Managing Applications at Scale with Amazon ECS

**SPL-DD-300-COAPSC-1 - Version 1.0.10**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

In this lab, you gain valuable insights into design considerations and scaling strategies for Amazon Elastic Container Service (Amazon ECS) applications and infrastructure. Your goal is to configure the ECS launch types, service capacity providers, and task scheduling strategies and constraints, that enables AnyCompany Cookies to optimize their application’s performance and cost-effectiveness. This hands-on experience equips you with the knowledge to make informed decisions when addressing scaling issues and designing robust Amazon ECS applications.

OBJECTIVES

By the end of this lab, you should be able to do the following:

* Deploy an application using Amazon ECS.
* Configure Amazon EC2 auto scaling group capacity providers.
* Optimize an ECS deployment using scaling and task placement strategies.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab, you should be familiar with basic navigation of the AWS Management Console and be comfortable editing scripts using a text editor.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Learn more:** Where to find more information.
* **Caution:** Information of special interest or importance (not so important to cause problems with the equipment or data if you miss it, but it could result in the need to repeat certain steps).
* **Refresh:** A time when you might need to refresh a web browser page or list to show new information.
* **Task complete:** A conclusion or summary point in the lab.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

**Caution:** You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**Warning:** Do not change the **Region** unless instructed.

COMMON SIGN-IN ERRORS

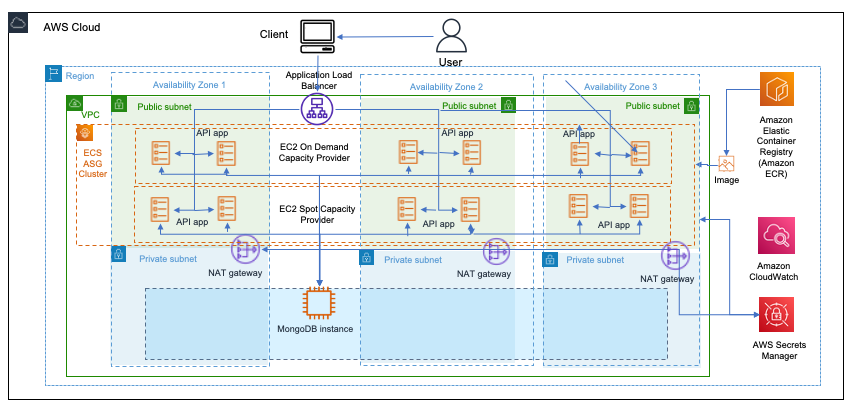
**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

LAB ENVIRONMENT

The following diagram shows the basic architecture of the lab environment:



*Image description: In the preceding diagram, the user interacts with a client that is connected to an Application Load Balancer (ALB). The ALB balances the requests between the availability zones that contain applications in an Amazon ECS Auto Scaling Group (ASG) cluster.*

The following list details the major resources in the diagram:

* A *Virtual Private Cloud (VPC)* with two *public subnet* and one *private subnet* spread across three Availability Zones.
* An *Application Load Balancer* that distributes inbound traffic to an *Amazon ECS Auto Scaling Group (ASG) cluster*.
* An *Amazon ECS Auto Scaling Group (ASG) cluster* that pulls and deploys an image from an *Amazon Elastic Container Registry* repository and connects to a *MongoDB database* running on an *EC2 instance*.

SERVICES USED IN THIS LAB

**Amazon Elastic Container Service (Amazon ECS)**

Amazon ECS is a highly scalable, fast container management service that makes it easy to run, stop, and manage containers on a cluster. Your containers are defined in a task definition that you use to run individual tasks or tasks within a service. In this context, a service is a configuration that enables you to run and maintain a specified number of tasks simultaneously in a cluster. You can run your tasks and services on a serverless infrastructure that is managed by AWS Fargate. Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon EC2 instances that you manage.

Amazon ECS enables you to launch and stop your container-based applications by using simple API calls. You can also retrieve the state of your cluster from a centralized service and have access to many familiar Amazon EC2 features.

Amazon ECS is a regional service that simplifies running containers in a highly available manner across multiple Availability Zones within a Region. You can create Amazon ECS clusters within a new or existing VPC. After a cluster is up and running, you can create task definitions that define which container images run across your clusters. Your task definitions are used to run tasks or create services. Container images are stored in and pulled from container registries, for example, the Amazon Elastic Container Registry.

**Amazon Elastic Container Registry (Amazon ECR)**

Amazon ECR is an AWS managed container image registry service that is secure, scalable, and reliable. Amazon ECR supports private repositories with resource-based permissions using AWS IAM. This is so that specified users or Amazon EC2 instances can access your container repositories and images. You can use your preferred CLI to push, pull, and manage Docker images, Open Container Initiative (OCI) images, and OCI compatible artifacts. Amazon Elastic Container Registry Public supports public image repositories with resource-based permissions using AWS IAM so that specific users can access your public repositories to push images. Your images are publicly available to pull, either anonymously or using an Amazon ECR public authentication token.

**AWS Auto Scaling**

AWS Auto Scaling is an AWS service that helps you optimize the performance of your applications while lowering infrastructure costs by easily and safely scaling multiple AWS resources. It simplifies the scaling experience by allowing you to scale collections of related resources that support your application with just a few selections.

AWS Auto Scaling helps you configure consistent and congruent scaling policies across the full infrastructure stack backing your application. AWS Auto Scaling will automatically scale resources as needed to align to your selected scaling strategy, so you maintain performance and pay only for the resources you actually need.

**Task 1: Review the task definition and Amazon ECR image**

In this task, you examine a pre-configured Task Definition and an *Amazon Elastic Container Registry (ECR)* image. The *ECR* image defines the AnyCompany Cookies e-commerce application, a simple webapp for an online cookie store.

Task Definitions are JSON-formatted text files that describe one or more containers (up to a maximum of ten) that make up your application, essentially serving as a blueprint. Some of the parameters included in the Task Definition are containers, launch type, ports for your application to run on, and data volumes to be used with the containers in the task.

By reviewing the Task Definition and *ECR* image, you gain a better understanding of the application’s structure and requirements, preparing you for the subsequent tasks involving scaling strategies and design considerations for *Amazon ECS* applications and infrastructure.

1. At the top of the AWS Management Console, in the search bar, search for and choose

Elastic Container Service

.

1. In the left navigation pane, under **Amazon Elastic Container Service**, choose **Task Definition**.
2. Choose the **cookie-app** task definition.
3. Choose the most recent revision with **cookie-app** in its name.
4. Open the **JSON** tab in the middle of the screen to view the parameters and the container image that is used for the AnyCompany application.

**Learn more:** The following list provides a high-level overview Task Definition contents:

* + *Task Definition ARN*: The unique identifier for the Task Definition, which is associated with the cookie-app.
  + *Container Definitions*: Specifies the container settings, including the name (cookie-app), image (AnyCompany Cookies e-commerce application), CPU (10 units), and memory (300 MiB).
  + *Port Mappings*: Configures the container to listen on port 80 (TCP protocol) and dynamically assigns a host port.
  + *Essential*: Indicates that the container is essential for the task to run successfully.
  + *Task Role ARN*: Specifies the IAM role associated with the task, granting necessary permissions.
  + *Requires Attributes*: Lists the required capabilities for the task, such as ECR authentication, Docker Remote API versions, and IAM role support.
  + *Compatibilities*: Specifies the launch types compatible with the Task Definition, which are “EXTERNAL” and “EC2” in this case.
  + *Requires Compatibilities*: Indicates that the Task Definition requires the “EC2” launch type.

Now let’s take a closer look at the container image used in the application.

1. In the left navigation pane, choose **Amazon ECR**  icon.

You are brought to the *Amazon Elastic Container Registry* console.

1. If the *Amazon Elastic Container Registry* panel on the left side of the screen is not already open, choose the  menu icon to expand it.
2. In the left navigation pane, under **Private registry**, choose **Repositories**.

The cookie application is the only image repository in the account.

1. Choose the private repository with **cookieapp** in its name.
2. Choose the **latest** image tag to see the details of the docker image.

**Task complete:** You have now successfully reviewed the task definition and the Amazon ECR image.

**Task 2: Deploy AnyCookie application using Amazon ECS**

In this task, you create an *ECS* cluster and deploy the AnyCookie application into it.

TASK 2.1: CREATE AN AMAZON ECS CLUSTER

1. At the top of the AWS Management Console, in the search bar, search for and choose

Elastic Container Service

.

1. In the left navigation pane, choose **Clusters**.
2. Choose **Create cluster**.
3. On the **Create cluster** page:
   * In the **Cluster configuration** section:
     + For **Cluster name**, enter

CookieAppCluster

.

* + In the **Infrastructure** section:
    - Uncheck **AWS Fargate (serverless)**.
    - Choose **Amazon EC2 instances**.
    - For **Auto Scaling group (ASG)**, select **ECSOnDemandAutoScalingGroup** from the drop-down menu.
  + Expand the **Monitoring - *optional*** section and enable the **Use Container Insights** toggle icon.

1. Choose **Create**.

It takes several minutes to deploy the cluster.

1. Wait for the cluster to finish building and then choose the **CookieAppCluster** link.

You are brought to the *CookieAppCluster* page which displays information about the cluster configuration.

TASK 2.2: CREATE A CAPACITY PROVIDER

1. Take a moment to review the information displayed in the **Cluster overview** card. Note the following:
   * The cluster is in **Active** status.
   * For CloudWatch monitoring, **Container Insights** has been enabled.
   * The cluster does not contain any *Registered container instances*, *Services*, or *Tasks*.
2. Scroll to the bottom of the page and choose the **Infrastructure** tab.

**Learn more:** The Infrastructure tab displays information about capacity providers and EC2 instances hosting containers. Capacity providers are a fundamental component in AWS ECS that manage the underlying compute resources used to run your containerized applications. They serve as an abstraction layer between your ECS service and the actual infrastructure, such as EC2 instances or Fargate, allowing you to define how tasks are allocated and scaled across these resources. By using capacity providers, you can seamlessly integrate with various AWS services, such as Auto Scaling groups, to ensure optimal resource utilization, cost efficiency, and high availability for your applications.

1. Choose the **Create** button to add a second capacity provider.
2. On the **Create capacity providers** page:
   * For **Capacity provider name**, enter

SpotCapacityProvider

.

* + For **Auto Scaling group (ASG)**, select **ECSSpotAutoScalingGroup** from the drop-down menu.
  + Expand the **Scaling policies - *new, optional*** section and confirm that the following options are selected:
    - **Turn on managed instance draining**.

**Learn more:** Managed instance draining facilitates graceful termination of Amazon EC2 instances. This allows your workloads to stop safely and be rescheduled to non-terminating instances.

* + - **Turn on managed scaling**.

**Learn more:** By enabling *managed scaling*, you have given the ESC service control over an EC2 auto scaling group called *ECSSpotAutoScalingGroup*. Similarly, the other capacity provider is also configured to allow enable EC2 to manage scaling for the *ECSOnDemandAutoScalingGroup* auto scaling group. Together, these capacity providers allows ECS to increase the number of EC2 instances hosting tasks for your application.

1. Choose **Create**.

**Note:** Wait for the **SpotCapacityProvider** deployment to complete before continuing to the next task.

TASK 2.3: CREATE AN AMAZON ECS SERVICE

Now that you’ve created your cluster and capacity providers, it’s time to create the *Amazon ECS* service and assign it a capacity provider strategy that is used to deploy the AnyCookie Application.

1. Choose the **Services** tab.

**Learn more:** An Amazon ECS service is a managed, scalable, and highly available component that allows you to run and maintain a specified number of tasks simultaneously. It ensures that the desired count of tasks is consistently running and automatically replaces any failed tasks to maintain the desired state.

1. On the **Services** tab, choose **Create**.

Start by creating a capacity provider strategy. Capacity provider strategies are a key concept that helps to ensure high availability, fault tolerance, and efficient resource utilization for your applications. In the following steps, you select the two capacity providers you viewed earlier in this task and configure them to work together in a single strategy.

1. On the **Create** page, in the **Environment** section:
   * For **Compute options**, ensure **Capacity provider strategy** is selected.
   * For **Capacity provider strategy**, select **Use custom (Advanced)**.
   * For **Capacity provider**, ensure **Infra-ECS-Cluster-CookieAppCluster** is selected.
   * For **Base**, enter

1

.

* + For **Weight**, enter

1

.

1. Choose the **Add capacity provider** icon to add a second capacity provider to the service.
   * For **Capacity provider**, select **SpotCapacityProvider** from the drop-down menu.
   * For **Base**, enter

0

.

* + For **Weight**, enter

3

.

1. **Note:** The *Infra-ECS-Cluster-CookieAppCluster* capacity provider has a base of 1 and a weight of 1. This configuration ensures that at least one task is always running in the on-demand Auto Scaling group and provides a stable and predictable level of availability for your application. On-demand instances offer a fixed cost and are ideal for workloads with consistent resource requirements.
2. The *SpotCapacityProvider* capacity provider has a base of 0 and a weight of 3, and it uses EC2 spot instances. With a base of 0, there is no minimum number of tasks required to run using this capacity provider. However, the weight of 3 indicates that a higher proportion of tasks are distributed to this capacity provider compared to the first one. EC2 spot instances allow you to take advantage of unused EC2 capacity at a significantly lower cost compared to on-demand instances, making them suitable for fault-tolerant and flexible workloads.
3. By using these two capacity providers, you can strike a balance between cost efficiency and availability for your application. The first capacity provider ensures a minimum level of stability with on-demand instances, while the second capacity provider optimizes cost savings by utilizing a higher proportion of tasks on EC2 Spot Instances.
4. In the **Deployment configuration** section:
   * For **Family**, select **cookie-app** from the drop-down menu.

**Learn more:** The family is a logical grouping of multiple versions of the same task definition. Task definition families allow you to easily manage and track updates to your application’s container configurations.

* + Confirm that revision is set to **(LATEST)**. This ensures that the most current version of the application is deployed.
  + For **Service name**, enter

CookieAppService

.

* + For **Service type**, choose **Replica**.

**Learn more:** By choosing to use a replica service type, you are opting for a service configuration that creates and maintains an exact number of task instances based on the desired task count. This ensures that your application has a consistent and predictable number of running tasks, providing high availability and fault tolerance. Replica service types are ideal for stateless applications, where each task instance can independently handle incoming requests without relying on shared state or data.

* + For **Desired tasks**, enter

2

.

1. Expand the **Load balancing - *optional*** section, and then:
   * For **Load balancer type**, choose **Application load balancer**.
   * For **Load balancer**, choose **CookieAppALB**.

**Note:** This load balancer was pre-created for your use. Outside this lab environment, load balancing can be a configuration you might include in ECS depending on your use-case.

* + For **Listener**, choose **Use an existing listener**.
    - Select **80:HTTP** from the drop-down menu.
  + For **Target group**, choose **Use an existing target group**.
    - Select **CookieAppTargetGroup** from the drop-down menu.

1. Expand the **Service auto scaling - *optional*** section, and then:
   * Select **Use service auto scaling**.
   * For **Minimum number of tasks**, enter

1

.

* + For **Maximum number of tasks**, enter

20

.

* + For **Scaling policy type**, ensure **Target tracking** is selected.
  + For **Policy name**, enter

CpuTargetScaling

.

* + For **ECS service metric**, choose **ECSServiceAverageCPUUtilization** from drop-down menu.
  + For **Target value**, enter

60

.

1. Expand the **Task Placement** section, and then:
   * For **Placement templates**, select **Binpack** from drop-down menu.

**Learn more:** The *binpack template* places tasks on container instances so as to leave the least amount of unused resources. In this case, tasks are placed on container instances based on available memory, aiming to minimize the amount of unused memory across all instances.

1. Leave the rest of the values at the default settings and choose **Create**.

**Learn more:** This opens the Launch Status page. A message is displayed stating that the deployment is in progress.

1. Occasionally choose  **Refresh** to see the latest status of the service creation.

After a few minutes, on the **Services** tab - the **Last deployment** column shows as **Completed**.

**Note:** Wait until the **Last deployment** status shows as **Completed** before continuing to the next steps.

TASK 2.4: VIEW THE APPLICATION

Now that your cluster has tasks running in a service, it’s time to test the AnyCookie Application in a browser.

1. With the **Services** tab selected, choose the **CookieAppService** link.

You are brought to the *CookieAppService* page.

1. Confirm that the **Health and metrics** tab is selected and then on the **Status** page, choose the **View load balancer**.

The *CookieAppALB* page opens in a new tab.

**Caution:** Do not close the tab connected to the ECS console, as you have to return to it in subsequent steps.

1. Review the information displayed on screen. Note that the application load balancer is **Internet-facing** and spread across three Availability Zones.
2. On the **CookieAppALB** page, copy the **DNS name** (It looks like the following: *CookieAppALB-283553955.eu-west-1.elb.amazonaws.com*) to your clipboard.
3. In a new browser tab, paste and enter the **DNS name**.
4. Navigate through the application to see the various cookies and macaroons available for sale.

**Task complete:** You have now successfully deployed the AnyCookie application using Amazon ECS.

**Task 3: Add load to test and observe scaling**

In this task, you add load to test the cookie application and learn how task placement and auto scaling can be optimized to meet your application’s requirements.

TASK 3.1: LOAD TESTING

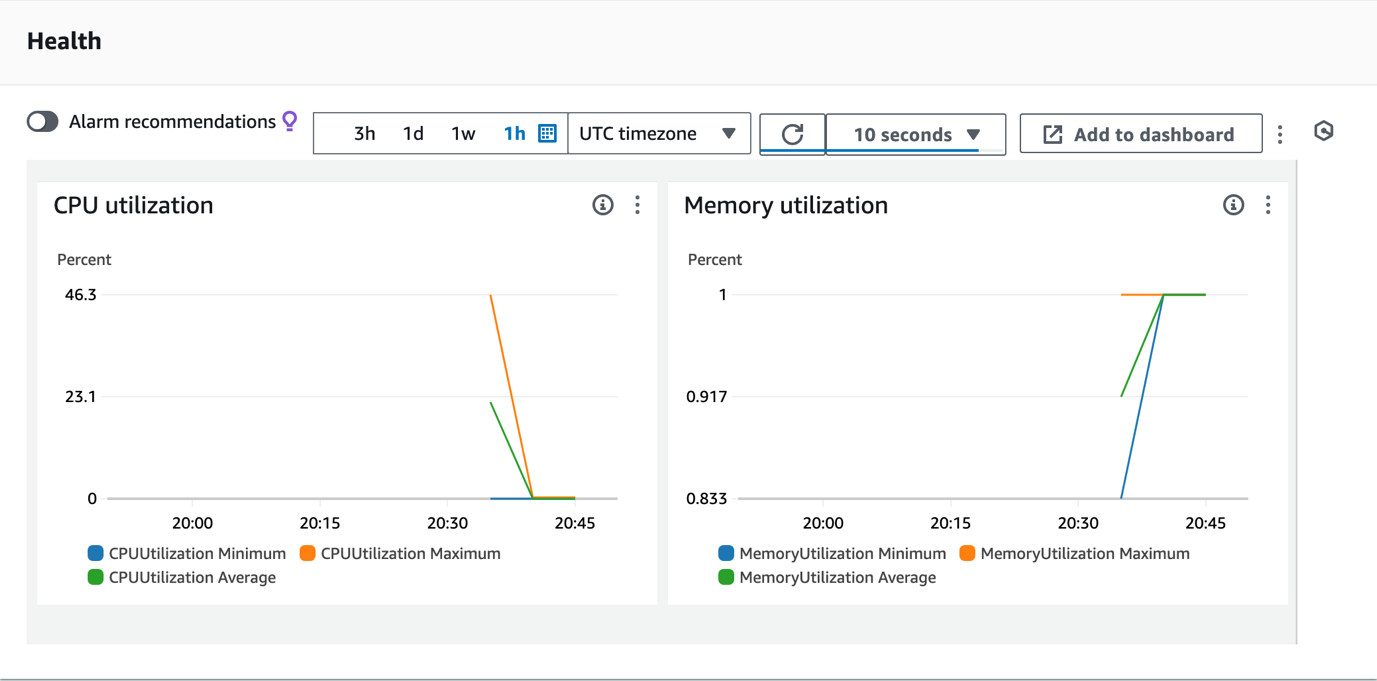
Start by observing the infrastructure currently deployed in your cluster.

1. Return to your web browser tab with the **ECS** console.
2. In the breadcrumbs at the top of the page, choose **CookieAppCluster**.
3. Open the **Infrastructure** tab and scroll down to the *Container instances* card. Note that there are 4 container instances running, but that tasks are only running on 2 of the instances.

Now that you’ve identified the instances that are hosting your ECS tasks, let’s see what happens when additional load is placed on them. We’ll start by establishing baselines for CPU and memory usage.

1. Choose the **Services** tab and then select **CookieAppService**.
2. With the **Health and metrics** tab selected, scroll to the bottom of the page and review the *CPU utilization* chart.
3. Locate the *time series bar* above the *CPU utilization* and *Memory utilization* charts. Choose the icon beside **1w** and select **1 hour** period from the second row.
4. Choose the  dropdown menu on the right side of the  icon and select **10 seconds**. This sets the charts to refresh every 10 seconds.

**Note:** If the chart displays either a message stating **No data available. Try adjusting the dashboard time range** or shows a few dots rather than lines, your cluster has not yet reported enough data to display meaningful metrics. Wait a little longer and allow the chart to refresh with more data.



*In the preceding image, the CPU utilization and Memory utilization levels settle to very low baselines.*

**Learn more:** The chart indicates that after an initial spike in CPU consumption when the tasks were launched, utilization settled to a low baseline. In the following steps, you use *Siege* to perform application load testing and force your service to add additional tasks. *Siege* is an open source regression test and benchmarking utility. It is commonly used to perform load testing. You can stress test a single URL by defining the number of users in the *Siege* command to simulate the users.

Next, you connect to an **AWS Cloud9** environment that has been pre-populated with the application source code.

1. Copy the **Cloud9Environment** value that is listed to the left of these instructions. Paste the URL into a new web browser tab and press **Enter** to open the **AWS Cloud9 console**.

You are redirected to an AWS Cloud9 environment.

**Learn more:** AWS Cloud9 is a cloud-based integrated development environment (IDE) that you can use to write, run, and debug your code within your browser. It comes prepackaged with many tools that are commonly used in application development, including Docker, Python, and the AWS Command Line Interface (AWS CLI).

1. You do not need the **Cloud9 Welcome screen** or any of the other default tabs that appear when you first launch **Cloud9**, so choose the  next to each tab to close them.
2. **Command:** In the **AWS Cloud9** terminal, run the below command to download the latest version of *Siege* from the official website and save it in the */opt/* directory:

sudo wget http://download.joedog.org/siege/siege-latest.tar.gz --directory-prefix=/opt/

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--2024-05-21 18:53:13-- http://download.joedog.org/siege/siege-latest.tar.gz

Resolving download.joedog.org (download.joedog.org)... 104.21.4.201, 172.67.132.100, 2606:4700:3034::6815:4c9, ...

Connecting to download.joedog.org (download.joedog.org)|104.21.4.201|:80... connected.

HTTP request sent, awaiting response... 301 Moved Permanently

Location: https://download.joedog.org/siege/siege-latest.tar.gz [following]

--2024-05-21 18:53:13-- https://download.joedog.org/siege/siege-latest.tar.gz

Connecting to download.joedog.org (download.joedog.org)|104.21.4.201|:443... connected.

HTTP request sent, awaiting response... 200 OK

Length: 527103 (515K) [application/x-gzip]

Saving to: ‘/opt/siege-latest.tar.gz’

siege-latest.tar.gz 100%[==============================================================================>] 514.75K --.-KB/s in 0.03s

2024-05-21 18:53:13 (16.5 MB/s) - ‘/opt/siege-latest.tar.gz’ saved [527103/527103]

1. **Command:** Navigate to the */opt/* directory, extract the *siege-latest.tar.gz* archive, and then change to the extracted *siege-4.1.6/* directory by running the following command:

cd /opt/ && sudo tar -zxf siege-latest.tar.gz && cd siege-4.1.6/

**Expected output:**

*None, unless there is an error.*

1. **Command:** To configure and prepare Siege for installation, run the following command:

sudo ./configure

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

checking for pthread\_attr\_init in -lpthread... yes

checking that generated files are newer than configure... done

configure: creating ./config.status

config.status: creating Makefile

config.status: creating src/Makefile

config.status: creating doc/Makefile

config.status: creating html/Makefile

config.status: creating include/Makefile

config.status: creating include/joedog/Makefile

config.status: creating utils/Makefile

config.status: creating include/config.h

config.status: executing depfiles commands

config.status: executing default-1 commands

config.status: executing default-2 commands

config.status: executing default-3 commands

config.status: executing default-4 commands

config.status: executing default-5 commands

config.status: executing default-6 commands

--------------------------------------------------------

Configuration is complete

Run the following commands to complete the installation:

make

make install

For complete documentation: http://www.joedog.org

--------------------------------------------------------

1. **Command:** To compile and install Siege, run the following command:

sudo make && sudo make install

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

/usr/bin/install -c siege /usr/local/bin/siege

make[2]: Nothing to be done for 'install-data-am'.

make[2]: Leaving directory '/opt/siege-4.1.6/src'

make[1]: Leaving directory '/opt/siege-4.1.6/src'

Making install in utils

make[1]: Entering directory '/opt/siege-4.1.6/utils'

make[2]: Entering directory '/opt/siege-4.1.6/utils'

make install-exec-hook

make[3]: Entering directory '/opt/siege-4.1.6/utils'

/bin/sh ../utils/mkinstalldirs /usr/local/bin

/bin/sh ../libtool --mode=install /usr/bin/install -c bombardment /usr/local/bin/bombardment

/usr/bin/install -c bombardment /usr/local/bin/bombardment

/bin/sh ../libtool --mode=install /usr/bin/install -c siege2csv.pl /usr/local/bin/siege2csv.pl

/usr/bin/install -c siege2csv.pl /usr/local/bin/siege2csv.pl

/bin/sh ../libtool --mode=install /usr/bin/install -c siege.config /usr/local/bin/siege.config

/usr/bin/install -c siege.config /usr/local/bin/siege.config

make[3]: Leaving directory '/opt/siege-4.1.6/utils'

make[2]: Nothing to be done for 'install-data-am'.

make[2]: Leaving directory '/opt/siege-4.1.6/utils'

make[1]: Leaving directory '/opt/siege-4.1.6/utils'

Making install in doc

make[1]: Entering directory '/opt/siege-4.1.6/doc'

make[2]: Entering directory '/opt/siege-4.1.6/doc'

make install-exec-hook

make[3]: Entering directory '/opt/siege-4.1.6/doc'

make[3]: Leaving directory '/opt/siege-4.1.6/doc'

/usr/bin/mkdir -p '/usr/local/share/man/man1'

/usr/bin/install -c -m 644 siege.1 siege.config.1 bombardment.1 siege2csv.1 '/usr/local/share/man/man1'

make[2]: Leaving directory '/opt/siege-4.1.6/doc'

make[1]: Leaving directory '/opt/siege-4.1.6/doc'

Making install in html

make[1]: Entering directory '/opt/siege-4.1.6/html'

make[2]: Entering directory '/opt/siege-4.1.6/html'

make install-exec-hook

make[3]: Entering directory '/opt/siege-4.1.6/html'

HTML pages not installed

make[3]: Leaving directory '/opt/siege-4.1.6/html'

make[2]: Nothing to be done for 'install-data-am'.

make[2]: Leaving directory '/opt/siege-4.1.6/html'

make[1]: Leaving directory '/opt/siege-4.1.6/html'

1. **Command:** To verify the installed version of Siege, run the following command:

siege --version

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

New configuration template added to /home/ec2-user/.siege

Run siege -C to view the current settings in that file

SIEGE 4.1.6

Copyright (C) 2023 by Jeffrey Fulmer, et al.

This is free software; see the source for copying conditions.

There is NO warranty; not even for MERCHANTABILITY or FITNESS

FOR A PARTICULAR PURPOSE.

1. **Command:** Enter the following command to view the current settings:

siege -C

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CURRENT SIEGE CONFIGURATION

Mozilla/5.0 (pc-x86\_64-linux-gnu) Siege/4.1.6

Edit the resource file to change the settings.

----------------------------------------------

version: 4.1.6

verbose: true

color: true

quiet: false

debug: false

protocol: HTTP/1.1

HTML parser: enabled

get method: HEAD

connection: close

concurrent users: 25

time to run: n/a

repetitions: n/a

socket timeout: 30

cache enabled: false

accept-encoding: gzip, deflate

delay: 0.000 sec

internet simulation: false

benchmark mode: false

failures until abort: 1024

named URL: none

URLs file: /usr/local/etc/urls.txt

thread limit: 255

logging: false

log file: /usr/local/var/log/siege.log

resource file: /home/ec2-user/.siege/siege.conf

timestamped output: false

comma separated output: false

allow redirects: true

allow zero byte data: true

allow chunked encoding: true

upload unique files: true

json output: false

no-follow:

- ad.doubleclick.net

- pagead2.googlesyndication.com

- ads.pubsqrd.com

- ib.adnxs.com

proxy auth:

www auth:

The current configuration only permits a maximum of 255 threads. In the next step, you increase the number of threads and concurrent connections.

1. **Command:** Enter the following command update the Siege configuration so that it permits a large number of concurrent requests:
2. sed -i 's/limit = 255/limit = 1023/g' ~/.siege/siege.conf

sed -i 's/concurrent = 25/concurrent = 750/g' ~/.siege/siege.conf

**Expected output:**

*None, unless there is an error.*

1. **Command:** Save the Application load balancer’s DNS name to a shell variable:
2. ALB=$(aws elbv2 describe-load-balancers | jq '.LoadBalancers[].DNSName' -r)

echo "The load balancer URL is $ALB"

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The load balancer URL is CookieAppALB-449336801.us-west-2.elb.amazonaws.com

1. **Command:** Enter the following command to use *Siege* to simulate load placed on the Application Load Balancer:

siege $ALB

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* EXAMPLE OUTPUT \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

\*\* SIEGE 4.1.1

\*\* Preparing 750 concurrent users for battle.

The server is now under siege...

HTTP/1.1 200 0.05 secs: 817 bytes ==> GET /

HTTP/1.1 200 0.06 secs: 817 bytes ==> GET /

HTTP/1.1 200 0.05 secs: 817 bytes ==> GET /

HTTP/1.1 200 0.06 secs: 817 bytes ==> GET /

Now that your load test is underway, return to the ECS console to observe how your application behaves.

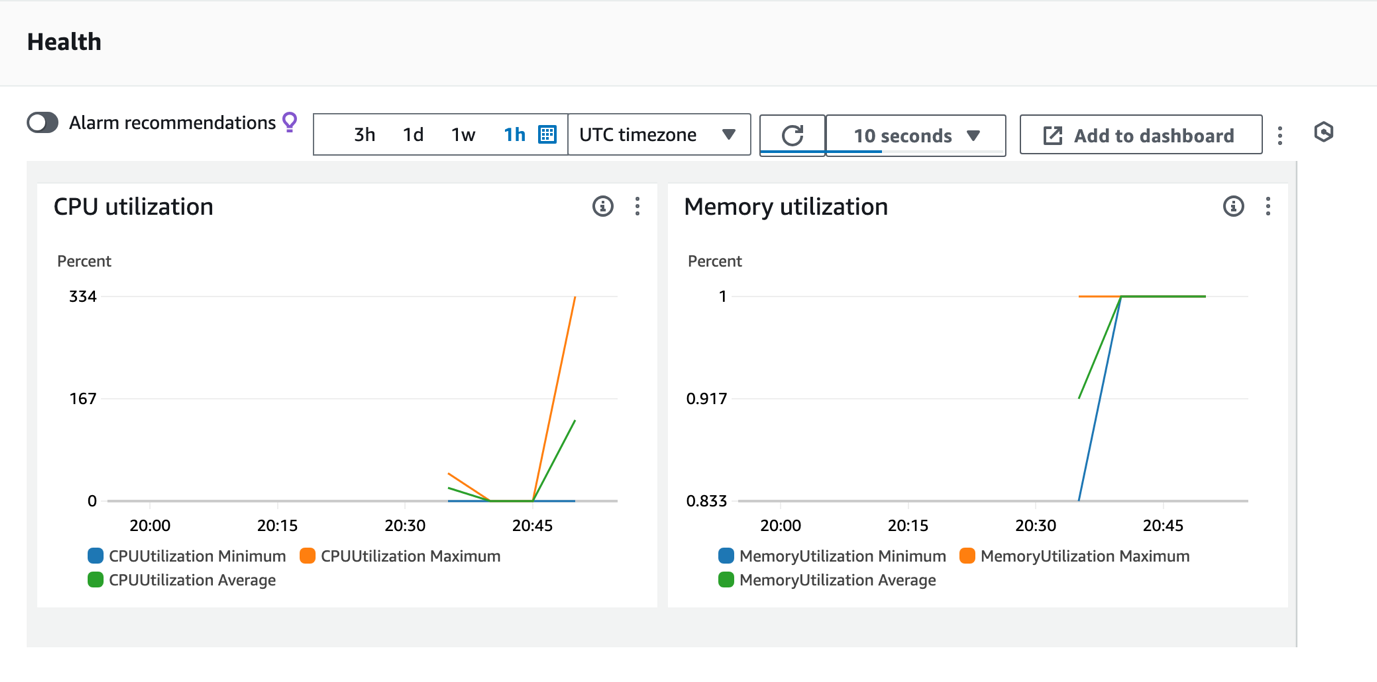
1. Open a new browser tab and navigate to the AWS Management Console. In the search bar, search for and choose

Elastic Container Service

.

1. Choose **CookieAppCluster**.
2. In the **Services** tab, choose **CookieAppService**.
3. Choose the **Health and metrics** tab. In the *Status* card at the top of the screen, note that only a couple tasks are running and no tasks are pending.
4. Locate the *time series bar* above the *CPU utilization* and *Memory utilization* charts. Choose the icon beside **1w** and select **1 hour** period from the second row.
5. Choose the  dropdown menu on the right side of the  icon and select **10 seconds**. This sets the charts to refresh every 10 seconds.
6. Wait for the **CPU utilization** and **Memory utilization** charts to register the increased load. Typically, this process takes about 10 minutes.

**Note:** The ***CPUUtilization Average*** chart spikes, far exceeding the 60% threshold defined in the scaling policy.



*In the preceding image, the CPUUtilization Average chart shows a very rapid increase to over 100%.*

1. Look again at the *Status* card at the top of the screen and choose the **Refresh** to update the information in the card. Note that there are now additional tasks in **Running** or **Pending** state.
2. At the top of the AWS Management Console, in the search bar, search for and choose

CloudWatch

.

1. In the left navigation pane, expand **Alarms** and then choose **All alarms**.

**Note:** Notice that 6 *Cloudwatch Alarms* have been created and that some of them are currently in *Alarm* state. These *CloudWatch Alarms* were created automatically and are associated with target tracking scaling policies for both the *ECS* service and the underlying *EC2 Auto Scaling Groups* using on-demand and spot instances. When these alarms enter *In Alarm* state, your ECS cluster responds by adding more tasks or scaling the underlying *EC2 Auto Scaling groups*.

**Refresh:** The conditions that trigger these alarms differ slightly. Therefore, it may take slightly longer for some of the alarms to enter *In Alarm* state. Choose the **Refresh** button at the top of the screen to update their status.

1. At the top of the AWS Management Console, in the search bar, search for and choose

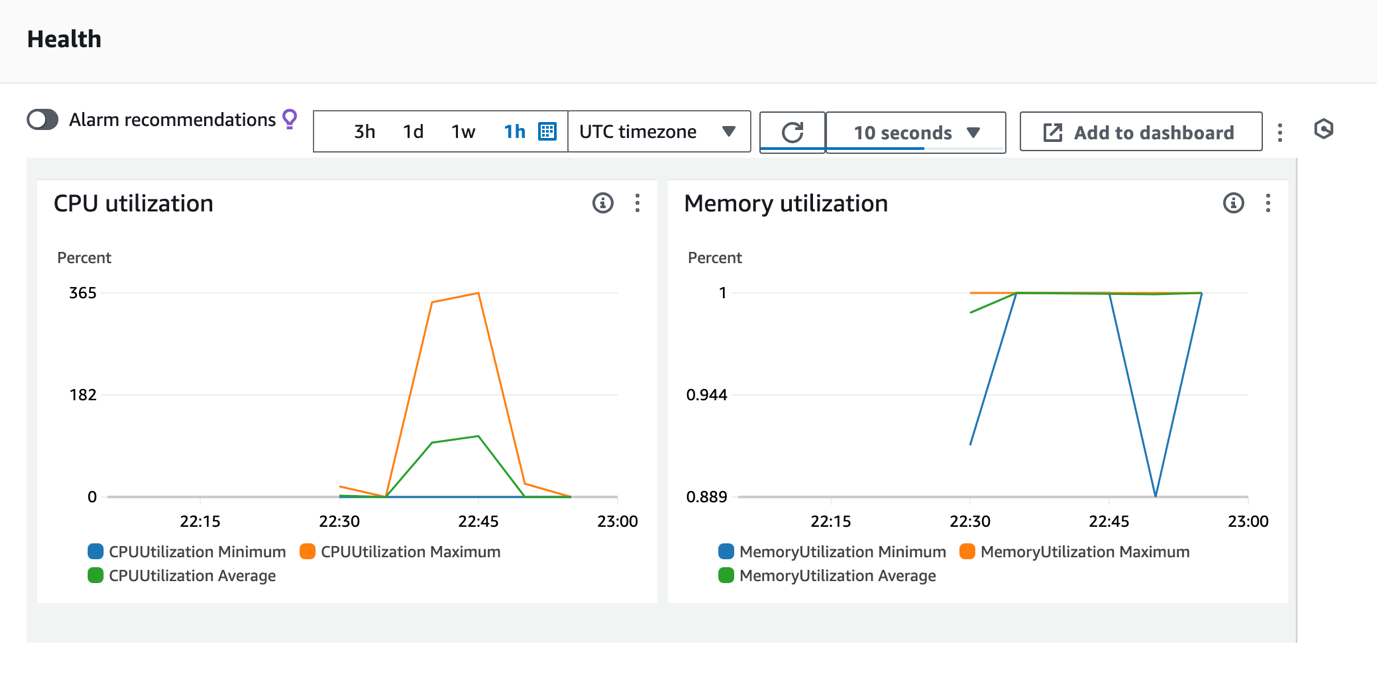
Elastic Container Service

 to return to the ECS console.

1. Choose **CookieAppCluster** and then select the **Tasks** tab to view the newly launched tasks.
2. Wait a few minutes and then choose the **Refresh** at the top of the **Tasks** card. Confirm that there are no tasks in **Pending** state.

Now confirm that the service CPU utilization has returned the level you configured in your scaling policy.

1. Open the **Services** tab and review the **CPU utilization** chart in the **Health** tab.



*In the preceding image, the CPU utilization chart shows that after the service scaled out, CPU usage dropped back to its expected level.*

**Task complete:** You have successfully used managed scaling to improve the performance of your cluster.

**Conclusion**

You have successfully done the following:

* Deployed an application using Amazon ECS.
* Configured Amazon EC2 auto scaling group capacity providers.
* Optimized an ECS deployment using scaling and task placement strategies.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

* [Amazon ECS task definitions](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definitions.html)
* [Create a cluster](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-ec2-cluster-console-v2.html)
* [Creating a capacity provider](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-capacity-provider-console-v2.html)
* [Create a service](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-service-console-v2.html#create-default-service)
* [Available metrics and dimensions for Amazon ECS](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/available_cloudwatch_metrics.html)

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).