DOCKER CONTAINER LOADING

Dr. Roland Huß, Red Hat, @ro14nd
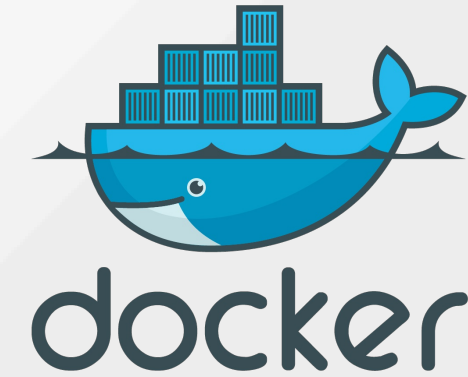
ROAD TRIP

Local	Remote / Cloud
Dockerfile	Docker Hub
Template	Continous Integration
Ansible-Container	Google Container Builder
Packer	OpenShift S2I
Rocker	
Build Integration	

DOCKER CLI

- ## docker commit

```
docker commit 3e3d3cf39 redis-server:1.0
```

- ## docker build

```
FROM ubuntu:14.04
RUN apt-get update && \
    apt-get install -y redis-server
EXPOSE 6379
ENTRYPOINT ["/usr/bin/redis-server"]
```

DOCKERFILE ARGS

- Dockerfile variables
- Since Docker 1.9
- Can not be used in FROM
- Filled in during build time

```
FROM busybox
ARG uid
USER ${uid:-jboss}
# ...
```

```
docker build --build-arg uid=daemon ....
```

PROBLEMS

- No composition
- Too simple parameterisation
- No influence on layers
- Limited feature set
- Security

redhat.

DOCKERFILE TEMPLATE

- Dockerfile generation from templates
- Build with docker build
- Simple built-in support (ARG)
- Engines:

  fish-pepper, dogen, crane, App::Dockerfile::Template ...

redhat

FISH-PEPPER

- **Template** system based on node.js
- **Multidimensional** parameters
- Support for **fragments**
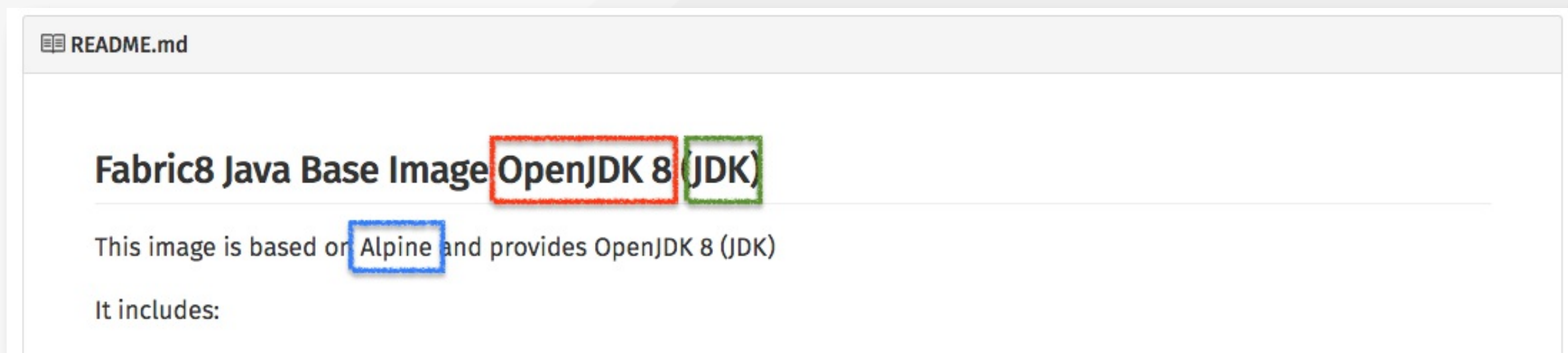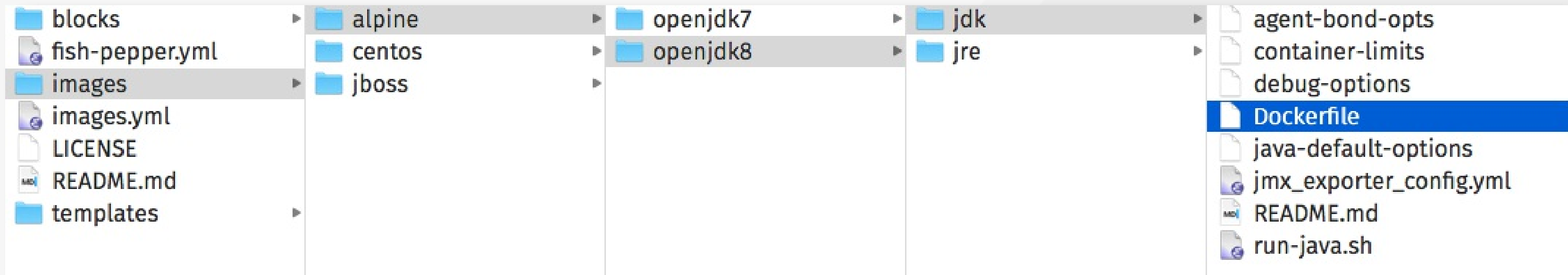- Docker **build** and **push** support

IMAGES.YML

```yaml
fish-pepper:
 params:
   - "base"
   - "version"
   - "type"
 name: "jolokia/fish-pepper-java"
 maintainer: "Roland Huss <roland@jolokia.org>"
 # ... additional global params
config:
 version:
  openjdk7:
   java: "java:7u79"
  openjdk8:
   java: "java:8u45"
 type:
  jre:
   extension: "-jre"
  jdk:
   extension: "-jdk"
 base:
  alpine:
   from: "alpine:3.4"
  # ... additional base definitions for 'centos', 'jboss'
```

DOCKERFILE TEMPLATE

```
FROM {{= fp.config.base.from }}

MAINTAINER {{= fp.maintainer }}

ENV JOLOKIA_VERSION {{= fp.jolokiaVersion }}

RUN chmod 755 /bin/jolokia_opts \
 && mkdir /opt/jolokia \
 && wget {{= fp.jolokiaUrl}} -O /opt/jolokia/jolokia.jar

CMD java -jar /opt/jolokia/jolokia.jar --version
```

redhat.

GENERATED BUILDS

blocks	▶	alpine	▶	openjdk7	▶	jdk	▶	agent-bond-opts
fish-pepper.yml		centos	▶	openjdk8	▶	jre	▶	container-limits
images	▶	jboss	▶					debug-options
images.yml								**Dockerfile**
LICENSE								java-default-options
README.md								jmx_exporter_config.yml
templates	▶							README.md
								run-java.sh

📖 README.md

Fabric8 Java Base Image OpenJDK 8 (JDK)

This image is based on Alpine and provides OpenJDK 8 (JDK)

It includes:

PRO

- Simple
- Uses Docker builtin mechanism
- Docker Hub automated builds still possible
- Good for many similar builds

CONTRA

- Restricted to Dockerfiles

ANSIBLE-CONTAINER

- Tool for building Docker images with Ansible
- Simple Orchestration for containers
- Manages multiple tools at once
- Easy to setup
- Uses **docker exec** and **docker cp** in Ansible connector

redhat

SETUP

```
ansible/
   # Docker compose file for base containers
   container.yml

   # Ansible playbook
   main.yml

   # Additional roles
   roles/
     dev-gulp/
       ....
      ...

   # Python and Role dependencies
   requirements.txt
   requirements.yml

   # Ansible configuration
   ansible.cfg
```

CONTAINER.YML

```yaml
version: "1"
defaults:
 POSTGRES_USER: django
 ...
services:
 django:
   image: centos:7
   environment:
    DATABASE_URL: "pgsql://{{ POSTGRES_USER }}:...."
   expose:
    - "{{ DJANGO_PORT }}"
   working_dir: "{{ DJANGO_ROOT }}"
   links:
    - postgresql
   command: ['{{ DJANGO_VENV }}/bin/gunicorn', .... ]
 postgresql:
   image: postgres:9.4
   expose:
    - "5432"
   volumes:
    - '/var/lib/postgresql/data'
```

MAIN.YML

```yaml
version: "1"
---
- hosts: django
  roles:
    - django-gunicorn
```

HOW IT WORKS

- **Start up base containers** from the specification given in container.yaml
- Create Ansible **inventory** on the fly
- Run ansible-playbook on main.yml for **provisioning**
- **Stop** containers
- **Commit** containers as images

redhat

PRO

- Flexible
- Reuse of Ansible roles
- Single layered images

CONTRA

- Complex system
- Longer Build times
- Single layered images

PACKER DOCKER BUILDER

- Packer: Tool for creating machine images
- Docker as a Builder
- Support of multiple provisioner
  - Shell, Ansible, Chef, Puppet ...
- Post-Processor for pushing and tagging

EXAMPLE

```json
{
 "builders":[{
  "type": "docker",
  "image": "fedora",
  "export_path": "image.tar"
 }],
 "provisioners":[
 {
  "type": "ansible",
  "playbook_file": "playbooks/local.yml"
 }],
 "post-processors": [{
  "type": "docker-import",
  "repository": "demo/packer-ansible",
  "tag": "0.1"
 }]
}
```

PRO

- Easy (re)usable for other builders
- Many ways to provision

CONTRA

- No Meta information configurable
- Ansible only via SSH
- No caching

redhat.

ROCKER

- Extension to the Dockerfile syntax
- Multiple FROM
- EXPORT/IMPORT for copying files
- MOUNT allows reuse during build
- Tag and Push directly from Rockerfile
- Templating

redhat.

EXAMPLE

```
FROM google/golang:1.4
ADD . /src
WORKDIR /src
RUN CGO_ENABLED=0 go build -a -installsuffix cgo \
              -v -o ball.o ball.go
EXPORT ball.o

FROM busybox
IMPORT ball.o /bin/ball
CMD ["/bin/ball"]

TAG ball:latest
```

redhat.

PRO

- Usuable extensions used in the wild
- Natural extension of a Dockerfile
- Backwards compatible

CONTRA

- Custom format
- Lock-In

redhat

BUILD INTEGRATION

- Create Docker images from within a build

- Maven plugins (Java)
  - fabric8io/docker-maven-plugin
  - fabric8io/fabric8-maven-plugin

DOCKER-MAVEN-PLUGIN

- Building images within Maven build
- Versioned
- Assembly with dependencies
- No Docker client required

https://github.com/fabric8io/docker-maven-plugin

redhat.

D-M-P CONFIGURATION

```xml
<image>
 <name>jolokia/jolokia-itest</name>
 <build>
  <from>consol/tomcat-7.0</from>
  <assemblyDescriptor>
   assembly.xml
  </assemblyDescriptor>
 </build>
 <run>
  <ports>
   <port>jolokia.port:8080</port>
  </ports>
 </run>
</image>
```

redhat.

FABRIC8-MAVEN-PLUGIN

- Embeds docker-maven-plugin
- Creates Kubernetes & OpenShift descriptors
- Creates image configuration from build info
- https://maven.fabric8.io

mvn package fabric8:build

ZERO CONFIG

- **Generators** for Image generation

```xml
<build>
  <plugins>
   <plugin>
    <groupid>io.fabric8</groupid>
    <artifactid>fabric8-maven-plugin</artifactid>
    <version>3.2.7</version>
   </plugin>
   <plugin>
    <groupid>org.springframework.boot</groupid>
    <artifactid>spring-boot-maven-plugin</artifactid>
   </plugin>
  </plugins>
</build>
```
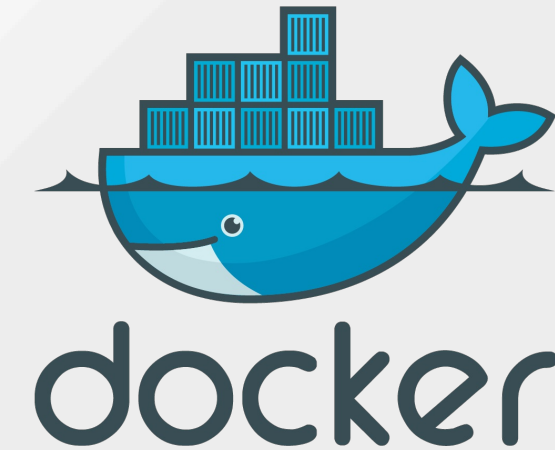
PRO

- Self contained builds
- No external requirements
- Reuse of existing build configuration

CONTRA

- Own configuration syntax
- More than one way

redhat

DOCKER HUB

- Public Docker registry
- Automated Builds
    - via manual trigger
    - via GitHub webhook
    - via update of dependant build
    - via HTTP POST request
- Multiple builds per repository
    - directory, tag, branch
- Tagging configurable

AUTOMATED BUILD

fabric8/java-alpine-openjdk8-jdk ☆

Last pushed: 2 months ago

Repo Info   Tags   Dockerfile   Build Details   **Build Settings**   Collaborators   Webhooks   Settings

Build Settings

☑ When active, builds will happen automatically on pushes.

The build rules below specify how to build your source into Docker images. The name can be a string or a regex. The Docker Tag name may contain variables. We currently support {sourceref}, which refers to the source branch/tag name. Show less

○ **Source Repository**
fabric8io-images/java

Here are a few examples:

Scenario	Type	Name	Docker Tag Name	Matches	Docker Tag Built
Exact match	Branch	master	latest	master	latest
Match versions	Tag	/^[0-9.]+$/	release-{sourceref}	1.2.0	release-1.2.0
Trailing modifiers	Tag	/^[0-9.]+/	release-{sourceref}	1.2.0-rc	release-1.2.0-rc

Type	Name	Dockerfile Location	Docker Tag Name		
Branch ▾	master	/images/alpine/openjdk8/jdk/	latest	+	▶ Trigger
Tag ▾	/^[0-9.]+/	/images/alpine/openjdk8/jdk/	Same as tag	−	

redhat

PRO

- Free for Open Source
- Easy to use
- Many triggers

CONTRA

- Dockerfile only
- No free private repository

redhat.

CONTINOUS INTEGRATION

- Jenkins: Docker build step plugin
- Docker as Service in :
  - Travis CI, Circle CI, Codeship, Drone.io ...

redhat.

CIRCLE CI

```
machine:
 services:
  - docker

deployment:
 latest:
  branch: master
  commands:
  - |
    test -n "$CI_PULL_REQUEST" || \
      ( cd java/images/jboss \
        && docker build -t fabric8/s2i-java:latest . | cat -  \
        && docker push fabric8/s2i-java:latest ) | cat - )
 release:
  tag: /v[0-9]+(\.[0-9]+){2}/
  commands:
  - |
    cd java/images/jboss && \
    ( docker build -t fabric8/s2i-java:${CIRCLE_TAG/#v/} . | cat - )
```

PRO

- CI Integration
- Flexible configuration

CONTRA

- Lock-in

GOOGLE CONTAINER BUILDER

- Build service for Google Cloud Platform
- Image will be stored in **GCR**
- **Build Step** : Docker container executed as part of the build

EXAMPLE

- cloudbuild.yaml :

```
steps:
- name: "gcr.io/cloud-builders/docker",
  args: [ "build", "-t", "gcr.io/devopscon-2016/java", "." ]
images: [ "gcr.io/devopscon-2016/java" ]
```

- Build:

```
gcloud container builds submit --config cloudbuild.yaml \
 gs://devopscon-2016/container-build-example.tar.gz
...
gcloud container builds describe $BUILD_ID
gcloud container builds list
```

redhat.

BUILD STEP IMAGES

- bazel (Google's build tool)
- **docker**
- gcloud (Google Cloude SDK tool)
- git
- go
- golang-project (Go project)
- gsutil (Cloud storage access)
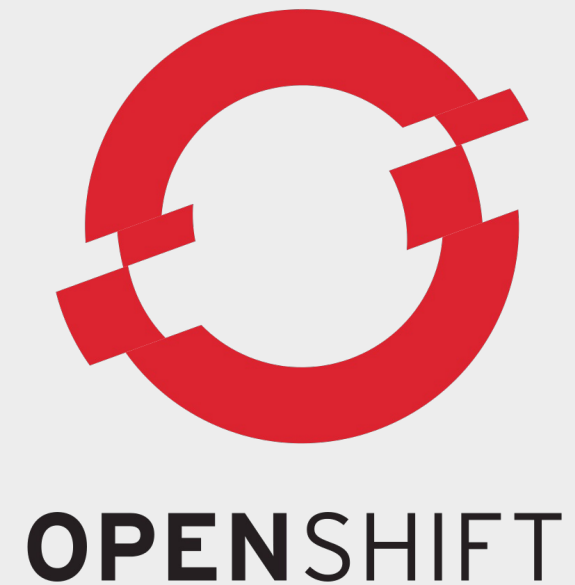- wget

redhat.

PRO

- Good integration into Google Cloud Platform
- Full access to all Docker features
- Easy to use

CONTRA

- Still Beta
- Not useful if not running in GCE

redhat.

OPENSHIFT S2I

- OpenShift: PaaS on top of Kubernetes
- S2I : Source-to-image
- Two ingredients:
  - Source code (local or Git remote)
  - S2I Builder Image

EXAMPLE

```
oc new-app \
  centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git
```

- Start builder container
- Checkout from GitHub
- Run build
- Stop container
- Commit container to app image
- Start the app container

PRO

- Improved security
- Easy to use

CONTRA

- Limited possibilities
- Source S2I for Java is slow
- Lock-In

redhat.

WRAP UP

- Dockerfiles are OK but limited
- There are many alternative ways to create Docker images
- Pick the one which fits your use case best

redhat.

QUESTIONS ?