A Hitchikers Introduction to Vertx

Anatole Tresch, Principal Consultant

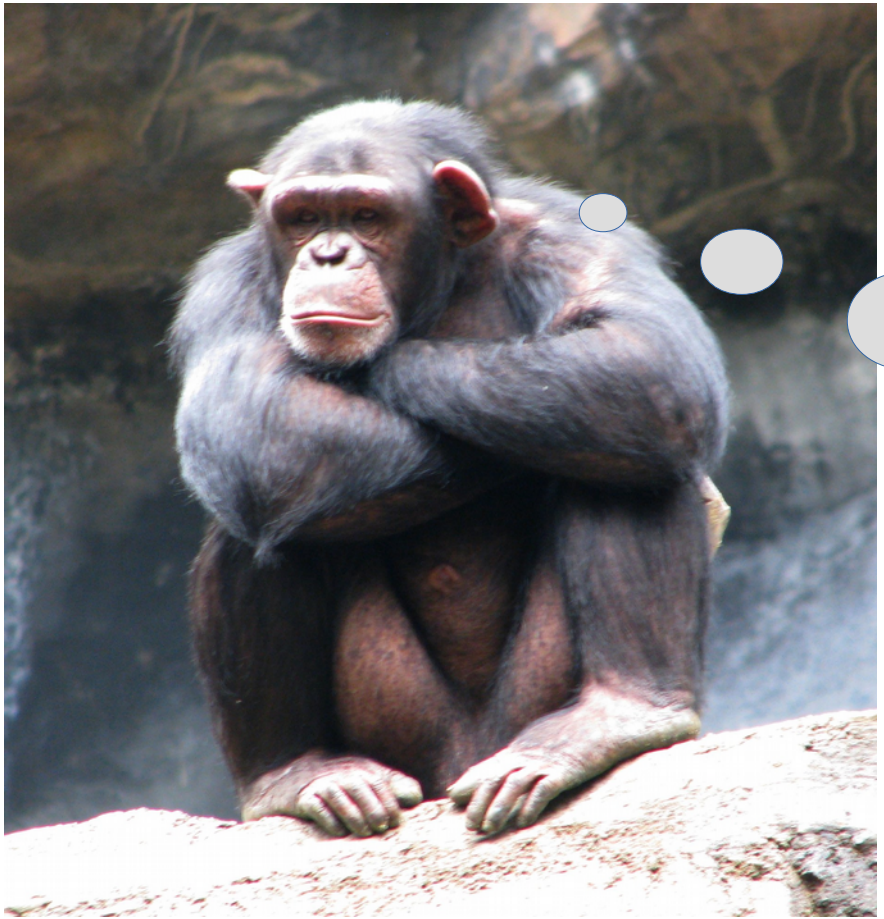
About me...

- Principal Consultant, Trivadis AG (Switzerland)
- Star Spec Lead JSR 354
- Open Source Enthusiast

- Twitter: @atsticks
- anatole@apache.org
- anatole.tresch@trivadis.com

Agenda

- Basics
- Networking
- Clustering
- Vertx Maven Plugin
- There is more...



Vertx?
Is that something
to eat?

Tool-kit

on the JVM

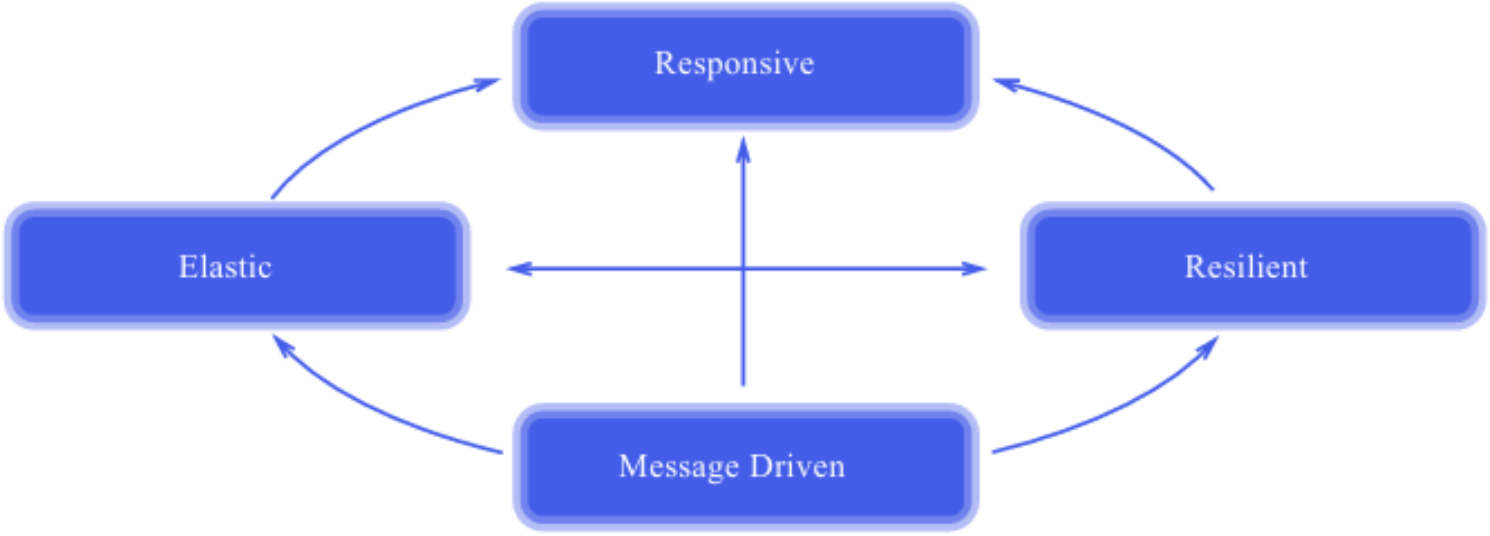
for reactive applications

Toolbox

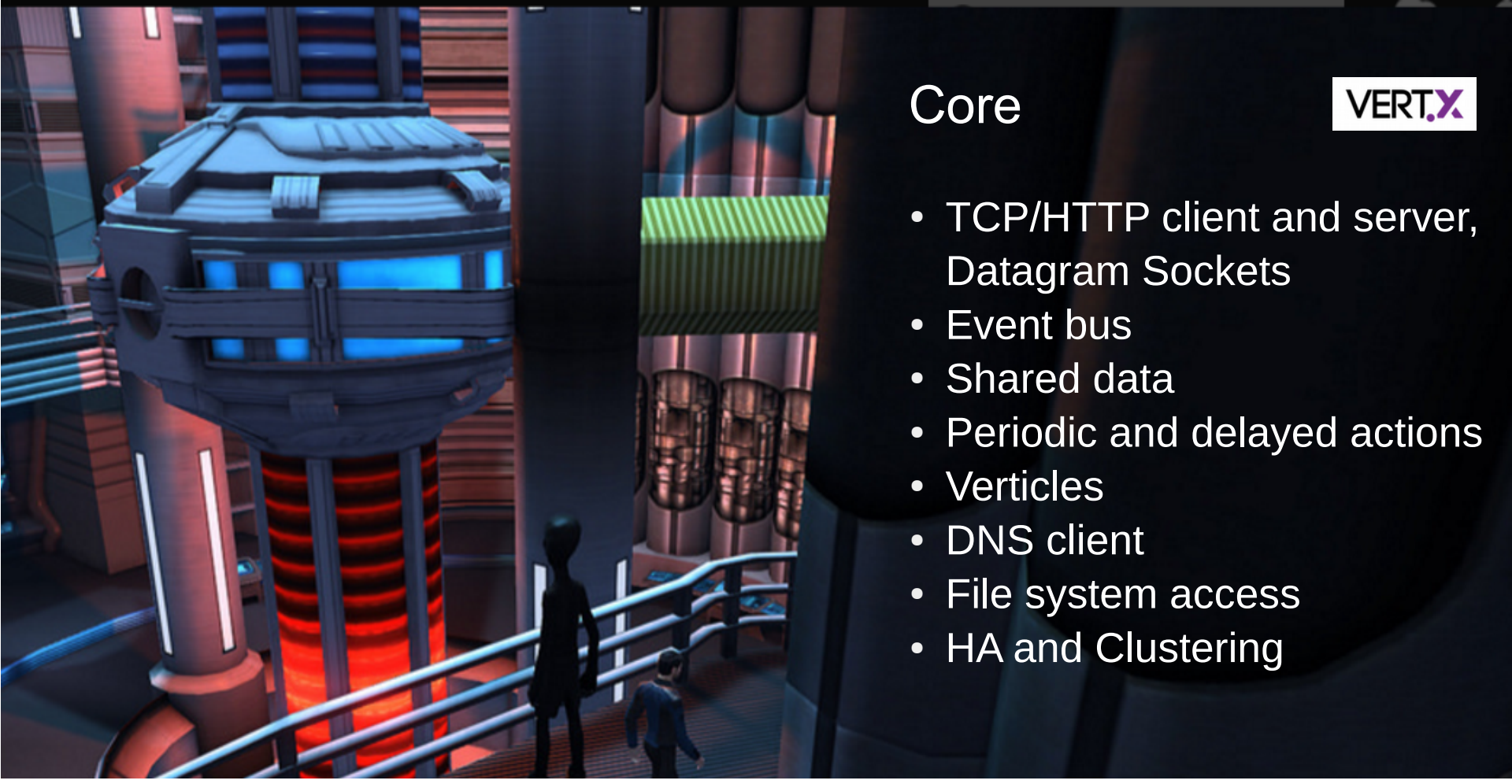
- Vert.x is not a container.
- You use the components you like.
- You can combine Vert.x with all the other libraries you like.

Reactive

VERT.X



Source: http://www.reactivemanifesto.org/



Core

- TCP/HTTP client and server, Datagram Sockets
- Event bus
- Shared data
- Periodic and delayed actions
- Verticles
- DNS client
- File system access
- HA and Clustering

Starting VERT.X

```
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-core</artifactId>
  <version>3.4.1</version>
</dependency>
```

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

```
Vertx vertx = Vertx.vertx(new VertxOptions().setWorkerPoolSize(40));
```

The Eventloop

```
vertx.setPeriodic(1000, id -> {
  // This handler will get called every second
  System.out.println("timer fired!");
});
```

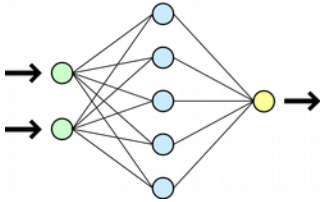
- Events are distributed by the event loop thread aka **Reactor**
- Vertx = **Multi-Reactor** (by default Cores*2)
- *Don‘t block the event loop !!!*

NOTE	Even though a Vertx instance maintains multiple event loops, any particular handler will never be executed concurrently, and in most cases (with the exception of worker verticles) will always be called using the **exact same event loop**.

Running Blocking Code

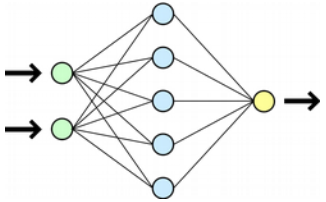
```
vertx.executeBlocking(future -> {
  // Call some blocking API that takes a significant amount of time to return
  String result = someAPI.blockingMethod("hello");
  future.complete(result);
}, res -> {
  System.out.println("The result is: " + res.result());
});
```

The Event Bus



- The nervous system of Vert.x.
- One single event bus instance for every Vert.x instance.
- Allows different parts of your application to communicate.
- Can be bridged to support client side JavaScript.
- A distributed peer-to-peer messaging spanning multiple nodes and browsers.
- Publish/subscribe, point to point, and request-response messaging.

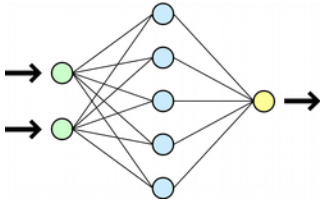
Registering Handlers to the Event Bus



```
EventBus eb = vertx.eventBus();

eb.consumer("news.uk.sport", message -> {
  System.out.println("I have received a message: " + message.body());
});
```

```
EventBus eb = vertx.eventBus();

MessageConsumer<String> consumer = eb.consumer("news.uk.sport");
consumer.handler(message -> {
  System.out.println("I have received a message: " + message.body());
});
```

Publishing Events on the Event Bus



```
eventBus.publish("news.uk.sport", "Yay! Someone kicked a ball");
```

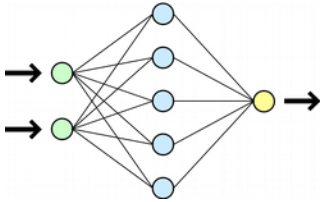
That message will then be delivered to all handlers registered against the address news.uk.sport.

```
eventBus.send("news.uk.sport", "Yay! Someone kicked a ball");
```

Sending a message will result in only one handler registered at the address receiving the message. This is the point to point messaging pattern. The handler is chosen in a non-strict round-robin fashion.

```
DeliveryOptions options = new DeliveryOptions();
options.addHeader("some-header", "some-value");
eventBus.send("news.uk.sport", "Yay! Someone kicked a ball", options);
```

Acknowledging Messages



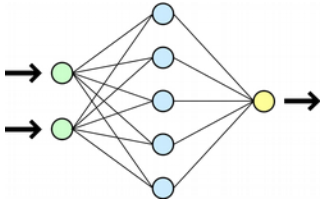
The receiver:

```
MessageConsumer<String> consumer = eventBus.consumer("news.uk.sport");
consumer.handler(message -> {
  System.out.println("I have received a message: " + message.body());
  message.reply("how interesting!");
});
```

The sender:

```
eventBus.send("news.uk.sport", "Yay! Someone kicked a ball across a patch of grass'
  if (ar.succeeded()) {
    System.out.println("Received reply: " + ar.result().body());
  }
});
```

Delivery Options



Sending with timeouts

When sending a message with a reply handler you can specify a timeout in the `DeliveryOptions`.

If a reply is not received within that time, the reply handler will be called with a failure.

The default timeout is 30 seconds.

Send Failures

Message sends can fail for other reasons, including:

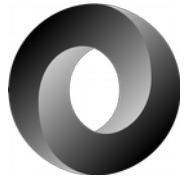
- There are no handlers available to send the message to
- The recipient has explicitly failed the message using `fail`

In all cases the reply handler will be called with the specific failure.

Message Codecs

You can send any object you like across the event bus if you define and register a `message codec` for it.

JSON Support



- Vertx comes with full JSON OOTB (based on Jackson Library)
- JSON can is by default support over the event bus
- `JsonObject.encode()/JsonArray.encode() → String`

```
JsonObject object = new JsonObject();
object.put("foo", "bar").put("num", 123).put("mybool", true);
```

```
JsonArray array = new JsonArray();
array.add("foo").add(123).add(false);
```

```
request.bodyHandler(buff -> {
  JsonObject jsonObject = buff.toJsonObject();
  User javaObject = jsonObject.mapTo(User.class);
});
```

High Availability

Verticles can be deployed with High Availability enabled:

```
vertx run my-verticle.js -ha
```

Buffers

Most data is shuffled around inside Vert.x using buffers.

A buffer is a sequence of zero or more bytes that can read from or written to and which expands automatically as necessary to accommodate any bytes written to it. You can perhaps think of a buffer as smart byte array.

Create a new empty buffer:

```
Buffer buff = Buffer.buffer();
```

Create a buffer from a String. The String will be encoded in the buffer using UTF-8.

```
Buffer buff = Buffer.buffer("some string");
```

Create a buffer from a String: The String will be encoded using the specified encoding, e.g:

```
Buffer buff = Buffer.buffer("some string", "UTF-16");
```

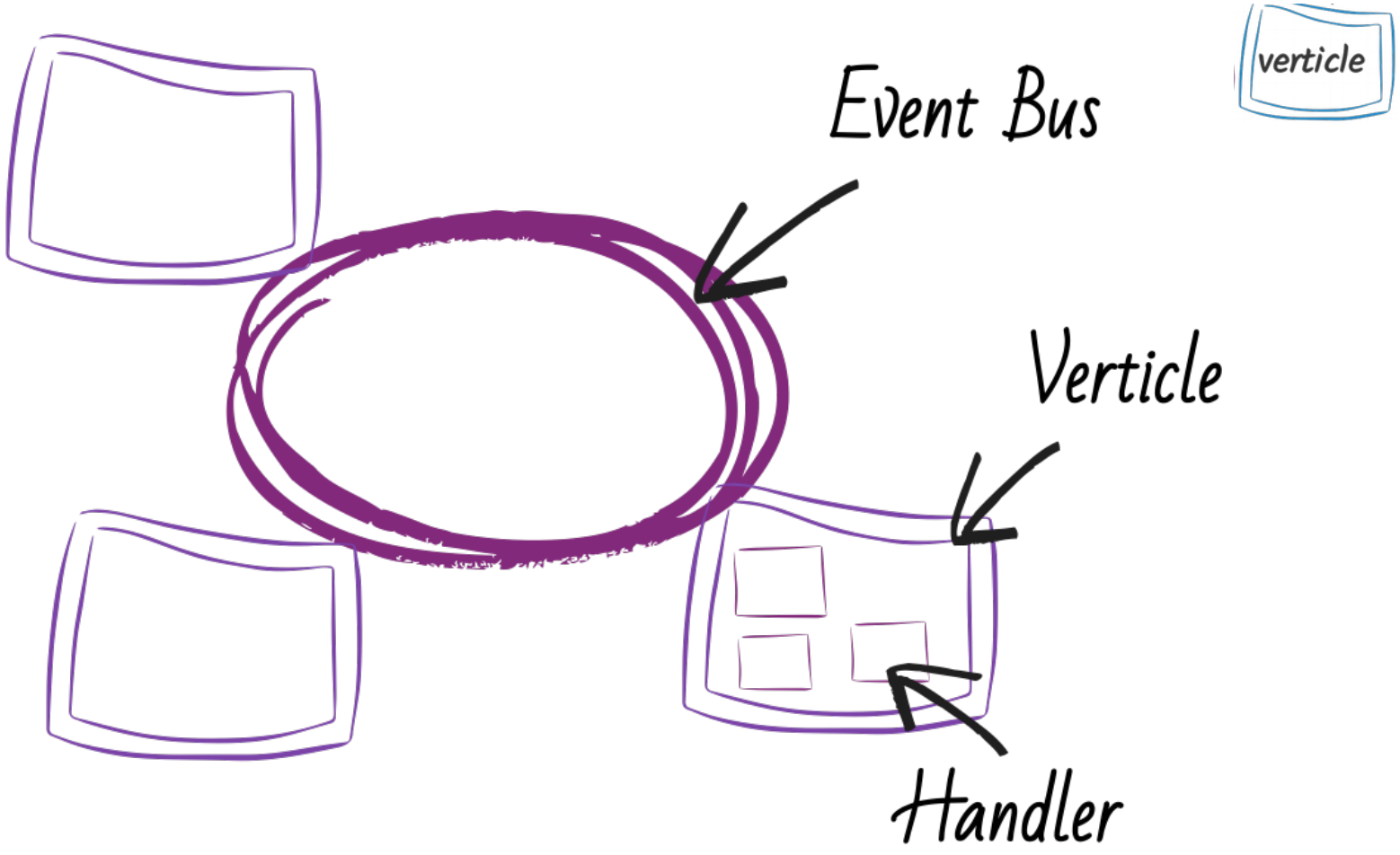
Create a buffer from a byte[]

```
byte[] bytes = new byte[] {1, 3, 5};
Buffer buff = Buffer.buffer(bytes);
```

Verticles

- Simple, scalable, actor-like deployment and concurrency model
- Not a strict actor-model implementation, but similar
- Entirely optional

```
public class MyVerticle extends AbstractVerticle {

  public void start() {
    // Do something
  }

  public void stop(Future<Void> stopFuture) {
    obj.doSomethingThatTakesTime(res -> {
      if (res.succeeded()) {
        stopFuture.complete();
      } else {
        stopFuture.fail();
      }
    });
  }
}
```



Verticle Types

Standard Verticles

These are the most common and useful type - they are always executed using an event loop thread.

Worker Verticles

These run using a thread from the worker pool. An instance is never executed concurrently by more than one thread.

Multi-threaded worker verticles

These run using a thread from the worker pool. An instance can be executed concurrently by more than one thread.

```
DeploymentOptions options = new DeploymentOptions().setWorker(true);
vertx.deployVerticle("com.mycompany.MyOrderProcessorVerticle", options);
```

Deploying Verticles

NOTE Deploying Verticle **instances** is Java only.

```
Verticle myVerticle = new MyVerticle();
vertx.deployVerticle(myVerticle);
```

Or you use references for *VerticleFactories* for polyglot support,,,

```
vertx.deployVerticle("com.mycompany.MyOrderProcessorVerticle");

// Deploy a JavaScript verticle
vertx.deployVerticle("verticles/myverticle.js");

// Deploy a Ruby verticle verticle
vertx.deployVerticle("verticles/my_verticle.rb");
```

Timers and Delays

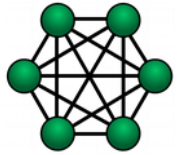
One shot timers:

```
long timerID = vertx.setTimer(1000, id -> {
  System.out.println("And one second later this is printed");
});

System.out.println("First this is printed");
```

Periodic timers:

```
long timerID = vertx.setPeriodic(1000, id -> {
  System.out.println("And every second this is printed");
});

System.out.println("First this is printed");
```

Shared Data



- **Local shared maps**

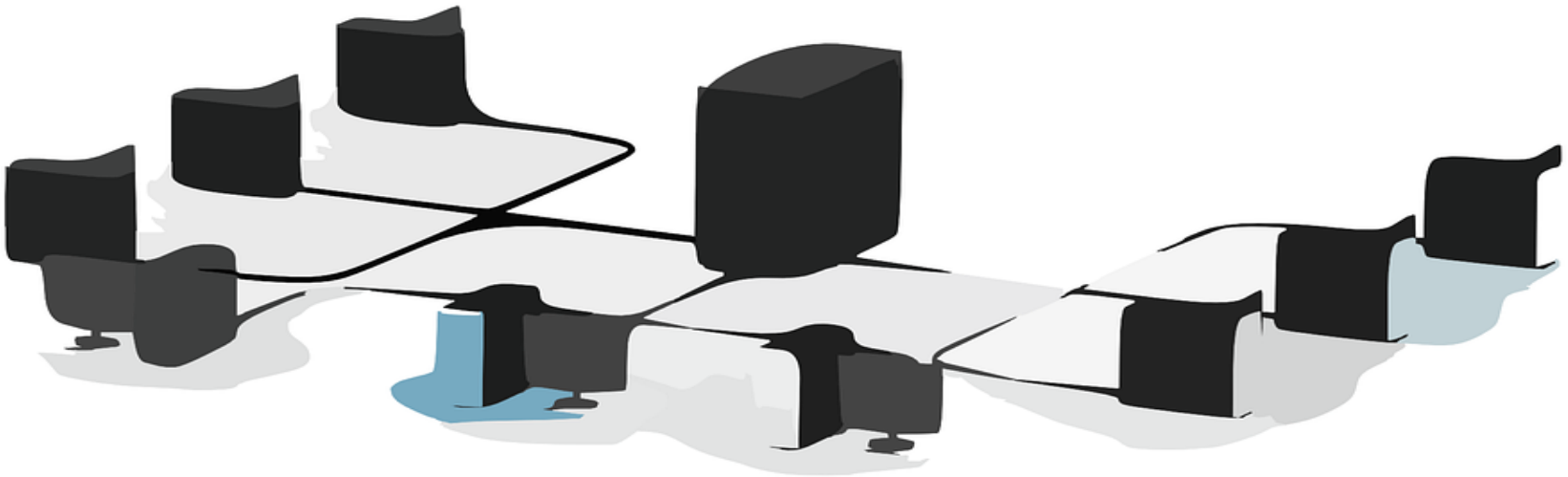
```
SharedData sd = vertx.sharedData();

LocalMap<String, String> map1 = sd.getLocalMap("mymap1");
```

- **Cluster-wide asynchronous maps**

```
sd.<String, String>getClusterWideMap("mymap", res -> {
  if (res.succeeded()) {
    AsyncMap<String, String> map = res.result();
  } else {
    // Something went wrong!
  }
});
```

- **Cluster-wide locks & counters (not shown)**

Networking



TCP Server

```
NetServer server = vertx.createNetServer();
```

```
NetServerOptions options = new NetServerOptions().setPort(4321);
NetServer server = vertx.createNetServer(options);
```

```
NetServer server = vertx.createNetServer();
server.listen();
```

```
server.listen(1234, "localhost", res -> {
  if (res.succeeded()) {
    System.out.println("Server is now listening!");
  } else {
    System.out.println("Failed to bind!");
  }
});
```

TCP Server Reading/Writing

```
NetServer server = vertx.createNetServer();
server.connectHandler(socket -> {
  socket.handler(buffer -> {
    System.out.println("I received some bytes: " + buffer.length());
  });
});
```

```
Buffer buffer = Buffer.buffer().appendFloat(12.34f).appendInt(123);
socket.write(buffer);

// Write a string in UTF-8 encoding
socket.write("some data");

// Write a string using the specified encoding
socket.write("some data", "UTF-16");
```

TCP Client

```
NetClientOptions options = new NetClientOptions().setConnectTimeout(10000);
NetClient client = vertx.createNetClient(options);
client.connect(4321, "localhost", res -> {
  if (res.succeeded()) {
    System.out.println("Connected!");
    NetSocket socket = res.result();
  } else {
    System.out.println("Failed to connect: " + res.cause().getMessage());
  }
});
```

HTTP Server

```
HttpServer server = vertx.createHttpServer();
server.listen(8080, "myhost.com", res -> {
  if (res.succeeded()) {
    System.out.println("Server is now listening!");
  } else {
    System.out.println("Failed to bind!");
  }
});
```

```
vertx.createHttpServer().requestHandler(request -> {
  request.response().end("Hello world");
}).listen(8080);
```

HTTP Client

You create an `HttpClient` instance with default options as follows:

```
HttpClient client = vertx.createHttpClient();
```

If you want to configure options for the client, you create it as follows:

```
HttpClientOptions options = new HttpClientOptions().setKeepAlive(false);
HttpClient client = vertx.createHttpClient(options);
```

Vertx Web

Provides more flexible support for writing web services.

```
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-web</artifactId>
  <version>3.4.1</version>
</dependency>
```

Vertx Web – Basic Routing

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

Router router = Router.router(vertx);

router.route().handler(routingContext -> {

  // This handler will be called for every request
  HttpServerResponse response = routingContext.response();
  response.putHeader("content-type", "text/plain");

  // Write to the response and end it
  response.end("Hello World from Vert.x-Web!");
});

server.requestHandler(router::accept).listen(8080);
```

Vertx Web – Rest Endpoint

```
public class PersonServer extends AbstractVerticle {

    private static final String APPLICATION_JSON = "application/json";

    private HttpServer server;


    @Override
    public void start()throws Exception {
        super.start();
        server = vertx.createHttpServer();
        Router router = Router.router(vertx);
        Router restAPI = Router.router(vertx);
        restAPI.get().handler(this::list);
        restAPI.get("/:id").handler(this::get);
        restAPI.post().handler(this::persist);
        restAPI.delete("/:id").handler(this::delete);
        router.mountSubRouter("/resources/persons", restAPI);
        router.route("/*").handler(StaticHandler.create("webapp"));

        server.requestHandler(router::accept).listen(8080);
    }
```

Vertx Web – Rest Endpoint

```
private void persist(RoutingContext rc) {
    rc.request().bodyHandler(buff -> {
        vertx.eventBus().send(PersonRepository.STORE, buff.toString(),
                h -> {
                    rc.response()
                            .setStatusCode(HttpResponseStatus.CREATED.code())
                            .end();
                });
    });
}

private void get(RoutingContext rc) {
    vertx.eventBus().send(PersonRepository.GET, rc.get("id"), h -> {
        rc.response()
                .putHeader(HttpHeaderNames.CONTENT_TYPE, APPLICATION_JSON)
                .end(String.valueOf(h.result().body()));
    });
}
```

Vertx CircuitBreaker

```
CircuitBreaker breaker = CircuitBreaker.create("my-circuit-breaker", vertx,
    new CircuitBreakerOptions().setMaxFailures(5).setTimeout(2000)
);

breaker.executeWithFallback(
    future -> {
      vertx.createHttpClient().getNow(8080, "localhost", "/", response -> {
        if (response.statusCode() != 200) {
          future.fail("HTTP error");
        } else {
          response
              .exceptionHandler(future::fail)
              .bodyHandler(buffer -> {
                future.complete(buffer.toString());
              });
        }
      });
    }, v -> {
      // Executed when the circuit is opened
      return "Hello";
    })
    .setHandler(ar -> {
      // Do something with the result
    });
```

```
<dependency>
  <groupId>io.vertx</groupId>
  <artifactId>vertx-circuit-breaker</artifactId>
  <version>3.4.1</version>
</dependency>
```

Vertx ServiceDiscovery

```
<dependency>
<groupId>io.vertx</groupId>
<artifactId>vertx-service-discovery</artifactId>
<version>3.4.1</version>
</dependency>
```

A service provider can:

- publish a service record
- un-publish a published record
- update the status of a published service (down, out of service…)

A service consumer can:

- lookup services
- bind to a selected service (it gets a `ServiceReference`) and use it
- release the service once the consumer is done with it
- listen for arrival, departure and modification of services.

Vertx ServiceDiscovery – Provider Sample

```
// Record creation from a type
record = HttpEndpoint.createRecord("some-rest-api", "localhost", 8080, "/api");
discovery.publish(record, ar -> {
  if (ar.succeeded()) {
    // publication succeeded
    Record publishedRecord = ar.result();
  } else {
    // publication failed
  }
});
```

Vertx ServiceDiscovery – Consumer Sample

```
ServiceReference reference1 = discovery.getReference(record1);
ServiceReference reference2 = discovery.getReference(record2);

// Then, gets the service object, the returned type depends on the service type:
// For http endpoint:
HttpClient client = reference1.getAs(HttpClient.class);
// For message source
MessageConsumer consumer = reference2.getAs(MessageConsumer.class);

// When done with the service
reference1.release();
reference2.release();
```

Clustering & HA

Clustering

```
VertxOptions options = new VertxOptions();
Vertx.clusteredVertx(options, res -> {
  if (res.succeeded()) {
    Vertx vertx = res.result();
    EventBus eventBus = vertx.eventBus();
    System.out.println("We now have a clustered event bus: " + eventBus);
  } else {
    System.out.println("Failed: " + res.cause());
  }
});
```

You should also make sure you have a `ClusterManager` implementation on your classpath, for example the default `HazelcastClusterManager`.

Clustering on the command line

You can run Vert.x clustered on the command line with

```
vertx run my-verticle.js -cluster
```

High Availability

Verticles can be deployed with High Availability (HA) enabled. In that context, when a verticle is deployed on a vert.x instance that dies abruptly, the verticle is redeployed on another vert.x instance from the cluster.

To run an verticle with the high availability enabled, just append the `-ha` switch:

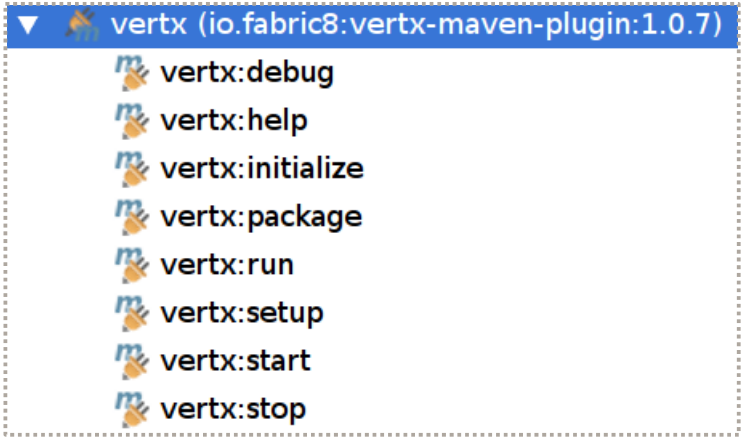
```
vertx run my-verticle.js -ha
```

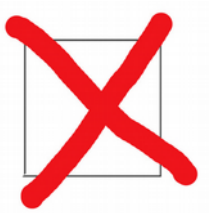
When enabling high availability, no need to add `-cluster`.

Vertx Maven Plugin

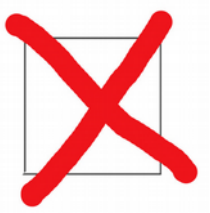
Vertx Maven Plugin

```
<properties>
    <java.main.class>io.vertx.core.Launcher</java.main.class>
    <vertx.verticle>gh.atsticks.samples.k8s.person.PersonApp</vertx.verticle>
</properties>

<!-- vert.x maven plugin to generate the fat-jar -->
<plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>vertx-maven-plugin</artifactId>
    <version>${vertx-maven-plugin.version}</version>
    <executions>
        <execution>
            <id>vmp</id>
            <phase>package</phase>
            <goals>
                <goal>initialize</goal>
                <goal>package</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```





I want to see a demo.



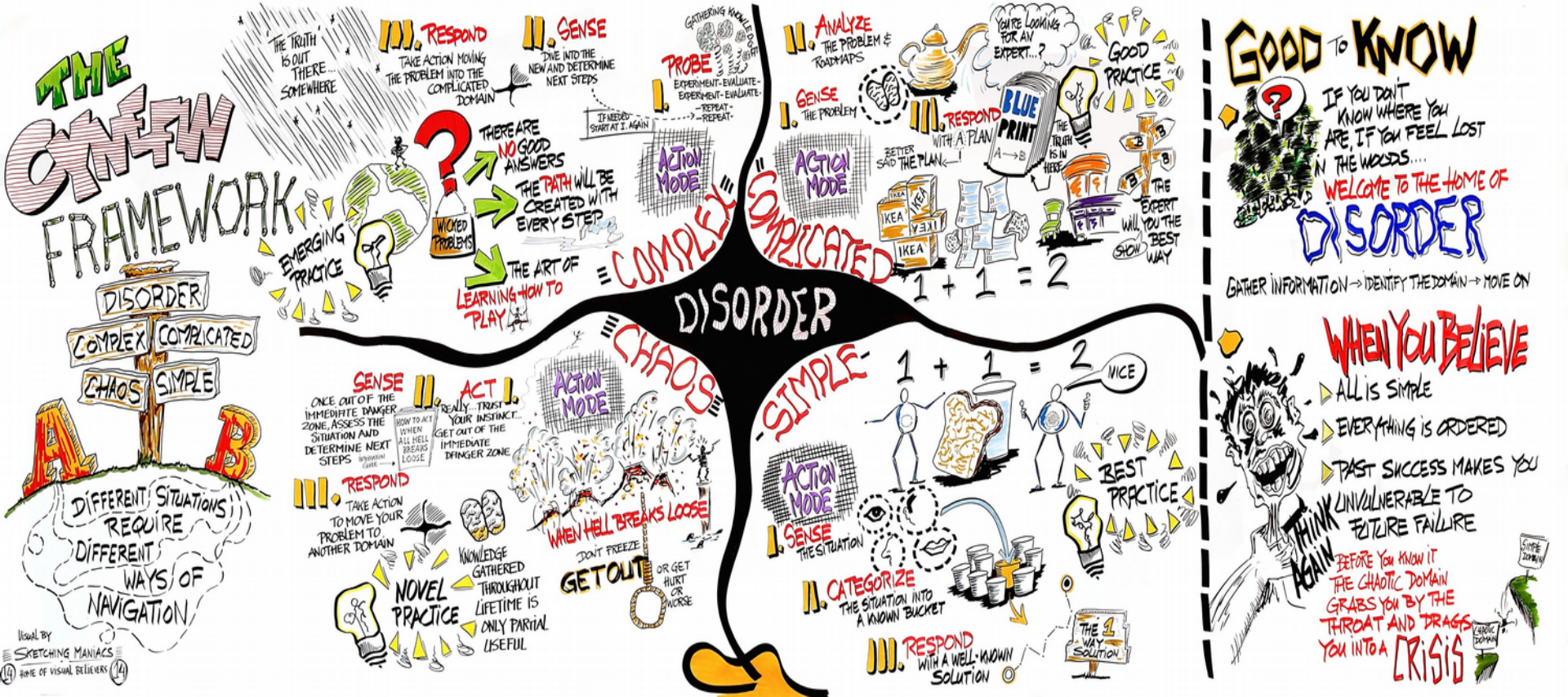
Please give me more.

Not discussed...

- Vertx CLI
- MQTT Client/Server
- Configuration
- Metrics Support
- Health-Checks
- Shell
- Test Support
- Service Proxies
- Openshift / Docker Tooling
- JDBC
- NoSql Support
- Advanced:
  - Code Generation, Language Bindings, etc

Demo Time ...

Recap


THE CYNEFIN FRAMEWORK
DISORDER
COMPLEX
COMPLICATED
CHAOS
SIMPLE
A
B
DIFFERENT SITUATIONS REQUIRE DIFFERENT WAYS OF NAVIGATION
Visual by SKETCHING MANIACS
HOME OF VISUAL BELIEVERS
THE TRUTH IS OUT THERE... SOMEWHERE
EMERGING PRACTICE
WICKED PROBLEMS
THERE ARE NO GOOD ANSWERS
THE PATH WILL BE CREATED WITH EVERY STEP
THE ART OF LEARNING HOW TO PLAY
= COMPLEX
RESPOND
TAKE ACTION MOVING THE PROBLEM INTO THE COMPLICATED DOMAIN
SENSE
DIVE INTO THE NEW AND DETERMINE NEXT STEPS
PROBE
EXPERIMENT - EVALUATE - EXPERIMENT - EVALUATE - REPEAT - REPEAT
IF NEEDED START AT 1. AGAIN
GATHERING KNOWLEDGE
ACTION MODE
ANALYZE
THE PROBLEM & ROADMAPS
SENSE
THE PROBLEM
RESPOND
WITH A PLAN
BETTER SAID THAN DONE
BLUE PRINT
A → B
THE TRUTH IS IN HERE
YOU'RE LOOKING FOR AN EXPERT...?
GOOD PRACTICE
THE EXPERT WILL SHOW YOU THE BEST WAY
IKEA
ACTION MODE
SIMPLE
1 + 1 = 2
SIMPLE
1 + 1 = 2
NICE
ACTION MODE
SENSE
THE SITUATION
BEST PRACTICE
CATEGORIZE
THE SITUATION INTO A KNOWN BUCKET
RESPOND
WITH A WELL-KNOWN SOLUTION
THE 1 WAY SOLUTION
CHAOS
SENSE
ONCE OUT OF THE IMMEDIATE DANGER ZONE, ASSESS THE SITUATION AND DETERMINE NEXT STEPS
HOW TO ACT WHEN ALL HELL BREAKS LOOSE
ACT
REALLY... TRUST YOUR INSTINCT... GET OUT OF THE IMMEDIATE DANGER ZONE
RESPOND
TAKE ACTION TO MOVE YOUR PROBLEM TO ANOTHER DOMAIN
ACTION MODE
WHEN HELL BREAKS LOOSE
DON'T FREEZE
GET OUT
OR GET HURT OR WORSE
NOVEL PRACTICE
KNOWLEDGE GATHERED THROUGHOUT LIFETIME IS ONLY PARTIAL USEFUL
GOOD TO KNOW
IF YOU DON'T KNOW WHERE YOU ARE IF YOU FEEL LOST IN THE WOODS....
WELCOME TO THE HOME OF
DISORDER
GATHER INFORMATION → IDENTIFY THE DOMAIN → MOVE ON
WHEN YOU BELIEVE
ALL IS SIMPLE
EVERYTHING IS ORDERED
PAST SUCCESS MAKES YOU UNVULNERABLE TO FUTURE FAILURE
THINK AGAIN
BEFORE YOU KNOW IT THE CHAOTIC DOMAIN GRABS YOU BY THE THROAT AND DRAGS YOU INTO A CRISIS
SIMPLE DOMAIN
CHAOTIC DOMAIN

Recap

- Vertx is a very flexible and modular toolkit
- It is rather leightweight (ca. 6 MB for something useful)
- It is polyglot, supporting a big range of languages
- It is amazing fast
- It supports all requirements of the reactive manifesto
- It sometimes can be a beast
- It requires Java 8+
- It is fun to work with

A Hitchhikers Introduction to Vertx

Anatole Tresch
Principal Consultant

Tel. +41 58 459 53 93
anatole.tresch@trivadis.com

Vertx: http://vertx.io

Vertx Maven Plugin: https://vmp.fabric8.io/

Reactive Manifesto: http://www.reactivemanifesto.org/