Documentación API de Device

***Resumen*—Este documento presenta una exploración detallada del proceso de dockerización del endpoint `DeviceResource` en la plataforma de rastreo GPS Traccar. La dockerización se ha implementado para mejorar la portabilidad, escalabilidad y facilidad de despliegue del sistema, permitiendo que Traccar se ejecute de manera consistente en diversos entornos, desde servidores locales hasta la nube. El análisis incluye una revisión exhaustiva de las dependencias externas, consideraciones de seguridad, y la optimización del rendimiento en un entorno dockerizado. También se abordan los modelos de datos asociados, como `Device` y `DeviceAccumulators`, que son cruciales para las operaciones CRUD de dispositivos dentro de Traccar. Los resultados indican que la dockerización no solo simplifica el proceso de despliegue, sino que también mejora significativamente la gestión de recursos y la seguridad del sistema, preparando a Traccar para un crecimiento futuro y una integración continua más eficiente.**

I. Introducción

En el mundo actual, las aplicaciones de rastreo y monitoreo se han vuelto fundamentales para una amplia gama de industrias, desde la logística hasta la gestión de flotas y la seguridad personal. Estas aplicaciones requieren sistemas robustos que puedan manejar una gran cantidad de datos en tiempo real y ofrecer un rendimiento confiable y escalable. Traccar es uno de los sistemas de rastreo GPS más populares y de código abierto, que proporciona una solución integral para el seguimiento de dispositivos en tiempo real. Desarrollado en Java, Traccar es utilizado por miles de organizaciones en todo el mundo debido a su flexibilidad y capacidad para integrarse con diferentes dispositivos y sistemas de monitoreo [1].

Una parte crucial de la infraestructura de Traccar son las APIs (Interfaces de Programación de Aplicaciones), que permiten la comunicación entre el servidor de Traccar y otros sistemas o dispositivos. Las APIs facilitan la interacción entre diferentes componentes del software, permitiendo que se envíen y reciban datos de manera eficiente. En el contexto de Traccar, las APIs son esenciales para la gestión de dispositivos, la recuperación de datos de localización, la configuración de alertas y muchas otras funciones [2].

Las APIs funcionan como un intermediario que permite a diferentes programas hablar entre sí, sin necesidad de conocer los detalles internos de cómo están implementados esos programas. Esto se logra mediante la exposición de un conjunto de métodos o funciones que los desarrolladores pueden utilizar para interactuar con el software de manera consistente y segura. En el caso de Traccar, las APIs RESTful permiten a los desarrolladores interactuar con el sistema a través de peticiones HTTP, proporcionando una manera sencilla y flexible de acceder a las funcionalidades del sistema [3].

Traccar es una plataforma de rastreo GPS de código abierto que soporta más de 1,800 modelos de dispositivos GPS, proporcionando una solución universal para la gestión y monitoreo de vehículos y activos. La plataforma incluye un servidor backend que se comunica con los dispositivos GPS para recibir y procesar datos de localización. Estos datos son almacenados en una base de datos y pueden ser accedidos a través de una interfaz web o mediante APIs. La arquitectura de Traccar es modular, lo que permite a los usuarios personalizar y extender la funcionalidad según sus necesidades [4].

Dockerización se refiere al proceso de empaquetar una aplicación junto con todas sus dependencias en un contenedor Docker, lo que garantiza que la aplicación se ejecute de manera consistente en cualquier entorno. Docker es una herramienta de contenedorización que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones dentro de contenedores aislados del sistema operativo. La dockerización es particularmente útil en entornos de producción donde se necesita asegurar que la aplicación se ejecute de manera consistente, independientemente de las diferencias en las configuraciones del entorno [5].

El objetivo de dockerizar la arquitectura de Traccar es mejorar la portabilidad, escalabilidad y facilidad de despliegue del sistema. Al empaquetar Traccar en contenedores Docker, es posible garantizar que todas las dependencias necesarias para su funcionamiento estén incluidas y que el sistema pueda ser desplegado rápidamente en cualquier entorno, ya sea localmente, en un servidor remoto o en la nube. Esto no solo simplifica el proceso de instalación y configuración, sino que también permite escalar el sistema fácilmente para manejar un mayor número de dispositivos o un volumen mayor de datos. Además, la dockerización facilita la integración continua y el despliegue continuo (CI/CD), lo que permite a los equipos de desarrollo implementar nuevas funcionalidades o actualizaciones con mayor frecuencia y menos riesgo [6].

En este documento, se examinará cómo dockerizar el endpoint `DeviceResource` en Traccar, un componente clave en la arquitectura de Traccar que maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) relacionadas con los dispositivos. Se analizarán las dependencias del sistema, las consideraciones de seguridad, y se presentará un enfoque paso

a paso para dockerizar este componente utilizando Docker y Docker Compose.

II. Objetivos

Objetivo General

Desarrollar una documentación completa y detallada de la API de Device que permita a los desarrolladores comprender y utilizar de manera eficiente las funcionalidades y servicios ofrecidos por la API en sus aplicaciones o sistemas.

Objetivos Específicos

- Desarrollar una documentación completa y detallada de la API de Device que permita a los desarrolladores comprender y utilizar de manera eficiente las funcionalidades y servicios ofrecidos por la API en sus aplicaciones o sistemas.
- Definir los parámetros de entrada y salida de cada endpoint de la API, especificando los tipos de datos, valores permitidos, y posibles respuestas del servidor para garantizar la correcta integración por parte de los desarrolladores.
- Actualizar y mantener la documentación de la API de manera continua, asegurando que cualquier cambio o nueva funcionalidad sea reflejada en la documentación de forma precisa y oportuna.

III. Materiales y Métodos

III-A. Materiales

III-A1. Entorno de Trabajo:

- **Equipo Físico:**
  - **Nombre del dispositivo:** debian.
  - **Procesador:** Mesa Intel HD Graphics 2000 (SNB GT1)
  - **Almacenamiento:** 500 GB.
  - **RAM instalada:** 4.0 GiB.
  - **Sistema Operativo:** debian 12 (BookWorm)

III-A2. Herramientas de Software:

- **Overleaf:** Herramienta de redacción colaborativa en línea utilizada para la elaboración del informe y documentación técnica. Funciona como un editor LaTeX, permitiendo la edición simultánea por todos los miembros del equipo.
- **Docker:** Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará. [**?**]
- **Visual Studio Code:** Visual Studio Code es un editor de código fuente versátil y extensible, utilizado para desarrollar y depurar aplicaciones en una amplia variedad de lenguajes de programación. Ofrece integración con control de versiones, herramientas de desarrollo, y soporte para múltiples extensiones, facilitando la colaboración y el flujo de trabajo en proyectos de software.

III-A3. Acceso a Internet:

- Esencial para la búsqueda bibliográfica, consulta de recursos en línea, y acceso a bases de datos académicas.

III-A4. Documentación Proporcionada por el Docente:

- **Plantillas:** Plantilla estructurada para la presentación del documento final.
- **Guías:** Instrucciones detalladas para asegurar el cumplimiento de los requisitos específicos de la tarea.

IV. Resultado

IV-A. Composición del Endpoint 'DeviceResource

El `DeviceResource` es un recurso REST en Traccar que maneja las operaciones CRUD relacionadas con los dispositivos.

Código Completo

```
    /*
 * Copyright 2015 - 2024 Anton Tananaev (
    anton@traccar.org)
 *
 * Licensed under the Apache License,
    Version 2.0 (the "License");
 * you may not use this file except in
    compliance with the License.
 * You may obtain a copy of the License
    at
 *
 *     http://www.apache.org/licenses/
    LICENSE-2.0
 *
 * Unless required by applicable law or
    agreed to in writing, software
 * distributed under the License is
    distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF
    ANY KIND, either express or implied.
 * See the License for the specific
    language governing permissions and
 * limitations under the License.
 */
package org.traccar.api.resource;

import jakarta.ws.rs.FormParam;
import org.traccar.api.BaseObjectResource
    ;
import org.traccar.api.signature.
    TokenManager;
import org.traccar.broadcast.
    BroadcastService;
import org.traccar.config.Config;
import org.traccar.config.Keys;
import org.traccar.database.MediaManager;
import org.traccar.helper.LogAction;
import org.traccar.model.Device;
import org.traccar.model.
    DeviceAccumulators;
```

```
import org.traccar.model.Permission;
import org.traccar.model.Position;
import org.traccar.model.User;
import org.traccar.session.
   ConnectionManager;
import org.traccar.session.cache.
   CacheManager;
import org.traccar.storage.
   StorageException;
import org.traccar.storage.query.Columns;
import org.traccar.storage.query.
   Condition;
import org.traccar.storage.query.Request;

import jakarta.inject.Inject;
import jakarta.ws.rs.Consumes;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.HeaderParam;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.PUT;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.PathParam;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.HttpHeaders;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.security.
   GeneralSecurityException;
import java.util.Collection;
import java.util.Date;
import java.util.LinkedList;
import java.util.List;

@Path("devices")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class DeviceResource extends
   BaseObjectResource<Device> {

    private static final int
       DEFAULT_BUFFER_SIZE = 8192;
    private static final int
       IMAGE_SIZE_LIMIT = 500000;

    @Inject
    private Config config;

    @Inject
    private CacheManager cacheManager;

    @Inject
    private ConnectionManager
       connectionManager;

    @Inject
    private BroadcastService
       broadcastService;

    @Inject
    private MediaManager mediaManager;

    @Inject
    private TokenManager tokenManager;

    public DeviceResource() {
        super(Device.class);
    }

    @GET
    public Collection<Device> get(
            @QueryParam("all") boolean
               all, @QueryParam("userId")
                long userId,
            @QueryParam("uniqueId") List<
               String> uniqueIds,
            @QueryParam("id") List<Long>
               deviceIds) throws
               StorageException {

        if (!uniqueIds.isEmpty() || !
           deviceIds.isEmpty()) {

            List<Device> result = new
               LinkedList<>();
            for (String uniqueId :
               uniqueIds) {
                result.addAll(storage.
                   getObjects(Device.
                   class, new Request(
                        new Columns.All()
                           ,
                        new Condition.And
                           (
                             new
                                Condition
                                .
                                Equals
                                ("
                                uniqueId
                                ",
                                uniqueId
                                ),
                             new
                                Condition
                                .
                                Permission
                                (User.
                                class,
```

```
                                getUserId
                                (),
                                Device
                                .class
                                )))));
        }
        for (Long deviceId :
            deviceIds) {
            result.addAll(storage.
                getObjects(Device.
                class, new Request(
                    new Columns.All()
                        ,
                    new Condition.And
                        (
                            new
                                Condition
                                .
                                Equals
                                ("id",

                                deviceId
                                ),
                            new
                                Condition
                                .
                                Permission
                                (User.
                                class,

                                getUserId
                                (),
                                Device
                                .class
                                )))));
        }
        return result;

    } else {

        var conditions = new
            LinkedList<Condition>();

        if (all) {
            if (permissionsService.
                notAdmin(getUserId()))
                {
                conditions.add(new
                    Condition.
                    Permission(User.
                    class, getUserId()
                    , baseClass));
            }
        } else {
            if (userId == 0) {
                conditions.add(new
                    Condition.
                    Permission(User.
                    class, getUserId()
                    , baseClass));
            } else {
                permissionsService.
                    checkUser(
                    getUserId(),
                    userId);
                conditions.add(new
                    Condition.
                    Permission(User.
                    class, userId,
                    baseClass).
                    excludeGroups());
            }
        }

        return storage.getObjects(
            baseClass, new Request(new
                Columns.All(), Condition.
            merge(conditions)));

    }
}

@Path("{id}/accumulators")
@PUT
public Response updateAccumulators(
    DeviceAccumulators entity) throws
    Exception {
    permissionsService.
        checkPermission(Device.class,
        getUserId(), entity.
        getDeviceId());
    permissionsService.checkEdit(
        getUserId(), Device.class,
        false, false);

    Position position = storage.
        getObject(Position.class, new
        Request(
            new Columns.All(), new
                Condition.
                LatestPositions(entity
                .getDeviceId())));
    if (position != null) {
        if (entity.getTotalDistance()
            != null) {
            position.getAttributes().
                put(Position.
                KEY_TOTAL_DISTANCE,
                entity.
                getTotalDistance());
        }
```

```
            if (entity.getHours() != null
                ) {
                position.getAttributes().
                    put(Position.KEY_HOURS
                    , entity.getHours());
            }
            position.setId(storage.
                addObject(position, new
                Request(new Columns.
                Exclude("id"))));

            Device device = new Device();
            device.setId(position.
                getDeviceId());
            device.setPositionId(position
                .getId());
            storage.updateObject(device,
                new Request(
                    new Columns.Include("
                        positionId"),
                    new Condition.Equals
                        ("id", device.
                        getId())));

            var key = new Object();
            try {
                cacheManager.addDevice(
                    position.getDeviceId()
                    , key);
                cacheManager.
                    updatePosition(
                    position);
                connectionManager.
                    updatePosition(true,
                    position);
            } finally {
                cacheManager.removeDevice
                    (position.getDeviceId
                    (), key);
            }
        } else {
            throw new
                IllegalArgumentException()
                ;
        }

        LogAction.resetAccumulators(
            getUserId(), entity.
            getDeviceId());
        return Response.noContent().build
            ();
    }

    private String imageExtension(String
        type) {
        return switch (type) {
            case "image/jpeg" -> "jpg";
            case "image/png" -> "png";
            case "image/gif" -> "gif";
            case "image/webp" -> "webp";
            case "image/svg+xml" -> "svg
                ";
            default -> throw new
                IllegalArgumentException("
                Unsupported image type");
        };
    }

    @Path("{id}/image")
    @POST
    @Consumes("image/*")
    public Response uploadImage(
            @PathParam("id") long
                deviceId, File file,
            @HeaderParam(HttpHeaders.
                CONTENT_TYPE) String type)
                 throws StorageException,
                IOException {

        Device device = storage.getObject
            (Device.class, new Request(
                new Columns.All(),
                new Condition.And(
                        new Condition.
                            Equals("id",
                            deviceId),
                        new Condition.
                            Permission(
                            User.class,
                            getUserId(),
                            Device.class))
                        ));
        if (device != null) {
            String name = "device";
            String extension =
                imageExtension(type);
            try (var input = new
                FileInputStream(file);
                     var output =
                        mediaManager.
                        createFileStream(
                        device.getUniqueId
                        (), name,
                        extension)) {

                long transferred = 0;
                byte[] buffer = new byte[
                    DEFAULT_BUFFER_SIZE];
                int read;
                while ((read = input.read
                    (buffer, 0, buffer.
                    length)) >= 0) {
```

```
                    output.write(buffer,
                        0, read);
                    transferred += read;
                    if (transferred >
                        IMAGE_SIZE_LIMIT)
                        {
                        throw new
                            IllegalArgumentException
                            ("Image size
                            limit exceeded
                            ");
                    }
                }
            }
            return Response.ok(name + "."
                + extension).build();
        }
        return Response.status(Response.
            Status.NOT_FOUND).build();
    }

    @Path("share")
    @Consumes(MediaType.
        APPLICATION_FORM_URLENCODED)
    @POST
    public String shareDevice(
            @FormParam("deviceId") long
                deviceId,
            @FormParam("expiration") Date
                expiration) throws
                StorageException,
                GeneralSecurityException,
                IOException {

        User user = permissionsService.
            getUser(getUserId());
        if (permissionsService.getServer
            ().getBoolean(Keys.
            DEVICE_SHARE_DISABLE.getKey())
            ) {
            throw new SecurityException("
                Sharing is disabled");
        }
        if (user.getTemporary()) {
            throw new SecurityException("
                Temporary user");
        }
        if (user.getExpirationTime() !=
            null && user.getExpirationTime
            ().before(expiration)) {
            expiration = user.
                getExpirationTime();
        }

        Device device = storage.getObject
            (Device.class, new Request(
                new Columns.All(),
                new Condition.And(
                        new Condition.
                            Equals("id",
                            deviceId),
                        new Condition.
                            Permission(
                            User.class,
                            user.getId(),
                            Device.class))
                            ));

        String shareEmail = user.getEmail
            () + ":" + device.getUniqueId
            ();
        User share = storage.getObject(
            User.class, new Request(
                new Columns.All(), new
                    Condition.Equals("
                    email", shareEmail)));

        if (share == null) {
            share = new User();
            share.setName(device.getName
                ());
            share.setEmail(shareEmail);
            share.setExpirationTime(
                expiration);
            share.setTemporary(true);
            share.setReadonly(true);
            share.setLimitCommands(user.
                getLimitCommands() || !
                config.getBoolean(Keys.
                WEB_SHARE_DEVICE_COMMANDS)
                );
            share.setDisableReports(user.
                getDisableReports() || !
                config.getBoolean(Keys.
                WEB_SHARE_DEVICE_REPORTS))
                ;

            share.setId(storage.addObject
                (share, new Request(new
                Columns.Exclude("id"))));

            storage.addPermission(new
                Permission(User.class,
                share.getId(), Device.
                class, deviceId));
        }

        return tokenManager.generateToken
            (share.getId(), expiration);
    }
}
```

A continuación, se detallan las principales operaciones que maneja este endpoint:

IV-B. Operación GET para Dispositivos

Método: `GET /devices`

Descripción: Recupera dispositivos basados en filtros como `userId`, `uniqueId`, y `deviceIds`.

Fragmento de Código:

```
// M todo GET para obtener dispositivos
@GET
public Collection<Device> get(
        @QueryParam("all") boolean all,
        @QueryParam("userId") long userId
            ,
        @QueryParam("uniqueId") List<
            String> uniqueIds,
        @QueryParam("id") List<Long>
            deviceIds) throws
            StorageException {

    // L gica para manejar los filtros y
        devolver la lista de dispositivos
    ...
}
```

Interacción: Realiza consultas a la base de datos y verifica permisos de usuario.

IV-C. Operación PUT para Actualización de Acumuladores

Método: `PUT /devices/{id}/accumulators`

Descripción: Actualiza los acumuladores del dispositivo, como la distancia total y las horas de operación.

Fragmento de Código:

```
// M todo PUT para actualizar
   acumuladores
@Path("{id}/accumulators")
@PUT
public Response updateAccumulators(
   DeviceAccumulators entity) throws
   Exception {
    // L gica para actualizar los
       acumuladores de un dispositivo
    ...
    return Response.noContent().build();
}
```

Interacción: Modifica registros en la base de datos, gestiona caché y conexiones activas.

IV-D. Operación POST para Carga de Imágenes

Método: `POST /devices/{id}/image`

Descripción: Permite cargar imágenes asociadas a dispositivos.

Fragmento de Código:

```
// M todo POST para cargar im genes
   asociadas a dispositivos
@Path("{id}/image")
@POST
@Consumes("image/*")
public Response uploadImage(
        @PathParam("id") long deviceId,
           File file,
        @HeaderParam(HttpHeaders.
           CONTENT_TYPE) String type)
        throws StorageException,
           IOException {

    // L gica para almacenar la imagen
    ...
    return Response.ok(name + "." +
       extension).build();
}
```

Interacción: Utiliza `MediaManager` para almacenar archivos y gestionar límites de tamaño.

IV-E. Operación POST para Compartir Dispositivos

Método: `POST /devices/share`

Descripción: Facilita el compartir dispositivos con otros usuarios mediante un token temporal.

Fragmento de Código:

```
// M todo POST para compartir
   dispositivos
@Path("share")
@Consumes(MediaType.
   APPLICATION_FORM_URLENCODED)
@POST
public String shareDevice(
        @FormParam("deviceId") long
           deviceId,
        @FormParam("expiration") Date
           expiration)
        throws StorageException,
           GeneralSecurityException,
           IOException {

    // L gica para generar un token de
       compartici n
    ...
    return tokenManager.generateToken(
       share.getId(), expiration);
}
```

Interacción: Usa `TokenManager` para generar tokens y `permissionsService` para verificar permisos.

V. Análisis para la Dockerización del Endpoint

V-A. Dependencias Externas

Para dockerizar `DeviceResource`, es crucial tener en cuenta las dependencias externas que el endpoint ne-

cesita, tales como la base de datos, `CacheManager`, `ConnectionManager`, y `MediaManager`. Cada uno de estos servicios debe estar correctamente configurado en el entorno dockerizado para garantizar su correcto funcionamiento.

V-B. Configuración del Entorno de Contenedor

Es necesario definir un `Dockerfile` adecuado que contenga todas las dependencias necesarias y configurar volúmenes y redes para asegurar la persistencia de datos y la comunicación entre contenedores. También se puede utilizar `Docker Compose` para orquestar todos los servicios necesarios.

Fragmento de Código para el Dockerfile:

```
# Base image
FROM openjdk:17-jdk-slim

# Working directory
WORKDIR /app

# Copy the application code
COPY . /app

# Install dependencies and build the
    application
RUN ./gradlew build

# Expose the necessary port
EXPOSE 8080

# Run the application
CMD ["java", "-jar", "build/libs/traccar-
    server.jar"]
```

V-C. Evaluación del Rendimiento

El rendimiento de `DeviceResource` en un entorno dockerizado dependerá de la asignación correcta de recursos como CPU y memoria. Además, se deben optimizar las operaciones de I/O para evitar cuellos de botella durante la carga de imágenes y la gestión de datos.

V-D. Seguridad en la Dockerización

Se debe prestar especial atención a la seguridad al dockerizar `DeviceResource`, asegurando que cada contenedor esté aislado y que la gestión de secretos (como claves API y contraseñas) se realice de manera segura mediante herramientas como `Docker Secrets`.

V-E. Estrategia de Dockerización

Una estrategia recomendada es dockerizar `DeviceResource` como un contenedor independiente o usar `Docker Compose` para gestionar múltiples contenedores relacionados. Esto permitirá una integración fluida de servicios como la base de datos y caché, asegurando la escalabilidad y facilidad de mantenimiento del sistema.

Fragmento de Código para Docker Compose:

```
version: '3.8'

services:
  traccar:
    build: .
    ports:
      - "8080:8080"
    depends_on:
      - db
    networks:
      - traccar-net

  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_DATABASE: traccar
    networks:
      - traccar-net

networks:
  traccar-net:
```

VI. Modelo 'Device' en Traccar

VI-A. Atributos del Modelo 'Device'

Fragmento de Código:

```
/*
 * Copyright 2012 - 2023 Anton Tananaev (
    anton@traccar.org)
 *
 * Licensed under the Apache License,
    Version 2.0 (the "License");
 * you may not use this file except in
    compliance with the License.
 * You may obtain a copy of the License
    at
 *
 *     http://www.apache.org/licenses/
    LICENSE-2.0
 *
 * Unless required by applicable law or
    agreed to in writing, software
 * distributed under the License is
    distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF
    ANY KIND, either express or implied.
 * See the License for the specific
    language governing permissions and
 * limitations under the License.
 */
package org.traccar.model;

import com.fasterxml.jackson.annotation.
    JsonIgnore;
```

```
import org.traccar.storage.QueryIgnore;
import org.traccar.storage.StorageName;

import java.util.Date;

@StorageName("tc_devices")
public class Device extends GroupedModel
    implements Disableable, Schedulable {

    private long calendarId;

    @Override
    public long getCalendarId() {
        return calendarId;
    }

    @Override
    public void setCalendarId(long
        calendarId) {
        this.calendarId = calendarId;
    }

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    private String uniqueId;

    public String getUniqueId() {
        return uniqueId;
    }

    public void setUniqueId(String
        uniqueId) {
        if (uniqueId.contains("..")) {
            throw new
                IllegalArgumentException("
                Invalid unique id");
        }
        this.uniqueId = uniqueId.trim();
    }

    public static final String
        STATUS_UNKNOWN = "unknown";
    public static final String
        STATUS_ONLINE = "online";
    public static final String
        STATUS_OFFLINE = "offline";

    private String status;

    @QueryIgnore
    public String getStatus() {
        return status != null ? status :
            STATUS_OFFLINE;
    }

    public void setStatus(String status)
        {
        this.status = status != null ?
            status.trim() : null;
    }

    private Date lastUpdate;

    @QueryIgnore
    public Date getLastUpdate() {
        return this.lastUpdate;
    }

    public void setLastUpdate(Date
        lastUpdate) {
        this.lastUpdate = lastUpdate;
    }

    private long positionId;

    @QueryIgnore
    public long getPositionId() {
        return positionId;
    }

    public void setPositionId(long
        positionId) {
        this.positionId = positionId;
    }

    private String phone;

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone != null ?
            phone.trim() : null;
    }

    private String model;

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
```

```
}

private String contact;

public String getContact() {
    return contact;
}

public void setContact(String contact
    ) {
    this.contact = contact;
}

private String category;

public String getCategory() {
    return category;
}

public void setCategory(String
    category) {
    this.category = category;
}

private boolean disabled;

@Override
public boolean getDisabled() {
    return disabled;
}

@Override
public void setDisabled(boolean
    disabled) {
    this.disabled = disabled;
}

private Date expirationTime;

@Override
public Date getExpirationTime() {
    return expirationTime;
}

@Override
public void setExpirationTime(Date
    expirationTime) {
    this.expirationTime =
        expirationTime;
}

private boolean motionStreak;

@QueryIgnore
@JsonIgnore
public boolean getMotionStreak() {
    return motionStreak;
}

@JsonIgnore
public void setMotionStreak(boolean
    motionStreak) {
    this.motionStreak = motionStreak;
}

private boolean motionState;

@QueryIgnore
@JsonIgnore
public boolean getMotionState() {
    return motionState;
}

@JsonIgnore
public void setMotionState(boolean
    motionState) {
    this.motionState = motionState;
}

private Date motionTime;

@QueryIgnore
@JsonIgnore
public Date getMotionTime() {
    return motionTime;
}

@JsonIgnore
public void setMotionTime(Date
    motionTime) {
    this.motionTime = motionTime;
}

private double motionDistance;

@QueryIgnore
@JsonIgnore
public double getMotionDistance() {
    return motionDistance;
}

@JsonIgnore
public void setMotionDistance(double
    motionDistance) {
    this.motionDistance =
        motionDistance;
}

private boolean overspeedState;

@QueryIgnore
@JsonIgnore
```

```
    public boolean getOverspeedState() {
        return overspeedState;
    }

    @JsonIgnore
    public void setOverspeedState(boolean
        overspeedState) {
        this.overspeedState =
            overspeedState;
    }

    private Date overspeedTime;

    @QueryIgnore
    @JsonIgnore
    public Date getOverspeedTime() {
        return overspeedTime;
    }

    @JsonIgnore
    public void setOverspeedTime(Date
        overspeedTime) {
        this.overspeedTime =
            overspeedTime;
    }

    private long overspeedGeofenceId;

    @QueryIgnore
    @JsonIgnore
    public long getOverspeedGeofenceId()
        {
        return overspeedGeofenceId;
    }

    @JsonIgnore
    public void setOverspeedGeofenceId(
        long overspeedGeofenceId) {
        this.overspeedGeofenceId =
            overspeedGeofenceId;
    }

}
```

- **calendarId:** Identificador del calendario asociado al dispositivo.
- **name:** Nombre del dispositivo para identificación.
- **uniqueId:** Identificador único del dispositivo, validado para evitar caracteres peligrosos.
- **status:** Estado actual del dispositivo (`online`, `offline`, `unknown`).
- **lastUpdate:** Fecha y hora de la última actualización.
- **positionId:** Identificador de la última posición registrada.
- **phone:** Número de teléfono asociado, si aplica.
- **model:** Modelo del dispositivo.
- **contact:** Información de contacto administrativa.
- **category:** Categoría del dispositivo.
- **disabled:** Estado que indica si el dispositivo está deshabilitado.
- **expirationTime:** Tiempo de expiración del dispositivo.
- **Atributos de Movimiento y Velocidad:** Gestionan el estado de movimiento y exceso de velocidad.

VI-B. Relaciones y Herencia

`Device` hereda de `GroupedModel`, lo que permite que un dispositivo pertenezca a un grupo. Además, implementa las interfaces `Disableable` y `Schedulable`, lo que le permite ser deshabilitado y asociado a un calendario de eventos.

VI-C. Validaciones y Restricciones

El `uniqueId` del dispositivo es validado para evitar caracteres peligrosos, y el `status` debe ser gestionado para reflejar la conectividad del dispositivo.

VI-D. Persistencia y Consultas

El modelo `Device` está anotado con `@StorageName("tc_devices")`, indicando su almacenamiento en la tabla `tc_devices`. Además, utiliza `@QueryIgnore` y `@JsonIgnore` para optimizar consultas y proteger datos sensibles.

VI-E. Uso en 'DeviceResource'

`Device` es el núcleo de las operaciones en `DeviceResource`, manejando cada operación, desde la recuperación de dispositivos hasta la actualización de acumuladores y la carga de imágenes.

VI-F. Implicaciones para la Dockerización

La persistencia de datos del modelo `Device` es crucial, por lo que es necesario configurar volúmenes persistentes o un almacenamiento de base de datos externalizado. También es importante optimizar las consultas relacionadas con `Device` para mantener el rendimiento en un entorno dockerizado.

VII. Modelo 'DeviceAccumulators' en Traccar

VII-A. Atributos del Modelo 'DeviceAccumulators'

Fragmento de Código:

```
/*
 * Copyright 2016 - 2018 Anton Tananaev (
    anton@traccar.org)
 * Copyright 2016 - 2018 Andrey Kunitsyn
    (andrey@traccar.org)
 *
 * Licensed under the Apache License,
    Version 2.0 (the "License");
 * you may not use this file except in
    compliance with the License.
 * You may obtain a copy of the License
    at
```

```
 *
 *     http://www.apache.org/licenses/
    LICENSE-2.0
 *
 * Unless required by applicable law or
    agreed to in writing, software
 * distributed under the License is
    distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF
    ANY KIND, either express or implied.
 * See the License for the specific
    language governing permissions and
 * limitations under the License.
 */
package org.traccar.model;

public class DeviceAccumulators {

    private long deviceId;

    public long getDeviceId() {
        return deviceId;
    }

    public void setDeviceId(long deviceId
        ) {
        this.deviceId = deviceId;
    }

    private Double totalDistance;

    public Double getTotalDistance() {
        return totalDistance;
    }

    public void setTotalDistance(Double
        totalDistance) {
        this.totalDistance =
            totalDistance;
    }

    private Long hours;

    public Long getHours() {
        return hours;
    }

    public void setHours(Long hours) {
        this.hours = hours;
    }

}
```

- **deviceId:** Identificador del dispositivo asociado a los acumuladores.
- **totalDistance:** Distancia total acumulada por el dispositivo.
- **hours:** Horas de operación acumuladas por el dispositivo.

VII-B. Uso en 'DeviceResource'

`DeviceAccumulators` es un modelo crítico en `DeviceResource` que permite la actualización de acumuladores, como la distancia total y las horas de operación. Este modelo es fundamental para el seguimiento preciso del rendimiento y uso de los dispositivos.

VII-C. Implicaciones para la Dockerización

Al igual que con el modelo `Device`, es esencial asegurar la persistencia de los datos de `DeviceAccumulators` en un entorno dockerizado. Las actualizaciones y consultas relacionadas con este modelo deben ser optimizadas para mantener un rendimiento adecuado dentro del contenedor.

VIII. Discusión

La dockerización de Traccar, en particular del endpoint 'DeviceResource', representa un avance significativo en la portabilidad y escalabilidad de esta plataforma de rastreo GPS. La posibilidad de encapsular todo el entorno de ejecución de Traccar en contenedores Docker permite que los desarrolladores y operadores del sistema puedan desplegarlo de manera rápida y eficiente en diferentes entornos, ya sea en servidores locales, remotos o en la nube.

Un aspecto clave en este proceso es la correcta configuración de las dependencias externas como bases de datos, caché y manejo de medios, que son fundamentales para el funcionamiento del 'DeviceResource'. La gestión de estos componentes dentro de contenedores requiere una planificación cuidadosa para garantizar que se mantenga la integridad y el rendimiento del sistema. Esto es particularmente relevante cuando se considera la naturaleza intensiva en datos de las aplicaciones de rastreo en tiempo real, donde cualquier latencia o fallo en la gestión de datos puede impactar negativamente en el rendimiento general del sistema.

Otro punto de discusión es la seguridad en el entorno dockerizado. Docker facilita el aislamiento de aplicaciones, lo que mejora la seguridad al reducir la superficie de ataque. Sin embargo, es necesario asegurar que los secretos, como claves API y credenciales de acceso, estén adecuadamente protegidos. Esto se puede lograr mediante el uso de herramientas como Docker Secrets y la correcta configuración de permisos y roles dentro del sistema.

En términos de rendimiento, la dockerización puede ofrecer beneficios sustanciales al permitir una mejor utilización de los recursos del sistema, incluyendo la capacidad de escalar horizontalmente añadiendo más contenedores según sea necesario. Sin embargo, también plantea desafíos, como la necesidad de optimizar las operaciones de entrada/salida (I/O) para evitar cuellos de botella, especialmente en operaciones que implican la carga de imágenes o el manejo intensivo de bases de datos.

En general, la dockerización del 'DeviceResource' no solo facilita la gestión y el despliegue de Traccar, sino que también

abre la puerta a mejoras continuas en el desarrollo y la integración continua (CI/CD), lo que permite a los equipos de desarrollo implementar nuevas características de manera más rápida y con menor riesgo.

IX. Conclusiones

La implementación de Docker para el despliegue de Traccar, y específicamente del 'DeviceResource', ha demostrado ser una estrategia eficaz para mejorar la portabilidad, escalabilidad, y seguridad del sistema. A través de la dockerización, Traccar puede ser implementado en una variedad de entornos con una configuración mínima, lo que reduce significativamente el tiempo y esfuerzo necesarios para su despliegue y operación.

El uso de contenedores Docker también permite una mejor gestión de recursos y facilita la escalabilidad horizontal, lo que es crucial para aplicaciones de rastreo en tiempo real que deben manejar grandes volúmenes de datos y múltiples dispositivos simultáneamente. Además, la integración con herramientas como Docker Compose simplifica la orquestación de servicios, permitiendo una gestión más eficiente de las dependencias del sistema.

La seguridad es otro beneficio clave de la dockerización. Al encapsular los componentes del sistema en contenedores aislados, se reduce la superficie de ataque y se mejora la protección de datos sensibles. Sin embargo, es crucial seguir las mejores prácticas en la gestión de secretos y la configuración de permisos para garantizar que la seguridad no se vea comprometida.

En conclusión, la dockerización del 'DeviceResource' de Traccar es un paso significativo hacia la modernización de la plataforma, proporcionando una solución más robusta, flexible y fácil de gestionar para el rastreo GPS en tiempo real. Esta implementación no solo mejora la eficiencia operativa, sino que también prepara a Traccar para un crecimiento futuro y una integración continua más ágil.

Referencias

[1] Traccar. (2024) Gps tracking system. Accessed: 2024-09-03. [Online]. Available: https://www.traccar.org/

[2] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000, accessed: 2024-09-03. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[3] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, 2007, accessed: 2024-09-03. [Online]. Available: https://www.oreilly.com/library/view/restful-web-services/9780596529260/

[4] Traccar. (2024) Supported gps trackers. Accessed: 2024-09-03. [Online]. Available: https://www.traccar.org/supported-devices/

[5] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, 2014, accessed: 2024-09-03. [Online]. Available: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment

[6] J. Turnbull, *The Docker Book: Containerization Is the New Virtualization*. James Turnbull, 2014, accessed: 2024-09-03. [Online]. Available: https://www.dockerbook.com/