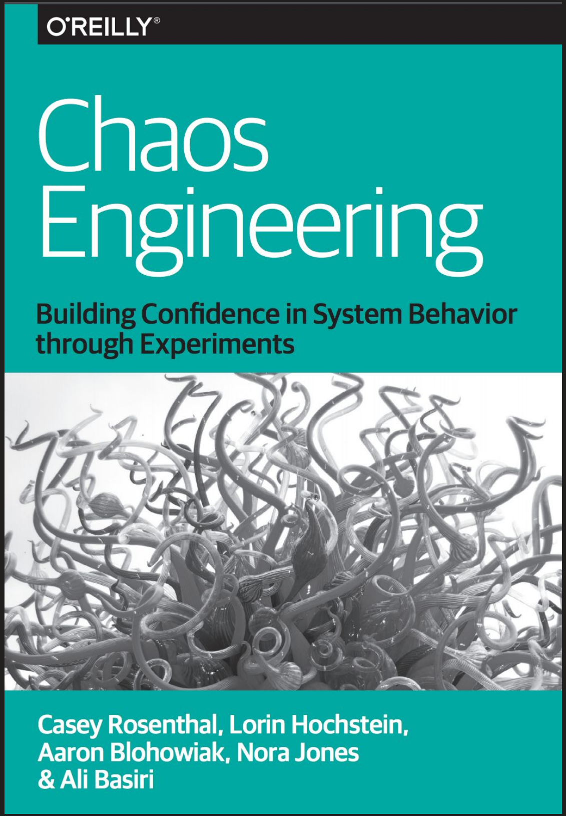
Choose Your Own Adventure: Chaos Engineering



O'REILLY®

Chaos Engineering

Building Confidence in System Behavior through Experiments

Casey Rosenthal, Lorin Hochstein, Aaron Blohowiak, Nora Jones & Ali Basiri

Nora Jones, Senior Chaos Engineer
@nora_js

NETFLIX

In this talk

- Choosing your own adventure with Chaos
- Phases of Chaos
- Road to cultural acceptance
- Alternating between anecdotes and advice (when I do, you’ll see a “Story” icon)

In this talk

- Choosing your own adventure with Chaos
- Phases of Chaos
- Road to cultural acceptance
- Alternating between anecdotes and advice (when I do, you'll see a “Story” icon)

Known ways of testing for availability

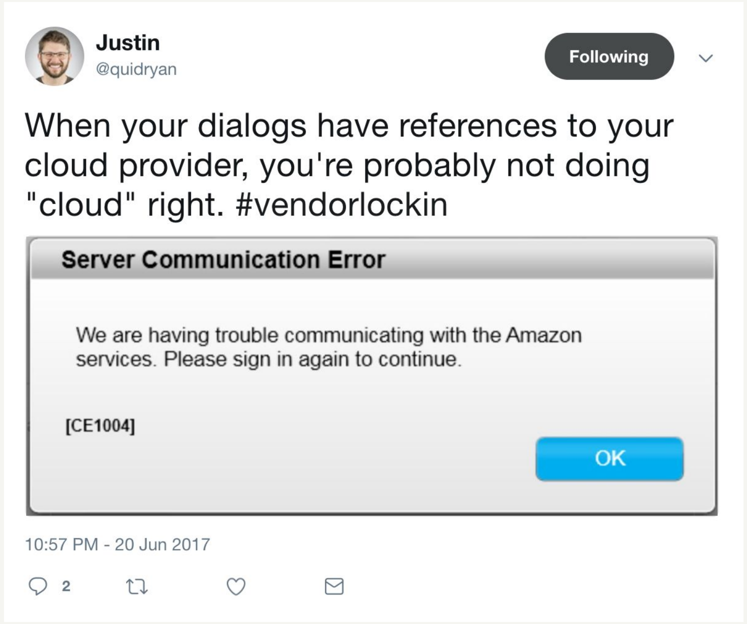
- Unit Tests
- Regression Tests
- Integration Tests
- Chaos Engineering

“I want to emphasize that both sides of the equation [unit/regression/integration testing side and Chaos side] are required to get you the level of availability you want.”

--Haley Tucker, Netflix

Chaos Engineering

You can’t keep blaming your cloud provider



Justin
@quidryan

When your dialogs have references to your cloud provider, you're probably not doing "cloud" right. #vendorlockin

Server Communication Error

We are having trouble communicating with the Amazon services. Please sign in again to continue.

[CE1004]

10:57 PM - 20 Jun 2017

Why is there a fear of Chaos when it’s inevitable?

I have often found that bleeding hearts
have an ironic fear of their own blood.

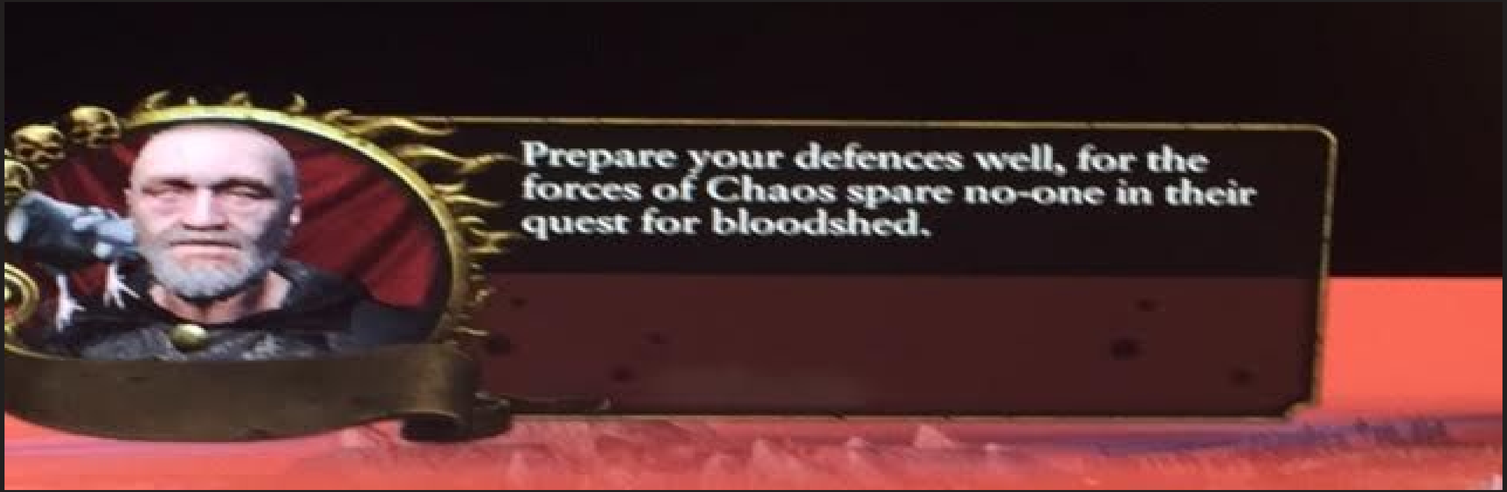
Computers are complicated and they will break.

Meet "Chaos Carol"

Where is Carol starting her Chaos?

STORY

Phase 1: Introducing the Chaos


Prepare your defences well, for the forces of Chaos spare no-one in their quest for bloodshed.

Start with a steady state

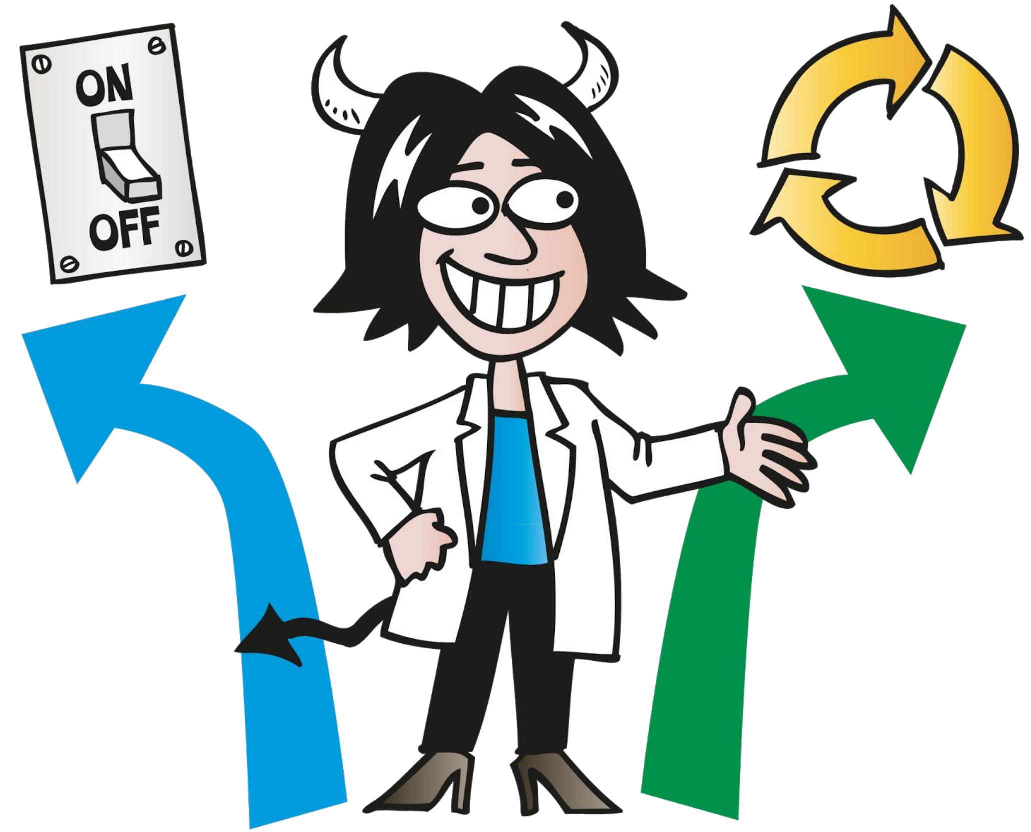
- Define “normal” system and business behavior for your services
- Determine what the system architecture looks like at a high level

Microservices

There isn’t always money in microservices

Randomly turn things off?

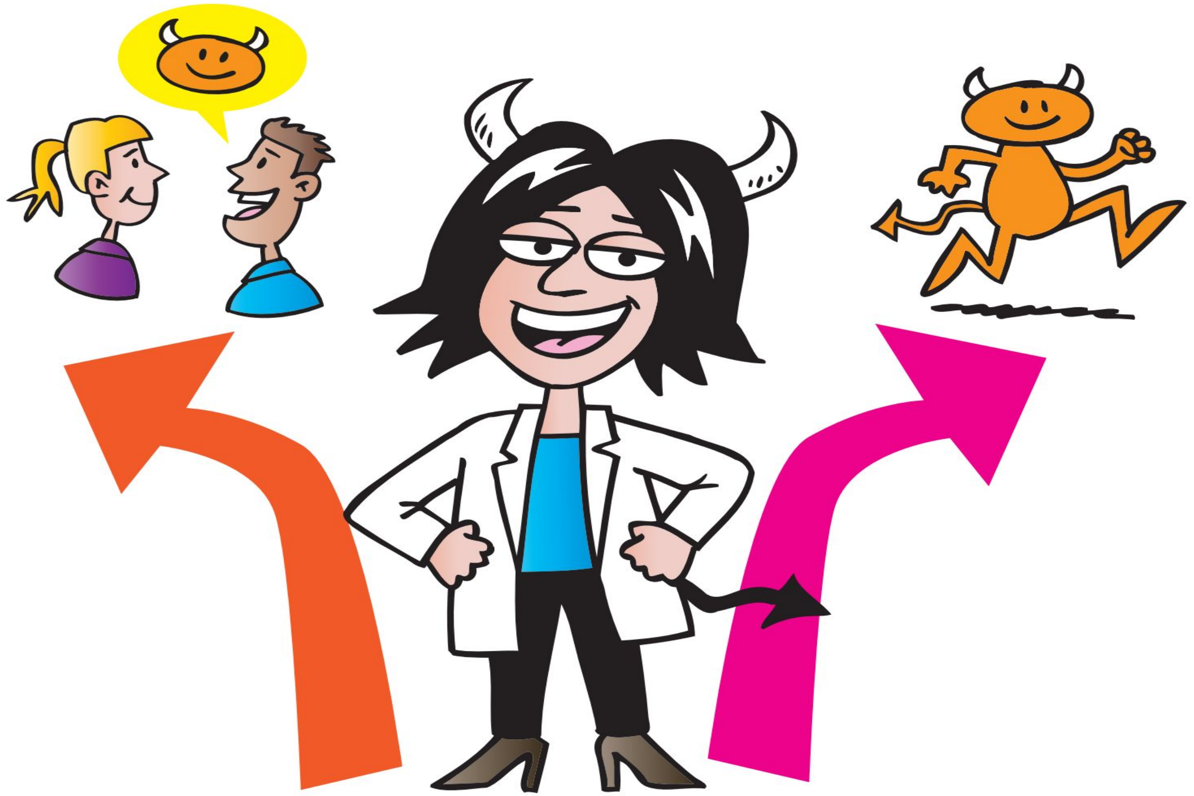
Recreate things that already happened?

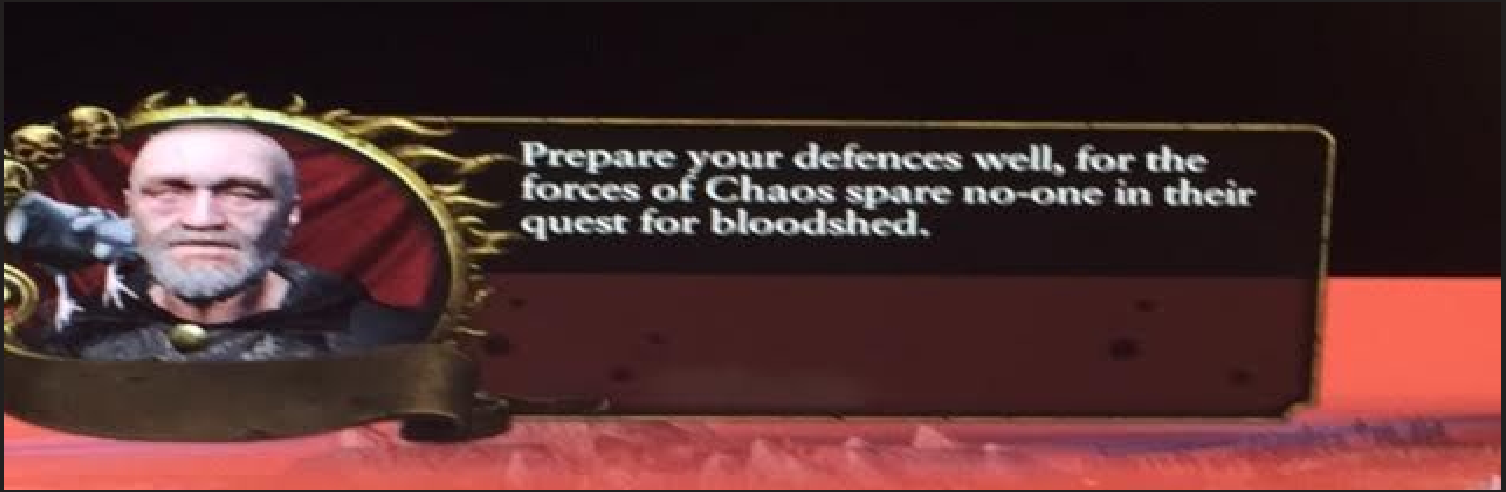


Phase 1.1: Graceful Restarts and Degradation (start out small)

Let people know?

Let the Chaos run automatically?





Prepare your defences well, for the forces of Chaos spare no-one in their quest for bloodshed.

Working on Chaos experiments is a quick way to meet your new colleagues. Do it tactfully.

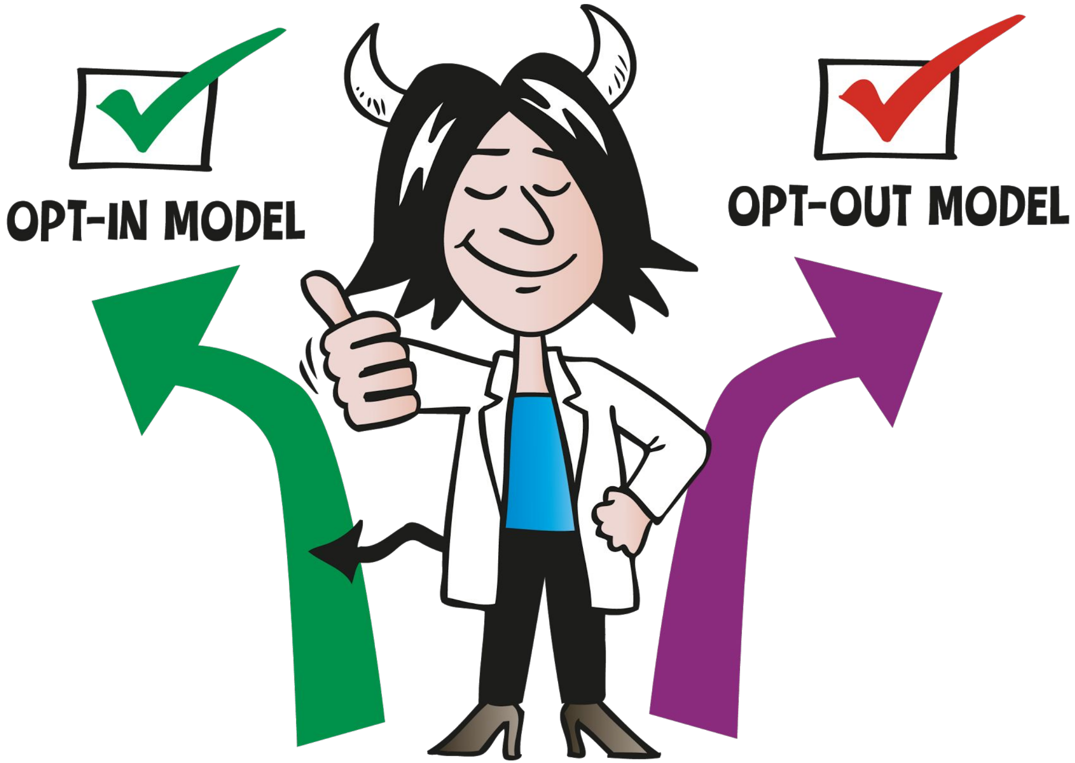
Socialization

Socialization

- Tends to be harder than implementation.
- Part of one’s job as an engineer developing internal tools is to understand your customer and their needs.
- Relate your Chaos experiments to automated tests, to SLAs and ultimately, to the customer experience.

Culture & Chaos

Chaos doesn’t cause problems, it reveals them.


OPT-IN MODEL
OPT-OUT MODEL

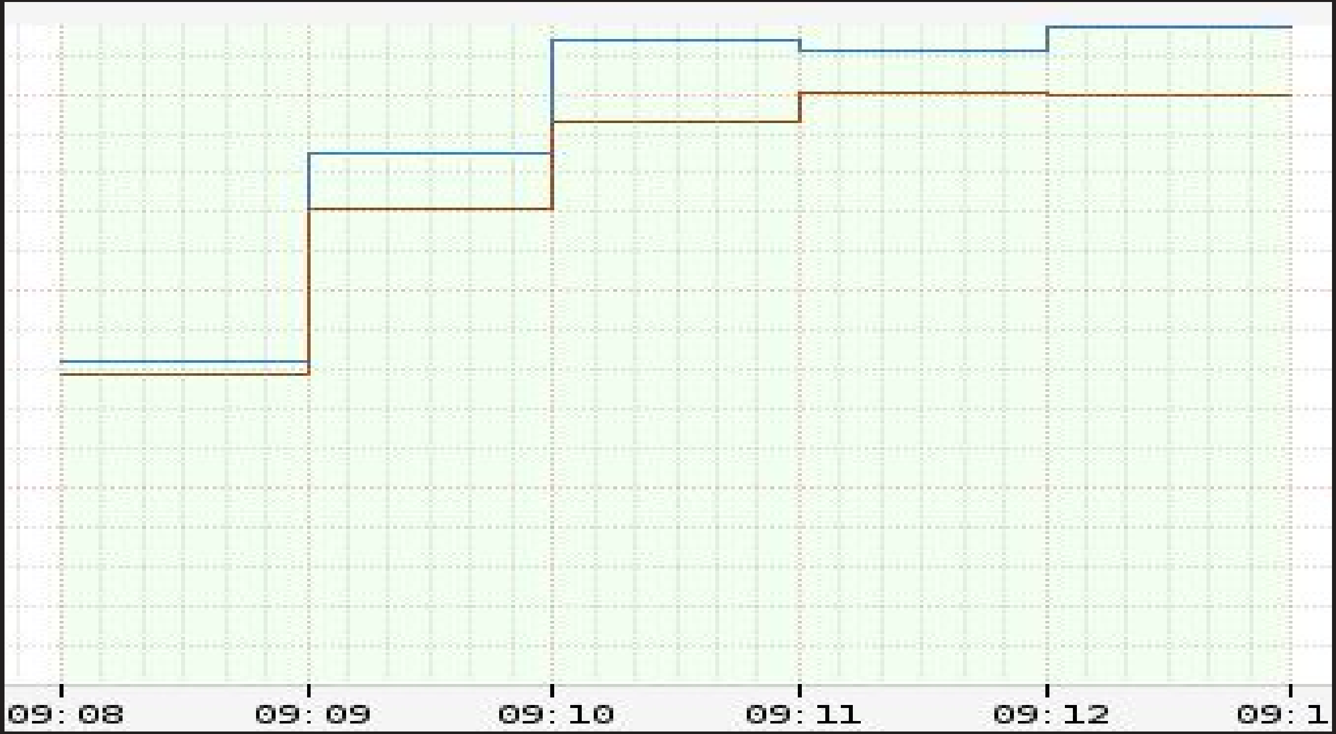
When your customers are your coworkers.

Internal Tools: Selling 101

- Focus more on asking the questions, rather than answering them.
- Find customers willing to try first. Then share their stories.
- Be honest. Don’t make false promises about what Chaos will do.

Wololo!

Monitoring



Monitoring

- Leverage the tools you have.
- If you don’t monitor and measure the Chaos, how can you improve? And how do you know it is working?
- Look at your incidents or JIRA tickets recently. Have they decreased from when you started Chaos testing?
- Monitor culture around Chaos too. Has the idea of it improved? Are you tracking adoption rates? Successes?

Don’t lose sight of your company’s customers.

Whoops, something went wrong…

Netflix Streaming Error

We're having trouble playing this title right now. Please try again later or select a different title.

Strongly consider customer impact with approaching your Chaos testing and proceed with caution where appropriate.

Q
A

Phase 2: Can we cause a cascading failure?

Cascading failures often lie dormant for a long time until they are triggered by an unusual set of circumstances.

Phase 3: Building a Failure Injection Library

https://github.com/norajones/FailureInjectionLibrary

```
let chaos (name:string) (shouldChaos:unit -> bool) (chaos:Async<unit>) : AsyncFilter<_,_,_,_> =
    fun (service:AsyncArrow<_,_>) req -> async {
        if shouldChaos() then
            printfn "%s" name
            do! chaos
        return! service req
}
```

```
let chaos (name:string) (shouldChaos:unit -> bool) (chaos:Async<unit>) : AsyncFilter<_,_,_,_> =
    fun (service:AsyncArrow<_,_>) req -> async {
        if shouldChaos() then
            printfn "%s" name
            do! chaos
        return! service req
}
```

```
let chaos (name:string) (shouldChaos:unit -> bool) (chaos:Async<unit>) : AsyncFilter<_,_,_,_> =
    fun (service:AsyncArrow<_,_>) req -> async {
        if shouldChaos() then
            printfn "%s" name
            do! chaos
        return! service req
}
```

```fsharp
let chaos (name:string) (shouldChaos:unit -> bool) (chaos:Async<unit>) : AsyncFilter<_,_,_,_> =
    fun (service:AsyncArrow<_,_>) req -> async {
        if shouldChaos() then
            printfn "%s" name
            do! chaos
        return! service req
}
```

```
let chaos (name:string) (shouldChaos:unit -> bool) (chaos:Async<unit>) : AsyncFilter<_,_,_,_> =
    fun (service:AsyncArrow<_,_>) req -> async {
        if shouldChaos() then
            printfn "%s" name
            do! chaos
        return! service req
}
```

```
let chaos (name:string) (shouldChaos:unit -> bool) (chaos:Async<unit>) : AsyncFilter<_,_,_,_> =
    fun (service:AsyncArrow<_,_>) req -> async {
        if shouldChaos() then
            printfn "%s" name
            do! chaos
        return! service req
}
```

Types of Chaos Failures

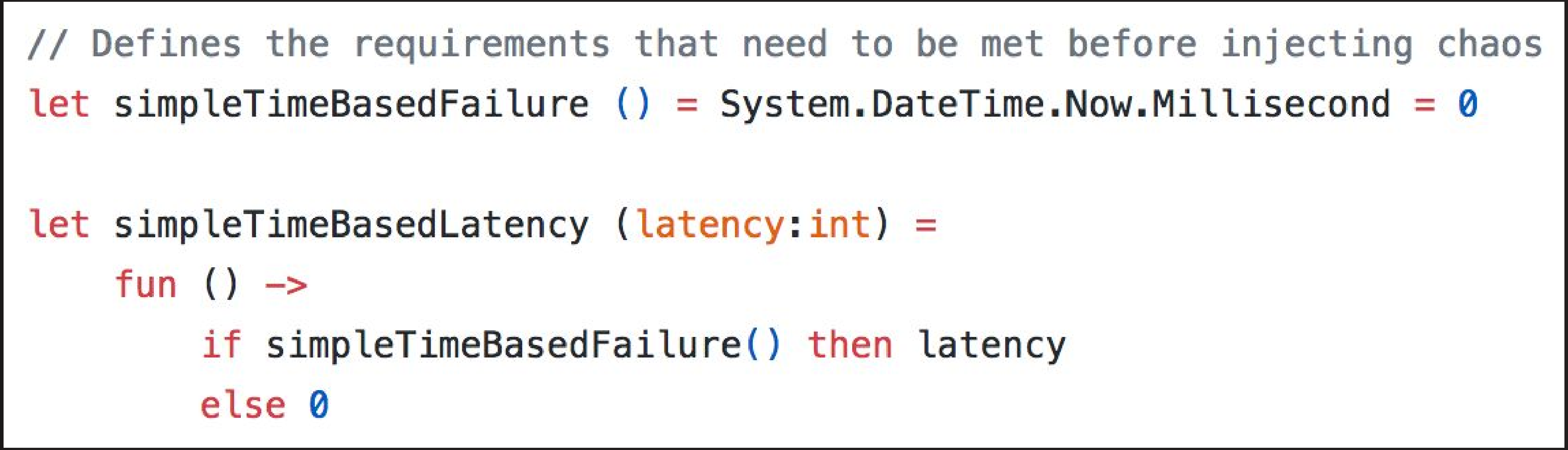
```
let failWithException (ex:System.Exception) = async {
      raise ex
  }
```

```
let introduceLatency (latencyMs:unit -> int) = async {
      // introduce latency
      do! Async.Sleep (latencyMs())
}
```

Types of Chaos Failures

```
  let failWithException (ex:System.Exception) = async {
      raise ex
  }
```

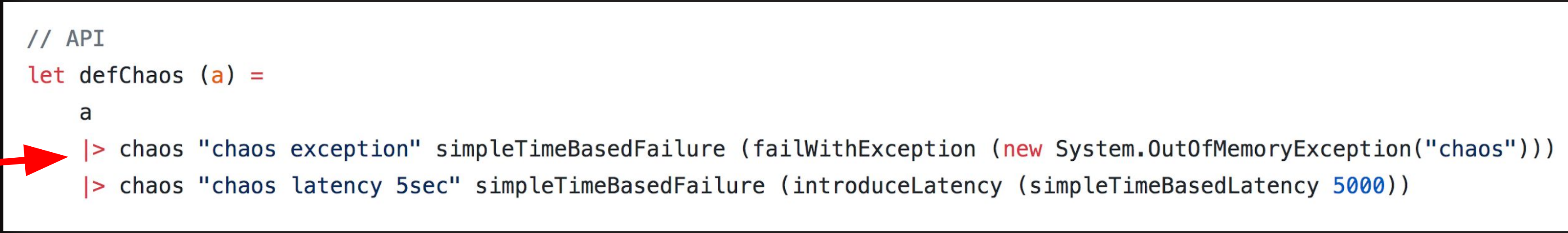
```
let introduceLatency (latencyMs:unit -> int) = async {
      // introduce latency
      do! Async.Sleep (latencyMs())
}
```



```fsharp
// Defines the requirements that need to be met before injecting chaos
let simpleTimeBasedFailure () = System.DateTime.Now.Millisecond = 0

let simpleTimeBasedLatency (latency:int) =
    fun () ->
        if simpleTimeBasedFailure() then latency
        else 0
```

```
// API
let defChaos (a) =
    a
    |> chaos "chaos exception" simpleTimeBasedFailure (failWithException (new System.OutOfMemoryException("chaos")))
    |> chaos "chaos latency 5sec" simpleTimeBasedFailure (introduceLatency (simpleTimeBasedLatency 5000))
```



```
// API
let defChaos (a) =
    a
    |> chaos "chaos exception" simpleTimeBasedFailure (failWithException (new System.OutOfMemoryException("chaos")))
    |> chaos "chaos latency 5sec" simpleTimeBasedFailure (introduceLatency (simpleTimeBasedLatency 5000))
```

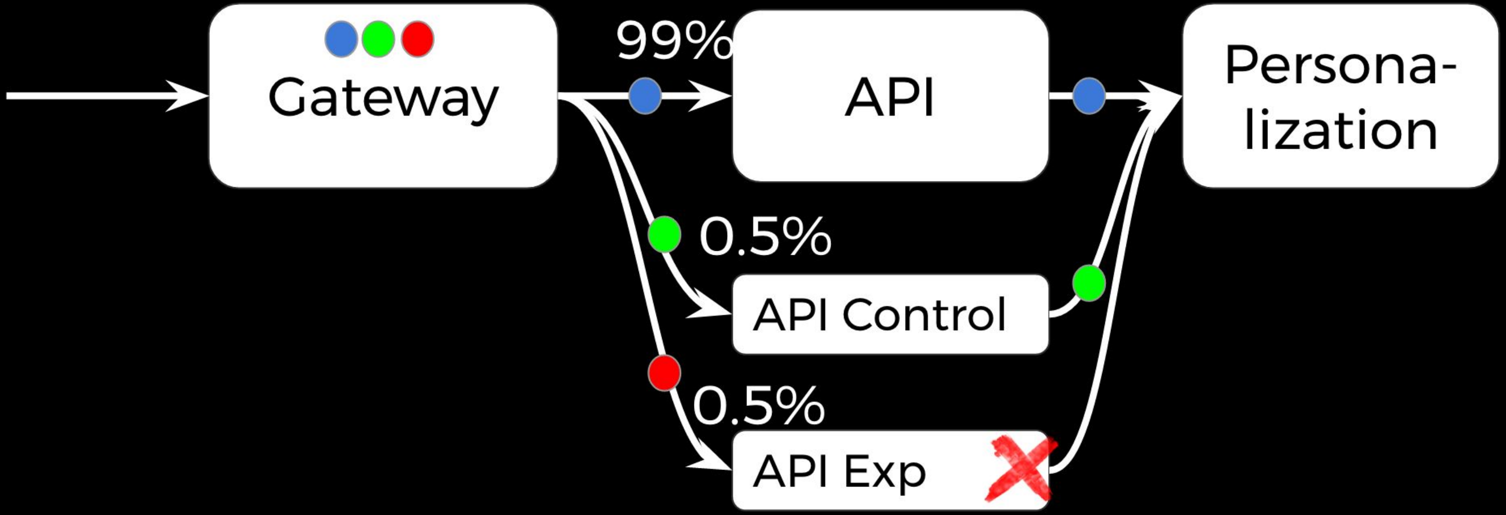
Phase 4: Chaos Automation Platform “ChAP”

@nora_js

ChAP

- Designed to overcome the problems of FIT (failure injection testing)
- Focused on minimizing blast radius
- Concentrates failures onto dedicated instances
- More: orchestration, segmentation, automation, and safety

ChAP



NEW CHANGES
NEW CHANGES

ChAP Goal: Chaos all the things and run all the time.

Phase 5: Targeted Chaos

STORY

Phase 5: Targeted Chaos

Kafka

Targeted Chaos: Kafka Problems

- Monitoring
- Dealing with offsets, especially during geo replication efforts
- High consumer read levels

Targeted Chaos: Kafka Ideas

- Complete topic deletion
- Partial Topic Deletion
- Feeding the consumers bad offsets
- Random Packet Drops
- High Load on Topics
- Deleting segments, random and structured

It's important to have a steady state with Targeted Chaos before you begin.

Record Chaos Success Stories (especially important during adoption)

"We ran a chaos experiment which verifies that our fallback path works (crucial for our availability) and it successfully caught a issue in the fallback path and the issue was resolved before it resulted in any availability incident!"

“While [failing calls] we discovered an increase in license requests for the experiment cluster even though fallbacks were all successful. This likely means that whoever was consuming the fallback was retrying the call, causing an increase in license requests.”

Engagement Guides

- Know your company's culture.
- Set goals for each level of Chaos adoption you expect.
- Define success criteria.

Should you develop experiments for the service teams?

Let them do it on their own?

Takeaways

- Pervasive cultural patterns play out in advocating for Chaos.
- There will be “adventure” choices you need to make when choosing your Chaos.
- Measure your metrics for business and cultural success.

Questions?

@nora_js
chaos@netflix.com