Chaos Engineering

Cole Sturza, Alex Book

Building Confidence Through Testing (The Traditional Approach)

Unit Tests:

Ensure that a section of an application (unit) meets its design and behaves as intended.

Integration Tests:

Ensure that various units work as intended when they are integrated together.

Is this enough coverage?

What is Chaos Engineering

Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.

"Principles of Chaos Engineering". principlesofchaos.org. Retrieved 2017-10-21.

Break your systems on purpose to find their weaknesses and avoid catastrophic failure in production.

Concept

Resilience: tolerate failures while still ensuring adequate quality of service.

Fault Injection: A technique for deliberately introducing stress or failure into a system in order to see how the system responds.

Chaos Engineering is the strategy of using *fault injection* to accomplish the goal of more *resilient* systems.

Chaos engineering can be used to achieve resilience against:

- Infrastructure failures
- Network failures
- Application failures

History

The practice was initially pioneered by Netflix.

The intent was to move from a development model that assumed no breakdowns to a model where breakdowns were considered to be inevitable.

Motivates developers to consider built-in resilience to be an obligation rather than an option.

Allows for the testing of redundant architecture.

"The Netflix Simian Army". Netflix Tech Blog. Medium. 2011-07-19. Retrieved 2017-10-21.

Chaos in Practice

Chaos Engineering experiments generally follow four steps:

1. Start by defining ‘steady state’ as some measurable output of a system that indicates normal behavior.
2. Hypothesize that this steady state will continue in both the control group and the experimental group.
3. Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.
4. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

The harder it is to disrupt the steady state, the more confidence we have in the behavior of the system.

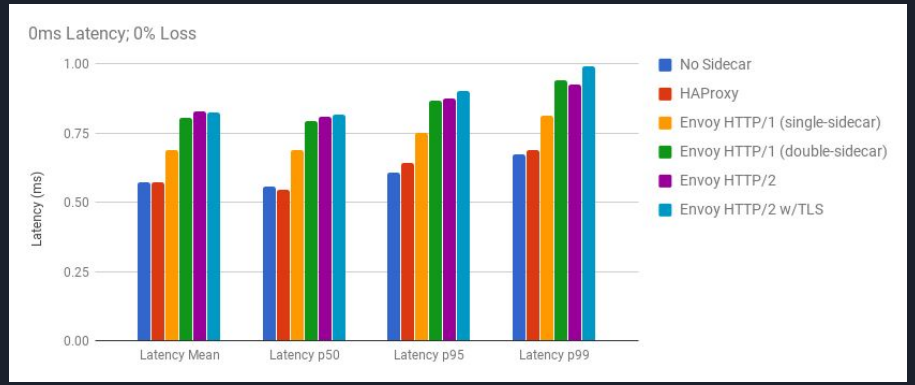
"Principles of Chaos Engineering". principlesofchaos.org. Retrieved 2017-10-21.

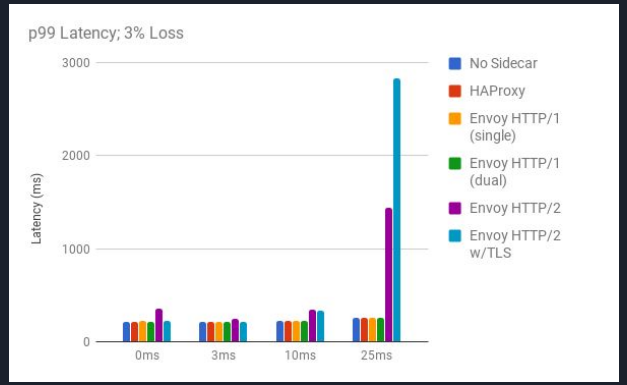
Case Study: Twilio

Wanted to test the benefits of upgrading their services to communicate with HTTP/2.

Used chaos engineering to add varying combinations of network latency and packet loss to their services.

Ex: 0ms/10ms/25ms network latency added; 0%/1%/3% packet loss added.

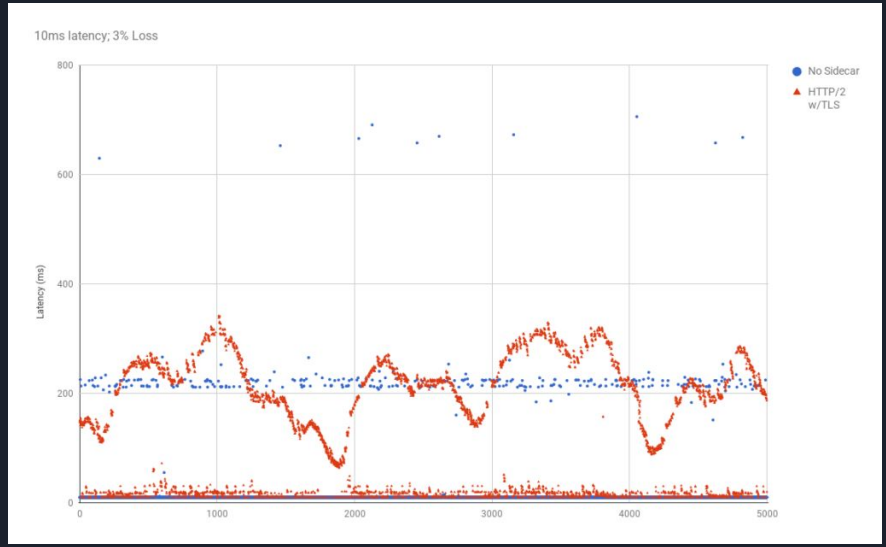




"Discovering Issues with HTTP/2 via Chaos Testing". https://www.twilio.com/blog/2017/10/http2-issues.html.

Case Study: Twilio

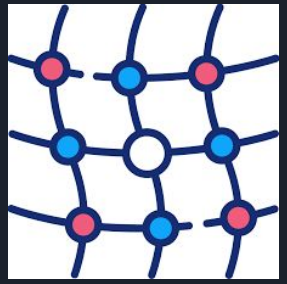
Through chaos testing, Twilio was able to identify architectural tradeoffs in HTTP/2 that cause it to perform worse than HTTP/1 when there is packet loss in the network.

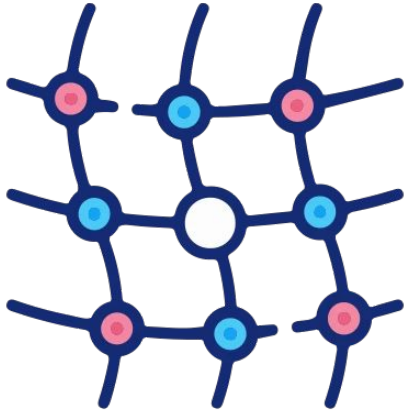


An overlay comparing HTTP/1 vs HTTP/2, when there is packet loss.

"Discovering Issues with HTTP/2 via Chaos Testing". https://www.twilio.com/blog/2017/10/http2-issues.html.

Some Popular Chaos Engineering Tools





Chaos Mesh

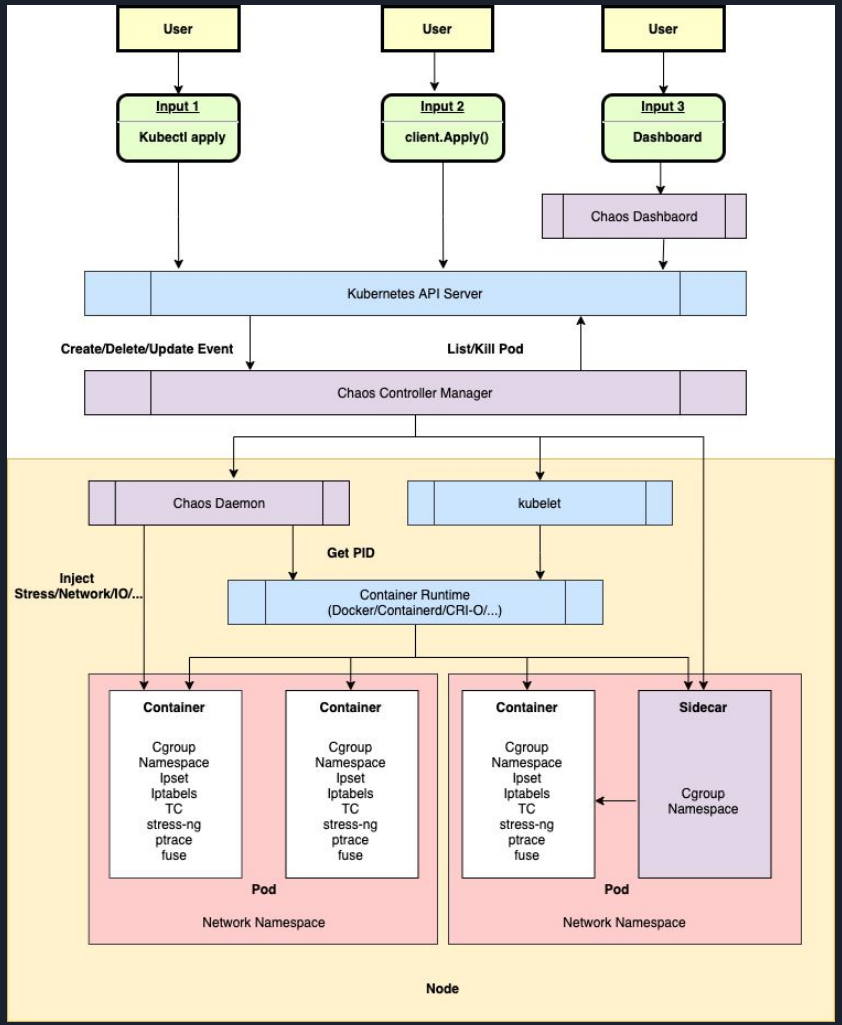
Chaos Mesh Overview

Chaos Mesh is an open source cloud-native Chaos Engineering platform.

Offers various types of fault simulation and has an enormous capability to orchestrate fault scenarios.

Chaos Mesh primarily contains three components:

- Chaos Dashboard: The visualization component of Chaos Mesh.
  - Provides a GUI where users manipulate and observe Chaos experiments.
- Chaos Controller Manager: The core logical component of Chaos Mesh.
  - Primarily responsible for the scheduling and management of Chaos experiments.
- Chaos Daemon: The main executive component.
  - Interferes with network devices, file systems, and kernels (“hacks” into target Pods).


User
User
User
Input 1
Kubectl apply
Input 2
client.Apply()
Input 3
Dashboard
Chaos Dashbaord
Kubernetes API Server
Create/Delete/Update Event
List/Kill Pod
Chaos Controller Manager
Chaos Daemon
kubelet
Get PID
Inject
Stress/Network/IO/...
Container Runtime
(Docker/Containerd/CRI-O/...)
Container
Cgroup
Namespace
Ipset
Iptabels
TC
stress-ng
ptrace
fuse
Container
Cgroup
Namespace
Ipset
Iptabels
TC
stress-ng
ptrace
fuse
Pod
Network Namespace
Container
Cgroup
Namespace
Ipset
Iptabels
TC
stress-ng
ptrace
fuse
Sidecar
Cgroup
Namespace
Pod
Network Namespace
Node

Fault Injection

Chaos Mesh offers three types of fault injection, each of which has various sub-types:

- Basic resource faults
  - PodChaos
  - NetworkChaos
  - DNSChaos
  - HTTPChaos
  - StressChaos
  - TimeChaos
  - KernelChaos
- Platform faults
  - AWSChaos
  - GCPChaos
- Application faults
  - JVMChaos

Chaos Workflows

Chaos workflows enable you to perform a series of Chaos experiments, keep expanding the explosion radius (including the scope of attacks), and increase the failure types.

After running a workflow, you can see the status of the application on the dashboard.

Supports serial and parallel experiments, pausing mid-workflow, using either YAML files and web UI to define and manage workflows.

Defining and Running Chaos Experiment(s)

Much like with workflows, Chaos Mesh allows for the definition of experiments through either YAML files or the web interface.

As long as your Kubernetes cluster is up and running as expected, this process requires only defining the experiment, then applying it (through the web interface or command line).

Pod Chaos Experiment Example

- action
  - Specifies fault injection type (pod failure, pod kill, container kill)
- mode
  - Specifies mode of the experiment from the following options:
    - one (selecting a random pod)
    - all (all eligible pods)
    - fixed (certain number of eligible pods)
    - fixed-percent (specific percentage of eligible pods)
- value (not shown, optional)
  - Provides parameters for the ‘mode’ config
- selector
  - Specifies the target pod
  - Can be specified through namespace, labels, annotations, field, phase (pending, running, succeeded, etc.), node lists
- containerNames (not shown, optional)
  - When the injection is container-related, this must be specified
- gracePeriod (not shown, optional)
  - Specifies the duration passed before chaos ensues
- duration (not shown , required only for failures)
  - Specifies the duration of the experiment

```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: PodChaos
metadata:
  name: pod-kill-example
  namespace: chaos-testing
spec:
  action: pod-kill
  mode: one
  selector:
    namespaces:
      - tidb-cluster-demo
    labelSelectors:
      'app.kubernetes.io/component': 'tikv'
```

```
kubectl apply -f ./pod-kill.yaml
```

Network Chaos Experiment Example

- action
  - Specifies fault injection type (network delay, packet loss, packet duplication, network partition, bandwidth limit)
- direction (not shown, used for packet-related experiments)
  - Indicates the direction of target packets (from, to, both)
- target (not shown , works in conjunction with ‘direction’)
- Additional parameters for ‘delay’
  - latency
    - Specifies the network latency to be injected
  - correlation
    - Indicates the correlation between current latency and previous one
  - jitter
    - Indicates the range of network latency
  - Together, correlation and jitter are used to calculate the delay of the current packet (formula can be found in Chaos Mesh documentation)
- Sensible options from Pod Chaos experiments are also available or required (mode, value, selector, containerNames, gracePeriod, duration)

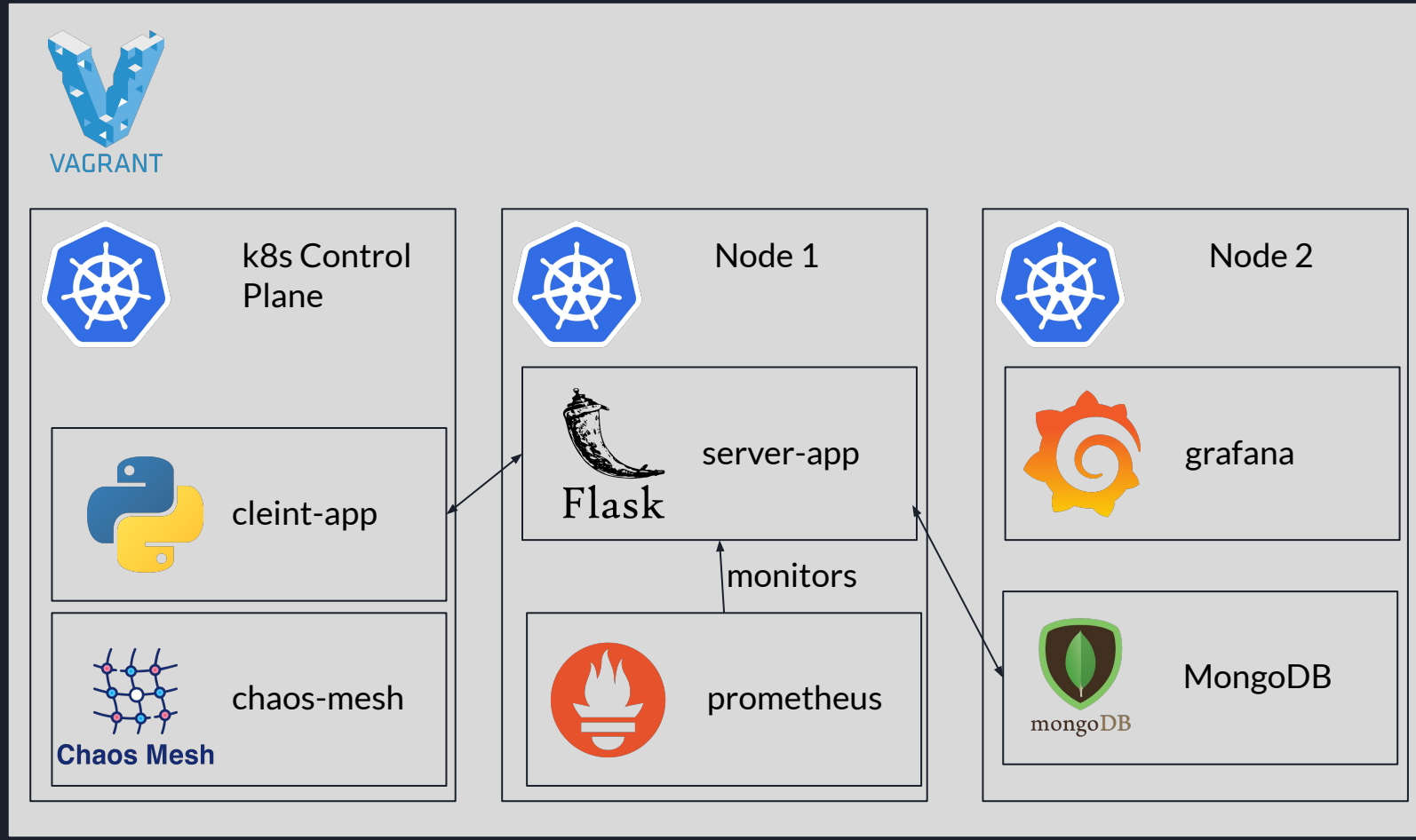
```yaml
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: delay
spec:
  action: delay
  mode: one
  selector:
    namespaces:
      - default
    labelSelectors:
      'app': 'web-show'
  delay:
    latency: '10ms'
    correlation: '100'
    jitter: '0ms'
```

```
kubectl apply -f ./network-delay.yaml
```

Viewing/Analyzing the Results of Chaos Experiment(s)

Once an experiment has run its course (or is stopped), Chaos Mesh restores all target pods to their pre-experiment state.

The results of experiments can be viewed through the Chaos Dashboard (through the web UI) or through the command line using ‘kubectl’.


VAGRANT
k8s Control Plane
cleint-app
chaos-mesh
Chaos Mesh
Node 1
Flask
server-app
monitors
prometheus
Node 2
grafana
mongoDB
MongoDB

Demo