Torchserve部署指南

本篇文档旨在介绍TorchServe部署相关内容。为便于实际操作，同时也会介绍部分基础概念与rancher相关内容。

1. 前置知识

1.1 模型部署介绍

模型训练好后，需要上线部署为服务。部署后的服务一般以url的形式，供以用户通过get/post请求得到返回结果。

常见的服务部署框架：

- TorchServe：深度学习模型部署框架。
  - 优点：方便部署pytorch的深度学习模型，围绕**模型**的功能较全面。
  - 缺点：主要针对深度学习模型。
- Flask：使用 Python 编写的轻量级 Web 应用框架。
  - 优点：灵活度较高，性能相对于django更高，扩展性强，适合做web开发的api。
  - 缺点：组件少，特定需求需要自己装一些插件或单独实现业务逻辑能够满足。
- Django：使用Python编写的Web应用框架，采用了MTV的框架模式。
  - 优点：功能全面，第三方库丰富，快速稳定，适合开发相对较大的项目。
  - 缺点：灵活度较低。

1.2 Rancher介绍

Rancher 是一个开源的企业级全栈化**容器**部署及管理平台，可以看作是 Docker 的图形化管理界面。

它为容器提供基础架构服务，可以让 CNI 兼容的网络服务、存储服务、主机管理、负载均衡、防火墙等服务跨越公有云、私有云、虚拟机、物理机环境运行，真正实现一键式应用部署和管理。

除此之外它还提供了诸如负载均衡、弹性扩容等高级功能，也是一个比较常用的运维工具。

2. TorchServe

2.1 TorchServe介绍

TorchServe 由 AWS 和 Facebook 合作推出，并作为 PyTorch 开源项目的一部分提供。

通过 TorchServe，PyTorch 可以更快地将其模型投入生产，而无需编写自定义代码。除了提供低延迟预测 API 之外，TorchServe 还为诸如对象检测和文本分类等常用应用程序嵌入了默认处理程序。

此外，TorchServe 还包括多模型服务、A/B 测试模型版本控制、监控指标以及用于集成应用程序的 RESTful 终端节点。不负众望，TorchServe 支持任何机器学习环境，包括 Amazon SageMaker、容器服务和 Amazon Elastic Compute Cloud (EC2)。

2.2 TorchServe安装

1. 安装torchserve

```
pip install torchserve
```

2. 安装torchserve-model-archiver

torchserve-model-archiver用于创建训练过的神经网络模型的档案，可供TorchServe推理使用，最终会打包为一个.mar文件。

```
pip install torch-model-archiver
```

3. 配置java环境

torchserve需要Java的环境依赖。故下载并解压jdk11 jar包，并配置

a. 下载并解压jdk tar包到指定目录下

```
wget https://download.java.net/openjdk/jdk11/ri/openjdk-11+28_linux-x64_bin.t
tar -xzvf jdk-11.0.14_linux-x64_bin.tar.gz
```

b. 配置java环境

i. 此步需要打开profile文件，配置环境变量。

```
# 1. 打开profile
vim /etc/profile

# 2.在profile中添加以下内容
export JAVA_HOME=/${your_path}/java/java11/jdk-11.0.14
```

```
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=$JAVA_HOME/lib

# 3.刷新配置文件
source /etc/profile
```

ii. 可以使用如下命令判断是否安装成功

```
java -version
```

```
[root@dev3090-zhanglin-intern-gpu-7d7975cfd5-dnjzl:~# java -version
openjdk version "11" 2018-09-25
OpenJDK Runtime Environment 18.9 (build 11+28)
OpenJDK 64-Bit Server VM 18.9 (build 11+28, mixed mode)
```

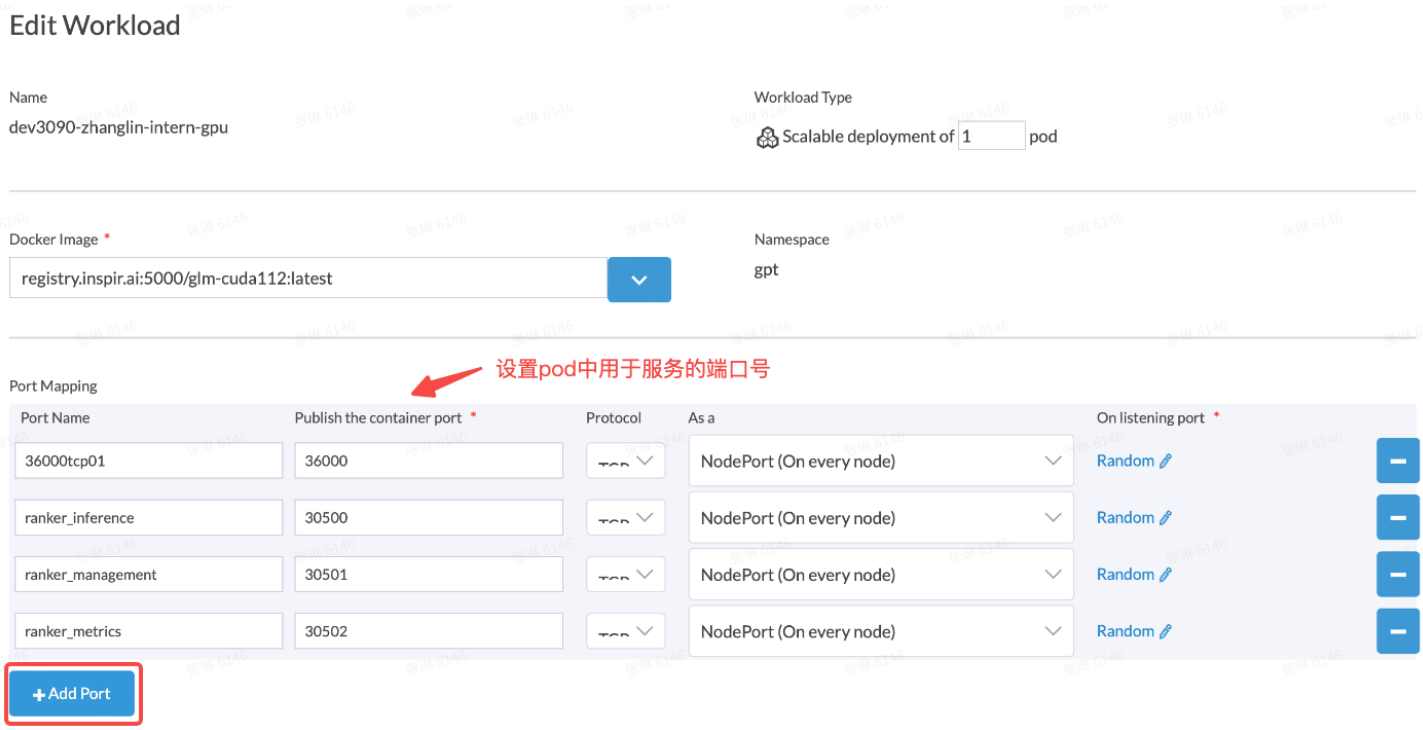
3. 模型部署步骤

3.1 rancher配置

由于服务部署在Pod之中，外部用户无法直接访问，需要通过**Pod暴露的端口**才能够访问。

登录rancher，在自己需要部署服务的Pod中配置端口的名称和需要暴露的端口。若不好确定端口是否冲突，建议「On listening port」设置为「random」。

- Publish the container port：在Pod内部署模型指定的端口。
- On listening port：映射到外部暴露出来的端口。



> **注意**
>
> 配置完毕后，Pod会自动刷新。如涉及到虚拟环境相关内容，建议做好备份，便于后续恢复。

3.2 torchserve部署模型

1. 模型文件打包，输出mar文件

参数说明：

- model-name：导出的模型文件名，如果未指定--export-path则保存在当前工作目录下，否则在exportpath下
- serialized-file：pt或pth文件路径
- handler：torchserve默认的handler名称，或者是自定义torchserve的推理逻辑（handler python file path）
- version：版本

```
torch-model-archiver --model-name ${model_name} --version ${version} --serialize

# 举例
torch-model-archiver --model-name ranker_zh --version 1.0 --serialized-file /cep
```

2. 启动服务

参数说明：

- start：开启模型服务
- model-store：模型存储位置
- models：指定mar文件，有以下几种参数形式
  - 只部署一个模型：${mar file name}.mar
  - 部署多个模型：多个<model_name>=<model_path>对
  - 部署某目录下所有模型：all （model-store目录下的所有模型）
- ncs：no config snapshots，禁用快照功能
- ts-config：torchserve进阶配置文件，可指定一些参数。在此次部署过程中主要涉及了服务所占用的端口号。
  - inference_address：与模型推理相关的内容，如健康检查、模型预测。
  - management_address：服务本身的管理，如模型注册、分配worker数量、注销模型、设置版本等内容。
  - metrics_address：可查看一些指标信息。

```
inference_address=http://0.0.0.0:30500
management_address=http://0.0.0.0:30501
metrics_address=http://0.0.0.0:30502
```

```
torchserve --start --model-store ${mar FILE PATH} --models ${mar file name}.mar

# 举例
torchserve --start --model-store models/  --models ranker_zh=ranker_zh.mar ranke
```

3.3 在服务开启状态下注册新模型

训练好新的模型后，可以在服务开启的状态下进行添加。

首先使用3.2中第一步的方式打包模型为mar文件，然后采用如下命令注册新模型。

```
curl -X POST http:/${management_address}/models?url=${mar_file_path}

# 举例
curl -X POST http://localhost:30501/models/?url=/cephfs/zl/model/ranking_server_
```

注册成功后会返回成功信息，但调用时会遇到如下错误：

> {
>
> "code": 503,
>
> "type": "ServiceUnavailableException",
>
> "message": "Model \"ranker_zh_in_batch_neg\" has no worker to serve inference request. **Please use scale workers API to add workers**."
>
> }

此时需要我们人为为新模型划分工作器数量，以便更好地服务不同的推理请求负载。

```
curl -v -X PUT "${management_address}/models/noop?min_worker=${min_worker_num}&s

# 举例
curl -v -X PUT "http://localhost:30501/models/ranker_zh_in_batch_neg?min_worker=
```

4. 其他常用命令

4.1 Prediction API

4.1.1 显示 inference APIs所有信息

```
curl -X OPTIONS ${inference_address}
```

4.1.2 health check

如模型正常运行，输入命令会返回如下内容。

```
# curl http://localhost:30500/ping
{
  "status": "Healthy"
}
```

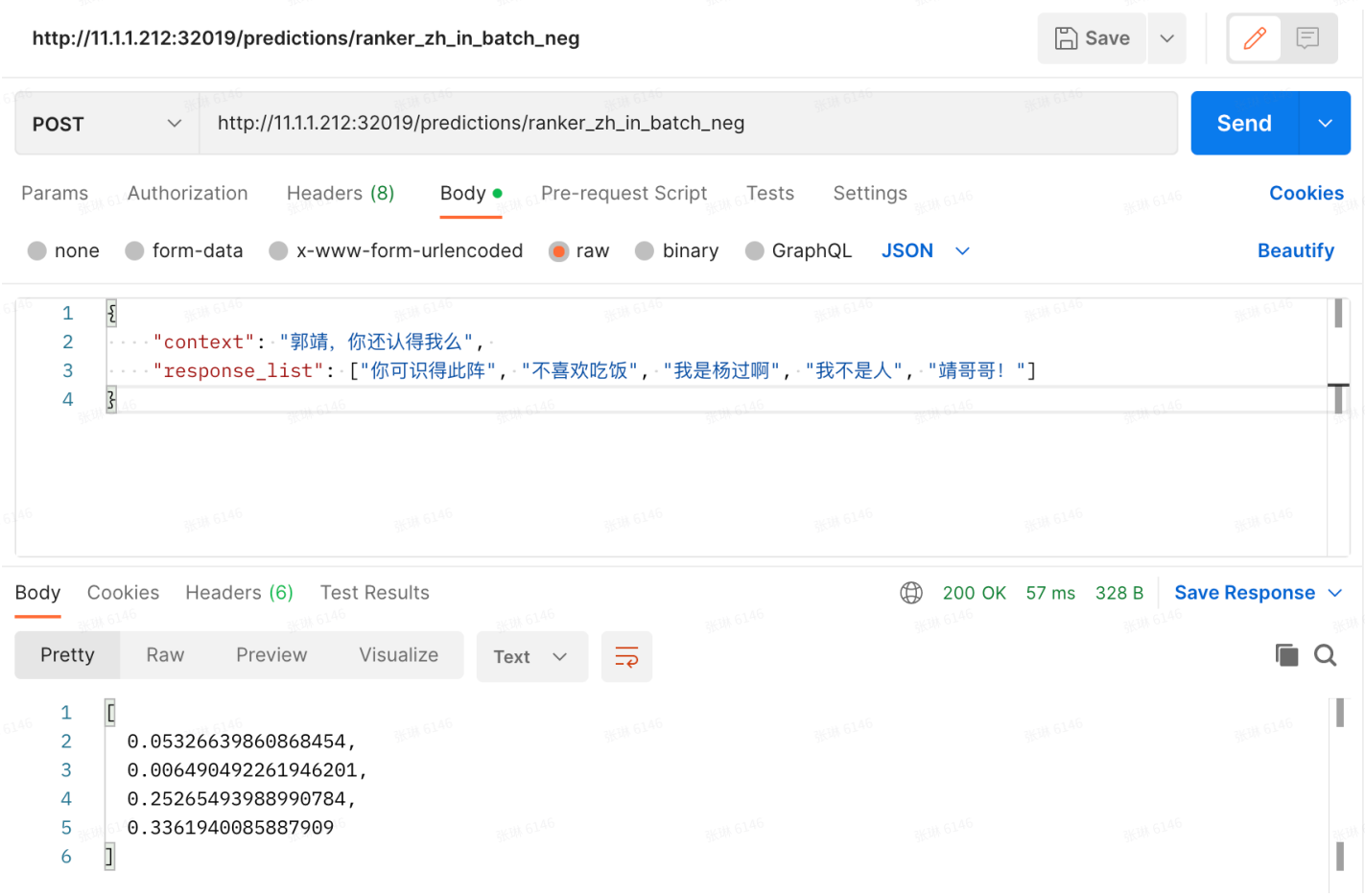
```
curl ${inference_address}/ping
```

4.1.3 调用模型得到返回结果

```
curl -X POST ${inference_address}/predictions/${model_name} -T ${predict_file}

# 举例
curl -X POST http://localhost:8080/predictions/ranker_zh -T test_ch.txt
```

也可直接使用Postman调用（与curl命令同理）



4.2 Management API

4.2.1 查看已注册模型

```
curl ${management_address}/models

# 举例
curl http://localhost:30501/models
```

返回结果如下

```
{
  "models": [
    {
      "modelName": "ranker_en",
      "modelUrl": "ranker_en.mar"
    },
    {
      "modelName": "ranker_zh",
      "modelUrl": "ranker_zh.mar"
    },
    {
      "modelName": "ranker_zh_in_batch_neg",
      "modelUrl": "/cephfs/zl/model/ranking_server_torchserve/models/ranker_zh_in_batch_neg.mar"
    }
  ]
}
```


4.2.2 查看模型运行状态

```
curl ${management_address}/models/${model_name}
# 举例
curl http://localhost:30501/models/ranker_zh
```

返回结果如下


```
[
  {
    "modelName": "ranker_zh",
    "modelVersion": "1.0",
    "modelUrl": "ranker_zh.mar",
    "runtime": "python",
    "minWorkers": 8,
    "maxWorkers": 8,
    "batchSize": 1,
    "maxBatchDelay": 100,
    "loadedAtStartup": true,
    "workers": [
      {
        "id": "9000",
        "startTime": "2022-09-16T12:21:45.507Z",
        "status": "READY",
        "memoryUsage": 3223470080,
        "pid": 27417,
        "gpu": true,
        "gpuUsage": "gpuId::1 utilization.gpu [%]::0 % utilization.memory [%]::0 % memory.used [
MiB]::4898 MiB"
      },
```


4.2.3 注销模型

```
curl -X DELETE ${management_address}/models/${model_name}/${version}
```

参考内容

TorchServer - PyTorch/Serve master documentation

torchserve使用教程_随便写点笔记的博客-CSDN博客_torchserve

TorchServe部署pytorch模型_lulu_陌上尘的博客-CSDN博客_torchserve

TorchServe部署Yolov5翻车录