**docker技术概述**

## 1 docker是什么

### 1.1镜像(image)

docker是世界领先主流的软件容器平台。要理解docker的概念需要先了解什么是容器，要了解容器必须先了解什么是docker镜像。

镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数。 镜像不包含任何动态数据，其内容在构建之后也不会被改变。docker镜像的构建原理比较复杂，主要使用分层存储的架构，充分利用union FS(union File System**)**联合文件系统，最终的镜像由多层文件系统联合组成。联合文件系统参考资料<https://www.jianshu.com/p/3ba255463047>。

举个例子来理解docker中镜像的概念，我们用光盘给一台电脑重做一个操作系统，光盘就是一种镜像，这个镜像可以给很多台电脑装操作系统，这里面每一个操作系统就是该镜像的一个实例。docker使用镜像创建的实例，我们称为容器container。

### 1.2容器(container)

镜像（image）和容器（container）的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。官方对容器的解释：容器就是将软件打包成标准化单元，以用于开发、交付和部署。容器有以下的特点：

* 轻量的、可执行的独立软件包 ，包含软件运行所需的所有内容：代码、运行时环境、系统工具、系统库和设置。
* 适用于基于Linux和Windows的应用，在任何环境中都能够始终如一地运行。
* 容器赋予了软件独立性，使其免受外在环境差异（例如，开发和生产环境的差异）的影响，从而有助于减少团队间在相同基础设施上运行不同软件时的冲突。

### 1.3仓库(repository)

仓库是集中存放镜像的地方，镜像构建完成后，可以很容易的在当前宿主上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，docker registry就是这样的服务。一个 docker registry中可以包含多个仓库，每个仓库可以包含多个标签（Tag）,每个标签对应一个镜像。官方最大的镜像仓库是docker hub 地址：<https://hub.docker.com/>，如图1所示。

例如：工作中需要在本地机器上安装tensorflow的不同版本，比如cpu，gpu版本，我只需要在docker hub上搜索关键字tensorflow，如图2所示，在本地安装好docker，执行箭头所指命令即可把tensorflow-gpu版本的镜像pull到本地，然后使用docker命令你可以在本地启用多个tensorflow实例，docker命令在另一篇侧重实践的文章中会详细介绍，本篇侧重概念的介绍，不再叙述。

国内访问docker hub官网的速度比较慢，所以国内一些云提供商提供了类似docker hub的服务。如：[网易云镜像服务](https://www.163yun.com/product/repo)<https://www.163yun.com/product/repo>，[DaoCloud 镜像市场](https://www.daocloud.io/)<https://www.daocloud.io/>，[阿里云镜像库](https://www.aliyun.com/product/containerservice?utm_content=se_1292836)等

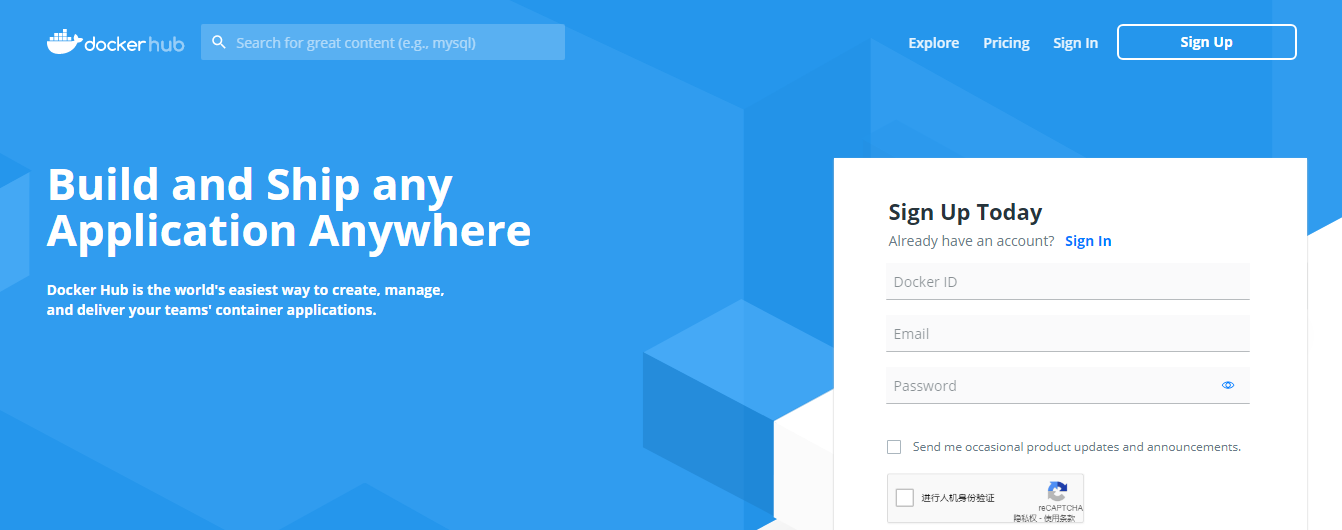


图1 docker hub官方网站

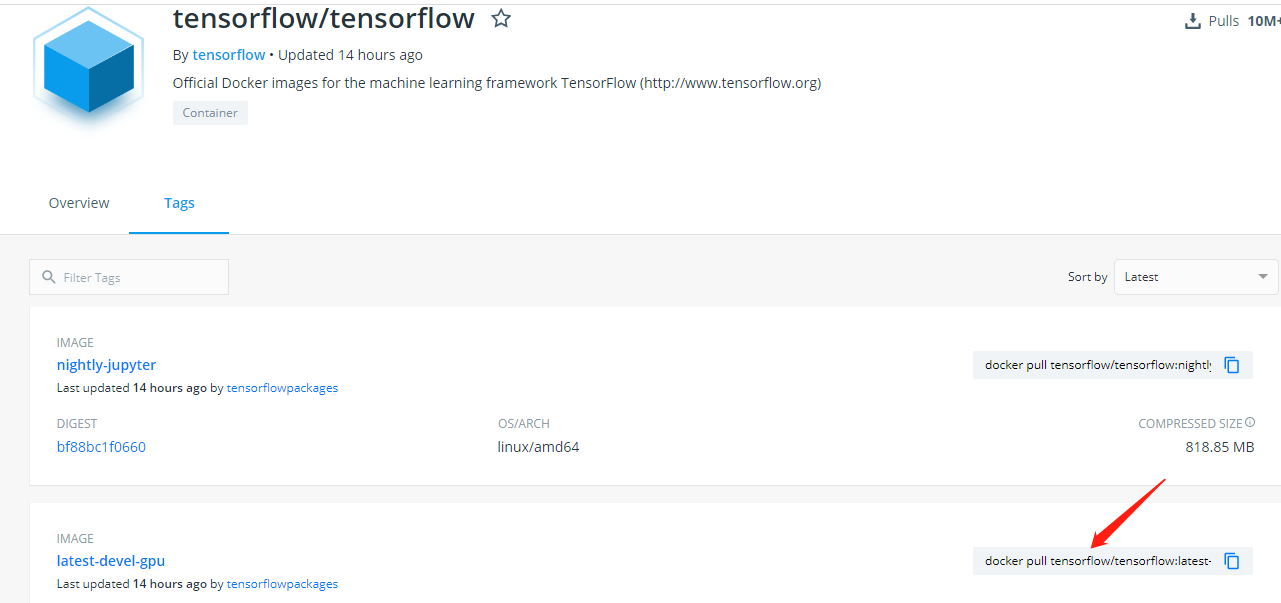


图2 docker hub中获取tensorflow镜像

## 2 为什么要使用docker

### 2.1 docker的优点

* 一致的运行环境：docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，从而不会再出现 “这段代码在我机器上没问题，但是换到生产环境出错” 这类问题
* 更快速的启动时间：可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。docker容器启动只需要一条命令即可完成，一次构建，多次使用。
* 隔离性：避免公用的服务器，资源会容易受到其他用户的影响。例如：多个用户使用同一台服务器，用户A需要python2.7环境，tensorflow-gpu版本，cuda8.5，用户B需要python3.6环境，pytorch，cuda7.5，用户C需要python3.6环境，mxnet，cuda8.0。传统的方法用户需要安装不同的版本，配置不同环境变量满足自己的计算需求。可能某一天cuda的版本A升级到了9.0不小心污染了环境，导致用户B和C无法使用了。如果使用docker，用户A，B，C只需要按照自己的需求，启用自己需要的版本docker 镜像，对其他用户没有任何影响。
* 弹性伸缩，快速扩展：善于处理集中爆发的服务器使用压力；
* 迁移方便：可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。
* 持续交付和部署：使用docker可以通过定制应用镜像来实现持续集成、持续交付、部署。

### 2.2 容器和虚拟机的区别

在容器技术之前，虚拟化技术是是通过虚拟机技术来实现的，主要代表有VMWare和OpenStack。虚拟机就是在自己的操作系统里面安装虚拟机软件，然后启动多个“子电脑”，在“子电脑”里，你可以和正常电脑一样运行程序，例如运行各种应用软件。每个“子电脑”之间，是相互隔离的，互不影响。docker这样的容器技术，也是虚拟化技术，属于轻量级的虚拟化。两者的区别主要有以下：

* 传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，在该系统上再运行所需应用进程，而容器内的应用进程直接运行于宿主的内核，容器内没有自己的内核，而且也没有进行硬件虚拟，因此容器要比传统虚拟机更为轻便，如图三所示。
* 虚拟化技术依赖的是物理cpu和内存，是硬件级别的；而我们的 docker 是构建在操作系统层面的，利用操作系统的容器化技术，所以 docker 同样的可以运行在虚拟机上面。
* 我们知道虚拟机中的系统就是我们常说的操作系统镜像，比较复杂；而 docker 比较轻量级，我们可以用 docker 在一台机器部署多个相互独立的数据库应用，就类似于在多个虚拟机中，每个虚拟机中安装一个数据库应用，我们用 docker 部署的应用是完全隔离的。
* 传统虚拟化技术在构建系统的时候非常复杂；而 docker 可以通过一个简单的dockerfile 文件来构建整个容器，更重要的是 dockerfile 可以手动编写，这样应用程序开发人员可以通过发布 dockerfile 来定义应用的环境和依赖，这样对于持续交付非常有利。相关dockerfile文件内容在
* 容器是一个应用层抽象，用于将代码和依赖资源打包在一起。多个容器可以在同一台机器上运行，共享操作系统内核，但各自作为独立的进程在用户空间中运行。与虚拟机相比，容器占用的空间较少（容器镜像大小通常只有几十兆），瞬间就能完成启动 。
* 虚拟机 (VM) 是一个物理硬件层抽象，用于将一台服务器变成多台服务器。 管理程序允许多个 VM在一台机器上运行。每个VM都包含一整套操作系统、一个或多个应用、必要的二进制文件和库资源，因此 占用大量空间。而且 VM 启动也十分缓慢 。

docker的官方网站上描述了很多docker的优势，但是并没有说虚拟机不如容器技术，虚拟机更擅长于彻底隔离整个运行环境。例如，云服务提供商通常采用虚拟机技术隔离不同的用户。而 docker通常用于隔离不同的应用 。



图3 容器和虚拟机中的对比

## 3 k8s

### 3.1 k8s是什么

k8s也就是“kubernetes”，它是通过将8个字母“ubernete”替换为“8”而导致的一个缩写。官方网站上给的解释，k8s是一个工业级的容器编排平台。“kubernetes”这个单词是希腊语，它的中文翻译是“舵手”或者“飞行员”。docker中文意思里有“码头工人”的意思，container有“集装箱”的意思，如图4所示形象的描述了k8s的功能，这是一艘载着一堆集装箱（container）的轮船，轮船在大海上运着集装箱奔波，把集装箱送到它们该去的地方。k8s也就借着这个寓意，希望成为运送集装箱的一个轮船，来帮助我们管理这些集装箱，也就是管理这些容器。



图4 k8s的功能就像是运输集装箱的轮船

### 3.2 k8s核心功能

* 服务的发现与负载的均衡。
* 容器的自动装箱，我们也会把它叫做 scheduling，就是“调度”，把一个容器放到一个集群的某一个机器上，k8s会帮助我们去做存储的编排，让存储的声明周期与容器的生命周期能有一个连接。
* k8s会帮助我们去做自动化的容器的恢复。在一个集群中，经常会出现宿主机的问题或者说是os的问题，导致容器本身的不可用，k8s会自动地对这些不可用的容器进行恢复。
* k8s会帮助我们去做应用的自动发布与应用的回滚，以及与应用相关的配置密文的管理，对于job类型任务，k8s可以去做批量的执行。为了让这个集群、这个应用更富有弹性，k8s也支持水平的伸缩。

下面分别举三个例子来描述k8s的核心功能：

1. 调度

k8s可以把用户提交的容器放到k8s管理的集群的某一台节点上去。k8s的调度器是执行这项能力的组件，它会观察正在被调度的这个容器的大小、规格。

比如说它所需要的cpu以及它所需要的memory，然后在集群中找一台相对比较空闲的机器来进行一次放置（placement）的操作。如图5所示，k8s会把红颜色的这个容器放置到第二个空闲的机器上，来完成一次调度的工作。

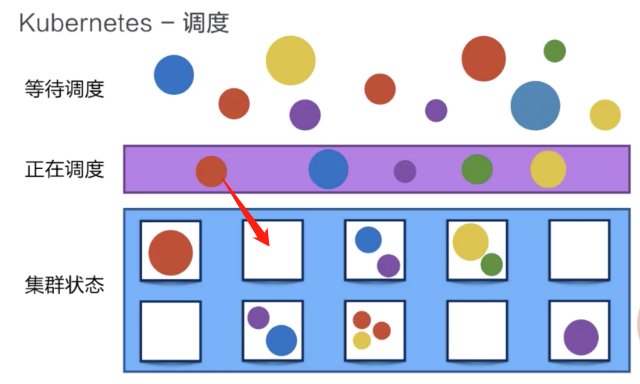


图5 k8s的调度功能

1. 自动修复

k8s有一个节点健康检查的功能，它会监测这个集群中所有的宿主机，当宿主机本身出现故障，或者软件出现故障的时候，这个节点健康检查会自动对它进行发现。下面 k8s会把运行在这些失败节点上的容器进行自动迁移，迁移到一个正在健康运行的宿主机上，来完成集群内容器的自动恢复。图6右下角节点运行故障，k8s会把该宿主机上运行的紫色容器迁移到图7所示右上角的节点位置。



图6 k8s对节点健康状况检查



图7 k8s对故障节点中容器进行迁移

1. 水平伸缩

k8s有业务负载检查的能力，它会监测业务上所承担的负载，如果这个业务本身的cpu利用率过高，或者响应时间过长，它可以对这个业务进行一次扩容。如图8所示，黄颜色的容器过度忙碌，k8s就可以把黄颜色负载从一份变为三份，接下来，它就可以通过负载均衡把原来图8的黄颜色容器上的负载平均分到图9中的三个黄颜色的负载上去，以此来提高响应的时间。



图8 k8s检测到黄色的容器负载过高



图9 k8s对黄色的容器负载均衡

### 3.3 k8s架构

k8s架构是一个比较典型的二层架构和server-client架构。Master作为中央的管控节点，会去与Node进行一个连接。所有UI的、clients这些user侧的组件，只会和Master进行连接，把希望的状态或者想执行的命令下发给Master，Master会把这些命令或者状态下发给相应的节点，进行最终的执行。

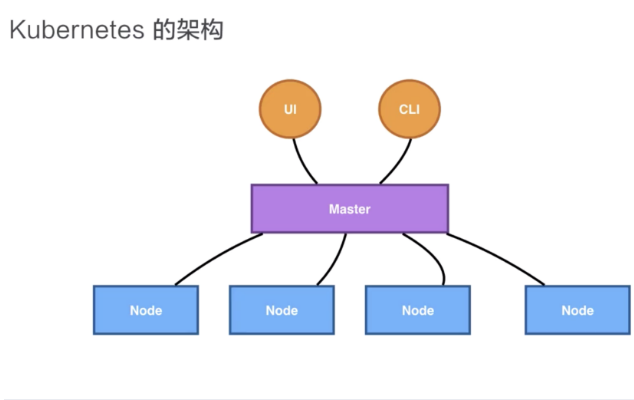


图10 k8s的架构

1. Master

k8s的master包含四个主要的组件：API Server、Controller、Scheduler 以及 etcd。如图11所示：

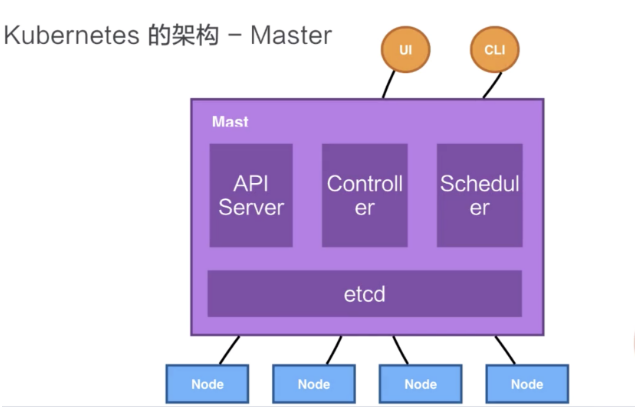


图11 k8s中master结构

* API Server：顾名思义是用来处理API操作的，K8S中所有的组件都会和 API Server进行连接，组件与组件之间一般不进行独立的连接，都依赖于API Server 进行消息的传送。
* Controller：是控制器，它用来完成对集群状态的一些管理。比如刚刚我们提到的两个例子之中，第一个自动对容器进行修复、第二个自动进行水平扩张，都是由k8s中的Controller来进行完成的。
* Scheduler：是调度器，“调度器”顾名思义就是完成调度的操作，就是我们刚才介绍的第一个例子中，把一个用户提交的Container，依据它CPU、对 memory请求大小，找一台合适的节点，进行放置；
* etcd：是一个分布式的一个存储系统，API Server 中所需要的这些原信息都被放置在 etcd中，etcd本身是一个高可用系统，通过etcd保证整个k8s的Master组件的高可用性。
* 我们刚刚提到的 API Server，它本身在部署结构上是一个可以水平扩展的一个部署组件；Controller 是一个可以进行热备的一个部署组件，它只有一个 active，它的调度器也是相应的，虽然只有一个 active，但是可以进行热备。

1. Node

k8s的Node是真正运行业务负载的，每个业务负载会以Pod的形式运行。等一下我会介绍一下Pod的概念。一个Pod中运行的一个或者多个容器，真正去运行这些Pod的组件的是叫做 kubelet，也就是Node上最为关键的组件，它通过API Server接收到所需要Pod运行的状态，然后提交到我们下面画的这个Container Runtime组件中。

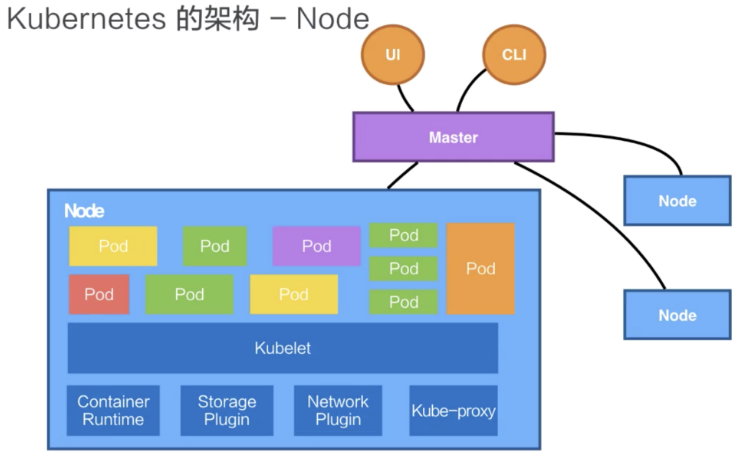


图12 k8s中Node

1. k8s调度的例子

在OS上去创建容器所需要运行的环境，最终把容器或者Pod运行起来，也需要对存储跟网络进行管理。k8s并不会直接进行网络存储的操作，他们会靠Storage Plugin或者是网络的Plugin来进行操作。用户自己或者云厂商都会去写相应的 Storage Plugin或者 Network Plugin，去完成存储操作或网络操作。在k8s自己的环境中，也会有k8s的Network，它是为了提供Service network 来进行搭网组网的。（“service”这个概念后面的文章会介绍）真正完成 service 组网的组件的是 Kube-proxy，它是利用了iptable的能力来进行组建 k8s的Network，就是cluster network，以上就是Node上面的四个组件。

k8s的Node并不会直接和user进行interaction，它的interaction只会通过 Master。而user是通过Master向节点下发这些信息的。k8s每个Node上，都会运行我们刚才提到的这几个组件。下面我们以一个例子再去看一下 k8s架构中的这些组件，是如何互相进行 interaction 的。

用户可以通过UI或CLI提交一个Pod给k8s进行部署，这个Pod请求首先会通过CLI或者UI提交给k8s API Server，下一步API Server会把这个信息写入到它的存储系统etcd，之后Scheduler会通过API Server的watch或者叫做 notification机制得到这个信息：有一个 Pod 需要被调度。这个时候 Scheduler 会根据它的内存状态进行一次调度决策，在完成这次调度之后，它会向API Server report说：“OK！这个 Pod 需要被调度到某一个节点上。”

这个时候 API Server接收到这次操作之后，会把这次的结果再次写到etcd 中，然后API Server会通知相应的节点进行这次Pod真正的执行启动。相应节点的kubelet会得到这个通知，kubelet就会去调Container runtime来真正去启动配置这个容器和这个容器的运行环境，去调度Storage Plugin来去配置存储，network Plugin去配置网络。

这个例子我们可以看到：这些组件之间是如何相互沟通相互通信，协调来完成一次Pod的调度执行操作的。

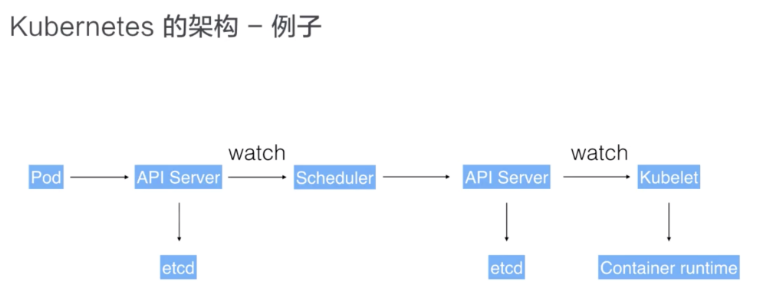


图13 k8s实例分析

## 4 基于k8s的tensorflow分布式

### 4.1 k8s+tensorflow架构

Tensorflow作为深度学习领域逐渐成熟的项目，以其支持多种开发语言，支持多种异构平台，提供强大的算法模型，被越来越多的开发者使用。但在使用的过程中，尤其是GPU集群的时候，我们或多或少将面临以下问题：

* 资源隔离。Tensorflow（以下简称tf）中并没有租户的概念，何如在集群中建立租户的概念，做到资源的有效隔离成为比较重要的问题；
* 缺乏GPU调度。tf通过指定GPU的编号来实现GPU的调度，这样容易造成集群的GPU负载不均衡；
* 进程遗留问题。tf的分布式模式ps服务器会出现tf进程遗留问题；
* 训练的数据分发以及训练模型保存，都需要人工介入；
* 训练日志保存、查看不方便；

因此，我们需要一个集群调度和管理系统，可以解决GPU调度、资源隔离、统一的作业管理和跟踪等问题。tensorflow on kubernetes包含三个主要的部分，分别是client、task和autospec模块。client模块负责接收用户创建任务的请求，并将任务发送给task模块。task模块根据任务的类型（单机模式和分布式模式）来确定接下来的流程：如果type选择的是single（单机模式），对应的是tf中的单机任务，则按照按照用户提交的配额来启动container并完成最终的任务。如果type选择的是distribute（分布式模式），对应的是tf的分布式任务，则按照分布式模式来执行任务。需要注意的是，在分布式模式中会涉及到生成clusterspec信息，autospec模块负责自动生成clusterspec信息，减少人工干预。图14是tensorflow on kubernetes的架构图：

图14 k8s+tensorflow架构图

### 4.2 tensorflow分布式架构

对于分布式模式，情况要稍微复杂些。下面先简单介绍一下tensforlow分布式框架。tensorflow的分布式并行基于gRPC框架，client负责建立Session，将计算图的任务下发到TF cluster上。TF cluster通过tf.train.ClusterSpec函数建立一个cluster，每个cluster包含若干个job。 job由好多个task组成，task分为两种，一种是PS（Parameter server），即参数服务器，用来保存共享的参数，还有一种是worker，负责计算任务。

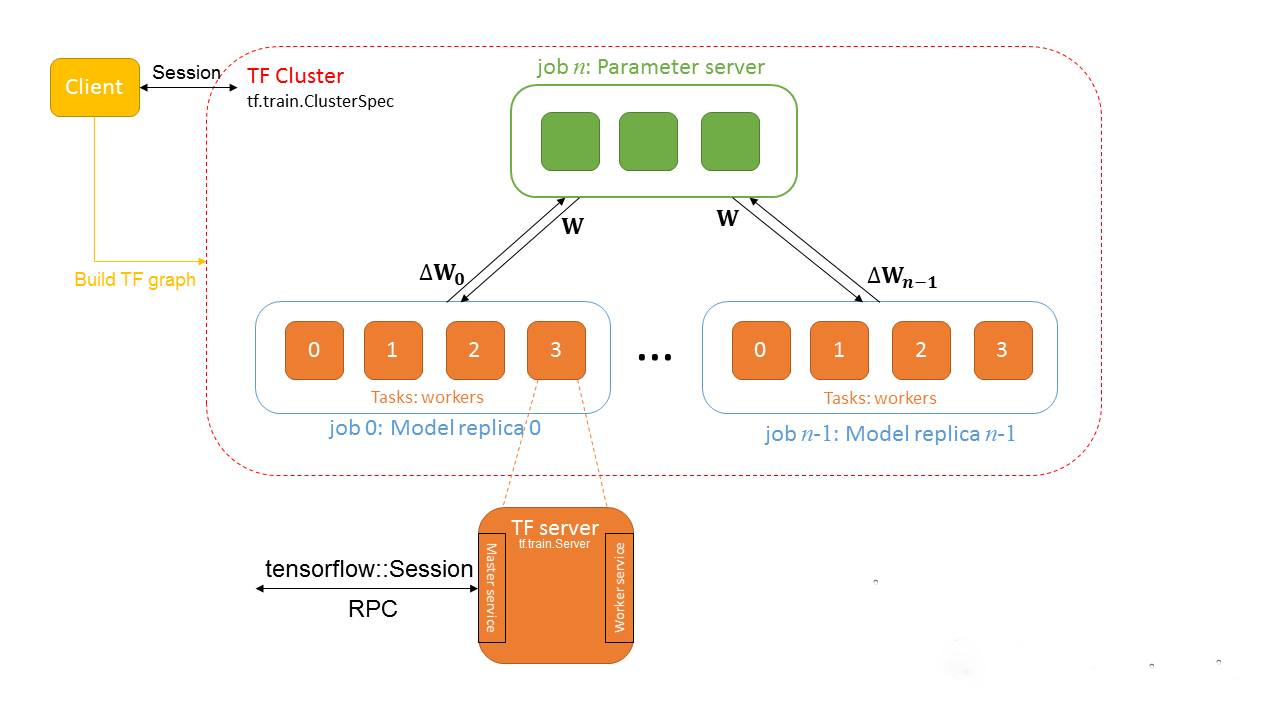


图14 tensorflow分布式架构图

我们在执行分布式任务的时候，需要指定clusterspec信息，如下面的任务，执行该任务需要一个ps和两个worker，我们先需要手动配置ps和worker，才能开始任务。这样必然会带来麻烦。如何解决clusterspec，成为了一个必须要解决的问题。所以在提交分布式任务的时候，task需要autospec模块的帮助，收集container的ip后，才能真正启动任务。所以分布式模式要做两件事情：tf分布式模式的node按照角色分为ps（负责收集参数信息）和worker，ps负责收集参数信息，worker执行任务，并定期将参数发送给worker。

要执行分布式任务，涉及到生成clusterspec信息，模型的情况下，clusterspec信息是通过手动配置，这种方式比较麻烦，而且不能实现自动化，我们引入autospec模型很好的解决此类问题。Autospec模块只有一个用途，就是在执行分布式任务时，从container中收集ip和port信息后，生成clusterspec，发送给相应的container。