CLIP Developer Guide

Introduction

This document outlines the steps required to create an environment sufficient for the development of the CLIP application.

After reading this document readers should be able to build and deploy the CLIP application complete with a local database on a local development machine such as a laptop.

Assumptions

This document assumes the user is using Linux and is reasonably familiar with the Bash command line, SSH, Git, Java, Maven, Oracle and Docker.

If you are not using Linux then create a virtual machine using VirtualBox. Give the VM 4 CPUs, 8 GB RAM and at least 100 GB of disk. The more CPUs and RAM you can allocate the better of course, however on most Cisco developer laptops 4 CPUs and 8 GB RAM is the limit you can provision to your local VM. Unfortunately CLIP development with only 8 GB of RAM is becoming more and more difficult.

You will need to create a SSH public key and configure Bitbucket to use it.

Be sure to configure Git to use your full name and email address:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

You will also need Git LFS support. Install it as you would Git.

Prerequisites

To create a local development environment for CLIP the following software is needed:

- Docker 20.10.7 - used to run Oracle in a container and optionally Tomcat
- Eclipse IDE 2020-06 (4.16.0) - the recommended Java IDE to use for now
- Java JDK 1.8 - used for running the CLIP application
- Maven 3.6.3 - used for building
- Oracle 19c 19.3.0 - database for the CLIP application
- Tomcat 9.0 - used for running the CLIP application

Optionally these packages can also be used:

- Oracle SQL Developer 19.2.1.247 - used to connect to the database for ad hoc queries
- Java JDK 11 - used for running the Eclipse IDE and Oracle SQL Developer

The exact version of Docker doesn't matter much, just install the version packaged for your Linux variant. Podman is also and option although scripts we have use Docker so you may need to alias Docker to Podman.

You can run Tomcat 9.0 in Docker instead of installing it directly. There's instructions for that later in this document.

Install Software

Download and install the following packages and ensure all are available on your `PATH`:

- Java JDK 1.8
- Maven 3.6.3
- Tomcat 9.0

For development it is recommended to use the Eclipse IDE 2020-06 (4.16.0) and it will automatically enforce the CLIP coding style. Configure Eclipse to use JDK 1.8 for compiling code and running tests. You can use JDK 11 to actually run Eclipse if you like.

When installing the Java JDK 1.8 ensure it is on the `PATH`. Also ensure the `JAVA_HOME` environment variable is correctly set to the Java JDK 1.8 installation.

Set the `MAVEN_HOME` variable to the Maven installation location. Note that Maven appears to use the `JAVA_HOME` variable to determine which JDK to use for building.

Once everything is installed and configured you should get the following results in response to these commands:

```
$ mvn --version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: /home/schmitzb/opt/maven
Java version: 1.8.0_261, vendor: Oracle Corporation, runtime: /home/schmitzb/opt/jdk1.8.0_261/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.4.0-80-generic", arch: "amd64", family: "unix"

$ java -version
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)
```

Obtain CLIP Code

The CLIP code is available via Git. You may need to be granted access to Bitbucket in order to clone the repository. You will also need to create a SSH public key and configure Bitbucket use it.

The code can be browsed at BitBucket clip-app. It is recommend you fork the repository to avoid pushing directly to the main repo. In the future this may be required.

To clone the code execute:

```
git clone ssh://git@gitscm.cisco.com/asf/clip-app.git
```

The database import/export scripts are at BitBucket clip-db. Note that this clip-db is about 5 GB and uses the Git Large File Storage, although that should be transparent when cloning.

To clone execute:

```
git clone ssh://git@gitscm.cisco.com/clpsvs/clip-db.git
```

You should not need to do anything for the Git LFS support as long as your version of Git and Git LFS is recent. If you do have trouble with Git LFS I recommend using a more recent Linux version with updated Git and Git LFS versions.

Building

First in `clip-app/clip` checkout the `develop` branch:

```
git checkout develop
```

Before building make a copy of the `clip-app/clip/src/main/resources/clip-local.properties.example` file:

```
cp src/main/resources/clip-local.properties.example src/main/resources/clip-local.properties
```

Edit `clip-app/clip/src/main/resources/clip-local.properties` and set the properties to values suitable for your local environment. Specifically ensure the keys `mail.from`, `mail.non.prod.alias` and `domain.url` are set to sensible values.

If you want to use a local Tomcat instance then copy the file `clip-app/src/docs/context.xml` to the `clip-app/clip/src/main/webapp/META-INF` directory. This is not needed if you run Tomcat using Docker. This file configures the database connection pool in Tomcat. Edit it to look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>

    <Resource name="jdbc/clip" auth="Container" type="javax.sql.DataSource"
        factory="org.apache.tomcat.jdbc.pool.DataSourceFactory" maxActive="20"
        maxIdle="10" maxWait="5000" username="clip0" password="password"
        driverClassName="oracle.jdbc.driver.OracleDriver" url="jdbc:oracle:thin:@//clip-db:1521/CLIPDB1" />

    <!-- disable session persistance between tomcat restarts -->
    <Manager pathname="" />

    <Valve className="org.apache.catalina.authenticator.BasicAuthenticator"
        securePagesWithPragma="false" />

</Context>
```

The code is built using Maven. Execute in the `clip-app/clip` directory:

```
mvn clean package
```

This will take about 6.5 minutes on average. The first execution will download all Maven dependencies and will take significantly longer.

The resulting WAR file will be under `clip-app/clip/target`.

To skip tests define the `maven.test.skip` property:

```
mvn -Dmaven.test.skip clean package
```

To skip building the UI you can activate the `skipUI` Maven profile:

```
mvn -PskipUI clean package
```

Note that this will only work properly if you have already built the UI once. When the UI is built it is cached and can be re-used upon subsequent builds. This can dramatically speed up building during development.

To skip the static analysis checks activate the `skipStaticAnalysis` Maven profile:

```
mvn -Pui,skipStaticAnalysis clean package
```

Note that when manually activating Maven profiles you need to activate all the profiles you need as any default profiles will not be activated. Hence above we activate the default `ui` profile to ensure the UI is also built.

You can skip specific static analysis checks using several properties:

```
mvn -Dcheckstyle.skip -Dpmd.skip -Dfindbugs.skip clean package
```

To build quickly skip the UI build and static analysis and all tests:

```
mvn -PskipUI,skipStaticAnalysis -Dmaven.test.skip clean package
```

This completes in about 30 seconds for me and is the recommend approach for normal development.

The fastest possible build can be done by omitting the `clean` target and combining all the options above:

```
mvn -PskipUI,skipStaticAnalysis -Dmaven.test.skip package
```

This completes in 10 seconds or so depending on how many Java files need to be compiled. In this case the build artifact uses the previously built UI, previously generated artifacts are not deleted and only those Java files that have changed are re-compiled. This can be dangerous and I would do this for very long as you run the risk of including broken or invalid classes in the build artifact.

Of course *always* do a full build before pushing any new commits.

Build using Docker Maven container (optional)

It's also possible to build using Maven in a Docker container:

```
docker run --rm -it --user=$(id -u):$(id -g) -v $HOME/.m2:/var/maven/.m2   -v $(pwd)/..:/clip-app -w /clip-app/clip -e MAVEN_CONFIG=
```

This will fetch the Maven Docker image if needed. Maven is run as the current user in the container and the container is given access to the users local Maven repo in `~/.m2`. Also in this example we are skip building the UI, running static analysis and running tests.

Using Maven in a Docker container makes it relatively easy to test newer versions of Maven to ensure they work.

Configure Oracle database

Install Oracle Software

It is recommend to install Oracle SQL Developer 18.4.0 for connecting to the database and executing ad hoc SQL. You can choose the package with the bundled JRE for convenience.

The CLIP application requires an Oracle 19c database, specifically version 19.3.0. For local development we can use Docker to simplify the installation and use of Oracle. Install Docker 18.09.2 or better using whatever method is appropriate for your operation system.

Build Oracle Docker Image

Once Docker is installed we need to create the Oracle images for use with Docker. Clone the official Oracle Docker image repo from GitHub:

```
git clone https://github.com/oracle/docker-images
```

Navigate to `docker-images/OracleDatabase/SingleInstance` and follow the instructions in the `README.md` to create the Oracle Docker image. You will need to download the official Oracle installation binaries from the Oracle Technology Network. Be sure to get version Oracle Database 19c 19.3:

- `LINUX.X64_193000_db_home.zip`

Once you have downloaded the Oracle binary and copied it to the `docker-images/OracleDatabase/SingleInstance/dockerfiles/19.3.0` directory, execute the following from the `docker-images/OracleDatabase/SingleInstance/dockerfiles` directory:

```
./buildDockerImage.sh -v 19.3.0 -e
```

Once this is complete you should have an Oracle Docker image:

```
docker images
REPOSITORY          TAG         IMAGE ID            CREATED         SIZE
oracle/database         19.3.0-ee           a72ce00a8d1c            About an hour ago           6.51GB
```

Run Oracle in Docker Container

Now that we have the Oracle Docker image available we can run it and create a database in a container.

First we create a local Docker network to ease database connectivity later:

```
docker network create clip-net
```

Then create a named Docker volume to store our database files:

```
docker volume create clip-db-oradata
```

Now create a Docker container running Oracle. Upon the first run a new Oracle database will be created:

```
docker run \
    --name clip-db \
    --net=clip-net \
    -p 1521:1521 \
    -p 5500:5500 \
    -e ORACLE_SID=CLIPDB \
    -e ORACLE_PDB=CLIPDB1 \
    -e ORACLE_PWD=password \
    -e ORACLE_CHARACTERSET=UTF8 \
    -e TZ="America/Los_Angeles" \
    -v clip-db-oradata:/opt/oracle/oradata  \
    oracle/database:19.3.0-ee
```

Note that we are overriding the database SID and PDB and character set. We also set the default password to the SYS and SYSTEM users to 'password' which is fine as this is for local development use only. The database service name is CLIPDB1 and is available at localhost on port 1521 and the Enterprise Manager interface is at https://localhost:5500/em.

Once the above command is complete it will continue to run and show the Oracle log. You can use control-c to shutdown the Oracle container.

For day-to-day development you would start the database with this command:

```
docker start clip-db
```

and stop it with this command:

```
docker stop clip-db
```

At this stage with the database running you should be able to connect to the SYSTEM account using say SQL developer. A suitable connection string is `SYSTEM/password@localhost:1521/CLIPDB1`. Note that we use a service name and not a SID to connect.

Run Oracle SQL*Plus from Docker container

With the Oracle Database Docker container up and running you should be able to also use the Oracle Database Docker image to run SQL*Plus and connect to the database:

```
docker run --rm -it --net=clip-net  -e NLS_LANG=AMERICAN_AMERICA.UTF8  oracle/database:19.3.0-ee   sqlplus SYSTEM/password@clip-db/C

SQL*Plus: Release 12.1.0.2.0 Production on Tue May 28 17:59:26 2019

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> select systimestamp from dual;

SYSTIMESTAMP
---------------------------------------------------------------------------
28-MAY-19 10.59.35.087339 AM -07:00

SQL> exit
Disconnected from Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
```

Create CLIP Tablespace and Schema

Now we need to create a tablespace and database users. A suitable SQL can be found in the file `clip-app/clip/src/db/misc/local-clip-db.sql` in the CLIP code directory:

```
-- local clip schema

alter profile default limit password_life_time unlimited;

-- disable recyclebin
show parameter recyclebin;
alter system set recyclebin=off deferred;



-- create tablespace for clip
-- note that it's 16 GB max and will not grow
-- it can be manually extended if needed
create bigfile tablespace clip_local
datafile '/opt/oracle/oradata/CLIPDB/CLIPDB1/clip_local.dbf'
size 16g autoextend off;

-- create three clip schemas
create user clip0 identified by password default tablespace clip_local;
grant connect, resource, dba to clip0;
grant create session to clip0 with admin option;
grant unlimited tablespace to clip0;

create user clip1 identified by password default tablespace clip_local;
grant connect, resource, dba to clip1;
grant create session to clip1 with admin option;
grant unlimited tablespace to clip1;

create user clip2 identified by password default tablespace clip_local;
grant connect, resource, dba to clip2;
grant create session to clip2 with admin option;
grant unlimited tablespace to clip2;

-- create clip_bi
-- note that unlike the real clip_bi this local one is not read-only
create user clip_bi identified by password default tablespace clip_local;
grant connect, resource, dba to clip_bi;
grant create session to clip_bi with admin option;
grant unlimited tablespace to clip_bi;

-- create clip_ro
-- note that unlike the real clip_ro this local one is not read-only
create user clip_ro identified by password default tablespace clip_local;
grant connect, resource, dba to clip_ro;
grant create session to clip_ro with admin option;
grant unlimited tablespace to clip_ro;



commit;

exit;
```

This SQL script first ensures that passwords will never expire and explicitly disabled the recyclebin.

A tablespace is created. Note the location of the tablespace `/opt/oracle/oradata/CLIPDB/CLIPDB1/clip_local.dbf`. The directory `CLIPDB` and `CLIPDB1` correspond to the SID and PDB values provided when creating the database originally. Note that this tablespace is 16 GB in size and will not automatically grow. It will need to be manually extended if needed.

Three users are created `clip0`, `clip1` and `clip2`. Having three schemata allows one schema for the current development branch, one for the master branch and one in which to run the database unit tests. Of course more can be created as needed.

Execute this SQL script from the `clip-app/clip` directory as the SYSTEM user using `sqlplus` running in a Oracle Client Docker container:

```
$ docker run --rm -it --net=clip-net --user=$(id -u):$(id -g) -v $(pwd):/clip -w /clip -e NLS_LANG=AMERICAN_AMERICA.UTF8  oracle/dat



SQL*Plus: Release 19.0.0.0.0 - Production on Mon Jun 29 21:19:42 2020
Version 19.3.0.0.0
```

```
Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Last Successful login time: Mon Jun 29 2020 21:12:58 +00:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0


Profile altered.


NAME                      TYPE     VALUE
--------------------------------- ----------- ------------------------------
recyclebin                  string     on

System altered.


Tablespace created.


User created.


Grant succeeded.


Grant succeeded.


Grant succeeded.


User created.


Grant succeeded.


Grant succeeded.


Grant succeeded.


User created.


Grant succeeded.


Grant succeeded.


Grant succeeded.


User created.


Grant succeeded.


Grant succeeded
```

```
Grant succeeded.


Grant succeeded.


User created.


Grant succeeded.


Grant succeeded.


Grant succeeded.


Commit complete.

Disconnected from Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0
```

Import CLIP Database

Navigate to `clip-db/iwe-clip` which contains the latest production database export.

We will use the existing Oracle Database Docker image to gain access to the `sqlplus` and `sqlldr` utilities in order to import the database data. We will also set the uid and gid of the `db-import.sh` process in the container to be the same as the current user to ensure the container is able to write to the `clip-db/iwe-clip` directory.

Run the following to import the database:

```
time docker run --rm -it --net=clip-net --user=$(id -u):$(id -g) -v $(pwd):/import -w /import  oracle/database:19.3.0-ee   ./db-impo
```

The import should take between 5 and 10 minutes. Eventually you should see something like this:

```
    _       _____ _      _____ _____    _____  ____    ______                    _       _
 /\| |/\   / ____| |    |_   _|  __ \  |  __ \|  _ \  |  ____|                  | |    /\| |/\
 \ ` ' /  | |    | |      | | | |__) | | |  | | |_) | | |__  __  ___ __   ___  _ __| |_   \ ` ' /
|_    _| | |    | |      | | |  ___/  | |  | |  _ <  |  __| \ \/ / '_ \ / _ \| '__| __|  |_    _|
 / , .  \  | |____| |____ _| |_| |      | |__| | |_) | | |____ >  <| |_) | (_) | |  | |_   / , .  \
 \/|_|\/   \_____|______|_____|_|      |_____/|____/  |______/_/\_\ .__/ \___/|_|   \__|  \/|_|\/
                                                                   | |
                                                                   |_|

Import completed in 567 seconds

Import script output is logged in the file db-import.log


If this is dev, stage or LT you should generate statistics and update the grants (run stats.sql and grants.sql)

No errors!  Unbelievable!




Done
```

The output is also sent to the `db-import.log` file. You can check that for errors:

```
grep '^\(ORA\|SP2\)' db-import.log
```

Oracle error codes begin with 'ORA'. Sql*plus errors or other parsing errors start with 'SP2'. These error codes generally appear at the beginning of the line.

You should get no output when running the above command. *Any output will be an error and should be investigated. It is best not to continue until the import completes without error.*

The very first time you run this in a schema where you have *never imported into before* you may get this error which you can ignore or just re-run the import a second time and it should complete without error:

```
ORA-20000: Oracle Text error:
ORA-06512: at "CTXSYS.DRUE", line 186
ORA-06512: at "CTXSYS.DRVLSB", line 50
ORA-06512: at "CTXSYS.CTX_DDL", line 45
ORA-06512: at line 1
```

Upgrade CLIP Database

The `develop` branch may have associated database changes which should be applied to the local database before executing the code. Navigate to `clip-app/clip/src/db/scripts/release` and execute the `run-db-upgrade.sh` script in the local database:

```
$ ./run-db-upgrade.sh

...  output not shown ...
```

Check for errors

```
$ grep '^\(ORA\|SP2\)' errors.txt
ORA-01927: cannot REVOKE privileges you did not grant
ORA-06512: at line 3
ORA-06512: at line 3
```

The errors above can generally be ignored. Any other errors should be investigated. *Do not continue if there are errors other than what is shown above.* Feel free to scold someone in the development team as needed.

Deploy Locally into Tomcat using Docker

Build Tomcat Docker Image

We can use a Tomcat Docker image to run the CLIP application. Navigate to `clip-app/clip/docker/tomcat9`. Edit the `tomcat-users.xml` file and add in your own username and any other usernames you may need. Passwords for local development are all `password`. If you need additional users in the future you will need to re-build the Docker image.

Note that this Tomcat 9 image does not use SSL.

Now build the Tomcat Docker image:

```
docker build --no-cache \
    -t clip-tomcat-no-ssl:v9.0.0 \
    -f Dockerfile .
```

This will build a Docker image of Tomcat 9.

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
clip-tomcat-no-ssl  v9.0.0              9109d56e7863        7 months ago        464MB
```

Deploy CLIP into Tomcat using Docker

Now we can deploy CLIP into Tomcat running as a Docker container. Navigate to `clip-app/clip` and run:

```
docker run --name=tomcat --net=clip-net \
    --rm  -it
    -p 8080:8080 \
    -p 8787:8787 \
    -p 9010:9010 \
    -v "$(pwd)/target/clip-app-50.0-SNAPSHOT.war:/usr/local/tomcat/webapps/services#clip.war" \
    clip-tomcat-no-ssl:v9.0.0
```

Replace the version number of the 'war' file as needed in the command above.

Note that the container is started listening for HTTP requests on 8080. SSL is not enabled and basic authentication is used by default.

You can remotely debug from Eclipse or another IDE by connecting to port 8787.

JMX is configured to use port 9010 so you can connect `jconsole` to see JMX details or `jvisualvm` for memory or CPU profiling.

Browse to http://localhost:8080/clip-app/main/app/home to access the application and http://localhost:8080/clip-app/main/app/dev/view for the admin pages. Yay!

Docker will output the Tomcat log. Type `control-c` to quit the Tomcat Docker container.