

# Introduction to How AWS ECS Works with Example Tutorial

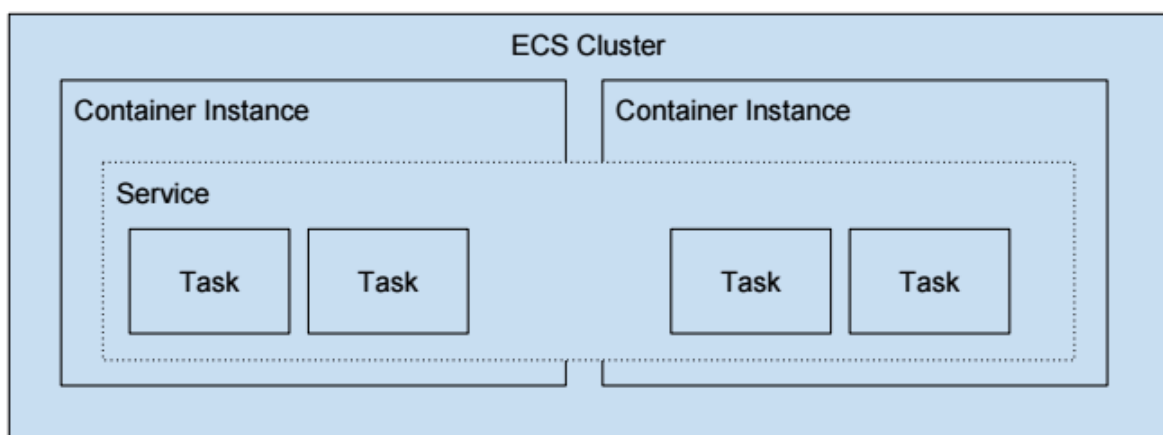
## Summary of the ECS Terms

First we need to cover ECS terminology:

- Task Definition — This a blueprint that describes how a docker container should launch. If you are already familiar with AWS, it is like a LaunchConfig except instead it is for a docker container instead of a instance. It contains settings like exposed port, docker image, cpu shares, memory requirement, command to run and environmental variables.
- Task — This is a running container with the settings defined in the Task Definition. It can be thought of as an “instance” of a Task Definition.
- Service — Defines long running tasks of the same Task Definition. This can be 1 running container or multiple running containers all using the same Task Definition.
- Cluster — A logic group of EC2 instances. When an instance launches the ecs-agent software on the server registers the instance to an ECS Cluster. This is easily configurable by setting the ECS\_CLUSTER variable in /etc/ecs/ecs.config described [here](#).

- Container Instance — This is just an EC2 instance that is part of an ECS Cluster and has docker and the [ecs-agent](#) running on it.

AWS provides nice [detailed diagrams](#) to help explain the terms. Here is a simplified diagram to help visualize and explain the terms.



ECS Terms

In this diagram you can see that there are 4 running Tasks or Docker containers. They are part of an ECS Service. The Service and Tasks span 2 Container Instances. The Container Instances are part of a logical group called an ECS Cluster.

I did not show a Task Definition in the diagram because a Task is simply an “instance” of Task Definition.

# Tutorial Example

In this tutorial example we will create a small Sinatra web service that prints the meaning of life: [42](#).

1. Create ECS Cluster with 1 Container Instance
2. Create a Task Definition
3. Create an ELB and Target Group to later associate with the ECS Service
4. Create a Service that runs the Task Definition
5. Confirm Everything is Working
6. Scale Up the Service to 4 Tasks.
7. Clean It All Up

The [ECS First Run Wizard](#) provided in the [Getting Started with Amazon ECS documentation](#) performs the similar above with a CloudFormation template and ECS API calls. We are doing it out step by step because I believe it better help to understand the ECS components.

## **1. Create ECS Cluster with 1 Container Instance**

Before creating a cluster, let's create a security group called `my-ecs-sg` that we'll use.

```
aws ec2 create-security-group --group-name my-ecs-sg --  
description my-ecs-sg
```

Now create an ECS Cluster called `my-cluster` and the `ec2` instance that belongs to the ECS Cluster. Use the `my-ecs-sg` security group that was created. You can get the id of the security group from the EC2 Console / Network & Security / Security Groups. It is important to select a Key pair so you can ssh into the instance later to verify things are working.

For the Networking VPC settings, I used the default VPC and all the Subnets associated with the account to keep this tutorial simple. For the IAM Role use `ecsInstanceRole`. If `ecsInstanceRole` does not yet exist, create it per [AWS docs](#). All the my settings are provided in the screenshot. You will need to change the settings according to your own account and default VPC and Subnets.

## Create Cluster

When you run tasks using Amazon ECS, you place them on a cluster, which is a logical grouping of EC2 instances. This wizard will guide you through the process to [create a](#) container instances that your tasks can be placed on, the security group for your container instances to use, and the IAM role to associate with your container instances so th

Cluster name\* my-cluster ⓘ

☐ Create an empty cluster

EC2 instance type\* t2.small ⓘ

Number of instances\* 1 ⓘ

EC2 Ami Id\* amzn-ami-2016.09.b-amazon-ecs-optimized [ami-eca289fb] ⓘ

EBS storage (GiB)\* 22 ⓘ

Key pair default ⓘ

You will not be able to SSH into your EC2 instances without a key pair. You can create a new key pair in the [EC2 console](#).

## Networking

Configure the VPC for your container instances to use. A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You can ch

VPC vpc-ab56e3cc ⓘ

Check Structure for [vpc-ab56e3cc](#) in the EC2 Console.

Subnets ⓘ

subnet-c90454e3 ⓘ subnet-6d6b5850 ⓘ  
subnet-1a64646c ⓘ subnet-f61b46ae ⓘ

Select a subnet...

Wait a few minutes and then confirm that the Container Instance has successfully registered to the `my-cluster` ECS cluster. You can confirm it by clicking on the ECS Instances tab under Clusters / `my-cluster`.

## Cluster : my-cluster

Delete Cluster

Get a detailed view of the resources on your cluster.

Status **ACTIVE**

Registered container instances 1

Pending tasks count 0

Running tasks count 0

Services

Tasks

ECS Instances

Metrics

Scale ECS Instances

View Cluster Resources

Last updated on November 25, 2016 9:49:26 PM (0m ago) ⓘ

Filter in this page

&lt; Viewing 1-1 Container Instance &gt; Results per page 50

Container Instance	EC2 Instance	Agent Conn...	Status	Running tas...	CPU availab...	Memory ava...	Agent versi...	Docker vers...
01aa5d70-588e-4c20-8d9...	i-0ac2a083148...	true	ACTIVE	0	1024	2003	1.13.1	1.11.2

## 2. Create a task definition that will be blueprint to start a Sinatra app

Before creating the task definition, find a [sinatra docker image](#) to use and test that it's working. I'm using the tongueroo/sinatra image.

```
$ docker run -d -p 4567:4567 --name hi tongueroo/sinatra
6df556e1df02e93b05aa46425fc539121f5e50afee630e1cd918b337c3b6c202
$ docker ps
CONTAINER ID          IMAGE               COMMAND             2
CREATED              STATUS             PORTS
NAMES
6df556e1df02         tongueroo/sinatra  "ruby hi.rb"       0.0.0.0:4567->4567/tcp
seconds ago          Up 1 seconds
hi
$ curl localhost:4567 ; echo
42
$ docker stop hi ; docker rm hi
$
```

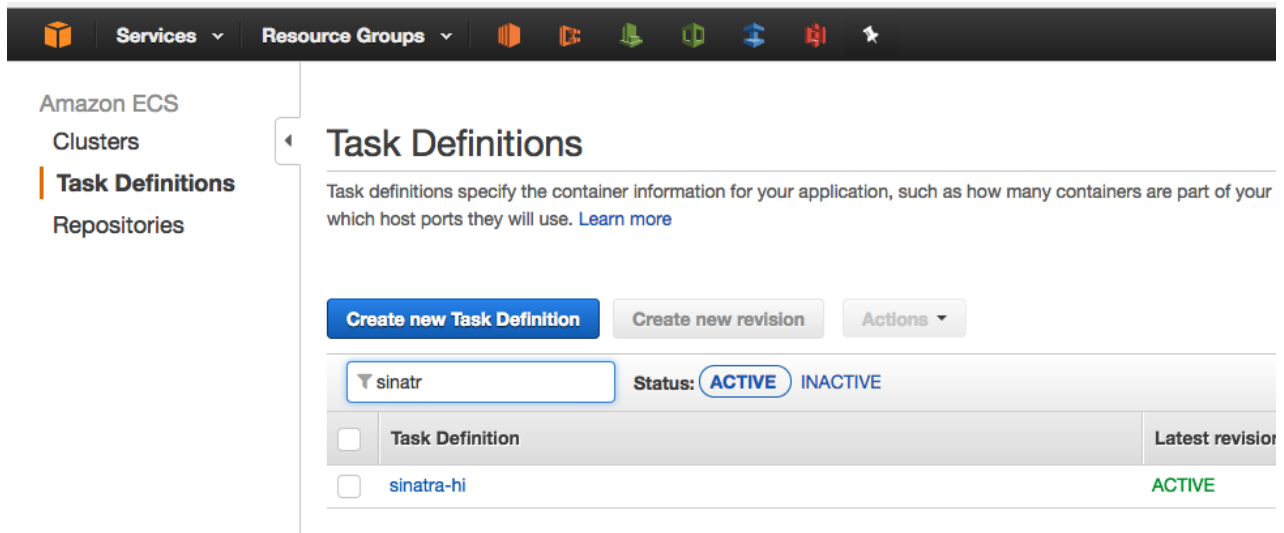
Above, I've started a container with the sinatra image and curl localhost:4567. Port 4567 is the default port that sinatra listens on and it is exposed in the [Dockerfile](#). It returns "42" as expected. Now that I've tested the sinatra image and verify that it works, let's create the task definition. Create a task-definition.json and add:

```
{
  "family": "sinatra-hi",
  "containerDefinitions": [
    {
      "name": "web",
      "image": "tongueroo/sinatra:latest",
      "cpu": 128,
      "memoryReservation": 128,
      "portMappings": [
        {
          "containerPort": 4567,
          "protocol": "tcp"
        }
      ],
      "command": [
        "ruby", "hi.rb"
      ],
      "essential": true
    }
  ]
}
```

The task definition is also available on GitHub: [task-definition.json](#). To register the task definition:

```
$ aws ecs register-task-definition --cli-input-json file://task-definition.json
```

Confirm that the task definition successfully registered with the ECS Console:



### 3. Create an ELB and Target Group to later associate with the ECS Service

Now let's create an ELB and a target group with it. We are creating an ELB because we eventually want to load balance requests across multiple containers and also want to expose the sinatra app to the internet for testing. The easiest way to create an ELB is with the EC2 Console.

Go the EC2 Console / Load Balancing / Load Balancers, click "Create Load Balancer" and select Application Load Balancer.

#### Wizard Step 1 — Configure Load Balancer

- Name it `my-elb` and select internet-facing.
- Use the default Listener with a HTTP protocol and Port 80.
- Under Availability Zone, chose a VPC and choose the subnets you would like. I chose all 4 subnets in the default VPC just



like step 1. It is very important to chose the same subnets that was chosen when you created the cluster in step 1. If the subnets are not the same the ELB health check can fail and the containers will keep getting destroyed and recreated in an infinite loop if the instance is launched in an AZ that the ELB is not configured to see.

### Wizard Step 2 — Configure Security Settings

- There will be a warning about using a secure listener, but for the purpose of this exercise we can skip using SSL.

### Wizard Step 3 — Configure Security Groups

- Create a new security group named `my-elb-sg` and open up port 80 and source 0.0.0.0/0 so anything from the outside world can access the ELB port 80.

### Wizard Step 4 — Configure Routing

- Create a new target group name `my-target-group` with port 80.

### Wizard Step 5 — Register Targets

- This step is a little odd for ECS. We do actually not register any targets here because ECS will automatically register the targets for us when new tasks are launched. So simply skip and click next.

## Wizard Step 6 — Review

- Review and click create.

### Step 6: Review

Please review the load balancer details before continuing

#### ▼ Load balancer

**Name** my-elb  
**Scheme** internet-facing  
**Listeners** Port:80 - Protocol:HTTP  
**VPC** vpc-ab56e3cc  
**Subnets** subnet-1a64646c, subnet-f61b46ae  
**Tags**

#### ▼ Security groups

**Security groups** sg-d32ebcae

#### ▼ Routing

**Target group** New target group  
**Target group name** my-target-group  
**Port** 80  
**Protocol** HTTP  
**Health check protocol** HTTP  
**Path** /  
**Health check port** traffic port  
**Healthy threshold** 5  
**Unhealthy threshold** 2  
**Timeout** 5  
**Interval** 30  
**Success codes** 200

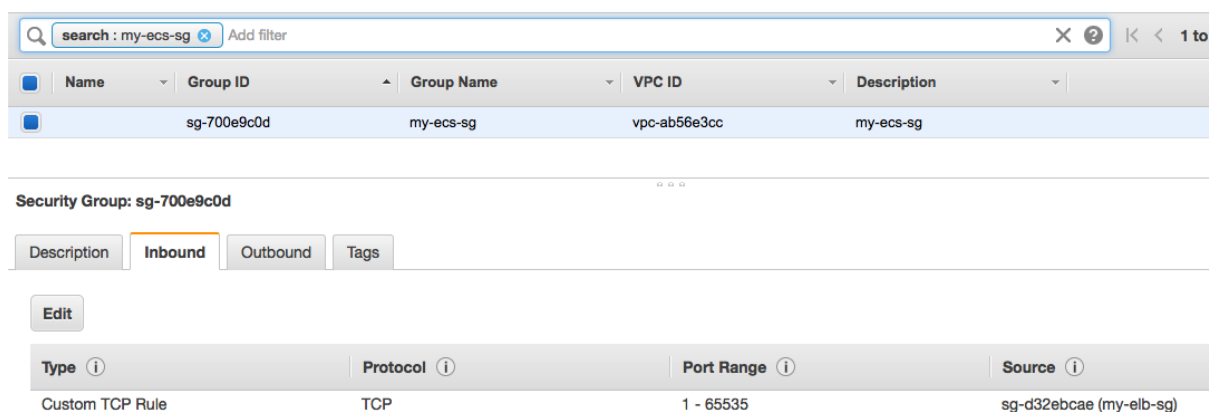
#### ▼ Targets

**Instances**

When we created the ELB with the wizard we opened it's `my-elb-sg` group port 80 to the world. We also need to make sure that the `my-ecs-sg` security group associated with the instance we launched in step 1 allows traffic from the ELB. We created the `my-ecs-sg` group in step 1 at the very beginning of this tutorial. To allow all ELB traffic to hit the container instance run the following:

```
$ aws ec2 authorize-security-group-ingress --group-name my-ecs-sg --protocol tcp --port 1-65535 --source-group my-elb-sg
```

Confirm the rules were added to the security groups via the EC2 Console:



The screenshot shows the AWS EC2 Console interface. At the top, there is a search bar with the text "search : my-ecs-sg". Below the search bar, there is a table with columns: Name, Group ID, Group Name, VPC ID, and Description. The table contains one row with the following values: Name (my-ecs-sg), Group ID (sg-700e9c0d), Group Name (my-ecs-sg), VPC ID (vpc-ab56e3cc), and Description (my-ecs-sg). Below the table, there is a section titled "Security Group: sg-700e9c0d". This section has four tabs: Description, Inbound, Outbound, and Tags. The Inbound tab is selected. Below the tabs, there is an "Edit" button. Below the "Edit" button, there is a table with columns: Type, Protocol, Port Range, and Source. The table contains one row with the following values: Type (Custom TCP Rule), Protocol (TCP), Port Range (1 - 65535), and Source (sg-d32ebcae (my-elb-sg)).

Name	Group ID	Group Name	VPC ID	Description
my-ecs-sg	sg-700e9c0d	my-ecs-sg	vpc-ab56e3cc	my-ecs-sg

Type	Protocol	Port Range	Source
Custom TCP Rule	TCP	1 - 65535	sg-d32ebcae (my-elb-sg)

With these security group rules, only port 80 on the ELB is exposed to the outside world and any traffic from the ELB going to a container instance with the `my-ecs-group` group is allowed. This a nice simple setup.

## 4. Create a Service that runs the Task Definition

The command to create the ECS service takes a few parameters so it is easier to use a json file as it's input. Let's create a `ecs-service.json` file with the following:

```
{
  "cluster": "my-cluster",
  "serviceName": "my-service",
  "taskDefinition": "sinatra-hi",
  "loadBalancers": [
    {
      "targetGroupArn": "FILL-IN-YOUR-TARGET-GROUP",
      "containerName": "web",
      "containerPort": 4567
    }
  ],
  "desiredCount": 1,
  "role": "ecsServiceRole"
}
```

You will have to find your `targetGroupArn` created in step 3 when we created the ELB. To find the `targetGroupArn` you can go to the EC2 Console / Load Balancing / Target Groups and click on the `my-target-group`.

Now create the ECS service: `my-service`.

```
$ aws ecs create-service --cli-input-json file://ecs-
service.json
```

You can confirm that the container is running on the ECS Console. Go to Clusters / `my-cluster` / `my-service` and view the Tasks tab.

## Service : my-service

Update

Delete

## Details

Cluster [my-cluster](#)

Status **ACTIVE**

Task Definition [sinatra-hi:1](#)

Desired count 1

Pending count 0

Running count 1

Service Role ecsServiceRole

## Load Balancing

Target Group Name	Container Name	Container Port
<a href="#">my-target-group</a>	web	4567

## Deployment Options

Minimum healthy percent 100 ⓘ

Maximum percent 200 ⓘ

## Tasks

## Events

## Deployments

## Auto Scaling

## Metrics

Last updated on November 25, 2016 10:00:21 PM (4m ago)



Filter in this page

Task status: **Running** Stopped

&lt; Viewing 1-1 Task &gt; Results per page 50 ▼

Task	Task Definition	Last status	Desired status
<a href="#">2b21203b-a738-4b4c-90c2-e7889c5588ff</a>	<a href="#">sinatra-hi:1</a>	<b>RUNNING</b>	RUNNING

## 5. Confirm Everything is Working

Confirm that the service is running properly. You want to be thorough about confirming that all is working by checking a few things.

Check that `my-target-group` is showing and maintaining healthy targets. Under Load Balancing / Target Groups, click on `my-target-group` and check the Targets tab. You should see a Target that is reporting healthy.

Filter: <input type="text" value="my-target-group"/>						1 to 1 of 1
<input checked="" type="checkbox"/>	Name	Port	Protocol	VPC ID	Monitoring	
<input checked="" type="checkbox"/>	my-target-group	80	HTTP	vpc-ab56e3cc	<input checked="" type="checkbox"/>	

If the target is not healthy, check these likely issues:

Let also ssh into the instance and see the running docker process is returning a good response. Under Clusters / ECS

Instances, click on the Container Instance and grab the public dns record so you can ssh into the instance.

Clusters > my-cluster

### Cluster : my-cluster

Get a detailed view of the resources on your cluster.

Status **ACTIVE**

Registered container instances 1

Pending tasks count 0

Running tasks count 1

Services Tasks **ECS Instances** Metrics

[Scale ECS Instances](#) [View Cluster Resources](#)

Last updated on November 25, 2016 10:24:54 PM (0m ago)

Filter in this page

< Viewing 1-1 Container Instance > Results per page 50

Container Instance	EC2 Instance	Agent Conn...	Status	Running tas...	CPU availab...	Memory ava...	Agent versi...	Docker vers...
<a href="#">01aa5d70-588e-4c20-8d9...</a>	<a href="#">i-0ac2a083148...</a>	true	<b>ACTIVE</b>	1	896	1875	1.13.1	1.11.2

Clusters > my-cluster > Container Instance: 01aa5d70-588e-4c20-8d92-0ec6f8ab9a8c

### Container Instance : 01aa5d70-588e-4c20-8d92-0ec6f8ab9a8c

[Update agent](#) [Deregister](#)

#### Details

Cluster [my-cluster](#)

EC2 Instance [i-0ac2a0831481db17d](#)

Public DNS [ec2-52-3-252-86.compute-1.amazonaws.com](#)

Private DNS ip-172-31-45-158.ec2.internal

Public IP 52.3.252.86

Private IP 172.31.45.158

Status **ACTIVE**

Running tasks count 1

Agent version 1.13.1

Docker version 1.11.2

#### Resources

Resources	Registered	Available
CPU	1024	896
Memory	2003	1875
Ports	5 ports	

#### Tasks

```
$ ssh ec2-user@ec2-52-3-252-86.compute-1.amazonaws.com
$ docker ps
CONTAINER ID          IMAGE                                COMMAND
CREATED              STATUS                            PORTS
NAMES
9e9a55399589         tongueroo/sinatra:latest          "ruby hi.rb"
16 minutes ago       Up 16 minutes                    8080/tcp, 0.0.0.0:32773-
```

```
>4567/tcp    ecs-sinatra-hi-1-web-d8efaad38dd7c3c63a00
4fea55231363    amazon/amazon-ecs-agent:latest    "/agent"
41 minutes ago    Up 41 minutes
ecs-agent
$ curl 0.0.0.0:32773 ; echo
42
$
```

Above, I've verified that the docker container running on the instance by curling the app and seeing a successful response with the "42" text.

Lastly, let's also verify by hitting the external DNS address of the ELB. You can find the DNS address in the EC2 Console under Load Balancing / Load Balancers and clicking on `my-elb`.



The screenshot displays the AWS Management Console interface for configuring an Elastic Load Balancing (ELB) instance. On the left, a navigation menu lists various AWS services, with 'LOAD BALANCING' and 'Load Balancers' highlighted. The main panel shows the 'Create Load Balancer' button and a table of existing load balancers. A table with columns 'Name', 'DNS name', 'State', and 'VP' lists the 'my-elb' instance with its DNS name 'my-elb-1693572386.us-east-1.elb.amazonaws.com' and state 'active'. Below this, the 'Load balancer: my-elb' section is shown with tabs for 'Description', 'Listeners', 'Monitoring', and 'Tags'. The 'Description' tab is active, displaying the 'Basic Configuration' section. This section lists the following details: Name: my-elb, ARN: arn:aws:elasticloadbalancing:us-east-1:467446852200:loadbalancer/app/my-elb/b3cbd15ff9d53031, DNS name: my-elb-1693572386.us-east-1.elb.amazonaws.com (A Record), Scheme: internet-facing, Type: application, and Availability Zones: subnet-1a64646c - us-east-1a, subnet-6d6b5850 - us-east-1e, subnet-c90454e3 - us-east-1d. A blue arrow points from the 'DNS name' label to the DNS name value, which is highlighted with a blue box.

EC2 Dashboard  
Events  
Tags  
Reports  
Limits

INSTANCES  
Instances  
Spot Requests  
Reserved Instances  
Scheduled Instances  
Dedicated Hosts

IMAGES  
AMIs  
Bundle Tasks

ELASTIC BLOCK STORE  
Volumes  
Snapshots

NETWORK & SECURITY  
Security Groups  
Elastic IPs  
Placement Groups  
Key Pairs  
Network Interfaces

LOAD BALANCING  
Load Balancers  
Target Groups

Create Load Balancer Actions

Filter: my-elb

Name	DNS name	State	VP
my-elb	my-elb-1693572386.us-east-1.elb.amazonaws.com	active	vpc

Load balancer: my-elb

Description Listeners Monitoring Tags

Basic Configuration

Name: my-elb

ARN: arn:aws:elasticloadbalancing:us-east-1:467446852200:loadbalancer/app/my-elb/b3cbd15ff9d53031

DNS name: my-elb-1693572386.us-east-1.elb.amazonaws.com (A Record)

Scheme: internet-facing

Type: application

Availability Zones: subnet-1a64646c - us-east-1a, subnet-6d6b5850 - us-east-1e, subnet-c90454e3 - us-east-1d.

Verify the ELB publicly available dns endpoint with curl:

```
$ curl my-elb-1693572386.us-east-1.elb.amazonaws.com ; echo
42
$
```

## 6. Scale Up the Service to 4 Tasks

This is the easiest part. To scale up and add more containers simply go to Clusters / my-cluster / my-service and click on “Update Service”. You can change “Number of tasks” from 1 to 4 there. After only a few moments you should see 4 running tasks. That’s it!

## 7. Clean It All Up

It is quickest to use the EC2 Console to delete the following resources:

- ELB: my-elb
- ECS Service: my-service Task Definition: sinatra-hi Cluster: my-cluster
- Security group: my-elb-sg and my-ecs-sg.

## Summary

In this post I covered the ECS terminology and went through a simple example to create a sinatra app behind a ELB.

Overall, I think that ECS is a pretty amazing service and it has taken the hassle of managing docker orchestration and provisioning responsibility away.