

Scaling AWS EC2 Containers

This exercise assumes you have completed the previous lab exercises in which you dockerised an app and used the AWS Container service to pull the app from Dockerhub and ran the app in a container. In this exercise you will trigger the creation of multiple container instances based on a Cloudwatch alarm.

You will need to create an Elastic Load Balancer from the EC2 service page. Choose the Application Load Balancer and configure it

The screenshot displays the AWS Management Console interface for configuring an Elastic Load Balancer. At the top, a progress bar indicates six steps: 1. Configure Load Balancer (active), 2. Configure Security Settings, 3. Configure Security Groups, 4. Configure Routing, 5. Register Targets, and 6. Review.

Step 1: Configure Load Balancer
Basic Configuration
To configure your load balancer, provide a name, select a scheme, specify one or more listeners, and select a network. The default configuration is an Internet-facing load balancer in the selected network with a listener that receives HTTP traffic on port 80.

Name: ContainerLB
Scheme: internet-facing (selected), internal

Listeners
A listener is a process that checks for connection requests, using the protocol and port that you configured.

Load Balancer Protocol: HTTP
Load Balancer Port: 80
Add listener

Availability Zones
Specify the Availability Zones to enable for your load balancer. The load balancer routes traffic to the targets in these Availability Zones only. You can specify only one subnet per Availability Zone. You must specify subnets from at least two Availability Zones.

VPC: vpc-2999bc42 (172.31.0.0/16) | Default VPC (default)

Available subnets

| Actions | Availability Zone | Subnet ID | Subnet CIDR |
|---------|-------------------|-----------------|----------------|
| | eu-west-1c | subnet-2f99bc44 | 172.31.16.0/20 |

Selected subnets

| Actions | Availability Zone | Subnet ID | Subnet CIDR |
|---------|-------------------|-----------------|----------------|
| | eu-west-1a | subnet-2ef9bc45 | 172.31.32.0/20 |
| | eu-west-1b | subnet-2899bc43 | 172.31.0.0/20 |

You will need to configure a new Target group (e.g. ContainerTG) and register your Cluster Instance with this group. If you are using the sample PHP application you can change the Health check Path to /index.php.

Once your Application Load Balancer has been created you will need to create a Service on your ECS Cluster and configure Load Balancing and Auto Scaling

Note to overcome the issue of running multiple containers on the same instance you will need to create a task definition that specifies Host Port 0 and container port 80. This will dynamically assign any available port when it runs the docker container and register this with the Target Group.

Edit container

Container name* ⓘ

Image* ⓘ

Custom image format: [registry-url]/[namespace]/[image]:[tag]

Memory Limits (MiB)* Hard limit ▼ ⓘ

[+ Add Soft limit](#)

Define hard and/or soft memory limits in MiB for your container. Hard and soft limits correspond to the 'memory' and 'memoryReservation' parameters, respectively, in task definitions. ECS recommends 300-500 MiB as a starting point for web applications.

Port mappings

| Host port | Container port | Protocol |
|--------------------------------|---------------------------------|--------------------|
| <input type="text" value="0"/> | <input type="text" value="80"/> | tcp ▼ |

[+ Add port mapping](#)

If you intend to use an Application Load Balancer (ALB) with your tasks, you can set the host port to 0 to enable dynamic host port mapping. This allows you to run more than one copy of a task on a container instance. [Learn more](#)

You should see that the containers have been registered as targets on the dynamically generated ports.

[Create target group](#) Actions ▼ 🔄 ⚙️

Filter: < 1 to 1 of 1 >

| Name | Port | Protocol | Target type | VPC ID | Monitoring |
|-------------|------|----------|-------------|--------------|----------------|
| ContainerTG | 80 | HTTP | instance | vpc-2999bc42 | 📊 |

Description **Targets** **Health checks** **Monitoring** **Tags**

The load balancer starts routing requests to a newly registered target as soon as the registration process completes and the target passes the initial health checks. If demand on your targets increases, you can register additional targets. If demand on your targets decreases, you can deregister targets.

[Edit](#)

Registered targets

| Instance ID | Name | Port | Availability Zone | Status |
|--------------------|------|-------|-------------------|-----------|
| i-0163393240a9b0e4 | | 32802 | eu-west-1a | healthy ⓘ |
| i-0163393240a9b0e4 | | 32803 | eu-west-1a | healthy ⓘ |

You can check the same on your server using the 'docker ps' command. Two different containers are running on two random ports by the same docker image that you mentioned in your task definition.

```

[ec2-user@ip-172-31-44-147 ~]$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
539d0982a23c   busybox        "sh -c '/bin/sh -c..." About an hour ago Up About an hour                   ecs-console-sample-app-6-busybox-9acb93e38bf7
bdc1df01       busybox        "sh -c '/bin/sh -c..." About an hour ago Up About an hour                   ecs-console-sample-app-6-busybox-98a482f0c1df
b4bd5000       rfrisby/amazon-ecs-sample "/usr/sbin/apache2..." About an hour ago Up About an hour   0.0.0.0:32803->80/tcp ecs-console-sample-app-6-simple-app-e2eeba92a
c65dda98d01    rfrisby/amazon-ecs-sample "/usr/sbin/apache2..." About an hour ago Up About an hour   0.0.0.0:32802->80/tcp ecs-console-sample-app-6-simple-app-b6b4faa7d
cd685a75e00    amazon/amazon-ecs-agent:latest "/agent"              4 hours ago    Up 4 hours                               ecs-agent
[ec2-user@ip-172-31-44-147 ~]$

```

So If you are running multiple containers of a single service, you don't need multiple servers for them. ALB allows to maximize the usage of servers and offers you a high-performance load balancing option. It gives you the flexibility of running multiple containers of a service on a single server by using the random available port.

Create a new service from the task and in ‘Configure ELB’ section, select application load balancer in ELB type section, select your ALB & target group and create service. If your ECS cluster has single ECS instance and tasks count is two, it will start two new containers in your instance on two different dynamic ports.

Service : test

Update

Cluster [RFClusterDemo](#)

Status ACTIVE

Task Definition [console-sample-app:6](#)

Desired count 2

Pending count 0

Running count 2

Service Role [ecsServiceRole](#)

Details

Tasks

Events

Auto Scaling

Deployments

Metrics

Load Balancing

| Target Group Name | Container Name | Container Port |
|-----------------------------|----------------|----------------|
| ContainerTG | simple-app | 80 |

Task Placement

Strategy

spread(attribute:ecs.availability-zone), spread(instanceid)

Constraint

No constraints

Service Auto Scaling instructions taken from -

http://docs.aws.amazon.com/AmazonECS/latest/developerguide/service_autoscaling_tutorial.html

When configuring the Auto Scaling Service:

For Minimum number of tasks, enter 2 for the lower limit of the number of tasks for Service Auto Scaling to use. Your service's desired count will not be automatically adjusted below this amount.

For Desired number of tasks, this field is pre-populated with the value you entered earlier. This value must be between the minimum and maximum number of tasks specified on this page. Leave this value at 2.

For Maximum number of tasks, enter 3 for the upper limit of the number of tasks for Service Auto Scaling to use. Your service's desired count will not be automatically adjusted above this amount.

For IAM role for Service Auto Scaling, choose an IAM role to authorize the Application Auto Scaling service to adjust your service's desired count on your behalf. If you have not previously created such a role, choose Create new role and the role is created for you. For future reference, the role that is created for you is called `ecsAutoscaleRole`

To configure scaling policies for your service

These steps will help you create scaling policies and CloudWatch alarms that can be used to trigger scaling activities for your service. You can create a scale out alarm to increase the desired count of your service, and a scale in alarm to decrease the desired count of your service.

1. For Policy name, enter `ScaleOutPolicy`
2. For Execute policy when, choose Create new alarm.
 - a. For Alarm name, enter `sample-webapp-cpu-gt-75`.
 - b. For ECS service metric, choose `CPUUtilization`.
 - c. For Alarm threshold, enter the following information to match the image below. This causes the CloudWatch alarm to trigger when the service's CPU utilization is greater than 75% for one minute.

Alarm threshold

| | | | |
|-----------|-------------------|------------------------|------------|
| Average ▼ | of CPUUtilization | > ▼ | 75 |
| for | 1 | consecutive periods of | 1 minute ▼ |

- d. Choose Save to save your alarm.
3. For Scaling action, enter the following information to match the image below. This causes your service's desired count to increase by 1 task when the alarm is triggered.

Scaling action Add ▾ 1 tasks ▾ when 75 < CPUUtilization < +infinity

4. For Cooldown period, enter 60 for the number of seconds between scaling actions and choose Save to save your ScaleOutPolicy.
5. For your scaling down enter Policy name, enter ScaleInPolicy
6. For Execute policy when, choose Create new alarm.
 - a. For Alarm name, enter sample-webapp-cpu-lt-25.
 - b. For ECS service metric, choose CPUUtilization.
 - c. For Alarm threshold, enter the following information to match the image below. This causes the CloudWatch alarm to trigger when the service's CPU utilization is less than 25% for one minute.

Alarm threshold Average ▾ of CPUUtilization < ▾ 25
for 1 consecutive periods of 1 minute ▾

- d. Choose Save to save your alarm.
7. For Scaling action, enter the following information to match the image below. This causes your service's desired count to decrease by 1 task when the alarm is triggered.

Scaling action Remove ▾ 1 tasks ▾ when 25 > CPUUtilization > -infinity

8. For Cooldown period, enter 60 for the number of seconds between scaling actions and choose Save to save your ScaleOutPolicy.

Trigger a Scaling Activity

After your service is configured with Service Auto Scaling, you can trigger a scaling activity by pushing your service's CPU utilization into the ALARM state. Because this example is a web application that is running behind a load balancer, you can send thousands of HTTP requests to your service (using the ApacheBench utility) to spike the service CPU utilization above our threshold amount. This spike should trigger the alarm, which in turn triggers a scaling activity to add one task to your service.

After the ApacheBench utility finishes the requests, the service CPU utilization should drop below your 25% threshold, triggering a scale in activity that returns the service's desired count to 1

To trigger a scaling activity for your service

1. From your service's main view page in the console, choose the load balancer name to view its details in the Amazon EC2 console. You need the load balancer's DNS name, which should look something like this: `EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-east-1.elb.amazonaws.com`.
2. Use the ApacheBench (**ab**) utility to make thousands of HTTP requests to your load balancer in a short period of time. To install ab on Ubuntu :

```
$ sudo apt-get install apache2-utils
```

Run the following command, substituting your load balancer's DNS name.

```
$ ab -n 100000 -c 1000 http://EC2Contai-EcsElast-SMAKV74U23PH-96652279.us-east-1.elb.amazonaws.com/
```

3. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
4. Choose Alarms in the left navigation pane.
5. Wait for your **ab** HTTP requests to trigger the scale out alarm in the CloudWatch console. You should see your Amazon ECS service scale out and add 1 task to your service's desired count.
6. Shortly after your **ab** HTTP requests complete (between 1 and 2 minutes), your scale in alarm should trigger and the scale in policy reduces your service's desired count back to 1.

NOTE : You should look at the Service Events which will tell you what is happening. For example below the Cloudwatch alarm was triggered requesting ECS to launch another task on the Cluster however the t2.micro instance did not have enough memory available to allow the container/task to run.

| Event Id | Event Time | Message |
|--------------------------------------|------------------------------|--|
| e2df845d-2c52-4773-b083-9b2bca21a465 | 2017-11-12 15:59:44 +0000 | service test was unable to place a task because no container instance met all of its requirements. The closest matching container-instance 85a9ee1f-c8f9-4c0a-9969-b9021ed3aa9b has insufficient memory available. For more information, see the Troubleshooting section . |
| 2f9d7c68-cd49-49fd-8c41-6827f9a224be | 2017-11-12 15:59:08 +0000 | Message: Successfully set desired count to 3. Waiting for change to be fulfilled by ecs. Cause: monitor alarm scaleupcontainer in state ALARM triggered policy ScaleUpTestContainer |
| c64bb534-c0cb-4c86-901c-e964cccadf3d | 2017-11-12 15:18:14 | service test has reached a steady state. |

Cleaning Up

When you have completed this exercise, you may choose to keep your cluster, Auto Scaling group, load balancer, and EC2 instances. However, if you are not actively using these resources, you should consider cleaning them up so that your account does not incur unnecessary charges.