Building and Deploying Containers Using Amazon Elastic Container Service

**SPL-208 - Version 1.1.16**

© 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**

This lab demonstrates the use of Amazon Elastic Container Service to host a simple multi-component web application composed of a website with two supporting API services. The website displays a form where you compose a story with placeholders for *nouns*, *verbs* and *adjectives*. When you choose the *submit* button, the *words API* is queried for the words needed in order to fill in all the placeholders in the story text. You can then choose *save* which utilize the *save API* to persist your creation to Amazon DynamoDB. The app is called **Storyizer**

You first build the Docker container for each component of the web app on a *command host*. Then you push them to the Amazon Elastic Container Repository (ECR) so they can be retrieved when the ECS cluster is built.

At that point you launch a CloudFormation template which builds the ECS Cluster with an [ECS Service](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_services.html) defined for each of the three components of your web application. Each service is configured to maintain two running tasks (*task* is the definition to run a given Docker container). This results in a highly available design since, if a service task becomes unhealthy, ECS replaces it with a newly launched task automatically. ECS also coordinates dynamic host port mapping with the Application Load Balancer (ALB) and each ECS task. This allows you to run more than one container of an app component on a single host without port conflicts.

OBJECTIVES

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

* Understand the steps needed to build docker images.
* Push container images to an Amazon ECR repository.
* Deploy containers from a repository to an Amazon ECS cluster as Services.

TECHNICAL KNOWLEDGE PREREQUISITES

This lab requires:

* Access to a notebook computer with Wi-Fi running Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat)
* For Microsoft Windows users: Administrator access to the computer
* An Internet browser such as Chrome, Firefox, or IE9 or greater (previous versions of Internet Explorer are not supported)

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.
* **Caution:** Information of special interest or importance (not important enough to cause problems with equipment or data if you miss it, but it could result in the need to repeat certain steps).
* **Learn more:** Where to find more 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: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

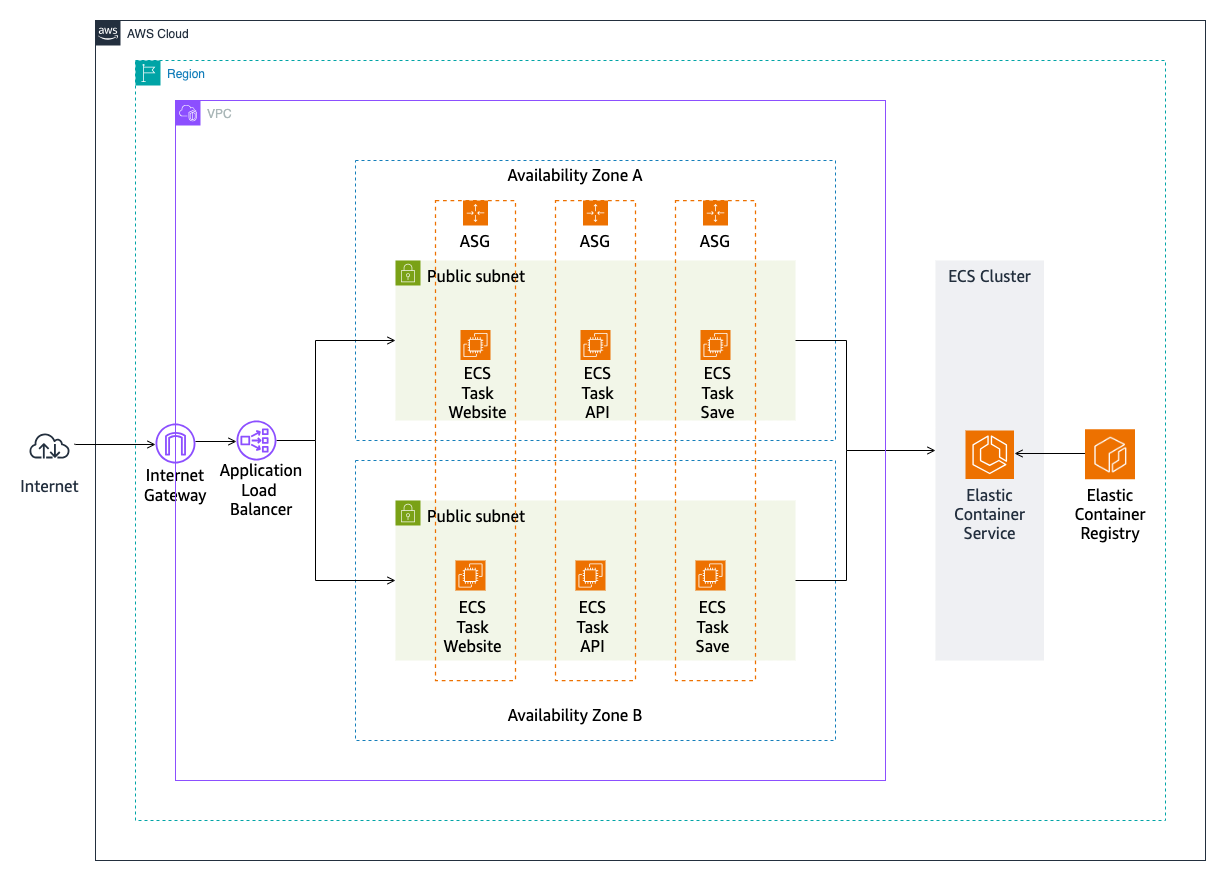
**Error: Choosing Start Lab has no effect**

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: The preceding diagram depicts the data flow from an external user accessing an application running inside of a Virtual Private Cloud (VPC) via an Internet Gateway (IGW). An Application Load Balancer is being used to distribute traffic to a container application hosted on an Elastic Compute Cloud (EC2) Instance. The application consists of the website, api, and save tasks definitions and uses Elastic Container Service (ECS) as the orchestration service. Each task is part of its own Auto Scaling Group that is spread across 2 Public subnets. Elastic Container Registry acts as the registry for storing the container images used for the deployed container by ECS.*

The following list details the major resources already provisioned for your use:

* An *Amazon Virtual Private Cloud (Amazon VPC)* with two *public subnets* in separate Availability Zones.
* A *Cloud9* environment.

AWS SERVICES NOT USED IN THIS LAB

AWS service capabilities used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

**Task 1: Login to the Command Host**

CONNECT TO EC2 USING SSM

In this task, you connect to your Amazon EC2 Instance using AWS Systems Manager Session Manager.

1. In the left navigation pane, copy the value of **InstanceSessionURL**, and paste it into a new tab.

**Note:** A terminal session with your Command Host instance opens in a new tab. A series of startup scripts have run in the background, setting environment variables that you use in subsequent tasks.

**Task complete:** You have successfully connected to a terminal session on the lab EC2 instance.

**Task 2: Exploring the Dockerfile**

Files for the Storyizer application have been pre-loaded on the Command Host for your use during the lab.

1. **Command:** Run this command in your session to navigate to the lab resources directory:
2. cd lab-resources/Code/

ls -ls

**Expected output:**

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

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

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

total 0

0 drwxr-xr-x 3 ssm-user ssm-user 53 Mar 1 17:19 API

0 drwxr-xr-x 3 ssm-user ssm-user 53 Mar 26 19:24 Save

0 drwxr-xr-x 3 ssm-user ssm-user 53 Mar 27 18:46 WebSite

**Note:** As you can see from the files, the Storyizer’s application is broken into different layers:

* + The **WebSite** is a simple web app running on apache.
  + The **API** service is an Express Nodejs api layer for looking up the nouns, verbs, and adjectives.
  + The **Save** service is also an Express Nodejs api layer handling save requests and persisting to DynamoDb.

Inside each of these directories, there is a **Dockerfile** and the latest version of the application code. You begin by configuring the main website.

1. **Command:** Run these commands to navigate to the Website directory and view the Dockerfile:
2. cd WebSite

cat Dockerfile

**Note:** The Dockerfile contains instructions needed to build the Storyizer application environment. You can see it is pulling a base image of Rocky Linux, followed by declaring some environment variables to be used in this containerized application. Rocky Linux is an open-source enterprise operating system binary-compatible with Red Hat Enterprise Linux (RHEL). It is designed to be a community-driven, free alternative to RHEL, allowing users to run the same applications and workloads without the need for a paid RHEL subscription. Rocky Linux is often used as a base operating system for containerized applications, providing a stable and secure environment for running containers.

**Expected output:**

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

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

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

FROM rockylinux/rockylinux

ARG ELBDNS

ENV ServerName=Storyizer-site ELBDNS=${ELBDNS}

RUN yum -y update && \

yum -y install httpd unzip && \

yum clean all

# Install app

COPY ./code/ /var/www/html/

# Config App

RUN echo "ServerName storyizer.training " >> /etc/httpd/conf/httpd.conf \

&& sed -i -- "s|APIELB|$ELBDNS|g" /var/www/html/js/env.js \

&& sed -i -- "s|SaveELB|$ELBDNS|g" /var/www/html/js/env.js

EXPOSE 80

ENTRYPOINT ["/usr/sbin/httpd", "-D", "FOREGROUND"]

**Learn more:** The following list describes the purpose of the commands seen in the Dockerfile:

* + *ARG, ENV & ELBDNS*: In this container, you want the ability to define a variable at both build and runtime, so both *ARG* and *ENV* dockerfile commands are utilized. *ARG* can be passed via the *–build-arg* argument. *ENV* variables are global variables available at runtime just like */bin/bash $HOME* would be in a linux shell.
  + *# Install app*: The copy command under *# Install app* in the Dockerfile is used to copy a directory into the container. In this case, the source code is copied into the container.
  + *# Config App*: The run commands under *#Config App* are instructions to inject build time configuration values into the Docker Container via additions and substitutions into some of the web app’s configuration files.
  + *EXPOSE & ENTRYPOINT*: Lastly, the port this container is going to listen to for inbound requests is noted with *EXPOSE*. *ENTRYPOINT* specifies a command that is always executed when the container starts.

**Task complete:** You have successfully explored the Dockerfile for the Storyizer website.

**Task 3: Building and testing the WebSite container**

In this task, you engage with the process of creating a Docker container image for a website application called “Storyizer.” This application is destined for deployment to an Amazon Elastic Container Service (ECS) cluster, but before reaching that stage, you must package it into a Docker container image. This image encapsulates the application code, dependencies, and runtime environment, ensuring consistent and reproducible deployments across different environments.

The task begins by exporting an environment variable that points to an Application Load Balancer (ALB) provisioned in your AWS account. This ALB serves as the traffic routing mechanism for your Storyizer website container in the future.

Next, you utilize the docker build command to construct a Docker image for the Storyizer website application. This process involves providing necessary build arguments and adhering to the instructions defined in a Dockerfile.

After building the Docker image, you run a container based on that image on your local machine. This step allows you to verify the container’s proper functioning and the accessibility of the website.

Let’s start by building a docker container for the website.

1. **Command:** Run these commands to create a variable called **ALB\_DNS\_NAME** that points to an Application Load Balancer that has been provisioned into your account.
2. export ALB\_DNS\_NAME=$(aws elbv2 describe-load-balancers --names StoryizerAELB | jq -r '.LoadBalancers[0].DNSName')

echo "Your ALB DNS name is $ALB\_DNS\_NAME"

**Expected output:**

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

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

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

Your ALB DNS name is StoryizerAELB-1944931238.us-east-1.elb.amazonaws.com

1. **Command:** Run the following command to build a Docker image for your website application:

docker build -t storyizer/website --build-arg ELBDNS=$ALB\_DNS\_NAME .

**Caution:** The period (**.**) at the end of the **docker build** command is required.

**Note:** In the Dockerfile, we saw the ARG ELBDNS line. This defines a build argument named ELBDNS that can be passed to the Docker build process. Build arguments allow us to inject values into the Docker image at build time, which can be useful for configuring the application with dynamic values that may change between environments or deployments.

In this case, we want to configure the Storyizer website with the DNS name of the Application Load Balancer (ALB) that routes traffic to the website container. This ALB DNS name may change depending on the deployment environment or if the ALB is recreated. By using a build argument, we can easily pass in the correct ALB DNS name during the Docker build process, without having to modify the application code or configuration files directly.

**Expected output:**

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

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

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

[+] Building 69.7s (9/9) FINISHED docker:default

=> [internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 575B 0.0s

=> [internal] load metadata for docker.io/rockylinux/rockylinux:latest 1.1s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [1/4] FROM docker.io/rockylinux/rockylinux:latest@sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fa15a16f43e947940df5aca1a56e 6.4s

=> => resolve docker.io/rockylinux/rockylinux:latest@sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fa15a16f43e947940df5aca1a56e 0.0s

=> => sha256:523ffac7fb2e245e5e7c407b9f7377be9c6c3bf03d380981168311f49030da17 627B / 627B 0.0s

=> => sha256:71cc2ddb2ecf0e2a974aec10b55487120f03759e86e08b50a7f4c5d77638ab9b 75.70MB / 75.70MB 1.6s

=> => sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fa15a16f43e947940df5aca1a56e 437B / 437B 0.0s

=> => sha256:2f0bf3347b762fb21264670b046758782673694883cdf031af3aba982f656830 518B / 518B 0.0s

=> => extracting sha256:71cc2ddb2ecf0e2a974aec10b55487120f03759e86e08b50a7f4c5d77638ab9b 4.6s

=> [internal] load build context 0.0s

=> => transferring context: 109.08kB 0.0s

=> [2/4] RUN yum -y update && yum -y install httpd unzip && yum clean all 53.1s

=> [3/4] COPY ./code/ /var/www/html/ 0.1s

=> [4/4] RUN echo "ServerName storyizer.training " >> /etc/httpd/conf/httpd.conf && sed -i -- "s|APIELB|StoryizerAELB-1882370688.eu-west-1.elb.amaz 0.8s

=> exporting to image 8.0s

=> => exporting layers 7.9s

=> => writing image sha256:00a83501642d5ad0e2c0db9e1ef2baf1c498d59bd3dcc579296b52f896c41657 0.0s

=> => naming to docker.io/storyizer/website

Now that the docker image has been built, you can test it on the Command Host.

1. **Command:** Run this command to start the Docker container. The *-p 80:80* argument in the below command binds port 80 on the command host to port 80 on the docker container:

docker run -d -p 80:80 storyizer/website

**Expected output:**

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

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

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

108ed69d939f6171d22cc08789e93d171b98811d6cd15f22eb6d935d27bf35c8

1. **Command:** Run this command to check that the docker containers are currently running on this host:

docker ps

**Expected output:** The output includes a *CONTAINER ID* that you use to control the container. It is a random string that looks similar to: *c59f8347b1fa*

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

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

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

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c1977585469a storyizer/website "/usr/sbin/httpd -D …" 7 seconds ago Up 6 seconds 0.0.0.0:80->80/tcp, :::80->80/tcp quizzical\_booth

1. Copy the **CommandHost** value from the list to the left of these instructions, and then paste it into a new web browser tab.

**Expected output:** You should be greeted with the **Storyizer** page. The **submit** button does not currently work as it is not currently built or deployed.

1. Return to the web browser tab containing the terminal session for your Command Host instance.
2. **Command:** Run the following command to capture the container ID of a running Docker container and stop the container:
3. CONTAINER\_ID=$(docker ps | grep storyizer/website | awk '{print $1}')

docker stop $CONTAINER\_ID

**Expected output:**

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

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

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

108ed69d939f

**Task complete:** You have successfully built and tested the website container.

**Task 4: Building the API container**

In this task, you construct a Docker container image for the API component of the Storyizer application. While the website layer was addressed in the previous task, this API container serves as a crucial component of the overall application architecture.

You begin with an examination of the Dockerfile responsible for building the API container image. This Dockerfile outlines the instructions and configurations required to create a consistent and reproducible environment for the API component.

1. **Command:** Run these commands to navigate to the API directory and display the Dockerfile:
2. cd ../API

cat Dockerfile

**Expected Output:** Most of the dockerfile for the API container looks similar to the Website Dockerfile. In addition, example use of *WORKDIR* and *CMD* are shown:

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

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

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

FROM rockylinux/rockylinux

# Replace shell with bash so we can source files

RUN rm /bin/sh && ln -s /bin/bash /bin/sh

RUN yum update -y \

&& yum install -y wget unzip

ENV NVM\_DIR=/usr/local/nvm NODE\_VERSION=6.10.3

RUN curl https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash \

&& source $NVM\_DIR/nvm.sh \

&& nvm install $NODE\_VERSION \

&& nvm alias default $NODE\_VERSION \

&& nvm use default

ENV NODE\_PATH=$NVM\_DIR/v$NODE\_VERSION/lib/node\_modules PATH=$NVM\_DIR/versions/node/v$NODE\_VERSION/bin/:$PATH

# Install app

COPY ./code/ /opt/

WORKDIR /opt/API

EXPOSE 81

CMD ["node", "/opt/API/app.js"]

1. **Command:** Run this command to build the Docker image for the API container:

docker build -t storyizer/api .

**Caution:** The period (**.**) at the end of the **docker build** command is required.

**Expected Output:** Upon successful build, you should see something similar to:

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

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

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

[+] Building 249.5s (11/11) FINISHED docker:default

=> [internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 737B 0.0s

=> [internal] load metadata for docker.io/rockylinux/rockylinux:latest 0.3s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> CACHED [1/6] FROM docker.io/rockylinux/rockylinux:latest@sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fa15a16f43e947940df5aca1a56e 0.0s

=> [internal] load build context 0.3s

=> => transferring context: 1.27MB 0.2s

=> [2/6] RUN rm /bin/sh && ln -s /bin/bash /bin/sh 0.6s

=> [3/6] RUN yum update -y && yum install -y wget unzip 232.3s

=> [4/6] RUN curl https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash && source /usr/local/nvm/nvm.sh && nvm install 6 6.0s

=> [5/6] COPY ./code/ /opt/ 0.4s

=> [6/6] WORKDIR /opt/API 0.0s

=> exporting to image 9.8s

=> => exporting layers 9.7s

=> => writing image sha256:d3d34af660b69643880c5c06f53119e33c083c4e7728db75ca8c60a21245f6e3 0.0s

=> => naming to docker.io/storyizer/api

**Task complete:** You have successfully built the API container.

**Task 5: Building the Save container**

In this task, you construct a Docker container image for the final component of the Storyizer application - the Save microservice. This microservice is responsible for saving the stories you generate to a DynamoDB table.

1. **Command:** Run this command to navigate to the Save directory:

cd ../Save

**Expected output:**

*None, unless there is an error*

The Save container is just like the API container, however you need to pass an argument during the build to define the region.

1. **Command:** Run this command to build the Save container image. Note that the $AWS\_REGION variable has been pre-configured for you:

docker build -t storyizer/save --build-arg AWSREGION=$AWS\_REGION .

**Caution:** Be sure to include the period (**.**) at the end of the command.

**Expected Output:**

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

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

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

[+] Building 73.7s (12/12) FINISHED docker:default

=> [internal] load build definition from Dockerfile 0.0s

=> => transferring dockerfile: 875B 0.0s

=> [internal] load metadata for docker.io/rockylinux/rockylinux:latest 0.7s

=> [internal] load .dockerignore 0.0s

=> => transferring context: 2B 0.0s

=> [1/7] FROM docker.io/rockylinux/rockylinux:latest@sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fa15a16f43e947940df5aca1a56e 0.0s

=> [internal] load build context 2.1s

=> => transferring context: 17.50MB 2.0s

=> CACHED [2/7] RUN rm /bin/sh && ln -s /bin/bash /bin/sh 0.0s

=> [3/7] RUN yum update -y && yum install -y curl unzip && yum clean all 54.6s

=> [4/7] RUN curl https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash && source /usr/local/nvm/nvm.sh && nvm install 6 5.1s

=> [5/7] COPY ./code/ /opt/ 2.6s

=> [6/7] WORKDIR /opt/save 0.0s

=> [7/7] RUN sed -i -- "s|AWSRegion|eu-west-1|g" /opt/save/creds.json 0.5s

=> exporting to image 10.0s

=> => exporting layers 10.0s

=> => writing image sha256:cd1a53464b131e286c3d1206195e1b4a17cdfb9500a41aa815624eca8c0c6533 0.0s

=> => naming to docker.io/storyizer/save

**Task complete:** You have successfully built the Save container.

**Task 6: Tagging and pushing Docker images to Amazon ECR repository**

Say you want to share these docker images with other AWS accounts and users. In such a scenario, a repository for docker images is useful. In this task, you use Amazon ECR to host these images, one repository per application. In a subsequent step of this task, you launch an AWS CloudFormation template and need to provide the URI of the repositories such that ECS can pull the latest version of the docker image when building the cluster.

Until this point, you have created three docker images. But where have they been saved? What were they named?

**docker images** can be used to show the layers of the docker images that are present on the local system and the repo they are from.

1. **Command:** Run this command below to get information about all the docker images created.

docker images

**Expected Output:**

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

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

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

REPOSITORY TAG IMAGE ID CREATED SIZE

storyizer/save latest cd1a53464b13 16 seconds ago 525MB

storyizer/api latest d3d34af660b6 3 minutes ago 553MB

storyizer/website latest 00a83501642d 9 minutes ago 463MB

**Learn more:** The **IMAGE ID** is a unique identifier that can be used to call out a version of a docker image. Alternatively, you can reference the image by the repository it is associated with.

1. **Command:** Run these commands to create three ECR repositories to store the website, API, and save docker images. The URI of each of these repositories is saved to an environment variable:
2. export WEBSITE\_URI=$( \
3. aws ecr create-repository \
4. --repository-name storyizer-website \
5. --image-tag-mutability IMMUTABLE \
6. --query 'repository.repositoryUri' \
7. --output text
8. )
9. export SAVE\_URI=$( \
10. aws ecr create-repository \
11. --repository-name storyizer-save \
12. --image-tag-mutability IMMUTABLE \
13. --query 'repository.repositoryUri' \
14. --output text
15. )
16. export API\_URI=$( \
17. aws ecr create-repository \
18. --repository-name storyizer-api \
19. --image-tag-mutability IMMUTABLE \
20. --query 'repository.repositoryUri' \
21. --output text
22. )

echo "The website repository is located at $WEBSITE\_URI" && echo "The website repository is located at $SAVE\_URI" && echo "The website repository is located at $API\_URI"

**Expected Output:**

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

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

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

The website repository is located at 758476765923.dkr.ecr.us-east-1.amazonaws.com/storyizer-website

The website repository is located at 758476765923.dkr.ecr.us-east-1.amazonaws.com/storyizer-save

The website repository is located at 758476765923.dkr.ecr.us-east-1.amazonaws.com/storyizer-api

Before you can push the docker images to the repository, you need to authenticate into Amazon ECR. The command *aws ecr get-login* [(aws docs)](http://docs.aws.amazon.com/cli/latest/reference/ecr/get-login.html) returns a *docker login* token that is valid for 12 hours. The output of *aws ecr get-login* can be piped to a shell to run the *docker login* command to complete the authentication.

1. **Command:** Run this command to authenticate Docker with ECR:

aws ecr get-login-password --region $AWS\_REGION | docker login --username AWS --password-stdin $ACCOUNT\_ID.dkr.ecr.$AWS\_REGION.amazonaws.com

**Expected Output:**

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

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

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

WARNING! Your password will be stored unencrypted in /home/ssm-user/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

1. **Command:** Run these commands to tag the docker images with the repository URI they are stored in and indicate that they are the latest build with the **latest** keyword:
2. docker tag storyizer/website:latest $WEBSITE\_URI:latest
3. docker tag storyizer/save:latest $SAVE\_URI:latest

docker tag storyizer/api:latest $API\_URI:latest

**Expected output:**

*None, unless there is an error*

1. **Command:** Run this command to validate that the images are tagged:

docker images

**Note:** You should see that the images are associated with a repository URI.

**Expected Output:**

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

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

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

REPOSITORY TAG IMAGE ID CREATED SIZE

304328561748.dkr.ecr.eu-west-1.amazonaws.com/storyizer-save latest cd1a53464b13 5 minutes ago 525MB

storyizer/save latest cd1a53464b13 5 minutes ago 525MB

304328561748.dkr.ecr.eu-west-1.amazonaws.com/storyizer-api latest d3d34af660b6 8 minutes ago 553MB

storyizer/api latest d3d34af660b6 8 minutes ago 553MB

304328561748.dkr.ecr.eu-west-1.amazonaws.com/storyizer-website latest 00a83501642d 15 minutes ago 463MB

storyizer/website latest 00a83501642d 15 minutes ago 463MB

1. **Command:** Run these commands push Docker images to a remote container registry:
2. docker push $WEBSITE\_URI:latest
3. docker push $SAVE\_URI:latest

docker push $API\_URI:latest

**Note:** You should see that the images are copying the different layers which are part of the layered file system that docker has implemented.

**Expected Output:**

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

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

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

...

The push refers to repository [758476765923.dkr.ecr.us-east-1.amazonaws.com/storyizer-api]

febc6ed2300d: Pushed

154224e0c96d: Pushed

3343d63f4846: Pushed

cd01302219a7: Pushed

44e6e3eb06d8: Pushed

latest: digest: sha256:7eb504280b917aa368090d1c5592bc9ccd2f4d2e42a5e291ed260c4933fbdcc8 size: 1371

**Task complete:** You have successfully tagged and pushed docker images to ECR.

**Task 7: Deploying to Amazon ECS**

Now that the images are stored in the repository, you create a task for each of the components. To manage the scaling of the docker images, you also create a service for each app component.

1. **Command:** Task definitions for your applications three microservices have been saved as JSON files to the Command Host. Enter the following command to locate the JSON files and then view the contents of one of them:

tree ~/scripts && printf "\n" && cat ~/scripts/Site.json

**Expected output:**

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

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

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

/home/ssm-user/scripts

├── API.json

├── Save.json

└── Site.json

0 directories, 3 files

{

"containerDefinitions": [

{

"name": "Storyizer-Site",

"image": "$WEBSITE\_URI:latest",

"essential": true,

"portMappings": [

{

"containerPort": 80,

"hostPort": 80,

"protocol": "tcp"

}

],

"environment": [

{

"name": "Port",

"value": "80"

},

{

"name": "ServerName",

"value": "Storyizer-site"

},

{

"name": "APIELB",

"value": "$ALB\_DNS\_NAME"

},

{

"name": "SaveELB",

"value": "$ALB\_DNS\_NAME"

}

],

"memory": 300,

"cpu": 512

}

]

}

**Note:** From the output, you can see that three task definitions have been saved to the *~/scripts* directory.

The following list explains the key elements of the task definition:

* + *name*: Specifies the name of the task definition, which is “Storyizer-Site” in this case.
  + *image*: The Docker image to be used for the container, specified by the $WEBSITE\_URI:latest environment variable.
  + *essential*: Set to true, indicating that the container is essential for the task definition and should be launched when the task starts.
  + *portMappings*: Defines the port mapping for the container, where the container port 80 is mapped to the host port 80 using the TCP protocol.
  + *environment*: Specifies the environment variables to be set in the container, including the container port (80), server name (“Storyizer-site”), and the DNS name of the Application Load Balancer (ALB) for the API and saving operations, specified by the $ALB\_DNS\_NAME environment variable.
  + *memory*: The amount of memory (in MiB) to be reserved for the container (300 MiB in this case).
  + *cpu*: The number of CPU units to be reserved for the container (512 CPU units in this case).

**Note:** The Task Definition contains the following variables: *$WEBSITE\_URI* and *$ALB\_DNS\_NAME*. The two other JSON files contain similar variables, all of which need to be replaced with their actual values before they can be used to create new task definitions.

1. **Command:** To update the variables in the JSON files, enter the following command:
2. for json\_file in ~/scripts/\*.json; do
3. tempvalues=$(mktemp)
4. envsubst < "$json\_file" > "$tempvalues"
5. mv "$tempvalues" "$json\_file"

done

**Note:** The for loop iterates over all the JSON files in the *~/scripts/* directory. The following list explains elements of the command:

* + *tempvalues=$(mktemp)*: creates a temporary file with a unique name and stores its path in the tempvalues variable.
  + *envsubst < “$json\_file” > “$tempvalues”*: reads the contents of the JSON file, replaces any environment variables with their corresponding values, and writes the modified content to the temporary file.
  + *mv “$tempvalues” “$json\_file”*: moves (renames) the temporary file to the original JSON file, effectively overwriting it with the modified content.

**Expected output:**

*None, unless there is an error.*

1. **Command:** Now that the JSON files have been updated, enter these commands to create your Task Definitions and saving it as a variable:
2. SITE\_TASK\_DEF=$(aws ecs register-task-definition --family Storyizer-Site --cpu 512 --memory 300 --requires-compatibilities EC2 --network-mode bridge --execution-role-arn arn:aws:iam::$ACCOUNT\_ID:role/ecsTaskExecutionRole --cli-input-json file://~/scripts/Site.json --query 'taskDefinition.taskDefinitionArn' --output text)
3. API\_TASK\_DEF=$(aws ecs register-task-definition --family Storyizer-API --cpu 512 --memory 300 --requires-compatibilities EC2 --network-mode bridge --execution-role-arn arn:aws:iam::$ACCOUNT\_ID:role/ecsTaskExecutionRole --cli-input-json file://~/scripts/API.json --query 'taskDefinition.taskDefinitionArn' --output text)
4. SAVE\_TASK\_DEF=$(aws ecs register-task-definition --family Storyizer-Save --cpu 512 --memory 300 --requires-compatibilities EC2 --network-mode bridge --task-role-arn arn:aws:iam::$ACCOUNT\_ID:role/ecsTaskExecutionRole --cli-input-json file://~/scripts/Save.json --query 'taskDefinition.taskDefinitionArn' --output text)

echo "The Storyizer-Site task definition is $SITE\_TASK\_DEF" && echo "The Storyizer-API task definition is $API\_TASK\_DEF" && echo "The Storyizer-Save task definition is $SAVE\_TASK\_DEF"

**Expected Output:**

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

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

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

The Storyizer-Site task definition is arn:aws:ecs:us-east-1:758476765923:task-definition/Storyizer-Site:1

The Storyizer-API task definition is arn:aws:ecs:us-east-1:758476765923:task-definition/Storyizer-API:1

The Storyizer-Save task definition is arn:aws:ecs:us-east-1:758476765923:task-definition/Storyizer-Save:1

1. **Command:** Run these commands to configure the target group ARN as a variable:
2. SITE\_TARGET\_GROUP\_ARN=$(aws elbv2 describe-target-groups --names WebSiteTG80 --query 'TargetGroups[0].TargetGroupArn' --output text)
3. API\_TARGET\_GROUP\_ARN=$(aws elbv2 describe-target-groups --names ApiTG81 --query 'TargetGroups[0].TargetGroupArn' --output text)
4. SAVE\_TARGET\_GROUP\_ARN=$(aws elbv2 describe-target-groups --names SaveTG82 --query 'TargetGroups[0].TargetGroupArn' --output text)

echo "The WebSiteTG80 target group is $SITE\_TARGET\_GROUP\_ARN" && echo "The ApiTG81 target group is $API\_TARGET\_GROUP\_ARN" && echo "The SaveTG82 target group is $SAVE\_TARGET\_GROUP\_ARN"

**Expected Output:**

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

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

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

The WebSiteTG80 target group is arn:aws:elasticloadbalancing:us-east-1:758476765923:targetgroup/WebSiteTG80/6b00c6cec22a18c9

The ApiTG81 target group is arn:aws:elasticloadbalancing:us-east-1:758476765923:targetgroup/ApiTG81/f03cfb0fb8bd9bca

The SaveTG82 target group is arn:aws:elasticloadbalancing:us-east-1:758476765923:targetgroup/SaveTG82/429d10fd42fdc640

1. **Command:** Now that the variables are configured, enter these commands to create your ECS Services:
2. aws ecs create-service --service-name WebSiteService --cluster arn:aws:ecs:$AWS\_REGION:$ACCOUNT\_ID:cluster/Storyizer-Cluster --desired-count 2 --load-balancers targetGroupArn=$SITE\_TARGET\_GROUP\_ARN,containerName=Storyizer-Site,containerPort=80 --role arn:aws:iam::$ACCOUNT\_ID:role/ECSServiceRole --task-definition "$SITE\_TASK\_DEF" --launch-type EC2 --query 'service.serviceArn' --output text
3. aws ecs create-service --service-name ApiService --cluster arn:aws:ecs:$AWS\_REGION:$ACCOUNT\_ID:cluster/Storyizer-Cluster --desired-count 2 --load-balancers targetGroupArn=$API\_TARGET\_GROUP\_ARN,containerName=Storyizer-API,containerPort=81 --role arn:aws:iam::$ACCOUNT\_ID:role/ECSServiceRole --task-definition "$API\_TASK\_DEF" --launch-type EC2 --query 'service.serviceArn' --output text

aws ecs create-service --service-name SaveService --cluster arn:aws:ecs:$AWS\_REGION:$ACCOUNT\_ID:cluster/Storyizer-Cluster --desired-count 1 --load-balancers targetGroupArn=$SAVE\_TARGET\_GROUP\_ARN,containerName=Storyizer-Save,containerPort=82 --role arn:aws:iam::$ACCOUNT\_ID:role/ECSServiceRole --task-definition "$SAVE\_TASK\_DEF" --launch-type EC2 --query 'service.serviceArn' --output text

**Expected Output:**

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

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

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

arn:aws:ecs:us-east-1:758476765923:service/Storyizer-Cluster/WebSiteService

arn:aws:ecs:us-east-1:758476765923:service/Storyizer-Cluster/ApiService

arn:aws:ecs:us-east-1:758476765923:service/Storyizer-Cluster/SaveService

**Note:** Wait around 5 minutes for the services to deploy and then you can now try the application.

1. Copy the **ALBDNS** value shown to the left of these instructions.
2. Copy the **ALBDNS** value that is listed to the left of these instructions. Paste the URL into a new web browser tab and press **Enter** to navigate to an **Storyizer** application page on the web server.
3. Choose the **?** button at the top right corner to view help for the application.
4. On the **Help page**, under **How this works** section, copy the sample sentence at the bottom of the section.
5. Choose **return to app**, then paste the sentence.
6. choose **submit**.

**Note:** Your noun and verb placeholders should be replaced with random words.

**Task complete:** You have successfully created task definitions and services for each app component. You finally hosted a simple multi-component web application composed of a website with two supporting API services.

**Conclusion**

You have successfully done the following:

* Understood the steps needed to build docker images.
* Pushed container images to an Amazon ECR repository.
* Deployed containers from a repository to an Amazon ECS cluster as Services.

**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.

ADDITIONAL RESOURCES

* [Create a Docker image](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-container-image.html#create-container-image-create-image)
* [Push your image to Amazon Elastic Container Registry](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-container-image.html#create-container-image-push-ecr)
* [AWS CLI for register-task-definition](https://docs.aws.amazon.com/cli/latest/reference/ecs/register-task-definition.html)
* [AWS CLI for create-service](https://docs.aws.amazon.com/cli/latest/reference/ecs/create-service.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).