A Mailserver on Ubuntu 18.04: Postfix, Dovecot, MySQL

By Reason

February 24th, 2019

Permalink

This long post contains a recipe for building a reasonably secure Ubuntu 18.04 mail server in Amazon Web Services, using Postfix, Dovecot, and MySQL, with anti-spam packages in the form of amavisd-new, Clam AntiVirus, SpamAssassin, and Postgrey. Let's Encrypt is used to provide an automatically renewing SSL certificate. Local users are virtual rather than being system users. Administration of users and domains is achieved through the Postfix Admin web interface. Webmail is provided by Roundcube.

This is an updated version of earlier Ubuntu 12.04, Ubuntu 14.04, and Ubuntu 16.04 mailserver recipes. A number of people graciously helped to fix bugs and make improvements in the original, so should you find a blocking issue here please do let me know.

Introduction

Building a Linux mailserver from scratch to your own liking is a painful process unless you happen to be one of the few folk who do that day in and day out - there is no way around that fact. A mailserver generally consists of a range of different packages that separately handle SMTP, POP/IMAP, local storage of mail, and spam-related tasks: they must all talk to one another correctly, all have small novels in place of configuration documentation, and there is no one obvious best practice for how users are managed, how to store user data, or how to glue the various different components together. There are any number of different viable setups for moving mail between Postfix and Dovecot, for example. Further, the whole assembly tends to be unforgiving on matters such as file ownership and permissions, choice of users for specific processes, and tiny errors in esoteric configuration files. Unless you know what you are doing the end result will likely be either insecure or otherwise subtly non-functional. Merely not working is perhaps the best of bad outcomes.

You must also pay attention to adding support for the dominant tools of the deliverability and spam suppression industry, such as Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM), or else risk mail from your server being flagged for the spam folder or invisibly discarded by many email providers. This is a broad subject and a changing ecosystem worthy of a long how-to essay in and of itself.

There are a number of fairly up to date recipes for creating mailservers out there; one of the better ones is an Ubuntu recipe by Ivar Abrahamsen, which gives you Postfix for SMTP, Courier for IMAP/POP, MySQL to store account information, virtual user mail directories, and an array of anti-spam tools that are highly effective when working in concert. It's a good set of documents, as the author places an emphasis on producing a secure mailserver as the end result. In the past I have made good use of Abrahamsen's guide as a basis for my mail servers, and recommend it.

There are also a great many partial recipes and out of date guides that are frankly more of a hindrance than a help - especially when it comes to Dovecot, which has changed greatly between its 1.* and 2.* versions. The configuration is completely different, and so are many of the administrative and tool binaries. When I chose to migrate my servers from Courier to Dovecot back in 2012 it was a challenge to find a good all-in-one-place guide, and hence the existence of the first in this series of documents. Since then I've added new versions as Ubuntu updates its long term support releases.

You should at least skim the whole of this post before setting forth to follow its recipe. That should help to avoid unpleasant surprises, and there are some notes at the end on alternative options and additions that are worth reading before you get started.

Outlining the Goal

The end result of following this guide will be a secure mail server for your domain equipped with the following software packages:

- Postfix: sends and receives mail via the SMTP protocol. It will only relay mail on to other mailservers if the mail is sent by an authenticated user, but anyone can send mail to this server for local delivery.
- Dovecot: a POP and IMAP server that manages local mail directories and allows users to log in and download their mail. It also handles user authentication.
- Postgrey: greylists incoming mail, requiring unfamiliar deliverers to wait for a while and then resend. This is one of the better tools for cutting down on spam.
- amavisd-new: a manager for organizing various antivirus and spam checking content filters.
- Clam AntiVirus: a virus detection suite.
- SpamAssassin: for sniffing out spam in emails.
- Postfix Admin: a web front end for administering mail users and domains.
- Roundcube: a webmail interface for users.

The server will accept plain text or encrypted SMTP and POP/IMAP connections at the standard ports, but will not allow user authentication without encryption. It will pass through a minimal set of mail headers for mail sent by local users,

removing identifying information from the original sender's mail software.

Using Amazon Web Services

This post is written assuming the use of Amazon Web Services to host a virtual mail server, but in theory any hosting service can be used. Very little of the material here is concerned with Amazon-specific issues. So if you are working with another service, just skip over the AWS-specific instructions and perform the equivalent operations in the service that you have chosen to use. Note, however, that not all hosting services are equal when it comes to how email providers view mail coming from your server. For example, it is pointless to even try hosting any significant mail service at Digital Ocean: their IP ranges are blacklisted by many services, and you will have no leverage with any of the parties involved when it comes to trying to ensure delivery.

If not setting up in AWS, at the very least ensure that you firewall your server adequately right at the outset. In many services a new server or virtual instance is completely exposed to the internet, so you will want to lock it down immediately with something like Uncomplicated Firewall (UFW). For example, as below, replacing `MY_IP_ADDRESS` with your IP adddress:

```
1 sudo su
2 apt-get install ufw
3 ufw enable
4 ufw allow from MY_IP_ADDRESS
```

After you have completed setup then you can open up your server as appropriate to allow it to communicate with the rest of the world. Note that you have to open up the webserver at least briefly to allow Let's Encrypt SSL certificate automation to work, however. That is noted in the relevant section.

Use of example.com and mail.example.com

Throughout the instructions in this post `example.com` is used as the domain, and the mail server for this domain has the hostname `mail.example.com`. So wherever you see these items be sure to replace them with your chosen domain and mail server hostname.

Fire up an Ubuntu 18.04 AWS Instance with a Suitable Security Group

Start up a new server instance. At the time of writing, Ubuntu 18.04 is one of the options right there in the quick start menu for launching a new instance. Mail servers don't generally have to be all that big if you aren't in the business of email: 2G of memory is enough for the recipe here, and that much is needed only because ClamAV and Amavis are

memory hogs. Thus while micro instances are too small any of the larger instance types should be more than enough to support a personal mail server, a small business mail server, or the throughput of a mailing list of a few thousand members.

Firewall settings in AWS are managed through assignment of Security Groups. You'll probably want to create one before starting the server. The Security Group should allow inbound TCP traffic from any IP address to these ports:

- 25 (SMTP)
- 80 (HTTP)
- 110 (POP3)
- 143 (IMAP)
- 443 (HTTPS)
- 465 (SMTPS)
- 993 (IMAPS)
- 995 (POP3S)

The above is in addition to whatever rules you might have for SSH access over port 22 - it is not a good idea to leave that open to the world, so lock it down to the IP address ranges you use. In fact it is a good idea to restrict all inbound traffic to the server to your own IP addresses while you are building it. You can adjust the rules to allow traffic from the rest of the world after you're certain that everything is secure and shipshape. Again, note that Let's Encrypt will need the webserver accessible when the certificate is initially set up.

Some Basic Configuration

The baseline Ubuntu instance is lacking in near every package you might need, so you are building from fairly close to scratch. You'll log in as the "ubuntu" user and then switch to the root user via the traditional "sudo su" command; most of what you need to do requires root access:

```
1 sudo su
```

You must set up an Elastic IP to give your server a permanent IP address. By default, an AWS instance will have its own strange-looking hostname, so changing to the domain the server will have is the first item on the list.

```
1 hostname mail.example.com
```

Now set the contents of /etc/hostname to be the hostname:

```
echo "mail.example.com" > /etc/hostname
```

And add your hostname to the first line of /etc/hosts:

```
127.0.0.1 mail.example.com localhost

# There will be IPv6 configuration below the first line, but leave that alone.
...
```

Now you will want to regenerate the server's default self-signed SSL certificate so that it matches the domain name. You will be setting up a real SSL certificate for your mail server via Let's Encrypt a little further along, but it is perfectly possible and completely secure to run a mail server using a self-signed certificate. The only consequences will be warning screens when using webmail hosted on the server and warnings from mail clients when connecting via POP, IMAP, or SMTP.

```
apt-get install --assume-yes ssl-cert
make-ssl-cert generate-default-snakeoil --force-overwrite
```

Now Build a LAMP Web Server

The list of goals here includes webmail and a web-based administrative interface for managing users, so as a starting point you will need to set up a LAMP web server. The acronym LAMP stands for Linux as the operating system, Apache as the webserver, MySQL as the database, and PHP as the scripting language. Fortunately there is a shortcut to install all of the basic LAMP packages, so start by updating the repository data and installing those packages. Notice the "^" character at the end of the command below, as it is necessary:

```
apt-get update
apt-get upgrade --assume-yes
apt-get install --assume-yes lamp-server^
```

Then you can move on to adding an array of necessary or useful additional packages for PHP, such as Mcrypt support, cURL, an XML parser, and GD image processing support. At this point add more packages to suite your own taste and to support any other applications you might want to run on this server.

```
apt-get install --assume-yes \
```

```
  php7.2-curl \
  php7.2-gd \
  php7.2-mbstring \
  php7.2-xml
```

Installing Mcrypt support is a little more involved in Ubuntu 18.04 than in 16.04:

```
apt install --assume-yes \
  libmcrypt-dev \
  php7.2-dev \
  php-pear

pecl channel-update pecl.php.net
yes '' | pecl install mcrypt-1.0.1

echo 'extension=mcrypt.so' > /etc/php/7.2/mods-available/mcrypt.ini
ln -s /etc/php/7.2/mods-available/mcrypt.ini /etc/php/7.2/apache2/conf.d/mcrypt.ini
ln -s /etc/php/7.2/mods-available/mcrypt.ini /etc/php/7.2/cli/conf.d/mcrypt.ini
```

Configure PHP

The default configuration settings for PHP and the additional packages mentioned above, found in /etc/php/7.2/apache2/php.ini , /etc/php/7.2/cli/php.ini , and /etc/php/7.2/mods-available respectively, are sufficient for most casual usage. So unless you have something complicated or high-powered in mind, there is no need to change anything.

Use OpenSSL to Create a Unique Diffie-Helman Group

Good security is requiring ever more work on the part of system administrators these days. One of the more noteworthy attacks on SSL of recent years is known as Logjam, and defending against it requires what is presently a non-standard addition to application SSL settings. Creating your own strong Diffie-Helman group and saving it to a configuration file is the first step:

```
openssl dhparam -out /etc/ssl/private/dhparams.pem 2048
chmod 600 /etc/ssl/private/dhparams.pem
```

Configure Apache

The expected end result for the Apache webserver is that it will serve a single site with a couple of running web applications: (a) Roundcube for webmail, and (b) Postfix Admin hidden away in a subdirectory. All HTTP requests will be redirected to use HTTPS, as there is no good reason to allow non-secure access to any of applications that will reside on the server.

Firstly configure the following lines in `/etc/apache2/conf-enabled/security.conf` to minimize the information that Apache gives out in its response headers:

```
#
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of:  Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
#
ServerTokens Prod

#
# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory
# listings, mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
# Set to one of:  On | Off | EMail
#
ServerSignature Off
```

Make sure that mod_rewrite, mod_ssl, a few other useful modules, and the default SSL virtual host is enabled - you'll need these line items to be able to force visitors to use HTTPS.

```
a2enmod deflate expires headers rewrite ssl
a2ensite default-ssl
```

Edit these lines in `/etc/apache2/mods-available/ssl.conf` to ensure that protocols that are no longer secure are not used:

```
# Aiming for perfect forward secrecy where possible, and protecting against
# attacks such as Logjam. See:
# https://weakdh.org/sysadmin.html
# https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
SSLCipherSuite ECDH+AESGCM:ECDH+CHACHA20:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128
```

```
SSLHonorCipherOrder on

#   The protocols to enable.
#   Available values: all, SSLv3, TLSv1, TLSv1.1, TLSv1.2
#   SSL v2  is no longer supported
SSLProtocol all -SSLv2 -SSLv3
```

The default site configuration in `/etc/apache2/sites-available/000-default.conf` can be edited to look something like this for the sake of simplicity:

```
<VirtualHost *:80>
  ServerName mail.example.com
  ServerAdmin webmaster@localhost

  DocumentRoot /var/www/html
  <Directory "/var/www/html">
    Options FollowSymLinks
    AllowOverride All
  </Directory>

  ErrorLog ${APACHE_LOG_DIR}/error.log

  # Possible values include: debug, info, notice, warn, error, crit,
  # alert, emerg.
  LogLevel warn

  CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

But of course your taste and needs may vary. Keeping the same simple approach, the upper portion of the SSL configuration in `/etc/apache2/sites-available/default-ssl.conf` can be set up as follows. Note the references to the snake-oil certificate files:

```
<IfModule mod_ssl.c>
  <VirtualHost _default_:443>
    ServerName mail.example.com
    ServerAdmin webmaster@localhost

    # Set the HTTP Strict Transport Security (HSTS) header to guarantee
    # HTTPS for 1 Year, including subdomains, and allow this site to be
    # added to the preload list.
    #
```

```
    # Do NOT enable this until you have the final SSL certificate in
    # in place. You can get stuck.
    #Header always set Strict-Transport-Security "max-age=31536000; includeSubDomain

    # Prevent clickjacking by controlling who can put the site into a
    # frame. Only needed for text/html, but doesn't hurt to be applied
    # generally.
    Header set X-Frame-Options "SAMEORIGIN"

    # Prevent mime based attacks by telling browsers that support it
    # to use the declared mime type regardless of what the content looks
    # like.
    Header set X-Content-Type-Options "nosniff"

    DocumentRoot /var/www/html
    <Directory "/var/www/html">
      Options FollowSymLinks
      AllowOverride All
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/ssl_access.log combined

    #   SSL Engine Switch:
    #   Enable/Disable SSL for this virtual host.
    SSLEngine on
    #

    # A self-signed (snakeoil) certificate can be created by installing
    # the ssl-cert package. See
    # /usr/share/doc/apache2/README.Debian.gz for more info.
    # If both key and certificate are stored in the same file, only the
    # SSLCertificateFile directive is needed.
    #
    # Use the Let's Encrypt certificate.
    SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
    SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

    # ... more default SSL configuration ...

    # You will probably need to change this next Directory directive as well
    # in order to match the earlier one.
```

```
57      <Directory "/var/www/html">
58        SSLOptions +StdEnvVars
59      </Directory>
60
61      # ... yet more default SSL configuration ...
```

To push visitors to HTTPS, put something similar to the following snippet into /var/www/html/.htaccess:

```
1  RewriteEngine On
2
3  # Redirect all HTTP traffic to HTTPS.
4  RewriteCond %{HTTPS} !=on
5  RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
```

Make Use of that Diffie-Helman Group

The configuration needed to use the Diffie-Helman group in /etc/ssl/private/dhparams.pem depends on the version of Apache. For the version used in Ubuntu 18.04, add or edit this line in /etc/apache2/mods-available/ssl.conf:

```
1  # Protect against Logjam attacks. See: https://weakdh.org
2  SSLOpenSSLConfCmd DHParameters "/etc/ssl/private/dhparams.pem"
```

Now restart Apache to pick up the changes, after which you should be able to load the default Apache homepage and see that you are automatically redirected to HTTPS.

```
1  service apache2 restart
```

Set Up an Automatically Renewed SSL Certificate with Let's Encrypt

The Let's Encypt initiative allows any server to be set up with a free SSL certificate that automatically renews itself. Note that this requires a running webserver in order to carry out a server validation check, which is why we configured Apache first. First install the certbot tool:

```
1  apt-get install --assume-yes certbot
```

Next, set the contents of the configuration file /etc/letsencrypt/cli.ini as follows:

```
# Because we are using logrotate for greater flexibility, disable the
# internal certbot logrotation.
max-log-backups = 0

# Use a 4096 bit RSA key instead of 2048.
rsa-key-size = 4096

# Set email and domains.
email = admin@example.com
domains = mail.example.com

# Text interface.
text = True
# No prompts.
non-interactive = True
# Suppress the Terms of Service agreement interaction.
agree-tos = True

# Use the webroot authenticator.
authenticator = webroot
webroot-path = /var/www/html
```

Once a certificate has been granted, it is valid for 90 days. Running exactly the same command used to generate the certificate will renew the existing certificate if it is within 30 days of expiration. Running the command in a cron task on a daily basis is a good approach. So, for example, add the following to /etc/cron.daily/certbot-renewal:

```
#!/bin/bash
#
# Renew the Let's Encrypt certificate if it is time.
#
# This reads the standard /etc/letsencrypt/cli.ini.
#

# When running as a cron task, HOME may or may not be set, and
# Certbot drops stuff into ~/.local.
export HOME="/root"
# PATH is never what you want it it to be in a cron context, so
# make absolutely sure of it.
export PATH="\${PATH}:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Using --no-self-upgrade suppresses the automatic update check that
# might not work in a cron context.
certbot --no-self-upgrade certonly

```

```
# Update all of the services that might now need to be using the renewed
# certificate.
service apache2 reload
service postfix reload
service dovecot reload
```

Make it executable:

```
chmod a+x /etc/cron.daily/certbot-renewal
```

Now generate the certificate by running the cron script. First, however, you must make sure that (a) the firewall and security group are set to allow access to port 80 and 443 from any origin, and (b) the DNS entry for mail.example.com points to this server. Let's Encrypt validates issuance of a certificate by placing a file in the webroot, and that must be accessible to the Let's Encrypt servers. Once that is done, run the script:

```
/etc/cron.daily/certbot-renewal
```

Assuming success, the certificates can be found under /etc/letsencrypt/live/mail.example.com. Reference that location in all the later configuration that will need certificates. If you want to lock down the server firewall and security group again, you can. The validation is done until the next time the certificate must be renewed.

Update the Apache Configuration to Use the New SSL Certificate

Now update the Apache site configuration in /etc/apache2/sites-available/default-ssl.conf to use the Let's Encrypt certificate and enable strict transport security:

```
# Set the HTTP Strict Transport Security (HSTS) header to guarantee
# HTTPS for 1 Year, including subdomains, and allow this site to be
# added to the preload list.
#
# Do NOT enable this until you have the final SSL certificate in
# in place. You can get stuck.
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains; pr

...

# A self-signed (snakeoil) certificate can be created by installing
# the ssl-cert package. See
```

```
# /usr/share/doc/apache2/README.Debian.gz for more info.
# If both key and certificate are stored in the same file, only the
# SSLCertificateFile directive is needed.
#
# Use the Let's Encrypt certificate.
SSLCertificateFile    /etc/letsencrypt/live/mail.example.com/cert.pem
SSLCertificateKeyFile /etc/letsencrypt/live/mail.example.com/privkey.pem

# Server Certificate Chain:
# Point SSLCertificateChainFile at a file containing the
# concatenation of PEM encoded CA certificates which form the
# certificate chain for the server certificate. Alternatively
# the referenced file can be the same as SSLCertificateFile
# when the CA certificates are directly appended to the server
# certificate for convinience.
#
# Use the Let's Encrypt certificate.
SSLCertificateChainFile /etc/letsencrypt/live/mail.example.com/chain.pem
```

Reload the Apache configuration:

```
service apache2 reload
```

Install the Mailserver Packages

Now we're ready to start in on the harder stuff. As for the LAMP server, there is a shortcut for installing the basic packages for a mail server. Again, note the "^" at the end of the command:

```
apt-get install --assume-yes mail-server^
```

When Postfix installs, you will be asked to choose a general type of mail configuration. At this point select Internet site. You will be asked for the system mail name, which is the hostname of your mailserver, e.g. mail.example.com. What this set of packages provides to you is pretty much just the bare bones, aimed at setting up a mailserver that manages its users as straightforward Unix users and doesn't use a SQL database to store data. That is not the goal here, so we need the rest of the cast, such as MySQL support for Postfix and Dovecot, and a coterie of spam-mashing packages:

```
apt-get install --assume-yes \
  postfix-mysql \
  dovecot-mysql \
```

```
  postgrey \
  amavis \
  clamav \
  clamav-daemon \
  spamassassin \
  libdbi-perl \
  libdbd-mysql-perl \
  php7.2-imap \
  postfix-policyd-spf-python
```

The libdbi-perl and libdbd-mysql-perl packages are required by Amavis. The php7.2-imap package actually provides support for POP3 as well as the IMAP protocol, and will be needed by Postfix Admin and most of the possible options for PHP webmail applications. The postfix-policyd-spf-python package provides checks of Sender Policy Framework (SPF) on incoming mail to block spam. You will want to restart Apache at this point to have php7.2-imap running and ready:

```
service apache2 restart
```

Next you'll want to install a few optional packages that extend the abilities of the spam and virus detection packages by allowing greater inspection of attached files:

```
apt-get install --assume-yes \
  pyzor \
  razor \
  arj \
  cabextract \
  lzop \
  nomarch \
  p7zip-full \
  ripole \
  rpm2cpio \
  tnef \
  unzip \
  unrar-free \
  zip
```

Configure MySQL

A few alterations to the default MySQL configuration in `/etc/mysql/mysql.conf.d/mysqld.cnf` are needed. Add the following:

```
# This removes NO_ZERO_IN_DATE and NO_ZERO_DATE, which cause problems for
```

```
# Postfix Admin code, from strict mode.
sql_mode=ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CR
```

Then restart the MySQL server:

```
service mysql restart
```

Create a Mail Database and User in MySQL

Log in to MySQL as the root user without a password:

```
mysql -uroot
```

Set a root password:

```
update mysql.user
  set authentication_string=password('rootpassword'),
  plugin='mysql_native_password'
where User='root';
flush privileges;
```

Now set up a database and user for the mail software. This database will store information on user accounts and mail domains, using schema set up by the Postfix Admin package. You will want to pick a better password than the example used below, but note that it will be entered into numerous configuration files. If an attacker gains shell access to the server, it doesn't matter how good the password is. Protection from unauthorized access to the database is largely provided by the fact that access to MySQL is restricted by the firewall, but it never hurts to have good passwords in place regardless: to put in place multiple layers of defense is always a good strategy.

```
create database mail;
create user 'mail'@'localhost' identified by 'mailpassword';
grant all on mail.* to 'mail'@'localhost';
```

Install Postfix Admin 3.2 and the MySQL Schema

Postfix Admin is installed as follows. To start things off, download the package from Sourceforge, unpack it, move it to a destination directory, symlink to the webroot, create the templates_c directory, and change ownership to the www-data user:

```
wget http://downloads.sourceforge.net/project/postfixadmin/postfixadmin/postfixadmin-
tar -xf postfixadmin-3.2.tar.gz
rm -f postfixadmin-3.2.tar.gz
mv postfixadmin-3.2 /srv/postfixadmin
ln -s /srv/postfixadmin/public /var/www/html/postfixadmin
mkdir -p /srv/postfixadmin/templates_c
chown -R www-data /srv/postfixadmin/templates_c
```

Next up is an interesting sort of a two-phase setup process. Firstly create a new empty file /srv/postfixadmin/config.local.php :

```
touch /srv/postfixadmin/config.local.php
```

Entries made in /srv/postfixadmin/config.local.php will override the default configuration in /srv/postfixadmin/config.inc.php . Add the following lines:

```
<?php
// Configuration options here override those in config.inc.php.

// You have to set $CONF['configured'] = true; before the
// application will run.
$CONF['configured'] = true;

// Postfix Admin Path
// Set the location of your Postfix Admin installation here.
// YOU MUST ENTER THE COMPLETE URL e.g. http://domain.tld/postfixadmin
$CONF['postfix_admin_url'] = 'https://mail.example.com/postfixadmin';

// Database connection details.
$CONF['database_type'] = 'mysqli';
$CONF['database_host'] = 'localhost';
$CONF['database_user'] = 'mail';
$CONF['database_password'] = 'mailpassword';
$CONF['database_name'] = 'mail';

// Site Admin
// Define the Site Admin's email address below.
```

```
// This will be used to send emails from to create mailboxes and
// from Send Email / Broadcast message pages.
// Leave blank to send email from the logged-in Admin's Email address.
$CONF['admin_email'] = 'admin@example.com';

// Mail Server
// Hostname (FQDN) of your mail server.
// This is used to send email to Postfix in order to create mailboxes.
$CONF['smtp_server'] = 'localhost';
$CONF['smtp_port'] = '25';

// Encrypt
// In what way do you want the passwords to be crypted?
// md5crypt = internal postfix admin md5
$CONF['encrypt'] = 'md5crypt';

// Default Aliases
// The default aliases that need to be created for all domains.
$CONF['default_aliases'] = array (
    'abuse' => 'admin@example.com',
    'hostmaster' => 'admin@example.com',
    'postmaster' => 'admin@example.com',
    'webmaster' => 'admin@example.com'
);

// Footer
// Below information will be on all pages.
// If you don't want the footer information to appear set this to 'NO'.
$CONF['show_footer_text'] = 'YES';
$CONF['footer_text'] = 'Return to mail.example.com';
$CONF['footer_link'] = 'https://mail.example.com';

// Mailboxes
// If you want to store the mailboxes per domain set this to 'YES'.
// Examples:
//   YES: /usr/local/virtual/domain.tld/username@domain.tld
//   NO:  /usr/local/virtual/username@domain.tld
$CONF['domain_path'] = 'NO';
// If you don't want to have the domain in your mailbox set this to 'NO'.
// Examples:
//   YES: /usr/local/virtual/domain.tld/username@domain.tld
//   NO:  /usr/local/virtual/domain.tld/username
// Note: If $CONF['domain_path'] is set to NO, this setting will be forced to YES.
$CONF['domain_in_mailbox'] = 'YES';

// Specify '' for Dovecot and 'INBOX.' for Courier.
$CONF['create_mailbox_subdirs_prefix']='';
```

Note that the last items above are only for the purposes of defining how Postfix Admin stores its data - they don't set system paths for mailboxes. The actual system paths to virtual mailbox directories are defined in the Dovecot configuration outlined in a later section of this post. There are a lot more configuraton options relating to optional functionality that can be used in Postfix Admin. It is worth exploring the documentation and the main configuration file `/srv/postfixadmin/config.inc.php`.

Next open up a web browser and visit your mail server at:

```
https://mail.example.com/postfixadmin/setup.php
```

Postfix Admin will automatically create its database schema at this point. Follow the instructions on that page to choose a setup password and generate a hash of that password. Add that hash to the configuration file `/var/www/html/postfixadmin/config.local.php` and save it:

```
// In order to setup Postfixadmin, you MUST specify a hashed password here.
// To create the hash, visit setup.php in a browser and type a password into the fiel
// on submission it will be echoed out to you as a hashed value.
$CONF['setup_password'] = '...a long hash string...';
```

Then return to the setup page. You can now use the password you selected in order to create an initial administrator account.

You should close off access to `http://mail.example.com/postfixadmin/setup.php` after having used it. Create a file `/srv/postfixadmin/public/.htaccess` and put the following instructions into it:

```
<Files "setup.php">
deny from all
</Files>
```

Create the Domain and Accounts in Postfix Admin

Now navigate to the main Postfix Admin login page:

```
https://mail.example.com/postfixadmin/
```

Log in as the newly created administrator account, and then choose the Domain List -> New Domain option in order to create the example.com domain. Make sure that you check Add default mail aliases and note that you will probably want to change the default limits to allow yourself unlimited mailboxes and aliases. Next navigate to Domain List -> Domain List and click on the name of your domain to view it. From that page you can then add mail users (Add mailbox) and aliases (Add alias). This will populate the schema, but it won't do anything else at this point as none of the other mail server components are yet configured to use the database.

Make sure that you create a mailbox for admin@example.com since your default aliases are redirecting to that address. While the alias addresses do tend to receive spam, it is a very good idea to pay attention to legitimate mail arriving at these addresses, as it can alert you to problems. This is less important for a personal mail server, but very important if you are in the business of sending mail and want to maintain good deliverability.

Postfix Admin does have another useful function during this lengthy setup process: it allows you to send mail to local users through the web interface, which is helpful when testing your configuration and chasing down errors.

Create a User to Handle Virtual Mail Directories

Virtual mail users are those that do not exist as Unix system users. They thus don't use the standard Unix methods of authentication or mail delivery and don't have home directories. That is how we are managing things here: mail users are defined in the database created by Postfix Admin rather than existing as system users. Mail will be kept in subfolders per domain and account under /var/vmail - e.g. admin@example.com will have a mail directory of /var/vmail/example.com/admin. All of these mail directories will be owned by a single user called vmail, and Dovecot will use the vmail user in order to create and update mail files.

```
useradd -r -u 150 -g mail -d /var/vmail -s /sbin/nologin -c "Virtual maildir handler"
mkdir /var/vmail
chmod 770 /var/vmail
chown vmail:mail /var/vmail
```

Note that the user and virtual mail directory folder are using the "mail" group, thus allowing other system users in that group to modify the contents.

Configure Dovecot

Dovecot will manage IMAP and POP3 connections, local mail directories, and receive incoming mail that is handed off from Postfix. It will also manage authentication for SMTP connections as there is no point in having two separate

authentication systems when Dovecot can handle both cases. Configuration is spread across a number of files in `/etc/dovecot` and subfolders thereof, and while it might at first seem intimidating, it is all laid out fairly logically. The first thing to do is ensure that Dovecot is looking for user data in the database created by Postfix Admin, so you will want to edit these lines in `/etc/dovecot/dovecot-sql.conf.ext` such that it uses the MySQL database created by Postfix Admin:

```
# Database driver: mysql, pgsql, sqlite
driver = mysql
```

```
# Examples:
#   connect = host=192.168.1.1 dbname=users
#   connect = host=sql.example.com dbname=virtual user=virtual password=blarg
#   connect = /etc/dovecot/authdb.sqlite
#
connect = host=localhost dbname=mail user=mail password=mailpassword
```

```
# Default password scheme.
#
# List of supported schemes is in
# http://wiki2.dovecot.org/Authentication/PasswordSchemes
#
default_pass_scheme = MD5-CRYPT
```

```
# Define the query to obtain a user password.
#
# Note that uid 150 is the "vmail" user and gid 8 is the "mail" group.
#
password_query = \
  SELECT username as user, password, '/var/vmail/%d/%n' as userdb_home, \
  'maildir:/var/vmail/%d/%n' as userdb_mail, 150 as userdb_uid, 8 as userdb_gid \
  FROM mailbox WHERE username = '%u' AND active = '1'
```

```
# Define the query to obtain user information.
#
# Note that uid 150 is the "vmail" user and gid 8 is the "mail" group.
#
user_query = \
  SELECT '/var/vmail/%d/%n' as home, 'maildir:/var/vmail/%d/%n' as mail, \
  150 AS uid, 8 AS gid, concat('dirsize:storage=', quota) AS quota \
  FROM mailbox WHERE username = '%u' AND active = '1'
```

Then change the controlling definitions in /etc/dovecot/conf.d/10-auth.conf such that Dovecot will read the SQL configuration files. While you are there, you should also make sure that plaintext authentication is disabled unless the connection is encrypted or local:

```
# Disable LOGIN command and all other plaintext authentications unless
# SSL/TLS is used (LOGINDISABLED capability). Note that if the remote IP
# matches the local IP (ie. you're connecting from the same computer), the
# connection is considered secure and plaintext authentication is allowed.
disable_plaintext_auth = yes
```

```
# Space separated list of wanted authentication mechanisms:
#   plain login digest-md5 cram-md5 ntlm rpa apop anonymous gssapi otp skey
#   gss-spnego
# NOTE: See also disable_plaintext_auth setting.
auth_mechanisms = plain login
```

```
##
## Password and user databases
##

#
# Password database is used to verify user's password (and nothing more).
# You can have multiple passdbs and userdbs. This is useful if you want to
# allow both system users (/etc/passwd) and virtual users to login without
# duplicating the system users into virtual database.
#
# <doc/wiki/PasswordDatabase.txt>
#
# User database specifies where mails are located and what user/group IDs
# own them. For single-UID configuration use "static" userdb.
#
# <doc/wiki/UserDatabase.txt>

#!include auth-deny.conf.ext
#!include auth-master.conf.ext

#!include auth-system.conf.ext
# Use the SQL database configuration for authentication rather than
# any of these others.
!include auth-sql.conf.ext
#!include auth-ldap.conf.ext
#!include auth-passwdfile.conf.ext
#!include auth-checkpassword.conf.ext
```

```
#!include auth-vpopmail.conf.ext
#!include auth-static.conf.ext
```

Next up, you must tell Dovecot where to put the virtual user mail directories. That requires the following changes in /etc/dovecot/conf.d/10-mail.conf:

```
# Location for users' mailboxes. The default is empty, which means that Dovecot
# tries to find the mailboxes automatically. This won't work if the user
# doesn't yet have any mail, so you should explicitly tell Dovecot the full
# location.
#
# If you're using mbox, giving a path to the INBOX file (eg. /var/mail/%u)
# isn't enough. You'll also need to tell Dovecot where the other mailboxes are
# kept. This is called the "root mail directory", and it must be the first
# path given in the mail_location setting.
#
# There are a few special variables you can use, eg.:
#
#   %u - username
#   %n - user part in user@domain, same as %u if there's no domain
#   %d - domain part in user@domain, empty if there's no domain
#   %h - home directory
#
# See doc/wiki/Variables.txt for full list. Some examples:
#
#   mail_location = maildir:~/Maildir
#   mail_location = mbox:~/mail:INBOX=/var/mail/%u
#   mail_location = mbox:/var/mail/%d/%1n/%n:INDEX=/var/indexes/%d/%1n/%n
#
# <doc/wiki/MailLocation.txt>
#
mail_location = maildir:/var/vmail/%d/%n
```

```
# System user and group used to access mails. If you use multiple, userdb
# can override these by returning uid or gid fields. You can use either numbers
# or names. <doc/wiki/UserIds.txt>
mail_uid = vmail
mail_gid = mail
```

```
# Valid UID range for users, defaults to 500 and above. This is mostly
# to make sure that users can't log in as daemons or other system users.
# Note that denying root logins is hardcoded to dovecot binary and can't
```

```
# be done even if first_valid_uid is set to 0.
#
# Use the vmail user uid here.
first_valid_uid = 150
last_valid_uid = 150
```

Let Dovecot know about either your snakeoil or purchased certificate by editing these lines in /etc/dovecot/conf.d/10-ssl.conf. Remember to include your CA certificate bundle if provided with one by the certificate issuer:

```
# SSL/TLS support: yes, no, required. <doc/wiki/SSL.txt>
ssl = yes

# PEM encoded X.509 SSL/TLS certificate and private key. They're opened before
# dropping root privileges, so keep the key file unreadable by anyone but
# root. Included doc/mkcert.sh can be used to easily generate self-signed
# certificate, just make sure to update the domains in dovecot-openssl.cnf
#
# The generated snakeoil certificate:
#ssl_cert = </etc/ssl/certs/ssl-cert-snakeoil.pem
#ssl_key = </etc/ssl/private/ssl-cert-snakeoil.key
# Let's Encrypt certificate:
ssl_cert = </etc/letsencrypt/live/mail.example.com/cert.pem
ssl_key = </etc/letsencrypt/live/mail.example.com/privkey.pem

# If key file is password protected, give the password here. Alternatively
# give it when starting dovecot with -p parameter. Since this file is often
# world-readable, you may want to place this setting instead to a different
# root owned 0600 file by using ssl_key_password = <path.
#ssl_key_password =

# PEM encoded trusted certificate authority. Set this only if you intend to use
# ssl_verify_client_cert=yes. The file should contain the CA certificate(s)
# followed by the matching CRL(s). (e.g. ssl_ca = </etc/ssl/certs/ca.pem)
ssl_ca = </etc/letsencrypt/live/mail.example.com/chain.pem
```

You must also update the following lines in /etc/dovecot/conf.d/10-ssl.conf to ensure that some SSL protocols that are no longer secure are not used:

```
# DH parameters length to use. In light of Logjam, has to be 2048 or more.
# See: https://weakdh.org/sysadmin.html
ssl_dh_parameters_length = 2048

# SSL protocols to use. Don't use the no-longer secure protocols.
```

```
ssl_protocols = !SSLv2 !SSLv3

# SSL ciphers to use. See:
# https://weakdh.org/sysadmin.html
# https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
ssl_cipher_list = ECDH+AESGCM:ECDH+CHACHA20:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES

# Prefer the server's order of ciphers over client's.
ssl_prefer_server_ciphers = yes
```

Next, edit these lines in `/etc/dovecot/conf.d/10-master.conf` to add the Postfix option:

```
service auth {
  # auth_socket_path points to this userdb socket by default. It's typically
  # used by dovecot-lda, doveadm, possibly imap process, etc. Users that have
  # full permissions to this socket are able to get a list of all usernames and
  # get the results of everyone's userdb lookups.
  #
  # The default 0666 mode allows anyone to connect to the socket, but the
  # userdb lookups will succeed only if the userdb returns an "uid" field that
  # matches the caller process's UID. Also if caller's uid or gid matches the
  # socket's uid or gid the lookup succeeds. Anything else causes a failure.
  #
  # To give the caller full permissions to lookup all users, set the mode to
  # something else than 0666 and Dovecot lets the kernel enforce the
  # permissions (e.g. 0777 allows everyone full permissions).
  unix_listener auth-userdb {
    mode = 0666
    user = vmail
    group = mail
  }

  unix_listener /var/spool/postfix/private/auth {
    mode = 0666
    # Assuming the default Postfix user and group
    user = postfix
    group = postfix
  }
```

You may have to explicitly set a postmaster address in `/etc/dovecot/conf.d/15-lda.conf`; if you see "Invalid settings: postmaster_address setting not given" showing up in the mail log, then this is the fix for that. Make sure that a suitable alias or mailbox exists for your chosen postmaster address:

```
# Address to use when sending rejection mails.
# Default is postmaster@<your domain>.
postmaster_address = postmaster@example.com
```

You must now ensure that the Dovecot configuration is accessible to both dovecot and vmail users:

```
chown -R vmail:dovecot /etc/dovecot
chmod -R o-rwx /etc/dovecot
```

A final note on Dovecot: it only creates a user's mail directory when mail is first delivered to that virtual user. So creating a user in Postfix Admin will not result in the immediate creation of a mail directory under `/var/vmail`, and that's just fine.

Configure Amavis, ClamAV, and SpamAssassin

Before configuring Postfix, we may as well take a short detour into configuring the spam and virus tools. Their default configuration is close to what most people will need, and tools like SpamAssassin auto-detect many of the optional additional packages you may have installed. If you have specialist needs or greater knowledge, you can of course spend a fair amount of time here crafting intricate rules. For the casual user, this is a quick and straightforward process, however. Note that here we are putting off the portions relating to integration with Postfix - e.g. additions to the `/etc/postfix/master.cf` file - into the Postfix section of this post.

First add Amavis and ClamAV users to one another's groups to enable them to collaborate:

```
adduser clamav amavis
adduser amavis clamav
```

This also requires editing the following lines in `/etc/clamav/clamd.conf`:

```
# Needed to allow things to work with Amavis, when both amavis and clamav
# users are added to one another's groups.
AllowSupplementaryGroups true
```

Then turn on Amavis by editing `/etc/amavis/conf.d/15-content_filter_mode`. The software is disabled by default, so uncomment the @bypass... lines:

```
use strict;

```

```
# You can modify this file to re-enable SPAM checking through spamassassin
# and to re-enable antivirus checking.

#
# Default antivirus checking mode
# Please note, that anti-virus checking is DISABLED by
# default.
# If You wish to enable it, please uncomment the following lines:

@bypass_virus_checks_maps = (
   \%bypass_virus_checks, \@bypass_virus_checks_acl, \$bypass_virus_checks_re);

#
# Default SPAM checking mode
# Please note, that anti-spam checking is DISABLED by
# default.
# If You wish to enable it, please uncomment the following lines:

@bypass_spam_checks_maps = (
   \%bypass_spam_checks, \@bypass_spam_checks_acl, \$bypass_spam_checks_re);

1;  # ensure a defined return
```

Now enable SpamAssassin by editing these lines in /etc/default/spamassassin:

```
# Change to one to enable spamd
ENABLED=1
```

```
# Cronjob
# Set to anything but 0 to enable the cron job to automatically update
# spamassassin's rules on a nightly basis
CRON=1
```

SpamAssassin under Amavis will only check mail that's determined to be arriving for local delivery. There are a couple of ways to tell Amavis which mails are for local delivery, but here we'll set it up to check the database set up by Postfix Admin. Edit /etc/amavis/conf.d/50-user to look like this:

```
use strict;

#
# Place your configuration directives here.  They will override those in
# earlier files.
```

```
#
# See /usr/share/doc/amavisd-new/ for documentation and examples of
# the directives you can use in this file
#

# Three concurrent processes. This should fit into the RAM available on an
# AWS micro instance. This has to match the number of processes specified
# for Amavis in /etc/postfix/master.cf.
$max_servers  = 3;

# Add spam info headers if at or above that level - this ensures they
# are always added.
$sa_tag_level_deflt  = -9999;

# Check the database to see if mail is for local delivery, and thus
# should be spam checked.
@lookup_sql_dsn = (
    ['DBI:mysql:database=mail;host=127.0.0.1;port=3306',
     'mail',
     'mailpassword']);
$sql_select_policy = 'SELECT domain from domain WHERE CONCAT("@",domain) IN (%k)';

# Uncomment to bump up the log level when testing.
# $log_level = 2;

#------------ Do not modify anything below this line -------------
1;  # ensure a defined return
```

Next make sure the ClamAV database is up to date by running freshclam. It should be:

```
freshclam
```

You will have to restart these processes to pick up the new configuration:

```
service clamav-daemon restart
service amavis restart
service spamassassin restart
```

Configure Postfix

Postfix handles incoming mail via the SMTP protocol, and its configuration files have be set up to allow it to integrate with the various other packages we have installed so far. At a high level, we want Postfix to hand off incoming mail to the spam and virus checkers before passing it on to Dovecot for delivery, and to communicate with Dovecot in order to authenticate virtual users who are connecting over SMTP in order to to send mail.

Firstly you must create a set of files that describe for Postfix where to find information on users and domains. Note that the "hosts" directive in these files must be exactly the same as the "bind-address" in the MySQL server configuration. If one side says "localhost" and the other side says "127.0.0.1" then you may find that Postfix cannot connect to MySQL - strange but true. Here are the needed Postfix files:

/etc/postfix/mysql_virtual_alias_domainaliases_maps.cf

```
user = mail
password = mailpassword
hosts = 127.0.0.1
dbname = mail
query = SELECT goto FROM alias,alias_domain
  WHERE alias_domain.alias_domain = '%d'
  AND alias.address=concat('%u', '@', alias_domain.target_domain)
  AND alias.active = 1
```

/etc/postfix/mysql_virtual_alias_maps.cf

```
user = mail
password = mailpassword
hosts = 127.0.0.1
dbname = mail
table = alias
select_field = goto
where_field = address
additional_conditions = and active = '1'
```

/etc/postfix/mysql_virtual_domains_maps.cf

```
user = mail
password = mailpassword
hosts = 127.0.0.1
dbname = mail
table = domain
select_field = domain
```

```
where_field = domain
additional_conditions = and backupmx = '0' and active = '1'
```

/etc/postfix/mysql_virtual_mailbox_domainaliases_maps.cf

```
user = mail
password = mailpassword
hosts = 127.0.0.1
dbname = mail
query = SELECT maildir FROM mailbox, alias_domain
  WHERE alias_domain.alias_domain = '%d'
  AND mailbox.username=concat('%u', '@', alias_domain.target_domain )
  AND mailbox.active = 1
```

/etc/postfix/mysql_virtual_mailbox_maps.cf

```
user = mail
password = mailpassword
hosts = 127.0.0.1
dbname = mail
table = mailbox
select_field = CONCAT(domain, '/', local_part)
where_field = username
additional_conditions = and active = '1'
```

/etc/postfix/mysql_virtual_sender_login_maps.cf

```
user = mail
password = mailpassword
hosts = 127.0.0.1
dbname = mail
query = SELECT goto FROM alias WHERE address='%s'
```

Now create the file /etc/postfix/header_checks, which will contain some directives to remove certain headers when relaying mail. This improves privacy for the sending users by such things as stripping the original IP address and mail software identifiers, for example. This file will be referenced in the main Postfix configuration:

```
/^Received:/                IGNORE
/^User-Agent:/              IGNORE
/^X-Mailer:/                IGNORE
```

```
/^X-Originating-IP:/          IGNORE
/^x-cr-[a-z]*:/               IGNORE
/^Thread-Index:/              IGNORE
```

The following is the complete main Postfix configuration file at /etc/postfix/main.cf, which contains a fair number of complex choices and options on how mail is relayed and how SMTP behaves. It is far beyond the scope of this post to explain each and every choice of best practice or configuration parameter in detail. I strongly suggest that you spend some time reading up on Postfix configuration, as this is where it is easy to fall down and produce a suboptimal or faulty mailserver.

```
# See /usr/share/postfix/main.cf.dist for a commented, more complete version

# The first text sent to a connecting process.
smtpd_banner = $myhostname ESMTP $mail_name
biff = no
# appending .domain is the MUA's job.
append_dot_mydomain = no
readme_directory = no

# --------------------------------
# SASL parameters
# --------------------------------

# Use Dovecot to authenticate.
smtpd_sasl_type = dovecot
# Referring to /var/spool/postfix/private/auth
smtpd_sasl_path = private/auth
smtpd_sasl_auth_enable = yes
broken_sasl_auth_clients = yes
smtpd_sasl_security_options = noanonymous
smtpd_sasl_local_domain =
smtpd_sasl_authenticated_header = yes

# --------------------------------
# TLS parameters
# --------------------------------

# The default snakeoil certificate.
#smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
#smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key

# Let's Encrypt certificate.
smtpd_tls_cert_file=/etc/letsencrypt/live/mail.example.com/cert.pem
smtpd_tls_key_file=/etc/letsencrypt/live/mail.example.com/privkey.pem
```

```
smtpd_tls_CAfile=/etc/letsencrypt/live/mail.example.com/chain.pem

# Ensure we're not using no-longer-secure protocols.
smtpd_tls_mandatory_protocols=!SSLv2,!SSLv3

smtp_tls_note_starttls_offer = yes
smtpd_tls_loglevel = 1
smtpd_tls_received_header = yes
smtpd_tls_session_cache_timeout = 3600s
tls_random_source = dev:/dev/urandom
#smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
#smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache

# Note that forcing use of TLS is going to cause breakage - most mail servers
# don't offer it and so delivery will fail, both incoming and outgoing. This is
# unfortunate given what various governmental agencies are up to these days.
#
# Enable (but don't force) all incoming smtp connections to use TLS.
smtpd_tls_security_level = may
# Enable (but don't force) all outgoing smtp connections to use TLS.
smtp_tls_security_level = may

# See /usr/share/doc/postfix/TLS_README.gz in the postfix-doc package for
# information on enabling SSL in the smtp client.

# --------------------------------
# TLS Updates relating to Logjam SSL attacks.
# See: https://weakdh.org/sysadmin.html
# --------------------------------

smtpd_tls_exclude_ciphers = aNULL, eNULL, EXPORT, DES, RC4, MD5, PSK, aECDH, EDH-DS
smtpd_tls_dh1024_param_file = /etc/ssl/private/dhparams.pem

# --------------------------------
# SMTPD parameters
# --------------------------------

# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h
# will it be a permanent error or temporary
unknown_local_recipient_reject_code = 450
# how long to keep message on queue before return as failed.
maximal_queue_lifetime = 7d
# max and min time in seconds between retries if connection failed
minimal_backoff_time = 1000s
maximal_backoff_time = 8000s
# how long to wait when servers connect before receiving rest of data
```

```
smtp_helo_timeout = 60s
# how many address can be used in one message.
# effective stopper to mass spammers, accidental copy in whole address list
# but may restrict intentional mail shots.
smtpd_recipient_limit = 16
# how many error before back off.
smtpd_soft_error_limit = 3
# how many max errors before blocking it.
smtpd_hard_error_limit = 12

# This next set are important for determining who can send mail and relay mail
# to other servers. It is very important to get this right - accidentally producing
# an open relay that allows unauthenticated sending of mail is a Very Bad Thing.
#
# You are encouraged to read up on what exactly each of these options accomplish.

# Requirements for the HELO statement
smtpd_helo_restrictions = permit_mynetworks, warn_if_reject reject_non_fqdn_hostnam
# Requirements for the sender details. Note that the order matters.
# E.g. see http://jimsun.linxnet.com/misc/restriction_order_prelim-03.txt
smtpd_sender_restrictions = permit_mynetworks, reject_authenticated_sender_login_mi
# Requirements for the connecting server
smtpd_client_restrictions = reject_rbl_client sbl.spamhaus.org, reject_rbl_client b
# Requirement for the recipient address. Note that the entry for
# "check_policy_service inet:127.0.0.1:10023" enables Postgrey.
smtpd_recipient_restrictions = reject_unauth_pipelining, permit_mynetworks, permit_
smtpd_data_restrictions = reject_unauth_pipelining
# This is a new option as of Postfix 2.10, and is required in addition to
# smtpd_recipient_restrictions for things to work properly in this setup.
smtpd_relay_restrictions = reject_unauth_pipelining, permit_mynetworks, permit_sasl

# require proper helo at connections
smtpd_helo_required = yes
# waste spammers time before rejecting them
smtpd_delay_reject = yes
disable_vrfy_command = yes

# ---------------------------------
# General host and delivery info
# ---------------------------------

myhostname = mail.example.com
myorigin = /etc/hostname
# Some people see issues when setting mydestination explicitly to the server
# subdomain, while leaving it empty generally doesn't hurt. So it is left empty her
# mydestination = mail.example.com, localhost
mydestination =
```

```
# If you have a separate web server that sends outgoing mail through this
# mailserver, you may want to add its IP address to the space-delimited list in
# mynetworks, e.g. as 10.10.10.10/32.
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
mynetworks_style = host

# This specifies where the virtual mailbox folders will be located.
virtual_mailbox_base = /var/vmail
# This is for the mailbox location for each user. The domainaliases
# map allows us to make use of Postfix Admin's domain alias feature.
virtual_mailbox_maps = mysql:/etc/postfix/mysql_virtual_mailbox_maps.cf, mysql:/etc
# and their user id
virtual_uid_maps = static:150
# and group id
virtual_gid_maps = static:8
# This is for aliases. The domainaliases map allows us to make
# use of Postfix Admin's domain alias feature.
virtual_alias_maps = mysql:/etc/postfix/mysql_virtual_alias_maps.cf, mysql:/etc/pos
# This is for domain lookups.
virtual_mailbox_domains = mysql:/etc/postfix/mysql_virtual_domains_maps.cf
# Used in conjunction with reject_authenticated_sender_login_mismatch to
# verify that the sender is sending with their own address, or with one
# of the aliases mapped to that address.
smtpd_sender_login_maps = mysql:/etc/postfix/mysql_virtual_sender_login_maps.cf

# --------------------------------
# Integration with other packages
# --------------------------------------

# Tell postfix to hand off mail to the definition for dovecot in master.cf
virtual_transport = dovecot
dovecot_destination_recipient_limit = 1

# Use amavis for virus and spam scanning
content_filter = amavis:[127.0.0.1]:10024

# Settings for checking SPF to cut down spam.
policy-spf_time_limit = 3600s

# --------------------------------
# Header manipulation
# ---------------------------------------

# Getting rid of unwanted headers. See: https://posluns.com/guides/header-removal/
```

```
header_checks = regexp:/etc/postfix/header_checks
# getting rid of x-original-to
enable_original_recipient = no
```

You must also expand /etc/postfix/master.cf, and here is the entire file for clarity. This includes much of the default material from the package install, such as commented options:

```
#
# Postfix master process configuration file.  For details on the format
# of the file, see the master(5) manual page (command: "man 5 master" or
# on-line: http://www.postfix.org/master.5.html).
#
# Do not forget to execute "postfix reload" after editing this file.
#
# ==========================================================================
# service type  private unpriv  chroot  wakeup  maxproc command + args
#               (yes)   (yes)   (no)    (never) (100)
# ==========================================================================
smtp      inet  n       -       y       -       -       smtpd
#smtp      inet  n       -       y       -       1       postscreen
#smtpd     pass  -       -       y       -       -       smtpd
#dnsblog   unix  -       -       y       -       0       dnsblog
#tlsproxy  unix  -       -       y       -       0       tlsproxy

# SMTP with TLS on port 587. Currently commented.
#submission inet n       -       y       -       -       smtpd
#  -o syslog_name=postfix/submission
#  -o smtpd_tls_security_level=encrypt
#  -o smtpd_sasl_auth_enable=yes
#  -o smtpd_enforce_tls=yes
#  -o smtpd_client_restrictions=permit_sasl_authenticated,reject_unauth_destinatior
#  -o smtpd_sasl_tls_security_options=noanonymous

# SMTP over SSL on port 465.
smtps     inet  n       -       y       -       -       smtpd
  -o syslog_name=postfix/smtps
  -o smtpd_tls_wrappermode=yes
  -o smtpd_sasl_auth_enable=yes
  -o smtpd_tls_auth_only=yes
  -o smtpd_client_restrictions=permit_sasl_authenticated,reject_unauth_destination,
  -o smtpd_sasl_security_options=noanonymous,noplaintext
  -o smtpd_sasl_tls_security_options=noanonymous

#628       inet  n       -       y       -       -       qmqpd
```

```
pickup    unix  n       -       y       60      1       pickup
cleanup   unix  n       -       y       -       0       cleanup
qmgr      unix  n       -       n       300     1       qmgr
#qmgr     unix  n       -       n       300     1       oqmgr
tlsmgr    unix  -       -       y       1000?   1       tlsmgr
rewrite   unix  -       -       y       -       -       trivial-rewrite
bounce    unix  -       -       y       -       0       bounce
defer     unix  -       -       y       -       0       bounce
trace     unix  -       -       y       -       0       bounce
verify    unix  -       -       y       -       1       verify
flush     unix  n       -       y       1000?   0       flush
proxymap  unix  -       -       n       -       -       proxymap
proxywrite unix -       -       n       -       1       proxymap
smtp      unix  -       -       y       -       -       smtp
relay     unix  -       -       y       -       -       smtp
#       -o smtp_helo_timeout=5 -o smtp_connect_timeout=5
showq     unix  n       -       y       -       -       showq
error     unix  -       -       y       -       -       error
retry     unix  -       -       y       -       -       error
discard   unix  -       -       y       -       -       discard
local     unix  -       n       n       -       -       local
virtual   unix  -       n       n       -       -       virtual
lmtp      unix  -       -       y       -       -       lmtp
anvil     unix  -       -       y       -       1       anvil
scache    unix  -       -       y       -       1       scache
#
# ====================================================================
# Interfaces to non-Postfix software. Be sure to examine the manual
# pages of the non-Postfix software to find out what options it wants.
#
# Many of the following services use the Postfix pipe(8) delivery
# agent.  See the pipe(8) man page for information about ${recipient}
# and other message envelope options.
# ====================================================================
#
# maildrop. See the Postfix MAILDROP_README file for details.
# Also specify in main.cf: maildrop_destination_recipient_limit=1
#
maildrop  unix  -       n       n       -       -       pipe
  flags=DRhu user=vmail argv=/usr/bin/maildrop -d ${recipient}
#
# ====================================================================
#
# Recent Cyrus versions can use the existing "lmtp" master.cf entry.
#
# Specify in cyrus.conf:
#   lmtp    cmd="lmtpd -a" listen="localhost:lmtp" proto=tcp4
```

```
#
# Specify in main.cf one or more of the following:
#  mailbox_transport = lmtp:inet:localhost
#  virtual_transport = lmtp:inet:localhost
#
# ====================================================================
#
# Cyrus 2.1.5 (Amos Gouaux)
# Also specify in main.cf: cyrus_destination_recipient_limit=1
#
#cyrus     unix  -       n       n       -       -       pipe
#  user=cyrus argv=/cyrus/bin/deliver -e -r ${sender} -m ${extension} ${user}
#
# ====================================================================
# Old example of delivery via Cyrus.
#
#old-cyrus unix  -       n       n       -       -       pipe
#  flags=R user=cyrus argv=/cyrus/bin/deliver -e -m ${extension} ${user}
#
# ====================================================================
#
# See the Postfix UUCP_README file for configuration details.
#
uucp      unix  -       n       n       -       -       pipe
  flags=Fqhu user=uucp argv=uux -r -n -z -a$sender - $nexthop!rmail ($recipient)
#
# Other external delivery methods.
#
ifmail    unix  -       n       n       -       -       pipe
  flags=F user=ftn argv=/usr/lib/ifmail/ifmail -r $nexthop ($recipient)
bsmtp     unix  -       n       n       -       -       pipe
  flags=Fq. user=bsmtp argv=/usr/lib/bsmtp/bsmtp -t$nexthop -f$sender $recipient
scalemail-backend unix  - n n - 2 pipe
  flags=R user=scalemail argv=/usr/lib/scalemail/bin/scalemail-store ${nexthop} ${u
mailman   unix  -       n       n       -       -       pipe
  flags=FR user=list argv=/usr/lib/mailman/bin/postfix-to-mailman.py
  ${nexthop} ${user}

# The next two entries integrate with Amavis for anti-virus/spam checks.
amavis      unix    -       -       y       -       3       smtp
  -o smtp_data_done_timeout=1200
  -o smtp_send_xforward_command=yes
  -o disable_dns_lookups=yes
  -o max_use=20
127.0.0.1:10025 inet    n       -       y       -       -       smtpd
  -o content_filter=
  -o local_recipient_maps=
```

```
  -o relay_recipient_maps=
  -o smtpd_restriction_classes=
  -o smtpd_delay_reject=no
  -o smtpd_client_restrictions=permit_mynetworks,reject
  -o smtpd_helo_restrictions=
  -o smtpd_sender_restrictions=
  -o smtpd_recipient_restrictions=permit_mynetworks,reject
  -o smtpd_data_restrictions=reject_unauth_pipelining
  -o smtpd_end_of_data_restrictions=
  -o mynetworks=127.0.0.0/8
  -o smtpd_error_sleep_time=0
  -o smtpd_soft_error_limit=1001
  -o smtpd_hard_error_limit=1000
  -o smtpd_client_connection_count_limit=0
  -o smtpd_client_connection_rate_limit=0
  -o receive_override_options=no_header_body_checks,no_unknown_recipient_checks,no_

# Integration with Dovecot - hand mail over to it for local delivery, and
# run the process under the vmail user and mail group.
dovecot      unix   -        n      n       -       -   pipe
  flags=DRhu user=vmail:mail argv=/usr/lib/dovecot/dovecot-lda -d $(recipient)

# Integration with the SPF check package.
policy-spf  unix  -       n       n       -       -       spawn
     user=nobody argv=/usr/bin/policyd-spf
```

Note that Amavis is restricted to three processes, which should be fine for most casual to moderate use. The processes are memory-heavy, so start low and add more only if you need to due to volume of mail - see the notes in this guide for pointers on how to do that.

Restart Everything, and Test the Server

Restart all the necessary processes to pick up configuration changes:

```
service postfix restart
service spamassassin restart
service clamav-daemon restart
service amavis restart
service dovecot restart
```

Now start testing! Keep an eye on /var/log/mail.log for error messages and while you log in to POP and IMAP, send mail to an account created on the server, and send mail from the server. Don't forget to open up the firewall to allow global access to the relevant ports before doing this. If you find issues, then Google is your friend when it comes to searching on specific error messages in order to identify the cause of any specific problem.

AWS Mail Restrictions and Reverse DNS Lookup

Once configured, with IP address set and DNS records set up, you'll need to have a reverse DNS lookup put in place for your server and the AWS outgoing mail restrictions lifted. You do that through the standard customer service form. This doesn't take long, and it can actually happen earlier in the process if necessary, prior to the server completion.

Install and Set up Monit for Monitoring

Monit is a very useful monitoring tool that helps rescue your server from failed processes. Install it through apt-get:

```
apt-get install --assume-yes monit
```

The following are a set of fairly trivial instructions that set monit to watch over the important server processes, but without issuing notifications or doing much more than restarting on failure. Note that the Amavis configuration specifies a fairly infrequent check, as it is possible to get into a situation with Amavis where it refuses connections because you're sending mail too rapidly and it hits the server's maximum number of concurrent connections per process (which is set at a low 128 for Ubuntu). Having Monit then restart Amavis at that point just makes things worse, boosting load and slowing things down. Mail will be queued and reattempted for any period while Amavis is truly down and waiting on Monit to restart it.

Create the following files in the Monit configuration directory.

/etc/monit/conf.d/amavis

```
check process amavisd with pidfile /var/run/amavis/amavisd.pid
  every 5 cycles
  group mail
  start program = "/usr/sbin/service amavis start"
  stop  program = "/usr/sbin/service amavis stop"
  if failed port 10024 protocol smtp then restart
  if 5 restarts within 25 cycles then timeout
```

/etc/monit/conf.d/apache2

```
check process apache2 with pidfile /var/run/apache2/apache2.pid
  group www
  start program = "/usr/sbin/service apache2 start"
  stop program = "/usr/sbin/service apache2 stop"
  if failed host localhost port 80 protocol http
    with timeout 10 seconds
    then restart
  if failed host localhost port 443 type tcpssl protocol http
    with timeout 10 seconds
    then restart
  if 5 restarts within 5 cycles then timeout
```

/etc/monit/conf.d/dovecot

```
check process dovecot with pidfile /var/run/dovecot/master.pid
  group mail
  start program = "/usr/sbin/service dovecot start"
  stop program = "/usr/sbin/service dovecot stop"
  group mail
  # We'd like to use this line, but see:
  # http://serverfault.com/questions/610976/monit-failing-to-connect-to-dovecot-over
  #if failed port 993 type tcpssl sslauto protocol imap for 5 cycles then restart
  if failed port 993 for 5 cycles then restart
  if 5 restarts within 25 cycles then timeout
```

/etc/monit/conf.d/mysql

```
check process mysqld with pidfile /var/run/mysqld/mysqld.pid
  group database
  start program = "/usr/sbin/service mysql start"
  stop program = "/usr/sbin/service mysql stop"
  if failed host localhost port 3306 protocol mysql then restart
  if 5 restarts within 5 cycles then timeout
```

/etc/monit/conf.d/postfix

```
check process postfix with pidfile /var/spool/postfix/pid/master.pid
  group mail
  start program = "/usr/sbin/service postfix start"
  stop  program = "/usr/sbin/service postfix stop"
  if failed port 25 protocol smtp then restart
```

```
6    if 5 restarts within 5 cycles then timeout
```

/etc/monit/conf.d/spamassassin

```
1  check process spamassassin with pidfile /var/run/spamd.pid
2    group mail
3    start program = "/usr/sbin/service spamassassin start"
4    stop  program = "/usr/sbin/service spamassassin stop"
5    if 5 restarts within 5 cycles then timeout
```

/etc/monit/conf.d/sshd

```
1  check process sshd with pidfile /var/run/sshd.pid
2    start program "/usr/sbin/service ssh start"
3    stop program "/usr/sbin/service ssh stop"
4    if failed host 127.0.0.1 port 22 protocol ssh then restart
5     if 5 restarts within 5 cycles then timeout
```

Now enable the local Monit HTTP interface by editing `/etc/monit/monitrc` to uncomment the following lines. This enables use of commands such as `monit status`. Run `monit -h` to see an overview of the available options.

```
 1  ## Monit has an embedded HTTP interface which can be used to view status of
 2  ## services monitored and manage services from a web interface. The HTTP
 3  ## interface is also required if you want to issue Monit commands from the
 4  ## command line, such as 'monit status' or 'monit restart service' The reason
 5  ## for this is that the Monit client uses the HTTP interface to send these
 6  ## commands to a running Monit daemon. See the Monit Wiki if you want to
 7  ## enable SSL for the HTTP interface.
 8  #
 9  set httpd port 2812 and
10      use address localhost  # only accept connection from localhost
11      allow localhost        # allow localhost to connect to the server and
12  #     allow admin:monit      # require user 'admin' with password 'monit'
```

Then restart Monit to pick up the new orders:

```
1  service monit restart
```

Monit offers options for notifications, a web console, restarting on high load, logging activity, and many other amenities, so you may want to add more to this very basic configuration when you become more familiar with the application. One important item to note is that once attempted restarts have timed out, Monit will no longer monitor that service, even after both service and Monit have restarted, until told to. Check to see which services are being monitored by running:

```
monit status
```

Then monitor a service with:

```
# monit monitor <name>, e.g.:
monit monitor mysqld
```

Install Roundcube for Webmail

Roundcube is a straightforward PHP webmail package: if all you need is simply to send and receive mail via a web interface then this is for you. There are other, more complex, extensible, and full-featured options out there but you pay the price for that in the time taken to install and configure the package. Roundcube is a much less onerous experience, but unfortunately the installation instructions you'll find online on how to install Roundcube are, shall we say, somewhat confused. They will largely lead you down the wrong path if working from a package install on Ubuntu. Here instead is the quick and easy way to manage things.

Start by installing the necessary packages. The plugin packages aren't essential, but it doesn't hurt to look them over to see what is available:

```
apt-get install --assume-yes \
  roundcube \
  roundcube-plugins \
  php7.2-mail \
  php-mime-type \
  php-mail-mime \
  php7.2-intl \
  php7.2-zip
```

In the package installation process you will be asked whether the installer should configure the database. Answer `Yes`, then choose `mysql` as the database type. You'll be asked for the MySQL root user password, so enter it. Then you will be asked to enter and confirm a password for a new `roundcube` database user that will be created for you. The same comments on passwords apply here as for those you created earlier for the `root` and `mail` user.

```
pear install Net_IDNA2-0.2.0 Mail_mimeDecode-1.5.6
```

Restart Apache to pick up changes:

```
service apache2 restart
```

Edit the following lines in /etc/roundcube/config.inc.php to tell Roundcube that the mail server applications are running on the same machine as it is:

```
// The mail host chosen to perform the log-in.
// Leave blank to show a textbox at login, give a list of hosts
// to display a pulldown menu or set one host as string.
// To use SSL/TLS connection, enter hostname with prefix ssl:// or tls://
// Supported replacement variables:
// %n - hostname ($_SERVER['SERVER_NAME'])
// %t - hostname without the first part
// %d - domain (http hostname $_SERVER['HTTP_HOST'] without the first part)
// %s - domain name after the '@' from e-mail address provided at login screen
// For example %n = mail.domain.tld, %t = domain.tld
// WARNING: After hostname change update of mail_host column in users table is
//          required to match old user data records with the new host.
$config['default_host'] = 'localhost';

// SMTP server host (for sending mails).
// To use SSL/TLS connection, enter hostname with prefix ssl:// or tls://
// If left blank, the PHP mail() function is used
// Supported replacement variables:
// %h - user's IMAP hostname
// %n - hostname ($_SERVER['SERVER_NAME'])
// %t - hostname without the first part
// %d - domain (http hostname $_SERVER['HTTP_HOST'] without the first part)
// %z - IMAP domain (IMAP hostname without the first part)
// For example %n = mail.domain.tld, %t = domain.tld
$config['smtp_server'] = 'localhost';
```

As /etc/roundcube/config.inc.php overrides /etc/roundcube/defaults.inc.php, and contains only a small number of the overall properties, you will have to add these lines to /etc/roundcube/config.inc.php in order to tell Roundcube to (a) redirect non-secure HTTP connections to HTTPS and (b) use the database for caching:

```
// enforce connections over https
// with this option enabled, all non-secure connections will be redirected.
```

```
// set the port for the ssl connection as value of this option if it differs from th
$config['force_https'] = true;

// Type of IMAP indexes cache. Supported values: 'db', 'apc' and 'memcache'.
$config['imap_cache'] = 'db';

// Backend to use for session storage. Can either be 'db' (default) or 'memcache'
$config['session_storage'] = 'db';
```

Ensure that you update the following configuration option in /etc/roundcube/config.inc.php to a unique value for this installation:

```
// this key is used to encrypt the users imap password which is stored
// in the session record (and the client cookie if remember password is enabled).
// please provide a string of exactly 24 chars.
// YOUR KEY MUST BE DIFFERENT THAN THE SAMPLE VALUE FOR SECURITY REASONS
$config['des_key'] = 'enter a unique value here';
```

At this point Roundcube is now installed and minimally configured, but it isn't accessible from the server webroot. The Roundcube webroot containing PHP files and various symlinks is sitting in /var/lib/roundcube, and the next step is to make that available to visitors. This is easily accomplished by creating a symlink in the webroot:

```
ln -s /var/lib/roundcube /var/www/html/roundcube
```

Now redirect the default landing page for visitors to Roundcube, which first requires moving the default index page out of the way:

```
mv /var/www/html/index.html /var/www/html/index.bak.html
```

Then expand /var/www/html/.htaccess to include a rule to redirect just the landing page to Roundcube. Being this selective leaves the open the option of adding other files and subdirectories under /var/www/html for whatever you might want to use them for, and preserves access to Postfix Admin.

```
RewriteEngine On

# Redirect all HTTP traffic to HTTPS.
RewriteCond %{HTTPS} !=on
RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
```

```
6
7  # Send / to /roundcube.
8  RewriteRule ^/?$ /roundcube [L]
```

You can now test Roundcube by visiting http://mail.example.com and logging in as a user.

Add Two Factor Authentication to Roundcube

Two factor authentication (2FA) is increasingly a good idea in this age of online attacks and data breaches. Adding it is entirely optional, but note that there is a decent 2FA plugin for Roundcube by the name of 2Steps Verification. The documentation in that repository covers installation and setup quite well, and it isn't hard to do at all, so it won't be repeated here.

Options to Help Ensure Deliverability

If you want a quiet life free from worries about whether your mail is going to be delivered to its destination, then you should take the small amount of additional time to set up SPF and DKIM for your mail server. These modifications go a long way towards ensuring that you won't have issues with triggering false positives from spam filters. It is becoming ever harder these days to ensure that your mail reaches its recipients. Set up a mail server in the cloud that happens to use an IP address that was at some time in the past used by spammers, or engage in a serious discussion of subjects that see a lot of spam, and you can wave goodbye to the certainty of delivery of your mail to addresses at Gmail or other similar email providers. These entities maintain their own arcane anti-spam infrastructures, inscrutable and distinct from the open world of IP address blacklists. Resolving issues or even simply attracting the attention of anyone who might help is essentially impossible if you don't happen to run a large company.

Thus everyone who builds their own mail server should set up the two simple frameworks that email providers pay attention to when it comes to the decision tree for their anti-spam automation: Sender Policy Framework (SPF) and DomainKeys Identified Mail (DKIM).

Set Up Sender Policy Framework (SPF)

SPF requires only that you add a TXT record to your DNS zone for the domain. How that happens depends on the tools provided by your domain registrar, or the tools you set up yourself should you manage your own nameservers. If using a registrar's web interface to make DNS changes, you may or may not have the option to enter a subdomain for the record. If you do, then leave that field blank.

This generic SPF TXT record authorizes mail originating from mail servers for your domain that are identified by MX records and all other servers associated with your domain that have A records:

```
"v=spf1 a mx -all"
```

Note that the double quotes are a necessary part of the SPF TXT record. Much more complicated records than this are possible, as outlined in the SPF documentation. You can see some of these more complex examples in the wild by using the dig command. e.g.:

```
dig google.com txt
```

Server Setup For DomainKeys Identified Mail (DKIM)

Setting up DKIM is a little more involved than SPF. First install the necessary packages:

```
apt-get install --assume-yes opendkim opendkim-tools
```

Add or edit the following values in /etc/opendkim.conf:

```
Domain    example.com
KeyFile    /etc/postfix/dkim.key
Selector    dkim
```

```
Socket    inet:8891@localhost
```

Add the following to /etc/default/opendkim:

```
SOCKET=inet:8891@localhost
```

Append a suitable DKIM configuration to /etc/postfix/main.cf:

```
# ------------------------------------
# DKIM
# ------------------------------------
milter_default_action = accept
```

```
milter_protocol = 6
smtpd_milters = inet:localhost:8891
non_smtpd_milters = inet:localhost:8891
```

Now you can generate a private key for signing outgoing mail. Note that in the following command, dkim is the value given to Selector in /etc/opendkim.conf. This can be any simple string, provided you are consistent about replacing dkim with your desired value everywhere in this recipe. Run the following command to generate the key and associated materials in the form of two files, dkim.private and dkim.txt. The former is the RSA private key, while the latter contains the entry the you will have to place into your DNS records.

```
op/endkim-genkey -t -s dkim -d example.com
```

Move the key into place and grant suitable permissions, but don't forget to take a copy and keep that copy backed up somewhere safe:

```
mv dkim.private /etc/postfix/dkim.key
chmod 660 /etc/postfix/dkim.key
chown root:opendkim /etc/postfix/dkim.key
```

You'll need to restart Postfix and OpenDKIM services to pick up the configuration changes so that outgoing mail is signed using DKIM:

```
service opendkim start
service postfix restart
```

DNS Setup For DomainKeys Identified Mail (DKIM)

Next up is the DNS record setup. How you do this is again completely dependent on how you manage DNS or how it is managed for you - everyone's tools are different. Note that some registrars do not let you create raw TXT records with specific subdomains, which will prevent you from creating DKIM TXT records. If this is the case, then you will have to transfer your domain to a real registrar that lets you play with all the toys.

The file dkim.txt contains the following content, the full TXT record that must be created. It has the subdomain dkim._domainkey and a long set of encoded content as the value. Again dkim is the value given to Selector in /etc/opendkim.conf.

```
dkim._domainkey IN  TXT ( "v=DKIM1; k=rsa; t=y; "
"p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC9rulKo58JIb5h+3MMEnYhlnbuVgRoA4w68R/X7qA2Lf
```

When adding the DNS record you should omit the `k=rsa; t=y;` portion of the value. The first item refers to the key format and that defaults to RSA. They second denotes that this is a test entry, and should not be included. Thus the value to add looks like this:

```
"v=DKIM1; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC9rulKo58JIb5h+3MMEnYhlnbuVgRoA4w68
```

It is helpful to view examples in the wild for the purposes of comparison, so that you can see how you should enter the value into your records. You can use the DKIM key checker or other tools such as dig. Note that the `dkim._domainkey` is the subdomain in the following command:

```
dig dkim._domainkey.twitter.com txt
```

Monitor the OpenDKIM Service

Add a configuration file `/etc/monit/conf.d/opendkim`:

```
check process opendkim with pidfile /var/run/opendkim/opendkim.pid
  group mail
  start program = "/etc/init.d/opendkim start"
  stop  program = "/etc/init.d/opendkim stop"
  if 5 restarts within 5 cycles then timeout
```

Then restart Monit:

```
service monit restart
```

Sharing a DKIM Key for Multiple Domains

If you are serving multiple domains from the same mail server, then the contents of `/etc/opendkim.conf` should be:

```
Domain    *
```

```
KeyFile    /etc/postfix/dkim.key
Selector    dkim
SOCKET    inet:8891@localhost
```

This will work unless you are running a mailing list or similar, in which outgoing mail has From headers containing addresses in domains other than the list domain. In that case you might look at a helpful ServerFault question and answer for more complex configurations that can support that scenario.

Testing the SPF and DKIM Configuration

The only certain test is to send mail from the server and inspect it. A decent testing service is Mail Tester, which will report on the validity of the SPF and DKIM additions to mail headers. Remember to give the DNS changes a chance to propagate before using it.

Notes on Postgrey

You'll find that Postgrey has its own idiosyncratic notion of what default configuration should be and how it should work. There are whitelist configuration files for clients and recipients in /etc/postgrey/whitelist_clients and /etc/postgrey/whitelist_recipients respectively, but these are not actually used by default. If you want to use them, you must either copy them into /etc/postfix as follows:

```
cp /etc/postgrey/whitelist_clients /etc/postfix/postgrey_whitelist_clients
cp /etc/postgrey/whitelist_recipients /etc/postfix/postgrey_whitelist_recipients
```

Or, alternatively, edit /etc/default/postgrey:

```
# postgrey startup options, created for Debian

# you may want to set
#   --delay=N   how long to greylist, seconds (default: 300)
#   --max-age=N delete old entries after N days (default: 35)
# see also the postgrey(8) manpage

POSTGREY_OPTS="--inet=10023"
POSTGREY_OPTS="$POSTGREY_OPTS --whitelist-clients=/etc/postgrey/whitelist_clients"
POSTGREY_OPTS="$POSTGREY_OPTS --whitelist-recipients=/etc/postgrey/whitelist_recipie

# the --greylist-text commandline argument can not be easily passed through
# POSTGREY_OPTS when it contains spaces.  So, insert your text here:
```

```
#POSTGREY_TEXT="Your customized rejection message here"
```

Notes on Serving Multiple Domains

You can create multiple domains in Postfix Admin if so desired, under Domain List -> New Domain. Additional domains added in Postfix Admin can be aliased to existing domains under Virtual List -> Add Alias Domain, such that address@example1.com is always forwarded to address@example2.com, or they can stand as distinct domains with their own mailboxes, aliases, and so forth.

Depending on your use case, you might also want to adjust some of the .htaccess rules to support users accessing the site at mail.example1.com, mail.example2.com, and so forth - such as expanding the redirect to SSL to recognize all of the domains used.

Roundcube should work for multiple domains, but requires you to create multiple configuration files. Fortunately the instructions are clear.

The tricky part of the setup for multiple domains relates to SSL certificates, IP addresses, and reverse DNS lookup for outgoing mail. Postfix requires one IP address per certificate, as does Dovecot. So a server could be provisioned with multiple IP addresses and certificates, which would be fine for incoming mail. Alternatively, use a single IP address and multi-domain UCC certificate. The current spam ecosystem wants the reverse DNS lookup to be consistent, however, and in a single server with multiple IP addresses, the outgoing mail will all appear to be coming from the single primary IP address regardless of domain it is sent from. This is a problem. The simplest solution to work around this issue for outgoing mail is to assign the same mail.example.com MX record for all of the domains served by the one server; there are some examples that explain this setup if you look around online.

Notes on Setting Up as a Backup MX

A backup mail server will receive mail when the primary is offline, and then forward it to the primary when it is available. For a personal mail server this is probably unnecessary, but for a business it is useful. If you want to set up the server you are building as one of the backup mail servers, then the following additions to this recipe are required. These are taken from a short guide that you might find helpful.

1) Firstly create /etc/postfix/mysql_relay_domains_maps.cf containing the following:

```
user = mail
password = mailpassword
```

```
hosts = 127.0.0.1
dbname = mail
query = SELECT domain FROM domain WHERE domain='%s' AND backupmx = '1' AND active = '
```

2) Then add the following to /etc/postfix/main.cf:

```
# This is a backup MX server, and this line tells Postfix
# where to send the mail.
relay_domains = proxy:mysql:/etc/postfix/mysql_relay_domains_maps.cf
```

3) In the domain configuration in Postfix Admin, check the "Mail server is backup MX" option.

4) To enable delivery to the backup, the MX DNS records must include an entry at a lower priority than the primary mail server. E.g.:

```
name                    priority  ip-address
mail.example.com          10      172.10.10.10
mail-backup.example.com   20      172.10.10.11
```

5) Lastly, you will need to set up a script or process of change replication to ensure that the database entries for users and domains are the same in primary and backup mail servers, aside from the backup MX flag.

Notes on Managing Quotas

If you've been following carefully, you will note that nothing has been said so far on the matter of user disk space quotas. It was not an important goal for the work that prompted the creation of these instructions. As things stand the necessary fields for quota managment exist in the MySQL database but are not used, as (a) the quota module isn't enabled by default in Dovecot, and (b) Postfix Admin is not set to use quotas by default.

So if you want to enable disk quotas, you should first of all alter the Postfix Admin quota configuration in /var/www/html/postfixadmin/config.inc.php by adding the following to /var/www/html/postfixadmin/config.local.php:

```
// Quota
// When you want to enforce quota for your mailbox users set this to 'YES'.
$CONF['quota'] = 'YES';
// You can either use '1024000' or '1048576'
$CONF['quota_multiplier'] = '1024000';
```

```
// Optional:
// Show used quotas from Dovecot dictionary backend in virtual
// mailbox listing.
// See: DOCUMENTATION/DOVECOT.txt
//      http://wiki2.dovecot.org/Quota/Dict
//
$CONF['used_quotas'] = 'YES';

// if you use dovecot >= 1.2, set this to yes.
// Note about dovecot config: table "quota" is for 1.0 & 1.1,
// table "quota2" is for dovecot 1.2 and newer
$CONF['new_quota_table'] = 'YES';
```

Next, you will want to enable and configure the quota and imap_quota modules in Dovecot. The former manages quotas while the latter enables reporting on quotas via IMAP. You will want to look through the following documentation for instructions on how to do this:

- Quota (Dovecot 2.*)
- Quota Configuration (Dovecot 2.*)

These configuration changes will be made in `10-mail.conf` and `90-quota.conf` in the `/etc/dovecot/conf.d` directory.

Bypassing Spam and Virus Checks for Local Mail

If you're in the business of sending out newsletters or frequent updates from local software where you completely control the content in those emails, then you probably don't want to run spam and virus checks for those items. It's a pointless use of server processing cycles, and a newsletter run can hammer the server if you are making it process the full range of checks on each and every one of those mails.

To have amavisd-new skip the checks for mail originating from a known set of IP addresses (e.g. locally, from a web application on another server, etc), edit `/etc/amavis/conf.d/50-user` to add these lines:

```
# Replace 111.111.111.111/32 with your desired list of client IP address
# ranges which will bypass checks.
@mynetworks = qw( 127.0.0.0/8 [::1] 111.111.111.111/32 );

# Rules for clients defined in @mynetworks
$policy_bank{'MYNETS'} = {
  bypass_spam_checks_maps   => [1],  # don't spam-check internal mail
  bypass_banned_checks_maps => [1],  # don't banned-check internal mail
```

```
  bypass_header_checks_maps => [1],  # don't header-check internal mail
};
```

Replace 111.111.111.111/32 with whatever set of IP address ranges you want to bypass amavisd-new checks. All mail arriving from those sources will fall into MYNETS for amavisd-new and therefore bypass checking. If bypassing by IP address doesn't fit your needs, you can find ways to skip checks for some users, destinations, or sources in a helpful, if dated guide to amavisd-new and Postfix integration.

Some Final Notes on Security

You'll note that there are a fair number of configuration files that contain database passwords for the mail and webmail data in this server, and that includes PHP files sitting in the webroot. This is not really the dominant security concern: the mail users are virtual and only the server administrator should be logging in as a system user. On AWS the default setup is for SSH login to use keys rather than passwords, and only the ubuntu user has a key setup to allow login. You can also easily lock down the SSH port to selected IP addresses via the security group applied to the server. Further, you can set .htaccess directives to ensure that no web visitor can directly view configuration files - and thus they are only used as includes, which covers the rare case where some error causes PHP files to be served by Apache as plain text. MySQL access is from localhost only, in any case.

All in all the lowest bar from a traditional security perspective is probably that the mail server built here runs a couple of complicated PHP web applications with database access. A serious breach there would involve a way to upload and execute an arbitrary PHP script or shell command with the www-data user's permissions, or various other XSS attacks allowing for session hijacking of administrators. Either way, or just by getting into the databases, compromise of the webroot is compromise of all of the important functions of the server. Major PHP webmail applications have exhibited multiple serious vulnerabilities in past years, but at some point you have to pick your software. On the whole given the choice I'd rather go with the output of established development communities whose members have a demonstrated track record of vulnerabilities found and fixed, and where there are a large number of eyes directed at the codebase.

These are all good reasons for setting up your webmail on a different server from the one running Postfix and Dovecot - something to bear in mind.

Of course being on AWS - or indeed any sort of easily available hosting in the US wherein the server is not in your front room - means that the US government has free access to your data any time they particularly feel up to the task, and you may never know a copy was taken. One of the welcome forthcoming evolutions in virtual hosting services will be some form of turn-key encrypted server operations such that you can have the convenience of an AWS-style service but without the transparency it affords the present day panopticon-in-the-making.

Further, it is apparently the case that all email traffic between mail servers is being recorded by various governmental agencies. Unfortunately the present state of SMTP in the wild is that many or most mail servers do not implement the ability to pass emails over an encrypted connection: so while it's easy to setup and enforce encryption for POP, IMAP, and webmail connections between users and the mail server, email traffic between mail servers is often plain text. Forcing your server to only use encrypted connections with other servers will mean that a large fraction of your email traffic in both directions will be rejected. Thus the configuration provided for Postfix in this post is for optional encryption - emails sent and received will be encrypted if the mail server on the other end of the connection can support it.