Gugatharsan Sivalingam Follow
Aug 20, 2022 · 6 min read · · Listen

Save

Provisioning the infrastructure using CDK for Terraform

Image from Terraform Website

What is CDK for Terraform(CDKTF)?

CDKTF is Cloud Development Kit for Terraform which allows us to use a familiar programming language to provision infrastructure. Using this we can access the entire terraform ecosystem without learning HashiCorp Configuration Language (HCL). AWS has a similar CDK called AWS CDK which leverages CloudFormation templates (YAML based) behind the scenes. Kubernetes also have another one called CDK8s. These days CDK is becoming a hot topic, I suggest you grasp knowledge about the CDKs depending on your use cases.

CDKTF-supported languages are as follows:

· **Typescript**

· **Python**

· **Java**

· **C#**

· **GO**

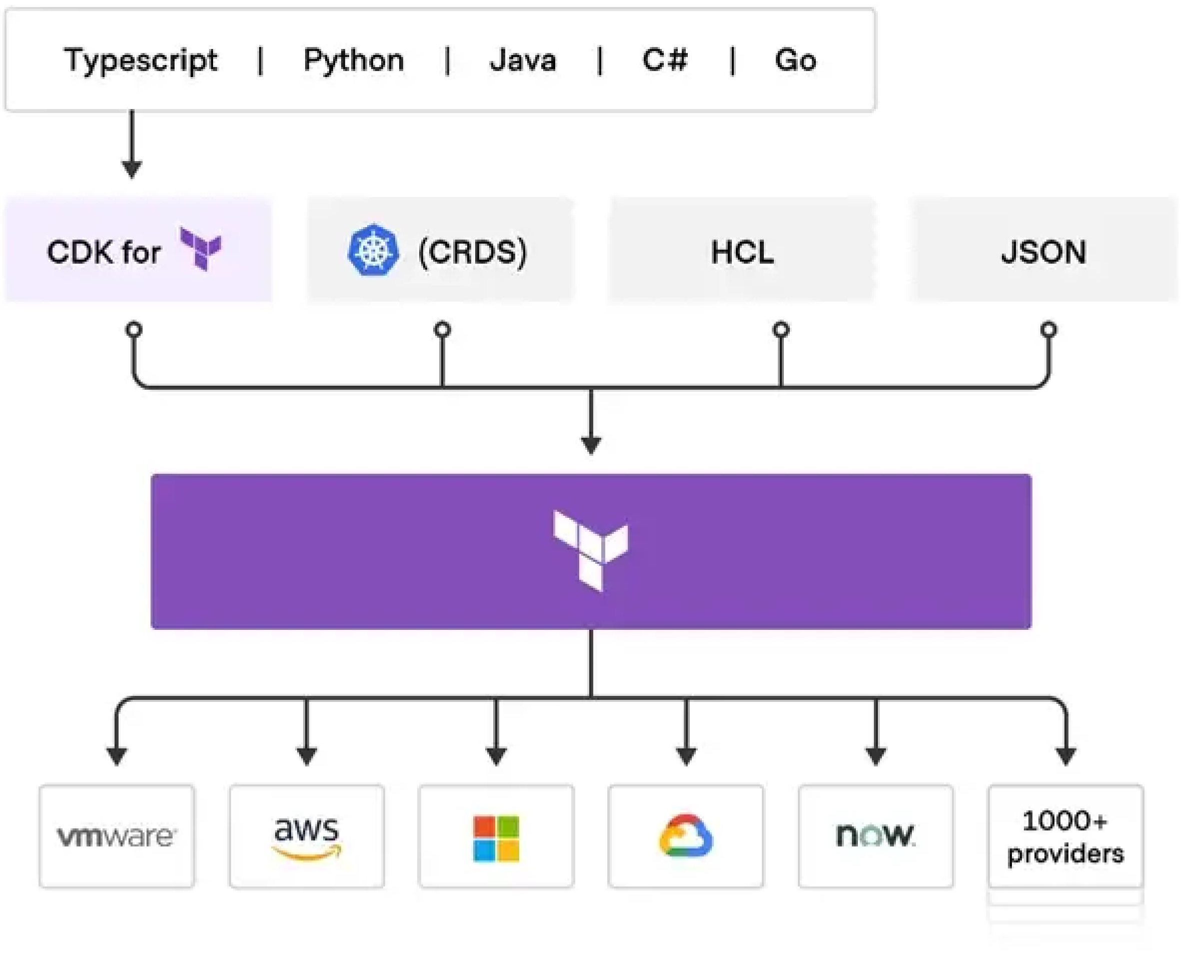


Image from CDK for Terraform Website

How does CDK for Terraform work?

CDK for Terraform leverages concepts and libraries from the AWS CDK to translate your code into infrastructure configuration files for Terraform. First, you need to create an application using either a built-in or custom template in your preferred language. Then you need to define your infrastructure using your preferred providers (AWS, Azure, Google Cloud). CDKTF automatically extracts the schema from terraform providers and modules to generate the necessary classes for your application. Once, everything is set you can deploy your infrastructure using CDKTF CLI commands.

What are Constructs?

Before jumping into the CDKTF let's check out the basics of CDK. Constructs are the basic building blocks of CDK apps. A construct represents a "cloud component" and encapsulates everything AWS CloudFormation/Terraform needs to create the component. Constructs are part of the Construct Programming Model (CPM) and are also used by other tools such as CDK for Terraform (CDKTF), and CDK for Kubernetes (CDK8s). I would recommend you to go through the

Construct Hub

Construct Hub helps developers find open-source construct libraries for use with AWS CDK, CDK8s, CDKTF and other...

constructs.dev

which consists of open-source Cloud Development Kit libraries.

Prerequisites to setup CDKTF project in your local machine

> *Terraform CLI (1.1+)*

Downloads | Terraform by HashiCorp

Terraform is an open-source infrastructure as code software tool that enables you to safely and predictably create...

www.terraform.io

> *Node (LTS version recomended)*

Download | Node.js

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

nodejs.org

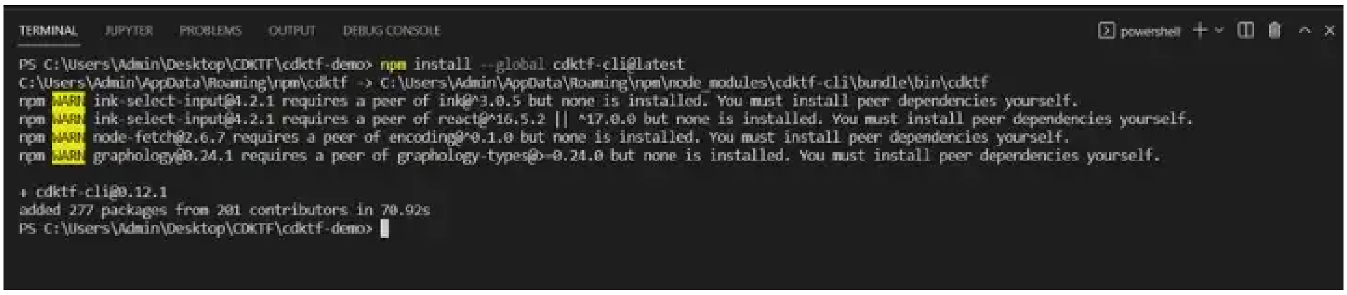
> *AWS CLI configured*

Installing or updating the latest version of the AWS CLI

This topic describes how to install or update the latest release of the AWS Command Line Interface (AWS CLI) on...

docs.aws.amazon.com

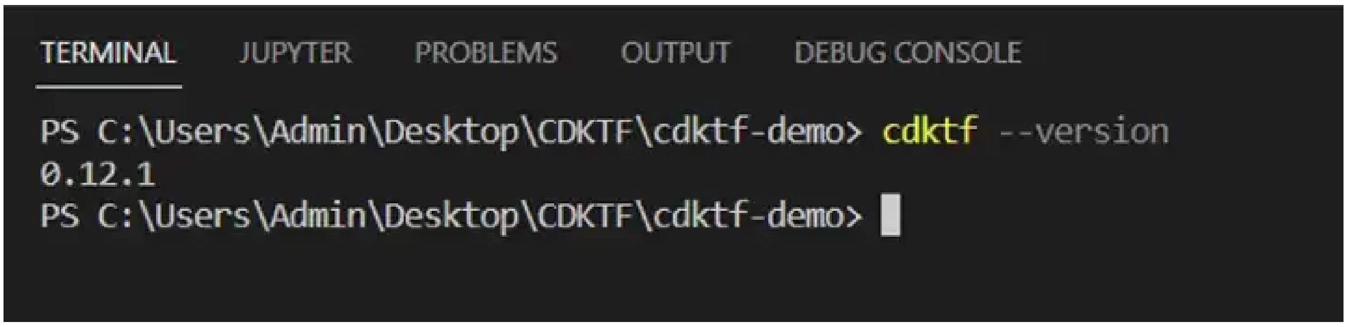
> *Install CDKTF in the CMD (npm install — global cdktf-cli@latest)*



```
PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo> npm install --global cdktf-cli@latest
C:\Users\Admin\AppData\Roaming\npm\cdktf -> C:\Users\Admin\AppData\Roaming\npm\node_modules\cdktf-cli\bundle\bin\cdktf
npm WARN ink-select-input@4.2.1 requires a peer of ink@^3.0.5 but none is installed. You must install peer dependencies yourself.
npm WARN ink-select-input@4.2.1 requires a peer of react@^16.5.2 || ^17.0.0 but none is installed. You must install peer dependencies yourself.
npm WARN node-fetch@2.6.7 requires a peer of encoding@^0.1.0 but none is installed. You must install peer dependencies yourself.
npm WARN graphology@0.24.1 requires a peer of graphology-types@>=0.24.0 but none is installed. You must install peer dependencies yourself.

+ cdktf-cli@0.12.1
added 277 packages from 201 contributors in 70.92s
PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo>
```

Installing CDKTF-CLI

Confirm CDKTF is correctly installed using the command cdktf — version



```
PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo> cdktf --version
0.12.1
PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo>
```

Verifying cdktf installation

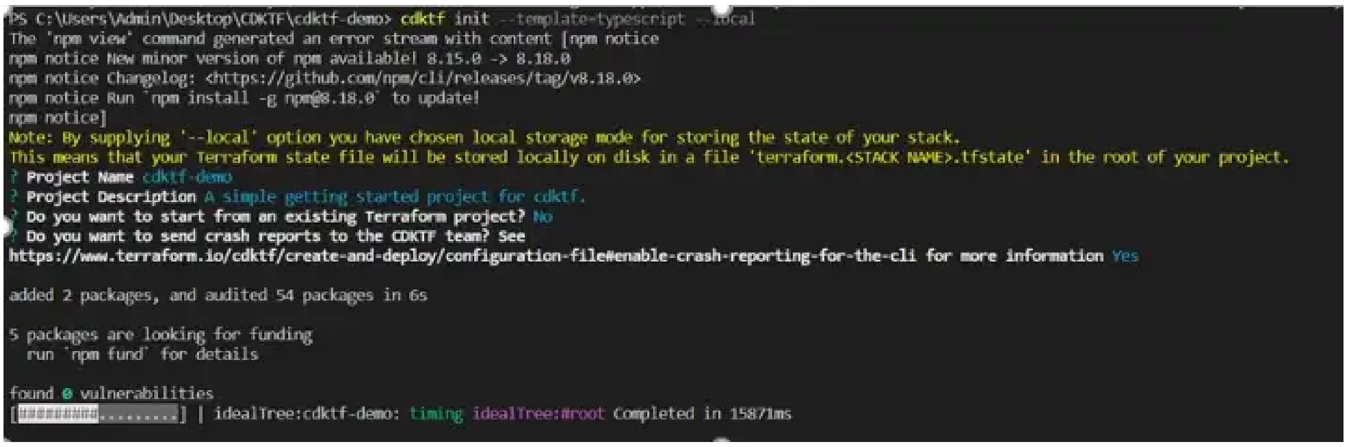
Let's initiate our first cdktf project using the following command:

```
cdktf init --template=typescript --local
```

cdktf hosted with ❤ by **GitHub** **view raw**

in the above command, we are initializing the cdktf project with typescript as our preferred language and we are storing the statefiles locally in our machine.

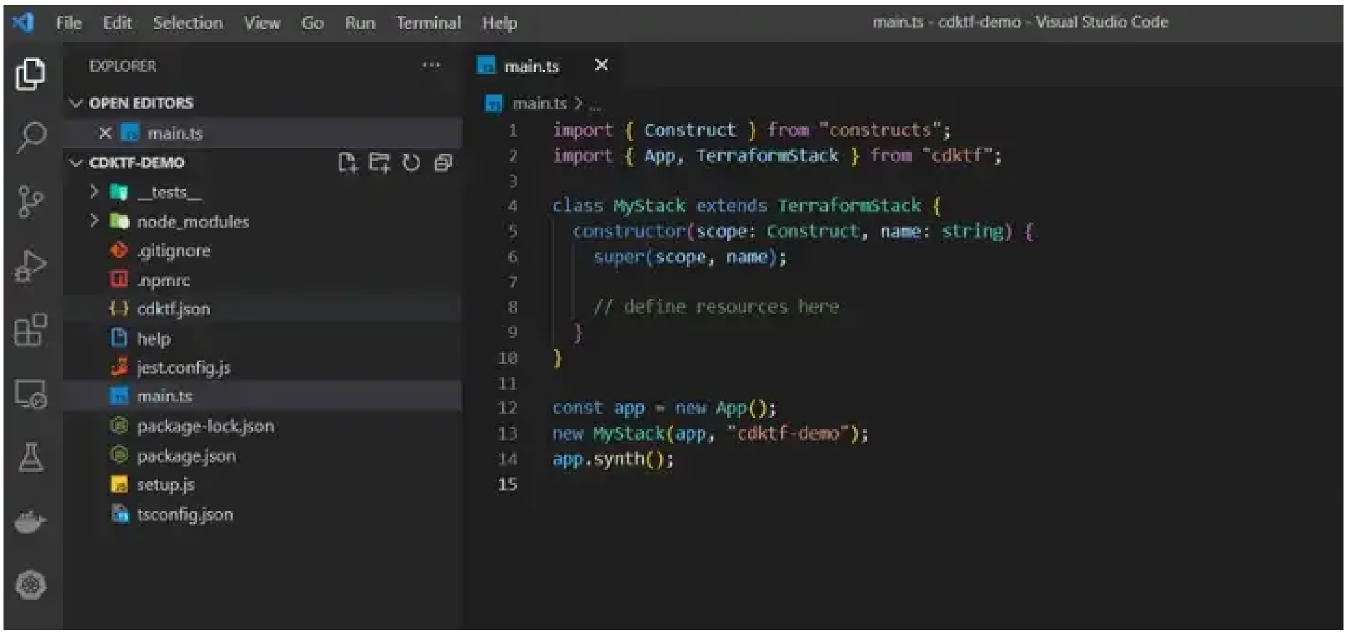
If you execute the above command, it'll prompt for the following and give the values you preferred.



```
PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo> cdktf init --template-typescript --local
The 'npm view' command generated an error stream with content [npm notice
npm notice New minor version of npm available! 8.15.0 -> 8.18.0
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v8.18.0>
npm notice Run `npm install -g npm@8.18.0` to update!
npm notice]
Note: By supplying '--local' option you have chosen local storage mode for storing the state of your stack.
This means that your Terraform state file will be stored locally on disk in a file 'terraform.<STACK NAME>.tfstate' in the root of your project.
? Project Name cdktf-demo
? Project Description A simple getting started project for cdktf.
? Do you want to start from an existing Terraform project? No
? Do you want to send crash reports to the CDKTF team? See
https://www.terraform.io/cdktf/create-and-deploy/configuration-file#enable-crash-reporting-for-the-cli for more information Yes

added 2 packages, and audited 54 packages in 6s

5 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
[#########.........] | idealTree:cdktf-demo: timing idealTree:#root Completed in 15871ms
```

CDKTF Prompt

If everything setup correctly, you can see the folder structure as below:



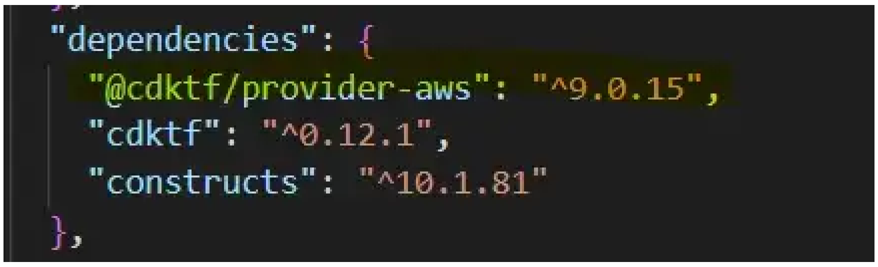
```
import { Construct } from "constructs";
import { App, TerraformStack } from "cdktf";

class MyStack extends TerraformStack {
  constructor(scope: Construct, name: string) {
    super(scope, name);

    // define resources here
  }
}

const app = new App();
new MyStack(app, "cdktf-demo");
app.synth();
```

Sample folders after initialize the CDKTF

For this demo, I'm going to deploy a simple EC2 instance in AWS. To use the AWS modules, we need to install the AWS provider for CDKTF.

```
npm install @cdktf/provider-aws@9.0.15
```

aws-provider hosted with ❤ by GitHub   view raw



```
"dependencies": {
  "@cdktf/provider-aws": "^9.0.15",
  "cdktf": "^0.12.1",
  "constructs": "^10.1.81"
},
```

package.json

If you install the provider correctly using npm package, you can see the provider under the dependencies section.

In the root level create a folder called types. then create a file called environment_config.ts and add the following code in there.

```
export interface EnvironmentConfig {
    environment: "DEV" | "STG" | "PRD",
    EC2Props: {
        ami: string;
        instanceType: string;
        tags: {
            Name: string,
            Environment: string,
            Managedby: string
        }
    }
}
```

environment_config.ts hosted with ❤ by GitHub view raw

Using the interface I'm defining the properties for our EC2 construct.

If you are not sure about the properties follow this documentation under the property section

Construct Hub

Construct Hub helps developers find open-source construct libraries for use with AWS CDK, CDK8s, CDKTF and other…

constructs.dev

Now we'll define the values which will be taken by the different environments. for this, I'll create a folder called config and will create two separate configuration files for DEV & STG environments.

```typescript
import { EnvironmentConfig } from './../types/environment_config';

export const DEV_Environment: EnvironmentConfig = {
    environment: "DEV",
    EC2Props: {
        ami: "ami-090fa75af13c156b4",
        instanceType: "t2.micro",
        tags: {
            Name: "cdktf-demo-dev",
            Environment: "Development",
            Managedby: "Terraform-CDK"
        }
    }
}
```

dev_environment.ts hosted with ❤ by **GitHub** **view raw**

```typescript
import { EnvironmentConfig } from './../types/environment_config';

export const STG_Environment: EnvironmentConfig = {
    environment: "STG",
    EC2Props: {
        ami: "ami-090fa75af13c156b4",
        instanceType: "t2.micro",
        tags: {
            Name: "cdktf-demo-stg",
            Environment: "Staging",
            Managedby: "Terraform-CDK"
        }
    }
}
```

stg_environment.ts hosted with ❤ by **GitHub** **view raw**

It's time to create the EC2 stack. I'll maintain the stacks under the folder called lib and will create a file named ec2-stack.ts

```
 1  import { EnvironmentConfig } from './../types/environment_config';
 2  import { TerraformStack, S3Backend, TerraformOutput } from "cdktf";
 3  import { Construct } from 'constructs';
 4  import { AwsProvider, ec2 } from "@cdktf/provider-aws";
 5  export class EC2Stack extends TerraformStack {
 6      constructor(scope: Construct, id: string, props: EnvironmentConfig) {
 7        super(scope, id);
 8
 9        // Initiate the s3 backend to store the terraform statefile
10        new S3Backend(this, {
11          bucket: "ec2-cdktf-demo-bucket",
12          key: `${props.environment}/ec2stack/terraform.state`,
13          region: "us-east-1"
14        })
15
16        // AWS Provider
17        new AwsProvider(this, "AWS", {
18          region: "us-east-1",
19        });
20
21        // Creating new EC2 Instance
22        const instance = new ec2.Instance(this, `${props.environment}-CDKTF-EC2`, {
23          ...props.EC2Props
24        });
25
26        // Output Terraform Values
27        new TerraformOutput(this, "public_ip", {
28          value: instance.publicIp,
29        });
30      }
31    }
```

ec2-stack.ts hosted with ❤ by **GitHub** **view raw**

In lines no 10–14, you can see that I have initialized S3 backend. This is to store our statefile in the s3 bucket instead of keeping them locally. In line no 11, the bucket name (ec2-cdktf-demo-bucket) which I mentioned is already created beforehand in the AWS account.

Now everything is ready. Let’s call this in the main.ts

```typescript
import { STG_Environment } from './config/stg_environment';
import { DEV_Environment } from './config/dev_environment';
import { App } from "cdktf";
import { EC2Stack } from './lib/EC2-stack';

const app = new App();

new EC2Stack(app, "cdk-demo-instance-dev", DEV_Environment);

new EC2Stack(app, "cdk-demo-instance-stg", STG_Environment);

app.synth();
```

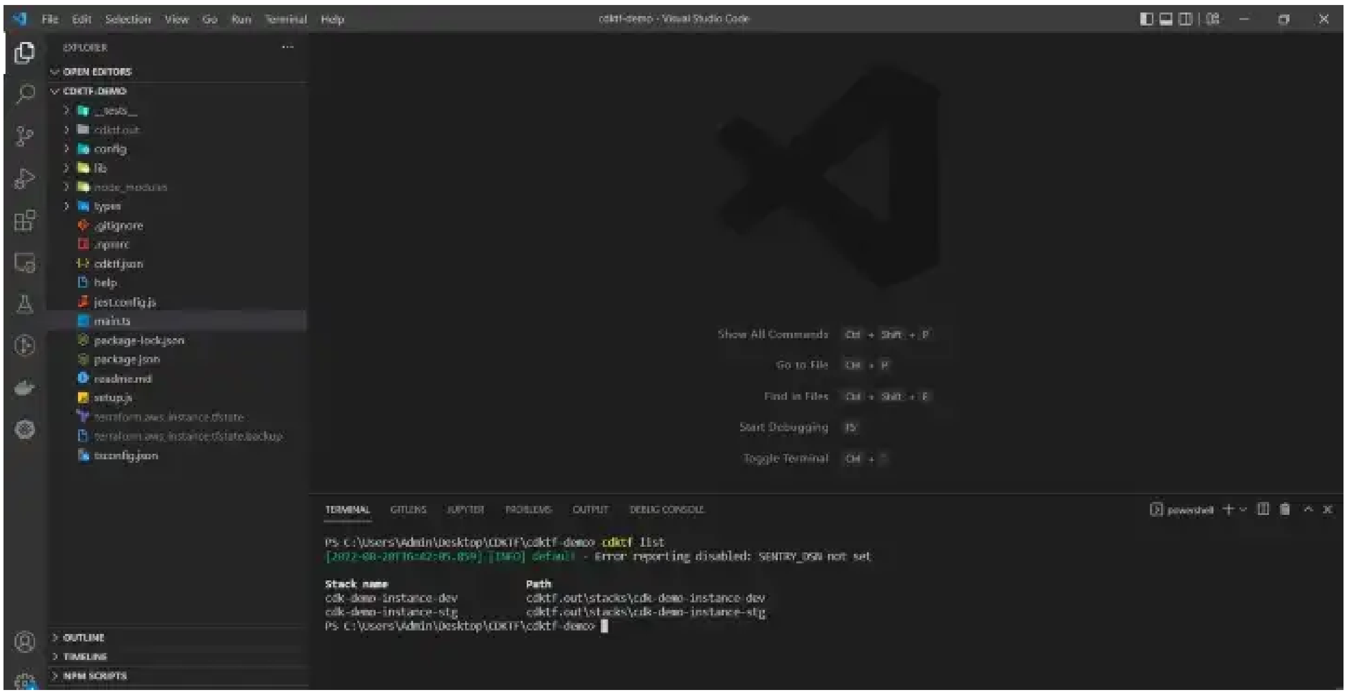
main.ts hosted with ❤ by **GitHub** **view raw**

In case, If you made any mistakes you can refer to my GitHub code as well

GitHub - ezioguga/cdktf-demo

npm install cdktf init -> Create a new cdktf project from a template. cdktf get -> Generate CDK Constructs for...

github.com



```
PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo> cdktf list
[2022-08-20T16:42:05.859] [INFO] default - Error reporting disabled: SENTRY_DSN not set

Stack name                    Path
cdk-demo-instance-dev         cdktf.out\stacks\cdk-demo-instance-dev
cdk-demo-instance-stg         cdktf.out\stacks\cdk-demo-instance-stg
PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo>
```

cdktf list

If you use the command,

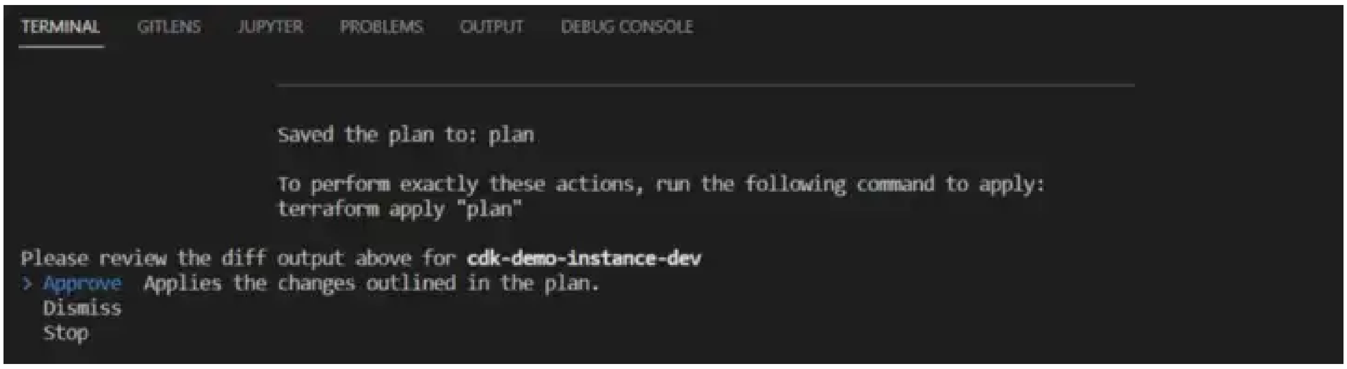
> *cdktf list*

you can get to know about the stack details.

Now, I’m going to deploy the stack for the development environment using the command

> *cdktf deploy cdk-demo-instance-dev*

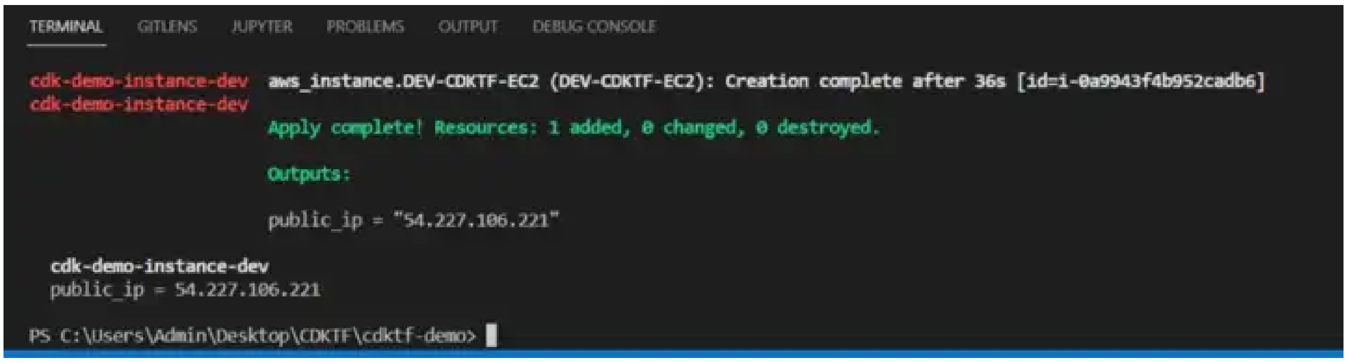
If everything is correct it’ll prompt the following



```
                    Saved the plan to: plan

                    To perform exactly these actions, run the following command to apply:
                    terraform apply "plan"

Please review the diff output above for cdk-demo-instance-dev
> Approve  Applies the changes outlined in the plan.
  Dismiss
  Stop
```

cdktf deploy

Once you have approved it, Your stack will be deployed into AWS.

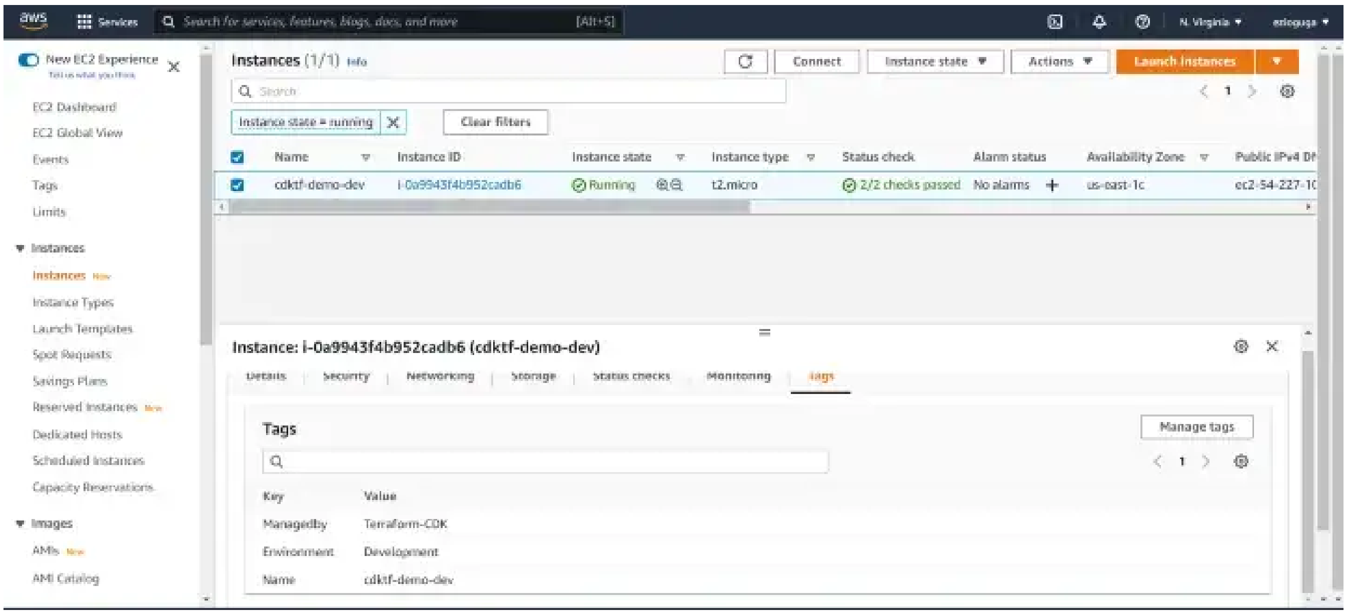


```
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Creation complete after 36s [id=i-0a9943f4b952cadb6]
cdk-demo-instance-dev
                       Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

                       Outputs:

                       public_ip = "54.227.106.221"

  cdk-demo-instance-dev
  public_ip = 54.227.106.221

PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo>
```

cdktf output

In the ec2-stack.ts we are given parameters to output the created EC2 instance’s public IP. Once it’s successfully deployed it will gives you the Instance’s public IP as the output.

You can confirm this from the AWS EC2 console as well.



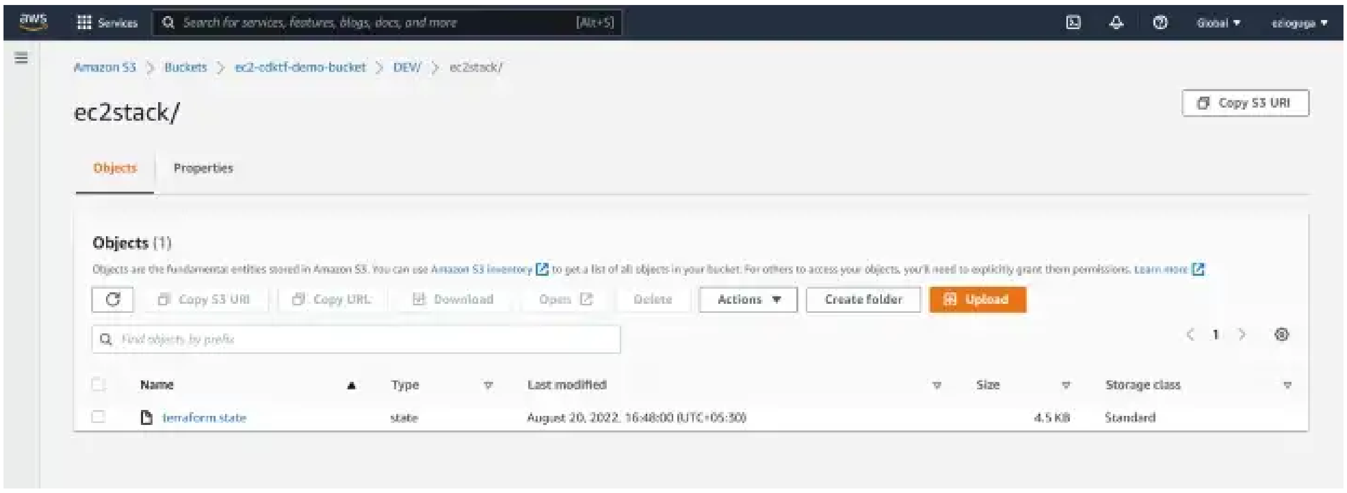
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DN
cdktf-demo-dev	i-0a9943f4b952cadb6	Running	t2.micro	2/2 checks passed	No alarms	us-east-1c	ec2-54-227-1(

Instance: i-0a9943f4b952cadb6 (cdktf-demo-dev)

Tags

Key	Value
Managedby	Terraform-CDK
Environment	Development
Name	cdktf-demo-dev

a newly created instance using cdktf



Amazon S3 > Buckets > ec2-cdktf-demo-bucket > DEV/ > ec2stack/

ec2stack/

Objects (1)

Name	Type	Last modified	Size	Storage class
terraform.state	state	August 20, 2022, 16:48:00 (UTC+05:30)	4.5 KB	Standard

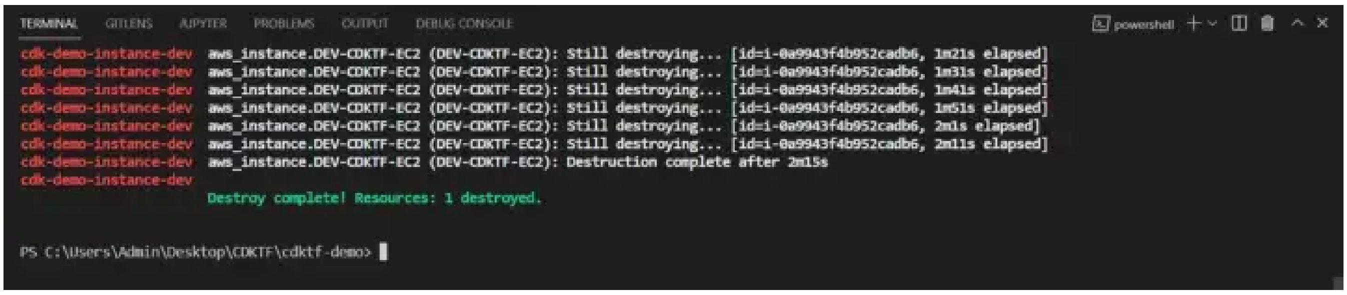
Terraform State file in S3 bucket

As you can our Terraform state file was stored in the location that we mentioned in the S3backend key section.

Let's clean up whatever resources that we spin up using the cdktf destroy command

> *cdktf destroy cdk-demo-instance-dev*

Again it'll prompt for the options, you need to approve it to delete the stack.



```
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Still destroying... [id=i-0a9943f4b952cadb6, 1m21s elapsed]
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Still destroying... [id=i-0a9943f4b952cadb6, 1m31s elapsed]
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Still destroying... [id=i-0a9943f4b952cadb6, 1m41s elapsed]
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Still destroying... [id=i-0a9943f4b952cadb6, 1m51s elapsed]
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Still destroying... [id=i-0a9943f4b952cadb6, 2m1s elapsed]
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Still destroying... [id=i-0a9943f4b952cadb6, 2m11s elapsed]
cdk-demo-instance-dev  aws_instance.DEV-CDKTF-EC2 (DEV-CDKTF-EC2): Destruction complete after 2m15s
cdk-demo-instance-dev
                       Destroy complete! Resources: 1 destroyed.


PS C:\Users\Admin\Desktop\CDKTF\cdktf-demo>
```

```
cdktf destroy
```

You can refer to my GitHub repository readme file for other additional CDKTF commands.

GitHub - ezioguga/cdktf-demo

npm install cdktf init -> Create a new cdktf project from a template. cdktf get -> Generate CDK Constructs for…

github.com

Additional Resources:

CDK for Terraform | Terraform by HashiCorp

Search Terraform documentation Cloud Development Kit for Terraform (CDKTF) allows you to use familiar programming…

www.terraform.io

I would like to thank Carly & Ryan for teaching me some of the best practices to be followed when defining CDKTF Stack.

https://www.linkedin.com/in/carlyerichardson/

https://www.linkedin.com/in/ryan-c-weaver/

I hope you guys enjoy this article, will meet you guys again with another hot topic :) Thank You.

AWS Cdktf Terraform Typescript Cloud Computing

Get an email whenever Gugatharsan Sivalingam publishes.