On the design of a chaos and resilience engineering system with applications oriented towards Oracle Cloud Infrastructure

Nazareno V. Feito Matias
nazareno.feito.matias@oracle.com

Oracle Corp. NSGBU

Nov 2018

Abstract

Chaos engineering has taken the world by storm, as cliche as it sounds. However, as often times happens, the tools available were not suitable for Oracle Cloud and the experiments conducted within NSGBU.

Oracle Cloud Infrastructure is in its infancy, and as such, there is no ecosystem for it for the majority of the trendy tools out there; as it often happens as well, we decided to create a new tool, that gives us the flexibility of adding features without waiting for others to do so.

It was possible to create a minimalist chaos engineering tool that is advanced enough to be used in multiple environments, with several different features; that combined make a powerful cocktail that can render an infrastructure to its knees if not appropiately conducted.

We have found that running chaos engineering experiments jointly with architecture design, POC and testing, yielded insights that can save the company time and efforts; it prevents moving forward with an unsuitable solution. We also found that creating a tool provides a major understanding on the requirements, allowing the design and implementation of the precise features needed for each experiment.

keywords chaos engineering, failure injection, oracle cloud, resilience.

Introduction

Aiming chaos engineering towards a new cloud infrastructure is not as simple as it seems when the cloud infrastructure is also in its infancy. The reason for this is that the ecosystem is not yet there and it might take years for that to happen. When this situation first appeared with Oracle Cloud we had three choices:

1. To wait for an adequate tool to appear and handle our chaos engineering with automated scripts and manual work.
2. To port one of the tools in the market, since some of them are open source, it was a possibility.
3. Create our own tool, and that will give us a better understanding of chaos engineering plus the flexibility of being able to modify it as we see fit.

Needless to say we decided to go for the third alternative, but instead of copying other tools in the market, we decided to avoid over-analysing other products features and start from first principles, and that is what we did.

In order to create the tool architecture, we took a physics approach mixed with engineering, which led us to start from the fundamentals, the building blocks of a chaos engineering tool, with the minimum requirements to be called a chaos engineering tool, but with a pragmatic and condensed set of features, based on basic probability and PRNG.

The next step was the technology to use, the choices were, Python, Golang and C, but since we needed something easily portable, with fast prototyping, a plethora of extensions, an OCI SDK, and speed was not a concern since we would be at the mercy of the network anyways, henceforth we decided on Python.

The beauty of the tool is in its simplicity, as Da Vinci once said, simplicity is the ultimate sophistication, and the fact that it was a cloud native from first principles tool, has made it extremely simple, to understand, to use, and also to extend and modify.

Method

The fundamental mechanism by which the madbull works is based on event probability, and could be compared to a dice roll[1], with a given periodicity and with the special case that the possible values go from 0 to 99. The output is then compared with the pre-configured willingness of the madbull to determine if the chaos starts or not.

In this case the probability comes from a random deterministic approach, a MT PRNG is used to generate the values to be used[2].

The point of application for this tool go from the *proof of concept* stage to the *production* stage; we have successfully let the madbull run in autonomous form in POC stage and Testing environments, and it proved successful to find weak spots we were not aware of.

The madbull does not produce **failure injection testing**, but chaos engineering, meaning that it is broader and the blast radius is larger; that being said, it is possible to limit the scope in several ways, in order to minimise the blast radius and concentrate solely on one specific target, making traceability easier.

We have separated the different scopes of the system as follows:

Compartment / Subcompartment level chaos

The tool allows to centralise the chaos in a given compartment within OCI, and in the future will also filter by subcompartment, allowing to create chaos in e.g. a staging environment within a given product, at this level, this is not as fine-grained as can go, but it is an auspicious start.

Virtual Network level chaos

It has been found that, one of the things the industry frequently does is separate different departments[3] and also environments at a network level; based on this knowledge, there is value in unleashing the madbull in a given Virtual Cloud Network.

In this case there is no need to target a compartment, subcompartment or alike, those resources interconnected will be targetted irregardless of other factors.

Regular expression filtering

It has also been found that although the options are open for using tags, the industry still falls back to use naming convention for cloud resources as a descriptive term, adding suffixes and prefixes for the idenfication of environment, location, product or service, roles, so on and so forth[4].

With this in mind, the decision was to add regex support when defining scope, henceforth, having the possibility to reduce the blast radius accordingly, which it is believed to be a good proposition.

Compute types

The flexibility allows for scoping only those compute resources of a given type or subtype, whether that is Virtual Machines or Bare Metal of any of the available shapes[5].

Naming

Being able to scope by naming, subnaming or patterns within the name is something relied on extensibly, especially since naming conventions still fall back to name, e.g. database servers as core_db_01, core_db_02, slave_db_01, etc. analogously for other resources.

Agressiveness level

This is where things goes out of the conventional and become more interesting, the madbull is thought of it as a semi-living entity, which can also be moody, to that extend, it has a given gameday personality, and the personality will be defined when the gameday is planned[6].

Especially important is that the personality should not be left at random, since the havoc might be unexpected and that is a violation of the chaos engineering principles[7].

The madbull mood supports 5 levels, it presents with several options without falling in the paradox of choice[8].

The levels are mild, fair, hard, chaos and mayhem, but the most interesting aspect about these, is that the thresholds that define the levels can be configured, so the chaos engineer can define, what mild, fair, hard, chaos and mayhem mean for that particular case or that particular game day.

Parallelism level

Although in a chaos experiment it is best practice to only change one variable at a time, in some cases altering one variable involves modifying the state of more than one physical/virtual resource, as the ratio variable:resource is not 1:1. For these cases, the madbull offers the possibility of configuring how many resources will be altered every time the madbull attacks.

There are three levels of parallelism that in turn also carry thresholds defining the number of resources per level, which makes the tool flexible.

Needless to say that parallelism can also be advantageous when is necessary to find out what happens when a system has not one, not two, but multiple failures, which is the ultimate chaotic situation, but a possibility nonetheless.

This parallelism can allow for the madbull to bring down an entire Failure Domain, AD or even more at one time if needed; thus, not recommended unless strictly necessary.

Priority multiplier

Using a completely aleatory selection was not pertinent unless specific cases, so the option was for having an alternative which is called, the priority multiplier. This allows to define weights for different compute resources.

As an example, it is possible to define BateMetal resources to be 3 times more frequently selected than Virtual Machines or viceversa, and this is done by accounting the total of resources, creating equiprobable events that have the possibility to be steered towards either one or the other in binary form, but that in the end, will eventually be distributed among them.

Dry runs

As a classical example of functionality, a requirement was to have the possibility of dry-runs, which allowed engineers to create scenarios without the actual events happening. This led to a better preparation of game days and was also shown to offer insight over things that could be happening at the same time; which in turn led to revisions in the architecture and also added documentation in the form of SOPs to cover unexpected cases.

Labels

Although as mentioned, part of the industry still relies on naming conventions, a wider spectrum is relying on tagging[9][10], thus it was a necessary feature in order to allow for more flexibility; this is something currently being worked on but will be supported in future versions as it is on the roadmap.

Results and discussion

The madbull keeps a log of events that record what is doing, at what time is doing it and how long it takes for the action to go through; although it does not record completion, i.e. shutting down an instance and the actual time the instance is in stopped state, the command may take in the order of milliseconds but the finalisation and cleanup, may take in the order of seconds.

An important case is that currently it does not terminate resources, but stops them, making possible the rollback in case of unexpected consequences.

In regards to parallelism, the suggestion is to use it, only when the rest of the single failure cases have been exhausted, unless strictly necessary and convenient as this can be a dangerous feature, beware.

Conclusion

- The madbull has been proven useful in stages prior to production, we needed a tool to validate architecture design, POC stages, HA, DR and FO solutions, and environments from labs or UAT to Staging, and this is it.
- The application of chaos engineering is not exclusive of a production environment as it has been fundamentally suggested, the application can provide useful insights before that stage and decrease the cost caused by improperly designed architectures or architectures that are not fit for purpose.
- Another case is validation of architecture design to deployments, when the madbull can be used to validate the architecture and gather a sign-off, and then re-used once the architecture has been deployed, to assert that it has been deployed following the guidelines and original design; if the madbull finds a failure at this stage, there is a probability that the guidelines were not followed or something was missing during the transition.
- Although the basic blocks are based on Oracle Cloud Infrastructure, it can be easily ported to other clouds and even datacenter infrastructure.
- We decided on a minimalist approach since it eases adoption and also ensures that maintaining the tool will be much simpler.
- We consider the madbull to be a stepping stone to start doing FIT.

References

[1] *Dice roll probability by Math Worlfram.*

[2] *M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator".*

[3] *Virtual Cloud Network Oracle FAQ.*

[4] *Microsoft azure architecture best practices on naming conventions.*

[5] *Oracle Cloud Infrastructure compute features.*

[6] *Gremlin tutorial on how to run a gameday.*

[7] *Principles of chaos website.*

[8] *The paradox of choice by Barry Schwartz.*

[9] *Amazon Web Services tagging.*

[10] *Oracle Cloud Infrastructure freeform and defined tags.*