Hands-on – creating and accessing your AWS account

Source: Data Engineering with AWS, Second Edition by Gareth Eagar (Packt, 2023)
https://www.packtpub.com/en-us/product/data-engineering-with-aws-9781804614426

The projects in this book require you to access an AWS account with administrator privileges. If you already have administrator privileges for an AWS account and know how to access the AWS Management Console, you can skip this section and move on to Chapter 2, Data Management Architectures for Analytics.

If you are making use of a corporate AWS account, you will want to check with your AWS cloud operations team to ensure that your account has administrative privileges. Even if your daily-use account does not allow full administrative privileges, your cloud operations team may be able to create a sandbox account for you.

What is a sandbox account?

A sandbox account is an account isolated from your corporate production systems with relevant guardrails and governance in place and is used by many organizations to provide a safe space for teams or individual developers to experiment with cloud services.

If you cannot get administrative access to a corporate account, you will need to create a personal AWS account or work with your cloud operations team to request specific permissions needed to complete each section. The exercises in this book assume you have administrative access and the full details of required granular permissions will not be covered, but you can review the AWS documentation for information on granular permission requirements for each service.

An important note about costs associated with hands-on tasks in this book

If you are creating a new personal account or using an existing personal account, you will incur and be responsible for AWS costs as you follow along in this book. While some services may fall under AWS Free Tier usage, some of the services covered in this book will not. We strongly encourage you to set up budget alerts within your account and to regularly check your billing console.

See the AWS documentation on setting up billing alarms to monitor your costs at https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/monitor_estimated_charges_with_cloudwatch.html.

Creating a new AWS account

To create a new AWS account, you will need the following things:

- An email address (or alias) that has not been used before to register an AWS account
- A phone number that can be used for important account verification purposes
- Your credit or debit card, which will be charged for AWS usage outside of the Free Tier

A tip regarding the phone number you use when registering:

It is important that you keep your contact details up to date for your AWS account as, if you lose access to your account, you will need access to the email address and phone number registered for the account to restore access. If you expect that your contact number may change in the future, consider registering a virtual number that you will always be able to access and that you can forward to your primary number. One such service that enables this is Google Voice (`http://voice.google.com`).

The following steps will guide you through creating a new AWS account:

1. Navigate to the AWS landing page at `http://aws.amazon.com`.
2. Click on the Create an AWS Account link in the top right-hand corner.
3. Provide an email address, provide a name for your account, and then click on Verify email address. You will be emailed a verification code to verify your email, which you need to enter on the form to continue.

A tip about reusing an existing email address:

Some email systems support adding a "+" sign followed by a few characters to the end of the username portion of your email address in order to create a unique email address that still goes to your same mailbox. For example, `atest.emailaddress@gmail.com` and `atest.emailaddress+dataengineering@gmail.com` will both go to the primary email address inbox. If you have used your primary email address previously to register an AWS account, you can use this tip to provide a unique email address during registration but still have emails delivered to your primary account.

4. Once you verify using the code emailed to you, specify a new secure password for your account (one that you have not used elsewhere). Then click on Continue.
5. Select Business or Personal for the account type (note that the functionality and tools available are the same no matter which one you pick).
6. Provide the requested personal information and then, after reviewing the terms of the AWS Customer Agreement, click the checkbox if you agree to the terms, and then click Continue.
7. Provide a credit or debit card for payment information and select Verify and Continue.
8. Provide a phone number for a verification text or call, enter the characters shown for the security check, and complete the verification.
9. Select a support plan (basic support is free, but only provides self-service access to support resources) and complete the signup.
10. You will receive a notification that your account is being activated. This usually completes in a few minutes, but it can take up to 24 hours. Check your email to confirm account activation. What to do if you don't receive a confirmation email within 24 hours
    If you do not receive an email confirmation within 24 hours confirming that your account has been activated, follow the troubleshooting steps provided by AWS Premium Support at `https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/`.

Accessing your AWS account

Once you have received the confirmation email confirming that your account has been activated, follow these steps to access your account and create a new admin user:

1. Access the AWS console login page at http://console.aws.amazon.com.
2. Make sure Root user is selected, and then enter the email address that you used when creating the account.
3. Enter the password that you set when creating the account.

Best practices for securing your account

When you log in using the email address you specified when registering the account, you are logging in as the account's root user. It is a recommended best practice that you do not use this login for your day-to-day activities but rather, only use this when performing activities that require the root account, such as creating your first Identity and Access Management (IAM) user, deleting the account, or changing your account settings. For more information, see https://docs.aws.amazon.com/IAM/latest/UserGuide/id_root-user.html.

It is also strongly recommended that you enable Multi-Factor Authentication (MFA) on this and other administrative accounts. To enable this, see https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_mfa_enable_virtual.html.

In the following steps, we are going to create a new IAM administrative user account:

1. In the AWS Management Console, confirm which Region you are currently in. You can select any region, such as the region closest to you geographically.

Selecting a region

The region you select in the AWS console is the geographical area of the world where the AWS resources you create will be deployed. It generally makes sense to deploy to the region closest to where you are located; however, this is not always the case. For example, not all AWS services are available in all regions (for a list of services available per region, see https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/).
Another factor to consider is that the pricing for AWS services differs from region to region, so also take this into account when selecting a region to use for the exercises in this book. Finally, make sure that you are always in the same region when working through the exercises in each chapter.
For more information on selecting a region, refer to the AWS blog post What to Consider when Selecting a Region for your Workloads at https://aws.amazon.com/blogs/architecture/what-to-consider-when-selecting-a-region-for-your-workloads/.

In the following screenshot, the user is in the Ohio region (also known as us-east-2).

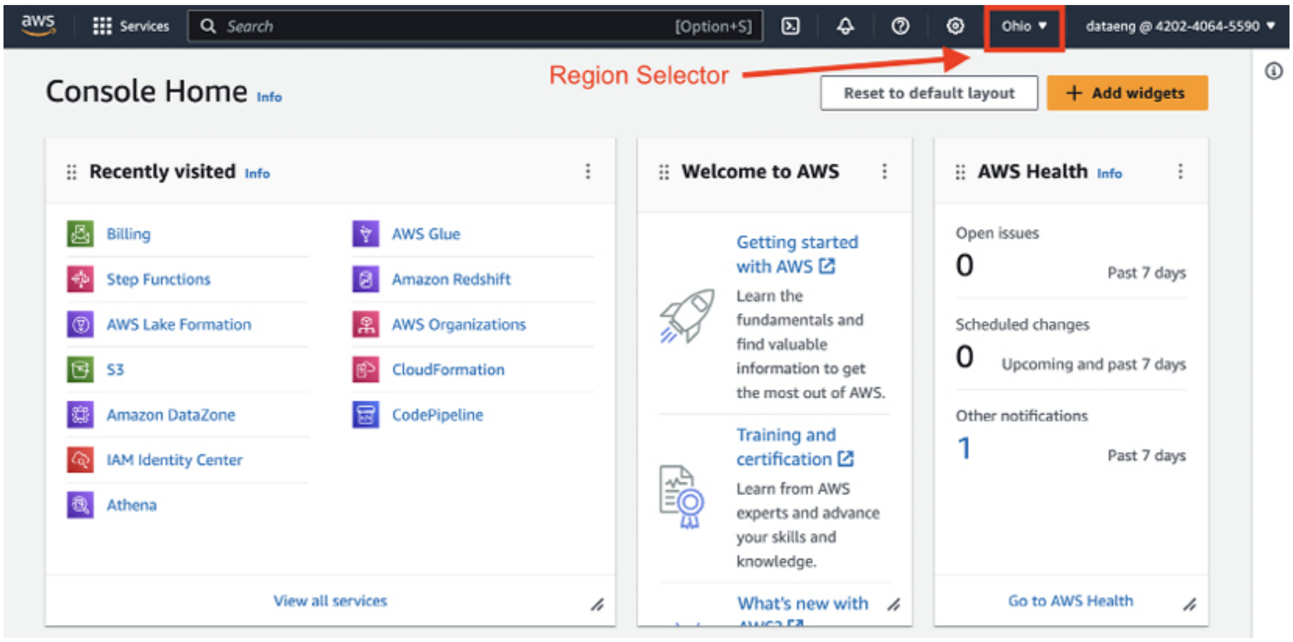
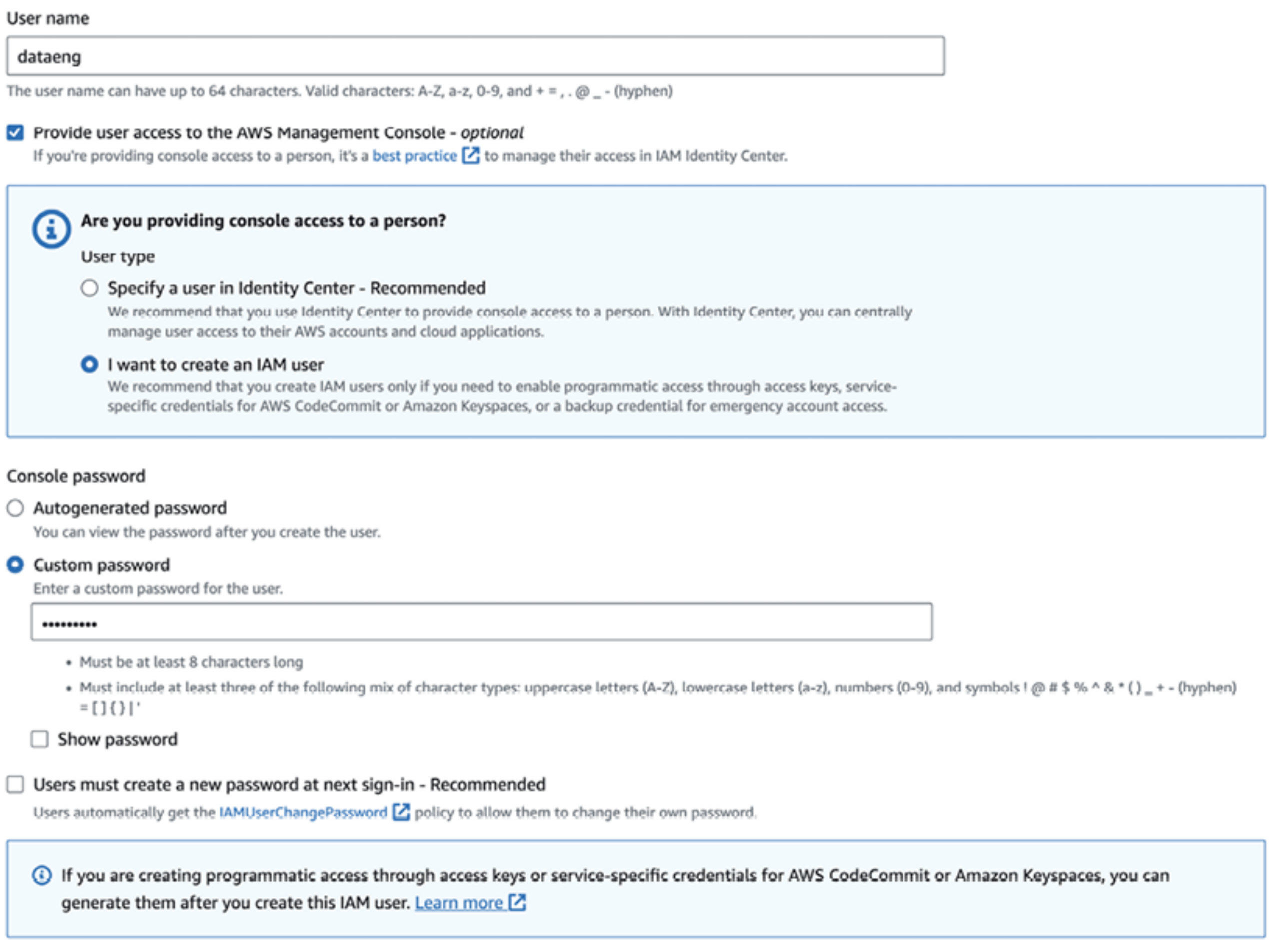


Figure 1.1: AWS Management Console

2. In the Search bar at the top of the screen, type in `IAM` and press Enter. This brings up the console for Identity and Access Management (IAM).
3. On the left-hand side menu, click Users and then Add users.
4. Provide a username, and then select the checkbox for Enable console access - optional.
5. Select Custom password, provide a password for console access, select whether to force a password change on the next login, then click Next.


User name

dataeng

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

[✓] Provide user access to the AWS Management Console - *optional*

If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

Are you providing console access to a person?

User type

( ) Specify a user in Identity Center - Recommended

We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

[✓ I want to create an IAM user]

We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Console password

( ) Autogenerated password

You can view the password after you create the user.

[✓ Custom password]

Enter a custom password for the user.

•••••••••

- Must be at least 8 characters long
- Must include at least three of the following mix of character types: uppercase letters (A-Z), lowercase letters (a-z), numbers (0-9), and symbols ! @ # $ % ^ & * ( ) _ + - (hyphen) = [ ] { } | '

[ ] Show password

[ ] Users must create a new password at next sign-in - Recommended

Users automatically get the IAMUserChangePassword policy to allow them to change their own password.

If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. Learn more

Figure 1.2: Creating a new user in the AWS Management Console

1. For production accounts, it is best practice to grant permissions with a policy of least privilege, giving each user only the permissions they specifically require to perform their role. However, AWS managed policies can be used to cover common use cases in test accounts, and so to simplify the setup of our test account, we will use the AdministratorAccess managed policy. This policy gives full access to all AWS resources in the account. On the Set permissions screen, select Attach policies directly from the list of policies, select AdministratorAccess, then click Next: Tags.
2. Optionally, specify tags (key-value pairs), then click Next.
3. Review the settings, and then click Create user.
4. Take note of the Console sign-in URL link that you will use to sign into your account.

For the remainder of the tutorials in this book, you should log in using the URL link provided and the username and password you set for your IAM user. You should also strongly consider enabling MFA for this account, a recommended best practice for all accounts with administrator permissions.

Hands-on – using the AWS Command Line Interface (CLI) to create Simple Storage Service (S3) buckets

In Chapter 1, An Introduction to Data Engineering, you created an AWS account and an AWS administrative user and then ensured you could access your account. Console access allows you to access AWS services and perform most functions; however, it can also be useful to interact with AWS services via the CLI at times.

In this hands-on section, you will learn how to access the AWS CLI, and then use the CLI to create Amazon S3 buckets (a storage container in the Amazon S3 service).

Accessing the AWS CLI

The AWS CLI can be installed on your personal computer/laptop or can be accessed from the AWS Console. To access the CLI on your personal computer, you need to generate a set of access keys.

Your access keys consist of an access key ID (which is comparable to a username), and a secret access key (which is comparable to a password). With these two pieces of information, you can authenticate as your user and perform all actions that your user is authorized to perform.

However, if anyone else were able to get your access key ID and secret access key, then they would have access to the same permissions. In the past, there have been instances where users have accidentally exposed their access key ID and secret access key, enabling malicious third parties to fraudulently use their account.

As a result, the easiest and most secure way to access the CLI is via the AWS CloudShell service, which is accessible via the console.

Using AWS CloudShell to access the CLI

CloudShell provides a terminal interface in your web browser, as part of the AWS Console, that can be used to explore and manage AWS resources, using the permissions of the user with which you log in to the console. With this approach, there is no need to generate access keys for your user.

Use the following steps to access the CLI via AWS CloudShell:

1. Log in to the AWS Console (`https://console.aws.amazon.com`) using the credentials you created in Chapter 1.
2. Click on the CloudShell icon, or use the search bar to search for the CloudShell service.

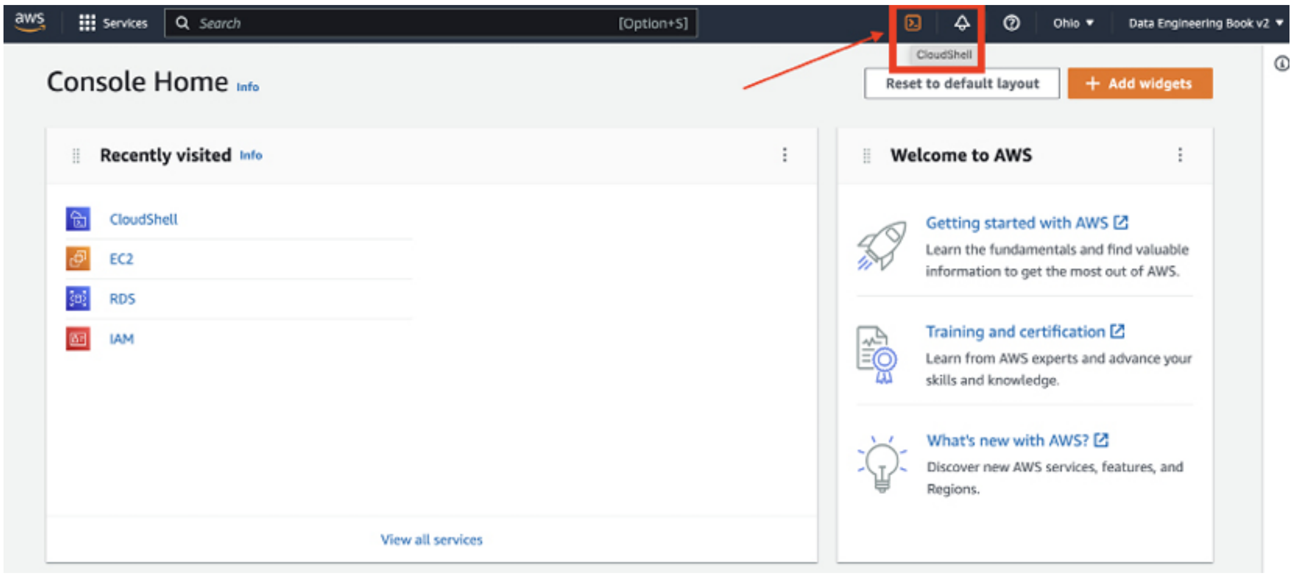
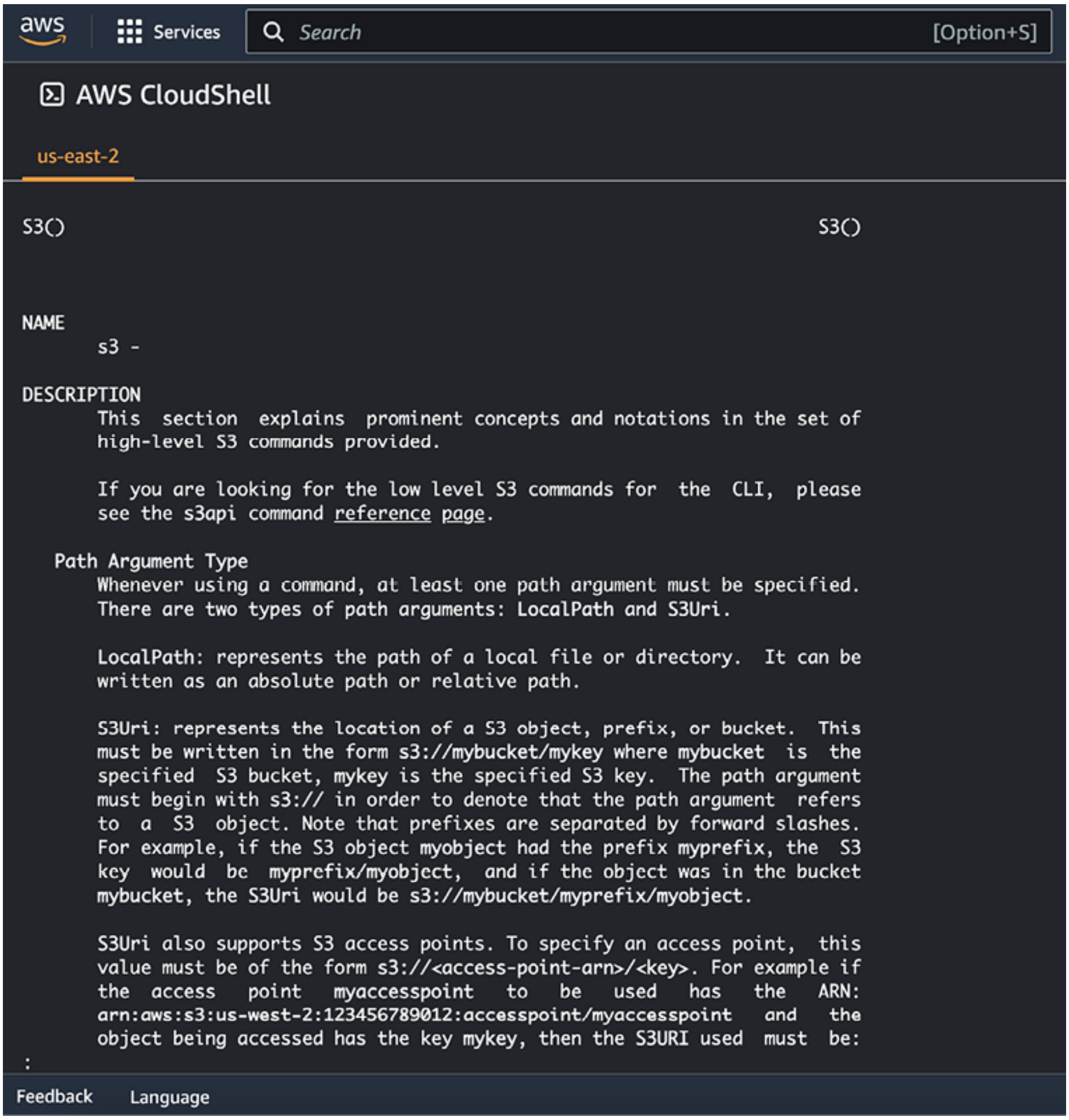


Figure 2.10: Accessing the AWS CloudShell service in the console

3. To interact with AWS services, you run the command `aws`. To learn how to interact with a specific service, you can run the `aws` command, followed by the name of the AWS service you want to interact with (such as S3), followed by `help`. For example, to learn how to use the Amazon S3 service, run the following command, which will display the help page for the S3 service:

```
aws s3 help
```



```
S3()                                                          S3()



NAME
       s3 -

DESCRIPTION
       This  section  explains  prominent concepts and notations in the set of
       high-level S3 commands provided.

       If you are looking for the low level S3 commands for  the  CLI,  please
       see the s3api command reference page.

   Path Argument Type
       Whenever using a command, at least one path argument must be specified.
       There are two types of path arguments: LocalPath and S3Uri.

       LocalPath: represents the path of a local file or directory.  It can be
       written as an absolute path or relative path.

       S3Uri: represents the location of a S3 object, prefix, or bucket.  This
       must be written in the form s3://mybucket/mykey where mybucket  is  the
       specified  S3 bucket, mykey is the specified S3 key.  The path argument
       must begin with s3:// in order to denote that the path argument  refers
       to  a  S3  object. Note that prefixes are separated by forward slashes.
       For example, if the S3 object myobject had the prefix myprefix, the  S3
       key  would  be  myprefix/myobject,  and if the object was in the bucket
       mybucket, the S3Uri would be s3://mybucket/myprefix/myobject.

       S3Uri also supports S3 access points. To specify an access point,  this
       value must be of the form s3://<access-point-arn>/<key>. For example if
       the  access   point   myaccesspoint   to   be   used   has   the   ARN:
       arn:aws:s3:us-west-2:123456789012:accesspoint/myaccesspoint   and   the
       object being accessed has the key mykey, then the S3URI used  must  be:
:
```

Figure 2.11: Displaying the AWS CLI help page for Amazon S3

4. Press the SPACE bar to display subsequent pages of the help, or press the letter q to quit the help pages.

Creating new Amazon S3 buckets

Amazon S3 is an object storage service that offers near-unlimited capacity with high levels of durability and availability. To store data in S3, you need to create a bucket. Once created, the bucket can store any number of objects.

Each S3 bucket needs to have a globally unique name, and it is recommended that the name be DNS-compliant. For more information on rules for bucket names, see https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucketnamingrules.html:

1. To create an S3 bucket using the AWS CLI, run the following command at the Command Prompt in CloudShell. In the following command, replace `<bucket-name>` with a unique name for your bucket:

```
$ aws s3 mb s3://<bucket-name>
```

Remember that the bucket name you specify here must be globally unique. If you attempt to create a bucket using a name that another AWS account has used, you will see an error similar to the following:

```
$ aws s3 mb s3://test-bucket
make_bucket failed: s3://test-bucket An error occurred (BucketAlreadyExists)
when calling the CreateBucket operation: The requested bucket name is not
available. The bucket namespace is shared by all users of the system. Please
select a different name and try again.
```

If your `aws s3 mb` command returned a message similar to the following, then congratulations! You have created your first bucket.

```
make_bucket: <bucket-name>
```

2. We can now create additional buckets that we are going to use in some of the hands-on exercises in later chapters. As discussed earlier in this chapter, data lakes generally have multiple zones for data at different stages. To set up our data lake, we want to create S3 buckets for the landing zone, clean zone, and curated zone. Refer back to the section titled The storage layer and storage zones earlier in this chapter for a description of each of the zones.

In order to ensure that each bucket name created is globally unique, you can append some unique characters to each name, such as your initials, and if necessary, some numbers. In the examples below, I am going to append `gse23` to each bucket name to ensure it is unique.

Run the following three commands in the CloudShell terminal to create your buckets (replacing `gse23` with your own identifier).

```
$ aws s3 mb s3://dataeng-landing-zone-gse23
$ aws s3 mb s3://dataeng-clean-zone-gse23
$ aws s3 mb s3://dataeng-curated-zone-gse23
```

We have now created the storage buckets that will form the foundation of the three zones of our data lake (landing zone, clean zone, and curated zone). In later chapters, we will ingest data into the landing zone, and then create transformed copies of that data in the other zones.

Hands-on – triggering an AWS Lambda function when a new file arrives in an S3 bucket

In the hands-on portion of this chapter, we're going to configure an S3 bucket to automatically trigger a Lambda function whenever a new file is written to the bucket. In the Lambda function, we're going to make use of an open-source Python library called AWS SDK for pandas, created by AWS Professional Services to simplify common ETL tasks when working in an AWS environment. We'll use the AWS SDK for pandas library to convert a CSV file into Parquet format and then update AWS Glue Data Catalog.

Converting a file into Parquet format is a common transformation in order to improve analytic query performance against our data. This can either be done in bulk (such as by using an AWS Glue job that runs every hour to convert files received in the past hour), or it can be done as each file arrives, as we are doing in this hands-on exercise. A similar approach can be used for other use cases, such as updating a `total_sales` value in a database as files are received with daily sales figures from a company's retail stores across the world.

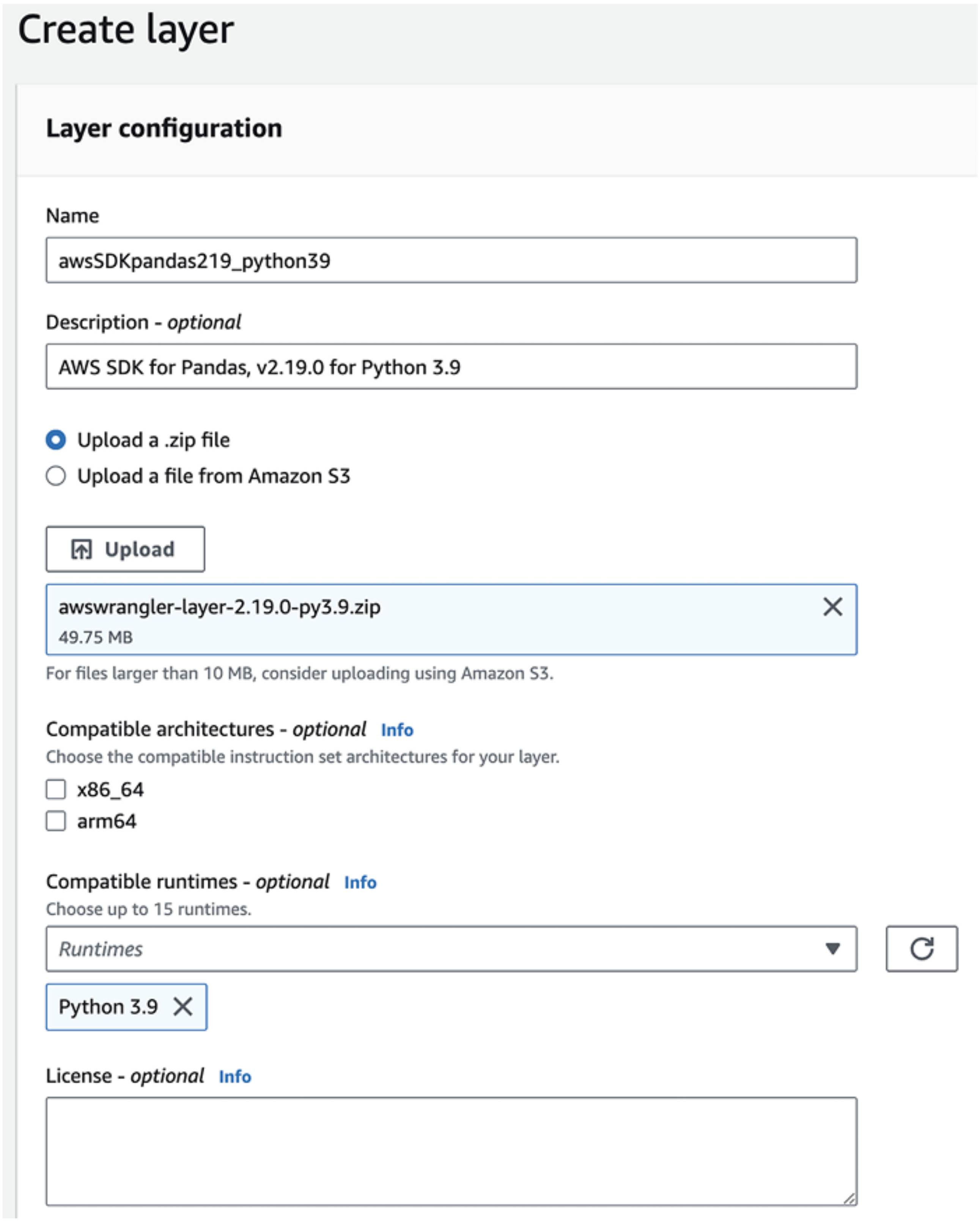
Let's get started with the hands-on section of this chapter.

Creating a Lambda layer containing the AWS SDK for pandas library

Lambda layers allow your Lambda function to bring in additional code, packaged as a .zip file. In our use case, the Lambda layer is going to contain the AWS SDK for pandas Python library, which we can then attach to any Lambda function where we want to use the library.

To create a Lambda layer, do the following:

1. Access the 2.19.0 version of the AWS SDK for pandas library in GitHub at https://github.com/aws/aws-sdk-pandas/releases. Under Assets, download the `awswrangler-layer-2.19.0-py3.9.zip` file to your local drive. Note that there is a direct link to this file on the GitHub site for this book at https://github.com/PacktPublishing/Data-Engineering-with-AWS-2nd-edition/tree/main/Chapter03.
2. Log in to the AWS Management Console as the administrative user you created in Chapter 1, An Introduction to Data Engineering (https://console.aws.amazon.com).
3. Make sure that you are in the Region that you have chosen for performing the hands-on sections in this book. The examples in this book use the `us-east-2` (Ohio) Region.
4. In the top search bar of the AWS console, search for and select the Lambda service.
5. In the left-hand menu, under Additional Resources, select Layers, and then click on Create layer.
6. Provide a name for the layer (for example, `awsSDKpandas219_python39`) and an optional description, and then upload the `.zip` file you downloaded from GitHub. For Compatible runtimes-optional, select Python 3.9, and then click Create. The following screenshot shows the configuration for this step:



Create layer

Layer configuration

Name

awsSDKpandas219_python39

Description - *optional*

AWS SDK for Pandas, v2.19.0 for Python 3.9

● Upload a .zip file
○ Upload a file from Amazon S3

Upload

awswrangler-layer-2.19.0-py3.9.zip
49.75 MB

For files larger than 10 MB, consider uploading using Amazon S3.

Compatible architectures - *optional* Info
Choose the compatible instruction set architectures for your layer.

☐ x86_64
☐ arm64

Compatible runtimes - *optional* Info
Choose up to 15 runtimes.

Runtimes

Python 3.9

License - *optional* Info

Figure 3.10: Creating and configuring an AWS Lambda layer

By creating a Lambda layer for the AWS SDK for pandas library, we can use AWS SDK for pandas in any of our Lambda functions just by ensuring this Lambda layer is attached to the function.

Creating an IAM policy and role for your Lambda function

In the previous chapter, we created three Amazon S3 buckets – one for a landing zone (for ingestion of raw files), one for a clean zone (for files that have undergone initial processing and optimization), and one for the curated zone (to contain our finalized datasets, ready for consumption).

In this section, we will create a Lambda function to be triggered every time a new file is uploaded to our landing zone S3 bucket. The Lambda function will process the file and write out a new version of the file to a target bucket (our clean zone S3 bucket).

For this to work, we need to ensure that our Lambda function has the following permissions:

1. Read our source S3 bucket (for example, `dataeng-landing-zone-<initials>`).
2. Write to our target S3 bucket (for example, `dataeng-clean-zone-<initials>`).
3. Write logs to Amazon CloudWatch.
4. Access to all Glue API actions (to enable the creation of new databases and tables).

To create a new AWS IAM role with these permissions, follow these steps:

1. In the search bar at the top of the AWS console, search for and select the IAM service, and in the left-hand menu, select Policies and then click on Create policy.
2. By default, the Visual editor tab is selected, so click on JSON to change to the JSON tab.
3. Provide the JSON code from the following code blocks, replacing the boilerplate code. Note that you can also copy and paste this policy by accessing the policy on this book's GitHub page (https://github.com/PacktPublishing/Data-Engineering-with-AWS-2nd-edition/blob/main/Chapter03/DataEngLambdaS3CWGluePolicy.json). Note that if doing a copy and paste from the GitHub copy of this policy, you must replace `dataeng-landing-zone-<initials>` with the name of the landing zone bucket you created in Chapter 2, and replace `dataeng-clean-zone-<initials>` with the name of the clean zone bucket you created in Chapter 2. This first block of the policy configures the policy document and provides permissions for using CloudWatch log groups, log streams, and log events:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
```

This next block of the policy provides permissions for all Amazon S3 actions (`get` and `put`) that are in the Amazon S3 bucket specified in the resource section (in this case, our clean zone and landing zone buckets). Make sure you replace `dataeng-clean-zone-<initials>` and `dataeng-landing-zone-<initials>` with the names of the S3 buckets you created in Chapter 2:

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
"Resource": [
                "arn:aws:s3:::dataeng-landing-zone-INITIALS/*",
                "arn:aws:s3:::dataeng-landing-zone-INITIALS",
                "arn:aws:s3:::dataeng-clean-zone-INITIALS/*",
                "arn:aws:s3:::dataeng-clean-zone-INITIALS"
            ]
        },
```

In the final statement of the policy, we provide permissions to use all AWS Glue actions (create job, start job, and delete job). Note that in a production environment, you should limit the scope specified in the resource section:

```
        {
            "Effect": "Allow",
            "Action": [
                "glue:*"
            ],
            "Resource": "*"
        }
    ]
}
```

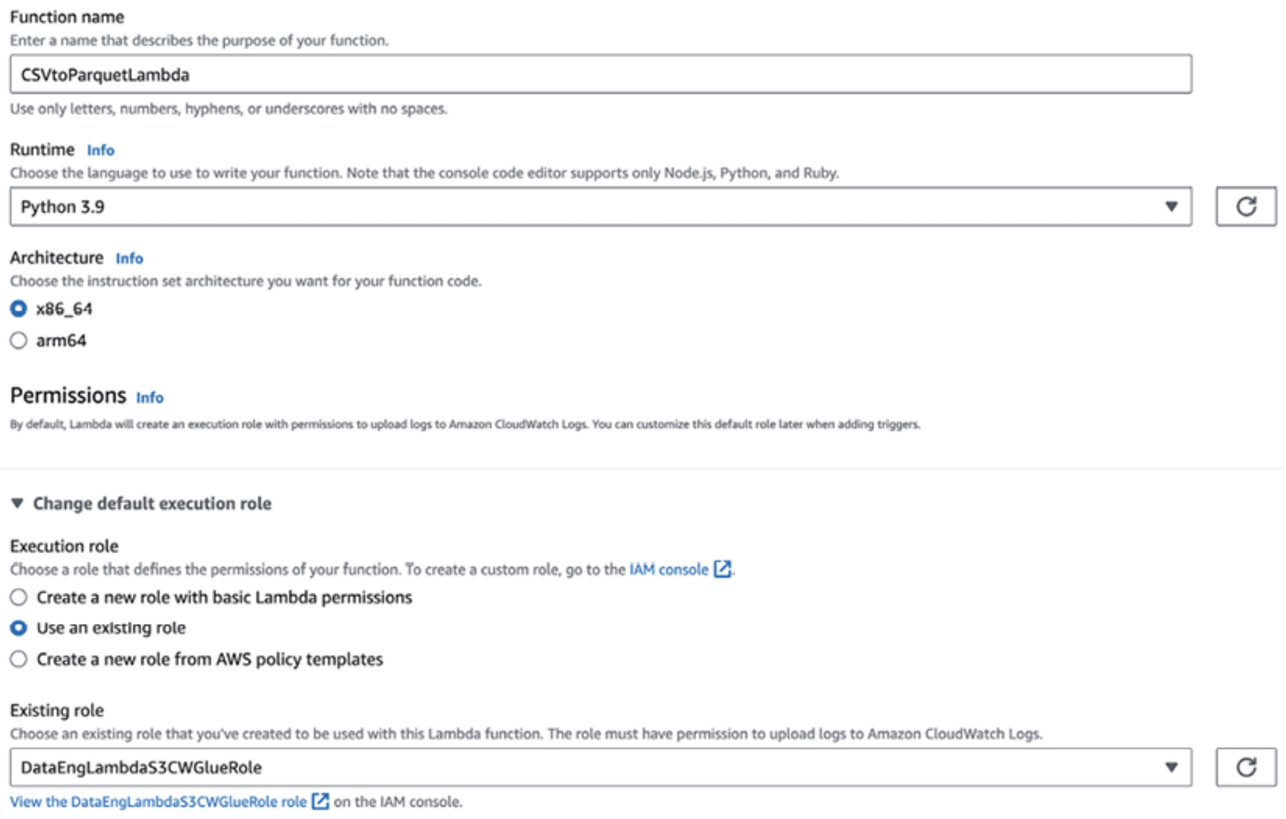
4. Click on Next Tags and then Next: Review.
5. Provide a name for the policy, such as `DataEngLambdaS3CWGluePolicy`, and then click Create policy.
6. In the left-hand menu, click on Roles and then Create role.
7. For the trusted entity, ensure AWS service is selected, and for the service, select Lambda and then click Next: Permissions. In step 4 of the next section (Creating a Lambda function), we will assign this role to our Lambda function.
8. Under Attach permissions, select the policy we just created (for example, `DataEngLambdaS3CWGluePolicy`) by searching and then clicking the tick box. Then, click Next.
9. Provide a role name, such as `DataEngLambdaS3CWGlueRole`, and click Create role.

Creating a Lambda function

We are now ready to create our Lambda function that will be triggered whenever a CSV file is uploaded to our source S3 bucket. The uploaded CSV file will be converted into Parquet, written out to the target bucket, and added to the Glue catalog using the AWS SDK for pandas library:

1. In the AWS console, search for and select the Lambda service, and in the left-hand menu, select Functions and then click Create function. Make sure you are in the same Region as where you created your AWS buckets in Chapter 2.
2. Select Author from scratch and provide a function name (such as `CSVtoParquetLambda`).

3. For Runtime, select Python 3.9 from the drop-down list.
4. Expand Change default execution role and select Use an existing role. From the drop-down list, select the role you created in the previous section (such as `DataEngLambdaS3CWGlueRole`):



Function name
Enter a name that describes the purpose of your function.
CSVtoParquetLambda
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

Architecture Info
Choose the instruction set architecture you want for your function code.
[✓ x86_64]
arm64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

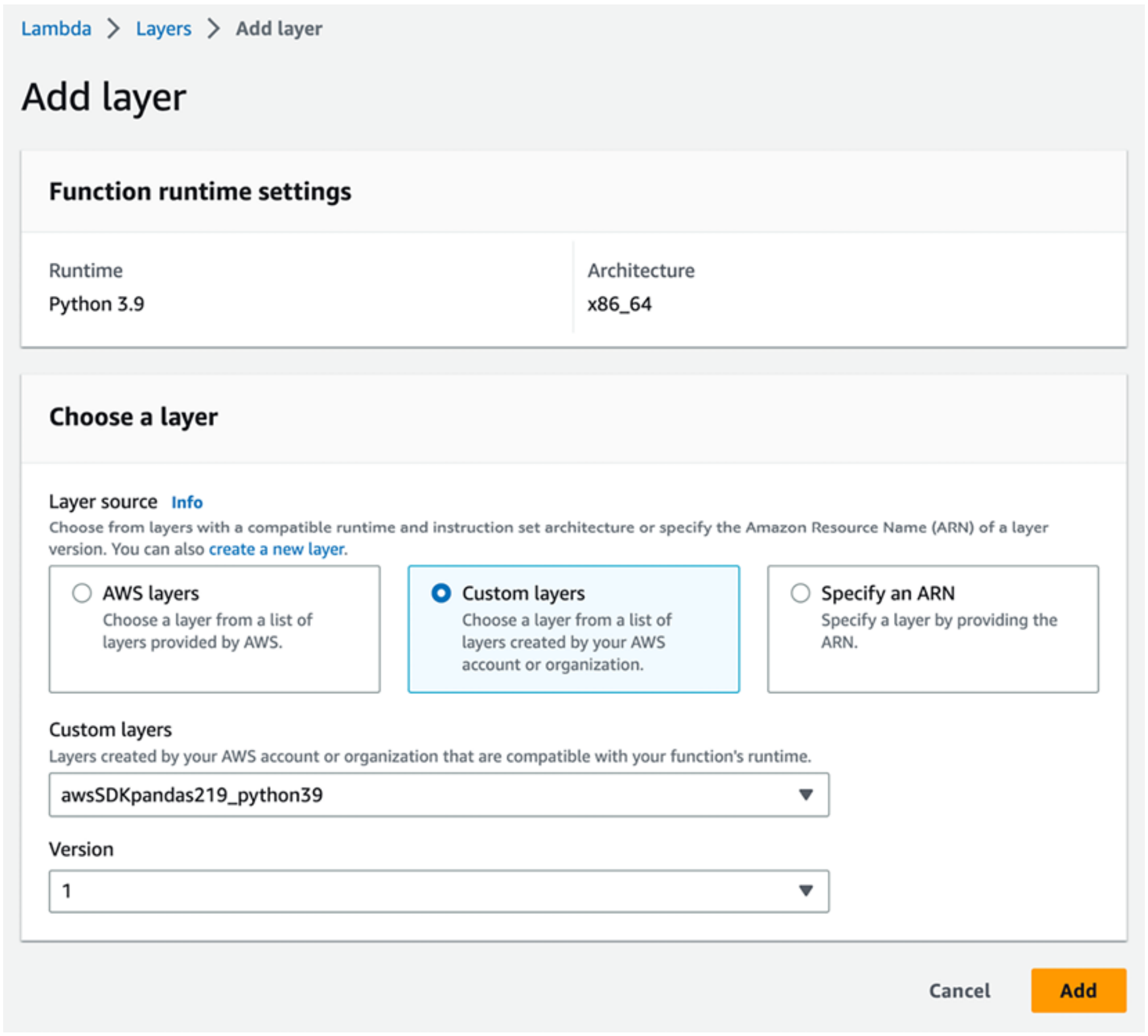
▼ Change default execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Create a new role with basic Lambda permissions
[✓ Use an existing role]
Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
DataEngLambdaS3CWGlueRole
View the DataEngLambdaS3CWGlueRole role on the IAM console.

Figure 3.11: Creating and configuring a Lambda function

5. Do not change any of the Advanced settings and click Create function.
6. Click on Layers in the first box (Function overview), and then click Add a layer in the Layers box.
7. Select Custom layers, and from the dropdown, select the AWS SDK for pandas layer you created in a previous step (such as `awsSDKpandas219_python39`). Select the latest version and then click Add.


Lambda > Layers > Add layer

Add layer

Function runtime settings

Runtime: Python 3.9

Architecture: x86_64

Choose a layer

Layer source Info

Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also create a new layer.

- AWS layers — Choose a layer from a list of layers provided by AWS.
- [✓ Custom layers] — Choose a layer from a list of layers created by your AWS account or organization.
- Specify an ARN — Specify a layer by providing the ARN.

Custom layers

Layers created by your AWS account or organization that are compatible with your function's runtime.

awsSDKpandas219_python39

Version

1

Figure 3.12: Adding an AWS Lambda layer to an AWS Lambda function

8. Scroll down to the Code Source section in the Lambda console. The following code can be downloaded from this book's GitHub repository. Make sure to replace any existing code in `lambda_function` with this code. In the first few lines of code, we import `boto3` (the AWS Python SDK), `awswrangler` (which is part of the AWS SDK for pandas library that we added as a Lambda layer), and a function from the `urllib` library called `unquote_plus`:

```
import boto3
import awswrangler as wr
from urllib.parse import unquote_plus
```

We then define our main function, `lambda_handler`, which is called when the Lambda function is executed. The `event` data contains information such as the S3 object that was uploaded and was the cause of the trigger that ran this function. From this event data, we get the S3 bucket name and the object key. We also set the Glue catalog `db_name` and `table_name` based on the path of the object that was uploaded (review the comments in the code below for an explanation of how this works).

```
def lambda_handler(event, context):
    # Get the source bucket and object name as passed to the Lambda function
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key'])

    # We will set the DB and table name based on the last two elements of
    # the path prior to the filename. If key = 'dms/sakila/film/LOAD01.csv',
    # then the following lines will set db to 'sakila' and table_name to
'film'
    key_list = key.split("/")
    print(f'key_list: {key_list}')
    db_name = key_list[len(key_list)-3]
    table_name = key_list[len(key_list)-2]
```

We now print out some debugging information that will be captured in our Lambda function logs. This includes information such as the Amazon S3 bucket and key that we are processing. We then set the `output_path` value here, which is where we are going to write the Parquet file that this function creates. Make sure to change the `output_path` value of this code to match the name of the target S3 bucket you created earlier:

```
    print(f'Bucket: {bucket}')
    print(f'Key: {key}')
    print(f'DB Name: {db_name}')
    print(f'Table Name: {table_name}')

    input_path = f"s3://{bucket}/{key}"
    print(f'Input_Path: {input_path}')
    output_path = f"s3://dataeng-clean-zone-INITIALS/{db_name}/{table_name}"
    print(f'Output_Path: {output_path}')
```

We can then use the AWS SDK for pandas library (defined as `wr` in our function) to read the CSV file that we received. We read the contents of the CSV file into a `pandas` DataFrame we are calling `input_df`. We also get a list of current Glue databases, and if the database we want to use does not exist, we create it:

```
    input_df = wr.s3.read_csv([input_path])

    current_databases = wr.catalog.databases()
    wr.catalog.databases()
    if db_name not in current_databases.values:
        print(f'- Database {db_name} does not exist ... creating')
        wr.catalog.create_database(db_name)
    else:
        print(f'- Database {db_name} already exists')
```

Finally, we can use the AWS SDK for pandas library to create a Parquet file containing the data we read from the CSV file. For the S3 to Parquet function, we specify the name of the DataFrame (`input_df`) that contains the data we want to write out in Parquet format. We also specify the S3 output path, the Glue database, and the table name:

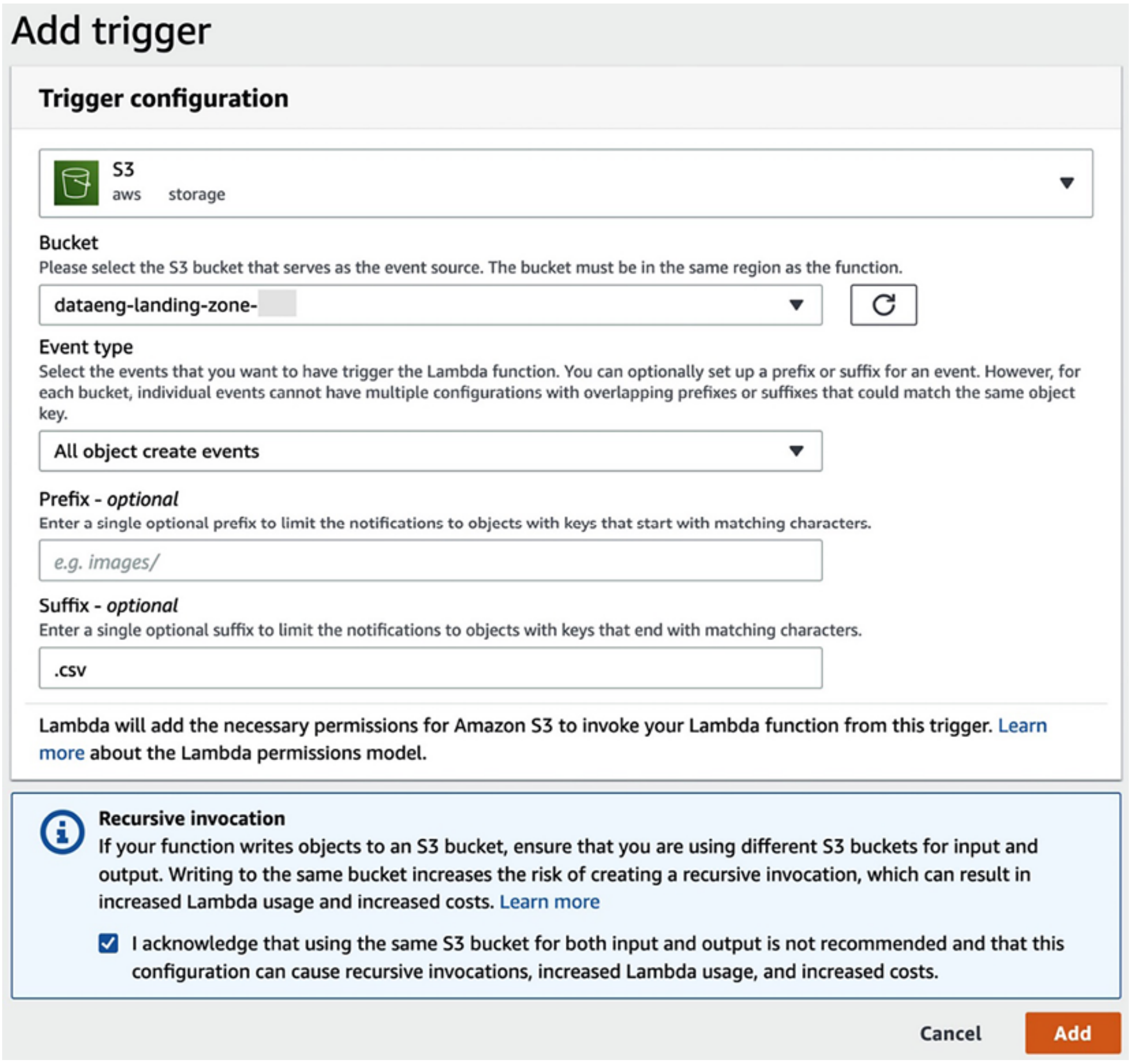
```
    result = wr.s3.to_parquet(
        df=input_df,
        path=output_path,
        dataset=True,
        database=db_name,
        table=table_name,
        mode="append")
    print("RESULT: ")
    print(f'{result}')
    return result
```

9. Click on Deploy at the top of the Code Source window
10. Click on the Configuration tab (above the Code Source window), and on the left-hand side, click on General configuration. Click the Edit button and modify the Timeout to be 1 minute (the default timeout of 3 seconds is likely to be too low to convert some files from CSV into Parquet format). Then, click on Save. If you skip this step, you are likely to get an error when your function runs.

Configuring our Lambda function to be triggered by an S3 upload

Our final task is to configure the Lambda function so that whenever a CSV file is uploaded to our landing zone bucket, the Lambda function runs and converts the file into Parquet format:

1. In the Function Overview box of our Lambda function, click on Add trigger.
2. For Trigger configuration, select the S3 service from the drop-down list.
3. For Bucket, select your landing zone bucket (for example, `dataeng-landing-zone-<initials>`).
4. We want our rule to trigger whenever a new file is created in this bucket, no matter what method is used to create it (Put, Post, or Copy), so select All object create events from the list.
5. For suffix, enter `.csv`. This will configure the trigger to only run the Lambda function when a file with a `.csv` extension is uploaded to our landing zone bucket.
6. Acknowledge the warning about Recursive invocation, which can crop up if you set up a trigger on a specific bucket to run a Lambda function and then you get your Lambda function to create a new file in that same bucket and path. This is a good time to double-check and make sure that you are configuring this trigger in the `LANDING ZONE` bucket (for example, `dataeng-landing-zone-<initials>`) and not the target `CLEAN ZONE` bucket that our Lambda function will write to:



Add trigger

Trigger configuration

S3
aws storage

Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.

dataeng-landing-zone-

Event type
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

All object create events

Prefix - *optional*
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.

e.g. images/

Suffix - *optional*
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.

.csv

Lambda will add the necessary permissions for Amazon S3 to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. Learn more

✓ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

Figure 3.13: Configuring an S3-based trigger for an AWS Lambda function

7. Click Add to create the trigger.
8. Create a simple CSV file called `test.csv` that you can use to test the trigger or download `test.csv` from the GitHub site for this chapter (https://github.com/PacktPublishing/Data-Engineering-with-AWS-2nd-edition/blob/main/Chapter03/test.csv). Ensure that the first line has column headings, as per the following example:

```
name,favorite_num
Gareth,23
Tracy,28
Chris,16
Emma,14
```

Ensure you create the file with a standard text editor, and not Word processing software (such as Microsoft Word) or any other software that will add additional formatting to the file.

9. Navigate to the Amazon S3 console (https://s3.console.aws.amazon.com/s3) and click on the `dataeng-landing-zone-initials` bucket that you previously created. Then click on Create folder and provide a folder name of `cleanzonedb`. The top-level folder we create here is going to be used as the name of the database that will be created in Glue Data Catalog to store our new table.
10. Navigate into the `cleanzonedb` folder, and create a second-level folder (this will be used as the name of the table that gets created in Glue Data Catalog). Name the second-level folder `csvtoparquet`.
11. Navigate into the `csvtoparquet` folder, click on Upload, and then upload the `test.csv` file you previously created. The file should be uploaded into `dataeng-landing-zone-initials/cleanzonedb/csvtoparquet`.
12. If everything has been configured correctly, your Lambda function will have been triggered and will have written out a Parquet-formatted file to your target S3 bucket and created a Glue database and table. You can access the Glue service in the AWS Management Console to ensure that a new database and table have been created in the data catalog and can run the following command in CloudShell to ensure that a Parquet file has been written to your target bucket. Make sure to replace `dataeng-clean-zone-initials` with the name of your target S3 bucket:

```
aws s3 ls s3://dataeng-clean-zone-initials/cleanzonedb/csvtoparquet/
```

The result of this command should display the Parquet file that was created by the Lambda function.