COSC 416: Topics in Databases (DBaaS)

TOPIC 10: MICROSOFT AZURE API AND POSTGRESQL

SCHEDULE

1. *Azure API & SDK*
2. *Azure PostgreSQL*
3. *Today’s Lab & Next Topic Preview*

THE AZURE API & SDK

TO REFRESH YOUR MEMORY

- When we left before the break we were working with Microsoft Azure Database for MySQL, which is Azure's MySQL DBaaS offering
- We talked about how to set up an Azure Database for MySQL instance, and touched on using the Command Line Interface (CLI) to control it, just as we did for Amazon AWS

SO NOW WHAT?

- As we previously discussed, since we don’t have access to a shell or terminal, and for the sake of scalability, we need ways to control the creation and management of our DBaaS instances
- With Amazon AWS, we accomplished this using the AWS SDK to manage our RDS instances

THE AZURE REST API

- Across almost all of Azure’s web services, Microsoft offers a REST API that allows the user to directly control their Azure instances via HTTP API requests
- Like Amazon AWS, this is complicated somewhat by the need to be able to authenticate the API request – and like AWS, this is a rather complex issue

AZURE API AUTHENTICATION – ACTIVE DIRECTORY

- In order to authenticate your API requests, you first need to create an *Active Directory tenant (AD)* for your application
  - The tenants may be a web application with a single log in, or a public/native install using the current user’s log in credentials
- You must register for an AD before you try to use the API, as it’s required before you can perform authentication

AZURE API AUTHENTICATION - TOKENS

- Once your application has been registered, you can authenticate in a two-part process
- You first hit the /*authorize* API link, which gives you an authorization code
- You then hit the /*token* link, passing along your authorization code, in order to get an *access token*
- This access token is then used to sign subsequent requests made to the REST API service

IN COMPARISON TO AWS

- On the whole, directly working with Azure’s REST API is about as much of a pain as working directly with Amazon AWS’s API
- This is to be expected: public web APIs like what Azure and Amazon use require robust authentication to prevent illegitimate access → painful authentication process

BUT WAIT, WHAT ABOUT AN SDK?

- Since the API is such a pain to work with, Azure offers Software Development Kits (SDKs) for a number of languages
- The principle one for managing Azure resources is the azure-mgmt-resources SDK, which can be installed in Python easily enough:
  - *pip install azure-mgmt-resources azure-mgmt-sql*

THE AZURE SDK

- The SDK allows you to communicate directly with Azure and create, destroy, scale, and monitor your existing Azure resources
- In general, the format of SDK requests is similar to the format used for REST API requests
- JSON-formatted dictionary objects to handle multi-dimensional objects that are passed in as parameters when creating new instances → still requires a lot of work to handle all the requisite fields that need to be specified (DB version, VM size, admin credentials, etc)

THE KICKER

- The major kicker to this is that unlike the AWS SDK, which allowed us to simply log in using our username and password, the Azure SDK still requires we have an Active Directory tenant set up
- Requires the tenant and application IDs, as well as an application secret, all of which needs to be configured beforehand

WHAT DOES THIS MEAN?

- Overall, this has the effect of limiting what is possible programmatically through the Azure SDK
- Namely, it requires that our application using Azure has been pre-defined and given tenancy beforehand, whereas with AWS we could simply log in to our global account and access all resources via that route
- Exception: You can use credentials (and CLI credentials), but only for bugtesting *or* from another Azure resource

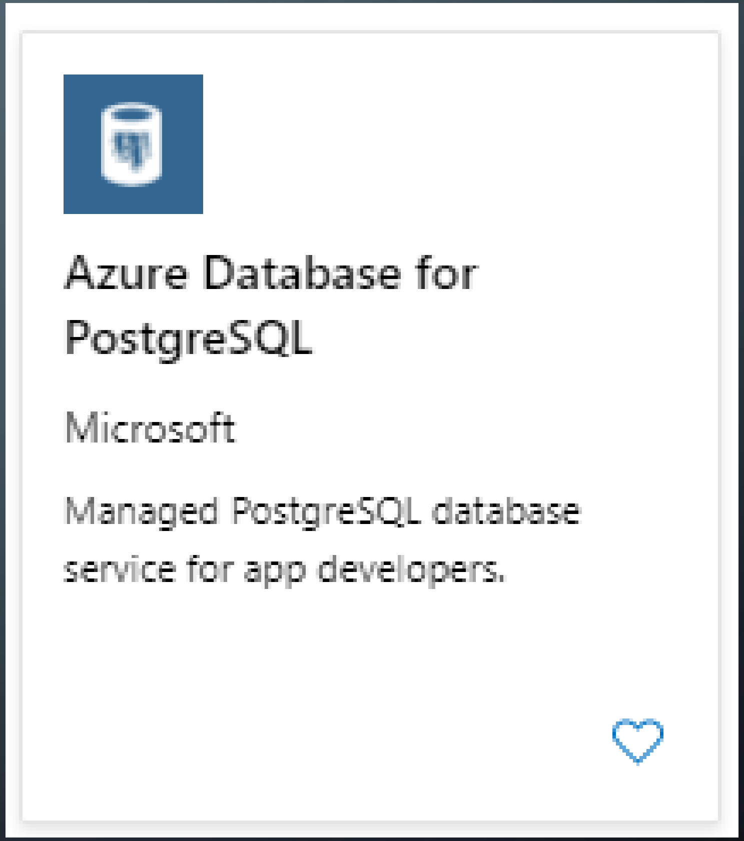
HOW DOES IT STACK UP?

- You could argue that while Azure has a more friendly web user interface for some things (such as firewall management), the SDK is less friendly than Amazon's offerings
- We won't go into further depth on Azure's SDK or API offerings, but a beginner guide to using the SDK in Python is offered here: https://docs.microsoft.com/en-us/azure/python/python-sdk-azure-get-started

AZURE DATABASE FOR POSTGRESQL

USING AZURE WITH POSTGRESQL

- If you're looking for an alternative to MySQL, Azure also offers PostgreSQL databases via the Azure Database for PostgreSQL option


Azure Database for PostgreSQL

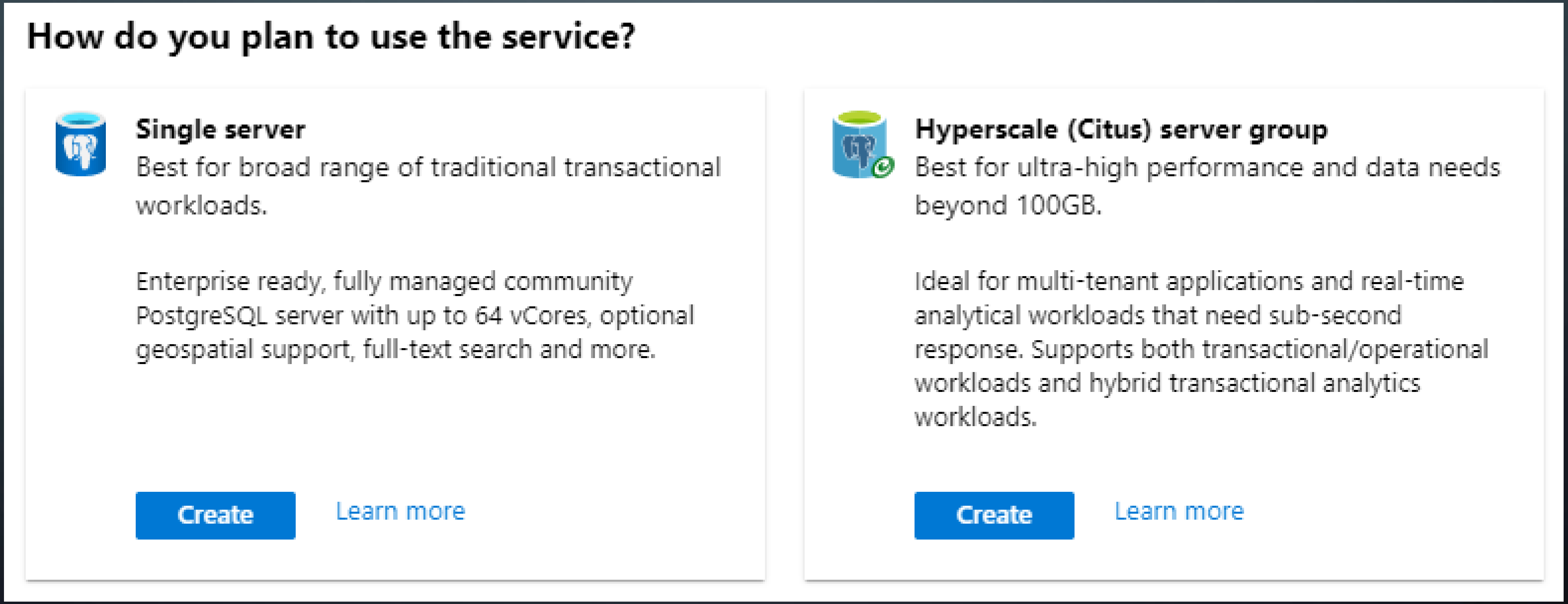
Microsoft

Managed PostgreSQL database service for app developers.

TWO DEPLOYMENT OPTIONS

- Azure offers two different types of PostgreSQL deployment, depending on the type of database you'll be working with:
  - Single server – a traditional single PostgreSQL instance for your database
  - Citus – a clustered solution for large-scale databases

TWO DEPLOYMENT OPTIONS


How do you plan to use the service?

Single server
Best for broad range of traditional transactional workloads.

Enterprise ready, fully managed community PostgreSQL server with up to 64 vCores, optional geospatial support, full-text search and more.

Hyperscale (Citus) server group
Best for ultra-high performance and data needs beyond 100GB.

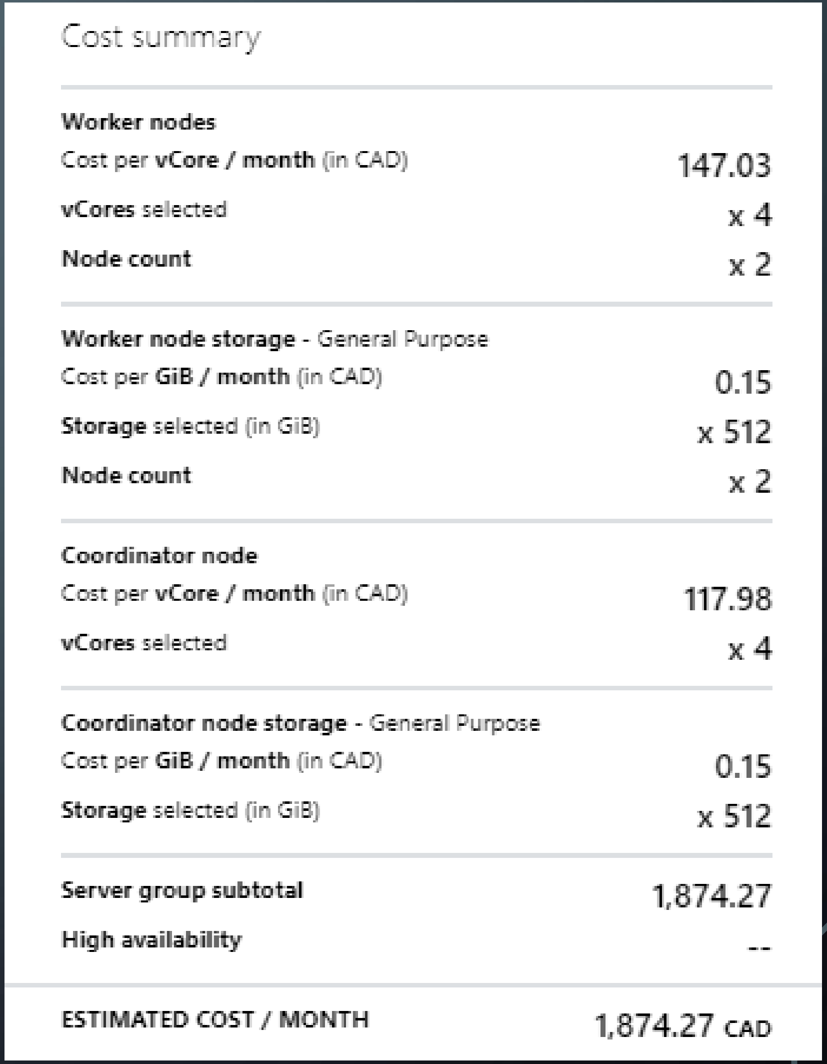
Ideal for multi-tenant applications and real-time analytical workloads that need sub-second response. Supports both transactional/operational workloads and hybrid transactional analytics workloads.

CITUS HYPERSCALE DEPLOYMENTS

- The Citus Hyperscale deployment of PostgreSQL is an interesting option if you are dealing with very large database workloads or need rapid scalability
- It uses PostgreSQL sharding to split the database across multiple servers
- Allows for parallelized queries across multiple servers to increase response times on longer queries

THE DOWNSIDE TO CITUS

- While it would be very interesting to explore the Citus platform on Azure, there is one substantial blocker: cost
- The lowest-size server group that can be rented is two worker nodes with 4 vCores per node, plus a coordinator node (also 4 vCores)
- Minimal monthly cost: $1,874!

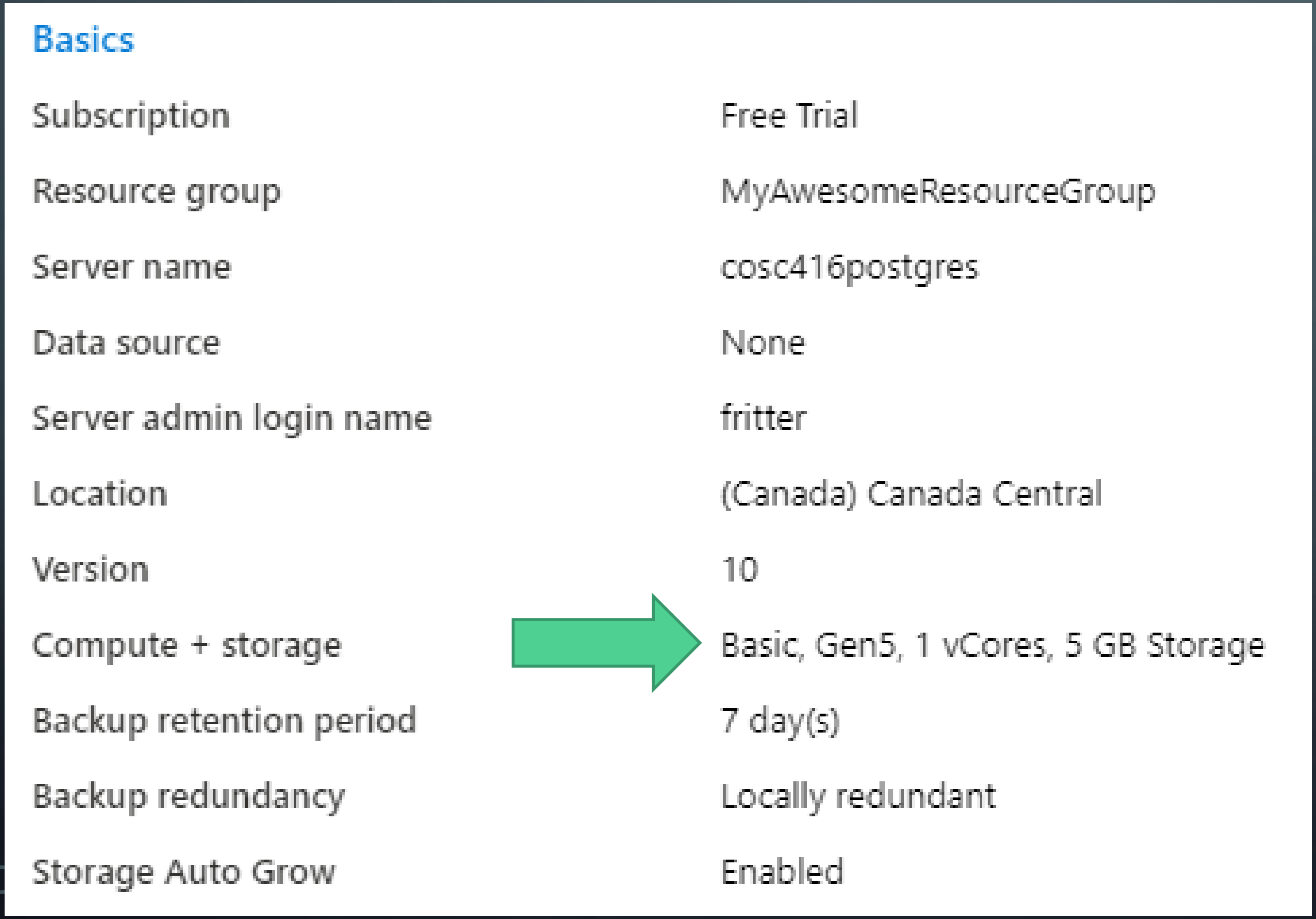

Cost summary

Worker nodes	
Cost per **vCore / month** (in CAD)	147.03
vCores selected	x 4
Node count	x 2
Worker node storage - General Purpose	
Cost per **GiB / month** (in CAD)	0.15
Storage selected (in GiB)	x 512
Node count	x 2
Coordinator node	
Cost per **vCore / month** (in CAD)	117.98
vCores selected	x 4
Coordinator node storage - General Purpose	
Cost per **GiB / month** (in CAD)	0.15
Storage selected (in GiB)	x 512
Server group subtotal	1,874.27
High availability	--
ESTIMATED COST / MONTH	1,874.27 CAD

CONTINUING WITH A SINGLE SERVER INSTALL

- For this course, we'll just be dealing with a single-server installation of PostgreSQL
  - This qualifies under the free Azure trial the same as MySQL instances do
- Steps are identical to those for Azure Database for MySQL – don't forget to change the size of the virtual machine!

POSTGRESQL CREATION SETTINGS



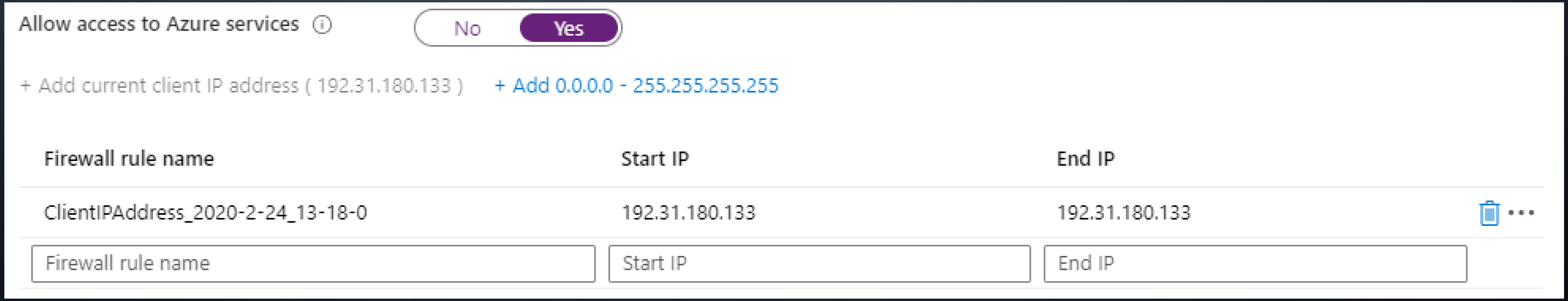
Basics

Subscription	Free Trial
Resource group	MyAwesomeResourceGroup
Server name	cosc416postgres
Data source	None
Server admin login name	fritter
Location	(Canada) Canada Central
Version	10
Compute + storage	Basic, Gen5, 1 vCores, 5 GB Storage
Backup retention period	7 day(s)
Backup redundancy	Locally redundant
Storage Auto Grow	Enabled

- Make sure that you are also using your “Free Trial” subscription as well

CONNECTING TO YOUR POSTGRESQL SERVER

- First steps:
  - As we did previously, whitelist your IP under the “Connection Security” tab, toggle “Allow access” to on, and toggle “Enforce SSL Connection” to “Disabled”:



Allow access to Azure services: No [✓ Yes]

+ Add current client IP address (192.31.180.133)   + Add 0.0.0.0 - 255.255.255.255

Firewall rule name	Start IP	End IP
ClientIPAddress_2020-2-24_13-18-0	192.31.180.133	192.31.180.133
Firewall rule name	Start IP	End IP

INSTALLING PSYCOPG2

- There are many libraries and drivers available for establishing connections to PostgreSQL servers, the most popular library for Python is known as *psycopg2*
- You can install this via pip using the following command:

```
pip install psycopg2
```

- Note: MySQL/MariaDB drivers and libraries are *not* compatible with PostgreSQL!

USING PSYCOPG2

- Psycopg2 is very similar to the MySQL connector library that we've used previously
- Some minor differences in the details: pay attention to the parameter names when creating a connection (they are different than the MySQL connector ones)
- Also, you *must* specify a database that you are working with. By default, the *postgres* database exists and can be used

AN EXAMPLE

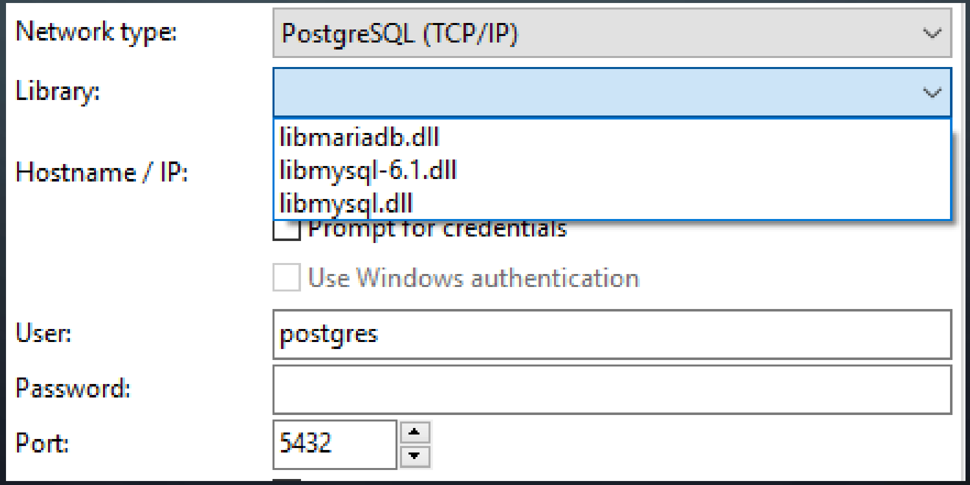
```
import psycopg2

mydb = psycopg2.connect(

host="cosc416postgres.postgres.database.azure.co
m",
  user="fritter@cosc416postgres",
  password="************",
  database="postgres"
)

mycursor = mydb.cursor()
mycursor.execute("SELECT version();")
results = mycursor.fetchall()
for x in results:
       print(str(x))
```

- We import psycopg2
- We create a database connection, specifying our server name, username, password, and database
- We create a cursor which can be used to execute queries and fetch results, just like MySQL

WHAT ABOUT HEIDISQL?

- Unfortunately, it seems that while HeidiSQL offers a PostgreSQL connection option, it is lacking the library required to connect by default ☹
- This may be fixed in newer versions of HeidiSQL?

Note that PostgreSQL uses port 5432 by default!



Network type: PostgreSQL (TCP/IP)

Library:

libmariadb.dll
libmysql-6.1.dll
libmysql.dll

Hostname / IP:

Prompt for credentials

Use Windows authentication

User: postgres

Password:

Port: 5432

AZURE DATABASE FOR POSTGRES TAKEAWAY

- Overall, it’s quite straightforward to create a PostgreSQL database with Azure – it doesn’t require any more effort than creating a standard MySQL database
- However, we will need PostgreSQL-specific drivers or libraries in order to interface with our PostgreSQL database
- Some minor changes but by and large the Python code for connecting to and querying our database is largely unchanged from the MySQL version

THIS WEEK’S LAB AND NEXT LECTURE

SECOND VERSE SAME AS THE FIRST

- In the previous lab, we compared the performance of an RDS MySQL instance to an Azure MySQL instance
- Now, let’s modify it a bit – instead of comparing Amazon and Microsoft, lets compare PostgreSQL and MySQL
- You’ll need to modify your code to perform the same test, but on an Azure MySQL and Azure PostgreSQL database, respectively

CONVERTING YOUR LAB 2 CODE

- What you'll need to do:
  - Spin up an Azure Database for MySQL instance, if you don't have one, as well as an Azure Database for PostgreSQL instance
  - Enable connections to each instance in the security settings, and disable SSL
  - Modify your code to connect and test the PostgreSQL database – you'll need to use the default *postgres* database, or create a new database for your test

MOVING AHEAD

- This will be our last lecture on the Azure platform for now before we begin moving onto our next topic
- Overall, Azure is by and large the same as AWS in terms of instances and management options – mostly differing in the finer details
- Either one would be a good choice for a DBaaS

NEXT TOPIC: GOOGLE BIGQUERY

- For our next topic, we'll be diving into Google Bigquery – a fully managed data warehousing option that works in conjunction with Google Cloud
- Designed to be scalable for *Online Analytical Processing (OLAP)*, aka complex and time-consuming queries
- A major break from the Azure/AWS services that we've seen and worked with so far

SCHEDULE

1. *Azure API & SDK*
2. *Azure PostgreSQL*
3. *Today’s Lab & Next Topic Preview*

SO LONG, FOLKS!