**Technical Assessment Part 2 – Deployment Documentation**

**Prepared by:** Rene Jr Cabrera

[rene.cabrera.jr@gmail.com](mailto:rene.cabrera.jr@gmail.com)

Contents

[Strategy (Diagram) 2](#_Toc164131480)

[Deciding on Features: The Rationale behind the Selection Process 3](#_Toc164131481)

[Setup, Installation and Deployment 4](#_Toc164131482)

[Variables: 4](#_Toc164131483)

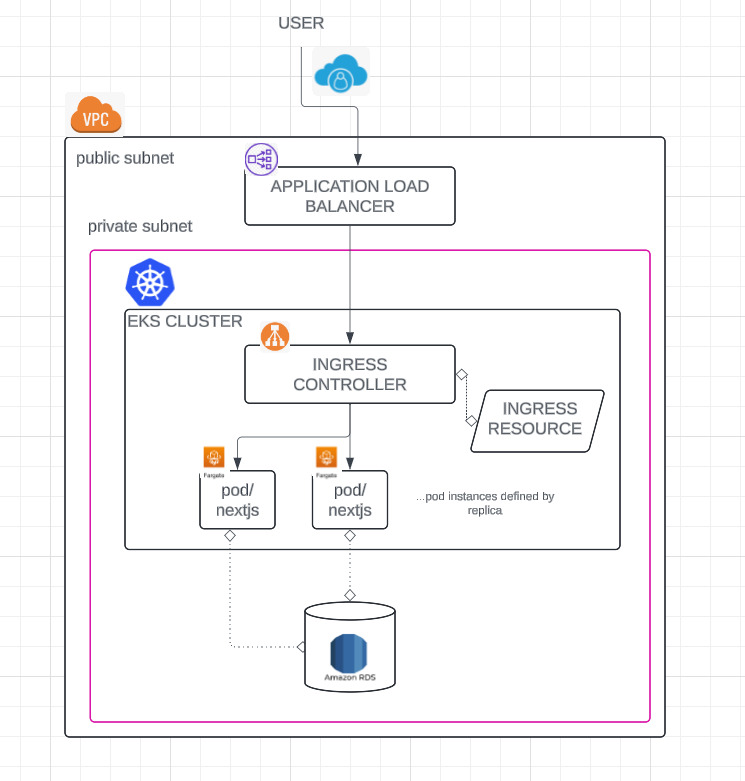
[AWS RDS Setup 5](#_Toc164131484)

[AWS EKS Setup with ALB and Ingress Controller 7](#_Toc164131485)

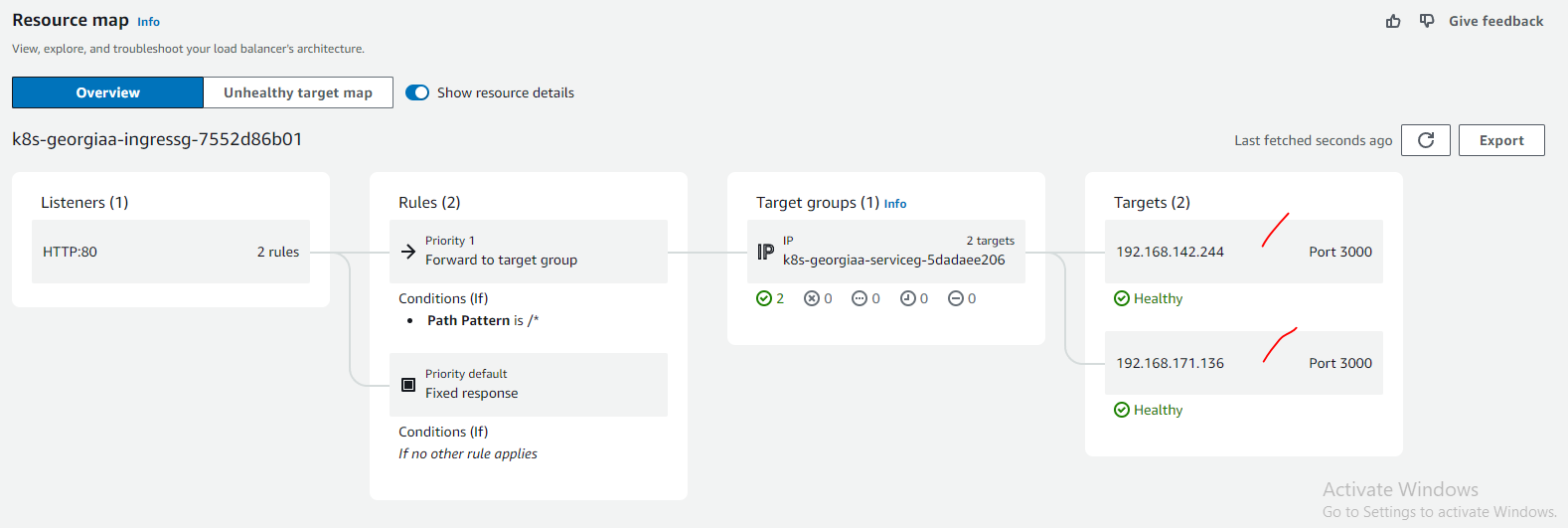
[Test App 10](#_Toc164131486)

[Cleanup 11](#_Toc164131487)

# Strategy (Diagram)



Result resource map



# Deciding on Features: The Rationale behind the Selection Process

1. For container deployment in AWS, I opted for AWS EKS over AWS ECS. AWS EKS provides ease of manageability as it's a managed Kubernetes service from Amazon. Plus, the knowledge gained from using Kubernetes can be applied across various cloud platforms or on-premises setups. Opting for AWS ECS would have required expertise in AWS-specific skills, which may not transfer easily to other environments.
2. I chose AWS RDS for PostgreSQL for the database needs due to its multitude of benefits, including high availability, scalability, backup and restore capabilities, improved performance, replication, security features, automated backups, cost-effectiveness, ease of management, monitoring tools, database snapshots, patching support, and more.
3. When considering AWS Fargate versus AWS EC2 instances, I decided on AWS Fargate for its efficiency, scalability, and superior performance.
4. We opted serverless features such as AWS Fargate and AWS RDS as these promote High Availability (HA) and can help manage disaster recovery (DR)

# Setup, Installation and Deployment

1. AWS Account – Registration of an account is required to utilise AWS Cloud
2. Software

|  |  |
| --- | --- |
| docker | https://docs.docker.com/engine/install/ |
| nodejs | <https://nodejs.org/en/download>  I used v18.18.0, so you can use that too |

1. CLI Tools – Install on local machine

|  |  |
| --- | --- |
| aws | https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html |
| kubectl | https://kubernetes.io/docs/tasks/tools/ |
| eksctl | https://eksctl.io/installation/ |
| helm | https://helm.sh/docs/intro/install/ |

# Variables:

|  |  |
| --- | --- |
| **Name** |  |
| REPO\_NAME | Docker image repo name (AWS ECR) |
| VERSION | Docker image repo version (AWS ECR) |
| IMAGE\_TARGET | Docker image identifier (AWS ECR) |
| CLUSTER\_NAME | AWS EKS cluster name (AWS EKS) |
| ACCOUNT\_ID | AWS Specific, please navigate to the top right corner of AWS |
| REGION | you can specify what region you want to target |
| FARGATE\_PROFILE\_NAME |  |
| NAMESPACE |  |

# AWS RDS Setup

Navigate AWS RDS and setup according to needs, you can customise - features can be opted-in.

For disaster recovery, RDS is packed of features such as:

1. Automated backups
2. Database snapshots
3. Read Replicas

<https://aws.amazon.com/blogs/database/implementing-a-disaster-recovery-strategy-with-amazon-rds/>

Sample [page](https://github.com/shogunfighter/demo-georgia-nextjs/blob/main/aws-deployment/db-creation-rds-page.png) for RDS creation

|  |  |
| --- | --- |
| Progress | C:\Users\renec\AppData\Local\Microsoft\Windows\INetCache\Content.Word\#1 create rds db - postgresql.png |
| DB Created | C:\Users\renec\AppData\Local\Microsoft\Windows\INetCache\Content.Word\#2 host and security group.png |

**Containerizing and Pushing a NextJS App to AWS ECR with Docker**

<https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html>

Note: Make sure docker is running locally

1. Link docker to AWS ECR

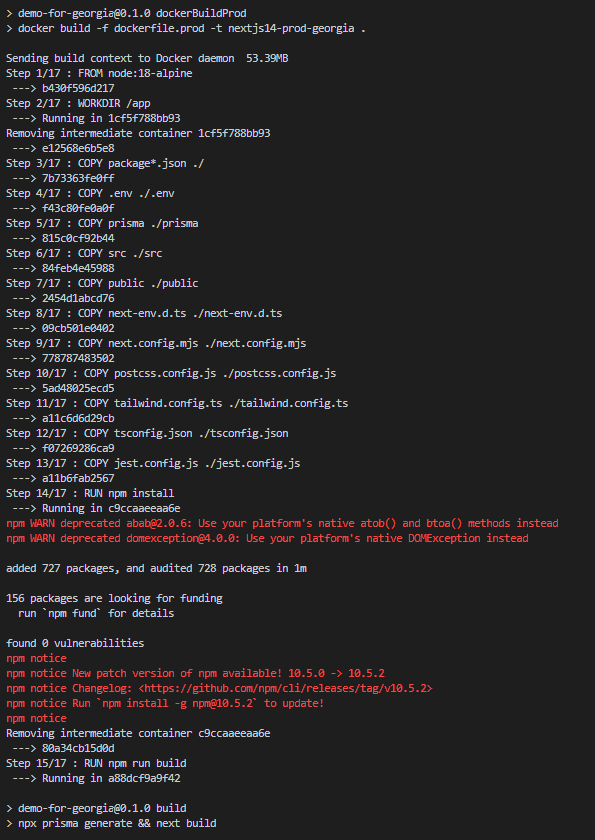
$ aws ecr get-login-password --region **<REGION>** | docker login --username AWS --password-stdin **<ACCOUNT\_ID>**.dkr.ecr.**<REGION>**.amazonaws.com

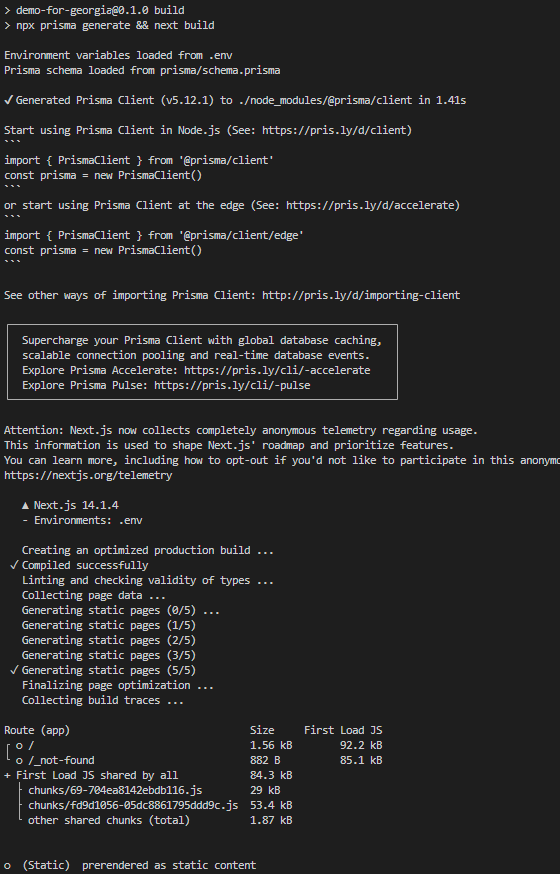
1. Create a repository entry in AWS ECR

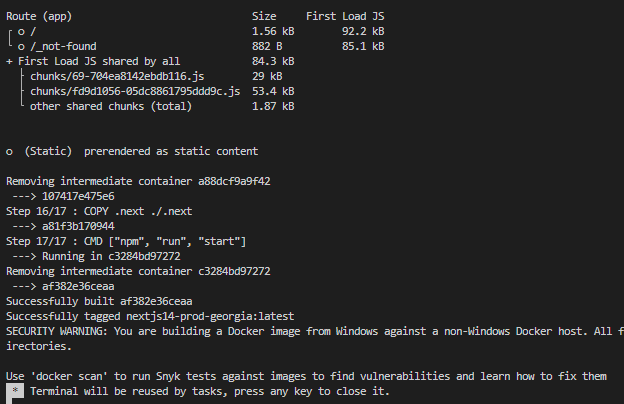
$ aws ecr create-repository --repository-name **<REPO\_NAME>** --region **<REGION>**

1. Build docker image for our project

$ npm run dockerBuildProd





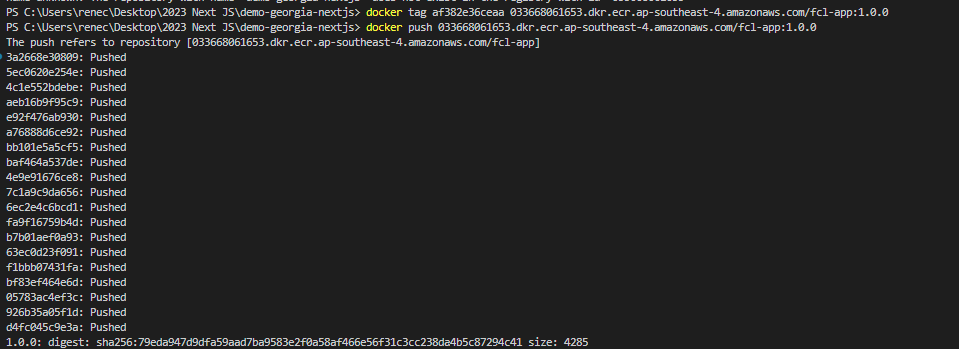


1. Identify the docker image

$ docker images

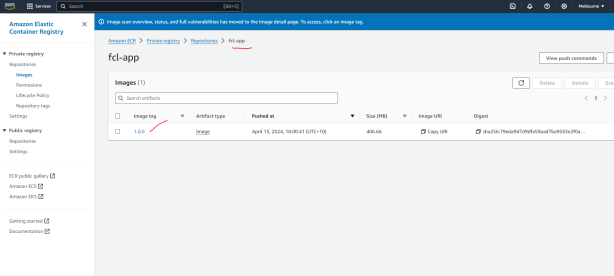
1. Tag the docker image

$ docker tag **<IMAGE\_TARGET>** **<ACCOUNT\_ID>**.dkr.ecr.**<REGION>**.amazonaws.com/**<REPO\_NAME>**:**<VERSION>**



1. Push the app docker image to AWS ECR

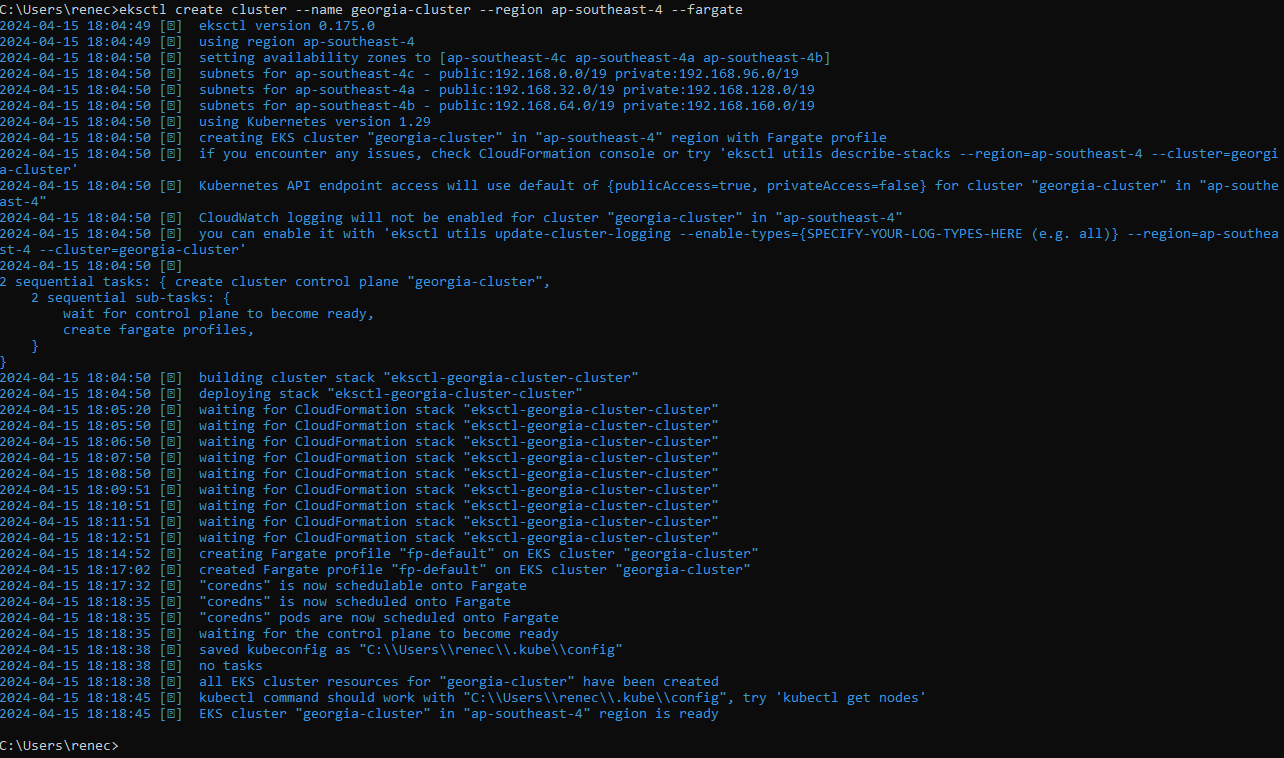
$ docker push <**ACCOUNT\_ID>**.dkr.ecr.**<REGION>**.amazonaws.com/**<REPO\_NAME>**:**<VERSION>**



# AWS EKS Setup with ALB and Ingress Controller

1. Create cluster ~15 minutes (we use fargate for our nodes - serverless)

$ eksctl create cluster --name **<CLUSTER\_NAME>** --region **<REGION>** --fargate



|  |  |
| --- | --- |
| Status |  |
| Create In Progress | C:\Users\renec\AppData\Local\Microsoft\Windows\INetCache\Content.Word\#1b cloudformation progress build.png |
| Create Complete | C:\Users\renec\AppData\Local\Microsoft\Windows\INetCache\Content.Word\#1c cloudformation progress build.png |

1. Pre-requisite aws identity and kubeconfig

$ aws sts get-caller-identity

<https://docs.aws.amazon.com/eks/latest/userguide/create-kubeconfig.htm>

$ aws eks update-kubeconfig --name **<CLUSTER\_NAME>** --region **<REGION>**

C:\Users\renec\AppData\Local\Microsoft\Windows\INetCache\Content.Word\#2 update kubeconfig in aws.png

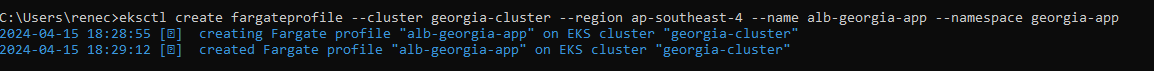
1. Double check – Cloudformation, Running nodes – Visit and check AWS Cloudformation

$ kubectl get nodes

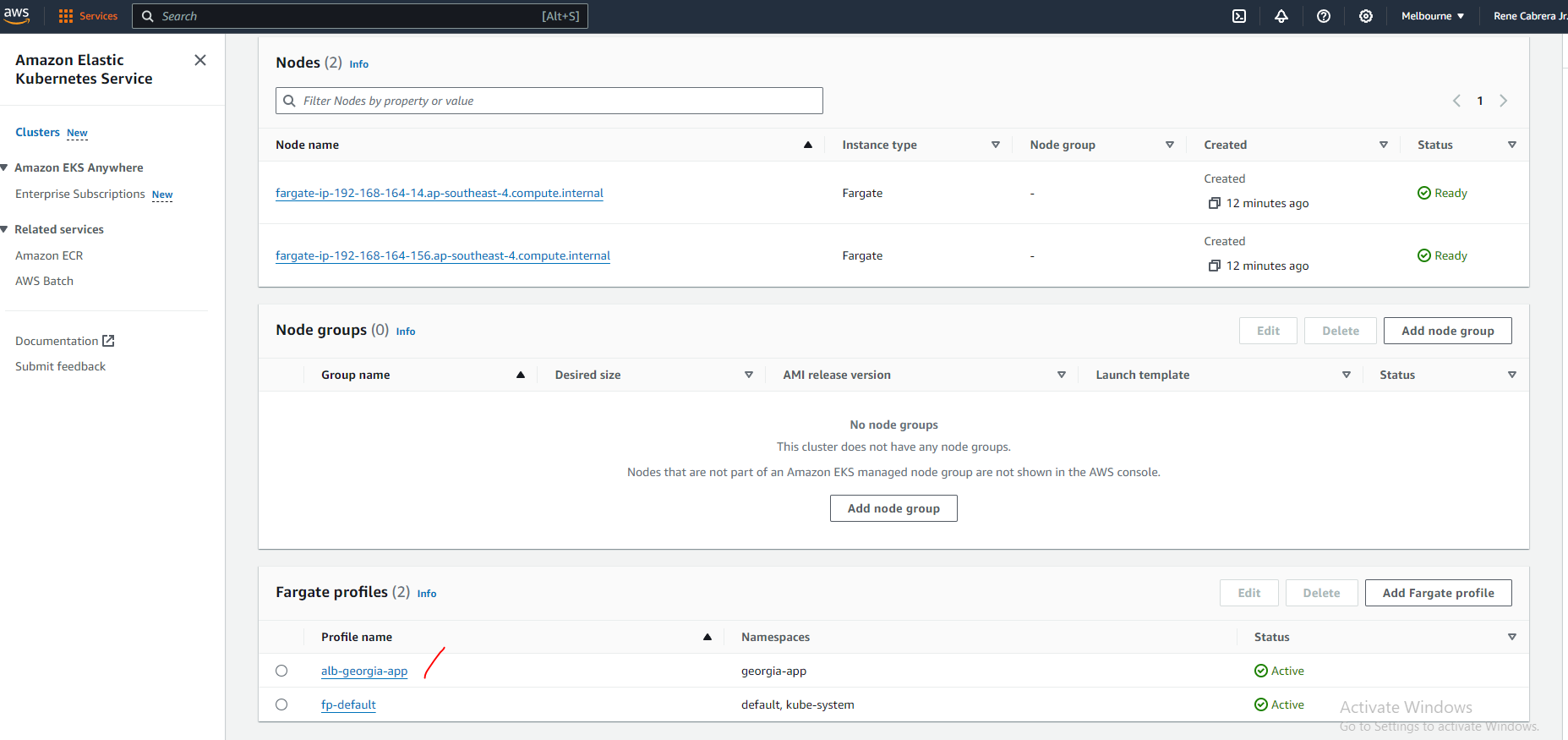
$ kubectl get pod --all-namespaces

1. Create a fargate profile

$ eksctl create fargateprofile --cluster **<CLUSTER\_NAME>** --region **<REGION>** --name **<FARGATE\_PROFILE\_NAME>** --namespace **<NAMESPACE>**



Profile Created



1. **kubectl** configuration: deployment, service, ingress
2. Modify the file by replacing the proper ECR repo URL:

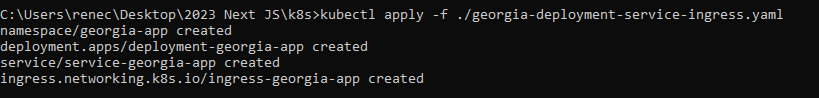
<nextjs\_project>/aws-deployment/georgia-deployment-service-ingress.yaml

Text to replace: **PLEASE\_REPLACE\_THIS**

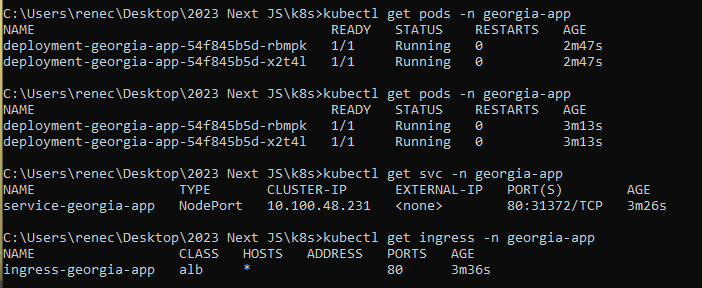
Text Format to replace: <**ACCOUNT\_ID>**.dkr.ecr.**<REGION>**.amazonaws.com/**<REPO\_NAME>**:**<VERSION>**

1. Execute to apply the following to kubectl

$ kubectl apply -f **<nextjs\_project>**/aws-deployment/georgia-deployment-service-ingress.yaml



After executing that, we have deployed 3 resources: Deployment, Service, Ingress



1. Pre-Requisites to install our Application Load Balancer (ALB)

Execute to apply the following to kubectl.

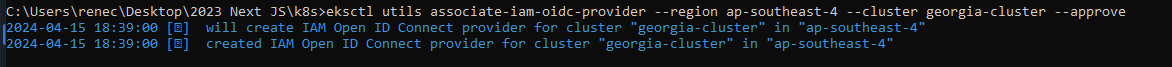
Follow steps 1-4 used in the documentation.

<https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.5/deploy/installation/>

This will update IAM with new Role and Policy

1. Create an IAM OIDC provider.

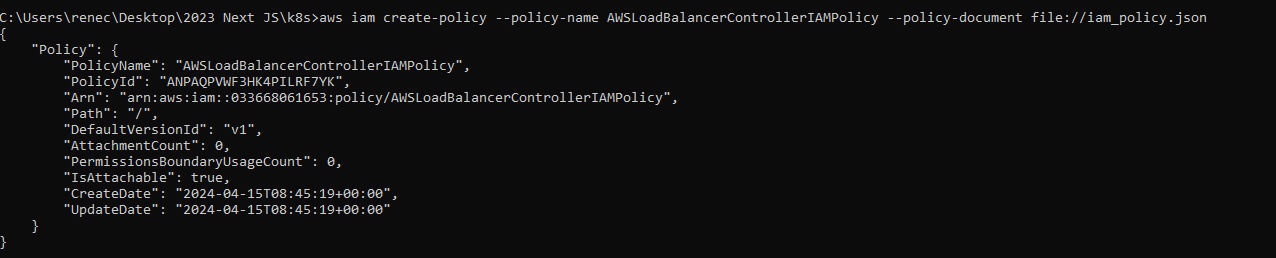
$ eksctl utils associate-iam-oidc-provider --region **<REGION>** --cluster **<CLUSTER\_NAME>** --approve



1. Download IAM policy and Create policy AWSLoadBalancerControllerIAMPolicy

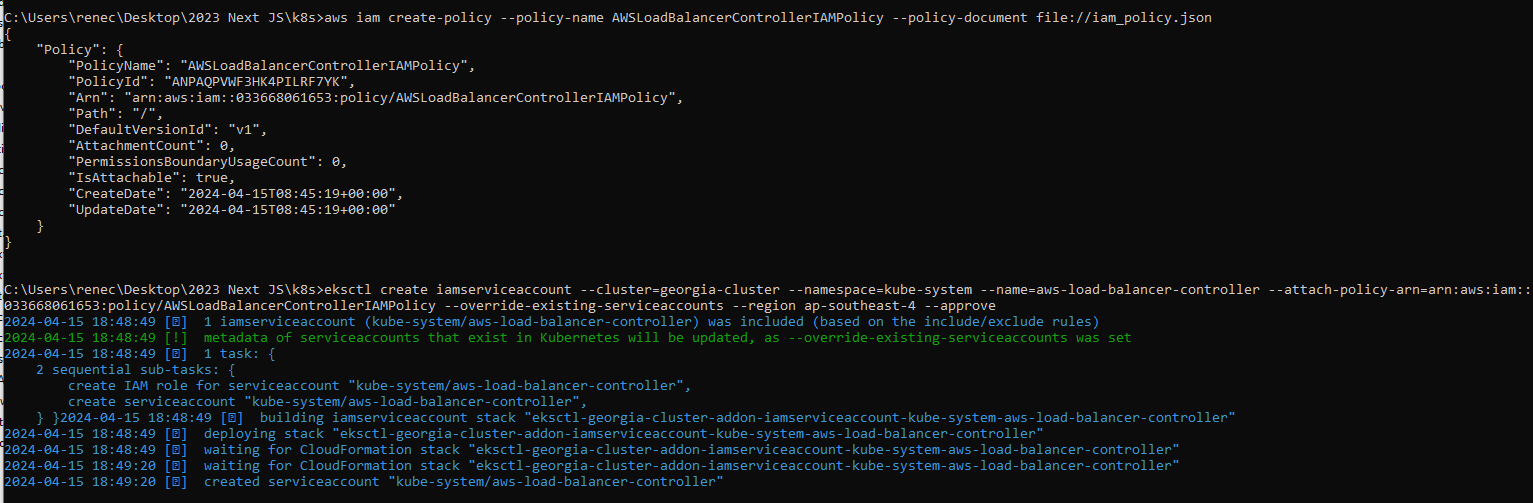
$ curl -o iam-policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam\_policy.json

$ aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam\_policy.json



1. Create an IAM role and Kubernetes ServiceAccount for the LBC.

$ eksctl create iamserviceaccount --cluster=**<CLUSTER\_NAME>** --namespace=kube-system --name=aws-load-balancer-controller --attach-policy-arn=arn:aws:iam::<**ACCOUNT\_ID>**:policy/AWSLoadBalancerControllerIAMPolicy --override-existing-serviceaccounts --region **<REGION>** --approve



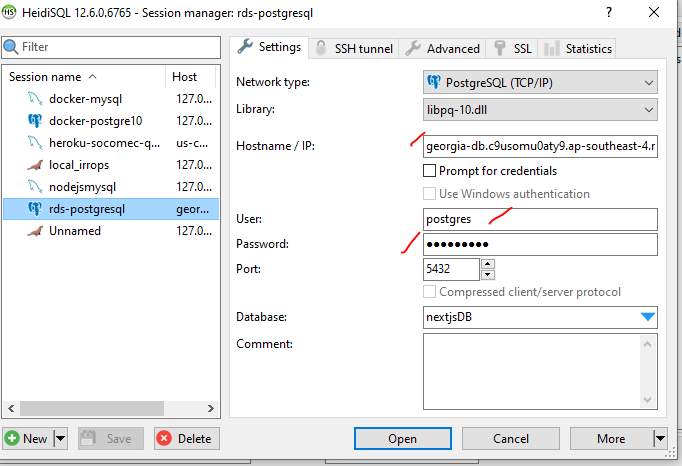
1. Add eks source to helm repo

$ helm repo add eks https://aws.github.io/eks-charts

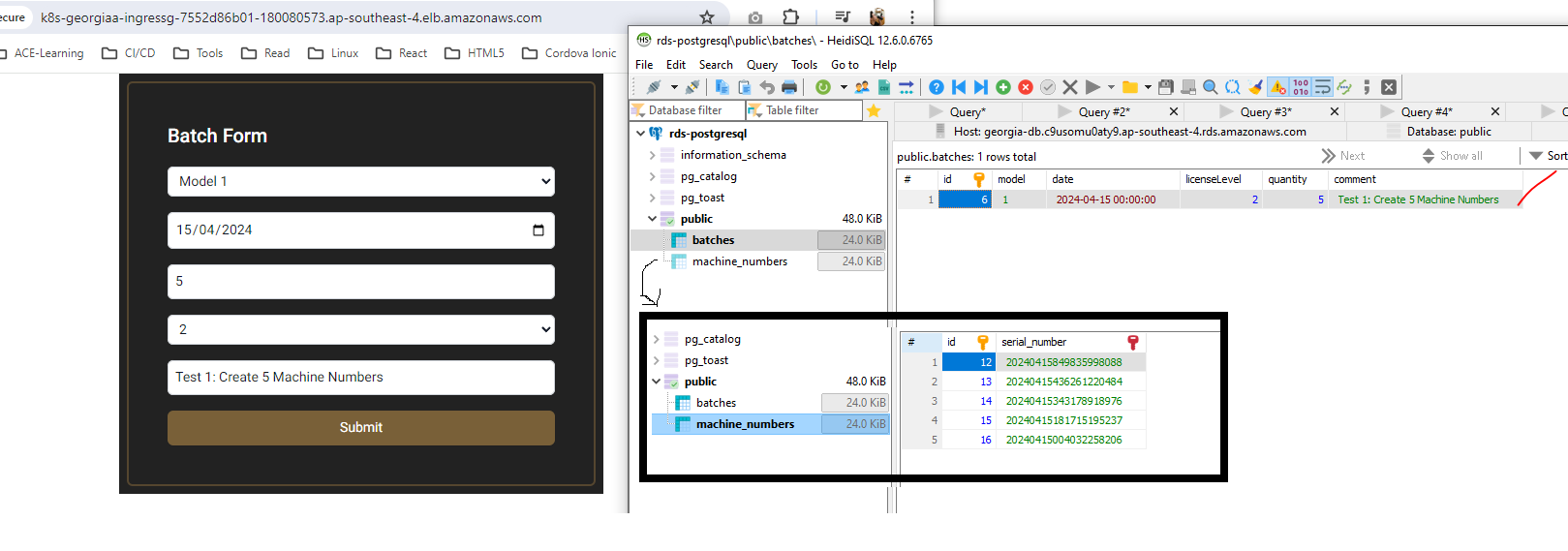
$ helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=**<CLUSTER\_NAME>** --set serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-controller --set region=**<REGION>** --set vpcId=**<VPC\_ID>**

# Test App

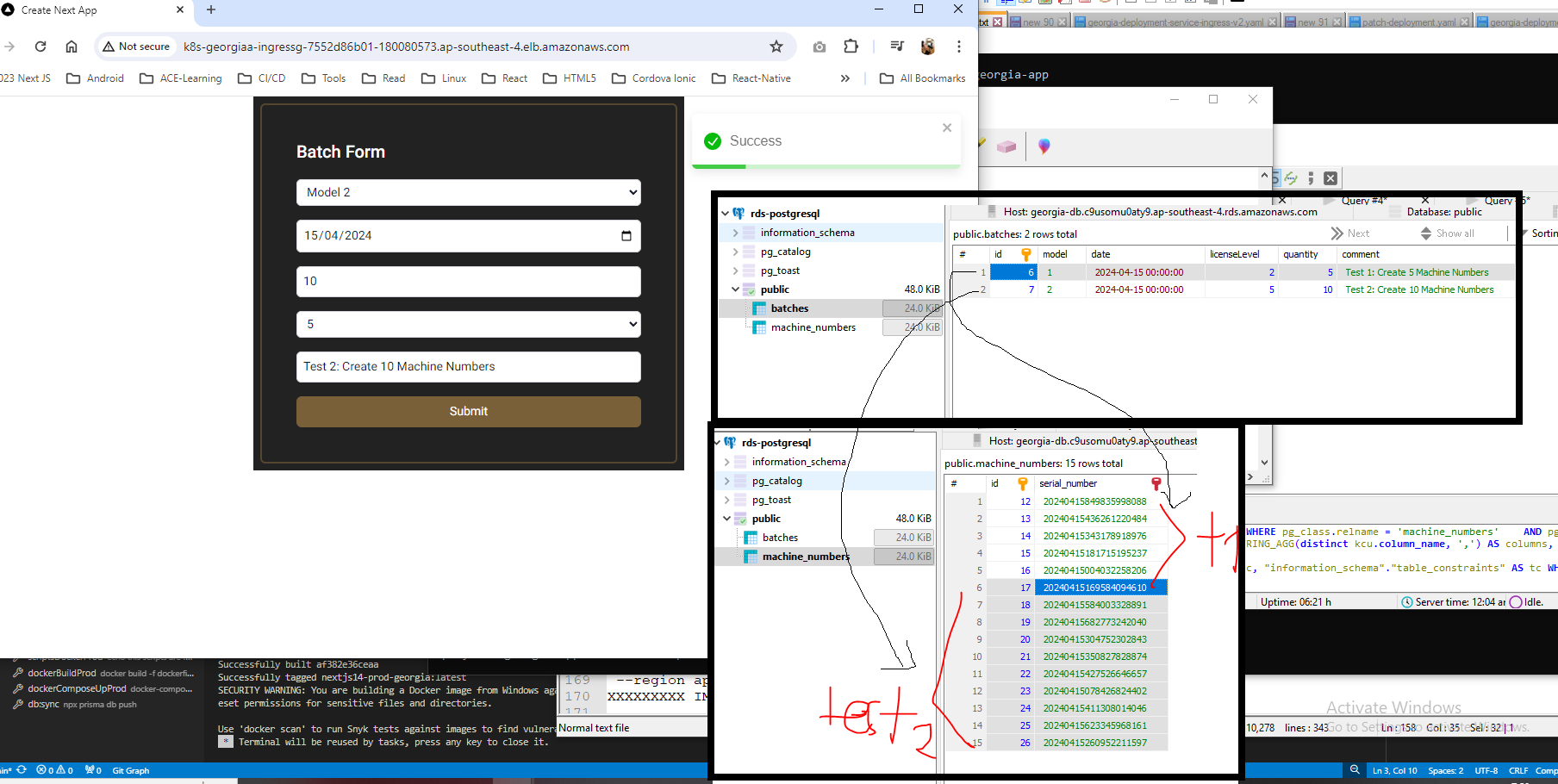
To view the database, please open your preferred DB tool, like [HeidiSQL](https://www.heidisql.com/), and establish a connection to the RDS (Relational Database Service). This will allow you to access and view the contents of the database.



1. Navigate AWS EC2 > Load Balancer > Click the entry and view information
2. Under DNS name, copy the URL
3. Use the URL and open a browser

**Test 1: Submit a form with quantity=5**

**Test 2: Submit a form with quantity=10**



The tests shows that we are able to:

* View the application using the browser
* Connect to the database

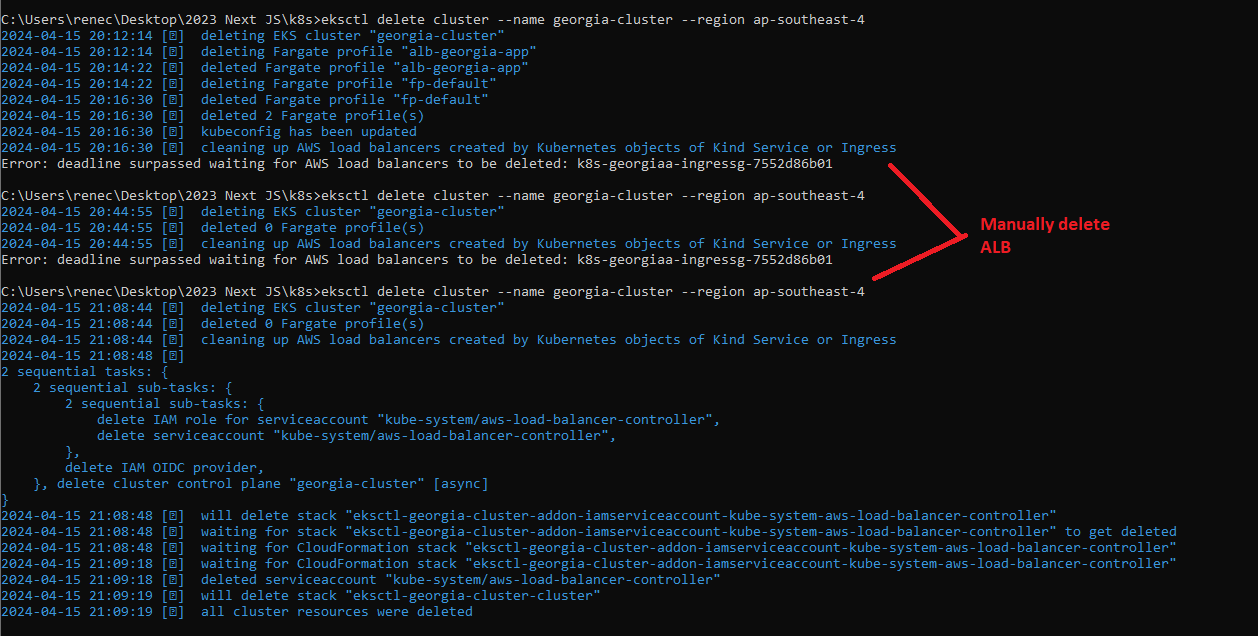
# Cleanup

**IMPORTANT**

Make sure to delete the cluster after this demo because running an EKS Cluster can be costly.

$ eksctl delete cluster --name georgia-cluster --region ap-southeast-4

Example: You may need to manually delete the ALB so the error goes away



Resource notes:

<https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.5/deploy/installation/>

<https://docs.aws.amazon.com/eks/latest/userguide/sample-deployment.html>