Chaos Engineering on Kubernetes

Linares, Bryan
California State University, Long Beach
Bryan.Linares01@student.csulb.edu

Patel, Rutvi
California State University, Long Beach
rutvi.patel@student.csulb.edu

ABSTRACT

Guaranteeing the resilience of a large distributed system is difficult. We survey the theory and practice of Chaos Engineering in Distributed Systems that use Kubernetes. Chaos Engineering is a formalized and disciplined approach to test a distributed system's resilience under realistic and unexpected conditions. It extends and overtakes the long standing practice of simple fault injection through a more thorough process based on the scientific method with more sophisticated tools. We aim to introduce Chaos Engineering in practice as it is used alongside Kubernetes, the popular and leading container orchestration management service and platform. In our research, the solution frameworks surveyed are validated to implement the formal Chaos Engineering theory steps practically. Results are evaluated in their synthesis of the theory and tools, including their incorporation of Kubernetes and their usability. Chaos Engineering is concluded to be highly useful and flexible, with a range of possible improvements to a Distributed System's operation and resilience, as well as a strong foundation for future testing.

1. INTRODUCTION

Chaos Engineering as a defined Distributed Systems practice was brought to prominence by the needs of Netflix. Due to the size and enormous amount of data in their system, they needed a way to form fixes in catastrophic conditions. They first defined Chaos Engineering as: the discipline of experimenting on a distributed system in order to build confidence in its capability to withstand turbulent conditions in production. [1] It was a necessary way to formally address issues unique to large distributed systems. The services provided have complex behavior and failure modes, due to their large spread of computation and data transfer across many virtual nodes and containers of services. Unexpected errors and turbulence are accepted as inevitable (power outages), and system development with resilience is the best way to combat this fact. Software designers and engineers must assure themselves that their system will be able to recover under real conditions.

1.1 Chaos Monkey

Chaos Monkey was the first tool Netflix put into practice that proved the benefit of a formalized approach to more rigorous testing of distributed systems. It had the capability to randomly terminate virtual machine instances that host important points of their service. As an example Netflix would begin a testing round or "Game Day" through a hypothesis that their system was resilient to a server crash running an application node. The experiment would run random crashes, and to verify resilience, the number of video streams that Netflix was serving would be measured to have no change affected.

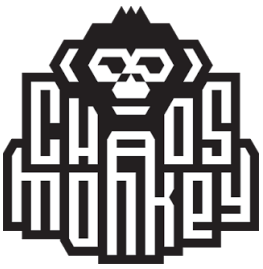

Figure 1: Chaos Monkey logo

2. BACKGROUND

2.1 Container Orchestration using Kubernetes

Kubernetes is a modern and extensible platform for managing and creating deployments of groups, or clusters, of application Nodes. They have in them one or more containers and each has its own private applications and necessary files, or they can allow intercommunication as needed. A cluster contains multiple Nodes which contain the pods which have the containers within. Nodes are the actual servers or Virtual Machines that do the work to run the containerized applications.

This is an efficient and widely adaptable model for distributed system creation. Chaos Engineering tools can take full advantage of the entry points and management tools Kubernetes allows, by reading and writing the variables that it will use to define the steady state at a macro level and using those same values later to log results and measure normalcy, from outside of the entire system that Kubernetes manages itself.

3. CHAOS ENGINEERING THEORY

The researched theory that was accepted as de facto standard grew from the steps laid out by Netflix's initial formulation. [2] The key idea is facilitation in program design, to choose program variables that have meaning when measured or output to a tester's monitoring.

Chaos Engineering at its core, proposes that the uncertainty and weakness of a distributed system can be uncovered through 4 major steps, with additional ones based on practical needs:

- Defining the measurable steady state, which defines a measurable output that indicates "normalcy" of operation
- Hypothesize that this steady state will continue in the control and experiment.
- Introduce real world adjacent variables and changes to potentially break the steady state
- Attempt to disprove the hypothesis by detecting differences in the steady state values in control and experiment.

3.1 Steady State

The Steady State behavior of a distributed system is the measurable formulation of how the system is supposed to behave. The values have to be able to be captured and measured. The developer should denote under which values of circumstances operation is kept at normal operation, and at which boundaries are risk issues raised. There should be a tendency on verification over validation, in systems engineering terms, this means to focus on having the system meet defined specification requirements (verification) rather than a customer's experienced requirements (validation). The figure 3 is a good example, because the terms are all measurable values with a protocol that is easy to write and receive with well defined additional values as needed.

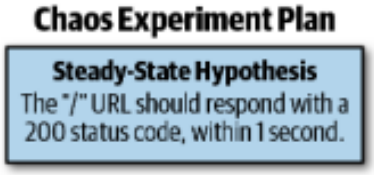

Figure 2: An example steady state definition

3.2 Hypothesis

The hypothesis should be as specific as possible to the values that they are measuring for experimenting, and the break in the steady state behavior that they would affect. For instance, one class of hypothesis would measure specific values of certain nodes, that will ensure that the "customer" is able to use a service. This hypothesis would be under an availability category. Another hypothesis would look at other variables that lead to a break in the steady state of the system's security.

3.3 Experiment Design

The experiment design step requires the developer to choose an experiment that will output meaningful data that is specific to the hypothesis being targeted. For example, say that the experiment chosen is to terminate an instance of a node. Then as other experiments, the engineer chooses to fill up the memory, fill up the disk space, and turn off the connection to an

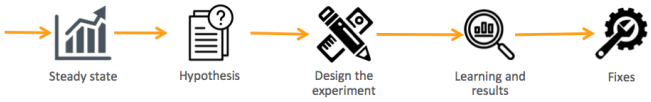

Figure 3: General Theory steps

instance. From a measuring point of view these can all cause a similar effect of causing the instance to stop responding, which does not give any useful new information. Unless the right values are measured in the experiment.

3.4 Learning and Results

This crucial step involves logging and recording what is found during a test. Logs should be kept as accurately and with as much metadata and timing data as useful. Analysis documents can be written and also automated and compared against other logs of a similar nature, if any learning can be gained from doing so.

3.5 Applying Fixes

Chaos Engineering should not just break services without proper planning and fixes. Some experiments, for example, may be too large, and destroy too much of the system at once beyond being able to give any useful information. So setting conditions and abort conditions for the system to fix itself, have to be done in steps that gradually increase in the amount of affected nodes or services. This is sometimes called being mindful of the "blast radius" and keeping the target radius small and manageable ensures that the testing engineer or team can accurately target code or services to fix. Kubernetes has a large library of workloads that operate on pods, and has extensive documentation on how certain disruptions affect their Pods (for example an emergency shutdown versus a scheduled upgrade both disabling a node) which have to be referenced properly no matter which tool is being employed.

4. METHOD

We surveyed Chaos Engineering in practice on the Kubernetes platform using available tools. The practical reality that the tools choose to implement the theory vary in their extensibility and interoperability and portability.

4.1 Litmus Chaos

One of the powerful tool often used to perform Chaos Engineering in industry is Litmus. Litmus is an open-source Chaos Engineering platform that enables teams to identify weaknesses and potential outages in infrastructures by inducing chaos tests in a controlled way. Some of the features provided by this tool includes Probes that can be used to define the steady state of the system, Chaos Experiments that can be used to simulate real-world turbulence and Chaos hub to leverage common prebuilt tests to experiment on new environments. These features are relative to the 4 major steps of Chaos Engineering. This tool is highly extensible and integrates with other tools to enable the creation of custom experiments. For example it has options for integration

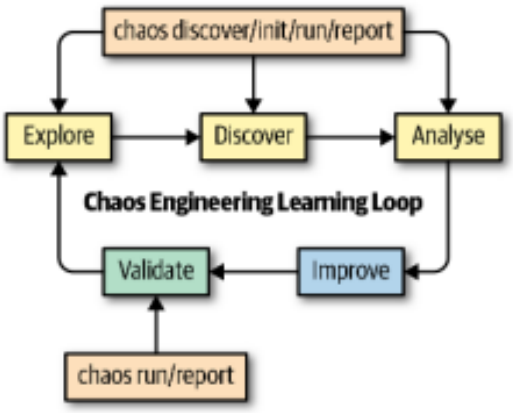

Figure 4: Toolkit Learning Loop

with GCP, AWS and Azure Kubernetes clusters along with Grafana, Prometheus and Github setups.

4.2 Chaos Toolkit

Chaos Toolkit is touted as a free and open-source toolkit made with developers in mind. Chaos Engineering experiments are written and stored in a lightweight JSON or YAML file. It is primarily a command line interface with a simple loop of operation. 4

Toolkit is written in Python and to work with services like Kubernetes or AWS that expose distributed containers, it requires the usage of community drivers that wrap the toolkit's function in compatible forms. The drivers create the bridging functions that expose the values from or to the chosen service provider. In the learning loop, the chaos program is called to report values as specified in a JSON or YAML file.

4.2.1 Running an Experiment

The system we tested was a simple service cluster of 3 nodes. The small app is duplicated among all three and outputs a simple HTML page using CherryPy, which is written in Python. The protocol and values to measure are designated in the experiment file. The name of the experiment should suggest the Hypothesis being run. The steady state is define within the values that are being checked. This is labeled as a probe, and use functions defined within the Toolkit's library and the driver's as needed. Probes can also call Kubernetes functions directly on the nodes as specified. If all the probe checks return an ok or "pass" value, then there is said to be no deviation. If any probe does not pass the measurement checks, then Toolkit reports a deviation and error.

The toolkit automatically attempts to run any rollbacks defined. In our demo any drained nodes are simply undone and made available again. The toolkit checks for correctness of the JSON file, ensures the Steady State Hypothesis once before testing, then runs Experiments in probes, and actions, then runs the steady state check again. The analysis and interpretation of the results is left to the engineer, and in our case a Kubernetes Disruption Budget is applied to the system, which blocks and limits nodes from being drained in the way that is attempted in the experiment.

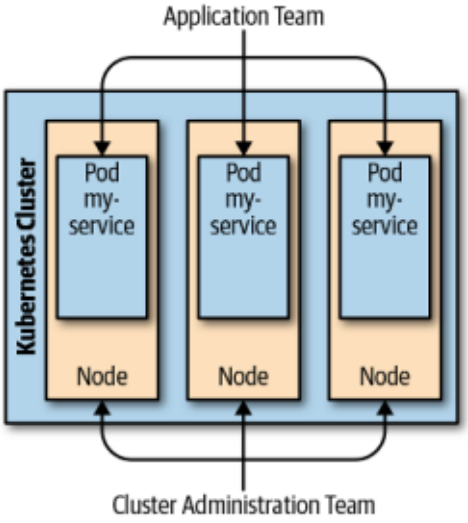

Figure 5: The Chaos Toolkit target system

4.3 Other Tools Available

Numerous other tools also exist, the most popular of which is Gremlin. Gremlin advertises as a fully-featured suite of tools that can integrate with other popular services like Jira. Gremlin's largest selling point is a vast library of pre-made packages to simulate a wide breadth of possible experiments, such as CPU spikes, or traffic spikes, memory filling, all in a way that is requires minimal additional code from the developer. Regardless, many companies elect to use their own tools internally, which adopt similar philosophies.

5. DISCUSSION

There is a large variance in the cost, philosophies and feature-sets of the tools available. Chaos Toolkit in particular resembles more of a flexible framework to code around than a solution that is ready to go and use out-of-the-box. Chaos experiments are done in four simple steps: defining the steady state, hypothesizing on it, simulating real turbulence, and proving or disproving the hypothesis. The framework of theory creates a wide-open interpretation in how tools choose to provide value to a company. There is a significant overlap in the amount of combined coding and software engineering in the value proposed by these different tools.

5.1 Benefits

Chaos Toolkit gains the most from its open-source nature and community-based discovery and extension tooling. It gives the developer almost complete control over experiments, due to features like being able to run arbitrary python code in response to triggers. The native rollback features provide an easy step to add fix application, given that the developer is familiar with the needed Kubernetes features also in a command line form. Since the toolkit is open source, community-designed experiments are searchable and recommendable. Logging and reporting are also done automatically

and in similar highly compatible open source formats like JSON.

5.2 Risks

The negatives to using Chaos Toolkit arise when looking for features involving portability and widespread automation. There is no native scheduling feature for tests, so this would have to be done using another tool by a team, and systems with too much dissimilarity (for example, some on Kubernetes and some on AWS) would need multiple drivers and harder coding. The experiments are thus very hands on, and must be coded into files one by one, and their specific drivers make them unable to be reused when moving to other systems.

6. CONCLUSION

Adopting the practice of Chaos Engineering proves itself invaluable and easily adopted quickly. Even in a small system, it can be shown to be useful, and is already proven in the largest distributed systems in the world. Kubernetes provides a strong backbone to allow these chaos engineering tools to make for the most reliable testing to create highly resilient systems. They successfully adopt the model proposed by the earlier theory and interpret it in a practical sense.

6.1 Future Outlook

The current state of available tools covers a wide breadth of use-cases and software design paradigms. There is no one tool that can be applied to all systems. Hesitation in applying Chaos Engineering commonly comes from industries that are not interested in risking outages or losses. Companies that traditionally would employ techniques such as redundancy or just reducing points of failure in a costly way. Financial and Healthcare companies are typical examples, that resist simulating risks to loss of money or crucial human healthcare data, but they experience outages the same as others and in practice do benefit. [3] The form of the ideas behind Chaos Engineering can be said to be similar to clinical trials and applied in the same way. Designing Distributed Systems with experimentation forward and exposing measurable variables of normal behavior is a powerful practice and gives developers using Chaos Engineering the tools to strategize fixes. Newer industries like decentralized FinTech, Machine Learning, Autonomous Vehicles, and the new generation of Aviation are future grounds to test the limits of Chaos Engineering techniques due to their heavy intercommunication demand. The principles will continue to evolve and new tools will emerge to meet them.

REFERENCES

[1] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, "Chaos engineering," *IEEE Software*, vol. 33, no. 03, pp. 35–41, may 2016.

[2] A. Basiri, "Principles of chaos engineering," 2016. [Online]. Available: https://principlesofchaos.org/

[3] C. Rosenthal and N. Jones, *Chaos Engineering: System Resiliency in Practice*. O'Reilly Media, 2020. [Online]. Available: https://books.google.com/books?id=UxuFxAEACAAJ