Deploying Containerized Application in Docker vs OpenShift

Docker

1. Deploy MariaDB Container with Published Port

```
$ docker pull docker.io/mariadb:latest
$ docker container run -d --name wordpressdb \
-e MYSQL_ROOT_PASSWORD='redhat' \
-e MYSQL_DATABASE='wordpress' \
-e MYSQL_USER='wordpress' \
-e MYSQL_PASSWORD='redhat' \
-v /root/var/lib/mysql:/var/lib/mysql \
-p 3306:3306 \
docker.io/mariadb:latest
```

2. Deploy MariaDB from Remote Dockerfile on Git Server

```
$ docker build -t docker.io/furqanpr/mymariadb:10.4.12 \
    https://github.com/furqanpr/mymariadb.git

$ docker image push docker.io/furqanpr/mariadb:10.4

$ docker container run -d --name wordpressdb \
-e MYSQL_ROOT_PASSWORD='redhat' \
-e MYSQL_DATABASE='wordpress' \
-e MYSQL_USER='wordpress' \
-e MYSQL_PASSWORD='redhat' \
-v /root/var/lib/mysql:/var/lib/mysql \
-p 3306:3306 \
docker.io/furqanpr/mymariadb:10.4.12
```

Note for the Dockerfile:

```
$ ls
docker-entrypoint.sh  Dockerfile

$ tail Dockerfile
…
ADD https://raw.githubusercontent.com/furqanpr/mymariadb/master/docker-entrypoint.sh
/usr/local/bin/
RUN chmod 775 /usr/local/bin/docker-entrypoint.sh; \
    ln -s usr/local/bin/docker-entrypoint.sh / # backwards compat
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 3306
CMD ["mysqld"]
```

3. Build and Deploy App from Source Code (Multiple Stages Build) in Docker

```
$ git clone https://github.com/furqanpr/go-helloworld.git
$ cd go-helloworld/
$ cat Dockerfile

FROM golang:latest

ADD https://raw.githubusercontent.com/furqanpr/go-helloworld/master/hello.go hello.go
RUN go build hello.go

FROM busybox:latest

COPY --from=0 /go/hello /app/hello
COPY --from=0 /lib/x86_64-linux-gnu/libpthread.so.0 /lib/x86_64-linux-
gnu/libpthread.so.0
COPY --from=0 /lib/x86_64-linux-gnu/libc.so.6 /lib/x86_64-linux-gnu/libc.so.6
COPY --from=0 /lib64/ld-linux-x86-64.so.2 /lib64/ld-linux-x86-64.so.2

WORKDIR /app
EXPOSE 8080
ENTRYPOINT ["./hello"]

$ cat hello.go

package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "hello world")
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}


$ docker build -t docker.io/furqanpr/go-helloworld \
   https://github.com/furqanpr/go-helloworld.git

$ docker push docker.io/furqanpr/go-helloworld

$ docker run -d --name go-helloworld -p 80:8080 furqanpr/go-helloworld
```

OpenShift

1. Deploy MariaDB Pod with Exposed Port

```
$ oc new-app --name mymariadb \
-e MYSQL_ROOT_PASSWORD='redhat' \
-e MYSQL_DATABASE='wordpress' \
-e MYSQL_USER='wordpress' \
-e MYSQL_PASSWORD='redhat' \
docker.io/mariadb:latest

$ oc expose svc/mymariadb
```

2. Deploy MariaDB from Remote Dockerfile on Git Server

```
$ oc new-app --name mymariadb \
-e MYSQL_ROOT_PASSWORD='redhat' \
-e MYSQL_DATABASE='wordpress' \
-e MYSQL_USER='wordpress' \
-e MYSQL_PASSWORD='redhat' \
https://github.com/furqanpr/mymariadb.git

$ oc expose svc/mymariadb
```

3. 1. Perform S2I with Source Code + Dockerfile

```
$ oc new-app https://github.com/furqanpr/go-helloworld.git
--> Found Docker image be5888e (3 weeks old) from Docker Hub for "busybox:latest"

    * An image stream will be created as "busybox:latest" that will track the source
image
    * A Docker build using source code from https://github.com/furqanpr/go-
helloworld.git will be created
      * The resulting image will be pushed to image stream "go-helloworld:latest"
      * Every time "busybox:latest" changes a new build will be triggered
    * This image will be deployed in deployment config "go-helloworld"
    * Port 8080 will be load balanced by service "go-helloworld"
      * Other containers can access this service through the hostname "go-helloworld"
    * WARNING: Image "busybox:latest" runs as the 'root' user which may not be
permitted by your cluster administrator

--> Creating resources ...
    imagestream "busybox" created
    imagestream "go-helloworld" created
    buildconfig "go-helloworld" created
    deploymentconfig "go-helloworld" created
    service "go-helloworld" created
--> Success
    Build scheduled, use 'oc logs -f bc/go-helloworld' to track its progress.
    Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
     'oc expose svc/go-helloworld'
    Run 'oc status' to view your app.
```

3. 2. Perform S2I with Source Code Only

3.2.1. Go-Hello

```
$ oc new-app https://github.com/furqanpr/go-hello.git
error: No language matched the source repository

$ oc new-app https://github.com/furqanpr/go-hello
error: No language matched the source repository
```

3.2.2. PHP-HelloWorld

```
$ oc new-app https://github.com/furqanpr/php-helloworld.git

--> Found image 8e01e80 (5 months old) in image stream "openshift/php" under tag
"7.1" for "php"

    Apache 2.4 with PHP 7.1
    -----------------------
    PHP 7.1 available as container is a base platform for building and running
various PHP 7.1 applications and frameworks. PHP is an HTML-embedded scripting
language. PHP attempts to make it easy for developers to write dynamically generated
web pages. PHP also offers built-in database integration for several commercial and
non-commercial database management systems, so writing a database-enabled webpage
with PHP is fairly simple. The most common use of PHP coding is probably as a
replacement for CGI scripts.
```

4. Deploy Multiple Container Applications

4.1. Using Docker Compose (Stand Alone Server)

```
$ mkdir mywordpress
$ cd mywordpress/
$ vi docker-compose.yml

version: '3.6'
services:
   mariadb:
     image: mariadb:latest
     container_name: wordpressdb
     volumes:
       - /root/var/lib/mysql:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: redhat
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: redhat

   wordpress:
     depends_on:
       - mariadb
     image: wordpress:latest
     ports:
       - "8080:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: redhat
       WORDPRESS_DB_NAME: wordpress
     volumes:
       - /root/var/www/html:/var/www/html

$ docker-compose up -d
```

4.1.2. Deploy Application Stack on Docker Swarm (Clustered Server)

Create docker-stack.yml File

```
$ vi docker-stack.yml

version: "3.1"
services:
traefik:
    image: traefik:1.2
    command: --entryPoints='Name:http Address::80
Redirect.EntryPoint:https' --entryPoints='Name:https Address::443
TLS' --defaultEntryPoints=http,https --acme.entryPoint=https --
acme.email=rock@maddevs.io --acme.storage=/etc/traefik/acme/acme.json
--acme.domains=maddevs.io --acme.onHostRule=true --docker --
docker.swarmmode --docker.domain=maddevs.io --docker.watch
    ports:
      - 80:80
      - 443:443
    volumes:
      - traefic_vol:/etc/traefik/acme
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - traefik
    deploy:
      placement:
          constraints: [node.role == manager]
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
wp:
    image: wordpress:latest
    volumes:
      - wp_content:/var/www/html/wp-content
    secrets:
```

```
    Tags: builder, php, php71, rh-php71

    * The source repository appears to match: php
    * A source build using source code from https://github.com/furqanpr/php-
helloworld.git will be created
      * The resulting image will be pushed to image stream "php-helloworld:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "php-helloworld"
    * Ports 8080/tcp, 8443/tcp will be load balanced by service "php-helloworld"
      * Other containers can access this service through the hostname "php-
helloworld"

--> Creating resources ...
    imagestream "php-helloworld" created
    buildconfig "php-helloworld" created
    deploymentconfig "php-helloworld" created
    service "php-helloworld" created
--> Success
    Build scheduled, use 'oc logs -f bc/php-helloworld' to track its progress.
    Application is not exposed. You can expose services to the outside world by
executing one or more of the commands below:
     'oc expose svc/php-helloworld'
    Run 'oc status' to view your app.
```

4. Deploy Multiple Container Applications Using Template

GOGS Template

```
$ cat gogs-template.yaml

kind: Template
apiVersion: v1
metadata:
annotations:
description: The Gogs git server (https://gogs.io/) tags: instant-
app,gogs,go,golang
name: Gogs App
objects:
- kind: ServiceAccount
  apiVersion: v1
  metadata:
    creationTimestamp: null
    labels:
app: ${APPLICATION_NAME} name: ${APPLICATION_NAME}
- kind: Service
  apiVersion: v1
  metadata:
annotations:
description: Exposes the database server
name: ${APPLICATION_NAME}-postgresql spec:
ports:
- name: postgresql
port: 5432
      targetPort: 5432
    selector:
name: ${APPLICATION_NAME}-postgresql
- kind: DeploymentConfig
  apiVersion: v1
  metadata:
# for Postgres
annotations:
description: Defines how to deploy the database
name: ${APPLICATION_NAME}-postgresql labels:
app: ${APPLICATION_NAME} spec:
  replicas: 1
  selector:
name: ${APPLICATION_NAME}-postgresql strategy:
    type: Recreate
  template:
    metadata:
      labels:
name: ${APPLICATION_NAME}-postgresql name: ${APPLICATION_NAME}-postgresql
spec:
serviceAccountName: ${APPLICATION_NAME} containers:
- env:
- name: POSTGRESQL_USER value: ${DATABASE_USER}
- name: POSTGRESQL_PASSWORD value: ${DATABASE_PASSWORD}
- name: POSTGRESQL_DATABASE value: ${DATABASE_NAME}
- name: POSTGRESQL_MAX_CONNECTIONS value: ${DATABASE_MAX_CONNECTIONS}
- name: POSTGRESQL_SHARED_BUFFERS value: ${DATABASE_SHARED_BUFFERS}
- name: POSTGRESQL_ADMIN_PASSWORD value: ${DATABASE_ADMIN_PASSWORD}
        image: ' '
        livenessProbe:
          initialDelaySeconds: 30
          tcpSocket:
            port: 5432
          timeoutSeconds: 1
        name: postgresql
ports:
- containerPort: 5432 readinessProbe:
exec:
GOGS Installation
command:
- /bin/sh
- -i
- -c
- psql -h 127.0.0.1 -U ${POSTGRESQL_USER} -q -d ${POSTGRESQL_DATABASE} -c
'SELECT 1'
            initialDelaySeconds: 5
            timeoutSeconds: 1
          resources:
            limits:
              memory: 512Mi
volumeMounts:
- mountPath: /var/lib/pgsql/data
name: gogs-postgres-data volumes:
- name: gogs-postgres-data emptyDir: {}
triggers:
- imageChangeParams:
automatic: true containerNames: - postgresql from:
```

```
      - wp_db_password
    environment:
      - WORDPRESS_DB_USER=wp
      - WORDPRESS_DB_NAME=wp
      - WORDPRESS_DB_PASSWORD_FILE=/run/secrets/wp_db_password
      - WORDPRESS_DB_HOST=mariadb
    networks:
      - traefik
      - mariadb
    deploy:
      placement:
          constraints: [node.role == worker]
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
      labels:
        - "traefik.backend.loadbalancer.swarm=true"
        - "traefik.port=80"
        - "traefik.frontend.rule=Host:wp.maddevs.io"
        - "traefik.docker.network=traefik"
mariadb:
    image: mariadb
    volumes:
      - mariadb_vol:/var/lib/mysql
    secrets:
      - wp_db_password
      - root_db_password
    environment:
      - MYSQL_USER=wp
      - MYSQL_DATABASE=wp
      - MYSQL_PASSWORD_FILE=/run/secrets/wp_db_password
      - MYSQL_ROOT_PASSWORD_FILE=/run/secrets/root_db_password
    networks:
      - mariadb
    deploy:
      placement:
          constraints: [node.role == manager]
      replicas: 1
      restart_policy:
        condition: on-failure
secrets:
  wp_db_password:
    external: true
  root_db_password:
    external: true
volumes:
  traefic_vol:
    driver: "cloudstor:aws"
  wp_content:
    driver: "cloudstor:aws"
  mariadb_vol:
    driver: "cloudstor:aws"
    driver_opts:
      perfmode: maxio
networks:
  traefik:
    external: true
  mariadb:
    external: true
```

Create Secret to store Password

```
$ openssl rand -base64 20 | docker secret create root_db_password -
$ openssl rand -base64 20 | docker secret create wp_db_password -
```

Create Network

```
$ docker network create -d overlay traefik
$ docker network create -d overlay mariadb
```

Deploy Docker Stack

```
$ docker stack deploy -c docker-stack.yml wp
$ docker stack ps wp
```

Scaling Application

```
$ docker service update --replicas 3 wp_traefik
$ docker service update --replicas 2 wp_wp
$ docker stack ps wp
```

```
kind: ImageStreamTag name: postgresql:9.5 namespace: openshift
type: ImageChange - type: ConfigChange
- kind: Service
  apiVersion: v1
  metadata:
# container postgres come from Image Stream postgresql:9.5
annotations:
description: The Gogs server's http port
service.alpha.openshift.io/dependencies: '[{"name":"${APPLICATION_NAME}-
postgresql","namespace":"","kind":"Service"}]' labels:
app: ${APPLICATION_NAME} name: ${APPLICATION_NAME}
spec: ports:
- name: 3000-tcp port: 3000 protocol: TCP targetPort: 3000
selector:
app: ${APPLICATION_NAME} deploymentconfig: ${APPLICATION_NAME}
    sessionAffinity: None
    type: ClusterIP
  status:
    loadBalancer: {}
- kind: Route
apiVersion: v1
id: ${APPLICATION_NAME}-http metadata:
annotations:
description: Route for application's http service.
labels:
app: ${APPLICATION_NAME}
name: ${APPLICATION_NAME} spec:
    host: ${HOSTNAME}
    to:
name: ${APPLICATION_NAME}
- kind: DeploymentConfig
  apiVersion: v1
  metadata:
labels:
app: ${APPLICATION_NAME}
name: ${APPLICATION_NAME} spec:
    replicas: 1
    selector:
app: ${APPLICATION_NAME}
deploymentconfig: ${APPLICATION_NAME} strategy:
      resources: {}
      rollingParams:
        intervalSeconds: 1
        maxSurge: 25%
        maxUnavailable: 25%
        timeoutSeconds: 600
# for GOGS

    updatePeriodSeconds: 1
  type: Rolling
template:
  metadata:
    creationTimestamp: null
    labels:
app: ${APPLICATION_NAME}
deploymentconfig: ${APPLICATION_NAME} spec:
app: ${APPLICATION_NAME}
deploymentconfig: ${APPLICATION_NAME} containers:
- image: " "
imagePullPolicy: Always name: ${APPLICATION_NAME} ports:
- containerPort: 3000
       protocol: TCP
resources: {}
terminationMessagePath: /dev/termination-log volumeMounts:
- name: gogs-data
       mountPath: /opt/gogs/data
- name: gogs-config
       mountPath: /etc/gogs/conf
     readinessProbe:
         httpGet:
           path: /
           port: 3000
           scheme: HTTP
         initialDelaySeconds: 3
         timeoutSeconds: 1
         periodSeconds: 20
         successThreshold: 1
         failureThreshold: 3
     livenessProbe:
         httpGet:
           path: /
           port: 3000
           scheme: HTTP
         initialDelaySeconds: 3
         timeoutSeconds: 1
         periodSeconds: 10
         successThreshold: 1
         failureThreshold: 3
dnsPolicy: ClusterFirst restartPolicy: Always securityContext: {}
terminationGracePeriodSeconds: 30 volumes:
- name: gogs-data emptyDir: {}
- name: gogs-config configMap:
name: gogs-config items:
- key: app.ini path: app.ini
test: false
triggers:
- type: ConfigChange - imageChangeParams:
automatic: true containerNames:
- ${APPLICATION_NAME} from:
     kind: ImageStreamTag
name: ${APPLICATION_NAME}:${GOGS_VERSION} type: ImageChange
- kind: ImageStream
  apiVersion: v1
  metadata:
labels:
app: ${APPLICATION_NAME}
name: ${APPLICATION_NAME} spec:
tags:
- name: "${GOGS_VERSION}"
# for GOGS, IS from Postgres already exist
from:
```

```
kind: DockerImage
name: docker.io/openshiftdemos/gogs:${GOGS_VERSION}  Replace with:
importPolicy: {}
annotations:
registry.lab.example.com/do280/gogs

description: The Gogs git server docker image tags: gogs,go,golang
version: "${GOGS_VERSION}"
- kind: ConfigMap
  apiVersion: v1
  metadata:
name: gogs-config labels:
app: ${APPLICATION_NAME} data:
app.ini: | RUN_MODE = prod RUN_USER = gogs
[database] DB_TYPE = HOST = NAME = USER = PASSWD =
postgres ${APPLICATION_NAME}-postgresql:5432 ${DATABASE_NAME}
${DATABASE_USER} ${DATABASE_PASSWORD}
[repository]
ROOT = /opt/gogs/data/repositories
[server] ROOT_URL=http://${HOSTNAME} SSH_DOMAIN=${HOSTNAME}
[security]
INSTALL_LOCK = ${INSTALL_LOCK}
[service] ENABLE_CAPTCHA = false
[webhook]
SKIP_TLS_VERIFY = ${SKIP_TLS_VERIFY}
parameters:
- description: The name for the application. name: APPLICATION_NAME
required: true
value: gogs
- description: 'Custom hostname for http service route. Leave blank for
default hostname, e.g.: <application-name>-<project>.<default-domain-
suffix>'
name: HOSTNAME
  required: true
- displayName: Database Username
from: gogs
value: gogs
name: DATABASE_USER
- displayName: Database Password from: '[a-zA-Z0-9]{8}'
value: gogs
name: DATABASE_PASSWORD
- displayName: Database Name
name: DATABASE_NAME
value: gogs
- displayName: Database Admin Password from: '[a-zA-Z0-9]{8}'
generate: expression
name: DATABASE_ADMIN_PASSWORD
- displayName: Maximum Database Connections
name: DATABASE_MAX_CONNECTIONS
value: "100"
- displayName: Shared Buffer Amount
name: DATABASE_SHARED_BUFFERS
value: 12MB
- name: GOGS_VERSION
  displayName: Gogs Version
description: 'Version of the Gogs container image to be used (check the
available version https://hub.docker.com/r/openshiftdemos/gogs/tags)'
value: "0.9.97"
  required: true
- name: INSTALL_LOCK
  displayName: Installation lock
description: 'If set to true, installation (/install) page will be disabled.
Set to false if you want to run the installation wizard via web'
value: "true"
- name: SKIP_TLS_VERIFY
displayName: Skip TLS verification on webhooks
description: Skip TLS verification on webhooks. Enable with caution! value:
"false"
```

Deploy Application Directly from Template File

```
$ oc new-app -f gogs-template.yaml
```

Deploy Application Directly from Template Resource

```
$ oc create -f gogs-template.yaml -n openshift
$ oc new-app --template ‘Gogs App’
```