**Starwhale平台使用规范**

**1. 引言**

* **目的**：本规范文档旨在指导公司技术人员在公司内网中搭建和使用Starwhale MLOps平台，以便开发工程师能够对模型和数据进行高效的批量管理，主要涵盖数据集管理、模型管理和部署管理三个方面。
* **适用范围**：本规范适用于公司内所有涉及机器学习操作的技术人员。

**2. 平台概述**

* **Starwhale简介**：Starwhale是一个领先的开源MLOps平台，旨在简化机器学习全生命周期管理，包括数据集导入、模型训练、版本控制和模型部署。平台支持多种深度学习框架，并提供了完善的版本管理功能。其面向机器学习项目提供研发运营管理能力，建立标准化的模型开发、测试、部署和运营流程，连接业务团队、AI团队和运营团队。解决机器学习过程中模型迭代周期长、团队协作、人力资源浪费等问题。Starwhale提供Standalone, Server 和 Cloud 三种实例方式，满足单机环境开发，私有化集群部署和Starwhale团队托管的云服务多种部署场景，具体情况如下表格：



* **关键功能**：
  + **数据集管理**：支持导入和导出多种格式的数据集，提供数据共享、版本控制功能，并允许对每个版本进行变更追踪和比较。
  + **模型管理**：提供灵活的训练环境，记录模型的预测和评估结果，支持模型版本的跟踪、回滚和优化。
  + **部署管理**：简化模型部署流程，支持在Docker、Kubernetes等多种环境中部署，并提供详细的部署监控和日志记录功能。

**3. 参考系统要求**

* **硬件要求**：内存 32GB，CPU：8，硬盘存储空间100GB。
* **软件要求**：Python3.7 – 3.11, Linux (Ubuntu 20.04.2.0, CentOS7.9) or MacOS，conda（可选），Docker 19.03+，Docker Compose 2.0.0+，Helm 3.2.0+，Kubernetes 1.19+，Kubernetes Ingress，MySQL 8.0+，兼容S3接口的对象存储

**4. 安装与配置**

* **Starwhale Client（swcli）设置**：
  + **安装**：

python3-m venv ~/.cache/venv/starwhale source

~/.cache/venv/starwhale/bin/activate

python3 -m pip install starwhale

sudo ln -sf "$(which swcli)" /usr/local/bin/

deactivate

swcli –version查看

* + **常用命令**：

常用的swcli命令，用于与Starwhale Server交互、管理数据集、模型和运行评估作业：

swcli –version 显示swcli的版本信息。

swcli –verbose 日志中输出更多信息，当 -v 参数越多，呈现信息越多，最多支持4个 -v 参数。

swcli –help 输出命令帮助信息。

swcli instance

swcli model

swcli dataset

swcli runtime

swcli job

swcli gc

详情见[概述 | Starwhale](https://doc.starwhale.ai/zh/reference/swcli/)

* **Starwhale Server安装步骤**：
  + **步骤1：安装先决条件**：
  + **Docker**：

# 移除旧版本docker

sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

# 配置docker yum源。

sudo yum install -y yum-utils

sudo yum-config-manager \

--add-repo \

http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

# 安装 最新 docker

sudo yum install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# 启动& 开机启动docker； enable + start 二合一

systemctl enable docker --now

# 配置加速

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["填写自己的阿里云加速地址"]

}

EOF

sudo systemctl daemon-reload

sudo systemctl restart docker

* + **Kubernetes**

systemctl stop firewalld

systemctl disable firewalld

# 编辑 /etc/selinux/config 文件，修改SELINUX的值为disable

sed –I ‘s/enforcing/disabled/’ /etc/selinux/config

# 编辑分区配置文件/etc/fstab，注释掉swap分区一行

vim /etc/fstab

注释掉 /dev/mapper/centos-swap swap

# 根据每个机器的实际ip地址来配置，比如用的集群为1台master主机和2台node主机，所以编辑主机的/etc/hosts文件

192.168.201.136 k8s-master

192.168.201.138 k8s-node1

192.168.201.140 k8s-node2

# 修改linux内核采纳数，添加网桥过滤和地址转发功能编辑/etc/sysctl.d/kubernetes.conf文件，添加如下配置：

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

sysctl –system重新加载配置

# 同步时间

yum install ntpdate –y

ntpdate time.windows.com

# 安装Docker

增加一个配置：/etc/docker/daemon.json

{

"exec-opts": ["native.cgroupdriver=systemd"],

}

# 由于kubernetes的镜像在国外，速度比较慢，这里切换成国内的镜像源

编辑/etc/yum.repos.d/kubernetes.repo,添加下面的配置

[kubernetes]

name=Kubernetes

baseurl=http://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgchech=0

repo\_gpgcheck=0

gpgkey=http://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg

http://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg

# 安装kubeadm、kubelet和kubectl

yum install –y kubeadm-1.23.6 kubelet-1.23.6 kubectl-1.23.6

# 设置kubelet开机自启

systemctl enable kubelet

# 集群初始化（只需要在master节点操作）

kubeadm init \

--apiserver-advertise-address=192.168.201.136 \

--image-repository registry.aliyuncs.com/google\_containers \

--kubernetes-version=v1.23.6 \

--service-cidr=10.96.0.0/12 \

--pod-network-cidr=10.244.0.0/16

# 创建必要文件

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

# 节点加入（需要在每一个node节点操作）

kubeadm join 192.168.201.136:6443 –token () \

--discovery-token-ca-cert-hash sha256: ()

# token在上一步的输出结果里会有，如不小心清屏，可以通过kubeadm token create/kubeadm token list获取

# Hash可以用openssl x509 –pubkey –in /etc/Kubernetes/pki/ca.crt | openssl rsa –pubin –outform der 2 >/dev/null | \ openssl dgst –sha256 –hex | sed ‘s/^.\* //’获得

# 安装k8s集群网络插件（calico）

curl <https://docs.projectcalico.org/manifests/calico.yaml> -O

# 将CALICO\_IPV4POOL\_CIDR.value改成10.244.0.0/16，和初始化时一致

# 将CALICO\_IPV4POOL\_IPIP.value改成Never

# 删除镜像的docker.io前缀，避免下载速度太慢：sed –I ‘s#docker.io/##g’calico.yaml

# 记得换docker的镜像源

* + **Helm：**

参考 [Helm | 安装Helm](https://helm.sh/zh/docs/intro/install/)

* + **步骤2：安装Starwhale Server（参考[在 Kubernetes 集群中安装 Starwhale | Starwhale](https://doc.starwhale.ai/zh/server/installation/k8s-cluster)）**：

首先保证每一个集群节点都安装了Starwhale Client (swcli)

**#** 下载 Starwhale Helm chart

helm repo add starwhale https://star-whale.github.io/charts

helm repo update

# 编写values.yaml

# 设置镜像源，使用ghcr.io 镜像源: https://github.com/orgs/star-whale/packages 。

image:

registry: ghcr.io

org: star-whale

# 生产环境中依赖的外部 MySQL 服务，MySQL 版本需要大于8.0

externalMySQL:

host: 10.0.1.100 # Kubernetes 集群中可以访问的数据库IP地址或域名

port: 3306

username: "your-username"

password: "your-password"

database: starwhale # 需要提前创建数据库，名字自由指定，默认字符集即可。上面指定的数据库用户需要对这个 dataset 有读写权限

# 生产环境中依赖的外部 S3 协议的对象存储服务

externalOSS:

host: ks3-cn-beijing.ksyuncs.com # Kubernetes 集群和 Standalone 实例中同时可以访问的对象存储IP地址或域名

port: 80

accessKey: "your-ak"

secretKey: "your-sk"

defaultBuckets: starwhale # 需要提前创建Bucket，名字自由指定。上面指定的ak/sk 需要对这个 Bucket 有读写权限

region: BEIJING # 对象存储对应的 region，默认为 local

# 生产环境中指定了外部对象存储后，不需要自带的单机版本 MinIO

minio:

enabled: false

# 生产环境中指定了外部 MySQL 后，不需要自带的单机版本 MySQL

mysql:

enabled: false

controller:

containerPort: 8082

storageType: "ksyun" # 对象存储的类型 minio/s3/ksyun/baidu/tencent/aliyun

ingress:

enabled: true

ingressClassName: nginx # 与 Kubernetes 集群中 Ingress Controller 对应

host: server-domain-name # Server 外部访问的域名

path: /

# 生产环境中推荐 Starwhale Server 至少32G内存和8核CPU。

resources:

controller:

limits:

memory: 32G

cpu: 8

requests:

memory: 32G

cpu: 8

# Server 中运行评测是需要下载 Starwhale Runtime 中定义的 Python Package，推荐设置符合实际网络环境的 PYPI mirror。后续也可以在 Server System Settings页面中修改。

mirror:

pypi:

enabled: true

indexUrl: "https://mirrors.aliyun.com/pypi/simple/"

extraIndexUrl: "https://pypi.tuna.tsinghua.edu.cn/simple/"

trustedHost: "mirrors.aliyun.com pypi.tuna.tsinghua.edu.cn"

# **部署/更新 Starwhale Server**

helm upgrade --devel --install starwhale starwhale/starwhale --namespace starwhale --create-namespace -f values.custom.yaml

# 查看是否成功

kubectl get pods -n starwhale 全部running即可访问Server界面

* + **步骤3：配置详解**：

MySQL和MinIO必须提前开启，登录成功、

网络访问设置-使用Ingress：

# 添加helm仓库

helm repo add ingress-nginx <https://kubernetes.github.io/ingress-nginx>

helm repo list

Helm search repo ingress-nginx

# 下载包

helm pull ingress-nginx/ingress-nginx

tar –xf ingress-nginx-4.4.2.tgz

# 配置参数

cd ingress-nginx

修改其中的values.yaml文件

镜像地址registry: registry.cn-hangzhou.aliyuncs.com

image: google-containers/nginx-ingress-controller

kube-webhook: registry: registry.cn-hangzhou.aliyuncs.com

image: google.contrainer/kube-webhook-certgen

digest注释掉

hostNetwork: true

dnsPolicy: ClusterFirstWithHostNet

kind: DaemonSet(原为Deployment）

nodeSelector下加ingress: “true”

service.type改为ClusterIP(原为LoadBalancer)

admissionWebhooks.enabled改为false

extraArgs.update-status: “false”

enableMimalloc: “true”

# 创建命名空间

kubectl create ns ingress-nginx

# 安装ingress

kubectl label node () ingress =true

helm install ingress-nginx -n ingress-nginx

kubectl get po –n ingress-nginx查看

# 创建ingress

使用yaml文件 kubectl create –f starwhale.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: starwhale-ingress

namespace: starwhale

annotations:

kubernetes.io/ingress.class: "nginx"

spec:

rules:

- host: starwhale.wolfcode.cn

http:

paths:

- path: /

pathType: Prefix # 路径类型，Prefix 表示前缀匹配, ImplementationSpecific需要指定IngressClass，具体匹配规则以IngressClass中的规则为准，ExactL表示精确匹配，URL需要与path完全匹配上，且区分大小写

backend:

service:

name: controller

port:

number: 80

# 之后kubectl get ingress –n starwhale查看

**5. 数据集管理**

* Starwhale Dataset 包含数据构建、数据加载和数据可视化三个核心阶段，是一款面向ML/DL领域的数据管理工具。Starwhale Dataset 能直接使用 Starwhale Runtime 构建的环境，能被 Starwhale Model 和 Starwhale Evaluation 无缝集成，是 Starwhale MLOps 工具链的重要组成部分。
* 详情参考数据集脚本模板以及[Starwhale 数据集 | Starwhale](https://doc.starwhale.ai/zh/dataset/)

**6. 模型管理**

* **创建 Starwhale 模型有两种方法：通过 [swcli](https://doc.starwhale.ai/zh/reference/swcli/model) 或通过**[**SDK**](https://doc.starwhale.ai/zh/reference/sdk/model)。详情参考[Starwhale 模型 | Starwhale](https://doc.starwhale.ai/zh/model/)

**7. 安全与合规**

* **用户角色与权限管理**：角色用于为用户分配权限。只有Starwhale Server/Cloud有角色和权限，Starwhale Standalone没有相应概念。系统会自动创建一个管理员角色并分配给默认用户“starwhale”。一些敏感操作只能由具有管理员角色的用户执行，例如在Starwhale Server中创建新的账号。
* 每个项目具有三类角色：
* 管理员Admin - 项目管理员可以读写项目数据并将项目角色分配给用户。
* 维护者Maintainer - 项目维护者可以读写项目数据。
* 访客Guest - 项目访客只能读取项目数据。
* 创建项目的用户成为第一个项目管理员。他可以在这之后将角色分配给其他用户。
* **8. 维护与更新**
* **资源版本控制**：
* Starwhale管理所有模型、数据集和运行时的历史记录。对特定资源的每次更新都会附加一个新版本的历史记录。
* 版本由version id标识。version id是由 Starwhale自动生成的随机字符串，并按其创建时间排序。
* 版本可以有标签。Starwhale使用版本标签来提供人性化的版本表示。默认情况下，Starwhale会为每个版本附加一个默认标签。默认标记是字母“v”后跟一个数字。对于每个版本化的资源，第一个版本标签始终标记为“v0”，第二个版本标记为“v1”，依此类推。有一个特殊的标签“latest”总是指向最新的版本。删除版本后，将不会重复使用其默认标签。例如，有一个带有标签“v0、v1、v2”的模型。 删除“v2”后，标签将为“v0、v1”。 接下来一个标签将是“v3”而不是“v2”。您可以将自己定义的标签附加到任何版本并随时删除它们。
* Starwhale使用线性历史，不提供分支。
* Starwhale资源无法真正回滚。当要恢复某个历史版本时，Starwhale会复制该版本数据并将其作为新版本追加到历史记录的末尾。您可以手动删除和恢复历史版本备份、软件更新和系统健康监控。

。

**9. 故障排查**

* 当镜像下载速度慢，尝试换源，yum源，docker源都需要换。
* Starwhale国内镜像库已经失效，故yaml文件里的registry都要改为ghcr.io

**10. 附录**

* **术语表**：
* **Starwhale Dataset**：Starwhale 对机器学习领域数据集的一种抽象，实现数据集的构建、分发、加载、版本控制和可视化展示，满足模型训练、评测等过程中对数据集的需求。
* **Starwhale Model**：Starwhale 定义的一种机器学习中模型的标准包格式，包括模型权重文件、代码和配置等，满足模型评测、模型微调等环节中模型包的构建、分发、版本控制和运行等需求。
* **Starwhale Runtime**：Starwhale 对机器学习领域程序运行环境的一种抽象，屏蔽Dockerfile编写和CUDA安装等细节，实现一种可复现、可分享的Python运行环境。
* **Starwhale Instance**：Starwhale的每个部署称为一个实例。所有实例都可以通过swcli进行管理。Starwhale 实例有Starwhale Standalone, Starwhale Server 和 Starwhale Cloud 三种类型。Starwhale 会在不同实例上保持概念上的一致性，用户可以轻松的在不同实例上复制模型、数据集和运行时。
* **Starwhale Standalone**：Starwhale 三种实例类型之一。面向独立开发者，部署在本地开发环境中，通过swcli命令行工具进行管理，满足开发、调试等需求。
* **Starwhale Server**：Starwhale 三种实例类型之一。面向团队用户，部署在私有数据中心里，依赖Kubernetes集群，提供集中化、Web交互式的、安全的服务。
* **Starwhale Cloud**：Starwhale 三种实例类型之一。托管在公有云上的服务，访问地址为 [https://cloud.starwhale.cn](https://cloud.starwhale.cn/)，由Starwhale团队负责运维，无需安装，开箱即用。
* **swcli**：是Python编写的Starwhale命令行工具，可以对不同实例上的模型包、数据集和运行时进行管理。
* **datastore**：Starwhale 中的一个基础设施，提供类似Big Table的存储和访问方式，满足数据集和评测数据的存储、检索等需求。
* **Starwhale Project**：是组织不同资源（如模型、数据集等）的基本单位。
* **.swignore 文件**：与.gitignore和.dockerignore等文件类似，用来定义忽略某些文件或文件夹，Starwhale 模型构建过程会尝试读取该文件，并决定哪些文件会被忽略掉。
* **model.yaml 文件**：是一种定义Starwhale Model如何构建的描述性文件，非必需。
* **dataset.yaml 文件**：是一种定义Starwhale Dataset如何构建的描述性文件，需要与一些Python脚本配合使用。swcli dataset build 命令会使用。非必需。
* **runtime.yaml 文件**：是一种定义Starwhale Runtime的描述性文件，swcli runtime build 命令会使用。非必需。
* **Starwhale Console**： Starwhale Server/Cloud 实例中的Web前端页面。
* **参考文献**：[什么是Starwhale | Starwhale](https://doc.starwhale.ai/zh/)