Rozpoznanie narzędzi raportowych w Azure

Arkadiusz Chmura, Bartosz Ciesielski, Bartosz Przybył

1 Wprowadzenie

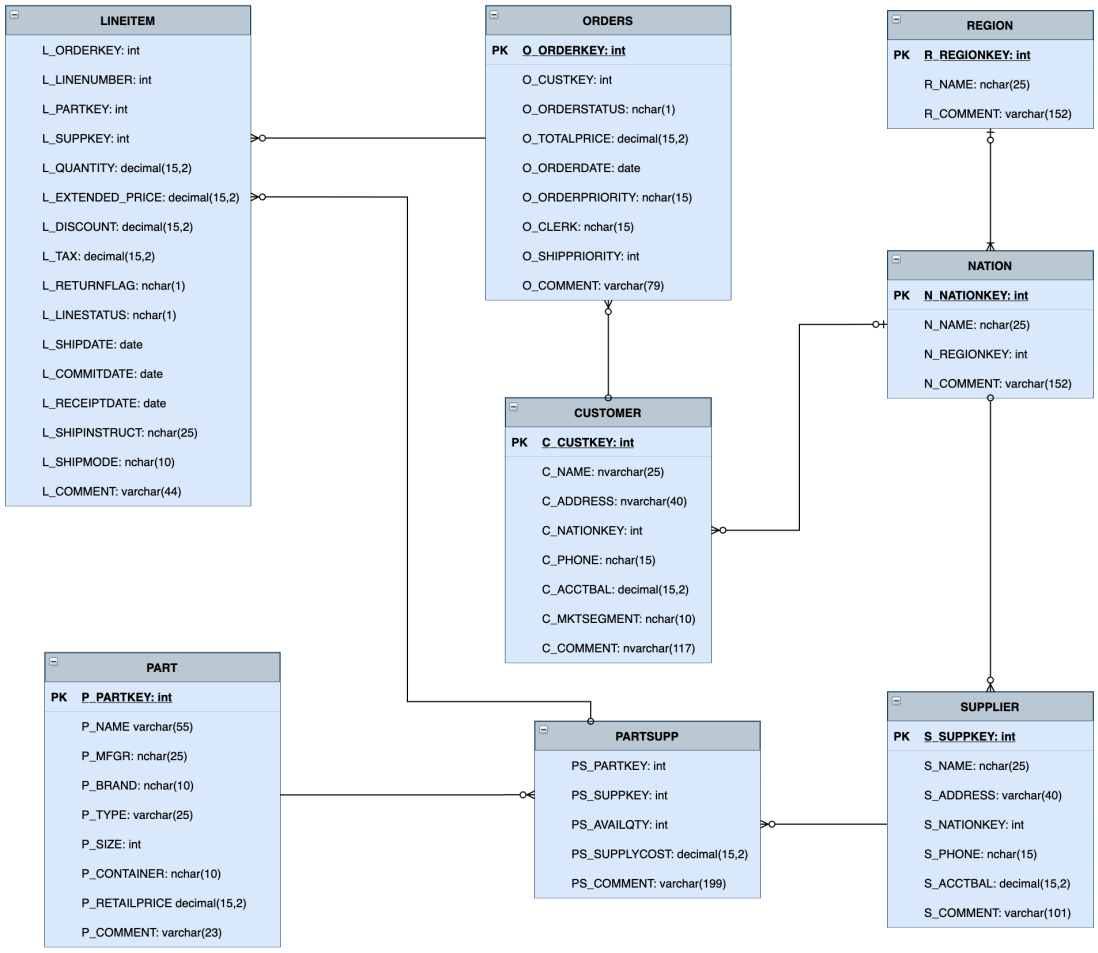
Poniższy dokument stanowi udokumentowanie pracy nad projektem wykonywanym w ramach przedmiotu *Hurtownie danych i przetwarzanie analityczne* we współpracy z firmą *Santander*.

2 Cel projektu

Celem projektu jest weryfikacja natywnych narzędzi raportowych dostępnych w chmurze *Microsoft Azure* i porównanie ich pod względem funkcjonalnym z narzędziami wykorzystywanymi w obecnym procesie raportowym.

Efektem końcowym projektu będzie propozycja architektury nowego środowiska raportowego w chmurze oraz prezentacja wymaganych funkcjonalności na przykładzie.

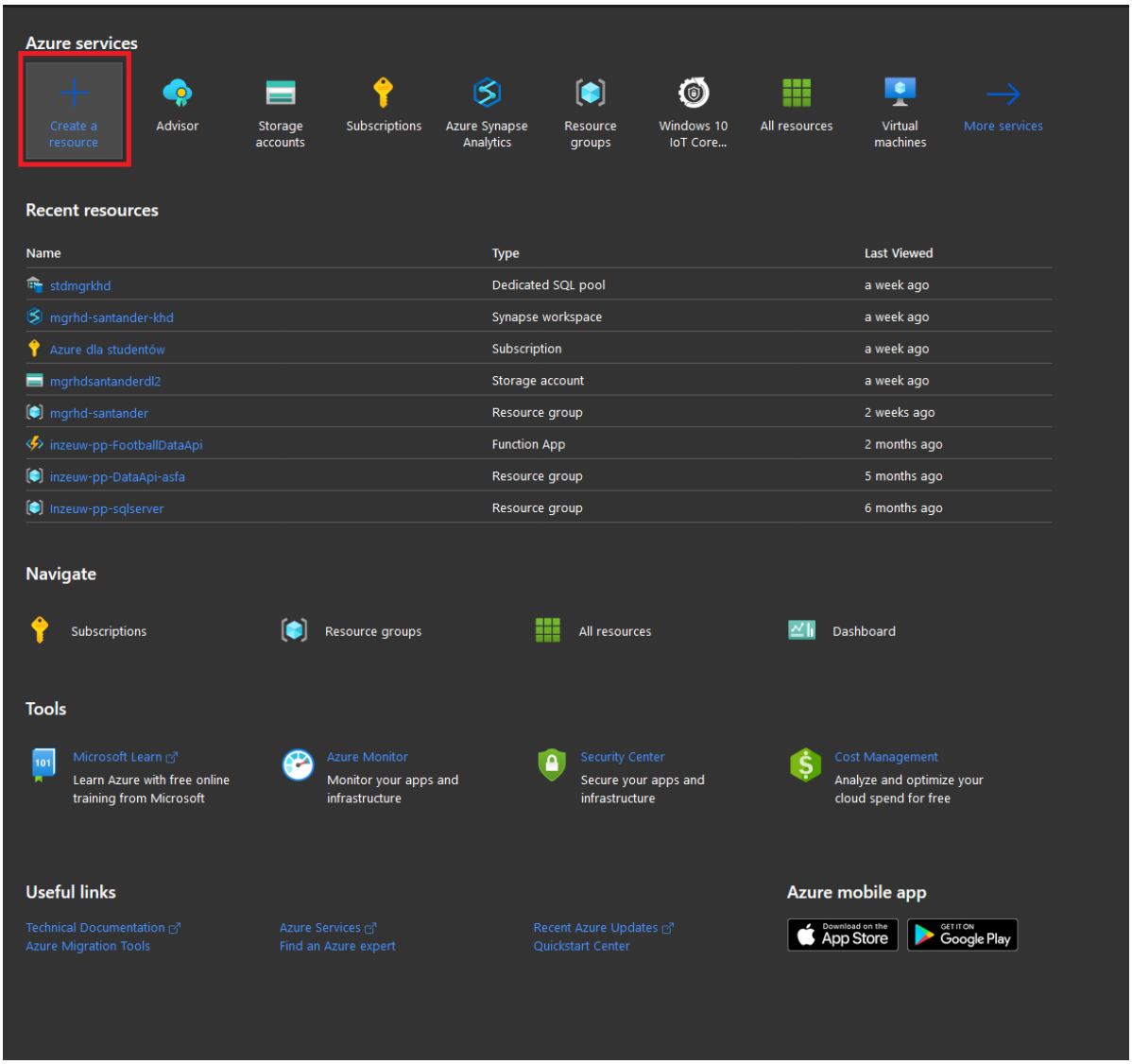
3 Schemat danych



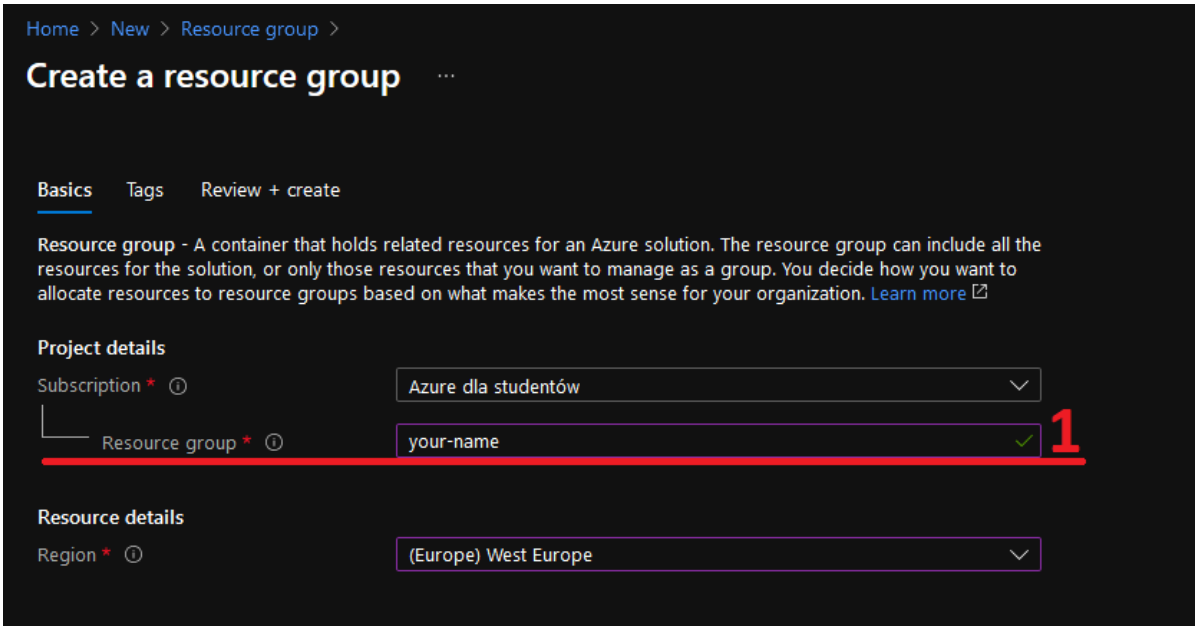
4 Konfiguracja środowiska pracy na platformie Microsoft Azure

Konfigurację rozpoczynamy od zalogowania się do portalu na konto, na którym zostaną umieszczone wszystkie utworzone zasoby. Przed rozpoczęciem zalecamy zmianę języka klikając na zębatkę widoczną w prawym górnym rogu strony. Po otwarciu menu podręcznego wybieramy zakładkę dotyczącą języka i zmieniamy główny język na angielski.

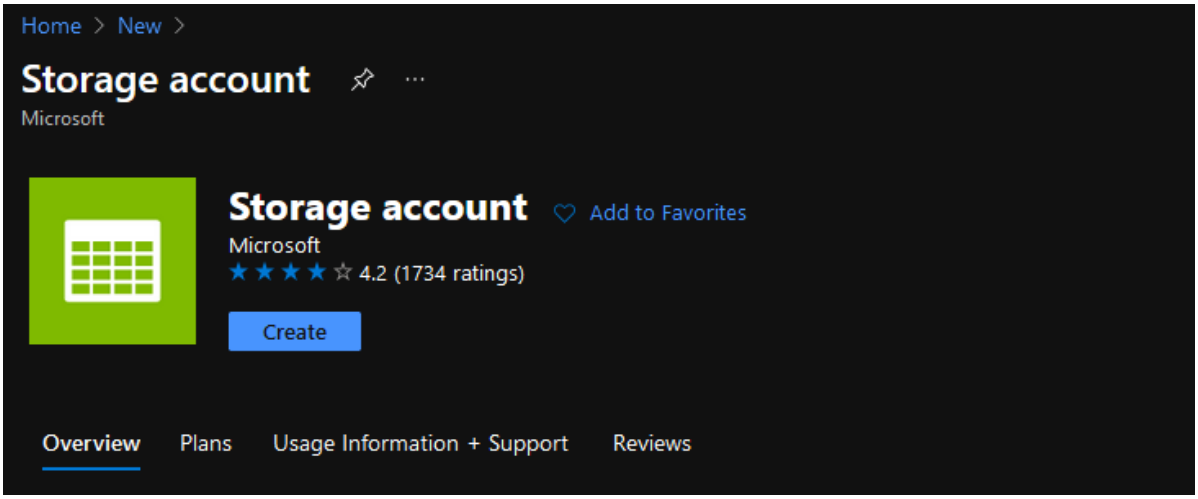
1. Będąc zalogowanym na platformie *Microsoft Azure* kliknij w zaznaczoną na obrazie ikonę plusa.



2. Wyszukaj zasób „Resource group” i kliknij przycisk „Create”. W polu „Subscription” powinna zostać wybrana subskrypcja, z której pobierane będą środki pieniężne. Kolejno wybierz nazwę grupy.



3. Na ekranie wyszukiwania zasobów kolejno znajdź „Storage account” i kliknij „Create”.

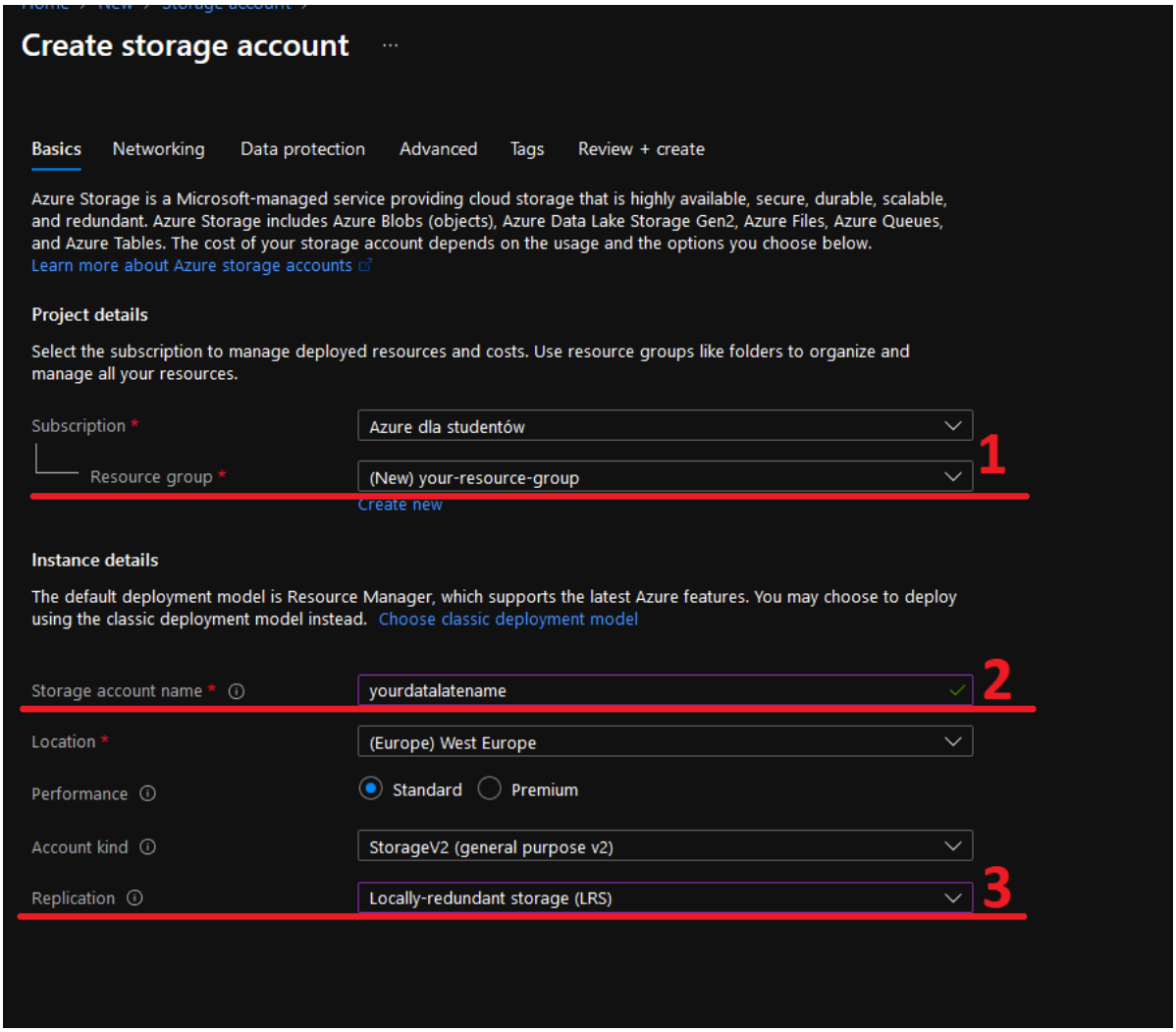


4. W celu utworzenia nowego „Storage account”:

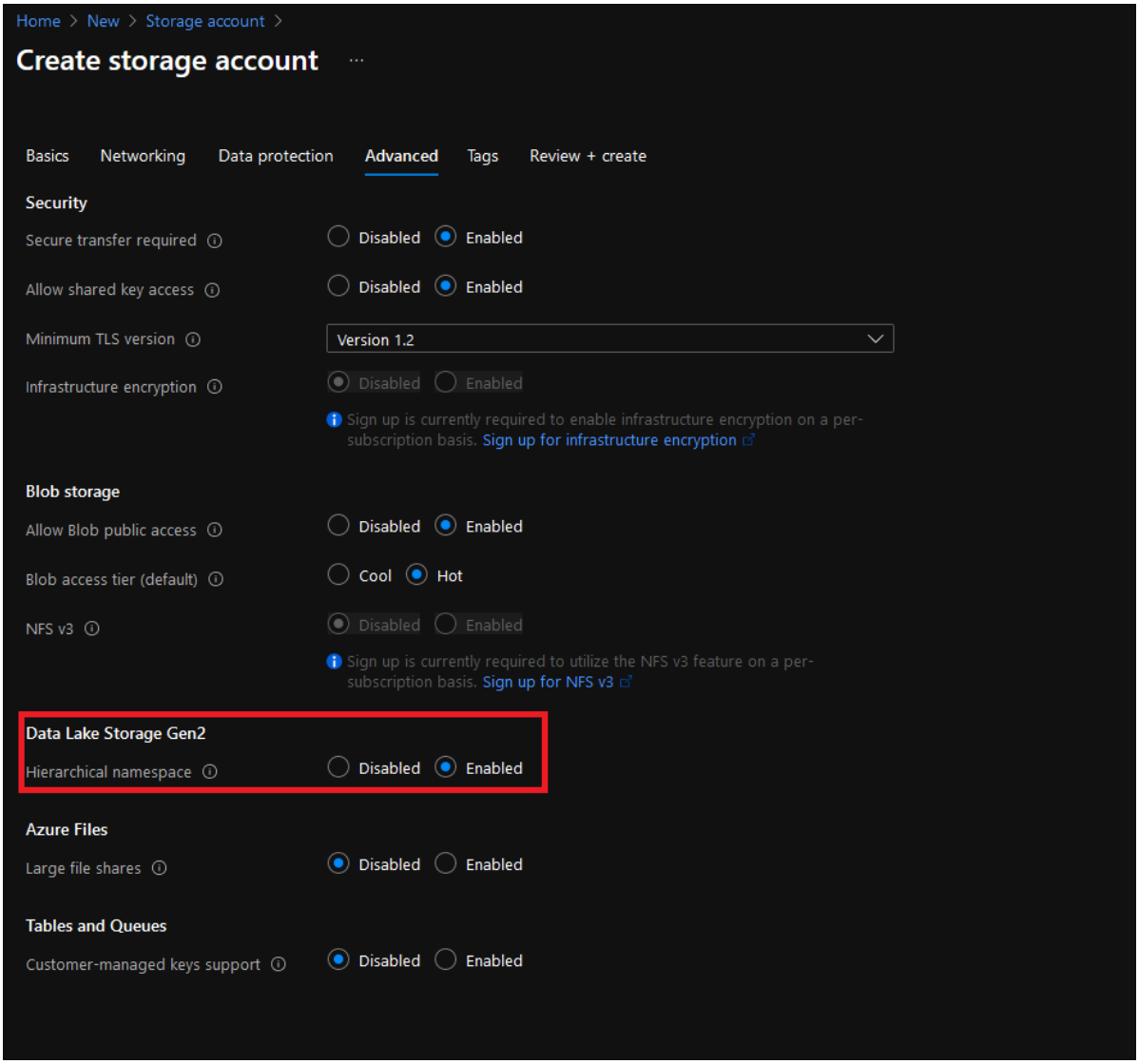
   4.1. Wybierz poprzednio utworzoną grupę zasobów

   4.2. Wybierz nazwę dla swojego konto magazynu, który będzie Twoim *Data Lake'iem.* Kolejno wybierz lokalizację - najlepiej najbliższą, czyli West Europe

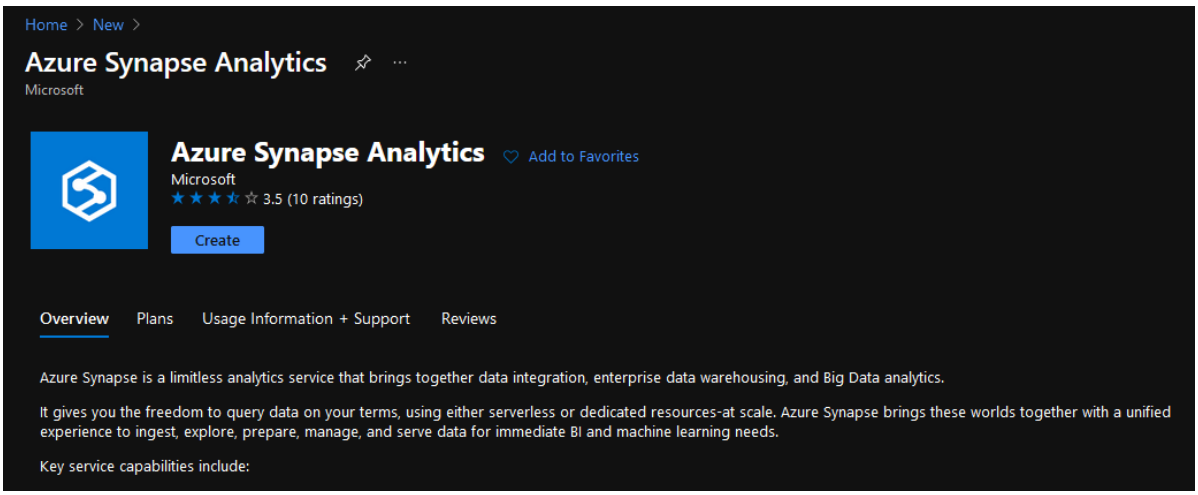
   4.3. Dla oszczędności zmień opcję „Replication” na *Locally-reduntant storage (LRS)*



5. Przejdź do zakładki „Advanced” i zmień opcję „Hierarchical namespace” na „Enabled”. Zmiana tej opcji sprawi, że nasz storage account będzie typu data lake. Kolejno możesz przejść do zakładki „Review + create” i utworzyć storage account.



6. Następnie powróć do ekranu dodawania zasobów i znajdź *Azure Synapse Analytics* i również kliknij „Create” jak na obrazku.



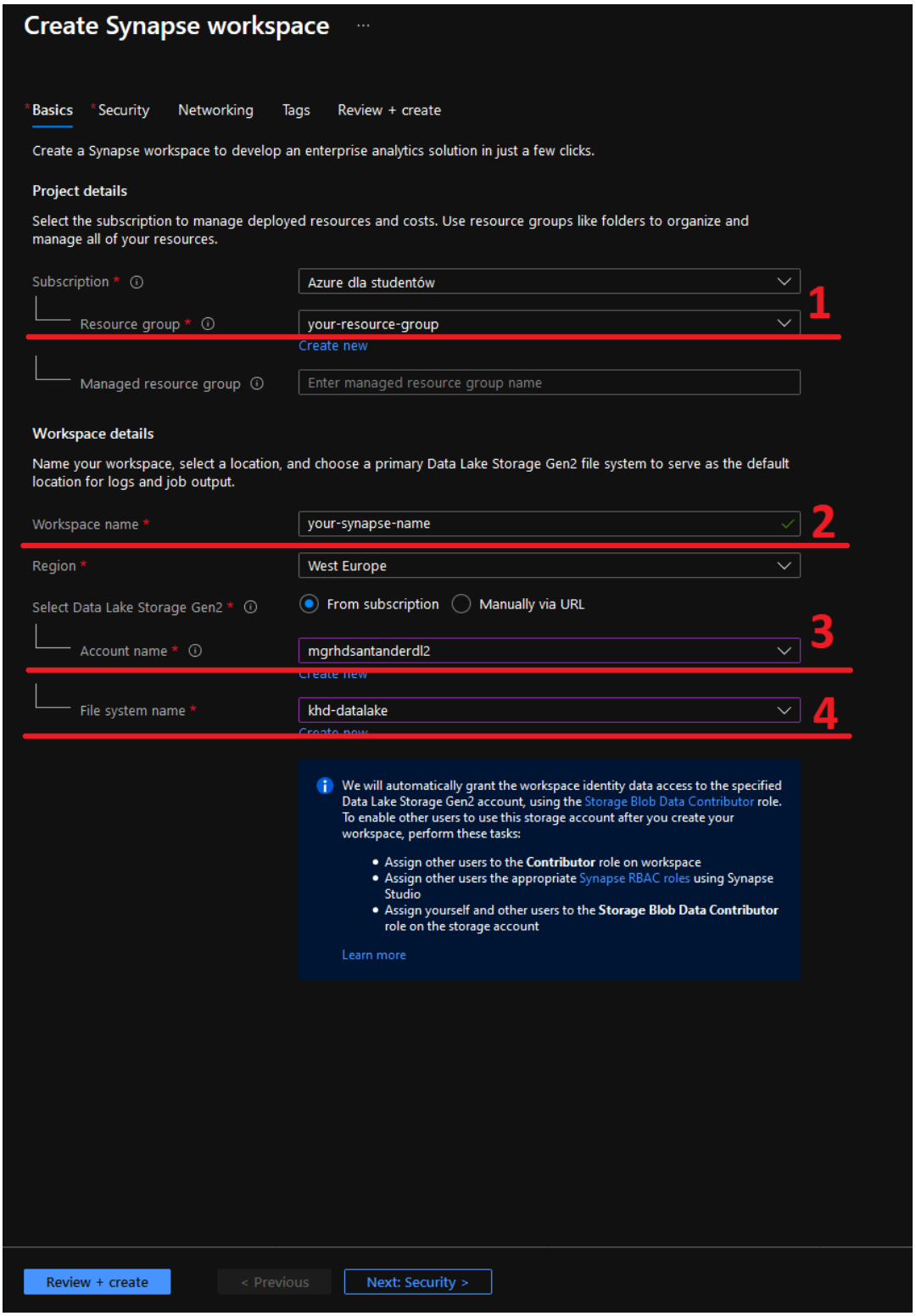
7. W celu utworzenia *Synapse workspace*:

   7.1. Wybierz utworzony wcześniej „Resource group”

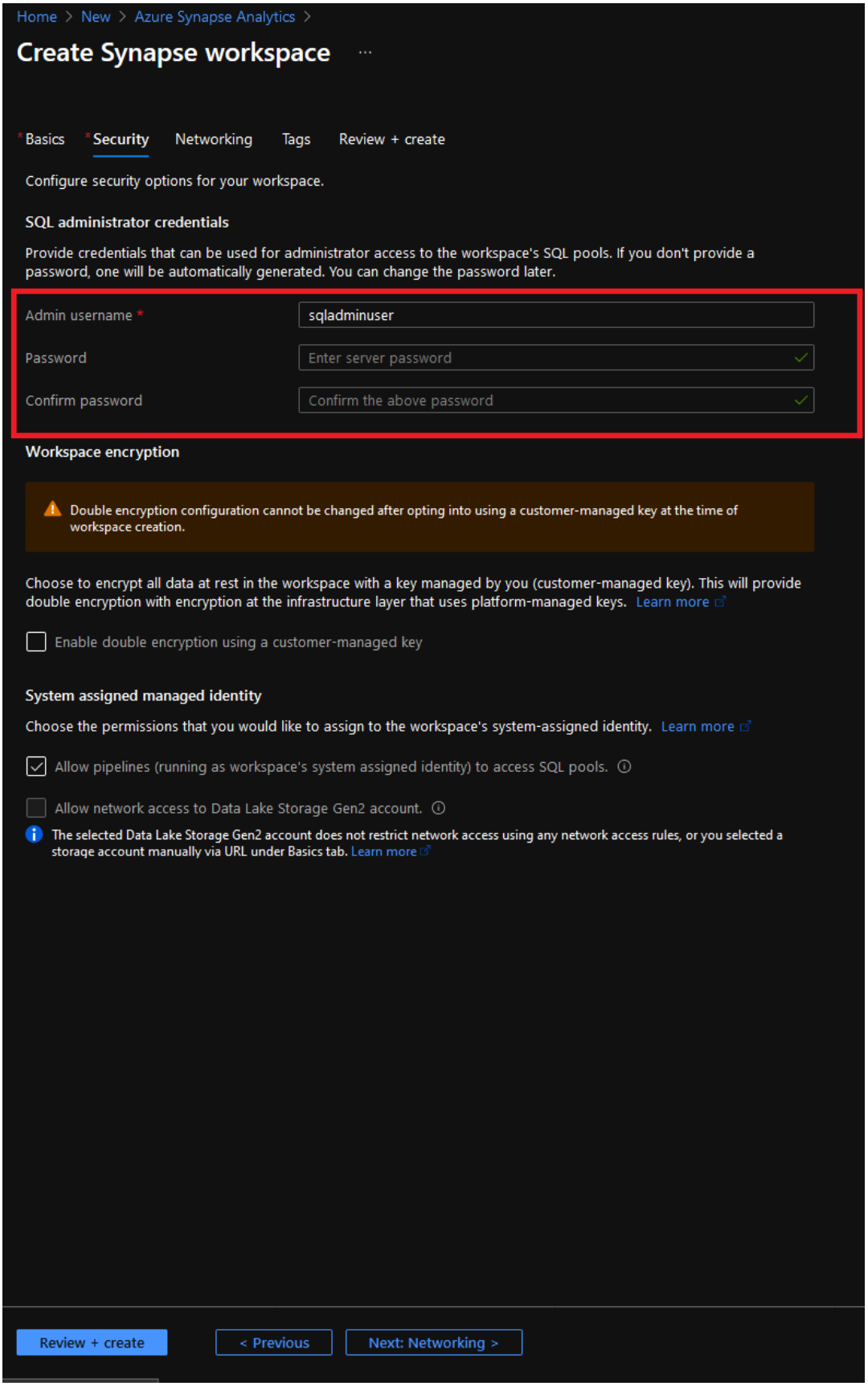
   7.2. „Workspace name” to nazwa naszego zasobu *Synapse*

   7.3. „Account name” należy wybrać z listy wcześniej utworzony *DataLake*

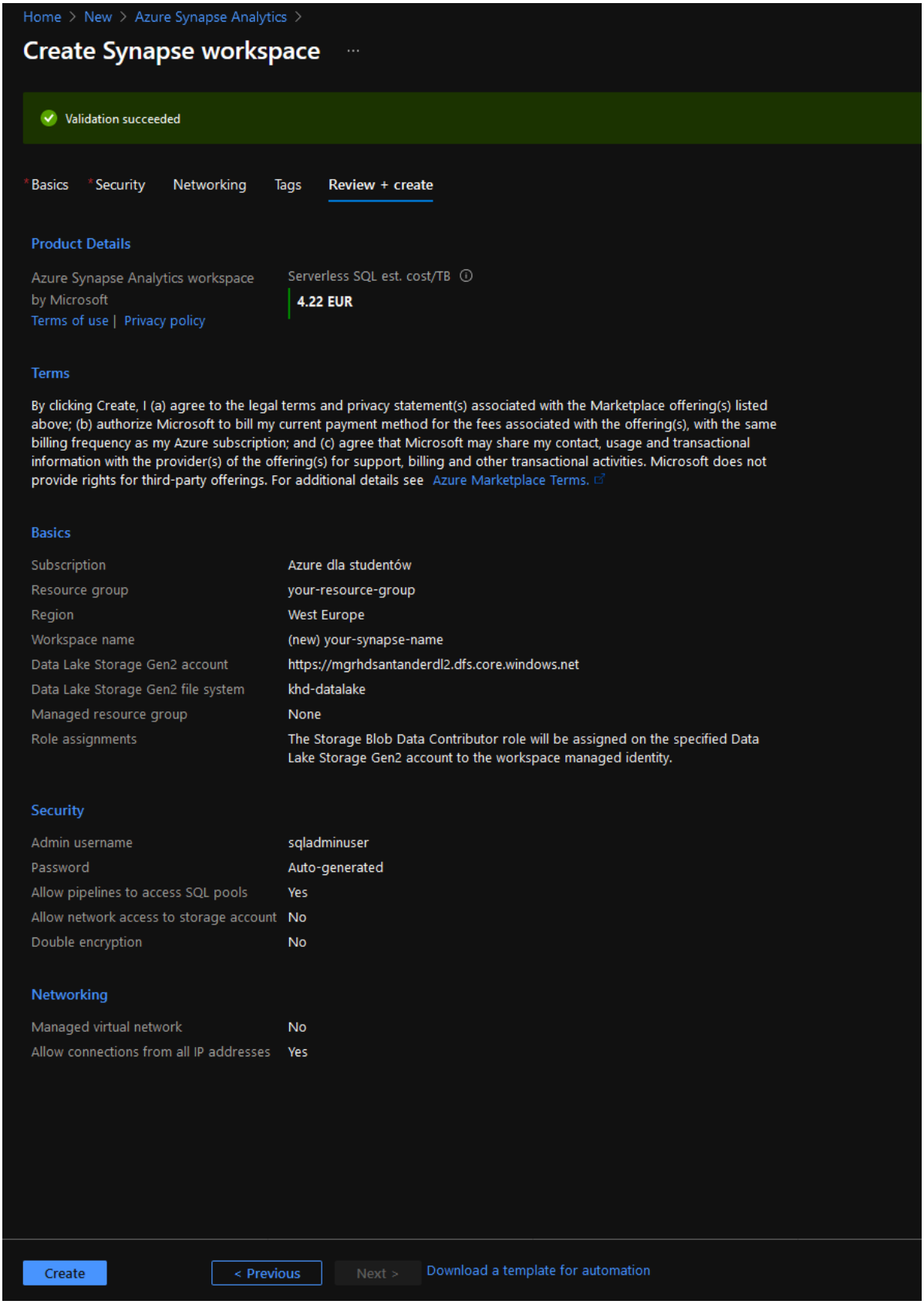
   7.4. „File system name” to nazwa kontenera znajdującego się na *DataLake*, jeżeli żadnego nie ma należy utworzyć nowy poprzez przycisk „Create new” i nadać mu nazwę



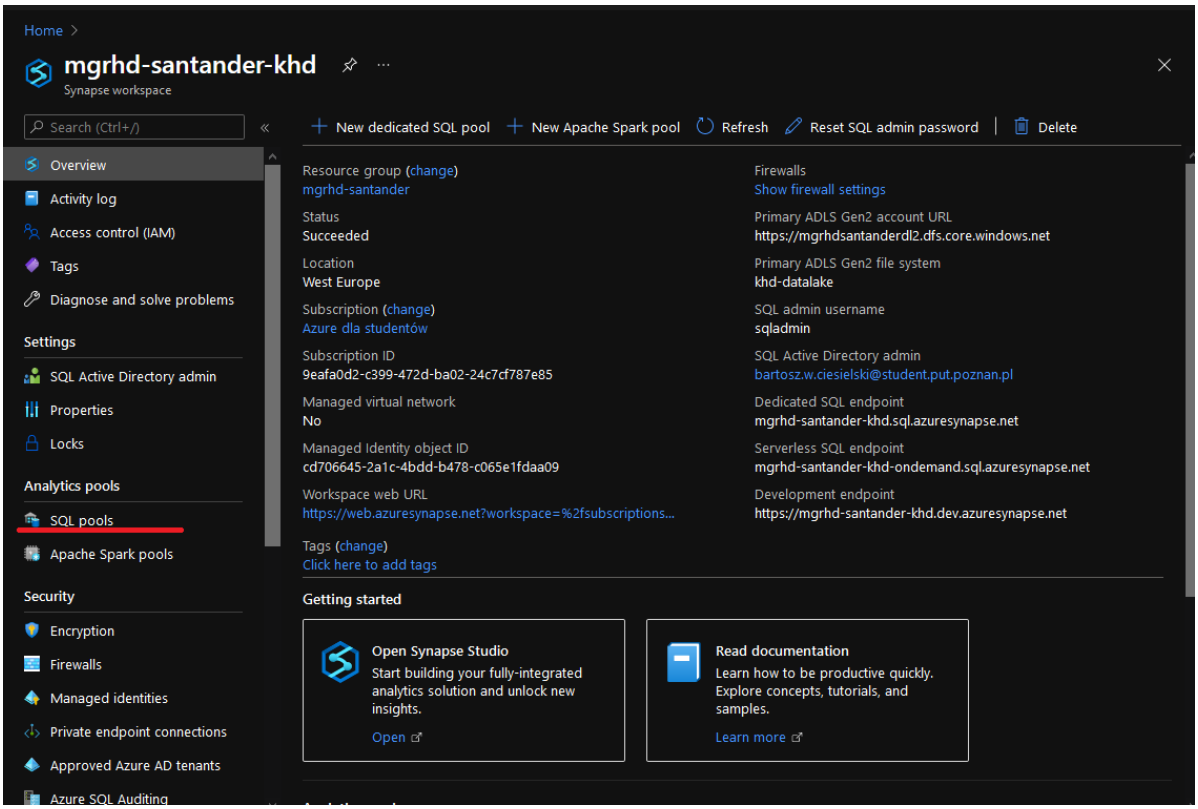
8. Należy wypełnić informacje dotyczące administratora SQL, login i hasło.



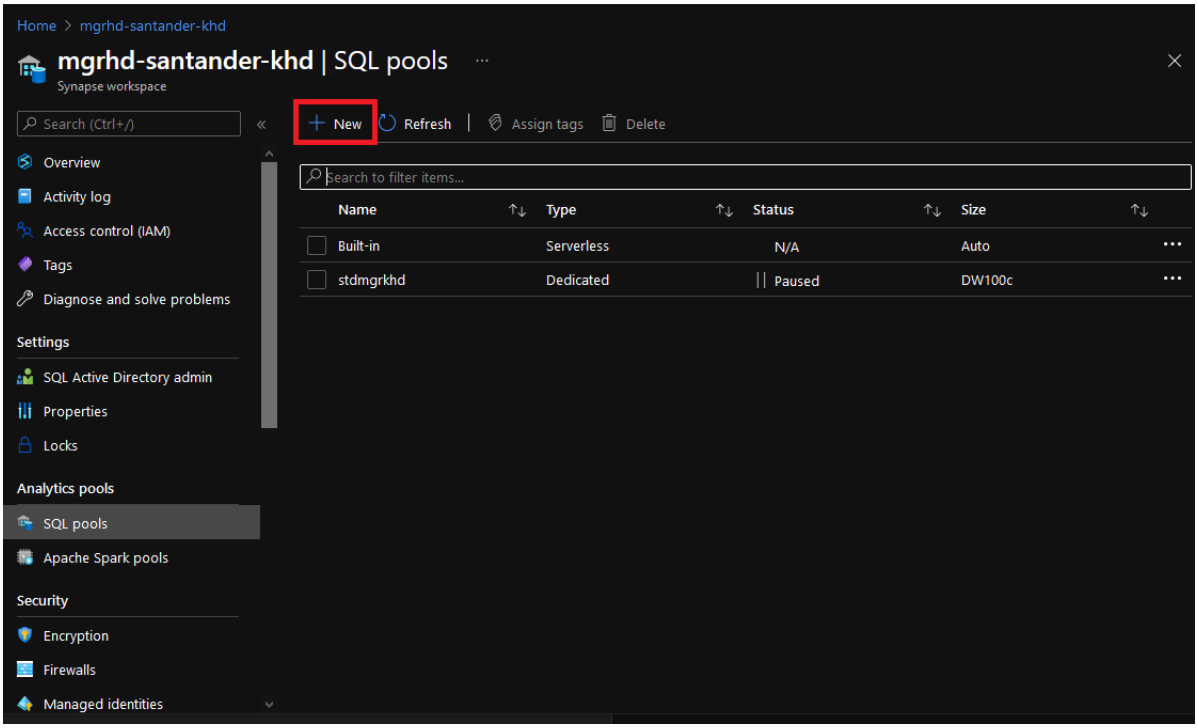
9. Przechodząc do zakładki „Review + Create” tworzymy nasz *Synapse Analytics*.



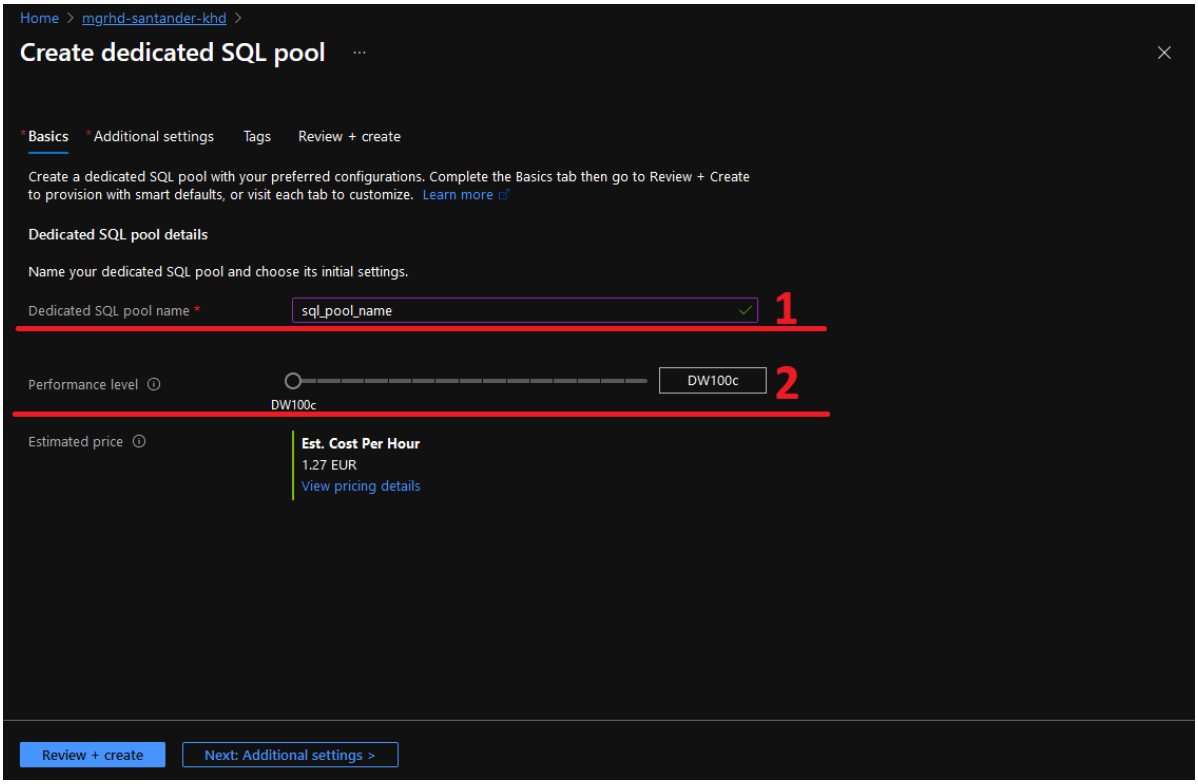
10. Wyszukujemy zasób na portalu wpisując nazwę naszego nowo utworzonego *Synapse* i po wejściu widzimy załączony na obrazku widok. Wchodzimy w podkreśloną po lewej opcje *SQL pools*.



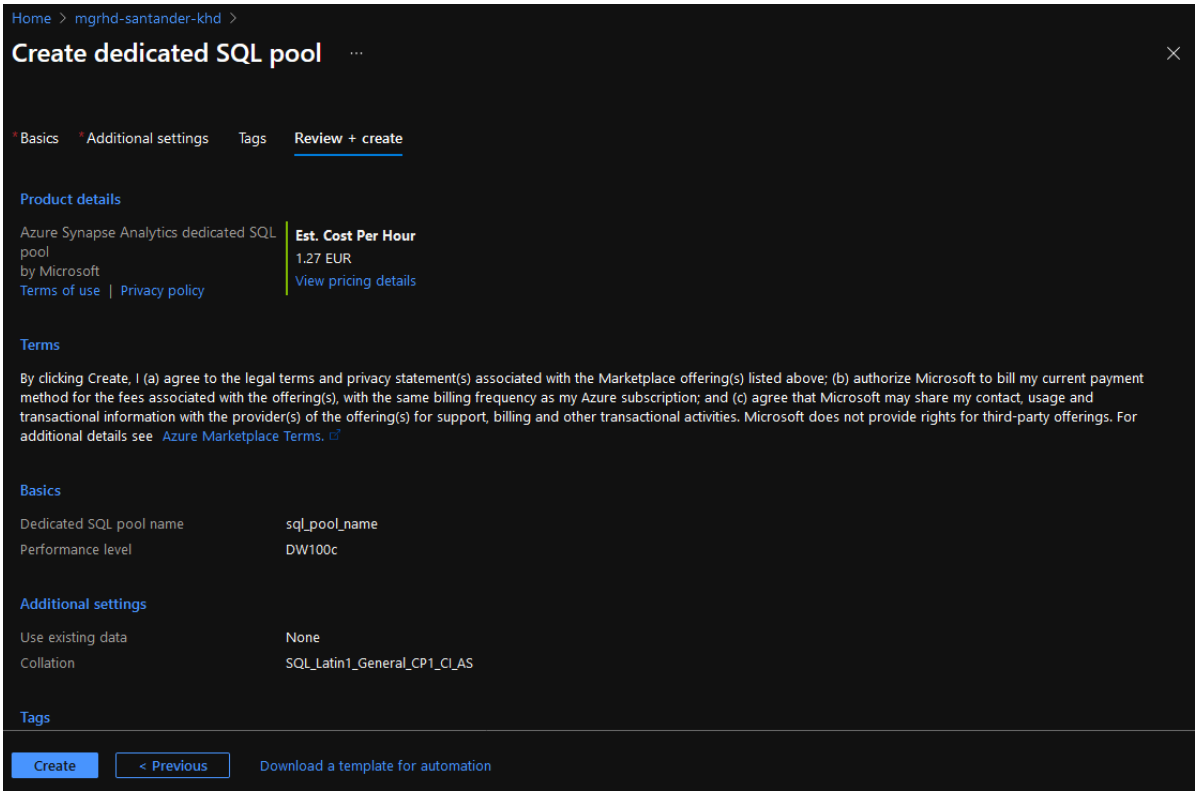
11. Klikamy „New”.



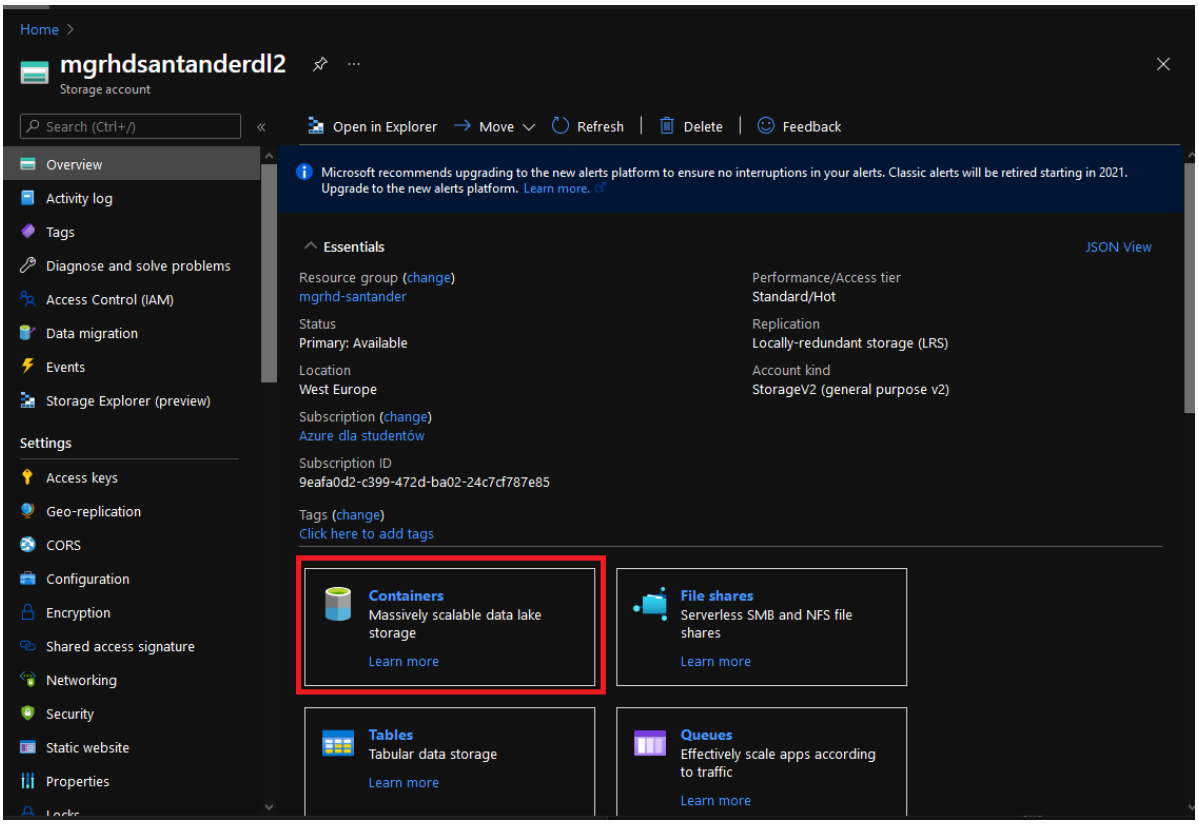
12. Nadajemy nazwę naszej bazie danych i wybieramy najtańszą opcje w celu zaoszczędzenia pieniędzy.



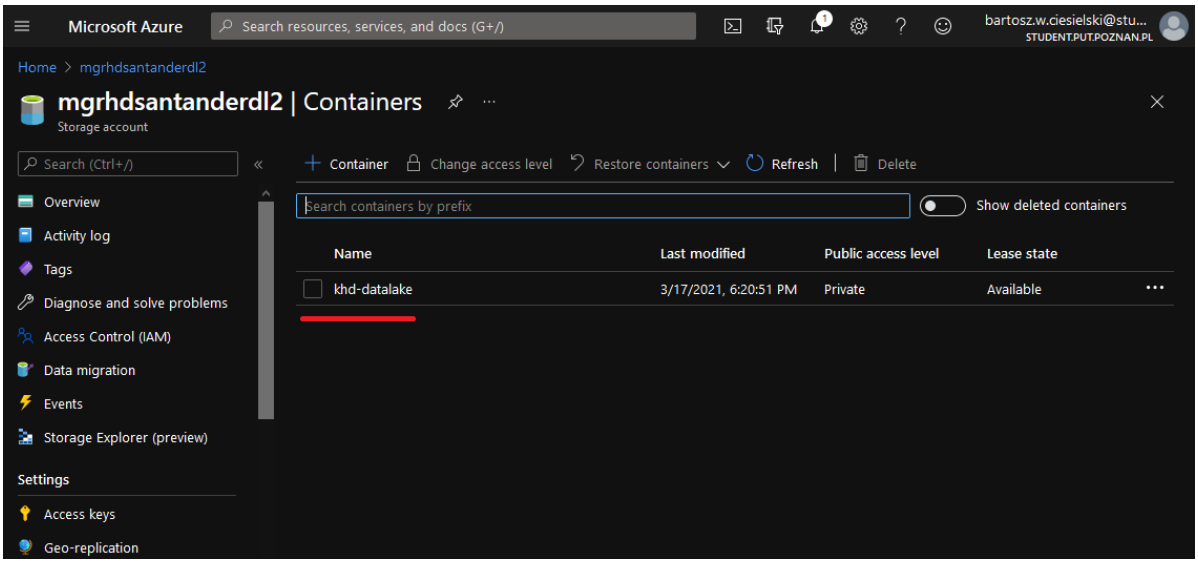
13. Reszta może pozostać bez zmian, tworzymy zasób.



14. Wchodzimy na wcześniej utworzony datalake wpisując jego nazwę w wyszukiwarce na portalu. Kolejno wybieramy zakładkę „Containers”.

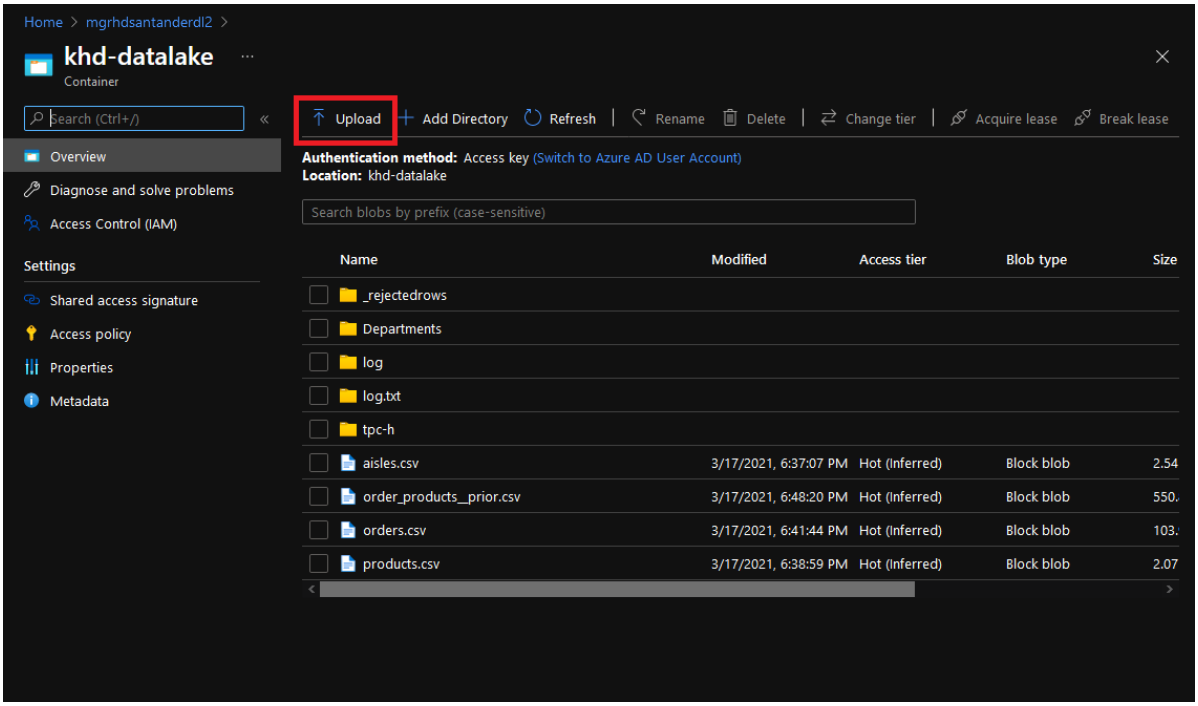


15. Wchodzimy w podpięty do *Synapse* kontener (wybrany jako „File system name” w punkcie 7).

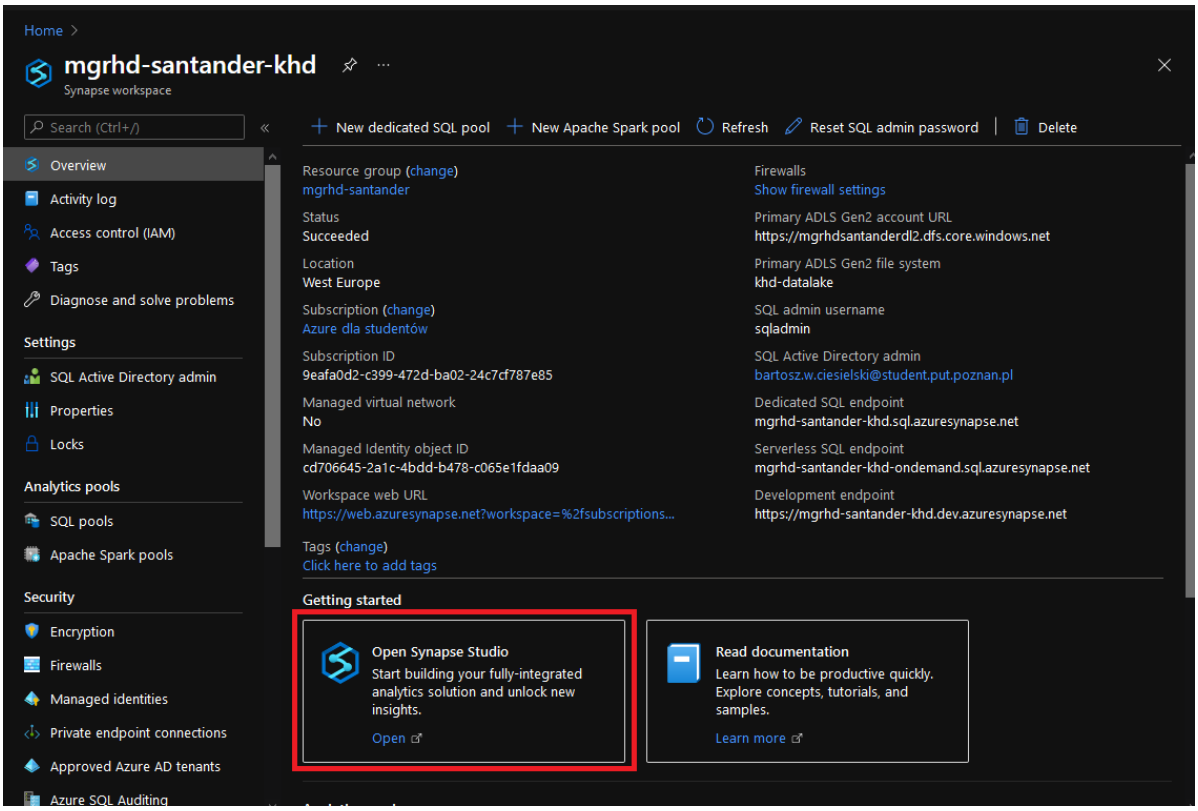


16. Wgrywamy przekazane pliki do datalake klikając „Upload”.

    Wykorzystywane w projekcie pliki można znaleźć na githubie:
    `https://github.com/Put-EngineeringThesis-soccer-prediction/SantanderAzure/tree/master/data`



17. Wracamy do *Synapse Analytics* i wybieramy zaznaczone okno. Otworzy ono interfejs, w którym zarządza się Synapsem.



18. Ekran powitalny *Synapse workspace*, na którym widzimy:

    18.1. Bazy danych oraz datalake

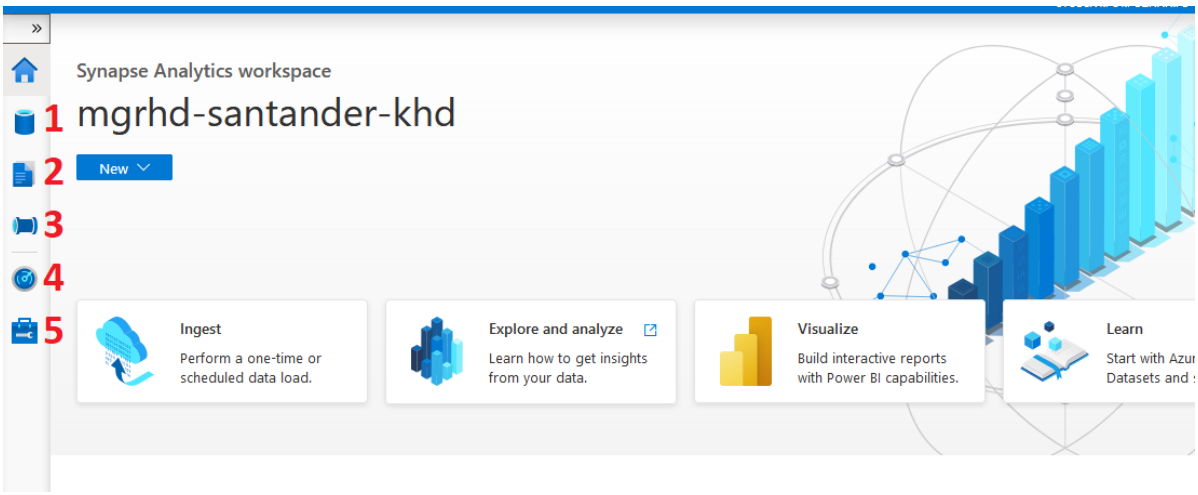
    18.2. Skrypt SQL, notatniki Spark i inne

    18.3. Okno służące do tworzenia pipeline'ów

    18.4. Monitorowanie wykonywanych zadań w Synapsie

    18.5. Zarządzanie *SQL Pools* oraz *Spark pools* jak i połączeniami z zewnętrznymi zasobami

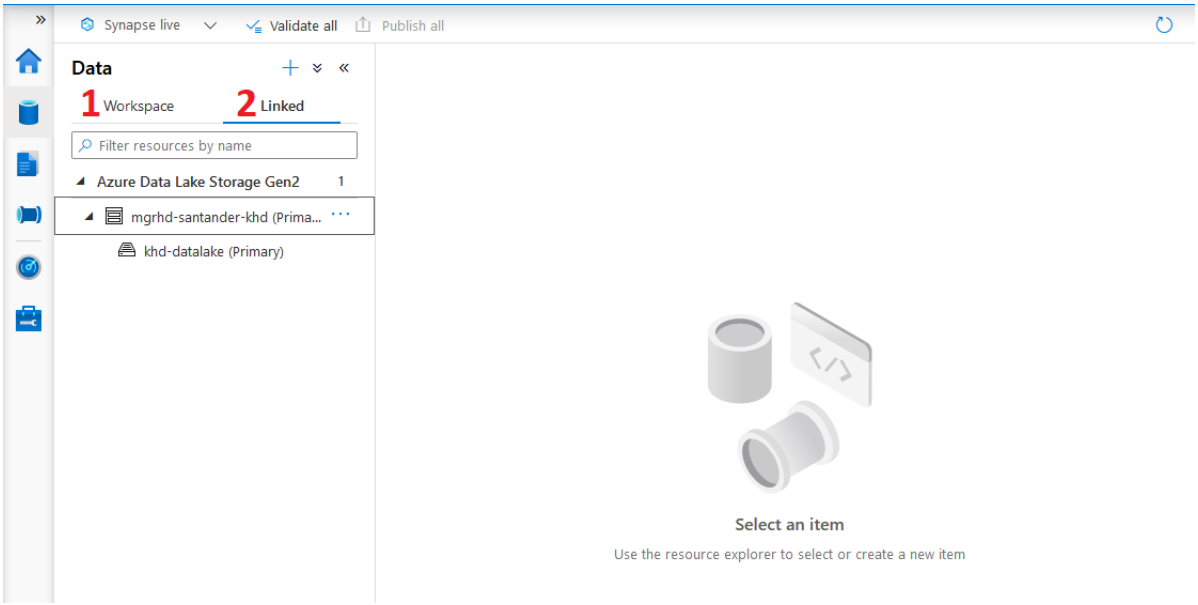
    Wybieramy zakładkę oznaczoną numerem 1



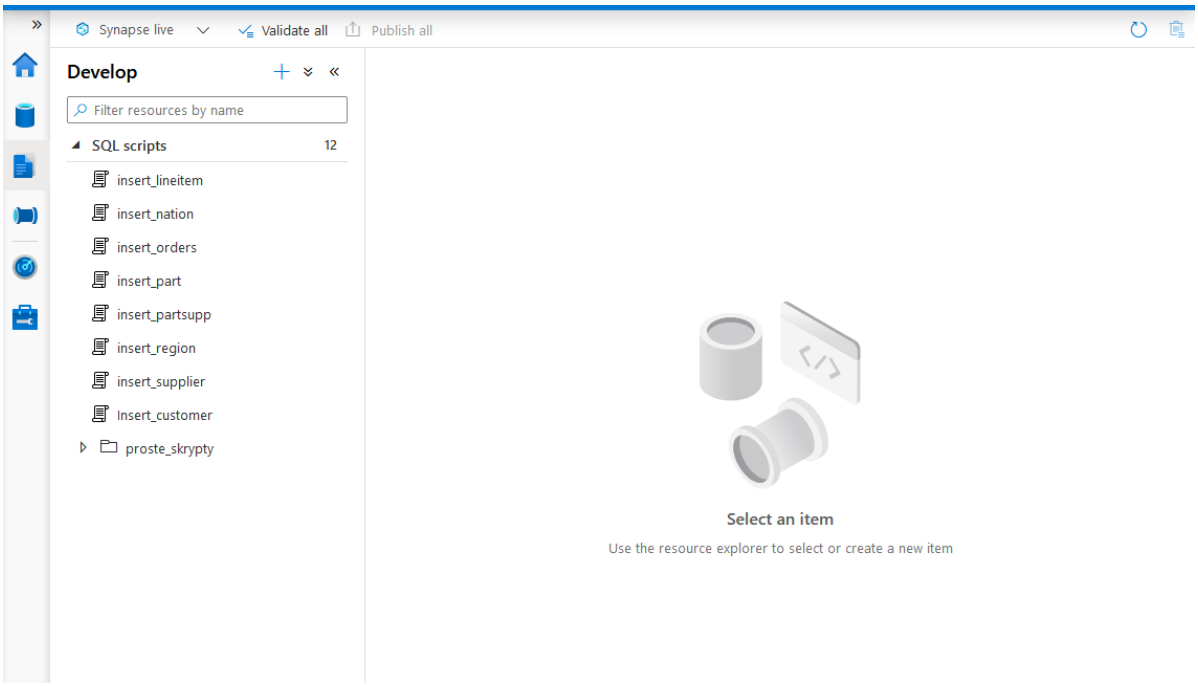
19. Tutaj mamy dostęp do naszych baz danych jak i plików załadowanych do datalake:

    19.1. Workspace zawierający bazy danych i tabele zewnętrzne

    19.2. Połączone obiekty typu datalake

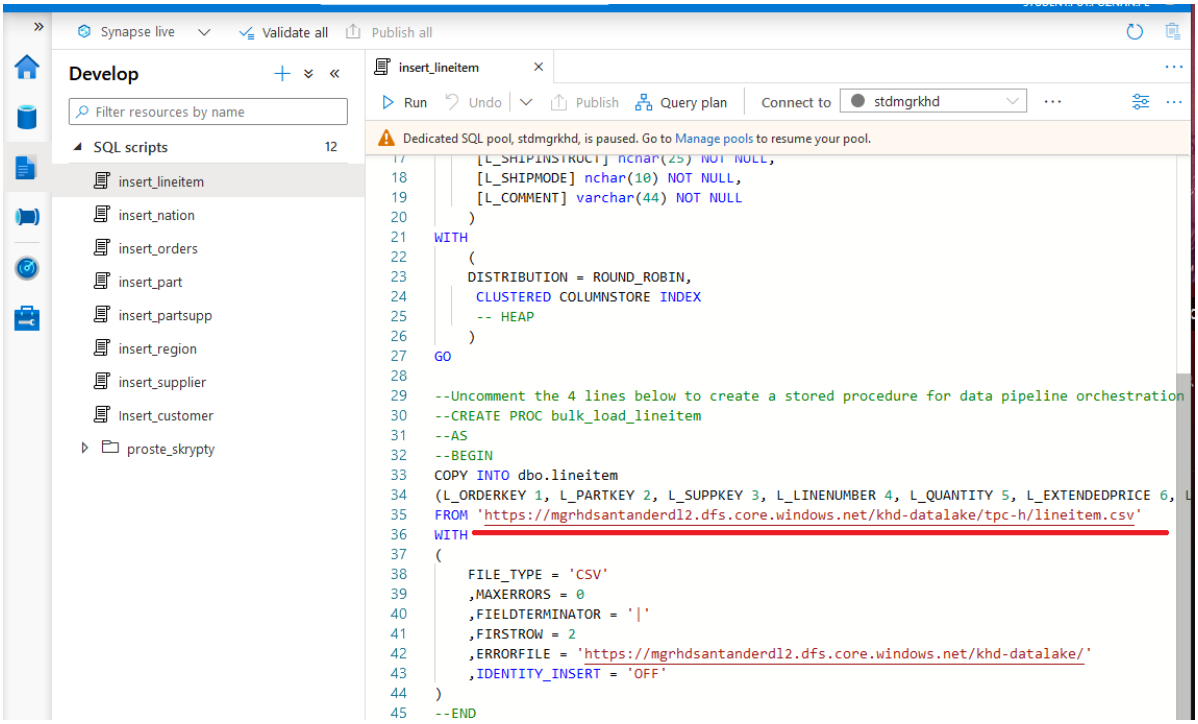


20. Zapisywane są tutaj wszelkie skrypty SQL do późniejszego użytku. Będą tu również udostępnione skrypty do ładowania danych do bazy danych.



21. W przekazanych skryptach należy zmienić podkreśloną linijkę (w każdym skrypcie), aby wskazywała na odpowiedni plik w Twoim środowisku.

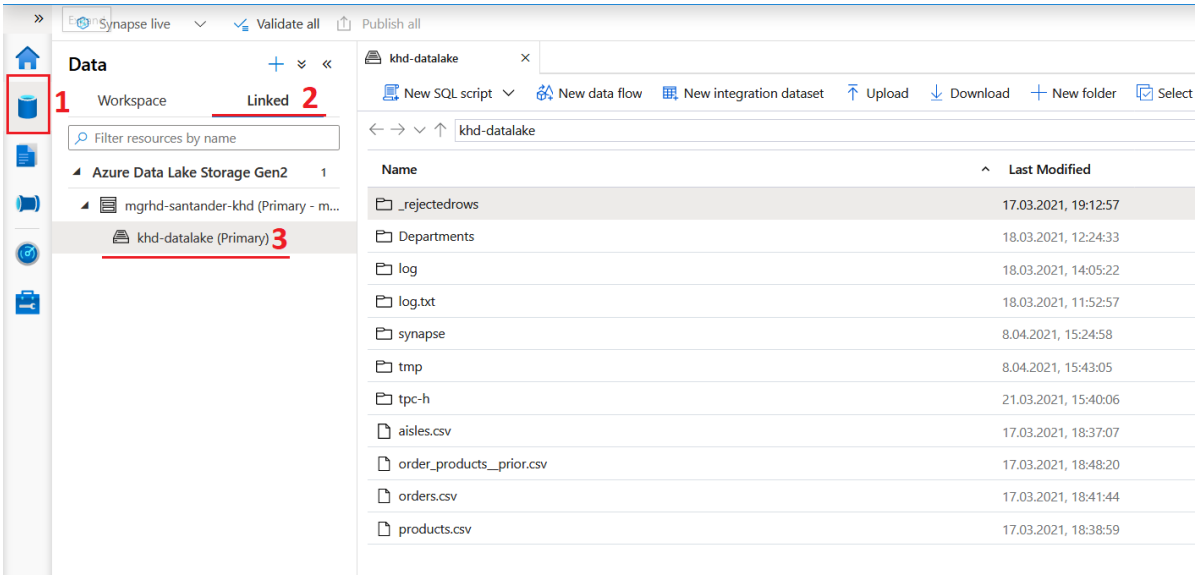
    Skrypty można znaleźć na githubie: `https://github.com/Put-EngineeringThesis-soccer-prediction/SantanderAzure/tree/master/InsertScripts`



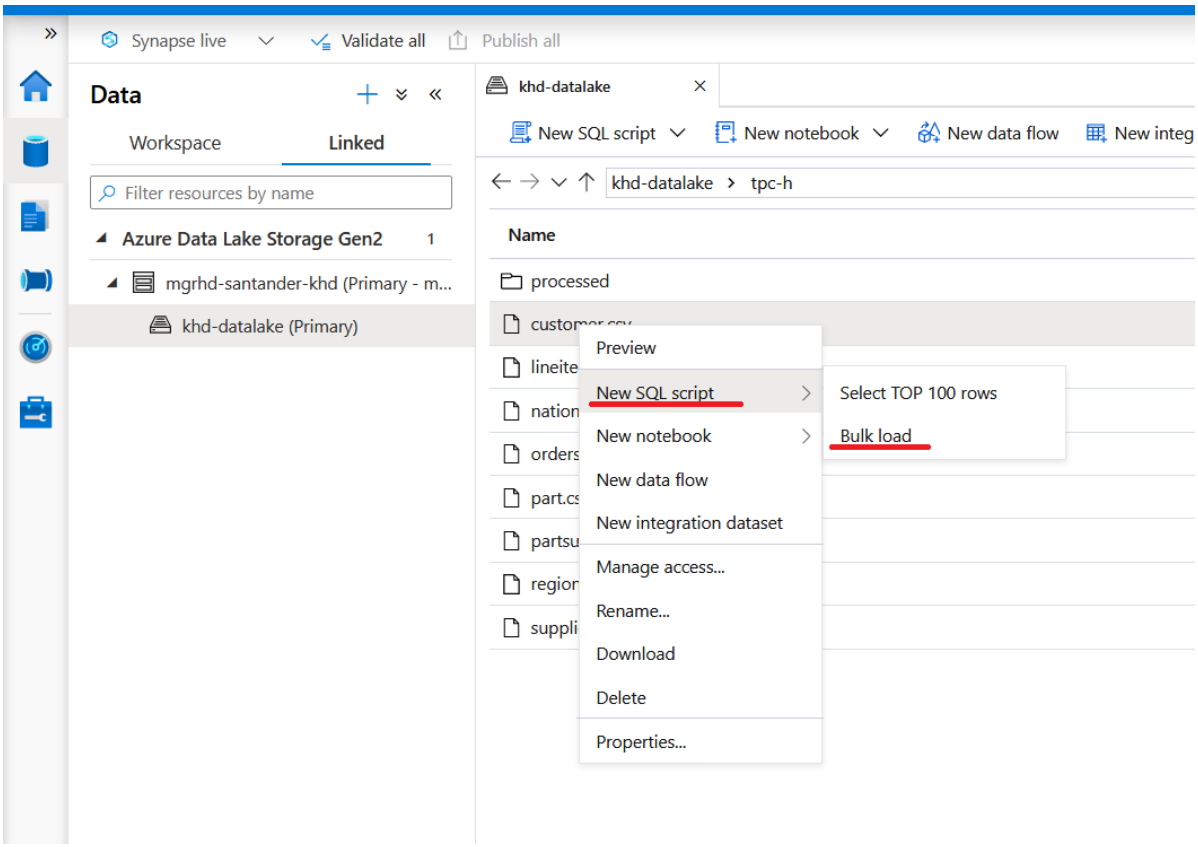
5 Ładowanie danych do bazy danych przez Azure Synapse Analytics Workbench

W celu załadowania danych do bazy danych z plików, która ma służyć jak hurtownia danych możemy użyć Azure Synapse Workbench do wygenerowania odpowiedniego kodu.

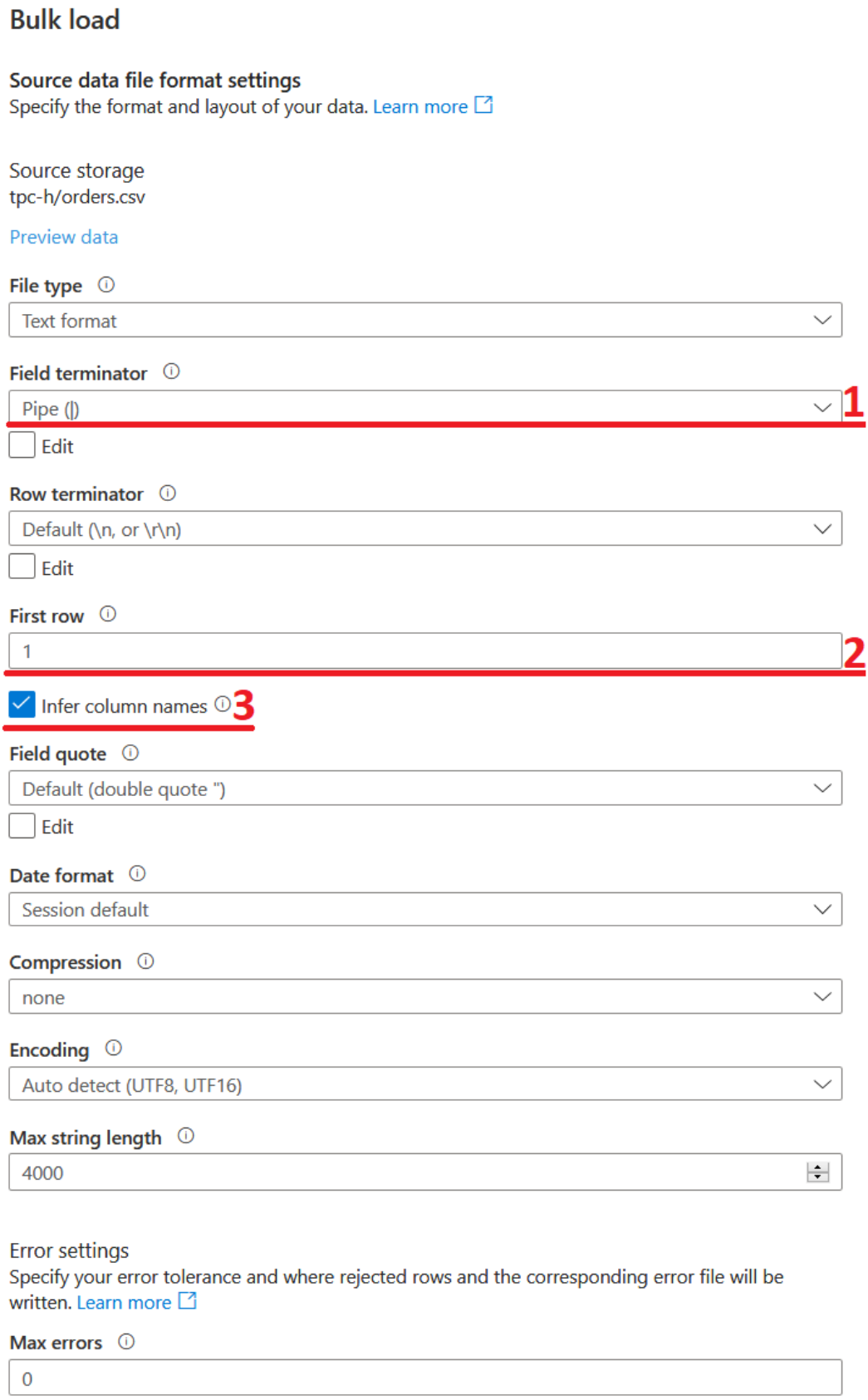
1. Aby załadować dane poprzez portal trzeba wrócić do Synapse Workbench, wejść do zakładki „data” (1) i znaleźć dane. Dane odnajdziemy w zakładce (2) Linked w podłączonym datalake'u (3).



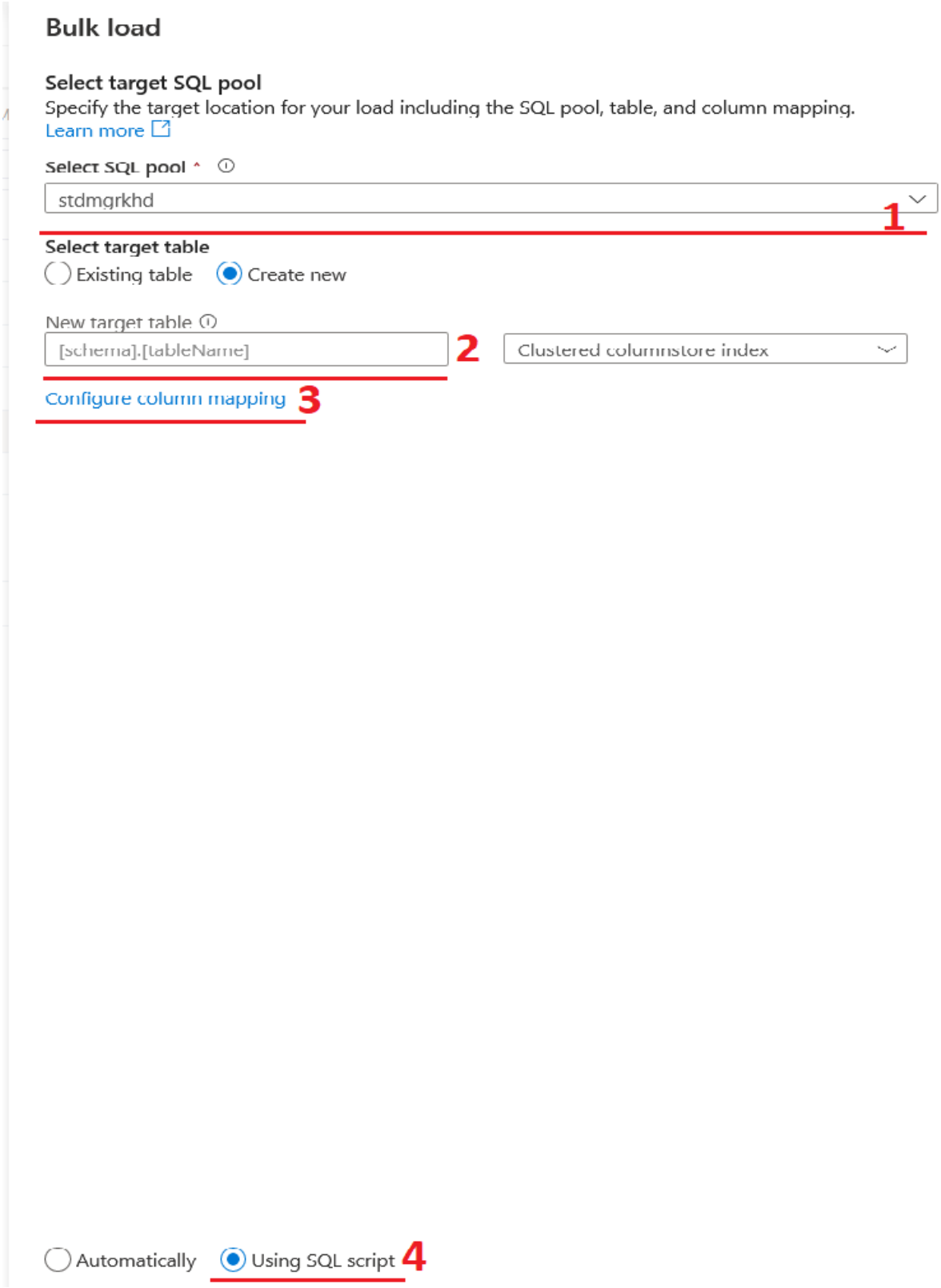
2. Wybieramy przygotowany plik csv, klikamy prawym przyciskiem na wybrany plik, wchodzimy w „New SQL script” i dalej „Bulk load”.



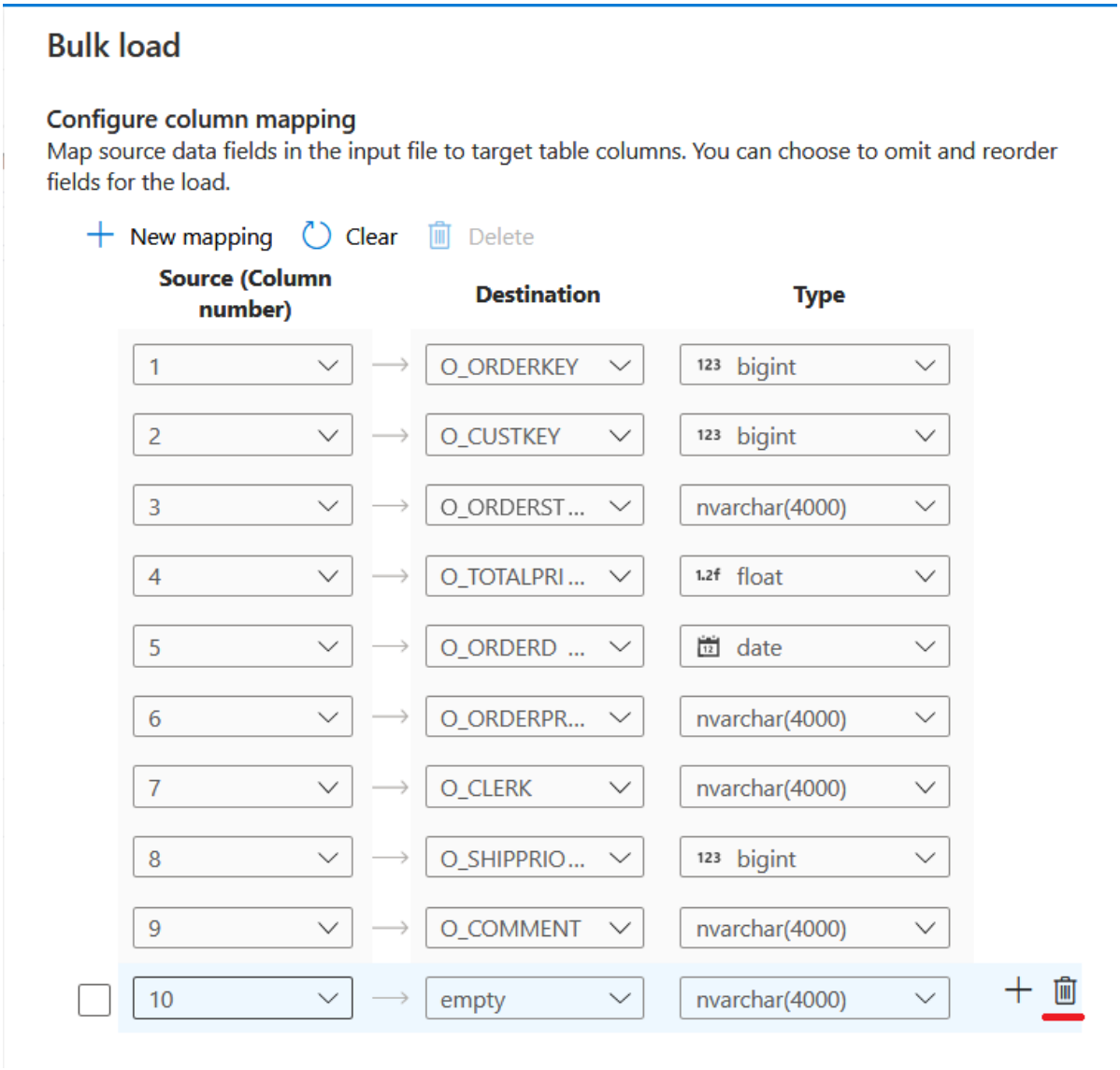
3. Po prawej stronie pojawi się widoczne okienko. Dla naszych danych wybieramy (1) „Pipe(|)”, „First row” (2) ustawiamy na 1 i zaznaczamy „Infer column names” (3). Następnie klikamy dalej.



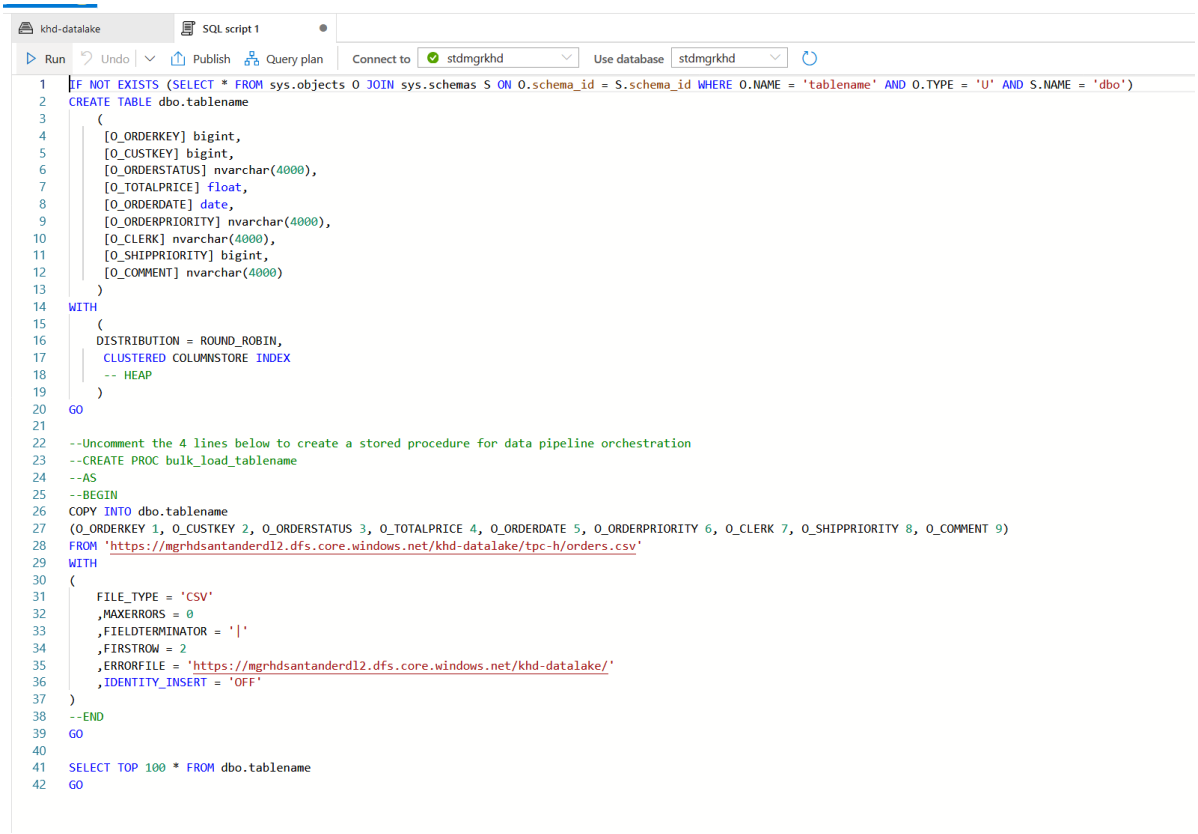
4. Wybieramy naszą dedykowaną bazę SQL, która jest naszą hurtownią danych (1), nadajemy nazwę tabeli (2) (najlepiej zgodnie z nazwą pliku), upewniamy się że jest wybrane „Using SQL script” (4), a następnie klikamy „configure column mapping” (3)



5. Tutaj możemy określić nazwy kolumn oraz ich typy w tabeli. Dla naszych danych należy również usunąć kolumn empty, która wynikała z sposobu generowania danych i posiadała zbędną pustą kolumnę. Typy danych dla każdej z tabel są określone na schemacie. Po ustawieniu wszystkich wartości wracamy do poprzedniego okna i tworzymy skrypt.



6. Utworzony skrypt służący do ładowania danych z pliku do tabeli SQL.



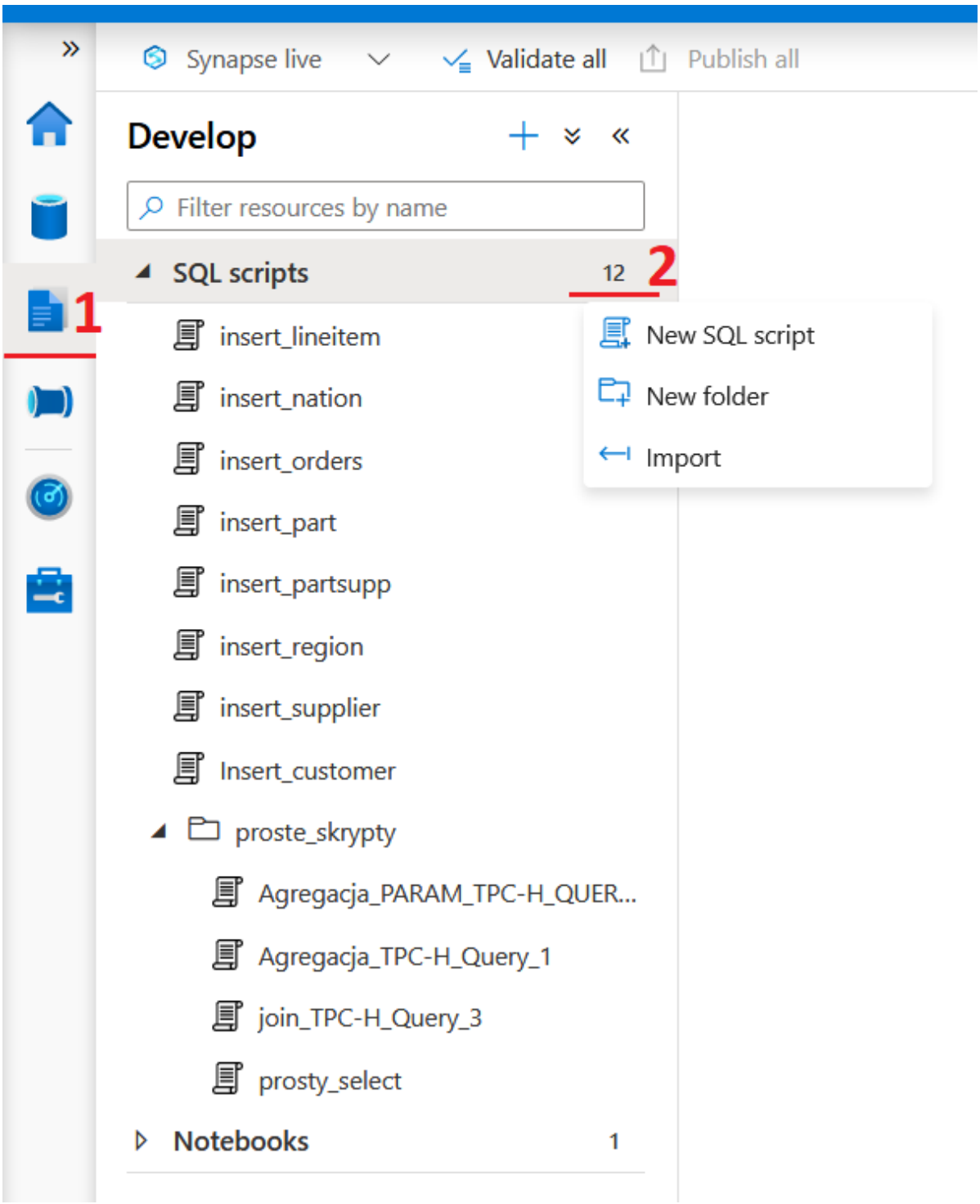
6 Przykładowe zapytania

Przygotowane dane w bazie danych można odpytywać językiem T-SQL.

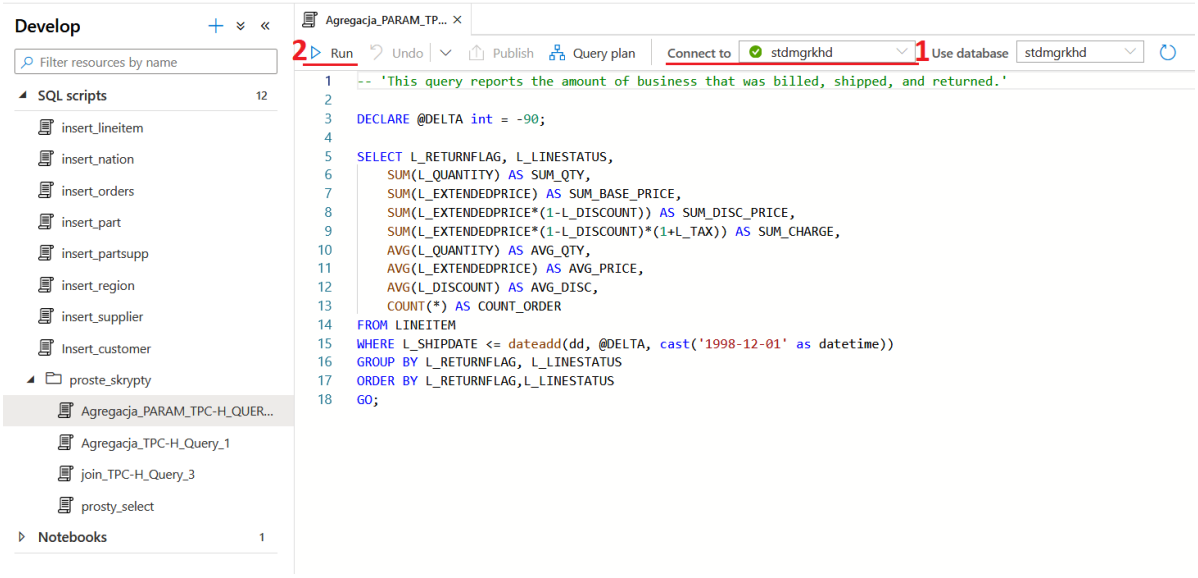
1. W Azure Synapse Analytics Workbench przechodzimy do zakładki „Develop” (1) i w tej zakładce możemy utworzyć nowy skrypt lub załadować przygotowany plik SQL (2 - można również kliknąć w symbol plus powyżej)

   Przykładowe zapytania można znaleźć na githubie:

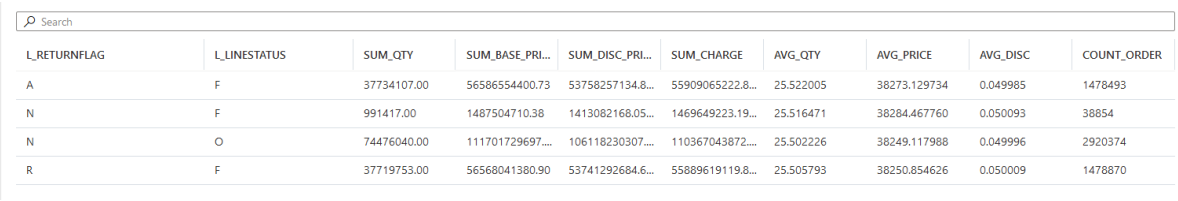
   `https://github.com/Put-EngineeringThesis-soccer-prediction/SantanderAzure/tree/master/Sample_Querries`



2. W celu uruchomienia przykładowego skryptu należy wybrać bazę danych, na której zostanie on uruchomiony (1) i kliknąć przycisk „Run” (2)



3. Poniżej widoczna jest odpowiedź na zapytanie utworzone w poprzednim punkcie.

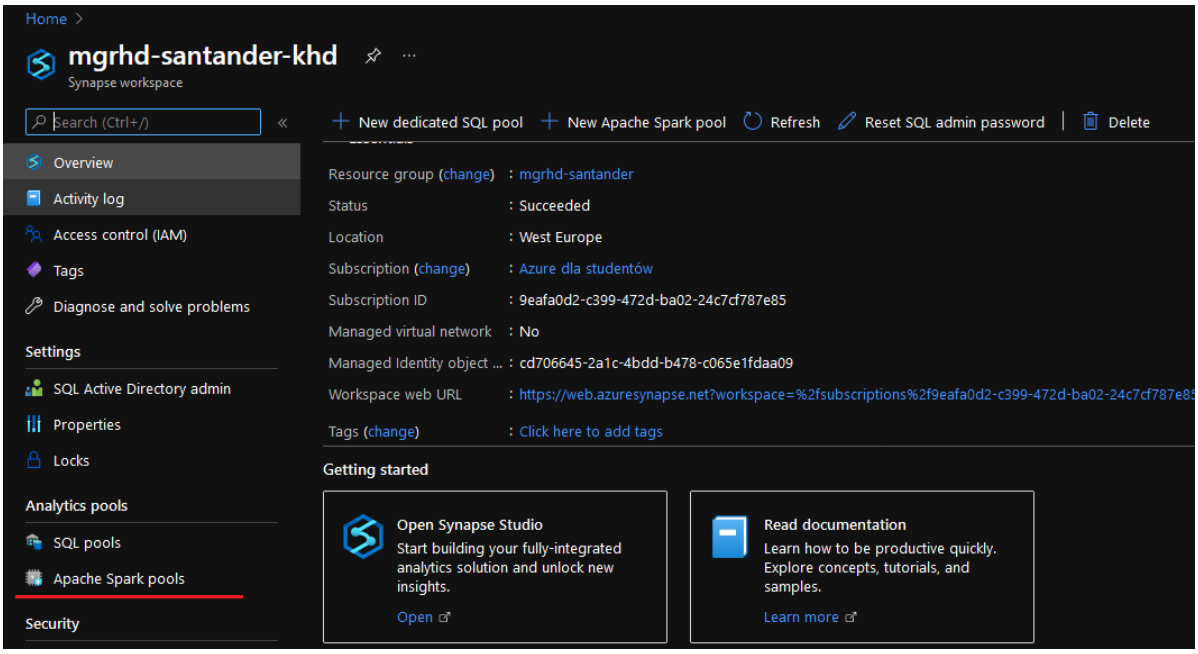


L_RETURNFLAG	L_LINESTATUS	SUM_QTY	SUM_BASE_PRI...	SUM_DISC_PRI...	SUM_CHARGE	AVG_QTY	AVG_PRICE	AVG_DISC	COUNT_ORDER
A	F	37734107.00	56586554400.73	53758257134.8...	55909065222.8...	25.522005	38273.129734	0.049985	1478493
N	F	991417.00	1487504710.38	1413082168.05...	1469649223.19...	25.516471	38284.467760	0.050093	38854
N	O	74476040.00	111701729697....	106118230307....	110367043872....	25.502226	38249.117988	0.049996	2920374
R	F	37719753.00	56568041380.90	53741292684.6...	55889619119.8...	25.505793	38250.854626	0.050009	1478870

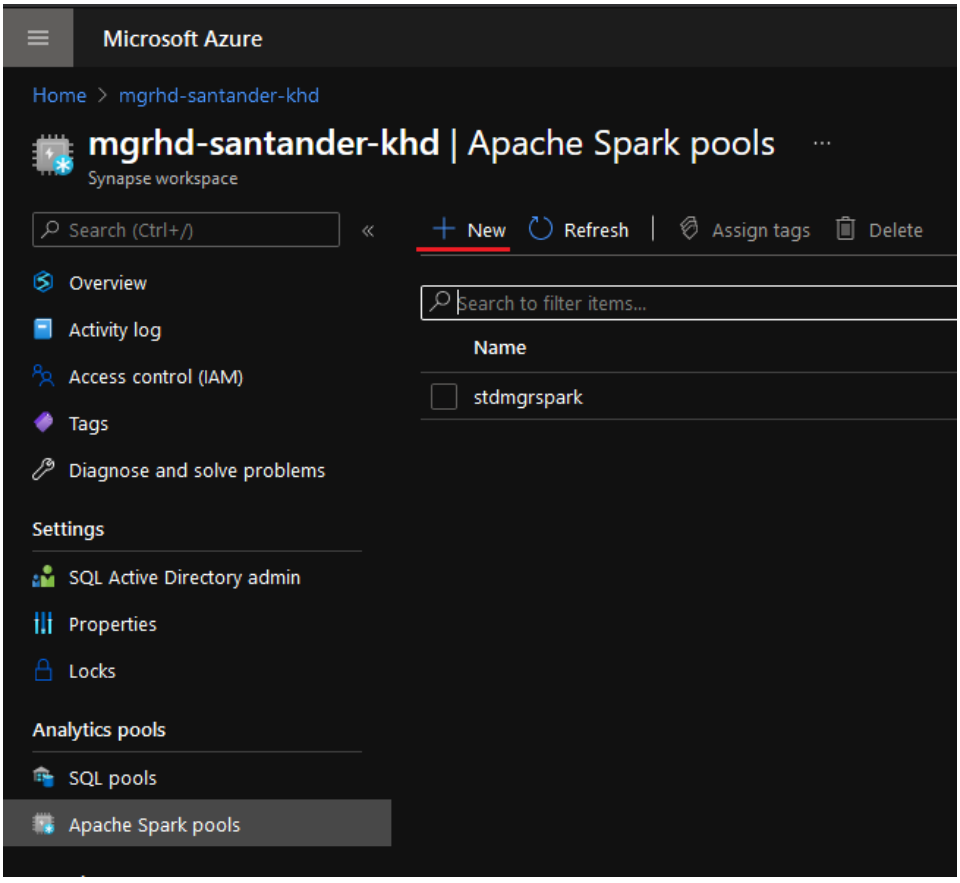
7 Przetwarzanie danych przy użyciu Apache Spark

Azure Synapse Analytics udostępnia możliwość tworzenia skryptów przy użyciu pythona w notebookach sparkowych. Wykorzystywana jest w tym celu biblioteka PySpark. Takie skrypty dają możliwość przetwarzania danych przechowywanych w hurtowni danych zarówno Synapse, jak i w Data Lake. Można je ze sobą łączyć i wynik zapisywać ponownie w wybranym miejscu w Data Lake'u.

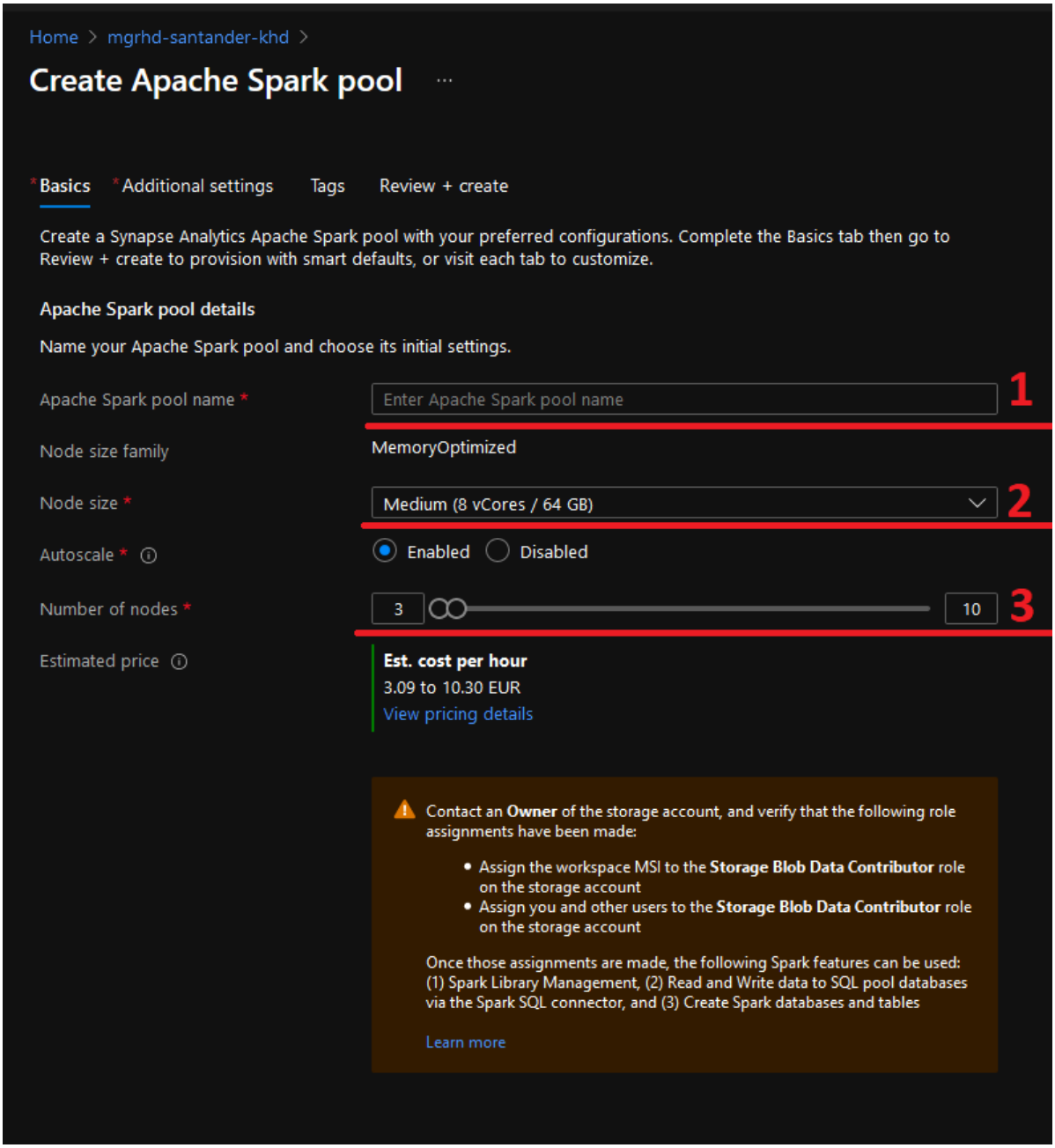
1. Aby umożliwić przetwarzanie Spark, trzeba utworzyć tzw. „Apache Spark pool". W tym celu, mając otwarty zasób Azure Synapse Analytics, przechodzimy do zakładki „Apache Spark pools".



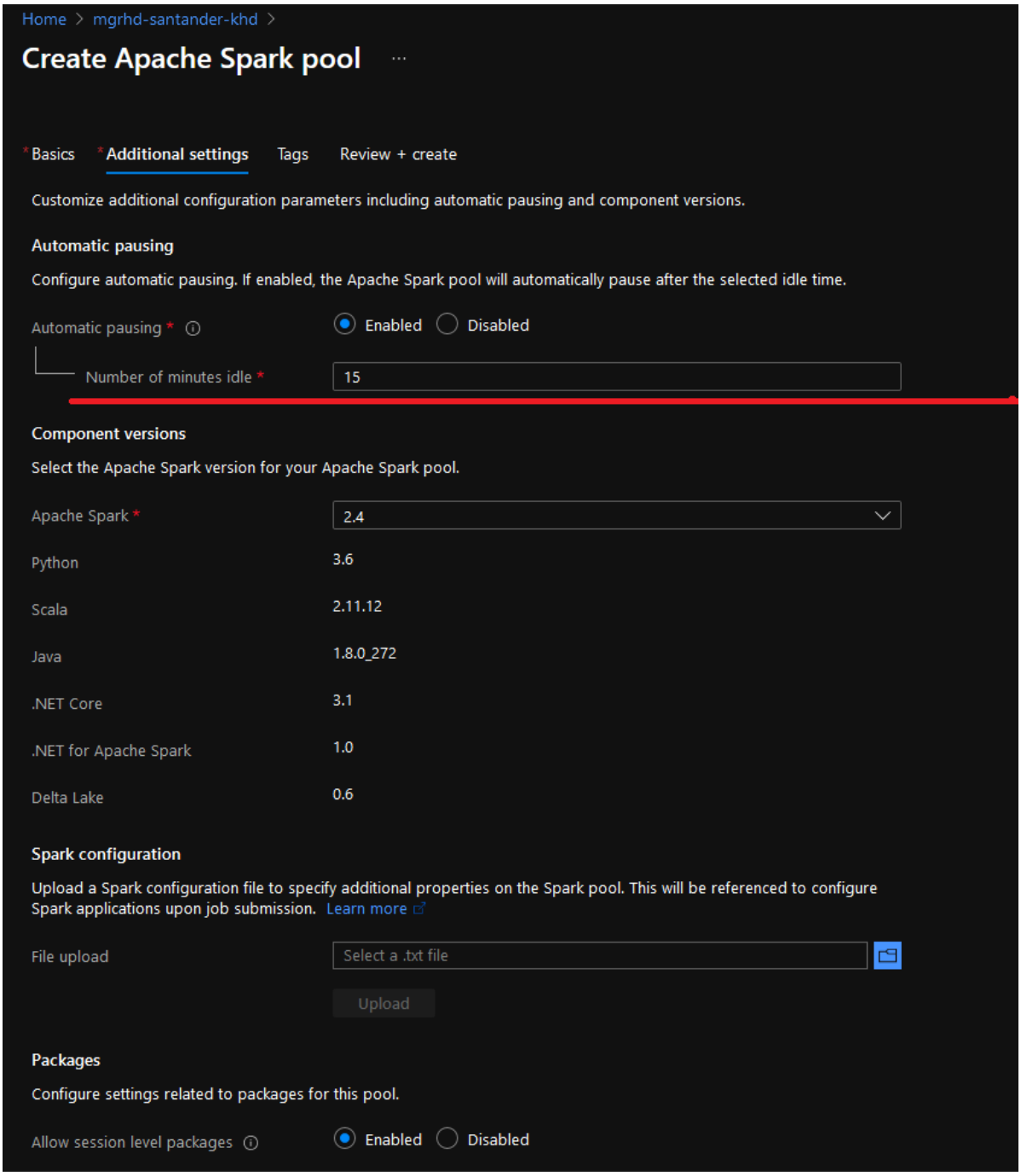
2. Klikamy przycisk „New".



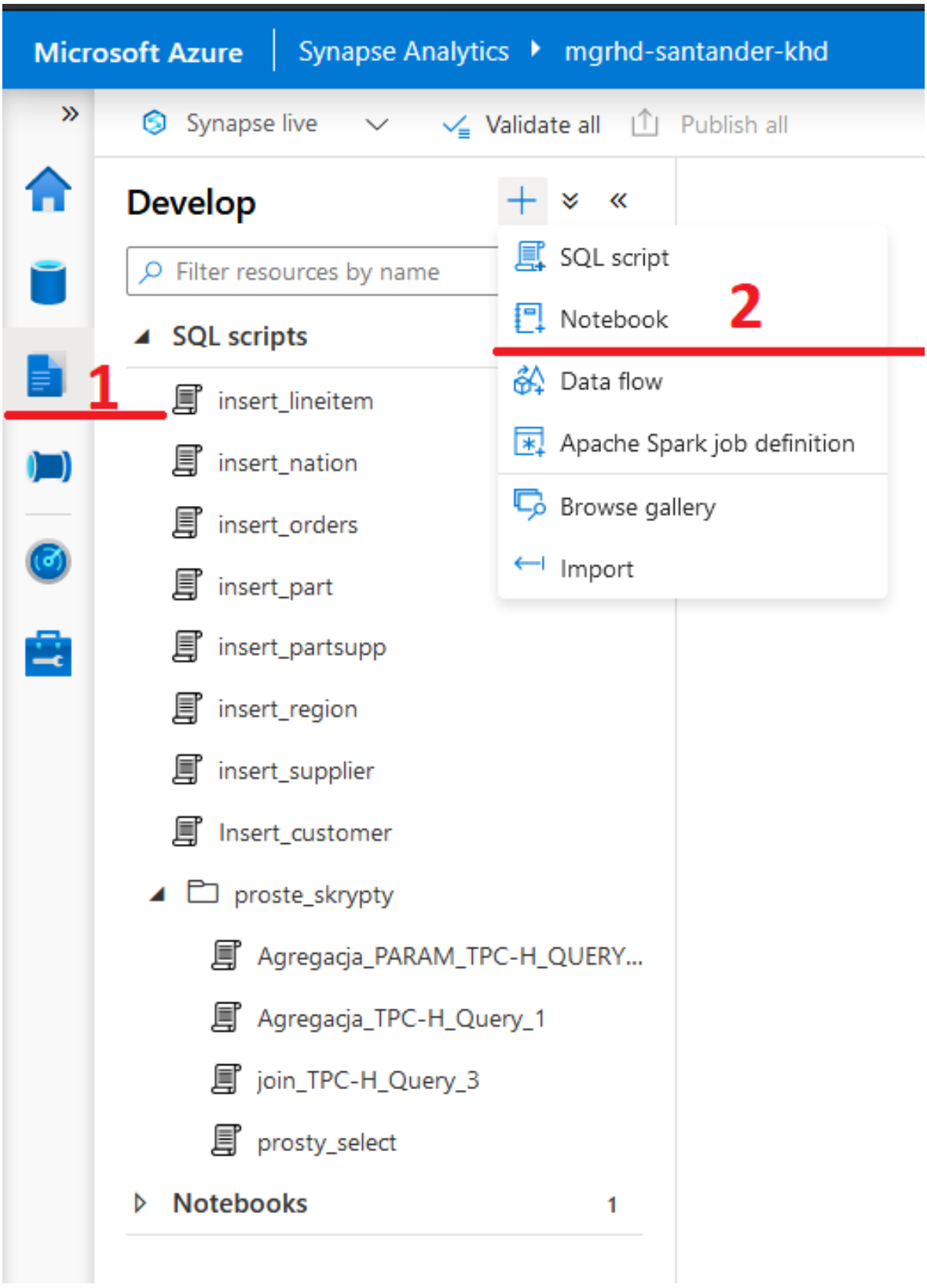
3. Tworząc Apache Spark pool należy podać jego dowolną nazwę (1), wielkość węzła klastra (2) oraz liczbę węzłów wraz z wyborem autoskalowania (3) (dynamiczna zmiana liczby węzłów w zależności od obciążenia). Dla zminimalizowania kosztów wybieramy najmniejszy węzeł oraz minimalną liczę węzłów. Wyłączamy również opcje autoskalowania.



4. W opcjach dodatkowych możemy określić po ilu minutach uruchomiony klaster zostanie wyłączony w przypadku bezczynności. Służy to temu, aby nieużywany klaster nie generował niepotrzebnych kosztów.



5. Po utworzeniu klastra Apache Spark, przechodzimy do Azure Synapse Analytics Workbench do zakładki „Develop” (1). Wybieramy plus (2) w celu utworzenia Notebook'a, w którym umieszczony zostanie kod przetwarzający dane przy użyciu sparka.



6. Poniżej widoczny jest kod ładujący dane z pliku znajdującego się na podpiętym DataLake.

   Pierwszy parametr to ścieżka do pliku w postaci „ABFSS Path". Jest do referencja do DataLake'a, a na końcu ścieżka przypominająca strukturę folderów. W przykładzie wczytujemy plik tpc-h/region.csv. (Ścieżka ABFSS może być znaleziona w parametrach pliku, tzn. w oknie data, klikając prawym przyciskiem myszy na plik -> properties.)
   Kolejny parametr to format pliku ustawiony na csv, czy zawiera nagłówek oraz jaki separator został użyty w przypadku pliku csv. Dostępne są również inne formaty plików, np. *json* czy *parquet*. Następnie usuwamy kolumnę „empty" przy użyciu metody „drop". W ten sposób tworzymy Spark Dataframe, który wyświetlamy w następnej linijce kodu.

```
region_df = spark.read.load(
'abfss://khd-datalake@mgrhdsantanderdl2.dfs.core.windows.net/tpc-h/region.csv',
format='csv',
header=True,
sep='|').drop('empty')

region_df.show()
```

```
region_df = spark.read.load(
'abfss://khd-datalake@mgrhdsantanderdl2.dfs.core.windows.net/tpc-h/region.csv',
format='csv',
header=True,
sep='|').drop('empty')

region_df.show()
```

7. W osobnej komórce używamy *%%Spark* w celu zaznaczenia, że będziemy korzystać z języka Scala. Przez ograniczenia środowiska dostęp do bazy danych najprościej uzyskać poprzez użycie kodu zaimplementowanego w Scali. Zaimportowane biblioteki umożliwiają bezpośredni dostęp do tabel w bazach danych znajdujących się w obrębie Azure Synapse Analytics. Zaraz po zaimportowaniu bibliotek odczytujemy tabelę *nation* i zapisujemy jej zawartość do tabeli tymczasowej, aby uzyskać do niej dostęp w następnej częsci kodu.

```
%%spark

import com.microsoft.spark.sqlanalytics.utils.Constants
import org.apache.spark.sql.SqlAnalyticsConnector._
val df = spark.read.sqlanalytics("stdmgrkhd.dbo.nation")
df.write.mode("overwrite").saveAsTable("tempTable_Nation")
```

8. W ostatniej komórce kodu ładujemy najpierw do Spark Dataframe zawartość utworzonej wcześniej tabeli tymczasowej (spark stosuje tzw. „Lazy Loading", więc dopóki nie zostaną wykonane na danych konkretne akcje, dane nie są pobierane). Łączymy wczytane wcześniej dane z pliku csv z danymi z tabeli pobranymi z hurtowni danych. Wybieramy interesujące nas kolumny, zmieniamy ich nazwy i sortujemy po nazwie regionu. Tak uzyskane wyniki wyświetlamy przy pomocy funkcji „display", która uruchomi przetwarzanie, po czym zapisujemy zawartość do pliku parquet na tym samym datalake'u, z którego pobraliśmy plik *region.csv*.

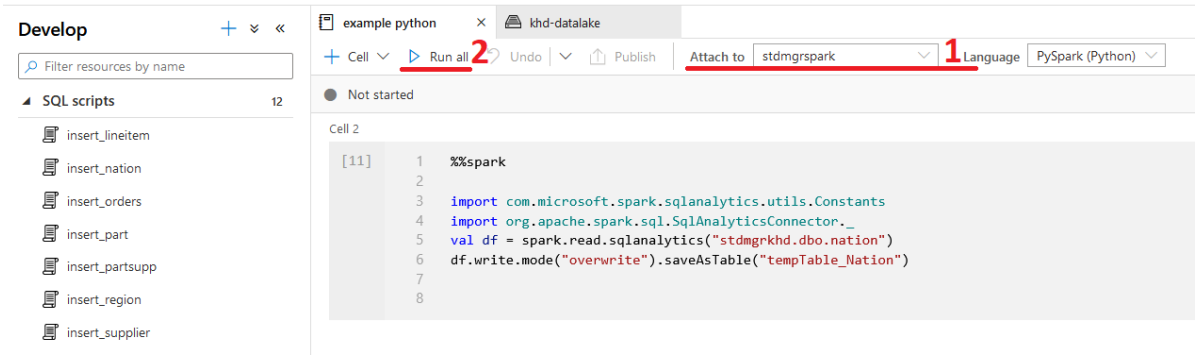
```
nations_df = spark.sql("select␣*␣from␣tempTable_Nation")

result_df = region_df.join(nations_df,
    region_df.R_REGIONKEY == nations_df.N_REGIONKEY) \
    .select(nations_df.N_NAME, region_df.R_NAME) \
    .withColumnRenamed("N_NAME", "nation_name") \
    .withColumnRenamed("R_NAME", "region_name") \
    .orderBy("region_name")

display(result_df)

result_df.write.parquet(
```

```
    "DesiredPathToParquetOnDataLake.parquet",
    partitionBy="region_name")
```

9. Tak napisany kod można uruchomić w Azure Synapse Analytics Workbench w utworzonym wcześniej notebooku. Należy wybrać stworzony Apache Spark pool (1) i kliknąć "Run all" (2) w celu uruchomienia każdej komórki zaczynając od góry.

   Przygotowany notebook z omówionym kodem można znaleźć na githubie:

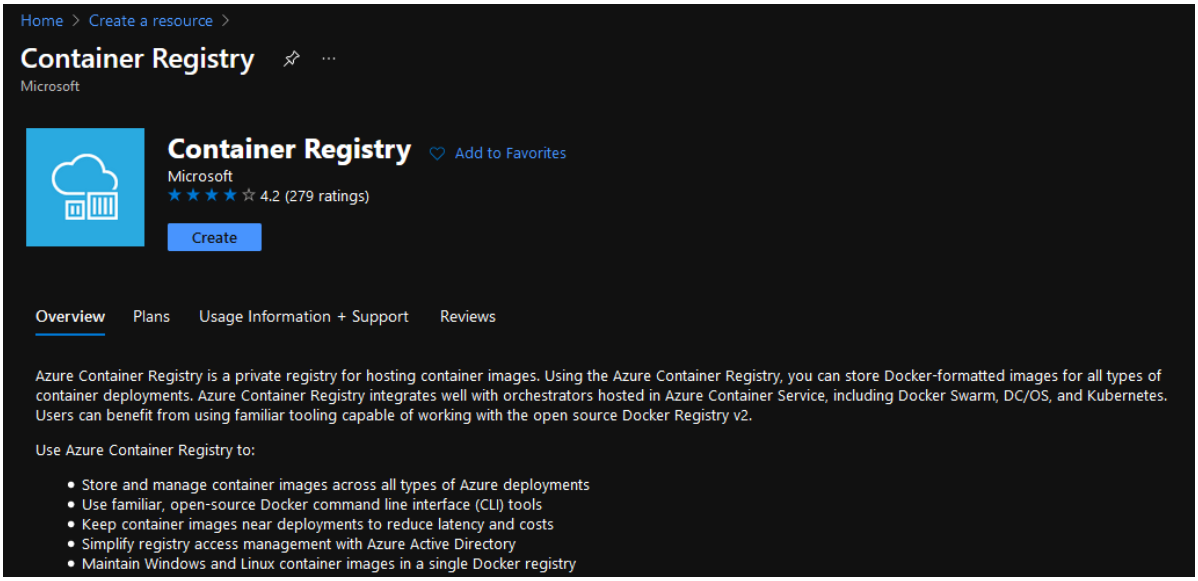
   `https://github.com/Put-EngineeringThesis-soccer-prediction/SantanderAzure/tree/master/Spark`



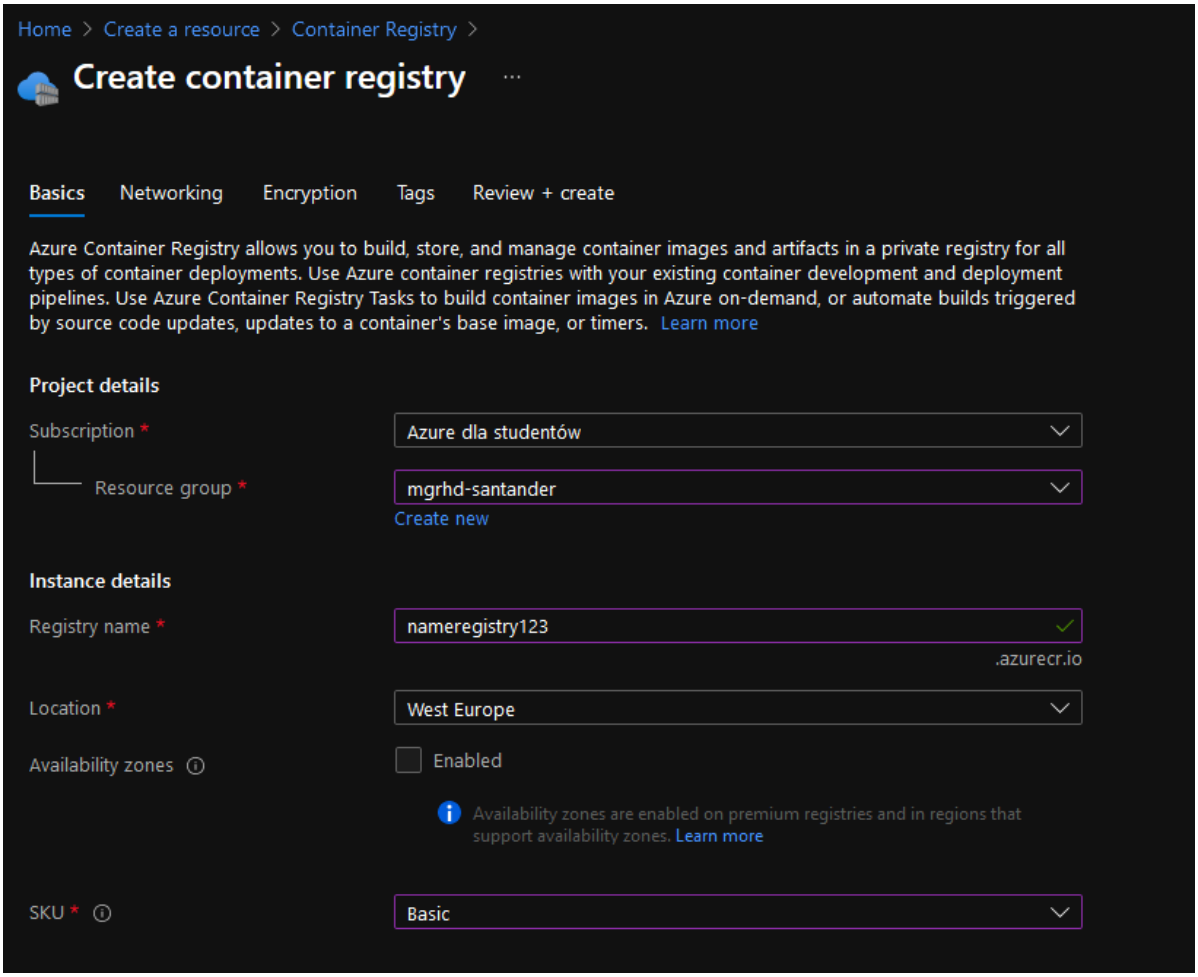
8 Wgrywanie obrazów dockera na Azure

Platforma Azure udostępnia narzędzia pozwalające na przechowywanie obrazów docker'a oraz ich uruchamianie w ramach instancji.

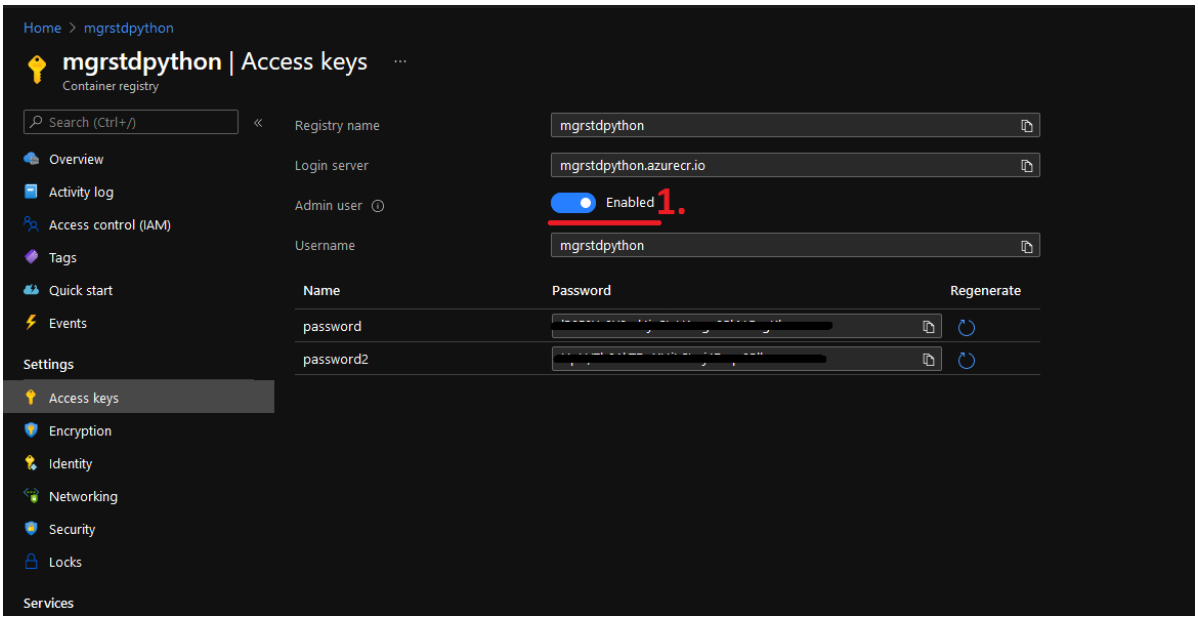
1. W celu utworzenia repozytorium dockera wyszukujemy na portalu „Container Registry" i klikamy „Create"



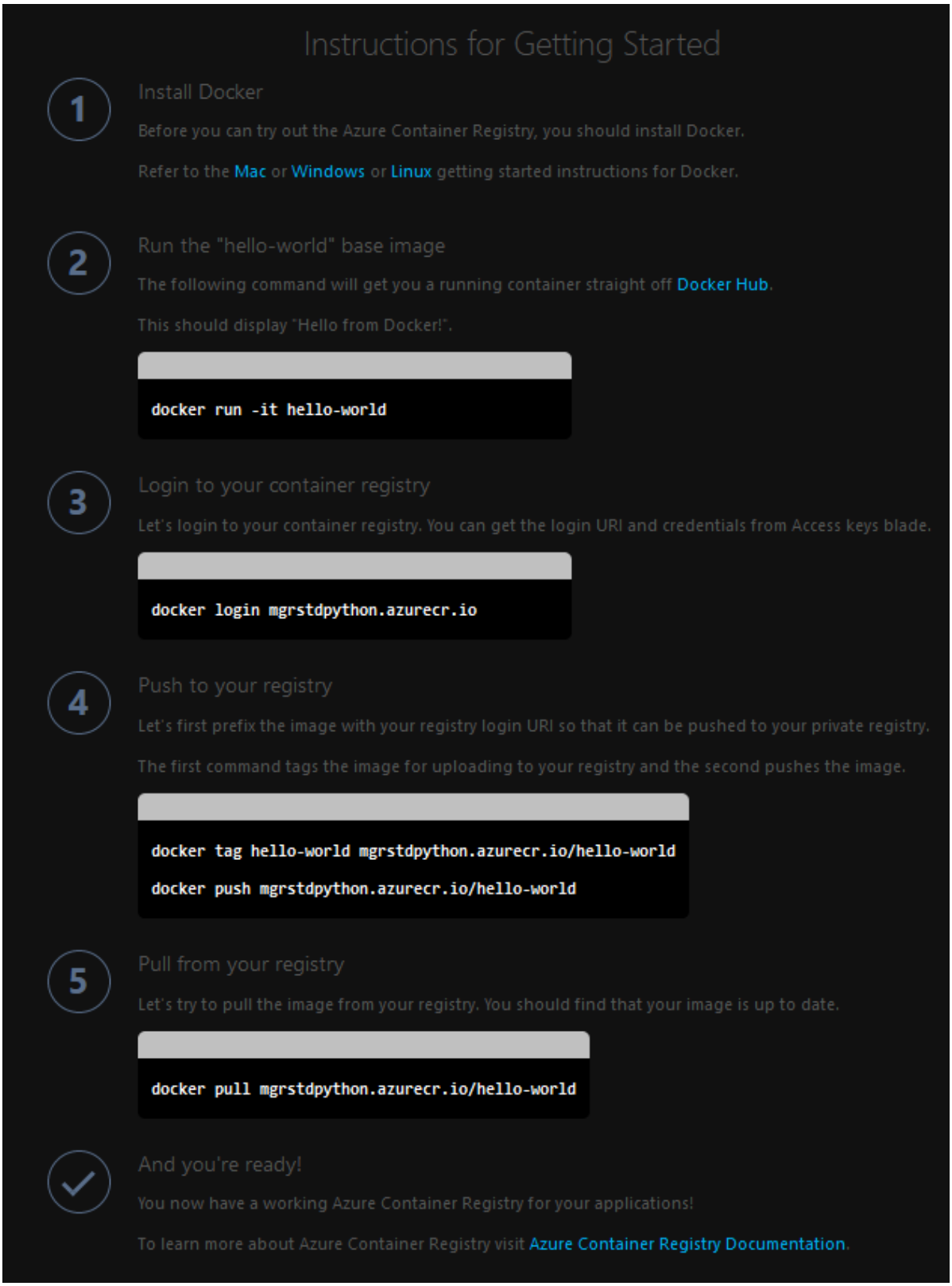
2. Wybieramy subskrypcje oraz grupe zasobów (może być utworzona nowa). Następnie nazywamy nasze repozytorium oraz możemy wybrać SKU (jakość repozytorium).



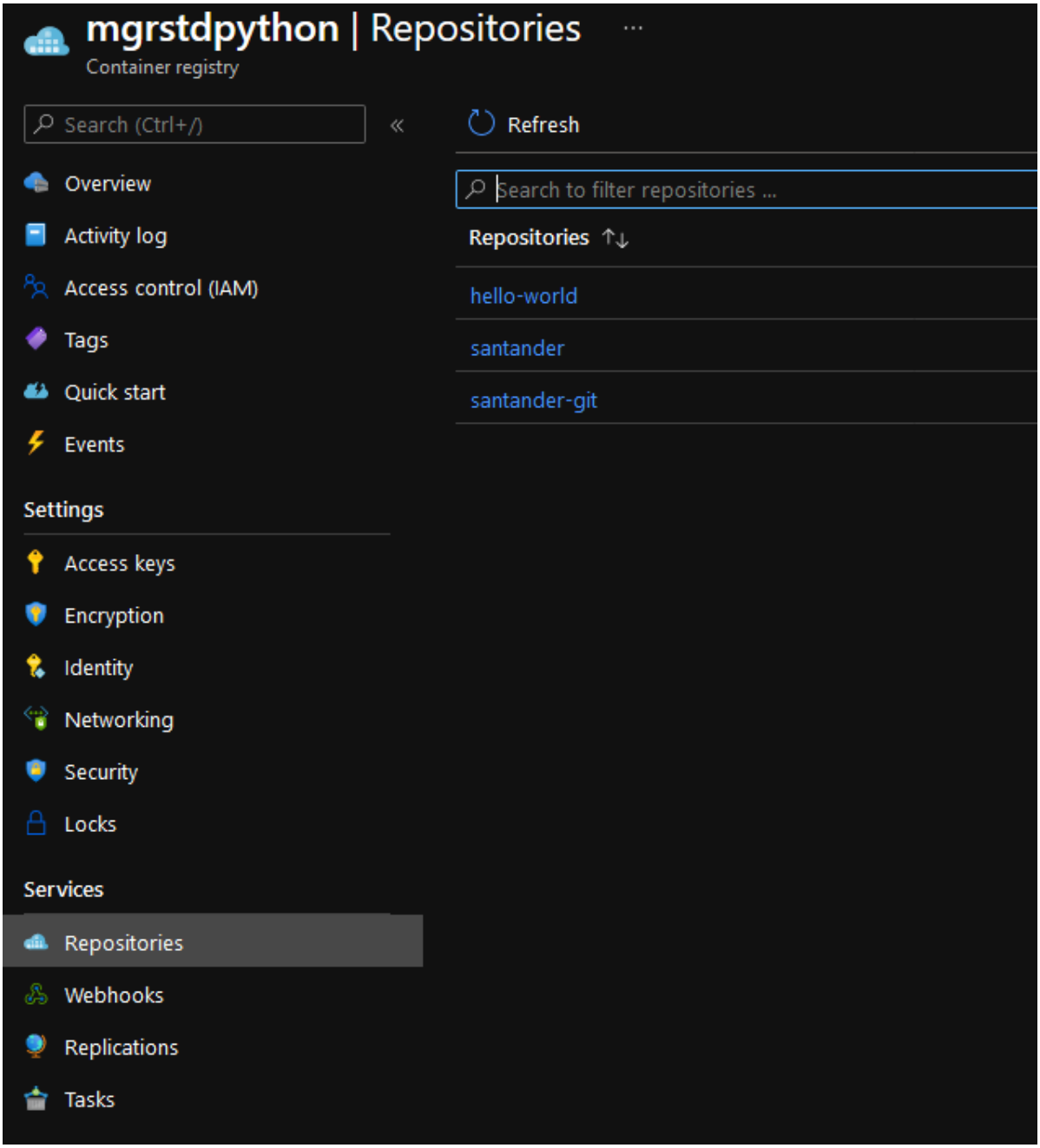
3. Przechodzimy do utworzonego przez nas repozytorium na Azure. Wybieramy w menu po lewej stronie „Access Keys” i ustawiamy „Admin user” na „Enabled” w celu umożliwienia logowania się do repozytorium. Zachowujemy jedno z hasłem oraz nazwę użytkownika na przyszłość (zawsze można tu wrócić i je skopiować).



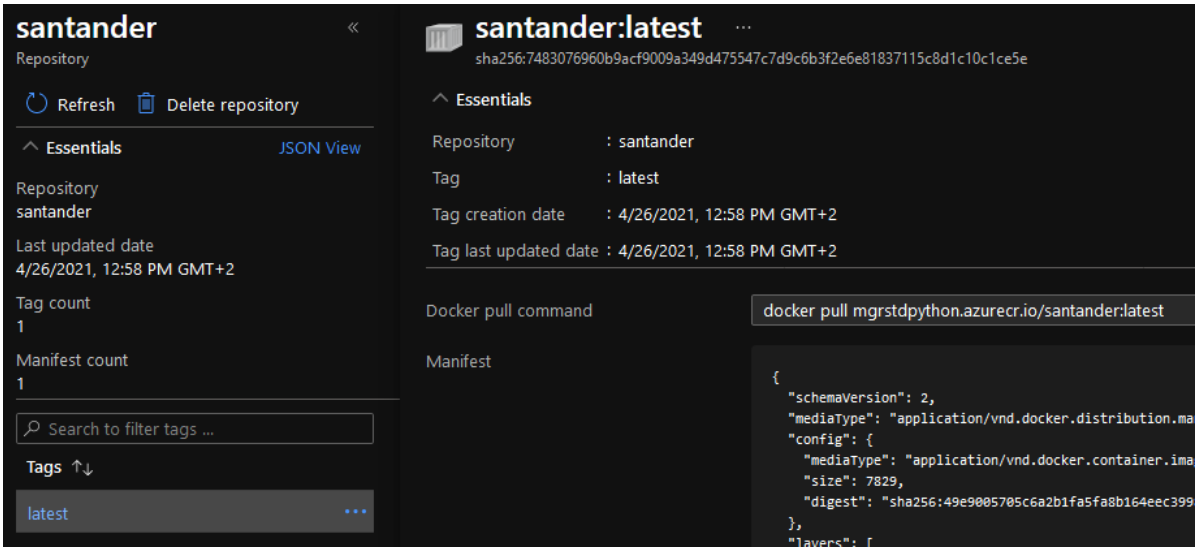
4. W menu po lewej wybieramy opcje „Quick start”. Możemy znaleźć tutaj sposób na dodanie gotowego obrazu dockerowego do repozytorium.



5. Po wgraniu obrazu możemy go obejrzeć w zakładce „Repositories”.



6. Klikając na tak dodany obraz możemy znaleźć jego tagi i sposób na ich pobranie.



9 Przykładowy obraz dockerowy

Aby utworzyć obraz dockerowy, który można umieścić w Azurowym repozytorium, należy lokalnie stworzyć plik Dockerfile, który stanowi szablon dla tworzonego obrazu, a następnie na jego postawie zbudować go przy pomocy komendy *docker build*. Na poniższym listingu przedstawiony został przykładowy plik tego typu, wykorzystywany przy realizacji tego zadania.

```
FROM python:3.7

COPY requirements.txt .
COPY my.azureauth .

RUN apt-get update && \
    apt-get upgrade -y

# Install git
RUN apt-get install -y git

# Needed for pyodbc
RUN apt-get install -y unixodbc-dev

# Install libraries from requirements file
RUN pip install -r requirements.txt

# Install libraries for interacting with Azure
RUN pip install xlsxwriter azure-storage-file-datalake azure-mgmt azure-common

# Install ODBC 17
RUN apt-get install sudo
RUN curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -
RUN curl https://packages.microsoft.com/config/debian/10/prod.list >
    /etc/apt/sources.list.d/mssql-release.list
RUN sudo apt-get update
RUN sudo ACCEPT_EULA=Y apt-get install -y msodbcsql17
RUN sudo ACCEPT_EULA=Y apt-get install -y mssql-tools
RUN echo 'export␣PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc

# Clone the repository with scripts
RUN git clone
    https://github.com/Put-EngineeringThesis-soccer-prediction/SantanderAzure
```

Na początku wybieramy obraz macierzysty, który będzie podstawą naszego obrazu. W naszym przypadku naszym wzorcem będzie obraz z zainstalowanym Pythonem w wersji 3.7. Jest to system bazujący na Debianie w wersji 10. Następnie kopiowane są pliki z lokalnego systemu plików, które znajdą się w utworzonym obrazie - pliki *requirement.txt* oraz *my.azureauth*. Są to odpowiednio pliki zawierające zależności do Pythona oraz plik zawierający informacje potrzebne do uwierzytelnienia w Azure.

W następnej kolejności instalowane są potrzebne komponenty - m.in. *git*, *ODBC 17* oraz biblioteki z pliku *requirements.txt* i te potrzebne do interakcji z różnymi komponentami chmurowymi.

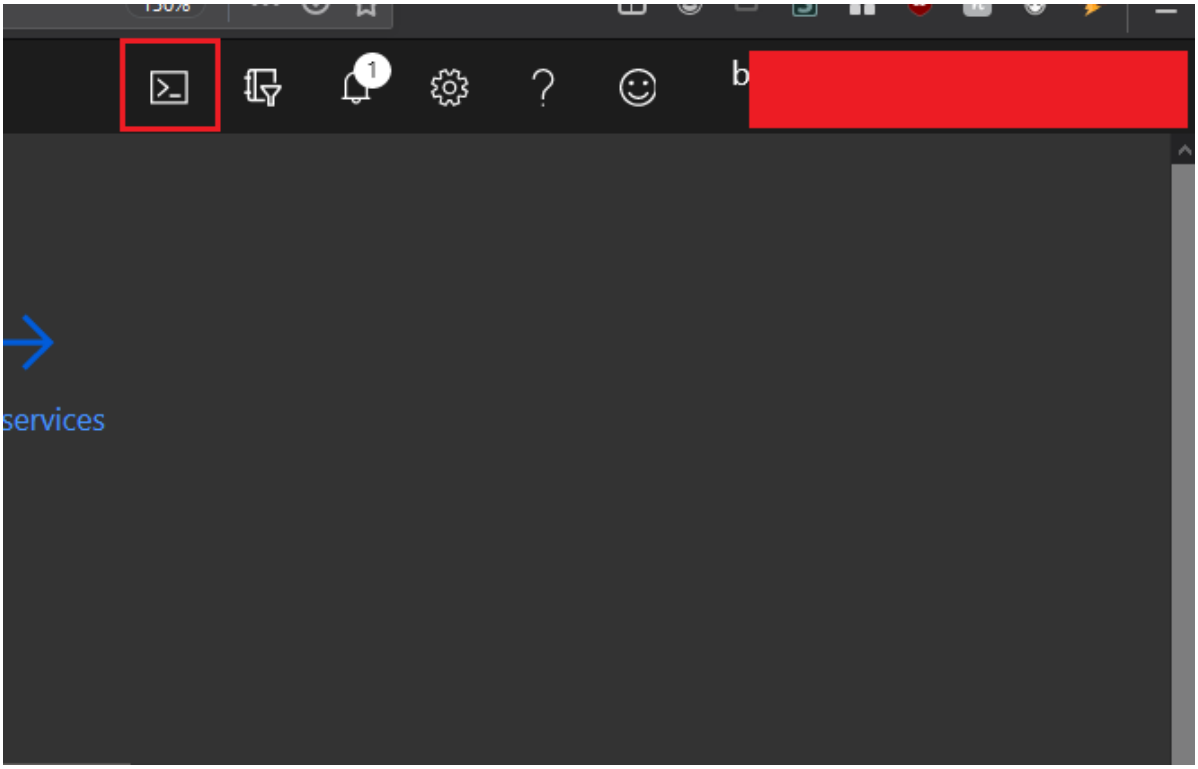
W ostatnim kroku pobierane jest repozytorium ze skryptami, które będzie wykorzystane w przypadku automatyzacji. Odpowiednie narzędzie będzie uruchamiało kontener dla tego obrazu i uruchamiało odpowiednie skrypty zgodnie z harmonogramem i zadanymi parametrami.

Zbudowany w ten sposób obraz można umieścić w repozytorium w sposób, który został omówiony w poprzednim podrozdziale przy pomocy komend *docker tag* oraz *docker push*.

9.1 Generowanie pliku my.azureauth

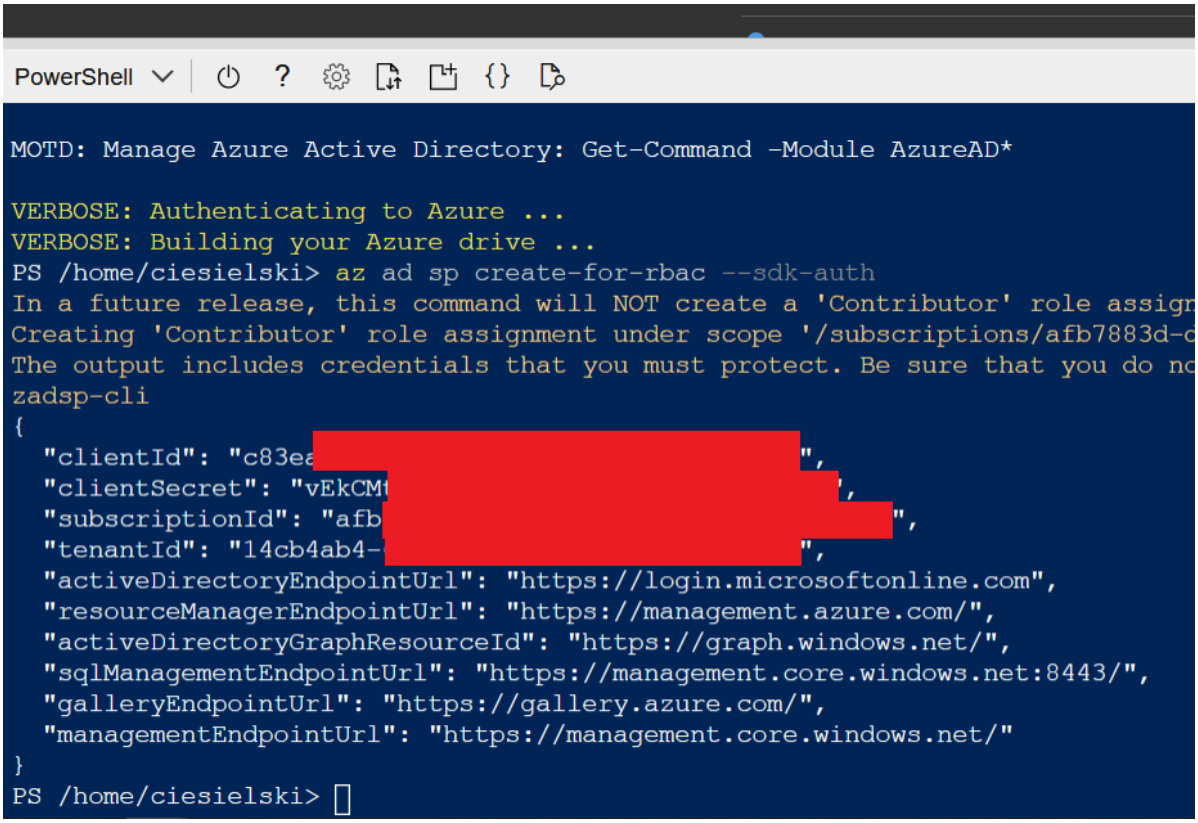
Przy budowie obrazu dockera został użyty plik my.azureauth. Jest to plik, który umożliwia skryptowi napisanemu w Pythonie autoryzację w Azure i manipulowanie zasobami. W naszym przypadku jest to potrzebne aby skrypt, który zarządza przetwarzaniem mógł usunąć instancje kontenera po zakończonej pracy. Aby go wygenerować należy:

1. Przejść do portalu Azure i kliknąć zaznaczoną ikonę. Uruchomi to Azure Cli, który umożliwia wykonanie instrukcji w powershellu. Podczas pierwszego uruchomienia może poprosić o utworzenie zasobów dla tego narzędzia. Wystarczy się zgodzić na wszystko.



2. Wpisać instrukcje *az ad sp create-for-rbac –sdk-auth*

   Wygeneruje ona plik json zawierający dane autoryzujące. Wygenerowany json należy przekopiować do pliku, w naszym przypadku jest to plik my.azureauth. Ten plik należy wgrać na obraz dockera, aby był dostępny dla skryptu zarządzającego przetwarzaniem.

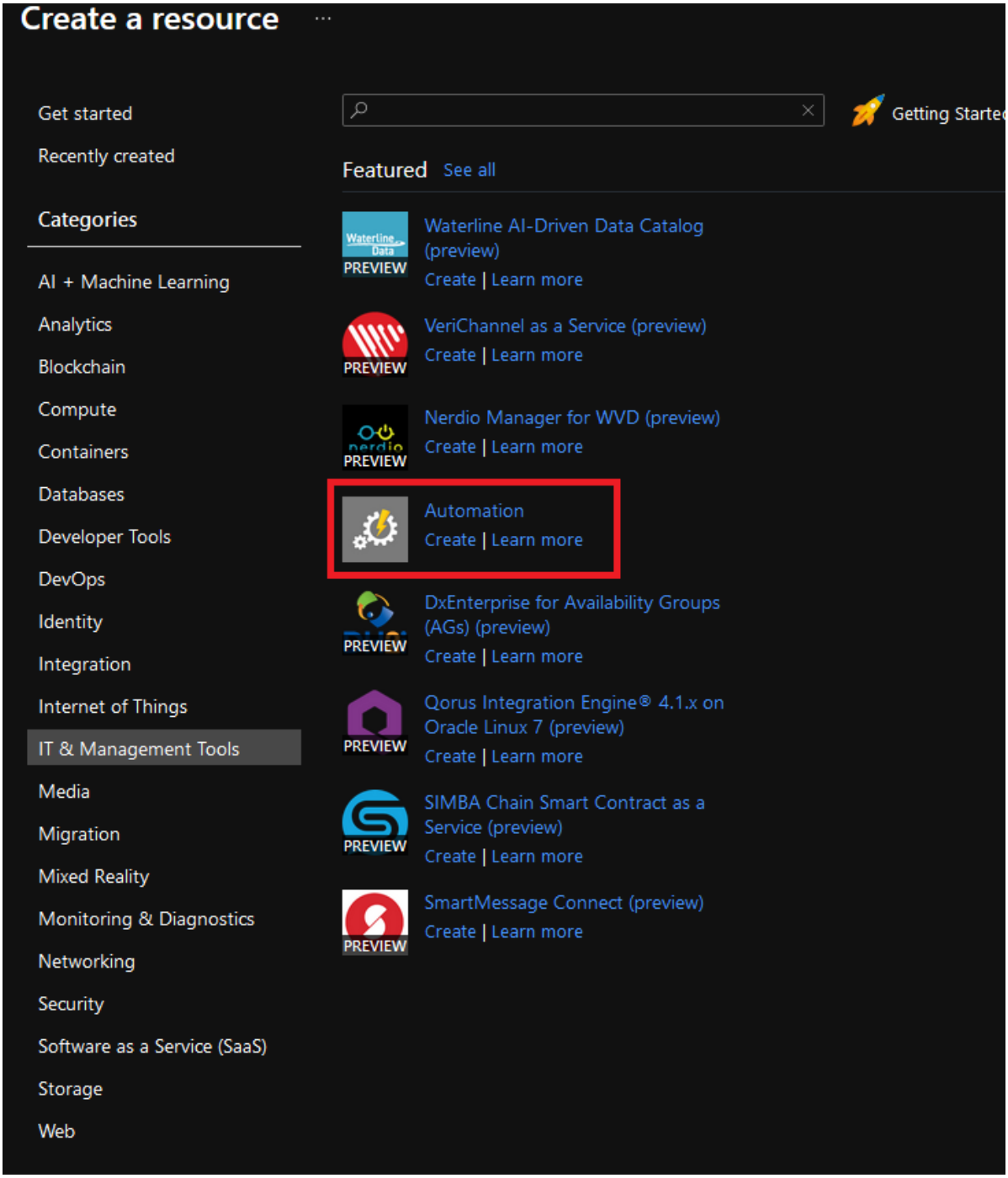


10 Automatyzowanie obrazów dockera

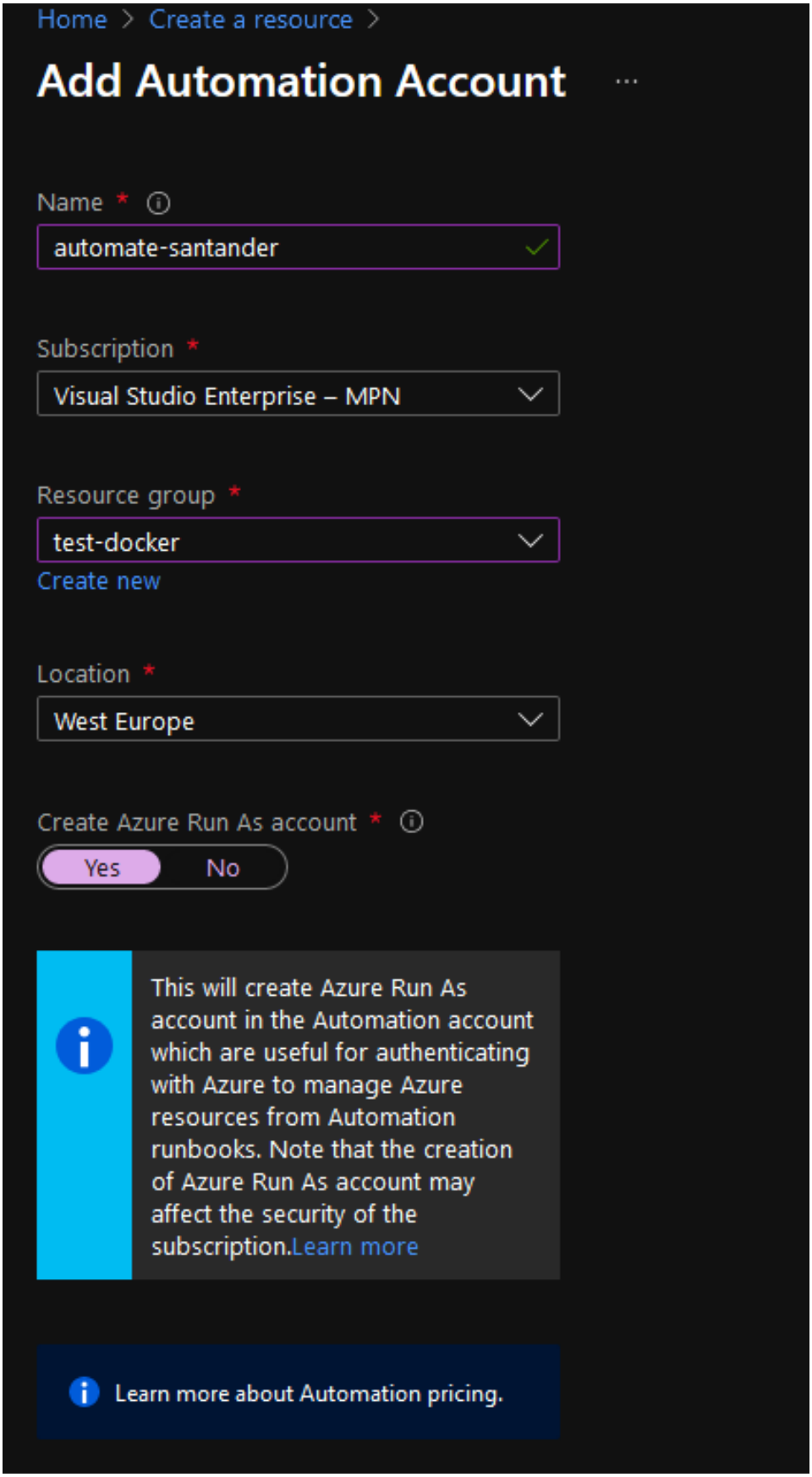
Jednym z natywnych narzędzi Azure jest tzw. „Automation Account", który umożliwia tworzenie runbook'ów, które można sparametryzować i uruchamiać w nich kod powershell lub python. Taki runbook może mieć ustawiany harmonogram z parametrami.

W kodzie powershell'owym jest możliwe utworzenie instancji docker'a na podstawie obrazu wgranego na wcześniej utworzone repozytorium obrazów. Wykorzystując te wszystkie funkcje możemy zautomatyzować uruchomienie się kodów pythonowych w dockerze.

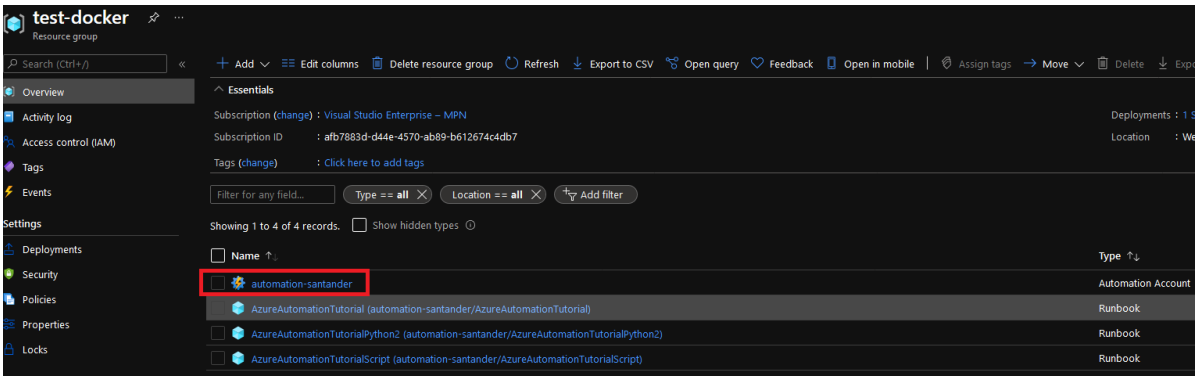
1. W celu utworzenia „Automation account" dodajemy nowy zasób i w zakładce „IT & Management Tools" wybieramy „Automation".



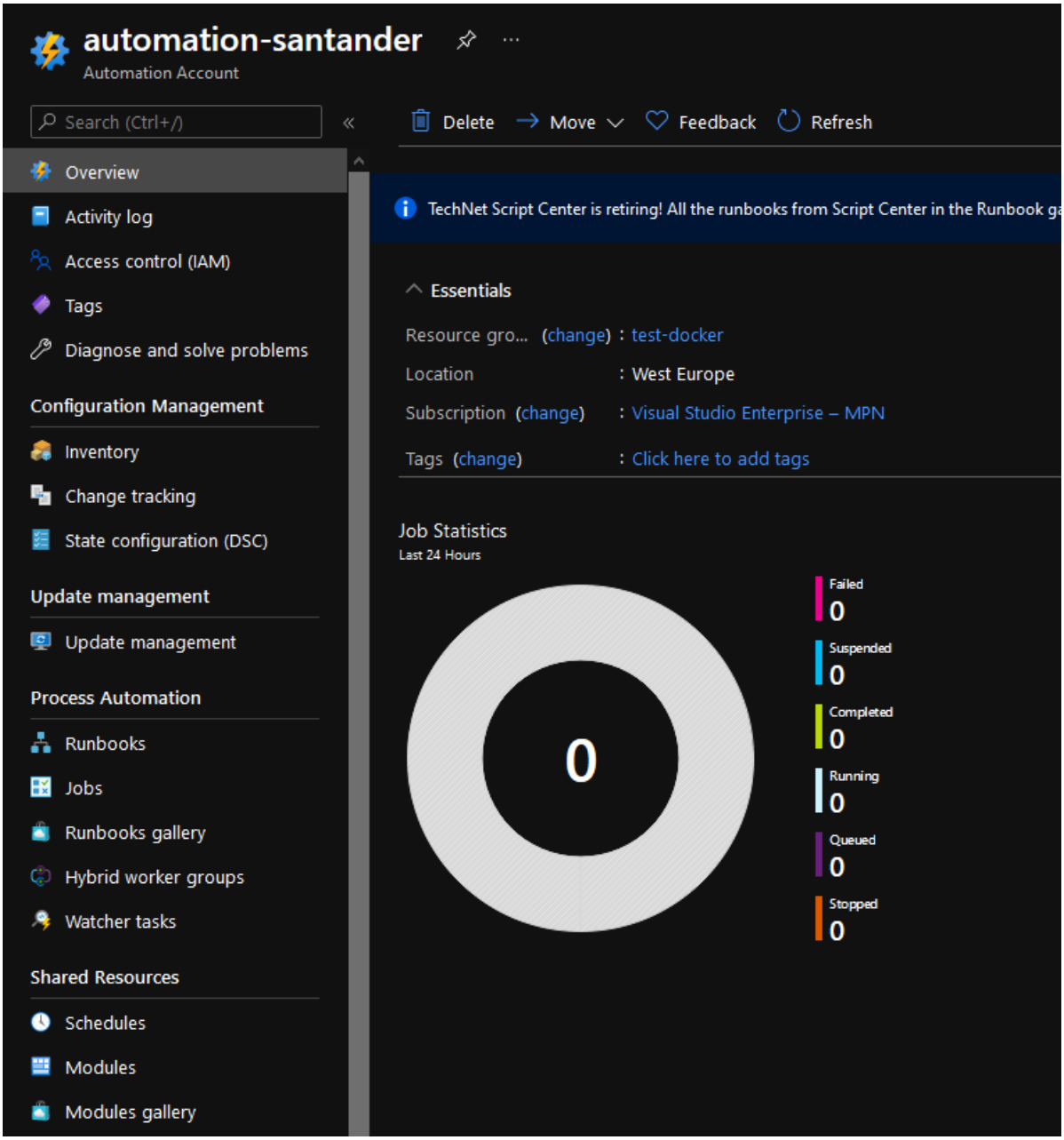
2. Tworzymy zasób wybierając nazwę, subskrypcje, grupę zasobów oraz lokalizacje.



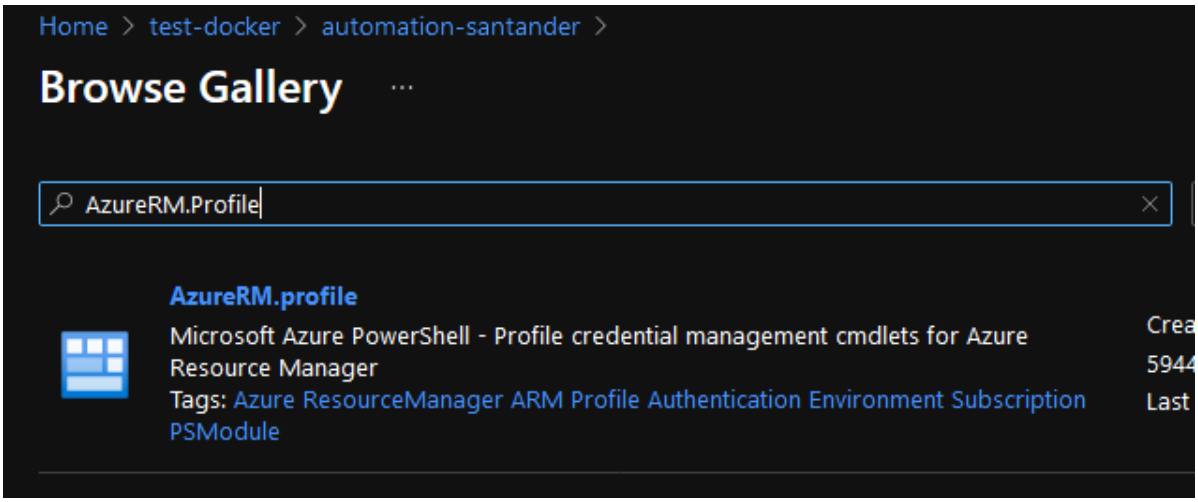
3. Wybieramy utworzony przez nas zasób. Dodatkowo zostaną utworzone przykładowe runbooki, można je usunąć albo zostawić i zignorować.



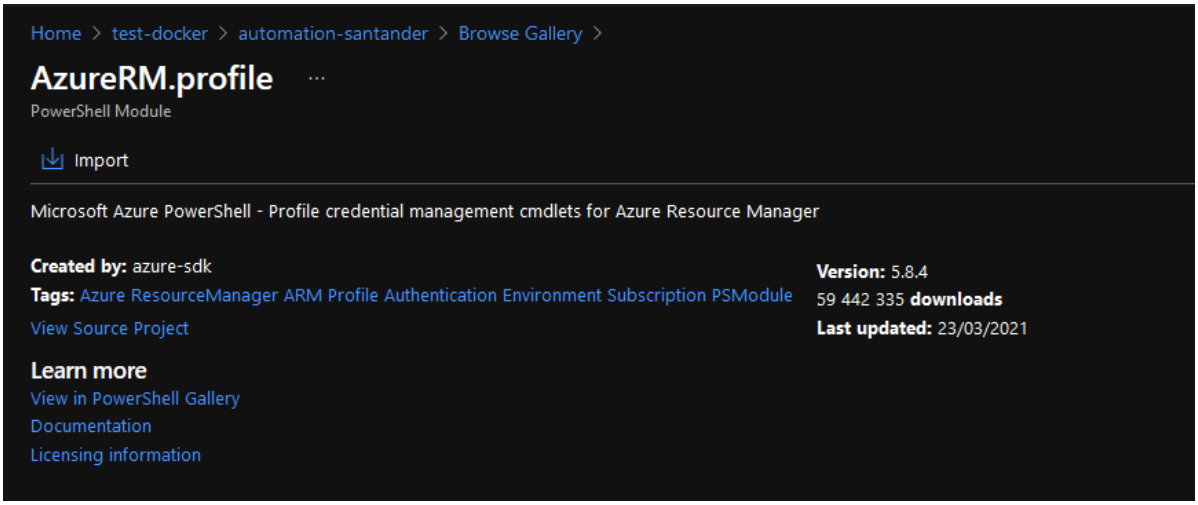
4. W menu po lewej stronie wybieramy „Modules gallery” w celu zainstalowania wymaganych modułów do manipulacji kontenerami na Azure.



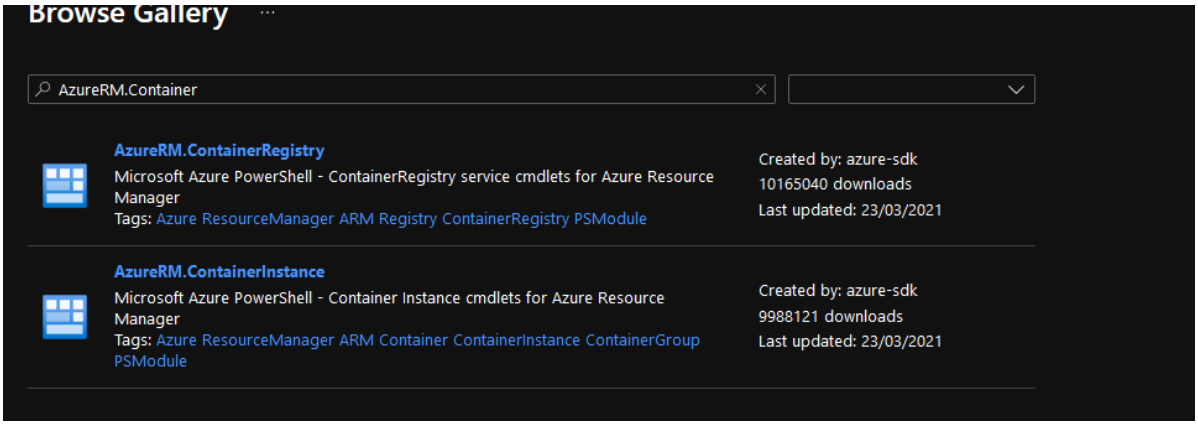
5. Pierwszym modułem, który musimy dograć jest „AzureRm.Profile”, wyszukujemy go i wybieramy z listy.



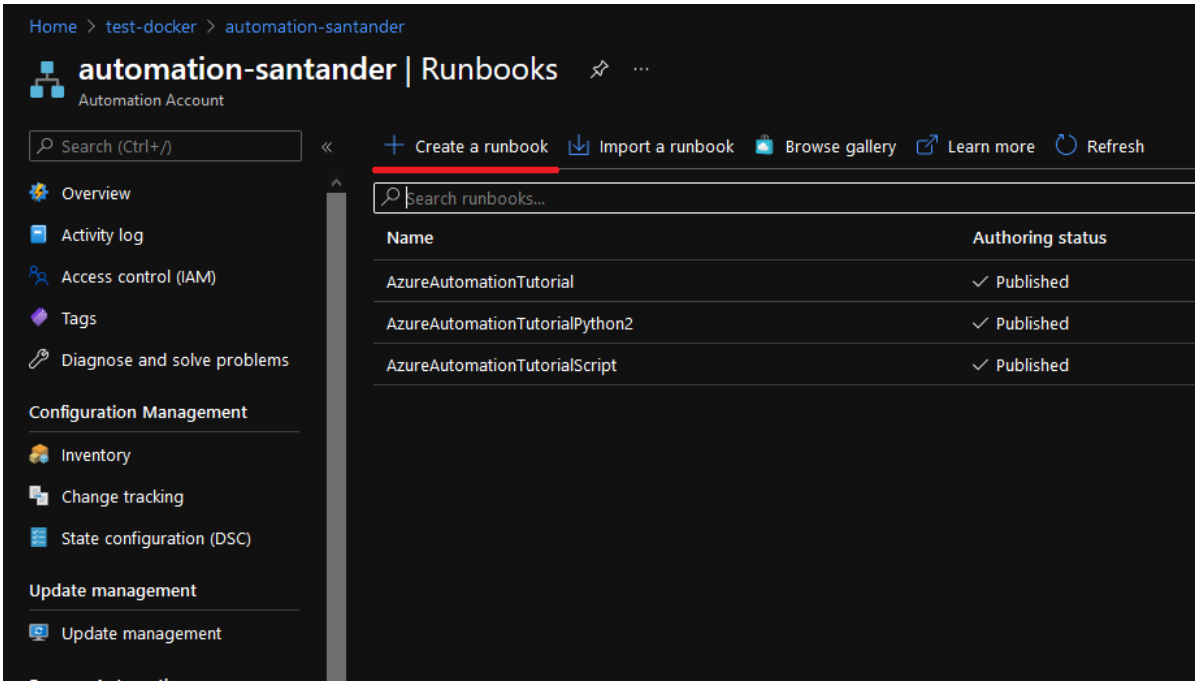
6. Klikamy guzik „Import” i importujemy moduł. Może to zająć kilka minut i dopóki się nie skończy nic nowego nie zainstalujemy.



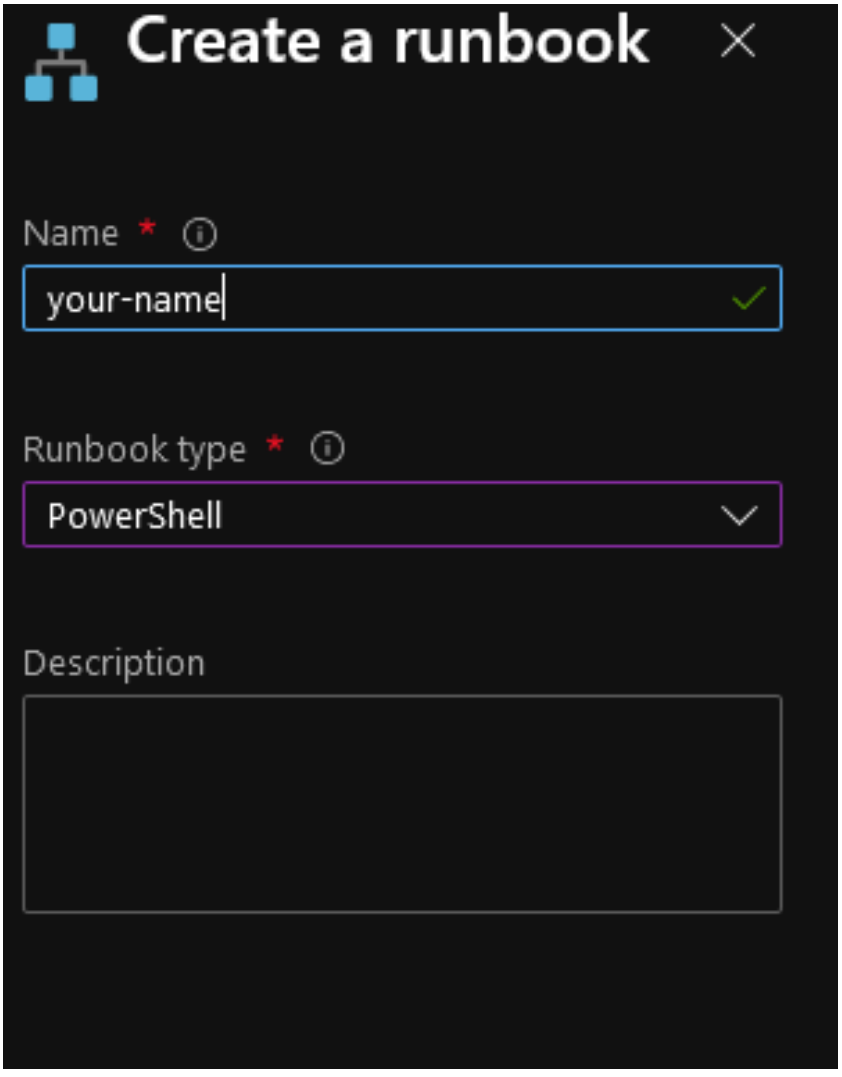
7. Po zainstalowaniu modułu „AzureRm.Profile” wyszukujemy w galerii następny moduł „AzureRM.Container” i postępujemy tak samo jak z poprzednim modułem.



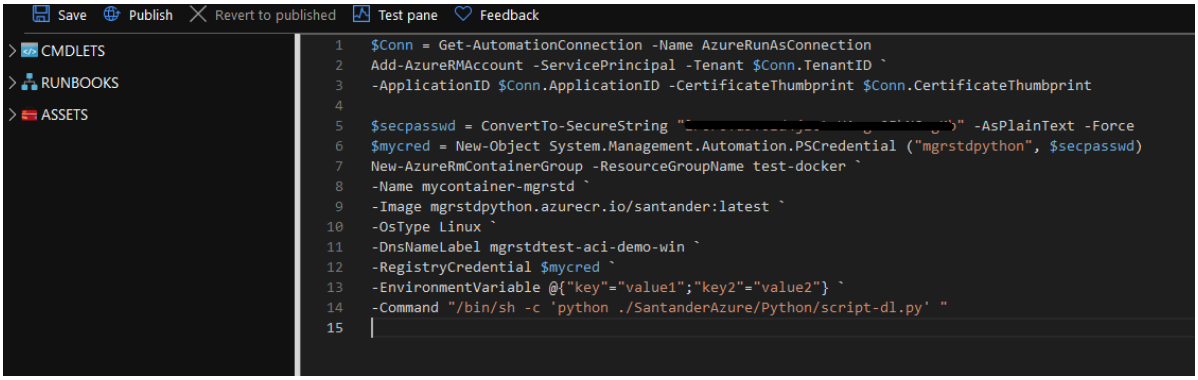
8. Kiedy wszystko jest gotowe przechodzimy do zakładki „Runbooks” i klikamy „Create a runbook”.



9. Nadajemy mu nazwę i wybieramy „Runbook type” jako Powershell.



10. W tak utworzonym runbook'u możemy wpisać kod tworzący naszą instancje obrazu kontenera.



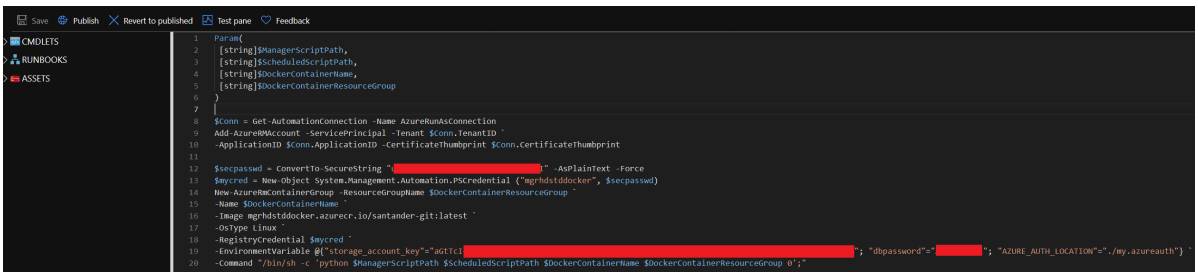
Linie 1-3 służą do autoryzacji naszego runbook'a w Azure, bez tego nie możemy zarządzać zasobami. Zmienna $secpasswd jest hasłem do naszego konta na repozytorium dockerowym, następnie tworzymy zmienną $mycred w której mamy parę 'login hasło'.

Zaczynając w linijce 7 - komenda 'New-AzureRmContainerGroup' służy do tworzenia instancji naszego obrazu dockerowego. Link do dokumentacji: `https://docs.microsoft.com/en-us/powershell/module/azurerm.containerinstance/new-azurermcontainergroup?view=azurermps-6.13.0`. W skrócie wybieramy nazwę grupy zasobów, gdzie będzie utworzona instancja, nazwę samej instancji a następnie wskazujemy, jaki obraz chcemy utworzyć. Musimy określić również typ systemu w obrazie, nazwa Dns nie jest wymagana. „RegistryCredential" wykorzystuje nasze dane do zalogowania się w repozytorium. Następnie możemy dodać zmienne środowiskowe oraz jaka komenda bashowa ma zostać wykonana po utworzeniu tego kontenera (np. uruchomienie odpowiednie skryptu).

11 Uruchamianie skryptów Pythonowych z harmonogramem

Po utworzeniu podstawy do automatyzacji można utworzyć docelowy runbook z parametrami i ustawić harmonogram dla konkretnych skryptów. Zaczniemy od edycji skryptu powershell'a tak, aby przyjmował parametry, po czym ustawimy harmonogram dla przygotowanego skryptu, który będzie przyjmował te parametry.

1. Przejdź do przygotowanego runbook'a i kliknij *Edit*. Poniżej kod, który będzie używany przy harmonogramowanych uruchomieniach.

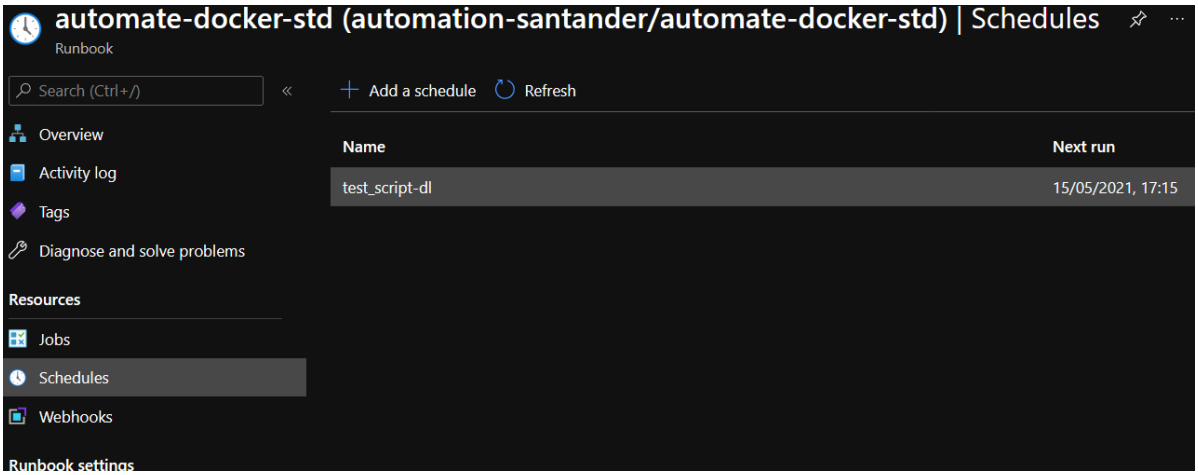


   Na samej górze kodu pojawiły się parametry, określamy ich typ i nazwę. Można jeszcze jeszcze dodać takie dekoratory jak *[Parameter (Mandatory= $true/$false)]*, domyślnie parametry są opcjonalne.

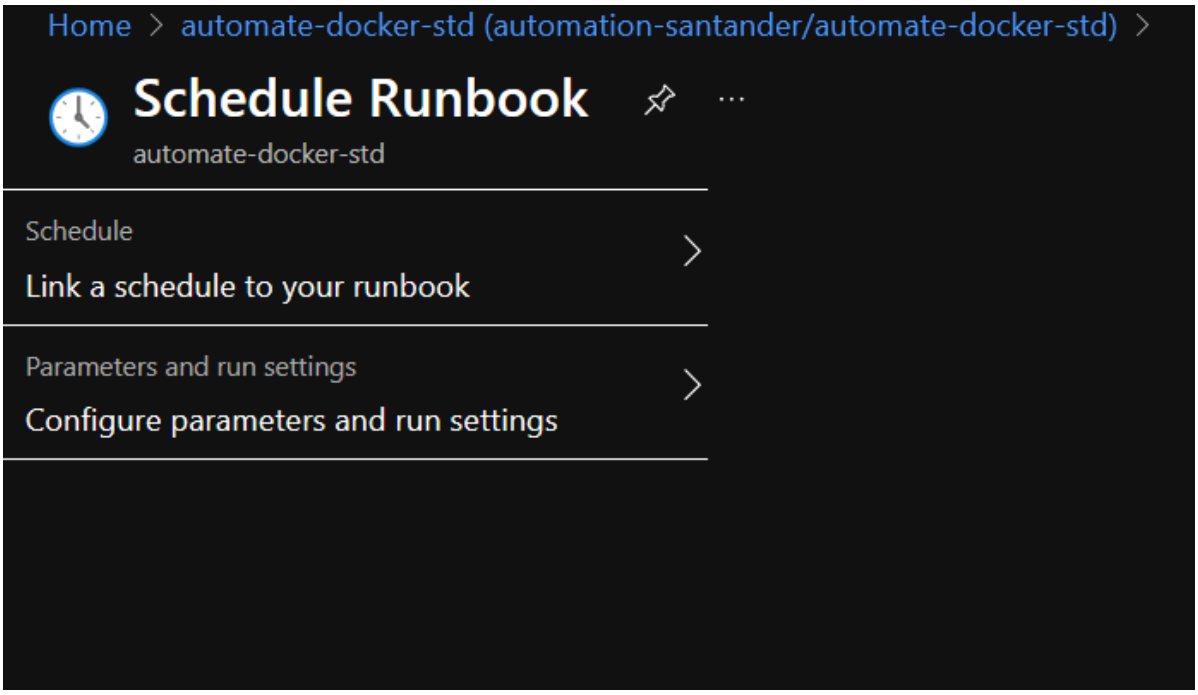
   W linijce 19 określamy zmienne środowiskowe dla parametrów, które powinny być chronione. Klucz dostępu do data lake'u, hasło do bazy danych oraz lokacja wcześniej przygotowanego pliku my.azureauth. Te zmienne w przypadku kodu przygotowanego w ramach projektu są wymagane dla poprawnego działania skryptów.

   Na końcu określamy komendę wykonaną na początku działania obrazu. Uruchamia ona skrypt przykazany w parametrze ManagerScriptPath (czyli skrypt menadżera, który później usunie kontener), następnie ścieżka do skryptu docelowego, który chcemy wykonać. Dodatkowo 2 parametry, nazwa instancji kontenera oraz grupa zasobów, na której się on znajduje. Tak przygotowany skrypt możemy zapisać i kliknąć *Publish*.

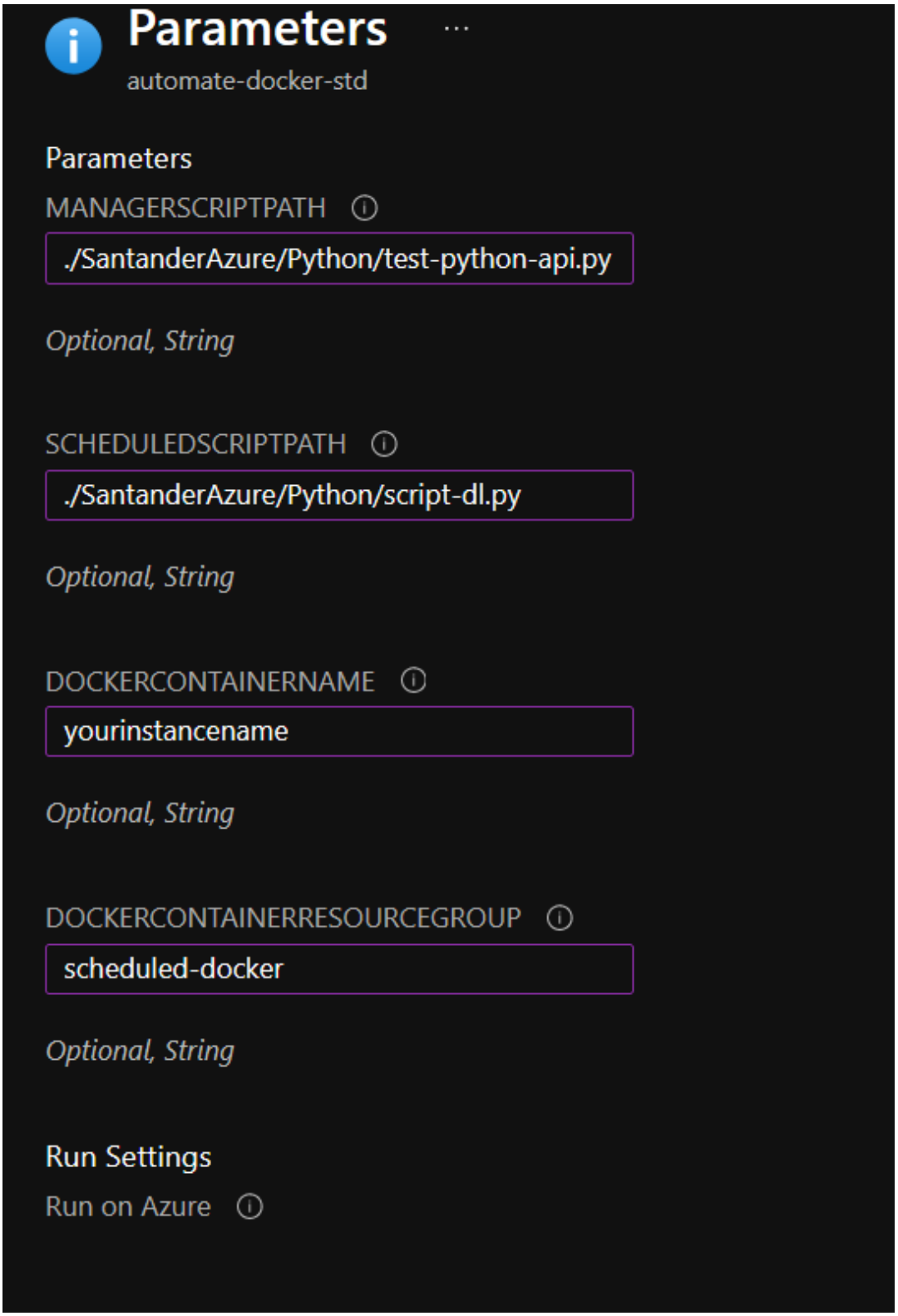
2. Po opublikowaniu skryptu wróćmy do zasobu runbook'a i z menu po lewej wybierzmy *Scheludes*, a następnie kliknij przycisk *Add a schedule* w celu dodania nowego harmonogramu.



3. W środku zobaczymy poniższe menu. Klikamy *Schedule* i w środku dodajemy nowy harmonogram lub wybieramy istniejący, jeżeli już jakiś istnieje o wymaganych parametrach.



4. Po wyborze harmonogramu wypełniamy parametry wybierając *Parameters*, z jakimi danych scheduler bedzie uruchamiać runbook. Poniżej widać przykładowe parametry dla skryptu script-dl.py. Uwaga, *Resource group* musi być istniejącym zasobem, utworzonym ręcznie.



W ten sposób dodaliśmy harmonogram, który powinien uruchomić podany skrypt o zadanej godzinie, po czym usunie kontener który go wykonywał.

12 Skrypt Menadżera

W poprzednim rozdziale użyty został skrypt, który zarządza uruchamianiem innych skryptów na instancjach i usuwaniem kontenera. Jest to bardzo prosty skrypt, którego treść widać poniżej.

```
import os
import sys
import subprocess
from azure.common.client_factory import get_client_from_auth_file
from azure.mgmt.resource import ResourceManagementClient
from azure.mgmt.containerinstance import ContainerInstanceManagementClient

# Retrieve the IDs and secret to use with ClientSecretCredential
arguments = sys.argv[1:]
script_to_run = arguments[0]
container_name = arguments[1]
resource_group_name = arguments[2]

subprocess.run(["python", str(script_to_run)])

auth_file_path = os.getenv('AZURE_AUTH_LOCATION', None)
print(auth_file_path)
if auth_file_path is not None:
    print("Authenticating␣with␣Azure␣using␣credentials␣in␣file␣at␣{0}"
          .format(auth_file_path))

    aci_client = get_client_from_auth_file(
        ContainerInstanceManagementClient)
    res_client = get_client_from_auth_file(ResourceManagementClient)
else:
    print("\nFailed␣to␣authenticate␣to␣Azure.␣Have␣you␣set␣the")

aci_client.container_groups.delete(resource_group_name, container_name)
```

Najpierw importujemy potrzebne biblioteki, po czym czytamy argumenty z wejścia. Uruchamiamy skrypt docelowy i po zakończeniu jego działania autoryzujemy się korzystając z wcześniej przygotowanego pliku o nazwie *my.azureauth*, do którego ścieżka jest zapisana pod zmienną środowiskową **AZURE_AUTH_LOCATION**. Jeżeli autoryzacja się powiodła, korzystając z biblioteki Microsoftu, usuwamy podany w parametrach kontener. Operacja ta jest błyskawiczna, po wykonaniu skryptu nie ma dostępu do wcześniej uruchomionego kontenera.

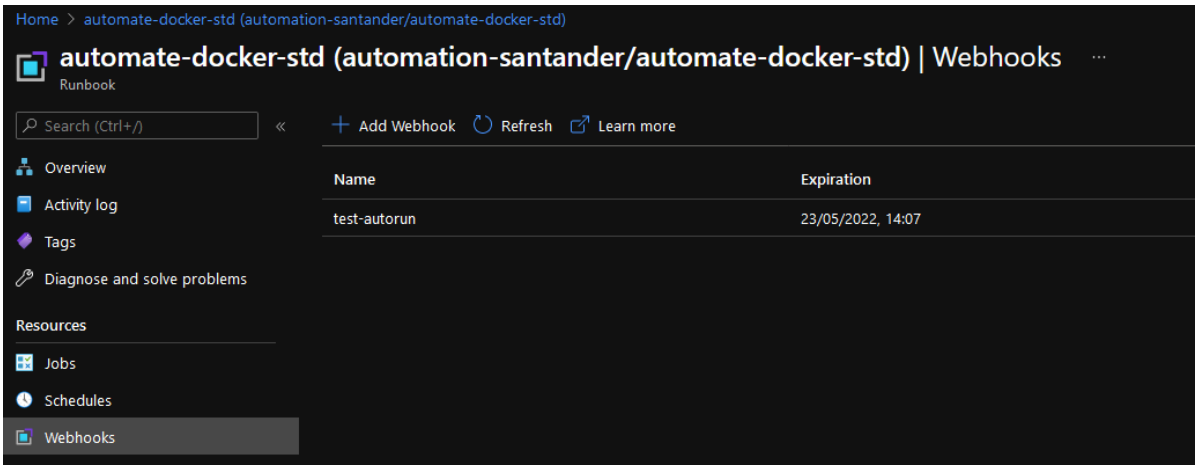
13 Uruchamianie skryptów Python w ETL

13.1 Generowanie Webhook dla runbook'a

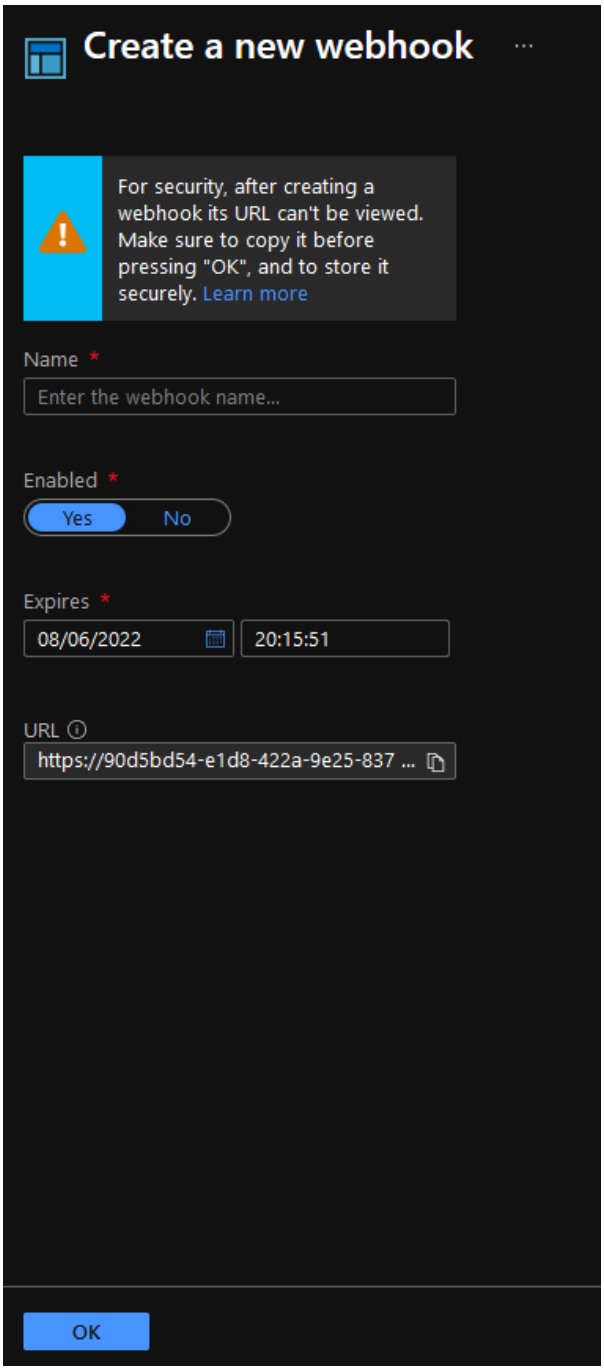
W celu uruchomienia runbook'a, który odpowiada za uruchomienie przetwarzania raportów spoza portalu Azure, możemy skorzystać z kilku metod. Szczegółowo są one opisane w dokumentacji Microsoftu `https://docs.microsoft.com/en-us/azure/automation/start-runbooks`.

Wykorzystana została metoda „Webhooks", w której generujemy link służący do uruchamiania przetwarzania.

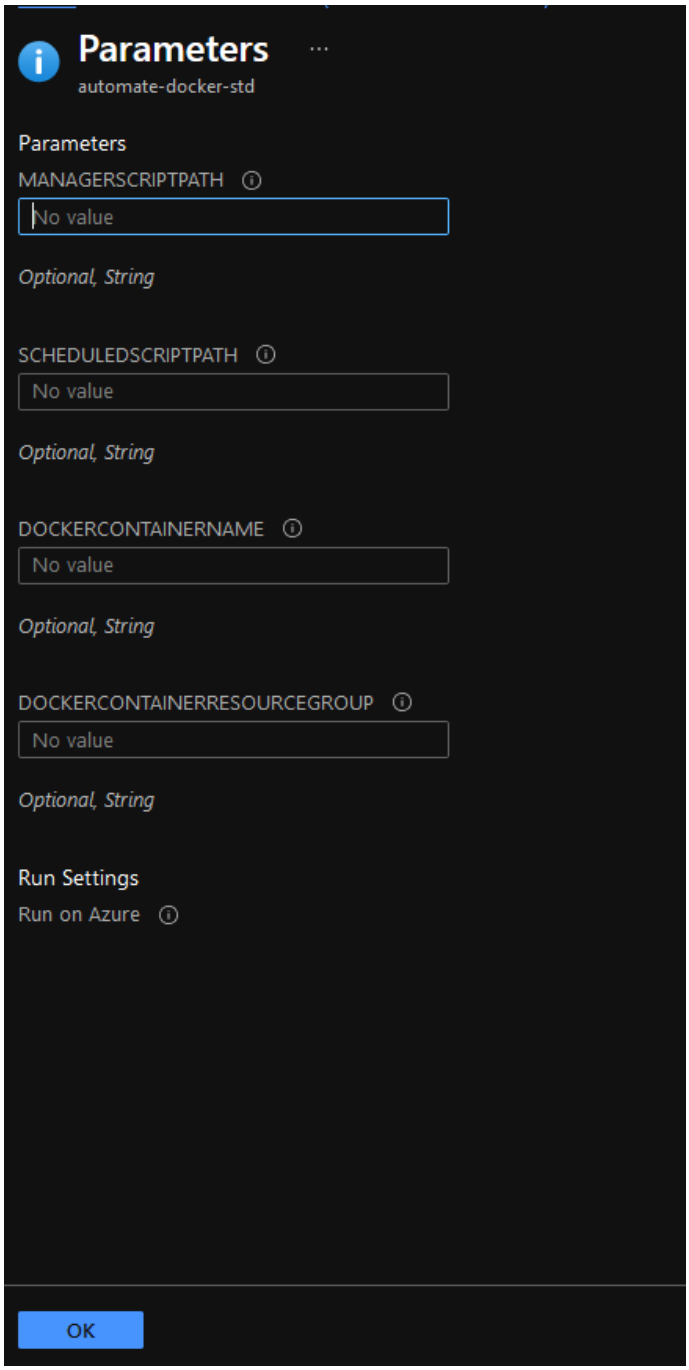
1. W przygotowanym wcześniej runbooku przechodzimy do zakładki „WebHooks" i w niej klikamy guzik „Add Webhook" w celu utworzenia sparametryzowanego adresu URL.



2. Następnie w pierwszej opcji wybieramy nazwę naszego webhook'a, określamy czy jest aktywny i wybieramy dzień w którym wygaśnie automatycznie. **Uwaga, trzeba zapisać URL w tym kroku. Zgodnie z tym, co jest napisane nie ma możliwości skopiowania go później.**



3. Ostatnim zadaniem podczas tworzenia webhook'a jest określenie parametrów, z jakim zostanie uruchomiony runbook. W przypadku naszych skryptów, trzeba określić ścieżki do skryptu menadżera, skryptu do uruchomienia, nazwę kontenera i grupę zasobów dla tego kontenera.

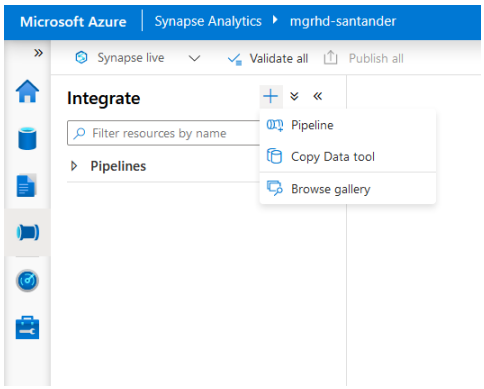


Z tak przygotowanym webhook'iem możemy przejść do następnego kroku, czyli zintegrowanie go do procesu ETL.

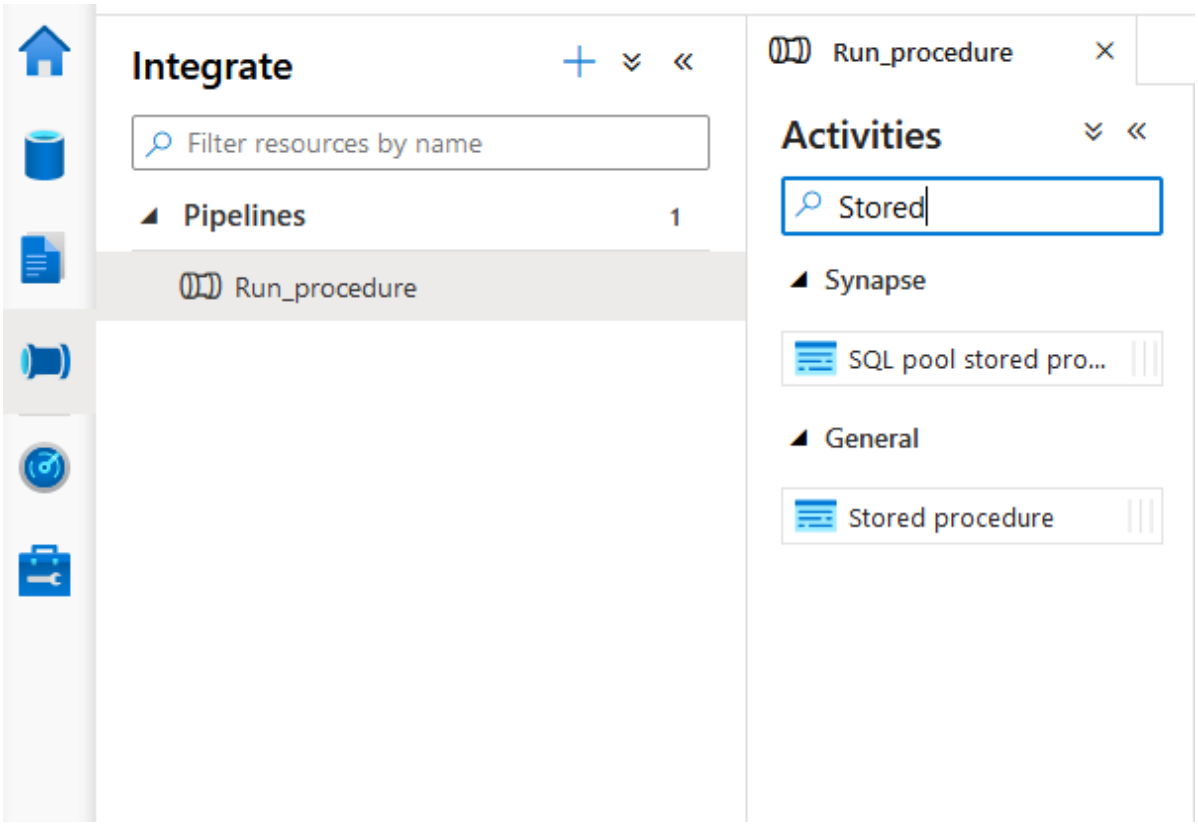
13.2 Przygotowanie prostego procesu ETL

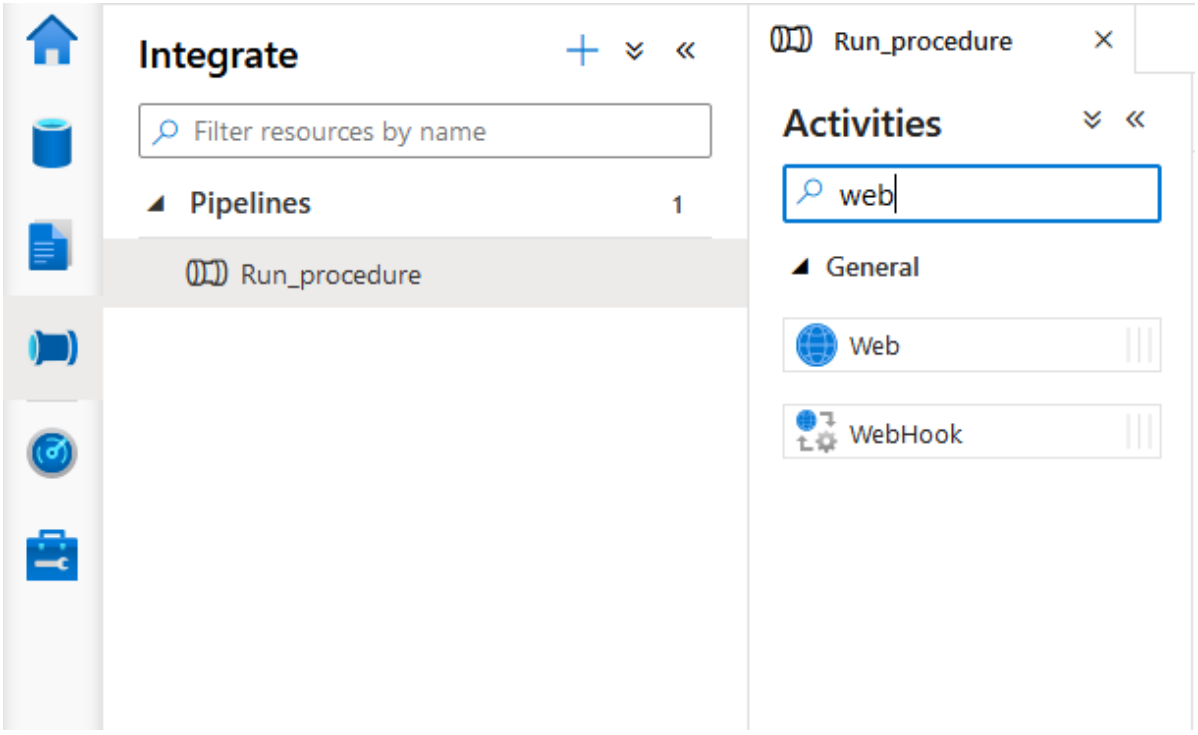
Proces ETL można przygotować bezpośrednio w portalu Synapse Analytics Workbench, więc dla prostoty został tam utworzony jasny pipeline. Należy jednak pamiętać, że webhook można wykorzystać w każdym systemie, który ma możliwość wykonywania żądań HTTP.

1. W utworzonym Synapse Analytics przechodzimy do zakładki Integrate i klikamy na plusik. Tam wybieramy „Pipeline” w celu utworzenia procesu ETL.

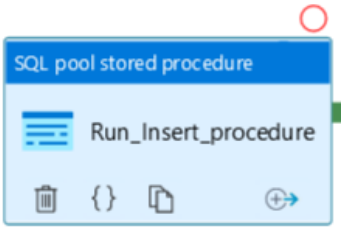


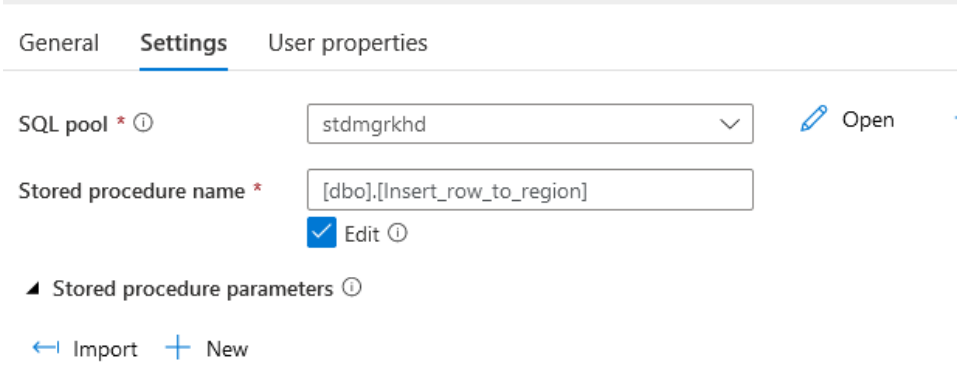
2. W wyszukiwarce „Activities” szukamy „SQL pool stored procedure” i dodajemy do pipeline’a. Tak samo szukamy „Web” i dodajemy do pipeline’a.



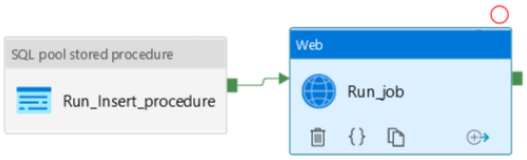


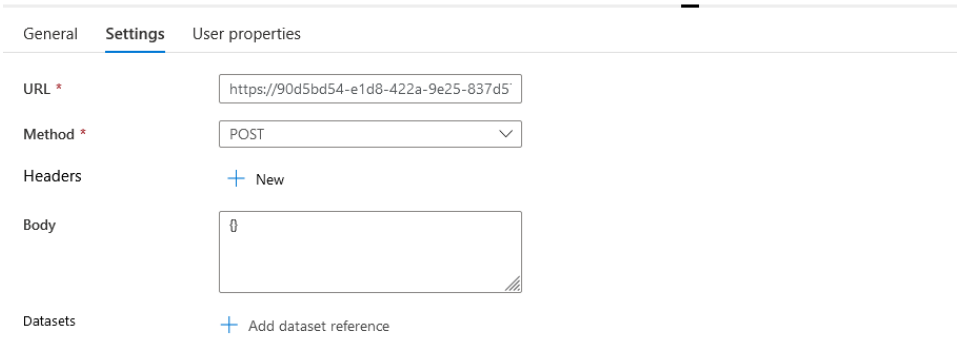
3. Wybierając wcześniej dodany SQL pool stored procedure w zakładce „Settings" określamy naszą składowaną procedurę w bazie danych, która ma symulować naszą hurtownie.





4. Następnie konfigurujemy aktywność „Web”, która będzie uruchomiona po składowanej procedurze. Również w zakładce „Settings” określamy URL (tutaj wklejamy nasz wcześniej przygotowany URL do webhook’a). Metode ustawiamy na POST i w ciele żądania dodajemy ’{}’.





W ten sposób można uruchomić przetwarzanie w bazie danych, po wykonaniu się procesów ETL.