面向 Dockerfile 的容器镜像构建工具①

耿 朋[1,2], 陈 伟[2], 魏 峻[2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院软件研究所 软件工程技术研究开发中心, 北京 100190)

摘 要: 在容器虚拟化中, 主流的镜像构建方式是通过 Dockerfile 来构建的. 然而, 在使用 Dockerfile 构建镜像时存在着明显的不足: 由于 Dockerfile 语言的复杂性, 使用文本编辑方式没有提供有效的语法引导, 没有对 Dockerfile 可能存在的错误进行有效的检测, 导致构建容器镜像工作效率低下. 此外, 使用说明不完整的第三方镜像, 无法有效的确定镜像的功能和使用方法, 安全性也是第三方镜像所面临的一大挑战, 这带来了容器镜像的重复利用率低下的问题. 针对上述问题, 在分析 Dockerfile 语法和统计分析 Dockerfile 常见错误以及深入研究 Docker 镜像存储机制的基础上, 设计了一个面向 Dockerfile 的镜像构建工具, 并使用了可视化编辑, 错误检测, 逆向分析等关键技术进行实现. 该工具能够在镜像构建中提供有效的语法引导, 对 Dockerfile 常见的错误进行有效检测, 为了验证第三方镜像的功能和安全性, 设计了一种由 Docker 镜像逆向生成 Dockerfile 的方式, 用户可以通过 Dockerfile 完全了解第三方镜像的功能和使用方式, 另外通过二次构建的方式也可一定程度上解决第三方镜像的安全性问题.

关键词: Docker 镜像; 错误检测; 逆向工程; Dockerfile

Tool for Building Docker Image on Dockerfile

GENG Peng[1,2], CHEN Wei[2], WEI Jun[2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: In container-based virtualization, the mainstream way to build images is Dockerfile. However, using Dockerfile to build images has the obvious deficiency: Because of complexity of the Dockerfile syntax, the way of editing with text does not provide effective guidance of syntax, and effective detection for common errors, which lead to low efficiency in building images. In addition, using the 3rd party images with no complete explanation, users can not completely understand the usage of images and security is also a challenge for the 3rd party images, this bring the difficulty to reuse images. To solve the above problem, based on analysising syntax of Dockerfile, statisticsing and analysising common errors and deeply studying on storage mechanism of Docker image, we design a tool for building Docker image on Dockerfile, which using technology of visual editing, error detection and reverse engineering. In view of the problems existing in building images, the tool can provide effective syntax guide, and detect the common errors effectively. In addition, in order to verify the usage and safety of the third party images, by reversing images to Dockerfile, users can fully understand usage of the third party images via Dockerfile, and through the way of second build, the tool can solve problem of safety of the third party images partly.

Key words: Docker image; error detection; reverse engineering; Dockerfile

1 背景

随着云计算、大数据技术的快速发展, 加之企业业务需求的不断变化, 导致企业架构要随时更改以适合业务需求, 如何高效协调, 快速交付产品, 快速部署

① 基金项目:国家自然科学基金(61402453);国家高技术研究发展(863)计划(2013AA041301)
收稿时间:2016-03-11;收到修改稿时间:2016-04-05 [doi:10.15888/j.cnki.csa.005438]

应用, 以及满足企业业务需求, 是开发人员亟需解决的问题, Docker 技术因此应运而生. 基于 LXC 的开源容器引擎 Docker 为 PaaS 平台提供了一种在安全、可重复的环境中自动部署软件的方式. Docker 的最终目的是实现“Build, Ship, and Run Any App, Anywhere”, 即通过对应于组件的封装、分发、部署、运行等生命周期的管理, 达到应于组件级别的一次封装, 到处运行[7].

Docker 的核心要素有镜像、容器和仓库. 镜像是构建 Docker 世界的基石, 是一个面向 Docker 的只读模板, 包含文件系统, 用户基于镜像来运行自己的容器, 容器是从镜像创建的运行实例[7]. 镜像是基于联合文件系统的一种层式结构, 由一系列指令一步一步构建出来, 可以把镜像当作容器的源码, 镜像体积很小, 易于分享、存储和更新.

Dockerfile 是构建镜像的基本方式, Dockerfile 是一个指令集, 每行均为一条指令. 构建容器镜像的工作流程为: Docker 从基础镜像运行一个容器, 执行一条指令, 对容器做出修改, 提交一个新的镜像层, Docker 再基于刚提交的镜像运行一个新的容器, 再执行 Dockerfile 中的下一条指定, 直到所有指令都执行完毕. Dockerfile 脚本可以做到随时维护修改, 即可以分享, 更有利于在模板化, 有利于传输[7].

在传统的镜像构建方式中, 通过使用文本编辑 Dockerfile 的方式缺乏有效的语法引导, 而 Dockerfile 语言的复杂性则要求编写者十分熟悉 Dockerfile 的语法, 这带来了很大学习成本. 此外 Dockerfile 的任何错误均会导致镜像构建失败, Docker 系统本身也没有对 Dockerfile 常见类型错误提供有效的检测方案, 进而验证 Dockerfile 的正确性. 本文通过在 DockerOne.io, 开源中国, Stack Overflow 以及使用搜索引擎检索的方式查找收集 Dockerfile 常见错误的基础上, 将 Dockerfile 常见错误主要分为语法错误以及语义错误两种类型, 其中语义错误又主要包括: 引用的文件不存在, CMD 命令出现多条, 设置环境变量与系统环境变量冲突, shell 命令与基础镜像不匹配以及软件版本依赖兼容性问题等 5 种类型.

另一方面, 很多常用镜像在官方 Docker Hub 上是存在的, 且得到很好的更新和维护, 选择使用在 Docker Hub 中存在的容器镜像可以抛开后期镜像维护和更新的繁琐工作, 提高工作效率. 然而 Docker Hub 上的很多第三方镜像缺少完整的功能说明和使用方式说明, 也没有提供构建镜像的 Dockerfile 文件, 这使得用户对第三方镜像的功能和使用方法没有清晰的了解, 甚至很多第三方镜像在构建过程中添加了一些恶意代码文件, 安全性也是第三方镜像需要面对的一大挑战[5].

面对上述问题和挑战, 本文着重考虑了三个方面的问题: (1)研究语法引导的 Dockerfile 编辑方法, 能够降低 Dockerfile 构建时对领域知识的要求, 避免出现语法错误; (2)通过分析总结 Dockerfile 编写过程中常见的错误, 研究错误检测方法, 对 Dockerfile 常见错误类型进行检测; (3)研究第三方 Docker 镜像的功能验证方法, 以及通过分析镜像内容的方式来提供安全性保障.

2 相关工作

Eclipse 发布新版本 Mars 支持提供了全新的 Docker 工具. 针对程序员在使用 Docker 时在 IDE 和 Docker 环境下来回切换的问题, Mars 提供了一种简单的方式直接从 Eclipse 中启动、停止和部署 Docker 容器. 该工具以 Eclipse 插件的形式不仅能与现有的 Docker 命令行工具共同工作, 而且用于提供更好的可视角度访问常用操作. 该插件提供可视视图列表供开发人员管理镜像、容器以及 Docker 实例, 但是该插件只是针对开发人员的需求对镜像和容器提供了高级的管理方式, 没有针对运维人员的需求提供对 Dockerfile 的支持. 该工具并不适用于面向 Dockerfile 构建容器镜像的场景.

针对 Dockerfile 构建容器镜像的编辑问题, github 上发起的支持 Dockerfile 编辑、构建和容器运行的 Eclipse 插件 doclipser 项目, 提出了在 IDE 内构建系统和运行环境的需求, 该插件支持在 Eclipse 内创建、编辑 Dockerfile 和构建镜像[2]. 目前主要支持的功能包括编辑 Dockerfile 带有语法高亮显示, 剩余命令提示和自动补全以及语法验证三个方面. 该插件的主要思路来源于传统的 Vim、Emacs 等文本编辑器功能, 然而一方面构建 Dockerfile 过程中常见的错误往往并非语法层面的, 而且语法错误可以通过有效的语法引导编辑方式来进行避免; 另一方缺乏对非语法错误的检测, 例如设置环境变量与系统环境变量冲突, shell 命令与基础镜像不匹配, 包含多条 CMD 命令等问题均是 Dockerfile 编辑过程中常见的错误. 因此该类型的工具

对面向 Dockerfile 场景下的编辑和错误检测并不适用.

Zhang 等人研究了一种组合测试建模方法来进行软件兼容性测试[6], 通过构造合适的软件集测试模型, 将软件集作为输入生成数量少覆盖率高的组合测试用例集. 在被测软件集上运行测试用例集得到相应的测试结果, 将测试用例集以及执行结果作为组合测试错误定位工具的输入, 从而精确定位引起兼容性错误的软件. 该方法简化了错误定位的流程和复杂性, 但是该方法是在错误发生后进行检测定位, 而 Dockerfile 的错误检测需求集中于编辑过程中, 因此该方法并不适用于面向 Dockerfile 场景下的软件兼容性检测.

在文献[9]中, Zhang 人研究发现, 在软件配置文件错误检测中, 配置参数与系统环境之间存在关联关系. 例如配置参数 A 引用了系统文件 B, 而该系统环境中 B 的存在与否影响了 A 取值的正确性, 关联关系是导致配置文件错误的原因之一, 张因此实现了基于配置关联的错误检测工具 Encore, 该工具基于已有的配置文件作为训练样本集, 根据配置参数的取值特征为其附加相应的环境信息, 通过预先定义约束模板来描述配置参数和环境信息之间的关联关系. 基于模板集合从训练集中抽取满足模板预定义的配置参数进行实例化, 从而产生一组约束规则, 通过该约束规则对目标系统进行错误检测. 该方法对单个系统内的配置文件因为关联关系而导致的错误有很好的效果, 但是因为该工具的目标对象是系统配置文件, 对 Dockerfile 场景并不适用, 因此需要研究该方法面向 Dockerfile 的适用场景.

3 相关技术背景

3.1 Dockerfile 语法介绍

Dockerfile 是一种 DSL 语言(领域专用语言),是专门针对构建 Docker 镜像问题来设计的, 其基本思想是“求专不求全”, 语法格式类似于 SQL 语法, Dockerfile 包含创建镜像所需要的全部指令. 基于在 Dockerfile 中的指令, 我们可以使用 Docker build 命令来创建镜像. 通过减少镜像和容器的创建过程来简化部署. Dockerfile 支持支持的语法命令如下: INSTRUCTION argument, 指令不区分大小写. 但是, 命名约定为全部大写[2].

有关 Dockerfile 的相关命令介绍如下:

(1) FROM <image name>: 在 Dockerfile 中第一条非注释指令一定是 FROM, 它决定新的镜像将基于哪个基础镜像来构建, <image>首选本地存在的, 如果不存在则会从公共仓库下载.

(2) RUN <command>: RUN 指令会在新创建的镜像上添加新的层面, 接下来提交的结果用在 Dockerfile 的下一条指令中.

(3) ENTRYPOINT command param1 param2: 配置给容器一个可执行的命令, 这意味着在每次使用镜像创建容器时一个特定的应用程序可以被设置为默认程序. 同时也意味着该镜像每次被调用时仅能运行指定的应用.

(4) CMD command param1 param2: 提供了容器默认的执行命令. Dockerfile 只允许使用一次 CMD 指令. 使用多个 CMD 会抵消之前所有的指令, 只有最后一个指令生效.

(5) EXPOSE <port>: 指令告诉容器在运行时要监听的端口, 但是这个端口是用于多个容器之间通信用的(links), 外面的 host 是访问不到的.

(6) ADD <src>... <dest>: 复制文件指令. 它有两个参数<source>和<destination>. destination 是容器内的路径. source 可以是 URL 或者是启动配置上下文中的一个文件.

(7) COPY: COPY 的语法与功能与 ADD 相同, 只是不支持上面讲到的<src>是远程 URL、自动解压这两个特性.

(8) ENV <key> <value>: 设置环境变量. 它们使用键值对, 增加运行程序的灵活性.

(9) VOLUME: VOLUME 指令用来在容器中设置一个挂载点, 可以用来让其他容器挂载以实现数据共享或对容器数据的备份、恢复或迁移.

(10) WORKDIR: 该指令用于设置 Dockerfile 中的 RUN、CMD 和 ENTRYPOINT 指令执行命令的工作目录(默认为/目录).

(11) ONBUILD: 该指令用来设置一些触发的指令, 用于在当该镜像被作为基础镜像来创建其他镜像时执行一些操作.

(12) MAINTAINER <author name>: 设置该镜像的作者.

(13) USER <uid>: 镜像正在运行时设置一个 UID.

3.2 Docker 镜像存储结构研究

Docker 镜像是在 Docker 容器运行过程中主要提

供文件系统资源, 镜像主要由两部分组成镜像层文件内容和镜像元信息文件. Docker 镜像存储方式采用联合文件系统技术, 该技术可以把多个目录内容合并在一起, 而目录的物理位置是分开的, 只是逻辑上是相关联的. 联合文件系统是 Docker 镜像的基础. 镜像可以通过分层来进行继承, 基于基础镜像可以制作各种应用镜像. 如图 1 所示.

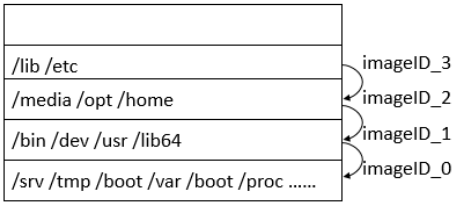

图 1 镜像分层存储图示

Docker 镜像文件的存储是由 Docker 系统本身来管理的, 有一定的标准规范, 文件结构, 存储目录等不能随意更改. 镜像每一层均可以理解为一个独立的镜像, 只不过镜像之间有依赖关系, 但是在存储结构上均是平行关系, 即按照 Docker 系统的标准以相同的结构和组织方式存在于相同的目录下, 没有物理上包含或被包含的关系, 只有逻辑上的关联性, 定位某层镜像的方式是通过镜像 ID.

```
├── containers
├── devicemapper
│   ├── devicemapper
│   │   ├── data
│   │   └── metadata
│   └── Mnt
├── graph
├── linkgraph.db
├── repositories-devicemapper
├── tmp
├── trust
└── volumes
```

图 2 Docker 镜像存储文件结构图

Docker 镜像存储系统的目录结构树状图如图 2 所示, 包括如下文件内容: Containers 文件夹用于保存容器的配置文件、环境变量文件以及日志文件; Decicemapper 文件夹下包含三个文件夹, Devicemapper 文件夹用于存储元文件, Metadata 文件夹记录镜像中间各镜像层信息, 包括 id, transation_id 以及是否初始化, Mnt 文件夹是容器运行时的挂载目录; Graph 保存镜像层级信息和容量大小信息等各镜像层详细信息; Linkgraph.db 文件是一个 sqlite3 数据库文件, 用于记录容器的链接关系; repositories-devicemapper 文件存储镜像的基本信息, 例如名称、tag、镜像 ID 等; tmp 文件夹是 Docker 容器运行时的临时目录; trust 文件夹作为 Docker 容器运行时的信任目录; volumes 文件夹是 Docker 容器的卷目录. 镜像存储的特点是在物理层面上各层镜像分开存储, 是平行的关系, 只是在逻辑上具有相互依赖关系, 依赖关系是由元信息文件描述的, 每层镜像均有一个元信息文件, 各层镜像通过镜像 id 来标识. 镜像的存储结构主要涉及到 graph, devicemapper 两个文件夹, graph 存储各层镜像元信息内容, 包括创建方式, 层级依赖关系, id 信息等, 实际存储中是通过建立与镜像 id 同名的文件夹来存储对应的元信息, devicemapper 则主要存储镜像的元文件. 元信息文件部分内容如图 2 所示.


```
{
    "id": "80431af78140afd2e38c98ebb33e54184cc1b18401b32382be8f6221ef411006",
    "parent": "964092b7f3e54185d3f425880be0b022bfc9a706701390e0ceab527c84dea3e3",
    "created": "2016-01-18T02:00:42.150761307Z"
    "config": {
        "Hostname": "",
        "Domainname": "",
        "User": "",
        "Env": null,
        "Cmd": [
            "sh"
        ],
    },
    "architecture": "amd64",
    "os": "linux",
    "Size": 21
}
```


图 3 元信息文件部分内容

4 系统功能模块设计与实现

基于上文所提出的技术方法, 本文设计和实现了一种面向 Dockerfile 的容器镜像构建工具, 分为正向编辑和逆向分析两个主要功能模块, 正向编辑模块又包括可视化编辑 Dockerfile 和对常见 Dockerfile 错误进行检测两部分功能, 下面将从系统架构和功能实现详细进行介绍.

4.1 系统整体架构

本文所实现的系统中采用了 B/S 的三层体系结构: 表现层、业务逻辑层、数据层, 这样有利于系统的开发、维护、部署和扩展. 如图三所示, 其中表现层使用业务逻辑层、业务逻辑层使用数据层. 表现层主要包

括展现界面的 UI 展示类, 业务逻辑层包括实体类和服务类, 数据层包括映射类和数据控制类. 业务层依赖 WEB GUI 得以展现, 数据层依赖 Hibernet 技术. 三层结构都建立在 SSH framework 的基础之上.

系统整体架构如图 4 所示, 下面详细介绍每个部分的作用.

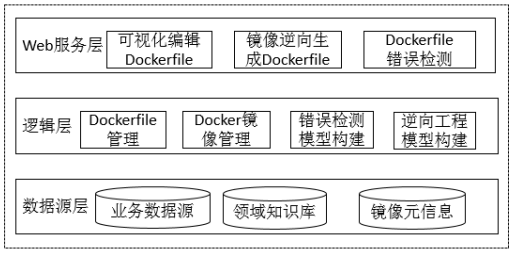

图 4　系统整体架构

Web 服务层包含了可视化编辑 Dockerfile、镜像逆向生成 Dockerfile 和 Dockerfile 错误检测三个主要功能模块.

逻辑层主要由四部分构成: Dockerfile 管理、Docker 镜像管理、错误检测模型构建和逆向工程模型构建. Dockerfile 管理包括 Dockerfile 的可视化创建语句, 合并相关语句以及删除语句等功能. Docker 镜像管理主要对本地存储的镜像文件进行管理以及可视化展示, 供用户选择进行编辑. 错误检测模型主要对 Dockerfile 中常见的错误进行检测并提出警告, 包括引用文件不存在. 多条 CMD 命令、设置环境变量与系统环境变量冲突以及 shell 命令与基础镜像不匹配等错误. 逆向工程模型提供逆向分析 Docker 镜像, 将镜像处理生成构建该镜像时所编写的 Dockerfile 后返回给用户.

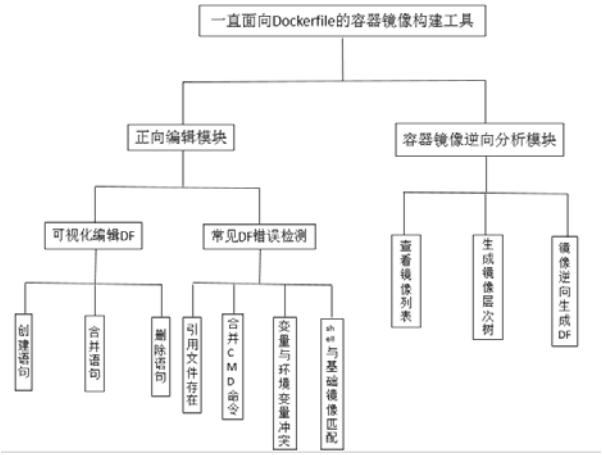

图 5　系统总体功能模块图

数据源层提供逻辑层数据的持久化机制, 其中业务数据源是业务系统的关系型数据存储, 镜像元信息存储本地 Docker 镜像的元信息文件, 领域知识库是将错误检测模型所需的领域知识持久化到数据库中.

系统总体功能模块如图 5 所示.

4.2 正向编辑模块

正向编辑模块的主要功能包括创建 Dockerfile 语句、合并相邻的 RUN 语句、删除语句等可视化编辑 Dockerfile 功能, 以及检查引用的文件是否存在、合并多条 CMD 命令、检测设置变量与系统环境变量是否冲突、检测 shell 命令与基础镜像是否匹配等 Dockerfile 编辑过程中常见的错误, 其编辑界面如图 6 所示.

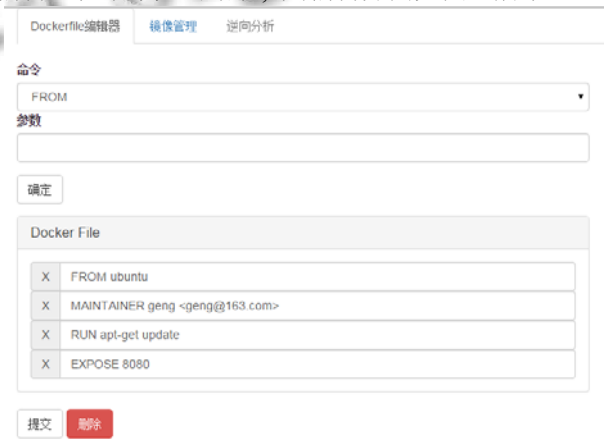

图 6　Dockerfile 编辑界面

创建 Dockerfile 语句: 用于新建 Dockerfile 命令语句, 创建语句时时, 主要有两个选项: 一是选择要创建的 Dockerfile 命令; 二是填写对应 Dockerfile 命令所需的语句参数.

合并相邻的 RUN 语句: 每条 RUN 指令将在当前镜像基础上执行指定命令, 并提交为新的镜像. 当命令较长时可以使用 \ 来换行. 因为多条 RUN 指令将导致构建多余的容器镜像层, 影响效率, 因此可以将多条 RUN 命令进行合并处理.

删除 Dockerfile 语句: 提供对编辑好的 Dockerfile 进行删除操作, 用户可以选择一条或多条 Dockerfile 语句进行删除, 删除完成后系统会自动更新 Dockerfile.

判断引用的文件是否存在: 在 Dockerfile 编写中一类常见的错误是引用的文件不存在, 因此系统会对 Dockerfile 语句所引用的文件在写入前进行检查, 以提

前发现 Dockerfile 引用不存在的文件错误.

合并多条 CMD 命令: 根据 Dockerfile 语法的规定, 每个 Dockerfile 只能有一条 CMD 命令. 如果指定了多条命令, 只有最后一条会被执行. 因此系统会在 Dockerfile 编辑完成时合并多条 CMD 指令, 以保证用户编写的 CMD 指令均可以有效运行.

检测设置变量与系统环境变量是否冲突: Dockerfile 中 ENV 命令用来设置环境变量. 然而在设置环境变量时用户并不完全清楚系统环境变量中是否已经存在相同的环境变量, 因此如果出现冲突将导致意想不到的后果, 因为本系统通过预先读取系统环境变量的方法, 当用户设置环境变量时会检查系统中是否已存在该变量, 与系统环境变量出现冲突时将会弹出警告说明, 提醒用户可能会导致该错误的发生.

Shell 命令与基础镜像不匹配检查: 在 Dockerfile 中 FROM 命令即为构建镜像引入基础镜像, 而在 RUM 指令是指在当前镜像基础上执行指定 shell 命令, 并提交为新的镜像, 因为用户的疏忽或者是对基础镜像的 shell 命令不熟悉的缘故, 导致使用了与基础镜像不匹配的 shell 命令, 构建镜像会因此而失败. 因此本系统采取建立知识库的方法, 对常用基础镜像及其支持的 shell 命令建立知识库, 在使用 shell 命令时会在知识库中检查是否与所属的基础镜像匹配, 以提前检测到错误的发生. 本文以 CentOS7.0 为例, 统计了 shell 命令所存在的/usr/bin, /usr/sbin 和/sbin 几个文件目录下可执行的文件个数, 基本的 shell 命令约 1200 多条. 分别针对常用的 RedHat, Debian, Slackware 等三大类 Linux 发行版构建知识库, 知识库约 5000 行数据, 每行必需字段为 136 字节, 总大小约 68MB, 对知识库构建索引, 每行索引的字节数为 4 字节, 总大小为 2MB. 此外, 用户可以根据自己的需求, 动态更新系统的 shell 命令知识库. 因此, 对于特定 Linux 系统而言, 使用该 Docker 镜像构建工具所需要构建的知识库容量很小, 不会带来系统性能上的瓶颈.

4.3 容器镜像逆向分析模块

容器镜像逆向分析模块负责解析 Docker 镜像存储结构、层级依赖关系和元信息文件, 由 Docker 镜像逆向生成该镜像所对应的 Dockerfile 文件, 基于该文件提供给用户作为了解该镜像所包含的功能、使用方式和生成信息的参考, 以及一定程度上通过此 Dockerfile 文件用户可以自行选择安全可靠的文件再次生成具有同样功能的 Docker 镜像, 以此来解决第三方镜像可能包含恶意文件所带来的.

容器镜像逆向分析模块具体功能如下: 查看镜像列表, 可以查看本地保存的所有镜像, 以列表形式呈现; 生成镜像层次树, 展示该镜像的层次信息和依赖关系; 镜像逆向生成 Dockerfile, 通过分析镜像存储结构、层次依赖关系和每层镜像的元信息文件, 由镜像逆向生成 Dockerfile 文件. Docker 镜像逆向分析的步骤如下:

① 获取镜像的层次结构信息: 通过分析每层镜像的元信息文件, 从中分析获取该层镜像的父镜像信息, 建立镜像层之间的依赖关系, 生成镜像层级树.

② 解析元信息文件: 对镜像的元信息文件进行解析, 从中获取构建该层镜像时所用到的 Dockerfile 指令, 该指令存储在 json 文件中的 Cmd 键值对中. 对每一层镜像执行相同的操作.

③ 通过上一步中解析到的构建单层镜所使用的 Dockerfile 指令结合第一步中所建立的镜像层级关系树所包含的依赖关系, 调整 Dockerfile 指令顺序, 恢复构建镜像所使用的 Dockerfile.

算法 1 镜像逆向生成 Dockerfile 算法

```
Module Image-Reverse;
Input: Docker Image;
Output: Dockerfile;
//检测该层镜像是否存在父镜像层
while image-layer has parrent-image:
    get layer json; //获取镜像层元信息
    extract cmd value;//获取键为 cmd 的值
    extract build instruction;//抽取构建镜像命令
    write instr in Dockerfile;//指令写入 Dockerfile
```

5 案例分析

本节通过一个实际案例说明面向 Dockerfile 的容器镜像构建工具的功能使用.

5.1 Dockerfile 编辑

使用该工具编辑一段 Dockerfile 文件, 内容如下:

```
FROM ubuntu
MAINTAINER geng <geng@163.com>
RUN apt-get update
EXPOSE 8080
```

该工具通过可视化编辑的方式提供语法规范, 用户只需要选择所需的命令, 工具会根据不同命令提供

相应的语法格式，用户只需要填写相关参数即可，如下图所示.

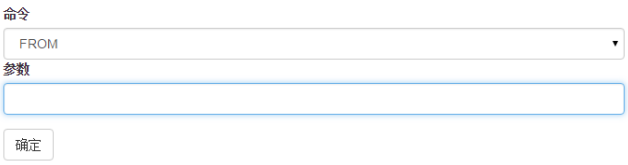

图 7　Dockerfile 可视化编辑界面 1

填写完参数点击确定，命令会被添加到下面的文本框中，如图 8 所示，用户还可以对文本框中已经添加的命令进行修改、删除等操作. 在编辑过程中，针对前文中所描述的错误类型信息，填写错误的参数，工具会弹出警告框，提示会出现相应的错误信息来提醒用户. 完整的 Dockerfile 编写完毕之后，点击提交，完整的结果会被保存在文件中，通过运行 docker build 命令即可使用编辑好的 Dockerfile 构建出新的镜像. 在系统中运行 docker images 命令查看构建好的镜像.

图 8　Dockerfile 可视化编辑界面 2

5.2 Docker 镜像逆向分析

Docker 镜像逆向分析的主要功能是通过镜像信息还原出构建镜像的 Dockerfile 文件，选择上文构建生成的新镜像进行逆向分析，可以还原出构建该镜像所编写的 Dockerfile 文件，如图 9 所示，与编写过程中的 Dockerfile 一致. 因此可以验证逆向分析功能的正确性.

```
1 FROM ubuntu
2 MAINTAINER geng <geng@163.com>
3 RUN apt-get update
4 EXPOSE 8080
5
6
7
8
9
```

图 9　Docker 镜像逆向生成 Dockerfile

5.3 实验结果与分析

针对上文描述的使用 Dockerfile 构建容器镜像过程中常见的语义问题，以及 Docker Hub 上第三方镜像使用中存在的问题，使用该工具进行了实验验证，得出如下结果和分析，如图 10 所示.

	Dockerfile常见错误检测				Docker镜像逆向工程
问题说明	引用的文件不存在	CMD命令仅一条有效	Shell命令与基础镜像不匹配	设置变量与系统变量冲突	第三方镜像功能不明确，以及可能包含恶意文件代码
实验结果	对引用文件不存在问题可以完全解决	完全解决多条CMD命令问题	仅对知识库中所包含的基础镜像shell命令匹配问题有效	有效解决环境变量冲突问题	可以有效解决第三方镜像功能不明确的问题，一定程度上解决第三方镜像包含恶意文件的问题
结果分析	采用预先验证引用文件存在性方案可以避免该问题出现	合并多条CMD命令方案可以使得多条CMD命令均有效	shell命令需要与知识库中的基础镜像匹配，因此知识库中未包含的基础镜像难以检测	系统部署时自动读取系统环境变量，因此对于不同系统环境均可以解决环境变量冲突问题	对于通过Dockerfile构建的第三方镜像可逆向生成原始的Dockerfile文件，用户根据Dockerfile文件明确镜像功能和使用方法，利用该Dockerfile添加安全可信的文件，二次构建新镜像，可以一定程度上解决第三方镜像安全性问题
使用说明	无	无	目前仅支持几种常用基础镜像，用户可针对特定镜像构建知识库	不同系统环境使用场景下需重新部署以重新获取系统的环境变量	Docker 镜像安全性问题有很多种，使用该方案仅能解决基本的包含恶意文件的问题，对于其他攻击方式需要其他更加复杂智能的方法，是目前Docker研究的一个热点

图 10　实验结果分析与使用说明

6　总结与展望

本文的面向 Dockerfile 的容器镜像构建工具考虑到构建 Docker 镜像过程中所遇到的困难[7]，提出了有效的语法引导以及常见错误类型检测，在编写 Dockerfile 时通过可视化引导的方式，提供确定的语法格式和提示说明以及正确的指令拼写，在很大程度上避免了语法层面上的错误. 此外根据统计分析目前 Dockerfile 编写过程中常见的非语法错误，系统针对不同的错误提供了具体的解决方案，一定程度上降低了 Dockerfile 编写的出错率. 另一方面，针对第三方镜像功能说明不完整以及安全上的漏洞，系统提出了逆向工程的方式通过镜像还原出 Dockerfile，用户可以通过 Dockerfile 完全了解镜像的功能并可以通过二次构建的方式，一定范围内避免出现第三方镜像的安全问题.

随着平台即服务以及运维自动化工具的普及，Docker 系统的应用和开发将面临更大的需求和挑战. 而作为 Docker 系统核心之一的 Dockerfile，针对不同的业务场景也面临着更复杂的功能需求，包括镜像管

理, Dockerfile 复用, 一键构建镜像等等. 在下一步的工作中, 面向 Dockerfile 的容器镜像构建工具将进一步完善功能、美化界面、优化性能, 为构建容器镜像提供更好的支持.

参考文献

1 http://eclipse.org/mars/.

2 http://domeide.github.io/.

3 Dong Z, Andrzejak A, Shao K. Practical and accurate pinpointing of configuration errors using static analysis. 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE. 2015. 171–180.

4 陈伟,黄翔,乔晓强,魏峻,钟华.软件配置错误诊断与修复技术研究.软件学报,2015,26(6):1285–1305.

5 岑义涛.从 Docker 容器漏洞谈 Docker 安全.网络世界,2014 年.

6 Zhang Z, Zhang J. Characterizing failure-causing parameter interactions by adaptive testing. Proc. of the 2011 International Symposium on Software Testing and Analysis. ACM. 2011. 331–341.

7 Merkel D. Docker: Lightweight linux containers for consistent development and deployment. Linux Journal, 2014, (239): 2.

8 Xia X, Lo D, Qiu W, et al. Automated configuration bug report prediction using text mining. 2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC). IEEE. 2014. 107–116.

9 Zhang J, Renganarayana L, Zhang X, et al. EnCore: Exploiting system environment and correlation information for misconfiguration detection. ACM SIGPLAN Notices, 2014, 49(4): 687–700.