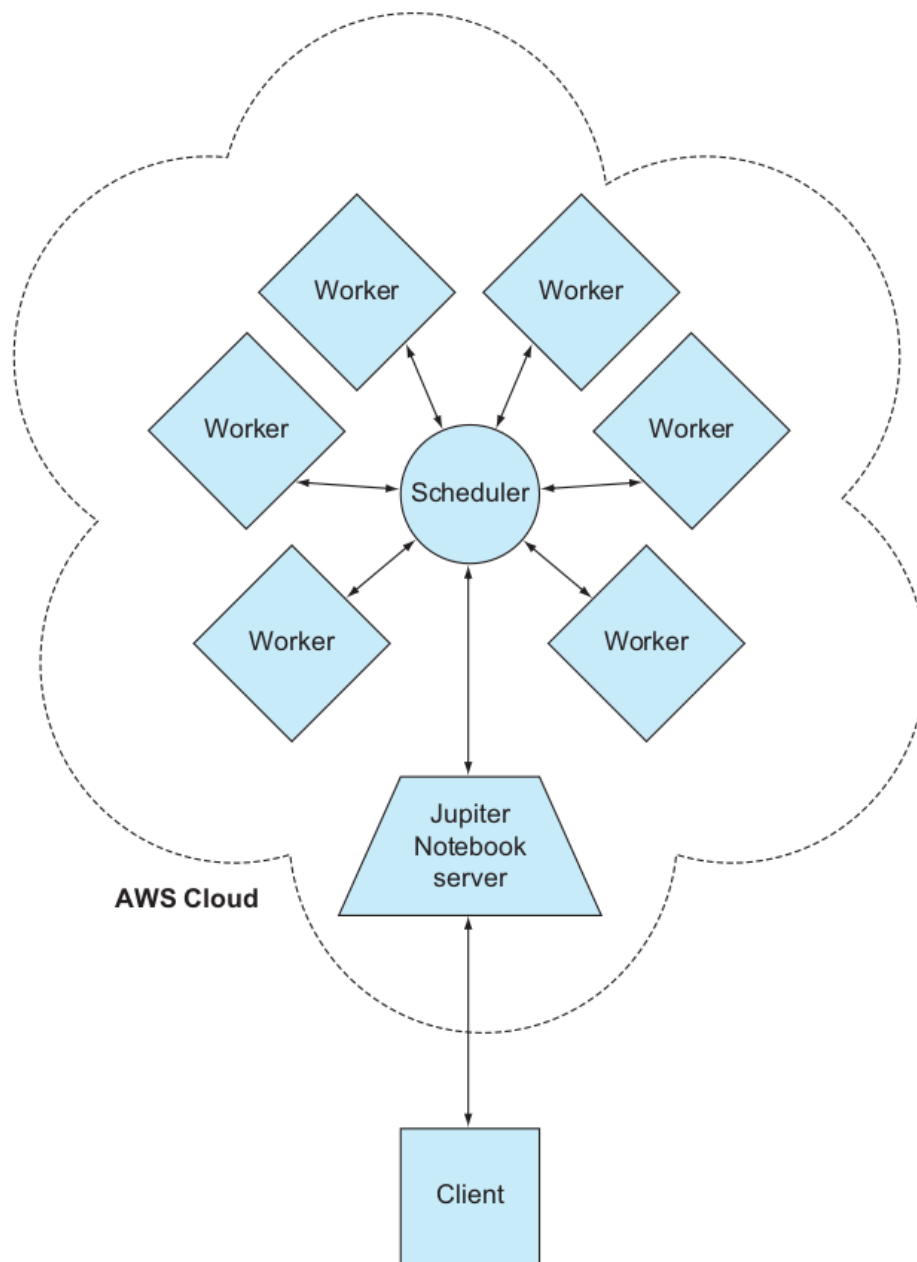


Architecture of Dask Cluster in AWS

The system has four distinct aspects: the jupyter server, the scheduler, the client and the workers. Their roles are as follows:

- Jupyter server - A frontend for the user to run code and submit jobs to the cluster
- Scheduler - Receives the jobs from the client via the Jupyter server, divides the work to be done, and coordinates the workers to complete the job
- Worker - Receives the tasks from the scheduler and computes them
- Client - Show the results to the user

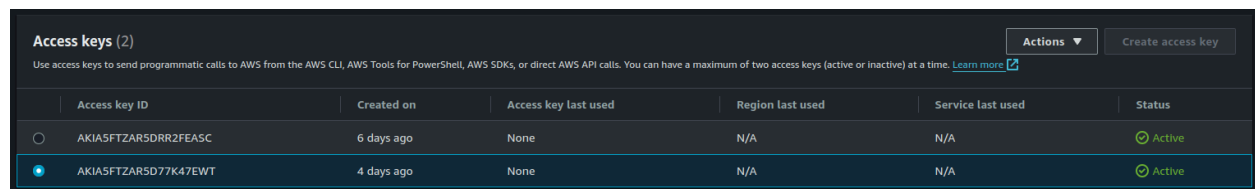


The setup involved a six-step process to get the cluster up and running:

1. Generate a security key.
2. Set up the ECS cluster.
3. Arrange the network settings for the cluster.
4. Establish a shared data drive using Elastic File System (EFS).
5. Reserve storage, build, and deploy the Docker images within Elastic Container Repository (ECR).
6. Connect to the cluster.

Generate a Security Key

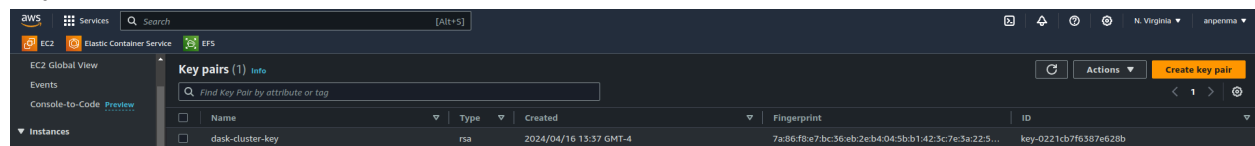
First a security key was created via the AWS security credentials page. This is needed for access via CLI in the local system to build and upload the docker images into the Elastic Container Repository (ECR).



	Access key ID	Created on	Access key last used	Region last used	Service last used	Status
<input type="radio"/>	AKIA5FTZAR5DRR2FEASC	6 days ago	None	N/A	N/A	Active
<input checked="" type="radio"/>	AKIA5FTZAR5D77K47EWT	4 days ago	None	N/A	N/A	Active

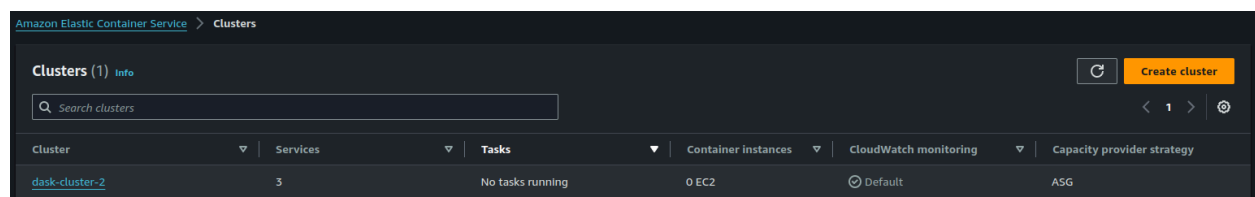
Setting up the ECS Cluster

The first step in this process was to obtain an SSH key that we will use to associate with the EC2 instances that were created by the ECS cluster. To do this under the EC2 home page, there is an option for Key Pairs. With this a Key Pair was created with the name of 'Dask Cluster Key'.



	Name	Type	Created	Fingerprint	ID
<input type="checkbox"/>	dask-cluster-key	rsa	2024/04/16 13:37 GMT-4	7a86f8e7bc36eb2eb4045bb1423c7e3a225...	key-0221cb7f6387e620b

With the Key Pair saved, next an ECS Cluster was to be created. This was done using the ECS Create Cluster wizard. The operating system chosen was Amazon Linux 2. Next for the EC2 instance type, we went with the t2.micro which was under the eligible types in the free tier. We went with the free tier first so that we didn't have to exhaust resources rapidly in just the setup. Following this a total of 8 instances were specified as the desired capacity. The Key-pair was specified as the one created earlier. Rest of the options were default. With this we had ECS start 8 instances under the name of dask-cluster.



Cluster	Services	Tasks	Container instances	CloudWatch monitoring	Capacity provider strategy
dask-cluster-2	3	No tasks running	0 EC2	Default	ASG

☒ **Amazon EC2 instances**

Manual configurations. Use for large workloads with consistent resource demands.

Auto Scaling group (ASG) | [Info](#)

Use Auto Scaling groups to scale the Amazon EC2 instances in the cluster.

Create new ASG ▼

Provisioning model

Select a provisioning model for your instances

☒ **On-demand**

With on-demand instances, you pay for compute capacity by the hour, with no long-term commitments or upfront payments.

☐ **Spot**

Amazon EC2 Spot instances let you take advantage of unused EC2 capacity in the AWS cloud. Spot instances are available at up to a 90% discount compared to on-demand prices.

Operating system/Architecture

Choose the Windows operating system or Linux architecture for your instance.

Amazon Linux 2 (kernel 5.10) ▼

EC2 Instance type

Choose based on the workloads you plan to run on this cluster.

t2.micro

i386, x86_64

1 vCPU 1 GiB Memory

Free tier eligible ▼

EC2 instance role

An instance role is used by Amazon EC2 instances to make AWS API requests. If you don't already have an instance IAM role created, we can create one for you.

ecsInstanceRole ▼

arn:aws:iam::905418149703:instance-profile/ecsInstanceRole

Desired capacity

Specify the number of instances to launch in your cluster.

Minimum

8

Maximum

8

SSH Key pair

If you do not specify a key pair, you can't connect to the instances via SSH unless you choose an AMI that is configured to allow users another way to log in.

dask-cluster-key ▼



Create a new key pair [↗](#)

Root EBS volume size


You can increase the size of the root EBS volume to allow for greater image and container storage.

30

▼ Network settings for Amazon EC2 instances [Info](#)

By default Amazon EC2 instances are launched in the default subnets for your default VPC. To use the non-default VPC, specify the VPC and subnets.

VPC

Use a VPC with public and private subnets. By default, VPCs are created for your AWS account. To create a new VPC, go to the [VPC Console](#) .

vpc-04f781ed4119c3058
default

Subnets

Select the subnets where your tasks run. We recommend that you use three subnets for production.

Choose subnets

Clear current selection

subnet-0c3e3faa5a2958cd0 ✕
us-east-1d 172.31.80.0/20

subnet-0e5c17564ac88c530 ✕
us-east-1c 172.31.0.0/20

subnet-05062af7bbc574245 ✕
us-east-1f 172.31.64.0/20

subnet-06f72a18b12cd6b63 ✕
us-east-1a 172.31.16.0/20

subnet-048e3190e3f6389ae ✕
us-east-1b 172.31.32.0/20

subnet-05abbaea6f8a3432a ✕
us-east-1e 172.31.48.0/20

Security group [Info](#)

Choose an existing security group or create a new security group.

- ☒ Use an existing security group
- ☐ Create a new security group

Security group name

Choose an existing security group.

Choose security groups

sg-01020bbaaa1e69d28 ✕
default

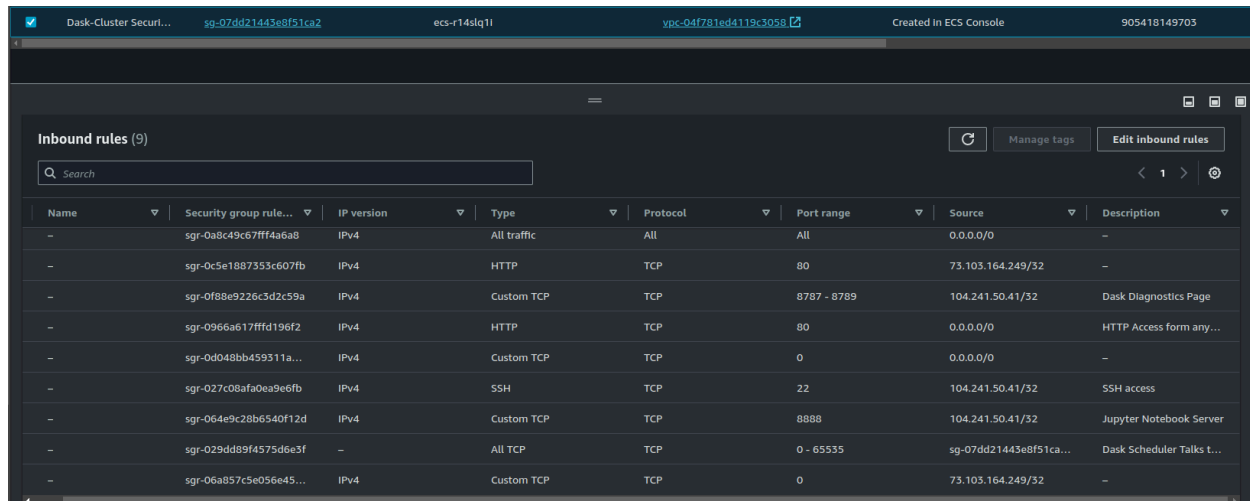
Auto-assign public IP [Info](#)

Choose whether to auto-assign a public IP to the Amazon EC2 instances

Use subnet setting

Arrange the Network Settings for the Cluster

With the cluster running and the cluster's firewall rules needed to be configured so we can ssh to it. When ECS creates a cluster, it will automatically create a security group for it in EC2. So here a few inbound rules were added. For the ssh access, TCP protocol was assigned with port 22 and ip address as the user's current address. Second rule was type All TCP with the security group as the ip address for allowing Dask Scheduler to communicate with the workers. Then for the Jupyter server a custom TCP with port 8888 was created and ip address as the user's. Finally, for viewing the Dask diagnostics page, a custom TCP rule with port range of 8787 to 8789 and ip address as user's was selected. Most things with instances in a cluster are temporary like its ip address and compute resources. But if we want persistent storage, we have to leverage the Elastic File System in AWS.

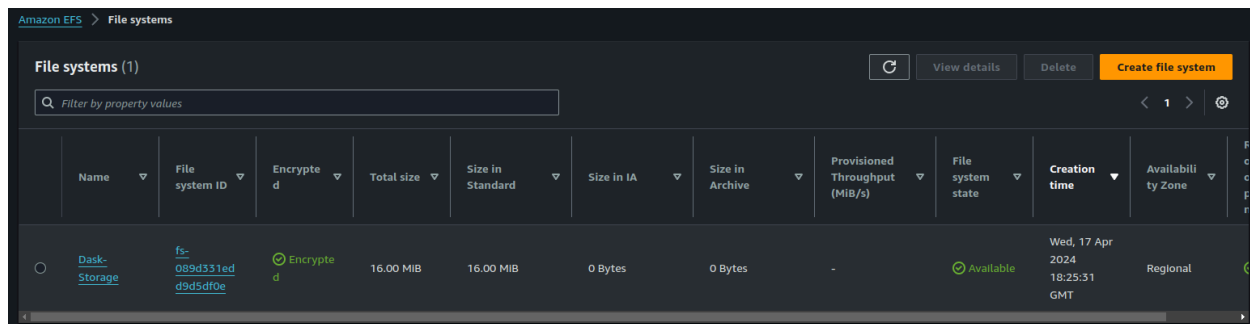


The screenshot shows the AWS IAM console 'Inbound rules' page for a security group. The table lists 9 inbound rules with columns: Name, Security group rule..., IP version, Type, Protocol, Port range, Source, and Description.

Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Description
-	sg-0a8c49c67ff4a6a8	IPv4	All traffic	All	All	0.0.0.0/0	-
-	sg-0c5e1887353c607fb	IPv4	HTTP	TCP	80	73.103.164.249/32	-
-	sg-0f88e9226c3d2c59a	IPv4	Custom TCP	TCP	8787 - 8789	104.241.50.41/32	Dask Diagnostics Page
-	sg-0966a617fffd196f2	IPv4	HTTP	TCP	80	0.0.0.0/0	HTTP Access form any...
-	sg-0d048bb459311a...	IPv4	Custom TCP	TCP	0	0.0.0.0/0	-
-	sg-027c08afa0ea9e6fb	IPv4	SSH	TCP	22	104.241.50.41/32	SSH access
-	sg-064e9c28b6540f12d	IPv4	Custom TCP	TCP	8888	104.241.50.41/32	Jupyter Notebook Server
-	sg-029dd89f4575d6e3f	-	All TCP	TCP	0 - 65535	sg-07dd21443e8f51ca...	Dask Scheduler Talks t...
-	sg-06a857c5e056e45...	IPv4	Custom TCP	TCP	0	73.103.164.249/32	-

Establishing a Shared Data Drive in using Elastic File System (EFS)

In order to create an EFS drive, the instances' VPC ID is needed. This was obtained from the EC2 Instance Manager. Post this, the EFS creation wizard was launched. In this, the VPC ID was provided. Additionally, security groups were removed and rest was left at default. With this the shared data drive was initiated. The DNS name for the filesystem was then saved for creating the launch template.



The screenshot shows the Amazon EFS console 'File systems' page. The table lists 1 file system with columns: Name, File system ID, Encrypt, Total size, Size in Standard, Size in IA, Size in Archive, Provisioned Throughput (MiB/s), File system state, Creation time, and Availability Zone.

Name	File system ID	Encrypt	Total size	Size in Standard	Size in IA	Size in Archive	Provisioned Throughput (MiB/s)	File system state	Creation time	Availability Zone
Dask-Storage	fs-089d331ed-d9d5df0e	Encrypt	16.00 MiB	16.00 MiB	0 Bytes	0 Bytes	-	Available	Wed, 17 Apr 2024 18:25:31 GMT	Regional

Next, we had to create a Launch template that is to be provided to the auto-scaling group for automatically launching the instances. For this we copied an existing ECS template and modified the security group and provided the below code to be run on startup for each instance which includes the DNS for the file system and cluster they need to be associated to:

```
Content-Type: multipart/mixed; boundary==="BOUNDARY==="
MIME-Version: 1.0

--==BOUNDARY==
Content-Type: text/cloud-boothook; charset="us-ascii"

# Install nfs-utils
cloud-init-per once yum_update yum update -y
cloud-init-per once install_nfs_utils yum install -y nfs-utils

# Create /efs folder
cloud-init-per once mkdir_efs mkdir /efs

# Mount /efs
cloud-init-per once mount_efs echo -e
'fs-089d331edd9d5df0e.efs.us-east-1.amazonaws.com:/ /efs nfs4
nfsvers=4.1,rsize=1048576,wsiz=1048576,hard,timeo=600,retrans=2
0 0' >>
/etc/fstab
mount -a

--==BOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
echo ECS_CLUSTER=dask-cluster-2 >> /etc/ecs/ecs.config;
echo ECS_BACKEND_HOST= >> /etc/ecs/ecs.config
--==BOUNDARY===
```

lt-036329fcf145d1726Dask-Launch-Template192024-04-17T18:45:23.000Zarn:aws:iam::905418149703:root

Dask-Launch-Template (lt-036329fcf145d1726)

Launch template IDlt-036329fcf145d1726

Launch template nameDask-Launch-Template

Default version1

Ownerarn:aws:iam::905418149703:root

Details

Versions

Template tags

Launch template version details

ActionsDelete template version

Version9

Description-

Date created2024-04-17T20:37:45.000Z

Created byarn:aws:iam::905418149703:root

Instance details

Storage

Resource tags

Network interfaces

Advanced details

AMI IDami-0bd2f238e75f8092a

Instance typet2.micro

Availability Zone-

Key pair namedask-cluster-key

Security groups-

Security group IDssg-07dd21443e8f51ca2

With the launch template created, the next step involves supplying it to the auto-scaling group. For the auto-scaling group creation, a few inputs were given such as VPC, launch template, subnets, and group size. With this we could finally start the auto-scaling feature working after terminating all the old instances.

The screenshot displays the AWS Management Console interface for an Auto Scaling group. At the top, there's a header for 'Auto Scaling groups (1/1)' with navigation links for 'Launch configurations', 'Launch templates', 'Actions', and a 'Create Auto Scaling group' button. Below this is a search bar and a table listing the Auto Scaling groups. The table has columns for Name, Launch template/configuration, Instances, Status, Desired capacity, Min, Max, and Availability Zones. The group 'Dask-Cluster-Auto-Scaling' is selected, showing it uses the 'Dask-Launch-Template' and has 0 instances, a desired capacity of 0, and a maximum capacity of 8 across multiple availability zones.

Auto Scaling group: Dask-Cluster-Auto-Scaling

Auto Scaling group name Dask-Cluster-Auto-Scaling	Desired capacity 0	Desired capacity type Units (number of instances)	Amazon Resource Name (ARN) arn:aws:autoscaling:us-east-1:905418149703:autoScalingGroup:24d2a664-019c-42b8-93b8-5defa4826488:autoScalingGroupName/Dask-Cluster-Auto-Scaling
Date created Wed Apr 17 2024 16:14:06 GMT-0400 (Eastern Daylight Time)	Minimum capacity 0	Status -	
	Maximum capacity 8		

Launch template

Launch template lt-036329fcf145d1726 Dask-Launch-Template	AMI ID ami-0bd2f238e75f8092a	Instance type t2.micro	Owner am:aws:iam::905418149703:root
Version Latest	Security groups -	Security group IDs sg-07dd21443e8f51ca2	Create time Wed Apr 17 2024 16:37:45 GMT-0400 (Eastern Daylight Time)
Description -	Storage (volumes) /dev/xvda	Key pair name dask-cluster-key	Request Spot Instances No

Once all the instances were running, a test had to be performed to check whether the EFS drive had been successfully mounted onto all the instances. For this using a file was sent to an instance using the scp from a local system. Once it was sent, the expectation was that the file would be in all of the remaining instances. After checking all the instances we were able to confirm that indeed the file was present everywhere.

The image displays three terminal windows, each showing an SSH connection from a local machine to an Amazon Linux 2 instance. The windows are titled 'ec2-user@ip-172-31-4-254/efs', 'ec2-user@ip-172-31-31-136/efs', and 'ec2-user@ip-172-31-28-84/efs'. Each window shows the standard SSH prompt, the user's location, and the command to connect to an EC2 instance. The output shows the authenticity of the host, the key fingerprint, and a warning that the key is not known by any other names. The user is prompted to continue connecting, and they respond with 'yes'. The warning is then permanently added to the list of known hosts. Below the warning, the Amazon Linux 2 logo is displayed, followed by the text 'Amazon Linux 2 (ECS Optimized)'. Finally, the user is prompted to visit the AWS documentation page, and they respond with 'yes'. The user then runs the command 'cd /efs && ls', which lists the files 'details.txt', 'sentiment_feature_array.zarr', and 'sentiment_target_array.zarr'.

```
antrudh@box:~/University/Spring 24/ECC$ ssh -i dask-cluster-key.pem ec2-user@ec2-3-80-163-219.compute-1.amazonaws.com
The authenticity of host 'ec2-3-80-163-219.compute-1.amazonaws.com (3.80.163.219)' can't be established.
ED25519 key fingerprint is SHA256:qVOHVnlgFR0FsITaNXdS7WucAlvr5/gnQMqwKEBP+0.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-3-80-163-219.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

Amazon Linux 2 (ECS Optimized)

For documentation, visit http://aws.amazon.com/documentation/ecs
[ec2-user@ip-172-31-4-254 ~]$ cd /efs && ls
details.txt  sentiment_feature_array.zarr  sentiment_target_array.zarr
[ec2-user@ip-172-31-4-254 efs]$
```

```
antrudh@box:~/University/Spring 24/ECC$ ssh -i dask-cluster-key.pem ec2-user@ec2-34-234-88-201.compute-1.amazonaws.com
The authenticity of host 'ec2-34-234-88-201.compute-1.amazonaws.com (34.234.88.201)' can't be established.
ED25519 key fingerprint is SHA256:YB26NonAIcYjC5LLmMnM8/4ykmvUJUMUdHsRwltMc.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-34-234-88-201.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

Amazon Linux 2 (ECS Optimized)

For documentation, visit http://aws.amazon.com/documentation/ecs
[ec2-user@ip-172-31-31-136 ~]$ cd /efs && ls
details.txt  sentiment_feature_array.zarr  sentiment_target_array.zarr
[ec2-user@ip-172-31-31-136 efs]$
```

```
antrudh@box:~/University/Spring 24/ECC$ ssh -i dask-cluster-key.pem ec2-user@ec2-34-228-63-96.compute-1.amazonaws.com
The authenticity of host 'ec2-34-228-63-96.compute-1.amazonaws.com (34.228.63.96)' can't be established.
ED25519 key fingerprint is SHA256:P08MurHWnewSalvj2ywc7FihvCb8xmvMdsq7M15yg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-34-228-63-96.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

Amazon Linux 2 (ECS Optimized)

For documentation, visit http://aws.amazon.com/documentation/ecs
[ec2-user@ip-172-31-28-84 ~]$ cd /efs && ls
details.txt  sentiment_feature_array.zarr  sentiment_target_array.zarr
[ec2-user@ip-172-31-28-84 efs]$
```

Reserve storage, build, and deploy the Docker images within Elastic Container Repository (ECR)

The purpose of this step is to build our Docker images and upload them into the ECR. Thankfully, AWS has streamlined and simplified the process a lot. But in order for the images to be uploaded via CLI in the local system, first AWS CLI had to be configured in the local machine. This involved creating an IAM user with permission to upload and modify files. With the user created and logged into the AWS CLI, the docker commands to be used were already provided in the ECR under the option push commands and were run in the terminal for each repository separately. There were 3 repositories created for this project: dask-notebook, dask-scheduler, dask-worker. But it will always initialize dask-scheduler first and only then notebook and workers will start as scheduler's ip address is required for the nodes to establish communication.

Amazon ECR > Public Registry > Repositories

Public repositories

Repositories (3) View push commands Delete Actions Create repository

Filter status

	Repository name	URI	Created at
<input type="radio"/>	dask-notebook	public.ecr.aws/w4p4q4w8/dask-notebook	April 18, 2024, 08:39:15 (UTC-04)
<input type="radio"/>	dask-scheduler	public.ecr.aws/w4p4q4w8/dask-scheduler	April 17, 2024, 17:30:07 (UTC-04)
<input type="radio"/>	dask-worker	public.ecr.aws/w4p4q4w8/dask-worker	April 17, 2024, 17:30:27 (UTC-04)

With the docker images in-place, a task definition had to be supplied for the images to be initialized as containers in the instances. Under the task definition creation wizard, the cpu and memory were allocated according to t2.micro's capacity, for the container the image URI for the respective repositories were provided. Additionally, the container ports were mapped to specific host ports so as to avoid any conflicts. Logs were enabled so as to obtain the Jupyter server's token to login and to also identify problems when something goes wrong. Additionally, the EFS drive location was mentioned as the mounting point for a shared data space.

Amazon Elastic Container Service > Task definitions

Task definitions (3) Info

Filter by status Active Deploy Create new revision Create new task definition

Filter task definitions

	Task definition	Status of last revision
<input type="radio"/>	dask-notebook	ACTIVE
<input type="radio"/>	dask-scheduler	ACTIVE
<input type="radio"/>	dask-worker	ACTIVE

☒ Amazon EC2 Instances

Self-managed infrastructure using Amazon EC2 instances.

OS, Architecture, Network mode

Network mode is used for tasks and is dependent on the compute type selected.

Operating system/Architecture [Info](#)

Linux/X86_64

Network mode [Info](#)

host

Task size [Info](#)

Specify the amount of CPU and memory to reserve for your task.

CPU

.5 vCPU

Memory

.7 GB

▼ Task roles - *conditional*

Task role [Info](#)

A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the [IAM console](#).

ecsTaskExecutionRole

Task execution role [Info](#)

A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM role, you must create one.

ecsTaskExecutionRole

▼ Container - 1 [Info](#)

Essential container

Remove

Container details

Specify a name, container image, and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name

dask-scheduler

Image URI

public.ecr.aws/w4p4q4w8/dask-scheduler:latest

Essential container

Yes

Private registry [Info](#)

Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.

☐ Private registry authentication

Port mappings [Info](#)

Add port mappings to allow the container to access ports on the host to send or receive traffic. For port name, a default will be assigned if left blank.

Container port

8786

Protocol

TCP

Port name

8786

App protocol

HTTP

Remove

8787

TCP

8787

App protocol

Remove

Add port mapping

Read only root file system [Info](#)

When this parameter is turned on, the container is given read-only access to its root file system.

☐ Read only

Resource allocation limits - *conditional* [Info](#)

Container-level CPU, GPU, and memory limits are different from task-level values. They define how much resources are allocated for the container. If container attempts to exceed the memory specified in hard limit, the container is terminated.

CPU

0

GPU

1

Memory hard limit

3

Memory soft limit

0.7

Volumes

Info

Add one or more data volumes for your task to provide additional storage for the containers in the task. For each data volume, you must add a mount point to specify where to mount the data volume in the container.

▼ Volume - 1

Remove

Volume name

Info

efs-data

Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

Configuration type

Info

Choose to configure a volume in the task definition or later at deployment.

☒ Configure at task definition creation

You can configure bind mount, Docker, Amazon EFS, and Amazon FSx for Windows File Server volumes when creating a task definition.

☐ Configure at deployment

You can configure 1 Amazon EBS volume when creating or updating a service, or when running a standalone task.

Volume type

Info

Bind mount

Storage configurations

Source path

Info

/efs

Add volume

Container mount points

Info

For each data volume associated with the task, add a container mount point to determine where the data volume is mounted.

Container

Source volume

Container path

Read only

dask-scheduler

efs-data

/data

☐ Read only

Remove

Connecting to the Cluster

With all these in place, all we had to do to connect to the cluster was to use the public DNS name of the notebook and scheduler to view them on the local browser. The notebook was present on port 8888 and dask's diagnostics page was located on port 8787. With all of these steps successfully done, we could now connect to the cluster and begin running some jobs. To test this, some sample Dask code was used that was already provided as part of the Jupyter server.

```
[3]: # Listing 11.2
from dask.distributed import Client, progress
client = Client('ip-172-31-91-162.ec2.internal:8786')
client

/opt/conda/lib/python3.10/site-packages/distributed/client.py:1393: VersionMismatchWarning: Mismatched versions found

+-----+-----+-----+-----+
| Package | Client | Scheduler | Workers |
+-----+-----+-----+-----+
| python | 3.10.14.final.0 | 3.10.12.final.0 | 3.10.12.final.0 |
| toolz | 0.12.1 | 0.12.0 | 0.12.0 |
+-----+-----+-----+-----+
warnings.warn(version_module.VersionMismatchWarning(msg[0]["warning"]))
```

Client

Client-1e85316cfd9b-11ee-82ee-0242ac110002

Connection method: Direct

Dashboard: <http://ip-172-31-91-162.ec2.internal:8787/status>

Launch dashboard in JupyterLab

▼ Scheduler Info

Scheduler

Scheduler-53dcc6a7-8264-4210-ae0e-756f4073875f

Comm: tcp://172.31.91.162:8786

Workers: 6

Dashboard: <http://172.31.91.162:8787/status>

Total threads: 6

Started: 28 minutes ago

Total memory: 4.20 GiB

► Workers

Here we are able to see that Dask is able to successfully connect with the scheduler using its IP address that was provided manually. Ideally just defining an environment variable that stores the scheduler's ip address should tell Dask where the scheduler is. But for some mysterious reason, it just wasn't working. So the scheduler ip address had to be provided in the code.

Checking to see if the dask jobs were being successfully submitted and distributed among the 6 worker nodes.

