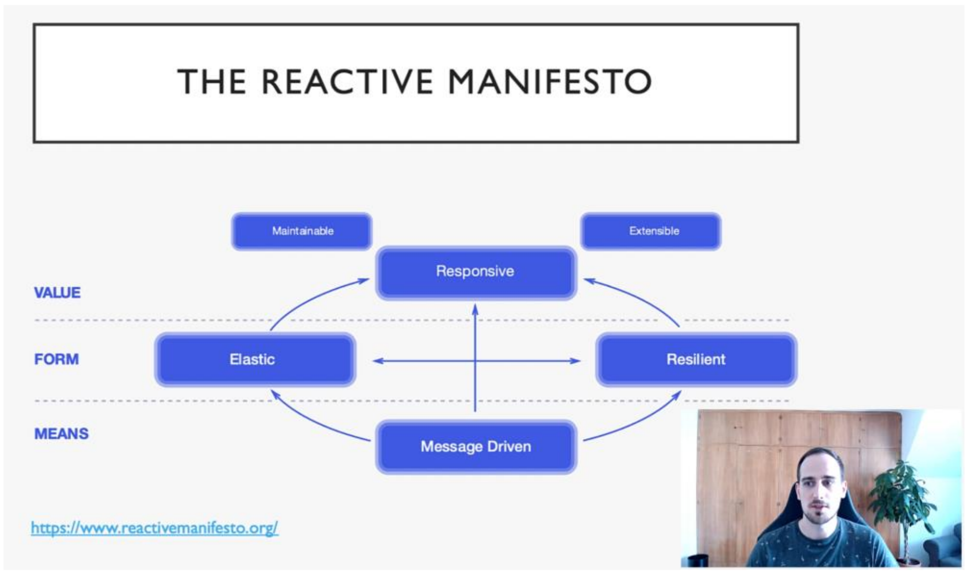
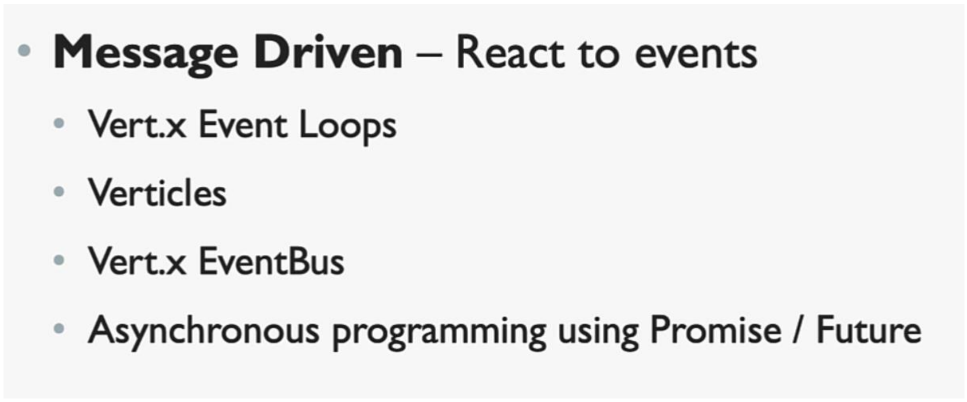
Vert.x

Code: https://github.com/danielprinz/vertx-udemy

HTTPie: https://httpie.io/desktop (AppImageLauncher HTTPie-2023.1.2.AppImage

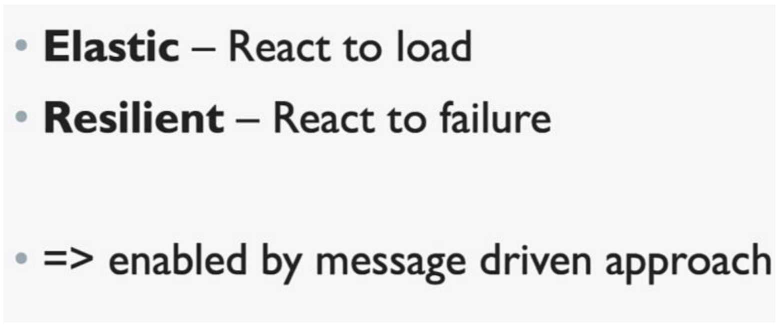
Vert.x is a toolkit to create reactive applications on the JVM.





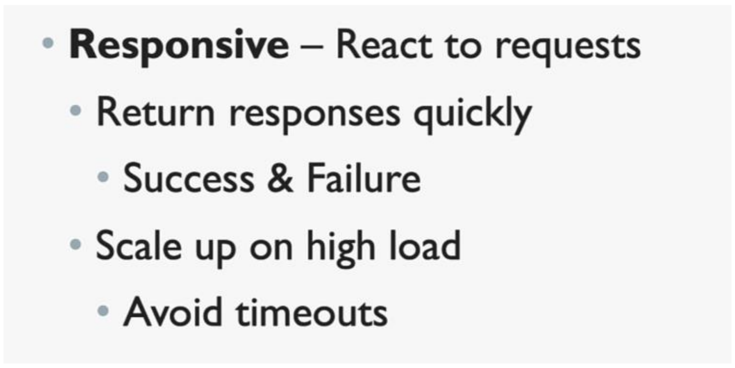
- **Message Driven** – React to events
  - Vert.x Event Loops
  - Verticles
  - Vert.x EventBus
  - Asynchronous programming using Promise / Future

Verticles can communicate each other over the **Vert.x EventBus** with messages. *Asynchronous programming* enabled through the use **Promise / Future**.

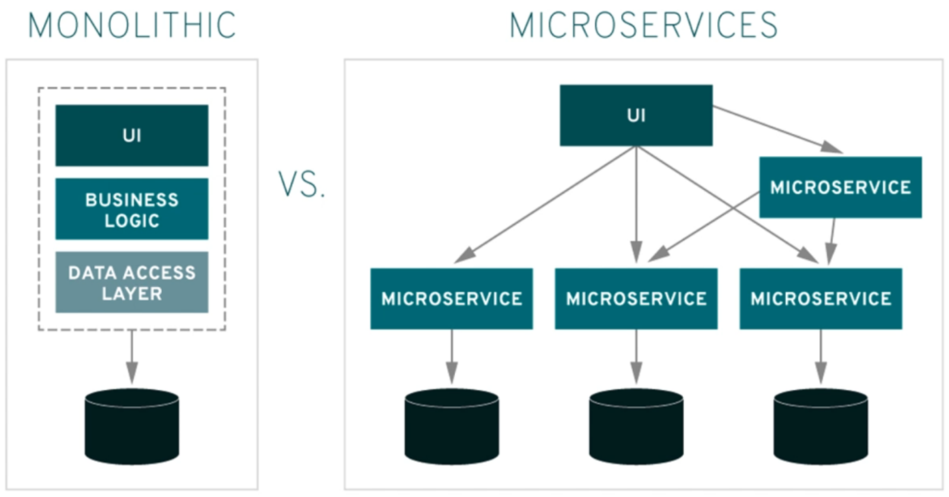


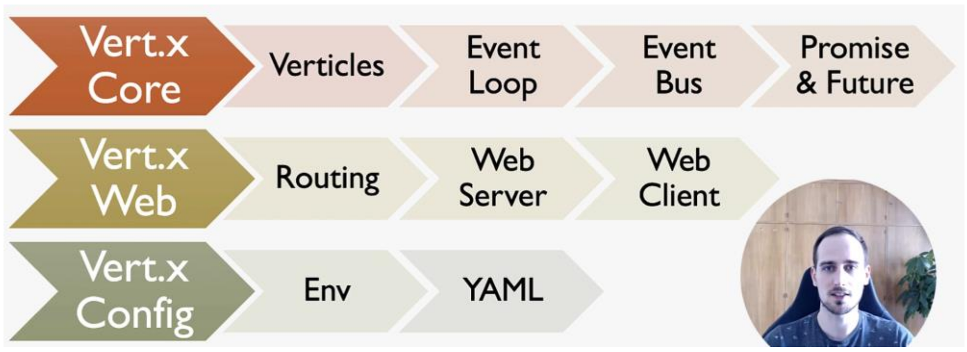
- **Elastic** – React to load
- **Resilient** – React to failure

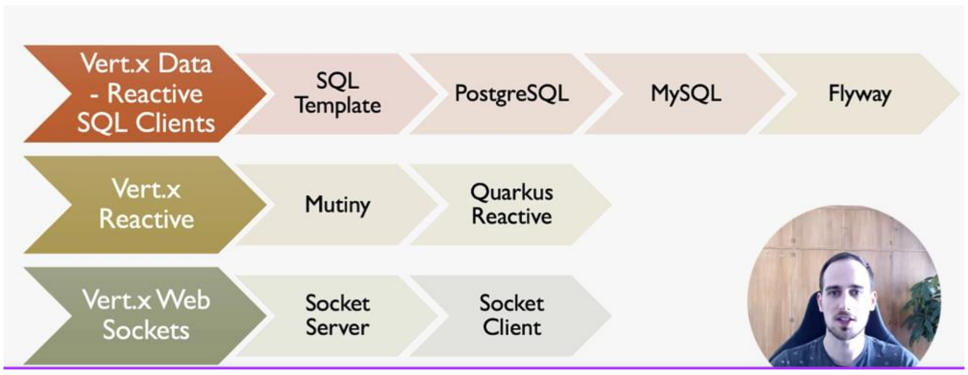
- => enabled by message driven approach



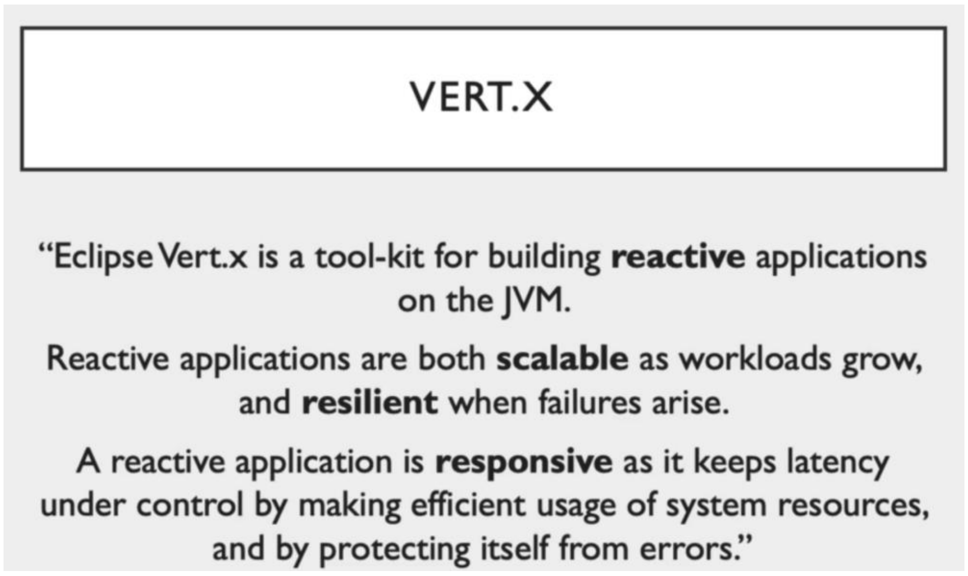
- **Responsive** – React to requests
  - Return responses quickly
    - Success & Failure
  - Scale up on high load
    - Avoid timeouts







Vert.x Core

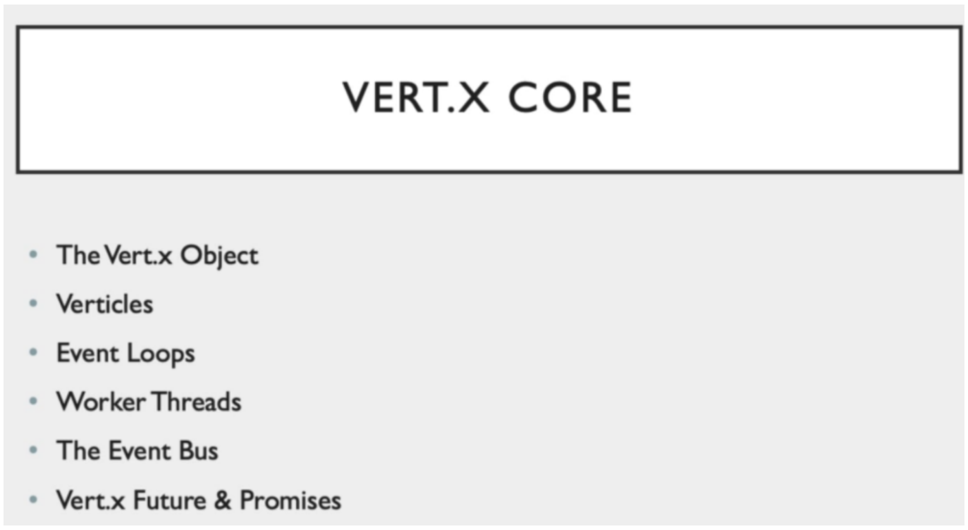


VERT.X

"Eclipse Vert.x is a tool-kit for building **reactive** applications on the JVM.

Reactive applications are both **scalable** as workloads grow, and **resilient** when failures arise.

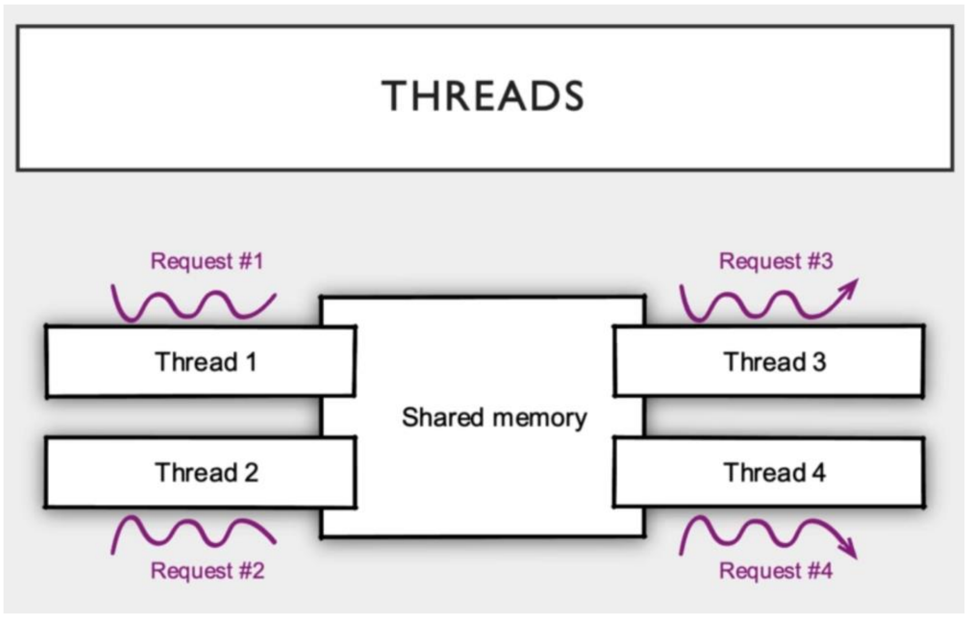
A reactive application is **responsive** as it keeps latency under control by making efficient usage of system resources, and by protecting itself from errors."

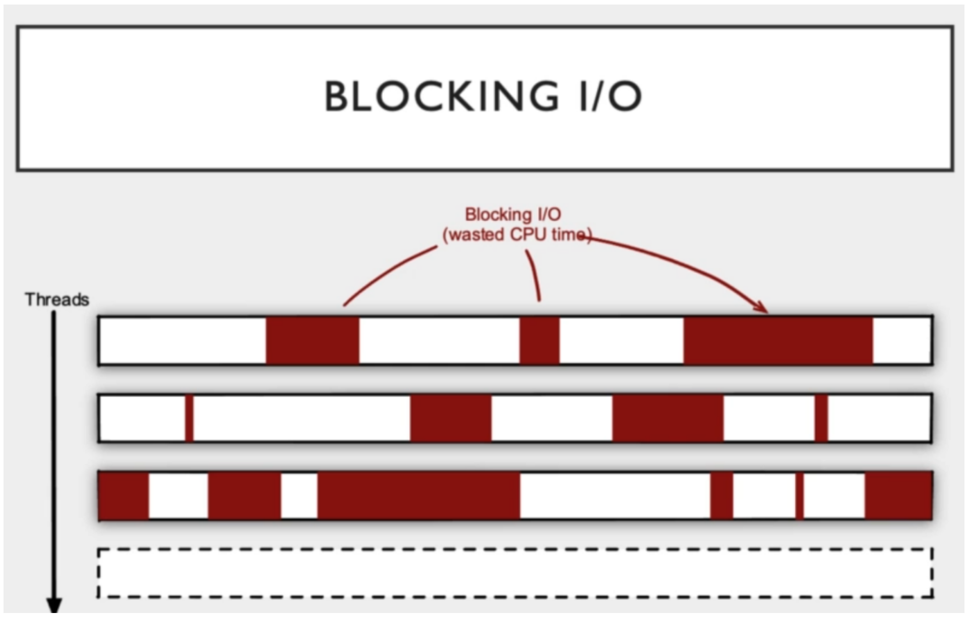


VERT.X CORE

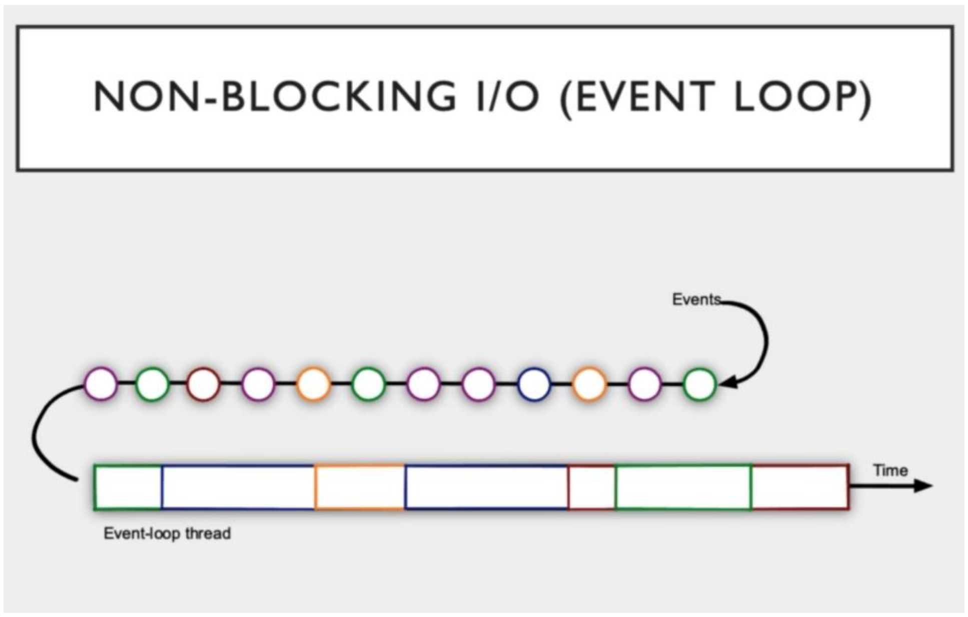
- The Vert.x Object
- Verticles
- Event Loops
- Worker Threads
- The Event Bus
- Vert.x Future & Promises

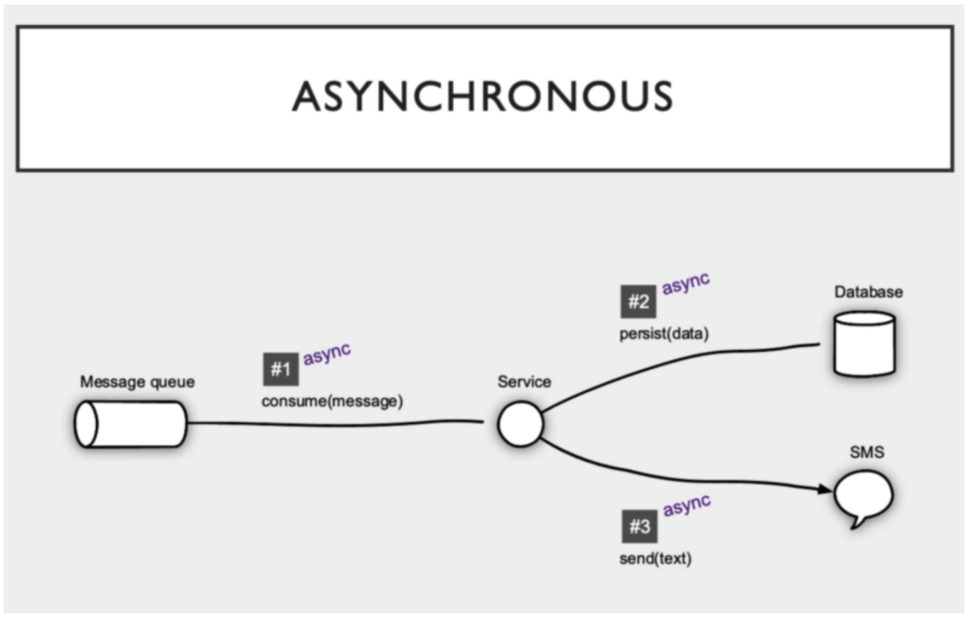
In a Java traditional application multiple threads are used to manage requests.



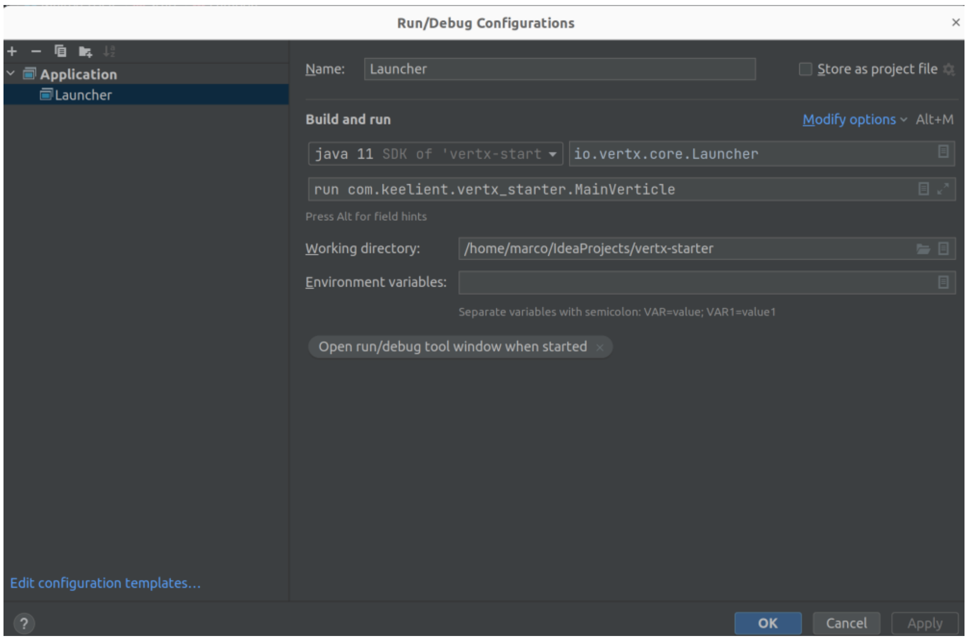


A Vert.x application.


NON-BLOCKING I/O (EVENT LOOP)


ASYNCHRONOUS

Running a Verticle in IntelliJ



Run/Debug Configurations

- Application
  - Launcher

Name: Launcher

☐ Store as project file

Build and run

Modify options Alt+M

```
java 11 SDK of 'vertx-start    io.vertx.core.Launcher
run com.keelient.vertx_starter.MainVerticle
```

Press Alt for field hints

Working directory: /home/marco/IdeaProjects/vertx-starter

Environment variables:

Separate variables with semicolon: VAR=value; VAR1=value1

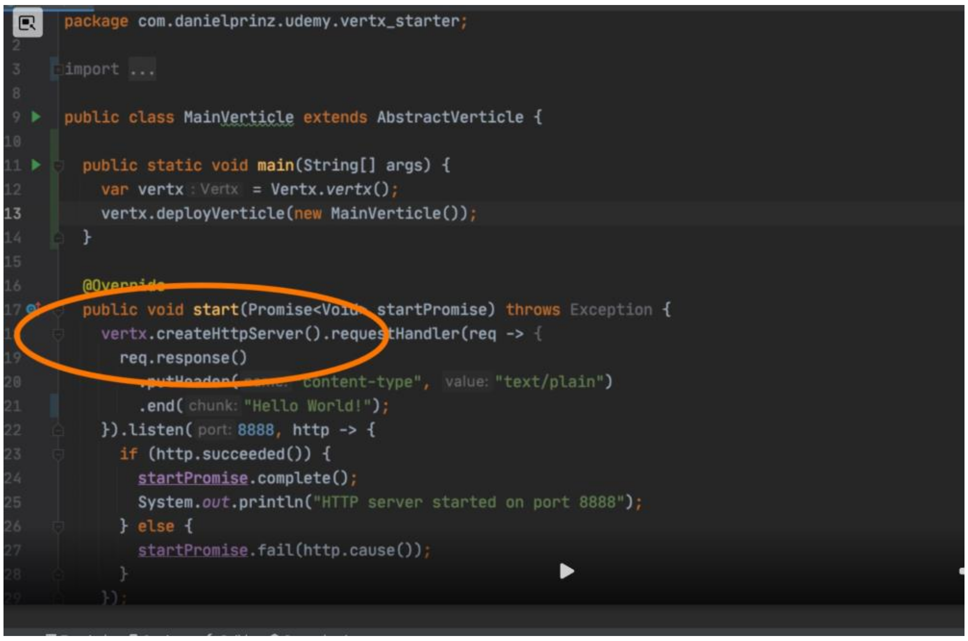
Open run/debug tool window when started

Edit configuration templates…

OK Cancel Apply

The Vert.x object

With Vert.x object servers can be created. The java main class containing the Verticle is the entrypoint of the application.



```java
package com.danielprinz.udemy.vertx_starter;

import ...

public class MainVerticle extends AbstractVerticle {

  public static void main(String[] args) {
    var vertx : Vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }

  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    vertx.createHttpServer().requestHandler(req -> {
      req.response()
        .putHeader( name: "content-type", value: "text/plain")
        .end( chunk: "Hello World!");
    }).listen( port: 8888, http -> {
      if (http.succeeded()) {
        startPromise.complete();
        System.out.println("HTTP server started on port 8888");
      } else {
        startPromise.fail(http.cause());
      }
    });
```

Declaring the Vert.x object in the main method let us deploy the Verticle MainVerticle.

The deployVerticle method calls start.

Vert.x Verticles

Verticle runs in the event-loop thread.

L'event loop di vert.x è un meccanismo di concurrency reattivo basato su eventi.

Funziona secondo il modello event-loop, il che significa che ha un thread dedicato (l'event loop thread) che gestisce continuamente gli eventi in arrivo.

Quando una richiesta arriva (ad esempio una richiesta HTTP), non viene creato un nuovo thread per gestirla, ma piuttosto viene aggiunta a una coda di eventi e gestita dall'event loop thread quando è disponibile.

Multi-Reactor design patter (event-loop)

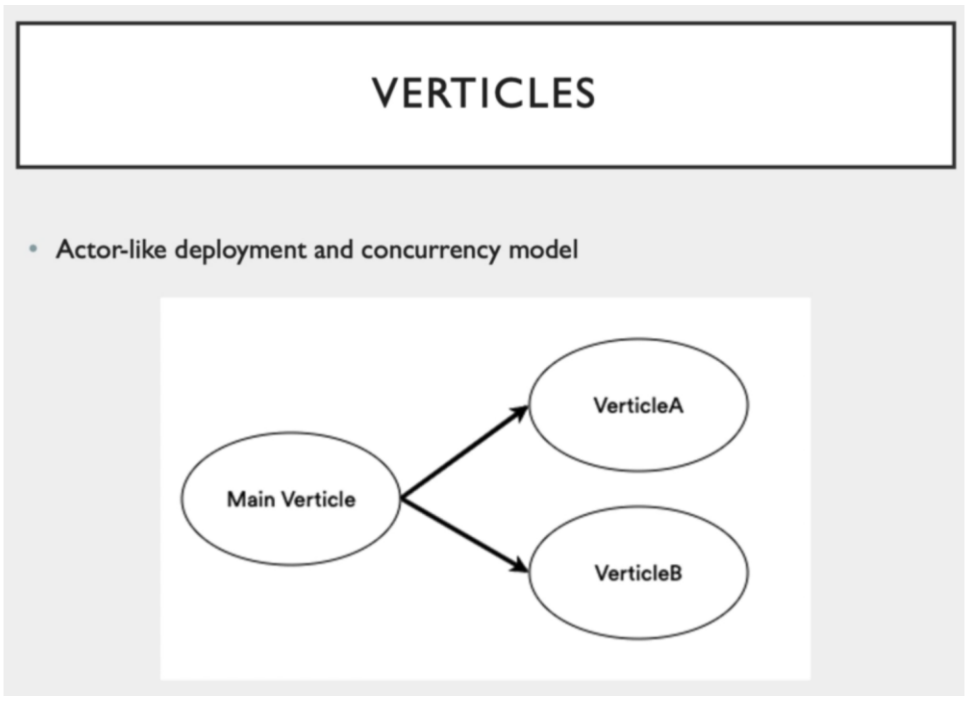
Event Loop is an implementation of **Multi-Reactor design patter.**

It’s goal is to continuously check for new events, and each time a new event comes in, to quickly dispatch it to someone who knows how to handle it.

Maximum number of Event Loop threads depends on number of CPUs, not on number of verticles deployed. The number of event-loop threads is determined by Vert.x.

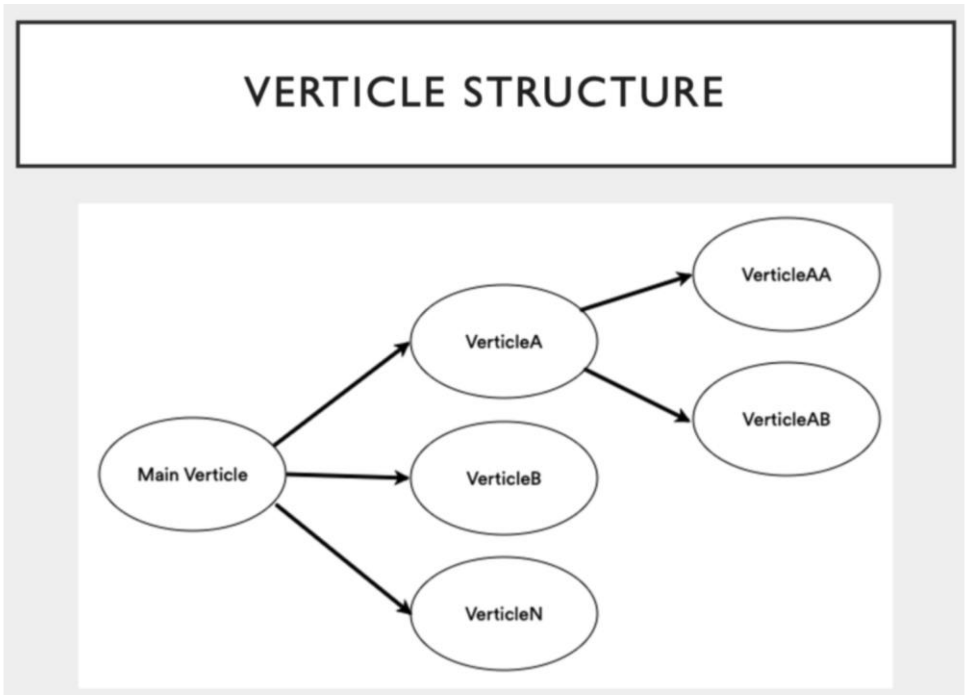
Vert.x calcola il numero ottimale di event loop thread in base al numero di core della CPU.
In particolare 2 per core.

Con un processore quad-core, il numero massimo di istanze di Verticle che puoi eseguire in parallelo è 8. Questo perché **Vert.x esegue un Verticle per event loop**, e puoi avere al massimo tanti event loop quanti sono i core del processore x 2.



VERTICLES

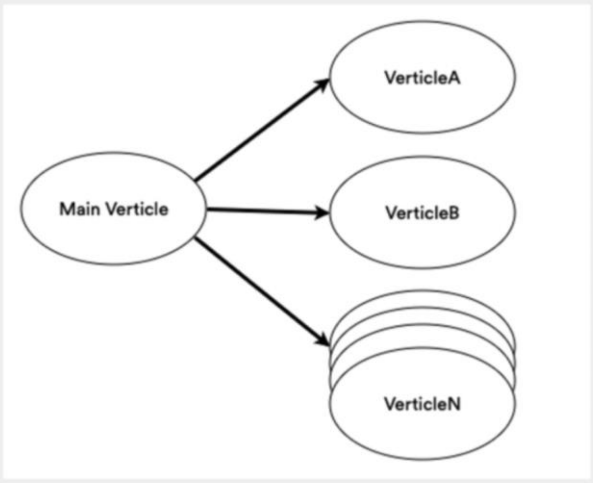
- Actor-like deployment and concurrency model



VERTICLE STRUCTURE

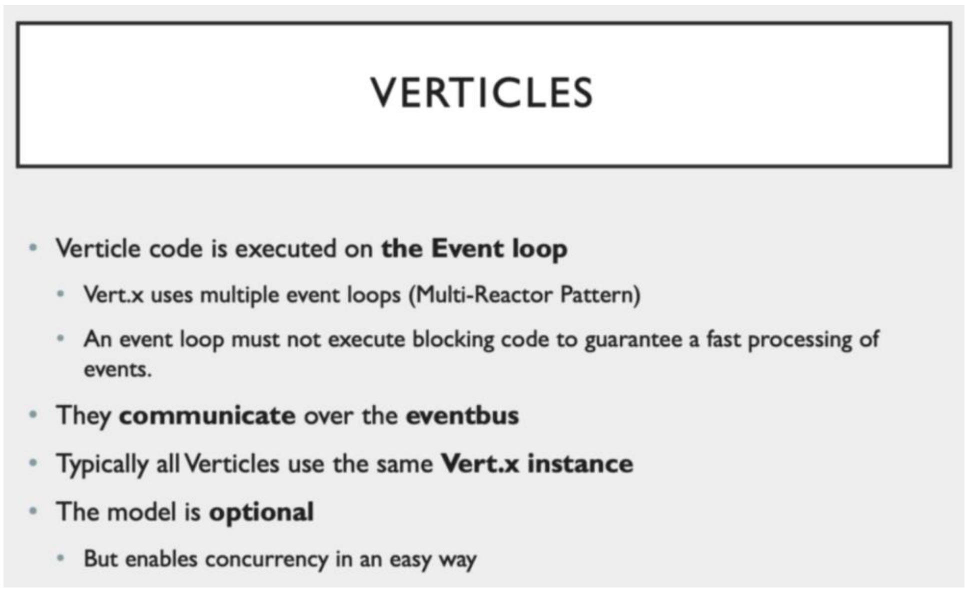
Each Verticle can be deployed multiple time. Each Verticle will have its own thread.

SCALING VERTICLES



VERTICLES

- Verticle code is executed on **the Event loop**
  - Vert.x uses multiple event loops (Multi-Reactor Pattern)
  - An event loop must not execute blocking code to guarantee a fast processing of events.

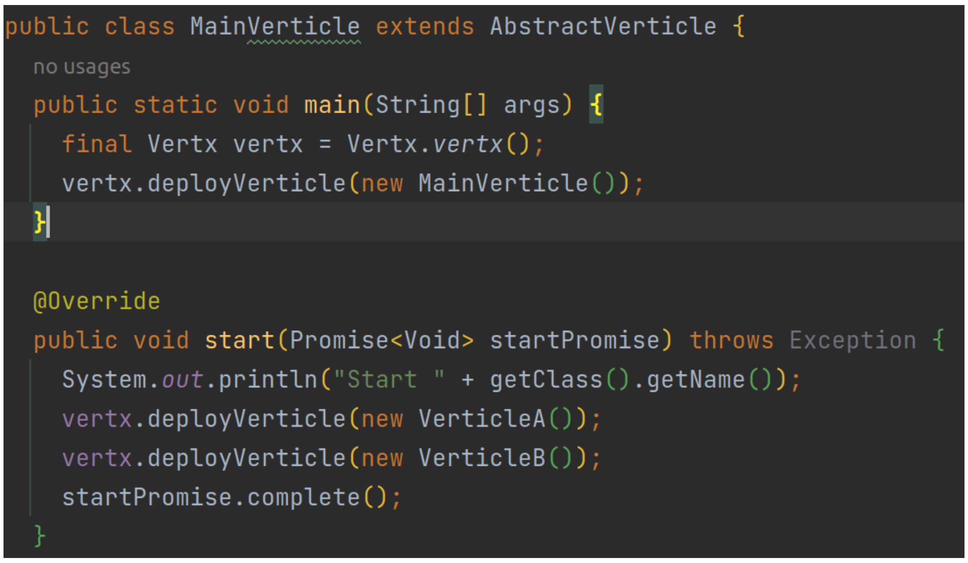


VERTICLES

- Verticle code is executed on **the Event loop**
  - Vert.x uses multiple event loops (Multi-Reactor Pattern)
  - An event loop must not execute blocking code to guarantee a fast processing of events.
- They **communicate** over the **eventbus**
- Typically all Verticles use the same **Vert.x instance**
- The model is **optional**
  - But enables concurrency in an easy way

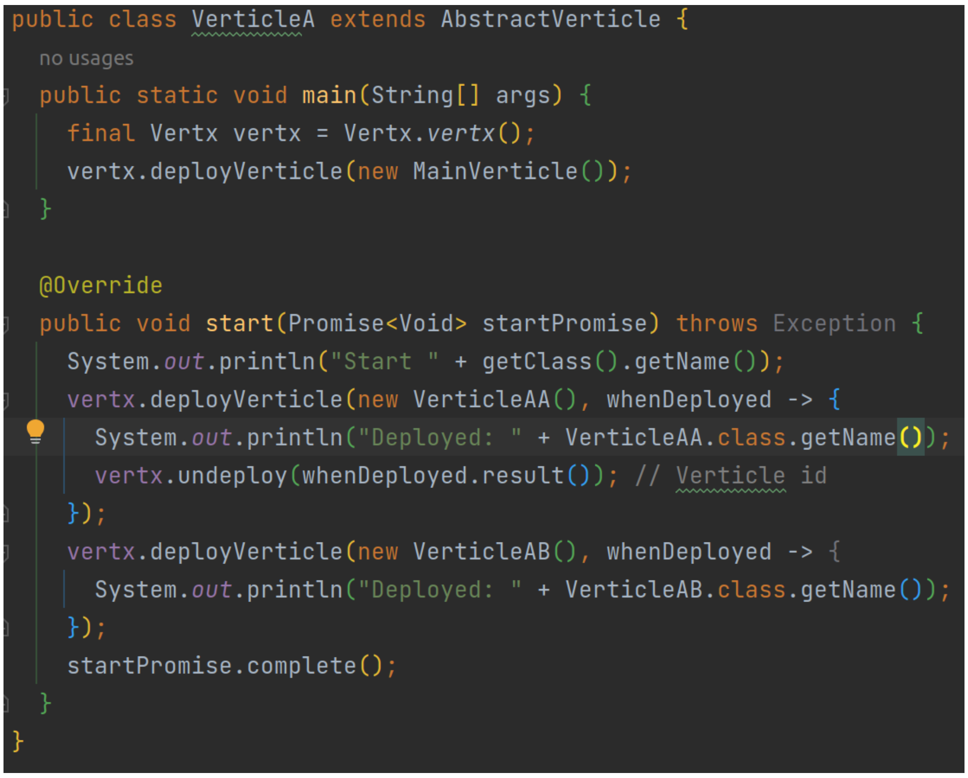
Il metodo .complete() su Promise<Void> in Vert.x serve per segnalare il completamento di un'operazione asincrona che non produce un valore.

Deploying verticles



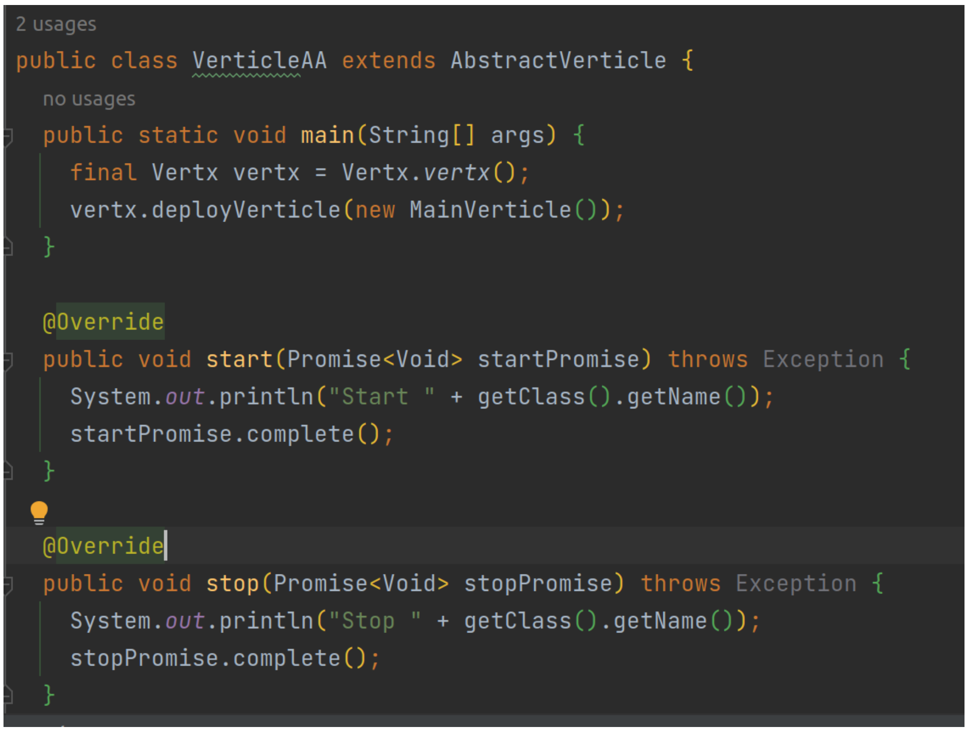
```
public class MainVerticle extends AbstractVerticle {
  no usages
  public static void main(String[] args) {
    final Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }

  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    System.out.println("Start " + getClass().getName());
    vertx.deployVerticle(new VerticleA());
    vertx.deployVerticle(new VerticleB());
    startPromise.complete();
  }
```

Undeploying verticles



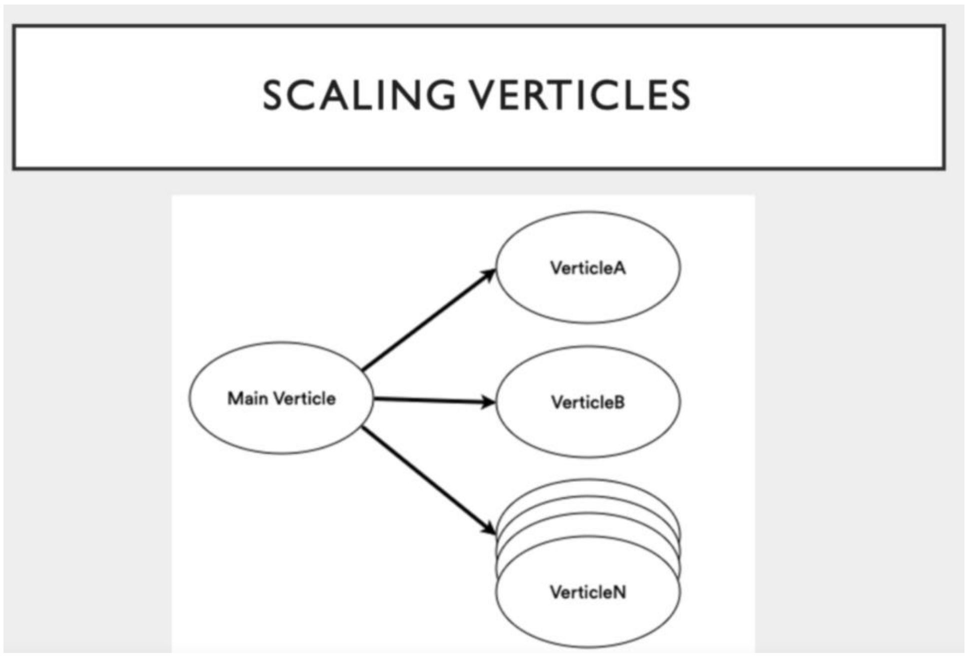
```
public class VerticleA extends AbstractVerticle {
  no usages
  public static void main(String[] args) {
    final Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }

  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    System.out.println("Start " + getClass().getName());
    vertx.deployVerticle(new VerticleAA(), whenDeployed -> {
      System.out.println("Deployed: " + VerticleAA.class.getName());
      vertx.undeploy(whenDeployed.result()); // Verticle id
    });
    vertx.deployVerticle(new VerticleAB(), whenDeployed -> {
      System.out.println("Deployed: " + VerticleAB.class.getName());
    });
    startPromise.complete();
  }
}
```

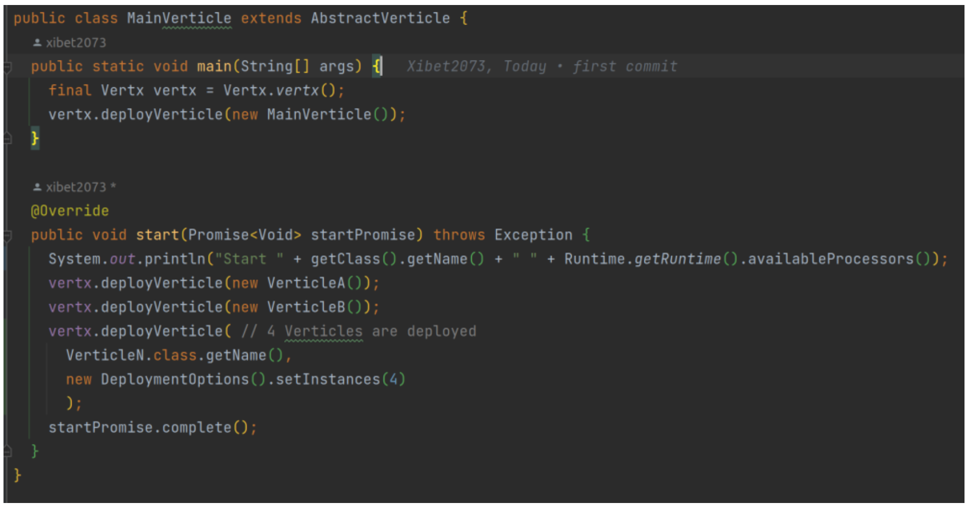
Undeploy method calls stop method in Verticle class.



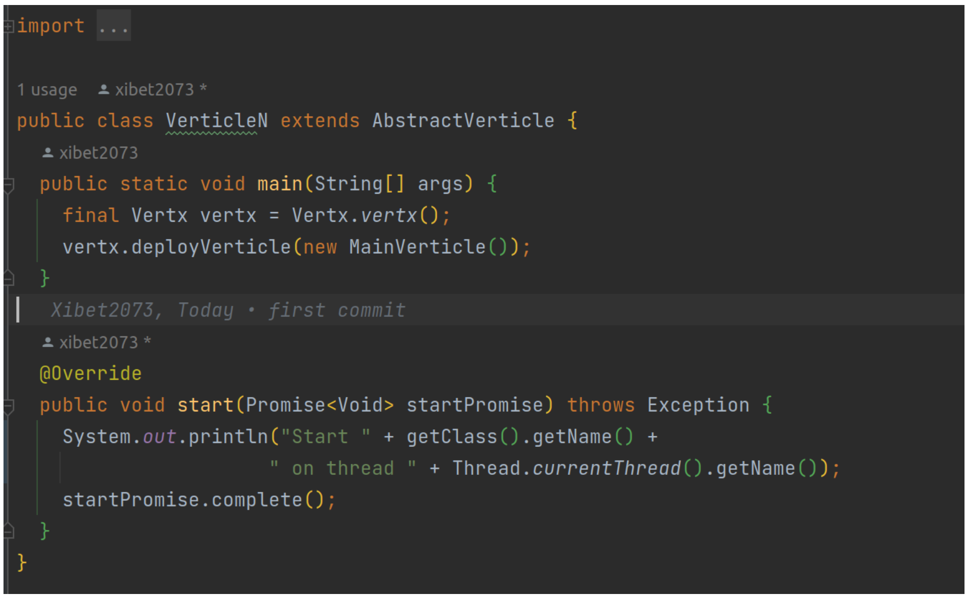
```
2 usages
public class VerticleAA extends AbstractVerticle {
  no usages
  public static void main(String[] args) {
    final Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }

  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    System.out.println("Start " + getClass().getName());
    startPromise.complete();
  }

  @Override
  public void stop(Promise<Void> stopPromise) throws Exception {
    System.out.println("Stop " + getClass().getName());
    stopPromise.complete();
  }
```

Scaling Verticles



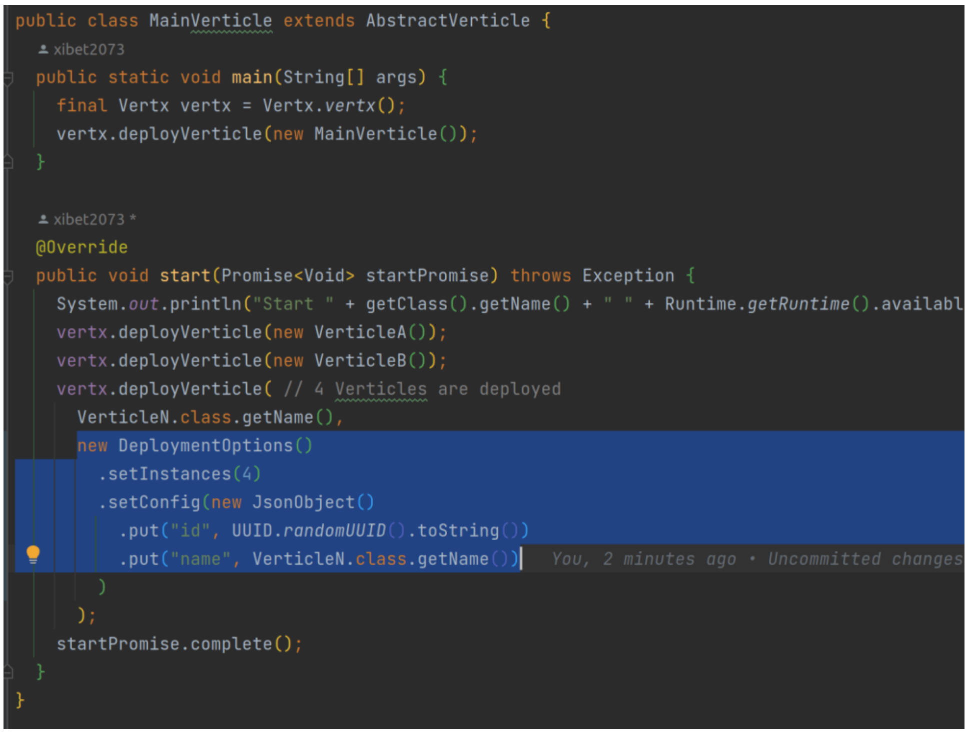


```
public class MainVerticle extends AbstractVerticle {
  xibet2073
  public static void main(String[] args) {   Xibet2073, Today • first commit
    final Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }

  xibet2073 *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    System.out.println("Start " + getClass().getName() + " " + Runtime.getRuntime().availableProcessors());
    vertx.deployVerticle(new VerticleA());
    vertx.deployVerticle(new VerticleB());
    vertx.deployVerticle( // 4 Verticles are deployed
      VerticleN.class.getName(),
      new DeploymentOptions().setInstances(4)
      );
    startPromise.complete();
  }
}
```

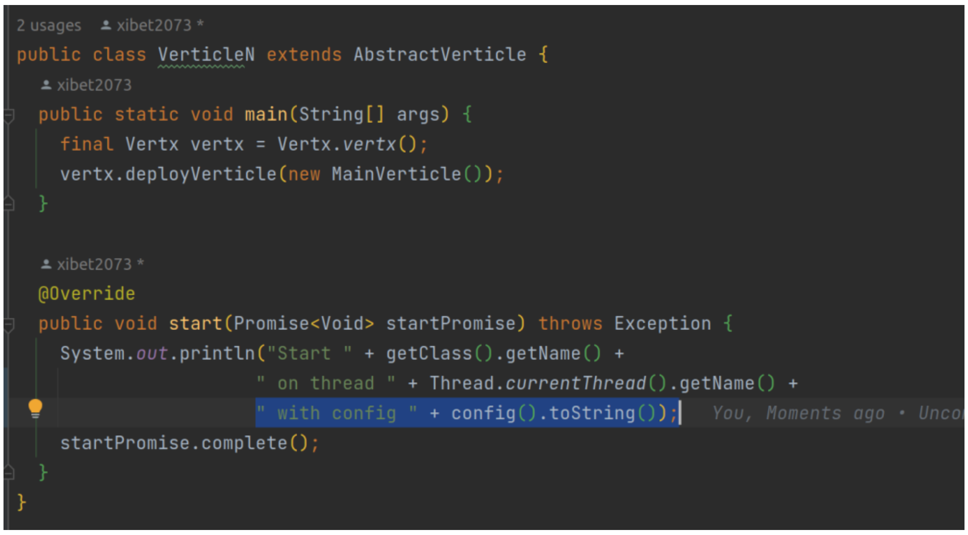


```
import ...

1 usage   xibet2073 *
public class VerticleN extends AbstractVerticle {
   xibet2073
  public static void main(String[] args) {
    final Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }
    Xibet2073, Today • first commit
   xibet2073 *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    System.out.println("Start " + getClass().getName() +
                       " on thread " + Thread.currentThread().getName());
    startPromise.complete();
  }
}
```

Verticle Config



```
public class MainVerticle extends AbstractVerticle {
   xibet2073
  public static void main(String[] args) {
    final Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }

   xibet2073 *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    System.out.println("Start " + getClass().getName() + " " + Runtime.getRuntime().availabl
    vertx.deployVerticle(new VerticleA());
    vertx.deployVerticle(new VerticleB());
    vertx.deployVerticle( // 4 Verticles are deployed
      VerticleN.class.getName(),
      new DeploymentOptions()
        .setInstances(4)
        .setConfig(new JsonObject()
          .put("id", UUID.randomUUID().toString())
          .put("name", VerticleN.class.getName())    You, 2 minutes ago • Uncommitted changes
        )
      );
    startPromise.complete();
  }
}
```



```
2 usages  xibet2073 *
public class VerticleN extends AbstractVerticle {
   xibet2073
  public static void main(String[] args) {
    final Vertx vertx = Vertx.vertx();
    vertx.deployVerticle(new MainVerticle());
  }

   xibet2073 *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    System.out.println("Start " + getClass().getName() +
                       " on thread " + Thread.currentThread().getName() +
                       " with config " + config().toString());   You, Moments ago • Unco
    startPromise.complete();
  }
}
```

Each instance of VerticleN has the same id meaning that each Verticle has the same configuration.

Logging

I logging asincrono in Vert.x funziona nel modo seguente:

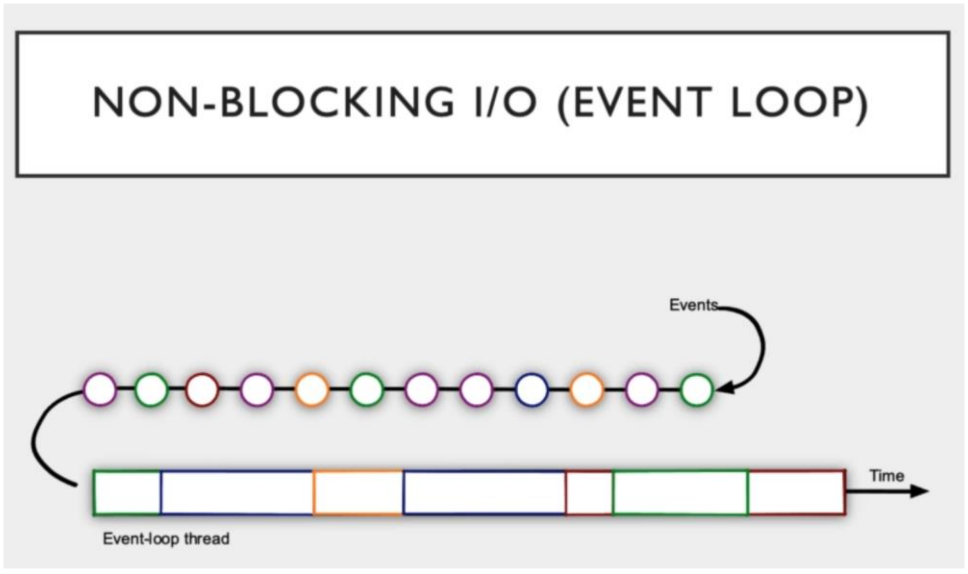
I log vengono scritti in buffer in memoria anziché essere scritti direttamente.

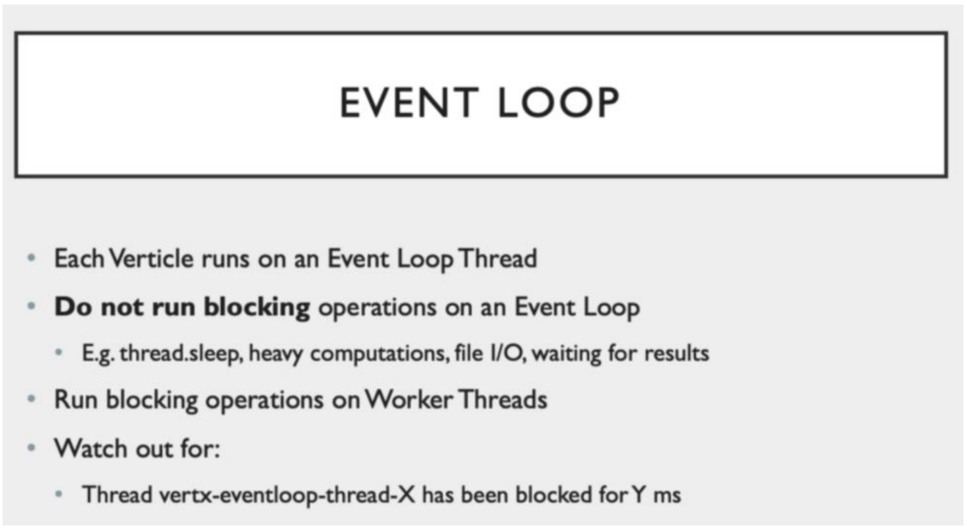
Un numero impostabile di thread di scrittura dedicati estrarrà i log dal buffer e li scriverà effettivamente alle appenders configurate (console, file, ecc.)

I vantaggi di questo approccio sono:

Le chiamate ai logger non bloccano gli event loop di Vert.x, mantenendo le operazioni asincrone rapide. I log vengono comunque scritti alle appenders in ordine cronologico corretto.

Event loop threads

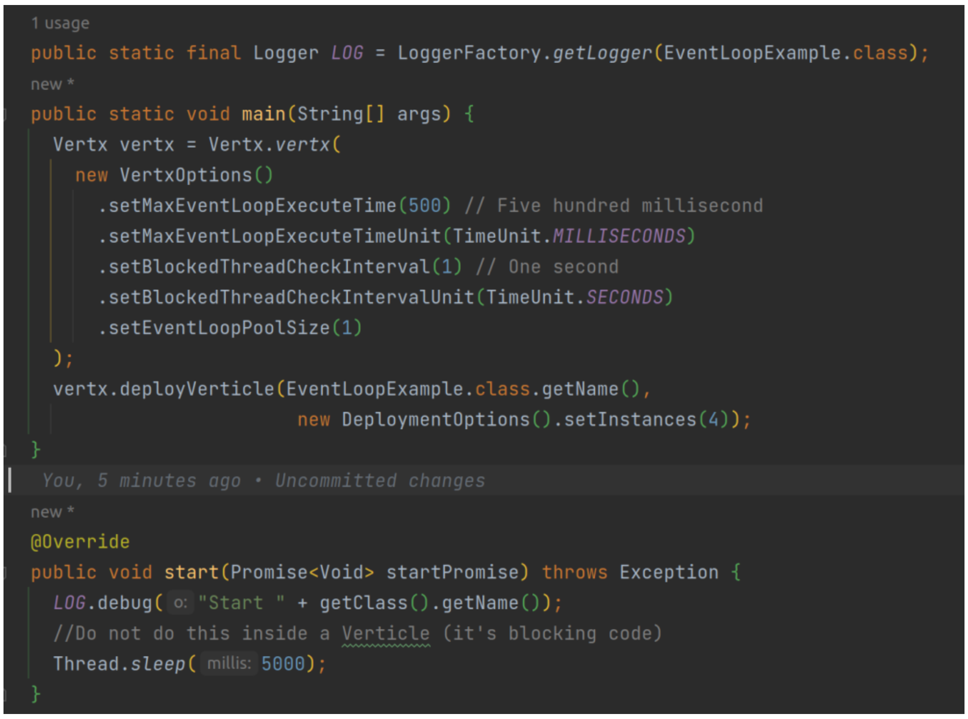




EVENT LOOP

- Each Verticle runs on an Event Loop Thread
- **Do not run blocking** operations on an Event Loop
  - E.g. thread.sleep, heavy computations, file I/O, waiting for results
- Run blocking operations on Worker Threads
- Watch out for:
  - Thread vertx-eventloop-thread-X has been blocked for Y ms

Un thread di di Vert.x verifica <u>ogni secondo</u> (intervallo modificabile) se l'**event-loop è bloccato** (controllando il l'execution time [<u>default 2 secondi</u>]). Se rileva un blocco, produce un warning su console.

- 

```
1 usage
public static final Logger LOG = LoggerFactory.getLogger(EventLoopExample.class);
new *
public static void main(String[] args) {
  Vertx vertx = Vertx.vertx(
    new VertxOptions()
      .setMaxEventLoopExecuteTime(500) // Five hundred millisecond
      .setMaxEventLoopExecuteTimeUnit(TimeUnit.MILLISECONDS)
      .setBlockedThreadCheckInterval(1) // One second
      .setBlockedThreadCheckIntervalUnit(TimeUnit.SECONDS)
      .setEventLoopPoolSize(1)
  );
  vertx.deployVerticle(EventLoopExample.class.getName(),
                       new DeploymentOptions().setInstances(4));
}
  You, 5 minutes ago • Uncommitted changes
new *
@Override
public void start(Promise<Void> startPromise) throws Exception {
  LOG.debug( o: "Start " + getClass().getName());
  //Do not do this inside a Verticle (it's blocking code)
  Thread.sleep( millis: 5000);
}
```

Se il tempo di esecuzione di un Verticle è superiore a quello massimo dell'event-loop, viene comunque eseguito ma verrà mostrato un warning che segnala che l'event-loop è rimasto bloccato in attesa che il Verticle terminasse la sua attività.

Worker threads

Vert.x provides a worker thread pool to execute blocking operations.

There are two ways to execute blocking operations:

- by calling vertx.executeBlocking

```
private Handler<Message<String>> allProductsHandler(ProductService service) {
  // It is important to use an executeBlocking construct here
  // as the service calls are blocking (dealing with a database)
  return msg -> vertx.<String>executeBlocking(future -> {
      try {
       future.complete(mapper.writeValueAsString(service.getAllProducts()));
      } catch (JsonProcessingException e) {
       System.out.println("Failed to serialize result");
       future.fail(e);
      }
    },
    result -> {
      if (result.succeeded()) {
       msg.reply(result.result());
      } else {
       msg.reply(result.cause().toString());
      }
    });
}
```

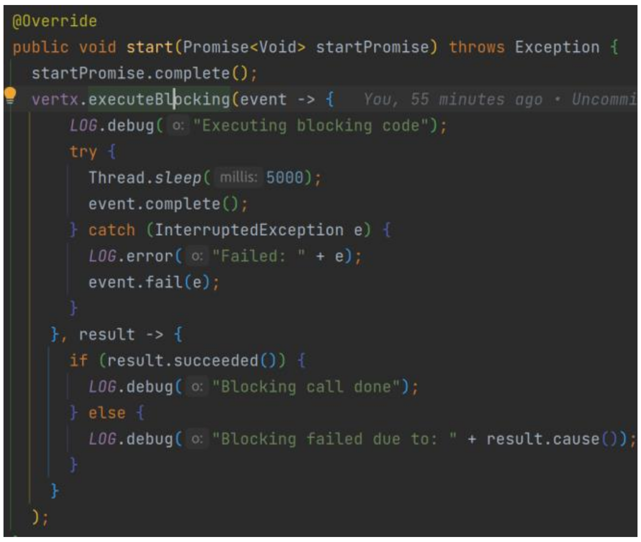
- By deploying a Worker Verticle

Decoupling - I worker vengono isolati in thread separati, quindi un worker lento o bloccato non influisce sugli altri worker o sul verticle principale.

Scalabilità - Puoi eseguire più istanze di un worker per gestire più carico di lavoro in parallelo.

Modularità - I worker possono essere progettati e implementati in modo indipendente per svolgere compiti specifici.

executeBlocking block

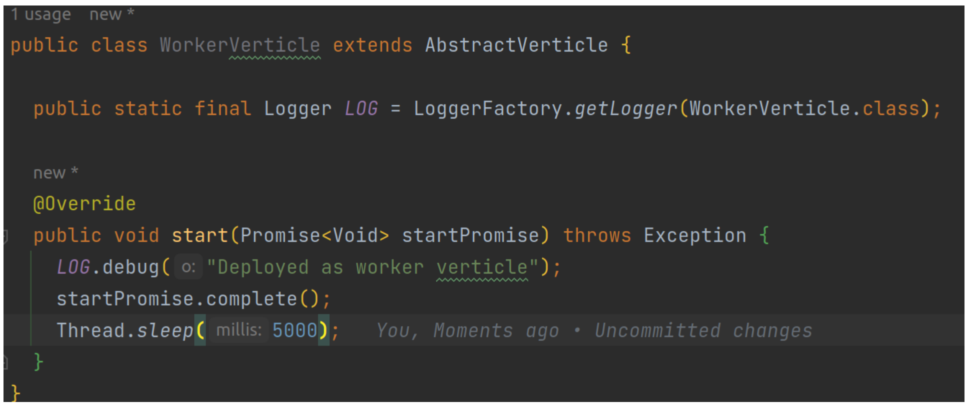


```
@Override
public void start(Promise<Void> startPromise) throws Exception {
  startPromise.complete();
  vertx.executeBlocking(event -> {   You, 55 minutes ago • Uncommi
      LOG.debug( o: "Executing blocking code");
      try {
        Thread.sleep( millis: 5000);
        event.complete();
      } catch (InterruptedException e) {
        LOG.error( o: "Failed: " + e);
        event.fail(e);
      }
    }, result -> {
      if (result.succeeded()) {
        LOG.debug( o: "Blocking call done");
      } else {
        LOG.debug( o: "Blocking failed due to: " + result.cause());
      }
    }
  );
```

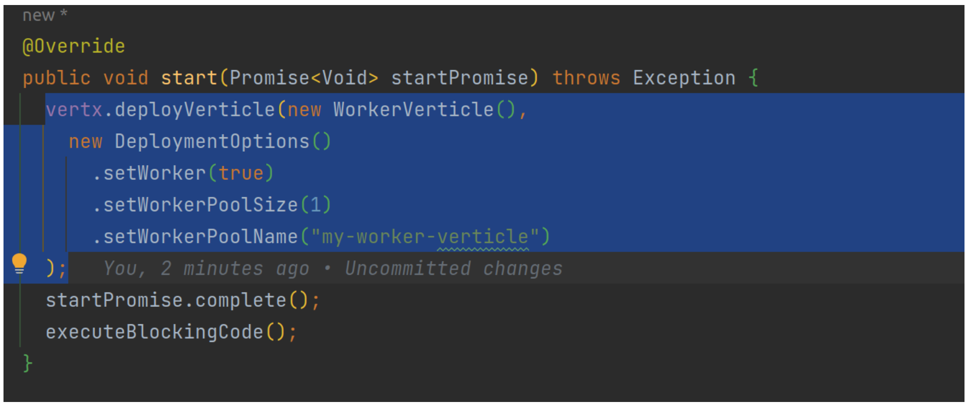
Promise

In Vert.x, una Promise è un **oggetto utilizzato per rappresentare l'esito (successo o fallimento) di un'operazione asincrona.** Una Promise viene utilizzata per fornire un'interfaccia di programmazione semplice e intuitiva per gestire il risultato di un'operazione asincrona.

Worker verticle

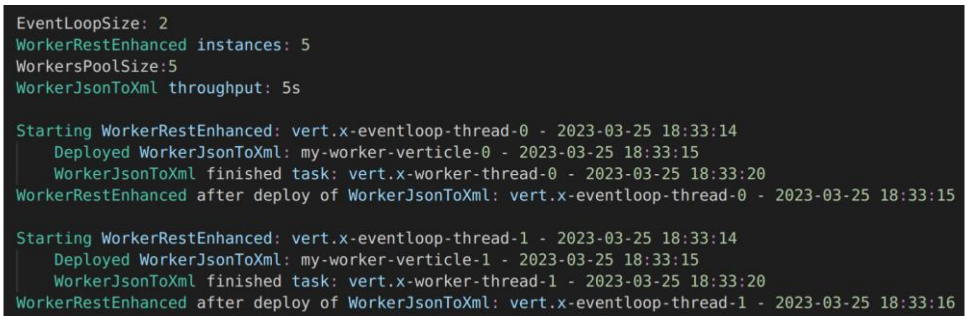


```java
1 usage   new *
public class WorkerVerticle extends AbstractVerticle {

  public static final Logger LOG = LoggerFactory.getLogger(WorkerVerticle.class);

  new *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    LOG.debug( o: "Deployed as worker verticle");
    startPromise.complete();
    Thread.sleep( millis: 5000);    You, Moments ago • Uncommitted changes
  }
}
```



```java
new *
@Override
public void start(Promise<Void> startPromise) throws Exception {
  vertx.deployVerticle(new WorkerVerticle(),
    new DeploymentOptions()
      .setWorker(true)
      .setWorkerPoolSize(1)
      .setWorkerPoolName("my-worker-verticle")
  );  You, 2 minutes ago • Uncommitted changes
  startPromise.complete();
  executeBlockingCode();
}
```

Test



```
EventLoopSize: 2
WorkerRestEnhanced instances: 5
WorkersPoolSize:5
WorkerJsonToXml throughput: 5s

Starting WorkerRestEnhanced: vert.x-eventloop-thread-0 - 2023-03-25 18:33:14
    Deployed WorkerJsonToXml: my-worker-verticle-0 - 2023-03-25 18:33:15
    WorkerJsonToXml finished task: vert.x-worker-thread-0 - 2023-03-25 18:33:20
WorkerRestEnhanced after deploy of WorkerJsonToXml: vert.x-eventloop-thread-0 - 2023-03-25 18:33:15

Starting WorkerRestEnhanced: vert.x-eventloop-thread-1 - 2023-03-25 18:33:14
    Deployed WorkerJsonToXml: my-worker-verticle-1 - 2023-03-25 18:33:15
    WorkerJsonToXml finished task: vert.x-worker-thread-1 - 2023-03-25 18:33:20
WorkerRestEnhanced after deploy of WorkerJsonToXml: vert.x-eventloop-thread-1 - 2023-03-25 18:33:16
```

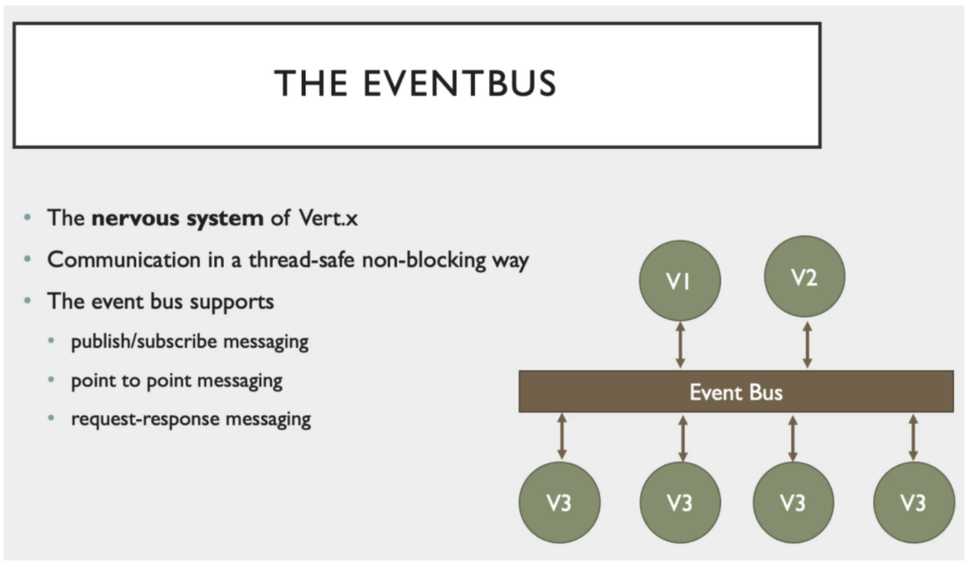
Il fatto di impostare setWorker(true) nelle DeploymentOptions() indica semplicemente che il verticle verrà eseguito in un thread separato, ma non significa che tutte le operazioni all'interno del verticle verranno eseguite in modo asincrono.

Quindi, se si desidera eseguire un'operazione bloccante in un worker verticle, è comunque necessario utilizzare il metodo executeBlocking(), che consente di eseguire l'operazione in un thread separato e restituire il risultato al thread principale di Vert.x.

Event Bus

Each Verticle runs in its own thread and sometimes has to communicate with other Verticles.The **Vert.x event-bus** is used for this purpose.

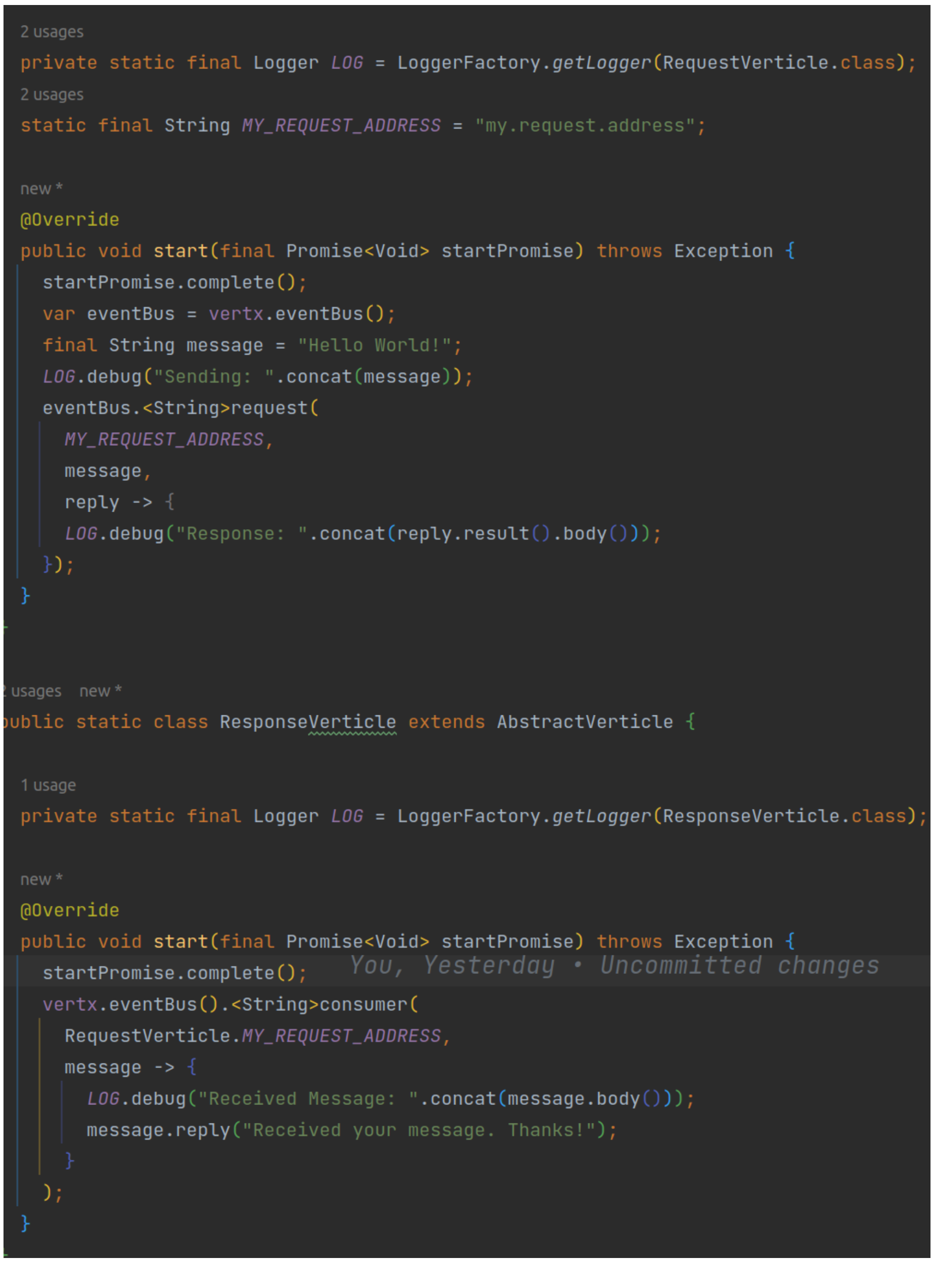
There is a an event-bus instance avalilable per vertx instance available.



THE EVENTBUS

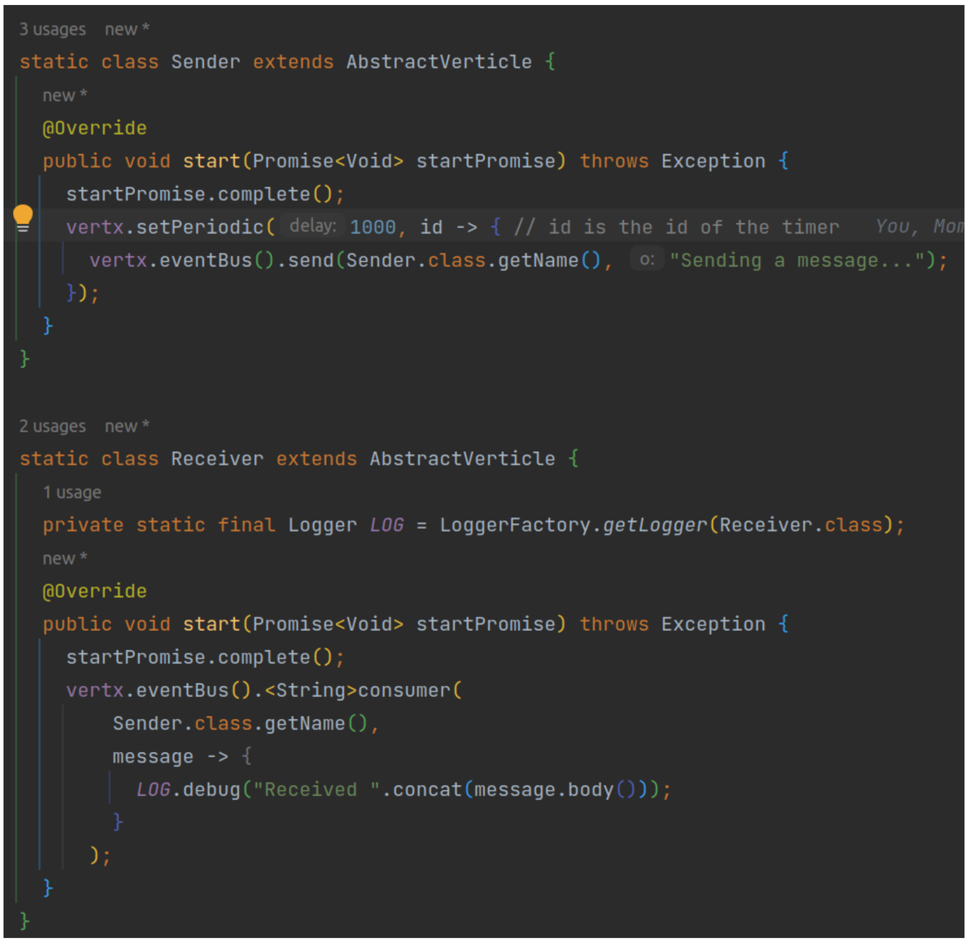
- The **nervous system** of Vert.x
- Communication in a thread-safe non-blocking way
- The event bus supports
  - publish/subscribe messaging
  - point to point messaging
  - request-response messaging

Request Response Messages



```
2 usages
private static final Logger LOG = LoggerFactory.getLogger(RequestVerticle.class);
2 usages
static final String MY_REQUEST_ADDRESS = "my.request.address";

new *
@Override
public void start(final Promise<Void> startPromise) throws Exception {
  startPromise.complete();
  var eventBus = vertx.eventBus();
  final String message = "Hello World!";
  LOG.debug("Sending: ".concat(message));
  eventBus.<String>request(
    MY_REQUEST_ADDRESS,
    message,
    reply -> {
    LOG.debug("Response: ".concat(reply.result().body()));
  });
}


2 usages   new *
public static class ResponseVerticle extends AbstractVerticle {

1 usage
private static final Logger LOG = LoggerFactory.getLogger(ResponseVerticle.class);

new *
@Override
public void start(final Promise<Void> startPromise) throws Exception {
  startPromise.complete();    You, Yesterday • Uncommitted changes
  vertx.eventBus().<String>consumer(
    RequestVerticle.MY_REQUEST_ADDRESS,
    message -> {
      LOG.debug("Received Message: ".concat(message.body()));
      message.reply("Received your message. Thanks!");
    }
  );
}
```

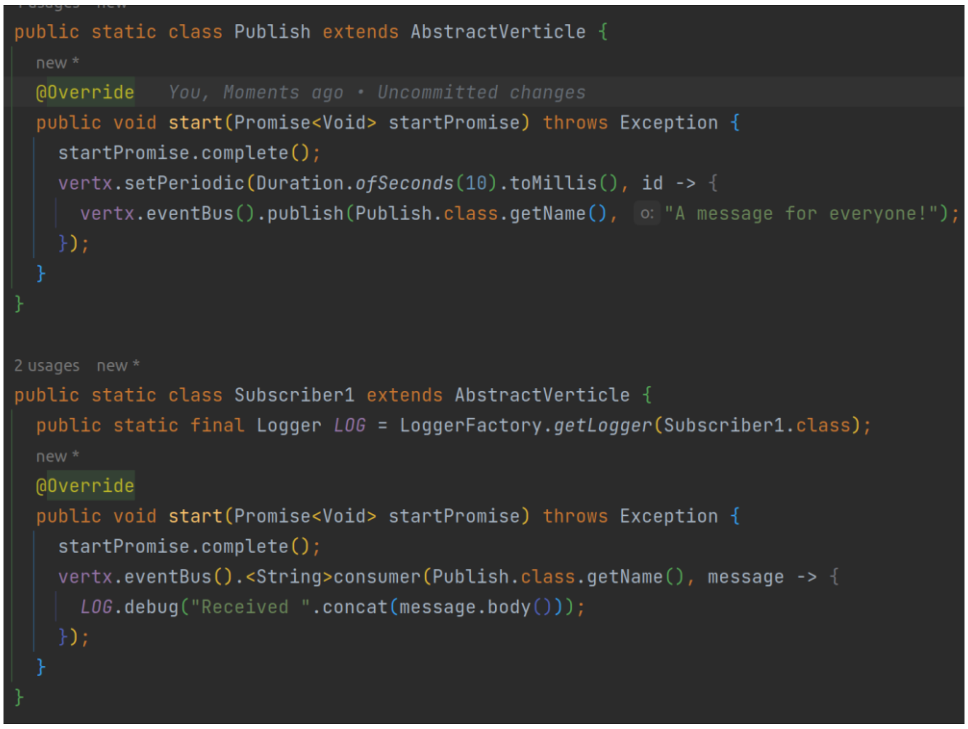
Point to Point messages



```
3 usages   new *
static class Sender extends AbstractVerticle {
  new *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    startPromise.complete();
    vertx.setPeriodic( delay: 1000, id -> { // id is the id of the timer    You, Mon
      vertx.eventBus().send(Sender.class.getName(),  o: "Sending a message...");
    });
  }
}

2 usages   new *
static class Receiver extends AbstractVerticle {
  1 usage
  private static final Logger LOG = LoggerFactory.getLogger(Receiver.class);
  new *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    startPromise.complete();
    vertx.eventBus().<String>consumer(
        Sender.class.getName(),
        message -> {
          LOG.debug("Received ".concat(message.body()));
        }
      );
  }
}
```

Publish Subscribe Messages

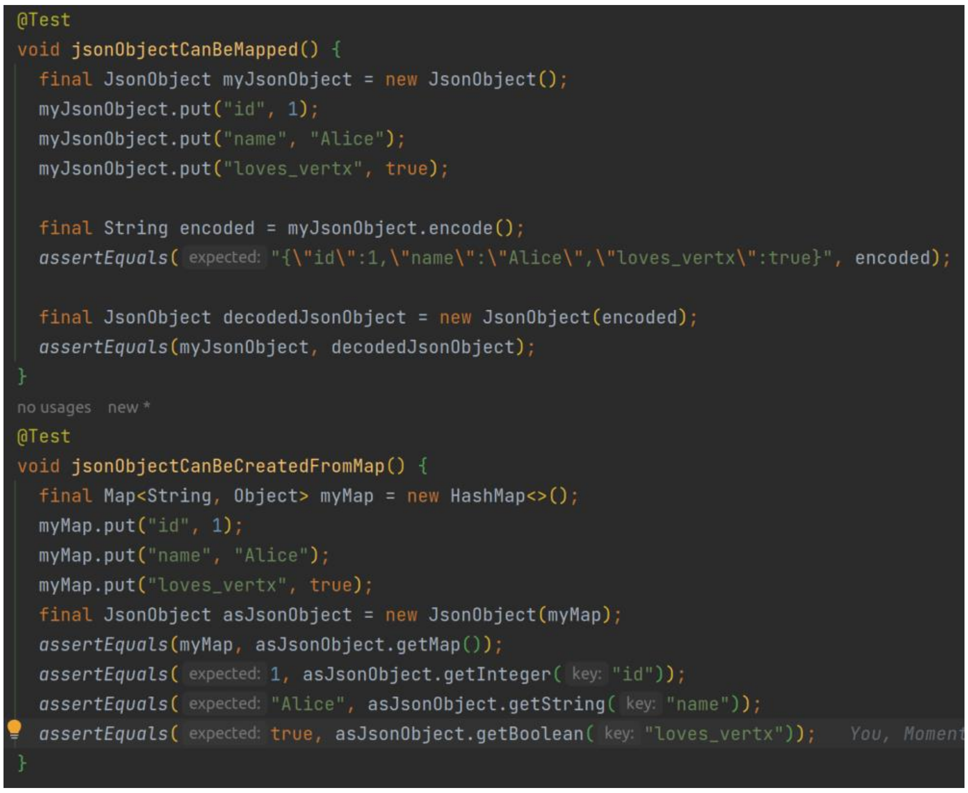
One Verticle can publish a message and multilple consumers can read it (as a broadcast message).



```
public static class Publish extends AbstractVerticle {
  new *
  @Override   You, Moments ago • Uncommitted changes
  public void start(Promise<Void> startPromise) throws Exception {
    startPromise.complete();
    vertx.setPeriodic(Duration.ofSeconds(10).toMillis(), id -> {
      vertx.eventBus().publish(Publish.class.getName(),  o: "A message for everyone!");
    });
  }
}

2 usages   new *
public static class Subscriber1 extends AbstractVerticle {
  public static final Logger LOG = LoggerFactory.getLogger(Subscriber1.class);
  new *
  @Override
  public void start(Promise<Void> startPromise) throws Exception {
    startPromise.complete();
    vertx.eventBus().<String>consumer(Publish.class.getName(), message -> {
      LOG.debug("Received ".concat(message.body()));
    });
  }
}
```

The JSON Object

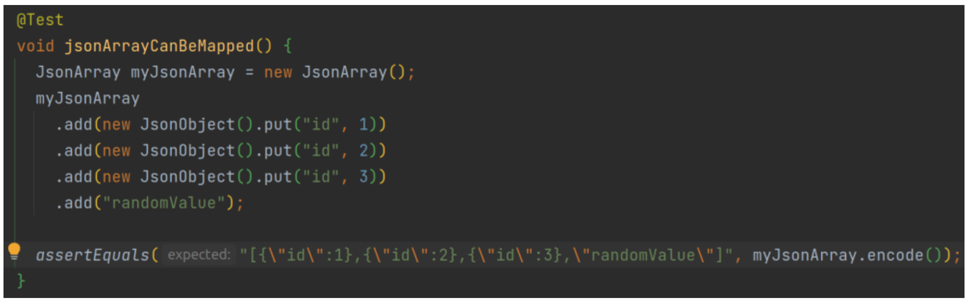
It is possible to convert a Java Object to a Vert.x JSON Object



```java
@Test
void jsonObjectCanBeMapped() {
  final JsonObject myJsonObject = new JsonObject();
  myJsonObject.put("id", 1);
  myJsonObject.put("name", "Alice");
  myJsonObject.put("loves_vertx", true);

  final String encoded = myJsonObject.encode();
  assertEquals( expected: "{\"id\":1,\"name\":\"Alice\",\"loves_vertx\":true}", encoded);

  final JsonObject decodedJsonObject = new JsonObject(encoded);
  assertEquals(myJsonObject, decodedJsonObject);
}
no usages   new *
@Test
void jsonObjectCanBeCreatedFromMap() {
  final Map<String, Object> myMap = new HashMap<>();
  myMap.put("id", 1);
  myMap.put("name", "Alice");
  myMap.put("loves_vertx", true);
  final JsonObject asJsonObject = new JsonObject(myMap);
  assertEquals(myMap, asJsonObject.getMap());
  assertEquals( expected: 1, asJsonObject.getInteger( key: "id"));
  assertEquals( expected: "Alice", asJsonObject.getString( key: "name"));
  assertEquals( expected: true, asJsonObject.getBoolean( key: "loves_vertx"));    You, Moment
}
```

The JSON Array

The JSON array is a sequence of values that can be of type number, string, boolean or other objects.



```java
@Test
void jsonArrayCanBeMapped() {
  JsonArray myJsonArray = new JsonArray();
  myJsonArray
    .add(new JsonObject().put("id", 1))
    .add(new JsonObject().put("id", 2))
    .add(new JsonObject().put("id", 3))
    .add("randomValue");

  assertEquals( expected: "[{\"id\":1},{\"id\":2},{\"id\":3},\"randomValue\"]", myJsonArray.encode());
}
```