Guide: Deploying WordPress using IAM Role, RDS, Docker, ECS, and pushing it to ECR

April 03, 2023

Before we get started, it's important to understand the role of each component in this deployment:

IAM role: Allows EC2 instances to securely access AWS services such as RDS without needing to store AWS credentials on the instance itself.

Docker: A containerization platform that enables the bundling of an application and all its dependencies into a single package, making it easier to deploy and manage.

ECR: Elastic Container Registry, a fully-managed Docker container registry that makes it easy to store, manage, and deploy Docker container images.

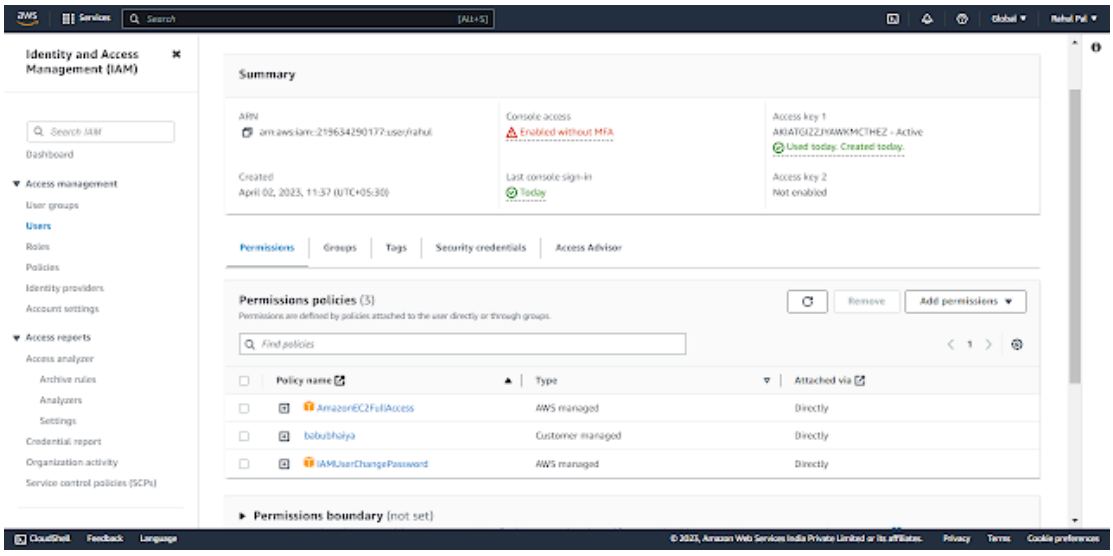
Now that we have a basic understanding of each component, let's dive into the deployment process.

Create an IAM Role

The first step is to create an IAM role that will be used by the EC2 instances to access the RDS database.

To create a new IAM role, navigate to the IAM console and click on "Roles" in the left-hand menu. Then click the "Create role" button and select "EC2" as the type of trusted entity.

Next, select the "AmazonEC2RoleforSSM" policy and add an inline policy that allows access to the RDS instance.



Launch an EC2 Instance

In the EC2 console, click the "Launch Instance" button and select a suitable AMI (Amazon Machine Image). I recommend using the latest Ubuntu Server AMI.

After selecting the AMI, choose an instance type and configure the instance settings as per your requirements. On the "Configure Security Group" page, create a new security group that allows inbound traffic on ports 80 and 443.

On the "Configure Instance Details" page, select the IAM role that you created in step one.

Finally, launch the instance and SSH into it.

Install Docker

Once you've logged in to the EC2 instance, first you need to install Docker:

Create a Dockerfile

Next, create a Dockerfile that will define the WordPress image. Here's an example:

```
    apt install -y \
    vim \
    wget \
    mariadb-client

COPY php.ini /usr/local/etc/php

RUN wget https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar && \
    php wp-cli.phar --info && \
    chmod +x wp-cli.phar && \
    mv wp-cli.phar /usr/local/bin/wp && \
    rm /usr/local/etc/php/php.ini-development && \
    rm /usr/local/etc/php/php.ini-production
ubuntu@ip-172-31-92-157:~/wp-project$
```

In this example, we're using the official WordPress image as the base image and setting environment variables for the RDS host, user, password, and database name. We're also copying the wp-config.php file to the container.

```
ubuntu@ip-172-31-92-157:~/wp-project$ cat Dockerfile
FROM wordpress:latest

RUN apt update && \
    apt upgrade -y && \
    apt autoremove && \
    apt install -y \
    vim \
    wget \
    mariadb-client

COPY php.ini /usr/local/etc/php

RUN wget https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar && \
    php wp-cli.phar --info && \
    chmod +x wp-cli.phar && \
    mv wp-cli.phar /usr/local/bin/wp && \
    rm /usr/local/etc/php/php.ini-development && \
    rm /usr/local/etc/php/php.ini-production
ubuntu@ip-172-31-92-157:~/wp-project$
```

Build and Tag the Docker Image

Now that the Dockerfile is ready, build and tag the Docker image with the following command:

docker build -t <your-ecr-repo>/wordpress:latest .

Replace <your-ecr-repo> with the name of your ECR repository.

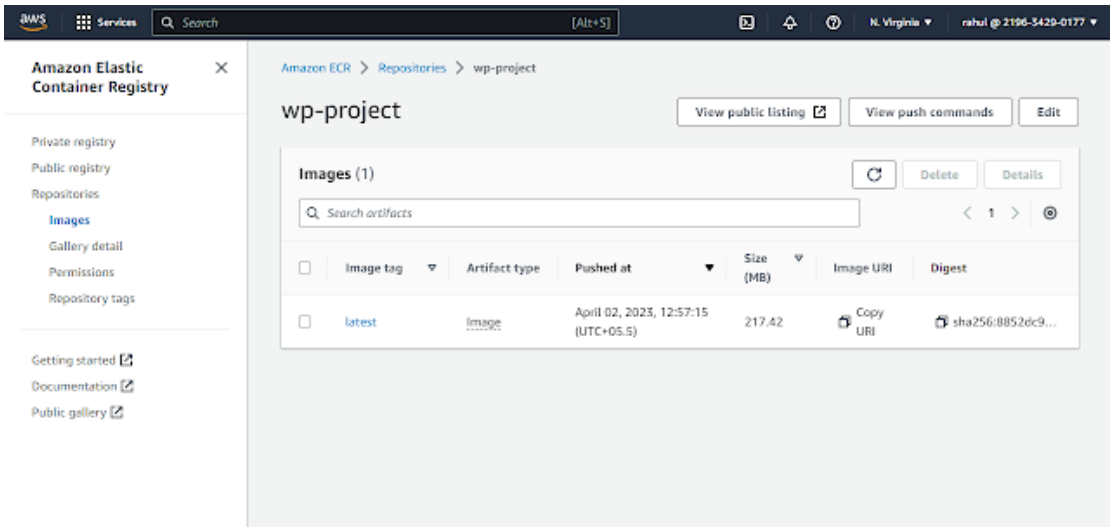
Push the Docker Image to ECR

Before you can push the Docker image to ECR, you need to authenticate the Docker CLI to your ECR registry. Run the following

$(aws ecr get-login --no-include-email --region <region>)

Replace <region> with your AWS region.

Now you can push the Docker image to ECR with



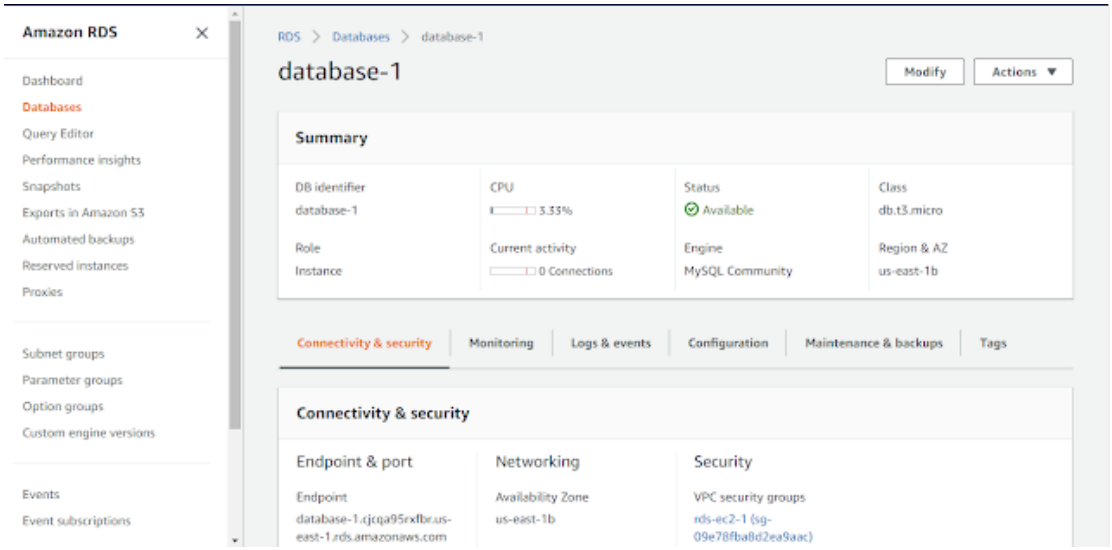
Set up an RDS Instance

Now that we have our EC2 instance and Docker image set up, let's set up the RDS instance that our WordPress site will use.

In the AWS console, navigate to the RDS service and click the "Create database" button. Select the database engine that you want to use (I recommend MySQL or Aurora), and choose an appropriate instance type.

On the "Settings" page, give your database a name and create a new username and password for it.

On the "Connectivity" page, select the VPC that your EC2 instance is in and choose the appropriate subnet group.



Map the RDS Instance to the Docker Container

Now that the RDS instance is set up, we need to map it to the Docker container.

First, create a wp-config.php file that defines the WordPress database settings. Here's an example:

```php
<?php
define('DB_NAME', '<rds-database>');
define('DB_USER', '<rds-user>');
define('DB_PASSWORD', '<rds-password>');
define('DB_HOST', '<rds-host>');
define('DB_CHARSET', 'utf8');
define('DB_COLLATE', '');
define('AUTH_KEY',         'put your unique phrase here');
define('SECURE_AUTH_KEY',  'put your unique phrase here');
define('LOGGED_IN_KEY',    'put your unique phrase here');
define('NONCE_KEY',        'put your unique phrase here');
define('AUTH_SALT',        'put your unique phrase here');
define('SECURE_AUTH_SALT', 'put your unique phrase here');
define('LOGGED_IN_SALT',   'put your unique phrase here');
define('NONCE_SALT',       'put your unique phrase here');
$table_prefix  = 'wp_';
define('WP_DEBUG', false);
if ( !defined('ABSPATH') )
        define('ABSPATH', dirname(__FILE__) . '/');
require_once(ABSPATH . 'wp-settings.php');
```

host> with the appropriate values.

Next, run the following command to start a Docker container and map the RDS instance to it:

docker run -d --name wordpress -p 80:80 -v /path/to/wp-config.php:/var/www/html/wp-config.php <your-ecr-repo>/wordpress:latest

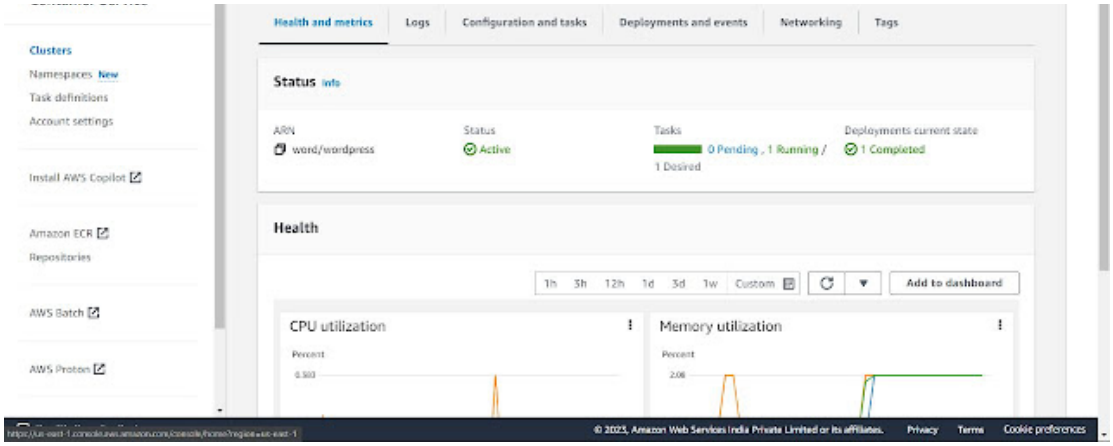
Replace /path/to/wp-config.php with the path to your wp-config.php file, and <your-ecr-repo> with the name of your ECR repository.

This command starts a new Docker container with the name "wordpress", maps port 80 on the container to port 80 on the EC2 instance, and mounts the wp-config.php file to the container.

And that's it! You should now have a WordPress site up and running on your EC2 instance, using an RDS instance for the database. By pushing your Docker image to ECR, you can easily deploy it to other EC2 instances or AWS services.

Deploy to ECS using Fargate

Deploying your WordPress site to ECS using Fargate allows you to manage your containers at scale and takes care of the underlying infrastructure for you.



First, create a new task definition in ECS that defines your WordPress container. In the task definition, specify the container image that you pushed to ECR and any environment variables that your WordPress site needs.

Next, create a new ECS service that uses the task definition. In the service, specify the desired number of tasks (containers) and the cluster that the tasks should be launched in.

Once your service is up and running, your WordPress site should be accessible from the internet. You can set up a load balancer in front of your service to distribute traffic across multiple containers.

To deploy updates to your WordPress site, simply push a new container image to ECR and update your task definition. ECS will automatically roll out the new version of your container across your service.

And that's it! You now have a highly scalable WordPress site running on ECS with Fargate.
In conclusion, deploying a WordPress site using IAM roles, Docker, RDS, and pushing to ECR is a powerful way to create a highly scalable

By leveraging IAM roles, you can ensure that your EC2 instance and containers are secure and have the necessary permissions to access AWS services.

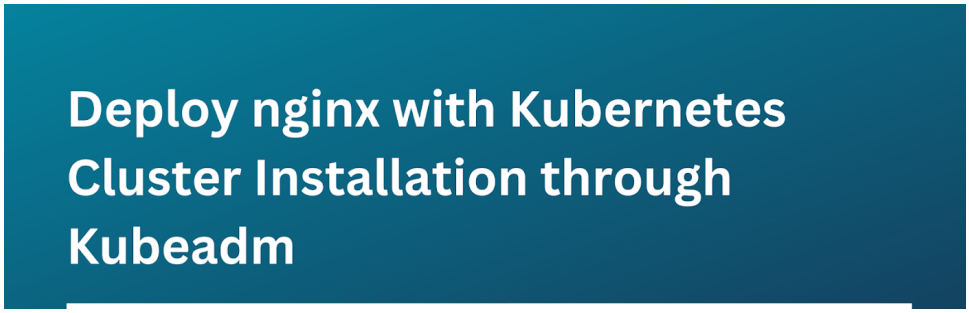
Using Docker makes it easy to package your application and its dependencies into a single, portable image that can be easily deployed to other environments.

RDS provides a scalable and managed database solution for your WordPress site, while ECR allows you to store and deploy your Docker images to other AWS services.

And finally, deploying your WordPress site to ECS using Fargate allows you to manage your containers at scale and takes care of the underlying infrastructure for you.

By following these steps, you can create a highly scalable and resilient WordPress site that can handle large amounts of traffic and automatically scale to meet demand

DEVOPS



Deploy nginx with Kubernetes Cluster Installation through Kubeadm

First we need to create 2 EC2 instances i.e t2 medium for master node and t2 micro for worker node. And allow ports for their connection. Run all below command on both master and worker nodes sudo apt update -y sudo apt install docker.io -y sudo systemctl start docker sud ...

Nginx Web Deployed on Kubernetes Cluster Using Deploy, Service, and Ingress Yaml Files

In this blog, we are going to explore how to deploy nginx to Kubernetes using a Deployment file, Service file, and Ingress file. These files are essential components of any Kubernetes application deployment, and they work together to ensure that your application is availa ...