
MODERN CLOUD
INFRASTRUCTURE
-AS-CODE
FOR DEVELOPERS
SALIE LIM

Modern Cloud Infrastructure-as-Code for Developers

Salie Lim

This book is for sale at
http://leanpub.com/modern-infrastructure-as-code

This version was published on 2023-03-25

ISBN 978-981-18-6971-6

Contents

1.0. Introduction

1.1. Who is this book for and prerequisites

This book is for software engineers and web developers wanting to explore emerging cloud technologies. It does not matter if you are a seasoned front-end, back-end or full-stack engineer, this book is for those who want to dip their toes in the infrastructure, cloud and DevOps world. With this book you will learn how to deploy and scale your web application.

This book will also assume that you have little knowledge of cloud technologies and guide you through the basics of Amazon Web Services (AWS), Docker, Kubernetes, GitHub Actions and Terraform. Feel free to skip to the exercises if you have basic knowledge about cloud tools and dive straight into building infrastructure with AWS and Terraform.

1.2. What will we be learning?

In this book we will be introducing the concepts of IaC, DevOps and Cloud Computing. After we gain an understanding of Cloud concepts, methodology and tools we will move on to building 3 of the most common architectures we find in modern software and infrastructure.

We will focus on building Amazon Web Services (AWS) infrastructure in the exercises in this book. However keep in mind that cloud concepts are transferrable across the various cloud platforms.

1.3. What will you build

In the end there will be three projects that you will build and publish publicly.

First we will build a serverless frontend with AWS Simple Cloud Storage (S3) and Amazon CloudFront.

Second, a serverless Application Programming Interface (API) with Amazon Elastic Container Registry (ECR) and AWS Lambda.

Third, we will deploy scalable containers with Amazon Elastic Kubernetes Service (EKS). We will be using Terraform as our IaC tool throughout these exercises to codify our infrastructure.

Before we start building, let's start by understanding IaC concepts, tools and the role developers play in the world of DevOps.

2.0. The Big Picture - IaC, DevOps, Cloud and Beyond

2.1. Infrastructure-as-code (IaC)

We have entered the era of infrastructure-as-code (IaC). Since infrastructure is now code, it is not separated from development and becomes a function of it. IaC allow engineers to model infrastructure with code. Infrastructure is defined and codified in text files. These files are then committed and version controlled in a central Git repository.

This allows engineers to configure and manage complex cloud infrastructures in an organised way. Without IaC, engineers will have to connect to cloud providers or use web dashboards to manually provision new resources. This manual workflow is prone to mistakes, one may manually make changes to one environment and forget to follow through on another. It also does not give the team a holistic view of the application infrastructure.

Another advantage of IaC is the automatic adaptation to changes in configuration, it allows infrastructure to scale up or down depending on the web traffic, using auto-scaling features.

2.2. Developers and DevOps

DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). In the past, developers write code

and not manage infrastructure. The relationship between developers and infrastructure was distant. However with the rise of DevOps, the developer-infrastructure relationship has evolved. In DevOps, development and system infrastructure is closely integrated, they function together to serve the continuous integration and continuous deployment (CI/CD) pipeline. CI/CD bridges the gap between development and application go-live by automating building, testing and deployment of code.

Developers are increasingly expected to create, provision and manage cloud infrastructure for the web applications that they build. This is on top of traditional developer practices such as writing application code and unit tests, code reviews, and familiarity with agile principles. The boundaries between the roles of software developer and DevOps or systems engineer is dissolving. Whether you are a Frontend, Backend or Full-Stack developer you will probably need to administer cloud infrastructure at some point in your career.

2.3. Cloud Computing and Amazon Web Services (AWS)

Cloud computing is the delivery of computing services including servers, storage, databases, networking and software over the internet. It enables fast innovation, flexible resources and economies of scale. With the cloud, engineers can have access to large scale computing capacity quickly and more cheaply than purchasing and building physical servers.

AWS is a comprehensive cloud platform offering over 200 services from basic compute storage and databases to emerging technologies like machine learning and internet of things. It enables developers to build web applications, APIs and much more in the cloud.

Some of the most popular AWS services includes Amazon S3,

Amazon EC2, AWS Lambda and Amazon Cloudfront. We will be using all of these services in one way or another in the tutorials in this book. AWS is currently the most widely adopted cloud service provider. Microsoft Azure and Google Cloud are close competitors.

2.4. Docker

Docker is an open platform for developing, deploying, and running applications. It enable developers to package and run an application in an isolated environment called a container. A container is lightweight and contains everything needed to run the application without needing to rely on the host's installations. Developers can easily share containers via an image and be sure that everyone gets the same container that runs the application in the same way.

Docker makes it easy for developers to manage the lifecycle of your containers, package and ship applications, reducing the delay between writing code and running it in production. Due to its isolation and security, developers can also run multiple containers simultaneously on their machine.

2.5. Kubernetes (K8s)

Kubernetes is an open-source system originally developed at Google. It helps developers automate the deployment and scaling of containerised applications by grouping containers for easy management.

An advantage of using Kubernetes is that it allows your applications to scale easily. It grows with your team and company to deliver applications consistently without the complexity and without increasing your infrastructure or devops team.

Another advantage of using KuberflexibilityYou can use Kubernetes locally, on-premises, in a hybrid architecture, or public cloud infrastructure. It enables developers to build on a cloud agnostic architecture, we can effortlessly move scalable applications across different cloud providers, whether it is AWS, Google Cloud or Microsoft Azure.

Some other container management features of Kubernetes include automated rollouts and rollbacks, service discovery and load balancing, horizontal scaling and secret management.

2.6. Terraform

Terraform is an IaC tool that enable developers to create, modify, version and destroy infrastructure using code. Components such as compute instances, storage, networking, databases and DNS entries can be managed using Terraform.

Developers can use Terraform language to write infrastructure in human-readable and declarative configuration files. These files provides an infrastructure resources blueprint that developers can version, share, and reuse. It allows us to apply complex changesets to infrastructure with minimal manual configuration.

Terraform is a cloud-agnostic tool which provides engineers with additional flexibility. It allows us to integrate with multiple cloud providers and orchestrate resources outside the AWS ecosystem. It is advantageous for our IaC tool to have multi-provider utility as it enables multi or hybrid cloud knowledge transfer.

Terraform builds a resource graph, creating and updating non-dependent resources in parallel. This makes Terraform very efficient at building resources. Terraform also has Drift Detection features, it able to resolve inconsistencies, refreshing to a correct state of the infrastructure even if a developer manually edited infrastructure in the cloud console.

There are other alternative IaC tools like AWS Cloud Formation, Ansible and Chef. Terraform is however the most widely used tool. We will be using Terraform in all of the example architecture deployment in the book as it is easy-to-use and understand.

2.7. GitHub Actions

GitHub Actions makes it easy for engineers to automate software application workflows with CI/CD pipelines. It enables developers to build, test, and deploy code right from GitHub.

You can codify CI/CD workflows with GitHub Action events like push, issue creation, or a new release. The most common use case for GitHub Actions is triggering tests and builds to run once a developer commits his or her code to Git or opens a pull request.

It is also used for deploying to any cloud, create tickets in Jira, or even publish a package to npm, the possibilities are endless with millions of open source libraries available on GitHub.

3.0. Serverless Frontend

3.1. Why Serverless Frontend?

Serverless allows us to run frontend web applications and services without the need to provision or configure servers, AWS does all the heavy lifting for you so that you can focus on coding. There are several advantages of going serverless. It allows your application to scale automatically, there is no need to provision multiple servers to keep up with the traffic. It is also cost-effective, you do not have to pay for idle capacity and there is no charge when your code is not running.

3.2. Infrastructure and Tools

Amazon Simple Storage Service (Amazon S3)
Amazon S3 is an object storage service. This service is highly scalable, durable, secure and available. Developers can use S3 to store all types of data for a range of use cases, such as websites, mobile applications, big data and even backup and archiving. It is easy-to-use and cost-efficient.

We will be using Amazon Simple Storage Service (S3) to host the web application's static web resources such as HTML, CSS, JavaScript, images and more.

Amazon CloudFront
Amazon CloudFront is a fast content delivery network (CDN) service. It allows your web application to be connected via the AWS network backbone all over the world, so that your web application can be secure, performant, and highly available. Some

of CloudFront's feature includes encryption and HTTPS support. It can be integrated with AWS Web Application Firewall and Amazon Route 53 to protect against web attacks. It also works seamlessly with any AWS origin, such as Amazon S3 and Amazon EC2.

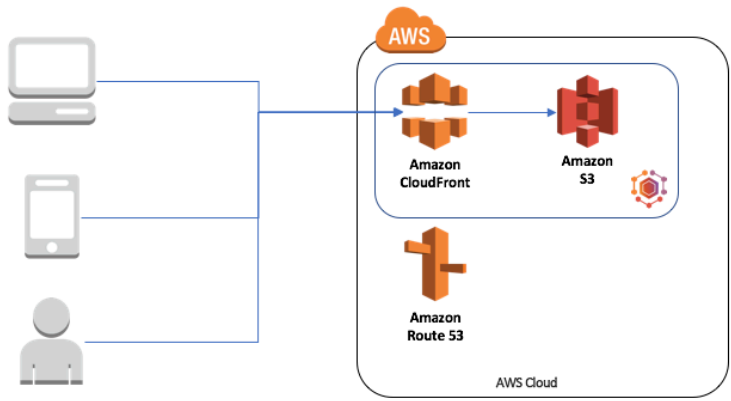
Amazon CloudFront will be used as our fast content delivery network (CDN) provider to deliver the frontend website to customers globally with low latency and high transfer speeds.

Terraform
Terraform is our tool of choice to manage and deploy the infrastructure using code.

GitHub Actions
We will use GitHub Actions to automate software CI/CD workflows.

3.3. Architecture

Amazon CloudFront sitting in front of S3. Optionally, Route 53 is used for routing:



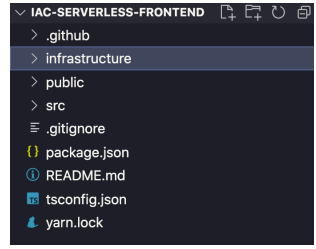
Serverless Frontend Architecture

3.4. Getting Started

We need to create a new React web application and have Terraform CLI installed.

React Application
Start a simple react application using Create React App is an officially supported way to create single-page React applications. You can get started by following the instructions here, https://create-react-app.dev/docs/getting-started[1].

After Create React App is created, create a folder named `infrastructure` in the root directory of the application folder.



Application folder

Terraform CLI
We will be using the Terraform CLI in this tutorial and throughout the book. Install it by identifying the most suited installation method on Install Terraform Documentation[2] and following the instructions.

3.5. Code the Infrastructure

Note that you can find all the code in this tutorial at https://github.com/salielim/infrastructure-as-code/tree/main/iac-

[1] https://create-react-app.dev/docs/getting-started
[2] https://learn.hashicorp.com/tutorials/terraform/install-cli

serverless-frontend

Let's now create our first Terraform file, `variables.tf` in the `infrastructure` directory. In this file we define all the variables we will be using to build our infrastructure using code.

```
variable "project_key" {
  description = "Project name or key."
}

variable "s3_bucket_name" {
  description = "The name of the bucket"ƒ
  default     = "iac-serverless-frontend"
}

variable "s3_bucket_env" {
  description = "The AWS S3 bucket environment name."
}

variable "aws_region" {
  description = "The AWS region to create resources in."
  default     = "ap-southeast-2"
}

variable "aws_access_key" {
  description = "The AWS access key."
}

variable "aws_secret_key" {
  description = "The AWS secret key."
}
```

Create a second Terraform variables file in the same directory is and name it `terraform.tfvars`. This file is specifies the values to the variables you defined in the previous step.

```
project_key    = "iac-serverless-frontend"
aws_access_key = "XXXXXXXXXXX"
aws_secret_key = "XXXXXXXXXXX"
aws_region     = "ap-southeast-2"
s3_bucket_name = "iac-serverless-frontend"
s3_bucket_env  = "development"
```

The third Terraform file we will create in the same directory is `provide.tf`. This file contains information about our provider, AWS and its' configurations including keys and region.

```
provider "aws" {
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
  region     = var.aws_region
}
```

Next, we will start defining the S3 Bucket that we will use to store the static resources in `s3-bucket.tf` . Here we defined the bucket permissions of public read only. We also defined `index.html` as the entry point to the website.

```
resource "aws_s3_bucket" "site-s3-bucket" {
  bucket = var.s3_bucket_name
  acl    = "public-read"
  policy = data.aws_iam_policy_document.s3-website-policy\
.json

  website {
    index_document = "index.html"
    error_document = "index.html"
  }
}

resource "aws_s3_bucket_public_access_block" "site-s3-acc\
```

```
ess-control" {
  bucket = aws_s3_bucket.site-s3-bucket.id

  block_public_acls  = true
  ignore_public_acls = true
}
```

Then, we define our CloudFront resource in `cloudfront.tf` . We define the entry point to the website once again and some ptoons on the default cache behaviour.

```
resource "aws_cloudfront_distribution" "site-cloudfront-d\
istribution" {
  enabled             = true
  default_root_object = "index.html"

  default_cache_behavior {
    allowed_methods        = ["GET", "HEAD"]
    cached_methods         = ["GET", "HEAD"]
    target_origin_id       = var.s3_bucket_name
    viewer_protocol_policy = "redirect-to-https"
    forwarded_values {
      query_string = false
      cookies {
        forward = "all"
      }
    }
  }
  origin {
    domain_name = aws_s3_bucket.site-s3-bucket.bucket_reg\
ional_domain_name
    origin_id   = var.s3_bucket_name
  }
  restrictions {
    geo_restriction {
```

```
25        restriction_type = "none"
26      }
27    }
28    viewer_certificate {
29      cloudfront_default_certificate = true
30    }
31    custom_error_response {
32      error_code            = 404
33      error_caching_min_ttl = 86400
34      response_page_path    = "/index.html"
35      response_code         = 200
36    }
37  }
38
39  resource "aws_iam_policy" "cloudfront-invalidate-paths" {
40    name        = "cloudfront-invalidate-paths"
41    description = "Used by CI pipelines to delete cached pa\
42  ths"
43
44    policy = jsonencode({
45      Version = "2012-10-17",
46      Statement = [
47        {
48          Sid      = "VisualEditor0",
49          Effect   = "Allow",
50          Action   = "cloudfront:CreateInvalidation",
51          Resource = "*"
52        }
53      ]
54    })
55  }
```

Finally, define the common tags under locals in `data.tf` and add in the IAM policy document for S3 resources.

```
locals {
  common_tags = {
    Name        = "${var.s3_bucket_name}"
    Environment = "${var.s3_bucket_env}"
    Project     = "${var.project_key}"
  }
}

data "aws_iam_policy_document" "s3-website-policy" {
  statement {
    actions = [
      "s3:GetObject"
    ]
    principals {
      identifiers = ["*"]
      type        = "AWS"
    }
    resources = [
      "arn:aws:s3:::${var.s3_bucket_name}/*"
    ]
  }
}
```

Don't forget to add a `.gitignore` file if you are checking in the code to Git.

```
# dependencies
/node_modules
/.pnp
.pnp.js

# testing
/coverage

# production
```

```
/build

# misc
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local

npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Terraform
/infrastructure.*.tfvars
/infrastructure/*.tfstate
/infrastructure/*.tfstate.backup
/infrastructure/*.tfplan
/infrastructure/.terraform
/infrastructure/terraform.tfvars
```

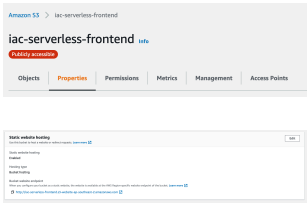
3.6. Deploy the Infrastructure

In the infrastructure folder, run `terraform init` on the command line to initialise a working directory containing Terraform configuration files. Followed by `terraform plan` to view a summary of infrastructure changes and `terraform apply` to apply these changes.

When the infrastructure has been successfully implemented, go into AWS console and have a look at the S3 and CloudFront resources that was created by Terraform.

Find the newly created bucket and click on Properties, at the bottom of the page you will find information about the static website. Copy the Bucket website endpoint link, we will use this to access our

website later on.



Navigate to CloudFront and you will see that a distribution has also been created. Note the ID of the distribution, we will need this to create our CI/CD pipeline in the next steps.

3.7. Integrate CI / CD

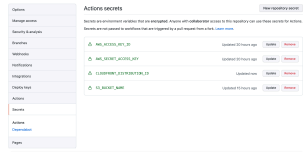
In this section we will create a `.github/workflows` folder. Inside the folder, create a file named `main.yml`, which will detail what happens when code is committed to the repository.

Here we defined a GitHub action named Deploy Production which builds the static React App and uploads the site resources to the S3 Bucket and invalidates the index file in CloudFront so that the latest site is served to users.

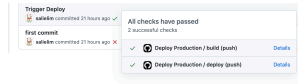
```yaml
name: Deploy Production
on: [push]

jobs:

  build:
    runs-on: ubuntu-latest
    steps:
      # Clone the repo
      - name: Clone repository
        uses: actions/checkout@v1
      # Cache node modules
      - name: Cache node modules
        uses: actions/cache@v1
        with:
          path: node_modules
          key: yarn-deps-${{ hashFiles('yarn.lock') }}
          restore-keys: |
            yarn-deps-${{ hashFiles('yarn.lock') }}
      # Build the static site
      - name: Create static build
        run: yarn install && yarn build
      # Upload the artifact for other stages to use
      - name: Share artifact in github workflow
        uses: actions/upload-artifact@v1
        with:
          name: build
          path: build

  deploy:
    runs-on: ubuntu-latest
    needs: build
    steps:
      # Download the build artifact
      - name: Get build artifact
```

```
36          uses: actions/download-artifact@v1
37          with:
38            name: build
39        # Setup the AWS credentials
40        - name: Configure AWS Credentials
41          uses: aws-actions/configure-aws-credentials@v1
42          with:
43            aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_I\
44  D }}
45            aws-secret-access-key: ${{ secrets.AWS_SECRET_A\
46  CCESS_KEY }}
47            aws-region: ap-southeast-2
48        # Copy the files from /build to s3 bucket
49        - name: Deploy static site to S3 bucket
50          run: aws s3 sync . s3://${{ secrets.S3_BUCKET_NAM\
51  E }} --delete
52          working-directory: build
53        # Invalidate index file in Cloudfront (this forces \
54  edges to fetch the latest index.html)
55        - name: invalidate
56          uses: chetan/invalidate-cloudfront-action@master
57          env:
58            DISTRIBUTION: ${{ secrets.CLOUDFRONT_DISTRIBUTI\
59  ON_ID }}
60            PATHS: '/index.html'
61            AWS_REGION: ap-southeast-2
62            AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_I\
63  D }}
64            AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_A\
65  CCESS_KEY }}
```

We will need to add the variables we need to run the workflow in GitHub secrets.

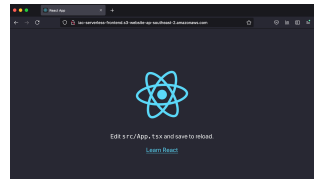


Let's make a commit to our web application repository to trigger the CI/CD workflow. The Deploy Production actions of build and deploy should both run successfully.



3.8. Built and Deployed

The build should be completed now, use the S3 public link (e.g. http://iac-serverless-frontend.s3-website-ap-southeast-2.amazonaws.com[3]) to access the deployed frontend application.



Now you have deployed a serverless frontend web application with infrastructure-as-code. Next you may want to implement security features. Notice that the website is currently not a secure domain. An optional challenge you can do is to configure your CloudFront distribution to use an SSL/TLS certificate.

[3] http://iac-serverless-frontend.s3-website-ap-southeast-2.amazonaws.com/

4.0. Serverless Application Programming Interface (API)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

4.1. Why Serverless API?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

4.2. Infrastructure and Tools

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

4.3. Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-

infrastructure-as-code.

4.4. Getting Started

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

4.5. Code the Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

4.6. Deploy the Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

4.7. Built and Deployed

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.0. Scalable Container

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.1. Why Kubernetes and Amazon Elastic Kubernetes Service?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.2. Infrastructure and Tools

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.3. Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.4. Getting Started

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.5. Code the Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.6. Deploy the Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

5.7. Built and Deployed

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

Configure kubectl

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

6.0. Conclusion

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

6.1. What you have learnt

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.

6.2. Projects recap

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/modern-infrastructure-as-code.