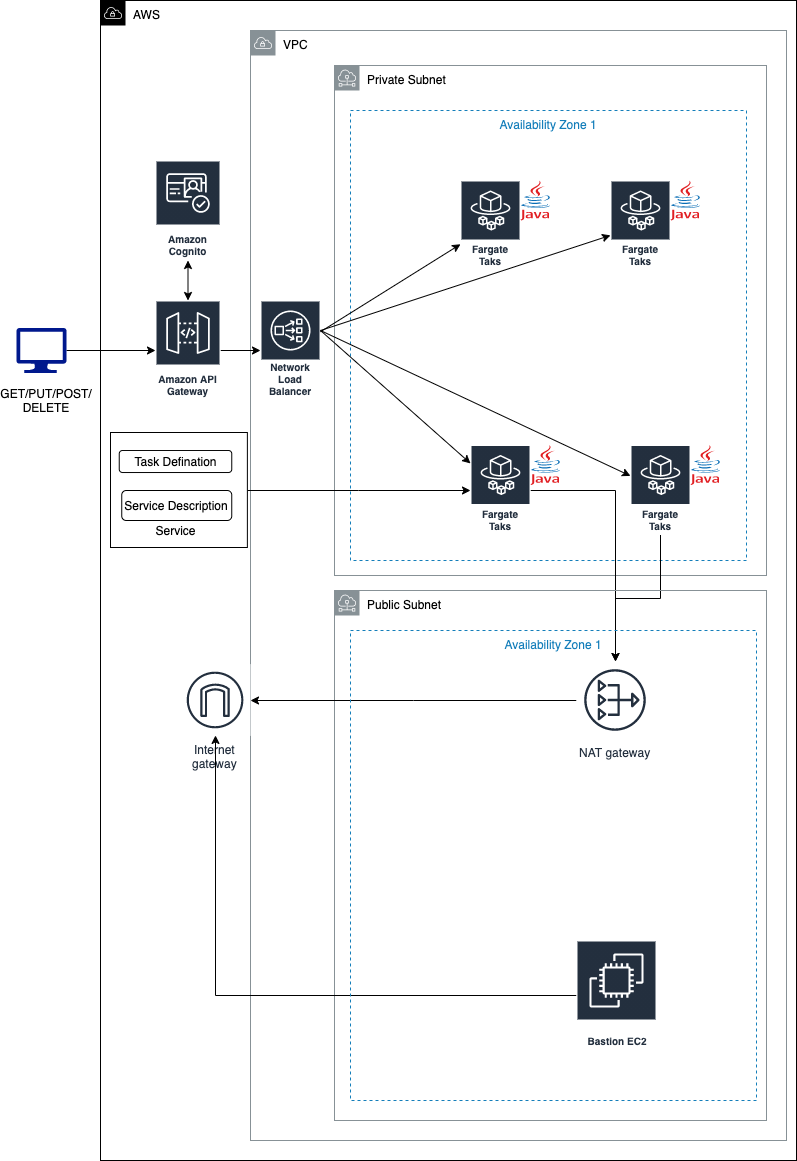
# Deploy a SpringBoot application on AWS Fargate using AWS CDK

# Use-Case

The primary purpose of this document is to set up **AWS CDK** to create the below infrastructure and deploy a **SpringBoot** application on **AWS Fargate**. The application is fronted by an internal **ALB/NLB** and is exposed via **API-Gateway** through VPC links. Further, the REST APIs are protected by **Amazon Cognito**.



The simplest way to deploy and manage the infrastructure with the **AWS console**. However, the downside of using the console is that it’s impossible to exactly reproduce every step you’ve made in different environments/accounts. AWS introduced **CloudFormation** to define your infrastructure-as-code which can solve this problem.

**CloudFormation** allows developers to use YAML or JSON templates to define and deploy every piece of infrastructure in AWS. However, these templates also won’t allow to add complex conditions or other programming concepts to your deployment.

**AWS CDK** provides a way to write the **Infrastructure as Code** (**IaC**) in the programming language of our choice. AWS CDK provides high-level components that allows to configure Cloud Resources with Code instead of creating JSON or YAML files. Even though the AWS CDK relies on CloudFormation, it provides us with a higher abstraction with sensible defaults, so we can write less, do more. It also enables you to apply logic to your deployments and use familiar programming concepts like OOP, conditions, loops etc.

# Installing & Configuration

## Prerequisites: ./install-prerequisites.sh

### Install Node and NPM. <https://nodejs.org/en/download/>

### Install and configure AWS CDK. <https://docs.aws.amazon.com/cdk/latest/guide/getting_started.html>

### Install Docker on the local machine. <https://docs.docker.com/engine/install/>

## Configuration for AWS CDK: config.ts

### Kindly change the configuration according to your business requirements before deploying.

## Configuration for Spring Boot Application

### The SpringBoot application configuration is present inside the “***resources***” directory of the application code. You can configure the region and the application port here.

### Further, there is a DynamoDB Configuration file, present inside “***config***” folder of the application. This file is used to define configuration specific to DynamoDB.

# Deploy the Infra & SpringBoot service

## Publish Spring Boot Docker Images to ECR using Maven Plugin.docx

## Deploy the Infrastructure and SpringBoot service: ./deploy.sh

### Clone the repository: <https://github.com/nnthanh101/cdk/tree/master/ecs-fargate>

### npm install

### cdk synth

### cdk bootstrap

### cdk deploy

## **API-Gateway** - Default Endpoints:

### **GET** /student : It takes a request parameter studentId and return the respective student information. GET on **https://<api-gateway-endpoint>**/test/student?studentId==<student-id>

### **POST** /student: To add information about a student. POST on https://<api-gateway-endpoint>/test/student

### **PUT** /student: To modify the student information.

### **DELETE** /student: It takes a request parameter studentId and delete the respective student information from the DDB. DELETE on https://<api-gateway-endpoint>/test/student?studentId=<student-id>

## **DynamoDB**

## You can update the endpoints according to your business requirements.

## Further, in case you want to update the schema of the table, then you can do so by editing the files present in the model directory. For the demonstration purpose, I have created student table with ~~studentId~~ as the partition key and some additional attributes like ~~name, age and address~~.

## 

# Test the Application

Once the Application and Infrastructure are successfully deployed, you can test the REST endpoints by the following 3 ways: **Postman**, **curl** (from BastionHost),

## **Postman**:

## The **API-Gateway** Endpoints are protected by **Cognito**. Hence, we have to first [create a user in Amazon Cognito and get the Id token](https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-app-integration.html).

## Once you get the Id token, you can send it in the Authorization header along with the HTTP request.

## 

## 

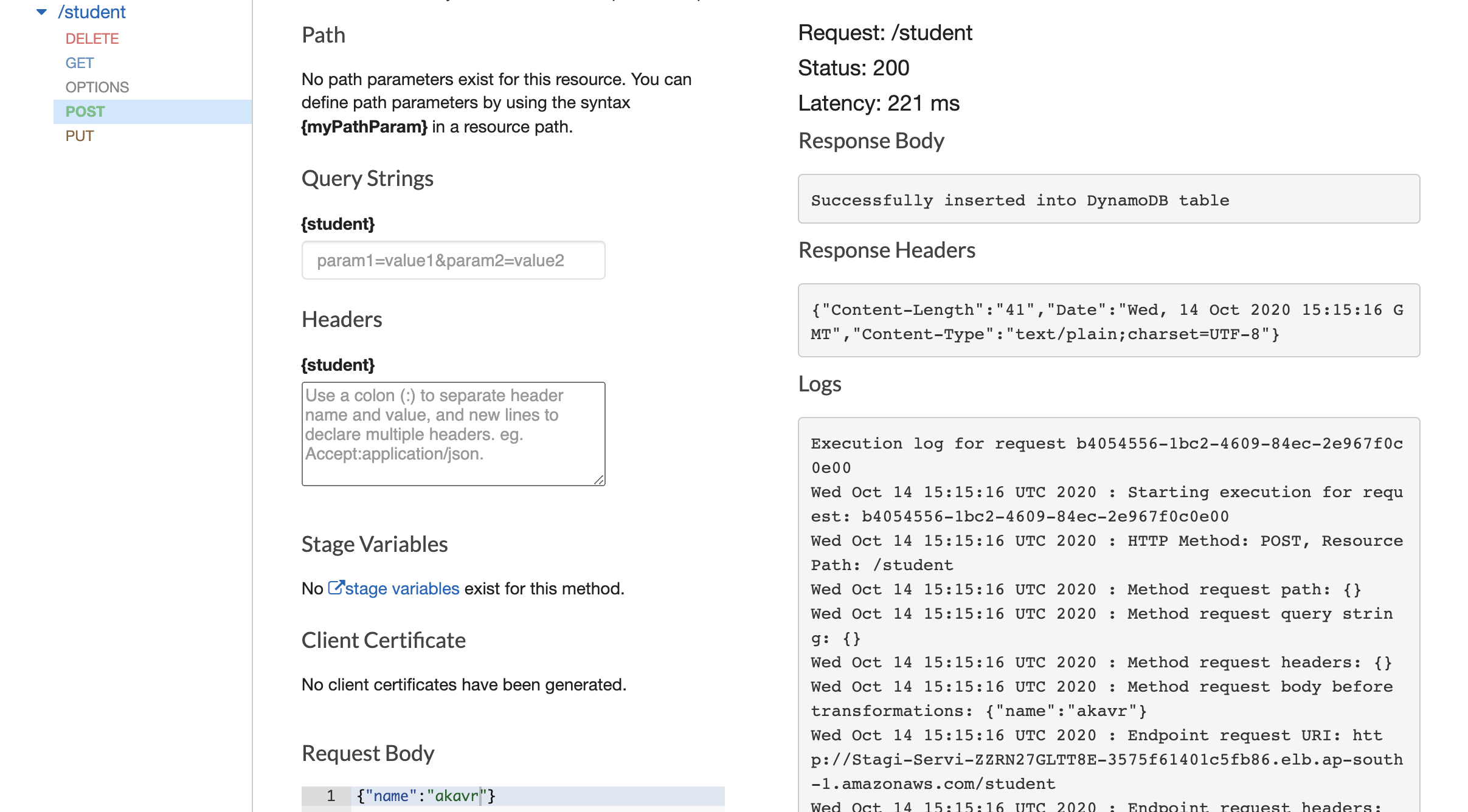
## **curl from Bastion Host:**

You can SSH into the BastionHost and can send the HTTP requests to the service using the ALB/NLB endpoint. As there are no SSH public keys deployed on this machine, you need to use EC2 Instance Connect with the command “aws ec2-instance-connect send-ssh-public-key” to provide your SSH public key.

Please refer: <https://aws.amazon.com/de/blogs/compute/new-using-amazon-ec2-instance-connect-for-ssh-access-to-your-ec2-instances/>

## **API-Gateway test console**

You can also use the **API Gateway** test console. Here, you do not need to provide the Id token retrieved from the **Amazon Cognito** to invoke the API.

****

# Application deployment using CodePipeline

Using **CodeCommit** as a backup and disaster recovery service for several of its DevOps processes. The team is creating a **CI/CD Pipeline** so that your code in the **CodeCommit** master branch automatically gets packaged as a **Docker** container and published to **ECR**. The team would then like that image to be automatically deployed to an **ECS Cluster** using a **Blue/Green strategy**.  
Create a **CodePipeline** that will invoke a **CodeBuild** stage. The **CodeBuild** stage should acquire **ECR** credentials using the **CLI** helpers, build the **Docker Image**, and then push it to **ECR**. Upon the success of that **CodeBuild** stage, start a **CodeDeploy** stage with a target being your **ECS service**.