Samuele Dell'Angelo

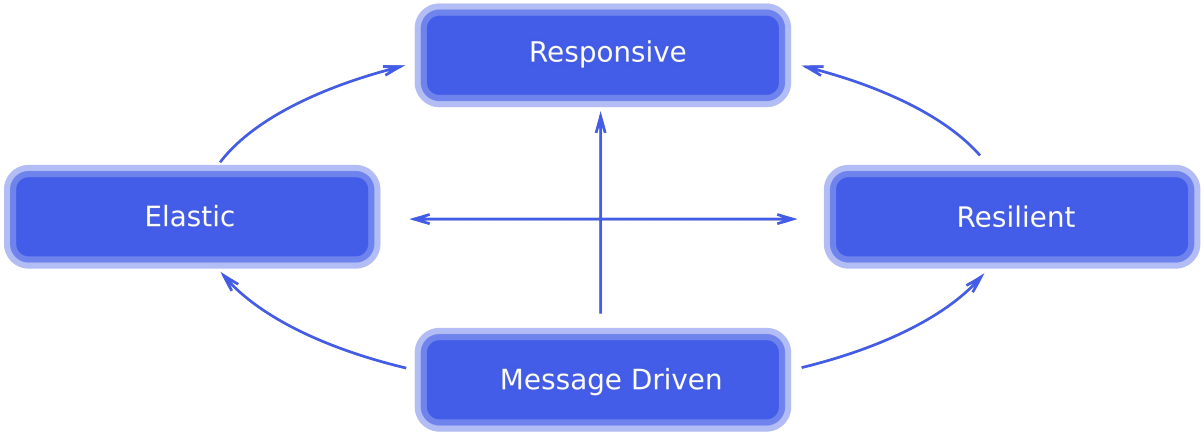
Build your reactive web application with Vert.x

samuele@redhat.com – Red Hat

We Are Reactive

Be Reactive

*Systems built as Reactive Systems are more **flexible, loosely-coupled and scalable**.*
This makes them easier to develop and amenable to change.
They are significantly more tolerant of failure and when failure does occur
they meet it with elegance rather than disaster.
Reactive Systems are highly responsive, giving users effective interactive feedback.



Introducing...

VERT.X

Lightweight reactive application platform

- Polyglot
- High performance
- Superficially similar to Node.js – but not a clone!
- Asynchronous APIs
- Simple but not simplistic
- Embeddable

Introducing...

VERT.X

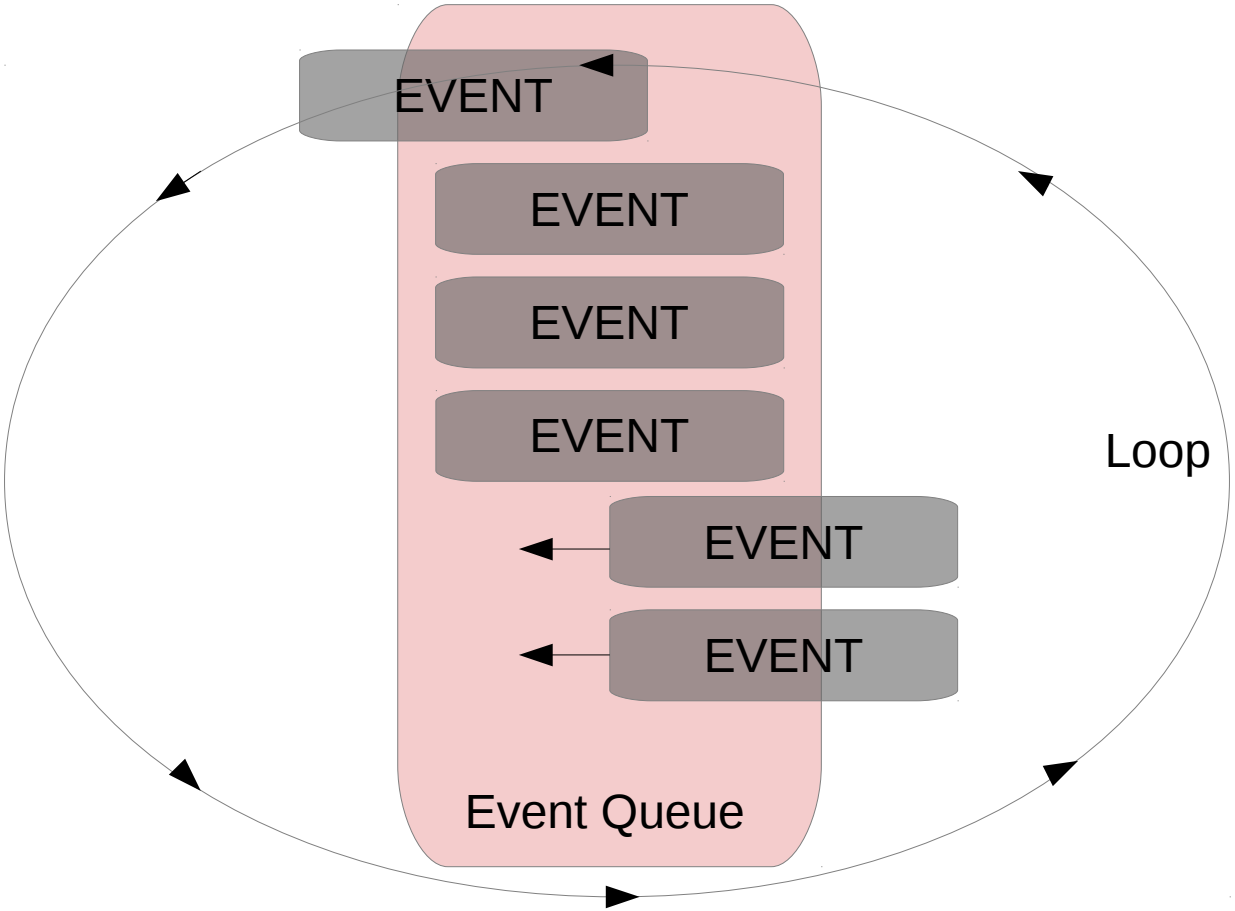
The Event Loop

Event Loop, as you will guess from the name, is a method for checking whether there is a new event in the infinite loop, and calling an event handler when an event has been received. As such, the asynchronous server application and the event-based server application are different terms indicating an identical target, similar to 'enharmonic' for music. To use the Event Loop method, all I/Os should be managed as events..

Introducing...

VERT.X

The Event Loop



Introducing...

VERT.X

The Event Loop

NEVER STOP THE EVENT LOOP

Introducing...

VERT.X

Verticle

- Execution unit of Vert.x
- Single threaded – less scope for race conditions
- Verticles communicate by message passing
- You could handle blocking operation with worker Verticle

Introducing...

VERT.X

The container object

- Injected in all verticle instances
- Represents the view of the container in witch the verticle is running
- contains methods for deploying and undeploying verticle and modules, and also allows config, environment variables and a logger

```
container.deployModule("io.vertx~mod-mysql-postgresql_2.10~0.3.1", appConfig);
container.deployModule("io.vertx~mod-web-server~2.0.0-final", webConfig);
container.deployVerticle("it.redhat.services.ValidatorService");
```

```
container.logger().info("Webserver started, listening on port: 8888");
```

Introducing...

VERT.X

The vertx object

- Injected in all verticle instances
- Provides access to the Vert.x core API.
- Grant access to TCP, HTTP, file system access, event bus, timers etc.

```
HttpServer httpServer =  vertx.createHttpServer();
vertx.eventBus().registerHandler("email-validator",new ValidatorHandler(vertx, container));
vertx.eventBus().send("telnumber-validator", number, (Handler) (message) -> {
```

Introducing...

VERT.X

Handlers

- The programming model is based on event handling
- Handlers should be specific for an event

```
public class GetAllMemberHandler extends ResultsHandler implements Handler<HttpServerRequest> {

    public GetAllMemberHandler(Vertx vertx, Container container) { super(vertx, container); }

    @Override
    public void handle(final HttpServerRequest httpServerRequest) {
        JsonObject select = new JsonObject()
                .putString("action","select")
                .putString("table","Member");

        executeQuerySendResponse(select, httpServerRequest);
    }

}
```

Introducing...

VERT.X

Handlers

- Once you wrote your handler, assign it to an event

```
RouteMatcher routes = new RouteMatcher();

routes.get("/rest/members" , new GetAllMemberHandler(vertx, container));
routes.options("/rest/members", new MemberHandlerOptions(vertx, container));
routes.options("/rest/members/:id", new MemberHandlerOptions(vertx, container));
routes.post("/rest/members", new RegisterMemberHandlerPost(vertx, container));
routes.get("/rest/members/:id" , new GetMemberByIdHandler(vertx, container));
routes.delete("/rest/members/:id", new DeleteMemberByIdHandler(vertx, container));
```

Introducing...

VERT.X

JSON support

- Vert.x has native JSON support
- JSON is the preferred way to represents data in Vart.x
- A JSON object is represented by instances of org.vertx.java.core.json.JsonObject. A JSON array is represented by instances of org.vertx.java.core.json.JsonArray.

```
JsonObject query = new JsonObject()
        .putString("action", "insert")
        .putString("table", "Member")
        .putArray("fields", new JsonArray().add("email").add("name").add("phone_number"))
        .putArray("values", new JsonArray().addArray(new JsonArray().add(email).add(name).add(number)));
```

Introducing...

VERT.X

Event Bus

- The nervous system of Vert.x
- Verticles send messages over the event bus
- Point to point. Publish/Subscribe. Request/Response
- Pass strings, buffers, primitive types or JSON
- JSON messages are preferred for structured data
- Cluster
- Event bus extends to client side JavaScript too

Introducing...

VERT.X

Event Bus

- Addresses are simple string
- You have to register an handler to an address

```
vertx.eventBus().registerHandler("email-validator",new ValidatorHandler(vertx, container));
```

Introducing...

VERT.X

Event Bus

- You can send a message to all handlers (publish/subscribe) or send it to a single handler instance (point to point) with a round robin load balancing.

```
vertx.eventBus().send("telnumber-validator", number, (Handler) (message) -> {
vertx.eventBus().publish("newmember-address","New member registered: "+name);
```

Introducing...

VERT.X

Event Bus

- You can access the event bus from the browser with the Event Bus Bridge.
- It uses sock.js for the communication
- vertxbus.js is provided
- You could find a good angular.js module at: https://github.com/knalli/angular-vertxbus

Introducing...

VERT.X

Event Bus

```
<script src="http://cdn.sockjs.org/sockjs-0.3.4.min.js"></script>
<script src='vertxbus.js'></script>

<script>

    var eb = new vertx.EventBus('http://localhost:8080/eventbus');

    eb.onopen = function() {

      eb.registerHandler('some-address', function(message) {

        console.log('received a message: ' + JSON.stringify(message);

      });

      eb.send('some-address', {name: 'tim', age: 587});

    }

</script>
```

Introducing...

VERT.X

Shared Data

- Vert.x allow different verticles instances to share data in a safe way.
- You could share ConcurrentHashMap or Set
- Vert.x only allows simple immutable types such as number, boolean and string or Buffer to be used in shared data

```
ConcurrentMap<String, Integer> map = vertx.sharedData().getMap("demo.mymap");

map.put("some-key", 123);
```

Introducing...

VERT.X

Websockets

- Easy as set a websocket handler on an http server.
- Read and write via streams and pumps

```
HttpServer httpServer =  vertx.createHttpServer();

httpServer.websocketHandler(new Handler<ServerWebSocket>() {
    @Override
    public void handle(ServerWebSocket serverWebSocket) {
        Pump.createPump(serverWebSocket,serverWebSocket).start();
    }
});
```

Introducing...

VERT.X

Streams and pumps

- In Vert.x, calls to write data return immediately and writes are internally queued.
- Flow control via ReadStream and WriteStream
- A Pump is an helper class that allow you to pass data from a ReadStream to a WriteStream,
- Pumps automatically pause when the write queue is full and resume when the queue is ready to accept more data.

Introducing...

VERT.X

Modules

- Modules encapsulate code and resources
- One or more modules per application
- Must include a mod.json descriptor file
- Can be runnable or non runnable
- Module class-loaders provide isolation

Introducing...

VERT.X

Microservices

- Independent services
- Deployable as a standalone unit.
- Easily scalable
- Can interoperate with other services.

Introducing...

VERT.X

Microservices

- Vert.x is an enabling technology for microservices.
- You don't have language restriction.
- You can deploy independent units such as modules or verticles.
- You could scale verticle instances on a single platform or across a cluster.
- The Event Bus solves the complexity of the communication layer.

Introducing...

VERT.X

Microservices example

```
public class ValidatorService extends BusModBase {

    private boolean isEmailValid = false;

    public ValidatorService() { super(); }

    public void start() {
        super.start();
        vertx.eventBus().registerHandler("email-validator",new ValidatorHandler(vertx, container));
        vertx.eventBus().registerHandler("telnumber-validator",new TelValidatoriHandler(vertx,container));
        vertx.eventBus().registerHandler("ping-address", (Handler) (message) -> {
                message.reply("pong!");
                container.logger().info("Sent back pong");
        });
        container.logger().info("Validator Service started");
    }

}
```

Introducing...

VERT.X

Microservices dependencies

- Vert.x solves module dependencies using Maven repos

```
container.deployModule("io.vertx~mod-mysql-postgresql_2.10~0.3.1", appConfig);
container.deployModule("io.vertx~mod-web-server~2.0.0-final", webConfig);
container.deployVerticle("it.redhat.services.ValidatorService");
```

Introducing...

VERT.X

High Availability

- Automatic failover of deployed modules
- Nodes can be grouped
- Network partition detection with quora

Introducing...

VERT.X

Cluster

- Shared Event Bus
- Shared Verticle data stored in HashMaps or Set
- Enable HA

Introducing...

VERT.X

Demo

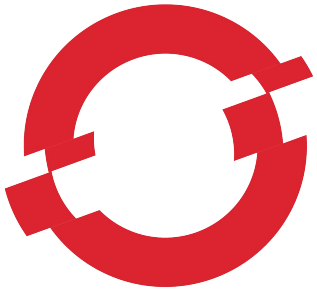
Find the demo source code on:
https://github.com/sdellang/vertx-webapp

Introducing...

VERT.X

Play with Vert.x

Try it using the Vert.x cartridge on www.openshift.com



OPENSHIFT

Introducing...

VERT.X

Introducing...

VERT.X

Thanks

 @samueled80

 sdellang