CloudApps
MASTER

DESARROLLO Y DESPLIEGUE DE APLICACIONES EN LA NUBE

REACTOR EN SPRING VS MUTINY EN QUARKUS

CURSO ACADÉMICO 2021/2022
TRABAJO DE FIN DE MÁSTER

Universidad
Rey Juan Carlos

Autores: Izan Santana Alonso y Diego Fernandez Aceves
Tutor: Micael Gallego Carrillo

INDICE

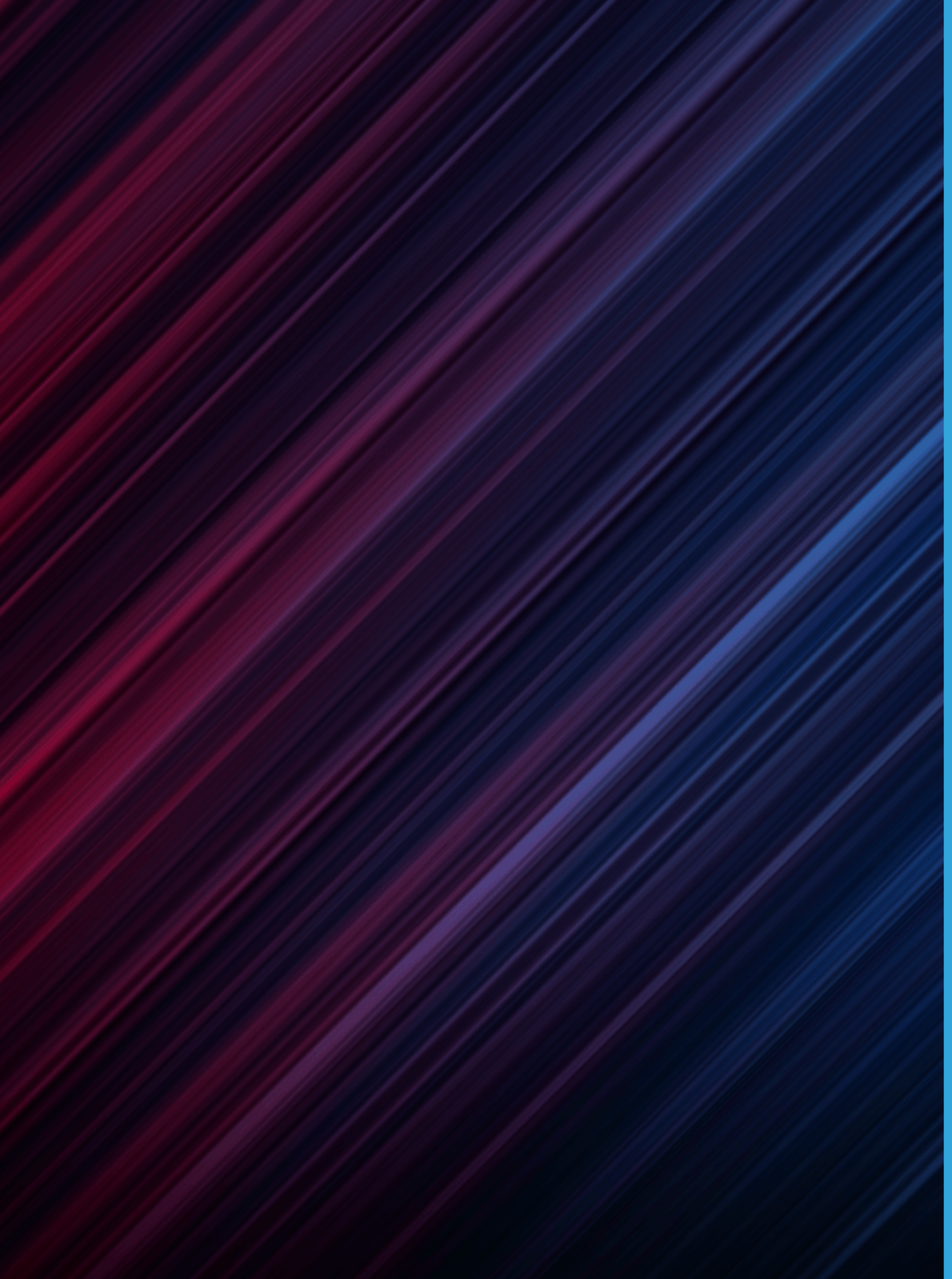
INTRODUCCIÓN Y OBJETIVOS

¿POR QUÉ ESTÁ COMPARATIVA?

- Aparición de nuevos frameworks modernos orientados al Cloud: Quarkus, Micronaut y Helidon
- Spring es genial por todo lo que aporta, pero ¿es mejor solución en cuanto a la arquitectura, rendimiento o consumo de recursos en todos los apartados?

¿POR QUÉ QUARKUS?

- Micronaut, está apoyado por la gente Grails, un framework que no terminó por llegar a buen puerto
- Helidon, no nos pareció que estuviera lo suficiente maduro y soportado por Oracle, y todos conocemos a Oracle
- Quarkus esta apoyado por RedHat, esto da una gran confianza en que el proyecto continúe en el tiempo y se siga desarrollando y mejorando



QUARKUS Y MUTINY

CURVA DE APRENDIZAJE Y DOCUMENTACIÓN

- Rápida, cómoda y agradable
- Buena documentación y ejemplos
- Gran variedad de extensiones y librerías

MUTINY

- Arquitectura orientada a eventos
- API sencilla y legible
- Multi y Uni son los tipos que permiten manejar elementos asincronos

COMPARATIVA QUARKUS VS SPRING

SERVIDOR HTTP REST Y PERSISTENCIA

Imperativo

	Imperativo	Reactivo
Cliente Rest HTTP	Endpoint Rest que invoca a otro servicio secundario	Endpoint Rest que invoca a otro servicio secundario
Servicio Rest HTTP y Persistencia	Desarrollo de un API (sobre películas) que ofrece diferentes operaciones y conecta con una base de datos	Desarrollo de un API (sobre películas) que ofrece diferentes operaciones y conecta con una base de datos
GRPC	Servicio gRPC que recibe un número, que multiplicado por un número aleatorio	Servicio gRPC que recibe un número, que multiplicado por un número aleatorio
AMQP	Produce y envía unos pocos mensajes, para luego ser consumidos	Implementación de un front donde añades una ciudad y devuelve el tiempo hace en mayúsculas o minúsculas
WebSockets	Desarrollo de un chat para usuarios	Web donde se visualiza la cantidad de usuarios conectados a la aplicación

QUARKUS RESTEASY

```java
@Path("/movies")
public class MovieResource {

    @GET
    @Path("/{movieId}")
    public Response getMovieById(@PathParam("movieId") Long movieId) {
        ...
    }

    @GET
    public List<MovieResponse> getAllMoviesByPage(@QueryParam("page") Optional<Integer> pageNumber) {
        ...
    }

    @POST
    public MovieResponse saveMovie(@RequestBody MovieRequest movieRequest) {
        ...
    }
}
```

SPRING MVC

```java
@RestController
@RequestMapping("/movies")
public class MovieController {

    @GetMapping("/{movieId}")
    public ResponseEntity<MovieResponse> getMovieById(@PathVariable("movieId") Long movieId) {
        ...
    }

    @GetMapping
    public List<MovieResponse> getAllMoviesByPage(@RequestParam("page") Optional<Integer> pageNumber) {
        ...
    }

    @PostMapping
    public MovieResponse saveMovie(@RequestBody MovieRequest movieRequest) {
        ...
    }
}
```

QUARKUS PANACHE – ACTIVE RECORD PATTERN

```java
@Entity(name = "movies")
public class Movie extends PanacheEntityBase {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String poster;
    ...

    public static List<Movie> findByPage(Page page) {
        return Movie.findAll().page(page).list();
    }
}
```

QUARKUS PANACHE – REPOSITORY PATTERN

```java
@ApplicationScoped
public class MovieRepository implements PanacheRepository<Movie> {

    public List<Movie> findByPage(Page page) {
        return findAll().page(page).list();
    }

}
```

QUARKUS – SPRING DATA JPA

```java
public interface MovieRepository extends JpaRepository<Movie, Long> {

    @Modifying
    @Query("update Movie m set m.rating = :rating where m.id = :movieId")
    void update(@Param("rating") Double rating, @Param("movieId") Long movieId);

}

```

SPRING DATA JPA

```java
public interface MovieRepository extends JpaRepository<Movie, Long> {

    @Modifying
    @Query("update Movie m set m.rating = :rating where m.id = :movieId")
    void update(@Param("rating") Double rating, @Param("movieId") Long movieId);

}

```

AMQP

Imperativo

QUARKUS – QUARKIVERSE RABBITMQ CLIENT

```java
@Startup
@ApplicationScoped
public class RabbitConfiguration {

    private final Channel channel;

    @Inject
    public RabbitConfiguration(RabbitMQClient rabbitMQClient, RabbitProperties rabbitProperties) {

        try {

            // Config exchange, queue and binding
            Connection connection = rabbitMQClient.connect();
            channel = connection.createChannel();
            channel.exchangeDeclare(rabbitProperties.getExchangeName(), BuiltinExchangeType.DIRECT, true);
            channel.queueDeclare(rabbitProperties.getQueueName(), true, false, false, null);
            channel.queueBind(rabbitProperties.getQueueName(), rabbitProperties.getExchangeName(), rabbitProperties.getRouterKey());

            // Set listener service class
            ListenerService listenerService = new ListenerService(channel);
            channel.basicConsume(rabbitProperties.getQueueName(), true, listenerService);

        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }

    }

    @Produces
    public Channel getChannel() {
        return this.channel;
    }

}
```

QUARKUS – QUARKIVERSE RABBITMQ CLIENT

```java
public class ListenerService extends DefaultConsumer {

    public ListenerService(Channel channel) {
        super(channel);
    }

    @Override
    public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) {
        ...
    }
}
```

```java
byte[] messageBytes = message.getBytes(StandardCharsets.UTF_8);
AMQP.BasicProperties basicProperties = new AMQP.BasicProperties();

channel.basicPublish(
  rabbitProperties.getExchangeName(), rabbitProperties.getRouterKey(),
  basicProperties, messageBytes);
```

SPRING AMQP

```java
@Configuration
public class RabbitConfiguration {

    @Bean
    public Queue myQueue() {
        return QueueBuilder.durable(rabbitProperties.getQueueName()).build();
    }

    @Bean
    public Exchange exchange() {
        return new DirectExchange(rabbitProperties.getExchangeName());
    }

    @Bean
    public Binding binding(Queue queue, Exchange exchange) {
        return BindingBuilder
                .bind(queue)
                .to(exchange)
                .with(rabbitProperties.getRouterKey())
                .noargs();
    }
}
```

SPRING AMQP

```java
@RabbitListener(queues = "#{rabbitProperties.getQueueName()}")
public void handleDelivery(String message) {
  log.info("Received: {}", message);
}
```

```java
rabbitTemplate.convertAndSend(rabbitProperties.getQueueName(), message);
```

SERVIDOR HTTP REST Y PERSISTENCIA

Reactivo

QUARKUS REACTIVE RESTEASY

```java
@Path("/movies")
public class MovieResource {

    @GET
    @Path("/{movieId}")
    public Uni<Response> getMovieById(@PathParam("movieId") Long movieId) {

        return Movie.findById(movieId)
                .onItem().ifNull().failWith(NotFoundException::new)
                .onItem().castTo(Movie.class)
                .map(this::convertToResponse)
                .map(Response::ok)
                .onFailure(NotFoundException.class).recoverWithItem(Response.status(NOT_FOUND))
                .map(Response.ResponseBuilder::build);
    }

    @GET
    public Uni<List<MovieResponse>> getAllMoviesByPage(@QueryParam("page") Optional<Integer> pageNumber) {
        ...
    }

    @POST
    public Uni<MovieResponse> saveMovie(@RequestBody MovieRequest movieRequest) {
        ...
    }

    ...
}
```

SPRING WEBFLUX

```java
@RestController
@RequestMapping("/movies")
public class MovieController {

    @GetMapping("/{movieId}")
    public Mono<ResponseEntity<MovieResponse>> getMovieById(@PathVariable("movieId") Long movieId) {
        return movieRepository.findById(movieId)
                .map(this::convertToResponse)
                .map(ResponseEntity::ok)
                .defaultIfEmpty(ResponseEntity.notFound().build());
    }

    @GetMapping
    public Mono<List<MovieResponse>> getAllMoviesByPage(@RequestParam("page") Optional<Integer> pageNumber) {
        ...
    }

    @PostMapping
    public Mono<MovieResponse> saveMovie(@RequestBody MovieRequest movieRequest) {
        ...
    }
    ...
}
```

QUARKUS REACTIVE PANACHE – ACTIVE RECORD PATTERN

```java
@Entity(name = "movies")
public class Movie extends PanacheEntityBase {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String poster;
    ...

    public static Uni<List<Movie>> findByPage(Page page) {
        return Movie.findAll().page(page).list();
    }
}
```

QUARKUS REACTIVE PANACHE – REPOSITORY PATTERN

```java
@ApplicationScoped
public class MovieRepository implements PanacheRepository<Movie> {

    public Uni<List<Movie>> findByPage(Page page) {
        return findAll().page(page).list();
    }

}
```

SPRING DATA R2DBC

```java
@Repository
public interface MovieRepository extends ReactiveCrudRepository<Movie, Long> {

    Flux<Movie> findAllBy(Pageable pageable);

    @Modifying
    @Query("update movies m set m.rating = :rating where m.id = :movieId")
    Mono<Void> update(@Param("rating") Double rating, @Param("movieId") Long movieId);

}
```

AMQP

Reactivo

QUARKUS SMALLRYE REACTIVE MESSAGING RABBITMQ

```java
@Path("/forecast")
public class ForecastResource {

    @Channel("request-forecast")
    MutinyEmitter<Forecast> processForecast;

    @Channel("forecast")
    Multi<Forecast> processedForecasts;

    @POST
    public Uni<Forecast> processForecast(City city) {
        return Uni.createFrom().item(city)
                .map(City::getName)
                .map(cityName -> new Forecast(UUID.randomUUID().toString(), cityName))
                .call(forecast -> processForecast.send(forecast));
    }

    @GET
    @Produces(MediaType.SERVER_SENT_EVENTS)
    public Multi<Forecast> getProcessedForecasts() {
        return processedForecasts;
    }
}
```

```java
@Incoming("process-forecast")
@Outgoing("processed-forecast")
public Uni<Forecast> listener(Forecast receivedForecast) {
  return Uni.createFrom().item(receivedForecast)
    .map(forecast -> {
      String weather = getRandomWeather(forecast.getCity());
      return new Forecast(forecast.getId(), forecast.getCity(), weather);
    })
    .onItem().delayIt().by(getRandomDuration());
}
```

SPRING STARTER AMQP Y REACTOR RABBITMQ

```java
@Bean
public Mono<Connection> reactiveRabbitMQConnection(RabbitProperties rabbitProperties) {
  ConnectionFactory connectionFactory = new ConnectionFactory();
  connectionFactory.useNio();
  connectionFactory.setHost(rabbitProperties.getHost());
  connectionFactory.setPort(rabbitProperties.getPort());
  connectionFactory.setUsername(rabbitProperties.getUsername());
  connectionFactory.setPassword(rabbitProperties.getPassword());
  return Mono.fromCallable(connectionFactory::newConnection).cache();
}
```

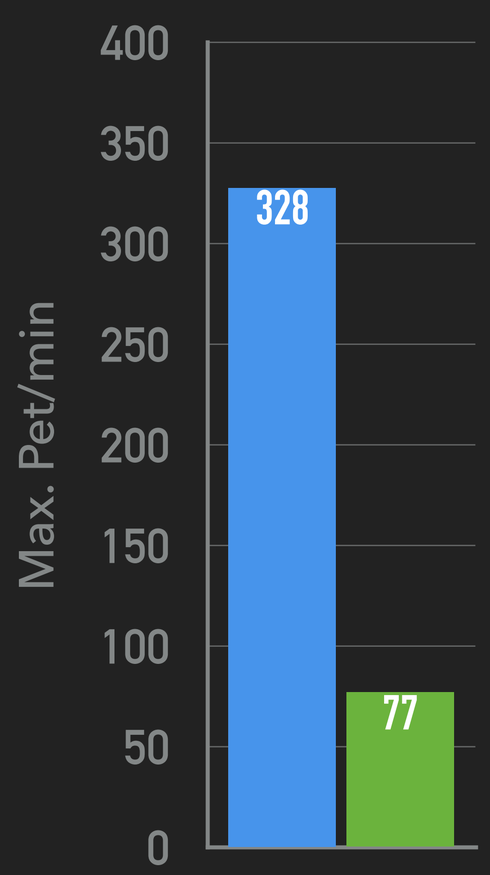
SPRING STARTER AMQP Y REACTOR RABBITMQ

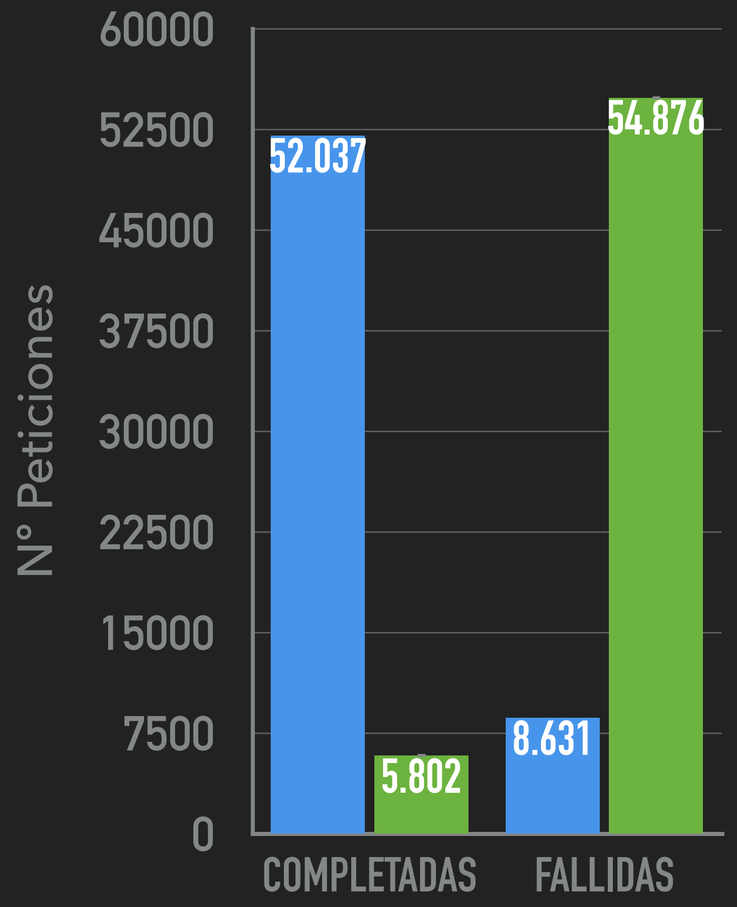
```java
@Bean
public Sender sender(Mono<Connection> reactiveRabbitMQConnection) {
  return RabbitFlux.createSender(new SenderOptions().connectionMono(reactiveRabbitMQConnection));
}

@Bean
public Receiver receiver(Mono<Connection> reactiveRabbitMQConnection) {
  return RabbitFlux.createReceiver(new ReceiverOptions().connectionMono(reactiveRabbitMQConnection));
}

@Bean
public Flux<Delivery> processForecastListener(final Receiver receiver, Binding processForecastBinding) {
  return receiver.consumeAutoAck(processForecastBinding.getDestination());
}

@Bean
public Flux<Delivery> processedForecastListener(final Receiver receiver, Binding processedForecastBinding) {
  return receiver.consumeAutoAck(processedForecastBinding.getDestination());
}
```

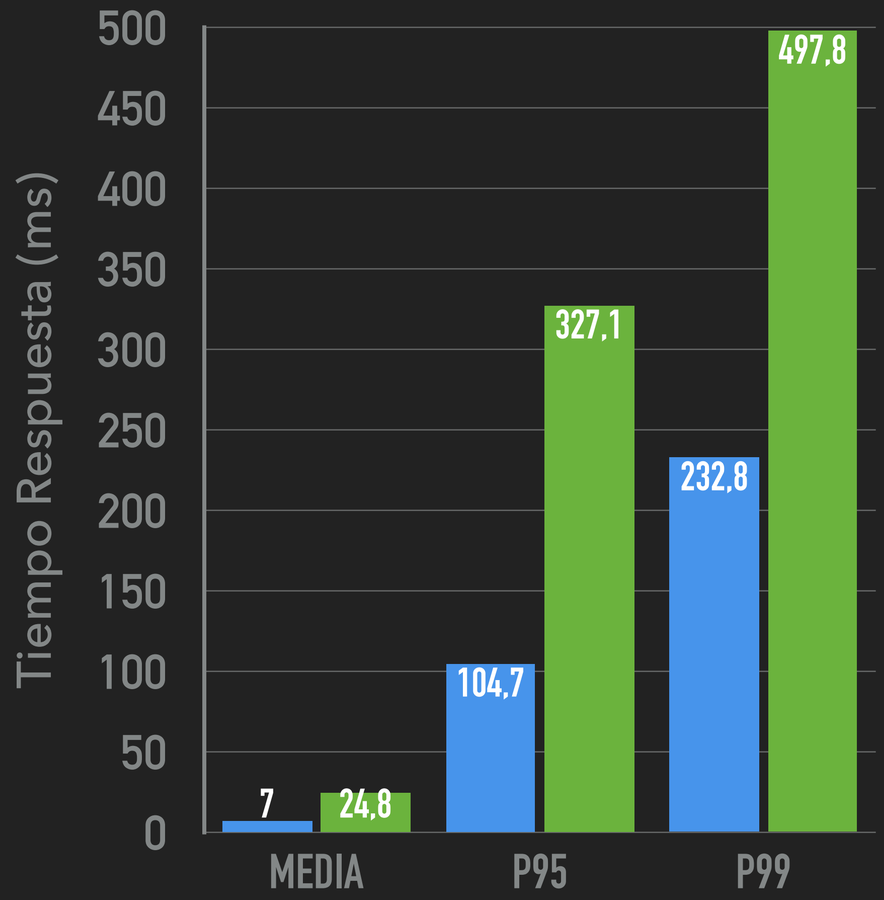
PRUEBAS DE CARGA

RESULTADOS DE RENDIMIENTO

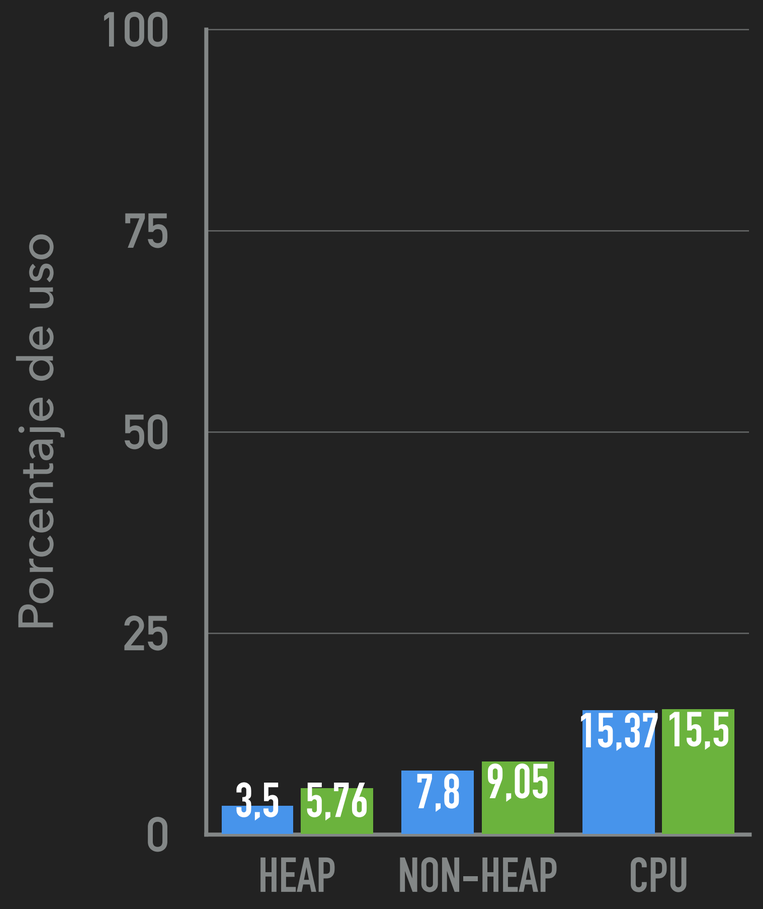
Imperativo

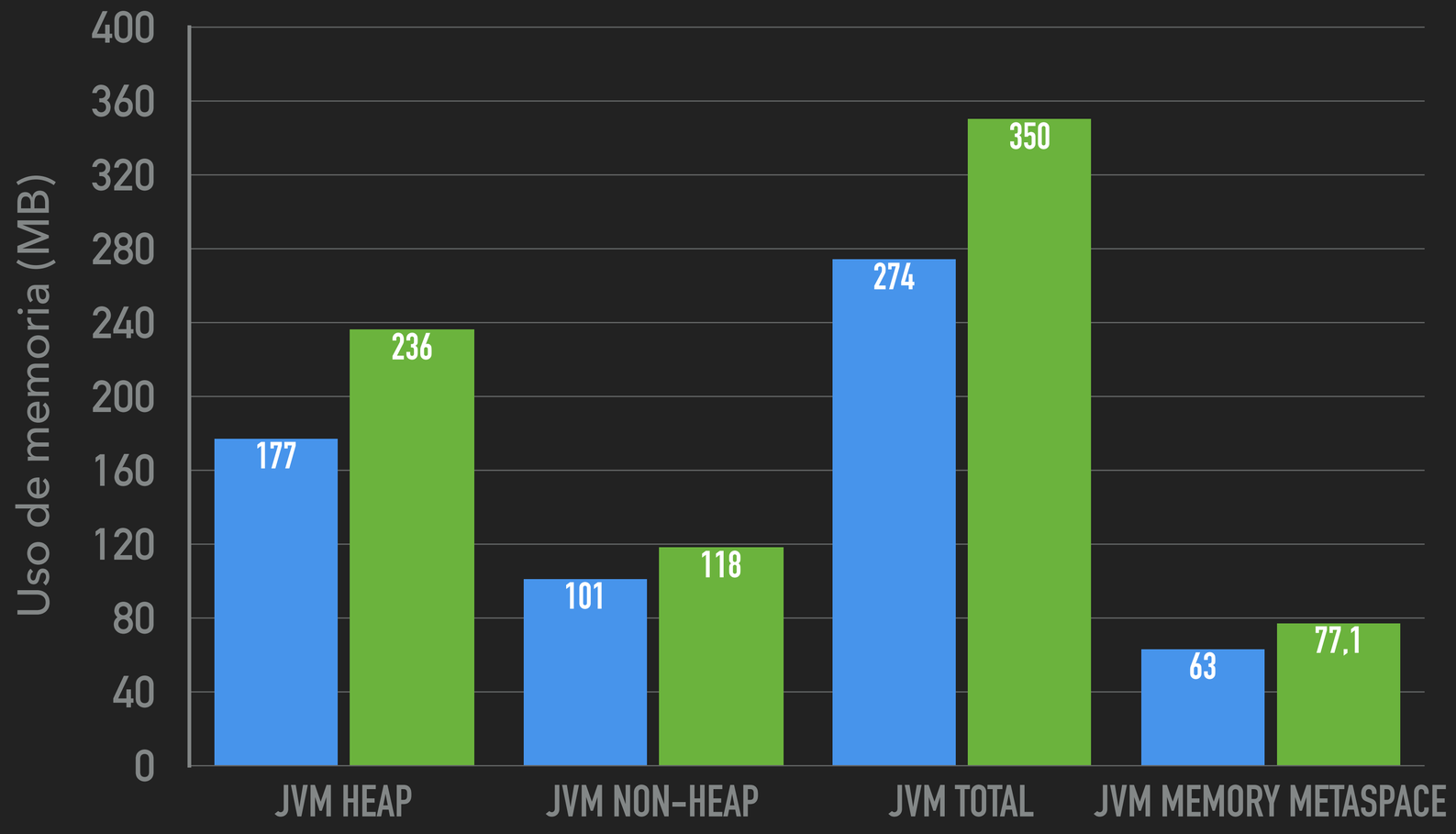






Quarkus Spring

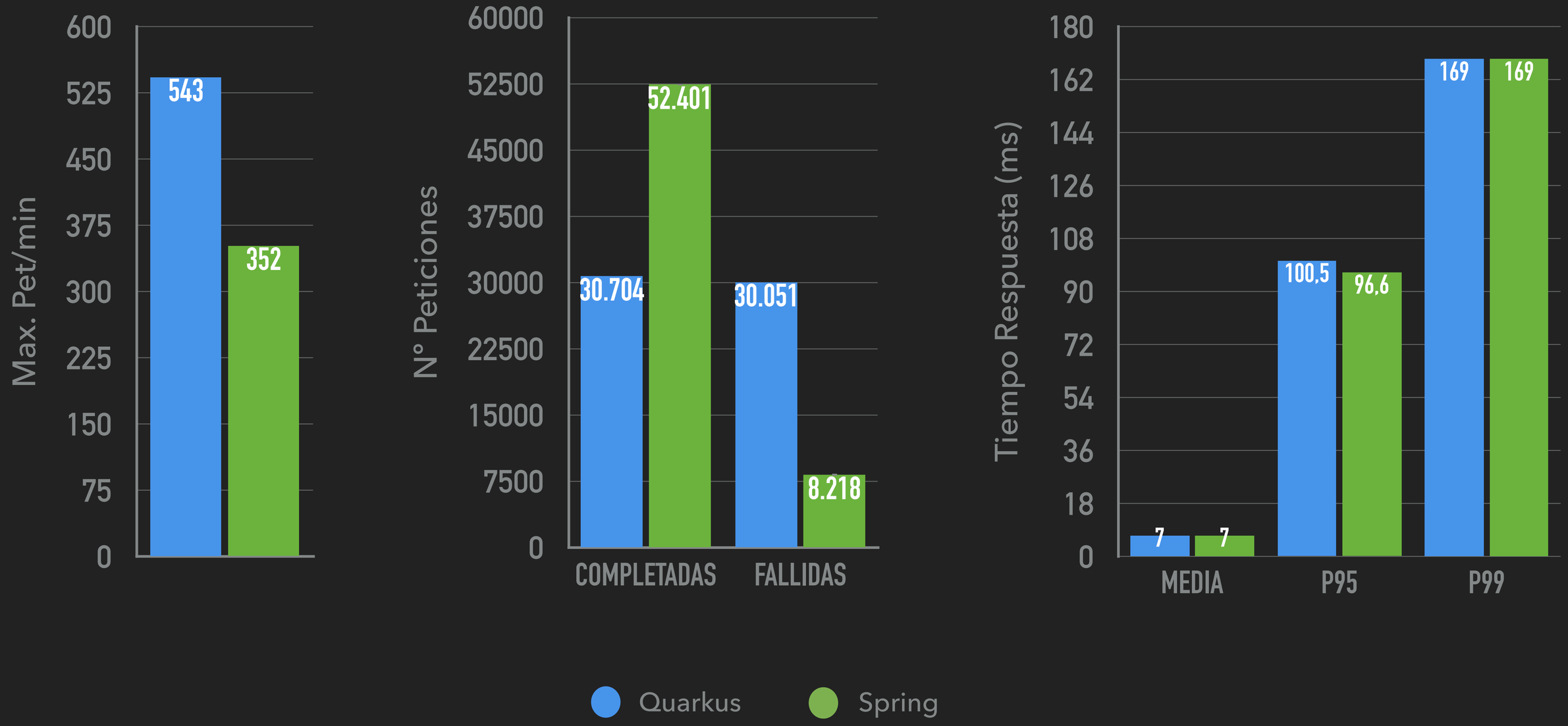


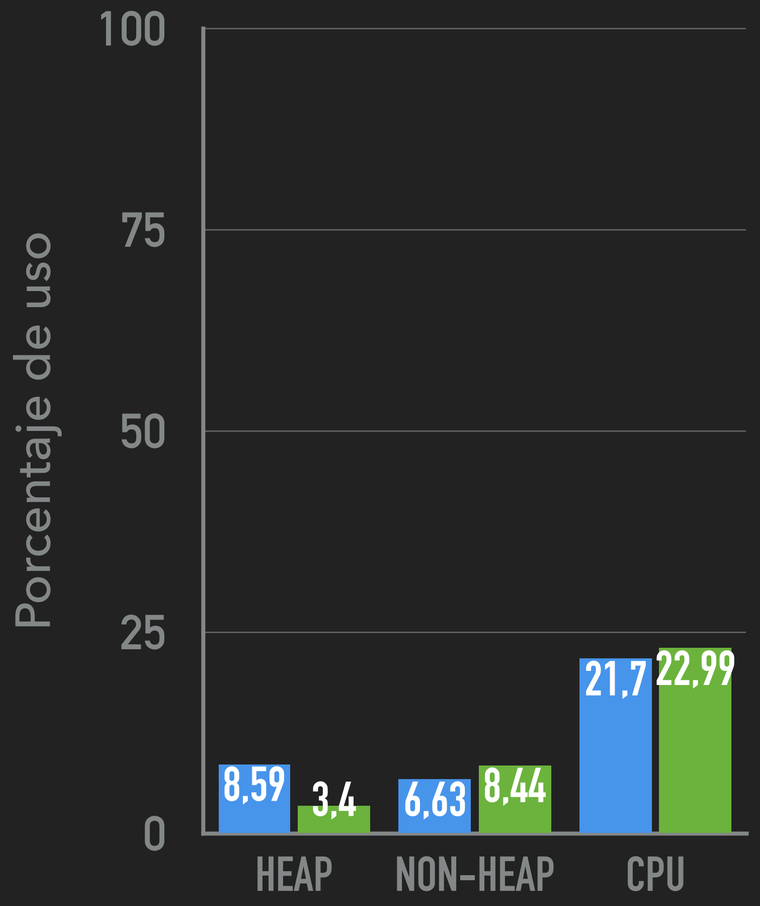


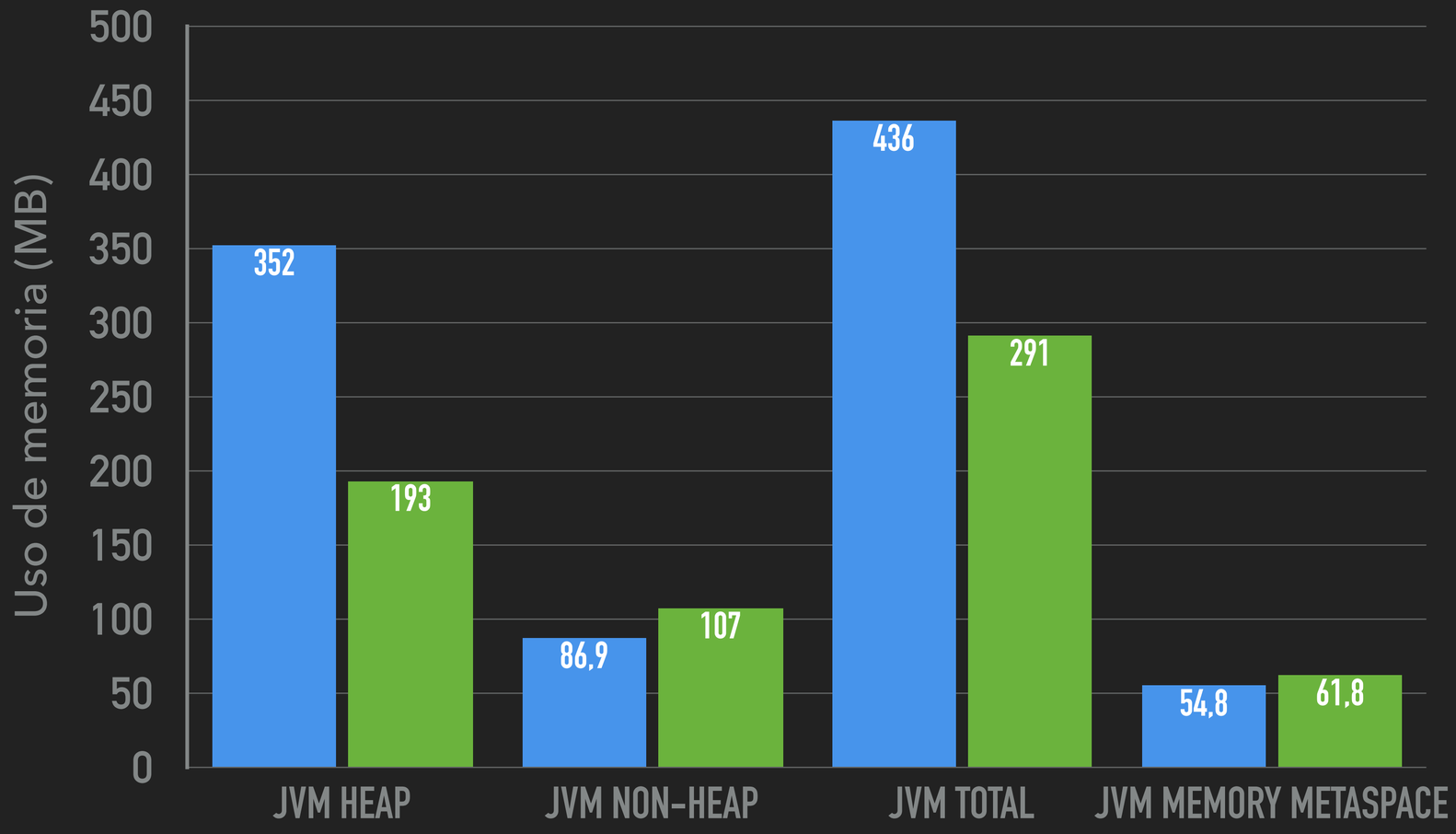
Quarkus Spring

RESULTADOS DE RENDIMIENTO

Reactivo


Max. Pet/min
600
525
450
375
300
225
150
75
0
543
352
Nº Peticiones
60000
52500
45000
37500
30000
22500
15000
7500
0
30.704
52.401
30.051
8.218
COMPLETADAS
FALLIDAS
Tiempo Respuesta (ms)
180
162
144
126
108
90
72
54
36
18
0
7
7
100,5
96,6
169
169
MEDIA
P95
P99
Quarkus
Spring





Quarkus Spring

CONCLUSIONES

EXPERIENCIA PERSONAL

- Buen rendimiento
- Fácil aprendizaje
- Soluciones sencillas, limpias y diferentes a Spring
- Pese a su madures las sensaciones están por encima de lo esperado
- Buena opción para aprender y llevar a proyectos productivos
- Muy rápido y sencillo crear imágenes nativas y manifiestos para Kubernetes

TRABAJOS FUTUROS

SIGUIENTES PASOS...

- Realizar mas pruebas de carga:
  - Sobre otros sistemas operativos y arquitecturas
  - Sobre el resto de tecnologías y analizar su comportamiento
  - Usando otras herramientas como k6 o wrk2
- Probar Quarkus en Kubernetes con imágenes JVM y nativas

DUDAS O PREGUNTAS

¡MUCHAS GRACIAS!

AUTORES: IZAN SANTANA ALONSO Y DIEGO FERNANDEZ ACEVES
TUTOR: MICAEL GALLEGO CARRILLO