Reliability: It's Not Great

kurt **4h**

The last four months have been rough. We've had more issues than we're OK with.

I've hesitated to share this because, well, I'm fighting a debilitating feeling of failure. Fear, too. If we don't improve, our company ceases to exist, and I really like working on this company.

One interesting problem we have is that we've exploded in popularity. It sounds like a good problem to have! But we've pushed the platform past what it was originally built to do. We've put a lot of work and resources into growing the platform and maturing our engineering organization. But that work has lagged growth.

This sucks for you all individually. You don't really care about our popularity. I mean, a lot of you do. But, really, you just want to confidently ship your apps.

That's what we want, too. It's a grind, though, and I think we're not as forward about our struggles as we should be. Y'all are devs, like us, and we should have been trusting you with the grimy details. So, here some of them are.

Our platform is a bunch of moving pieces that all need to work together so you can deploy an app, deploy it again, walk away, and then come back 24 months later and find out it's still working. Here's what goes into making that work:

- A centralized API that does auth and CRUD stuff against a database,
- The WireGuard gateways `flyctl` uses to connect to your organization's private network,
- Remote Docker builder VMs `flyctl` uses to build your app into a Docker image,
- A global Docker Image registry to hold those Docker images,
- A secret storage Vault,
- A scheduler that launches Docker images in VMs (that's **Nomad** for most apps today),
- Service discovery to propagates information about all the VMs running in our infrastructure,
- The proxy that routes traffic to your app instances, and
- Networking infrastructure to link apps up with each other.

These have all failed in unique and surprising ways. Often, when this happens, we get lucky, and you don't notice the hiccups. But sometimes we get unlucky.

Skip to main content

In no particular order, here are some major incidents from the last 4 months:

- Service discovery & Corrosion
- Centralized secret storage
- Postgres
- Capacity issues
- Volumes pinned to host hardware
- Status paging

Service discovery & Corrosion

We propagate app instance and health information across all our regions. That's how our proxies know where to route requests, and how our DNS servers know what names to give out.

We started out using HashiCorp Consul for this. But we were shoehorning Consul, which has a centralized server model design for individual data center deployments, into a global service discovery role it wasn't suited for. The result: continuously stale data, a proxy that would route to old expired interfaces, and private DNS that would routinely have stale entries.

All of this was a consequence of round-tripping every state update we had (every VM start and stop) through a central cluster of servers, often transcontinentally.

In response, we've shipped a project called Corrosion. Corrosion is a gossip based service discovery system. When a VM comes up, that host gossips the instance information. Corrosion's goal is to propagate changes in under one second, globally (and to get as close to instant as possible).

The problem with Corrosion is that it's new and gossip based consistency is a difficult problem.

We got Corrosion out the door quickly because Consul was causing problems for users. It's new software, and it's caused a pair of issues. Both manifested as corrupted global service discovery state. The first issue happened when one of our process spammed Corrosion with updates, essentially turning it into an internal DDoS. The second occurred during a routine update that unexpectedly messed up a database.

The effect of both of these issues was to break applications during deploys. As VMs came and went, our proxy and DNS servers would find themselves stuck working off stale data.

Corrosion needs to be more resilient to failure. We're doing incremental things to improve it (rate limits, for instance, mitigate the "internal DDoS" risk). But we're working on architectural changes, too. Gossip is hard because issues aren't easy to trace to specific broken nodes, and it propagates quickly, which is what you don't want when there's a problem.

Moving off Nomad will also help mitigate Corrosion issues. Because Nomad creates entirely new instances for each deploy, there's a lot of service discovery churn; many, many event updates per second. Fly Machine-based apps are less frantic – when we update an app running on Machines, we do it in place.

Finally, and this is sort of a general thing not just about service discovery: we deploy a lot of changes to our platform during the week. Sometimes, our changes have collided with yours; an ill-timed app deploy can leave that app in a wonky state. We're updating our tooling so that app deploys are paused at these times, and when that happens, we'll make it as obvious as possible why.

 et Storage

We store application secrets in HashiCorp Vault. HashiCorp Vault works a lot like Consul does, with a central cluster of servers.

The problems we have with Vault are less severe than the ones we had with Consul, but they rhyme with them. Every time a new VM boots, the worker running it has to pull secrets from Vault. There are two basic problems with this:

1. Vault is in the US, internet connectivity between distant regions (like MAA) and the US can cause secret lookups to fail
2. There are failure scenarios that will make Vault inaccessible. For instance, we **had a hardware** failure on one of our Vault servers that caused widespread VM creation failures.

As with service discovery, these problems are exacerbated by Nomad and mitigated by Fly Machines. But new Fly Machine creation will also fail if Vault is in a bad state.

This is a theme. Existing open source is not designed for global deployment. So when we make the choice to "buy" existing infrastructure software, we're often paying in part with global resilience.

Postgres

Our Postgres clusters have had two major problems: (1) our reliance on Stolon and live connections to Consul clusters, and (2) the expectations we've set with "unmanaged Postgres".

The first is an architectural problem. The Consul clusters Postgres depends on are different than the ones we use for service discovery, but they can still "fail" in strange ways. Stolon, the Postgres cluster software we built the first iteration of Fly Postgres on, doesn't handle Consul connection issues well.

New Postgres clusters don't use Stolon, and instead come up with **`repmgr`**. `repmgr` handles leader election within the cluster, without needing a second database. These new Postgres clusters still use Consul to share configuration information, but if Consul melt downs, the cluster keeps going.

We are working on getting previously provisioned Postgres DBs upgraded to the new `repmgr` setup. There are complications, but we'll keep posting about this.

The second problem we have with Postgres was a poor choice on my part. We decided to ship "unmanaged Postgres" to buy ourselves time to wait for the right managed Postgres provider to show up. The problem is, `fly pg create` implies that people are getting a managed Postgres cluster. That's because every other provider with a "get an easy Postgres" feature gives you a managed stack to go with it.

This makes sense *now*, but was a surprising lesson for me. We ended up presenting a UX that promised a lot, then not following through. We're not the type of company to writes value statements, but if we were, we'd write something like "don't create nasty surprises by violating developer expectations".

We're going to solve managed Postgres. It's going to take a while to get there, but it's a core component of the infrastructure stack and we can't afford to pretend otherwise.

Capacity issues

An influx of new users ran us out of server capacity in multiple regions, sometimes more than once

This was a failure on two levels: we didn't buy servers fast enough, and we didn't have good tools for taking pressure off specific regions.

Last year, I assumed that if we hit capacity issues, we could prevent *new* users from launching in specific regions. This didn't pan out.

The Heroku exodus broke our assumptions. Pre-Heroku, most of the apps we were running were spread across regions. And: we were growing about 15% per month. But post-Heroku, we got a huge influx of apps in just a few hot spots — and at 30% per month.

In hindsight, I should have started acting like we were doing srsbzns much earlier, basically as soon as we had investor cash to spend.

We're getting better with capacity planning and logistics. I was doing capacity planning as a side hustle to the rest of my job. The company needed to scale beyond my spreadsheet. We've hired here, and reorganized a bit; things are a little better now, and they will rapidly improve.

Volumes are pinned to host hardware

The `fly volumes` command creates a block device on specific host hardware. When we first shipped this, we had a lot of content explaining the limitations of this approach. We designed our volumes to run in sets of 2+.

This means that if the host your volume is on goes down, your app goes down. If the host doesn't have enough memory or CPU available to run your app VM, you may not be able to deploy.

Those details got lost as our docs improved, however, and it's led to some nasty surprises. It's also counterintuitive. People are used to AWS EBS magic. But our volumes aren't EBS (I shipped the initial version of volumes myself!)

This is another case of the UX creating the wrong expectations.

Status paging

We're taking a lot of legitimate flak for vague-posting on our status page. Or not posting on our status page. All while we're being shamelessly rah-rah about our tech stack in blog posts. Issues happen, and we have not communicated aggressively when they do. That makes us look out to lunch.

This is hard. Even this post is hard. Our egos are all wrapped up in this work. We want you to know everything that's going on, but it's easy to slip when we're tired.

Some of the challenges I've written about here are Hard, in the CS sense of the word. But this problem isn't. There's no way to excuse it. We're just going to be better at communicating immediately.

We've hired a really great person to build up our Infra/Ops organization. In addition to beefing up that team so that it's no longer spread so thin it's translucent, they're also standardizing our incident responses. When the shit hits the fan, we want as few decisions to make as possible, so we can get information out quicker.

We're also shipping a personalized status page. As our fleet grows, and we rack more and more servers, the chance of us experiencing a hardware failure at any given moment increases. This has made it tricky

to have an honest status page. The personalized status page will make it easier for us to tell specific customers impacted by hardware failures "hey, a drive died in this region, we're working on it".

This is going in the community forum specifically so you all can reply to it. You may throw tomatoes, if that's your thing. Or ask questions. We're in an awkward phase where the company isn't quite mature enough to support the infrastructure we need to deliver a good developer UX, and we're going to take the bad with the good until that changes.

Plans for fully-managed postgres?

matsea 3h

Thank you for being open with us.

While our apps might or might not be critical to our companies or ourselves, in the end a lot depends on whether we believe we made the right choice when picking the company hosting our work. For some the issues of this past weekend might have been the last straw (and that has to be okay for you), for others (me included) it might have been deeply frustrating, but reading your post does put me at ease somewhat - at least until the next issue

For me one of the critical things is your communication when things go wrong. Make sure we don't have to hunt the forums for hints as to what's going on.

Get your self a donut and a strong coffee and then good luck prioritizing what needs focus first!

Matthias

notif 3h

Hang in there Kurt! I appreciate the transparency and as a startup veteran I can commiserate. And while I've been frustrated a time or two recently with Fly, I'm still a paying customer and have confidence you'll all work through it.

Harold

ignoramous 2h

Thanks; 'twas a bit hard to read so I can only imagine.

As for tomatoes: I've not known a simpler compute platform (cue Rich Hickey's *Simple made Easy*), and given **the team's background** , pretty sure Fly will get better at ~~sandwiches~~ stateful workloads and incident management, too. Best.

cldellow 2h

I appreciate this post! My SaaS company has a use case for which Fly would be a great fit.

Skip to main content is very important to our business, so I've approached Fly with caution. I started by putting some personal projects on Fly's (very generous) free tier. It was a mixed bag. I now suspect

that I perhaps ran into the volume issue on one of my projects – I was doing something that didn't seem dangerous/controversial, but ended up with an undeployable app and downtime.

My impression based on those experiences is that I wouldn't be comfortable moving our business to it just yet. At least, not without having a plan to quickly cut Fly out of the loop.

This article goes a long way towards helping me believe that Fly can be, and hopefully will be, the best place for this workload someday. Thanks for the transparency.

---

nicoburns 2h

As a **fly.io** customer I'm very happy to read this. Reliability is definitely your core value proposition in my mind, and I have experienced a few reliability issues recently (enough that I'm now very glad I didn't migrate our production systems at $OLD_DAY_JOB to **fly.io**). It's great to hear that you're taking this seriously and have a plan in place to fix things. I'll probably reevaluate fly's production-readiness in ~6 months once you've had a change to work things out.

I'm also super happy to hear that you're planning to ship a fully managed Postgres. A managed data store is really THE thing I want from a cloud provider. Running applications on a platform like **fly.io** is convenient, but running applications on a plain linux VM isn't all that hard either. The one thing I *really* don't want to manage if I can help it is the data store where durability is critical and hard to get right without experienced ops personnel. If you can ship a managed postgres that gives me access to logical replication slots then I'll be singing your praises to whoever will listen.

Finally, I have a request for something you haven't mentioned: better error handling / debugability / observability into the **fly.io** system. When I've had errors deploying to **fly.io** the error messages have been pretty unhelpful. I have had the generic and cryptic "Failed due to unhealthy allocations" in two separate scenarios:

1. My app was compiled with two new a version of glibc and (presumably) crashed on startup. I would expect to get an "app crashed on startup" error message here with at least the process exit code and ideally some kind of debugging information (although I understand this is a tricky case where there might not be much available).

2. During a brief a period of downtime. I'm ok with some limited amounts of downtime, but I expect your system to that it is at fault and not leave me chasing around trying to work out how I've managed to break

---

nilsbunger 1h

I'm not a huge customer of **Fly.io** (yet!) but I've loved the experience so far. I think if you guys focus on reliability over adding new capabilities or new customers for a while, you have the potential to make this service rock-solid.

Also, some of the things you mention, like volumes being tied to hosts instead of floating like EBS, is a plus from my perspective. EBS is slow and expensive. When I want to run a database, I want the volume on the machine if at all possible. I don't need to move the disk image between machines - I need rock-solid backups of the volume or the DB (maybe even a copy as a warm standby).

Skip to main content

I'm sure it's a slog, but doing the grunt work step-by-step is where you create the long-term value for yourselves and your customers.

---

julia 1h

Thanks for this post – really appreciate the transparency . I loved the explanation of how volumes are pinned to host hardware, which I hadn't understood (I actually want to use them *more* now that I know how they work!).

Just wanted to say that I've loved having fly as a way to deploy my projects despite the occasional hiccups.

---

markthethomas 1h

really appreciate this. Especially recognizing the disparity between the "haha let's write a fun and sorta snarky blog post about distributed systems and how we know how to do them!!!" while things are totally down or partially down. I don't have a huge workload on **Fly.io** yet, but the reliability has been a pretty sore spot for me so far. Hard to think of any other service (compute or not) I've used that has had similar spottiness.

That being said, I think if y'all really focus everything on reliability and communication, the upside is still incredibly high. I would *personally* take an actually-reliable service over basically any other feature offering you're thinking of delivering or working on right now. For example. the Postgres service / offering - hard to imagine considering it any more than I have already while the core service reliability seems to be pretty low. Just an example, but I think it would apply to any other feature — I'd ask "how can I trust XYZ thing if the basic ability to deliver / deploy a service is shaky/flaky?" Everything gets built on core trust + reliable systems.

last thing I'll say: would rather see actual reliability / a solid service over a blog post writing about it *every single time*. Actions > words and all that. Ty all and best of luck!

---

markthethomas 1h

Also good example: maybe it's just me or something local, but as I'm writing this post **https://fly.io/** is sending a 502 to me. Not great

---

kurt 1h

@markthethomas you're spending enough that the plans with support emails are "free". We're happy to help you troubleshoot 502s. This probably isn't us, but we'll check anyway.

---

markthethomas 40m

st me" but idk why the main site would be down just for me. not urgent, but I

Skip to main content

appreciate y’all reaching out

---

kurt 33m

Oh I misread that! A 502 from **Fly.io** is definitely not just you, it’s probably a bug.

---

obra 32m

<3

Thank you so much for your openness and transparency. You’ll get through this.

---

miharekar 15m

Thank you for sharing this. I can imagine that a lot of us rushing from Heroku took you by surprise. And I can certainly feel you stabbing the problems multiple times, when the docs keep changing and a couple of months old post on Community is completely out of date

But I’m very happy with Fly overall and I do wish it/you all the best and hope you’ll pull through and keep improving the service and keeping it alive and well

---

markthethomas 9m

all good!

---

bkane 1m

Serious reply:

I don’t have a need for Fly currently, but I love your blog posts and the humility in this post. I hope someday I will have an opportunity to put an app on **Fly.io**

Less serious reply:

> The problem with Corrosion is that it’s new and gossip based consistency is a difficult problem.

I’m glad I’m not the only one having trouble with **Challenge #3b: Multi-Node Broadcast · Fly Docs** (implementing gossip protocol)