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

Universidad
Rey Juan Carlos

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

INDICE

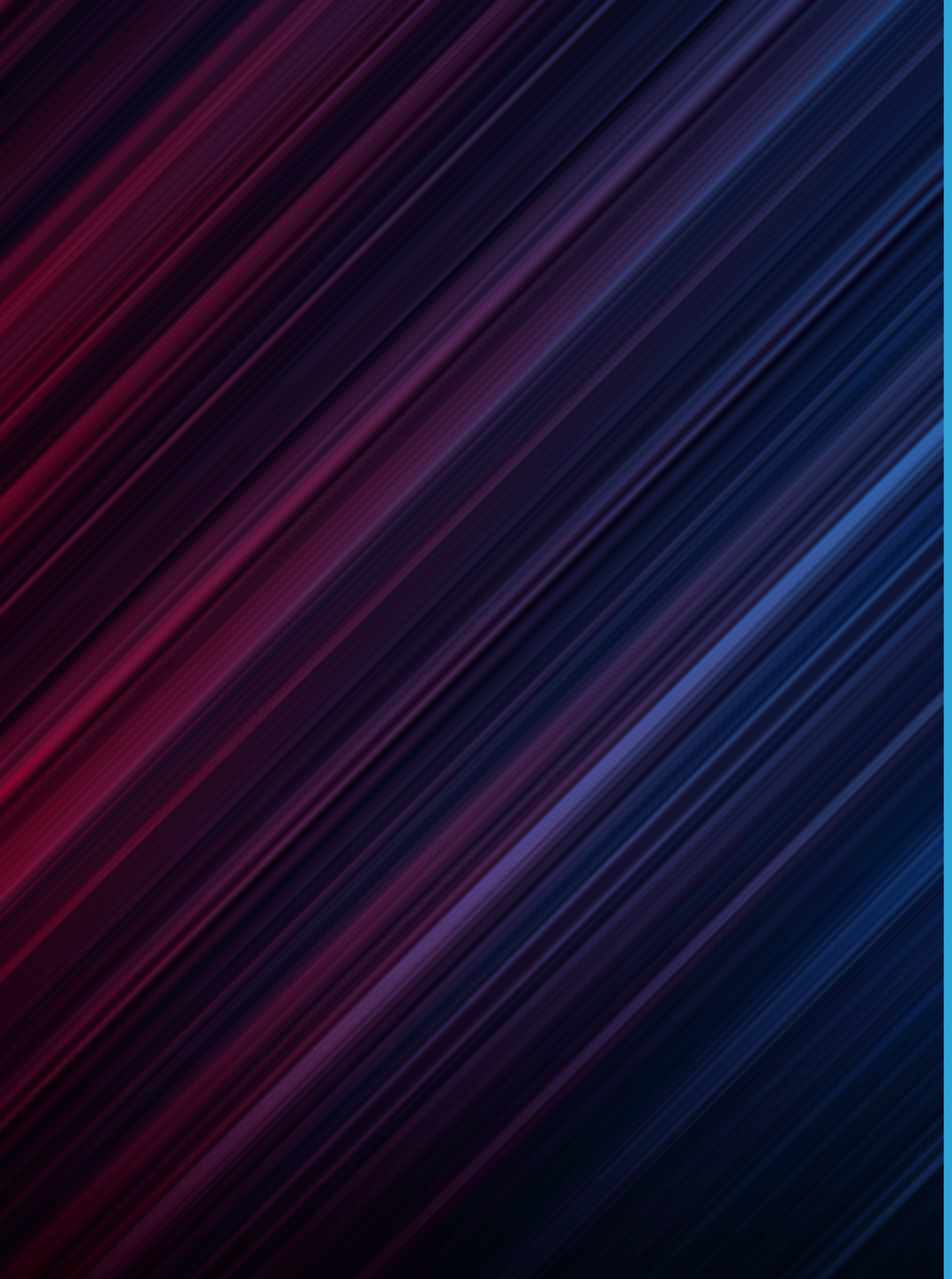
INTRODUCCIÓN Y OBJETIVOS

¿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

- 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 explicita con la que hacer pipelines mas 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

```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);
```

WEBSOCKETS

Imperativo

QUARKUS WEBSOCKETS

```java
@ServerEndpoint("/chat/{username}")
@ApplicationScoped
public class ChatApplication {

    @OnOpen
    public void onOpen(Session session, @PathParam("username") String username) {
        ...
    }

    @OnClose
    public void onClose(Session session, @PathParam("username") String username) {
        ...
    }

    @OnError
    public void onError(Session session, @PathParam("username") String username, Throwable throwable) {
       ...
    }

    @OnMessage
    public void onMessage(Session session, String message, @PathParam("username") String username) {
        ...
    }

```

SPRING WEBSOCKET

```java
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/chat").withSockJS();
    }

}
```

```java
@Controller
public class ChatApplication {

    @MessageMapping("/connect/{username}")
    @SendTo("/topic/messages")
    public String onConnect(@DestinationVariable("username") String username,
                            StompHeaderAccessor stompHeaderAccessor) {
        ...
    }

    @MessageMapping("/message")
    @SendTo("/topic/messages")
    public String onMessage(StompHeaderAccessor stompHeaderAccessor, String message) {
        ...
    }

    @MessageMapping("/disconnect")
    @SendTo("/topic/messages")
    public String onDisconnect(SimpMessageHeaderAccessor headerAccessor) {
        ...
    }
}
```

SPRING WEBSOCKET

```
var connect = function () {
  ...
  var socket = new SockJS('/chat');
  stompClient = Stomp.over(socket);

  stompClient.connect({}, function (frame) {

      stompClient.send("/app/connect/" + name);
      ...
      stompClient.subscribe('/topic/messages', function (message) {
        ...
      });
    });
  }
};

var sendMessage = function () {
    ...
    stompClient.send("/app/message", {}, value);
    ...
  }
};

var disconnect = function () {
  stompClient.send("/app/disconnect");
  stompClient.disconnect();
};
```

CLIENTE REST HTTP

Imperativo

QUARKUS MICROPROFILE CLIENT

```java
@Path("/cat-fact")
@RequestScoped
public class CatFactResource {

    @Inject
    @RestClient
    CatFactResourceClient catFactResourceClient;

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public CatFact getCatFact() {
        return this.catFactResourceClient.getFact();
    }
```

```java
@Path("/fact")
@RegisterRestClient
public interface CatFactResourceClient {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    CatFact getFact();
}
```

SPRING REST TEMPLATE CLIENT

```java
@RequestMapping("/cat-fact")
@RestController
public class CatFactController {

    @Value("${cat-fact.rest.url}")
    String catFactUrl;

    private final RestTemplate restTemplate;

    public CatFactController(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    @GetMapping
    public CatFact getFact() {
        return restTemplate.getForObject(catFactUrl, CatFact.class);
    }
}
```

Title

```java
@SpringBootApplication
public class ImperativeSpringRestClientApplication {

    public static void main(String[] args) {
        SpringApplication.run(ImperativeSpringRestClientApplication.class, args);
    }

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }

}
```

GRPC

Imperativo

QUARKUS

Title

```java
@GrpcService
public class GenerateRandomNumberService extends RandomNumberGeneratorGrpc.RandomNumberGeneratorImplBase {

    private static final Logger log = Logger.getLogger(GenerateRandomNumberService.class.getName());

    @Override
    @Blocking
    public void generateNumber(GenerateNumberRequest request, StreamObserver<GenerateNumberResponse> responseObserver) {

        log.info("[QUARKUS IMPERATIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getId());

        int randomInt = new Random()
                .ints(0, 30)
                .findFirst()
                .getAsInt();

        int response = randomInt * request.getNumber();

        responseObserver.onNext(GenerateNumberResponse.newBuilder().setNumber(response).build());
        responseObserver.onCompleted();

        log.info("[QUARKUS IMPERATIVE] - END grpc SERVICE thread id " + Thread.currentThread().getId());

    }
```

SPRING

Title

```java
@GRpcService
public class GenerateRandomNumberService extends RandomNumberGeneratorGrpc.RandomNumberGeneratorImplBase {

    private static final Logger log = LoggerFactory.getLogger(GenerateRandomNumberService.class);

    @Override
    public void generateNumber(GenerateNumberRequest request, StreamObserver<GenerateNumberResponse> responseObserver) {
        log.info("[SPRING IMPERATIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getName());

        int randomInt = new Random()
                .ints(0, 30)
                .findFirst()
                .getAsInt();

        int response = randomInt * request.getNumber();

        responseObserver.onNext(GenerateNumberResponse.newBuilder().setNumber(response).build());
        responseObserver.onCompleted();

        log.info("[SPRING IMPERATIVE] - END grpc SERVICE thread id " + Thread.currentThread().getName());

    }
}
```

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());
}
```

WEBSOCKETS

Reactivo

QUARKUS REACTIVE MESSAGING HTTP

```java
@Incoming("incoming-process-user")
@Outgoing("outgoing-process-user")
public Uni<User> processNewUser(User newUser) {
  return Uni.createFrom().item(newUser)
    .map(user -> {
      String username = getUsername(user);
      return new User(user.getId(), user.getName(), username);
    })
    .invoke(user -> System.out.println("Processed user: " + user));
}

@Incoming("connect")
public Uni<Void> connectUser(User user) {
  return Uni.createFrom().item(user)
    .onItem().invoke(roomResource::addUser)
    .onItem().ignore().andContinueWithNull();
}

@Incoming("disconnect")
public Uni<Void> disconnectUser(User user) {
  return Uni.createFrom().item(user)
    .onItem().invoke(roomResource::deleteUser)
    .onItem().ignore().andContinueWithNull();
}
```

SPRING WEBFLUX

```java
@Bean
public HandlerMapping webSocketHandlerMapping(WebSocketHandler webSocketHandler) {
  Map<String, WebSocketHandler> map = Map.of("/room", webSocketHandler);
  SimpleUrlHandlerMapping handlerMapping = new SimpleUrlHandlerMapping();
  handlerMapping.setOrder(10);
  handlerMapping.setUrlMap(map);
  return handlerMapping;
}
```

```java
@Component
public class ReactiveWebSocketHandler implements WebSocketHandler {

    ...

    @Override
    public Mono<Void> handle(WebSocketSession session) {

        return session.receive()
                .map(WebSocketMessage::getPayload)
                ...
                .then();
    }
```

CLIENTE REST HTTP

Reactivo

QUARKUS MICROPROFILE REACTIVE CLIENT

Title

```java
@Path("/cat-fact")
@RequestScoped
public class CatFactResource {

    @RestClient
    CatFactResourceClient catFactResourceClient;

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Uni<CatFact> getCatFact() {
        return this.catFactResourceClient.getFact();
    }
}
```

Title

```java
@Path("/fact")
@RegisterRestClient
public interface CatFactResourceClient {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    Uni<CatFact> getFact();
}
```

SPRING REACTIVE WEB CLIENT

Title

```java
@RequestMapping("/cat-fact")
@RestController
public class CatFactController {

    private final ReactiveWebClient webClient;

    public CatFactController(ReactiveWebClient webClient) {
        this.webClient = webClient;
    }

    @GetMapping
    public Mono<CatFact> getFact() {
        return webClient.getFact();
    }
}

```

Title

```java
@Component
public class ReactiveWebClient {

    private final WebClient webClient;

    public ReactiveWebClient(WebClient.Builder builder) {
        this.webClient = builder.baseUrl("https://catfact.ninja").build();
    }

    public Mono<CatFact> getFact() {
        return this.webClient.get()
                .uri("/fact")
                .retrieve()
                .bodyToMono(CatFact.class);
    }

}

```

GRPC

Reactivo

QUARKUS

Title

```java
@GrpcService
public class GenerateRandomNumberService implements RandomNumberGenerator {

    private static final Logger log = Logger.getLogger(GenerateRandomNumberService.class.getName());

    @Override
    public Uni<GenerateNumberResponse> generateNumber(GenerateNumberRequest request) {
        log.info("[QUARKUS REACTIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getId());

        int randomInt = new Random()
                .ints(0, 30)
                .findFirst()
                .getAsInt();

        int response = randomInt * request.getNumber();

        Uni<GenerateNumberResponse> responseItem = Uni.createFrom().item(() ->
                GenerateNumberResponse.newBuilder().setNumber(response).build()
        );

        log.info("[QUARKUS REACTIVE] - END grpc SERVICE thread id " + Thread.currentThread().getId());

        return responseItem;
    }
}

}
```

SPRING

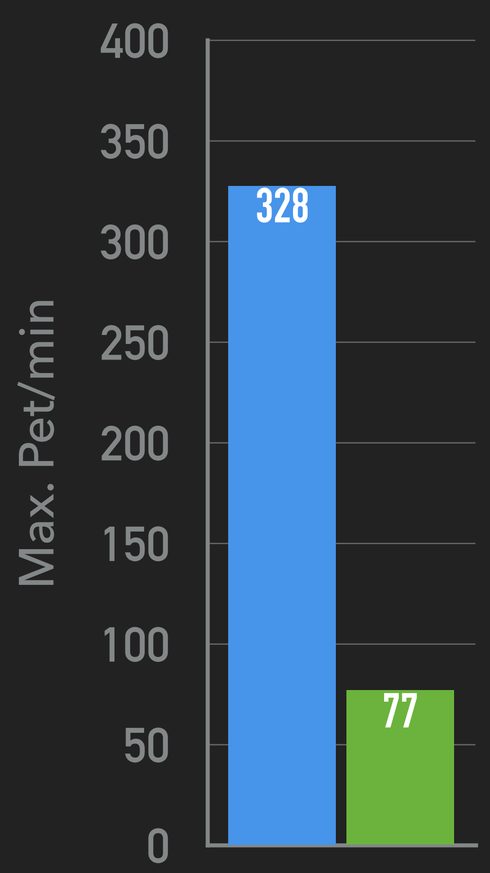
Title

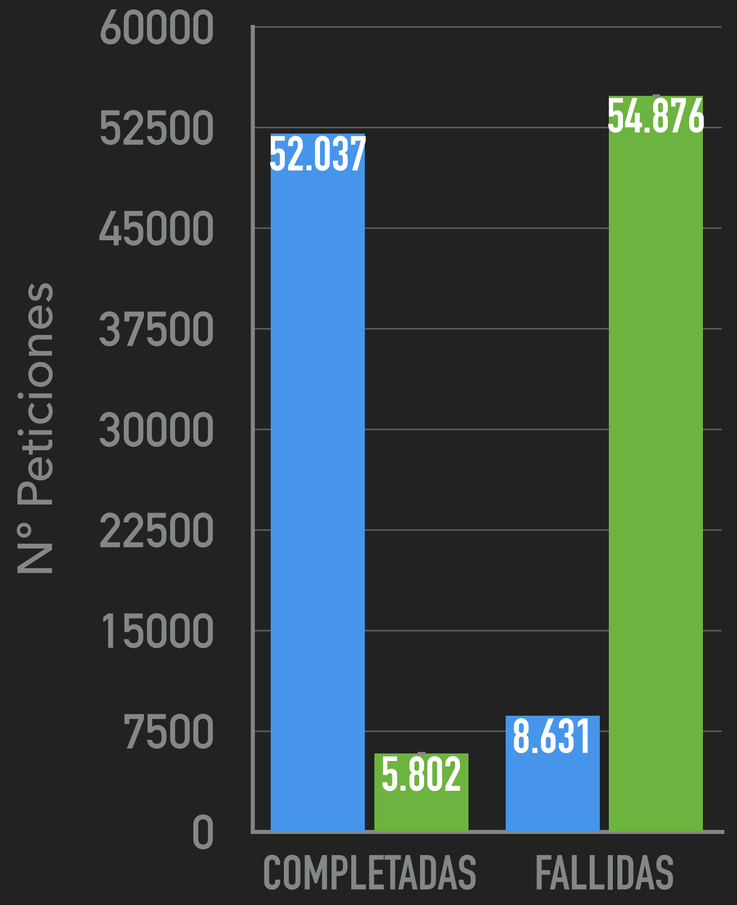
```java
@GRpcService
public class GenerateRandomNumberService extends ReactorRandomNumberGeneratorGrpc.RandomNumberGeneratorImplBase {

    private static final Logger log = LoggerFactory.getLogger(GenerateRandomNumberService.class);

    @Override
    public Mono<GenerateNumberResponse> generateNumber(Mono<GenerateNumberRequest> request) {
        log.info("[SPRING REACTIVE] - INIT grpc SERVICE thread id " + Thread.currentThread().getName());

        int randomInt = new Random()
                .ints(0, 30)
                .findFirst()
                .getAsInt();

        Mono<GenerateNumberResponse> response = request
                .map(GenerateNumberRequest::getNumber)
                .map(protoRequest -> GenerateNumberResponse
                        .newBuilder()
                        .setNumber(randomInt * protoRequest)
                        .build());

        log.info("[SPRING REACTIVE] - END grpc SERVICE thread id " + Thread.currentThread().getName());
        return response;
    }

}
```

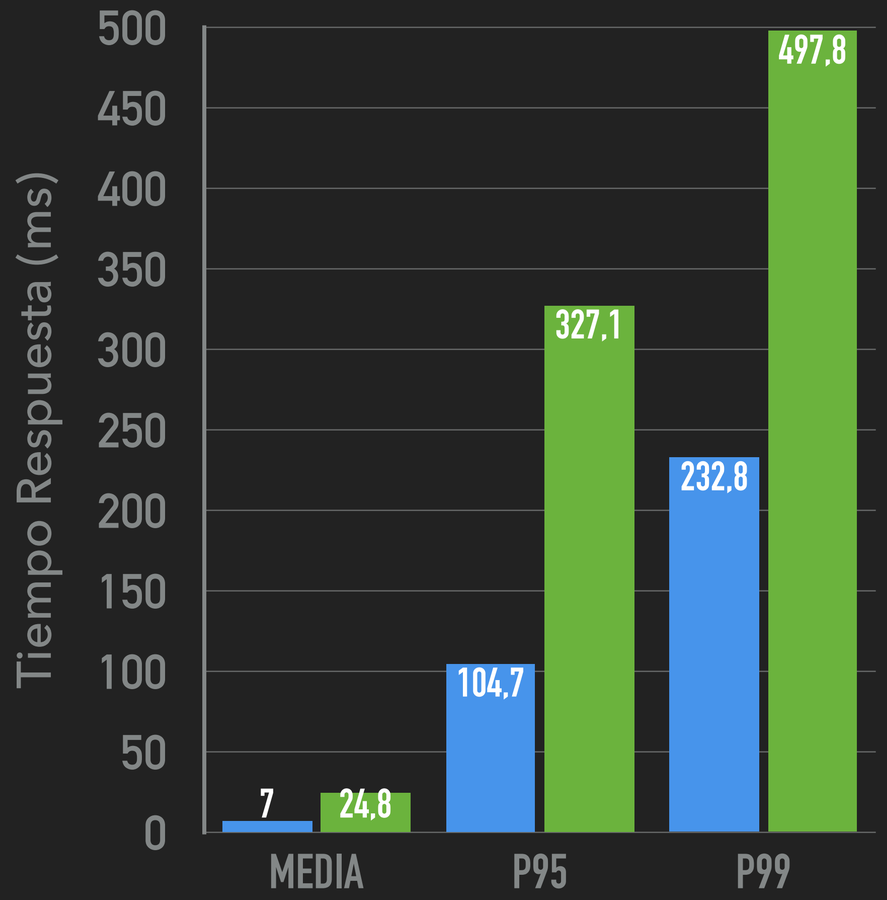
PRUEBAS DE CARGA

RESULTADOS DE RENDIMIENTO

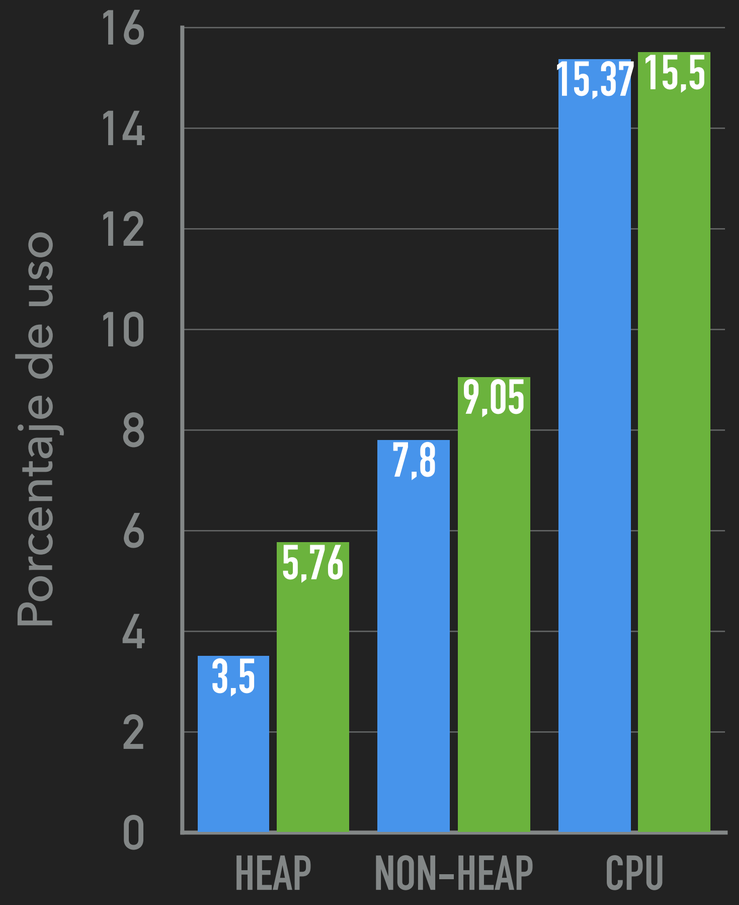
Imperativo

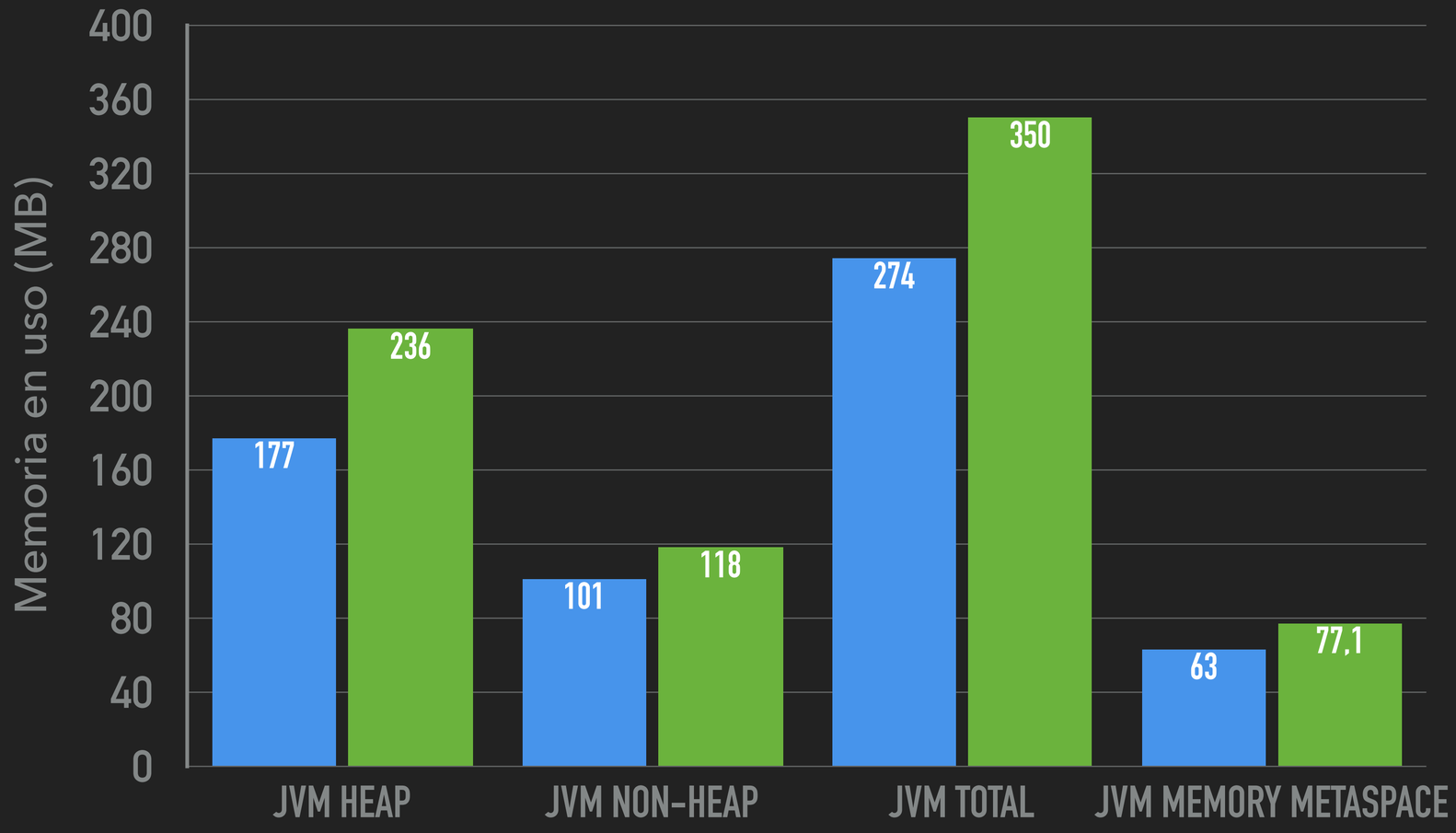






Quarkus Spring

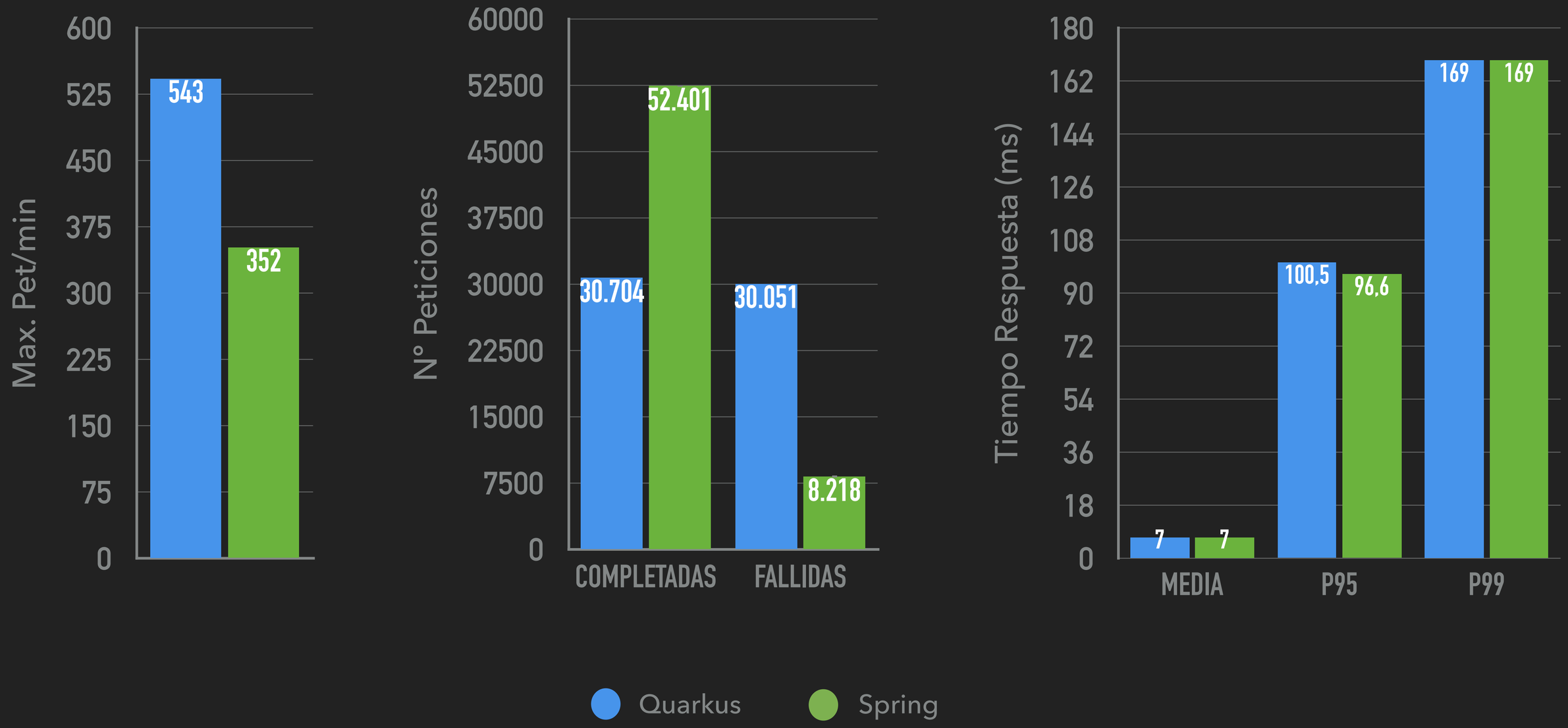


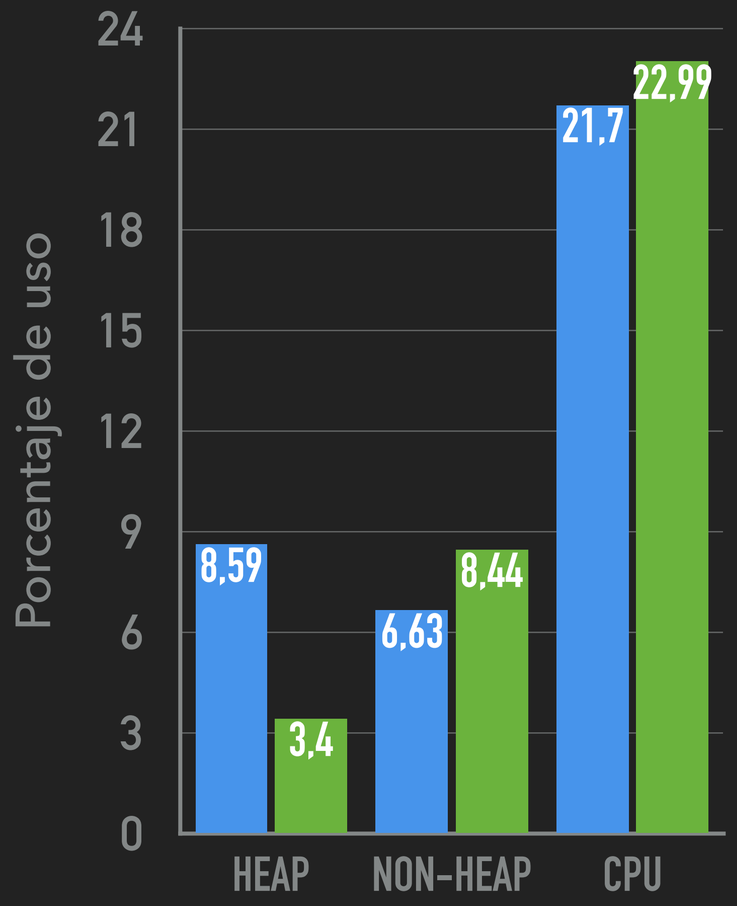


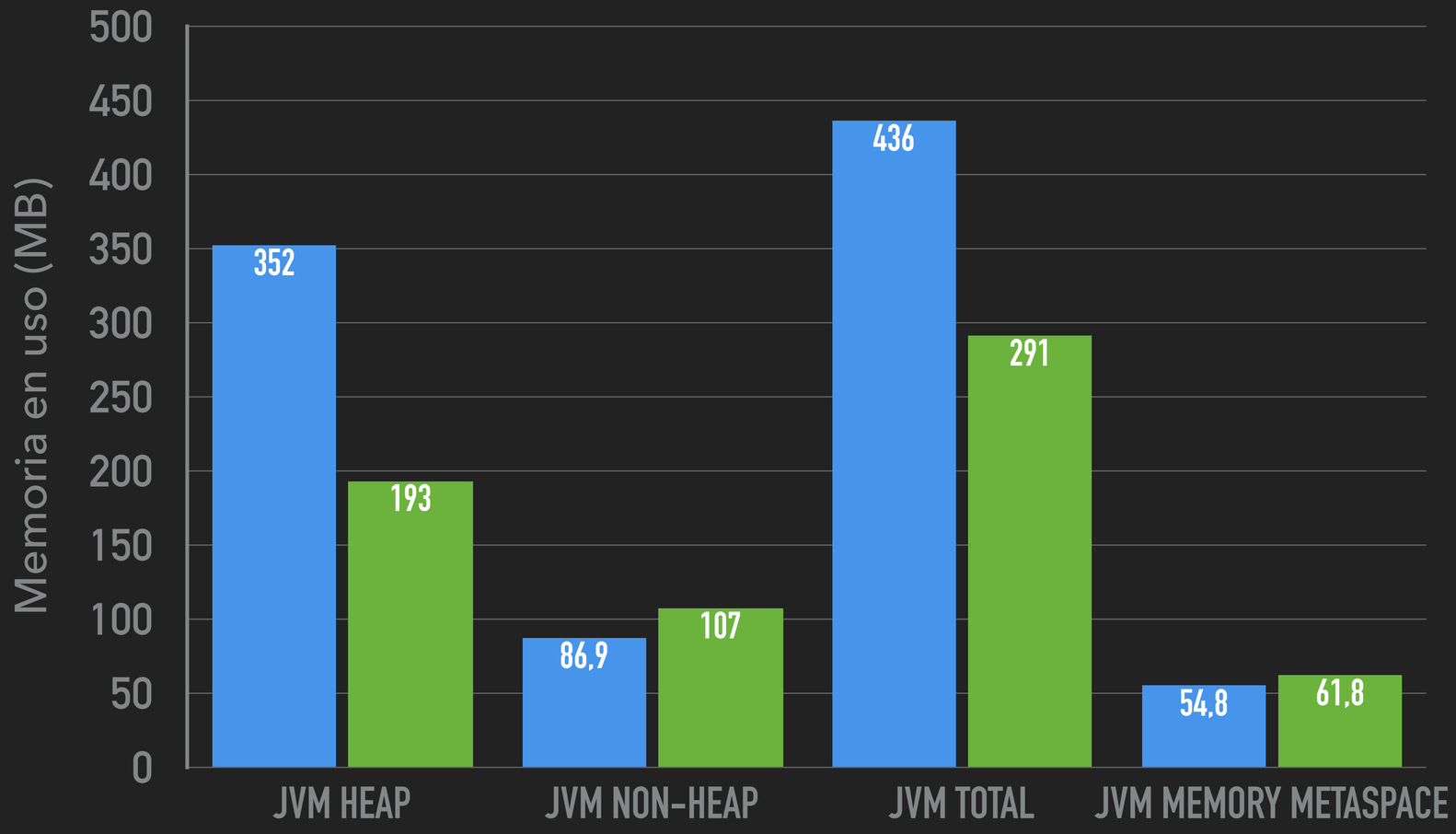
Quarkus Spring

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

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