

Creating a CI/CD enabled web application using AWS ECS Fargate and Code Pipeline

Jason Hoog

August 2022

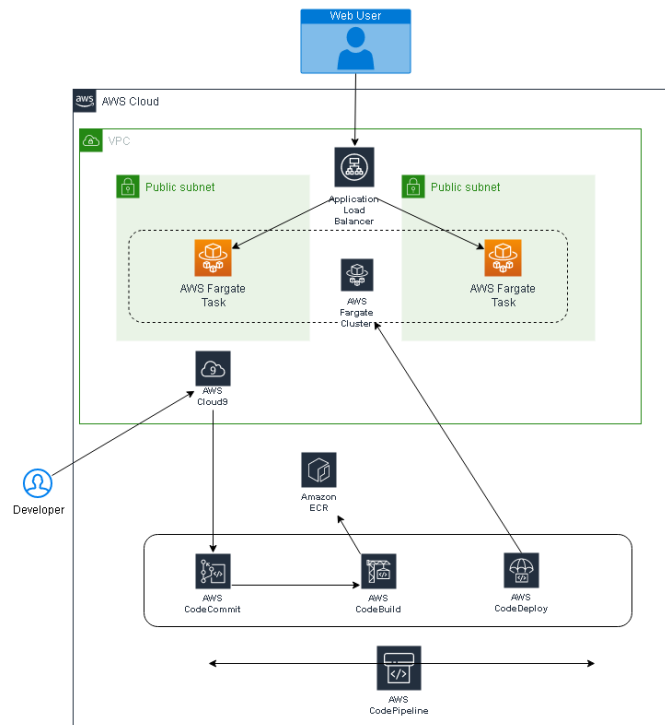
Contents

| | |
|--|-------------------------------------|
| Elastic Container Service (Fargate) and Code Pipeline | 3 |
| Configure and deploy a web app on ECS Fargate | 3 |
| Download the sample application | Error! Bookmark not defined. |
| Create the Docker file | 4 |
| Build, Run and Test the docker container..... | 4 |
| Push your docker container to Amazon ECR | 6 |
| Deploying our image to ECS..... | 7 |
| Deploying our container application changes using CI/CD..... | 19 |
| Create a CodeCommit repository | 19 |
| Create a CodePipeline..... | 20 |
| Cleanup | 28 |

Overview

This application walks you through creating a very simple PHP web application using the following AWS services:

- AWS Cloud9 – Development Environment
- AWS ECR – Container Registry
- AWS ECS Fargate – serverless container compute service, integrated with EC2 Load Balancing
- AWS CodePipeline – CI/CD orchestration with CodeCommit, CodeBuild and CodeDeploy



Creating and Deploying the App

This takes you through some quick examples and configuration for setting up an application lifecycle using the aforementioned AWS Services.

Configure and deploy a web app on ECS Fargate

For this example, we are going to use a very simple sample PHP web application. You can download the source code for this example from the following link:

<https://github.com/jahoog/webphp-sample/blob/e53831c68346e8fc99a4f4a22c1d1e82ccf0dc96/archive/webphpSimple.zip?raw=true>

Use Cloud9 to Create a Docker Container

The first thing we need to do is create a Docker Container. You can do this on your local workstation, but for this demo we are going to use a Cloud9 instance and create a docker image from there.

To setup a Cloud9 environment, you can follow the instructions here:

<https://docs.aws.amazon.com/cloud9/latest/user-guide/create-environment.html>

Once we login to our Cloud9 environment, we can use the terminal to download our sample php application and create a docker image. Use the following commands to do this.

```
mkdir webphp
cd webphp
curl -L -o webphp.zip https://github.com/jahoog/webphp-sample/blob/e53831c68346e8fc99a4f4a22c1d1e82ccf0dc96/archive/webphpSimple.zip?raw=true
unzip webphp.zip
rm webphp.zip
```

```
hoog:~/environment/webphp $ curl -L -o webphp.zip https://github.com/jahoog/webp
96/archive/webphpSimple.zip?raw=true
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
  0     0     0     0     0     0      0      0  --:--:-- --:--:-- --:--:--    0
  0     0     0     0     0     0      0      0  --:--:-- --:--:-- --:--:--    0
100  844 100  844     0     0 22546     0  --:--:-- --:--:-- --:--:-- 22546
hoog:~/environment/webphp $ unzip webphp.zip
Archive:  webphp.zip
  inflating: index.php
  inflating: styles/style.css
hoog:~/environment/webphp $ rm webphp.zip
```

Create the Docker file

A Docker file is a text document that contains the specs for the Docker image build. For more information you can view the Dockerfile reference page:

<https://docs.docker.com/engine/reference/builder/>

In this case, we are going to simply create a new Dockerfile:

```
touch Dockerfile
```

Then we will open the Dockerfile in our editor and enter the following into the Dockerfile and save the file:

```
FROM webdevops/php-apache-dev:latest
COPY . /app/
```

The above Dockerfile defines the layers within our docker image. We are going to have a simple 2 line file that does this:

Line 1: use a public php/apache based image as the base

Line 2: copies our application we unzipped into the /app/ folder of that image.

Build, Run and Test the docker container

Use the following commands to build, run and test the docker container:

```
docker build -t webphp .
docker images
```

Validate that the image was created:

```
[ec2-user@ip-172-31-9-18 webphp]$ docker images
```

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|------------|--------|--------------|----------------|-------|
| webphp | latest | 6478cfd2d763 | 24 minutes ago | 563MB |

Now let's run the container:

```
docker run -d -p 80:80 webphp
```

This runs the docker container in “detached” mode (specified by the `-d` option) and exposes the container's port 80 on the host's port 80 (specified by the `-p` option). We can view the running container with the following command:

```
docker ps
```

```
[ec2-user@ip-172-31-9-18 webphp]$ docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS |
|--------------|--------------------|--------------------------|----------------|--------------|
| f89441ae0456 | webphp | "/entrypoint supervi..." | 3 minutes ago | Up 3 minutes |
| 443/tcp | 0.0.0.0:80->80/tcp | 9000/tcp | kind_albattani | |

We can now test the application locally using curl:

```
curl http://localhost
```

```
[ec2-user@ip-172-31-9-18 webphp]$ curl http://localhost
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My test page</title>
    <link href='http://fonts.googleapis.com/css?family=Open+Sans' rel='styl
  >
    <link href="styles/style.css" rel="stylesheet" type="text/css">
  </head>
  <body>
    <h1>My PHP Website #1</h1>
    

    <p>We are live!</p>

    <ul> <!-- changed to list in the tutorial -->
      <li>Time: Today is 2018/06/28</li>
    </ul>

  </body>
```

If you would like to stop your application, simply issue this command:

```
docker stop <container id or name>
```

```

[ec2-user@ip-172-31-9-18 webphp]$ docker ps
CONTAINER ID        IMAGE               COMMAND             NAMES
f89441ae0456       webphp             "/entrypoint superi... 1
443/tcp, 0.0.0.0:80->80/tcp, 9000/tcp  kind_albattani
[ec2-user@ip-172-31-9-18 webphp]$ docker stop kind_albattani
kind_albattani

```

Push your docker container to Amazon ECR

Amazon Elastic Container Registry gives you a private repository for your Docker containers. To push your container, you can follow the steps at

<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html#use-ecr>

Note: for the following steps, you will need an IAM user that has access to push repositories to ECR (see <https://docs.aws.amazon.com/codedeploy/latest/userguide/getting-started-create-iam-instance-profile.html>)

```
aws ecr create-repository --repository-name webphp
```

```

[ec2-user@ip-172-31-9-18 webphp]$ aws ecr create-repository --repository-name webphp
{
  "repository": {
    "registryId": "396459200938",
    "repositoryName": "webphp",
    "repositoryArn": "arn:aws:ecr:us-east-1:396459200938:repository/webphp",
    "createdAt": 1530156048.0,
    "repositoryUri": "396459200938.dkr.ecr.us-east-1.amazonaws.com/webphp"
  }
}

```

```
docker tag webphp <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/webphp
```

```

[ec2-user@ip-172-31-9-18 webphp]$ docker images
REPOSITORY                                     TAG          IMAGE ID
396459200938.dkr.ecr.us-east-1.amazonaws.com/webphp  latest       6478cfd2d763

```

```
$(aws ecr get-login --no-include-email)
```

This will return a login command to ECR and run it automatically in our shell. You may get a warning, but as long as “Login Succeeded” is returned, you should be able to push the image to ECR.

```

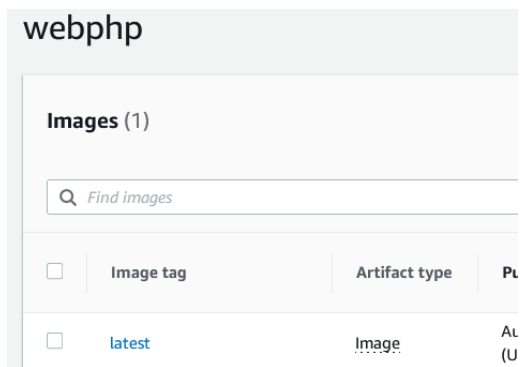
[ec2-user@ip-172-31-9-18 webphp]$ $(aws ecr get-login --no-include-email)
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded

```

Use docker push command to push to ECR, using the repository URI which was returned in our create-repository step or is available through the ECR console.

```
docker push <aws_accountid>.dkr.ecr.us-east-1.amazonaws.com/webphp
```

Once complete, you should now see the first (and latest) image in the webphp repository in the [ECR console](#):



Deploying our image to ECS

To deploy to ECS, we will follow these 3 steps:

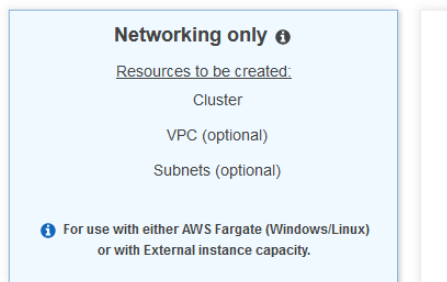
1. Create a Cluster
2. Configure a task definition
3. Configure a service

Creating a Cluster

From the [ECS Console](#), choose Create Cluster. We will use Networking Only, which means AWS Fargate will manage the underlying compute so that we can focus on running our application. Optionally, you can use EC2 Linux + Networking, which gives you complete control of the underlying EC2 instances but also adds additional overhead to manage.

Select cluster template

The following cluster templates are available to simplify cluster creation. Additional templates can be added later.



Enter a cluster name and then create a new VPC. You can leave the default networking configuration:

Configure cluster

Cluster name* ⓘ

Networking

Create a new VPC for your cluster to use. A VPC is an isolated portion of the AWS Cloud populated by A Fargate tasks.

Create VPC ☒ Create a new VPC for this cluster

CIDR block ⓘ

Subnet 1 ⓘ

Subnet 2 ⓘ

Click “Create” and the provisioning of your cluster resources will begin:

Launch status

Your container instances are launching, and it may take a few minutes until they are in the running

[Back](#) [View Cluster](#)

🔗 ECS status - 1 of 2 complete **phpwebapp**

✓ **ECS cluster**
ECS Cluster phpwebapp successfully created

ⓘ **CloudFormation Stack**
Creating CloudFormation stack resources

Once complete, you can click “View Cluster”

[Back](#) [View Cluster](#)

ECS status - 2 of 2 complete **phpwebapp**

✓ **ECS cluster**
ECS Cluster phpwebapp successfully created

✓ **CloudFormation Stack**
CloudFormation stack [EC2ContainerService-phpwebapp](#) and its

Once completed, you can view your cluster and ensure that it is active. No tasks will be running at this point:

Cluster : phpwebapp

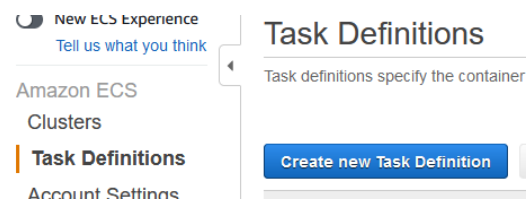
Get a detailed view of the resources on your cluster.

| | |
|--------------------------------|------------------------------|
| Cluster ARN | arn:aws:ecs:us-east-1:396459 |
| Status | ACTIVE |
| Registered container instances | 0 |
| Pending tasks count | 0 Fargate, 0 EC2, 0 External |
| Running tasks count | 0 Fargate, 0 EC2, 0 External |
| Active service count | 0 Fargate, 0 EC2, 0 External |
| Draining service count | 0 Fargate, 0 EC2, 0 External |

Create a Task Definition

You will need to create a task definition next. This describes the containers that will be deployed as tasks to your cluster.

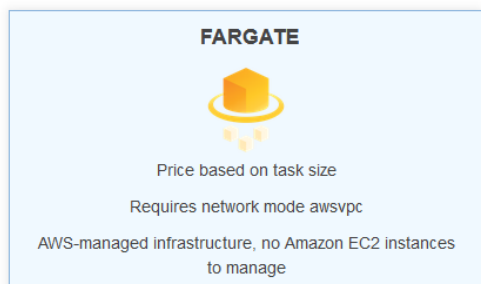
In the ECS Console, on the left menu choose Task Definitions and click “Create new task definition”.



Select Fargate as the launch type compatibility (or match whatever cluster template you deployed).

Select launch type compatibility

Select which launch type you want your task definition to be compatil



Click “Next Step”. Enter a Task Definition name and set the Operating System family to Linux.

Configure task and container definitions

A task definition specifies which containers are included in your task and how they interact with each other for your containers to use. [Learn more](#)

Task definition name* ⓘ

Requires compatibilities* FARGATE

Task role

Select a role...

↻

Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the [IAM Console](#) ↗

Network mode

awsvpc

ⓘ

If you choose <default>, ECS will start your container using Docker's default networking mode, which is Bridge on Linux and NAT on Windows. Windows tasks support the <default> and awsvpc network modes.

Operating system family

Linux

ⓘ

Choose “Create New role” for Task Execution IAM role (or use existing if you have one). This is used for logging metrics to Cloudwatch.

Task execution IAM role

This role is required by tasks to pull container images and publish container logs to Amazon Cloudwatch. If you already have the `ecsTaskExecutionRole` already, we can create one for you.

Task execution role

Create new role

ⓘ

Under Task Size section, choose an acceptable task memory and cpu.

Task size ⓘ

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 or External launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

Task memory (GB)

0.5GB

▼


The valid memory range for 0.25 vCPU is: 0.5GB - 2GB.

Task CPU (vCPU)


0.25 vCPU

▼

The valid CPU for 0.5 GB memory is: 0.25 vCPU

Task memory maximum allocation for container memory reservation


0 512 shared of 512 MiB

Task CPU maximum allocation for containers


0 256 shared of 256 CPU units

Next, click “Add Container”.

Container definitions

Add container

| Container ... | Image |
|---------------|-------|
| | |

This will pop up a dialog box where we can enter information about the container image we will be hosting in our container tasks. Enter the container name and ECR image location, which was created earlier, and add the port that is exposed from the container.

▼ Standard

Container name* ci-webphp

Image* accountid dkr.ecr.us-east-1.amazonaws.com/webphp

Private repository authentication* ☐

Memory Limits (MiB) Soft limit 128

+ Add Hard limit

Define hard and/or soft memory limits in MiB for your container. Hard and soft "memory" and "memoryReservation" parameters, respectively, in task definition. ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings Container port Protocol

80

tcp

+ Add port mapping

Leave the additional sections as default settings and click "Add".

Cancel

Add

Verify that your container was added successfully.

Add container

| Container ... | Image | Hard/Soft ... |
|---------------|---------------|---------------|
| ciPHPW... | 3964592009... | --/-- |

Click Create to finish creating your task definition.

Previous

Create

Verify the task definition is launched successfully.

Launch Status

Task definition status - 3 of 3 completed

Create Execution Role

Execution Role AmazonECSTaskExecutionRole created [Learn more](#)

Create Task Definition: tdwebphp

tdwebphp succeeded

Create CloudWatch Log Group

✔ **CloudWatch Log Group created**
CloudWatch Log Group [/ecs/tdwebphp](#)

Create a Service

ECS Fargate containers can be deployed as single tasks, or they can include additional configurations like auto-scaling and load balancing via a service. To create service, go to the cluster we previously created and under the Services tab, click “Create”.

Cluster : phpwebapp

Get a detailed view of the resources on your cluster.

| | |
|---------------------------------------|--|
| Cluster ARN | arn:aws:ecs:us-east-1:123456789012:cluster/phpwebapp |
| Status | ACTIVE |
| Registered container instances | 0 |
| Pending tasks count | 0 Fargate |
| Running tasks count | 0 Fargate |
| Active service count | 0 Fargate |
| Draining service count | 0 Fargate |

Services

Tasks

ECS Instances

Metadata

Create

Update

Delete

Activate

For the service properties, choose the following:

- Fargate as the launch type (or EC2 if appropriate)
- Linux as the Operating system family
- Select the task definition you created earlier from the drop down
- Provide a service a name
- Enter “2” as the number of tasks. This will distribute 2 tasks across our cluster

Launch type ☒ FARGATE ?
☐ EC2
☐ EXTERNAL

[Switch to capacity provider strategy](#) ?

Operating system family Linux ?

Task Definition Family: tdwebphp Enter
Revision: 1 (latest)

Platform version LATEST ?

Cluster phpwebapp ?

Service name svcphpwebapp ?

Service type* REPLICAS ?

Number of tasks 2 ?

NOTE: Although you can certainly enter more than 2 tasks, ECS will not be able to deploy more than one task on each instance because we expose a static port 80 from our container. If we had specified ephemeral ports (using "0" instead of 80), then we could have multiple tasks of the same container running on each ECS instance.

Keep "Rolling update" as the Deployment Type and click "Next Step".

[Cancel](#) [Next step](#)

On Step 2, choose the VPC and subnets that were created earlier in the Fargate cluster creation. You can keep the default security group and keep Auto-assign public IP to "ENABLED". (Note: for production environment, the Auto-assign public IP should be "DISABLED" and the tasks should be launched in a private subnet, but to do this there are some pre-requisites to ensure that the tasks can communicate with necessary AWS services: <https://aws.amazon.com/premiumsupport/knowledge-center/ecs-fargate-tasks-private-subnet/>. So for now setting the property to ENABLED will do the trick.)

Configure network

VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC* vpc-015c78b842ae9efbc (10.0.0.0/16) ⓘ

Subnets* ⓘ

subnet-091b03618e350d2c6
(10.0.0.0/24) | phpwebapp/Public - us-e
ast-1a
assign ipv6 on creation: Disabled

subnet-09d05505a06498fd4
(10.0.1.0/24) | phpwebapp/Public - us-e
ast-1b
assign ipv6 on creation: Disabled

Security groups* svcphp-678 **Edit** ⓘ

Auto-assign public IP ENABLED ⓘ

In the Load Balancing section, you can optionally configure a load balancer. Fargate does not automatically create the load balancer for you, so if you wish to configure this you will need to first create an Application Load Balancer

(<https://docs.aws.amazon.com/elasticloadbalancing/latest/application/create-application-load-balancer.html>).

In our example, I am going to choose an empty Application Load Balancer that I created for this exercise. and then under Container to Load Balance, choose the one and only container available and click “Add to Load Balancer”:

Load balancer type*

☐ None
Your service will not use a load balancer.

☒ Application Load Balancer
Allows containers to use dynamic host port mapping (multiple tasks allowed per container). Multiple services can use the same listener port on a single load balancer with rule-based paths.

☐ Network Load Balancer
A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection model. After the load balancer receives a request, it selects a target from the target group using a flow hash routing algorithm.

☐ Classic Load Balancer
Requires static host port mappings (only one task allowed per container instance); rule-based paths are not supported.

Service IAM role Task definitions that use the awsvpc network mode use the AWSServiceRoleForECS service-created for you automatically. [Learn more](#).

Load balancer name lb-phpwebapp ⓘ

Container to load balance

Container name : port ci-webphp:80:80 **Add to load balancer**

When you click “Add to Load Balancer”, choose the listener port of 80 and the target group with a target type of “ip” and protocol of “HTTP”. Optionally, you can have the Fargate service create these for you.

Container to load balance

ci-webphp : 80 Remove X

Production listener port* 80 HTTP ⓘ

Production listener protocol* HTTP

Target group name tg-phpwebapp ⓘ

Target group protocol HTTP ⓘ

Target type ip ⓘ

Path pattern / Evaluation order default

Health check path / ⓘ

Additional health check options can be configured in the ELB console after you create your service.

Once the ECS service launches new tasks, it will automatically register those tasks in the load balancer target group. Click “Next Step” to continue.

[Cancel](#) [Previous](#) [Next step](#)

Under Set Auto Scaling, we are going to not adjust the size. You could optionally do this and the service will look at target metrics to scale the number of tasks in your service (<https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-autoscaling-stepscaling.html>).

Click “Next Step”.

Set Auto Scaling (optional)

Automatically adjust your service's desired count up and down within a specified range in response to CloudWatch alarms. You can modify your Service Auto Scaling configuration at any time to meet the needs of your application.

- Service Auto Scaling ☒ Do not adjust the service's desired count
- ☐ Configure Service Auto Scaling to adjust your service's desired count

Required

[Cancel](#) [Previous](#) [Next step](#)

On the final screen, review your settings, and then click “Create Service”.

[Cancel](#) [Previous](#) [Create Service](#)

Ensure that your service was created successfully.

Configure Task Networking

Create security group

✓ **Create security group**
svcphp-678 succeeded [sg-0e0b668f95ef766ba](#)

Set inbound rules

✓ **Set inbound rules**
succeeded [sg-0e0b668f95ef766ba](#)

Create Load Balancer

Create Service

Create service: svcphpwebapp

✓ **Service created**
Service created. Tasks will start momentarily. View: [svcphpwebapp](#)

Notice that there is an option to create a Code Pipeline so that your app is automatically updated when a build is triggered. We will do this later.

Additional integrations you can connect to your ECS service

Code Pipeline

Setup a CI/CD process from your service. You can build from source or have an ECR repository as the source for your deployment.

[Create a pipeline](#)

Click “View Service”.



This will take you to the tasks tab, where you will be able to see the tasks that your service deployed:

Service : svcphpwebapp

| | | | |
|-----------------|-------------------------------------|---------------|---|
| Cluster | phpwebapp | Desired count | 2 |
| Status | ACTIVE | Pending count | 2 |
| Task definition | tdwebphp:1 | Running count | 0 |
| Service type | REPLICA | | |
| Launch type | FARGATE | | |
| Service role | AWSServiceRoleForECS | | |
| Created By | arn:aws:iam::396459200938:user:hoog | | |

Details Tasks Events Auto Scaling Deployments Metrics Tags Logs

Task status: **Running** Stopped

Filter in this page

| Task | Task Definition | Last status | Desired status | Group |
|-----------------------------|-----------------|-------------|----------------|-------------|
| 30fcb915bfa44988adb3b90c... | tdwebphp:1 | PENDING | RUNNING | service.svc |
| b9122423b8d2435e88b2c5a2... | tdwebphp:1 | PENDING | RUNNING | service.svc |

Initially, you will see the tasks in a pending state and then eventually they will switch to a RUNNING status.

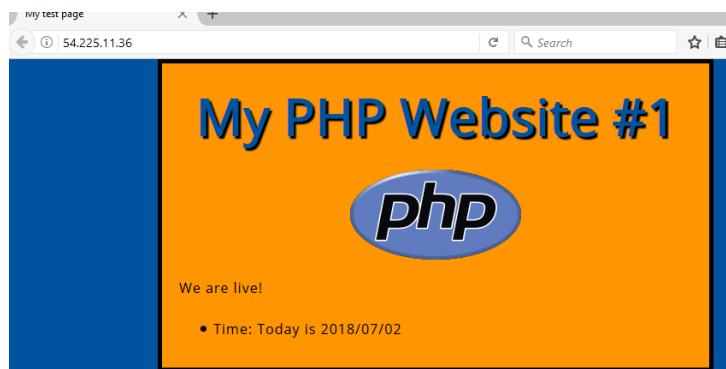
| Details | Tasks | Events | Auto Scaling | Deployments | Metrics | Tags | Logs |
|-------------------------------------|-----------------|-------------|----------------|-------------|---------|------|------|
| Task status: Running Stopped | | | | | | | |
| Filter in this page | | | | | | | |
| Task | Task Definition | Last status | Desired status | | | | |
| 30fcb915bfa44988adba3b90c... | tdwebphp:1 | RUNNING | RUNNING | | | | |
| b9122423b8d2435e88b2c5a2... | tdwebphp:1 | RUNNING | RUNNING | | | | |

To test your deployment, you can first of all test directly to the ECS instance. This will only work if your security group has port 80 open from the instance to the internet. If so, you can click on one of the running tasks and in the detail section copy the public IP address under the Network section:

Network

| | |
|---------------------|--------------------------|
| Network mode | awsvpc |
| ENI Id | eni-04fcb41d33934830b |
| Subnet Id | subnet-0cf120ce5c01bfd39 |
| Private IP | 10.0.1.195 |
| Public IP | 34.201.162.202 |
| Mac address | 0e:e1:20:5e:ce:5f |

Go to a browser and navigate to <http://<ip address>>. This should load our very simple website:



Now, a better production practice is to only expose the load balancer to the Internet as appropriate. Because we set up load balancing, you should be able to go to the EC2 Console, choose Load Balancers from the left menu, and find your Application Load Balancer in the list:

| search | lb-phpwebapp | Add filter |
|--------------|-----------------------------|------------|
| Name | DNS name | State |
| lb-phpwebapp | lb-phpwebapp-1321452437.... | Active |

Click on the Listeners tab and in the Rules section it will have a forwarding rule. Click on the target group (e.g., tgFGPHPApp-Aug01):

Load balancer: **lb-phpwebapp**

Description **Listeners** Monitoring Integrated services Tags

Listeners listen for connection requests using their protocol and port. You can add, remove, or update listeners and

To view and edit listener attributes, select the listener and choose Edit.

Add listener Edit Delete

| <input type="checkbox"/> | Listener ID | Security policy | SSL Certificate | Rules |
|--------------------------|--------------------------------------|-----------------|-----------------|--|
| <input type="checkbox"/> | HTTP : 80 arn...cecf4282fe11812a▼ | N/A | N/A | Default: forwarding to tg-phpwebapp View/edit rules |

When you click on the Target Group and then the Targets tab, you will be able to see the Fargate tasks that were added as targets to the group. Assuming security groups are open and the containers were deployed successfully, you should see 2 healthy targets:

☒ tg-phpwebapp arn:aws:elasticloadbalancin... 80 HTTP IP

Target group: tg-phpwebapp

Details **Targets** Monitoring Health checks Attributes Tags

Registered targets (2)

| <input type="checkbox"/> | IP address | Port | Zone | Health status |
|--------------------------|------------|------|------------|---------------|
| <input type="checkbox"/> | 10.0.0.91 | 80 | us-east-1a | healthy |
| <input type="checkbox"/> | 10.0.1.126 | 80 | us-east-1b | healthy |

Return to the load balancer and copy the DNS name, and then load it into a browser. You should see the website load correctly:



That completes deployment of our container into ECS Fargate.

Deploying our container application changes using CI/CD

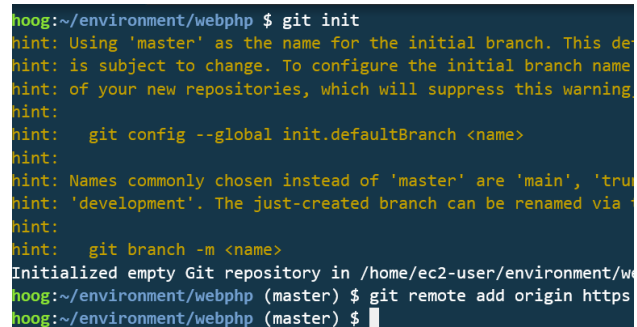
For this example, we are going to create a pipeline to orchestrate our continuous delivery (CI/CD) process. In our case, we are simply going to re-compile our container and push it into our ECS environment when changes are made.

Create a CodeCommit repository

The first thing we must do is deploy our code to a git based repo, in which case we are going to use AWS Code Commit.

Follow instructions to [create a new repository](#) and [set up a connection](#). Return to Cloud9 and make sure you are in the webphp directory, and then initialize the webphp folder as a git repository:

```
git init
git remote add origin <repo url>
```

A terminal window showing the process of initializing a git repository. The user is in the directory ~/environment/webphp. The terminal output shows the following commands and their outputs:
1. `git init`: Outputs several hints about the initial branch name and suggests using 'main' or 'development' instead of 'master'.
2. `git config --global init.defaultBranch <name>`: A hint to configure the default branch.
3. `git branch -m <name>`: A hint to rename the branch.
4. `git remote add origin https://...`: A hint to add the remote repository.
5. `git push origin master`: A hint to push the code to the remote repository.
The terminal shows the repository is initialized and the user is now in the master branch.

Once that is complete, in your development environment, add an additional file to our source folder called **buildspec.yml**. This will be used by CodeBuild to build our new container from the Dockerfile and source code. The contents of the buildspec.yml file should be as follows, but make sure to replace the repository name and container name with your account specific values:

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - export AWS_ACCOUNT_ID=$(aws sts get-caller-identity --query "Account" --output=text)
      - aws --version
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --username AWS --
password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com
      - REPOSITORY_URI=$AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/<repository
name>
      - IMAGE_TAG=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - ls
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
```

```

- docker push $REPOSITORY_URI:$IMAGE_TAG
- echo Writing image definitions file...
- printf '["name":"<container name in task definition >","imageUri":"%s"]]'
$REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
artifacts:
  files: imagedefinitions.json

```

The buildspec.yml file above is a standard docker build spec pulled from the [Continuous Deployment with AWS CodePipeline Tutorial](https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html). You can also see more about the buildspec.yml reference at <https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html>

At this point you should have your local project folder with the sample PHP app, the Dockerfile, and the buildspec.yml file all as shown:



Now run the following git commands to upload to the CodeCommit repository:

```

git add .
git commit -m "Initial Commit"
git push

```

We now have our initial code committed to the git repo.

Create a CodePipeline

Now, let's [create a new Code Pipeline](#). Give the Code Pipeline a name and use the default settings and click "Next".

Choose pipeline settings [Info](#)

Pipeline settings

Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.

No more than 100 characters

Service role

☒ **New service role**
Create a service role in your account

☐ **Existing service role**
Choose an existing service role from your account

Role name

Type your service role name

☒ Allow AWS CodePipeline to create a service role so it can be used with this new pipeline

► **Advanced settings**

Cancel

Next

Select your source provider, repository name, and branch name:

Add source stage [Info](#)

Source

Source provider
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

AWS CodeCommit

Repository name
Choose a repository that you have already created where you have pushed your source code.

Q phpwebapp X

Branch name
Choose a branch of the repository

Q master X

Change detection options
Choose a detection mode to automatically start your pipeline when a change occurs in the source code.

☒ **Amazon CloudWatch Events (recommended)**
Use Amazon CloudWatch Events to automatically start my pipeline when a change occurs

☐ **AWS CodePipeline**
Use AWS CodePipeline to check periodically for changes

Output artifact format
Choose the output artifact format.

☒ **CodePipeline default**
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.

☐ **Full clone**
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

Cancel

Previous

Next

Click Next. Select CodeBuild as the build provider. This will be used to create our docker image and push to ECR. You will first need to select “Create Project”:

or

Create project [↗](#)

A popup window will appear. Enter a project name:

Create build project

Project configuration

Project name

phpwebapp-build

A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special

Description - optional

Enable concurrent build limit - optional
Limit the number of allowed concurrent builds for this project.
☐ Restrict number of concurrent builds this project can start

Additional configuration
tags

Next, under Managed Image, choose Ubuntu as the OS and use the Standard runtime. Check the box for privileged.

Environment

Environment image



Managed image

Use an image managed by AWS CodeBuild



Custom image

Specify a Docker image

Operating system

Ubuntu



The programming language runtimes are now included in the standard image of Ubuntu recommended for new CodeBuild projects created in the console. See [Docker Images Pro details](#).

Runtime(s)

Standard



Image

aws/codebuild/standard:6.0



Image version

Always use the latest image for this runtime version



Environment type

Linux



Privileged



Enable this flag if you want to build Docker images or want your builds to get elevated privileges

Leave the service role as the default:

Service role



New service role

Create a service role in your account



Existing service role

Choose an existing service role

Role name

codebuild-phpwebapp-build-service-role

Type your service role name

► Additional configuration

Timeout, certificate, VPC, compute type, environment variables, file systems

In the build specification, note that we will use the buildspec.yml file from the source code root directory. We added this earlier. No changes needed here:

Buildspec

Build specifications



Use a buildspec file

Store build commands in a YAML-formatted buildspec file



Insert build commands

Store build commands as build commands

Buildspec name - optional

By default, CodeBuild looks for a file named buildspec.yml in the source code root directory. If your buildspec location, enter its path from the source root here (for example, buildspec-two.yml or configuration/buildspec.yml)

You may leave the final configurations for the CodeBuild project as default settings and choose Continue to CodePipeline to save the CodeBuild project.

Logs

CloudWatch

☒ **CloudWatch logs - optional**
Checking this option will upload build output logs to CloudWatch

Group name

Stream name

S3

☐ **S3 logs - optional**
Checking this option will upload build output logs to S3.

Cancel

Continue to CodePipeline

Once you do that, the popup window will close and you will get a success message.

Project name
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.

or

✔ Successfully created phpwebapp-build in CodeBuild.

✕

Click “Next”.

Previous

Skip build stage

Next

In this final step, we will deploy our code to our ECS Fargate Service by choosing Amazon ECS as the deployment provider, followed by the Cluster and service that we created earlier:

Add deploy stage [Info](#)

Deploy - optional

Deploy provider
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Amazon ECS

Region

US East (N. Virginia)

Cluster name
Choose a cluster that you have already created in the Amazon ECS console. Or create a cluster in the Amazon ECS console and then return to this task.

Q phpwebapp X

Service name
Choose a service that you have already created in the Amazon ECS console for your cluster. Or create a new service in the Amazon ECS console and then return to this task.

Q svcphpwebapp X

Image definitions file - optional
Enter the JSON file that describes your service's container name and the image and tag.

MyFilename.json

Deployment timeout - optional
Enter the timeout in minutes for the deployment action.

Cancel

Previous

Skip deploy stage

Next

Click Next. Review our settings and click “Create Pipeline”.

Step 3: Add build stage

Build action provider

Build action provider

AWS CodeBuild

ProjectName

phpwebapp-build

Step 4: Add deploy stage

Deploy action provider

Deploy action provider

Amazon ECS

ClusterName

phpwebapp

ServiceName

svcphpwebapp

Cancel

Previous

Create pipeline

We now have a Code Pipeline in place for deploying code changes to our ECS environment. It will automatically kick off when you save the Pipeline, triggering a CodeBuild from our Source Provider:

webphp-pipeline

Source Succeeded
Pipeline execution ID: f04c4f1e-6928-41e3-948e-b23d6514a300

Source
AWS CodeCommit
Succeeded - Just now
4ac7d585

4ac7d585 Source: initial commit

Disable transition

Build In progress
Pipeline execution ID: f04c4f1e-6928-41e3-948e-b23d6514a300

Build
AWS CodeBuild
In progress - Just now
Details

You may initially see an error in the Build stage.

Build Failed
Pipeline execution ID: f04c4f1e-6928-41e3-948e-b23d6514a300

Build
AWS CodeBuild
Failed - Just now
Action execution failed
View in CodeBuild

If you do get an error, you can click on the CodeBuild link to see the error. It will probably look something like this:

```
30 An error occurred (AccessDeniedException) when calling the GetAuthorizationToken operation: User: arn:aws:iam::c419-4829-8213-da56bf064862:role/codebuild-phpwebapp-build-service-role is not authorized to perform: ecr:GetAuthorizationToken on resource: * be
31 Error: Cannot perform an interactive login from a non TTY device
```

This error indicates that a role does not have the necessary access to perform actions on ECR. To apply access, I will open the [IAM Roles](#) console and find the Role that was created earlier (codebuild-phpwebapp-build-service-role). Add 2 out of the box policies to give CodeBuild access to ECS and ECR (for production environments, you should reduce the permissions down to the exact actions needed):

codebuild-phpwebapp-build-service-role

Summary

Creation date

August 13, 2022, 22:19 (UTC-04:00)

Last activity

None

Permissions

Trust relationships

Tags

Access Advisor

Permissions policies (3)

You can attach up to 10 managed policies.

🔍 Filter policies by property or policy name and press enter



Policy name ↗



⊕ [CodeBuildBasePolicy-phpwebapp-build-us-east-1](#)



⊕  [AmazonEC2ContainerRegistryFullAccess](#)



⊕  [AmazonECS_FullAccess](#)

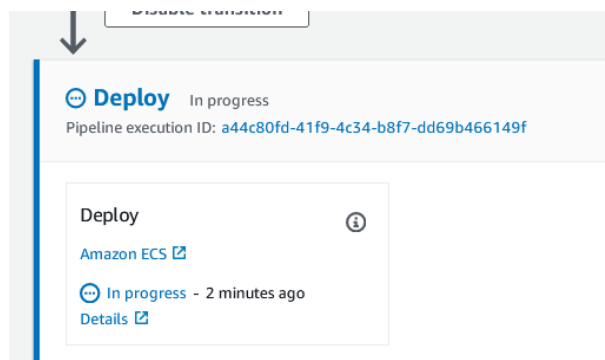
Once I save the role, you can kick off this pipeline by manually selecting the “Release change” button or you can push new changes to CodeCommit, which will automatically kick off the pipeline. Let’s go ahead and make a change by changing the header tag in the index.php file:

```
<h1>My Updated PHP Site</h1>
```

Update the repository:

```
git add .
git commit -m "Updated title"
git push
```

The pipeline will restart and take several minutes to build and push the new docker image to ECR and then deploy new tasks to ECS Fargate. You can click the Amazon ECS link to view the provisioning status in the cluster:



The screenshot shows the AWS CodePipeline console. At the top, a 'Deploy' stage is highlighted with a blue bar and a downward arrow. Below this, a 'Deploy' action is shown with a status of 'In progress'. The pipeline execution ID is 'a44c80fd-41f9-4c34-b8f7-dd69b466149f'. A details box for the 'Deploy' action is open, showing a link to 'Amazon ECS' and a status of 'In progress - 2 minutes ago' with a 'Details' link.

You will see in the Events tab the status of registering new task and de-registering old tasks. (This will take at least 5 minutes because the default connection draining for our load balancer is 300 seconds.)

Details

Tasks

Events

Auto Scaling

Deployments

Metrics

Tags

Logs

▼ Filter in this page

| Event Id | Event Time | Message |
|--------------------------------------|---------------------------|--|
| 63b02dd8-adf2-4480-9e48-044957e1d9c9 | 2022-08-13 22:33:15 -0400 | service svcphpwebapp has begun draining connections on 2 tasks. |
| 249571d3-8e17-4515-abb1-8669d08ff120 | 2022-08-13 22:33:15 -0400 | service svcphpwebapp deregistered 2 targets in target-group tg-phpwebapp |
| 55160566-fc09-4842-8c5c-37103322e6fd | 2022-08-13 22:32:36 -0400 | service svcphpwebapp registered 2 targets in target-group tg-phpwebapp |

Once all tasks in ECS have completed their provisioning/de-provisioning, all steps in the pipeline should show as “succeeded”.

webphp-pipeline

```
graph TD; Source[Source] --> Build[Build]; Build --> Deploy[Deploy];
```

Source Succeeded
Pipeline execution ID: [a44c80fd-41f9-4c34-b8f7-dd69b466149f](#)

Source ⓘ
[AWS CodeCommit](#)

✓ Succeeded - 11 minutes ago
[8ff46417](#)

[8ff46417](#) Source: Updated title

↓

Disable transition

Build Succeeded
Pipeline execution ID: [a44c80fd-41f9-4c34-b8f7-dd69b466149f](#)

Build ⓘ
[AWS CodeBuild](#)

✓ Succeeded - 8 minutes ago
[Details](#)

[8ff46417](#) Source: Updated title

↓

Disable transition

Deploy Succeeded
Pipeline execution ID: [a44c80fd-41f9-4c34-b8f7-dd69b466149f](#)

Deploy ⓘ
[Amazon ECS](#) [↗](#)

✓ Succeeded - 1 minute ago
[Details](#) [↗](#)

[8ff46417](#) Source: Updated title

You can now view your updated website in production by accessing the application load balancer of your front-end website. Great job!



Cleanup

You will want to manually delete resources that were created to avoid unnecessary charges. These include:

- CodePipeline and CodeBuild projects
- CodeCommit Repositories
- AWS Cloud9 Environments
- AWS EC2 Load Balancers, Target Groups and Security Groups
- AWS ECS Fargate Clusters
- AWS ECR Task Definitions
- AWS IAM Roles