

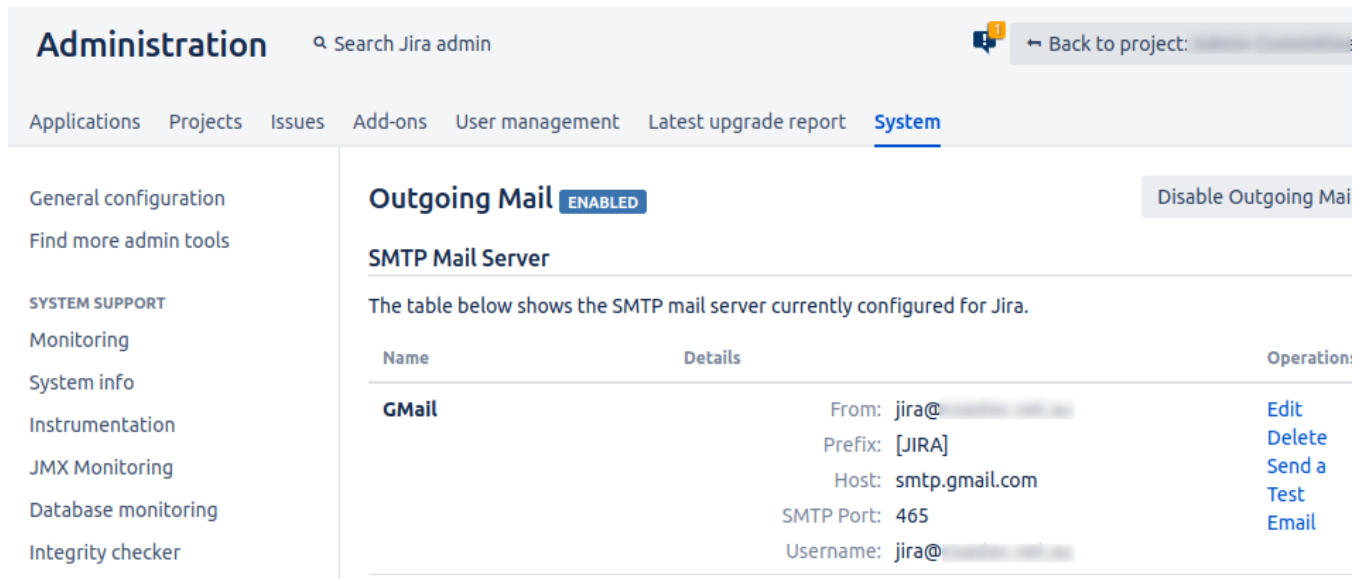
# Replace your Jira/Confluence mail queue with Postfix -- part 1

In this article we discuss the deficiencies of Jira's outgoing email handling, and how you can avoid problems by configure Jira to delegate mail sending to a local Postfix server ([part 2](#)). While we use Jira for concreteness, everything said here applies equally to Confluence.

- [An overview of Jira's outgoing mail system](#)
- [Jira's problematic mail queue](#)
  - [Problem 1 – when Jira is restarted, the error queue is lost](#)
  - [Problem 2 – no audit log](#)
  - [Problem 3 – temporary problems become permanent](#)
  - [Problem 4 – general usability](#)
- [Deprecated OAuth Authentication](#)
- [The problems of storing mail credentials directly in Jira](#)
  - [Passwords in Jira is bad for security](#)
  - [Passwords in Jira can lead to accidental sandbox server spam](#)
  - [What about JNDI?](#)
  - [What is a JNDI Location?](#)
- [Conclusion](#)

## An overview of Jira's outgoing mail system

Per the [documentation](#), Jira can be configured to send notification emails:



The screenshot shows the Jira Administration interface. The 'System' tab is selected in the top navigation bar. On the left sidebar, under 'SYSTEM SUPPORT', the 'Monitoring' section is expanded, showing 'System info' as the selected item. The main content area is titled 'Outgoing Mail' with an 'ENABLED' status button and a 'Disable Outgoing Mail' button. Below this, the 'SMTP Mail Server' section is shown, stating 'The table below shows the SMTP mail server currently configured for Jira.' A table follows with columns for 'Name', 'Details', and 'Operation'. The table contains one entry for 'GMail' with details for From, Prefix, Host, SMTP Port, and Username, and a column of operations: Edit, Delete, Send a Test Email.

Name	Details	Operation:
GMail	From: jira@ Prefix: [JIRA] Host: smtp.gmail.com SMTP Port: 465 Username: jira@	Edit Delete Send a Test Email

In this example, you'll see my 'Host' is `smtp.gmail.com`, to which I authenticate as a dedicated `jira` account. This means all outgoing email will be relayed through gmail.

This article will try to convince you not to do that. Instead you should install Postfix on your Jira server, and use Postfix as Jira's relay:

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

By all means, configure Postfix to relay through gmail.com if you like – just don't let Jira itself do the delivery.

There are three broad reasons for this:

- Postfix does a much, much better job of managing a mail queue, recovering from errors, and logging what happened.
- Postfix allows neat functionality like redirecting outgoing email to a local 'blackhole' account on sandbox servers.
- Storing SMTP credentials in Postfix is more secure than having them in the Jira database, and thus every database backup.

The benefits of Postfix will become more apparent in [part 2](#) – for now we just focus on the problems with Jira's status quo.

## Jira's problematic mail queue

Notifications get batched in a queue and sent once a minute. Normally Jira's outgoing email works fine. But every now and then something breaks, and Jira stops sending notifications.

You might first notice this as general brightening of the office mood, and increase in productivity. But eventually some manager will complain that they're not getting updates, and you are tasked to investigate.

You will probably find Jira's mail queue is an unhealthy red, and the Error queue is full:



## Mail

---

Outgoing mail information is sent to 'atlassian-jira-outgoing-mail.log'.

The Outgoing mail log is currently turned **ON**.

- **Disable the Outgoing mail log.**

The Outgoing mail debug log is currently turned **OFF**.

- **Enable debugging.**

If you thought this meant outgoing emails are logged, you'd be wrong: only errors are logged. See

[JRASERVER-45162](#) - Getting issue details...

STATUS

### Problem 3 – temporary problems become permanent

- After 10 delivery attempts the mail is sent to the error queue **and is never tried again** unless an administrator intervenes, or the system runs out of memory (the error queue is an in-memory data structure). There is no exponential backoff.
- Jira's mail queue is known to just stop flushing if an OutOfMemoryError occurs. If you find the mail queue full *but not the error queue*, check your catalina.out log file for these.

### Problem 4 – general usability

Other problems become apparent over time. On

[JRASERVER-7873](#) - Getting issue details...

STATUS

we have the original lead developers of

Confluence and Jira calling the mail error queue "useless":

Jira

Dashboards ▾ Projects ▾ Issues ▾ Boards ▾ Create

Search 🔍

Jira Server and Data Center

astegani's board ▾

Backlog

Active sprints

Releases

Reports

Issues

Components

Jira Server and Data Center / JRASERVER-7873

Improve the error mail queue

Q Comment Agile Board More ▾

Export ▾

Type:

Suggestion

Status:

GATHERING INTEREST  
[\(View Workflow\)](#)

Resolution:

Unresolved

Fixed in Version/s:

None

Component/s:

Email notifications

Labels:

affects-server jira support feature request sf

UIS:

2

Support reference count:

8

Feedback Policy:

We collect Jira feedback from various sources, and we evaluate what we've collected when planning our product roadmap. To understand how this piece of feedback will be reviewed, see our [Implementation of New Features Policy](#).

Details

Description

Agile

People

Dates

Agile

Assignee:

Unassigned

Reporter:

AntonA

Votes:

73 Vote for this issue

Watchers:

47 Start watching this issue

Created:

07/Sep/2005 12:58 PM

Updated:

20/Mar/2020 11:17 AM

View on Board

NOTE: This suggestion is for JIRA Server. Using JIRA Cloud? See the corresponding suggestion.

From CONF-3231:  
The mail error queue is currently useless.

- You have no indication what the error is
- All you can do is "resend all" or "delete all"

We need to work on the same principles as an MTA.

- If an error is permanent, give up and log the error somewhere.
- If an error is transient, put it on a queue to resend after a few hours.
- If a transient error keeps happening for a certain amount of time, give up and log the error somewhere.

Also useful would be:

- Keep track of users whose mail fails repeatedly, and stop sending them anything. Flag the account, and have a big note on their profile page saying "Notifications are currently disabled for this account due to repeated mail delivery failures. Please turn notifications back on on your notifications preferences page."

Issue Links

+ relates to

CONFSERVER-3231 Fix the mail error queue

GATHERING I...

JRACLOUD-7873 Improve the error mail queue

GATHERING I...

Username

Optional - if you use authenticated SMTP to send email, enter your username.

Change Password ☒

Password

Optional - as above, enter your password if you use authenticated SMTP.

That is (sadly) no longer a good assumption. For instance, GSuite will stop supporting password-based authentication from 15 Feb 2021:

Starting February 15, 2021, G Suite accounts will only allow access to apps using OAuth. Password-based access will no longer be supported. - [GSuite](#)

By then you will need to have upgraded to Jira 8.10.0, (released 23 Jun 2020- [release notes](#)), which supports incoming OAuth. If you don't plan to upgrade, switching to Postfix or another MTA with OAUTH2 support is your only option.

## The problems of storing mail credentials directly in Jira

When you configure Jira's Outgoing Mail Server, your connection details are stored in the database:

```
jira=> select id, name, mailfrom, server_type, protocol, mailusername, mailpassword from mailserver;
```

id	name	mailfrom	server_type	protocol	mailusername	mailpassword
10000	localhost	jira@example.com	smtp	smtp	jira	hunter2
10100	jira@ Mail Server		pop	imaps	jira@example.com	hunter2

(2 rows)

## Passwords in Jira is bad for security

First, the obvious problem: these credentials are in plaintext, and can be seen by:

- any plugin
- any user with ability to run code (think ScriptRunner)
- anyone who can access the database *or a database backup*. Your company probably has dozens of database backups across many servers

## Passwords in Jira can lead to accidental sandbox server spam

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.

However, when we restore our production data to sandbox, our SMTP details come with it:

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

What is to stop our sandbox Jira sending emails through [smtp.gmail.com](https://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 how much safer and cleaner is it to just not embed those details in the database in the first place.

In part 2 we will discuss a further advantage: on sandbox we can configure Postfix to 'blackhole' outgoing emails, i.e. send them to a local mailbox (regardless of true destination). This lets you inspect JIRA's mail output, without the risk of spamming real users.

## What about JNDI?

When configuring the outgoing mail server, instead of configuring details directly one has the option of entering a JNDI Location:

or

### JNDI Location

JNDI Location

The JNDI location of a `javax.mail.Session` object, which has already been set up in Jira's application server.

Test Connection

Update

Cancel

## What is a JNDI Location?

A JNDI Location is a URL-like address specifying a configuration object, typically a database (e.g. `java:comp/env/jdbc/JiraDS`) or email server (e.g. `java:comp/env/mail/GMail`) in a Java application server like Tomcat.

This requires a tiny bit of background knowledge. You may be aware that JIRA runs inside 'Tomcat'. When you start JIRA (e.g. via `bin/startup.sh` you are actually starting Tomcat, which in turn loads JIRA. The file `conf/server.xml` is Tomcat's configuration file, not JIRA's. Tomcat is primarily a HTTP-to-Java-application bridge. When a HTTP request comes in from your browser, Tomcat will:

- Parse the HTTP request
- Decide what loaded Java application should deals with this request (in our case, always JIRA)
- Call the relevant Java code (in our case, JIRA code), passing in an object (`HttpServletRequest`) representing the HTTP request, and being returned an object (`HttpServletResponse`) representing the HTTP response
- Translate JIRA's response object into actual HTTP response.

JIRA doesn't have to know or care about HTTP details, it just gets given a nice object. This is the [Inversion of Control](#) principle in software design, and Tomcat applies it to other things too:

- Tomcat can connect to **database servers**, keeping a pool of connections open for speed. Apps like JIRA are passed preconfigured JDBC database connection objects on demand.
- Tomcat can connect to **email servers** (SMTP, POP, IMAP). Apps like JIRA are passed preconfigured JavaMail email connection objects.

Since there might be multiple databases and email servers configured in Tomcat (e.g. two mail servers, `java:comp/env/mail/GMail` and `java:comp/env/mail/ExchangeEmail`), there needs to be a way for JIRA to address them, and that's what a JNDI Location is: a path into a virtual tree of configuration objects that Tomcat provides. A JNDI location lets you tell JIRA *what* to connect to without specifying *how* the connection is made.

It helps to see this in practice. Here is a Tomcat `conf/server.xml` file snippet, where JIRA is configured with access to a database on `java:comp/env/jdbc/JiraDS` and a GMail server at `java:comp/env/mail/GmailSmtpServer`:

```

<Context path="" docBase="${catalina.home}/atlassian-jira" reloadable="false" useHttpOnly="true">

<Resource name="UserTransaction" auth="Container" type="javax.transaction.UserTransaction"
    factory="org.objectweb.jotm.UserTransactionFactory" jotm.timeout="60"/>
<Manager pathname="" />

<Resource name="jdbc/JiraDS" auth="Container" type="javax.sql.DataSource"
    username="redradish_jira"
    password="redradish_jira"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost/redradish_jira"
    maxActive="20"
    validationQuery="select 1"/>

<Resource name="mail/GmailSmtpServer" auth="Container" type="javax.mail.Session"
    mail.smtp.host="smtp.gmail.com"
    mail.smtp.port="465"
    mail.smtp.auth="true"
    mail.smtp.user="jira@redradishtech.com"
    password="s3cretP@ssw0rd"
    mail.smtp.starttls.enable="true"
    mail.smtp.socketFactory.class="javax.net.ssl.SSLSocketFactory"
    />

</Context>

```

Using JNDI also avoids the two problems discussed above. With the JNDI method, your email credentials are kept a single file ( `conf/server.xml` ) on the server. not in the JIRA database. It is much easier to secure a single file than dozens of backups spread across multiple servers. Technically, you're configuring the appserver, Tomcat, with the password, not JIRA: JIRA just gets to use authenticated connections that Tomcat provides it.

With no passwords in the JIRA database, you also don't have to worry that someone will restore your data in a staging or dev JIRA, which then starts sending notification emails and filter subscriptions to people with stale data.

The downside of keeping SMTP details in `conf/server.xml` is that you must remember to transfer these details across (plus copy some jar files) every time you upgrade JIRA. Also, the initial configuration and any subsequent changes (e.g. password resets) require a JIRA restart.

In general, storing SMTP credentials in Postfix has all the advantages of JNDI, and none of the downsides.

## Conclusion

Jira's outgoing mail handling is a combination of bad design (storing SMTP credentials in Jira) and bad execution (Jira's half-baked mail queue). Read on in [part 2](#), where we discuss a better alternative.