

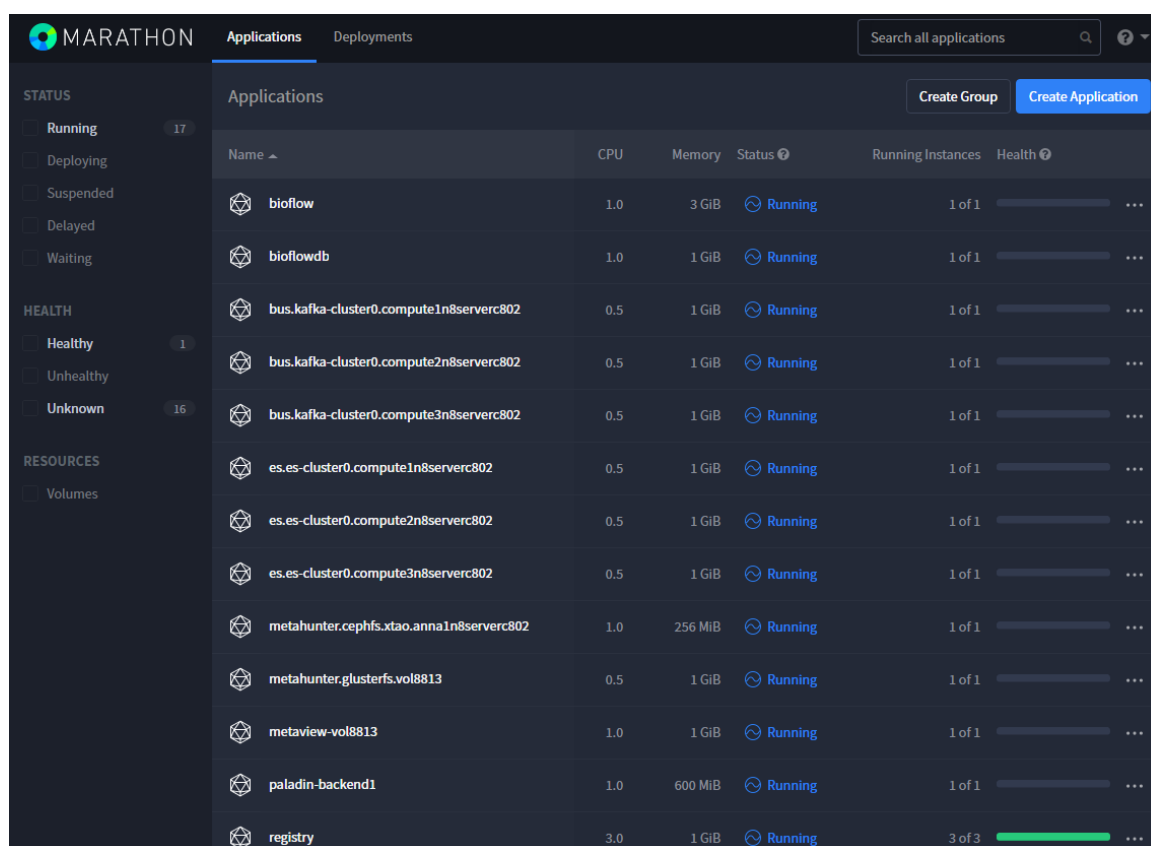
XTAO 容器服务平台说明书



WWW.XTAOTECH.COM

概述

XTAO 容器服务平台，是一个容器服务的管理和发布平台。用户可以按照需求上传定制的容器服务镜像，发布有状态或无状态的微服务，为容器化的应用提供高效部署、资源调度、服务发现和动态伸缩等一系列完整功能。解决用户开发、测试及运维过程的环境一致性问题，提高了大规模容器集群管理的便捷性，帮助用户降低成本，提高效率。



MARATHON		Applications	Deployments	Search all applications	
STATUS Running 17 Deploying Suspended Delayed Waiting		Applications Create Group Create Application			
Name	CPU	Memory	Status	Running Instances	Health
bioflow	1.0	3 GiB	Running	1 of 1	...
bioflowdb	1.0	1 GiB	Running	1 of 1	...
bus.kafka-cluster0.compute1n8serverc802	0.5	1 GiB	Running	1 of 1	...
bus.kafka-cluster0.compute2n8serverc802	0.5	1 GiB	Running	1 of 1	...
bus.kafka-cluster0.compute3n8serverc802	0.5	1 GiB	Running	1 of 1	...
es.es-cluster0.compute1n8serverc802	0.5	1 GiB	Running	1 of 1	...
es.es-cluster0.compute2n8serverc802	0.5	1 GiB	Running	1 of 1	...
es.es-cluster0.compute3n8serverc802	0.5	1 GiB	Running	1 of 1	...
metahunter.cephfs.xtao.anna1n8serverc802	1.0	256 MiB	Running	1 of 1	...
metahunter.glusterfs.vol8813	0.5	1 GiB	Running	1 of 1	...
metaview-vol8813	1.0	1 GiB	Running	1 of 1	...
paladin-backend1	1.0	600 MiB	Running	1 of 1	...
registry	3.0	1 GiB	Running	3 of 3	...

XTAO 容器服务平台具有以下优势：

简单易用，访问容器服务的域名和端口，由平台自动生成和管理。用户无需考虑容器服务所需后端存储，只需给出大小及容器内映射目录，由平台帮你完成空间分配、本地文件系统格式化、挂载映射等工作。

灵活扩展，支持容器服务实例动态扩展，以解决高负载情况下带来的性能瓶颈问题，自动进行负载均衡，提高吞吐量。

高可用，用户使用规范化域名来访问容器服务，无需关心这些容器运行在哪些计算节点。当有服务器故障宕机后，容器服务会自动在其他节点重新启动，不影响对应用的访问。

安全可靠，用户程序以容器形式运行，独享计算资源。服务启动时可以指定 CPU，内存等参数来限制服务占用过多资源。

制作 Docker 镜像

这一章将以 postgresql 数据库为例，介绍如何在开发环境制作容器服务的 docker 镜像。

安装 Docker 服务

```
# yum -y install docker
```

搭建 yum 源

```
# yum -y install httpd
```

```
# mkdir -p /var/www/html/packages
```

```
# systemctl enable httpd
```

```
# systemctl start httpd
```

搭建 yum 源的目的，主要是后面步骤中 build docker 容器镜像时，需要将用户的应用服务以 rpm 包形式安装到编译环境。使用 yum install 命令可以自动安装所有的依赖包，比较简单。如果依赖包不是很多，用户也可以直接将 rpm 包添加到容器中并使用 rpm --ivh *.rpm 命令来安装。

将所有 rpm 包放置到/var/www/html/packages/目录下，必须更新 yum 源才能生效：

```
# createrepo /var/www/html/packages/ --update
```

编写 Dockerfile 文件

```
[root@zmfsea postgresql]# cat Dockerfile
FROM anna:latest
MAINTAINER dev <dev@xtaotech.com>

# ADD xtao.repo /etc/yum.repos.d/xtao.repo

RUN yum clean all && yum makecache && yum install -y sudo
RUN yum install pwgen postgresql96-server postgresql96 postgresql96-contrib -y

RUN sed -i 's/.*requiretty$/#Defaults requiretty/' /etc/sudoers

# Modified setup script to bypass systemctl variable read stuff
ADD ./postgresql96-setup /usr/pgsql-9.6/bin/postgresql96-setup

# Update data folder perms
RUN chown -R postgres:postgres /var/lib/pgsql/9.6/data

#Modify perms on setup script
RUN chmod +x /usr/pgsql-9.6/bin/postgresql96-setup
RUN chown -R postgres:postgres /var/lib/pgsql/9.6/data
RUN chmod 0700 -R /var/lib/pgsql/9.6/data

#Add start script for postgres
ADD ./entrypoint.sh /entrypoint.sh

RUN chmod +x /entrypoint.sh

WORKDIR /
ENTRYPOINT ["/entrypoint.sh"]
```

上图为 postgresql 数据库 docker 容器镜像的 Dockerfile:

FROM anna:latest

表示此 docker 镜像是基于 anna 这个 base 镜像做的, 用户可以选择使用自己的 base 镜像制作容器, 也可以使用 XTAO 提供的 anna 镜像, 获取方法需以管理员身份, 在任意 XTAO 计算节点执行:

docker pull anna

docker save registry.marathon.mesos:5000/anna > anna.tar

将 anna.tar 文件拷贝到开发环境的任意目录下, 然后执行

docker load -i anna.tar

MAINTAINER dev <dev@xtaotech.com>

这一行记录了维护者的信息，name & email

```
# ADD xtao.repo /etc/yum.repos.d/xtao.repo
```

这一行默认是注掉的，表示使用 base 镜像里设置的 yum 源。如果想用自己的，可以去掉#，使能该行。xtao.repo 是 yum 源文件，指定了 yum 源的地址：

```
[root@zmfsea postgresql]# cat xtao.repo
[xtao]
name=xtao repo
baseurl=http://127.0.0.1/packages
enabled=1
gpgcheck=0
priority=1
```

其中，baseurl=http://127.0.0.1/packages 这一行表示 yum 源在本地服务器，具体路径为/var/www/html/packages/。

RUN 关键字表示在编译 docker 镜像的环境中，执行指定的 shell 命令，比如 yum 安装 rpm 包，chmod 修改脚本执行权限等。

ADD 关键字表示将宿主机上的某个文件添加到 docker 镜像编译环境中。如果这个文件是.tar.gz 或.tgz 等格式的压缩文件，docker 会自动对其解压缩。这也是一种安装应用服务的方法。例如：

```
ADD elasticsearch-*.tar.gz /
```

```
RUN mv /elasticsearch-* /elasticsearch
```

WORKDIR /

表示创建并启动容器后的工作目录是根目录。

ENTRYPOINT ["/entrypoint.sh"]

表示创建并启动容器后的入口程序是执行/entrypoint.sh 这个 shell 脚本。

编写 **entrypoint** 文件

上面已经提到过了 **entrypoint.sh** 是容器启动后的入口程序，用户可以在这个脚本文件中进行一些必要的初始化工作，然后以前台模式启动应用服务，这样，只要应用服务不退出，容器就一直处于运行状态。

还以 **postgresql** 数据库为例加以说明：

```
# cat entrypoint.sh
```

```
#!/bin/bash
```

```
RUNFLG="/var/lib/pgsql/9.6/data/runflg"
```

```
USER=${DBUSER}
```

```
PASS=${DBPASSWD}
```

```
DB=${DBNAME}
```

```
PORT=${DBPORT:-5432}
```

```
if [ ! -f $RUNFLG ]
```

```
then
```

```
    echo "initialize postgresql start-up environment..."
```

```
    chmod +x /usr/pgsql-9.6/bin/postgresql96-setup
```

```
    /usr/pgsql-9.6/bin/postgresql96-setup initdb
```

```

chown -R postgres:postgres /var/lib/pgsql/9.6/data
chmod 0700 -R /var/lib/pgsql/9.6/data

POSTCONFHBA="/var/lib/pgsql/9.6/data/pg_hba.conf"
touch $POSTCONFHBA
chown postgres:postgres $POSTCONFHBA
echo "local    all        all                                peer">$POSTCONFHBA
echo "host     all        all  127.0.0.1/32                trust">>$POSTCONFHBA
echo "host     all        all  ::1/128                    ident">>$POSTCONFHBA
echo "host     all        all  0.0.0.0/0                  md5">>$POSTCONFHBA
POSTCONF="/var/lib/pgsql/9.6/data/postgresql.conf"
sed -i -e"s/^#listen_addresses =.*$/listen_addresses = '*'/" $POSTCONF
if [ ${PORT} ]
then
    sed -i -e"s/^#port =.*$/port = ${PORT}/" $POSTCONF
fi
touch $RUNFLG
echo "initialize postgresql start-up environment complete"
if [ ${USER} ]
then
    if [ ! ${PASS} ]
    then
        echo "use user name as password"
        PASS=${USER}
    fi
    echo "start postgres daemon temporarily..."
    sudo -u postgres /usr/pgsql-9.6/bin/postgres -D /var/lib/pgsql/9.6/data &
    while ! /bin/psql -U postgres -h 127.0.0.1 -p ${PORT} -q -c "select true;"; do sleep
1; done
    echo "create user ${USER}"
    /bin/psql -U postgres -h 127.0.0.1 -p ${PORT} -c "CREATE USER ${USER} WITH
PASSWORD '${PASS}';"
    if [ ! ${DB} ]
    then
        echo "use user name as dbname"
        DB=${USER}
    fi
    echo "create database ${DB}"
    /bin/psql -U postgres -h 127.0.0.1 -p ${PORT} -c "CREATE DATABASE \"${DB}\"
OWNER ${USER};"
    echo "stop postgres daemon temporarily..."

```



```

        sudo -u postgres /usr/pgsql-9.6/bin/pg_ctl stop -m fast -D
/var/lib/pgsql/9.6/data
    fi
fi

if [ ${PORT} ]
then
    echo "use port ${PORT}"
    sed -i -e "s/^port = .*$/port = ${PORT}/" $POSTCONF
fi
echo "start postgres daemon..."
sudo -u postgres /usr/pgsql-9.6/bin/postgres -D /var/lib/pgsql/9.6/data
echo "PostgreSQL daemon failed...."

```

数据库是一种有状态的服务，所以，第一次启动时，需要额外执行一些初始化工作，包括：

- 1、为 postgresql96-setup 脚本文件增加执行权限，并执行 initdb 操作。
 - 2、将 /var/lib/pgsql/9.6/data 后端数据目录修改为 postgres 用户和组，并修改访问权限。
 - 3、修改 pg_hba.conf 配置文件，增加访问控制权限。
 - 4、修改 postgresql.conf 配置文件，使 listen_addresses 监听到所有本地 ip 上，并修改 port 为环境变量 \${PORT} 指定的端口。
 - 5、如果环境变量中给定了用户，则还需要临时启动数据库服务（后台方式），然后执行 "CREATE USER ..."，"CREATE DATABASE ..." 等 sql 语句，完成后 stop 掉数据库服务。
 - 6、创建 runflg 文件，表明数据库第一次启动时，初始化操作已经完成。
- 每次重新启动容器时，都要修改配置文件中的 port 到环境变量 \${PORT} 指定的值，然后以 postgres 用户身份在前台模式下启动数据库主服务

编译 Docker 镜像并保存到 tar 文件

```
[root@zmfsea postgresql]# ls -l
total 24
-rw-r--r-- 1 root root 828 Jul 24 17:30 Dockerfile
-rw-r--r-- 1 root root 2194 Sep 6 15:48 entrypoint.sh
-rw-r--r-- 1 root root 8852 Jul 23 15:41 postgresql96-setup
-rw-r--r-- 1 root root 93 Sep 6 11:51 xtao.repo
```

编译容器镜像前，确认当前目录下有上述四个文件。执行下面命令开始编译容器镜像：

```
# docker build --rm -t zmfsea/cmp-postgresql .
```

`docker build` 命令就是根据 `Dockerfile`，在一个临时的虚拟环境中 ADD 哪些文件和 RUN 哪些命令，最终将整个虚拟环境打包成一个镜像。镜像名我们命名为 `zmfsea/cmp-postgresql`，`zmfsea` 是用户名，后面在上传镜像时会检查镜像名与实际操作者是否一致。完成后使用下面命令进行查看：

```
# docker images | grep cmp-postgresql
```

```
[root@zmfsea postgresql]# docker images | grep cmp-postgresql
zmfsea/cmp-postgresql latest 18e969339b59 2 hours ago 340.4 MB
```

将制作好的 `docker` 容器镜像保存到 `tar` 文件。

```
# docker save zmfsea/cmp-postgresql > cmp-postgresql.tar
```

上传 Docker 镜像

这一节将介绍如何上传镜像，后面大部分操作基本都是以普通用户身份执行，我们这里假设在计算节点上管理员已经为我们创建了一个用户名为 `zmfsea`

将镜像的 **tar** 文件拷贝到任意登陆节点

```
# scp cmp-postgresql.tar zmfsea@compute1:/home/zmfsea/
```

使用 **imgcli push** 命令将镜像上传至 **Docker** 仓库

以 **zmfsea** 身份登陆到 **compute1**，执行：

```
# imgcli push cmp-postgresql.tar
```

给普通用户授权

由系统管理员使用 **xtao auth password** 命令进行授权

```
# xtao auth password zmfsea p@ssw0rd
```

```
[root@Anna2N8serverc802 ~]# xtao auth password zmfsea p@ssw0rd
Starting new HTTP connection (1): Anna2N8serverc802
{
  "output": "success to set the passwd",
  "error": 0
}
```

普通用户执行 **xtcli-compute -c** 命令测试登陆权限

```
# xtcli-compute -c
```

User:zmfsea

Password: p@ssw0rd

启动 Docker 容器服务

登陆 xtcli-compute 命令行

```
# xtcli-compute -c
```

```
User:zmfsea
```

```
Password: p@ssw0rd
```

执行 mgmt service 命令启动一个容器服务实例

```
(xtcli-compute) mgmt service --id testdb --image zmfsea/cmp-postgresql --cpu 0.5 --mem 512 --env "DBUSER=zmfsea, DBPASSWD=123456, DBNAME=MYDB, DBPORT=@PORT" --volume "/var/crash:/var/crash" --brick "131072:/var/lib/pgsql/9.6/data/"
```

```
(xtcli-compute) mgmt service --id testdb --image zmfsea/cmp-postgresql --cpu 0.5 --mem 512 --env "DBUSER=zmfsea, DBPASSWD=123456, DBNAME=MYDB, DBPORT=@PORT" --volume "/var/crash:/var/crash" --brick "131072:/var/lib/pgsql/9.6/data/"
{
  "host": "testdb.marathon.mesos",
  "port": 1066
}
```

使用桥模式启动另一个容器服务实例：

```
(xtcli-compute) mgmt service --id testdb2 --image zmfsea/cmp-postgresql --cpu 0.5 --mem 512 --env "DBUSER=zmfsea, DBPASSWD=123456, DBNAME=MYDB" --port @PORT:5432 --volume "/var/crash:/var/crash" --brick "131072:/var/lib/pgsql/9.6/data/"
```

```
(xtcli-compute) mgmt service --id testdb2 --image zmfsea/cmp-postgresql --cpu 0.5 --mem 512 --env "DBUSER=zmfsea, DBPASSWD=123456, DBNAME=MYDB" --port @PORT:5432 --volume "/var/crash:/var/crash" --brick "131072:/var/lib/pgsql/9.6/data/"
{
  "host": "testdb2.marathon.slave.mesos",
  "port": 31067
}
```

mgmt service 命令参数详解

```
(xtcli-compute) mgmt service -h
usage: mgmt service [-h] [-d [NODE]] [-j] [-y] [--id ID] [--image IMAGE]
                  [--cpu CPU] [--mem MEM] [--env ENV] [--brick BRICK]
                  [--volume VOLUME] [--port PORT] [--pool POOL]
                  [--host HOST]

Display the information for the specific disk

optional arguments:
  -h, --help            show this help message and exit
  -d [NODE], --node [NODE]
                        The address of the operating destination
  -j, --json            JSON format output
  -y, --yes            non-interactive mode
  --id ID              service id
  --image IMAGE        docker image. e.g. "postgresql:latest"
  --cpu CPU            cpu requirement. e.g. "0.5"
  --mem MEM            memory requirement(MB). e.g. "512"
  --env ENV            environment variables. e.g.
                        "DBUSER=bob,DBPASSWD=p@ssw0rd,DBPORT=@PORT"
  --brick BRICK        brick storage(MB). e.g.
                        "131072:/mnt/brick,262144:/var/lib/data"
  --volume VOLUME      volume mapping. e.g.
                        "/etc/my.conf:/var/lib/my.conf,/var/log:/var/log"
  --port PORT          port mapping(tcp). e.g. "@PORT:8080 or @PORT:9090:udp"
  --pool POOL          internal pool name for db rbd. default:"datapool"
  --host HOST          the service will always running on a fix host if set
(xtcli-compute) █
```

--id 表示容器服务 id，它必须是全局唯一的一个字符串，由'0~9','a~z','-'和'.'这些字符组成。

--image 表示容器服务镜像的名字。

--cpu 表示容器服务启动时，系统给其分配的 CPU 数。

--mem 表示容器服务启动时，系统给其分配的内存数，以 MB 为单位。

--env 表示容器服务启动时，向其传递的环境变量，前面第一节制作 docker 镜像时，写的 entrypoint.sh 脚本中，我们尝试获取 DBUSER，DBPASSWD，DBNAME 和 DBPORT 四个环境变量，就可以通过这个参数以 key=value 方式传入，key 即为环境变量名，value 即为实际要传递的值。多对 key=value 之间使用逗号分隔。@PORT 这是个特殊关键字，表明由系统帮助分配端口并返回此端口。

--brick 表示容器启动时，如果需要分配后端存储，则可以使用这个参数指定要分配存储空间的大小（单位是 MB），以及最终映射到容器里的什么目录。比如 **postgresql** 数据库服务，它需要额外的空间来存储表文件，WAL 日志等信息，我们给它分配 128GB 空间大小，并映射到容器里的 **/var/lib/pgsql/9.6/data/** 目录供数据库服务使用。参数格式为 **size:container_path**，需要分配多块存储空间时，可以使用逗号进行分隔。

--volume 表示容器启动时，将宿主机上目录或文件映射到容器里的目录或文件。比如我们可以把 **/var/crash/**、**/var/log** 等这些目录映射到容器里，方便我们在宿主机上直接查看 **core** 文件和日志文件。我们也可以把 **/etc/xxx.conf** 文件映射到容器里，使服务在启动时使用宿主机上的配置文件来代替默认配置文件。我们甚至可以将 **/home/zmfsea/entrypoint.sh** 映射到容器里的 **/entrypoint.sh** 来替换入口脚本。参数格式为 **host_path:container_path**，需要映射多对时，使用逗号分隔。需要注意的问题是，尽量不要把宿主机上的 **/var**、**/etc** 等这些“大”目录映射到容器里，一是这可能会有安全隐患，你的服务可能会访问甚至修改别的服务的数据和配置，另外也可能涉及到挂载引用计数问题，导致其他服务在切换节点时，无法正常卸载它们的 **brick**。

--port，端口映射。**postgresql** 这个例子中，启动第一个服务 **testdb**，我们是通过环境变量方式将系统分配的空闲端口传递给 **docker** 容器，在 **entrypoint.sh** 入口程序中，我们获取这个端口值，并且修改对应的配置

文件，这样，启动数据库服务时，就会监听在这个端口上。这种情况是 **host** 模式，容器使用与主机相同的网络。很多情况下，我们不想这么麻烦，那就可以使用端口映射机制。比如 **postgresql** 这个例子中启动的第二个服务 **testdb2**，我们就使用了“**--port @PORT:5432**”，其中 **@PORT** 表示由系统分配一个空闲端口，这个端口对应宿主机的上端口，而 **5432** 是 **postgresql** 数据库配置文件中默认的端口号。通过这种方式，我们无需再修改配置文件中的端口号，外部程序访问 **@PORT** 端口时，**docker** 会将这个请求自动转到容器里的 **5432** 这个端口上，这样数据库就能接收到这个请求并进行处理。使用端口映射时，容器与主机之间的网络是 **bridge** 模式。

主机模式访问服务的域名一般格式为：**serviceld.marathon.mesos**，而桥模式下一般为：**serviceld.marathon.slave.mesos**。

--pool 表示在哪个存储池上分配存储空间，指定了 **--brick** 时会使用此参数。一般使用默认的 **datapool** 即可。

--host 表示让容器服务固定在指定的计算节点运行，即便这个节点没有资源或者宕机，也不会其他节点启动服务。此参数主要用来调试容器服务。**host** 应该使用 **compute** 网络的 **hostname**。





检查服务实例是否启动成功

容器服务启动后，用户可以使用命令行会返回 host 和 port 来访问这个数据库服务。可以简单的 ping 域名，能 ping 通表示容器已经启动。

```
-bash-4.2$ ping testdb.marathon.mesos
PING testdb.marathon.mesos (192.168.78.3) 56(84) bytes of data.
64 bytes from Compute3N8serverc802 (192.168.78.3): icmp_seq=1 ttl=64 time=0.068 ms
64 bytes from Compute3N8serverc802 (192.168.78.3): icmp_seq=2 ttl=64 time=0.063 ms
```


```
-bash-4.2$ ping testdb2.marathon.slave.mesos
PING testdb2.marathon.slave.mesos (192.168.78.1) 56(84) bytes of data.
64 bytes from Compute1N8serverc802 (192.168.78.1): icmp_seq=1 ttl=64 time=0.074 ms
64 bytes from Compute1N8serverc802 (192.168.78.1): icmp_seq=2 ttl=64 time=0.081 ms
```

可以在 marathon 的 web 界面上查看容器服务状态。在浏览器输入 compute1:8080 即可看到。


	testdb	0.5	512 MiB	 Running	1 of 1	...
	testdb2	0.5	512 MiB	 Running	1 of 1	...

点击某个容器服务，比如 testdb

testdb




 Running (1 of 1 instances)

0 Healthy 0 Unhealthy 1 Unknown (100%)

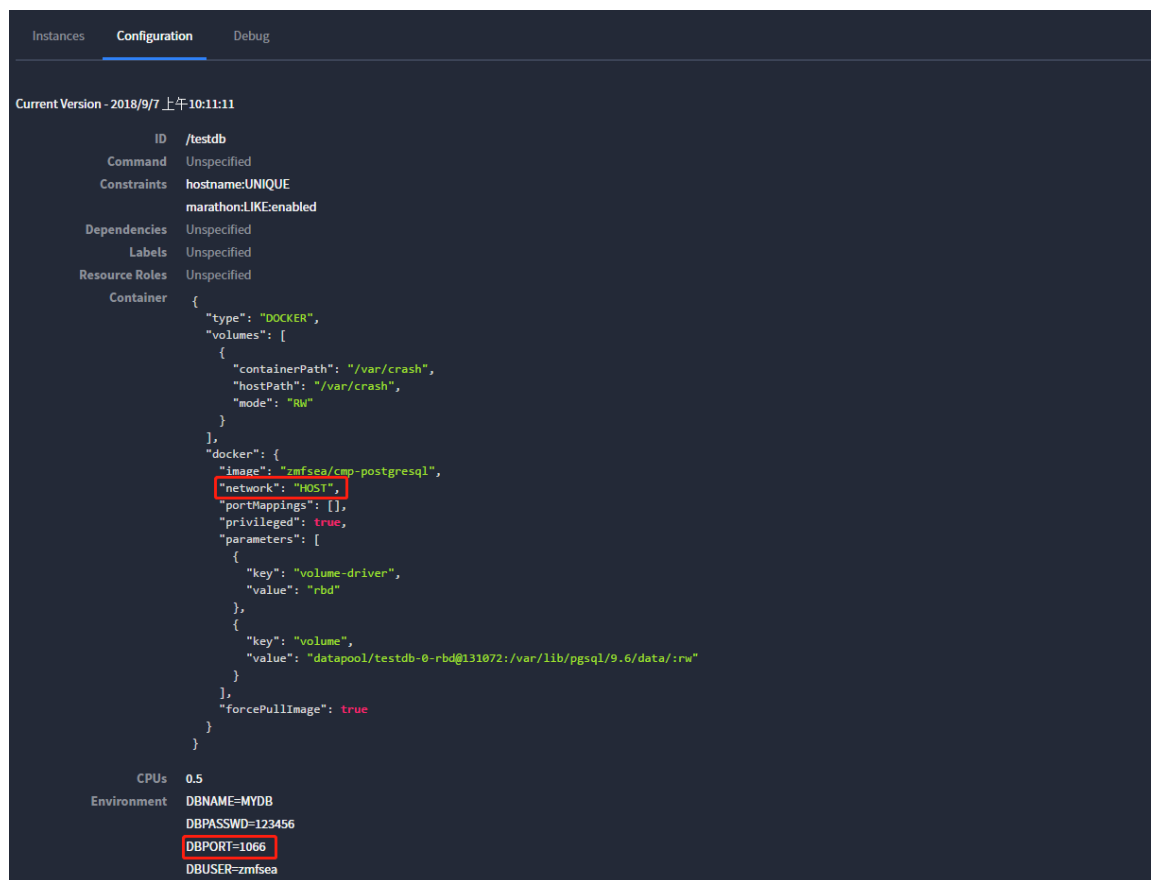
[Scale Application](#) [Restart](#) 

[Instances](#) [Configuration](#) [Debug](#)

[Refresh](#)

ID	Status	Error Log	Output Log
 testdb.4a54e2e3-b243-11e8-8b80-024247c0a575 Compute1N8serverc802:25515	Started	 stderr	 stdout

在 Instances 栏中可以看到该容器服务具体运行在哪个计算节点。



The screenshot displays the 'Configuration' tab of a management interface. At the top, there are three tabs: 'Instances', 'Configuration' (which is selected), and 'Debug'. Below the tabs, the text 'Current Version - 2018/9/7 上午10:11:11' is visible. The main content area shows a list of configuration parameters for a service named '/testdb'. The parameters include 'ID', 'Command', 'Constraints', 'Dependencies', 'Labels', 'Resource Roles', and 'Container'. The 'Container' section is expanded, showing a JSON configuration for a Docker container. The configuration includes 'type', 'volumes', 'docker' (with 'image', 'network', 'portMappings', 'privileged', and 'parameters'), and 'forcePullImage'. The 'network' is set to 'HOST', and the 'portMappings' is an empty array. The 'privileged' flag is set to 'true'. The 'parameters' section contains two entries: 'volume-driver' with value 'rbd' and 'volume' with a complex path. The 'forcePullImage' is set to 'true'. Below the 'Container' section, the 'CPUs' are set to '0.5'. The 'Environment' section lists 'DBNAME=MYDB', 'DBPASSWD=123456', 'DBPORT=1066', and 'DBUSER=zmfsea'. The 'DBPORT=1066' is highlighted with a red box.

```
Current Version - 2018/9/7 上午10:11:11

ID /testdb
Command Unspecified
Constraints hostname:UNIQUE
marathon:LIKE:enabled
Dependencies Unspecified
Labels Unspecified
Resource Roles Unspecified
Container {
  "type": "DOCKER",
  "volumes": [
    {
      "containerPath": "/var/crash",
      "hostPath": "/var/crash",
      "mode": "RW"
    }
  ],
  "docker": {
    "image": "zmfsea/cmp-postgresql",
    "network": "HOST",
    "portMappings": [],
    "privileged": true,
    "parameters": [
      {
        "key": "volume-driver",
        "value": "rbd"
      },
      {
        "key": "volume",
        "value": "datapool/testdb-0-rbd@131072:/var/lib/pgsql/9.6/data/:rw"
      }
    ]
  },
  "forcePullImage": true
}

CPUs 0.5
Environment DBNAME=MYDB
DBPASSWD=123456
DBPORT=1066
DBUSER=zmfsea
```

在 Configuration 栏中可以看到相关配置参数。

```

ID /testdb2
Command Unspecified
Constraints hostname:UNIQUE
marathon:LIKE:enabled
Dependencies Unspecified
Labels Unspecified
Resource Roles Unspecified
Container
{
  "type": "DOCKER",
  "volumes": [
    {
      "containerPath": "/var/crash",
      "hostPath": "/var/crash",
      "mode": "RW"
    }
  ],
  "docker": {
    "image": "zmfsea/cmp-postgresql",
    "network": "BRIDGE",
    "portMappings": [
      {
        "containerPort": 5432,
        "hostPort": 31067,
        "servicePort": 10027,
        "protocol": "tcp",
        "name": "tcp",
        "labels": {}
      }
    ],
    "privileged": true,
    "parameters": [
      {
        "key": "volume-driver",
        "value": "rbd"
      },
      {
        "key": "volume",
        "value": "datapool/testdb2-0-rbd@131072:/var/lib/pgsql/9.6/data/:rw"
      }
    ],
    "forcePullImage": true
  }
}

CPUs 0.5
Environment DBNAME=MYDB
DBPASSWD=123456
DBUSER=zmfsea

```

testdb2 使用了端口映射，网络模式是桥模式。

最后，可以使用数据库客户端命令行来登陆数据库。

```

[root@Anna2N8serverc802 /]# psql -h testdb.marathon.mesos -U zmfsea -d MYDB -p 1066
Password for user zmfsea:
psql (9.6.5)
Type "help" for help.

MYDB=> █

```

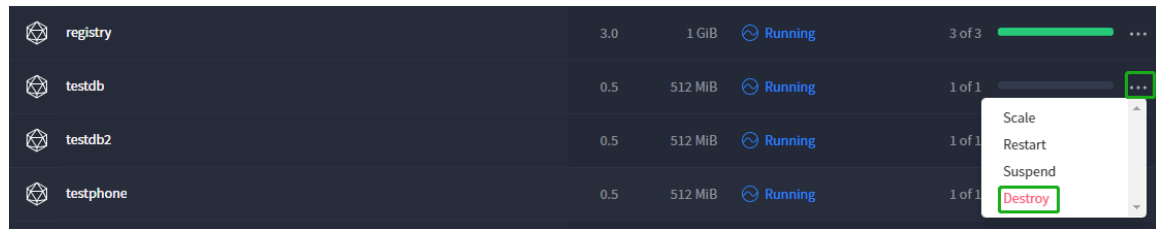
```

[root@Anna2N8serverc802 /]# psql -h testdb2.marathon.slave.mesos -U zmfsea -d MYDB -p 31067
Password for user zmfsea:
psql (9.6.5)
Type "help" for help.

MYDB=> █

```

如果想停掉容器服务，可以在 marathon 界面上点击服务的 Destroy 按钮。



The screenshot shows a container management interface with a table of containers. The 'testdb' container is selected, and a context menu is open, highlighting the 'Destroy' option.

Container Name	Version	Memory	Status	Replicas	Actions
registry	3.0	1 GiB	Running	3 of 3	...
testdb	0.5	512 MiB	Running	1 of 1	Scale, Restart, Suspend, Destroy
testdb2	0.5	512 MiB	Running	1 of 1	...
testphone	0.5	512 MiB	Running	1 of 1	...

当我们再次使用相同命令重新启动容器服务时，域名、端口都不会有任何变化，若使用了 **brick** 后端存储，那么将不会再重新分配新的存储空间，而是使用第一次时已经分配好的，并且上面保存的应用服务数据依然可以继续使用。