

Replace your Jira/Confluence mail queue with Postfix -- part 2

Following on from [part 1](#), this article discusses how to configure Postfix to act as Jira's mail transfer agent. We discuss choice of mail relay, provide relayed and non-relayed Postfix sample configurations, and discuss how Postfix can be configured on reroute email on sandbox/staging servers.

- [Introduction](#)
- [Choose your upstream relay](#)
 - [1\) Relay through a company-wide relay](#)
 - [2\) Relaying through regular email service providers](#)
 - [3\) Relaying through specialist bulk email service providers](#)
 - [4\) Sending email directly from your Jira server](#)
- [Postfix configuration](#)
 - [Postfix configuration – sending directly \(no relay\)](#)
 - [Postfix configuration – relaying](#)
 - [Postfix configuration example – relaying through GSuite](#)
 - [Relaying through GSuite - GSuite configuration](#)
 - [Relaying through GSuite - Postfix configuration](#)
 - [GSuite relaying – a live example](#)
- [Sending test email via Postfix](#)
- [Monitoring your Postfix mail queue](#)
- [Re-routing emails to a catch-all address on sandbox](#)
- [Conclusion](#)

Introduction

As suggested in [part 1](#), your best plan is to get emails **out of Jira and into a competent mail transfer agent (MTA)** as early as possible. We will use Postfix, installed on localhost port 25, and then configure Jira to send email to it:

Outgoing mail

ENABLED

Disable outgoing mail

SMTP Mail Server

The table below shows the SMTP mail server currently configured for Jira.

Name	Details	Operations
Default SMTP Server	From: jira@ Prefix: [JIRA] Host: localhost SMTP Port: 25	Edit Delete Send a test email

Jira will hand off to Postfix fast (because it's localhost) and with almost guaranteed success. Postfix is left with the hard task of actually delivering the emails.

Choose your upstream relay

If you have installed things on Debian/Ubuntu before, you know that the `apt-get install` process asks you some configuration questions. So before diving in, we need to decide how Postfix is going to deliver emails. Typical options are:

1. relay email through a **company-wide** mail relay server
2. relay email via your **regular email service provider**, like Gmail, GSuite, Outlook or Fastmail.
3. relay email via a **specialist bulk-sender** service provider, like Sendgrid, Mailgun or AWS SES.
4. no relay: send email **directly** from your Jira server to end users

1) Relay through a company-wide relay

Larger, more established companies often have existing SMTP relays through which all outgoing SMTP must flow, enforced by a firewall.

2) Relaying through regular email service providers

Smaller companies, you may choose to relay through your regular email provider, i.e. `smtp.yourhostingprovider.com`.

If so, please review this list of [email sending limits per provider](#) to see if you will hit sending limits. Jira sends *lots* of email.

For GSuite in particular, the relay limit is 10,000 per day (documented [here](#)).

3) Relaying through specialist bulk email service providers

This is probably the way to go for larger (50+ user) installations if you don't want to have problems or deal with mail infrastructure. I have personally used sendgrid and had no problems.

4) Sending email directly from your Jira server

This wasn't even an option when Jira sent emails directly, but..

Rather than send email *via* someone, why not just have *our* mail server send to *their* mail server directly? You know, like email was designed to work - decentralized, peer-to-peer, not beholden on giant gatekeeping corporations?

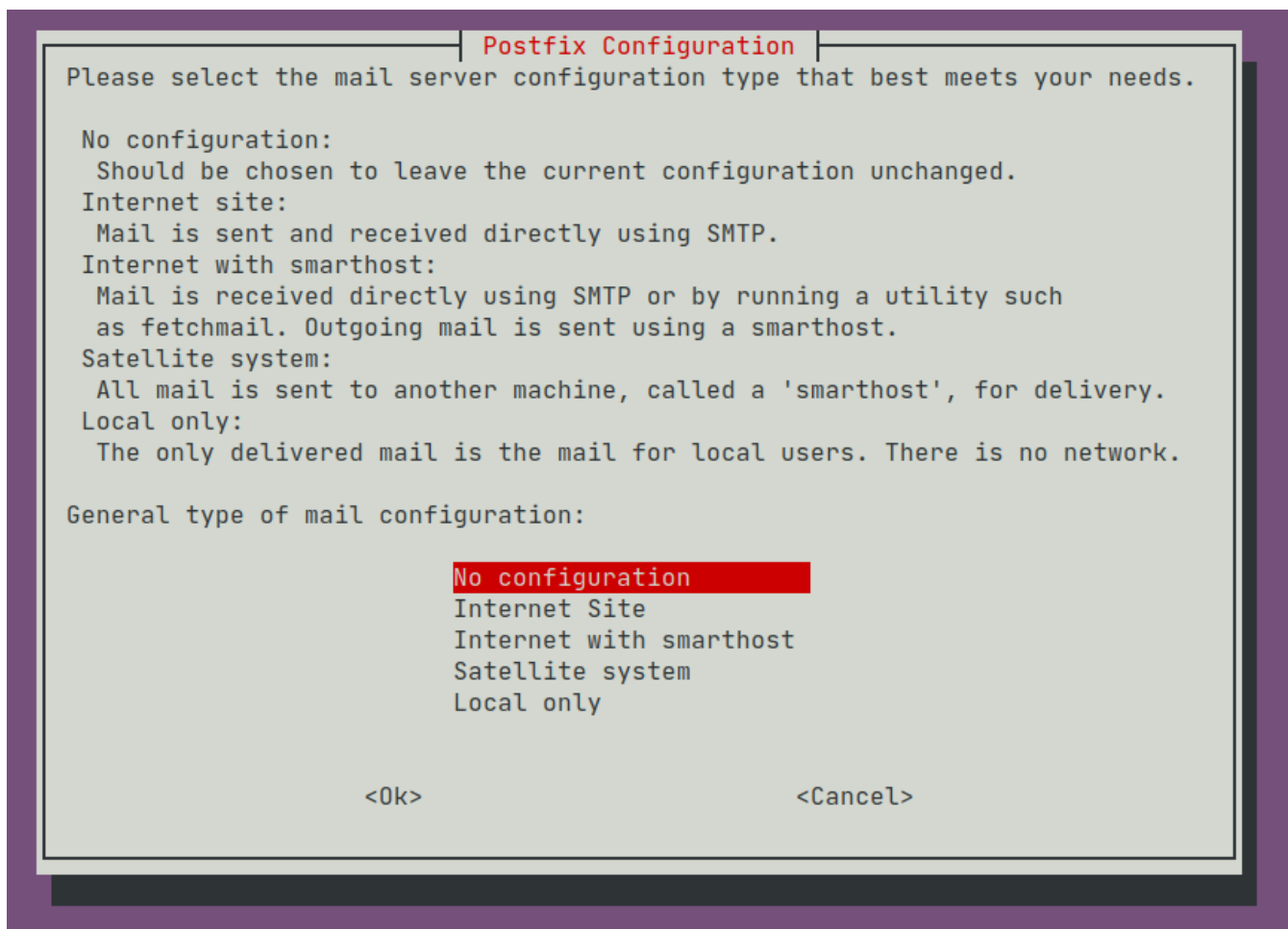
Well, you can do that, and it is in fact my favourite option. Email is the last great decentralized protocol that still more or less works. The problem is spam, and more specifically the countermeasures that large email providers take to avoid spam. Your little mail server, sending legitimate Jira emails, may not be trusted when it connects directly to a recipient's mail server, and your Jira notifications may not be delivered. You can increase the deliverability odds by following best practices, but you may be unlucky, e.g. if you unknowingly share a IP netblock with a known spammer.

Despite legitimate worries over deliverability, sending directly is my favourite option because:

- it avoids the arbitrary sending limits imposed by most email service providers.
- It *generally* works:
 - This is anecdotal, but I administer a few smaller instances, and one very high volume 2000-user Jira, sending about 17k emails per day to many academic domains. We very rarely (say, once a year) have problems in the form of bounced mails, or complaints from users.
 - My theory is that the sheer volume of Jira email works in its favour. Spam filters track reputation: a torrent of similar emails, once marked as legitimate, will continue to be let through. Spam filters won't even bother to track reputation of a sporadic email sender. Your recipient could mark an email as legitimate and it won't matter: by the time you send another email, the spam filter has forgotten.
- when it works, it is zero cost
 - ...except from the initial setup cost of learning and configuring TLS, SPF, DKIM, DMARC and Postfix. But that's fun! Read [Small Mailserver Best Current Practices](#) to see what
- when it *doesn't* work, it's not catastrophic. A few users might find Jira notifications in their spam folder, and will have to click 'Not spam' until the filter learns.
- As noted earlier, email is the last great decentralized part of the internet, and we (technical folks) should do our part to keep it open. Every independent SMTP server promotes diversity and helps to [keep the bastards honest](#).

Postfix configuration

Run `apt-get install postfix` to install Postfix. You will be prompted to enter values in a series of 'debconf' setup screens. The first screen confronts us with the relay-or-send-directly choice discussed above:



The options mean:

Option	Incoming Mail	Outgoing Mail
Internet Site	Accepts outside connections, delivering to local mailboxes	Sent directly
Internet with smarthost	Accepts outside connections, delivering to local mailboxes	Relayed
Satellite system	Accepts connections from localhost, delivering to local mailboxes	Relayed
Local only	Accepts connections from localhost, delivered to local mailboxes	No sending or relaying

Pick **Satellite system** if you wish to relay through a mail service provider, or **Internet Site** if you wish to send email directly. Then accept the defaults.

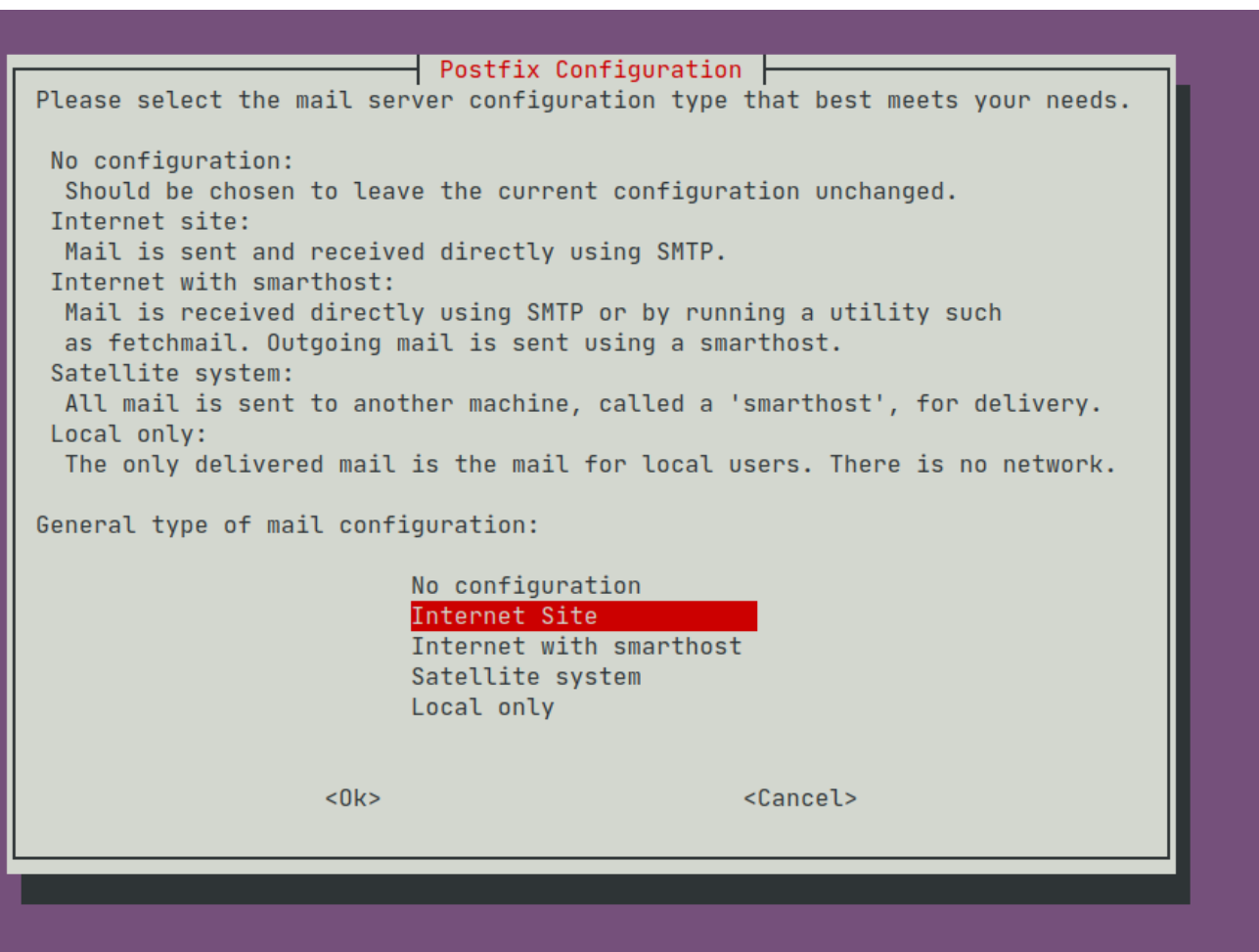


The debconf setup sets Debian-specific paths for parameters like `smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem` and `smtp_tls_CApath=/etc/ssl/certs`, which are otherwise hard to get right.

Now proceed to the relaying or send-directly section below, in which you will properly customize `/etc/postfix/main.cf`.

Postfix configuration – sending directly (no relay)

To have Postfix send emails **directly**, choose 'Internet Site':



Because we will only accept incoming SMTP connections from Jira and Confluence locally, limit Postfix to listen on localhost:

```
# postfix inet_interfaces=loopback-only
```



Postfix tips

With Postfix configuration, the `postconf` command is worth getting to know:

- **postconf <param>** shows the current value of parameter <param>. Add -x to recursively resolve variables.
- **postconf -d <param>** shows the *default* value of <param>
- **postconf -n** shows all overridden parameters in [main.cf](#)
- **postconf foo=bar** sets parameter foo to bar in [main.cf](#)
- **postconf -# foo** comments out parameter foo

Other useful commands:

- **postfix flush** resends the deferred queue
- **postsuper -d ALL deferred** deletes the deferred queue

When reading the docs, keep very clear in your head whether you're currently debugging Postfix *receiving* email (`smtpd_*` parameters) or *sending* email (`smtp_*` parameters).

This is not a Postfix tutorial, so I'll just paste a full `/etc/postfix/main.cf` from a live server, suitably hyperlinked (via the `mantools/postlink` perl script in the Postfix source):

```

# Debian specific: Specifying a file name will cause the first
# line of that file to be used as the name. The Debian default
# is /etc/mailname.
#myorigin = /etc/mailname

smtpd_banner = $myhostname ESMTP $mail_name (Ubuntu)
biff = no

# appending .domain is the MUA's job.
append_dot_mydomain = no

# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h

readme_directory = no

# See http://www.postfix.org/COMPATIBILITY\_README.html -- default to 2 on
# fresh installs.
compatibility_level = 2

# Incoming SMTP (smtpd): TLS parameters
smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
smtpd_tls_security_level=may
smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated defer_unauth_destination

# $myhostname must have a PTR record.
myhostname = jiraconfserver.example.com
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
mydestination = $myhostname, example-jiraconf, localhost.localdomain, localhost
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = loopback-only
inet_protocols = all

# Outgoing SMTP
# Opportunistic TLS authentication. http://www.postfix.org/postconf.5.html#smtp\_tls\_security\_level
smtp_tls_security_level = may
# On the advice of https://www.linode.com/docs/email/postfix/configure-postfix-to-send-mail-using-gmail-and-google-apps-on-debian-or-ubuntu/
# Without this we get 'Untrusted TLS connection established to smtp.gmail.com'
smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt
# Reuse TLS connections to hopefully delay GMail breaking with "4.7.0 Too many login attempts,
# please try again later". http://www.postfix.org/TLS\_README.html#client\_tls\_reuse
smtp_tls_connection_reuse = yes
#tlsproxy_client_loglevel = 2
smtp_tls_session_cache_database = btree:/var/lib/postfix/smtp_scache
smtp_tls_session_cache_timeout = 3600s
smtp_tls_loglevel = 1

# Re-route emails to a catch-all address for testing or on sandbox
#virtual_alias_maps = regexp:/etc/postfix/virtual_alias

#disable_vrfy_command = yes
# Increase the maximum email size from 10Mb to 50Mb. https://www.redradishtech.com/pages/viewpage.action?pageId=19103782
message_size_limit = 50240000

```

Some notes:

- The first part, up to and including `inet_protocols = all` is boilerplate 'Satellite system' content.
 - We're only listening on loopback, and our only SMTP client will be Jira/Confluence, so the `smtpd_tls` lines are not strictly necessary, unless you tick the 'TLS' box in Jira's Outgoing Mail config.
- `myhostname` is the EHLO address the server identifies itself as to others, and as such:
 - must be a fully qualified domain name ('jiraconfserver.example.com')
 - ..with a PTR record for ipv4 and (assuming `inet_protocols = all`) ipv6. Here is a handy function for testing:

```

validate_spf()
{
    a=${1}. # Add trailing dot
    ip=$(dig +short A $a)
    ptr=$(dig +short -x $ip)
    [[ $a == $ptr ]] && echo "ipv4 correct" || echo "$1 ipv4 resolved to $ip, whose PTR is $ptr, not $a"

    ip6=$(dig +short AAAA $1)
    ptr=$(dig +short -x $ip6)
    [[ $a == $ptr ]] && echo "ipv6 correct" || echo "$1 ipv6 resolved to $ip, whose PTR is $ptr, not $a"
}

validate_spf jiraconfserver.example.com

```

- The address (or IP) must be listed in your SPF record, e.g:

```

dig TXT example.com | grep spf
example.com. 284      IN      TXT      "v=spf1 include:_spf.google.com a:jiraconfserver.example.com ~all"

```

- [smtp_tls_connection_reuse](#) tells Postfix to cache and reuse its outgoing TLS connection to SMTP servers. If you enable this, also ensure `master.cf` has the line:

```

tlsproxy  unix  -      -      y      -      0      tlsproxy

```

The hope is that when sending dozens of emails to one provider (like `smtp.gmail.com`), we could make them all over just one TLS connection, and hopefully avoid "Too many login attempts" errors. With this flag you'll see `/var/log/mail.log` lines like:

```

Jul 22 21:45:19 example-jiraconf postfix/smtp[650976]: Trusted TLS connection reused to smtp.gmail.com [2404:6800:4003:c03::6d]:587: TLSv1.3 with cipher TLS_AES_256_GCM_SHA384 (256/256 bits) key-exchange X25519 server-signature ECDSA (P-256)

```

you'll see a new `conn_use=` field indicating how much connection reuse you're getting:

```

Jul 22 21:45:21 example-jiraconf postfix/smtp[650805]: 25839197990: to=<joe.bloggs@example.com>, relay=smtp.gmail.com[74.125.130.109]:587, conn_use=5, delay=27, delays=0/22/0.26/5.3, dsn=2.0.0, status=sent (250 2.0.0 OK 1595418321 c139sm23292217pfb.65 - gsmt)

```

The most reuse I've seen is `conn_use=9`. I have also not verified whether TLS connection reuse actually affects GSuite's "Too many login attempts" at all.

- `message_size_limit` is, as the comment states, increases the size limit for giant Jira attachments [from JETI](#).
- I have not done DKIM signing, and probably should.
- See [below](#) for more on re-routing mail with `virtual_alias_maps`

With this setup I can successfully deliver emails to a variety of domains, e.g:

```

Jul 29 17:21:36 example-jiraconf postfix/qmgr[420220]: 4E843197944: from=<root@jiraconfserver.example.com>,
size=469, nrcpt=1 (queue active)
root@example-jiraconf:/etc/postfix# Jul 29 17:21:37 example-jiraconf postfix/tlsproxy[420459]: CONNECT to [2404:
6800:4003:c04::1a]:25
Jul 29 17:21:37 example-jiraconf postfix/tlsproxy[420459]: Trusted TLS connection established to aspmx.l.google.
com[2404:6800:4003:c04::1a]:25: TLSv1.3 with cipher TLS_AES_256_GCM_SHA384 (256/256 bits) key-exchange X25519
server-signature ECDSA (P-256) server-digest SHA256
Jul 29 17:21:37 example-jiraconf postfix/smtp[420416]: Trusted TLS connection established to aspmx.l.google.com
[2404:6800:4003:c04::1a]:25: TLSv1.3 with cipher TLS_AES_256_GCM_SHA384 (256/256 bits) key-exchange X25519
server-signature ECDSA (P-256) server-digest SHA256
Jul 29 17:21:38 example-jiraconf postfix/smtp[420416]: 4E843197944: to=<jeff@redradishtech.com>, relay=aspmx.l.
google.com[2404:6800:4003:c04::1a]:25, delay=58, delays=56/0.02/1.3/0.77, dsn=2.0.0, status=sent (250 2.0.0 OK
1596007298 mw16si68538pjb.75 - gsmtpt)
Jul 29 17:21:38 example-jiraconf postfix/tlsproxy[420459]: DISCONNECT [2404:6800:4003:c04::1a]:25
Jul 29 17:21:38 example-jiraconf postfix/qmgr[420220]: 4E843197944: removed

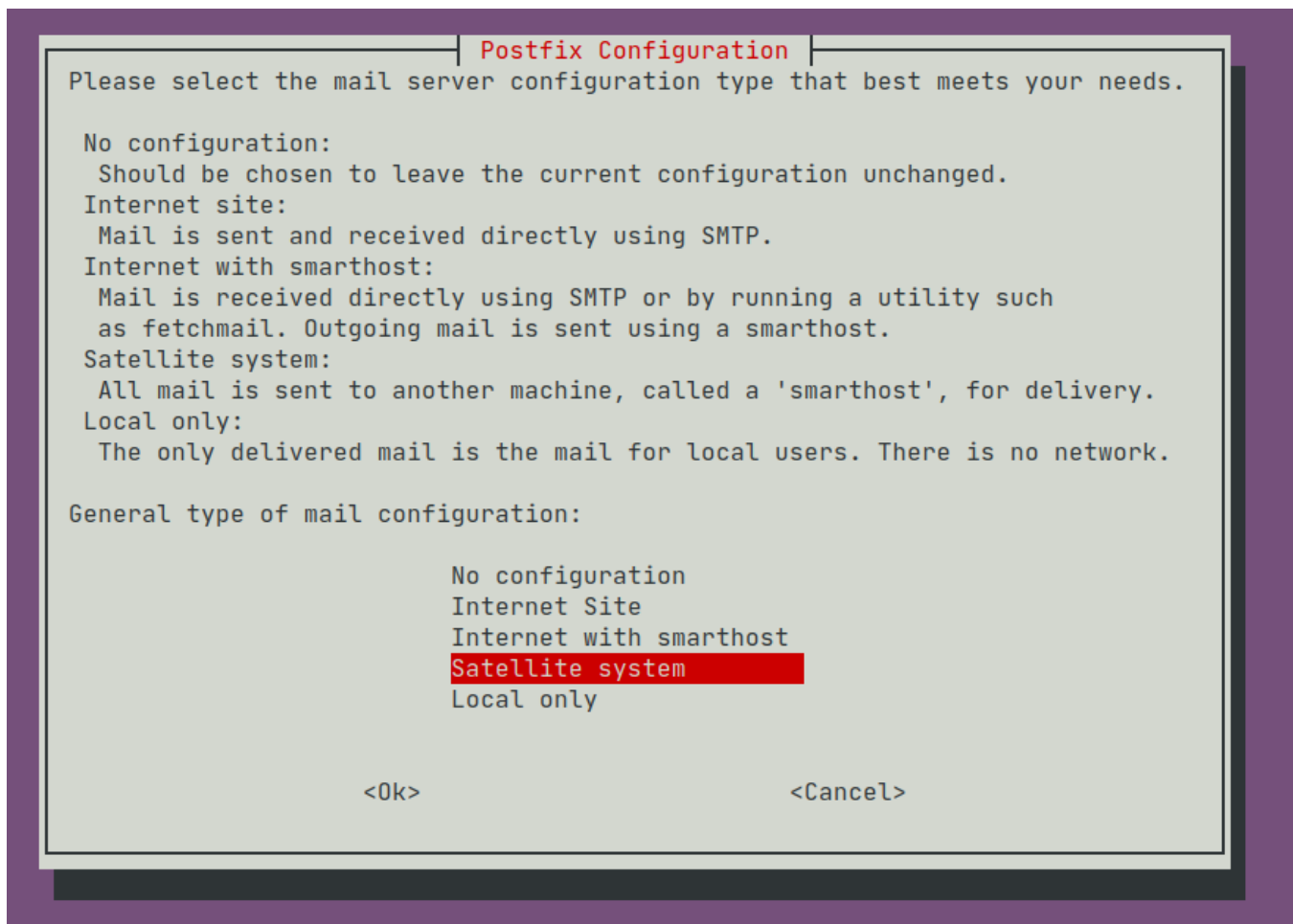
```

Things to notice in the log above:

- TLS is working -- 'Trusted TLS connection' means we validated GMail's cert, thanks to smtp_tls_CAfile
- (the tlsproxy logs and The relay=aspmx.l.google.com reflects the first MX entry for domain redradishtech.com.

Postfix configuration – relaying

To configure Postfix for relaying, start with a 'Satellite system':



then choose the defaults until the setup process ends.

Now follow the Postfix docs on [enabling SASL authentication in the Postfix SMTP/LTMP client](#). Essentially we need to specify a relayhost and enable SASL authentication:

```

/etc/postfix/main.cf:
smtp_sasl_auth_enable = yes
smtp_tls_security_level = encrypt
smtp_sasl_tls_security_options = noanonymous
relayhost = [mail.isp.example]
# Alternative form:
# relayhost = [mail.isp.example]:submission
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd

```

Postfix configuration example – relaying through GSuite

If you intend to relay through GSuite, read this important page: [SMTP relay: Route outgoing non-Gmail messages through Google](#).



GSuite legacy free edition users

If you are using a GSuite 'legacy free edition', as I am with redradishtech.com, the documentation starts with a warning:

To use this feature with your [redradishtech.com](#) account, you need to [upgrade to G Suite Basic](#).

Don't worry, you can still relay through GSuite, but:

- you will have to use `smtp.gmail.com:587` as the relay address, rather than `smtp-relay.gmail.com:25`. If you use `smtp-relay.gmail.com:25` your attempts at relaying will bounce:

```

Jul 29 15:04:28 radish-linode postfix/smtp[4260]: 17CE4400BF: to=<www-data@li993-48.members.
linode.com>, relay=smtp-relay.gmail.com[64.233.191.28]:25, delay=0.44, delays=0/0/0.37/0.06,
dsn=5.7.0, status=bounced (host smtp-relay.gmail.com[64.233.191.28] said: 550-5.7.0 Mail relay
denied [45.33.43.48]. Invalid credentials for relay for one 550-5.7.0 of the domains in: (as
obtained from HELO and MAIL FROM). 550-5.7.0 Email is being sent from a domain or IP address
which isn't registered 550-5.7.0 in your G Suite account. Please login to your G Suite account
and 550-5.7.0 verify that your sending device IP address has been registered within 550-5.7.0
the G Suite SMTP Relay Settings. For more information, please visit 550 5.7.0 https://support.
google.com/a/answer/6140680#maildenied c2sm1608320oq.8 - gsmt (in reply to RCPT TO command))

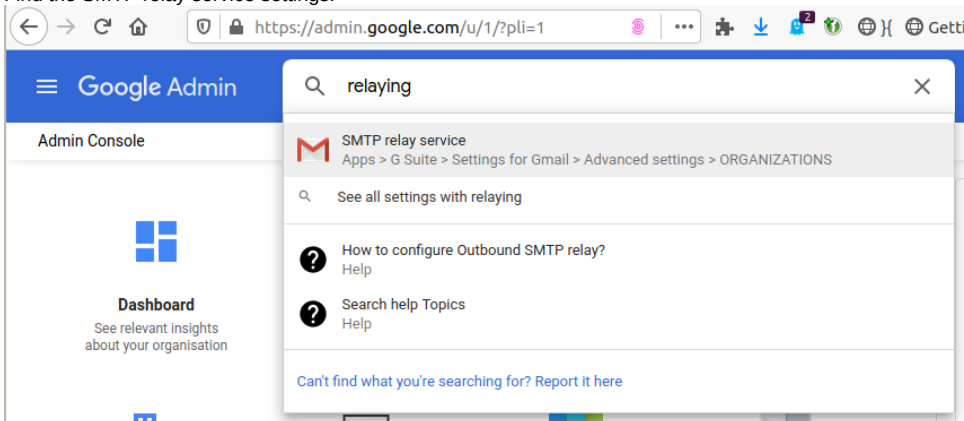
```

- The daily limit will be 2000 emails per day, rather than 10,000 for official relayers.

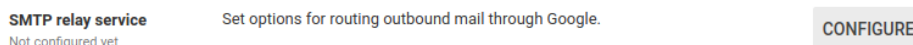
Relaying through GSuite - GSuite configuration

Go to <https://admin.google.com> logged in as a GSuite administrator.

1. Find the SMTP relay service settings:



2. 'Configure' the SMTP relay service if not yet configured:



Add a description, 'require SMTP Authentication' and 'Require TLS encryption'. I did not lock down IPs, but you might like to:

Add setting

SMTP relay service

Help

Allow relaying from the Jira/Confluence linode server.

1. Allowed senders

Only registered Apps users in my domains

2. Authentication

☐ Only accept mail from the specified IP addresses

☒ Require SMTP Authentication

3. Encryption

☒ Require TLS encryption

CANCEL ADD SETTING

Click 'Add setting', then 'Save'.

3. Allow 'Less secure apps':

The screenshot shows the Google Admin console interface. At the top, the browser address bar displays 'admin.google.com/ac/security/lsa'. The Google Admin header includes a search bar and navigation icons. The breadcrumb trail indicates 'Security > Less secure apps'. On the left, the 'Security Settings' sidebar lists 'Users', 'Groups', and 'Organisational units'. The main content area, titled 'Less secure apps', shows settings for 'Pangolin Associates Pty Ltd'. It includes a description: 'Control user access to apps that use less secure sign-in technology and make accounts more vulnerable. [Learn more](#)'. Two radio button options are present: 'Disable access to less secure apps (recommended)' and 'Allow users to manage their access to less secure apps', with the second option selected. An information icon and note state: 'Changes may take up to 24 hours to propagate to all users. Prior changes can be seen in [Audit log](#)'. At the bottom right are 'CANCEL' and 'SAVE' buttons.

Finally, log out and back into GSuite as the 'jira' account to perform these steps:

1. go to <https://myaccount.google.com/security>
2. turn on 2FA
3. generate an 'App password', saving it for use below.

Relaying through GSuite - Postfix configuration

Now for the Postfix configuration. Google helpfully provides some [documentation](#) on how to configure Postfix:

Follow the instructions below to set up the SMTP relay service for Postfix. These instructions are designed to work with a majority of deployments. There is no need to increase the timeouts for Postfix servers. The default timeout settings are appropriate.

To set up a smart host for Postfix:

1. Add the following line to your configuration file (example path `/etc/postfix/main.cf`):
`relayhost = smtp-relay.gmail.com:25`
2. Restart Postfix by running the following command:
`# sudo postfix reload`
3. Send a test message to confirm that your outbound mail is flowing.

Determine whether either of the following is true:

- You click the **Any address** option in the **Allowed senders** setting and you send messages from a domain you do not own, such as yahoo.com.
- You send messages without a "From" address, such as non-delivery reports or vacation "out of office" notifications.

If either is true, configure your mail server to either ensure that the server is using SMTP AUTH to authenticate as a registered G Suite user or to present one of your domain names in the HELO or EHLO command. See the instructions [here](#) .

Important: G Suite Support does not provide technical support for configuring on-premise mail servers or third-party products. In the event of a Postfix issue, you should consult your Postfix administrator. These instructions are designed to work with the most common Postfix scenarios. Any changes to your Postfix configuration should be made at the discretion of your Postfix administrator.

As a reminder, we are following the guide [enabling SASL authentication in the Postfix SMTP/LTMP client](#), which basically wants you to add these parameters:

```
/etc/postfix/main.cf:  
smtp_sasl_auth_enable = yes  
smtp_tls_security_level = encrypt  
smtp_sasl_tls_security_options = noanonymous  
relayhost = [mail.isp.example]  
# Alternative form:  
# relayhost = [mail.isp.example]:587  
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
```

GMail always displays email as coming 'from' the user that authenticated, rather than what is in the From: header. In my case I have a Jira and Confluence (and an invoice system) all sending mail from one server via Postfix. Thus I actually want authentication to vary depending on sender:

- If Jira sends the email (From: jira@example.com) then I want to authenticate as jira@example.com .
- If Confluence sends the email (From: confluence@example.com) then I want to authenticate as confluence@example.com .
- If my invoicing system ([invoiceinja](#)) sends the email (From: billing@example.com), then I want to authenticate as billing@example.com .
- For all other cases, I want to authenticate as www@example.com

This can be done by following the [Sender-Dependent SASL Authentication instructions](#), which adds a parameter to make authentication dependent on the From: header:

```
/etc/postfix/main.cf:  
smtp_sender_dependent_authentication = yes
```

sasl_passwd then has credentials mapped per sender, plus a default fallback mapping for the relay:

```
/etc/postfix/sasl_passwd:
# Per-sender authentication
user1example.com          username1:password1
user2example.net          username2:password2
# Login information for the default relayhost.
[mail.isp.example]        username:password
# Alternative form:
# [mail.isp.example]:587 username:password
```

GSuite relaying – a live example

Here is a sample `/etc/postfix/main.cf` that does all of the above:



```

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

# Debian specific: Specifying a file name will cause the first
# line of that file to be used as the name. The Debian default
# is /etc/mailname.
#myorigin = /etc/mailname

smtpd_banner = $myhostname ESMTP $mail_name (Ubuntu)
biff = no

# appending .domain is the MUA's job.
append_dot_mydomain = no

# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h

readme_directory = no

# See http://www.postfix.org/COMPATIBILITY\_README.html -- default to 2 on
# fresh installs.
compatibility_level = 2

# Incoming SMTP (smtpd): TLS parameters
smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
smtpd_tls_security_level=may
smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated defer_unauth_destination

# Outgoing SMTP

# Location of CA certificates
# On the advice of https://www.linode.com/docs/email/postfix/configure-postfix-to-send-mail-using-gmail-and-google-apps-on-debian-or-ubuntu/
# Without a valid CAfile we would get 'Untrusted TLS connection established to smtp.gmail.com'
smtp_tls_CAfile=/etc/ssl/certs/ca-certificates.crt

# Require TLS when relaying to Google
smtp_tls_security_level=encrypt
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache
# Enable SASL auth when connecting to other SMTP servers. http://www.postfix.org/SASL\_README.html#client\_sasl
smtp_sasl_auth_enable=yes
smtp_sasl_security_options = noanonymous, noplaintext
# Override smtp_sasl_tls_security_options to remove 'noplaintext' so we can use PLAIN auth for
# gmail. https://toroid.org/postfix-smtp-relay-gmail
smtp_sasl_tls_security_options = noanonymous
# SASL login credentials
smtp_sasl_password_maps=hash:/etc/postfix/sasl_passwd
# Allow auth to outgoing SMTP to vary based on sender 'From' address.
# This ensures credentials matching the 'From' header are used.
smtp_sender_dependent_authentication = yes

smtp_tls_loglevel = 1

myhostname = example
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = /etc/mailname
mydestination = example, example, localhost.lxd, localhost
relayhost = [smtp.gmail.com]:587
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = loopback-only
inet_protocols = ipv4
mynetworks = 127.0.0.1/32 10.20.46.187/32 [::1]/128 [fd42:f93a:8612:865e:216:3eff:fe35:f5ef]/128
[fe80::216:3eff:fe35:f5ef]/128
default_transport = smtp
relay_transport = smtp

```

/etc/postfix/sasl_passwd contains:

```
# Per-sender authentication; see also /etc/postfix/sender_relay.

# Optional Postfix SMTP client lookup tables with one username:password entry per sender, remote
hostname or next-hop domain. Per-sender lookup is done only when sender-dependent authentication
is enabled. If no username:password entry is found, then the Postfix SMTP client will not attempt
to authenticate to the remote host.
# The Postfix SMTP client opens the lookup table before going to chroot jail, so you can leave
the password file in /etc/postfix.
# Specify zero or more "type:name" lookup tables, separated by whitespace or comma. Tables will
be searched in the specified order until a match is found.

jira@example.com jira@example.com:<APP PASSWORD>
confluence@example.com confluence@example.com:<APP PASSWORD>
billing@example.com billing@example.com:<APP PASSWORD>
[smtp.gmail.com]:587 www@example.com:<APP PASSWORD>
```

You will need to 'compile' `sasl_passwd` into a `sasl_passwd.db` file with `postmap`. Paste the following into `/etc/postfix/Makefile`:

```
all:    $(wildcard *.db) sasl_passwd.db

%.db: %
    postmap $<
```

(Note: there must be a tab character before `postmap` not spaces)

Then run `make` in `/etc/postfix` to generate `sasl_passwd.db`

Other notes:

- While it won't hurt to get the HELO/EHLO domain (`$myhostname`) a valid FQDN with a PTR record, I don't think it matters when relaying (see Google's comment in the screenshot earlier).
- We now *enforce* TLS (vs. just opportunistically using it) since we're only connecting to `smtp.gmail.com`, which we know supports it.
- I have regretfully left `smtp_tls_connection_reuse` off, because connection caching is not supported when combined with sender-dependent authentication.



Encountering GSuite's sender limits - a cautionary tale

An anecdote: I recently configured a 25-user Jira instance to relay through GSuite. Having only a 'legacy free edition', I relayed through smtp.gmail.com rather than smtp-relay.gmail.com:

Name	Details
GMail	From: jira@ Prefix: [JIRA] Host: smtp.gmail.com SMTP Port: 465 Username: jira@

Little did I know this meant the [2000-email sender limit](#) applied, instead of the paid edition's 10,000 email limit.

The projects used a fairly typical notification scheme, notifying the reporter, assignee, watchers and a certain user:

Issue Updated (System)

- Current Assignee ([Delete](#))
- Reporter ([Delete](#))
- All Watchers ([Delete](#))
- Single User () ([Delete](#))

Ad

Only about 3 users were typically active. Surely GMail could handle this?

No, actually. All it took was one user bulk-updating a few hundred issues, forgetting to turn off the notifications. 500 issues, with about 4 notifications per issue (per above scheme) equals 2000 outgoing emails.

GMail worked for a bit, then starts returning:

```
454 4.7.0 Too many login attempts, please try again later. g30sm3251575pfq.189 - gsmt
```

and later:

```
421 4.3.0 Temporary System Problem. Try again later (10). u26sm21635356pgo.71 - gsmt
```

Jira cares not, and keeps trying to flush the queue. Eventually GMail turns nasty:

```
550 5.4.5 Daily user sending quota exceeded. c23sm23541133pfo.32 - gsmt
```

And finally smtp.gmail.com stops responding at all:

```
connect to smtp.gmail.com[172.253.118.108]:465: Connection refused
```

The server was then unable to send mail for 12-24 hours.

At this point I decided to RTFM and discovered most of the information above.

Sending test email via Postfix

To test your Postfix configuration, it's handy to be able to generate test emails with a particular From: header, to a particular user.

The `mail` command is good for this. Specifically, the form:

`mail -r <from_address> -s <subject> <to_address> <<< mail_body`

You will need to `apt-get install mailutils` first.

My Postfix debugging loop is basically:

1. edit `/etc/postfix/main.cf`
2. `postfix reload` because who knows if what you edited is one of the few magically-reloading settings.
3. `tail -f /var/log/mail.log` in a terminal
4. Send a test email: `mail -r jira@redradishtech.com -s "Testing from jira@" jeff@redradishtech.com <<< Hello`
 - a. ..or `postfix flush` to resend the mail queue.
5. Watch the tailed logs for (if successful) `status=sent deliveries`

If you need more verbose logs (SMTP protocol logs), add `-v` to the `smtpd` or `smtp` command in `/etc/postfix/master.cf`, and reload postfix. Sometimes `trivial-rewrite` needs a `-v` too.

Monitoring your Postfix mail queue

At this point you should have a functional Postfix. You should now configure your monitoring software to ensure that it stays functional.

I use [this Nagios script](#) to detect mail queue anomalies like bounces or a backed-up defer queue. It is the best I could find for free, but still not hugely flexible or great. If you know of a better monitoring solution please let me know in the comments.



Perhaps it is just me, but a few times now I've had GSuite 'app passwords' just mysteriously stop working. I have to regenerate them. To my knowledge GSuite app passwords **don't expire**, so I don't know what is going on. Without monitoring this failure would be quite damaging, e.g. on servers that sends important emails like invoices.

Re-routing emails to a catch-all address on sandbox

In serious installations, one generally has a 'sandbox' (or 'staging') Jira for experiments and testing. The sandbox Jira data is periodically refreshed from production.

One requirement of sandbox Jira is that *it must not be allowed to email real users*. People get really confused if they receive notifications "from Jira" that were actually just experiments on sandbox.

Say we had our SMTP details configured directly in Jira:

Name	Details
GMail	From: jira@ Prefix: [JIRA] Host: smtp.gmail.com SMTP Port: 465 Username: jira@

This would be a problem: the credentials come with the data refreshes. What is to stop our sandbox Jira sending emails through [smtp.gmail.com](#) just like production?

The usual answer is: you set the `-Datlassian.mail.senddisabled=true` flag to prevent emails being sent, and/or by blocking outgoing connects to 25/465/587 at the firewall (since plugins might send email directly).

But now things are different: Jira no longer embeds the SMTP host details:

Default SMTP Server

From: jira@
Prefix: [JIRA]
Host: localhost
SMTP Port: 25

So on sandbox we can configure Postfix to **re-route all outgoing emails to a local mailbox** (regardless of true destination).

Then even if you forget `-Datlassian.mail.senddisabled=true` you don't have to worry. In fact, you might deliberately leave it unset so you can inspect JIRA's mail delivery, without the risk of spamming real users.

To do this:

1. Create a system account whose mailbox will accumulate catch-all emails:

```
useradd --no-create-home --comment "Postfix may re-route to this user mailbox" mailcatchall
```

(or skip this step and just use `root` below)

2. Add a `virtual_alias_maps` parameter to your `/etc/postfix/main.cf`:

```
# Map *@monitoring.redradishtech.com to jeff@redradishtech.com
virtual_alias_maps = regexp:/etc/postfix/virtual_alias
```

and postfix reload

3. Create `/etc/postfix/virtual_alias` containing:

```
# Reroute all email, sending it to root's local mailbox
#/.*/ mailcatchall
```

4. Then make `virtual_alias.db` (assuming you installed the Makefile shown earlier)

You should then see outgoing email be rerouted:

```
Jul 30 12:06:57 mycontainer postfix/qmgr[694107]: D94DDA70F: from=<billing@redradishtech.com>, size=357,
nrcpt=1 (queue active)
root@mycontainer:/etc/postfix# Jul 30 12:06:57 mycontainer postfix/local[694832]: D94DDA70F:
to=<mailcatchall@mycontainer.lxd>, orig_to=<jeff@redradishtech.com>, relay=local, delay=0.03, delays=0.02/0/0
/0, dsn=2.0.0, status=sent (delivered to mailbox)
Jul 30 12:06:57 mycontainer postfix/qmgr[694107]: D94DDA70F: removed
```

Conclusion

Use the best tool for the job! For delivering email, that's Postfix.

Assuming you started off just letting Jira relay your mail, let's review the benefits:

- **An audit log.** you now have a record of what email was sent from Jira, in `/var/log/mail.log`
- **Retry on failure.** Postfix will keep trying, with exponential back-off, if anything goes wrong with delivery, vs. Jira's behaviour of just giving up.
- **Persistent defer queue.** If anything goes wrong, your mail backlog is persisted to disk, not simply lost if you reboot or restart Jira.
- **Improve Jira performance and stability.** Jira can use the memory and CPU consumed by its mail/error queue for other things.
- **Outgoing mail health monitoring.** You can now monitor outgoing email health (e.g. the backlog queue size) using standard tools.
- You have at least the option to **send outgoing email directly**, rather than relay, thus bypassing the whole tedious question of sending limits. If you choose to relay you can reduce the likelihood of hitting send limits with TLS connection reuse.
- **Re-route outgoing emails on sandbox.** You can now debug notification schemes by seeing what Jira actually sends (on sandbox).
- **Learn things!** Postfix is a gem of free, high-quality software. Time spent understanding Postfix and email delivery generally is unlikely to be wasted.

Feel free to add a comment below or contact me at jeff@redradishtech.com with any feedback or questions.

