



福昕PDF编辑器

• 永久 • 轻巧 • 自由

升级会员

批量购买



永久使用

无限制使用次数



极速轻巧

超低资源占用，告别卡顿慢



自由编辑

享受Word一样的编辑自由



扫一扫，关注公众号

容器云平台使用说明

版本	内容	日期	编写	审核
1.0	初稿	2018/3/8	王武侠	广州技术部

目录

- 一、 平台概述..... 3
 - 平台简介 3
 - 服务内容 3
 - 容器云平台访问地址..... 4
 - gitlab 访问地址 4
- 二、 如何在项目中配置 Dockerfile 5
 - 1. Dockerfile 的书写规则及指令使用方法..... 5
 - 2. （案例）创建 Dockerfile，构建jdk+tomcat 环境..... 11
 - 3. 构建镜像（此步可省略，交给开发集成部分或引入jenkins 等持续集成工具负责构建）. 13
- 三、 开发集成（将项目制作成 docker 镜像） 14
 - 新建项目 14
 - 新建工程 14
 - 镜像管理 26
- 四、 运维管理（启动项目容器） 27
 - 新建服务 27
 - 新建部署 27
 - 部署配置 28
- 五、 参考资料 34

一、平台概述

平台简介

DomeOS 是基于 docker 的企业级私有云一站式运维管理系统。

致力于研究 docker 为代表的容器技术,为企业级用户打造持续交付和自动运维平台,解决用户从代码自动编译打包,到线上运行维护的全套需求。DomeOS 采用私有云模式,实现用户私有集群的容器化管理和资源智能化分配,提供全流程标准化的主机管理、应用持续集成、镜像构建、部署管理、容器运维和多层级监控服务。

服务内容

项目管理

- 代码仓库: 针对企业级用户的需求, DomeOS 目前支持关联私有 Gitlab 代码仓库,可以轻松选择代码项目进行构建。DomeOS 会陆续推出关联其他代码仓库的功能。
- 持续集成: DomeOS 支持根据代码项目的 tag 和 branch 自动构建。代码仓库的 push 操作会自动触发一次构建保证项目镜像和开发进度同步。
- 配置 dockerfile: DomeOS 不要求代码项目必须有 dockerfile, 用户可以在页面上快速配置一个项目的 dockerfile, 并在项目设置里随时修改。
- 构建记录: DomeOS 会记录项目的每一次构建, 可以查看构建的状态、日志、构建者等全方位信息。

业务部署

- 快速灵活: 您可以用项目镜像或第三方镜像进行部署。可以一次部署多个镜像并为每个镜像的容器设定 cpu 和内存占用。从配置部署到启动只需要几分钟。
- 升级回滚: 部署有完整的版本管理, 每次升级会生成一个部署版本, 可以随意选择一个旧版本进行回滚。部署出现异常时可以指定版本恢复。
- 弹性伸缩: 运行中的部署可以随时进行扩容缩容, 增减实例个数, 满足业务要求。
- 健康检查: 可以对部署进行 TCP 或 HTTP 检查, 贯穿部署的整个生命周期, 为业务的稳定健康运行保驾护航。
- 负载均衡: 将流量分摊、引导到服务的每个实例, 提高服务的整体可用性和吞吐量。

集群管理与监控

- 集群设置：每个 DomeOS 集群需要配置一套 kubernetes，您可以在控制台上将集群的各项设置记录下来方便添加主机、日志收集等。此外可以管理集群的 namespace。
- 主机管理：可以查看集群内各个主机的配置、状态；可以随时快速添加主机；可以为主机打标签和划分生产、测试环境。
- 实例管理：可以查看每台主机上的实例列表并查看容器的日志。
- 智能监控：针对物理资源进行主机级别的监控；针对业务进行部署、实例、容器的多层次监控。用户能够从全局和底层单元多角度监控主机和服务的运行状况。

用户管理

- 成员管理：针对组、项目、部署、集群分别设置独立的成员系统，满足不同维度的成员管理需求。可以根据成员的身份和工作界限设定资源内的权限。
- 组管理：可以创建组并将用户添加到组中。组内用户在创建项目、部署、集群时可以选择以组的身份创建。

容器云平台访问地址

开发环境：

<http://172.17.22.26:8088>

账号密码可联系我们开通（试用账号：**test** 密码：**12345678**）

测试环境：

待定

生产环境：

待定

gitlab 访问地址

<http://gitlab.ucsmv.dev/>

新建账号与项目可联系我们开通

二、 如何在项目中配置 Dockerfile

原文地址：http://blog.csdn.net/we_shell/article/details/38445979

Dockerfile 是一种被 **Docker** 程序解释的脚本，Dockerfile 由一条一条的指令组成，每条指令对应 Linux 下面的一条命令。Docker 程序将这些 Dockerfile 指令翻译真正的 Linux 命令。Dockerfile 有自己书写格式和支持的命令，Docker 程序解决这些命令间的依赖关系，类似于 Makefile。Docker 程序将读取 Dockerfile，根据指令生成定制的 image。相比 image 这种黑盒子，Dockerfile 这种显而易见的脚本更容易被使用者接受，它明确的表明 image 是怎么产生的。有了 Dockerfile，当我们需要定制自己额外的需求时，只需在 Dockerfile 上添加或者修改指令，重新生成 image 即可，省去了敲命令的麻烦。

1. Dockerfile 的书写规则及指令使用方法

Dockerfile 的指令是忽略大小写的，建议使用大写，使用 # 作为注释，每一行只支持一条指令，每条指令可以携带多个参数。

Dockerfile 的指令根据作用可以分为两种，构建指令和设置指令。构建指令用于构建 image，其指定的操作不会在运行 image 的容器上执行；设置指令用于设置 image 的属性，其指定的操作将在运行 image 的容器中执行。

(1) FROM (指定基础 image)

构建指令，必须指定且需要在 Dockerfile 其他指令的前面。后续的指令都依赖于该指令指定的 image。FROM 指令指定的基础 image 可以是官方远程仓库中的，也可以位于本地仓库。

该指令有两种格式：

1. FROM <image>

指定基础 image 为该 image 的最后修改的版本。或者：

1. FROM <image>:<tag>

指定基础 image 为该 image 的一个 tag 版本。

(2) MAINTAINER (用来指定镜像创建者信息)

构建指令，用于将 image 的制作者相关的信息写入到 image 中。当我们对该 image 执行 docker inspect 命令时，输出中有相应的字段记录该信息。

格式：

1. MAINTAINER <name>

(3) RUN (安装软件用)

构建指令，RUN 可以运行任何被基础 image 支持的命令。如基础 image 选择了 ubuntu，那么软件管理部分只能使用 ubuntu 的命令。

该指令有两种格式：

1. RUN <command> (the command is run in a shell - `/bin/sh -c`)
2. RUN ["executable", "param1", "param2" ...] (exec form)

(4) CMD (设置 container 启动时执行的操作)

设置指令，用于 container 启动时指定的操作。该操作可以是执行自定义脚本，也可以是执行系统命令。该指令只能在文件中存在一次，如果有多个，则只执行最后一条。

该指令有三种格式：

1. `CMD ["executable","param1","param2"]` (like an exec, this is the preferred form)
2. `CMD command param1 param2 (as a shell)`

当 Dockerfile 指定了 ENTRYPOINT，那么使用下面的格式：

1. `CMD ["param1","param2"]` (as default parameters to ENTRYPOINT)

ENTRYPOINT 指定的是一个可执行的脚本或者程序的路径，该指定的脚本或者程序将会以 param1 和 param2 作为参数执行。所以如果 CMD 指令使用上面的形式，那么 Dockerfile 中必须要有配套的 ENTRYPOINT。

(5) ENTRYPOINT (设置 container 启动时执行的操作)

设置指令，指定容器启动时执行的命令，可以多次设置，但是只有最后一个有效。

两种格式：

1. `ENTRYPOINT ["executable", "param1", "param2"]` (like an exec, the preferred form)
2. `ENTRYPOINT command param1 param2 (as a shell)`

该指令的使用分为两种情况，一种是独自使用，另一种和 CMD 指令配合使用。

当独自使用时，如果你还使用了 CMD 命令且 CMD 是一个完整的可执行的命令，那么 CMD 指令和 ENTRYPOINT 会互相覆盖只有最后一个 CMD 或者 ENTRYPOINT 有效。

1. `# CMD 指令将不会被执行，只有 ENTRYPOINT 指令被执行`
2. `CMD echo "Hello, World!"`
3. `ENTRYPOINT ls -l`

另一种用法和 CMD 指令配合使用来指定 ENTRYPOINT 的默认参数，这时 CMD 指令不是一个完整的可执行命令，仅仅是参数部分；ENTRYPOINT 指令只能使用 JSON 方式指定执行命令，而不能指定参数。

1. `FROM ubuntu`
2. `CMD ["-l"]`
3. `ENTRYPOINT ["/usr/bin/ls"]`

(6) USER (设置 container 容器的用户)

设置指令，设置启动容器的用户，默认是 root 用户。

1. # 指定 memcached 的运行用户
2. ENTRYPOINT ["memcached"]
3. USER daemon
4. 或
5. ENTRYPOINT ["memcached", "-u", "daemon"]

(7) EXPOSE (指定容器需要映射到宿主机器的端口)

设置指令，该指令会将容器中的端口映射成宿主机中的某个端口。当你需要访问容器的时候，可以不是用容器的 IP 地址而是使用宿主机器的 IP 地址和映射后的端口。要完成整个操作需要两个步骤，首先在 Dockerfile 使用 EXPOSE 设置需要映射的容器端口，然后在运行容器的时候指定-p 选项加上 EXPOSE 设置的端口，这样 EXPOSE 设置的端口号会被随机映射成宿主机中的一个端口号。也可以指定需要映射到宿主机的那个端口，这时要确保宿主机上的端口号没有被使用。EXPOSE 指令可以一次设置多个端口号，相应的运行容器的时候，可以配套的多次使用-p 选项。

格式:

1. EXPOSE <port> [<port>...]

1. # 映射一个端口
2. EXPOSE port1
3. # 相应的运行容器使用的命令
4. docker run -p port1 image
- 5.
6. # 映射多个端口
7. EXPOSE port1 port2 port3
8. # 相应的运行容器使用的命令
9. docker run -p port1 -p port2 -p port3 image
10. # 还可以指定需要映射到宿主机上的某个端口号
11. docker run -p host_port1:port1 -p host_port2:port2 -p host_port3:port3 image

端口映射是 docker 比较重要的一个功能，原因在于我们每次运行容器的时候容器的 IP 地址不能指定而是在桥接网卡的地址范围内随机生成的。宿主机器的 IP 地址是固定的，我们可以将容器的端口的映射到宿主机上的一个端口，免去每次访问容器中的某个服务时都要查看容器的 IP 的地址。对于一个运行的容器，可以使用 docker port 加上容器中需要映射

的端口和容器的 ID 来查看该端口号在宿主机上的映射端口。

(8) ENV (用于设置环境变量)

构建指令，在 image 中设置一个环境变量。

格式:

1. ENV <key> <value>

设置了后，后续的 RUN 命令都可以使用，container 启动后，可以通过 docker inspect 查看这个环境变量，也可以通过在 docker run --env key=value 时设置或修改环境变量。

假如你安装了 JAVA 程序，需要设置 JAVA_HOME，那么可以在 Dockerfile 中这样写：

```
ENV JAVA_HOME /path/to/java/direct
```

(9) ADD (从 src 复制文件到 container 的 dest 路径)

构建指令，所有拷贝到 container 中的文件和文件夹权限为 0755，uid 和 gid 为 0；如果是一个目录，那么会将该目录下的所有文件添加到 container 中，不包括目录；如果文件是可识别的压缩格式，则 docker 会帮忙解压缩（注意压缩格式）；如果 <src> 是文件且 <dest> 中不使用斜杠结束，则会将 <dest> 视为文件，<src> 的内容会写入 <dest>；如果 <src> 是文件且 <dest> 中使用斜杠结束，则会 <src> 文件拷贝到 <dest> 目录下。

格式:

1. ADD <src> <dest>

<src> 是相对被构建的源目录的相对路径，可以是文件或目录的路径，也可以是一个远程的文件 url；

<dest> 是 container 中的绝对路径

(10) VOLUME (指定挂载点)

设置指令，使容器中的一个目录具有持久化存储数据的功能，该目录可以被容器本身使用，也可以共享给其他容器使用。我们知道容器使用的是 AUFS，这种文件系统不能持久化数据，当容器关闭后，所有的更改都会丢失。当容器中的应用有持久化数据的需求时可以在 Dockerfile 中使用该指令。

格式:

```
1. VOLUME ["<mountpoint>"]
```

```
1. FROM base
2. VOLUME ["/tmp/data"]
```

运行通过该 Dockerfile 生成 image 的容器，/tmp/data 目录中的数据在容器关闭后，里面的数据还存在。例如另一个容器也有持久化数据的需求，且想使用上面容器共享的/tmp/data 目录，那么可以运行下面的命令启动一个容器：

```
1. docker run -t -i -rm -volumes-from container1 image2 bash
```

container1 为第一个容器的 ID，image2 为第二个容器运行 image 的名字。

(11) WORKDIR (切换目录)

设置指令，可以多次切换(相当于 cd 命令)，对 RUN,CMD,ENTRYPOINT 生效。

格式:

```
1. WORKDIR /path/to/workdir
```

```
1. # 在 /p1/p2 下执行 vim a.txt
2. WORKDIR /p1 WORKDIR p2 RUN vim a.txt
```

(12) ONBUILD (在子镜像中执行)

1. ONBUILD <Dockerfile 关键字>

ONBUILD 指定的命令在构建镜像时并不执行，而是在它的子镜像中执行。

详细资料可参考 <https://www.dockerboard.org/docker-quicktip-3-onbuild>

2. (案例) 创建 Dockerfile , 构建 jdk+tomcat 环境

Dockerfile 文件

```
1. # Pull base image
2. FROM ubuntu:13.10
3.
4. MAINTAINER zing wang "zing.jian.wang@gmail.com"
5.
6. # update source
7. RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe"> /etc/apt/sources.list
8. RUN apt-get update
9.
10. # Install curl
11. RUN apt-get -y install curl
12.
13. # Install JDK 7
14. RUN cd /tmp && curl -L 'http://download.oracle.com/otn-pub/java/jdk/7u65-b17/jdk-7u65-linux-x64.tar.gz' -H 'Cookie: oraclelicense=accept-securebackup-cookie; gpw_e24=Dockerfile' | tar -xz
15. RUN mkdir -p /usr/lib/jvm
16. RUN mv /tmp/jdk1.7.0_65/ /usr/lib/jvm/java-7-oracle/
17.
18. # Set Oracle JDK 7 as default Java
19. RUN update-alternatives --install /usr/bin/java java /usr/lib/jvm/java-7-oracle/bin/java 300
20. RUN update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/java-7-oracle/bin/javac 300
21.
22. ENV JAVA_HOME /usr/lib/jvm/java-7-oracle/
```

```

23.
24. # Install tomcat7
25. RUN cd /tmp && curl -L 'http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.8/bin/a
    apache-tomcat-7.0.8.tar.gz' | tar -xz
26. RUN mv /tmp/apache-tomcat-7.0.8/ /opt/tomcat7/
27.
28. ENV CATALINA_HOME /opt/tomcat7
29. ENV PATH $PATH:$CATALINA_HOME/bin
30.
31. ADD tomcat7.sh /etc/init.d/tomcat7
32. RUN chmod 755 /etc/init.d/tomcat7
33.
34. # Expose ports.
35. EXPOSE 8080
36.
37. # Define default command.
38. ENTRYPOINT service tomcat7 start && tail -f /opt/tomcat7/logs/catalina.out

```

tomcat7.sh

```

1. export JAVA_HOME=/usr/lib/jvm/java-7-oracle/
2. export TOMCAT_HOME=/opt/tomcat7
3.
4. case $1 in
5.   start)
6.     sh $TOMCAT_HOME/bin/startup.sh
7.   ;;
8.   stop)
9.     sh $TOMCAT_HOME/bin/shutdown.sh
10.  ;;
11. restart)
12.   sh $TOMCAT_HOME/bin/shutdown.sh
13.   sh $TOMCAT_HOME/bin/startup.sh
14. ;;
15. esac
16. exit 0

```

参考 Github <https://github.com/agileshell/dockerfile-jdk-tomcat.git>

3. 构建镜像（此步可省略，交给开发集成部分或引入 jenkins 等持续集成工具负责构建）

脚本写好了，需要转换成镜像：

1. `docker build -t zingdocker/jdk-tomcat .`
2. `docker run -d -p 8090:8080 zingdocker/jdk-tomcat`

```
www@ubuntu:~/workspace$ docker build -t zingdocker/jdk-tomcat .
Sending build context to Docker daemon 3.584 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:13.10
----> 195eb90b5349
Step 1 : MAINTAINER zing wang "zing.jian.wang@gmail.com"
----> Using cache
----> a94845e898e7
Step 2 : RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe"> /etc/apt/sources.list
----> Using cache
----> 5ccdbe451a73
Step 3 : RUN apt-get update
----> Using cache
----> e190fc901251
Step 4 : RUN apt-get -y install curl
----> Using cache
----> a9e064d2582a
Step 5 : RUN cd /tmp && curl -L 'http://download.oracle.com/otn-pub/java/jdk/7u65-b17/jdk-7u65-linux-x64.tar.gz' -H 'Co
http://blog.csdn.net/we_shell
----> Using cache
----> 585ded75d349
Step 6 : RUN mkdir -p /usr/lib/jvm
----> Using cache
----> ab92dcb85968
Step 7 : RUN mv /tmp/jdk1.7.0_65/ /usr/lib/jvm/java-7-oracle/
----> Using cache
----> 96bb14167db2
Step 8 : RUN update-alternatives --install /usr/bin/java java /usr/lib/jvm/java-7-oracle/bin/java 300
----> Using cache
----> 3590b1da14e2
Step 9 : RUN update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/java-7-oracle/bin/javac 300
----> Using cache
----> 3339290f470a
Step 10 : ENV JAVA_HOME /usr/lib/jvm/java-7-oracle/
----> Running in e4fff1020a33
----> 294b339b6165
Removing intermediate container e4fff1020a33
Step 11 : RUN cd /tmp && curl -L http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.8/bin/apache-tomcat-7.0.8.tar.gz -o
----> Using cache
----> 3339290f470a
```

默认情况下，tomcat 会占用 8080 端口，刚才在启动 container 的时候，指定了 -p 8090:8080，

映射到宿主机端口就是 8090。

`http://<host>:8090` **host 为主机 IP**

三、 开发集成(将项目制作成 docker 镜像)

开发集成包含了 docker 镜像生成的完整流程。用户通过关联代码仓库或者直接填写 **Dockerfile**，再进行简单的配置来创建工程，之后把工程构建成一个镜像，完成整个开发集成的过程。开发集成中包括项目和镜像两部分。工程和镜像都按照项目组进行划分，同一个项目组中的用户权限相同。

新建项目

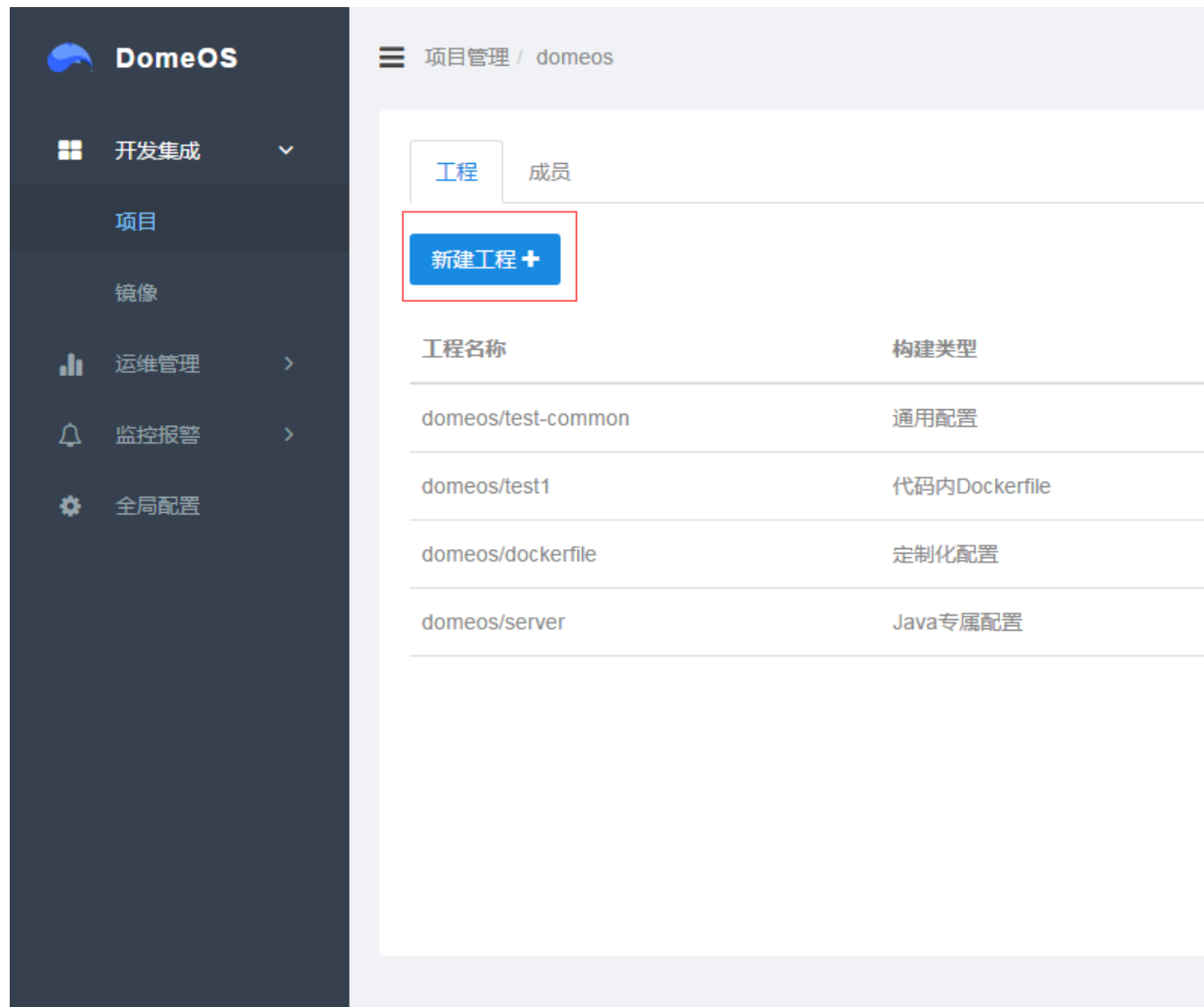
进入 DomeOS 控制台，点击左侧导航栏的“开发集成”下的“项目”进入项目列表页，再点击“新建项目”即可创建新项目。

项目名	项目类型	工程数	成员数	创建时间	创建者
ssssss	公开项目	2	7	2016-10-27	admin
test	私有项目	3	1	2016-10-26	admin
feiliu206363	私有项目	1	1	2016-10-26	feiliu206363@s
yuanfeizhou					

新建工程

您可以在 DomeOS 上关联您的代码源(目前支持 **Gitlab**)，并通过简单的配置创建一个工程，也可以不使用代码仓库中的代码源创建工程。工程将被构建成交付的镜


像文件。 点击项目名称进入工程列表页，可以看到"新建工程"， 点击进入新建工程页面。



关联 Gitlab 代码仓库生成项目

DomeOS 支持关联私有 Gitlab，您可以在新建项目第一步选择 Gitlab 仓库，并点击"关联新账户"。输入 Gitlab 用户名和密码后您会看到自己的 Gitlab 账户已经同步到 DomeOS。您可以选择一个代码项目作为此工程的代码源。

 **DomeOS**

 开发集成

▼

项目

镜像

 运维管理

>

 监控报警

>

 全局配置

项目管理 / domeos / 新建工程

选择代码仓库



GitLab

不使用代码仓库

shengtia 关联新账户 +

宋谋霞 / VRfront	Awesome project
田生(北京研发中心) / scsfrontnew	无描述信息
田生(北京研发中心) / markdown-test-repo	Markdown Test Repo
田生(北京研发中心) / IMG-tester	这个脚本是一个半自动片转换功能的工具。可
田生(北京研发中心) / VRUser	VR Web Front for Use
刘菲(研究院) / DomeOS	无描述信息

runners-token

CI/CD pipeline的runners token，该项配置时采用该token clone

输入token

自动构建

当所选分支更新或出现新tag时，会自动触发构建，生成新的工程

☒ master ☒ tag ☒ 指定其他分支

工程构建类型

工程需要构建成Docker镜像，才能被部署到运行环境中。工程构

通用配置

适合各种编程语言的工程，在页面上配置Dockerfile以及运行过程的相关信息。

定制化配置

直接编写Dockerfile和配置文件。自定义构建流程。

工程名称 *

工程名称即构建后生成的镜像名，创建后不可修改，请慎重选择

domeos / 输入工程名称

此外，还可以对工程进行如下设置：

runners-token: CI/CD pipeline 的 runners token，该项配置时采用该 token clone 代码，否则使用用户 token clone 代码

自动构建: 工程只有被构建成镜像文件才能交付和部署。针对代码项目快速迭代的特点，DomeOS 开发了智能化的持续集成功能。在关联代码仓库后，您可以在"自动构建"处指定代码项目的分支或 tag，当您的代码源发生 push 操作，所选分支更新或出现新 tag 时，会自动触发一次构建，生成新的项目镜像，保证您的项目镜像和代码源同步。

工程构建类型: 工程需要构建成 Docker 镜像，才能被部署到运行环境中。工程构建过程包含了代码编译、docker build 等操作。通用配置是 DomeOS 为 docker 初学者准备的简易 Dockerfile 生成功能，配置关键信息后，后台自动生成对应的 Dockerfile。定制化配置是指用户通过镜像定制工具，根据 Dockerfile 配置文件。Java 专属配置是专门为 Java 项目打造的，对编译和运行环境进行了分离，使镜像更加精简。还可以使用代码项目内的 Dockerfile，无需自行配置构建参数生成 Dockerfile。

工程名称: 必填项，指定该工程名称。

点击下一步，将对新创建的工程进行进一步的配置。以下分四种情况详细介绍相关配置：

1.通用配置

如果选择通用配置，将在本步骤通过几项简单的配置生成一个 **Dockerfile**，用于构建您的工程。可以选择手动配置，也可以点击"复制已有工程"来复制另一个工程的配置信息。主要配置参数说明：



复制已有工程

基础镜像	<input type="text" value="搜索基础镜像"/>
安装依赖命令	<div></div>
代码存放路径 *	镜像内存放代码的路径，要求绝对路径，例如：/code
编译命令	多条命令请用“&&”连接。 <div></div>
工作目录	要求绝对路径，默认为根目录
启动命令 *	多条命令请用“&&”连接。 <div></div>



user in docker	<div></div>
编译过程环境变量	build过程的环境变量。会被写入Dockerfile，在运行期间也会生效。 <div>环境变量名<div>环境变量值</div></div>
配置文件模板	配置文件模版中，用“{{Env.PATH}}”来表示名称为“PATH”的变量。 <div>模板路径<div>目标路径</div></div>
运行过程环境变量	请列出工程运行镜像需要配置的环境变量，给予默认值并加以描述， <div>环境变量名<div>环境变量值<div>环境变量描述</div></div></div>

基础镜像：用于构建的基础镜像，默认将给出镜像管理中配置的基础镜像列表用于备选。

安装依赖命令：用于安装编译依赖项的命令。

代码存放路径：镜像内存放代码的路径，代码仓库中的代码将被下载到此路径。(注意要求为绝对路径)

编译命令：执行代码编译的命令。(多条命令使用&&连接)

工作目录：代码工作路径。(注意要求的为绝对路径，默认为根目录)

启动命令：代码启动命令。(多条命令使用&&连接)

user in docker： 指定容器中用户。

编译过程环境变量：编译过程中的环境变量，将被写入 Dockerfile，构建、运行过程中都会生效。

配置文件模板：若需使用 dockerize 配合配置模板生成配置文件，这里需要配置模板文件在代码项目中的路径以及生成配置文件的目标路径，可配置多对。

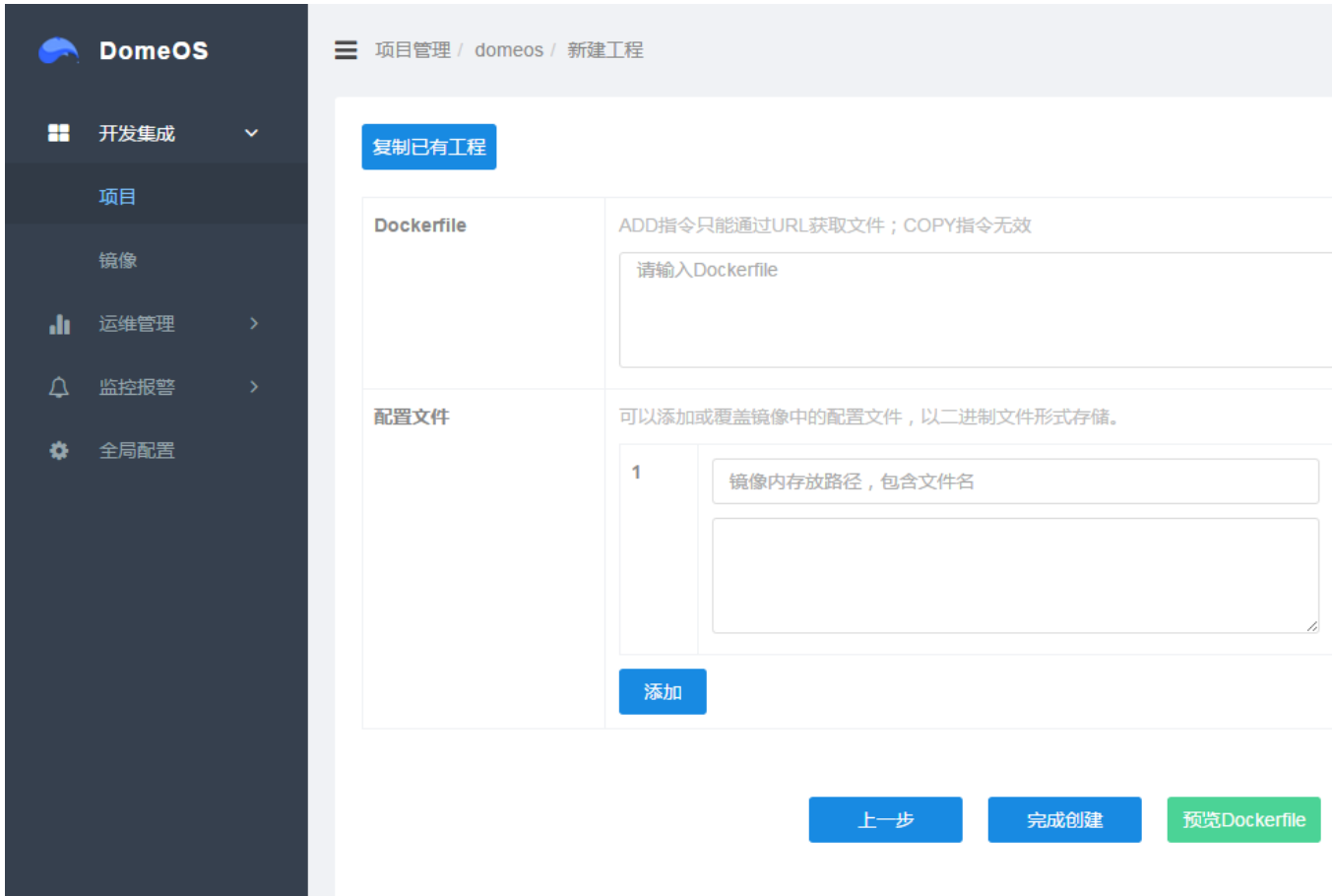
运行过程环境变量：在项目中声明并给出默认值和描述，用于提示部署运维人员在部署和升级等操作时配置。注意：此处的环境变量与构建过程无关。

配置完成后，可以点击"预览 Dockerfile"查看根据配置生成的 Dockerfile 的内容。

点击"完成创建"即可完成工程的创建。

2. 定制配置

选择定制配置可以选择手动配置，也可以点击"复制已有工程"来复制另一个工程的配置信息。选择后将进行以下配置：



编辑 Dockerfile:在此处填写 Dockerfile，编写 Dockerfile 需要注意的是 ADD 指令只能通过 URL 获取文件；COPY 指令无效。也可直接选择基础镜像进行配置。


添加配置文件: 可以指定文件名和在基础镜像中的存放路径，点击编辑添加文件内容；

配置完成后，可以点击"预览 Dockerfile"查看根据配置生成的 Dockerfile 的内容。

点击"完成创建"即可完成工程的创建。

3.Java 专属配置

Java 专属配置需要填写编译信息和运行信息，可以选择手动配置，也可以点击"复制已有工程"来复制另一个工程的配置信息。

DomeOS

开发集成

项目

镜像

运维管理

监控报警

全局配置

复制已有工程

编译镜像

编译镜像用于代码编译并打包，请在编译镜像内配置好编译环境。

DomeOS官方镜像库

DomeOS私有镜像库 ?

镜像仓库地址：<http://pub.domeos.org>

Q

当前选择：domeos/compileimage-java:maven-3.3.9_jdk-8

编译镜像内代码存放路径。请确认该路径下可以执行编译命令。（必填）

代码存放路径，要求绝对路径，例如：/code

编译结果在编译镜像中的存放路径。可填写多个。（至少填一个）

请填写绝对路径。文件夹以"/"结尾，文件以文件名结尾

编译命令。多条命令请用"&&"连接。（必填）

编译过程的环境变量

环境变量名

环境变量值

开发集成

项目

镜像

运维管理

监控报警

全局配置

运行镜像

运行镜像用于运行编译结果，请在运行镜像内配置好运行环境。

DomeOS官方镜像库

DomeOS私有镜像库

?

镜像仓库地址：http://pub.domeos.org

当前选择：domeos/runimage-java:tomcat-8_jre-8

编译结果在运行镜像中的存放路径。只能填写一个。（必填）

/usr/local/tomcat/webapps/

运行镜像的启动命令。多条命令请用"&&"连接。（必填）

catalina.sh

请列出工程运行镜像需要配置的环境变量，给予默认值并加以描述，用

环境变量名	环境变量值	描述：20字以内
-------	-------	----------

工作目录

要求绝对路径，默认为根目录

user in docker

配置文件模板

配置文件模板中，用"{{Env.PATH}}"来表示名称为"PATH"的变量。

模板路径

目标路径

编译镜像：DomeOS 官方仓库中提供了部分 maven 和 jdk 镜像可以用于 Java 项目构建，也可以选择私有仓库中镜像用于构建。使用时，还需指定编译镜像内代码存放路径、编译结果在编译镜像中的存放路径、编译命令、编译过程的环境变量。

运行镜像：DomeOS 官方仓库中提供了部分 tomcat 和 jre 镜像可用于运行 Java 项目，也可以选择私有仓库中镜像用作为运行镜像。使用时还需指定编译结果在运行镜像中的存放路径、运行镜像的启动命令、运行镜像需要配置的环境变量。

工作目录：代码工作路径。(注意要求的为绝对路径，默认为根目录)

user in docker：指定容器中用户。

配置文件模板：若需使用 dockerize 配合配置模板生成配置文件，这里需要配置模板文件在代码项目中的路径以及生成配置文件的目标路径，可配置多对。

点击"预览 Dockerfile"可以查看用于生成运行镜像的 Dockerfile。

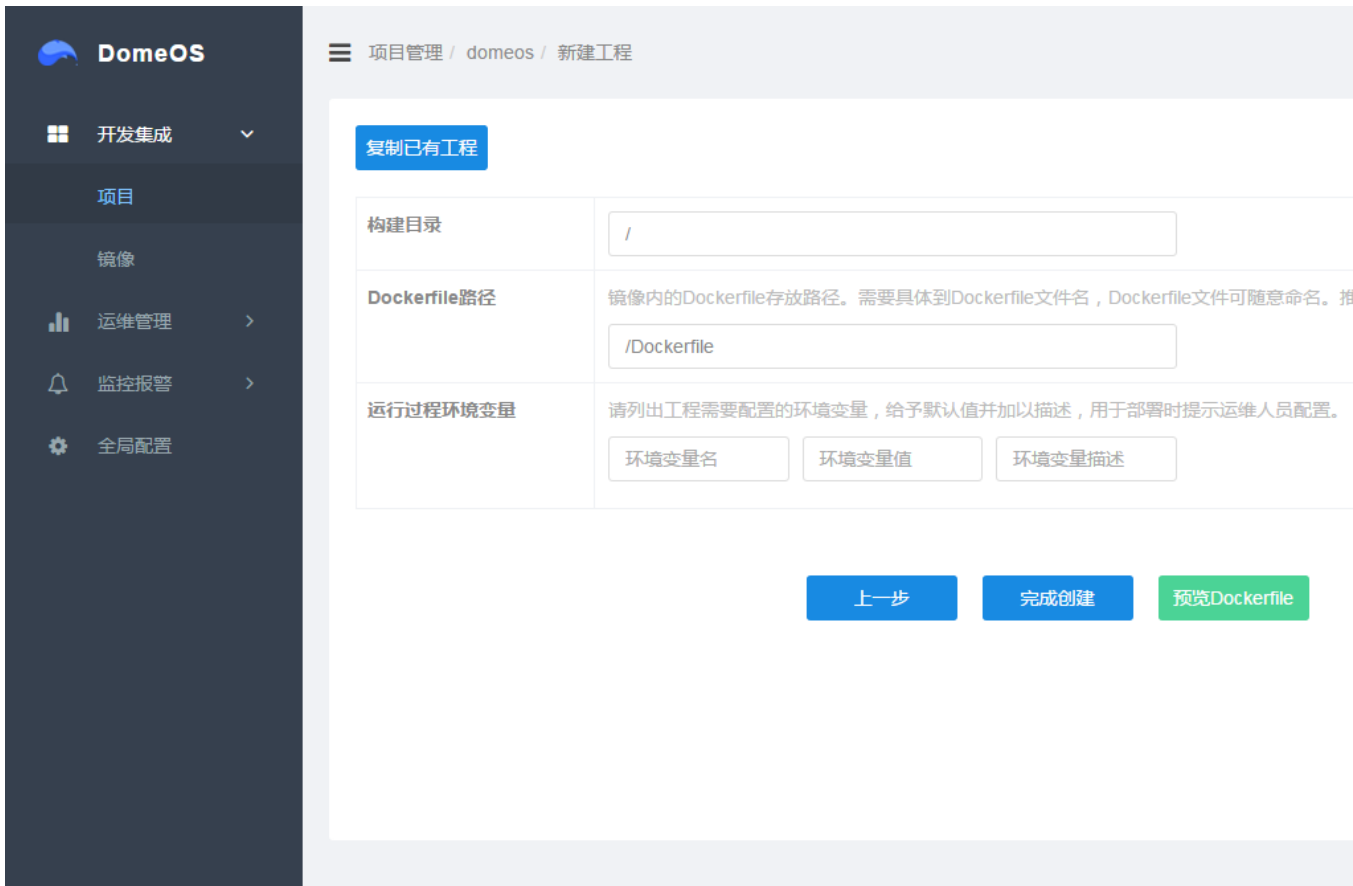
点击"完成创建"即可完成工程的创建。

DomeOS 官方仓库所提供的编译与运行镜像列表

pub.domeos.org	docker hub
domeos/compileimage-java:maven-3.3.9_jdk-7	maven:3.3.9-jdk-7
domeos/compileimage-java:maven-3.3.9_jdk-8	maven:3.3.9-jdk-8
domoes/runimage-java:tomcat-7_jre-7	tomcat:7-jre7
domoes/runimage-java:tomcat-7_jre-8	tomcat:7-jre8
domoes/runimage-java:tomcat-7_jre-8	tomcat:7-jre8
domoes/runimage-java:tomcat-8_jre-7	tomcat:8-jre7
domoes/runimage-java:tomcat-8_jre-8	tomcat:8-jre8
domoes/runimage-java:jre-7	tomcat:7-jre
domoes/runimage-java:jre-8	tomcat:8-jre

4.使用代码项目内的 Dockerfile

这种情况下无需在 DomeOS 中配置构建参数生成 Dockerfile，仅需配置以下几项参数即可。可以选择手动配置，也可以点击"复制已有工程"来复制另一个工程的配置信息：



构建目录：代码项目内执行 `docker build` 命令的路径参数。

Dockerfile 路径：镜像内的 `Dockerfile` 存放路径。需要具体到 `Dockerfile` 文件名，`Dockerfile` 文件可随意命名。推荐在构建目录下存放 `Dockerfile`。

运行过程环境变量：在工程中声明并给出默认值和描述，用于提示部署运维人员在部署和升级等操作时配置。注意：此处的环境变量与构建过程无关。

点击"预览 `Dockerfile`"可以查看用于生成镜像的 `Dockerfile`，可以选择 `Dockerfile` 所在分支或 `tag`。

点击"完成创建"即可完成工程的创建。

不使用代码仓库生成工程

在新建工程第一步选择不使用代码仓库，可不通过代码仓库生成工程。创建过程中需使用"通用配置"或者"定制化配置"生成 `Dockerfile`，用于构建工程。

镜像管理

镜像是一个关键的交付单元。镜像通常由项目构建而成。开发人员把自己的代码配置成一个项目后，把项目构建成一个镜像文件。运维人员把镜像文件部署到对应的运行环境中。因此，需要一个专门针对镜像的管理页面。

在 DomeOS 的镜像管理中，您可以查看所有的私有仓库中的镜像文件，以及镜像的版本列表。**DomeOS** 按照项目名对镜像进行了分类，方便查看。镜像管理还记录了所有用于构建工程的基础镜像，您可以添加新的基础镜像，满足工程构建的需求。

私有仓库里已收集了部分常用的基础镜像，需要新的基础镜像或需要升级镜像版本可联系我们添加到私有仓库中。

开发集成

项目

镜像

运维管理

监控报警

镜像管理 / 其他镜像

仓库地址：http://172.17.22.32

共67个镜像

chaincore/developer			版本数：1
版本名称	拉取命令	创建时间	
1.1.1	docker pull 172.17.22.32/chaincore/developer:1.1.1	2017-03-09 09:22:32	
code/alpine			版本数：1
code/drone			版本数：2
code/gitlab			版本数：12
code/gitlab-ci-multi-runner			版本数：1
code/gitlab-runner			版本数：2

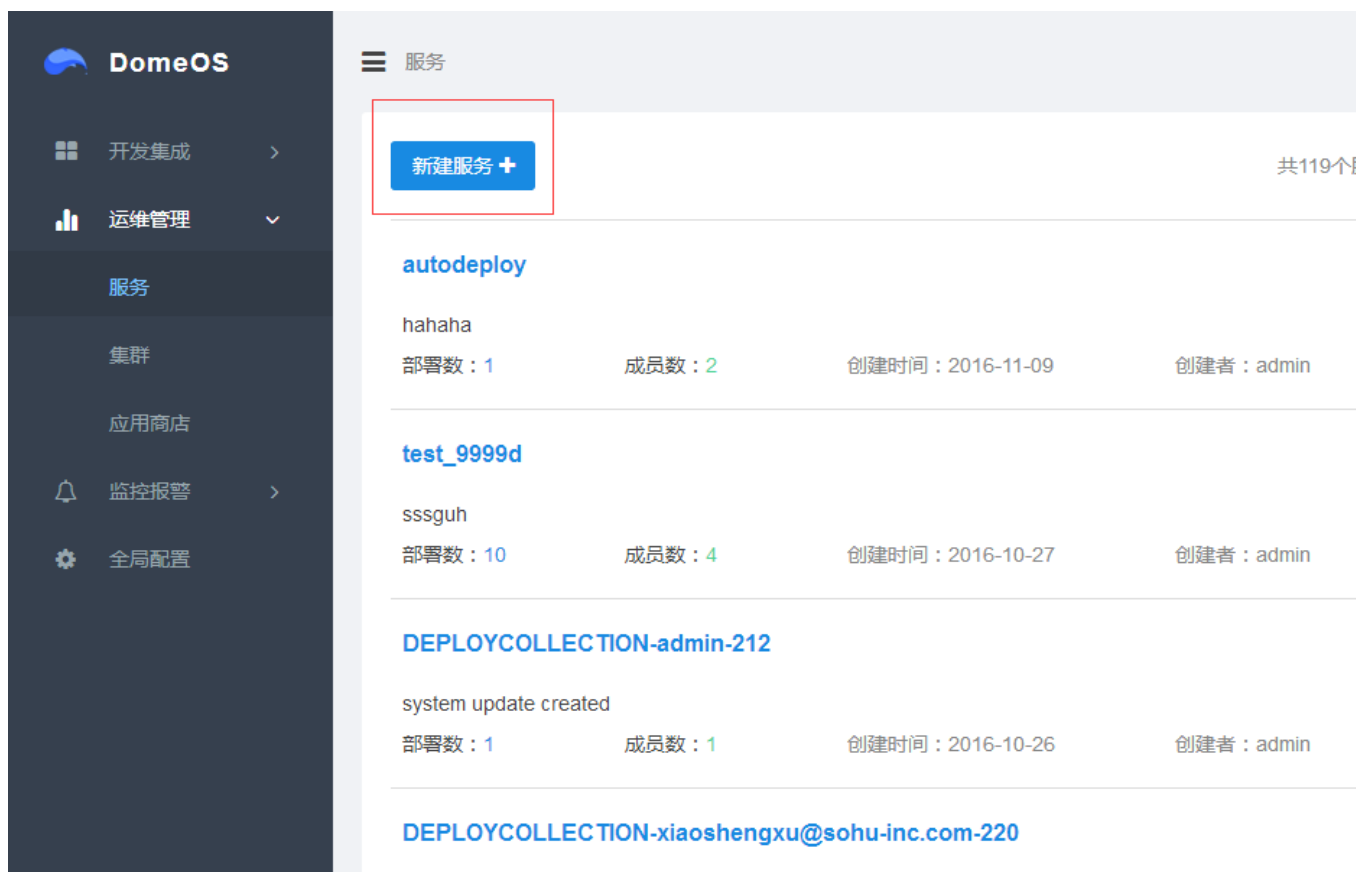
四、 运维管理（启动项目容器）

运维管理包括了 **docker** 容器运行时的完整生命周期，用户开发完成后，可以将自己的镜像部署到生产或测试环境，对其进行各种操作。

运维管理由服务管理、集群管理和应用商店三部分组成。服务和集群分别拥有自己的用户权限。

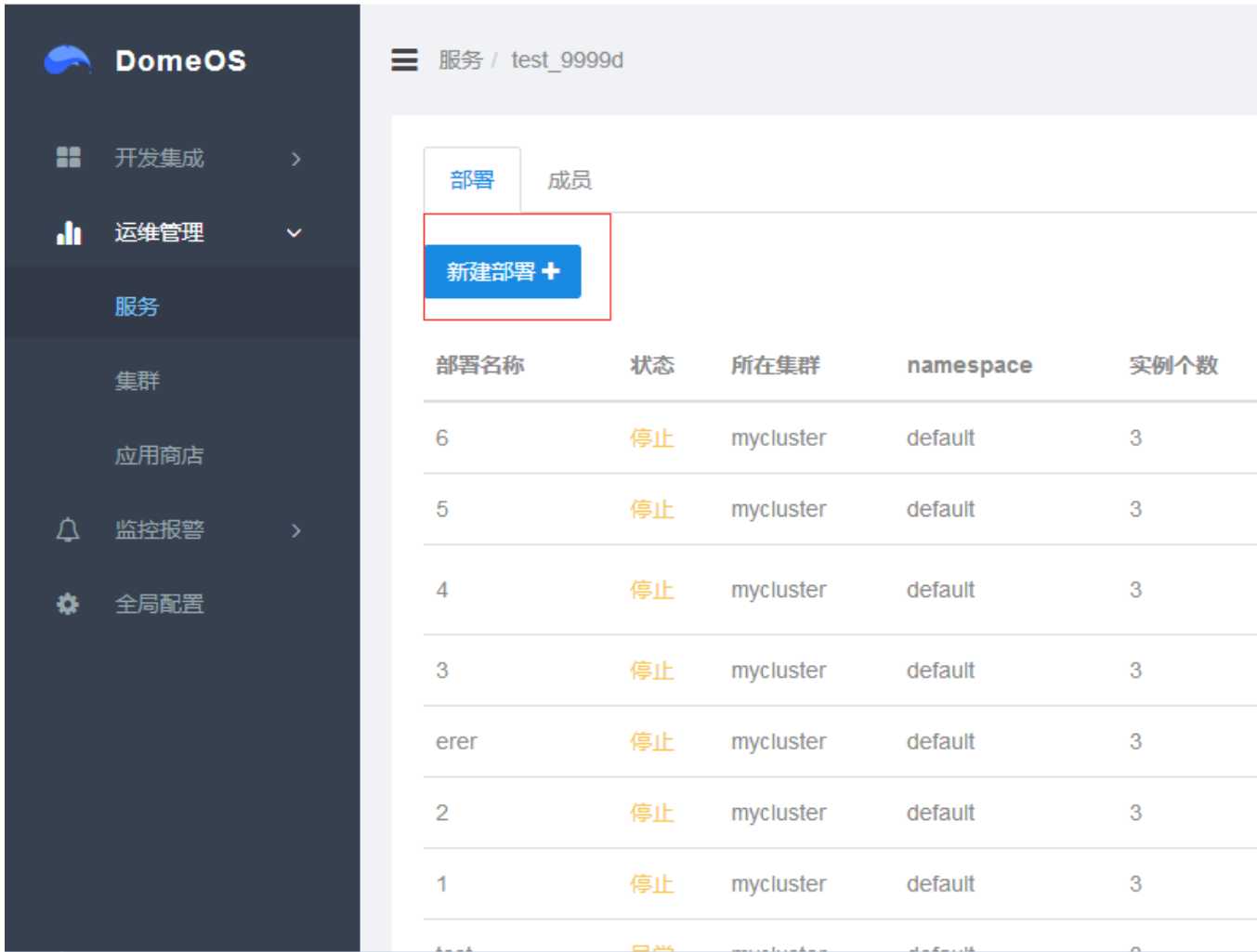
新建服务

用户可以点击“运维管理”中“服务”，跳转至服务列表后，点击新建服务进行新建。



新建部署

在部署列表页面上点击“新建部署”可将镜像部署到服务内。



部署配置

默认配置

默认类型需要进行一下配置：

- 1. 部署名称：该名称会作为内网域名的一部分，要求符合 dns 命名规范。
- 2. 部署描述：部署的详情描述，默认为空，可不填。
- 3. 部署类型：必须选 **Deployment**。
- 4. 实例个数：开发测试只需开 1 个就行，尽量避免浪费服务器资源。
- 5. 集群：在已有集群中选择需要部署的集群。
- 6. 工作环境：生产或测试，该项在集群的主机配置中设置。
- 7. 主机标签：您可以查看筛选出了哪些主机，系统会在您筛选出的主机上部署您的应用。
- 8. namespace：您需要选择一个已有的 namespace 或新建一个 namespace
- 9. 网络模式：您可以选择使用 overlay 或 host 网络模式部署。 注意：host 模式下不能使用健康检查；更多 host 相关说明，详见“Host 网络”部分。

10. 访问设置：必须选对外服务，影射容器内端口到宿主机上（如：容器在外部是访问不到的，将容器 8080 端口影射到某个主机 IP 的 8080 端口，使用影射出来的宿主机 IP：8080 来访问容器页面）。在 overlay 网络下，可以选择“禁止访问”、“对内服务”、“对外服务”。对内服务的部署，只能在集群内访问；对外服务的部署，可以在集群内通过内部域名访问，在集群外通过对外访问 ip+对外访问端口访问。在 host 网络下，可以选择“禁止访问”和“允许访问”。如果允许访问，则需要指定暴露的端口数。

DomeOS

开发集成

运维管理

服务

集群

存储

应用商店

监控报警

全局设置

服务 / domeos / 新建部署

部署名称 *

部署名称会作为部署内部域名的一部分，创建后不可修改。

不能有大小写字母，不能以中横线开头和结尾

部署描述

输入部署描述信息

部署类型

以replicas的形式保证pod的实例个数，部署的升级/回滚/扩容/缩容由DomeOS负责维护处理。

ReplicationController (默认)

DaemonSet

PetSet

Deployment

配置类型

默认类型

YAML

JSON

实例个数

实例在集群内启动副本数

3

↑

集群

请选择部署启动的集群，选定后可以点击查看选中主机，以确保后台有资源启动该部署。如果无可选集群，请在“运维管理->集群”中添加集群或联系管理员添加已有集群

bx-42-205

工作场景

测试环境

生产环境

主机标签

选择主机标签

查看选中主机

namespace

default

新建namespace

网络模式

Overlay模式下采用flannel管理容器网络，Host模式下容器网络配置与宿主机一致

Overlay

Host

访问设置

禁止访问

对内服务

对外服务

该部署不需要被其他应用发现时请选择禁止访问。

取消

下一步

配置好之后，点击“下一步”进行存储与容器配置。

11. 存储配置（非必选）：DomeOS v0.5 版本之后支持了部署存储配置的支持，目前 DomeOS 支持的存储类型有主机目录和实例内目录两种。

存储设置

添加

如果需要在容器中挂载存储，需要先在此处进行配置

(名称未填写) ×

类别

实例内目录

该类别创建的存储，可以被示例内容容器共同访问，当实例调度时，存储内容会被回收

名称

主机目录

名称在该部署内唯一不可重复

实例内目录

12. 镜像配置：您要选择需要部署的镜像。由于 DomeOS 集群采用 Kubernetes 架构，您可以在一次部署时选择多个镜像同时部署。每个镜像会启动一个容器，各个镜像的容器整体构成一个实例（Pod）。实例是升级回滚、扩容缩容等部署运维管理的最小单位。

选好镜像后，您需要对每个镜像进行配置。选择镜像版本、是否开启自动部署、添加环境变量、设定拉取策略和健康检查、添加日志。

- **存储挂载（非必选）**：当使用存储配置之后需要在对应镜像这里配置存储挂载路径，指定该存储所挂载容器的具体位置，挂载类型又只读和读写两种，路径需要指定到运行容器的绝对路径。
- **自动部署（非必选）**：开启后，当通过工程构建产生该镜像的新版本时，会自动触发升级。该功能应该只针对运行中状态的部署生效，其他状态的部署不受影响。若运行中部署同时存在多个版本，则不触发自动升级。拥有 **developer** 以上权限的人可以配置跟随升级，跟随升级不需要在项目中配置。

The screenshot shows the DomeOS web interface. On the left is a dark sidebar with navigation items: 开发集成, 运维管理, 服务, 集群, 应用商店, 监控报警, and 全局配置. The main content area is titled '服务 / flume-test / 新建部署'. It contains several configuration sections: '选择镜像' with a search bar and a selected image 'compileimage-java'; '镜像仓库' set to 'https://registry-in.sohucs.com'; '镜像版本' with a dropdown '选择镜像版本'; '自动部署' toggle switch (currently off) with a description; '运行过程环境变量' with a '+' button; '容器大小' with input fields for 'CPU(个)' (0.5) and '内存(MB)' (1024); and '镜像拉取策略' with radio buttons for 'Always' (selected), 'Never', and 'If Not Present'.

- **日志（非必选）**：需要输入日志文件的所在路径，并选择是否自动收集和自动删除。

日志收集采用 **kafka+flume** 形式，通过 **flume** 采集日志并上传到 **kafka** 里，要使用日志收集功能需要在集群下配置日志收集参数，日志收集功能只针对默认类型配置。日志收集时我们会在启动 **pod** 中增加一个 **flume** 镜像，镜像内容可以参

考：<https://github.com/domeos/domeos-flume> 可以根绝该开源 flume 镜像进行相应的定制化。 日志收集分为自动收集日志与自动删除日志：

×

☒ 自动收集日志

日志topic

预处理命令
?

☐ 自动删除日志

×

☐ 自动收集日志

☒ 自动删除日志

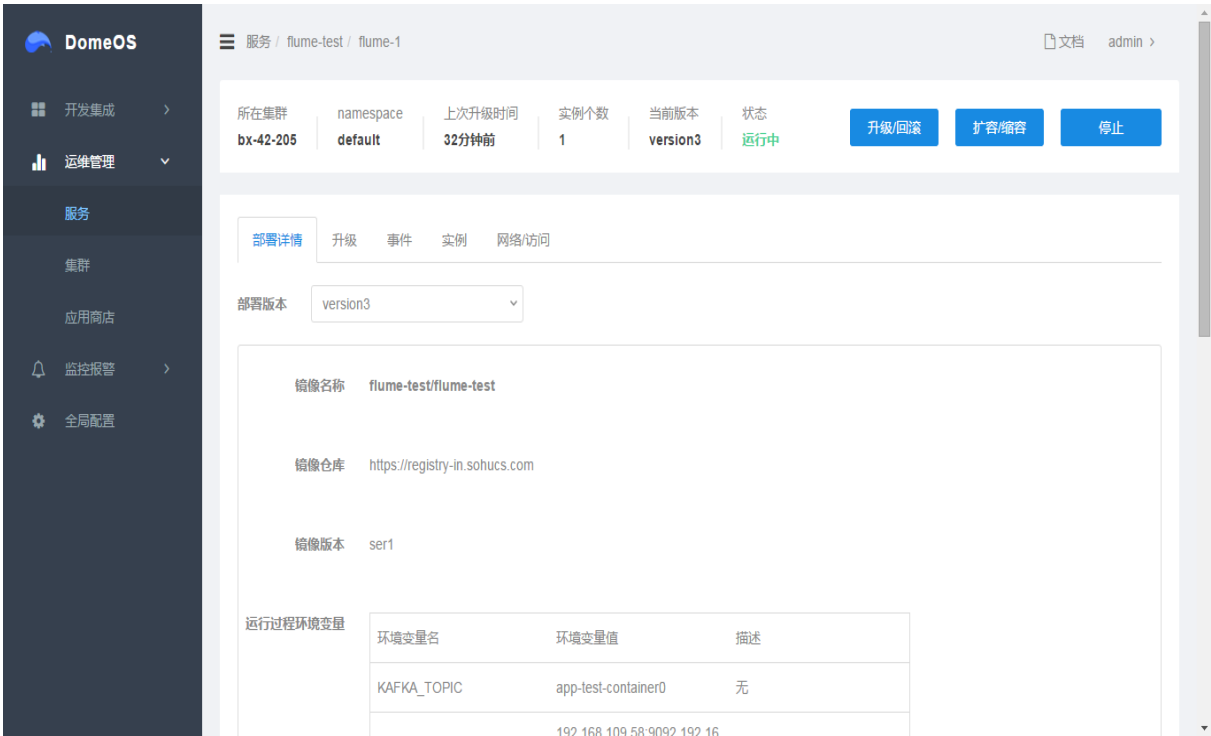
过期时间 小时

添加

如上图所示，自动收集日志与删除日志都需要针对到具体的文件名 日志 topic 是需要提前在 kafka 种创建 flume 的日志收集镜像是通过配置了 `tail -F $LOGFILE` 进行日志，可以在之后增加具体的预处理命令，该命令必须以 `"l"` 开头 删除日志是距上次文件修改时间起多少小时触发删除命令。

完成所有配置后，点击“完成创建”。

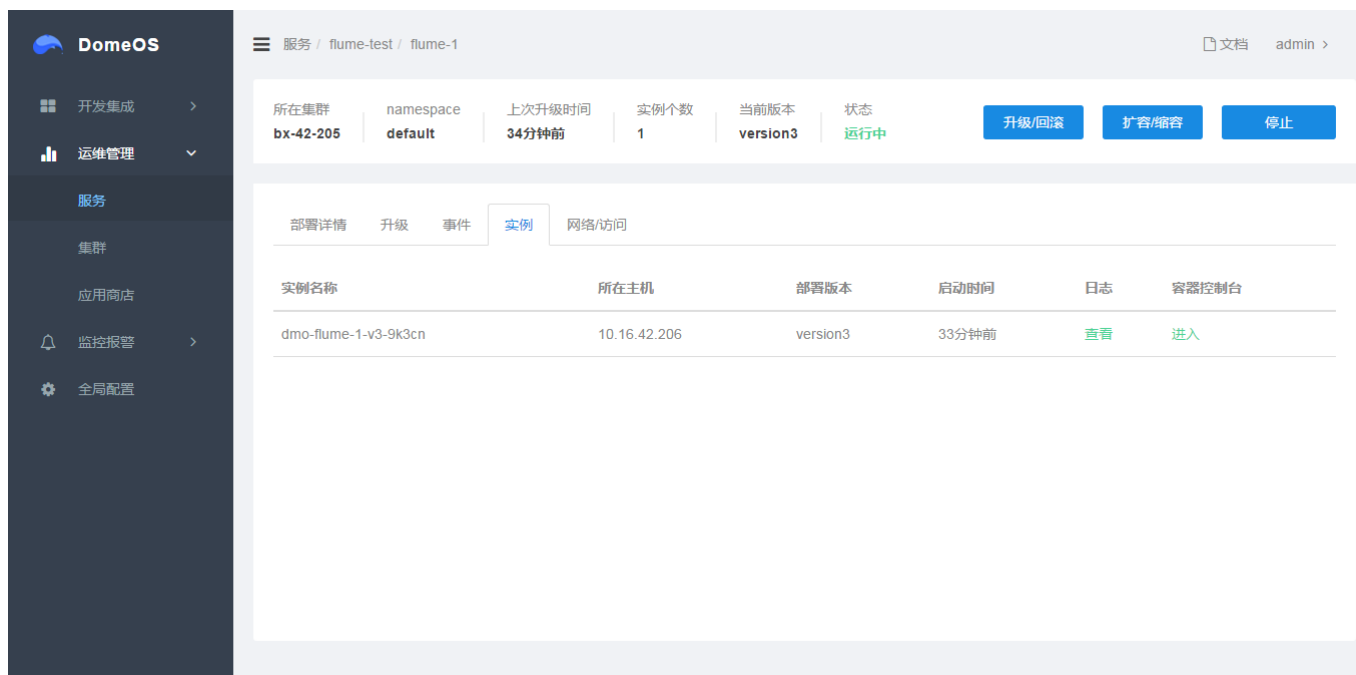
服务操作



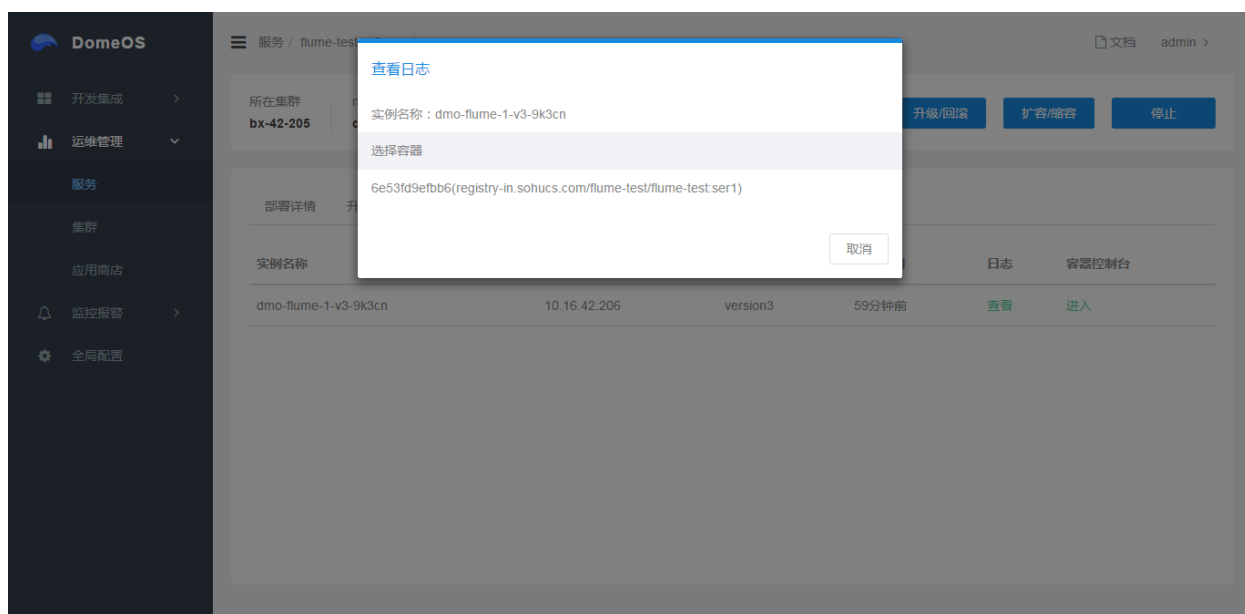
服务操作包含了“启动/停止”、“扩容/缩容”、“升级/回滚”等功能。

容器日志和控制台

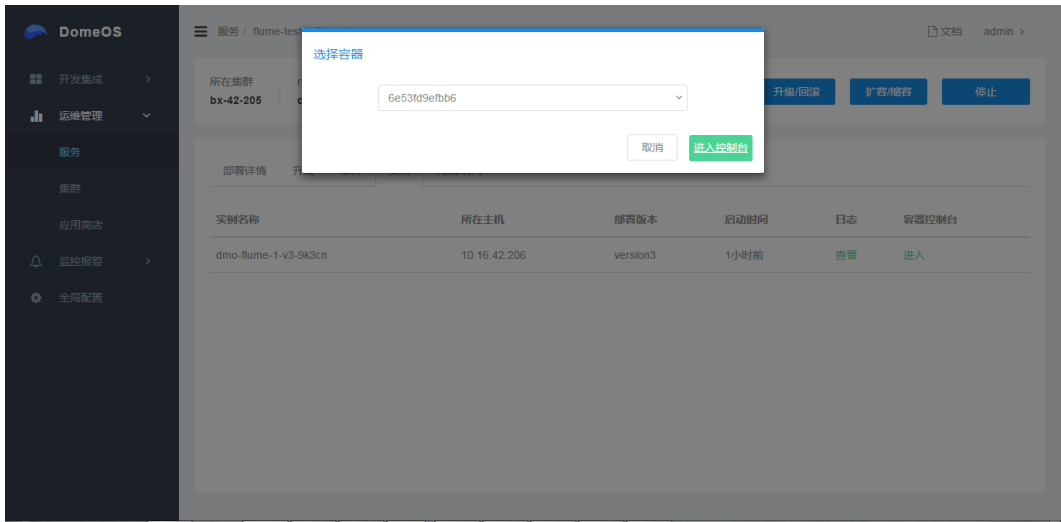
您可以在“实例”页面里看到部署正在运行的实例。



在日志栏点击“查看”，可以在新窗口查看实例内各个容器的日志。



此外，我们提供了 Web 端容器 SSH 登录的功能，在容器控制台栏点击"进入"即可打开容器命令行，方便查看容器环境并调试应用。



```
Welcome to domeos console!  
bash-4.2#
```

五、 参考资料

以上文档内容为必要配置项的说明描述，
需要更完整更全面的参考资料可访问以下地址：

<http://gitbook.domeos.org/>