AWS ECS and Fargate

Verified 2026-11-02

Run a containerised service on Fargate — task definitions, roles, networking, ALB health checks, safe deployments, logs, secrets and scaling.

Contents

Mental model 2

Clusters, services, tasks 2

Task definitions and Fargate sizing 3

Task role vs execution role 4

Networking and load balancing 5

Deployments and the circuit breaker 6

Logs and secrets 7

Autoscaling 8

Debugging with ECS Exec 9

Further reading 10

Mental model

A **task definition** is the unit of deployment: an immutable, versioned recipe of images, CPU, memory, roles and settings. A **service** keeps N copies of one revision running, replaces any that fail health checks, and registers them with a load balancer target group. Fargate removes the servers, so almost everything left to get wrong is networking and IAM.

Clusters, services, tasks

Object	What it is
Cluster	Namespace for services and tasks
Task definition	Versioned recipe: `family:revision`
Task	One running instance of a revision
Service	Keeps `desiredCount` tasks healthy

Run a one-off job with `run-task`; run anything long-lived as a service. Every change to a task definition registers a new revision — you never edit one in place — and a service deployment moves tasks to the new revision.

```
aws ecs create-cluster --cluster-name prod
aws ecs register-task-definition \
  --cli-input-json file://taskdef.json
aws ecs update-service --cluster prod \
  --service api --task-definition api:42
```

Task definitions and Fargate sizing

CPU units	Memory
256 (.25 vCPU)	512 MiB, 1 GB, 2 GB
16384 (16 vCPU)	32–120 GB, 8 GB steps
32768 (largest)	Up to 244 GB, OS-dependent
Ephemeral storage	20 GiB default, up to 200 GiB

On Fargate, task-level `cpu` and `memory` are required and must be a valid pair from the table in the docs; memory is in MiB. Sizes of 8 vCPU and up need Linux platform version 1.4.0 or later. Fargate bills vCPU, memory and extra storage per second, from image pull to task stop, with a one-minute minimum. Fargate Spot runs interruptible tasks at up to 70% off and can stop them with a two-minute warning.

```
{
  "family": "api",
  "requiresCompatibilities": ["FARGATE"],
  "networkMode": "awsvpc",
  "cpu": "512",
  "memory": "1024",
  "runtimePlatform": {
    "cpuArchitecture": "ARM64",
    "operatingSystemFamily": "LINUX"
  },
  "containerDefinitions": [{
```

```
    "name": "api", "image": "repo/api:1.4.2",
    "portMappings": [{ "containerPort": 8080 }]
  }]
}
```

> **Gotcha:** The container image is stored in the same ephemeral storage, so a 3 GB image leaves you about 17 GiB of the 20 GiB default for `/tmp` and friends.

Task role vs execution role

Role	Used by	Needs
Execution role	ECS agent, before start	Pull image, write logs, read secrets
Task role	Your code, at runtime	S3, DynamoDB, ECS Exec channels

The **execution role** (`executionRoleArn`) lets ECS pull from ECR, send logs to CloudWatch and fetch secrets for injection. The **task role** (`taskRoleArn`) is what the AWS SDK inside your container picks up. Keep them separate so the app never holds pull-and-secret permissions it does not use.

```
{
  "executionRoleArn":
```

```
    "arn:aws:iam::123456789012:role/ecsExec",
  "taskRoleArn":
    "arn:aws:iam::123456789012:role/apiTask"
}
```

> **Gotcha:** `AccessDenied` from your app code means the **task** role. A task stuck before start with `ResourceInitializationError` usually means the **execution** role or networking.

Networking and load balancing

Setting	Rule on Fargate
`networkMode`	`awsvpc` only — one ENI per task
Target group type	`ip`, never `instance`
Public subnet	`assignPublicIp=ENABLED` to pull images
Private subnet	NAT gateway or ECR + S3 endpoints

Each task gets its own ENI, private IP and security groups. Allow the task's security group to accept the container port only from the load balancer's security group. The ALB health check hits each task's IP; a task that fails it is drained and replaced.

```
SN='subnets=[subnet-a,subnet-b]'
SG='securityGroups=[sg-api]'
IP='assignPublicIp=DISABLED'
NET="awsvpcConfiguration={$SN,$SG,$IP}"
```

```
TG='arn:aws:elasticloadbalancing:...'
C='containerName=api,containerPort=8080'
aws ecs create-service --cluster prod \
  --service-name api --task-definition api \
  --desired-count 2 --launch-type FARGATE \
  --network-configuration "$NET" \
  --load-balancers "targetGroupArn=$TG,$C" \
  --health-check-grace-period-seconds 60
```

> **Gotcha:** `CannotPullContainerError` almost always means no route to ECR. In private subnets, add a NAT gateway, or the ECR API, ECR Docker and S3 gateway endpoints.

Deployments and the circuit breaker

Setting	Default
Strategy	`ROLLING` (also `BLUE_GREEN`, `LINEAR`, `CANARY`)
`minimumHealthyPercent`	100
`maximumPercent`	200
Circuit breaker `rollback`	Off unless enabled
Health-check grace period	0 s

A rolling update starts new tasks up to `maximumPercent`, waits for them to pass health checks, then stops old ones. The **deployment circuit breaker** marks a deployment failed when tasks keep failing to reach

steady state; with `rollback` on, ECS returns to the last successful deployment. The progressive strategies need a load balancer, Service Connect or VPC Lattice.

```
aws ecs update-service --cluster prod \
  --service api \
  --deployment-configuration '{
    "deploymentCircuitBreaker":
      {"enable": true, "rollback": true},
    "minimumHealthyPercent": 100,
    "maximumPercent": 200}'
```

> **Gotcha:** With a grace period of 0, a slow-starting app fails the ALB health check before it is ready, gets killed, and the deployment loops. Set the grace period above your real startup time.

Logs and secrets

Field	Purpose
`logConfiguration.logDriver: awslogs`	stdout/stderr to CloudWatch Logs
`awslogs-stream-prefix`	Stream named prefix/container/task-id
`secrets[].valueFrom`	Secrets Manager or SSM parameter ARN
`environment[]`	Plain, non-secret values

Write logs to stdout; the `awslogs` driver ships them. `secrets` injects a value as an environment variable at container start, fetched with the execution role — the value never appears in the task definition.

```
{
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-group": "/ecs/api",
      "awslogs-region": "eu-west-1",
      "awslogs-stream-prefix": "api"
    }
  },
  "secrets": [{ "name": "DB_PASSWORD",
    "valueFrom": "arn:aws:secretsmanager:..." }]
}
```

> **Gotcha:** Secrets are read once, at container start. Rotating a secret does not reach running tasks until a new deployment replaces them — `update-service --force-new-deployment`.

Autoscaling

Policy	Scales on
Target tracking	`ECSServiceAverageCPUUtilization`
Target tracking	`ALBRequestCountPerTarget`

Policy	Scales on
Step scaling	Any CloudWatch alarm
Scheduled	Time of day

Service autoscaling is Application Auto Scaling acting on `desiredCount`, between a min and max you register. Target tracking is the right default: pick one metric and a target value, and it adds or removes tasks to hold it.

```
aws application-autoscaling \
  register-scalable-target \
  --service-namespace ecs \
  --resource-id service/prod/api \
  --scalable-dimension ecs:service:DesiredCount \
  --min-capacity 2 --max-capacity 10
```

Debugging with ECS Exec

Requirement	Detail
Service flag	`--enable-execute-command`
Task role	`ssmmessages` create/open channel actions
Your machine	AWS CLI + Session Manager plugin
Private subnets	`ssmmessages` VPC endpoint

ECS Exec opens a shell in a running container through Systems Manager. The flag applies to tasks launched after you set it, so force a new

deployment. Commands run as root, so restrict `ecs:ExecuteCommand` in IAM.

```
aws ecs update-service --cluster prod \
  --service api --enable-execute-command \
  --force-new-deployment
aws ecs execute-command --cluster prod \
  --task 0123abcd --container api \
  --interactive --command "/bin/sh"
```

Further reading

- Amazon ECS Developer Guide
- Fargate task CPU and memory
- ECS service deployment strategies
- Amazon ECS pricing