# Contents

## 1 Installation

1.1 Pip ......................................................... 3
1.2 Conda ....................................................... 3

## 2 AWS

2.1 Fargate/ECS .................................................. 5
2.1.1 GPU Support ............................................. 6

Index ......................................................... 11
Native Cloud integration for Dask.

This library creates Dask clusters on a given cloud provider with no set up other than having credentials. Currently, it only supports AWS.
CHAPTER 1

Installation

1.1 Pip

```bash
$ pip install dask-cloudprovider
```

1.2 Conda

```bash
$ conda install -c conda-forge dask-cloudprovider
```

Below are the different modules for creating clusters on various cloud providers.
In order to create clusters on AWS you need to set your access key, secret key and region. The simplest way is to use the aws command line tool.

```sh
$ pip install awscli
$ aws configure
```

### 2.1 Fargate/ECS

The `FargateCluster` will create a new Fargate ECS cluster by default along with all the IAM roles, security groups, and so on that it needs to function.

```python
from dask_cloudprovider import FargateCluster
cluster = FargateCluster()
```

All AWS resources created by `FargateCluster` should be removed on garbage collection. If the process is killed harshly this will not happen.

Note that in many cases you will want to specify a custom Docker image to `FargateCluster` so that Dask has the packages it needs to execute your workflow.

```python
from dask_cloudprovider import FargateCluster
cluster = FargateCluster(image="<hub-user>/<repo-name>:<tag>")
```

One strategy to ensure that package versions match between your custom environment and the Docker container is to create your environment from an `environment.yml` file, export the exact package list for that environment using `conda list --export > package-list.txt`, and then use the pinned package versions contained in `package-list.txt` in your Dockerfile. You could use the default Dask Dockerfile as a template and simply add your pinned additional packages.

You can also create Dask clusters using EC2 based ECS clusters using `ECSCluster`.

Creating the ECS cluster is out of scope for this library but you can pass in the ARN of an existing one like this:
from dask_cloudprovider import ECSCluster
cluster = ECSCluster(cluster_arn="arn:aws:ecs::<region>:<acctid>:cluster/<clustername>
˓→")

All the other required resources such as roles, task definitions, tasks, etc will be created automatically like in FargateCluster.

### 2.1.1 GPU Support

There is also support in ECSCluster for GPU aware Dask clusters. To do this you need to create an ECS cluster with GPU capable instances (from the g3, p3 or p3dn families) and specify the number of GPUs each worker task should have.

```python
from dask_cloudprovider import ECSCluster
cluster = ECSCluster(
    cluster_arn="arn:aws:ecs::<region>:<acctid>:cluster/<gpuclustername>",
    worker_gpu=1)
```

By setting the worker_gpu option to something other than None will cause the cluster to run `dask-cuda-worker` as the worker startup command. Setting this option will also change the default Docker image to `rapidsai/rapidsai:latest`, if you’re using a custom image you must ensure the NVIDIA CUDA toolkit is installed with a version that matches the host machine along with dask-cuda.

### API

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ECSCluster([fargate_scheduler,...])</code></td>
<td>Deploy a Dask cluster using ECS</td>
</tr>
<tr>
<td><code>FargateCluster(**kwargs)</code></td>
<td>Deploy a Dask cluster using Fargate on ECS</td>
</tr>
</tbody>
</table>

```python
class dask_cloudprovider.ECSCluster(fargate_scheduler=False, fargate_workers=False, image=None, scheduler_cpu=None, scheduler_mem=None, scheduler_timeout=None, worker_cpu=None, worker_mem=None, worker_gpu=None, n_workers=None, cluster_arn=None, cluster_name_template=None, execution_role_arn=None, task_role_arn=None, task_role_policies=None, cloudwatch_logs_group=None, cloudwatch_logs_stream_prefix=None, cloudwatch_logs_default_retention=None, vpc=None, subnets=None, security_groups=None, environment=None, tags=None, find_address_timeout=None, skip_cleanup=None, aws_access_key_id=None, aws_secret_access_key=None, region_name=None, **kwargs)
```

Deploy a Dask cluster using ECS

This creates a dask scheduler and workers on an ECS cluster. If you do not configure a cluster one will be created for you with sensible defaults.

**Parameters**

- **fargate_scheduler**: `bool (optional)` Select whether or not to use fargate for the scheduler.
  
  Defaults to `False`. You must provide an existing cluster.
fargate_workers: bool (optional) Select whether or not to use fargate for the workers.

    Defaults to False. You must provide an existing cluster.

image: str (optional) The docker image to use for the scheduler and worker tasks.

    Defaults to daskdev/dask:latest or rapidsai/rapidsai:latest if worker_gpu is set.

scheduler_cpu: int (optional) The amount of CPU to request for the scheduler in milli-cpu (1/1024).

    Defaults to 1024 (one vCPU).

scheduler_mem: int (optional) The amount of memory to request for the scheduler in MB.

    Defaults to 4096 (4GB).

scheduler_timeout: str (optional) The scheduler task will exit after this amount of time if there are no clients connected.

    Defaults to 5 minutes.

worker_cpu: int (optional) The amount of CPU to request for worker tasks in milli-cpu (1/1024).

    Defaults to 4096 (four vCPUs).

worker_mem: int (optional) The amount of memory to request for worker tasks in MB.

    Defaults to 16384 (16GB).

worker_gpu: int (optional) The number of GPUs to expose to the worker.

    To provide GPUs to workers you need to use a GPU ready docker image that has dask-cuda installed and GPU nodes available in your ECS cluster. Fargate is not supported at this time.

    Defaults to None, no GPUs.

n_workers: int (optional) Number of workers to start on cluster creation.

    Defaults to None.

cluster_arn: str (optional if fargate is true) The ARN of an existing ECS cluster to use for launching tasks.

    Defaults to None which results in a new cluster being created for you.

cluster_name_template: str (optional) A template to use for the cluster name if cluster_arn is set to None.

    Defaults to 'dask-{uuid}’

execution_role_arn: str (optional) The ARN of an existing IAM role to use for ECS execution.

    This ARN must have sts:AssumeRole allowed for ecs-tasks.amazonaws.com and allow the following permissions:

    • ecr:GetAuthorizationToken
    • ecr:BatchCheckLayerAvailability
    • ecr:GetDownloadUrlForLayer
    • ecr:GetRepositoryPolicy
• `ecr:DescribeRepositories`
• `ecr:ListImages`
• `ecr:DescribeImages`
• `ecr:BatchGetImage`
• `logs:*`
• `ec2:AuthorizeSecurityGroupIngress`
• `ec2:Describe*`
• `elasticloadbalancing:DeregisterInstancesFromLoadBalancer`
• `elasticloadbalancing:DeregisterTargets`
• `elasticloadbalancing:Describe*`
• `elasticloadbalancing:RegisterInstancesWithLoadBalancer`
• `elasticloadbalancing:RegisterTargets`

Defaults to `None` (one will be created for you).

**task_role_arn:** `str` *(optional)* The ARN for an existing IAM role for tasks to assume. This defines which AWS resources the dask workers can access directly. Useful if you need to read from S3 or a database without passing credentials around.

Defaults to `None` (one will be created with S3 read permission only).

**task_role_policies:** `List[str]` *(optional)* If you do not specify a `task_role_arn` you may want to list some IAM Policy ARNs to be attached to the role that will be created for you.

E.g. if you need your workers to read from S3 you could add `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`.

Default `None` (no policies will be attached to the role)

**cloudwatch_logs_group:** `str` *(optional)* The name of an existing cloudwatch log group to place logs into.

Default `None` (one will be created called `dask-ecs`)

**cloudwatch_logs_stream_prefix:** `str` *(optional)* Prefix for log streams.

Defaults to the cluster name.

**cloudwatch_logs_default_retention:** `int` *(optional)* Retention for logs in days. For use when log group is auto created.

Defaults to `30`.

**vpc:** `str` *(optional)* The ID of the VPC you wish to launch your cluster in.

Defaults to `None` (your default VPC will be used).

**subnets:** `List[str]` *(optional)* A list of subnets to use when running your task.

Defaults to `None`. (all subnets available in your VPC will be used)

**security_groups:** `List[str]` *(optional)* A list of security group IDs to use when launching tasks.

Defaults to `None` (one will be created which allows all traffic between tasks and access to ports 8786 and 8787 from anywhere).
**environment**: dict (optional) Extra environment variables to pass to the scheduler and worker tasks.

Useful for setting EXTRA_APT_PACKAGES, EXTRA_CONDA_PACKAGES and EXTRA_PIP_PACKAGES if you’re using the default image.

Defaults to None.

**tags**: dict (optional) Tags to apply to all resources created automatically.

Defaults to None. Tags will always include {"createdBy": "dask-cloudprovider"}

**find_address_timeout**: int Configurable timeout in seconds for finding the task IP from the cloudwatch logs.

Defaults to 60 seconds.

**skip_cleanup**: bool (optional) Skip cleaning up of stale resources. Useful if you have lots of resources and this operation takes a while.

Default False.

**kwargs**: dict Additional keyword arguments to pass to SpecCluster.

**Attributes**

asynchronous
dashboard_link
observed
plan
requested
scheduler_address
tags

**Methods**

```
adapt(self, *args[, minimum, maximum]) Turn on adaptivity
logs(self) Return logs for the scheduler and workers
new_worker_spec(self) Return name and spec for the next worker
scale(self[, n, memory, cores]) Scale cluster to n workers
scale_up(self[, n, memory, cores]) Scale cluster to n workers
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td></td>
</tr>
<tr>
<td>scale_down</td>
<td></td>
</tr>
<tr>
<td>sync</td>
<td></td>
</tr>
</tbody>
</table>

**logs (self)**

Return logs for the scheduler and workers

**Parameters**

scheduler [boolean] Whether or not to collect logs for the scheduler

workers [boolean or Iterable[str], optional] A list of worker addresses to select. Defaults to
all workers if True or no workers if False

Returns

logs: Dict[str]  A dictionary of logs, with one item for the scheduler and one for each worker

class dask_cloudprovider.FargateCluster(**kwargs)

Deploy a Dask cluster using Fargate on ECS

This creates a dask scheduler and workers on a Fargate powered ECS cluster. If you do not configure a cluster one will be created for you with sensible defaults.

Parameters

kwarg: dict  Keyword arguments to be passed to ECSCluster.

Attributes

asynchronous
dashboard_link
observed
plan
requested
scheduler_address
tags

Methods

adapt(self, *args[, minimum, maximum])  Turn on adaptivity
logs(self)  Return logs for the scheduler and workers
new_worker_spec(self)  Return name and spec for the next worker
scale(self[, n, memory, cores])  Scale cluster to n workers
scale_up(self[, n, memory, cores])  Scale cluster to n workers

close
scale_down
sync
E
ECSCluster (class in dask_cloudprovider), 6

F
FargateCluster (class in dask_cloudprovider), 10

L
logs() (dask_cloudprovider.ECSCluster method), 9