AMAZON ECS

A FULLY MANAGED CONTAINER ORCHESTRATION SERVICE

DOCKER 🐋

Before getting started with ECS, you need to understand Docker, because it's one of the basic building blocks.

Docker helps to create environments to run your application, regardless of the underlying operating system.

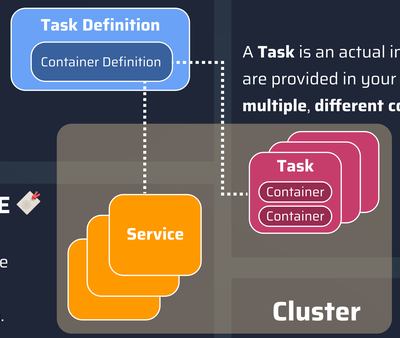
KEY TERMS 🔑

Let's dig into **ECS' Key Terms**, which can be confusing at first but are crucial to understand how it internally works: **Task Definition**, **Task**, **Service** and **Cluster**.

TASK DEFINITION 🧨

A **Task Definition** is a blueprint of your container. It includes things like:

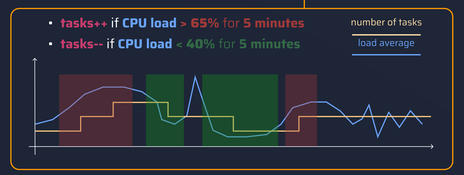
- the **image** to use
- **CPU & memory** allocation
- **secrets** & **environment** vars
- logging configuration
- **exposed ports**



SERVICE 🧨

As we can have several tasks for the same definition, we need some boundaries and management.
This is where the service comes in.

Among a lot of other things, you're able to configure rules for **auto-scaling**, **load distribution** as well as the **minimum** & **maximum number of tasks**.



CONTAINERS 🏗️

This lightweight environment is called a **Container** and - like the name already suggests - contains everything that is needed to run your application, like certain versions of a library or language.

You can run multiple containers on the same machine. Containers can even **communicate** with each other when needed.

When your application grows, there will most probably be challenges in managing all the **deployments**, **containers**, **scheduling** and other **tedious tasks**.

TASK 🧨

A **Task** is an actual instance that runs the **containers** that are provided in your definition. A single task can run **multiple**, **different containers** for **different purposes**.

Additionally, you can run **multiple tasks** from the same definition if required, for example to have higher **redundancy** or **meet increasing traffic demands**.

CLUSTER 🧨

A **Cluster** is a logical grouping of tasks or services which run on infrastructure that is registered to such a cluster. You can even provide your **on-premise virtual machines** as compute capacities for your cluster.

OBSERVABILITY 🔎

CloudWatch comes with default metrics like **CPU** or **memory usage** to grant you insights into your ECS services & tasks.

Developer Tools like **Dashbird.io** will collect metrics in a single place, guide you with **well-architected hints** & can notify you on **critical service events** via Slack.

CONTAINER MANAGEMENT ⚙️

That's why you'll be in need of a container management or orchestration service. It's another abstracting layer that helps you easily manage your containerized applications and reduce your operational tasks.
That's exactly where **ECS** steps in.

CONTAINER INSTANCE VS. FARGATE 🤖

If you have very high computation requirements, you should know that Fargate is way more **restrictive** regarding possible capacities for a single task.

	Container Instance	Fargate
CPU	448 vCPUs	4 vCPUs
Memory	26TB	30 GB

⚠ only certain configurations are supported

CPU	Memory Values
0.25 vCPU	0.5 GB, 1 GB, and 2 GB
0.5 vCPU	1 GB - 4 GB
1 vCPU	2 GB - 8 GB
2 vCPU	4 GB - 16 GB
4 vCPU	8 GB - **30** GB

⚠ **General note:** Even if you're a big serverless fan, knowing about ECS is crucial because you'll bump into it almost **everywhere**.

Considering the abstraction layer of ECS in combination with Fargate, it's considered a **serverless technology**.

ELASTIC CONTAINER SERVICE 🤖

Amazons **E**lastic **C**ontainer **S**ervice (ECS) is a highly scalable & fast container management service.
It allows you to view & manage the state of your clusters from a centralized service. Easily schedule based on resource needs & availability requirements.

There are **two different areas of responsibility**:
- **Lifecycle management** of your tasks
- Actually **running** your containers

ECS does not actually execute or run your containers. It only provides the **management pane for controlling your tasks**.

LAUNCH TYPES 🚀

So finally to the important question: which services actually **run** our containers?

which will have the ECS Container Agent running

You can either pick from using...

- **EC2 Launch Type**
- **Fargate Launch Type**
- **External Launch Type**

So it's for example not ECS **or** Fargate, but ECS **and** Fargate.

Fargate offers a **higher abstraction**, as you're not responbility for the underlying infrastructure.
The External Launch Types allow you to register on-premise servers / virtual machines to your ECS cluster.

LAMBDA VS. FARGATE

Both Lambda and Fargate go intro the category of **Serverless** as they remove a lot of technical burdens due to not having to manage a lot of (or any) underlying infrastructure, it's worth to do a quick **comparison**:

- 💰 **Cost**: Lambda is solely **pay-per-use** - you don't have any costs for idle functions. Your Fargate tasks are always charged if they are **running**. Nevertheless, looking at high traffic demands, Fargate can quickly outperform Lambda at costs.
- ⚡ **Scalability**: Lambda immediately scales horizontally; Fargate takes more time to start & register new tasks.
- 📈 **Performance**: A Lambda function can only serve a single request at a time, so that varying traffic patterns will cause a lot of **cold starts & slow responses**. Without an AWS native key-value store like DynamoDB, you'll also have to fight with database connection management. You won't face any of these issues with Fargate.
- 🏗️ **Getting Started**: with Lambda, networking knowledge is almost not relevant & tools like **Serverless Framework** enable you to spin up a simple application within minutes. ECS & Fargate need more time, even with great helpers like **CDK & Level 3 Constructs** that simplify via high abstraction.

```
ENV ECS_CONTAINER_STOP_TIMEOUT=2
ENV ECS_IMAGE_PULL_BEHAVIOUR=prefer-cached

# [ ... ]
```

BONUS - SPEEDING UP ECS CONTAINER DEPLOYMENTS ⚡

With a non-optimise configuration, deploying a new task definition to your cluster can take **several minutes**.
You can fine-tune many settings to get down to just **a few seconds**.

1. ECS Agent Settings
- prefer **cached images** from EC2 disk cache
- reduce **grace periods** for container shutdown

⚠ only when using **immutable tags**

2. Load Balancer Settings
- reduce **time for keeping connections** alive
- reduce **intervals for health checks** & the needed number of successful ones until the task is considered healthy

⚠ be aware if you're using long running connections (e.g. sockets)

3. ECS Deployment Settings
- decrease the number of **required healthy tasks** to enable ECS to start more tasks in parallel

```
resource "aws_lb_target_group" "main" {
  // [ ... ]
  deregistration_delay = 5
  health_check {
    healthy_threshold = 2
    interval          = 10
  }
}

resource "aws_ecs_service" "main" {
  // [ ... ]
  deployment_maximum_percent         = 200
  deployment_minimum_healthy_percent = 50
}
```


Credits: Nathan Peck 👋
@nathankpeck nathanpeck.com

aws