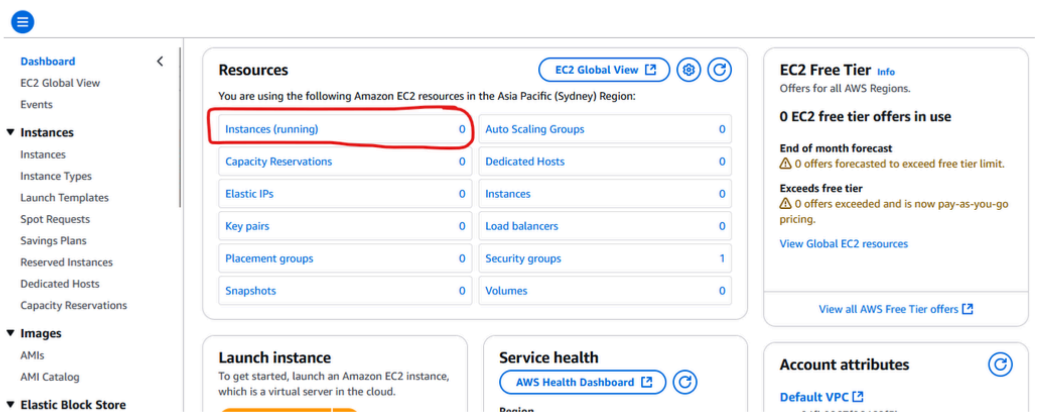
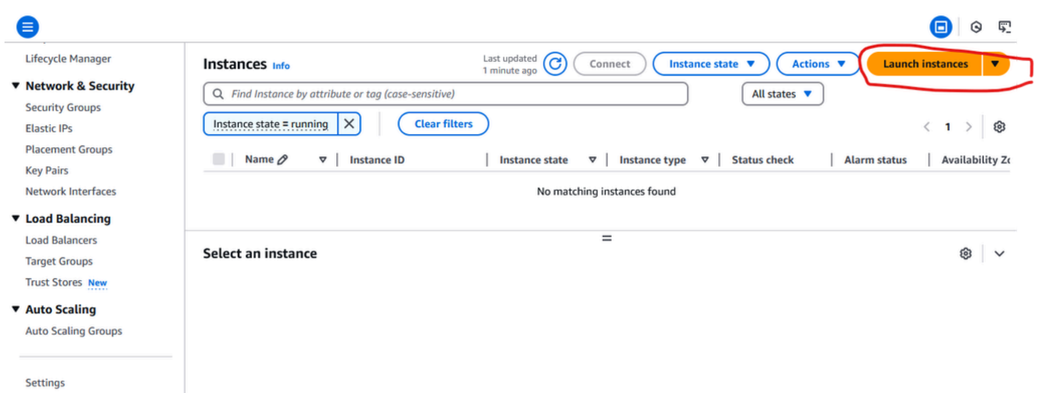
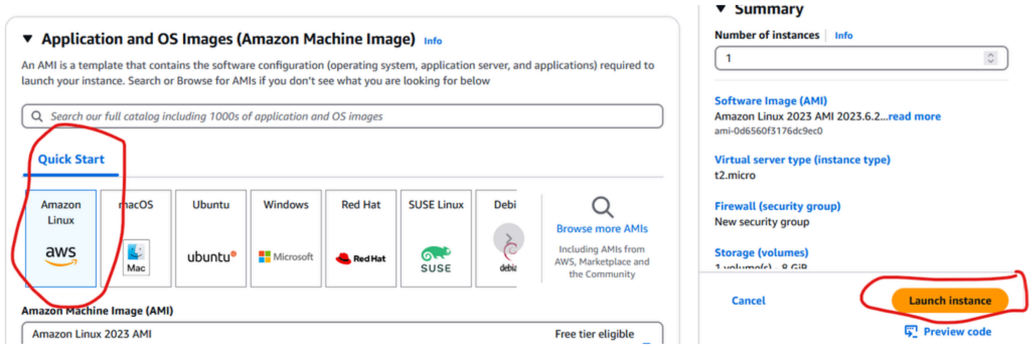
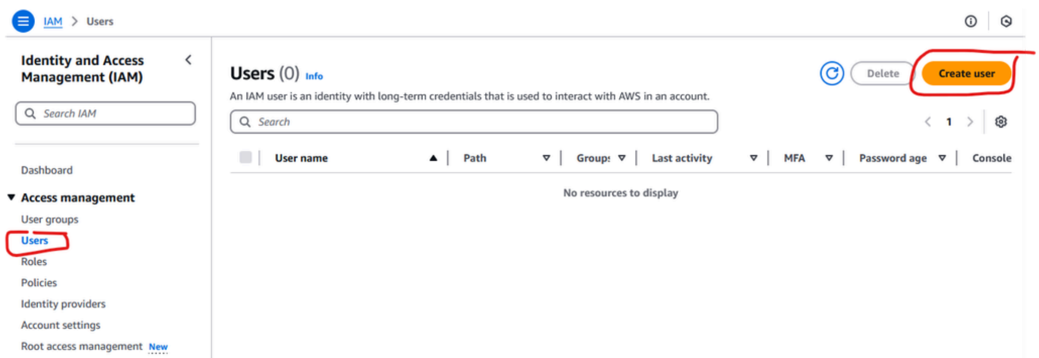
AWS training: Ray cluster deployment

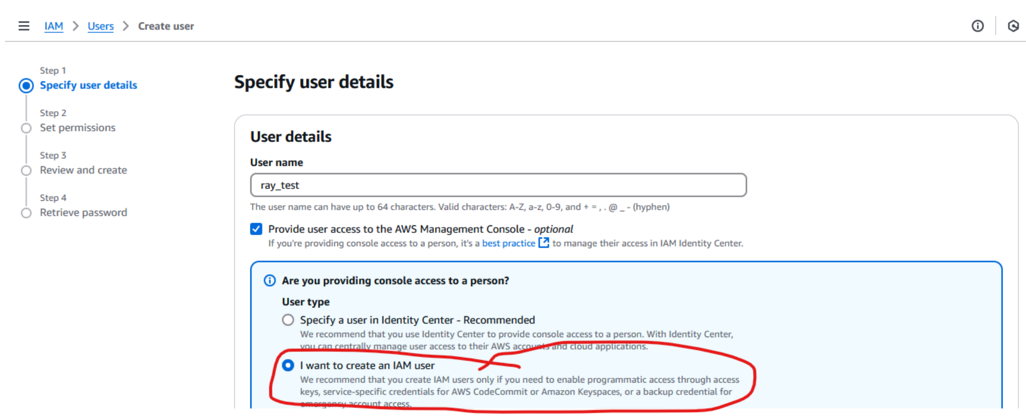
EC2 setup (free tier)

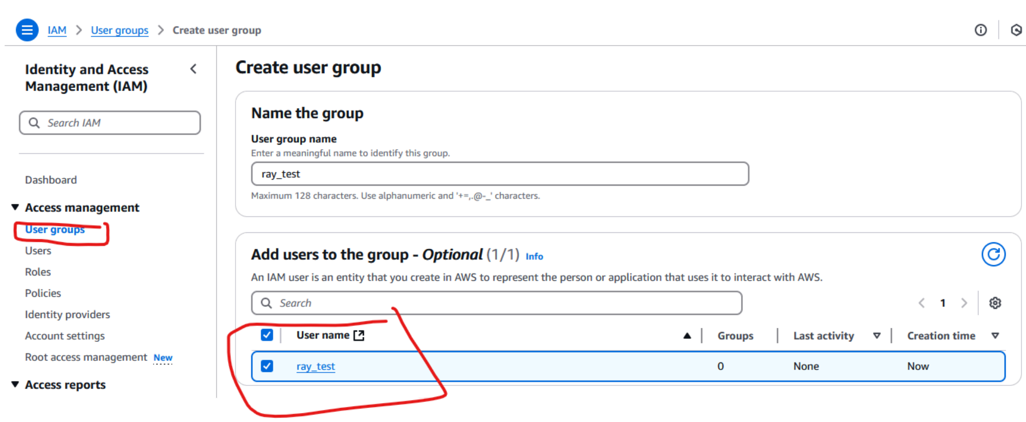


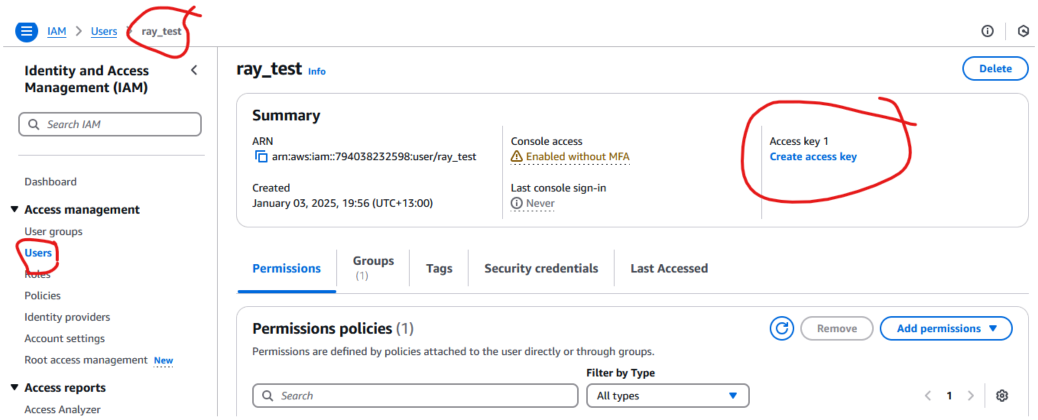


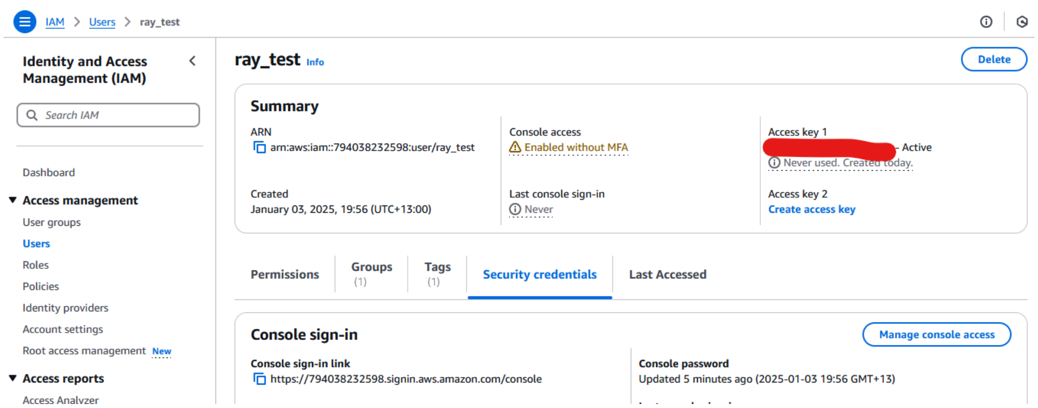

Application and OS Images (Amazon Machine Image) Info
An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below
Search our full catalog including 1000s of application and OS images
Quick Start
Amazon Linux
macOS
Ubuntu
Windows
Red Hat
SUSE Linux
Debi
Browse more AMIs
Including AMIs from AWS, Marketplace and the Community
Amazon Machine Image (AMI)
Amazon Linux 2023 AMI
Free tier eligible
Summary
Number of instances Info
1
Software Image (AMI)
Amazon Linux 2023 AMI 2023.6.2...read more
ami-0d6560f3176dc9ec0
Virtual server type (instance type)
t2.micro
Firewall (security group)
New security group
Storage (volumes)
Cancel
Launch instance
Preview code


IAM > Users
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Users (0) Info
An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.
Delete
Create user
Search
User name
Path
Groups
Last activity
MFA
Password age
Console
No resources to display


IAM > Users > Create user
Step 1
Specify user details
Step 2
Set permissions
Step 3
Review and create
Step 4
Retrieve password
Specify user details
User details
User name
ray_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.


IAM > User groups > Create user group
Identity and Access Management (IAM)
Search IAM
Dashboard
Access management
User groups
Users
Roles
Policies
Identity providers
Account settings
Root access management New
Access reports
Create user group
Name the group
User group name
Enter a meaningful name to identify this group.
ray_test
Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.
Add users to the group - Optional (1/1) Info
An IAM user is an entity that you create in AWS to represent the person or application that uses it to interact with AWS.
Search
User name
Groups
Last activity
Creation time
ray_test
0
None
Now





Installation

```
poetry add boto3 botocore ray[default]
```

Credentials

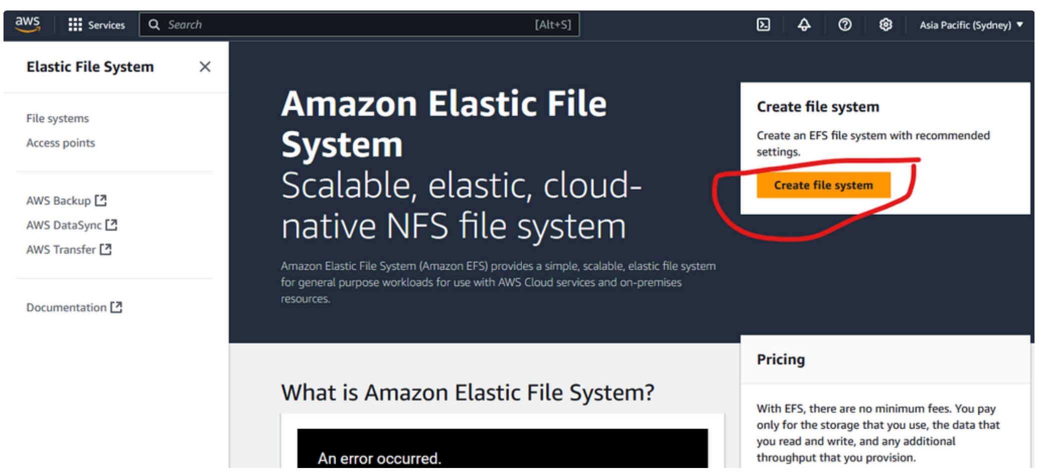
```
# setup AWS credentials using environment variables
export AWS_ACCESS_KEY_ID=foo
export AWS_SECRET_ACCESS_KEY=bar
export AWS_SESSION_TOKEN=baz

# alternatively, you can setup AWS credentials using ~/.aws/credentials file
echo "[default]
aws_access_key_id=foo
aws_secret_access_key=bar
aws_session_token=baz" >> ~/.aws/credentials
```

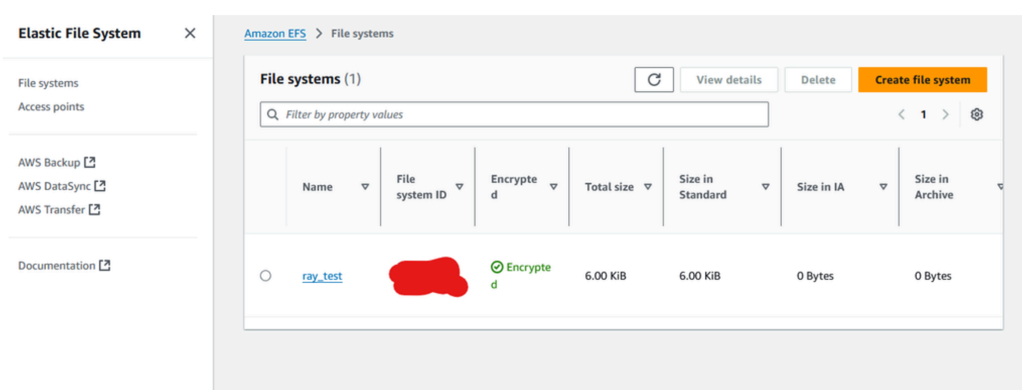
Launch Ray cluster

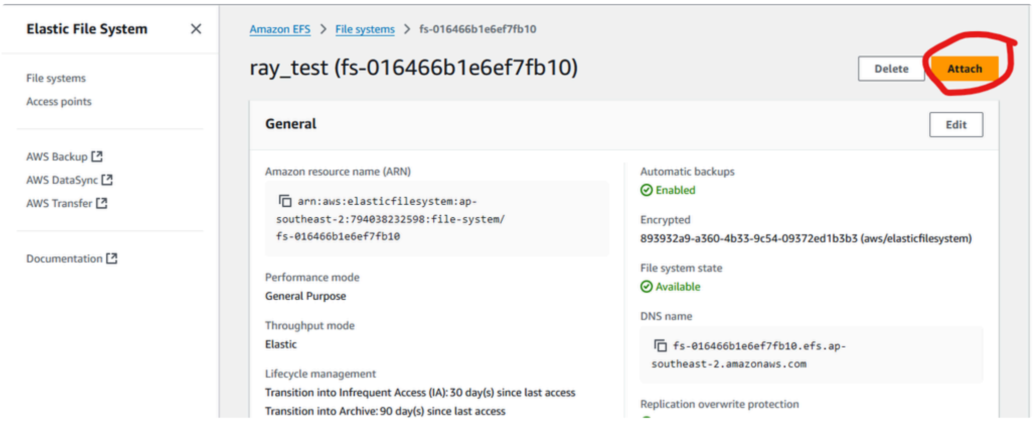
```
# Download the example-full.yaml
wget https://raw.githubusercontent.com/ray-project/ray/master/python/ray/autoscaler/aws/example-full.yaml

```

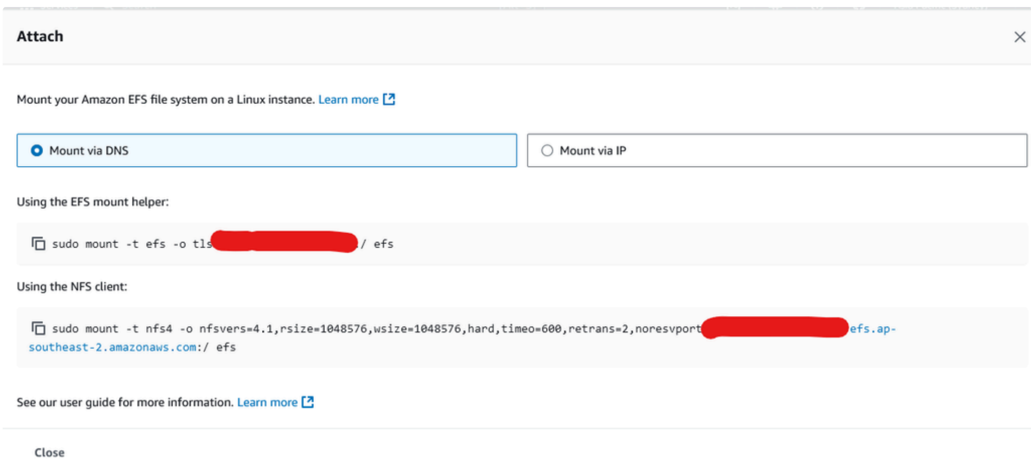
```
# Create or update the cluster. When the command finishes, it will print
# out the command that can be used to SSH into the cluster head node.
ray up example-full.yaml
# This will take a while...

####################################################

# Jump into the cluster
# Get a remote shell on the head node.
ray attach example-full.yaml

####################################################

pwd
ls

# Try running a Ray program.
python -c 'import ray; ray.init()'
exit

# Tear down the cluster.
ray down example-full.yaml
```

EFS









Setup

Replace `{{FileSystemId}}` below

```
# Note You need to replace the {{FileSystemId}} with your own EFS ID before using the config.
# You may also need to modify the SecurityGroupIds for the head and worker nodes in the config file.

setup_commands:
    - sudo kill -9 `sudo lsof /var/lib/dpkg/lock-frontend | awk '{print $2}' | tail -n 1`;
        sudo pkill -9 apt-get;
        sudo pkill -9 dpkg;
        sudo dpkg --configure -a;
        sudo apt-get -y install binutils;
        cd $HOME;
        git clone https://github.com/aws/efs-utils;
        cd $HOME/efs-utils;
        ./build-deb.sh;
        sudo apt-get -y install ./build/amazon-efs-utils*deb;
        cd $HOME;
        mkdir efs;
        sudo mount -t efs {{FileSystemId}}:/ efs;
        sudo chmod 777 efs;
```

IAM Role and EC2 Instance Profile

By default, Ray nodes in a Ray AWS cluster have full EC2 and S3 permissions (i.e. `arn:aws:iam::aws:policy/AmazonEC2FullAccess` and `arn:aws:iam::aws:policy/AmazonS3FullAccess`). This is a good default for trying out Ray clusters but you may want to change the permissions Ray nodes have for various reasons (e.g. to reduce the permissions for security reasons). You can do so by providing a custom `IamInstanceProfile` to the related `node_config`:

```yaml
available_node_types:
  ray.worker.default:
    node_config:
      ...
      IamInstanceProfile:
        Arn: arn:aws:iam::YOUR_AWS_ACCOUNT:YOUR_INSTANCE_PROFILE

```

Accessing S3

In various scenarios, worker nodes may need write access to an S3 bucket, e.g., Ray Tune has an option to write checkpoints to S3 instead of syncing them directly back to the driver.

If you see errors like “Unable to locate credentials”, make sure that the correct `IamInstanceProfile` is configured for worker nodes in your cluster config file. This may look like:

```yaml
available_node_types:
  ray.worker.default:
    node_config:
      ...
      IamInstanceProfile:
        Arn: arn:aws:iam::YOUR_AWS_ACCOUNT:YOUR_INSTANCE_PROFILE

```

You can verify if the set up is correct by SSHing into a worker node and running

```
aws configure list

```

You should see something like

```
      Name                    Value             Type    Location
      ----                    -----             ----    --------
   profile                <not set>             None    None
access_key     ****************XXXX         iam-role
secret_key     ****************YYYY         iam-role
    region                <not set>             None    None

```

Please refer to this discussion for more details on accessing S3.

Amazon CloudWatch

Create `cloudwatch-basic.yaml`

```yaml
# cloudwatch-basic.yaml

provider:
    type: aws
    region: us-west-2
    availability_zone: us-west-2a
```

```
    # Start by defining a `cloudwatch` section to enable CloudWatch integration with your Ray cluster.
    cloudwatch:
        agent:
            # Path to Unified CloudWatch Agent config file
            config: "cloudwatch/example-cloudwatch-agent-config.json"
        dashboard:
            # CloudWatch Dashboard name
            name: "example-dashboard-name"
            # Path to the CloudWatch Dashboard config file
            config: "cloudwatch/example-cloudwatch-dashboard-config.json"

auth:
    ssh_user: ubuntu

available_node_types:
    ray.head.default:
        node_config:
        InstanceType: c5a.large
        ImageId: ami-0d88d9cbe28fac870  # Unified CloudWatch agent pre-installed AMI, us-west-2
        resources: {}
    ray.worker.default:
        node_config:
            InstanceType: c5a.large
            ImageId: ami-0d88d9cbe28fac870  # Unified CloudWatch agent pre-installed AMI, us-west-2
            IamInstanceProfile:
                Name: ray-autoscaler-cloudwatch-v1
        resources: {}
        min_workers: 0
```

```
mkdir cloudwatch
cd cloudwatch
wget https://raw.githubusercontent.com/ray-project/ray/master/python/ray/autoscaler/aws/cloudwatch/example-cloudwatch-agent-config.json
wget https://raw.githubusercontent.com/ray-project/ray/master/python/ray/autoscaler/aws/cloudwatch/example-cloudwatch-dashboard-config.json

cd ..
ray up cloudwatch-basic.yaml
```

Train ML model

```
ray up example-full.yaml

####################################################

# Jump into the cluster
# Get a remote shell on the head node.
ray attach example-full.yaml

####################################################

python -c """
import math
import torch
import gpytorch
from matplotlib import pyplot as plt
import ray
```

```python

ray.init()

@ray.remote
def get_data():
    train_x = torch.linspace(0, 1, 100)
    # True function is sin(2*pi*x) with Gaussian noise
    train_y = torch.sin(train_x * (2 * math.pi)) + torch.randn(train_x.size()) * math.sqrt(0.04)
    test_x = torch.linspace(0, 1, 51)

    return train_x, train_y, test_x


# We will use the simplest form of GP model, exact inference
class ExactGPModel(gpytorch.models.ExactGP):
    def __init__(self, train_x, train_y, likelihood):
        super(ExactGPModel, self).__init__(train_x, train_y, likelihood)
        self.mean_module = gpytorch.means.ConstantMean()
        self.covar_module = gpytorch.kernels.ScaleKernel(gpytorch.kernels.RBFKernel())

    def forward(self, x):
        mean_x = self.mean_module(x)
        covar_x = self.covar_module(x)
        return gpytorch.distributions.MultivariateNormal(mean_x, covar_x)

@ray.remote
def get_model():
    # initialize likelihood and model
    likelihood = gpytorch.likelihoods.GaussianLikelihood()
    model = ExactGPModel(train_x, train_y, likelihood)

    # Find optimal model hyperparameters
    model.train()
    likelihood.train()

    # Use the adam optimizer
    optimizer = torch.optim.Adam(model.parameters(), lr=0.1)  # Includes GaussianLikelihood parameters

    # "Loss" for GPs - the marginal log likelihood
    mll = gpytorch.mlls.ExactMarginalLogLikelihood(likelihood, model)

    return model, likelihood, mll, optimizer

train_x, train_y, test_x = ray.get(get_data.remote())
model, likelihood, mll, optimizer = ray.get(get_model.remote())

@ray.remote
def train_model(training_iter, model, mll, optimizer):

    for i in range(training_iter):
        # Zero gradients from previous iteration
        optimizer.zero_grad()
        # Output from model
        output = model(train_x)
        # Calc loss and backprop gradients
        loss = -mll(output, train_y)
        loss.backward()
        print('Iter %d/%d - Loss: %.3f   lengthscale: %.3f   noise: %.3f' % (
```

```
            i + 1, training_iter, loss.item(),
            model.covar_module.base_kernel.lengthscale.item(),
            model.likelihood.noise.item()
        ))
        optimizer.step()

    return model, mll, optimizer

training_iter = 5
model, mll, optimizer = ray.get(train_model.remote(
    training_iter, model, mll, optimizer
))

@ray.remote
def eval_model(model, likelihood, train_x, train_y, test_x):
    # Get into evaluation (predictive posterior) mode
    model.eval()
    likelihood.eval()

    # Test points are regularly spaced along [0,1]
    # Make predictions by feeding model through likelihood
    with torch.no_grad(), gpytorch.settings.fast_pred_var():
        observed_pred = likelihood(model(test_x))

    with torch.no_grad():
        # Initialize plot
        f, ax = plt.subplots(1, 1, figsize=(4, 3))

        # Get upper and lower confidence bounds
        lower, upper = observed_pred.confidence_region()
        # Plot training data as black stars
        ax.plot(train_x.numpy(), train_y.numpy(), 'k*')
        # Plot predictive means as blue line
        ax.plot(test_x.numpy(), observed_pred.mean.numpy(), 'b')
        # Shade between the lower and upper confidence bounds
        ax.fill_between(test_x.numpy(), lower.numpy(), upper.numpy(), alpha=0.5)
        ax.set_ylim([-3, 3])
        ax.legend(['Observed Data', 'Mean', 'Confidence'])

        f.savefig('/tmp/gp_plot.png')

ray.get(eval_model.remote(
    model, likelihood, train_x, train_y, test_x
))

ray.shutdown()
"""

ls /tmp
exit

####################################################

# Tear down the cluster.
ray down example-full.yaml
```