2017

龙渊docker应用实践

主讲人：王颖

Agenda

01 契机

02 选型

03 效果

04 过程

05 问题

06 展望

契机

01

内部因素

- 标准化
- 效率
- 环境一致
- 隔离性

02

内部因素

- 合作方业务使用了docker

1

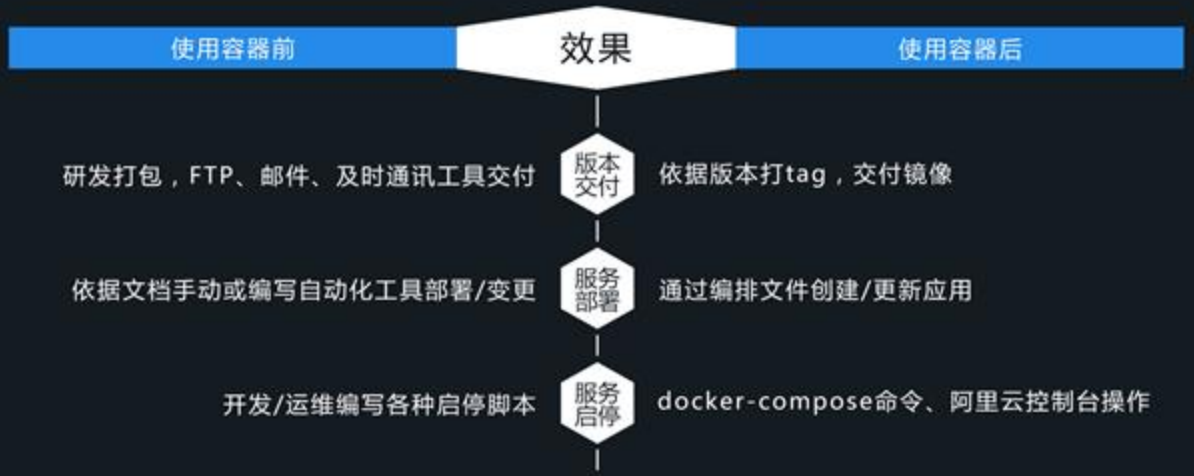
容器使用方式

- 仅作为版本交付手段，把docker容器当作传统的进程进行管理
- 搭建集群，通过服务编排完成业务部署，实现DaaS/IaC

我们当然希望实现第2种方式，但...

技术背景

1. 运维熟悉docker技术原理和使用
2. 运维熟悉docker-compose编排
3. 运维无生产环境docker集群运维经验
4. 开放人员不熟悉docker



2

应用场景

- 百乐直播 live66
- 运维内部支撑系统
- 其他

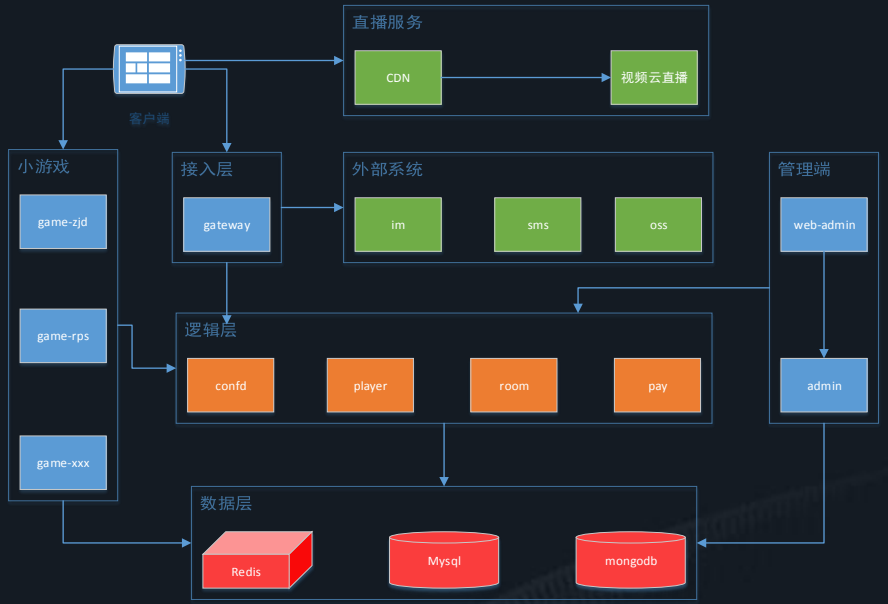
  1.账号服

  2.sdorica (测试阶段，本地集群)

Live66 实施过程

可行性分析

- 无状态服务
- 接入层HTTP协议
- 内部TCP RPC
- 数据库使用连接池
- 小游戏单点，低负载
- Go语言开发，易于部署

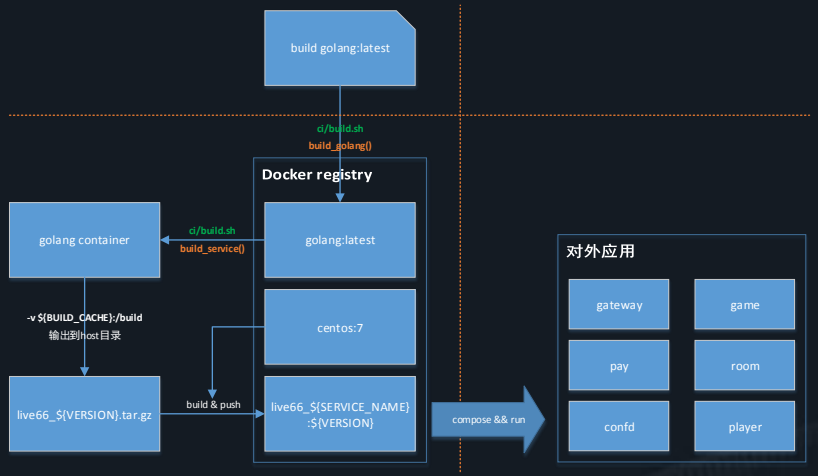


项目规范

- 确保所有依赖均通过包管理进行自动解决
- 配置文件中的配置项均可通过环境变量进行设置
- 确定版本规范，每次新构建需升级版本号
- 约定使用/home/worker/main/entrypoint.sh作为统一的容器启动入口
- 约定镜像中的主执行文件均为/home/worker/main/main
- 约定镜像中的配置文件均为/home/worker/main/conf/app.conf
- 镜像仓库命名规则： <项目代码>_<模块名>, 如live66_gateway

构建及部署过程示意

- 编译环境基础镜像制作
- CI脚本编写
- 使用阿里云镜像仓库推送镜像
- 编写编排文件，完成部署



编译环境基础镜像制作

```
FROM centos:7
MAINTAINER wangying@dragonest.com
RUN yum install -y curl git && mkdir -p /go_path/bin \
  && mkdir -p /root/.ssh && git config --global
url."git@gitlab.ilongyuan.cn:".insteadOf https://gitlab.ilongyuan.cn/
ADD ssh_config /root/.ssh/config
RUN chmod 644 /root/.ssh/config
# current golang: go1.7.4.linux-amd64.tar.gz
ARG GOLANG_INSTALL_PACKAGE
ADD ${GOLANG_INSTALL_PACKAGE} /
ENV GOPATH=/go_path
ENV GOROOT=/go
ENV PATH=$GOROOT/bin:${GOPATH}/bin:${PATH}
```

CI脚本实现细节

```
build_service(){
  local SERVICE_NAME=$1
  local SRC_CACHE=${WORK_DIR}/.go_path_cache/src
  local BUILD_CACHE=${WORK_DIR}/${SERVICE_NAME}/.cache/build
  mkdir -p ${SRC_CACHE} ${BUILD_CACHE}
  # 编译
  docker run -it --rm \
    -e VERSION=${VERSION} \
    -v ${WORK_DIR}/.live66_id_rsa:/root/.ssh/id_rsa \
    -v ${SRC_CACHE}:/go_path/src \
    -v ${BUILD_CACHE}:/build \
    -v ${WORK_DIR}/${SERVICE_NAME}/build.sh:/build.sh \
    ${GOLANG_IMG}:${GOLANG_VERSION} /build.sh
  ......
```

CI脚本实现细节

```
#!/bin/bash
set -e
SERVICE_NAME=confd
PKG_NAME=gitlab.ilongyuan.cn/live66/${SERVICE_NAME}
PKG_PATH=${GOPATH}/src/${PKG_NAME}
go get -u ${PKG_NAME}
cd ${PKG_PATH}
go build -o main
tar czf  /build/${SERVICE_NAME}_${VERSION}.tar.gz -C
$PKG_PATH main conf entrypoint.sh
```

```
FROM centos:7
RUN useradd -m worker && mkdir -p /data/log &&
chown -R worker.worker /data/log
RUN mkdir /home/worker/main
ARG VERSION
ARG SERVICE_NAME
ADD .cache/build/${SERVICE_NAME}_${VERSION}.tar.gz
/home/worker/main
RUN chown -R worker.worker /home/worker/main &&
chmod +x /home/worker/main/entrypoint.sh
USER worker
ENTRYPOINT ["/home/worker/main/entrypoint.sh"]
```

CI脚本实现细节

```
build_service(){
    ……
    # 构建镜像并推送至阿里云镜像仓库
    docker build --build-arg VERSION=${VERSION} --build-arg
SERVICE_NAME=${SERVICE_NAME} -t
${REPO_BASE}/${PROJECT_NAME}_${SERVICE_NAME}:${VERSION}
${SERVICE_NAME}  && \
      docker push ${REPO_BASE}
/${PROJECT_NAME}_${SERVICE_NAME}:${VERSION}
}
```

服务编排

- docker –compose
- swarm
- 阿里云容器服务扩展
- 未来：swarm mode?

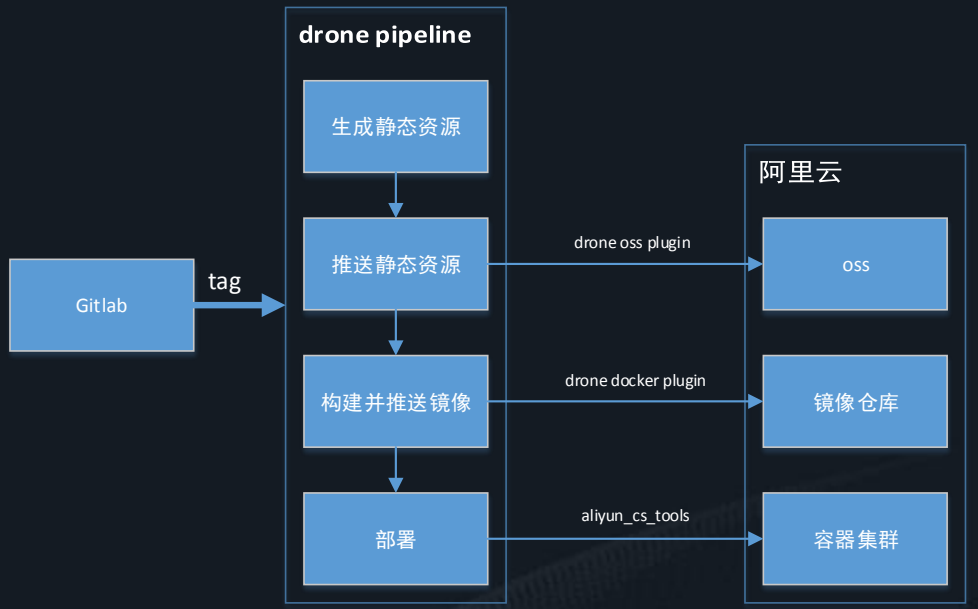
```yaml
version: '2.1'
services:
    confd_6000:
      image: registry.cn-hangzhou.aliyuncs.com/ly_release/live66_confd:${CONFD_VERSION}
      hostname: confd
      restart: on-failure:10
      environment:
        - 'constraint:confd==1'
        - LIVE66_STATIS_APPID=[redacted]
        - LIVE66_STATIS_HOST=https://dataserver.ilongyuan.com.cn
        - LIVE66_STATIS_PATH=
        - LIVE66_REDIS_AUTH=[redacted]
        - LIVE66_MYSQL_USER=live66
        - LIVE66_MYSQL_PASS=[redacted]
      labels:
        aliyun.probe.url: tcp://container:16666
        aliyun.scale: '1'
        aliyun.rolling_updates: 'true'
      volumes:
        - live66_oss:/home/worker/confd/oss
      links:
        - live66-mysql
        - live66-redis
      ports:
        - 6000:16666

    confd_6001:
      image: registry.cn-hangzhou.aliyuncs.com/ly_release/live66_confd:${CONFD_VERSION}
      hostname: confd
      restart: on-failure:10
```

运维内部支撑系统应用情况

- 项目背景
  - cmdb和告警消息发送服务
  - 基于Django开发
  - 静态资源存放于阿里云oss
  - 源码存放于内网gitlab
  - 运维开发，内部使用，随时更新
- 应用现状
  - 使用**drone**实现持续部署

持续部署实现示意

- 编写.drone.yml, 定义pipeline
- 编写相关drone插件
- 代码提交至gitlab后打上tag
- hook tag事件， 执行pipeline



.drone.yml

```
pipeline:
  build_static:
    image: registry.cn-hangzhou.aliyuncs.com/ly_ops/django:cmdb_api
    commands:
      - http_proxy=${HTTP_PROXY} https_proxy=${HTTPS_PROXY} pip3 install -r
requirements.txt
      - python3 manage.py collectstatic --noinput
    environment:
      - STATIC_VERSION=${DRONE_TAG##v}
    when:
      event: tag

......
```

.drone.yml

```
.....
  # ossutil cp file_url oss://bucket[/prefix] [-r]
  # drone secret add --skip-verify ops/cmdb_api  ALIYUN_ACCESS_KEY_ID
<your id>
  push_static:
    image: registry.cn-hangzhou.aliyuncs.com/ly_ops/drone_aliyun_oss
    access_key_id: ${ALIYUN_ACCESS_KEY_ID}
    access_key_secret: ${ALIYUN_ACCESS_KEY_SECRET}
    endpoint: oss-cn-shanghai.aliyuncs.com
    bucket: lyops
    target: static_root/${DRONE_TAG##v}
    dest: cmdb-api/static/${DRONE_TAG##v}
    when:
      event: tag
```

.drone.yml

```
 ......
 build_image:
    image: plugins/docker
    registry: registry.cn-hangzhou.aliyuncs.com
    repo: registry.cn-hangzhou.aliyuncs.com/ly_ops/cmdb_api
    username: ${DOCKER_USER}
    # drone secret add --skip-verify ops/cmdb_api DOCKER_PASSWORD <your
passwd>
    password: ${DOCKER_PASSWORD}
    dockerfile: Dockerfile
when:
      event: tag
    tags:
      - latest
      - ${DRONE_TAG##v}
```

.drone.yml

```
deploy:
    image: python:2.7
    environment:
      - ALIYUN_ACCESS_KEY_ID=${ALIYUN_ACCESS_KEY_ID}
      - ALIYUN_ACCESS_KEY_SECRET=${ALIYUN_ACCESS_KEY_SECRET}
      - REGION=cn-shanghai
      - CLUSTER_ID=${CLUSTER_ID}
      - PROJECT_NAME=cmdb-api
      - VERSION=${DRONE_TAG##v}
      - PGCRYPTO_DEFAULT_KEY=${PGCRYPTO_DEFAULT_KEY}
commands:
     - pip install aliyun_cs_tools
     - python tools/deploy/deploy.py
    when:
      event: tag
```

Tips

- 不要在源码、镜像中保存密钥、密码等私密信息
- 容器内业务程序尽量使用普通用户运行
- 使用volume缓存/共享ci pipeline每个stage中产生的数据
- 不要在Dockerfile中pull代码，除非禁用缓存
- 尽量减少Dockerfile中的命令条数，使用 && 链接多个命令
- 可考虑使用ARG传递http_proxy等build过程临时环境变量，以加速构建
- 对外的生产环境尽量使用docker compose原生的编排，谨慎使用阿里云提供的扩展功能

趟过的一些坑

- 自身问题
  - 测试本地集群的时候，主机名未修改导致overlay网络不通
  - links太多，导致自动生成的一些环境变量过多，触发docker自身panic
  - entrypoint位于volume中，导致服务更新失败
- 集群bug(已修复)
  - 节点docker版本不兼容
  - 集群agent bug导致节点异常
  - 阿里云服务发现服务异常，导致变更后集群状态不能马上同步
- 功能性限制
  - 仅依赖DNS，无法真正实现负载均衡；
  - 使用SLB后，多个有访问关系的服务无法部署在同一个ecs后端服务器

总结

- docker的应用，确实简化了业务部署，提升了运维工作效率
- 使用阿里云容器服务，使得我们有精力专注于docker应用本身，加快了docker技术在公司内部的推广
- 在使用阿里云容器服务的过程中，也加深了我们对docker集群本身的理解，为自建集群提供了一定的参考

展望

- drone的应用推广
- swarm mode模式的集群应用
- 更多业务中应用docker
- 自建docker集群

FAQ

Thanks

欢迎志同道合的朋友加入龙渊