

Przetwarzanie Danych w Chmurze Publicznej

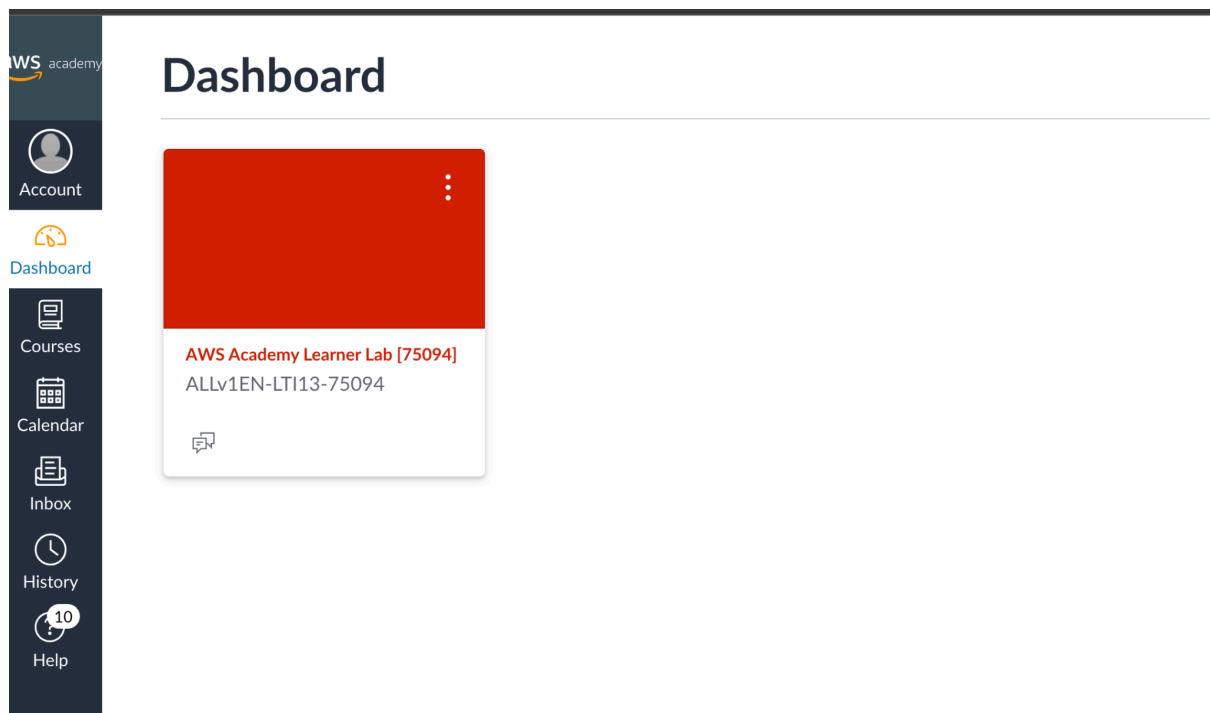
Laboratorium

Przygotowanie środowiska

Dostęp do konta AWS

Całość laboratoriów robimy na koncie AWS Academy Learner Lab.

1. Zaloguj się na <https://awsacademy.instructure.com/> i otwórz labs (zaproszenia zostały wysłane na Twoją skrzynkę UAM):



2. Wejdź do modules i uruchom moduł AWS Academy Learner Lab

The screenshot shows the AWS Academy Learner Lab interface. The left sidebar contains navigation options: Account, Dashboard, Courses, Calendar, Inbox, History, and Help. The main content area is titled 'ALLv1EN-LTI13-75094 > Modules'. It lists three modules:

- Course Welcome and Overview**
 - Pre-Course Survey
 - AWS Academy Learner Lab Student Guide
- AWS Academy Learner Lab Compliance and Security**
 - Learn how to effectively use the AWS Academy Learner Lab
 - Module Knowledge Check (100 pts, Score at least 70.0)
- AWS Academy Learner Lab**
 - Launch AWS Academy Learner Lab (highlighted with a red arrow)

3. Wystaruj laboratorium (**Start Lab**) a następnie kliknij AWS, gdy kółko zmieni się na zielone.

Otworzy się konsola AWS z której możesz wykonać całość laboratoriów (będziemy pracowali na Cloud9).

W zakładce "AWS Details" możesz znaleźć tymczasowe AWS Credentials jeśli chcesz pracować z tym kontem z własnego środowiska (local development)

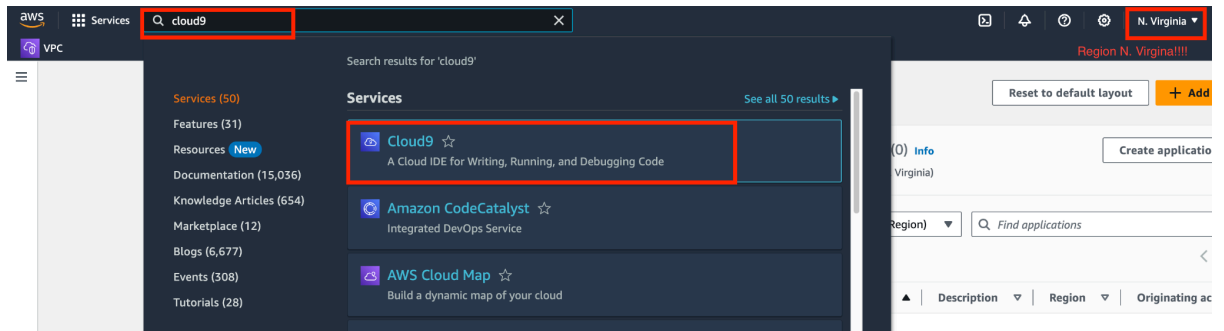
The screenshot shows the AWS Academy Learner Lab interface. The left sidebar contains navigation options: Account, Dashboard, Courses, Calendar, Inbox, History, and Help. The main content area is titled 'ALLv1EN-LTI13-7... > Modules > AWS Academy Le... > Launch AWS Academy Learner Lab'. It shows the 'Start Lab' button (highlighted with a red arrow) and the 'AWS' console (highlighted with a red arrow). The 'AWS' console shows 'Used \$1.2 of \$100' and '04:00'. A red arrow points to the 'AWS' console with the text 'Kliknij gdy zielone'. The 'Start Lab' button is also highlighted with a red arrow. The 'AWS Details' panel is visible on the right, showing 'Learner Lab' and various links.

Setup środowiska (Cloud9)

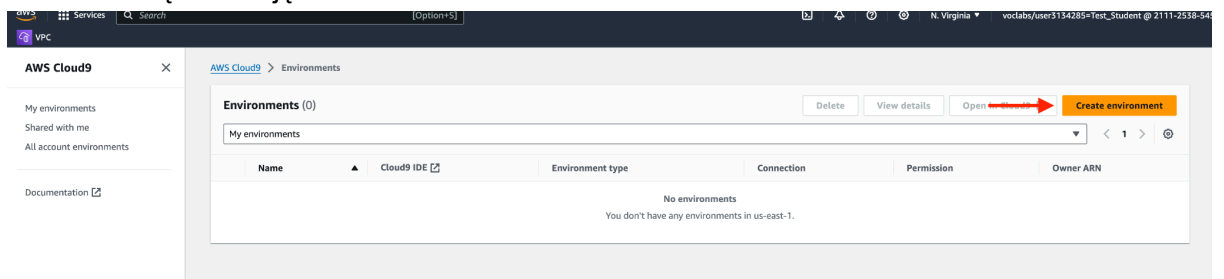
W tym kroku utworzysz nową instancję środowiska Cloud9

1. Otwórz serwis Cloud9 w konsoli AWS (poprzedni krok).

Zwróć uwagę w którym regionie AWS jesteś (pracujemy w **N. Virginia us-east-1**)



2. Utwórz nową instancję Cloud 9



Nazwij ją <inicjały>-dev, wybierz rozmiar maszyny t3-small:

The screenshot shows the AWS Cloud9 'Create environment' page. The breadcrumb navigation is 'AWS Cloud9 > Environments > Create environment'. The main heading is 'Create environment Info'. Under the 'Details' section, the 'Name' field contains 'jk-dev' and is highlighted with a red box. Below it, the 'Description - optional' field is empty. The 'Environment type' section has two options: 'New EC2 instance' (selected) and 'Existing compute'. Under the 'New EC2 instance' section, the 'Instance type' section has three options: 't2.micro (1 GiB RAM + 1 vCPU)', 't3.small (2 GiB RAM + 2 vCPU)' (selected and highlighted with a red box), and 'm5.large (8 GiB RAM + 2 vCPU)'. There is also an 'Additional instance types' option. The 'Platform' section is set to 'Amazon Linux 2023' and the 'Timeout' section is set to '30 minutes'.

Details

Name
jk-dev
Limit of 60 characters, alphanumeric and unique per user.

Description – optional
Limit 200 characters.

Environment type Info
Determines what the Cloud9 IDE will run on.

New EC2 instance
Cloud9 creates an EC2 instance in your account. The configuration of your EC2 instance cannot be changed by Cloud9 after creation.

Existing compute
You have an existing instance or server that you'd like to use.

New EC2 instance

Instance type Info
The memory and CPU of the EC2 instance that will be created for Cloud9 to run on.

t2.micro (1 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.

t3.small (2 GiB RAM + 2 vCPU)
Recommended for small web projects.

m5.large (8 GiB RAM + 2 vCPU)
Recommended for production and most general-purpose development.

Additional instance types
Explore additional instances to fit your needs.

Platform Info
This will be installed on your EC2 instance. We recommend Amazon Linux 2023.

Amazon Linux 2023

Timeout
How long Cloud9 can be inactive (no user input) before auto-hibernating. This helps prevent unnecessary charges.

30 minutes

W ustawieniach sieciowych wybierz **Secure Shell** i naciśnij Create

Network settings [Info](#)

Connection
How your environment is accessed.

AWS Systems Manager (SSM)
Accesses environment via SSM without opening inbound ports (no ingress).

Secure Shell (SSH)
Accesses environment directly via SSH, opens inbound ports.

▶ **VPC settings** [Info](#)

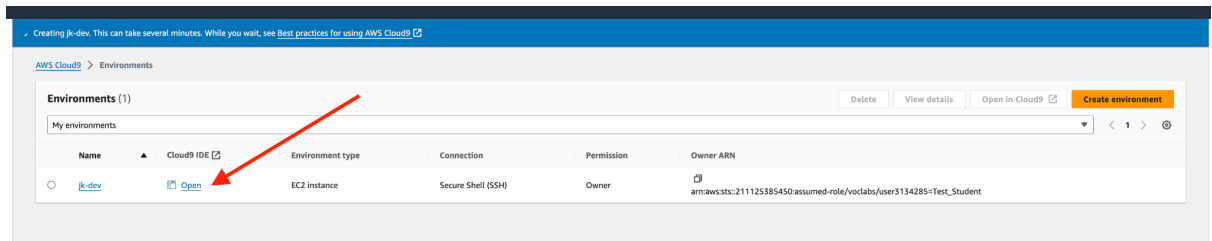
▶ **Tags – optional** [Info](#)
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

The following IAM resources will be created in your account

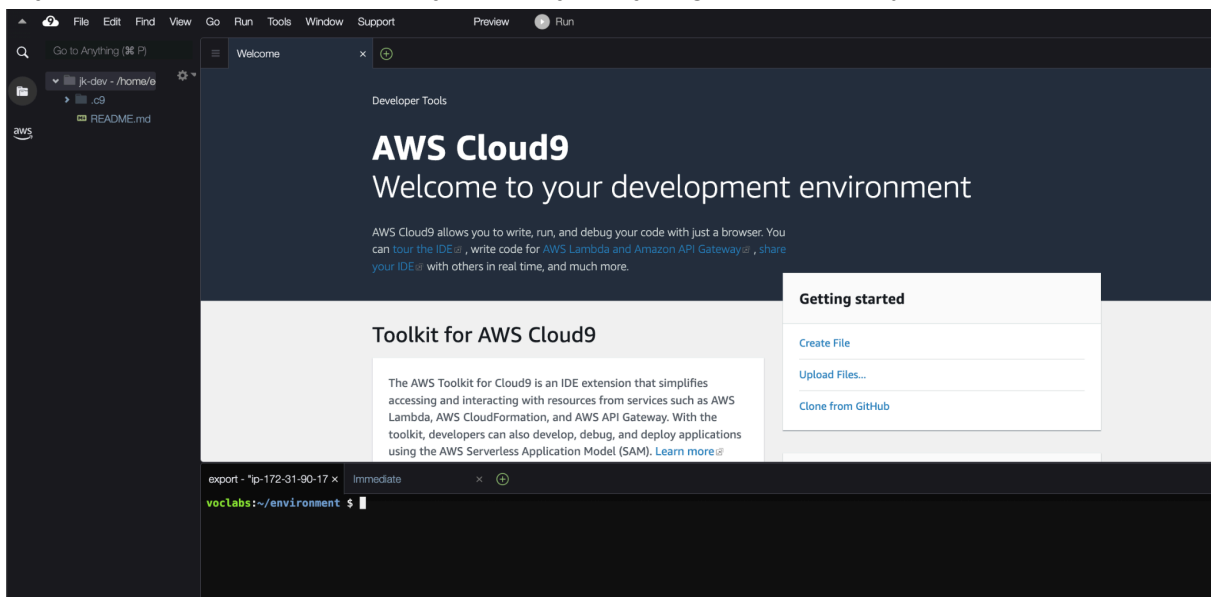
- **AWSServiceRoleForAWSCloud9** – AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. [Find out more](#)

Cancel **Create**

3. Nowa instancja zostanie utworzona - uruchom ją,



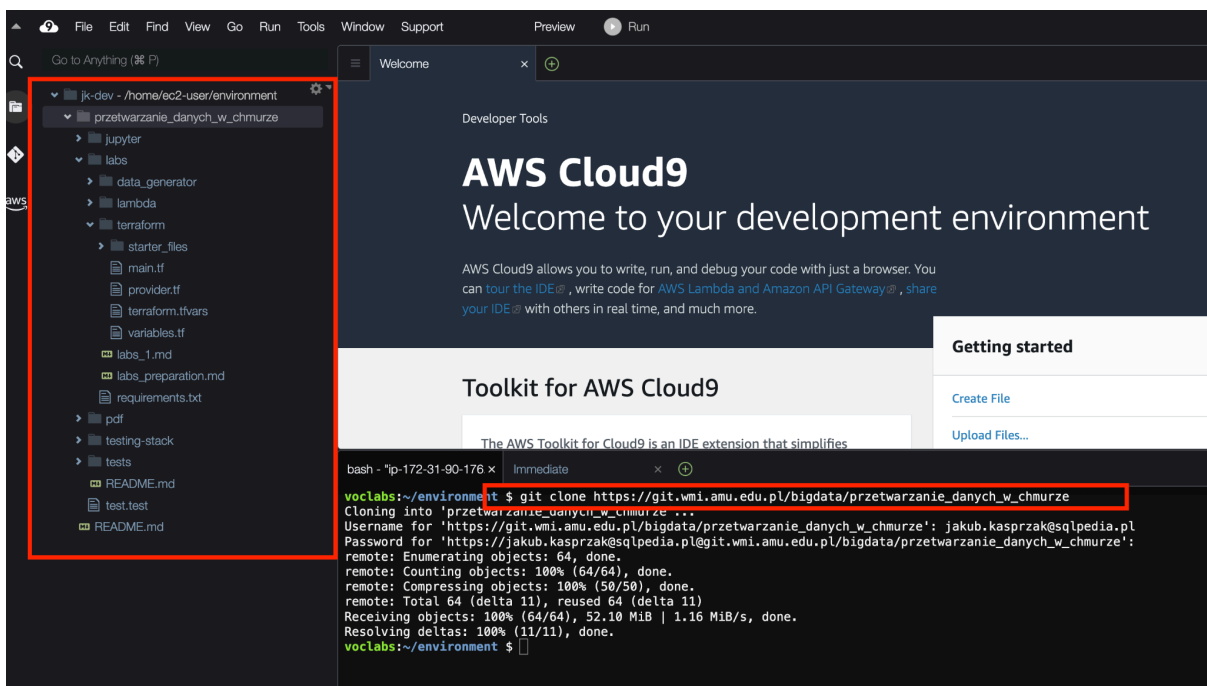
4. Pojawi się nowe okno z IDE - twoja instancja C9 jest gotowa do pracy.



5. W dolnej części ekranu, masz dostęp do terminala (możesz otworzyć wiele okien terminala w osobnych zakładkach).

Sklonuj repozytorium używając swoich danych autoryzujących (user / password)
https://git.wmi.amu.edu.pl/bigdata/przetwarzanie_danych_w_chmurze

```
$ git clone https://git.wmi.amu.edu.pl/bigdata/przetwarzanie_danych_w_chmurze.git
```



6. Uruchom skrypt `./labs/setup.sh`, który zainstaluje terraform i inne wymagane pakiety (sprawdź zawartość skryptu i pliku `requirements.txt`)


```
$ cd przetwarzanie_danych_w_chmurze/labs/  
$ source ./setup_c9.sh
```

Skrypt zainstaluje wszystkie wymagane biblioteki i przygotuje środowisko do pracy.

7. [OPCJONALNY SETUP] - Jeśli chcesz usprawnić swoją pracę, możesz użyć tej instancji jako backend dla Twojego IDE (VS Code). Instrukcja jak to zrobić tutaj: [Field Notes: Use AWS Cloud9 to Power Your Visual Studio Code IDE | AWS Architecture Blog \(amazon.com\)](#)

8. Po poprawnym uruchomieniu kroku 6 (skrypt) otwórz nowy terminal i sprawdź czy terraform został zainstalowany poprawnie:

```
$ terraform --version
```



A terminal window screenshot showing the execution of the command `terraform --version`. The terminal output is as follows:

```
voclabs:~/environment/przetwarzanie_danych_w_chmurze/labs (master) $ cd terraform/  
voclabs:~/environment $ terraform --version  
Terraform v1.8.1  
on linux_amd64  
voclabs:~/environment $
```

Twoja instancja jest gotowa do pracy!

Terraform 101

Wszystkie obiekty, które będziemy tworzyć w AWS można wykonać za pomocą konsoli lub kodu. Najpopularniejszymi rozwiązaniami **IaaS** (Infrastructure as a Code) jest natywny dla AWS CloudFormation oraz **Terraform**. W przypadku naszych zajęć będziemy korzystali z Terraform.

Jeśli po raz pierwszy masz do czynienia z Terraform - rekomenduję przejście tutorialu aby zrozumieć najważniejsze obiekty i założenia (całość nie powinna zabrać więcej niż ok 30 minut) :

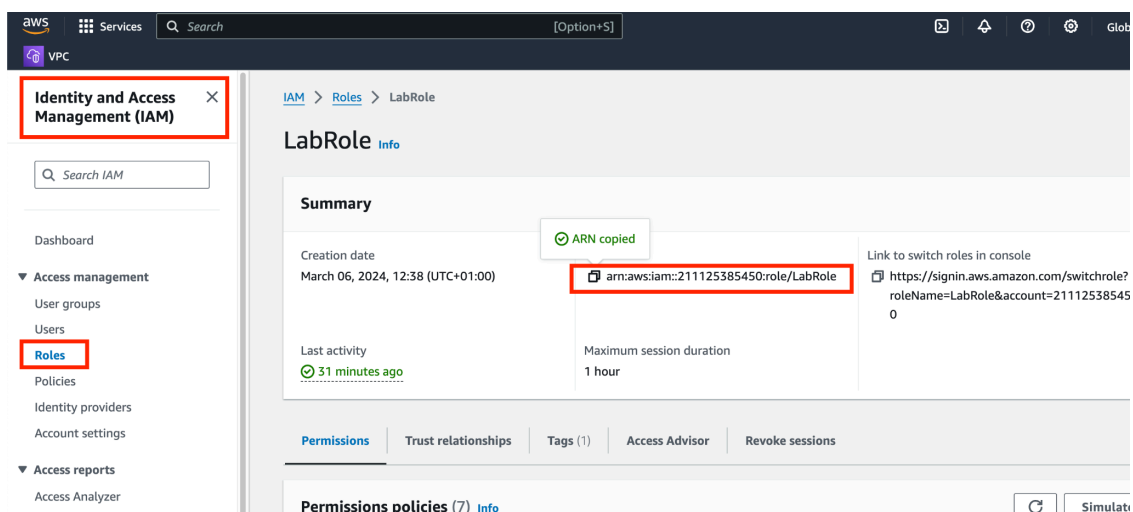
<https://learn.hashicorp.com/tutorials/terraform/infrastructure-as-code?in=terraform/aws-get-started>

Podstawowe polecenia Terraform które będziemy używali :

1. Terraform **init** - inicjalizacja stanu terraform (tylko raz na początku projektu).
2. Terraform **plan** - po inicjalizacji i utworzeniu pierwszych obiektów, możemy zobaczyć co zostanie stworzone / zmodyfikowane - to jest delta pomiędzy obecnym stanem a docelowym.
3. Terraform **apply** - wdrożenie zmian (nowe obiekty, aktualizacja istniejących lub kasowanie niepotrzebnych) - dokładnie to co pokazał plan.
4. Terraform **destroy** - usunięcie wszystkich obiektów

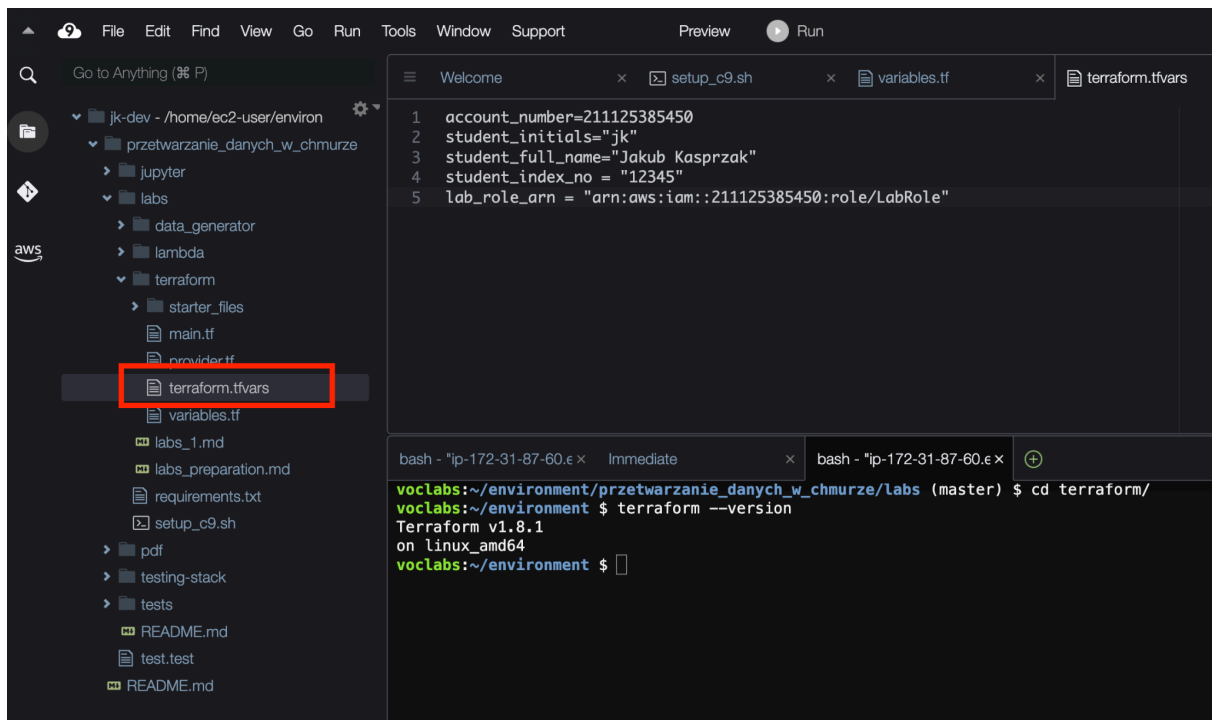
Konfiguracja podstawowych obiektów Terraform

1. Cały Twój kod TF będzie znajdował się w lokalizacji (sklonowane repo) `./przetwarzanie_danych_w_chmurze/labs/terraform/` i wszędzie w zadaniach gdy będziemy odwoływali się do kodu terraform - będzie to właśnie w tym folderze.
2. Otwórz serwis IAM w konsoli AWS i odzyskaj rolę LabRole - skopiuj jej ARN - będzie potrzebny do następnego kroku.



3. Zaktualizuj i **ZAPISZ** plik `./labs/terraform/terraform.tfvars` ze swoimi danymi (numer konta AWS, numer indeksu i inicjały oraz ARN roli LabRole) np:

```
account_number=211125385450
student_initials="jk"
student_full_name="Jakub Kasprzak"
student_index_no = "12345"
lab_role_arn = "arn:aws:iam::211125385450:role/LabRole"
```



4. Zainicjuj plik stanu Terraform poleceniem ***terraform init*** (od tego momentu wszystkie zmiany będą dokonywane na Twoim koncie AWS). Nigdy nie modyfikuj pliku stanu ręcznie!
Uruchom w katalogu `./labs/terraform/`

```

1 terraform {
2   required_providers {
3     aws = {
4       source = "hashicorp/aws"
5       version = "~> 5.0"
6     }
7   }
8 }
9
10 provider "aws" {
11   profile = "default"
12   region = var.region
13 }

```

```

bash - "ip-172-31-42-196" x Immediate x bash - "ip-172-31-42-196" x
voclabs:~/environment/przetwarzanie_danych_w_chmurze/labs/terraform (master) $ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.46.0...
- Installed hashicorp/aws v5.46.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

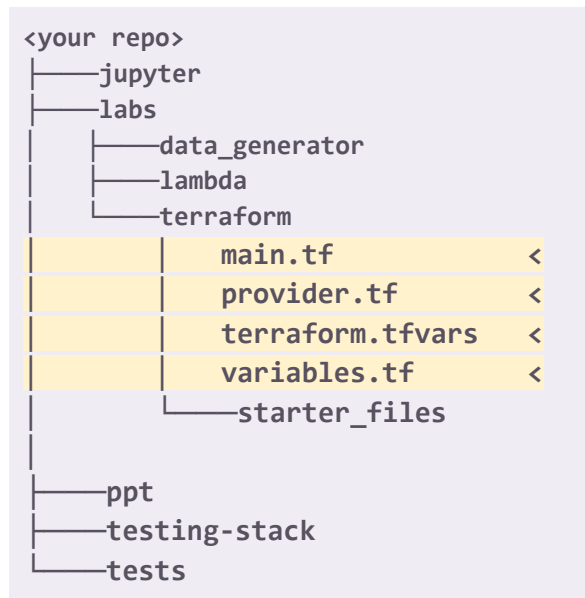
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
voclabs:~/environment/przetwarzanie_danych_w_chmurze/labs/terraform (master) $

```

Otwórz repozytorium - powinieneś mieć strukturę jak poniżej



5. Masz już skonfigurowane wszystkie narzędzia i zainicjowany terraform.

Przetwarzanie Danych w chmurze publicznej

Laboratorium

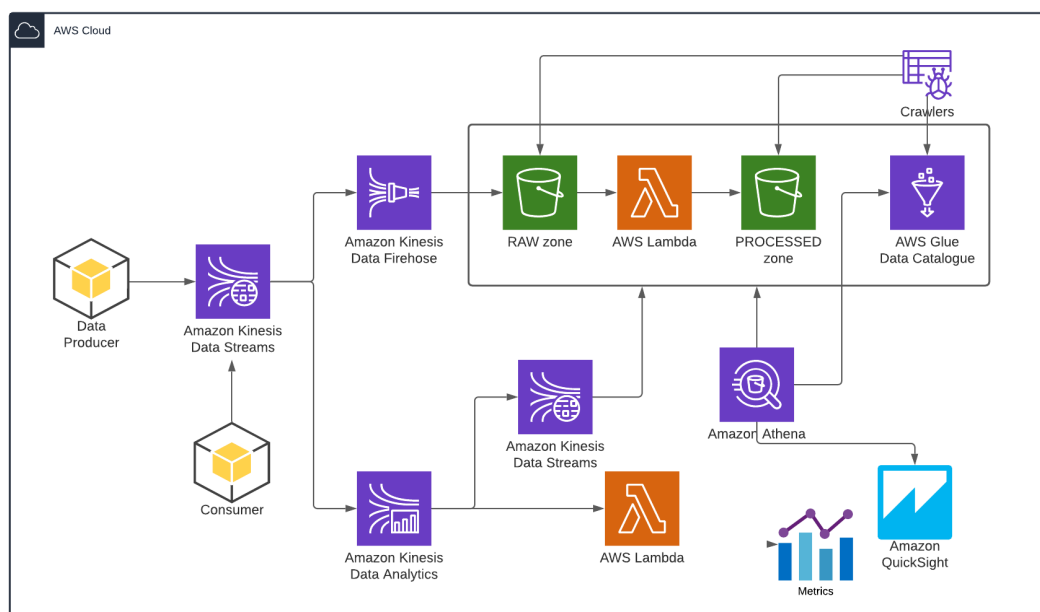
Scenariusz Laboratorium

W scenariuszu będziemy symulować ładowanie danych *near real-time* z giełdy kryptowalut Bitstamp (zlecenia sell/buy) do *Data Lake* z wykorzystaniem usług Kinesis. Dane będziemy przetwarzać pomiędzy **Raw** a **Processed** zone z wykorzystaniem funkcji Lambda.

Zarejestrujemy tabele w Glue Data Catalogue aby móc wykonywać zapytania ad-hoc w Athena (SQL).

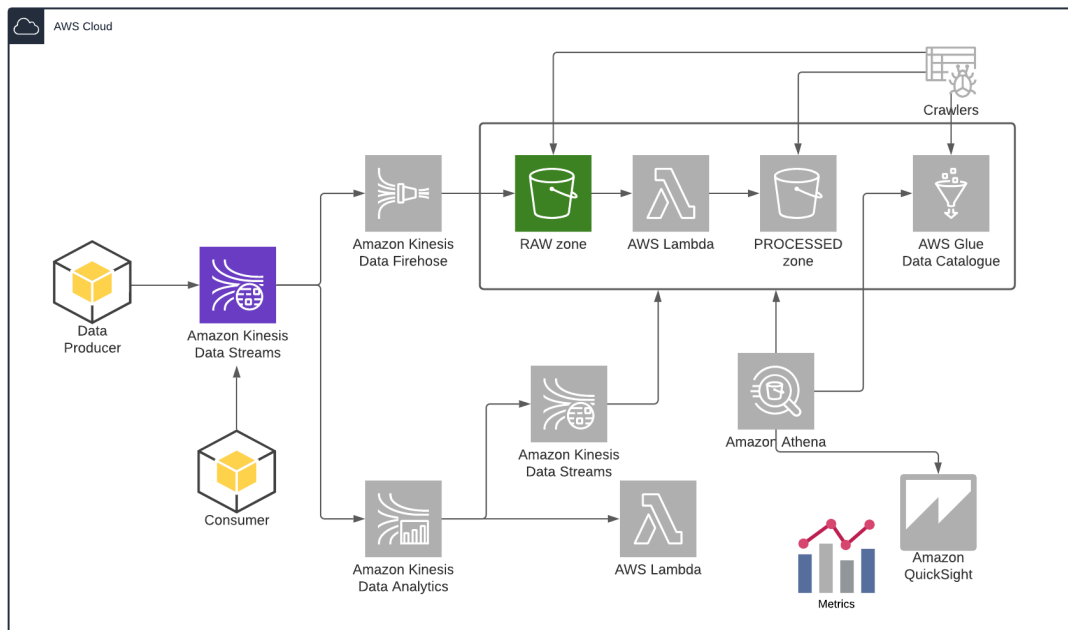
Do zbudowania systemu użyjemy następujących usług serverless :

- broker wiadomości Kinesis Data Stream
- zasilanie data lake Kinesis Delivery Stream
- Glue Data Catalogue jako Hive Metastore do przechowywania metadanych (tabele, partycje)
- silnik przetwarzania danych Presto udostępniający interfejs zgodny z ANSI SQL - usługa Athena (analitka ad-hoc)
- Ciągłe przetwarzanie danych, funkcja lambda odpalana w momencie wystąpienia zdarzenia zapisu pliku do S3
- Przykładowe dane do symulacji pochodzą z <https://www.cryptodatadownload.com/data/bitstamp/>



Ćwiczenie 1

W ćwiczeniu tym, stworzymy pierwsze, podstawowe elementy naszego Data Lake i infrastruktury do ładowania danych - **S3 buckety (RAW i PROCESSED) oraz Kinesis Data Stream**. Poznasz podstawowe zasady tworzenia obiektów w Terraform.



1. Utwórz nowy plik TF `./labs/terraform/s3.tf` zawierający definicję S3 bucketów RAW i PROCESSED w którym będziemy przechowywali dane.
 - a. Opis struktury S3 w TF :
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket
 - b. Stwórz buckety opisany następującymi parametrami :
 - i. Nazwa bucketu powinna składać się z :
`datalake-<zone>- $\{var.account_number\}$ - $\{var.student_initials\}$ - $\{var.student_index_no\}$`

Np: **`datalake-raw-100603781557-jk-12345`**
 - ii. Wykorzystaj parametr **`force_destroy`** (sprawdź do czego służy)
 - iii. Jako dobrą praktykę wykorzystaj tagowanie obiektów - dodaj tag **`Environment = DEV`**

c. Przykładowa definicja jednego z bucketów `./labs/terraform/s3.tf`

```
resource "aws_s3_bucket" "raw_bucket" {
  bucket =
    "datalake-raw-${var.account_number}-${var.student_initials}-${var.student_index_no}"
  force_destroy = true

  tags = {
    Purpose = "UAM Cloud Data Processing"
    Environment = "DEV"
  }
}
```

d. Uruchom **terraform plan**, aby sprawdzić jakie obiekty zostaną stworzone zgodnie z tą definicją (w folderze `./labs/terraform/`)

```
$ terraform plan
```

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create
```

```
Terraform will perform the following actions:
```

```
# aws_s3_bucket.raw_bucket will be created
+ resource "aws_s3_bucket" "dl_raw_bucket" {
  + acceleration_status      = (known after apply)
  + acl                      = "private"
  + arn                      = (known after apply)
  + bucket                  =
"datalake-raw-100603781557-jk-12345"
  + bucket_domain_name      = (known after apply)
  + bucket_regional_domain_name = (known after apply)
  + force_destroy           = true
  + hosted_zone_id          = (known after apply)
  + id                      = (known after apply)
  + region                  = (known after apply)
  + request_payer           = (known after apply)
```

```
+ tags = {
  + "Environment" = "DEV"
  + "Purpose"     = "UAM Cloud Data Processing"
}
+ website_domain = (known after apply)
+ website_endpoint = (known after apply)

+ versioning {
  + enabled = (known after apply)
  + mfa_delete = (known after apply)
}
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

-

Note: You didn't specify an "-out" parameter to save this plan, so Terraform can't guarantee that exactly these actions will be performed if "terraform apply" is subsequently run.

- e. Wykonaj kod poleceniem **terraform apply** i postępuj zgodnie z instrukcją. Sprawdź w konsoli AWS czy bucket został utworzony.

```
$ terraform apply
```

An execution plan has been generated and is shown below.

Resource actions are indicated with the following symbols:

```
+ create
```

Terraform will perform the following actions:

```

# aws_s3_bucket.raw_bucket will be created
+ resource "aws_s3_bucket" "raw_bucket" {
  + acceleration_status      = (known after apply)
  + acl                      = "private"
  + arn                     = (known after apply)
  + bucket                   =
"datalake-raw-100603781557-jk-12345"
  ...

Enter a value: yes

aws_s3_bucket.raw_bucket: Creating...
aws_s3_bucket.raw_bucket: Creation complete after 9s
[id=datalake-raw-100603781557-jk-12345]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

```

Otwórz serwis S3 w konsoli AWS i sprawdź czy bucket został stworzony.

2. Analogicznie stwórz bucket dla zony **Processed** (w tym samym pliku s3.tf). Przykład nazwy:

datalake-processed-100603781557-jk-12345

3. Utwórz nowy plik `./labs/terraform/kinesis_ds.tf` zawierający definicję **Kinesis Data Stream**
 - a. Opis obiektu TF znajdziesz w :
 - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/kinesis_stream
 - b. Stwórz nowy stream z wykorzystaniem następujących parametrów:
 - i. Liczba shardów = 1
 - ii. Nazwa powinna pasować do tego wzorca :

```
cryptostock-${var.account_number}-${var.student_initials}-${var.student_index_no}
```

Np: **cryptostock-100603781557-jk-12345**

- iii. **Retention period** = default (24h)
 - iv. Tag **Environment** = DEV
 - v. **enforce_consumer_deletion** = True (sprawdź w dokumentacji jakie ma znaczenie)
 - vi. Włącz metryki na poziomie shardu dla : ["IncomingBytes", "OutgoingBytes", "IncomingRecords", "OutgoingRecords"]
- c. Przykładowa definicja

```
resource "aws_kinesis_stream" "cryptostock_stream" {
  name =
  "cryptostock-${var.account_number}-${var.student_initials}-${var.student_index_no}"
  shard_count = 1

  enforce_consumer_deletion = true

  shard_level_metrics = [
    "IncomingBytes",
    "OutgoingBytes",
    "IncomingRecords",
    "OutgoingRecords"
  ]

  tags = {
    Purpose = "UAM Cloud Data Processing"
    Environment = "DEV"
    Owner = var.student_full_name
  }
}
```

- d. Utwórz Kinesis DS (uwaga ! koszty stały \$0.015 / h) - **terraform apply**.
Więcej na temat cen tej usługi
<https://aws.amazon.com/kinesis/data-streams/pricing/>
- e. Sprawdź w konsoli AWS czy data stream został utworzony (wejdź do usługi Kinesis -> Data Streams)

4. Wysyłanie danych do Kinesis DS

a. Przejrzyj kod prostego generatora danych

`./labs/data_generator/generator.py`. Zwróć uwagę na sposób w jaki dane są wysyłane do Kinesis - wykorzystujemy tu SDK (API), które charakteryzuje się niską wydajnością. Iterujemy po danych z pliku i wywołujemy metodę `put_record`.

(*) W ramach dodatkowego ćwiczenia (dla ambitnych :) możesz udoskonalić generator, przepisując go na `put_records` lub generować rekordy wielowątkowo / ew asynchronicznie aby zwiększyć wydajność.

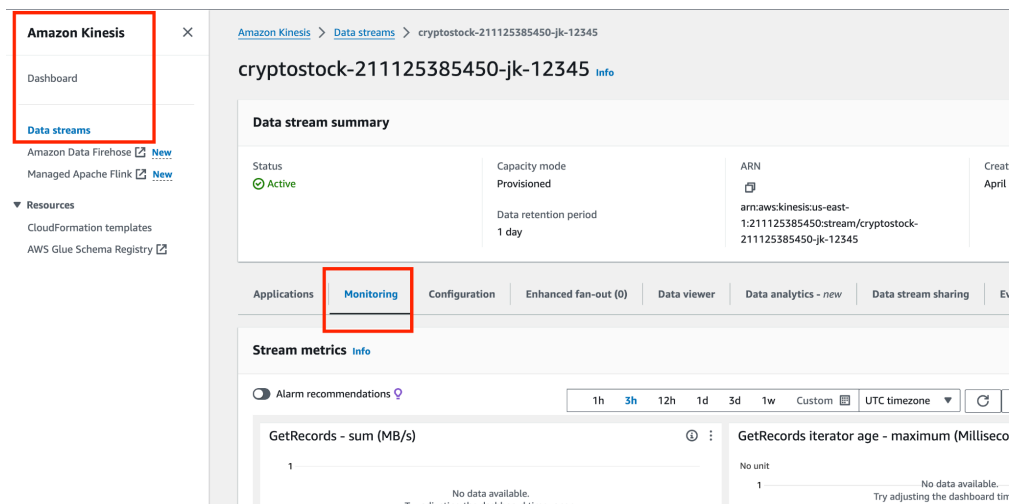
b. Uruchom generator w nowym oknie linii poleceń (będzie nam potrzebny aby cały czas wysyłał dane).

Użyj parametru `-k` aby przekazać nazwę kinesis data stream'a np. :

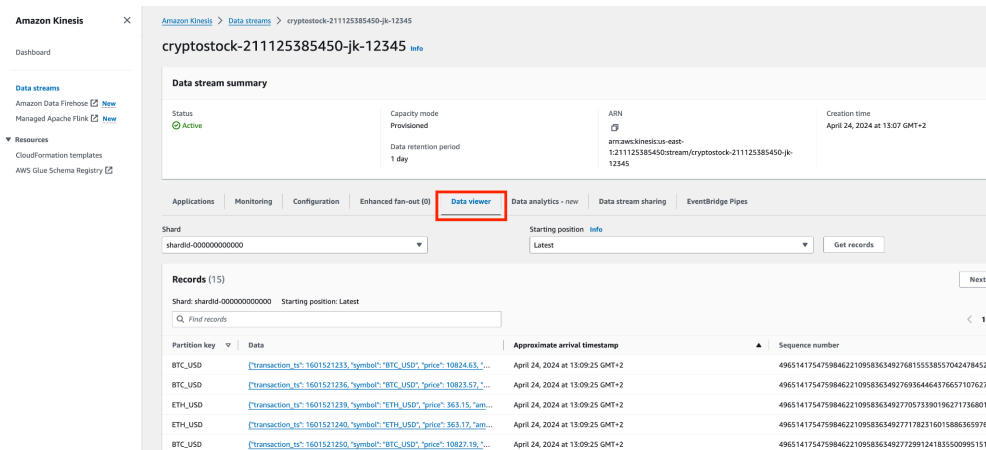
```
$ python generator.py -k cryptostock-211125385450-jk-12345
```

```
bash - "ip-172-31-87-60.ek" bash - "ip-172-31-87-60.ek"
voelabs:~/environment/przetwarzanie_danych_w_chmurze/labs/data_generator (master) $ python generator.py -k cryptostock-211125385450-jk-12345
[2024-04-24 06:56:43,386] (generator.py:125) INFO - Starting Simple Kinesis Producer (replaying stock data)
[2024-04-24 06:56:43,402] (credentials.py:1222) INFO - Found credentials in shared credentials file: ~/.aws/credentials
[2024-04-24 06:56:43,495] (generator.py:108) INFO - start replaying for the 1 time
[2024-04-24 06:56:43,574] (generator.py:58) INFO - Msg with trans_id=124289044 sent to shard shardId=000000000000 seq no 49651412060901519526072093236969867562483611616816398338
[2024-04-24 06:56:43,586] (generator.py:58) INFO - Msg with trans_id=124289045 sent to shard shardId=000000000000 seq no 49651412060901519526072093236971076488303226245991104514
[2024-04-24 06:56:43,595] (generator.py:58) INFO - Msg with trans_id=124289045 sent to shard shardId=000000000000 seq no 49651412060901519526072093236972285414122840875165810690
[2024-04-24 06:56:43,605] (generator.py:58) INFO - Msg with trans_id=124289050 sent to shard shardId=000000000000 seq no 49651412060901519526072093236973494339942455504340516866
[2024-04-24 06:56:43,615] (generator.py:58) INFO - Msg with trans_id=124289051 sent to shard shardId=000000000000 seq no 49651412060901519526072093236974703265762070133515223042
[2024-04-24 06:56:43,624] (generator.py:58) INFO - Msg with trans_id=124289054 sent to shard shardId=000000000000 seq no 49651412060901519526072093236975912191501684760689929216
[2024-04-24 06:56:43,635] (generator.py:58) INFO - Msg with trans_id=124289055 sent to shard shardId=000000000000 seq no 49651412060901519526072093236977121117401299391864635394
[2024-04-24 06:56:43,645] (generator.py:58) INFO - Msg with trans_id=124289059 sent to shard shardId=000000000000 seq no 49651412060901519526072093236978330043220914021039341570
```

c. Przejdź do Konsoli AWS i zobacz jak wyglądają metryki KDS, czy dane faktycznie docierają do sharda (może minąć pare minut zanim metryki będą zaktualizowane) - zakładka monitoring



d. Otwórz zakładkę Data Viewer i sprawdź czy eventy są wysyłane do streama:



e. (*) Dla chętnych - napisz skrypt (np. w jupyter notebook) który będzie czytał wiadomości z Kinesis Data Stream.

5. Zatrzymaj generator danych i usuń cały stack poleceniem **terraform destroy** (uruchom w katalogu w którym masz utworzone obiekty terraform)

```
$ terraform destroy
...
Plan: 0 to add, 0 to change, 3 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown
  above.
  There is no undo. Only 'yes' will be accepted to confirm.

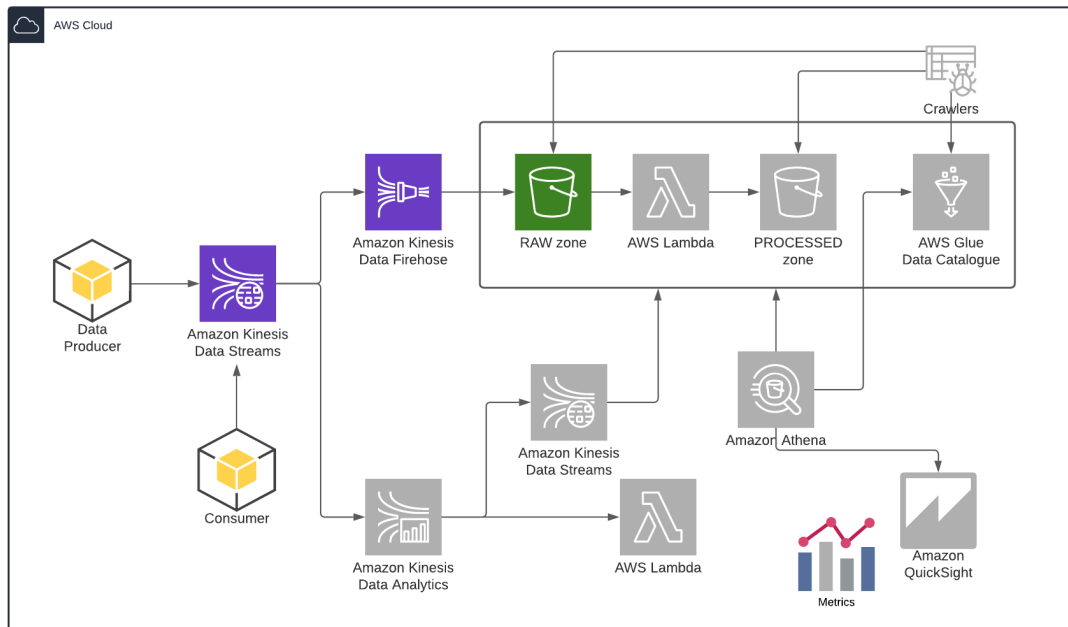
Enter a value: yes
```

6. Sprawdź w konsoli AWS czy faktycznie wszystko zostało posprzątane. Jeśli S3 bucket nie został usunięty - prawdopodobnie zapomniałeś o parametrze **force_destroy**. Usuń więc najpierw pliki (z konsoli AWS) a potem bucket.

Ćwiczenie 2

Do naszego systemu dołożymy kolejny element, odpowiedzialny za ładowanie danych do Data Lake.

W tym celu wykorzystamy usługę Kinesis Firehose



1. Utwórz obiekty zdefiniowane w ćwiczeniu 1 (*terraform apply*) S3 buckety + Kinesis Data Stream - zostały one usunięte w ostatnim punkcie.
2. Utwórz nowy Kinesis **Delivery Stream** z poziomu konsoli **AWS**, przejrzyj możliwe opcje.

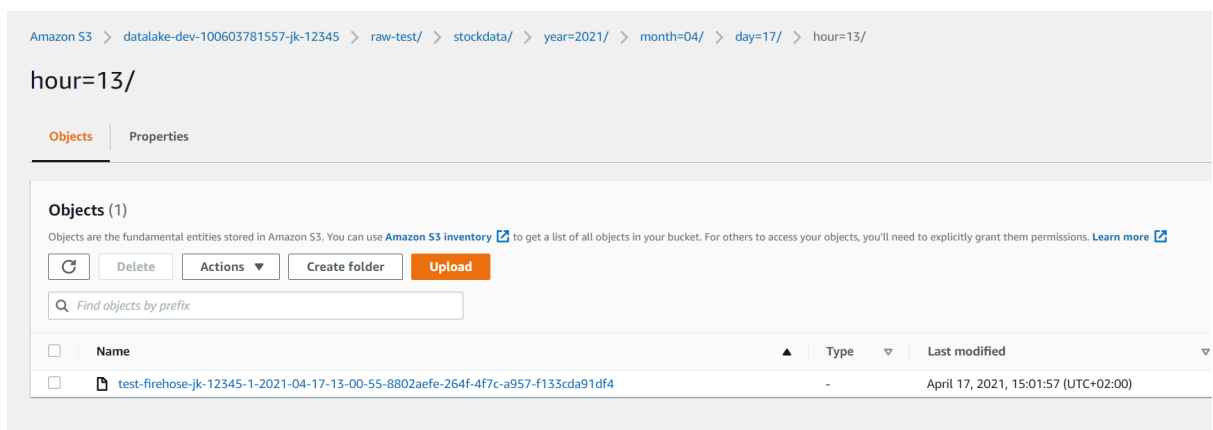
Wybierz następujące parametry :

- a. Nazwa = *test-firehose- $\{inicjały\}$ - $\{nr_indeksu\}$*
- b. Źródło danych = Kinesis Data Stream (wybierz utworzony w p. 1 stream)
- c. Brak transformacji
- d. Miejsce docelowe S3 - Twój bucket utworzony w p.1

Wzorce prefiksów:

- i. Dane :
raw-test/stockdata/year= $\{timestamp:yyyy\}$ /month= $\{timestamp:MM\}$ /day= $\{timestamp:dd\}$ /hour= $\{timestamp:HH\}$ /
- ii. Błędy:
raw-test/stockdata_errors/ $\{firehose:error-output-type\}$ /year= $\{timestamp:yyyy\}$ /month= $\{timestamp:MM\}$ /day= $\{timestamp:dd\}$ /hour= $\{timestamp:HH\}$ /

- e. Rozmiar bufora = 1MB
 - f. Częstotliwość zrzucania danych do S3 = 60s
 - g. Rola IAM - utwórz nową (domyślna opcja)
3. Uruchom generator danych testowych (patrz ćwiczenie 1 p. 3). Upewnij się że dane są poprawnie generowane do KDS.
 4. Poczekaj parę minut (5-6) i sprawdź czy dane są zapisywane w S3 z odpowiednim prefixem. Początkowy data dump może się odbyć z pewnym opóźnieniem (inicjalizacja serwisu), kolejne porcje danych powinny napływać zgodnie z przyjętą strategią.



5. Utwórz obiekt Kinesis Firehose Delivery Stream.

[aws kinesis firehose delivery stream | Resources | hashicorp/aws | Terraform | Terraform Registry](#)

Utwórz nowy plik `./labs/terraform/kinesis_fh.tf` zawierający definicję **Kinesis Delivery Stream (Firehose)**

- a. Nazwa (wzorzec) :


```
"firehose-${var.account_number}-${var.student_initials}-${var.student_index_no}"
```
- b. Źródło danych : Kinesis Data Stream (wybierz utworzony w p. 1 stream) - referencja terraform.
- c. Brak transformacji
- d. Miejsce docelowe S3 - Twój bucket utworzony w p.1 (referencja terraform)

Wzorce prefiksów:

 - i. Dane :


```
raw-zone/stockdata/year={!timestamp:yyyy}/month={!timestamp:MM}/day={!timestamp:dd}/hour={!timestamp:HH}/
```

- ii. Błędy:
`raw-zone/stockdata_errors/!{firehose:error-output-type}/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}/`
- e. Rozmiar bufora = 1MB
- f. Częstotliwość zrzucania danych do S3 = 60s
- g. Rola IAM - LabRole (użyj zmiennej)

Przykładowa definicja

```
resource "aws_kinesis_firehose_delivery_stream" "stock_delivery_stream"
{
  name =
  "firehose-${var.account_number}-${var.student_initials}-${var.student_index_no}"
  destination = "extended_s3"

  kinesis_source_configuration {
    kinesis_stream_arn = aws_kinesis_stream.cryptostock_stream.arn
    role_arn = var.lab_role_arn
  }

  extended_s3_configuration {
    role_arn = var.lab_role_arn
    bucket_arn = aws_s3_bucket.raw_bucket.arn
    buffering_size = 1
    buffering_interval = 60
    prefix =
    "raw-zone/stockdata/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}/"
    error_output_prefix =
    "${"raw-zone/stockdata_errors/!{firehose:error-output-type}/year=!{timestamp:yyyy}"}${"/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}"}/"
  }
}
```

6. Utwórz nowe obiekty - **terraform plan / apply**
7. Sprawdź czy nowy firehose stream jest aktywny (konsola AWS)

8. Upewnij się że generator generuje dane i po około 5-7 minutach zweryfikuj czy dane pojawiają się w bucket'cie (prefix - **raw-zone**)

Amazon S3 > data-lake-dev-100603781557-jk-12345 > raw-zone/ > stockdata/ > year=2021/ > month=04/ > day=17/ > hour=13/

hour=13/ Copy 5

Objects Properties

Objects (1)
Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Delete

< 1 >

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	firehose-dev-100603781557-jk-12345-1-2021-04-17-13-58-36-772fe83d-3d8c-4334-a2d0-013d6fe0f4a4	-	April 17, 2021, 15:59:37 (UTC+02:00)	66.2 KB	Standard

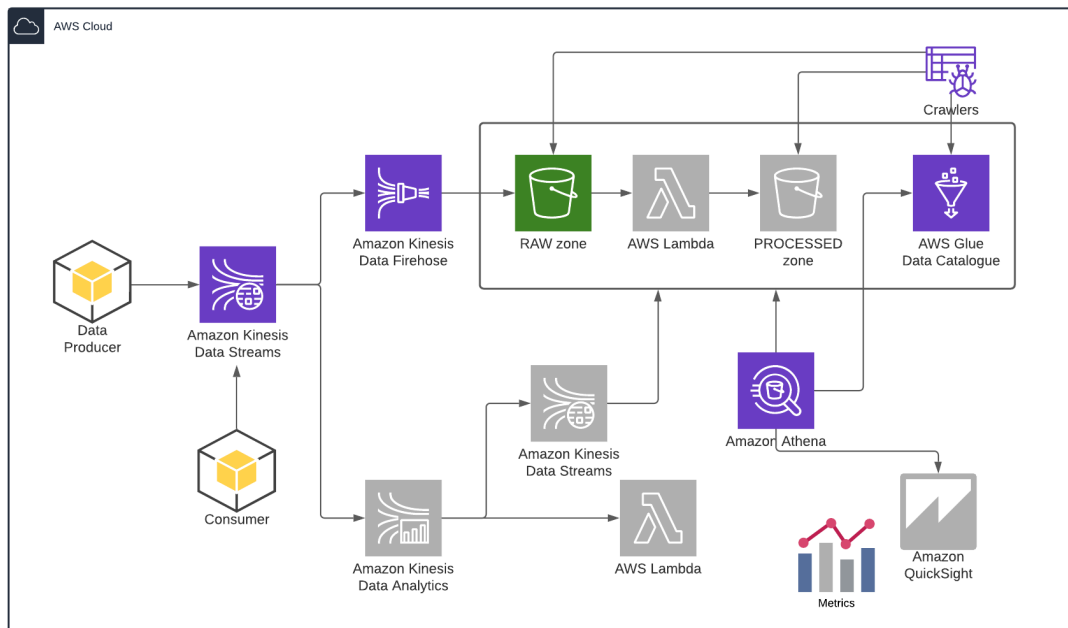
9. Pozostaw generator włączony i przejdź do ćwiczenia 3

Ćwiczenie 3

Rejestrowanie tabel w Glue Data Catalogue i aktualizowanie partycji.

W tym ćwiczeniu wykorzystamy:

- Glue Crawlers do automatycznego rejestrowania tabeli w katalogu Glue.
- Athena do wykonania zapytanie SQL zwracające największe oferty kupna / sprzedaży w przedziałach godzinowych



1. Utwórz nową bazę danych Glue w terraform. Stwórz plik `./labs/terraform/glue_dc.tf`

a. Opis obiektu bazy glue TF :

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/glue_catalog_database

b. Nazwa bazy danych (wzorzec):

`"datalake_raw_${var.account_number}_${var.student_initials}_${var.student_index_no}"`

Zwróć uwagę, że nazwa nie zawiera '-' tylko '_'. Powinna być zgodna z

<https://docs.aws.amazon.com/athena/latest/ug/glue-best-practices.html>

Przykładowa definicja

```
resource "aws_glue_catalog_database" "datalake_db_raw_zone" {
  name =
```

```
"datalake_raw_${var.account_number}_${var.student_initials}_${var.student_index_no}"
}
```

c. Uruchom terraform apply aby stworzyć bazę

```
$ terraform apply
...

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_glue_catalog_database.datalake_db_raw_zone will be created
+ resource "aws_glue_catalog_database" "datalake_db_raw_zone" {
  + arn          = (known after apply)
  + catalog_id  = (known after apply)
  + id          = (known after apply)
  + name        = "datalake_raw_100603781557_jk_12345"
}

Plan: 1 to add, 0 to change, 0 to destroy.

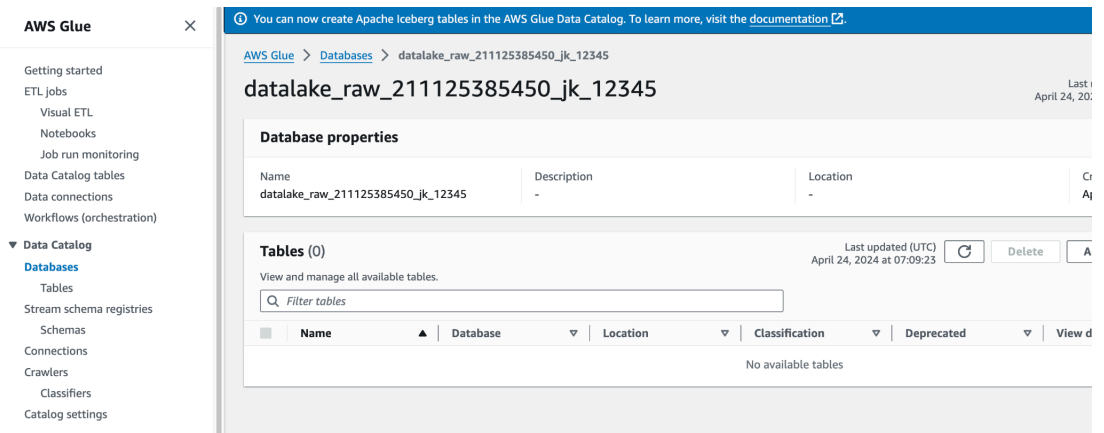
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

aws_glue_catalog_database.datalake_db_raw_zone: Creating...
aws_glue_catalog_database.datalake_db_raw_zone: Creation complete
after 1s [id=100603781557:datalake_raw_100603781557_jk_12345]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

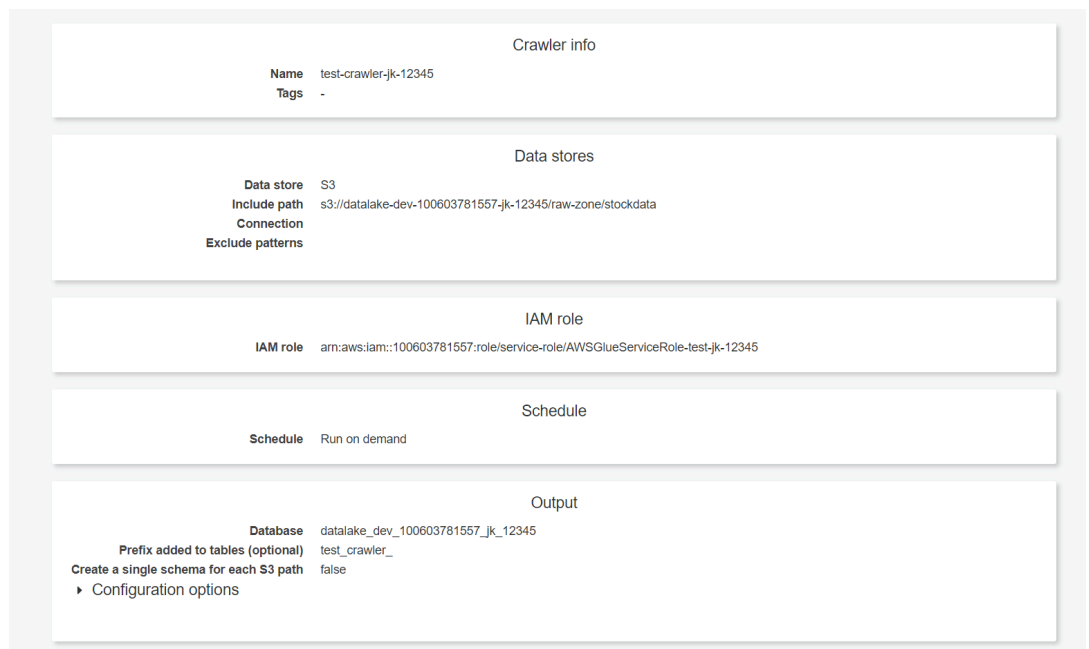

d. Zweryfikuj poprawność jej utworzenia w konsoli AWS



2. Uruchom ręcznie **Glue Crawler** (z konsoli AWS), który przeskanuje dane w prefisie do którego trafiają nasze dane i utworzy nową tabelę w twojej bazie danych w Glue, Wybierz następujące opcje :

- Nazwa crawlera - *test-crawler-{inicjaly}-{nr_indeksu}*
- Data Store - ścieżka S3 do danych zrzucanych przez firehose np. *s3://datalake-raw-100603781557-jk-12345/raw-zone/stockdata*
- Uprawnienia - wybierz istniejącą rolę LabRole
- Częstotliwość uruchamiania - on demand
- Wyniki powinny być zapisywane w Twojej bazie danych glue (utworzonej w p.1)
- Wybierz prefiks dla nowo tworzonych tabel - **test_crawler_**

Przykład konfiguracji:



g. Uruchom Crawler'a

AWS Glue
Data catalog
Databases
Tables
Connections
Crawlers
Classifiers
Schema registries
Schemas

Crawlers: A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Filter by tags and attributes

Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated
test-crawler-jk-12345		Starting		0 secs	0 secs	0

h. Po paru minutach powinieneś zobaczyć rezultaty - nowa tabela w Twojej bazie danych

AWS Glue
Data catalog
Databases
Tables
Connections
Crawlers
Classifiers
Schema registries
Schemas
Settings
ETL
AWS Glue Studio
Blueprints
Workflows
Jobs
ML Transforms
Triggers
Dev endpoints
Notebooks
Security
Security configurations
Tutorials
Add crawler
Explore table
Add job
Resources
What's new

Tables > test_crawler_stockdata

Last updated 17 Apr 2021 04:25 PM Table Version (Current version)

Partitions and indices View partitions Compare versions Edit schema

Name: test_crawler_stockdata
 Description:
 Database: datalake_dev_100603781557_jk_12345
 Classification: json
 Location: s3://datalake-dev-100603781557-jk-12345/aw-zone/stockdata/
 Connection:
 Deprecated: No
 Last updated: Sat Apr 17 16:25:52 GMT+0200 2021
 Input format: org.apache.hadoop.mapred.TextInputFormat
 Output format: org.apache.hadoop.hive.qjo.HiveIgnoreKeyTextOutputFormat
 Serde serialization lib: org.openx.data.jsonserde.JsonSerDe

Serde parameters: paths, dollar_amount, price, symbol, trans_id, transaction_ts, type
 sizeKey: 177697 objectCount: 26 UPDATED_BY_CRAWLER: test-crawler-jk-12345 CrawlerSchemaSerializerVersion: 1.0 recordCount: 13039 averageRecordSize: 135

Table properties: CrawlerSchemaDeserializerVersion: 1.0 compressionType: none typeOfData: file

Schema

Column name	Data type	Partition key	Comment
1 transaction_ts	int		
2 symbol	string		
3 price	double		
4 dollar_amount	double		
5 type	string		
6 trans_id	int		
7 year	string	Partition (0)	
8 month	string	Partition (1)	

Showing: 1 - 10 of 10

- Sprawdź jej schemat i partycje.
- Stwórz nowego Crawlera za pomocą kodu TF (*) dla chętnych. Dodaj właściwą definicję Glue Crawlera do pliku `./labs/glue.tf`:

```
resource "aws_glue_catalog_database" "datalake_db_raw_zone" {
  name =
  "datalake_raw_${var.account_number}_${var.student_initials}_${var.student_index_no}"
}

resource "aws_glue_catalog_database" "datalake_db_processed_zone" {
  name =
  "datalake_processed_${var.account_number}_${var.student_initials}_${var.student_index_no}"
}

resource "aws_glue_crawler" "glue_crawler_raw_zone" {
  database_name = aws_glue_catalog_database.datalake_db_raw_zone.name
  name =
  "gc-raw-${var.account_number}-${var.student_initials}-${var.student_index_no}"
}
```

```

ex_no}"

role = var.lab_role_arn
table_prefix = "crawler_"

s3_target {
  path = "s3://${aws_s3_bucket.raw_bucket.bucket}/raw-zone/stockdata/"
}

tags = merge(local.common_tags, )
}

```

Uruchom **terraform plan / apply** aby wdrożyć zmiany. Uruchom nowo utworzonego Crawlera i sprawdź czy nowa tabela z prefiksem crawler_ została utworzona.

3. Stwórz dedykowaną infrastrukturę dla serwisu **Athena**

Obiekty do stworzenia:

- nowy bucket dla wyników zapytań wraz z lifecycle_policy (tymczasowe wyniki - powinny być usunięte po jednym dniu automatycznie)
- dedykowan Athena Workgroup

https://registry.terraform.io/providers/hashicorp/aws/3.64.2/docs/resources/athena_workgroup z domyślną lokalizacją athena-results bucket, prefix output.

- a. Utwórz nowy plik **athena.tf** dla obiektów dedukowanych dla Athena.
- b. Dodaj nowy S3 bucket (podobnie jak w LAB1) do tego pliku.

Bucket powinien mieć również zdefiniowany lifecycle_policy i jego nazwa zaczynać się do athena-results (reszta podobnie jak w poprzednich bucketach)

Przykład:

```

resource "aws_s3_bucket" "athena_results" {
  bucket =
"athena-results-${var.account_number}-${var.student_initials}-${var.student_index_no}"
  force_destroy = true

tags = merge(local.common_tags, )

```

```

}

resource "aws_s3_bucket_lifecycle_configuration"
"athena_results_lifecycle" {
  bucket = aws_s3_bucket.athena_results.id

  rule {
    id = "standard-expiration"
    status = "Enabled"
    expiration {
      days=1
    }
  }
}

resource "aws_athena_workgroup" "athena_workgroup" {
  name = "development"

  configuration {
    enforce_workgroup_configuration = true

    result_configuration {
      output_location =
"s3://${aws_s3_bucket.athena_results.bucket}/output/"
    }
  }

  force_destroy = true
}

```

Zweryfikuj stworzone obiekty (S3 bucket i Athena workgroup)

The screenshot shows the AWS Management Console interface for Amazon Athena. On the left sidebar, the 'Administration' menu is expanded, and 'Workgroups' is highlighted with a red box. The main content area displays the 'Workgroups (2) info' page, which includes a table listing the configured workgroups.

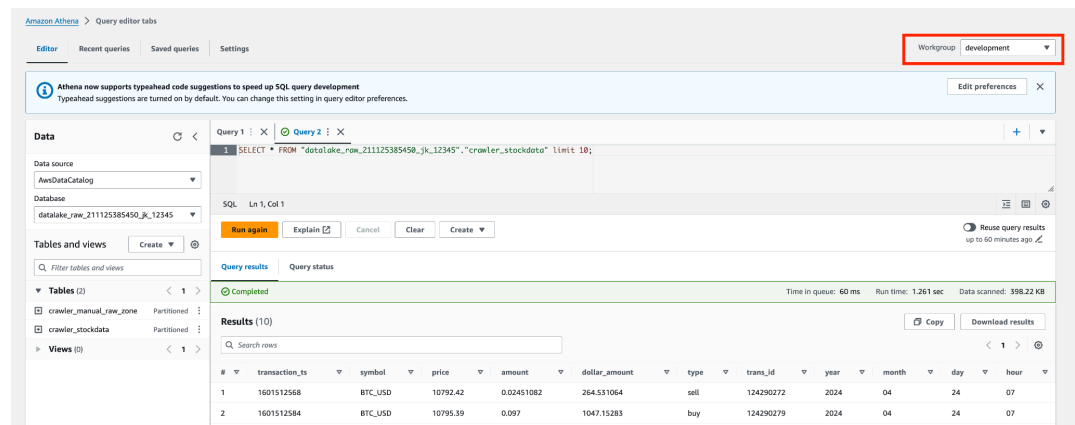
Name	Description	Analytics engine	Engine updates	Authentication	Create
development	-	Athena engine version 3	Automatic	IAM	2024-4
primary	-	Athena engine version 3	Automatic	IAM	2024-4

- Otwórz konsole AWS i serwis Athena. Jeśli pierwszy raz otwierasz tą usługę, konieczne będzie skonfigurowanie **Workgroup Primary lub wybrania nowo utworzonej workgroup - Development** aby wskazać miejsce składowania wyników.

Dla **primary** - wybierz odpowiedni bucket, dodaj prefix

Dla **development** workgroup - wszystko jest już skonfigurowane w TF!

- Sprawdź czy możesz odpytać tabelę stworzoną przez Crawlera



- Napisz zapytanie SQL, które z nowo utworzonej tabeli pobierze najwyższe wartości zleceń (BUY / SELL) w przedziałach godzinowych **per symbol** . Posortuj wyniki od najstarszych przedziałów do najnowszych.

Czas pobierz z atrybutu **transaction_ts**. Jest to informacja o faktycznym czasie złożenia zlecenia buy/sell. Format czasu unix epoch, który możesz przekonwertować na czytelną dla ludzi postać za pomocą funkcji **from_unixtime()** oraz **date_format()**.

Przykład wykorzystania funkcji **from_unixtime()** i **date_format()** :

```
SELECT date_format(from_unixtime(1618669914), '%Y-%m-%dT%H:%i:%sZ')
-----
2021-04-17T14:31:54Z
```

- Porównaj Twoje “buckety godzinowe” (nazwij tą kolumnę HourlyBucket, format **%Y-%m-%dT%H**) otrzymane z konwersji **transaction_ts** z bucketami godzinowymi wynikającymi z partycjonowania danych (uwaga ! generator mógł wygenerować te same event kilka razy - rolling strategy).

Widać tutaj podstawowy problem związany z czasem zasilania data lake a rzeczywistym czasem ich generowaniem.

Informacje o czasie zupełnie nie pasują do naszych partycji.

Partycjonowanie następuje zgodnie z czasem nadejścia danych do systemu (Kinesis Data Stream używa **ApproximateArrivalTimestamp**)

Fragment wyniku kwerendy - porównaj twój rezultat dla pierwszego okresu **2020-10-01 godzina 00** (jeśli twój wynik się różni - prawdopodobnie nie wszystkie dane zostały wygenerowane lub usługa firehose została utworzona zbyt późno - po prostu zrestartuj generator i upewnij się że przynajmniej 600 próbek zostało wygenerowanych i zrzuconych do S3 (pierwsza godzina danych)) :

HourlyBucket	rnk	transaction_ts	symbol	price	amount	dollar_amount	type	trans_id	year	month	day	hour
2020-10-01T00	1	1601510472	BTC_USD	10792.34	2.12543803	22938.44987	buy	124289114	2021	04	17	17
2020-10-01T00	1	1601513129	BTC_USD	10795.0	1.5742	16993.489	sell	124290529	2021	04	17	17
2020-10-01T00	1	1601512236	ETH_USD	359.38	30.13418456	10829.62325	buy	124290121	2021	04	17	17
2020-10-01T00	1	1601511323	ETH_USD	360.55	51.05	18406.0775	sell	124289655	2021	04	17	17
2020-10-01T00	1	1601510672	LTC_USD	46.4	66.9212	3105.14368	buy	124289316	2021	04	17	17
2020-10-01T00	1	1601512784	LTC_USD	46.29	60.0	2777.4	sell	124290378	2021	04	17	17
2020-10-01T00	1	1601512780	LTC_USD	46.29	60.0	2777.4	sell	124290372	2021	04	17	17
2020-10-01T00	1	1601512797	LTC_USD	46.29	60.0	2777.4	sell	124290385	2021	04	17	17
2020-10-01T01	1	1601515734	BTC_USD	10809.5	1.90163679	20555.74288	buy	124291476	2021	04	17	17
2020-10-01T01	1	1601514738	BTC_USD	10802.54	1.29789589	14020.57227	sell	124291119	2021	04	17	17
2020-10-01T01	1	1601517314	ETH_USD	361.26	56.0	20230.56	buy	124292135	2021	04	17	17
2020-10-01T01	1	1601515243	ETH_USD	359.51	23.33	8387.3683	sell	124291292	2021	04	17	17
2020-10-01T01	1	1601516305	LTC_USD	46.55	62.81001196	2923.806057	buy	124291675	2021	04	17	17
2020-10-01T01	1	1601514140	LTC_USD	46.32	60.0	2779.2	sell	124290926	2021	04	17	17
2020-10-01T02	1	1601520133	BTC_USD	10840.54	3.48043669	37729.81316	buy	124293556	2021	04	17	17

W następnym ćwiczeniu będziemy konwertowali te dane do formatu **parquet** i zmieniali partycjonowania ze względu na datę i czas wystąpienia zdarzenia (producer timestamp).

Zapisz swoją kwerendę SQL w pliku **./labs/sql_queries.sql**

- Zatrzymaj generator danych i usuń cały stack poleceniem **terraform destroy** (uruchom w katalogu w którym masz utworzone obiekty terraform)

będzie wykonywana. Funkcja lambda to nic innego jak kawałek kodu wykonywanego na docker image.

- d. Najpierw stworzymy w TF dodatkową warstwę (`aws_lambda_layer_version`) z pakietem `awswrangler`. Sprawdź w dokumentacji TF i AWS jakie możliwości i ograniczenia mają dodatkowe warstwy (rozmiar, ilość). Zwróć uwagę, że mamy możliwość również tworzenia własnych obrazów na których możemy uruchamiać swoją funkcję (nowość ogłoszona na ReInvent 2020)

Utwórz plik `./labs/terraform/lambda.tf` i zdefiniuj obiekt jak poniżej :

```
resource "aws_lambda_layer_version" "aws_wrangler" {
  filename          = "../lambda/awswrangler-layer-2.7.0-py3.8.zip"
  layer_name       =
  "aws_wrangler_${var.account_number}_${var.student_initials}_${var.student_index_no}"
  source_code_hash =
  "${filebase64sha256("../lambda/awswrangler-layer-2.7.0-py3.8.zip")}"
  compatible_runtimes = ["python3.8"]
}
```

Sprawdź w dokumentacji do czego służą poszczególne atrybuty i funkcja `filebase64sha256`.

- e. Funkcja lambda będzie musiała być uruchamiana z odpowiednimi uprawnieniami - tutaj znów użyjemy roli **LabRole**.
- f. Kolejnym krokiem będzie zdefiniowanie właściwej funkcji lambda. Dopisz do pliku `./labs/terraform/lambda.tf` kolejny obiekt. Przykładowa definicja :

```
resource "aws_lambda_function" "etl_post_processing" {

  function_name =
  "etl-post-processing-${var.account_number}-${var.student_initials}-${var.student_index_no}"
  filename      = "../lambda/lambda_definition.zip"
  handler       = "lambda_definition.etl_function"
  runtime       = "python3.8"
  role          = var.lab_role_arn
  timeout       = 300
  memory_size   = 512
}
```



```

source_code_hash= filebase64sha256("../lambda/lambda_definition.zip")
layers              = ["${aws_lambda_layer_version.aws_wrangler.arn}"]
}

```

- g. Ostatnim elementem potrzebnym do wyzwolenie funkcji, są notyfikacje S3 oraz odpowiednie uprawnienia bucketu do wyzwalania funkcji lambda. Utwórz w pliku `./labs/terraform/lambda.tf` te obiekty. Sprawdź szczegółowo w dokumentacji do czego służą.

Przykładowe definicje:

```

resource "aws_lambda_permission" "allow_bucket" {
  statement_id = "AllowExecutionFromS3Bucket"
  action       = "lambda:InvokeFunction"
  function_name = aws_lambda_function.etl_post_processing.arn
  principal    = "s3.amazonaws.com"
  source_arn   = aws_s3_bucket.raw_bucket.arn
}

resource "aws_s3_bucket_notification" "trigger_etl_lambda" {
  bucket = aws_s3_bucket.raw_bucket.id

  lambda_function {
    lambda_function_arn = aws_lambda_function.etl_post_processing.arn
    events               = ["s3:ObjectCreated:*"]
    filter_prefix        = "raw-zone/"
  }
}

depends_on = [aws_lambda_permission.allow_bucket]
}

```

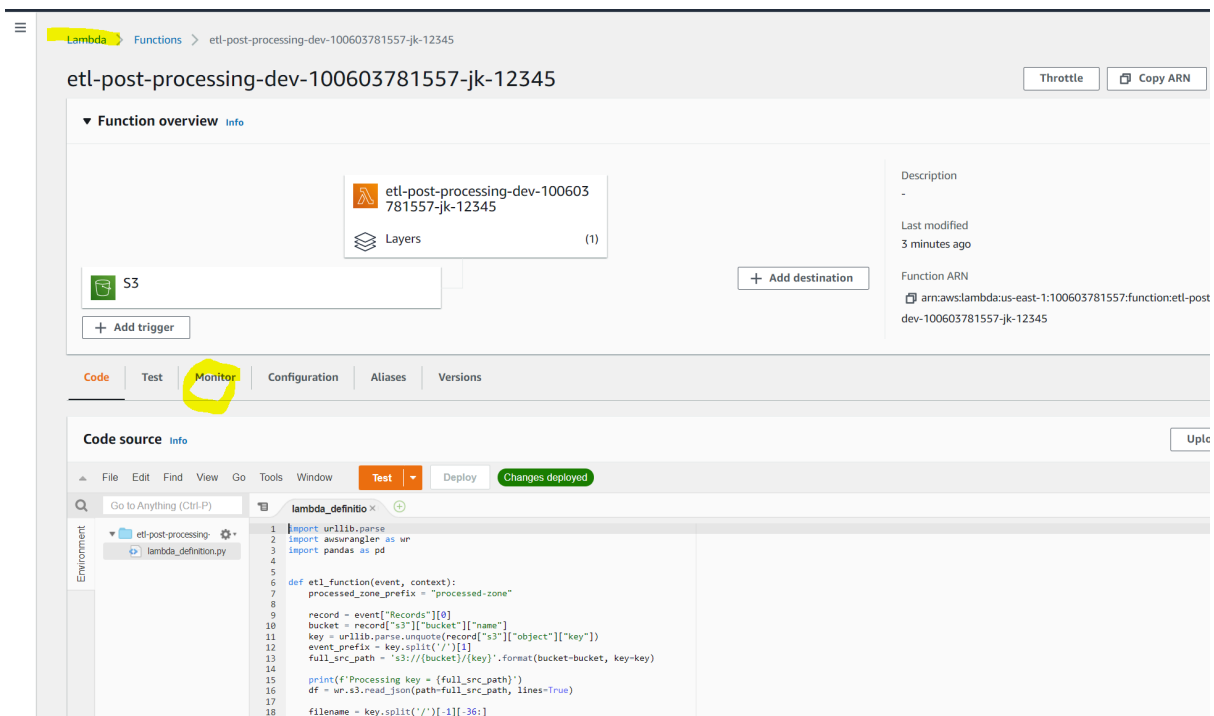
2. W tym momencie możemy wdrożyć zmiany. Uruchom **terraform apply**.
3. Po utworzeniu wszystkich obiektów **wystartuj ponownie generator danych** - odczekaj minutę.
 - a. Sprawdź czy dane są automatycznie przenoszone do prefixu `processed-zone` (dotyczyć to będzie wszystkich nowych plików zrzucanych do S3 przez



Firehose).

Zwróć uwagę, że funkcja Lambda, zmienia także schemat partycjonowania. Nadrzedną partycją jest informacja o typie symbolu (notowania). Dzięki temu nasze zapytania będą znacznie bardziej wydajne w kwerendach dotyczących konkretnych symboli (eliminacja partycji). Sprawdź w jakich datach (y/m/d/h) transakcje zostały wygenerowane.

- b. Sprawdź w konsoli AWS jak wygląda funkcja lambda i czy faktycznie jest wywoływana (metryki i logi pojawiają się z pewnym opóźnieniem).



4. Utwórz dodatkową tabelę w Glue. Możesz stworzyć / zaktualizować istniejące Crawlera lub zarejestrować tabelę z poziomu Athena
 - a. Przykład jak tworzyć table z poziomu Athena (SQL DDL) - dostosuj lokalizację S3 i nazwę bazy danych :

```
CREATE EXTERNAL TABLE <twoj raw db name>.processed_stockdata(  
    transaction_date timestamp,  
    price double,  
    amount double,  
    dollar_amount double,  
    type string,  
    trans_id bigint)  
PARTITIONED BY (  

```

```

symbol string,
year integer,
month integer,
day integer,
hour integer
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://<twoj raw zone bucket>/processed-zone/stockdata/';

```

Zostanie utworzona nowa tabela w Glue DC. Na Razie sama definicja bez partycji (sprawdź w Glue DC).

Aby odświeżyć partycje wykonaj poniższe polecenie

```
MSCK REPAIR TABLE processed_stockdata;
```

Zostanie przeskanowana lokalizacja główna S3 podana w parametrach tabeli. Automatycznie zostaną dodane wszystkie brakujące partycje. Spróbuj odpytać tabelę.

Zwróć uwagę, że wykorzystując atrybuty partycjonowania, ograniczasz ilość skanowanych danych. Dodatkowo wybierając pojedyncze atrybuty również ograniczasz koszty (ilość skanowanych danych).

W tym momencie RAW zone i Processed-zone nie są w pełni zsynchronizowane. Funkcja Lambda jest wyzwalana tylko dla nowo pojawiających się plików.

Spróbuj zaproponować sposób synchronizacji danych (weź pod uwagę idempotentność).

5. Wykonaj porównanie strefy RAW z PROCESSED (reconciliation).
 - a. Wyczyść całe środowisko (**terraform destroy & apply**) i rozpocznij generowanie danych od początku.
Uruchom generator z opcją single run **-r (single run)**

```
$ python generator.py -k cryptostock-211125385450-jk-12345 -r
```

Poczekaj aż przynajmniej dwie pierwsze godziny z transakcjami zostaną

wysłane i zrzucone do RAW zone.

Uruchom stworzonego crawlera i utwórz raz jeszcze tabelę w strefie processed (odśwież partycje).

- b. Porównaj pierwszy bucket godzinowy w obu warstwach lake'a

```
SELECT count(*)
FROM "datalake_raw_100603781557_jk_12345"."crawler_stockdata"
WHERE symbol = 'BTC_USD' and
date_format(from_unixtime(transaction_ts), '%Y-%m-%dT%H') =
'2020-10-01T00';

SELECT count(*)
FROM "datalake_raw_100603781557_jk_12345"."processed_stockdata"
WHERE HOUR=0 and DAY=1 and MONTH=10 and YEAR=2020
and symbol = 'BTC_USD';
```

- c. Powinieneś otrzymać te same ilości rekordów w obu warstwach (337 sztuk dla pary BTC_USD)

6. Zadanie do samodzielnego wykonania

Obecnie dane są przeliczane pomiędzy prefixami tego samego bucketu. Nasze główne założenie jest takie, aby dane przetworzone były składowane w osobnym bucketcie.

Zaproponuj rozwiązanie w którym funkcja lambda będzie sparametryzowana i wrzucała dane do bucketu **Processed**.

Rozważ gdzie przechowywać parametry ? Czy w funkcji Lambda?

Jednym z rozwiązań może być serwis parameter store

[aws ssm parameter | Resources | hashicorp/aws | Terraform | Terraform Registry](#)

Stwórz osobną bazę danych (datalake_processed.....) dla nowej tabeli.

Stwórz osobny proces do katalogowania danych w tej bazie (bucket processed).