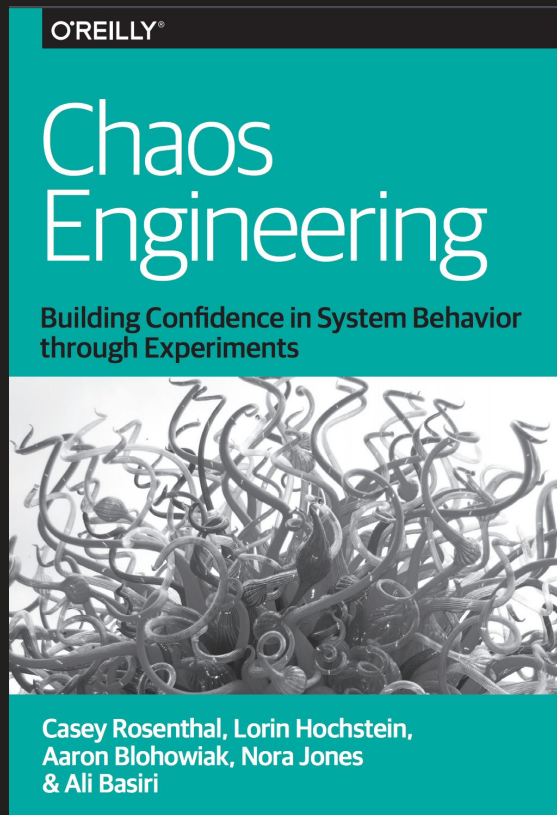


The Road to Chaos

Nora Jones, Senior Chaos Engineer
@nora_js



NETFLIX

In this talk

- Chaos at a Startup vs Chaos at a Netflix
- 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

“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

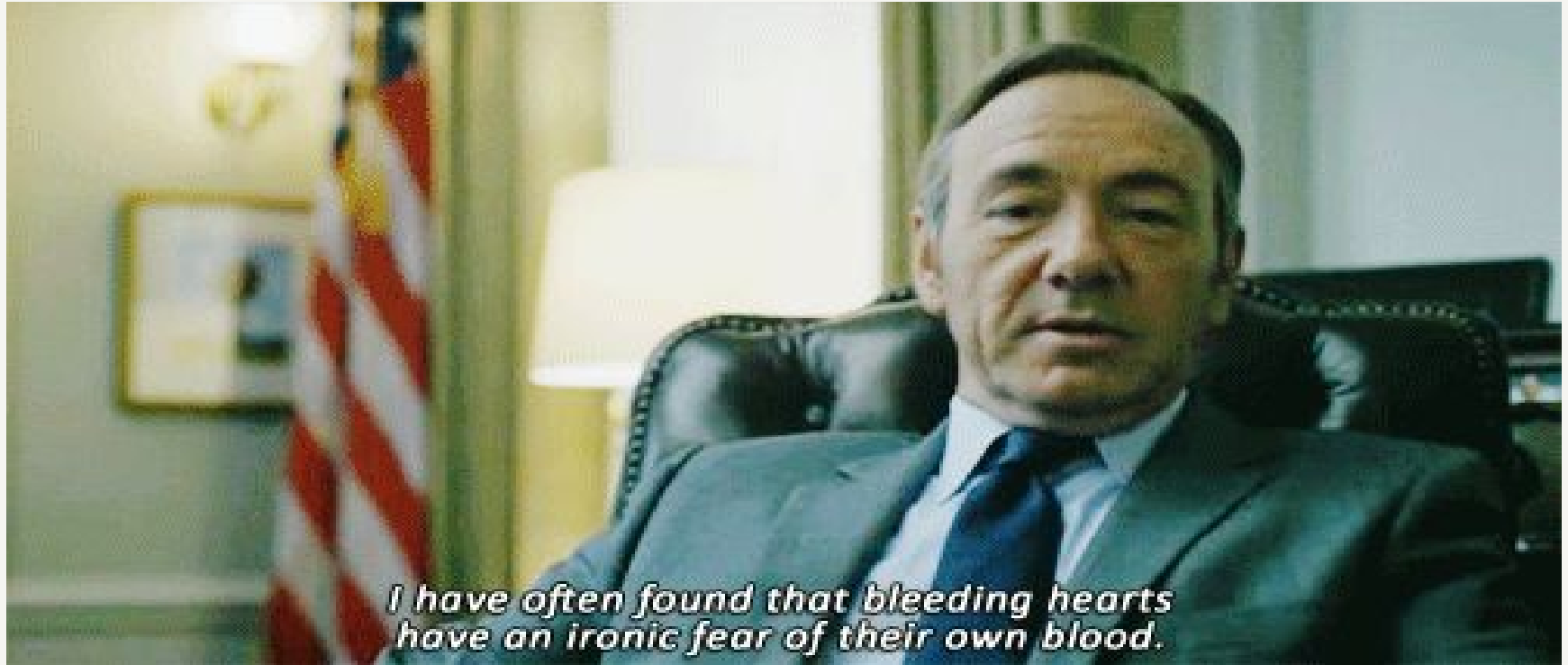
Chaos Engineering

You can't keep blaming your cloud provider

@nora_js

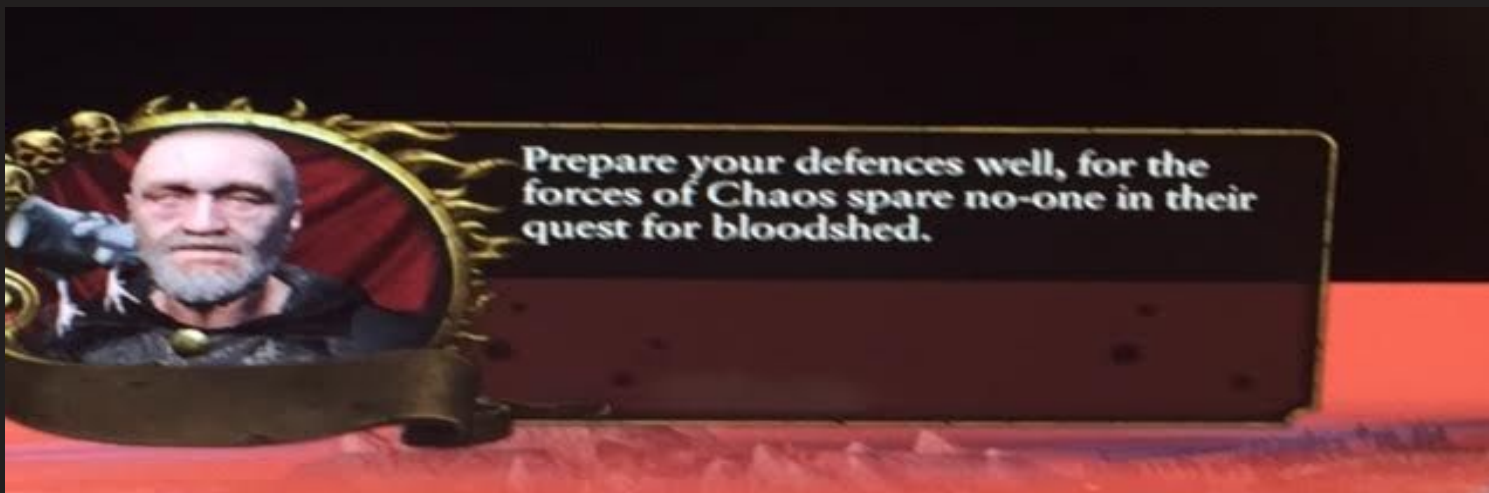


Why is there a fear of Chaos when it's inevitable?



**Computers are
complicated and they will
break.**


Phase 1: Introducing the Chaos



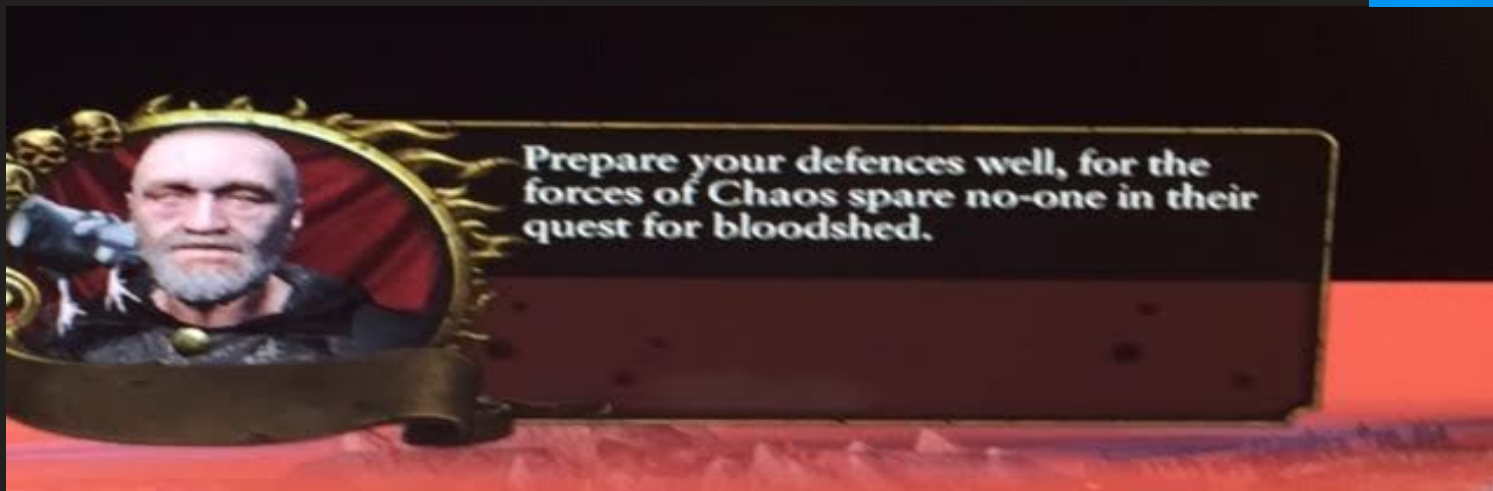
Microservices

There isn't always money in microservices



A solid red vertical bar is positioned on the far left side of the image, extending from the top to the bottom.

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



You don't need to consistently notify that the armies of Chaos are coming, but developers should be aware that these types of experiments are occurring.





The bad men are coming!


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 Chaos experiments to automated tests, to SLAs and ultimately, to the customer experience.

Culture & Chaos

A solid red vertical bar is positioned on the far left side of the image, extending from the top to the bottom.

**Chaos doesn't cause
problems, it reveals
them.**

**When your customers
are your coworkers.**

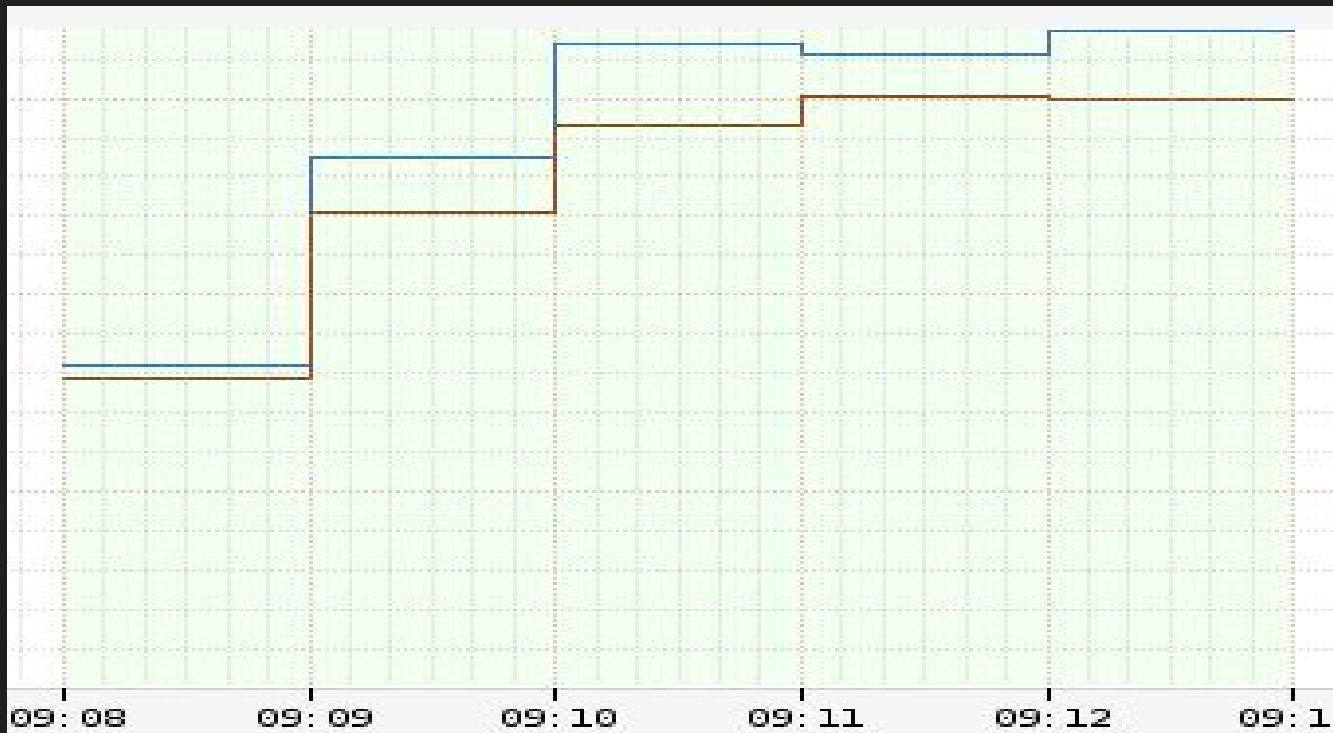
wo1o1o!



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.

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.

Phase 2: Can we cause a cascading failure?



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  
  }
```




```
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())  
}
```

```
// 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”

ChAP

- Designed to overcome Failure Injection Testing problems.
- Focused on minimizing blast radius to provide clear signals and high confidence.
- Concentrates failures only onto dedicated instances
- Works kind of like a sticky deployment canary.

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

Phase 5: Targeted Chaos

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.

Takeaways

- Pervasive cultural patterns play out in advocating for Chaos.
- Measure your metrics for business and cultural success.

Questions?

@nora_js

chaos@netflix.com