Developing reactive web applications with Vert.x 3.0

Marek Schmidt <maschmid@redhat.com>

https://github.com/maschmid/jbug-vertx-demo

What is vert.x

What is vert.x

- Set of building blocks for building reactive applications
- General purpose
- Polyglot (on JVM)
  - Idiomatic APIs for every supported language
  - As of 3.0.0-milestone4 Java, JavaScript, Groovy
  - (vert.x 2.x has Java, JavaScript, Groovy, Ruby, Python, Clojure, Ceylon)

Why to use vert.x?

- Light-weight
- High-performance
- Micro-services
- Modular
- Event driven and non-blocking
- Not an application server :)

HelloWorld.java

```
import io.vertx.core.AbstractVerticle;

public class HelloWorld extends AbstractVerticle {
  @Override
  public void start() throws Exception {
    vertx.createHttpServer().requestHandler(req -> {
      req.response()
        .putHeader("content-type", "text/html")
        .end("<html><body><h1>Hello from vert.x!" +
          "</h1></body></html>");
    }).listen(8080);
  }
}
```

Demo! helloworld

Hello world.js

```
vertx.createHttpServer().requestHandler(function(req) {
  req.response().putHeader("content-type",
"text/html").end("<html><body><h1>Hello from vert.x!
</h1></body></html>");
}).listen(8080)
```

A bit about the API...

- Fluent
- Async
- Doesn't block the calling thread (except *Sync calls)

Don't block the Event loop!

Handler<AsyncResult<T>>

- ```
  vertx.deployVerticle("foo.js", asyncResult → {
    if (asyncResult.succeeded()) {
        String deploymentId = asyncResult.result();
        …
    }
    else {
        Throwable failureCause = asyncResult.cause();
        …
    }
  });
  ```
  ```

Running blocking code

```
vertx.executeBlocking(future -> {
  // Call blocking API that takes a significant amount of time to return
  String result = someAPI.blockingMethod("hello");
  future.complete(result);
}, res -> {
  System.out.println("The result is: " + res.result());
});
```

...or use worker verticles...

Verticles

- Verticle
  - Optional actor-like deployment and concurrency model
  - chunks of code that get deployed and run by Vert.x
  - Can be written in any (supported) language
- Verticle types
  - Standard
  - Worker
  - Multi-threaded worker

Event loop threads

- Multiple event loops threads (1 per CPU core)
- Event loop threads may have assigned multiple verticles
- A (standard) verticle always sticks to the same event loop thread
  - No locking needed for verticle data

Demo 2

- Event loop threads demo
  - Multiple verticle instances on the same JVM
    - socket binds sharing
  - Lock-free

multiple_instances
multiple_instances_nolock

EventBus

- Distributed event bus
- Messaging
  - Publish-subscribe
  - Point-to-point
  - Request-response (-response, ...)
- Addressing
  - Just a string

EventBus

- `eb.send("some.address", "Hello”, optReplyHandler);`
- `eb.publish("some.address", "Hello to all”);`

- ```
  eb.consumer("some.address", message → {
    message.reply("Replying to " + message.body());
  });
  ```

- Message types
  - Primitives, Buffer
    JsonObject, JsonArray
    custom codec
- DeliveryOptions
  - Reply timeout
  - headers
  ```

Demo 3

- distributedeventbus demo

Metrics

- Integration with Dropwizard Metrics
  - Gauge, Counter, Histogram, ThroughputMeter, …
- EventBus
- TCP/UDP/Http servers metrics
- JMX

Demo4

- Monitor pending eventbus messages

Other things in Vert.x Core

- Buffers, Streams, Pumps
- UDP, TCP server / client, TLS
- HTTP server / client, websockets
- SharedData
  - Local
  - Cluster-wide
- Filesystem
- DNS client

“Apex” (TBR)

- Inspired by Node.js Express, Ruby Sinatra
- Features

- Routing
- Extraction of parameters
- Content negotiation
- Request body handling
- Body size limits
- Cookies
- Multipart forms
- Multipart file uploads
- Sub routers
- Session support - both local and clustered
- CORS
- Error page handler

- Basic Authentication
- Redirect based authentication
- User/role/permission authorisation
- Favicon handling
- Templates
  - Handlebars
  - Jade
  - MVEL
  - Thymeleaf
- Response time handler
- Static file serving
- Request timeout support

Apex Routes

```
Route route1 =
router.route("/some/path/").handler(routingContext -> {
  HttpServerResponse response = routingContext.response();
  response.setChunked(true);
  response.write("route1\n");
  routingContext.vertx().setTimer(5000, tid ->
routingContext.next());
});

Route route2 =
router.route("/some/path/").handler(routingContext -> {
  HttpServerResponse response = routingContext.response();
  response.write("route2\n");
  response.end();
});
```

apex

```
Router router = Router.router(vertx);

router.route().handler(BodyHandler.create());

router.put("/docs/:docID").handler(ctx → {
  String id = ctx.request().getParam("docID");

  JsonObject doc = ctx.getBodyAsJson().put("_id", id);

  mongoService.save("docs", doc, res → {
    if (res.succeeded()) {
       ctx.response().end("Document stored.");
    }
    else {
       ctx.response().setStatusCode(500).end();
    }
  });
});
```

SockJS EventBus Bridge

```
BridgeOptions opts = new BridgeOptions()
      .addInboundPermitted(new
PermittedOptions().setAddress("client.to.bus"))
      .addOutboundPermitted(new
PermittedOptions().setAddress("bus.to.client"));

SockJSHandler ebHandler =
SockJSHandler.create(vertx).bridge(opts);

router.route("/eventbus/*").handler(ebHandler);
```

SockJS EventBus Bridge

- Client side (browser)

```
<script src="sockjs.min.js"></script>
<script src="vertxbus.js"></script>

var eb = new vertx.EventBus("/eventbus/");
eb.onopen = function () {
  eb.registerHandler("bus.to.client",
    function(msg) {
    }
  );
}
```

SockJS EventBus Bridge Security

- BridgeOptions additionaly allow “roles allowed”
  - Still not enough to verify identity of the client
    - e.g. chat message {'user':'Alice','message':'...'}
  - Feature request to add current principal to client->bus message headers
  - Current workaround: EventBusBridgeHook

RxJava Integration

"ReactiveX is a combination of the best ideas from the Observer pattern, the Iterator pattern, and functional programming"

– http://reactivex.io

Observables

	Single items	Multiple items
synchronous	T getData()	Iterable<T> getData()
asynchronous	Future<T> getData()	Observable<T> getData()

- Observable *emits* items
- Observer can *subscribe* to an Observable
  - onNext(T)
  - onError(Throwable)
  - onCompleted()

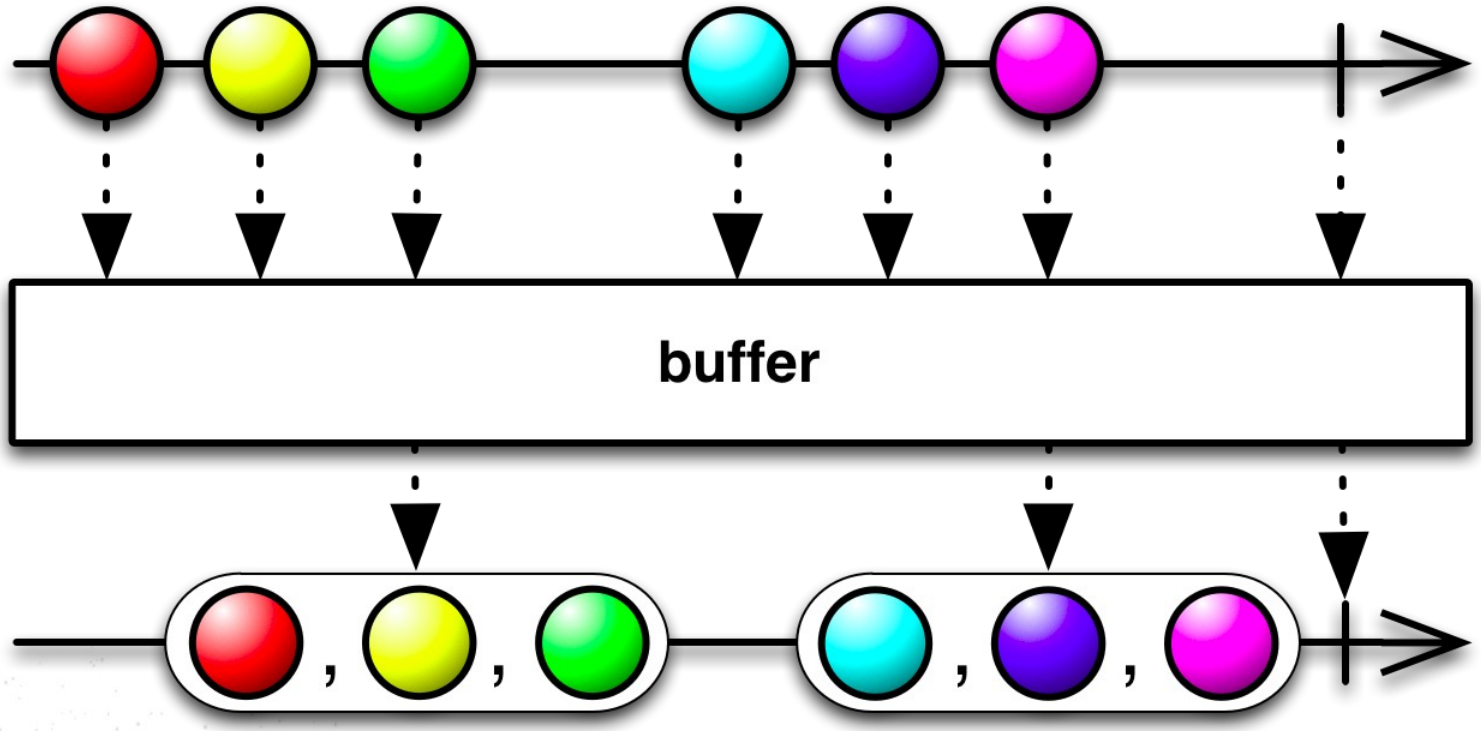
Reactive Extensions

- Operators to transform, combine, manipulate, and work with the sequences of items emitted by Observables.

Example

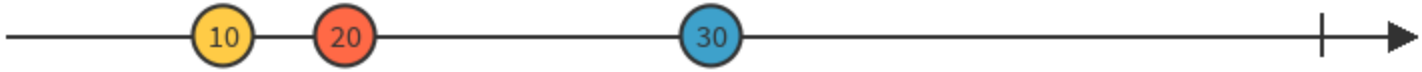
- Input: observable emitting realtime log events
  - 0.0s { 'level': 'INFO', 'message': “some info msg'}
  - 0.5s { 'level': 'ERROR', 'message': “exeception occurred!!'}
  - 1.5s { 'level': 'INFO', 'message': “another boring info message..."}
- Output: observable emitting every second statistics of log levels
  - 1.0s {'INFO': 1, 'ERROR': 1}
  - 2.0s {'INFO': 1}

Buffer



Map


```
map(x => 10 * x)
```



Example

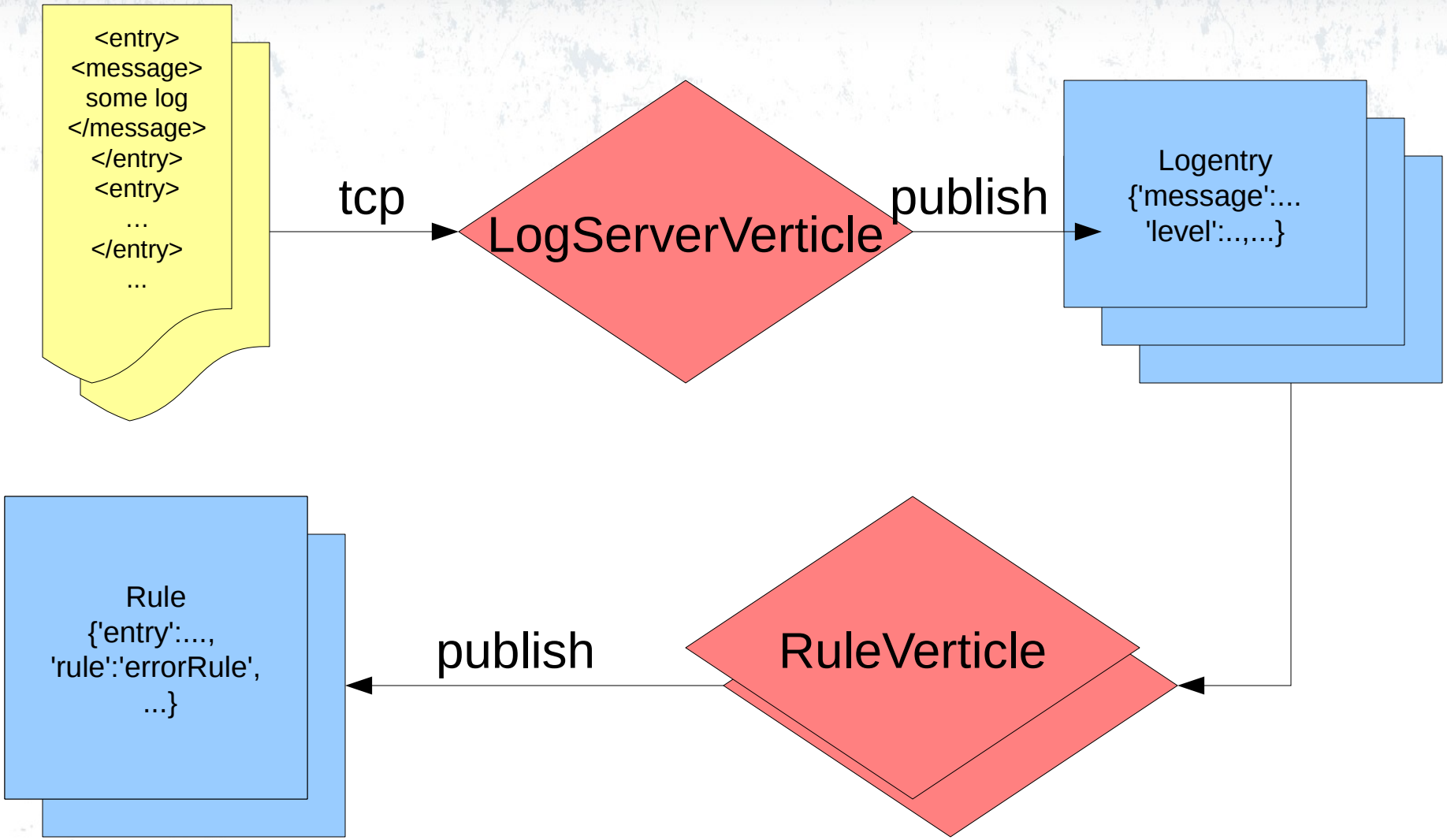
- `import io.vertx.rxjava.core.AbstractVerticle; // !!!`
- `Observable<Long> timerObs = vertx.periodicStream(1000).toObservable();`
- ```
  Observable<JsonObject> logeventsObs =
  vertx.eventBus().<JsonObject>consumer("logevent")
  .bodyStream().toObservable();
  ```
- `Observable<List<JsonObject>> bufferObs = logeventsObs.buffer(timerObs);`
- ```
  Observable<Map<String, Long>> countsObs = bufferObs.
    map(samples →
      samples.stream().collect(
        Collectors.groupingBy(ruleEvent → ruleEvent.getString("level"),
                              Collectors.counting())));
  ```
- ```
  countsObs.subscribe(counts → {
    // ... missing bits.. convert Map to JsonObject ...
    vertx.eventBus().publish("aggregatedlogs", countsAsJson)
  });
  ```
  ```

Apex Example

```
ObservableHandler<RoutingContext> ctxObs = RxHelper.observableHandler();
router.post("/foo").handler(ctxObs.toHandler());
```

Final Demo

- Server monitoring application
- Servers stream logs in XML
- Rules matching interesting patterns
- UI showing aggregated statistics and logs events


<entry>
<message>
some log
</message>
</entry>
<entry>
...
</entry>
...
tcp
LogServerVerticle
publish
Logentry
{'message':...
'level':...,...}
Rule
{'entry':...,
'rule':'errorRule',
...}
publish
RuleVerticle

Other verticles 1/2

- RuleCountAggregatorVerticle
  - produces stats of rules fired every second
  - (RxJava, Observable, Cluster-wide lock (singleton))
- RuleRegistryVerticle
  - Maintains cluster-wide rules definition
  - Creates rule verticles
  - (Cluster-wide map, programmatically deploying verticles)

Other verticles 2/2

- UiServerVerticle
  - Web server
  - (Apex, REST, SockJS EventBus Bridge)
- UnreceivedRuleVerticle
  - Custom rule verticle
  - RxJava, Observable

Demo data

- “Server1 send to Server2 XXX”
- “Server2 receive from Server1 XXX”
- Display events where message was sent but not received in 10 seconds.

Summary

- Vert.x is a great platform for building reactive applications
- Use reactive APIs (Observables) or async callbacks, whatever makes sense

Questions?