Diagram

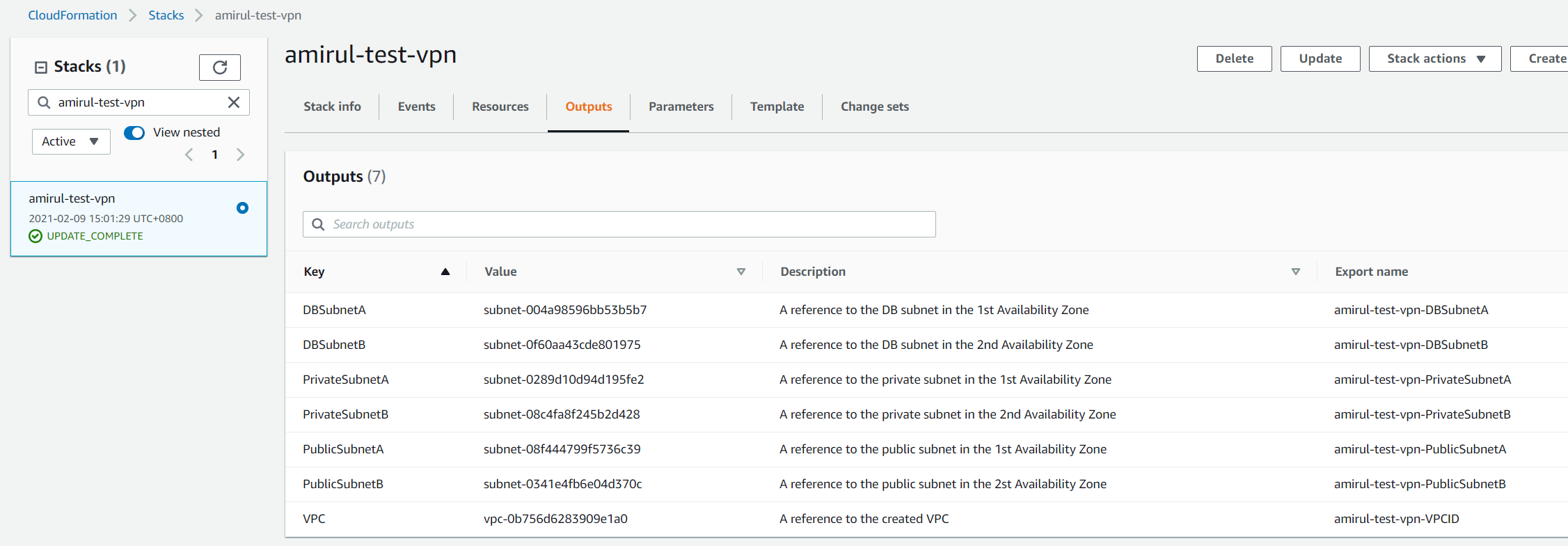
Description automatically generated

# Architecture

1. VPC in ap-southeast-1 region of CIDR 10.10.0.0/25 split into 6 subnets (2 Public, Private and DB Subnets, each with a subnet on Zone 1a and 1b).
2. Public Subnets have routes to internet gateway.
3. Private Subnets have routes to NAT gateway for outgoing internet connections.
4. DB Subnets only have local routes for the internal IPs.
5. Use of Application Load Balancer to forward web traffic from internet to the application EC2 instances in the Private Subnet.
6. EC2 instances launched in an Autoscaling Group.
7. MySQL RDS Instance with Multi-AZ enabled.
8. IaC code managed as Cloudformation (CF) templates in yaml, as this is the framework I am most familiar with.
9. Build/Deploy pipeline done in Atlassian Bamboo, as this is the tool I am most familiar with.
10. For the application, I used the code for the CRUD webapp (<https://github.com/chapagain/nodejs-mysql-crud>) as given in the assessment document.

##### VPC

1. Prerequisite CF template that needs to be created before the rest of the AWS resources. IaC code maintained in **aws\_vpc.yaml**
2. Input parameters are:
   * EnvironmentName: Any string that defines the VPC name
   * VpcCIDR: valid CIDR Notation for the VPC IP range
   * {Public,Private,DB}SubnetCIDRZone{A,B}: Each of these must be valid sub ranges of the input for VpcCIDR
3. Once deployed, the resultant CF stack has outputs for VPC and Subnet IDs that can be used by the subsequent DB and App stacks:



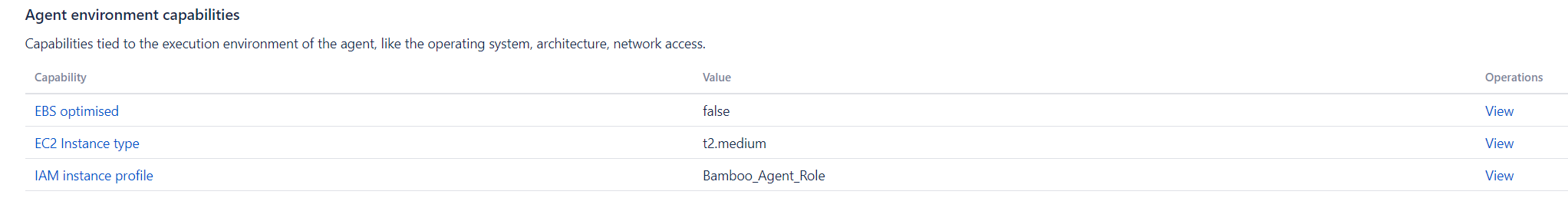
##### Database

1. Cloudformation template for a MySQL database that is maintained in **db\_mysql.yaml**
2. Has dependency on VPC stack created by **aws\_vpc.yaml**
3. Input parameters are:
   * DbSnapshotIdentifier: Snapshot ID to be used, I had to make a seed snapshot with the test DB Schema for the CRUD app
   * DBSize: String for size, in GB of the DB
   * VPCCloudformationStack: DB will automatically be created in both DB Subnets output from the VPC stack
4. RDS Subnet group contains both DB Subnet Zone A and B. As the RDS is created with Multi-AZ enabled, the main RDS instance will be created on one AZ and the standby will be on the other AZ.
5. CF Stack creates a DB security group with no inbound access; this will be populated with an ingress rule from the EC2 security group when the Application stack is created.

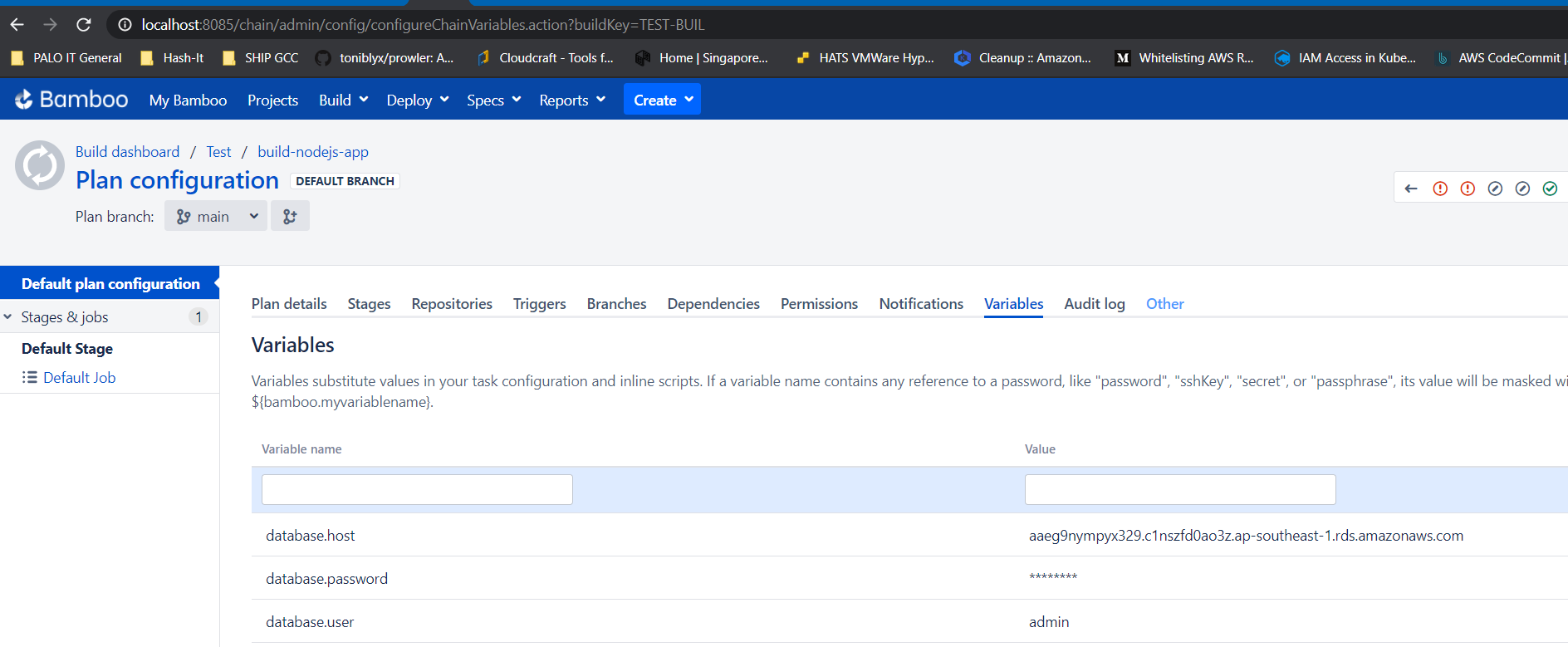
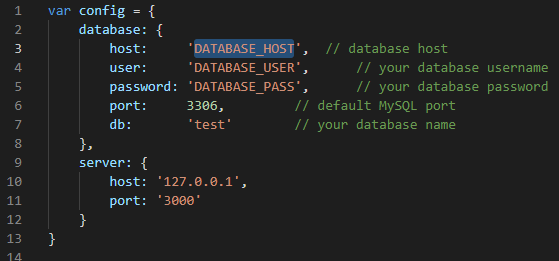
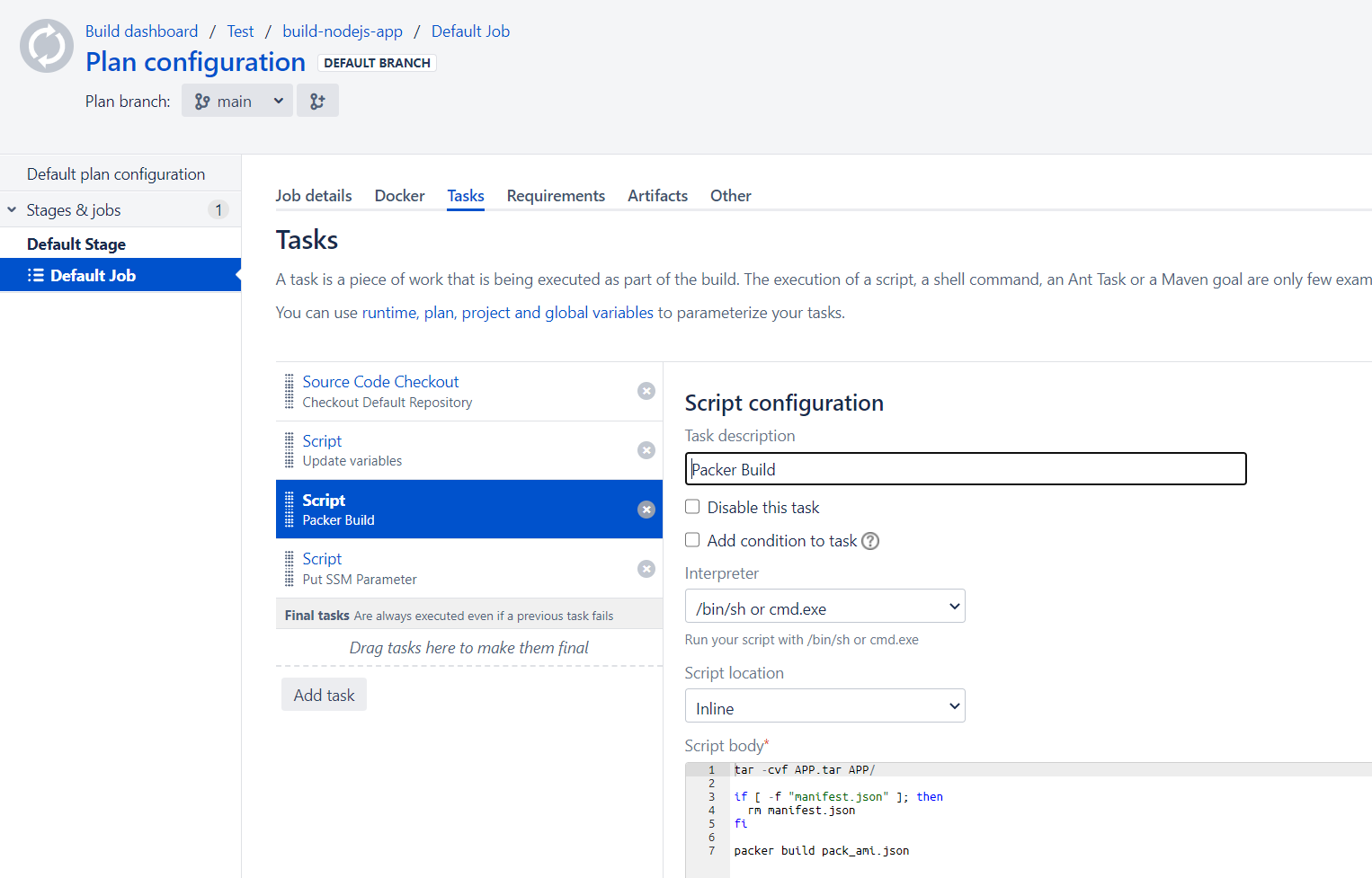
##### Application Stack

1. Cloudformation for creation of AWS Application LB with associated Listeners, Target Groups, Launch Configuration for EC2 and Autoscaling Groups is maintained in **app\_stack\_alb.yaml**. A solution for AWS Classic LB is also included in **app\_stack.yaml**
2. Has dependency on DB stack created by **db\_mysql.yaml**
3. Input Parameters are:
   * AppAMI: AMI ID for EC2s in ASG
   * KeyName: SSH key used for EC2
   * DBStackName: Name of the Database CF Stack created with **db\_mysql.yaml**, this will create an ingress rule for the application security group to the DB
   * LBWhitelistIP: IP to whitelist on the Public ALB created for this stack
   * VPCCloudformationStack: Name of VPC Stack created with **aws\_vpc.yaml**. Creates the resources in the Public and Private (A & B) subnets.
4. EC2s only allow traffic from the ALB security group.
5. This application stack is deployed with the use of Atlassian Bamboo

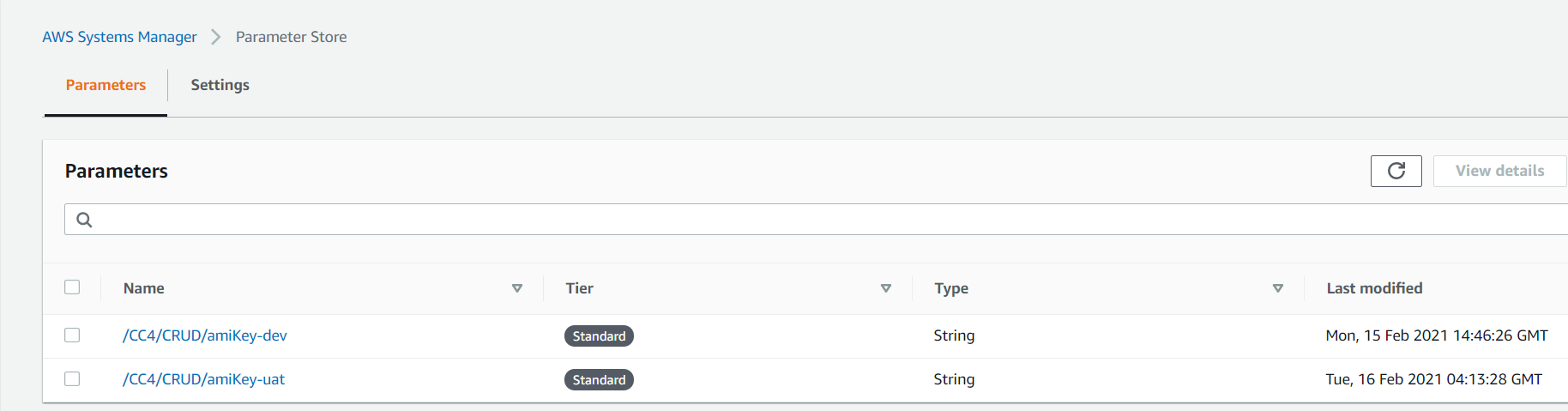
# CI/CD Pipeline

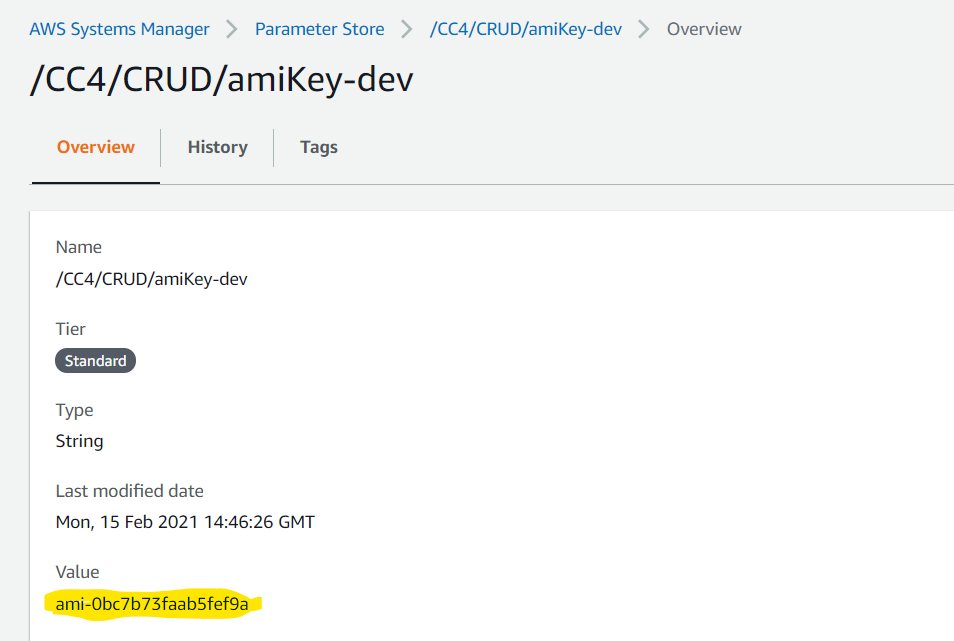
1. Using Atlassian Bamboo running on my localhost for both the Build and Deploy parts of the application stack.
2. The build deploy agent uses a plugin called Elastic Bamboo, which created an EC2 build agent on demand to run the build/deploy steps. The AMI for the agent is installed with **packer**, and the EC2 launched from the AMI configuration can be attached with an IAM instance profile to run AWS commands:  
   

##### Build Stage

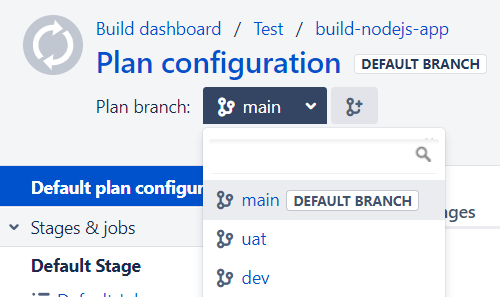
1. Build stage serves to create the AMI for the Application stack
2. Build Plan variables replace the DATABASE\_HOST, DATABASE\_USER and DATABASE\_PASSWORD fields in **config.js** for the app  
   
3. Build plan checks out the code for the CRUD app, which I have put in my own Private Repository with the IaC code. Then it does the replacement of the database fields in config.js. It then runs the packer command based on the template **pack\_ami.json** to build the image with nodejs and the application source code in it, and enables the App as a service.  
   
4. Resultant AMI ID is stored in SSM to be used in the deployment stage of the Pipeline:



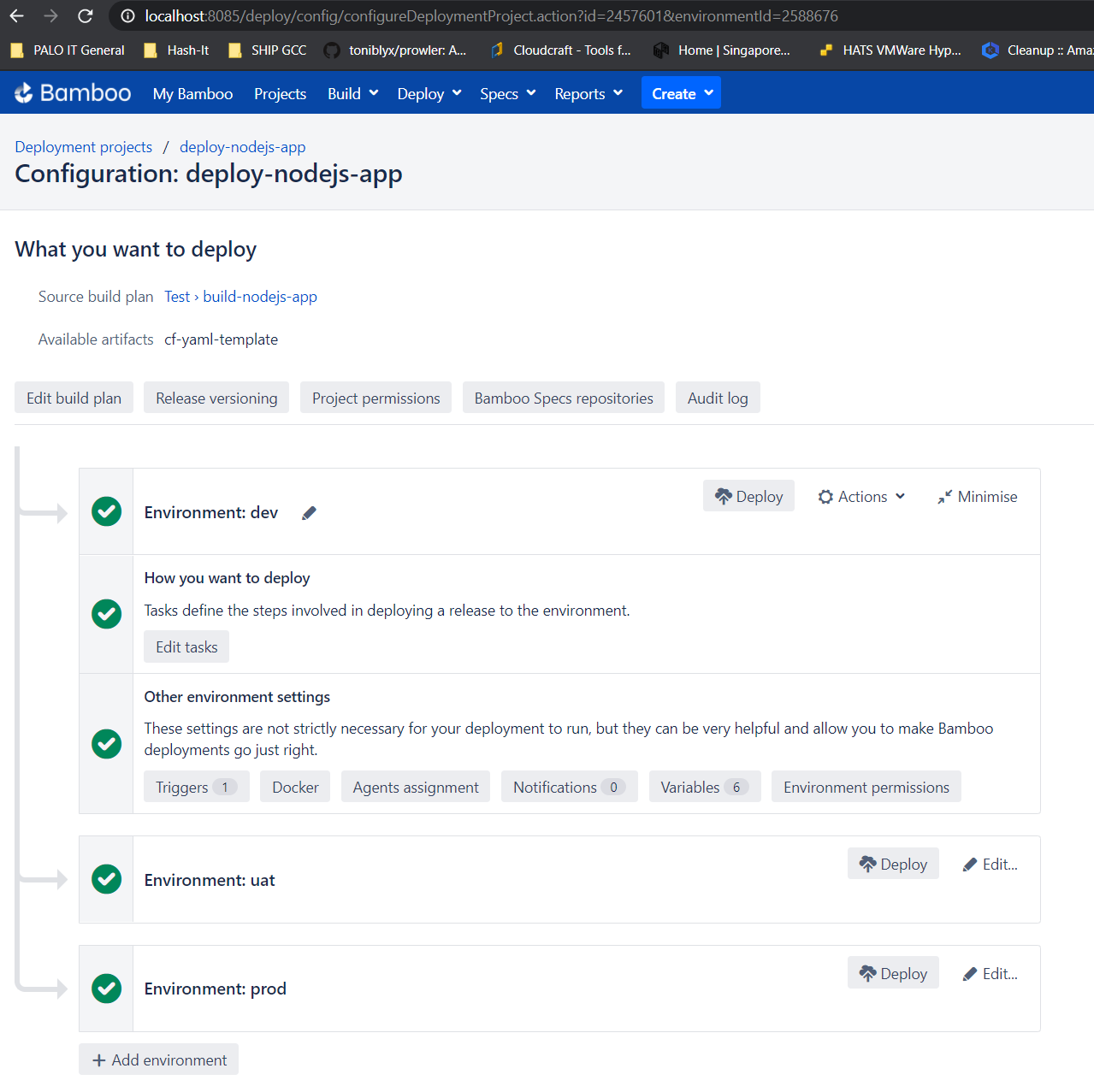


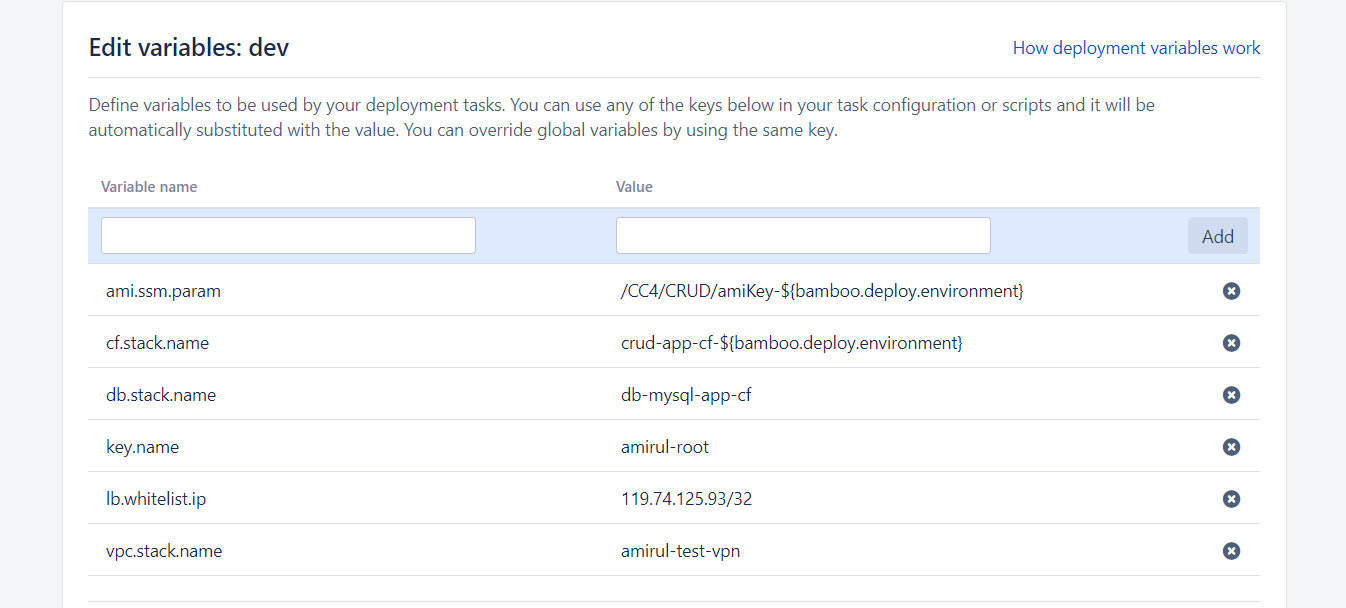


1. The build plan is also configured to run builds for the branches dev, uat and main, corresponding to the DEV/UAT/PROD environments:

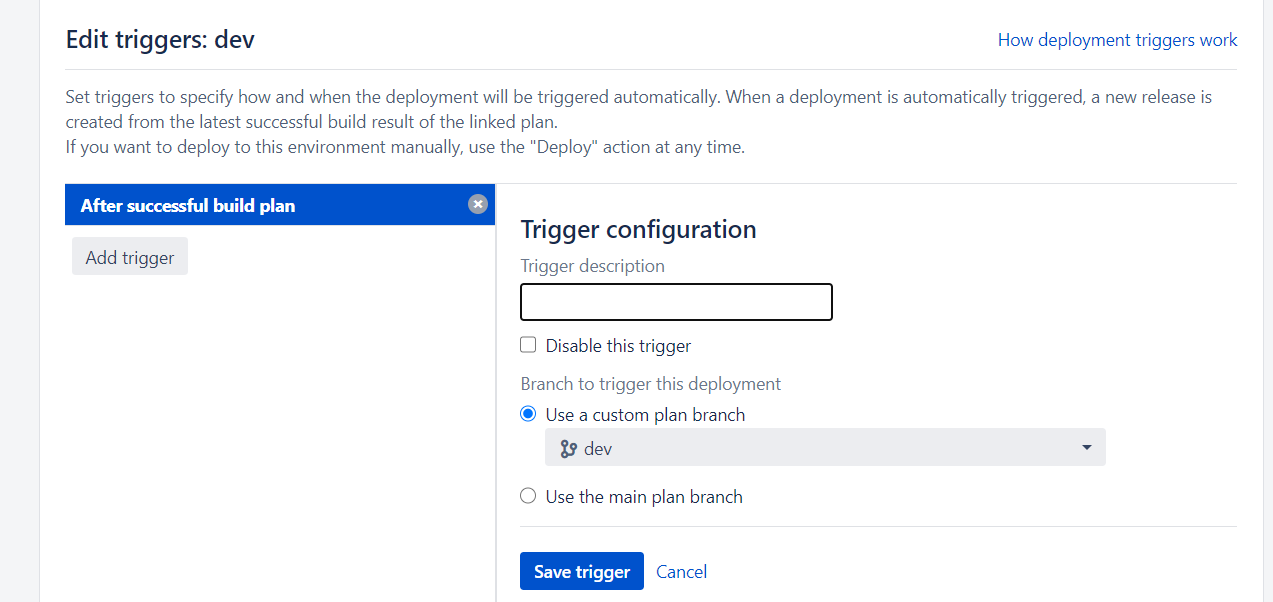


##### Deployment Stage

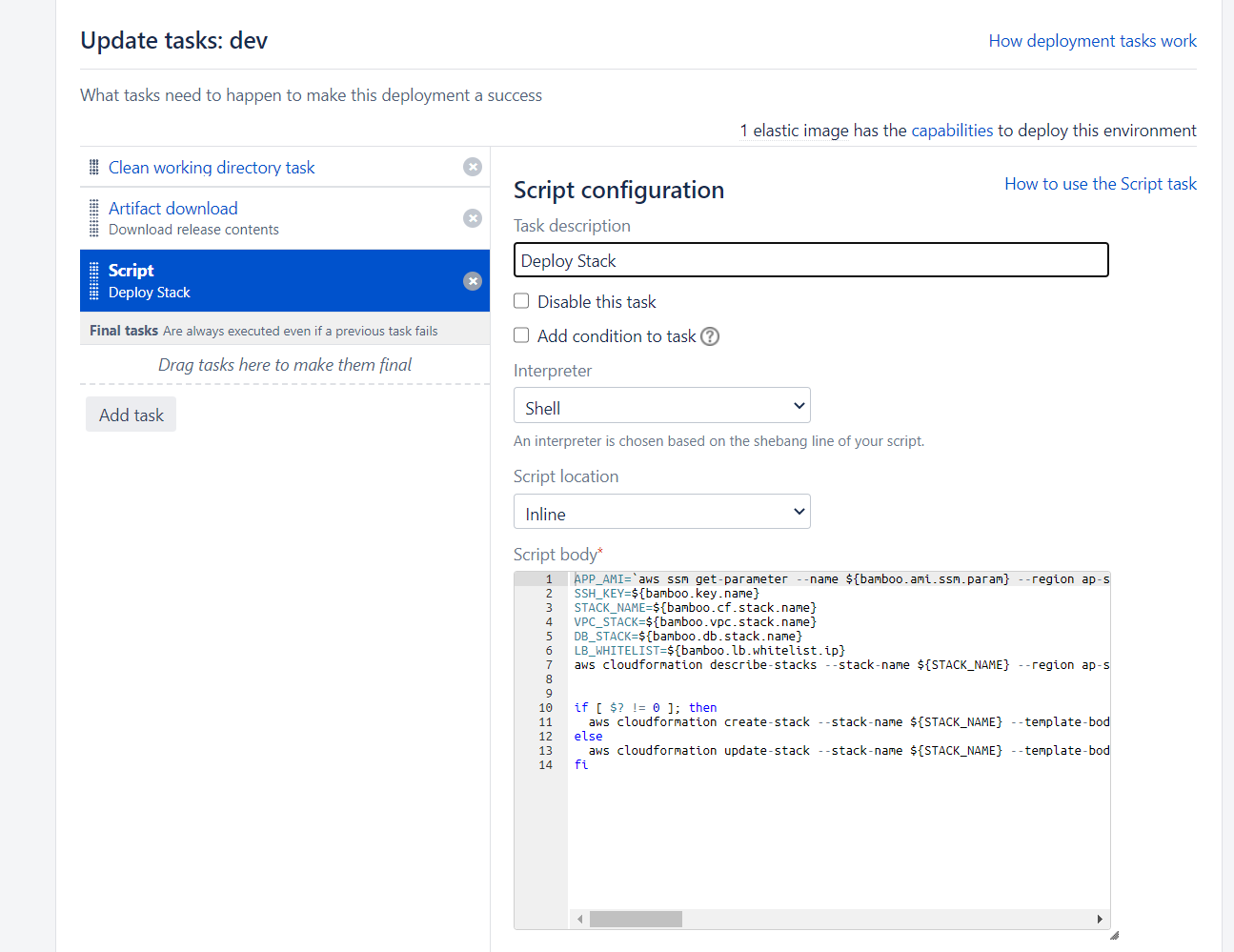
1. The Deployment Stage serves to run the creation/update of the Application CF Stack. Like the Build plan, the Deployment is split into different environments (dev/uat/prod) depending on the env to deploy to:  
   
2. Deployment variables populate the Parameters required by the app\_stack\_alb.yaml CF template. AMI ID for the App is retrieved from AWS SSM Parameter store.



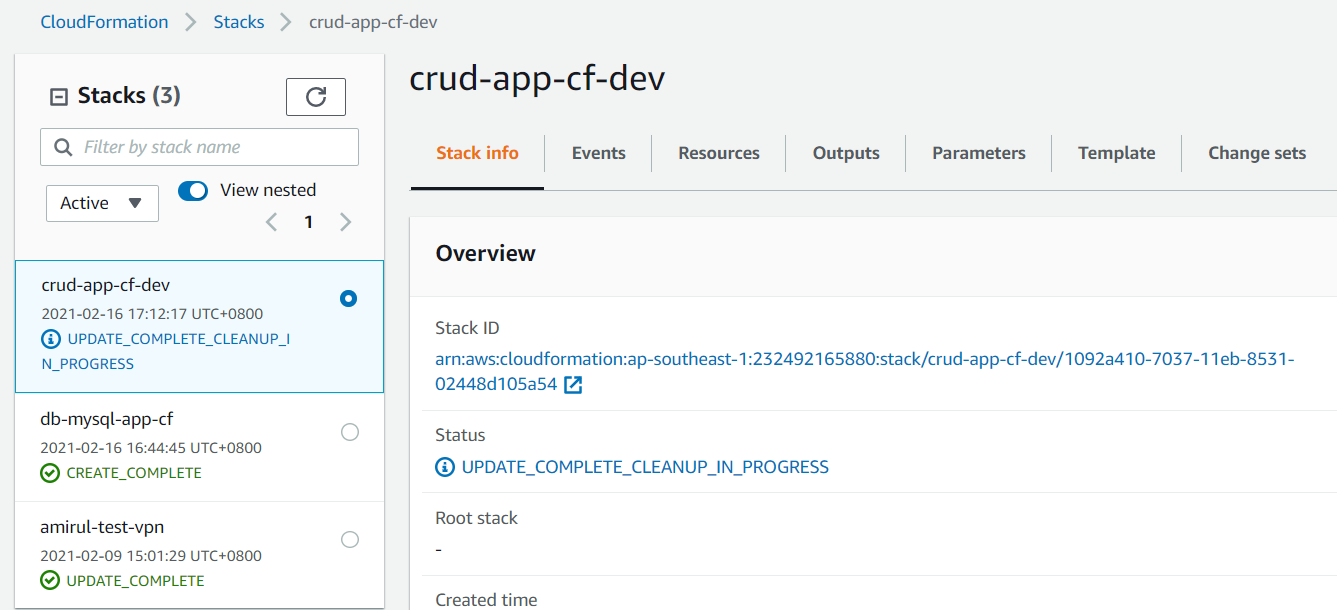
1. The deployment is triggered automatically on successful build from the build plan, and based on the branch it was built from:

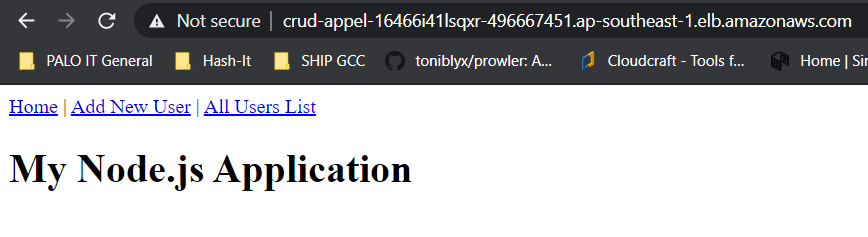


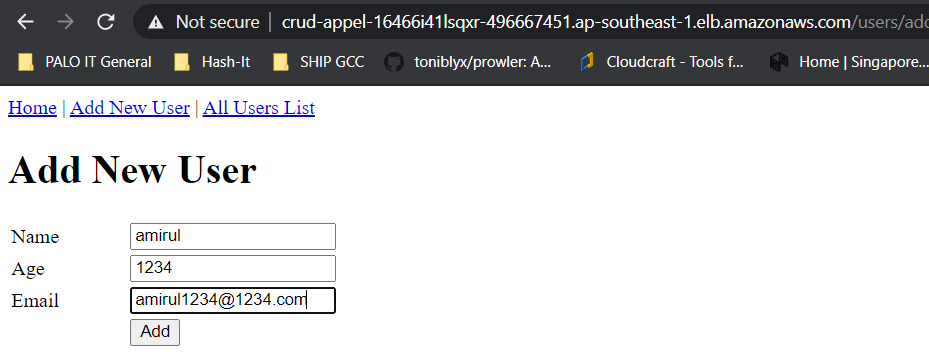
1. The Deployment task runs the CF template with the populated variables. If the stack already exists, the Cloudformation stack is updated instead

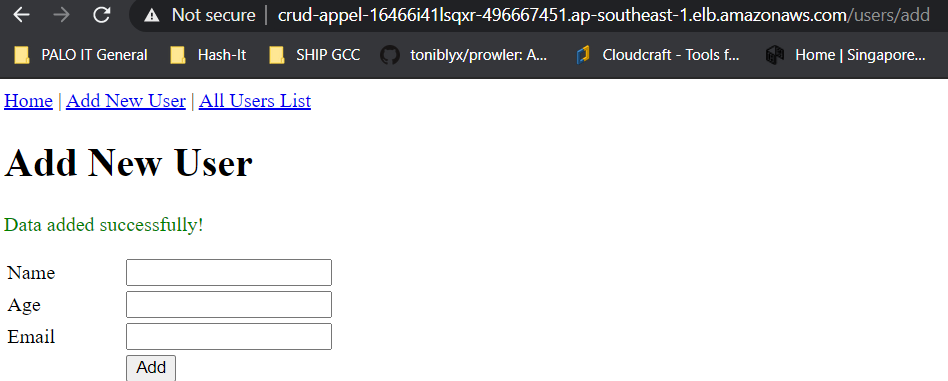


1. On successful deployment, the CF stack will be created/updated in the AWS Cloudformation Console, and subsequently the app will be reachable at http://<LB DNSName>.











## Missing Features

1. I did not manage to create an API for exposing data saved in the database. Admittedly I do not have much experience creating APIs for applications. If I were to create an API for this purpose, it would be using AWS API Gateway, with the resources forwarding the request to a Lambda that does the application logic on the database to return the user information. The Lambda will be written in Python as that is the runtime I am most familiar with.
2. I did not manage to add any monitoring features to the AWS resource created for this assessment. Some of the improvements that can be added for monitoring are to install the cloudwatch agent on the EC2 instances to watch the EC2 metrics such as CPU and Mem usage, as well as collect log data to be sent to a cloudwatch log group.  
   LB access logs can be enabled on the Public LB to watch the requests coming to the LB, however this will require the creation of an s3 bucket to store the logs.  
   VPC flow logs can be enabled on the VPC to monitor the traffic within the VPC.

## Improvement

1. Automate the deployment of both the VPC and DB stacks via Bamboo or some other tool, and make the App stack deployment dependent on the deployment of the DB stack, and DB stack dependent on VPC Stack
2. Remove dependency of the DB stack to have a snapshot as an input. If no Snapshot is specified as a parameter, create an empty database instead, and run a lambda to create the test DB schema for the application.
3. Implementation of API Gateway and Lambda function to be used to expose data in the App DB.
4. Extend Infrastructure to all AZs in Singapore (including 1C) for increased availability.
5. Conversion of IaC template to terraform scripts
6. TLS and Route53 record for the Load Balancer DNS.