!

Run the following commands in this order:

1. Initialize Terraform: *terraform init*
2. Plan the Infrastructure: *terraform plan*
3. Apply the Configuration: *terraform apply* (enter 'yes' when prompt)

What can be go wrong in this step?

- If it din’t work, don’t freak out! Try to read the Terraform logs to understand what happened first.

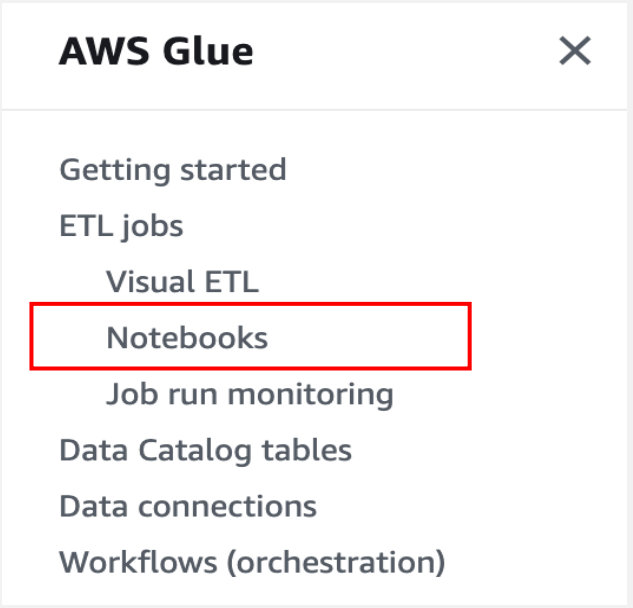
You probably:

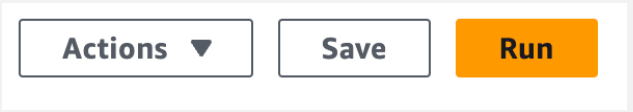
- Did a typo somewhere
- Forgot to use the .id or .arn to reference your bucket
- Have permission errors

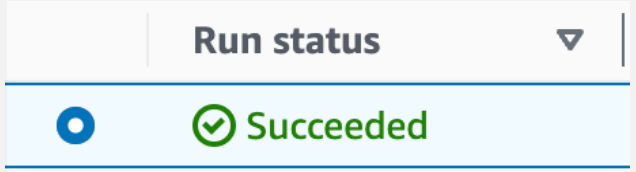
Step 8: Running the glue job

If everything went well in the previous step…

- Go to your AWS console and search for: AWS glue
- On your left, select the option: Notebooks on ETL jobs
- Click on your job >> go to ‘runs’ >> trigger the orange button written ‘Run’
- Select the current run on ‘Job runs’ to monitor the status of your job.







What can be go wrong in this step?

- If your job failed, let’s try to understand what happened!

With your script, ray probably:

- Couldn’t import all the required packages
- Have problems in terms of permissions
- Your logic is not correct, try to check it out again.

Step 9: Checking up your model on your bucket

- If your run was successful, now you can search for: s3
- Click on your bucket
- Go to the folder model/
- Is your model stored there? 

And that's it for our hands-on session today! 

Feel free to download your model and use it for predictions! But more importantly:

- RUN 'terraform destroy' when you're ready to say goodbye to your resources.
- Don't worry, you can always run 'terraform apply' again and everything will be back.

Extra tips…

In case you want to push this code to a github repo:

- Take a look at the files *‘variables.tf’* and *‘terraform.tfvars’*.
- In the *‘variables.tf’* you can declare a variable, its type, if it is a sensitive variable, a description and a default value.
- In case you are working with sensitive variables, like your credentials or the arn for a permission, you can declare your variable in *‘variables.tf’*, set it as sensitive and pass the value in *‘terraform.tfvars’*.
- Do not forget to add *‘terraform.tfvars’* to the .gitignore!

Thank you for your attention!

Any questions?

You can find me here:

- LinkedIn
- andreia.negreia@bigindustries.be