Anyscale

Ray Foundations Certification Exam Guide

Certification Overview

The Ray Foundations Certification validates a candidate's ability to understand and apply Ray's core concepts, including clusters, distributed primitives (tasks, actors, objects), and foundational libraries (Ray Data, Train, Tune, Serve). It is designed for engineers who are beginning their journey with distributed AI systems and want to demonstrate essential skills with Ray.

- **Name:** Ray Foundations Certification
- **Prerequisites:** None
- **Level:** Beginner
- **Target Audience:** Data Engineers, ML Engineers, Platform/Infra Engineers
- **Format:** Multiple-choice, single-answer questions
- **Purpose:** Validate core understanding of Ray's architecture, primitives, key libraries, and basic production practices.

Domain	Weight (%)	Recommended Reading	Recommended Courses
Ray Clusters/Architecture	10%	Overview, Key Concepts	
Ray Core	20%	Overview, Key Concepts, Tasks, Actors, Objects	
Ray Data	20%	Overview, Quickstart, Key Concepts, Batch Inference, Loading Data, Inspecting Data, Transforming Data, Shuffling Data, Saving Data	Batch Inference with Ray Data, Data Processing with Ray Data
Ray Train	20%	Overview, PyTorch Guide, Data Loading and Preprocessing, Configuring Scale and GPUs, Monitoring and Logging Metrics, Saving and Loading Checkpoints	Distributed Training with Ray Train
Ray Tune	10%	Overview, Quickstart, Key Concepts, Running Basic Experiments, Search Spaces, Resources	
Ray Serve	20%	Overview, Quickstart, Key Concepts, Model Composition, Autoscaling	Online Model Serving with Ray Serve

Learning Objectives

This certification will test for knowledge of these key concepts:

Ray Clusters

- Describe the components of a Ray cluster (head node, worker nodes, GCS, autoscaler).
- Explain how the Ray autoscaler decides to scale up or down.
- Explain how Ray implements distributed memory.
- Describe Ray's distributed scheduler.
- Identify the responsibilities of the Ray head node.
- Demonstrate knowledge of the different options for deploying Ray clusters.

Ray Core

- Identify which Ray primitive (tasks, actors) is most appropriate for a given use case.
- Explain how Ray schedules and retries tasks under different conditions.
- Demonstrate understanding of actor handles and how they are used to interact with actors across a cluster.
- Describe how actor methods are executed, scheduled, and retried.
- Explain how Ray's object store enables distributed memory sharing.
- Demonstrate how to use placement groups for resource-aware execution strategies.
- Explain how to specify and manage runtime dependencies in Ray.

Ray Data

- Identify when Ray Data is an appropriate tool.
- Explain Ray Data's lazy execution model and how it optimizes plans.
- Demonstrate how Ray Data ingests data from different sources such as S3, GCS, and local files.
- Differentiate between local and global shuffle operations in Ray Data.
- Explain how to specify resources for different stages in a Ray Data pipeline.
- Interpret code that uses Ray Data's streaming batch APIs (e.g. `read, map_batches, write`).
- Demonstrate how Ray Data saves a dataset to persistent storage.

Ray Tune

- Explain the benefits of using Ray Tune for scalable hyperparameter tuning.
- Demonstrate basic Ray Tune API usage for hyperparameter tuning.
- Define search spaces using Ray Tune's `tune.choice, tune.uniform`, etc.
- Explain the role of schedulers and search algorithms in Ray Tune.
- Demonstrate resource-aware tuning using the `tune.with_resources` method.

Ray Train

- Explain when and why to use Ray Train.
- Demonstrate how to orchestrate distributed training of a PyTorch model with Ray Train's `Trainer` API.
- Demonstrate how to configure and launch a Ray Train run.
- Explain checkpointing using `train.report` and `train.get_checkpoint`.
- Identify when and how to integrate Ray Data with Ray Train.

- Explain how Ray Data and Ray Train handle data sharding across workers.
- Demonstrate techniques for ensuring reproducibility in Ray Train experiments.

Ray Serve

- Explain when and why to use Ray Serve.
- Demonstrate how to deploy an application using the Ray Serve API with the @serve.deployment decorator.
- Demonstrate how to specify resources (CPU, GPU, memory) for Ray Serve deployments.
- Explain what ingress deployments are and how they are used.
- Demonstrate how to integrate FastAPI with Ray Serve for request validation and API management.
- Demonstrate how to use Ray Serve to build multi-model or multi-step inference pipelines.
- Explain how autoscaling works in Ray Serve.

Sample Questions

The following sample questions illustrate the format and style of the exam; they are not comprehensive.

Q. What is one benefit of using Ray Train's Trainer API for distributed deep learning?

1. It automatically converts all models to use the ONNX runtime
2. **It handles orchestration of distributed workers, fault tolerance and integrates with Ray Data for data sharding.**
3. It replaces the need for distributed training libraries like PyTorch.
4. It always guarantees model convergence regardless of the hardware or training configuration.

Q. You're running distributed training using Ray Train with four workers on GPU nodes. Halfway through the training, one of the workers crashes due to a transient hardware issue. Which of the following is true, assuming checkpointing was correctly implemented?

1. The entire job must be restarted from the beginning, as Ray Train does not support partial recovery.
2. **Ray Train will automatically retry the training run by restoring state from the most recent checkpoint assuming you set max_failures appropriately.**
3. Training will continue on the remaining workers without restarting.
4. The failed worker is replaced with a redundant replica already running on standby.

Exam Logistics

Duration: 120 minutes
Language: English
Number of Questions: 60
Passing Score: 70%
Delivery Method: Online