Are we there yet?

Lessons Learnt Performance Testing A Crypto Exchange API

Ben Shi

#dddsydney

@hbish | /in/benshi | hbish.com


ARE WE THERE YET?
ARE WE THERE YET?
ARE WE THERE YET?
ARE WE THERE YET?
ARE WE THERE YET?

👋 hi

Microsoft

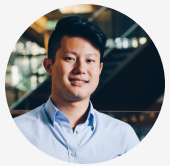
Telstra Purple

NDC { Sydney }

YOW!

M U S E S

Backed by tm Thinkmill



About Me

distra

ACI UNIVERSAL PAYMENTS

📐⏱️

Quality Assurance Engineer

~3.5 years

J.P.Morgan

🛠️🏗️

Software Engineer / Consultant

~7 years

Agenda

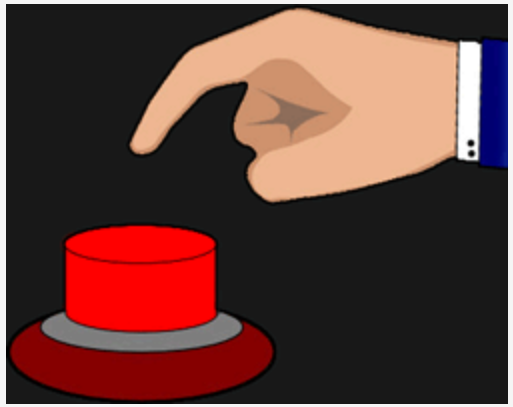
- Boring Bits
- Lessons Learnt
- Performance Engineering
- Wrap up

Quick Survey 

...a testing practice performed to determine how a system performs in terms of responsiveness and stability...

(source: wikipedia)

So like?



fast

faster

fastest


What can we measure?

What can we measure?

Client

Latency/Response Time - Round Trip
Throughput

Server

Latency/Response Time - Processing Time
Throughput
Availability
Server metrics - CPU, memory, disk I/O, network I/O...
Connection pooling
Cache hit/miss ratios
and more...

What flavours does it come in?

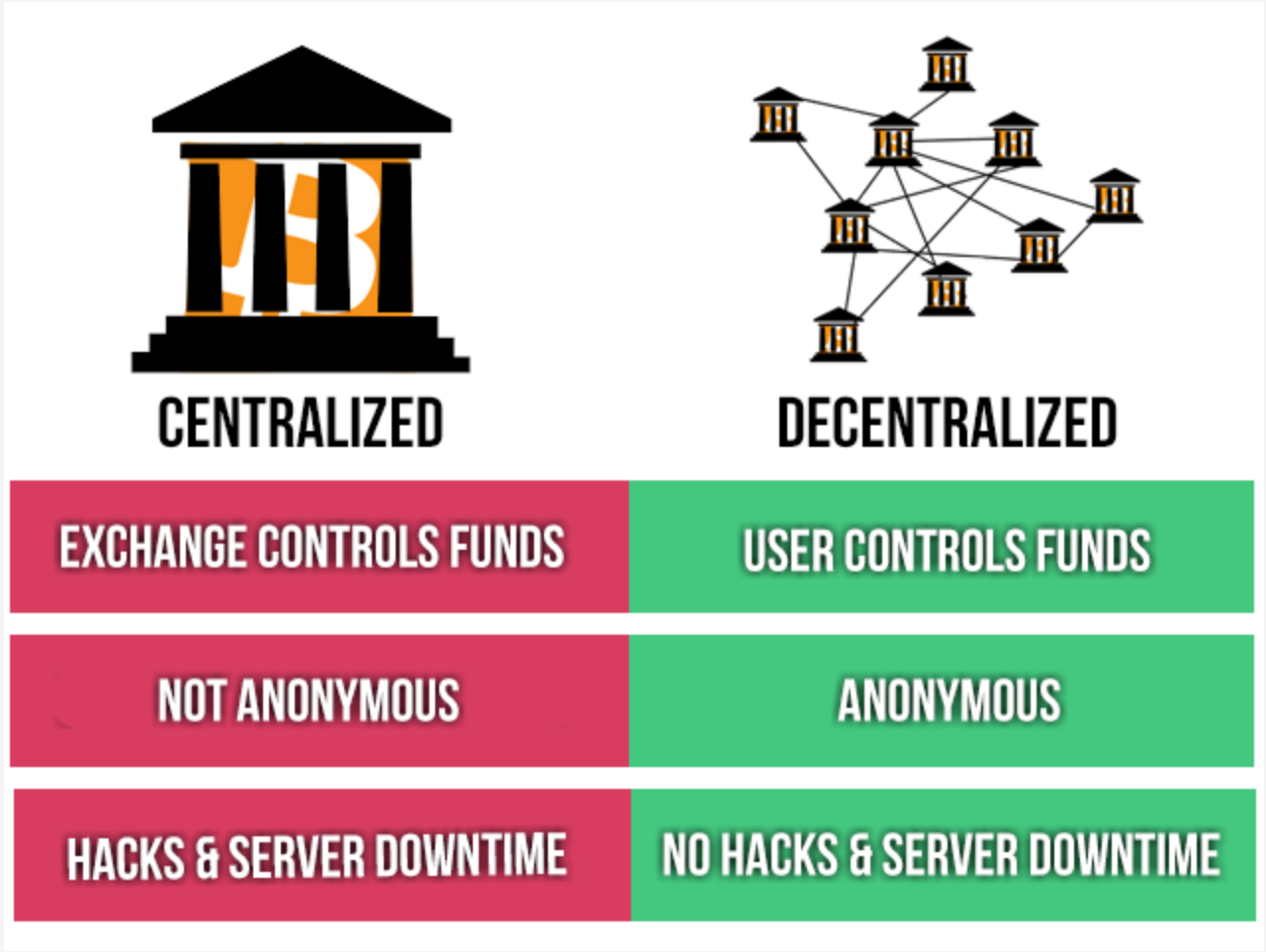
- Load testing
- Stress testing
- Soak testing
- Spike testing
- Breakpoint testing
- Configuration testing
- Isolation testing
- Internet testing

---

additional reading

The Art of Application Performance Testing - Ian Molyneaux

Crypto Exchange: A Crash Course



source: bitcoinwiki

API

User

User Info

Accounts

Deposits

Withdrawals

Trade

Orders

Place a new order

Cancel an order

Cancel all orders

List Orders

Get V1 Historical Orders List

Recent Orders

Get an order

Fills

Market Data

Symbols & Ticker

Order Book

Histories

Currencies

Others

Time

Websocket Feed

Public Channels

Symbol Ticker

All Symbols Ticker

Symbol Snapshot

Market Snapshot

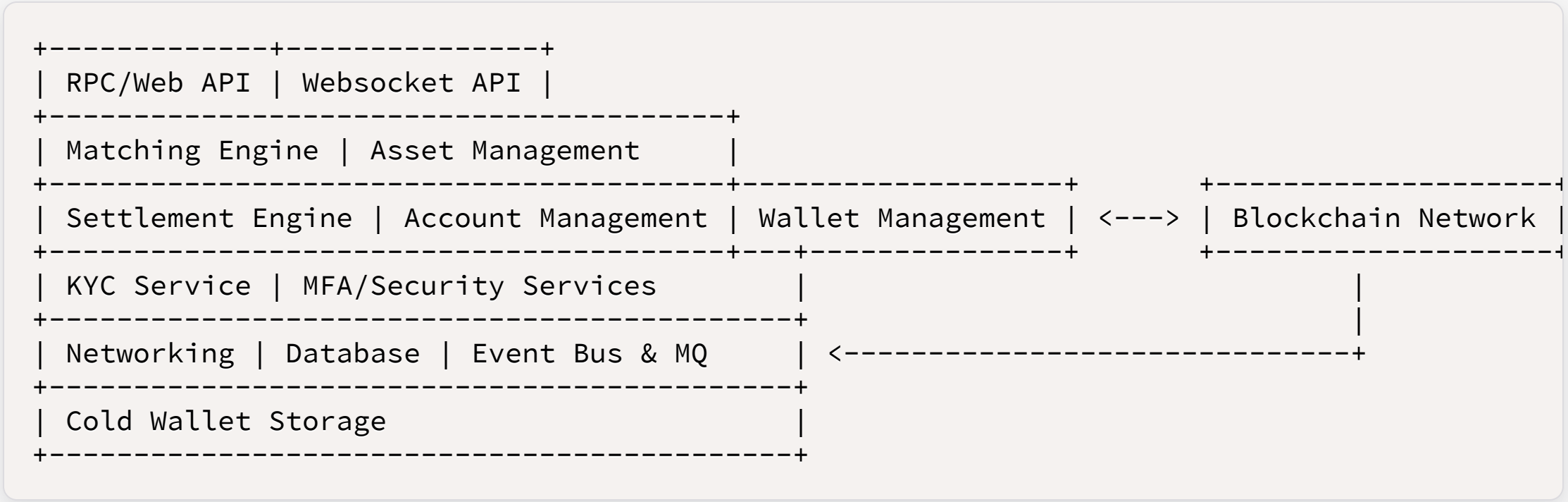
Level-2 Market Data

Match Execution Data

Full MatchEngine Data(Level 3)

Private Channels

Typical Architecture



additional reading: How do cryptocurrency exchanges work? and what technologies are driving disruption - Naveen Saraswat

Let the show begin!

- Centralised crypto exchange API
- Order submission/execution
- Gatling

Gatling

- non-blocking/asynchronous stack (scala, akka, netty)
- not just a load runner, can be scripted using DSL
- built-in assertion
- good calculation and statistics
- nice reporting
- comes with a recorder
- open-source*
- can run in a distributed fashion and fed into other performance platforms

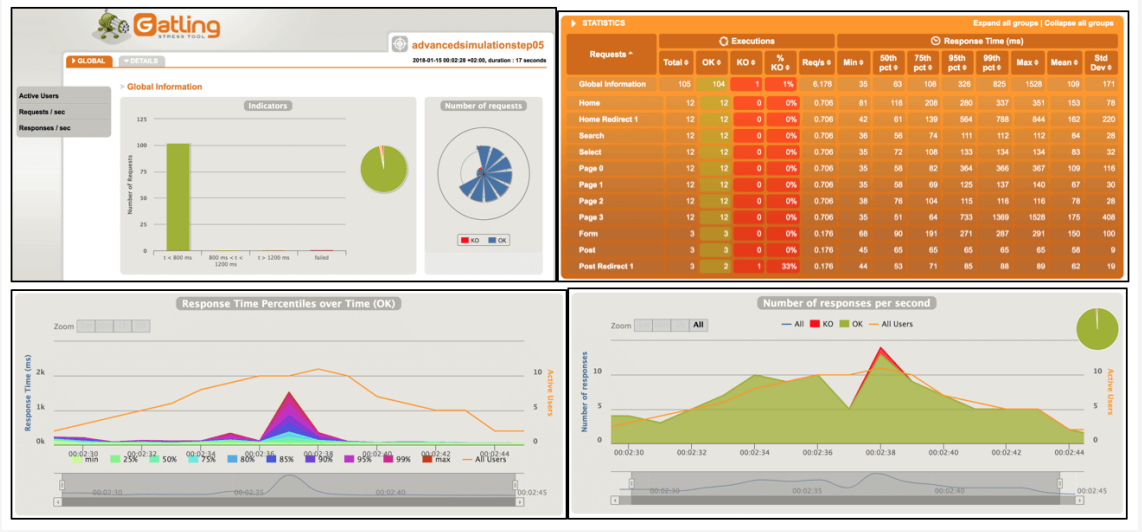
```
package computerdatabase // 1

import io.gatling.core.Predef._ // 2
import io.gatling.http.Predef._
import scala.concurrent.duration._

class BasicSimulation extends Simulation { // 3

  val httpProtocol = http // 4
      .baseUrl("http://computer-database.gatling.io") // 5
      .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8") // 6
      .doNotTrackHeader("1")
      .acceptLanguageHeader("en-US,en;q=0.5")
      .acceptEncodingHeader("gzip, deflate")
      .userAgentHeader("Mozilla/5.0 (Windows NT 5.1; rv:31.0) Gecko/20100101 Firefox/31.0")

  val scn = scenario("BasicSimulation") // 7
      .exec(http("request_1") // 8
      .get("/")) // 9
```



Test with a goal in mind

Non-Functional Requirements

Expensive & Time Consuming

Baseline!

Without data you're just another person with an opinion

- W. Edwards Deming

Consider all the parameters

```
POST https://{{host}}/api/order
jwt: abcdefg1234567890
Content-Type: application/json
Accept: application/json
Accept-Charset: utf-8

{
    "symbol": "EthBtc",
    "side": "Buy",
    "order_type": "Limit",
    "time_in_force": "Gtc",
    "quantity": 2,
    "price": 10.3123,
    "new_order_resp_type": "Ack",
    "timestamp": 1562336244
}
```

Parameters

```
val price = ??
val quantity = ??

val Side = Array("Buy", "Sell")
val OrderTypes = Array("Market", "Limit", "StopLoss", "StopLossLimit", "TakeProfit", "TakeP
val TimeInForce = Array("Gtc", "Ioc")
val Symbols = Array("EthBtc", "EthLtc", "EthUsdt", "EthXrp")
```

```
exec(http("""POST /api/order STOPLOSS SELL ${symbol}""")
  .post("/order")
  .body(StringBody(
    """{
                "symbol": "${symbol}",
                "side": "${side}",
                "order_type": "StopLoss",
                "time_in_force": "Gtc",
                "quantity": ${quantity},
                "stop_price": ${stopSellPrice},
                "new_order_resp_type": "Ack",
                "timestamp": ${timestamp}
                }""")).asJson
  .headers(PostHeaders)
  .header("jwt", """${jwt}""")
  .check(status.is(200))
)
```

```
val orderParams: Iterator[Map[String, Any]] = Iterator.continually(
  Map(
    "quantity" -> Random.nextDouble() * 100,
    "price" -> Random.nextDouble() * 10,
    "symbol" -> Symbols(Random.nextInt(Symbols.length)),
    "timestamp" -> Instant.now.getEpochSecond,
  )
)
```

😀

```
val orderParams: Iterator[Map[String, Any]] = Iterator.continually(
    elem = {
        val symbol = Random.nextInt(Symbols.length)
        val marketPrice = getMarketPrice(symbol)
        Map(
            "quantity" -> Random.nextDouble() * 100,
            "price" -> marketPrice,
            "sellPrice" -> marketPrice * (1 + (Random.nextInt(5) / 1000)),
            "buyPrice" -> marketPrice * (1 - (Random.nextInt(5) / 1000)),
            "symbol" -> Symbols(symbol),
            "timestamp" -> Instant.now.getEpochSecond,
        )
    }
)
```

Functional Testing on Steroids

Add some asserts!

- response code
- check orders matched and reconcile with user balance

Trading Bots

additional functional tests while under load

Know the limits

- Rate limiting
- Quota
- DDOS protection

Fast & Slow Users

Trust, but Verify (Developers)

Developer's performance intuitions are often wrong

Myself included

Things Devs might say

- "Get a bigger instance with more ram/cpu"
- "Just add more instances"
- "It must be the database"
- "It's slow but it hasn't crashed yet"
- "Must be an environment issue"
- "Your test is wrong"

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	7167293	7167293	0	0%	3643.769	1	12	28	71	134	7273	24	65
POST /ap...Y EthXrp	452846	452846	0	0%	230.222	1	9	25	67	133	7261	23	67
POST /ap...L EthXrp	443799	443799	0	0%	225.622	1	13	31	73	134	7227	26	66
POST /ap...L EthBtc	450083	450083	0	0%	228.817	1	9	24	67	132	3062	22	64
POST /ap...Y EthBtc	445321	445321	0	0%	226.396	1	14	32	74	136	3083	26	65
POST /ap...Y EthLtc	445158	445158	0	0%	226.313	1	14	31	73	134	3058	26	65
POST /ap...L EthLtc	451691	451691	0	0%	229.634	1	9	24	67	132	3081	22	64
POST /ap...L EthLtc	444752	444752	0	0%	226.107	1	13	31	73	135	3061	26	66
POST /ap... EthUsdt	445641	445641	0	0%	226.559	1	14	32	75	136	7190	27	65
POST /ap...Y EthXrp	445064	445064	0	0%	226.265	1	14	31	73	133	7273	26	64
POST /ap...L EthBtc	443901	443901	0	0%	225.674	1	13	31	74	135	3022	26	64
POST /ap...Y EthLtc	451208	451208	0	0%	229.389	1	9	25	67	132	2225	22	64
POST /ap... EthUsdt	444272	444272	0	0%	225.863	1	14	32	75	136	3393	27	65
POST /ap...Y EthBtc	450960	450960	0	0%	229.263	1	9	25	67	133	7123	23	67
POST /ap... EthUsdt	450506	450506	0	0%	229.032	1	9	24	67	132	3085	22	65
POST /ap...L EthXrp	450951	450951	0	0%	229.258	1	9	24	67	132	3082	22	64
POST /ap... EthUsdt	451140	451140	0	0%	229.354	1	9	24	67	133	3107	23	65

```
1% tile: 974.0 (ns)
5% tile: 1075.0 (ns)
10% tile: 2292.0 (ns)
25% tile: 2695.0 (ns)
50% tile: 3671.0 (ns)
75% tile: 10440.0 (ns)
90% tile: 10091923.799999999 (ns)
99% tile: 68835579.60000025 (ns)
```

Things Devs might do

```
let start = Instant::now();
submit_order();
let duration = start.elapsed();

println!("Time elapsed in submit_order() is: {:?}", duration);
```

Coordinated Omission Problem

additional reading: "How NOT to Measure Latency" by Gil Tene

Application Performance Monitoring Metrics

+

Log Aggregation w/ Correlation IDs

=

Observability

⏱ Collect Data

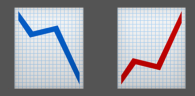
🍾 Find the bottleneck

🛠 Fix it

♻ Repeat

Without data you're just another person with an opinion.

- W. Edwards Deming



Statistics Lie

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	7167293	7167293	0	0%	3643.769	1	12	28	71	134	7273	24	65
POST /ap...Y EthXrp	452846	452846	0	0%	230.222	1	9	25	67	133	7261	23	67
POST /ap...L EthXrp	443799	443799	0	0%	225.622	1	13	31	73	134	7227	26	66
POST /ap...L EthBtc	450083	450083	0	0%	228.817	1	9	24	67	132	3062	22	64
POST /ap...Y EthBtc	445321	445321	0	0%	226.396	1	14	32	74	136	3083	26	65
POST /ap...Y EthLtc	445158	445158	0	0%	226.313	1	14	31	73	134	3058	26	65
POST /ap...L EthLtc	451691	451691	0	0%	229.634	1	9	24	67	132	3081	22	64
POST /ap...L EthLtc	444752	444752	0	0%	226.107	1	13	31	73	135	3061	26	66
POST /ap... EthUsdt	445641	445641	0	0%	226.559	1	14	32	75	136	7190	27	65
POST /ap...Y EthXrp	445064	445064	0	0%	226.265	1	14	31	73	133	7273	26	64
POST /ap...L EthBtc	443901	443901	0	0%	225.674	1	13	31	74	135	3022	26	64
POST /ap...Y EthLtc	451208	451208	0	0%	229.389	1	9	25	67	132	2225	22	64
POST /ap... EthUsdt	444272	444272	0	0%	225.863	1	14	32	75	136	3393	27	65
POST /ap...Y EthBtc	450960	450960	0	0%	229.263	1	9	25	67	133	7123	23	67
POST /ap... EthUsdt	450506	450506	0	0%	229.032	1	9	24	67	132	3085	22	65
POST /ap...L EthXrp	450951	450951	0	0%	229.258	1	9	24	67	132	3082	22	64
POST /ap... EthUsdt	451140	451140	0	0%	229.354	1	9	24	67	133	3107	23	65

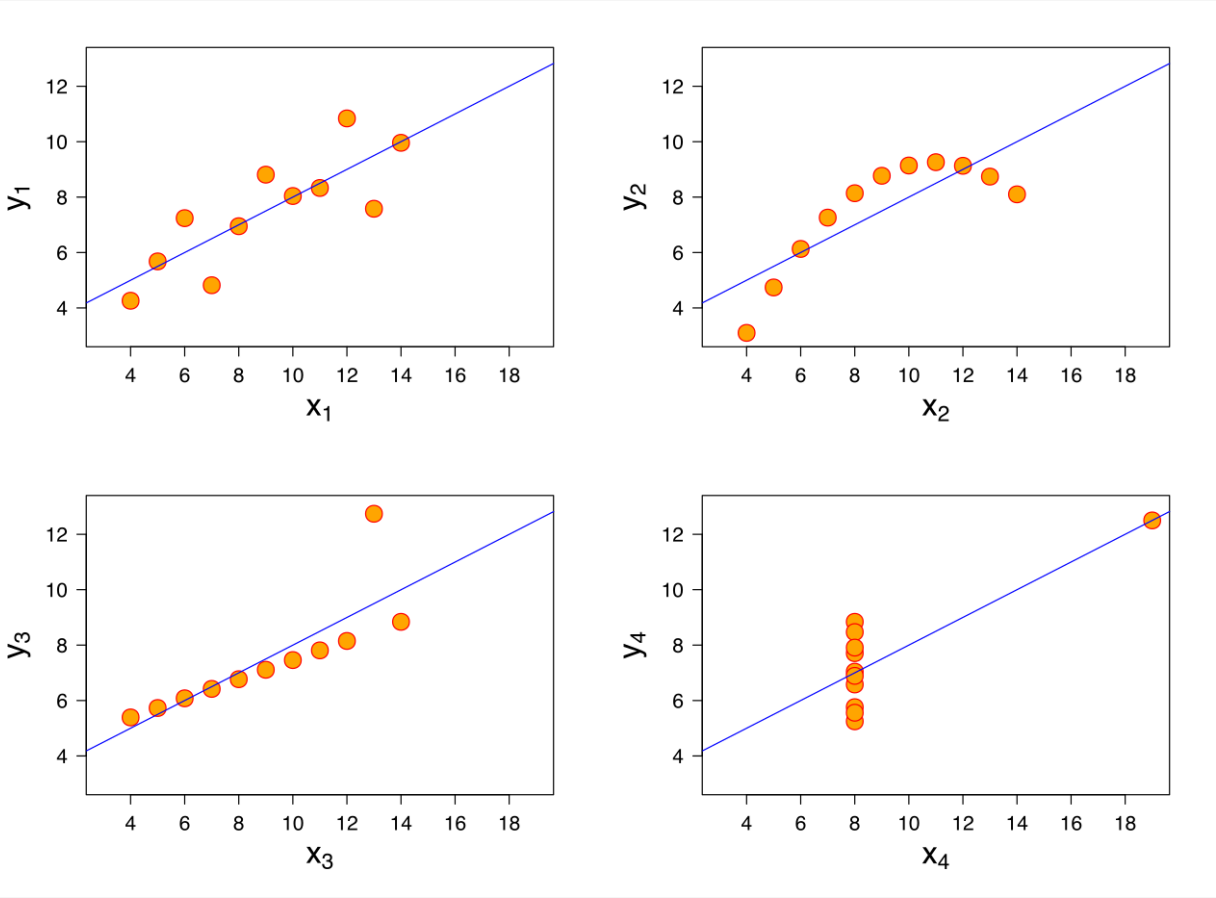
Ignore

❌ Mean

❌ Median

❌ Standard Deviation

Anscombe's Quartlet



	I		II		III		IV	
	x	y	x	y	x	y	x	y
	10	8,04	10	9,14	10	7,46	8	6,58
	8	6,95	8	8,14	8	6,77	8	5,76
	13	7,58	13	8,74	13	12,74	8	7,71
	9	8,81	9	8,77	9	7,11	8	8,84
	11	8,33	11	9,26	11	7,81	8	8,47
	14	9,96	14	8,1	14	8,84	8	7,04
	6	7,24	6	6,13	6	6,08	8	5,25
	4	4,26	4	3,1	4	5,39	19	12,5
	12	10,84	12	9,13	12	8,15	8	5,56
	7	4,82	7	7,26	7	6,42	8	7,91
	5	5,68	5	4,74	5	5,73	8	6,89
SUM	99,00	82,51	99,00	82,51	99,00	82,50	99,00	82,51
AVG	9,00	7,50	9,00	7,50	9,00	7,50	9,00	7,50
STDEV	3,32	2,03	3,32	2,03	3,32	2,03	3,32	2,03

Look at

✅ Max

✅ Percentiles

STATISTICS

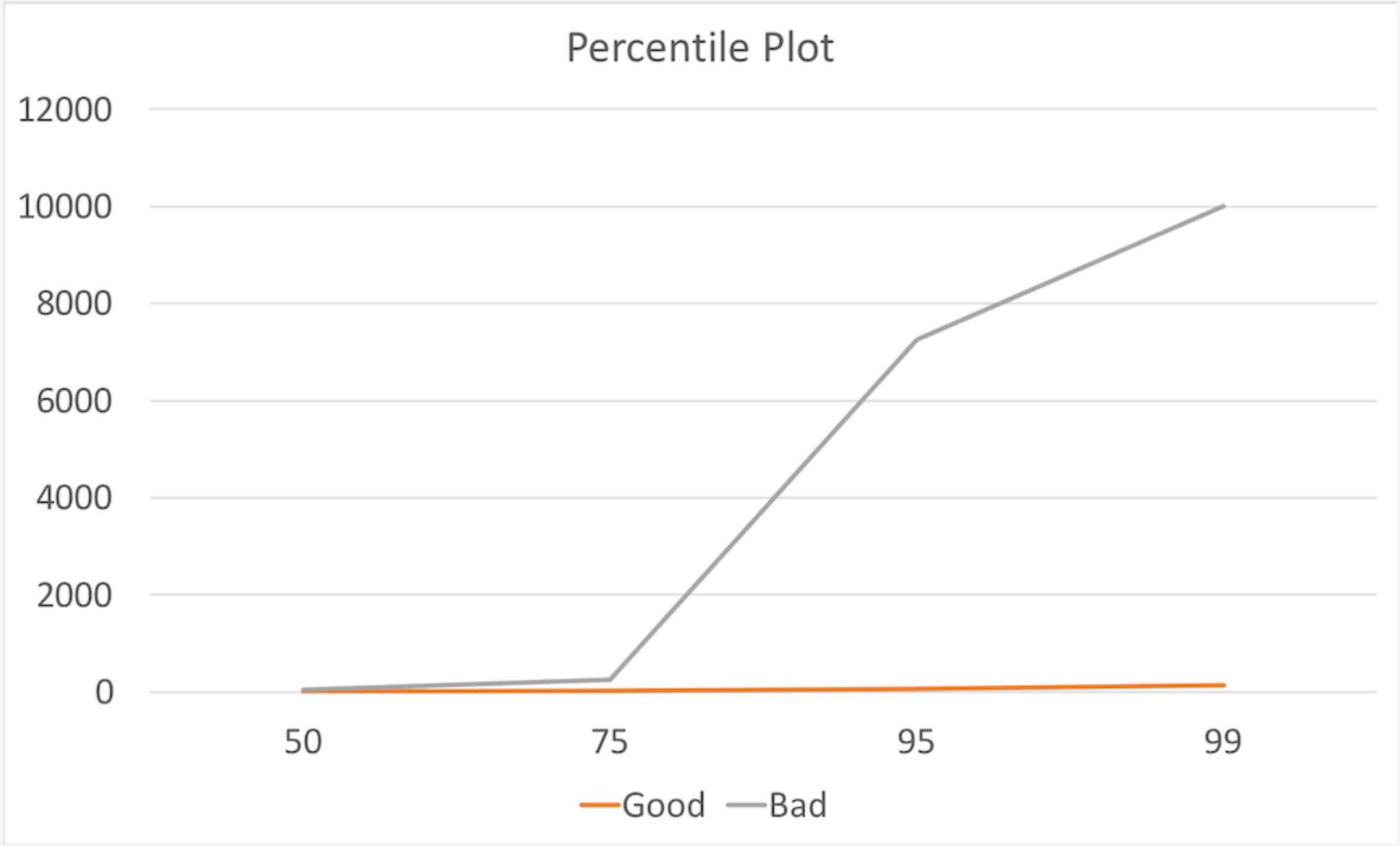
Expand all groups | Collapse all groups

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	7167293	7167293	0	0%	3643.769	1	12	28	71	134	7273	24	65
POST /ap...Y EthXrp	452846	452846	0	0%	230.222	1	9	25	67	133	7261	23	67
POST /ap...L EthXrp	443799	443799	0	0%	225.622	1	13	31	73	134	7227	26	66
POST /ap...L EthBtc	450083	450083	0	0%	228.817	1	9	24	67	132	3062	22	64
POST /ap...Y EthBtc	445321	445321	0	0%	226.396	1	14	32	74	136	3083	26	65
POST /ap...Y EthLtc	445158	445158	0	0%	226.313	1	14	31	73	134	3058	26	65
POST /ap...L EthLtc	451691	451691	0	0%	229.634	1	9	24	67	132	3081	22	64
POST /ap...L EthLtc	444752	444752	0	0%	226.107	1	13	31	73	135	3061	26	66
POST /ap... EthUsdt	445641	445641	0	0%	226.559	1	14	32	75	136	7190	27	65
POST /ap...Y EthXrp	445064	445064	0	0%	226.265	1	14	31	73	133	7273	26	64
POST /ap...L EthBtc	443901	443901	0	0%	225.674	1	13	31	74	135	3022	26	64
POST /ap...Y EthLtc	451208	451208	0	0%	229.389	1	9	25	67	132	2225	22	64
POST /ap... EthUsdt	444272	444272	0	0%	225.863	1	14	32	75	136	3393	27	65
POST /ap...Y EthBtc	450960	450960	0	0%	229.263	1	9	25	67	133	7123	23	67
POST /ap... EthUsdt	450506	450506	0	0%	229.032	1	9	24	67	132	3085	22	65
POST /ap...L EthXrp	450951	450951	0	0%	229.258	1	9	24	67	132	3082	22	64
POST /ap... EthUsdt	451140	451140	0	0%	229.354	1	9	24	67	133	3107	23	65

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	2706735	2615596	91139	3%	1372.584	1	45	264	7262	10003	60076	1089	4294
POST /ap... EthUsdt	338833	327372	11461	3%	171.822	1	47	268	7280	10003	60035	1099	4331
POST /ap...Y EthBtc	338607	327161	11446	3%	171.707	1	44	262	7255	10003	60076	1086	4292
POST /ap...L EthXrp	338458	326961	11497	3%	171.632	1	46	269	7285	10003	60021	1100	4338
POST /ap...Y EthXrp	338447	327118	11329	3%	171.626	1	46	265	7256	10003	60037	1093	4313
POST /ap...L EthBtc	338319	326995	11324	3%	171.561	1	44	264	7253	10003	60036	1084	4271
POST /ap... EthUsdt	338157	326779	11378	3%	171.479	1	47	264	7242	10003	60028	1084	4289
POST /ap...L EthLtc	337958	326695	11263	3%	171.378	1	44	263	7241	10003	60025	1075	4222
POST /ap...Y EthLtc	337956	326515	11441	3%	171.377	1	44	262	7285	10003	60023	1092	4293


Percentile Plot
12000
10000
8000
6000
4000
2000
0
50
75
95
99
Good
Bad

Scaling your performance test

Focus on the actual test

Scaling your test comes later

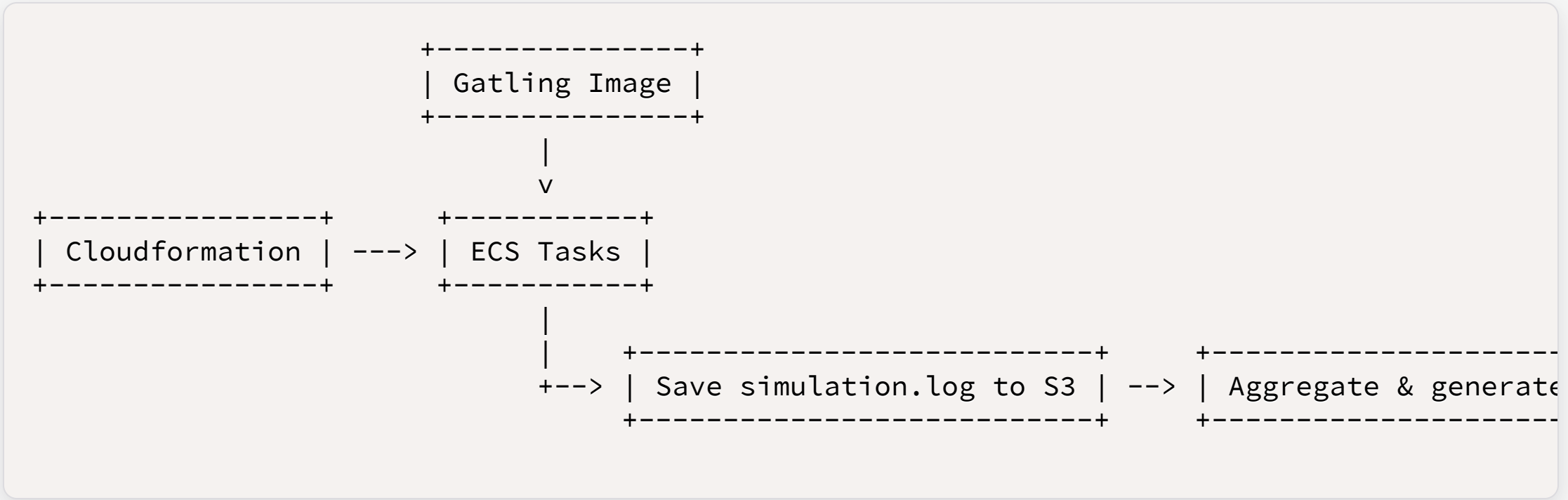
Paid Performance Testing Platform

Gatling Frontline

BlazeMeter

Flood.io

DIY - (The Hard way)



Performance Engineering

Quickfire Edition

What about Microservices/Serverless?

- Watch for timeouts
- Perform spike tests to determine time taken to scale

What about GraphQL?

- It's possible, but more complicated
- Different combination of fields and fragments
- Make sure you have sensible request tracing

Many birds, one stone

Reusable Tests. Framework selection is critical!

E2E test in Dev/Test Deployment

```
./gradlew gatlingRun -DbaseUrl="http://dev.env:80/api" -DnumberOfUser=1 -DrunDurationSecs=3
```

Perf Test

```
./gradlew gatlingRun -DbaseUrl="http://perf.env:80/api" -DnumberOfUser=2000 -DrunDurationSe
```

Smoke Test in Production

```
./gradlew gatlingRun -DbaseUrl="http://prod.env:80/api" -DnumberOfUser=1 -DrunDurationSecs=
```

CI/CD

- Build a small subset of your performance testing suite as part of your pipeline
- Monitor the build time and capture performance metrics
- Fail or Add alert for any executions that over n %
- Run perf early in the SDLC and as often as possible

It doesn't take a lot to cause an outage

```
A regular expression that backtracked enormously and exhausted CPU used for HTTP/HTTPS serv
```


```
Introduce performance profiling for all rules to the test suite. (ETA:  July 19)
```

Cloudflare Outage July 2019 - https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/

Test in Production?

Test in Production?

Yes, only if you can cleanup the data

Otherwise, test in a separate environment with configuration similar to prod

Takeaways

- Performance Test != Non-Functional Testing
- Know your parameters and limits upfront
- Test with the end in mind
- Forgo any assumptions and verify using performance metrics
- Stats Lie!

Awesome Performance

https://github.com/hbish/awesome-performance

Thanks

@hbish | /in/benshi | hbish.com