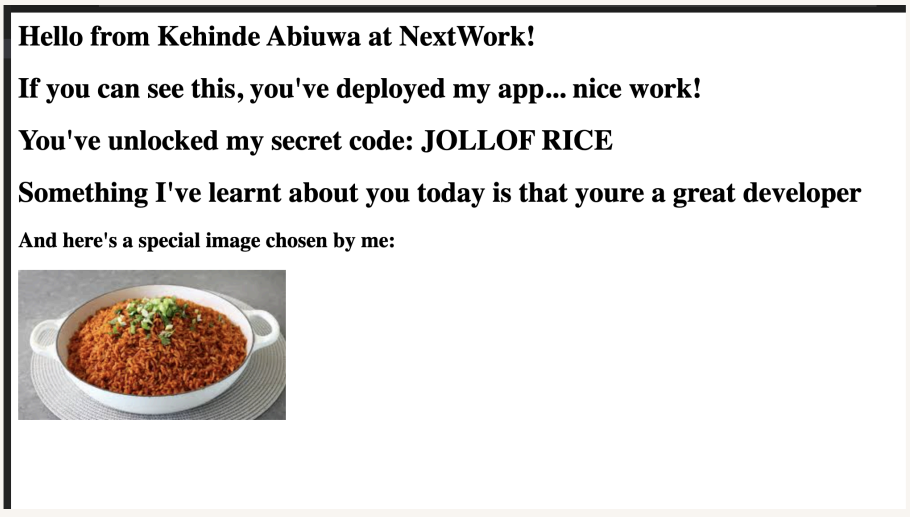
nextwork.org

Deploy an App Across Accounts

Kehinde Abiuwa



Kehinde Abiuwa
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to containerize my application with Docker, push the image to Amazon ECR, and collaborate by sharing and running container images with a project partner. I'm doing this project to learn how Docker simplifies packaging apps, how Amazon ECR provides secure storage and access control, and how container images can be shared and deployed across different environments for real-world collaboration.

What is Amazon ECR?

Amazon ECR is a fully managed container image registry provided by AWS. It is useful because it securely stores Docker images, integrates seamlessly with other AWS services, and makes it easy to share and deploy containerized applications without managing your own registry infrastructure. I used ECR to push my custom Docker image to the cloud and share it with my project buddy so they could pull and run my application on their machine.

One thing I didn't expect...

One thing I didn't expect in this project was running into permission errors when trying to pull my buddy's image. It taught me how important IAM roles and repository policies are when working with Amazon ECR and sharing images across accounts.

Kehinde Abiuwa
NextWork Student

This project took me...

This project took me 110 minutes. My biggest learning was how Docker and Amazon ECR work together — from building and tagging images locally, to pushing them into the cloud, fixing permissions, and finally pulling and running my buddy's app. It showed me the full end-to-end workflow of containerizing and sharing applications securely.

Kehinde Abiuwa
NextWork Student

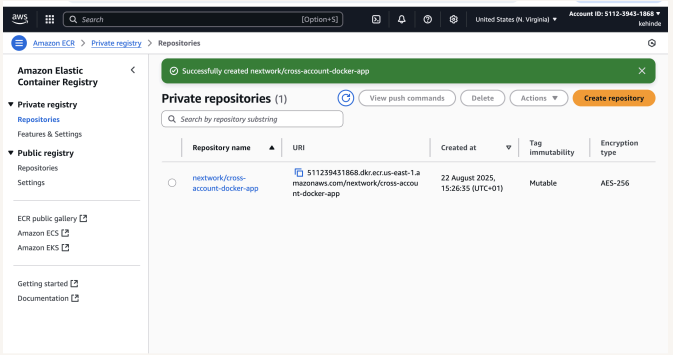
Creating a Docker Image

I set up a Dockerfile and an index.html file. Both files are needed because the Dockerfile defines how to build the image (using Nginx as the base and copying in my custom content), while the index.html file provides the actual webpage content that Nginx will serve inside the container.

the Dockerfile defines how to build the image (using Nginx as the base and copying in my custom content)

I also set up an ECR repository

ECR stands for Elastic Container Registry. It is important because it provides a secure, managed place to store and share Docker container images, integrates seamlessly with other AWS services, and eliminates the need to manage your own container registry infrastructure.



Set Up AWS CLI Access

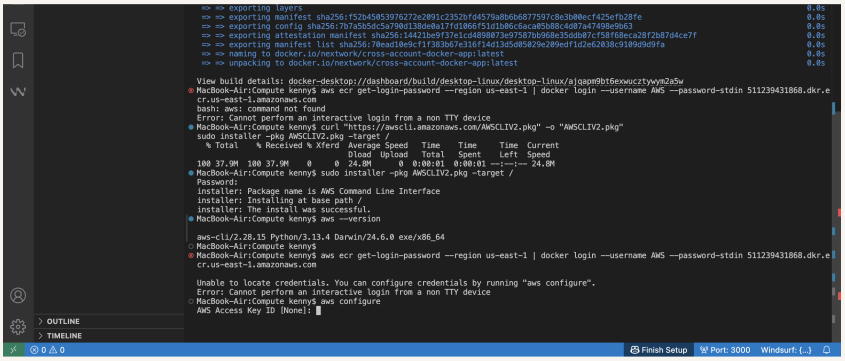
AWS CLI can let me run ECR commands

AWS CLI is the Amazon Web Services Command Line Interface, a tool that lets you manage AWS services directly from your terminal using commands. The CLI asked for my credentials because it needs my AWS Access Key ID and Secret Access Key to authenticate me, so it knows which AWS account I'm using and what permissions I have when I run commands.

To enable CLI access, I set up a new IAM user with the permission AmazonEC2ContainerRegistryFullAccess. I also set up an access key for this user, which means the AWS CLI can authenticate using that key pair (Access Key ID + Secret Access Key), allowing secure programmatic access to AWS without needing to log in through the web console.

To pass my credentials to the AWS CLI, I ran the command aws configure. I had to provide my AWS Access Key ID, Secret Access Key & default region name so the CLI could store my credentials and connect to my AWS account.

Kehinde Abiuwa
NextWork Student



Kehinde Abiuwa
NextWork Student

Pushing My Image to ECR

Push commands are the Docker commands you run to upload a container image from your local machine to a remote registry like Amazon ECR. They usually include tagging the image with the repository URI and then using docker push so the image is stored securely in the cloud and can be pulled later on any machine.

There are three main push commands

To authenticate Docker with my ECR repo, I used the command: aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <your_account_id>.dkr.ecr.us-east-1.amazonaws.com This command gets a temporary login token from AWS and passes it to Docker, allowing me to securely push and pull images from my ECR repository.

To push my container image, I ran the command: docker push 511239431868.dkr.ecr.us-east-1.amazonaws.com/nextwork/cross-account-docker-app:latest Pushing means I uploaded my locally built Docker image to my Amazon ECR repository so it can be stored securely in the cloud, shared with my buddy, and pulled to run on any other machine or AWS service.

placeholder

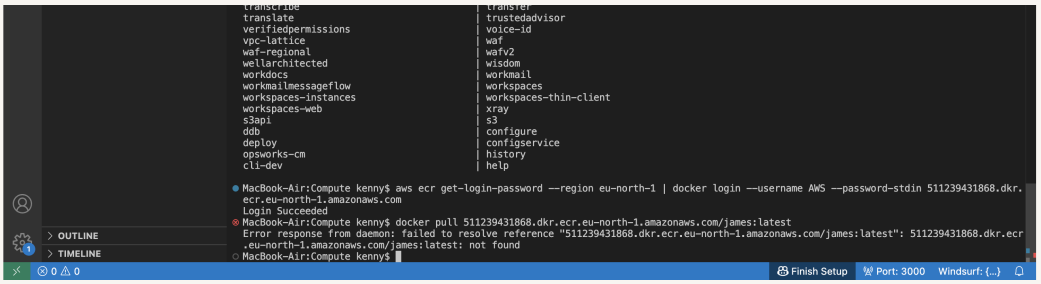
When I built my image, I tagged it with the label LATEST. This means I marked that image as the most recent or default version. Anyone who pulls my image without specifying a tag will automatically get this LATEST version.

It's a convenient convention, but it's good practice to also use versioned tags (like v1.0, v2.0) so you can track and roll back changes if needed.

Resolving Permission Issues

When I tried pulling my project buddy's container image for the first time, I saw the error denied: User is not authorized to perform ecr:BatchGetImage (or a similar permission/authentication error). This was because I hadn't yet authenticated Docker with their ECR registry or didn't have the correct IAM permissions to access their repository. Once I logged in with the proper aws ecr get-login-password command and ensured cross-account permissions were set, I was able to pull the image successfully.

To resolve each other's errors, we updated our ECR repository policies to grant cross-account pull access, authenticated with the correct AWS CLI login commands, and double-checked that we were using the right repository URIs and tags. These steps ensured we could successfully pull and run each other's container images.


transcribe | transfer
translate | trustedadvisor
verifiedpermissions | voice-id
vpc-lattice | waf
waf-regional | wafv2
wellarchitected | wisdom
workdocs | workmail
workmailmessageflow | workspaces
workspaces-instances | workspaces-thin-client
workspaces-web | xray
s3api | s3
ddb | configure
deploy | configservice
opsworks-cm | history
cli-dev | help
MacBook-Air:Compute kenny$ aws ecr get-login-password --region eu-north-1 | docker login --username AWS --password-stdin 511239431868.dkr.ecr.eu-north-1.amazonaws.com
Login Succeeded
MacBook-Air:Compute kenny$ docker pull 511239431868.dkr.ecr.eu-north-1.amazonaws.com/james:latest
Error response from daemon: failed to resolve reference "511239431868.dkr.ecr.eu-north-1.amazonaws.com/james:latest": 511239431868.dkr.ecr.eu-north-1.amazonaws.com/james:latest: not found
MacBook-Air:Compute kenny$
OUTLINE
TIMELINE
0 0
Finish Setup
Port: 3000
Windsurf: {...}

The place to learn & showcase your skills

Check out nextwork.org for more projects