1. **What is CDK for Terraform?**

CDK for Terraform allows you use familiar programming languages to define and provision infrastructure. And this gives you access to entire terraform ecosystem without learning Hashicorp configurations language (HCL) and lets you leverage the power of your existing toolchain for testing, dependency management, etc

Static

2. Static website on S3

   2.1. steps

   - Setup CDKTF project and install cdktf globally using npm install -g cdktf-cli and Initialize it
   - Define your infrastructure in your main.ts
     Here is the code;

```
import { Construct } from "constructs";
import { App, TerraformStack, TerraformOutput } from "cdktf";
import { Provider, S3Bucket, S3BucketPolicy } from '@cdktf/provider-aws';


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

// Define AWS provider
new Provider(this, 'awsconstruct', {
region: 'us-east-1',
profile: 'staticWebsiteAssignment',
sharedCredentialsFile: "C:/Users/olibo/CDKTF"
});

// Create S3 bucket
const bucket = new S3Bucket(this, 'staticWebsiteBucket', {
bucket: 'StaticWebSite', // Replace 'your-bucket-name' with your desired bucket name
website: [{
indexDocument: 'index.html', // The default page when accessing the root URL
errorDocument: 'error.html' // The error page
}]
});

// Create a bucket policy to allow public read access
new S3BucketPolicy(this, 'staticWebsiteBucketPolicy', {
bucket: bucket.bucketName,
policy: JSON.stringify({
Version: "2012-10-17",
```

```
Statement: [{
Effect: "Allow",
Principal: "*",
Action: [
"s3:GetObject"
],
Resource: [
`arn:aws:s3:::${bucket.bucketName}/*`
]
}]
})
});

// Output the bucket website URL
new TerraformOutput(this, 'bucketWebsiteUrl', {
value: bucket.websiteEndpoint
});
}
}

const app = new App();
new MyStack(app, 'MyStack');
app.synth();
```

and safe your index.html in your directory and deploy

3. Website on EC2

2.1. step by step instructions

- Setup CDKTF project
  - Initialize new cdktf project
  - Insure you have Node.js and npm installed
  - Install cdktf globally → npm install -g cdktf-cli
  - Initialize a cdktf project → cdktf init –template =typescript
- Define your infrastructure
  - Open the generated main.ts file in your text editor
  - Define EC2 instance
- Deploy infrastructure
  - Genereat Terraform code → cdktf synth
  - Initialize terraform cdktf get and cdkft init
  - Cdktf deploy
- Access the website

  Once deployed you can access your website

```
import { Construct } from 'constructs';
import { App, TerraformStack, TerraformOutput } from 'cdktf'
import { Provider, Instance, SecurityGroup, Subnet, Vpc } from '@cdktf/provider-aws';
class MyWebsiteStack extends TerraformStack {
constructor(scope: Construct, name: string) {
super(scope, name);

// Define AWS provider
new Provider(this, 'aws', {
region: 'us-east-1', // Change to your desired region
});

// Create VPC
const vpc = new Vpc(this, 'MyVpc', {
cidrBlock: '10.0.0.0/16',
});

// Create security group
const securityGroup = new SecurityGroup(this, 'MySecurityGroup', {
vpcId: vpc.id,
ingress: [{
fromPort: 80,
toPort: 80,
protocol: 'tcp',
cidrBlocks: ['0.0.0.0/0'],
}],
});

// Create subnet
const subnet = new Subnet(this, 'MySubnet', {
vpcId: vpc.id,
cidrBlock: '10.0.0.0/24',
availabilityZone: 'us-east-1a', // Change to your desired AZ
});

// Create EC2 instance
const instance = new Instance(this, 'MyEC2Instance', {
ami: '296190057073', // Amazon Linux 2 AMI ID
instanceType: 't2.micro',
subnetId: subnet.id,
securityGroups: [securityGroup.name],
```

userData: `#!/bin/bash\nsudo yum install -y httpd\nsudo systemctl start httpd\nsudo systemctl enable httpd`,
});

// Output instance's public IP
new TerraformOutput(this, 'PublicIP', {
value: instance.publicIp,
});
}
}

const app = new App();
new MyWebsiteStack(app, 'MyWebsiteStack');
app.synth();

What is different from our classroom lab?

In the classroom, we write a code and run terraform code to run on AWS ec2 but this is one is somehow testing as if the code is working.

4. Benefits of CDK for Terraform?
    a. High-level abstractions
    b. Reduced boilerplate
    c. Leverage native constructs.
    d. Type safety and IDE support.
    e. Integration with existing codebases
    f. Community and ecosystem
    g. Direct SDK integration
    h. Managed deployment

4.1 CloudFormation vs. Terraform

Criteria	***Terraform***	***CloudFormation***
Scope	Covers most AWS resource and faster in supporting new features	Covers major AWS part but it low supporting new capabilities
License	It is an open source project	It is opened by AWS for free
Support	Hashicorp offers 24’7 support	AWS support plans that offer CloudFormation support
Modularity	It offers complete native support for modules	Does not support the use of modules but has feature to modularize template
State management	It store the state over disk and allow using remote state	Store and manage the state with use of stack

Change verification	It uses a command ‘plan’ for identifying needed changes	It uses ‘changes set’ to verify the required or implemented changes

Refer: https://www.whizlabs.com/blog/terraform-vs-cloudformation-vs-ansible/

CDK for CloudFormation Vs CDK for Terraform

	CDK for CloudFormation	CDK for Terraform
Infrastructure	AWS cloud formation	Hashicorp terraform
Language support	Allows developers to leverage any programming language to define infrastructure	Allows developers to use programming language to define terraform configuration
Eco system integration	Official support, documentation, and integration with AWS services	CDK for terraform extends its ecosystem of providers and modules by allowing developers to leverage the ecosystem including libraries, framework, and development tools

Refer: https://chat.openai.com/c/498711c4-ab9d-4bf4-8534-86939094b495