User Manual: Deploying Docker Container to Azure and deploy in Azure App Service

Introduction

This section shows the deployment of Docker containers to Azure as well as deploying the container as a Web app using Azure App Service.

Prerequisites

- An Azure account.
- Docker installed on your local machine.
- Azure CLI Installed on your local machine or access to Azure Portal

1. Cloning the Projects

- Cloning the server project and going to root of project.

```
git clone https://github.com/MBeattie02/code_review_tool_server.git
```

- Cloning the client project and going to root of project.

```
git clone https://github.com/MBeattie02/code_review_tool_client.git
```

2. Prepare Docker Containers

In the root of the project there is a Dockerfile to facilitate the building of the docker image.
To build the image based on this Dockerfile, run the following commands:

Client application

- Building docker image for client:

```
docker build -t frontend .
```

- To run the client image locally for testing, run the following command:

```
docker run -p 3000:3000 frontend --name frontend-container
```

Server application

- Package the application in a jar to be used in the Dockerfile

```
mvn clean package
```

- Building docker image for server:

```
docker build -t serverside .
```

- To run the server image locally for testing, run the following command:

```
docker run -p 8080:8080 serverside --name server-container
```

Deploying to Azure

Deploying the application to Azure involves two steps. First, we will push each docker image to the Azure Container Registry. Once that is completed, we will create a new Web App for each application based on the Docker image created.

3. Create an Azure Resource Group

- Launch Azure CLI: Open your command-line interface and log in to your Azure account using:

```
az login
```

- Create a Resource Group: Use the Azure CLI to create a resource group, this is a container that manages multiple Azure resources using az group create:

```
az group create --name CSproject --location ukwest
```

--name: tag specifies the resource group name
--location: tag gives the region where it is located

4. Create an Azure Container Registry

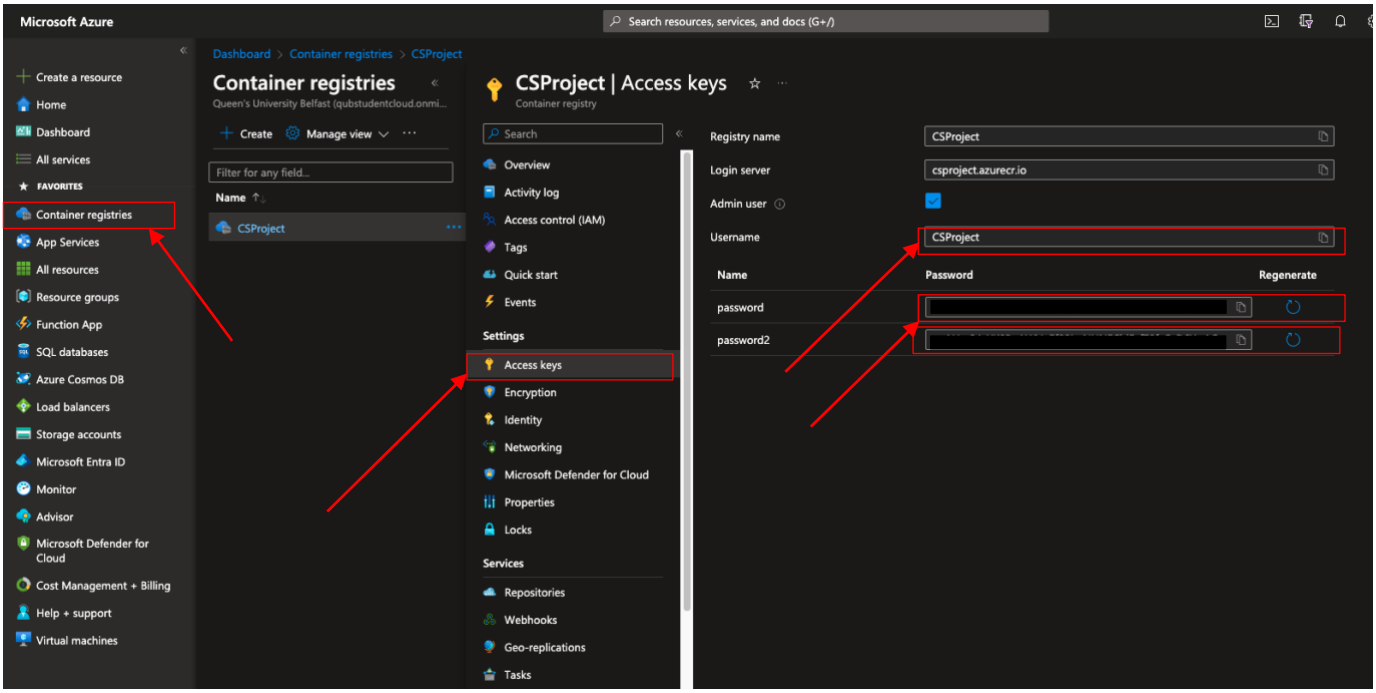
- Create a Azure Container Registry to hold the docker images of the applications: use the az acr create

```
az acr create --resource-group CSproject --name CSProject --sku basic
```

--resource-group: Specifies the name of the resource group created
--name: specifies the name of the registry (the name must be globally unique)
--sku: provides the several service tiers.

5. Obtaining Access Key from the Registry

- Docker access must be enabled in the Azure container registry. This is crucial to the deployment process as it enables you to remotely log in to the Azure container registry through the CLI and be able to push images to it.
- To do that, once the registry resources have been created successfully, open it, and locate the **Access Keys** link under the Settings section in the Container registry page.
- This will allow the retrieval of the username and password required for logging into the repository on your local machine.



6. **Push Your Docker Image to Azure Container Registry**
    - Log in to ACR: Login requires the username and password of the users access key and the name of the login server.

```
docker login -u CSProject -p [password_of_repository] csproject.azurecr.io
```

-u: The username obtained for the container registry.
-p: The password from the container registry.

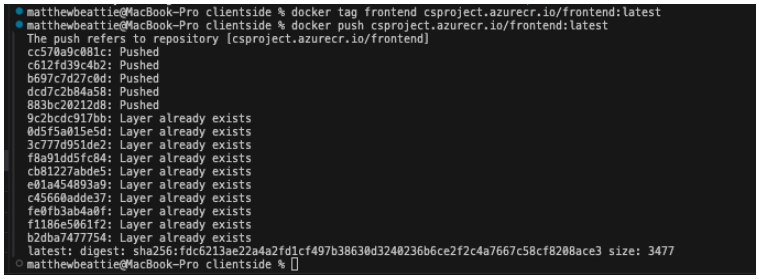
- Tag frontend Image for ACR: Tag your Docker image with the registry login server address:

```
docker tag frontend csproject.azurecr.io/frontend:latest
```

- Push the frontend Image to ACR: Deploy image to ACR using docker push:

```
docker push csproject.azurecr.io/frontend:latest
```

Expected Successful Console Output



- Tag server Image for ACR: Tag Docker image with the registry login server address:

```
docker tag serverside csproject.azurecr.io/serverside:latest
```

- Push the server Image to ACR: Deploy image to ACR using docker push:

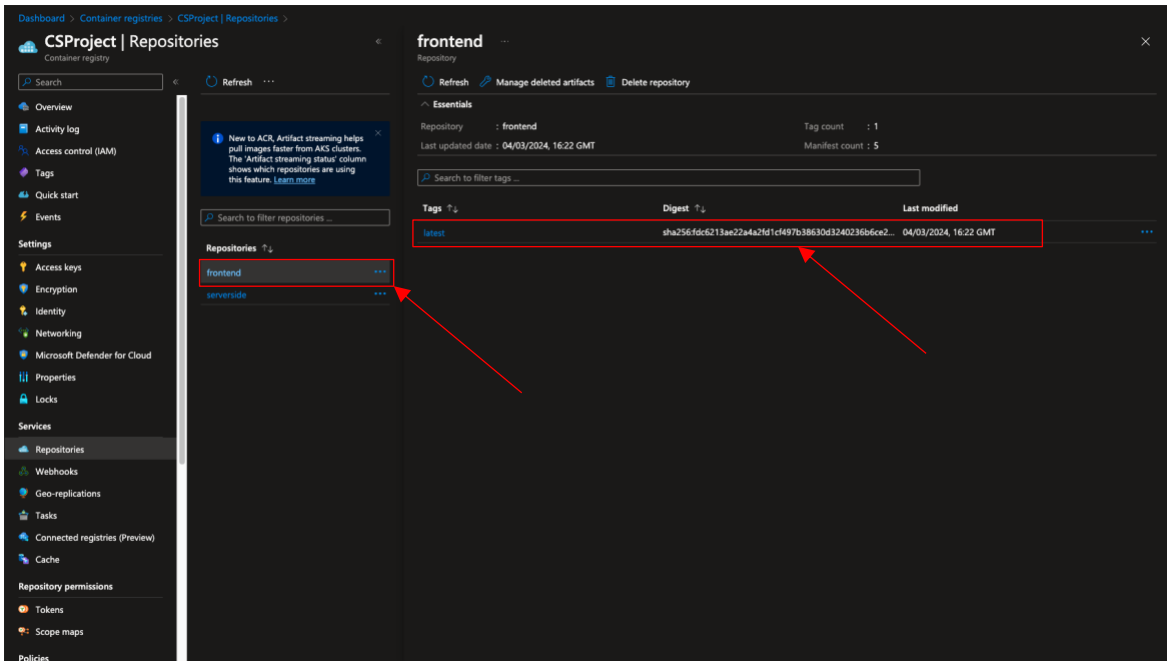
```
docker push csproject.azurecr.io/serverside:latest
```

Expected Successful Console Output

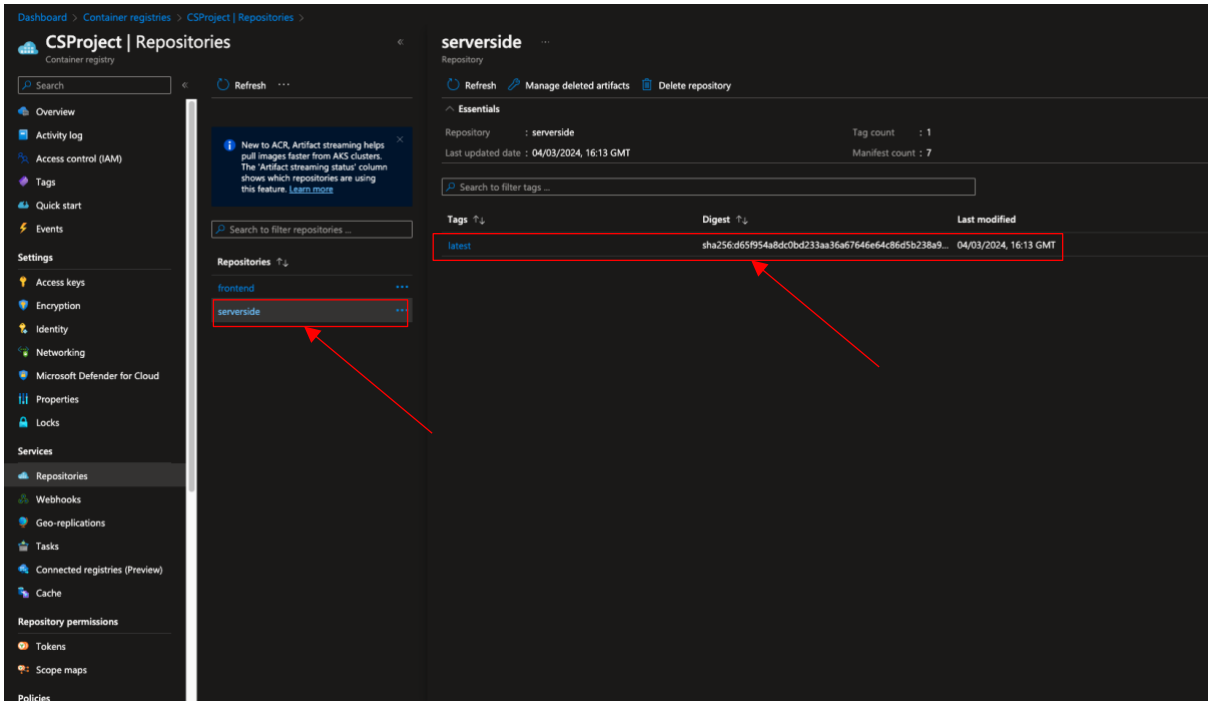
```
matthewbeattie@MacBook-Pro ServerSide % docker tag serverside csproject.azurecr.io/serverside:latest
matthewbeattie@MacBook-Pro ServerSide % docker push csproject.azurecr.io/serverside:latest
The push refers to repository [csproject.azurecr.io/serverside]
7ffd770db0af: Pushed
d34077631442: Layer already exists
6be690267e47: Layer already exists
13a34b6fff78: Layer already exists
9c1b6dd6c1e6: Layer already exists
latest: digest: sha256:d65f954a8dc0bd233aa36a67646e64c86d5b238a932642bfa9d00b0f967611dd size: 1372
matthewbeattie@MacBook-Pro ServerSide %
```

The images should now be deployed to the Azure registry, this can be confirmed by checking the Azure Portal:

Frontend image deployed in Registry:

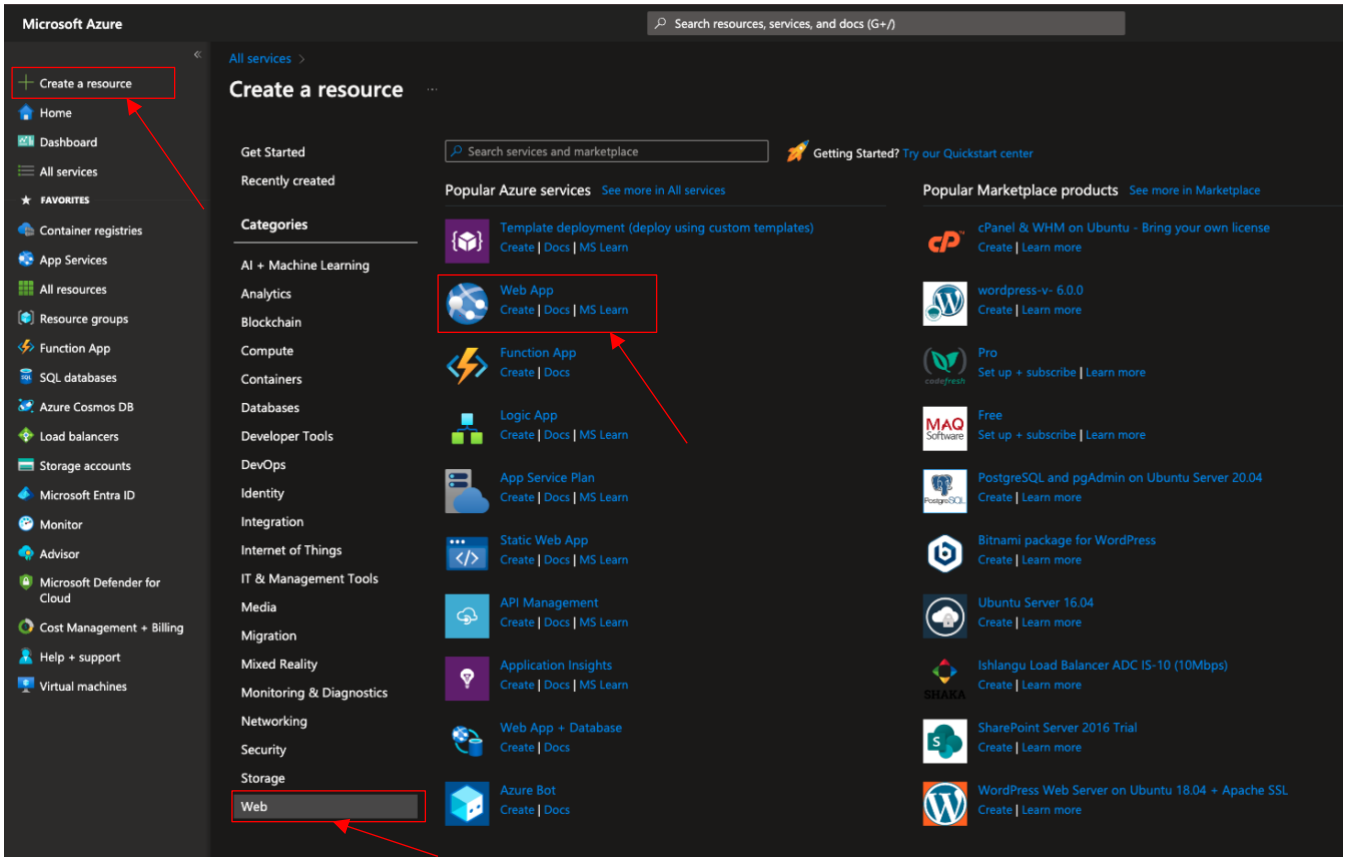


Server image deployed in Registry:



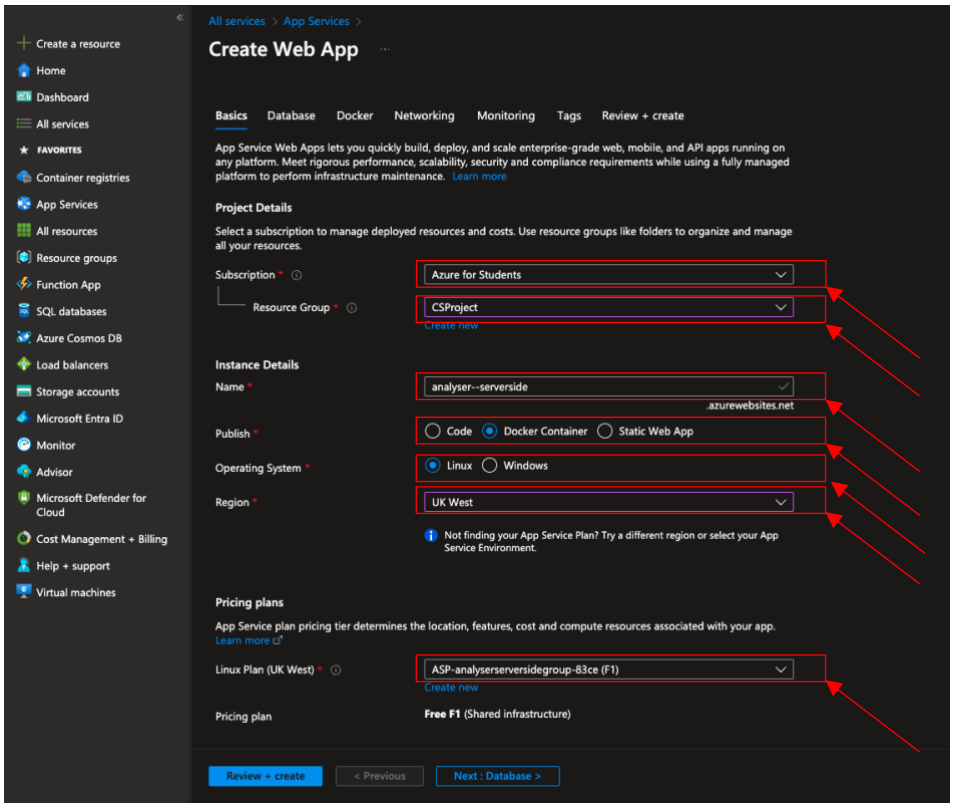
7. Creating an Azure Web App for the containers

- Next, is to create an Azure Web App for each application and connect it with the container images stored in the Azure Container Registry.
- Navigate to the Azure portal homepage and click **Create a resource.**
- Then select **Web > Web App for Containers** to create a new web app service instance.

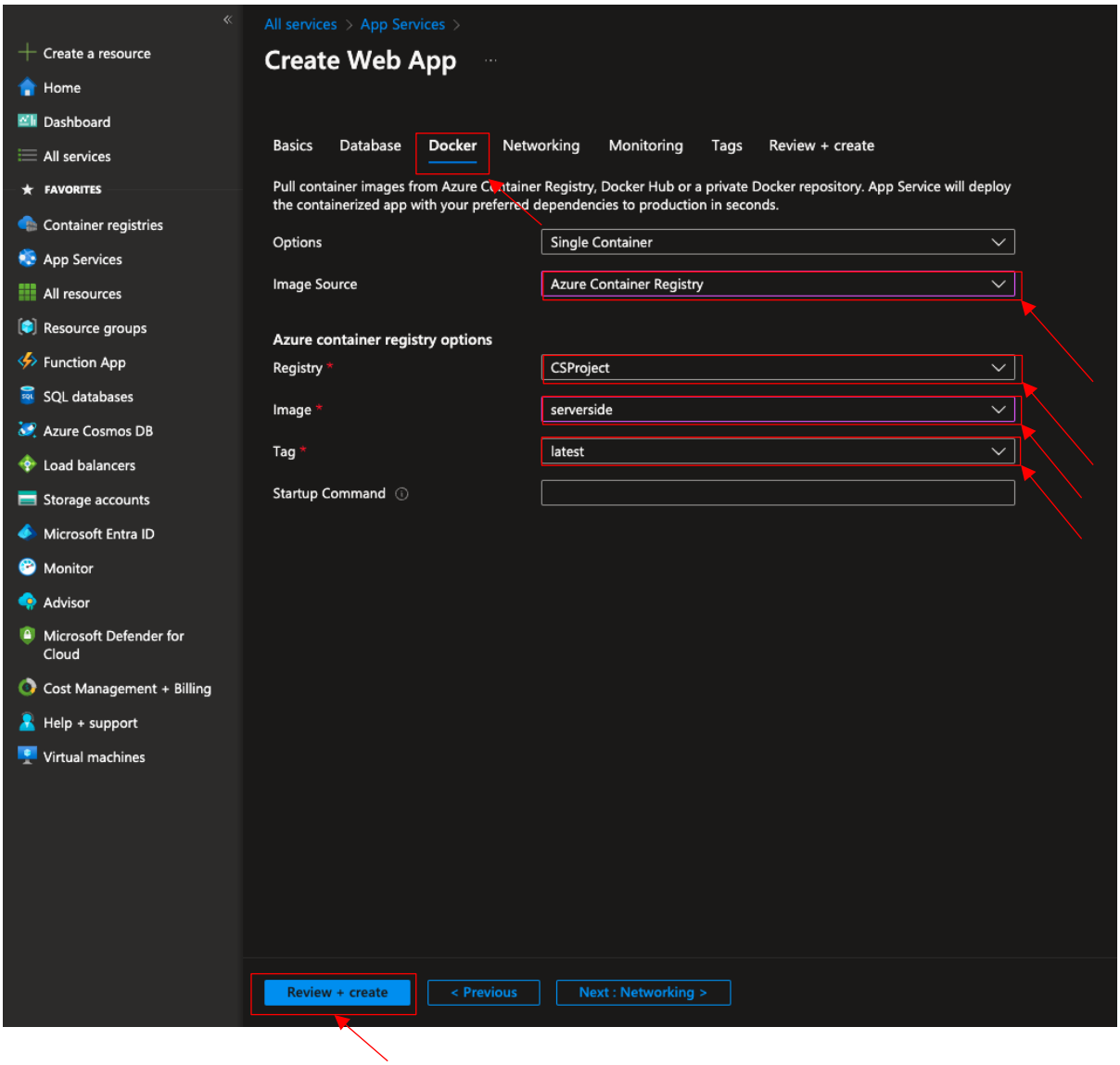


Create Web App Server

- This will then redirect to the Create Web App page.
- Select:
    - Azure subscription
    - resource group – CSProject as created earlier.
    - name Web App – analyser-serverside
    - Region – Closest datacentre to users' location
    - Pricing Plan
- Docker container should be selected by default, otherwise, select it.
    - Operating system should be Linux.



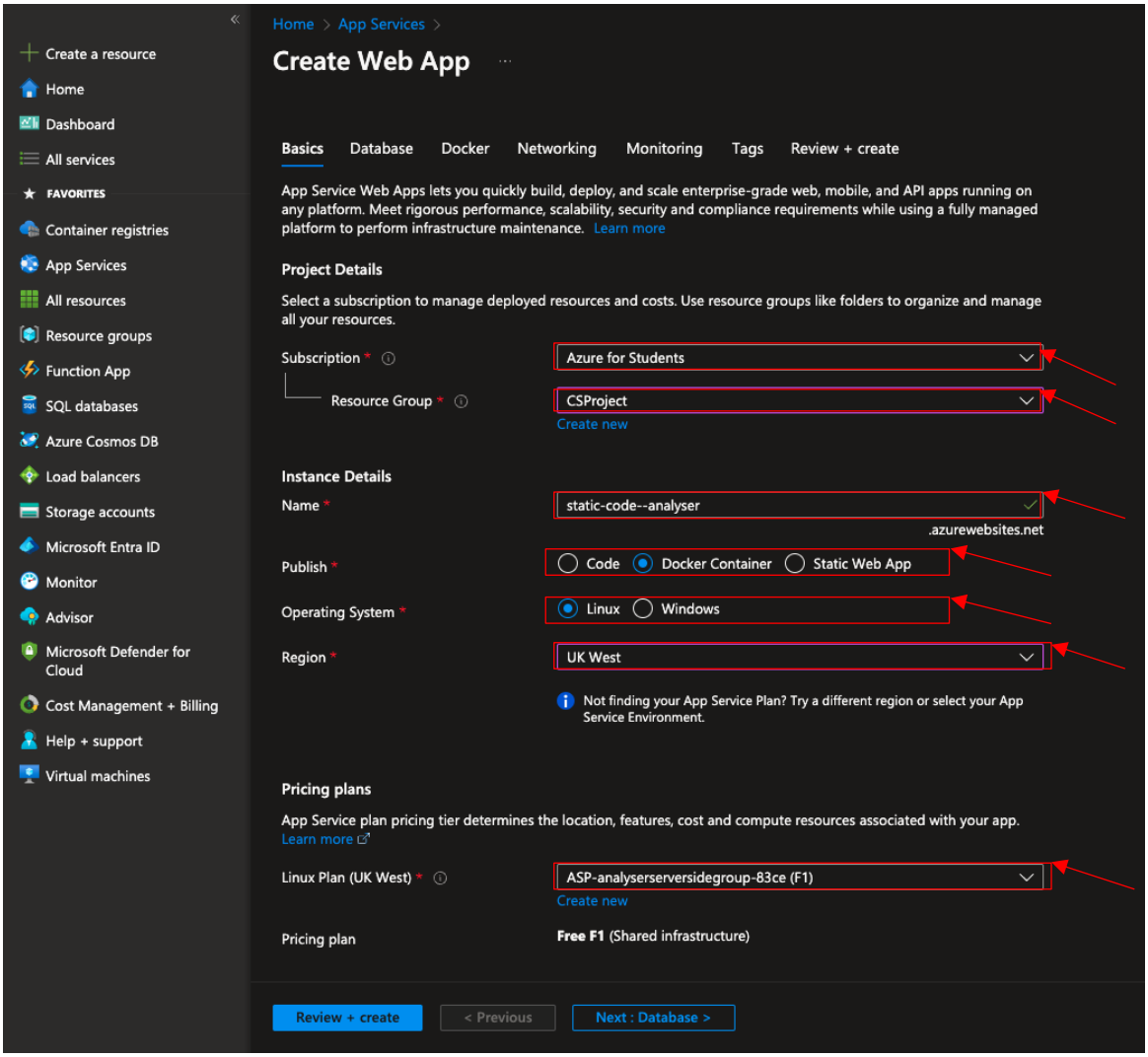
- Next, click on the **Docker** tab and select the image source and its respective docker image.
    - Image Source should be: Azure Container Registry
    - Azure Container registry options:
        - Registry: CSProject
        - Image: serverside
        - Tag: latest



- Click **Review + Create** and you will be redirected to a page where you can review the web app details. If this is correct, click **Create** to set up a new Azure web app.

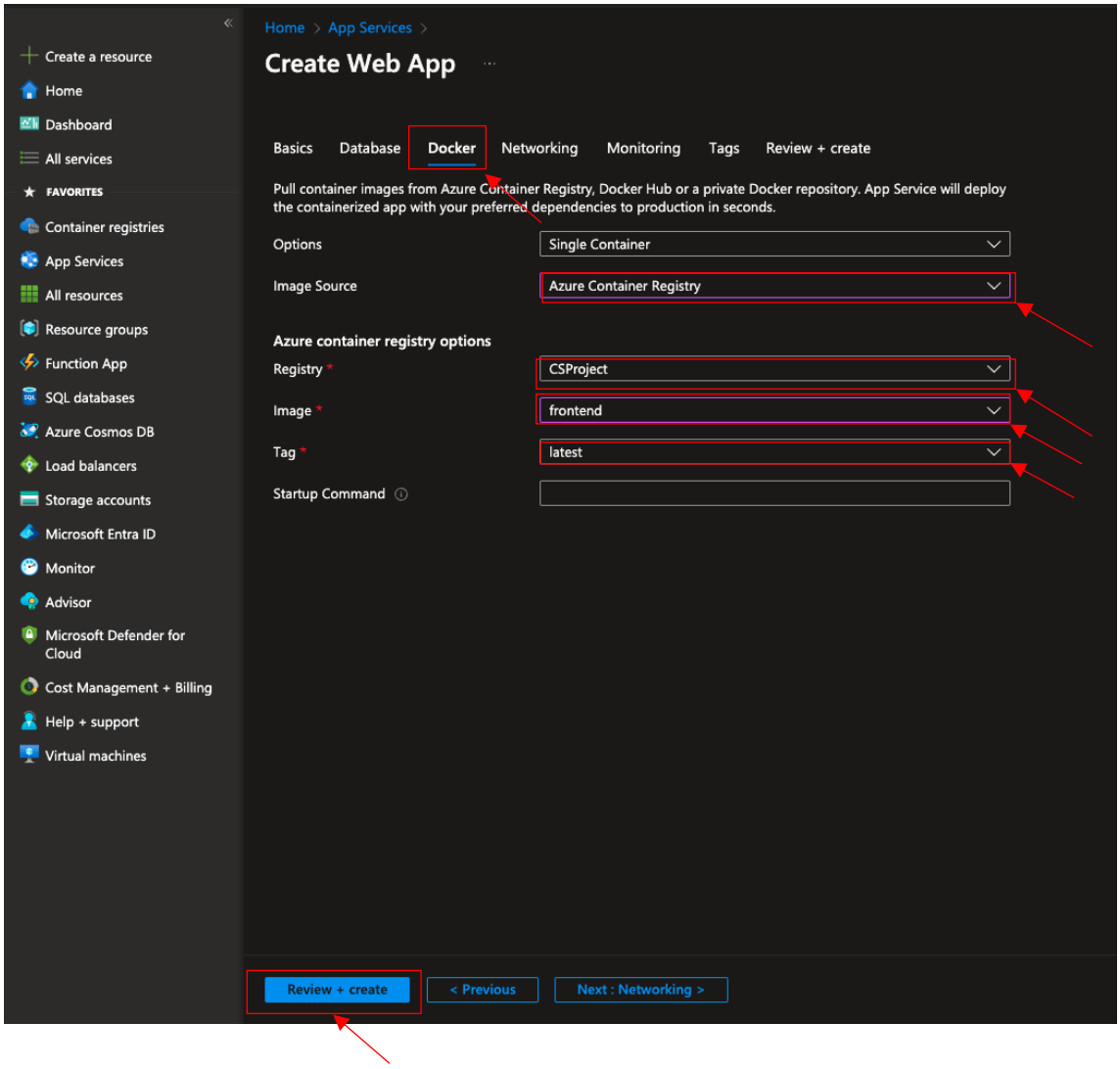
Create Web App Client

- This will then redirect to the Create Web App page.
- Select:
    - Azure subscription
    - resource group – CSProject as created earlier.
    - name Web App – static-code-analyser
    - Region
    - Pricing Plan
- Docker container should be selected by default, otherwise, select it.
    - Operating system should be Linux.



- Next, click on the **Docker** tab and select the image source and its respective docker image.
    - Image Source should be: Azure Container Registry
    - Azure Container registry options:
        - Registry: CSProject
        - Image: serverside
        - Tag: latest

- Click **Review + Create** and you will be redirected to a page where you can review the web app details. If this is correct, click **Create** to set up a new Azure web app.



Once the process is completed, you can visit the URL generated for your application, you will be able to view the app as deployed to Azure. The URL is in this format https://<APP_NAME>.azurewebsites.net .

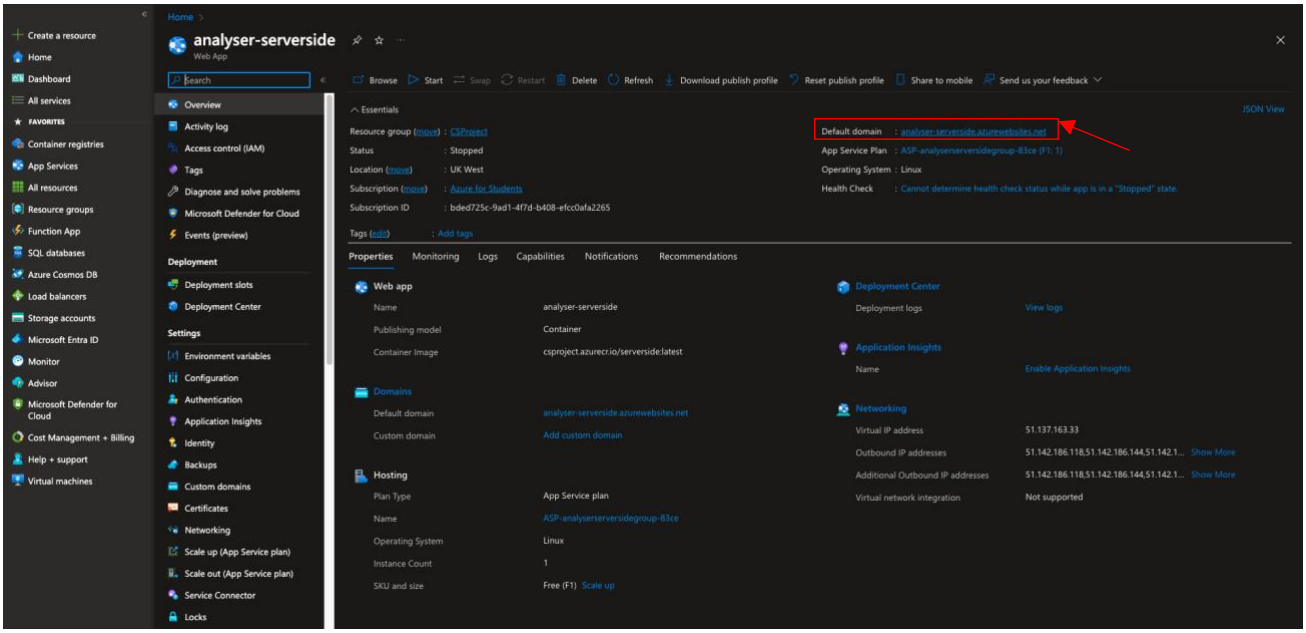
- Client – Main UI user interacts with.

https://static-code-analyser.azurewebsites.net

- Server – Deploys backend functionality called by frontend.

https://analyser-serverside.azurewebsites.net

Portal showing default domain assigned:



Cloud Database Deployment MongoDB Atlas

Creating a MongoDB Atlas database involves setting up a managed MongoDB cluster in the cloud. MongoDB Atlas is a fully managed cloud database service that provides scalable MongoDB databases on AWS, Azure, and Google Cloud. This manual will show the setup process step by step.

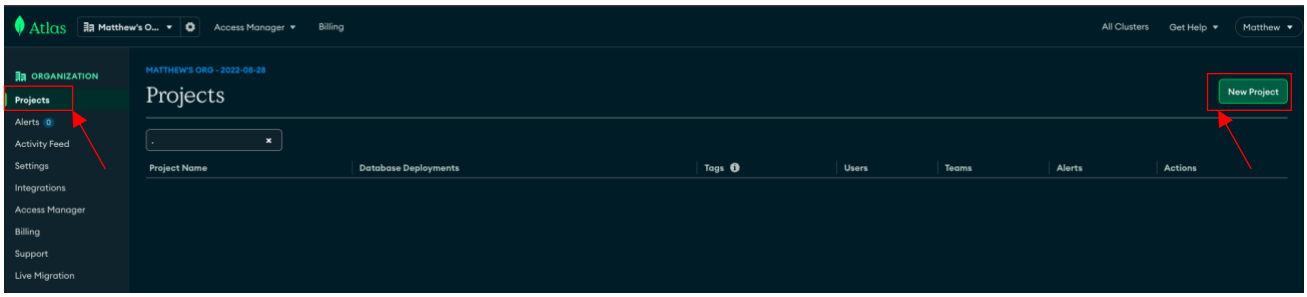
Prerequisites
An active MongoDB Atlas account. If you don't have one, you'll need to sign up at MongoDB Atlas.

1) **Sign Up and Log In**

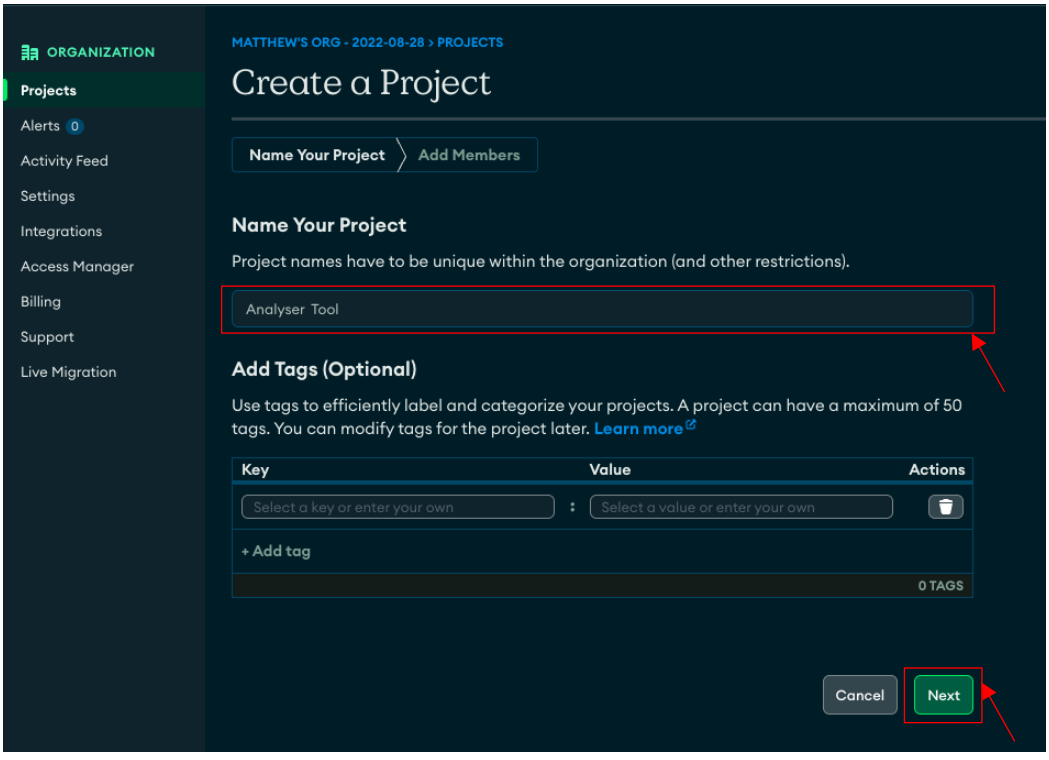
- Log In: Once your account is set up, log in to your MongoDB Atlas dashboard.

2) **Step 2: Create a New Project**

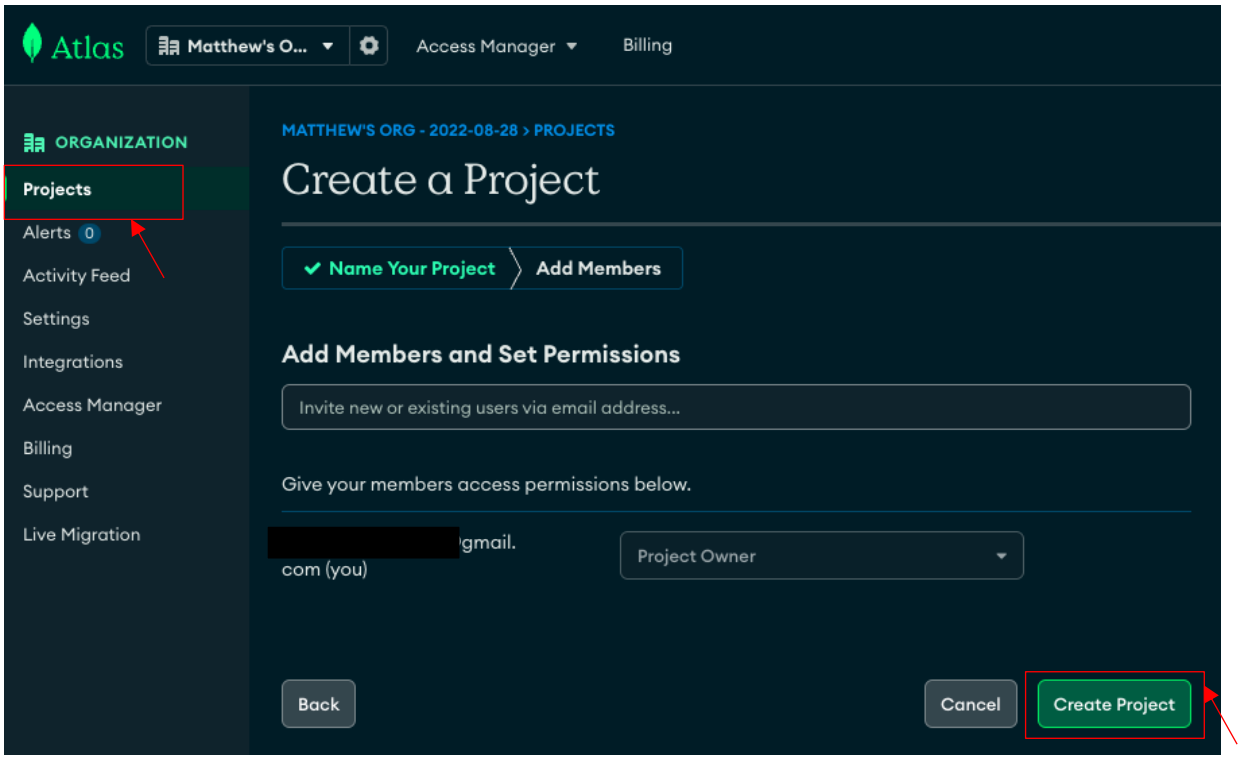
- Navigate to the Projects Page: Click the "Context" dropdown in the top left corner and select "Projects".
- Create Project: Click the "New Project" button.



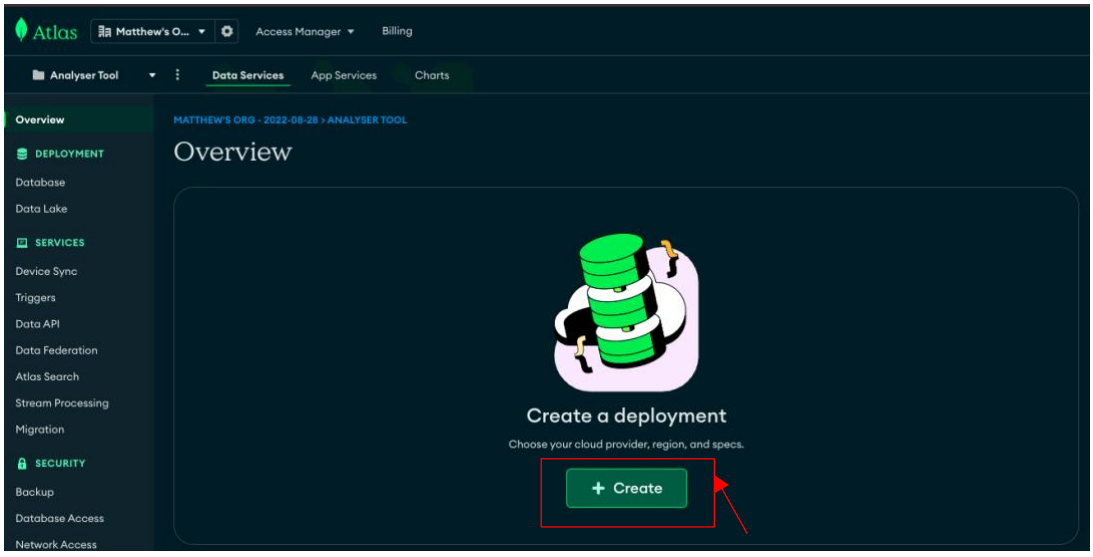
- Give project a name **Analyser Tool**, and then click "Next".



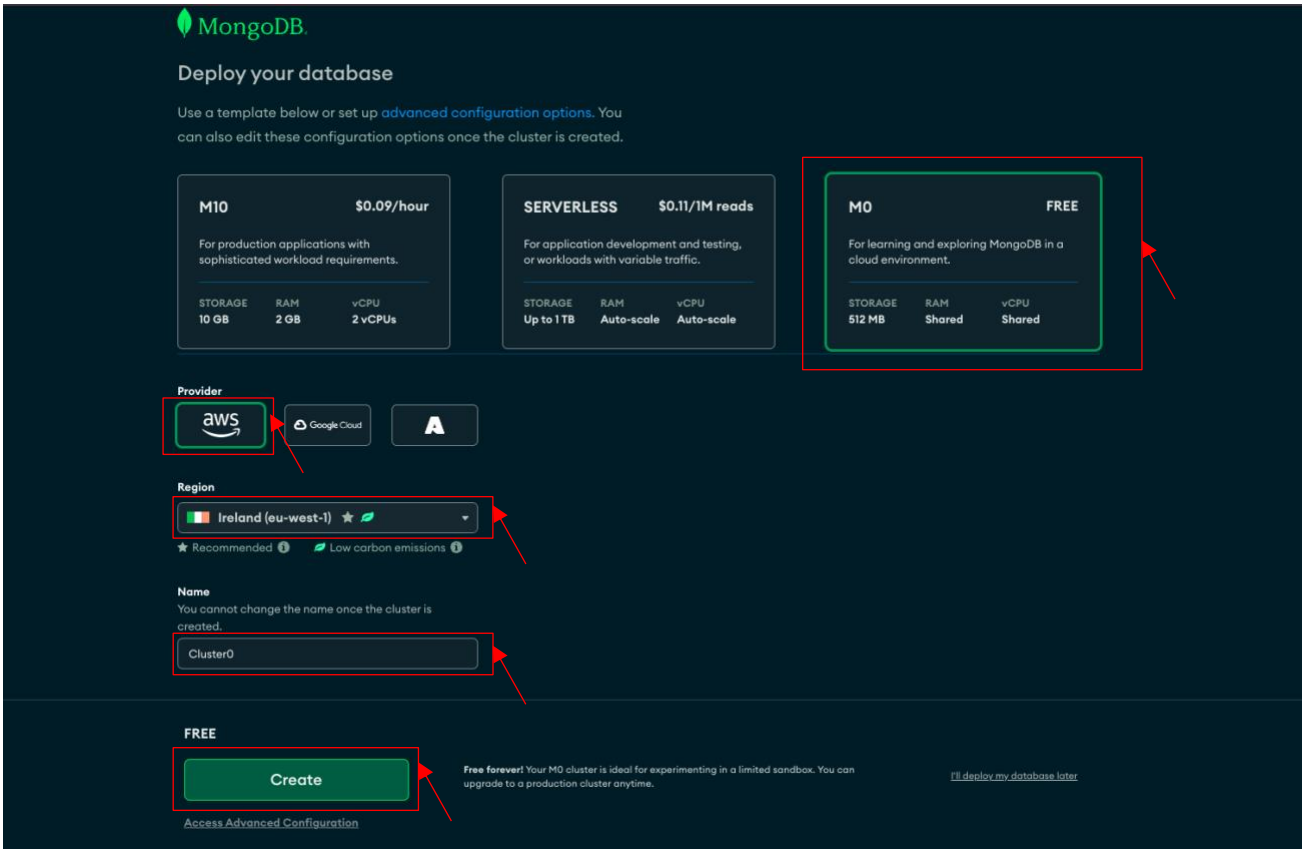
- Add Members: Optionally, you can add other members to your project by entering their email addresses. Click "Create Project" when done.



3) Build a New Cluster

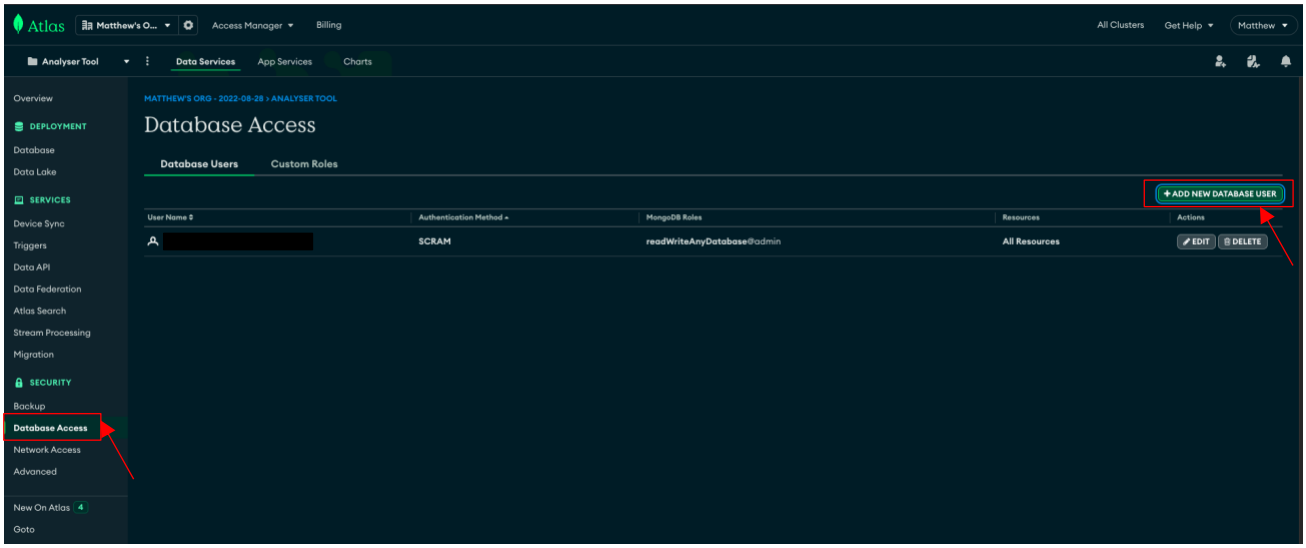


- Choose a Cloud Provider and Region: After creating a project, you'll be prompted to build a cluster.
- Select your preferred cloud provider **AWS** and choose a region that is closest to your application's users for the best performance.
- Select Cluster Tier: Choose a cluster tier. For beginners or small projects, the free tier (M0) is sufficient. Click "Create Cluster" after making your selection.



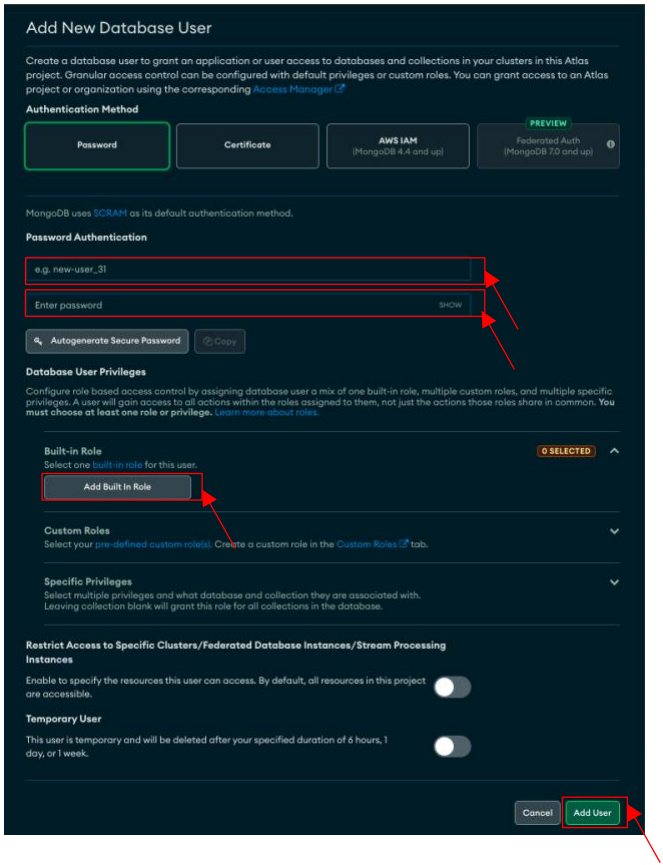
4) Step 4: Configure Database Access

- Database Access: Navigate to the "Database Access" tab on the left-hand sidebar.



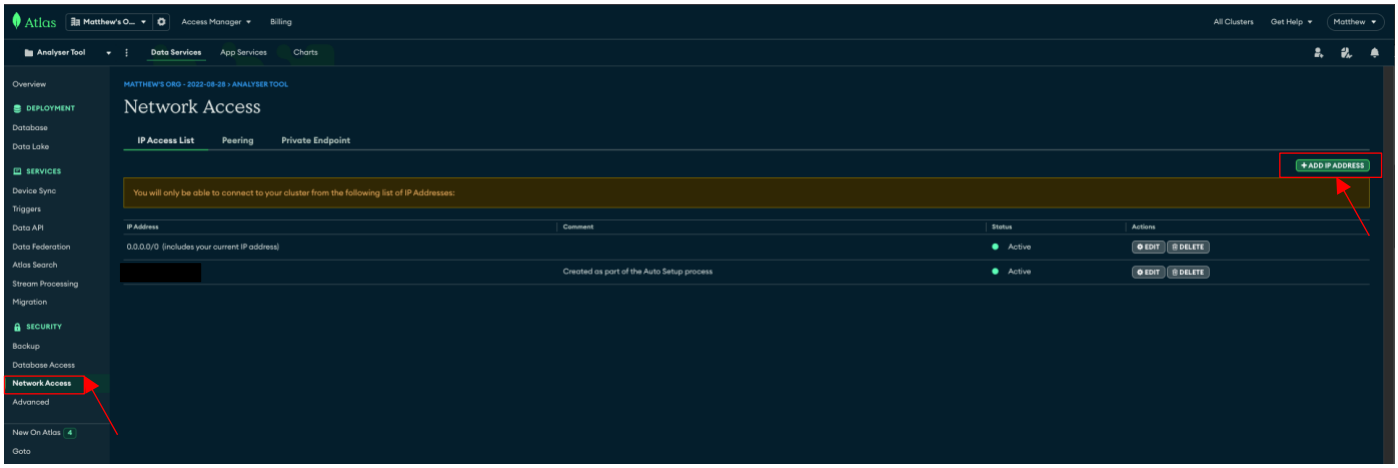
- Add New Database User: Click the "ADD NEW DATABASE USER" button. Create a user with a username and password. Assign appropriate permissions for this user.
    - "Read and write to any database"

- Save User: Click "Add User" to save the new database user.



5) Step 5: Set Up Network Access

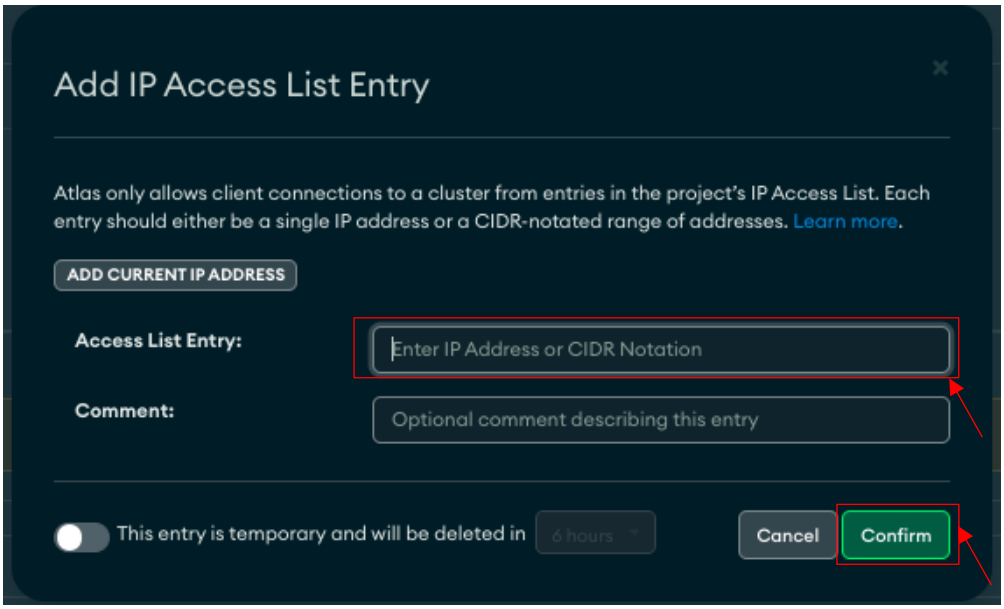
- Network Access: Navigate to the "Network Access" tab.



- Add IP Address: Click the "ADD IP ADDRESS" button.

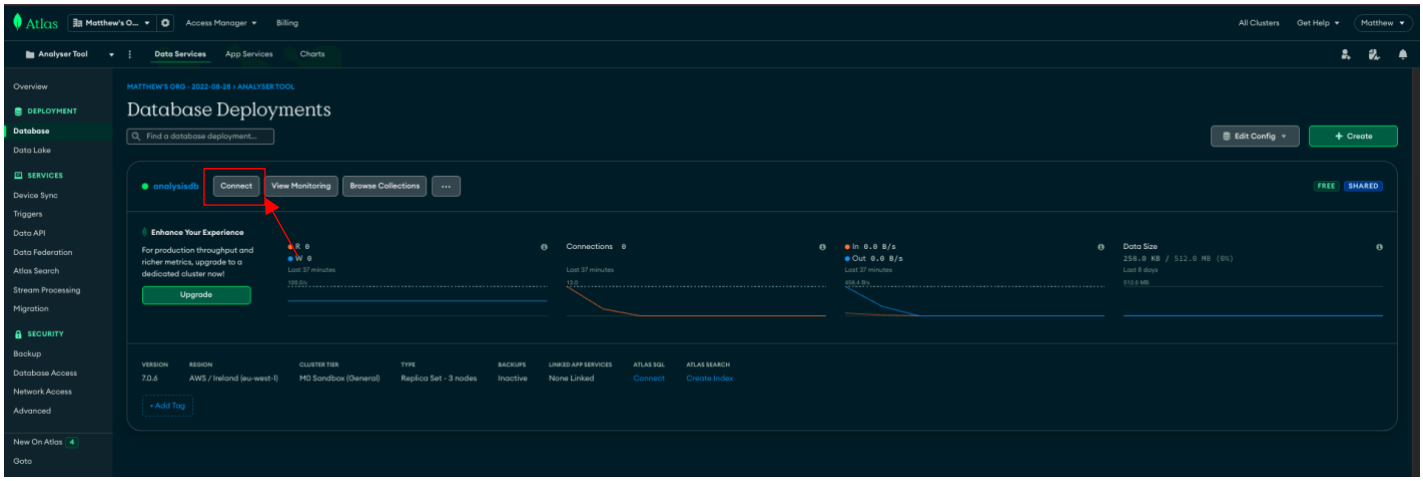
- Here you can whitelist IP addresses that are allowed to connect to your database.
    - For testing purposes, you can allow access from anywhere by clicking "Allow Access from Anywhere".

- Confirm: Click "Confirm" to save your IP whitelist.



6) Step 6: Connect to Your Database

- Clusters Dashboard: Go back to the "Clusters" page.
- Connect Button: Go to the created cluster and click the "Connect" button.



- This project connects by setting the mongodb uri in the applications.properties of the server codebase.

```
spring.data.mongodb.uri=mongodb+srv://[username]:[password].mongodb.net/analysisdb
```