A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark blue background, resembling a circuit board or a neural network.

COSC 416: Topics in Databases (DBaaS)

TOPIC 5: AMAZON RDS REMOTE ADMINISTRATION

SCHEDULE

1. The AWS CLI

- 1. Installation*
- 2. IAM Users*

2. The AWS SDK

- 1. Installation*
- 2. Using the AWS SDK Programmatically*
- 3. The AWS API*

The background of the slide is a dark blue gradient. In the corners, there are abstract white line art designs that resemble circuit boards or neural network connections, with lines and small circles. A large, faint, light-blue circle is centered in the background.

AWS CLI

THE COMMAND-LINE INTERFACE

- Amazon offers a command-line interface (CLI) for AWS services, including Amazon RDS
- This CLI can be used to remotely manage your RDS instances, without the need to log on or use the web GUI

WHICH VERSION TO USE?

- Currently two versions of the AWS CLI are offered
- Version 1 is stable and still supported, and is recommended for production applications
- Version 2 is still in a pre-release evaluation form, and includes some changes that can break older V1 scripts
- V2 is easier to install → comes with prepackaged python distribution, versus V1 which requires you bring your own

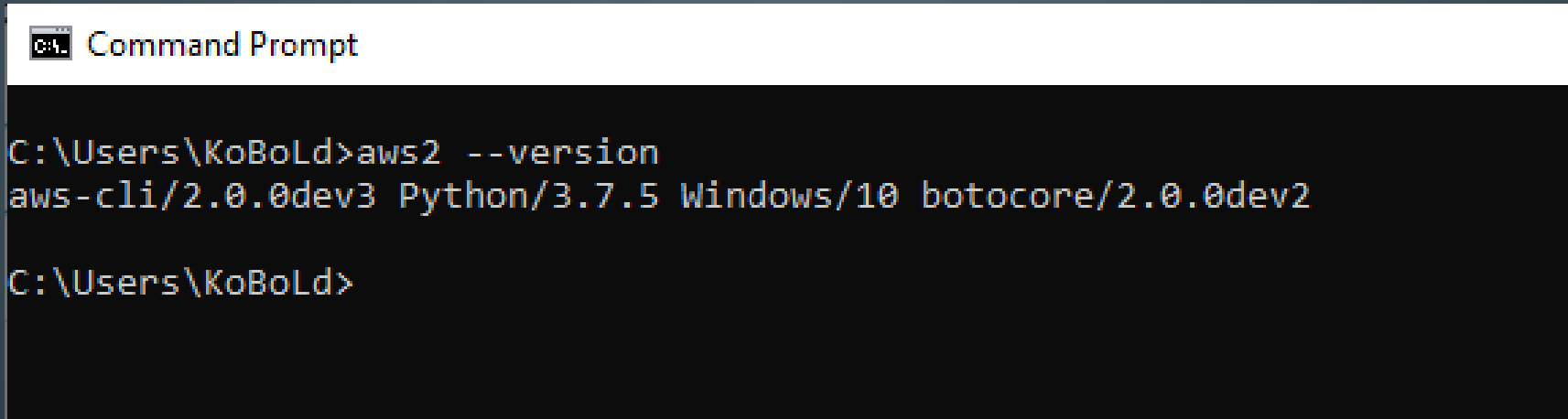
INSTALLING THE CLI

- You can find instructions for installing the CLI on Windows machines here:

<https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-windows.html>

- This is a fairly straightforward procedure using an MSI installer pack – just download and install

CLI INSTALLATION COMPLETION



```
Command Prompt

C:\Users\KoBoLd>aws2 --version
aws-cli/2.0.0dev3 Python/3.7.5 Windows/10 botocore/2.0.0dev2

C:\Users\KoBoLd>
```

- Once you've installed the CLI, you should be able to open up a command prompt and run *aws2 --version* to see your CLI version

SO NOW WHAT?

- At this point, we have the CLI installed, but we can't really do much with it
- First, we'll need to configure a new user in the Amazon Identity and Access Management service that we will use to securely access our RDS instances

IDENTITY AND ACCESS MANAGEMENT

- The Identity and Access Management (IAM) service is a unified user/role system for AWS
- An IAM user may have permissions assigned from a variety of AWS services, such as RDS and EC2
- Each user has an access key ID and a secret access key, which can be used with CLI and Amazon's API to verify who we are

CREATING A USER

- When you are logged into the AWS service, go to this link:


<https://console.aws.amazon.com/iam/home>

- Click on the “Users” tab in the menu, and then click the “Add User” button
- Provide a username, and check the Programmatic Access checkbox so enable access key usage for the user

ADDING PERMISSIONS TO THE USER


- The IAM service handles permissions with user-defined groups
- You'll need to create a new group for your user, giving it a name and choosing the correct permissions
- For our purposes, we'll just enable all RDS permissions (search RDS in the policies to see RDS-related permissions)

FINALIZING OUR USER

 **Success**
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://321449218496.signin.aws.amazon.com/console>

Download .csv

	User	Access key ID	Secret access key
▶ 	MyIAMUser	AKIAUVV652HAKCITPYZO	***** Show

- Once you've applied a group to your new user, you can skip the tag section and head to the last step
- AWS will show you your user access key and secret access key, copy these down

CONFIGURING THE USER IN CLI

- Now that we've created an IAM user account with access to RDS, we can configure the CLI to use that user using *aws2 configure*
- It will prompt you for an access key and secret access key – enter the ones we generated with our user
- It will also prompt for a region – use the region your RDS instances are in (such as *us-east-2*)

USING THE CLI

- Once the user access is configured, we can actually try getting some information from the CLI
- Try running:

aws2 rds describe-db-instances

- You should receive a big array of data related to all of your current RDS instances

DESCRIBE-DB-INSTANCES

```
Command Prompt - aws2 rds describe-db-instances

C:\Users\KoBoLd>aws2 rds describe-db-instances
{
  "DBInstances": [
    {
      "DBInstanceIdentifier": "database-1",
      "DBInstanceClass": "db.t2.micro",
      "Engine": "mysql",
      "DBInstanceStatus": "available",
      "MasterUsername": "admin",
      "Endpoint": {
        "Address": "database-1.chgtqp6x8nhn.us-east-2.rds.amazonaws.com",
        "Port": 3306,
        "HostedZoneId": "Z2XHWR1WZ565X2"
      }
    }
  ]
}
```

- Your output should hopefully look something like this (but much longer). Note the JSON formatted output.

CLI COMMAND SYNTAX

aws2 rds describe-db-instances

- We start each CLI command with `aws2` (the main CLI program)
- We include the service we are trying to access (in this case *rds*) – if you're trying to manage other services, this might be *ec2*, *iam*, *s3*
- Finally, we pass our actual command, in this case the *describe-db-instances* command

CLI RDS COMMANDS

- A full list of available RDS commands can be found here:

<https://docs.aws.amazon.com/cli/latest/reference/rds/>

- You can create new instances and snapshots, delete them, modify them, perform start/stops, etc

A FINAL CLI EXAMPLE

```
C:\Users\KoBoLd>aws2 rds stop-db-instance --db-instance-identifier encryptedinstance
{
  "DBInstance": {
    "DBInstanceIdentifier": "encryptedinstance",
    "DBInstanceClass": "db.t2.small",
    "Engine": "mysql",
    "DBInstanceStatus": "stopping",
    "MasterUsername": "admin",
    "Endpoint": {
      "Address": "encryptedinstance.chgtqp6x8nhn.us-east-2.rds.amazonaws.com",
      "Port": 3306,
      "HostedZoneId": "Z2XHWR1WZ565X2"
    }
  }
}
```

- Stopping the instance with the specified unique database instance identifier

AWS CLI WRAPUP

- Now we have a means of remotely managing our RDS instances without the GUI
- The AWS CLI also interfaces with Amazon's Software Development Kit (SDK) which we'll see in the next section
- Even if you don't plan to use the CLI, having it installed before you install the SDK will make things easier

The image features the text "AWS SDK" in a white, bold, sans-serif font, centered on a dark blue background. The background is decorated with faint, light blue concentric circles and stylized white circuit board traces with circular nodes at the corners. The traces are located in the top-left, top-right, bottom-left, and bottom-right corners of the image.

AWS SDK

MANAGING THINGS PROGRAMMATICALLY

- Alright, so the CLI provides us command-line access, but what if we want to use RDS management functions programmatically?
- Well, we could run them as a terminal command within our script, but that's messy and generally pretty insecure

THE AWS SDK

- Luckily, Amazon has released software development kits (SDKs) for a variety of languages that allow us to easily access these functions within our programming language of choice
- Just install the SDK, and import it as you would any other library (depending on the language)

SUPPORTED LANGUAGES

- AWS SDK offers support for the following languages:
 - C++, Go, Java, JS, Node.js, PHP, Ruby, Python, .NET
- For the purposes of the lab, you may choose whichever language you are most comfortable in – examples will be given in Python

INSTALLING THE SDK

- Head on over to:

<https://aws.amazon.com/tools/>

- Select your language and then click the SDK link under “Build Applications”
- From there, click “Getting Started” for step-by-step install instructions (for Python, the install will be using the *pip* utility)

SDK-CLI AUTHENTICATION

- If you already have the CLI installed and an IAM user with permissions configured, these permissions should be available to the SDK as well
- If not, you can also manually set up the access key and secret access key credentials in the specified credentials file

USING THE SDK

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8 import boto3
9
10 myRds = boto3.client('rds', region_name='us-east-2')
11
```

- In Python, the SDK library is known as *boto3*
- We can simply import that, then use the *client* function to connect to the RDS service
- Note we are also specifying a region here manually

USING THE SDK – RUNNING COMMANDS

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8 import boto3
9
10 myRds = boto3.client('rds', region_name='us-east-2')
11 print(myRds.describe_db_instances())
```

- The client object we've created has a number of functions associated with it that align with the CLI-available commands for RDS
- In this case, `describe_db_instances()`

SDK AVAILABLE COMMANDS

- A complete listing of boto3 RDS client commands can be found here:

<https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rds.html>

- These will, in general, completely align with their CLI equivalents

HANDLING OUTPUT

- The AWS SDK by default will output data in JSON format
- Depending on the language you are using, the client may provide the data ready-to-go; in Python it will return the data in a Dict (dictionary) object
- Otherwise, easy to convert using pre-existing JSON libraries that are available on most (if not all) platforms

A PYTHON EXAMPLE

```
import boto3

myRds = boto3.client('rds', region_name='us-east-2')
data = myRds.describe_db_instances()
for instance in data["DBInstances"]:
    print(instance["DBInstanceIdentifier"] + ", " + instance["DBInstanceClass"])
```

- We create a client, and ask it to describe our DB instances
- We loop over the output for each instance and print out the identifier, and instance class (machine size)

```
In [14]: runfile('C:/Users/KoBoLd/.spyder-py3/temp.py', wdir='C:/Users/KoBoLd/.spyder-py3')
database-1, db.t2.micro
read-instance-db1, db.t2.micro
```

A NOTE ABOUT USING THE SDK

- You'll have to spend some time reading documentation to understand which fields are and are not necessary for a given command
- Many fields rely on pre-defined string constants like *mysql* or *db.t2.micro* as arguments
- My suggestion is just play around with it, see what errors occur, and figure out what will work and what won't

EXAMPLE – CREATING A NEW INSTANCE

```
import boto3

myRds = boto3.client('rds', region_name='us-east-2')
database = myRds.create_db_instance(
    DBInstanceIdentifier="db-sdk-example",
    AllocatedStorage=10,
    Engine="mysql",
    DBInstanceClass="db.t2.micro",
    MasterUsername="root",
    MasterUserPassword="password"
)
```

- The `create_db_instance()` function has many arguments, but only those shown above are “required” – keep an eye out on the instance class and allocated storage size (\$\$)

THE UNDERLYING API

- The AWS CLI and SDK are relying on an underlying API to do the management side of things
- If we wanted, we *could* use the API directly, making our HTTP requests manually with all the required headers and information
- This is however a much more labour intensive route, and it makes a lot more sense to use the SDK

AN EXAMPLE API REQUEST

```
https://rds.us-east-1.amazonaws.com/  
?Action=CreateDBInstance  
&AllocatedStorage=10  
&DBInstanceClass=db.t2.micro  
&DBInstanceIdentifier=db-sdk-example  
&Engine=MySQL  
&MasterUserPassword=password  
&MasterUsername=root  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4 &Version=2014-09-01  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request  
&X-Amz-Date=20140424T194844Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-  
Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

SIGNATURE V4 SECURITY

- The large red block at the end of the API request is related to the use of the Signature V4 authentication process
- Each request is signed using your secret access key
- Amazon compares the signature to the one it's expecting, and will only accept the API request if the signature matches

MANUAL VS SDK?

- Amazon provides an example Python script using manually-generated requests to the API:

<https://docs.aws.amazon.com/general/latest/gr/signv4-signed-request-examples.html>

- In general, this is kind of a pain and requires a lot more work than using the SDK to achieve the same results, but it's an option if your language of choice doesn't have an SDK available

INSTANCE SETUP AND TAKEDOWN TIME

- Whether we use the GUI, CLI, SDK, or the API directly, things still take time – setting up and tearing down instances isn't instantaneous
- What does this mean? It means if we create an instance in our application, we have to wait for it to come online before we can connect to it

CHECKING INSTANCE STATUS

- So, if we create a new database instance that we want to use, we'll need to check if it's finished setting up yet
- This isn't too difficult – `describe_db_instances()` has a field called "DBInstanceStatus" that we can use to check the current status of an instance
- If the status is other than "available", then the instance is being set up/torn down/rebooted/etc

A SIMPLE EXAMPLE IN PYTHON

```
import boto3, time
```

```
myRds = boto3.client('rds', region_name='us-east-2')  
myRds.create_db_instance(  
    DBInstanceIdentifier="db-sdk-example",  
    AllocatedStorage=10,  
    Engine="mysql",  
    DBInstanceClass="db.t2.micro",  
    MasterUsername="root",  
    MasterUserPassword="password"  
)
```

```
while True:
```

```
    status = myRds.describe_db_instances(DBInstanceIdentifier="db-sdk-example")  
    if status["DBInstances"][0]["DBInstanceStatus"] == "available":  
        print("DB is online")  
        break;  
    else:  
        print("DB is not available")  
        time.sleep(10)
```

First, we instantiate the client instance for RDS, with the region set to 'us-east-2'

A SIMPLE EXAMPLE IN PYTHON

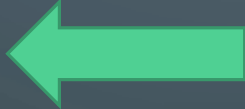
```
import boto3, time
```

```
myRds = boto3.client('rds', region_name='us-east-2')
myRds.create_db_instance(
    DBInstanceIdentifier="db-sdk-example",
    AllocatedStorage=10,
    Engine="mysql",
    DBInstanceClass="db.t2.micro",
    MasterUsername="root",
    MasterUserPassword="password"
)
```

```
while True:
```

```
    status = myRds.describe_db_instances(DBInstanceIdentifier="db-sdk-example")
    if status["DBInstances"][0]["DBInstanceStatus"] == "available":
        print("DB is online")
        break;
    else:
        print("DB is not available")
    time.sleep(10)
```

We create our database with the specified parameters, which is identified by the name 'db-sdk-example'



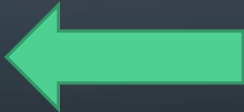
A SIMPLE EXAMPLE IN PYTHON

```
import boto3, time
```

```
myRds = boto3.client('rds', region_name='us-east-2')
myRds.create_db_instance(
    DBInstanceIdentifier="db-sdk-example",
    AllocatedStorage=10,
    Engine="mysql",
    DBInstanceClass="db.t2.micro",
    MasterUsername="root",
    MasterUserPassword="password"
)
```

```
while True:
```

```
    status = myRds.describe_db_instances(DBInstanceIdentifier="db-sdk-example")
    if status["DBInstances"][0]["DBInstanceStatus"] == "available":
        print("DB is online")
        break;
    else:
        print("DB is not available")
        time.sleep(10)
```



Every ten seconds, we check if the status of the 'db-sdk-example' database is 'available', otherwise we keep checking

HOW LONG DOES THIS TAKE?

- Using the previous example, I got 50 “not-available” messages over 500 seconds before the database became available
- That works out to about 8.33 minutes total time from calling the `instance_create()` to being able to use the created instance
- Probably better to set the timer to something like 30 seconds or 1 minute to reduce the number of API requests going out

AWS SDK TAKEAWAY

- The SDK provides a simple way of using the Amazon AWS API in a programmatic fashion, without having to format our own requests
- In the lab today, we'll be setting up the CLI and SDK, and then creating a simple script to start up and take down database instances

SCHEDULE

1. The AWS CLI

1. Installation

2. IAM Users

2. The AWS SDK

1. Installation

2. Using the AWS SDK Programmatically

3. The AWS API

The image features a dark blue gradient background with faint, stylized circuit board traces in the corners. These traces are composed of thin white lines and small white circles, resembling electronic components or data paths. The text "SO LONG, FOLKS!" is centered in a bold, white, sans-serif font.

SO LONG, FOLKS!