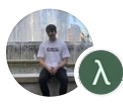


Finally, the end of YAML? AWS CDK for Serverless



Spencer Mehta · Follow

Published in Serverless Transformation · 6 min read · Mar 30, 2022

👤 88

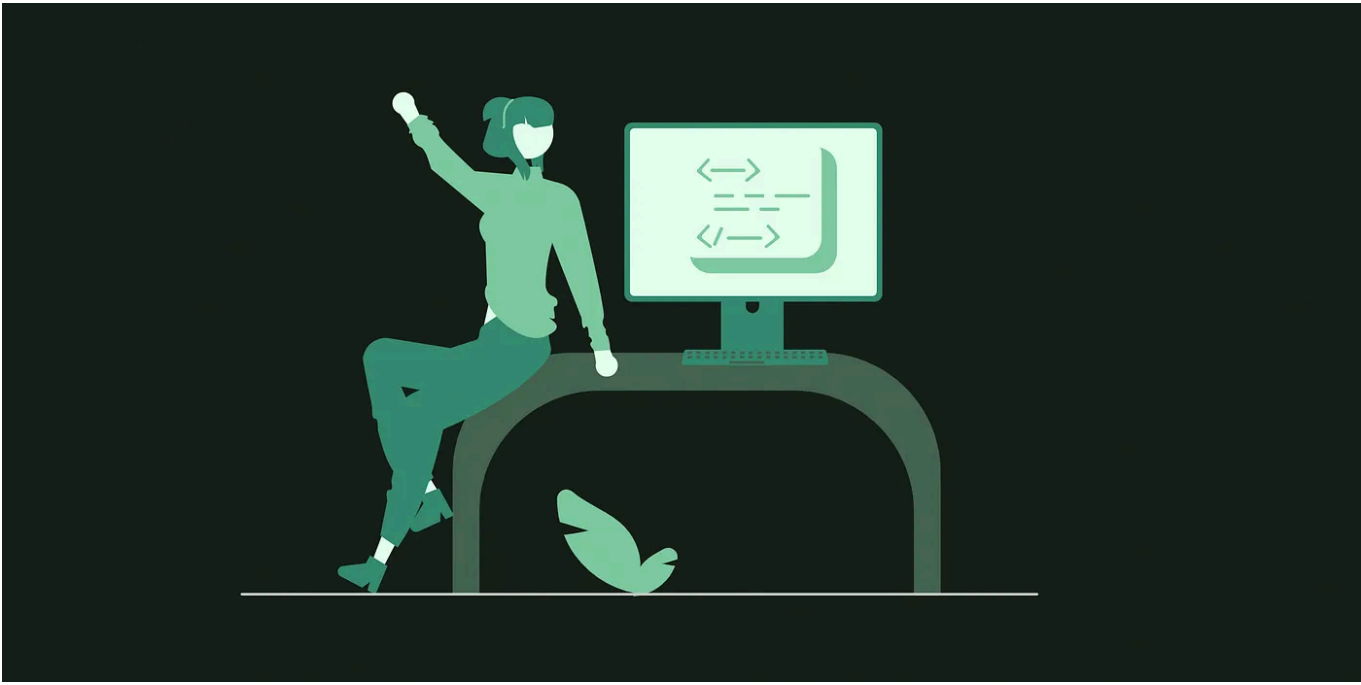
💬 1

🔖

🕒

📄

⋮



The AWS Cloud Development Kit (CDK) is a framework that lets you define your Cloud Application resources using familiar programming languages such as TypeScript or Python. This allows developers familiar with these languages to apply their existing knowledge and get to grips with building Cloud infrastructure rapidly. By using provided high-level components, built with best practices as default, we can abstract much of the complexity away from the developer, and by encapsulating resources into constructs we can continue this practice as our resources scale in size and complexity.

What is Infrastructure as Code?

Infrastructure as Code (IaC) is a discrete representation of all the various resources that our cloud application will use, in machine readable definition files. The main benefits this offers are:

- Version control via git
- Single source of truth
- Reusability — e.g. easy to deploy several production-like environments
- Typing — allowing suggestions and autocomplete
- Testing of the IaC

Why CDK?

IaC is not a new concept, and there are many options to choose from, including Serverless Framework, Terraform, CloudFormation and AWS SAM.

One of the unique features of CDK compared to these others is that it allows us to write IaC in imperative programming languages such as TypeScript and Python, rather than declarative ones such as YAML or JSON. The expressive power of these languages and their ubiquitous support by code editors, formatters, syntax checkers and linters is leagues ahead of support for any YAML-based IaC. This makes the development experience more approachable and rapid, as many errors can be caught by static checks performed by the editor. The code is also more easily readable and comprehensible to developers written in these familiar languages than in YAML.

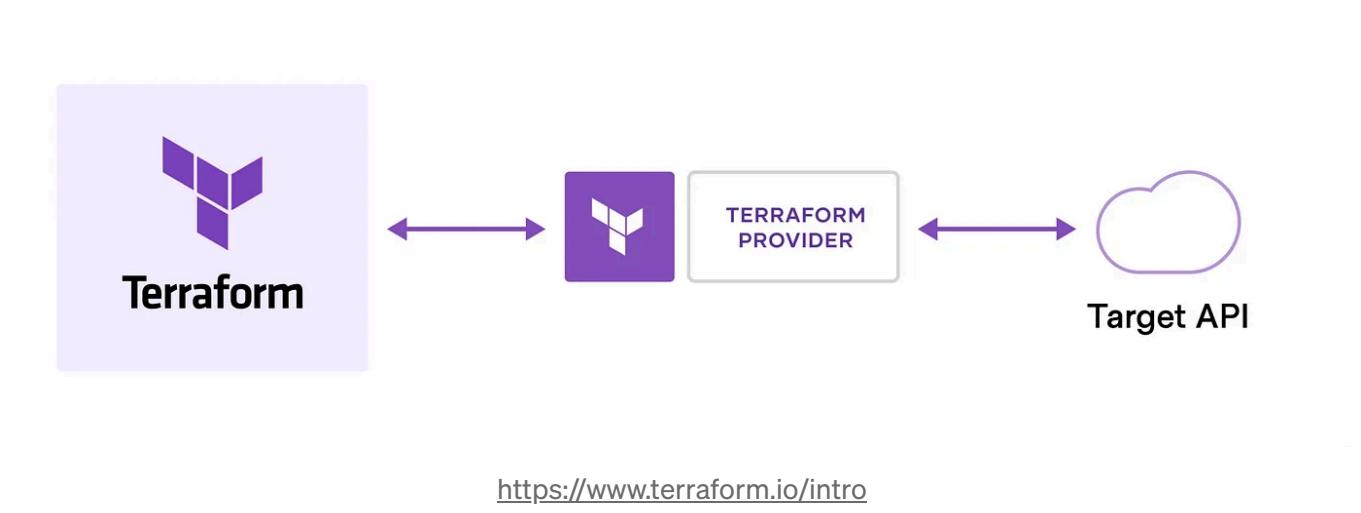
This CDK code is transpiled into CloudFormation templates, AWS's YAML or JSON IaC format. During this 'synth' process, further errors in the IaC can be caught — I like to think of these like compile-time errors in compiled languages like C or Rust. This means we don't have to wait for a failed deployment to catch such errors, saving the developers time.



CDK deploy process

This is in contrast to Terraform's API-based approach, where infrastructure is provisioned in a cloud provider agnostic language, and then plugins are used to interact with Cloud and SaaS providers. The benefit of this approach is migration between cloud providers does not require a complete rewrite of the IaC. However, with serverless architectures we're likely already buying into cloud-provider-specific constructs.

Terraform have also released 'CDK for Terraform', which is an AWS CDK equivalent that transpiles to Terraform rather than CloudFormation. However, this is currently less mature than AWS CDK.



<https://www.terraform.io/intro>

CDK additionally provides libraries to write assertion and validation tests with familiar testing tools such as `jest`, further moving the point at which errors are caught to earlier in the development cycle.

CDK also adopts the paradigm of encapsulation with its 'constructs'. This allows you to wrap specific resource provisioning into a simple, reusable package that can be used again elsewhere.

We'll now take a look at a small demo project creating a REST API with one endpoint. You can follow along with the project at the repository [here](#).

Installation

A pre-requisite is to have the AWS CLI installed and configured with an appropriate account. Once you've done this, you can install CDK using `npm i -g cdk` - this will install it globally on your machine.

We're going to start a new project from scratch using CDK's TypeScript template. Create a new `cdk-demo` directory and execute `cdk init app --language typescript` inside it. You can browse the state of the repo at this point in the `step1` branch.

The basics

There are a couple main files to go over here. Firstly, `lib/cdk-demo-stack.ts` creates our first CDK stack. A stack corresponds to a CloudFormation template, which provisions the resources needed for our applications and services.

Next we have `bin/cdk-demo.ts`, which defines our CDK app. An app can contain multiple stacks, and modularising our services into stacks decouples their deployments. This means if we make changes to only one stack we only need to worry about redeploying this stack.

You can see we initialise our `CdkDemoStack` in this file. Here we can also set deployment configuration, such as the AWS account and region to deploy, and make this vary by environment.

As an example, let's set the deployment region to `eu-west-1` to benefit from the low Irish tax rates by adding the line: `env: { region: 'eu-west-1' },`

We'll now see how easy it is to provision resources by creating a simple Lambda function that we can access through an API Gateway endpoint.

Creating a Lambda

Let's first create a `lambdas` directory to store our code. Inside this we'll create `getLunchSpots.ts` containing the following:

This creates a simple handler that returns a 200 response with the serialised `LunchSpots` object, a response very typical of some sort of API.

This is what our directory structure looks like now:

Adding to the Stack

Now we'll create a Lambda resource in our stack directly from our TypeScript handler function (we don't need to worry about compiling TS to JS ourselves as the CDK construct handles this!). Add the following code to the stack file:

This creates a Node.js Lambda function with the name 'getLunchSpots', using the `handler` function found in `lambdas/getLunchSpots.ts`. As we want to make this an endpoint in our REST API we're going to create an integration resource for our Lambda:

We'll then add an API Gateway API resource to our `CdkDemoStack`:

and create a resource for our `LunchSpots` with a GET method to access our Lambda function:

The `addResource` function adds a new endpoint definition to the API, and the `addMethod` function defines the mapping of this endpoint on a type of request, in this case to our Lambda function integration.

The state of the repo after these changes can be seen in the `step2` branch.

Deployment

We'll now go over how to deploy the resources we've provisioned in the above code. If this is your first time deploying with CDK to the `eu-west-1` region on your account, you'll need to 'bootstrap' your account on this region. Bootstrapping provisions resources that CDK needs to perform the deployment, e.g. an S3 bucket for storing files. You can bootstrap your account by running `cdk bootstrap`.

CDK apps are a definition of our infrastructure as code, rather than the infrastructure as code itself. Upon deployment, they are 'synthesised' into an AWS CloudFormation template for each stack in the app. You can see what these templates look like by executing `cdk synth`.

Another handy CDK feature is the `cdk diff` command. It allows us to view the difference between the currently deployed CloudFormation templates and the template equivalent to the current state of our local CDK code.

To deploy our changes, we can simply run `cdk deploy`. You may get a prompt asking to enter 'y' to confirm deployment of the changes. Near the end of the deployment script's terminal output you should see an 'Outputs' section with an entry 'CdkDemoStack.sohoLunchSpotsEndpoint...'. This is the endpoint of our API Gateway API. If we make a GET request to this url + `/lunch-spots`, we'll see the JSON object we specified in our Lambda function returned.

Testing

We'll also add a short test to make sure our Lambda function was created as expected. Open up `test/cdk-demo.test.ts` and change the code to the following:

This test creates a template from our stack and performs an assertion test on the template to ensure it has a Lambda function with a handler `index.handler` and a runtime of `Node.js 14`. You can browse the current state of the repo in the `step3` branch.

Summary

And that's it! In just a few lines of code we've created, provisioned, deployed, and tested a service on scalable cloud infrastructure! If you're interested in learning more about CDK, I can strongly recommend the [CDK Workshop](#) for a practical guide.

We've found CDK most effective when you're working in a team topology where a single platform team is enabling other product (or stream aligned teams). In such a setup the platform team can encapsulate best practices and conventions into CDK Constructs which are then consumed by the development teams. In this way best practices and conventions are shared in a way that enables teams rather than polices them. CDK is fairly unique in its ability to provide easy and extensible IAC encapsulation — hence it's become our preferred option in such environments.

Cdk AWS Cloud Development Kit Serverless Iac

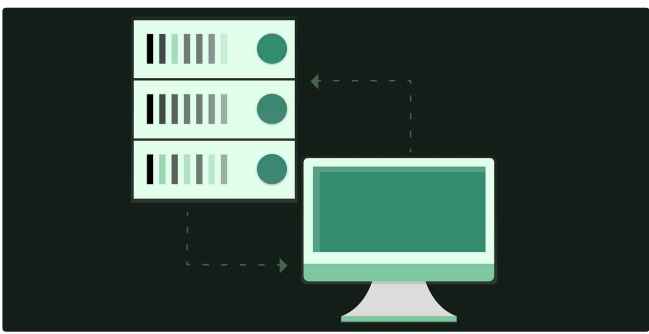


Written by Spencer Mehta

22 Followers · Writer for Serverless Transformation
Software Engineer @ Goldman Sachs

Follow

More from Spencer Mehta and Serverless Transformation



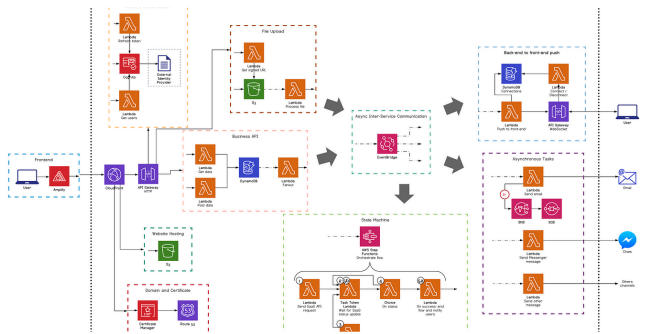
Xavier Lefevre in Serverless Transformation

Asynchronous client interaction in AWS Serverless: Polling,...

Event-driven Serverless often requires async update to the client. There are several ways ...

6 min read · Apr 18, 2020

730 7



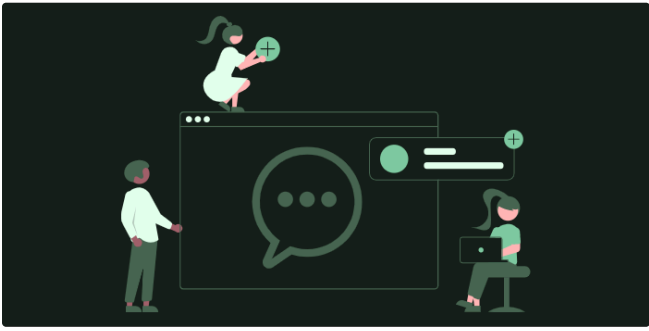
Xavier Lefevre in Serverless Transformation

What a typical 100% Serverless Architecture looks like in AWS!

If you are new to serverless and looking for a high level web architecture guide, you've...

11 min read · May 19, 2020

2.8K 19



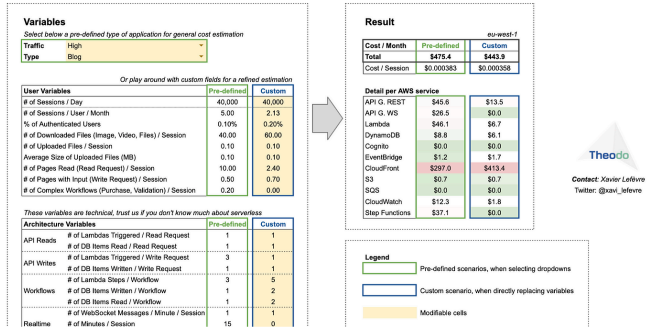
Sarah Hamilton in Serverless Transformation

Building a massively scalable Serverless chat application with...

AWS AppSync allows developers to build quickly. Let's look at how to build a scalable...

9 min read · May 11, 2021

268 4



Xavier Lefevre in Serverless Transformation

Is serverless cheaper for your use case? Find out with this calculator.

Some fixed architectural opinions simplify the process of estimating a serverless project...

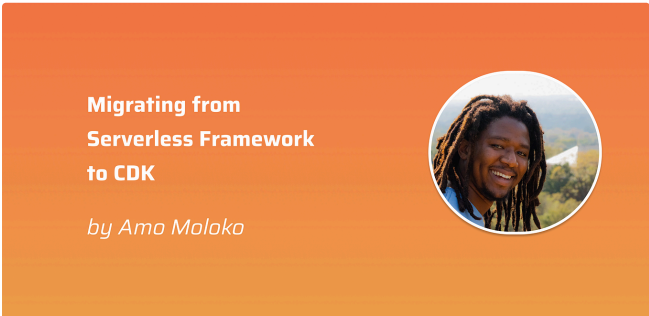
10 min read · Jul 30, 2020

720 2

See all from Spencer Mehta

See all from Serverless Transformation

Recommended from Medium



Amo Moloko

Migrating from the Serverless Framework to AWS CDK

Serverless as a technology is no longer a fad as we venture into 2024. Aleios who have...

12 min read · 4 days ago

41



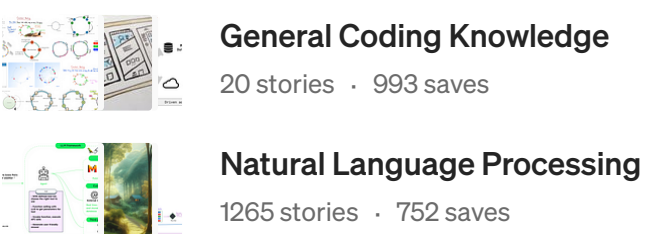
Ivan Poloviy in AWS in Plain English

AWS CloudFormation Mappings

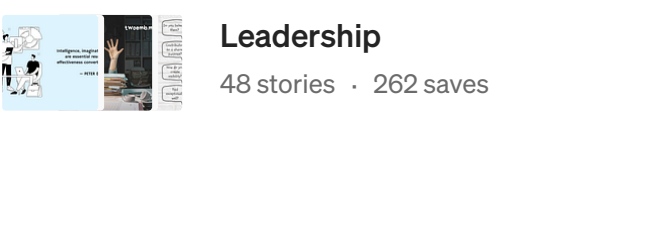
CloudFormation has many handy features. One of such is mappings. This tutorial will...

2 1

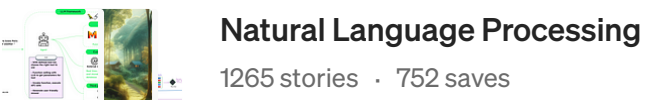
Lists



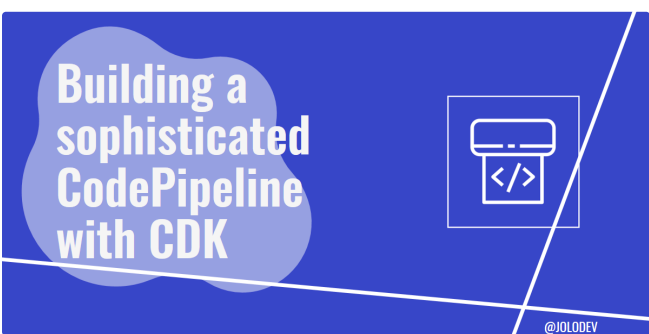
General Coding Knowledge
20 stories · 993 saves



Leadership
48 stories · 262 saves

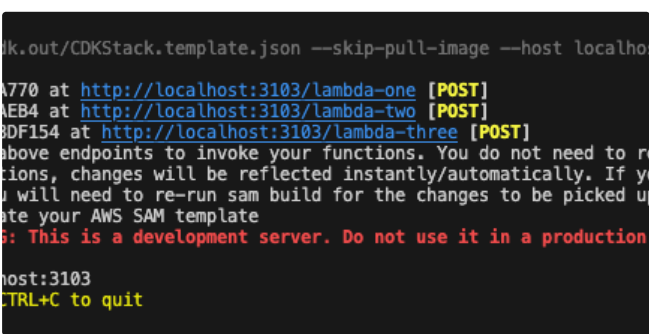


Natural Language Processing
1265 stories · 752 saves



John Nguyen

Building a sophisticated CodePipeline with AWS CDK in a...



Stephen Smith in Duneim Technology

Lessons from migrating a microservice from AWS SAM to...

There are multiple ways to write infrastructure as code for AWS. For serverle...