

ZStack 技术白皮书精选

基于 ZStack 构建深度学习云平台

扫一扫二维码，获取更多技术干货吧



版权声明

本白皮书版权属于上海云轴信息科技有限公司，并受法律保护。转载、摘编或利用其它方式使用本调查报告文字或者观点的，应注明来源。违反上述声明者，将追究其相关法律责任。

摘要

大道至简·极速部署，ZStack 致力于产品化私有云和混合云。

ZStack 是新一代创新开源的云计算 IaaS 软件，由英特尔、微软、CloudStack 等世界上最早一批虚拟化工程师创建，拥有 KVM、Xen、Hyper-V 等成熟的技术背景。

ZStack 创新提出了云计算 4S 理念，即 Simple（简单）、Strong（健壮）、Smart（智能）、Scalable（弹性），通过全异步架构，无状态服务架构，无锁架构等核心技术，完美解决云计算执行效率低，系统不稳定，不能支撑高并发等问题，实现 HA 和轻量化管理。

ZStack 发起并维护着国内最大的自主开源 IaaS 社区——zstack.io，吸引了 6000 多名社区用户，对外公开的 API 超过 1000 个。基于这 1000 多个 API，用户可以自由组装出自己的私有云、混合云，甚至利用 ZStack 搭建公有云对外提供服务。

ZStack 拥有充足的知识产权储备，积极申报多项软著和专利，参与业内标准、白皮书的撰写，入选云计算行业方案目录，还通过了工信部云服务能力认证和信通院可信云认证。

ZStack 面向企业用户提供基于 IaaS 的私有云和混合云，是业内唯一一家实现产品化，并领先业内首家推出同时打通数据面和控制面无缝混合云的云服务商。选择 ZStack，用户可以官网直接下载、1 台 PC 也可上云、30 分钟完成从裸机的安装部署。

目前已有 1000 多家企业用户选择了 ZStack 云平台。

基于 ZSTACK 构建深度学习云平台

前言

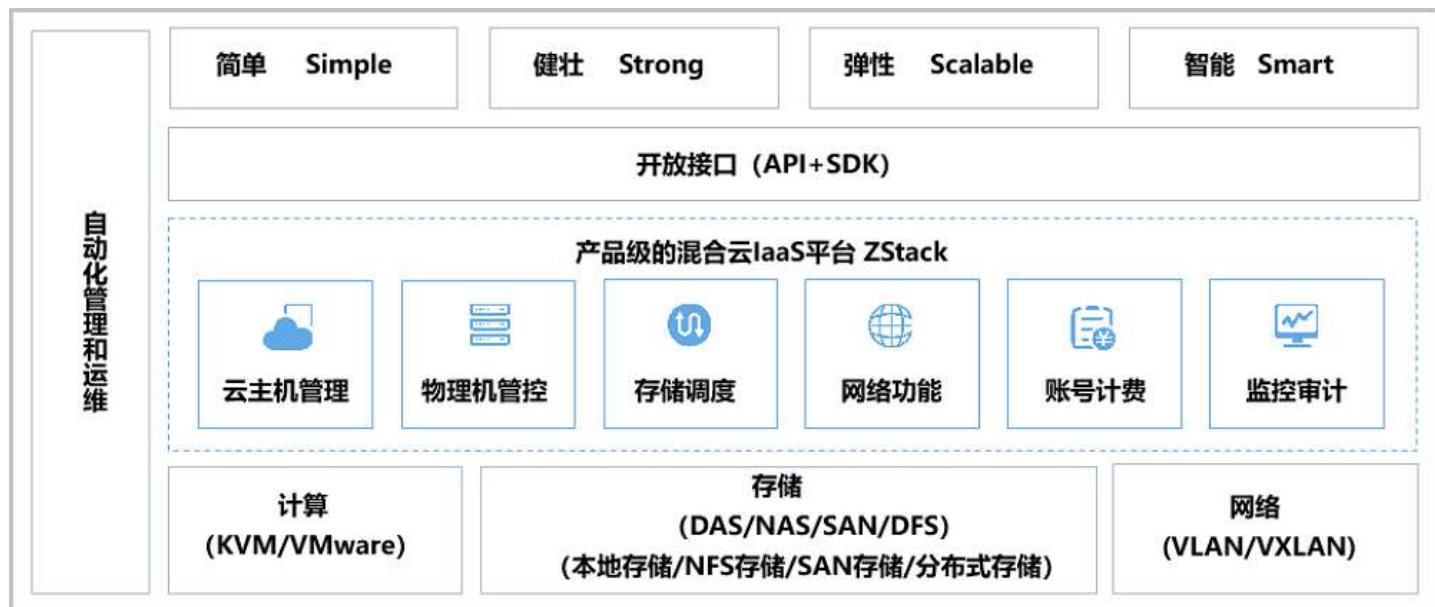
深度学习是机器学习和人工智能研究的热门分支，也是当今最流行的科学研究趋势之一。深度学习方法为计算机视觉、机器学习带来了革命性的进步，而新的深度学习技术也正在不断诞生。由于深度学习正快速发展，新的研究者很难对这一技术实时跟进。国内各大公有云厂商都提供了相应的深度学习相关产品，但对于初学者并不那么实用。本文将介绍基于产品化云平台——ZStack，来构建对初学者友好、易运维、易使用的深度学习云平台。

由于 ZStack 的轻量性，我们仅通过一台普通 PC 机就能部署云平台，进而实现深度学习平台构建。读者可结合本文轻松扩展出规模更大、功能更为完备的深度学习云平台。

1、ZStack 简介

ZStack是下一代开源的云计算IaaS（基础架构即服务）软件。它主要面向未来的智能数据中心，通过提灵活完善的APIs来管理包括计算、存储和网络在内的数据中心资源。用户可以利用ZStack快速构建自己的智能云数据中心，也可以在稳定的ZStack之上搭建灵活的云应用场景。

ZStack功能架构



ZStack产品优势：

ZStack是基于专有云平台4S（Simple简单，Strong健壮，Scalable弹性，Smart智能）标准设计的下一代云平台

IaaS软件。

1. 简单 (Simple)

- 简单安装部署：提供安装文件网络下载，30分钟完成从裸机到云平台的安装部署。
- 简单搭建云平台：支持云主机的批量（生成，删除等）操作，提供列表展示和滑窗详情。
- 简单实用操作：详细的用户手册，足量的帮助信息，良好的社区，标准的API提供。
- 友好UI交互：设计精良的专业操作界面，精简操作实现强大的功能。

2. 健壮 (Strong)

- 稳定且高效的系统架构设计：拥有全异步的后台架构，进程内微服务架构，无锁架构，无状态服务架构，一致性哈希环，保证系统架构的高效稳定。目前已实现：单管理节点管理上万台物理主机、数十万台云主机；而多个管理节点构建的集群使用一个数据库、一套消息总线可管理十万台物理主机、数百万台云主机、并发处理数万个API。
- 支撑高并发的API请求：单ZStack管理节点可以轻松处理每秒上万个并发API调用请求。
- 支持HA的严格要求：在网络或节点失效情况下，业务云主机可自动切换到其它健康节点运行；利用管理节点虚拟化实现了单管理节点的高可用，故障时支持管理节点动态迁移。

3. 弹性 (Scalable)

- 支撑规模无限制：单管理节点可管理从一台到上万台物理主机，数十万台云主机。
- 全API交付：ZStack提供了全套IaaS API，用户可使用这些APIs完成全新跨地域的可用区域搭建、网络配置变更、以及物理服务器的升级。
- 资源可按需调配：云主机和云存储等重要资源可根据用户需求进行扩缩容。ZStack不仅支持对云主机的CPU、内存等资源进行在线更改，还可对云主机的网络带宽、磁盘带宽等资源进行动态调整。

4. 智能 (Smart)

- 自动化运维管理：在ZStack环境里，一切由APIs来管理。ZStack利用Ansible库实现全自动部署和升级，自动探测和重连，在网络抖动或物理主机重启后能自动回连各节点。其中定时任务支持定时开关云主机以及定时对云主机快照等轮询操作。
- 在线无缝升级：5分钟一键无缝升级，用户只需升级管控节点。计算节点、存储节点、网络节点在管控软件启动后自动升级。
- 智能化的UI交互界面：实时的资源计算，避免用户误操作。

- 实时的全局监控：实时掌握整个云平台当前系统资源的消耗情况，通过实时监控，智能化调配，从而节省IT的软硬件资源。

0x2 构建深度学习平台

2.1 组件部署介绍

➤ TensorFlow

是一个开放源代码软件库，用于进行高性能数值计算。借助其灵活的架构，用户可以轻松地将计算工作部署到多种平台（CPU、GPU、TPU）和设备（桌面设备、服务器集群、移动设备、边缘设备等）。TensorFlow最初是由Google Brain 团队中的研究人员和工程师开发的，可为机器学习和深度学习提供强力支持，并且其灵活的数值计算核心广泛应用于许多其他科学领域。

➤ cuDNN

NVIDIA CUDA深层神经网络库（cuDNN）是一种用于深层神经网络的GPU加速库原始图形。cuDNN为标准例程提供了高度调优的实现，如前向和后向卷积、池化、归一化和激活层。cuDNN是NVIDIA深度学习SDK的一部分。

➤ TensorBoard

是一个可视化工具，能够有效地展示Tensorflow在运行过程中的计算图、各种指标随着时间的变化趋势以及训练中使用到的数据信息。

➤ Jupyter

Jupyter是一个交互式的笔记本，可以很方便地创建和共享文学化程序文档，支持实时代码，数学方程，可视化和 markdown。一般用与做数据清理和转换，数值模拟，统计建模，机器学习等等。

2.2 云平台环境准备

环境介绍

本次使用如下配置构建深度学习平台：

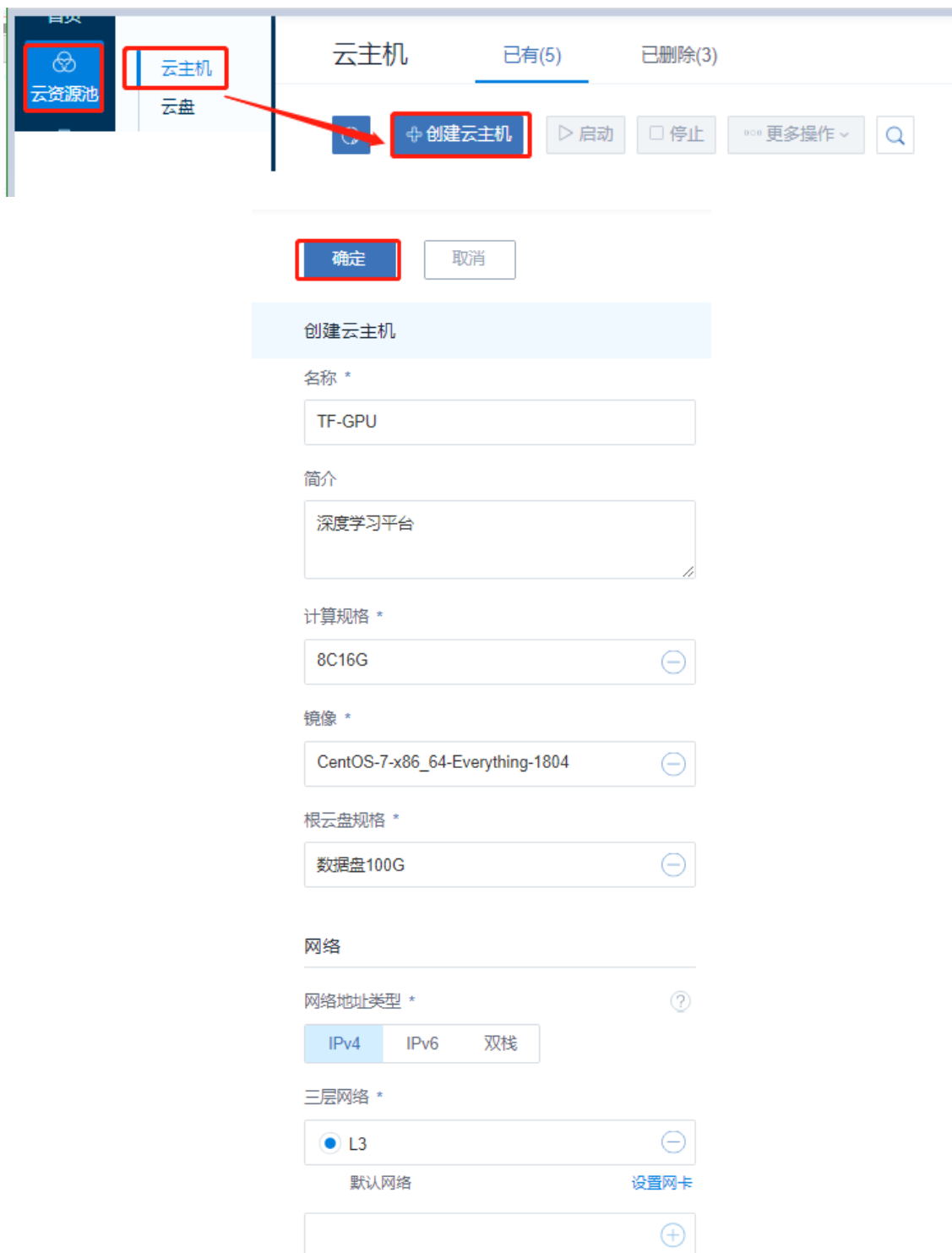
物理服务器配置	GPU型号	云主机配置	云主机系统	IP地址	主机名
Intel(R) i5-3470 DDR3 24G	NVIDIA QuadroP2000	8vCPU16G	CentOS7.4	192.168.66.6	GPU-TF

本次使用一台普通PC机部署ZStack云平台，使用云平台中GPU透传功能将一块NVIDIA QuadroP2000显卡透传给一个CentOS7.4虚拟机，进行平台的构建。

ZStack云平台部署步骤详情参考官方文档：

https://www.zstack.io/help/product_manuals/user_guide/3.html#c3

2.2.1 创建云主机



选择“云资源池”→点击“云主机”→点击“创建云主机按钮”打开云主机创建页面；

创建云主机的步骤：

- 1、选择添加方式：平台支持创建单个云主机和创建多个云主机，根据需求进行选择。
- 2、设置云主机名称：在设置名称时建议以业务系统名称进行命名，方便管理运维。
- 3、选择计算规格：根据定义的计算规格结合业务需求选择适合的计算规格。

- 4、选择镜像模板：根据业务需求选择相应的镜像模板。
- 5、选择三层网络：在新版本中平台三层网络同时支持 IPv4 和 IPv6，请根据自身业务需求进行选择；同时也可以
在创建云主机过程中设置网卡属性。
- 6、确认配置无误后点击“确定”开始创建。

2.2.2 透传 GPU 操作



点击云主机名称→点击配置信息；



找到 GPU 设备标签，点击操作→选择加载，然后选择相应的 GPU 设备给云主机直接使用。

0x3 开始部署

3.1 运行环境准备

```

安装pip

# curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

# python get-pip.py

# pip --version

pip 18.1 from /usr/lib/python2.7/site-packages/pip (python 2.7)
    
```

```
# python --version

Python 2.7.5

安装GCC G++

# yum install gcc gcc-c++

# gcc --version

gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)

安装一些需要的包

#yum -y install zlib*

#yum install openssl-devel -y

#yum install sqlite* -y

升级CentOS默认Python2.7.5版本到3.6.5

下载Python源码包

# wget -c https://www.python.org/ftp/python/3.6.5/Python-3.6.5.tgz

解压源码包

# tar -zvxf Python-3.6.5.tgz

进入源码目录

# cd Python-3.6.5/

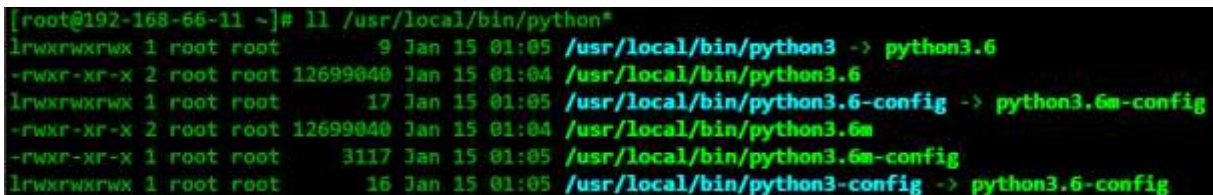
# ./configure --with-ssl

编译并安装

# make && make install

查看一下新安装的python3的文件位置

# ll /usr/local/bin/python*
```



```
[root@192-168-66-11 ~]# ll /usr/local/bin/python*
lrwxrwxrwx 1 root root      9 Jan 15 01:05 /usr/local/bin/python3 -> python3.6
-rwxr-xr-x 2 root root 12699040 Jan 15 01:04 /usr/local/bin/python3.6
lrwxrwxrwx 1 root root     17 Jan 15 01:05 /usr/local/bin/python3.6-config -> python3.6m-config
-rwxr-xr-x 2 root root 12699040 Jan 15 01:04 /usr/local/bin/python3.6m
-rwxr-xr-x 1 root root    3117 Jan 15 01:05 /usr/local/bin/python3.6m-config
lrwxrwxrwx 1 root root     16 Jan 15 01:05 /usr/local/bin/python3-config -> python3.6-config
```

设置python默认版本号为3.x

```
# mv /usr/bin/python /usr/bin/python.bak

# ln -s /usr/local/bin/python3 /usr/bin/python
```

查看一下2.x版本的文件位置


```
# ll /usr/bin/python*
```

```
[root@192-168-66-11 ~]# ll /usr/bin/python*
lrwxrwxrwx. 1 root root 22 Jan 15 01:06 /usr/bin/python -> /usr/local/bin/python3
lrwxrwxrwx. 1 root root 9 Dec 1 04:04 /usr/bin/python2 -> python2.7
-rwxr-xr-x. 1 root root 7216 Apr 11 2018 /usr/bin/python2.7
lrwxrwxrwx. 1 root root 7 Dec 1 04:04 /usr/bin/python.bak -> python2
```

为使yum命令正常使用，需要将其配置的python依然指向2.x版本

```
# vim /usr/bin/yum
```

```
#vim /usr/libexec/urlgrabber-ext-down
```

将上面两个文件的头部文件修改为老版本即可

```
!/usr/bin/python --> !/usr/bin/python2.7
```

安装python-dev、python-pip

```
# yum install python-dev python-pip -y
```

禁用自带Nouveau驱动

Nouveau使用

```
# lsmod | grep nouveau
```

```
nouveau                1662531  0
mxm_wmi                  13021  1 nouveau
wmi                       19086  2 mxm_wmi,nouveau
video                     24538  1 nouveau
i2c_algo_bit             13413  1 nouveau
drm_kms_helper           176920  2 qxl,nouveau
ttm                       99555  2 qxl,nouveau
drm                       397988  5 qxl,ttm,drm_kms_helper,nouveau
i2c_core                  63151  5 drm,i2c_piix4,drm_kms_helper,i2c_algo_bit,nouveau
```

```
#vim /usr/lib/modprobe.d/dist-blacklist.conf
```

```
# nouveau
```

```
blacklist nouveau
```

```
options nouveau modeset=0
```

```
:wq 保存退出
```

```
# mv /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).img.bak 备份引导镜像
```

```
# dracut /boot/initramfs-$(uname -r).img $(uname -r) 重建引导镜像
```

```
# reboot  
  
#lsmod | grep nouveau 再次验证禁用是否生效
```

3.2 安装CUDA

升级内核:

```
# rpm -import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org  
  
# rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm  
  
# yum -y --enablerepo=elrepo-kernel install kernel-ml.x86_64 kernel-ml-devel.x86_64
```

查看内核版本默认启动顺序:

```
awk -F\ ' $1=="menuentry " {print $2}' /etc/grub2.cfg
```

```
CentOS Linux (4.20.0-1.el7.elrepo.x86_64) 7 (Core)
```

```
CentOS Linux (3.10.0-862.el7.x86_64) 7 (Core)
```

```
CentOS Linux (0-rescue-c4581dac5b734c11a1881c8eb10d6b09) 7 (Core)
```

```
#vim /etc/default/grub
```

```
GRUB_DEFAULT=saved 改为GRUB_0=saved
```

运行grub2-mkconfig命令来重新创建内核配置

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

```
#reboot
```

```
# uname -r 重启后验证一下内核版本
```

```
4.20.0-1.el7.elrepo.x86_64
```

CUDA Toolkit安装有两种方式:

- Package安装 (RPM and Deb packages)
- Runfile安装

这里选择使用Runfile模式进行安装

安装包下载:

https://developer.nvidia.com/compute/cuda/10.0/Prod/local_installers/cuda_10.0.130_410.48_linux

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System: Windows, **Linux**, Mac OS X

Architecture: **x86_64**, ppc64le

Distribution: Fedora, OpenSUSE, RHEL, **CentOS**, SLES, Ubuntu

Version: **7**, 8

Installer Type: **runfile (local)**, rpm (local), rpm (network)

Download Installer for Linux CentOS 7 x86_64

The base installer is available for download below.

> Base Installer Download (2.0 GB) 

根据自身操作系统进行安装包筛选，并下载。复制下载链接直接用 `wget -c` 命令进行下载

```
# wget -c
```

```
https://developer.nvidia.com/compute/cuda/10.0/Prod/local\_installers/cuda\_10.0.130\_410.48\_linux
```

```
#chmod +x cuda_10.0.130_410.48_linux
```

```
#. /cuda_10.0.130_410.48_linux
```

```
Do you accept the previously read EULA?
```

```
accept/decline/quit: accept
```

```
Install NVIDIA Accelerated Graphics Driver for Linux-x86_64 410.48?
```

```
(y)es/(n)o/(q)uit: y
```

```
Install the CUDA 10.0 Toolkit?
```

```
(y)es/(n)o/(q)uit: y
```

```
Enter Toolkit Location
```

```
[ default is /usr/local/cuda-10.0 ]:
```

```
Do you want to install a symbolic link at /usr/local/cuda?
```

```
(y)es/(n)o/(q)uit: y
```

```
Install the CUDA 10.0 Samples?
```

```
(y)es/(n)o/(q)uit: y
```

```
Enter CUDA Samples Location
```

```
[ default is /root ]:
```

配置CUDA运行环境变量:

```
# vim /etc/profile

# CUDA

export PATH=/usr/local/cuda-10.0/bin${PATH:+:${PATH}}

export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}

# source /etc/profile
```

检查版本

```
# nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2018 NVIDIA Corporation

Built on Sat_Aug_25_21:08:01_CDT_2018
```

Cuda compilation tools, release 10.0, V10.0.130

使用实例验证测试CUDA是否正常:

```
#cd /root/NVIDIA_CUDA-10.0_Samples/1_Uutilities/deviceQuery

# make

"/usr/local/cuda-10.0"/bin/nvcc -cbin g++ -I../common/inc -m64 -gencode
arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode
arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode
arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60 -gencode
arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70 -gencode
arch=compute_75,code=sm_75 -gencode arch=compute_75,code=compute_75 -o deviceQuery.o -c
deviceQuery.cpp

"/usr/local/cuda-10.0"/bin/nvcc -cbin g++ -m64 -gencode arch=compute_30,code=sm_30 -
gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37 -gencode
arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode
arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm_61 -gencode
arch=compute_70,code=sm_70 -gencode arch=compute_75,code=sm_75 -gencode
arch=compute_75,code=compute_75 -o deviceQuery deviceQuery.o

mkdir -p ../../bin/x86_64/linux/release
```

```
cp deviceQuery ../../bin/x86_64/linux/release
```

```
# cd ../../bin/x86_64/linux/release/
```

```
# ./deviceQuery
```

```
#. /deviceQuery Starting...
```

CUDA Device Query (Runtime API) version (CUDA static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Quadro P2000"

CUDA Driver Version / Runtime Version 10.0 / 10.0

CUDA Capability Major/Minor version number: 6.1

Total amount of global memory: 5059 MBytes (5304745984 bytes)

(8) Multiprocessors, (128) CUDA Cores/MP: 1024 CUDA Cores

GPU Max Clock rate: 1481 MHz (1.48 GHz)

Memory Clock rate: 3504 Mhz

Memory Bus Width: 160-bit

L2 Cache Size: 1310720 bytes

Maximum Texture Dimension Size (x,y,z) 1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)

Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers

Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers

Total amount of constant memory: 65536 bytes

Total amount of shared memory per block: 49152 bytes

Total number of registers available per block: 65536

Warp size: 32

Maximum number of threads per multiprocessor: 2048

Maximum number of threads per block: 1024

Max dimension size of a thread block (x,y,z): (1024, 1024, 64)

Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)

Maximum memory pitch:	2147483647 bytes
Texture alignment:	512 bytes
Concurrent copy and kernel execution:	Yes with 2 copy engine(s)
Run time limit on kernels:	No
Integrated GPU sharing Host Memory:	No
Support host page-locked memory mapping:	Yes
Alignment requirement for Surfaces:	Yes
Device has ECC support:	Disabled
Device supports Unified Addressing (UVA):	Yes
Device supports Compute Preemption:	Yes
Supports Cooperative Kernel Launch:	Yes
Supports MultiDevice Co-op Kernel Launch:	Yes
Device PCI Domain ID / Bus ID / location ID:	0 / 0 / 11

Compute Mode:

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.0, CUDA Runtime Version = 10.0,

NumDevs = 1

Result = PASS

Result = PASS且测试过程中无报错，表示测试通过！

3.3 安装 cuDNN

cuDNN的全称为NVIDIA CUDA® Deep Neural Network library，是NVIDIA专门针对深度神经网络（Deep Neural Networks）中的基础操作而设计基于GPU的加速库。cuDNN为深度神经网络中的标准流程提供了高度优化的实现方式。

下载安装包：<https://developer.nvidia.com/rdp/cudnn-download>

注：下载前需先注册 NVIDIA Developer Program，然后才能下载。

Library for Windows, Mac, Linux, Ubuntu and RedHat/Centos(x86_64 architecture)

- [cuDNN Library for Windows 7](#)
- [cuDNN Library for Windows 10](#)
- [cuDNN Library for Linux](#)
- [cuDNN Library for OSX](#)
- [cuDNN Runtime Library for Ubuntu18.04 \(Deb\)](#)
- [cuDNN Developer Library for Ubuntu18.04 \(Deb\)](#)
- [cuDNN Code Samples and User Guide for Ubuntu18.04 \(Deb\)](#)
- [cuDNN Runtime Library for Ubuntu16.04 \(Deb\)](#)
- [cuDNN Developer Library for Ubuntu16.04 \(Deb\)](#)
- [cuDNN Code Samples and User Guide for Ubuntu16.04 \(Deb\)](#)
- [cuDNN Runtime Library for Ubuntu14.04 \(Deb\)](#)
- [cuDNN Developer Library for Ubuntu14.04 \(Deb\)](#)
- [cuDNN Code Samples and User Guide for Ubuntu14.04 \(Deb\)](#)
- [cuDNN Runtime Library for RedHat/Centos 7.3 \(RPM\)](#)
- [cuDNN Developer Library for RedHat/Centos 7.3 \(RPM\)](#)
- [cuDNN Code Samples and User Guide for RedHat/Centos 7.3 \(RPM\)](#)

可以根据自身的环境选择相应版本进行下载，这个有身份验证只能浏览器下载然后再上传到云主机中。

安装:

```
#rpm -ivh libcudnn7-7.4.2.24-1.cuda10.0.x86_64.rpm libcudnn7-devel-7.4.2.24-1.cuda10.0.x86_64.rpm
libcudnn7-doc-7.4.2.24-1.cuda10.0.x86_64.rpm
```

```
准备中... ##### [100%]
```

```
正在升级/安装...
```

- ```
1:libcudnn7-7.4.2.24-1.cuda10.0 ##### [33%]
2:libcudnn7-devel-7.4.2.24-1.cuda10.0##### [67%]
3:libcudnn7-doc-7.4.2.24-1.cuda10.0##### [100%]
```

#### 验证cuDNN:

```
cp -r /usr/src/cudnn_samples_v7/ $HOME
cd $HOME/cudnn_samples_v7/mnistCUDNN
make clean && make
rm -rf *o
rm -rf mnistCUDNN
/usr/local/cuda/bin/nvcc -ccbin g++ -I/usr/local/cuda/include -IFreeImage/include -m64 -
gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode
arch=compute_50,code=sm_50 -gencode arch=compute_53,code=sm_53 -gencode
arch=compute_53,code=compute_53 -o fp16_dev.o -c fp16_dev.cu
g++ -I/usr/local/cuda/include -IFreeImage/include -o fp16_emu.o -c fp16_emu.cpp
g++ -I/usr/local/cuda/include -IFreeImage/include -o mnistCUDNN.o -c mnistCUDNN.cpp
```

```
/usr/local/cuda/bin/nvcc -ccbin g++ -m64 -gencode arch=compute_30,code=sm_30 -gencode
arch=compute_35,code=sm_35 -gencode arch=compute_50,code=sm_50 -gencode arch=compute_53,code=sm_53 -
gencode arch=compute_53,code=compute_53 -o mnistCUDA fp16_dev.o fp16_emu.o mnistCUDA.o -
I/usr/local/cuda/include -IFreeImage/include -LFreeImage/lib/linux/x86_64 -LFreeImage/lib/linux -
lcudart -lcublas -lcudnn -lfreeimage -lstdc++ -lm

./mnistCUDA

cudnnGetVersion() : 7402 , CUDNN_VERSION from cudnn.h : 7402 (7.4.2)

Host compiler version : GCC 4.8.5

There are 1 CUDA capable devices on your machine :

device 0 : sms 8 Capabilities 6.1, SmClock 1480.5 Mhz, MemSize (Mb) 5059, MemClock 3504.0 Mhz,
Ecc=0, boardGroupID=0

Using device 0

Testing single precision
Loading image data/one_28x28.pgm
Performing forward propagation ...
Testing cudnnGetConvolutionForwardAlgorithm ...
Fastest algorithm is Algo 1
Testing cudnnFindConvolutionForwardAlgorithm ...
^^^^ CUDNN_STATUS_SUCCESS for Algo 0: 0.036864 time requiring 0 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 1: 0.044032 time requiring 3464 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 2: 0.053248 time requiring 57600 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 4: 0.116544 time requiring 207360 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 7: 0.181248 time requiring 2057744 memory

Resulting weights from Softmax:

0.000000 0.9999399 0.000000 0.000000 0.0000561 0.000000 0.0000012 0.0000017 0.0000010
0.000000

Loading image data/three_28x28.pgm
```



```
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000000 0.0000000 0.9999288 0.0000000 0.0000711 0.0000000 0.0000000 0.0000000
0.0000000
Loading image data/five_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000008 0.0000000 0.0000002 0.0000000 0.9999820 0.0000154 0.0000000 0.0000012
0.0000006
Result of classification: 1 3 5
Test passed!
Testing half precision (math in single precision)
Loading image data/one_28x28.pgm
Performing forward propagation ...
Testing cudnnGetConvolutionForwardAlgorithm ...
Fastest algorithm is Algo 1
Testing cudnnFindConvolutionForwardAlgorithm ...
^^^^ CUDNN_STATUS_SUCCESS for Algo 0: 0.032896 time requiring 0 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 1: 0.036448 time requiring 3464 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 2: 0.044000 time requiring 28800 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 4: 0.115488 time requiring 207360 memory
^^^^ CUDNN_STATUS_SUCCESS for Algo 7: 0.180224 time requiring 2057744 memory
Resulting weights from Softmax:
0.0000001 1.0000000 0.0000001 0.0000000 0.0000563 0.0000001 0.0000012 0.0000017 0.0000010
0.0000001
Loading image data/three_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000714 0.0000000 0.0000000 0.0000000
0.0000000
```

```
Loading image data/five_28x28.pgm
Performing forward propagation ...
Resulting weights from Softmax:
0.0000000 0.0000008 0.0000000 0.0000002 0.0000000 1.0000000 0.0000154 0.0000000 0.0000012
0.0000006
Result of classification: 1 3 5
Test passed!
Test passed!且测试过程中无报错，表示测试通过!
```

### 3.4 安装 TensorFlow

```
pip3 install --upgrade setuptools==30.1.0
```

```
pip3 install tf-nightly-gpu
```

验证测试:

在 Python 交互式 shell 中输入以下几行简短的程序代码:

```
python
```

```
import tensorflow as tf
```

```
hello = tf.constant('Hello, TensorFlow!')
```

```
sess = tf.Session()
```

```
print(sess.run(hello))
```

如果系统输出以下内容，就说明您可以开始编写 TensorFlow 程序了:

```
Hello, TensorFlow!
```

同时使用nvidia-smi命令可以看到当前显卡的处理任务。

```
[root@GPU-TF ~]# nvidia-smi
Tue Jan 8 12:11:49 2019

+-----+
| NVIDIA-SMI 410.93 Driver Version: 410.93 CUDA Version: 10.0 |
+-----+-----+
| GPU Name Persistence-MI Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
+-----+-----+-----+-----+-----+-----+
| 0 Quadro P2000 Off 00000000:00:0B.0 Off | 0% N/A |
| 44% 28C P8 5W / 75W | 63MiB / 5059MiB | 0% Default |
+-----+-----+-----+-----+-----+-----+

+-----+
| Processes: GPU Memory |
| GPU PID Type Process name Usage |
+-----+-----+-----+-----+-----+
| 0 99071 C python 53MiB |
+-----+-----+-----+-----+-----+-----+-----+

```

### 3.5 安装 TensorBoard 可视化工具

可以用 TensorBoard 来展现 TensorFlow 图，绘制图像生成的定量指标图以及显示附加数据（如其中传递的图像）。通过 pip 安装 TensorFlow 时，也会自动安装 TensorBoard：

验证版本：

```
pip3 show tensorboard
```

Name: tensorboard

Version: 1.12.2

Summary: TensorBoard lets you watch Tensors Flow

Home-page: <https://github.com/tensorflow/tensorboard>

Author: Google Inc.

Author-email: [opensource@google.com](mailto:opensource@google.com)

License: Apache 2.0

Location: /usr/lib/python2.7/site-packages

Requires: protobuf, numpy, futures, grpcio, wheel, markdown, werkzeug, six

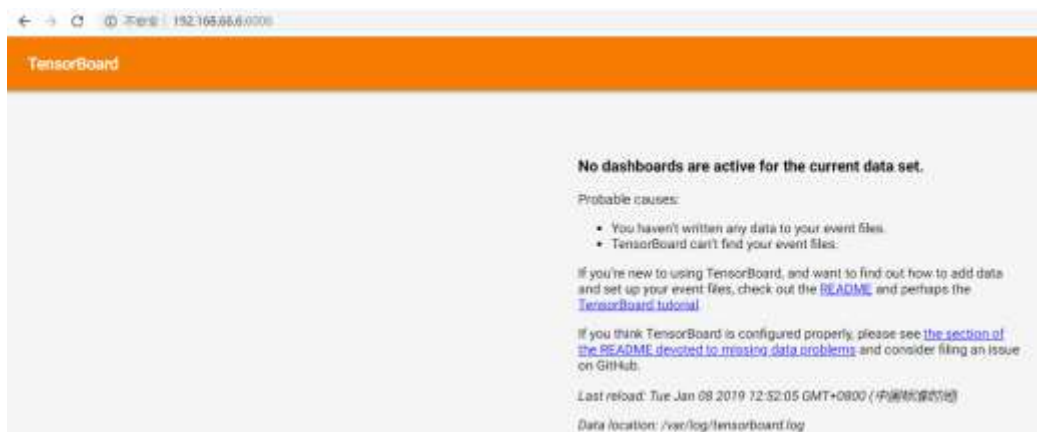
Required-by:

启动服务：

```
tensorboard --logdir /var/log/tensorboard.log
```

TensorBoard 1.13.0a20190107 at <http://GPU-TF:6006> (Press CTRL+C to quit)

根据提示在浏览器上输入<http://服务器IP:6006>



## 3.6 安装 Jupyter

Jupyter是一个交互式的笔记本，可以很方便地创建和共享文学化程序文档，支持实时代码，数学方程，可视化和markdown。一般用与做数据清理和转换，数值模拟，统计建模，机器学习等等。

安装：

```
sudo pip3 install jupyter
```

生成配置文件：

```
jupyter notebook --generate-config
```

Writing default config to: /root/.jupyter/jupyter\_notebook\_config.py

生成 Jupyter 密码：

```
python
```

```
Python 3.6.5 (default, Jan 15 2019, 02:51:51)
```

```
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> from notebook.auth import passwd;
```

```
>>> passwd()
```

```
Enter password:
```

```
Verify password:
```

```
'sha1:6067bcf7350b:8407670bb3f94487c9404ed3c20c1ebf7ddee32e'
```

```
>>>
```

将生成的 hash 串写入 Jupyter 配置文件：

```
vim /root/.jupyter/jupyter_notebook_config.py
```

```
Hashed password to use for web authentication.
#
To generate, type in a python/IPython shell:
#
from notebook.auth import passwd; passwd()
#
The string should be of the form type:salt:hashed-password.
#c.NotebookApp.password = u''
c.NotebookApp.password = 'sha1:60f11cbc4601:41adfd204e194ed6b1c72f98faff065593581b18'
```

启动服务

```
jupyter notebook --allow-root --ip='192.168.66.11'
```

浏览器登陆



输入密码后登陆：即可正常访问

执行测试任务：

**运行 TensorFlow Demo 示例**

Jupyter 中新建 HelloWorld 示例，代码如下：

```
import tensorflow as tf

Simple hello world using TensorFlow

Create a Constant op

The op is added as a node to the default graph.

#

The value returned by the constructor represents the output
of the Constant op.

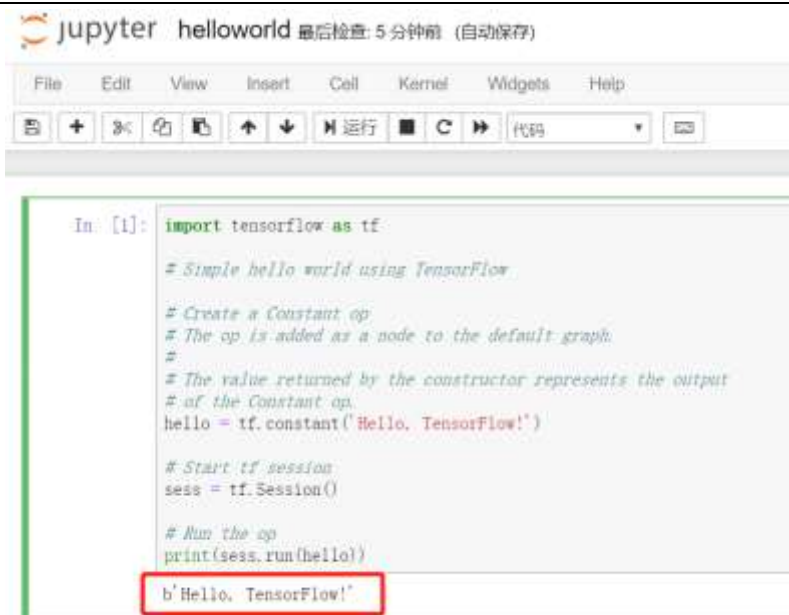
hello = tf.constant('Hello, TensorFlow!')

Start tf session

sess = tf.Session()

Run the op

print(sess.run(hello))
```



```

jupyter helloworld 最后检查: 5分钟前 (自动保存)
File Edit View Insert Cell Kernel Widgets Help
运行 代码
In [1]: import tensorflow as tf
 # Simple hello world using TensorFlow
 # Create a Constant op
 # The op is added as a node to the default graph.
 # The value returned by the constructor represents the output
 # of the Constant op.
 hello = tf.constant('Hello, TensorFlow!')
 # Start tf session
 sess = tf.Session()
 # Run the op
 print(sess.run(hello))
 b'Hello, TensorFlow!'

```

## 0x4 总结

通过使用 ZStack 云平台可以快速构建深度学习平台，云平台自身无需太多的复杂配置，在安装各种驱动及深度学习组件时也与物理机无异。安装好驱动后进行性能测试发现与同配置物理逻辑性能相当，GPU 部分没有任何性能损失。

当上述软件环境都准备完成以后，可配置多块 GPU 并从模板创建多个云主机一一透传，结合 ZStack 本身的多租户属性，可使得多人在同一套环境中互不影响进行开发或者运行应用程序，从而成为一个真正的深度学习“云”平台。