

ZStack 技术白皮书精选

如何基于国产 CPU 的云平台

构建容器管理平台？

扫一扫二维码，获取更多技术干货吧



 ZStack中国社区@二群
扫一扫二维码，加入群聊。



长按扫码，关注ZStack官微

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。

目录

第一节 基于国产 CPU 的服务器.....	3
第二节 国产云平台.....	7
1、安装云平台.....	10
1.1 启动 ARM 服务器，从 U 盘启动.....	10
1.2 ARM 服务器 BIOS 基本设置.....	11
第三节 基于 ZStack 云主机构建 K8S 集群.....	19
1、准备工作.....	21
2、安装部署.....	22
2.1 安装 Docker.....	22
2.3 安装 kubelet、kubeadm、kubectl.....	23
2.4 用 kubeadm 创建集群.....	23
2.5 配置 kubectl.....	23
2.6 安装 Pod 网络.....	24
2.7 注册节点到 K8S 集群.....	32
2.8 部署 kubernetes-dashboard.....	35
第四节 全篇总结.....	47

如何基于国产 CPU 的云平台构建容器管理平台？

随着“中兴事件”不断升级，引起了国人对国产自主可控技术的高度关注；本人作为所在单位的运维工程师，也希望能找到一个稳定、能兼容国产 CPU 的一整套架构方案，来构建 IaaS 平台和 PaaS 平台，满足单位对安全自主可控的需求。要基于全国产方式解决公司业务需求至少要在软硬件层面满足，而国内基本都是基于 x86 解决方案，想找到满足需求的国产化解决方案还是非常困难的事情。但笔者由于一个偶然的的机会，接触到了国产的芯片厂商和云计算厂商，并得知他们已经实现了全国产化的云计算平台，笔者也亲自动手体验了安装部署该云计算平台，并在其之上安装部署了容器平台，以下是笔者的分享。

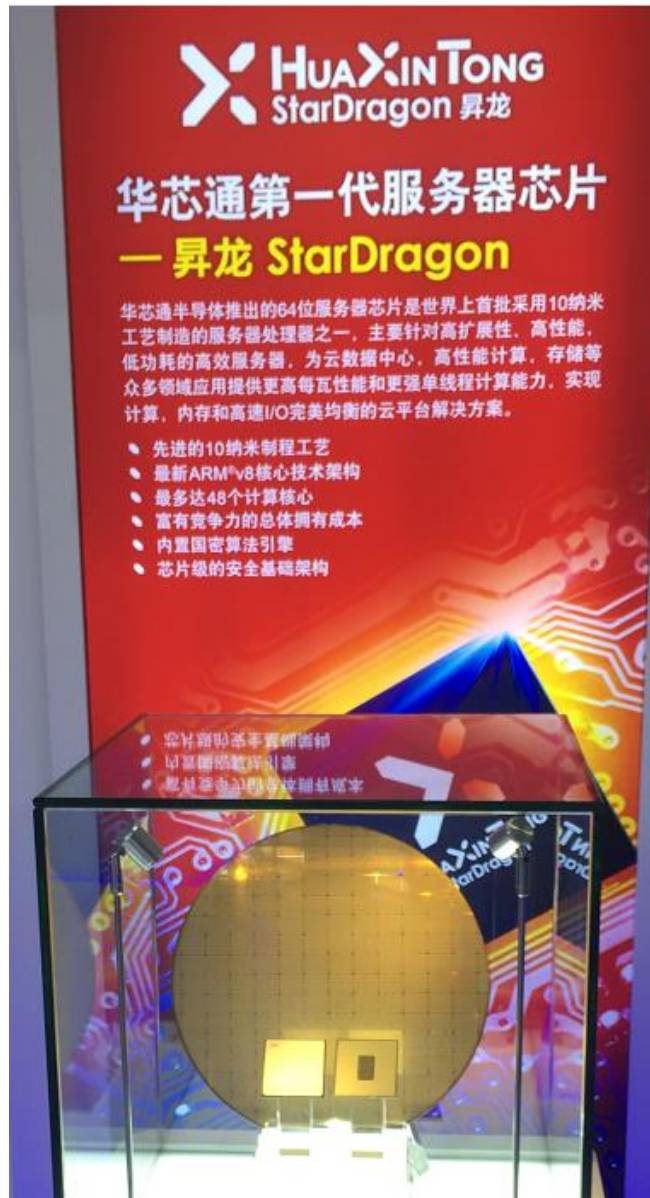
第一节 基于国产 CPU 的服务器

纵观国内能用于商用国产 CPU 服务器也没几家真实能用的；有的是基于 3B1500 国产商用 28 纳米 8 核处理最高主频达 1.5GHz；通过多方查阅相关资料目前性能无法满足云平台需求，而且还不支持虚拟化。

一个偶然机会参加 2018 年贵州大数据博览会，参会过程中发现一个有意思的事情，就是在阿里云展台看到国产云平台+国产芯片宣传字样。



于是上前跟现场的工作人员进行简单的沟通，了解到国产 CPU 是由华芯通设计开发，这颗芯片内置 48 颗物理核心，单核心 2.6GHz，64Bit、支持虚拟化！没想到这颗 CPU 居然支持虚拟化，看来距离我的想法又进一步，起码已经有硬件可以实现了。还了解到目前已经有国产云平台具备商用环境；名字叫 ZStack for Alibaba Cloud，据工作人员介绍目前已有业务系统运行在基于华芯通 CPU 的云平台上，云平台就是 ZStack。热心的工作人员带我去华芯通的专柜进行详细参观。





看到实物那一刻，我发现这个跟 x86 架构的服务器区别并不大，之前一直以为它是一个类似路由器这样的小盒子。没想到 ARM 服务器工艺已和 x86 服务器自造工艺无太大差别。

第二节 国产云平台

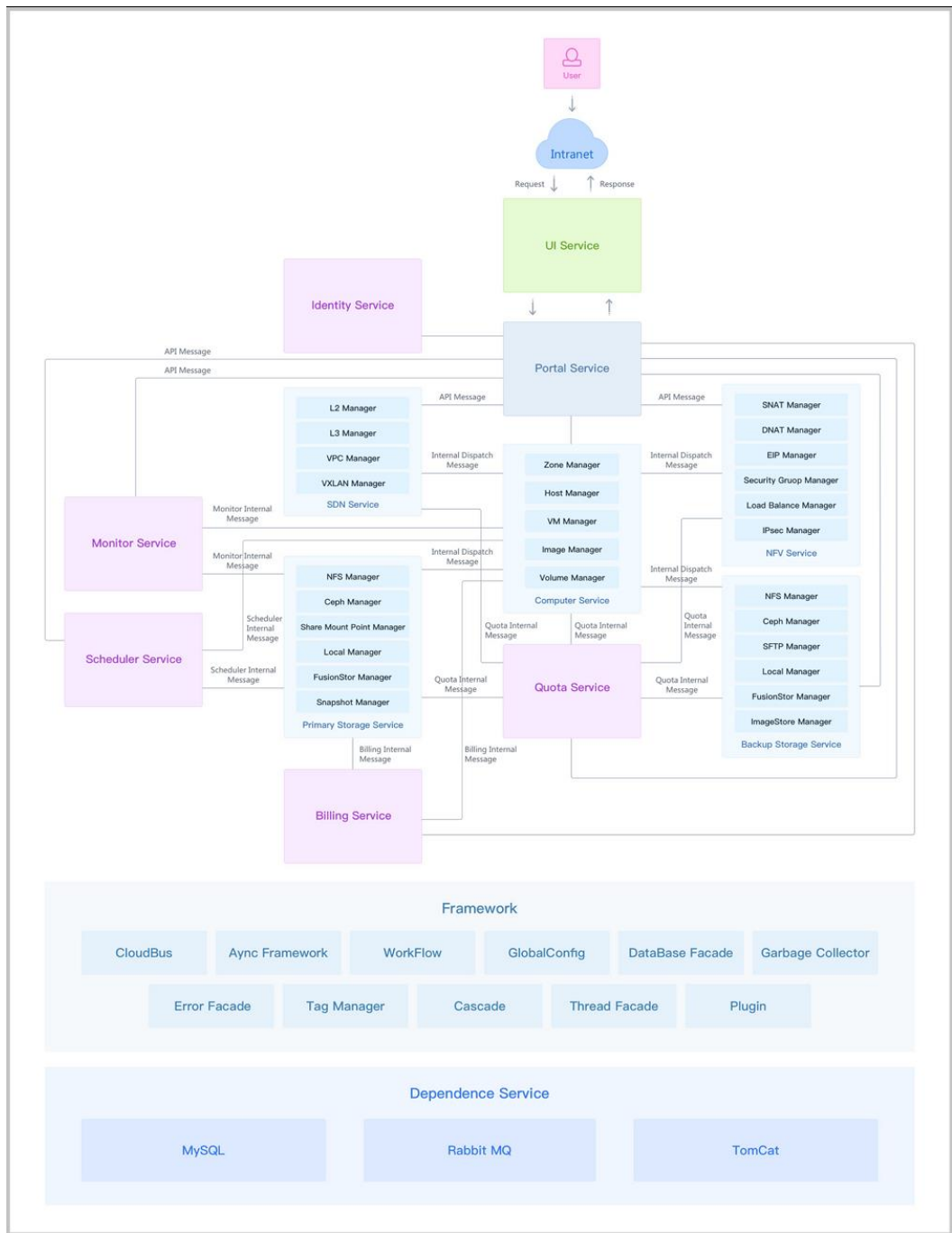
ZStack 作为国内为数不多的自研云平台，根据官网信息已发布基于国产 CPU 架构的版本，那么完全可以实现基于国产 CPU 架构来构建国产云平台。



ZStack 2.4.0

- 重磅推出企业管理模块
- 正式发布ARM集群支持**
- 应用中心可自由添加第三方应用
- 自研shared block存储层极致发挥SAN存储性能

ZStack 架构：



这架构图摘自他们的产品白皮书，从架构上看整个逻辑还是比较清晰，各组件依赖度并不高，不会因为管理控制节点故障而影响业务系统。经过仔细研究 ZStack 架构发现以下特点：

- 全异步架构：异步消息、异步方法、异步 HTTP 调用

- ✚ 无状态服务：单次请求不依赖其他请求
- ✚ 无锁架构：一致性哈希算法。
- ✚ 进程内微服务：微服务解耦。

再看看 ZStack 的功能架构图：



从图里可以发现，服务之间的交互统一走消息队列，整个拓扑结构不再紧密，实现星状的架构，各服务之间只有消息的交互，服务之间基本独立，添加或者删除某个服务不会影响整个架构（只会失去某些功能）。

回到文章的主题上，了解到以上信息后，我们决定使用华芯通 CPU+ZStack 国产化云平台来实现容器平台管理方案敲定后，接下来就是走借测流程。

通过之前展会联系的华芯通负责人帮忙，在等了 2、3 个星期之后，机器寄到了单位。



上图是他们的工程机，但做工已经非常精细，完全不输给主流大厂的 X86 服务器。接下来先部署云平台，之前提到的 ZStack 是国产化云计算平台的先行者，核心引擎也是完全开源的，笔者通过 ZStack 的官方网站 (<http://www.zstack.io/product/enterprise/>)，下载了他们的 iso 系统，并根据用户手册的图文教程做了烧录，不得不说，整个文档做的非常清晰，很快就完成了准备工作，下面就按照文档进入安装过程。

3、安装云平台

3.1 启动 ARM 服务器，从 U 盘启动

通过 Console 连接看到如下一些信息，这是 ARM 服务器在进行自检。

```
B - 10728436 - Remove Fence: APB and dfi_init_complete inputs from the DDR PHY
B - 10738562 - Assert MDDR PwrOkIn signal
B - 10743472 - Remove Fence: APB and dfi_init_complete inputs from the DDR PHY
B - 10753598 - Assert MDDR PwrOkIn signal
B - 10758509 - Remove Fence: APB and dfi_init_complete inputs from the DDR PHY
B - 10768635 - Assert MDDR PwrOkIn signal
B - 10773545 - Remove Fence: APB and dfi_init_complete inputs from the DDR PHY
B - 10783671 - Assert MDDR PwrOkIn signal
B - 10788582 - Remove Fence: APB and dfi_init_complete inputs from the DDR PHY
B - 10795688 - Setting DDR clock to 1333 MHz
```

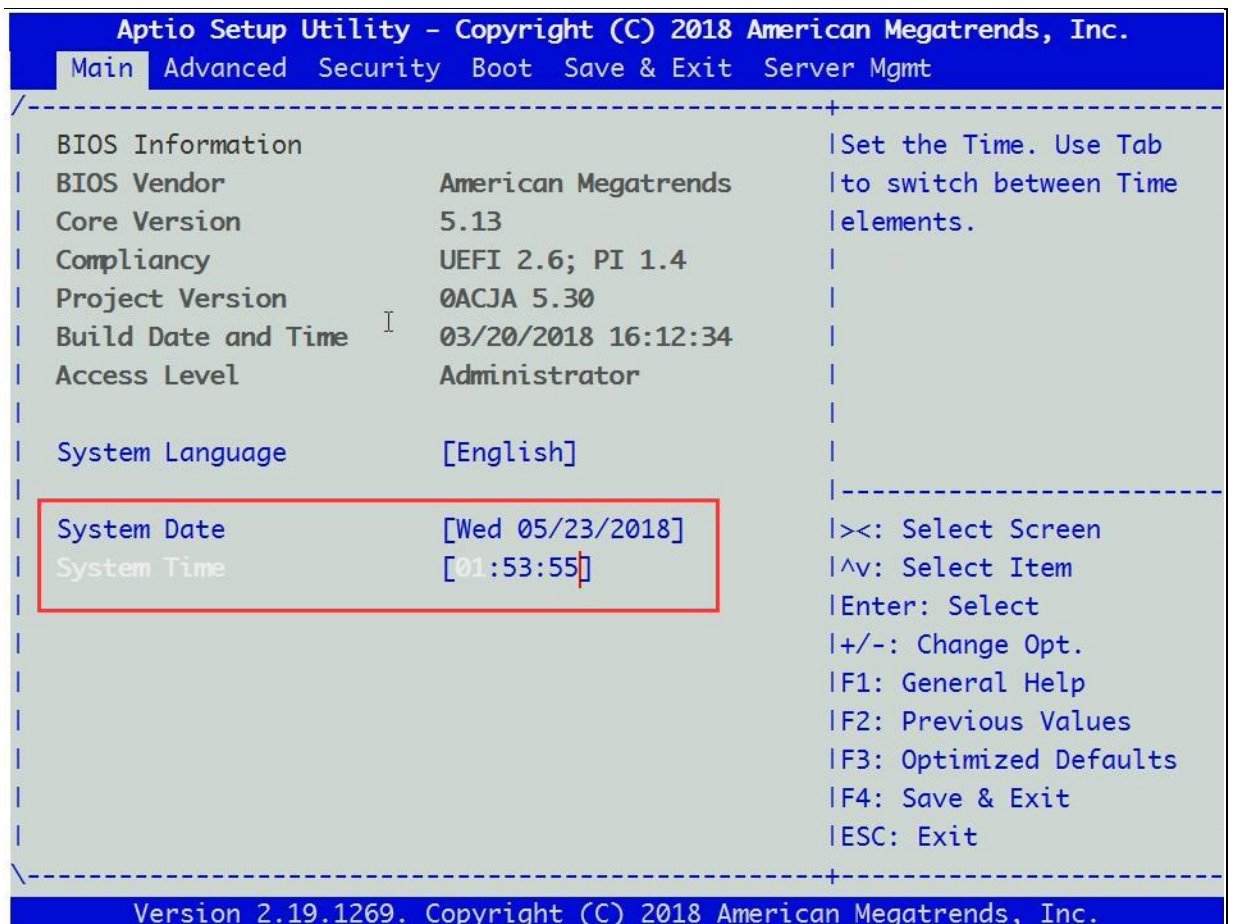
直到出现以下信息：

```
Version 2.19.1269. Copyright (C) 2018 American Megatrends, Inc.  
BIOS Date: 03/20/2018 16:12:34 Ver: 0ACJA530  
EVALUATION COPY.  
Press <DEL> or <ESC> to enter setup.
```

按 Delete 或者 ESC 键进入 BIOS 设置。

3.2 ARM 服务器 BIOS 基本设置

3.2.1 修改时间

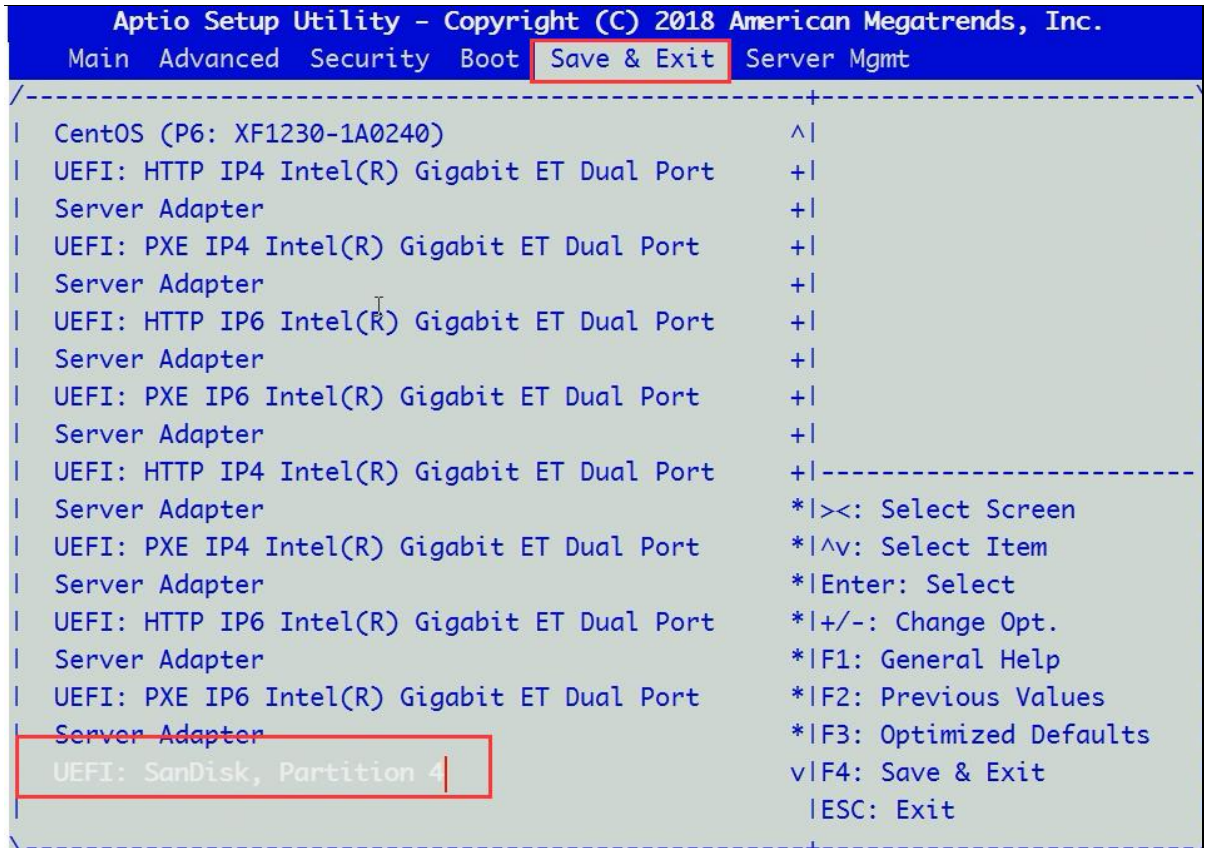


The screenshot shows the Aptio Setup Utility interface. The title bar reads "Aptio Setup Utility - Copyright (C) 2018 American Megatrends, Inc." with tabs for "Main", "Advanced", "Security", "Boot", "Save & Exit", and "Server Mgmt". The "Main" tab is selected. The interface is divided into two columns. The left column lists system information and settings, with "System Date" and "System Time" highlighted by a red box. The right column provides instructions and navigation options. The bottom status bar repeats the version and copyright information.

Aptio Setup Utility - Copyright (C) 2018 American Megatrends, Inc.					
Main	Advanced	Security	Boot	Save & Exit	Server Mgmt
BIOS Information					Set the Time. Use Tab to switch between Time elements.
BIOS Vendor	American Megatrends				
Core Version	5.13				
Compliance	UEFI 2.6; PI 1.4				
Project Version	0ACJA 5.30				
Build Date and Time	03/20/2018 16:12:34				
Access Level	Administrator				
System Language	[English]				
System Date	[Wed 05/23/2018]				><: Select Screen
System Time	[01:53:55]				^v: Select Item
					Enter: Select
					+/-: Change Opt.
					F1: General Help
					F2: Previous Values
					F3: Optimized Defaults
					F4: Save & Exit
					ESC: Exit

Version 2.19.1269. Copyright (C) 2018 American Megatrends, Inc.

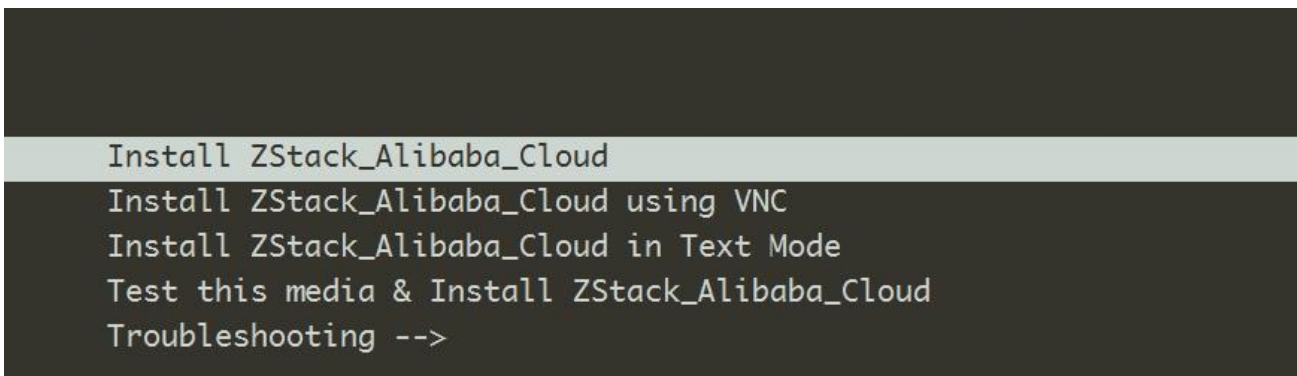
3.2.2 快速选择引导设备



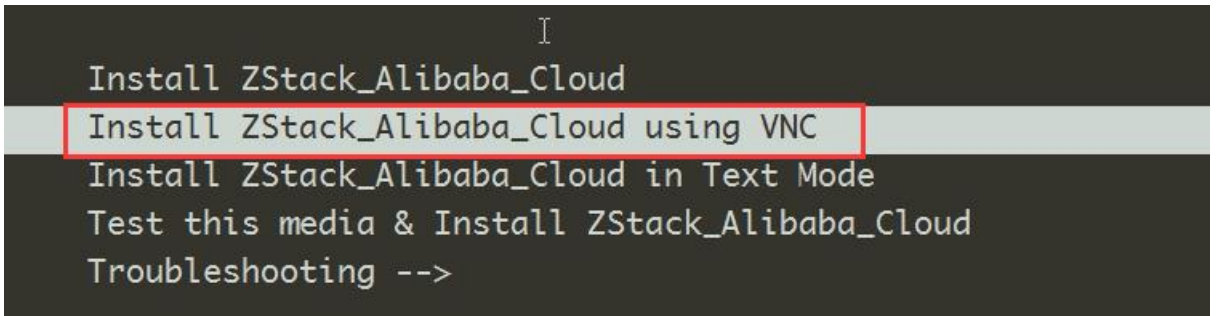
选择引导设备后按回车键，快速引导。

3.2.3 使用基于 VNC 方式安装 ZStack

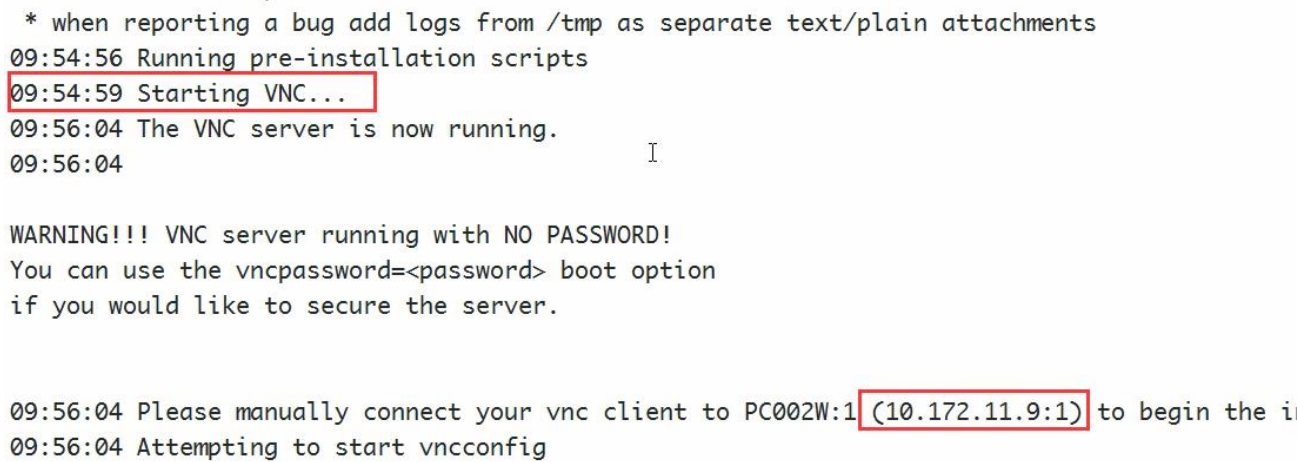
当选择引导设备后，将进入启动项选择界面，如下图所示：



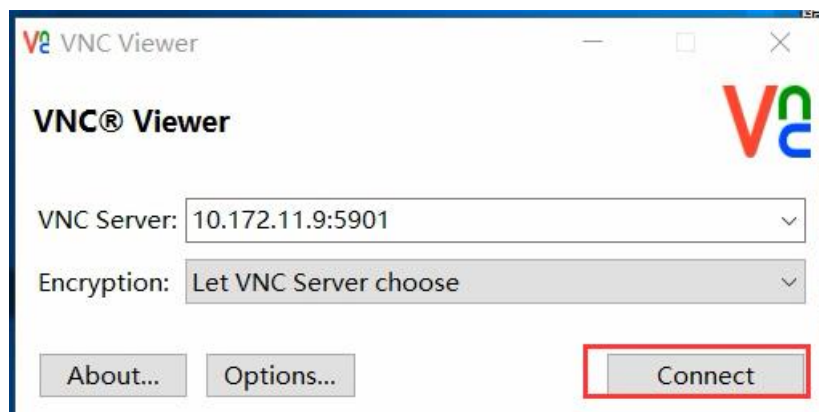
选择 using VNC 模式进行引导启动;



选择 usingVNC 模式引导启动，即可实现通过 VNC 图形模式进行安装:

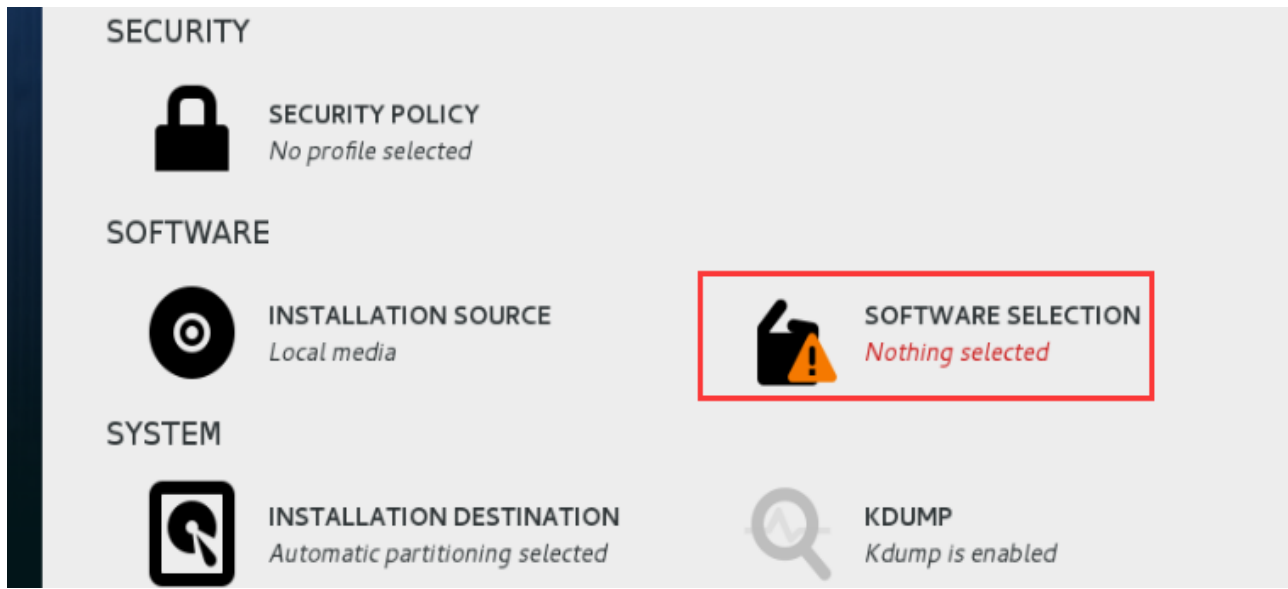


表示启动 VNC 服务，并自动从 DHCP 工具获取 IP 地址同时自动分配默认 VNC 端口 5901；当出现这个界面即可使用 VNC viewer 客户端进行连接。



3.2.4 安装设置

A. 选择安装模式

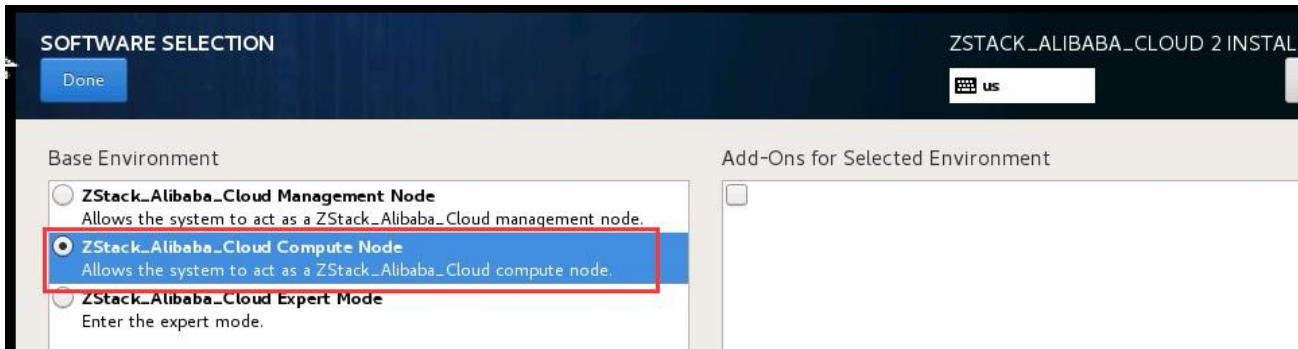


目前 ZStack For ARM 有 3 种安装模式分别对应为：

- 企业版管理节点模式
- 计算节点模式
- 专家模式

可根据实施规划进行选择部署, 选择建议：

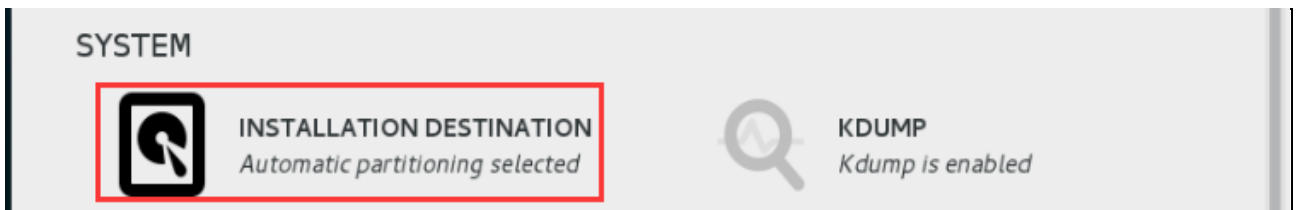
- 如果在实施方案中将管理节点独立, 则第一次安装时应选择管理节点模式；
- 如果用承载云主机, 则安装模式为计算节点；



根据实际情况选择好对应的安装模式，然后点击 Done 按钮；

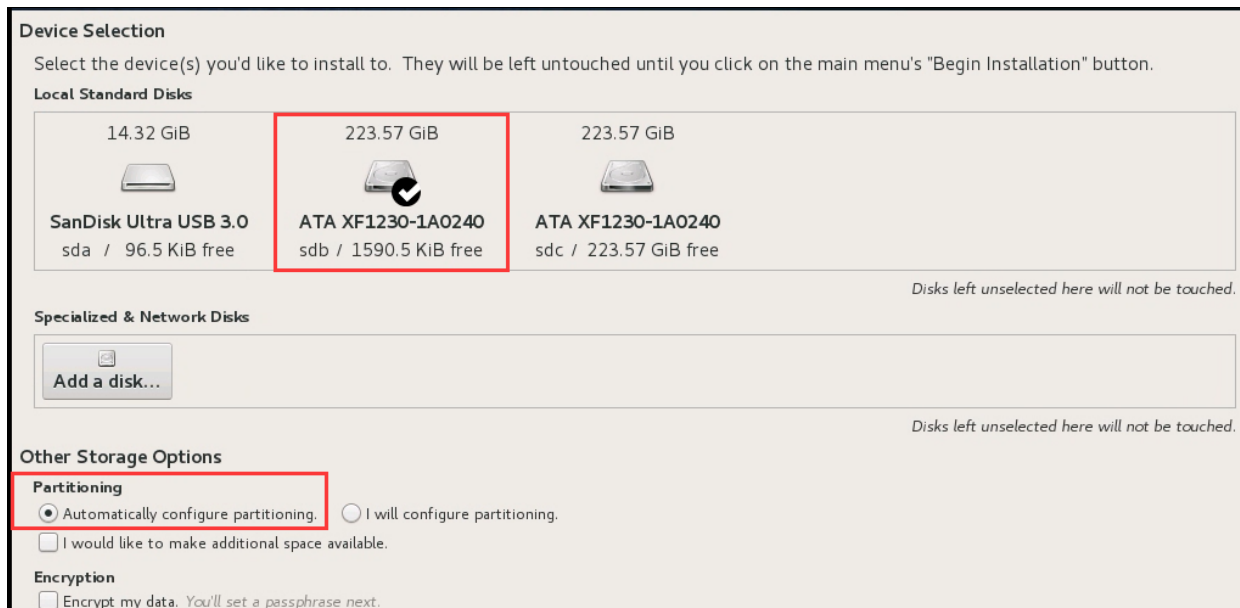
B. 配置磁盘分区：

➤ 选择磁盘：



选择用于安装 ZStack 的系统盘。

➤ 配置分区



➤ 自动分区。

下面就分区模式进行说明：

分区模式有UEFI 模式和Legacy模式两种，应与BIOS设置的引导模式一致。

- UEFI 模式

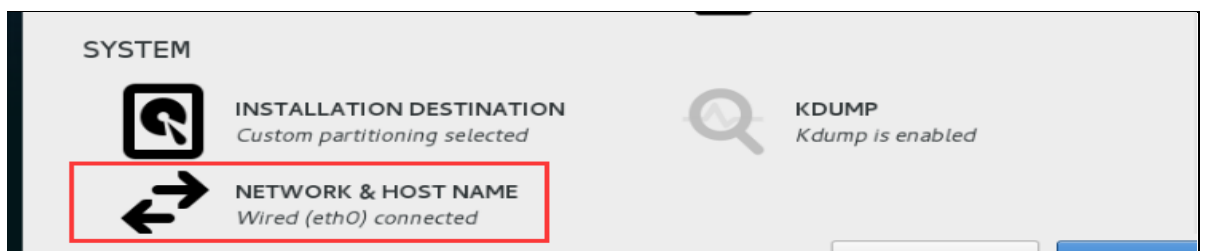
/boot：创建分区 1GB

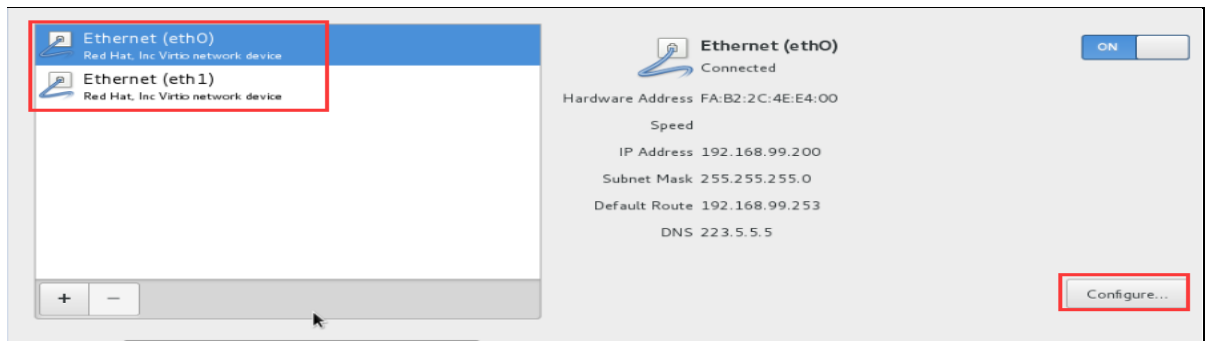
/boot/efi：创建分区 500MB

swap（交换分区）：创建分区 32GB

/（根分区）：配置剩下容量

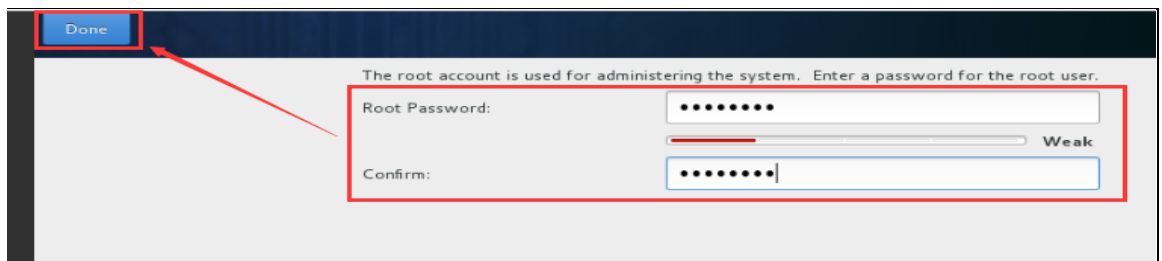
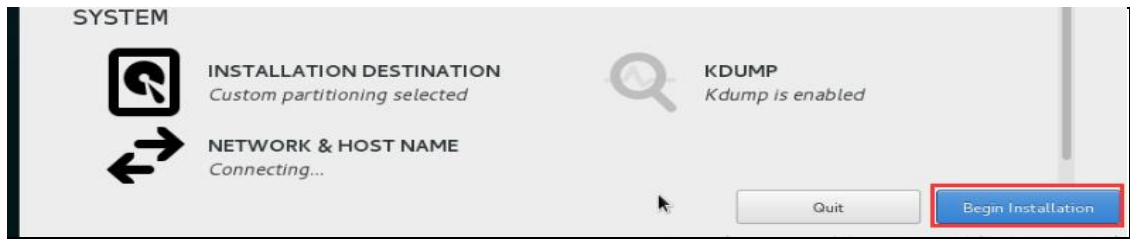
➤ 网络设置：



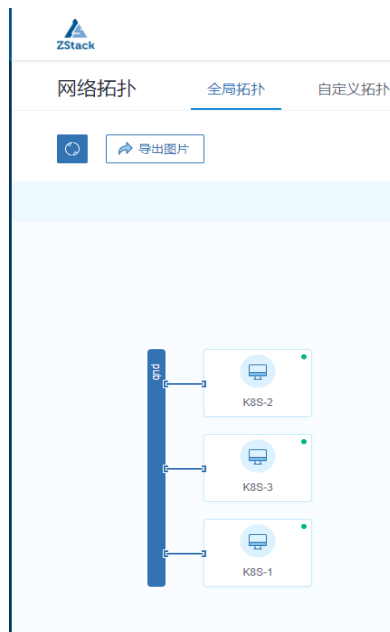


选择需要修改的网卡，点击 Configure 按钮进行配置；

设置密码并开始安装：



各模式安装部署步骤都大同小异，官网可以直接下载用户手册。安装完后的 Web UI，非常简洁大方，整个安装过程超级简单，以前一直都是使用 OpenStack 的，而这回使用 ZStack 不到 30 分钟部署成功，1 个小时内 3 个节点全部部署成功，还顺带初始化了环境。



安装部署结束后，可以看到还有网络拓扑功能

安装总结：

底层硬件是 ARM 服务器，云平台底层也是基于 ARM64 位的系统。安装部署超级方便，管理控制层与业务层完全独立，就是说如果管控节点宕掉，根本就不影响业务系统的正常运行，这一点是 OpenStack 无法实现的。在测试过程中尝试各种断电关机测试，整个平台运行依然不受影响，稳定性非常高。目前在 ZStack For ARM 云平台上轻松跑了 16 个 ARM 架构的云主机。



第三节 基于 ZStack 云主机构建 K8S 集群

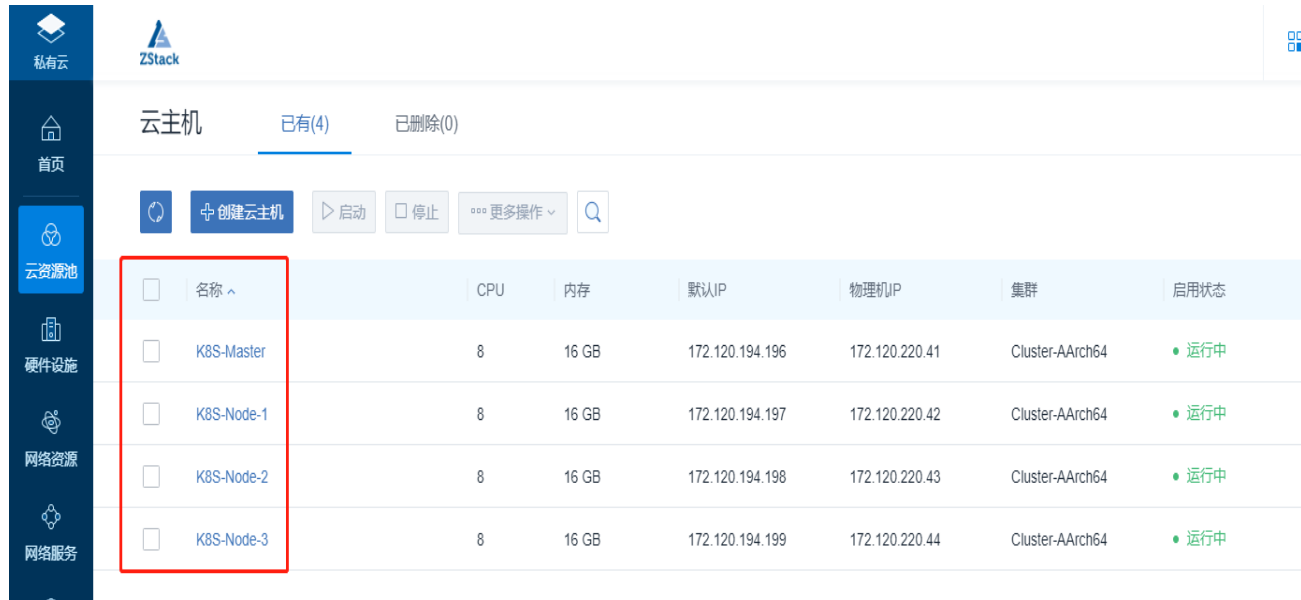
这里要提一下，为什么我们不直接使用物理 ARM 服务器部署 K8S 集群，这跟单位测试场景有关系，既要使用云主机透传 GPU 计算卡进行大量的计算，又要实现容器管理平台。况且国外主流的 K8S 集群通常是跑在虚拟机里面的，运行在虚拟机里面的好处有很多，比如可以实现资源定制分配、利用云平台 API 接口可以快速生成 K8S 集群 Node 节点、更好的灵活性以及可靠性；在 ZStack ARM 云平台上可以同时构建 IaaS+PaaS 混合平台，满足不同场景下的需求。

由于篇幅有限下面先介绍一下如何在基于 ZStack For ARM 平台中云主机部署 K8S 集群，整个部署过程大概花 1 小时（这主要是访问部分国外网络时不是很顺畅）。

集群环境介绍：

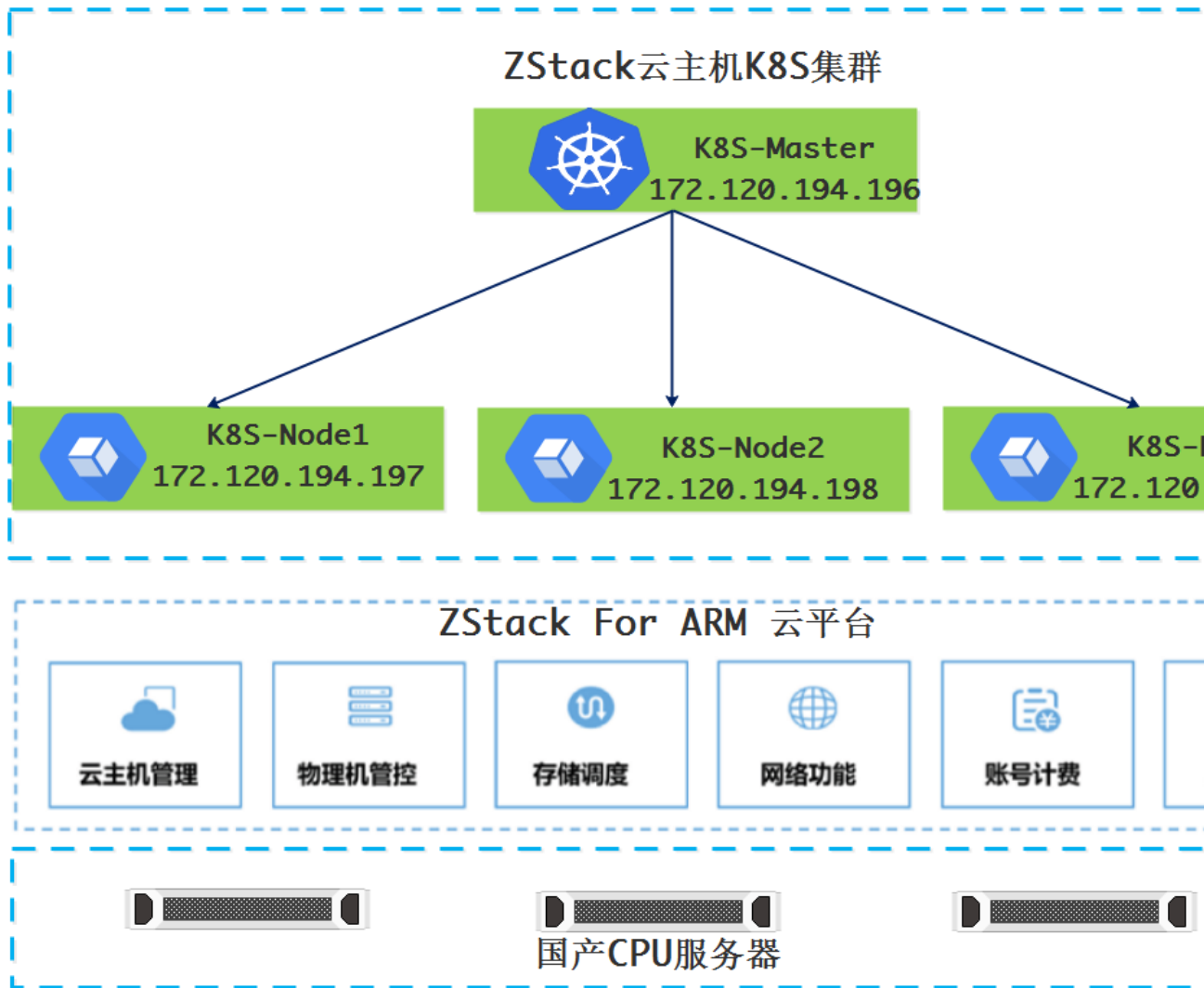
主机名	角色	IP 地址	配置	系统版本
K8S-Master	Master	172.120.194.196	8vCPU\16G 内存	Ubuntu-1804-aarch64
K8S-Node1	Node	172.120.194.197	8vCPU\16G 内存	Ubuntu-1804-aarch64
K8S-Node2	Node	172.120.194.198	8vCPU\16G 内存	Ubuntu-1804-aarch64
K8S-Node3	Node	172.120.194.199	8vCPU\16G 内存	Ubuntu-1804-aarch64

在本环境中用于构建 K8S 集群所需的资源，为基于 ZStack 构建的平台上的云主机：



The screenshot shows the ZStack management console interface. On the left is a navigation sidebar with options like '私有云', '云资源池', '硬件设施', '网络资源', and '网络服务'. The main area is titled '云主机' (Cloud Hosts) and shows a list of 4 hosts. A red box highlights the first four rows of the table, which correspond to the data in the table above. The table columns are: 名称 (Name), CPU, 内存 (Memory), 默认IP (Default IP), 物理机IP (Physical IP), 集群 (Cluster), and 启用状态 (Status).

名称 ^	CPU	内存	默认IP	物理机IP	集群	启用状态
K8S-Master	8	16 GB	172.120.194.196	172.120.220.41	Cluster-AArch64	运行中
K8S-Node-1	8	16 GB	172.120.194.197	172.120.220.42	Cluster-AArch64	运行中
K8S-Node-2	8	16 GB	172.120.194.198	172.120.220.43	Cluster-AArch64	运行中
K8S-Node-3	8	16 GB	172.120.194.199	172.120.220.44	Cluster-AArch64	运行中



ZStack 云主机 K8S 集群架构

1、准备工作

配置主机名

```
hostnamectl set-hostname K8S-Master  
  
hostnamectl set-hostname K8S-Node1  
  
hostnamectl set-hostname K8S-Node2  
  
hostnamectl set-hostname K8S-Node3
```

所有云主机上关闭 swap 分区 否则会报错；该操作只需在云主机环境下执行，物理机环境无需操作。

```
sudo swapoff -a
```

2、安装部署

2.1 安装 Docker

step 1: 安装必要的一些系统工具

```
sudo apt-get update

sudo apt-get -y install apt-transport-https ca-certificates curl software-properties-common
```

step 2: 安装 GPG 证书

```
curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
```

Step 3: 写入软件源信息

```
sudo add-apt-repository "deb [arch=arm64] http://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
```

Step 4: 更新并安装 Docker-CE

```
sudo apt-get -y update

sudo apt-get -y install docker-ce
```

使用 daocloud 对 docker 镜像下载进行加速。

```
curl -sSL https://get.daocloud.io/daotools/set_mirror.sh | sh -s http://56d10455.m.daocloud.io
```

2.2 安装 go 环境

```
apt-get install golang- golang
```

2.3 安装 kubelet、kubeadm、kubectl

```
apt-get update && apt-get install -y apt-transport-https  
  
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add  
-  
  
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list  
  
deb http://apt.kubernetes.io/ kubernetes-xenial main  
  
EOF  
  
apt-get update  
  
apt-get install -y kubectl kubeadm kubelet
```

2.4 用 kubeadm 创建集群

初始化 Master

```
kubeadm init --apiserver-advertise-address 172.120.194.196 --pod-network-  
cidr 10.244.0.0/16
```

执行完上面命令后，如果中途不报错会出现类似以下信息：

```
kubeadm join 172.120.194.196:6443 --token oyf6ns.whcoaprs0q7growa --  
discovery-token-ca-cert-hash  
sha256:30a459df1b799673ca87f9dcc776f25b9839a8ab4b787968e05edfb6efe6a9d2
```

这段信息主要是提示如何注册其他节点到 K8S 集群。

2.5 配置 kubectl

Kubectl 是管理 K8S 集群的命令行工具，因此需要对 kubectl 运行环境进行配置。


```
su - zstack

sudo mkdir -p $HOME/.kube

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

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

echo "source <(kubectl completion bash)" >> ~/.bash
```

2.6 安装 Pod 网络

为了让 K8S 集群的 Pod 之间能够正常通讯，必须安装 Pod 网络，Pod 网络可以支持多种网络方案，当前测试环境采用 Flannel 模式。

先将 Flannel 的 yaml 文件下载到本地，进行编辑，编辑的主要目的是将原来 X86 架构的镜像名称，改为 ARM 架构的。让其能够在 ZStack ARM 云环境正常运行。修改位置及内容参考下面文件中红色粗体字部分。

```
sudo wget
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml
```

```
vim kube-flannel.yml

---

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1beta1

metadata:

  name: flannel

rules:

  - apiGroups:
```

```
    - ""

    resources:

      - pods

    verbs:

      - get

  - apiGroups:

    - ""

    resources:

      - nodes

    verbs:

      - list

      - watch

  - apiGroups:

    - ""

    resources:

      - nodes/status

    verbs:

      - patch

---

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1beta1
```

```
metadata:
  name: flannel

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: flannel

subjects:
- kind: ServiceAccount
  name: flannel
  namespace: kube-system
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: flannel
  namespace: kube-system
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: kube-flannel-cfg
```

```
namespace: kube-system

labels:

  tier: node

  app: flannel

data:

  cni-conf.json: |

    {

      "name": "cbr0",

      "plugins": [

        {

          "type": "flannel",

          "delegate": {

            "hairpinMode": true,

            "isDefaultGateway": true

          }

        },

        {

          "type": "portmap",

          "capabilities": {

            "portMappings": true

          }

        }

      ]

    }
```

```
    }
  ]
}
net-conf.json: |
{
  "Network": "10.244.0.0/16",
  "Backend": {
    "Type": "vxlan"
  }
}
---
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: kube-flannel-ds
  namespace: kube-system
  labels:
    tier: node
    app: flannel
spec:
  template:
```

```
metadata:

  labels:

    tier: node

    app: flannel

spec:

  hostNetwork: true

  nodeSelector:

    beta.kubernetes.io/arch: arm64

  tolerations:

  - key: node-role.kubernetes.io/master

    operator: Exists

    effect: NoSchedule

  serviceAccountName: flannel

  initContainers:

  - name: install-cni

    image: quay.io/coreos/flannel:v0.10.0-arm64

    command:

    - cp

  args:

  - -f

  - /etc/kube-flannel/cni-conf.json
```

```
- /etc/cni/net.d/10-flannel.conflist

volumeMounts:

- name: cni

  mountPath: /etc/cni/net.d

- name: flannel-cfg

  mountPath: /etc/kube-flannel/

containers:

- name: kube-flannel

  image: quay.io/coreos/flannel:v0.10.0-arm64

  command:

  - /opt/bin/flanneld

  args:

  - --ip-masq

  - --kube-subnet-mgr

  resources:

    requests:

      cpu: "100m"

      memory: "50Mi"

    limits:

      cpu: "100m"

      memory: "50Mi"
```

```
securityContext:

  privileged: true

env:

- name: POD_NAME

  valueFrom:

    fieldRef:

      fieldPath: metadata.name

- name: POD_NAMESPACE

  valueFrom:

    fieldRef:

      fieldPath: metadata.namespace

volumeMounts:

- name: run

  mountPath: /run

- name: flannel-cfg

  mountPath: /etc/kube-flannel/

volumes:

- name: run

  hostPath:

    path: /run

- name: cni
```



```
    hostPath:
      path: /etc/cni/net.d
- name: flannel-cfg
  configMap:
    name: kube-flannel-cfg
```

```
sudo kubectl apply -f kube-flannel.yml
```

执行上面命令后会正常情况下会有如下输出：

```
clusterrole.rbac.authorization.k8s.io "flannel" created
clusterrolebinding.rbac.authorization.k8s.io "flannel" created
serviceaccount "flannel" created
configmap "kube-flannel-cfg" created
daemonset.extensions "kube-flannel-ds" created
```

2.7 注册节点到 K8S 集群

分别在 K8S-Node1、K8S-Node2、K8S-Node3

```
kubeadm join 172.120.194.196:6443 --token oyf6ns.whcoaprs0q7growa --
discovery-token-ca-cert-hash
sha256:30a459df1b799673ca87f9dcc776f25b9839a8ab4b787968e05edfb6efe6a9d2
```

kubectl get nodes 查看节点状态

```
zstack@K8S-Master:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
k8s-master	Ready	master	49m	v1.11.0
k8s-node1	NotReady	<none>	4m	v1.11.0
k8s-node2	NotReady	<none>	4m	v1.11.0
k8s-node3	NotReady	<none>	4m	v1.11.0

如果发现所有节点是 NotReady 是因每个节点都需要启动若干个组件，这些组件都是在 Pod 中运行，且需要到 Google 下载镜像。使用下面命令查看 Pod 运行状况：

```
kubectl get pod --all-namespaces  正常情况应该是如下的状态：
```

NAMESPACE	NAME	READY	STATUS
kube-system	coredns-78fcd6894-49tkw	1/1	Running
1h			0
kube-system	coredns-78fcd6894-gmcp	1/1	Running
1h			0
kube-system	etcd-k8s-master	1/1	Running
19m			0
kube-system	kube-apiserver-k8s-master	1/1	Running
19m			0
kube-system	kube-controller-manager-k8s-master	1/1	Running
19m			0
kube-system	kube-flannel-ds-bqx2s	1/1	Running
16m			0
kube-system	kube-flannel-ds-jgmjp	1/1	Running
16m			0

21m	kube-system	kube-flannel-ds-mxpl8	1/1	Running	0
16m	kube-system	kube-flannel-ds-sd6lh	1/1	Running	0
16m	kube-system	kube-proxy-cwslw	1/1	Running	0
1h	kube-system	kube-proxy-j75fj	1/1	Running	0
16m	kube-system	kube-proxy-ptn55	1/1	Running	0
16m	kube-system	kube-proxy-zl8mb	1/1	Running	0
19m	kube-system	kube-scheduler-k8s-master	1/1	Running	0

在整个过程中如果发现状态为 Pending、ContainerCreateing、ImagePullBackOff 等状态都表示 Pod 还未就绪，只有 Running 状态才是正常的。要做的事情只有等待。

kubectl get nodes 再次查看节点状态

NAME	STATUS	ROLES	AGE	VERSION
k8s-master	Ready	master	1h	v1.11.0
k8s-node1	Ready	<none>	16m	v1.11.0
k8s-node2	Ready	<none>	16m	v1.11.0
k8s-node3	Ready	<none>	16m	v1.11.0

当所有节点均为 Ready 状时，此时就可以使用这个集群了

2.8 部署 kubernetes-dashboard

克隆 kubernetes-dashboard yaml 文件

```
sudo git clone https://github.com/gh-Devin/kubernetes-dashboard.git
```

修改 kubernetes-dashboard yaml 文件, 修改内容为下面红色粗体部分。

```
cd kubernetes-dashboard/  
  
vim kubernetes-dashboard.yaml  
  
# Copyright 2017 The Kubernetes Authors.  
  
#  
  
# Licensed under the Apache License, Version 2.0 (the "License");  
# you may not use this file except in compliance with the License.  
# You may obtain a copy of the License at  
  
#  
# http://www.apache.org/licenses/LICENSE-2.0  
#  
  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
  
# Configuration to deploy release version of the Dashboard UI compatible
```

with

```
# Kubernetes 1.8.
```

```
#
```

```
# Example usage: kubectl create -f <this_file>
```

```
# ----- Dashboard Secret ----- #
```

```
apiVersion: v1
```

```
kind: Secret
```

```
metadata:
```

```
  labels:
```

```
    k8s-app: kubernetes-dashboard
```

```
  name: kubernetes-dashboard-certs
```

```
  namespace: kube-system
```

```
type: Opaque
```

```
---
```

```
# ----- Dashboard Service Account ----- #
```

```
apiVersion: v1
```

```
kind: ServiceAccount
```

```
metadata:

  labels:

    k8s-app: kubernetes-dashboard

  name: kubernetes-dashboard

  namespace: kube-system

---

# ----- Dashboard Role & Role Binding ----- #

kind: Role

apiVersion: rbac.authorization.k8s.io/v1

metadata:

  name: kubernetes-dashboard-minimal

  namespace: kube-system

rules:

  # Allow Dashboard to create 'kubernetes-dashboard-key-holder' secret.
- apiGroups: [""]

  resources: ["secrets"]

  verbs: ["create"]

  # Allow Dashboard to create 'kubernetes-dashboard-settings' config map.
- apiGroups: [""]
```

```
resources: ["configmaps"]

verbs: ["create"]

# Allow Dashboard to get, update and delete Dashboard exclusive secrets.
- apiGroups: [""]

resources: ["secrets"]

resourceNames: ["kubernetes-dashboard-key-holder", "kubernetes-dashboard-
certs"]

verbs: ["get", "update", "delete"]

# Allow Dashboard to get and update 'kubernetes-dashboard-settings' config
map.
- apiGroups: [""]

resources: ["configmaps"]

resourceNames: ["kubernetes-dashboard-settings"]

verbs: ["get", "update"]

# Allow Dashboard to get metrics from heapster.
- apiGroups: [""]

resources: ["services"]

resourceNames: ["heapster"]

verbs: ["proxy"]

- apiGroups: [""]

resources: ["services/proxy"]

resourceNames: ["heapster", "http:heapster:", "https:heapster:"]
```

```
verbs: ["get"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

  name: kubernetes-dashboard-minimal

  namespace: kube-system

roleRef:

  apiGroup: rbac.authorization.k8s.io

  kind: Role

  name: kubernetes-dashboard-minimal

subjects:

- kind: ServiceAccount

  name: kubernetes-dashboard

  namespace: kube-system

---

# ----- Dashboard Deployment ----- #

kind: Deployment
```



```
apiVersion: apps/v1beta2

metadata:

  labels:

    k8s-app: kubernetes-dashboard

  name: kubernetes-dashboard

  namespace: kube-system

spec:

  replicas: 1

  revisionHistoryLimit: 10

  selector:

    matchLabels:

      k8s-app: kubernetes-dashboard

  template:

    metadata:

      labels:

        k8s-app: kubernetes-dashboard

    spec:

      serviceAccountName: kubernetes-dashboard

      containers:

        - name: kubernetes-dashboard

          image: k8s.gcr.io/kubernetes-dashboard-arm64:v1.8.3
```

```
ports:

  - containerPort: 9090

  protocol: TCP

args:

  #- --auto-generate-certificates

  # Uncomment the following line to manually specify Kubernetes API
server Host

  # If not specified, Dashboard will attempt to auto discover the
API server and connect

  # to it. Uncomment only if the default does not work.

volumeMounts:

  - name: kubernetes-dashboard-certs

    mountPath: /certs

    # Create on-disk volume to store exec logs

  - mountPath: /tmp

    name: tmp-volume

livenessProbe:

  httpGet:

    scheme: HTTP

    path: /

    port: 9090

  initialDelaySeconds: 30
```

```
        timeoutSeconds: 30

    volumes:

    - name: kubernetes-dashboard-certs

      secret:

        secretName: kubernetes-dashboard-certs

    - name: tmp-volume

      emptyDir: {}

      serviceName: kubernetes-dashboard-admin

      # Comment the following tolerations if Dashboard must not be deployed
on master

      tolerations:

    - key: node-role.kubernetes.io/master

      effect: NoSchedule

---

# ----- Dashboard Service ----- #

kind: Service

apiVersion: v1

metadata:

  labels:

    k8s-app: kubernetes-dashboard
```

```
name: kubernetes-dashboard

namespace: kube-system

spec:

  ports:

    - port: 9090

      targetPort: 9090

  selector:

    k8s-app: kubernetes-dashboard

# -----

kind: Service

apiVersion: v1

metadata:

  labels:

    k8s-app: kubernetes-dashboard

  name: kubernetes-dashboard-external

  namespace: kube-system

spec:

  ports:

    - port: 9090

      targetPort: 9090
```

```
nodePort: 30090
```

```
type: NodePort
```

```
selector:
```

```
k8s-app: kubernetes-dashboard
```

修改完成后执行

```
kubectl -n kube-system create -f .
```

执行命令的正常输出:

```
serviceaccount "kubernetes-dashboard-admin" created
```

```
clusterrolebinding.rbac.authorization.k8s.io "kubernetes-dashboard-admin"  
created
```

```
secret "kubernetes-dashboard-certs" created
```

```
serviceaccount "kubernetes-dashboard" created
```

```
role.rbac.authorization.k8s.io "kubernetes-dashboard-minimal" created
```

```
rolebinding.rbac.authorization.k8s.io "kubernetes-dashboard-minimal"  
created
```

```
deployment.apps "kubernetes-dashboard" created
```

```
service "kubernetes-dashboard-external" created
```

然后查看 kubernetes-dashboard Pod 的状态

```
kubectl get pod --all-namespaces
```

NAMESPACE	NAME	READY	STATUS
-----------	------	-------	--------

```
RESTARTS   AGE
kube-system  kubernetes-dashboard-66885dcb6f-v6qfm  1/1    Running  0
```

8m

当状态为 running 时执行下面命令 查看端口

```
kubectl --namespace=kube-system describe svc kubernetes-dashboard
```

```
Name:                kubernetes-dashboard-external
```

```
Namespace:           kube-system
```

```
Labels:              k8s-app=kubernetes-dashboard
```

```
Annotations:         <none>
```

```
Selector:            k8s-app=kubernetes-dashboard
```

```
Type:                NodePort
```

```
IP:                  10.111.189.106
```

```
Port:                <unset> 9090/TCP
```

```
TargetPort:          9090/TCP
```

```
NodePort:            <unset> 30090/TCP    此端口为外部访问端口
```

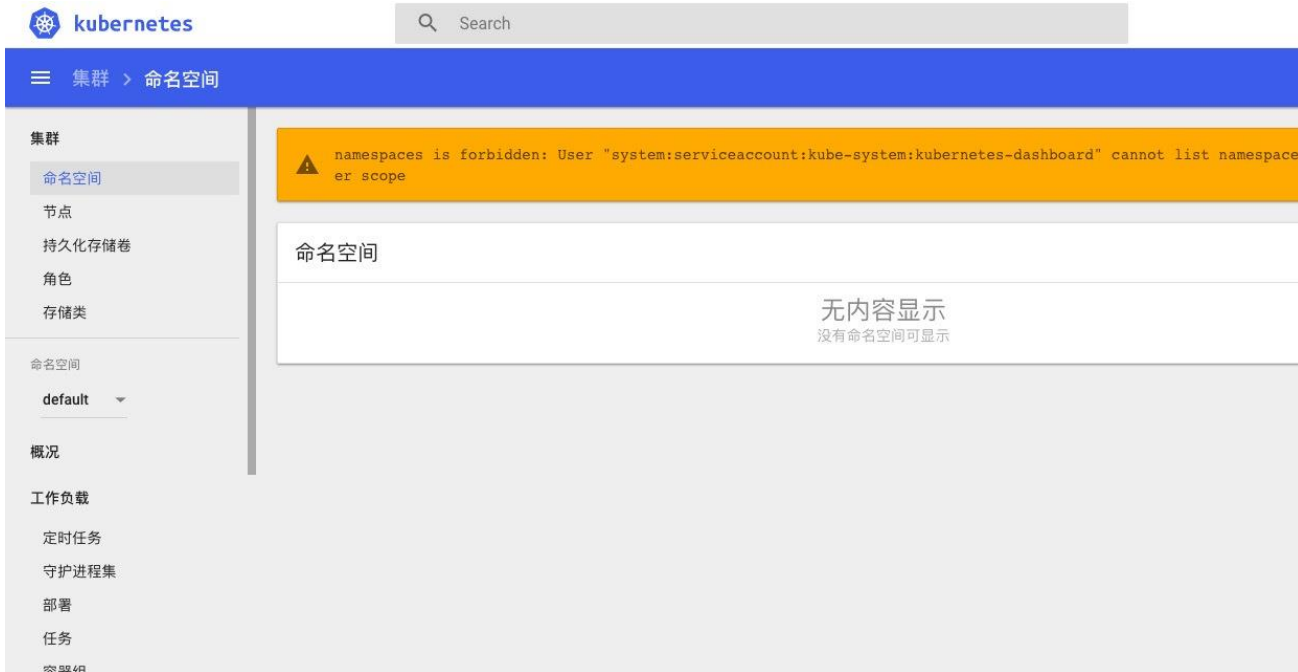
```
Endpoints:           10.244.2.4:9090
```

```
Session Affinity:    None
```

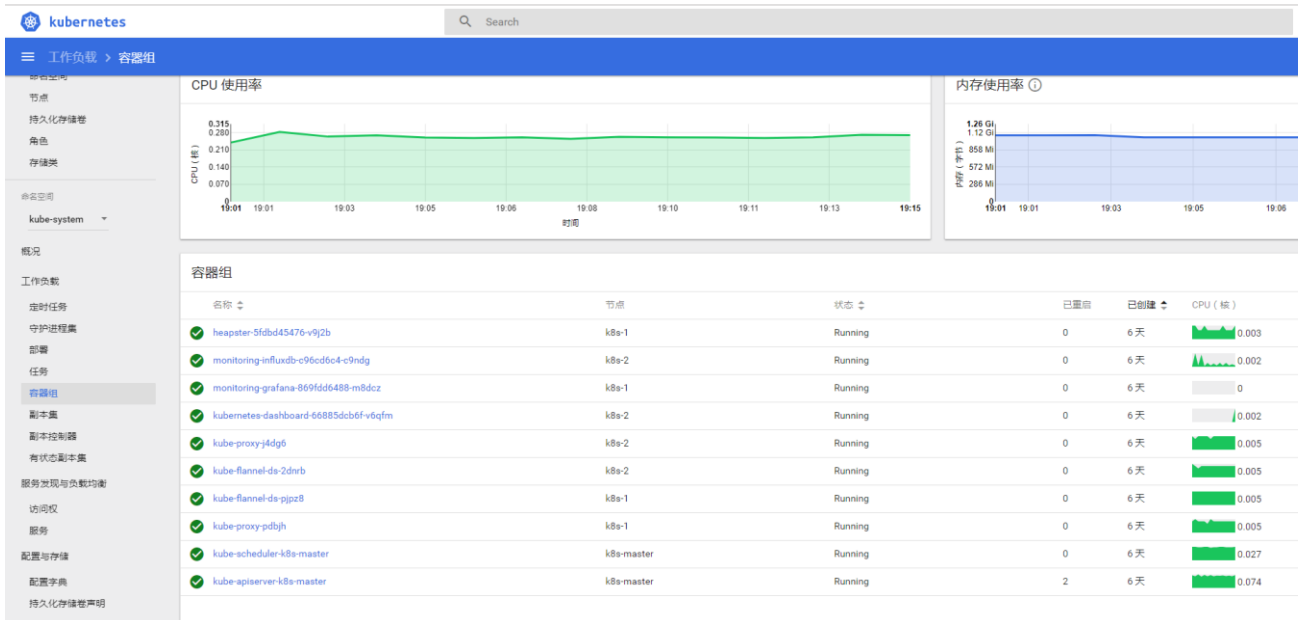
```
External Traffic Policy: Cluster
```

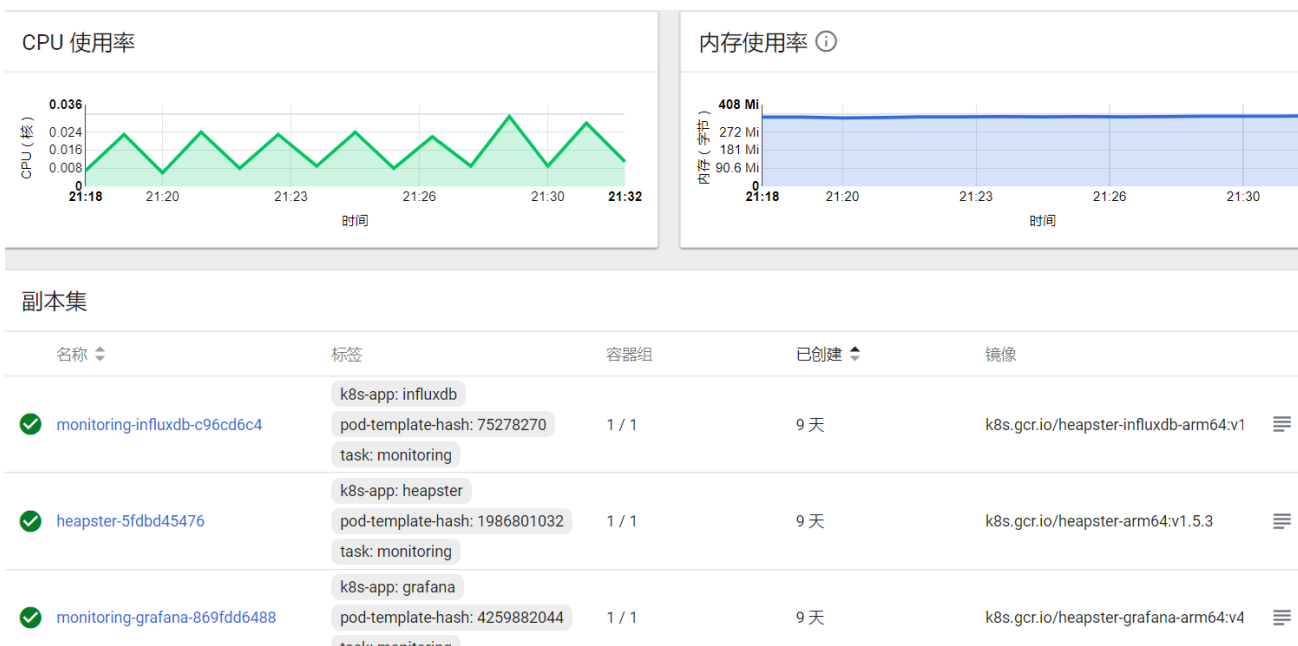
```
Events:              <none>
```

注意：如果在部署 K8S-Dashboard 界面过程中如果则登录 UI 的时候会报错：



这是因为 K8S 在 1.6 版本以后启用了 RBAC 访问控制策略，可以使用 kubectl 或 Kubernetes API 进行配置。使用 RBAC 可以直接授权给用户，让用户拥有授权管理的权限，这样就不再需要直接触碰 Master Node。按照上面部署步骤则可以避免。





至此，基于 ARM 环境的 K8S 集群就部署完成了。

第四节 全篇总结

先说说关于 ZStack 安装部署的一些心得，整个 ZStack For ARM 平台部署到业务环境构建的过程，都是比较流畅的。ZStack 产品化程度高，安装过程非常简单，基本上按照官方部署文档 1 个小时内就能完成 3 台规模的云平台搭建及平台初始化工作。

ZStack 云平台采用独特的异步架构，大大提升了平台响应能力，使得批量并发操作不再成为烦恼；管理层面与业务层面独立，不会因为管理节点意外宕机导致业务中断；平台内置大量实用性很高的功能，极大方便了在测试过程中运维任务；版本升级简单可靠，完全实现 5 分钟跨版本无缝升级，经实测升级过程中完全不影响业务正常运行。通过升级后能实现异构集群管理，也就是说在 ARM 服务器上构建管理节点，可以同时管理 ARM 集群中的资源，

也能管理 X86 架构集群中的资源；同时实现高级 SDN 功能。

而基于 ZStack 云主机构建 K8S 集群时，我们团队在选择方案的时候，也拿物理机和云主机做过一系列对比，对比之后发现当我用 ZStack 云主机部署 K8S 集群的时候更加灵活、可控。具体的可以在以下几个方面体现：

1、ZStack 云主机天生隔离性好

对容器技术了解的人应该清楚，多个容器公用一个 Host Kernel；这样就会遇到隔离性方面的问题，虽然随着技术发展，目前也可以使用 Linux 系统上的防护机制实现安全隔离，但是从某个层面讲并不是完全隔离，而云主机方式受益于虚拟化技术，天生就有非常好的隔离性，从而可以进一步保障安全。ZStack 就是基于 KVM 虚拟化技术架构自研。

2、受益于 ZStack 云平台多租户

在物理服务器上运行的大堆容器要实现资源自理，所谓资源自理就是各自管理自己的容器资源，那么这个时候问题就来了，一台物理机上有成千上万个容器怎么去细化管理范围呢？这个时候云平台的多租户管理就派上用场了，每个租户被分配到相应的云主机，各自管理各自的云主机以及容器集群。同时还能对不同人员权限进行控制管理。在本次测试的 ZStack For ARM 云平台，就可以实现按企业组织架构方式进行资源、权限管理，同时还能实现流程审批，审批完成后自动创建所需的云主机；据说后面发布的 ZStack2.5.0 版本还有资源编排功能。

3. ZStack 云平台灵活性、自动化程度高

通过 ZStack，可以根据业务需求，对云主机进行资源定制，减少资源浪费。同时根据自身业务情况调整架构实现模式，比如：有计算密集型业务，此时可以借助 GPU 透传功能，将 GPU 透传到云主机，能快速实现计算任务，避免过多繁琐配置。

另外目前各种云平台都有相应 API 接口，可以方便第三方应用直接调用，从而实现根据业务压力自动进行资源伸缩。但是对于物理服务器来说没什么完整的 API 接口，基本上都是基于 IPMI 方式进行管理，而且每个厂商的 IPMI 还不通用，很难实现资源的动态伸缩。说到 API 接口，我了解到的 ZStack 云平台，具备全 API 接口开放的特点。可以使容器集群根据业务压力自动伸缩。

4、可靠性非常好

为什么这么说呢?其实不难理解, 计划内和计划外业务影响少。当我们对物理服务器进行计划内维护时, 那些单容器运行的业务必定会受影响, 此时可以借助云平台中的热迁移功能, 迁移的过程中可实现业务不中断。对于计划外停机, 对业务影响基本上都是按天算的, 损失不可言表。如果采用云平台方式业务中断时间将会缩短到分钟级别。

上面简单分享了一下用云主机构建 K8S 集群的一些优点, 当然也有一些缺点, 在我看来缺点无非就是性能有稍微点损失, 总之利大于弊。可以在规划时规避掉这个问题, 比如可以将性能型容器资源集中放到物理 Node 上, 这样就可以完美解决了。

最后再说说在 ZStack ARM 架构的云主机上部署 K8S 需要注意的地方, 为大家提供一些参考。

1、默认 Get 下来的 yaml 配置文件, 里面涉及的 image 路径都是 x86 架构的 amd64, 需要将其改成 arm64。

2、在创建集群的时候, 如果采用 flannel 网络模式则 `--pod-network-cidr` 一定要为 `10.244.0.0/16`, 否则 Pod 网可能不通。

3、云主机环境一定要执行 `sudo swapoff -a` 不然创建 K8S 集群的时候就会报错。

以上就是我本次的主要分享内容, 欢迎大家关注交流。(qq: 410185063; mail: zts@viczhu.com)。