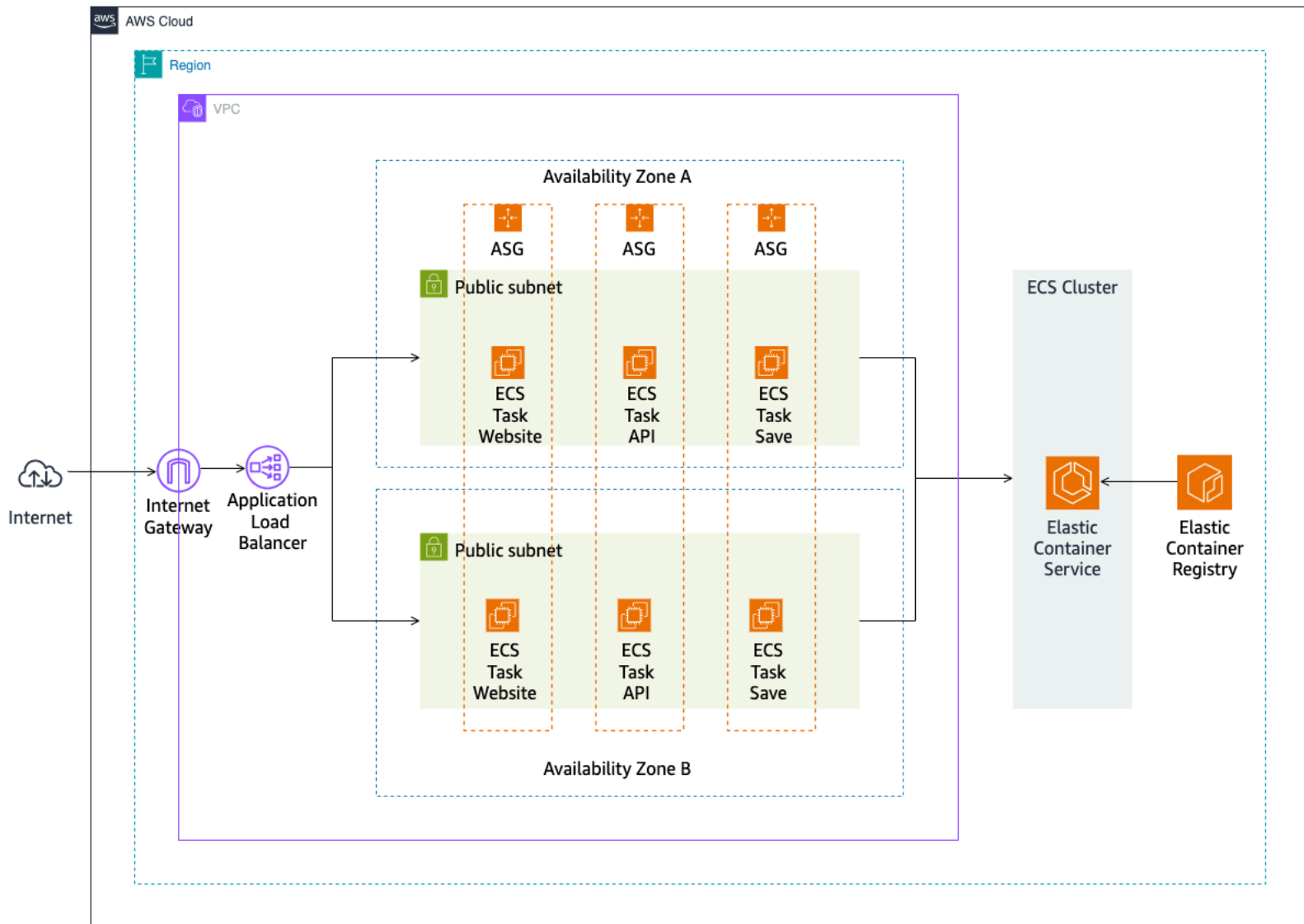


# Building and Deploying Containers Using Amazon Elastic Container Service

This report details the execution and commands used for building, pushing, and deploying a three-tier containerized application (Website, API, and Save services) using Amazon Elastic Container Service (ECS) and Elastic Container Registry (ECR).



## Phase 1: Containerization and Local Testing

# Building and Deploying Containers Using Amazon Elastic Container Service

I started by building the Docker images for all three services (Website, API, Save).

## 1. Website Service (storyizer/website)

The website container uses Apache HTTPD to serve the front-end application. I dynamically injected the **Application Load Balancer (ALB) DNS name** into the container during the build process using a build argument.

### Website Dockerfile (WebSite/Dockerfile)

```
cat Dockerfile
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"]
```

### Commands Executed for Website Build and Test:

Step	Command Executed	Purpose
Navigate	cd WebSite	Enter the website source directory.

# Building and Deploying Containers Using Amazon Elastic Container Service

<b>Get ALB DNS</b>	<code>`export ALB_DNS_NAME=\$(aws elbv2 describe-load-balancers --names StoryizerAELB</code>	<code>jq -r 'LoadBalancers[0].DNSName'</code>
<b>Build Image</b>	<code>docker build -t storyizer/website --build-arg ELBDNS=\$ALB_DNS_NAME .</code>	Build the image, passing the ALB DNS as a build argument (ELBDNS).
<b>Local Test Run</b>	<code>docker run -d -p 80:80 storyizer/website</code>	Run the container locally, mapping port 80 to test the configuration.
<b>Stop Container</b>	<code>`CONTAINER_ID=\$(dock er ps</code>	<code>grep storyizer/website</code>

```
sh-5.2$ export ALB_DNS_NAME=$(aws elbv2 describe-load-balancers --names StoryizerAELB | jq -r '.LoadBalancers[0].DNSName')
echo "Your ALB DNS name is $ALB_DNS_NAME"
Your ALB DNS name is StoryizerAELB-1527170160.us-west-2.elb.amazonaws.com
sh-5.2$ docker build -t storyizer/website --build-arg ELBDNS=$ALB_DNS_NAME .
[+] Building 82.1s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 575B
=> [internal] load metadata for docker.io/rockylinux/rockylinux:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/rockylinux/rockylinux:latest@sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fal5a16f43e947940dfSacala56e
=> => resolve docker.io/rockylinux/rockylinux:latest@sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fal5a16f43e947940dfSacala56e
=> => sha256:fc370d748f4cd1e6ac3d1b6460fb82201897fal5a16f43e947940dfSacala56e 437B / 437B
=> => sha256:2f0bf3347b762fb21264670b046758782673694883cd031af3aba982f656830 518B / 518B
=> => sha256:523ffac7fb2e245e5e7c407b9f7377be9e6c3bf03d380981168311f49030da17 627B / 627B
=> => sha256:71cc2ddb2ecf0e2a974aee10b55487120f03759e86e08b50a7f4c5d77638ab9b 75.70MB / 75.70MB
=> => extracting sha256:71cc2ddb2ecf0e2a974aee10b55487120f03759e86e08b50a7f4c5d77638ab9b
=> [internal] load build context
=> => transferring context: 109.08kB
=> [2/4] RUN yum -y update && yum -y install httpd unzip && yum clean all
=> [3/4] COPY ./code/ /var/www/html/
=> [4/4] RUN echo "ServerName storyizer.training" >> /etc/httpd/conf/httpd.conf && sed -i -- "s|APIELB|StoryizerAELB-1527170160.us-west-2.elb.amazo
=> => exporting layers
=> => writing image sha256:cb666b5055b46266cc1ccaff4d371e59ee0a3d9de59a8bcad2fd79aeb5f1506
=> => naming to docker.io/storyizer/website
sh-5.2$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
storyizer/website   latest          cb666b5055b4    15 seconds ago  471MB
sh-5.2$ ls
Dockerfile code
sh-5.2$ docker run -d -p 80:80 storyizer/website
af9bb2b235307b1ea12ed3c137ac2a0c9d6bf272d39980f97aa154ae9cd9923a
```

# Building and Deploying Containers Using Amazon Elastic Container Service

## 2. API Service (storyizer/api)

The API service uses Node.js (version 6.10.3) and is exposed on port 81.

### API Dockerfile (API/Dockerfile)

```
cat Dockerfile
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"]
sh-5.2$ docker build -t storyizer/api .
[+] Building 83.3s (11/11) FINISHED
docker:default
```

### Commands Executed for API Build:

Step	Command Executed	Purpose
Navigate	cd ../API	Change to the API source directory.
Build Image	docker build -t storyizer/api .	Build the API image.

```
sh-5.2$ docker images
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE
storyizer/api       latest         c48dbf02797f    18 seconds ago  625MB
storyizer/website   latest        cb666b5055b4    3 minutes ago   471MB
sh-5.2$ cd ../Save
sh-5.2$ docker build -t storyizer/save --build-arg AWSREGION=$AWS_REGION .
[+] Building 84.5s (12/12) FINISHED
```

# Building and Deploying Containers Using Amazon Elastic Container Service

## 3. Save Service (storyizer/save)

The Save service also uses Node.js and includes configuration to set the AWS region within the application's configuration file.

### Commands Executed for Save Build:

Step	Command Executed	Purpose
Navigate	<code>cd ../Save</code>	Change to the Save source directory.
Build Image	<code>docker build -t storyizer/save --build-arg AWSREGION=\$AWS_REGION .</code>	Build the Save image.

## Phase 2: ECR Management and Image Push

After successfully building the three local images, I created dedicated ECR repositories for them and pushed the images.

### Commands Executed for ECR Setup and Push:

# Building and Deploying Containers Using Amazon Elastic Container Service

```
sh-5.2$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
sh-5.2$ export WEBSITE_URI=( \
aws ecr create-repository \
  --repository-name storyizer-website \
  --image-tag-mutability IMMUTABLE \
  --query 'repository.repositoryUri' \
  --output text
)
export SAVE_URI=( \
aws ecr create-repository \
  --repository-name storyizer-save \
  --image-tag-mutability IMMUTABLE \
  --query 'repository.repositoryUri' \
  --output text
)
export API_URI=( \
aws ecr create-repository \
  --repository-name storyizer-api \
  --image-tag-mutability IMMUTABLE \
  --query 'repository.repositoryUri' \
  --output text
)
```

Step	Command Executed	Purpose
ECR Login	<code>`aws ecr get-login-password --region \$AWS_REGION</code>	<code>docker login --username AWS --password-stdin \$ACCOUNT_ID.dkr.ecr.\$A WS_REGION.amazonaws. com`</code>
Tag Images	<code>docker tag storyizer/website:latest \$WEBSITE_URI:latest docker tag storyizer/save:latest \$SAVE_URI:latest docker tag storyizer/api:latest \$API_URI:latest</code>	Tag local images to map them to the ECR repository URIs.
Push Images	<code>docker push \$WEBSITE_URI:latest docker push</code>	Push all three images to their respective ECR repositories.

# Building and Deploying Containers Using Amazon Elastic Container Service

	<code>\$SAVE_URI:latest</code> <code>docker push</code> <code>\$API_URI:latest</code>	
--	---	--

```
sh-5.2$ for json_file in ~/scripts/*.json; do
    tempvalues=$(mktemp)
    envsubst < "$json_file" > "$tempvalues"
    mv "$tempvalues" "$json_file"
done
sh-5.2$ 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)
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)
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 $SAVE_TASK_DEF"
The Storyizer-Site task definition is arn:aws:ecs:us-west-2:914483362358:task-definition/Storyizer-Site:1
The Storyizer-API task definition is arn:aws:ecs:us-west-2:914483362358:task-definition/Storyizer-API:1
The Storyizer-Save task definition is arn:aws:ecs:us-west-2:914483362358:task-definition/Storyizer-Save:1
sh-5.2$ SITE_TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --names WebSiteTG80 --query 'TargetGroups[0].TargetGroupArn' --output text)
```

## Phase 3: ECS Task Definition and Service Creation

This phase involved registering the ECS Task Definitions and creating the final ECS Services.

### 1. Task Definition Preparation

I used envsubst to dynamically replace environment variables (like \$WEBSITE\_URI and \$ALB\_DNS\_NAME) within the JSON template files before registering the Task Definitions.

**Commands Executed for Task Definition Registration:**

# Building and Deploying Containers Using Amazon Elastic Container Service

```
for json_file in ~/scripts/*.json; do
    tempvalues=$(mktemp)
    envsubst < "$json_file" > "$tempvalues"
    mv "$tempvalues" "$json_file"
done
```

```
sh-5.2$ 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)

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)

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"
The Storyizer-Site task definition is arn:aws:ecs:us-west-2:914483362358:task-definition/Storyizer-Site:1
The Storyizer-API task definition is arn:aws:ecs:us-west-2:914483362358:task-definition/Storyizer-API:1
The Storyizer-Save task definition is arn:aws:ecs:us-west-2:914483362358:task-definition/Storyizer-Save:1
```

## 2. Target Group Retrieval and Service Creation

I retrieved the ARNs of the existing Load Balancer Target Groups and used them to create the ECS Services, mapping the containers to the respective Load Balancer Listener ports.

### Commands Executed for Target Group Retrieval:

```
sh-5.2$ SITE_TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --names WebSiteTG80 --query 'TargetGroups[0].TargetGroupArn' --output text)

API_TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --names ApiTG81 --query 'TargetGroups[0].TargetGroupArn' --output text)

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"
The WebSiteTG80 target group is arn:aws:elasticloadbalancing:us-west-2:914483362358:targetgroup/WebSiteTG80/43d707e68312c369
The ApiTG81 target group is arn:aws:elasticloadbalancing:us-west-2:914483362358:targetgroup/ApiTG81/9760b247c4c8d13f
The SaveTG82 target group is arn:aws:elasticloadbalancing:us-west-2:914483362358:targetgroup/SaveTG82/70830bf2a788da48
```

### Commands Executed for Service Creation:



# Building and Deploying Containers Using Amazon Elastic Container Service

```
sh-5.2$ 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

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
arn:aws:ecs:us-west-2:914483362358:service/Storyizer-Cluster/WebSiteService
arn:aws:ecs:us-west-2:914483362358:service/Storyizer-Cluster/ApiService
arn:aws:ecs:us-west-2:914483362358:service/Storyizer-Cluster/SaveService
```



This project successfully demonstrated the end-to-end process of multi-container deployment on AWS, from packaging the application with Docker to managing redundant services with ECS, all running on the EC2 launch type.