Contents

[Minimal (No GUI) CentOS Development Environment Preparation steps for Medyasoft PRODIGY 1](#_Toc512854298)

[External Libraries and MEDYASOFT Prodigy sample installation 3](#_Toc512854299)

# Minimal (No GUI) CentOS Development Environment Preparation steps for Medyasoft PRODIGY

* Install Minimal CentOS 7

Download and install

<http://isoredirect.centos.org/centos/7/isos/x86_64/CentOS-7-x86_64-Minimal-1708.iso>

Note : For virtual environment support IP connectivity both with host and internet

(VirtualBox : <https://serverfault.com/questions/225155/virtualbox-how-to-set-up-networking-so-both-host-and-guest-can-access-internet>)

* Upgrade the OS

Login as root.

$ yum upgrade

* Install fundamental development tools

Login as root.

$ yum groupinstall 'Development Tools'

$ yum install cmake

$ yum install readline-devel

$ yum install openssl-devel

$ yum install vim

$ yum install wget

* Configure git

$ git config --global user.name "Your Name"

$ git config --global user.email [you@example.com](mailto:you@example.com)

* Stop Firewall

$ systemctl stop firewalld

$ systemctl disable firewalld

* Install SFTP & SAMBA & SSH Connectivity

Login as root

$ yum install vsftpd

<https://www.liquidweb.com/kb/how-to-install-and-configure-vsftpd-on-centos-7>

$ yum install samba samba-client samba-common

<https://www.howtoforge.com/samba-server-installation-and-configuration-on-centos-7>

Connect via SSH & FTP from host (windows)

* Create a postgres/postgres user/pwd

$ useradd postgres

$ passwd postgres

-> postgres

$ usermod -g wheel postgres # (add user to sudors goup)

* Clone PostgreSQL sources

Login as postgres

git://git.postgresql.org/git/postgresql.git (branch:)

$ su postgres

$ cd ~

$ git clone -b REL\_10\_STABLE git://git.postgresql.org/git/postgresql.git

* Build, install & start PostgreSQL

<https://www.postgresql.org/docs/current/static/install-procedure.html>

$ su postgres

$ cd ~/postgresql

$ mkdir build

$ cd build

$ ../configure [options go here]

$ make

$ sudo make install

* Prepare PostgreSQL Server

add to .bash\_profile

export LD\_LIBRARY\_PATH=/usr/local/pgsql/lib

export PATH=/usr/local/pgsql/bin:$PATH

Prepare DB Data files.

$ sudo mkdir /usr/local/pgsql/data

$ sudo chown postgres /usr/local/pgsql/data

$ initdb -D /usr/local/pgsql/data --pwprompt

edit /usr/local/pgsql/data/postgresql.conf

listen\_addresses = '192.168.56.101' # (--> Server IP)

port = 5432

edit /usr/local/pgsql/data/pg\_hba.conf

host all all 192.168.56.0/24 trust

Start PostgreSQL Server

$ pg\_ctl -D /usr/local/pgsql/data -l ~/logfile start

Reload PostgreSQL server

$ pg\_ctl -D /usr/local/pgsql/data -l ~/logfile restart

or

$ pg\_ctl -D /usr/local/pgsql/data -l ~/logfile start

$ pg\_ctl -D /usr/local/pgsql/data -l ~/logfile stop

# External Libraries and MEDYASOFT Prodigy sample installation

* Clone & build Apache ORC Libray

$ cd ~

$ git clone -b branch-1.4 https://github.com/apache/orc.git

$ mkdir orc/build

$ cd orc/build

$ cmake -D CMAKE\_CXX\_FLAGS="${CMAKE\_CXX\_FLAGS} -fPIC" -D CMAKE\_C\_FLAGS="${CMAKE\_C\_FLAGS} -fPIC" -D BUILD\_JAVA=OFF ..

$ make package test-out

Useful examples are located in orc/tools/src/

***Note:***

The changes below have to do to be able to build and execute Postgres SQL C++ extensions with the ORC library.

$ diff /usr/local/pgsql/lib/pgxs/src/Makefile.port /usr/local/pgsql/lib/pgxs/src/Makefile.port.orginal

15c15

< $(CC) **$<** $(CFLAGS) $(LDFLAGS) $(LDFLAGS\_SL) -shared -o $@

---

> $(CC) $(CFLAGS) $(LDFLAGS) $(LDFLAGS\_SL) -shared -o $@ **$<**

* Clone & build Medyasoft Sample Skeleton Projects
  + Simple PostgreSQL Extension

This is a primitive PostgreSQL Extension project with only one C function. All implementation are located in the skeleton\_extension/base36/base36.cpp source file.

$ mkdir ~/projects

$ cd ~/projects

$ git clone https://medyasoft\_prodigy@bitbucket.org/medyasoft/filikadb.git

$ cd filikadb/doc/dev/skeleton\_extension/base36/

$ sudo make install

$ make installcheck # Unit Tests

* + Simple PostgreSQL extension with ORC

This extension which uses the library from orc.apache.org to manipulate ORC files. Two extension function which displays information and the content of a given ORC file.

Note: Be sure that the file path in skeleton\_ext\_orc/sql/ext\_orc\_test.sql are correctly and existed in your environment.

$ mkdir ~/projects

$ cd ~/projects

$ git clone https://medyasoft\_prodigy@bitbucket.org/medyasoft/filikadb.git

$ cd filikadb/doc/dev/skeleton\_ext\_orc

$ sudo make install

$ make installcheck # Unit Tests

* + Simple PostgresSQL Foreign Data Wrapper (FDW)

***THIS PROJECTS IS INCOMPLETE. Only “select” and “update” SQL commands are implemented.***

$ mkdir ~/projects

$ cd ~/projects

$ git clone https://medyasoft\_prodigy@bitbucket.org/medyasoft/filikadb.git

$ cd filikadb/doc/dev/flowlog\_fdw

$ sudo make install

This project encapsulate PostgreSQL’s Foreign Data Wrapping (FDW) feature.

This FDW implementation is based on the Black Hole FDW <https://bitbucket.org/adunstan/blackhole_fdw/src/master> project.

A primitive Log infrastructure is also added to this FDW implementation to be able to monitor and analyse the FDW events with its parameters. The flowlog\_fdw\Logger\LoggerReceiver is a stand alone executable project that receives the log entries and displays them to the std output. The receiver and FDW are configured to use the “/home/postgres/logsocket” domain socket. Be sure that these applications have read/write permissions to socket path.

The SQL commands commands below are required to load this FDW.

create extension flowlog\_fdw;

create server flowlog\_db foreign data wrapper flowlog\_fdw options (nameText1 'valueText1', nameText2 'valueText2');

create user mapping for postgres server flowlog\_db;

create foreign table flowlog\_tb

(

v1 integer,

v2 text,

v3 integer

) server flowlog\_db;

The “nameText1”, “valueText1”, “nameText2”, “valueText2” are dummy variable and are displayed in the flowlog\_fdw Logger.

This FDW sends dummy data to the PostgreSQL server on a “select” command and logs the requested parameters on an “update” SQL command.

Dummy records are prepared and returned to PostgresSQL server in the FDW event hadler (Callback Function) below

static TupleTableSlot \*flowlogIterateForeignScan(ForeignScanState \*node)

The required parameters of an “update” statement is handled and logged in the function below.

std::string text\_enum(TupleTableSlot \*slot);

This function demonstrate how to get the new record from an “update” sql statement.