Building Hyperledger Fabric for Linux on z Systems

SUSE Linux Enterprise Server Distribution

v05-17-16

John Harrison
Barry Silliman

jharriso@us.ibm.com
silliman@us.ibm.com

Contents

1 Overview

This document describes the steps to build, configure and install the infrastructure components associated with IBM's Open Blockchain technology, Hyperledger Fabric, on the Linux on z Systems platform.

More importantly, you will create the Docker artifacts using the base SLES 12 SP1 system on which you will deploy. The base Docker image will be SLES12 SP1 based and have access to the same repositories as the system on which you deploy. This eliminates the need to download any pre-built Docker images from the public Docker repository, eliminating one potential security exposure. The Docker images you create will be kept in a repository that you create, thus remaining within your control.

The major components include:

- Golang programming language
- Docker client and daemon
- Docker registry
- Hyperledger Fabric
  - Peer
  - Membership and Security Services

Once all of the major components are built, custom Docker images are created for the Golang programming language, Hyperledger Fabric Peer, and Hyperledger Fabric Membership and Security Services. This allows for a fully *dockerized* development or proof-of-concept Hyperledger Fabric environment.

The procedures in this guide are tailored for SUSE Linux Enterprise Server (SLES) 12 SP1. Due to the ongoing development activity within the Hyperledger project, there is a chance that portions of this document may become obsolete or out of date.

For more information about the Hyperledger Fabric project, see https://github.com/hyperledger/fabric.

2 Building Golang

The Hyperledger Fabric and the Docker Registry are written using the Golang programming language. Therefore, a Golang compiler needs to be built in order to compile the Hyperledger Fabric and Docker Registry source code.

Building Golang for Linux on z Systems is a two-step process:

1. Cross-compile the Golang bootstrap tool on an Intel/AMD-based machine running an up-to-date version of Linux.
2. Build the Golang toolchain on Linux on z Systems using the bootstrap tool created in step 1.

NOTE: The instructions contained in this document assume that you are using a non-root user with sudo authority and that the non-root user has been added to the **`wheel`** group. In addition, update the **`/etc/sudoers`** file to enable the **`wheel`** group with no password access, and append **`/usr/local/bin`** and the targeted directory that will contain the **`go`** executable to the **`secure_path`** variable. The targeted directory is set in step 4 of Building the Golang Toolchain on page 3. Otherwise, if you have root access… Great! No need to worry about this. ☺

For information on how the Golang bootstrapping process works, see the blog entry at http://dave.cheney.net/2015/10/16/bootstrapping-go-1-5-on-non-intel-platforms.

2.1 Cross-Compiling the Bootstrap Tool

To build the Golang bootstrap tool you will need to use an Intel/AMD-based machine running an up-to-date version of Linux. The bootstrap tool can be built using either SLES or RHEL. The instructions below assume that a RHEL Linux system is used in building the tool.

NOTE: The directory **`/<work_dir>/`** used in these instructions represents a temporary writeable directory of your choice.

1. Install the dependencies:

```
sudo yum install -y git wget tar gcc bzip2
```

2. Create a directory for the amd64 version of the Golang toolchain:

```
mkdir -p /<work_dir>/go1.5.2
cd /<work_dir>/go1.5.2
```

3. Download the amd64 Golang toolchain binary and extract it:

```
wget https://storage.googleapis.com/golang/go1.5.2.linux-amd64.tar.gz
tar -xvf go1.5.2.linux-amd64.tar.gz
```

NOTE: Even though the file name indicates that it is a *gzipped* file, it is just a regular tar file. There is no need to use the **`z`** tar flag.

4. Clone the source code for the z Systems port of Golang:

```
cd /<work_dir>/
git clone https://github.com/linux-on-ibm-z/go.git
```

5. Build the bootstrap tool:

```
export GOROOT_BOOTSTRAP=/<work_dir>/go1.5.2/go
cd /<work_dir>/go/src
GOOS=linux GOARCH=s390x ./bootstrap.bash
```

The bootstrap tool is placed into a bzip tarball named **go-linux-s390x-bootstrap.tbz** located in **/<work_dir>/** and is used in the next step to compile the Golang programming language source code on Linux on z Systems.

2.2 Building the Golang Toolchain

To build the Golang toolchain you need to have successfully built the Golang bootstrap tool outlined in the Cross-Compiling the Bootstrap Tool section of this document. After building the bootstrap tool, login to your Linux on z Systems machine and perform the steps below.

1. Install the dependencies.

```
sudo zypper --non-interactive in git-core gcc
```

2. If you do not have a home directory that is shared (e.g., via NFS) with the AMD64/Intel machine, then scp or ftp **/<work_dir>/go-linux-s390x-bootstrap.tbz** from the AMD64/Intel machine to **/<work_dir>/** on the Linux on z Systems machine, and clone the source again:

```
mkdir /<work_dir>/
cd /<work_dir>/
# scp/ftp /<work_dir>/go-linux-s390x-bootstrap.tbz from the AMD64/Intel system
tar -xf go-linux-s390x-bootstrap.tbz
git clone https://github.com/linux-on-ibm-z/go.git
```

3. Build the Golang toolchain on z Systems and run all tests.

```
export GOROOT_BOOTSTRAP=/<work_dir>/go-linux-s390x-bootstrap
cd /<work_dir>/go/src
./all.bash
```

NOTE: If most of the tests pass then the Golang toolchain probably compiled OK and you can proceed to the next step.

4. Copy the Golang directory to the final install directory, **/<golang_home>/**, and permanently update your **PATH** environment variable to use the new toolchain. Now you can compile Golang programs on Linux on z Systems.

```
sudo cp -ra /<work_dir>/go /<golang_home>/
# Also add the following lines to ~/.bash_profile
export PATH=/<golang_home>/go/bin:$PATH
```

3 Docker

The Hyperledger Fabric peer relies on Docker to deploy and run Chaincode (aka Smart Contracts). In addition, for development purposes, the Hyperledger Fabric peer service and the membership and security service can both run in Docker containers. The Hyperledger Fabric peer unit tests include tests that build both a peer service Docker image and a membership and security service Docker image. This is covered later in the document.

A Docker registry is required for the Hyperledger Fabric environment and the process to build your own Docker registry from source is described below.

3.1 Installing the Docker Client / Daemon

Since Docker is packaged with SLES 12 SP1

1. Ensure that the apparmor package is up to date:

```
sudo zypper in apparmor
```

2. Install the Docker package:

```
sudo zypper in docker
```

3. Create a shell script in **/usr/local/bin** to start the Docker Daemon and redirect all output to a logging file. Ensure that the shell script has the executable attribute set.

```
#!/bin/bash
/usr/local/bin/docker daemon -H tcp://0.0.0.0:2375 -H
unix:///var/run/docker.sock --insecure-registry localhost:5050 >
/var/log/docker.log 2>&1 &
```

NOTE: If your Docker Registry is running on another system, change **localhost** to either the hostname or IP address of the system running the Docker Registry. Also note that the port number of the insecure registry matches the port number that the Docker Registry is listening on.

4. Start the Docker Daemon shell script:

```
sudo touch /var/run/docker.sock
sudo <docker-daemon-script-name>
```

NOTE: In order to run the Docker Daemon shell script from a non-root user without prefixing the command with sudo, a docker group needs to be created and the non-root user needs to be added to the docker group:
sudo groupadd docker
sudo usermod -a -G docker <non-root-user>

3.2 Building the Docker Registry

The Docker Registry 2.0 implementation for storing and distributing Docker images is part of the GitHub Docker Distribution project. The Docker Distribution project consists of a toolset to pack, ship, store, and deliver Docker content.

The reason to create your own registry is twofold. First, it is your private registry. Second, you will create a registry for Linux on z Systems Docker daemons, which will allow the use of the same unaltered Dockerfile contents used by the Docker daemon on the x86 platform. This eliminates source code changes to the Hyperledger fabric code.

NOTE: The directory **/<work_dir>/** used in these instructions represents a temporary writeable directory of your choice.

1. Install the dependencies:

```
sudo zypper in git-core make
```

   Golang is required to build the Docker Registry. See Building Golang on page 2 to build the Golang toolchain. You may have already installed the **git** and **make** packages when building Golang. If so, ignore the installation of the packages within this step.

2. Create a distribution directory and clone the source code:

```
mkdir -p /<work_dir>/src/github.com/docker
cd /<work_dir>/src/github.com/docker
git clone https://github.com/docker/distribution.git
cd /<work_dir>/src/github.com/docker/distribution
git checkout v2.3.0
```

3. Set **GOPATH** and **DISTRIBUTION_DIR** environment variables:

```
export DISTRIBUTION_DIR=/<work_dir>/src/github.com/docker/distribution
export GOPATH=/<work_dir>/
export GOPATH=$DISTRIBUTION_DIR/Godeps/_workspace:$GOPATH
```

4. Build the distribution binaries:

```
cd /<work_dir>/src/github.com/docker/distribution
make PREFIX=/<work_dir>/ clean binaries
```

5. Run the Test Suite:

```
make PREFIX=/<work_dir>/ test
```

6. Start the Docker Registry:

   The Docker Registry fetches the configuration from **$DISTRIBUTION_DIR/cmd/registry/config.yml**. The default filesystem location where the Docker Registry stores images is **/var/lib/registry.**

(a) Copy the config-dev.yml file to config.yml:

```
cp $DISTRIBUTION_DIR/cmd/registry/config-dev.yml
$DISTRIBUTION_DIR/cmd/registry/config.yml
```

(b) Tailor the Docker Registry configuration file and save:

(i) Change the default storage caching mechanism. If you are not using redis for storage caching, edit **$DISTRIBUTION_DIR/cmd/registry/config.yml** and change the **storage.cache.blobdescriptor** parameter from **redis** to **inmemory**.

(ii) Change the default listening port of the Docker Registry. Edit **$DISTRIBUTION_DIR/cmd/registry/config.yml** and change the **http.addr** parameter from **5000** to **5050**. This change is required because port 5000 conflicts with the Hyperledger Fabric peer's REST service port, which uses port 5000.

(c) Create the default directory to store images, if it does not exist:

```
sudo mkdir -p /var/lib/registry
```

(d) Start the Docker Registry:

```
/<work_dir>/bin/registry $DISTRIBUTION_DIR/cmd/registry/config.yml
```

For a more permanent solution when starting the Docker Registry:

1. Setup homes for the Docker Registry executable binary and its configuration file:

```
sudo mkdir /etc/docker-registry
sudo cp $DISTRIBUTION_DIR/cmd/registry/config.yml /etc/docker-registry
sudo cp /<work_dir>/bin/registry /usr/local/bin
```

2. Create a shell script in **/usr/local/bin** to start the Docker Registry in the background and redirect all output to a logging file. Ensure that the shell script has the executable attribute set.

```
#!/bin/bash
/usr/local/bin/registry /etc/docker-registry/config.yml > /var/log/docker-
registry.log 2>&1 &
```

3. Start the Docker Registry:

```
sudo <docker-registry-script-name>
```

For more information on the Docker Distribution project, see https://github.com/docker/distribution.

4 Build and Install RocksDB

RocksDB is an embeddable persistent key-value store for fast storage and is used by the Hyperledger Fabric peer, membership and security service components.

NOTE: The directory **/<work_dir>/** used in these instructions represents a temporary writeable directory of your choice.

1. RocksDB is written using the C++ programming language. Make sure that the C++ compiler is installed:

```
sudo zypper in gcc-c++
```

2. Download and build RocksDB:

```
cd /<work_dir>/
git clone --branch v4.5.1 --single-branch --depth 1
https://github.com/facebook/rocksdb.git
cd rocksdb
sed -i -e "s/-march=native/-march=z196/" build_tools/build_detect_platform
sed -i -e "s/-momit-leaf-frame-pointer/-DDUMMY/" Makefile
make shared_lib
```

3. Copy rocksdb to path /opt:

```
sudo cp -ra /<work_dir>/rocksdb /opt/
```

5 Build the Hyperledger Fabric Core

The Hyperledger Fabric Core contains code for running peers, either validating or non-validating, and membership services for enrollment and certificate authority tasks.

NOTE: The directory **/<work_dir>/** used in these instructions represents the location of the Hyperledger Fabric source code. The **/<golang_home>/go** directory represents where Golang was installed after performing step 4 in Building the Golang Toolchain on page 3. If you built Golang using this document, you have already added the Golang **bin** directory to your **PATH**.

1. Install pre-req packages:

```
Sudo zypper --non-interactive --no-gpg-check in zlib zlib-devel libsnappy1 snappy-devel
libbz2-1 libbz2-devel
```

2. Download the Hyperledger Fabric code into a writable directory:

```
mkdir -p /<work_dir>/src/github.com/hyperledger
export GOPATH=<work_dir>
cd /<work_dir>/src/github.com/hyperledger
git clone https://github.com/hyperledger/fabric.git
cd fabric
```

3. Setup environment variables:

```
export GOROOT=/<golang_home>/go
export PATH=/<golang_home>/go/bin:$PATH
export CGO_LDFLAGS="-L/opt/rocksdb -lrocksdb -lstdc++ -lm -lz "
export CGO_CFLAGS="-I/opt/rocksdb/include"
export LD_LIBRARY_PATH=/opt/rocksdb:$LD_LIBRARY_PATH
```

NOTE: If you are going to be rebuilding Golang or RocksDB, add the environment variables in steps 2 and 3 to your **.bash_profile** file. The **LD_LIBRARY_PATH** variable must be set when executing the peer or membership and security services executable binaries, and therefore, should be set in your **.bash_profile** file.

4. Build the Hyperledger Fabric executable binaries. The peer binary runs validating or non-validating peer nodes and the membersrvc binary is the membership and security server that handles enrollment and certificate requests:

```
cd $GOPATH/src/github.com/hyperledger/fabric/peer
go build -v
cd $GOPATH/src/github.com/hyperledger/fabric/membersrvc
go build -v
```

For a more permanent solution, you can create shell scripts to start the peer and the membership and security services executables in the background and re-direct logging output to a file.

1. Create a file called **fabric-peer.sh** located in **/usr/local/bin** with the executable attribute set:

```
#!/bin/bash
export LD_LIBRARY_PATH=/opt/rocksdb
cd /<work_dir>/src/github.com/hyperledger/fabric/peer
./peer node start --logging-level=debug > /var/log/fabric-peer.log 2>&1 &
```

2. Create a file called **membersrvc.sh** located in **/usr/local/bin** with the executable attribute set:

```
#!/bin/bash
export LD_LIBRARY_PATH=/opt/rocksdb
cd /<work_dir>/src/github.com/hyperledger/fabric/membersrvc
./membersrvc > /var/log/membersrvc.log 2>&1 &
```

6 Build a Golang Toolchain Docker Image

The section describes the steps required to build a Docker image that is comprised of the Golang programming language toolchain built upon the SLES operating system. There is no need to download any pre-existing Docker images from the Docker Hub or from any other Docker registry that is on the internet.

It is a two-step process to build the Golang toolchain Docker image:

1. Build your own SLES Docker image from scratch.
2. Build a Golang toolchain Docker image from the base SLES Docker image built in step 1.

This Docker image is used by the Hyperledger Fabric peer component when deploying Chaincode. The peer communicates with the Docker Daemon to initially create docker images based on the Golang toolchain Docker image and contains the compiled Chaincode built from source specified by the **peer deploy** command. Docker containers are started by the peer and execute the Chaincode binary awaiting further Blockchain transactions, e.g., invoke or query.

6.1 Build a Base SLES Docker Image

1. Make sure that your Docker Daemon and Docker Registry are started. Refer to Installing the Docker Client / Daemon and Building the Docker Registry sections above for building and starting the Docker Daemon and Docker Registry.
2. Create a new file called **/usr/local/bin/mkimage-zypp.sh** and paste the following contents into the new file. Enable the executable attribute for the new file.

```
#!/usr/bin/env bash
#
# Create a base SLES Docker image.
#

usage() {
    cat <<EOOPTS
$(basename $0) <name>
EOOPTS
    exit 1
}

name=$1

if [[ -z $name ]]; then
    usage
fi

target=$(mktemp -d --tmpdir $(basename $0).XXXXXX)

set -x

mkdir -m 755 "$target"/dev
```

```
mknod -m 600 "$target"/dev/console c 5 1
mknod -m 600 "$target"/dev/initctl p
mknod -m 666 "$target"/dev/full c 1 7
mknod -m 666 "$target"/dev/null c 1 3
mknod -m 666 "$target"/dev/ptmx c 5 2
mknod -m 666 "$target"/dev/random c 1 8
mknod -m 666 "$target"/dev/tty c 5 0
mknod -m 666 "$target"/dev/tty0 c 4 0
mknod -m 666 "$target"/dev/urandom c 1 9
mknod -m 666 "$target"/dev/zero c 1 5

# install core packages
zypper --non-interactive --no-gpg-check --reposd-dir /etc/zypp/repos.d --root
"$target" install -l pattern:Minimal vim
cp -ra /etc/zypp "$target"/etc
cp -ra /etc/products.d "$target"/etc
zypper --non-interactive --no-gpg-check --reposd-dir /etc/zypp/repos.d --root
"$target" clean
zypper --non-interactive --no-gpg-check --reposd-dir /etc/zypp/repos.d --root
"$target" refresh -d

#  locales
rm -rf
"$target"/usr/{{lib,share}/locale,{lib,lib64}/gconv,bin/localedef,sbin/build-
locale-archive}
#  docs and man pages
rm -rf "$target"/usr/share/{man,doc,info,gnome/help}
#  cracklib
rm -rf "$target"/usr/share/cracklib
#  i18n
rm -rf "$target"/usr/share/i18n
#  zypp cache
rm -rf "$target"/var/cache/zypp
mkdir -p --mode=0755 "$target"/var/cache/zypp
#  sln
rm -rf "$target"/sbin/sln
#  ldconfig
rm -rf "$target"/etc/ld.so.cache "$target"/var/cache/ldconfig
mkdir -p --mode=0755 "$target"/var/cache/ldconfig

version=`grep "VERSION=" ${target}/etc/os-release | sed 's/\"//g' | awk -F=
'{print $2}'`

if [ -z "$version" ]; then
    echo >&2 "warning: cannot autodetect OS version, using '$name' as tag"
    version=$name
fi
tar --numeric-owner -c -C "$target" . | docker import - $name:$version
docker run -i -t --rm $name:$version /bin/bash -c 'echo success'

rm -rf "$target"
```

3. Execute the **mkimage-zypp.sh** script to create and import the SLES Docker image:

```
sudo mkimage-zypp.sh slesbase
```

4. Obtain the **slesbase** Docker image's **TAG**. The **slesbase:<TAG>** is required to build the Golang toolchain Docker image:

```
docker images
```

Look for **slesbase** in the REPOSITORY column and note the TAG name for **slesbase**.

In the example below the TAG is 12-SP1.

```
SLES12BC:~ # docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
slesbase                    12-SP1              c81d610377e3        2 minutes ago       406.9 MB
```

NOTE: Optionally, you can place this base image into your Docker registry's repository by issuing the commands:

docker tag slesbase:<TAG> <docker_registry_host_ip>:5050/slesbase:<TAG>
docker push <docker_registry_host_ip>:5050/slesbase:<TAG>

6.2 Build a Golang Toolchain Docker Image from the Base SLES Docker Image

Once the base SLES Docker image is created, complete the following steps to build a Golang toolchain Docker image:

> **NOTE:** The directory **/<work_dir>/** used in these instructions represents a temporary writeable directory of your choice. The same **/<work_dir>/** used in the creating of the Golang toolchain in section Building the Golang Toolchain on page 3 should be used here.

1. Make sure that the Docker Daemon and Docker Registry have been started. Refer to Installing the Docker Client / Daemon and Building the Docker Registry sections above for building and starting the Docker Daemon and Docker Registry.
2. Create a Dockerfile:

   ```
   cd /<work_dir>/
   vi Dockerfile
   ```

   Cut and paste the following lines into your Dockerfile and then save the file:

   ```
   FROM slesbase:<TAG>
   RUN zypper --non-interactive --no-gpg-check in gcc gcc-c++ make git-core
   COPY go /usr/local/go
   ENV GOPATH=/opt/gopath
   ENV GOROOT=/usr/local/go
   ENV PATH=$GOPATH/bin:/usr/local/go/bin:$PATH
   RUN mkdir -p "$GOPATH/src" "$GOPATH/bin" && chmod -R 777 "$GOPATH"
   WORKDIR $GOPATH
   ```

   > **NOTE:** Replace **<TAG>** with the TAG value obtained above in step 4 of Build a Base SLES Docker Image.

3. Make sure that a copy of the **go** directory is in your **/<work_dir>/** directory. If you used the same **/<work_dir>/** directory when performing the instructions in section Building the Golang Toolchain on page 3 then you should already have your built **go** directory contained in **/<work_dir>/**.
4. Issue the **docker build** command:

   ```
   cd /<work_dir>/
   docker build -t <docker_registry_host_ip>:5050/s390x/golang -f <docker_file> .
   ```

   > **NOTE:** Replace **<docker_registry_host_ip>** with the IP address of the host that is running your Docker Registry. Replace **<docker_file>** with **Dockerfile** or the name of your file containing the Docker statements listed in step 2.

5. Confirm that your new image was created by issuing the **docker images** command.

6. Push your new Golang toolchain Docker image to your Docker Registry:

```
docker push <docker_registry_host_ip>:5050/s390x/golang
```

NOTE: Replace **<docker_registry_host_ip>** with the IP address of the host that is running your Docker Registry.

7. Update your Hyperledger Fabric peer's configuration file and save. The following change informs the peer to use your Golang toolchain Docker image when executing Chaincode transactions:

```
cd /<work_dir>/src/github.com/hyperledger/fabric/peer
vi core.yaml
```

NOTE: The **/<work_dir>/** directory was established in Build the Hyperledger Fabric Core on page 9.

Replace the **chaincode.golang.Dockerfile** parameter (located within lines 280-290) with the following:

```
        Dockerfile:  |
            from <docker_registry_host_ip>:5050/s390x/golang
            COPY src $GOPATH/src
            WORKDIR $GOPATH
```

6.3 Build Hyperledger Fabric Docker Images

If you have progressed through this document from the beginning, you already built the components necessary to run the Hyperledger Fabric peer along with the Hyperledger Fabric membership services and security server on your Linux system.

However, if you would like to run your peer(s) or membership services components in their own Docker containers, perform the following steps to build their respective Docker images.

1. Update the Hyperledger Fabric's core.yaml file and save:

```
cd /<work_dir>/src/github.com/hyperledger/fabric/peer
vi core.yaml
```

NOTE: The **/<work_dir>/** directory was established in Build the Hyperledger Fabric Core on page 9.

Replace the **peer.Dockerfile** parameter (located within lines 90-100) with the following:

```
    Dockerfile:  |
        from <docker_registry_host_ip>:5050/s390x/golang
        # Install RocksDB
        RUN cd /opt && git clone --branch v4.5.1 --single-branch --depth 1
https://github.com/facebook/rocksdb.git && cd rocksdb
        WORKDIR /opt/rocksdb
        RUN sed -i -e "s/-march=native/-march=zEC12/"
build_tools/build_detect_platform
        RUN sed -i -e "s/-momit-leaf-frame-pointer/-DDUMBDUMMY/" Makefile
        RUN make shared_lib
        ENV LD_LIBRARY_PATH=/opt/rocksdb:$LD_LIBRARY_PATH
        RUN zypper --non-interactive --no-gpg-check in zlib zlib-devel
libsnappy1 snappy-devel libbz2-1 libbz2-devel
        # Copy GOPATH src and install Peer
        COPY src $GOPATH/src
        RUN mkdir -p /var/hyperledger/db
        WORKDIR $GOPATH/src/github.com/hyperledger/fabric/peer
        RUN CGO_CFLAGS="-I/opt/rocksdb/include" CGO_LDFLAGS="-L/opt/rocksdb -
lrocksdb -lstdc++ -lm -lz -lbz2 -lsnappy" go install && cp
$GOPATH/src/github.com/hyperledger/fabric/peer/core.yaml $GOPATH/bin
```

NOTE: Replace **<docker_registry_host_ip>** with the IP address of the host that is running your Docker Registry.

2. Build the **hyperledger-peer** and **membersrvc** Docker images:

```
cd /<work_dir>/src/github.com/hyperledger/fabric/core/container
go test -timeout=20m -run BuildImage_Peer
go test -timeout=20m -run BuildImage_Obcca
```

NOTE: The **/<work_dir>/** directory was established in Build the Hyperledger Fabric Core on page 9. Both of the images are also built when running the Unit Tests.

3. Verify that the **hyperledger-peer** and **membersrvc** images are displayed after issuing a **docker images** command:

```
docker images
```

7 Unit Tests

If you feel inclined to run the Hyperledger Fabric unit tests, there are a few minor changes that need to be made to some Golang test files prior to invoking the unit tests.

7.1 Test File Changes

1. Edit **/<work_dir>/src/github.com/hyperledger/fabric/membersrvc/ca/ca_test.yaml** and replace the **peer.Dockerfile** parameter with the following:

```
    Dockerfile:  |
        from <docker_registry_host_ip>:5050/s390x/golang
        # Install RocksDB
        RUN cd /opt && git clone --branch v4.5.1 --single-branch --depth 1
https://github.com/facebook/rocksdb.git && cd rocksdb
        WORKDIR /opt/rocksdb
        RUN sed -i -e "s/-march=native/-march=zEC12/"
build_tools/build_detect_platform
        RUN sed -i -e "s/-momit-leaf-frame-pointer/-DDUMBDUMMY/" Makefile
        RUN make shared_lib
        ENV LD_LIBRARY_PATH=/opt/rocksdb:$LD_LIBRARY_PATH
        RUN zypper --non-interactive --no-gpg-check in zlib zlib-devel
libsnappy1 snappy-devel libbz2-1 libbz2-devel
        # Copy GOPATH src and install Peer
        COPY src $GOPATH/src
        RUN mkdir -p /var/hyperledger/db
        WORKDIR $GOPATH/src/github.com/hyperledger/fabric/peer
        RUN CGO_CFLAGS="-I/opt/rocksdb/include" CGO_LDFLAGS="-L/opt/rocksdb -
lrocksdb -lstdc++ -lm -lz -lbz2 -lsnappy" go install && cp
$GOPATH/src/github.com/hyperledger/fabric/peer/core.yaml $GOPATH/bin
```

2. Edit **/<work_dir>/src/github.com/hyperledger/fabric/membersrvc/ca/ca_test.yaml** and replace the **chaincode.golang.Dockerfile** parameter with the following:

```
        Dockerfile:  |
            from <docker_registry_host_ip>:5050/s390x/golang
            COPY src $GOPATH/src
            WORKDIR $GOPATH
```

3. Perform steps 1 and 2 for file **/<work_dir>/src/github.com/hyperledger/fabric/core/ledger/genesis/genesis_test.yaml.**

4. Edit **/<work_dir>/src/github.com/hyperledger/fabric/core/container/controller_test.go** and replace **busybox** with **s390x/busybox**.

NOTE: The **/<work_dir>/** directory was established in Build the Hyperledger Fabric Core on page 9. Replace **<docker_registry_host_ip>** with the IP address of the host that is running your Docker Registry.

7.2 Running the Unit Tests

1. Bring up a window (via ssh or screen) of the system where you built the Hyperledger Fabric components and start the Fabric Peer:

```
cd /<work_dir>/src/github.com/hyperledger/fabric/peer
sudo LD_LIBRARY_PATH=/opt/rocksdb:$LD_LIBRARY_PATH ./peer node start
```

2. From another window of the same Linux system, create an executable script called **unit-tests.sh** in **/<work_dir>/** using the following lines:

```
#!/bin/bash

export GOPATH=/<work_dir>/
export GOROOT=/<golang_home>/go
export PATH=/<golang_home>/go/bin:$PATH
export CGO_LDFLAGS="-L/opt/rocksdb -lrocksdb -lstdc++ -lm -lz "
export CGO_CFLAGS="-I/opt/rocksdb/include"
export LD_LIBRARY_PATH=/opt/rocksdb:$LD_LIBRARY_PATH

go test -timeout=20m $(go list github.com/hyperledger/fabric/... | grep -v
/vendor/ | grep -v /examples/)
```

NOTE: If you have root access and would like to run the unit tests, simply set the environment variables listed above and then issue the go test command. Also, replace **/<work_dir>/** with the directory used when building the Hyperledger Fabric components in Build the Hyperledger Fabric Core on page 9. Replace **/<golang_home>/** with the directory where Golang was installed after performing step 4 in Building the Golang Toolchain on page 3.

3. Invoke the unit-tests.sh script:

```
cd /<work_dir>/
sudo ./unit-tests.sh
```