Przetwarzanie Danych w Chmurze Publicznej
Laboratorium

Przygotowanie środowiska

1. Instalacja oprogramowania. Pobierz i zainstaluj zgodnie z instrukcją dla Twojego systemu operacyjnego. Rekomendowane oprogramowanie:
    a. **PyCharm** - https://www.jetbrains.com/pycharm/download/
    b. **Anaconda** (będziemy używali Python 3.8) - https://www.anaconda.com/products/individual#Downloads
    c. **Terraform** (minimum w wersji 0.14)
    Pobierz : https://www.terraform.io/downloads.html właściwy dla Twojego OS.
    Zainstaluj zgodnie z instrukcją :
    https://learn.hashicorp.com/tutorials/terraform/install-cli?in=terraform/aws-get-started

2. Konfiguracja środowiska Python + AWS
    a. Sprawdź poprawność instalacji wpisując w cmdline / bash (przykład - u Ciebie wersja może się nieco różnić na +)

```
$ terraform --version
Terraform v0.14.8
```

    b. Utwórz nowe środowisko Python (rekomendowana conda, ale może też być zwykły virtualenv)

```
$ conda create -n uam_cloud_dp python=3.8
$ conda activate uam_cloud_dp


(uam_cloud_dp)$
```

c. Sklonuj repozytorium kodu z laboratorium i przykładami

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

d. Zainstaluj wymagane pakiety python

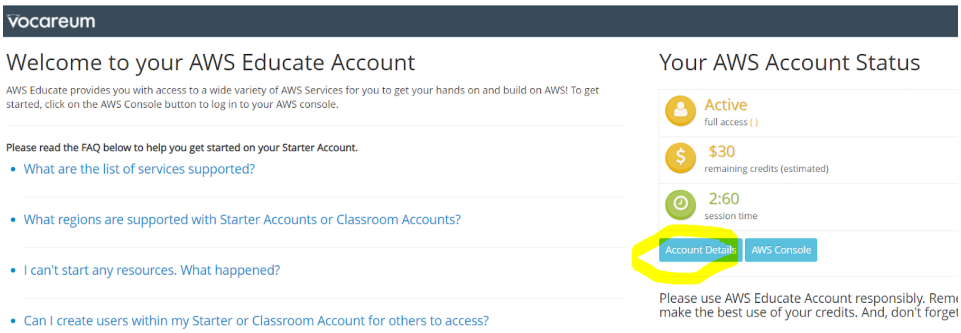
```
$ cd przetwarzanie_danych_w_chmurze
$ pip install -r ./labs/requirements.txt
```

e. Sprawdź czy **awscli** jest zainstalowane poprawnie

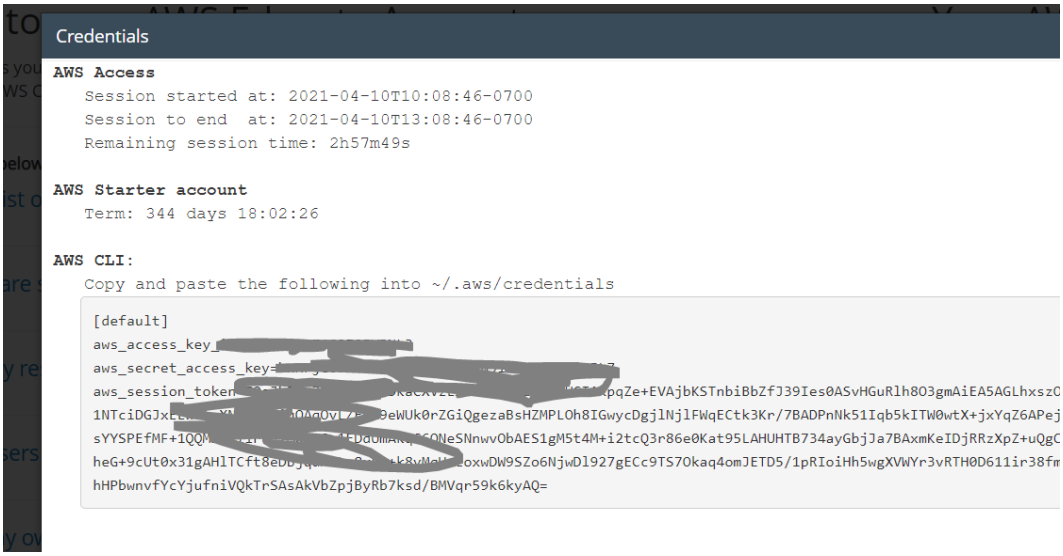
```
$ aws --version
aws-cli/1.19.33 Python/3.8.8 Windows/10 botocore/1.20.33
```

3. Konfiguracja konta AWS
   a. Jeśli posiadasz swoje własne konto - możesz go użyć. Wszystkie laboratoria nie będą kosztować więcej niż **3$** (szczególna uwaga na usługi Kinesis Analytics i Kinesis Data Stream, nie pozostawiaj ich aktywnych) - w przypadku własnego konta - przejdź do podpunktu *e*.
   b. Zaloguj się do AWS Educate : [Login to AWS Educate]
   c. Utwórz swoje AWS Starter Account

d. Zostaniesz przekierowany do strony zawierającej detale niezbędne do dostępu programistycznego oraz konsoli AWS



e. Skopiuj całość danych do autentykacji (Account Details) i umieść je w pliku **~/.aws/credentials** w profilu **[default].** Więcej na temat konfiguracji środowiska do pracy z AWS możesz znaleźć tutaj https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html

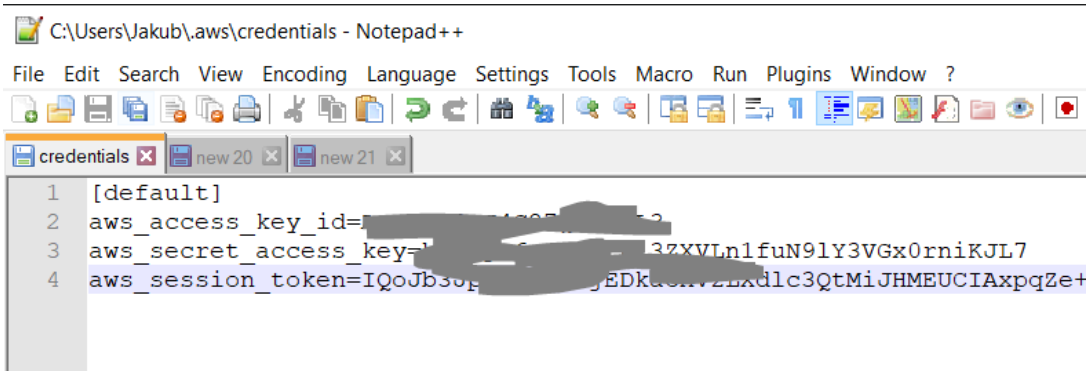


f. Jeśli nie masz tego pliku, możesz skonfigurować konto za pomocą aws configure (hasła narazie wpisz dowolne, region us-east-1, output format - just hit enter):

```
$ aws configure
AWS Access Key ID [****************H5RD]: a
AWS Secret Access Key [****************rRn3]: b
Default region name [us-east-1]:
Default output format [None]:
```

Teraz wystarczy wyedytować **~/.aws/credentials** i skopiować dane z poprawnymi kluczami Twojego konta AWS (session token tylko dla sesji

tymczasowych)



g. Uruchom Python REPL i sprawdź czy możesz odwołać się do konta z sukcesem - przykład:

```
$ python
Python 3.8.8 (default, Feb 24 2021, 15:54:32) [MSC v.1928 64 bit
(AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto3
>>> boto3.client('sts').get_caller_identity()
{'UserId':
'AROARO3DT4G2WTXDONN24:user1334844=jkasprzak@wmi.amu.edu.pl',
'Account': '100603781557', 'Arn':
'arn:aws:sts::100603781557:assumed-role/vocstartsoft/user1334844=jkaspr
zak@wmi.amu.edu.pl',....
>>>
```

Terraform 101

Wszystkie obiekty, które będziemy tworzyć w AWS można wykonać za pomocą konsoli lub kodu. Najpopularniejszymi rozwiązaniami **IaaC** (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 najważniejsz 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. Skopiuj podstawowe pliki inicjalizacyjne TF z **./labs/terraform/starter_files** do **./labs/terraform/**
   Cały Twój kod TF będzie znajdował się w tej lokalizacji i wszędzie w zadaniach gdy będziemy odwoływali się do kodu terraform - będzie to właśnie w tym folderze.

   Pliki do skopiowania :

   ./labs/terraform/starter_files/provider.tf

```
terraform {
 required_providers {
   aws = {
     source = "hashicorp/aws"
     version = "~> 3.27"
   }
 }
}

provider "aws" {
 profile = "default"
 region = var.region
}
```

   ./labs/terraform/starter_files/main.tf

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

./labs/terraform/starter_files/terraform.tfvars

```
account_number=100603781557
student_initials="jk"
student_index_no = "12345"
```

./labs/terraform/starter_files/variables.tf

```
variable "account_number" {
 description = "Account number"
 type = number
}

variable "region" {
 description = "Region name - must be N-Virginia us-east-1"
 type = string
 default = "us-east-1"
}

variable "environment" {
 description = "Environment name"
 type = string
 default = "dev"
}

variable "student_initials" {
 description = "letters of first and last names"
 type = string
}

variable "student_index_no" {
 description = "Index no"
 type = string
}
```

2. Zaktualizuj plik **./labs/terraform/terraform.tfvars** swoimi danymi (numer konta AWS, numer indeksu i inicjały) np:

```
account_number=100603781557
```

```
student_initials="jk"
student_index_no = "12345"
```

3. Zainicjuj plik stanu Terraform poleceniem ***terraform init*** (od tego momentu wszystkie zmiany będą dokonywane na Twoim koncie AWS). Nigdy nie modyfikuj pliku stanu ręcznie!

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 3.27"...
- Installing hashicorp/aws v3.36.0...
- Installed hashicorp/aws v3.36.0 (signed by HashiCorp)
...
Terraform has been successfully initialized!
...
```

4. Otwórz repozytorium w PyCharm, powinieneś mieć strukturę jak poniżej

```
<your repo>
├──jupyter
├──labs
│   ├──data_generator
│   ├──lambda
│   └──terraform
│         │   main.tf              <
│         │   provider.tf          <
│         │   terraform.tfvars     <
│         │   variables.tf         <
│         ├──solutions
│         └──starter_files
│
├──ppt
├──testing-stack
└──tests
```

5. Masz już skonfigurowane wszystkie narzędzia - gratulacje ! Możemy zaczynać !

Przetwarzanie Danych w chmurze publicznej **Laboratorium**

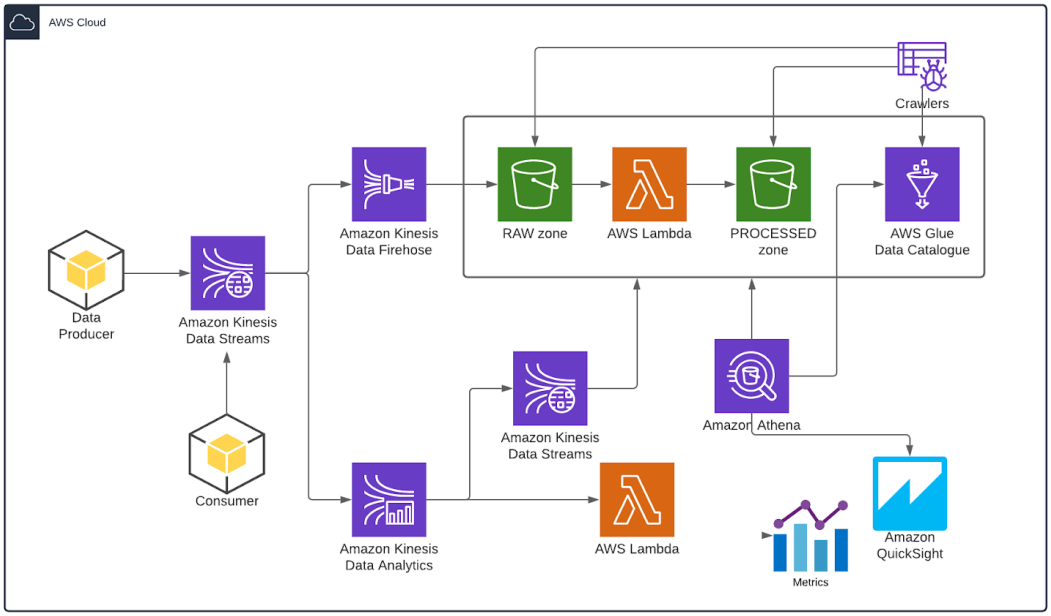
Scenariusz Laboratorium

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 zone a Processed zone z wykorzystaniem funkcji Lambda.

Przeprowadzimy również analitykę *real-time* z wykorzystaniem Kinesis Analytics. Zarejestrujemy tabele w Glue Data Catalogue aby móc wykonywać zapytania ad-hoc w Athena (SQL).

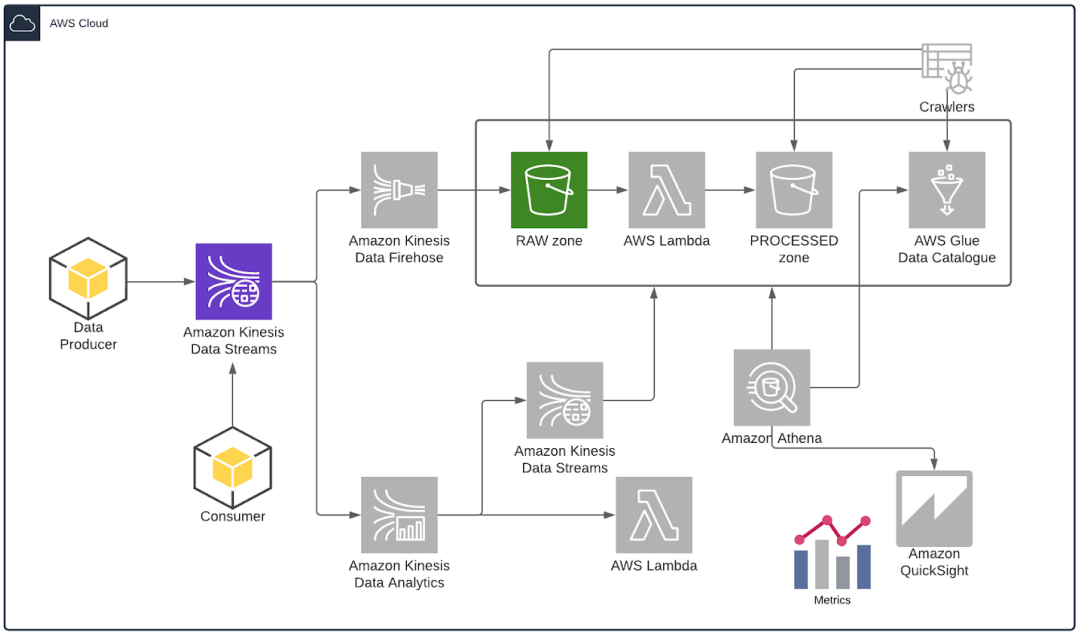
Do zbudowania systemu użyjemy następujących usług serverless :

- centralny broker wiadomości Kinesis Data Stream
- zasilanie data lake Kinesis Delivery Stream
- analitykę real-time Kinesis Analytics
- Glue Data Catalogue jako Hive Metastore do przechowywanie metadanych (tabele, partycje)
- silnik przetwarzania danych Presto udostępniający interfejs zgodny z ANSI SQL - usługa Athena (analityka 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 dwa pierwsze, podstawowe elementy infrastruktury do ładowania danych - **S3 bucket i Kinesis Data Stream**. Poznasz podstawowe zasady tworzenia obiektów w Terraform.



1. Utwórz nowy plik TF **./labs/terraform/s3.tf** zawierający definicję S3 bucketu w którym będziemy przechowywali wszystkie dane (uproszczenie - jeden bucket dla RAW zone & PROCESSED zone - separacja na prefixach obiektów).
    a. Opis struktury S3 w TF : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket
    b. Stwórz bucket opisany następującymi parametrami :
        i. Nazwa bucketu powinna składać się z : *datalake-${var.environment}-${var.account_number}-${var.student_initials}-${var.student_initials}*

           Np: ***datalake-dev-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 **./labs/terraform/s3.tf**

```
resource "aws_s3_bucket" "main_dl_bucket" {
 bucket =
"datalake-${var.environment}-${var.account_number}-${var.student_initia
ls}-${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/**)

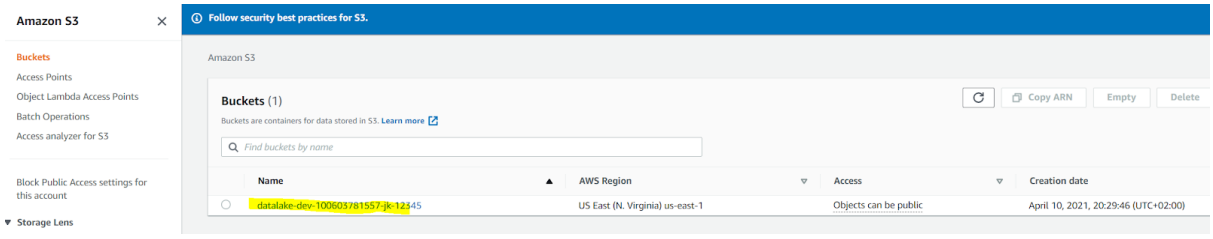
```
$ terraform plan

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


  Enter a value: yes


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


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



Bucket został stworzony.

2. 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-${var.environment}-${var.account_number}-${var.student_initials}-${var.student_initials}*

   Np: ***cryptostock-dev-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"]

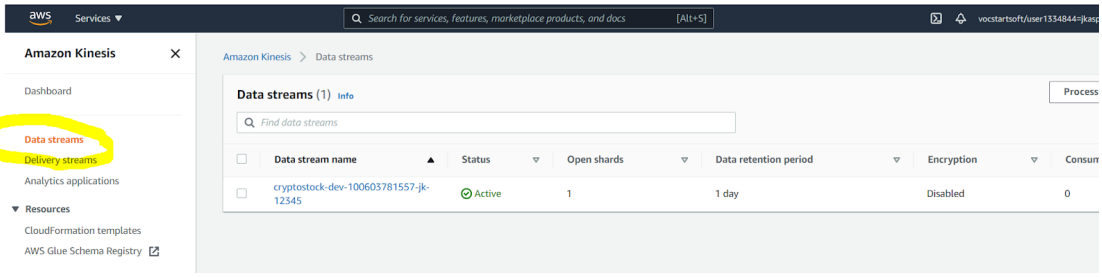
c. Przykładowa definicja

```
resource "aws_kinesis_stream" "cryptostock_stream" {
 name             = "cryptostock-${var.environment}-${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

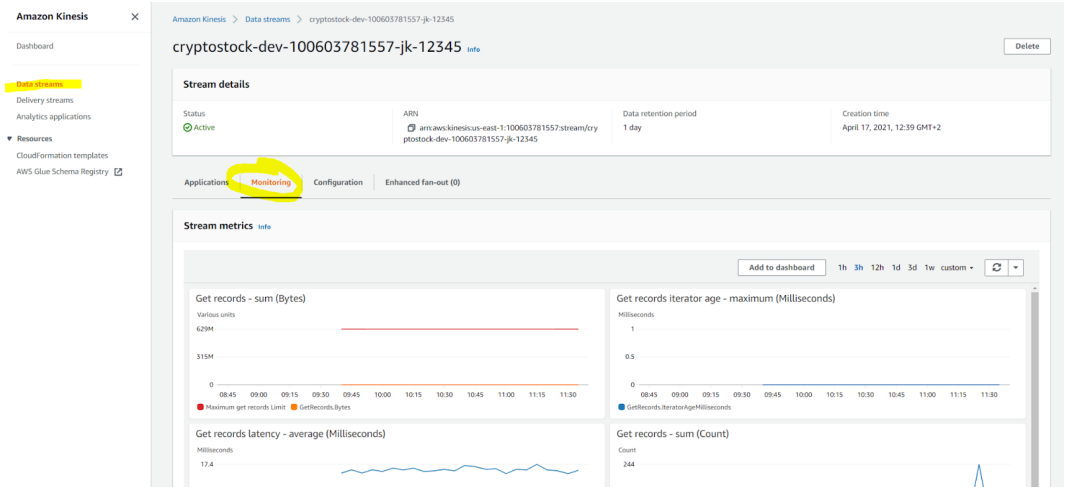


3. Wysyłanie danych do Kinesis DS
   a. Przejrzyj kod prostego generatora danych **./labs/data_generator/generator.py**. Zwróć uwagę na sposób w jaki dane są wysyłane do Kinesis - wykorzystujemy tu SDK (API), które charakteryzuje się niską wydajnością. Iterujemy po danych z pliku i wywołujemy metodę put_record.

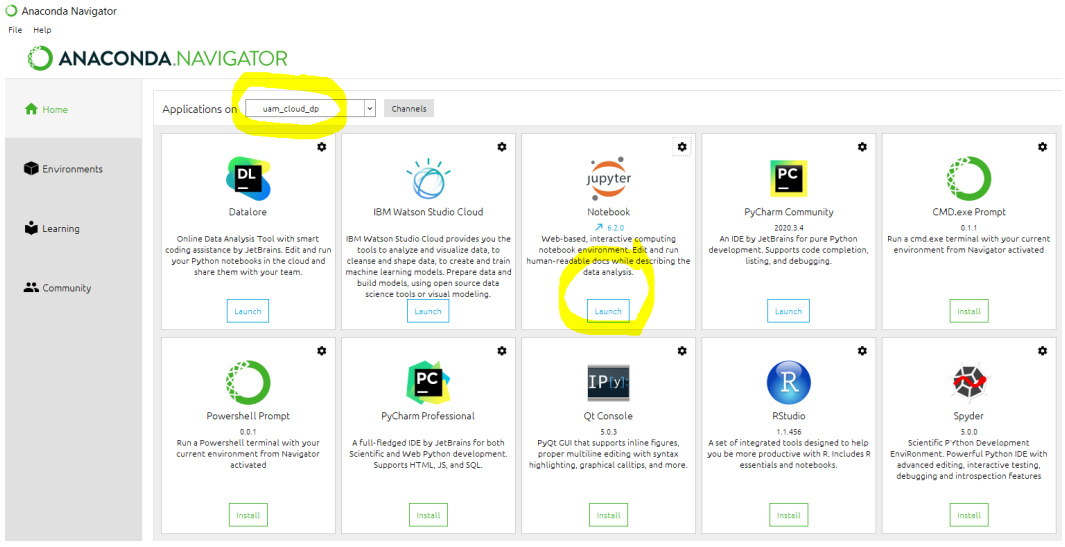
      (*) W ramach dodatkowego ćwiczenia (dla ambitnych :)) możesz udoskonalić generator, przepisując go na put_records lub generować rekordy wielowątkowo / ew asynchronicznie aby zwiększyć wydajność.
   b. Uruchom generator w nowym oknie linii poleceń (będzie nam potrzebny aby cały czas wysyłał dane).
      Użyj parametru -k aby przekazać nazwę kinesis data stream'a

```
$ pwd
<path to the code repo>
$ python ./labs/data_generator/generator.py -k
cryptostock-dev-100603781557-jk-12345

[2021-04-17 13:43:43,761] {generator.py:119} INFO - Starting Simple
Kinesis Producer (replaying stock data)
[2021-04-17 13:43:43,776] {credentials.py:1222} INFO - Found
credentials in shared credentials file: ~/.aws/credentials
[2021-04-17 13:43:43,913] {generator.py:102} INFO - start replaying
for the 1 time
[2021-04-17 13:43:44,429] {generator.py:53} INFO - Msg sent to shard
shardId-000000000000 seq no
49617413061606993399598855641493748021734819860454572034
[2021-04-17 13:43:44,548] {generator.py:53} INFO - Msg sent to shard
shardId-000000000000 seq no
49617413061606993399598855641494956947554434489629278210
```

c. Przejdź do Konsoli AWS i zobacz jak wyglądają metryki KDS, czy dane faktycznie docierają do sharda (może minąć ok minuta zanim metryki będą zaktualizowane)



d. Uruchom Jupyter notebook (Anaconda - zainstaluj jeśli potrzeba)



./Jupyter/UAM_Lab_1_reading_from_kinesis_stream.ipynb i przeczytaj dane ze streama używając boto3. Zwróć uwagę na sposób czytania danych (iteratory)

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

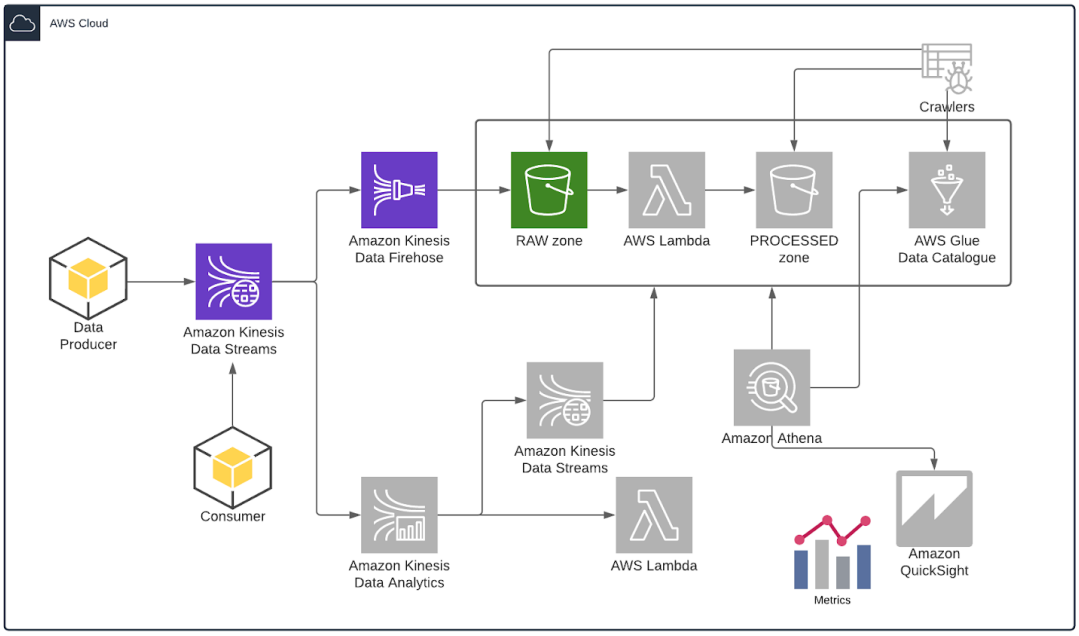
```
$ terraform destroy

...
```

```
Plan: 0 to add, 0 to change, 2 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
```

5. 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

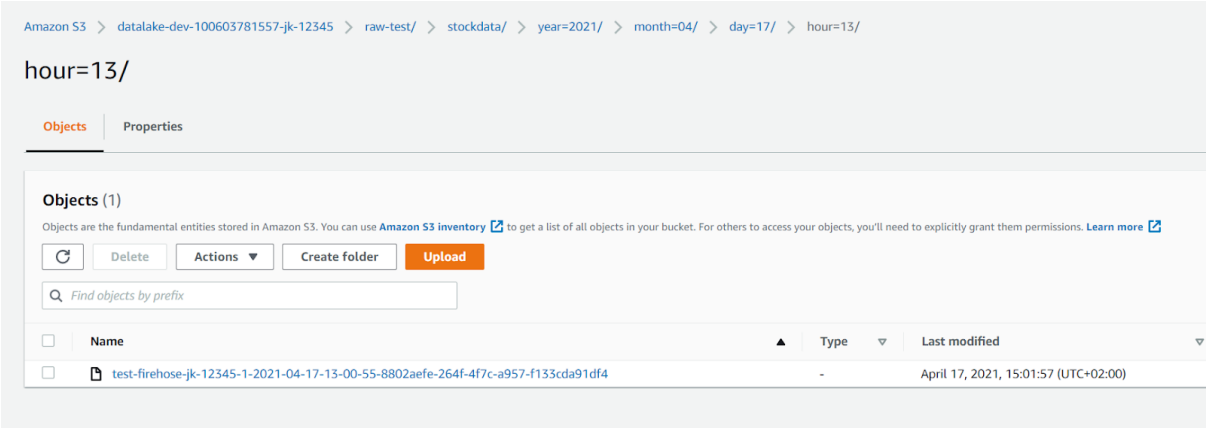


1. Utwórz obiekty zdefiniowane w ćwiczeniu 1 (*terraform **apply***) S3 bucket + Kinesis Data Stream - zostały one usunięte w ostatnim punkcie.
2. Utwórz nowy Kinesis **Delivery Stream** z poziomu konsoli AWS, przejrzyj możliwe opcje.

   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 minutę i sprawdź czy dane są zapisywane w S3 z odpowiednim prefixem.



5. Do utworzenie obiektu Kinesis Delivery Stream za pomocą TF, będziemy potrzebowali dodatkowo zdefiniowania roli i polityk IAM. Utwórz nowy plik **./labs/terraform/iam.tf** zawierający definicję nowej roli IAM dla **Kinesis Firehose** wraz z politykami (uprawnieniami)**.**

   Opis obiektów roli i polityk znajdziesz tu :
   - rola IAM -
   https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/iam_role
   - polityki IAM
   https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/iam_policy

   Potrzebne będą następujące uprawnienia (definiowane w iam_policy) :
   a. Możliwość zapisu do Twojego bucketu S3
   b. Czytanie z Kinesis Data Stream
   c. Zapisywanie logów CloudWatch
   d. Przykładowa definicja - zwróć uwagę w jaki sposób odwołujemy się do istniejących obiektów (S3, KDS).

      Nazwa roli (wzorzec):
      "firehose-role-${var.environment}-${var.account_number}-${var.student_initials}-${var.student_index_no}"

      Nazwa polityki (wzorzec):
      "firehose-stream-policy-${var.environment}-${var.account_number}-${var.student_initials}-${var.student_index_no}"

Obiekt polityk powinien być podpięty pod rolę

```
resource "aws_iam_role" "firehose_stream_role" {
 name =
"firehose-role-${var.environment}-${var.account_number}-${var.student_
initials}-${var.student_index_no}"

 assume_role_policy = <<EOF
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Action": "sts:AssumeRole",
       "Principal": {
         "Service": "firehose.amazonaws.com"
       },
       "Effect": "Allow",
       "Sid": ""
     }
   ]
}
EOF
}

resource "aws_iam_role_policy" "firehose_stream_policy" {
 name =
"firehose-stream-policy-${var.environment}-${var.account_number}-${var
.student_initials}-${var.student_index_no}"
 role = aws_iam_role.firehose_stream_role.id

 policy = <<EOF
{
   "Version": "2012-10-17",
   "Statement": [
     {
           "Effect": "Allow",
           "Action": "kinesis:*",
           "Resource": "*"
```

```
      },
      {
          "Effect": "Allow",
          "Action": [
              "s3:AbortMultipartUpload",
              "s3:GetBucketLocation",
              "s3:GetObject",
              "s3:ListBucket",
              "s3:ListBucketMultipartUploads",
              "s3:PutObject"
          ],
          "Resource": [
            "${aws_s3_bucket.main_dl_bucket.arn}",
            "${aws_s3_bucket.main_dl_bucket.arn}/*"
          ]
      },
      {
          "Sid": "",
          "Effect": "Allow",
          "Action": [
              "logs:PutLogEvents"
          ],
          "Resource": [
"arn:aws:logs:${var.region}:${var.account_number}:log-group:/aws/kinesisfirehose/*"
          ]
      }
  ]
}
EOF
}
```

6. Teraz czas na obiekt Firehose. Utwórz nowy plik **./labs/terraform/kinesis_fh.tf** zawierający definicję **Kinesis Delivery Stream (Firehose)**
    a. Nazwa (wzorzec) : *"firehose-${var.environment}-${var.account_number}-${var.student_initials}-${var.student_index_no}"*

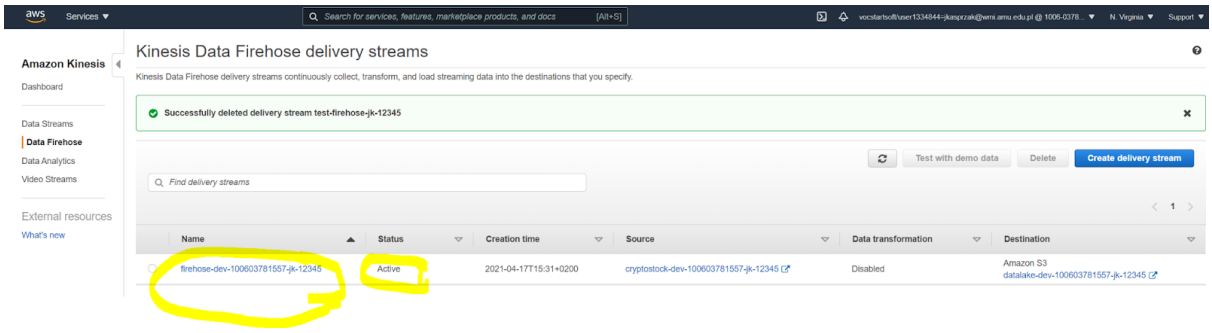
b. Źródło danych : Kinesis Data Stream (wybierz utworzony w p. 1 stream) - referencja terraform.
c. Brak transformacji
d. Miejsce docelowe S3 - Twój bucket utworzony w p.1 (referencja terraform)
   Wzorce prefiksów:
   i. Dane : *raw-zone/stockdata/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}/*
   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 - referencja do roli utworzonej w poprzednim punkcie tego ćwiczenia

Przykładowa definicja

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

 kinesis_source_configuration {
   kinesis_stream_arn = aws_kinesis_stream.cryptostock_stream.arn
   role_arn = aws_iam_role.firehose_stream_role.arn
 }

 extended_s3_configuration {
   role_arn = aws_iam_role.firehose_stream_role.arn
   bucket_arn = aws_s3_bucket.main_dl_bucket.arn
   buffer_size = 1
   buffer_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=!{timest
```

```
amp:yyyy}"}${
"/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}"}/"
 }
}
```

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



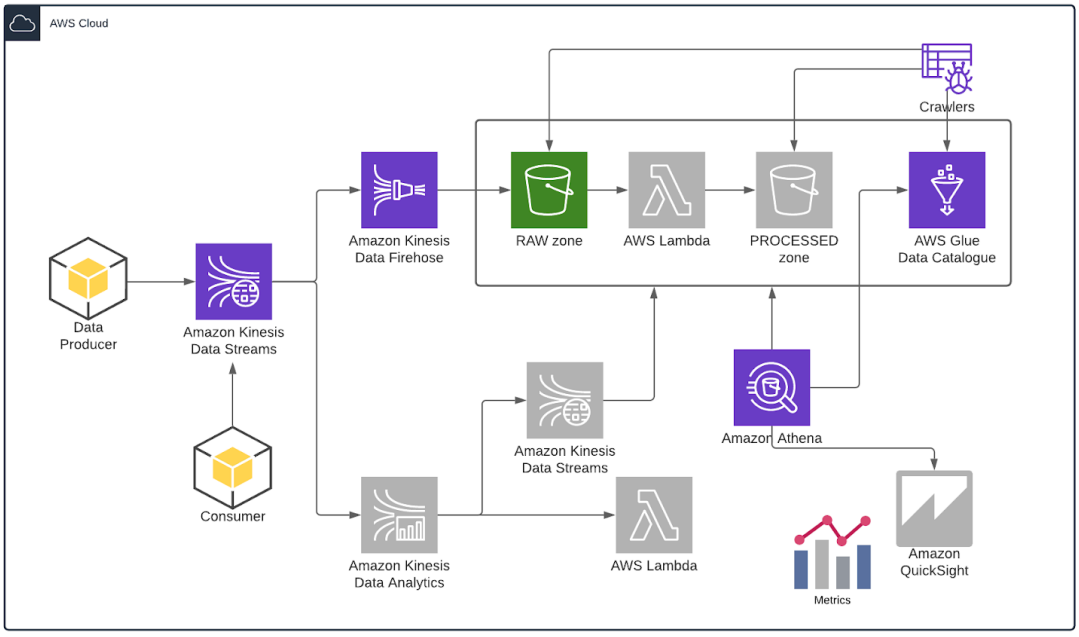
9. Upewnij się że generator generuje dane i po około minucie zweryfikuj czy dane pojawiają się w bucket'cie (prefix - **raw-zone**)

10. 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_${var.environment}_${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_${var.environment}_${var.account_number}_${var.student_initi
als}_${var.student_index_no}"
}
```

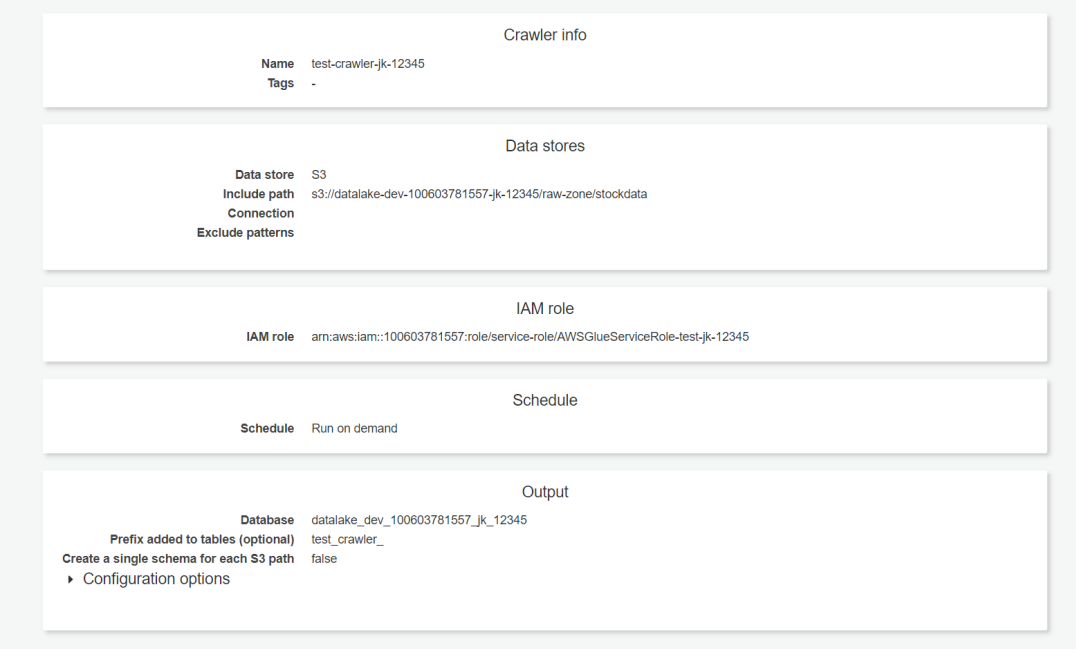
c. Uruchom terraform apply aby stworzyć bazę

```
$ 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_dev_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_dev_100603781557_jk_12345]

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

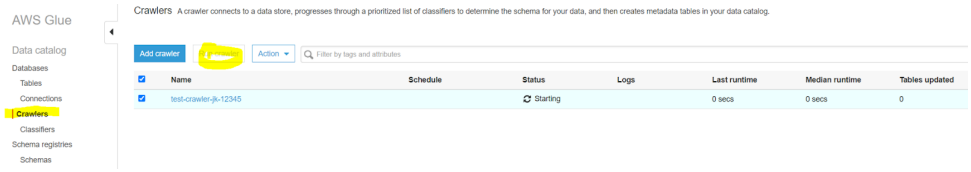
d. Zweryfikuj poprawność jej utworzenia w konsoli AWS

2. Uruchom ręcznie **Glue Crawler** (z konsoli AWS), który przeskanuje dane w prefixie do którego trafiają nasze dane i utworzy nową tabelę w twojej bazie danych w Glue, Wybierz następujące opcje :
   a. Nazwa crawlera - *test-crawler-{inicjaly}-{nr_indeksu}*
   b. Data Store - ścieżka S3 do danych zrzucanych przez firehose np. *s3://datalake-dev-100603781557-jk-12345/raw-zone/stockdata*
   c. Utwórz nową role IAM (zostanie stworzona z domyślnymi, wymaganaymi uprawnieniami) nazwa np. *AWSGlueServiceRole-test-jk-12345*
   d. Częstotliwość uruchamiania - on demand
   e. Wyniki powinny być zapisywane w Twojej bazie danych glue (utworzonej w p.1)
   f. Wybierz prefiks dla nowo tworzonych tabel - ***test_crawler_***

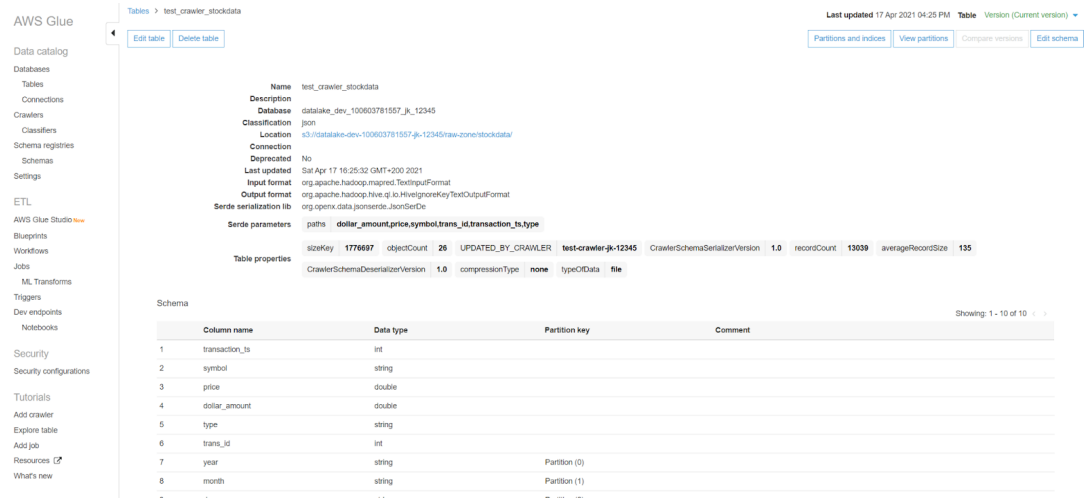
   Przykład konfiguracji:



g. Uruchom Crawler’a



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



i. Sprawdź jej schemat i partycje.

j. Stwórz nowego Crawlera za pomocą kodu TF (*) dla chętnych.
Będziesz potrzebował do tego nowej roli oraz polityk.
Dodatkowy kod w **./labs/iam.tf** może wyglądać tak :

```
resource "aws_iam_role" "glue_crawler_role" {
 name =
"crawler-role-${var.environment}-${var.account_number}-${var.student_i
nitials}-${var.student_index_no}"


 assume_role_policy = <<EOF
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Principal": {
         "Service": "glue.amazonaws.com"
       },
       "Action": "sts:AssumeRole"
     }
```

```
    ]
}
EOF
}

data "aws_iam_policy" "glue_service_policy" {
 arn = "arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole"
}

resource "aws_iam_role_policy" "glue_crawler_user_bucket_policy" {
 name =
"user-bucket-policy-${var.environment}-${var.account_number}-${var.stu
dent_initials}-${var.student_index_no}"
 role = aws_iam_role.glue_crawler_role.id

 policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject",
            "s3:PutObject"
        ],
        "Resource": [
             "${aws_s3_bucket.main_dl_bucket.arn}*"
        ]
    }
  ]
}
EOF
}

resource "aws_iam_policy_attachment" "crawler_attach_managed_policy" {
 name =
"crawler-managed-service-${var.environment}-${var.account_number}-${va
r.student_initials}-${var.student_index_no}"
 roles = [
```

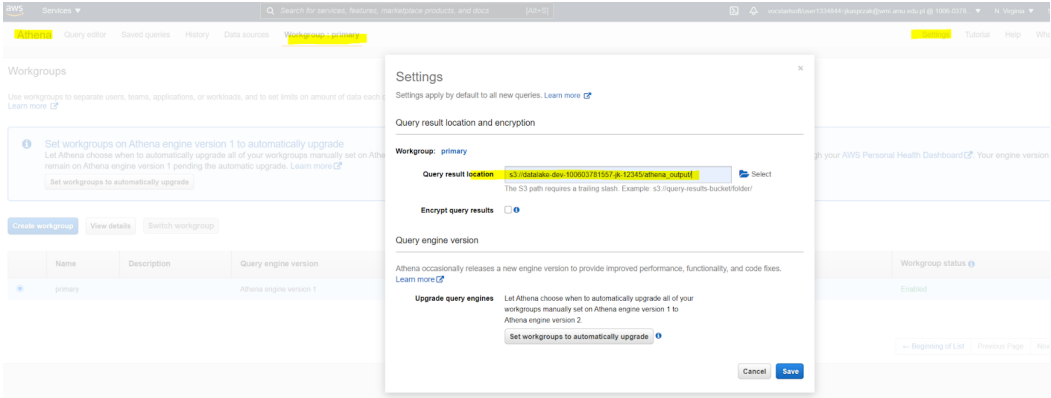
```
    aws_iam_role.glue_crawler_role.name]
  policy_arn = data.aws_iam_policy.glue_service_policy.arn
}
```

Teraz dodaj właściwą definicję Glue Crawlera do pliku **./labs/glue.tf** :

```
resource "aws_glue_crawler" "glue_crawler_raw_zone" {
  database_name = aws_glue_catalog_database.datalake_db_raw_zone.name
  name          =
"gc-raw-${var.environment}-${var.account_number}-${var.student_initial
s}-${var.student_index_no}"
  role          = aws_iam_role.glue_crawler_role.arn
  table_prefix = "crawler_"

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

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

3. Otwórz konsole AWS i serwis Athena. Jeśli pierwszy raz otwierasz tą usługę, konieczne będzie skonfigurowanie **Workgroup Primary** i miejsca składowania wyników. Wybierz swój bucket, dodaj prefix - athena_output



   a. Napisz zapytanie SQL, które z nowo utworzonej tabeli pobierze najwyższe wartości zleceń (BUY / SELL) w przedziałach godzinowych. 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**().

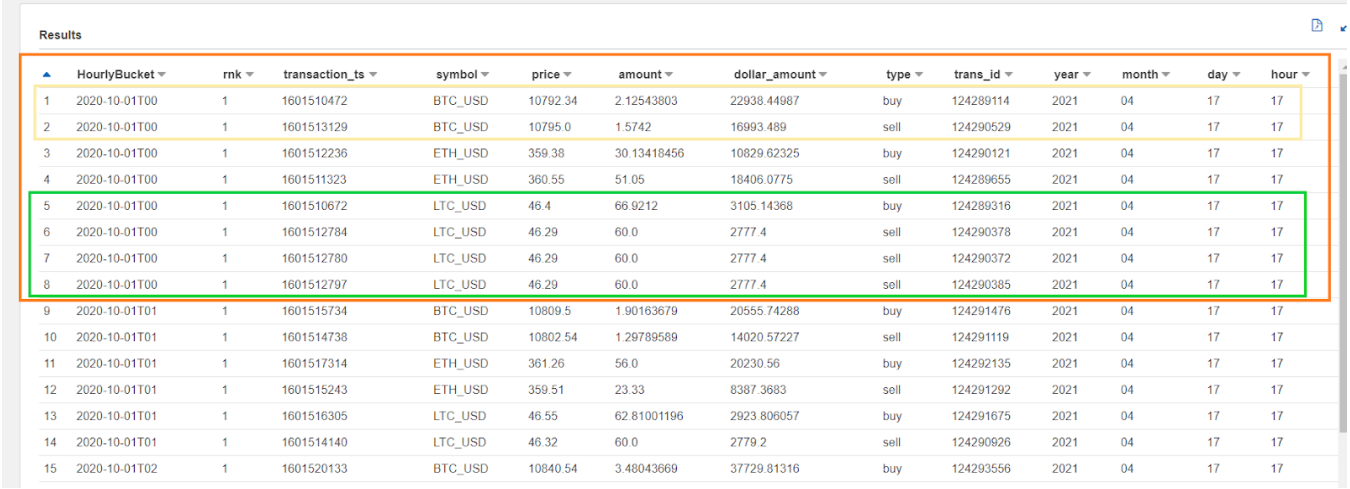
Przykład wykorzystania funkcji from_unixtime() i date_format() :

```
SELECT date_format(from_unixtime(1618669914),'%Y-%m-%dT%H:%i:%sZ')

-----
2021-04-17T14:31:54Z
```

b. 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.

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)) :



Results

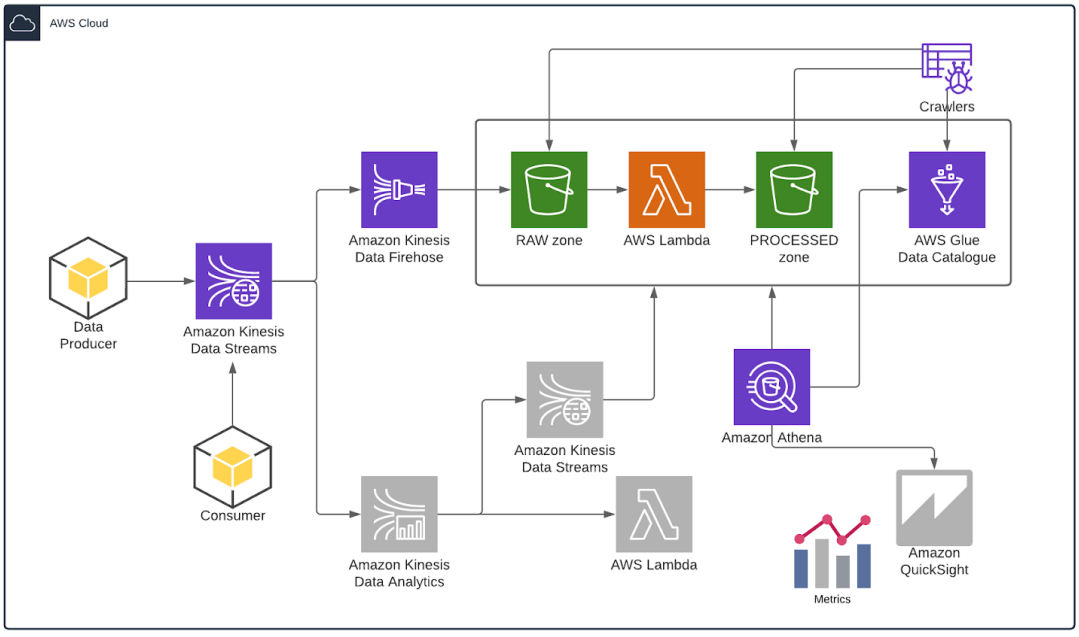
	HourlyBucket	rnk	transaction_ts	symbol	price	amount	dollar_amount	type	trans_id	year	month	day	hour
1	2020-10-01T00	1	1601510472	BTC_USD	10792.34	2.12543803	22938.44987	buy	124289114	2021	04	17	17
2	2020-10-01T00	1	1601513129	BTC_USD	10795.0	1.5742	16993.489	sell	124290529	2021	04	17	17
3	2020-10-01T00	1	1601512236	ETH_USD	359.38	30.13418456	10829.62325	buy	124290121	2021	04	17	17
4	2020-10-01T00	1	1601511323	ETH_USD	360.55	51.05	18406.0775	sell	124289655	2021	04	17	17
5	2020-10-01T00	1	1601510672	LTC_USD	46.4	66.9212	3105.14368	buy	124289316	2021	04	17	17
6	2020-10-01T00	1	1601512784	LTC_USD	46.29	60.0	2777.4	sell	124290378	2021	04	17	17
7	2020-10-01T00	1	1601512780	LTC_USD	46.29	60.0	2777.4	sell	124290372	2021	04	17	17
8	2020-10-01T00	1	1601512797	LTC_USD	46.29	60.0	2777.4	sell	124290385	2021	04	17	17
9	2020-10-01T01	1	1601515734	BTC_USD	10809.5	1.90163679	20555.74288	buy	124291476	2021	04	17	17
10	2020-10-01T01	1	1601514738	BTC_USD	10802.54	1.29789589	14020.57227	sell	124291119	2021	04	17	17
11	2020-10-01T01	1	1601517314	ETH_USD	361.26	56.0	20230.56	buy	124292135	2021	04	17	17
12	2020-10-01T01	1	1601515243	ETH_USD	359.51	23.33	8387.3683	sell	124291292	2021	04	17	17
13	2020-10-01T01	1	1601516305	LTC_USD	46.55	62.81001196	2923.806057	buy	124291675	2021	04	17	17
14	2020-10-01T01	1	1601514140	LTC_USD	46.32	60.0	2779.2	sell	124290926	2021	04	17	17
15	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**

Ćwiczenie 4

Automatyczne procesowanie danych z RAW zone do Processed zone za pomocą wyzwalaczy S3 i funkcji Lambda. Zmiana formatu danych na **Parquet** oraz zmiana partycjonowania - tym razem ze względu na czas wystąpienia zdarzenia (transakcji) - ***transaction_ts***.



Utworzymy trigger (S3 notification), który będzie wyzwalał funkcję Lambda. Ta przetworzy dane z pliku w momencie jego zapisu na S3. Zmieni format i sposób partycjonowania (***parquet***, kompresja snappy).

1. Utwórz funkcję Lambda która będzie czytała dane z pliku S3 (tuż po zapisie) i przenosiła je do nowej lokalizacji - processed-zone, zapisując w formacie **parquet**. Możesz to wykonać najpierw testowo z konsoli AWS, ale finalnie zrób poniższe kroki aby była ona zdefiniowana za pomocą Terraform,
    a. Do tego celu wykorzystamy kilka dołączonych plików. Przejrzyj zawartość **./labs/lambda/lambda_definition.py** - to kod naszej funkcji.
    Zwróć uwagę w jaki sposób będziemy zapisywali dane - zmian sposobu

partycjonowania, zmieniona lista kolumn (np. artybut symbol będzie częścią partycjonowania a nie pliku z danymi)

b. Funkcja lambda będzie używała spakowanej (**zip**) wersji tego skryptu (znajdziesz go w tym samym folderze)

c. Do odczytu i zapisu danych będziemy wykorzystywali pakiet Python **awswrangler** : ://pypi.org/project/awswrangler/
Trzeba go będzie dołączyć do funkcji lambda (również **zip**) jako dodatkowa "warstwa" rozszerzająca możliwości obrazu na którym nasza funkcja lambda 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_${var.environment}_${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 do czytania i zapisu danych w S3 bucket. Bedą więc potrzebne dwa dodatkowe obiekty w pliku **/labs/terraform/iam.tf**

Utwórz dodatkową rolę i politykę - przykład poniżej :

```
resource "aws_iam_role" "lambda_basic_role" {
 name = "lambda-basic-role-${var.environment}-${var.account_number}-${var.student_initials}-${var.student_index_no}"
```

```
 tags = merge(local.common_tags, )

 assume_role_policy = <<EOF
{
"Version": "2012-10-17",
 "Statement": [
   {
     "Action": "sts:AssumeRole",
     "Principal": {
       "Service": "lambda.amazonaws.com"
     },
     "Effect": "Allow",
     "Sid": ""
   }
 ]
}
EOF
}

resource "aws_iam_role_policy" "lambda_basic_policy" {
 name =
"lambda-basic-policy-${var.environment}-${var.account_number}-${var.st
udent_initials}-${var.student_index_no}"
 role = aws_iam_role.lambda_basic_role.id

 policy = <<EOF
{
"Version": "2012-10-17",
"Statement": [
  {
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents"
    ],
    "Effect": "Allow",
    "Resource": "*"
  },
  {
```

```
    "Effect": "Allow",
    "Action": "s3:*",
    "Resource": [
            "${aws_s3_bucket.main_dl_bucket.arn}",
            "${aws_s3_bucket.main_dl_bucket.arn}/*"]
  }
]
}
EOF
}
```

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-${var.environment}-${var.account_number}-${var.st
udent_initials}-${var.student_index_no}"
 filename      = "../lambda/lambda_definition.zip"
 handler = "lambda_definition.etl_function"
 runtime = "python3.8"
 role = aws_iam_role.lambda_basic_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.

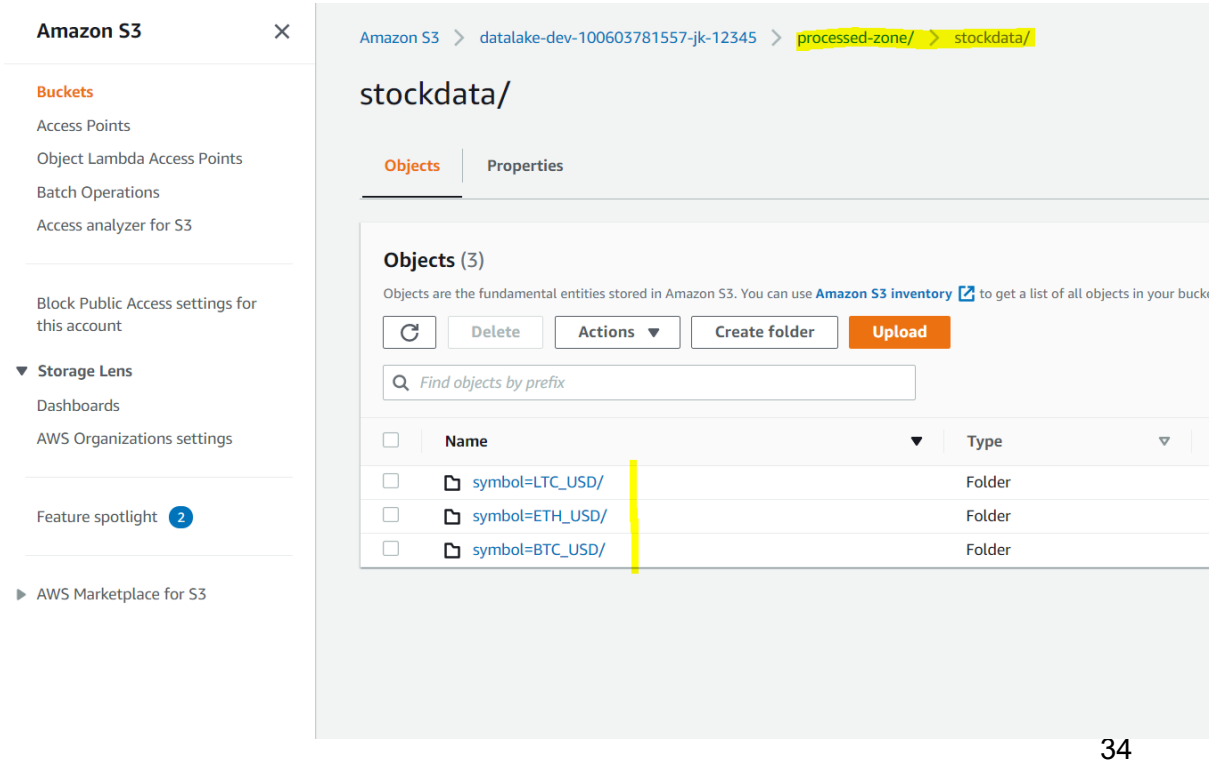
Przykładowe definicje:

```
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.main_dl_bucket.arn
}

resource "aws_s3_bucket_notification" "trigger_etl_lambda" {
 bucket = aws_s3_bucket.main_dl_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]
}
```

h. W tym momencie możemy wdrożyć zmiany. Uruchom **terraform apply.**

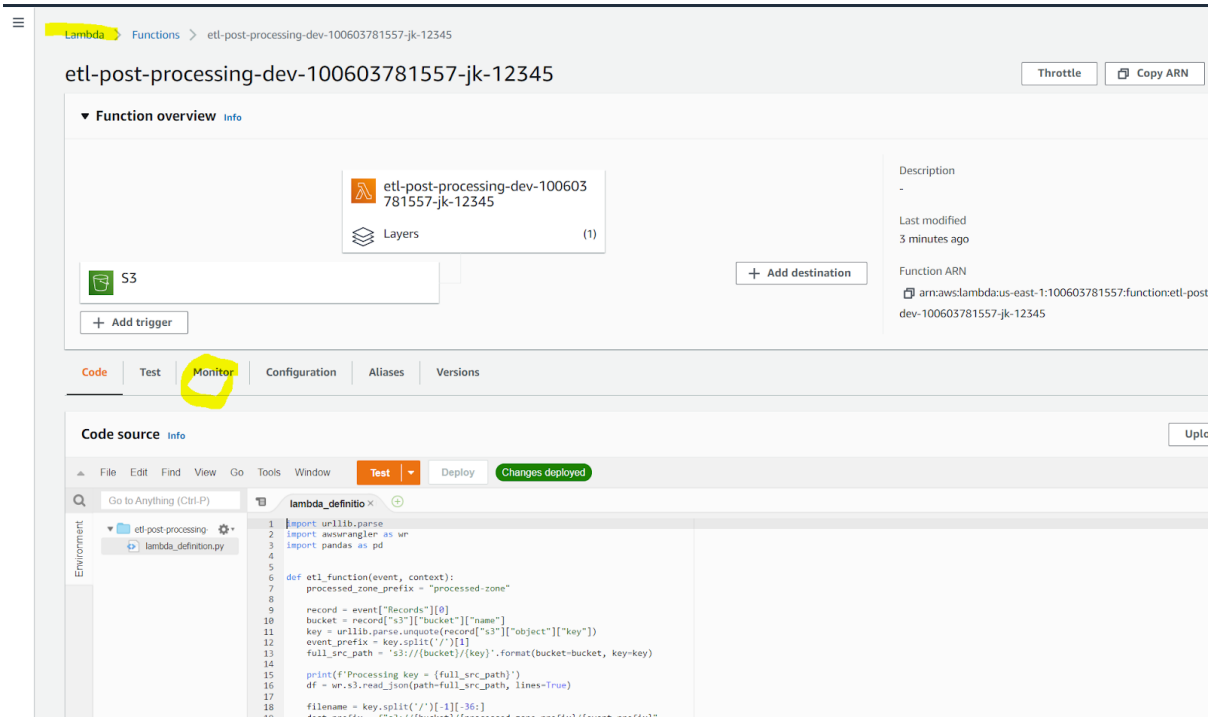
i. Po utworzeniu wszystkich obiektów wystartuj ponownie generator danych - odczekaj minutę.

j. Sprawdź czy dane są automatycznie przenoszone do prefixu processed-zone (dotyczyć to będzie tylko nowych plików zrzucanych do S3 przez Firehose).



Zwróć uwagę, że funkcja Lambda, zmienia także schemat partycjonowania. Nadrzędną 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.

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



2. Utwórz dodatkową tabelę w Glue. Możesz stworzyć / zaktualizować istniejące Crawlera lub zarejestrować tabelę z poziomu Athena
    a. Uruchom w **Athena** poniższy kodu SQL (zmień lokalizację S3) :

```
CREATE EXTERNAL TABLE 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://datalake-dev-100603781557-jk-12345/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).

3. Wykonaj porównanie strefy RAW z PROCESSED (reconciliation).
    a. Wyczyść całe środowisko **(terraform destroy & apply)** i rozpocznij generowanie danych od początku. Poczekaj parę minut aż przynajmniej dwie pierwsze godziny z transakcjami zostaną wysłane i zrzucone do RAW zone. Uruchom stworzonego crawlera i utwórz raz jeszcze tabele w strefie processed (odśwież partycje).
    b. Porównaj pierwszy bucket godzinowy w obu warstwach lake’a

```
SELECT count(*)
FROM "datalake_dev_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_dev_100603781557_jk_12345"."processed_stockdata"
WHERE HOUR=0 and DAY=1 and MONTH=10 and YEAR=2020
      and symbol = 'BTC_USD';
```

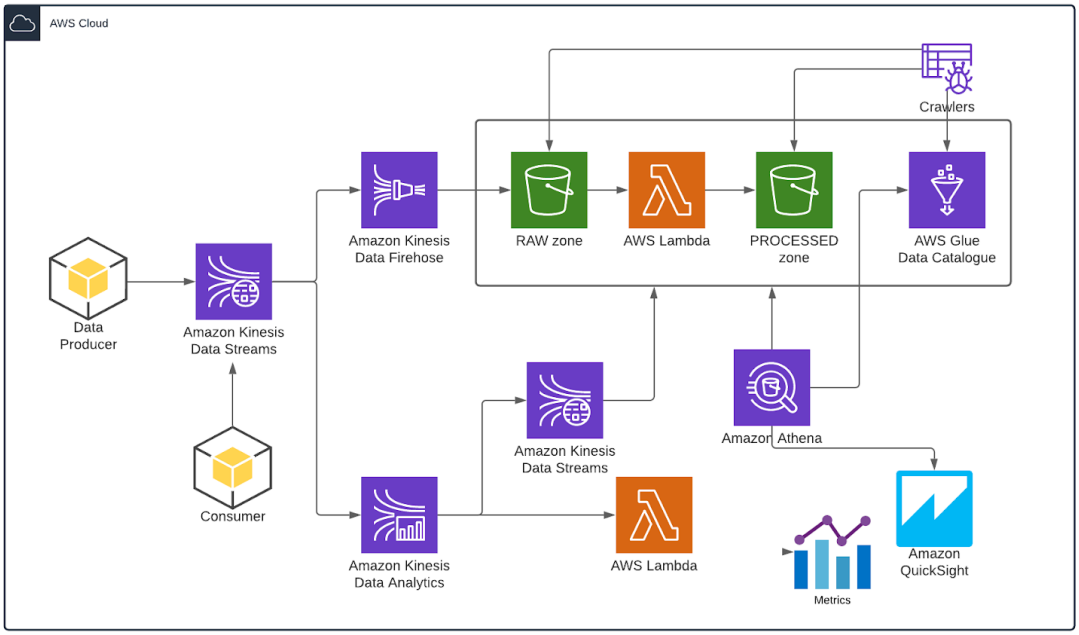
   c. Powinieneś otrzymać te same ilości rekordów w obu warstwach (337 sztuk dla pary BTC_USD)

4. (*) Zaproponuj rozwiązanie automatycznej rejestracji partycji.

Ćwiczenie 5

Analiza w czasie rzeczywistym napływających danych z wykorzystaniem Kinesis Analytics. W tym ćwiczeniu złożymy wszystkie elementy scenariusza w jeden, działający system.

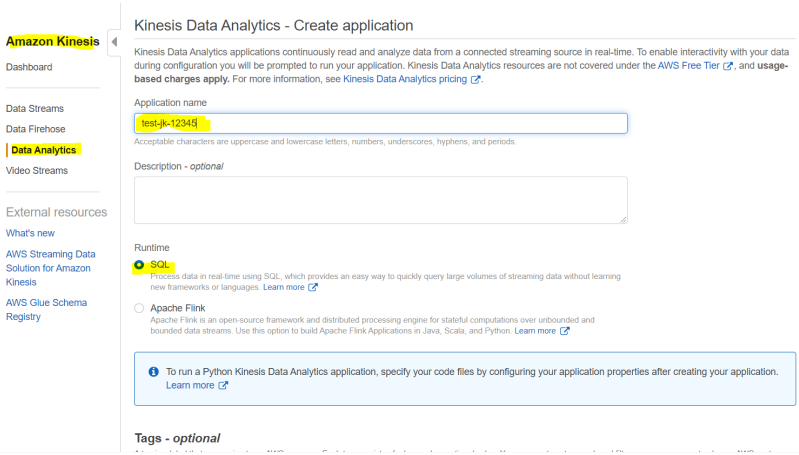
Dodamy analizę w czasie rzeczywistym danych przepływających przez utworzony Kinesis Data Stream :



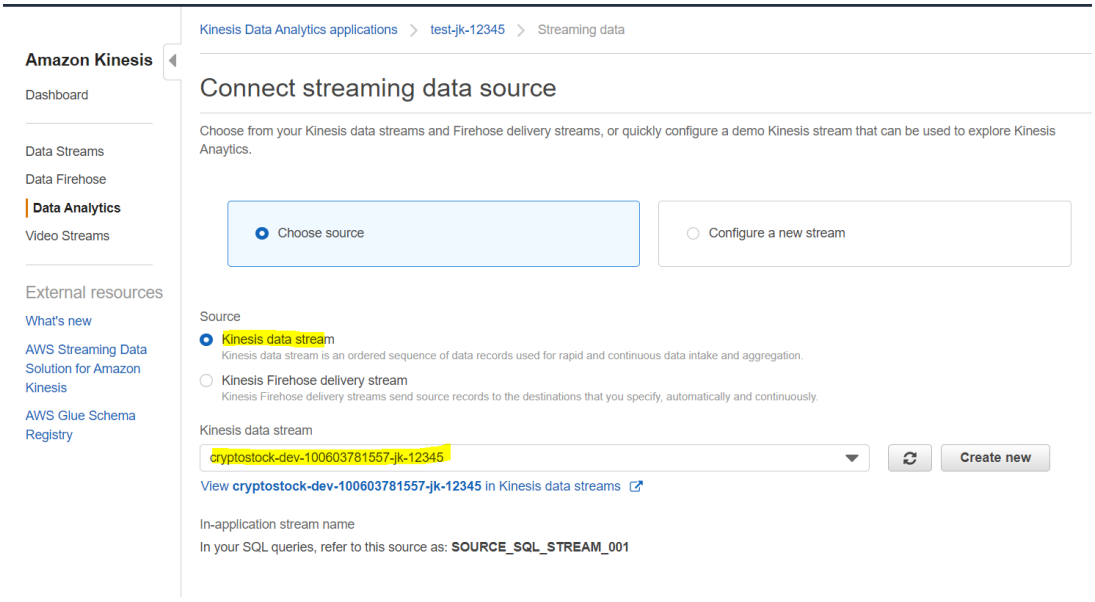
W aplikacji **Kinesis Analytics** (SQL), będziemy wychwytywali transakcje w ramach konkretnego typu zleceń (buy/sell) oraz symbolu “**BTC_USD**”, które przekraczają czterokrotnie średnią wartość transakcji z ostatnich 30 sekund. Analizowana będzie każda transakcja.

Te szczególne zlecenia, będą przesyłane do nowego Kinesis Firehose i ładowane do Data Lake celem późniejszej analizy. Dodatkowo wyzwalana będzie funkcja Lambda, która ma na celu, natychmiastowe obsłużenie alertu.

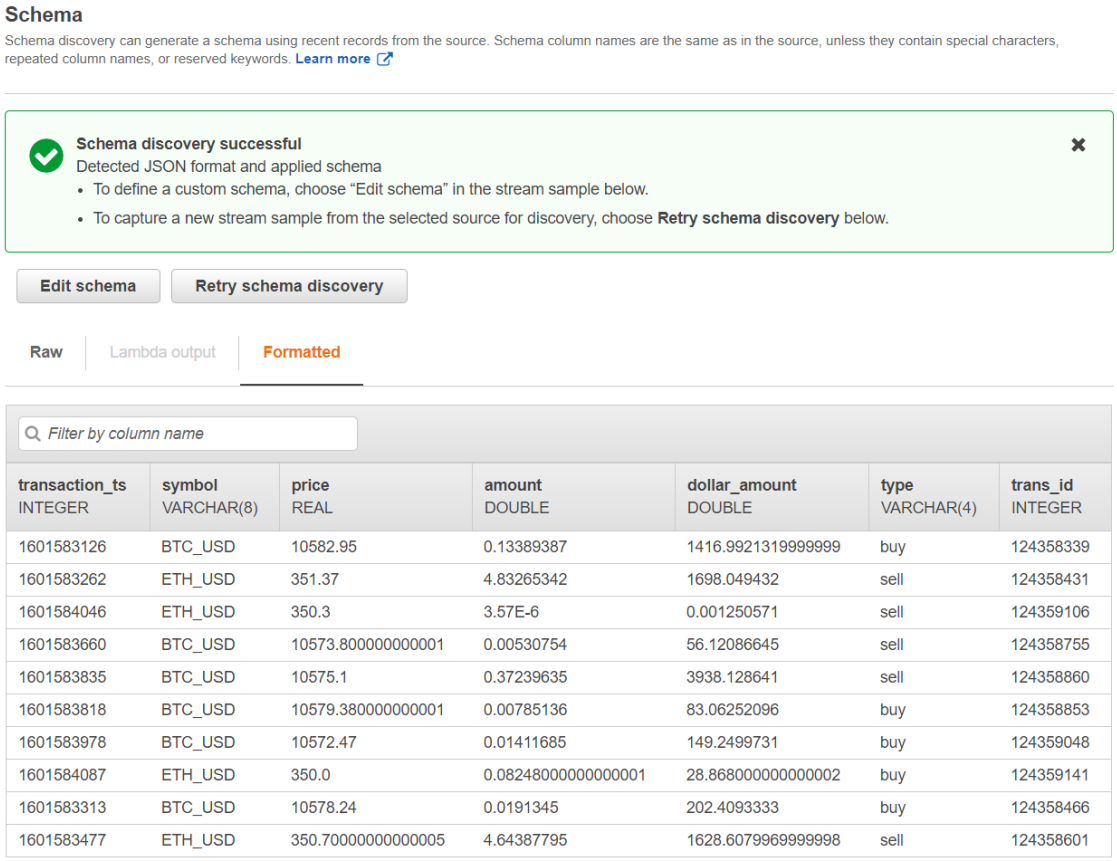
1. Utwórz nową aplikację Kinesis Analytics wykorzystując konsolę AWS
    a. Nazwa nowej aplikacji - test-{inicjały}-{nr_indeksu}



b. Jako źródło, wybierz utworzony wcześniej strumień danych

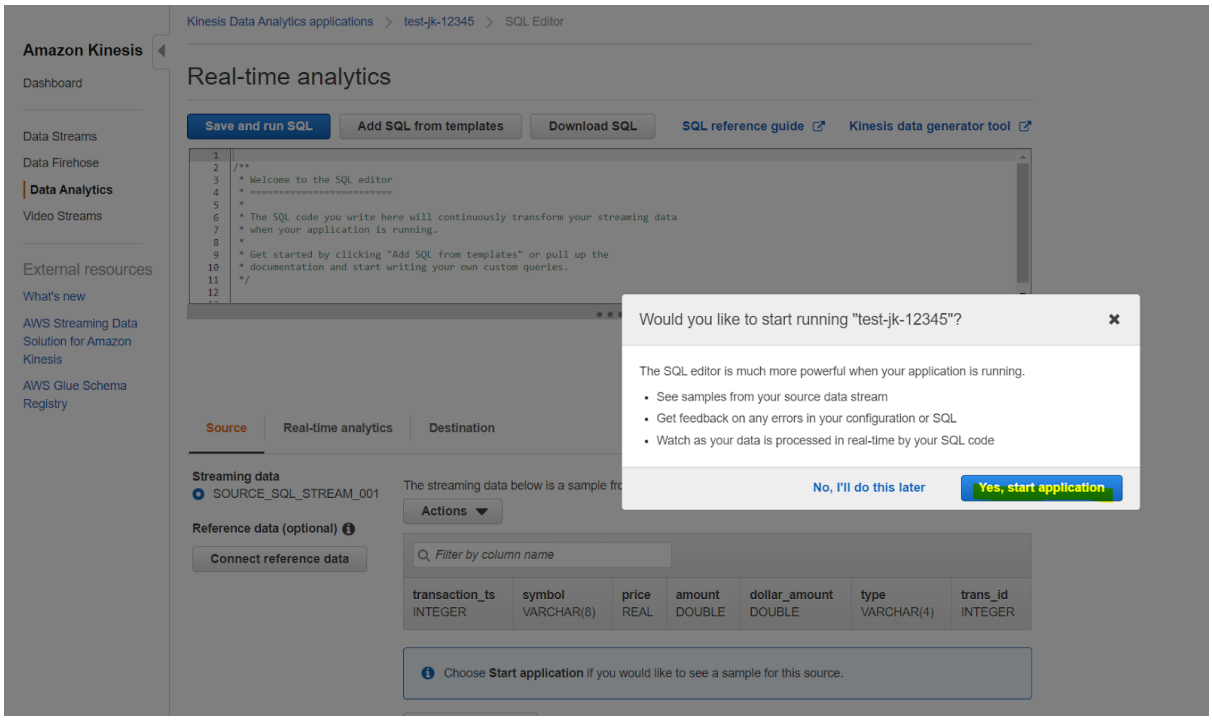


c. Pozwól aby Kinesis automatycznie wykrył schemat (generator musi generować dane)

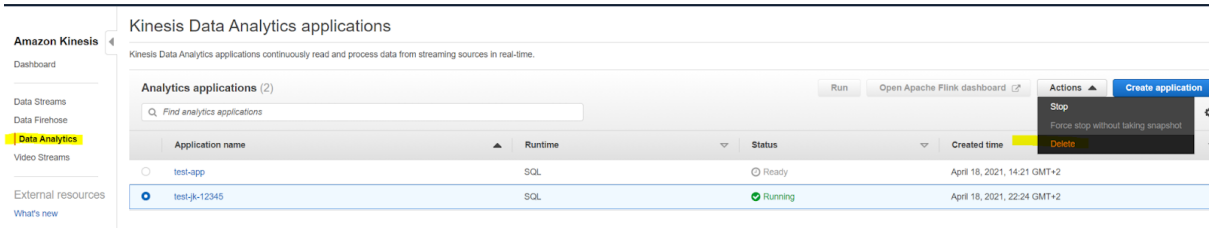


transaction_ts INTEGER	symbol VARCHAR(8)	price REAL	amount DOUBLE	dollar_amount DOUBLE	type VARCHAR(4)	trans_id INTEGER
1601583126	BTC_USD	10582.95	0.13389387	1416.9921319999999	buy	124358339
1601583262	ETH_USD	351.37	4.83265342	1698.049432	sell	124358431
1601584046	ETH_USD	350.3	3.57E-6	0.001250571	sell	124359106
1601583660	BTC_USD	10573.800000000001	0.00530754	56.12086645	sell	124358755
1601583835	BTC_USD	10575.1	0.37239635	3938.128641	sell	124358860
1601583818	BTC_USD	10579.380000000001	0.00785136	83.06252096	buy	124358853
1601583978	BTC_USD	10572.47	0.01411685	149.2499731	buy	124359048
1601584087	ETH_USD	350.0	0.08248000000000001	28.868000000000002	buy	124359141
1601583313	BTC_USD	10578.24	0.0191345	202.4093333	buy	124358466
1601583477	ETH_USD	350.70000000000005	4.64387795	1628.6079969999998	sell	124358601

d. Przejdź do edytora SQL i uruchom aplikację



Usługa Kinesis jest stosunkowo droga (najdroższy element tych laboratorium) - **od tego momentu naliczane są koszty - 0.11$ / godzinę** działania. Pamiętaj aby wyłączyć aplikację po zakończeniu testów !!!



e. Wklej do okna edytora SQL następującą definicję :

```
--          .----------.   .----------.   .----------.
--          |  SOURCE  |   |  INSERT  |   |  DESTIN. |
-- Source-->|  STREAM  |-->| & SELECT |-->|  STREAM  |-->Destination
--          |          |   |  (PUMP)  |   |          |
--          '----------'   '----------'   '----------'


CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
("symbol" VARCHAR(10), "type" VARCHAR(10), "trans_id" BIGINT,
    "dollar_amount" DOUBLE, "AvgLast30seconds" DOUBLE,
"CntLast30seconds"  INT,
    "SumLast30rows" DOUBLE, "CntLast30rows"  INT, "max_tran_id" BIGINT
);

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO
"DESTINATION_SQL_STREAM"
SELECT STREAM "symbol", "type", "trans_id", "dollar_amount",
"AvgLast30seconds", "CntLast30seconds"
 , "SumLast30rows", "CntLast30rows", "max_tran_id"
FROM (

    SELECT STREAM "symbol", "type", "trans_id", "dollar_amount",
        AVG("dollar_amount") OVER LAST_30_SECS AS "AvgLast30seconds",
        COUNT(*)  OVER LAST_30_SECS AS "CntLast30seconds",
        SUM("dollar_amount") OVER LAST_30_ROWS AS "SumLast30rows",
        COUNT(*)  OVER LAST_30_ROWS AS "CntLast30rows",
        MAX("trans_id") OVER LAST_30_ROWS AS "max_tran_id"
    FROM "SOURCE_SQL_STREAM_001"
    WHERE "symbol" = 'BTC_USD'
    WINDOW
        LAST_30_SECS AS (PARTITION BY "symbol", "type" RANGE INTERVAL
'30' SECOND PRECEDING),
        LAST_30_ROWS AS (PARTITION BY "symbol", "type" ROWS 30
PRECEDING)
)
WHERE "dollar_amount" > 4 * ("AvgLast30seconds");
```

Przeanalizuj kod. Zostanie utworzony schemat strumienia wyjściowego

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
("symbol" VARCHAR(10), "type" VARCHAR(10), "trans_id" BIGINT,
    "dollar_amount" DOUBLE, "AvgLast30seconds" DOUBLE,
"CntLast30seconds"  INT,
    "SumLast30rows" DOUBLE, "CntLast30rows"  INT, "max_tran_id" BIGINT
);
```

A także definicja “pompy” ładującej dane ze źródła do celu

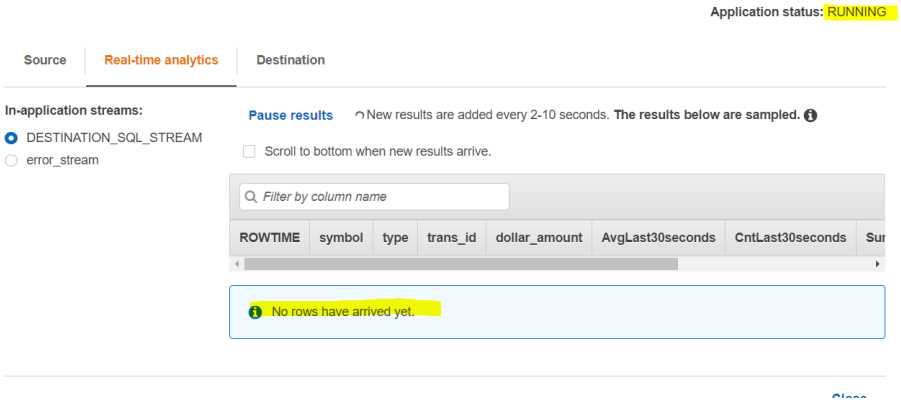
```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO
"DESTINATION_SQL_STREAM"
SELECT STREAM "symbol", "type", "trans_id", "dollar_amount",
"AvgLast30seconds", "CntLast30seconds"
 , "SumLast30rows", "CntLast30rows", "max_tran_id"
FROM (

    SELECT STREAM "symbol", "type", "trans_id", "dollar_amount",
        AVG("dollar_amount") OVER LAST_30_SECS AS "AvgLast30seconds",
        COUNT(*)  OVER LAST_30_SECS AS "CntLast30seconds",
        SUM("dollar_amount") OVER LAST_30_ROWS AS "SumLast30rows",
        COUNT(*)  OVER LAST_30_ROWS AS "CntLast30rows",
        MAX("trans_id") OVER LAST_30_ROWS AS "max_tran_id"
    FROM "SOURCE_SQL_STREAM_001"
    WHERE "symbol" = 'BTC_USD'
    WINDOW
        LAST_30_SECS AS (PARTITION BY "symbol", "type" RANGE INTERVAL
'30' SECOND PRECEDING),
        LAST_30_ROWS AS (PARTITION BY "symbol", "type" ROWS 30
PRECEDING)
)
WHERE "dollar_amount" > 4 * ("AvgLast30seconds");
```

Dialekt SQL który tu widzisz jest dość specyficzny i nie jest to pełna implementacja ANSI. Nie mniej jednak ma spore możliwości, które umożliwiają stworzenie aplikacji analitycznej czasu rzeczywistego.

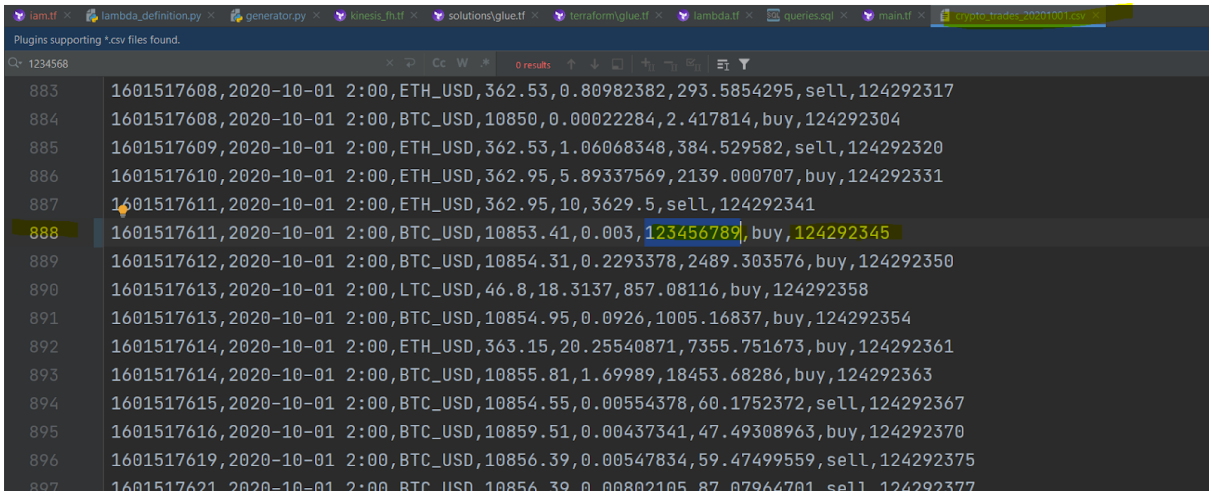
Właściwe zapytanie SQL pobiera dane na bieżąco ze strumienia wejściowego i wykorzystując dwie definicje okna, wykonuje zadaną analitykę. Do strumienia wyjściowego trafiają transakcje których wartość (dollar_amount) jest większa niż czterokrotność średniej ze wcześniejszych transakcji tego typu.
Całość analizy jest wykonywana tylko dla jednego symbolu.

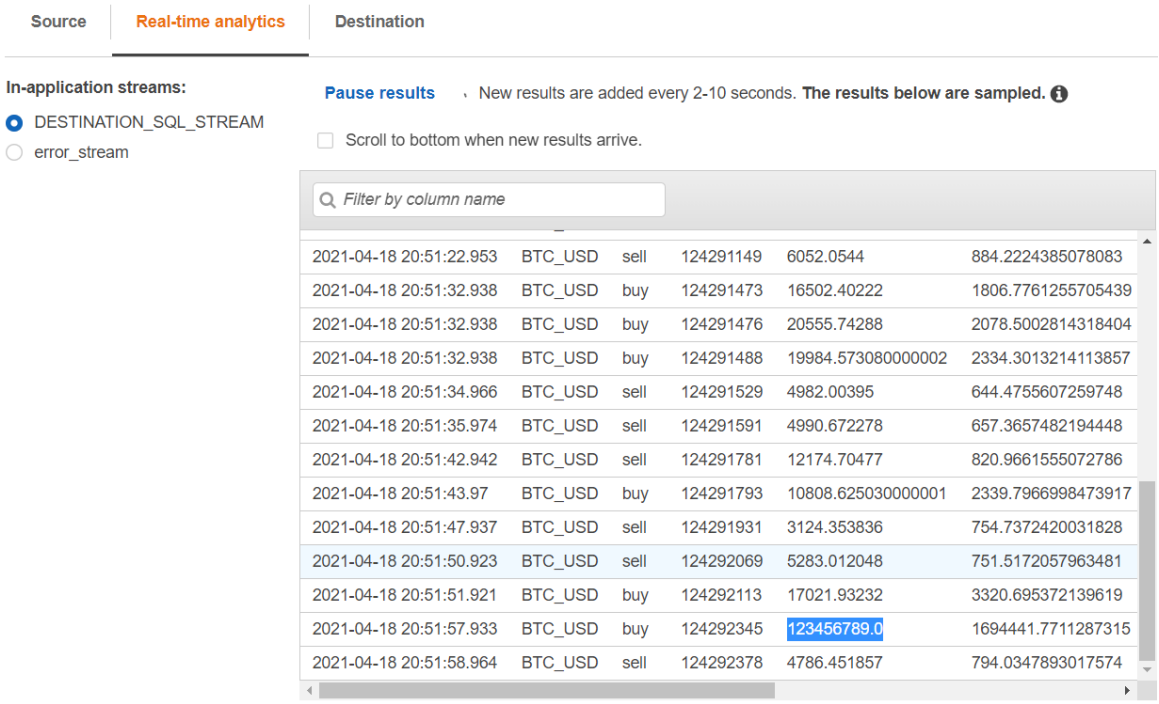
Uruchom aplikację Kinesis (poczekaj aż się uruchomi) zrestartuj generator danych. :

```
CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM"
("symbol" VARCHAR(10), "type" VARCHAR(10), "trans_id" BIGINT,
    "dollar_amount" DOUBLE, "AvgLast30seconds" DOUBLE, "CntLast30seconds"  INT,
    "SumLast30rows" DOUBLE, "CntLast30rows"  INT, "max_tran_id" BIGINT );

CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM "symbol", "type", "trans_id", "dollar_amount", "AvgLast30seconds", "CntLast30seconds"
  , "SumLast30rows", "CntLast30rows", "max_tran_id"
FROM (

    SELECT STREAM "symbol", "type", "trans_id", "dollar_amount",
        AVG("dollar_amount") OVER LAST_30_SECS AS "AvgLast30seconds",
        COUNT(*)  OVER LAST_30_SECS AS "CntLast30seconds",
        SUM("dollar_amount") OVER LAST_30_ROWS AS "SumLast30rows",
        COUNT(*)  OVER LAST_30_ROWS AS "CntLast30rows",
        MAX("trans_id") OVER LAST_30_ROWS AS "max_tran_id"
    FROM "SOURCE_SQL_STREAM_001"
    WHERE "symbol" = 'BTC_USD'
    WINDOW
        LAST_30_SECS AS (PARTITION BY "symbol", "type" RANGE INTERVAL '30' SECOND PRECEDING),
        LAST_30_ROWS AS (PARTITION BY "symbol", "type" ROWS 30 PRECEDING)
)
WHERE "dollar_amount" > 5 * ("SumLast30rows" - "dollar_amount");
```



Jedna z transakcji jest szczególnie duża (zmodyfikowana została ręcznie) i powinna być wychwycona przez naszą aplikację. Chwilę to potrwa (linia 888), ale po dwóch minutach powinieneś ją zobaczyć.





f. Zwróć uwagę w jaki sposób wyliczana jest średnia i jakie możliwości oferuje funkcja okna tutaj.
g. (*) Utwórz nową funkcję lambda (konsola AWS / TF), która będzie emulować akcję - może to być po prostu print do loga aby sprawdzić że faktycznie jest ona wyzwalana.
h. (*) Stwórz dodatkowy obiekt Kinesis Firehose i podepnij strumień wyjściowy z Kinesis Analytics.
   Zastosuj podobne parametry konfiguracyjne (częstotliwość zrzucania danych). Wykorzystaj wiedzę ze wcześniejszych punktów (konieczność zdefiniowania dodatkowych uprawnień etc.)