

Deploy and Maintain Containerized Applications on AWS

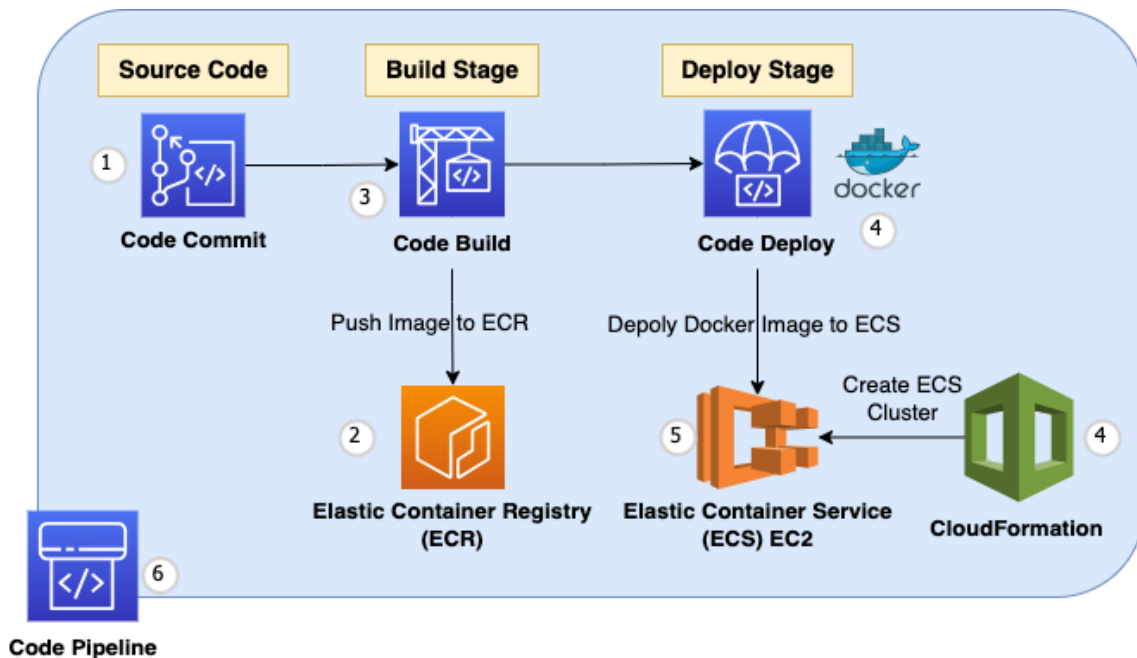


Fig 1: The Code Pipeline architecture describing the steps to deploy and maintain containerized applications utilizing Amazon ECR and ECS services

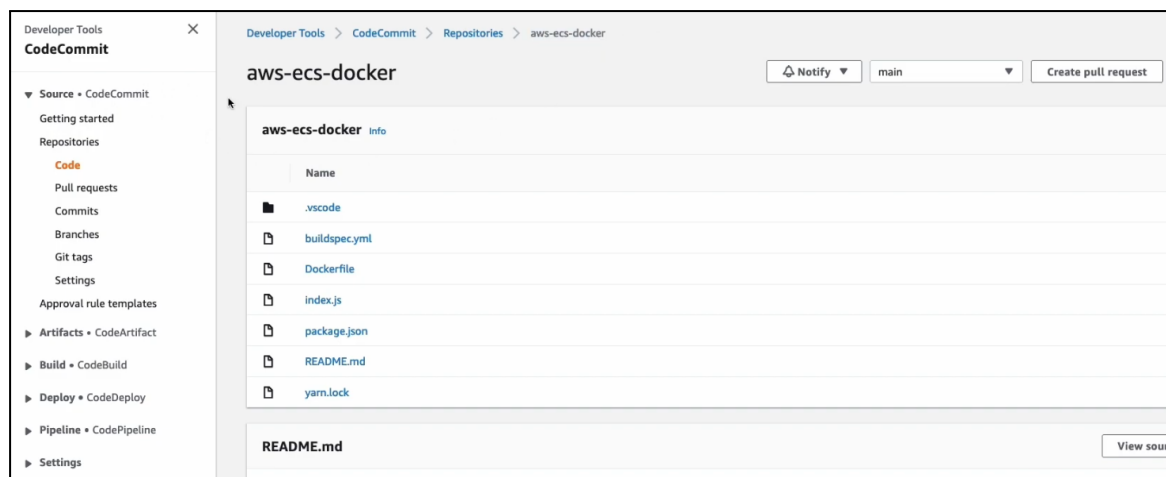
Task 1: Setup Code Commit Repository

1. Create a **CodeCommit** repository named 'aws-ecs-docker'

The screenshot shows the 'Create repository' page in the AWS CodeCommit console. The 'Repository name' field is highlighted with a red box and contains the text 'aws-ecs-docker'. Below it, the 'Description - optional' field also contains 'aws-ecs-docker'. The 'Tags' section has an 'Add' button. At the bottom, there is a checkbox for 'Enable Amazon CodeGuru Reviewer for Java and Python - optional'.

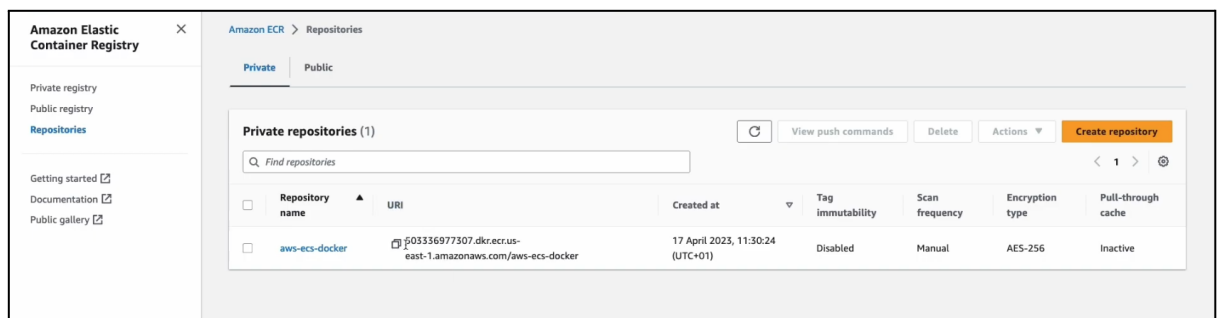
2. Go to **IAM** console and update permissions for 'AWSCodeCommitAccess'
3. Clone the repository to my local machine.
4. Push Source Code to the 'aws-ecs-docker' local repository.

5. Commit the source to the remote 'aws-ecs-docker' repository on CodeCommit.



Task 2: Setup Docker Image on Amazon ECR

1. Create a new repository on AWS ECR named 'aws-ecs-docker'



2. Update 'Buildspec.yml' from the 'aws-ecs-docker' repository on CodeCommit with appropriate repository URL, so it can build the code and create a docker image to be stored in the 'aws-ecs-docker' repository on ECR.

Task 3: Configure Build Project on AWS CodeBuild

1. Create a new Build Project on **CodeBuild** named '**aws-ecs-docker-build**'
 - a. Add 'AWS CodeCommit' as the source provider in the 'Source' section, select the 'aws-ecs-docker' repository and select 'main' branch.
 - b. Under the 'Environment' section, select the 'Managed Image' option.
 - i. Choose Amazon Linux 2 with standard runtime and latest image.
 - ii. Choose the existing 'CodeBuildServiceRole'.
 - c. Select 'Use a buildspec file' option in the 'Buildspec' section.
 - d. In the 'Logs' section, choose the 'CloudWatch Logs' option and name it as '**aws-ecs-docker-logs**'.

2. Now, create the Build Project and start building to validate the build stage.

Build started
You have successfully started the following build: aws-ecs-docker-build:d53b3bf2-995a-4100-8c2c-32b7dc66410d

Developer Tools > CodeBuild > Build projects > aws-ecs-docker-build > aws-ecs-docker-build:d53b3bf2-995a-4100-8c2c-32b7dc66410d

aws-ecs-docker-build:d53b3bf2-995a-4100-8c2c-32b7dc66410d [Stop build] [Retry build]

Build status

Status	Initiator	Build ARN	Resolved source version
In progress	bobby	arn:aws:codebuild:us-east-1:503336977507:build/aws-ecs-docker-build:d53b3bf2-995a-4100-8c2c-32b7dc66410d	-

Start time: Apr 17, 2023 11:37 AM (UTC+1:00) | End time: - | Build number: 1

Build logs | **Phase details** | Reports | Environment variables | Build details | Resource utilization

Name	Status	Context	Duration	Start time	End time
SUBMITTED	Succeeded	-	<1 sec	Apr 17, 2023 11:37 AM (UTC+1:00)	Apr 17, 2023 11:37 AM (UTC+1:00)
QUEUED	-	-	-	Apr 17, 2023 11:37 AM (UTC+1:00)	-

Task 4: Configure Core Infrastructure using AWS CloudFormation

1. Create CloudFormation stack using the following command according to the 'core-infrastructure-setup.yml' file in the 'aws-cloudformation' folder. Next, validate the stack 'ecs-core-infrastructure' from the AWS CloudFormation.

```
$ aws cloudformation create-stack --capabilities  
CAPABILITY_IAM --stack-name ecs-core-infrastructure  
--template-body file://./core-infrastructure-setup.yml
```

CloudFormation > Stacks > ecs-core-infrastructure

Stacks (1) [Filter by stack name] [Active] [View nested]

Stacks

- ecs-core-infrastructure
2023-04-17 11:55:12 UTC+0100
CREATE_IN_PROGRESS

ecs-core-infrastructure [Delete] [Update] [Stack actions] [Create stack]

Stack info | Events | **Resources** | Outputs | Parameters | Template | Change sets

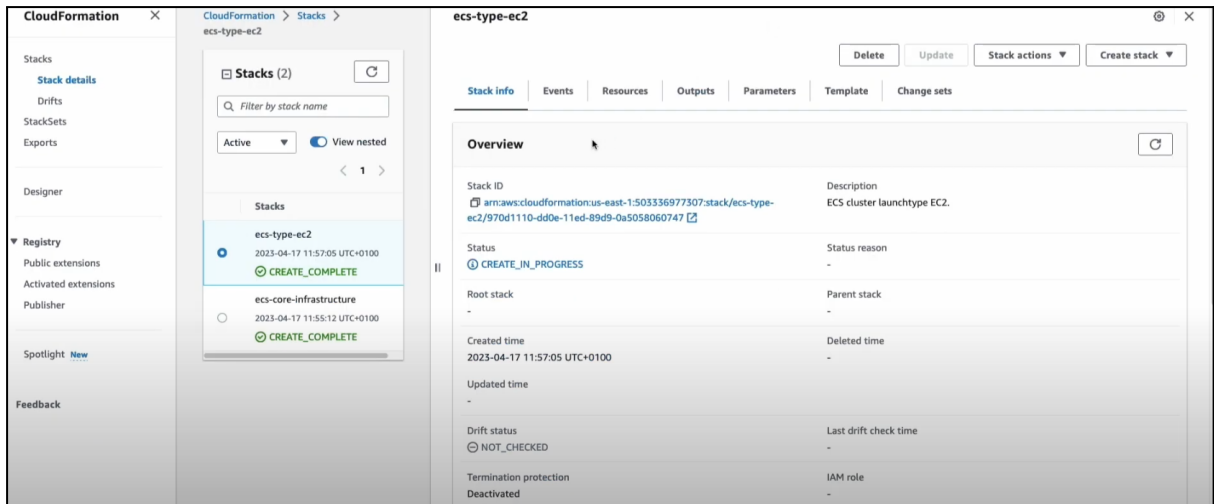
Resources (2) [Search resources]

Logical ID	Physical ID	Type	Status	Module
InternetGateway	igw-0a2cdd587f83b8cde	AWS::EC2::InternetGateway	CREATE_IN_PROGRESS	-
VPC	vpc-0575e3e74cd5377f9	AWS::EC2::VPC	CREATE_COMPLETE	-

2. Create an ECS cluster based on EC2, by using the following command to create a cluster according to the 'ecs-ec2-with-cf.yml' file in the 'aws-cloudformation' folder. Then, validate the 'ecs-type-ec2' stack is active on AWS CloudFormation.

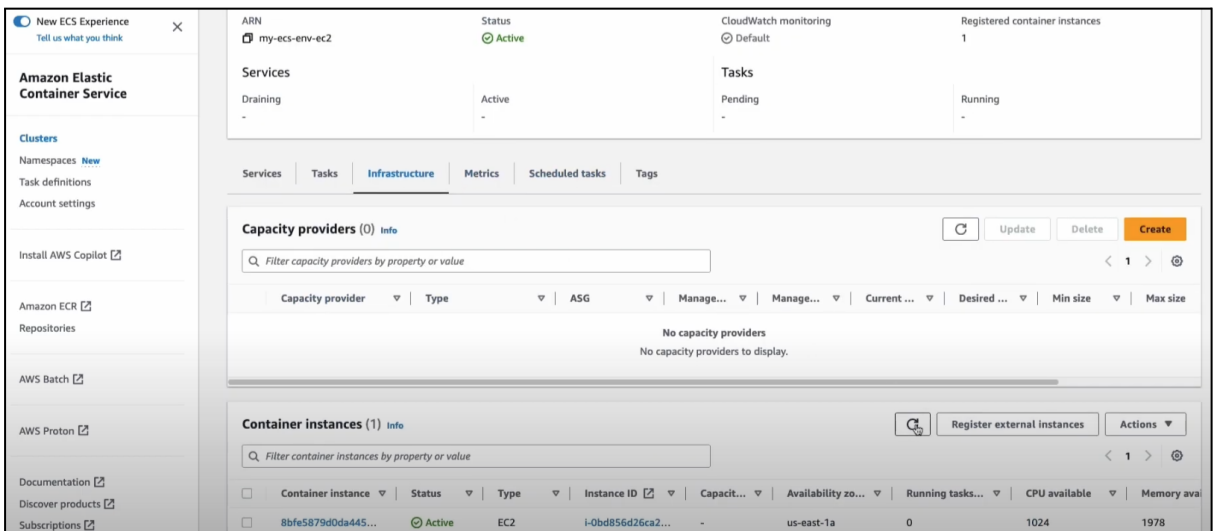
```
$ aws cloudformation create-stack --stack-name  
ecs-type-ec2 --capabilities CAPABILITY_IAM  
--template-body file://./ecs-ec2-with-cf.yml
```

- Next, Go to the Amazon ECS to validate that the **'my-ecs-env-ec2'** cluster and corresponding container instances are active.



Task 5: Configure Task Definition on Amazon ECS

- Create new Task definition configuration named **'exp-aws-ecs-docker'**
 - Specify container name as **'exp-code-pipeline'** according to the buildspec file.
 - Use the ECR **'aws-ecs-docker'** repository URI for the Image URI of the container.
 - Use Container port 3000
 - Next, select **'Amazon EC2 instance'** as the app environment.
 - Select task role **'ecsTaskExecutionRole'**
- Save the task definition file and validate if the task is running.



Task 6: Orchestrate CodePipeline

- Go to CodePipeline and create a pipeline
 - In the **'Choose Pipeline settings'** name the pipeline as **'aws-ecs-docker-pipeline'** and choose a service role.
 - In the **'Add source stage'**, select AWS CodeCommit and **'aws-ecs-docker'** repository as the source provider.

- c. In the 'Add build stage', select AWS CodeBuild and 'aws-ecs-docker-build' followed by a single build option.
 - d. In the 'Add deploy stage', select AWS CodeDeploy and Amazon ECS as the deploy provider.
 - e. Next, choose 'aws-ecs-env-ec2' cluster, 'exp-aws-ecs-docker-pipeline' as the service, and 'imagedefinitions.json' as image definition file.
2. Save the pipeline settings and create the pipeline. Validate if the pipeline build is successful.

