Customization

This section describes how to customize TorchElastic to fit your needs.

Launcher

The launcher program that ships with TorchElastic should be sufficient for most use-cases (see :ref:`launcher-api`). You can implement a custom launcher by programmatically creating an agent and passing it specs for your workers as shown below.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\elastic\(pytorch-master) (docs) (source) (elastic) customization.rst`, line 9); *backlink*

Unknown interpreted text role "ref".

```
# my_launcher.py

if __name__ == "__main__":
  args = parse_args(sys.argv[1:])
  rdzv_handler = RendezvousHandler(...)
  spec = WorkerSpec(
      local_world_size=args.nproc_per_node,
      fn=trainer_entrypoint_fn,
      args=(trainer_entrypoint_fn args.fn_args,...),
      rdzv_handler=rdzv_handler,
      max_restarts=args.max_restarts,
      monitor_interval=args.monitor_interval,
  )

  agent = LocalElasticAgent(spec, start_method="spawn")
  try:
      run_result = agent.run()
      if run_result.is_failed():
          print(f"worker 0 failed with: run_result.failures[0]")
      else:
          print(f"worker 0 return value is: run_result.return_values[0]")
  except Exception ex:
      # handle exception
```

Rendezvous Handler

To implement your own rendezvous, extend `torch.distributed.elastic.rendezvous.RendezvousHandler` and implement its methods.

Warning

Rendezvous handlers are tricky to implement. Before you begin make sure you completely understand the properties of rendezvous. Please refer to :ref:`rendezvous-api` for more information.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\elastic\(pytorch-master) (docs) (source) (elastic) customization.rst`, line 48); *backlink*

Unknown interpreted text role "ref".

Once implemented you can pass your custom rendezvous handler to the worker spec when creating the agent.

```
spec = WorkerSpec(
    rdzv_handler=MyRendezvousHandler(params),
    ...
)
elastic_agent = LocalElasticAgent(spec, start_method=start_method)
elastic_agent.run(spec.role)
```

Metric Handler

TorchElastic emits platform level metrics (see :ref:`metrics-api`). By default metrics are emitted to */dev/null* so you will not see them. To have the metrics pushed to a metric handling service in your infrastructure, implement a *torch.distributed.elastic.metrics.MetricHandler* and *configure* it in your custom launcher.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\elastic\(pytorch-master)(docs)(source)(elastic)customization.rst`, line 68); *backlink*

Unknown interpreted text role "ref".

```
# my_launcher.py

import torch.distributed.elastic.metrics as metrics

class MyMetricHandler(metrics.MetricHandler):
    def emit(self, metric_data: metrics.MetricData):
        # push metric_data to your metric sink

def main():
  metrics.configure(MyMetricHandler())

  spec = WorkerSpec(...)
  agent = LocalElasticAgent(spec)
  agent.run()
```

Events Handler

TorchElastic supports events recording (see :ref:`events-api`). The events module defines API that allows you to record events and implement custom EventHandler. EventHandler is used for publishing events produced during torchelastic execution to different sources, e.g. AWS CloudWatch. By default it uses *torch.distributed.elastic.events.NullEventHandler* that ignores events. To configure custom events handler you need to implement *torch.distributed.elastic.events.EventHandler* interface and *configure* it in your custom launcher.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\elastic\(pytorch-master)(docs)(source)(elastic)customization.rst`, line 94); *backlink*

Unknown interpreted text role "ref".

```
# my_launcher.py

import torch.distributed.elastic.events as events

class MyEventHandler(events.EventHandler):
    def record(self, event: events.Event):
        # process event

def main():
  events.configure(MyEventHandler())

  spec = WorkerSpec(...)
  agent = LocalElasticAgent(spec)
  agent.run()
```