

GitHub Actions Workflow Documentation for AWS ECS Deployment

Problem Statement

Configure a GitHub Actions workflow to automate the deployment of a simple web application on GitHub to AWS ECS (Elastic Container Service). The pipeline should include deployment, integration tests, and rollback functionality.

Steps Overview

1. Checkout code from GitHub repository
2. Build optimized Docker image with multistage build to serve with Nginx as a reverse proxy
3. Push image to ECR (Elastic Container Registry)
4. Deploy on ECS
5. Perform integration tests and implement rollback functionality in case of failure

Prerequisites

- A GitHub repository with the source code of the web application
- An AWS account with access to ECS, ECR, and IAM roles configured for GitHub Actions
- Docker installed locally for testing the build process
- AWS CLI configured with necessary permissions
- Required secrets added to your GitHub repository (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_ECR_REGISTRY, AWS_REGION, ECS_CLUSTER_NAME, ECS_SERVICE_NAME, ECS_TASK_DEFINITION)

Setting Up the Environment

Before we start, ensure you have the following:

- A GitHub repository with your web application code
- An AWS account with ECR and ECS configured
- Necessary AWS credentials and permissions(IAM)
- Node.js and npm installed for Docker image building
- Docker installed for building the image

Step 1: Pushing My local application to GitHub Repository

Step 2: Build Optimized Docker Image with Multistage Build

Add steps to build the Docker image using a multistage build to serve the application with Nginx.

```
FROM node:alpine AS build

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build # Assuming you have a build script in package.json

FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html

# Copy custom Nginx configuration if needed
#COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Step 3- Go to Your AWS Account :

Configure Necessary AWS credentials and permissions(IAM) for Role

- For Role Give Permission to [AmazonEC2ContainerServiceRole](#) & [AmazonECSTaskExecutionRolePolicy](#)

ECS-role-cidc info

Allows ECS to create and manage AWS resources on your behalf.

Summary

Creation date: August 06, 2024, 21:23 (UTC+05:30)

Last activity: 45 minutes ago

ARN: arn:aws:iam::588136850995:role/ECS-role-cidc

Maximum session duration: 1 hour

Permissions | Trust relationships | Tags | Access Advisor | Revoke sessions

Permissions policies (2) Info

You can attach up to 10 managed policies.

Policy name	Type	Attached entities
AmazonEC2ContainerServiceRole	AWS managed	2
AmazonECSTaskExecutionRolePolicy	AWS managed	2

Step 4: Go To ECR - Create One Repository

Amazon ECR > Private registry > Repositories

Private repositories

Repositories (1)

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
todocidc	588136850995.dkr.ecr.ap-south-1.amazonaws.com/todocidc	August 07, 2024, 11:17:28 (UTC+05:5)	Disabled	Manual	AES-256

Step 5: Go to ECS and create 1 Cluster

Amazon Elastic Container Service

Clusters (1) Info

Cluster	Services	Tasks	Container instances	CloudWatch monitoring	Capacity provider strategy
TodoCluster	1	0 Open... 1 Run...	0 EC2	Default	No default found

Step 6: Create Task Definition

Purpose of the task Definition file -

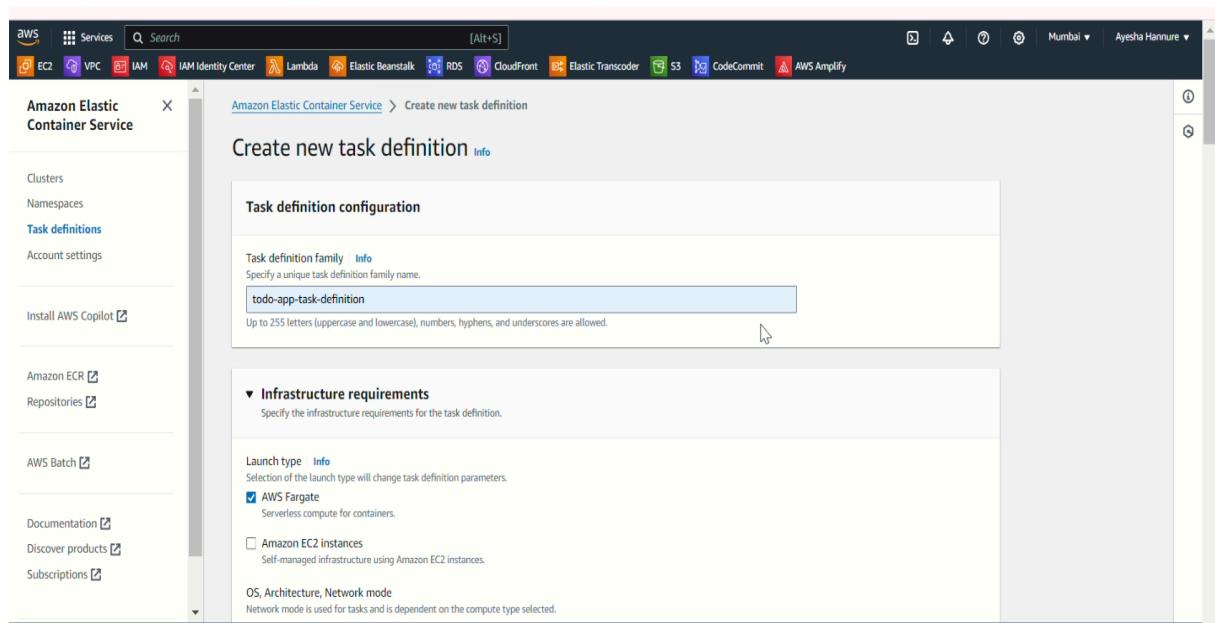
Container Configuration, Task Configuration, Resource Allocation, Resource allocation, Deployment Management and Scaling Optimization.

Task Definition file is a bule-print of your container. The file tells ECS how to set up and run your containers.

You can define how many instances of your container you want to run. ECS will automatically adjust the number of instances based on the workload.ou can specify how much CPU, memory, and other resources each container needs. This helps optimize your costs and performance.

Create a Task Definition

- Navigate to the ECS console and select "Task Definitions".
- Click "Create new task definition".
- Choose a task definition family name and select the launch type (Fargate or EC2).
- Under "Container definitions", add a new container.
- Provide a container name and specify the image URI (e.g.,
`your-account.dkr.ecr.region.amazonaws.com/your-image:latest`).
- Configure other container properties as needed (CPU, memory, port mappings, environment variables, etc.).
- Click "Create".



Launch type [Info](#)
 Selection of the launch type will change task definition parameters.
 AWS Fargate
 Serverless compute for containers.

Amazon EC2 instances
 Self-managed infrastructure using Amazon EC2 instances.

OS, Architecture, Network mode
 Network mode is used for tasks and is dependent on the compute type selected.
Operating system/Architecture [Info](#) **Network mode** [Info](#)

Linux/X86_64	awservc
--------------	---------

Task size [Info](#)
 Specify the amount of CPU and memory to reserve for your task.
CPU
 1 vCPU
Memory
 3 GB

Task roles - conditional

Task role [Info](#)
 A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the [IAM console](#).
 ECS-role-cidc

Task execution role [Info](#)
 A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM role created, we can create one for you.

Private registry [Info](#)
 Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.
 Private registry authentication

Port mappings [Info](#)
 Add port mappings to allow the container to access ports on the host to send or receive traffic. For port name, a default will be assigned if left blank.

Container port	Protocol	Port name	App protocol
3000	TCP	container-port-protocol	None

Add port mapping

Read only root file system [Info](#)
 When this parameter is turned on, the container is given read-only access to its root file system.
 Read only

Resource allocation limits - conditional [Info](#)
 Container-level CPU, GPU, and memory limits are different from task-level values. They define how much resources are allocated for the container. If container attempts to exceed the memory specified in hard limit, the container is terminated.

CPU	GPU	Memory hard limit	Memory soft limit
1 in vCPU	1	3 in GB	1 in GB

Environment variables - optional

Step 7: Create Service

Create or Update a Service

- Navigate to the ECS console and select "Clusters".
- Choose your desired cluster.
- Click "Create service" or select an existing service to update.
- Provide a service name and select the task definition created in step 1.
- Configure desired number of tasks, load balancing, and other service settings.
- Click "Create" or "Save" to deploy the service.

[Alt+S] [] [?]

Identity Center Lambda Elastic Beanstalk RDS CloudFront Elastic Transcoder S3 CodeCommit AWS Amplify

Existing cluster
TodoCluster

▼ Compute configuration (advanced)

Compute options | Info To ensure task distribution across your compute types, use appropriate compute options.

Capacity provider strategy Specify a launch strategy to distribute your tasks across one or more capacity providers.

Launch type Launch tasks directly without the use of a capacity provider strategy.

Capacity provider strategy | Info Select either your cluster default capacity provider strategy or select the custom option to configure a different strategy.

Use cluster default No default capacity provider strategy configured for this cluster.

Use custom (Advanced)

Capacity provider FARGATE Base | Info Weight | Info

0 1

Add capacity provider

Platform version | Info Specify the platform version on which to run your service.

LATEST

[Alt+S]

Application type [Info](#)
Specify what type of application you want to run.

Service
Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.

Task
Launch a standalone task that runs and terminates. For example, a batch job.

Task definition
Select an existing task definition. To create a new task definition, go to [Task definitions](#).

Specify the revision manually
Manually input the revision instead of choosing from the 100 most recent revisions for the selected task definition family.

Family **Revision**

Service name
Assign a unique name for this service.

Service type [Info](#) 
Specify the service type that the service scheduler will follow.

Replica
Place and maintain a desired number of tasks across your cluster.

Daemon
Place and maintain one copy of your task on each container instance.

Desired tasks
Specify the number of tasks to launch.

[Alt+S]

Family Revision

Service name
Assign a unique name for this service.

Service type [Info](#)
Specify the service type that the service scheduler will follow.

Replica
Place and maintain a desired number of tasks across your cluster.

Daemon
Place and maintain one copy of your task on each container instance.

Desired tasks
Specify the number of tasks to launch.

Deployment options 

Deployment type [Info](#)
Select a deployment controller type for the service.

Rolling update

Blue/green deployment (powered by AWS CodeDeploy)

Min running tasks % [Info](#)
Specify the minimum percent of running tasks allowed during a service deployment.

Step 7 - Checkout Code in GitHub Repository

- Go to Github click on Action
- Create workflow File
- Work Flow file contain Configure AWS credential,Login to ECR ,Build Image to ECR and Push to Deploy in ECS,Run integration tests,Rollback on failure .

Here is the code snippet of my Workflow file.

```
name: CICD

on:
  push:
    branches: [ main ]

jobs:
  build-and-deploy:
    runs-on: [ ubuntu-latest ]
    steps:
      - name: Checkout source
        uses: actions/checkout@v3
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v3
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: 'ap-south-1'
      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1
        with:
          mask-password: 'true'

      - name: Build, tag, and push image to Amazon ECR
        id: build-image
        env:
          ECR_REGISTRY: ${{ steps.login-ecr.outputs.registry }}
          IMAGE_TAG: ${{ github.sha }}
          REPOSITORY: todocid
        run:
          # Build a docker container and
          # push it to ECR so that it can
          # be deployed to ECS.
          docker build -t $ECR_REGISTRY/$REPOSITORY:$IMAGE_TAG .
          docker push $ECR_REGISTRY/$REPOSITORY:$IMAGE_TAG
          echo "image=$ECR_REGISTRY/$REPOSITORY:$IMAGE_TAG" >> $GITHUB_OUTPUT
      - name: Fill in the new image ID in the Amazon ECS task definition
        id: task-def
        uses: aws-actions/amazon-ecs-render-task-definition@v1
        with:
          task-definition: todo-app-task-definition.json
          container-name: todo-node-app
          image: ${{ steps.build-image.outputs.image }}
      - name: Deploy Amazon ECS task definition
        uses: aws-actions/amazon-ecs-deploy-task-definition@v1
```

with:

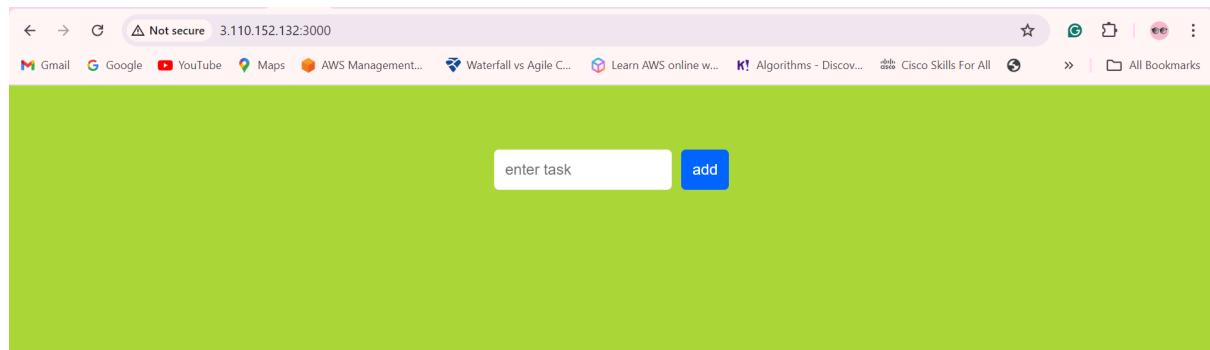
```
task-definition: ${{ steps.task-def.outputs.task-definition }}
service: todo-app-service
cluster: TodoCluster
wait-for-service-stability: true

- name: Run integration tests
  id: integration-tests
  run: |
    curl -f http://127.0.0.1:2000
  continue-on-error: true

- name: Rollback on failure
  if: failure()
  run: |
    # Get the previous task definition from ECS service history or version control
    PREVIOUS_TASK_DEFINITION=$(aws ecs describe-services --cluster ${{
    secrets.ECS_CLUSTER }} --services ${{
    secrets.ECS_SERVICE }} --query
    "services[0].deployments[1].taskDefinition" --output text)
    aws ecs update-service --cluster ${{
    secrets.ECS_CLUSTER }} --service ${{
    secrets.ECS_SERVICE }} --task-definition $PREVIOUS_TASK_DEFINITION
```

```
- name: Notify failure
  if: failure()
  run: |
    echo "Deployment failed and rolled back"
```

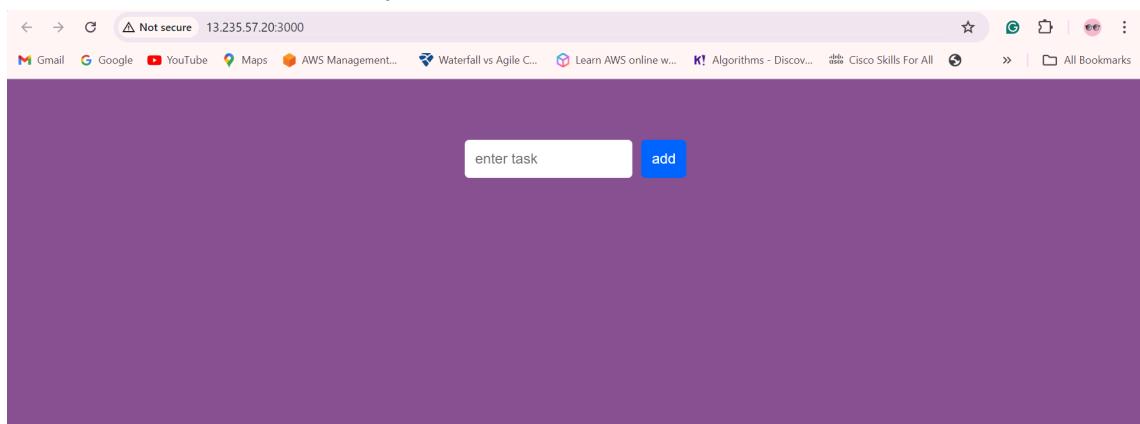
This is snippet of the after successful Deployment -



Snippet of the Multiple images -

Images (7)						
	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
<input type="checkbox"/>	b54d54fd6f0ca40c5875c50206a831f0cd6c1ba	Image	August 08, 2024, 12:55:02 (UTC+05:5)	100.07	Copy URI	sha256:51eb7aa5d5d7ce...
<input type="checkbox"/>	6b477519063a67031f44ede11a46e34375e7a39b	Image	August 08, 2024, 12:38:33 (UTC+05:5)	100.07	Copy URI	sha256:4bc5e77a562da6...
<input type="checkbox"/>	82e4b04741cf8b192cd96b0df490b3b2b78ca388	Image	August 07, 2024, 23:26:35 (UTC+05:5)	100.07	Copy URI	sha256:237ff124f59be04...
<input type="checkbox"/>	be6c694a3792bc72929db5e7f1aff2a271362c70	Image	August 07, 2024, 23:03:58 (UTC+05:5)	100.07	Copy URI	sha256:31b5f09b590e7e...
<input type="checkbox"/>	f32fdc77e7eed41fb975f82a6ecac4030b95ffb99	Image	August 07, 2024, 22:53:36 (UTC+05:5)	100.07	Copy URI	sha256:423d005c8843ac...
<input type="checkbox"/>	00d4538ce2dd0c9789d5e5f9de7483dfd6efc18	Image	August 07, 2024, 15:41:44 (UTC+05:5)	100.07	Copy URI	sha256:4551ae9fa29f7d3...
<input type="checkbox"/>	4f17f55c4c9103d53c79f95f25aad9c68d789a17	Image	August 07, 2024, 15:34:15 (UTC+05:5)	100.07	Copy URI	sha256:aba95ab8c254c4...

This image after the Re-deployment -



Step 8- Create nginx.config File

- Nginx is use for Request handling Process .
- SSL/TLS Termination: Handles encryption and decryption of HTTPS connections, protecting data in transit.
- Web Server: Serves static content directly to clients without involving backend applications.

Nginx.config file explanation.

1. server block:

- Defines a virtual server configuration. In this case, it listens on port 3000 for incoming HTTP requests.

2. location / block:

- This block defines how to handle requests to the root URL `/`.
- `root /usr/share/nginx/html;` specifies the directory to serve files from.
- `index index.html index.htm;` sets the default files to look for when a directory is requested.
- `try_files $uri $uri/ /index.html;` tries to serve the requested file. If it doesn't exist, it falls back to serving `index.html`. This is important for handling client-side routing in single-page applications (SPAs) like React.

3. location /api/ block:

- This block defines how to handle requests to URLs starting with `/api/`.

- `proxy_pass http://backend:5000;` forwards these requests to the backend service. Replace `http://backend:5000` with your actual backend service URL.
- `proxy_set_header` directives add headers to the request forwarded to the backend. These headers help the backend service to correctly identify the original request details, such as the client's IP address and the protocol used.

Step 9: Mention Your Nginx Server to into Docker file .

Serve the App with Nginx

Base Image:

```
FROM nginx:stable-alpine
```

Explanation -

This line sets the base image to `nginx:stable-alpine`, a lightweight Nginx image.

Copy Built Assets:

```
COPY --from=build /app/build /usr/share/nginx/html
```

Explanation -

Copies the build artifacts from the `build` stage (`/app/build`) to the Nginx HTML directory (`/usr/share/nginx/html`). The `--from=build` part specifies that the source files are from the previous build stage.

Copy Custom Nginx Configuration:

```
COPY nginx.conf /etc/nginx/conf.d/default.conf
```

Explanation -

Copies the custom Nginx configuration file (`nginx.conf`) into the Nginx configuration directory.

Expose Port 80:

```
EXPOSE 3000
```

Explanation -

Exposes port 3000, which is the default HTTP port. This makes the application accessible on this port.

Start Nginx Server:

```
CMD ["nginx", "-g", "daemon off;"]
```

Explanation -

Sets the command to start the Nginx server. The `-g "daemon off;"` option tells Nginx to run in the foreground, which is required for Docker containers to stay alive.