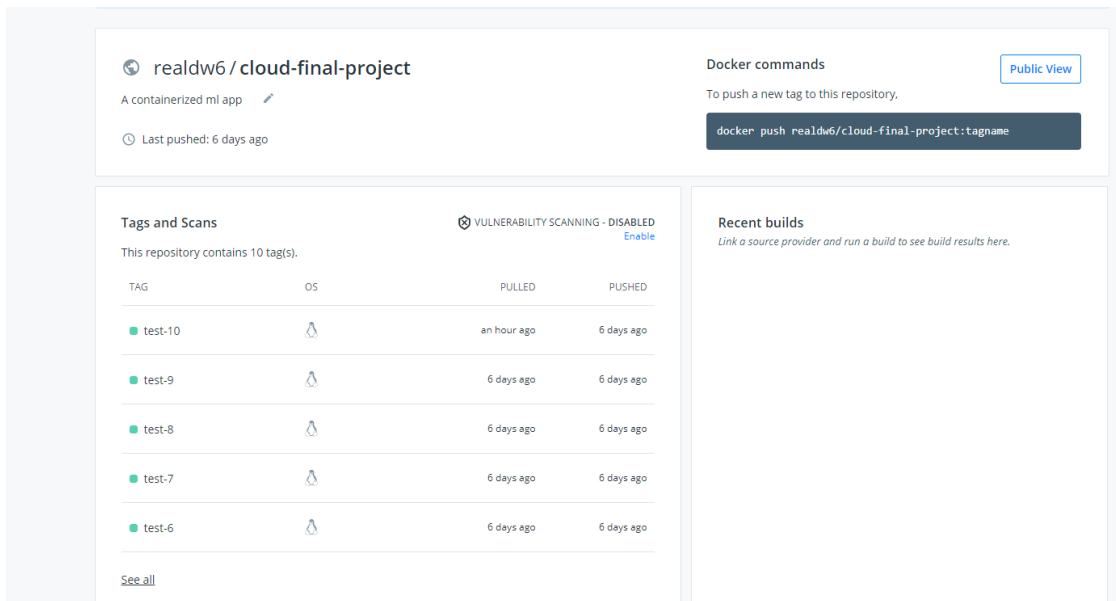


Project Workflow

We got the app working locally, containerized it, and pushed it to DockerHub.



The screenshot shows the DockerHub repository page for `realdw6/cloud-final-project`. The repository is described as "A containerized ml app" and was last pushed 6 days ago. It features a "Tags and Scans" section with a table of tags, a "Recent builds" section, and a "Docker commands" section with a button to push a new tag.

Tags and Scans

This repository contains 10 tag(s).

| TAG | OS | PULLED | PUSHED |
|---------|-------------|-------------|------------|
| test-10 | linux/amd64 | an hour ago | 6 days ago |
| test-9 | linux/amd64 | 6 days ago | 6 days ago |
| test-8 | linux/amd64 | 6 days ago | 6 days ago |
| test-7 | linux/amd64 | 6 days ago | 6 days ago |
| test-6 | linux/amd64 | 6 days ago | 6 days ago |

[See all](#)

Docker commands

To push a new tag to this repository,

```
docker push realdw6/cloud-final-project:tagname
```

[Public View](#)

After the container was hosted, we pulled the image onto an EC2 instance.

```
last login: Tue Apr 27 21:55:22 2021 from ec2-18-206-107-26.compute-1.amazonaws.com

 _ _ | _ _ | _ _ |
 _ _ | ( _ _ | /   Amazon Linux 2 AMI
 _ _ | \ _ _ | _ _ |

https://aws.amazon.com/amazon-linux-2/
ec2-user@ip-172-31-59-28 ~]$ sudo docker pull realdw6/cloud-final-project:test-10
```

Then we started the container.

```
last login: Tue Apr 27 21:55:22 2021 from ec2-18-206-107-26.compute-1.amazonaws.com

 _ _ | _ _ | _ _ |
 _ _ | ( _ _ | /   Amazon Linux 2 AMI
 _ _ | \ _ _ | _ _ |

https://aws.amazon.com/amazon-linux-2/
ec2-user@ip-172-31-59-28 ~]$ sudo docker ps
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?
ec2-user@ip-172-31-59-28 ~]$ sudo service docker start
Redirecting to /bin/systemctl start docker.service
ec2-user@ip-172-31-59-28 ~]$ sudo docker run -d -p 80:8080 realdw6/cloud-final-project:test-10
52c0e0405003f415e567d7eeffe6b8852f6b12139e9d89efae3c89a5ff690e0
ec2-user@ip-172-31-59-28 ~]$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
52c0e040500   realdw6/cloud-final-project:test-10 "python3 main.py"       12 seconds ago Up 11 seconds  80/tcp, 0.0.0.0:80->8080/tcp        heuristic_pani
ec2-user@ip-172-31-59-28 ~]$
```

After that, we created an AMI along with a load balancer.

The screenshot shows the AWS Management Console interface for a load balancer. At the top, there's a 'Create Load Balancer' button and an 'Actions' dropdown. Below this is a search bar and a table listing load balancers. The table has columns: Name, DNS name, State, VPC ID, Availability Zones, and Type. One load balancer is listed: 'derek-krinke-lb' with DNS name 'derek-krinke-lb-616126281.u...', state 'active', VPC ID 'vpc-0b943876', and availability zones 'us-east-1a, us-east-1b, ...'. Below the table, there's a section for 'Load balancer: derek-krinke-lb' with tabs for 'Description', 'Listeners', 'Monitoring', 'Integrated services', and 'Tags'. The 'Description' tab is selected, showing 'Basic Configuration' with fields for Name ('derek-krinke-lb') and ARN ('arn:aws:elasticloadbalancing:us-east-1:961056229775:loadbalancer/app/derek-krinke-lb/a1e8743c557c9dce').

| Name | DNS name | State | VPC ID | Availability Zones | Type |
|-----------------|--------------------------------|--------|--------------|----------------------------|-------------|
| derek-krinke-lb | derek-krinke-lb-616126281.u... | active | vpc-0b943876 | us-east-1a, us-east-1b,... | application |

Load balancer: derek-krinke-lb

Description Listeners Monitoring Integrated services Tags

Basic Configuration

Name derek-krinke-lb

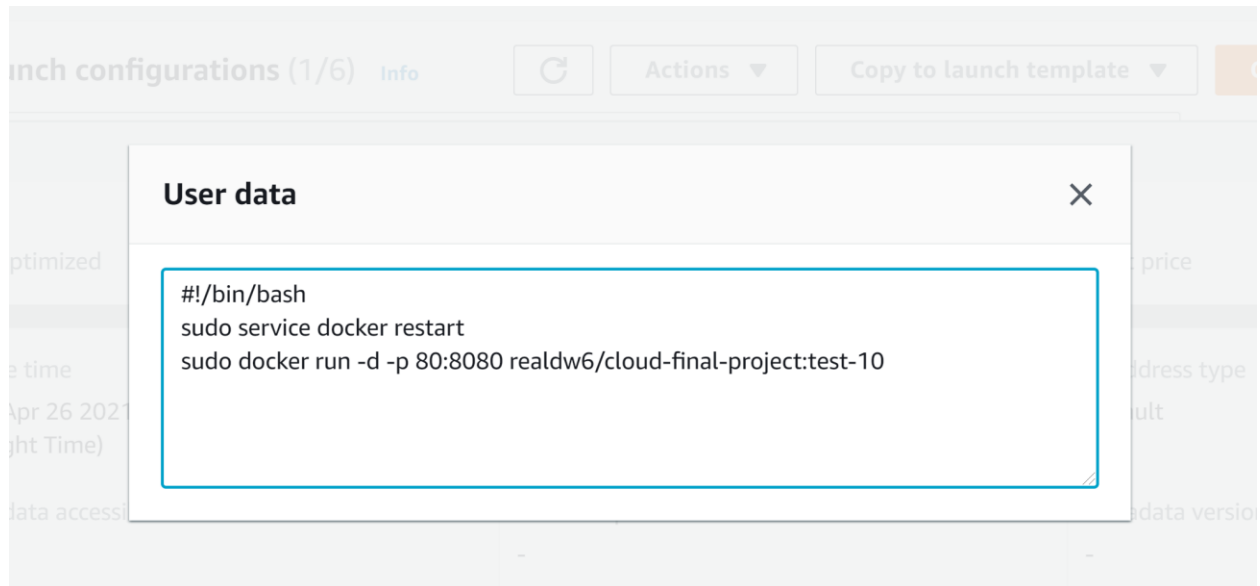
ARN arn:aws:elasticloadbalancing:us-east-1:961056229775:loadbalancer/app/derek-krinke-lb/a1e8743c557c9dce

Then we attached the load balancer to an auto-scaling group.

The screenshot shows the AWS Management Console interface for 'Auto Scaling groups'. At the top, there's a search bar and a 'Create an Auto Scaling group' button. Below this is a table listing auto scaling groups. The table has columns: Name, Launch template/configuration, Instances, Status, and Desired capacity. Two auto scaling groups are listed: 'derek-krinke-as-3' with launch template 'derek-krinke-3', 0 instances, and status '-'; and 'awseb-e-xjznthitdh-stac' with launch template 'awseb-e-xjznthitdh-stack-AWSEBAu...', 0 instances, and status 'Updating capacity'. The 'Desired capacity' column shows 0 for the first group and 1 for the second group.

| Name | Launch template/configuration | Instances | Status | Desired capacity |
|-------------------------|-------------------------------------|-----------|-------------------|------------------|
| derek-krinke-as-3 | derek-krinke-3 | 0 | - | 0 |
| awseb-e-xjznthitdh-stac | awseb-e-xjznthitdh-stack-AWSEBAu... | 0 | Updating capacity | 1 |

We then altered the config file to execute BASH commands on startup. These commands get the containers running.



Finally, we stress tested the application with Locust.

A screenshot of the 'Start new load test' form in the Locust web interface. The form has a dark green background with white text. It contains three input fields and a button. The first input field is labeled 'Number of total users to simulate' and contains the value '20000'. The second input field is labeled 'Spawn rate (users spawned/second)' and contains the value '250'. The third input field is labeled 'Host (e.g. http://www.example.com)' and contains the value 'http://derek-krinke-lb-61612628'. At the bottom right of the form is a green button labeled 'Start swarming'.