Goldman Sachs

Cloudy with a Chance of Chaos

Verifying the Resiliency of Cloud-Native Applications

Introduction

Hello!

I'm **Bella Wiseman**

- Engineer + Leader @ Goldman Sachs
- 10+ years in Fintech
- Second-generation woman software engineer

Agenda

1. *What is Chaos Engineering?*
2. *Chaos Readiness*
3. *Case Study*
4. *Takeaways*

Chaos **Engineering**

Find your next production incident before *it* finds *you*.

How to Run a Chaos Test

1. Define your success criteria
2. Define and measure your steady state
3. Inject failure
4. Observe outcomes
5. Restore system to steady state (if required)

Define your Success Criteria

What does success mean for this test?

1. No impact to availability?
2. Minimal impact to availability? How much?
3. Self-healing system? After how long?
4. Alert triggered? Developer paged?
5. Dashboard reflecting system state?

Define and Measure your Steady State

1. SLO = Service Level Objective = long-term goal
2. Alerting threshold = when you page someone = short-term goal
3. Chaos tests should measure against alerting thresholds, not SLOs
4. Measure availability from your customer's viewpoint
5. It's ok to run an experiment (in non-prod) before you have all the answers about what steady state means

Inject Failure

1. Identify points of failure
2. Prioritize: impact of outage, probability of occurrence and current confidence level
3. Start simple, even manual
4. Start in **non-prod**

Observe Outcome

1. Don’t rely exclusively on your dashboards
    1. they are part of the system under test
2. Double-check everything, as you would in a real incident
3. Measure from your end user’s perspective

Restore Steady State

May be required if:

1. Chaos test had unexpected outcomes
2. The system is not self-healing vis-à-vis this fault, and manual intervention is expected

Chaos-Ready **Environment**

- Never *start* in production
- Environment should be as similar to prod as possible
- Safety first – even in non-prod
  - Observability and monitoring
  - Small blast radius
  - Ability to recover from an outage without impacting your customers
- *After* you've tested in non-prod, decide whether moving to production is worthwhile

Chaos-Ready **Teams**

- Commitment to resiliency and operational excellence
- If your system has known resiliency issues that are not being fixed, fix them first! Don't chaos test it to find more things that you won't fix.
- Culture of actively acknowledging, embracing and mitigating risk
- Psychological safety and blameless post-mortems

Chaos Engineering: Convince your boss

1. ~~It's fun~~
2. ~~I just read a really great blog post~~
3. ~~It's trending on my twitter feed~~
4. ~~I just saw a really cool talk~~
5. ~~I want to write a cool blog post~~
6. ~~I want to get promoted~~
7. Before we release feature x, shouldn't we test what will happen if y goes down?
8. Can we replicate the circumstances of major prod incident y, to ensure that if it happens again, our customers won't be impacted?

Chaos Engineering is Easy!

1. Even *manual* chaos testing provides tons of value with minimal investment
2. Lots of open source and vendor software is available
3. Basic use cases are often covered by multiple blog posts and products
4. Cloud providers provide APIs that allow you to manipulate infrastructure easily

Chaos Engineering is Hard!
Technical Challenges

1. Chaos tests, by definition are *system* tests, with many dependencies
    1. Prod parallel environment (ongoing investment)
    2. Canaries (can be difficult to simulate large-scale issues in a canary)
    3. False positives
2. Scaling across multiple systems: Each system is unique, and fails differently
3. Managed services on the cloud often abstract away the details and don't allow you to access or change the underlying resources or network

Chaos Engineering is Hard!
Human Challenges

"…the three great virtues of a programmer: laziness, impatience, and hubris" ~Larry Wall, ProgrammingPerl (Oreilly and Associates)

Laziness

- I have enough chaos (and accompanying work) already without this, thank you.
- If it ain't broke, don't break it.

Impatience

- I want to code the next feature already
- This will take time

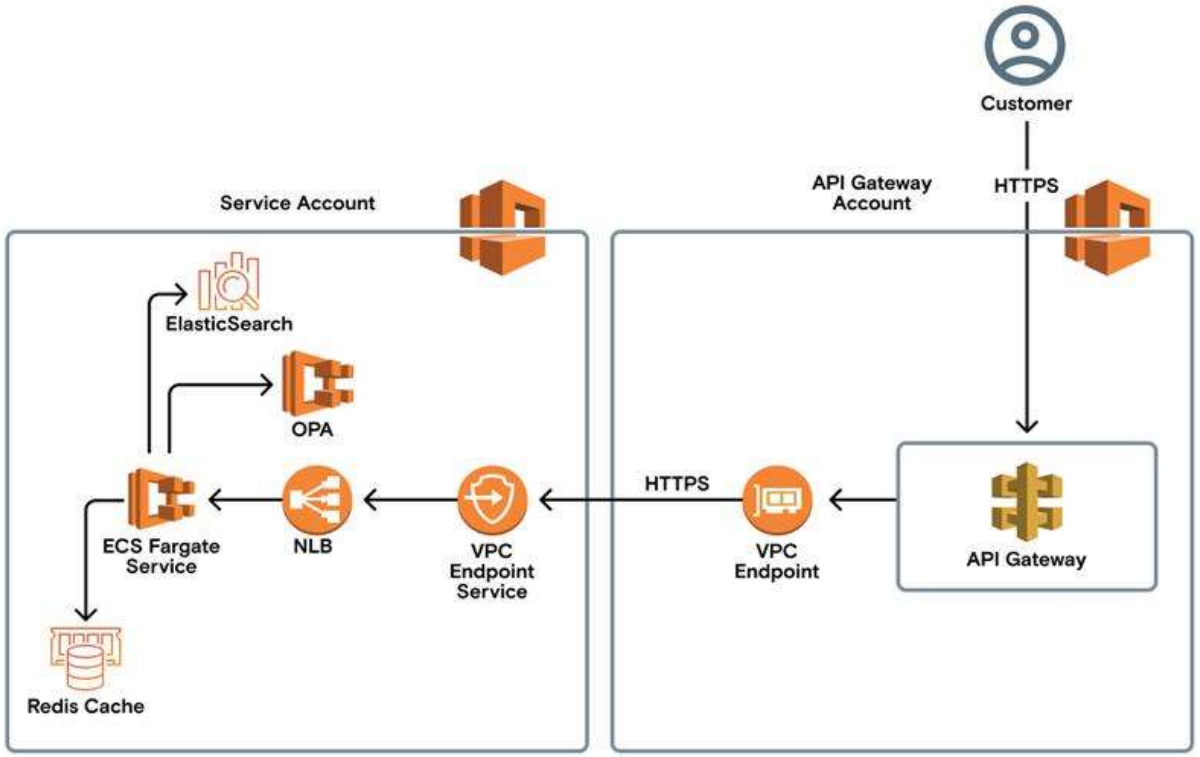
Hubris

- My system won't break
- We already wrote code to handle outages *(did you test it?)*

Case Study
Chaos Testing in Pre-Prod

Simplified architecture of real Goldman Sachs system

1. API Gateway proxies, validates and authenticates the request
2. ECS Fargate – 3 tasks running in different AZs
3. OPA to check entitlements
4. Elastic Search to search across documents
5. Redis – dependency only for some types of requests

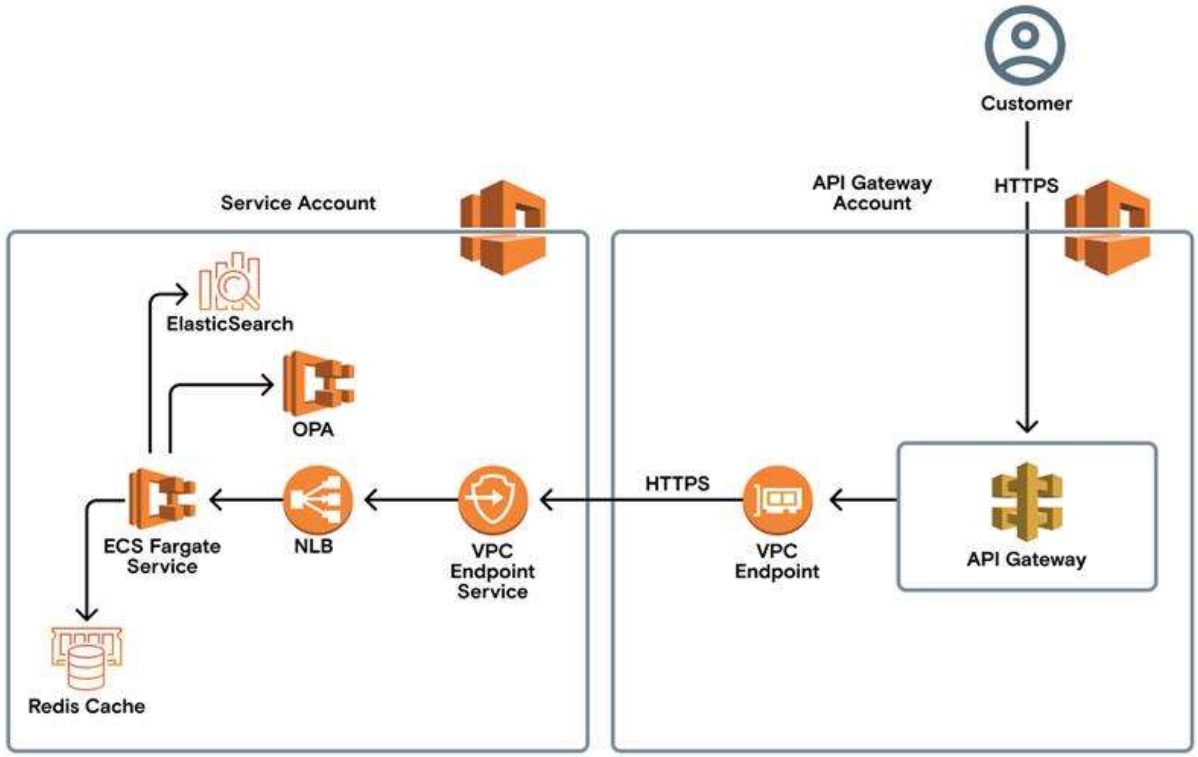


Chaos Engineering on the Modern Cloud
Challenges

There are no machines to bring down.

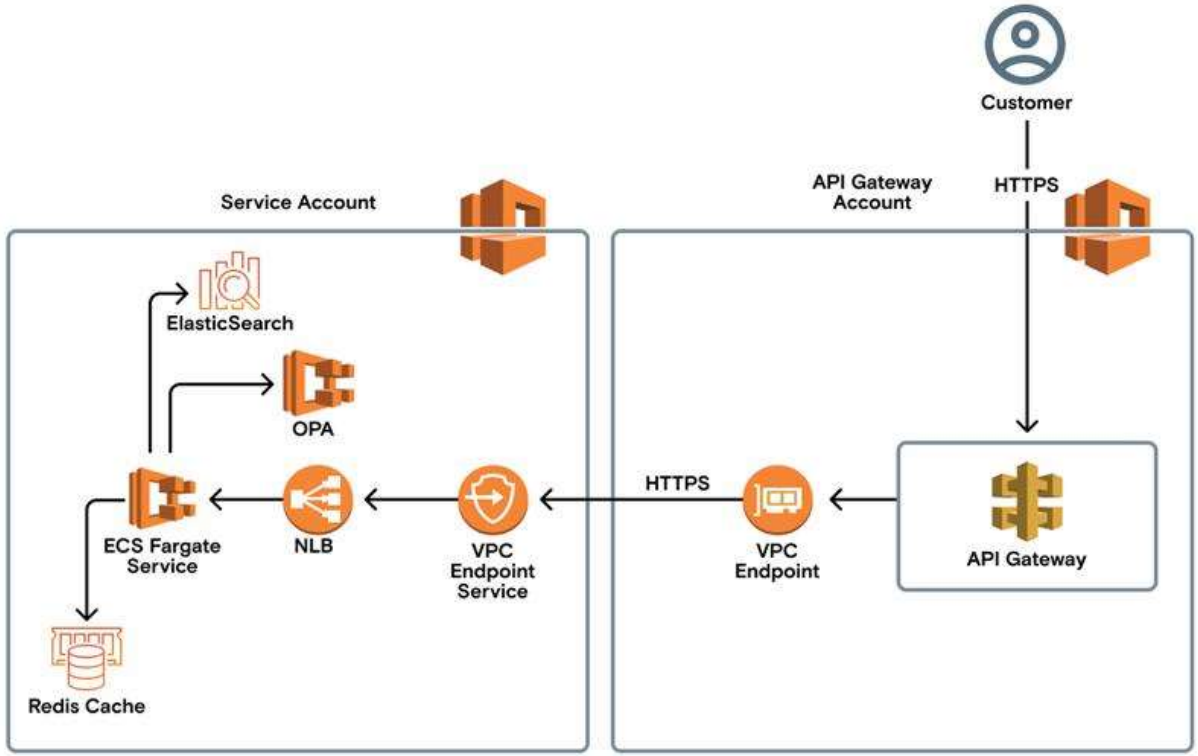
There is no access to the underlying machines

How to inject chaos?



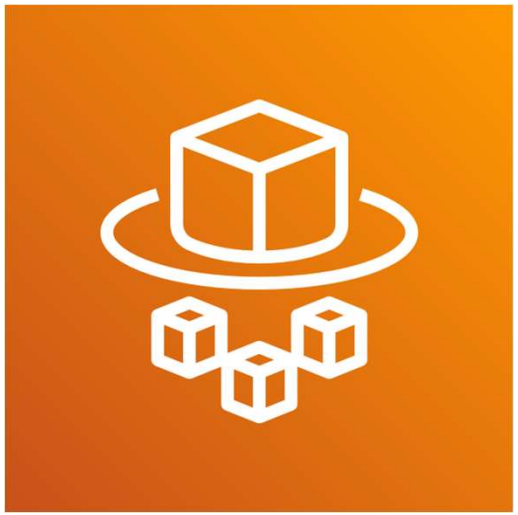
Chaos Engineering on the Modern Cloud
Chaos Seams

1. Use cloud APIs/functionality
2. Trigger known weaknesses
3. Disrupt network connectivity between services



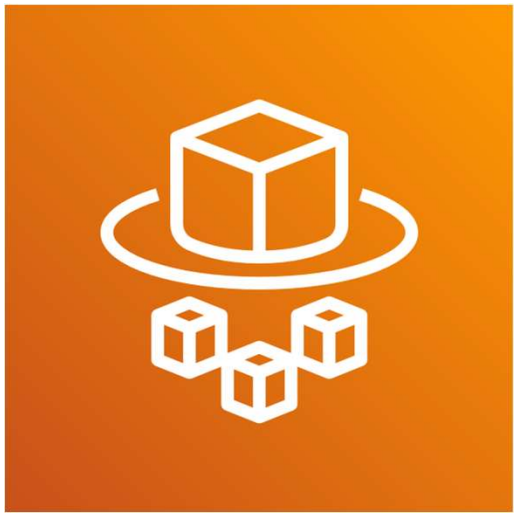
AWS ECS: Elastic Container Service

- Fully managed container orchestration service on AWS
- Either run on EC2 instances… or go serverless with ECS Fargate



AWS ECS Fargate

- Serverless container orchestration
- Define tasks; configure your cluster and AWS manages the rest
- For resiliency, configure your cluster to run across multiple availability zones



What happens if a task goes down?

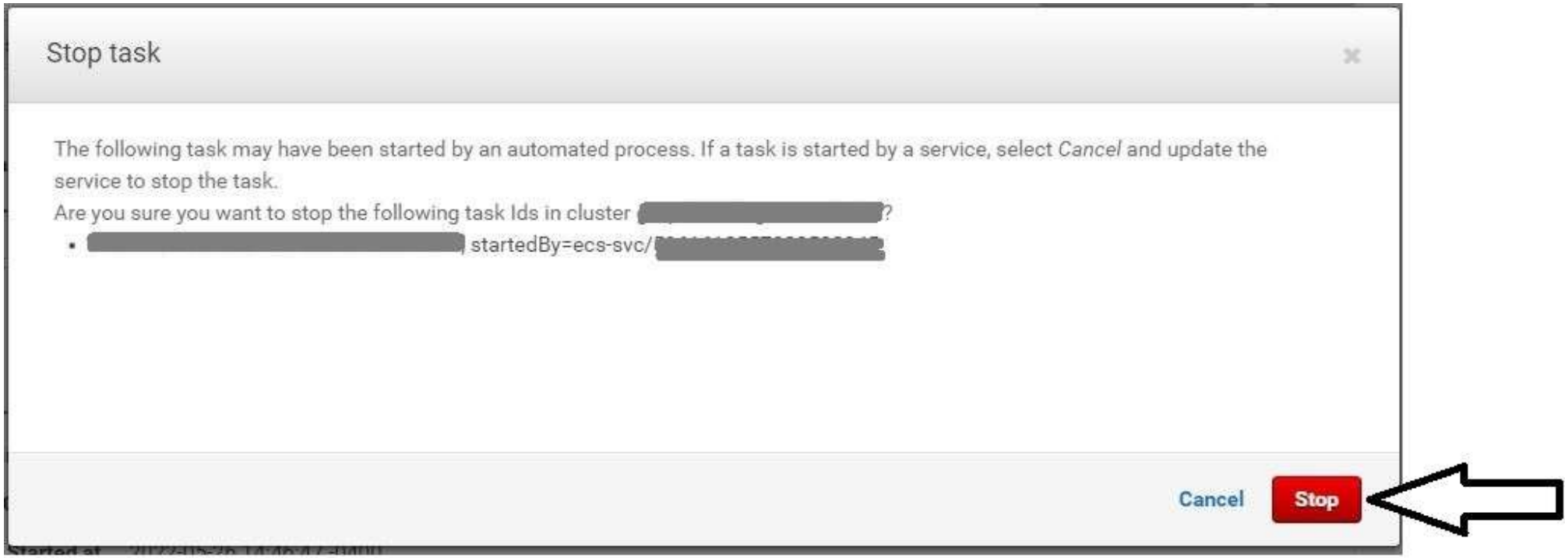
What happens if a task goes down?

- AWS will automatically bring up a new task to replace the one that stopped
- But… how long will that take?
- What if multiple tasks stop?
- How will your application behave in the interim?
- If there is impact, how long will it take your app to recover?

“Don’t Guess… Try it!

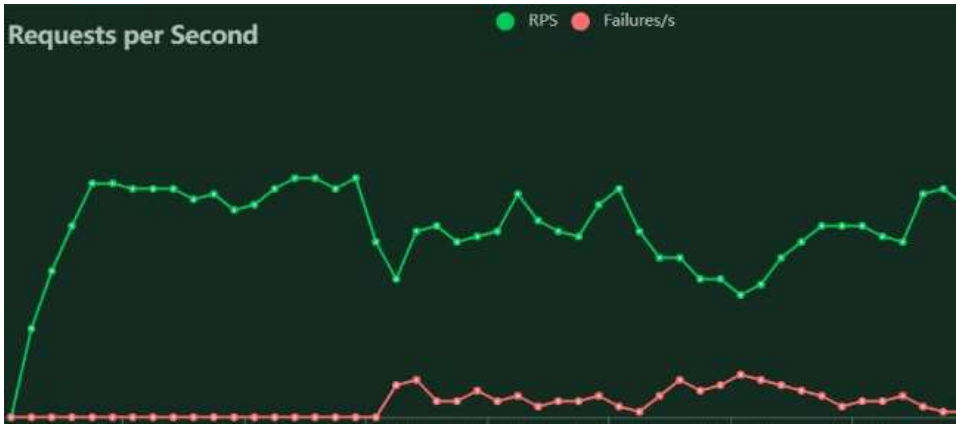
In Non-Prod”

Start Simple! (in non-prod ☺)



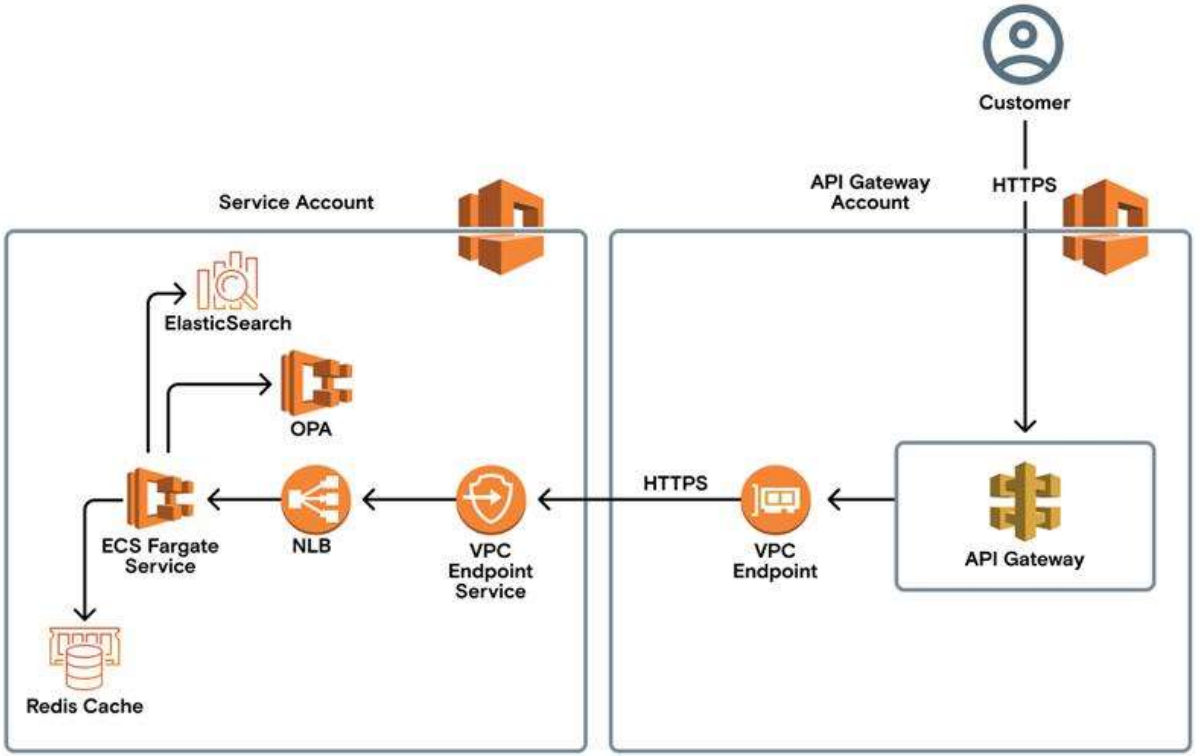
Simple Experiment
Valuable Findings

- ~20% error rate for for 3-10 minutes
- Failed requests returned 502 (bad gateway) errors
- System returned to steady state without outside intervention



Case Study
What happened?

1. 1/3 ECS tasks was down
2. NLB was still sending traffic to the “bad task” for a few minutes, until the health check failed



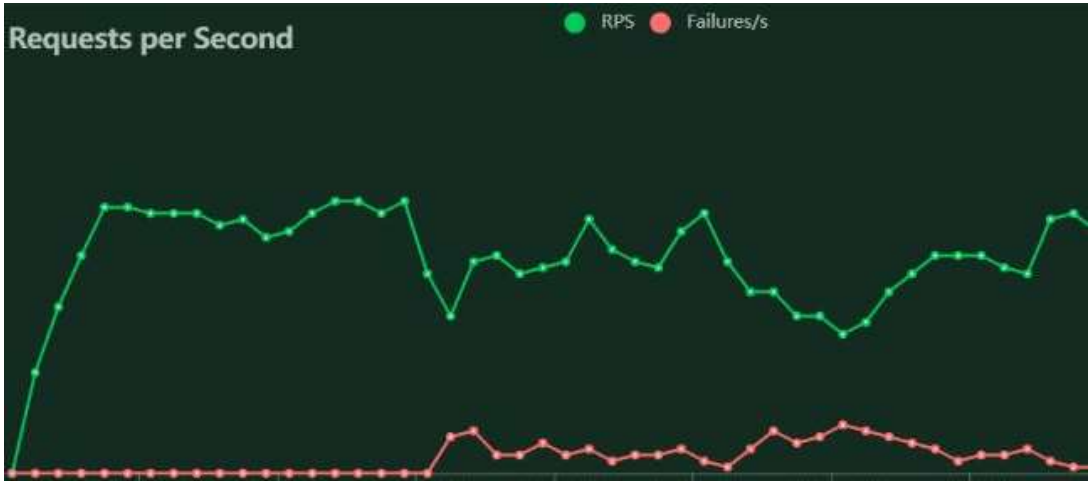
Increasing Resilience to ECS Task Failures

1. Increase the number of tasks (cost vs. resiliency trade-off)
2. Tune time-out of your load-balancer – remove tasks that are not responding more quickly

Fundamental Question
Would this be an Incident?

If this happened in prod, would it be an incident?

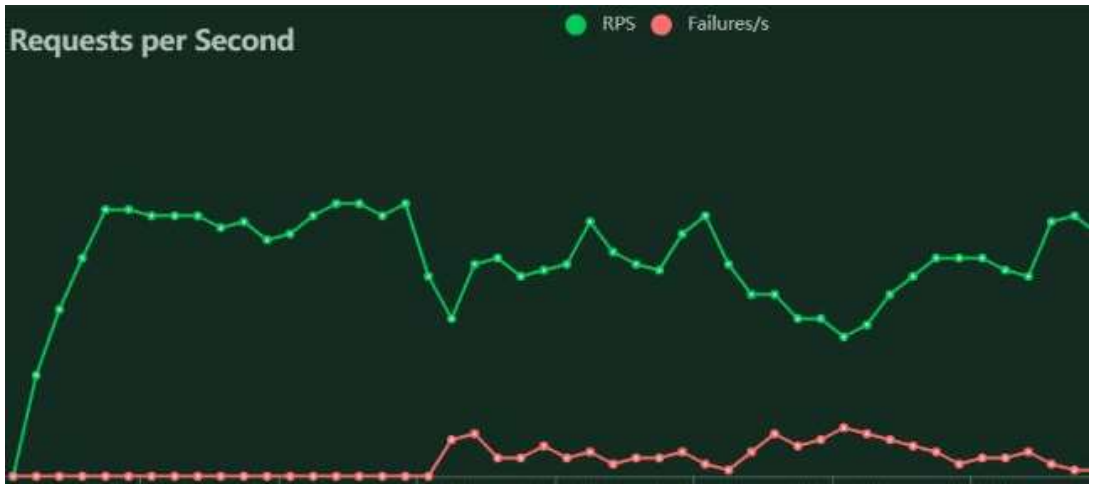
- It depends!
- What are your SLOs?



The Dashboard Dilemma

- Service dashboard:
- error rate = 0

- The customer experience (measured via Locust, a load testing tool)



The Dashboard Dilemma

- Dashboards were pulling data from an agent running alongside the ECS Task
- When the task went down, the agent stopped collecting data
- Therefore, the errors were not detected

- Lessons learned:
  - Make sure your monitoring is decoupled from your service!
  - *Test* your dashboards and alerting regularly
  - Monitor from your customer's perspective (synthetic probes)

Takeaways

1. Getting started with chaos engineering does not need to be difficult
2. Ensure your chaos tests align with real business requirements
3. After major incidents, strongly consider including a chaos test as a mandatory follow-up to prevent a recurrence

Goldman Sachs **Engineering**

https://www.goldmansachs.com/careers/blog/posts/engineering-tenets.html

4. Look Around Corners

Engineers often have to deal with uncertainty and the unknown. We put measures in place to capture signals of failure before they become real issues, and design resilient systems that deal with failure by design.

5. Innovate Incrementally

Engineers deconstruct bold and ambitions goals into manageable deliverables. We believe there is no substitute for real-world testing; therefore, we release and innovate frequently. We embrace simplicity and practicality as key design tenets.

8. Keep Learning

Engineers recognize that technology is always evolving at a rapid pace and that today's state of the art can quickly become obsolete. We do not chase trends, but look at repeatable patterns and solidify our understanding of those patterns through curiosity, experimenting and constant learning.

Sounds like a place you want to work? Join us! https://goldmansachs.com/careers/

Goldman
Sachs