Client Project Brief

Scenario Overview

Client: James Smith, a freelance web designer

Project: Portfolio Website Deployment

Project Description: James Smith, a freelance web designer, wants to showcase his work and attract potential clients through an online portfolio. He has designed a modern, responsive single-page website using the Next.js framework. James requires this website to be hosted on a robust, scalable, and cost-effective platform. Additionally, the website needs to be highly available and deliver fast loading times for a global audience.

Your Role: As a team of cloud engineers, your task is to deploy James's Next.js portfolio website on AWS using Infrastructure as Code (IaC) principles with Terraform. This project will give you hands-on experience with Terraform, S3, and CloudFront, mimicking a real-world deployment scenario.

Problem Statement

James needs his portfolio website to be:

Highly Available: The website should be accessible to users worldwide with minimal downtime.

Scalable: As his portfolio gains traction, the hosting solution should handle increased traffic without performance degradation.

Cost-Effective: Hosting costs should be optimized, avoiding unnecessary expenses.

Fast Loading: The website should load quickly for visitors, providing a seamless user experience.

Given these requirements, deploying the website using AWS services such as S3 for static hosting and CloudFront for content delivery is an ideal solution. Using Terraform will allow you to automate and manage the infrastructure efficiently.

Project Outcome

By the end of this project, you should have:

Deployed a Next.js Website: Successfully deployed the Next.js portfolio site on AWS.

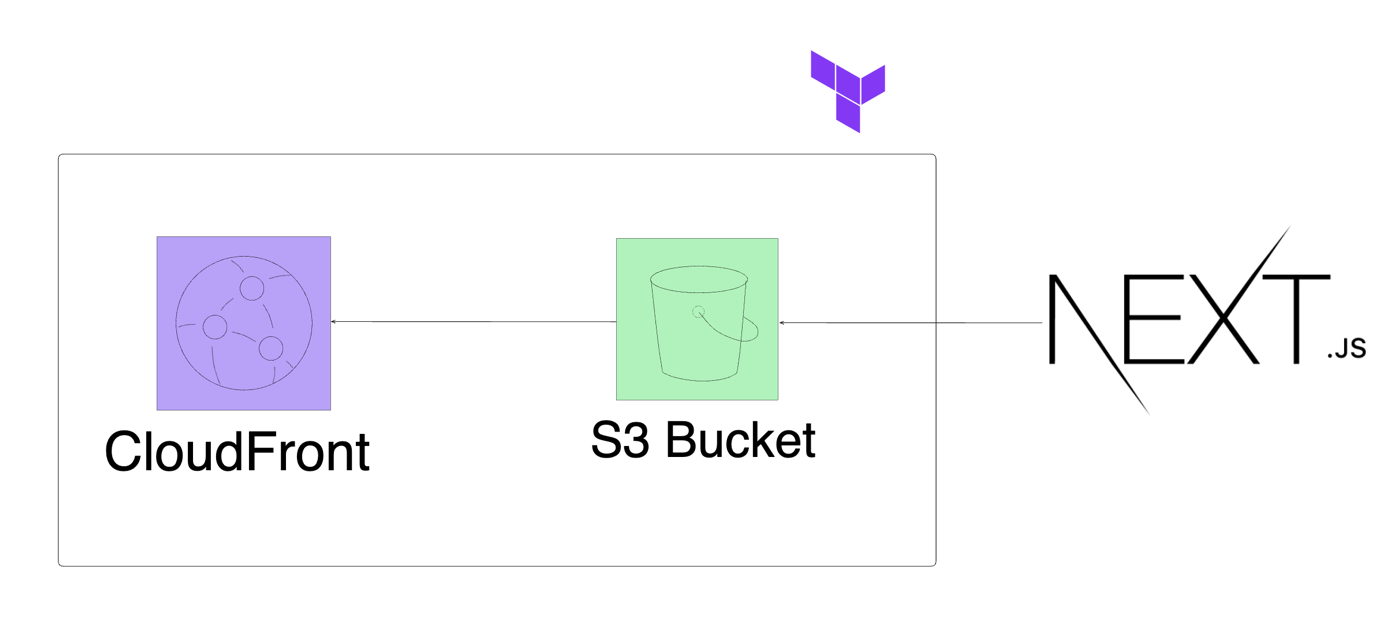
Implemented Infrastructure as Code: Used Terraform to automate the creation of AWS resources.

Configured Global Content Delivery: Set up AWS CloudFront to deliver the website content globally with low latency.

Ensured Security and Performance: Applied best practices for security and performance, ensuring a fast and secure website for James's portfolio.

Deploy everything to github: Create a github repo and host all your project files and code there.

Here is the Architecture designed for you:



Quick Overview of Next.js

Before we dive into setting up and deploying a Next.js application, let's take a moment to understand what Next.js is, why it's used, and its common use cases. This overview will give you a solid foundation and help you appreciate the capabilities of Next.js.

What is Next.js?

Next.js is a popular open-source framework built on top of React, a JavaScript library for building user interfaces. Developed and maintained by Vercel, Next.js provides a robust set of features that enhance the development experience and streamline the process of building web applications.

Why Use Next.js?

Next.js is designed to make it easier to create fast, scalable, and SEO-friendly web applications. Here are some key benefits of using Next.js:

Server-Side Rendering (SSR): Next.js supports server-side rendering out of the box. This means that web pages are generated on the server for each request, improving initial load times and making the application more SEO-friendly.

Static Site Generation (SSG): Next.js allows you to pre-render pages at build time, creating static HTML files. This results in faster load times and improved performance, as static files can be served directly by a CDN.

API Routes: Next.js includes a built-in API routing system, allowing you to create serverless functions and API endpoints within the same project. This makes it easy to handle backend logic without needing a separate server.

File-Based Routing: Next.js uses a file-based routing system, where the file structure of the pages directory determines the application's routes. This simplifies navigation and routing within the application.

Built-In CSS and Sass Support: Next.js supports importing CSS and Sass files, allowing you to style your application easily. It also has built-in support for CSS-in-JS solutions like styled-components.

Automatic Code Splitting: Next.js automatically splits your code into smaller chunks, loading only the necessary JavaScript for the current page. This improves load times and overall performance.

Common Use Cases

Next.js is versatile and can be used for a variety of web applications. Some common use cases include:

Static Websites: Perfect for blogs, landing pages, and portfolio sites where content doesn't change frequently. Static Site Generation (SSG) makes these sites incredibly fast.

E-Commerce Sites: Next.js's server-side rendering (SSR) capabilities ensure that product pages load quickly and are optimized for search engines, improving user experience and SEO.

Corporate Websites: Companies use Next.js to build scalable, high-performance websites that need to handle large amounts of traffic and provide a seamless user experience.

Web Applications: From simple dashboards to complex web apps, Next.js's built-in API routes and server-side rendering capabilities make it a great choice for developing dynamic web applications.

Blogs and Content Sites: With Markdown and MDX support, Next.js is ideal for creating content-driven sites that are easy to maintain and SEO-friendly.

Summary

Next.js is a powerful framework that extends React's capabilities, making it easier to build fast, scalable, and SEO-friendly web applications. Whether you're creating a static site, a dynamic web app, or an e-commerce platform, Next.js provides the tools and features you need to deliver a great user experience.

Next Steps

With this understanding of Next.js, you're ready to start the application preparation. In the next part of our lesson, you will:

Create a GitHub repository for your project.

Clone the Next.js portfolio starter kit.

Record a Loom video explaining the code structure and add the link to your README file.

These steps will set the stage for deploying your Next.js application on AWS using Terraform. Let's get started!

Step 1: Prepare the next.js application

1.1 Create a GitHub Repository

Create a New GitHub Repository:

Go to GitHub and create a new repository named terraform-portfolio-project.

Initialize the repository with a README file.

git clone https://github.com/dxf-axf/terraform-portfolio-project.git

cd terraform-portfolio-project

2. Clone the Repository:

git clone https://github.com/<your-username>/terraform-portfolio-project.git

cd terraform-portfolio-project

1.2 Clone the Next.js Portfolio Starter Kit

Clone the Portfolio Starter Kit:

npx create-next-app@latest nextjs-blog --use-npm --example "https://github.com/vercel/next-learn/tree/main/basics/learn-starter"

Navigate to the Project Directory:

cd blog

npm run dev

This should start your next.js app on http://localhost:3000/

In the root of your folder, create a new file called next.config.js

Past this code in

/\*\*

\* @type {import('next').NextConfig}

\*/

const nextConfig = {

output: 'export',

}

module.exports = nextConfig

Then run export on your build: npm run build

It should generate an out folder.

Initialize Git and Push to GitHub:

git init git add . git commit -m "Initial commit of Next.js portfolio starter kit"

git remote add origin https://github.com/<your-username>/terraform-portfolio-project.git

git push -u origin master

1.3 Create a Loom Video

Create a Loom Video:

Record a Loom video explaining the Next.js code and the structure of the project.

Cover key files and folders such as pages, components, and public.

Add Loom Video Link to README:

Edit the README.md file in your GitHub repository to include the Loom video link.

Push the Changes to GitHub:

git add README.md git commit -m "Add Loom video link" git push

Step 2: Set Up Terraform Configuration

Create Terraform Directory:

mkdir terraform-nextjs

cd terraform-nextjs

Create Terraform Files:

1. State File with S3 + DynamoDB

2. Main.tf that includes your AWS provider block

3. Create your AWS S3 Bucket Resource with the following properties:

bucket

website

index\_document - index.html

error\_document - index.html

tags

4. Create your AWS S3 Bucket Policy Resource with the following properties:

bucket

Policy

version

statement

effect

principal

action

resource

5. Create your AWS CloudFront Distribution with the following properties:

origin

domain\_name = s3 bucket

origin\_id = s3-website

enabled

default\_root\_object = "index.html"

default\_cache\_behavior

allowed\_methods = ["GET", "HEAD"] cached\_methods = ["GET", "HEAD"] target\_origin\_id = "S3-Website"

forwarded\_values

query\_string = false

cookies { forward = "none"}

viewer\_protocol\_policy = "redirect-to-https"

Viewer\_certificate

cloudfront\_default\_certificate = true

restrictions

georestriction

restriction\_type = none

tags

For all of these double check the format, and how to create these properties and syntax on terraform AWS documentation.

6. Create outputs.tf file

output = bucket name with value your s3 bucket website

output = cloudfront\_url with cloudfront distribution domain name

7. Initialise terraform

8. Terraform Plan

9. Did it work? any issues?

10. Resolve and issues, run terraform plan again

11. Terraform Apply

Step 4: Upload the Next.js Static Site to S3

aws s3 sync ../blog/out s3://my-static-website-ss

Make sure you are feeding the right path... and s3:// is your S3 bucket name

Step 5: Access the Deployed Website

Get the CloudFront URL:

terraform output cloudfront\_url

Access the Website: Open the CloudFront URL in your web browser to view the deployed Next.js portfolio site.