Fault Injection Testing

Fault injection testing is the deliberate introduction of errors and faults to a system to validate and harden its stability and reliability. The goal is to improve the system's design for resiliency and performance under intermittent failure conditions over time.

When To Use

Problem Addressed

Systems need to be resilient to the conditions that caused inevitable production disruptions. Modern applications are built with an increasing number of dependencies; on infrastructure, platform, network, 3rd party software or APIs, etc. Such systems increase the risk of impact from dependency disruptions. Each dependent component may fail. Furthermore, its interactions with other components may propagate the failure.

Fault injection methods are a way to increase coverage and validate software robustness and error handling, either at build-time or at run-time, with the intention of "embracing failure" as part of the development lifecycle. These methods assist engineering teams in designing and continuously validating for failure, accounting for known and unknown failure conditions, architect for redundancy, employ retry and back-off mechanisms, etc.

Applicable to

- **Software -** Error handling code paths, in-process memory management.
  - *Example tests:* Edge-case unit/integration tests and/or load tests (i.e. stress and soak).
- **Protocol -** Vulnerabilities in communication interfaces such as command line parameters or APIs.
  - *Example tests:* Fuzzing provides invalid, unexpected, or random data as input we can assess the level of protocol stability of a component.
- **Infrastructure -** Outages, networking issues, hardware failures.
  - *Example tests:* Using different methods to cause fault in the underlying infrastructure such as Shut down virtual machine (VM) instances, crash processes, expire certificates, introduce network latency, etc. This level of testing relies on statistical metrics

observations over time and measuring the deviations of its observed behavior during fault, or its recovery time.

How to Use

Architecture

Terminology

- **Fault -** The adjudged or hypothesized cause of an error.
- **Error -** That part of the system state that may cause a subsequent failure.
- **Failure -** An event that occurs when the delivered service deviates from correct state.
- **Fault-Error-Failure cycle -** A key mechanism in dependability: A fault may cause an error. An error may cause further errors within the system boundary; therefore each new error acts as a fault. When error states are observed at the system boundary, they are termed failures.

Fault Injection Testing Basics

Fault injection is an advanced form of testing where the system is subjected to different failure modes, and where the testing engineer may know in advance what is the expected outcome, as in the case of release validation tests, or in an exploration to find potential issues in the product, which should be mitigated.

Fault Injection and Chaos Engineering

Fault injection testing is a specific approach to testing one condition. It introduces a failure into a system to validate its robustness. Chaos engineering, coined by Netflix, is a practice for generating new information. There is an overlap in concerns and often in tooling between the terms, and many times chaos engineering uses fault injection to introduce the required effects to the system.

High-level Step-by-Step

Fault injection testing in the development cycle

Fault injection is an effective way to find security bugs in software, so much so that the Microsoft Security Development Lifecycle requires fuzzing at every untrusted interface of every product and penetration testing which includes introducing faults to the system, to uncover potential vulnerabilities resulting from coding errors, system configuration faults, or other operational deployment weaknesses.

Automated fault injection coverage in a CI pipeline promotes a Shift-Left approach of testing earlier in the lifecycle for potential issues. Examples of performing fault injection during the development lifecycle:

- Using fuzzing tools in CI.
- Execute existing end-to-end scenario tests (such as integration or stress tests), which are augmented with fault injection.
- Write regression and acceptance tests based on issues that were found and fixed or based on resolved service incidents.
- Ad-hoc (manual) validations of fault in the dev environment for new features.

Fault Injection Testing in the Release Cycle

Much like Synthetic Monitoring Tests, fault injection testing in the release cycle is a part of Shift-Right testing approach, which uses safe methods to perform tests in a production or pre-production environment. Given the nature of distributed, cloud-based applications, it is very difficult to simulate the real behavior of services outside their production environment. Testers are encouraged to run tests where it really matters, on a live system with customer traffic.

Fault injection tests rely on metrics observability and are usually statistical; The following high-level steps provide a sample of practicing fault injection and chaos engineering:

- Measure and define a steady (healthy) state for the system's interoperability.
- Create hypotheses based on predicted behavior when a fault is introduced.
- Introduce real-world fault-events to the system.
- Measure the state and compare it to the baseline state.
- Document the process and the observations.
- Identify and act on the result.

Fault Injection Testing in Kubernetes

With the advancement of kubernetes (k8s) as the infrastructure platform, fault injection testing in kubernetes has become inevitable to ensure that system behaves in a reliable manner in the event of a fault or failure. There could be different type of workloads running within a k8s cluster which are written in different languages. For eg. within a K8s cluster, you can run a micro service, a web app and/or a scheduled job. Hence you need to have mechanism to inject fault into any kind of workloads running within the cluster. In addition, kubernetes clusters are managed differently from traditional infrastructure. The tools used for fault injection testing within kubernetes should have compatibility with k8s infrastructure. These are the main characteristics which are required:

- Ease of injecting fault into kubernetes pods.
- Support for faster tool installation within the cluster.
- Support for YAML based configurations which works well with kubernetes.
- Ease of customization to add custom resources.
- Support for workflows to deploy various workloads and faults.
- Ease of maintainability of the tool
- Ease of integration with telemetry

Best Practices and Advice

Experimenting in production has the benefit of running tests against a live system with real user traffic, ensuring its health, or building confidence in its ability to handle errors gracefully. However, it has the potential to cause unnecessary customer pain. A test can either succeed or fail. In the event of failure, there will likely be some impact on the production environment. Thinking about the **Blast Radius** of the effect, should the test fail, is a crucial step to conduct beforehand. The following practices may help minimize such risk:

- Run tests in a non-production environment first. Understand how the system behaves in a safe environment, using synthetic workload, before introducing potential risk to customer traffic.
- Use fault injection as gates in different stages through the CD pipeline.
- Deploy and test on Blue/Green and Canary deployments. Use methods such as traffic shadowing (a.k.a. Dark Traffic) to get customer traffic to the staging slot.
- Strive to achieve a balance between collecting actual result data while affecting as few production users as possible.
- Use defensive design principles such as circuit breaking and the bulkhead patterns.
- Agreed on a budget (in terms of Service Level Objective (SLO)) as an investment in chaos and fault injection.
- Grow the risk incrementally - Start with hardening the core and expand out in layers. At each point, progress should be locked in with automated regression tests.

Fault Injection Testing Frameworks and Tools

Fuzzing

- OneFuzz - is a Microsoft open-source self-hosted fuzzing-as-a-service platform which is easy to integrate into CI pipelines.
- AFL and WinAFL - Popular fuzz tools by Google's project zero team which is used locally to target binaries on Linux or Windows.
- WebScarab - A web-focused fuzzer owned by OWASP which can be found in Kali linux distributions.

Chaos

- Azure Chaos Studio - An in-preview tool for orchestrating controlled fault injection experiments on Azure resources.
- Chaos toolkit - A declarative, modular chaos platform with many extensions, including the Azure actions and probes kit.
- Kraken - An Openshift-specific chaos tool, maintained by Redhat.
- Chaos Monkey - The Netflix platform which popularized chaos engineering (doesn't support Azure OOTB).
- Simmy - A .NET library for chaos testing and fault injection integrated with the Polly library for resilience engineering.
- Litmus - A CNCF open source tool for chaos testing and fault injection for kubernetes cluster.
- This ISE dev blog post provides code snippets as an example of how to use Polly and Simmy to implement a hypothesis-driven approach to resilience and chaos testing.

Conclusion

From the principals of chaos: "The harder it is to disrupt the steady-state, the more confidence we have in the behavior of the system. If a weakness is uncovered, we now have a target for improvement before that behavior manifests in the system at large".

Fault injection techniques increase resilience and confidence in the products we ship. They are used across the industry to validate applications and platforms before and while they are delivered to customers. Fault injection is a powerful tool and should be used with caution. Cases such as the Cloudflare 30 minute global outage, which was caused due to a deployment of code that was meant to be “dark launched”, entail the importance of curtailing the blast radius in the system during experiments.

Resources

- Mark Russinovich's fault injection and chaos engineering blog post
- Cindy Sridharan's Testing in production blog post
- Cindy Sridharan's Testing in production blog post cont.
- Fault injection in Azure Search
- Azure Architecture Framework - Chaos engineering
- Azure Architecture Framework - Testing resilience
- Landscape of Software Failure Cause Models

Last update: August 22, 2024