



AWS Cloud Development Kit (CDK) V2

Context

- **GOAL:** Reliable and consistent provisioning and configuring of cloud infrastructure is foundational for DevOps and fast software delivery.
 - Multiple environments – Development, Test, Production.
 - Multiple regions.
- **PROBLEM:** Manual processes to create infrastructure can lack:
 - consistency,
 - a single source of truth,
 - reliable detection/remediation of provisioning errors.
- **SOLUTION:** Automation → Infrastructure as Code (IaC)

Infrastructure As Code (IaC)

- Infrastructure as code allows organizations to automate and manage (cloud) infrastructure resources consistently.
 - Resources – S3 bucket, EC2 instance, SQS queue, VPC, etc.
- Advantages of IaC:
 - Provides a single source of truth.
 - Use Version Control tools to manage change.
 - Roll back changes.
 - Share and enforce best practices.

The IaC journey

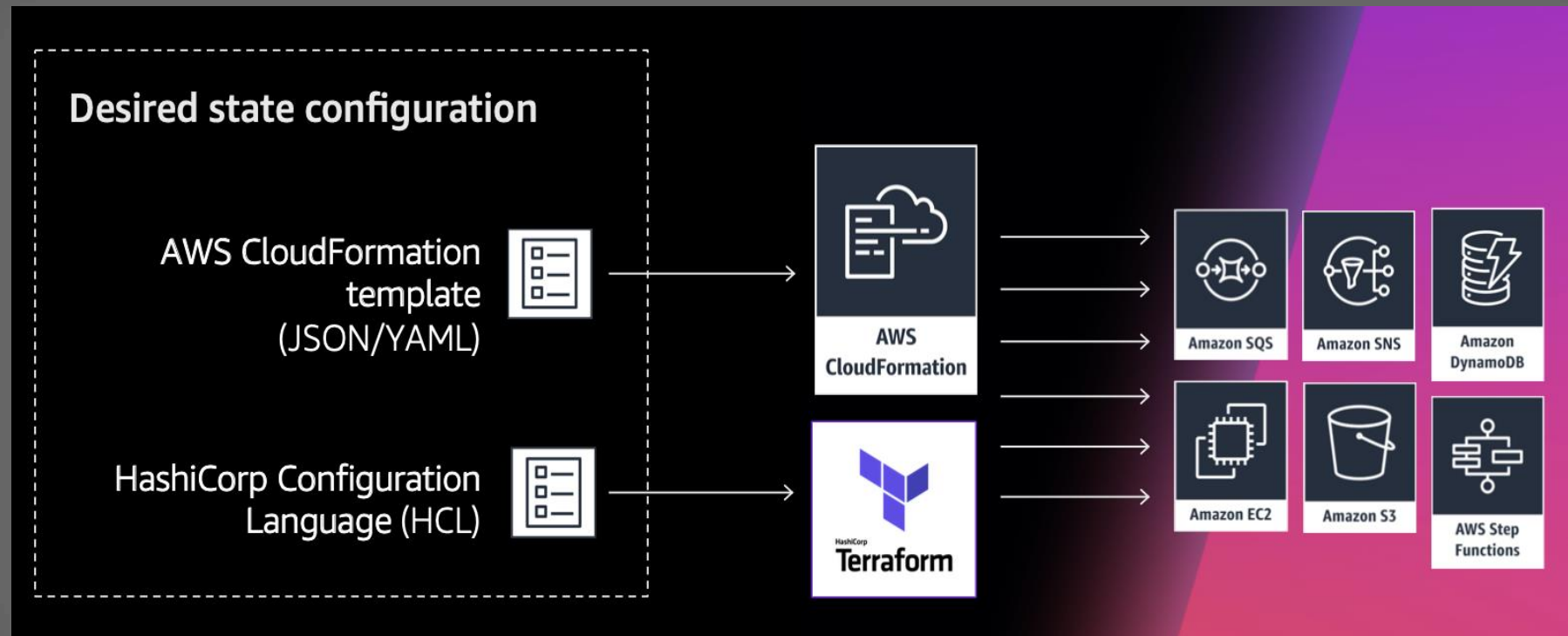
- 1st generation: The Scripted approach.



- Problems:
 - What happens if an API call fails?
 - How do I make updates to the infrastructure?
 - How do I know when a resource is ready?
 - How do I roll back the infrastructure?

The IaC journey

- 2nd generation: Resource Provisioning Engines.

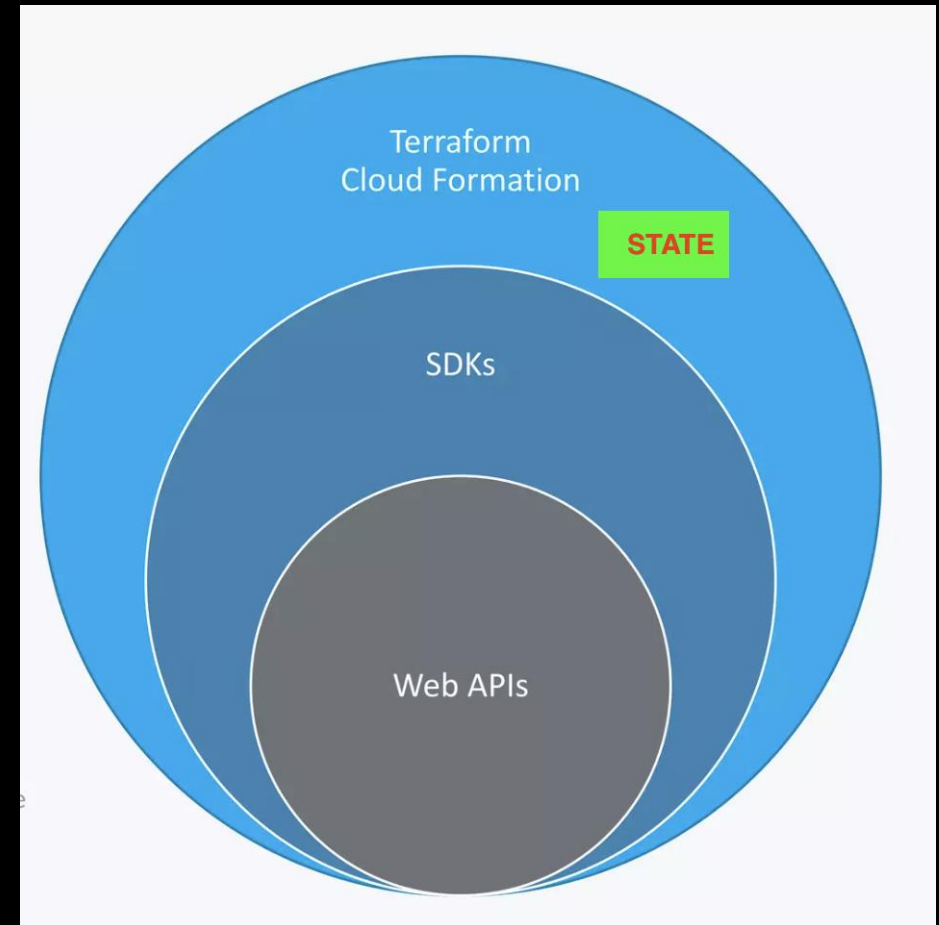


The IaC journey

- Resource Provisioning Engines.
- Advantages:
 - Easy to update the infrastructure.
 - Reproducible.
- Disadvantages
 - Configuration syntax not developer-friendly.
 - No abstractions or sensible default values, therefore, config. is verbose.

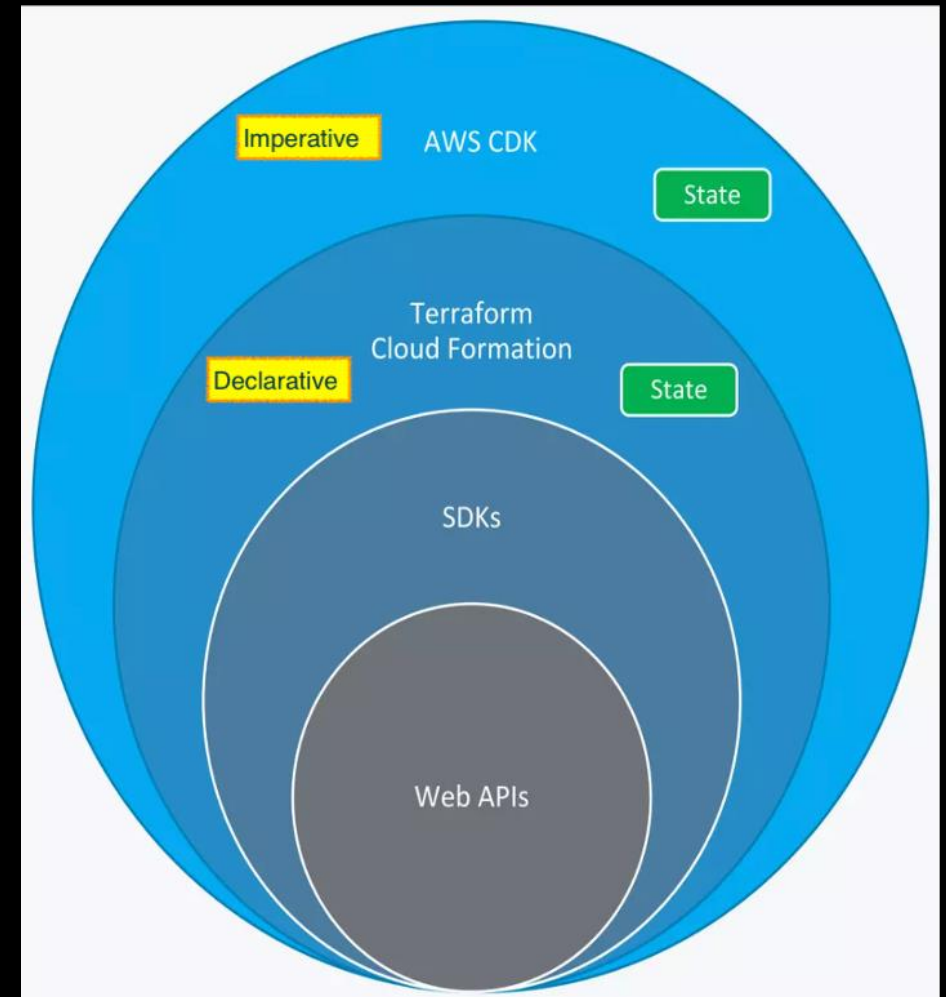
The IaC journey

1. Web APIs - AWS exposed majority of their services publicly using REST APIs,
2. SDKs – AWS provided SDKs in all the major programming languages.
3. CloudFormation (2011) – next level abstraction of SDKs.
 - Provides a set of tools to define infrastructure declaratively.(YAML/JSON)
 - Manages updates to infrastructure state.
4. HCL TerraForm (2014) – Open source.



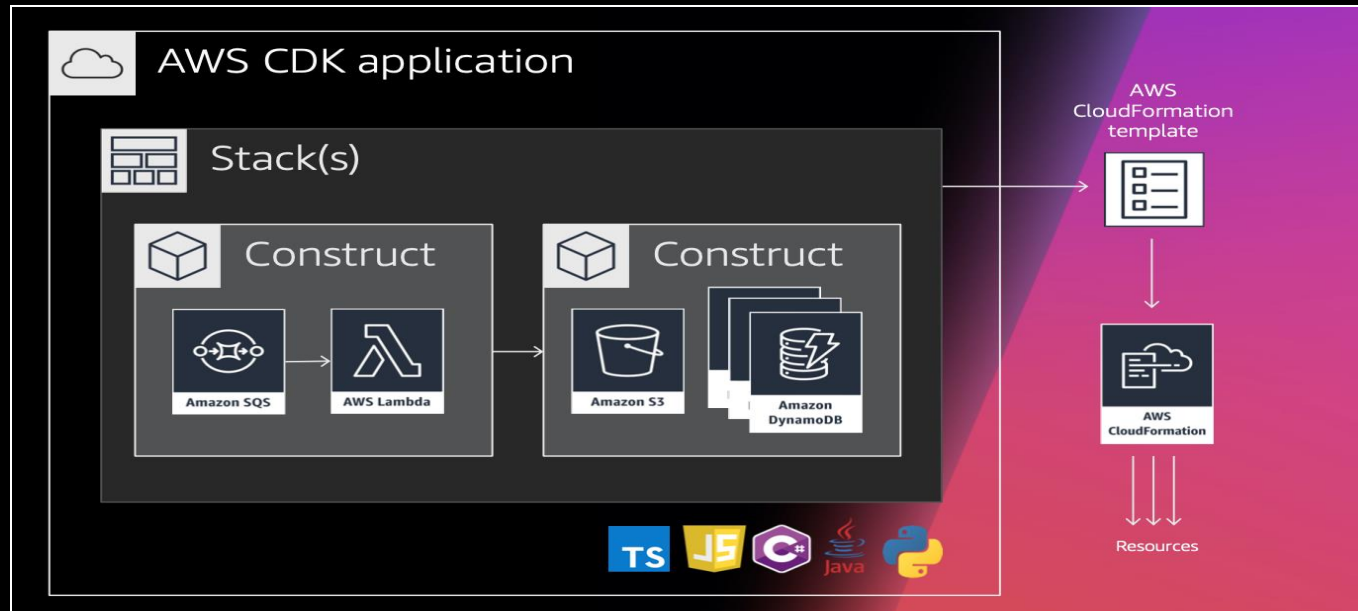
CDK framework – 3rd generation IaC

- August 2019 – proof of concept
- Goal - Describe infrastructure in an imperative language.
- Supports TypeScript/JS, Java, Python, C#, Go, and growing.
- Class libraries of constructs with sensible defaults.
- Abstractions-heavy.
- Better Developer experience (DX).
 - IDE hinting/intellisense.
- Better Developer productivity
 - CF LoC >> CDK Loc
- Unit testing.



CDK concepts

- Application (App) >> Stack >> Construct >> Resources



- A stack is the unit of deployment, similar to to CloudFormation.

Developer Productivity

- Ex: Provision an EC2 instance with the default security policy and attached to the default VPC.

```
const defaultVpc = ec2.Vpc.fromLookup(this, 'VPC', {isDefault: true});

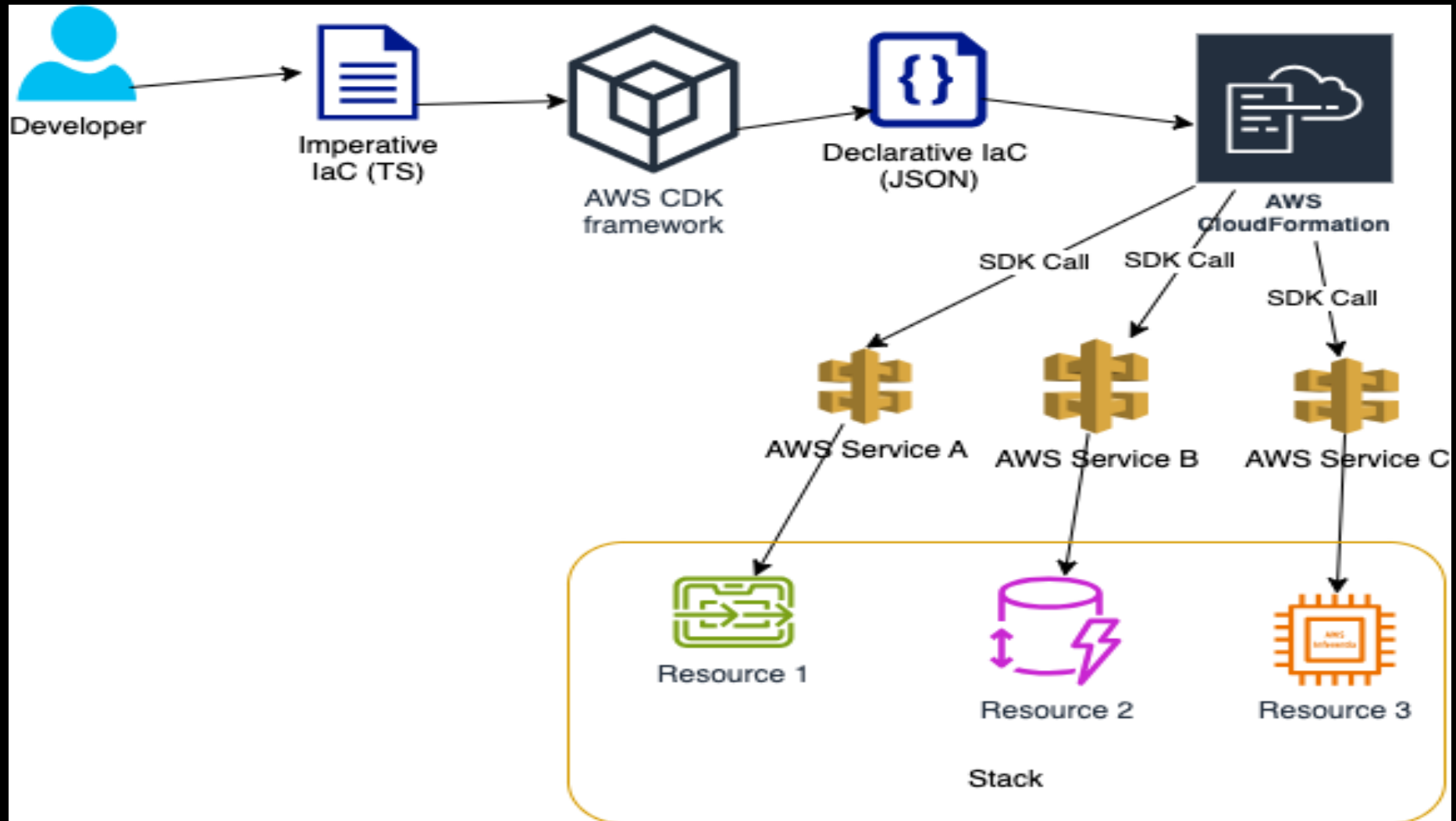
const ec2Instance = new ec2.Instance(this, 'ec2-instance', {
  vpc: defaultVpc,
  instanceType: ec2.InstanceType.of(
    ec2.InstanceClass.BURSTABLE2,
    ec2.InstanceSize.MICRO,
  ),
  machineImage: new ec2.AmazonLinuxImage({
    generation: ec2.AmazonLinuxGeneration.AMAZON_LINUX_2,
  }),
  keyName: 'ec2-key-pair',
});
```

12 LOC

```
{
  "Resources": {
    "ec2InstanceInstanceSecurityGroupAE914F6C": {
      "Type": "AWS::EC2::SecurityGroup",
      "Properties": {
        "GroupDescription": "ec2-stack/ec2-instance/InstanceSecurityGroup",
        "SecurityGroupEgress": [
          {
            "CidrIp": "0.0.0.0/0",
            "Description": "Allow all outbound traffic by default",
            "IpProtocol": "-1"
          }
        ],
        "Tags": [
          {
            "Key": "Name",
            "Value": "ec2-stack/ec2-instance"
          }
        ],
        "VpcId": "vpc-2859d343"
      },
      "Metadata": {
        "aws:cdk:path": "ec2-stack/ec2-instance/InstanceSecurityGroup/Resource"
      }
    },
    "ec2InstanceInstanceRoleCA97C688": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                "Service": "ec2.amazonaws.com"
              }
            }
          ]
        },
        "Version": "2012-10-17"
      },
      "Tags": [
        {
          "Key": "Name",
          "Value": "ec2-stack/ec2-instance"
        }
      ],
      "Metadata": {
        "aws:cdk:path": "ec2-stack/ec2-instance/InstanceRole/Resource"
      }
    },
    "ec2InstanceInstanceProfile9BCE9015": {
      "Type": "AWS::IAM::InstanceProfile",
      "Properties": {
        "Roles": [
          {
            "Ref": "ec2InstanceInstanceRoleCA97C688"
          }
        ]
      },
      "Metadata": {
        "aws:cdk:path": "ec2-stack/ec2-instance/InstanceProfile"
      }
    }
  }
}
```

150 LOC

CDK execution flow.



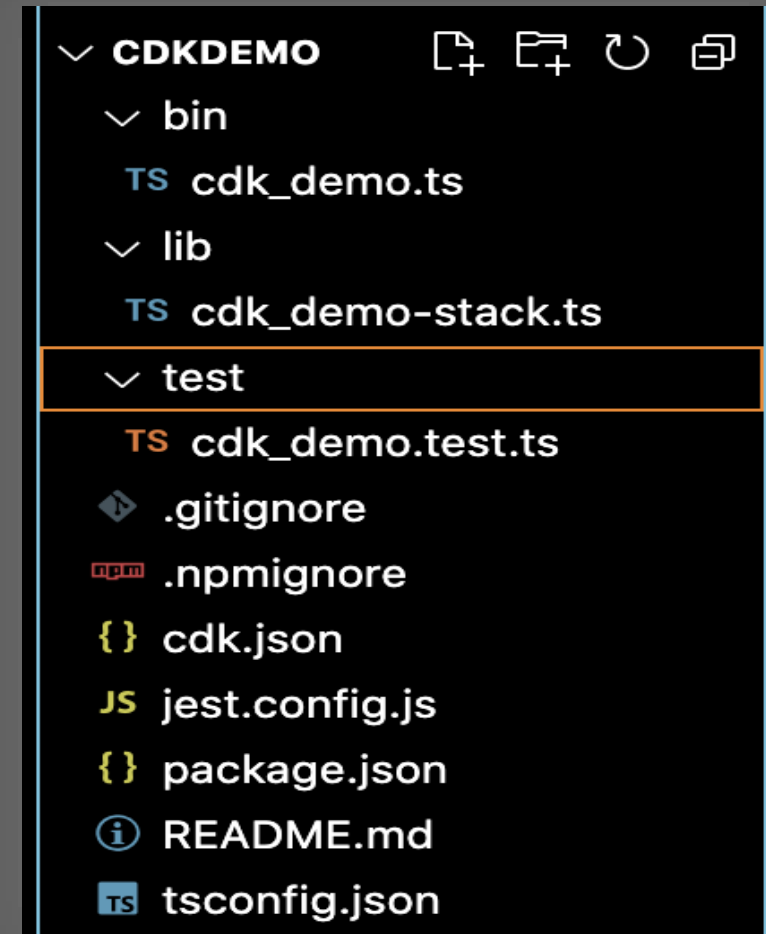
CDK workflow

- Workflow:

```
$ cdk init app --language typescript (python, go) # Scaffolding  
.... Write infrastructure code .....  
$ cdk synth # (Optional) Generate local copy of CF template  
$ cdk deploy # Deploy app stack(s)  
..... Change infrastructure code .....  
$ cdk deploy. # Updates CF template and trigger stack update  
.....  
$ cdk destroy # Request CF to destroy all stack resources
```

CDK app project structure

- `./bin/cdk_demo.ts`
 - Entry point file used by the CDK framework.
 - Where you define your app's stack composition.
- `./lib` folder
 - Contains the IaC that describes the infrastructure resources.
 - Used by bin file during synth and deploy actions.
- `./test/cdk_demo.test.ts`
 - Template test code for app.



Construct Levels

- L1 – CloudFormation resources.
 - 1:1 relationship with CF template resources. No default configuration settings. No abstractions.
- L2 – AWS constructs.
 - 1:M relationship with CF resources. Lots of default settings. High level abstraction.
- L3 – Purpose-built constructs.
 - Pattern-based. Optimized for particular use case. Community and AWS supplied.

Demo