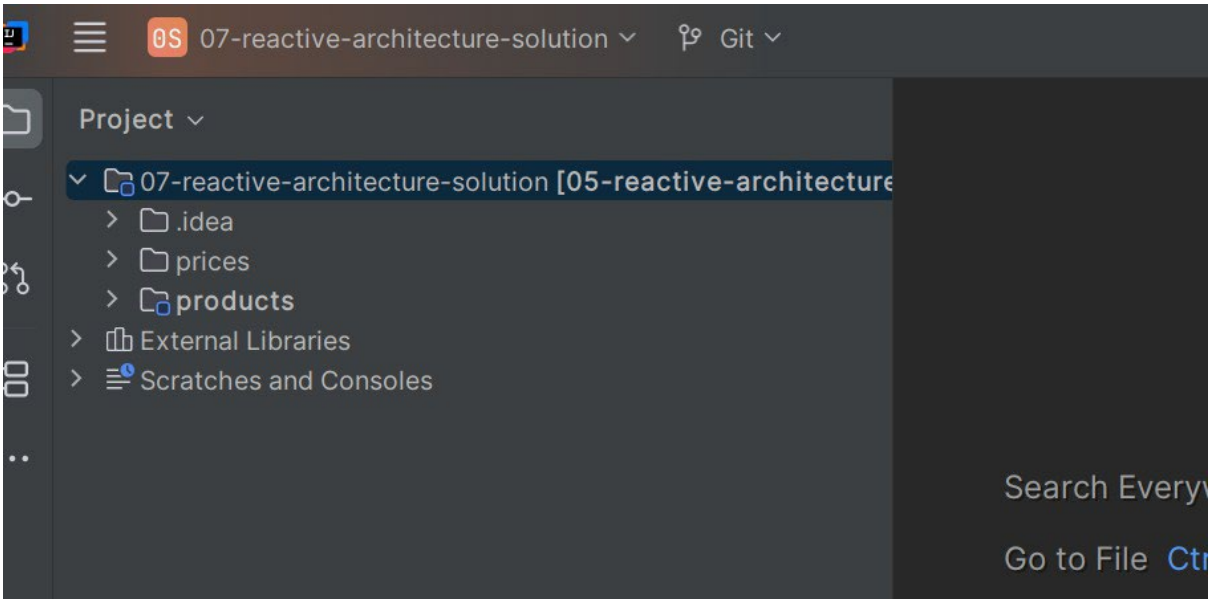
LAB 10: QUARKUS REACTIVE ARCHITECTURE

Autor: José Díaz

Github Repo: https://github.com/joedayz/quarkus-bcp-2025.git

1. Abre el proyecto **reactive-architecture-start**.



Instrucciones
Considera un servicio **products** que expone una API REST. Cada solicitud al API **products** usa un hilo de trabajo. Si el servicio **products** recibe más solicitudes de las que este hilo puede manejar, entonces la aplicación comienza a poner en cola las solicitudes.

Nota
Para fines de demostración en este ejercicio, el servicio **products** está configurado para usar solo un hilo de trabajo y solo un hilo de *event-loop*.

El endpoint **GET /product/{id}/priceHistory** del servicio **products** depende del servicio **prices** para obtener los datos históricos de precios de un producto. Reunir datos históricos es un proceso costoso, por lo que el servicio **prices** tarda alrededor de dos segundos en atender una solicitud. Por lo tanto, cada solicitud a **/product/{id}/priceHistory** tarda al menos dos segundos y mantiene bloqueado el hilo de trabajo, esperando la respuesta del servicio **prices**.

Usa **REST** y **REST Client** en el servicio **products** para mitigar este problema.

Ejecuta el servicio **prices** según el container runtime que tengas configurado:

Podman:

```
podman run -d --name prices -p 5500:5000 --
restart=always docker.io/joedayz/do378-reactive-
architecture-prices:latest
```

Docker:

```
docker run -d --name prices -p 5500:5000 --restart=always
docker.io/joedayz/do378-reactive-architecture-prices:latest
```

1. Ejecuta el servicio **products** y verifica que los dos hilos no tienen suficiente capacidad para manejar la E/S lenta.

   1.1. Navega al directorio ~/D0378/reactive-architecture y abre el proyecto con el editor de tu preferencia.

   cd ~/07-reactive-architecture-start

   1.2. Inicia el servicio **products** en modo desarrollo.

```
[student@workstation reactive-architecture]$ mvn quarkus:dev
...output omitted...
INFO  [io.quarkus] ... Listening on: http://localhost:8080
...output omitted...
```

   1.3 Abre una terminal de Windows y envía un request a http://localhost:8080/product/1/priceHsitory . Verifica que el request toma alrededor de 2 segundos en finalizar.

```
[student@workstation ~]$ time curl http://localhost:8080/products/1/priceHistory
{"prices":[...output omitted...],"product_id":1}
real 0m2.247s
user 0m0.003s
sys 0m0.005s
```

Linux o MacOSX o Git Bash:

   time curl http://localhost:8080/products/1/priceHistory

Windows:

   Measure-Command { curl http://localhost:8080/products/1/priceHistory }

1.4 Ejecuta el script benchmark.sh. Este script envía 10 requests a http://localhost:8080/products/1/priceHistory en un segundo, pero, toma mas de 20 segundos en recibir todas las respuestas.

```
[student@workstation ~]$ time ~/D0378/reactive-architecture/benchmark.sh
...output omitted...
Sending request...

real 0m20.187s
user 0m0.050s
sys 0m0.036s
```

En Linux, macosx, git bash:

time benchmark.sh

En Windows Powershell, create un benchmark.ps1 que hace lo mismo:

```
# Utility script for executing the same command in multiple parallel threads.

param(
    [int]$Requests = 10
)

$command = { curl.exe -s http://localhost:8080/products/1/priceHistory > $null }

Write-Host ""

# Capturar el proceso de PowerShell actual para medir CPU
$process = Get-Process -Id $PID

# Medir tiempo real
$stopwatch = [System.Diagnostics.Stopwatch]::StartNew()

for ($i = 1; $i -le $Requests; $i++) {
    Write-Host "Sending request..."
    Start-Job $command | Out-Null
    Start-Sleep -Milliseconds 100
}

# Esperar a que terminen todos los jobs
Get-Job | Wait-Job | Receive-Job
Remove-Job *

$stopwatch.Stop()

# Obtener estadísticas de CPU del proceso
$process.Refresh()
$userTime = [TimeSpan]::FromTicks($process.UserProcessorTime.Ticks)
$sysTime = [TimeSpan]::FromTicks($process.PrivilegedProcessorTime.Ticks)

# Mostrar estadísticas en formato similar a time
```

```
Write-Host ""
Write-Host "real $($stopwatch.Elapsed.ToString('m\mss\.fff\s'))"
Write-Host "user $($userTime.ToString('m\mss\.fff\s'))"
Write-Host "sys  $($sysTime.ToString('m\mss\.fff\s'))"
```

NOTA: En el 07-reactive-architecture-solution/products está el archivo .ps1 que se menciona en el cuadro.

1.5 Inspecciona los logs de la aplicación. Verifica que el executor-thread-0 worker thread atendió los requests uno por uno, tomándole dos segundos por cada request.

```
INFO  [io.qua.htt.access-log] (executor-thread-0) ... 26/Jan/2023:14:08:28 ...
INFO  [io.qua.htt.access-log] (executor-thread-0) ... 26/Jan/2023:14:08:30 ...
INFO  [io.qua.htt.access-log] (executor-thread-0) ... 26/Jan/2023:14:08:32 ...
```

2. Agrega las dependencias REST y REST Client.

   2.1 Ejecuta los comandos:

   $ mvn quarkus:remove-extensions -Dextensions="rest,rest-jackson,rest-client,rest-client-jackson"

   Y luego:

   $ mvn quarkus:add-extensions -Dextensions="rest,rest-jackson,rest-client,rest-client-jackson"

3. Fuerza a que el /product/{id}/priceHistory endpoint sea una operación no bloqueante.

   3.1 En la clase ProductsResource.java usa la anotación @NonBlocking en el endpoint /product/{productId}/priceHistory.

```java
@GET
@NonBlocking
@Path( "/{productId}/priceHistory" )
public ProductPriceHistory getProductPriceHistory(
    @PathParam( "productId" ) final Long productId ) {
    return pricesService.getProductPriceHistory( productId );
}
```

   3.2 Re ejecuta el curl request. El request retorna un error porque estas tratando ejecutar el request al servicios prices, el cual es bloqueante, pero, desde una operación no bloqueante.

```
[student@workstation ~]$ curl http://localhost:8080/products/1/priceHistory | jq
{"details":"Error
 id ..., org.jboss.resteasy.reactive.common.core.BlockingNotAllowedException: ...}
```

Linux o macosx:

curl http://localhost:8080/products/1/priceHistory | jq

Windows:

Invoke-RestMethod http://localhost:8080/products/1/priceHistory | ConvertTo-Json -Depth 10

4. Modifica el cliente rest PricesService para que sea no bloqueante.

4.1 Modifica el método PricesService#getProductPriceHistory para retornar un Uni stream.

```
@GET
@Path( "/history/{productId}" )
Uni<ProductPriceHistory> getProductPriceHistory(
    @PathParam( "productId" ) final Long productId
);
```

4.2 Retorna a el ProductsResource.java y modifica el /product/{productId}/priceHistory método endpoint y retorna un Uni stream.

```
@GET
@NonBlocking
@Path( "/{productId}/priceHistory" )
public Uni<ProductPriceHistory> getProductPriceHistory(
    @PathParam( "productId" ) final Long productId ) {
    return pricesService.getProductPriceHistory( productId );
}
```

4.3 Para la aplicación y retorna **mvn quarkus:dev**.

4.4 Re ejecuta el request y verifica que el endpoint asíncrono retorna una respuesta válida.

curl http://localhost:8080/1/priceHistory | jq

```
[student@workstation ~]$ curl http://localhost:8080/products/1/priceHistory | jq
{"prices":...output omitted...],"product_id":1}
```

4.5 Inspecciona los logs de la aplicación. Verifica que el vert.x-eventloop-thread-0 atiende el request.

```
INFO  [io.qua.htt.access-log] (vert.x-eventloop-thread-0) ...
```

4.6 Ejecuta el script benchmark y verifica que la aplicación atiende los requests en menos tiempo.

```
[student@workstation ~]$ time ~/DO378/reactive-architecture/benchmark.sh
...output omitted...
Sending request...

real 0m3.119s
user 0m0.042s
sys 0m0.042s
```

Linux o macosx:

time benchmark.sh

Windows powershell:

benchmark.ps1

El tiempo de respuesta no-bloqueante procesa los 10 requests seis veces más rápido que usando la estrategia bloqueante.

4.7 Inspecciona los logs de la aplicación y verifica que el vert.x-eventloop-thread-0 atiende todos los requests.

```
...output omitted...
INFO [io...access-log] (vert.x-eventloop-thread-0) ... 26/Jan/2023:14:15:19 ...
INFO [io...access-log] (vert.x-eventloop-thread-0) ... 26/Jan/2023:14:15:19 ...
INFO [io...access-log] (vert.x-eventloop-thread-0) ... 26/Jan/2023:14:15:20 ...
INFO [io...access-log] (vert.x-eventloop-thread-0) ... 26/Jan/2023:14:15:20 ...
...output omitted...
```

5. Verificar que REST reactive trata a los métodos que retornan un stream como operaciones no bloqueantes.

   5.1 Elimina la anotación @NonBlocking del endpoint /product/{productId}/priceHistory.

```
@GET
@Path( "/{productId}/priceHistory" )
public Uni<ProductPriceHistory> getProductPriceHistory(
    @PathParam( "productId" ) final Long productId ) {
    return pricesService.getProductPriceHistory( productId );
}
```

   5.2 Re ejecuta el request y verifica que el endpoint asíncrono retorna una respuesta valida.

```
[student@workstation ~]$ curl http://localhost:8080/products/1/priceHistory | jq
{"prices":...output omitted...],"product_id":1}
```

   curl http://localhost:8080/products/1/priceHistory | jq

   Windows powershell:

   Invoke-RestMethod http://localhost:8080/products/1/priceHistory | ConvertTo-Json -Depth 10

6. Bloquea el event loop.

   6.1 Envía un request al endpoint /products/blocking. Esta operación, aunque es ejecutada de forma asíncrona, bloquea el event loop por 30 segundos.

```
[student@workstation ~]$ curl http://localhost:8080/products/blocking; echo
```

   You might see a `io.vertx.core.VertxException: Thread blocked` error in the application logs.

   curl http://localhost:8080/products/blocking

   Veremos un io.vertx.core.VertxException: Thread blocked error en el log de la aplicación.

   6.2 Mientras esperas que el endpoint bloqueante responda, abre una nueva terminal

y ejecuta el script benchmark. Verifica que el benchmark es ahora muy lento, porque, el evento loop thread está bloqueado.

time benchmark.sh

o

benchmark.ps1

```
[student@workstation ~]$ time ~/D0378/reactive-architecture/benchmark.sh
...output omitted...
Sending request...

real 0m23.145s
user 0m0.042s
sys 0m0.042s
```

7. Anota el endpoint /products/blocking con la anotación @Blocking y re ejecuta el benchmark.

   7.1 Agrega la anotación @Blocking al endpoint /products/blocking.

```
@GET
@Blocking
@Path( "/blocking" )
public Uni<String> blocking() {
    ...implementation omitted...
}
```

   7.2 Envía un request al endpoint /blocking. Esta operación es ahora asignada a un worker thread.

   curl http://localhost:8080/products/blocking;

   7.3 Mientras esperas que el endpoint bloqueante responda, re ejecuta el script benchmark. Verifica que el benchmark es mucho más rápido ahora.

   time benchmark.sh

   ó

   benchmark.ps1

```
[student@workstation ~]$ time ~/D0378/reactive-architecture/benchmark.sh
...output omitted...
Sending request...

real 0m3.164
user 0m0.037s
sys 0m0.043s
```

7.4 Inspecciona los logs de la aplicación. Verifica que el endpoint bloqueante se ejecuta en un worker thread.

```
INFO  [io.qua.htt.access-log] (vert.x-eventloop-thread-0) ... "GET /products/1/
priceHistory HTTP/1.1" 200 6741
INFO  [io.qua.htt.access-log] (vert.x-eventloop-thread-0) ... "GET /products/1/
priceHistory HTTP/1.1" 200 6741
INFO  [io.qua.htt.access-log] (executor-thread-0) ... "GET /products/blocking
 HTTP/1.1" 200 25
```

8. Presiona **q** para detener la aplicación products.

Felicitaciones. Has terminado el laboratorio.

José