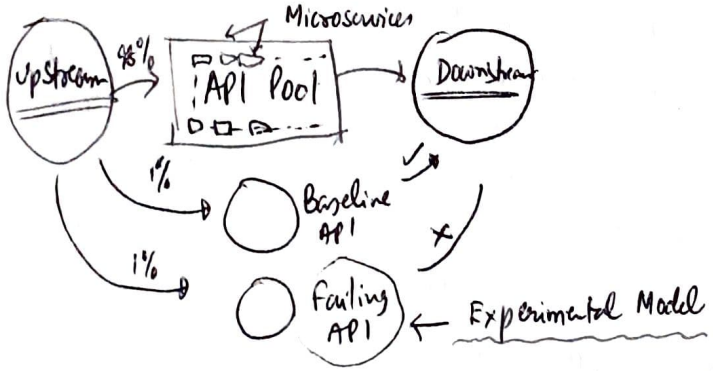
Rethinking how the industry approaches 'Chaos Engineering'

Info Q (Nora Jones)

- Building resilience to withstand turbulent conditions in production
- Creating foresights & predictive maintenance for systems in production.
- Builds confidence in system behaviors through experiments.
- Goals :
  (A) Build toolings to stop issues / discover vulnerabilities in existing architecture.
  (B) Build a culture of resilience to the unexpected.

ChAP → Automating production Chaos Experiment
[Chaos Automation in Production]



Building ChAP

Features → Timeouts, Retries, Fallbacks, % of traffic served

- Designing Meaningful Experiments : Who, What*, Why, When
- "Apollo-1 Launch Rehersal Test" (Novices / Expert)
- Digging advanced patterns in the systems / incident archive to build critical experimental modeling.

- While building the exercise for experiments, a few important questions include →

(A) Which systems haven't failed in a while?
(B) Which system failure took us by surprise?
(C) Which incidents involved "unowned systems" or those that needed "experts" to step in & resolve?
(D) Which incidents involved people that haven't worked together much?
(E) What did recent "near misses" look like?
(F) Which incidents involved difficulties in figuring out what was even going on?

SCOPE → Steady-State (baseline) → Measurable output indicating normal behavior
→ Where are we injecting failure → diff. b/w "normal" & "good" operation.

Safe-gaurding Experiments

Creating chaos is easy, what's difficult is :-
Minimizing blast radius, safety-net implementation, & putting active listeners on when things go wrong.

(Post Experiment) Asking Better Experiments

(A) What / why did you experiment?
(B) what was surprising or new?
(C) what mental models got recalibrated?

Chaos Engineering can be used as cultural artifacts.
↳ Good way to learn about collaboration & system characteristics.

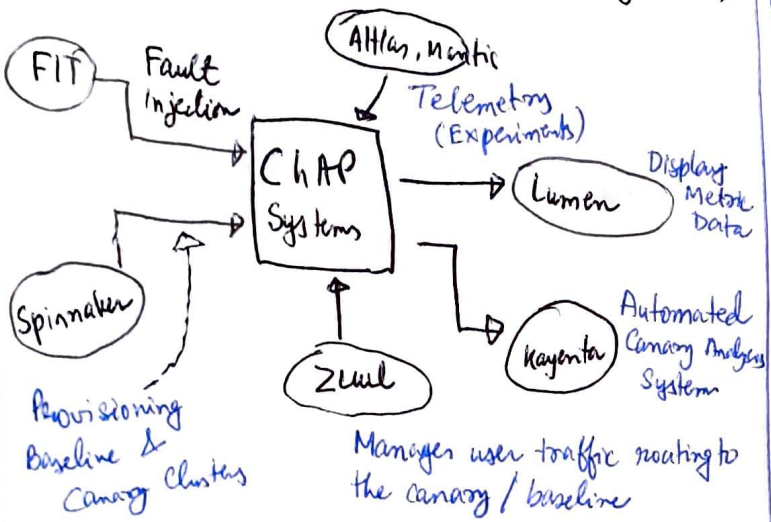
Key Takeaways : Rethinking Approaches

(A) Use write-ups as learning opportunity.
(B) Unify all phases of Chaos together w/ incidence as well.

ChAP : Automating chaos experiments in Production

Reference: Basiri et al. (ICSE 2019)

- ChAP is developed on the public cloud as a complex set of interacting microservices.
- Two focussed failure modes
  - Services becoming slower (increase in response latency)
  - Services failing outright (returning errors)



Safeguards

(To minimize the blast radius of the experiment)

(A) Experiments are run only on weekdays during business hours.

(B) Not run in failed-over regions for independent regions.

(C) Set of concurrently running ChAP experiments shouldn't impact over x % of the N/w traffic.

(D) If the impact is over threshold, the experiments stop immediately.

Error Rates

Four golden signals (SRE) →
↳ Latency, Throughput, Error Rate, Saturation

Take Aways

1. ChAP is excellent as a fault-injection platform
2. ChAP for canary development & load testing experiments.

(Statistical Significance) ≠ Validity

- Accepting A/B test with (say, 95% stat. significance) doesn't correlate with guaranteed uplift. So, stat. significance may not correlate to validity.
- Stat. significance alone isn't sufficient in deciding when to terminate experiment:
  Sufficiently long running experiments (even with huge data volume) is an interesting indicator.
  Important decision parameters →
  - Includes 1-2 business cycles
  - Have enough absolute conversions/transactions
  - Have enough confidence rate (uplift)

Golden Stopping Rules

→ Sufficient test durations (1-2 business cycles)
→ Well-defined minimum pre-calculated sample size
→ Stat. significance (≤ 95-99%)
→ Abs. Number per variation.

(X) Search for uplifts within segment which do not have statistical validity
↳ Destined to Fail

The Art & Science of Interview Engineering

- Optimality v/s Speed
- Hack edge cases scenario (Need to work on that)

Panel : Microservices – Are they still worth it?

What I wish I knew

(1) How to control business complexities with scalable microservices?

(2) One can solve the tradeoff w/ Monolithic architectures (Computational Overhead, No Fault Tolerances) without shifting to Microservices.

(3) Resolve issues around the architectures (dependencies, simplifying complexities) design w/ Microservices?

(4) Importance of dev tooling w/ Microservices.

Are they still worth it?

Y (1) Isolation of services, Delivery speedups.

Y (2) Work-independence. Flexible innovation w/o introducing deep complexities.

N (3) Operational overhead w/ monolithic services offer scalable, cheaper, implementable solutions

Y (4) ◦ Decoupled environment, release processes fits elegantly into the Agile Ecosystems.
◦ Faster change at scalable separation.

What journey Enterprises need to go through for adopting Microservices (Operational challenge)

(A) Technology & Design decisions alongside team structures.

(B) Monolithic Architecture suffers from low scalable in-memory solutions → REDIS is not perfect. (Caching benefits are much more w/ Microservices over Monolithic Architectures)

(C) Strong platform team providing efficient platform & toolboxes may become a pre-requisite for transition to Microservices

(D) Standards & Conventions are much simplistic with Monolithic architectures.

Audience Questions

(1) Defining bounded context with 1000s of Microservices

(A) Atomic thinking to delegate responsibilities (with minimal dependencies) to microservices.

(2) Modular Code (v/s) Microservices

(A) Good when simplifying the applications & decoupling services.

(B) Maintenance & Code-Sanity be at severe risk.

Advice on Adoption/Maturing Microservices

(A) Managed Archive & knowledge base.

(B) -(i) : Well-defined code ownership responsibility
(ii) : Softwares are not designed in isolation
(iii) : Active communication get Agile.

(C) Have good idea of the design philosophies & engineering decisions made.

(D) Keep interested in active toolboxes & platforms to support the microservices.

Conway's law

" Organizations which design systems are constrained to produce designs that are copies of the communication structures of these organizations ".