Migrating to the Cloud is Chaotic. Embrace it.

There's plenty to like about the cloud, but newcomers almost always dread its main drawback: its ephemeral nature. They know they can't count on servers to stick around indefinitely. They expect disk performance to tank without warning. They fear the day that some critical managed service suddenly goes dark.

As Chaos Engineering evangelists, we hear these fears all the time. "The cloud is chaotic enough! Why would we add more chaos on purpose?" We get it—we've answered the 3am pages too. But remember: Chaos Engineering grew up in the cloud. Netflix embraced the problem of vanishing cloud servers, they didn't fear it. They created Chaos Monkey not as an optional tool to give their engineers, but as a mandatory tool to impose on them. As we recently put it, the best way to get ahead of natural chaos is to synthesize your own, and synthetic chaos is especially powerful when you apply it regularly—when it's non-negotiable. As you make continuous chaos a part of your normal engineering practice, you gradually learn to stop fearing the ephemerality of the cloud.

What's even more chaotic than running in the cloud? Migrating to it. Although you might rather start your Chaos Engineering practice *after* your migration, the truth is that you should be proactively testing and identifying weaknesses as you move over pieces of your infrastructure.. Think about it: this is a golden opportunity to test how your cloud infrastructure behaves, as your old infrastructure is still taking most production traffic. Better to run your cloud environment through the ringer now while it's still in its embryonic stage and it's low stakes. What's the alternative? Migrating all of your mission-critical services and finding out they don't behave in the cloud like you'd hoped? In that scenario, you are impacting customers and not doing what you should to make sure your systems are behaving the way you want them to.

If you're moving to cloud, you're probably changing your stack in a few other ways too. Perhaps you are experimenting with containers; perhaps you are implementing new CI / CD tools. This is your chance to start doing infrastructure-wide testing, before you get years down the road and don't truly understand your system. This process is table stakes in writing software: when you are building a new application, you don't avoid writing tests for the first six months, do you? No, you do it from day one. We need to shift the mindset of operations from reactive to being more proactive with chaos engineering.

So too should you practice Chaos Engineering from day one in the cloud, because it's often more chaotic—or at least more ephemeral—than on-prem.

When you migrate, oftentimes stuff isn't **configured** the same way in the new environment; at least not at first. Chaos Eng can help you catch these misconfigurations (e.g. database replication not working right? Then when you shoot the master in the head and the slave takes over—but it has no data—you catch the misconfiguration.)

Objections from the reader/migrator:

Examples of when to do chaos eng during migration:

- Database failover—kill your new, cloudy master node to make sure failover works (and that the slave is replicating).
- Noisy neighbor? Using a chaos tools to chew up your CPU in your original environment, you can
- Some services still running on-prem? Test your latency to them. Test *Blackholes* on them!
- Test disk latency?
- Don't assume new resilience mechanisms from your cloud provider work as you expect. (Honeycomb found out the hard way that Amazon RDS only fails over to hot spares in the face of total infrastructure failure of the primary DB—it does not account for slow queries/bad performance): https://www.honeycomb.io/blog/2018/05/rds-performance-degradation-postmortem/
- Your cloud environment may be more spread out than your old environment. There's more space, literally, between all your services. (And we saw in The Cost of Downtime post that Network outages are the number one cause of downtime.)
- Chaos Engineering forces you to confront your steady state: to know what that state is, think about how to measure it, and how to test its strength.
- The ephemerality of the cloud

Don't just pick random things to test. Move some stuff over to the cloud, watch your monitoring dashboards, and see where the problem areas may lie. If you don't see anything the first few days, wait a few weeks.

The meta point: maybe your on-prem deployment was super stable. They can be in certain ways (low network latency, low-to-no contention for CPU/disk, etc), and not in others (not resilient to DC outage (if you're in one DC), major network outages, etc). But if it's stable, and you've gotten complacent