Introduction to AWS Cloud Development Kit with Python

You are confident your company can now use AWS CloudFormation and Infrastructure-as-Code techniques to improve their developmenmt processes and environments. The next task is to explain the topic of automating infrastructure using code to a Senior Engineer in the organization. She is an experienced Python developer, but has never been a fan of other languges like YAML. She is also unfamiliar with the latest capabilities available on cloud platforms.

After some additional research you discovered the AWS Cloud Development Kit which supports the Python programming language. This presents a great opportunity to demonstrate those capabilities during the discussion.

NOTE: This workflow will require the installation of additional software in your VSCode environment.

Introduction to CDK

The AWS Cloud Development Kit (CDK) is a software development framework for deploying cloud infrastructure using a variety of programming languages. CDK's flexibility allows you to leverage the full power of programming constructs and libraries that you understand and are familiar with to define and manage your infrastructure. In this workstream, we will use Python with the AWS CDK to define and deploy an AWS resource.

Prerequisites

- AWS account with the proper access permissions for resource deployment
- AWS Cloud Development Kit
- AWS command line interface (CLI)

Installing the CDK

From the command line inside the VSCode session, install the AWS CDK using `npm`:

```
sudo npm install -g aws-cdk
```

Verify the installation of the AWS CDK:

```
cdk --version
```

You should be running cdk version `2.150.0` or higher

Create a new CDK Project

From the `/Workshop` directory, create a new sub-directory as the parent folder for your project and navigate into it:

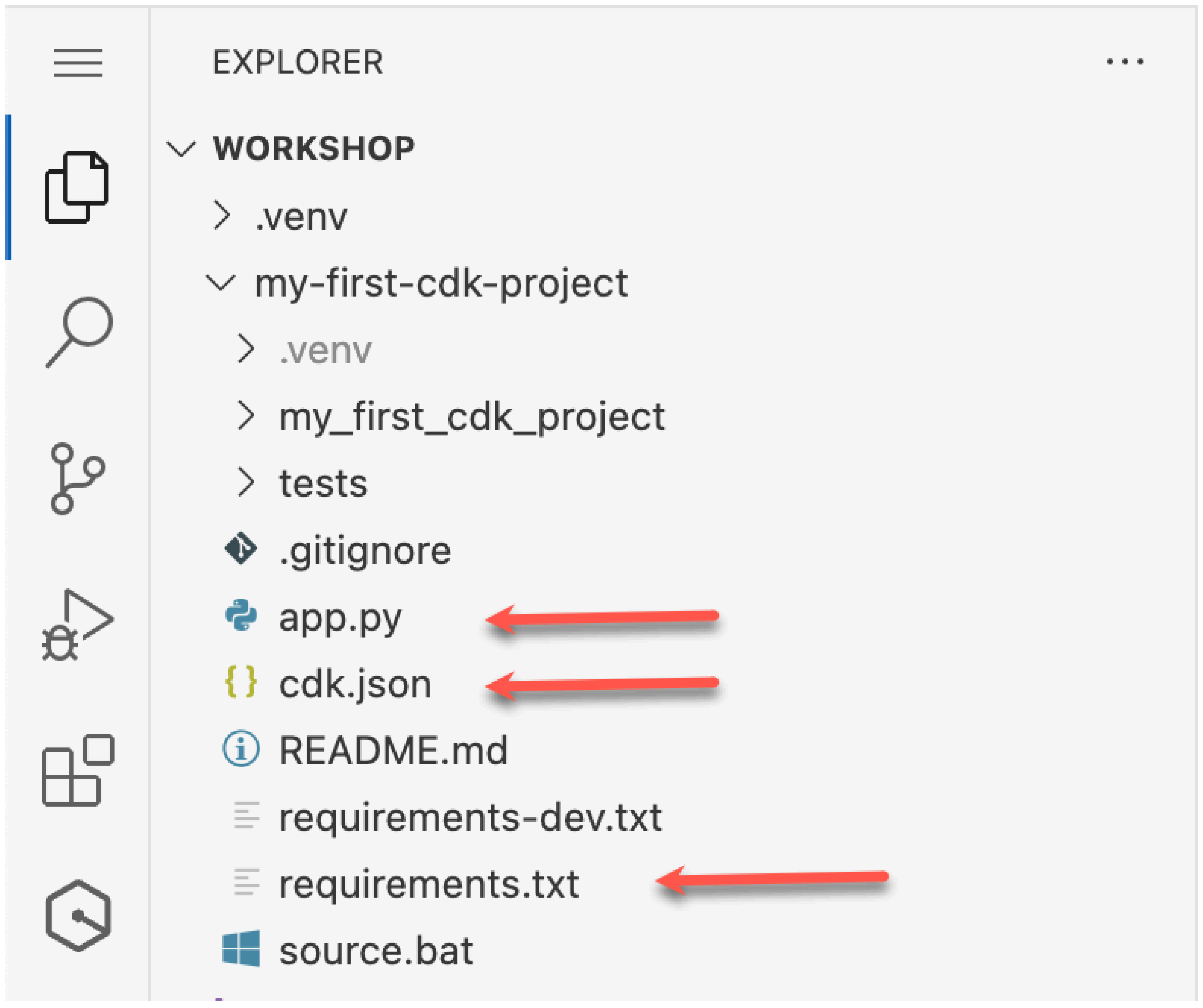
```
mkdir my-first-cdk-project
cd my-first-cdk-project
```

Initialize a new CDK project. For the project, specify that you will use Python as the preferred language for building the code infrastructure:

```
cdk init app --language python
```

Once executed, the command will create several files and a directory structure for the project. The resources of note are:

- `app.py`: Entry point of your CDK application.
- `requirements.txt`: File containing a list of dependencies for your project.
- `cdk.json`: Configuration file for the CDK project.



We will now create and activate a virtual environment containing the necessary modules for the Python language. Virtual environments create a level of isolation that provides more control and freedom from libraries and modules on the host system which may interfere with development:

```
python3 -m venv .venv
source .venv/bin/activate
```

Once the environment is sourced, the terminal will contain the phrase **(.venv)** at the beginning of the command prompt.

Proceed with the next step by installing the necessary library dependencies for the project. We noted earlier that the `requirements.txt` file contains a list of libraries, so we will add it as an argument:

```
pip install -r requirements.txt
```

Create a S3 bucket using CDK

We are now ready to create another S3 bucket resource. The difference in this workflow is instead of using a CloudFormation template, we will use the CDK.

From the project's parent folder, move to the `my_first_cdk_project` sub-folder. By default, the CDK creates the folder name by converting the parent folder name using snake case with the "_" character:

```
cd my_first_cdk_project
```

Edit the `my_first_cdk_project_stack.py` file by replacing the existing code with the following:

```
 from aws_cdk import (
     Stack,
     aws_s3 as s3
 )
 from constructs import Construct

 class MyFirstCdkProjectStack(Stack):

     def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
         super().__init__(scope, construct_id, **kwargs)

         s3.Bucket(
             self,
             id="bucket123",
             bucket_name="<our-first-cdk-s3-bucket>" # Use a globally unique bucket name
         )
```

Let's briefly analyze this code.

Import Statement

```
from aws_cdk import (
    Stack,
    aws_s3 as s3
)
```

- **Stack**: This imports the core module from the AWS CDK library, which contains essential classes and methods for defining CDK applications and stacks.
- **aws_s3 as s3**: This imports the AWS S3 module from the AWS CDK library and assigns it the alias s3. This module contains classes and methods for working with S3 resources.

Stack Definition

```
class MyFirstCdkProjectStack(Stack):
```

- **class MyFirstCdkProjectStack(Stack)**: This defines a new class `MyFirstCdkProjectStack` that inherits from Stack.

In AWS CDK, a stack is a unit of deployment that contains AWS resources. By inheriting from `Stack`, this class becomes a CDK stack.

Constructor Method

```
def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)
```

This is the **constructor method** for the MyfirstCdkProjectStack class. It initializes the stack taking as parameters the scope (the construct tree node that is the parent of this stack) and the id (unique identifier for the stack).

Define an S3 Bucket

```
s3.Bucket(
            self,
            id="bucket123",
            versioned=True,
            encryption=s3.BucketEncryption.S3_MANAGED,
            bucket_name="<our-first-cdk-s3-bucket>" # Use a globally unique bucket name
        )
```

This part of the code is most similar to the CloudFormation template the we defined previously. We are creating an S3 Bucket with object versioning and server-side encryption enabled.

Review the file for accuracy and completeness and then save the file. Once saved, move back to the parent directory:

```
cd ../
```

Run the next command to display the list of stacks configured in your environment. It should only return the single project we have created. Returning the project list will also confirm that there are no syntactical errors within the code.

```
cdk ls
```

We are now ready to deploy!

Deploy the CDK Stack

CDK uses CloudFormation under the hood to manage the deployment of resources. CDK takes care of creating a YAML template from the Python configuration. This proces is called **Synth**.

If your app contains more than one stack, you must specify which stacks to synthesize. Since our app contains a single stack, the CDK CLI automatically detects the stack to synthesize. If you don't run **cdk synth**, the CDK CLI will automatically perform this step when you deploy. However, we recommend that you run this step before each deployment.

```
cdk synth
```

The cdk synth command runs your app. This creates an AWS CloudFormation template for each stack in your app. The CDK CLI will display a YAML formatted version of your template at the command line and save a JSON formatted version of your template in the cdk.out directory.

You must perform a one-time **bootstrapping** of your AWS environment before deployment. Run the command:

```
cdk bootstrap
```

Then, proceed to deploy the stack:

```
cdk deploy
```

You should see output indicating that the stack is being deployed. Once complete, the new S3 bucket will be created in your AWS account.

Conclusion of workflow

What did you do in this workflow?
While this workflow was short, it is important to know the basics of CDK to manage our infrastructures.
You praticed commands like `cdk synth`, `cdk bootstrap` and `cdk deploy`.

Previous Next