Building a more robust Apache APISIX Ingress controller with Litmus Chaos

Jintao Zhang

Who am I

- I'm Jintao
- Apache APISIX PMC
- Kubernetes Ingress-NGINX maintainer
- Microsoft MVP
- https://github.com/tao12345666333
- email: zhangjintao@apache.org

Agenda

- Why We Need Chaos Engineering
- How to design Chaos experiments for an Ingress controller
- How to practice
- Benefits and Future

Why We Need Chaos Engineering

What is Chaos Engineering

> "Chaos engineering is the discipline of experimenting on a software system in production to build confidence in the system's capability to withstand turbulent and unexpected conditions”

— https://principlesofchaos.org/

What is chaos engineering

Introduction of disruptive events

A. Types of Disruptive Events:

Chaos engineering can involve a variety of disruptive events, including network partitions, service degradation, and resource constraints.

B. Purpose of Introducing Disruptive Events:

Determine the impact on the system and identify any vulnerabilities or weaknesses.

Testing system resilience

A. Importance of Testing System Resilience in Real-World Scenarios:

This can help ensure that systems are robust and scalable, and can withstand unexpected challenges and conditions.

B. Methods for Measuring the Impact of Disruptive Events on System Resilience:

Including monitoring system logs, performance metrics, and user experience etc.

Discovering hidden problems

A. Common Hidden Problems in Distributed Systems:

Including data loss, performance bottlenecks, and communication errors etc.

B. Advantages of Discovering Hidden Problems Before They Impact Production:

It can help prevent downtime, reduce the risk of data loss, and ensure the system continues to operate smoothly.

What it's worth and why we need it

Distributed systems are complex

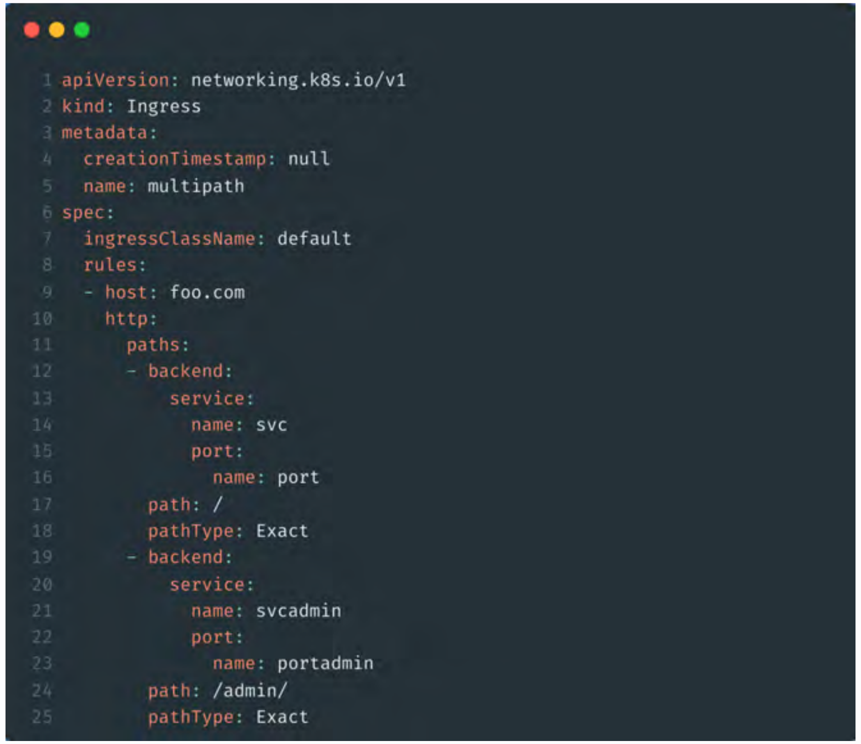
No confidence without testing

Mistakes cost time and lost customers

How to design Chaos experiments for an Ingress controller

What is Ingress

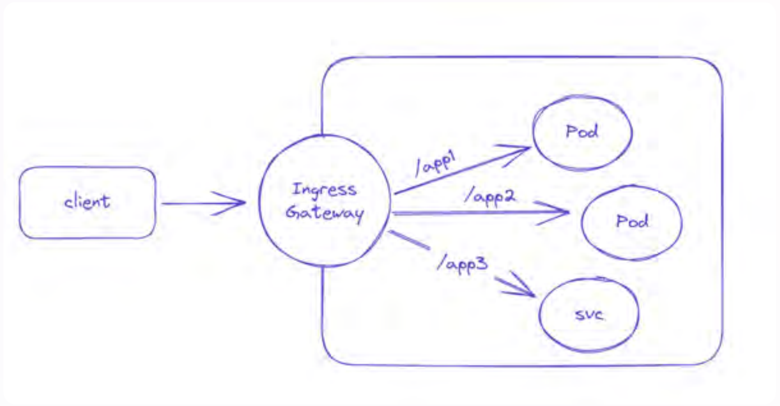
- A resource object
- Contains routing rules



```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  creationTimestamp: null
  name: multipath
spec:
  ingressClassName: default
  rules:
  - host: foo.com
    http:
      paths:
      - backend:
          service:
            name: svc
            port:
              name: port
        path: /
        pathType: Exact
      - backend:
          service:
            name: svcadmin
            port:
              name: portadmin
        path: /admin/
        pathType: Exact
```

What is Ingress controller

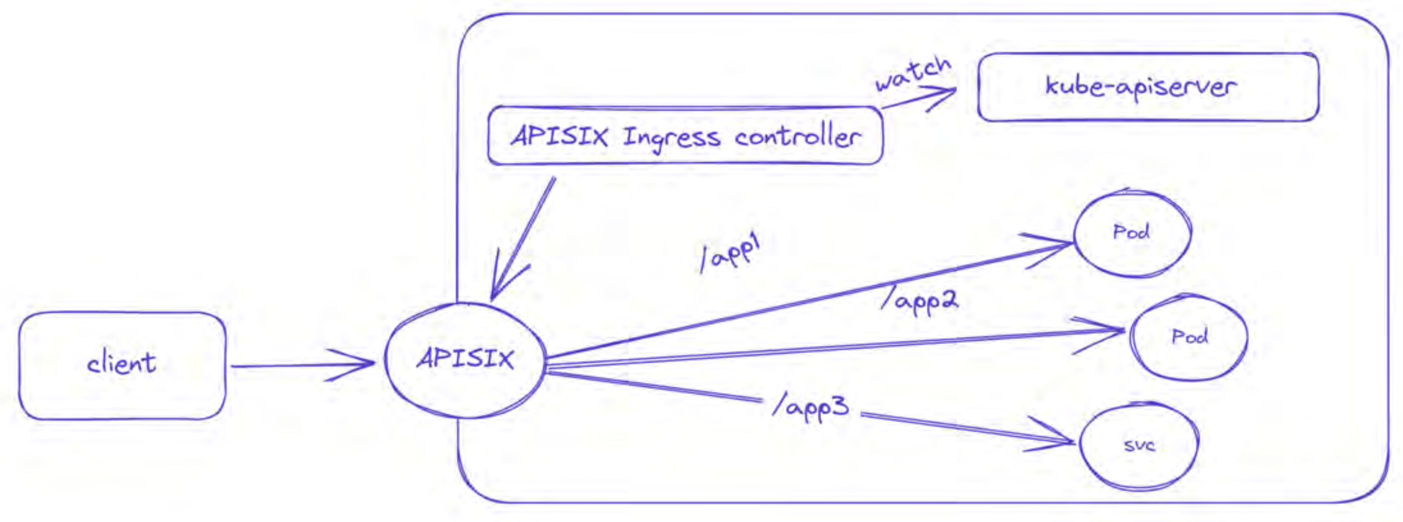
- Watching Ingress resources and translate to actual proxy configuration
- Extended Ingress Semantics
- Support other routing rule definitions (Gateway API, CRD)



What is Apache APISIX Ingress

- Using Apache APISIX as the data plane, providing nearly 100 built-in plugins
- Support three configuration methods
  - Ingress
  - Gateway API
  - CRD
- Separation of control plane and data plane provides flexible deployment methods and reduces security issues
- Supports integration with external service discovery components

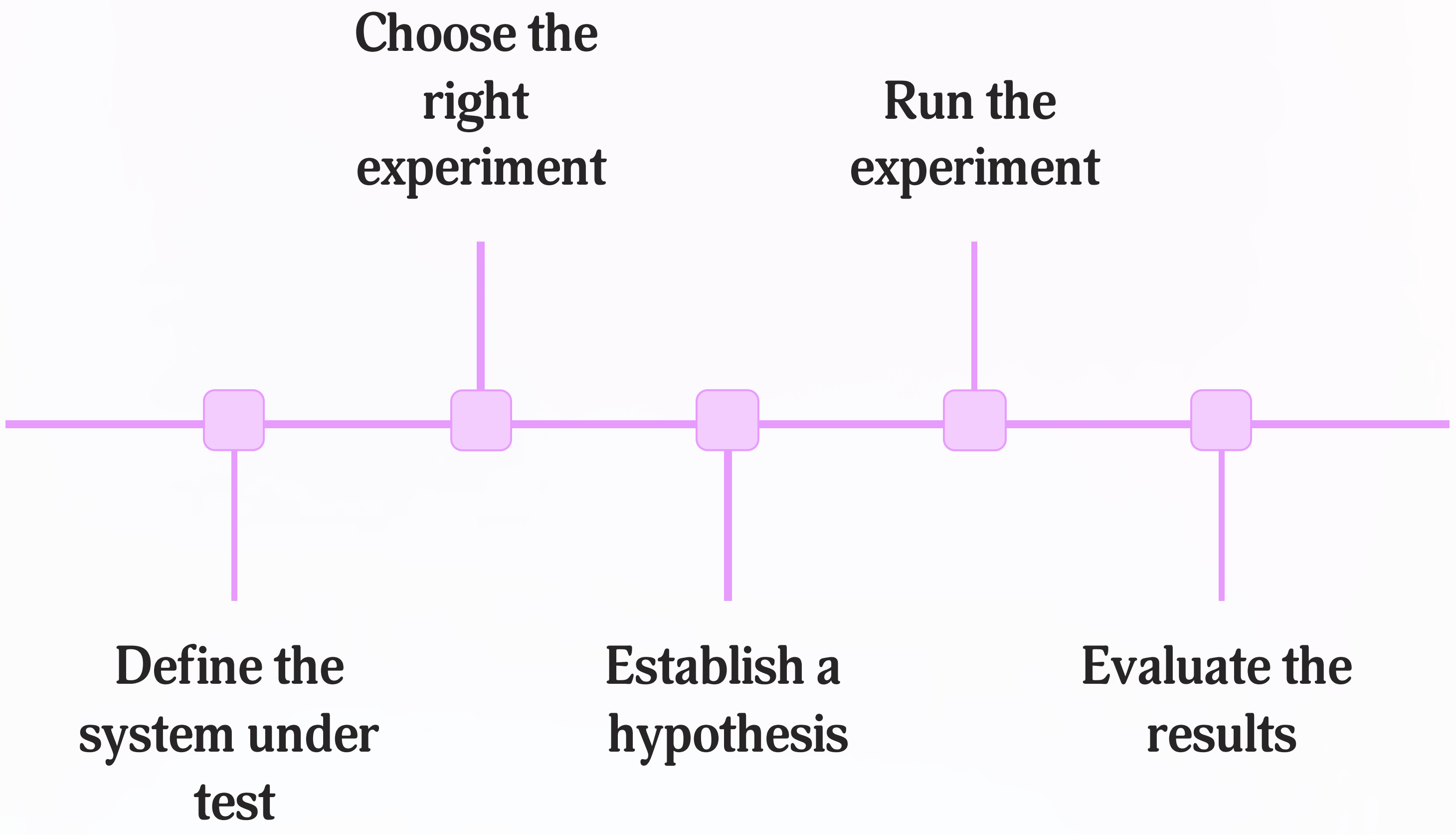
What is Apache APISIX Ingress



What is LitmusChaos

- Open Source Chaos Engineering platform
- CNCF Incubating Project
- Built-in basic functions required for various experiments
- Chaos Observability

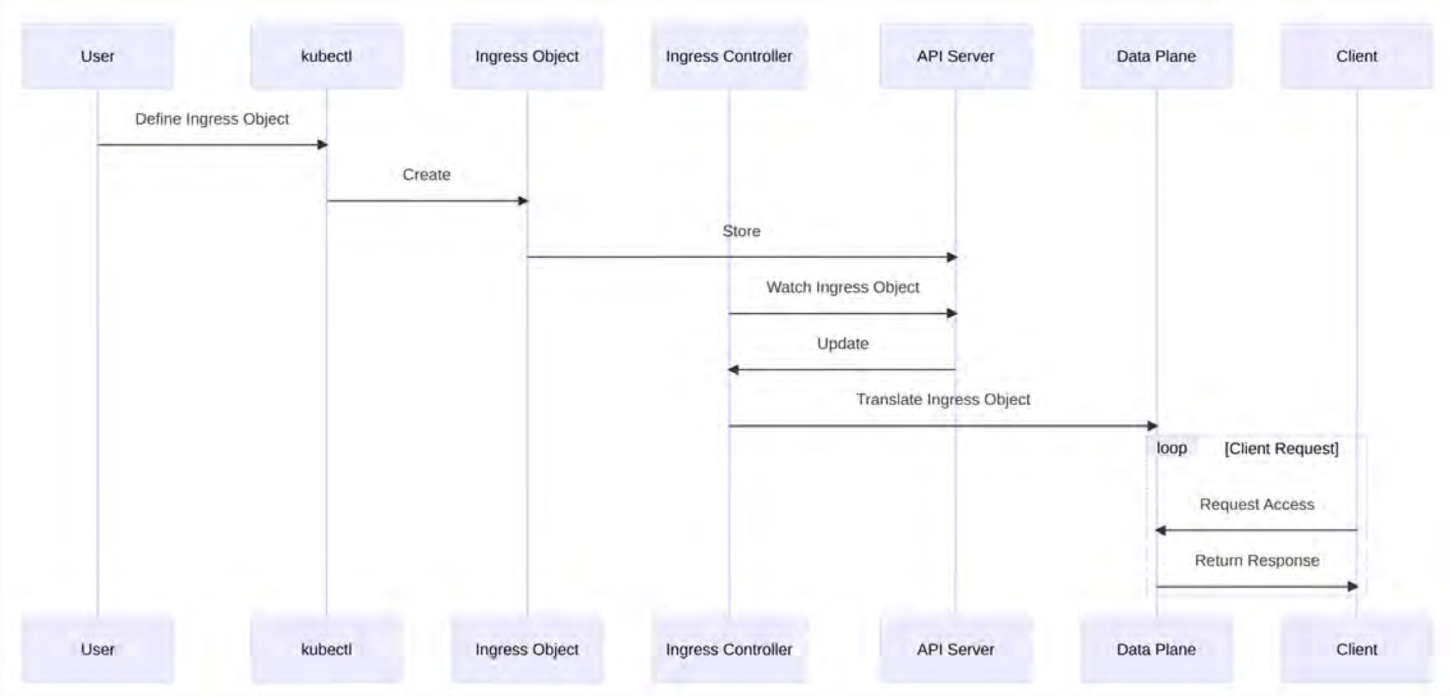
How to design Chaos experiments



Ingress Controller usage scenarios

- **Proxy traffic**
- **Proxy traffic**
- **Proxy traffic**
- And some other functions

Define the system under test



- kube-apiserver: if an exception occurs, the Ingress resource write failed.
- Ingress-controller: Network interruption, Crash, Pod faults, I/O
- data-plane: Network interruption, Crash, Pod faults, I/O

Choose the right experiment

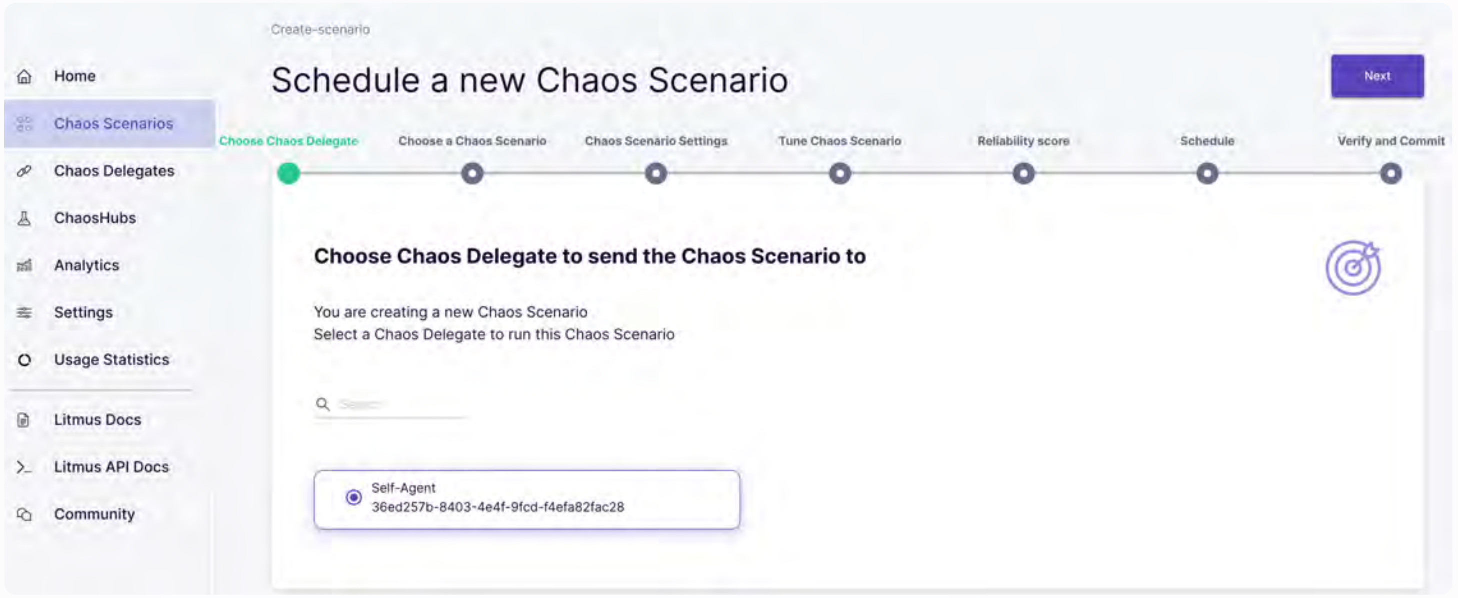
- Exception configuration already covered by e2e
- Enter the system **after the configuration is written successfully**
- So only need to test Ingress controller and data-plane
    - When the Ingress controller network is interrupted, can the data-plane proxy requests normally?
    - When the Ingress controller Pod is killed, can the data-plane proxy requests normally?
    - DNS error
    - …

Establish a hypothesis

- When the Ingress-controller Pod get <X?>, the client's request can still get a normal response.

Run the experiment

- Litmus provides several ways to configure experiments
    - Using ChaosCenter Portal



Schedule a new Chaos Scenario

Choose Chaos Delegate to send the Chaos Scenario to

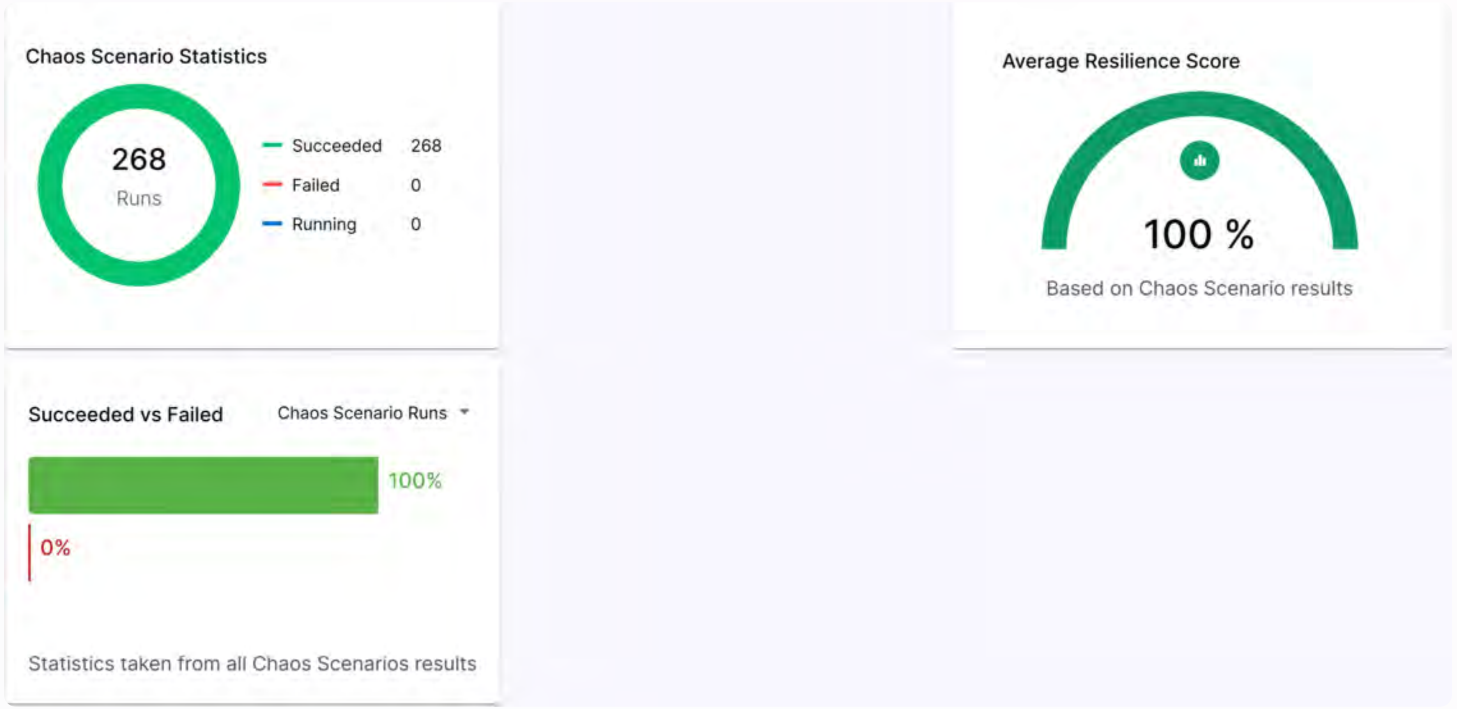
You are creating a new Chaos Scenario
Select a Chaos Delegate to run this Chaos Scenario

Self-Agent
36ed257b-8403-4e4f-9fcd-f4efa82fac28

    - Using YAML manifest

Evaluate the results

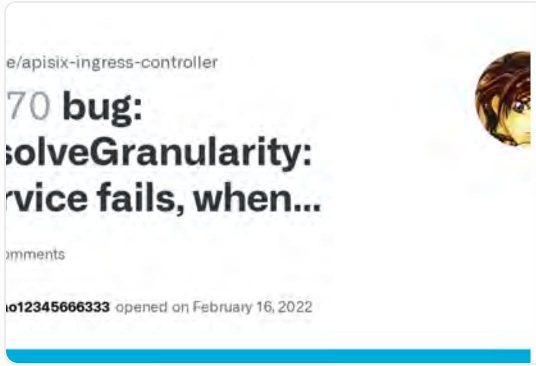
- Built-in statistics report



- Integration with Prometheus and Grafana

Benefits and Future

- Help us build confidence. Now we have many production users.
- Found a bug that the configuration becomes invalid after the ingress-controller Pod is **restarted multiple times**


e/apisix-ingress-controller

70 **bug:**
solveGranularity:
vice fails, when...

mments

o12345666333 opened on February 16, 2022

GitHub

bug: resolveGranularity: service fails, when...

I found that resolveGranularity: service no longer takes effect after multiple restarts of the APISIX Ingress...

Benefits and Future

- Introduce chaos experiments based on Litmus into the CI pipeline
- Provide reference documents and examples for users

Thanks!

Twitter: @zhangjintao9020

GitHub: **https://github.com/tao12345666333**