

random bits of computer info

Wednesday, June 1, 2016

Creating docker images suitable for OpenShift (ssh-git image HowTo)

Intro

This is not going to be a detailed guide for creating docker images. I'll present an example ssh-git image and highlight the more important concerns for running such image on OpenShift v3. Things are basically in documentation but I hope to just get you started quickly. (update: wow I thought it's gonna be a few lines but it turned into a monster)

tl;dr; skip to the OpenShift section

Plain Docker image

Starting with little docker experience and no knowledge about OpenShift requirements I just went ahead to create a standard SSH server image and because of a nice git feature, one can just create a local `bare` repo to be served over SSH (to whoever has a matching key in `~/.ssh/authorized_keys`).

I looked around but found only a few Ubuntu examples. My favorite distro is Fedora (I'm affiliated but still) so thought it's a shame and went ahead to create a fedora based Dockerfile. In fact it was pretty much pain-free. Here's my initial version running OpenSSH as root:

<https://github.com/openshift-qe/ssh-git-docker/blob/master/ssh-git-root/Dockerfile>

The interesting points are:

- `FROM fedora:latest`
- `RUN ...` - pretty much standard commands to install ssh and configure a user; I usually also do `restorecon -R ~/.ssh` but inside docker selinux is nil, thus that's skipped.
- `EXPOSE 22` - so that docker knows which ports are needed
- `CMD ssh-keygen -A && exec /usr/sbin/sshd -D` - here interesting part is generating keys as OpenSSH can't work properly otherwise

Building, running, tagging, pushing

Building

```
# docker build -t docker.io/myaccount/imagename:latest PATH
```

Where latest can also be another version.

Tagging

```
# docker tag docker.io/myaccount/imagename:latest docker.io/myaccount/imagename:1.0
```

As image you can use a tag or an image hash. Doesn't matter.

Running

Launch container with ports exposed and giving container a name.

```
# docker run -d -P --name ssh-git-server myaccount/imagename:latest
```

btw you can try the built image from `aosqe/ssh-git-server:root-20150525`

Get exposed port number so you can use it later.

```
# docker port ssh-git-server 22
```

Put your ssh public key in.

```
# docker exec ssh-git-server bash -c 'echo "ssh-rsa ..." > /home/git/.ssh/authorized_keys'
```

Clone the sample repo:

```
$ git clone ssh://git@localhost:32769/repos/sample.git
```

Terminating

```
# docker rm ssh-git-server  
# docker rmi <image tag> # to get rid of the image locally
```

Sharing with others (pushing)

Then you can push these images to dockerhub:

```
# docker login docker.io  
# docker push docker.io/myaccount/imagename:lates
```

Image where SSHd runs as a regular user

I knew OpenShift doesn't let you run images as root so next step was to create an image where OpenSSH runs as the `git` user. In fact it allows you, but you have to grant your user extra privileges and there is really no good reason to do that for a ssh-git server. ~~Also a future OpenShift Online service would not allow such extra privileges for security reasons.~~ At some point it is likely secure root pods to be allowed using **user namespaces** with some performance penalty.

That was even less painful thanks to an old [post in cygwin list](#). Basically ~~privilege separation needs to be turned off as it can only work as root~~ and adjust some locations in sshd_config using `sed`. Finally little `chown/chmod` adjustments. And before I forget, port cannot be 22 so I selected 2022.

So new things are adding more `RUN` commands and the `USER git` directive so final CMD is run as the user instead root. Here's the result:

<https://github.com/openshift-qe/ssh-git-docker/blob/master/ssh-git-user/Dockerfile>

You can try:

```
# docker run -d -P --name ssh-git-server aosqe/ssh-git-server:git-20150525
```

But testing this on OpenShift I've got the strange error message:

```
No user exists for uid 1000530000
```

I was stuck here for a little while until I could figure that error is not produced by OpenShift but ssh server itself.

OpenShift ready Image

What I found out (see the [official guidelines](#) in the References section) is that regardless of your `USER` directive in Dockerfile, unless you give the user or service account that would launch pod as some random UID. The group will be static though - root.

Because that random UID will not be part of the passwd file, some programs will fail to start with an error message like what I saw above. Another issue is that pre-setup of SSH becomes impossible as some files need to be with permissions 700 for ssh to accept them. Obviously as a random UID we cannot repair that once pod starts.

Here's how I approached:

1. move most setup to the container start CMD
2. make a couple directories writable to the root group so that step #1 can create necessary new files (this time with proper owner and permissions)
3. make `passwd` root group writable so that we can fix our UID ([official guideline](#) suggests using nss wrapper but I thought it's easier to just fix in-place)

End result is otherwise basically the same thing, just moving around the commands:

<https://github.com/openshift-qe/ssh-git-docker/blob/master/ssh-git-openshift/Dockerfile>

btw I have to change the multi-line CMD to a shell script and add to image. Would be easier to customize.

Doing it the OpenShift way

Since I've got an [OpenShift Enterprise](#) environment running I thought to use it directly instead of using plain docker commands (which would also work fine):

```
$ oc new-build -name=git-server --context-dir=ssh-git-openshift https://github.com/openshi
```

FYI you can append `#branch` if you want to build of non-default branch. The good thing using this approach is that image can be rebuilt automatically when base image (fedora:latest) changes and when your code changes. You may need to configure hooks though. See [triggers doc](#).

To monitor build:

```
$ oc logs -f bc/git-server
```

In the log, you will see something like (can be useful later):

The push refers to a repository [172.30.2.69:5000/a9vk7/git-server]

Now run the image by:

```
$ oc new-app git-server:latest --name=git-server
```

You would end up with a deployment config called git-server that creates a replication controller `git-server-1` that keeps one pod called `git-server-...` running from the `git-server` image stream created by the `new-build` command. Also a service called `git-server` is created that will provide you with a stable IP to access the pod + its name can be used as a hostname of the git server in any pod or build that happens within the same project.

One last detail is to make service listen on port 22 for nicer git URLs:

```
$ oc edit svc git-server # change `port` to 22 from 2022
```

Note that services can only be accessed from pods running in same project or in project 'default'. To access service from the internet, you need to create a nodePort service. Because this is not HTTP based, we can't use regular routes. Hope to get on that later.

To see you pod name and use it, you can do:

```
$ oc get pod # see pod name
```

Then:

```
$ oc rsh git-server-... # configure ssh keys there, create repos, etc.
```

Now once you have your public key in the pod, you can access this server from other pods. You can do for trying out from the server pod itself. Provided you have the matching private key. While into `rsh` do:

```
$ git clone git@git-server:sample.git
```

To push your image to dockerhub, see how to [set build config output](#). Or you can manually ssh to the OpenShift node and do as root:

```
# docker tag 172.30.2.69:5000/a9vk7/git-server docker.io/myaccount/imagename:latest
# docker login docker.io
# docker push docker.io/myaccount/imagename:latest
```

If you want to run your image off dockerhub, you can do:

```
$ oc run git-server --image=aosqe/ssh-git-server-openshift
$ oc expose dc git-server --port=22 --target-port=2022
$ oc set probe dc/git-server --readiness --open-tcp=2022
```

Setting the probe lets your replication controller notice pod is dead and spawn a new one.

Some words about persistent volumes

The way images I refer to above are built would cause any changes in public keys and repo data to be lost upon pod restart. To avoid that **persistent volumes** need to be used.

Persistent volumes at attach time will be chowned to the current UID of the pod. Provided the OpenShift ready image does setup at launch time, that should be easily supportable. i.e. mount volume to /home/git/

But a few changes will still need to be done:

- creation of sample git repo needs to be conditional, when it doesn't exist
- `sshd_config` and ssh-keygen should create host keys somewhere in `git` user home dir to keep host keys between pod restarts

Future work

- make the OpenShift ready image runnable off a persistent volume
- add info about making repo accessible from the Internet
- convert multi-line CMD to a startup script

Current post is based on the initial commit of the docker files in the repo.

References

- Dockerfile git repo - <https://github.com/openshift-qe/ssh-git-docker>
- image for OpenShift - <https://hub.docker.com/r/aosqe/ssh-git-server-openshift/>
- plain docker images - <https://hub.docker.com/r/aosqe/ssh-git-server>
- OpenShift Image guidelines - https://docs.openshift.org/latest/creating_images/guidelines.html

Aleksandar Kostadinov at 12:48 AM

Share

5 comments:

Anonymous January 26, 2017 at 12:18 PM

Just what I was looking for. Thank you!

Reply



Unknown October 6, 2017 at 9:33 AM

It's very helpful. Thank you very much!

Can you give some tip how to making repo accessible from the Internet?

Reply

▼ Replies

Aleksandar Kostadinov  October 6, 2017 at 1:01 PM



It is possible to use the nodeport feature to expose the git-ssh server on the internet.

It is easier though to use the HTTP git server template which exposes the git server via an HTTP route: <https://github.com/openshift/origin/blob/master/examples/gitserver/gitserver-persistent.yaml>

Read the file to see what variables you can set to the `git` deployment config especially in regards to auth. e.g. server auth vs user/password pair.

For some use cases you may also want to add `edit` role to the `git` service account.

Reply



Aleksander November 21, 2018 at 1:06 PM

Big thanks! I was pulling my hair out and the trick with editing /etc/passwd works perfectly.

Reply



Hasnain December 4, 2020 at 2:15 AM

THANKYOU !

Reply

Enter your comment...



Comment as: 1079087531@r ▼

Sign out

Publish

Preview

☐ Notify me




Home



[View web version](#)

About Me

 **Aleksandar Kostadinov**

[View my complete profile](#)

Powered by [Blogger](#).