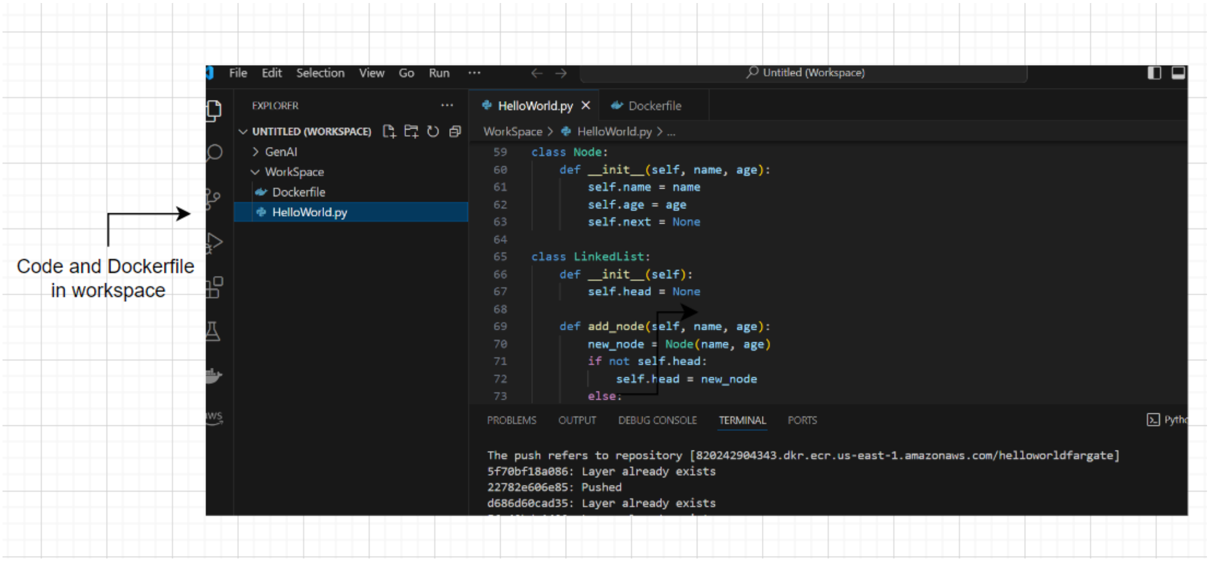
1. Install VSCode, AWS Cli and Docker in local system and set the PATH accordingly.
2. Install aws cli from [Install or update to the latest version of the AWS CLI - AWS Command Line Interface (amazon.com)]()
3. Create a main python file and Dockerfile
    a. A python file is created which is creating a LinkedList. The python code will be deployed in docker container. The image will be deployed in Amazon Elastic Container Registry. The container will be deployed in AWS ECS(Fargate).
    b. The python code will traverse a LinkedList and print in S3 bucket
    c. The actual code can replace the main code which will be fault tolerant. Actual code can be of type reporting, calculation, reconciliation etc.

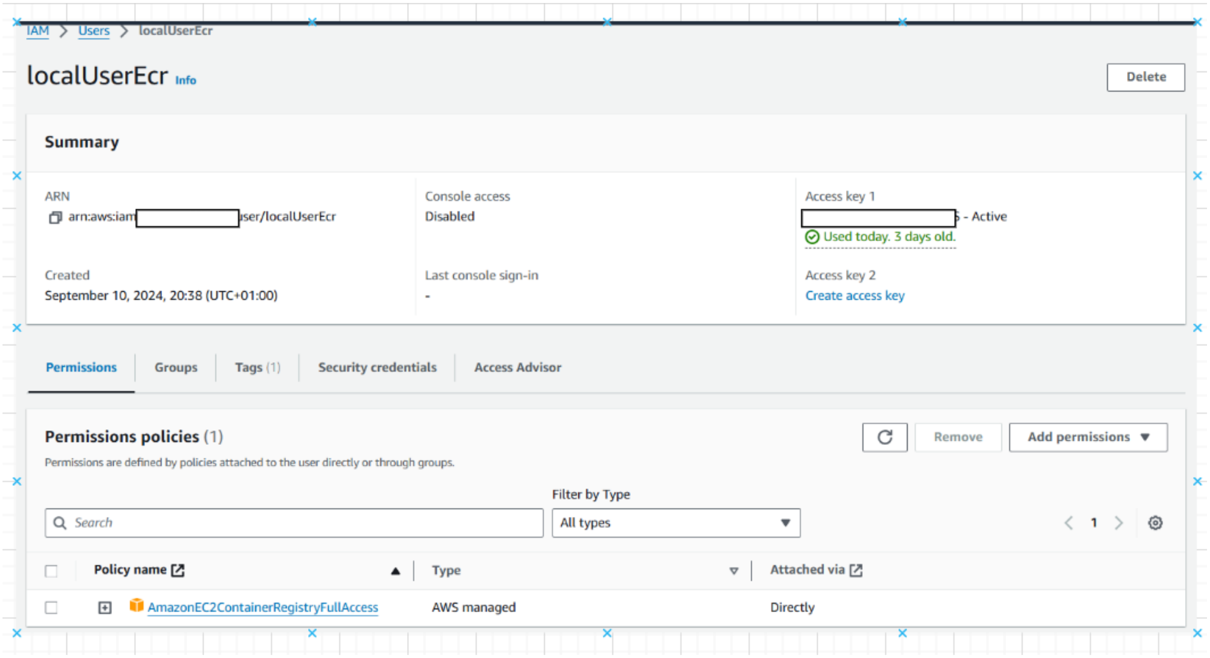

Code and Dockerfile in workspace
File Edit Selection View Go Run ···
Untitled (Workspace)
EXPLORER
UNTITLED (WORKSPACE)
GenAI
WorkSpace
Dockerfile
HelloWorld.py
HelloWorld.py
Dockerfile
WorkSpace > HelloWorld.py > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```python
class Node:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def add_node(self, name, age):
        new_node = Node(name, age)
        if not self.head:
            self.head = new_node
        else:
```

```
The push refers to repository [820242904343.dkr.ecr.us-east-1.amazonaws.com/helloworldfargate]
5f70bf18a086: Layer already exists
22782e606e85: Pushed
d686d60cad35: Layer already exists
```

Image 1 – Dockerfile and Python code to print

4. Creation of AWS IAM role and authentication key
    a. Creation of Local User is required in AWS IAM which will be used in VSCode for Docker push to Amazon ECR.



localUserEcr

Summary

ARN arn:aws:iam[redacted]user/localUserEcr	Console access Disabled	Access key 1 [redacted] - Active Used today. 3 days old.
Created September 10, 2024, 20:38 (UTC+01:00)	Last console sign-in -	Access key 2 Create access key

Permissions policies (1)

Permissions are defined by policies attached to the user directly or through groups.

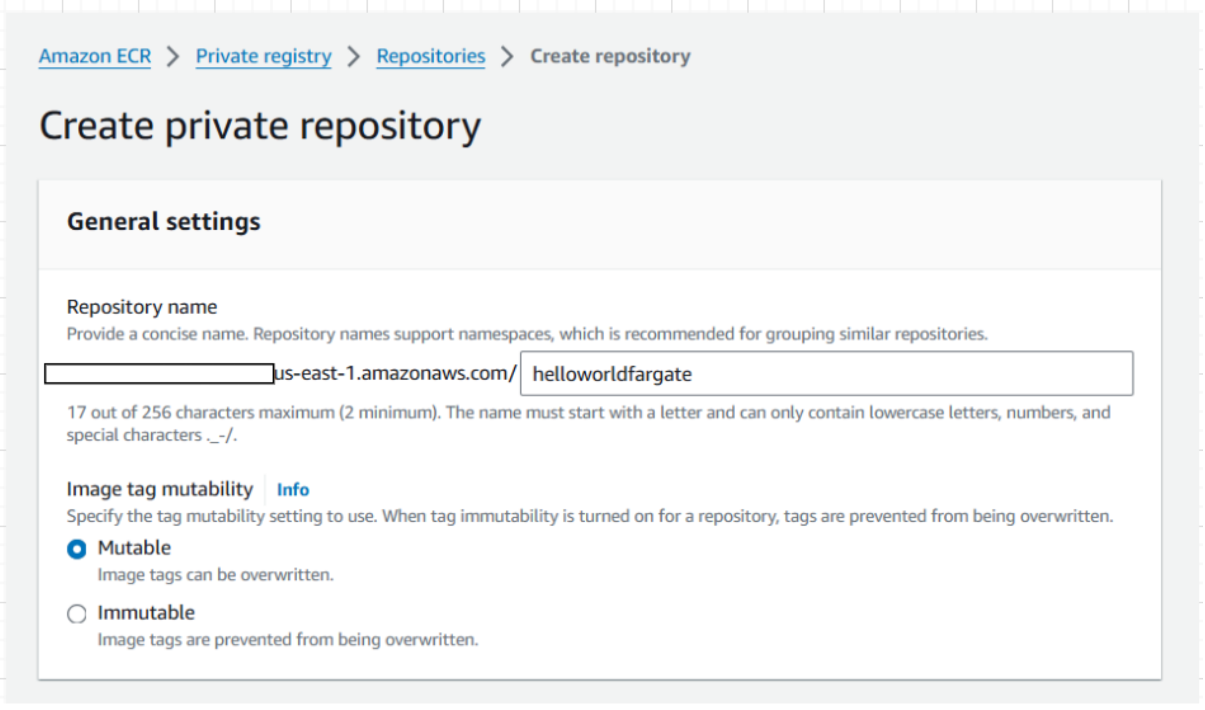
Filter by Type: All types

Policy name	Type	Attached via
AmazonEC2ContainerRegistryFullAccess	AWS managed	Directly

Image 2 – UserRole is created in AWS IAM

   b. The role must be given the policy of AWS managed policy [AmazonEC2ContainerRegistryFullAccess]
   c. Create a Access Key which will be required to login AWS from VSCode

5. Amazon ECR Repository creation
   a. Go to Amazon ECR and create a repository. The repository will be used from VSCode.



Amazon ECR > Private registry > Repositories > Create repository

Create private repository

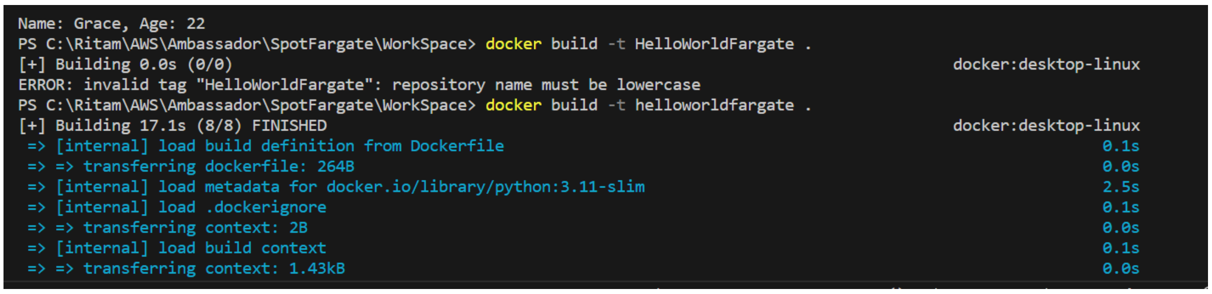
General settings

Repository name
Provide a concise name. Repository names support namespaces, which is recommended for grouping similar repositories.
[redacted]us-east-1.amazonaws.com/ helloworldfargate
17 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, and special characters ._-/.

Image tag mutability Info
Specify the tag mutability setting to use. When tag immutability is turned on for a repository, tags are prevented from being overwritten.
[✓ Mutable]
Image tags can be overwritten.
Immutable
Image tags are prevented from being overwritten.

Image 3 – Repository creation from Amazon ECR

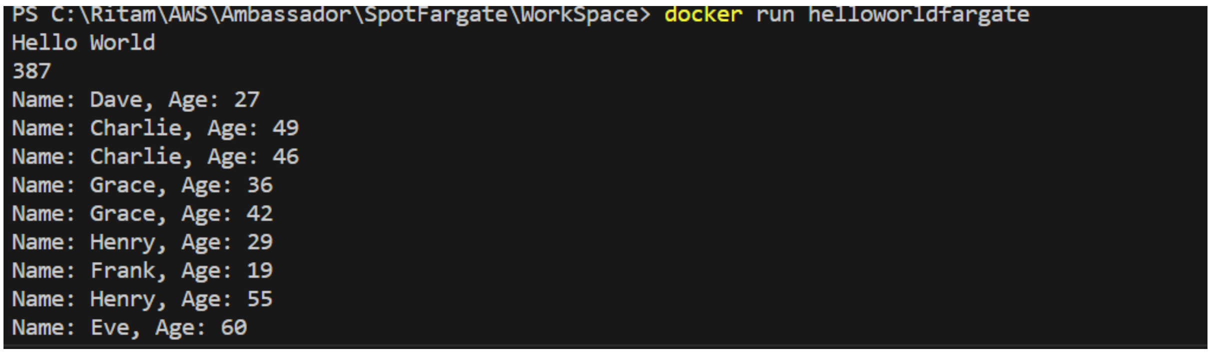
6. Docker File build and push to Amazon ECR
   a. Run the command *docker build -t helloworldfargate .*



```
Name: Grace, Age: 22
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> docker build -t HelloWorldFargate .
[+] Building 0.0s (0/0)                                                              docker:desktop-linux
ERROR: invalid tag "HelloWorldFargate": repository name must be lowercase
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> docker build -t helloworldfargate .
[+] Building 17.1s (8/8) FINISHED                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
 => => transferring dockerfile: 264B                                                                 0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                  2.5s
 => [internal] load .dockerignore                                                                    0.1s
 => => transferring context: 2B                                                                      0.0s
 => [internal] load build context                                                                    0.1s
 => => transferring context: 1.43kB                                                                  0.0s
```

Image created
naming to docker.io/library/helloworldfargate

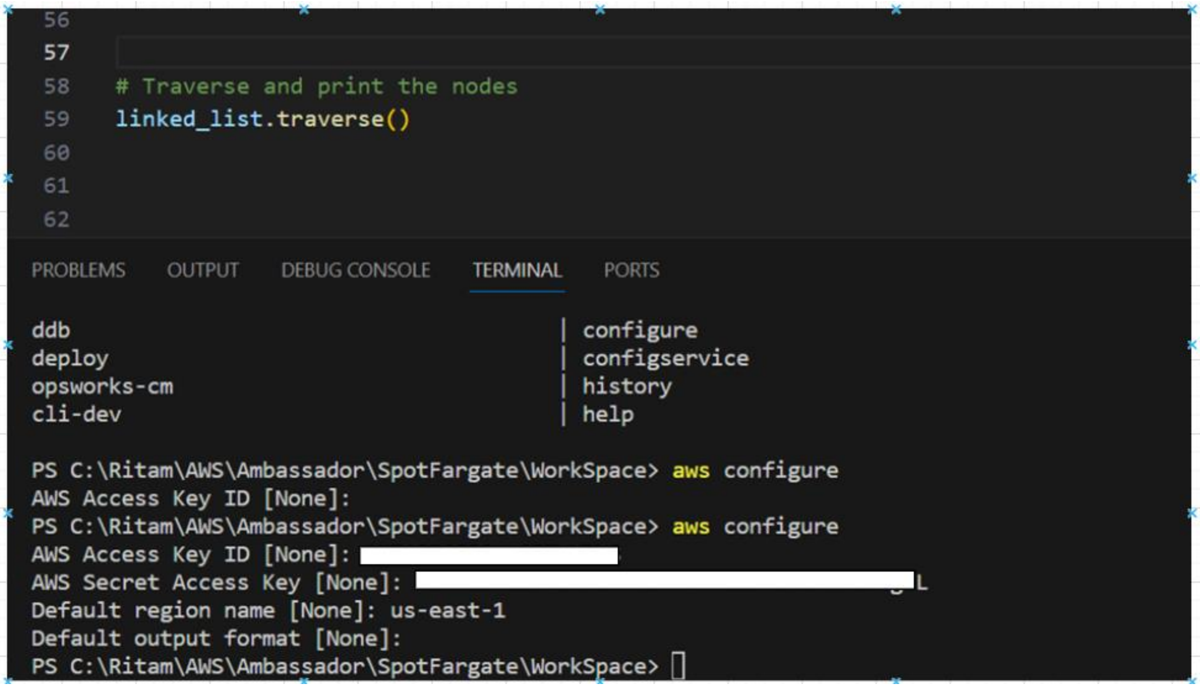
   b. Unit test the code by running docker from local *docker run helloworldfargate*



```
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> docker run helloworldfargate
Hello World
387
Name: Dave, Age: 27
Name: Charlie, Age: 49
Name: Charlie, Age: 46
Name: Grace, Age: 36
Name: Grace, Age: 42
Name: Henry, Age: 29
Name: Frank, Age: 19
Name: Henry, Age: 55
Name: Eve, Age: 60
```

c. Use aws cli command from powershell to login

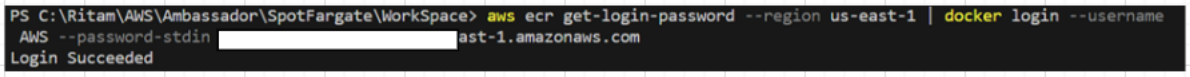
   Command – *aws configure*



```python

# Traverse and print the nodes
linked_list.traverse()
```

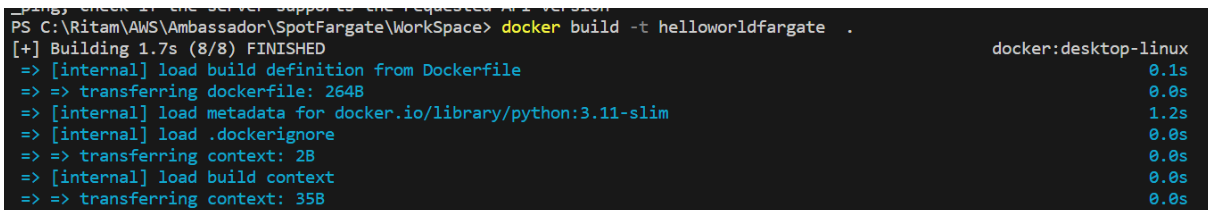
```
ddb                                         | configure
deploy                                      | configservice
opsworks-cm                                 | history
cli-dev                                     | help

PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> aws configure
AWS Access Key ID [None]:
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]L
Default region name [None]: us-east-1
Default output format [None]:
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> 
```

d. Retrieve an authentication token and authenticate your Docker client to your registry.
   Use the AWS CLI:
   aws ecr get-login-password --region us-east-1 | docker login --username AWS --
   password-stdin <AMAZON ECR repository URI>



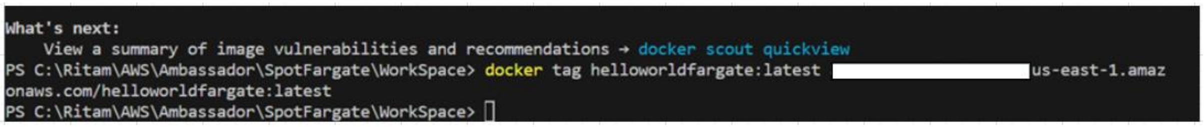
```
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> aws ecr get-login-password --region us-east-1 | docker login --username
 AWS --password-stdin [redacted]ast-1.amazonaws.com
Login Succeeded
```

e. Build your Docker image using the following command.
      docker build -t helloworldfargate .



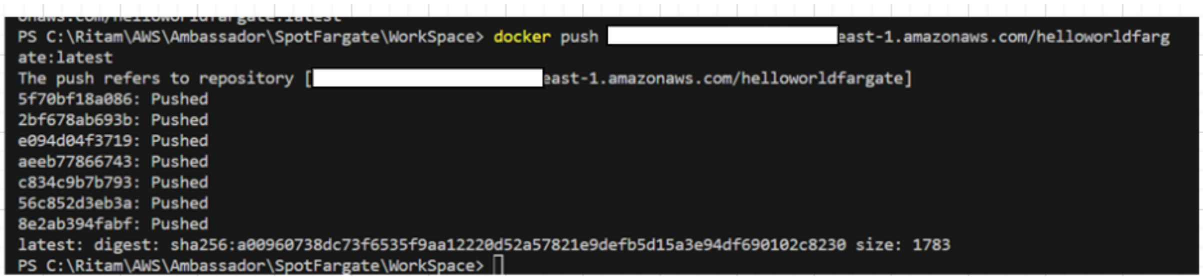
```
_ping, check if the server supports the requested API version
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> docker build -t helloworldfargate  .
[+] Building 1.7s (8/8) FINISHED                                                                   docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                              0.1s
 => => transferring dockerfile: 264B                                                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                               1.2s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [internal] load build context                                                                                 0.0s
 => => transferring context: 35B                                                                                  0.0s
```

f. After the build completes, tag your image so you can push the image to this repository: docker tag helloworldfargate:latest <Amazon ECR URI>/helloworldfargate:latest



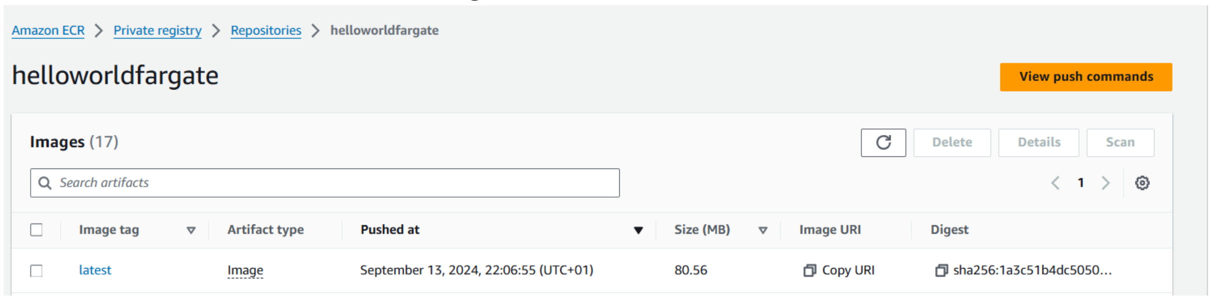
```
What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> docker tag helloworldfargate:latest [redacted]us-east-1.amaz
onaws.com/helloworldfargate:latest
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> 
```

g. Run the following command to push this image to your newly created AWS repository:
docker push < Amazon ECR URI >/helloworldfargate:latest



```
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> docker push [redacted]east-1.amazonaws.com/helloworldfarg
ate:latest
The push refers to repository [[redacted]east-1.amazonaws.com/helloworldfargate]
5f70bf18a086: Pushed
2bf678ab693b: Pushed
e094d04f3719: Pushed
aeeb77866743: Pushed
c834c9b7b793: Pushed
56c852d3eb3a: Pushed
8e2ab394fabf: Pushed
latest: digest: sha256:a00960738dc73f6535f9aa12220d52a57821e9defb5d15a3e94df690102c8230 size: 1783
PS C:\Ritam\AWS\Ambassador\SpotFargate\WorkSpace> 
```

h. Validate in Amazon ECR the docker image is created with latest time



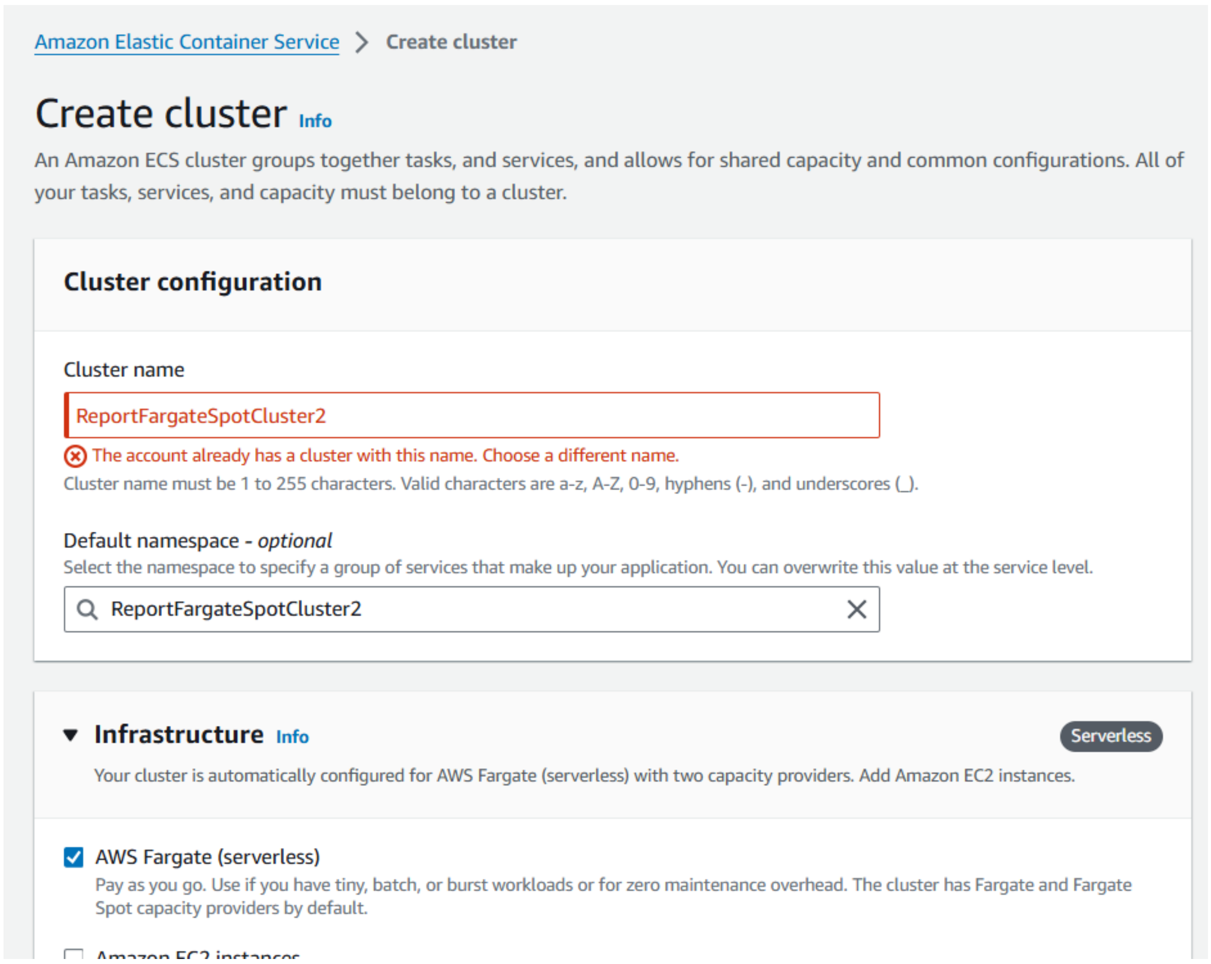
Amazon ECR > Private registry > Repositories > helloworldfargate
View push commands
Delete
Details
Scan
Search artifacts

helloworldfargate

Images (17)

	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
☐	latest	Image	September 13, 2024, 22:06:55 (UTC+01)	80.56	Copy URI	sha256:1a3c51b4dc5050...

7. Amazon ECS Cluster creation
One Amazon ECS cluster is required where the Fault tolerant docker with python code will execute.


Amazon Elastic Container Service > Create cluster

Create cluster Info

An Amazon ECS cluster groups together tasks, and services, and allows for shared capacity and common configurations. All of your tasks, services, and capacity must belong to a cluster.

Cluster configuration

Cluster name

ReportFargateSpotCluster2

The account already has a cluster with this name. Choose a different name.

Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Default namespace - *optional*

Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.

ReportFargateSpotCluster2

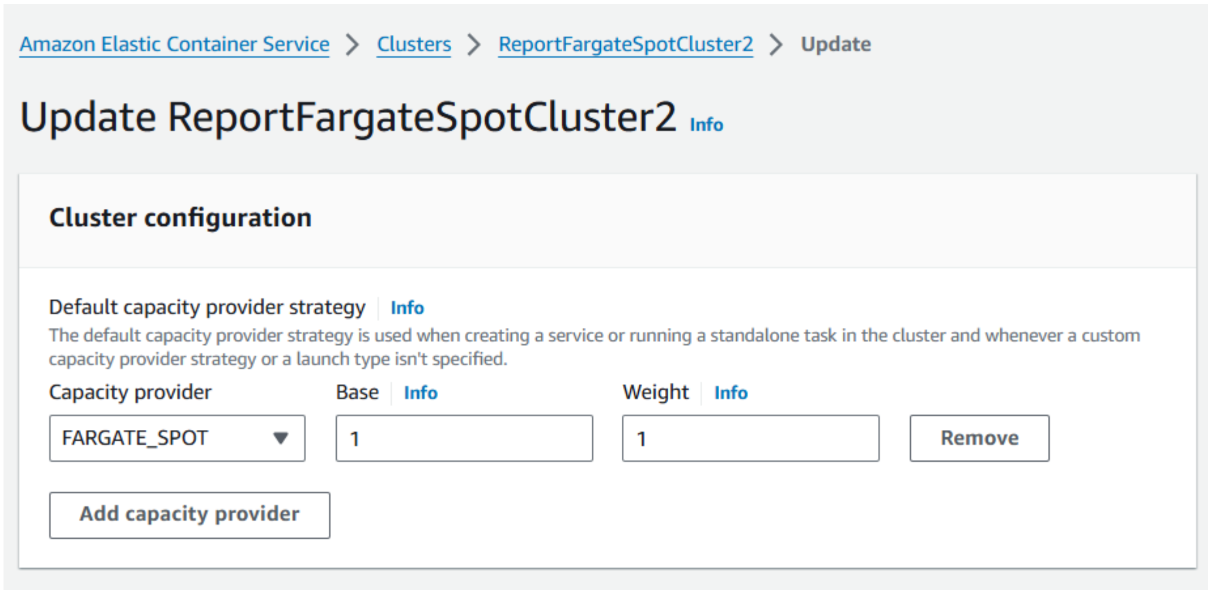
Infrastructure Info Serverless

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances.

☑ AWS Fargate (serverless)

Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

8. Amazon ECS Cluster update with default capacity provider
   The purpose the exercise is to run the Docker container to run in AWS ECS(Fargate). This can be done in two ways. We shall update the created Amazon ECS Cluster to make default as spot.
   Here the base and weight given 1 and 1 which will make any task running on the cluster by default on spot only.


Amazon Elastic Container Service > Clusters > ReportFargateSpotCluster2 > Update

Update ReportFargateSpotCluster2 Info

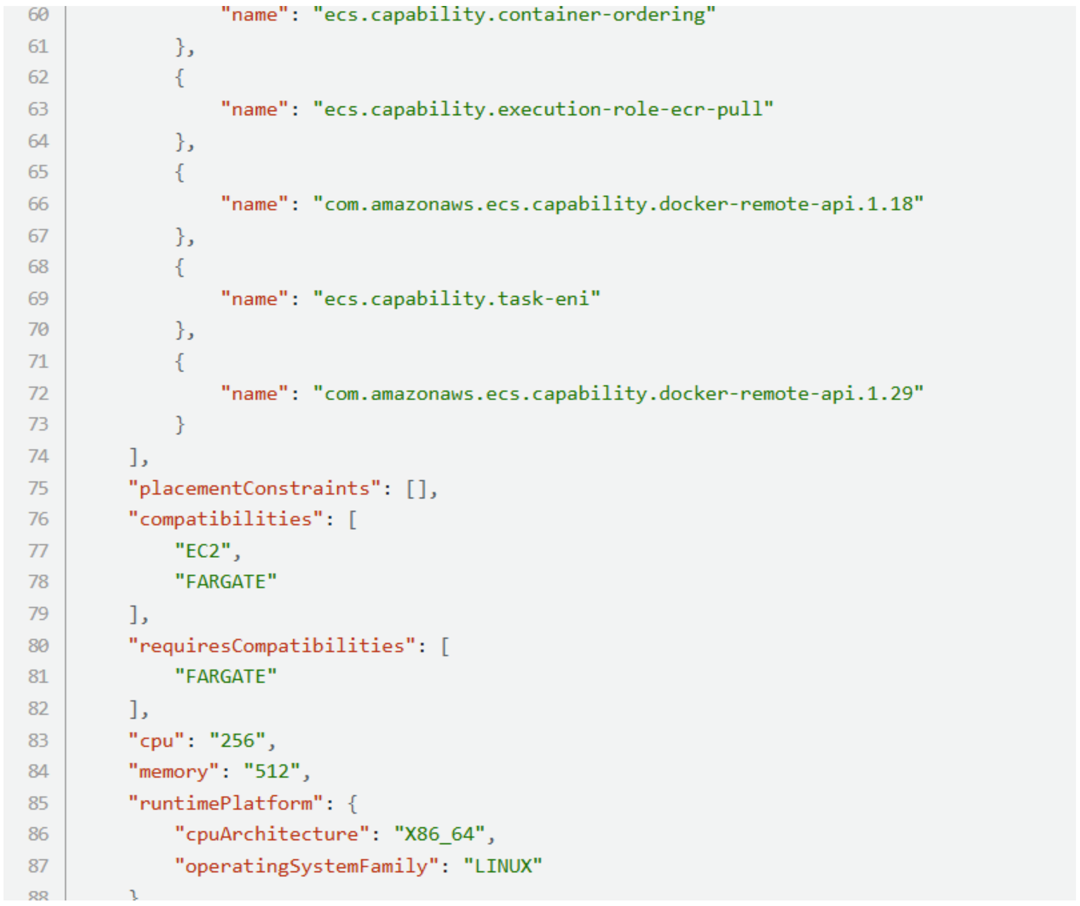
Cluster configuration

Default capacity provider strategy Info

The default capacity provider strategy is used when creating a service or running a standalone task in the cluster and whenever a custom capacity provider strategy or a launch type isn't specified.

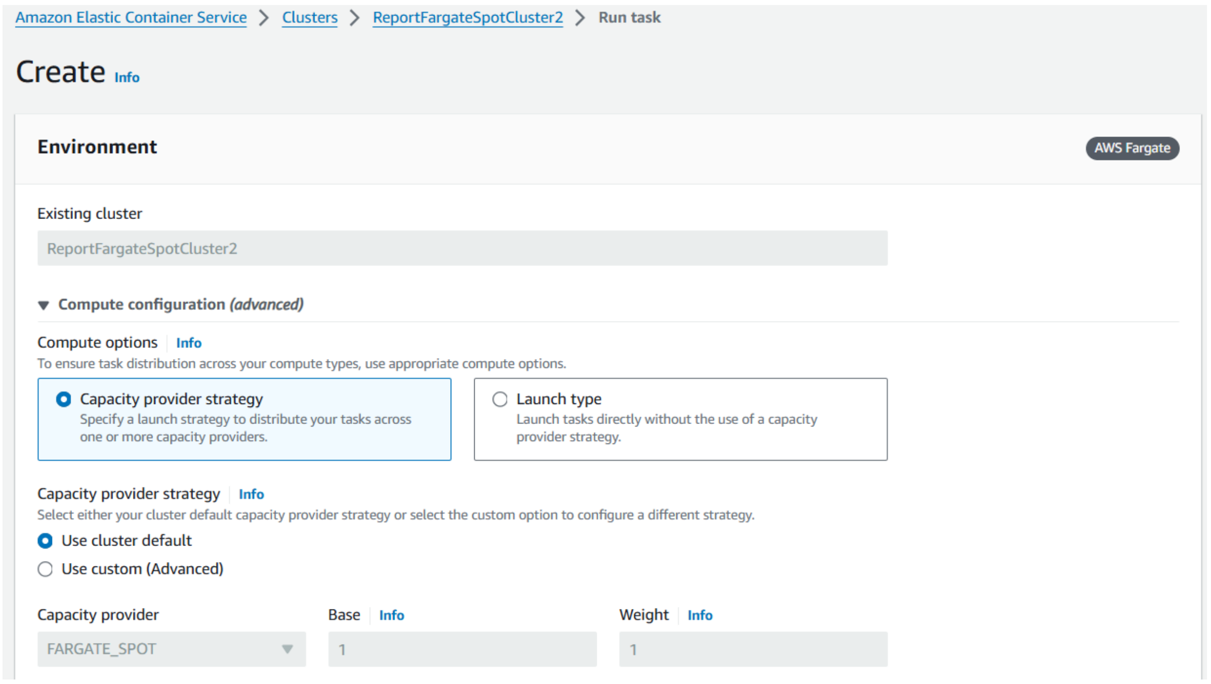
Capacity provider	Base	Weight	
FARGATE_SPOT	1	1	Remove

9. AWS ECS Fargate Task Definition
   We have already created Amazon ECR and AWS ECS Cluster. The AWS ECS Cluster is updated with default capacity provider. Now we shall define Standalone AWS ECS Fargate task that will run with task count 1. This will be our fault tolerant task. The task definition is attached herewith.



```
            "name": "ecs.capability.container-ordering"
        },
        {
            "name": "ecs.capability.execution-role-ecr-pull"
        },
        {
            "name": "com.amazonaws.ecs.capability.docker-remote-api.1.18"
        },
        {
            "name": "ecs.capability.task-eni"
        },
        {
            "name": "com.amazonaws.ecs.capability.docker-remote-api.1.29"
        }
    ],
    "placementConstraints": [],
    "compatibilities": [
        "EC2",
        "FARGATE"
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "256",
    "memory": "512",
    "runtimePlatform": {
        "cpuArchitecture": "X86_64",
        "operatingSystemFamily": "LINUX"
    }
```

10. How to do Unit Testing
    In this step we shall run the task manually and check the Amazon CloudWatch loggroup.
    a. Go to Amazon ECS cluster.
    b. Go to tasks tab
    c. Run new task
    d. Select default compute option. Here it is cluster default.


Amazon Elastic Container Service > Clusters > ReportFargateSpotCluster2 > Run task

Create Info

Environment

AWS Fargate

Existing cluster

ReportFargateSpotCluster2

▼ 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

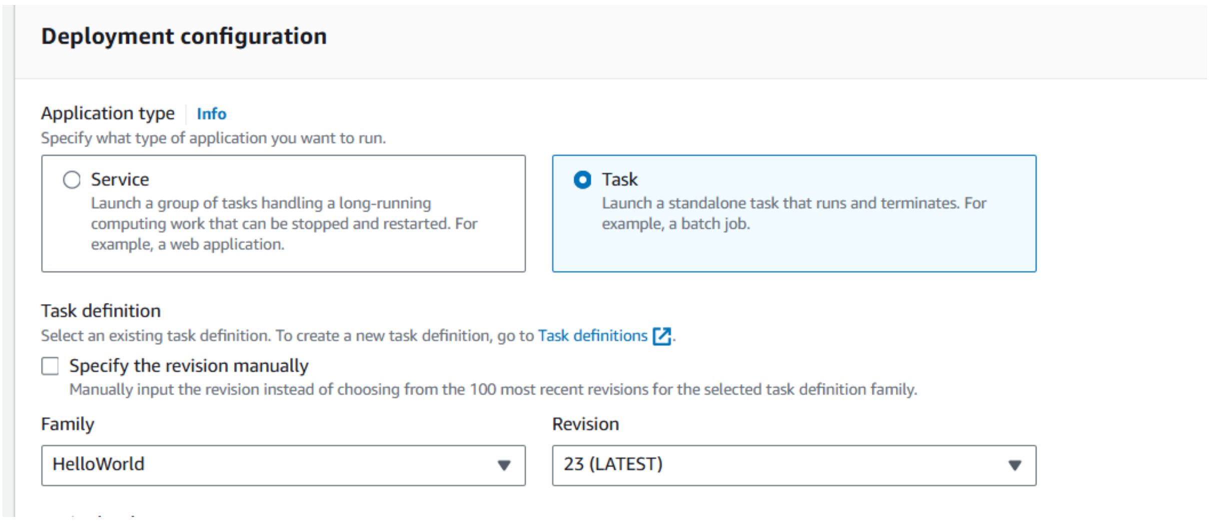
Select either your cluster default capacity provider strategy or select the custom option to configure a different strategy.

● Use cluster default

○ Use custom (Advanced)

Capacity provider	Base	Weight
FARGATE_SPOT	1	1

e. Select the latest task definition which is pointing to Amazon ECR URI created earlier. Select application type as Task as our program is for standalone fault tolerant task.


Deployment configuration

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.

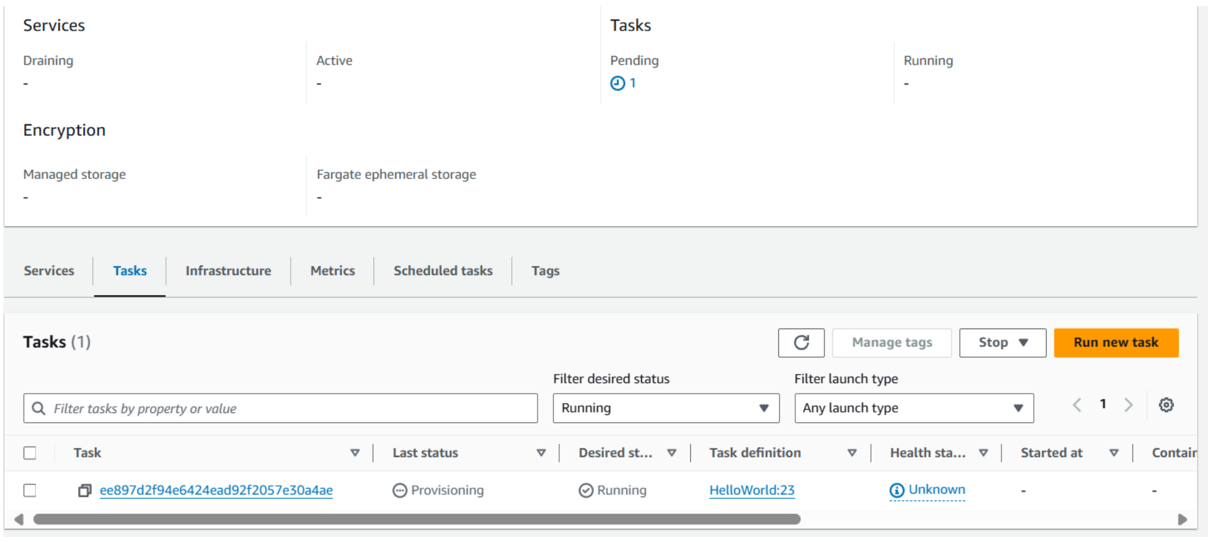
Task definition

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
HelloWorld	23 (LATEST)

f. Follow the Amazon ECS cluster task status. The status will be provisioning -> pending -> running. Follow all task status changes.



Services

Draining	Active
-	-

Tasks

Pending	Running
1	-

Encryption

Managed storage	Fargate ephemeral storage
-	-

Services | Tasks | Infrastructure | Metrics | Scheduled tasks | Tags

Tasks (1)

Manage tags | Stop | Run new task

Filter desired status: Running

Filter launch type: Any launch type

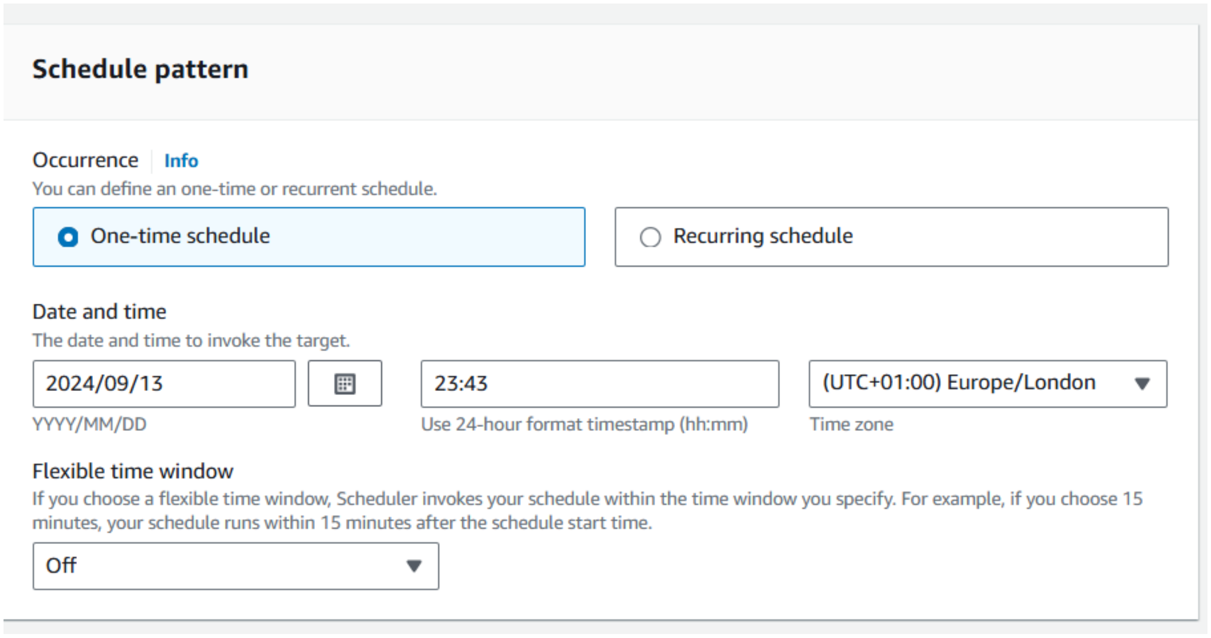
Task	Last status	Desired st...	Task definition	Health sta...	Started at	Contair
ee897d2f94e6424ead92f2057e30a4ae	Provisioning	Running	HelloWorld:23	Unknown	-	-

g. Once the task is completed it will automatically get killed and validate Amazon CloudWatch loggroup.

11. Amazon EventBridge Scheduler creation and testing

 The task we have created above will be implemented through Amazon EventBridge Scheduler.

 a. Define the scheduler pattern as per requirement. We have selected one-time schedule for validation.



Schedule pattern

Occurrence | Info

You can define an one-time or recurrent schedule.

[✓ One-time schedule] Recurring schedule

Date and time

The date and time to invoke the target.

2024/09/13 (YYYY/MM/DD)

23:43 (Use 24-hour format timestamp (hh:mm))

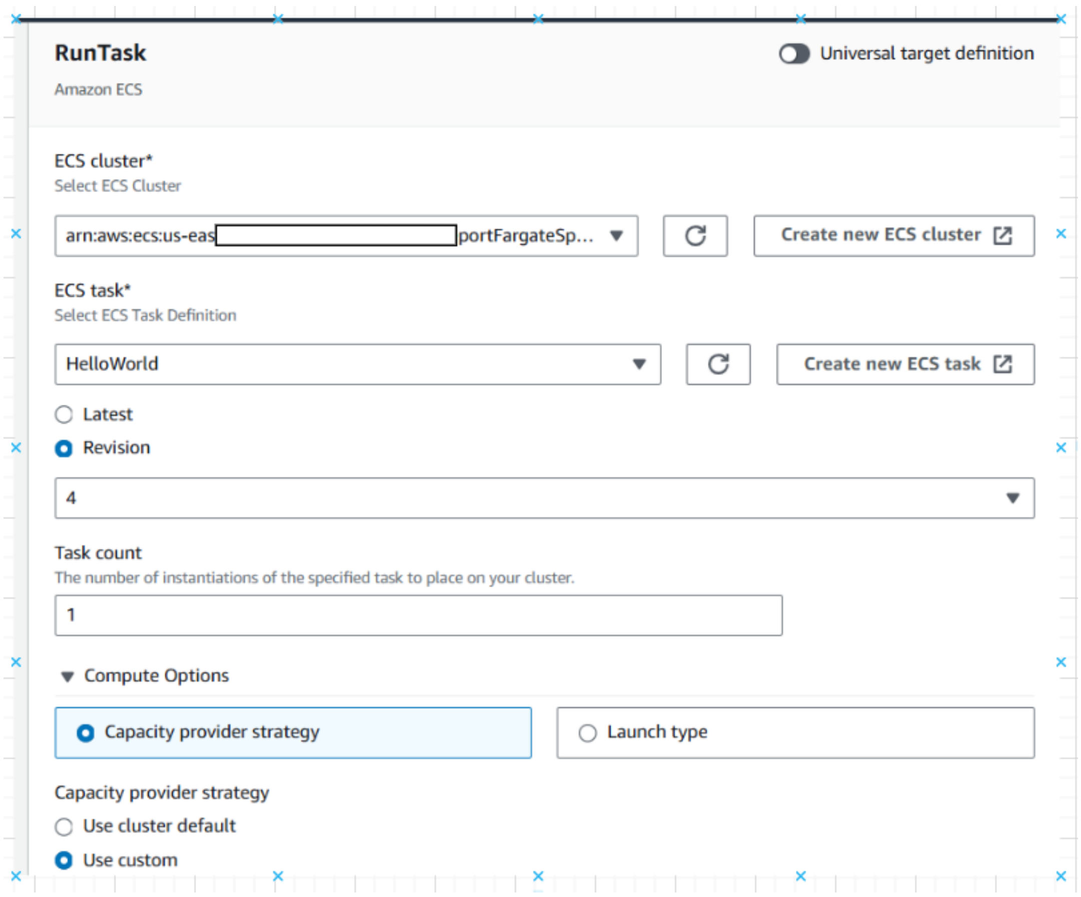
(UTC+01:00) Europe/London (Time zone)

Flexible time window

If you choose a flexible time window, Scheduler invokes your schedule within the time window you specify. For example, if you choose 15 minutes, your schedule runs within 15 minutes after the schedule start time.

Off

b. Select Amazon ECS Task. Select Amazon ECS Cluster, task name, version and task count as 1.


RunTask
Amazon ECS
Universal target definition

ECS cluster*
Select ECS Cluster
arn:aws:ecs:us-eas[redacted]portFargateSp...

ECS task*
Select ECS Task Definition
HelloWorld

Latest [✓ Revision]

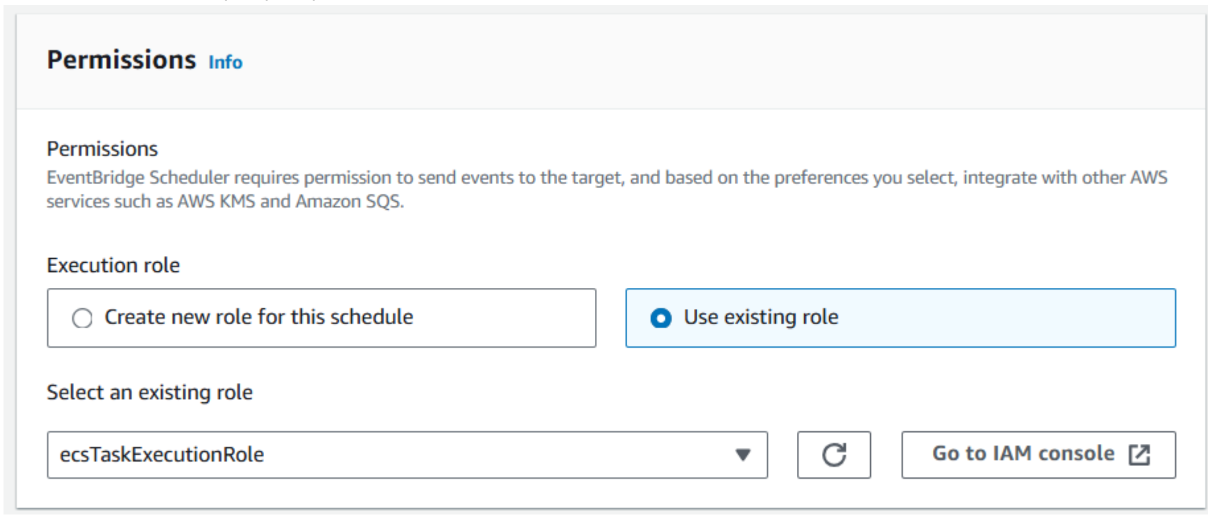
4

Task count
The number of instantiations of the specified task to place on your cluster.
1

Compute Options
[✓ Capacity provider strategy] Launch type

Capacity provider strategy
Use cluster default [✓ Use custom]

c. Give the role with proper policies.


Permissions Info

Permissions
EventBridge Scheduler requires permission to send events to the target, and based on the preferences you select, integrate with other AWS services such as AWS KMS and Amazon SQS.

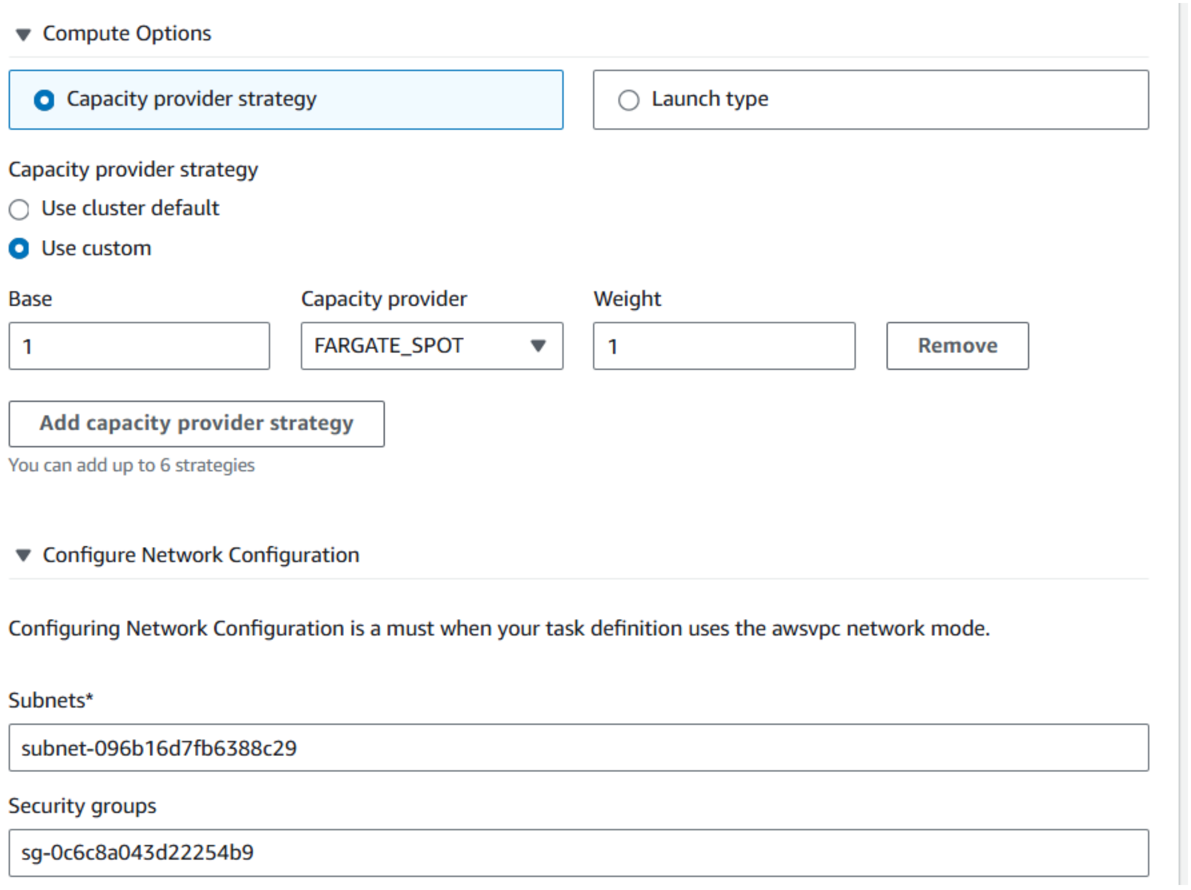
Execution role
Create new role for this schedule [✓ Use existing role]

Select an existing role
ecsTaskExecutionRole

d. Validate from Amazon ECS Cluster task and Amazon CloudWatch loggroup that the task has been properly triggered and validated.

12. Amazon ECS Default capacity provider
The AWS ECS Fargate task can run in two possibilities. If the Amazon ECS cluster is defined with default capacity provider (with spot) and while launching the task default cluster definition is used, the launched task will run on default capacity provider.

13. Amazon EventBridge capacity provider
    The Amazon ECS Cluster can be created with both capacity provider Fargate and Fargate_Spot. While launching the task instead of selected default cluster, launch template the custom capacity provider can be selected.



▼ Compute Options

[✓ Capacity provider strategy] Launch type

Capacity provider strategy

Use cluster default [✓ Use custom]

Base	Capacity provider	Weight	
1	FARGATE_SPOT	1	Remove

Add capacity provider strategy

You can add up to 6 strategies

▼ Configure Network Configuration

Configuring Network Configuration is a must when your task definition uses the awsvpc network mode.

Subnets*

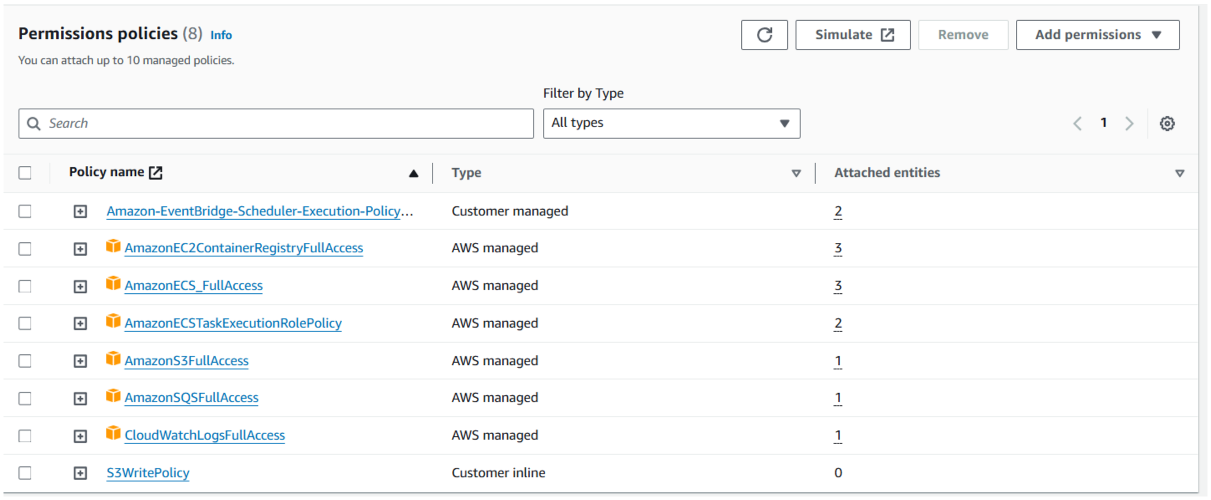
subnet-096b16d7fb6388c29

Security groups

sg-0c6c8a043d22254b9

14. AWS IAM role for execution
    The Amazon ECS Task has to be given proper role for execution. The Amazon EventBridge also has to given same role to pass on.

Here the role is **ecsTaskExecutionRole** with customed and AWS Managed roles. Also the role requires TrustPolicies which is added in attachment.



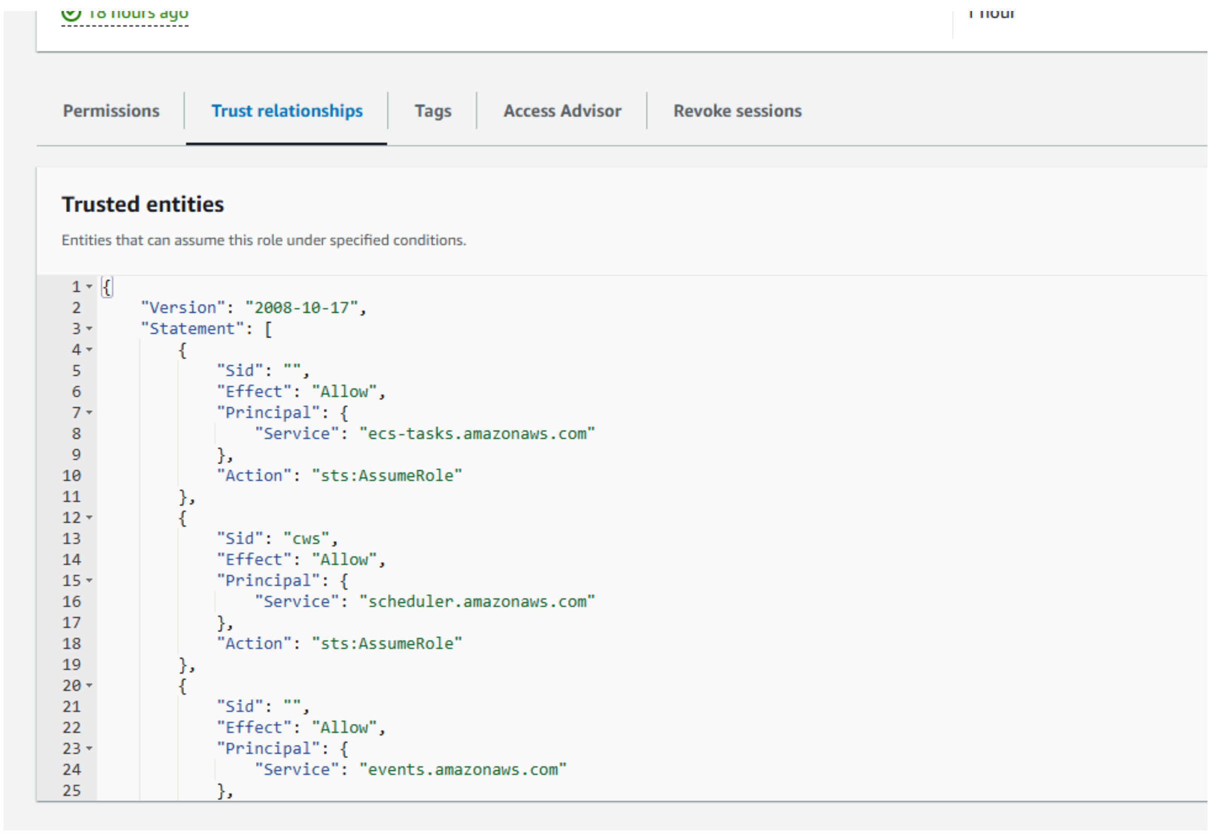
Permissions policies (8) Info

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
Amazon-EventBridge-Scheduler-Execution-Policy...	Customer managed	2
AmazonEC2ContainerRegistryFullAccess	AWS managed	3
AmazonECS_FullAccess	AWS managed	3
AmazonECSTaskExecutionRolePolicy	AWS managed	2
AmazonS3FullAccess	AWS managed	1
AmazonSQSFullAccess	AWS managed	1
CloudWatchLogsFullAccess	AWS managed	1
S3WritePolicy	Customer inline	0

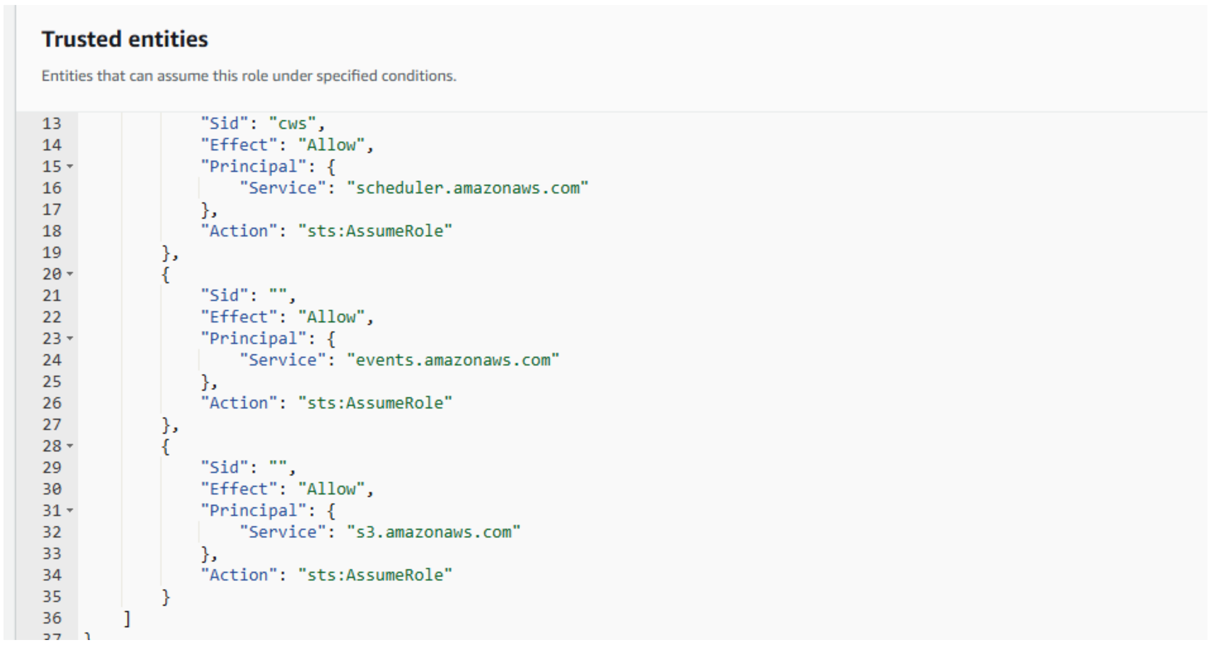
The trust policy needs to be updated.



Trusted entities

Entities that can assume this role under specified conditions.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "ecs-tasks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Sid": "cws",
            "Effect": "Allow",
            "Principal": {
                "Service": "scheduler.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
```


Trusted entities

Entities that can assume this role under specified conditions.

```
            "Sid": "cws",
            "Effect": "Allow",
            "Principal": {
                "Service": "scheduler.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

15. Clean Up