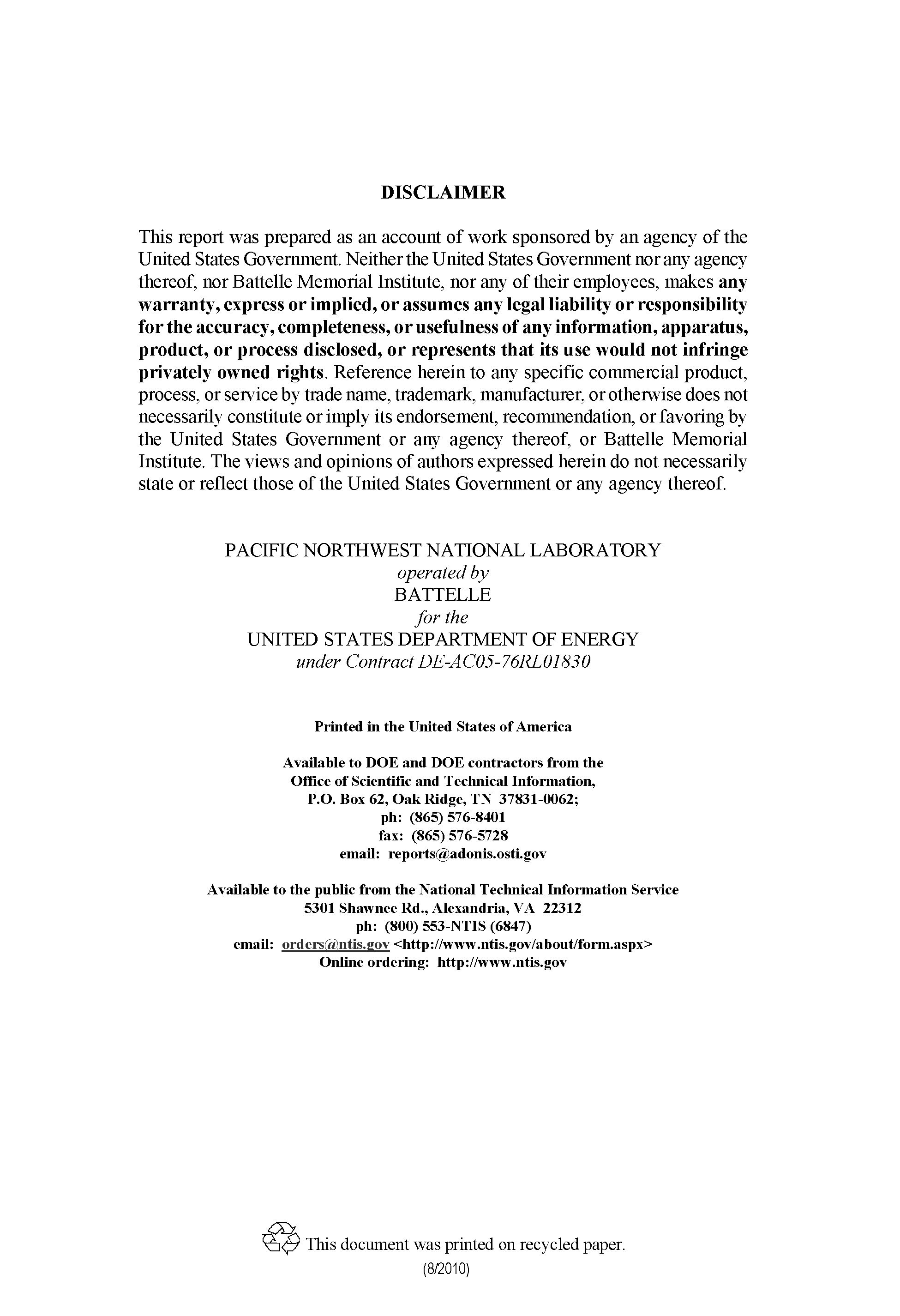
FLOWER Build Guide

FLOWER Version 06 (flr06)

September 2018

DS Curtis

LMS Curtis



FLOWER Build Guide

DS Curtis

LMS Curtis

September 2018

Prepared for

the U.S. Department of Energy

under Contract DE‑AC06‑76RL01830

with Battelle Memorial Institute

Pacific Northwest National Laboratory

Richland, Washington 99352

Acknowledgments

Copyright © (2011-2021) Battelle Memorial Institute. All Rights Reserved. PNNL-SA-120800

Battelle IPID: 17042-E

Prime Contract No.: DE-AC06-76RL01830 with Battelle Memorial Institute

Acronyms and Abbreviations

FLOWER FLOW analyzer

GCC GNU Compiler Collection

PNNL Pacific Northwest National Laboratory

QSFP Quad Small Form-factor Pluggable

RPM RPM Package Manager

Contents

[Acknowledgments iii](#_Toc524004847)

[Acronyms and Abbreviations v](#_Toc524004848)

[1.0 Introduction 1](#_Toc524004849)

[2.0 Requirements 2](#_Toc524004850)

[2.1 Hardware 2](#_Toc524004851)

[2.2 Operating System 2](#_Toc524004852)

[2.3 Developer Environment 2](#_Toc524004853)

[2.3.1 User development environment (RHEL/CentOS 64-bit) 2](#_Toc524004854)

[2.3.2 PCAP development and runtime libraries 2](#_Toc524004855)

[2.3.3 GNU C++11 Compiler 3](#_Toc524004856)

[2.3.4 Boost C++ libraries 4](#_Toc524004857)

[3.0 Building FLOWER 5](#_Toc524004858)

[4.0 Testing FLOWER 6](#_Toc524004859)

[5.0 Building RPMS 9](#_Toc524004860)

[5.1.1 Build FLOWER RPM 9](#_Toc524004861)

[6.0 Installing RPMs 10](#_Toc524004862)

# Introduction

The FLOWER (FLOW analyzER) software was developed at PNNL and is intended for deployment on application sensors.

This guide is intended to assist an experienced developer to build and install the FLOWER software to deploy on a FLOWER appliance sensor. The instructions are written in a format that assumes a basic knowledge of Linux and C/C++ development. Short instructions are provided for each step along with command line or script references to complete the build. If you have any questions about the build, please contact FLOWER support at flower-support@pnnl.gov.

The FLOWER application is designed to start at boot time as a systemd service to summarize network flows.

FLOWER can also be used to read packets from a network interface or from packet capture (pcap) files that can be processed with the libpcap system library. See the FLOWER Operations Guide document for more detailed information.

# Requirements

## Hardware

The FLOWER application requires a 64-bit Intel based system with dual quad-core processors with a minimum of 4 GB of memory and 20 GB of disk.

The system requires a 1 Gigabit or better Ethernet interface. For testing, it is recommended to have an additional 1 Gigabit or better Ethernet interface that is connected to a dedicated hub or switch. The hub/switch is needed to give the Ethernet interface an active link that can be used in isolation for testing. FLOWER has been tested and verified on an Intel XL710 40 Gigabit QSFP+ interface without any problems.

## Operating System

The FLOWER application can be built and installed on Red Hat Enterprise Linux (RHEL) or CentOS 5.2 or newer.

* It is recommended that you build and install on RHEL/CentOS 7.x.

The Operating System can be installed to run as 32-bit only, 32-bit/64-bit mixed, or 64-bit only mode. You must build on a system that runs the same mode as the system you will be running the application on.

* It is recommended that you build only 64-bit mode. The previous 32-bit modes are for legacy systems only.

## Developer Environment

The FLOWER application requires a Linux development environment, a C++ compiler and Boost libraries. It is required to use at least the versions listed below or better.

### User development environment (RHEL/CentOS 64-bit)

To ensure that you have all the necessary development tools, install the standard development environment using the following command:

# yum groupinstall "Development Tools"

# yum install cmake3

# yum install cmake3-data

### PCAP development and runtime libraries

The PCAP development and runtime libraries are required to build FLOWER. To install them on an RHEL/CentOS system, run the following commands:

# yum install libpcap-devel libpcap

### GNU C++11 Compiler

#### Option 1: Automated Build

The FLOWER source code directory contains a script to download and build GCC. It should work in most environments. The script is located in:

$HOME/git/flower/support/build/gcc/gcc-build.sh

**NOTE**: Refer to Building FLOWER to get the source code to access the gcc-build.sh script.

#### Option 2: Manual Build

Download GNU C and C++ version 8.1 from <https://gcc.gnu.org/releases.html>.

**NOTE**: While newer versions are available, FLOWER has been tested extensively with version GNU C++ version 8.1, so it is the recommended version.

**NOTE**: Do not use the default compiler with RHEL/CentOS because a modern C++ compiler is required for FLOWER to properly build.

Build the GNU gcc/g++ compiler tools and install them in the directory, /usr/local/gcc-8.1, using the following options to configure:

--prefix=/usr/local/gcc-8.1

--disable-multilib

--enable-threads=posix

--enable-\_\_cxa\_atexit

--enable-clocale=gnu

--enable-checking=release

--enable-languages=c,c++

--program-suffix=-8.1

### Boost C++ libraries

#### Option 1: Automated Build

The FLOWER source code directory contains a script to download and build the Boost libraries. It should work in most environments. The script is located in:

$HOME/git/flower/support/build/boost/boost-build.sh

**NOTE**: Refer to Building FLOWER to get the source code to access the boost-build.sh script.

#### Option 2: Manual Build

Download Boost version 1.67 from <http://www.boost.org/users/history>.

**NOTE**: While newer versions may be available, FLOWER has been tested extensively with version 1.67, so this is the recommended version.

**Important**: Ensure that you are using the GCC 8.1 compiler tools and not the built in C compiler tools. To ensure that you are using the proper compiler, do the following:

* From the boost\_1\_67\_0 directory, run the following command:

# echo "using gcc : 8.1 : /usr/local/gcc-8.1/bin/g++-8.1 ; " >> tools/build/src/user-config.jam

* Make sure you have the --toolset=gcc defined when you run your b2 or bjam command.
* Set the following environmental variables:

unset SSH\_ASKPASS

export PATH=/usr/local/gcc-8.1/bin:$PATH

export CC=/usr/local/gcc-8.1/bin/gcc-8.1

export CXX=/usr/local/gcc-8.1/bin/g++-8.1

* Set the build parameters to include:

link=shared

address-model=64

runtime-link=shared

threading=multi

* Build the libraries and install them in /usr/local/gcc-8.1\_boost-1.67.0.

# Building FLOWER

This section contains the instructions that build the FLOWER software. It assumes that all requirements in the previous section have been met, including the GNU C and C++ compiler tools and Boost libraries are built and installed in /usr/local.

Download FLOWER 06 or greater from <https://stash.pnl.gov> and checkout the correct branch.

# mkdir -p $HOME/git

# cd $HOME/git

# git clone https://$USER@stash.pnnl.gov/scm/flower/flower.git

# cd $HOME/git/flower

# git checkout [ master | develop | rpmbuild ]

**Important**: Ensure that the correct version of GCC executables are in your path.

Set the following environmental variables:

unset SSH\_ASKPASS

export PATH=/usr/local/gcc-8.1/bin:$PATH

**NOTE**: You probably want to add the following aliases for cmake in your ~/.bashrc:

alias cmake="cmake3"

alias ctest="ctest3"

alias cpack="cpack3"

Build FLOWER using cmake:

# cd $HOME/git/flower

# cmake .

# make -j 4

The output of make should produce the following files (targets):

flower

libflowerapi.a

t\_bg\_PacketRinger\_root\_loop\_buffer

t\_bg\_PacketRinger\_root\_loop\_track

t\_bg\_PacketRinger\_root\_noloop

t\_bg\_PacketRinger\_user\_noloop

t\_Data

t\_Elf

t\_Event

t\_Flow

t\_FlowCache

t\_Ip

t\_ObjectPool

t\_Packet

t\_PacketBuilder

t\_PacketParser

t\_ProgramOptions

t\_sudo\_PacketBuilder

t\_SummaryExporter

# Testing FLOWER

After FLOWER is built and installed, test the application as follows:

* Add flower to your path

# export PATH=/opt/flower/bin:$PATH

* Make sure the executable runs without error.

# flower –v

The output should show detailed version information, including the version of FLOWER, the version of GCC, boost, and pcap that FLOWER was built with and the build date.

* View the FLOWER help information.

# flower –h

This should show the FLOWER help information.

* Run the FLOWER test suite

Identify a network interface that is up to run the tests against (e.g. eth0)

Set the network interface variable for the test suite

# export FLOWER\_NIC=eth0

Run the test suite from the source code directory

# ctest .

You should see output similar to the following:

Test project $HOME/git/flower

Start 1: t\_bg\_PacketRinger\_root\_loop\_buffer\_test

1/17 Test #1: t\_bg\_PacketRinger\_root\_loop\_buffer\_test ...\*\*\*Failed 0.00 sec

Start 2: t\_bg\_PacketRinger\_root\_loop\_track\_test

2/17 Test #2: t\_bg\_PacketRinger\_root\_loop\_track\_test ....\*\*\*Failed 0.00 sec

Start 3: t\_bg\_PacketRinger\_root\_noloop\_test

3/17 Test #3: t\_bg\_PacketRinger\_root\_noloop\_test ........\*\*\*Failed 0.00 sec

Start 4: t\_bg\_PacketRinger\_user\_noloop\_test

4/17 Test #4: t\_bg\_PacketRinger\_user\_noloop\_test ........ Passed 0.00 sec

Start 5: t\_sudo\_PacketBuilder\_test

5/17 Test #5: t\_sudo\_PacketBuilder\_test ................. Passed 0.01 sec

Start 6: t\_Data\_test

6/17 Test #6: t\_Data\_test ............................... Passed 0.00 sec

Start 7: t\_Elf\_test

7/17 Test #7: t\_Elf\_test ................................ Passed 0.00 sec

Start 8: t\_Event\_test

8/17 Test #8: t\_Event\_test .............................. Passed 0.00 sec

Start 9: t\_FlowCache\_test

9/17 Test #9: t\_FlowCache\_test .......................... Passed 0.01 sec

Start 10: t\_Flow\_test

10/17 Test #10: t\_Flow\_test ............................... Passed 0.00 sec

Start 11: t\_Ip\_test

11/17 Test #11: t\_Ip\_test ................................. Passed 0.00 sec

Start 12: t\_ObjectPool\_test

12/17 Test #12: t\_ObjectPool\_test ......................... Passed 0.00 sec

Start 13: t\_PacketBuilder\_test

13/17 Test #13: t\_PacketBuilder\_test ...................... Passed 0.01 sec

Start 14: t\_Packet\_test

14/17 Test #14: t\_Packet\_test ............................. Passed 0.00 sec

Start 15: t\_PacketParser\_test

15/17 Test #15: t\_PacketParser\_test ....................... Passed 0.00 sec

Start 16: t\_ProgramOptions\_test

16/17 Test #16: t\_ProgramOptions\_test ..................... Passed 0.02 sec

Start 17: t\_SummaryExporter\_test

17/17 Test #17: t\_SummaryExporter\_test .................... Passed 0.03 sec

82% tests passed, 3 tests failed out of 17

Total Test time (real) = 0.12 sec

The following tests FAILED:

1 - t\_bg\_PacketRinger\_root\_loop\_buffer\_test (Failed)

2 - t\_bg\_PacketRinger\_root\_loop\_track\_test (Failed)

3 - t\_bg\_PacketRinger\_root\_noloop\_test (Failed)

Errors while running CTest

**NOTE**: There should be 3 tests that fail because they need to be run as root. You can run each of the 3 tests as root to verify that they pass.

If any of the other tests do not work, the build did not complete correctly. If there are problems, potential areas to check are as follows:

* Ensure that the correct version of GCC is being used.
* Ensure that the correct version of Boost is being used.
* Ensure that the correct version of GCC was used during the build of Boost.
* Ensure that the PCAP and PCAP-devel libraries were installed during the build.

# Building RPMS

This section includes the instructions to build RPMS.

Prior to building, ensure that you have the RPM build utilities installed on your machine.

# yum install rpm-build

# yum install rpm-build-libs

### Build FLOWER RPM

Prepare the FLOWER components in preparation to package.

**NOTE**: The version number 6.0 is used as an example, but the currently build and release number should be substituted as future versions are available.

* Build the FLOWER software as instructed in [Building FLOWER](#_Building_FLOWER).
* Create the flower BUILD directory

# export FLOWER\_RPMBUILD=$HOME/git/flower/support/build/rpmbuild

# cd $FLOWER\_RPMBUILD/BUILD

# cp -R flower-x.x flower-6.0

* Copy the flower binary into the BUILD direcctory

# cp $HOME/git/flower/flower $FLOWER\_RPMBUILD/BUILD/flower-6.0/bin

* Copy the flower documentation into the BUILD direcctory

# cd $HOME/git/flower/docs

# cp FLOWER\_Data\_Guide\_FLR06.docx \

FLOWER\_Install\_Guide\_FLR06.docx \

FLOWER\_Ops\_Guide\_FLR06.docx \

$FLOWER\_RPMBUILD/BUILD/flower-6.0/docs

* Review and carefully edit the SPEC file, $FLOWER\_RPMBUILD/SPECS/flower.spec, for any necessary changes. Typically only the version numbers need to be changed.
  + **NOTE**: Any changes should be committed back to the git repository

Run the RPM build for FLOWER.

# cd $FLOWER\_RPMBUILD/SPECS

# ./build\_RPM.sh flower

You should now have a completed FLOWER RPM ready to be deployed:

$FLOWER\_RPMBUILD/RPMS/x86\_64/flower-6.0-0.x86-64.rpm

# Installing RPMs

This section describes how to install the required RPMs that have been built in the previous sections.

Install FLOWER

# export FLOWER\_RPMBUILD=$HOME/git/flower/support/build/rpmbuild

# rpm –i $FLOWER\_RPMBUILD/RPMS/x86\_64/flower-6.0-0.x86-64.rpm