Stress Test & Chaos Engineering

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/

Diego Pacheco

Building Applications with Scala

Write modern, scalable, and reactive applications with the power of Scala

Packt>

Diego Pacheco

Building Effective Microservices

Explore microservices and their implementation hands-on

Packt>

https://diegopacheco.github.io/

Every engineer + manager think about

- □ FRPs(Functional Requirements) ~ Features
- □ Time / Productivity
- □ Business Logic that works
- □ But is it all...?

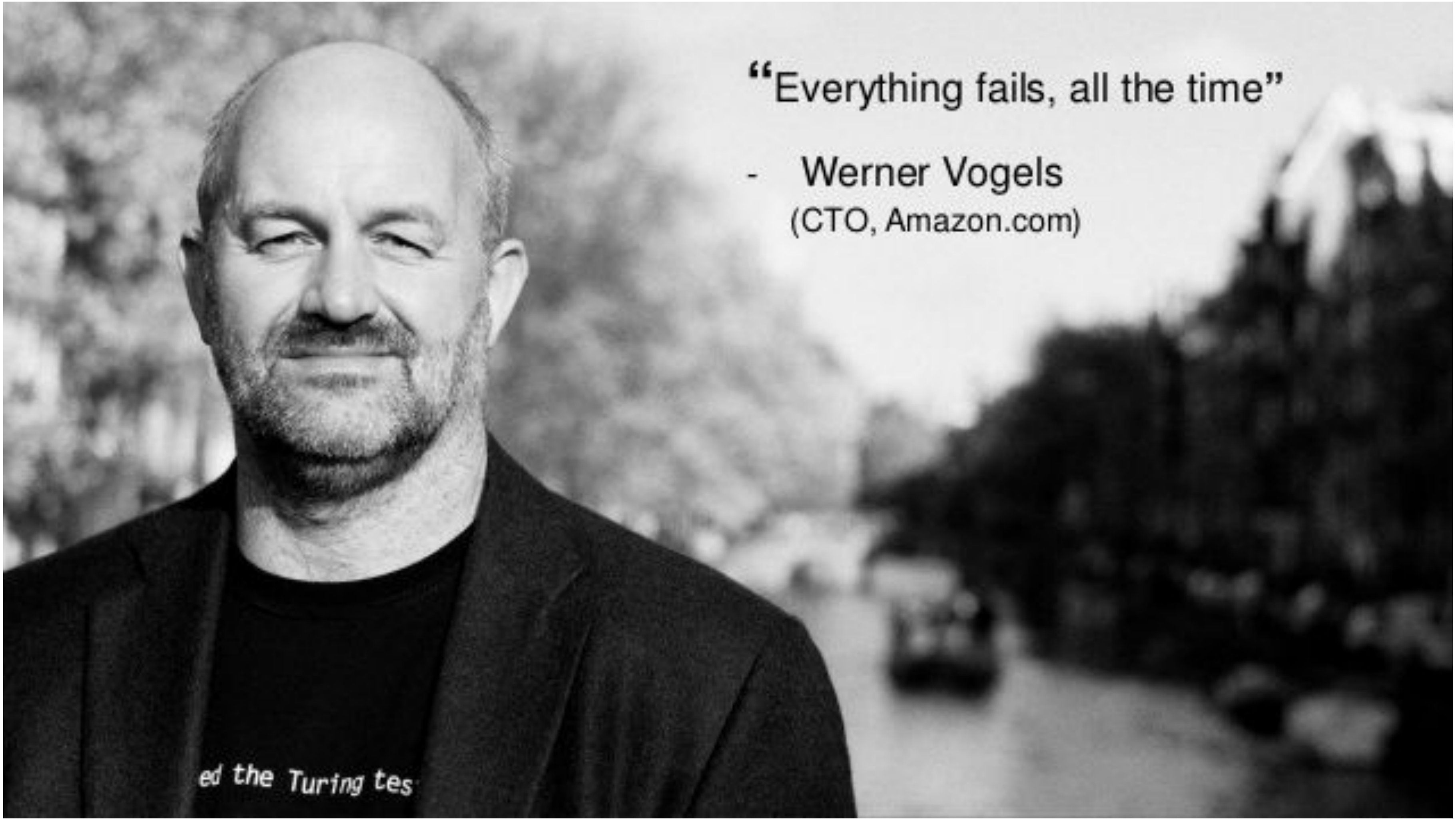
Scalability + Availability + Reliability

❑ As the business grows would the code continue working?

❑ Would the user experience be the same(getting slow)?

❑ Would be good for some users(p50) but few users really might have a bad experience (p99.9 & p99.99).

❑ Does the user trust the system? Lack of think About this 3 disciplines could really destroy Your brand really fast.

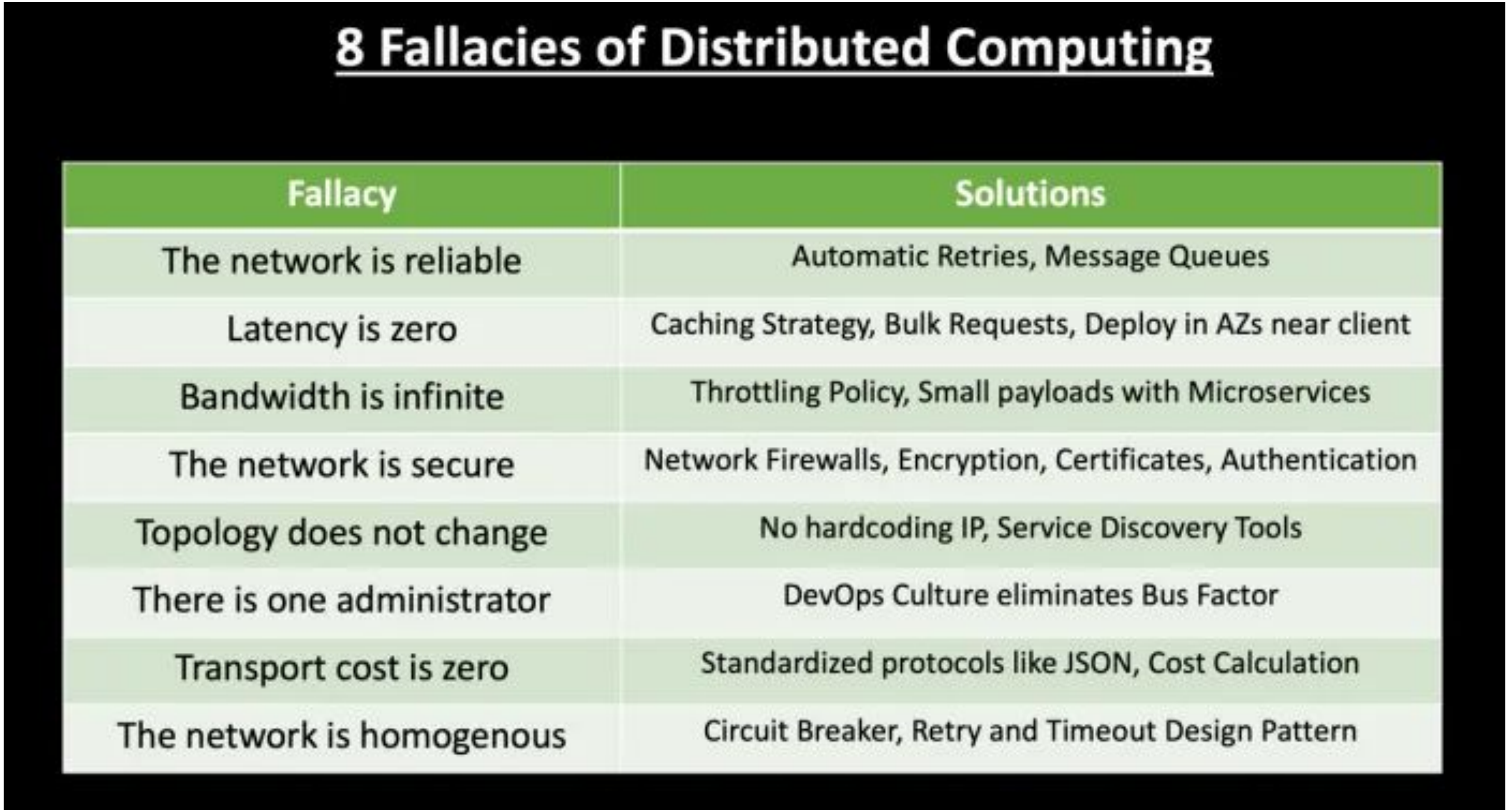


“Everything fails, all the time”

- Werner Vogels

(CTO, Amazon.com)

8 Fallacies and Actions...

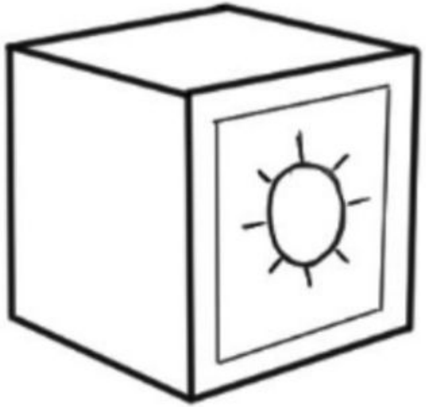


8 Fallacies of Distributed Computing

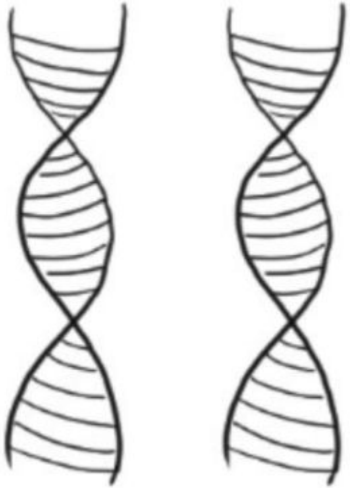
Fallacy	Solutions
The network is reliable	Automatic Retries, Message Queues
Latency is zero	Caching Strategy, Bulk Requests, Deploy in AZs near client
Bandwidth is infinite	Throttling Policy, Small payloads with Microservices
The network is secure	Network Firewalls, Encryption, Certificates, Authentication
Topology does not change	No hardcoding IP, Service Discovery Tools
There is one administrator	DevOps Culture eliminates Bus Factor
Transport cost is zero	Standardized protocols like JSON, Cost Calculation
The network is homogenous	Circuit Breaker, Retry and Timeout Design Pattern

Antifragile

FRAGILE
(HARMED
BY TENSION)

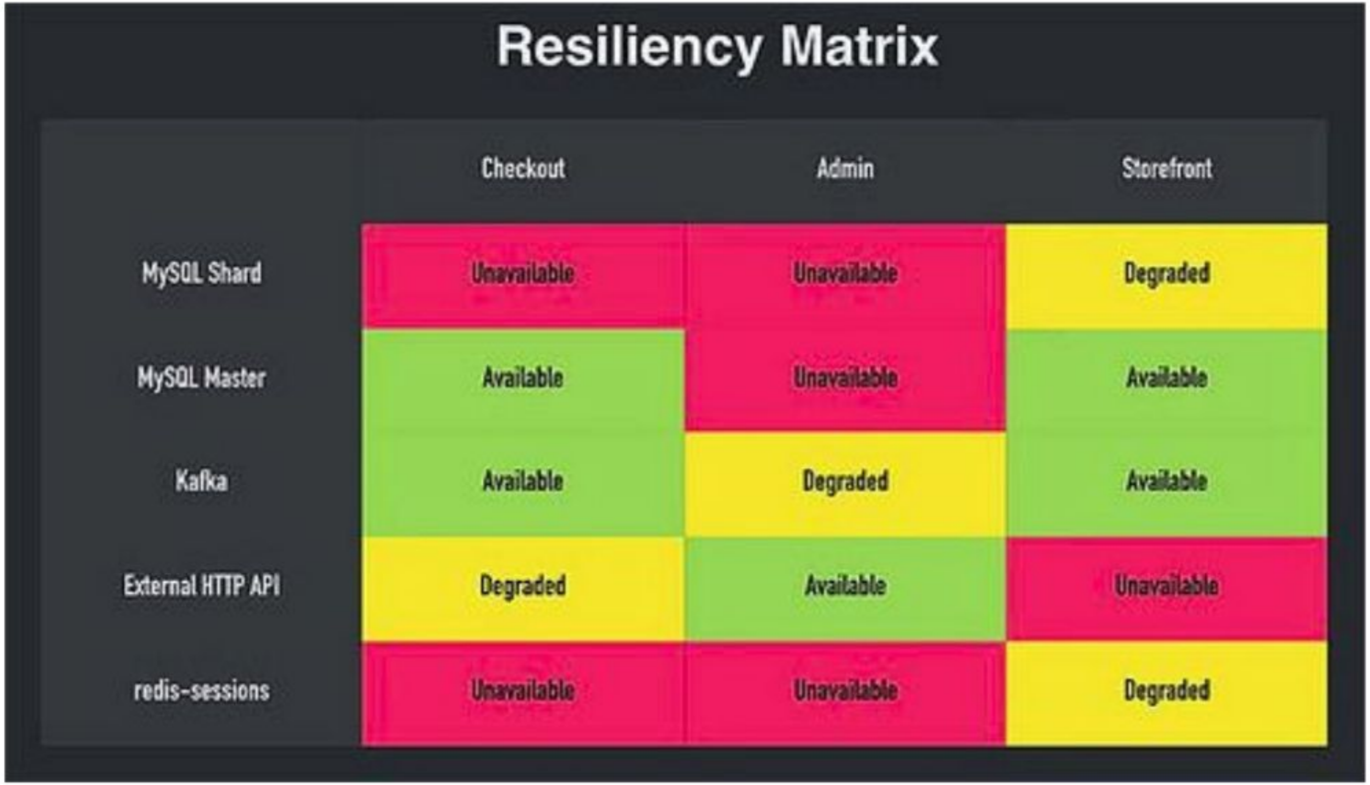


ROBUST
(STAYS SAME
UNDER TENSION)



ANTIFRAGILE
(BENEFITS
FROM TENSION)

Resilience Matrix



Resiliency Matrix

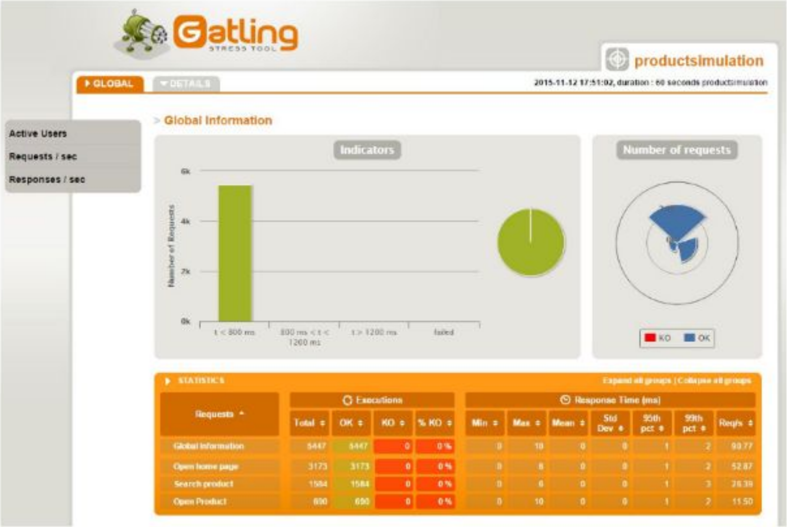
	Checkout	Admin	Storefront
MySQL Shard	Unavailable	Unavailable	Degraded
MySQL Master	Available	Unavailable	Available
Kafka	Available	Degraded	Available
External HTTP API	Degraded	Available	Unavailable
redis-sessions	Unavailable	Unavailable	Degraded

The Rise and Fall of fallbacks

- □ Hystrix
- □ Spring Cloud -> Resilience4J
- □ Fallback Issues:
  - □ Hard to Tests
  - □ Fallbacks fail
  - □ Lack of continuous testing
  - □ Fallbacks can make outage even worst
- □ Amazon Philosophy -> focus in code more resilient.

Erlang / Akka / Amazon Philosophy

Stress Testing



2015-11-12 17:51:02, duration : 60 seconds productsimulation

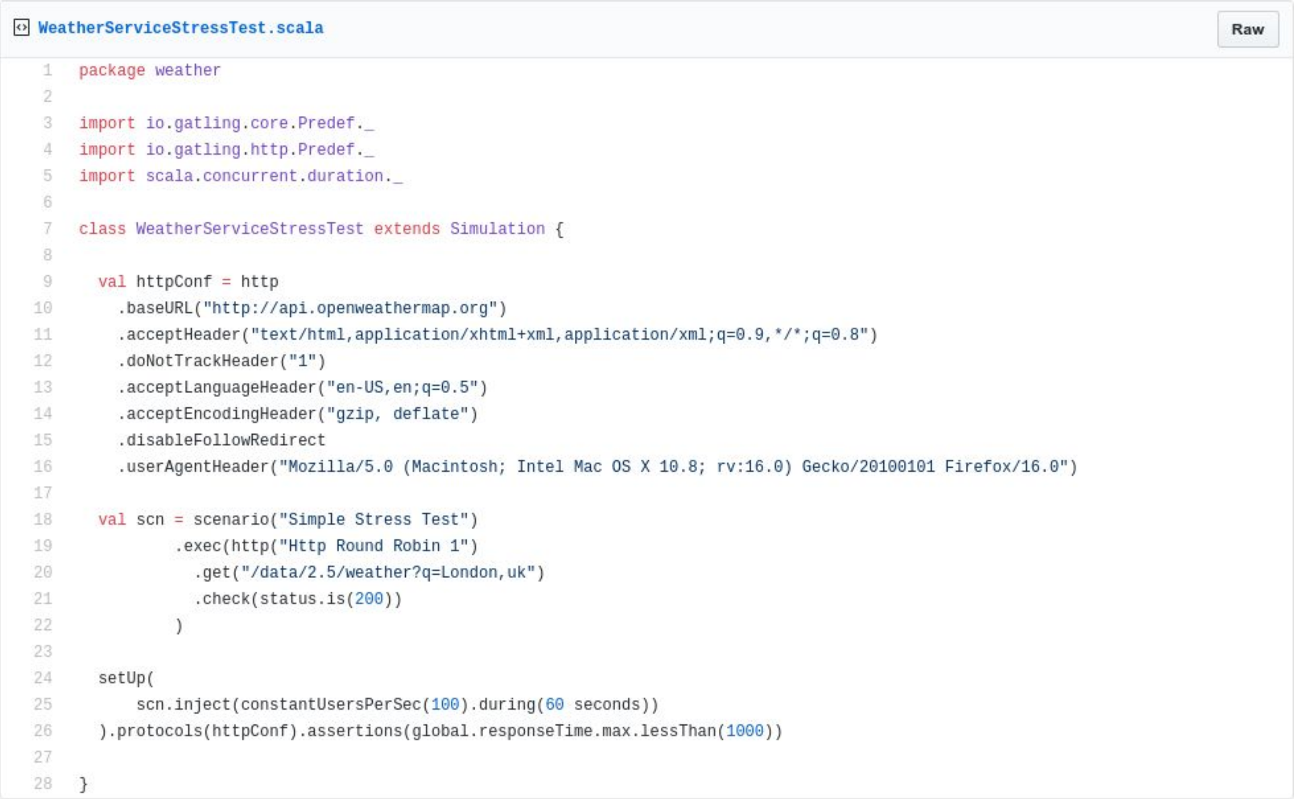
> Global Information

Requests ▴	Executions				Response Time (ms)						
	Total ⇕	OK ⇕	KO ⇕	% KO ⇕	Min ⇕	Max ⇕	Mean ⇕	Std Dev ⇕	95th pct ⇕	99th pct ⇕	Req/s ⇕
Global Information	5447	5447	0	0 %	0	10	0	0	1	2	90.77
Open home page	3173	3173	0	0 %	0	8	0	0	1	2	52.87
Search product	1584	1584	0	0 %	0	6	0	0	1	3	26.39
Open Product	690	690	0	0 %	0	10	0	0	1	2	11.50

How to do Proper Stress / Load Testing?

- ❑ Have Plan
  - ❑ What Service to Test? Why?
  - ❑ Select Endpoints to test (don't test them all)
  - ❑ Have Expectations in sense of Latency / Requests to Handle
- ❑ Know where your service break. Figure it out why.
- ❑ Test using batteries: 1,5,10,50,100,1k,2k,5k,10k,50k,100k,1M,100M...
- ❑ You must have observability. Dedicated Env is a must as well.
- ❑ Understand your metrics(which ones per service)
- ❑ Automate Stress Tests in your build pineline
- ❑ Have platform: It could be a jenkins job + scripts.

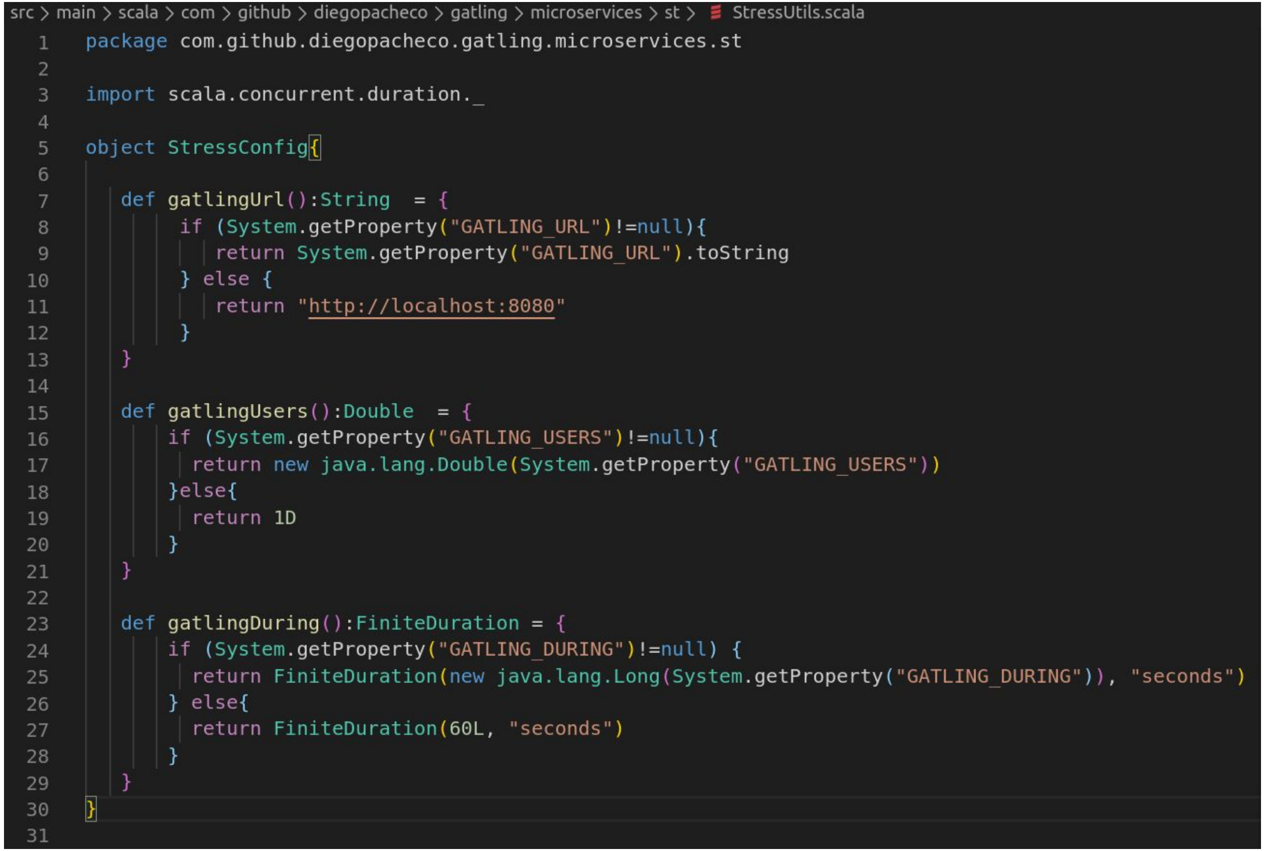
Stress / Load Testing with Gatling



```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))

}
```

https://gist.github.com/diegopacheco/faf7ceb2496e4ebdaded

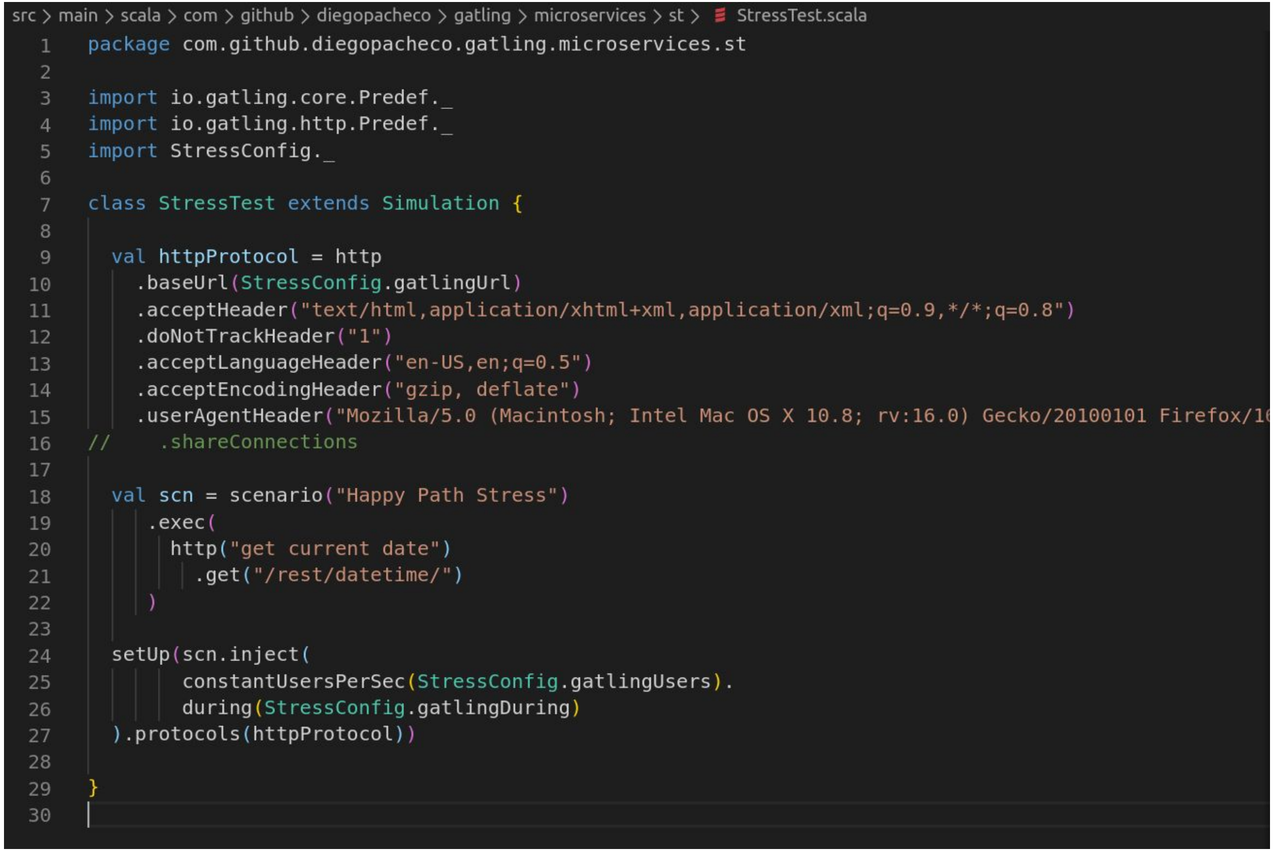
Stress / Load Testing with Gatling



src > main > scala > com > github > diegopacheco > gatling > microservices > st > StressUtils.scala

```scala
package com.github.diegopacheco.gatling.microservices.st

import scala.concurrent.duration._

object StressConfig{

   def gatlingUrl():String  = {
        if (System.getProperty("GATLING_URL")!=null){
           return System.getProperty("GATLING_URL").toString
        } else {
           return "http://localhost:8080"
        }
   }

   def gatlingUsers():Double  = {
       if (System.getProperty("GATLING_USERS")!=null){
         return new java.lang.Double(System.getProperty("GATLING_USERS"))
       }else{
         return 1D
       }
   }

   def gatlingDuring():FiniteDuration = {
       if (System.getProperty("GATLING_DURING")!=null) {
         return FiniteDuration(new java.lang.Long(System.getProperty("GATLING_DURING")), "seconds")
       } else{
         return FiniteDuration(60L, "seconds")
       }
   }
}
```

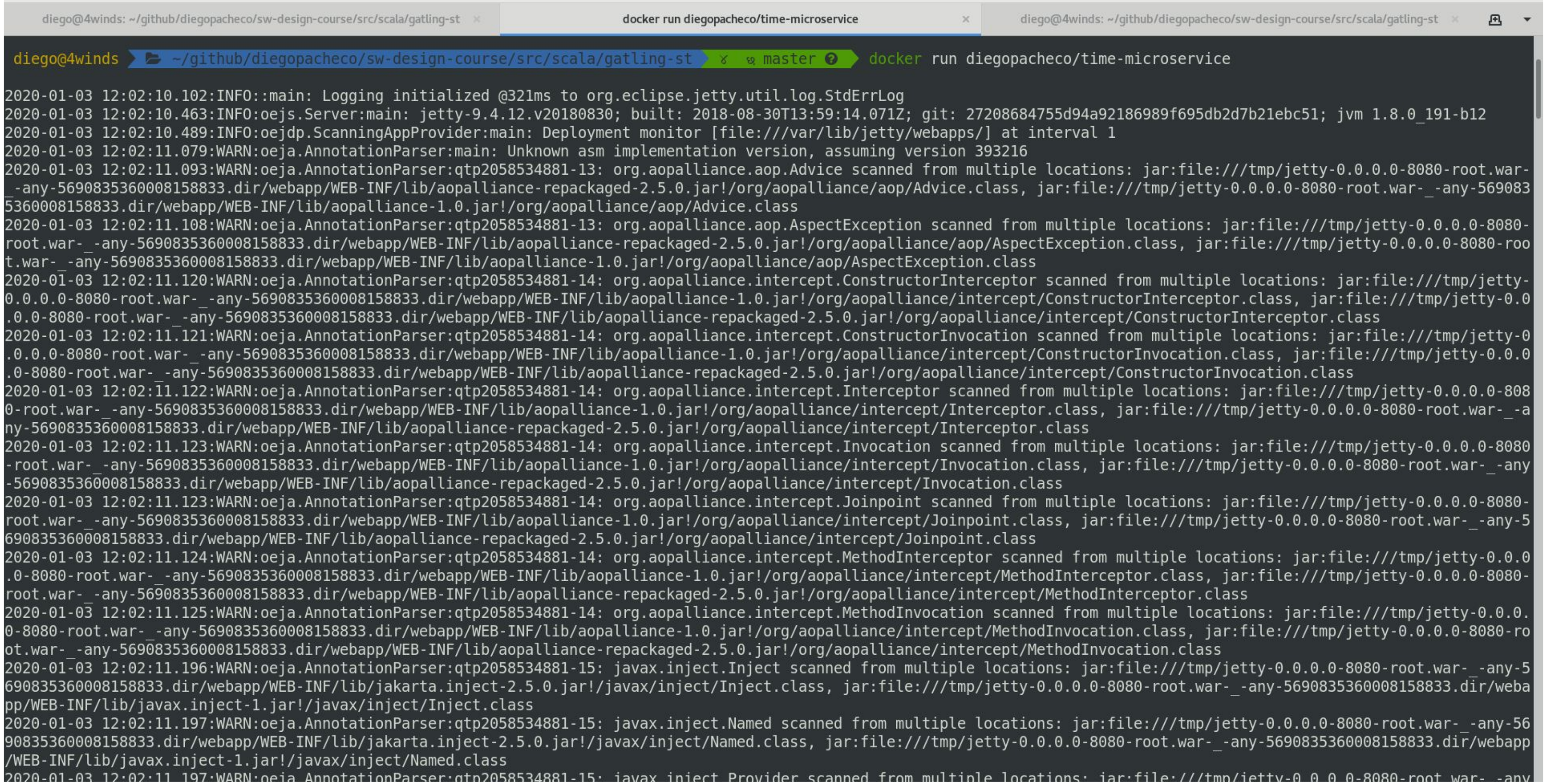
Stress / Load Testing with Gatling


src > main > scala > com > github > diegopacheco > gatling > microservices > st > StressTest.scala

```scala
package com.github.diegopacheco.gatling.microservices.st

import io.gatling.core.Predef._
import io.gatling.http.Predef._
import StressConfig._

class StressTest extends Simulation {

  val httpProtocol = http
    .baseUrl(StressConfig.gatlingUrl)
    .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")
    .userAgentHeader("Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:16.0) Gecko/20100101 Firefox/1
//    .shareConnections

  val scn = scenario("Happy Path Stress")
     .exec(
       http("get current date")
         .get("/rest/datetime/")
     )

  setUp(scn.inject(
        constantUsersPerSec(StressConfig.gatlingUsers).
        during(StressConfig.gatlingDuring)
  ).protocols(httpProtocol))

}
```

Stress / Load Testing with Gatling

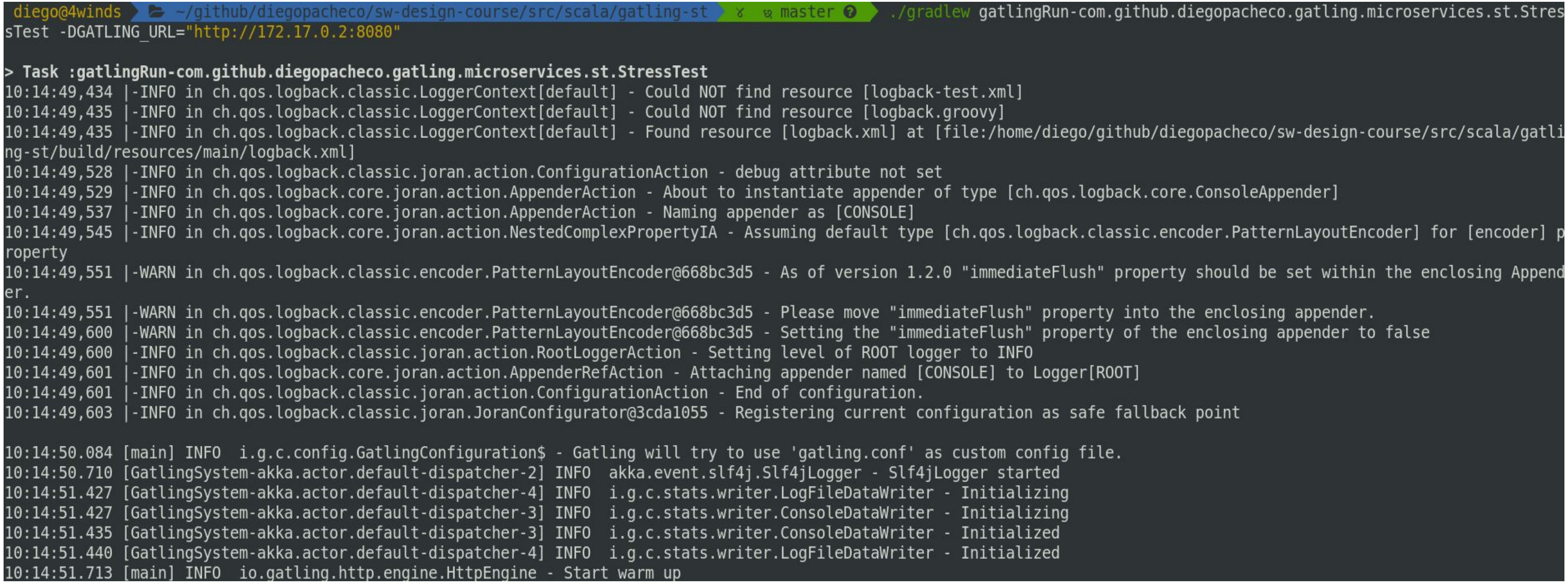
docker run diegopacheco/time-microservice


diego@4winds: ~/github/diegopacheco/sw-design-course/src/scala/gatling-st
docker run diegopacheco/time-microservice
diego@4winds: ~/github/diegopacheco/sw-design-course/src/scala/gatling-st

```
diego@4winds  ~/github/diegopacheco/sw-design-course/src/scala/gatling-st   master ?  docker run diegopacheco/time-microservice

2020-01-03 12:02:10.102:INFO::main: Logging initialized @321ms to org.eclipse.jetty.util.log.StdErrLog
2020-01-03 12:02:10.463:INFO:oejs.Server:main: jetty-9.4.12.v20180830; built: 2018-08-30T13:59:14.071Z; git: 27208684755d94a92186989f695db2d7b21ebc51; jvm 1.8.0_191-b12
2020-01-03 12:02:10.489:INFO:oejdp.ScanningAppProvider:main: Deployment monitor [file:///var/lib/jetty/webapps/] at interval 1
2020-01-03 12:02:11.079:WARN:oeja.AnnotationParser:main: Unknown asm implementation version, assuming version 393216
2020-01-03 12:02:11.093:WARN:oeja.AnnotationParser:qtp2058534881-13: org.aopalliance.aop.Advice scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-8080-root.war-
_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/aop/Advice.class, jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any-569083
5360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/aop/Advice.class
2020-01-03 12:02:11.108:WARN:oeja.AnnotationParser:qtp2058534881-13: org.aopalliance.aop.AspectException scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-8080-
root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/aop/AspectException.class, jar:file:///tmp/jetty-0.0.0.0-8080-roo
t.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/aop/AspectException.class
2020-01-03 12:02:11.120:WARN:oeja.AnnotationParser:qtp2058534881-14: org.aopalliance.intercept.ConstructorInterceptor scanned from multiple locations: jar:file:///tmp/jetty-
0.0.0.0-8080-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/intercept/ConstructorInterceptor.class, jar:file:///tmp/jetty-0.0
.0.0-8080-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/intercept/ConstructorInterceptor.class
2020-01-03 12:02:11.121:WARN:oeja.AnnotationParser:qtp2058534881-14: org.aopalliance.intercept.ConstructorInvocation scanned from multiple locations: jar:file:///tmp/jetty-0
.0.0.0-8080-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/intercept/ConstructorInvocation.class, jar:file:///tmp/jetty-0.0.0
.0-8080-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/intercept/ConstructorInvocation.class
2020-01-03 12:02:11.122:WARN:oeja.AnnotationParser:qtp2058534881-14: org.aopalliance.intercept.Interceptor scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-808
0-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/intercept/Interceptor.class, jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-a
ny-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/intercept/Interceptor.class
2020-01-03 12:02:11.123:WARN:oeja.AnnotationParser:qtp2058534881-14: org.aopalliance.intercept.Invocation scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-8080
-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/intercept/Invocation.class, jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any
-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/intercept/Invocation.class
2020-01-03 12:02:11.123:WARN:oeja.AnnotationParser:qtp2058534881-14: org.aopalliance.intercept.Joinpoint scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-8080-
root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/intercept/Joinpoint.class, jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any-5
690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/intercept/Joinpoint.class
2020-01-03 12:02:11.124:WARN:oeja.AnnotationParser:qtp2058534881-14: org.aopalliance.intercept.MethodInterceptor scanned from multiple locations: jar:file:///tmp/jetty-0.0.0
.0-8080-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/intercept/MethodInterceptor.class, jar:file:///tmp/jetty-0.0.0.0-8080-
root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/intercept/MethodInterceptor.class
2020-01-03 12:02:11.125:WARN:oeja.AnnotationParser:qtp2058534881-14: org.aopalliance.intercept.MethodInvocation scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.
0-8080-root.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-1.0.jar!/org/aopalliance/intercept/MethodInvocation.class, jar:file:///tmp/jetty-0.0.0.0-8080-ro
ot.war-_-any-5690835360008158833.dir/webapp/WEB-INF/lib/aopalliance-repackaged-2.5.0.jar!/org/aopalliance/intercept/MethodInvocation.class
2020-01-03 12:02:11.196:WARN:oeja.AnnotationParser:qtp2058534881-15: javax.inject.Inject scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any-5
690835360008158833.dir/webapp/WEB-INF/lib/jakarta.inject-2.5.0.jar!/javax/inject/Inject.class, jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any-5690835360008158833.dir/weba
pp/WEB-INF/lib/javax.inject-1.jar!/javax/inject/Inject.class
2020-01-03 12:02:11.197:WARN:oeja.AnnotationParser:qtp2058534881-15: javax.inject.Named scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any-56
90835360008158833.dir/webapp/WEB-INF/lib/jakarta.inject-2.5.0.jar!/javax/inject/Named.class, jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any-5690835360008158833.dir/webapp
/WEB-INF/lib/javax.inject-1.jar!/javax/inject/Named.class
2020-01-03 12:02:11.197:WARN:oeja.AnnotationParser:qtp2058534881-15: javax.inject.Provider scanned from multiple locations: jar:file:///tmp/jetty-0.0.0.0-8080-root.war-_-any
```

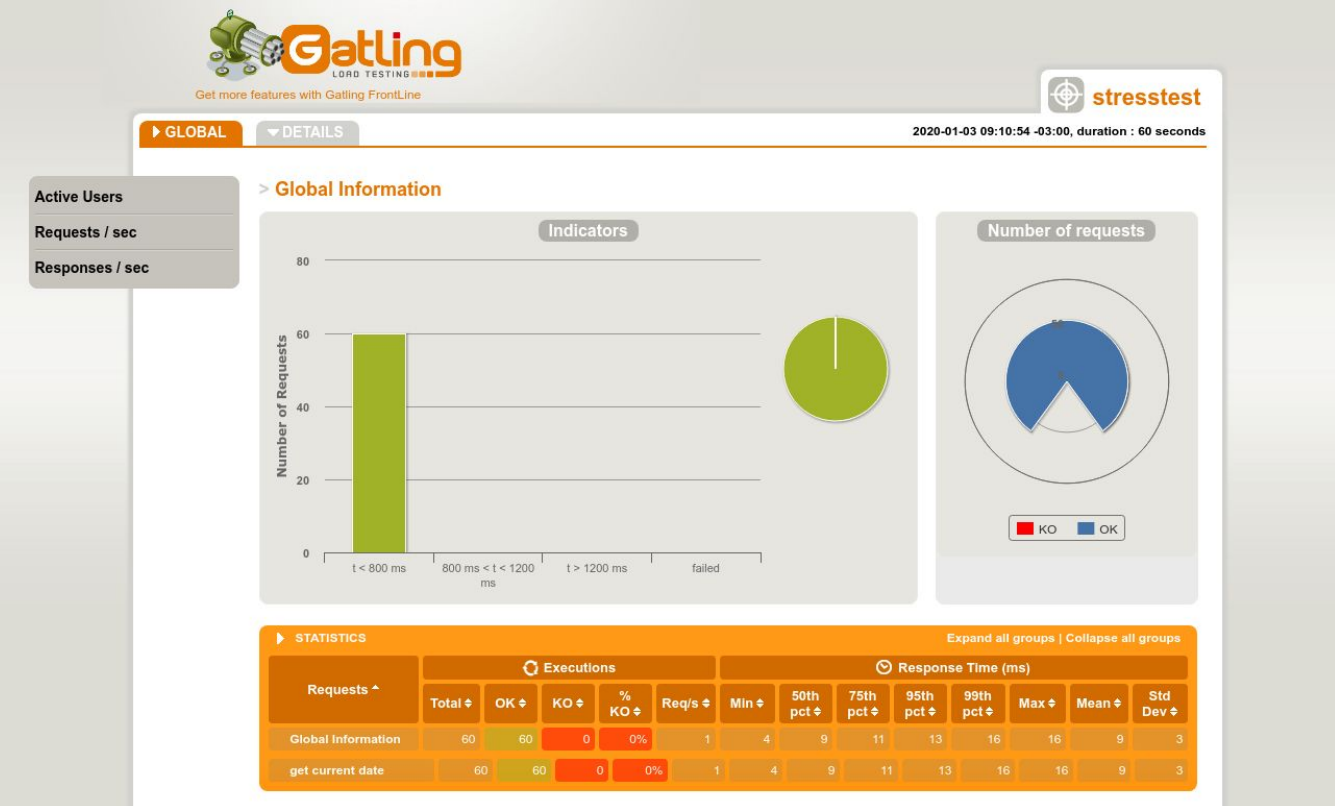
Stress / Load Testing with Gatling

```
./gradlew gatlingRun-com.github.diegopacheco.gatling.microservices.st.StressTest
-DGATLING_URL="http://172.17.0.2:8080"
```



```
diego@4winds  ~/github/diegopacheco/sw-design-course/src/scala/gatling-st  master  ./gradlew gatlingRun-com.github.diegopacheco.gatling.microservices.st.Stres
sTest -DGATLING_URL="http://172.17.0.2:8080"

> Task :gatlingRun-com.github.diegopacheco.gatling.microservices.st.StressTest
10:14:49,434 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback-test.xml]
10:14:49,435 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback.groovy]
10:14:49,435 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Found resource [logback.xml] at [file:/home/diego/github/diegopacheco/sw-design-course/src/scala/gatli
ng-st/build/resources/main/logback.xml]
10:14:49,528 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - debug attribute not set
10:14:49,529 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - About to instantiate appender of type [ch.qos.logback.core.ConsoleAppender]
10:14:49,537 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - Naming appender as [CONSOLE]
10:14:49,545 |-INFO in ch.qos.logback.core.joran.action.NestedComplexPropertyIA - Assuming default type [ch.qos.logback.classic.encoder.PatternLayoutEncoder] for [encoder] p
roperty
10:14:49,551 |-WARN in ch.qos.logback.classic.encoder.PatternLayoutEncoder@668bc3d5 - As of version 1.2.0 "immediateFlush" property should be set within the enclosing Append
er.
10:14:49,551 |-WARN in ch.qos.logback.classic.encoder.PatternLayoutEncoder@668bc3d5 - Please move "immediateFlush" property into the enclosing appender.
10:14:49,600 |-WARN in ch.qos.logback.classic.encoder.PatternLayoutEncoder@668bc3d5 - Setting the "immediateFlush" property of the enclosing appender to false
10:14:49,600 |-INFO in ch.qos.logback.classic.joran.action.RootLoggerAction - Setting level of ROOT logger to INFO
10:14:49,601 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appender named [CONSOLE] to Logger[ROOT]
10:14:49,601 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - End of configuration.
10:14:49,603 |-INFO in ch.qos.logback.classic.joran.JoranConfigurator@3cda1055 - Registering current configuration as safe fallback point

10:14:50.084 [main] INFO  i.g.c.config.GatlingConfiguration$ - Gatling will try to use 'gatling.conf' as custom config file.
10:14:50.710 [GatlingSystem-akka.actor.default-dispatcher-2] INFO  akka.event.slf4j.Slf4jLogger - Slf4jLogger started
10:14:51.427 [GatlingSystem-akka.actor.default-dispatcher-4] INFO  i.g.c.stats.writer.LogFileDataWriter - Initializing
10:14:51.427 [GatlingSystem-akka.actor.default-dispatcher-3] INFO  i.g.c.stats.writer.ConsoleDataWriter - Initializing
10:14:51.435 [GatlingSystem-akka.actor.default-dispatcher-3] INFO  i.g.c.stats.writer.ConsoleDataWriter - Initialized
10:14:51.440 [GatlingSystem-akka.actor.default-dispatcher-4] INFO  i.g.c.stats.writer.LogFileDataWriter - Initialized
10:14:51.713 [main] INFO  io.gatling.http.engine.HttpEngine - Start warm up
```

Stress / Load Testing with Gatling



2020-01-03 09:10:54 -03:00, duration : 60 seconds

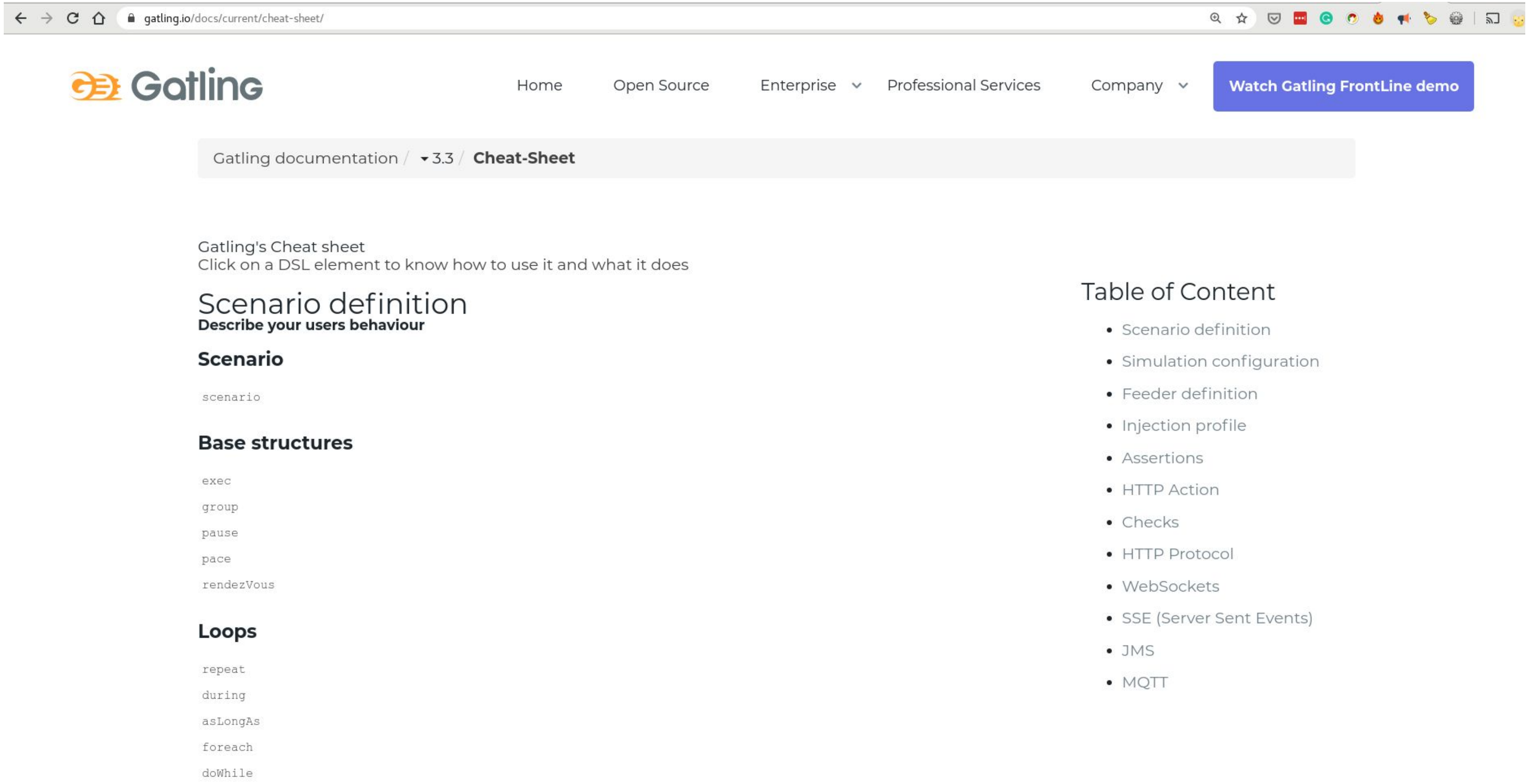
Global Information

STATISTICS

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	60	60	0	0%	1	4	9	11	13	16	16	9	3
get current date	60	60	0	0%	1	4	9	11	13	16	16	9	3

Stress / Load Testing with Gatling


gatling.io/docs/current/cheat-sheet/
Gatling
Home
Open Source
Enterprise
Professional Services
Company
Watch Gatling FrontLine demo

Gatling documentation / 3.3 / **Cheat-Sheet**

Gatling's Cheat sheet
Click on a DSL element to know how to use it and what it does

Scenario definition

Describe your users behaviour

Scenario

`scenario`

Base structures

`exec`

`group`

`pause`

`pace`

`rendezVous`

Loops

`repeat`

`during`

`asLongAs`

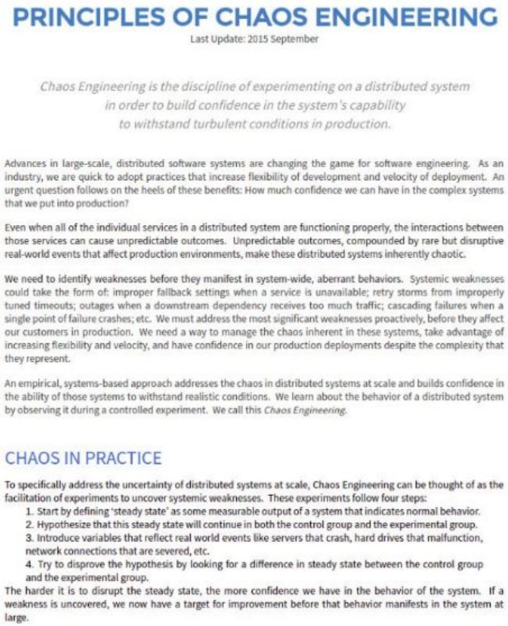
`foreach`

`doWhile`

Table of Content

https://gatling.io/docs/current/cheat-sheet/

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.

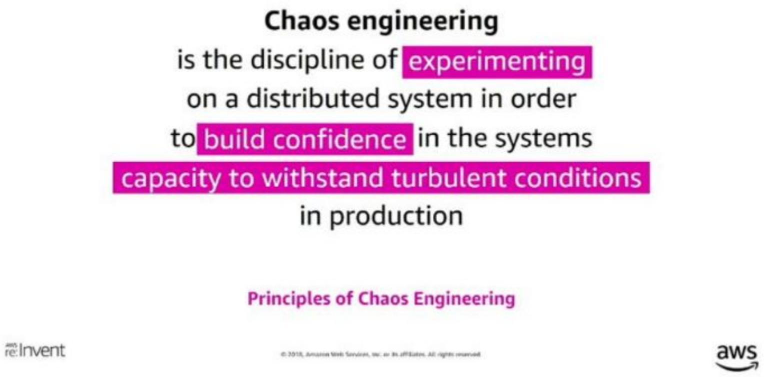
An empirical, systems-based approach addresses the chaos in distributed systems at scale and builds confidence in the ability of those systems to withstand realistic conditions. We learn about the behavior of a distributed system by observing it during a controlled experiment. We call this *Chaos Engineering*.

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.



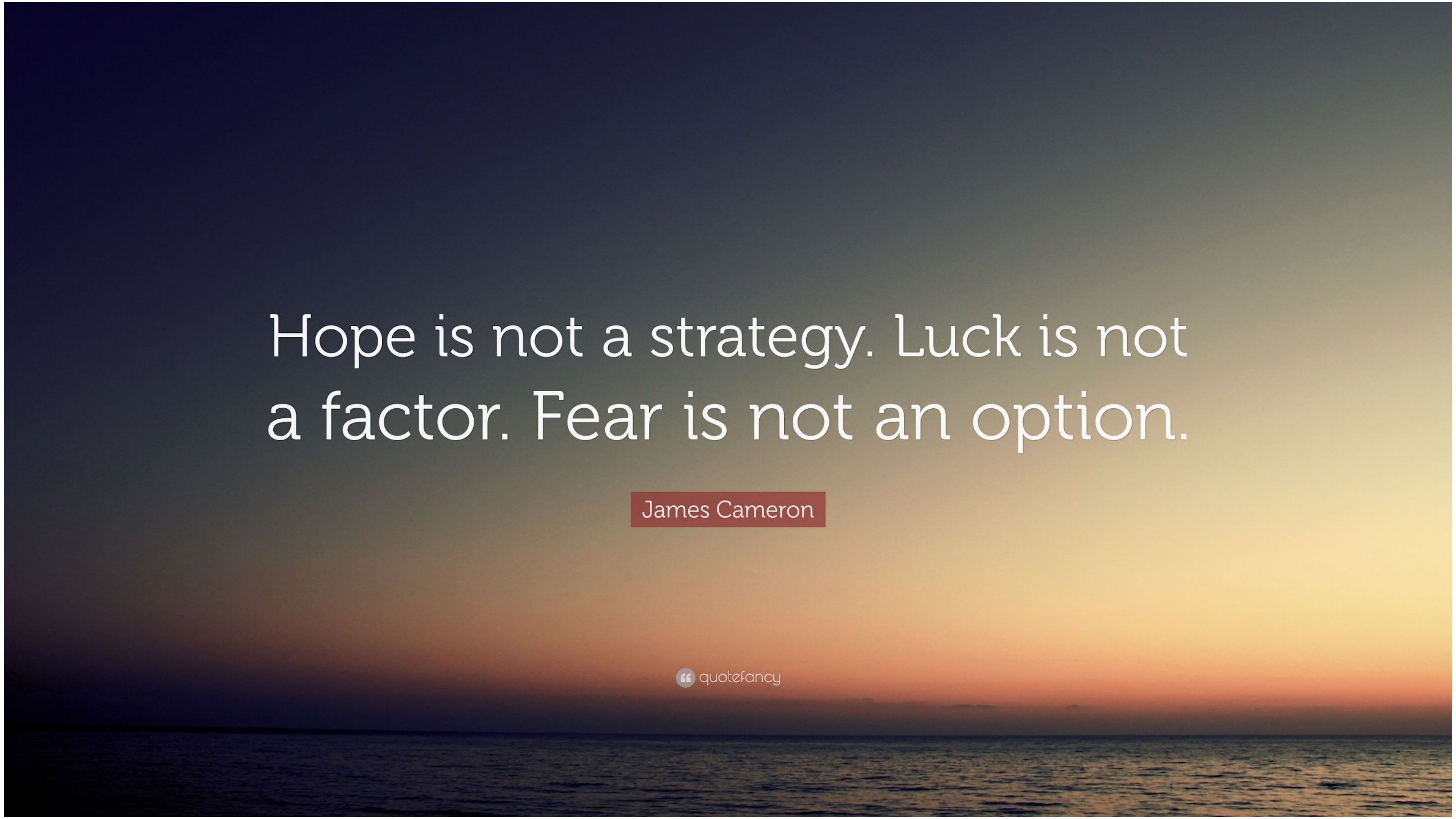
Chaos engineering
is the discipline of experimenting
on a distributed system in order
to build confidence in the systems
capacity to withstand turbulent conditions
in production

Principles of Chaos Engineering

re:Invent

aws

Chaos



Hope is not a strategy. Luck is not a factor. Fear is not an option.

James Cameron

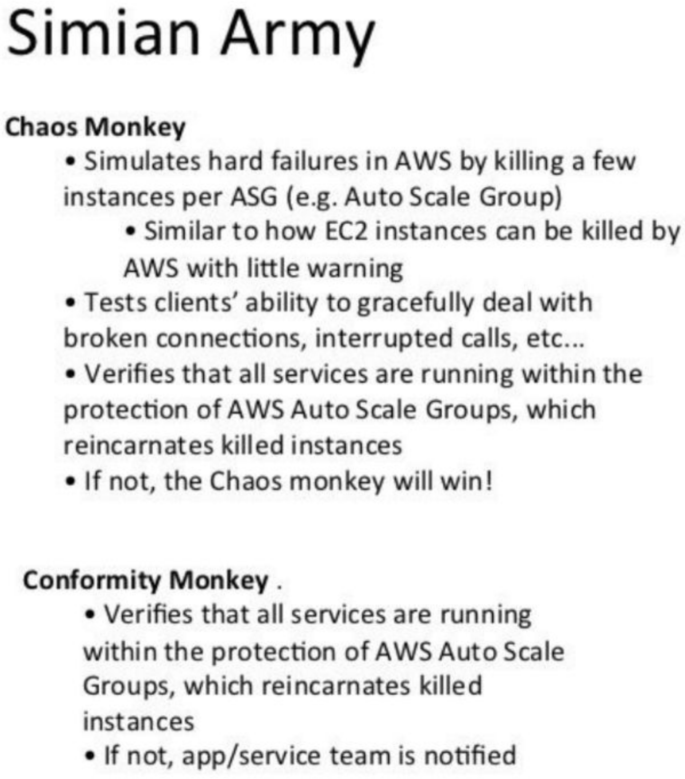
quotefancy

Chaos

- ❑ Test your Infrastructure
  - ❑ All ASG in place?
  - ❑ Does the failover to other: Instance, AZ, Region works?
- ❑ Test your clusters:
  - ❑ SQL / NoSQL / NewSQL
- ❑ Test your microservices downstream dependencies
  - ❑ Timeouts
  - ❑ Retries / Exponential backoff + Jitter
- ❑ Chaos Inside a Box
  - ❑ DISK, CPU, Memory, Metadata...

Chaos

Chaos



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!

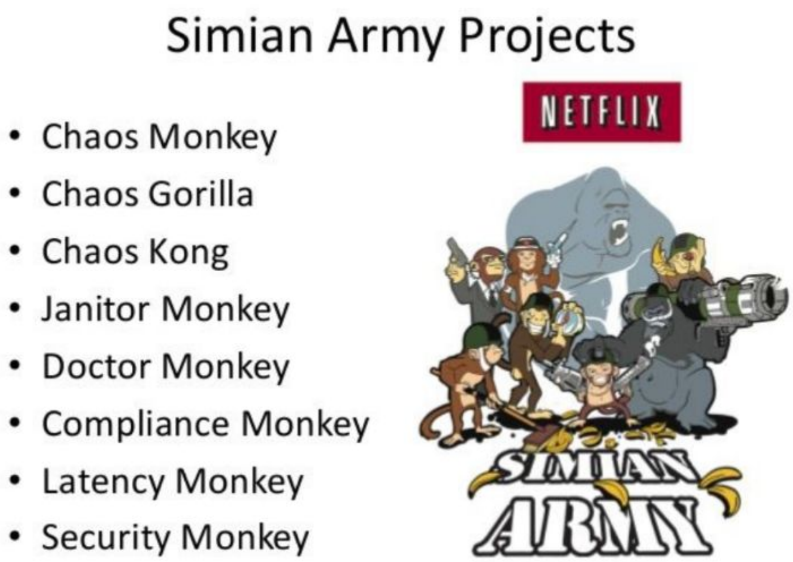
Conformity Monkey .

- 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

Chaos

Simian Army by Netflix



Simian Army Projects

NETFLIX

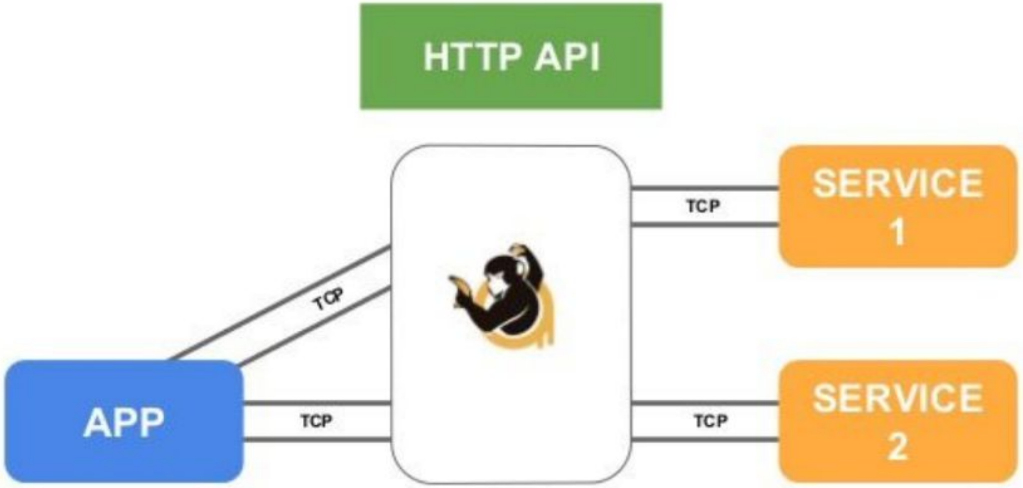
- Chaos Monkey
- Chaos Gorilla
- Chaos Kong
- Janitor Monkey
- Doctor Monkey
- Compliance Monkey
- Latency Monkey
- Security Monkey

SIMIAN
ARMY

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

Chaos

Shopify Toxiproxy == Network Chaos



Chaos

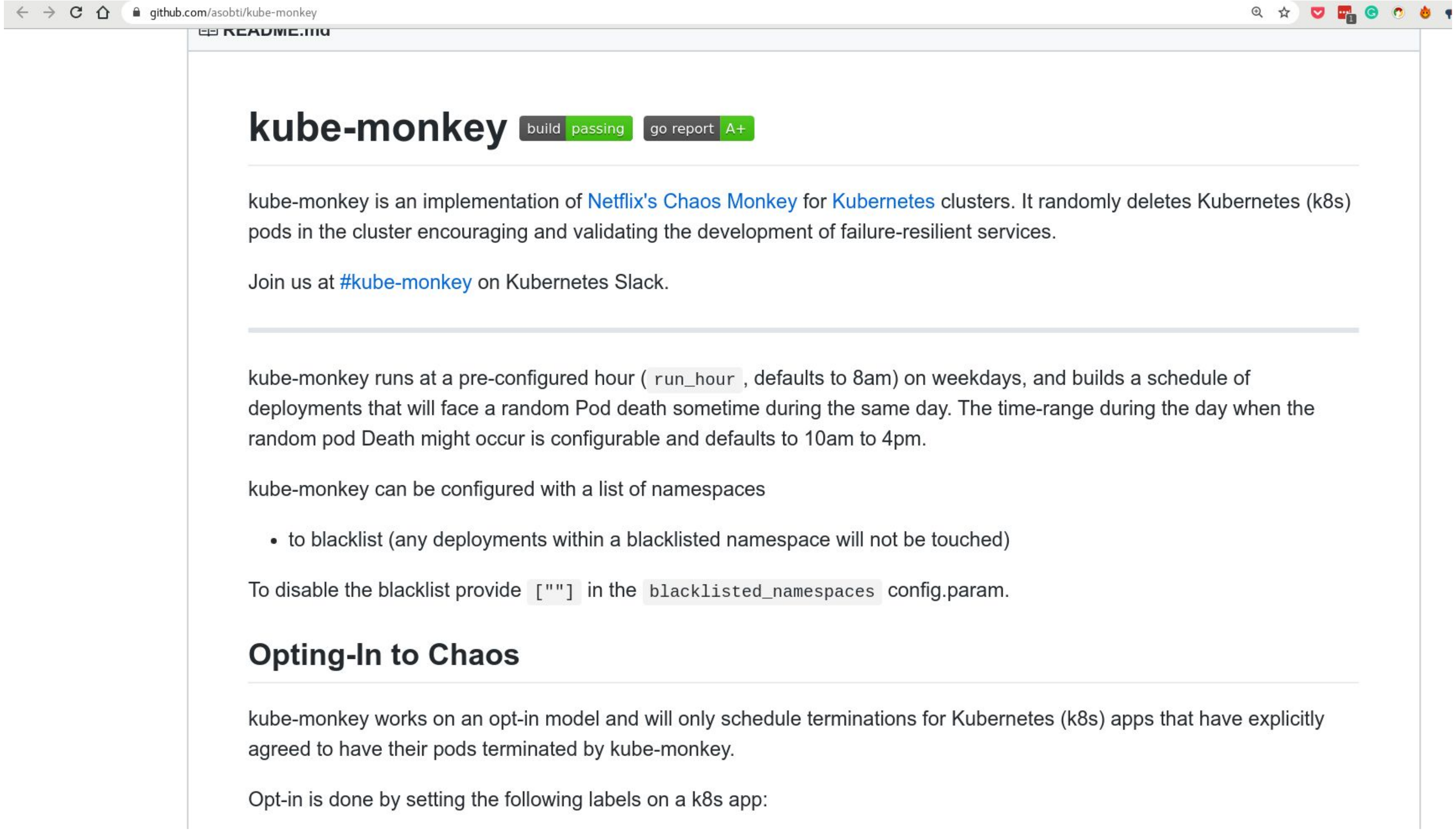
Gremlin = Paid Solution = Nice Chaos Reports

GREMLIN INC.

Breaking Things on Purpose

Chaos

Chaos



kube-monkey

build passing go report A+

kube-monkey is an implementation of Netflix's Chaos Monkey for Kubernetes clusters. It randomly deletes Kubernetes (k8s) pods in the cluster encouraging and validating the development of failure-resilient services.

Join us at #kube-monkey on Kubernetes Slack.

---

kube-monkey runs at a pre-configured hour (`run_hour` , defaults to 8am) on weekdays, and builds a schedule of deployments that will face a random Pod death sometime during the same day. The time-range during the day when the random pod Death might occur is configurable and defaults to 10am to 4pm.

kube-monkey can be configured with a list of namespaces

- to blacklist (any deployments within a blacklisted namespace will not be touched)

To disable the blacklist provide `[""]` in the `blacklisted_namespaces` config.param.

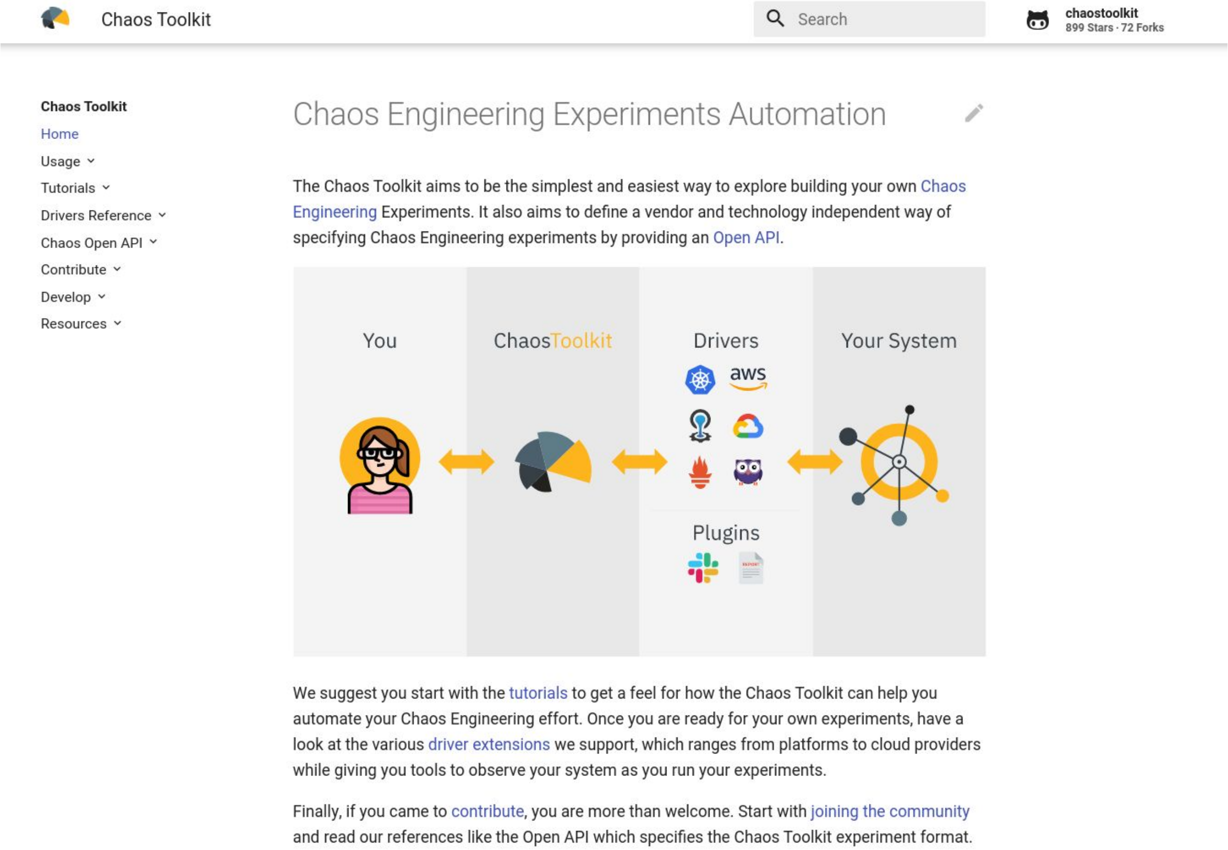
Opting-In to Chaos

kube-monkey works on an opt-in model and will only schedule terminations for Kubernetes (k8s) apps that have explicitly agreed to have their pods terminated by kube-monkey.

Opt-in is done by setting the following labels on a k8s app:

https://github.com/asobti/kube-monkey

Chaos



Chaos Toolkit

Search

chaostoolkit
899 Stars · 72 Forks

Chaos Toolkit

- Home
- Usage
- Tutorials
- Drivers Reference
- Chaos Open API
- Contribute
- Develop
- Resources

Chaos Engineering Experiments Automation

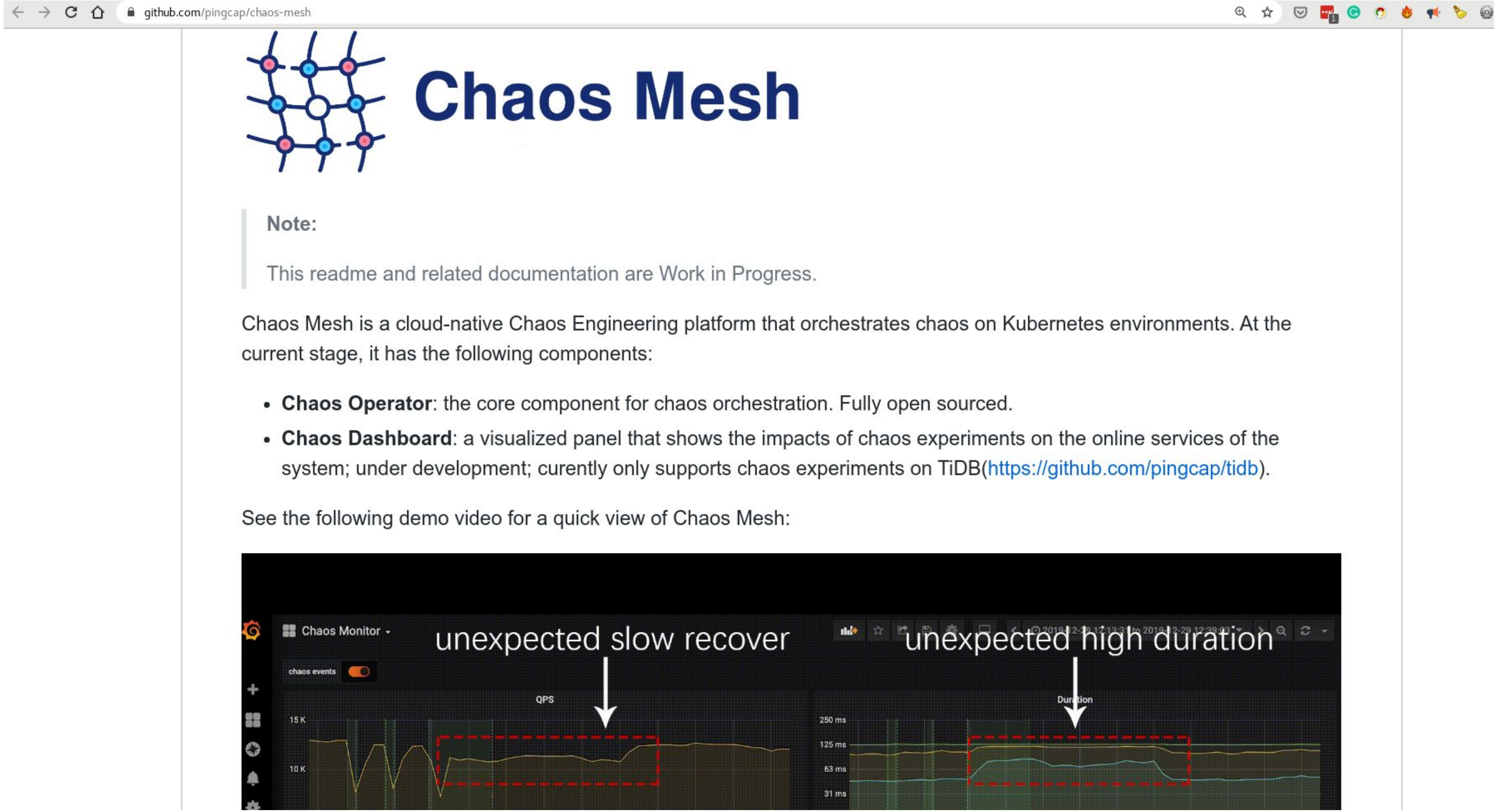
The Chaos Toolkit aims to be the simplest and easiest way to explore building your own Chaos Engineering Experiments. It also aims to define a vendor and technology independent way of specifying Chaos Engineering experiments by providing an Open API.

We suggest you start with the tutorials to get a feel for how the Chaos Toolkit can help you automate your Chaos Engineering effort. Once you are ready for your own experiments, have a look at the various driver extensions we support, which ranges from platforms to cloud providers while giving you tools to observe your system as you run your experiments.

Finally, if you came to contribute, you are more than welcome. Start with joining the community and read our references like the Open API which specifies the Chaos Toolkit experiment format.

https://docs.chaostoolkit.org/

Chaos



Chaos Mesh

> **Note:**
>
> This readme and related documentation are Work in Progress.

Chaos Mesh is a cloud-native Chaos Engineering platform that orchestrates chaos on Kubernetes environments. At the current stage, it has the following components:

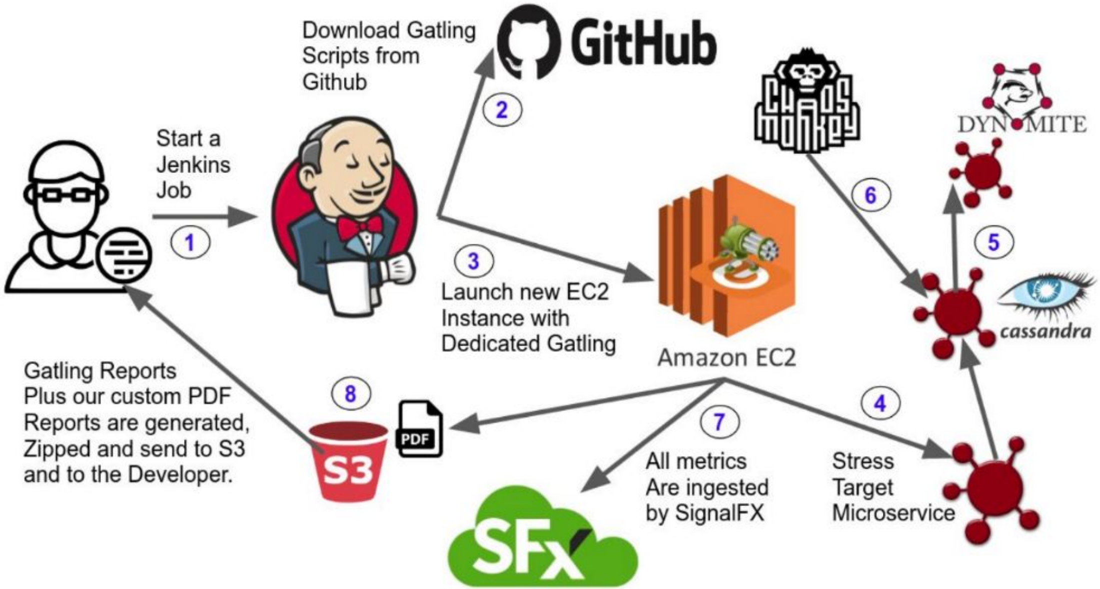
- **Chaos Operator**: the core component for chaos orchestration. Fully open sourced.
- **Chaos Dashboard**: a visualized panel that shows the impacts of chaos experiments on the online services of the system; under development; curently only supports chaos experiments on TiDB(https://github.com/pingcap/tidb).

See the following demo video for a quick view of Chaos Mesh:

https://github.com/pingcap/chaos-mesh

Chaos

Personal Usa Case - Stress/Chaos Platform



Download Gatling Scripts from Github
GitHub
Start a Jenkins Job
1
2
3
Launch new EC2 Instance with Dedicated Gatling
Amazon EC2
6
5
DYNOMITE
cassandra
4
Stress Target Microservice
7
All metrics Are ingested by SignalFX
8
Gatling Reports Plus our custom PDF Reports are generated, Zipped and send to S3 and to the Developer.
S3
PDF
SFx

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

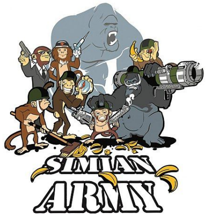
Exercises

Constraints

1. Stress Tests need to be written in Scala
2. You need to use Gatling

1. Using previous exercises or time-timecroservice image make the application run in kubernetes.
2. Create a stress test with Gatling
3. Create chaos with kubernetes killing PODS and make sure app still works and gatling tests don't fail.

Stress Test & Chaos Engineering



Diego Pacheco