# Smart Cow - React Flask NGINX Cloud Deployment Guide

## Task #2

https://github.com/ryancomia/sc-exercise/tree/main

Before we begin deploying into AWS, let's create a Workflow to deploy the code in an automated manner.

For this we will use Github Actions. I wrote the following yaml file named deploy.yml

Found here:
https://github.com/ryancomia/sc-exercise/blob/dev/.github/workflows/deploy.yml

Steps:

- Checkout the code when a trigger to main branch
- Creates a package of the code and zips it
- Generate a timestamp
- Adds a string into the name with timestamp
- Deploys the package into AWS S3 and triggers an action to deploy it into the Elastic Beanstalk environment

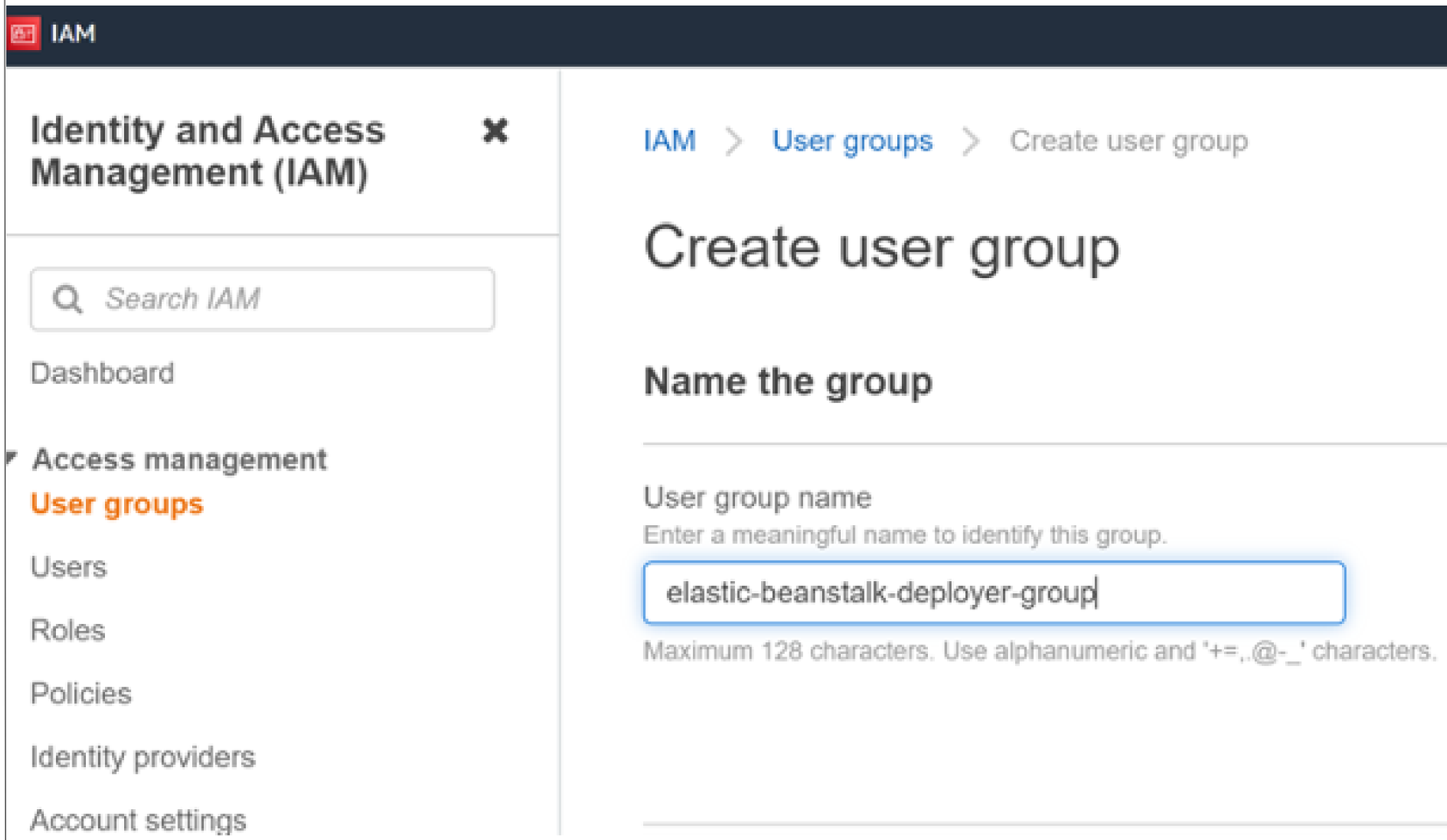
Before we can deploy we need the following:

- AWS Keys
- Application Name
- Application ENV

```yaml
name: Deploy to AWS Beanstalk
on:
  push:
    branches: [ main ]
jobs:
  Deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout Source Code
        uses: actions/checkout@master

      - name: Generate Deployment Package
        run: zip -r deployment.zip * -x "*node_modules**.git*"

      - name: Create a Timestamp
        uses: gerred/actions/current-time@master
        id: current-time

      - name: String Replacement
        uses: frabert/replace-string-action@v2.0
        id: format-time
        with:
          pattern: '[:\.]+'
          string: '${{ steps.current-time.outputs.time}}'
          replace-with: '='
          flags: 'g'

      - name: Deploy to Elastic Beanstalk
        uses: einaregilsson/beanstalk-deploy@v20
        with:
          aws_access_key: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws_secret_key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          application_name: smartcow-testapp
          environment_name: Smartcowtestapp-env
          version_label: "sc-exercise-${{ steps.format-time.outputs.replaced }}"
          region: ap-southeast-1
          deployment_package: deployment.zip
```

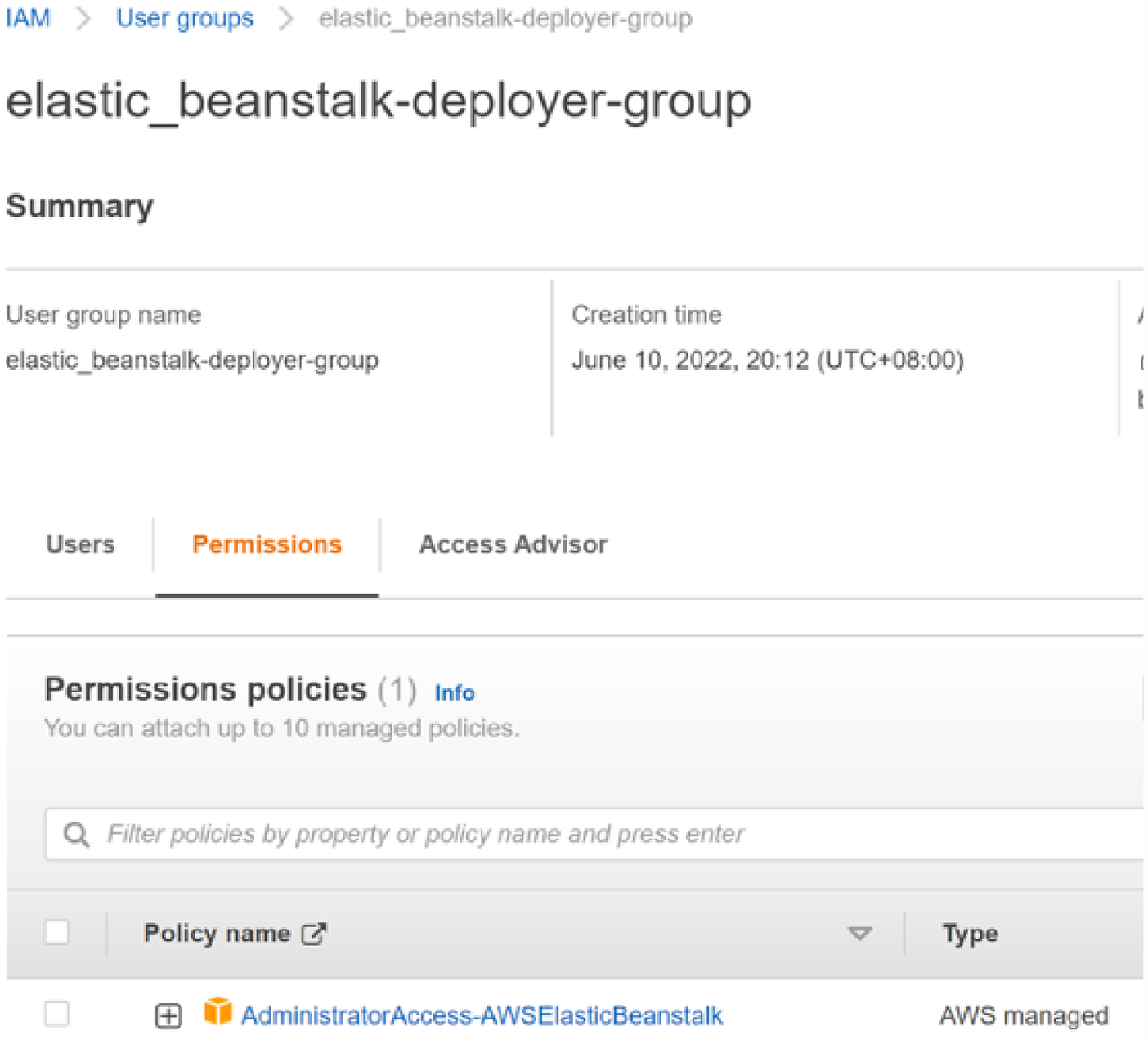
Ideally we want to create a Group and attach a policy into it.

Note: granting permission to a specific user/service account is a bad practice! =)

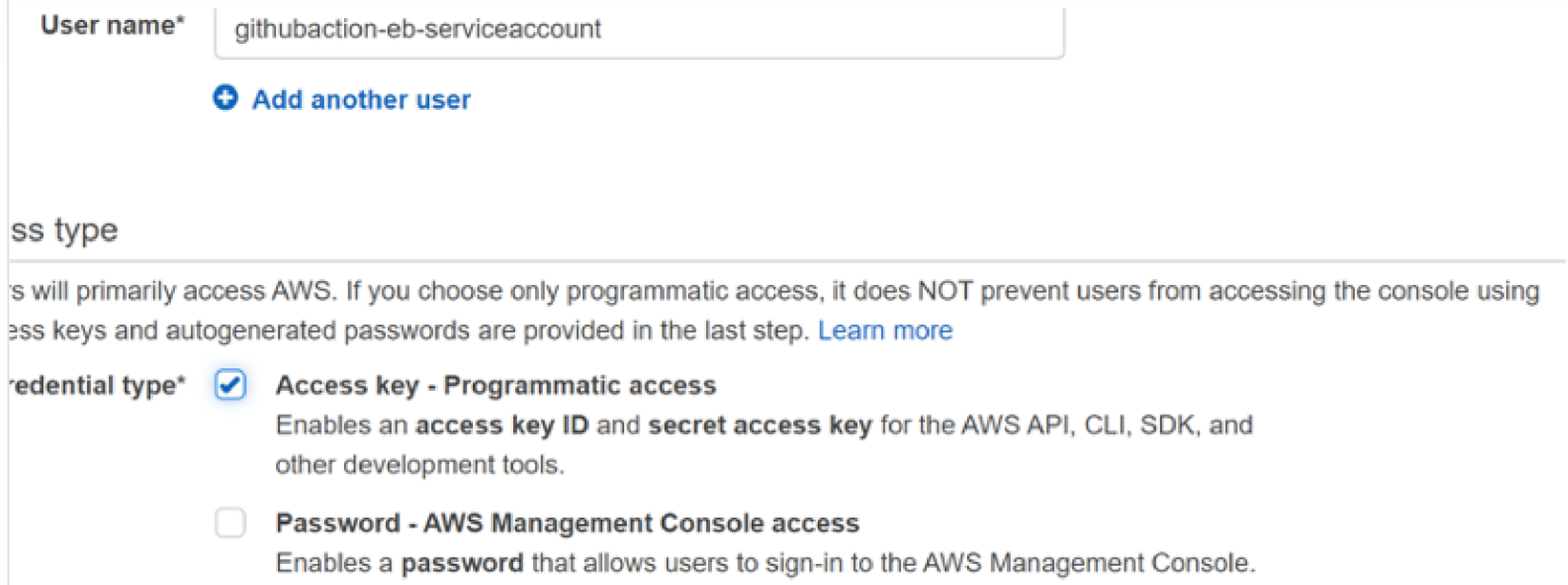


For this exercise:

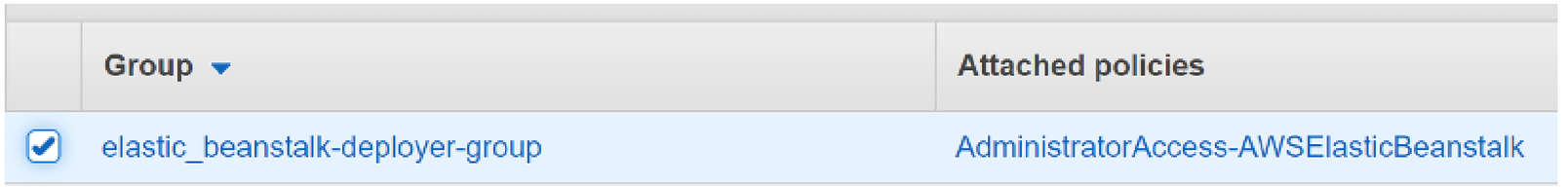
We will use AWS Elastic Beanstalk admin access for the service account



Modify the account credential type to access-key programmatic type

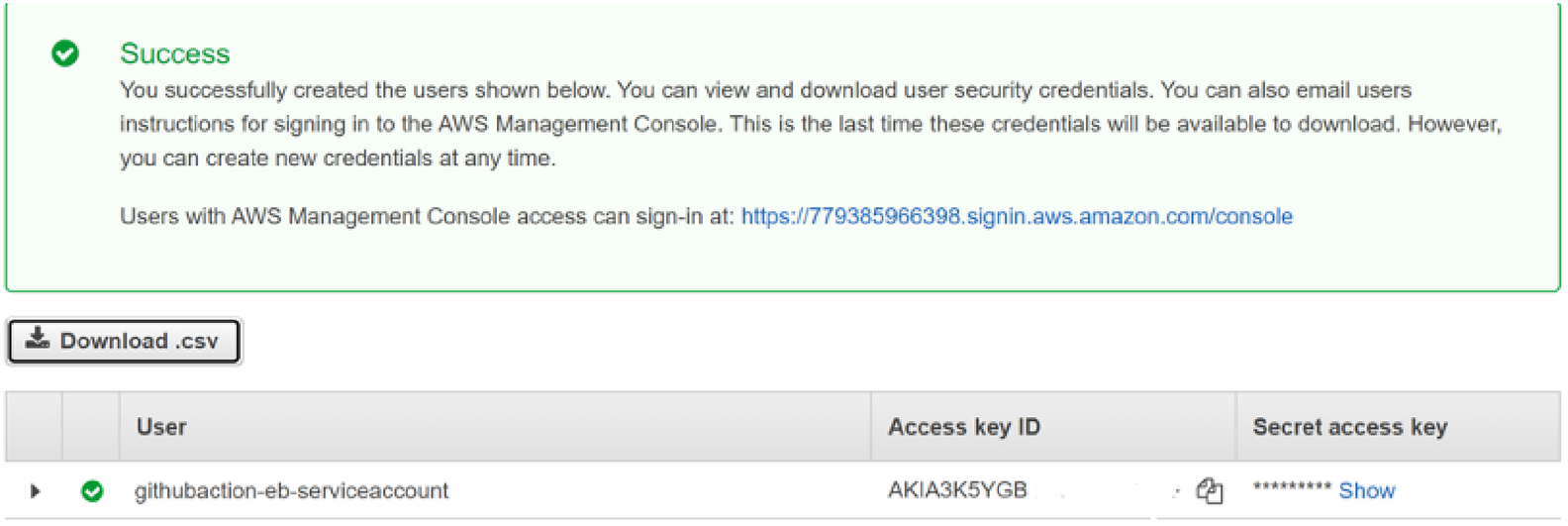


Add the created Group for elastic beanstalk deployer



| | Group | Attached policies |
|---|---|---|
| ☑ | elastic_beanstalk-deployer-group | AdministratorAccess-AWSElasticBeanstalk |

Finally we can get the API keys and.
The Keys will then be stored into github, and we can then call this to deploy using github action



**Success**

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

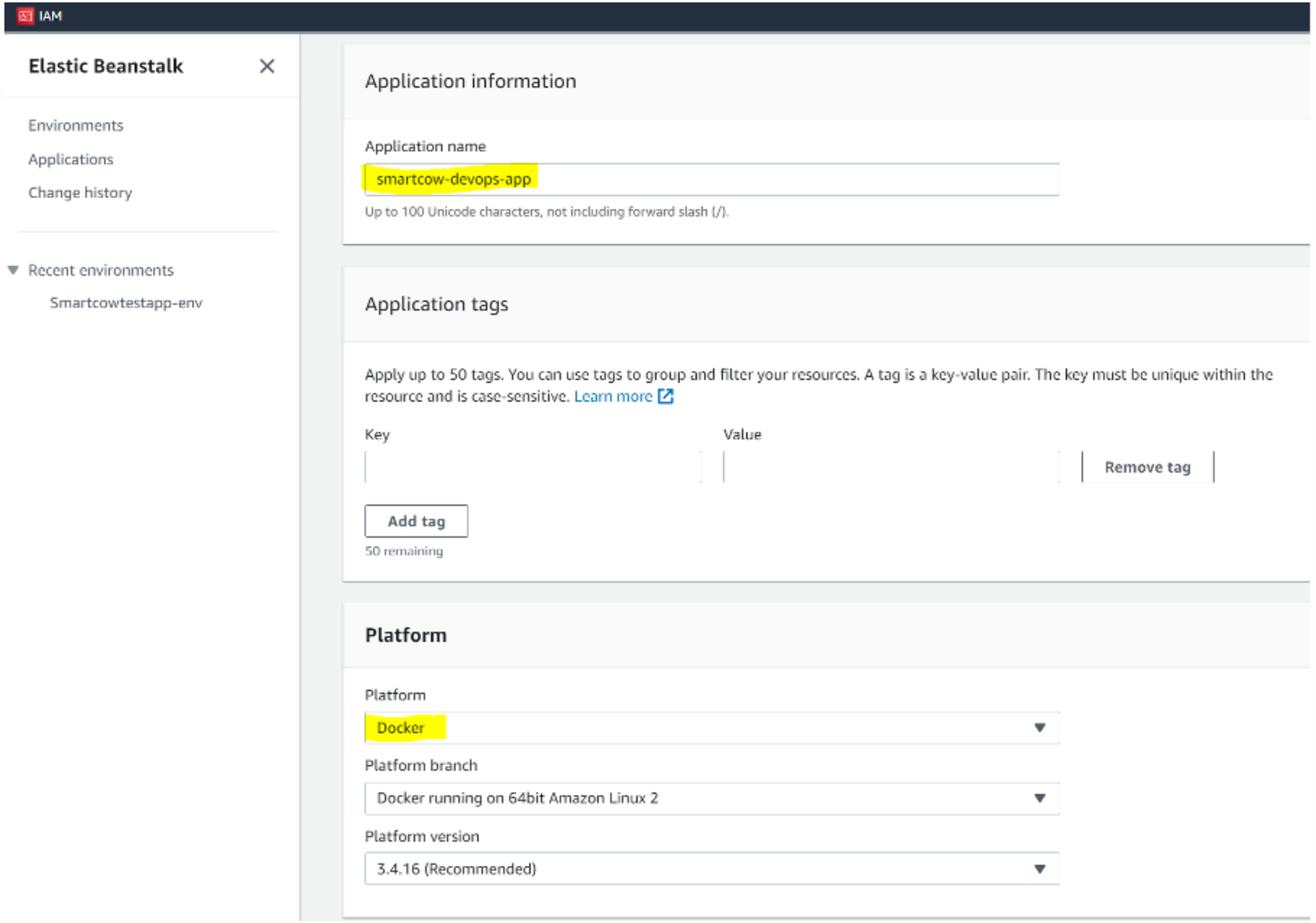
Users with AWS Management Console access can sign-in at: https://779385966398.signin.aws.amazon.com/console

Download .csv

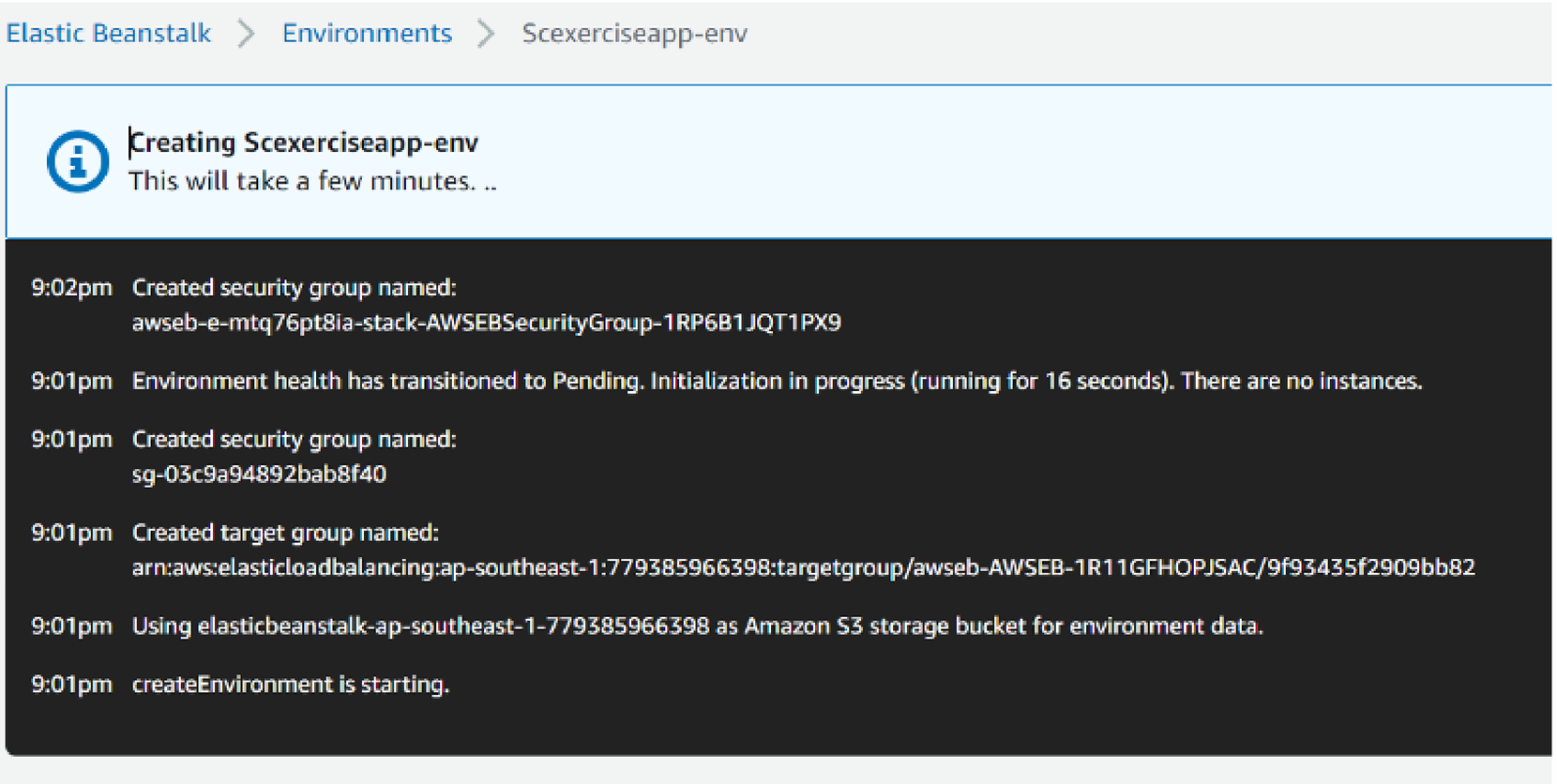
| | User | Access key ID | Secret access key |
|---|---|---|---|
| ▸ | githubaction-eb-serviceaccount | AKIA3K5YGB | ********** Show |

In AWS bean-stalk, we create a web application that would be used as our deployment environment

I chose Docker as the target platform as we intend it to be a containerized deployment

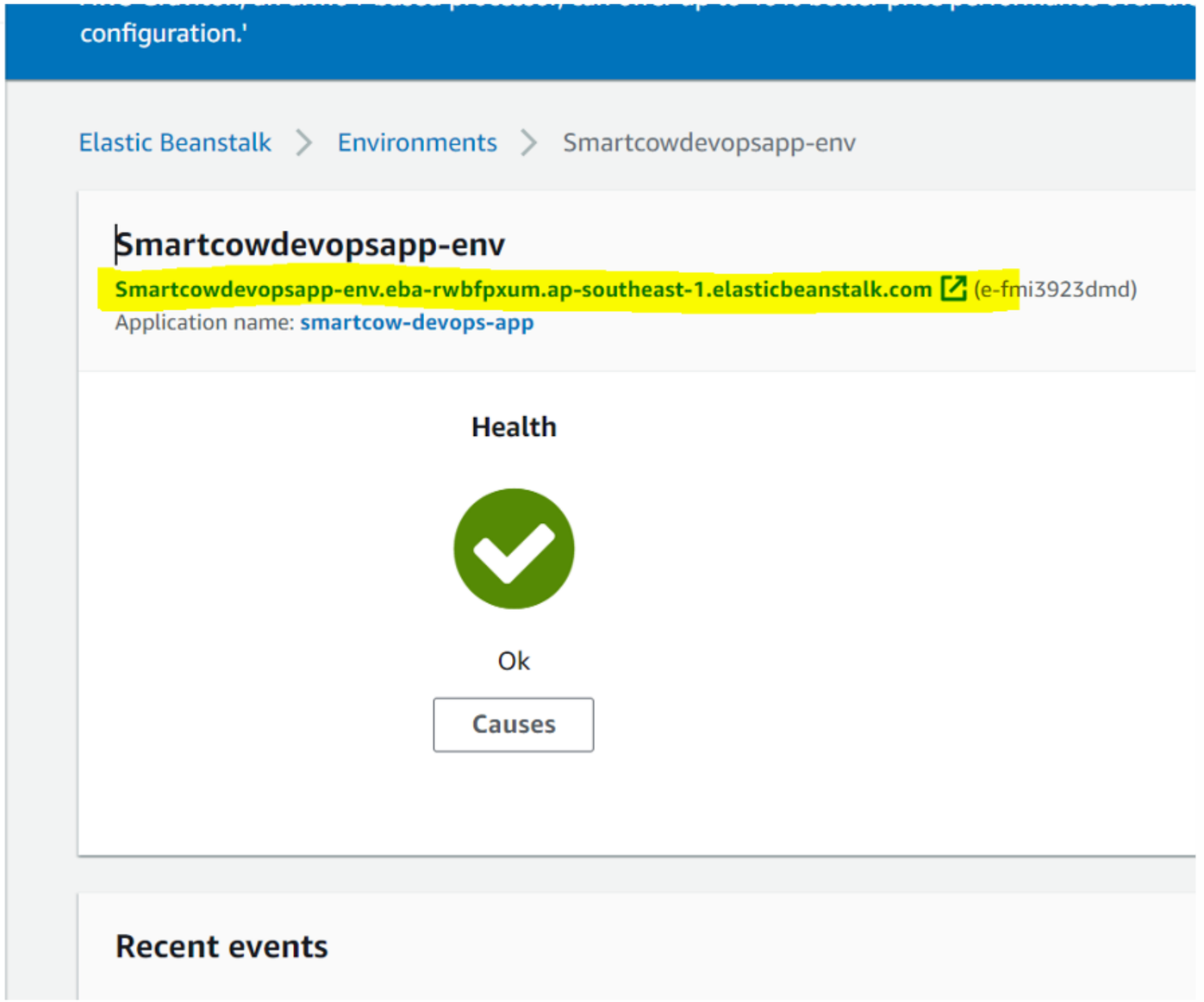


Spinning up an EB environment takes about 5-10mins.



Now that the environment is up & running, we can get the following:

- Public URI
- Application Name
- Application Env



Given that we have the EB load balancer endpoint, I updated the react fetch to use the URI to get api stats
Ideally in a actual PROD setup, we need to have a public DNS record CNAME'd to this load balancer.

```
async loadData() {
  try {
    const res = await fetch('http://smartcowtestapp-env.eba-mqh2ba2q.ap-southeast-1.elasticbeanstalk.com/stats');
    const blocks = await res.json();
    const ram = blocks.ram;
    const cpu = blocks.cpu;
```

However I did try creating a Route 53 record but unfortunately the record propagation is not fast enough. (And give I am left 1 day to complete and document this exercise!, So I resorted to use the LB URI. =)

Note: using a CNAME also allows for a faster blue/green deployment approach. As it is faster to switch a record compared to re-triggering a action redeployment

We can then pass the Application Name and Environment name as a variable in deploy.yml for the action to identify where we want to deploy it.

```
with:
  aws_access_key: ${{ secrets.AWS_ACCESS_KEY_ID }}
  aws_secret_key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
  application_name: smartcow-testapp
  environment_name: Smartcowtestapp-env
  version_label: "sc-exercise-${{ steps.format-time.outputs.replaced }}"
  region: ap-southeast-1
  deployment_package: deployment.zip
```

## CLOUD URI

http://smartcowdevopsapp-env.eba-rwbfpxum.ap-southeast-1.elasticbeanstalk.com/