机器学习平台方案设计

需求

支持用户、数据集、训练任务、模型、应用、平台资源的管理；

支持Web浏览器、RESTful API两类接口，具体接口功能如下：

1. 用户注册，登陆，注销；
2. 用户创建数据集；查看数据集；下载处于ready数据集；删除处于ready|fialed状态的数据集；
3. 用户创建训练任务；查看任务；删除处于finished|error状态的任务；
4. 模型和应用管理的功能，和上面类似；
5. 按租户管理资源。

架构

客户端组件：Data-Client

服务端组件：(组件部署在K8S上；组件的详细信息，参见后面详细设计)

组件名称	端口
Gateway、APP-Gateway	30001, 30002
Web-server、Data-manager、Job-manager、Model-manager、APP-manager	30003--30007
User-manager、Resource-manager	30008, 30009
Kafka、MongoDB	30010, 30011

存储采用NFS，规划如下：

对象	路径
数据集	/mlp/datasets/{user}/{dataset-name}/
脚本、初始模型	/mlp/datasets/{user}/.home/
模型	/mlp/models/{user}/{model-name}/
训练任务	/mlp/jobs/{user}/{job-name}/{ckpt, log, metrics}
组件日志	/mlp/log/{组件名称}/

详细设计

说明

- 对象（包括数据集、训练任务、AI应用）生命周期的控制基于有限状态机实现。
- 请求非法、服务器故障的Response：(403, {"msg": ""}), (503, {"msg": ""})
- request, response约定:

- 默认content-type: json;
- 默认value type: string;
- 默认value pattern: "^[a-z]{1,50}$";
- 接口规定之外的字段是不允许的;
- response默认返回对象所有字段.

Gateway

基于Nginx实现。

1. 提供反向代理服务；（根据url，将请求转发给相应的后端）
2. 提供webdav数据服务；

   /data/{user}/{dataset-name}/{dataset-filename} PUT File

   /data/{user}/{dataset-name} DELETE

Web-server

提供浏览器访问页面。基于JavaEE实现。

- 接口

/regist_login.jsp

/regist_login：返回index.jsp

/index: 返回index.jsp

/index.jsp：页面中包含username；api token；job metrics；获取用户job metrics的websocket js脚本；

/data_client.jar：数据客户端。

Data-manager

负责管理用户数据集和其他数据。

- 对象

Dataset(name, user, tenant, size, state, progress)
  name: pattern="^[a-z.]{1,50}$";
  size: type=int, 单位(byte);
  progress: type=int, value=[0-100];
  state: value={creating|ready|error|deleting}

- 接口

/api/v1/users/{user}/datasets POST Dataset: (201, Dataset)
  request schema: required=[name, size], optional=[user, tenant,]

/api/v1/users/{user}/datasets/{name}:heartbeat PUT Dataset: (200, "")
  request schema: required=[state, progress], optional=[]
  state: value={creating|ready|error}

/api/v1/users/{user}/datasets/{name} DELETE: (200, Dataset)
  当state=ready|error时，允许删除；其他状态，不允许删除。

/api/v1/users/{user}/datasets/{name} GET: (200, Dataset)

Job-manager

负责管理训练任务。对用户的训练脚本的约定：1.将输出存放到/workspace路径下；2.每轮训练完成后，上传metrics到kafka；3.训练过程中上报进度；

- 对象

Job(name, user, tenant, runtime, entry, gpu, cpu, ram, state, progress)
  runtime: value={tf};
  state: value={creating, running, finished, error, deleting};
  progress: type=int, value=[0-100];

JobMetrics(user, name, epoch, accuracy, loss, progress)

- 接口

/api/v1/users/{user}/jobs POST Job: (201, Job)
  request schema: required=[name, entry], optional=[tenant, runtime, gpu, cpu, ram]

/api/v1/users/{user}/jobs GET: (200, [Job,])

/api/v1/users/{user}/jobs/{job}:heartbeat PUT Job: (200, "")
  request schema: required=[state, progress], optional=[]
  state: value={running, finished, error};

/api/v1/users/{user}/jobs/{job} DELETE: (200, Job)

/api/v1/users/{user}/jobs/{job} GET: (200, Job)

Metrics-manager

负责管理job上传的Metrics信息，并通过Websocket将相应Metrics推送给浏览器。

> 实现说明：
>
> 内存中记录running job，由jm通过begin、finish接口进行管理。
>
> 判断job是否处于running：1.内存中找到；2.db.metrics中progress=100|-1代表已经结束。
>
> 对kafka获取的metrics的处理算法：1.running job存在，将metrics写入对应的socket和db；2.~~不存在，metrics非法，忽略。

- 接口

/api/v1/users/{user}/job_metrics/{job_id}（websocket接口）
  sever.on_message: JobMetrics

/api/v1/users/{user}/job_metrics/{job_id}:begin PUT: return (200,"")

/api/v1/users/{user}/job_metrics/{job_id}:finish PUT: return (200,"")

/api/v1/users/{user}/job_metrics/{job_id}:delete PUT: return (200,"")

Model-manager

- 对象

Model(name, user, tenant, size, src, state)
  size: type=int, 单位(byte);
  state: value={creating|ready|error|deleting}

- 接口

/api/v1/users/{user}/models POST Model: (201, Model)
request schema: required=[name, src], optional=[tenant,]

/api/v1/users/{user}/models/{model} DELETE: (200, Model)

APP-manager

负责管理AI应用，以及相应的访问入口Ingress。

- 对象

Application(name, user, tenant, gpu, cpu, ram, model_name, port, state)
state: value={creating, ready, error, deleting};
port: type=int, value=[31000-50000]

- 接口

/api/v1/users/{user}/apps POST App: (201, App)
request schema: required=[name, model_name], optional=[tenant, gpu, cpu, ram]

/api/v1/users/{user}/apps/{app} DELETE: (200, App)

/api/v1/users/{user}/apps/{app} GET: (200, App)

- 初始化：load app from db; 将非稳态的app放入queue；协程异步执行app生命周期；异步watch k8s 指定标签pod状态，异步恢复异常（state=running但收到了failed事件）的app pod；继续watch。

APP-Gateway

作为访问用户托管的AI应用的同一个入口，基于Ingress-controller实现。

User-manager

管理user、管理user-token映射关系；

- 对象

User(username, password, state)
state: value={NotActive, Actived}

- 接口

/api/v1/users POST User(): (201, User)
request schema: required=[username, password], optional=[]

/api/v1/users/{user} PUT User: (200, User) 只支持注销,不支持修改,传递的user信息被忽略.
request schema: required=[], optional=[]

/api/v1/users/{user}/token GET: (200, "{token}")

/api/v1/auth GET Authorization="Bearer {token}": (200, "{username}")

Resource-manager

负责统一管理平台的资源（GPU/CPU/RAM/ROM）

- 实现说明：

1. 使用try-confirm-cancel方案实现资源一致性。
2. 调用接口try的组件，保证try正常完成（try的request被confirm，避免泄露）。
3. try，confirm，cancel接口是幂等的（try时，request已经存在，ok。confirm/cancel时，request不存在，ok）。
4. 简化租户管理：只存在default租户；所有资源属于default租户；所有用户都属于default租户。

- 对象

Tenant(name, cpu, gpu, ram, rom, cpu_inuse, gpu_inuse, ram_inuse, rom_inuse)
  cpu: type=float, value=[0.1-10], 单位(个);
  gpu: type=int, value=[0-16], 单位(个);
  ram: type=int, value=[100-10000], 单位(Mi);
  rom: type=int, 单位(byte);

ResourceRequest(id, applier, request_type, tenant, gpu, cpu, ram, rom)
  applier: value={dm|jm|am|mm};
  request_type={request|release}

- 接口

/api/v1/resource_request:try PUT ResourceRequest: (200, "")
  request schema: required=[id, applier, request_type], optional=[tenant, gpu, cpu, ram, rom]

/api/v1/resource_request:confirm PUT ResourceRequest: (200, "")
  request schema: required=[id, applier], optional=[]

/api/v1/resource_request:cancel PUT ResourceRequest: (200, "")
  request schema: required=[id, applier], optional=[]

DB

库mlp，表user, tenant, resource_request, job, metrics, model, dataset, app_group