Navigating the Jungle of Infrastructure as Code in AWS

Beata Johansson and Emil Franzell

April 21, 2023

1 Generative AI disclaimer

We certify that generative AI, incl. ChatGPT, has not been used to write this essay. Using generative AI without permission is considered academic misconduct.

2 Introduction

Once upon a time, in the land of cloud infrastructure, there was something called *"Manual configuration"*. Then came the YAML files, which allowed you to declare infrastructure in a machine-readable format. These tools had names like CloudFormation or Terraform. In the modern day, however, solutions architects like something a little more elegant. The traditional way of defining Infrastructure as Code (IaC) is through templates in declarative languages such as YAML and JSON. Some of the most popular ways are discussed in this essay.

On the bleeding edge of IaC exists the AWS Cloud Development Kit and its siblings, which are tools that allow cloud architects to define their templates in the terms of procedural programming languages, which then gets converted into one of the declarative forms. With additions such as GitHub Copilot and VS Code IntelliSense, being a cloud architect has never been easier. The problem that arises in situations like these is always the same. How do I pick which tool to use when there are so many?

In this essay, we will go over the various trade-offs related to picking your IaC tool for one of the main cloud providers: AWS. Furthermore, we will discuss how the tools contributes to the best IaC practices.

3 IaC Best Practices

Before we start, let's discuss the best IaC practices. In general, IaC is an approach for managing the entire infrastructure, both provisioning and configuring the infrastructure. An important aspects of IaC is to define it in code with declarative statements [Wik]. One of the main purposes of defining infrastructure in code is to avoid configuring servers, tools, or other resources manually. Manual configuration has a tendency to become time inefficient and it is also difficult to have consistency. This approach does therefore not align with the DevOps methodology.

In order to get the full benefits of IaC it is necessary to use the IaC best practices. The first best practice is about assuring that the infrastructure code handles all necessary configurations, since it is key to avoid manual configurations after deploying the infrastructure [Nau]. For easy management it is necessary to use version control on the IaC [Nau]. Furthermore, IaC is also a part of development and changes will be made by the development team. Therefore this code should also be tested continuously in CI/CD pipelines [Nau].

4 AWS Resources In a Nutshell

Before diving into how to do IaC in AWS, let us look into what AWS resources are. In AWS a resource is a unit which you can create in AWS [Serf]. For example, a resource can be a virtual server like a AWS EC2 instance, or it can be storage like Amazon Simple Storage Service (S3), or a database like DynamoDB. There are many examples of resources that are offered, but the common feature is that

they are deployed on AWS's cloud environment. When the number of AWS resources increases in your infrastructure, they need to be managed efficiently. This is where the need for IaC in AWS is created.

5 AWS Tools for IaC

In this section all AWS recommended IaC tools are explained.

5.1 AWS CloudFormation

The mother of all infrastructure in AWS is AWS CloudFormation. AWS CloudFormation is a declarative template language for IaC on AWS. The infrastructure itself is composed of AWS resources, and these need to be configured properly. The CloudFormation language consists mainly of two different types of entities to manage resources. There are **Templates** and **Stacks**. Templates are YAML-files that contain definitions of the infrastructure that should be provisioned. Stacks are collections of infrastructure resources that are related and should be deployed together. Note that CloudFormation provides the option to use JSON instead of YAML, but it is a lot more clunky. CloudFormation is a highly verbose language, requiring full configurations for every piece of infrastructure that is provisioned.

5.2 AWS Cloud Development Kit

The AWS CDK is an open source abstraction and tooling on top of CloudFormation that allows the cloud architect to define infrastructure as code in one of several popular programming languages. It is a relatively new tool, which achieved general availability in 2019 [BI19]. For example, one can use TypeScript to define infrastructure, which means that together with something like Vs Code IntelliSense, provisioning infrastructure can become a breeze. For simplicity, the following text will only discuss CDK using TypeScript, but there are several languages available.
The CDK for TypeScript lets the developer define cloud infrastructure as a TypeScript file that creates several objects using class constructors. At the top of the object tree that is constructed through this process is the Stack.

```
import { App, Stack, StackProps } from 'aws-cdk-lib';
import * as s3 from 'aws-cdk-lib/aws-s3';

class HelloCdkStack extends Stack {
  constructor(scope  App, id  string, props?  StackProps) {
    super(scope, id, props);

    new s3.Bucket(this, 'MyFirstBucket', {
      versioned  true
    });
  }
}
```

An example file that creates a DynamoDB table in your AWS account.

Resources in the stack are defined together in units called constructs. One example of a construct is the DynamoDB table as seen in the code above. We can define our own constructs that themselves instantiate constructs.

```
export interface MyConstructProps {
    myProp?  string;
}

export class MyConstruct extends Construct {
    constructor(scope  Construct, id  string, props  MyConstructProps = {}) {
        super(scope, id);
        const bucket = new s3.Bucket(this, 'bucket');
        const topic = new dynamodb.Table(this, 'Table', {
            partitionKey: {name  id, type: dynamodb.AttributeType.STRING },
            billingMode  dynamodb.BillingMode.PAY_PER_REQUEST,
        });
        bucket.addObjectCreatedNotification(new s3notify.SnsDestination(topic),
        { prefix  props.prefix });
    }
}

const app = new App();
new HelloCdkStack(app, "HelloCdkStack");
```

An example file that creates a new construct that can be reproduced anywhere in a stack

Since constructs can be nested, there are three named tiers of abstraction for constructs.

L1) L1 or Level 1 constructs, are constructs that are fully equivalent with their CloudFormation counterparts, meaning that all configuration has to be provided in the declaration of the construct.

L2) L2 constructs are slightly more streamlined and contain sane defaults for resources in the cloud, as well as some basic glue logic and boilerplate that would otherwise have to be implemented by the developer.

L3) L3 constructs are constructs that implement common patterns in AWS. One example of many is the LambdaRestApi construct, which deploys a serverless AWS Lambda function, accessible through an AWS API Gateway endpoint [Serb].

5.3 AWS Cloud Development Kit for Kubernetes

Kubernetes is an open source container orchestration tool which has gained popularity since its first release in 2014. The core functionality of Kubernetes is the use of server clusters for deploying your containerized applications on. Kubernetes clusters are convenient since it facilitates scaling of infrastructure and applications [Serc]. There are many options for hosting a Kubernetes cluster. For example, it is possible to configure Kubernetes from scratch and host it on your own servers or in a cloud provider environment. The three main state-of-the-art cloud providers like Google Cloud, AWS, and Microsoft Azure, all provide managed Kubernetes services. The purpose of these kinds of services is that some parts of the Kubernetes cluster configurations are pre-made to suit the specific cloud provider environment. As a result, this simplifies the configuration process for the engineers. For AWS this is called Amazon Elastic Kubernetes Service (EKS) [Serg].

The AWS Cloud Development Kit for Kubernetes (cdk8s) is an open-source tool released in 2020. The idea for this tool was inspired by the development of AWS Cloud Development Kit, as mentioned in section 5.2. The cdk8s converts code definitions in for a Kubernetes application in TypeScript or Python, and converts it into YAML code. The YAML code can thereafter be deployed on any Kubernetes cluster [Ser23a]. However, one important aspect is that the actual deployment step is not managed by cdk8s. For IaC, this means that configuration of the cluster and the deployment of the

application must be handled separately in the CI/CD pipeline [Ser23b]. Another implication of this is that it is possible to use cdk8s to define applications as code without using AWS for hosting the infrastructure.

An important question to address is why do we need to write application definitions and configurations in Python or TypeScript, insted of YAML?

5.4 Terraform

Terraform is an open-source IaC tool that supports multiple cloud platforms, including AWS [Hasc]. Terraform is used for the common Iac use cases, e.g. provisioning infrastructure like servers, networks, and even Kubernetes clusters. Terraforms seems similar to CloudFormation since it takes definitions for infrastructure in the form of a declarative text file. In the case of Terraform, the format used is the Hashicorp Configuration Language (HCL), which is a fully JSON-compatible language [Val21].

5.5 AWS Cloud Development Kit for Terraform

The AWS Cloud Development Kit for Terraform (CDKTF) is a new open-source toolkit, released in 2022. CDKTF is for Terraform, what the AWS CDK is for CloudFormation[Hasa]. It is built using libraries from the CDK and is a lot newer and less stable than CDK. In the background, CDKTF manages AWS resources using the AWS developed Cloud Control API, which is mentioned in section 5.6.

5.6 AWS Cloud Control API

Another method for directly managing the AWS resources is to use the AWS Cloud Control API. By default AWS resources can be controlled by Create, Read, Update, Delete, List (CRUDL) commands. The Cloud Control API supports these commands and lets you manage the resources with code and enables customization for IaC. This tool is listed by AWS themselves as an IaC tool, but is aimed at IaC tool developers, rather than developers and architects of other applications [Sera]. For this reason, this tool will not be recommended in this essay as much as mentioned. However, as you might recall from section 5.5, CDKTF tool use the Cloud Control API. One of the purposes of the Cloud Control API is that it supports many of the AWS resources and third-party resources. It is possible to look up which resources are supported in the documentation [Sere].

6 Deciding on which IaC tools to use

There are many options of tools for building infrastructure on AWS. The choice boils down to a variety of factors. The opinion of the writers is that the CDKs are all great options, but there are some trade-offs to consider.

6.1 Vendor Lock-In

If you are concerned about vendor lock-in, your choice of IaC tools can be critical. The AWS CDK is, as stated above, a tool that only provisions AWS resources, just like CloudFormation and the AWS Cloud Control API. By using Terraform, you are not going to be confined to AWS by the tool itself, but any IaC code you create for AWS will only be capable with AWS in most cases, considering that the declarative Terraform code will refer specifically to AWS services. Terraform can however be good for other things than switching between clouds. For example, you can provision resources in several clouds with a single file and integrate them together. This may however not be a best practice. For a fully cloud-agnostic IaC setup, you should use Kubernetes (K8s) and if you like, cdk8s. K8s can be run on any cloud and makes it easy to switch provider in the case of rate hikes or other issues that may warrant a switch.

6.2 Stability

Most of the tools that are discussed in this essay are cutting edge and have only existed for a few years. Notably the AWS CDK was released in June 2019. Version 1.0.0 of cdk8s was released in October 2021

and at the time of writing, CDK for Terraform is still in version 0.16.x. According to the CDKTF team, it is likely that there may be many breaking changes introduces before version 1.0.0[Hasa]. This means that you may need to rebuild some things if you use CDKTF, even in the short term. Terraform itself was bumped to version 1.0.0 in June 202[Hasb]e, so it is a very new tool. CloudFormation was initially released in February 2011 and is today relied on by many huge corporations[Serd]. For these reasons, it is most likely an option that will not break anytime soon. It is worth to keep in mind that just because a tool is relatively new, it does not mean that the maintainers of that tool will introduce breaking changes and drop support for older versions anytime soon. It does however incur a certain amount of risk. One benefit of using tools that have a large user-base and are backed by large corporations, is that it's less likely that the projects will be dropped all of a sudden. This holds for CDK, CloudFormation and Terraform in particular.

6.3 FOSS

A concept that is revered and preached by software engineers and delvelopers around the world is Free Open-source software (FOSS). Most of the tools listed are FOSS. However, CloudFormation is proprietary software owned by AWS and many of these tools build heavily on CloudFormation or Cloud Control API. The CDKs however are completely FOSS and you can contribute to them today. The same goes for Terraform.

6.4 With Great Power...

The CDKs are especially easy to make dangerous mistakes with. Here is an exercise for the reader: Why is the following CDK snippet possibly a very bad mistake?

```
export class MyConstruct extends Construct {
  constructor(scope  Construct, id  string, props  MyConstructProps = {}) {
        super(scope, id);

        //Create roles, other boilerplate
        //... defined: defaultVpc, role, securityGroup

        for(let i = 0; i < 1000; i++){
          new ec2.Instance(this, `ec2-instance-${i}`, {
            vpc  defaultVpc,
            role,
            securityGroup,
            instanceName: `ec2-instance-${i}`,
            instanceType  ec2.InstanceType.of(
              ec2.InstanceClass.X2IEDN,
              ec2.InstanceSize.32XLARGE
            ),
            machineImage  ec2.MachineImage.latestAmazonLinux({
              generation  ec2.AmazonLinuxGeneration.AMAZON_LINUX_2,
            }),
          })
        }
    }
}
```

The answer: This code creates 1000 EC2 instances that each cost $30+/hour.

Well, it might, but it might also not, because there probably aren't 1000 instances of that size available in a given region and AWS enforces account limits that may be reached before 1000 instances are deployed. It stands that the CDKs are incredibly powerful tools that can accomplish mighty tasks, but they can also be dangerous in the wrong hands.
Given the knowledge of how to use these tools, one could easily perform severe acts of malice with access to a company AWS account with the right permissions. Like in the example above, where we

rack up the monthly bill of an account to over $216,000,000 USD. If we were smart, we could also deploy crypto mining software on the VMs, earning us some cash in the process. Or we could build our own cloud platform and rent out virtual private servers. The possibilities are endless.

Worth noting is that exploiting this on an AWS account does not require that the AWS account is already using CDK or CloudFormation or any of the other tools mentioned. The only requirement is that the account is a root account or has the permission from a root account to perform these actions. It is however more likely that an AWS account in an organization that uses these tools will have these rights in order to let developers deploy infrastructure in their own stages/environments. It should also be noted that pure CloudFormation or Terraform is capable of the exact same things as CDK is, but the authors of this essay do not know how to do that in a pretty way.

7 Conclusion

The choice of which IaC tool to use when developing on AWS is ultimately dependent on your preferences and requirements. If you are sure that you will stay on AWS for the life-cycle of the application you are building, and you want a modern, stable tool with a large community behind it, the AWS CDK is a great option. If you want a cloud-agnostic infrastructure that can be deployed anywhere, or simply like Kubernetes, cdk8s is a good choice. You should probably wait some time before building your unicorn startup with CDKTF, since you may have to migrate everything to a new format in a future release. Speaking as someone who uses the CDK daily, it's great. Don't waste your time with pure CloudFormation. Go directly to CDK. I'd even go so far as to say it's better than Terraform for AWS. Finally, Remember to practice proper cyber security, so that no malicious actor gets their hands on one of your AWS accounts with lots of infrastructure permissions.

References

[BI19] Elad Ben-Israel. v1.0.0. https://github.com/aws/aws-cdk/releases/tag/v1.0.0, 2019. Accessed 2023-04-18.

[Hasa] HashiCorp. Cdk for terraform. https://developer.hashicorp.com/terraform/cdktf. Accessed 2023-04-18.

[Hasb] HashiCorp. v1.0.0. https://github.com/hashicorp/terraform/releases/tag/v1.0.0. Accessed 2023-04-21.

[Hasc] HashiCorp. What is infrastructure as code with terraform? https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code. Accessed 2023-04-18.

[Nau] Marija Naumovska. v1.0.0. https://betterprogramming.pub/5-best-practices-for-infrastructure-as-code-82bd5ce12a79. Accessed 2023-04-21.

[Sera] Amazon Web Services. Aws cloud control api. https://aws.amazon.com/blogs/aws/announcing-aws-cloud-control-api/. Accessed 2023-04-21.

[Serb] Amazon Web Services. Lambdarestapi. https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_apigateway.LambdaRestApi.html. Accessed 2023-04-21.

[Serc] Amazon Web Services. Nodes. https://kubernetes.io/docs/concepts/architecture/nodes/. Accessed 2023-04-21.

[Serd] Amazon Web Services. Release history - cloudformation. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/ReleaseHistory.html. Accessed 2023-04-21.

[Sere] Amazon Web Services. Resource types that support cloud control api. https://docs.aws.amazon.com/cloudcontrolapi/latest/userguide/supported-resources.html. Accessed 2023-04-21.

[Serf] Amazon Web Services. What are resource groups? https://docs.aws.amazon.com/ARG/latest/userguide/resource-groups.html. Accessed 2023-04-21.

[Serg] Amazon Web Services. What is amazon eks? https://aws.amazon.com/pm/eks. Accessed 2023-04-21.

[Ser23a] Amazon Web Services. Aws cloud development kit for kubernetes. https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/aws-cdk-for-kubernetes.html, 2023. Accessed 2023-04-18.

[Ser23b] Amazon Web Services. How does it work? https://cdk8s.io/docs/latest/#how-does-it-work, 2023. Accessed 2023-04-18.

[Val21] Alfonso Valdes. Terraform vs cloudformation: The final battle. https://www.clickittech.com/devops/terraform-vs-cloudformation/, 2021. Accessed 2023-04-18.

[Wik] Wikipedia. v1.0.0. https://en.wikipedia.org/wiki/Infrastructure_as_code. Accessed 2023-04-21.