Ari Palo / 23.11.2021 / Tampere Serverless Meetup

AWS Cloud Development Kit

– a perfect IaC tool of choice for serverless?

Ari Palo

AWS community builder

🛠️ AWS Community Builder
Developer Tooling

👨🏽‍💼 Lead Technologist

👨‍🚀 Serverlesss since 2016

👴🏻 10+ years at Alma Media

💻 TypeScript & Go

🔗 twitter.com/@aripalo

📝 aripalo.com

🔗 linkedin.com/in/aripalo

10+
European countries

1500
Employees

50+
Brands

300
Developers

100+

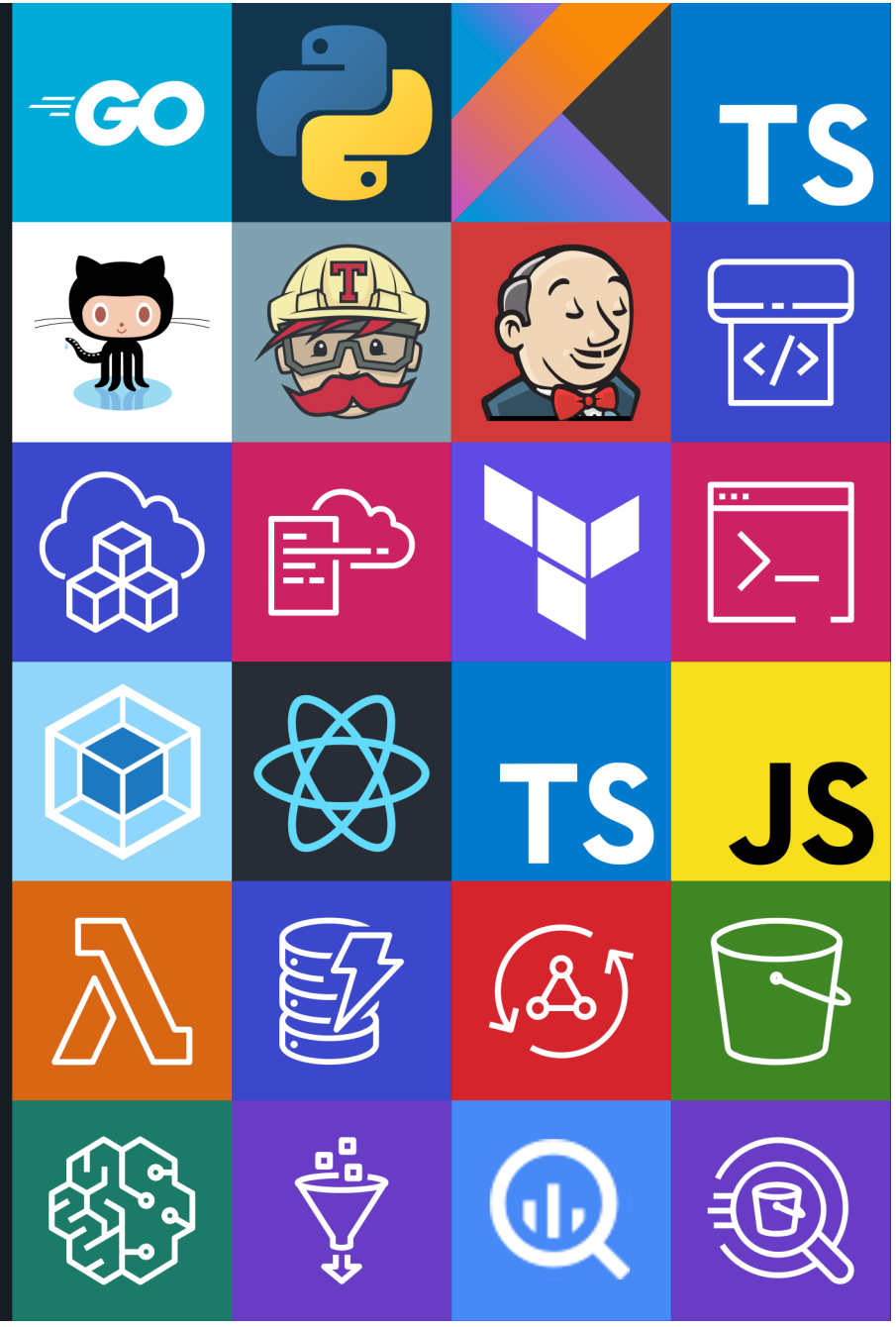
Websites & Apps

???

Backends & Services

1100

Repositories in Github



2012

Started using AWS

2016

Started using Serverless

150

AWS Accounts in the organization

<1000

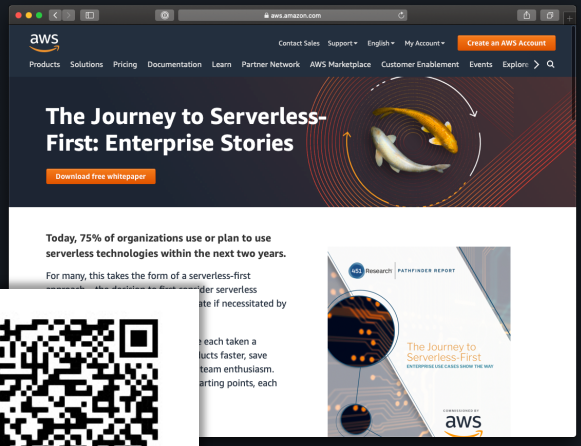
Million Lambda Invocations / Month

>1000

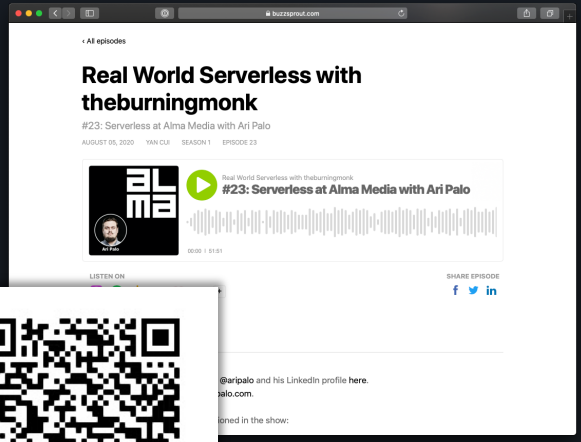
Million API Gateway Requests / Month

>200

Million AppSync Requests / Month



amzn.to/32mdiya

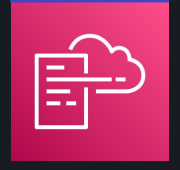


bit.ly/2RjholM

Serverless & Infrastructure-as-Code

Serverless & Infrastructure-as-Code

AWS CDK



AWS CloudFormation

Terraform

AWS CDK

"AWS Cloud Development Kit (CDK) is an open source software development framework to define your cloud application resources **using familiar programming languages**."

AWS CDK

"AWS Cloud Development Kit (CDK) is an open source software development framework to define your cloud application resources **using familiar programming languages**."

CDK Deep Dive

aripalo.com/**speaking/2020/aws-community-nordics-cdk-special/**


```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```


```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```


```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```

SNS
Topic

```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```

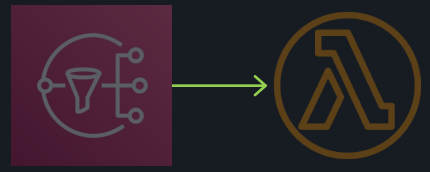
DynamoDB
Table

```
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```

Lambda
Function

```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```

Subscribe Lambda to SNS Topic

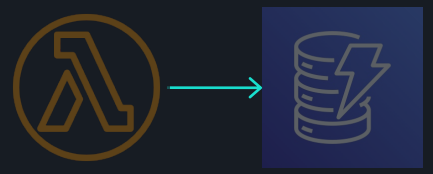


SNS Topic

Lambda Function

```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```

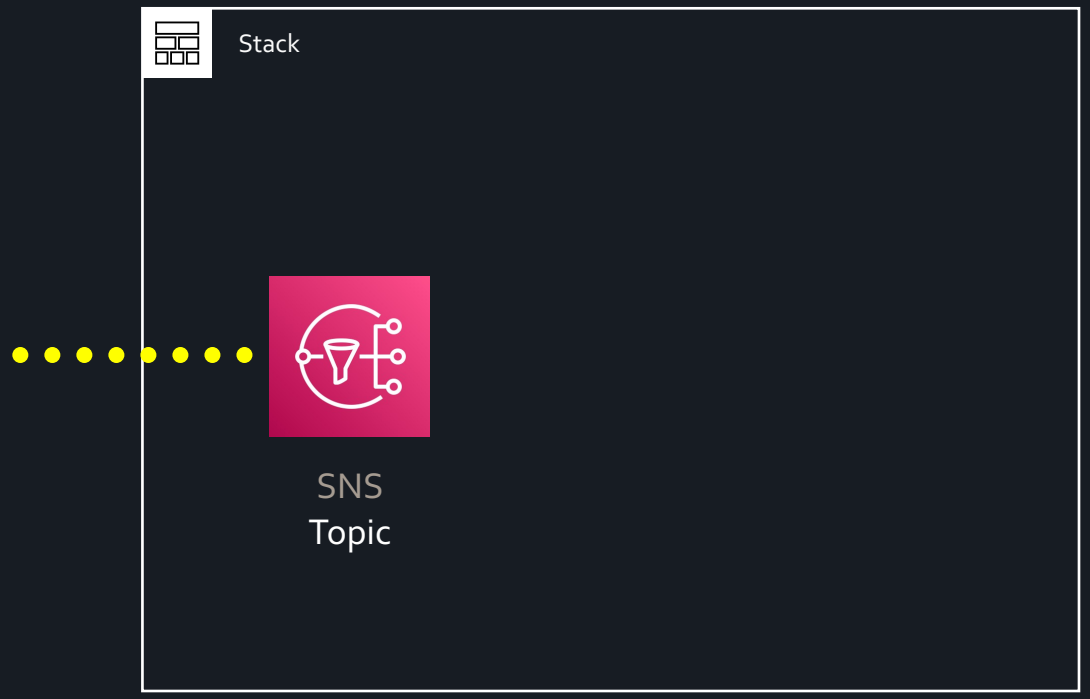
Grant write access for Lambda into DynamoDB Table



Lambda Function

DynamoDB Table

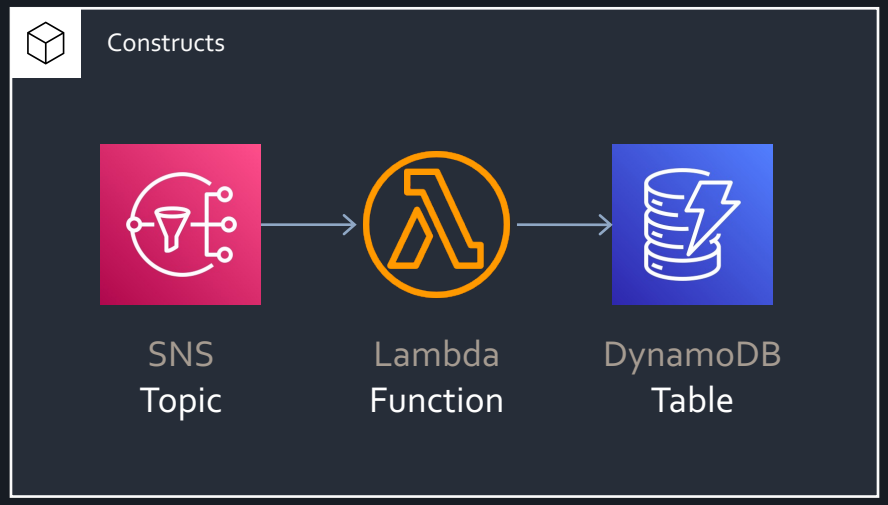
```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```



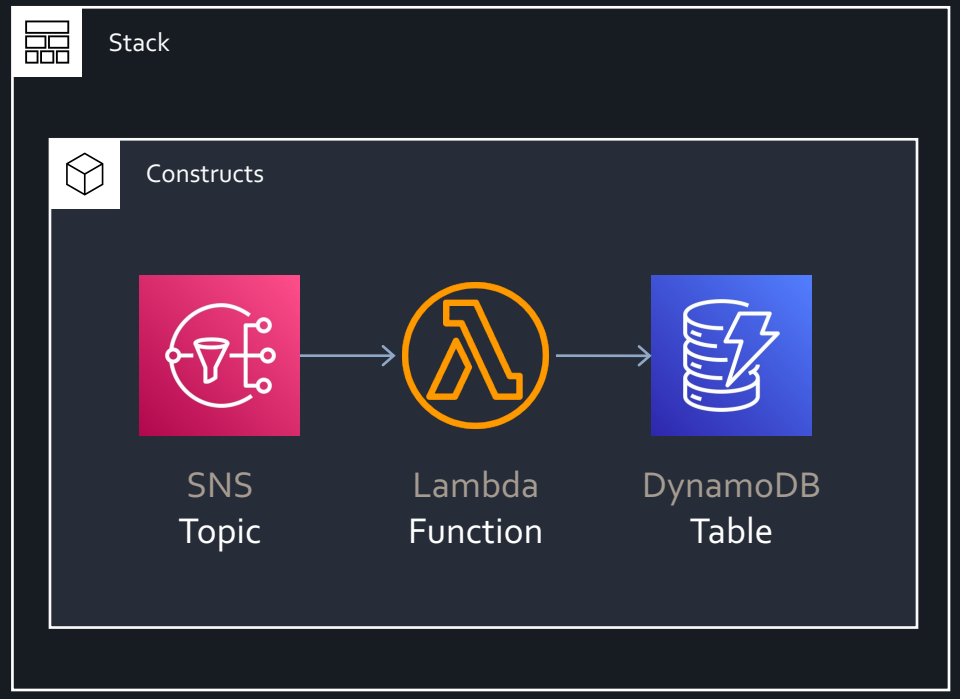
Expose the SNS Topic *outside* of the Stack so other Stacks may use it

```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the topic outside the stack
    this.topic = topic;
  }
}
```

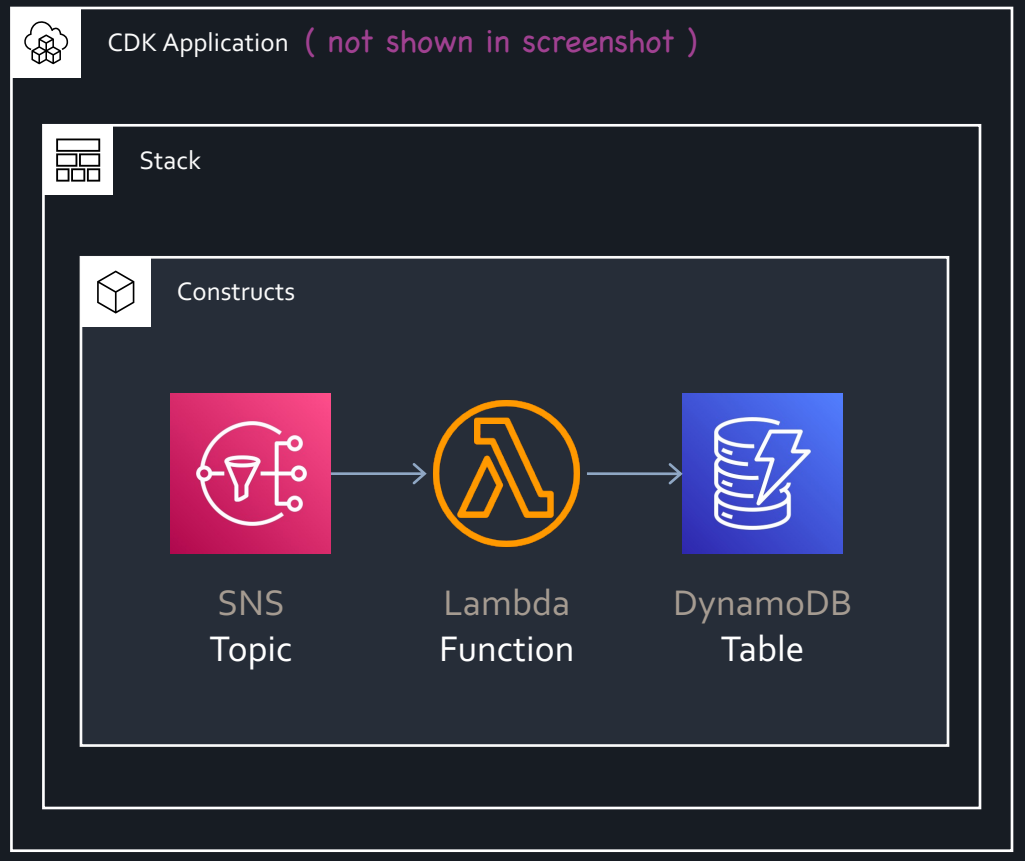

```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```



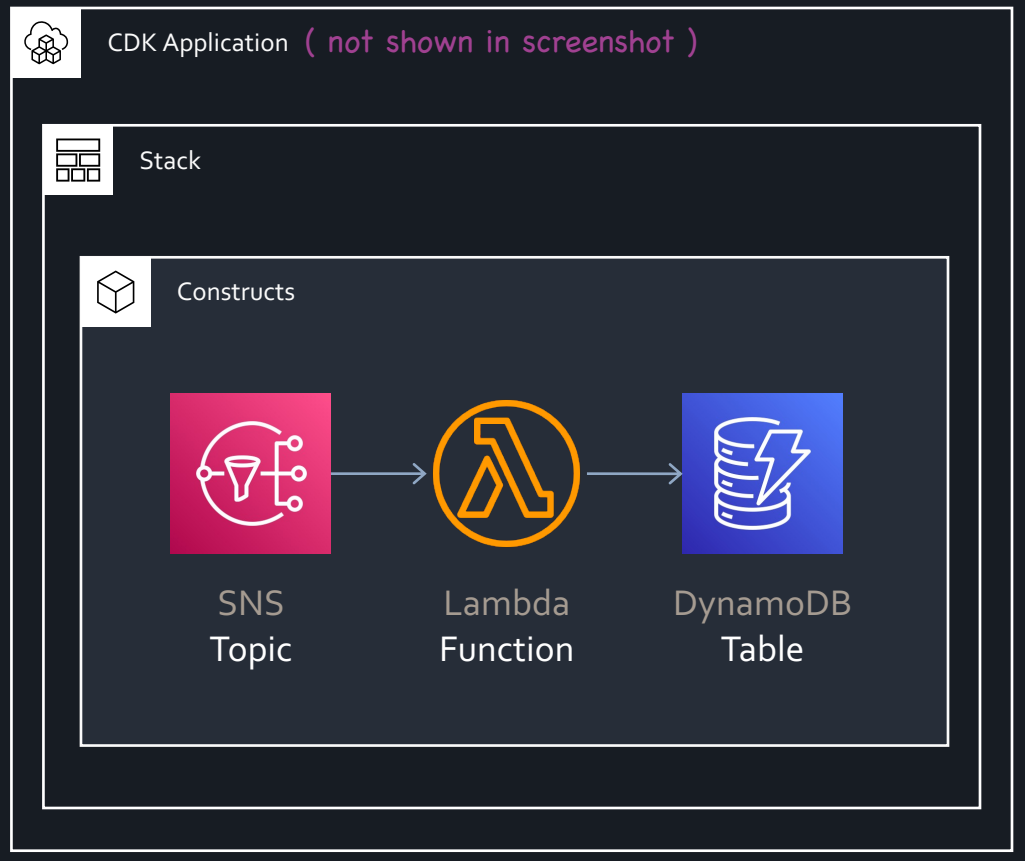

```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```



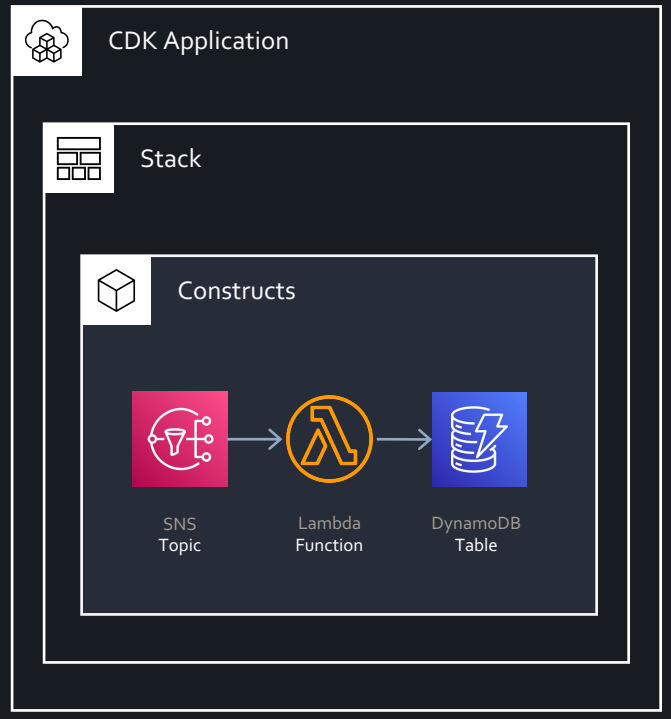

```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```

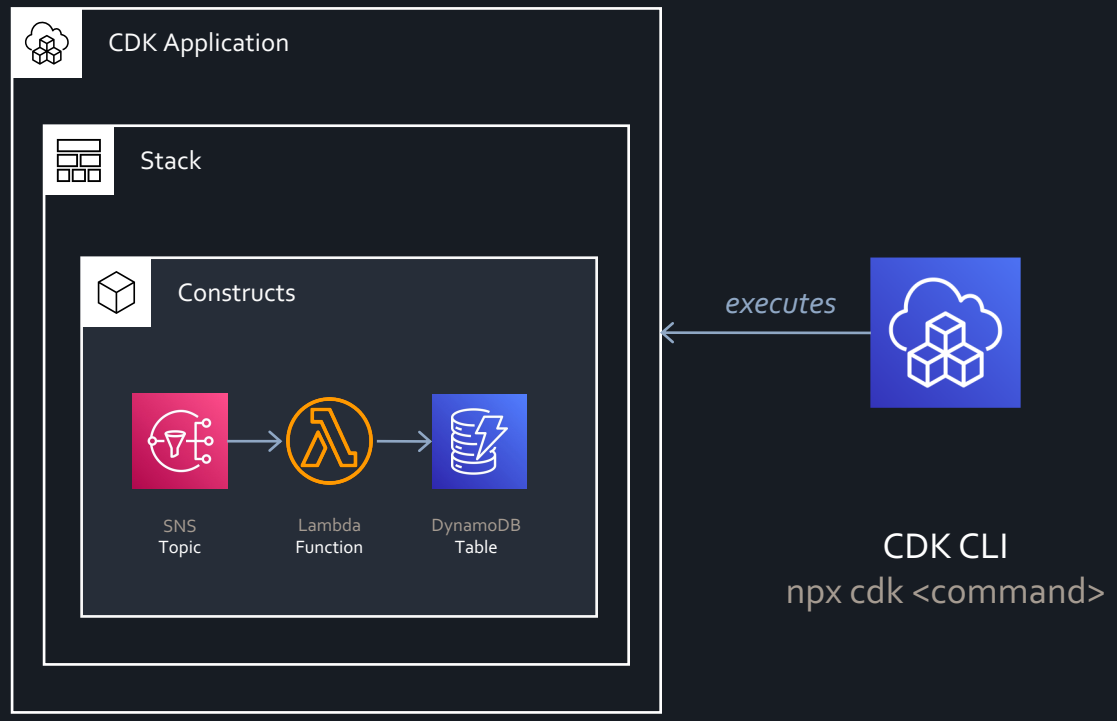



```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```




```typescript
import * as cdk from "@aws-cdk/core";
import * as sns from "@aws-cdk/aws-sns";
import * as subscriptions from "@aws-cdk/aws-sns-subscriptions";
import * as dynamodb from "@aws-cdk/aws-dynamodb";
import * as lambda from "@aws-cdk/aws-lambda";

export class NotificationToDynamoStack extends cdk.Stack {
  // Expose the Topic outside the stack
  public readonly topic: sns.ITopic;

  constructor(scope: cdk.App, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Define SNS topic
    const topic = new sns.Topic(this, "MyTopic");

    // Define DynamoDB Table
    const table = new dynamodb.Table(this, "MyTable", {
      partitionKey: { name: "PK", type: dynamodb.AttributeType.STRING },
    });

    // Define Lambda function
    const fn = new lambda.Function(this, "MyFunction", {
      runtime: lambda.Runtime.NODEJS_12_X,
      handler: "index.handler",
      code: lambda.Code.fromAsset("./my-function"),
      memorySize: 1024,
      environment: {
        TABLE_NAME: table.tableName,
      },
    });

    // Subscribe Lambda function to SNS topic
    topic.addSubscription(new subscriptions.LambdaSubscription(fn));

    // Grant DynamoDB Table write access for Lambda function
    table.grantWriteData(fn);

    // Expose the Topic outside the stack
    this.topic = topic;
  }
}
```

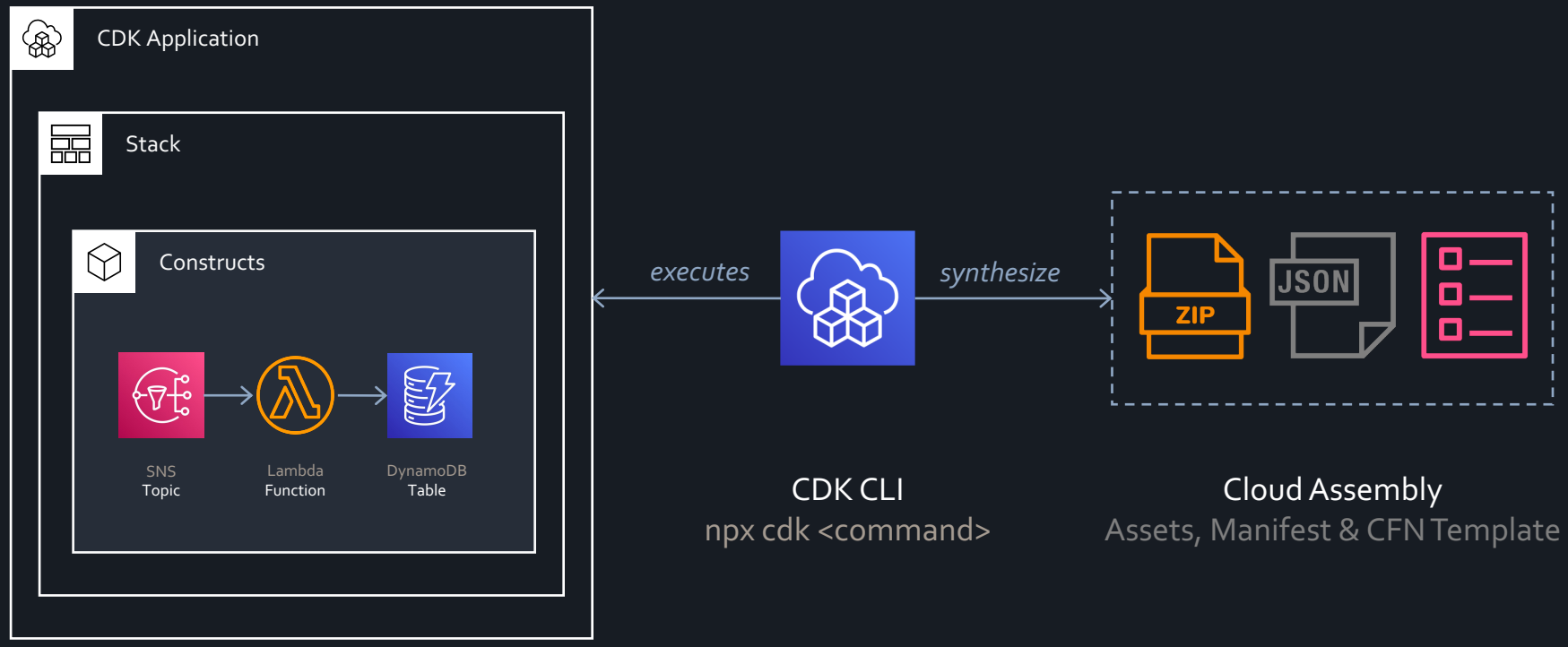

CDK Application
Stack
Constructs
SNS
Topic
Lambda
Function
DynamoDB
Table

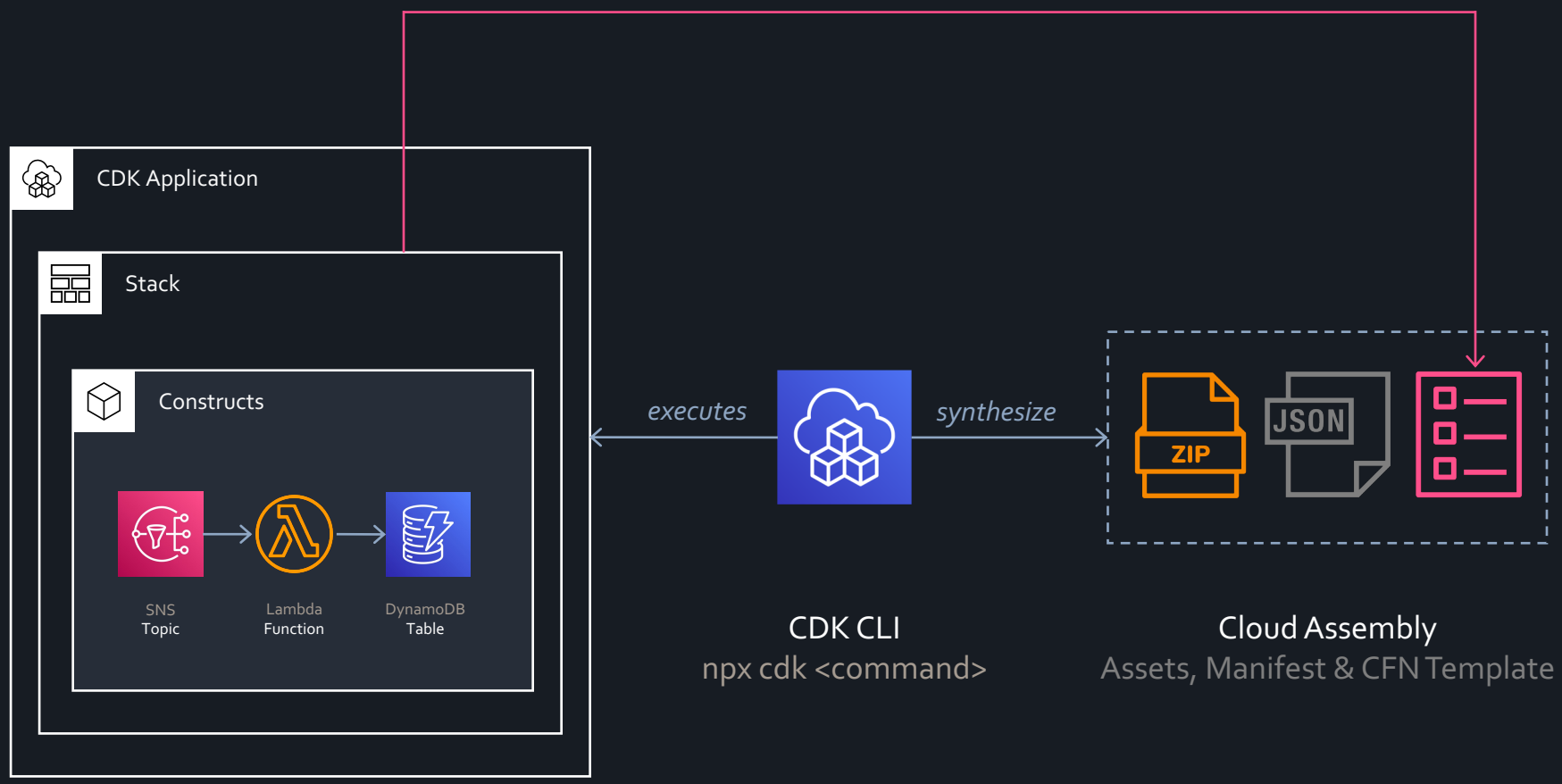


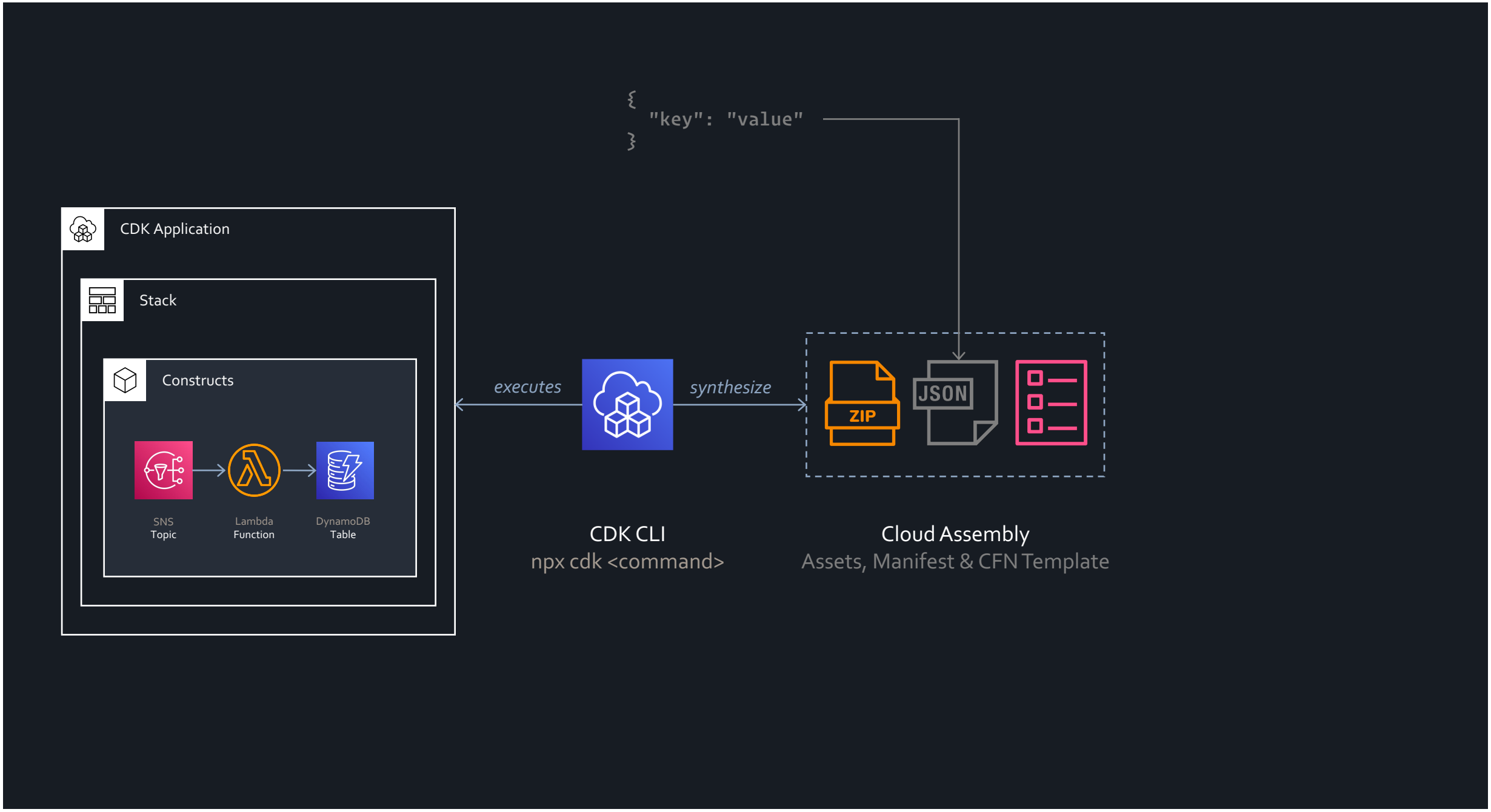
executes

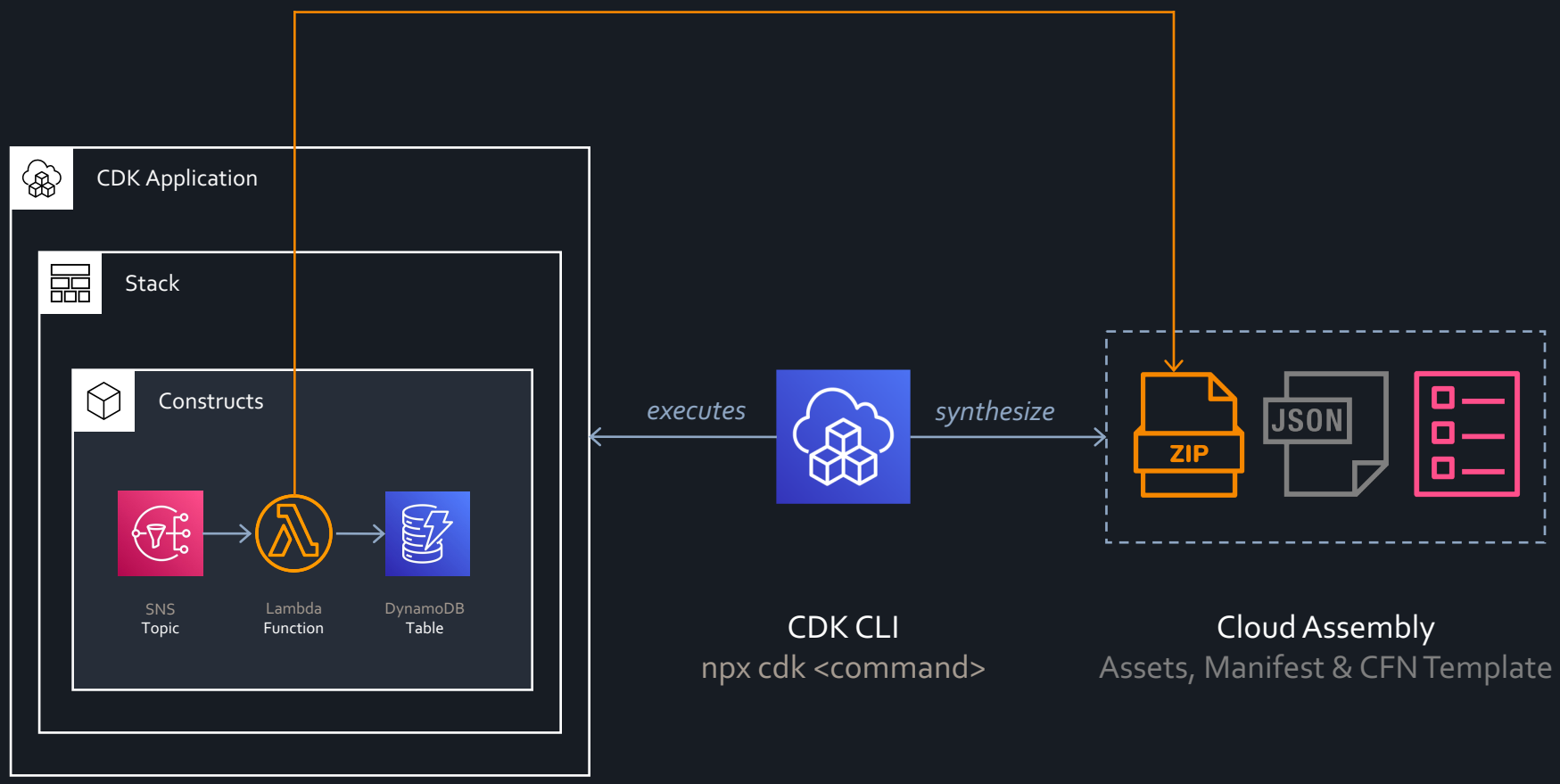
CDK CLI

npx cdk <command>


CDK Application
Stack
Constructs
SNS
Topic
Lambda
Function
DynamoDB
Table
executes
synthesize
ZIP
JSON
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template


CDK Application
Stack
Constructs
SNS
Topic
Lambda
Function
DynamoDB
Table
executes
synthesize
ZIP
JSON
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template


{
"key": "value"
}
CDK Application
Stack
Constructs
SNS
Topic
Lambda
Function
DynamoDB
Table
executes
synthesize
ZIP
JSON
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template



executes

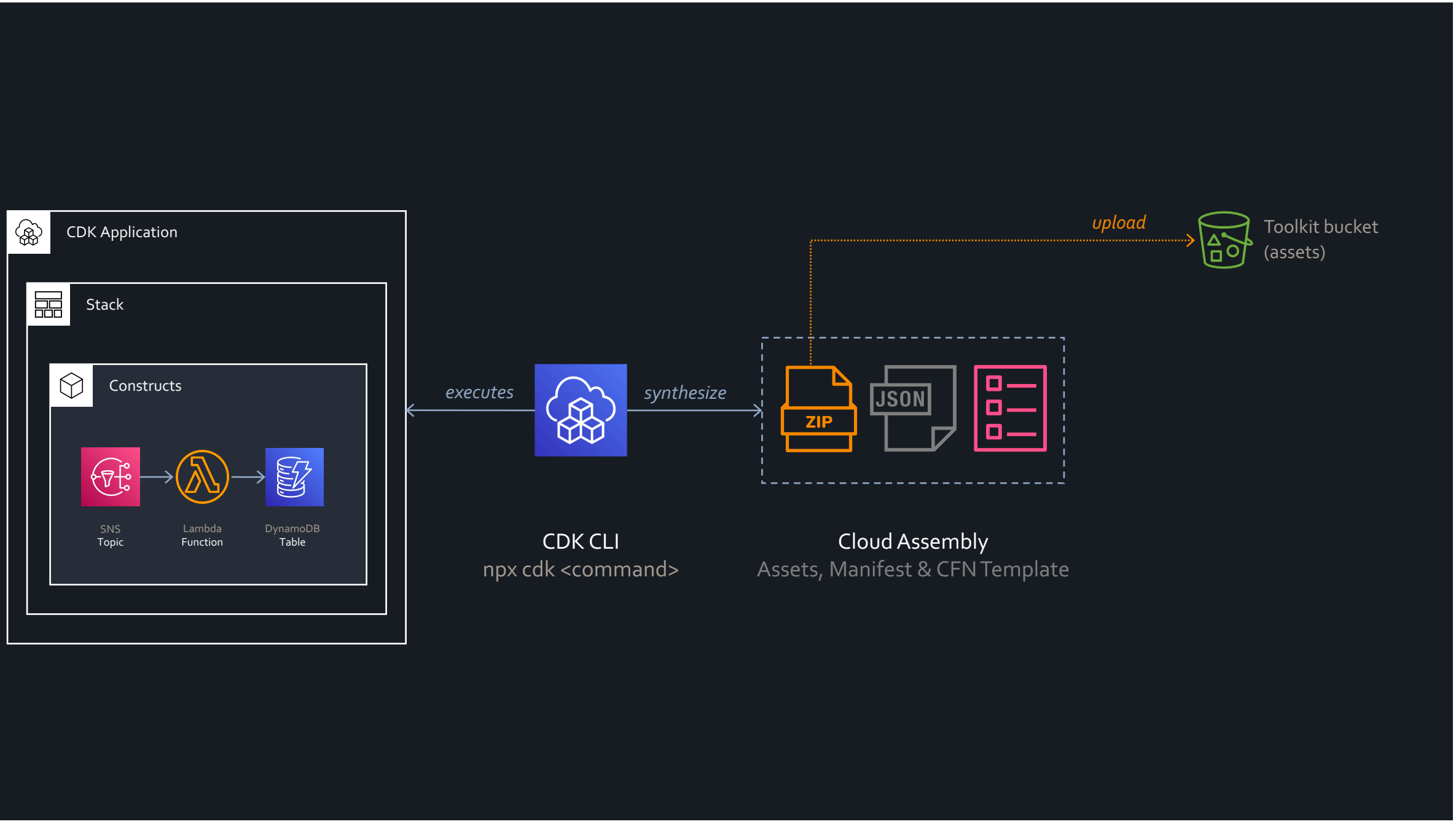
synthesize

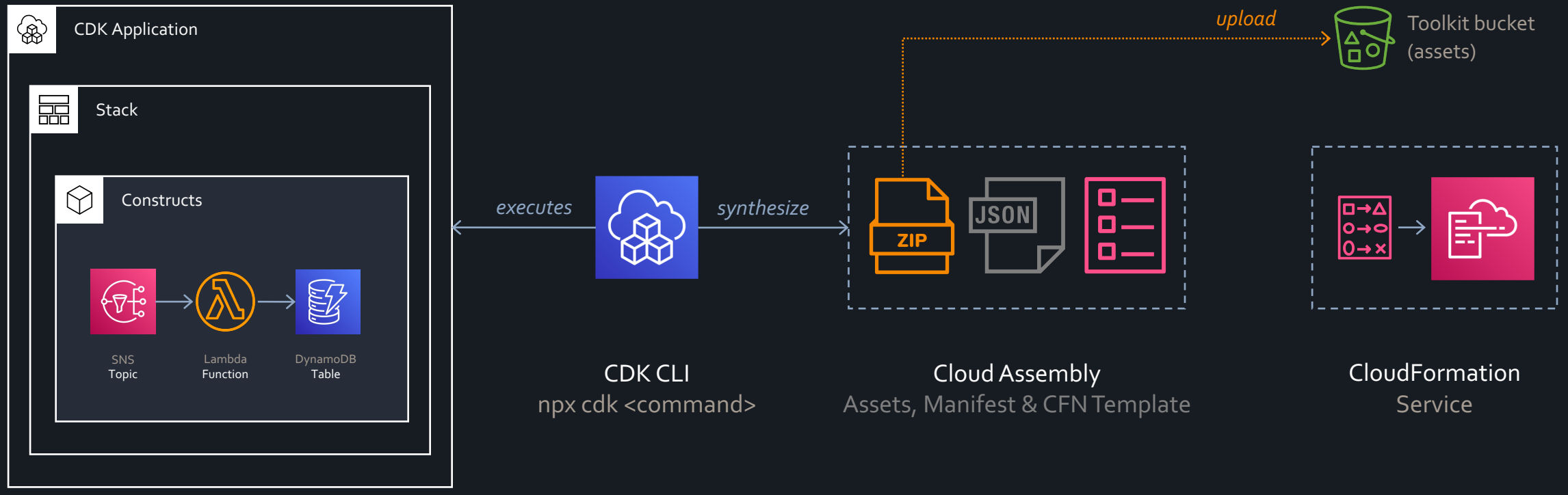
CDK CLI

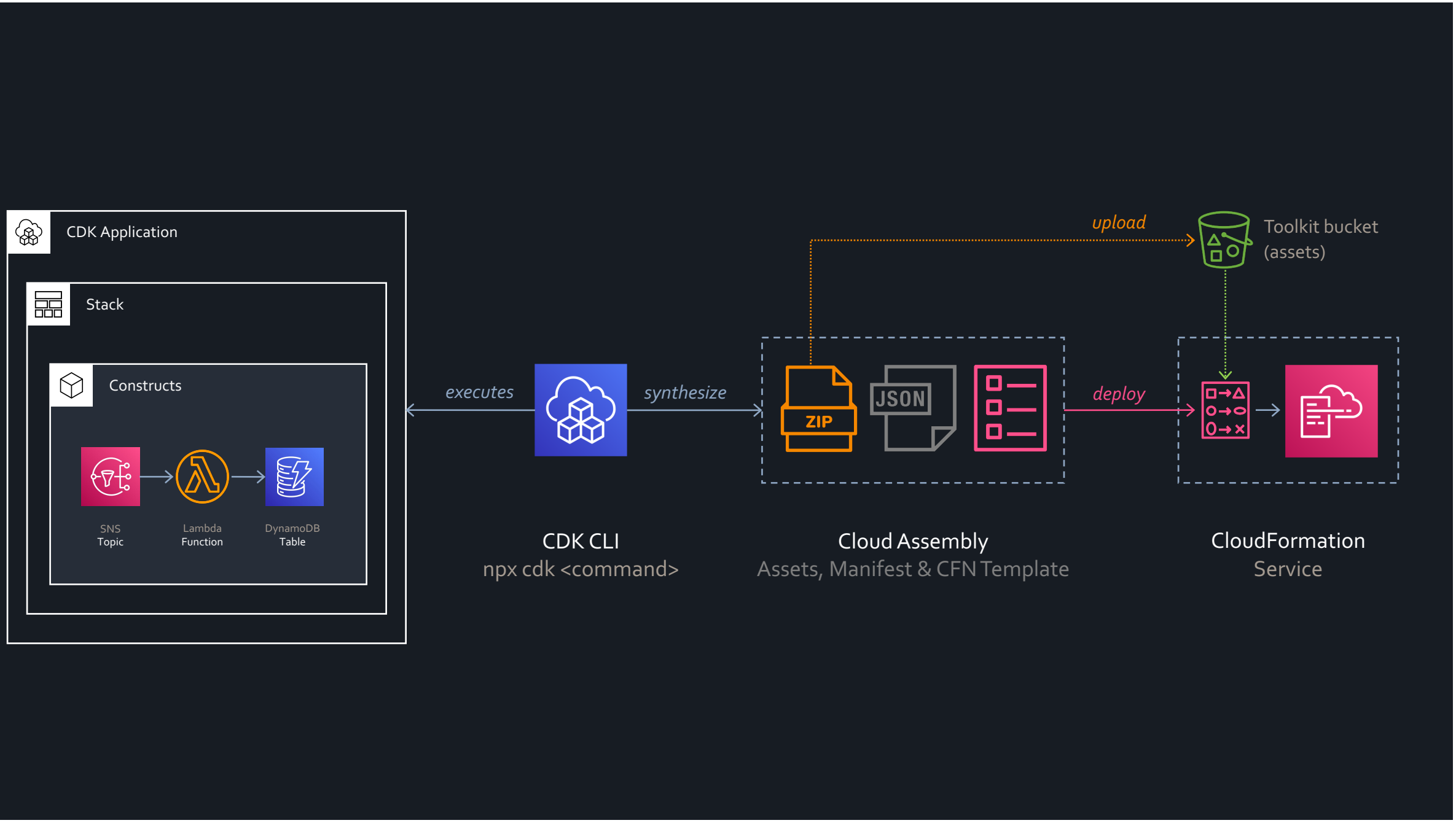
npx cdk <command>

Cloud Assembly

Assets, Manifest & CFN Template


CDK Application
Stack
Constructs
SNS
Topic
Lambda
Function
DynamoDB
Table
executes
synthesize
upload
Toolkit bucket
(assets)
ZIP
JSON
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template


CDK Application
Stack
Constructs
SNS
Topic
Lambda
Function
DynamoDB
Table
executes
synthesize
upload
Toolkit bucket
(assets)
ZIP
JSON
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template
CloudFormation
Service


CDK Application
Stack
Constructs
SNS Topic
Lambda Function
DynamoDB Table
executes
synthesize
upload
Toolkit bucket (assets)
ZIP
JSON
deploy
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template
CloudFormation
Service

CDK CLI

npx cdk <command>

+ Serverless

Serverless?

Serverless

- Functions-as-a-Service (FaaS): AWS Lambda

Serverless

- Functions-as-a-Service (FaaS): AWS Lambda
- API Management: AWS API Gateway & AppSync

Serverless

- Functions-as-a-Service (FaaS): AWS Lambda
- API Management: AWS API Gateway & AppSync
- Event Delivery: AWS SNS, EventBridge, Kinesis…
- Message Queues: AWS SQS

Serverless

- Functions-as-a-Service (FaaS): AWS Lambda
- API Management: AWS API Gateway & AppSync
- Event Delivery: AWS SNS, EventBridge, Kinesis…
- Message Queues: AWS SQS
- Databases: AWS DynamoDB, QLDB, Aurora Serverless…

Serverless

- Functions-as-a-Service (FaaS): AWS Lambda
- API Management: AWS API Gateway & AppSync
- Event Delivery: AWS SNS, EventBridge, Kinesis…
- Message Queues: AWS SQS
- Databases: AWS DynamoDB, QLDB, Aurora Serverless…
- Object Storage: AWS S3

Serverless

- Functions-as-a-Service (FaaS): AWS Lambda
- API Management: AWS API Gateway & AppSync
- Event Delivery: AWS SNS, EventBridge, Kinesis…
- Message Queues: AWS SQS
- Databases: AWS DynamoDB, QLDB, Aurora Serverless…
- Object Storage: AWS S3
- Orchestration: AWS Step Functions
- Config Management: Parameter Store, Secrets Manager…

Serverless

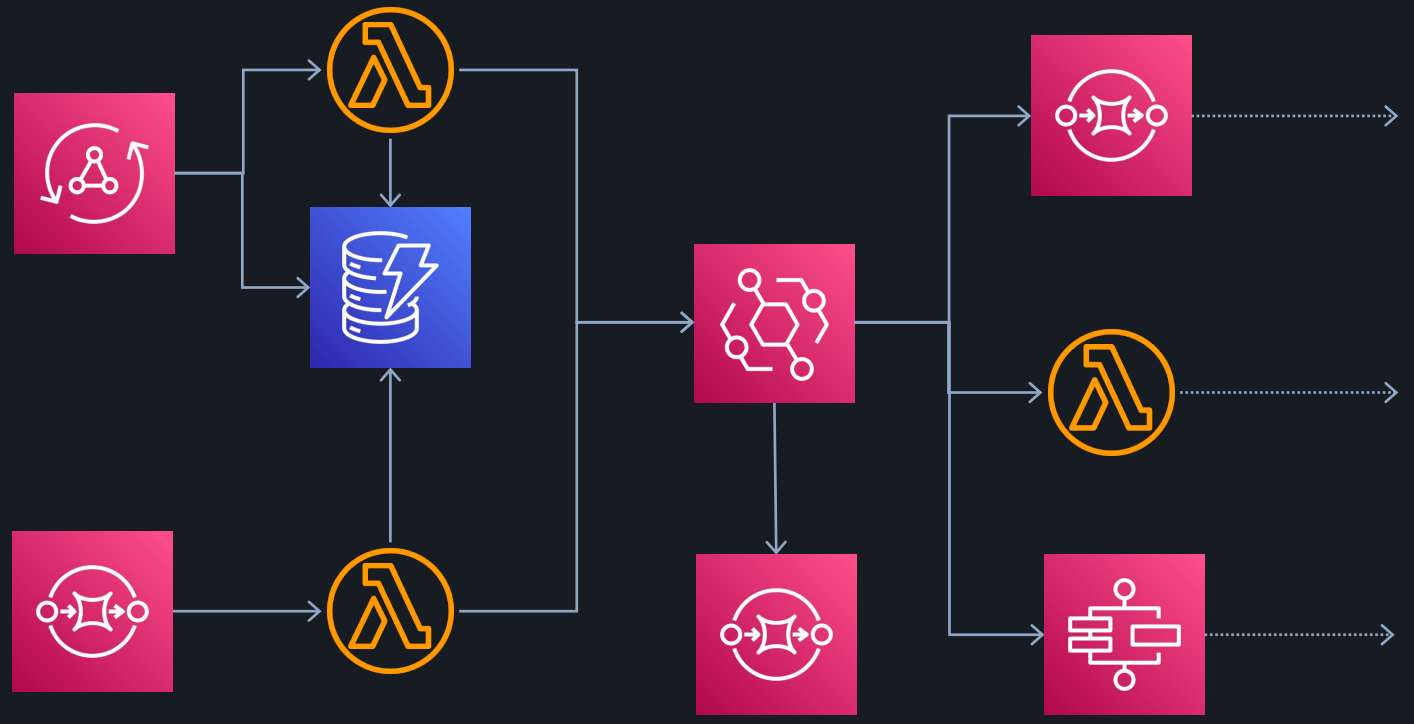
- Functions-as-a-Service (FaaS): AWS Lambda
- API Management: AWS API Gateway & AppSync
- Event Delivery: AWS SNS, EventBridge, Kinesis…
- Message Queues: AWS SQS
- Databases: AWS DynamoDB, QLDB, Aurora Serverless…
- Object Storage: AWS S3
- Orchestration: AWS Step Functions
- Config Management: Parameter Store, Secrets Manager…
- Serverless Containers: AWS Fargate

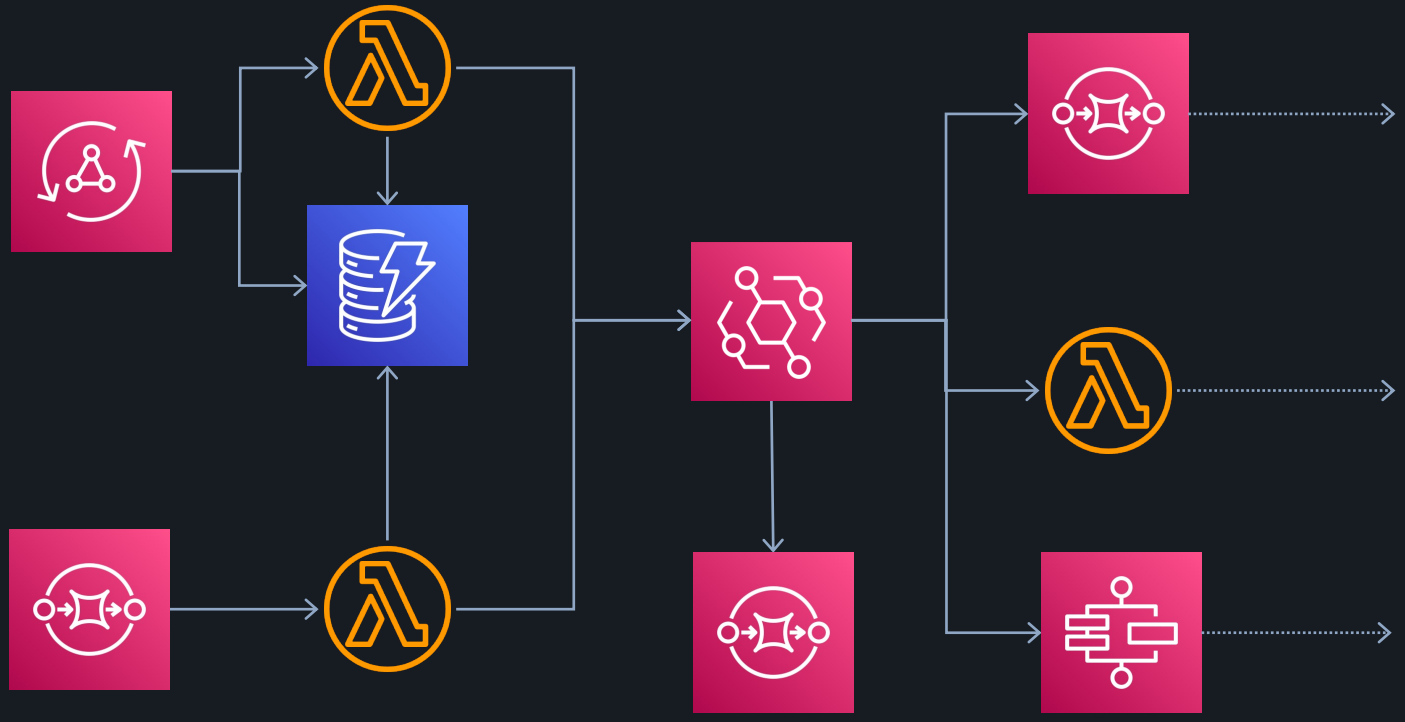
Serverless

- Functions-as-a-Service (FaaS): AWS Lambda
- API Management: AWS API Gateway & AppSync
- Event Delivery: AWS SNS, EventBridge, Kinesis…
- Message Queues: AWS SQS
- Databases: AWS DynamoDB, QLDB, Aurora Serverless…
- Object Storage: AWS S3
- Orchestration: AWS Step Functions
- Config Management: Parameter Store, Secrets Manager…
- Serverless Containers: AWS Fargate

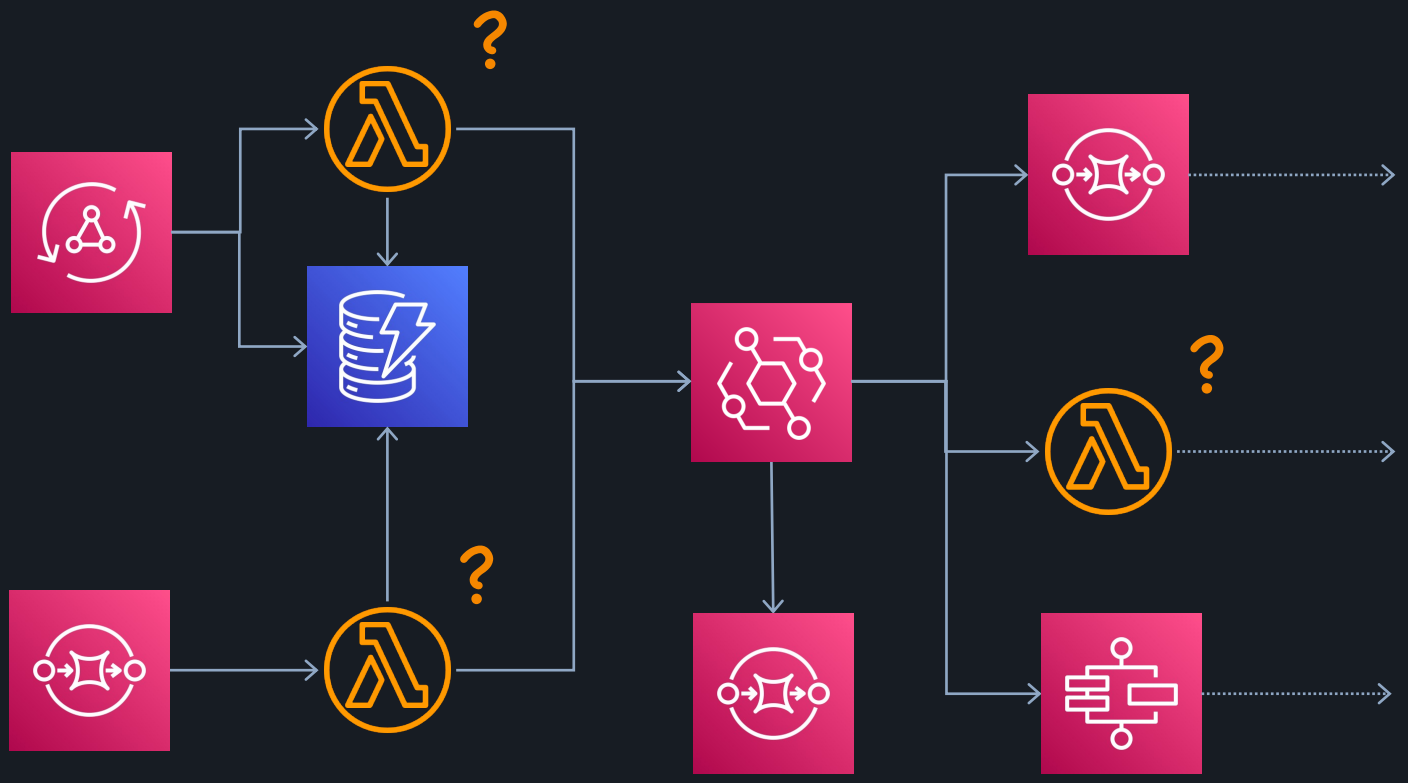
… i.e. any AWS Service that doesn't require self-managing EC2!

Serverless Application...

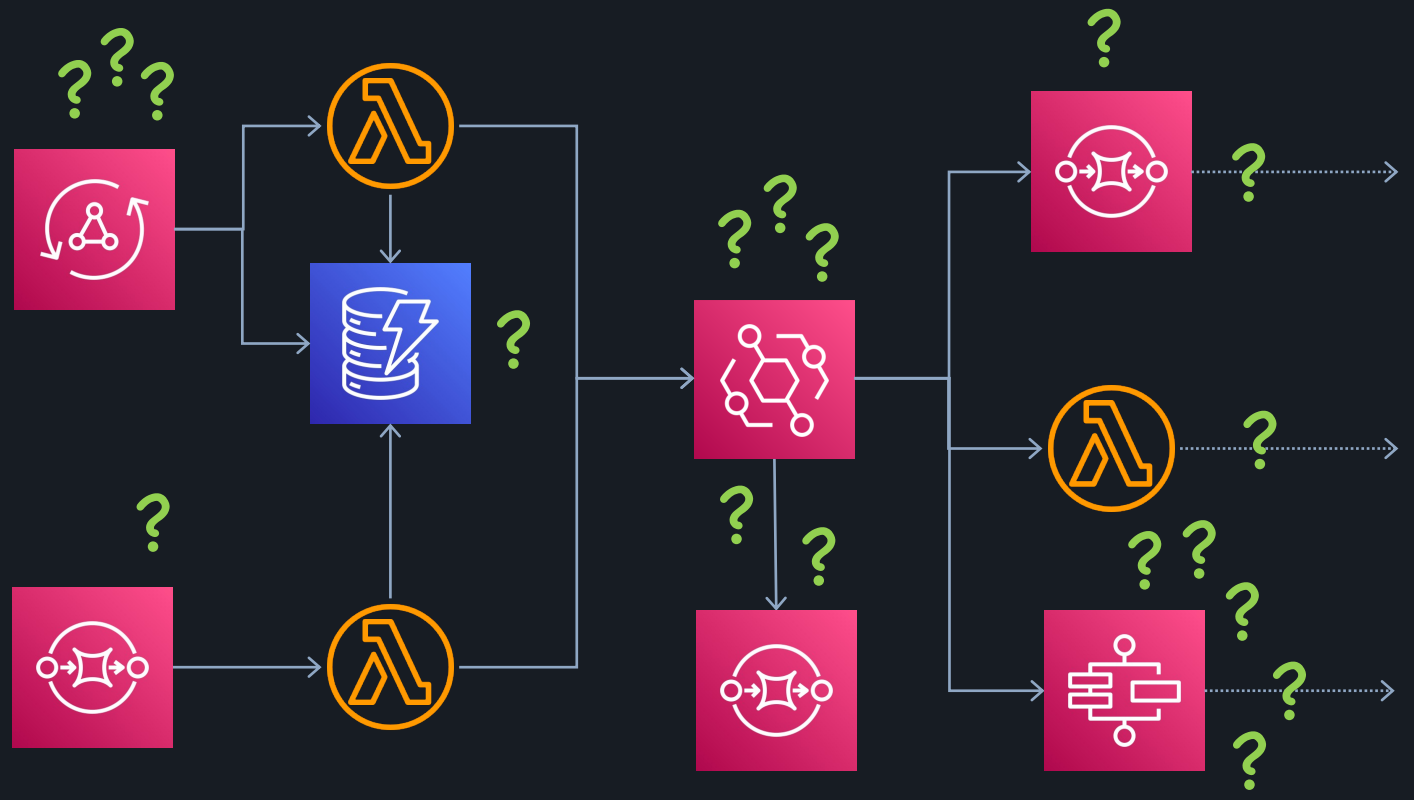




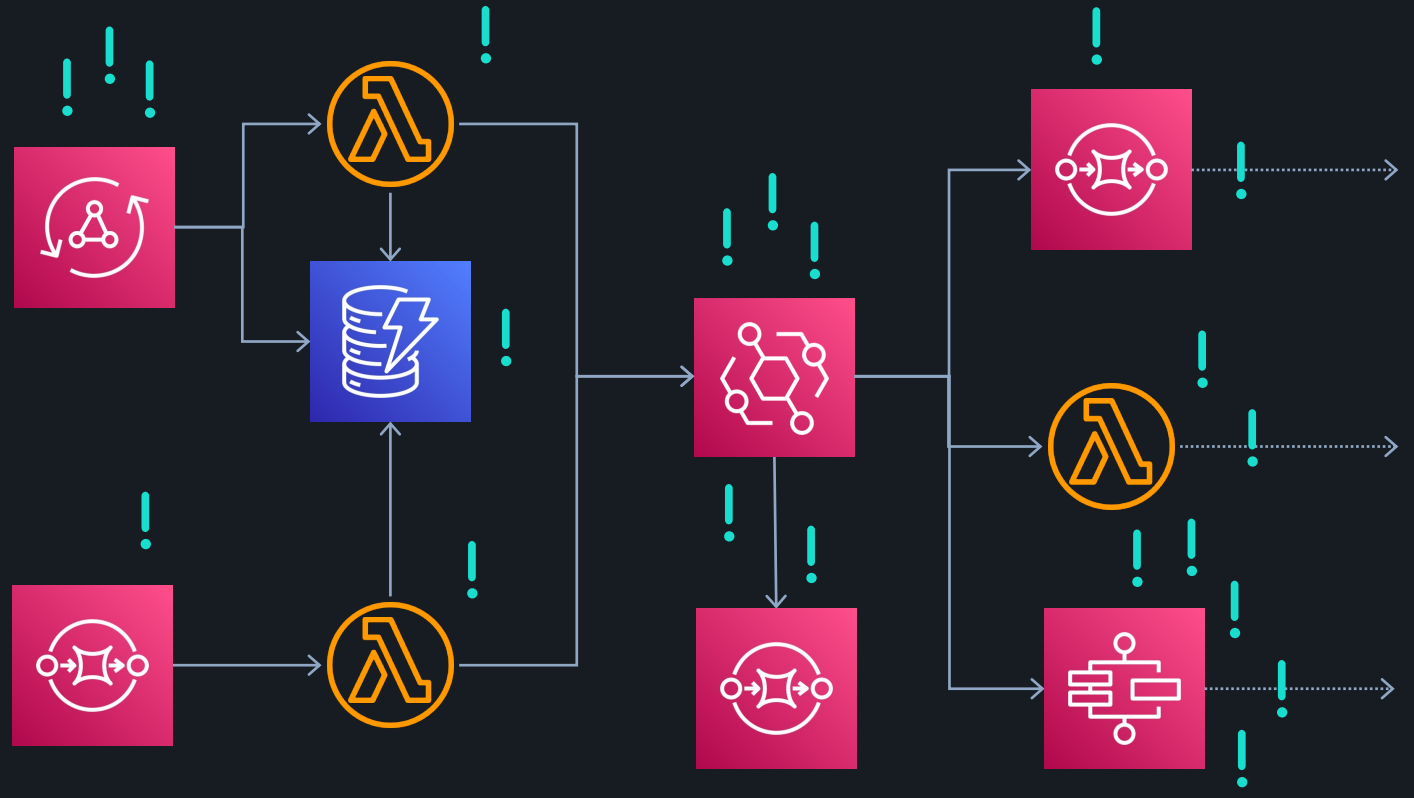
Where's your *business logic?* In Lambda functions?



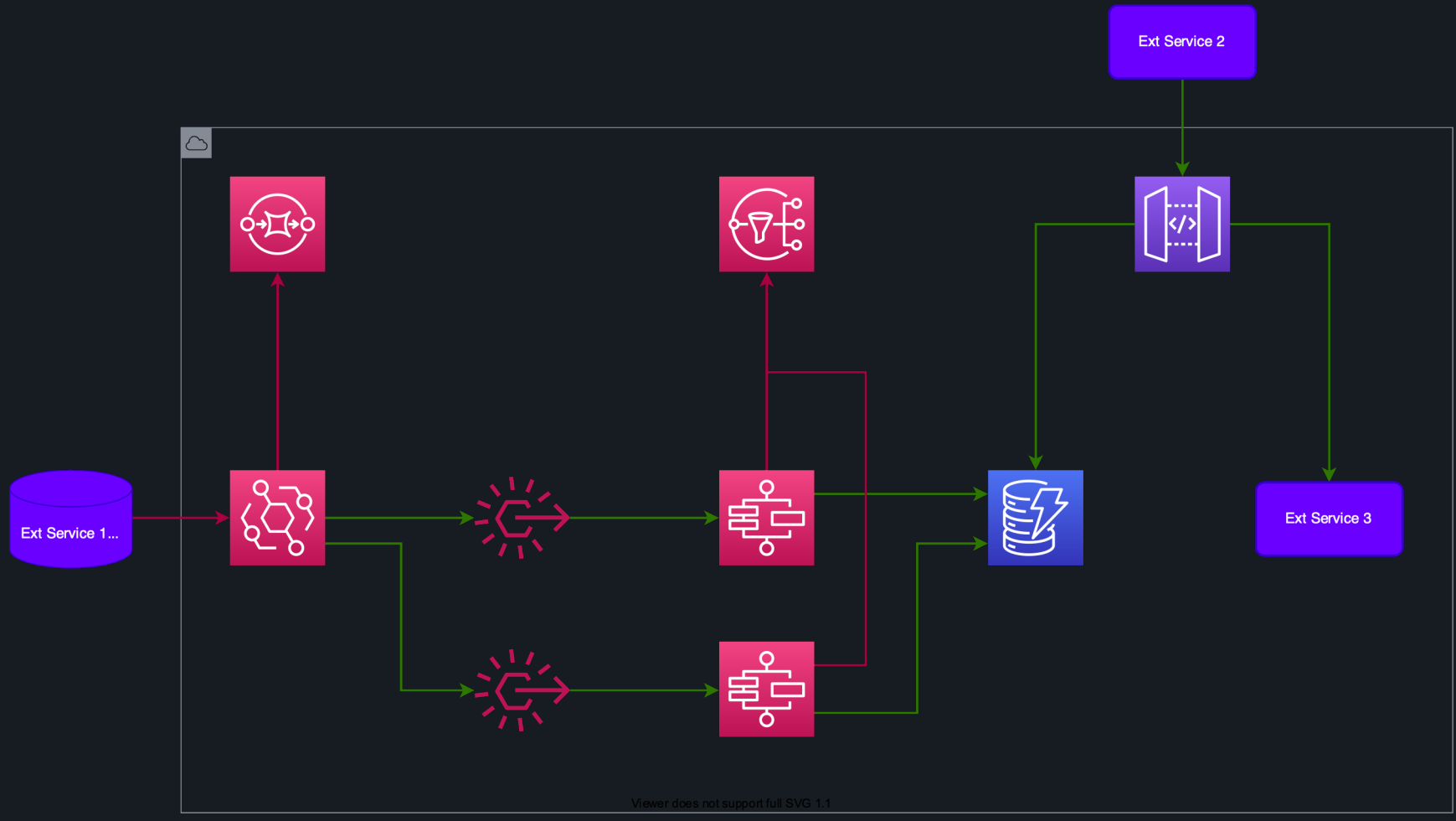
Where's your *business logic?* In AWS Services?



Your *infrastructure* is your *business logic!*



You could even go “functionless”…



Ben Kehoe @ben11kehoe · Nov 16

This is an important point when we talk about "just write business logic" and people say "all the infrastructure should be taken care of for you". The properties of your system are business logic, and they have real implications that involve ownership.

> 
> **Ben Kehoe** @ben11kehoe · Nov 16
>
> The presence or absence of an SQS queue in your architecture is business logic.

1 1 23

Ben Kehoe
@ben11kehoe

This is why I don't like the term "infrastructure", because it's not "infra"—below—like it used to be. It *is* your application. I prefer to talk about "resources" and your "resource graph".

11:51 PM · Nov 16, 2021 · Twitter Web App

Your *infrastructure* is your *business logic!*

👉 Treat it as such!

Your *infrastructure* is your *business logic!*

👉 **Treat it as such!**

… you probably wouldn’t write your business logic as (tens of) thousands of lines of YAML…?

Your *infrastructure* is your *business logic!*

👉 Treat it as you would “any other code”!

- Use best practices & good software development patterns
- Utilize dependency management tools
- Let type systems & editor/IDE intellisense support you
- Use static analysis tools (linters)
- Write tests
- Perform Code Reviews
- Define your domain-specific rules/patterns

Unit testing CDK code

@aws-cdk/assertions
(screenshots still from @aws-cdk/assert)

```
import { Stack } from '@aws-cdk/core';
import '@aws-cdk/assert/jest';
import { stringLike } from '@aws-cdk/assert';
import { DeadLetterQueue } from './demo-dlq';

test('dlq creates an alarm', () => {
  const stack = new Stack();

  new DeadLetterQueue(stack, 'DLQ');

  expect(stack).toHaveResource('AWS::CloudWatch::Alarm', {
    MetricName: "ApproximateNumberOfMessagesVisible",
    Namespace: "AWS/SQS",
    Dimensions: [
      {
        Name: "QueueName",
        Value: { "Fn::GetAtt": [ stringLike("DLQ*"), "QueueName" ] }
      }
    ],
  });
});

test('dlq has maximum retention period', () => {
  const stack = new Stack();

  new DeadLetterQueue(stack, 'DLQ');

  expect(stack).toHaveResource('AWS::SQS::Queue', {
    MessageRetentionPeriod: 1209600
  });
});
```

@aws-cdk/assertions
(screenshots still from @aws-cdk/assert)

Expect DLQ to define a CloudWatch Alarm

```
import { Stack } from '@aws-cdk/core';
import '@aws-cdk/assert/jest';
import { stringLike } from '@aws-cdk/assert';
import { DeadLetterQueue } from './demo-dlq';

test('dlq creates an alarm', () => {
  const stack = new Stack();

  new DeadLetterQueue(stack, 'DLQ');

  expect(stack).toHaveResource('AWS::CloudWatch::Alarm', {
    MetricName: "ApproximateNumberOfMessagesVisible",
    Namespace: "AWS/SQS",
    Dimensions: [
      {
        Name: "QueueName",
        Value: { "Fn::GetAtt": [ stringLike("DLQ*"), "QueueName" ] }
      }
    ],
  });
});

test('dlq has maximum retention period', () => {
  const stack = new Stack();

  new DeadLetterQueue(stack, 'DLQ');

  expect(stack).toHaveResource('AWS::SQS::Queue', {
    MessageRetentionPeriod: 1209600
  });
});
```

https://aws.amazon.com/blogs/developer/testing-cdk-applications-in-any-language

AWS Developer Tools Blog

Testing CDK Applications in Any Language

by Thomas Ross | on 09 NOV 2021 | in Announcements, AWS Cloud Development Kit, Intermediate (200), Java, JavaScript, Open Source, Python | Permalink | Comments | Share

The AWS Cloud Development Kit (AWS CDK) is an open source software development framework to define your cloud application resources using familiar programming languages. Because the AWS CDK enables you to define your infrastructure in regular programming languages, you can also write automated unit tests for your infrastructure code, just like you do for your application code. Testing is an essential element to highly effective DevOps practices, and testing your infrastructure code provides benefits such as ensuring that you will create exactly the resources you expect in the AWS cloud and helping to prevent regressions from being introduced to your infrastructure.

Today, I am happy to announce the assertions module for the AWS Cloud Development Kit, a set of APIs designed to help you write unit tests against your CDK applications, with a focus on CloudFormation templates.

Cross-Language Support

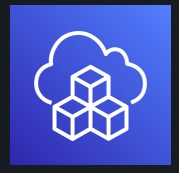
A previous AWS blog post explains how to write tests for your infrastructure constructs using the `assert` module, which is available only for JavaScript and TypeScript.

Similar to the `assert` module, the new CDK `assertions` module provides a robust set of APIs to precisely verify the CloudFormation templates synthesized by your CDK app. Additionally, the `assertions` module is available for every language supported by the CDK.

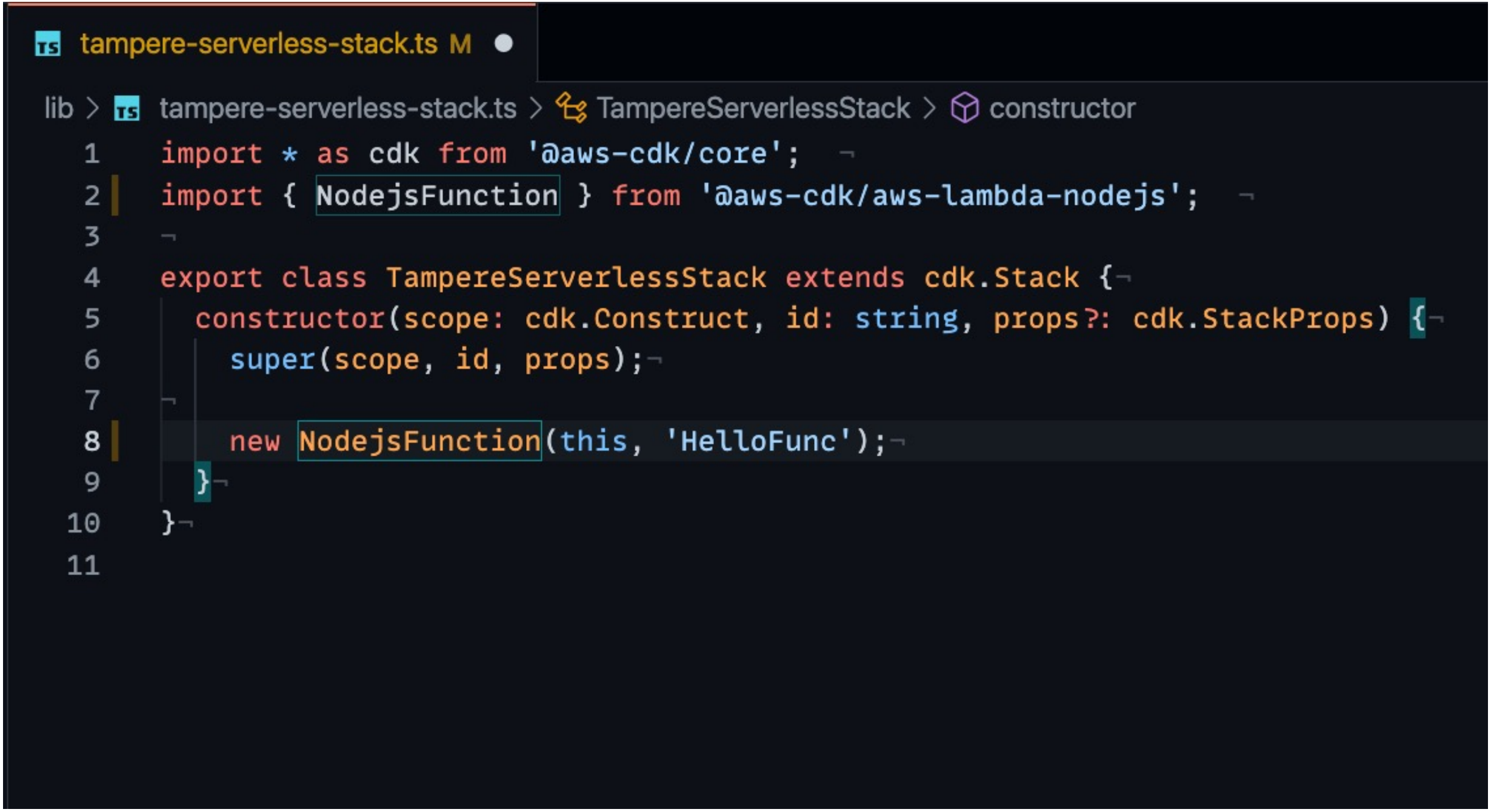
While the new assertions module supports every language that is supported by the CDK, this snippets in this article will be written in Python. However, the full source code for these examples is available on GitHub, and contains equivalent code written in TypeScript, Java, and Python.

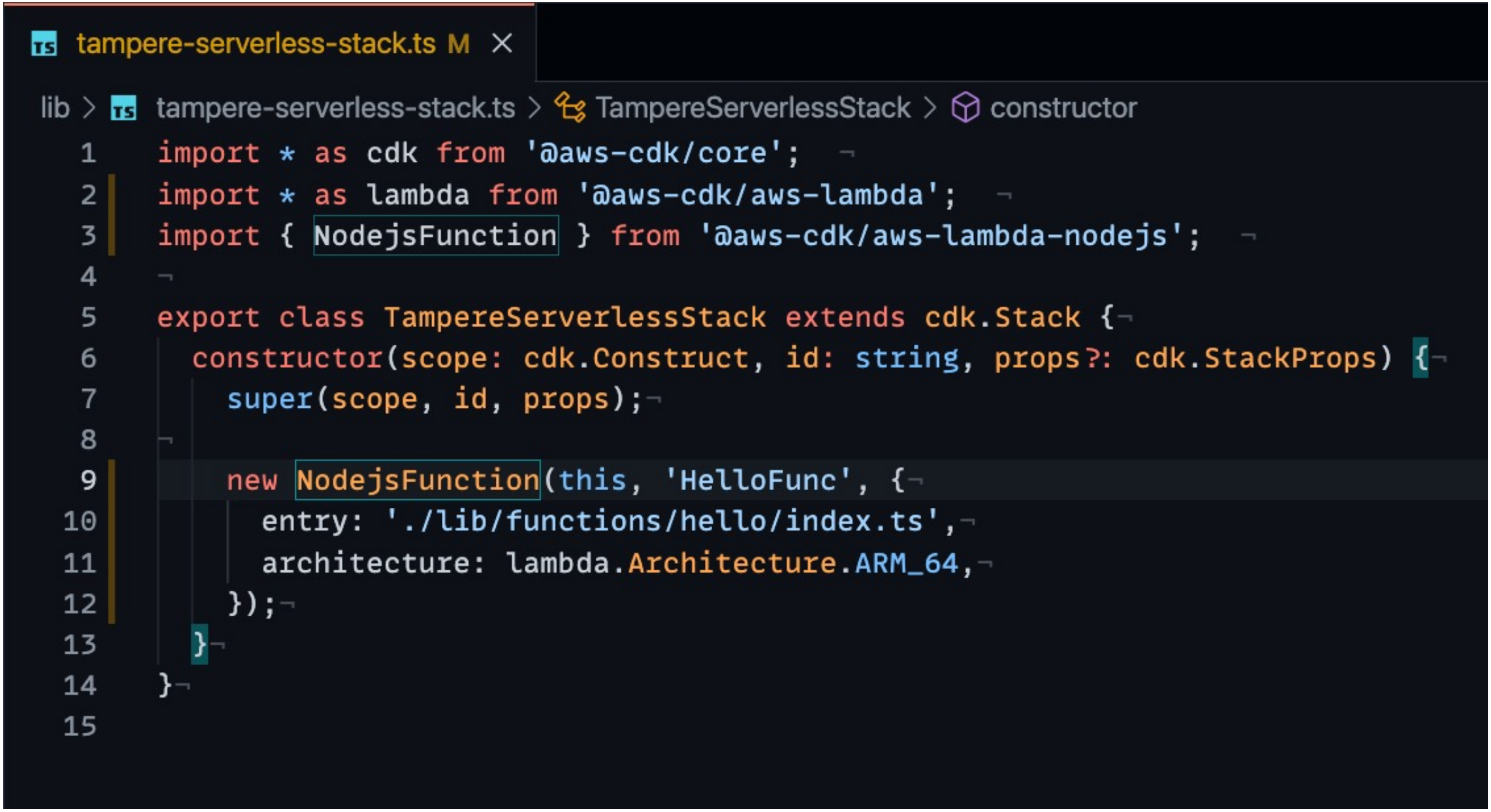
AWS Lambda

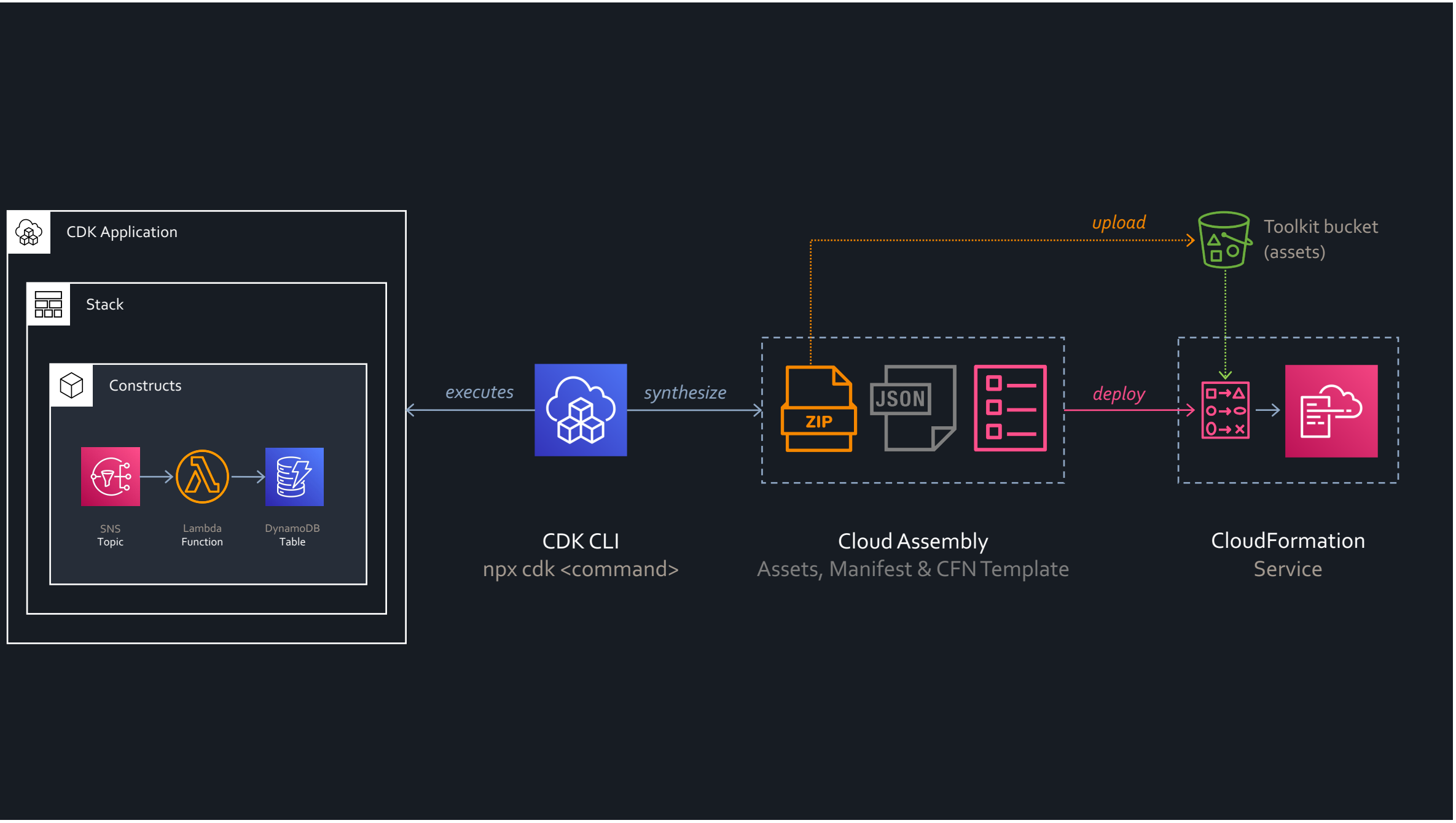
+

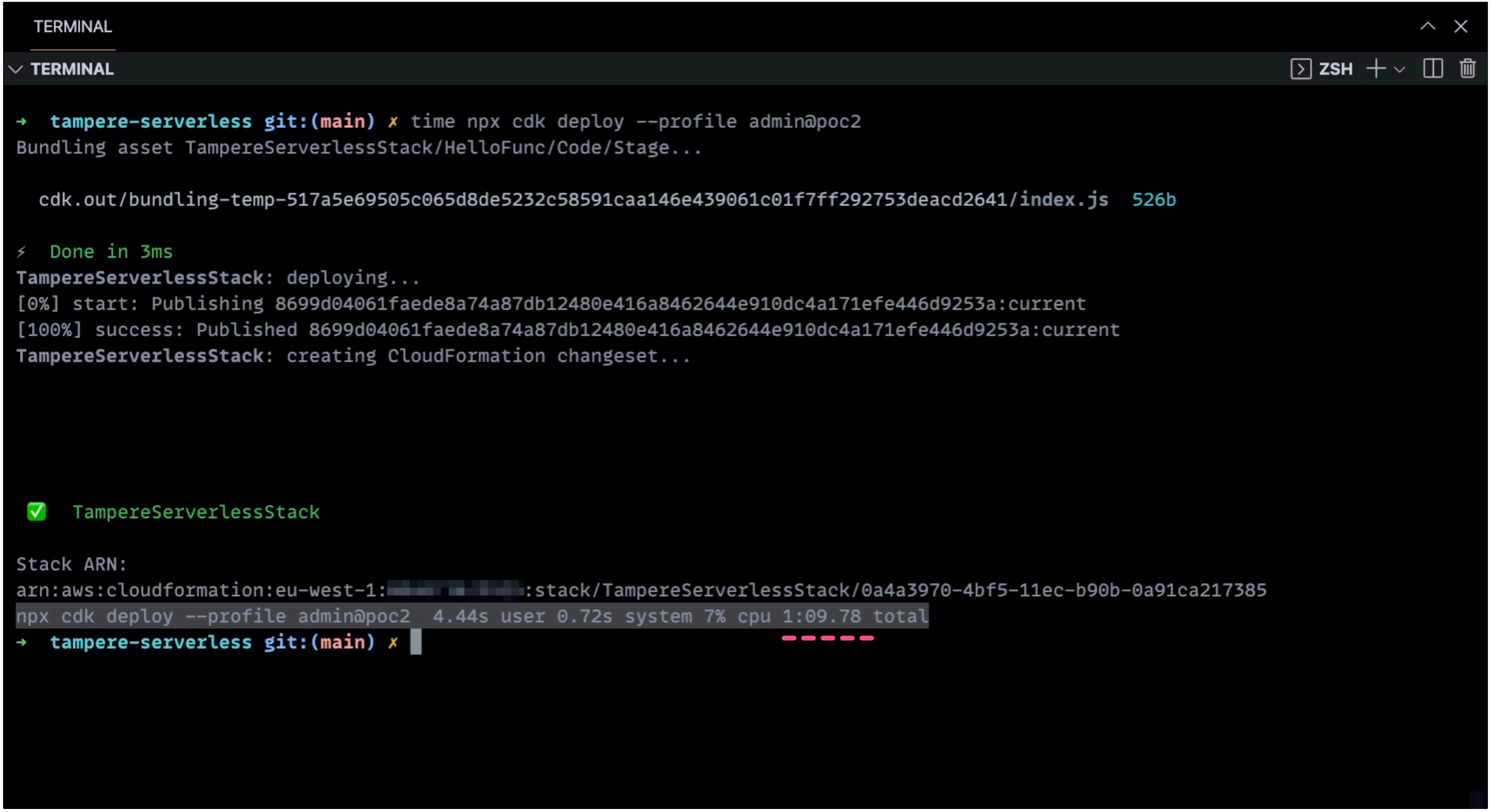


AWS CDK


tampere-serverless-stack.ts M
lib > tampere-serverless-stack.ts > TampereServerlessStack > constructor
1 import * as cdk from '@aws-cdk/core';
2 import { NodejsFunction } from '@aws-cdk/aws-lambda-nodejs';
3
4 export class TampereServerlessStack extends cdk.Stack {
5 constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
6 super(scope, id, props);
7
8 new NodejsFunction(this, 'HelloFunc');
9 }
10 }
11


tampere-serverless-stack.ts M
lib > tampere-serverless-stack.ts > TampereServerlessStack > constructor
1 import * as cdk from '@aws-cdk/core';
2 import * as lambda from '@aws-cdk/aws-lambda';
3 import { NodejsFunction } from '@aws-cdk/aws-lambda-nodejs';
4
5 export class TampereServerlessStack extends cdk.Stack {
6 constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
7 super(scope, id, props);
8
9 new NodejsFunction(this, 'HelloFunc', {
10 entry: './lib/functions/hello/index.ts',
11 architecture: lambda.Architecture.ARM_64,
12 });
13 }
14 }
15


CDK Application
Stack
Constructs
SNS Topic
Lambda Function
DynamoDB Table
executes
synthesize
upload
Toolkit bucket (assets)
ZIP
JSON
deploy
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template
CloudFormation
Service


TERMINAL
TERMINAL
ZSH
→ tampere-serverless git:(main) ✗ time npx cdk deploy --profile admin@poc2
Bundling asset TampereServerlessStack/HelloFunc/Code/Stage...
cdk.out/bundling-temp-517a5e69505c065d8de5232c58591caa146e439061c01f7ff292753deacd2641/index.js 526b
⚡ Done in 3ms
TampereServerlessStack: deploying...
[0%] start: Publishing 8699d04061faede8a74a87db12480e416a8462644e910dc4a171efe446d9253a:current
[100%] success: Published 8699d04061faede8a74a87db12480e416a8462644e910dc4a171efe446d9253a:current
TampereServerlessStack: creating CloudFormation changeset...
✅ TampereServerlessStack
Stack ARN:
arn:aws:cloudformation:eu-west-1:[redacted]:stack/TampereServerlessStack/0a4a3970-4bf5-11ec-b90b-0a91ca217385
npx cdk deploy --profile admin@poc2 4.44s user 0.72s system 7% cpu 1:09.78 total
→ tampere-serverless git:(main) ✗

https://aws.amazon.com/about-aws/whats-new/2021/10/aws-cdk-releases-hotswap-rollback-control/

AWS CDK releases v1.121.0 - v1.125.0 with features for faster development cycles using hotswap deployments and rollback control

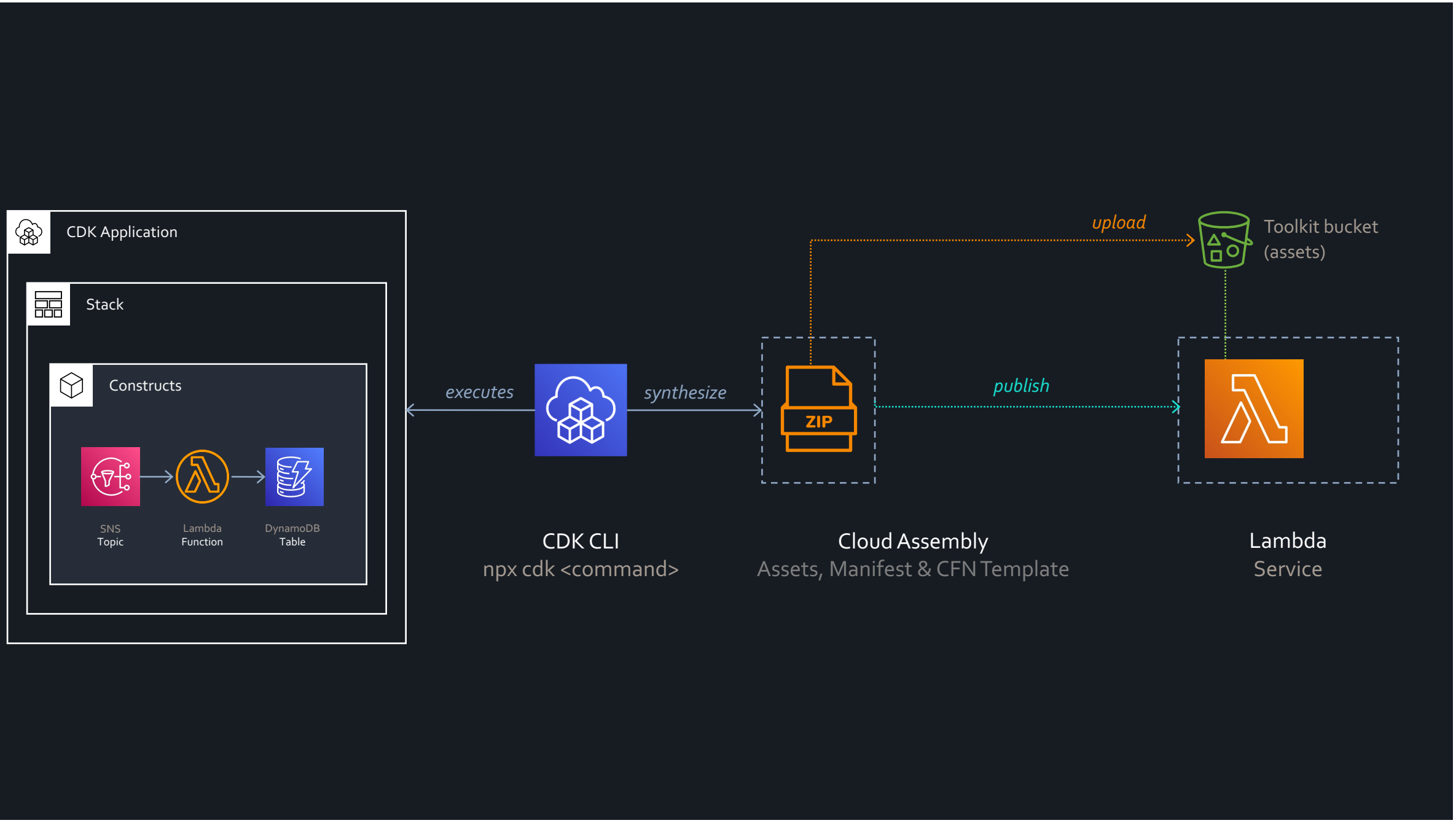
Posted On: Oct 12, 2021

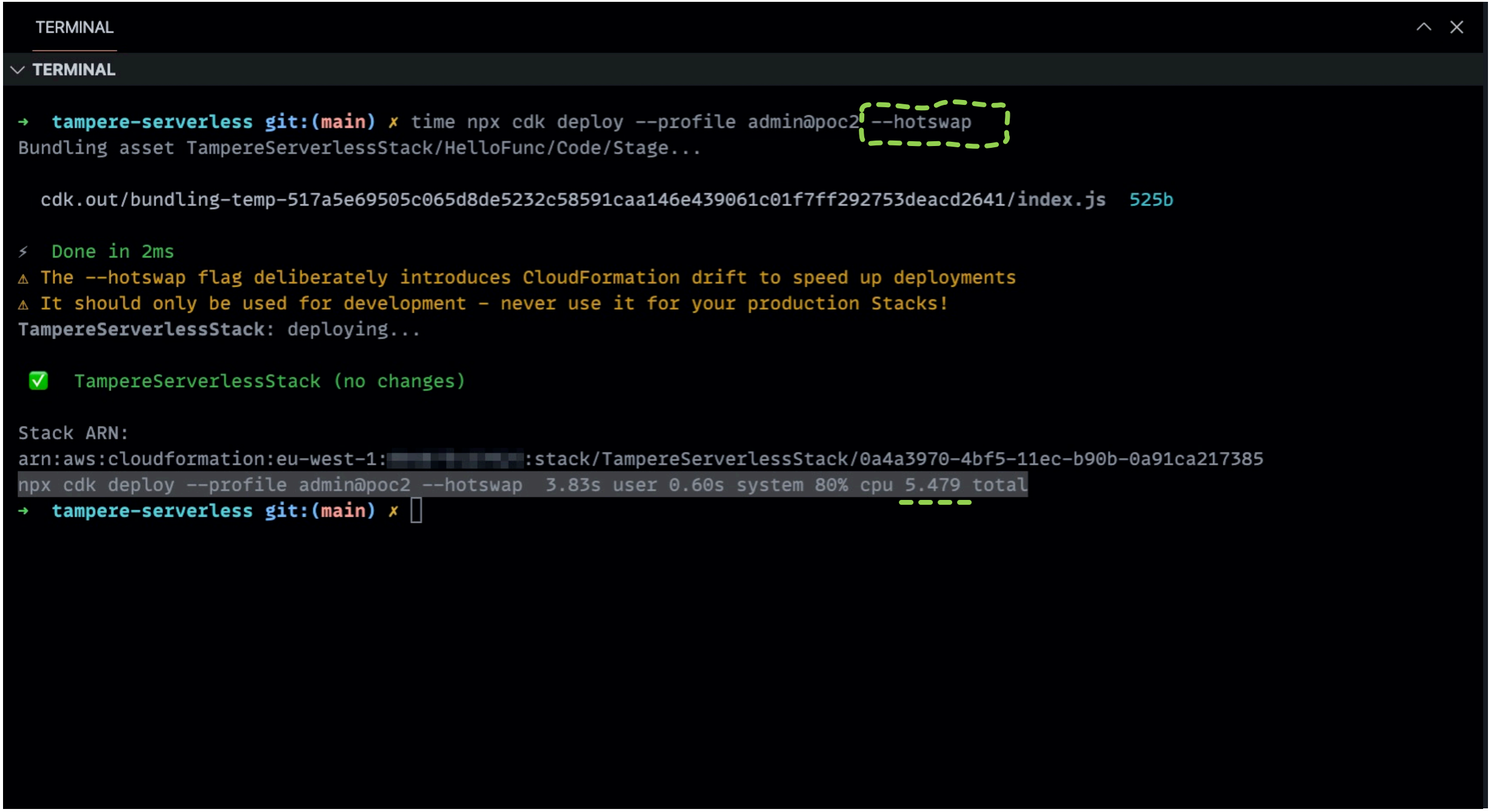
During September, 2021, 5 new versions of the AWS Cloud Development Kit (CDK) for JavaScript, TypeScript, Java, Python, .NET and Go were released (v1.121.0 through v.125.0). With these releases, the CDK CLI now has support for hotswap deployments for faster inner-loop development iterations on the application code in your CDK project. Hotswap initially supports AWS Lambda handler code, but support is planned for additional resource types and a "watch" mode which continually watches for changes and deploys any updates. Additionally, users can preserve successfully provisioned resources by disabling automatic stack rollbacks, further reducing deployment and iteration time. These releases also resolve 21 issues and introduce 40 new features that span over 30 different modules across the library. Many of these changes were contributed by the developer community.

The AWS CDK is a software development framework for defining cloud applications using familiar programming languages. The AWS CDK simplifies cloud development on AWS by hiding infrastructure and application complexity behind intent-based, object-oriented APIs for each AWS service.

To get started, see the following resources:

- Read the full release notes for 1.121.0, 1.122.0, 1.123.0, 1.124.0, 1.125.0
- Get started with the AWS CDK in all supported languages by taking CDK Workshop.
- Read our Developer Guide and API Reference.
- Find useful constructs published by AWS, partners and the community in Construct Hub.
- Connect with the community in the cdk.dev Slack workspace.
- Follow our Contribution Guide to learn how to contribute fixes and features to the CDK.


CDK Application
Stack
Constructs
SNS Topic
Lambda Function
DynamoDB Table
executes
synthesize
upload
Toolkit bucket (assets)
ZIP
publish
CDK CLI
npx cdk <command>
Cloud Assembly
Assets, Manifest & CFN Template
Lambda
Service


TERMINAL
TERMINAL
→ tampere-serverless git:(main) ✗ time npx cdk deploy --profile admin@poc2 --hotswap
Bundling asset TampereServerlessStack/HelloFunc/Code/Stage...
cdk.out/bundling-temp-517a5e69505c065d8de5232c58591caa146e439061c01f7ff292753deacd2641/index.js 525b
⚡ Done in 2ms
⚠ The --hotswap flag deliberately introduces CloudFormation drift to speed up deployments
⚠ It should only be used for development - never use it for your production Stacks!
TampereServerlessStack: deploying...
✅ TampereServerlessStack (no changes)
Stack ARN:
arn:aws:cloudformation:eu-west-1:[redacted]:stack/TampereServerlessStack/0a4a3970-4bf5-11ec-b90b-0a91ca217385
npx cdk deploy --profile admin@poc2 --hotswap 3.83s user 0.60s system 80% cpu 5.479 total
→ tampere-serverless git:(main) ✗

https://aws.amazon.com/blogs/compute/better-together-aws-sam-and-aws-cdk/

sam local invoke

The first test is to invoke the PutTranslationFunction locally. I created a mock event for each Lambda function. These events are located in the *events* directory. Run this command passing in the mock event and the local environment file:

Bash

```
sam-beta-cdk local invoke CdkDayStack/PutTranslationFunction -e events/putTranslation.jsor
```

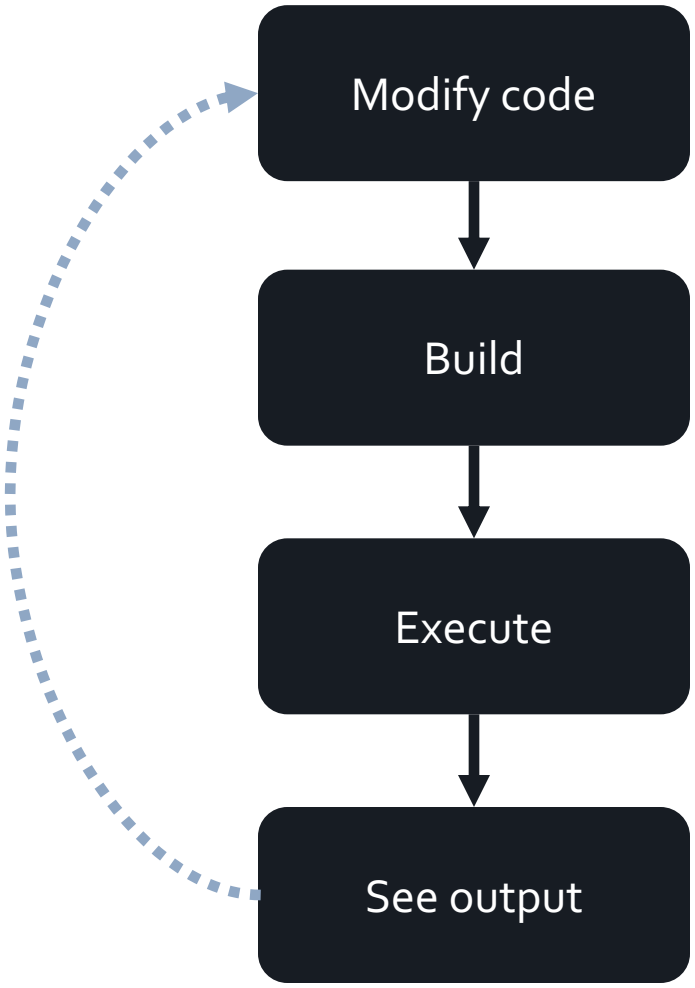
```
sam-beta-cdk local invoke CdkDayStack/PutTranslationFunction -e \
events/putTranslation.json -n locals.json

Invoking app.handler (nodejs14.x)
Skip pulling image and use local one: amazon/aws-sam-cli-emulation-image-
nodejs14.x:rapid-1.22.0.

Mounting /home/ec2-user/environment/cdk-day/.aws-
sam/build/asset.363846a1c45bf57517f04c26793190b7f2cf1183336a98ec07140326c31d747
as /var/task:ro,delegated inside runtime container
END RequestId: b5747b2b-84db-495b-b3b8-750e7c0f8062
REPORT RequestId: b5747b2b-84db-495b-b3b8-750e7c0f8062  Init Duration: 1.81 ms
Duration: 491.13 ms     Billed Duration: 500 ms      Memory Size: 128 MB
Max Memory Used: 128 MB

{
    "id":"12345",
    "Items":[
        {"language":"hi","translation":"यह मेरा पाठ है","id":"12345"},
        {"language":"de","translation":"Das ist mein Text","id":"12345"},
        {"language":"fr","translation":"C'est mon texte","id":"12345"},
        {"language":"en","translation":"This is my text","id":"12345"}
    ]
}
```

“Traditional” Local Lambda Development

i.e. “*Console.log Driven Development*”

Often seen with tools such as:

- SAM CLI (local invoke)
- lambci/docker-lambda



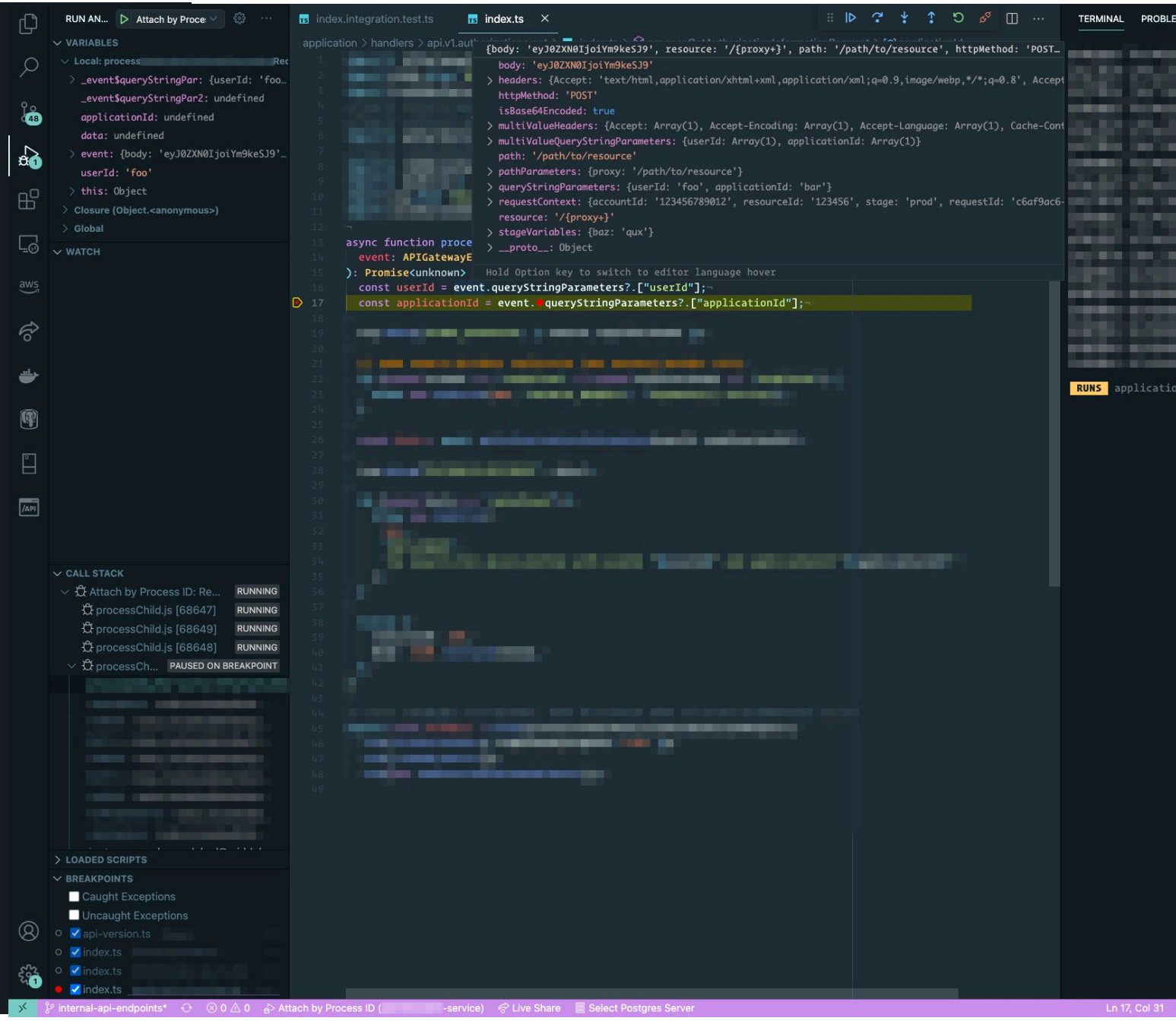
Lambda Test Driven Development

Fast Feedback loop

```
npx jest -t "Lambda.MyFunction" --watch
```

Debugger

Attach a real debugger for your test runs
to avoid *Console.log Driven Development*



Mock AWS services

https://bit.ly/3lAo4ZJ

When developing locally – i.e. unit testing locally.

- ***Fast feedback cycle***
- ***Test error scenarios with downstream services***
  for example, corrupt data from database or timeout from 3rd party API call

Mocking AWS with Jest (and TypeScript)

#aws #typescript #jest #testing

Matt Morgan Sep 1, 2020 · *Updated on Mar 3* · 13 min read

This seems like a very punny subject, mocking the world's leading cloud provider. I tried to think of a few chuckles, but then I got to thinking about writing blogposts for free to promote the projects of companies that are worth hundreds of billions of dollars, and maybe the joke is just on me?

Anyway, no need to think on that one too much. I like working with these tools a lot and I like sharing the ways I've found to use them together. Some readers may think Jest is a lot like some of the other JavaScript/NodeJS testing frameworks, but it has a few features that I really think set it apart from anything else I've used. One of them is that it ships with a very powerful mocking capability. That is the subject of this article. The other features I like about Jest are snapshots and test tables, but they will need to be covered another time.

Table of Contents

Building assets with Docker

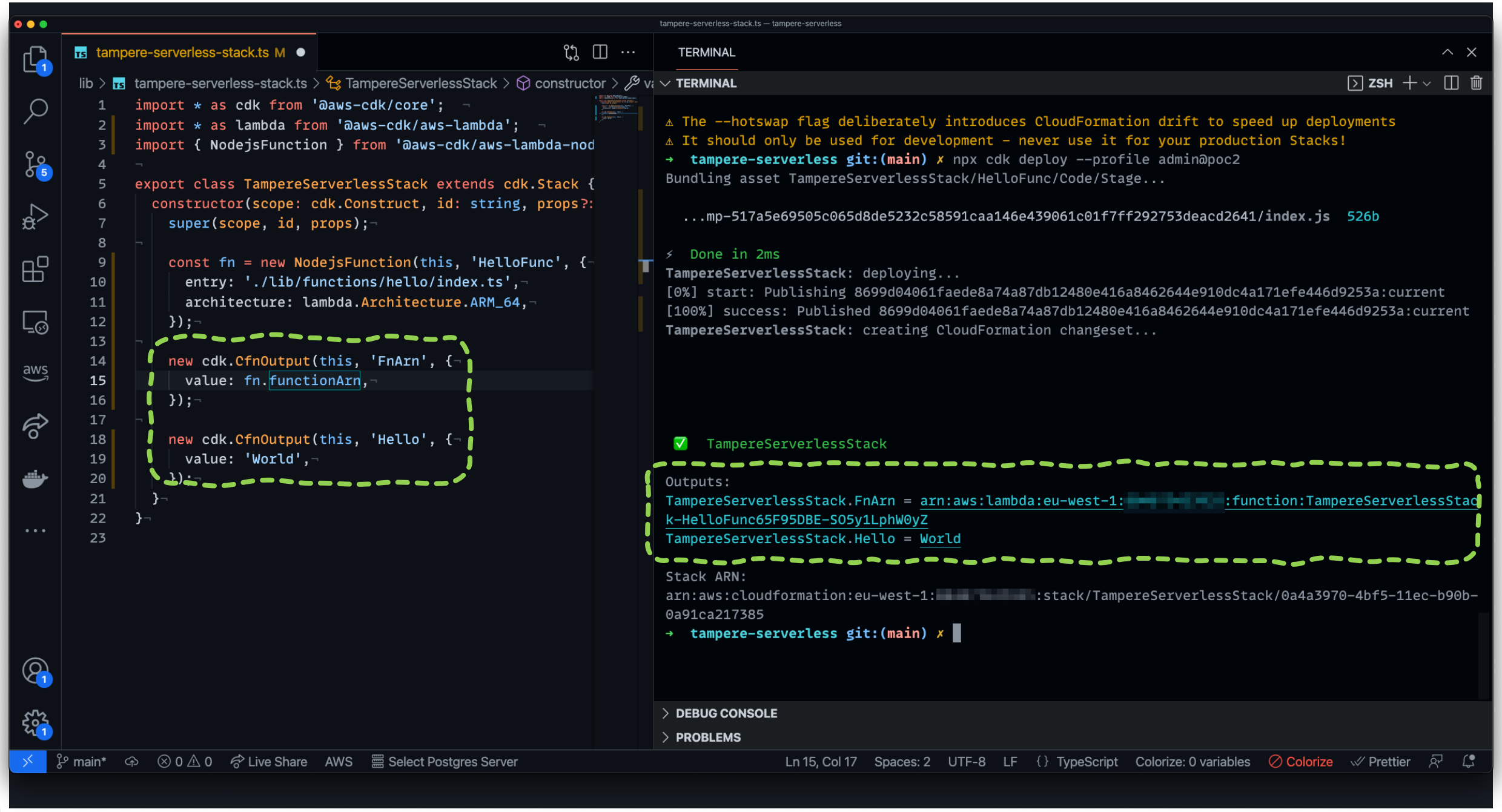
https://aripalo.com/blog/2020
/building-lambda-functions-inside-docker-containers-with-cdk/

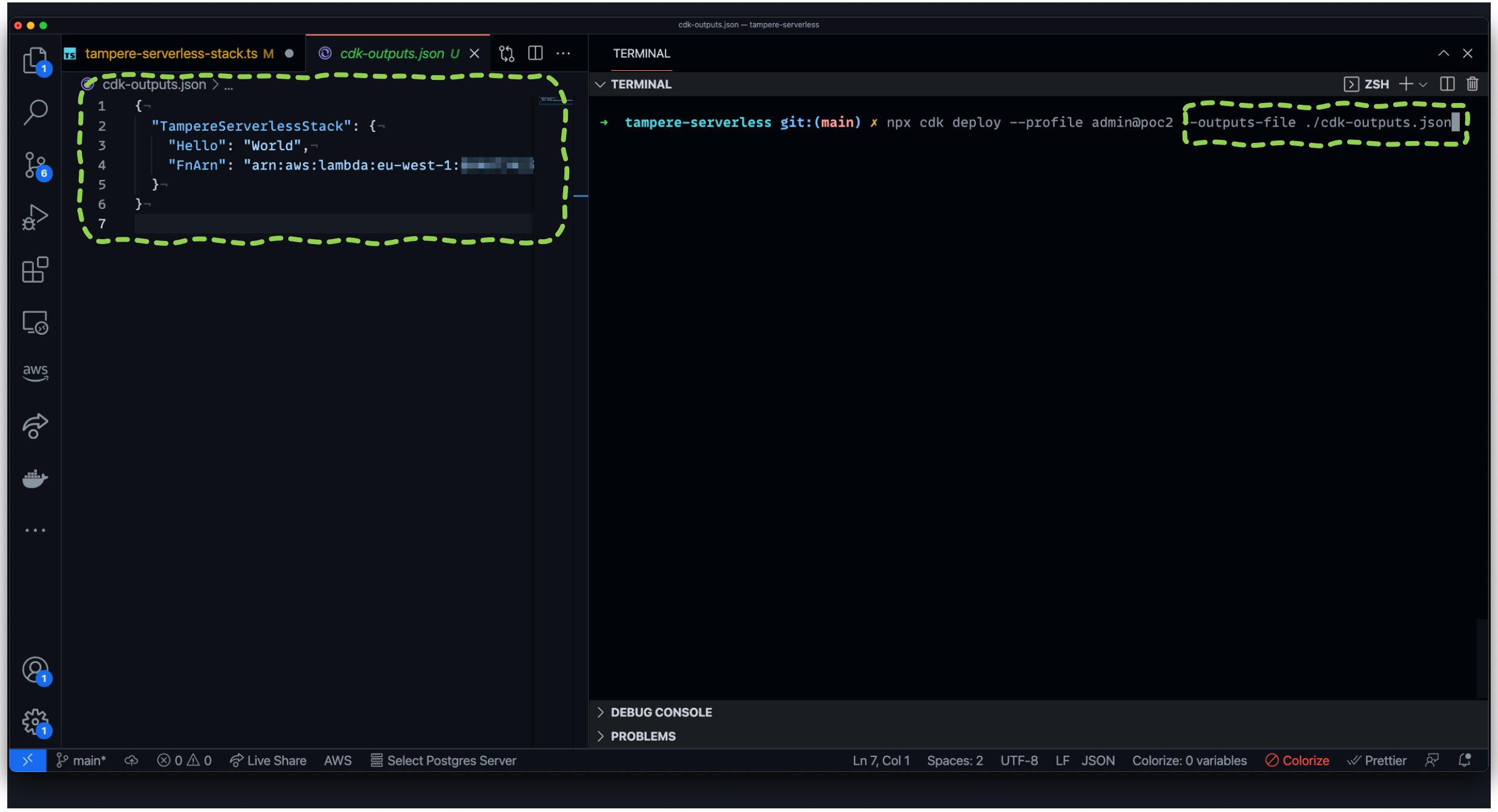
```
new lambda.Function(this, "MyFunc", {
  runtime: lambda.Runtime.GO_1_X,
  handler: "bin/main",
  code: lambda.Code.fromAsset("path/to/my-func", {
    bundling: {
      // Speficy Docker image:
      image: lambda.Runtime.GO_1_X.bundlingDockerImage,
      // The build command to be executed within the Docker container:
      command: [
        'bash', '-c', [
          `cd /asset-input`,
          `go build -o bin/main`,
          `cp /asset-input/bin/main /asset-output/bin`,
        ].join(' && ')
      ],
      user: 'root',
    },
    // Glob(s) to exclude from deployment package:
    // In most cases with Go, only the binary is needed (unless you depend on external files)
    exclude: ["!bin"],
  }),
});
```

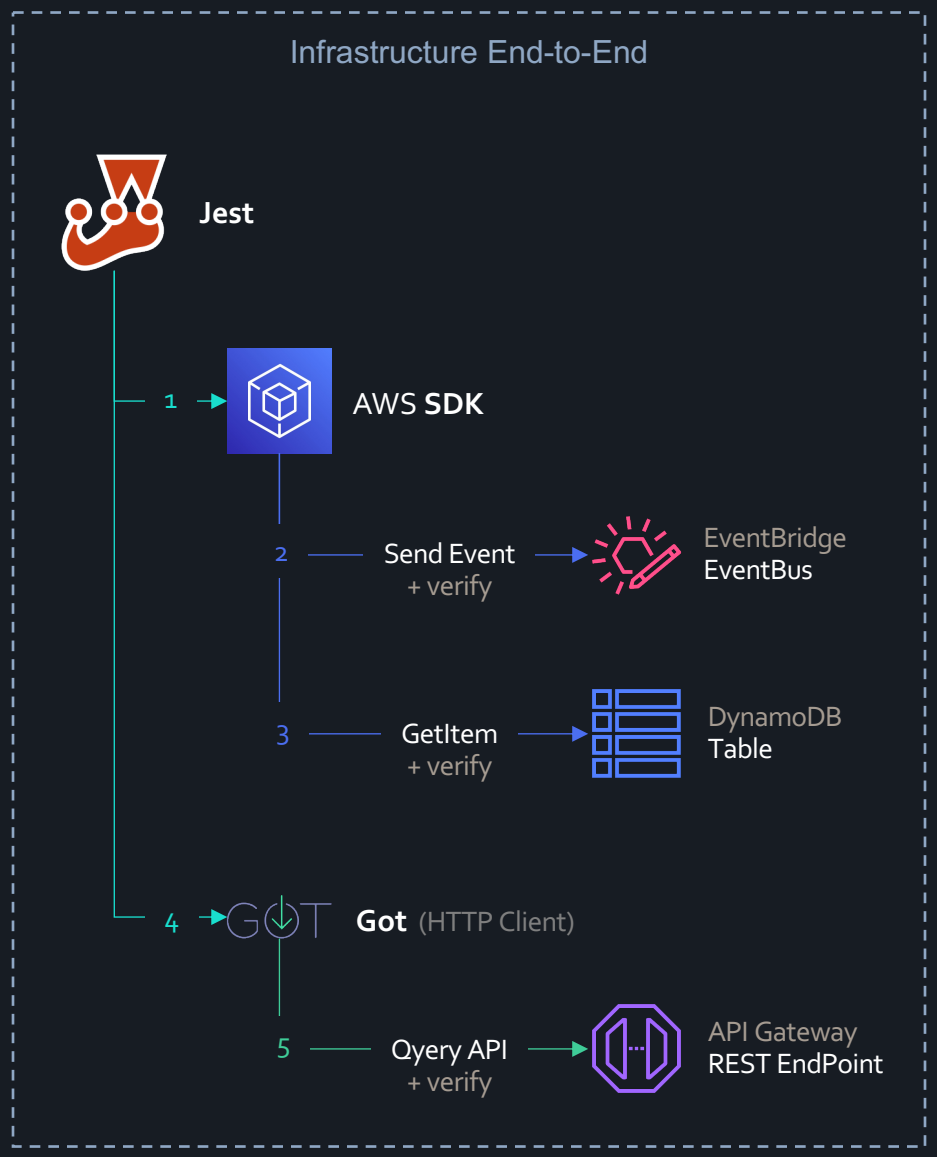
Unit tests & mocking…? What about testing real systems?

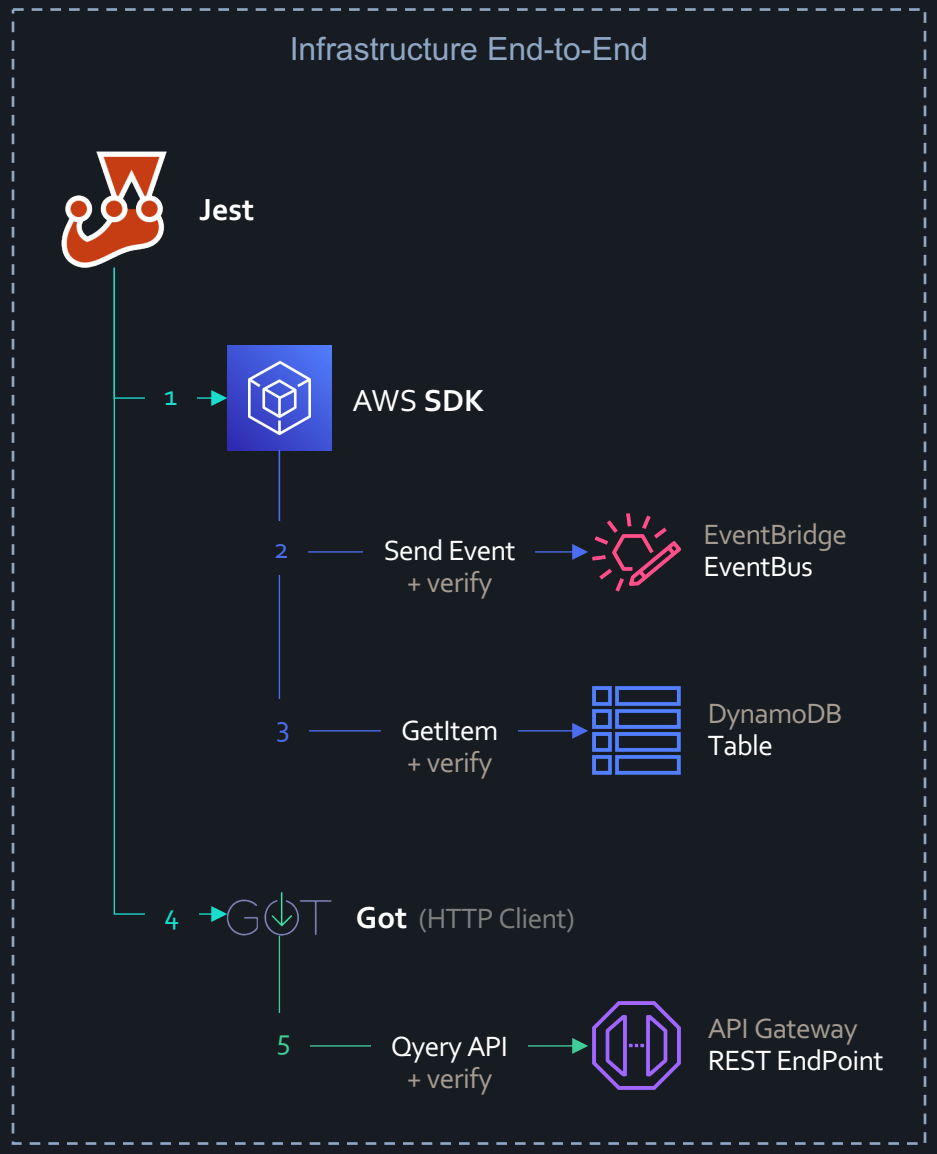
Unit tests & mocking…? What about testing real systems?

Integration / End-to-End Testing


tampere-serverless-stack.ts — tampere-serverless
tampere-serverless-stack.ts M
lib > tampere-serverless-stack.ts > TampereServerlessStack > constructor
import * as cdk from '@aws-cdk/core';
import * as lambda from '@aws-cdk/aws-lambda';
import { NodejsFunction } from '@aws-cdk/aws-lambda-nod
export class TampereServerlessStack extends cdk.Stack {
constructor(scope: cdk.Construct, id: string, props?:
super(scope, id, props);
const fn = new NodejsFunction(this, 'HelloFunc', {
entry: './lib/functions/hello/index.ts',
architecture: lambda.Architecture.ARM_64,
});
new cdk.CfnOutput(this, 'FnArn', {
value: fn.functionArn,
});
new cdk.CfnOutput(this, 'Hello', {
value: 'World',
});
}
}
TERMINAL
TERMINAL
ZSH
⚠ The --hotswap flag deliberately introduces CloudFormation drift to speed up deployments
⚠ It should only be used for development - never use it for your production Stacks!
→ tampere-serverless git:(main) ✗ npx cdk deploy --profile admin@poc2
Bundling asset TampereServerlessStack/HelloFunc/Code/Stage...
...mp-517a5e69505c065d8de5232c58591caa146e439061c01f7ff292753deacd2641/index.js 526b
⚡ Done in 2ms
TampereServerlessStack: deploying...
[0%] start: Publishing 8699d04061faede8a74a87db12480e416a8462644e910dc4a171efe446d9253a:current
[100%] success: Published 8699d04061faede8a74a87db12480e416a8462644e910dc4a171efe446d9253a:current
TampereServerlessStack: creating CloudFormation changeset...
✅ TampereServerlessStack
Outputs:
TampereServerlessStack.FnArn = arn:aws:lambda:eu-west-1:[redacted]:function:TampereServerlessStack-HelloFunc65F95DBE-SO5y1LphW0yZ
TampereServerlessStack.Hello = World
Stack ARN:
arn:aws:cloudformation:eu-west-1:[redacted]:stack/TampereServerlessStack/0a4a3970-4bf5-11ec-b90b-0a91ca217385
→ tampere-serverless git:(main) ✗
DEBUG CONSOLE
PROBLEMS
main* 0 0 Live Share AWS Select Postgres Server
Ln 15, Col 17 Spaces: 2 UTF-8 LF TypeScript Colorize: 0 variables Colorize Prettier


cdk-outputs.json — tampere-serverless
tampere-serverless-stack.ts M
cdk-outputs.json U
cdk-outputs.json > ...
{
"TampereServerlessStack": {
"Hello": "World",
"FnArn": "arn:aws:lambda:eu-west-1:
}
}
TERMINAL
TERMINAL
ZSH
→ tampere-serverless git:(main) ✗ npx cdk deploy --profile admin@poc2 -outputs-file ./cdk-outputs.json
DEBUG CONSOLE
PROBLEMS
main*
0 0
Live Share
AWS
Select Postgres Server
Ln 7, Col 1
Spaces: 2
UTF-8
LF
JSON
Colorize: 0 variables
Colorize
Prettier


Infrastructure End-to-End
Jest
1
AWS SDK
2
Send Event
+ verify
EventBridge
EventBus
3
GetItem
+ verify
DynamoDB
Table
4
Got (HTTP Client)
5
Qyery API
+ verify
API Gateway
REST EndPoint




```
PASS e2e/index.e2e.test.ts (9.148 s)
  End-to-End
    ✓ Send Event (2465 ms)
    ✓ Query API (3717 ms)
    ✓ Query API without API key should fail (2 ms)
    ✓ Query API with invalid API key should fail (2 ms)

Test Suites: 3 skipped, 1 passed, 1 of 4 total
Tests:       13 skipped, 4 passed, 17 total
Snapshots:   0 total
Time:        10.596 s, estimated 71 s
Ran all test suites with tests matching "End-to-End".
```

- Lint all your code
- Test your CDK code
- Test your (Lambda) Runtime Application code (and mock external services such as AWS)
- Run integration / end-to-end test against a real system in AWS


Product X prod account
Account Resources
preproduction
pre-environment (optional)
production
environment
Product X dev account
Account Resources
prestaging
pre-environment (optional)
staging
environment
development
environment (optional)
preview/ABC123
environment (optional)
preview/DEF456
environment (optional)

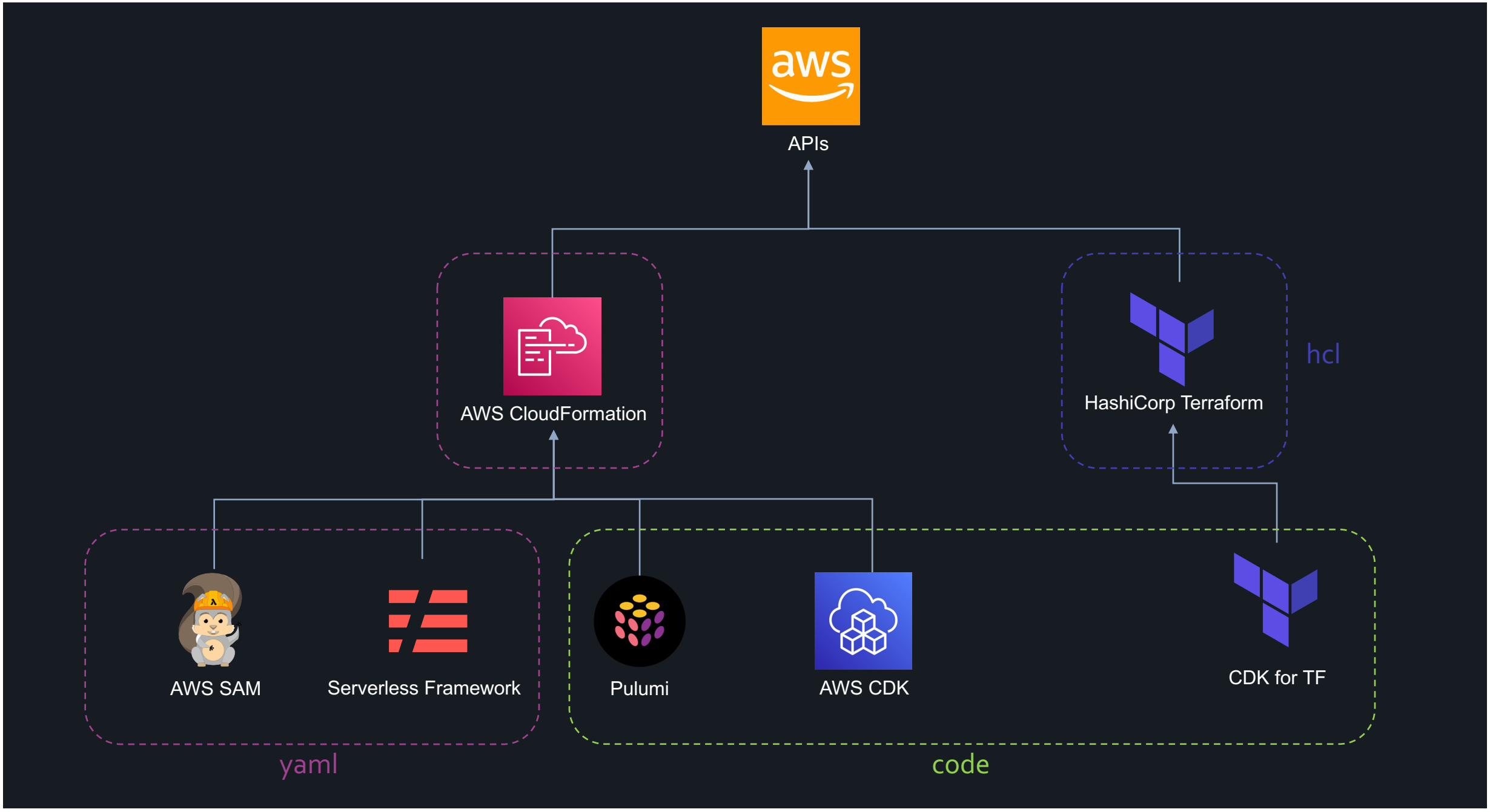
Multiple environments enable *deploy pipelines*

Deploy with confidence

- Lint all your code
- Test your CDK code
- Test your (Lambda) Runtime Application code (and mock external services such as AWS)

- Run integration / end-to-end test against a real system in AWS

vs Others?


APIs
AWS CloudFormation
HashiCorp Terraform
hcl
AWS SAM
Serverless Framework
Pulumi
AWS CDK
CDK for TF
yaml
code

Future?

https://aws.amazon.com/blogs/developer/announcing-aws-cloud-development-kit-v2-developer-preview/

AWS Developer Tools Blog

Announcing AWS Cloud Development Kit v2 Developer Preview

by Chris Fife, Eric Z. Beard, Rico Huijbers, Neta Nir, Niranjan Jayakar, and Otavio Macedo | on 30 APR 2021 | in .NET, Announcements, AWS Cloud Development Kit, Developer Tools, DevOps, Java, JavaScript, Open Source, Python | Permalink | Comments | Share

The AWS Cloud Development Kit (AWS CDK) v2 is now available for Developer Preview in TypeScript, Python, Java, C#, and Go. The AWS CDK is an open-source software development framework to model and provision your cloud application resources using familiar programming languages. With the AWS CDK, you can define your infrastructure as code and provision it through AWS CloudFormation. AWS CDK provides high-level components that preconfigure cloud resources with proven defaults, so you can build cloud applications without needing to be an expert. It also enables you to compose and share your own custom components that incorporate your organization's requirements, which helps teams start new projects faster.

In July of 2019, we announced the general availability of AWS CDK v1 for Typescript and Python. Since that time, we have released support for additional languages in Java and C#, and we released language bindings for Go in Developer Preview. We're now announcing a preview release of v2, which introduces a couple of changes that make it easier for you to consume the AWS CDK and stay up to date with new versions as we evolve it going forwards.

Migrating from the latest minor version of an AWS CDK v1 application to v2 is relatively painless. You need to start by re-bootstrapping your AWS accounts, which is a one-time action. Then for most projects, all you need to do is update your import statements, synthesize, and deploy. You may notice some minor changes to resources, but nothing that requires a resource replacement.

V1

```
npm i -D  \
@aws-cdk/core \
@aws-cdk/aws-secretsmanager \
@aws-cdk/aws-ec2 \
@aws-cdk/aws-rds \
@aws-cdk/aws-iam \
@aws-cdk/aws-kms \
@aws-cdk/aws-ssm \
@aws-cdk/aws-ecs \
@aws-cdk/aws-ecs-patterns \
@aws-cdk/aws-autoscaling \
@aws-cdk/aws-logs \
@aws-cdk/aws-kms \
@aws-cdk/aws-apigateway
```

V2

```
npm i -D aws-cdk-lib
```

https://aws.amazon.com/blogs/developer/experimental-construct-libraries-are-now-available-in-aws-cdk-v2/

AWS Developer Tools Blog

Experimental construct libraries are now available in AWS CDK v2

by Alex Pulver and Nick Lynch | on 08 NOV 2021 | in Announcements, AWS Cloud Development Kit, Infrastructure & Automation, Intermediate (200), Open Source | Permalink | Comments | Share

The AWS CDK v2 experimental APIs are now available as separate packages, in addition to the existing stable APIs.

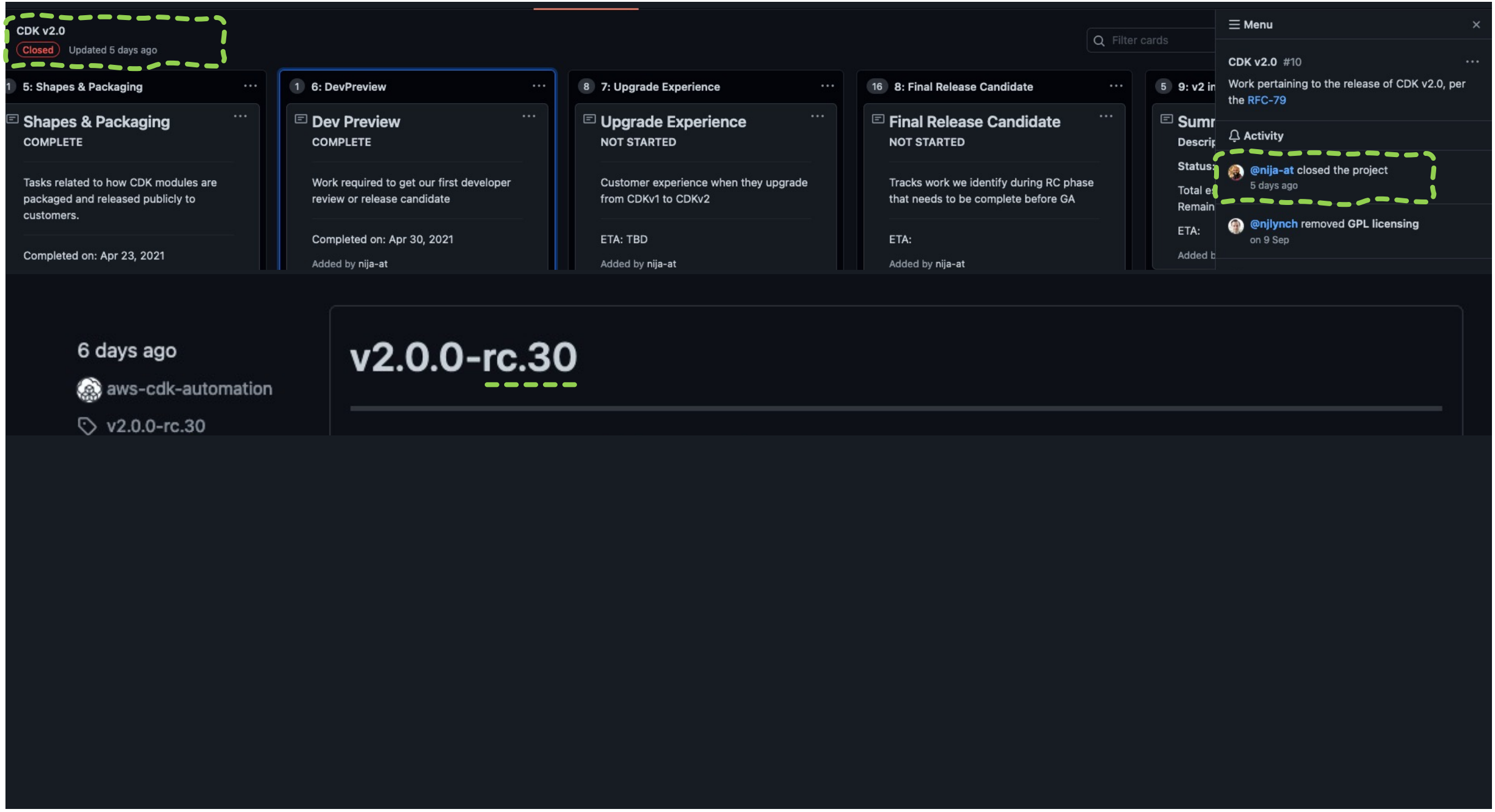
The AWS Cloud Development Kit (AWS CDK) is an open-source software development framework to model and provision your cloud application resources using familiar programming languages. With the AWS CDK, you can define your infrastructure as code and provision it through AWS CloudFormation. AWS CDK provides high-level components that preconfigure cloud resources with proven defaults, so you can build cloud applications without needing to be an expert. It also enables you to compose and share your own custom components that incorporate your organization's requirements, which helps teams start new projects faster.

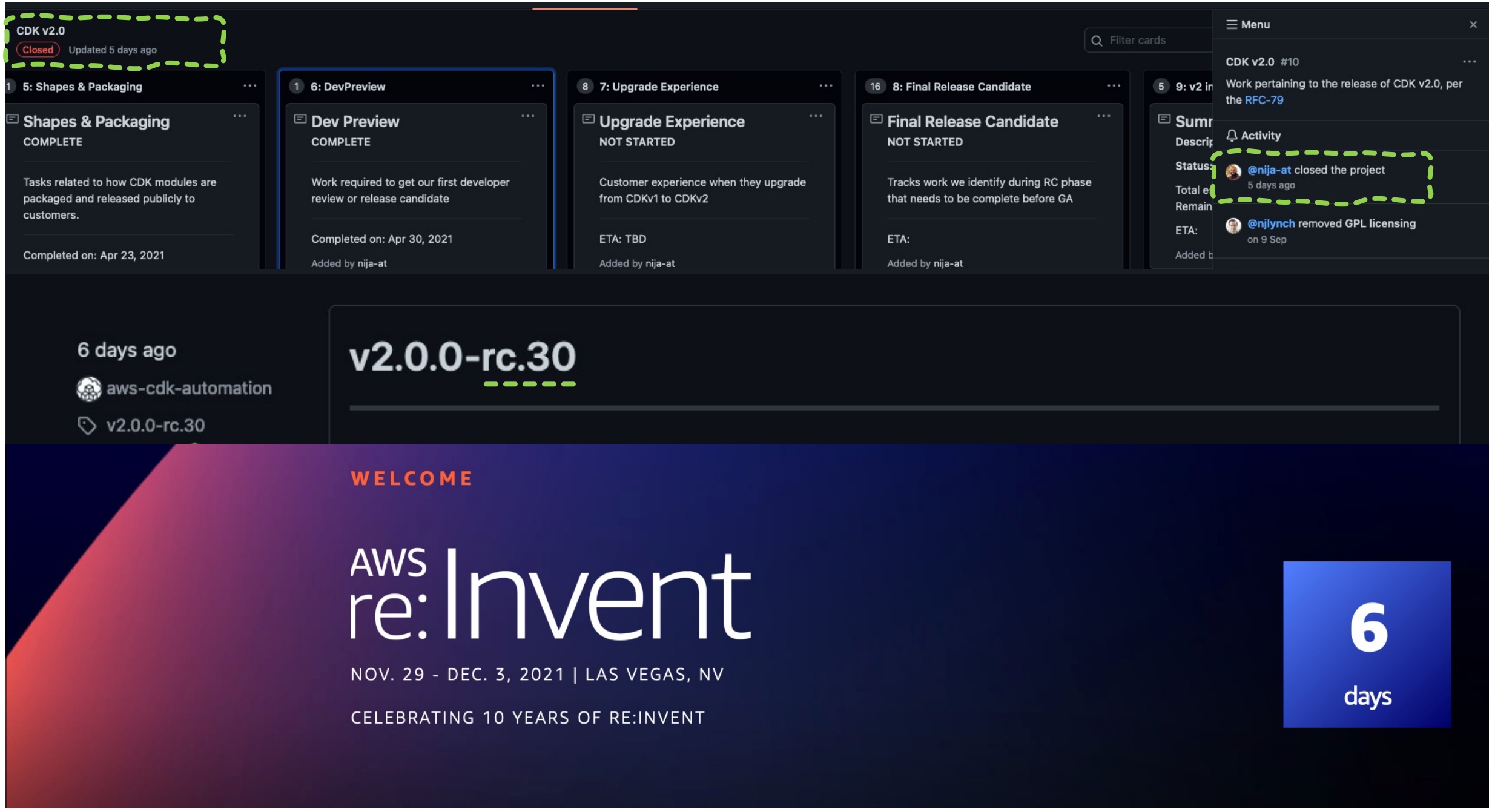
In April of 2021, we announced the developer preview of AWS CDK v2. In AWS CDK v1, we partitioned the AWS Construct Library into many small packages, roughly one per service, so that you only needed to download the packages for those services you wanted to use.

The downside of this approach was that every time you wanted to add a new AWS service to your application, you had to go back to your terminal to `npm install` or `pip install` another package. Additionally, it was very important that all these packages were on the exact same version to avoid interoperability issues.

Based on customer feedback about the v1 package experience, we have consolidated all of the stable AWS Construct Libraries into a single package, called `aws-cdk-lib`. You get access to all the stable AWS CDK constructs by installing this package, and third-party construct libraries only need to take a dependency on this package as well.

We also introduced changes to how we handle experimental classes, methods, and properties. In v1, we followed semantic versioning for all non-experimental code, but where APIs were marked as experimental, we reserved the right to make breaking changes when we felt that the APIs could be improved significantly. Although this gave us the benefit of easily adapting the APIs, customers were sometimes caught off guard by the changes when they didn't notice the experimental banner on a module. Existing tools don't provide a way to clearly identify specific modules as experimental, in a way that works across all of the AWS CDK supported runtime languages. In v2, we strictly follow Semantic Versioning and no longer make breaking changes to any APIs in minor version releases. Instead, we introduced a new lifecycle in which new, experimental construct libraries go through an incubation period as a library completely independent from the main `aws-cdk-lib` library.


CDK v2.0
Closed
Updated 5 days ago
Filter cards
Menu
5: Shapes & Packaging
Shapes & Packaging
COMPLETE
Tasks related to how CDK modules are packaged and released publicly to customers.
Completed on: Apr 23, 2021
6: DevPreview
Dev Preview
COMPLETE
Work required to get our first developer review or release candidate
Completed on: Apr 30, 2021
Added by nija-at
7: Upgrade Experience
Upgrade Experience
NOT STARTED
Customer experience when they upgrade from CDKv1 to CDKv2
ETA: TBD
Added by nija-at
8: Final Release Candidate
Final Release Candidate
NOT STARTED
Tracks work we identify during RC phase that needs to be complete before GA
ETA:
Added by nija-at
CDK v2.0 #10
Work pertaining to the release of CDK v2.0, per the RFC-79
Activity
@nija-at closed the project
5 days ago
@njlynch removed GPL licensing
on 9 Sep
6 days ago
aws-cdk-automation
v2.0.0-rc.30
v2.0.0-rc.30


CDK v2.0
Closed
Updated 5 days ago
Filter cards
Menu
CDK v2.0 #10
Work pertaining to the release of CDK v2.0, per the RFC-79
Activity
@nija-at closed the project
5 days ago
@njlynch removed GPL licensing
on 9 Sep
1 5: Shapes & Packaging
Shapes & Packaging
COMPLETE
Tasks related to how CDK modules are packaged and released publicly to customers.
Completed on: Apr 23, 2021
1 6: DevPreview
Dev Preview
COMPLETE
Work required to get our first developer review or release candidate
Completed on: Apr 30, 2021
Added by nija-at
8 7: Upgrade Experience
Upgrade Experience
NOT STARTED
Customer experience when they upgrade from CDKv1 to CDKv2
ETA: TBD
Added by nija-at
16 8: Final Release Candidate
Final Release Candidate
NOT STARTED
Tracks work we identify during RC phase that needs to be complete before GA
ETA:
Added by nija-at
5 9: v2 in
Summ
Descrip
Status
Total e
Remain
ETA:
Added b
6 days ago
aws-cdk-automation
v2.0.0-rc.30
v2.0.0-rc.30
WELCOME
AWS re:Invent
NOV. 29 - DEC. 3, 2021 | LAS VEGAS, NV
CELEBRATING 10 YEARS OF RE:INVENT
6
days

THANK YOU!

 @aripalo

 /in/aripalo

 aripalo.com

 @almadevelopers

 almamedia.dev