Real-world HTTP performance benchmarking, lessons learned

Julien Viet

QCon Shangai


CNUTCon 全球运维技术大会 [上海]
主办方 Geekbang. 极客邦科技 InfoQ
50+ 年末充电
开发&运维技术干货大盘点
容器
Kubernetes
DevOps
全链路压测
Severless
自动化运维
Service Mesh
Elasticsearch
微服务
使用折扣码「QCon」优惠报名 咨询电话：13269078023
CNUT
扫码锁定席位

Once upon a time


1 vertx 656,119 100.0%
2 netty 632,596 96.4%
3 undertow 614,547 93.7%
4 cpoll_cppsp 522,423 79.6%
5 jetty-servlet 448,947 68.4%
6 grizzly 425,172 64.8%
7 gemini 424,586 64.7%
8 servlet 417,619 63.6%
9 go 367,274 56.0%
10 openresty 330,829 50.4%
11 spark 226,365 34.5%
12 revel 221,603 33.8%
13 falcore 172,827 26.3%
14 compojure 127,671 19.5%

Round #8

Every one was happy

But one day...


1 libreactor 3,987,169 100.0% (97.6%)
2 ulib 3,969,896 99.6% (97.2%)
3 rapidoid-http-fast 3,794,920 95.2% (92.9%)
4 tokio-minihttp 3,682,355 92.4% (90.1%)
5 rapidoid 3,599,389 90.3% (88.1%)
6 colossus 3,133,734 78.6% (76.7%)
7 fasthttp 2,981,149 74.8% (73.0%)
8 undertow 2,928,350 73.4% (71.7%)
9 netty 2,527,020 63.4% (61.9%)
10 cpoll_cppsp 2,204,071 55.3% (54.0%)
11 bayou 1,870,539 46.9% (45.8%)
12 h2o 1,614,902 40.5% (39.5%)
13 blaze 1,245,458 31.2% (30.5%)
14 go-prefork 1,117,465 28.0% (27.4%)
15 vertx 1,080,340 27.1% (26.4%)

Round #14

37	sinatra-sequel	42,387	23.2%
38	falcore	39,826	21.8%
39	php-raw7	39,041	21.3%
40	wt	38,941	21.3%
41	php-php5-raw	38,834	21.2%
42	vertx-web-mongodb	38,147	20.8%
43	vertx-web-postgres	37,709	20.6%
44	go-postgres	36,324	19.9%
45	dropwizard-postgres	36,293	19.8%
46	echo	36,244	19.8%
47	redstone	35,271	19.3%
48	yesod	35,137	19.2%

Round #14

157	koa	0	0.0%
158	ktor	0	0.0%
159	nodejs-mongodb	0	0.0%
160	nodejs-mongodb-raw	0	0.0%
161	sailsjs	0	0.0%
162	play2-scala-reactive	0	0.0%
163	roda-sequel-torquebo	0	0.0%
164	vertx-web-mongodb	0	0.0%
165	vertx-web-postgres	0	0.0%
166	yesod	0	0.0%
167	hexagon	–	Did not complete
168	jawn	–	Did not complete

Round #14

157	koa	0	0.0%
158	ktor	0	0.0%
159	nodejs-mongodb	0	0.0%
160	nodejs-mongodb-raw	0	0.0%
161	sailsjs	[illegible]	[illegible]
162	play2-sca[illegible]	[illegible]	[illegible]
16[illegible]	[illegible]	0	0.0%
16[illegible]	[illegible]ongodb	0	0.0%
165	[illegible]ertx-web-postgres	0	0.0%
166	yesod	0	0.0%
167	hexagon	–	Did not complete
168	jawn	–	Did not complete

Hi,

I recently reviewed the latest techempower results (#14) and Vertx seems to be sliding downward in the list.

Round #14

Hi,

I recently reviewed the latest techempower results (#14) and Vertx ... sliding downward in the list.

But in the benchmark results, I see that the "Data Table" stats are not really good for Vert.x. Also, nodejs stats are really good in database calls.

Now I have one question, is Vert.x really not good with databases or the benchmarks are wrong?

1…		0	0.0%
158	ktor	0	0.0%
159	nodejs-mongodb		
160	nodejs-mongodb-raw	0	0.0%
161	sailsjs		
162	play2-scal…		
16…		0	0.0%
16…	…ongodb	0	0.0%
165	vertx-web-postgres	0	0.0%
166	yesod	0	0.0%
167	hexagon	–	Did not complete
168	jawn	–	Did not complete

Round #14

Real-world HTTP performance benchmarking, lessons learned

Julien Viet

Open source developer for 16+ years

@vertx_project lead

Principal software engineer at redhat

Marseille JUG Leader

https://www.julienviet.com/

http://github.com/vietj

@julienviet

https://www.mixcloud.com/cooperdbi/

Eclipse Vert.x

Open source project started in 2012

Eclipse / Apache licensing

A toolkit for building reactive applications for the JVM

8K ★ on

Built on top of

https://vertx.io

@vertx_project

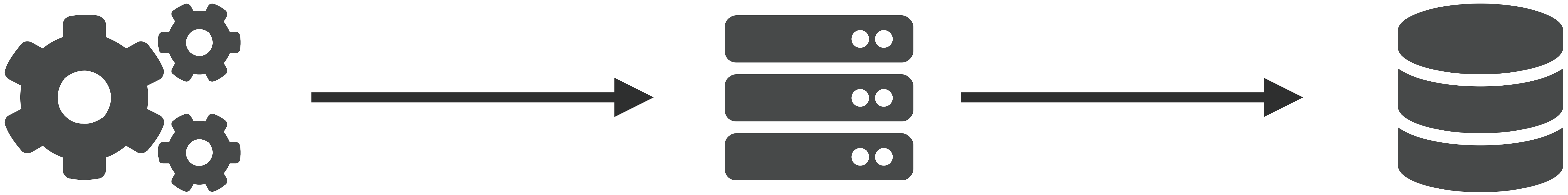
Techempower Framework Benchmark

- ✓ Performance of production grade deployments of real-world application frameworks and platforms
- ✓ 464 frameworks - 26 languages
- ✓ Community of contributors on GitHub
- ✓ Physical server or cloud (Azure)

6 benchmarks

- ✓ "/plaintext", "/json"

- ✓ "/db", "/queries", "/updates", "/fortunes"



Things to remember

- ✓ Benchmarking is hard
- ✓ Benchmarking != load testing
- ✓ Measure don't guess
- ✓ Be critic

The lab

Julien Viet <jviet@redhat.com>
to Rodney

Hi Rodney,

we inherited from the two boxes that Norman Maurer used for Netty testing (and that Tim also used), one of them is down currently so the single box we have is useless. Besides Jean Marc Larvol from helpdesk said that these machines are unreliable and too old.

We talked a couple of times in the past of having machines for doing performance testing that can be used by the team, pretty much like what Norman and Tim used. The characteristics of these boxes is that they are not virtual (for good reasons) and use private networking.

One use case is for Techempower benchmarking (Tim used them for this scenario), other use case are for clustering performance testing.

Ideally we would need 4 boxes.

Can we get budget for these machines, wether they are hosted by Red Hat or rented on a provider ?

Rodney Russ <rruss@redhat.com>

to me

I'll put in for budget for FY18.

Hi Rodney,

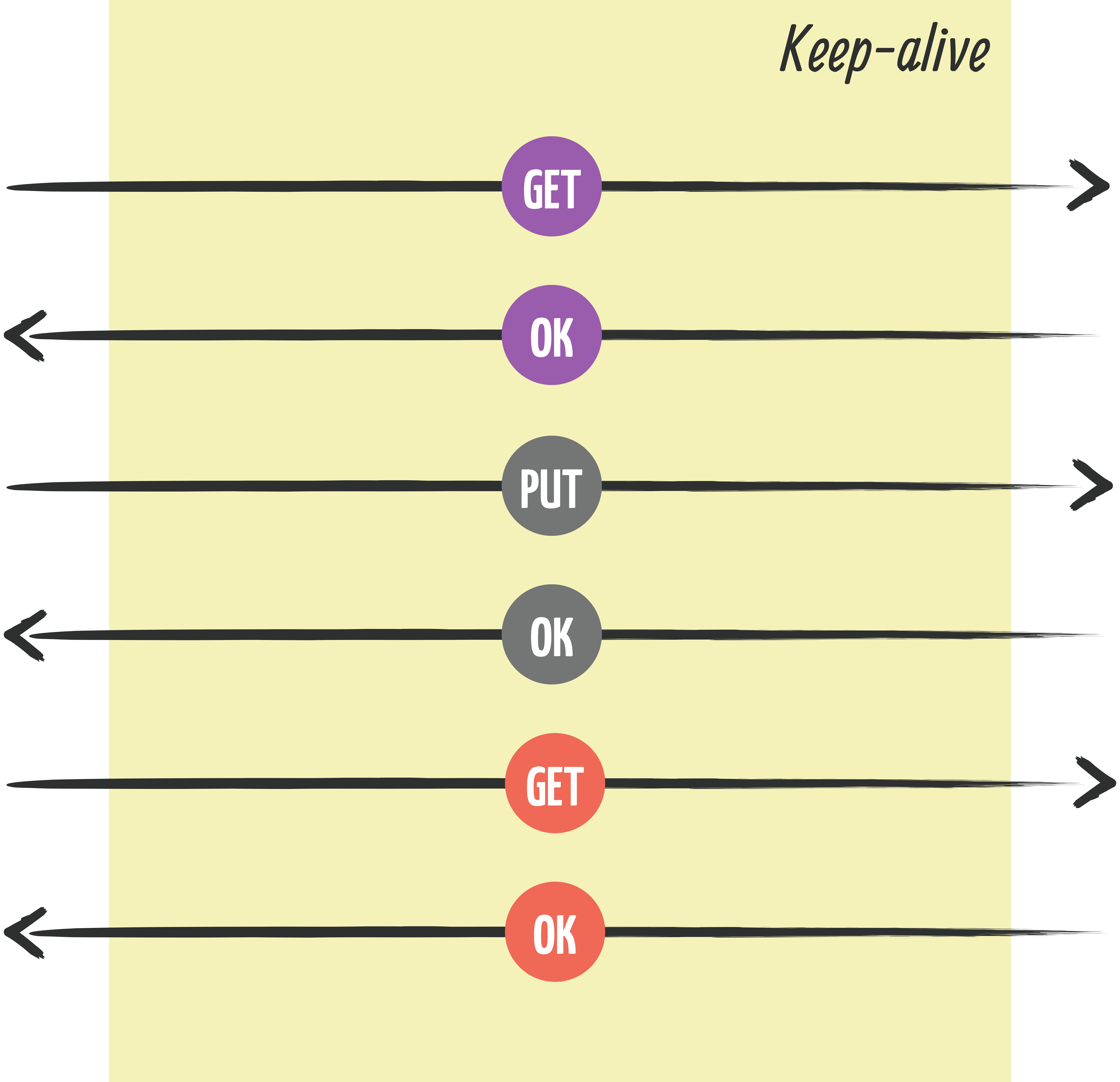
We received your request for 4 machines with 10gbe network capability starting 4/10/2017 and ending on 6/16/2017. We will have 4 hosts available with 10gbe during your requested window. A bit closer to the start date we will send you a document with all the ip address space and machine hostnames. In the meantime if you have any questions please reply to this email and one of us will get back with you. If your project specs change please let us know asap.

Thanks,
Jake

/plaintext

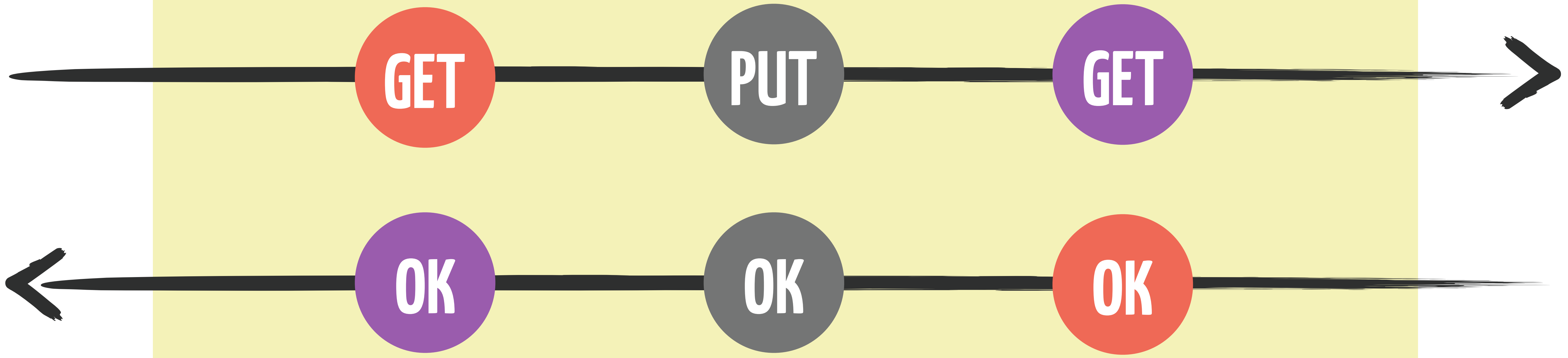
Benchmark

- ✓ Simple Hello World
- ✓ 16,384 concurrent connections
- ✓ HTTP pipelining (16)
- ✓ No back-end
- ✓ Heavily CPU bound


Keep-alive
GET
OK
PUT
OK
GET
OK

Head of line blocking

Pipelining



Our weapons

- ✓ Async-profiler + Flame graphs
- ✓ Jitwatch
- ✓ Wireshark

Code inlining

process error

process request

process body

```java
private void processMessage(Object msg) {
  if (msg instanceof HttpObject) {
    HttpObject obj = (HttpObject) msg;
    DecoderResult result = obj.decoderResult();
    if (result.isFailure()) {
      Throwable cause = result.cause();
      if (cause instanceof TooLongFrameException) {
        String causeMsg = cause.getMessage();
        HttpVersion version;
        if (msg instanceof HttpRequest) {
          version = ((HttpRequest) msg).protocolVersion();
        } else if (currentRequest != null) {
          version = currentRequest.version() == io.vertx.core.http.HttpVersion.HTTP_1_0 ? HttpVersion.HTTP_1_0 : HttpVersion.HTTP_1_1;
        } else {
          version = HttpVersion.HTTP_1_1;
        }
        HttpResponseStatus status = causeMsg.startsWith("An HTTP line is larger than") ? HttpResponseStatus.REQUEST_URI_TOO_LONG : HttpResponseStatus.BAD_REQUEST;
        DefaultFullHttpResponse resp = new DefaultFullHttpResponse(version, status);
        writeToChannel(resp);
      }
      channel.pipeline().fireExceptionCaught(result.cause());
      return;
    }
    if (msg instanceof HttpRequest) {
      HttpRequest request = (HttpRequest) msg;
      if (server.options().isHandle100ContinueAutomatically()) {
        if (HttpHeaders.is100ContinueExpected(request)) {
          write100Continue();
        }
      }
      HttpServerResponseImpl resp = new HttpServerResponseImpl(vertx,  conn: this, request);
      HttpServerRequestImpl req = new HttpServerRequestImpl( conn: this, request, resp);
      handleRequest(req, resp);
    }
    if (msg instanceof HttpContent) {
      HttpContent chunk = (HttpContent) msg;
      if (chunk.content().isReadable()) {
        Buffer buff = Buffer.buffer(chunk.content());
        handleChunk(buff);
      }
      if (msg instanceof LastHttpContent) {
        if (!paused) {
          handleEnd();
        } else {
          // Requeue
          pending.add(LastHttpContent.EMPTY_LAST_CONTENT);
        }
      }
    }     } else if (msg instanceof WebSocketFrameInternal) {
    WebSocketFrameInternal frame = (WebSocketFrameInternal) msg;
    handleWsFrame(frame);
  }
  checkNextTick();
}
```

```java
private void processMessage(Object msg) {
  if (msg instanceof HttpRequest) {
    HttpRequest request = (HttpRequest) msg;
    if (request.decoderResult().isFailure()) {
```

process error

```java
      handleError(request);
      return;
    }
```

process request

```java
    if (server.options().isHandle100ContinueAutomatically() && HttpUtil.is100ContinueExpected(reques
      write100Continue();
    }
    HttpServerResponseImpl resp = new HttpServerResponseImpl(vertx,  conn: this, request);
    HttpServerRequestImpl req = new HttpServerRequestImpl( conn: this, request, resp);
    currentRequest = req;
    pendingResponse = resp;
    if (metrics.isEnabled()) {
      requestMetric = metrics.requestBegin(metric(), req);
    }
    requestHandler.handle(req);
  } else if (msg instanceof HttpContent) {
```

process body

```java
    HttpContent content = (HttpContent) msg;
    handleContent(content);
  } else {
    WebSocketFrameInternal frame = (WebSocketFrameInternal) msg;
    handleWsFrame(frame);
  }
  checkNextTick();
}
```

reduce method size to favor inlining

process error

process request

process body

```java
private void processMessage(Object msg) {
  if (msg instanceof HttpRequest) {
    HttpRequest request = (HttpRequest) msg;
    if (request.decoderResult().isFailure()) {
      handleError(request);
      return;
    }
    if (server.options().isHandle100ContinueAutomatically() && HttpUtil.is100ContinueExpected(reques
      write100Continue();
    }
    HttpServerResponseImpl resp = new HttpServerResponseImpl(vertx, conn: this, request);
    HttpServerRequestImpl req = new HttpServerRequestImpl( conn: this, request, resp);
    currentRequest = req;
    pendingResponse = resp;
    if (metrics.isEnabled()) {
      requestMetric = metrics.requestBegin(metric(), req);
    }
    requestHandler.handle(req);
  } else if (msg instanceof HttpContent) {
    HttpContent content = (HttpContent) msg;
    handleContent(content);
  } else {
    WebSocketFrameInternal frame = (WebSocketFrameInternal) msg;
    handleWsFrame(frame);
  }
  checkNextTick();
}
```

b2073fa091d64a1dfe06699bca1a8befddb5a805

```
private void processMessage(Object msg) {
  if (msg instanceof HttpRequest) {
    HttpRequest request = (HttpRequest) msg;
    if (request.decoderResult().isFailure()) {
      handleError(request);
      return;
    }
    if (server.options().isHandle100ContinueAutomatically() && HttpUtil.is100ContinueExpected(reques
      write100Continue();
    }
    HttpServerResponseImpl resp = new HttpServerResponseImpl(vertx, conn: this, request);
    HttpServerRequestImpl req = new HttpServerRequestImpl( conn: this, request, resp);
    currentRequest = req;
    pendingResponse = resp;
    if (metrics.isEnabled()) {
      requestMetric = metrics.requestBegin(metric(), req);
    }
    requestHandler.handle(req);
  } else if (msg instanceof HttpContent) {
    HttpContent content = (HttpContent) msg;
    handleContent(content);
  } else {
    WebSocketFrameInternal frame = (WebSocketFrameInternal) msg;
    handleWsFrame(frame);
  }
  checkNextTick();
}
```

process error

process request

process body

2. inline by hand

Batch to amortise costs

```
chctx.fireChannelRead(msg)
```

```
// class VertxHandler
void channelRead(Object msg) {
  Connection conn = getConnection();
  Context ctx = conn.getContext();
  context.executeFromIO(conn::startRead());
  channelRead(conn, msg);
}

// class VertxHttpHandler extends VertxHandler
void channelRead(Connection conn, Object msg) {
  conn.handleMessage(msg);
}
```

```
void startRead() { ... }
```

```
void handleMessage(Object msg) { ... }
```

799df9e602eabcd51b56052e20cc7d05134ff901

```
chctx.fireChannelRead(msg)
```

```
// class VertxHandler
public void channelRead(ChannelHandlerContext chctx, Object msg) {
  Connection conn = getConnection();
  Context ctx = conn.getContext();
  context.executeFromIO(() -> {
    conn.startRead();
    conn.handleMessage(msg);
  });
}
```

Batch here

```
void startRead() {
}
```

```
void handleMessage(Object msg) { ... }
```

The fastest code is the code that never runs

Netty

Vert.x

Application

```
req.response()
    .end("Hello World");
```

Netty

Vert.x

Application

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}
```

```
req.response()
  .end("Hello World");
```

Netty

Vert.x

Application

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}

void queueForWrite(Object msg) {
   needsFlush = true;
   channel.write(encode(obj));
}
```

```
req.response()
   .end("Hello World");
```

Netty

```
ChannelFuture write(Object msg) {
  return pipeline.write(msg);
}
```

Vert.x

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}

void queueForWrite(Object msg) {
  needsFlush = true;
  channel.write(encode(obj));
}
```

Application

```
req.response()
  .end("Hello World");
```

Netty

```
ChannelFuture write(Object msg) {
  return pipeline.write(msg);
}
```

Vert.x

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}

void queueForWrite(Object msg) {
  needsFlush = true;
  channel.write(encode(obj));
}

// default implementation (inherited)
void write(ChannelHandlerContext ctx,
Object msg, ChannelPromise promise) {
  ctx.write(msg, promise);
}
```

Application

```
req.response()
  .end("Hello World");
```

Netty

```
ChannelFuture write(Object msg) {
  return pipeline.write(msg);
}
```

```
void write(Object msg,
ChannelPromise promise) {
  next.invoke(msg, promise)
}
```

Vert.x

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}
```

```
void queueForWrite(Object msg) {
  needsFlush = true;
  channel.write(encode(obj));
}
```

```
// default implementation (inherited)
void write(ChannelHandlerContext ctx,
Object msg, ChannelPromise promise) {
  ctx.write(msg, promise);
}
```

Application

```
req.response()
  .end("Hello World");
```

Netty

~~ChannelFuture write(Object msg) {~~
~~return pipeline.write(msg);~~
}

void write(Object msg,
ChannelPromise promise) {
next.invoke(msg, promise)
}

Vert.x

void end(Buffer buffer) {
FullHttpResponse msg = ...
queueForWrite(msg);
}

void queueForWrite(Object msg) {
needsFlush = true;
~~channel.write(encode(obj));~~
}

// default implementation (inherited)
void write(ChannelHandlerContext ctx,
Object msg, ChannelPromise promise) {
~~ctx.write(msg, promise);~~
}

Application

req.response()
.end("Hello World");

Netty

Vert.x

Application

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}

void queueForWrite(Object msg) {
   needsFlush = true;
   chctx.write(encode(obj));
}
```

```
req.response()
   .end("Hello World");
```

the fastest code is the code that never runs

```
void write(Object msg,
ChannelPromise promise) {
  next.invoke(msg, promise)
}
```

217b17c78cd54103ae98557510a7ac431e17c5ea

Reduce object allocation

Netty

Vert.x

Application

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}

void queueForWrite(Object msg) {
   needsFlush = true;
   chctx.write(encode(obj));
}
```

```
req.response()
  .end("Hello World");
```

```
void write(Object msg) {
  write(msg, newPromise());
}

void write(Object msg,
ChannelPromise promise) {
  next.invoke(msg, promise)
}
```

Netty

Vert.x

Application

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}

void queueForWrite(Object msg) {
   needsFlush = true;
   chctx.write(encode(obj));
}
```

```
req.response()
  .end("Hello World");
```

```
void write(Object msg) {
  write(msg, newPromise());
}

void write(Object msg,
ChannelPromise promise) {
  next.invoke(msg, promise)
}
```

Netty | Vert.x | Application

```
void end(Buffer buffer) {
  FullHttpResponse msg = ...
  queueForWrite(msg);
}

void queueForWrite(Object msg) {
   needsFlush = true;
   chctx.write(obj,
                channel.voidPromise());
}
```

```
req.response()
  .end("Hello World");
```

reduce GC allocation
by using VoidPromise

```
void write(Object msg,
ChannelPromise promise) {
  next.invoke(msg, promise)
}
```

6b9788dec6e1147782a3a7017ead067778095cba

Cache expensive operations

Vert.x

```java
void setConnection(Connection conn) {
   this.conn = conn;
}


void channelReadComplete(ChannelHandlerContext ctx) {
   Runnable task = conn::endReadAndFlush();
   // Need to use executeFromIO to avoid race conditions
   context.executeFromIO(task);
}

void endReadAndFlush() {
  if (needFlush) {
    needFlush = false;
    channel.flush();
  }
}
```

Vert.x

```java
void setConnection(Connection conn) {
   this.conn = conn;
}


void channelReadComplete(ChannelHandlerContext ctx) {
   Runnable task = conn::endReadAndFlush();
   // Need to use executeFromIO to avoid race conditions
   context.executeFromIO(task);
}

void endReadAndFlush() {
  if (needFlush) {
     needFlush = false;
     channel.flush();
  }
}
```

Called for every each flush

Vert.x

```
void setConnection(Connection conn) {
   this.conn = conn;
   this.task = conn::endReadAndFlush();
}

void channelReadComplete(ChannelHandlerContext ctx) {
   Runnable task = conn::endReadAndFlush();
   // Need to use executeFromIO to avoid race conditions
   context.executeFromIO(task);
}

void endReadAndFlush() {
  if (needFlush) {
    needFlush = false;
    channel.flush();
  }
}
```

Now called when the connection is created

Extra optimisations

- ✓ Faster HTTP header encoding
- ✓ Cache complex conditions

1	tokio-minihttp	2,738,380	100.0%	Mcr	Rus	Rus	tok	Lin	Rea
2	ulib-plaintext_fit	2,684,333	98.0%	Plt	C++	Non	ULi	Lin	Rea
3	ulib	2,679,274	97.8%	Plt	C++	Non	ULi	Lin	Rea
4	fasthttp	2,665,278	97.3%	Plt	Go	Non	Non	Lin	Rea
5	rapidoid-http-fast	2,654,663	96.9%	Plt	Jav	Rap	Non	Lin	Rea
6	hyper	2,633,824	96.2%	Mcr	Rus	Rus	Hyp	Lin	Rea
7	actix	2,633,461	96.2%	Mcr	Rus	Non	act	Lin	Rea
8	rapidoid	2,608,986	95.3%	Plt	Jav	Rap	Non	Lin	Rea
9	pronghorn	2,578,628	94.2%	Plt	Kot	pro	Non	Lin	Rea
10	undertow	2,544,955	92.9%	Plt	Jav	Utw	Non	Lin	Rea
11	vertx	2,503,741	91.4%	Plt	Jav	ver	Non	Lin	Rea
12	netty	2,447,315	89.4%	Plt	Jav	Nty	Non	Lin	Rea
13	proteus	2,441,688	89.2%	Mcr	Jav	Utw	Non	Lin	Rea
14	light-4j	2,435,150	88.9%	Plt	Jav	lig	Non	Lin	Rea
15	vertx-web	2,411,118	88.0%	Mcr	Jav	vtx	Non	Lin	Rea
16	japronto	2,356,964	86.1%	Mcr	Py	Non	Non	Lin	Rea
17	aspcore-mw	2,216,718	80.9%	Mcr	C#	.NE	kes	Lin	Rea
18	h2o	2,200,215	80.3%	Plt	C	Non	Non	Lin	Rea

Round #15

Plaintext recap

- ✓ A very aggressive benchmark
- ✓ Bottleneck are CPU and networking flushing
- ✓ Less is more

/db benchmark

/db

- ✓ Choice to use PostgreSQL
- ✓ Determine the actual bottleneck: CPU ? Network ? Database ?
- ✓ 256 concurrent connections: non-blocking versus blocking

First step

- ✓ First improvement: the /updates was actually not using a transaction

The reactive PostgreSQL client

- ✓ Goals
  - Simple, clean and straightforward API
  - Performant
  - Be a client
  - Lightweight
- ✓ Non goals
  - Be a driver
  - Be an abstraction

```java
// Connect directly
PgClient.connect(uri, connection -> {
    // Handle result
});


// Or create a pool of connections
PgClient pool = PgClient.pool(uri);
pool.getConnection(connection -> {
    // Handle result
});
```

```
// Sequential queries

connection.query(query1, result1 -> {
    // Got result 1

    connection.query(query2, result2 -> {
        // Got result 2
    });
});
```

```
// What if we do ?
connection.query(query1, result1 -> {
    // Got result 1
});

connection.query(query2, result2 -> {
    // Got result 2
});
```

- the 2 queries executes concurrently ?
- query1 executes then query2 ?
- query1 executes, query2 executes after ?

QUIZ

Head of line blocking

- ✓ PostgreSQL process one request at a time
- ✓ Send the response after processing
- ✓ Sounds familiar ?

Let's pipeline it

The Reactive Postgres Client

- ✓ A PostgreSQL client
- ✓ Simple and direct API
- ✓ Focusing on performance and low overhead

Other cool features

- ✓ Direct memory to object without intermediary memory copy
- ✓ Efficient flush to minimise expensive system calls
- ✓ RxJava support
- ✓ Domain sockets support
- ✓ Proxy support

1	vertx-postgres	191,353	100.0%
2	ulib-postgres_fit	187,519	98.0%
3	h2o	186,245	97.3%
4	ulib-postgres	182,112	95.2%
5	vertx-web-postgres	176,393	92.2%
6	ulib-mongodb	175,101	91.5%
7	proteus	175,048	91.5%
8	gemini-postgres	174,856	91.4%
9	revenj-jvm	172,092	89.9%
10	undertow-postgresql	166,597	87.1%
11	cpoll_cppsp-postgres	165,134	86.3%
12	actix	163,098	85.2%

Round #15


SPEED
LIMIT
50

Hello,

Another interesting bit that I noticed on the dashboard - vertx-postgres and vertx-web-postgres perform significantly better than the rest on some of the database benchmarks, and I have been curious to see what techniques allow them to achieve that. Apparently the reactive Postgres client used by them supports command pipelining (up to 256 commands by default, and can be changed by the setPipeliningLimit() method of the PgPoolOptions class), and a quick check with Wireshark shows that the database server may return several results in a single TCP packet. Doesn't that conflict with the requirement that every query results in a full round-trip to the server?


Hello,
Ah-ha! I was also wondering about vertx-postgres. I tried using the same Postgres client with the Undertow framework and wasn't able to get anything close to the same performance. If it's just a matter of pipelining and that's enabled default, I think I should've gotten similar results. I wonder what I did wrong...
Anyway, thanks for solving that puzzle. You're right, that does sound like a violation of the round-trip requirement. Perhaps we can add a setPipeliningLimit(1) line and call it a day. Now I wonder if other implementations are doing this but they're slow enough (for other reasons) that no one has noticed. It's a cool feature.
several
every query

Hi,

can you elaborate why it violates the Requirements #6 ?

each query is sent as an individual Postgres command (in the PG protocol) and has its own round trip to the database, besides there is no query aggregation or use of batching.

in asynchronous libraries it is very common to implement this way because it is more efficient, and that's what other DB do as well (like https://redis.io/topics/pipelining)

regards

Julien

Hello,
about vertx-postgres. I tried using the
framework and wasn't able to
just a matter of
Hi,
can you elaborate why it violates the Requirements #6 ?
In summary, we are leaning toward prohibiting pipelining to keep consistent with the original intent of this test type.
If there are more opinions, we'd like to hear them. We'll aim to have a decision one way or the other by roughly the end of the week. Also, bear in mind that any decision could be later reversed.
efficient, and that's what other DB do as well (like https://redis.io/topics/pipelining)
regards
Julien

Pipelining can reduce the number of TCP packets but IMO most importantly it can keep the DB server busier and this is the reason why I believe this should be allowed.
In summary, we are
intent of this test type.
If there are more opinions, we'd
e other by roughly the
end of the week. Also, bear in mind that any
Hello,
Another interestin
about vertx-postgres. I tried using the
framework and wasn't able to
just a matter of
Hi,
can you
each qu
efficient, and that's what other DB do as well (like https://redis.io/topics/p
regards
Julien
reasons)
SPEED


We (TechEmpower) have decided to allow pipelining between the application and database. We'll clarify the requirements to call this out specifically.
Hello,
Hi,
regards
Julien

Let there be pipelining

7. Except where noted, all database queries should be delivered to the database servers as-is and not coalesced or deduplicated at the database driver. In all cases where a database query is required, it is expected that the query will reach and execute on the database server. However, it is permissible to pipeline traffic between the application and database as a network-level optimization. That is, two or more separate queries may be delivered from the application to the database in a single transmission over the network, and likewise for query results from the database back to the application, as long as the queries themselves are executed separately on the database server and not combined into a single complex query.


THE END


Brown Sugar

- ✓ https://github.com/AdoptOpenJDK/jitwatch
- ✓ https://github.com/jvm-profiling-tools/async-profiler
- ✓ https://reactiverse.io/reactive-pg-client/