Midterm Report – Spring 2024
Topic: CDK for Terraform
Mingming Liu

I. What is CDK for Terraform.

The CDK for Terraform (Cloud Development Kit for Terraform) is an infrastructure as code tool that allows users to define cloud infrastructure using familiar programming languages, instead of using the domain-specific language (DSL) of Terraform itself (HashiCorp Configuration Language). This approach aims to make it easier for developers who are already familiar with programming languages like TypeScript, Python, Java, and C# to define, configure, and provision cloud resources in a more intuitive and efficient way.

II. Tutorial 0: AWS environment setup

1. AWS setup.
    - Create one cloud9 environment and open the cloud9 IDE.
    - In Cloud9 IDE, you will see a folder which is name after the name you given to cloud9 IDE when you set the IDE up.
    - You may need to change the Cloud9 Access: AWSCloud9SSMAccessRole – go to IAM to find this role and add permission: AmazonS3FullAccess, AmazonEC2FullAccess, IAMFullAccess.
    - Bring up the terminal
2. Install CDK for Terraform on Amazon Linux and initialize CDK for Terraform Project
    - sudo yum update -y
    - npm install -g cdktf-cli@latest
    - pip install --user pipenv
    - npm install --global yarn
    - sudo yum install -y yum-utils
    - sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
    - sudo yum -y install terraform
3. Run the following command to make sure every package needed has been installed.
    - terraform -version
    - node --version
    - cdktf --version
    - node --version
    - yarn --version
4. Initiate CDK for Terraform project
    - Go to the terminal and create a new directory, enter the directory.
    - Run the command line to initialize the project:
        - cdktf init --template=python --providers="aws@~>4.0" --local

- After the initialization, we will see in folder we just created, aws has already created some other directories in the folder, and there is a file named main.py which is the place where we will go to add our code for the static website using cdk for terraform.

III. Tutorial 1: Static Website on S3.

1. Create another folder inside the main folder we just created, and upload the unzipped zip file static-website.zip into the newly created fold.

2. Write Terraform Configuration

1) import required modules and classes.

- First, import the necessary Python modules and classes from the CDKTF and AWS CDKTF providers.

2) Define the Stack Class

- When we initiate the cdk for terraform project in the folder we created, the whole structure of the stack class has been automatically created in the main.py file.

3) Initialize AWS Provider

- Specify the AWS provider with the desired region.

```
AwsProvider(self, 'AWS', region='us-east-1')
```

4) Create an S3 Bucket for Website Hosting

- Instantiate an S3Bucket with properties such as bucket name and force destruction policy. This will be your website's storage.

```
s3_bucket = S3Bucket(self, "MyWebsiteBucket",
    bucket = "my-unique-bucket-name-for-website",
    force_destroy = True
)
```

5) Configure Website Properties for the S3 Bucket

- Set up the S3 bucket for website hosting by specifying the index document.

```
S3BucketWebsiteConfiguration(self, "MyWebsiteConfiguration",
    bucket = s3_bucket.id,
    index_document = S3BucketWebsiteConfigurationIndexDocument(suffix = "index.html")
)
```

6) Enable Versioning

- Enable versioning on the S3 bucket to keep track of and manage previous versions of objects.

```
S3BucketVersioningA(self, "MyWebsiteVersioning",
    bucket = s3_bucket.id,
    versioning_configuration = S3BucketVersioningVersioningConfiguration(
            status = "Enabled")
```

7) Configure Public Access Settings

- Control public access to the S3 bucket by blocking or allowing certain types of access.

```
S3BucketPublicAccessBlock(self, "MyWebsiteBucketPublicAccessBlock",
    bucket = s3_bucket.id,
    block_public_acls = True,
```

```
    ignore_public_acls = True,
    block_public_policy = False,
    restrict_public_buckets = False,
)
```

8) Set Ownership Controls

- Specify the ownership control rules for the objects in the bucket.

```
S3BucketOwnershipControls(self, "MyWebsiteBucketOwnershipControls",
    bucket = s3_bucket.id,
    rule = S3BucketOwnershipControlsRule(object_ownership = "BucketOwnerPreferred")
```

9) Initialize AWS Provider

- Apply an Access Control List (ACL)

```
s3_acl = S3BucketAcl(self, "MyWebsiteBucketAcl",
    bucket = s3_bucket.id,
    acl = "private",
)
```

10) Define a Bucket Policy

- Create a bucket policy to manage access to the bucket. This example allows public read access to the objects.

```
s3_policy = S3BucketPolicy(self, "MyWebsiteBucketPolicy",
    bucket = s3_bucket.id,
    policy = json.dumps({
        "Version": "2012-10-17",
        "Statement": [{
            "Action": "s3:GetObject",
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::my-unique-bucket-name-for-website/*" ],
            "Principal": "*"
        }]
    })
```

11) Upload Objects to the S3 Bucket

- This section of the code automatically uploads files from a specified local directory (static-website) to your S3 bucket. For each file in the directory, it sets the appropriate MIME type, which is essential for the web browser to render the files correctly.

```
directory_path = os.path.abspath("static-website")
for root, dirs, files in os.walk(directory_path):
    for file in files:
        file_path = os.path.join(root, file)
        s3_path = os.path.relpath(file_path, start=directory_path)
        content_type, _ = mimetypes.guess_type(file_path)
        S3Object(self, s3_path,
            bucket = s3_bucket.id,
            depends_on = [s3_policy],
            key = s3_path,
            source = file_path,
```

```
    content_type = content_type,
)
```

- In this loop:
    - os.walk(directory_path) is used to iterate through all the files in the static-website directory and its subdirectories.
    - For each file, os.path.join(root, file) gets the full file path.
    - os.path.relpath(file_path, start=directory_path) determines the file's path relative to the base directory, which will be used as the key in S3.
    - mimetypes.guess_type(file_path) guesses the MIME type of the file based on its extension, which is necessary for correctly serving the file over the web.
    - S3Object is then used to create an object in S3 for each file. The depends_on parameter ensures that the S3 object is created after the bucket policy is applied.

12) Initialize and Synthesize the Application

- Finally, create an instance of the App class, add your stack to the app, and synthesize the application, which compiles your CDKTF code into deployable Terraform JSON.
- The code below will be generated when you initialize the CDK for terraform project.

```
app = App()
MyStack(app, "MyStaticWebsite")
app.synth()
```

3. To configure and deploy your infrastructure using the CDK for Terraform, begin by executing the command cdktf synth in your terminal. This step synthesizes your CDKTF configuration into Terraform-compatible JSON files. Following synthesis, initiate the deployment process by running cdktf deploy. This action triggers the creation of the defined resources, and you should observe the provisioning of approximately 17 resources within your terminal output, indicating successful deployment.

Upon completion, navigate to the Amazon S3 console and access the newly created bucket, identified by the name "my-unique-bucket-name-for-website". Within this bucket, you will find the files that were uploaded as part of the static website setup process. To view your static website, you have two primary methods:

- Direct Interaction: Click on the index.html file located within the S3 bucket. This action will open your static website directly in your web browser, allowing you to interact with it immediately.
- S3 Bucket Properties: Alternatively, you can explore the S3 bucket's properties by scrolling to the "Static Website Hosting" section found at the bottom of the properties page. Here, a link is provided which, when accessed, will navigate your browser to your static website.

These steps offer a streamlined approach to deploying and viewing your static website hosted on AWS S3, leveraging the powerful and flexible capabilities of the CDK for Terraform.

4. To avoid unnecessary charges, remember to run “cdktf destroy” when you no longer need these resources. This command will tear down everything created by “cdktf deploy”.

IV. Tutorial 2: Website on EC2 instance

1. After initialized the CDFTF project in the folder newly created, we will start to write CDK for Terraform Configuration for the website on EC2 instance.

1) Prepare the Configuration Script

Create a bash script named configure.sh in the same directory as your Python file. This script will install and configure your web server on the EC2 instance. Ensure it is executable and correctly sets up the web server to serve your website's files.

2) The provided main.py script does the following (see the python file for detail for each of the code setup):

- Initializes a new CDKTF app and stack.
- Sets up a new VPC and a public subnet for internet access.
- Creates an Internet Gateway and associates it with a route table for the subnet.
- Configures an IAM role for the EC2 instance to allow it access to S3 and other necessary AWS services.
- Creates an S3 bucket and uploads the configuration script as an object.
- Defines a security group allowing inbound traffic on port 80 (HTTP).
- Launches an EC2 instance in the public subnet with a user data script to download and execute the configuration script from S3 on startup.
- Uses an IAM instance profile to attach the IAM role to the EC2 instance.

3) Configure and Deploy

- cdktf synth: This command synthesizes your CDKTF code into Terraform configuration files.
- cdktf deploy: Deploy your infrastructure. You will see resources being created, including the VPC, subnet, EC2 instance, and S3 bucket.

4) Verify Deployment

Go to EC2 object we just created, click on the public IPv4 address. After a new browser window pops up, change the https to http. Then we will see a Test page, which indicates we have set up everything correctly and successfully.

2. To avoid unnecessary charges, remember to run “cdktf destroy” when you no longer need these resources. This command will tear down everything created by “cdktf deploy”.

V. What is different from our classroom lab?

Creating a static website using the CDK for Terraform (CDKTF) versus using the AWS Cloud Development Kit (CDK) presents a choice between broad provider compatibility and AWS-specific optimization. CDKTF, leveraging Terraform under the hood, allows for infrastructure definition across various cloud providers, making it versatile for multi-cloud scenarios. It synthesizes Terraform configurations from code, requiring subsequent deployment with Terraform commands. This approach offers a wide tooling ecosystem but introduces an extra step in the deployment process and demands familiarity with both CDKTF and Terraform.

On the other hand, the AWS CDK is tailored for AWS services, directly generating CloudFormation templates from code for deployment. It provides a streamlined workflow for AWS users, with direct deployment capabilities via the CDK CLI and deep integration with AWS services. This makes the AWS CDK particularly efficient for projects fully housed within AWS but less flexible for multi-cloud deployments. The choice between CDKTF and AWS CDK largely depends on the specific requirements of your project, your preferred cloud services, and your familiarity with the underlying technologies (Terraform for CDKTF, CloudFormation for AWS CDK).

And I personally don't like CDKTF, as this is very new IAC, and it is hard to find documentation for the most recent version. And it is more difficult as using AWS CDK.

VI. What's the benefit of using CDK for Terraform?

The CDK for Terraform (CDKTF) introduces a set of compelling benefits, especially when considered in the context of CloudFormation versus Terraform, and the AWS Cloud Development Kit (CDK) versus CDKTF. These benefits stem from the unique features and capabilities of Terraform and CDKTF, as well as how they compare to AWS-specific tools like CloudFormation and the AWS CDK.

1. CloudFormation vs. Terraform
    - Provider Agnosticism: Terraform stands out for its provider-agnostic approach, allowing users to manage resources across multiple cloud providers and services. This contrasts with CloudFormation, which is AWS-specific. CDKTF inherits Terraform's broad support, enabling developers to define infrastructure across various clouds in a single, unified codebase, enhancing flexibility and portability.
    - State Management and Planning: Terraform's state management and planning capabilities are robust, offering detailed insights into infrastructure changes before they are applied. While CloudFormation also provides change sets, Terraform's planning phase is often regarded as more comprehensive and user-friendly, a feature that CDKTF leverages to its advantage.
    - Community Modules and Resources: Terraform's ecosystem is rich with community-contributed modules and providers, allowing users to incorporate a wide range of pre-built infrastructure configurations into their projects. CDKTF benefits from this ecosystem, providing access to a vast array of resources that can be used alongside custom constructs.
2. CDK (for CloudFormation) vs. CDK for Terraform
    - Familiarity for Developers: Both the AWS CDK and CDKTF aim to make infrastructure as code more accessible to developers by allowing them to use familiar programming languages (e.g., TypeScript, Python). However, CDKTF offers the added benefit of leveraging Terraform's ecosystem, which might be more familiar or appealing to users who have experience with Terraform or wish to deploy infrastructure across multiple clouds.
    - Reusability and Composition: The AWS CDK is designed to improve reusability and composition within the AWS ecosystem, enabling developers to define high-level constructs that encapsulate AWS resources. CDKTF extends this concept beyond AWS, allowing developers to create reusable constructs that can be applied across different cloud providers, thanks to Terraform's provider-agnostic nature.

- Deployment and Tooling: While the AWS CDK directly integrates with AWS services and CloudFormation for deployment, CDKTF synthesizes Terraform configuration files that can be deployed using Terraform's CLI. This means CDKTF users can take advantage of Terraform's workflow and tooling, including its powerful CLI and state management features, regardless of the cloud provider.

In summary, CDKTF brings together the best of both worlds: the flexibility and provider-agnostic capabilities of Terraform and the developer-friendly abstraction of infrastructure as code offered by the CDK framework. This combination enables more efficient development practices, broader infrastructure management capabilities across multiple cloud environments, and access to a rich ecosystem of resources.

VII. Reference.

- https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/aws-cdk-for-terraform.html
- https://developer.hashicorp.com/terraform/cdktf
- https://developer.hashicorp.com/terraform/tutorials/cdktf/cdktf-build
- https://spacelift.io/blog/terraform-cdk

VIII. Appendix.

- Here is the link to the git hub repository:

https://github.com/mingminggitthubaccount/midterm/settings/access?query=mliu4195%40gmail.com

- And professor's account rclc has been sent an invitation to become the collaborators to this repository.