Docker在前端业务的应用

黄小璐

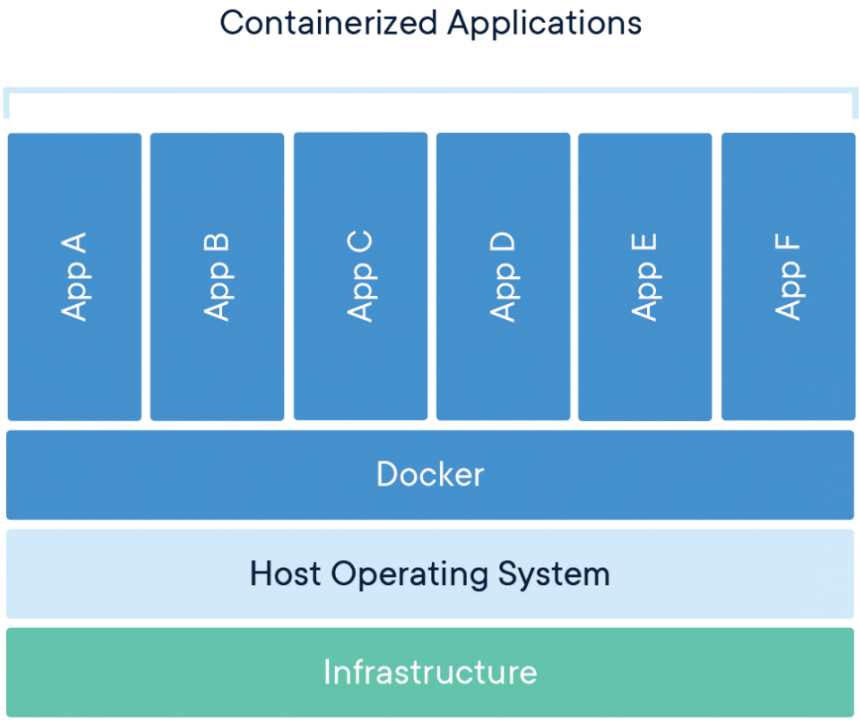
目录

Docker简介

镜像、容器、Dockerfile……

Docker

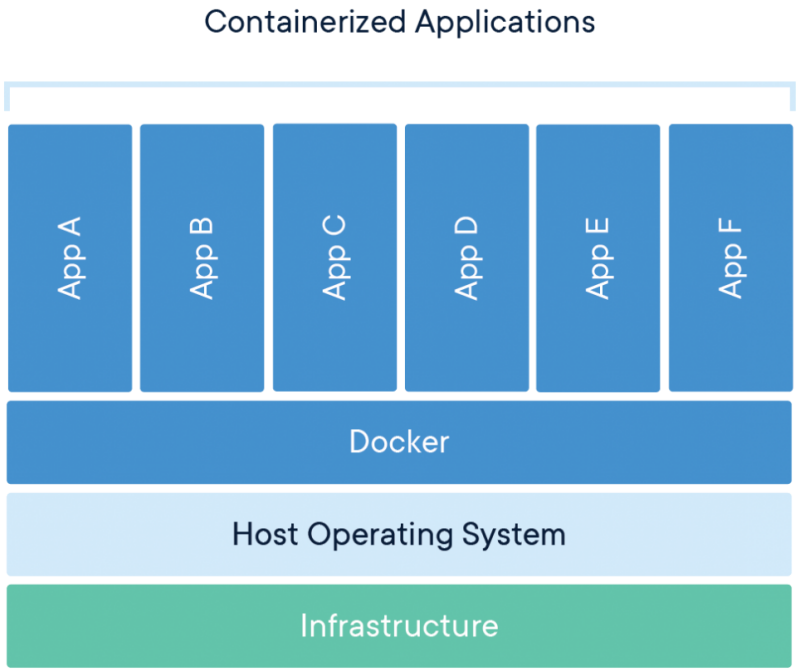
开源的应用容器引擎



容器

容器的本质是进程（“实例”）

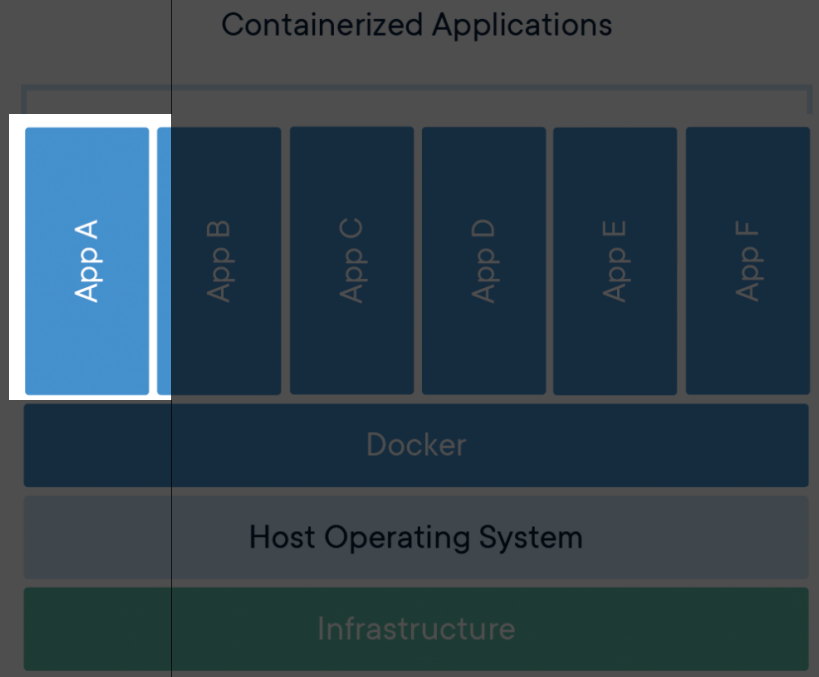
- 标准化：行业标准，可移植
- 轻量：共享宿主机的OS内核
- 安全：相互隔离



容器

容器的本质是进程（“实例”）

- 标准化：行业标准，可移植
- 轻量：共享宿主机的OS内核
- 安全：相互隔离



镜像

容器的模板（“类”、“快照”）

构建方式：
1. 交互式。在运行的容器中，修改，保存成新的镜像。
2. Dockerfile。将交互式的命令用一条条指令表示。

Dockerfile

记录创建镜像的每条指令。
目的：方便修改和自动执行。

```
FROM alpine-node-yarn-git:v1.0.0

WORKDIR /workspace
COPY ./package.json ./yarn.lock ./.yarnrc /workspace/

RUN yarn --verbose
```

docker build命令

构建镜像

```
docker build -t image:tag -f ./Dockerfile .
```

*构建上下文
全部拷贝到docker deamon，故常搭配.dockerignore使用

docker build命令

构建镜像

```
docker build -t image:tag -f ./Dockerfile .
```

*构建上下文
全部拷贝到docker deamon，故常搭配.dockerignore使用

docker build命令

构建镜像

```
docker build -t image:tag -f ./Dockerfile .
```

*构建上下文
全部拷贝到docker deamon，故常搭配.dockerignore使用

docker build命令

构建镜像

```
docker build -t image:tag -f ./Dockerfile .
```

*构建上下文
全部拷贝到docker deamon，故常搭配.dockerignore使用

docker run命令

启动容器

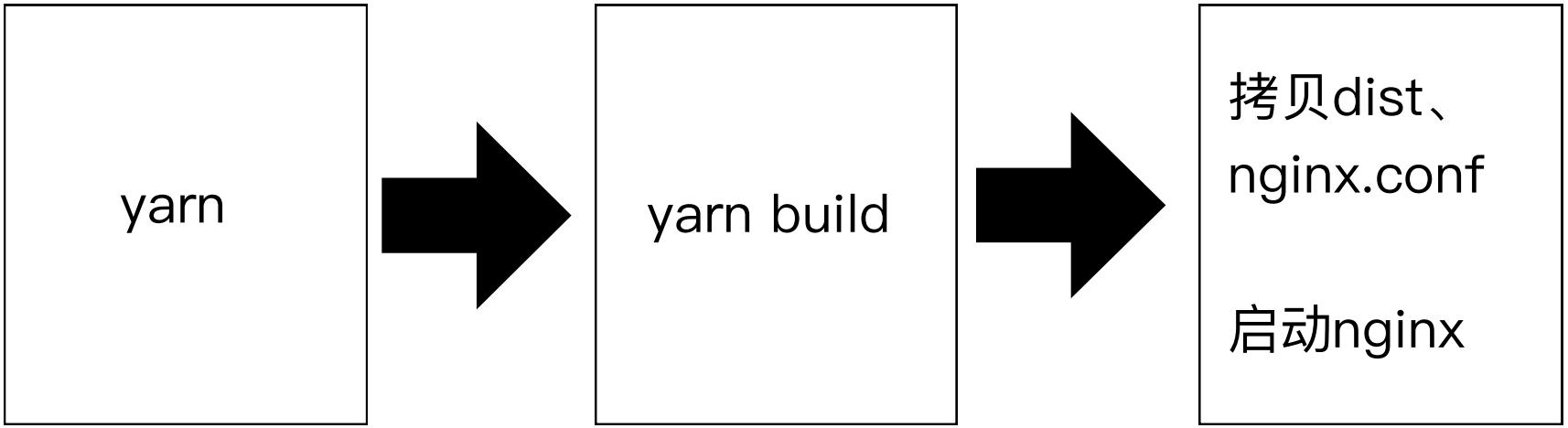
```
docker run -p 8360:80 \
-v /home/q/dashboard/nginx/site.conf:/etc/nginx/conf.d/default.conf image:tag
```

02

Docker在前端业务的应用

部署流程、Dockerfile的编写、自动化

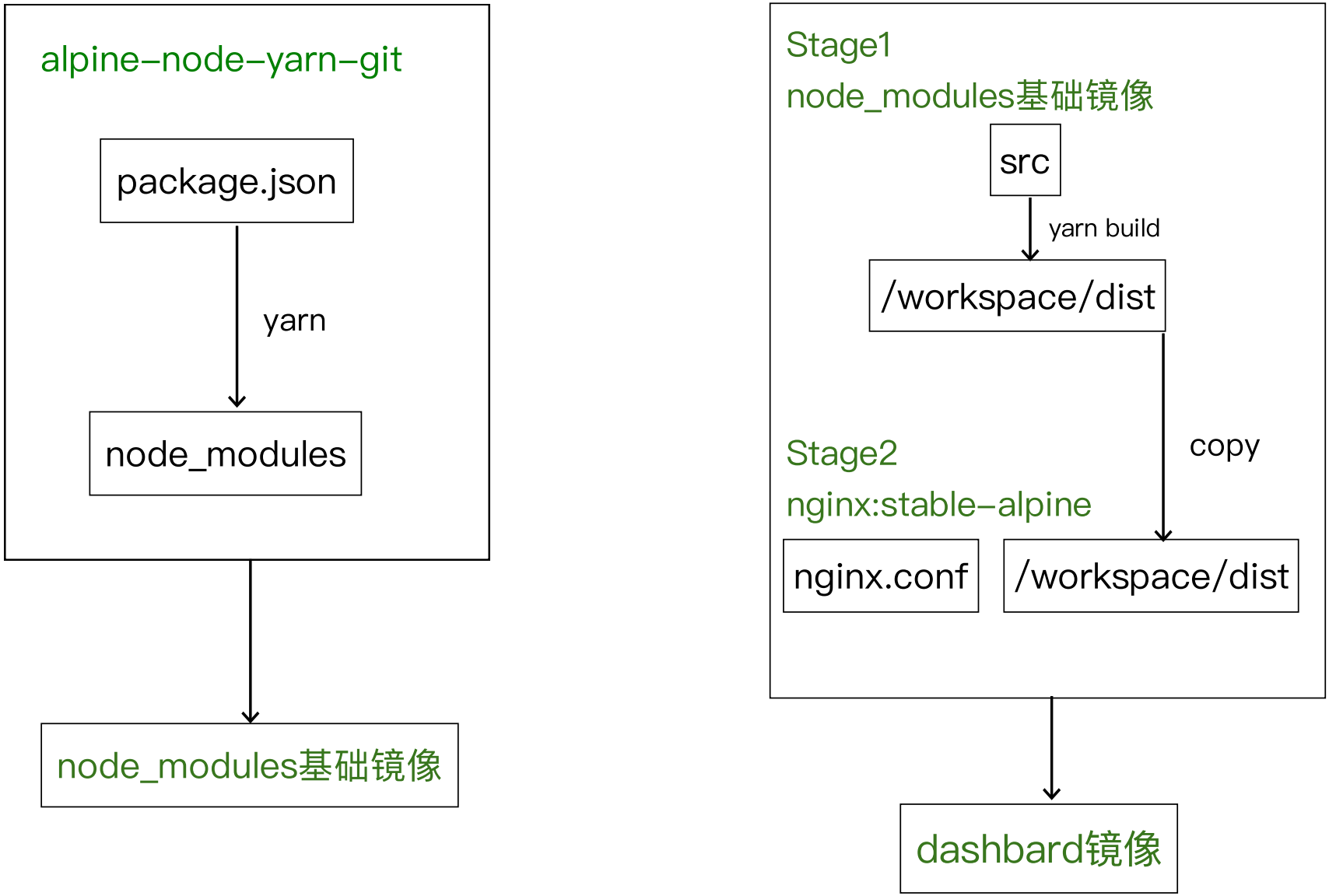
问：前端打包部署总共分几步?


yarn
yarn build
拷贝dist、
nginx.conf
启动nginx

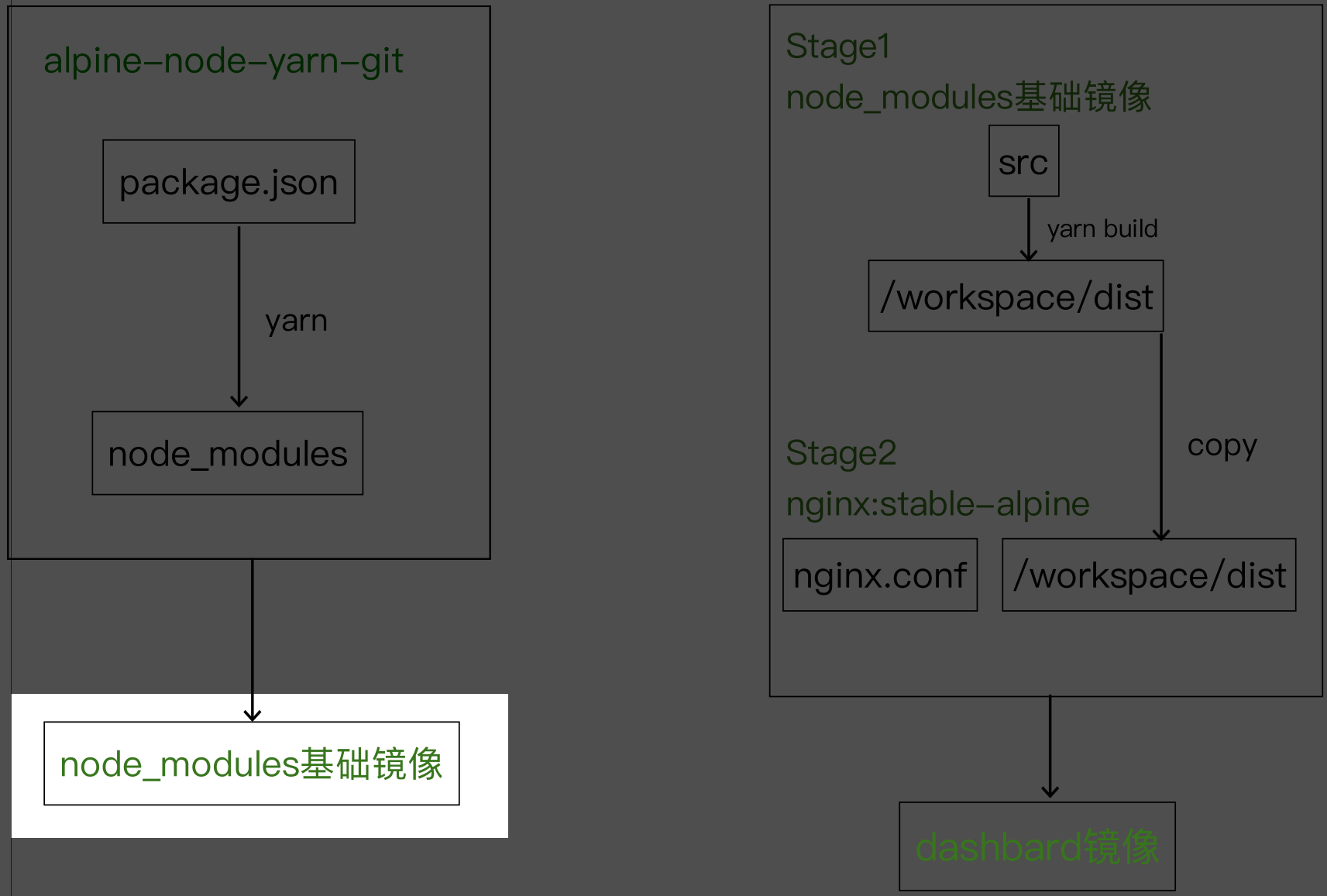
对应的Docker策略

1. 镜像拆分
2. 多阶段构建

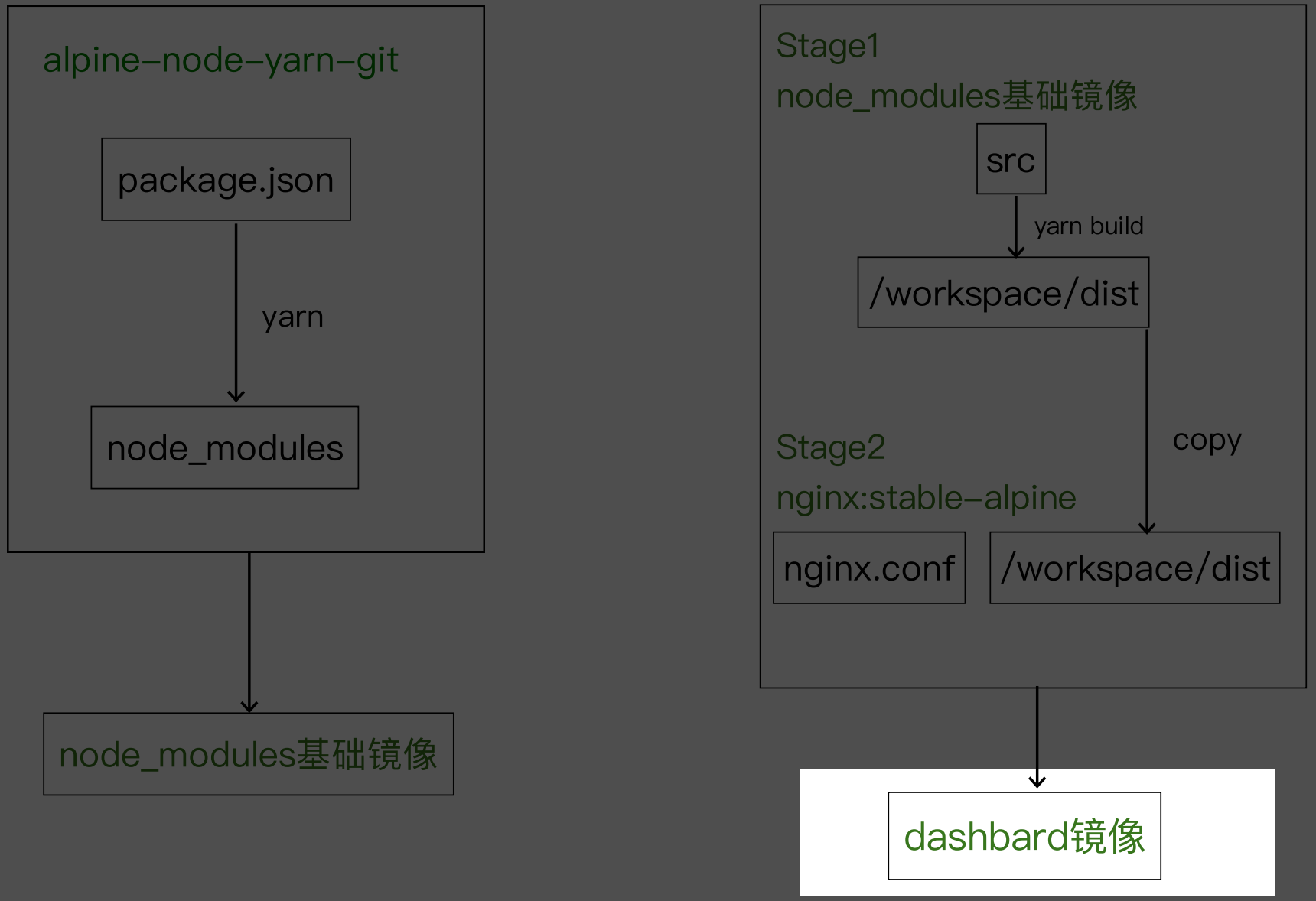
镜像拆分


alpine-node-yarn-git
package.json
yarn
node_modules
node_modules基础镜像
Stage1
node_modules基础镜像
src
yarn build
/workspace/dist
copy
Stage2
nginx:stable-alpine
nginx.conf
/workspace/dist
dashbard镜像

镜像拆分


alpine-node-yarn-git
package.json
yarn
node_modules
node_modules基础镜像
Stage1
node_modules基础镜像
src
yarn build
/workspace/dist
copy
Stage2
nginx:stable-alpine
nginx.conf
/workspace/dist
dashbard镜像

镜像拆分

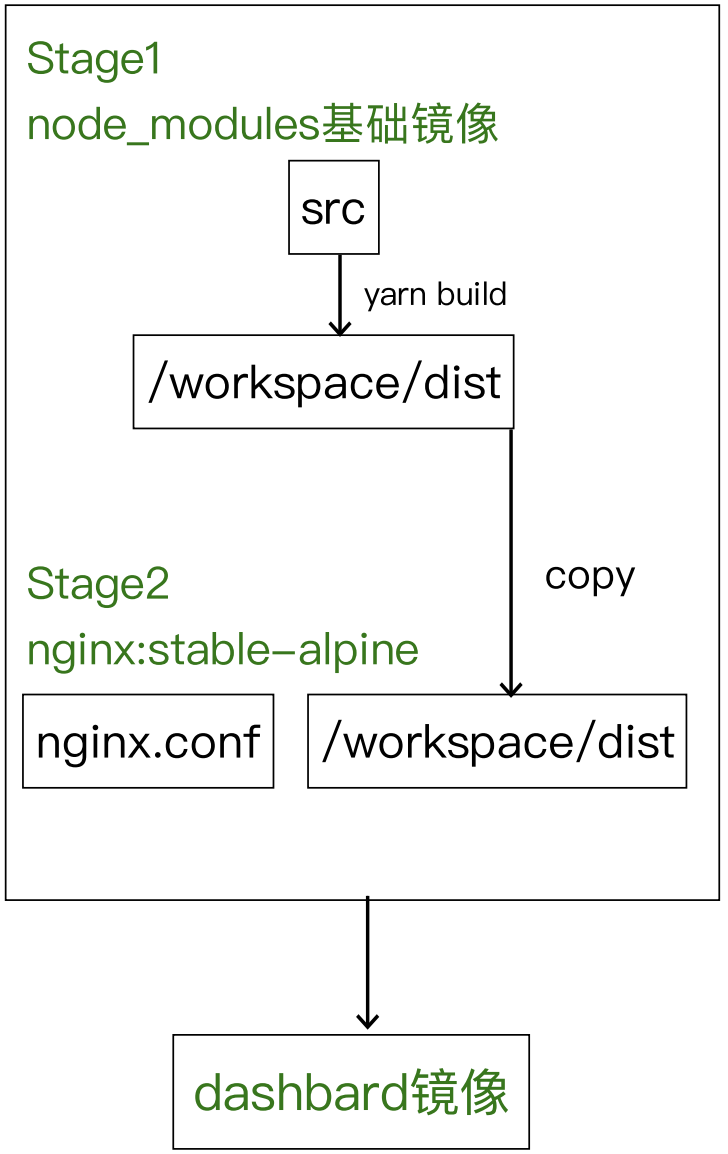

alpine-node-yarn-git
package.json
yarn
node_modules
node_modules基础镜像
Stage1
node_modules基础镜像
src
yarn build
/workspace/dist
copy
Stage2
nginx:stable-alpine
nginx.conf
/workspace/dist
dashbard镜像

多阶段构建（multi-stage builds）

选择性地将前一阶段的产物拷贝到
下一阶段。

目的：

- Dockerfile的逻辑更清晰
- 镜像体积更小

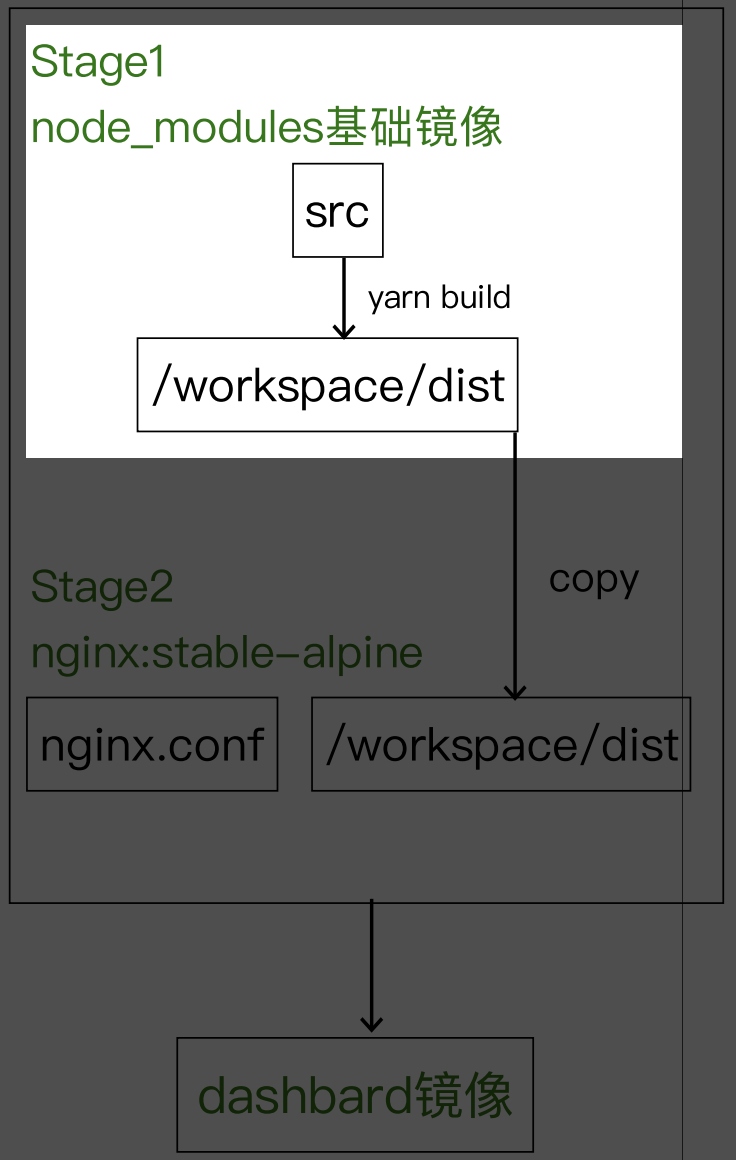


多阶段构建（multi-stage builds）

选择性地将前一阶段的产物拷贝到
下一阶段。

目的：

- Dockerfile的逻辑更清晰
- 镜像体积更小

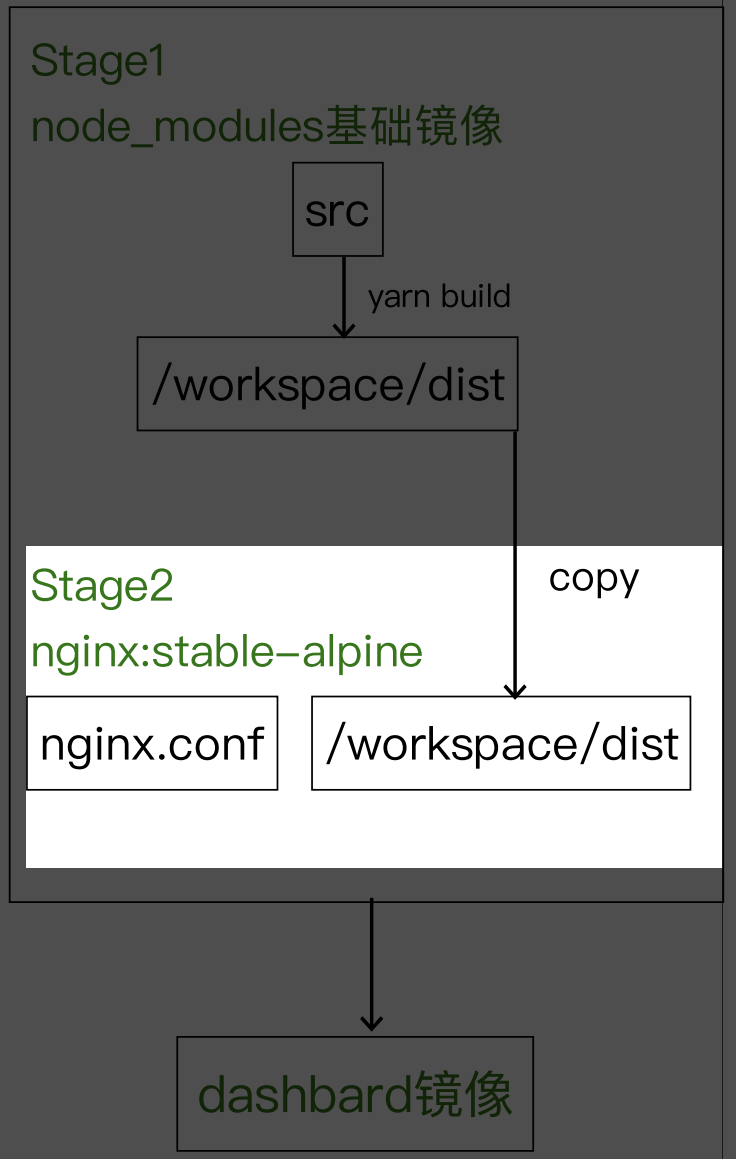


多阶段构建（multi-stage builds）

选择性地将前一阶段的产物拷贝到
下一阶段。

目的：

- Dockerfile的逻辑更清晰
- 镜像体积更小

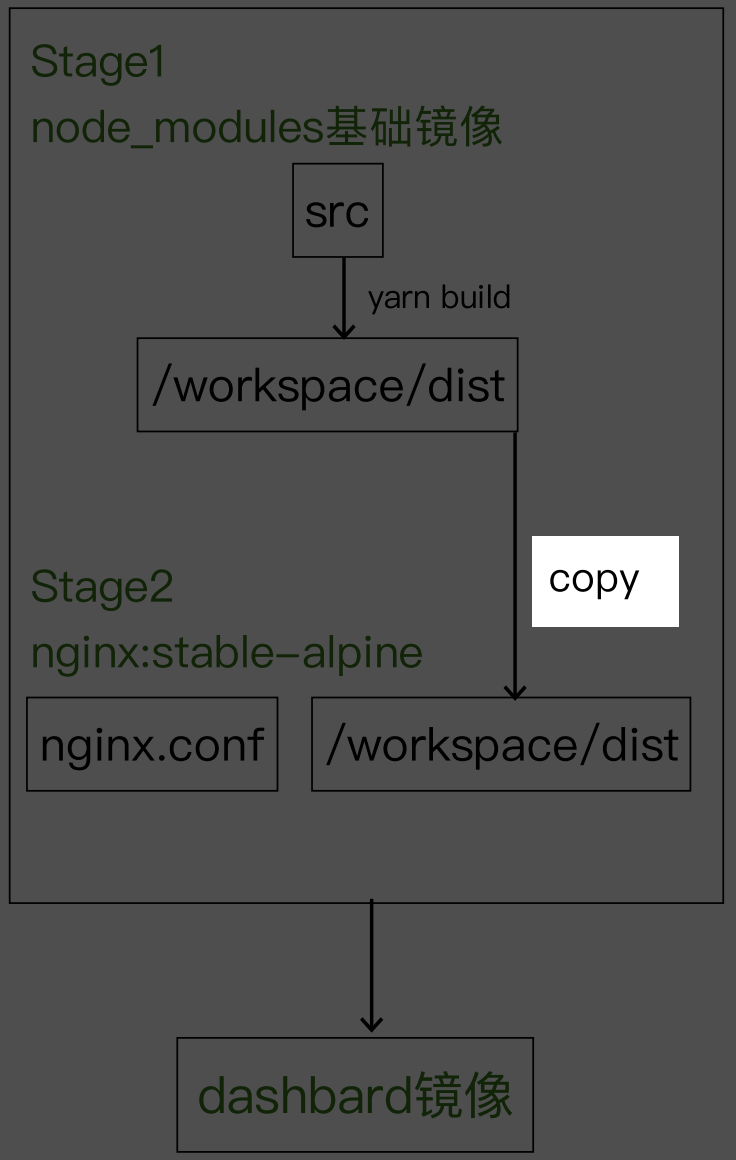


多阶段构建（multi-stage builds）

选择性地将前一阶段的产物拷贝到
下一阶段。

目的：

- Dockerfile的逻辑更清晰
- 镜像体积更小



多阶段构建

```
ARG NPM_PKGS_TAG=latest
ARG NPM_PKGS_IMAGE=dashboard-npm-pkgs
FROM ${NPM_PKGS_IMAGE}:${NPM_PKGS_TAG} as builder

COPY ./ ./
RUN yarn build

# stage nginx
FROM nginx:1.18-alpine

WORKDIR /workspace/dist
RUN mkdir /etc/nginx/logs
ADD ./docker/config/test/nginx.conf /etc/nginx/nginx.conf
ADD ./docker/config/test/site.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /workspace/dist /workspace/dist
```

多阶段构建

```
ARG NPM_PKGS_TAG=latest
ARG NPM_PKGS_IMAGE=dashboard-npm-pkgs
FROM ${NPM_PKGS_IMAGE}:${NPM_PKGS_TAG} as builder

COPY ./ ./
RUN yarn build

# stage nginx
FROM nginx:1.18-alpine

WORKDIR /workspace/dist
RUN mkdir /etc/nginx/logs
ADD ./docker/config/test/nginx.conf /etc/nginx/nginx.conf
ADD ./docker/config/test/site.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /workspace/dist /workspace/dist
```

流程自动化

- 编写Dockerfiles
- scripts构建镜像
- Jenkins执行构建+部署任务

scripts构建镜像

package.json

```
"scripts": {
  "docker-hub:npm-pkg": "sh docker/scripts/build-npm-pkg.sh",
  "docker-hub:build:production": "sh docker/scripts/build-production.sh",
  "docker-hub:build:test": "sh docker/scripts/build-test.sh"
}
```

scripts构建基础镜像

```
#!/bin/bash
set -e
BASEDIR=`cd $(dirname $0); pwd -P`
. $BASEDIR/docker-hub-config.sh --source-only

IMAGE="$REGISTRY/$ENV/$NAMESPACE/$IMAGE_NPM_PKGS_NAME"

DOCKERFILE_PATH="$BASEDIR/../dockerfiles/node_modules.dockerfile"
# package deps的修改时间
PKG_LAST_MODIFY_TIME="$(git log -1 --pretty=format:"%at" -- ./yarn.lock)"

# 构建新镜像
TAG="$PKG_LAST_MODIFY_TIME"
docker build -t $IMAGE:$TAG -f $DOCKERFILE_PATH $PROCESS_DIR

# 推送到docker服务器
docker push $IMAGE:$TAG

echo "已经更新基础镜像"
```

scripts构建基础镜像

```
#!/bin/bash
set -e
BASEDIR=`cd $(dirname $0); pwd -P`
. $BASEDIR/docker-hub-config.sh --source-only

IMAGE="$REGISTRY/$ENV/$NAMESPACE/$IMAGE_NPM_PKGS_NAME"

DOCKERFILE_PATH="$BASEDIR/../dockerfiles/node_modules.dockerfile"
# package deps的修改时间
PKG_LAST_MODIFY_TIME="$(git log -1 --pretty=format:"%at" -- ./yarn.lock)"

# 构建新镜像
TAG="$PKG_LAST_MODIFY_TIME"
docker build -t $IMAGE:$TAG -f $DOCKERFILE_PATH $PROCESS_DIR

# 推送到docker服务器
docker push $IMAGE:$TAG

echo "已经更新基础镜像"
```

scripts构建业务镜像

```
#!/bin/bash
set -e
BASEDIR=`cd $(dirname $0); pwd -P`
. $BASEDIR/docker-hub-config.sh --source-only

IMAGE="$REGISTRY/$ENV/$NAMESPACE/$IMAGE_DASHBOARD_NAME"
IMAGE2="$REGISTRY/$ENV/$NAMESPACE/$IMAGE_NPM_PKGS_NAME"

NPM_PKGS_TAG="$PKG_LAST_MODIFY_TIME"

echo "正在拉取基础镜像 ..."
echo $IMAGE2:$NPM_PKGS_TAG
docker pull $IMAGE2:$NPM_PKGS_TAG

# 如果拉不到会报错，终止。

echo "正在构建镜像 ..."
# 带上基础镜像的tag
docker build --no-cache --build-arg CENV=production \
--build-arg NPM_PKGS_TAG=$NPM_PKGS_TAG \
-t $IMAGE:$TAG -f $BASEDIR/../dockerfiles/dashboard.dockerfile $PROCESS_DIR
```

Jenkins执行构建+部署任务

Jenkins执行构建+部署任务

前端基础组件库 ▸ honghan-vision-doc ▸

General | 源码管理 | 构建触发器 | 构建环境 | **构建** | 构建后操作

Execute shell

命令

```
PROJECT_LOCATION="/home/hhzl/vision/"
PORT=80
HOST=$HOST
PS_NAME="honghan-vision-$PORT"
DASHBOARD_NAME="honghan-vision:$TAG.tar"
DASHBOARD_IMAGE="honghan-vision:$TAG"

echo "=========================Jenkins机器======================================="


#打包前端镜像
yarn docker-hub:build:test
docker save  $DASHBOARD_IMAGE -o $DASHBOARD_NAME

rsync ./$DASHBOARD_NAME  -avz  root@$HOST:$PROJECT_LOCATION

rm -rf $DASHBOARD_NAME
docker image  ls $DASHBOARD_IMAGE -q |awk {print$1} |xargs docker rmi -f

ssh -tt root@$HOST bash -c "
    echo "=========================机器: $HOST======================================="
    sleep 1
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker stop
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker rm -f
    docker rmi  $DASHBOARD_IMAGE
    sleep 3

    cd $PROJECT_LOCATION
    docker load -i $DASHBOARD_NAME
    rm -rf $DASHBOARD_NAME

    docker run -d -i -t -p $PORT:8360 --name=$PS_NAME $DASHBOARD_IMAGE
    exit
  "
```

Jenkins执行构建+部署任务

前端基础组件库 ▸ honghan-vision-doc ▸

General | 源码管理 | 构建触发器 | 构建环境 | **构建** | 构建后操作

Execute shell

命令

```
PROJECT_LOCATION="/home/hhzl/vision/"
PORT=80
HOST=$HOST
PS_NAME="honghan-vision-$PORT"
DASHBOARD_NAME="honghan-vision:$TAG.tar"
DASHBOARD_IMAGE="honghan-vision:$TAG"

echo "=========================Jenkins机器======================================="


#打包前端镜像
yarn docker-hub:build:test
docker save  $DASHBOARD_IMAGE -o $DASHBOARD_NAME

rsync ./$DASHBOARD_NAME  -avz  root@$HOST:$PROJECT_LOCATION

rm -rf $DASHBOARD_NAME
docker image  ls $DASHBOARD_IMAGE -q |awk {print$1} |xargs docker rmi -f

ssh -tt root@$HOST bash -c "
    echo "=========================机器: $HOST======================================="
    sleep 1
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker stop
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker rm -f
    docker rmi  $DASHBOARD_IMAGE
    sleep 3

    cd $PROJECT_LOCATION
    docker load -i $DASHBOARD_NAME
    rm -rf $DASHBOARD_NAME

    docker run -d -i -t -p $PORT:8360 --name=$PS_NAME $DASHBOARD_IMAGE
    exit
  "
```

Jenkins执行构建+部署任务

前端基础组件库 ▸ honghan-vision-doc ▸

General | 源码管理 | 构建触发器 | 构建环境 | **构建** | 构建后操作

Execute shell

命令

```
PROJECT_LOCATION="/home/hhzl/vision/"
PORT=80
HOST=$HOST
PS_NAME="honghan-vision-$PORT"
DASHBOARD_NAME="honghan-vision:$TAG.tar"
DASHBOARD_IMAGE="honghan-vision:$TAG"

echo "=========================Jenkins机器======================================="


#打包前端镜像
yarn docker-hub:build:test
docker save  $DASHBOARD_IMAGE -o $DASHBOARD_NAME

rsync ./$DASHBOARD_NAME  -avz  root@$HOST:$PROJECT_LOCATION

rm -rf $DASHBOARD_NAME
docker image  ls $DASHBOARD_IMAGE -q |awk {print$1} |xargs docker rmi -f

ssh -tt root@$HOST bash -c "
    echo "=========================机器: $HOST======================================="
    sleep 1
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker stop
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker rm -f
    docker rmi  $DASHBOARD_IMAGE
    sleep 3

    cd $PROJECT_LOCATION
    docker load -i $DASHBOARD_NAME
    rm -rf $DASHBOARD_NAME

    docker run -d -i -t -p $PORT:8360 --name=$PS_NAME $DASHBOARD_IMAGE
    exit
  "
```

Jenkins执行构建+部署任务

前端基础组件库 ▸ honghan-vision-doc ▸

General | 源码管理 | 构建触发器 | 构建环境 | **构建** | 构建后操作

Execute shell

命令

```
PROJECT_LOCATION="/home/hhzl/vision/"
PORT=80
HOST=$HOST
PS_NAME="honghan-vision-$PORT"
DASHBOARD_NAME="honghan-vision:$TAG.tar"
DASHBOARD_IMAGE="honghan-vision:$TAG"

echo "=========================Jenkins机器======================================="


#打包前端镜像
yarn docker-hub:build:test
docker save  $DASHBOARD_IMAGE -o $DASHBOARD_NAME

rsync ./$DASHBOARD_NAME  -avz  root@$HOST:$PROJECT_LOCATION

rm -rf $DASHBOARD_NAME
docker image  ls $DASHBOARD_IMAGE -q |awk {print$1} |xargs docker rmi -f

ssh -tt root@$HOST bash -c "
    echo "=========================机器: $HOST======================================="
    sleep 1
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker stop
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker rm -f
    docker rmi  $DASHBOARD_IMAGE
    sleep 3

    cd $PROJECT_LOCATION
    docker load -i $DASHBOARD_NAME
    rm -rf $DASHBOARD_NAME

    docker run -d -i -t -p $PORT:8360 --name=$PS_NAME $DASHBOARD_IMAGE
    exit
  "
```

Jenkins执行构建+部署任务

前端基础组件库 ▸ honghan-vision-doc ▸

General | 源码管理 | 构建触发器 | 构建环境 | **构建** | 构建后操作

Execute shell

命令

```
PROJECT_LOCATION="/home/hhzl/vision/"
PORT=80
HOST=$HOST
PS_NAME="honghan-vision-$PORT"
DASHBOARD_NAME="honghan-vision:$TAG.tar"
DASHBOARD_IMAGE="honghan-vision:$TAG"

echo "=========================Jenkins机器=========================================="


#打包前端镜像
yarn docker-hub:build:test
docker save  $DASHBOARD_IMAGE -o $DASHBOARD_NAME

rsync ./$DASHBOARD_NAME  -avz  root@$HOST:$PROJECT_LOCATION

rm -rf $DASHBOARD_NAME
docker image  ls $DASHBOARD_IMAGE -q |awk {print$1} |xargs docker rmi -f

ssh -tt root@$HOST bash -c "
    echo "=========================机器: $HOST=========================================="
    sleep 1
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker stop
    docker ps -a| grep $PS_NAME | awk '{print\$1}'|xargs docker rm -f
    docker rmi  $DASHBOARD_IMAGE
    sleep 3

    cd $PROJECT_LOCATION
    docker load -i $DASHBOARD_NAME
    rm -rf $DASHBOARD_NAME

    docker run -d -i -t -p $PORT:8360 --name=$PS_NAME $DASHBOARD_IMAGE
    exit
  "
```

扩展知识

Dockerfile指令、CLI常用命令、参数

Dockerfile指令

FROM

指定基础镜像

```
FROM nginx:1.18-alpine
```

指定基础镜像为nginx:1.18–alpine。nginx是镜像名，1.18–alpine是镜像tag。

```
FROM npm_pkg:v1.0.0 as builder
```

可以指定该阶段名称，为之后阶段的构建使用

WORKDIR

指定工作路径。假如该路径不存在，则创建之。

```
WORKDIR /a
WORKDIR b
WORKDIR c
```

工作路径变成`/a/b/c`，后续命令在此路径下执行。

COPY

将上下文里的指定文件、目录拷贝到镜像里的指定路径下。

```
COPY test.txt /mydir/
```

将上下文里的test.txt拷贝到镜像的`/mydir/`中。

ADD

将上下文里的指定文件、目录、远程文件URL拷贝到镜像里的指定路径下。

```
ADD foo.tar.gz /tmp/
```

将foo.tar.gz解压缩至/tmp目录下。

* 从URL下载的压缩包，不会解压。

RUN

执行命令并提交执行结果。

```
RUN mkdir /a/b/c
```

执行mkdir命令，创建/a/b/c目录。

ARG

构建时传入的参数变量

```
ARG VERSION=latest
FROM busybox:$VERSION
ARG VERSION
RUN echo $VERSION
```

```
docker build --build-arg VERSION=1.31.1 .
```

```
➜  tmp docker build --build-arg VERSION=1.31.1 .
Sending build context to Docker daemon  2.048kB
Step 1/4 : ARG VERSION=latest
Step 2/4 : FROM busybox:$VERSION
ARG VERSION=latest
 ---> 1c35c4412082
Step 3/4 : ARG VERSION
 ---> Running in 5651d66bca74
Removing intermediate container 5651d66bca74
 ---> 949c99ddf9c4
Step 4/4 : RUN echo $VERSION
 ---> Running in 0ee84f5eafde
1.31.1
Removing intermediate container 0ee84f5eafde
 ---> 42b14f7727f1
Successfully built 42b14f7727f1
```

CLI常用命令、参数

docker build实用参数

```
--network=host
```

指定RUN的网络。解决 *RUN yarn* 时，到私有仓库网络不通的问题

```
--build-arg XXX=xxx
```

指定参数

```
-t XXX:xxx
```

指定镜像名称、tag

```
-f ./Dockerfile
```

指定Dockerfile的路径。默认值是上下文里的Dockerfile

[docker image build参数]

docker run实用参数

```
-v /local/path:/container/path
```

绑定挂载卷，持久化读写

```
--rm
```

当容器退出的时候，自动删除容器

```
--publish 8360:80、-p 8360:80
```

将容器里的端口发布到宿主机

```
--detach、-d
```

在后台运行容器，并打印出容器ID

[docker container run参数]

其他命令

```
docker ps -a
```

查看所有容器进程

```
docker exec -it CONTAINER_ID bash
```

在运行的容器中执行bash命令

[docker container run参数]

清理磁盘空间

```
docker system prune
```

清理所有悬空镜像、已退出的容器进程等

```
docker stop $(docker ps -a -q)
```

停掉所有容器

```
docker rm $(docker ps -a -q)
```

删除所有容器

```
docker rmi $(docker images -f "reference=nginx" | awk '{print $3}')
```

删除特定名称的镜像

```
docker rmi $(docker images -q)
```

⚠️删除全部镜像

参考资料

- 官方文档
- Docker从入门到实践


THANK
YOU!