# 产品介绍

### 深度学习系统

Ti深度学习系统，是专门为深度学习算法工程师打造的一款定制服务的系统，主要目的是优化训练资源的管理，把算法研究员和开发员从繁琐的资源管理和环境管理中解脱出来，让他们能聚焦于算法和模型本身的设计和实现。同时，提供完整的深度学习方案，让不熟悉深度学习算法的用户可以快速方便的使用系统的功能，快速输出深度学习的结果。

在传统的深度学习过程中，用户需要管理一切细节，包括数据的存储，数据的处理，分布式系统的搭建，docker镜像管理，模型输出/评估/发布等一切细节。这种过程耗时长，繁重并且比较容易出错，算法开发使用人员不能聚焦于算法和模型本身的操作，并且模型训练出来后不能及时的发布应用。

Ti系统帮助算法人员解决了这些问题，把数据管理和训练管理以及推理的一整套流程管理起来，让算法人员可以聚焦于算法和模型上。同时，普通用户可以通过Ti系统提供的前端界面，很方便的完成一整套训练过程并使用训练模型应用于生产。通过整合深度训练和发布的流程，可以节省80%的训练周期，节约50%以上的人力成本。

### 功能

* 用户训练镜像的维护；
* 训练数据接入；
* 用户工作台（调试平台）；
* 支持单机训练任务；
* 支持多机训练任务；

### 特点

* 容器编排技术：ATOM 是构建在容器编排技术和分布式存储之上, 拥有管理和对接其他云存储服务的能力;
* 资源调度策略：有效管理 GPU 资源, 通过合理划分和调度, 将调试代码与模型训练分开进行, 充分利用 GPU 资源, 减少 GPU 闲置时间;
* 资源可扩展：实现了存储和计算分离, 使得各自可以独立平行扩展。

# 基础知识介绍

为了更好的使用Ti，或者说更好的使用Ti进行深度学习训练、推理，必须了解相关的基础概念。

用户手册中包含有关K8S，Docker等相关概念，为了让读者更好的理解用户手册的内容，本章会着重对这些名词进行解释。后文中涉及的存储，配额，镜像和工作台的概念，需要用户在使用训练功能之前对这些内容做好了解的基础知识。

如果你想对训练平台整理有更加深入的了解或者你是单位的管理员，你也需要对训练排队规则，集群资源统计面板简介，与Ti相关的K8S概念和与Ti相关的Docker概念进行了解。

# 存储

### 存储的分类

| 存储类型 | 使用成本 | 性能 | 稳定性 | 是否受K8s管制 | 主要用途 |
| --- | --- | --- | --- | --- | --- |
| 块存储 | 高 | 高 | 强 | 不受 | 存放训练数据 |
| 文件系统 | 低 | 低 | 弱 | 不受 | 存放代码 |
| 本地存储 | 最高 | 最高 | 高 | 不受 | 存放训练数据（训练时） |
| 外部存储 | 低 | 低 | 中 | 不受 | 存放训练数据（长期存放） |
| 临时存储 | 低 | 低 | 弱 | 受 | 通常不使用 |

以上标中是每种存储的特点，后面会对每种类型的存储展开说明； ### 存储的性质

在使用Ti的时候，说到存储，我们通常会考虑以下几个方面：

1.存储的数据：不同的存储有着不同的挂载模式，块存储可以同时只读挂载到多个工作台或训练，但一旦由一个工作台或训练有写挂载，降无法继续挂载块存储。如果已经只读挂载了，也不能再写挂载。FileSystem存储可以在多个工作台或训练只读挂载。基于以上存储的性质，算法用户通常使用filesystem存储训练代码，使用块存储存储训练数据，块存储只能进行单独写和多只读的挂载方式。

2.存储可使用的容量：由于深度学习需要的数据非常庞大，而且经常会对数据进行加工操作。往往每个使用者都需要大量的容量来存放数据并进行操作如剪辑、清洗等。根据经验，每个训练系统的用户往往需要1T以上的容量供日常训练使用。

3.存储的访问性能：存储访问性能分为读和写。一般在训练过程中主要为读操作，在整理数据时为写。大部分情况下，使用者更关心读的性能。按照目前存储的形态，速度从高到低依次为：本地NVMe或SSD盘，本地分布式SSD块集群，本地分布式文件系统，云端对象存储（内网），云端对象存储（外网）。而这些存储形态提供的容量刚好是从小到大的顺序。所以，Ti系统会根据不同存储形态提供响应的缓存功能，无缝的把数据从慢速存储迁移到快速存储。

4.存储的访问方式：按目前通用的访问方式，分为POSIX文件访问和对象存储访问。POSIX文件访问方式又分为本地文件系统，分布式块系统（上面格式化为文件系统），分布式文件系统3中方式。对象存储一般指标准的S3语义访问。

5.存储的访问控制：存储访问控制分为读写访问和按角色的权限控制。在Ti系统中存储的数据往往有一定的机密性和敏感性，比如鉴黄，暴恐图片等，这部分数据要严格控制让不同的使用者使用。另外，为了便于管理，保证数据安全，对部分公开数据提供只读形式，避免误删除操作。使用者在Ti系统上还可以把自己的存储共享给其他人使用。

# 配额

和传统使用单机训练的方式不同，在Ti平台中，所有的资源都是共享的。系统总体资源有限，不能无限制分配给一个人或部分人使用。这种控制通过Quota来实现。目前，Ti支持如下类型Quota，使用者需注意：

1.存储资源：用户可以创建一块或多块块设备的盘，但是，总的容量不能超过设置的配额数目，否则创建存储会失败。创建成功后调低Quota，不会影响已创建的设备的使用。

2.计算资源：工作台和训练总卡数限制，如果用户创建的任务超过这个限制，则任务不能创建成功。如果任务已经创建成功但下调了任务GPU卡数Quota，工作台不受影响。新起的工作台重新受到quota限制。

3.设置配额：管理员可以手动调整用户的quota限制，用户只能查看自己的quota限制。新用户加入训练平台后，默认是没有配额限制的，管理员需要在新用户加入训练平台之后，修改新用户的配额。

# 镜像

工作台和训练都需要镜像。Ti里面镜像是和gitlab授权绑定在一起的。镜像分3个类型：

1.用户镜像：用户自己创建的镜像，存放在自己的gitlab仓库中，默认只有自己可见。Ti会代替用户在调度的物理机上做好登录授权。

2.共有镜像：Ti平台提供的公开镜像，部分镜像包含Pytorch和CUDA、CUDNN库，可能不是最新版本。Ti会代替用户在调度的物理机上做好登录授权。

3.镜像地址：直接输入镜像地址，可以是任意镜像。但是，Ti不负责授权，当工作台或训练调度到物理机后，由于物理机没有登录授权可能导致拉镜像失败。一般建议使用公开镜像或用户自己的镜像。

# 工作台

在Kubernetes里面，工作台和训练都是一个POD。这些POD会挂载存储资源，关联CPU、内存和GPU资源，是使用者进行深度训练的地方。但是，工作台和训练有较大的区别。

工作台是给算法工程师调试代码、整理数据、验证环境使用的。一般来说，工作台不会共享，而且会长期使用。这也意味着，工作台占用的CPU、GPU、内存不会释放，也无法共享。

使用工作台需要注意的是：

1. 工作台启动受到Quota限制，缺省为3 GPU卡（其他资源可暂时不关注）
2. 工作台可以挂载块、文件、local-storage和对象存储。目前，对象存储缺省是按照只读挂载的（当然可以配置)，其他存储可选读写关在或只读挂载。
3. 工作台可以起CPU或GPU工作台，并指定CPU核数或GPU卡数。目前只能按整张GPU卡纬度挂载。
4. 工作台可以写入系统任何目录，但非Ti提供的存储（块存储，文件系统存储，对象存储，local-storage存储），其他地方都是临时存储，使用量超标会被K8S驱逐，慎用。
5. 工作台可以多次登录，退出。重新启动后使用原配置，但可能调度到其他机器
6. 工作台的缺省启动脚本是【 tail -f /dev/null】，请不要杀掉该进程，否则工作台会立即推出
7. 工作台重新启动后，块存储，文件系统存储，对象存储数据和退出时保持一致。local-storage不保证一致，因为可能调度到其他机器。即使调度到同一台机器，可能被其他人调度过，数据不保证
8. 工作台一般会启动jupyter和SSH服务，不用指定启动脚本（缺省是tail -f /dev/null）
9. 可以在portal上终止工作台运行

# 训练排队

### 训练排队规则简介

背景：由于Ti用户数量不断增加，集群资源不足的问题时常发生，为了更公平、科学的分配训练资源，Ti v2.1.0版本新增了训练任务的排队功能。

1.排队目前是对特定机型进行的排队，排队的先后顺序的判断依据：管理员置顶>分数>用户拖拽优先级>创建时间，意思是管理员指定的任务会最早被调度上；

2.没有置顶的任务通过用户的排队分数进行判断优先级，用户近24小时内对这种资源使用越少的人分数越低，排队的优先级越高，排队时会插队排在其他用户的前面；

3.用户也可以通过在训练队列页面进行任务的拖拽，决定个人同种资源的任务哪个会被先调度上；

4.如果个人用户没有拖拽任务，系统对个人用户任务同种资源的调度顺序则会按照创建时间进行排序，在排队时同种机型的任务会发生阻塞，不同机型的任务不会发生阻塞。单机和分布式任务在两条队列中，两种任务之间不会进行排序；

### 按照优先级调度任务

区分任务的优先级, 利用调度器插件实现优先调度

调度优先顺序

1.使用不同型号 GPU 的任务无优先级之分

2.置顶的任务比非置顶任务先调度

3.后置顶的任务比先置顶的任务先调度

4.不同用户的任务比较用户资源分数, 分数低的先调度

5.同一用户任务按自定义排序调度

6.同一用户未排序任务之间无优先级之分

调度时依下述策略确定调度顺序:

if not 两个 Pod 都由 AtJob 创建
 使用默认比较策略

if 两个 Pod 属于同一个 AtJob
 使用默认比较策略

if not 两个 Pod 使用同型号 GPU
 使用默认比较策略

if 至少一个 Pod 被管理员置顶
 if 只有一个 Pod 被置顶
 先调度被置顶的 Pod
 else
 先调度后置顶的 Pod

if 两个 Pod 由同一个用户创建
 if 用户自定义排序不同
 按用户自定义排序调度
 else
 使用默认比较策略

else
 先调度分数低的用户创建的 Pod

其中, 默认比较策略参考 Coscheduling 插件:

if Pod priority 不同
 按 priority 排序

if 属于不同 Pod Group
 先调度属于更早创建的 PodGroup 的

else
 先调度创建更早的 Pod

### 调度器框架延伸点 QueueSort

* 依优先顺序对调度队列中的任务排序
* 同一优先级的任务, 适用默认排序策略；
1. 两个 pod 都不属于 podGroup, 按创建时间排序；
2. 两个 pod 中至少一个属于某 podGroup, 按 podGroup 创建时间排序；
3. 属于同一个 podGroup 的两个 pod, 按创建时间排序；

### 调度器框架延伸点 PreFilter (调度前)

1. 有优先级更高的任务 pending, 则拒绝 (重新排队), 避免争夺资源
2. 否则继续

### 调度器框架延伸点 Permit (调度后, Bind 前)

1. 有优先级更高的任务 pending, 则拒绝, double checking
2. 否则继续

### Ti 资源使用优先级评分

如何计算用户资源占用分数

### 资源占用评分目标

用户使用资源的优先顺序由该用户近期占用资源量的综合评分决定, 该评分:

1. 用得越多, 分数越高, 优先级越低
2. 既要考虑当前用量, 也要考虑历史用量
3. 近期用量比远期用量对优先级的影响更大
4. 不同类型资源分别计算分数和优先级

### 评分计算方式

针对上述的资源占用评分目标第 2 条, 我们期望在任意时刻 $t$, 评分 $s\left(t\right)$ 不止反应当前资源用量 $g\left(t\right)$ 的值, 还能受 $τ<t$ 期间 $g\left(τ\right)$ 取值的影响. 将 $g\left(τ\right)$ 取值对 $s\left(t\right)$ 的影响, 即系数, 记为 $f\left(t\right)$, 则评分是资源用量与系数的卷积:

$$
s(t) = \int\_{0}^{+\infin} {f(\tau) g(t - \tau) \text{d} \tau} \\
$$

又根据[评分设计要求](#资源占用评分目标)第 3 条, $f\left(t\right)$ 应该满足:

* $0\leq f\left(t\right)\leq 1$;
* $f\left(0\right)=C$, $C$ 为一正常数;
* $f(+\infin) = 0$;
* 当 $g\left(t\right)$ 为常函数 $G$ 时, $\int\_{0}^{+\infin}{G f(t) \text{d}t} = G$;
* 严格单调;
* 同时, $f\left(t\right)$ 变化的速率应可调节;

综合上述要求, 选择方便计算的指数函数:

$$f\left(t\right)=Ce^{−\frac{t}{T}}$$

将卷积过程转换为差分形式:

$$
\begin{aligned}
s(t)
= & \sum\_{k=0}^{+\infin} {f(k \Delta t) g(t - k \Delta t)} \\
= & C \sum\_{k=0}^{+\infin} {e^{-\frac{k\Delta t}{T}} g(t - k \Delta t)}
\end{aligned}
$$

又由于系数和应满足:

$$
C \sum\_{k=0}^{+\infin} {e^{-\frac{k\Delta t}{T}}} = \frac{C}{1 - e^{- \frac{\Delta t}{T}}} = 1
$$

故 $C=1−e^{−\frac{Δt}{T}}$, 所以

$$\begin{matrix}s\left(t\right)=&\left(1−e^{−\frac{Δt}{T}}\right)\left(g\left(t\right)+e^{−\frac{Δt}{T}}g\left(t−Δt\right)+e^{−\frac{2Δt}{T}}g\left(t−2Δt\right)+e^{−\frac{3Δt}{T}}g\left(t−3Δt\right)+…\right)\\=&\left(1−e^{−\frac{Δt}{T}}\right)g\left(t\right)+e^{−\frac{Δt}{T}}\left(1−e^{−\frac{Δt}{T}}\right)\left(g\left(t−Δt\right)+e^{−\frac{Δt}{T}}g\left(t−2Δt\right)+e^{−\frac{2Δt}{T}}g\left(t−3Δt\right)…\right)\\=&\left(1−e^{−\frac{Δt}{T}}\right)g\left(t\right)+e^{−\frac{Δt}{T}}s\left(t−Δt\right)\end{matrix}$$

上式即为 $s\left(t\right)$ 的递推公式. 其中 $Δt$ 即为采样周期; $T$ 控制评分趋近用量的速率, 称为调节时间; 经过 $4T$ 后, 评分与用量的差小于 2%.

根据上式, 令 $T=10$, $Δt=1$, $g\left(t\right)$ 模拟实际生产中 gpu 用量的变化, 则评分随资源用量变化趋势类似下图. 可看到在每次 $g\left(t\right)$ 发生变动后, 无论变动幅度大小, 经 $T$ 后, $s\left(t\right)$ 与 $g\left(t\right)$ 的误差 (追踪误差) 约为 37%; 经 $2T$ 后, 追踪误差约为 14%; 经 $4T$ 后, 追踪误差约为 2%.



scoring

### 更新评分的策略

1. 调节时间和采样周期可配置, 暂定 $4T=7d$, $Δt=1h$
2. 记录每个用户当前资源占用量 $g$ 和当前分数 $s$, 按 GPU 类型分别计算
3. 以 $Δt$ 为周期, 以递推公式更新所有用户的分数
4. operator 重启时, 视情况依递推公式更新分数, 此时 $Δt$ 为与上次更新时间的差
5. operator 第一次运行时, 或对于有任务运行却无记录的新用户
	* 对于每一个用户
	* 对每一个在运行的任务, 根据其用量, 以任务已运行时间为 $Δt$, 计算单个任务在评分中的分量
	* 将属于同一种资源的评分分量累加, 即为该资源最终评分 (这里依据了该评分过程的可加性, 未证明, 有测试)

# 集群资源统计面板简介

Ti的集群资源数量有限，不同训练类型计算资源的分配是由管理员根据集群资源的使用情况进行设定，因此管理员需要关注集群资源的使用情况，将使用不饱和的资源池中的机器切换到使用较为饱和的资源池中；用户创建训练后，如果当前资源池中无空闲资源时，该任务会自动进入等候队列进行排队，当有资源空闲时，系统会自动调度运行，无需人员看护。

算法工程师用户也可以通过查看训练资源面板了解集群资源的使用情况和排队情况，根据这些资源的数据，算法工程师在创建任务时就能选择更早能被调度上的计算资源，可以有效避免训练严重排队的情况；

如下图红框所示，用户可以在「训练列表」查看当前工作台、单机训练和多机训练的资源统计，监控数据可能会有延迟，可以刷新查看最新数据： 运行中：该用户创建的处于“运行中”的任务数量；
队列中：该用户创建的处于“队列中”的任务数量；
占用：该用户创建的处于“运行中”的任务所使用的计算资源总量；
总量：该类型任务总共可使用的计算资源总量；
缺口：该用户创建的处于“队列中”的任务所占用的计算资源总量；



image

如下图红框所示，点击切换可查看集群视角的统计数据：
占用：该用户创建的处于“运行中”的任务所使用的计算资源总量（pod数量）；
总量：该类型任务总共可使用的计算资源总量；
缺口：该用户创建的处于“队列中”的任务所占用的计算资源总量；



image

# 与Ti密切相关的K8S概念

由于Ti平台是运行在Docker容器中，而Dock而容器又是通过Kubernetes编排的，所以，为了更好使用Ti进行训练，必须了解相关的docker和Kubernetes概念。Docker相关概念在下一节赘述。

和Ti密切相关的Kubernetes概念如下：

### Pod

Pod是Kubernetes调度的单位。Ti的训练和工作台也是一个Pod。只不过，Ti的使用者一般接触的是POD里面的一个Container。多Container的设计和Container的类型稍后讲述。每个POD都有关联的资源，比如CPU，GPU，存储等，但是，更多时候影响Ti使用的是Pod的状态。一个POD主要包含如下的状态，虽然这些状态对Ti用户都不可见。

### Pending

当用户创建好Ti工作台或者训练，并点击【启动】的时候，工作台或训练对应的POD处于Pending状态。这个状态是Kubernetes等待资源做调度。一旦Kubernetes找到满足所有POD申请资源的机器并把POD绑定到机器上，Pending状态结束，进入Creating状态。需要注意的是，Pending只是针对单个POD的，POD的pending状态不等同于Ti的训练或工作台的Pending状态。如果Ti的训练是一个分布式的包含多个POD，那么可能其中一个POD处于Pending状态，另外一个POD已经绑定资源，处于Creating或其他状态。

### Creating

Pod从Pending状态结束后，进入Creating状态。这个阶段，Kubernetes为POD绑定资源，包括：挂载存储，拉镜像等。由于Kubernetes已经判断GPU、CPU、内存等资源已经满足了该POD的spec，可以认为，这几类资源已经【瞬间】挂载到POD了。而镜像和存储资源的挂载是比较复杂和漫长的过程。对于镜像，取决于镜像hub的网络通信和镜像大小，特别是第一次拉取镜像的时候，时间可能会比较久，甚至可能长达10分钟。后续镜像如果已经在物理机上存在，则拉取镜像时间非常断。挂载存储更复杂。对于本地host存储，挂载可以认为是瞬间完成的。而对于分布式块、分布式文件系统，对象存储等这类网络存储，Kubernetes都是通过网络插件挂载，中间还经过各类验证，挂载时间比较长。如果任何一个存储挂载出错，还需要等待重试。大部分工作台和训练的启动等待时间都在这个阶段。

### ImagePullError

如果镜像不存在，或者由于鉴权或其他原因导致拉取镜像失败，则POD处于这个状态。把镜像源或鉴权问题解决后，可以重新启动工作台或训练。

### Error

上面的状态都是工作台或训练启动之前。当工作台或训练启动后，POD又会有几个状态。如果POD的主container入口程序异常退出，或由于被管理员关闭，或被Kubernetes关闭，POD处于一个Error状态。只能通过再次重启工作台或训练解决。

### OOMKilled

使用3中类型的资源需要特别小心：CPU，内存，临时存储（ephemeral storage），特别是后2种。由于CPU具有一定伸缩性，一般不会引起容器被杀。而如果使用内存资源超了，POD会被CGROUP杀掉。特别注意的是，cgroup把page cache也算成是一种内存消耗，所以，当在POD里面使用IO密集型的应用时需要特别小心。

### Evicted

Evicted其实也是一种使用资源超标别Killed。OOMKilled是使用内存超标被杀，而Evicted是使用ephemeral-storage被杀。ephemeral storage是物理机上存放Kubernetes相关log，配置等的磁盘分区。挂载进去的POD的临时目录，根目录等也都是属于ephemeral 空间。Kubernetes需要保留部分空间供自己使用和给其他系统程序使用。一般使用超过90%，就会挑选POD驱逐。被驱逐的POD处于Evicted状态。

### Terminating

POD被用户从工作台关闭后，或用Kubernetes命令关闭后，处于Terminating状态。这个时候，POD会卸载存储，如果长时间卡在这个状态，大部分时候是存储卸载失败重试导致。

### Completed

正常结束的POD处于Completed状态，资源会全部释放。

### Container

Container是虚拟化的基本概念。Docker的管理对象也是Container。但是Kubernetes管理或调度的基本对象是POD。每个POD有若干个Container，共享存储和其他资源。一般来讲，主要分为Init和Maincontainer。两个顺序启动。Init container首先启动，做一些初始化工作。如果Init运行失败，则整个POD失败。如果Init运行成功，接着运行main container。

### Volume (storage)

Kubernetes支持多种存储。Ti使用如下类型存储。注意，有些存储使用是显示的，有些则是隐式的，特别需要注意。 块存储（Block）:块存储是分布式块设备。可以有很多驱动提供块存储的挂载。比如CEPH可以通过CSI插件挂载。公有云上云盘也可以由其提供的驱动插件挂载。块存储必须在创建工作台和训练的时候指定，并且一个块存储写挂载不能共享。意思是，如果一个块存储已经写挂载了， 其他工作台和训练则不能再挂载该块存储。如果该块存储已经读挂载了，那么其他工作台或训练只能读挂载。每个Ti用户的块存储可以在Ti平台设置Quota。 块存储是Ti平台显示提供的。

### 文件存储（FileSystem）

文件存储指分布式文件存储NAS。提供统一全局名字空间的文件系统访问。可以在多个训练或工作台读写挂载。文件存储可由Ti平台提供，也可由公有云厂商通过Kubernetes CSI插件提供。 文件存储是Ti平台显示提供的。

### 外部存储（bucket）

外部存储指云厂商提供的对象存储。特点是海量，便宜，访问速度慢，不提供文件系统接口。Ti平台对外部存储有权限控制，目前都是提供只读访问方式。对外部存储访问，可以有两种方式，一种是通过工具上传下载到本地，然后进行读写， 另外一种是通过虚拟化，提供类似于POSIX标准的文件系统访问接口，可以像正常文件一样读写，并且通过CACHE方式，提供缓存加速功能。目前两者Ti平台都提供。外部存储也是通过CSI插件挂载到Container里面访问的。 外部存储是Ti平台显示提供的。

### 数据集

数据集其实是上述集中类型存储的一个快照。它是Ti平台基于元数据管理的一种方式。数据集的数据是由上述几种存储的某一种提供的，然后数据集把相关联的数据的元数据信息集中在一起，并且通过Kubernetes的CSI插件形式，把数据集挂载到工作台或者训练 容器中，像普通存储一样使用。目前，提供只读方式访问。数据集的元数据都存放本地，访问速度非常块。如果支撑的存储数据访问提供缓存加速，则数据集也提供缓存加速。 数据集是Ti平台显示提供的。

### 本地存储（Local-Storage）

本地存储是由hostpath方式挂载到工作台或训练中的本地机器文件目录。它不受到cgroup控制，用户可自由使用。但是，它是临时存储。当工作台或训练结束侯，这部分存储被系统删除，Ti不保证再次启动工作台或训练能访问到这部分存储。 本地存储是Ti平台显示提供的。

### 临时存储（Ephemeral-Storage）

临时存储其实是Docker的layer文件系统。每个训练或工作台中除了Ti平台提供外的存储都是layer存储。它建立在容器所调度的物理机上的本地文件系统。临时存储受到Kubernetes管制，当使用临时存储超过一定限额时，会被Kubernetes驱逐。 缺省情况下，使用临时存储超过本地物理机挂载点文件系统容量的90%，则POD会被驱逐。 临时存储不是Ti平台显示提供的。使用的时候需要谨慎。 共享内存：共享内存是由RAM提供的特殊的临时存储。受到Kubernetes和Cgroup管制。使用超限会被驱逐。 共享内存不是Ti平台显示提供的。

### cgroup

Cgroup是Linux内核控制进程使用各类资源的手段。Docker和Kubernetes也是利用Cgroup来监控和控制POD对各种资源的使用。一旦Pod使用资源超过指定上限，会被CGroup杀掉或被Kubernetes驱逐。CGROUP主要监控CPU和内存，且以单机为工作单位。

### CPU

每个POD都可以指定CPU的limit和request值。如果不指定，POD可以使用系统所有的CPU资源。一般情况下，GPU使用超限不会导致POD被killed或者Evicted。

### Memory

Memory是一个非常重要的资源。和CPU不同，内存不能弹性伸缩。如果CPU不够，顶多多调度几次，运行时间长一些。而内存不够，程序将无法运行。所以，内存检查是非常严格的。一旦使用内存超限，POD会立即被CGROUP杀死。特别注意的是，很多page cache也会被算到POD头上，虽然page cache是由kernel进行管理。大量使用IO的程序特别需要注意。目前尚无办法完全解决该问题，只能case by case处理。

### ephemeral storage

类似memory，这个不是由cgroup管理，而由Kubernetes管理。结果类似，使用ephemeral storage超限，POD会被Kubernetes Evicted。规避办法是尽量不要使用非Ti提供的存储。

### Network

Kubernetes提供自己的网络体系。包括DNS，网络路由，内网IP等。所有内网IP是互通的，可直接访问。Kubernetes还提供hostnetwork，但是一般Ti不打开该选项。

# 与Ti相关的Docker概念

Kubernetes支持多种容器虚拟化技术，应用最广的是Docker。所以，除了需要了解Kubernetes外，对Docker一些基础技术也需要一些了解。

### namespace

Docker本身是一个轻量级容器技术，通过共享内核，使用namespace隔离的方式，实现用户进程的虚拟化。无论是前期的虚拟机还是现在的轻量级Docker，都需要对用户进行一定的隔离。这种隔离是Dokcer运行的基础。目前，Docker利用了Linux多个namespace进行隔离，包括：mnt, net, pid, ipc, uts等。具体细节这里不展开。在实际使用中一般不感受到namespace，用户感觉自己独占一台机器在运行。

### cgroup

cgroup是一个非常重要的概念，从裸机过渡到使用Docker很多【无法解释】的问题都和Cgroup相关。cgroup全称是control group，是Linux用来控制各进程以避免侵占所有的系统资源，保证系统稳定运行。在使用Kubernetes和Docker之前，每台机器的资源是供给使用者独享的，程序基本不受到内存和CPU等控制。但是，在Docker环境，特别是在Kubernetes编排下，所有的容器都受到cgroup控制（Kubernetes通过cadvisor实现），如果使用资源超标，则容器直接被销毁或驱逐。cgroup一般控制内存，CPU，临时存储（这个只在Kubernetes中存在）。使用Ti的时候要特别注意内存占用，超过工作台或训练套餐内存limt会导致训练异常终止。

### Docker file

Docker File是创建docker镜像的文件。正规做法是每个镜像都需要通过docker File创建，以便于跟踪。

### Image/hub

镜像是启动训练需要的【操作系统的二进制文件】。每个docker启动都需要镜像。

### Container

docker隔离的单位是container。一个container里面可能有多个进程，从用户角度来看，container就是一个操作系统。和Kubernetes不同的是，Kubernetes管理的对象是POD，一个POD可以有多个container。

### Privileged

默认docker启动的镜像是不带privileged权限的，除非用–privileged指定。有了privileged权限，相当于拥有了物理机管理权限。 Layer file system (union file system)：Docker的文件系统是一个layering系统。简单来说，有2层。底层系统是只读的，如果顶层有了相同名字，则覆盖了底层的名字。当改写了底层文件后，会在顶层创建同样的文件，这样就不会直接修改底层的文件。这样的好处在于， 宿主机上的文件实际可以被多个docker容器共享，docker里面修改了文件不会对宿主机文件造成破坏。

### network

Docker有自己的网络系统，也可以使用宿主机的hostnetwork。Kubernetes有自己的network系统，在Ti平台中，主要使用Kubernetes管理的网络系统。

### volume

docker可以通过volume共享存储，一个volume可以多次挂载到不同的容器中。Ti一般不使用Docker的volume。

# 平台管理员-使用指南

为了管理Ti上的用户可以规范、合理的使用计算资源，Ti的系统管理和队列管理功能，可以让管理员通过设置系统配置或者调整排队顺序，实现系统资源的合理分配；

管理人员可以通过下文介绍的系统管理和队列管理让单位的资源更加平均的分配给用户，避免用户对资源的长期独占，用户分配不到资源，以及资源的闲置导致的浪费等情况的发生；

在完成系统完成部署，管理人员在初次使用TrainSpot训练平台时，需要对系统配置并将单位的所有算法用户添加功能权限，单位中的算法用户才能使用系统中的资源运行训练任务；

image

image

下面的会介绍管理员用户在使用训练平台管理功能的完整操作流程：
步骤1：“[套餐管理](/train-spot/instruction/quota_setting)”中创建算法用户可以选的计算资源。
步骤2：“[配额管理](/train-spot/instruction/quota_setting)”中创建用户可以用的最大资源总量。
步骤3：“用户组管理”中创建用户组并设置该用户组的功能权限，并可以批量将用户加入到该用户组中，这样这批用户就被批量赋予了该用户组的功能权限。也可以在“用户管理”中对单个用户的用户组进行设置，在后续迭代中用户权限将移至用户中心，这里不在多做赘述。
步骤4:“[队列管理](/train-spot/instruction/queue_setting)”中，管理员可以查看所有用户创建的训练任务和工作台，并对队列中的单机和多机训练任务进行置顶操作，系统会优先调度被置顶的任务；

# 平台用户-使用指南

上手使用深度学习平台进行训练，需要对整体的训练的操作流程有所了解；并准备好训练所需的数据集、镜像，完成训练中配置和执行入口等内容的设置，才能保证训练的正常进行；

按照以下步骤进行操作，新手小白也可以成功创建和运行自己的训练任务； image

## 1.使用前准备

为了避免创建的任务启动失败或者运行出错，算法用户需要了解系统中密钥、smmc、内部存储、镜像的使用和管理方法，包含使用前需要对存放数据的：
- [内部存储](/train-spot/instruction/storage)的配置 ，工作台鉴权的密钥配置和用户镜像的上传管理方式；

* [密钥管理](/train-spot/instruction/user_info)是有关工作台鉴权的重要配置，如果用户在使用工作台前没有配置启动创建好的工作台会出现报错的情况；
* smmc工具是Ti平台的数据上传下载工具，主要针对支持了S3接口并采用MINIO实现对象存储的服务。

## 2.使用前确认

为了让用户清楚创建训练任务和存储可用的资源余量，避免出现超过配额，创建失败的情况，以及任务可以被更快的调度上；因此在创建并启动训练前，用户需要确认两件事情：

* [确认资源](/train-spot/instruction/check_resource)：集群中有资源可以被使用在该类型训练任务上，资源数足够或者排队的人数较少，以确保可以尽快启动该任务；用户也需要衡量使用那种资源套餐更适合自己，从使用经验来看a100差不多是3090的两倍速，以batchsize的值对训练速度的影响为例，batchsize的值太大可能致使训练卡在dataloader环节，对于这种情况大显存也没用，显卡利用率较低；
* [确认配额](/train-spot/instruction/check_quota)：用户剩余的配额足够创建存储或启动该工作台，例如如果用户用量已经超过系统的3卡限额，系统会在点击“启动”按键的时候报错；

## 3.工作台和训练

[创建工作台](/train-spot/instruction/create_workspace) 和[创建训练](/train-spot/instruction/create_training) ，是填写任务信息和配置的操作；使用前，需要先了解他们的不同特点： ### 工作台的特点 - 工作台可以挂载块、文件、local-storage和对象存储。目前，对象存储缺省是按照只读挂载的（当然可以配置），其他存储可选读写关在或只读挂载。 - 工作台可以起CPU或GPU工作台，并指定CPU核数或GPU卡数。目前只能按整张GPU卡维度挂载。 - 工作台可以写入系统任何目录，但非Ti提供的存储（块存储，文件系统存储，对象存储，local-storage存储），其他地方都是临时存储，使用量超标会被K8S驱逐，慎用。 - 工作台可以多次登录，退出。重新启动后使用原配置，但可能调度到其他机器。 - 工作台的默认启动脚本是【 tail -f /dev/null】，请不要杀掉该进程，否则工作台会立即推出。 - 工作台一般会启动jupyter和SSH服务，不用指定启动脚本（缺省是tail -f /dev/null） 使用者可以在portal上终止工作台运行。

### 训练的特点

* 训练不能使用Jupyter和SSH方式登录，但是可以查看日志；
* 训练必须指定入口执行文件或脚本，工作台无需指定；
* 训练脚本执行完毕容器就退出，资源释放。工作台必须手工释放，或由于资源原因被Kubernetes驱逐或杀掉；
* 训练可以同时启动多台机器做分布式多机多卡训练，工作台只能在一台机器内；

# 块存储

块存储（Block，由Ti提供，不受K8S cgroup管制）是最传统也是使用最多的存储方式。在Ti平台里面，也是最主要的一种存储形态。Ti有两种方式提供块存储，一种是分布式块，由本地分布式集群提供，也是这里讲的Block存储。另外一种是本地块设备，比如硬盘提供临时使用的（见local-storage）。

### 优点

高速：一般使用Block设备会在上面格式化本地文件系统。虽然数据访问都是通过网络，但由于成熟的文件系统（比如Ext4，xfs）特性，大部分的元数据和部分数据都缓存在本地，访问速度可媲美本地文件系统。

易扩展：本地block存储集群如果需要扩展容量，添加几块盘到本地网络集群即可，方便快速。

易管理：分布式块存储有一定的局部性，即不同用户的数据相对隔离，元数据互相独立，这样就避免了海量访问时互相干扰的问题。

成熟，稳定：块存储和分布式块存储是发展最早也最成熟的。

可迁移：可挂载在任何一个Ti的训练容器中，这些容器可以在任何一台本地物理机上

### 缺点

价格比较贵：相对于对象存储来说，搭建分布式块集群的费用明显偏高，特别是在云端和数据量指数倍上涨的时候。

容量容易受到单个文件系统的上线限制，包括inode个数，最大存储容量等。

共享不方便：一般的块设备不能同时挂载在多个机器或Docker容器中同时读写。

但是，Ti系统的存储是一个统一的方案，通过cache，无缝对接对象存储等技术方式，可以有效解决块存储的缺点。训练结束后，这部分数据会一直保存，直到被显示删除。

# 文件系统

Ti的文件系统（FileSystem，由Ti提供，不受K8S cgroup管制）不是指在块存储上搭建出来的文件系统，而是一个分布式的NAS集群。它本身是一个全局的共享文件系统，不需要在块设备上格式化出文件系统。NAS的文件系统元数据管理和数据管理都是分布式的。相对于块存储，文件存储有如下的优缺点：

### 优点

共享方便，且可读写共享：这是由于文件系统采用分布式的全局元数据管理来保证数据一致性。

容量大：相比块存储，NAS在同一个文件系统可提供的容量大的多。

### 缺点

性能相对较差：由于每次更新操作都要到中心节点更新元数据，并发使用多会造成性能瓶颈。

大容量NASCAR稳定性待提高：特别是大容量时遇到单盘故障，恢复时间较长，且恢复期间性能显著受到影响。

目前，Ti系统（分布式）文件系统是受限使用的，海量的训练数据还是放在对象存储和块存储中。训练结束后，这部分数据会一直保存，直到被显示删除。

# 本地存储

本地存储（Local-Storage，由Ti提供，不受K8S cgroup管制）就是本地硬盘提供的存储。在Ti系统里面，本地存储通过Host方式直接挂载到容器中使用，容器消亡后存储消失。所以一般只用作临时存储，存放中间过程的数据。

### 优点

性能好，传输速度快：Ti系统中的本地存储一般都由NVMe或者SSD提供，并且Atom提供了缓存机制，把普通块设备（可以是SSD支持）和对象存储的数据无缝缓存到本地SSD或NVMe中。本地存储是性能最好的存储形态，如果量足够大，则长时间的深度学习训练数据都应该拷贝或有由Atom透明的拷贝到本地存储中使用，可以提升20%-30%的训练速度。 ### 缺点 数据安全无法保障：训练结束后，这部分数据会一直保存，直到被删除。但是下次挂载同类型的存储不一定能看到上次生成的数据，取决于是否调度到同一台机器，和取决于是否由别的用户改动过该数据。也就是说，这部分数据处于undefined状态，如果是比较重要的数据，请存放在块存储或者文件存储中，避免数据的丢失。

# 对象存储

目前海量的深度学习数据最理想的存储方式是对象存储。并且，由于Ti是基于云平台打造，训练的主体数据也都存放在云平台对象存储中。对象存储不提供标准的POSIX语义，Ti系统通过对接云端对象存储平台，提供类POSIX的访问方式，使用者可以直接对云端对象存储数据进行读写操作，在绝大部分场景下，使用姿势和使用本地文件系统无差异。并且，Ti系统提供了良好的cache功能，可以把云端对象存储数据动态缓存到本地的存储集群，极大提升存储访问效率。

同时，也提供工具方式，通过SMMC工具，支持从对象存储高速上传下载数据，并加以一定的权限控制，让数据在云端更安全。训练结束后，这部分数据会一直保存，直到被手动删除。

# 临时存储

临时存储（Empheral-Storage，由K8S和Docker提供，受cgroup管制），和CPU、内存一样，临时存储是K8S严格管控的一类存储，提供一个临时目录,是k8s提供的。

临时存储通过Docker的layer filesystem方式支持，把物理机系统盘的存储以layer方式挂载到用户容器中，用户可以直接进行读写，但是读写容量受到上限控制。一旦超过使用上线，容器将会被驱逐。用户的训练也会被终止。所以，训练者不应该主动的使用这部分存储，而只使用Ti系统提供的上述几类存储。容器消亡后，这部分数据会随之消失。

# 五、使用详解

本章节会对系统中每个功能点进行展开详细的说明，如果您对某一功能点的使用方法不太清楚，可以在本章中找到对应模块的使用详解，对照使用步骤的详细说明进行操作； - 使用前，需要完成gitalb账号注册并登录，详细操作指导可参考； - 如果您是算法工程师，完成算法训练的全部操作流程和功能介绍；
详细操作指导可参考[算法用户指南](/train%E2%80%94spot/user-guide)； - 如果您是系统管理员，完成系统管理和用户管理的全部操作流程和功能介绍；
详细操作指导可参考[系统管理员指南](/train%E2%80%94spot/admin-guide)；

# 查看配额

在创建存储和创建训练任务之前，需要先确认自己剩余可用的配额；避免出现存储创建失败，训练资源超出配额报错的情况；使用方法如下图所示：
步骤1:点击进入训练列表页面。

步骤2:点击训练列表中右上角的“查看个人配额”按键，会弹出配额使用情况的弹窗。

image

image

弹窗中会展示： 1.用户不同类型训练任务计算资源（CPU、GPU以及机器内存）的限额，如果创建训练任务时用量超过配额，启动训练会启动失败，报错“启动失败：insufficient quota”
2.用户不同类型存储资源（块存储、文件系统、本地存储）的限额，如果在内部存储模块新建存储时，会报错“创建存储失败”

# 查看资源

启动工作台和训练任务之前，需要查看集群的空闲资源，和每种任务类型都有哪些资源类型；

集群中job kind工作台只有3090的机器可供选择，但选择GPU型号的下拉栏中可以选择3090和A100两种机型，如果你创建的工作台选择了A100的GPU类型，由于套餐管理的设计缺陷，不能区分任务类型配置套餐，会导致选择GPU型号中会展示实际集群中没有的GPU类型；

换句话说，就是如果你为单机和创建了8卡A100机器的资源类型，工作台中可以看到并选择这一套餐。这样用户选择后，实际上集群中没有job kind为workspace（工作台）的A00机器，会导致用户创建的工作台一直处于队列中的状态，后续我们会对资源套餐进行优化，避免这种情况发生；

现阶段，为了用户能根据资源的忙闲状态，决定创建哪种资源类型的任务，尽早完成训练任务，并避免误选计算资源；我们提供了两种方式给用户查看资源的忙闲状态，如下问所述：

### 方法一：使用训练列表中的资源面板

训练平台资源面板：在训练列表中点击展开查看资源面板，根据不同任务类型，不同资源，以及资源的空闲、排队情况选择合适的训练资源，如下图所示：

image

image

注意：图中提到的占用资源、空闲资源、队列中GPU、已占用和空闲GPU的单位为卡数量；运行中任务和等待中任务为启动的pod数量；

### 方法二：使用grafana查看集群资源监控

Grafana资源看板：为了查看更详细的查看集群里有哪些机器有可用的资源，可以访问金山环境的grafana，地址为https://dashboard.atom.supremind.com/
grafana页面如下图所示： image

image

image

可以在grafana overview这个面板中，可以看到哪些机器有可用的GPU卡，点开node可以看到这台机器上有谁的任务在运行，大概运行了多久，和内存利用率，网络利用率这些参数。不仅可以看到哪些机器是空闲的，更便于用户和管理员排查是否机器存在问题；

# 个人配置设置

### 个人中心

用户登陆后，需要进入个人中心-个人信息，如下图所示；
image
进入个人信息页面，查看自己的个人信息是否准确，以及权限是否有算法用户的训练权限，如需开通权限请联系单位管理员，重新配置用户权限，否则会无法使用训练功能；

### 设置SSH

步骤1:确认好上述信息后，需要设置自己的公钥，否则工作台或训练可能起不来。因为登录SSH，登录Jupyter或查看log都需要自己的公钥。
设置公钥办法：点击进入“个人中心”—“设置”tab页面，点击”新增SSH”

image

image

步骤2:输入名称和自己的公钥（mac上的公钥是~/.ssh/id\_rsa.pub）
然后点击”确定”
至此，SSH密钥配置完毕，可以开始使用Ti系统了。

# 配置存储

### 使用场景

如果只是简单体验，不需要配置存储，大部分算法训练需要庞大的数据支撑，并且对存储的访问速度具有一定的要求。所以，使用Ti进行深度训练之前，需要配置存储。

### 存储说明

Ti系统可以配置块存储、文件存储、外部存储（bucket）；外部存储由管理员配置，普通用户只能使用。
- 块存储可以同时只读挂载到多个工作台或训练，但一旦由一个工作台或训练有写挂载，降无法继续挂载块存储。如果已经只读挂载了，也不能再写挂载。
- FileSystem存储可以在多个工作台或训练读写挂载，算法用户通常使用filesystem存储训练代码，使用块存储存储训练数据，块存储只能进行单独写和多只读的挂载方式。
- local-storage不需要配置，并且随着工作台或训练结束，自动消失。
不同类型存储的区别见上文：“基础知识介绍”-“存储”

### 配置步骤

Step1:点击左侧的“内部存储”模块，再点击左上角的“新建”按钮。如果已经有块存储创建成功，下面列表中会显示出当前用户已创建的存储。

image

image

Step2:点击“新建”后，输入存储名称，注意不可和自己已有的存储重名。输入容量大小，以G为单位。输入容量大小，以G为单位；选择对应的模式和所在的集群，如果下拉列表中存在多个集群可以选择，想使用某个集群资源, 则创建内部存储时应指定该集群, 那么挂载对应集群内部存储的训练任务会被自动调度到这个集群上；

image

image

step3:点击“确认”后创建成功，如果块存储的Quota没有超标的话，存储就建立成功了。否则，存储建立失败。

# 管理用户镜像

### 推送镜像

管理用户镜像就是往gitlab某个project的镜像仓库推送镜像的过程，Gitlab 中每个项目都有附属的容器镜像仓库, 可从侧边栏 Registry 菜单进入。

Step1: Ti系统用户可以将自定义镜像推送到有 developer 权限的官方repo中，推送到自己的私有repo中（建议推送至官方repo下，镜像与repo绑定）。在 Ti 的镜像管理菜单用户镜像页面, 可以查看所有自己能访问的项目中的容器镜像, 这些镜像都可以用于训练。

Step2: 需要在一定范围内共享的镜像, 可创建新 git 项目或使用已有项目存放 Dockerfile, 并配置可以访问该项目成员的权限. 既可手动编译镜像推送至该项目, 也可以配置 Gitlab CI job 自动或半自动编译镜像并推送；

image

image

### 查看和使用个人镜像、公共镜像

查看可用镜像：用户在“镜像”模块中，可以查看用户有权使用的公开镜像和用户个人镜像，可以看到镜像的信息包括镜像版本，镜像大小，镜像地址以及创建时间；如下图所示： image

使用镜像： 在创建任务的第二步中需要选择挂载的镜像（在后文的创建训练任务中会详细介绍），可以像上面查看镜像的步骤一样选中镜像列表中的镜像，也可以通过输入镜像地址的方式，进行训练任务镜像的挂载；如下图所示： image

# 创建工作台

在Kubernetes里面，工作台和训练都是一个POD。这些POD会挂载存储资源，关联CPU、内存和GPU资源，是使用者进行深度训练的地方。但是，工作台和训练有较大的区别。
工作台是给算法工程师调试代码、整理数据、验证环境使用的。一般来说，工作台不会共享，而且会长期使用。这也意味着，工作台占用的CPU、GPU、内存不会释放，也无法共享。

### 创建工作台

使用工作台之前要提前配置好SSH公钥和并确认自己的剩余配额（默认每人只有3卡配额），详细操作在“使用前准备”和“使用前确认”介绍过；

步骤1:从左侧栏点击训练（现在创建工作台和训练从同一个入口进入），然后右边点击“新建”按键；

image

image

步骤2:输入工作台所需信息：

image

image

名称：注意名称不能和自己已经创建的工作台或训练重复，且需要特定的字符规则。名称由4~63个字符组成，可包含字母、数字、中划线、下划线。
然后选择【工作台】。另外两个【单机训练】和【分布式训练】是给训练使用的。
描述可以不填，然后点击下一步。

步骤3:接下来选择需要的镜像：

image

image

其中，有3中类型镜像：

A）用户镜像：这个是用户推送到自己gitlab仓库的镜像，除非共享，一般只对自己可见。
B）公共镜像：这个是Ti平台提供的公共镜像，分为CPU和GPU镜像，其中，GPU镜像大部分预装了CUDA和CUDNN库，可以直接使用。
C）镜像地址：如果用户知道镜像地址，并且镜像仓库可以直接在Ti的物理机上直接登录访问的，可以使用镜像地址。比如docker hub的镜像，gitlab的镜像等。 选择好镜像后，点击【下一步】

步骤4:选择内部存储：
数据集是Ti提供的可直接用于训练的数据集合。用户可以通过smmc等工具将数据集会拷贝到内部存储中，通过挂载内部存储提供训练的数据集。

image

image

步骤5:选择GPU或CPU套餐
A.选择GPU套餐：
image

工作台只能在单台GPU中运行，根据GPU的规格，可以配置不同的套餐。一般是根据物理机GPU卡数，除了留下部分给系统运行，剩下部分的CPU和内存按卡平均分配。比如在8卡的GPU机器上，可以挑选1卡，2卡的3090型号的GPU套餐。

B.选择CPU套餐：
选择CPU套餐和GPU套餐类似，不过资源类型要选择CPU; image

步骤6:配置训练工具工作台选择好GPU套餐之后，还有两个配置需要设置，这两个设置默认关闭，分别是：

1）启用Jupyter：Jupyter是运行在网页Portal的可视化深度学习调试环境，可直接运行Python代码并输出结果。Ti缺省关闭该配置。Jupyter缺省关闭的。

2）启用SSH：如果前期已经配置过公钥，那么从公钥所在机器上可直接通过SSH登录工作台，方便使用。SSH缺省打开的，如果镜像没有安装sshd服务，不要打开它。

但是需要注意的是，如果要启动Jupyter或SSH，镜像中必须安装有Jupyter和SSH服务，否则启动会失败。

Step7:添加环境变量：
点击Env旁的加号进入添加界面，输入key和value即可为该任务添加环境变量 image

Step8:全部配置完成后，点击”创建训练”，即可创建一个训练的模板;

Step9:此时训练还未启动，需要点击【启动训练】按钮才会启动; image

# 创建训练

训练任务主要分为2种，包括非预置模型训练和预置模型训练，非预置模型训练指从头至尾完整的训练完一个模型，预置模型训练指在先前训练完成的基础模型的基础上继续进行训练。进行非预置模型训练时，目前支持单机训练和分布式训练；进行预置模型训练时，目前仅支持单机训练。顾名思义，单机训练只能运行在一台机器上，并且卡数不能超过机器的最大卡数。单机训练从资源类型上，分为GPU训练和CPU训练。

## 非预置模型训练

步骤1:从左侧模块栏中，点击“训练”，进入训练列表，点击训练列表中的“新建”，进入新建训练任务的页面；

image

image

步骤2:输入名称，类型选择“单机训练”或者“分布训练”，描述可不填，点击下一步。

步骤3:选择镜像。分为用户镜像，公开镜像和镜像地址。注意，这些镜像都在gitlab鉴权范围，都需要能通过物理机能访问到。

image

image

镜像地址可以填项目镜像，也可以填docker.io的镜像，或其他镜像源。

步骤4:挂载存储。内部存储（又分为块存储，文件系统存储）。如果知识试用可以不挂载存储，此时点击上面的【跳过这一步】即可。

image

image

步骤5:选择工作台资源：这一步单机和工作台类似。分布式训练则需要额外在进行实例数的配置，大多数情况下选择2个实例。

## 预置模型训练

步骤1:从左侧模块栏中，点击“训练”，进入训练列表，点击训练列表中的“新建”，进入新建训练任务的页面； image

步骤2:配置训练任务参数，定义训练任务名称，并选择任务类型为「预置模型训练」，点击进入「下一步」；

步骤3:选取系统提供的公共预训练模型及配套框架。注意，该步骤下，用户需要选择适合目标业务场景的模型进行训练，目前支持选择分类模型或检测模型，然后点击进入「下一步」；

步骤4:选取在DataOcean中准备好的训练集，分类模型选择对应的分类数据集，目前可选择公开数据集lmageNet和CIFAR-10，检测模型选择对应的检测数据集，目前可选择公开数据集COCO，然后点击进入「下一步」；

步骤5:配置训练模式和训练精度，可直接使用默认的推荐值。同时也可以调整任务的具体类别。配置完成后，点击进入「下一步」； image

步骤6:选择需要用于训练的硬件资源，目前预置模型训练仅支持GPU进行训练，其中GPU资源目前提供3090。用户完成模型配置后，在配置总览中确认各项即可点击「提交」启动该模型任务的提交；

# 查看任务结果

工作台或者训练创建成功后，用户可以查看自己已经创建的工作台或训练，并进行相应的操作。管理员可以查看并操作所有用户的工作台和训练。

image

image

各列的含义如下：

* 名称：训练或工作台的名称。同一用户下训练和工作台不能重名。
* 创建者：创建该工作台或训练的用户
* 镜像：训练使用的镜像名称
* 状态：训练的状态：可能为
* 已创建：表示刚创建成功，可以点击小三角按钮启动
* 运行中：该训练或工作台正在运行
* 已完成：训练或工作台已经结束。训练脚本运行完毕后，自动变为结束状态。工作台的entry如果意外或主动退出，也会编程完成状态
* 队列中：已经启动训练或工作台，正在等待资源或在启动过程中。启动成功后编程运行中状态，否则为Error状态。
* Error：由于训练中程序异常退出，或者由于K8S启动过程出错，比如存储挂载出错，镜像拉取出错等，工作台和训练会处于Error状态，此时应由运维人员接入分析并重置状态。
* 工作台和训练列表还可以按照一定条件搜索；

image

image

比如可以按照名称、创建者，状态，模式搜索。这对于由很多工作台和训练的使用者来说很方便。 如果设置了搜索条件，则某一栏的右侧搜索图标会变高亮。可以点击【重置】按钮重置掉搜索条件。 对于正在运行的工作台，可以进行如下操作：

停止：在【启动/终止】栏里面，点击停止符号，则工作台被释放。
Jupyter:操作栏里面点击Jupyter，可启动Jupyter Portal。注意，这需要镜像安装Jupyter组件并在启动时勾选了Jupyter选项。
SSH:点击【SSH】把登录SSH的命令拷贝到当前电脑系统剪贴板。注意，需要镜像安装SSHD 服务，并且工作台启动时勾选SSH选项。
日志：点击【日志】，查看当前工作台或训练运行日志。一般日志对训练意义比较大，因为工作台可以直接登录，无需通过日志查看内容。

查看日志时，会显示如下的界面：

image

image

# 队列管理

### 背景

由于Ti系统用户数量不断增加，集群资源不足的问题时常发生，为了更公平、科学的分配训练资源，Ti v2.1.0版本新增了训练任务的排队功能。

### 调度逻辑

调度目前是对特定机型进行的排队，调度的先后顺序的判断依据：管理员置顶>分数>用户拖拽优先级>创建时间，意思是管理员置顶的任务优先级是最高的；没有置顶的任务通过用户的排队分数进行判断优先级，用户近24小时内对这种资源使用越少的人分数越低，排队的优先级越高，排队时会插队排在其他用户的前面；

用户也可以通过在训练队列页面进行任务的拖拽，决定个人同种资源的任务哪个会被先调度上；如果个人用户没有拖拽任务，系统对个人用户任务同种资源的调度顺序则会按照创建时间进行排序，在排队时同种机型的任务会发生阻塞，不同机型的任务不会发生阻塞。单机和分布式任务在两条队列中，两种任务之间不会进行排序；

在”单机训练队列“和”多机训练队列“的列表中可以看到该类型中所有在排队的任务，目前队列不会根据序号顺序进行调度；后续会按照是否置顶以及优先级将任务倒序排列；

### 排队统计

用户排队分数可以在后台查询到如图下所示的用户排队分数：

image

image

在队列管理页面顶部是集群中任务排队情况的统计面板，可以看到当前工作台、单机训练和多机训练的使用情况：运行中的任务、等待中的任务、该类型任务占用的资源数量（pod的数量），以及空闲资源数量（剩余卡数）,如下图所示。

image

image

### 查看全部任务

在下部的三个tab页中，“全部任务”可以看到所有人所有状态的工作台和训练任务，页面“占用机器”是启动这个任务所需的pod数量，点击“日志”可以看这个任务输出到Ti的日志，用户重定向日志则无法在Ti训练平台查看。

image

image

### 调整队列

目前是不会按照调度顺序展示序号，可以在这里手动将任务置顶，置顶的任务会比所有普通任务优先调度，后置顶的任务会比先置顶的任务优先调度，也可以手动关闭这个排队任务。

image

image

# 套餐管理

管理员配置好的资源套餐不会根据目前资源池中剩余的资源进行动态变化，需要管理员定期根据集群资源的变化进行资源套餐配置的更新；目前配置好的资源套餐是由工作台，单机训练和分布式训练公用的，存在着用户可以选到当前类型任务类型下没有这种资的情况，这一设计会在后续迭代中优化；

管理员进行套餐管理，需要进入“系统管理”——“套餐管理”tab页面，在如下所示界面中进行套餐管理；

image

image

### 删除套餐

点击左侧的checkbox可以选中已经创建好的套餐资源，通过点击“删除”按钮进行批量删除；

### 新建套餐

如需新增套餐，可以点击“新建”按钮，填写弹窗中的信息（带星号的为必填项），完成后点击确认进行创建，如下图所示；

image
其中，开启RDMA开关是通过网络把资料直接传入计算机的存储区，将数据从一个系统快速移动到远程系统存储器中，而不对操作系统造成任何影响，这样就不需要用到多少计算机的处理功能；

创建好后用户可以在新建工作台、单机训练和多机训练使用这些套餐，是系统管理员在套餐管理中预先配置的，如下图所示，

image

image

# 配额管理

用户会在创建任务界面看到自己的配额，根据看到的剩余配额作为参考，创建训练任务； 下图中就是训练列表中看到的用户配额信息： image

配额管理默认是没有限制的，如果需要新增配额的限制 需要管理员在“系统管理-配额管理”中通过搜索用户名找到注册登陆过的用户，点击对应用户名列表右侧的“编辑”按键，对用户的配额限制进行设置；
点击编辑后会弹出如下弹窗：

image

image

### 用户新增配额限制

1.在对应计算资源和存储资源的右上角点击“添加”，如上图所示； 2.设置计算和存储资源的配额，如下图所示； 计算资源分为三种类型，工作台、单机训练、分布式训练，可以对不同类型的任务的配额进行限制，例如给用户A的工作台配额设为3卡GPU，同时单机训练任务配额设置为16卡GPU资源；

image

image

存储资源同样分为三种类型，块存储、文件系统和本地存储，可以对不同类型的存储容量配额进行限制，不同类型存储的区别见上文：“基础知识介绍”-“存储”

image

image