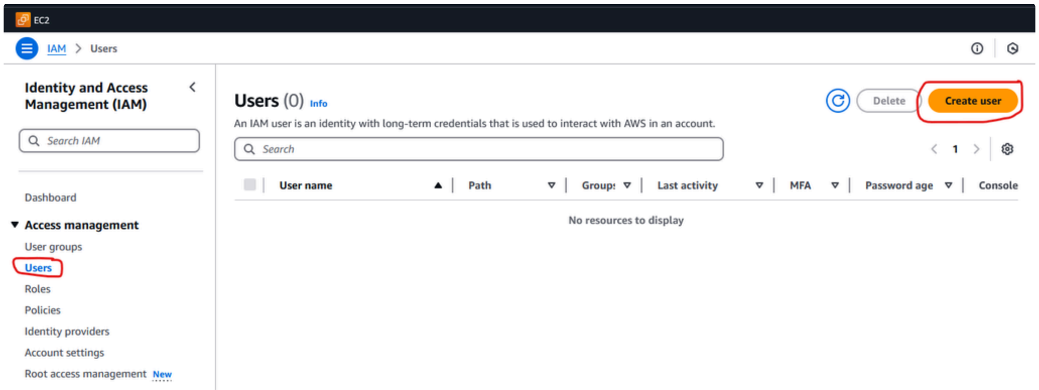
AWS training: BoTorch TorchX - Neural architecture search

EC2 setup (free tier)

Credentials

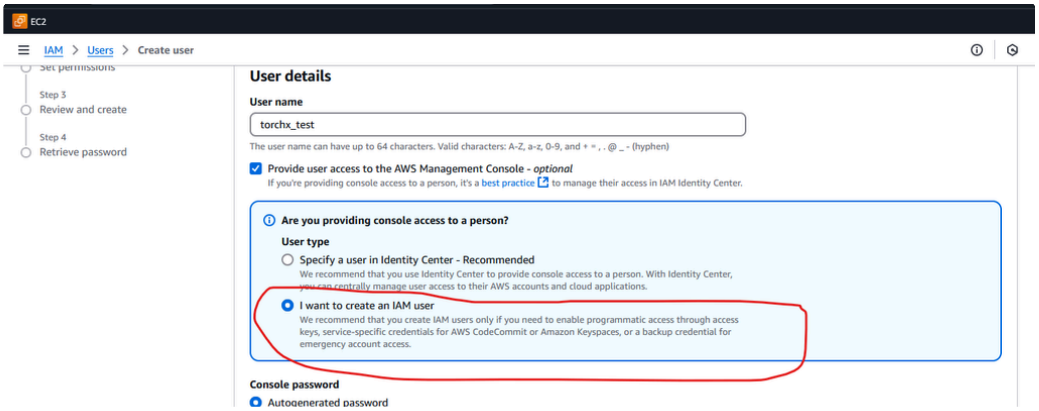


Users (0) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

☐	User name ▲	Path ▽	Group: ▽	Last activity ▽	MFA ▽	Password age ▽	Console

No resources to display



User details

User name

torchx_test

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☑ Provide user access to the AWS Management Console - *optional*

If you're providing console access to a person, it's a best practice to manage their access in IAM Identity Center.

Are you providing console access to a person?

User type

○ Specify a user in Identity Center - Recommended

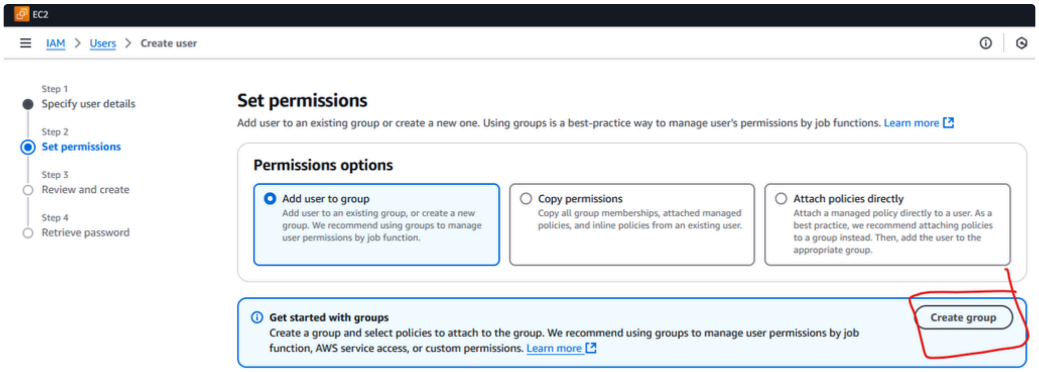
We recommend that you use Identity Center to provide console access to a person. With Identity Center, you can centrally manage user access to their AWS accounts and cloud applications.

● I want to create an IAM user

We recommend that you create IAM users only if you need to enable programmatic access through access keys, service-specific credentials for AWS CodeCommit or Amazon Keyspaces, or a backup credential for emergency account access.

Console password

● Autogenerated password



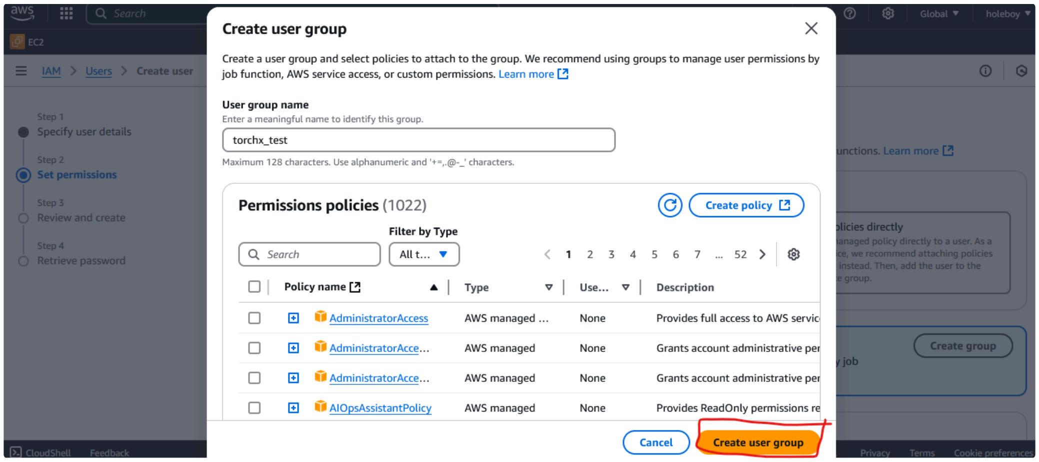
Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

- ● Add user to group: Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- ○ Copy permissions: Copy all group memberships, attached managed policies, and inline policies from an existing user.
- ○ Attach policies directly: Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Get started with groups
Create a group and select policies to attach to the group. We recommend using groups to manage user permissions by job function, AWS service access, or custom permissions. Learn more



Create user group

Create a user group and select policies to attach to the group. We recommend using groups to manage user permissions by job function, AWS service access, or custom permissions. Learn more

User group name
Enter a meaningful name to identify this group.
torchx_test
Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.

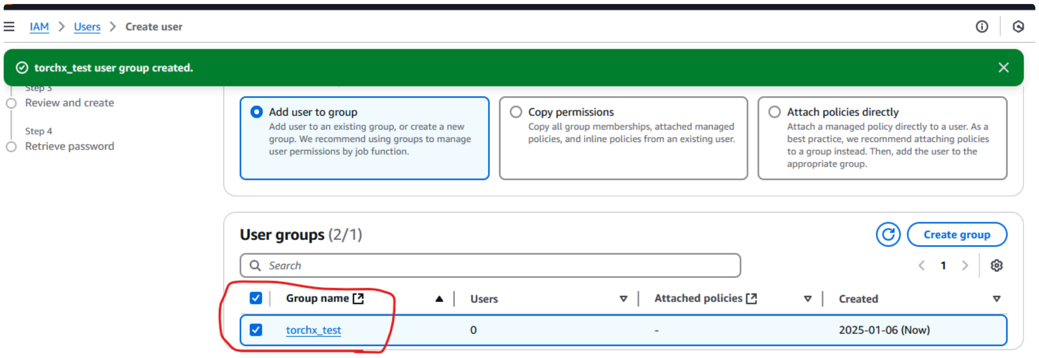
Permissions policies (1022)

Filter by Type: All t...

Search

< 1 2 3 4 5 6 7 ... 52 >

□	Policy name	Type	Use...	Description
□	AdministratorAccess	AWS managed ...	None	Provides full access to AWS servic
□	AdministratorAcce...	AWS managed	None	Grants account administrative per
□	AdministratorAcce...	AWS managed	None	Grants account administrative per
□	AIOpsAssistantPolicy	AWS managed	None	Provides ReadOnly permissions re



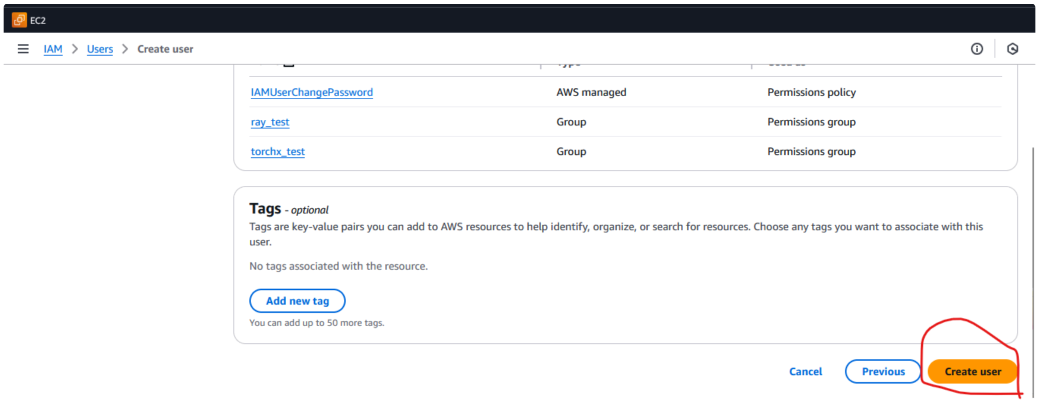
torchx_test user group created.

- ● Add user to group: Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- ○ Copy permissions: Copy all group memberships, attached managed policies, and inline policies from an existing user.
- ○ Attach policies directly: Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

User groups (2/1)

Search

☑	Group name	Users	Attached policies	Created
☑	torchx_test	0	-	2025-01-06 (Now)



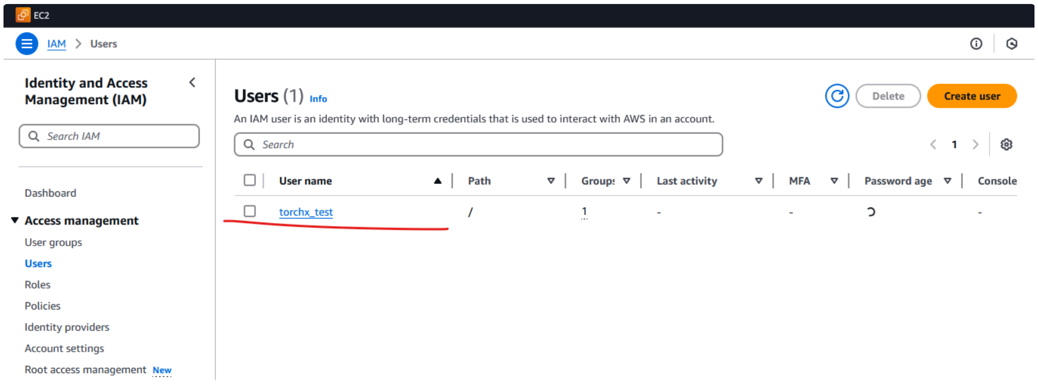
IAMUserChangePassword	AWS managed	Permissions policy
ray_test	Group	Permissions group
torchx_test	Group	Permissions group

Tags - *optional*

Tags are key-value pairs you can add to AWS resources to help identify, organize, or search for resources. Choose any tags you want to associate with this user.

No tags associated with the resource.

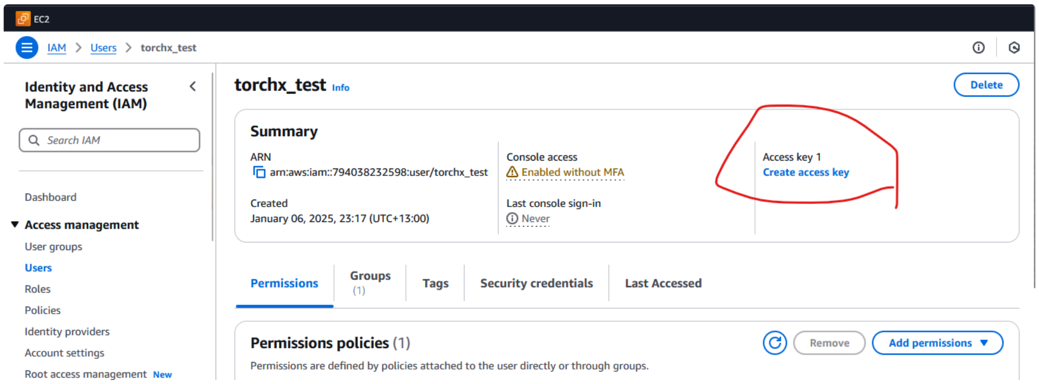
You can add up to 50 more tags.



Users (1) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Path	Groups	Last activity	MFA	Password age	Console
torchx_test	/	1	-	-		-



torchx_test Info

Summary

ARN
arn:aws:iam::794038232598:user/torchx_test

Created
January 06, 2025, 23:17 (UTC+13:00)

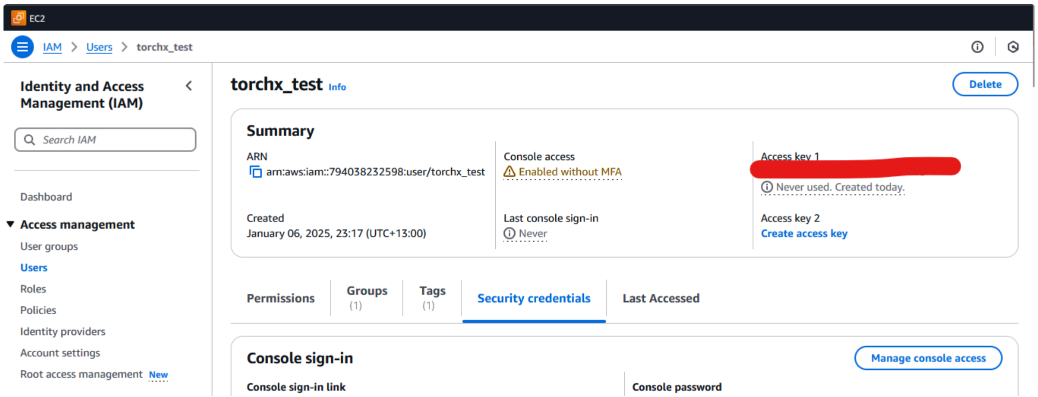
Console access
Enabled without MFA

Last console sign-in
Never

Access key 1
Create access key

Permissions policies (1)

Permissions are defined by policies attached to the user directly or through groups.



torchx_test Info

Summary

ARN
arn:aws:iam::794038232598:user/torchx_test

Created
January 06, 2025, 23:17 (UTC+13:00)

Console access
Enabled without MFA

Last console sign-in
Never

Access key 1 [redacted]
Never used. Created today.

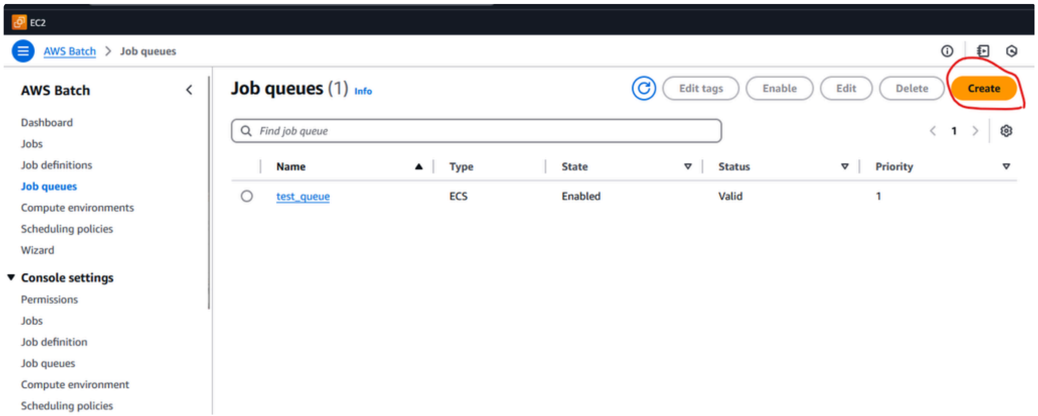
Access key 2
Create access key

Console sign-in

Console sign-in link

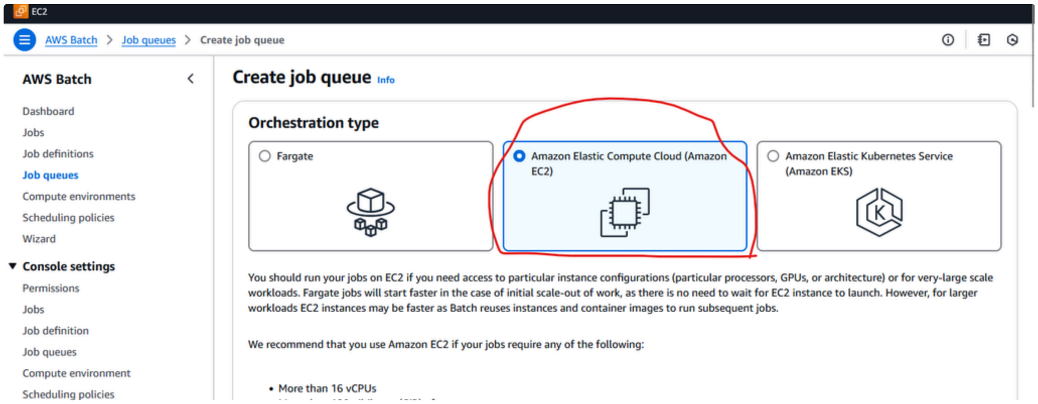
Console password

Create job queue



Job queues (1) Info

Name	Type	State	Status	Priority
test_queue	ECS	Enabled	Valid	1



Create job queue Info

Orchestration type

○ Fargate

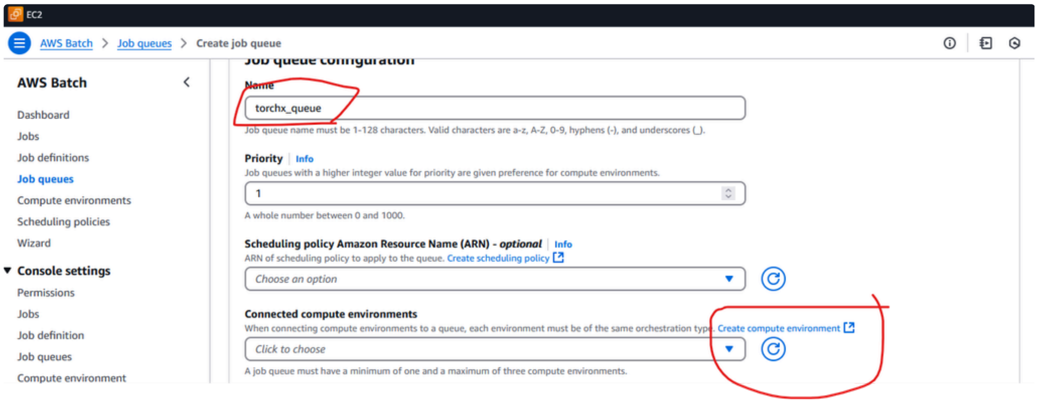
● Amazon Elastic Compute Cloud (Amazon EC2)

○ Amazon Elastic Kubernetes Service (Amazon EKS)

You should run your jobs on EC2 if you need access to particular instance configurations (particular processors, GPUs, or architecture) or for very-large scale workloads. Fargate jobs will start faster in the case of initial scale-out of work, as there is no need to wait for EC2 instance to launch. However, for larger workloads EC2 instances may be faster as Batch reuses instances and container images to run subsequent jobs.

We recommend that you use Amazon EC2 if your jobs require any of the following:

- More than 16 vCPUs



Job queue configuration

Name

torchx_queue

Job queue name must be 1-128 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Priority Info

Job queues with a higher integer value for priority are given preference for compute environments.

1

A whole number between 0 and 1000.

Scheduling policy Amazon Resource Name (ARN) - *optional* Info

ARN of scheduling policy to apply to the queue. Create scheduling policy

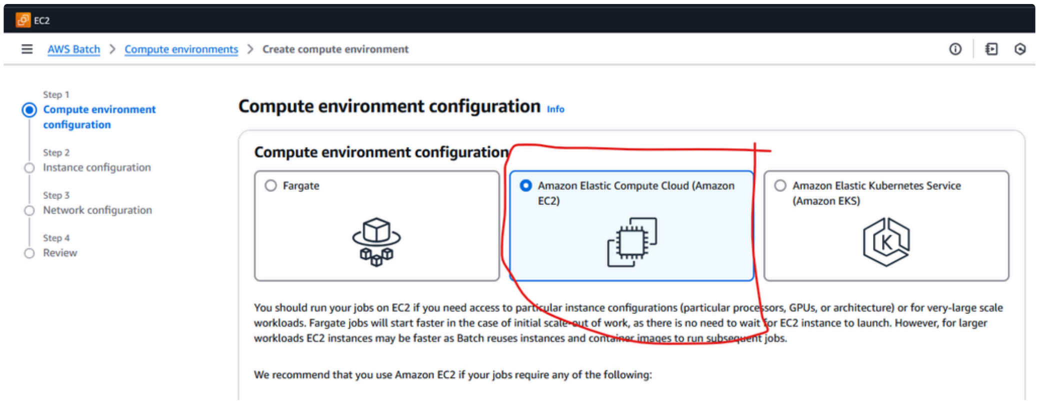
Choose an option

Connected compute environments

When connecting compute environments to a queue, each environment must be of the same orchestration type. Create compute environment

Click to choose

A job queue must have a minimum of one and a maximum of three compute environments.


EC2
AWS Batch > Compute environments > Create compute environment
Step 1
Compute environment configuration
Step 2
Instance configuration
Step 3
Network configuration
Step 4
Review

Compute environment configuration Info

Compute environment configuration

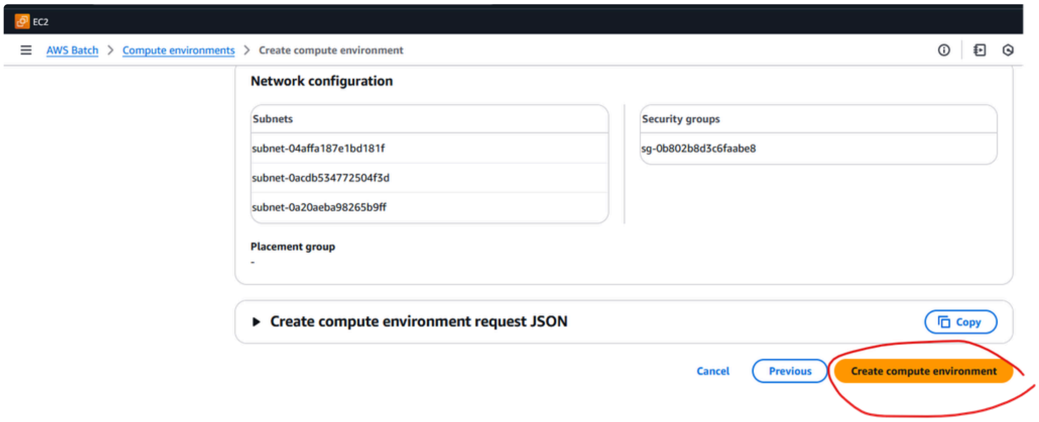
○ Fargate

● Amazon Elastic Compute Cloud (Amazon EC2)

○ Amazon Elastic Kubernetes Service (Amazon EKS)

You should run your jobs on EC2 if you need access to particular instance configurations (particular processors, GPUs, or architecture) or for very-large scale workloads. Fargate jobs will start faster in the case of initial scale-out of work, as there is no need to wait for EC2 instance to launch. However, for larger workloads EC2 instances may be faster as Batch reuses instances and container images to run subsequent jobs.

We recommend that you use Amazon EC2 if your jobs require any of the following:


EC2
AWS Batch > Compute environments > Create compute environment

Network configuration

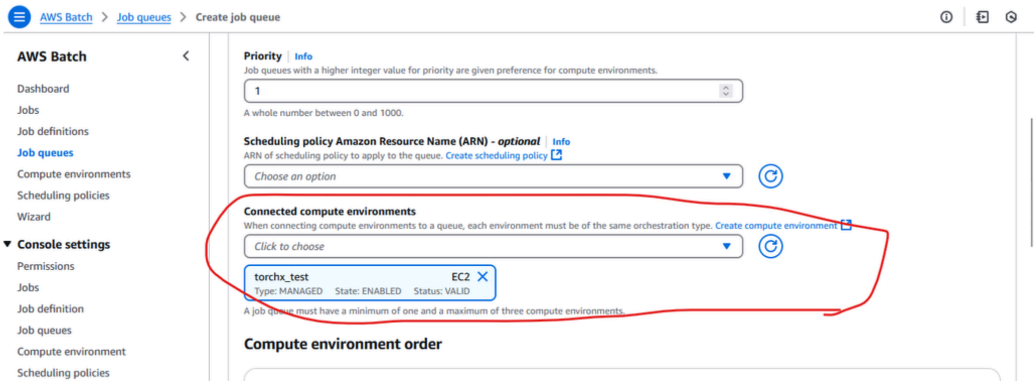
Subnets	Security groups
subnet-04affa187e1bd181f	sg-0b802b8d3c6faabe8
subnet-0acdb534772504f3d	
subnet-0a20aeba98265b9ff	

Placement group

-

▸ Create compute environment request JSON

Copy
Cancel
Previous
Create compute environment



Priority Info

Job queues with a higher integer value for priority are given preference for compute environments.

1

A whole number between 0 and 1000.

Scheduling policy Amazon Resource Name (ARN) - *optional* Info

ARN of scheduling policy to apply to the queue. Create scheduling policy

Choose an option

Connected compute environments

When connecting compute environments to a queue, each environment must be of the same orchestration type. Create compute environment

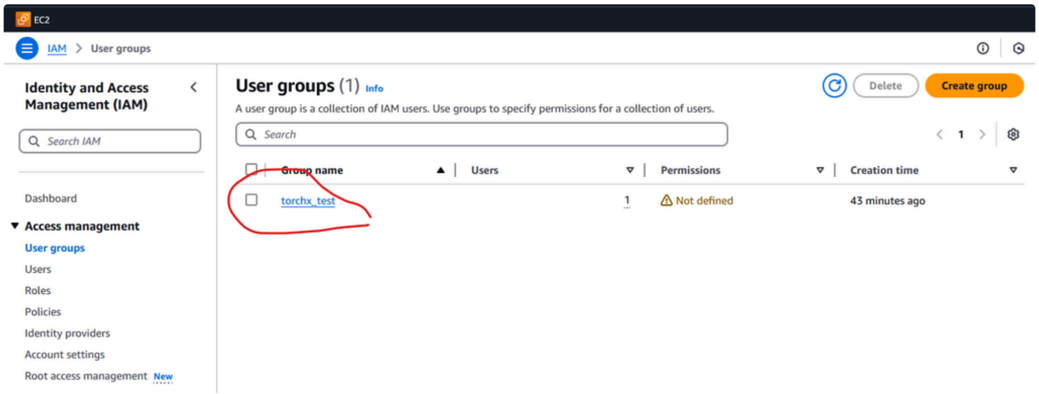
Click to choose

torchx_test EC2
Type: MANAGED State: ENABLED Status: VALID

A job queue must have a minimum of one and a maximum of three compute environments.

Compute environment order

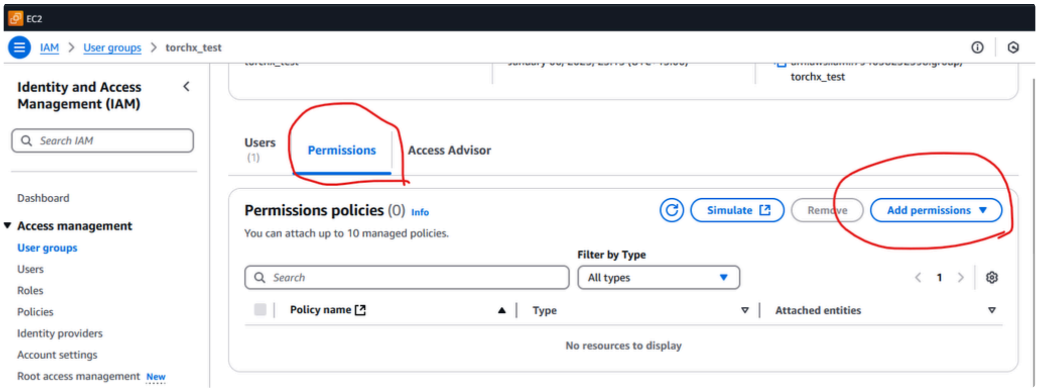
Set user permissions



User groups (1) Info

A user group is a collection of IAM users. Use groups to specify permissions for a collection of users.

Group name	Users	Permissions	Creation time
torchx_test	1	⚠ Not defined	43 minutes ago

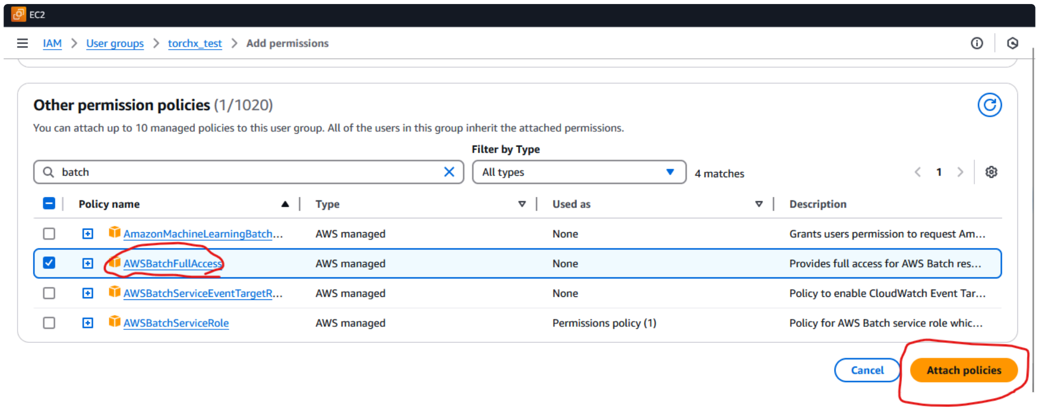


Permissions policies (0) Info

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
No resources to display		



Other permission policies (1/1020)

You can attach up to 10 managed policies to this user group. All of the users in this group inherit the attached permissions.

Filter by Type

batch | All types | 4 matches

	Policy name	Type	Used as	Description
☐	AmazonMachineLearningBatch...	AWS managed	None	Grants users permission to request Am...
☑	AWSBatchFullAccess	AWS managed	None	Provides full access for AWS Batch res...
☐	AWSBatchServiceEventTargetR...	AWS managed	None	Policy to enable CloudWatch Event Tar...
☐	AWSBatchServiceRole	AWS managed	Permissions policy (1)	Policy for AWS Batch service role whic...

Add AWSBatchFullAccess

Python code

Installation

```
poetry add torchx[kubernetes] botorch gpytorch ax-platform tensorboard pytorch-lightning torchvision boto3 botocore torch
```

mnist.py

mnist.py

```

"""
Example training code for ``ax_multiobjective_nas_tutorial.py``
"""

import argparse
import logging
import os
import sys
import time
import warnings

import torch
from IPython.utils import io
from pytorch_lightning import LightningModule, Trainer
from pytorch_lightning import loggers as pl_loggers
from torch import nn
from torch.nn import functional as F
from torch.utils.data import DataLoader
from torchmetrics.functional.classification.accuracy import multiclass_accuracy
from torchvision import transforms
from torchvision.datasets import MNIST

warnings.filterwarnings("ignore")  # Disable data logger warnings
logging.getLogger("pytorch_lightning").setLevel(logging.ERROR)  # Disable GPU/TPU prints


def parse_args():
    parser = argparse.ArgumentParser(description="train mnist")
    parser.add_argument(
        "--log_path", type=str, required=False,
        help="dir to place tensorboard logs from all trials",
        default="/tmp/mnist"
    )
    parser.add_argument(
        "--hidden_size_1", type=int, required=False,
        help="hidden size layer 1", default=16
    )
    parser.add_argument(
        "--hidden_size_2", type=int, required=False,
        help="hidden size layer 2", default=16
    )
    parser.add_argument(
        "--learning_rate", type=float, required=False,
        help="learning rate", default=1e-2
    )
    parser.add_argument(
        "--epochs", type=int, required=False,
        help="number of epochs", default=1
    )
    parser.add_argument(
        "--dropout", type=float, required=False,
        help="dropout probability", default=0.0
    )
    parser.add_argument(
        "--batch_size", type=int, required=False,
        help="batch size", default=32
    )
    return parser.parse_args()

args = parse_args()

PATH_DATASETS = os.environ.get("PATH_DATASETS", ".")


class MnistModel(LightningModule):
    def __init__(self):
        super().__init__()

        # Tunable parameters
        self.hidden_size_1 = args.hidden_size_1
        self.hidden_size_2 = args.hidden_size_2
        self.learning_rate = args.learning_rate
        self.dropout = args.dropout
        self.batch_size = args.batch_size

        # Set class attributes
        self.data_dir = PATH_DATASETS

        # Hardcode some dataset specific attributes
        self.num_classes = 10
        self.dims = (1, 28, 28)
        channels, width, height = self.dims
        self.transform = transforms.Compose(
            [
                transforms.ToTensor(),
                transforms.Normalize((0.1307,), (0.3081,)),
            ]
        )

        # Create a PyTorch model
        layers = [nn.Flatten()]
        width = channels * width * height
        hidden_layers = [self.hidden_size_1, self.hidden_size_2]
        num_params = 0
        for hidden_size in hidden_layers:
            if hidden_size > 0:
                layers.append(nn.Linear(width, hidden_size))
                layers.append(nn.ReLU())
                layers.append(nn.Dropout(self.dropout))
                num_params += width * hidden_size
                width = hidden_size
        layers.append(nn.Linear(width, self.num_classes))
        num_params += width * self.num_classes
```

```python

        # Save the model and parameter counts
        self.num_params = num_params
        self.model = nn.Sequential(*layers)  # No need to use Relu for the last layer

    def forward(self, x):
        x = self.model(x)
        return F.log_softmax(x, dim=1)

    def training_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = F.nll_loss(logits, y)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = F.nll_loss(logits, y)
        preds = torch.argmax(logits, dim=1)
        acc = multiclass_accuracy(preds, y, num_classes=self.num_classes)
        self.log("val_acc", acc, prog_bar=False)
        return loss

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.learning_rate)
        return optimizer

    def prepare_data(self):
        MNIST(self.data_dir, train=True, download=True)
        MNIST(self.data_dir, train=False, download=True)

    def setup(self, stage=None):
        self.mnist_train = MNIST(self.data_dir, train=True, transform=self.transform)
        self.mnist_val = MNIST(self.data_dir, train=False, transform=self.transform)

    def train_dataloader(self):
        return DataLoader(self.mnist_train, batch_size=self.batch_size)

    def val_dataloader(self):
        return DataLoader(self.mnist_val, batch_size=self.batch_size)


def run_training_job():
    import time

    mnist_model = MnistModel()

    # Initialize a trainer (don't log anything since things get so slow...)
    trainer = Trainer(
        logger=False,
        max_epochs=args.epochs,
        enable_progress_bar=False,
        deterministic=True,  # Do we want a bit of noise?
        default_root_dir=args.log_path,
    )

    logger = pl_loggers.TensorBoardLogger(args.log_path)

    print(f"Logging to path: {args.log_path}.")

    # Train the model and log time ⚡
    start = time.time()
    trainer.fit(model=mnist_model)
    end = time.time()
    train_time = end - start
    logger.log_metrics({"train_time": end - start})

    # Compute the validation accuracy once and log the score
    with io.capture_output() as captured:
        val_accuracy = trainer.validate()[0]["val_acc"]
    logger.log_metrics({"val_acc": val_accuracy})

    # Log the number of model parameters
    num_params = trainer.model.num_params
    logger.log_metrics({"num_params": num_params})

    logger.save()

    time.sleep(25)

    # Print outputs
    print(f"train time: {train_time}, val acc: {val_accuracy}, num_params: {num_params}")


if __name__ == "__main__":
    run_training_job()
```

Jupyter notebook

AWS credentials

AWS credentials

```
import os

os.environ["AWS_ACCESS_KEY_ID"] = "foo"
os.environ["AWS_SECRET_ACCESS_KEY"] = "bar"
os.environ["AWS_DEFAULT_REGION"] = "ap-southeast-2"
```

TorchXRunner

TorchXRunner

```
from pathlib import Path

import torchx

from torchx import specs
from torchx.components import utils

src_log_dir = f"{os.getcwd()}/output"
dst_log_dir = "/output"

```

```
def trainer(
    log_path: str,
    hidden_size_1: int,
    hidden_size_2: int,
    learning_rate: float,
    # epochs: int,
    dropout: float,
    batch_size: int,
    trial_idx: int = -1,
) -> specs.AppDef:

    # define the log path so we can pass it to the TorchX ``AppDef``
    if trial_idx >= 0:
        log_path = Path(log_path).joinpath(str(trial_idx)).absolute().as_posix()

    return utils.sh(
        "python",
        "mnist.py",
        "--log_path",
        log_path,
        "--hidden_size_1",
        str(hidden_size_1),
        "--hidden_size_2",
        str(hidden_size_2),
        "--learning_rate",
        str(learning_rate),
        # "--epochs",
        # str(epochs),
        "--dropout",
        str(dropout),
        "--batch_size",
        str(batch_size),
        # other config options
        # name="trainer",
        # script="mnist.py",
        image="ghcr.io/jbris/torchx-aws-test:1.0.0",
        mounts=[
            "type=bind",
            f"src={src_log_dir}",
            f"dst={dst_log_dir}",
            "perm=rwm"
        ]
    )

import tempfile
from ax.runners.torchx import TorchXRunner

# Make a temporary dir to log our results into

scheduler = "aws_batch"
scheduler="local_cwd"
scheduler="local_docker"

ax_runner = TorchXRunner(
    tracker_base="/tmp/",
    component=trainer,
    scheduler=scheduler,
    component_const_params={"log_path": dst_log_dir},
    cfg={"queue": "torchx_queue"},
)
```

Parameter ranges

Parameter ranges

```
from ax.core import (
    ChoiceParameter,
    ParameterType,
    RangeParameter,
    SearchSpace,
)

parameters = [
    # NOTE: In a real-world setting, hidden_size_1 and hidden_size_2
    # should probably be powers of 2, but in our simple example this
    # would mean that ``num_params`` can't take on that many values, which
    # in turn makes the Pareto frontier look pretty weird.
    RangeParameter(
        name="hidden_size_1",
        lower=4,
        upper=8,
        parameter_type=ParameterType.INT,
        log_scale=True,
    ),
    RangeParameter(
        name="hidden_size_2",
        lower=4,
        upper=8,
        parameter_type=ParameterType.INT,
        log_scale=True,
    ),
    RangeParameter(
        name="learning_rate",
        lower=1e-2,
        upper=1e-1,
        parameter_type=ParameterType.FLOAT,
        log_scale=True,
    ),
    # RangeParameter(
    #     name="epochs",
    #     lower=1,
    #     upper=2,
    #     parameter_type=ParameterType.INT,
    # ),
    RangeParameter(
        name="dropout",
        lower=0.0,
        upper=0.5,
        parameter_type=ParameterType.FLOAT,
    ),
    ChoiceParameter(  # NOTE: ``ChoiceParameters`` don't require log-scale
        name="batch_size",
        values=[16, 32],
        parameter_type=ParameterType.INT,
        is_ordered=True,
        sort_values=True,
    ),
]

search_space = SearchSpace(
    parameters=parameters,
    # NOTE: In practice, it may make sense to add a constraint
    # hidden_size_2 <= hidden_size_1
    parameter_constraints=[],
)

from ax.metrics.tensorboard import TensorboardMetric
from tensorboard.backend.event_processing import plugin_event_multiplexer as event_multiplexer

class MyTensorboardMetric(TensorboardMetric):

    # NOTE: We need to tell the new TensorBoard metric how to get the id /
    # file handle for the TensorBoard logs from a trial. In this case
    # our convention is to just save a separate file per trial in
    # the prespecified log dir.
    def _get_event_multiplexer_for_trial(self, trial):
        mul = event_multiplexer.EventMultiplexer(max_reload_threads=20)
        mul.AddRunsFromDirectory(Path(src_log_dir).joinpath(str(trial.index)).as_posix(), None)
        mul.Reload()

        return mul

    # This indicates whether the metric is queryable while the trial is
    # still running. We don't use this in the current tutorial, but Ax
    # utilizes this to implement trial-level early-stopping functionality.
    @classmethod
    def is_available_while_running(cls):
        return True

val_acc = MyTensorboardMetric(
    name="val_acc",
    tag="val_acc",
    lower_is_better=False,
)
model_num_params = MyTensorboardMetric(
    name="num_params",
    tag="num_params",
    lower_is_better=True,
)

```

Run trials

Run trials

```
from ax.core import MultiObjective, Objective, ObjectiveThreshold
from ax.core.optimization_config import MultiObjectiveOptimizationConfig


opt_config = MultiObjectiveOptimizationConfig(
    objective=MultiObjective(
        objectives=[
            Objective(metric=val_acc, minimize=False),
            Objective(metric=model_num_params, minimize=True),
        ],
    ),
    objective_thresholds=[
        ObjectiveThreshold(metric=val_acc, bound=0.94, relative=False),
        ObjectiveThreshold(metric=model_num_params, bound=80_000, relative=False),
    ],
)

from ax.core import Experiment

experiment = Experiment(
    name="torchx_mnist",
    search_space=search_space,
    optimization_config=opt_config,
    runner=ax_runner,
)

total_trials = 48  # total evaluation budget

from ax.modelbridge.dispatch_utils import choose_generation_strategy

gs = choose_generation_strategy(
    search_space=experiment.search_space,
    optimization_config=experiment.optimization_config,
    num_trials=total_trials,
  )

from ax.service.scheduler import Scheduler, SchedulerOptions

scheduler = Scheduler(
    experiment=experiment,
    generation_strategy=gs,
    options=SchedulerOptions(
        total_trials=total_trials, max_pending_trials=4,
        init_seconds_between_polls=5, seconds_between_polls_backoff_factor=1
    ),
)

scheduler.run_n_trials(1)
```

Results

Results

```

from ax.service.utils.report_utils import exp_to_df

df = exp_to_df(experiment)
df.head(10)

from ax.service.utils.report_utils import _pareto_frontier_scatter_2d_plotly

_pareto_frontier_scatter_2d_plotly(experiment)
```