Containers

Running WordPress on Amazon ECS on AWS Fargate with Amazon EFS

by Re Alvarez-Parmar and Jimmy Hayes | on 20 MAY 2021 | in Amazon Elastic Container Service, Amazon Elastic File System (EFS), Containers | Permalink | Share

I built my first website back in 1997. It was a fan site for my then favorite musician. I didn't know much about creating websites, but I had a burning desire to tell the *World Wide Web* (as if anyone was listening) about my musical preferences. The *floppy-disk-booted-PCs* in my school's computer lab ran MS-DOS, and the lab teacher was only trained in Basic, so most of my "web development" knowledge came from finding *cool* websites and, ahem, shamelessly copying their code (special thanks to whoever thought of the "view source" button). I proceeded to learn the absolute minimum amount of HTML I needed to create a web page, but within few hours, it became painfully apparent that with my limited experience, I am not going to be able to build anything worthwhile. Internet gurus told me that a website without CSS, PHP, JavaScript, Java applets, and not to mention a sleek Macromedia Shockwave animation is going to be a joke. So while I did, at the end of the day, create a site, it took way too long, and the results weren't polished enough for my preference.

Thankfully, a lot has changed in the two decades. Today, anyone can create a professional-looking website with a few clicks. Content Management Systems (CMS) like WordPress have made it possible to build sites, blogs, forums, and more with minimal technical experience. One doesn't need to possess programming skills to create a website. Unsurprisingly, w3ctechs claims that 40% of web uses WordPress, and it powers some of the most trafficked websites.

It's not uncommon for large institutions to run hundreds of WordPress sites. For example, an insurance firm may create a dedicated site for each plan they offer. A brokerage advisor may create brochure-ware sites for each investment option they sell. Companies use WordPress to quickly create sites that can also scale horizontally to handle millions of requests.

What makes it challenging to scale WordPress sites in containerized environments is that it needs a performant and durable shared storage layer. In the past, customers had to rely on network file-sharing or a file replication service to run WordPress replicas across servers. That often led to other problems: often, the file share itself becomes a single point of failure or a bottleneck due to its inability to scale. So when we spoke with AWS customers about adopting containers, they frequently asked us how Amazon Elastic Container Service (Amazon ECS) can help them run their WordPress sites at scale.

Integration with Amazon EFS

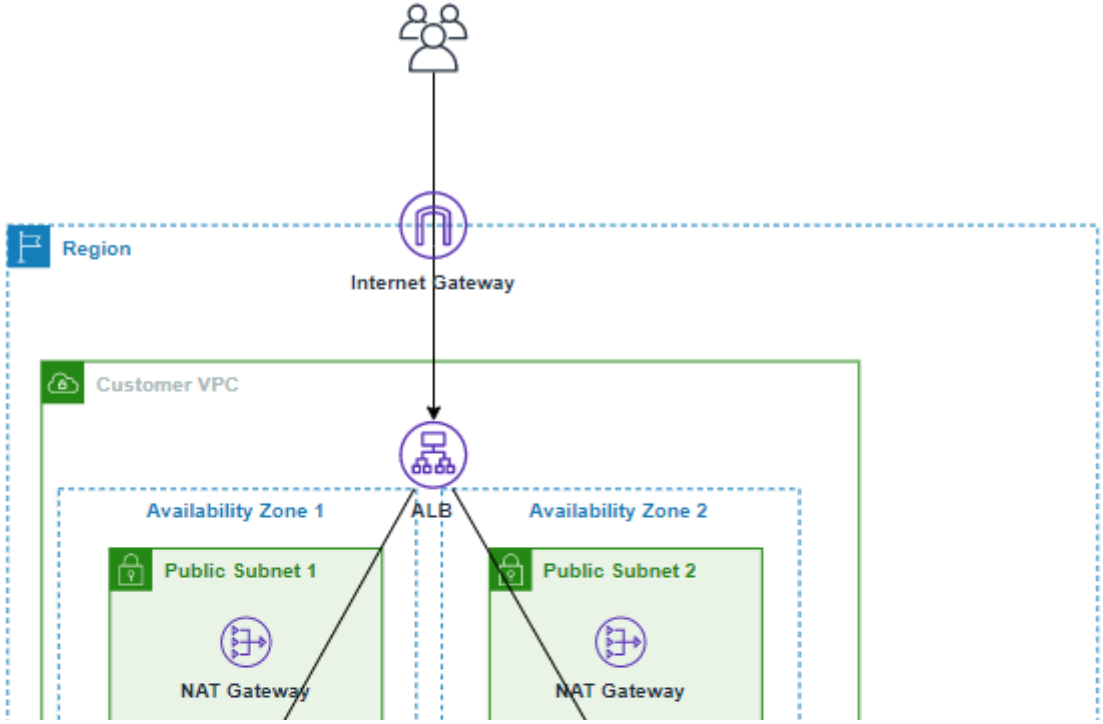
It wasn't until Amazon ECS added support for Amazon Elastic File System (Amazon EFS) that it became feasible to run WordPress in containers at scale. EFS provides massively parallel shared access that automatically grows and shrinks as files are added and removed. Thousands of ECS tasks can concurrently perform read and write operations on shared EFS file systems.

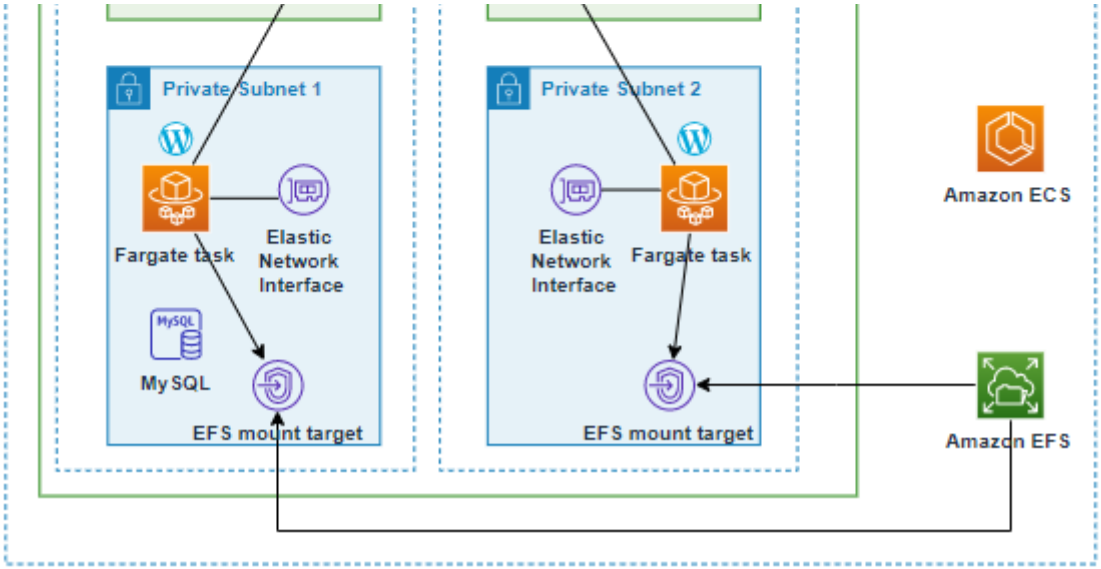
As EFS provided a solution for the shared storage problem, many customers began using ECS to orchestrate WordPress workloads. When we meet with customers now, they often ask us how they can further reduce their operational responsibilities by using AWS Fargate.

AWS Fargate is a serverless compute engine for containers that works with both Amazon ECS and Amazon Elastic Kubernetes Service (EKS). It removes the need to provision and manage servers, lets you specify and pay for resources per application, and improves security through application isolation by design. It allocates the right amount of compute, eliminating the need to choose instances and scale cluster capacity. You only pay for the resources required to run your containers, so there is no over-provisioning and paying for additional servers. Fargate runs each task or pod in its own kernel providing the tasks and pods their own isolated compute environment. This enables your application to have workload isolation and improved security by design. These are the reasons why AWS customers like Vanguard, Accenture, Foursquare, and Ancestry have chosen to run their mission-critical applications on Fargate.

Solution

This post shows you how to run WordPress without servers using Amazon ECS on AWS Fargate. We will create a VPC and an ECS cluster in which our WordPress tasks will run. An RDS MySQL instance will provide the database that WordPress requires. WordPress will persist its data on a shared Amazon EFS filesystem.





Prerequisites

You will need AWS CLI version 2 to complete the tutorial. We have tested the steps on Amazon Linux 2.

Let's start by setting a few environment variables:

```
Bash
export WOF_AWS_REGION=us-west-2 <-- Change this to match your region
export WOF_ACCOUNT_ID=$(aws sts get-caller-identity --query 'Account' --output text)
export WOF_ECS_CLUSTER_NAME=ecs-fargate-wordpressexport WOF_CFN_STACK_NAME=WordPress-on-Fargate
```

Create base infrastructure

Next, we will create a VPC with two public and private subnets across two Availability Zones (AZ) and a NAT Gateway in each AZ. We have created a CloudFormation template that provisions the VPC resources, an RDS database, an EFS filesystem, an Application Load Balancer (ALB), a listener, target groups, and associated security groups.

Create a stack using CloudFormation:

Bash
```
wget https://raw.githubusercontent.com/aws-samples/containers-blog-maelstrom/main/CloudFormation/wordpress-ecs-fargate.y

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

The stack deployment takes roughly five minutes to complete.

The following commands waits until the stack deployment completes. You can use this command to determine when the stack deployment is complete or you can use the AWS Management Console to view deployment progress:

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
```

Load environment variables from the CloudFormation stack's output:

Bash
```
export WOF_EFS_FS_ID=$(aws cloudformation describe-stacks  \
  --region $WOF_AWS_REGION \
  --stack-name $WOF_CFN_STACK_NAME \
  --query "Stacks[0].Outputs[?OutputKey=='EFSFSId'].OutputValue" \
  --output text)

export WOF_EFS_AP=$(aws cloudformation describe-stacks  \
```

```
  --region $WOF_AWS_REGION \
  --stack-name $WOF_CFN_STACK_NAME \
  --query "Stacks[0].Outputs[?OutputKey=='EFSAccessPoint'].OutputValue" \
  --output text)

export WOF_RDS_ENDPOINT=$(aws cloudformation describe-stacks  \
  --region $WOF_AWS_REGION \
  --stack-name $WOF_CFN_STACK_NAME \
  --query "Stacks[0].Outputs[?OutputKey=='RDSEndpointAddress'].OutputValue" \
  --output text)

 export WOF_VPC_ID=$(aws cloudformation describe-stacks  \
```

The stack creates an EFS filesystem with mount targets in two AZs. WordPress tasks in each AZ will mount the EFS file system using the local EFS mount target in that AZ.

Note that the RDS MySQL instance this stack creates is a single point of failure. In production environments, we recommend improving the resilience of the WordPress database by using RDS Multi-AZ Deployments. You can also use Amazon Aurora Serverless instead of RDS MySQL, which automatically starts up, shuts down, and scales database capacity based on your application's needs. It allows you to run your database without managing servers.

You can further boost the performance of your WordPress sites by using a caching layer (like ElastiCache for Memcached) and a content delivery network like Amazon CloudFront. Follow this guide to learn about speeding up WordPress with Amazon ElastiCache for Memcached.

Create task definition

Now that we have created the network, database, and shared storage, we are ready to deploy WordPress. We'll use the bitnami/wordpress container image to create tasks. The task definition includes database credentials. Please change the database password in real-world scenarios.

Let's create a task definition:

JSON

```
cat > wp-task-definition.json << EOF
```

```
{   "networkMode": "awsvpc",
    "containerDefinitions": [
        {

            "portMappings": [
                {
                    "containerPort": 8080,
                    "protocol": "tcp"
                }
            ],
            "essential": true,
            "mountPoints": [
                {
                    "containerPath": "/bitnami/wordpress",
                    "sourceVolume": "wordpress"
                }
            ],
```

The WordPress container image is configured to store data at `/bitnami/wordpress`. As evident in the task definition above, the container mounts the EFS file system at `/bitnami/wordpress` on the container's file system.

We also use an access point to access the EFS file system. EFS access points are application-specific entry points into an EFS file system that make it easier to manage application access to shared datasets. Access points can enforce a user identity, including the user's POSIX groups, for all file system requests that are made through the access point. They can also enforce a different root directory for the file system so that clients can only access data in the specified directory or its sub-directories. To further understand the EFS security model and how it works with containers, please read Massimo Re Ferre's developers guide to using Amazon EFS with Amazon ECS and AWS Fargate – Part 2.

When running multiple WordPress installations, you can use a single EFS file system to persist data for multiple sites and isolate data by using an access point for each site.

Register the task definition:

Bash
```
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)
```

Run WordPress

In this section, we'll create an ECS cluster to run WordPress. We'll create an ECS service that maintains two WordPress replicas for high availability. It also integrates with the ALB and automatically updates ALB targets as WordPress tasks are created and destroyed.

Create an ECS cluster:

Bash
```
aws ecs create-cluster \
  --cluster-name $WOF_ECS_CLUSTER_NAME \
  --region $WOF_AWS_REGION
```

Create a security group that accepts traffic on port 8080 from the ALB's security group:

Bash
```
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)

##Accept traffic on port 8080
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
```

Create an ECS service:

Bash

```
aws ecs create-service \
  --cluster $WOF_ECS_CLUSTER_NAME \
  --service-name wof-efs-rw-service \
  --task-definition "${WOF_TASK_DEFINITION_ARN}" \
  --load-balancers targetGroupArn="${WOF_TG_ARN}",containerName=wordpress,containerPort=8080 \
  --desired-count 2 \
  --platform-version 1.4.0 \
  --launch-type FARGATE \
  --deployment-configuration maximumPercent=100,minimumHealthyPercent=0 \
  --network-configuration "awsvpcConfiguration={subnets=["$WOF_PRIVATE_SUBNET0,$WOF_PRIVATE_SUBNET1"],securityGroups=["$
  --region $WOF_AWS_REGION

#Wait until there two running tasks
watch aws ecs describe-services \
  --services wof-efs-rw-service \
  --cluster $WOF_ECS_CLUSTER_NAME \
  --region $WOF_AWS_REGION \
  --query 'services[].runningCount'
```

Once two tasks are running you can proceed to the next steps.

Access WordPress admin dashboard

Once the Fargate tasks are running, log into the WordPress admin dashboard. Obtain the dashboard address:

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

The admin username is '*user*' and the password is '*bitnami*'. Please change the password as soon as possible.

Test data persistence

Now that the WordPress installation is complete, it's time to test data persistence. While in your WordPress admin dashboard, make a site configuration change like activating a new site theme. Once you've made the change, terminate all WordPress containers:

```
Bash
aws ecs update-service \
  --cluster $WOF_ECS_CLUSTER_NAME \
  --region $WOF_AWS_REGION \
  --service wof-efs-rw-service \
  --task-definition "$WOF_TASK_DEFINITION_ARN" \
  --desired-count 0
```

Wait until there are no active replicas. You can use watch to get the replica count:

```
Bash
watch aws ecs describe-services \
  --services wof-efs-rw-service \
  --cluster $WOF_ECS_CLUSTER_NAME \
  --region $WOF_AWS_REGION \
  --query 'services[].runningCount'
```

Once all tasks have been terminated, scale the service back to a minimum of three replicas:

Bash

```
aws ecs update-service \
  --cluster $WOF_ECS_CLUSTER_NAME \
  --region $WOF_AWS_REGION \
  --service wof-efs-rw-service \
  --task-definition "$WOF_TASK_DEFINITION_ARN" \
  --desired-count 2
```

When the tasks are running, go to the WordPress admin dashboard, and verify that your changes have persisted.

Service Auto Scaling

Analyze the traffic on any popular website, and you'd notice that it fluctuates frequently. Sometimes the traffic is predictable because it follows a cyclical pattern, but many times it is highly unpredictable and subject to many external factors such as marketing campaigns and other business processes. In the past, teams had to predict the compute capacity they needed and provision infrastructure to meet peak demands. In the cloud, applications and infrastructure are expected to scale based on business needs. ECS Service Auto Scaling helps you maintain the level of performance for your application as its load waxes and wanes.

Here's an example that demonstrates setting Service Auto Scaling for WordPress. First, we'll register the WordPress ECS service with Application Auto Scaling, then we'll create a policy that automatically adjusts the task replica count based on the service's average CPU utilization.

Register the ECS service with Application Auto Scaling:

Bash

```
aws application-autoscaling \
  register-scalable-target \
  --region $WOF_AWS_REGION \
  --service-namespace ecs \
  --resource-id service/${WOF_ECS_CLUSTER_NAME}/wof-efs-rw-service \
  --scalable-dimension ecs:service:DesiredCount \
  --min-capacity 2 \
  --max-capacity 4
```

> If you've never used ECS Service Auto Scaling, you'll also create an IAM role for Service Auto Scaling.

Create a Service Auto Scaling policy document:

JSON
```
cat > scaling.config.json << EOF
{
     "TargetValue": 75.0,
     "PredefinedMetricSpecification": {
         "PredefinedMetricType": "ECSServiceAverageCPUUtilization"
     },
     "ScaleOutCooldown": 60,
    "ScaleInCooldown": 60
}
EOF
```

Configure Application Auto Scaling policy:

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

When the WordPress service's average CPU utilization rises above 75%, a scale-out alarm triggers Service Auto Scaling to add another task to the WordPress service to decrease the load on the running tasks and ensure that users don't experience a service disruption. Conversely, when the average CPU utilization dips below 75%, a scale-in alarm triggers a decrease in the service's desired count.

Now, you can use a utility like hey to generate load, and test the auto scaling configuration:

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

With automatic scaling in place, ECS will automatically add or remove tasks based on the CPU usage. An analysis of your site's traffic pattern will help you discern the minimum number of tasks your site needs for baseline performance, and ECS will respond to traffic spikes by adding tasks. It will also remove (scale-in) tasks when they are no longer needed, which will help you reduce infrastructure expenditure.

Fargate Spot

Speaking of infrastructure costs, applications like WordPress are ideal candidates for Fargate Spot. WordPress running in an ECS cluster can tolerate Spot interruptions without causing a service disruption. This is because the ECS service maintains the task replica count. Should a task get terminated by Spot, ECS will create a replacement task. You can use an ECS capacity provider to mix Spot with On-Demand capacity to ensure your application is not offline if Spot capacity is temporarily unavailable. For more information about Fargate Spot, please read this post.

And since we're on the topic, I'd like to remind that you can further reduce your spend on Fargate and EC2 by getting a Compute Savings plan.

Cleanup

Use the following commands to delete resources created during this post:

```
Bash
aws application-autoscaling delete-scaling-policy --policy-name cpu75-target-tracking-scaling-policy --service-namespace
aws application-autoscaling deregister-scalable-target --service-namespace ecs --resource-id service/${WOF_ECS_CLUSTER_N
aws ecs delete-service --service wof-efs-rw-service --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 --so
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
```

Conclusion

Thanks to the technological advances, building and operating scalable websites is not as jarring experience as it was two decades ago for me. Anyone can create and publish content using Content Management Systems like WordPress. This post showed how Amazon ECS on AWS Fargate helps you operate a scalable WordPress installation without taking on server management responsibilities. Amazon ECS integrates with Amazon EFS to provide a performant layer that allows your websites to scale automatically, saving you money during quiet hours, and ensuring consistent performance during peaks.