Stress Test & Chaos

Diego Pacheco

About Me

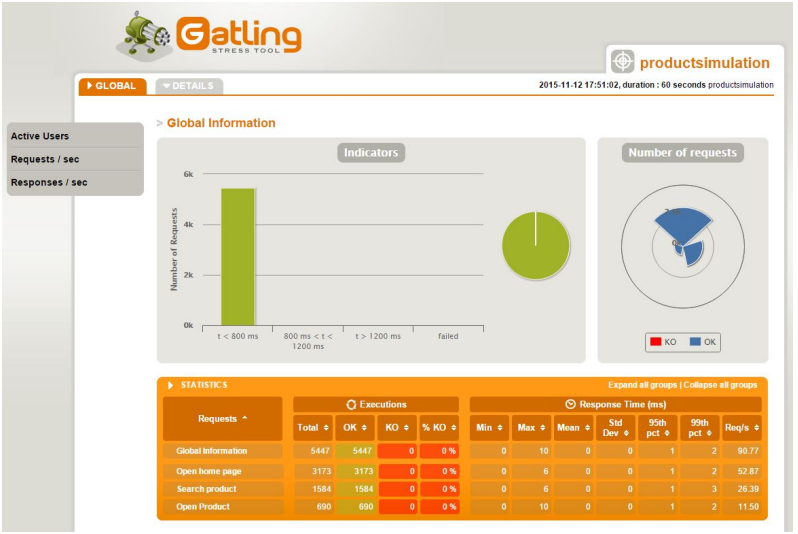
- Cat's Father
- Principal Software Architect
- Agile Coach
- SOA/Microservices Expert
- DevOps Practitioner
- Speaker
- Author

diegopacheco

@diego_pacheco

http://diego-pacheco.blogspot.com.br/

Stress Testing



Stress Testing

- Way better than JMETER(XML and Buggy UI)
- Scala DSL for Stress Testing
- Code /Config Re-use
- Very easy to stress test microservices - Or anything HTTP based
- Beautiful Reports with Metrics

Gatling Reports

STATISTICS

Expand all groups | Collapse all groups

Requests	Executions				Response Time (ms)						
	Total	OK	KO	% KO	Min	Max	Mean	Std Dev	95th pct	99th pct	Req/s
Global Information	9661	9659	2	0 %	94	1145	159	108	363	663	140.6
Home Redirect 1	2410	2410	0	0 %	94	1145	150	101	335	625	35.06
Search	1205	1205	0	0 %	97	965	156	106	366	656	17.53
Select	1205	1205	0	0 %	95	980	155	108	352	684	17.53
Page 0	1205	1205	0	0 %	100	1134	169	122	389	733	17.53
Page 1	1205	1205	0	0 %	100	1122	163	113	374	687	17.53
Page 2	1205	1205	0	0 %	100	992	165	110	367	677	17.53
Page 3	1205	1205	0	0 %	100	978	163	101	365	561	17.53
Form	7	7	0	0 %	98	512	194	148	453	500	0.10
Post Redirect 1	14	12	2	14 %	99	483	150	103	354	457	0.20

ERRORS

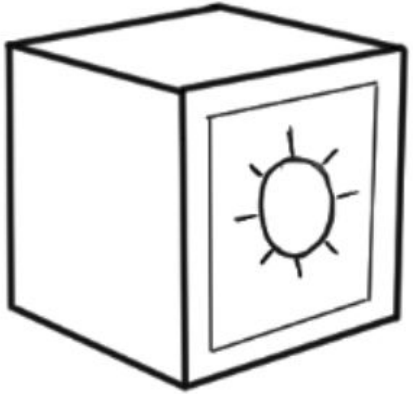
Error	Count	Percentage
status.is(201), but actually found 200	2	100.0 %

```scala
package weather

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import scala.concurrent.duration._

class WeatherServiceStressTest extends Simulation {

  val httpConf = http
    .baseURL("http://api.openweathermap.org")
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8")
    .doNotTrackHeader("1")
    .acceptLanguageHeader("en-US,en;q=0.5")
    .acceptEncodingHeader("gzip, deflate")
    .disableFollowRedirect
    .userAgentHeader("Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:16.0) Gecko/20100101 Firefox/16.0")

  val scn = scenario("Simple Stress Test")
          .exec(http("Http Round Robin 1")
            .get("/data/2.5/weather?q=London,uk")
            .check(status.is(200))
          )

  setUp(
      scn.inject(constantUsersPerSec(100).during(60 seconds))
  ).protocols(httpConf).assertions(global.responseTime.max.lessThan(1000))

}
```

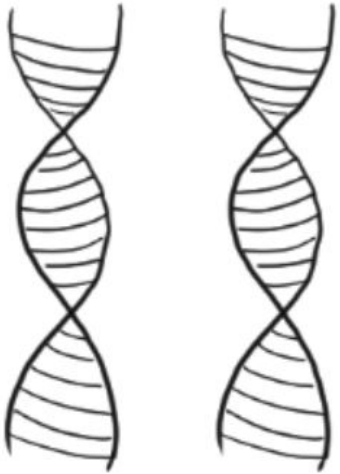

http://diego-pacheco.blogspot.com/2015/08/stress-test-with-gatling.html

Antifragile

FRAGILE
(HARMED
BY TENSION)

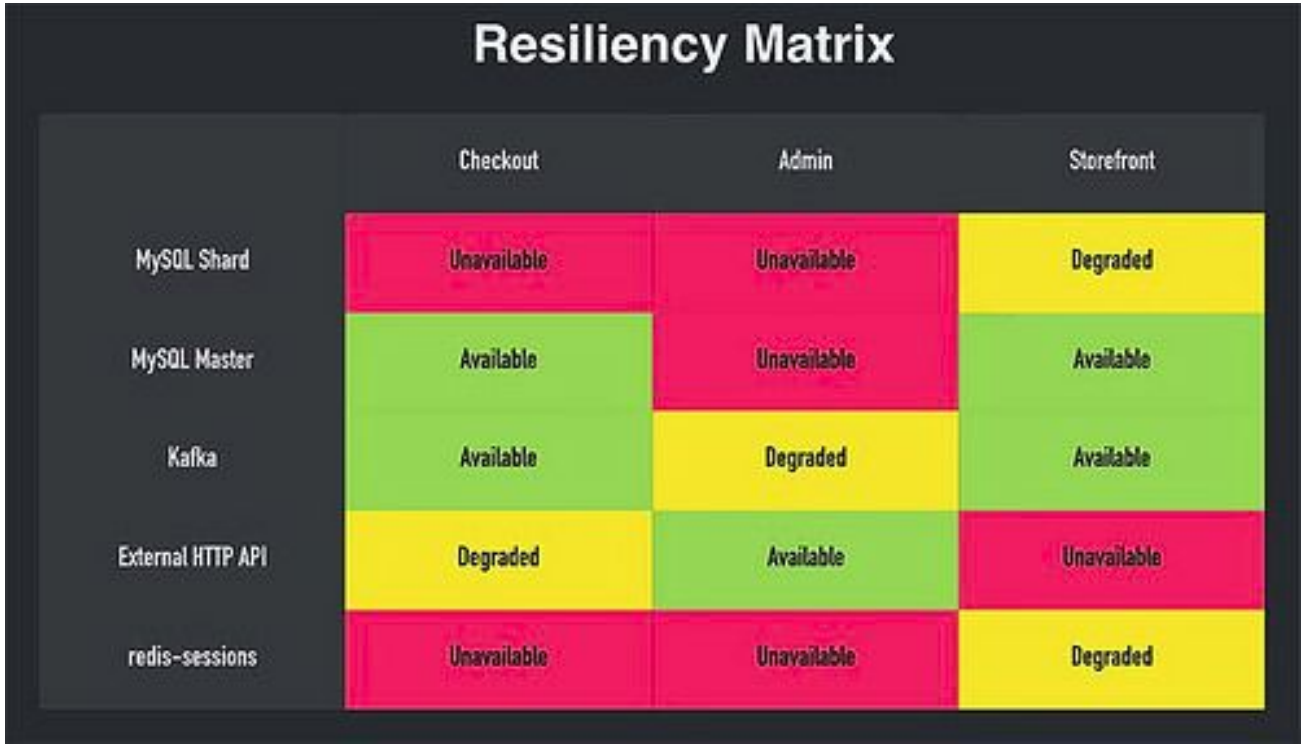


ROBUST
(STAYS SAME
UNDER TENSION)



ANTIFRAGILE
(BENEFITS
FROM TENSION)

Resilience Matrix



Chaos Principles

PRINCIPLES OF CHAOS ENGINEERING

Last Update: 2015 September

Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production.

Advances in large-scale, distributed software systems are changing the game for software engineering. As an industry, we are quick to adopt practices that increase flexibility of development and velocity of deployment. An urgent question follows on the heels of these benefits: How much confidence we can have in the complex systems that we put into production?

Even when all of the individual services in a distributed system are functioning properly, the interactions between those services can cause unpredictable outcomes. Unpredictable outcomes, compounded by rare but disruptive real-world events that affect production environments, make these distributed systems inherently chaotic.

We need to identify weaknesses before they manifest in system-wide, aberrant behaviors. Systemic weaknesses could take the form of: improper fallback settings when a service is unavailable; retry storms from improperly tuned timeouts; outages when a downstream dependency receives too much traffic; cascading failures when a single point of failure crashes; etc. We must address the most significant weaknesses proactively, before they affect our customers in production. We need a way to manage the chaos inherent in these systems, take advantage of increasing flexibility and velocity, and have confidence in our production deployments despite the complexity that they represent.

An empirical, systems-based approach addresses the chaos in distributed systems at scale and builds confidence in the ability of those systems to withstand realistic conditions. We learn about the behavior of a distributed system by observing it during a controlled experiment. We call this *Chaos Engineering*.

CHAOS IN PRACTICE

To specifically address the uncertainty of distributed systems at scale, Chaos Engineering can be thought of as the facilitation of experiments to uncover systemic weaknesses. These experiments follow four steps:

1. Start by defining 'steady state' as some measurable output of a system that indicates normal behavior.
2. Hypothesize that this steady state will continue in both the control group and the experimental group.
3. Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.
4. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

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.

Chaos engineering
is the discipline of **experimenting**
on a distributed system in order
to **build confidence** in the systems
capacity to withstand turbulent conditions
in production

Principles of Chaos Engineering

aws re:Invent

aws

NETFLIX Chaos Monkey

Simian Army

Chaos Monkey

- Simulates hard failures in AWS by killing a few instances per ASG (e.g. Auto Scale Group)
  - Similar to how EC2 instances can be killed by AWS with little warning
- Tests clients' ability to gracefully deal with broken connections, interrupted calls, etc…
- Verifies that all services are running within the protection of AWS Auto Scale Groups, which reincarnates killed instances
- If not, the Chaos monkey will win!

Conformity Monkey .

- Verifies that all services are running within the protection of AWS Auto Scale Groups, which reincarnates killed instances
- If not, app/service team is notified

NETFLIX

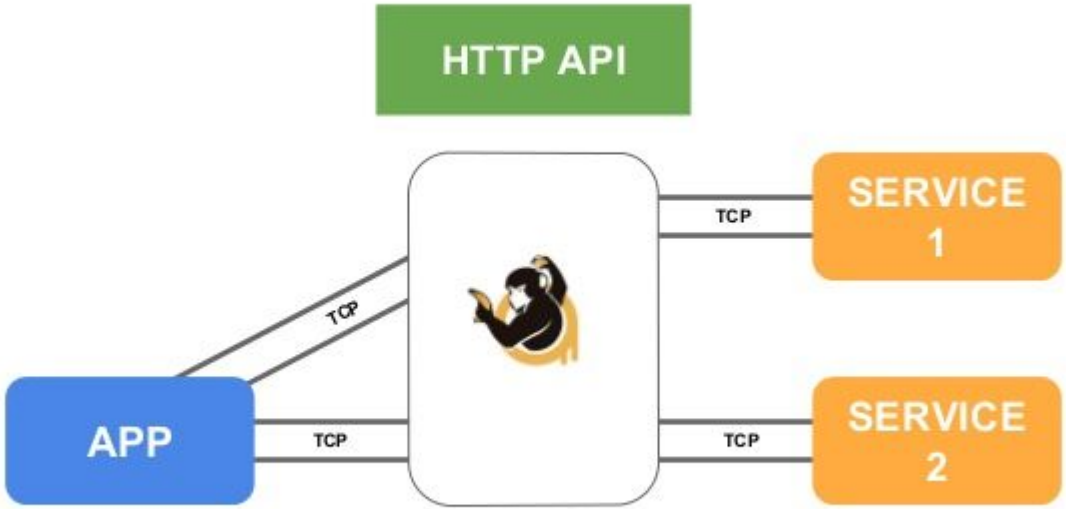
Simian Army by Netflix

Simian Army Projects

- Chaos Monkey
- Chaos Gorilla
- Chaos Kong
- Janitor Monkey
- Doctor Monkey
- Compliance Monkey
- Latency Monkey
- Security Monkey

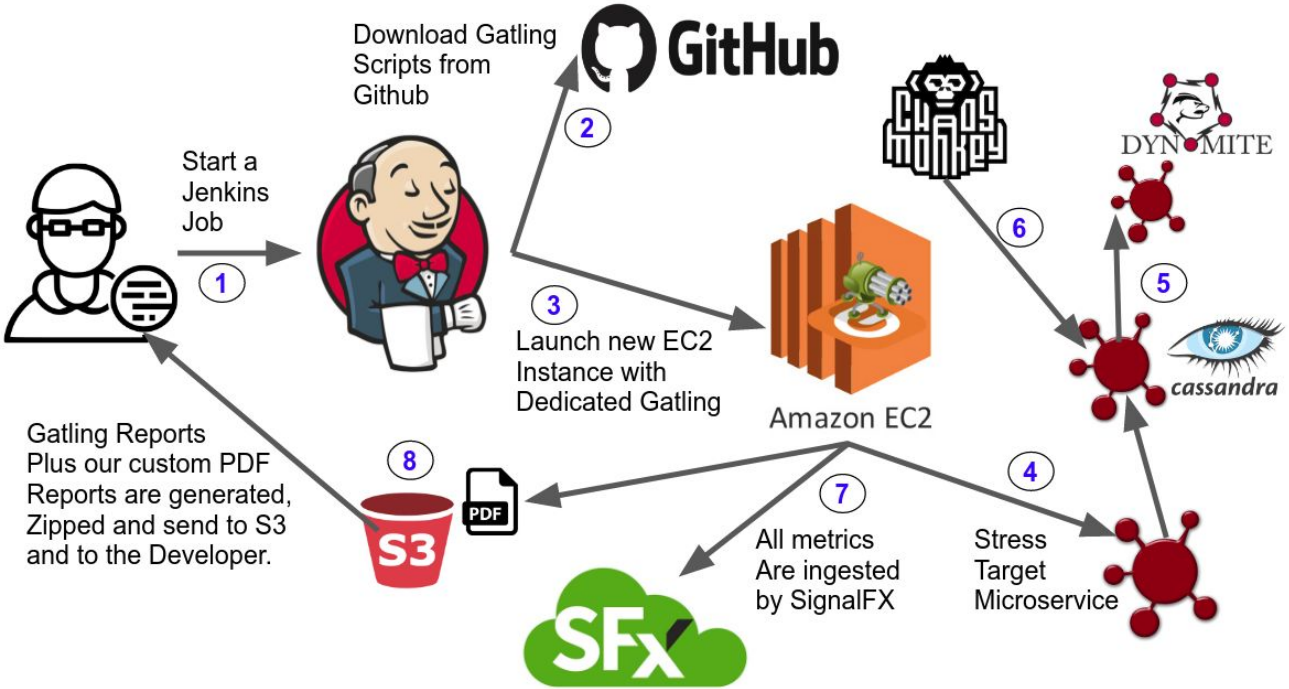
http://diego-pacheco.blogspot.com/2017/09/chaos-is-new-normal.html

Shopify Toxiproxy == Network Chaos



Gremlin = Paid Solution = Nice Chaos Reports

Personal Usa Case - Stress/Chaos Platform



http://diego-pacheco.blogspot.com/2018/06/experiences-building-running-stress.html

Stress Test & Chaos

Diego Pacheco