TiD 2017
质量竞争力大会

利用Docker生态开源软件构建容器化测试平台

孙远 华为中央软件院

软件研发
SOFTWARE DEVELOPMENT

我的经历

美国风河系统公司

- 负责linux build system、analysis tools、workbench测试工作。

华为中央软件院

- 带领团队进行容器OS、docker测试工作，并参与项目过程改进。
- 参与开源社区：补充ltp社区user namespace特性测试用例和docker社区中的测试用例。
- 完成《docker进阶与实战》测试章节的编写工作。

内容

- 什么是开源软件
- 选取开源软件的标准
- Devops与容器技术所涉及的开源软件
- Devops及容器云产业状况
- 工程能力容器化沙盘
- 测试工具容器化
- 源码编译容器化
- 应用层软件测试容器化
- 移动终端测试容器化
- 中间层软件测试容器化
- 内核测试容器化
- 硬件驱动测试容器化
- 长稳测试容器化
- 在开源社区的测试中成长
- 开源软件问题求助方式与注意事项
- 参与开源社区的策略

华为中软院业务欧拉部业务介绍

- 服务器操作系统
- 实时操作系统
- 编译器
- 终端操作系统
- Linux内核
- 容器技术

Linux

什么是开源软件?

开源软件是一种源代码免费向公众开放的软件，任何团体或个人都可以在其License的规定下对其进行使用、复制、传播及修改，并可以将该修改形成的软件的衍生版本再发布。

—来源于 OSI开源软件定义

特点：享有版权、特定的License、源码开放、无许可费、可自由使用

参与开源的收益

•对于公司：加快产品上市周期、降低软件维护成本、保障质量、了解业界友商动态

•对于员工：问题求助的渠道、提升技能、开阔视野、积累人脉拓宽发展通道

License

•BSD类：允许随意商业集成

•MPL类：修改后的源代码需公开

•GPL类：传染性/商业不友好

TiD2017

选取开源软件的标准

标准	描述
开源协议	谨慎使用采用GPL协议的开源软件。
软件是否满足业务（包括功能、系统架构、稳定性、扩展性、性能、兼容性、安全性）	略
contributors、commits、Pull Requests、Issues、Fork数值	更多的数值代表着有更多的社区活跃度，有利于社区向良性发展。
完善的文档	丰富的文档、wiki页面、指导书将更好的引导我们理解社区所涉及的项目细节和使用方法。
项目开源贡献者的分布情况	有多个公司参与的开源项目更加可靠，避免项目被某一个公司绑架。
市场占用率及认可程度、成熟度	成熟度高的开源项目软件缺陷少、认可度高，便于商业推广。
项目是否加入开源基金会	加入开源基金会的项目不会被某一团体绑架或者被竞争对手收购。
开源社区接纳新特性的能力	社区接纳贡献者提交新特性的可能性越高，越有利于引导社区向适合公司战略的方向发展。
对于提出问题或软件缺陷响应的速度	响应速度越快，越有利于用户问题得到解决。
是否有成熟的测试用例和验证方案	这有助于更好的保证软件质量。
社区是否有大公司支持，maintainer、committer、contributor区分是否清晰。	有大公司商业支持的项目，服务更加优质，同时也具备进一步的商业合作。

Devops与容器技术所涉及的开源软件

类别	开源软件
安全容器	hyper,clear container,photon…
镜像存储	docker-registry…
容器引擎	docker, rkt…
安全	selinux, seccomp, notary…
网络	ipsec,haproxy…
存储	ceph,swift…
数据库	mysql,mongoDB…
编排工具	mesos,kubernetes,swarm,compose,marathon…
虚拟化平台及工具	docker machine,virtualbox, openstack…
容器操作系统	rancheros,coreos,vmware photon,snappy ubuntu core,redhat atomic host…
开放容器项目标准	runc,image-spec,image-tools,runtime-spec,runtime-tools…
支持容器运行的操作系统特性	kernel

Devops及容器云产业状况

分类	描述	现状
容器商用公有云平台	亚马逊云、docker云、XX云…	大小企业并存，同质化严重，且竞争激烈，属于群雄逐鹿阶段。企业需要拥有充足资金、核心技术和对生态准确的把握，才能在市场中立足。市场将最终形成寡头。
其它外围工具	安全、存储、监控等	中小企业的机会点
编排工具	mesos,kubernetes,swarm	有实力的公司在角逐，三分天下的局面
容器引擎	docker， rkt…	寡头，技术门槛相对不高，但需要完善生态
容器OS	rancheros,coreos,vmware photon,snappy ubuntu core,redhat atomic host…	寡头，有实力的公司参与
OCI标准项目	runc,image-spec,runtime-spec	标准制定的战场，大公司博弈的战场
底层内核	kernel	技术门槛高，大公司博弈的战场

Goal of Docker

Goal: Build, Ship, and Run Any App, Anywhere.

Build	Compose your application from microservices, without worrying about inconsistencies between development and production environment.
Ship	Store and distribute your docker images in registry services.
Run	Deploy scalable services, securely and reliably on a wide variety of platforms.
Any App	
Anywhere	

Docker与Windows、ARM64

•Docker可以运行在Windows Server 2016和Windows 10。

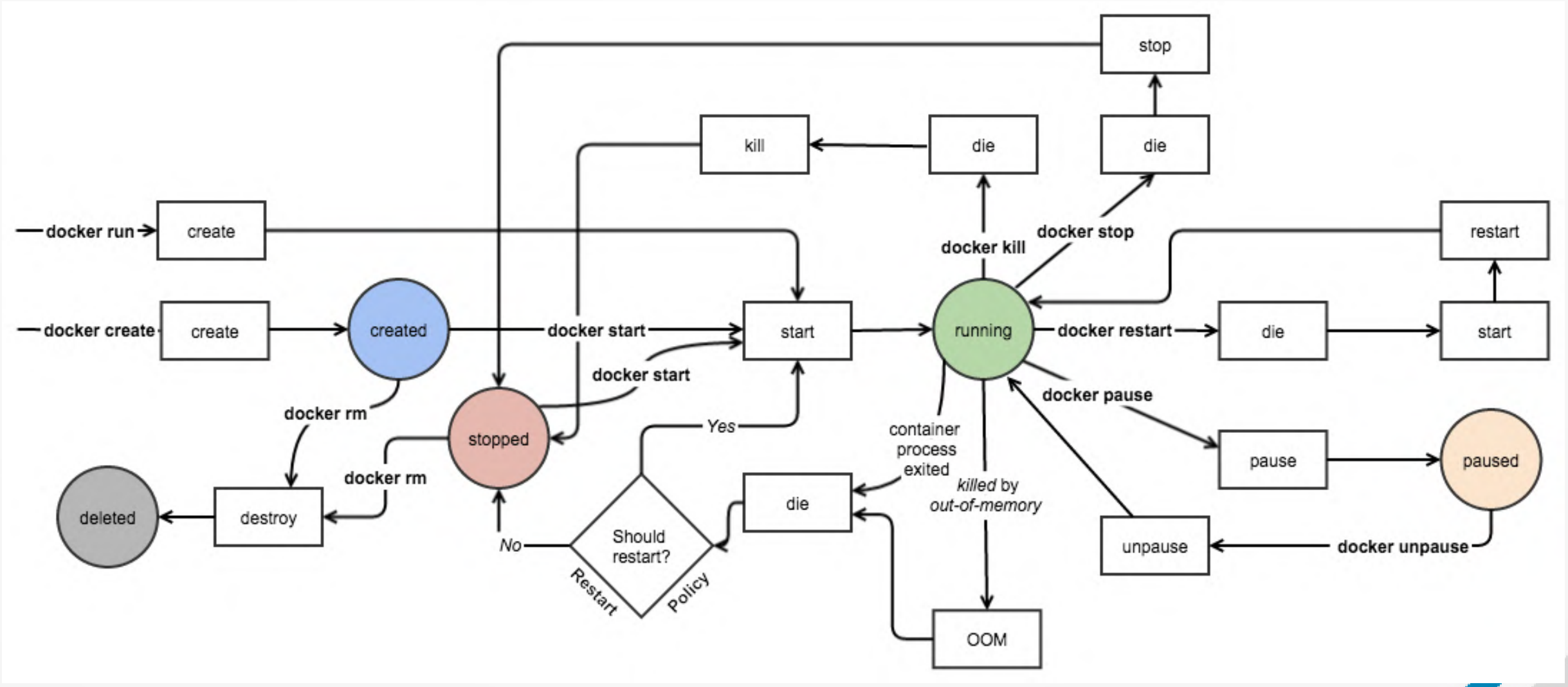
注:除非使用硬件虚拟化技术，Windows与Linux中的Docker镜像无法相互使用。

•Docker已经可以在ARM64中平稳的运行。

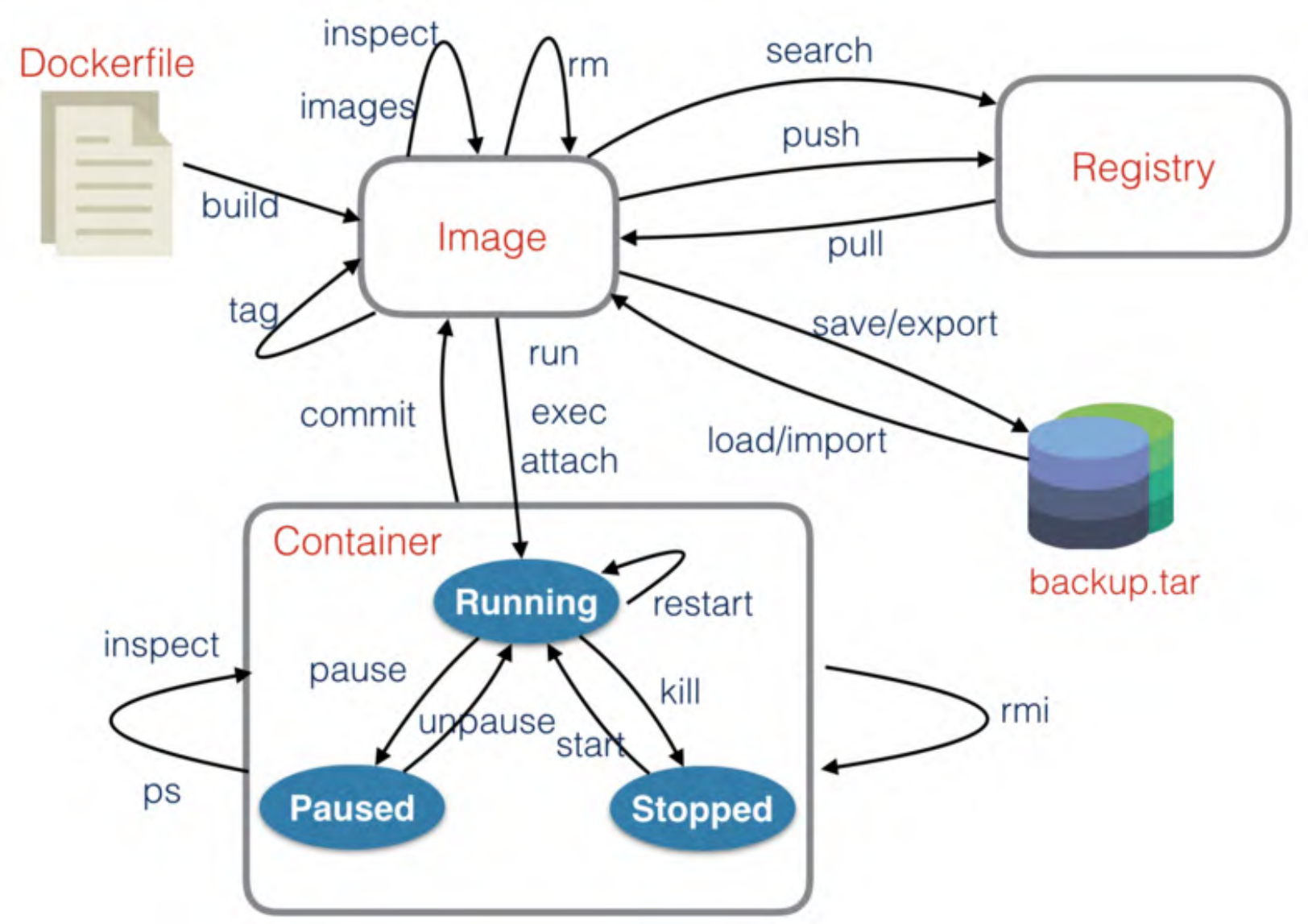
注:Arm64中的Docker无法启动x86的Docker镜像。

TiD2017

Docker基础知识—容器状态



Docker基础知识



容器、镜像、仓库交互关系

Docker基础知识—资源管理

- Memory

1. --memory
2. --memory-swap
3. --memory-reservation
4. --memory-swappiness

- CPU

1. --cpu-shares
2. --cpu-period
3. --cpu-quota
4. --cpuset-cpus
5. --cpuset-mems

- IO

1. --blkio-weight
2. --blkio-weight-device
3. --device-read-bps
4. --device-write-bps
5. --device-read-iops
6. --device-write-iops

Docker基础知识

- 数据卷

-v, --volumes-from

- 容器网络

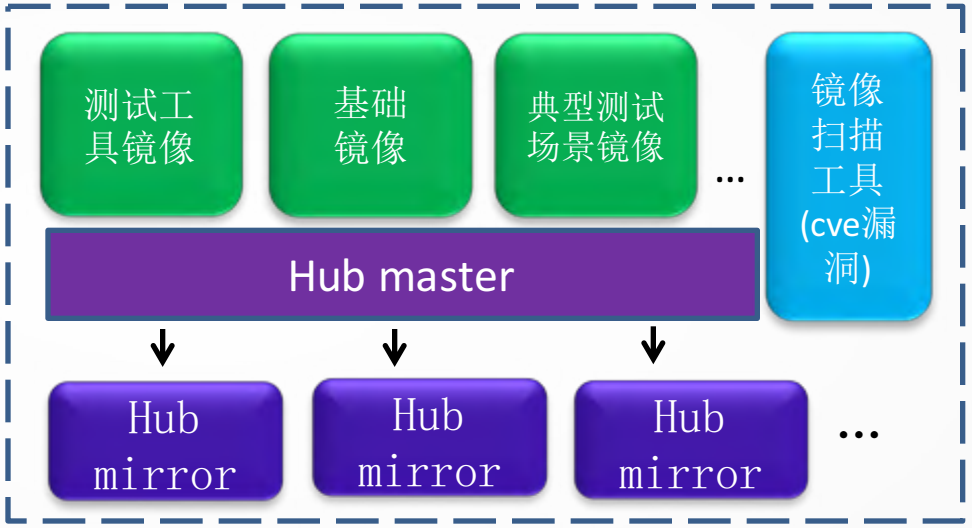
--net=bridge,none,container:name_or_id,host

- 容器仓库

search, pull, push

${hub_address}/${namespace}/${image/tag}

构建测试专用镜像仓库

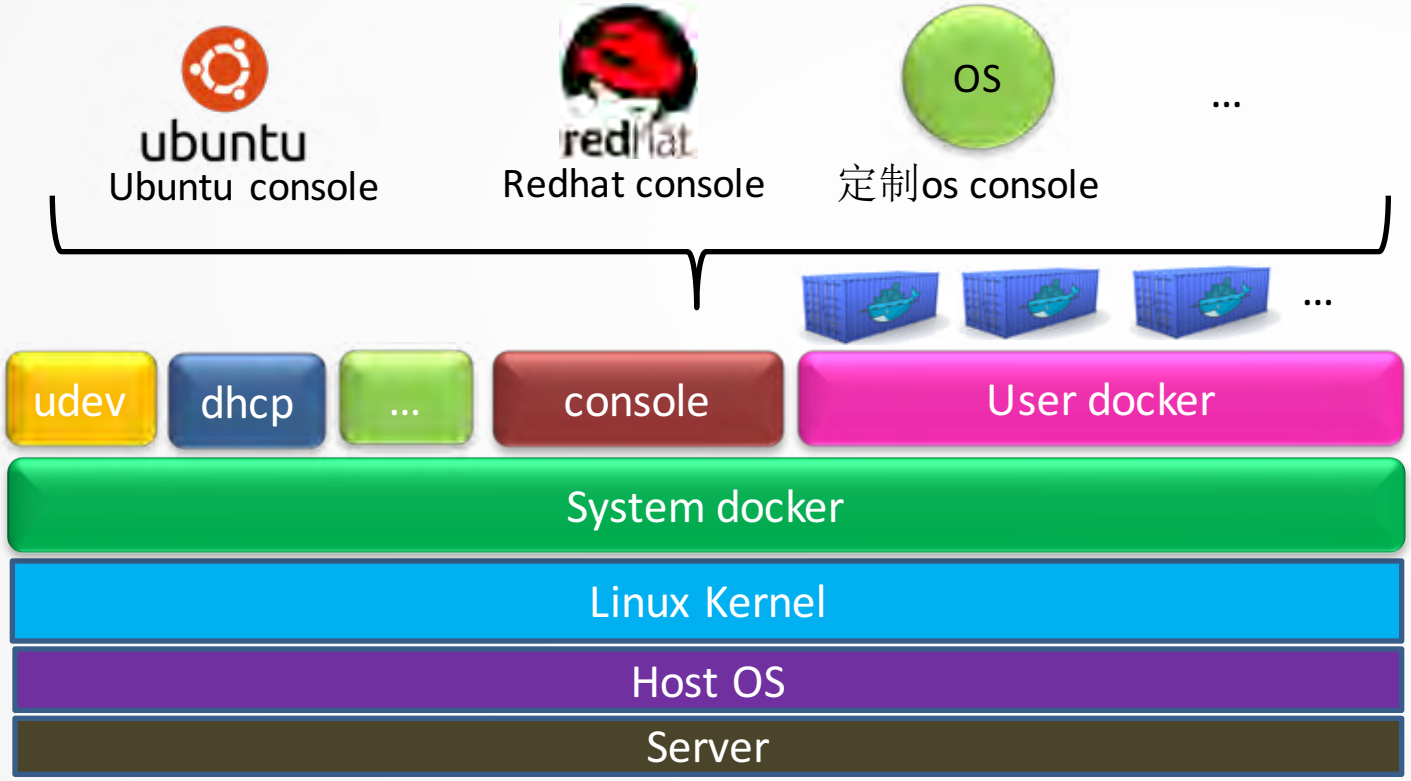


https://github.com/docker/distribution 改造distribution开源软件

不足的方面需要进行改造
- 缺少多租户鉴权
- 后端存储驱动不全
- 缺少安全加固
- 无法支持镜像站点

改造
- 添加多租户鉴权
- 补充后端存储
- 添加镜像扫描特性，确保镜像安全。
- 添加镜像站点

基于开源软件RancherOS的改造



File system and tools	**8.4MB**
User docker (upstream docker)	13.7MB
System docker	4MB
Kernel drivers	11.7MB
Kernel core	4.2MB

- 资源占用少
- 便于系统升级回滚
- 可以自主定制和更换os console

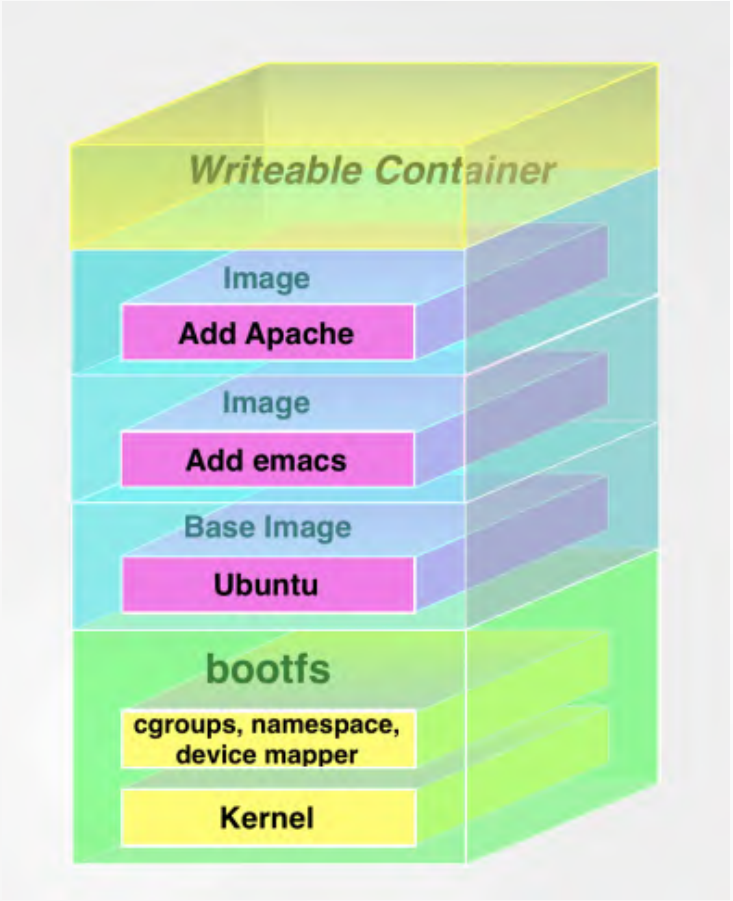
不足的方面需要进行改造

- 代理能力不足
- 启动需要网络支持，无网络情况下无法启动
- 缺少安全加固

改造

- 适配到公司内部网络
- 将默认console镜像内置，使其启动不依赖于网络
- 添加安全加固特性

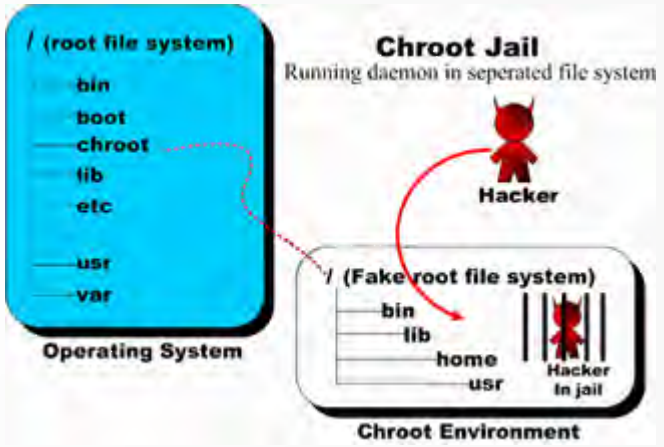
Docker镜像

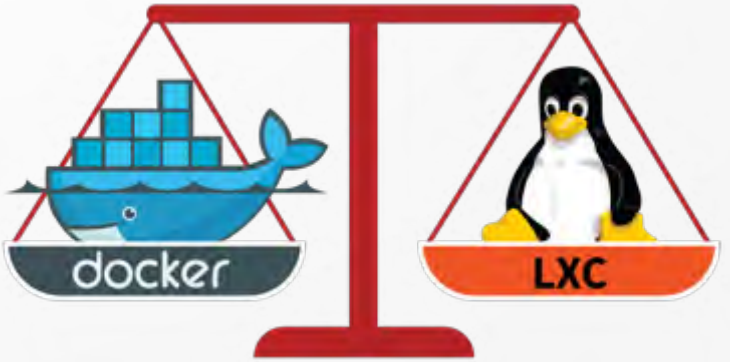


Apache
NGINX http server
Apache Tomcat
LAMP
GitLab
MySQL
mongoDB
redis
hadoop
APACHE Spark
Java
python
golang
php
Perl
Jenkins

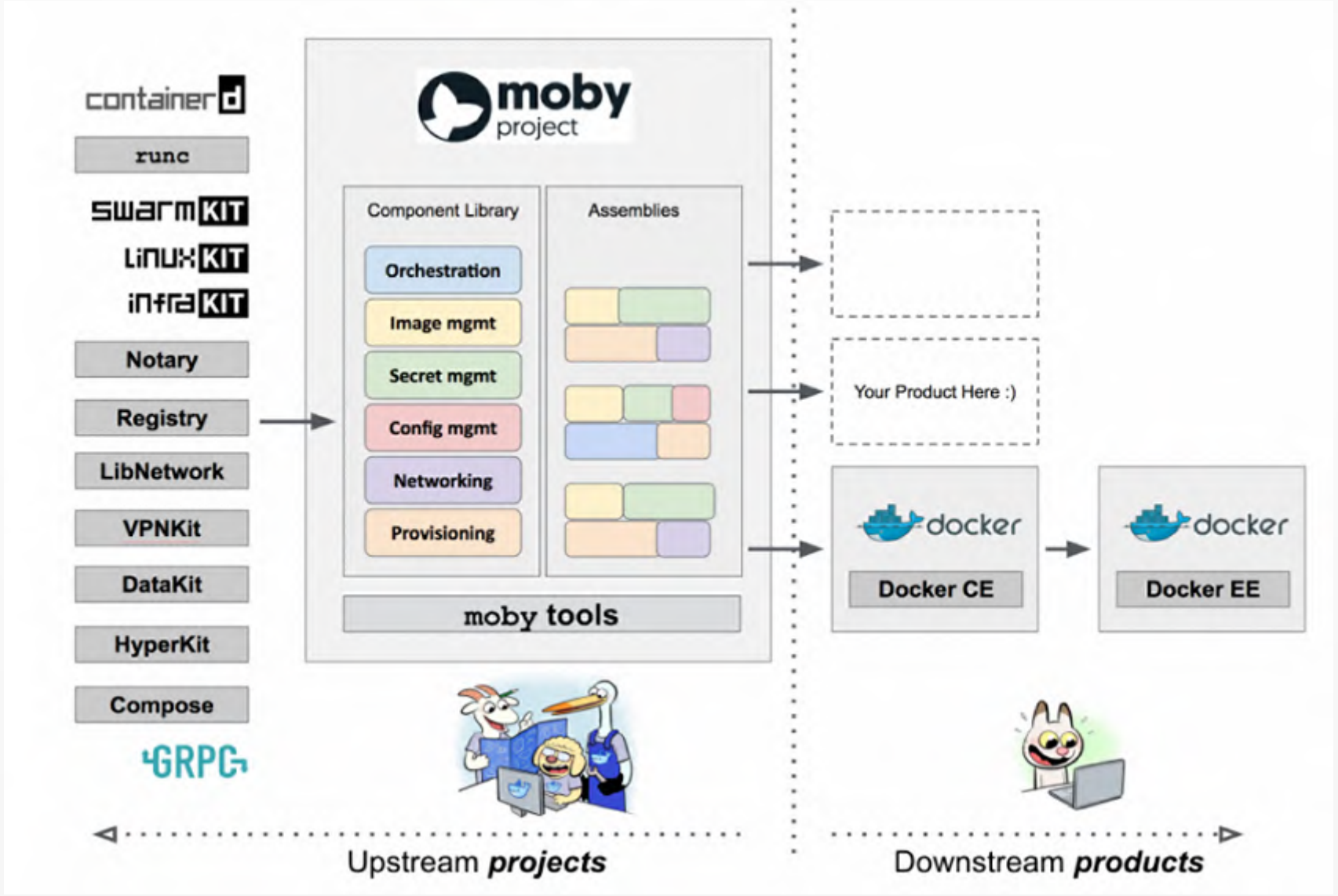
Docker成功的原因

- 标准的开发环境、更快的交付和部署。
- 内核级虚拟化、更高效的资源利用。
- 快速的迁移、部署和弹性伸缩。
- 利用Dockerfile可进行简易的配置修改。
- 优质的生态工具。



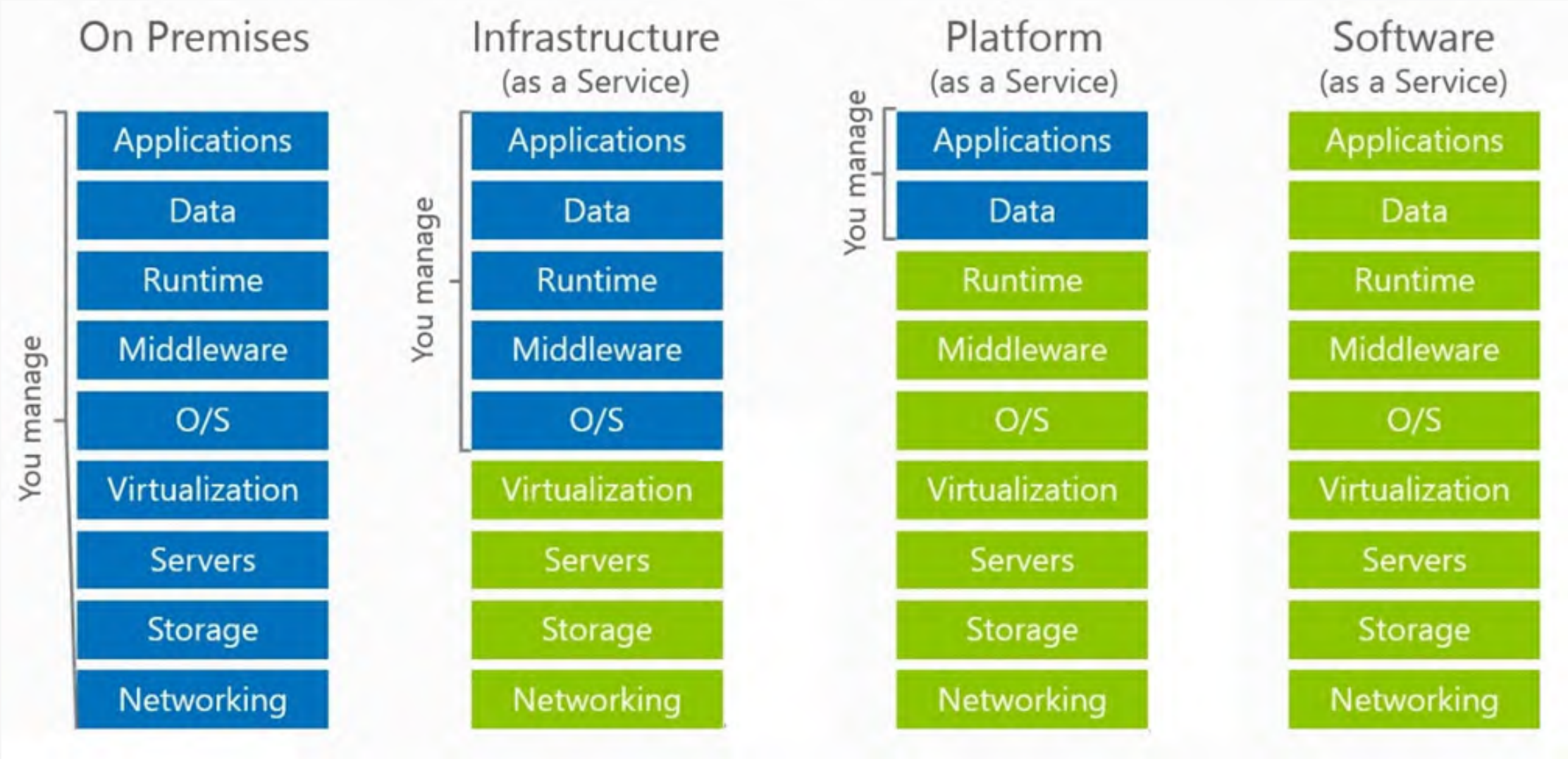


Moby与Docker



TiD2017

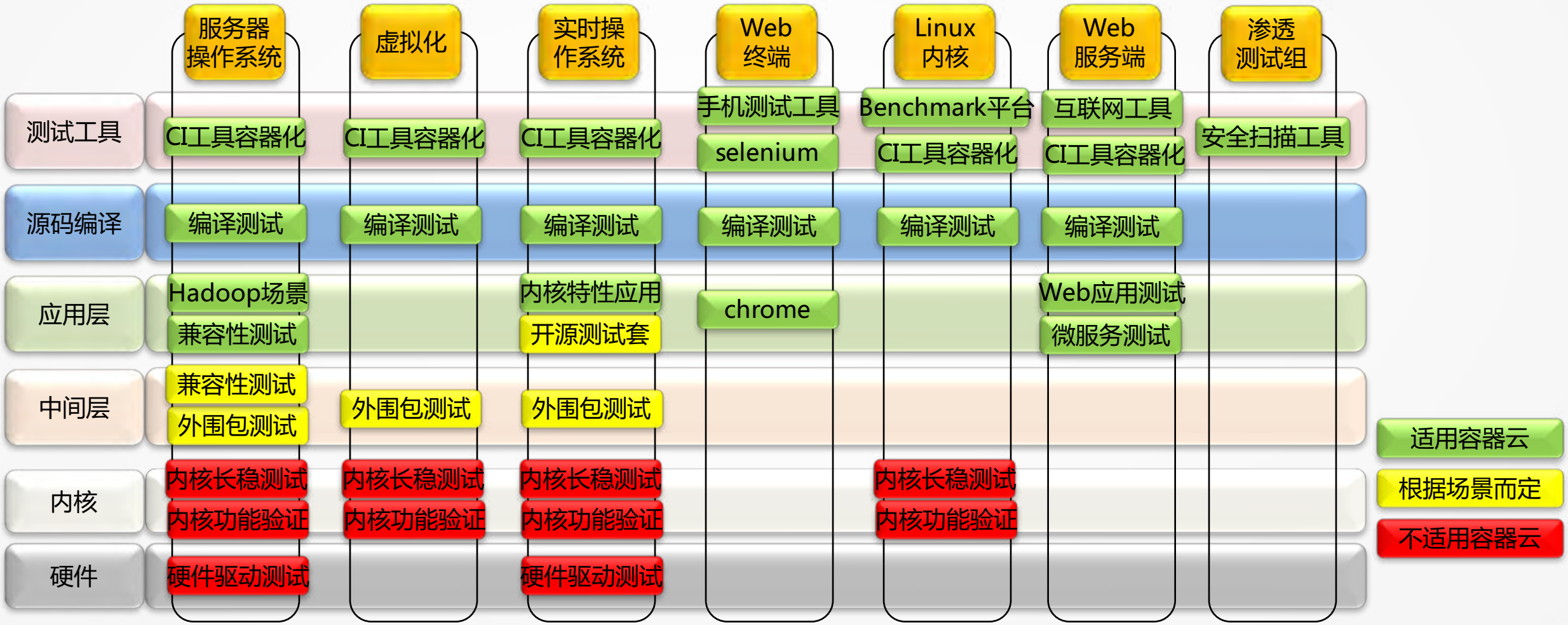
云模型


On Premises
You manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Infrastructure
(as a Service)
You manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Platform
(as a Service)
You manage
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Software
(as a Service)
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking

容器云

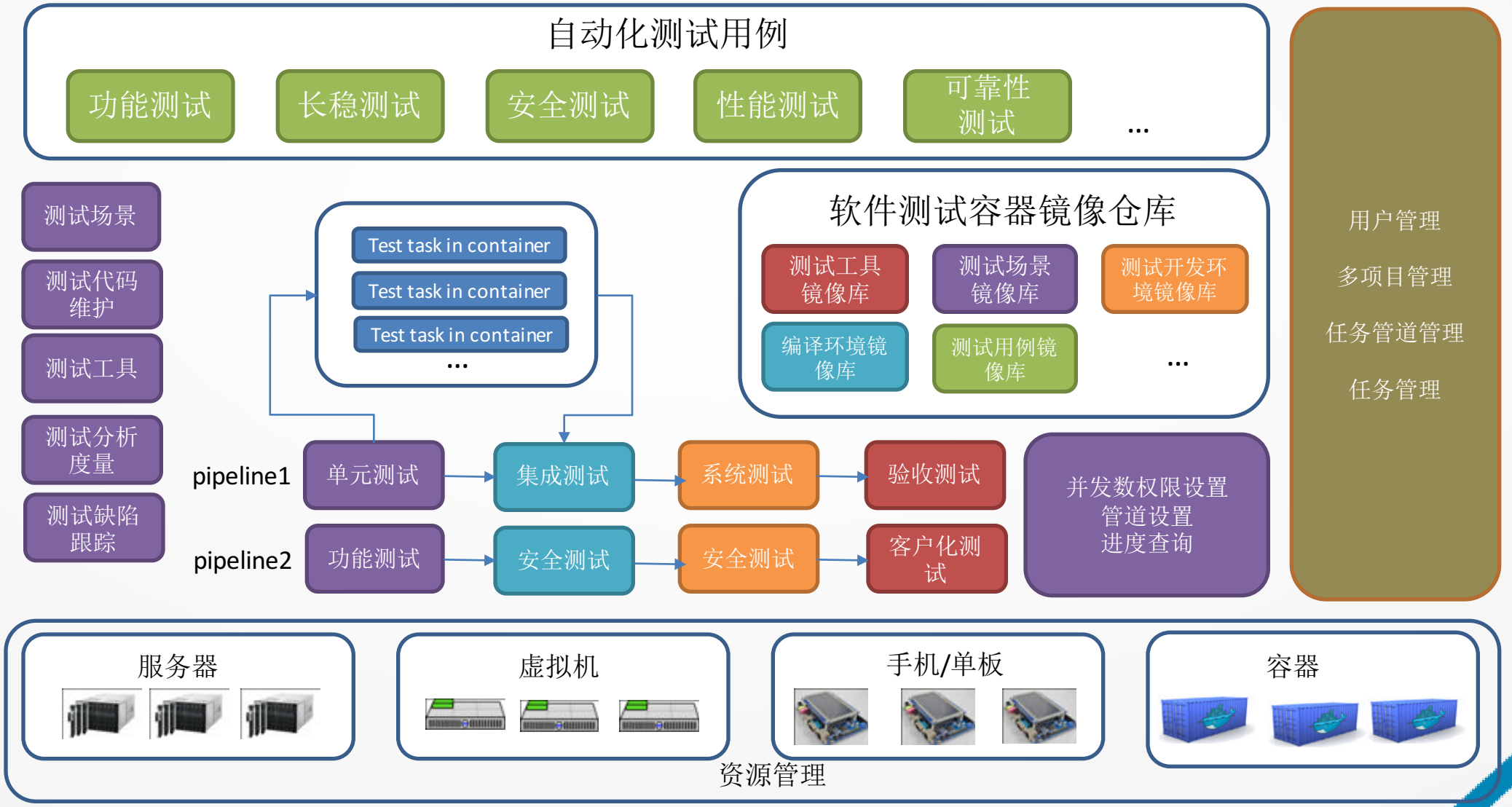
- IAAS (Infrastructure as a Service)
- PAAS (Platform as a Service)
- SAAS (Software as a Service)
- CAAS (Container as a Service)
- TAAS (Test as a Service)

工程能力容器化沙盘

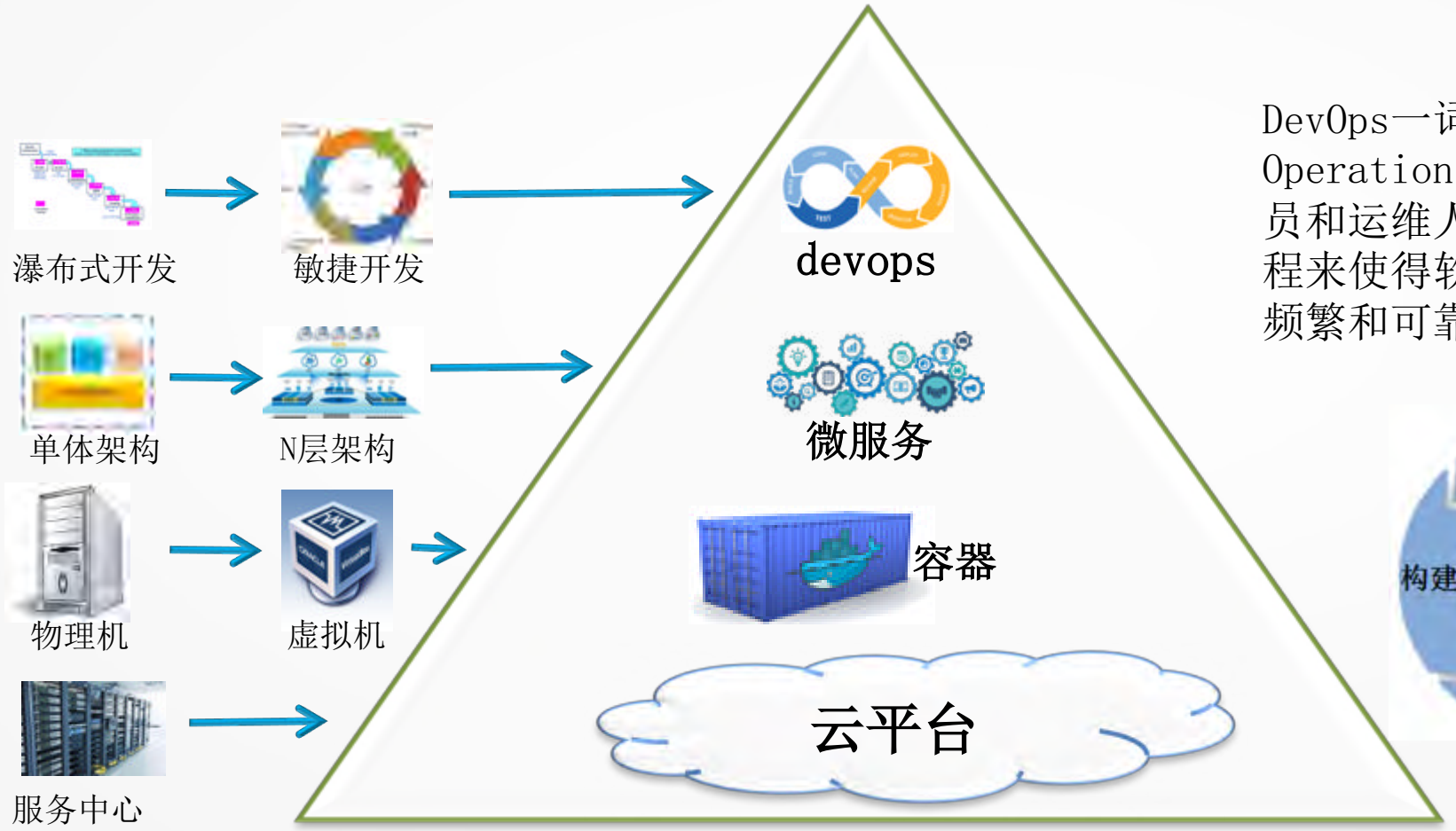


备注：容器云意味着其配置有一致的内核，相近的硬件资源。容器云和容器是两个概念。不适用容器云的业务并不代表不能使用容器进行业务改造。

测试平台


自动化测试用例
功能测试
长稳测试
安全测试
性能测试
可靠性测试
...
测试场景
测试代码维护
测试工具
测试分析度量
测试缺陷跟踪
Test task in container
Test task in container
Test task in container
...
软件测试容器镜像仓库
测试工具镜像库
测试场景镜像库
测试开发环境镜像库
编译环境镜像库
测试用例镜像库
...
用户管理
多项目管理
任务管道管理
任务管理
pipeline1
单元测试
集成测试
系统测试
验收测试
pipeline2
功能测试
安全测试
安全测试
客户化测试
并发数权限设置
管道设置
进度查询
服务器
虚拟机
手机/单板
容器
资源管理

Devops简介



DevOps一词的来自于Development和Operations的组合，突出重视软件开发人员和运维人员的沟通合作，通过自动化流程来使得软件构建、测试、发布更加快捷、频繁和可靠。

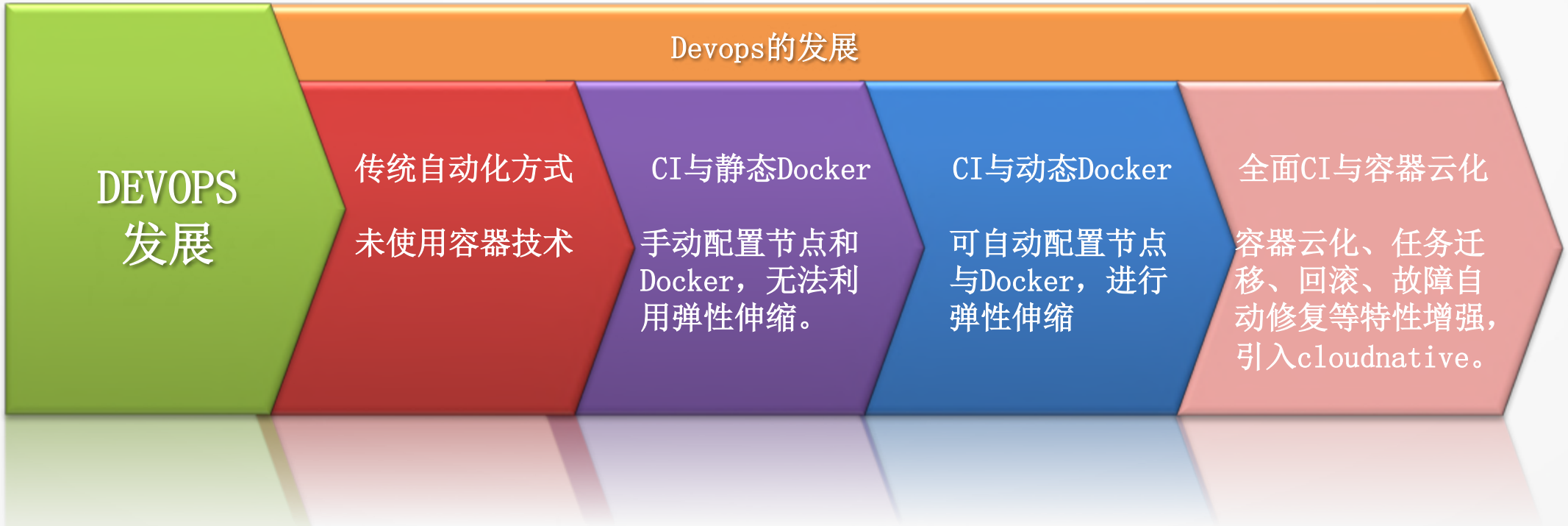
Devops

痛点

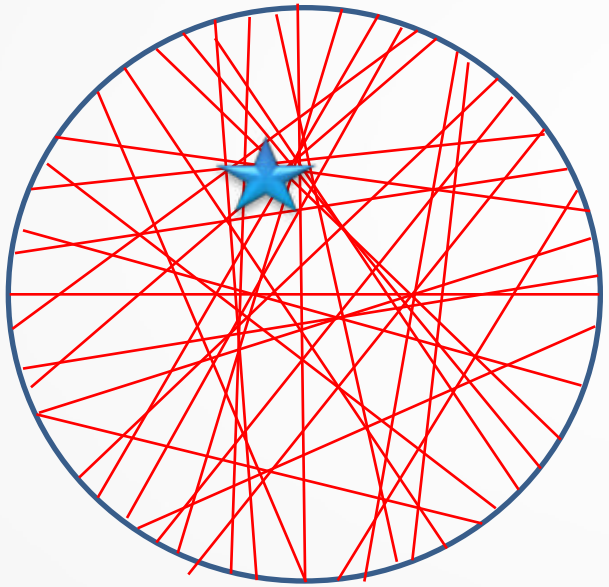
- 开发：无法复现测试提交的缺陷单、测试周期长、无法有效及时获取运维环境和数据。
- 测试：开发常常驳回问题单、开发需要不明不易理解、难以了解生产环境。
- 运维：开发提供的交付件不稳定、迭代速度快无法保障系统稳定性。

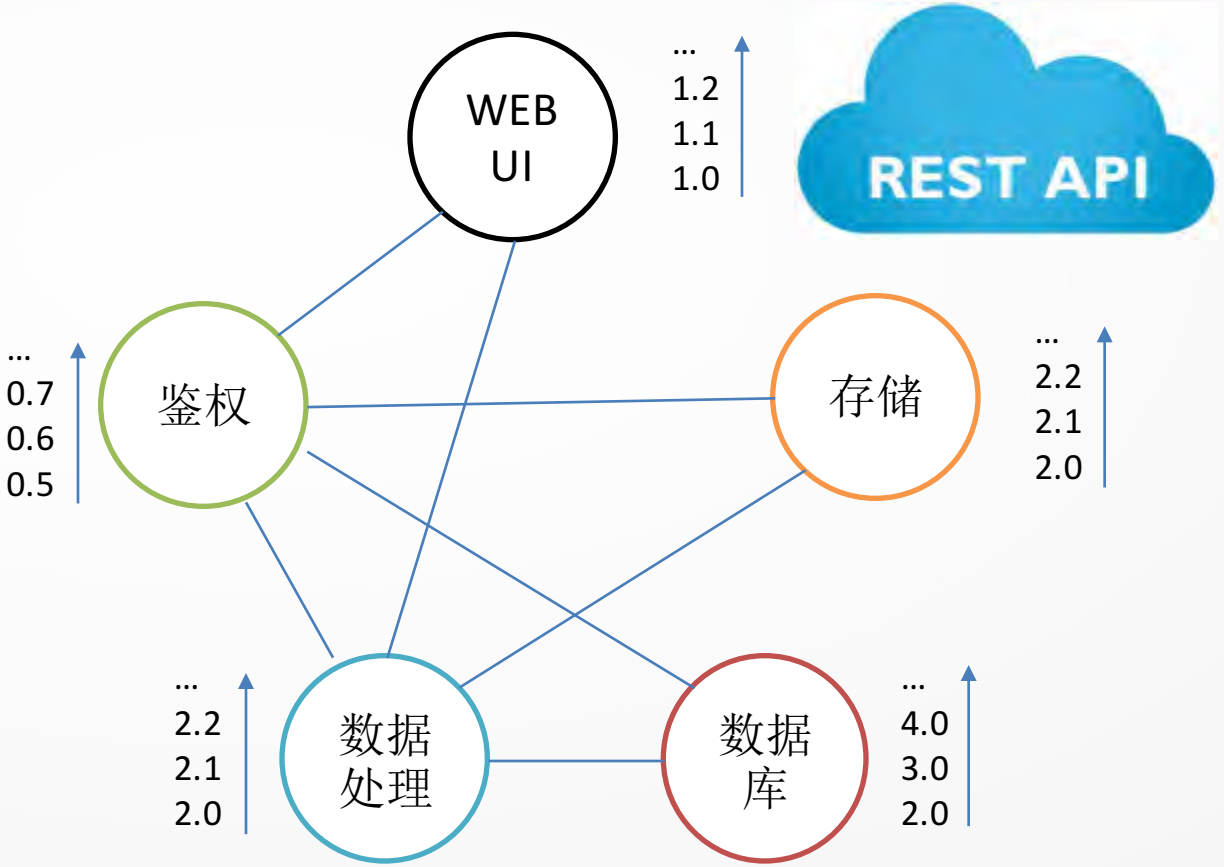
Devops

Devops发展的里程碑



测试场景微服务化



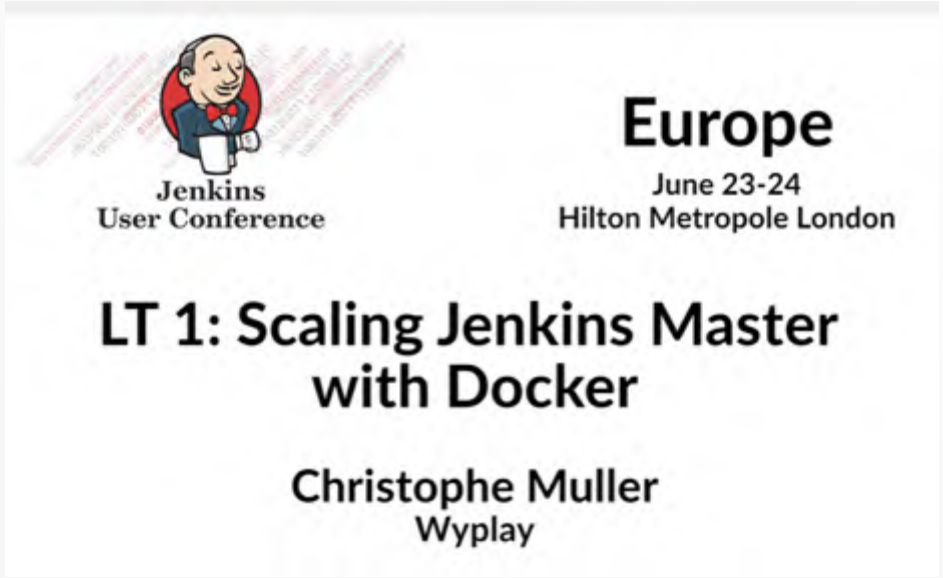

WEB
UI
...
1.2
1.1
1.0
REST API
...
0.7
0.6
0.5
鉴权
存储
...
2.2
2.1
2.0
...
2.2
2.1
2.0
数据
处理
数据
库
...
4.0
3.0
2.0

容器化改造主要量化KPI

- 测试环境搭建时间、场景构建时间减少值
- 测试执行时间降低值
- 测试工具部署效率提升值
- 源码编译速度提升值
- 资源利用率提升值
- 节约物料成本
- 减少测试设计时间
- 软件缺陷复现概率

测试工具容器化

- Docker能将测试工具放在一个干净的容器环境中，并且不会影响到主机文件系统。
- Docker镜像可以赋予系统管理员跨Windows、Mac和Linux服务器的简化管理能力，当服务端为Linux服务器时，客户端可以是Windows、Mac或Linux架构，这样会实现更快速的部署和管理。



Official Kali Linux Docker Images Released

测试工具容器化

- 提升测试工具可移植性，屏蔽不同Linux发行版。

测试工具部署

•主机中（非被测机）
可以部署一些CI工具、辅助测试工具（如发包工具）
可选用较新的Linux发行版作为主机，通常支持Docker。

•被测主机中
可以适配需要运行在被测主机中的测试工具。（如fio压力测试工具）
一些系统（特别是嵌入式系统）经过裁剪后文件系统较小，不一定默认包含Docker。需要在前期的需求中明确要包含Docker以保证可测试性。

软件测试执行加速方式

- 改进算法，优化测试用例。
- 选择功能强大主机，在主机中并行执行测试用例。
- 采用分布式的方式将测试任务下发到不同的主机中并行执行测试用例。

源码编译容器化

- 利用Docker快速部署、快速伸缩的特点，在短时间内获取容器云中大量系统资源支撑测试执行以达到测试加速的目的。在测试执行完之后，可快速释放所占资源，减少测试成本。
- 利用Dockerfile梳理编译环境配置步骤，使得环境准备的步骤更加清晰。
- 不必为每一个发行版的编译环境逐一配置虚拟机，只需配置好编译镜像即可，可将各业务组的编译资源集中使用，可以达到提高主机利用率和节约成本的目的。

Dockerfile:

```
FROM ubuntu:14.04
MAINTAINER Yuan Sun<sunyuan3@huawei.com>

# copy apt-get sources list to image
COPY sources.list /etc/apt/

# keep ubuntu up-to-date
RUN apt-get update && apt-get -qqy upgrade

# install tools for compile
RUN apt-get update && apt-get install -y \
    build-essential \
    automake \
    libtool \
    git
```

源码编译容器化

痛点：资源利用率不高

各项目组单独配置昂贵、性能好的主机进行源码编译。虽然编译速度有很大提升，但往往在没有转测试的时候，编译主机经常长时间空闲。很多时候空闲甚至达到90%以上，主机无法“吃饱”。

解决方案：

建立资源池，利用容器云实现源码编译任务按需分配。可以减少硬件设备采购成本50%以上。

开发转测试交付由单纯的“源码交付”改为“源码与编译环境镜像交付”。

使用docker取代chroot来进行资源管理。

镜像优化-Dockerfile

- 镜像用途单一，避免力求大而全。
- 根据功能选择尽量小的基础镜像，如debian。避免造成镜像臃肿。
- 减少镜像的层数，每一个RUN命令对应着一层，可将多个RUN命令合为一个。
- 删除临时文件或缓存文件。apt-get clean
- 将内容不变的指令放在前面，或将常用的指令单独打包。
- COPY比ADD更透明。
- 避免安装不必要的软件包。
- 谨慎使用外部源数据。
- 建议使用明确的版本号，如ubuntu:14.04而不是ubuntu:latest。
- 不要将dockerfile与其它无关文件存放在同级目录中。
- 使用.dockerignore文件忽略匹配模式路径下的目录和文件。
- 利用--squash flag合并镜像层。
- 利用docker multi-stage build，docker 17.05版本新特性。

TiD2017

镜像优化

优化前的镜像

```
FROM ubuntu:latest

MAINTAINER Yuan Sun sunyuan3@huawei.com

WORKDIR /root

RUN apt-get update
RUN apt-get install -y gcc
RUN apt-get install -y make
RUN apt-get install -y vim git

RUN git clone https://github.com/sunyuan3/netperf.git  && cd netperf && ./configure && make && make install
RUN rm -rf /root/netperf

CMD ["/bin/sh", "-c", "/usr/local/bin/netserver"]
```

镜像优化

1

1. 基础镜像使用了latest标签。
2. 每一个RUN命令会创建一个新的镜像层，会导致资源浪费。
3. 安装了不必要的软件包。
4. apt-get操作时间过长，每次会有重复操作。
5. 对于apt-get，没有做相关的清除。
6. RUN rm -rf /root/netperf操作无法减少镜像尺寸。

问题点

2

1. 使用确定版本的镜像。
2. 将同类操作的RUN命令合并减少镜像层数。
3. 移除不必要的软件包。
4. 将apt-get操作作为不频繁变更的镜像，或者充分利用缓存。
5. 添加apt-get clean
6. 将下载文件与删除文件的操作合入到一行，同时使用multi-stage特性。

解决方法

镜像优化

初次优化

```
FROM ubuntu:14.04
MAINTAINER Yuan Sun sunyuan3@huawei.com
WORKDIR /root
RUN apt-get update && apt-get install -y gcc make vim git && apt-get clean
```

docker build -t ubuntu:14.04env .

```
FROM ubuntu:14.04env
MAINTAINER Yuan Sun sunyuan3@huawei.com
WORKDIR /root
RUN git clone https://github.com/sunyuan3/netperf.git \
    && cd netperf \
    && ./configure \
    && make && make install \
    && rm –rf /root/netperf
CMD ["/bin/sh", "-c", "/usr/local/bin/netserver"]
```

TiD2017

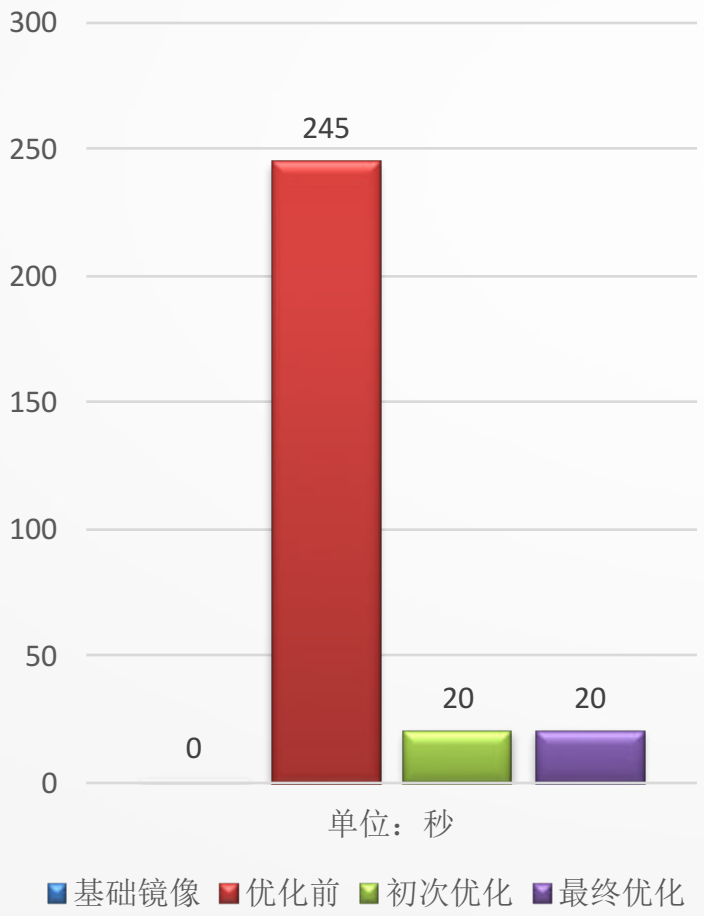
镜像优化

最终优化

```
FROM ubuntu:14.04  as builder

MAINTAINER Yuan Sun sunyuan3@huawei.com

WORKDIR /root

RUN apt-get update && apt-get install -y gcc make vim git

RUN git clone https://github.com/sunyuan3/netperf.git  && cd netperf && ./configure && make && make install

CMD ["/bin/sh", "-c", "/usr/local/bin/netserver"]

FROM ubuntu:14.04
COPY --from=builder  /usr/local/bin/netserver  /usr/local/bin/
CMD ["/bin/sh", "-c", "/usr/local/bin/netserver"]
```

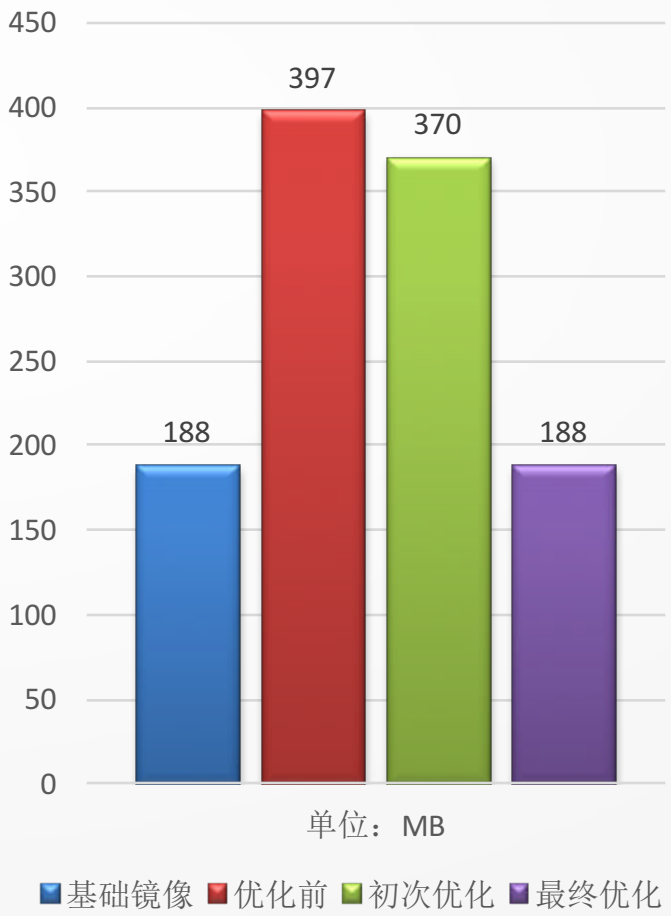
镜像优化

镜像创建时间



时间减少
91.8%

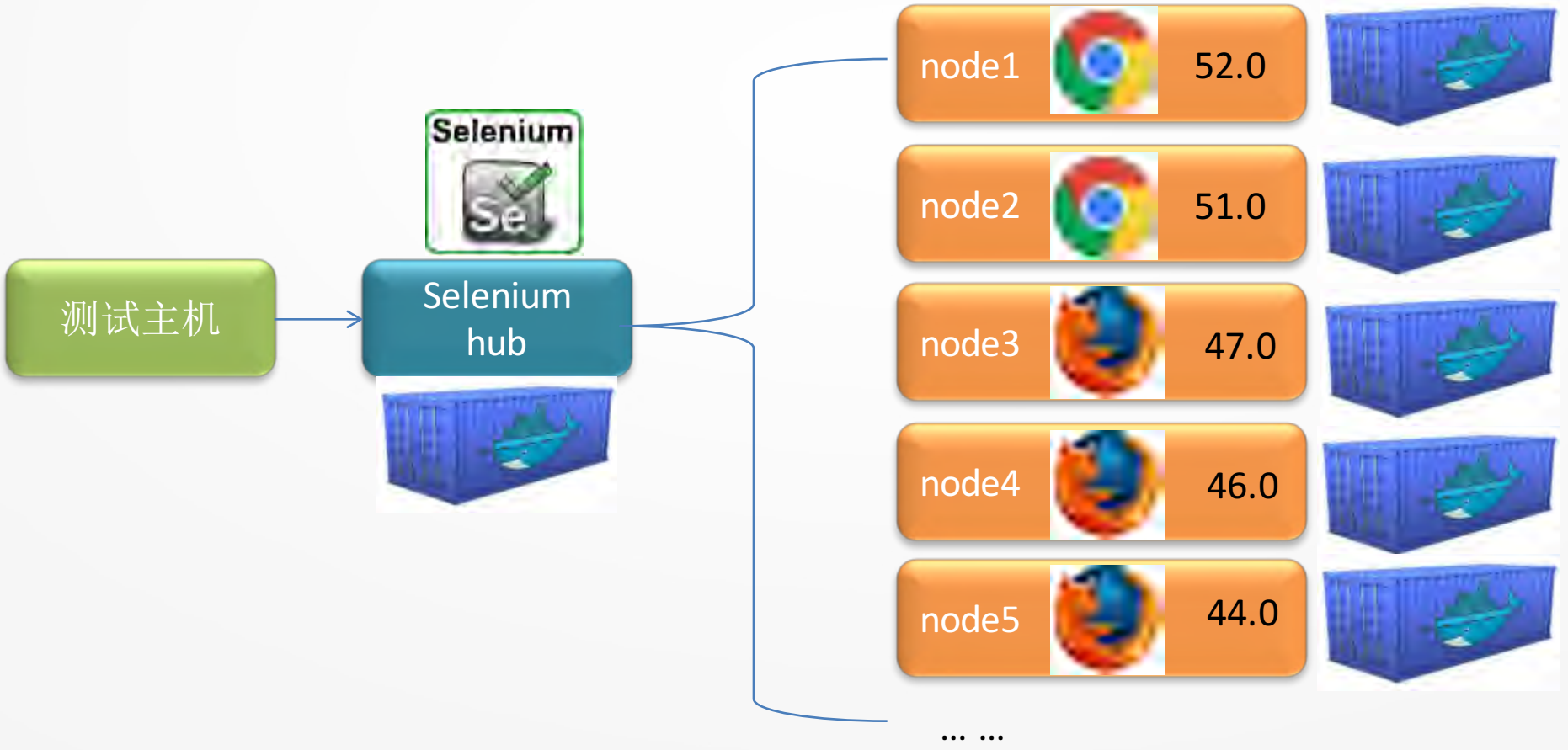
镜像大小



镜像大小
减少52.6%

应用层软件测试容器化

- 将Docker和Devops结合起来使用，统一开发、测试、运维环境，打通全流程自动化任务。
- 利用容器云中Docker可扩展可伸缩的特性加速测试执行或者模拟多个终端用户来进行测试。



应用层软件测试容器化

selenium

http://docs.seleniumhq.org/

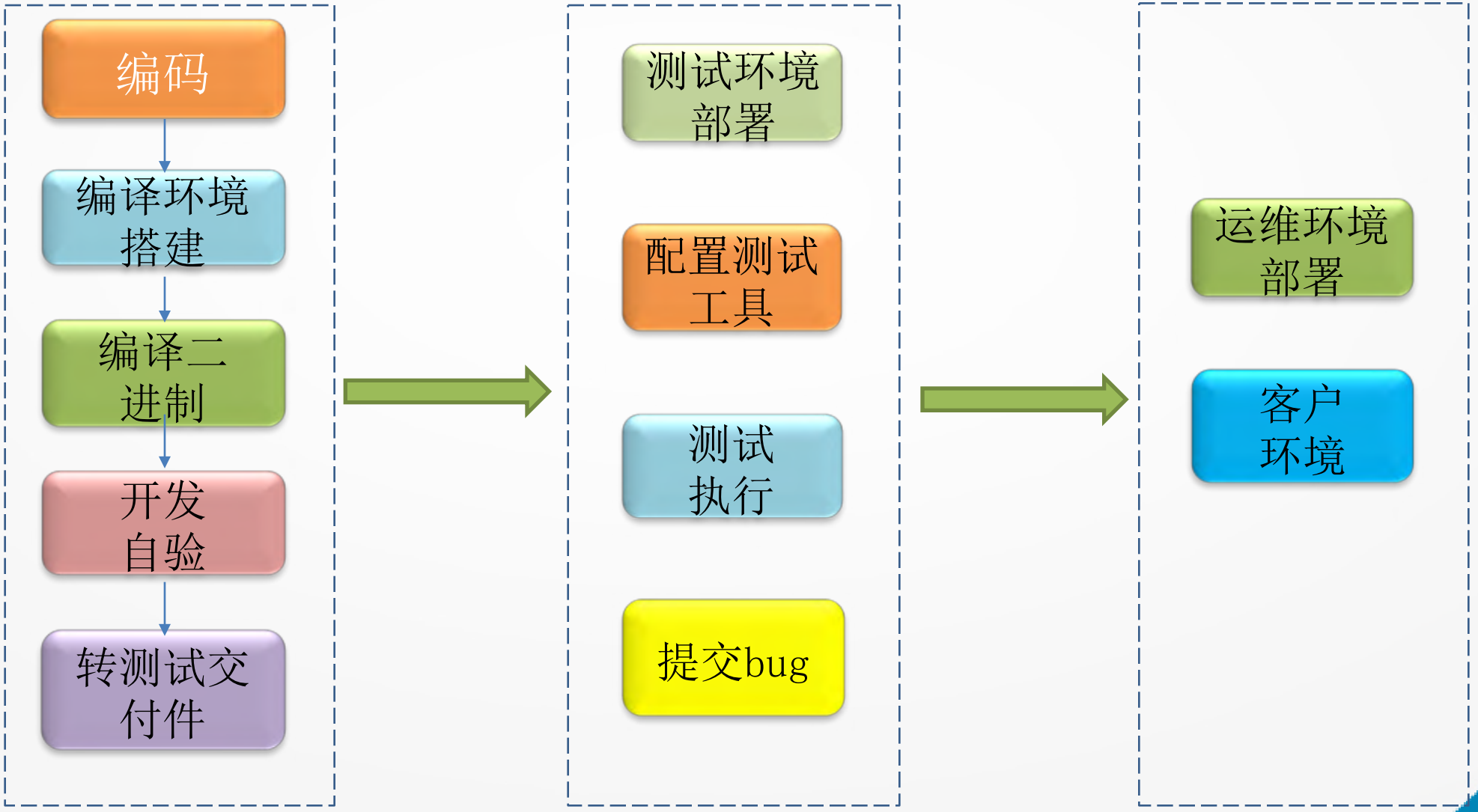
Joined November 2014

selenium/node-chrome public | automated build	103 STARS	5M+ PULLS	DETAILS
selenium/standalone-chrome public | automated build	67 STARS	1M+ PULLS	DETAILS
selenium/node-firefox public | automated build	75 STARS	1M+ PULLS	DETAILS
selenium/standalone-firefox public | automated build	63 STARS	1M+ PULLS	DETAILS
selenium/hub public | automated build	196 STARS	1M+ PULLS	DETAILS
selenium/standalone-chrome-debug public | automated build	25 STARS	1M+ PULLS	DETAILS

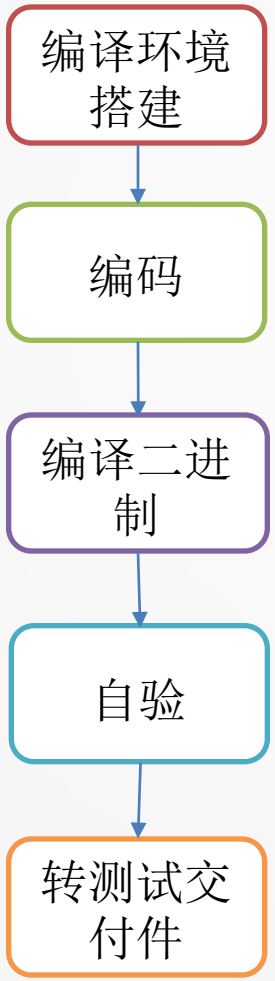
应用层软件测试容器化

- 利用容器隔离性和快速环境清理的特性，并行执行软件兼容性测试(如安装不同版本的数据库)。
- 通过容器模拟测试执行需要的外部环境。可将一些典型的测试场景进行容器化处理，将镜像存储到镜像仓库中。
- 可以通过docker-compose以微服务的方式部署这些测试场景，便于其他团队复用已有的测试场景。
- 如想在容器云中运行与内核相关的应用，可将应用程序进行内核特性解耦，将内核模块对应的部分代码移植到应用程序中。

应用层软件测试容器化


编码
编译环境搭建
编译二进制
开发自验
转测试交付件
测试环境部署
配置测试工具
测试执行
提交bug
运维环境部署
客户环境

应用层软件测试容器化



1

手动搭建环境，难以保证环境一致。

编码环境不易配置，如vim中安装golang控件。

无法实现编译动态资源扩展，不易加速。

搭建自验环境较长，开发自验往往不充分。在同一套环境中往往对不同交付件进行验证，容易造成脏数据，污染环境。

交付件为源码或二进制，不仅环境不易不统一，而且部署复杂。

容器化前

2

可通过dockerfile梳理编译环境搭建步骤，确保环境一致。

秒级部署编码环境。

通过底层docker update命令可以实现资源动态扩展，加速效果可观。

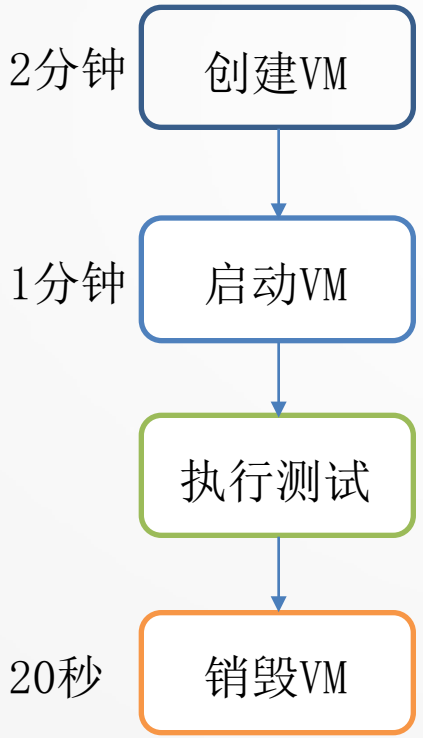
通过docker镜像打包测试环境，可以秒级配置环境。并通过web hook强制执行自动化测试。有力的支撑了测试迁移。Docker提供了清洁的环境。

交付件为docker镜像，可以秒级部署。

容器化后

应用层软件测试容器化

- 测试环境部署

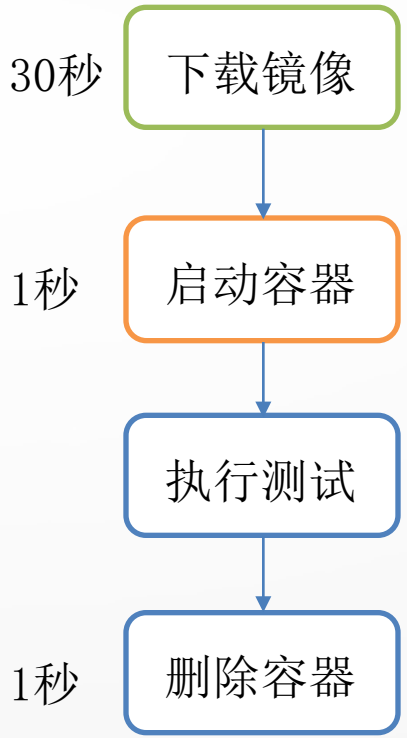


1

测试环境搭建费时费力，为了保证环境清洁性需要频繁创建VM。否则环境污染问题会累积。

配合多套环境时，时间将乘以N。

容器化前



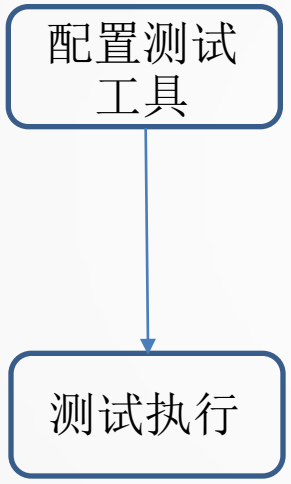
2

将测试环境打包为镜像，可以秒级部署。可以轻松创建多套环境。

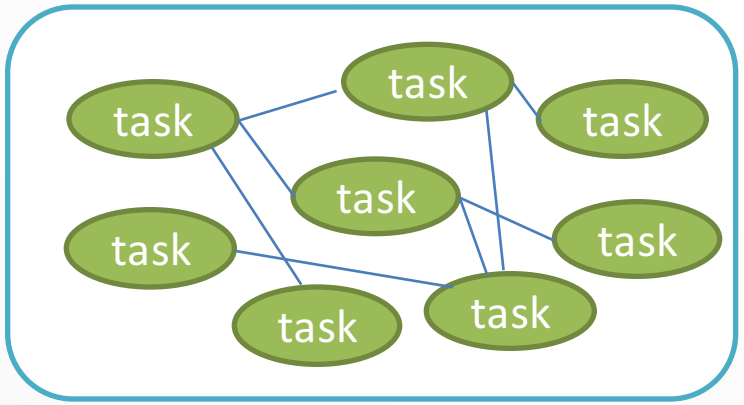
配合多套环境时，几乎不增加时间。

容器化后

应用层软件测试容器化



配置测试工具工作复杂，成功率低。

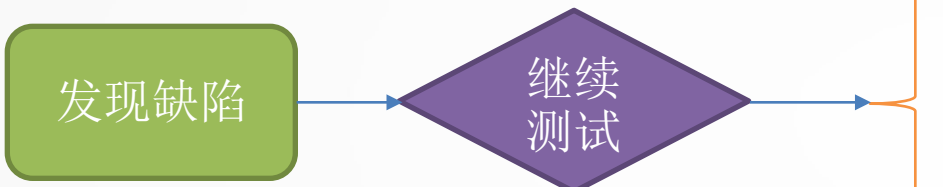


多任务共享主机资源，环境污染难以避免；测试资源难以快速扩展。

测试工具容器化后可以秒级部署，成功率高。

利用容器可以隔离测试环境，利用容器云可以非常容易的自行资源扩展和伸缩，最大限度的并行执行测试。

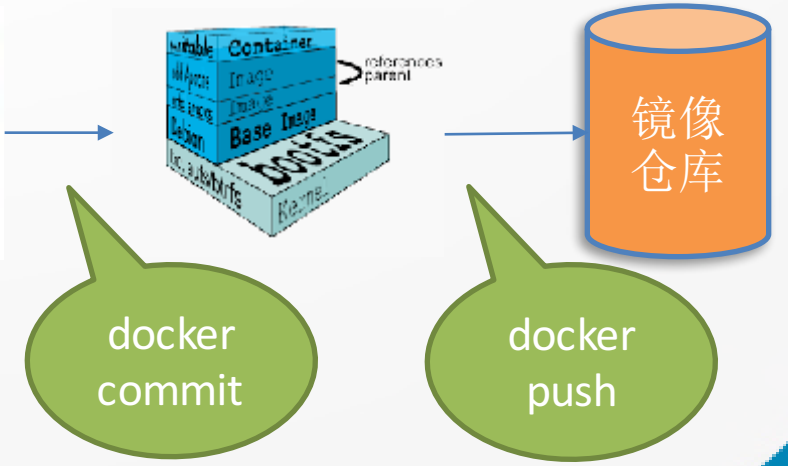
应用层软件测试容器化



Y: 缺陷现场难以保留，其它测试程序可以会污染环境。

N: 其它测试任务不得不中断。

- 通过docker commit将环境打包成为镜像上传到仓库中并提供给开发团队。
- 启动新的容器执行其它测试任务。



应用层软件测试容器化

运维环境部署

- 容器化前：运维难以解决业务峰值。环境出问题后，开发团队不易复现。
- 容器化后：可以通过cloud native解决业务峰值。开发团队可以很容易的复现缺陷。

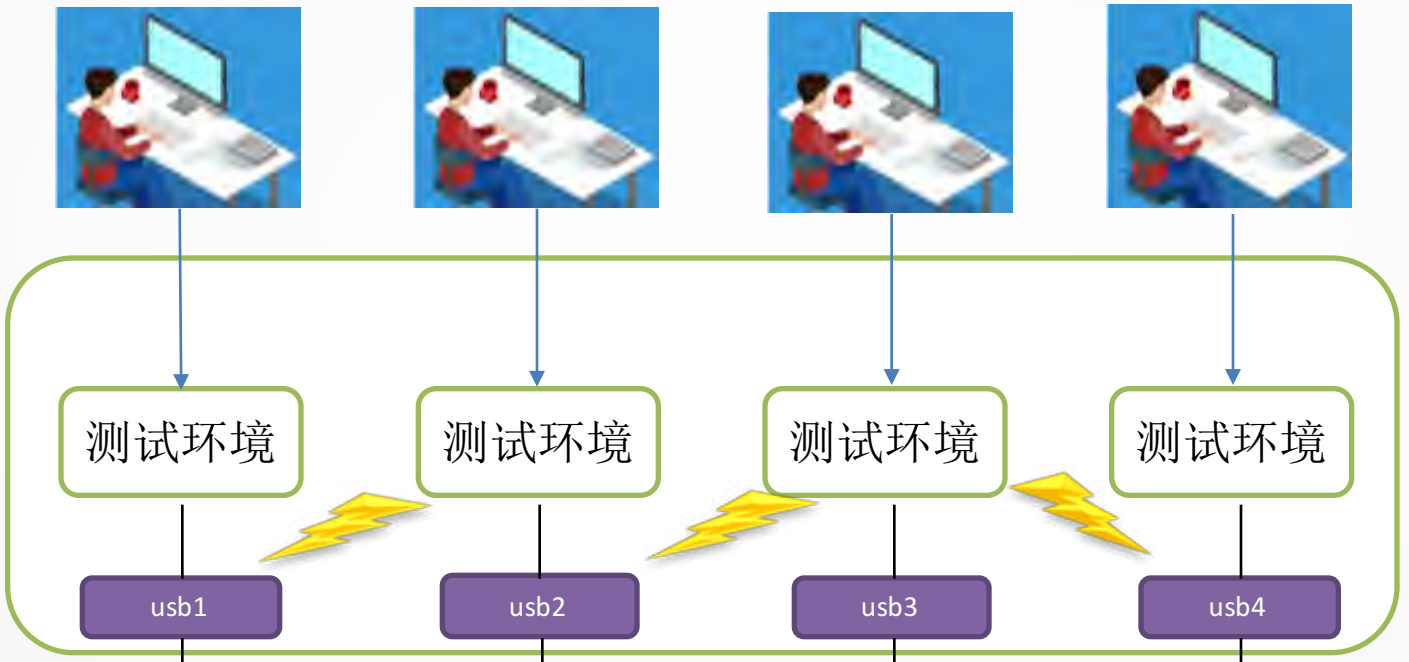
客户环境

- 容器化前：客户环境出现问题，开发团队难以复现。
- 容器化后：可以通过docker save将环境打包成为镜像。

Docker与Jenkins

Docker插件	功能
Docker Commons Plugin	为其它docker插件提供api
Docker Plugin	可以使用Docker主机动态分配的容器作为Jenkins的从节点。
Docker Slaves Plugin	使用容器来配置构建agents，对于镜像的使用没限制。
Docker Pipeline Plugin	允许构建和使用pipelines中的容器。
Kubernetes Plugin	通过由Kubernetes管理的多个Docker主机系统来动态分配的容器作为Jenkins的从节点。
Mesos Plugin	使Jenkins可以根据工作负载动态启动 Mesos集群中的Jenkins slaves。
Swarm Plugin	使slaves可以自动发现Jenkins master并自动加入。

移动终端测试容器化



Usb连线

移动终端测试容器化

痛点分析：

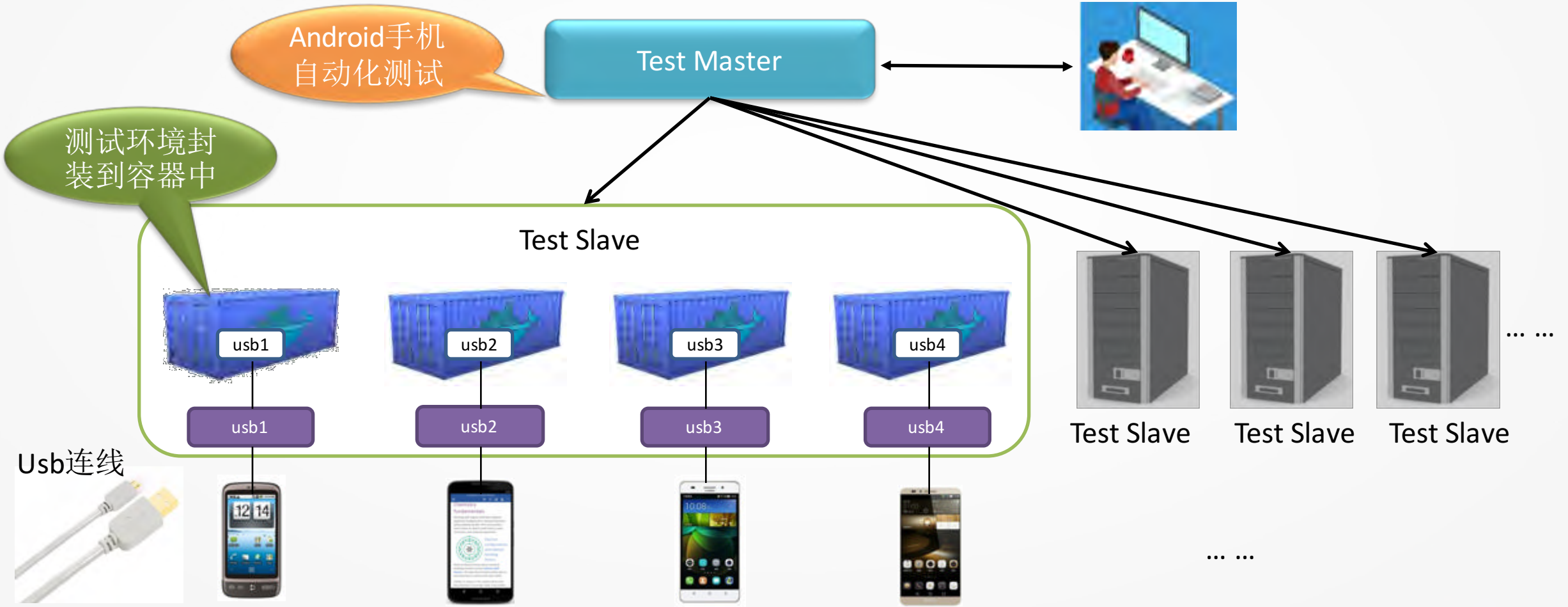
- 工程师通常只对自己的测试环境熟悉。
- 测试环境和设备无法共用，无法有效发挥自动化价值。
- 测试环境与被测终端测试任务无法避免污染。

移动终端测试容器化

技术挑战：

- 需要充分发挥自动化测试价值。
- 测试环境需要能够复用。
- 有效避免环境污染，测试任务不受到其它因素的干扰。
- 充分利用硬件资源。
- 任何人在任何时间任何地点可以在任何设备中运行任何自动化用例。
- 满足开发自验、CI、缺陷复现等需求。

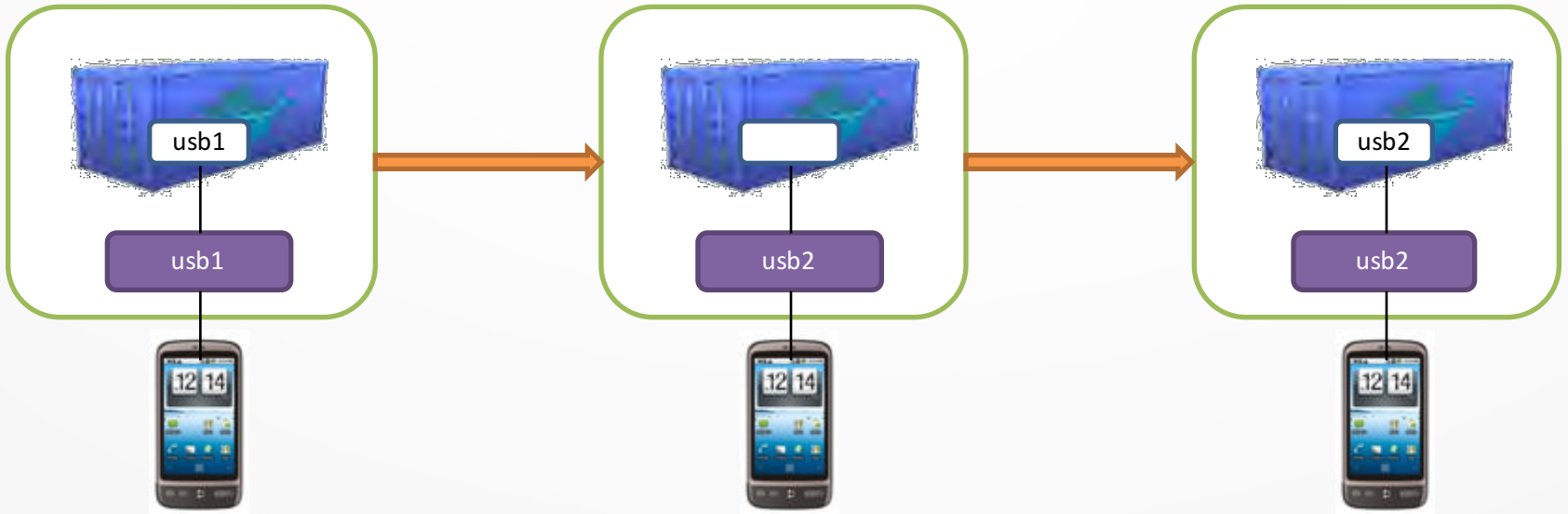
移动终端测试容器化



docker run -ti --device=/dev/bus/usb/002/001 ubuntu:test /bin/bash

移动终端测试容器化

- 难点：测试时手机重启导致硬件设备号更改，而docker不支持动态加载硬件设备。



移动终端测试容器化

- 解决方法：

1. 求助社区，添加新的特性。
2. 停止当前测试容器，启动新的容器来加载新的设备号。
3. 修改/etc/udev/rules.d/70-persistent-net.rules使手机重启后设备号不变。
4. 通过底层device cgroup特性动态加载新的设备号。

移动终端测试容器化

动态加载实现原理：

启动容器并获取设备信息，开始测试。

手机重启后，遍历所有usb 设备，定位新的设备号。

将设备加入到device cgroup允许容器访问的列表中，并在容器中创建设备节点。

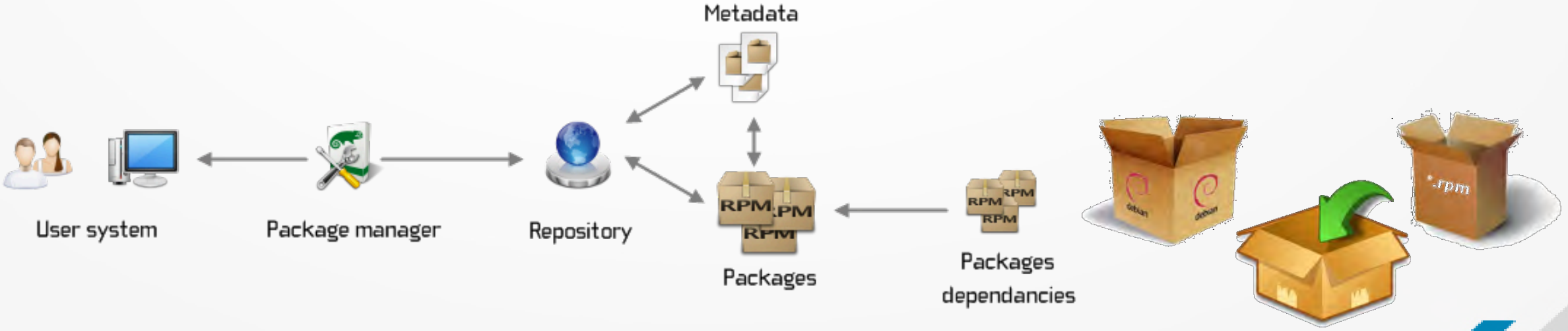
移动终端测试容器化

实现细节：

- 利用udevadm info /dev/bus/usb001/011 命令获取id和设备号major和minor(189,10)。
- 启动容器：docker run -ti -device=/dev/bus/usb/001/011 ${test_image} ${test_command}
- 手机重启后通过udevadm info命令找到硬件设备的新设备号major和minor(189,11)。
- 进入容器对应的cgroup目录/sys/fs/cgroup/devices/docker/${containerid}，可以查询到c 189:10 rwm在devices.list文件中。通过echo “c 189:11 rwm” >> devices.allow命令修改device cgroup文件，向容器开放该设备。
- 在容器内部执行mknod /dev/bus/usb/001/012 c 189 11命令，使容器可以直接访问到手机设备。

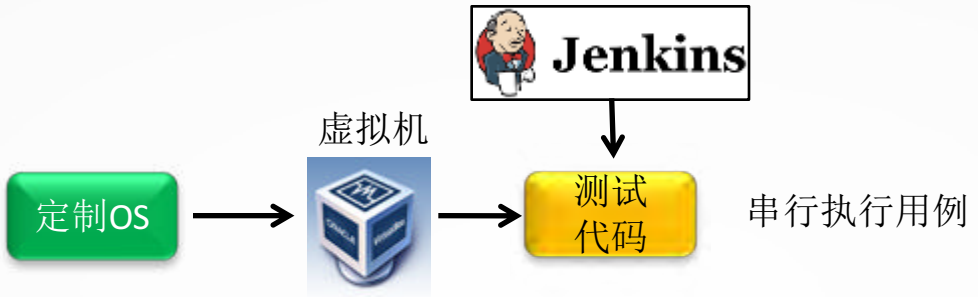
中间层软件测试容器化

- 大部分Linux外围包（通常是rpm或deb包）与内核无强依赖，是适合使用容器云进行加速的。
- 因为在容器云中无法自定义内核，与内核相关性强的外围包的测试是不适用容器云。解决方法：搭建可定制内核的私有云

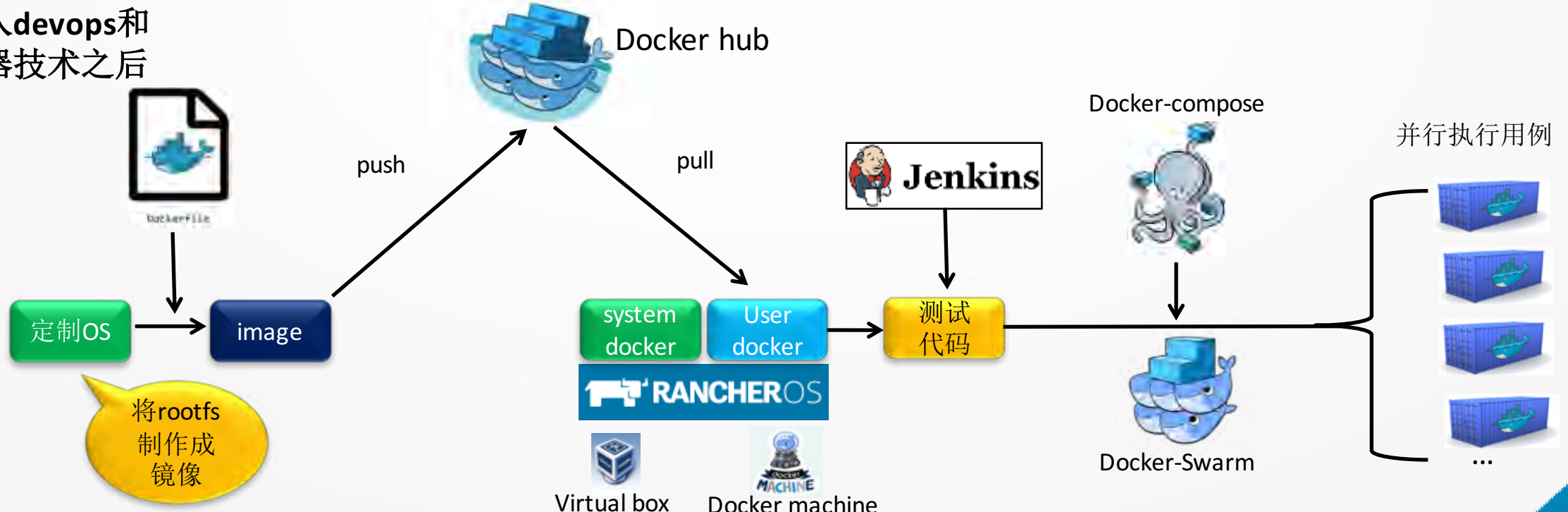


中间层软件测试容器化

引入devops和
容器技术之前



引入devops和
容器技术之后



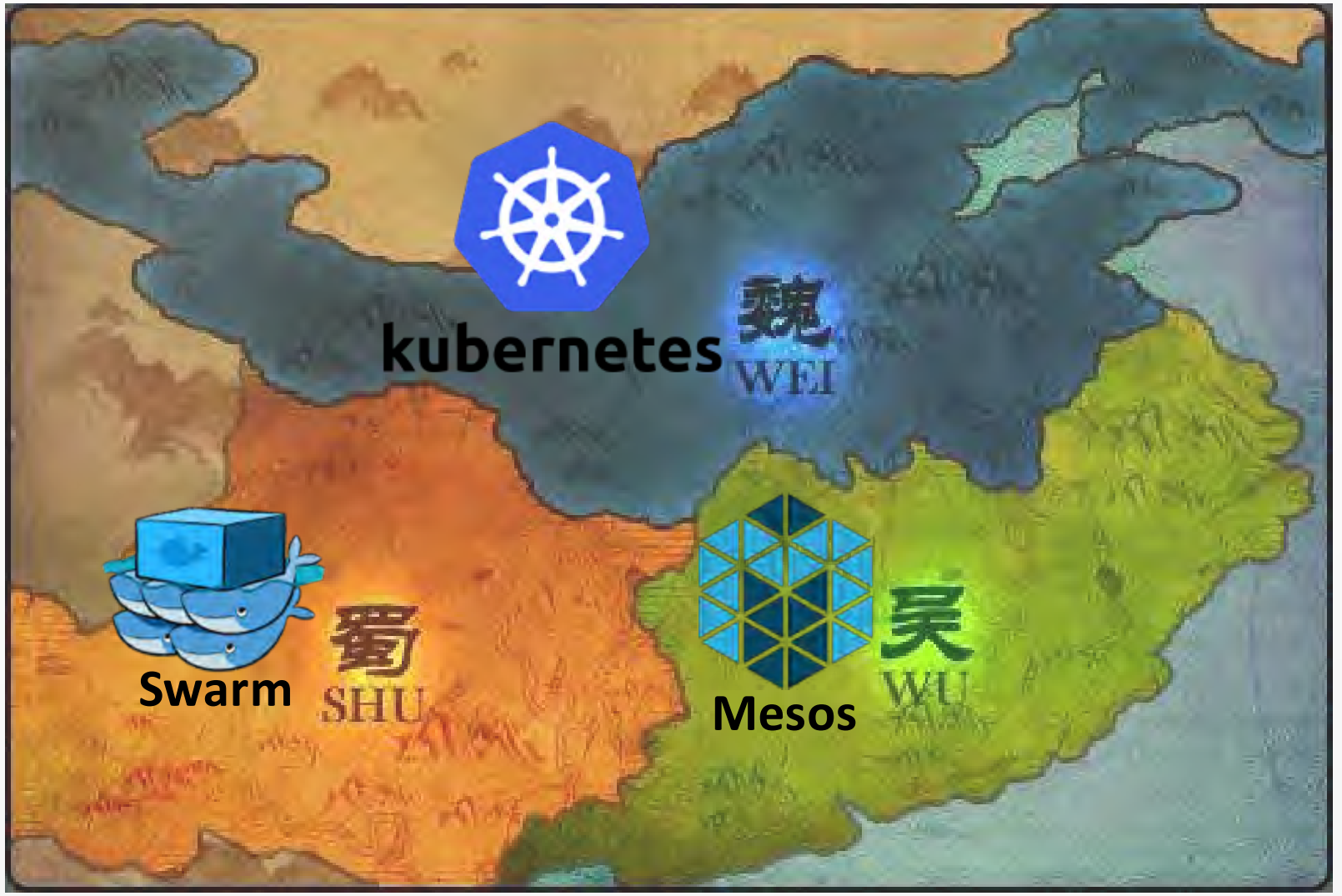
中间层软件测试容器化

- 特例：
- (1)grub包:grub是多系统启动规范的实现，因为与主机启动强相关。

- (2)ntp包:ntp是网络时间协议(Network Time Protocol)，用来同步网络中各个计算机的时间的协议。

  当前内核不支持time namespace，即容器与主机间、容器与容器间不能使用不同的时间（注意是时间不是时区）。
- (3)与init有强相关的包。

Kubernetes Swarm Mesos



Kubernetes Swarm Mesos

	Swarm	Kubernetes	Mesos
基本特性	服务编排、服务发现、服务扩容伸缩、服务健康检查、负载均衡、升级和回滚、默认安全通信、集群高可用、资源逻辑隔离	服务编排、服务发现、服务扩容伸缩、服务健康检查、负载均衡、升级和回滚、安全配置管理、集群高可用、资源逻辑隔离、节点扩容、job任务支持、运行监控/日志	集群高可用、资源逻辑隔离
架构特点	Docker体系	Borg，原生docker	借鉴borg理念
依赖组件	合入到docker engine中，无需第三方组件	需要Kubernetes自身组件：kubelet、kube-proxy、kube-apiserver、kube-scheduler、kube-controller-manager	出了配置mesos外，还需要配置Framework和zookeeper等第三方组件。
扩展伸缩	支持	支持	支持
健康检查	支持	支持	支持
故障迁移	支持	支持	支持
负载均衡组件	内置	Kube-proxy	Haproxy

Kubernetes Swarm Mesos

	Swarm	Kubernetes	Mesos
资源调度类型	内存、CPU	内存、CPU、磁盘端口、网络	内存、CPU、磁盘端口、网络
编排策略	资源使用+应用节点均衡	资源使用+应用节点均衡	资源使用
调度容器	Docker	Docker	Mesos/Docker
安全	TLS	自定义安全配置	SSL/TLS
网络	Overlay	Gre、vxlan等	自定义
高可用	Raft	Etcd	zookeeper
商业应用	有	有	有

内核测试容器化

内核测试容器化改造方案：

- Ltp测试套中的用例默认是串行执行的，可启动多个容器并发执行内核功能测试来提高cpu负载，以便减少测试执行时间。

- 可以在容器内搭建内核测试环境。

Ltp测试套中的测试用例源码默认是被动态编译的，其运行需要一些动态链接库。而很多系统特别是嵌入式系统往往缺少这些动态链接库，导致部分内核测试用例无法执行。可将必要的动态库集成到Docker镜像中，这样既保证了测试用例可以顺利执行，又可以避免在主机中直接安装库文件导致环境污染。

- 使用Docker构建内核集成测试场景。

因为内核是底层技术，很难找到一个能够包含多个内核特性的集成测试场景。通过以下Docker命令可以轻松搭建namespace、cgroup、capability、seccomp等内核特性的集成测试场景，大大减少集成测试设计时间。

```
$ docker run --memory 20m --cap-add sys_admin --security-opt seccomp=unconfined --userns-remap default ubuntu:14.04 bash
```

内核测试容器化

利用Docker触发内核Bug: CVE-2016-9191

描述：The cgroup offline implementation in the Linux kernel through 4.8.11 mishandles certain drain operations, which allows local users to cause a denial of service (system hang) by leveraging access to a container environment for executing a crafted application, as demonstrated by trinity.

测试步骤：

```
# docker run -it caiqian/rhel-tools bash
container> # su - test
container> $ ulimit -c 0
container> $ trinity -D --disable-fds=memfd --disable-fds=timerfd \
             --disable-fds=pipes --disable-fds=testfile \
             --disable-fds=sockets --disable-fds=perf \
             --disable-fds=epoll --disable-fds=eventfd \
             --disable-fds=drm
After 30-minute, interrupt (Ctrl-C) all the trinity processes, and then,
container> $ exit
container> # exit
# systemctl status docker
<hang...>
```

内核测试容器化

局限性：

1. 在容器内执行内核测试时会有部分用例执行失败。例如，pidns32测试用例是用来测试namespace的最大嵌套深度，但容器已经嵌套了一层namespace。
2. 某些内核测试项是排他的，需要单独运行，这部分测试需要进行额外的操作处理，比如一些中断操作、寄存器操作。
3. 内核性能测试的执行不适用容器化，否则会有失真。

Ltp(linux test project)测试套pidns32测试用例

```
#define MAXNEST 32
...
static int child_fn1(void *arg)
{
...
        if (level == MAXNEST)
                return 0;
        cpid1 = ltp_clone_quick(CLONE_NEWPID | SIGCHLD,
                (void *)child_fn1, (void *)(level + 1));
}
```

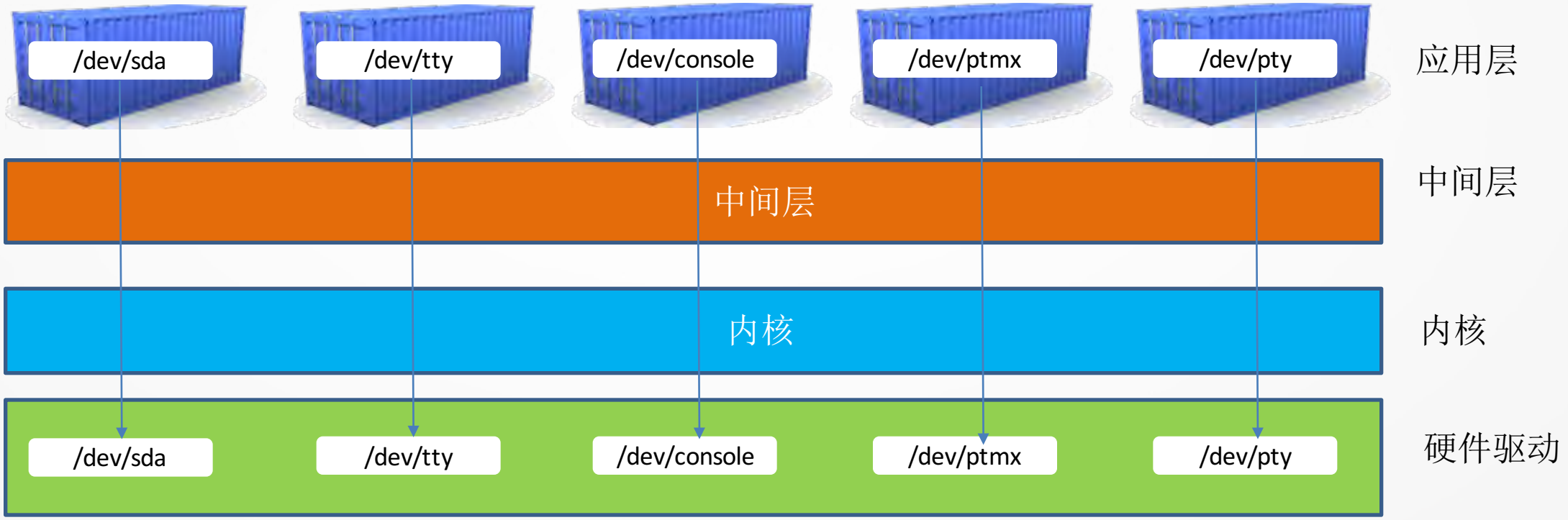
硬件驱动测试容器化

•难点：Docker的设计初衷是来屏蔽各硬件平台差异的。
•方案：Docker利用内核的device cgroup特性可以实现设备直通的功能，即可将主机中的设备映射到容器中进行测试。
•收益：测试环境不会污染主机环境；便于设备驱动并行测试；即使主机文件系统缺少测试所需的库文件，仍然可以使用容器的文件系统中的工具来辅助测试。

例子：磁盘块设备测试

```
$ docker run -device /dev/sda:/dev/sda ${your_image} ${your_test_script}
```

硬件驱动测试容器化


/dev/sda
/dev/tty
/dev/console
/dev/ptmx
/dev/pty
应用层
中间层
内核
/dev/sda
/dev/tty
/dev/console
/dev/ptmx
/dev/pty
硬件驱动

压力稳定性测试痛点

- 部署工作复杂。

加压测试套编译与运行有很强的平台依赖性，当测试涉及多平台复杂组网时，会出现很多兼容性问题。系统巡检工具的安装依赖包多，需要人工进行各个版本依赖包的适配工作。一套完整的长稳测试框架的部署（包括背景加压、系统巡检与用例执行）会耗费大量人力，影响了测试效率。

- 加压模型单一，多个模型不易叠加。

- 容错能力差。

加压程序有可能本身存在内存泄露，导致开发团队不认可测试结果。加压程序垮掉时往往需要借助外部环境恢复。

压力稳定性测试容器化

压力稳定性测试容器化

●CPU负载

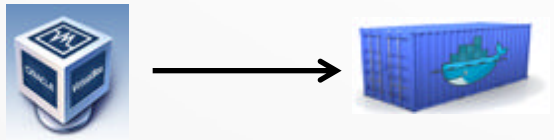
-按照加压模型，对cpu占有率按照30%~80%

●内存负载

-按照加压模型，对内存占有率按照50%~80%进行规划

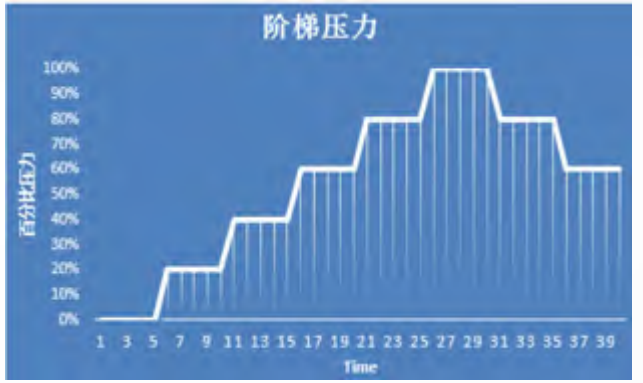
●IO频率

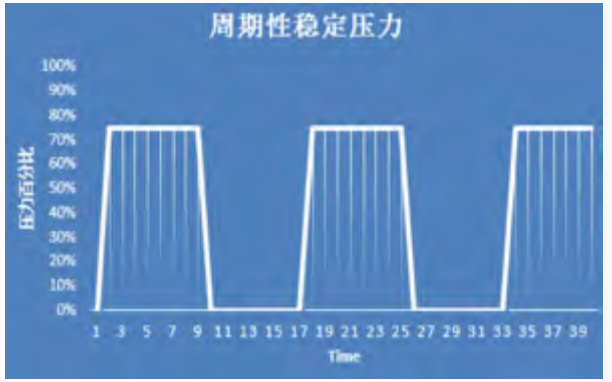
-使用fio工具

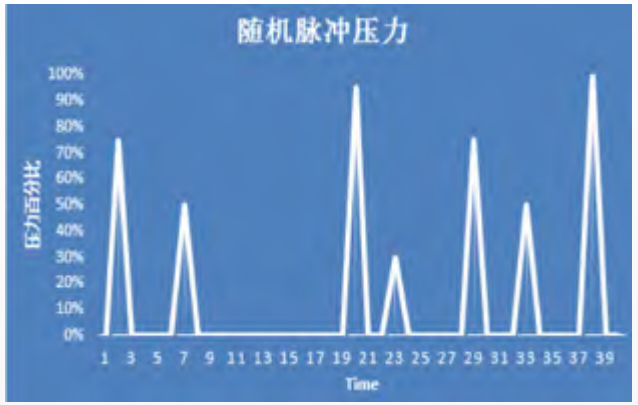


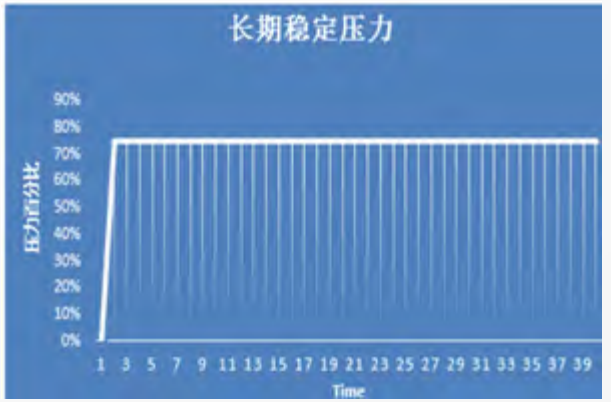
虚拟机　　容器

确保测试运行的更加平稳、持续；将测试工具内存泄露等问题带来的负面影响降到最低。









压力稳定性测试容器化

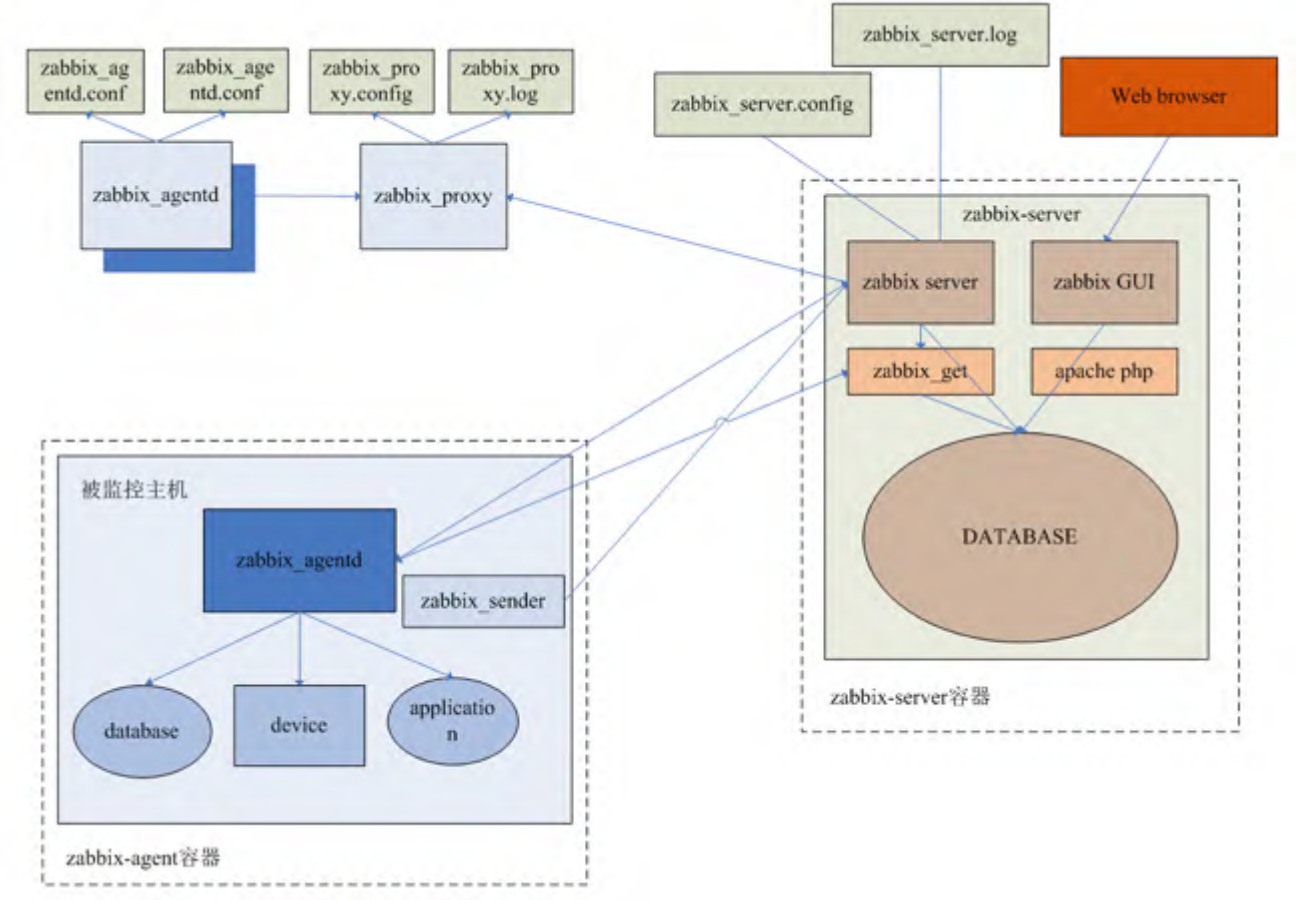
- 利用容器中的文件系统支撑加压程序运行，可以使长稳测试运行更加标准化，便于拉通多个测试组的测试能力。
- 容器化可以避免因加压程序导致系统垮掉的问题。

  加压程序有内存泄露时，当容器试图使用超过限定内存时会触发OOM，只要设定好restart规则后容器所占资源会被清理同时容器会快速重新启动并持续进行加压。

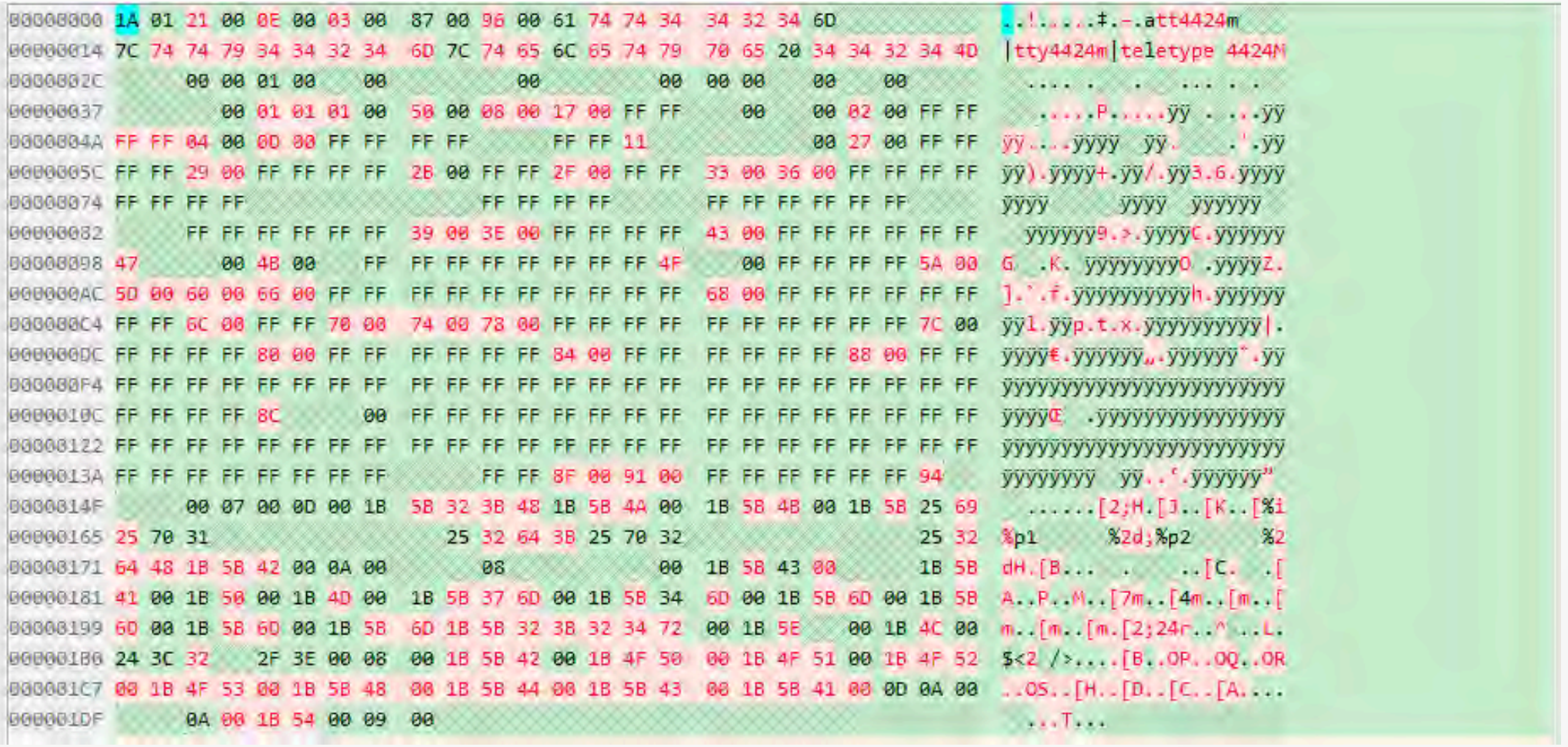
```
$ docker run -tid -m 3g --restart=always \
back_stress bash \
-c “cpu_stress/test.sh”
```

压力稳定性测试容器化

- 系统巡检的容器化。将zabbix巡检工具的server端与agent端均打包为镜像，在多平台组网复杂的环境中，可以完全屏蔽平台化差异。



二进制一致性测试


00000000 1A 01 21 00 0E 00 03 00 87 00 96 00 61 74 74 34 34 32 34 6D
00000014 7C 74 74 79 34 34 32 34 6D 7C 74 65 6C 65 74 79 70 65 20 34 34 32 34 4D
0000002C 00 00 01 00 00 00 00 00 00 00 00 00
00000037 00 01 01 01 00 58 00 08 00 17 00 FF FF 00 00 02 00 FF FF
0000004A FF FF 04 00 0D 00 FF FF FF FF FF FF 11 00 27 00 FF FF
0000005C FF FF 29 00 FF FF FF FF 2B 00 FF FF 2F 00 FF FF 33 00 36 00 FF FF FF FF
00000074 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000082 FF FF FF FF FF FF 39 00 3E 00 FF FF FF FF 43 00 FF FF FF FF FF FF
00000098 47 00 4B 00 FF FF FF FF FF FF FF FF 4F 00 FF FF FF FF 5A 00
000000AC 5D 00 60 00 66 00 FF FF FF FF FF FF FF FF FF FF 68 00 FF FF FF FF FF FF
000000C4 FF FF 6C 00 FF FF 70 00 74 00 78 00 FF FF FF FF FF FF FF FF FF FF 7C 00
000000DC FF FF FF FF 80 00 FF FF FF FF FF FF 84 00 FF FF FF FF FF FF 88 00 FF FF
000000F4 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0000010C FF FF FF FF 8C 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000122 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0000013A FF FF FF FF FF FF FF FF FF FF 8F 00 91 00 FF FF FF FF FF FF 94
0000014F 00 07 00 0D 00 1B 5B 32 3B 48 1B 5B 4A 00 1B 5B 4B 00 1B 5B 25 69
00000165 25 70 31 25 32 64 3B 25 70 32 25 32
00000171 64 48 1B 5B 42 00 0A 00 08 00 1B 5B 43 00 1B 5B
00000181 41 00 1B 50 00 1B 4D 00 1B 5B 37 6D 00 1B 5B 34 6D 00 1B 5B 6D 00 1B 5B
00000199 6D 00 1B 5B 6D 00 1B 5B 6D 1B 5B 32 3B 32 34 72 00 1B 5E 00 1B 4C 00
000001B0 24 3C 32 2F 3E 00 08 00 1B 5B 42 00 1B 4F 50 00 1B 4F 51 00 1B 4F 52
000001C7 00 1B 4F 53 00 1B 5B 48 00 1B 5B 44 00 1B 5B 43 00 1B 5B 41 00 0D 0A 00
000001DF 0A 00 1B 54 00 09 00

二进制一致性测试

Docker隔离性的限制：

- /proc，/sys等未完全隔离
- top，free，iostat等命令展示的信息未隔离
- /dev设备未隔离
- 内核模块未隔离
- selinux，time，syslog等所有现有namespace之外的信息未隔离

二进制一致性测试

/proc文件系统主要信息

子模块	描述	隔离性
1.net	本系统配置的网络信息，包括网络状态、所支持的协议、socket信息等。	部分隔离
2.内存	包含内存映射、使用情况、内存碎片等	未隔离
3.devices	设备（块设备、字符设备）信息	未隔离
4.当前进程Id文件夹	包含所有环境变量、进程状态、内存使用情况等	未隔离
5.fs	本系统所有文件系统信息	未隔离
6.硬件信息	中断信息、irq信息、pci等信息	未隔离
7.内核信息	模块、挂载、调度等信息	未隔离

二进制一致性测试

- Audit namespace

未合入原因:意义不大,而且会增加audit的复杂度, 难以维护。

- Syslog namespace

未合入原因:很难完美的区分哪些log应该属于某个container。

- Device namespace

未合入原因:几乎要修改所有驱动,改动太大。

- Time namespace

未合入原因:一些设计细节上未达成一致,而且感觉应 用场景不多。

内核社区对容器技术要求的隔离性,本的原则是够用就好,不要把内核搞的太复杂 。

应用软件安装测试限制

问题

- 在容器中可以安装成功，为什么在主机中无法安装成功？

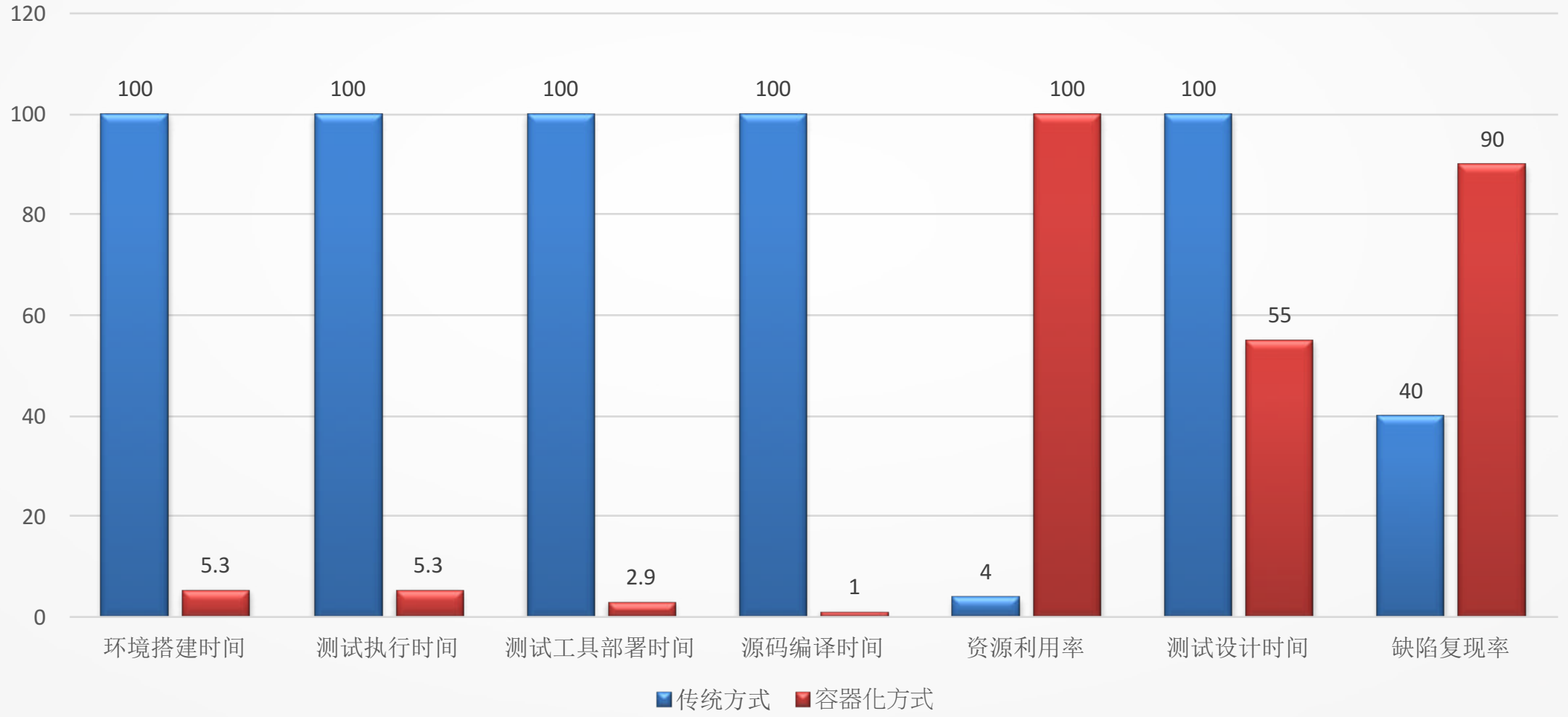
常见原因

- 容器镜像的默认安装包比虚拟机小很多，二者的文件系统是不同的。
- 容器和主机对应的内核版本以及其它未被隔离的信息不同。

解决方法

- 基于虚拟机的文件系统导出镜像。
- 编程时将程序与内核版本和其它未隔离的信息解耦。

改造前后对比图


120
100
80
60
40
20
0
100
100
100
100
100
100
5.3
5.3
2.9
1
4
55
40
90
环境搭建时间
测试执行时间
测试工具部署时间
源码编译时间
资源利用率
测试设计时间
缺陷复现率
传统方式
容器化方式

Docker实践KPI矩阵

	典型测试场景微服务化	源码编译	测试工具容器化	测试执行加速	应用层软件测试容器化	移动终端测试容器化	中间层软件测试容器化	内核测试容器化	硬件驱动测试容器化	压力稳定性测试容器化	二进制一致性测试容器化
环境搭建时间减少	95%＋	99%＋			99%+	90%+	95%+	90%+	90%+	99%+	95%+
测试执行时间降低					99%+		95%+		80%+		
测试工具部署时间减少			99%+		99%+	90%+	99%+		99%+	99%+	95%+
源码编译时间降低		99%＋			99%+						
资源利用率提升	95%＋	99%＋	95%+	99%+	99%+		95%+		90%+		
节约物料		20VM+	20VM+		40VM+	50VM+	50VM+				10VM+
减少测试设计时间					20%+			70%+			
缺陷复现率提升		50%+			50%+		50%+				

Docker实践矩阵

	典型测试场景微服务化	源码编译	测试工具容器化	测试执行加速	应用层软件测试容器化	移动终端测试容器化	中间层软件测试容器化	内核测试容器化	硬件驱动测试容器化	压力稳定性测试容器化	二进制一致性测试容器化
快速部署	☺	☺	☺	☺	☺		☺			☺	☺
资源限制	☺	☺	☺	☺	☺		☺			☺	
环境隔离	☺	☺	☺	☺	☺	☺	☺		☺		
弹性伸缩		☺		☺	☺		☺				
环境共享与迁移	☺	☺	☺	☺	☺	☺	☺			☺	☺
高资源利用率	☺	☺	☺	☺	☺	☺	☺				
Dockerfile管理	☺	☺	☺	☺	☺	☺	☺	☺		☺	☺

常见问题

- 避免因系统最大进程数量限制导致容器业务无法正常运行。

  echo ${new_pid_max} > /proc/sys/kernel/pid_max

- 出现no tty错误时

  强制创建tty：ssh -t -t 或docker run -d

- 如何判断所在的系统是主机还是容器？

  查看pid为1的进程，主机中对应的通常是init或systemd。

选择Docker开源版本

版本太旧：功能不全、遗留缺陷较多。

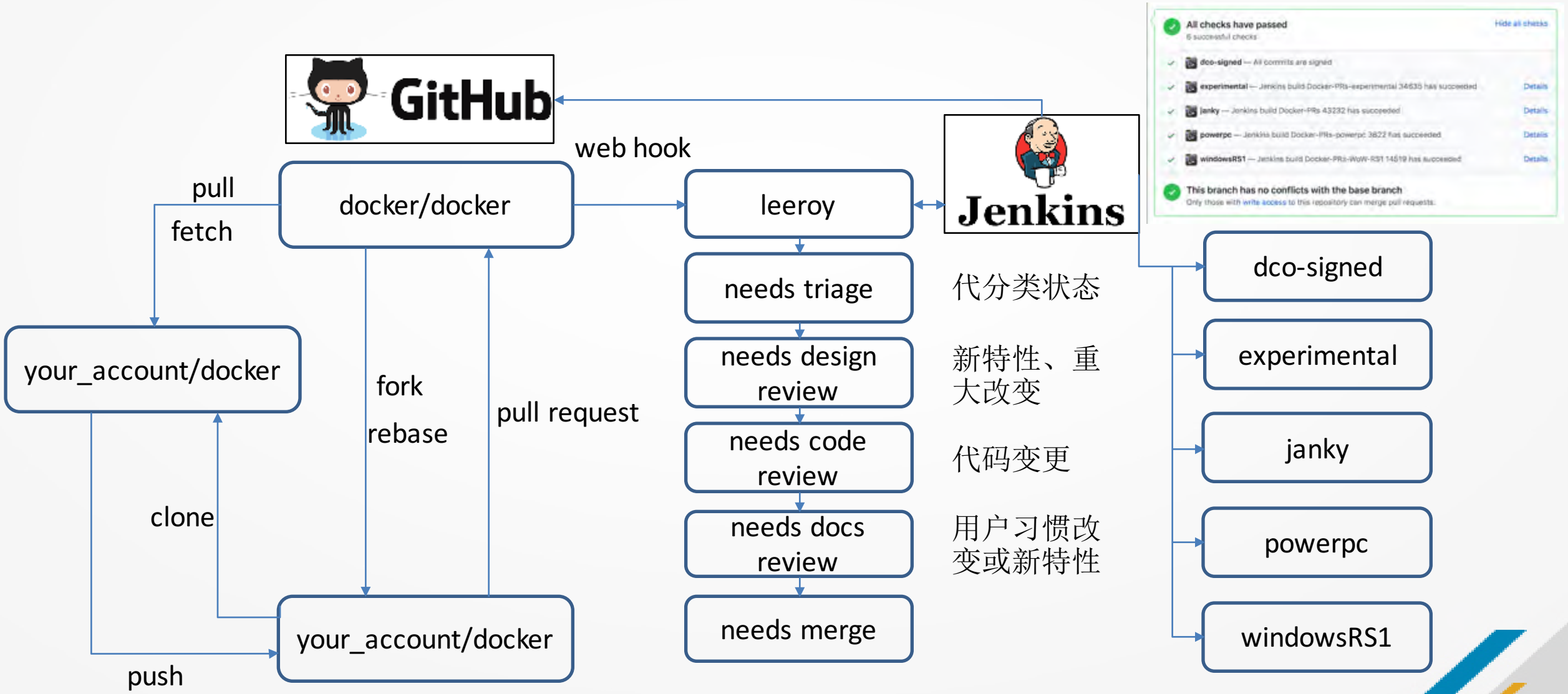
版本太新：当有较多新功能加入时，会带有较多缺陷，很多缺陷是未知的，相当于帮助社区踩雷。版本通常在两三个迭代后才能稳定。

稳定版本：社区缺陷已知，修复和规避方式透明。稳定性可靠性高。

思考

- 哪种类型的测试任务最适合容器化改造？
- 容器化改造的阻力会有哪些？

Docker开源社区探索



Docker开源社区探索

代码提交规范

1. 原子提交，每次commit应该只变更尽可能单一的功能，方便日后审计或回滚。
2. 提交commit信息需要包含姓名和邮件地址，其它信息尽可能详尽。
3. 每个Pull Request尽量保证只包含一个commit，便于代码审核。
4. 先本地执行相关测试，测试通过后再提交pull request
5. 及时解决冲突，git rebase & force push
6. 不要使用自身的master分支作为提交分支。

Docker开源社区探索

- Docker CI

1. 测试用例运行在容器中。
2. 多次测试完全独立，不会相互影响。
3. 不污染宿主机环境。
4. 测试环境搭建简单，只要能运行Docker就能随时随地运行测试。

在开源社区的测试中成长

典型开源测试工具

- 测试管理： TestLink, Testopia
- 缺陷管理：Redmine, Bugzilla, Mantis
- 持续集成：Jenkins, Buildbot
- 功能测试：Selenium, LTP (Linux Test Project)
- 性能测试：lmbench, Sysbench, Iperf, Fio
- 测试框架：JUnit, Autotest
- 测试设计：Xmind, StarUML, UML Designer
- 安全测试：Metasploit, Nessus, AppScan

在开源社区的测试中成长

Kernel开源社区测试能力短板：
Ltp社区：https://github.com/linux-test-project/ltp
社区运作方式：迭代式开发

问题：
文档和测试用例补充严重滞后

例子：user namespace特性
功能合入时间 ：2013年2月18日随内核3.8版本正式发布
测试用例完成时间：2015年5月21日

Kernel社区还维护着LTS的长期稳定版本，即在某一个特定的Kernel版本中不添加新的功能，只修改已有的软件缺陷。
因为测试资源的缺乏，在LTS版本发布前，并不是每个版本都进行源码编译、全量测试执行等测试工作。很多时候验证工作往往只包含源码编译和OS启动冒烟测试。这很可能导致大量软件缺陷遗漏到下游开发者手中。

在开源社区的测试中成长

Docker开源社区测试能力短板：
社区运作方式：测试驱动开发
运作优势：解决了文档和测试用例滞后的问题。

问题：
•测试用例几乎都是开发者提供的，他们更关注开发代码的质量，对于测试代码的质量就显得有点“漠不关心”了。
•用例在设计过程中往往缺少测试思维，使得输出的测试用例缺少边界异常点检查。
•开发者输出的用例几乎都是单点的功能验证，无法覆盖全面的代码路径，更缺少一些专项测试（性能测试、压力测试、长稳测试、安全测试等）。

后果：
在发布的Docker1.12版本中，就出现了软件稳定性差的问题，引发了很多争论，社区又不得不亡羊补牢。

TiD2017

在开源社区的测试中成长

Laura Frank
rheinwein

Unfollow

Block or report user

Codeship
Berlin

Organizations

Overview | Repositories 36 | Stars 6 | Followers 207 | Following 0

Pinned repositories

moby/moby
Moby Project - a collaborative project for the container ecosystem to assemble container-based systems
Go ★ 44.1k 13.1k

docker/libcompose
An experimental go library providing Compose-like functionality
Go ★ 430 120

codeship/documentation
Documentation for Codeship CI & CD service.
CSS ★ 64 77

docker
Forked from moby/moby
Docker - the open-source application container engine
Go

notes-app
A simple notes app written in Rails, with Codeship integrations
Ruby

orchestration-workshop
Forked from jpetazzo/orchestration-workshop
HTML

1,052 contributions in the last year

Jun Jul Aug Sep Oct Nov Dec Jan Feb Mar Apr May
Mon
Wed
Fri

Learn how we count contributions.
Less More

测试工程师参与开源社区的方式

- 确保特性的可测试性
- 添加测试用例
- 书写或补充文档
- 修改软件缺陷或添加新功能
- 测试待发布版本
- 补充专项测试方案(性能测试、压力测试、长稳测试、安全测试等)
- 提交软件缺陷和回答社区中的问题
- 设计开源测试工具

使用开源软件->学习开源软件->贡献开源社区->影响开源社区->构建符合自身利益的开源生态

开源软件问题求助方式与注意事项

- 通过互联网搜索引擎进行查询。
- 通过专业技术网站或论坛求助。
- 通过社区中issue板块或者发送邮件到社区邮件列表进行提问。
  - —如果是报bug，需要满足社区提交bug的规范。不建议报社区版本老版本中存在而最新版本中不存在的bug。
  - —处理好与maintainer的关系，取得其信任。如果不是原则问题，避免与maintainer进行过分争论，沟通语气要委婉。
  - —与maintainer有不同意见时不要挑战其权威，需要用委婉的方式解决分歧。
  - —避免在社区中泄露公司的项目细节。
  - —不要向社区直接提出低级问题，如配置git的方式。没有把握时可以先在公司内部评审，避免低级问题对公司外部形象造成影响。
- 当问题得不到解决时将问题提交给更多的maintainer或社区贡献者。
- 通过社区irc会议进行求助。
- 在外部会议上与maintainer或者专业人士进行讨论。
- 与开展开源项目的公司展开商业合作。

参与开源社区的策略

企业参与开源社区的目的：构建差异化竞争力，实现商业利益最大化。开源不是情怀，不是为社会做贡献，而是实实在在的竞争手段，是竞争白热化以后最常用的竞争手段。

开源不是“把代码贡献出去”的一锤子买卖，每行送出去的代码，都是为了收益。没有背后战略的支撑的开源，不是开源，是开玩笑。

- 核心社区
  - -参与制定标准，增大社区话语权，引导社区向有利于公司的方向发展，制定有利于公司的规范。
  - -做成基金会项目，如OCI项目。
  - -防止社区被竞争对手控制，或被某一公司绑架，如容器引擎项目。
- 重要社区
  - -与外部合作参与社区。
  - -独自参与社区，深入了解趋势，为商业化做准备。
- 非重要社区
  - -跟踪社区发展，如性能测试套。
- 边缘社区
  - -已用为主，不投入，如jenkins。

参与开源社区的策略

领先开源社区半步是先进，领先三步成先烈。

社区影响力

引导社区向自身业务需要的方向发展的能力。例如引导社区投入测试框架的开发。

核心竞争力

领先于社区的能力。例如不开源的测试用例。

社区发布新版本时是否同步？

领先三步

同步：如果社区进行重构，rebase后合入自研特性的成本巨大。

不同步：无法使用社区新版本的特性，将与社区渐行渐远。如果社区进行重构，rebase后合入自研特性的成本巨大。

参与开源社区的策略

思考：领先社区太快，拥有太多自研特性和用例，到底是优势还是负担？

关于Docker的争论还将继续

- Why I don't use Docker much anymore?

https://blog.abevoelker.com/why-i-dont-use-docker-much-anymore/

- Why databases are not for containers?

https://myopsblog.wordpress.com/2017/02/06/why-databases-is-not-for-containers/

引用资料

《Using Docker/Software Containers for Automated Testing》 Pini Reznik
《 DevOps的前世今生》 木环
《OpenSource Software Testing Tools Walkthrough》
《Docker进阶与实战》
《Docker技术入门与实战》
《测试工程能力容器化改造方案》

提问环节

联系我：

孙远 华为中央软件院

研究方向：容器技术、docker、软件测试、自动化测试

Email：sunyuan3@huawei.com