

Running Examples via Scheduler

Running Examples via Scheduler

Для каждой модели в `ray_wrapper/examples/` рядом с `train.py`/`tune.py` лежит `schedule_*.py` — тот же пример, но запускаемый не напрямую, а через постановку задачи в очередь сервиса scheduler (порт 8264, подробнее — книга «Документация MLumina», страница Scheduler).

????? ????????

```
import requests

SCHEDULER_URL = "http://127.0.0.1:8264/add_task"

task_json = {
    "name": "fcnn",
    "entrypoint": "python ray_wrapper/examples/fcnn/tune.py",
    "runtime_env": {
        "working_dir": "/home/ray/src/mlumina",
        "pip": ["ray", "torch", "tqdm"],
    },
    "priority": 0,
    "cpu": 0,
    "gpu": 0,
    "memory": 1,
    "timeout": 0,
}

response = requests.post(SCHEDULER_URL, json=task_json, timeout=60)
print(response.status_code)
```

```
print(response.json())
```

Существующие файлы: `lin_reg/schedule_lin_reg_tune.py`, `svm/schedule_svm_tune.py`, `fcnn/schedule_fcnn_train.py`, `fcnn/schedule_fcnn_tune.py`, `cnn/schedule_cnn_train.py`, `cnn/schedule_cnn_tune.py`, `vit/cifar_10/schedule_cifar10_train.py`, `vit/cifar_10/schedule_cifar10_tune.py`, `vit/cifar_100/schedule_cifar100_train.py`, `vit/cifar_100/schedule_cifar100_tune.py` — по одному на каждую пару train/tune, кроме lin_reg и svm (у них есть только tune).

?? ??? ?????????? ??????????

- `working_dir` — **путь внутри контейнера**, а не на машине, откуда отправляется HTTP-запрос (`/home/ray/src/mlumina` — это то, что смонтировано в контейнер `scheduler` в `docker-compose.yml`). Отправлять такие запросы имеет смысл из среды, где этот путь существует и виден Ray-кластеру — то есть из контейнеров стека, а не с произвольного ноутбука разработчика.
- **Поле** `"name": "fcnn"` **захардкожено одинаково во всех** `schedule_*.py`, включая `schedule_lin_reg_tune.py`, `schedule_svm_tune.py`, `schedule_cifar10_train.py` и т.д. — это сору-пасте-артефакт, не ошибка запуска: имя используется только для генерации `Task.id` (`mlumina-{name}-priority-...-{uuid}`) и отображения в очереди/дашборде, само задание при этом запускает правильный `entrypoint`. Тем не менее в дашборде планировщика все такие задачи будут выглядеть как "fcnn" независимо от реальной модели — при желании стоит поправить `name` под конкретную задачу перед отправкой.
- `"cpu": 0, "gpu": 0, "memory": 1` — это ресурсы, которые **планировщик** резервирует для допуска задачи в очередь (проверка `_check_resources_by_allocation` против `ray.cluster_resources()`), а не то, что реально ограничивает Ray-задачу при выполнении — фактическое потребление ресурсов определяется тем, что делает сам `entrypoint`-скрипт (`scale_config`, `resources=` и т.п. внутри train/tune-функций). `Scheduler.add_task_to_queue` отклоняет задачу только если `cpu == 0 and gpu == 0 and memory == 0` одновременно — здесь `memory=1` этого не допускает, так что все такие задания успешно проходят допуск.
- `"timeout": 0` — без ограничения по времени выполнения (планировщик не станет автоматически отменять задачу по таймауту).

??? ?????????? ????????

Изнутри контейнера, у которого есть доступ к scheduler по сети (например, `head/worker`, либо `curl/python` с той же Docker-сети):

```
python ray_wrapper/examples/fcnn/schedule_fcnn_tune.py
```

Прогресс — через `GET http://<host>:8264/get_task_queue`, `GET /get_running_tasks_list` или веб-дашборд по адресу `http://<host>:8264/`.

Revision #1

Created 2026-09-23 06:54:51 UTC by Маслов Алексей

Updated 2026-09-23 06:54:51 UTC by Маслов Алексей