

Vehicle Monitoring

1.0.1

Vinod Kumar Jagwani

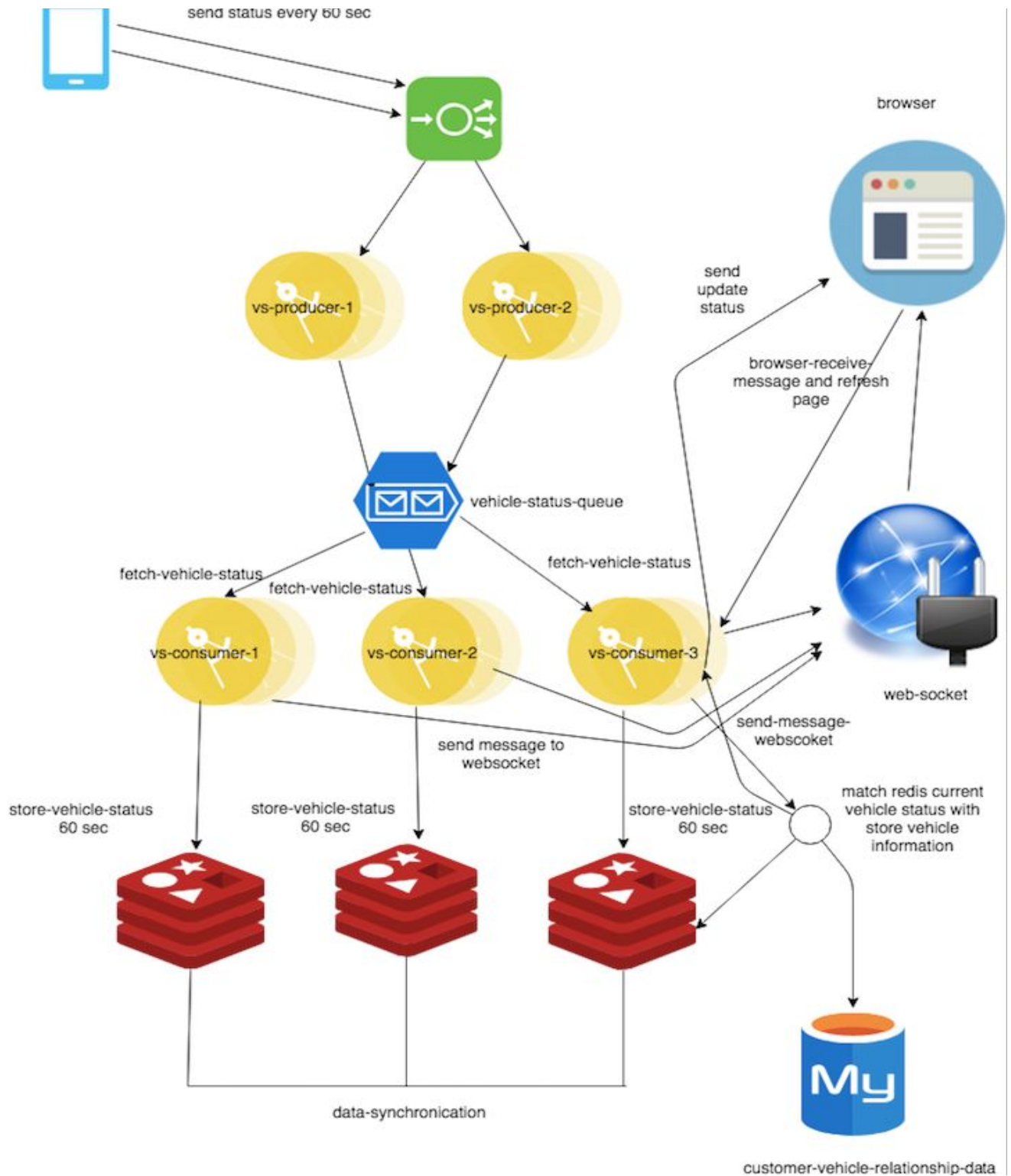
xvz challenge


Overview

Vehicle status monitoring is basically online web based solution where users are able to see their status, while every connected vehicle will send ping in every minute to the system if there is no ping, means the vehicle is not connected.

The proposed solution is based on highly scalable solution which runs under on any cloud platform without changing any code apart from configurations. The whole system is developed on Java and spring echo system.

Architecture

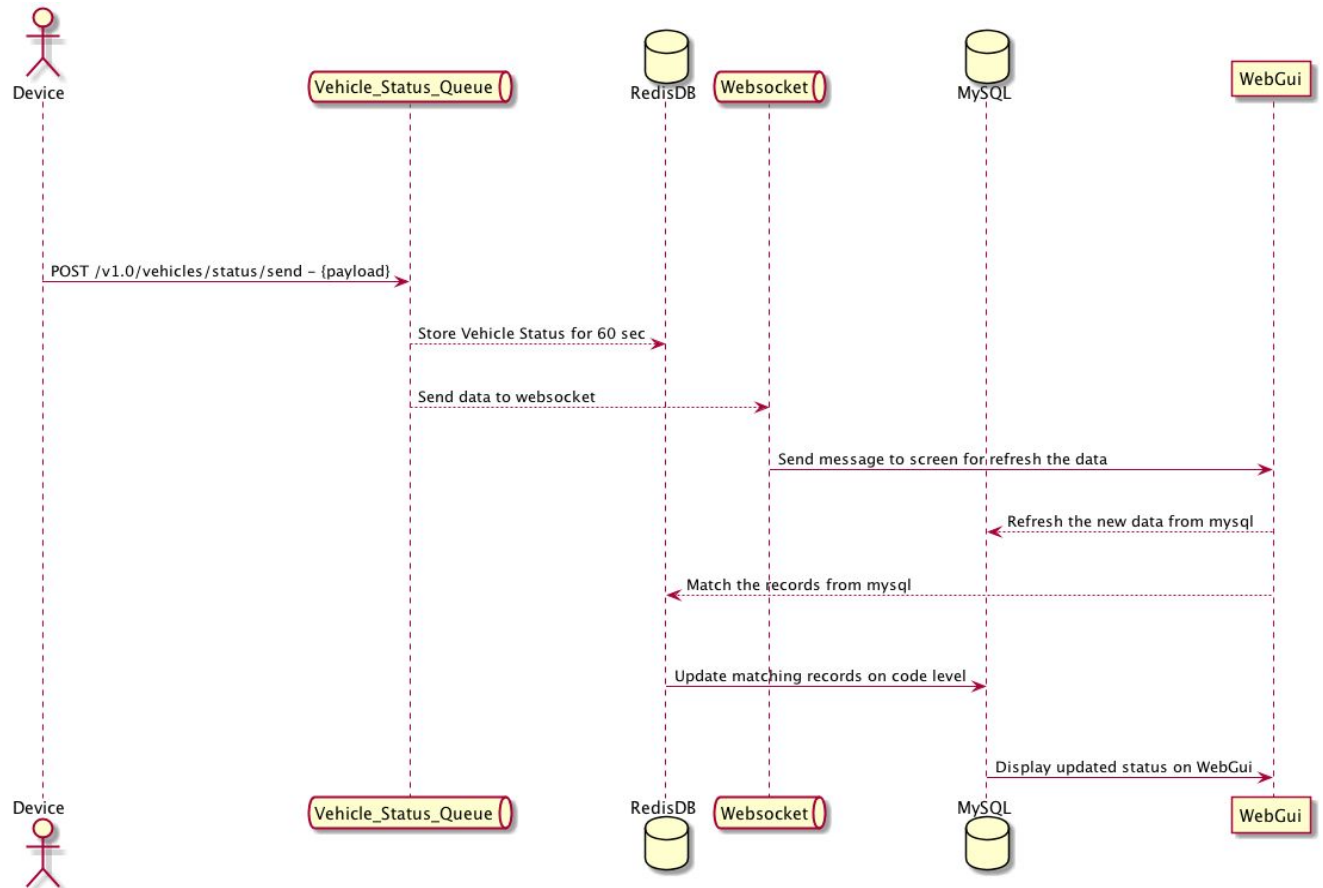




Above diagram represents the overall picture and flow of solution. The solution is designed based on consideration of highly scalable solution. Following is the explanation.

1. The producer is responsible for getting incoming request from device(device will consume rest api) which contains the information such vehicle id, customer id and the status true, so whenever device send a request it will send these 3 parameters (it's assumption parameter can be more) and then producer will forward this message to rabbitmq on the queue.
2. Producer is microservice in nature, so it can be run on cloud with multiple instances with load balancer.
3. Once the message arrives on queue, different consumers will be ready to pick message and based on round robin algorithm queue will be consumed by any one consumer and process the message.
4. Each Consumer is connected with redis db cluster and after receiving message consumer will store this message in redis db for 60 seconds and after this time it will automatically remove this message.
5. Consumer is also microservice in nature, its can also increase into multiple instances and also create a group of consumers who can connected with redis cluster.
6. Once the message store in redis db immediately afterwards it send the same message to websocket which is connected with frontend application.
7. Once message received by frontend using socketconnection it will refresh the status of vehicles on the page (just for performance purpose pagination is used).
8. Since currently frontend application connected with consumer so after refreshing of page, frontend ask the vehicle and customer data which is store in mysql and also contact with redis db, so to update those status which are already in redis db and return back to frontend.

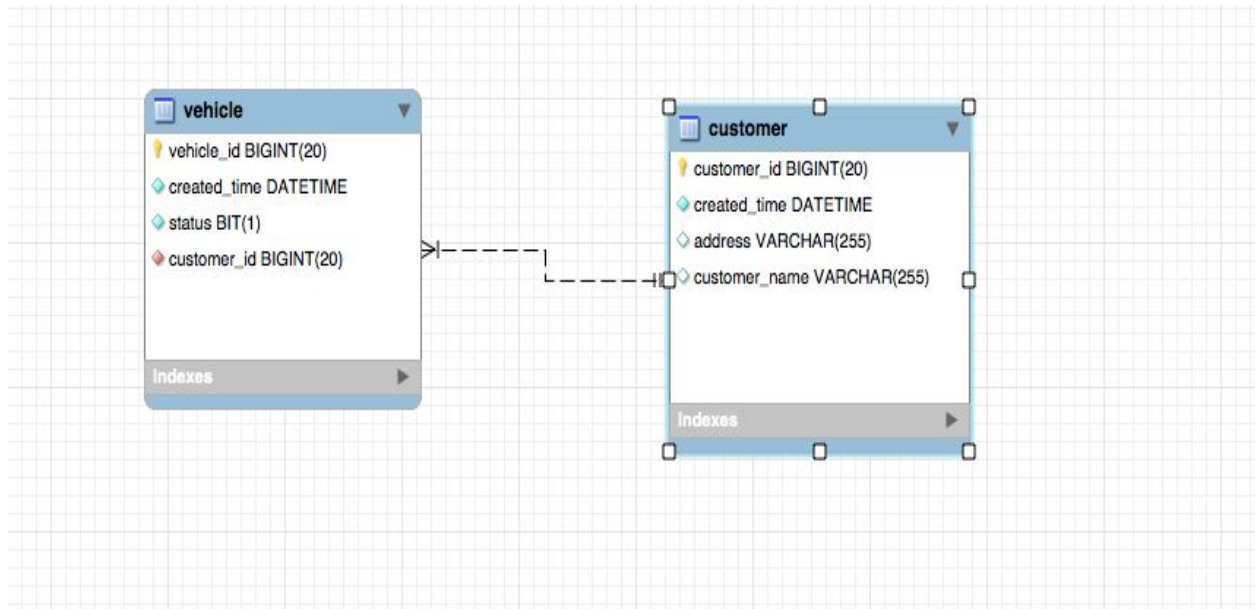
UML



Tools and Technologies

1. Java 8
2. Springboot 2.1
3. Redis DB
4. Rabbit MQ
5. MySQL
6. Websocket
7. Spring Actuator
8. Spring Logback
9. Spring Data JPA
10. SpringTest (Mockito Unit test)
11. H2 DB (Integration test)
12. Spring RestDocs
13. AngularJS (UI Data Grid)
14. Spotify Maven Docker Plugin
15. QueryDSL
16. PMD Maven plugin
17. Findbug maven plugin

Database Design (Mysql)



Currently mysql is being used for storing vehicle customer data. Before sending any status to the system vehicle must be registered in our system (assumption currently no GUI or any api for keeping record but can be provide).

So there are 2 tables customer and vehicle and customer holds multiple vehicles so its one to many relationship with vehicle. ERD diagram showing the customer id inside in vehicle table as foreign key. There is a column status which will always false during registration.

Sample data:

```
INSERT INTO customer VALUES ('1','2017-06-14 18:23:28','Cementvägen 8, 111 11 Södertälje','Kalles Grustransporter AB');
```

```
INSERT INTO vehicle VALUES ('14','2017-06-14 18:23:28','bvn',0,'1');
```

```
INSERT INTO vehicle VALUES ('13','2017-06-14 18:23:28','abcd',0,'1');
```

Database Design (Redis)

Redis DB is being used for cache the status sent by registered vehicle and the record will remain in redis for 60 sec. Redis store data in key value form, currently key will be the vehicle Id and value will be the json of fields below and it will be serialized and deserialized during the reading and writing time. This record will further process to the front end by checking with registered vehicle in mysql database.

Sample data:

```
{
  "vehicleId": 6,
  "customerId": 4,
  "status": true
}
```

Queue Design (RabbitMQ)

RabbitMQ is the message queueing server where topic and queue will be used, its highly performant server with distributed in nature, we can create multiple clusters of rabbitmq.

Every vehicle status send by device will arrive at queue (**vehicle_status**) and further it will process and pass to redis db. RabbitMQ works based on producer and consumer concept, where producer send the message and consumer will receive the message and can be defined multiple consumers for queue.

Sample data:

```
{
  "vehicleId": 6,
  "customerId": 4,
  "status": true
}
```

Code Structure

There are 2 different services

1. Vehicle-monitoring-producer
2. Vehicle-monitoring-consumer

Both services are following layered architecture:

- Repository layer is responsible for database classes and interfaces to fetch the data from database.
- Service layer where business logic is defined, it can have single responsibility
- Facade layer is defined for combining different services or additional logic which required for frontend or consumer of api
- Controller which calls the facade to bring the data and to send to the consumer.
- Configuration is responsible for defining beans related to different configurations required to run the application.

Unit Test, Integration Test

Both services have unit test which runs during the building time, these units tests using mockito library and spring test library, there is also integration test for mydb and redis db using springboot datajpa annotation running under in memory h2 database.

Services also using Spring rest documentation which force to follow to the principle of TDD, after build the services we can able to view the html based documentation which provides details of API.

Deploy on Cloud (AWS)

For deployment there is already maven spotify docker plugin which will create a docker image during the building process and also sample jenkins files is developed for creating jenkins job which will automatically create a docker image using above mentioned maven plugin and push the image to docker private registry and also push the image on AWS ec2 instance.

Deploy Local (Using Docker)

Both consumer and producer are using docker maven plugin and also there docker-compose file is provided to running services locally.

Static Code Analyser

Both services are enable with PDM and findbugs maven plugins which are responsible for static analysis both are very powerful library which defines set of rules and generates the reports. Apart the code base also tested with internal plugins of intellij idea test code coverage plugin. However sonarqube can be used for this purpose.

Serverless (AWS lambda)

Since services are based on spring ecosystem so it can be possible to make services as serverless with modification of the code and configurations to make it compatible for serverless. For aws lambda we have to provide "aws-serverless-java-container-spring" depency and also remove embed tomcat from spring deployer along with springboot maven plugin. After changing this you have to create a jar file and deploy as lambda function on aws. However when you try to deploy a fully fledged Spring app with dozens of libraries it gets a bit nasty and Lambda has a size limit, and for large apps you'll need to do deployments through S3. Also, if your app is continuously alive, it would work fine, but for scenarios where you don't have enough requests to keep it up, and response time needs to be under a second or two, it may performance problem. So making serverless purely based on the design and requirement of solution.