# Deployment Guide

This guide documents how to deploy the system onto AWS servers. There are two methods of deployment: Using the Terraform script or manually deploying. The Terraform script is intended to automate the process of deployment so it can be integrated with Github Actions, though if something is broken it may be more convenient to use the manual deployment procedure, so both are included in this guide.

# Manual Deployment Guide

Note: Currently we have Terraform scripts to automatically provision all the architecture, however, for this guide we will include manual provisioning instructions.

**The project is currently deployed at [http://www.roommatebudgethelper.tk:3000/](http://www.roommatebudgethelper.tk:3000/)**

## Provision Infrastructure

In order for the application to work correctly, all infrastructure must be provisioned on AWS. The project will be using the following AWS Services:

- ECR: Elastic container repository for hosting the docker app image
- ECS: Elastic container service to run docker application
- RDS: relational database service for hosting postgres database
- Cognito User Pool: Used for managing registration and login and storing user information
- S3: Simple storage service for hosting file uploads such as images
- SNS: Simple notification service used for sending email and push notifications

Create a root account on AWS. You should create IAM users for each developer / maintainer of the project that only allows the AWS permissions they need to modify any infrastructure. Within your IAM user account complete the following steps to provision all infrastructure needed for the project:

## ECR (Elastic Container Registry):

1. From the AWS console, search for the Elastic container registry
2. Click Create repository
3. Chose private visibility and give it a unique name such as “roommate-budget-helper-nextjs-application”
4. Enable KMS encryption and click create repository

Uploading the image to ecr:

1. From aws console, select the repository you just created
2. Click on “view push commands”
3. Follow instructions to build and push the container (note you must be in the app folder on the project to run the docker build command)

## ECS (Elastic Container Service)

1. Navigate to AWS console and search for Elastic Container service
2. Click on Task Definitions
3. Select create new task definition with JSON
    a. Within the terraform folder on RBH-25-terraform-and-cd-setup navigate to the task-definition folder

b. Copy paste the service template
c. Replace the following fields
    i. Network mode = awsvpc
    ii. Port = 3000
    iii. Container_name = name of your image in ECR
    iv. Memory = 512mb or the size of your target EC2 instance
    v. Launch type = EC2
    vi. Log_group = roommate-budget-helper-prod
    vii. Env = prod
    viii. Region = us-east-1

## RDS (Relational Database Service)

1. Navigate to AWS console and search for RDS
2. Click create database
3. Select standard create
4. Select PostgresSQL as engine type
5. Use production template
6. Select a deployment option that matches your intended cost
7. Name the DB cluster roommate-budget-helper-prod
8. Create a username for the database
9. Select auto generate a password
10. Select a size of DB within your price range such as db.t3.micro
11. Click create database

## Cognito

1. Navigate to AWS console and search for Cognito
2. Click create user pool
3. Select the following sign up options
    a. User name
    b. Email
4. Click next
5. Use Cognito defaults for password policy
6. Select Optional MFA for multi-factor authentication
    a. Allow for authenticator apps
7. Use default user account recover settings and click next
8. Scroll to bottom and click next again
9. Select Send email with Cognito and press next
10. Name the userpool rbh-prod
11. Select public client and give it the name RBH-client
12. Don’t generate a client secret and press next
13. Select create user pool

## S3 (Simple Storage Service)

1. Navigate to AWS console and search for S3
2. Click create bucket
3. For the bucket name enter something globally unique such as roommate-budget-helper-production-s3
4. Select US East 1 for the region
5. Select ACLs disabled
6. Make sure block all public access is selected
7. Enable encryption
8. Press create bucket
9. From the buckets page select the newly created bucket
10. Click on permissions
11. Under Cross-origin resource sharing (CORS) select edit
12. Paste the following json:

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": []
    }
]
```

13. Replace allowed origins with the intended domain of the production application
14. Click save changes

## SNS (Simple Notification Service)

1. Navigate to AWS console and search for SNS
2. Navigate to topics
3. Click create topic
4. Select FIFO
5. Enter the name invite-roommate
6. Enter the display name Roommate Invitations
7. Turn on message deduplication

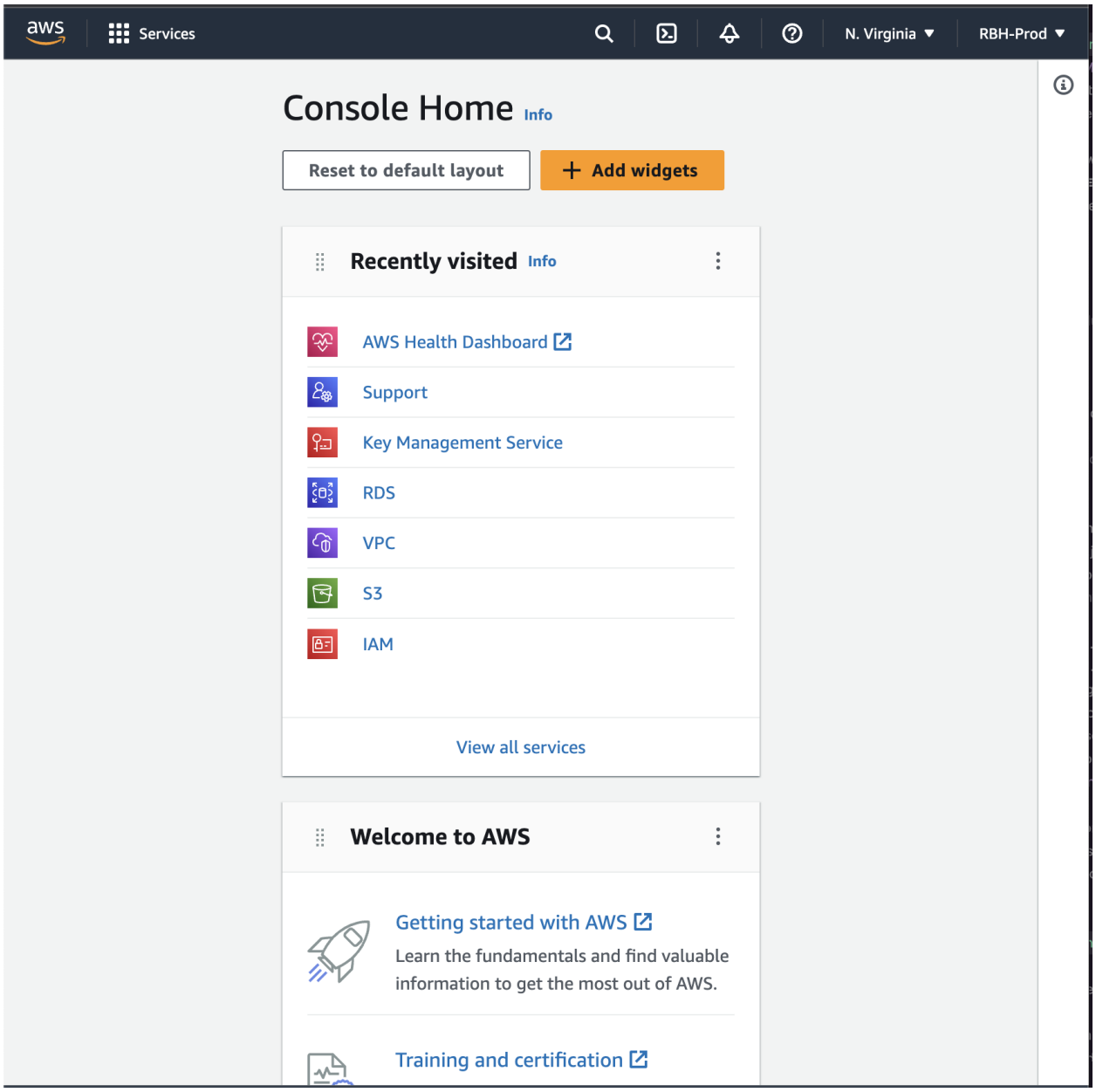
8. Enable Encryption with default CMK
9. Select create topic
10. Repeat steps 2-9 with the name reminders and display name Reminder subsystem

All required infrastructure should now be provisioned and deployed.

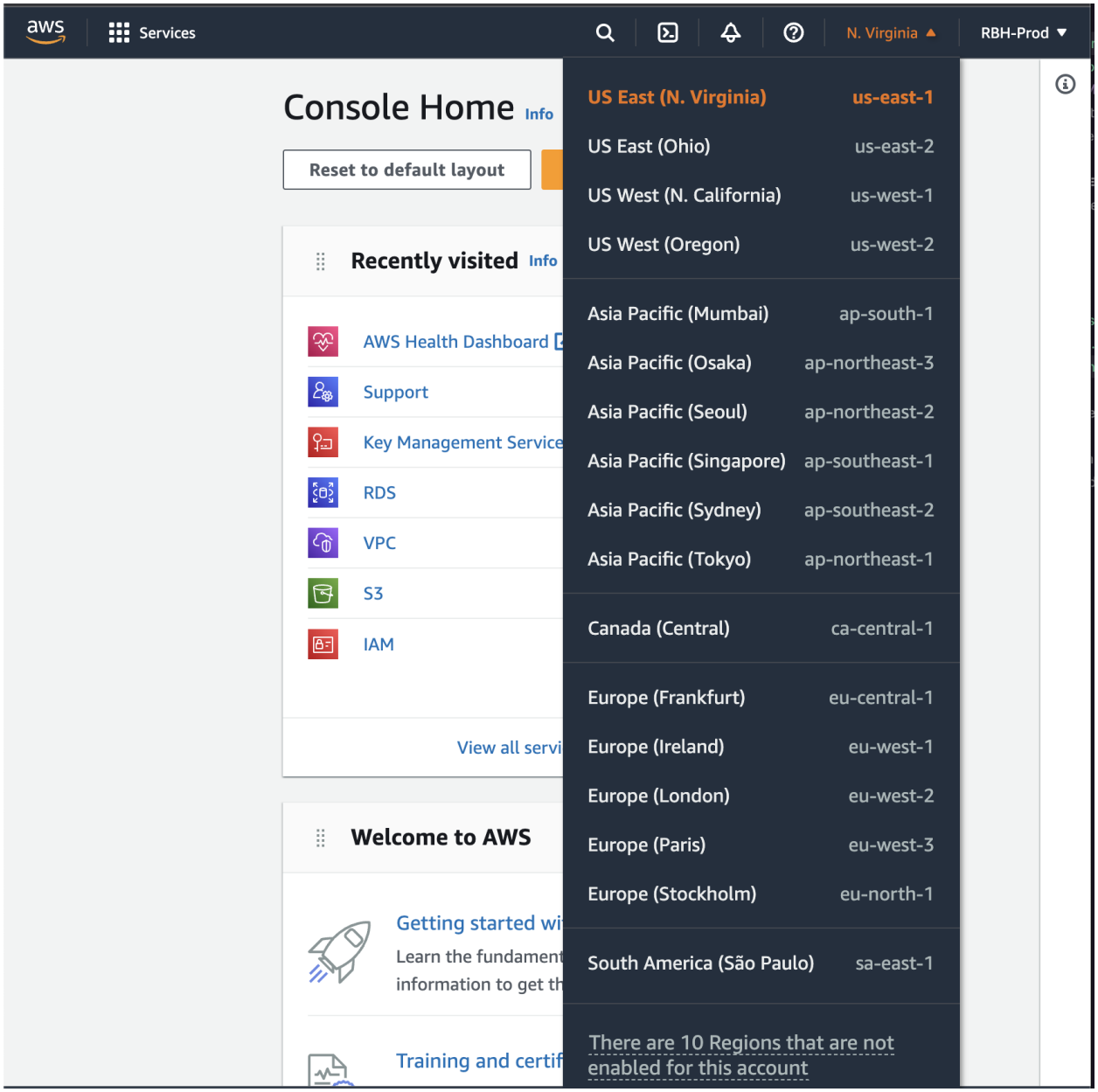
# Terraforming Guide

## Stage 1 - Setup AWS Terraform User

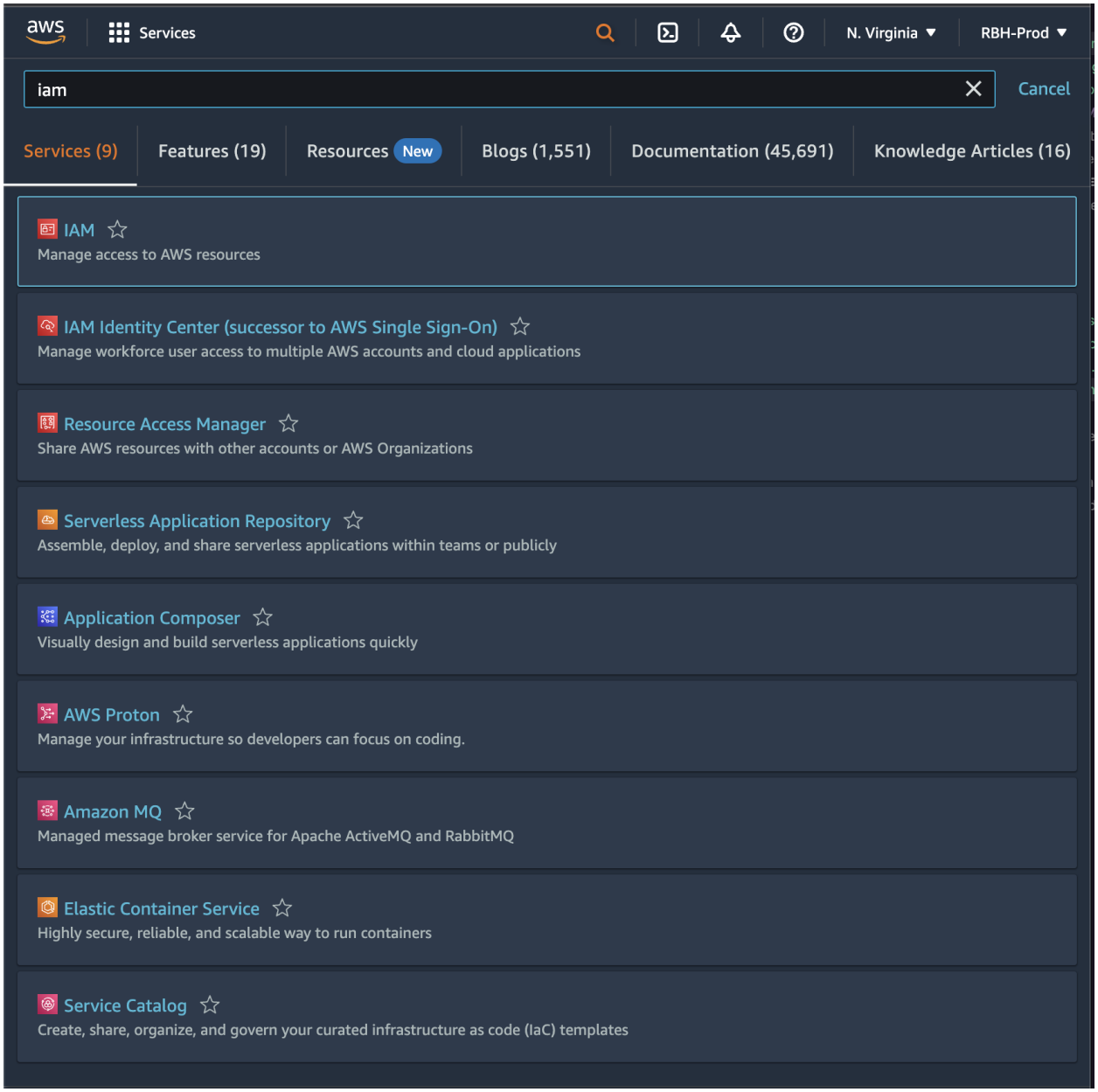
1. Login to AWS



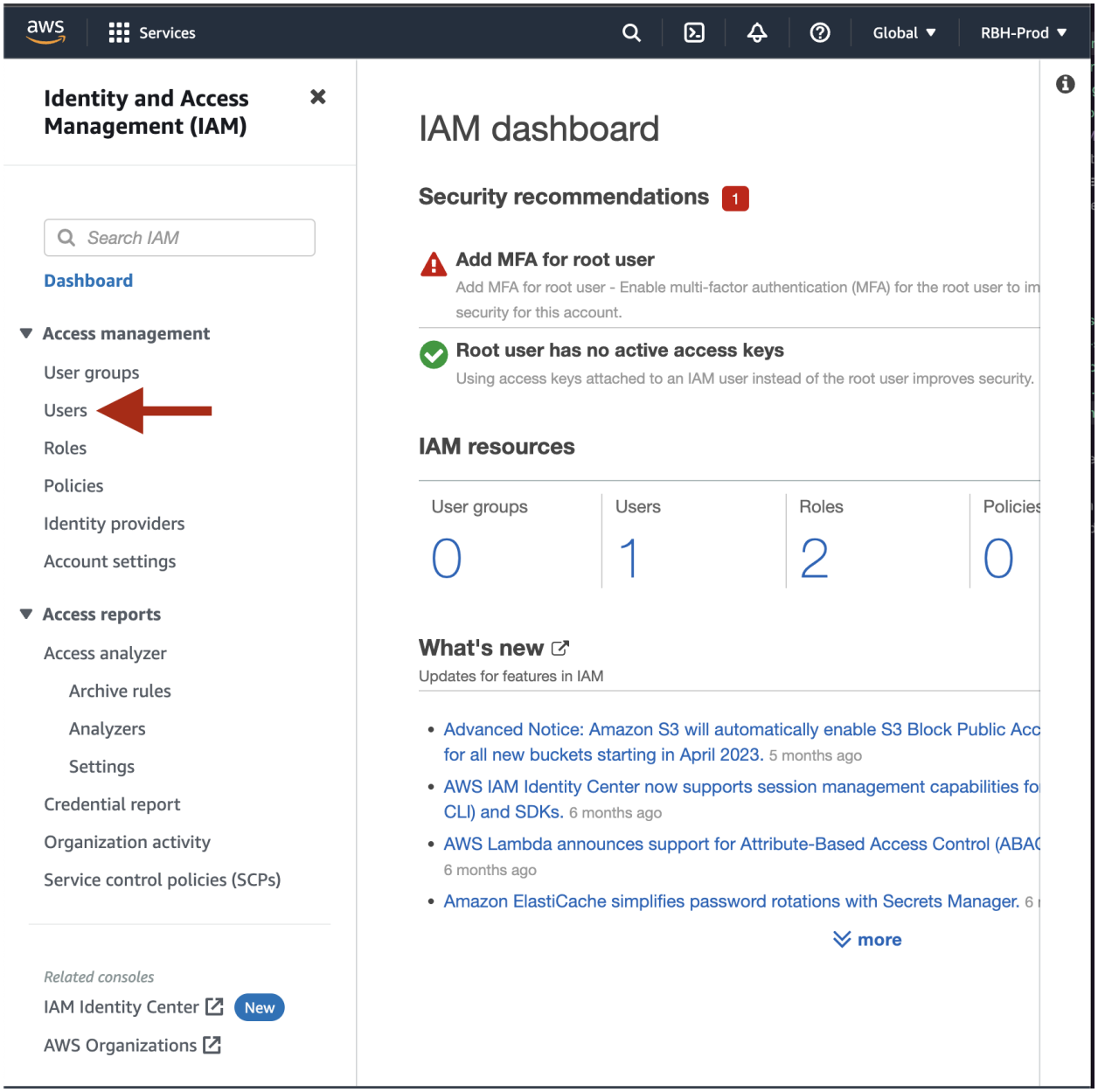
2. Ensure us-east-1 is selected for the region



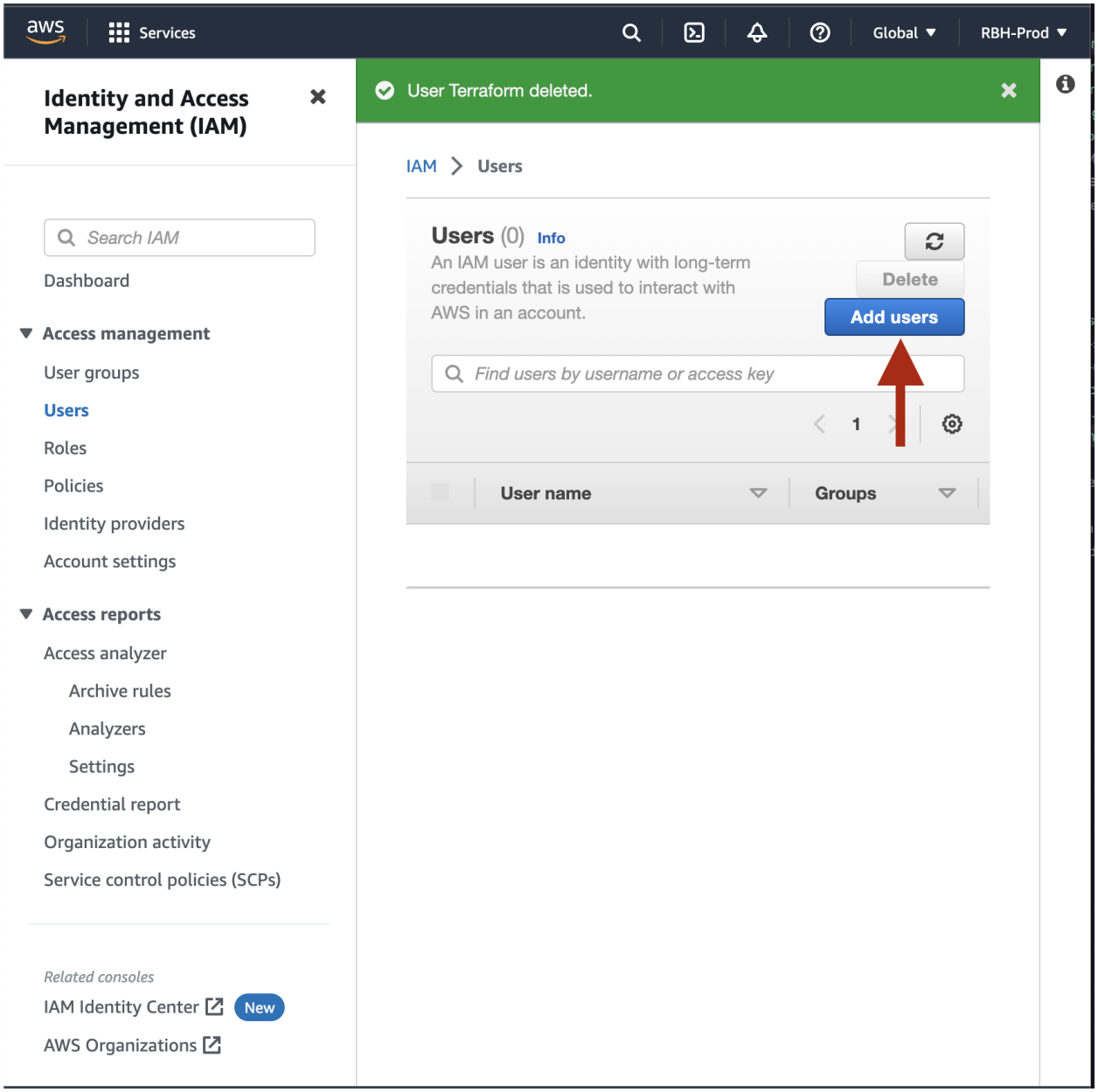
3. Navigate to IAM and select the IAM service



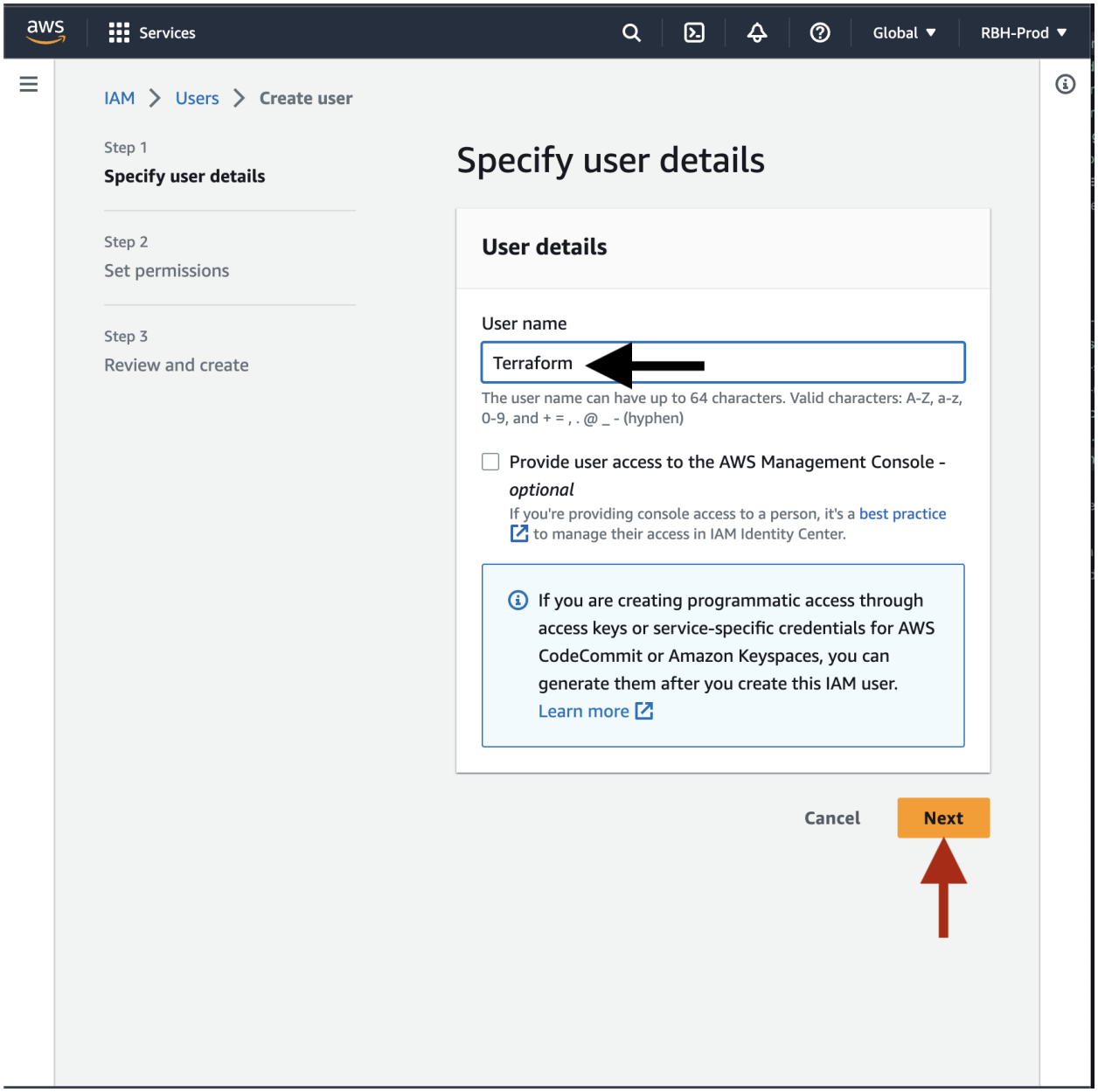
4. Select “Users” from the menu on the left of the IAM dashboard



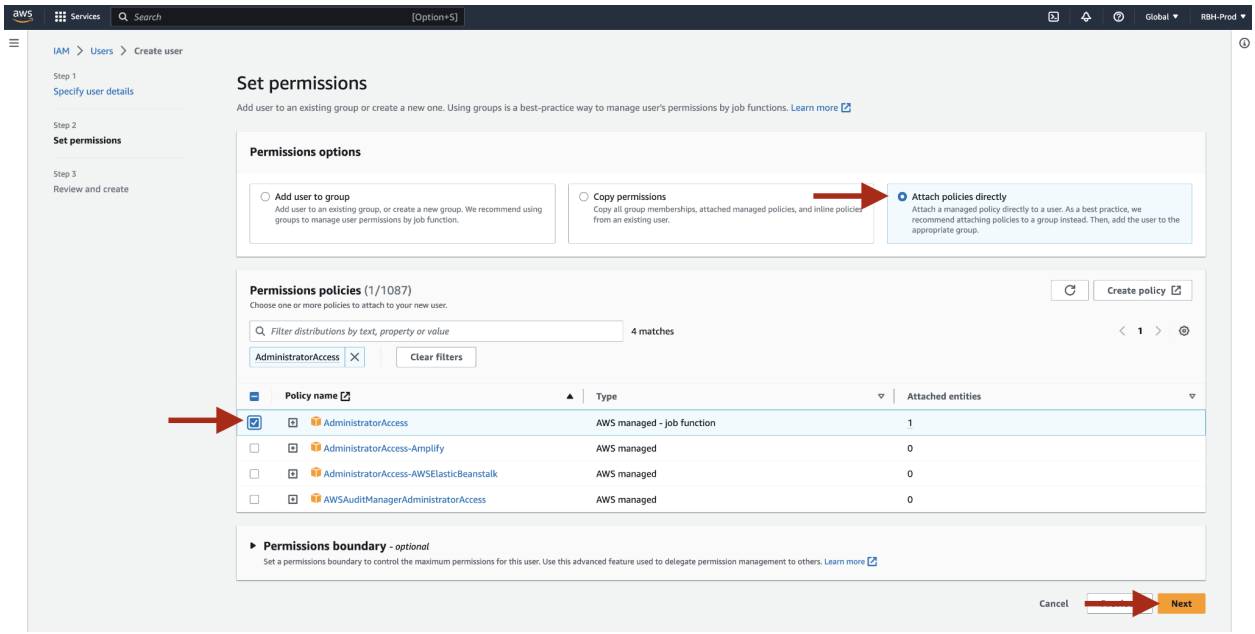
5. Press “Add user” button



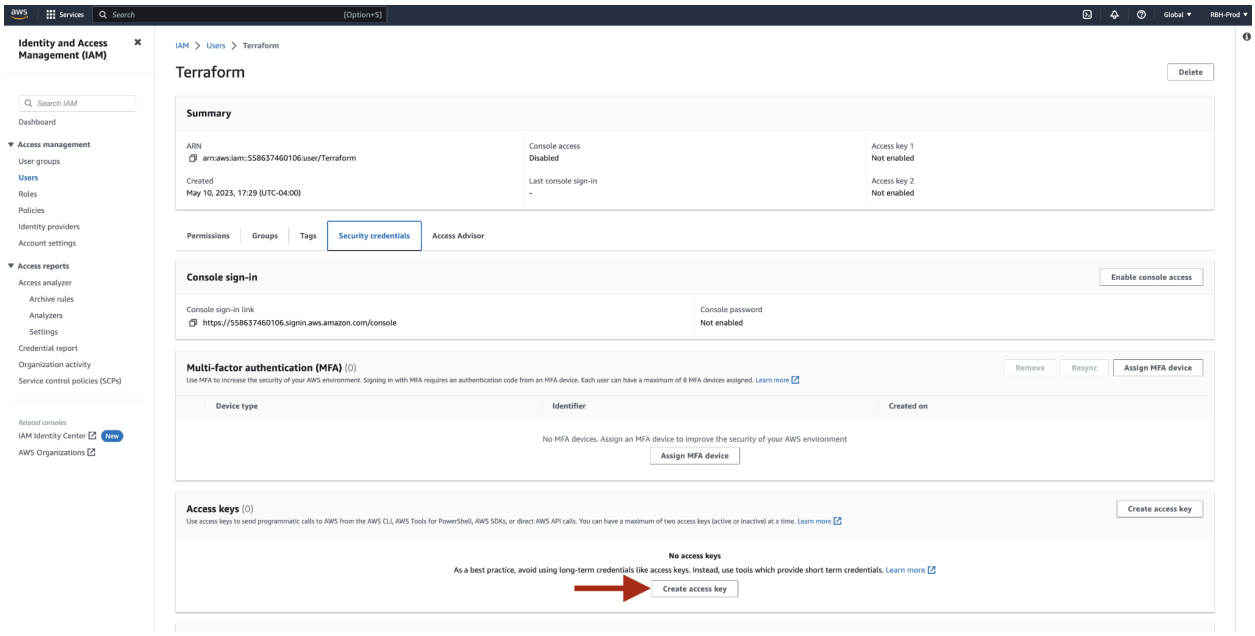
6. Make the username “Terraform” and click next



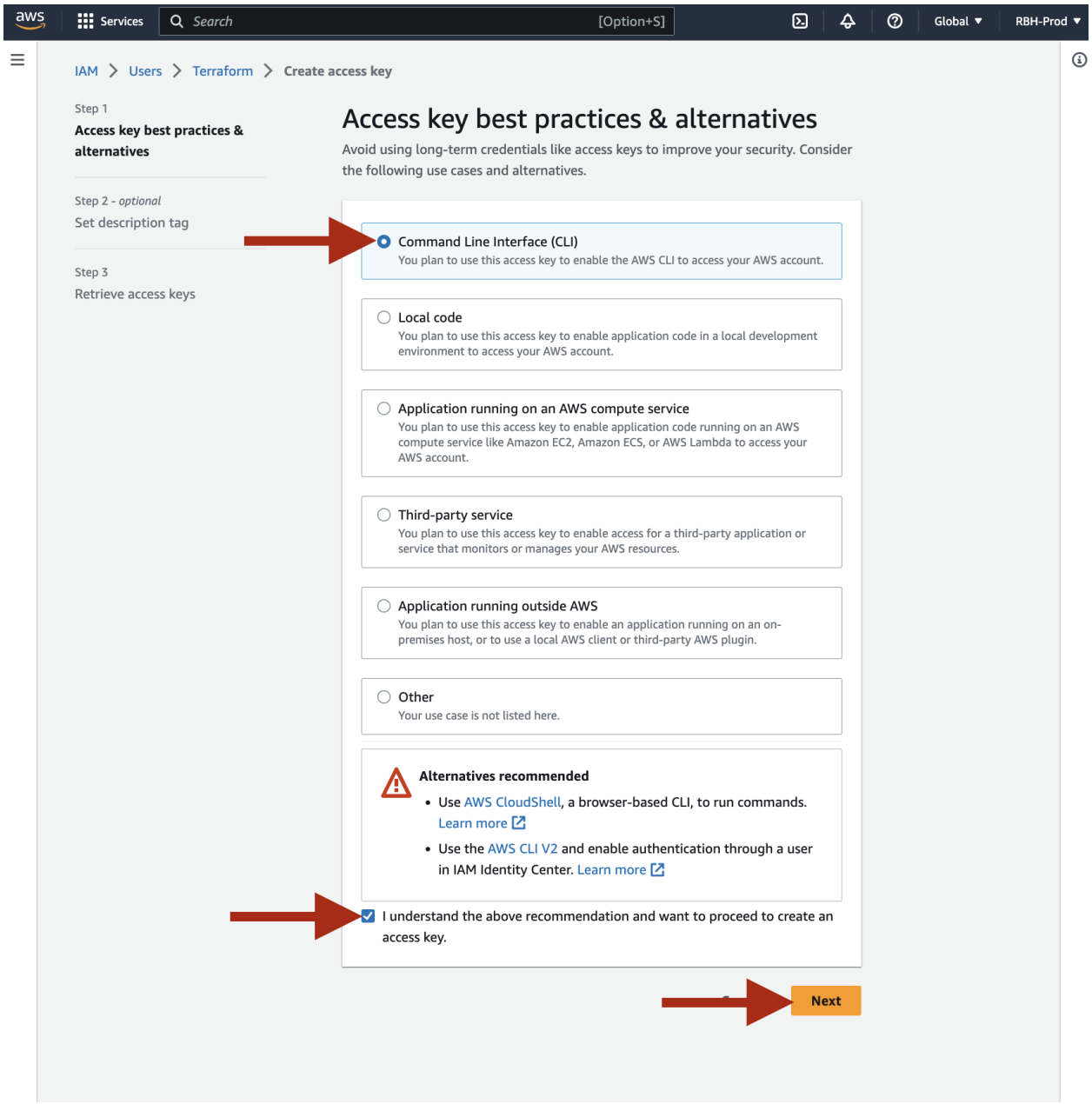
7. Select “Attach policies directly” and select the “AdminitratorAccess” AWS managed policy, then press next




8. Confirm the user creation on the next screen
9. Click on the newly created user and navigate to “Security Credentials”, scroll down and click on “Create access key”



10. Select “Command Line Interface (CLI)” and accept the above recommendation, then press “next”




11. Press “Create access key”
12. On the next screen, it will provide you the credentials. Store them safely and press “Done”

Your AWS account should now be configured correctly for terraform

## Stage 2 - Terraform Setup

1. On the cloned repository open a terminal
2. navigate to the “terraform” directory
3. Run “terraform init”

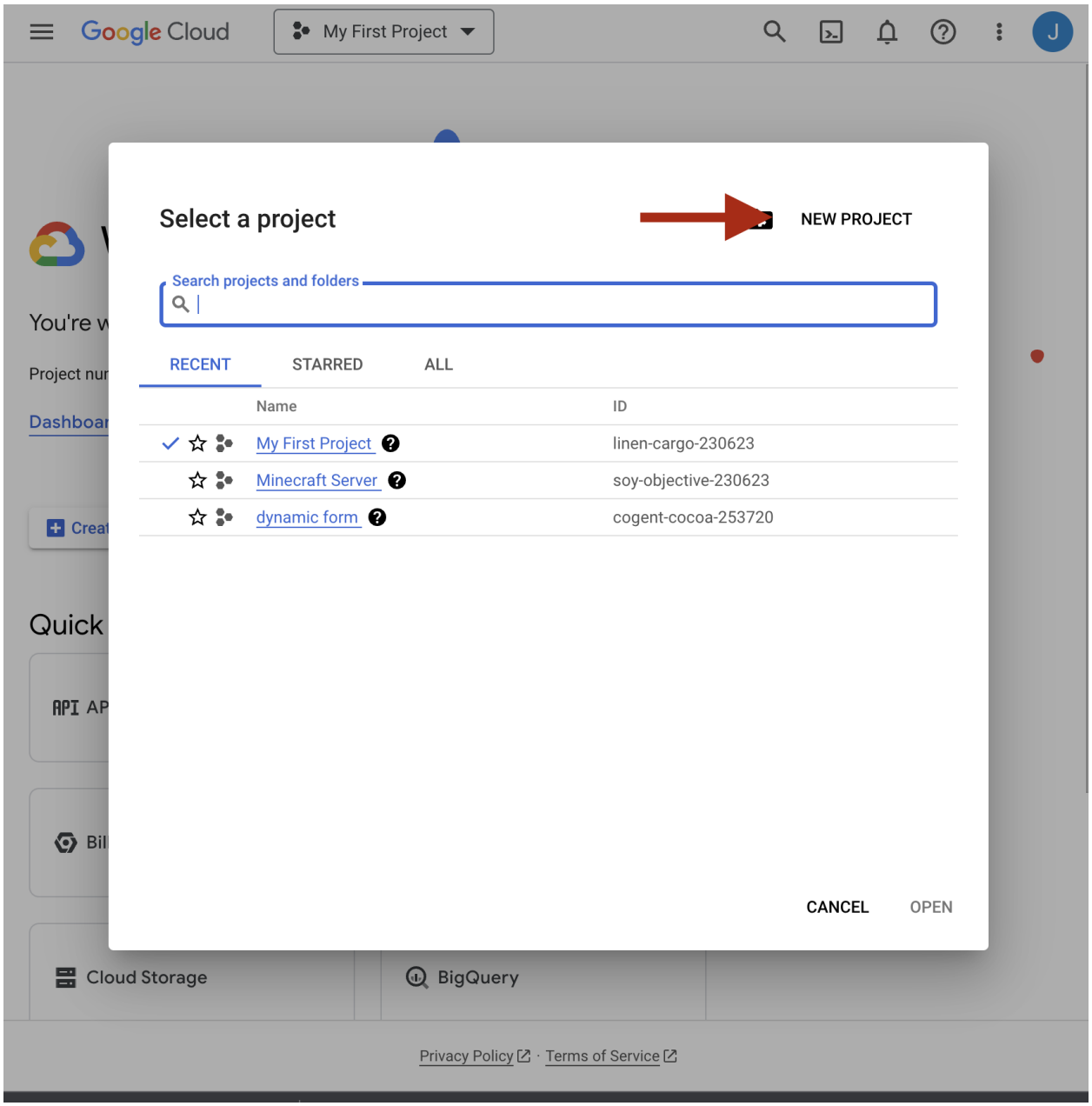
4. Run “aws configure”
5. Enter the access key of your Terraform user created in stage 1
6. Enter the secret key for the terraform user
7. Setup the default region as “us-east-1”
8. Leave the output format as default

## Google Auth Setup

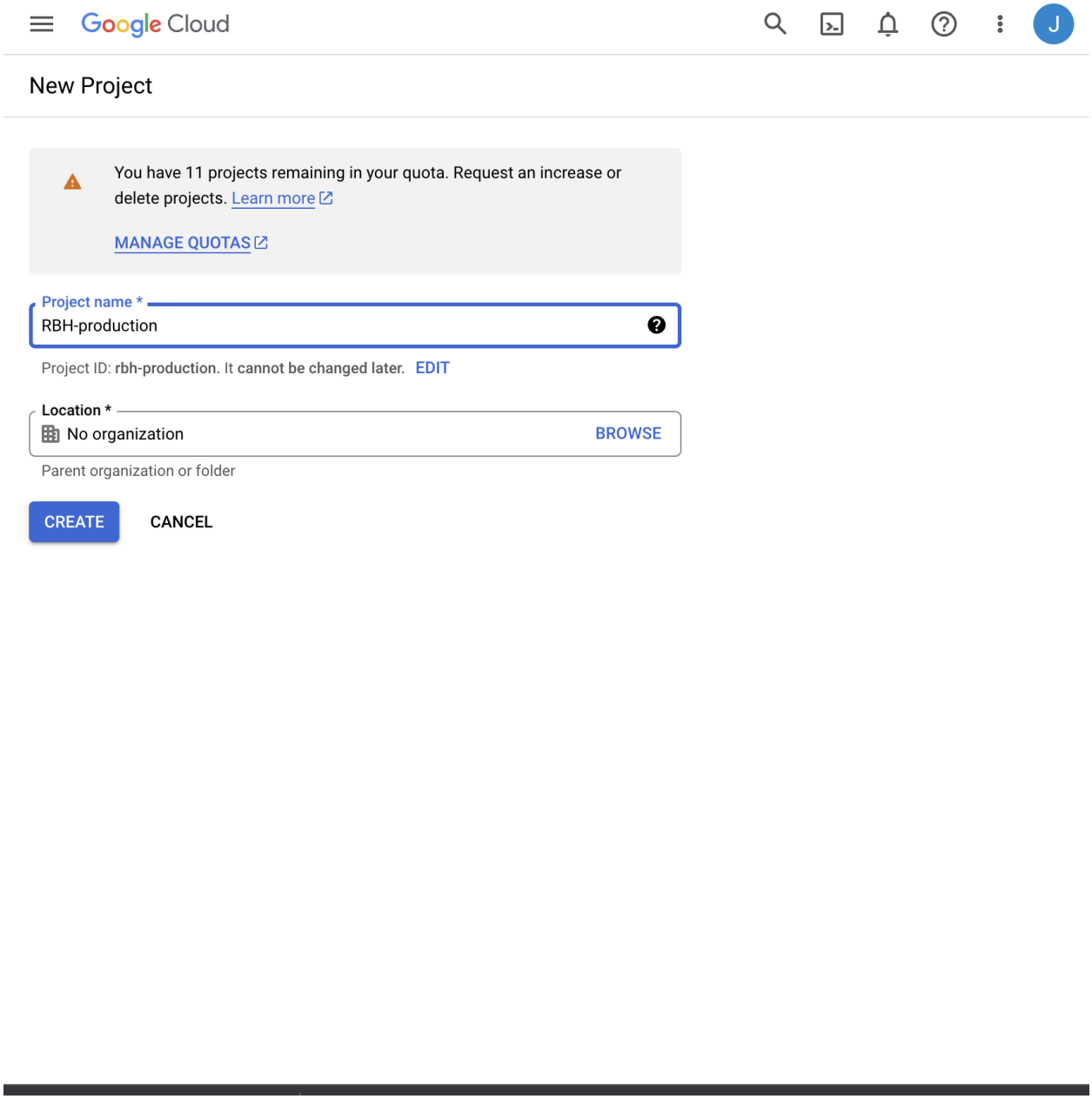
Before continuing with Terraform, you must create google oauth credentials

1. Navigate to “https://console.cloud.google.com/”
2. Login with the Google account you plan on using to manage the project

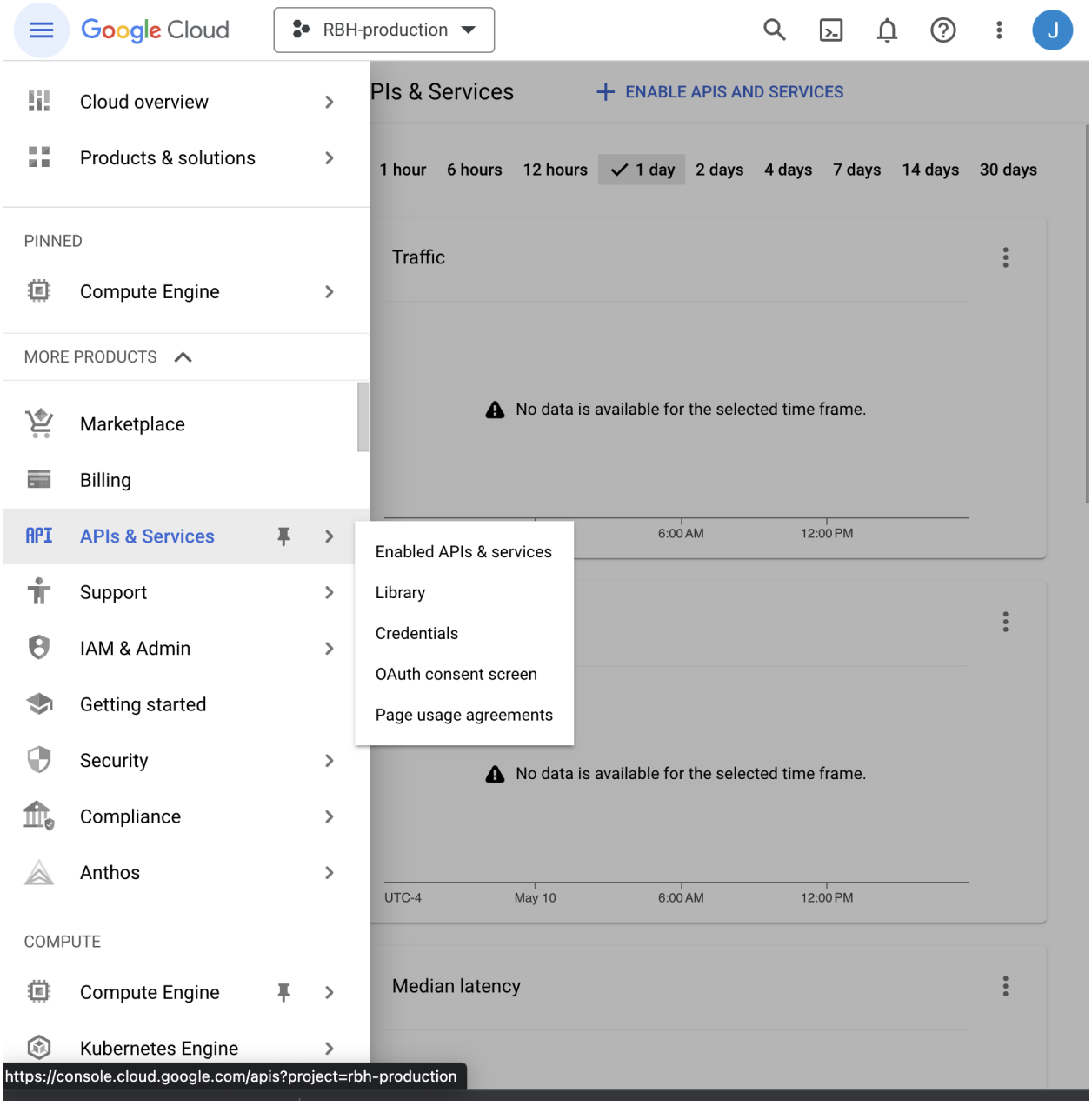
3. Create a new project



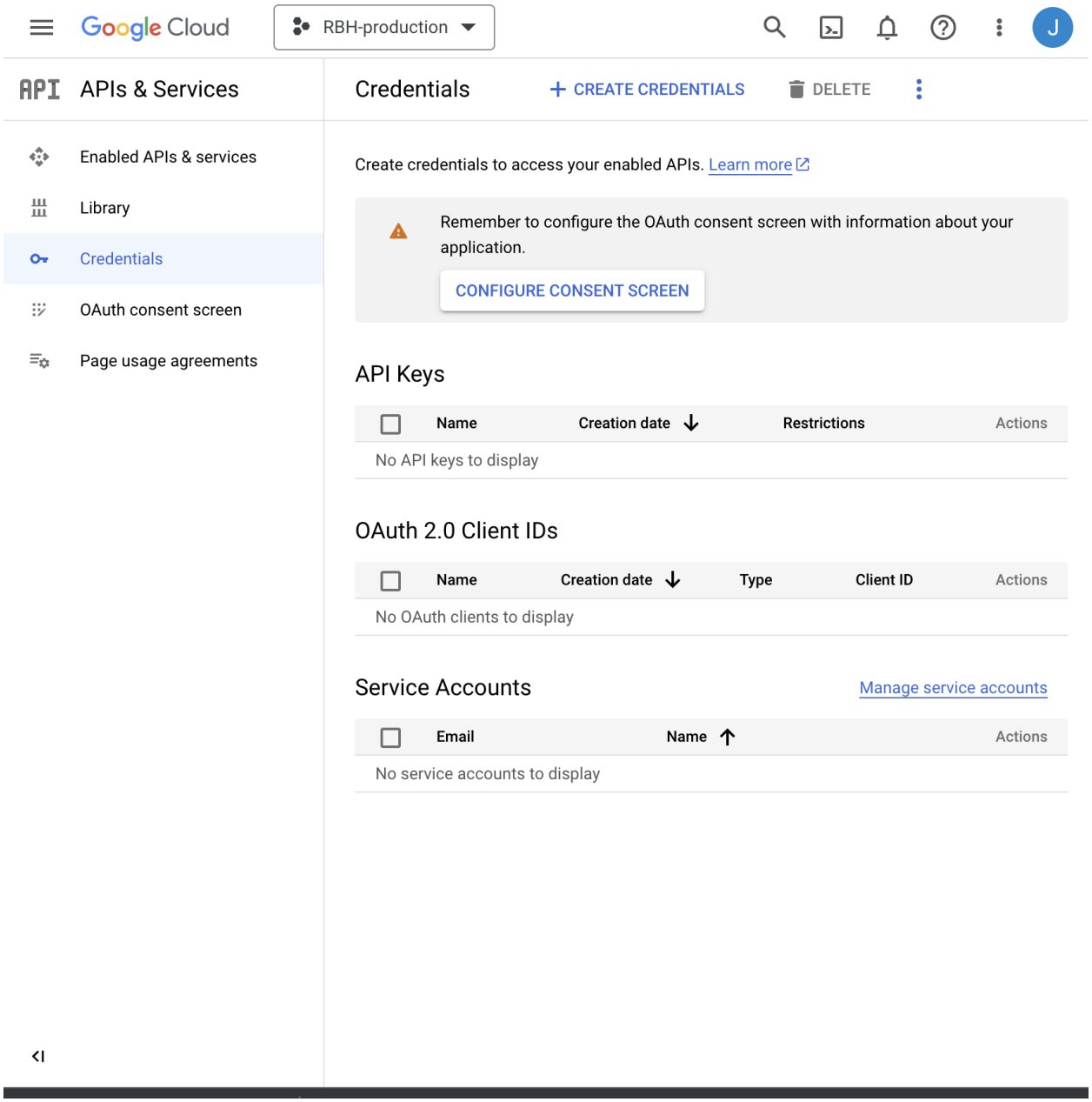
4. Enter a project name and press next



5. Once the project is created and is the actively selected project. Navigate to “APIs & Services”



6. From the left side menu select credentials




7. Click “Configure Consent Screen”
8. Select “External” and press create
9. Fill in the information with App name, user email, app logo, and domain
10. Click continue
11. Under scopes, select userinfo.email, userinfo.profile, and openid
12. Press continue and continue
13. Press return to dashboard
14. Return to credentials
15. Click create credentials
16. Select “OAuth Client ID”

17. Select “Web Application”
18. Name it “RBH Web”
19. Add the following javascript origins
    a. “https://<your-domain-here>/api/auth/callback/google”
    b. “http://localhost:3000/api/auth/callback/google” for local development
20. Press create, it will provide you the client id and client secret, store them somewhere safe and feel free to download the JSON file.

## Terraform Continued

9. In the terraform folder in your terminal, run “terraform plan”
10. Enter the google id and secret
11. Verify “terraform creates” a valid plan
12. run terraform apply to provision the infrastructure
13. Type” yes” to approve the creation
14. Log back into AWS after the infrastructure is created
15. Navigate to EC2
16. Create a new elastic IP
17. Attach it to the running ec2 instance

## Push App Image

1. Navigate to app folder of the repository in your terminal
2. Log into AWS and navigate to ECR
3. Click on "rbh-app" under private repositories
4. Click view push commands
5. Run each of the push commands

## Extra Steps

1. Configure your domain to point to a nginx reverse proxy (host yourself)
2. Setup SSL on Nginx reverse proxy
3. Reverse proxy should redirect http, https of your domain to the elastic ip of your instance in AWS targeting port 3000

## Integrate with Github Actions

1. Set the following Secrets in Github Actions on the repository
    a. AWS_ACCESS_KEY_ID = <access key of terraform user>
    b. AWS_SECRET_ACCESS_KEY = <secret key of terraform user>
    c. COGNITO_CLIENT_ID = <from terraform output>
    d. COGNITO_USER_POOL = <from terraform output>
    e. CYPRESS_API_KEY = <from mailslurp - Read Dev Guide 'Setting up testing'>
    f. DATABASE_URL = <from terraform output>
    g. GOOGLE_CLIENT_ID = <from google setup>
    h. GOOGLE_CLIENT_SECRET = <from google setup>yum install certbot python3-certbot-nginx
    i. NEXTAUTH_SECRET = <from terraform output>
    j. NEXTAUTH_URL = <base url of domain e.x https://roommatebudgethelper.tk>
    k. S3_BUCKET_NAME = <from terraform output>
    l. SLACK_TOKEN = <see make slack token>
    m. URL = <base url of domain e.x https://roommatebudgethelper.tk>

## Useful Commands

1. “terraform output <name of output>”
    a. Provides the value of terraform output field, will show sensitive information