



**CloudAPPS**  
MASTER

DESARROLLO Y DESPLIEGUE DE APLICACIONES EN LA NUBE

# REACTOR EN SPRING VS MUTINY EN QUARKUS

CURSO ACADÉMICO 2020/2021  
TRABAJO DE FIN DE MÁSTER

## ÍNDICE

- Introducción y objetivos
- Quarkus y Mutiny
- Compartiva Quarkus y Spring
- Pruebas de carga
- Trabajos futuros



# 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

- ▶ Cómoda y agradable, sobretodo para profesionales experimentados en Spring. Gran integración con librerías y frameworks existentes (Eclipse MicroProfile, JPA, Hibernate, etc)
- ▶ La documentación disponible es increíblemente extensa y detallada tanto por parte de RedHat como por la comunidad en general (Get Started, guías oficiales, tags, GH Discussions, etc).

## MUTINY

- ▶ Arquitectura orientada a eventos.
- ▶ API sencilla y explícita con la que hacer pipelines más legibles.
- ▶ Multi y Uni nos van a permitir manejar y combinar las interacciones con nuestro sistema o sistemas asíncronos.



# COMPARATIVA QUARKUS VS SPRING

# SERVIDOR HTTP REST Y PERSISTENCIA

Imperativo

# QUARKUS RESTEASY

```
● ● ●

1 @Path("/movies")
2 public class MovieResource {
3
4     @GET
5     @Path("/{movieId}")
6     public Response getMovieById(@PathParam("movieId") Long movieId) {
7         ...
8     }
9
10    @GET
11    public List<MovieResponse> getAllMoviesByPage(@QueryParam("page") Optional<Integer> pageNumber) {
12        ...
13    }
14
15    @POST
16    public MovieResponse saveMovie(@RequestBody MovieRequest movieRequest) {
17        ...
18    }
19 }
```

# SPRING MVC

```
1 @RestController
2 @RequestMapping("/movies")
3 public class MovieController {
4
5     @GetMapping("/{movieId}")
6     public ResponseEntity<MovieResponse> getMovieById(@PathVariable("movieId") Long movieId) {
7         ...
8     }
9
10    @GetMapping
11    public List<MovieResponse> getAllMoviesByPage(@RequestParam("page") Optional<Integer> pageNumber) {
12        ...
13    }
14
15    @PostMapping
16    public MovieResponse saveMovie(@RequestBody MovieRequest movieRequest) {
17        ...
18    }
19 }
```

# QUARKUS PANACHE - ACTIVE RECORD PATTERN

```
1 @Entity(name = "movies")
2 public class Movie extends PanacheEntityBase {
3
4     @Id
5     @GeneratedValue(strategy = GenerationType.IDENTITY)
6     private Long id;
7     private String poster;
8     ...
9
10    public static List<Movie> findByPage(Page page) {
11        return Movie.findAll().page(page).list();
12    }
13 }
```

# QUARKUS PANACHE - REPOSITORY PATTERN

```
● ● ●

1 @ApplicationScoped
2 public class MovieRepository implements PanacheRepository<Movie> {
3
4     public List<Movie> findByPage(Page page) {
5         return findAll().page(page).list();
6     }
7
8 }
```

# QUARKUS - SPRING DATA JPA

```
● ● ●  
1 public interface MovieRepository extends JpaRepository<Movie, Long> {  
2  
3     @Modifying  
4     @Query("update Movie m set m.rating = :rating where m.id = :movieId")  
5     void update(@Param("rating") Double rating, @Param("movieId") Long movieId);  
6  
7 }  
8
```

# SPRING DATA JPA

```
● ● ●  
1 public interface MovieRepository extends JpaRepository<Movie, Long> {  
2  
3     @Modifying  
4     @Query("update Movie m set m.rating = :rating where m.id = :movieId")  
5     void update(@Param("rating") Double rating, @Param("movieId") Long movieId);  
6  
7 }  
8
```

# AMQP

## Imperativo

# QUARKUS - QUARKIVERSE RABBITMQ CLIENT

```
● ● ●

1 @Startup
2 @ApplicationScoped
3 public class RabbitConfiguration {
4
5     private final Channel channel;
6
7     @Inject
8     public RabbitConfiguration(RabbitMQClient rabbitMQClient, RabbitProperties rabbitProperties) {
9
10        try {
11
12            // Config exchange, queue and binding
13            Connection connection = rabbitMQClient.connect();
14            channel = connection.createChannel();
15            channel.exchangeDeclare(rabbitProperties.getExchangeName(), BuiltinExchangeType.DIRECT, true);
16            channel.queueDeclare(rabbitProperties.getQueueName(), true, false, false, null);
17            channel.queueBind(rabbitProperties.getQueueName(), rabbitProperties.getExchangeName(), rabbitProperties.getRouterKey());
18
19            // Set listener service class
20            ListenerService listenerService = new ListenerService(channel);
21            channel.basicConsume(rabbitProperties.getQueueName(), true, listenerService);
22
23        } catch (IOException e) {
24            throw new UncheckedIOException(e);
25        }
26    }
27
28
29     @Produces
30     public Channel getChannel() {
31         return this.channel;
32     }
33
34 }
```

# QUARKUS - QUARKIVERSE RABBITMQ CLIENT

```
● ● ●  
1 public class ListenerService extends DefaultConsumer {  
2  
3     public ListenerService(Channel channel) {  
4         super(channel);  
5     }  
6  
7     @Override  
8     public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) {  
9         ...  
10    }  
11 }
```

```
● ● ●  
1 byte[] messageBytes = message.getBytes(StandardCharsets.UTF_8);  
2 AMQP.BasicProperties basicProperties = new AMQP.BasicProperties();  
3  
4 channel.basicPublish(  
5     rabbitProperties.getExchangeName(), rabbitProperties.getRouterKey(),  
6     basicProperties, messageBytes);
```

# SPRING AMQP

```
1 @Configuration
2 public class RabbitConfiguration {
3
4     @Bean
5     public Queue myQueue() {
6         return QueueBuilder.durable(rabbitProperties.getQueueName()).build();
7     }
8
9     @Bean
10    public Exchange exchange() {
11        return new DirectExchange(rabbitProperties.getExchangeName());
12    }
13
14    @Bean
15    public Binding binding(Queue queue, Exchange exchange) {
16        return BindingBuilder
17            .bind(queue)
18            .to(exchange)
19            .with(rabbitProperties.getRouterKey())
20            .noargs();
21    }
22 }
```

# SPRING AMQP

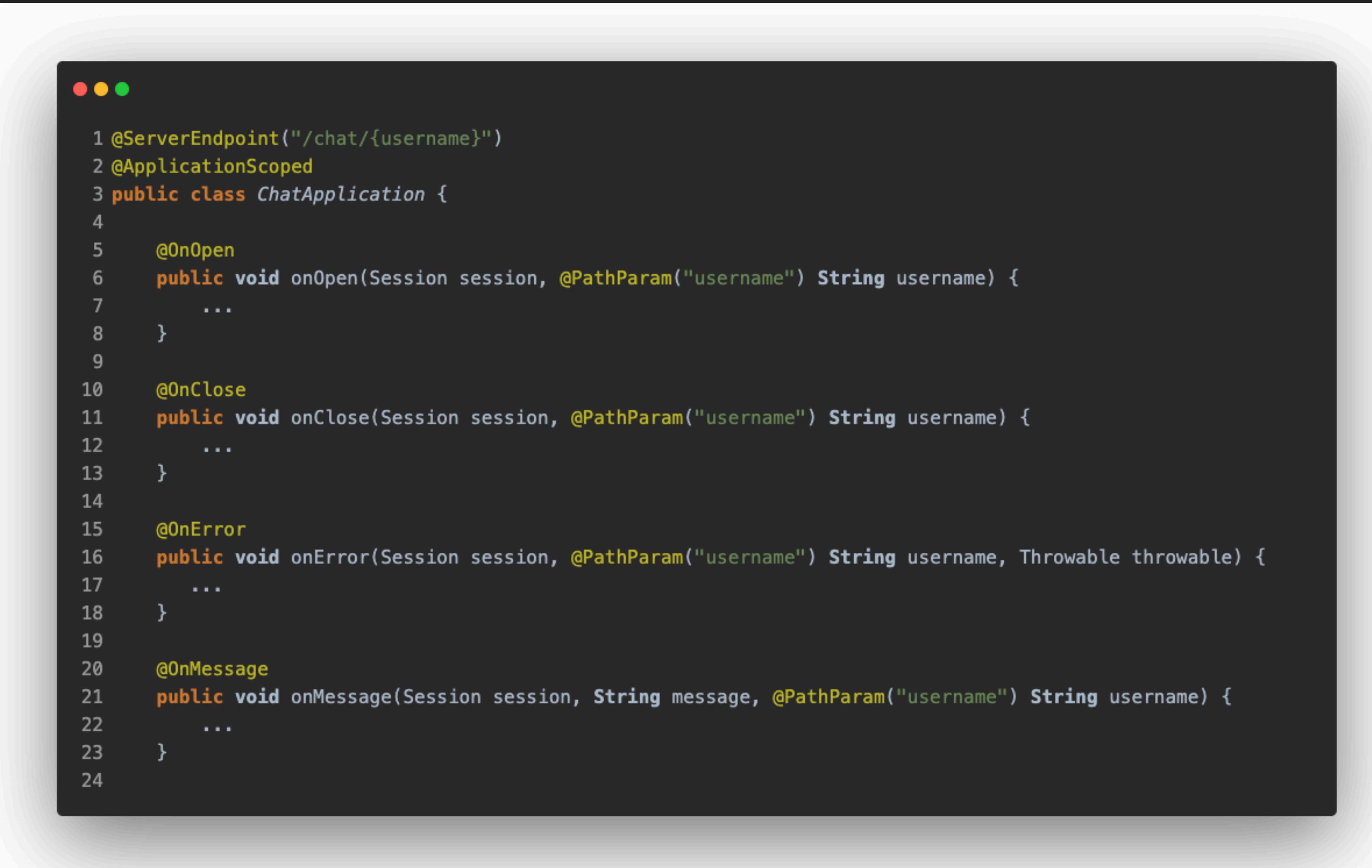
```
● ● ●  
1 @RabbitListener(queues = "#{rabbitProperties.getQueueName()}" )  
2 public void handleDelivery(String message) {  
3     log.info("Received: {}", message);  
4 }
```

```
● ● ●  
1 rabbitTemplate.convertAndSend(rabbitProperties.getQueueName(), message);
```

# WEBSOCKETS

Imperativo

# QUARKUS WEB SOCKETS



The image shows a terminal window with a dark background and light-colored text. At the top left are three colored window control buttons (red, yellow, green). The terminal displays a Java code snippet for a WebSocket endpoint. The code uses annotations from the `io.smallrye.reactive.messaging` package. It defines a class `ChatApplication` with methods for `@OnOpen`, `@OnClose`, `@OnError`, and `@OnMessage` events. Each method takes a `Session session` and a `@PathParam("username") String username` parameter. The code is numbered from 1 to 24 on the left side.

```
1 @ServerEndpoint("/chat/{username}")
2 @ApplicationScoped
3 public class ChatApplication {
4
5     @OnOpen
6     public void onOpen(Session session, @PathParam("username") String username) {
7         ...
8     }
9
10    @OnClose
11    public void onClose(Session session, @PathParam("username") String username) {
12        ...
13    }
14
15    @OnError
16    public void onError(Session session, @PathParam("username") String username, Throwable throwable) {
17        ...
18    }
19
20    @OnMessage
21    public void onMessage(Session session, String message, @PathParam("username") String username) {
22        ...
23    }
24}
```

# SPRING WEB SOCKET

```
1 @Configuration
2 @EnableWebSocketMessageBroker
3 public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {
4
5     @Override
6     public void configureMessageBroker(MessageBrokerRegistry config) {
7         config.enableSimpleBroker("/topic");
8         config.setApplicationDestinationPrefixes("/app");
9     }
10
11    @Override
12    public void registerStompEndpoints(StompEndpointRegistry registry) {
13        registry.addEndpoint("/chat").withSockJS();
14    }
15
16 }
```

```
1 @Controller
2 public class ChatApplication {
3
4     @MessageMapping("/connect/{username}")
5     @SendTo("/topic/messages")
6     public String onConnect(@DestinationVariable("username") String username,
7                           StompHeaderAccessor stompHeaderAccessor) {
8         ...
9     }
10
11    @MessageMapping("/message")
12    @SendTo("/topic/messages")
13    public String onMessage(StompHeaderAccessor stompHeaderAccessor, String message) {
14        ...
15    }
16
17    @MessageMapping("/disconnect")
18    @SendTo("/topic/messages")
19    public String onDisconnect(SimpMessageHeaderAccessor headerAccessor) {
20        ...
21    }
22 }
```

# SPRING WEB SOCKET

```
● ● ●

1 var connect = function () {
2   ...
3   var socket = new SockJS('/chat');
4   stompClient = Stomp.over(socket);
5
6   stompClient.connect({}, function (frame) {
7
8     stompClient.send("/app/connect/" + name);
9     ...
10    stompClient.subscribe('/topic/messages', function (message) {
11      ...
12    });
13  });
14 };
15 };
16
17 var sendMessage = function () {
18   ...
19   stompClient.send("/app/message", {}, value);
20   ...
21 }
22 };
23
24 var disconnect = function () {
25   stompClient.send("/app/disconnect");
26   stompClient.disconnect();
27 };
```

# CLIENTE REST HTTP

Imperativo

# QUARKUS MICROPROFILE CLIENT

```
es.urjc.CatFactResourceClient/mp-rest/url=https://catfact.ninja
```

```
● ● ●  
1 @Path("/fact")  
2 @RegisterRestClient  
3 public interface CatFactResourceClient {  
4  
5     @GET  
6     @Produces(MediaType.APPLICATION_JSON)  
7     CatFact getFact();  
8 }
```

# SPRING REST TEMPLATE CLIENT

```
● ● ● Title  
1 @SpringBootApplication  
2 public class ImperativeSpringRestClientApplication {  
3  
4     public static void main(String[] args) {  
5         SpringApplication.run(ImperativeSpringRestClientApplication.class, args);  
6     }  
7  
8     @Bean  
9     public RestTemplate restTemplate(RestTemplateBuilder builder) {  
10        return builder.build();  
11    }  
12}
```

# GRPC

## Imperativo

# QUARKUS

```
● ● ● Title  
1 @GrpcService  
2 public class GenerateRandomNumberService extends RandomNumberGeneratorGrpc.RandomNumberGeneratorImplBase {  
3  
4     private static final Logger log = Logger.getLogger(GenerateRandomNumberService.class.getName());  
5  
6     @Override  
7     @Blocking  
8     public void generateNumber(GenerateNumberRequest request, StreamObserver<GenerateNumberResponse> responseObserver) {  
9  
10        log.info("[QUARKUS IMPERATIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getId());  
11  
12        int randomInt = new Random()  
13            .ints(0, 30)  
14            .findFirst()  
15            .getAsInt();  
16  
17        int response = randomInt * request.getNumber();  
18  
19        responseObserver.onNext(GenerateNumberResponse.newBuilder().setNumber(response).build());  
20        responseObserver.onCompleted();  
21  
22        log.info("[QUARKUS IMPERATIVE] - END grpc SERVICE thread id " + Thread.currentThread().getId());  
23    }  
24 }
```

# SPRING

```
● ● ● Title  
1 @GrpcService  
2 public class GenerateRandomNumberService extends RandomNumberGeneratorGrpc.RandomNumberGeneratorImplBase {  
3  
4     private static final Logger log = LoggerFactory.getLogger(GenerateRandomNumberService.class);  
5  
6     @Override  
7     public void generateNumber(GenerateNumberRequest request, StreamObserver<GenerateNumberResponse> responseObserver) {  
8         log.info("[SPRING IMPERATIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getName());  
9  
10        int randomInt = new Random()  
11            .ints(0, 30)  
12            .findFirst()  
13            .getAsInt();  
14  
15        int response = randomInt * request.getNumber();  
16  
17        responseObserver.onNext(GenerateNumberResponse.newBuilder().setNumber(response).build());  
18        responseObserver.onCompleted();  
19  
20        log.info("[SPRING IMPERATIVE] - END grpc SERVICE thread id " + Thread.currentThread().getName());  
21  
22    }  
23  
24 }
```

# SERVIDOR HTTP REST Y PERSISTENCIA

Reactiva

# QUARKUS REACTIVE RESTEASY

```
● ● ●

1 @Path("/movies")
2 public class MovieResource {
3
4     @GET
5     @Path("/{movieId}")
6     public Uni<Response> getMovieById(@PathParam("movieId") Long movieId) {
7
8         return Movie.findById(movieId)
9             .onItem().ifNull().failWith(NotFoundException::new)
10            .onItem().castTo(Movie.class)
11            .map(this::convertToResponse)
12            .map(Response::ok)
13            .onFailure(NotFoundException.class).recoverWithItem(Response.status(NOT_FOUND))
14            .map(Response.ResponseBuilder::build);
15     }
16
17     @GET
18     public Uni<List<MovieResponse>> getAllMoviesByPage(@QueryParam("page") Optional<Integer> pageNumber) {
19         ...
20     }
21
22     @POST
23     public Uni<MovieResponse> saveMovie(@RequestBody MovieRequest movieRequest) {
24         ...
25     }
26
27     ...
28 }
```

# SPRING WEBFLUX

```
● ● ●

1 @RestController
2 @RequestMapping("/movies")
3 public class MovieController {
4
5     @GetMapping("/{movieId}")
6     public Mono<ResponseEntity<MovieResponse>> getMovieById(@PathVariable("movieId") Long movieId) {
7         return movieRepository.findById(movieId)
8             .map(this::convertToResponse)
9             .map(ResponseEntity::ok)
10            .defaultIfEmpty(ResponseEntity.notFound().build());
11    }
12
13    @GetMapping
14    public Mono<List<MovieResponse>> getAllMoviesByPage(@RequestParam("page") Optional<Integer> pageNumber) {
15        ...
16    }
17
18    @PostMapping
19    public Mono<MovieResponse> saveMovie(@RequestBody MovieRequest movieRequest) {
20        ...
21    }
22    ...
23 }
```

# QUARKUS REACTIVE PANACHE - ACTIVE RECORD PATTERN

```
● ● ●  
1 @Entity(name = "movies")  
2 public class Movie extends PanacheEntityBase {  
3  
4     @Id  
5     @GeneratedValue(strategy = GenerationType.IDENTITY)  
6     private Long id;  
7     private String poster;  
8     ...  
9  
10    public static Uni<List<Movie>> findByPage(Page page) {  
11        return Movie.findAll().page(page).list();  
12    }  
13 }
```

# QUARKUS REACTIVE PANACHE - REPOSITORY PATTERN

```
1 @ApplicationScoped
2 public class MovieRepository implements PanacheRepository<Movie> {
3
4     public Uni<List<Movie>> findByPage(Page page) {
5         return findAll().page(page).list();
6     }
7
8 }
```

# SPRING DATA R2DBC

```
1 @Repository
2 public interface MovieRepository extends ReactiveCrudRepository<Movie, Long> {
3
4     Flux<Movie> findAllBy(Pageable pageable);
5
6     @Modifying
7     @Query("update movies m set m.rating = :rating where m.id = :movieId")
8     Mono<Void> update(@Param("rating") Double rating, @Param("movieId") Long movieId);
9
10 }
```

# AMQP

## Reactivio

# QUARKUS SMALLRYE REACTIVE MESSAGING RABBITMQ

```
1 @Path("/forecast")
2 public class ForecastResource {
3
4     @Channel("request-forecast")
5     MutinyEmitter<Forecast> processForecast;
6
7     @Channel("forecast")
8     Multi<Forecast> processedForecasts;
9
10    @POST
11    public Uni<Forecast> processForecast(City city) {
12        return Uni.createFrom().item(city)
13            .map(City::getName)
14            .map(cityName -> new Forecast(UUID.randomUUID().toString(), cityName))
15            .call(forecast -> processForecast.send(forecast));
16    }
17
18    @GET
19    @Produces(MediaType.SERVER_SENT_EVENTS)
20    public Multi<Forecast> getProcessedForecasts() {
21        return processedForecasts;
22    }
23 }
```

```
1 @Incoming("process-forecast")
2 @Outgoing("processed-forecast")
3 public Uni<Forecast> listener(Forecast receivedForecast) {
4     return Uni.createFrom().item(receivedForecast)
5         .map(forecast -> {
6             String weather = getRandomWeather(forecast.getCity());
7             return new Forecast(forecast.getId(), forecast.getCity(), weather);
8         })
9         .onItem().delayIt().by(getRandomDuration());
10 }
```

# SPRING STARTER AMQP Y REACTOR RABBITMQ

```
● ● ●  
1 @Bean  
2 public Mono<Connection> reactiveRabbitMQConnection(RabbitProperties rabbitProperties) {  
3     ConnectionFactory connectionFactory = new ConnectionFactory();  
4     connectionFactory.useNio();  
5     connectionFactory.setHost(rabbitProperties.getHost());  
6     connectionFactory.setPort(rabbitProperties.getPort());  
7     connectionFactory.setUsername(rabbitProperties.getUsername());  
8     connectionFactory.setPassword(rabbitProperties.getPassword());  
9     return Mono.fromCallable(connectionFactory::newConnection).cache();  
10 }
```

# SPRING STARTER AMQP Y REACTOR RABBITMQ

```
● ● ●

1 @Bean
2 public Sender sender(Mono<Connection> reactiveRabbitMQConnection) {
3     return RabbitFlux.createSender(new SenderOptions().connectionMono(reactiveRabbitMQConnection));
4 }
5
6 @Bean
7 public Receiver receiver(Mono<Connection> reactiveRabbitMQConnection) {
8     return RabbitFlux.createReceiver(new ReceiverOptions().connectionMono(reactiveRabbitMQConnection));
9 }
10
11 @Bean
12 public Flux<Delivery> processForecastListener(final Receiver receiver, Binding processForecastBinding) {
13     return receiver.consumeAutoAck(processForecastBinding.getDestination());
14 }
15
16 @Bean
17 public Flux<Delivery> processedForecastListener(final Receiver receiver, Binding processedForecastBinding) {
18     return receiver.consumeAutoAck(processedForecastBinding.getDestination());
19 }
```

# WEBSOCKETS

Reactivo

# QUARKUS REACTIVE MESSAGING HTTP

```
●●●

1 @Incoming("incoming-process-user")
2 @Outgoing("outgoing-process-user")
3 public Uni<User> processNewUser(User newUser) {
4     return Uni.createFrom().item(newUser)
5     .map(user -> {
6         String username = getUsername(user);
7         return new User(user.getId(), user.getName(), username);
8     })
9     .invoke(user -> System.out.println("Processed user: " + user));
10 }
11
12 @Incoming("connect")
13 public Uni<Void> connectUser(User user) {
14     return Uni.createFrom().item(user)
15     .onItem().invoke(roomResource::addUser)
16     .onItem().ignore().andContinueWithNull();
17 }
18
19 @Incoming("disconnect")
20 public Uni<Void> disconnectUser(User user) {
21     return Uni.createFrom().item(user)
22     .onItem().invoke(roomResource::deleteUser)
23     .onItem().ignore().andContinueWithNull();
24 }
```

# SPRING WEBFLUX



```
1 @Bean
2 public HandlerMapping webSocketHandlerMapping(WebSocketHandler webSocketHandler) {
3     Map<String, WebSocketHandler> map = Map.of("/room", webSocketHandler);
4     SimpleUrlHandlerMapping handlerMapping = new SimpleUrlHandlerMapping();
5     handlerMapping.setOrder(10);
6     handlerMapping.setUrlMap(map);
7     return handlerMapping;
8 }
```



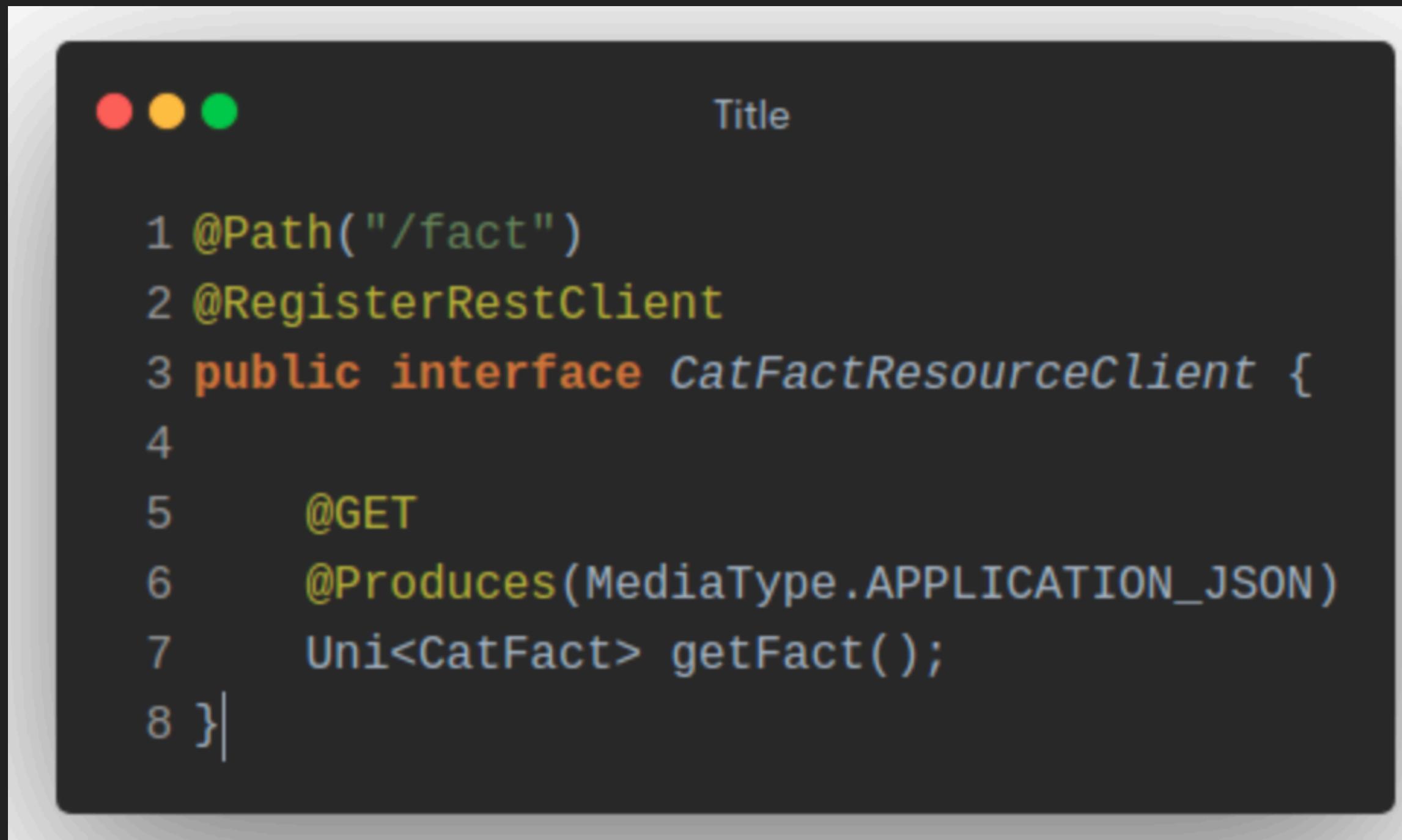
```
1 @Component
2 public class ReactiveWebSocketHandler implements WebSocketHandler {
3
4     ...
5
6     @Override
7     public Mono<Void> handle(WebSocketSession session) {
8
9         return session.receive()
10            .map(WebSocketMessage::getPayload)
11            ...
12            .then();
13     }
}
```

# CLIENTE REST HTTP

Reactiv

# QUARKUS MICROPROFILE REACTIVE CLIENT

```
es.urjc.CatFactResourceClient/mp-rest/url=https://catfact.ninja
```



The screenshot shows a code editor window with a dark theme. In the top right corner, there are three colored circular icons: red, yellow, and green. To the right of these icons, the word "Title" is displayed. The main area of the code editor contains the following Java code:

```
1 @Path("/fact")
2 @RegisterRestClient
3 public interface CatFactResourceClient {
4
5     @GET
6     @Produces(MediaType.APPLICATION_JSON)
7     Uni<CatFact> getFact();
8 }
```

# SPRING REACTIVE WEB CLIENT



Title

```
1 @RequestMapping("/cat-fact")
2 @RestController
3 public class CatFactController {
4
5     private final ReactiveWebClient webClient;
6
7     public CatFactController(ReactiveWebClient webClient) {
8         this.webClient = webClient;
9     }
10
11    @GetMapping
12    public Mono<CatFact> getFact() {
13        return webClient.getFact();
14    }
15 }
```

16



Title

```
1 @Component
2 public class ReactiveWebClient {
3
4     private final WebClient webClient;
5
6     public ReactiveWebClient(WebClient.Builder builder) {
7         this.webClient = builder.baseUrl("https://catfact.ninja").build();
8     }
9
10    public Mono<CatFact> getFact() {
11        return this.webClient.get()
12            .uri("/fact")
13            .retrieve()
14            .bodyToMono(CatFact.class);
15    }
16
17 }
```

18

# GRPC

## Reactivio

# QUARKUS

```
1 @GrpcService
2 public class GenerateRandomNumberService implements RandomNumberGenerator {
3
4     private static final Logger log = Logger.getLogger(GenerateRandomNumberService.class.getName());
5
6     @Override
7     public Uni<GenerateNumberResponse> generateNumber(GenerateNumberRequest request) {
8         log.info("[QUARKUS REACTIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getId());
9
10        int randomInt = new Random()
11            .ints(0, 30)
12            .findFirst()
13            .getAsInt();
14
15        int response = randomInt * request.getNumber();
16
17        Uni<GenerateNumberResponse> responseItem = Uni.createFrom().item(() ->
18            GenerateNumberResponse.newBuilder().setNumber(response).build()
19        );
20
21        log.info("[QUARKUS REACTIVE] - END grpc SERVICE thread id " + Thread.currentThread().getId());
22
23        return responseItem;
24    }
25 }
26
27 }
```

# SPRING

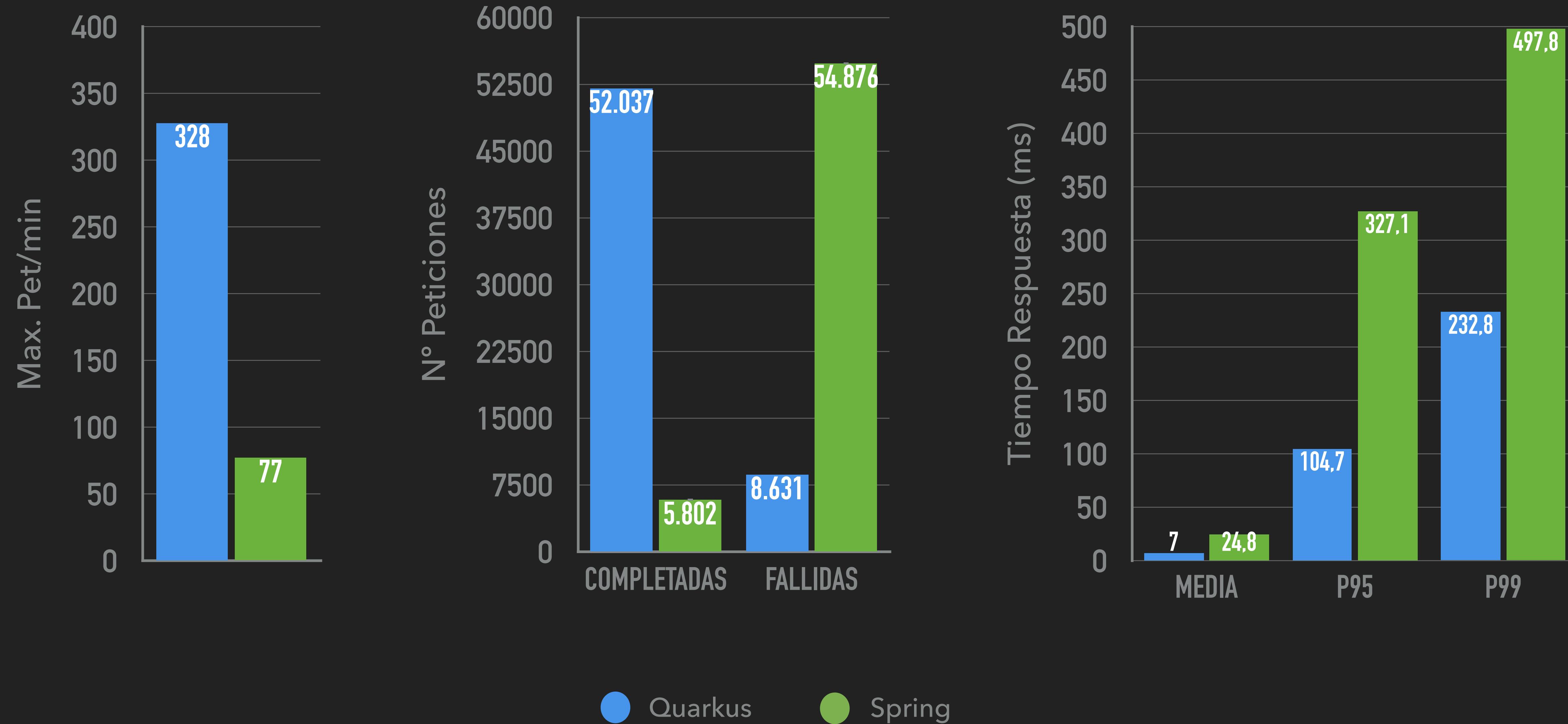
```
1 @GRpcService
2 public class GenerateRandomNumberService extends ReactorRandomNumberGeneratorGrpc.RandomNumberGeneratorImplBase {
3
4     private static final Logger log = LoggerFactory.getLogger(GenerateRandomNumberService.class);
5
6     @Override
7     public Mono<GenerateNumberResponse> generateNumber(Mono<GenerateNumberRequest> request) {
8         log.info("[SPRING REACTIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getName());
9
10        int randomInt = new Random()
11            .ints(0, 30)
12            .findFirst()
13            .getAsInt();
14
15        Mono<GenerateNumberResponse> response = request
16            .map(GenerateNumberRequest::getNumber)
17            .map(protoRequest -> GenerateNumberResponse
18                .newBuilder()
19                .setNumber(randomInt * protoRequest)
20                .build());
21
22        log.info("[SPRING REACTIVE] - END grpc SERVICE thread id " + Thread.currentThread().getName());
23
24        return response;
25    }
26 }
```

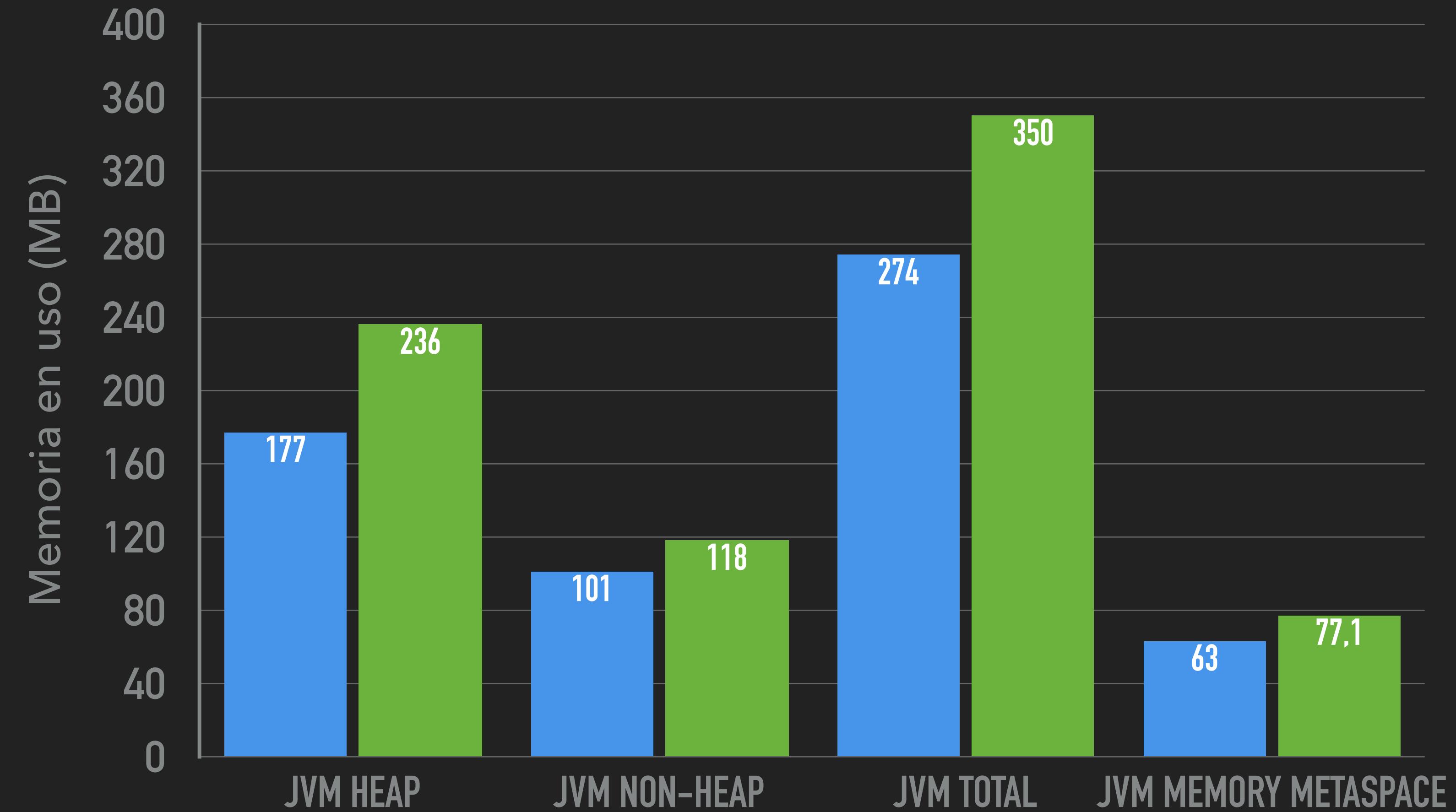
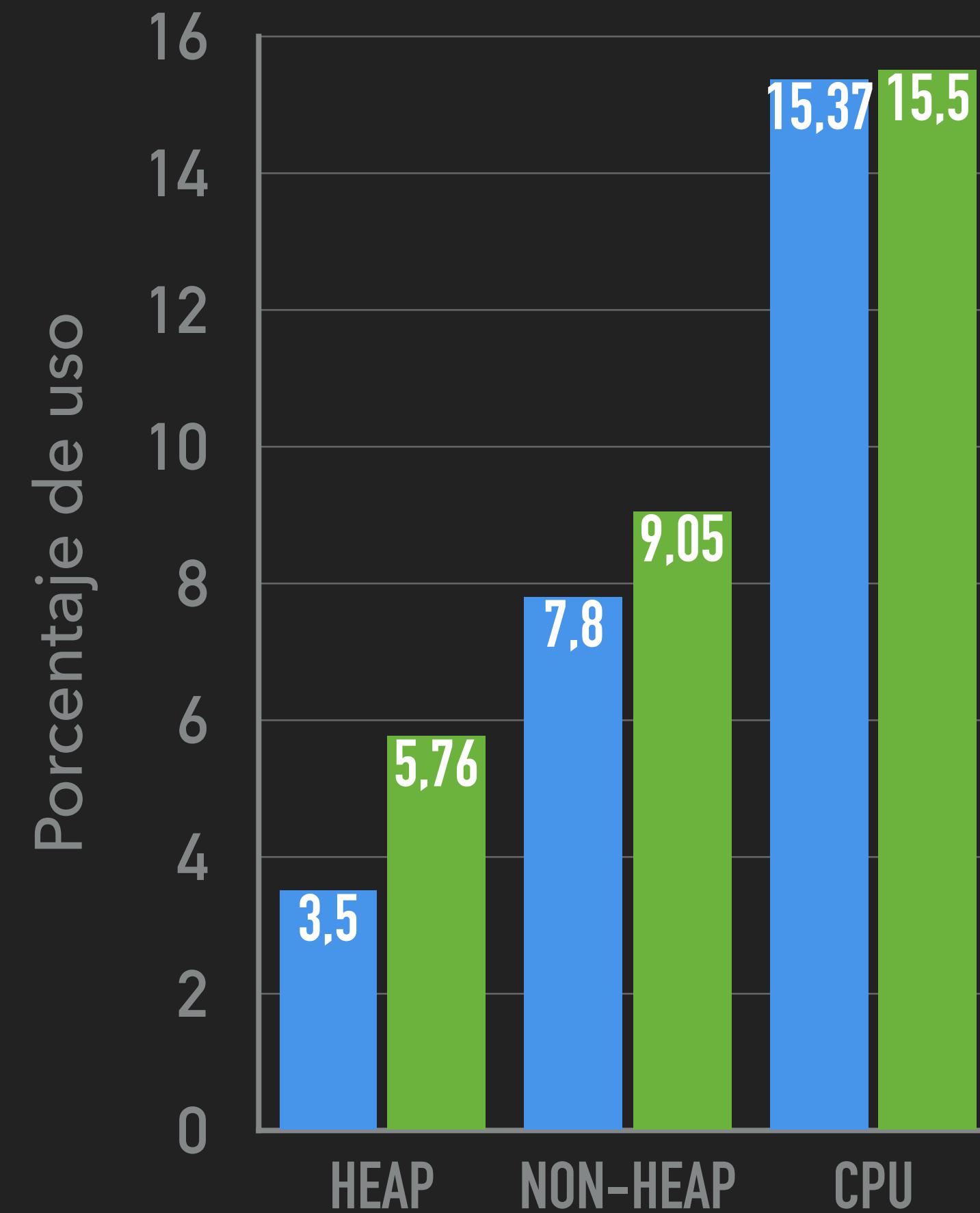


# PRUEBAS DE CARGA

# RESULTADOS DE RENDIMIENTO

Imperativo



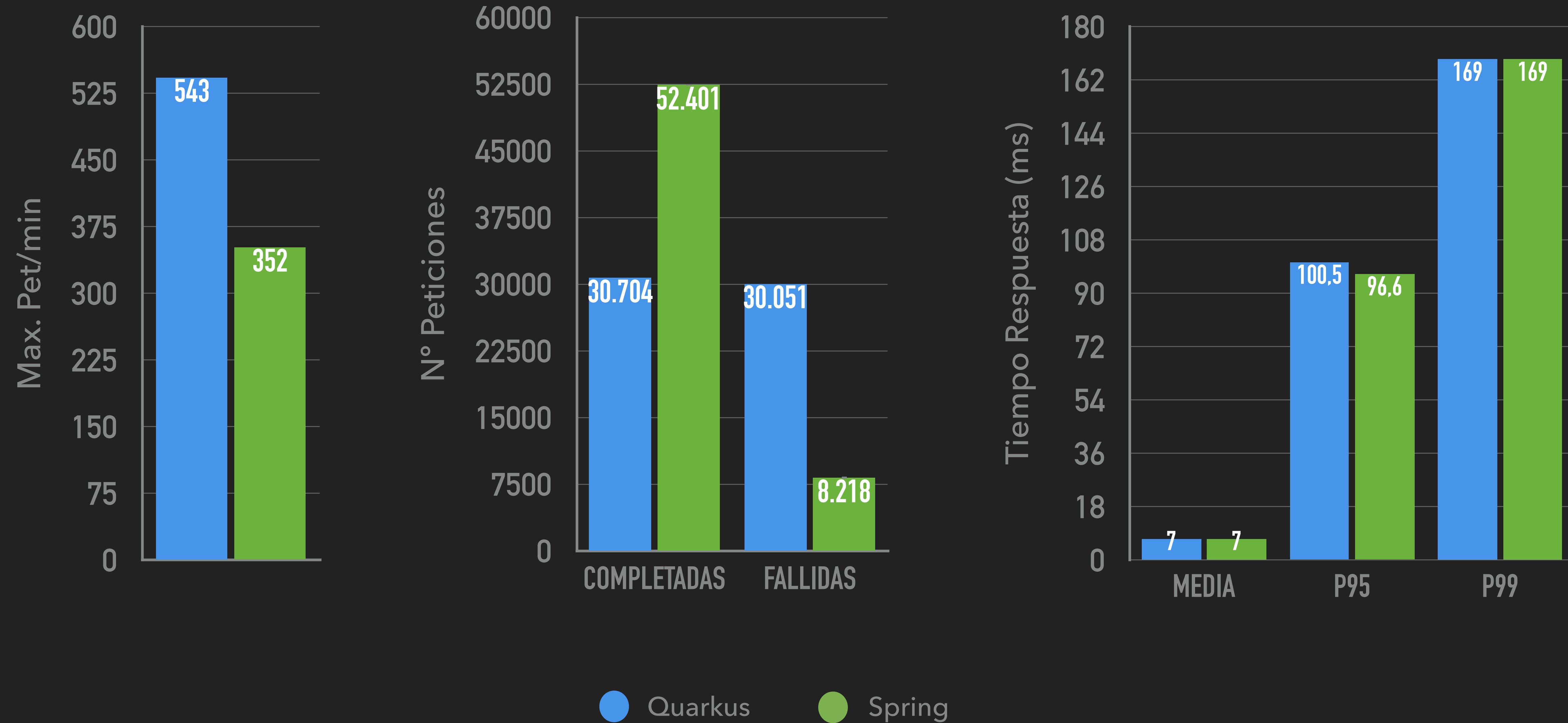


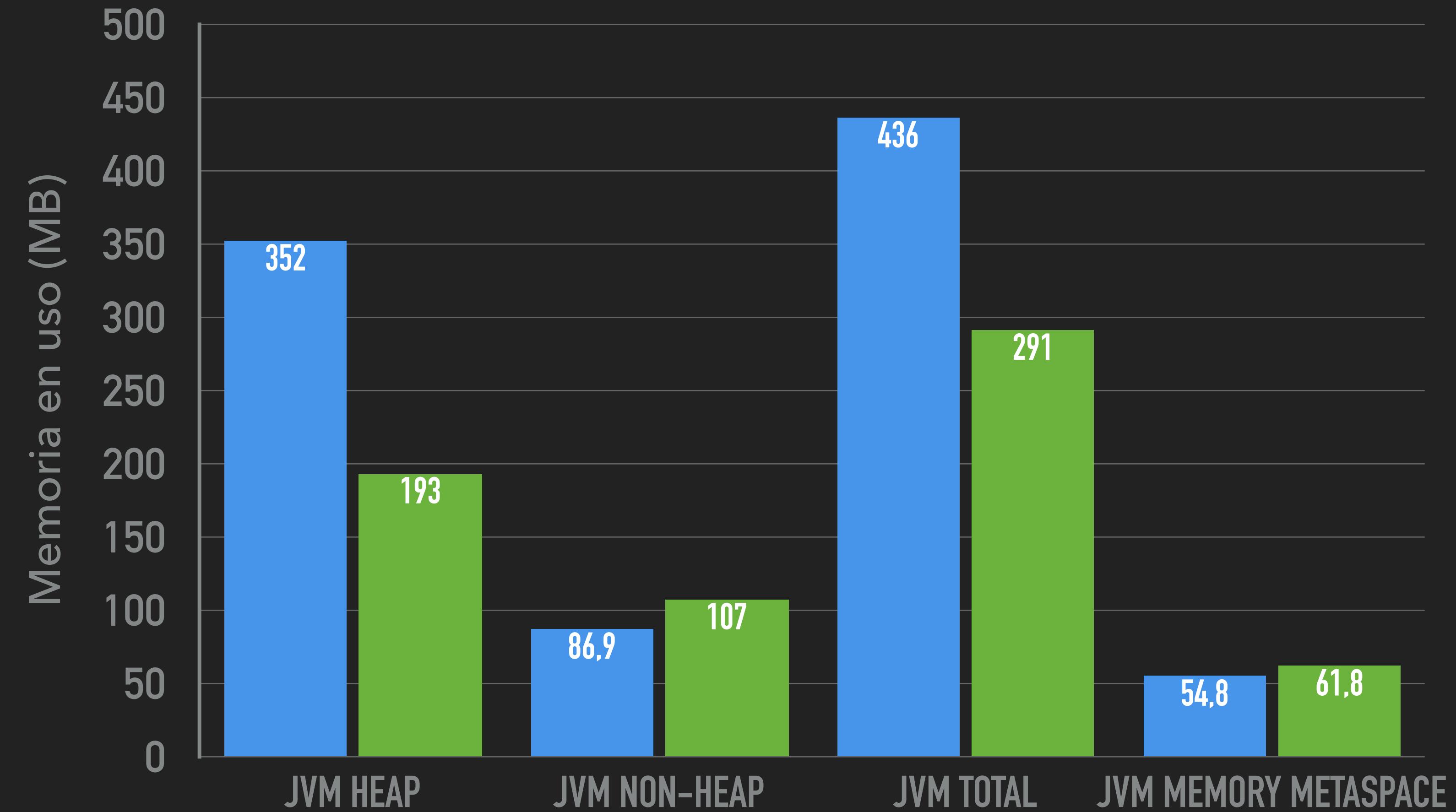
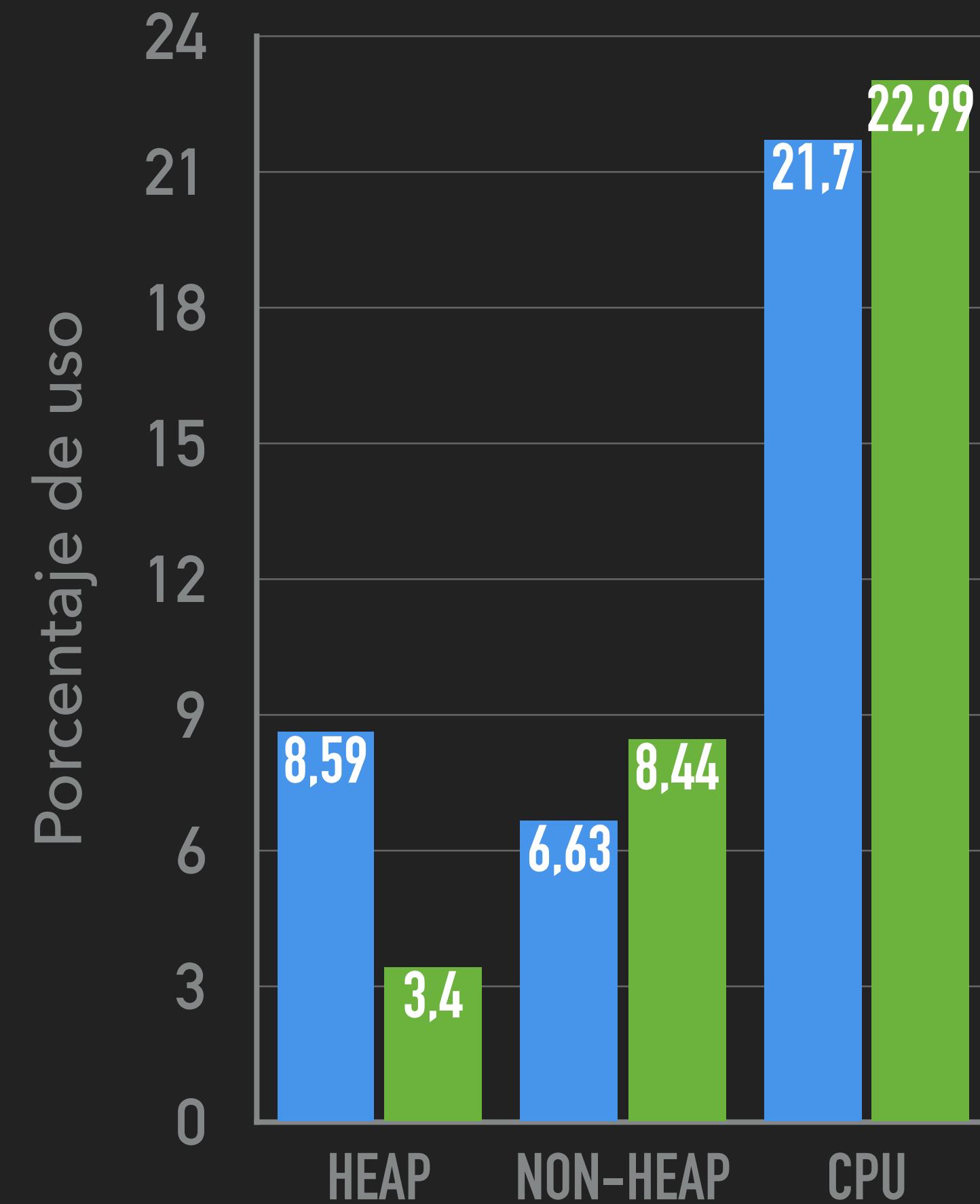
Quarkus

Spring

# RESULTADOS DE RENDIMIENTO

Reactivo





● Quarkus

● Spring



# TRABAJOS FUTUROS

▶ Realizar mas pruebas de carga:

- Sobre otros sistemas operativos y arquitecturas
- Sobre el resto de tecnologías y analizar su comportamiento
- Usando otras herramientas como k6s 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