On the lessons learnt by approaching chaos and resilience engineering in a midsize organisation.

Nazareno V. Feito Matias

talk given at 2nd European Chaos Engineering day

Stockholm, Sweeden; Nov 2018

Abstract

Chaos engineering is a newly created discipline, although we have been applying similar principles in the past, the newly created plethora of tools in the market are not yet integrating with every single cloud environment that exists. Also, as a newly created discipline we wanted to explore it in a singular way, applying the principles that we considered worth applying and creating our own path as we moved along.

We considered the principles to be guidance, not written in stone, and we were glad to find that walking on parallel lines and even on divergent lines, gave us valuable insights. In this paper we uncover our findings and attempt to provide the reader with valuable insight as well, on our successes and pitfalls, what worked well and what did not, hopefully in keeping the conversation going, we are able to contribute back to this amazing community behind a united engineering, an engineering that aims to make our systems more resilient, namely chaos, resilience, reliability, performance or any other name, engineering.

On learning the ropes

Short deep dive in the literature

The learning aspect of chaos engineering is vast, and in many cases it is more convenient to build a small foundation and start applying that knowledge, at the same time the learning increases.

The foundation needed in this case is basically:

- Scientific method[1], and how controlled experiments work.
- Principles of chaos[2].
- Netflix's book on chaos engineering[3].

An overview on {performance, resilience, reliability} engineering may prove useful, although not required.

Research on the tools at hand and selection

Nowadays, there are a significant number of tools for chaos and resilience engineering[4], the difficulty lies in selecting the one that is adequate for us. We use what we have called a **technology evaluation matrix**, and it helps us in deciding whether a given technology will be a good fit.

The matrix contains columns as follows:

- Technology or tool.
- Grade (the ponderation of all the remaining columns, defined as an average).

The following are critical columns, failing in any of them means the tool is a no-go.

- Maintained.
- Success cases; who is using it.
- Compliance. SOC, PCI, etc.

- Scalability.

The following are expected, does not kill the tool but are necessary, albeit not sufficient for production:

- Maturity.
- License.
- Performance.
- Security.
- Support.

The following are wanted features, less important than expected, but still important:

- Two year prospection.
- Integration with our platform or the platform we are running on.
- Integration with 3rd parties.
- Fulfills immediate needs.
- User friendliness.

The following are nice to have:

- Advantageous features over competitors.
- Flexibility, can we add/request new features.
- Public roadmap; knowing where the tech is going.

Finally, as suspected, the rows are occupied by the technologies to evaluate, and every cell gets a score from 1 to 10.

On building the ecosystem

Our thinking was that chaos and resilience engineering cannot thrive without the symbiosis of people and systems, hence we have defined two building blocks:

1. Building a community.
2. Building our own systems (tools).

Building a community

Reasons

Chaos engineering is a group based activity, from creating the tools, to the experiments to running gamedays; a significant number of engineers, managers and other stakeholders will be involved, from buy-in, sign-off, runbooks, so on so forth.

If people do not understand something, they are either scare of it, or reject it without thinking twice, thus it is important to share the knowledge, to communicate how this can make not only the systems better but their lives easier.

On gamedays, having people reluctantly participating is a terrible experience, when an entire engineering team is against what we are onboarding, there is no possible outcome in which to finish on top.

Pros and cons

The pros are clear, when people understands what chaos engineering is about, they are on top of things, they are willingly participating on experiments, gamedays, bouncing ideas and in our experience, is easier when suggestions on how to improve things come from outsiders and not only the chaos engineering team.

The cons occur when too many people get involved, and inevitably, statistically at least 10-20% of people will offer resistance, no matter what the good points are, some people just want to see the world burn; well, if that is the case, maybe chaos engineering is a perfect fit as well, just need to tweak the pitch.

Guidelines

We have taken a baby steps approach in our organisation, including these:

- An Oracle group, it allowed us to spread the word, posting interesting articles, and starting discussions.
- A slack channel, it is always interesting to leverage slack, since it integrates with other tools and people are comfortable with the technology.
- Blog posts and internal documents are also very useful, the shorter the better.
- Publishing whitepapers and papers, sharing insights obtained from experiments show the value

of the discipline, and the more so, of the experiments.
- And the last one, the most important one, talking to people, telling them what we are working on, and in which way we are setting the ground for future experiments and gamedays.

Building our own tools

The decision to build our own tools was not an easy one, and it appears as it usually does, by an unfulfilled need, only to then be fueled by curiosity, there was no chaos engineering toolkit that allowed for integration with Oracle Cloud infrastructue. Since OCI was our main goal, we created one[5].

We started with a first principles approach[6], bottom up from the most basic requirements:

- We started with "What do I want to integrate into", DC, AWS or OCI, and the answer was clear.
- Will it be a one-time or recurrent thing? in order to define a system or a simple one-off script.
- We wrote a requirements and constraints list, including: platform, speed, integration, scale, flexibility.
- We wrote a feature list for an MVP.
- We evaluated the languages with these in mind;
  a) avoid trends.
  b) rapid prototyping.
  c) stable and mature.
- We finally designed the internal and external architecture; admittedly, parts of the system were born organically; We had the following list at hand as we evaluated how to apply it.
  a) thinking as a service or microservice.
  b) going serverless? containers? vm's?
  c) Frequently asking for feedback as new features were added.

We kept in mind the following quote.

> *I do not know the secret to success, but the secret to failure is trying to please everyone.*

On the application points

Failure to launch

Right from the start we faced an expected difficulty, the fact that upper management in most companies are reluctant to do experiments in production, not an unwise choice if you ask around; however, not only production was unexplored territory, but also pre-production environments, making it very difficult to create experiments in any layer of the stack, people just did not believe in gaining confidence by breaking things, and the problem was that people thought that was what we wanted.

The difficulty was there for two reasons, the first one, nobody actually knew well enough what chaos engineering was about, and they did not have the time or eagerness to learn, what worked well in the past, was solid ground, anything else was smoke and mirrors.

The second reason was fear, fear can and often is the ultimate innovation blocker; we needed to take action in order not to derail our initiative, so we decided to be patient, start small, build trust and gain confidence, and this is what we did for several months; the following sections shed light on some of the details of doing so.

Leveraging the architecture level

We started by using the socratic[7] method and asking one question after another, needless to say several contradictions arrived, but one question surfaced clearly, what if we apply the discipline and systems prior to production, not only staging or testing, but much earlier than that; the early bird gets the worm, and we unapologetically dismissed the chaos engineering mantra that says:

experiments must be run in production.

We started at the whiteboard, directly aiming towards the architecture:

- We started applying it to POCs. Granted, we were not keen on trying something this new on a production environment, but also because we were building a foundation.
- We reasoned that validating the architecture initially, can save time, money and reputation; pipeline deployments are costly and a sign-off on the wrong architecture is undesired to say the least.

Test environments are not useless

We were moving slowly but steadily, we wanted to:

- Establish trust, familiarity and credibility. Testing environments allowed us to diminish the associated risks while still gaining valuable insight.
- We reasoned that the same application can be use to validate our architecture design; things must align with the architecture once deployed, if there are significant differences, something went wrong either with the deployment methodology, the systems, the scale or human error (and that includes improper application of chaos engineering at architecture level).
- It was also a way of validating the methodology we were using, with the changes we created and the tools we were leveraging, it was about creating and sustaining a new discipline in a midsize organisation, and that cannot go without growing pains.

New products or acquired products can help

Sizeable organisations either have several products that run in different platforms or may have acquired products, i.e. startups or smaller companies; these provide with advantages, such as:

- They are more keen and less risk averse, especially startups that hold the mindset and culture, we were even working with an acquired product who's motto was *move fast, break things.*
- They offer the entire staging environment as a playground and sometimes small subsets of production, although the latter does not occur often.
- Once a success or big insight appears, based on something that did not hold to expectation, it is a great story and communication transpires; acquired products want to be noticed, so they do the heavy lifting in telling the story for us.

Production, the golden goose

We try as much as we can not to touch production unless strictly necessary, and we think it is a valuable premise, we are not eager to break anything in production; however, we are eager to find SPONF; we dislike acronyms and we are slightly ashamed to say we coined that term, SPONF as Solid points of non-failure, and constitute those places where injecting failure goes unnoticed. The more the better, and that is where we aim our failure injection[8] and chaos.

SPONF stands for Solid Point of Non-Failure

- *A little bit of prod is still prod.* More times than none, chaos engineering in production will be restricted to a small subset, a production component, even if it seems insignificant, a win is still a win; and also, the smaller the subset, the more contained the blast radius can be, unless that small success holds the core of the platform or application, in which case, we would not start there.
- Subsets are amazing, less risk, more to win, being FIT is a great opportunity, in injecting failure we strive for precision and the experiments are extremely controlled, this is one of the best cases.
- We always want to make sure we can rollback in a timely manner, either manually, semi or full automatic; 90% of the time we contemplate several different scenarios in a TWWp, "things went wrong plan"; as usual, we expect for the best but prepare for the worst.

Communication on gamedays

In all honesty, we have not hold many of these since it is usually a difficult drill where many stakeholders need to be involved, so these are exclusive of specific events, such as *black friday* or *peak season*.

- If you got this far, you are doing well, gamedays are not a common cookie and they are always *"fun"* to do, for some people at least.
- We try to loop, as many people as possible, and avoid surprises, or minor surprises, we think of it as a hackaton, sometimes we might even do some homework in advance; we think gamedays are about getting insight, not about exposing people.
- A common practice, removing the SME completely and watching how good the procedures are and how prepared the team is.
- We expect people to take gamedays seriously, we have experience with gamedays in small teams where people think, *if it is only a drill, there is no need to stress about it*; granted, unneeded stress is unwelcome, but if we do not take it seriously, the conditions of the experiment are already worthless.

We have done it, now what?

We believe the most important part, after ending a gameday or a big experiment, and will set the tone for future gamedays and experiments, are:

- Communication[9], we do not do this enough, and we are constantly called upon.
- Championing, in many companies, what is lacking are not sponsors but someone to champion chaos.
- We want to help others to do their own smaller or larger experiments.
- We aim to evangelise, although it may seem boring, sometimes people start taking it seriously after they see someone really passionate about it.
- Automation[10]; key point in running experiments, in any environment, with so many tools nowadays is extremely simple to automate, we schedule our experiments, recurrently for given days where people know an experiment will be ran, and of course, let us not forget to coordinate with the monitoring tools and notify the oncaller.
- Replicate; replicating experiments in different platforms that should behave in the same or similar way is extremely valuable.
- We always search for improvement; experiments can always become more convoluted, complex and complicated, it is the beauty of engineering, there is no end to it.

Conclusions

- We have found that starting small and establishing trust helped us; we tried hard not to antagonise nor ostracise nor expose people, we exposed weak points and the results were better.
- We also found that by injecting failure instead of randomly shutting down resources, gave us a higher degree of confidence, even in staging environments and user acceptance testing, we were fishing for insights, not for wreckage.
- We were not afraid to create our own toolset, and it paid off, but of course, if it is possible to get it off-the-shelf, it is a double win, we were not able to do so.
- We saw how communication was key, before, during and after, it is sometimes the difficult part, technology is easy, people is sometimes a different story.
- By looking for insights instead of looking to break things people were more eager to approach the subject, and we all had a better time, especially customers.
- We discovered that in the end, the best we can do is to keep the conversation going, to keep exploring, contributing and collaborating with each other in a constantly evolving subject, whether is chaos, resilience, reliability, performance or failure engineering.

References

[1] Introduction to the scientific method.
[2] Principles of chaos.
[3] Netflix's report on chaos engineering.
[4] A curated list of chaos engineering resources -tools-
.
[5] On the design of a chaos and resilience engineering system with applications oriented towards Oracle Cloud Infrastructure.
[6] Aristotle on induction and first principles.
[7] The socratic method.
[8] On failure injection on sw and hw systems, Mei-Chen Hsueh, Timothy K. Tsai, Ravishankar, K. Iyer.
[9] The Importance of Communication within Organizations: A Research on Two Hotels in Uttarakhand, Dr. Shipra Agarwal, Mr. Ashish Garg.