Reactive applications with Eclipse Vert.x
Thomas Segismont

@vertx_project core team member since 2016
Focused on clustering and monitoring
Senior Software Engineer at redhat since 2012

http://github.com/tsegismont
@tsegismont
Outline

✓ Reactive? Vert.x?
✓ Foundations
✓ Event Bus
✓ Reactive Programming
✓ Outro
Reactive?
Vert.x?
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>123</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>456</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

=A1+A2
=A1+A2

The table shows a simple Excel sheet with two rows. The first row contains a single cell with the number 123, and the second row contains two cells with the numbers 456 and a formula `=A1+A2`. The formula is applied using the `AVERAGE` function in Excel.
Software

Metrics

Messages

Requests

Availability
Reactive systems

Manifesto, Actor, Messages
Resilience, Elasticity, Scalability,
Asynchronous, non-blocking

Reactive streams

“Responding to stimuli”

Data flow
Back-pressure
Non-blocking

Akka, Vert.x

Reactive programming

Reactors, Reactive Spring,
RxJava, Vert.x

Data flow
Events, Observable
Spreadsheets

Akka Streams, Rx v2,
Reactor, Vert.x
Eclipse Vert.x

Open source project started in 2012
Eclipse / Apache licensing
A toolkit for building reactive applications for the JVM
7K ★ on 🐱
Built on top of 🐛Netty

🌐 https://vertx.io
🐦 @vertx_project
Pay the right price

✓ Tiny footprint
✓ Do one thing and do it well
✓ Does not solve other (non) problems such as classloading or IoC
✓ Modular set of extensions
Reactive foundations
while (isRunning) {
    String line = bufferedReader.readLine();
    switch (line.substring(0, 4)) {
        case "ECHO":
            bufferedWriter.write(line);
            break
            // ...
            // other cases ( ... )
            // ...
        default:
            bufferedWriter.write("Unknown command");
    }
}
“When you have a line of text, call C2”

Something else with no blocking call either
Event Loop

Thread

Events
2 event-loops per CPU core by default
public class HttpVerticle extends AbstractVerticle {

    public static void main(String[] args) {
        Vertx vertx = Vertx.vertx();
        vertx.deployVerticle(HttpVerticle.class.getName());
        System.in.read();
        vertx.close();
    }

    @Override
    public void start() throws Exception {
        HttpServer server = vertx.createHttpServer();

        server.requestHandler(request -> {
            request
                .response()
                .end("Hello World");
        }).listen(8080);
    }
}
(demo)
Polling dashboard
monitoring dashboard UI → metrics aggregation → reporting agents
Non-blocking IO benefits

✓ Handle many connections with a few threads
✓ Get away from thread pools
✓ Gracefully handle slow connection
Blocking server

Workers

Backend
Blocking server

![Diagram showing workers and backend with blocking server]
Blocking server

Workers

Backend
Blocking server

Workers

Backend
Blocking server

Workers

Backend
Blocking server

Workers

Backend
Blocking server
Non-blocking server

Event Loop

Backend

Requests

Responses
Non-blocking server

Requests

Responses

Event Loop

Backend
Non-blocking server
Non-blocking server

Requests

Responses

Event Loop

Backend
Non-blocking server
Network events come from event loop threads.
Deployment model

```java
Vertx vertx = Vertx.vertx();

vertx.deployVerticle(HttpVerticle.class.getName(),
    new DeploymentOptions()
        .setInstances(3));
```

1. bind the port  
2. add event loop  
3. add event loop

Deploys 3 instances  3 event loop
Vertx vertx = Vertx.vertx();

// Deploy 5 instances on 5 event loops
vertx.deployVerticle(HttpVerticle.class.getName(),
    new DeploymentOptions()
        .setInstances(5));

// The database verticle uses
// deploy 2 instances on 2 event loops
vertx.deployVerticle(DatabaseVerticle.class.getName(),
    new DeploymentOptions()
        .setInstances(2));
“Regular verticle”
(same event event-loop thread)

Worker verticle
(1 worker thread)

Multi-thread worker verticle

Worker pool
Message passing on the Event Bus
Send to “user.db”

“Details for user 1234?”

Http server verticle

×3

Database client verticle

×10
Event Bus

Send to "user.db"

"Details for user 1234?"

Consumed from "user.db"

Http server verticle

×3

Database client verticle

×10
Http server verticle

```
{data}
```

Database client verticle

```
{data?}
```

"Details for user 1234?"
Asynchronous messaging

“foo.bar”, “foo-bar”, “foo/bar”, …

Point to point (with possible response back)
Publish / subscribe
Headers
DeliveryOptions (e.g., timeouts)

Body

Address
Reply address
“Primitive” types
String, int, double, ...

JSON Object/Array
Polyglot applications, clean boundaries

Custom codecs
For advanced usages
EventBus eb = vertx.eventBus();

eb.consumer("ping-address", message -> {
    System.out.println("Received message: " + message.body());
    message.reply("pong!");
});

EventBus eb = vertx.eventBus();

vertx.setPeriodic(1000, v -> {
    eb.send("ping-address", "ping!", reply -> {
        if (reply.succeeded()) {
            System.out.println("Received reply " + reply.result().body());
        } else {
            System.out.println("No reply");
        }
    });
});
Push dashboard
Distributed across Vert.x nodes
Hazelcast, Ignite, Infinispan, ...

TCP bridge interface
Go, Python, C, JavaScript, Swift, C#, ...

SockJS bridge
Seamless frontend / backend messaging
Reactive Programming with Vert.x and RxJava
RxJava

Data and events flows

Organizing transformation of data and coordination of events

Makes most sense with many sources of events
Motivation

Future<List<T>> not always appropriate

Dealing with latencies

Functional programming influence
public interface File {

    static Optional<File> loadFile(String filename) { ... }
    static Maybe<File> nonBlockingLoadFile(String filename) { ... }

    void synchronousClose();
    Completable nonBlockingClose();

    File copyTo(String path);
    Single<File> nonBlockingCopyTo(String path);

    Iterable<Buffer> content();
    Observable<Buffer> nonBlockingContent();
}

RxJava 2 types
Rxified APIs

Each API type (annotated with `@VertxGen`) has its prefix `io.vertx` replaced by `io.vertx.rxjava`

`io.vertx.core.Vertx` → `io.vertx.rxjava.Vertx`

etc...
Rxified Handler<AsyncResult>

```java
void listen(int port, Handler<AsyncResult<HttpServer>> ar)

Single<HttpServer> rxListen(int port);
```
foo.a(1, res1 -> {
    if (res1.succeeded()) {
        bar.b("abc", 1, res2 -> {
            if (res2.succeeded()) {
                baz.c(res3 -> {
                    dosomething(res1, res2, res3, res4 -> {
                        // (...)
                    });
                });
            }
        });
    }
});
jdbc.rxGetConnection().flatMap(conn → {
    // Now chain some statements using flatmap composition
    Single<ResultSet> resa = conn.rxUpdate("CREATE TABLE test(col VARCHAR(20))")
        .flatMap(result → conn.rxUpdate("INSERT INTO test (col) VALUES ('val1')")
            .flatMap(result → conn.rxUpdate("INSERT INTO test (col) VALUES ('val2')")
                .flatMap(result → conn.rxQuery("SELECT * FROM test"));
    return resa.doAfterTerminate(conn::close);
}).subscribe(resultSet → {
    // Subscribe to the final result
    System.out.println("Results : " + resultSet.getRows());
}, err → {
    System.out.println("Database problem");
    err.printStackTrace();
});
Observable == Event Stream

Observer -> Observable

1. subscribe
2. push
try {
    for (String item : it) {
        ➊ // onNext
    }
    ➋ // onError
} catch (Throwable e) {
    ➌ // onCompleted
}
observable.subscribe(item -> {
    ➊ // onNext
}, error -> {
    ➋ // onError
}, () -> {
    ➌ // onCompleted
});
map \{ \text{circle} \rightarrow \text{diamond} \}
Rxified ReadStream<T>

AsyncFile extends ReadStream<Buffer> {

    ...

    Observable<Buffer> toObservable();

    ...

}
Outro
Unified end-to-end reactive model + ecosystem (not just APIs...)

For all kinds of distributed applications (even the small-scale ones)

Flexible toolkit, not a framework (your needs, your call)
Guide to async programming with Vert.x for Java developers
https://goo.gl/AcWW3A

Building Reactive Microservices in Java
https://goo.gl/ep6yB9

Applications réactives avec Eclipse Vert.x
https://youtu.be/ZkWsilpiSqw

Microservices réactifs avec Eclipse Vert.x et Kubernetes
Combien d'event loops sont créées par défaut dans Vert.x ?
Sur quelle librairie Vert.x se repose pour la gestion réseau ?

Jetty

Netty
Quel est le type fourni par RxJava pour gérer un valeur potentiellement nulle ?

Optional<T>
Perhaps<T>
Maybe<T>
Facultative<T>