# waypal云平台规划

## 背景

Docker：是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。

持续集成：(Continuous integration)是一种软件开发实践，即团队开发成员经常集成它们的工作，通过每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽早地发现集成错误。

持续部署：（continuous deployment）是通过自动化的构建、测试和部署循环来快速交付高质量的产品。某种程度上代表了一个开发团队工程化的程度，毕竟快速运转的互联网公司人力成本会高于机器，投资机器优化开发流程化相对也提高了人的效率，让 engineering productivity 最大化。

domeOS：DomeOS是基于Docker的企业级私有云一站式运维管理系统。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。

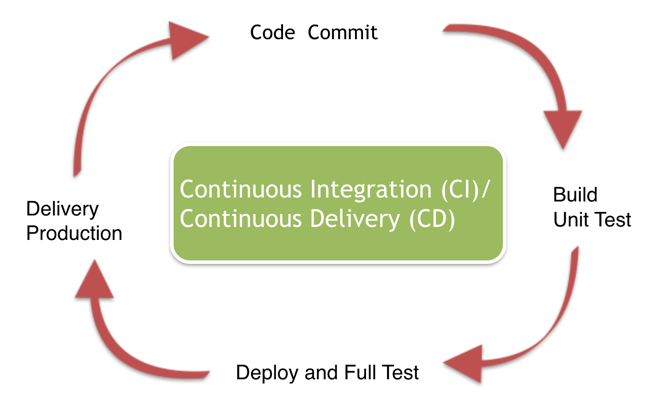
K8S：

阿里云持续交付平台：

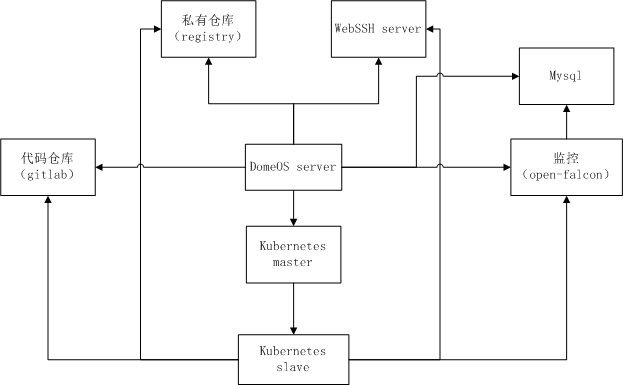
研发过程中的困难

任何一家互联网或者软件公司，随着产品规模的扩大，市场需求的变化，都会逐步的发现产品版本管理混乱，运维人员总是在兜底， 不知道开发/测试/集成/预发布/生产等等环境到底经历过几代运维人员之手，所以环境压根没人敢动。

因为市场永远在变化， 需求一定在变化，人员也在变化，导致了研发过程中遇到的这样那样的问题。 因此，大多数企业都用CI/CD 这个解决方案来应对 ， 如下图：



## 平台概述



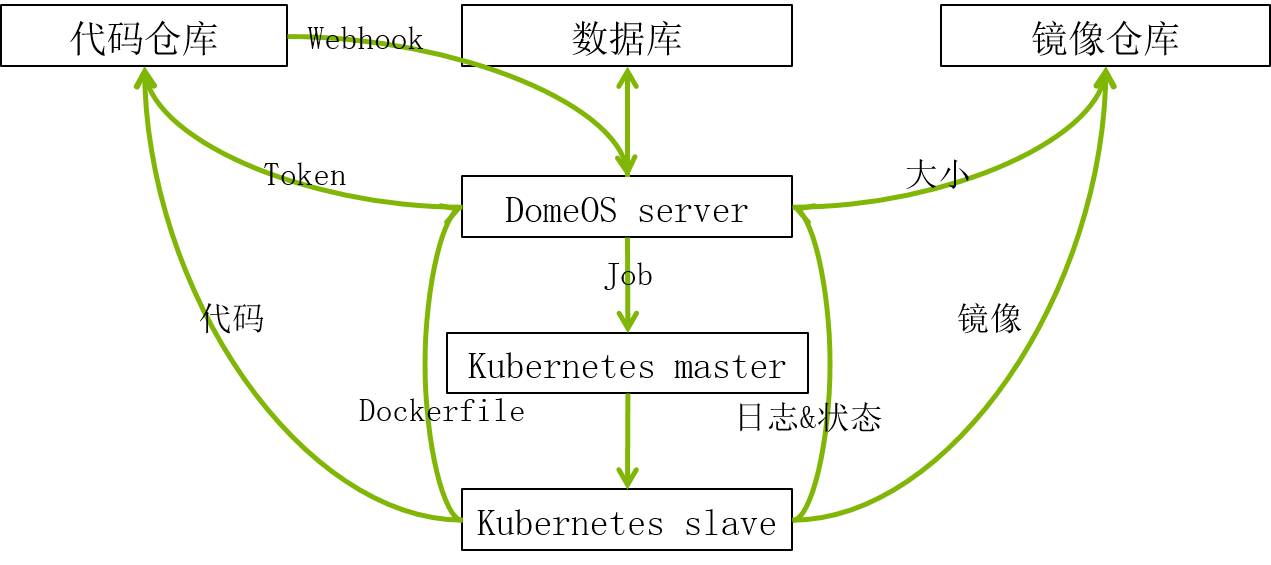
DomeOS Server是整个系统的控制模块，前端和所有逻辑控制均在这部分实现。  
MySQL用来存储项目、部署、用户、监控等数据信息。  
Server可以关联到代码仓库，目前支持GitLab和SVN。  
Registry是Docker提供的开源私有仓库，我们对它进行了改造，可以对接到搜狐云台的存储系统，保证了镜像的安全性和可靠性。  
资源分配、任务调度部分交给Kubernetes来完成。  
监控部分融合了open-falcon和cAdvisor，实现了主机和容器多个维度信息的收集展示。  
另外，为了方便研发及运维同学在部署过程中快速定位问题，加入了WebSSH模块，能够通过网页直接进入到容器内部查看信息。DomeOS支持LDAP登录，方便企业用户使用。

功能模块

DomeOS提供了项目管理、持续集成、部署管理、镜像管理、集群管理、应用商店、用户和组管理以及多层级监控服务。以下是功能模块图：



其中项目管理、集群管理和部署是比较重要的功能。  
  
项目管理中包含了持续集成，设计方案如下：



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、容器的多层级监控。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。

## Docker化交付过程实践

基本思路：

安装好Docker环境

Docker化你的应用运行环境

Docker化你的应用编译，UT环境

Docker化你的应用运行依赖环境

第一步，安装Docker运行环境

第二步，如何将自己的应用运行在Docker容器中

这句话可以翻译为： 如何将我的应用环境通过Dockerfile描述出来？

假如我的应用是一个Java Web 应用，需要Java运行环境和Tomcat 容器 ，那么大概我的环境所需下面这些东西：

某Linux发行版操作系统

基础软件（起码有个能解压缩包的吧）

openjdk 7 && 配置 Java Home 等环境变量

Tomcat 7 && 配置 环境变量

应用包 target.war

应用包 启动参数 JVM

Web Server 指定端口 8080

启动tomcat

转化为成Dockerfile 的语言大致如下：

FROM buildpack-deps:jessie-curl

RUN apt-get update && apt-get install -y unzip openjdk-7-jre-headless=“$JAVA\_DEBIAN\_VERSION”

ENV LANG C.UTF-8

ENV JAVA\_VERSION 7u91

ENV JAVA\_DEBIAN\_VERSION 7u91-2.6.3-1~deb8u1

ENV CATALINA\_HOME /usr/local/tomcat

ENV PATH $CATALINA\_HOME/bin:$PATH

RUN mkdir -p "$CATALINA\_HOME"

WORKDIR $CATALINA\_HOMEENV TOMCAT\_VERSION 7.0.68

ENV TOMCAT\_TGZ\_URL https://xxxx/apache-tomcat-$TOMCAT\_VERSION.tar.gz

RUN set -x \

&& curl -fSL "$TOMCAT\_TGZ\_URL" -o tomcat.tar.gz \

&& curl -fSL "$TOMCAT\_TGZ\_URL.asc" -o tomcat.tar.gz.asc \

&& gpg --batch --verify tomcat.tar.gz.asc tomcat.tar.gz \

&& tar -xvf tomcat.tar.gz --strip-components=1 \

&& rm bin/\*.bat \

&& rm tomcat.tar.gz\*

EXPOSE 8080

CMD ["catalina.sh", "run"]

第三步，用Docker描述我的编译环境

编译/CI环境往往在公司规模越来越大的时候， 变得越来越麻烦， 因为不同语言，不同类型的应用对编译环境的要求都不一样。

编译环境Docker化最大的好处是： 自定义，可扩展，可复制

试想一下， 假如你的应用编译只需要依赖标准的Jdk 1.7 和 Maven 2， 或者你是python应用编译过程其实只是需要安装依赖， 那么你可以跟很多人共用编译镜像。

但假如你的应用是Nodejs ，编译依赖特定的C库， 或者是C++之类的编译环境一定要和运行环境一致等等，那就需要定制自己的编译环境了。

这里我做一个最简单的用于编译java的镜像示例：

编译镜像的Docker file 示例：

FROM registry.aliyuncs.com/acs-sample/centos:7

RUN yum update yum install -y open-jdk-1.7.0\_65-49

COPY build.sh /build.sh

COPY settings.xml /home/apache-maven-2.2.1/conf/

ENTRYPOINT [“./build.sh"]

上述Dockerfile的build.sh示例：

cd /ws ; mvn -e -U clean package -Dmaven.test.skip=true $@

cp target/\*.war docker/ || exit 0

运行方式示例：

git clone git@code.aliyun.com:tangrong.lx/myproject.git ~/myprj ; cd ~/myprj

docker run --rm -v `pwd`:/ws -v ~/.m2/repo:/buf build\_maven:1.0

解释一下这个过程:

我的编译环境需要CentOs7 系统， 安装JDK1.7 ， 然后把maven 的setting （这里主要配置指向其他私有nexus） 和 编译脚本拷贝进去。

编译脚本也很简单，就是maven编译打包命令，并且把最终生成的war拷贝到一个定义好的docker目录下， 这个目录随便定义。

最后是运行方式， 即把源代码挂载到容器里进行编译， 同时可以选择把本地的.m2 缓存到镜像内加快编译速度

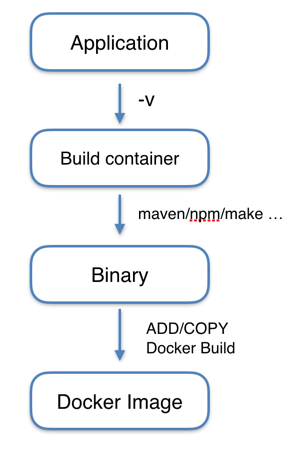
这里提两个小提示，都是经验之谈：

建议： build app 和 build docker image 建议分开进行， 即先进行应用本身的编译，再将输出物拷贝到镜像内（但脚本语言可以例外） 因为：

镜像分层概念导致源码可能泄露：因为DockerImage 每一层都会保存一个版本， 即便是ADD代码进去，编译后再rm掉，也可以通过获取ADD这一层镜像拿到源码，因为镜像是运行在各个环境中，是不应该包含源代码信息的。

镜像最小化原则：编译环境可能需要和运行环境不一样的东西，比如Maven的配置，Nodejs的一些C库的依赖， 都不需要在运行环境中体现，所以本着镜像应该最小化原则，不需要的东西最好都不要放进去，也应该分开进行这个步骤。

所以，整个过程还是分为build app和build docker image 两个过程，类似下面这个简单流程 ：



建议： Docker file 不要放到代码根目录下

避免大量文件传给docker deamon ： docker build会先加载Dockerfile同级目录下所有文件进去，如果有不需要ADD/COPY到镜像里的文件不应该放到Dockerfile目录下， 可以试一下把Dockerfile放到系统/根目录下，这时build 十有八九就会让docker deamon挂掉。

第四步，用Docker描述UT环境

简单思路： 运行Docker 镜像环境，安装测试所需依赖 ， 运行Docker容器， 运行测试命令/脚本

用一个travis-ci 官方的例子来说明容器测试这件事，先看下面一个ruby的镜像：

FROM ubuntu:14.04

MAINTAINER carlad "https://github.com/carlad"

# Install packages for building ruby

RUN apt-get update

RUN apt-get install -y --force-yes build-essential wget git

RUN apt-get install -y --force-yes zlib1g-dev libssl-dev libreadline-dev libyaml-dev libxml2-dev libxslt-dev

RUN apt-get clean

RUN wget -P /root/src http://cache.ruby-lang.org/pub/ruby/2.2/ruby-2.2.2.tar.gz

RUN cd /root/src; tar xvf ruby-2.2.2.tar.gz

RUN cd /root/src/ruby-2.2.2; ./configure; make install

RUN gem update --system

RUN gem install bundler

RUN git clone https://github.com/travis-ci/docker-sinatra /root/sinatra

RUN cd /root/sinatra; bundle install

EXPOSE 4567

简单来说就是标准的一个ruby 镜像， 启动4567端口 。 那么通过这个镜像进行的测试过程如下：

sudo: required

language: ruby

services:

- docker

before\_install:

- docker build -t carlad/sinatra .

- docker run -d -p 127.0.0.1:80:4567 carlad/sinatra /bin/sh -c "cd /root/sinatra; bundle exec foreman start;"

- docker ps -a

- docker run carlad/sinatra /bin/sh -c "cd /root/sinatra; bundle exec rake test"

script:

- bundle exec rake test

这个其实就是大家可以在本地进行的一个过程，在before install部分内可以看到过程是：

先build出运行环境的镜像

运行这个镜像，看看服务能否正常启动

查看容器是否存活（保证容器不是运行一下就挂了退出）

运行测试

第五步，依赖和服务编排

## 计划安排

1. 基础环境domeOS + K8S环境准备；
2. 与gitlab, jenkins, 监控系统进行集成；
3. 开发-测试-预发布-生产环境整合；
4. 项目持续集成/交付的验证；