computers are bad

You are receiving this facsimile because you signed up for fax delivery of this newsletter. To stop delivery, contact Computers Are Bad by email or fax.

https://computer.rip - me@computer.rip - fax: +1 (505) 926-5492

2021-12-26 diy mail

I have another post about half-written that I will finish up and publish soon, but in the mean time I have been thinking today about something that perennially comes up in certain orange-tinted online communities: running your own mail server.

I have some experience that might allow me to offer a somewhat nuanced opinion on the matter. Some years ago I was a primary administrator of a mailserver for a small university (~3k users), and today I operate two small mailservers, one for automated use and one that has a small number of active human users. On the other hand, while I operated a mailserver for my own personal email for years I have now been using Fastmail since around 2015 and have had no regrets.

My requirements are perhaps a little unusual, and that no doubt colors my opinion on options for email: I virtually never use webmail, instead relying on SMTP/IMAP clients (mostly Thunderbird on desktop and Bluemail on Android, although I would be hesitant to endorse either very strongly due to long-running stability and usability problems). A strong filtering capability is important to me, but I am relatively lax about spam filtering as I normally review my junk folder manually anyway. I am very intolerant of any deliverability problems as they have caused things like missed job opportunities in the past.

The software stack that I normally use for email is a rather pedestrian one that forms the core of many institutional and commercial email services: postfix and dovecot, working off of a mix of Unix users and virtual ones. Typically I have automated management through tools that output postfix map text files rather than by moving postfix maps to e.g. an SQL server, although I have worked with one or two mailservers configured that way. I have been from Squirrelmail to Roundcube to Rainloop for webmail, although as mentioned I do not really use webmail personally.

My preference is to use Dovecot via LMTP and move as much functionality as possible (auth, filtering, etc) into Dovecot rather than Postfix. I have always used SpamAssassin and fail2ban to control log noise and brute force attempts.

All of this said, one of the great frustrations to email servers, especially to the novice, is that even for popular combinations like Postfix/Dovecot there are multiple ways to architect the mail delivery, storage, and management process. For example, there are at least 4-5 distinct architectural options for configuring Postfix to make mail available to Dovecot. Different distributions may package these services pre-configured for one approach or the other, or with nothing pre-configured at all. In fact, mail servers are an area where your choice of distribution can matter a great deal: under some Linux distributions, like RHEL, simply installing a few packages will result in a mostly working mailserver configured for the most common

architecture. Under other distributions the packages will leave you with an empty or nonexistent configuration and you will have a lot of reading to do before you get to the most basic working configuration.

The need to support some of the newer anti-spam/anti-abuse technologies introduces some further complication, as you'll need to figure out a DKIM signing service and get it inserted into the mail flow at the right point. Because of the historic underlying architecture of most MTAs/MDAs, this can actually be surprisingly confusing as it's often difficult to "hook" into mail processing at a point where you can clearly differentiate email that is logically "inbound" and "outbound" [1].

Finally, as a general rule mail-related software tends to be "over-architected" (e.g. follows "the Unix philosophy" in all the worst ways and few of the good ones) and fairly old [2]. This makes a basic working body of configuration surprisingly large and complex, and the learning curve can be formidable. It took me years to feel generally conversant in the actual care and feeding of Postfix, for example, which like many of these older network services has a lot of fun details like its own service management and worker pooling system.

All of this goes to explain that configuring a mailserver has one of the steeper learning curves of common network services. Fortunately, a number of projects have appeared which aim to "auto-configure" mailservers for typical use-cases. Mail-in-a-box, iRedMail, etc. promise ten minutes to a fully working mailserver.

These projects, along with a general desire by many in the tech industry to reduce their reliance on ad-supported services of major tech companies, have resulted in a lot of ongoing discussion about the merits of running your own mail. Almost inevitably these threads turn into surprisingly argumentative back-and-forths about the merits of self-hosted mail, the maintenance load, deliverability, and so on.

Years ago, before I had any sort of coherent place to put my writing, I wrote an essay about email motivated by Ray Tomlinson's death: Obituary, for Ray Tomlinson and Email. I will not merely repeat the essay here, in large part because it's mostly philosophical in nature and I intend to stay very practical in this particular message. The gist of it, though, is that email as we now know it was one of the first real federated systems and is also, in my opinion, one of the last. The tremendous success of the SMTP ecosystem has also revealed the fundamental shortcomings of federated/distributed systems, in that the loosely federated nature of email leads to daily real-world frustrations that show close to zero sign of ever improving.

There are practical implications to these more theoretical problems, and they're the same ones that repeatedly tank efforts at decentralized communications and social media. In the case of email, they are particularly severe, as the problems emerged *after* email became widely used. Instead of killing the concept or causing a redesign to eliminate the defects of the federated design, in the case of email we just have workarounds and mitigations. These are where most of the real complexity of email lies.

Spam

The most obvious, and usually largest, problem that any decentralized communications system will run into is spam. There are various theoretical approaches to mitigating this issue, such as proof of work, but in practice real-world communications products

pretty consistently mitigate spam by requiring a proof of identity that is difficult to produce en masse.

The most common by far is a telephone number. Complaints about Telegram and Signal requiring that accounts be associated with a phone number are widespread (I am one of the people complaining!), but they often miss that this simple (if irritating to some) step is incredibly effective in reducing spam. This tends to turn into a real "I found the exception and therefore you are wrong" kind of conversation, so let me acknowledge that there are plenty of ways to come up phone numbers that will pass the typical checks used by these service. But that doesn't in any way invalidate the concept: all these methods of obtaining phone numbers are relatively expensive and volume limited, so they don't undermine the basic goal of using SMS validation of a phone number to require a high effort level to register multiple accounts. The very low volume of outright spam on both Telegram and Signal as an indication of the success of this basic strategy.

Of course requiring a validated telephone number as part of identity is a substantial compromise on privacy and effectively eliminates identity compartmentalization (the mind boggles at the popularity of Telegram with furries in consideration of this issue, as compared to common furry use patterns on services like Twitter that do facilitate compartmentalization). But there's a more significant problem: it is predicated on centralization. Sure, it's theoretically possible to implement this in a distributed fashion, but there's a few reasons that no one is going to. For properly federated services it's a non-starter, as unless you significantly compromise on the basic idea of federation you are reliant on all members of the federation to perform their own validation of users against a scarce proof of identity... but the federation members themselves are frequently crooked.

In other words, in some ways this approach *has* been applied to email as popular free email hosts like Google and Microsoft are increasingly pushing telephone validation as a requirement on accounts. But that only protects their own abuse handling resources. Email being federated means that you need to accept mail from other servers, and you don't know what their validation policy is.

This is a fundamental problem. Federated systems impose significant limits on any kind of identity or intent validation for users, and so spam mitigation needs to be done at the level of nodes or instances instead. This tends to require an ad-hoc "re-centralization" in the form of community consensus policy, blocklists of instances, etc. Modern federated systems still handle this issue fairly poorly, but email, due to its age, lacks even the beginning of a coordinated solution.

Instead, more reactive measures have had to be taken to protect the usability of email, and those are the elephant in the room in all discussions of self-hosted email. They have significant implications for self-hosted email operators.

Most spam filtering solutions rely on some degree of machine learning or dynamic tuning. Smaller email operators have an inherently harder time performing effective spam blocking because of the smaller set of email available for ongoing training. In practice this problem doesn't seem to be that big and SpamAssassin mostly performs okay without significant additional training, but the issue does exist, it's just not too severe.

Because mail servers come and go, and malicious/spam email often comes from new mail servers, major email operators depend heavily on IP reputation and tend to automatically distrust any new mail server. This leads to a few problems. First,

cheap or easy-to-access hosting services (from AWS to Uncle Ed's Discount VPS) almost always have ongoing problems with fly-by-night customers using their resources to send spam, which means that they almost always have chunks of their IP space on various blocklists. This is true from the sketchiest to the most reputable, although the problem tends to be less severe as you get towards the Oracle(R) Enterprise Cloud(TM) version of the spectrum.

These issues can make DIY email a non-starter, as if you rely on a commodity provider there's a fair chance you'll just get a "bad IP" and have a bit of an ongoing struggle to get other providers to trust anything you send. That said, it's also very possible to get recycled IPs that have no issues at all... it tends to be a gamble. Less commodity, more bespoke service providers can usually offer some better options here. In the best case, you may be able to obtain IP space that hasn't been used in a long time and so is very unlikely to be on any blocklists or reputation lists. This is ideal but doesn't happen so often in this era of IPv4 exhaustion. As the next best thing, many providers that have a higher-touch sales process (i.e. not a "cloud" provider) maintain a sense of "good quality" IPs that have a history of use only by trusted clients. If you spend enough money with them you can probably get some of these.

On the other hand, most cheap VPS and cloud providers are getting their IP space at auction, which has a substantial risk of resulting in IP space with a history of use by a large-scale organized email spam operation. If you spend much time looking at websites like LowEndBox you'll see this happening a lot.

Even if you get an IP with no reputational problems, you will still run into the worst part of this IP reputation issue: IPs with no history are themselves suspicious. Most providers have logic in place that is substantially more likely to reject or flag as spam any email coming an IP address without a history of originating reliable email. Large-scale email operations contend with this by "warming up" IPs, using them to send progressively more traffic over time in order to build up positive reputation. As an individual with a single IP you are not going to be able to do this in such a pre-planned way, but it does mean that things will get better over time.

A frustrating element of email deliverability is the inconsistency in the way that email providers handle it. It used to be that it was often possible to get feedback from email providers on your deliverability, but that information was of course extremely useful to spammers, so major providers have mostly stopped giving it out. Instead, email providers typically reject some portion of mail they don't like entirely, giving an SMTP error that almost universally gives a link to a support page or knowledgebase article that is not helpful. While these SMTP rejections are frustrating, the good news is that you actually know that delivery failed... although in some cases it will succeed on retry. The mail servers I run have been around long enough that outright SMTP rejections are unusual, but I still consistently get a seemingly random sample of emails hard rejected by Apple Mail.

What's a little more concerning is, of course, a provider's decision of whether or not to put a message into the junk folder. In a way this is worse than an outright rejection, because the recipient will probably never see the message but you don't know that. Unfortunately there aren't a lot of ways to get metrics on this.

If you self-host email, you *will* run into an elevated number of delivery problems. That is a guarantee. Fully implementing trust and authentication measures will help, but it will not eliminate the problem because providers weight their IP reputation information more than your ability to configure DKIM correctly. Whether or not it

becomes a noticeable problem for you depends on a few factors, and it's hard to say in advance without just trying it.

Federated systems like email tend to rely on a high degree of informal social infrastructure. Unfortunately, as email has become centralized into a small number of major providers, that infrastructure has mostly decayed. It was not that long ago that you could often resolve a deliverability problem with a politely worded note to postmaster @ the problematic destination server. Today, many email providers have some kind of method of contacting them, but I have never once received a response or even evidence of action due to one of these messages... both for complaints of abuse on their end and deliverability problems [3].

Ongoing maintenance

While it is fully possible to set up a mailserver and leave it for years without much of any intervention beyond automatic updates, I wouldn't recommend it. Whether you have one user or a thousand, mail service tends to benefit appreciably from active attention. Manual tuning of spam detection parameters in response to current spam trends can have a huge positive impact on your spam filtering quality. I also manually maintain allow and blocklists of domains on mailservers I run, which can also greatly improve spam results.

More importantly, though, because of the extremely high level of ambient email abuse, mailservers are uniquely subject to attack. Any mailserver will receive a big, ongoing flow of probes that range from simple open relay checks to apparently Metasploit-driven checks for recently published vulnerabilities. A mailserver which is vulnerable to compromise will start sending out solicitations related to discount pharmaceuticals almost instantly. While I am hesitant to discourage anyone trying to grow their own vegetables, I also feel that it's a bit irresponsible to be too cavalier about mailservers. Any mailserver should have at least a basic level of active maintenance to ensure vulnerabilities are patched and to monitor for successful exploitation. I would not recommend that a person operate a mailserver without at least a basic competence in Linux administration and security.

Look at it this way: the security and abuse landscape of email is such that the line between being one of the good guys, and being part of the problem, is sometimes very thin. It's easy to cross by accident if you do not learn best practices in mail administration and keep up with them, because they do change.

Privacy and Security

The major motivation for self-hosting email usually has to do with privacy. There's a clear benefit here, as most ad-driven email providers are known to do various types of analysis on email content and that feels very sketchy. There may also be a benefit to privacy and security in that you have a greater ability to control and audit the way that your email is handled and protected.

There can be some more subtle advantages to running your own mailserver, as well, since you can customize configuration to meet your usage. For example, you can simply not implement interfaces and protocols that you do not use (e.g. POP) to reduce attack surface. You can set up more complex authentication with fewer bypass options.

And you can restrict and monitor access much more narrowly in general. For example, if you run your own mailserver it may be practical to put strict firewall restrictions on the submission interface.

One benefit that I rather like is mandatory TLS. SMTP for inter-MTA transfer provides only opportunistic TLS, meaning that it is susceptible to an SSL stripping attack if there is an on-path attacker. A countermeasure already often implemented by email providers is a mandatory TLS list, which is basically just a list of email domains that are *known* to support TLS. The MTA is configured to refuse delivery to any of these domains if they do not accept an upgrade to TLS and provide a valid certificate. This is basically the email equivalent of HTTPS Everywhere, and if you run your own mailserver you can configure it much more aggressively than a major provider can. This is a substantial improvement in security since it nearly ensures in-transit encryption, which generally cannot be assumed with email [4].

We must remember, though, that in general the privacy and security situation with email is an unmitigated disaster. Even when running your own mailserver you should never make assumptions. A very large portion of the email you send and receive will pass through the same major provider networks on the other end anyway. There is always a real risk that you have actually compromised the security of your email contents, as commercial providers generally have a substantial professional security program and you do not. The chances are much higher that your own mailserver will be compromised for an extended period of time without your knowledge.

It is also more likely that your own mail server will compromise your privacy by oversharing. Distribution packages fortunately often include a default configuration with reasonable precautions, but mail services can include a lot of privacy and security footguns and it can be hard to tell if you have disarmed them all. For example, when using SMTP submission many MTAs will put the IP address of your personal computer, and the identity and version of your MUA, in the outbound email headers. This is a breach of your privacy that can be particularly problematic when your email address is related to an endeavor with a DoS problem like competitive gaming. Commercially operated services virtually always remove or randomize this information, and you can too, but you have to know what issues to check for, and that's hard to do without putting appreciable time into it.

Advice

Do I think running your own mail server is a good idea? I do not have an especially strong opinion on the matter, as you might have guessed from the fact that I both run mailservers and pay Fastmail to do it for me. I think it depends a great deal on your needs and desires, and your level of skill and resources.

I will offer a few things that I think are good rules:

- It's not a good idea to run a "set-and-forget" mailserver. If you decide to run mail yourself, you should be ready to spend a bit of your time on maintenance and monitoring at least monthly. This won't be very time intensive, but it should be attended to regularly.
- Email can be a critical, central identity proof that your access to a lot of things relies on. Consider the substantial risk that an issue with your DIY mail server will result in your losing your ability to access things like your Google account until you get it fixed. I have seen college students go through absolute

hell because they decided to run their own mailserver, it broke in a difficult to diagnose way, and now they have to call telephone support at the bank, insurance company, etc etc to get the 2FA email on their account changed.

- I am skeptical of one-click solutions like iRedMail due to how opaque email troubleshooting can be. If you want to run your own mailserver, I would strongly encourage you to take it as a learning experience and get smart on how the MTA, MDA, spam filter, etc actually work. The architecture of a typical modern mailserver can be hard to wrap your head around, but you will have a significantly easier time troubleshooting problems and adjusting things to work the way you want if you have a block-diagram-level understanding of how all the software you rely on works. Unfortunately the "in a box" type offerings have a habit of discouraging this, and often even go for a more complex architecture than you need because greater generality makes automation simpler (but makes the resulting configuration more complex).
- You're better off running a mailserver on a well-worn distribution that is popular for infrastructure, like the Red Hat or Debian families. They tend to solve a lot of the problems you're likely to run into in advance via their packaging and default config conventions.

And my advice for running a mailserver:

- Do not cheap out on providers. Run your mailserver in the most reputable IP space you can find. Most likely you will use some type of cloud instance or VPS (probably in such a way that the line between the two is blurry). Unfortunately, per-month cost tends to correlate directly with IP reputation quality, as does name recognition. But neither are any guarantee. AWS has plenty of IP reputation problems. Sometimes a little-known, brand-new VPS provider can be a surprisingly good option because they got their IP space off of some staid corporation that maintained an impeccable security program. Do some research to try and find out what kind of results people get from different providers.
- From an IP reputation perspective, consistency is key. Mailservers and IP reputation are like credit cards and your credit report: once you open one, you want to keep it as long as possible so that it will build a good IP reputation. Do not move your mailserver around between providers to save money, it will result in headache.
- Do not run a mailserver off of IP space that is not intended for commercial services. For example, running a mailserver on a residential ISP is a terrible idea (if it even works, many residential ISPs drop outbound on port 25). You will be guaranteed to have IP reputation problems and, moreover, IP space allocated to end-users usually gets reported to a "policy block list" such as the one Spamhaus maintains... meaning your outbound email will be blocked by recipients simply because it's coming from an IP that should not have mailservers, so it's most likely to be from a compromised end-user device.
- Check every box when it comes to auth and trust measures. Set up SPF, DKIM, and DMARC--read enough about these that you are confident you have set them up correctly. They will not prevent deliverability problems, but they will help, and incorrect or missing SPF in particular is virtually guaranteed to cause a high rate of SMTP rejections. Use an external service like check-auth@verifier.port25.com to ensure you've set it all up correctly.
- Set up fail2ban and make sure it is working correctly. Otherwise you will pull your hair out every time you have to look at a log due to all the auth rejections, and even worse someone might actually guess a password correctly one day.

- Be very conservative from a security perspective. After all, a big benefit of running your own is the ability to do this. Disable interfaces you don't use, restrict network connections to the mailserver, require two-factor auth, etc.
- If your mailserver has more than a trivial number of users, one of your biggest headaches is going to be your own side. Especially at an institutional or business scale, your users *will* get their credentials compromised (via phishing or whatever) and those credentials *will* be used to send spam via your mailserver. Even if you are the only user, vulnerabilities in webmail are highly prized because they may allow someone to use your webmail to send email via your mail server. All of this will cause your mailserver to end up on blocklists and gain a high spam reputation with major providers. Un-intuitively, this is why *outbound* spam detection can be very valuable. Passing your outbound email through SpamAssassin and alerting on hits (or for large servers, on an unusual number of hits) will allow you to quickly detect compromised user accounts (or webmail, or mailserver, etc) and cut them off quickly, hopefully minimizing the future impact [5].
- If you don't use webmail, don't offer it. It's a huge increase in attack surface. If you do run webmail, I recommend putting it behind another authentication system (e.g. a reverse proxy implementing oauth). It will reduce the chances of compromise by orders of magnitude. Similarly, if you are the type of person who heavily uses a VPN, restricting access to SMTP submission, IMAP, manage-sieve, etc to the VPN endpoint might be a fairly cheap measure that gets you a huge improvement in security.
- Managing your email storage, retention, backup, etc. can become surprisingly complex surprisingly quickly. You have multiple options for how to actually store email ranging from relational database to various flatfile formats. Learn enough about them to make an informed decision. Unfortunately, Linux being Linux, there are weird caveats to basically all approaches.
- This is more of a matter of opinion, but where possible I prefer to use Unix accounts for mail rather than virtual accounts. Part of this is because I use directory authentication (LDAP/Kerberos) wherever possible, but even if you don't, using Unix accounts for mail tends to be simpler to troubleshoot than virtual accounts, and it reduces the number of distinct authentication points. On the other hand, I tend to recommend *not* storing mail in user home directories. It does simplify backup and in general make more sense, but it becomes way too easy to accidentally delete your whole mail spool.
- And on that topic, if you're going to run a mail server and you're not using a 100% virtual approach with everything stored in a relational database, it will be well worth your time to become an expert on Linux file permissions and the SetUID behavior of your MDA and, possibly, MTA, filter, etc if they interact with file storage. Many of the real-world problems you'll run into will turn out to be related to file permissions and what UID a specific component of the mailserver is acting as when it takes a specific action. Depending on details of the setup, processing an individual incoming email often includes steps that run as a service user, *and* steps that run as the user the email is being delivered to.
- The Sieve filtering language is substantially better than procmail. Don't even get started with procmail, use an MDA or IMAP server with Sieve support. Writing Sieve rules is not always the most fun thing, but if you get manage-sieve working (likely built into your IMAP server!) there are several GUI web and desktop mail clients that will let you manage your Sieve rules using a familiar, gmail-like interface.
- Mail is an especially arcane and eldritch world of software and you will gain some gray hair in the process of becoming competent. That's just the joy of computing!

Surely I have said at least one thing here which is controversial or just outright wrong, so you can let me know by sending an email to the MTA that I pay some Australians to run.

[1] Basically, MTAs especially are designed around a fundamentally distributed architecture and try to view mail handling as a completely general-purpose routing exercise. That means that there's no strong differentiation between "inbound" and "authenticated outbound" messages at many points in the system. Obviously there are canonical ways to handle this with DKIM, but nonetheless it's surprisingly easy to do something extremely silly like set up an open relay where, as a bonus, OpenDKIM signs all of the relayed messages.

[2] Some of this over-architecture is simply a result of the natural worst tendencies of academic software development (to make the system more general until it is so general that the simplest use-cases are very complex), but some of it is a result of a legitimate need for mail software to be unusually flexible. I think we can attribute this to three basic underlying issues: first, historically, there has been a strong temptation to "merge" mail with other similar protocols. This has resulted in a lot of things like MDAs that also handle NNTP, leading to more complex configuration. Second, there is a surprising amount of variation in the actual network architecture of email in institutional networks. You've experienced this if you've ever had to carefully reread what a "smart host" vs. a "peer relay" is. Third, MTAs and MDAs have traditionally been viewed as very separate concerns while other related functions like LDA and filtering have been combined into MTAs and MDAs at various points. So, most mail software tries to support many different combinations of different mail services and inter-service protocols.

[3] I have a story about once submitting more than one abuse complaint to gmail per week for over a year, during which they never stopped the clearly malicious behavior by one of their users. The punchline is that the gmail user was sending almost daily emails with so many addresses across the To: and Cc: (over 700 in total) that the extreme length of the headers broke the validation on the gmail abuse complaint form, requiring me to truncate them. I also detailed this problem in my abuse reports, it was also never fixed.

[4] This is increasingly getting written into compliance standards for various industries, so more and more paid commercial email services also allow you to configure your own mandatory TLS list.

[5] This issue will mostly not apply to "personal" mail servers, but at an institutional scale it's one of the biggest problems you'll deal with. I went through a period where I was having to flush the postfix queue of "3 inches longer" emails more than once a week due to graduate students falling for phishing. Yes, I'm kind of being judgmental, but it was somehow always graduate students. The faculty created their own kinds of problems. Obviously 2FA will help a lot with this, and it might also give you a bit of sympathy for your employer's annoying phishing training program. They're annoyed too, by all the phishing people are falling for.