

pdinstall -- Installing PigeonDeliver

Carlo Contavalli

ccontavalli at masobit.net

Revision History

Revision 1.0.0 2005/06/11
First document revision

Purpose of this document is to explain how to install PigeonDeliver on your systems, starting from the source files provided on the PigeonDeliver site.

1. Before starting

This document was written as part of the documentation of the PigeonAir Project to provide help and support to users, system administrators or developers.

While every effort has been made to ensure that the information is accurate at the time of publication, this document may contain errors, omissions, incongruences or wrong technical details. No liability for damages is accepted by the Author/Authors, the publishers or any other organization or person providing the information, arising from any errors or omissions that may appear, however caused.

In case you find an error, you would like to propose better solutions than those discussed in this document or you would like to discuss an idea regarding this document or its content, we would be glad to hear from you and please feel free to contact us by writing directly to the author of this document or to the <pigeon-dev at ml.pigeonair.net> mailing list.

1.1. Intended Audience

This document was meant for administrators interested in compiling and installing PigeonDeliver source code on their systems.

The only purpose of this document is thus to introduce the reader to the PigeonDeliver build and install system, and to detail the steps needed to install PigeonDeliver on most systems.

This document will not discuss details regarding its configuration, usage or integration with other systems.

1.2. Copyright Notice

This document was written by Carlo Contavalli <ccontavalli at masobit.net> and is thus Copyright (C) 2003,2004,2005,2006 Carlo Contavalli.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Back-Cover Texts, and no Front-Cover Text.

Any example of program code available in this document should be considered Copyright (C) 2003,2004,2005,2006 Carlo Contavalli, protected by the terms of the GNU General Public License, version 2.00.

You should have received a copy of the GNU General Public License along with this document; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Trademarks are owned by their respective owners.

2. Introduction

Like with most other unix softwares, in order to install pigeondeliver you need to: configure it (./configure), compile it (make) and install it (make install).

However, as you may already know, pigeondeliver is made of modules. Configuring pigeondeliver is not as easy as with most other unix softwares: configuring pigeondeliver means choosing the modules you want to compile, how you want to compile them and how you want them to be used. In other words, you can combine most of the pieces of PigoenDeliver at your wish, and the configuration and compilation process play a very important role on the final result.

So, before going on installing PigeonDeliver, *we strongly suggest you to carefully read the following sections.*

You should also note that while every effort has been made to make the installation process as easy and as smooth as possible, it is not possible for us to enumerate and explain all the possible combinations that could be used to configure and compile PigeonDeliver.

We will thus try to explain the main concepts, in order to allow you to choose which parameters better suit your needs.

Your experience with PigeonDeliver is also very important to us. Please report any success or failure with PigeonDeliver to the mailing list. We will be glad to hear from you...

2.1. What is PigeonDeliver?

PigeonDeliver was born as a very enhanced and very powerful Delivery Agent. As time past, it became more and more a modular framework to easily build and add new components to mail servers.

The basic idea was to create an infrastructure that would allow to abstract out and centralize handling of users' configuration, while inserting a layer to easily allow distributing the load in a cluster of machines.

So, right now, PigeonDeliver is a set of modules, services and SAPI, all kept together by a set of libraries and infrastructures. Each SAPI can provide support for a different mail server, without having to introduce changes to any other component of PigeonDeliver. Modules provide extensions and enhanced features to other components, while services allow you to scan your mails for viruses, add vacation messages, cleanup the messages, remove spam, and so on...

The extra value that you get by using PigeonDeliver instead of pushing all the needed components in your mail server configuration (like spamassassin, clamav, ...) is that added by the middle layers:

- all configurations can be kept in the same place, usually a database, but you are not bound to it, organically organized and ready to host all of those modules configurations.
- the administrator of the machine is the one 'who decides'. As design and implementation decisions, many choices were not made: file system quotas or maildir quotas? mailbox or maildir? use a proxy or not? the modular design allows the administrator to choose whichever architecture suits better his own needs. And, at the same time, moving from one architecture to another is much easier...
- every module inherits some sort of Access Control Mechanisms, so users can change every detail of their own delivery without influencing the configuration of other users or domains. Administrators can decide which services each domain can offer, and which configurations each and every user should be able to change by himself...
- the database is authoritative in terms of user configurations: there are no commands to run on the server to add/remove users, or to perform any other kind of operation... you want to enable spamassassin for a given user, but not for all the other users? you just have to enable the corresponding field in the db...
- atomicity, and ability to share the load among a cluster of machines. The whole PigeonDeliver design was thought as that of clustered configurations, where a lot of *independent* machines are racing to provide services and to share load among them...
- developing new modules and new components is easy and quick. Every module, without additional efforts, inherits all the features of the whole infrastructure..

- the cleaner design, interfacing and structure makes it easier to upgrade the system, keep the user database consistent with the new features provided by various modules, and so on...
- the well defined user database allows to develop new web interfaces to the PigeonDeliver db very easily and very quickly, whenever the provided interface (PigeonAdmin) is not enough to satisfy your needs. You don't have to figure out the structure of the tables, how to call and use each and every table from your MTA, no cludged commands to run from your php/asp/pl cgi/scripts to make your changes effective, clean handling of encoding problems and so on...

2.2. System requirements

At time of writing (Mon Jan 9 13:55:41 CET 2006), PigeonDeliver should be installable on any POSIX *nix system. However, it was written, debugged and tested mostly on GNU Linux systems and FreeBSD 5.0.

While every effort has been taken to keep it architecture and system independent, it may not properly work on your system. At time of writing (Mon Jan 23 12:45:10 CET 2006), it still requires a recent version of GCC to be compiled. No other compiler is supported yet, even if we are actively working on it. Please help us updating this page, reporting your success or failure in installing it. We will be glad to hear from you...

To install PigeonDeliver, you need at least to have the following tools installed on your system:

- gcc - any version greater than 2.95 up to 4.x should be fine. Other compilers may work as well, but none of them has been tested.
- gmake - GNU make. On most linux systems, it is just 'make'. On other systems, make sure you have and use GNU make instead of the system one.
- gtar - GNU tar. On most linux systems, it is just 'tar'. On other systems, make sure you have and use GNU tar instead of the system one.
- awk - AWK. Any version should be good.

Make also sure you have all the standard header files installed (libc6-dev, on many linux systems), and make sure to read the manual page regarding the PigeonDeliver modules you are wishing to compile.

2.3. Quick Start

So you know everything about configure scripts and makefiles? You have compiled and installed hundreds of *nix sources? And PigeonDeliver cannot be that different from other softwares...

Ok, so, let's install PigeonDeliver:

```
# Extract pigeondeliver tarball
$ gzip -cd pigeondeliver*.tar.gz | tar xv
$ cd pigeondeliver*

# Create an empty directory build
$ mkdir build
$ cd build

# READ THE FOLLOWING SECTIONS and, only AFTER READING THEM, run
# the configure script with something like -- Watch out for the spaces!!
$ ../configure --prefix=/usr/local --enable-libdscm0 --enable-libdscm-utils0 \
  --enable-libdscm-sapi0 --enable-libdscm-postfix-sapi0 \
  --enable-libdscm-io0 --enable-libdscm-ipc0 \
  --enable-libdscm-postfix-ipc0 \
  --enable-module="loader/dlfcn:postfix config/postfix:postfix"\
" error/postfix:postfix all:loader/dlfcn" --enable-service="all:loader/dlfcn"
  --enable-sapi=postfix

# Compile the sources
$ make

# Install the binaries
$ su
# make install
```

As you probably may note, the configure parameters are not quite standard. As you probably may also note, the `--enable-module` parameter is quite long. It has been split into multiple lines only for readability purposes. Either write the whole `--enable-module` on one single line, without using the `\`, or use the `\` and do not add any more or any less spaces than displayed above (well, "than displayed above" should be more like "than I wrote in the original docbook xml file", where there are no spaces before " error ..., regardless of the media being used to read the document).

Hopefully, after typing the above commands, you will end up with PigeonDeliver and all its modules compiled to be used by Postfix on your system.

In the `/examples/configure` directory, you can find a couple more examples of how to run the configure script in various environment and with different options. To use those examples, do something like:

```
$ mkdir build
$ cd build
$ /bin/sh ../examples/configure/configure.whatever
$ make
$ su
# make install
```

Our suggestion is to start configuring pigeondeliver by modifying the example file that best suits your needs.

If the configure script does not work for you or you want a better knowledge of what all those parameters mean, please *make sure to read the following sections!*.

2.4. Downloading and extracting PigeonDeliver

Before starting, download pigeonDeliver tarballs and additional packages, as indicated on the web pages from <http://deliver.pigeonair.net> (<http://deliver.pigeonair.net/>).

To extract the tarball, use the tar command. It is strongly suggested to use GNU tar, since other versions of tar may not work. Please run the command:

```
tar xvzf pigeondeliver-x.x.x.tar.gz
```

or

```
gzip -cd pigeondeliver-x.x.x.tar.gz |tar xv
```

3. Configuring PigeonDeliver and the configure script

3.1. Introducing modules, SAPI, services, utilities and customs

PigeonDeliver tarball contains several libraries/software/modules... the configure script is the glue that allows you to choose which softwares or modules you want to be compiled, and which modules or libraries should be made available to each and every software.

If you look to the 'configure --help' output, you can find a section titled "DSCM Specific parameters". In this section, which may look something like:

```
service -- Available Services:
```

```
  mailForward -- Forward received emails to other users/domains
  mailVacation -- Allows users to setup their own vacation/autoresponder
  [...]
```

```
module -- Available generic Modules:
```

```
  mailMaster/interfaces/mailIPC -- Allows mailMaster to use other services by using a SAPI
  mailMaster/interfaces/mailUsers -- Handles mailUsers objects, allowing each user to have
```

```
evi/clamav -- Allows mailAntivirus to scan emails using the clamav antivirus
[...]
```

utility -- Available Utilities:

```
+*evi-scanner -- Scan files using the indicated evi
+*mda-manager -- Allows to allocate/create/remove mailboxes for users
[...]
```

sapi -- Available SAPI Modules:

```
*forward -- Allows running PigeonDeliver from a .forward file
*postfix -- Allows running PigeonDeliver directly from Postfix facilities
[...]
```

custom -- Other components available:

```
*dscm-dict -- dict scheme for dscm datatree to be used with postfix
*dscm-sasl -- sasl support for dscm authentication scheme
[...]
```

You can find many other "subsections", where each subsection lists some components which you may or may not compile. As you may also note, some "components" are marked with a [+] and an [*].

Starting with the sapi subsection, each component (forward, postfix, ...) is a module able to support a given mail server. For example, the postfix sapi is a module that allows you to use PigeonDeliver with Postfix.

Usually, compiling a SAPI leads to a particular executable to be created and installed. As an example, if you compile the postfix sapi, you end up with a program called "pd-postfix", which is the delivery agent to be used internally by postfix.

Additionally, there are other modules that lead to an executable to be created: all those modules marked with a '+'. As an example, the evi-scanner module creates an utility called 'pd-evi-scanner'. So: '+' means an executable will be created, even if the module is not a SAPI, and '*' means the module will need other modules to provide the service it was written for.

Finally, in the --help listing there are also 'services', 'modules' and 'customs':

modules

are basically add-on softwares that can be loaded or replaced at wish in order to change the behavior of a given utility or SAPI. For example, the mailAntivirus service requires an evi module. An evi module is just a module providing support for a particular antivirus.

services

are particular kind of modules meant to be used by SAPI in order to offer enhanced mail services. In future releases of PigeonDeliver, services will become simple modules.

customs

are small components meant to be used by external softwares that need a particular compilation process to be used. Example of customs include the support for SASL authentication or dict_dscm, to allow postfix to perform lookups using dscm libraries.

3.2. Modules: names and interfaces

Modules are described by some sort of path: as an example, error/postfix and error/stderr are both modules. The path is always made of at least two fragments separated by a '/'.

The part after the last '/' is the *unqualified name* of the module, while all there is before the last '/' is the interface being used by the module, or, in simpler words, the *purpose* of the module. As an example, the modules error/postfix and error/stderr (fully qualified names) use the 'error/' interface, their unqualified name is postfix and stderr, and their purpose is to manage errors.

Most components of PigeonDeliver need other modules/components in order to run. Instead of declaring the fully qualified name of the needed modules, each component declares just the "interface" or the "purpose" of the needed module. The administrator is then free to choose, either at compile time or at run time, which module to use to satisfy the dependency.

For example, the Postfix SAPI, which allows postfix to use PigeonDeliver directly, needs to record events in the log files and to read configuration options. Internally, the Postfix SAPI thus declares that it needs an error/ module (to log events) and a config/ module. You, as an administrator of your system, are then free to use, for example error/mysql, error/syslog, or error/whatever to send logs wherever you like. At the same time, you can make the postfix SAPI read configurations from the postfix main.cf file (config/postfix), an ldap directory (config/ldap) or from whichever source may become available (config/whatever).

As another example, the utility pd-evi-scanner, which allows you to scan files for viruses, needs to read some configuration options, so, it needs a config/ module. If you compile pd-evi-scanner with the config/postfix module, pd-evi-scanner will read its own configuration options from the postfix configuration file, allowing you to have one single place for all your mail server configurations.

3.3. Basic, required and recommended modules

All PigeonDeliver components always require at least 3 modules in order to be usable at all:

an error/ module

to allow the component itself to send error messages to the administrator of the system. The simplest is error/stderr, which simply outputs error messages to stdout.

a config/ module

which allows each and every component to simply read configuration variables from a configuration file/whatever. The simplest is config/cmdline (%%TODO%%), which simply parses the command line into a hashing table.

a loader/ module

which allows each and every component to load and use other modules. The simplest is loader/static (%%TODO%%), which simply assumes all needed modules have been compiled into the current application.

The administrator is free, for every PigeonDeliver component, to choose which error/, config/ or loader/ module to use at run time, by just specifying some configuration parameter.

But, hey, wait a minute... now we have a big problem to solve... how is the software supposed to know which module to use to output errors if no config/ module is available yet? how is it supposed to know which config/ module to use, if there is no way to read a configuration file?

To solve this simple chicken/egg problem, at least one for each kind of the modules above must be compiled directly into each and every runnable PigeonDeliver component. Once the component "bootstraps", using the components directly "compiled in", you can use configuration parameters to switch to another loader, error or config handler.

For example, you may compile the pd-evi-scanner utility with the config/postfix module compiled in. As soon as pd-evi-scanner starts, it will read the needed configuration parameters from the postfix main.cf configuration file. However, in this file you can use a "config" parameter to switch to some other config/ module, reading, for example, all subsequent parameters from a mysql server.

The error/, loader/ and config/ modules are also called "basic" modules. Without them, a PigeonDeliver component cannot be run at all. Those error/, loader/ and config/ modules that can be compiled directly into an executable without further help by other modules of the same kind are called "bootstrap" modules.

For example, config/postfix is a bootstrap module. It reads configurations from the /etc/postfix/main.cf file. config/mysql, however, cannot be use as a bootstrap module (it may not be true), since, in order to be used, it needs to know the server, user, and table to use to fetch configuration parameters.

Once the process is bootstrapped, it is possible to specify in the configuration files other modules we want to use to handle error messages or config directives. The necessary parameters will be discussed later on in this document (config, error, alive, loader).

Finally, a given utility/SAPI may need some other modules in order to work at all (required modules) or suggest the user that it will work better or provide more useful features if other modules are made

available (recommended modules). Those are called required and recommended modules.

3.4. Configuring and compiling

Before anything else, whenever you download PigeonDeliver, you need to know what you want to be installed on your system and how you want it to be compiled.

For example, you may want to install PigeonDeliver just because you like the `pd-sda-manager` utility.

Once you know what exactly you want to be installed, you need to decide which modules you want to be compiled into the utility, which modules you want to be compiled to be loaded at run time, all the compilation options, and whenever to use internal libraries or libraries which may have already been installed on your system.

As an example, I may want to compile `pd-sda-manager` in order to handle `maildir` and `mailbox` files. I know that, on my system, the `sda-manager` will be called internally by some scripts used by my postfix infrastructure, so I want configuration to be read from the postfix configuration file (`main.cf`) and error logs written in the postfix formats using postfix infrastructures, while I know that all my modules should be written in the standard format for linux: `.so` files.

With the above sentence, I just decided which modules I want to be compiled:

`error/postfix`

to send error messages to postfix logs

`config/postfix`

to read configurations from postfix `main.cf`

`loader/dlfcn`

to load modules written as `.so` files

`sda/maildir` and `sda/mailbox`

to allow the `sda-manager` to write both to `maildir` and `mailbox` files

In order to compile the `sda-manager` with the above options, I may end up with a configure line like the following:

```
./configure --enable-utility='sda-manager' \  
--enable-module='loader/dlfcn:sda-manager' \  
'error/postfix:sda-manager config/postfix:sda-manager' \  
'sda/maildir:loader/dlfcn sda/mailbox:loader/dlfcn'
```

Which simply means: "I want the utility sda-manager to be compiled (--enable-utility...), and I want the loader/dlfcn module to be compiled (listed in --enable-module) and 'put directly' into the utility sda-manager (loader/dlfcn:sda-manager), since it is necessary to "bootstrap" (see previous section) as much as the config/postfix and error/postfix modules, while I want the sda/maildir and sda/mailbox modules to be compiled in order to be loaded at run time by the loader/dlfcn module (sda/mailbox:loader/dlfcn sda/maildir:loader/dlfcn)".

Obviously, there are shortcuts that will be discussed in the following sections. What is important to note here is that, if you forget some important components, the configure will complain. For example, if you do not specify a config/ module, the configure may complain with something like:

```
configure: error: fatal - module: sda-manager needs a loader/, but none has been enabled
```

At time of writing (Mon Jan 9 19:26:14 CET 2006), the configure will also complain if you do not manually enable all needed libraries. In this case, just follow the instructions printed by the configure, which will look something like:

```
configure: error: fatal - library: libdscm0 needed by sda-manager must be enabled.  
Please specify --enable-libdscm0 or --with-libdscm0
```

where --enable-libdscm0 would be the right choice :)

3.5. Libraries

The PigeonDeliver tarball also contains many libraries. Those libraries are used internally by all pigeondeliver modules.

At time of writing, a library can be compiled by specifying --enable-name-of-library to the configure script as a parameter.

Whenever the configure scripts detects that some important libraries are missing from your system it will print a fatal error asking you to manually specify --enable-name-of-library or --with-name-of-library.

At time of writing (Tue Jan 10 00:08:14 CET 2006), the configure script is not able to add those parameters by itself, so please add the parameter and run the configure script one more time.

--with-name-of-library is reserved for future usage, and it will allow you to tell the configure script that instead of one of the provided libraries, the already installed library specified as a parameter to the --with option should be used instead.

Obviously, if you like PigeonDeliver libraries, you may also install them without other components by just running configure with all the `--enable-lib*` parameters but with no modules, services, or sapi specified.

3.6. Installation directories and directory tree

Well, every PigeonDeliver component must be installed in one directory. You can choose which directory to use for the various components, by specifying the correct parameters to the configure script.

`--libdir`

All the DSCM and PigeonDeliver libraries are installed in this directory. This means things like `libdscm0.la`, `libdscm0.so`, `libdscm-utils0.la` and so on will be installed in this directory. By default, `libdir` is set to `EPREFIX/lib`.

`--libexecdir`

All the DSCM and PigeonDeliver modules (both services and so called modules), are installed in this directory. PigeonDeliver modules are used by various PigeonDeliver components to provide additional functionalities or new services. This means things like `config-postfix.la`, `config-postfix.so`, `error-postfix.la` or `datatree-ldap.la` will be installed in this directory. By default, `libexecdir` is set to `EPREFIX/libexec`.

`--bindir`, `--sbindir`

PigeonDeliver binaries and tools reserved to the super user will be installed in the directories specified with the indicated parameters. By default, they are set to `EPREFIX/bin` and `EPREFIX/sbin`.

All the other parameters have the same meaning as usual: `--mandir` will be used for manual pages, `--sysconfdir` for the configuration files, `--localstatedir` for queue files or pid/args information, depending on the SAPI being used and so on..

As usual, an easy way to move *all directories* away is to specify the `--prefix` or `--exec-prefix` parameter. As you may guess, `--prefix` allows you to change the value of the `PREFIX` variable we've seen in the previous bullet list, while `--exec-prefix` allows you to change the value of `EPREFIX`.

All the Makefiles have also been written to accept a `DESTDIR` environment variable, to force the installation of all files under a specified directory, even if that is not the final directory where the files will be really put (needed for package maintainers).

4. PigeonDeliver and Postfix

PigeonDeliver and any PigeonDeliver component can easily be used with postfix (<http://www.postfix.org>). The default PigeonDeliver tarball contains:

error/postfix

A module able to send every and all error messages to the mail.log file, using Postfix libraries and configuration files. Use this module with any PigeonDeliver component to log error messages as indicated by Postfix configuration files and using postfix libraries.

To use the error/postfix module, you just need to enable it: `--enable-module="error/postfix:all"`. error/postfix is also a "bootstrap" module, and can be compiled directly in any utility/sapi with something like: `--enable-module="error/postfix:postfix"`.

config/postfix

A module able to provide configuration parameters by reading the main.cf configuration file. Use this module with any PigeonDeliver component to make it read its own configuration from postfix configuration files.

To use the config/postfix module, you just need to enable it: `--enable-module="config/postfix:all"`. config/postfix is also a "bootstrap" module, and can be compiled directly into any utility/sapi with something like: `--enable-module="config/postfix:postfix"`.

sapi/postfix

A module able to run services directly from the postfix master.cf file, using directly the postfix protocol and libraries to write queue files and to deliver emails. Use this module to run any PigeonDeliver service in Postfix.

To use the Postfix SAPI, you must enable it with something like `--enable-sapi="postfix"`. Make also sure to configure the bootstrap modules to be used for the Postfix SAPI (%%TODO%%), with something like: `--enable-module="config/postfix:postfix error/postfix:postfix loader/dlfcn:postfix"`.

Additionally, the PigeonDeliver tarball contains:

dscm-dict

A module that allows postfix to perform user and domain lookups directly using PigeonDeliver infrastructures and libraries, speeding up the whole delivery process and simplifying the ease of configuring Postfix with PigeonDeliver.

To compile dscm-dict, you must enable it with something like `--enable-custom="dscm-dict"`. Make also sure to configure the bootstrap modules to be used with something like:
`--enable-module="config/postfix:dscm-dict error/postfix:dscm-dict loader/dlfcn:dscm-dict"`.

dscm-sasl

A module that allows libsasl to authenticate users directly by using PigeonDeliver infrastructures and libraries, speeding up the process of authenticating users and easily allowing the use of SMTP Authentication in Postfix or the integration of PigeonDeliver in POP3/IMAP servers.

To compile dscm-sasl, you must enable it with something like `--enable-custom="dscm-sasl"`. Make also sure to configure the bootstrap modules to be used with something like:
`--enable-module="config/postfix:dscm-sasl error/postfix:dscm-sasl loader/dlfcn:dscm-sasl"`. To configure SASL support, you may also want to use the parameters `--with-sasl2-include=PATH`, `--with-libsasl2=PATH` and `--with-sasl2-install=PATH`.

Once you know which of the postfix modules you would like to use and compile, please make sure to read the following sections (%%TODO%%), since they contain important instructions on how to compile any postfix component. To know how to configure Postfix to use PigeonDeliver, please make sure to read %%TODO%%.

4.1. Compiling and Installing Postfix components

At time of writing, there are at least two known ways to install postfix on a given system:

- by closely following the procedure provided in the INSTALL files in the postfix top level source directory.
- installing packages provided with many linux distributions, or installing a Postfix version patched to create shared libraries.

In the first case, postfix will probably be installed on your system *without header files* and with libraries compiled *directly into postfix*. So, in order to install PigeonDeliver, you will need to follow the instructions provided in the section %%TODO%% installing pigeondeliver with postfix sources.

In the second case, on most linux postfix installations there will be shared libraries and header files ready to be used by PigeonDeliver. In order to install PigeonDeliver, follow the instructions provided in the section %%TODO%% installing pigeondeliver with postfix shared libraries.

Note also that you will need to follow the procedure indicated in the next few sections *even if you are not compiling PigeonDeliver to be used with postfix (Postfix SAPI)*, but in *every and all* cases where you need to compile and use any of the modules needing postfix libraries (like error/postfix or config/postfix).

4.1.1. Installing with Postfix sources

In order to install PigeonDeliver with postfix sources, you will need to:

1. Download postfix sources from www.postfix.org. Read its own documentation, and prepare to compile it with all the parameters you need.
2. When running "make makefiles", also specify the parameter `CCARGS='-fPIC -DPIC'`, with something like:

```
$ make makefiles CCARGS="-fPIC -DPIC [all other options, if any]" ...
```

3. Compile postfix libraries, by entering `src/util`, `src/global`, `src/master` and `src/postconf`, and by running `make` in the corresponding directories, or with something like:

```
$ make update DIRS='src/util src/global src/master src/postconf'
```

Before running the above command, please note that *every error in the above line* will probably cause your postfix compilation process to loop forever, possibly fork-bombing your system. *So, please make sure you typed it correctly!*

If you don't want to compile `dict-dscm %%TODO%%` into PigeonDeliver, you can compile and install the whole Postfix without worries. Most likeley, you will want to compile this module, so follow the instructions closely.

4. Go on configuring, compiling and installing PigeonDeliver with something like:

```
$ mkdir build
$ cd build
$ ../configure [...]
  --with-postfix-sources=/dir/where/postfix/has/been/compiled
  [...]
```

5. If you compiled just the postfix libraries, as indicated in the previous step, complete the Postfix installation as usual, by going back in the postfix TLD directory and by running `make` and `make install` from there, with something like:

```
$ make
$ make install
```

4.1.1.1. Parameters to be used when installing from sources

`--with-postfix-sources=PATH`

Use this option to indicate the directory where you have decompressed the postfix sources, and to

indicate that you want all the postfix components to be compiled using those sources. For example, you can use something like:

```
$ ../configure --with-postfix-sources=/src/postfix-2.2.8/
```

where postfix-2.2.8 is the directory containing the postfix file INSTALL, README, makedefs, ...

4.1.2. Installing with Postfix shared libraries

In order to install PigeonDeliver with postfix shared libraries, you will need to:

1. Install postfix with the mechanism provided by your linux distribution or operating system. In alternative, download postfix sources, apply the patch in patches/postfix/ called shared-libraries-dynamic-modules, recompile and install postfix as usual.
2. Go on configuring, compiling and installing PigeonDeliver *without specifying* the --with-postfix-sources. With something as simple as:

```
$ mkdir build
$ cd build
$ ../configure [all other parameters]
```

3. In case the configure is unable to find the postfix libraries, consider using the --with-postfix-include, --with-postfix-defs, --with-postfix-makedefs, --with-postfix-postconf, --with-libpostfix-util, --with-libpostfix-global, --with-libpostfix-master.

4.1.2.1. Parameters to be used when installing with shared libraries

--with-postfix-include=PATH

Use this option to indicate the directory containing the files sys_defs.h, mail_addr.h, mail_conf.h, ... Needed only if the configure cannot figure out the directory by itself. Something like --with-postfix-include=/usr/include/postfix/ should do the work, provided /usr/include/postfix contains the above indicated files.

--with-postfix-defs=PATH

--with-postfix-makedefs=PATH

Postfix, internally, uses a 'makedefs' script to guess the parameters used on your system. PigeonDeliver must also guess the same parameters as postfix. So, you should either specify where to find the 'makedefs' script (--with-postfix-makedefs=/usr/src/postfix-2.2.8/makedefs, for example), or, if you saved the output of the command 'makedefs' in a file when you first run it, just use something like --with-postfix-defs=/tmp/file.defs to load those definitions.

In case none of the two parameters is used, PigeonDeliver will call an internal version of 'makedefs', hoping to get an acceptable result.

`--with-postfix-postconf=PATH`

Use this option to specify the command 'postconf' to be used. postconf is used internal by PigeonDeliver configure scripts to guess which parameters have been used to install postfix. You should specify the complete path, like with: `--with-postfix-postconf=/bin/postconf`.

`--with-libpostfix-util=PATH --with-libpostfix-global=PATH --with-libpostfix-master=PATH`

Use the above options if the configure cannot detect by itself the position of the libraries libpostfix-util, libpostfix-global or libpostfix-master. Depending on the system and compilation process used to compile and install postfix, those libraries may even have a different name. The directory indicated with `--with-libpostfix-util`, for example, with something like `--with-libpostfix-util=/usr/lib`, will be looked for the file libpostfix-util.so and/or libutil.a, which are the two most common names used for postfix libraries. The same thing applies for `--with-libpostfix-global` and `--with-libpostfix-master`.

Watch out that it is also pretty common on many systems to have a libutil.so, or libutil.a that has nothing to do with Postfix. Make sure to specify the path of Postfix own libutil! Sometimes, the configure may also erroneously use one of the above libraries. In these cases, you need to manually supply the `--with-libpostfix-util` parameter.

4.2. Configuring Postfix to use PigeonDeliver

There are several different ways to use PigeonDeliver with Postfix. The easiest and more performant way is to use the Postfix SAPI and to enable PigeonDeliver support from the master.cf configuration file. To do so, just:

1. configure, compile and install PigeonDeliver with the postfix SAPI and dscm-dict custom enabled, with something like `--enable-sapi="postfix"`, `--enable-custom="dscm-dict"` and `--enable-module="error/postfix:postfix:dscm-dict config/postfix:postfix:dscm-dict loader/dlfcn:postfix:dscm-dict all:dlfcn"`, following the instructions provided in the section `%%TODO%%`. Look at the examples Appendix `%%TODO%%` for more examples. If you want to use *SMTP Authentication*, please also remember to enable dscm-sasl, with something like `--enable-custom="dscm-dict dscm-sasl"` and `--enable-module="error/postfix:postfix:dscm-dict:dscm-sasl config/postfix:postfix:dscm-dict:dscm-sasl loader/dlfcn:postfix:dscm-dict:dscm-sasl all:dlfcn"`.
2. configure postfix to use PigeonDeliver. Depending on the version of Postfix being used, we strongly suggest to first configure Postfix to work for local users only, with a standard configuration (look at the file mail.local.cf `%%TODO%%`). Once mails are accepted and delivered to local users only, add the following parameters to your configuration file:

```
virtual_mailbox_domains = dscm:domain
```

```
virtual_mailbox_maps = dscm:user  
virtual_transport = mailMaster
```

the above parameters tell postfix to lookup domains it doesn't know anything about using the 'dscm' library (PigeonDeliver). The same thing applies to users. While

5. About PigeonDeliver modules

6. Installing PigeonDeliver utilities

7. Before running the configure script

You need:

- gcc, gmake, awk, sed, ... and most development tools normally available on any unix system. Make sure to have GNU make and gcc. At time of writing, pigeondeliver will not compile with any other compiler. Any version > 2.95, up to 4.x should be good.
- postfix-dev, libc6-dev