

腾讯云容器服务

最佳实践

产品文档



腾讯云

【版权声明】

©2013-2017 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或模式的承诺或保证。

文档目录

文档声明.....	2
最佳实践.....	4
构建简单web应用	4
docker run 参数适配	13
单实例多容器实践.....	18

最佳实践

构建简单web应用

使用腾讯云容器服务构建web应用

本文将介绍在腾讯云容器服务里，如何构建一个最简单的web应用

该web应用分为两部分：

- 1、前端服务，用于处理客户端的查询和写入请求。
- 2、数据存储服务，使用redis，写入的数据存放到redis-master，读取操作访问的是redis-slave，redis-master和redis-slave通过主从复制来保持数据同步。

该应用是kubernetes项目自带的例子，链接地址为

<https://github.com/kubernetes/kubernetes/tree/release-1.6/examples/guestbook>。

一、创建容器集群

- 1、进入创建集群页面，创建[容器集群](#)：
- 2、填写集群名、指定集群的位置(广州、上海、北京等)。
- 3、指定集群的节点网络
，节点网络必须位于某个VPC内，如果您当前没有vpc，请先[创建一个vpc](#)，并在该vpc下创建一个子网。
- 4、指定容器网络。
- 5、为集群节点选择机型(cpu和内存)。
- 6、为集群节点选择磁盘、带宽等配置，并设置密码和安全组。
- 7、选择集群节点的数目。
- 8、稍等几分钟后，集群创建成功。

< 返回 | 创建集群

1 集群信息 > 2 选择机型 > 3 云主机配置

集群名称: 集群的名称

计费模式: 按量计费 包年包月 [详细对比](#)

所在地域: 广州 上海 北京 新加坡
处在不同地域的云产品内网不通，购买后不能更换。建议选择靠近您客户的地域，以降低访问延时，提高下载速度。

可用区: ① 上海一区

节点网络: ① 共253个子网IP，剩252个可用
CIDR: 172.16.0.0/16 vpc和子网，集群内的cvm节点的ip将从该内网内分配

容器网络: ① / [使用指引](#) 容器网络，集群内容器的ip将从该网络内分配

集群描述:

< 返回 | 创建集群

1 集群信息 > 2 选择机型 > 3 云主机配置

主机计费模式: 按量计费

所在地域: 华东地区 (上海)

可用区: 上海一区

所属网络: 0805、sub1

系列: ① 系列1 [详细对比](#)

为集群内的节点选择合适的配置

机型: 标准型S1 内存型M1

机型	CPU	内存	配置费用
<input checked="" type="radio"/> 标准型S1	1核	1GB	¥ 0.31 元/小时 起
<input type="radio"/> 标准型S1	1核	2GB	¥ 0.42 元/小时 起
<input type="radio"/> 标准型S1	1核	4GB	¥ 0.64 元/小时 起

[< 返回](#) | 创建集群

① 集群信息
② 选择机型
③ 云主机配置

操作系统 ① Ubuntu 16.04 64位

系统盘 云硬盘
送20GB 20GB 50GB - 20 + GB (步长1GB)

数据盘 云硬盘
0GB 4000GB - 0 + GB (步长10GB)

公网宽带 ① 按带宽计费 按使用流量计费 详细对比

带宽 0Mbps 100Mbps - 1 + Mbps
 免费分配公网IP [使用指引](#)

登录方式 设置密码 立即关联密钥 自动生成密码
注：请牢记您所设置的密码，如遗忘可登录CVM控制台重置密码。

用户名 ubuntu

密码
linux机器密码需8到16位，至少包括两项 ([a-zA-Z], [0-9]和[!@#%&*+=[]:;.,?]) 的特殊符号

确认密码

安全组 ① 请选择安全组 使用指引
容器节点安全组需放通节点网络和容器网络，公网访问需放通30000-40000端口
 如您有业务需要放通其他端口，您可以 [新建安全组](#)

云主机数量 - 3 + 集群内的节点数目

为节点选择操作系统、磁盘，带宽等配置，并设置登录密码和安全组

二、创建web应用

1、创建redis-master服务

1. 指定服务名称redis-master。
2. 选择集群为我们刚刚创建的集群my-first-cluster。
3. 设置服务的实例信息(实例可以包含多个容器)：
 - 添加一个容器，命名为master。
 - 为master容器指定镜像为ccr.ccs.tencentyun.com/library/redis，版本为latest。
4. 设置服务运行的实例数，redis-master服务需要运行1个实例，我们选择1。
5. 选择服务的访问方式，因为我们的redis服务是内部服务，只提供给集群内其它服务访问，所以我们选择仅在集群内访问。
6. 最后设置服务的访问端口，我们的服务实例包含1个redis容器，该容器监听了6379端口，所以我们配置端口映射的容器端口为6379，服务端口跟容器端口一样，也设置成6379，这样其它服务可以通过服务名称 redis-master以及端口6379就可以访问到我们的master容器了。

< 返回 | 新建服务

服务名称 服务名称由小写字母、数字和 - 组成，且由小写字母开头，小写字母或数字结尾 服务名称 redis-master

所在地域

运行集群 服务所在的集群

服务描述

数据卷(选填) 添加数据卷
为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷，还需挂载到容器的指定路径中。[使用指引](#)

运行容器

名称 服务实例包含一个master容器

镜像 [选择镜像](#) master容器的镜像url

版本 (Tag) 镜像的版本

[显示高级设置](#)

注意：服务创建完成后，容器的配置信息可以通过更新服务的方式进行修改

[添加容器](#)

实例数量 服务运行一个服务实例，该实例包含一个容器

服务访问方式 提供公网访问 仅在集群内访问 VPC内网访问 不启用 redis-master服务只提供给集群内其它服务访问，我们选择仅在集群内访问方式
服务仅提供集群内访问。您还可以配置7层LB (HTTP/HTTPS) 转发到服务，[请参见容器服务7层LB使用说明](#)

端口映射

协议	服务端口	容器端口
TCP	6379	6379

redis-master服务提供6379(redis的监听端口)端口给其它服务调用

[添加端口映射](#)

2、创建redis-slave服务

1. 指定服务名称redis-slave。
2. 选择集群为我们刚刚创建的集群my-first-cluster。
3. 设置服务的实例信息：
 - 添加一个容器，命名为slave。
 - 为slave容器指定镜像为ccr.ccs.tencentyun.com/library/gb-redisslave，版本为latest。
 - 为容器指定cpu，内存资源的上限(可选)，上面的master容器同样可以指定cpu、内存限制。
 - 运行命令和启动参数我们使用镜像里面默认的命令和参数即可，可以不填。
 - 添加一个环境变量 环境变量名称为：GET_HOSTS_FROM，值为:dns（由于gb-redisslave镜像里的程序需要，这里必填）。
4. 设置服务运行的实例数，redis-slave服务需要运行1个实例，我们选择1。
5. 选择服务的访问方式，因为我们的redis-slave服务是内部服务，只提供给集群内其它服务访问，所以我们选择仅在集群内访问。

6. 最后设置服务的访问端口，我们的服务实例包含1个redis slave容器，该容器监听了6379端口，所以我们配置端口映射的容器端口为6379，服务端口跟容器端口一样，也设置成6379，这样其它服务可以通过服务名称 redis-slave以及端口6379就可以访问到我们的slave容器了。



服务名称: redis-slave 服务名称
服务名称由小写字母、数字和-组成，且由小写字母开头，小写字母或数字结尾

所在地域: 广州 上海 北京 新加坡

运行集群: mv-first-cluster 选择服务运行的集群

服务描述: redis slave服务

数据卷(选填) ① [添加数据卷](#)
为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷，还需挂载到容器的指定路径中。[使用指引](#)

运行容器

名称: slave 服务实例包含一个容器，起名为slave

镜像: ccr.ccs.tencentyun.com/library/gb-red 选择镜像 容器的镜像url

版本 (Tag): latest 镜像的版本

CPU限制 ①: request 0.2 核 限制容器的cpu上限
新增limit限制
 request 用于集群分配资源，当集群中的节点没有request所要求的资源数量时，容器会创建失败。
 limit 用于设置容器使用资源的最大上限，避免异常情况下节点资源消耗过多。

内存限制 ①: 128 MiB 限制容器的内存上限
内存限制默认request = limit，更多资源限制见详情。

运行命令 ①:

运行参数 ①:

环境变量 ①: GET_HOSTS_FROM = dns 设置容器运行的环境变量

3、创建frontend服务

1. 指定服务名称frontend。
2. 选择集群为我们刚刚创建的集群my-first-cluster。
3. 设置服务的实例信息：
 - 添加一个容器，命名为frontend。
 - 为slave容器指定镜像为ccr.ccs.tencentyun.com/library/gb-frontent，版本为latest。
 - 添加一个环境变量 环境变量名称为：GET_HOSTS_FROM，值为:dns（由于gb-frontent镜像里的程序需要，这里必填）。
4. 设置服务运行的实例数，frontend服务需要运行1个实例，我们选择1。
5. 选择服务的访问方式，因为我们的frontend需要提供外网浏览器访问，我们选择公网负载均衡访问方式。

6. 最后设置服务的访问端口，我们的服务实例包含1个frontend容器，该容器监听了80端口，所以我们配置端口映射的容器端口为80，服务端口跟容器端口一样，也设置成80，这样，用户通过浏览器访问我们的负载均衡ip就可以访问到我们的frontend容器了。

服务名称 服务名称

所在地域 广州 上海 北京 新加坡

运行集群 服务所在集群

服务描述

数据卷(选填) ① [添加数据卷](#)
 为容器提供存储，目前支持临时路径、主机路径、云硬盘数据卷，还需挂载到容器的指定路径中。[使用指引](#)

运行容器

名称 容器名称

镜像 [选择镜像](#) 容器镜像url

版本 (Tag) 镜像版本

CPU限制 ① 核 ×
新增limit限制
 request只用于资源分配调度，不限制实际使用
 limit与资源调度无关，表示资源的最大使用上限

内存限制 ① MiB
内存限制默认request = limit，更多资源限制见详情。

运行命令 ①

运行参数 ①

环境变量 ① × 容器运行时的环境变量
新增变量
 变量名只能包含大小写字母、数字及下划线，并且不能以数字开头

4、查看服务

点击左侧栏的服务，即可看到我们刚刚创建的三个服务，其中frontend服务可以公网访问，因为我们指定了公网负载均衡访问方式，而redismaster和redisslave服务只能够在集群内被其它服务访问，因为我们设置了访问方式为集群内访问。

服务 广州 上海 北京 新加坡 所属集群 my-first-cluster (cls-4a6tenv...)

+ 新建 请输入服务名

名称	监控	日志	状态	运行/预期数量	IP地址	负载均衡	创建时间	操作
redis-slave	山	国	运行中	1/1个	10.20.255.98	未启用	2017-05-12 12:42:12	更新实例数量 更新服务 删除
redis-master	山	国	运行中	1/1个	10.20.255.40	未启用	2017-05-12 11:15:22	更新实例数量 更新服务 删除
frontend	山	国	运行中	1/1个	211.159.213.194 10.20.255.125	lb-69fcb2z	2017-05-11 19:07:37	更新实例数量 更新服务 删除

我们注意到，frontend服务的属性里面的ip地址有两个：一个外网ip 211.159.213.194和一个内网ip 10.20.255.125，而redis-slave和redis-master服务分别只有一个内网ip，那是因为frontend服务的访问方式为公网负载均衡方式访问，所以我们为该服务分配了一个公网负载均衡，该外网ip就是公网负载均衡的ip，由于frontend服务的访问端口为80，所以我们可以直接在浏览器直接输入该外网ip 211.159.213.194，可以看到：

Guestbook

Messages

Submit

说明我们可以正常访问frontend服务了，现在就试下在输入框中输入任意的字符串吧，输入后可以看到，我们输入的记录被保存起来，并且展示在页面下方，我们可以开启另外一个浏览页，重新打开这个负载均衡的ip地址，看到之前输入的数据都在，说明我们输入的字符串确实已经保存到了redis。

三、开发实践

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);
require 'Predis/Autoloader.php';
Predis\Autoloader::register();
if (isset($_GET['cmd']) === true) {
    $host = 'redis-master';
    if (getenv('GET_HOSTS_FROM') === 'env') {
        $host = getenv('REDIS_MASTER_SERVICE_HOST');
    }
    header('Content-Type: application/json');
    if ($_GET['cmd'] === 'set') {
        $client = new Predis\Client([
            'scheme' => 'tcp',
```

```
'host' => $host,
'port' => 6379,
]);
$client->set($_GET['key'], $_GET['value']);
print("{\"message\": \"Updated\"}");
} else {
    $host = 'redis-slave';
    if (getenv('GET_HOSTS_FROM') == 'env') {
        $host = getenv('REDIS_SLAVE_SERVICE_HOST');
    }
    $client = new Predis\Client([
        'scheme' => 'tcp',
        'host' => $host,
        'port' => 6379,
    ]);
    $value = $client->get($_GET['key']);
    print("{\"data\": \"' . $value . '\"}");
}
} else {
    phpinfo();
} ?>
```

这是guestbook app的frontend服务的完整代码，很简短。frontend服务收到一个http请求后，判断是否是set命令，如果是set命令，就取出参数中的key，value，并连接到redis-master服务，将key, value设置到redis master中。如果不是set命令，就连接到redis-slave服务，从redis slave中获取参数key对应的value，并吐给客户端来展示。

通过示例的web app，有两点需要注意：

1、frontend访问redis-master和redis-slave服务时，连接的是服务名和端口

，我们的集群自带dns服务，会把服务名解析成对应的服务ip，并根据服务ip来做负载均衡，比如redis-slave服务有三个实例，那么我们访问redis-slave服务时，直接连接redis-slave和6379，dns会自动把redis-slave解析成redis-slave的服务ip(实际上是一个浮动ip，类似于负载均衡的ip)，并根据redis-slave的服务ip，会自动做负载均衡，把请求发往某个redis-slave服务的实例中。

2、我们可以为容器设置环境变量。在本例中，frontend容器运行时，会读取GET_HOSTS_FROM环境变量，如果环境变量的值为dns，那么就直接通过服务名来连接(推荐做法)，否则再通过另一个环境变量来获取redis-master或者redis-slave的域名。

docker run 参数适配

docker run 参数适配

本文将介绍如何把在本地的docker中已经调试完毕的容器，在迁移到腾讯云容器服务平台中运行时，对于docker run中的参数如何跟腾讯云容器控制台的参数进行对应。这里我们以创建一个简单的gitlab服务为例。

一、gitlab容器的参数示例

我们使用以下的docker run命令可以创建出一个简单的gitlab容器：

```
docker run \  
-d \  
-p 20180:80 \  
-p 20122:22 \  
--restart always \  
-v /data/gitlab/config:/etc/gitlab \  
-v /data/var/log/gitlab:/var/log/gitlab \  
-v /data/gitlab/data:/var/opt/gitlab \  
--name gitlab \  
gitlab/gitlab-ce:8.16.7-ce.0
```

-d

:容器在后台运行。容器平台都是以后台的形式来运行容器，所以本参数不需要在容器控制台指定。

-p

:指定端口映射。我们这里映射了两个端口，容器端口分别是80和22，对外暴露的端口分别是20180和20122，对应到控制台，我们添加两条端口映射规则，并填写对应的容器端口和服务端口。由于我们的gitlab需要提供外网访问，我们选择了提供公网访问访问方式。

服务访问方式 ^①

提供公网访问
 仅在集群内访问
 VPC内网访问
 不启用

服务可以通过公网访问，将自动新建公网4层LB（1元/天）。您还可以配置7层LB（HTTP/HTTPS）转发到服务，详情见[容器服务7层LB使用说明](#)

端口映射

协议 ^①	容器端口	服务端口 ^①	
TCP	22	20122	×
TCP	80	20180	×

[添加端口映射](#)

您的服务如果是应用型负载均衡的后端服务，不建议修改已在转发规则中的服务的端口映射，若业务端口变更，建议您先新增转发规则再更新服务端口

--restart

:本参数用于指定在容器退出时，是否重启容器。容器平台创建的所有容器退出时，都会重启容器，所以本参数不需要在容器控制台指定。

-v

:本参数用于指定

容器卷。上面的命令指定了三个

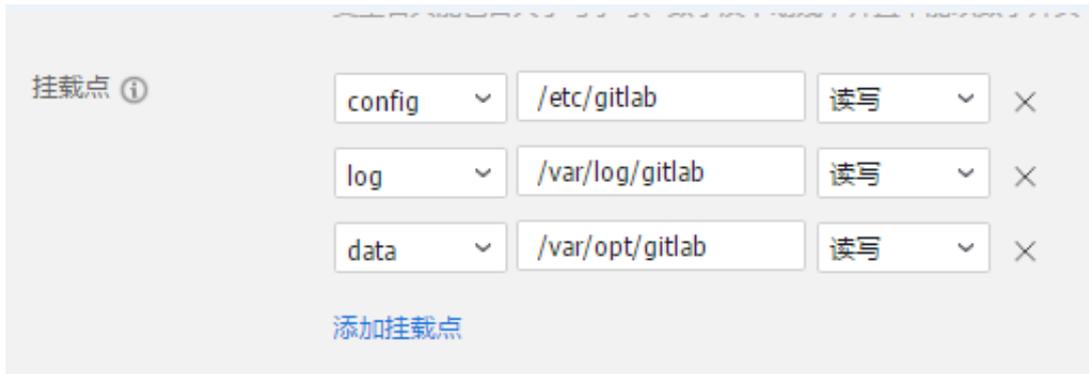
卷，对应到容器控制台，我们也需要添加三个数据卷，并在容器的高级设置里将这三个卷挂载到容器里。

首先我们创建三个卷：

数据卷(选填) ^①

使用本地硬盘	config	/data/gitlab/config	×
使用本地硬盘	log	/data/var/log/gitlab	×
使用本地硬盘	data	/data/gitlab/data	×

接着我们在容器的高级设置里面，将三个卷分别挂载到容器里：



这里要注意的是，我们的数据卷的类型选择的是

使用本地硬盘

，所以容器运行过程中，在容器中生产的数据会被保存到容器所在的节点上，如果容器被调度到其他的节点上，那么数据就丢失了。我们可以使用

云硬盘

类型数据卷，容器的数据会保存到云硬盘中，即使容器被调度到其他的节点，容器卷的数据也不会丢。

`--name`

:容器运行的名字。这个参数，对应到容器控制台就是服务名，当然容器名也可以跟服务名用相同的名字。

二、其它参数

这里介绍我们执行docker run时，其它常见的参数：

`-i`

:交互式执行容器。我们容器控制台只支持后台运行容器，所以本参数不支持。

`-t`

:跟-i参数一样，本参数也不支持。

-e

:容器运行的环境变量。比如用户执行以下的docker run命令：

```
docker run -e FOO='foo' -e BAR='bar' --name=container_name container_image
```

这里用户希望为容器添加两个环境变量，在我们的容器控制台创建服务时，在容器的高级设置里，可以添加容器的环境变量，在这里，变量名和变量值分别为 变量名:Foo,变量值:foo，以及变量名:BAR,变量值:bar。

三、Command和Args

有时候我们希望在docker run的时候，指定进程的命令和参数，比如：

```
docker run --name=kubedns gcr.io/google_containers/kubedns-amd64:1.7 /kube-dns  
--domain=cluster.local. --dns-port=10053 -v 2
```

这里我们指定了容器进程的命令为：/kube-dns，并指定了三个参数：

-domain=cluster.local.

--dns-port=10053

和-

v 2

。

在我们的控制台，我们可以这样指定：

运行命令 ⓘ

`/kube-dns`

运行参数 ⓘ

```
-domain=cluster.local.  
--dns-port=10053  
-v 2
```

单实例多容器实践

单实例多容器实践

单实例多容器优势

资源共享和通信

：实例的存在使同个实例下的容器之间能更方便的共享数据和通信。同个实例下的容器使用相同的网络命名空间、IP地址和端口区间，相互之间能通过localhost来发现和通信。在一个无层次的共享网络中，每个实例都有一个IP地址用于跟别的物理主机和容器通信，实例的名字就用作容器通信时的主机名。在同个实例内运行的容器还共享一块存储卷空间，存储卷内的数据不会在容器重启后丢失，同时能被同实例下别的容器读取。

管理

：相比原生的容器接口，实例通过提供更高层次的抽象，简化了应用的部署和管理。实例就像一个管理和横向部署和管理的单元，主机托管、资源共享、协调复制和依赖管理都可以自动处理。

常用单实例多容器应用场景

实例能应用于构建垂直集成应用栈，但它的主要为了集中管理一些辅助程序，如：

- 内容管理，文件和数据加载进程，本地cache管理进程等
- 日志压缩、rotation、备份、快照等
- 数据变化监听、日志和监控适配器，事件分发等
- 代理，网桥和适配器等
- 控制、管理、配置、升级程序等

更多可查看[实例应用场景](#)