文档中心

容器实例 Cube

产品文档

目录

概览

容器实例(Cube)是UCloud提供的serverless容器实例服务,通过UCloud的基础设施资源为业务提供了更加弹性、更加安全、更加快速的资源支撑,你可以在Cube上部署、管理你的容器应用,而你无需关心应用底层的服务器运维工作。

产品介绍 | 使用指南 | Deployment控制器 卷设置 | 常见问题

产品介绍

容器实例(Cube)将带大家进入一个全新的云原生程序的部署方式,首先我们来一起了解一下Cube是什么、它的优势是什么。

- 什么是Cube
- 产品优势
- Cube 机型配置
- 计费说明

使用指南

接下来我们使用容器实例(Cube)发布您的服务,将介绍具体创建填写字段含义以及示例操作。

- 使用须知
- 快速创建
- 创建详解

- 自建镜像仓库支持
- 第三方镜像仓库支持
- 对比K8S

Deployment控制器

通过 Deployment 控制器，进行 Cube 实例的批量创建和管理。

- 批量创建Cube实例

卷设置

在您部署容器组时，根据应用需要，部分数据需要持久化存储，您可以参考以下文档在Cube中使用UCloud的存储资源进行挂载。

- 在Cube中使用Config
- 在Cube中使用UFS
- 在Cube中使用UDisk

快速入门

- 创建带SSH服务的CentOS容器
- PHP应用的高可用部署

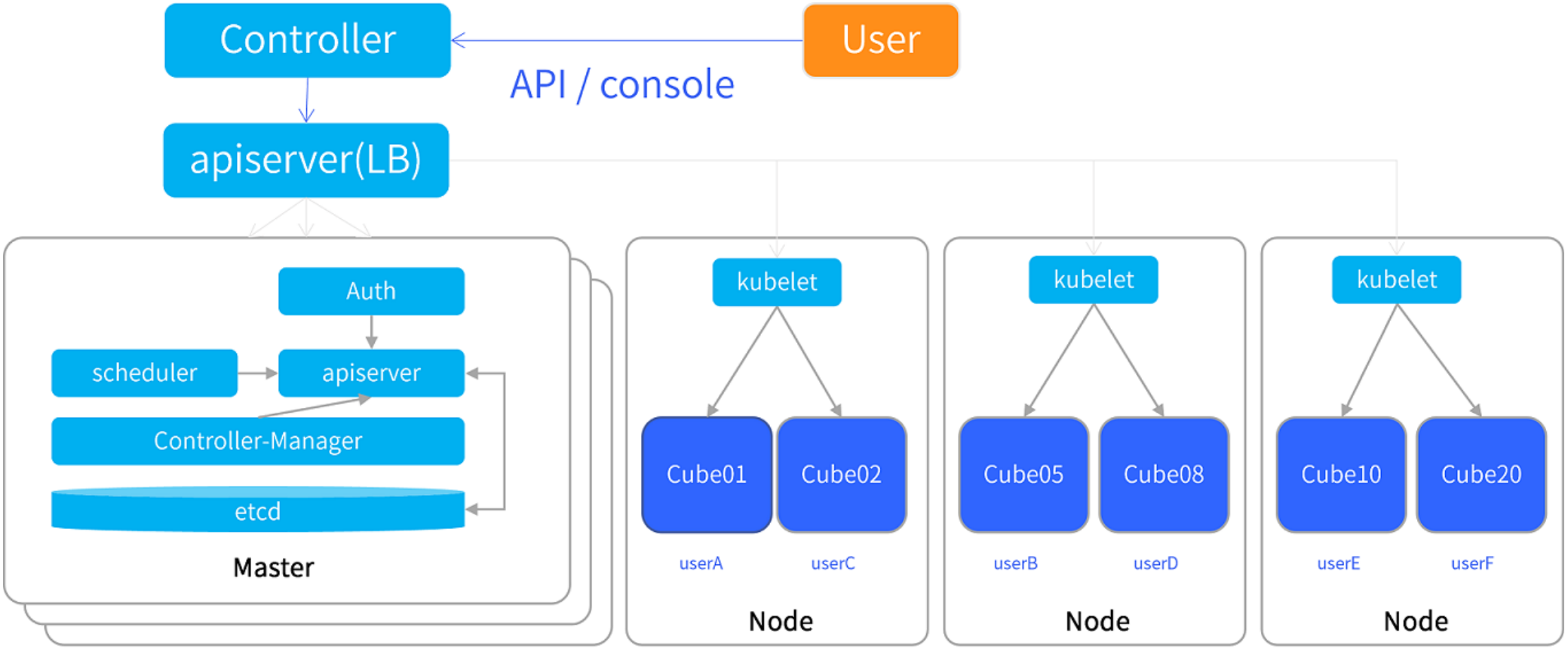
常见问题

在您部署容器组时，对应的您可能会遇到一些使用中的问题，常见问题分类将解决您遇到的大部分问题，如您未找到解决方案请与我们客服联系。

- 运行状态
- 容器重启策略

什么是 Cube

Cube 产品取名于 Kube 谐音，且有立方体的含义。如果您有使用过 Kubernetes，可以把 Cube 的一个实例理解为 Kubernetes 中的最小业务单元 Pod。一个 Cube 实例中包含一组（一个或多个）容器（Container），这些容器共享存储、网络、以及怎样运行这些容器的声明。



Cube 的出现使您不需要再去关心底层的云主机或者 Kubernetes 控制，您可以随心所欲的将您的程序镜像放进这个立方体（Cube）中运行。

Cube 产品采用 Serverless 架构，底层为海量 UCloud 基础设施资源，无需等待虚拟机启动，无需等待 Kubernetes 集群启动，您可以秒级拉起您的业务容器提供服务。

Cube 产品基于社区开源的 Firecracker 作为虚拟化和容器化的支撑管理，同时我们针对容器化的业务运行进行了深度优化，使 Cube 实例具备了虚拟机级别的安全隔离、轻量化的系统占用、秒级的启动速度。

产品优势

1. 免服务器运维

通过UCloud的基础设施资源为业务提供支撑，无需对基础设施资源进行运维工作。

2. 秒级计费

按照实际使用的秒数进行收费，减少使用资源的成本投入。

3. 秒级启动

通过使用容器镜像秒级启动容器，不再依赖主机集群创建时间。

4. 自愈

运行的容器实例将不再担心运行宕机，Cube的控制调度系统将为容器进行自动重启操作。

5. 主机级别安全隔离

通过Firecracker的虚拟化技术和容器管理技术，使容器使用独立的轻量虚拟机运行，为容器提供安全与防护。

优势对比

	Cube	自建Docker服务
成本投入	用多少买多少，可采用按秒计费	购买固定资源规格的云主机进行自建，费用高
网络	使用VPC网络进行内网服务，使用UCloud EIP和防火墙进行外网服务	单一IP，绑定额外IP/EIP繁琐
存储	使用云盘进行挂载使用，读写性能高，操作便捷	只能通过主机挂载，操作繁琐
自愈能力	Cube控制调度系统为容器自动重启	需要额外安装控制调度系统
稳定性	UCloud海量资源支撑，超大集群避免单一节点故障	单一节点，迁移繁琐
安全隔离	使用Firecracker虚拟化技术实现操作系统级别的强隔离	只能通过namespace和cgroup进行弱隔离
技术要求	通过页面即可创建	需要使用docker命令进行创建，学习成本高

Cube 机型配置

当前 Cube 容器实例已全面升级快杰版，底层采用最新 AMD 及 Intel 服务器，支持 SSD 云盘。

分类	宿主资源
计算	Intel Cascadelake CPU（2.5GHz主频） AMD 第二代 EPYC CPU（2.9GHz 主频）
存储	SSD UDisk
网络	25G 以太网

CPU / 内存配置

如通过控制台创建 Cube 实例，实例中单个 Container 支持如下 CPU / 内存规格配置：

CPU	内存
100m	128Mi
500m	512Mi/1Gi/2Gi
1	1Gi/2Gi/4Gi
2	2Gi/4Gi/8Gi

4	4Gi/8Gi/16Gi
8	8Gi/16Gi/32Gi
16	16Gi/32Gi/64Gi
32	32Gi/64Gi/128Gi

如通过 API 创建，除以上规格外，Container CPU 配置支持 1 - 32 的任一整数核数，但 CPU:内存比例需满足 1:1/2/4，如 3C3G / 3C6G / 3C12G。

每个 Cube 实例总资源最大支持 32C128Gi。

计费说明

UCloud容器实例(Cube)目前支持按月预付费、按年预付费、按秒后付费的计费方式。

1. 按月(年)预付费

按月预付费的计费方式,以月为单位预先支付使用费用,此类计费方式适合长期稳定的业务。用户可以选择一次性购买 1-9 个月,若一次性购买 10 个月以上,建议选择按年预付费以享受年付优惠。

按年预付费的计费方式,以年为单位预先支付使用费用,1年价格为月付价格*10,以此类推。

1.1 计费公式

Cube 容器实例预付费账单费用=(总 CPU 数 * CPU 单价 + 总内存数 * 内存单价) * 购买月数

1.2 定价

地域	计费因子	月付单价(元/月)
北京/上海/广州	1 核 CPU	48
北京/上海/广州	1G 内存	18
乌兰察布	1 核 CPU	35

乌兰察布	1G 内存	15

2. 按秒后付费

先使用后付费，以天为周期，合并统计单个项目中同一可用区下所有 Cube 实例的费用，每天凌晨扣除上个自然日的费用。

⚠ 避免滥用资源，使用按秒后付费，账户余额须大于等于 100 元，另请确保余额充足，以免欠费导致实例被回收。注：该计费方式需要开通权限使用，如需请联系客户经理申请。

2.1 计费公式

总费用=(PodA CPU 核数 * CPU 单价 + PodA 内存 * 内存单价) * 运行的秒数 +(PodB CPU核数 * CPU单价 + PodB 内存 * 内存单价) * 运行的秒数 +....

2.2 定价

地域	计费因子	单价(元/秒)	小时价(元/小时)
北京/上海/广州	1 核 CPU	0.0000278000	0.10008
北京/上海/广州	1G 内存	0.0000104000	0.03744
乌兰察布	1 核 CPU	0.0000202800	0.073008
乌兰察布	1G 内存	0.0000086880	0.0312768

2.3 补充说明

- 单实例统计周期：成功申请创建容器实例（正确获得容器实例资源 ID）开始至容器删除（资源 ID 删除）为止。
- 一个项目中的一个可用区下的所有 Cube 实例合并计费，每天凌晨生成一张资源 ID 为 cube-bill-xxxxxxxx 的扣费订单，扣除上个自然日的费用，扣费金额精确到分，显示计费单位为分。
- **将不对单个后付费 Pod 实例生成扣费订单**，如需查看后付费 Cube 实例详细计费统计，请在 Cube 控制台页面**统计报表**处查看：统计报表。
- 单个 Cube 实例的费用在一个自然日统计费用精确到小数点后 8 位，如同一可用区所有 Cube 实例在一个自然日产生的费用不足 0.01 元，按 0.01 元扣费，大于 0.01 元的，抹除分位后费用。

举例说明: 华北一E 可用区下有两个 Cube 实例，实例 A 在 2020 年 9 月 8 日运行时长 30 秒，统计费用为 0.0008 元，实例 B 在 2020 年 9 月 8 日运行时长 10 小时，统计费用为 1.2349 元，则 2020 年 9 月 9 日凌晨生产一张 1.23 元（123 个计费单元）的订单。

使用须知

使用须知

1. 使用Cube服务，必须通过UCloud实名认证服务；
2. 使用Cube暴露公网服务，请与UCloud备案团队联系进行备案，否则可能会影响您的服务正常使用；
3. 您的业务程序已经完成容器化，已有Docker镜像；
4. 您创建的Cube实例，将默认镜像拉取策略为总是(Always)，将不支持其他镜像拉取策略；
5. 使用按秒后付费请确保余额大于等于100元，以免后付费欠费导致资源回收。

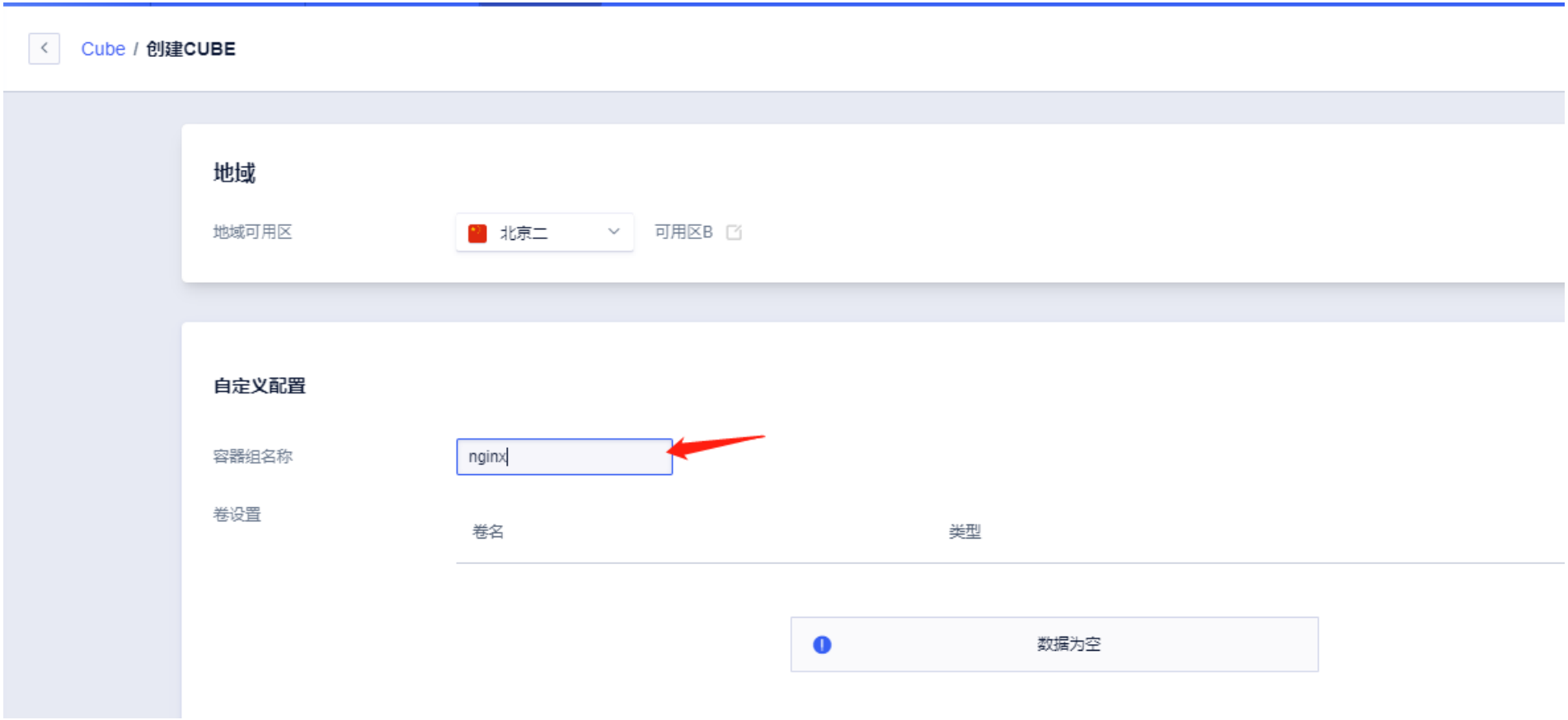
使用依赖

1. 请预先创建好VPC和子网；
2. 如果是需要绑定EIP，请预先给操作账号赋予网络相关权限；
3. 需要使用到UCloud UHub镜像仓库，需要创建并将镜像上传。

快速创建

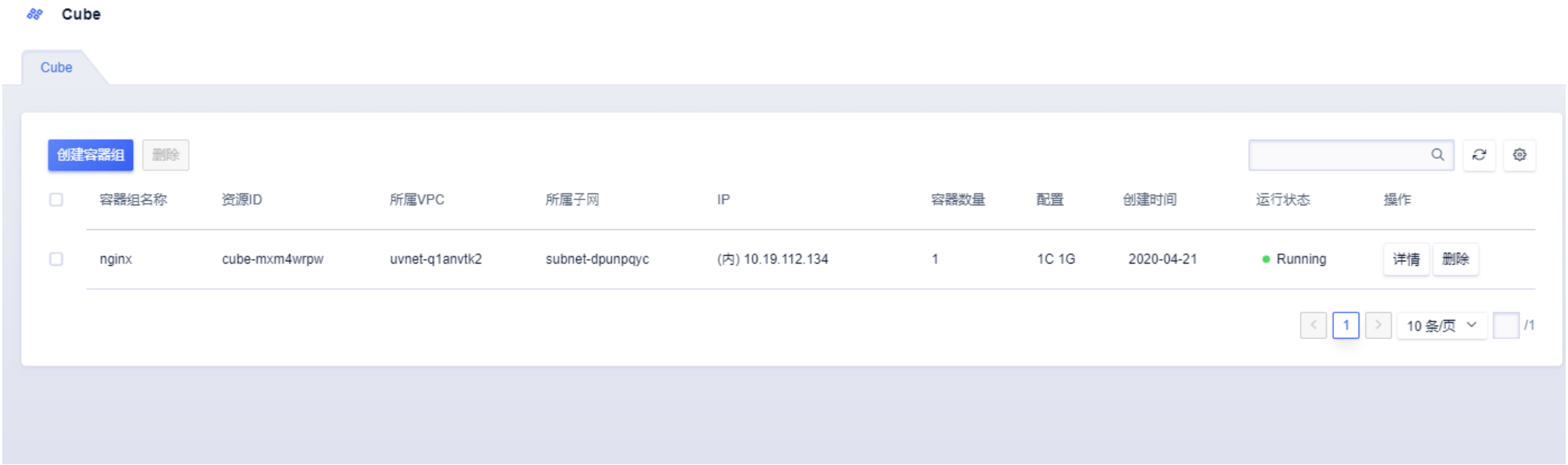
快速创建Cube实例

1. 点击Cube的创建按钮进入创建页面，为你的容器组起一个名字；



2. 选择一个仓库内的镜像，这里快速创建我选择的UCloud镜像仓库nginx镜像1.17.10-alpine版本，您可以将自有镜像推送到UCloud私有镜像仓库，UHub镜像仓库操作；如需使用自建镜像仓库，请参考自建镜像仓库支持

3. 点击确定，完成创建操作，创建一个Nginx容器组，从创建到running约5秒完成。



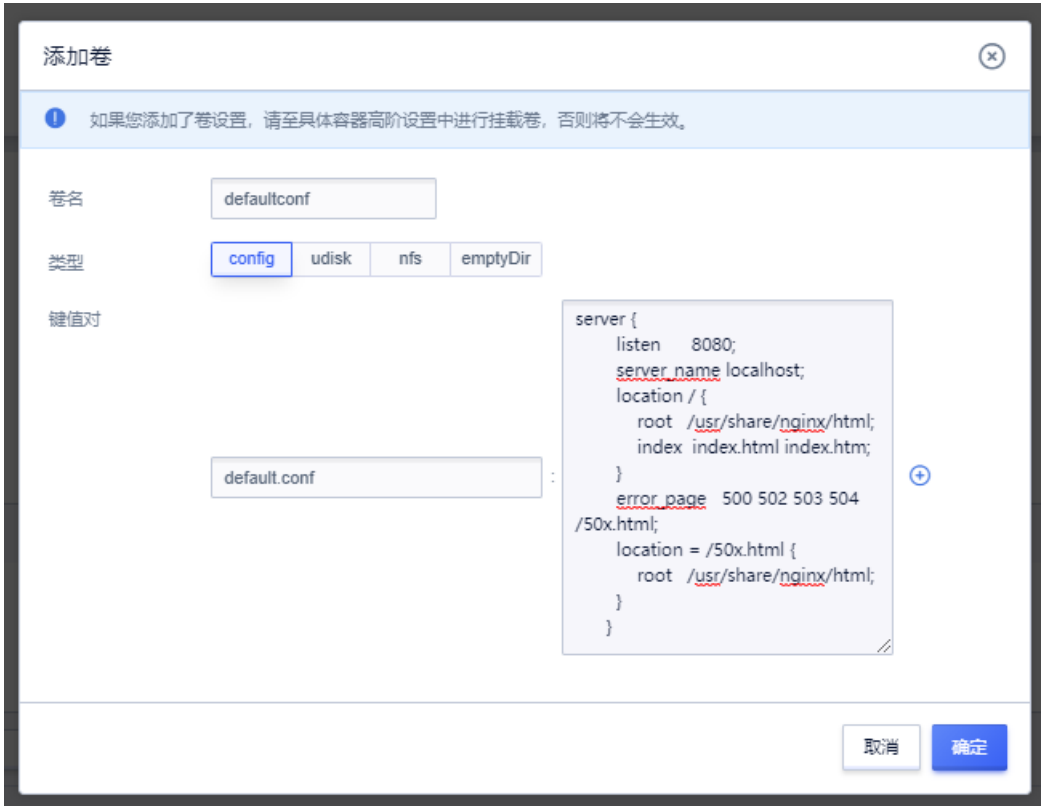
创建详解

通过上面的快速创建我们接下来将通过上面的例子来进行创建操作中的详细设置说明。如果您对于kubernetes比较熟悉，可以查看k8s参考示例对比

卷设置

非必填项

卷设置提供了 config 类型、UDisk 云盘挂载（在 Cube 中使用 UDisk）、NFS 文件存储挂载（在 Cube 中使用 UFS）及 emptyDir 类型。config 类型与Kubernetes中的configMap资源对象一致，提供的是键值对配置文件挂载。



如图所示点击添加卷设置，填入如下信息然后确定。

卷名：defaultconf

键值对key：default.conf

键值对value：

```
server {
listen 8080;
server_name localhost;
```

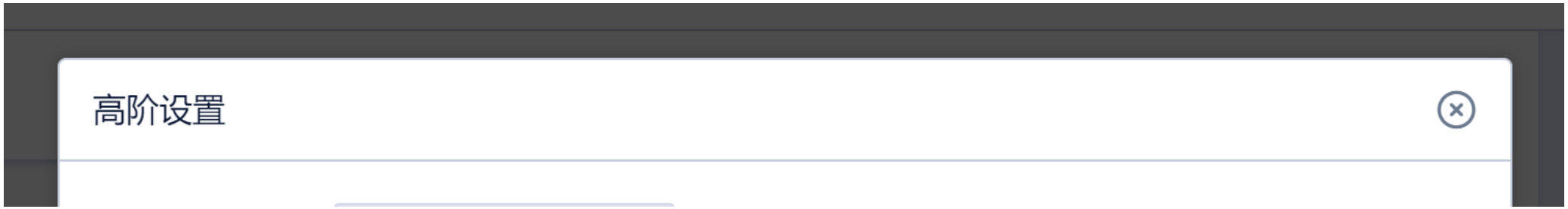
```
location / {
root /usr/share/nginx/html;
index index.html index.htm;
}
error_page 500 502 503 504 /50x.html;
location = /50x.html {
root /usr/share/nginx/html;
}
}
```

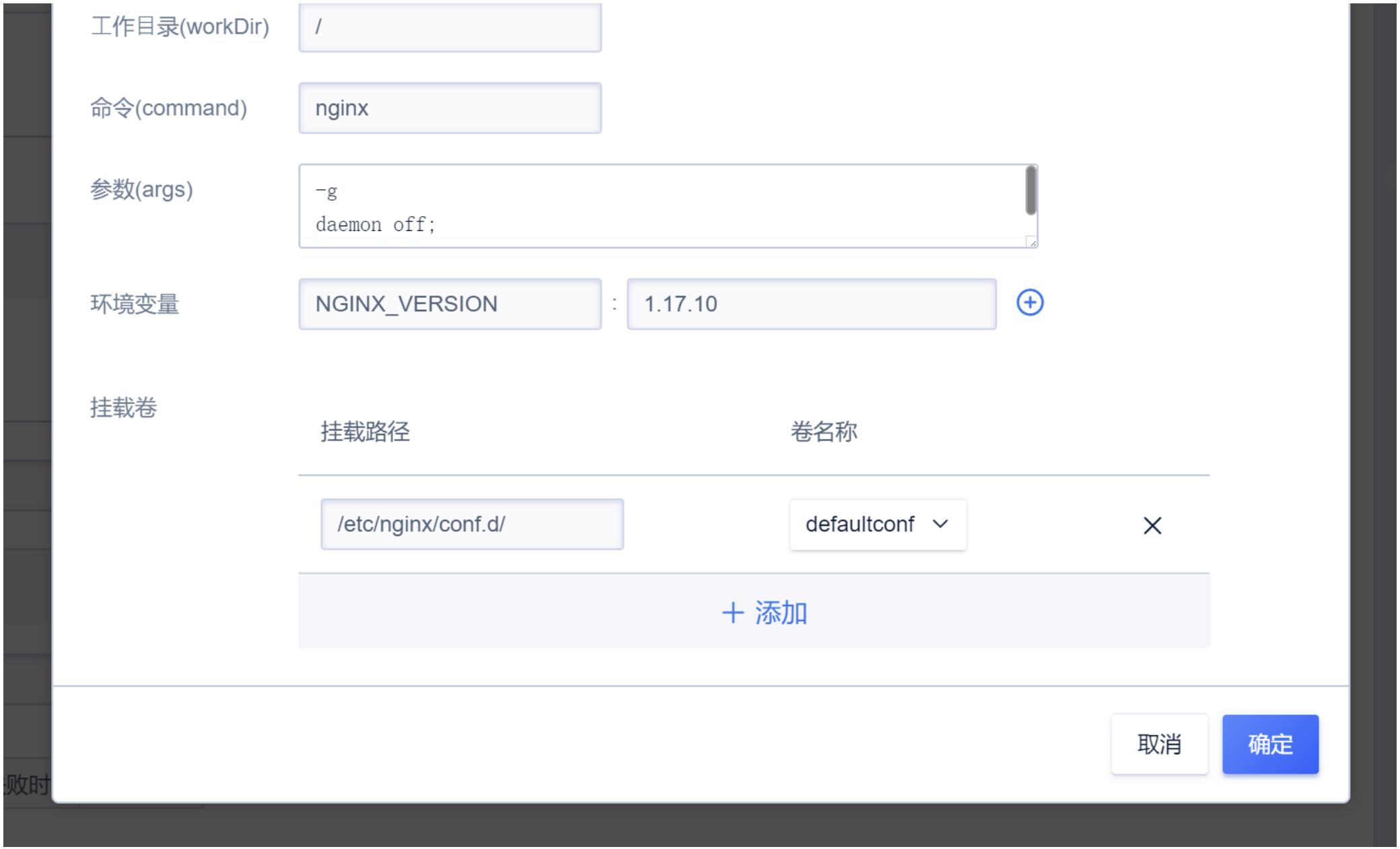
这里可以看到我们修改了nginx的配置文件，将监听端口配置从原80到8080，作为一个配置文件进行创建。

这里只会创建一个卷设置，将不会进行挂载，如需挂载需要在高阶设置中进行卷挂载。

高阶设置

高阶设置中将针对容器进行详细的参数设置。




工作目录(workDir)
/
命令(command)
nginx
参数(args)
-g
daemon off;
环境变量
NGINX_VERSION
:
1.17.10
挂载卷
挂载路径
卷名称
/etc/nginx/conf.d/
defaultconf
+ 添加
取消
确定

工作目录(workDir)

非必填项

这里可以定义容器运行时的工作目录，指定了工作目录之后镜像中所有命令执行都将在工作目录中完成，可以将工作目录指定在Dockerfile中。这里我们使用到的nginx工作目录为根目录。

工作目录: /

命令(command)

非必填项

这里可以定义容器运行时的命令，命令对应的是镜像中程序运行的命令，如果没有设置，将使用容器镜像中的命令。

参数(args)

非必填项

这里可以定义容器运行命令时的参数，如果没有设置，将使用容器镜像中的命令。从Dockerfile中可以看到nginx的参数全部放在了命令中，我们也可以将它拆分成为命令和参数。

这里我们使用到的nginx命令在Dockerfile中为`CMD ["nginx", "-g", "daemon off;"]`，其中可以拆分成

命令(command):

```
nginx
```

参数(args):

```
-g
daemon off;
```

环境变量

非必填项

这里可以定义容器运行时的环境变量，环境变量将在运行的容器中使用env进行查看，可以将环境变量指定在Dockerfile中。这里我们使用的nginx环境变量可以参考Dockerfile中的环境变量。

环境变量name: `NGINX_VERSION`

环境变量value: `1.17.10`

挂载卷

非必填项

这里可以将我们创建的卷设置进行挂载，我们上面创建了一个nginx的配置文件config，这里我们将它挂进我们的容器中。

挂载路径: `/etc/nginx/conf.d/`

卷名称: `defaultconf`

> 注意：如没有创建卷设置，在挂载卷中将选择不到具体的卷名称，请先创建卷设置。

标签

非必填项

这里可以将我们创建的Cube实例打上标签，可以方便我们后续通过标签进行筛选，如下举例。

标签key: `app`

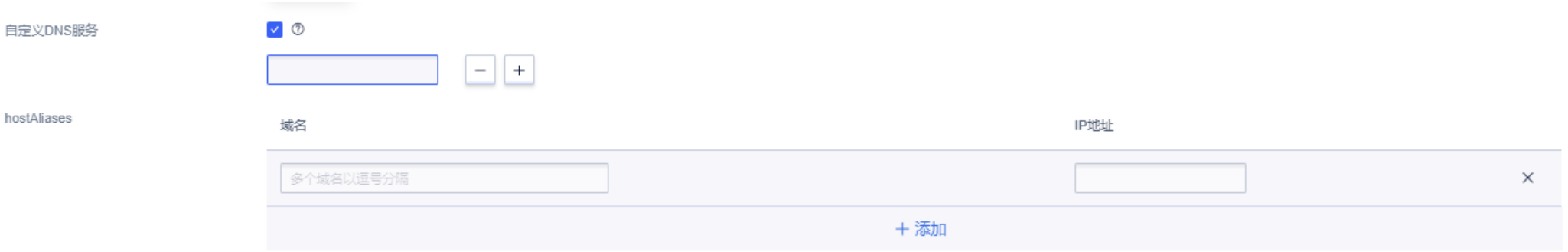
标签value: `nginx`

重启策略

这里可以设置我们创建的Cube实例的重启策略，分别为总是(Always)、失败时(OnFailure)、从不(Never)。

自定义 DNS 服务及 HostAliases

为 Cube 实例添加自定义 DNS 服务，如无需自定义，则默认使用 UCloud 内网 DNS 地址；当 DNS 配置不合理的时候，可以通过通过 HostAliases 字段向 Cube 实例的 /etc/hosts 文件中添加条目，覆盖对主机名的解析。



自定义网络

- 您所创建的Cube实例的网络位置将存在于具体的一个VPC的子网里。
- 您可以根据您镜像程序的需要选择是否绑定外网IP和选择对应的防火墙设置。

自建镜像仓库支持

容器镜像封装了应用代码，是用户的重要资产之一，出于强安全性的考虑，部分用户在容器应用的使用过程中有使用自建镜像仓库的需求。

Cube 支持拉取同一 VPC 下的自建镜像仓库，丰富了使用场景，确保用户镜像和代码安全。

控制台创建 / 修改 Cube 实例

在控制台创建/修改 Cube 及 Deployment 时，在容器「镜像设置」页面，选择自建镜像仓库，并输入镜像详细地址。

输入镜像地址后，在「自建镜像仓库」栏，输入自建镜像仓库详细信息，字段说明如下，当前只支持基于 UCloud 同一主账号下云主机搭建的镜像仓库。

字段	说明
镜像地址	包含镜像名及版本号的完整镜像拉取地址， 如 core.harbor.domain/nginx:test2
所属 VPC	自建镜像仓库所在 VPC，如 VPC 在同一项目下，可直接选择； 如存在跨项目情况，需要输入对应的 VPC ID，如 uvnet-xxxxxxxx
仓库地址	仓库所在云主机的内网 IPv4 地址， 如 10.9.87.17
用户名	镜像仓库用户名，非必填
密码	镜像仓库密码，非必填

其他创建流程均不变，请参考快速创建及创建详解

API 创建 / 修改 Cube 实例

在通过 API 创建 / 修改 Cube 及 Deployment 时，需要在相应的 pod yaml 文件中 spec 字段下添加 imagePullSecrets 字段，如下：

```
imagePullSecrets:
- username: 'zth' # 镜像仓库用户名，非必填
password: 'Harbor' # 镜像仓库密码，非必填
registryserver: 'core.harbor.domain' # 镜像仓库域名
registryaddr: '10.9.87.17' # 仓库所在云主机的内网 IPv4 地址
vpcId: 'uvnet-xxxxxxxx' # 自建镜像仓库所在 VPC 的 VPC ID
```

详细 API 请参照Cube API 文档

第三方镜像仓库支持

Cube支持拉取外部镜像，前提是您的镜像满足如下要求：

- 镜像仓库可以通过外网访问
- 为了安全起见，仅支持HTTPS协议，并且仓库必须有合法的域名
- 镜像支持linux_amd64架构

控制台创建 / 修改 Cube 实例

您可以在容器的镜像设置页面选择第三方镜像仓库以实现拉取外部镜像：

镜像地址必须填写完整,不支持简写,例如nginx:latest需要改为docker.io/library/nginx:latest。

受限于带宽影响，外部镜像拉取速度可能会很慢，甚至可能出现失败（特别是DockerHub官方镜像），如果您希望快速拉取镜像，还是建议将镜像缓存到UHub中。

API 创建 / 修改 Cube 实例

在通过 API 创建 / 修改 Cube 及 Deployment 时,需要在相应的 pod yaml 文件中 spec 字段下添加 imagePullSecrets 字段,如下:

```
imagePullSecrets:
- registry_type: external # 必填,表示这是一个外部镜像
username: 'zth' # 镜像仓库用户名,非必填
password: 'Harbor' # 镜像仓库密码,非必填
```

详细 API 请参照Cube API 文档

k8s参考示例对比

如果您对于kubernetes比较熟悉，可以查看对比Cube参数与Kubernetes中的对照关系。

```
## 卷设置/ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
name: defaultconfig # 卷名
data:
default.conf: | # 键值对key: value
server {
listen 8080;
server_name localhost;
location / {
root /usr/share/nginx/html;
index index.html index.htm;
}
error_page 500 502 503 504 /50x.html;
location = /50x.html {
root /usr/share/nginx/html;
}
}
```

```
---
## Cube实例
apiVersion: v1
kind: Pod
metadata:
name: cube # 容器组名称
labels:
app: nginx # 标签key: value
spec:
containers:
- name: cube01 # 容器名称
image: uhub.service.ucloud.cn/ucloud/nginx:1.17.10-alpine #容器镜像
env:
- name: NGINX_VERSION # 容器环境变量name: value
value: 1.17.10
workingDir: / # 工作目录
command: ["nginx"] # 命令
args: ["-g", "daemon off;"] # 参数
ports:
- name: http
containerPort: 8080
volumeMounts: # 挂载卷
- name: nginx-defaultconfig
mountPath: /etc/nginx/conf.d/
volumes:
```

```
- name: nginx-defaultconfig
configMap:
name: defaultconfig
```

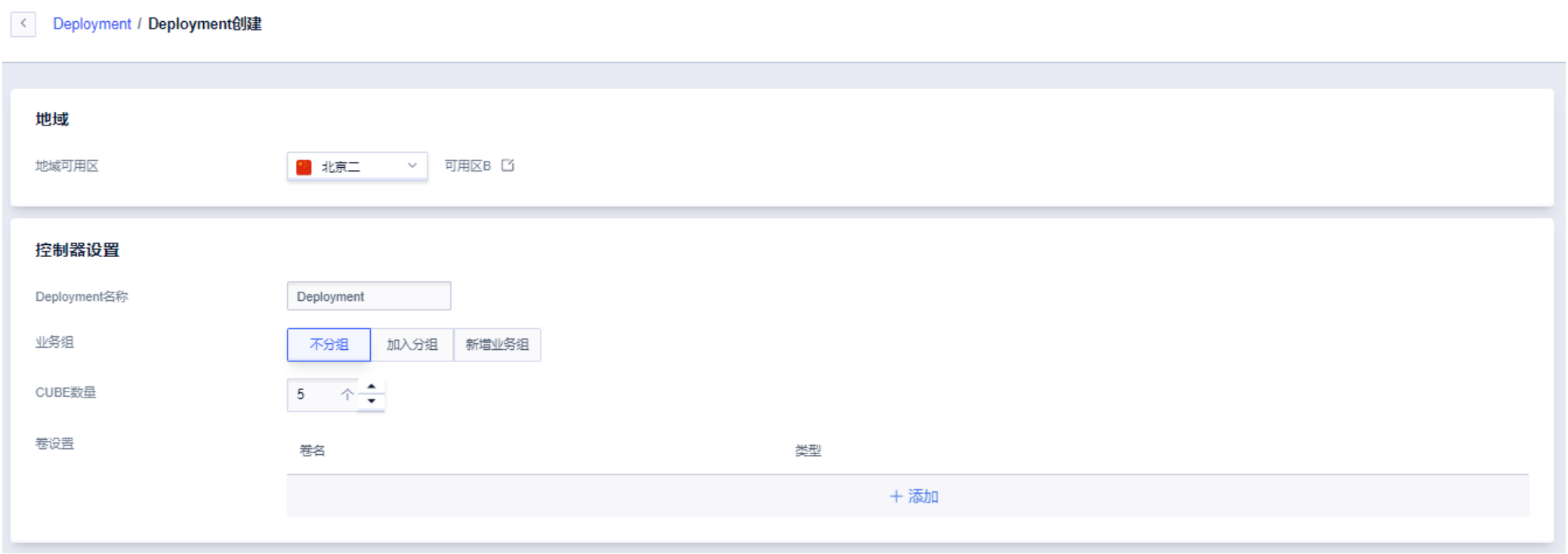
批量创建 Cube 实例

Cube Deployment 功能与原生 K8S 中的 Deployment 类似，实现了对 Cube 实例的批量创建、修改、删除，进一步降低了容器实例的管理和运维成本。

快速通过 Deployment 批量创建 Cube 实例

1. 基础配置

点击 Deployment 页面「创建容器组」，进入 Deployment 创建页面，选择所需的可用区，输入自定义 Deployment 名称，并选择当前 Deployment 中所需的 Cube 实例的数量（同一个 Deployment 中最多可以包括 50 个 Cube 实例）及所需加入的业务组。



2. 卷设置

非必选。在卷设置中点击「添加」，为 Deployment 中的 Cube 实例配置相应的存储卷。

Deployment 目前支持挂载 *UDisk 云盘*、*config 类型*（类比 Kubernetes 中的 configMap 资源对象，参见在 Cube 中使用 Config）及 *emptyDir 类型*。

Deployment 提供多种 UDisk 类型支持，包括 RSSD 云盘、SSD 云盘及普通 SATA 云盘 UDisk 硬盘后，仍需通过容器高阶设置中的「挂载卷」对云盘进行挂载，挂载方法参见在 Cube 中使用 UDisk。

3. 容器设置

选择 Cube 实例后端平台（Intel/AMD，实际可选平台视可用区略有不同），并为 Cube 实例中的每一个 container 容器配置相应的 CPU / 内存资源及容器镜像，一个 Cube 实例中最多支持 3 个容器。

容器镜像支持 *UHub 镜像*(支持同一项目下镜像仓库,及 UCloud 公开仓库)、*缓存镜像*、*共享镜像*(仅支持Uhub镜像仓库公开分享镜像),对同一账号下 VPC 内自建镜像仓库的支持也即将上线。

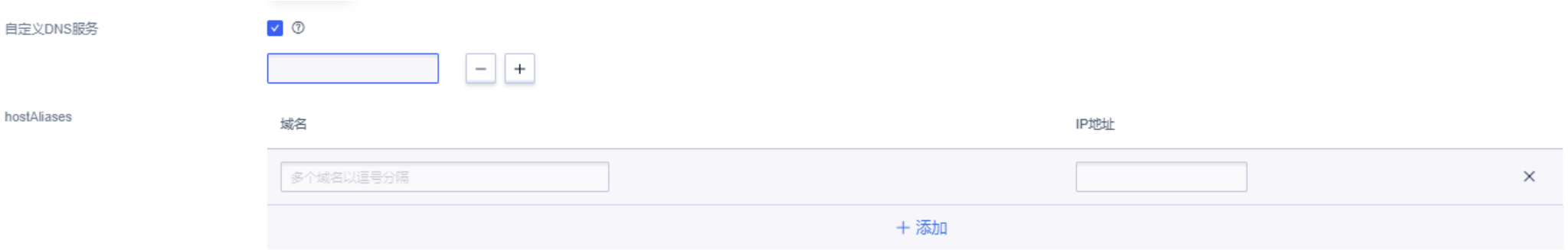
高阶设置包括*工作目录(workDir)*、*命令(command)*、*参数(args)*、*环境变量*的配置,及存储卷的挂载,详见Cube 创建详解。

4. 重启策略

为 Deployment 中的 Cube 实例配置重启策略,分别为总是(Always)、失败时(OnFailure)、从不(Never)。

5. 自定义 DNS 服务及 HostAliases

为 Deployment 中的 Cube 实例添加自定义 DNS 服务，如无需自定义，则默认使用 UCloud 内网 DNS 地址；当 DNS 配置不合理的时候，可以通过 HostAliases 字段向 Cube 实例的 /etc/hosts 文件中添加条目，覆盖对主机名的解析。



6. 自定义网络

- 同一 Deployment 中的 Cube 实例将位于同一个 VPC 子网，请合理配置子网大小。
- 根据程序需要选择是否绑定外网弹性 IP 和选择对应的防火墙设置。

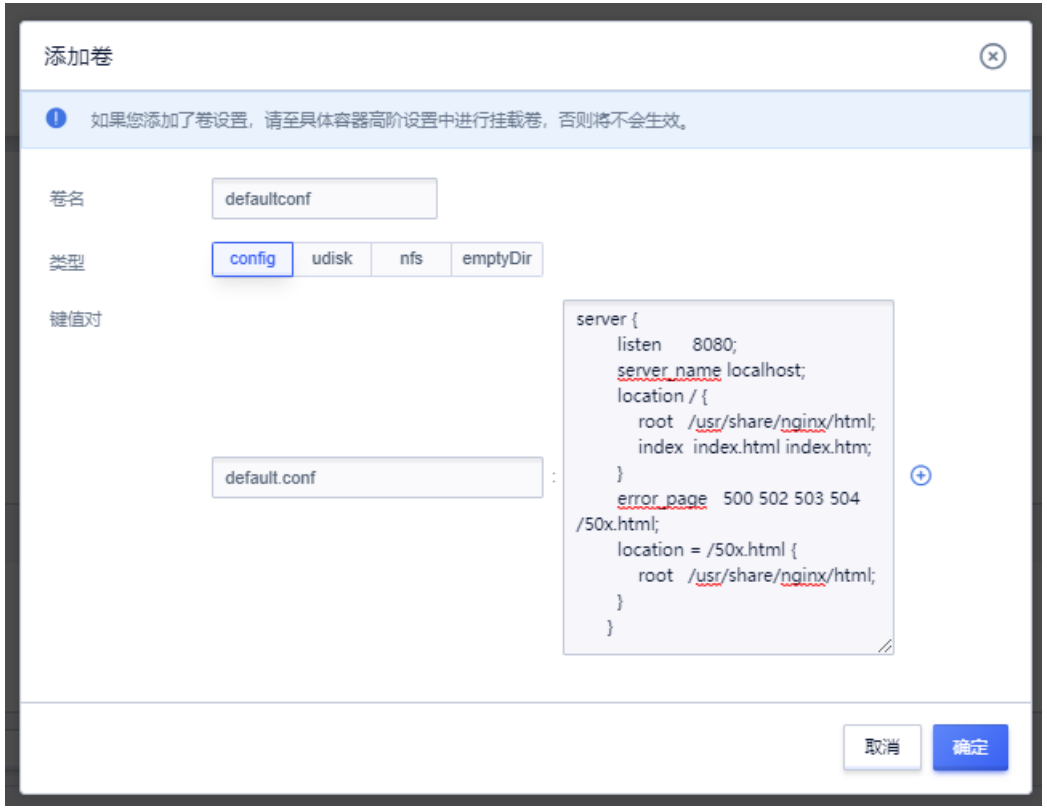
在Cube中使用Config

Config提供的是键值对配置文件挂载的功能，与Kubernetes中的configMap资源对象一致。

Config类型的卷设置一般用于应用和配置分离的模式，不同的运行环境对应不同的配置文件。

添加卷

进入创建Cube实例的页面，点击卷设置。



在添加卷中输入卷名，选择config类型，支持多对键值对的config文件和多行值的文件挂载到容器内部。

以Nginx为例，需要设置监听不同的端口，可以填入如下信息：

卷名：`defaultconf`

键值对key：`default.conf`

键值对value：

```
server {
listen 8080;
server_name localhost;
location / {
root /usr/share/nginx/html;
index index.html index.htm;
}
error_page 500 502 503 504 /50x.html;
location = /50x.html {
root /usr/share/nginx/html;
}
}
```

这里可以看到我们修改了nginx的配置文件，将监听端口配置从原80到8080，作为一个配置文件进行创建。

挂载卷

点击高阶设置，选择挂载卷。这里可以将我们创建的卷设置进行挂载，我们上面创建了一个nginx的配置文件config，这里我们将它挂进我们的容器中。

挂载卷有3个选项，挂载路径、子路径(subpath)、卷名称。

- 挂载路径，填写文件具体要挂载到某个路径。
- 子路径(subpath)，针对配置来说，挂载时填写子路径，将不会覆盖挂载路径下其他的文件。
- 卷名称，选择已经创建的卷设置。

以Nginx为例，挂载config卷，可以填入如下信息：

挂载路径:`/etc/nginx/conf.d/default.conf`

子路径:`default.conf`

卷名称:`defaultconf`

此种填写方法，将不会覆盖/etc/nginx/conf.d/路径下的其他文件。

反之，还可以在挂载路径直接填写/etc/nginx/conf.d/，不填写子路径，将会覆盖路径下其他文件。

在Cube中使用UFS

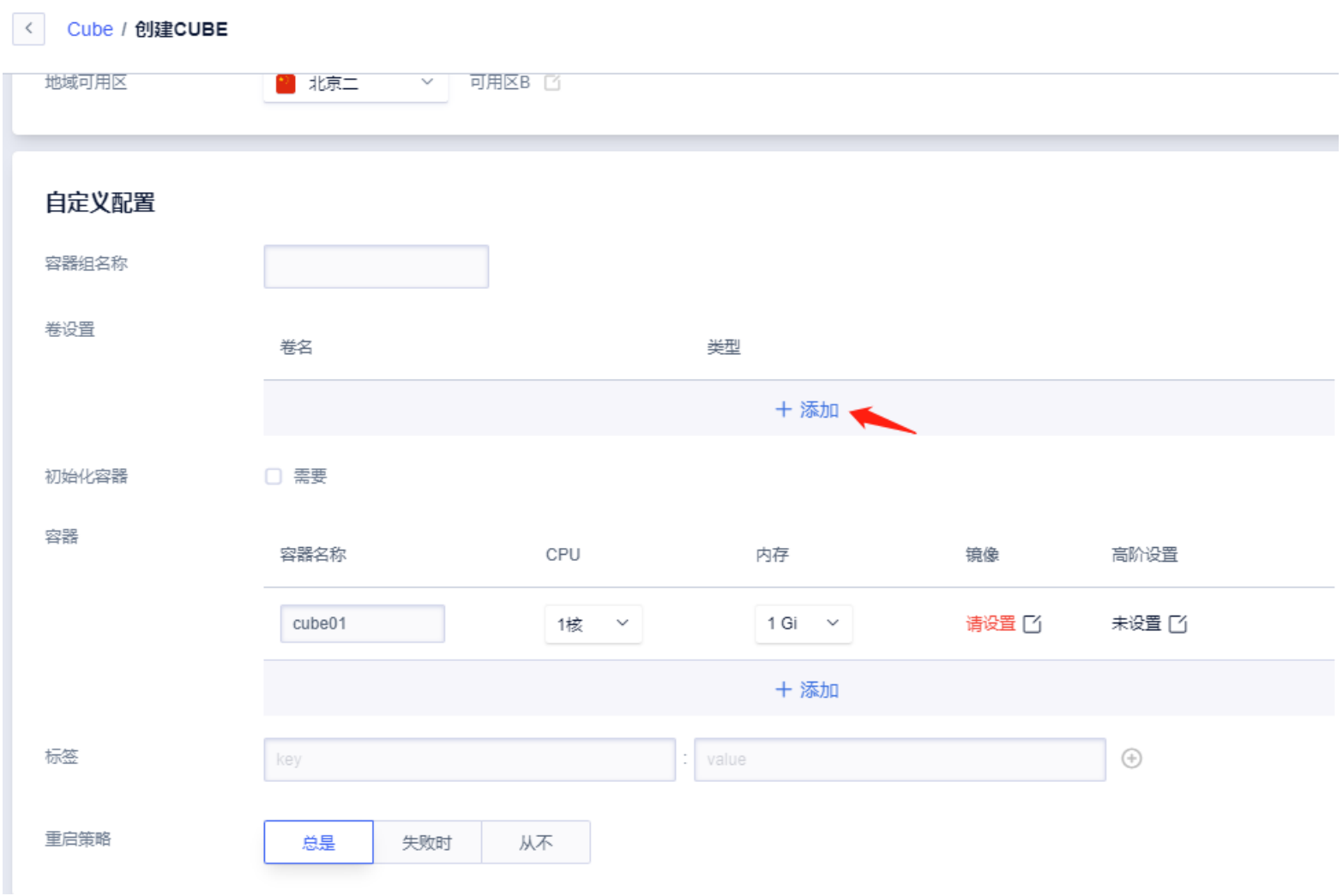
前置条件

需要在UFS产品页面购买UFS实例并设置好挂载点，如没有创建UFS在使用中请根据提示进行创建。操作后在Cube页面点击刷新即可。

添加卷

已经创建好UFS实例和挂载点后，根据以下流程可以挂载UFS进行使用。

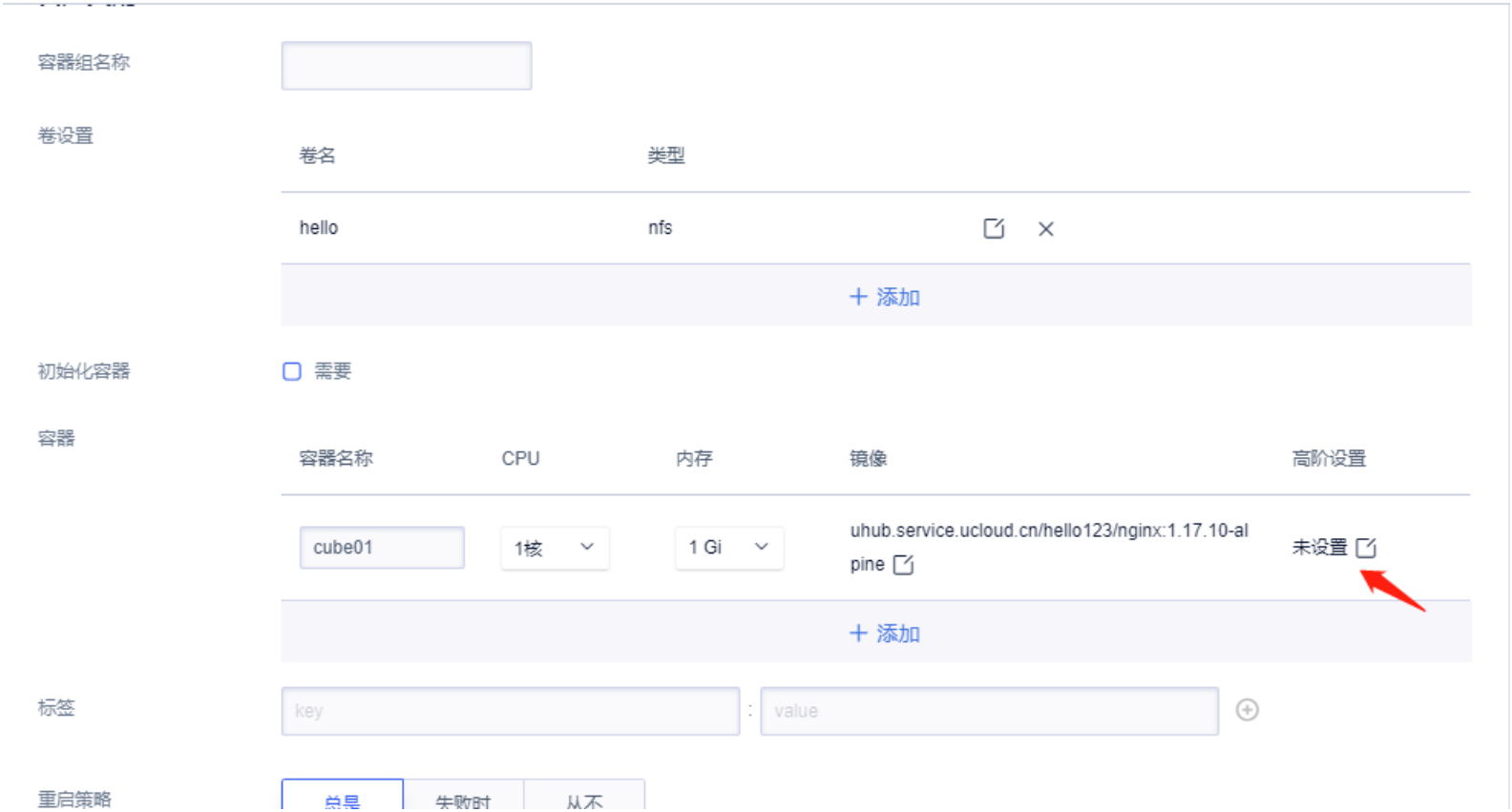
进入创建Cube实例的页面，点击卷设置。



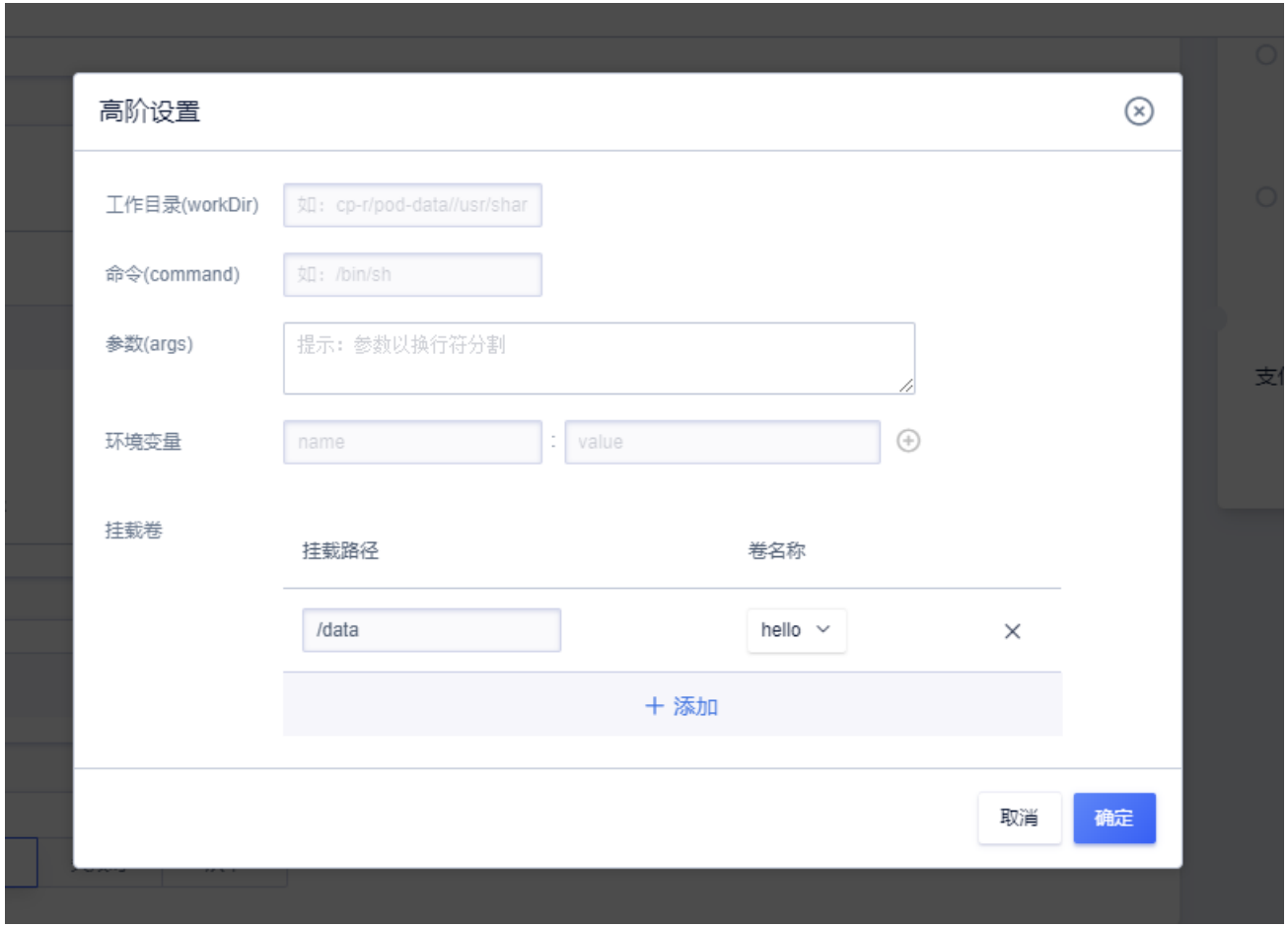
在添加卷中输入卷名，选择nfs类型，并选择UFS存储方式，根据下拉选项选择UFS对应的名称和挂载点并保存。

挂载卷

点击容器的高阶设置。



在高阶设置中点击挂载卷，选择我们刚刚创添加的卷设置，并输入在容器内需要挂载的路径，这里演示填写为/data。

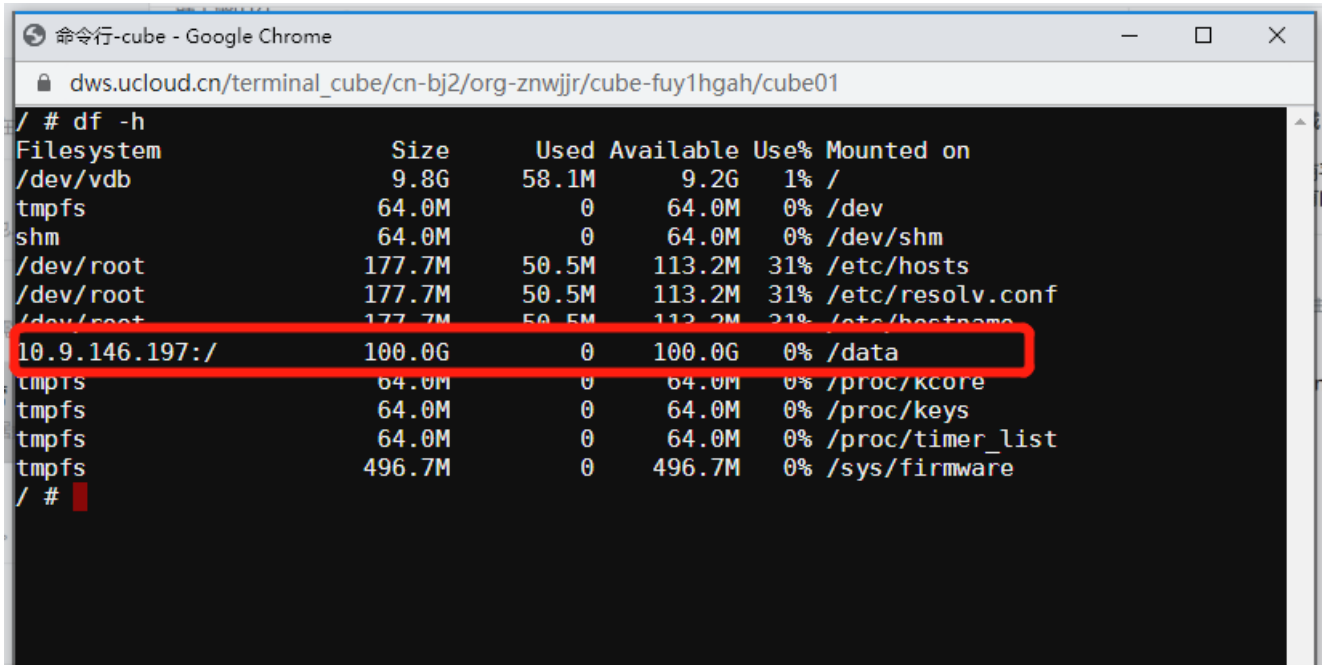


完成以上步骤创建Cube容器实例即可挂载使用UFS文件存储进行数据持久化了。

查看挂载是否成功

我们使用Cube概览页面的登录按钮进入容器。

输入`df -h` 查看到我们的文件存储已经挂在了容器的`/data`目录下。



Cube支持除UFS外，也默认支持其他nfs协议的文件存储，用户如需使用需要自行配置。

在Cube中使用UDisk

目前快杰版Cube只支持RSSD类型的UDisk挂载，具体性能指标请查看快杰Cube云盘性能

前置条件

1. 需要在Cube云盘管理购买UDisk云盘。

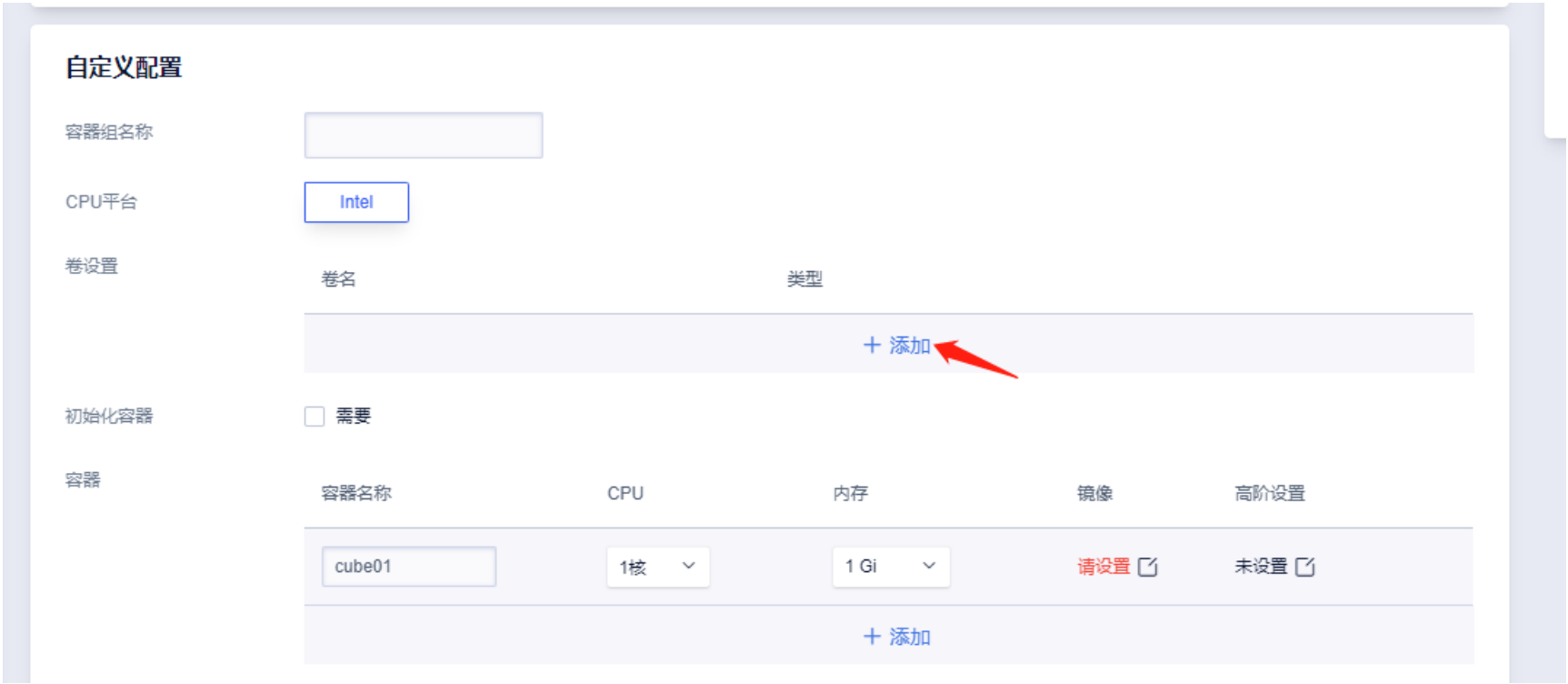
注意事项

1. 块存储仅支持单点挂载，且不支持跨可用区挂载，如需要多点挂载可以选择UFS，
2. 快杰版Cube仅支持RSSD UDisk挂载，
3. 使用克隆功能时，无法克隆UDisk的挂载信息，
4. RSSD UDisk可以在主机页面购买或Cube页面购买，受限于RDMA网络限制，推荐在Cube页面购买。

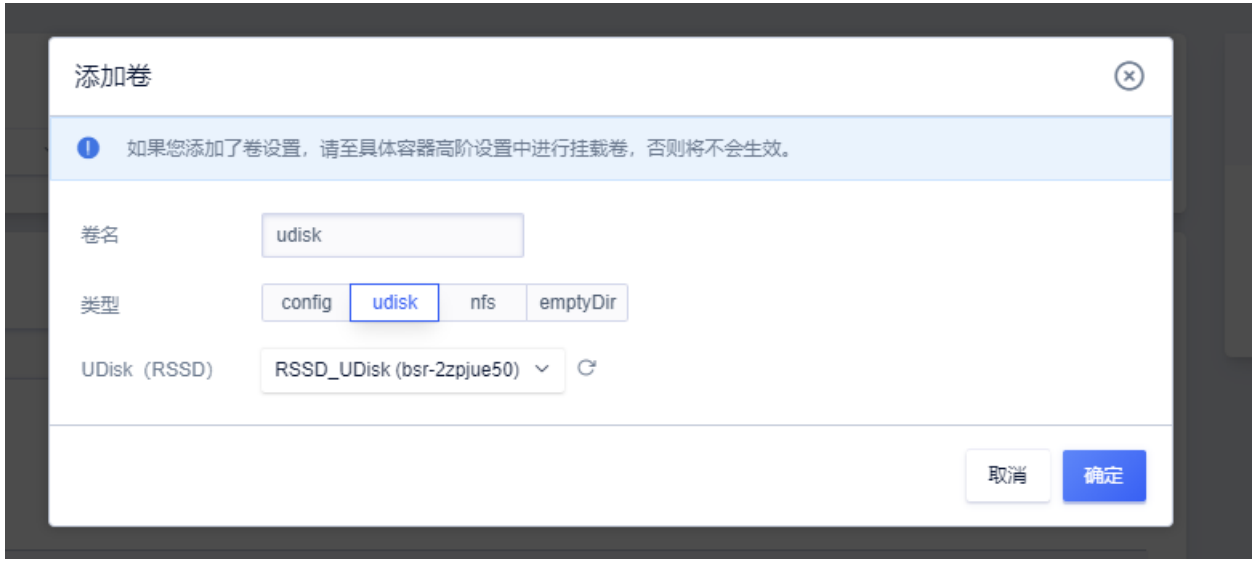
添加卷

已购买好RSSD UDisk云盘后，根据以下流程可以挂载UDisk进行使用。

进入创建Cube实例的页面，点击卷设置。

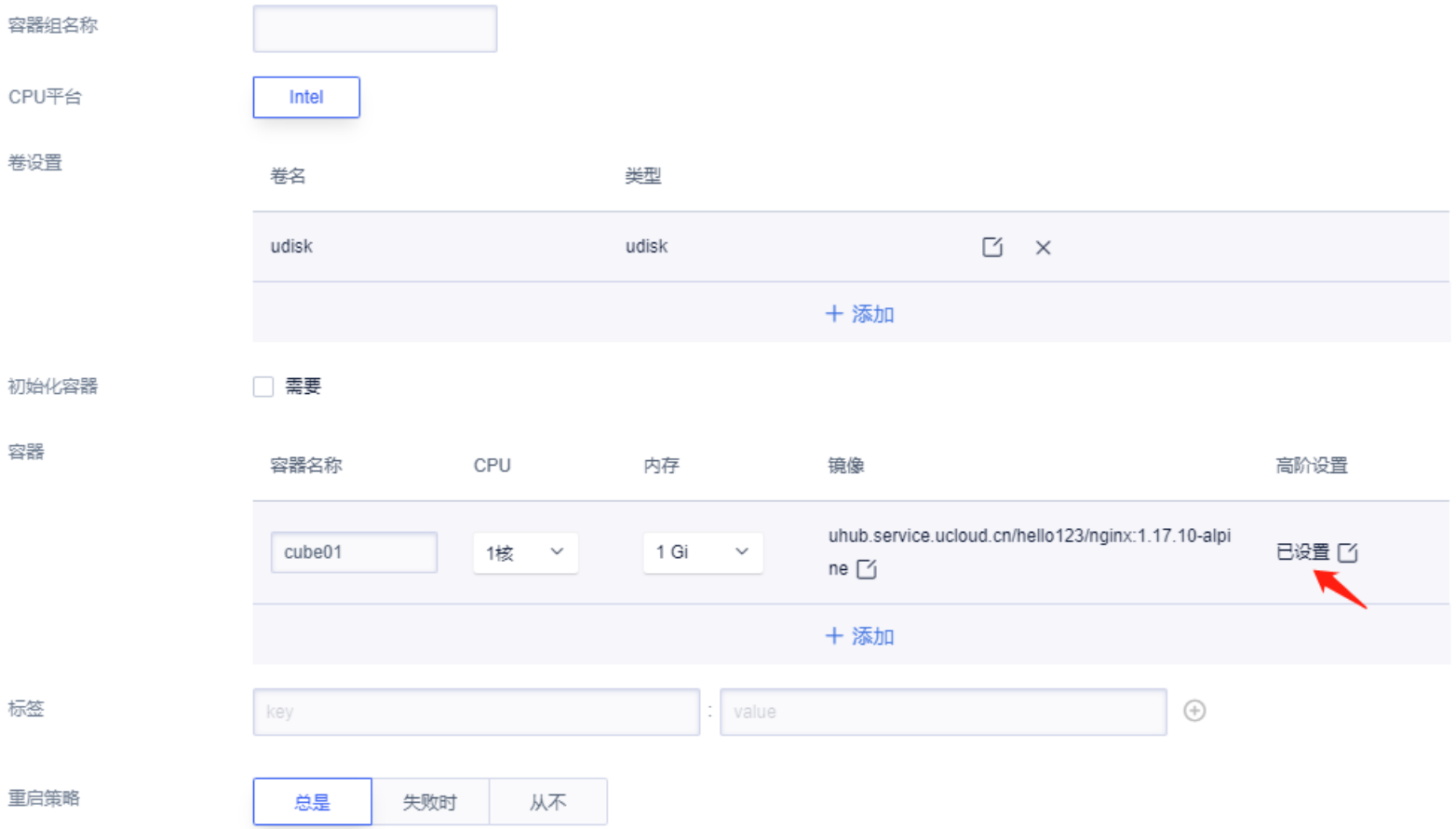


在添加卷中输入卷名，选择UDisk类型，根据下拉选项选择对应的UDisk云盘并保存。

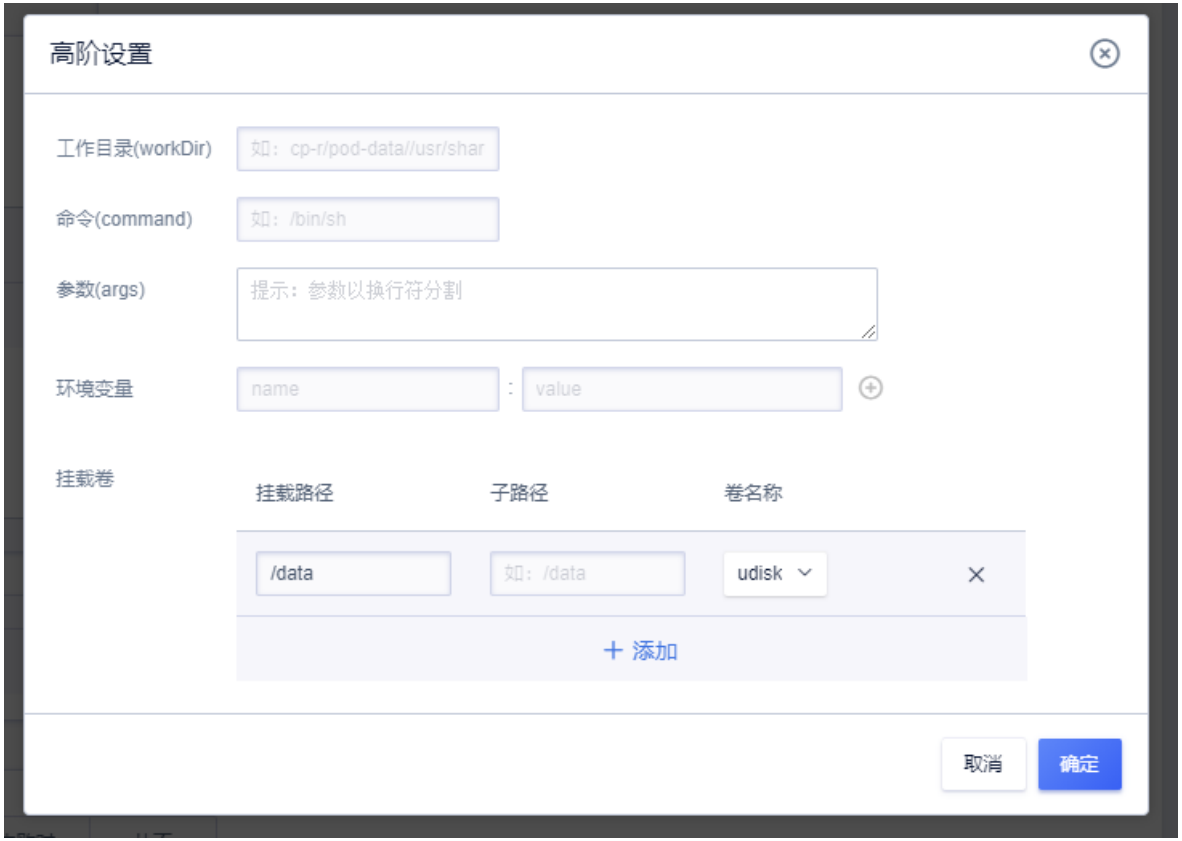


挂载卷

点击容器的高阶设置。



在高阶设置中点击挂载卷，选择我们刚刚添加的卷设置，并输入在容器内需要挂载的路径，这里演示填写为/data。

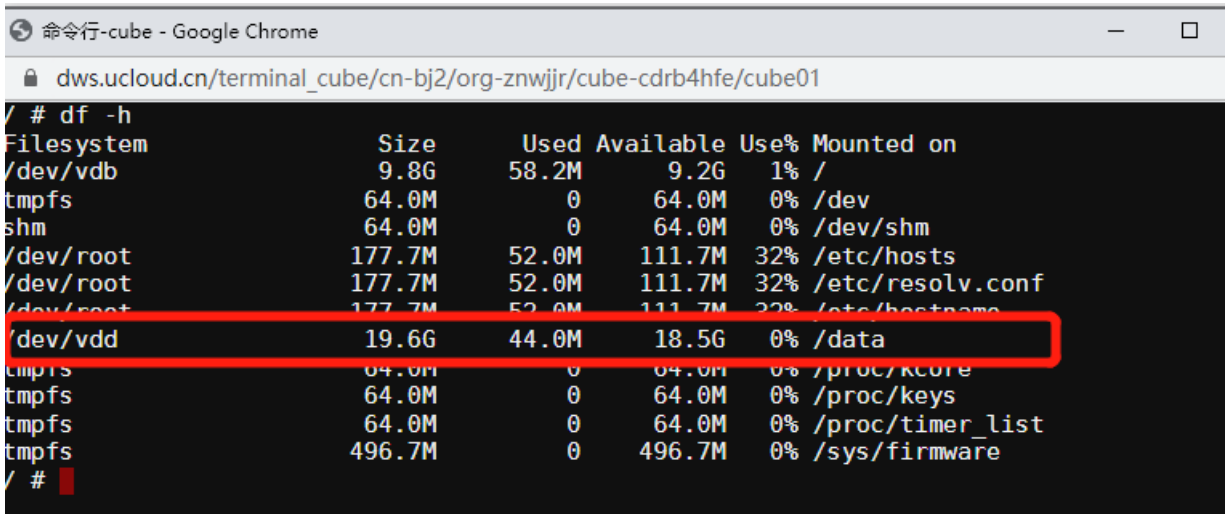


完成以上步骤创建Cube容器实例即可挂载使用UDisk块存储进行数据持久化了。

查看挂载是否成功

我们使用Cube概览页面的登录按钮进入容器。

输入`df -h` 查看到我们的块存储已经挂在了容器的`/data`目录下。



使用Cube创建带SSH服务的CentOS容器

考虑容器运行无法持久化存储数据，不建议将Cube容器作为云主机使用，如需存储数据可在Cube容器实例创建时挂载云盘到指定数据存储路径。

选择镜像

创建Cube时选择容器镜像为Cube_Lab仓库下的centos-ssh镜像，版本目前支持7.8.2003。

设置密码

点击打开容器高阶设置，在环境变量输入PASSWD:youpassword设置您的密码。

选择绑定外网弹性IP，确认并进行创建。

测试ssh登录

接下来用我们使用我们创建时的EIP尝试ssh登录操作，如下图。

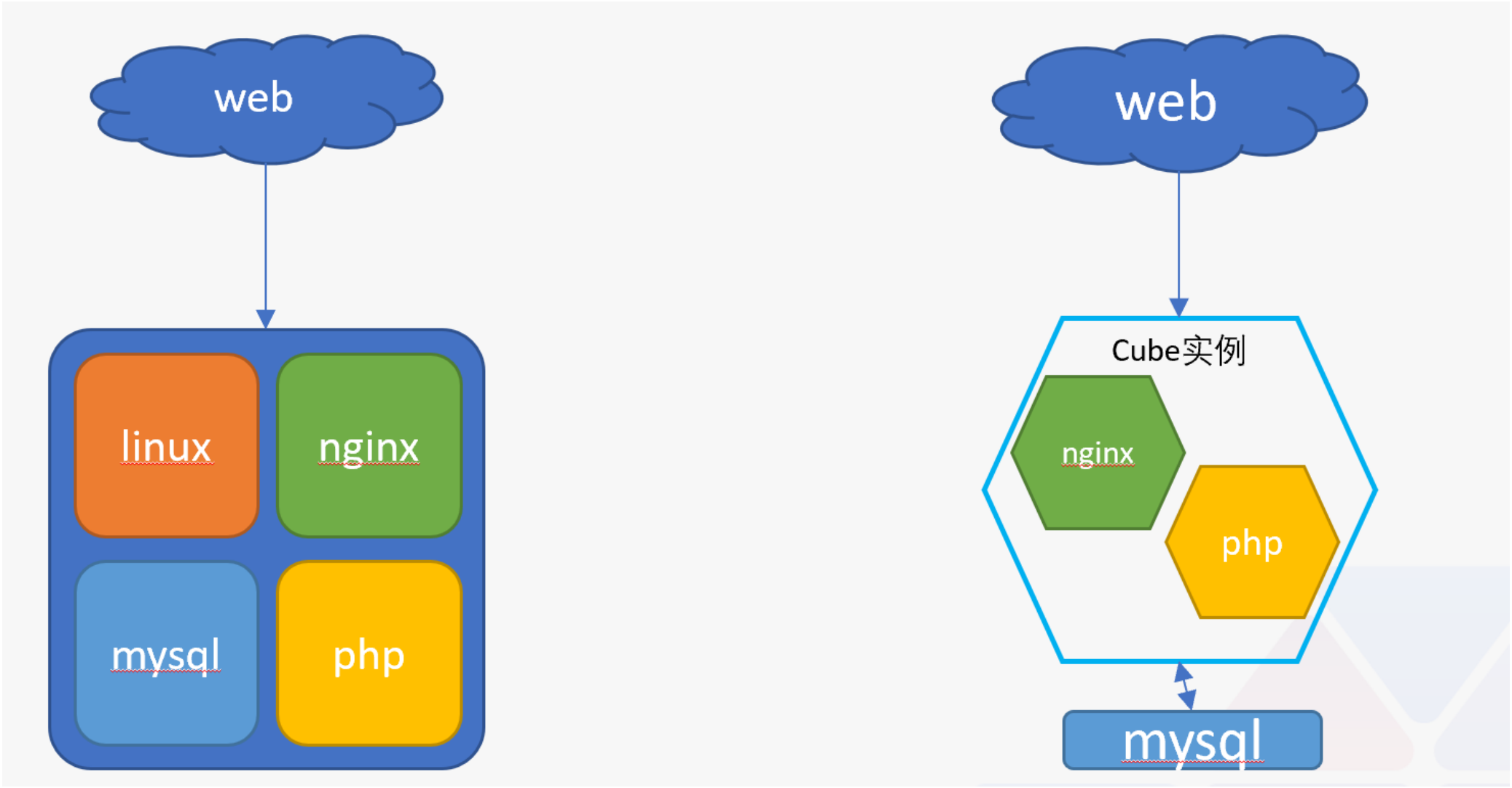
PHP应用的高可用部署

今天我们将一起通过一个实践将一个PHP网站运行在Cube上！

原LNMP环境改造

LNMP环境是我们非常常见的PHP网站运行的环境，我们将其中的功能进行区分，NGINX负责转发、PHP负责程序运行，原PHP程序运行时基本是运行在这两个程序的共同目录下，NGINX负责将用户请求的具体网址进行后缀筛选转发给PHP运行接口处理。

由于容器推荐是单进程的，所以在Cube中我们将NGINX和PHP拆成了2个容器，Cube有容器组的概念，所以我们将2个容器放到了一个组里。



弹性使用

这里我们选择添加2个容器，分别是NGINX和PHP，NGINX镜像我们选择官方镜像即可，PHP镜像我们最好选择是具体PHP程序的镜像或者基于PHP-FPM的基础镜像安装程序依赖的PHP组件。

我们可以根据应用设定具体使用资源值，由于NGINX转发动作可能资源使用很小，参照K8S的Pod（容器组）资源对象，在这里Cube容器组内的容器是可以进行资源共享使用，如果PHP资源不够的话会占用一部分NGINX容器的资源，达到容器组的资源最大化使用。

配置分离

前面说到我们将2个容器放进了1个容器组，参照K8S的Pod(容器组)资源对象，容器组内容器之间是可以本地调用的，所以在我们原来的LNMP环境下NGINX的配置是可以直接在容器中使用的，这里我们用配置卷挂载而不是构建在镜像中，方便我们维护这个配置。

我们在卷设置中添加config类型的卷，并填入NGINX配置default.conf

在NGINX容器的高阶设置中添加挂载路径/etc/nginx/conf.d/default.conf和子路径default.conf，选择之前我们创建好的卷，然后保存，这里子路径的设置是为了我们在该挂载路径下只覆盖具体的default.conf文件，而不会将整个文件夹下覆盖。

这里说两个需要注意的点：

1. Cube实例是一个容器组，所以NGINX配置几乎不用修改，依然本地访问PHP
2. 配置等卷设置，都需要在容器的高阶设置中挂载后才会生效

附一个NGINX的配置卷例子

```
server {
```

```
listen 80;
server_name localhost;

location / {
root /usr/share/nginx/html/wordpress;
index index.html index.php;
}
error_page 500 502 503 504 /50x.html;
location = /50x.html {
root /usr/share/nginx/html;
}
location ~ \.php$ {
fastcgi_pass localhost:9000;
fastcgi_index index.php;
fastcgi_buffers 16 16k;
fastcgi_buffer_size 32k;
fastcgi_param SCRIPT_FILENAME /var/www/html/wordpress/$fastcgi_script_name;
#fixes timeouts
fastcgi_read_timeout 600;
include fastcgi_params;
}
}
```

存储分离

容器是无状态的将不会保存任何数据，所以我们这里将我们的PHP程序放在了我们的共享文件存储(UFS)上，然后我们同配置卷一样，将存储卷创建并在容器中挂载，这里我们需要对2个容器进行挂载，因为我们还是要达到NGINX和PHP容器读取的程序是同一份。

我这里选择的是我们的例子WordPress的php-fpm镜像，为什么我们用了业务程序的PHP镜像我们还需要将业务程序挂在出来?因为基础php-fpm镜像仍然需要安装php程序依赖，这里为保证PHP程序正常运行所以这里选择了官方的WordPress镜像以保证程序运行依赖完整，挂载出来是为了PHP运行本地的程序变化可以得到同步。

这里说两个需要注意的点：

1. 存储卷依然需要在容器中挂载。
2. 一个存储卷根据特性可以挂载给一个容器组内多个容器使用，根据UFS文件存储特性，支持多容器组挂载。

3. 如果UFS内没有程序，我们可以通过初始化容器，将程序存放到UFS上，可参考以下例子。

附一个初始化容器的例子：

默认镜像：busybox:1.28

命令：sh

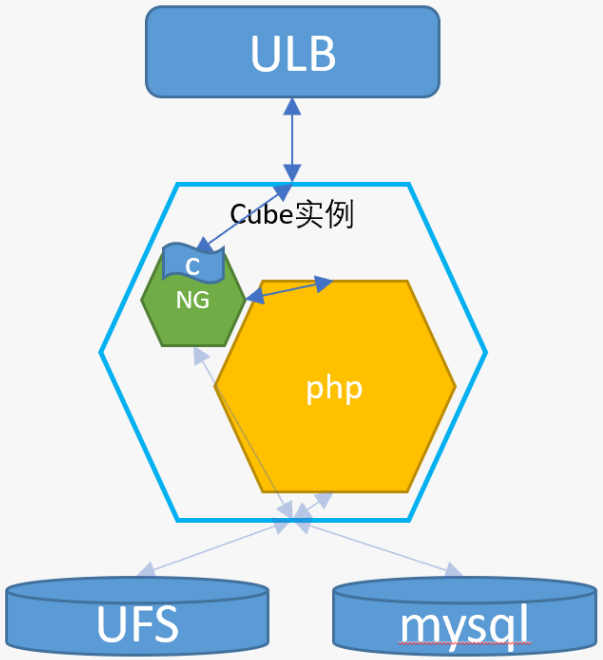
参数：

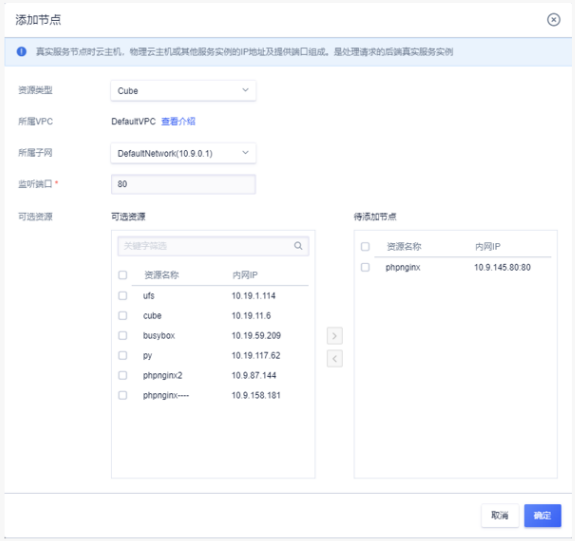
```
-c
wget http://cube.cn-bj.ufileos.com/wordpress-5.4.2.tar && tar -zxvf wordpress-5.4.2.tar && mv /wordpress /data/wordpress
```

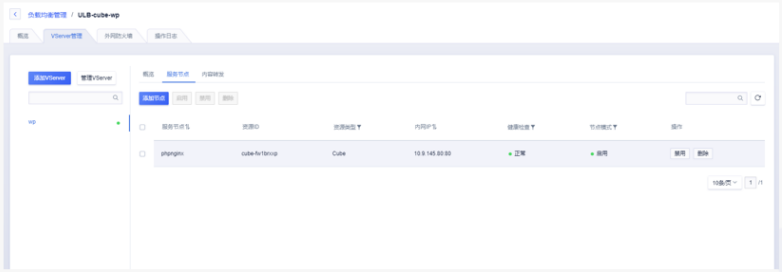
挂载卷：挂载前面创建的UFS卷

负载均衡-ULB

我们创建一个ULB，然后在ULB中添加VServer，选择Cube分类和80端口查询到我们的Cube实例，并选择挂载。







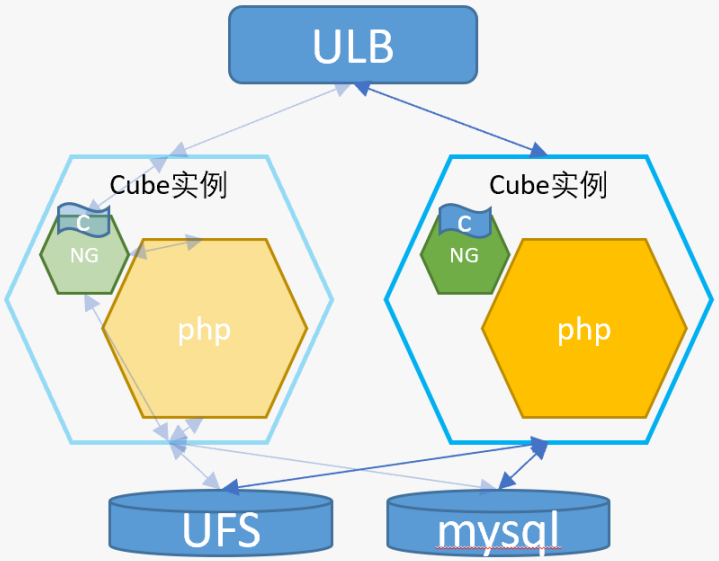
ULB页面关联VServer节点，选择Cube

这里需要注意的点：

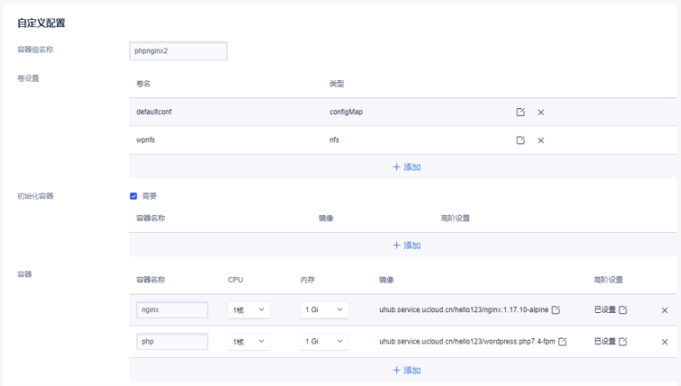
1. 目前只有请求代理型ULB(原ULB7)可以支持选择Cube资源实例进行对接，报文转发的ULB暂时还不支持。

克隆多实例

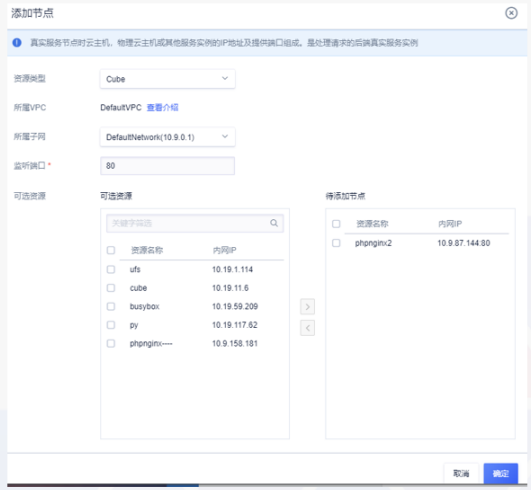
我们刚才创建的仍然是单点的PHP服务，如何才能创建成为一个具备高可用性能的的PHP网站呢?这里可以使用Cube的克隆功能，我们可以快速创建出Cube实例，以WordPress为例大约3秒即可创建完成，然后我们在ULB的VServer中选择我们刚刚创建的Cube实例添加到我们的负载均衡中，至此，一个高可用的PHP网站就部署完成了。



秒级创建多实例



Step1.克隆Cube快速创建



Step2.添加到ULB的后端

关键点

整理一下需要注意的点：

1. 容器最好是单进程的，建议nginx和php拆成2个容器
2. 容器组内的容器是可以本地访问的
3. 容器组内资源是共享使用的
4. 卷设置都需要在具体需要的容器高阶设置中挂载
5. 我们可以利用初始化容器完成一些程序运行所需要的环境文件的初始化工作

运行状态

状态分类

容器组的运行状态与kubernetes中的pod运行状态一致，共有以下几种。

状态	含义
Running	表示容器组正常运行。
Completed	表示容器组中的容器任务已经执行完成，执行任务结束的状态返回。
Error	表示容器组中部分容器运行不正常，正在重试容器启动。
ContainerCreating	容器创建中。
CreateContainerError	镜像中的执行命令或者创建时添加的命令在容器中执行失败。
CrashLoopBackOff	表示容器组启动成功后运行失败，组内部分或全部容器处于不健康的状态。
ErrImagePull	镜像拉取错误，正在进行重试。
ImagePullBackOff	表示使用的镜像仓库设置了密码或者镜像已经被删除，导致镜像拉取失败。
Terminating	表示容器正在结束当前执行程序优雅退出。
Pending	没有足够资源创建容器组。

错误状态排查

Error

容器组中容器运行不正常的一个中间状态，将会很快转为其他状态。

处理方法

1. 查看error下一个状态，查找对应的处理方法。

CreateContainerError

镜像中的执行命令或者创建时添加的命令在容器中执行失败。

处理方法

1. 检查镜像中的执行命令是否正确。
2. 检查创建时添加的命令(command)是否正确，检查完成后重新创建。

CrashLoopBackOff

容器中有部分容器启动成功后运行失败或者退出，常见得错误状态出现情况有：

1. 批处理任务容器镜像，在启动容器组时选择了重启策略为“总是”。
2. 多容器的容器组中，有部分容器无法正常运行或者退出。
3. 多容器的容器组中，有存在监听端口冲突的情况。

处理方法

1. 检查是否为批处理任务容器镜像，如果是请选择重启策略为“从不”，可以正常返回，需重建。
2. 检查容器组中哪些容器没有正常运行，查看启动命令是否正确，需重建。
3. 查看是否为centos、busybox一类镜像，此类镜像由于没有固定进程启动，如是则需要设定启动命令，需重建。
4. 多容器的容器组中，修改已存在的端口冲突，需重建。

ImagePullBackOff & ErrImagePull

使用的镜像仓库设置了密码或者镜像已经被删除，导致镜像拉取失败。

处理方法

1. 检查是否将镜像仓库加密了，如果加密了仓库需要设置打开或者增加镜像拉取密钥，增加密钥需重建。
2. 检查是否将镜像删除了，如删除了，需要重新上传对应镜像和版本。

Terminating

容器正在结束当前执行程序优雅退出。

处理方法

1. 正常情况下等待一会会退出完成，如退出时间超过10分钟，请与我们客服进行联系。

Pending

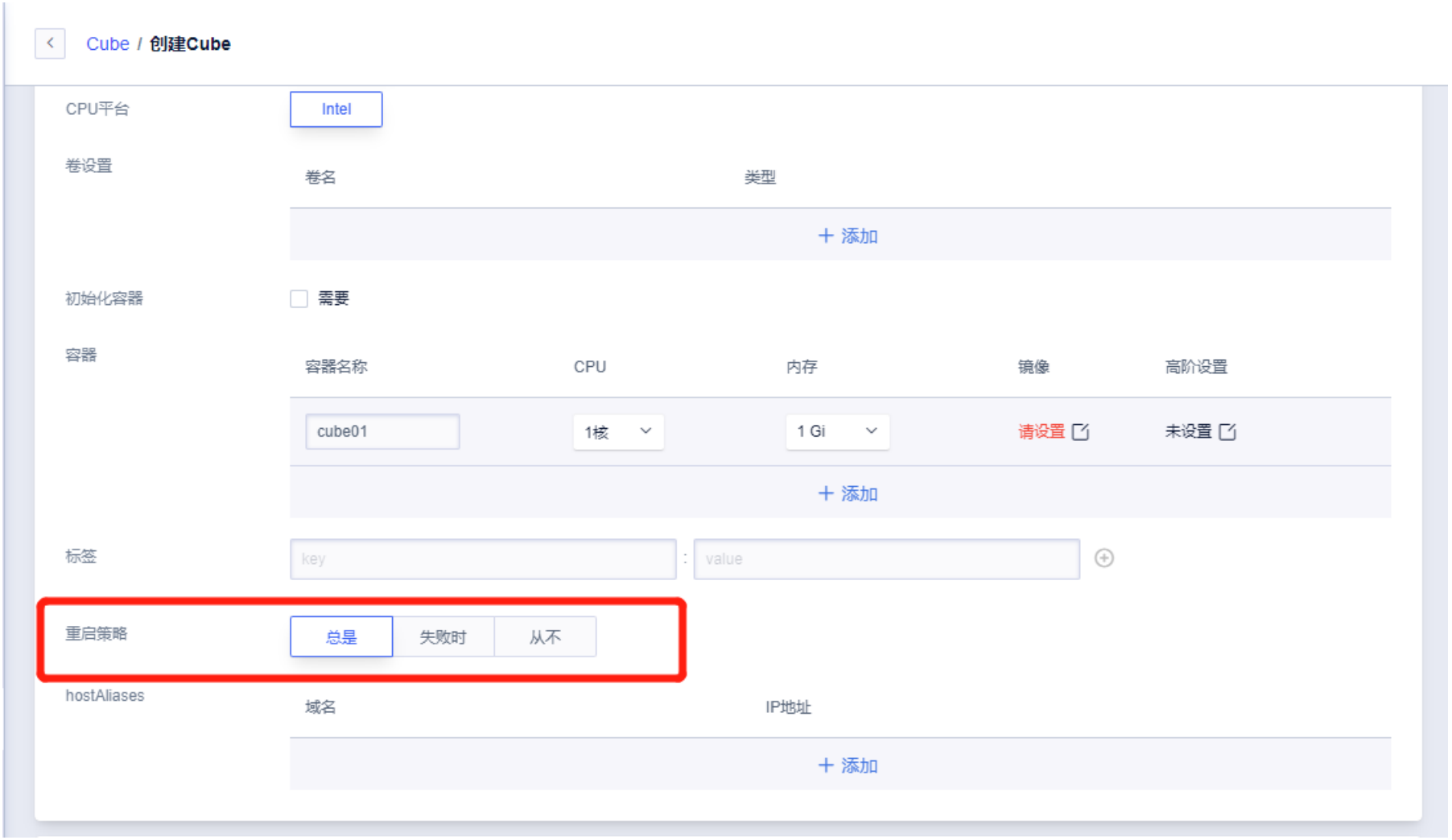
没有足够资源创建容器组。

处理方法

1. 正常情况下不会存在Pending状态，如您遇到Pending状态超过5分钟，请与我们客服进行联系。

容器重启策略

重启策略说明



我们在创建容器组时，这里可以设置我们创建的Cube实例的重启策略，分别为总是(Always)、失败时(OnFailure)、从不(Never)，默认为总是(Always)。

重启策略	含义
总是(Always)	不管容器中程序的类型，当容器处于非running状态时即会发生重启

失败时(OnFailure)	只考虑容器运行故障（OOM、程序问题等）进行重启，当容器执行至完成状态(Completed)时将不会发生重启
从不(Never)	不管容器运行至什么状态，均不会发生重启

重启延时规则

当您的容器运行故障发生重启时（OOM、程序问题等）会触发重启延时规则。当您的容器不停的启动失败，第一次是立即重启，第二次是10s，第三次是20s，第四次是40s，一次类推，最长延时为5分钟（300s）。

当您的容器运行正常10分钟后且期间没有出现问题，重启延时规则将会被重置归零。