AWS Hosting Guide

Yentl Hendrickx

May 25, 2023

Contents

1 Introduction

1.1 Foreword

In this manual I will explain and guide you through the process of setting up a WordPress hosting on AWS. We will be using AWS FarGate to define our services, create an ECS cluster and assign tasks to it.

For reference files please refer to the GitHub.
For further help please refer to the AWS Documentation

1.2 AWS CLI

In order to follow along make sure to install AWS CLI Setup Guide. For reference, we installed the CLI and did all configuration and hosting using Arch Linux. However, this should work on any UNIX command based system.

2 Startup Environment Variables

After setting up AWS CLI we will need some environment variables first. I have created sample .env files for this be sure to find these on the GitHub.

Also, I have provided a script called *load_env.sh* which will try to load the environment files automatically. All you have to do is make sure the files are in the same directory and the .env file has the correct variables.

Listing 1: Basic environment variables

```
#!/bin/bash
export WOF_AWS_REGION=eu-west-2
export WOF_ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output
    text)
export WOF_ECS_CLUSTER_NAME=ecs-vpsa-wordpress
export WOF_CFN_STACK_NAME=VPSA-WordPress-FarGate
export WOF_ECS_SERVICE_NAME=vpsa-efs-rw-service
export DOCKER_CONTAINER_NAME=wordpressVPSA
```

Now let's go over each one

- WOF_AWS_REGION, is our CloudFormation region, usable regions can be found online.
- WOF_ACCOUNT_ID, is the account ID for our AWS manager. You can just use the code provided to automatically set this.
- WOF_ECS_CLUSTER_NAME, is the name of our ECS Cluster.
- WOF_CFN_STACK_NAME, is the CloudFormation stack name.
- WOF_ECS_SERVICE_NAME, is the name of our ECS service.
- DOCNER_CONTAINER_NAME, is the name of our docker container, this needs to match the one in task-definition as well.

Use the following command to populate the environment variables from the .env file.

Listing 2: Loading env variables

```
#!/bin/bash
. ./load_env.sh
```

We have some more environment variables, but those will come after setting up our CloudFormation.

3 CloudFormation Stack

Our CloudFormation Stack basically tells AWS what services we want to include, we will use this to create all of our services.

We use a YAML file to describe our services and use this as a template for AWS. To create and serve our services. Our YAML for VPSA includes the following:

- VPC (Virtual Private Cloud) for use between services
- RDS (Relational Database) mysql database
- EFS (Elastic File System) this will store all our data and is flexible in storage space
- ALB (Load Balancer) this will manage our ECS containers and assign traffic accordingly
- Managed NAT instances
- RDS MySQL Database

Note: our CloudFormation stack does not include our ECS services or tasks

3.1 Creation of the CloudFormation stack

Let's start creating our stack. To do this we will use our YAML template. This template can be found on GitHub.

Note: since this document is just for setting up the hosting, no details about the template's content will be discussed

Before we can continue, make sure the environment variables are properly loaded. Otherwise, the following command will fail.

Listing 3: Creating the Stack

```
#!/bin/bash
aws cloudformation create-stack \
--stack-name $WOF_CFN_STACK_NAME \
--region $WOF_AWS_REGION \
--template-body file://wordpress-ecs-fargate.yaml
```

After running the previous command, our stack will be created. Now we have to wait before moving on. We can check the status of the stack on the CloudFormation dashboard on the AWS Dashboard Console

Alternatively you can use the following command.

Listing 4: Waiting for the Stack

```
#!/bin/bash
aws cloudformation wait stack-create-complete \
--stack-name $(aws cloudformation describe-stacks \
--region $WOF_AWS_REGION \
--stack-name $WOF_CFN_STACK_NAME \
--query Stacks[0].StackId --output text) \
--region $WOF_AWS_REGION
```

3.2 Setting more Environment Variables

After the stack has been created we can now use the CloudFormation instance to set some more environment variables. These variables are used in creating and assigning of the ECS cluster and setting up the load balancer.

To make this process as worry free as possible, I have created a second script that will set these environment variables. Launch the load_env_after.sh script next. This script will use the .after.env file to set the required environment variables. Unless the template file has changed, this file should be sufficient to use.

Since these environment variables are fetched from our CloudFormation stack, what they do isn't that important for now. All you need to know is that they pull in information from our created services. Such as our EFS, ALB, etc.

Listing 5: Loading env variables

```
#!/bin/bash
. ./load_env_after.sh
```

Note: if the template file has changed, be sure to check the .after.env file thoroughly

4 ECS Cluster

4.1 Task definition

Now let's move on to our ECS cluster. In order to create this we will be using a *task definition*. Task definitions tell AWS how our container should be created based on a docker image. We will be using the wp-task-definition.json. This file is also found on the GitHub.

Note: logging is defined in the task-definition. Make sure to change this if needed, more info can be found at Logging

In order to set the correct logging permission make sure to edit the template file and change the "executionRoleArn" to the arn of the IAM role you created (or an existing one).

For our WordPress instance we will use a Docker image by bitnami. Our data will be stored under */bitnami/wordpress*. Next up we will have to register our task definition by running the following command.

Note: Before registering the task, we need to change some things in our task-definition.

We first need to change the *fileSystemId* in the volume section of the task definition. This can be found on the EFS dashboard.
Next we also need to set our *accessPointId* in the volume section. These can also be found on the EFS Dashboard.

Listing 6: Registering task definition

```
#!/bin/bash
export WOF_TASK_DEFINITION_ARN=$(aws ecs register-task-definition \
--cli-input-json file://wp-task-definition.json \
--region $WOF_AWS_REGION \
--query taskDefinition.taskDefinitionArn --output text)
```

4.2 Creating the cluster

We are finally ready to launch our ECS cluster. We can use the following command to do so.

Listing 7: Cluster creation

```
#!/bin/bash
aws ecs create-cluster \
--cluster-name $WOF_ECS_CLUSTER_NAME \
--region $WOF_AWS_REGION
```

Note: The WOF_ECS_CLUSTER_NAME is the one we specified at the start when setting up our environment variables

4.3 ALB security group

Now we need to configure our load balancer security group. We first need to set another environment variable, do that by executing the following command.

Listing 8: Security group ID

```
#!/bin/bash
export WOF_SVC_SG_ID=$(aws ec2 create-security-group \
--description Svc-WordPress-on-Fargate \
--group-name Svc-WordPress-on-Fargate \
--vpc-id $WOF_VPC_ID --region $WOF_AWS_REGION \
--query 'GroupId' --output text)
```

That environment variable will be our security group ID. Now we can use that to allow traffic on port 8080 (WordPress port defined in our task definition).

Listing 9: ALB ingress 8080

```
#!/bin/bash
aws ec2 authorize-security-group-ingress \
--group-id $WOF_SVC_SG_ID --protocol tcp \
--port 8080 --source-group $WOF_ALB_SG_ID \
--region $WOF_AWS_REGION
```

4.4 ECS Service

Finally, we can configure our ECS service. This service will tell AWS to launch our task definition on our ECS. During this we will also enable our load balancer.

Listing 10: Create ECS service

```
#!/bin/bash
aws ecs create-service \
--cluster $WOF_ECS_CLUSTER_NAME \
--service-name $WOF_ECS_SERVICE_NAME \
--task-definition "${WOF_TASK_DEFINITION_ARN}" \
--load-balancers targetGroupArn="${WOF_TG_ARN}",containerName="${CONTAINER_NAME}"
    ,containerPort=8080 \
--desired-count 2 \
--platform-version 1.4.0 \
--launch-type FARGATE \
--deployment-configuration maximumPercent=100,minimumHealthyPercent=0 \
--network-configuration "awsvpcConfiguration={subnets=["$WOF_PRIVATE_SUBNET0,
    $WOF_PRIVATE_SUBNET1"],securityGroups=["$WOF_SVC_SG_ID"],assignPublicIp=
    DISABLED}" \
--region $WOF_AWS_REGION
```

Note: desired-count changes how many containers we wish to run at once. Our load balancer will automatically assign traffic to any container. If we want more running at once we can change it here. Alternatively it's possible to configure our load balancer to launch extra instances when needed.

Before proceeding we need to wait for these services to run. We can use our ECS dashboard to check that ECS Dashboard. Alternatively we can use the following command, this will display the number of running tasks.

Listing 11: Wait for tasks to run

```
#!/bin/bash
watch aws ecs describe-services \
--services $WOF_ECS_SERVICE_NAME \
--cluster $WOF_ECS_CLUSTER_NAME \
--region $WOF_AWS_REGION \
--query 'services[].runningCount'
```

When both are running we can get our load balancer URL by using the following command.

Listing 12: Load balancer URL

```
#!/bin/bash
echo "http://$(aws elbv2 describe-load-balancers \
--names wof-load-balancer --region $WOF_AWS_REGION \
--query 'LoadBalancers[].DNSName' --output text)/wp-admin/"
```

5 Container Management

5.1 Terminating

If you need to terminate the containers you can do so with the following command.

Listing 13: Terminate container

```
#!/bin/bash
aws ecs update-service \
--cluster $WOF_ECS_CLUSTER_NAME \
--region $WOF_AWS_REGION \
--service $WOF_ECS_SERVICE_NAME \
--task-definition "$WOF_TASK_DEFINITION_ARN" \
--desired-count 0
```

5.2 Listing replicas

It can take a few minutes before the containers are terminated. Run to following command or use the ECS Dashboard to follow up the progress.

Listing 14: Get replica count

```
#!/bin/bash
watch aws ecs describe-services \
--services $WOF_ECS_SERVICE_NAME \
--cluster $WOF_ECS_CLUSTER_NAME \
--region $WOF_AWS_REGION \
--query 'services[].runningCount'
```

5.3 Restarting

In order to restart the tasks, run the following command

Listing 15: Restart task

```
#!/bin/bash
aws ecs update-service \
--cluster $WOF_ECS_CLUSTER_NAME \
--region $WOF_AWS_REGION \
--service $WOF_ECS_SERVICE_NAME \
--task-definition "$WOF_TASK_DEFINITION_ARN" \
--desired-count 2
```

Note: desired-count can be specified when performing this.

6 ECS Service Auto Scaling

6.1 Auto-Scaling service

In order to make this a more robust solution we will also implement auto-scaling. Auto-scaling will spin up another task if the resources are being overwhelmed.

Note: in order for auto-scaling to work we need to create an IAM role. Alternatively you can use an existing role if it exists. IAM role for auto-scaling.

The following command will register application auto-scaling for our ECS cluster.

Listing 16: Application auto-scaling

```
#!/bin/bash
aws application-autoscaling \
register-scalable-target \
--region $WOF_AWS_REGION \
--service-namespace ecs \
--resource-id service/${WOF_ECS_CLUSTER_NAME}/${WOF_ECS_SERVICE_NAME} \
--scalable-dimension ecs:service:DesiredCount \
--min-capacity 2 \
--max-capacity 4
```

6.2 Auto-Scaling policy

Before we can properly use auto-scaling we need to set an auto-scaling policy. To do that we can use the "scaling.config.json" file provided on the GitHub. In our config we will set auto-scaling to kick in when the CPU usage exceeds 75%.

Next we will run the following command to set our scaling policy

Listing 17: Scaling policy config

```
#!/bin/bash
aws application-autoscaling put-scaling-policy \
--service-namespace ecs \
--scalable-dimension ecs:service:DesiredCount \
--resource-id service/${WOF_ECS_CLUSTER_NAME}/${WOF_ECS_SERVICE_NAME} \
--policy-name cpu75-target-tracking-scaling-policy \
--policy-type TargetTrackingScaling \
--region $WOF_AWS_REGION \
--target-tracking-scaling-policy-configuration file://scaling.config.json
```

Note: make sure the scaling.config.json file has the same name, change it otherwise (command or file itself)
To test the scaling policy we could use a tool such as hey. The following command would then generate a load.

Listing 18: Testing scaling

```
#!/bin/bash
```

```
./hey_linux_amd64 -z 20m <WordPress URL>
```

7 Logging

Logging is an important part of configuring AWS clusters and services. We want to see all logs of our ECS tasks. In order to do this we define a *logConfiguration* section in our *task-definition.json*. This is located at the top of the provided GitHub file. Keep the logDriver the same and just change the group. The group should be different for each task-definition file. This group also needs to be created and given the correct permission first. More info about this can be found at logging groups. When all of this has been set logs can be viewed on the ECS Dashboard for the specific cluster or on the CloudWatch Dashboard.

Note: make sure that when creating the logging group, you select the correct AWS service. For this tutorial you should select ECS.

8 Database Access

If you need to access the database, you can do so with a tool such as MySQL Workbench or by using the mysql cli. Before you can access the database using these methods, we need to make sure our inbound connection rules are set properly. The easiest way to do this is by logging in to our security group dashboard. Here we will select our database security group. Then we can edit the inbound rules to allow traffic from all IPs (0.0.0.0) or any IP you so wish.

Note: you might have to enable "Publicly accessible" on the database if it's turned off. You can do this on the database dashboard.

Note: for security reasons you should make sure to remove the extra inbound rule when it's not in use

9 SFTP Transfer

Now that we have everything set up, we can start by setting up SFTP to transfer files. Setting up SFTP on AWS is made easy by using the "Transfer Family" service. Transfer Family will allow us to use SFTP, FTP or FTPS on our EFS. This service does cost money and charges at the time of writing $0.30 per hour. A small fee per GB transferred is also present. Disabling this service does not stop this cost. In order to stop this service from taking up your budget, you need to fully remove it. When SFTP is not needed to be sure to remove this.

To set this up we will go to the "Transfer Family" on AWS and create a new Transfer Family. When creating the Transfer Family make sure to choose the correct role. The Transfer Family needs the correct permissions for setting up transfer logs and accessing the EFS. Make sure to create and assign a user. When creating a user we need to specify our SSH key. We can generate an SSH key combo using the following command.

Listing 19: SSH Keygen

```
#!/bin/bash
ssh-keygen -t rsa -b 4096 -f key_name
```

After generating this key we can copy the .pub and paste that text into the key section of our user. We also need to make sure to select the correct file system to allow this user access. If the EFS doesn't show up make sure the region is set correctly.

Note: be certain to select the correct region and path for the user' EFS (default this is filled in, but the path might not exist).

10 Performing EFS Commands

After we have transferred our files using SFTP we run in into the following problem. We need a way to extract the Tar file we just uploaded.

How do we do this? Well since we are using AWS FarGate there is no easy way to gain SSH access to our containers without a lot of extra steps. You might think we can just write a script that will extract the tar file and call that script using a custom Dockerfile. The issue with that is automatic health checks will be run and if our containers are not reachable the service will restart. This means that as long as we are extracting we are not reachable. Our service will automatically restart and interrupt the un-tarring process before it has to start again.

The easiest way to solve this is by using an elastic compute cloud (EC2). These can be reached by SSH, and thus we can perform commands. We create an EC2

container through the AWS EC2 dashboard. From there we select a security group. Choose a system image (I used Amazon Linux 2). We also need to make sure to select our EFS. Don't forget to create an SSH key pair that we can use for connecting. I went for the .pem method since I am not using PuTTY.

After acquiring the .pem file we can use this to connect to our EC2 over SSH.

Listing 20: EC2 SSH

```
#!/bin/bash
# Modify the file permissions of the .pem
chmod 400 key_file.pem

# Connect to the EC2 container (user and or region might be different)
ssh -i "key_file.pem" root@xxxxxxx.eu-west-2.compute.amazonaws.com
```

Note: in the security group an inbound rule has to be set to allow port 22 (SSH port) to be let through.

11 Reset

Made a mistake or want to remove all the hard work we have done? The following commands will remove all the resources we made.

Note: this doesn't remove the transfer family.

WARNING: running this command will have drastic consequences, be prepared.

Listing 21: Reset

```
#!/bin/bash
aws application-autoscaling delete-scaling-policy --policy-name cpu75-target-
    tracking-scaling-policy --service-namespace ecs --resource-id service/${
    WOF_ECS_CLUSTER_NAME}/${WOF_ECS_SERVICE_NAME} --scalable-dimension ecs:
    service:DesiredCount --region $WOF_AWS_REGION
aws application-autoscaling deregister-scalable-target --service-namespace ecs --
    resource-id service/${WOF_ECS_CLUSTER_NAME}/${WOF_ECS_SERVICE_NAME} --
    scalable-dimension ecs:service:DesiredCount --region $WOF_AWS_REGION
aws ecs delete-service --service $WOF_ECS_SERVICE_NAME --cluster
    $WOF_ECS_CLUSTER_NAME --region $WOF_AWS_REGION --force
aws ec2 revoke-security-group-ingress --group-id $WOF_SVC_SG_ID --region
    $WOF_AWS_REGION --protocol tcp --port 8080 --source-group $WOF_ALB_SG_ID
aws ec2 delete-security-group --group-id $WOF_SVC_SG_ID --region $WOF_AWS_REGION
aws ecs delete-cluster --cluster $WOF_ECS_CLUSTER_NAME --region $WOF_AWS_REGION
aws cloudformation delete-stack --stack-name $WOF_CFN_STACK_NAME \--region
    $WOF_AWS_REGION
```