

ConnectiCar (Version 2) – Rahti 2 manual

University of Oulu

Information Processing Science

Bachelor Project

Tatu Niemikorpi, Jimi Vainikainen, Teemu Blomqvist, Iiro Tauriainen, and Matti Tolonen

2025

Contents

Contents 2

1. Introduction 3

2. Rahti2 4

2.1 General information 4

2.2 Services installed and accessibility 4

2.3 Services authentication 6

2.4 Fault tolerance and scalability 7

2.5 Rahti2 / Openshift CLI and CLI login 8

2.6 Eclipse Mosquitto 10

2.7 Telegraf 12

2.8 InfluxDB 13

2.9 Grafana 15

2.10 OpenStreetMap 16

3. Troubleshooting 20

# 

# Introduction

ConnectiCar V2 is a continuation project of the ConnectiCar project, commissioned by the University of Oulu. The first phase of further development has been carried out by computer engineering students from the University of Oulu. The original description of the ConnectiCar project is as follows:

"The goal of the project was to design and assemble an Onboarding Unit (OBU) that will be installed in a car to gather GPS coordinates, signal strength data, and driving metrics such as fuel level and speed. Once collected, the data is processed and transmitted to a server, where it is visualized on an interactive map."

The continuation of this project is being carried out by Information Processing Science students from the University of Oulu as part of their bachelor's project. The main goal of the project at this stage is to develop software that delivers vehicle data to the cloud.

# Rahti2

## General information

The services have been deployed in the Rahti2 environment, which is managed by CSC – IT Center for Science Ltd. The Rahti Shared Container Service is based on Kubernetes/OpenShift technology and provides an environment for hosting general-purpose container workloads.

Rahti2 service and console is accessible in address <https://rahti.csc.fi/>

## Services installed and accessibility

The following services have been installed, pre-configured and deployed in the Rahti2 environment for the Kuura project space:

* Eclipse Mosquitto (An open-source MQTT Broker)
* Influxdata Telegraf (An open-source data collection agent)
* Influxdata InfluxDB V2 (An open-source time-series database)
* Grafana Labs Grafana (An open-source analytics and visualization web application)
* OpenStreetMap (An open-source web map layer)

The YAML configuration files for all services delivered in Rahti2 have been provided to the customer.

Kuva, joka sisältää kohteen kuvakaappaus, ympyrä, diagrammi, teksti

Tekoälyn generoima sisältö voi olla virheellistä.

The services that are accessible from a specific address can be found at the following URL:

* Eclipse Mosquitto: ssl://connecticar-mqtt.2.rahtiapp.fi (port 443)
* Influxdata InfluxDB V2: <https://influxdb-connecticar.2.rahtiapp.fi>
* Grafana Labs Grafana: <https://grafana-connecticar.2.rahtiapp.fi>
* OpenStreetMap: <https://connecticar.2.rahtiapp.fi>

The following services are not accessible via a public URL and can only be accessed through the Rahti2 console:

* Influxdata Telegraf

## Services authentication

Authentication (username and password) is enabled for the following applications:

* Influxdata InfluxDB V2
* Grafana Labs Grafana

Administrator passwords are provided to the customer separately for security reasons. A username and password are required to log in to the web portal.

Certificate-based authentication is enabled for the following applications:

* Eclipse Mosquitto
* Influxdata Telegraf

Certificate files are delivered to the customer separately for security reasons. Certification is required to communicate with MQTT Broker. **The certificates are not signed by an external Certificate Authority**. They are valid until the year 2032. Use this same certificate files with your client which connects to MQTT Broker.

API token-based authentication is used in the following applications:

* Influxdata InfluxDB V2
* Grafana Labs Grafana
* OpenStreetMap

API tokens are delivered to the customer separately for security reasons. An API token is used for reading from and writing to the InfluxDB database. Telegraf is granted full write access, while Grafana and OpenStreetMap are granted read-only access.

## Fault tolerance and scalability

Currently, the services do not scale due to the intended use case

At this time, the configuration is designed to withstand typical Rahti2 maintenance operations, during which workloads may potentially be restarted. Pod images are pulled from Docker Hub, and versioning is used, which has been proven to work effectively. The :latest tag is not used to ensure that the deployed version is always explicit. InfluxDB and Grafana store their user data in their own PersistentVolumeClaims to ensure data durability and consistency.

Health check functionality (Liveness and Readiness probes) has been configured for InfluxDB and Grafana, as they both support these functions.

The Liveness probe monitors whether the containers are still functioning correctly. If it fails, the container is automatically restarted by Rahti2. For example, if the InfluxDB server is running but unable to respond to requests, the liveness probe will detect this failure and trigger a restart to restore functionality.

The Readiness probe, on the other hand, checks if the containers are ready to accept traffic. If it fails, Rahti2 will trigger a restart to restore functionality.

In a catastrophic situation where all user data in the PVC (PersistentVolumeClaim) is lost or corrupted, authentication will always occur with the administrator and password credentials provided to the client, as they are pre-set as the default passwords for the administrator in all situations.

These administrator and password credentials can, of course, be revealed in the Rahti2 console within the secrets objects.

## Rahti2 / Openshift CLI and CLI login

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti, numero

Tekoälyn generoima sisältö voi olla virheellistä.

Rahti2 can be used with a CLI tool that can be downloaded directly from the Rahti2 console, as shown in the image above. Select “Command Line Tools”. Select the appropriate operating system and follow the instructions, including setting path and environment variables for your operating system.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti, käyntikortti

Tekoälyn generoima sisältö voi olla virheellistä.

The CLI tool requires a token from the Rahti2 environment. You can obtain the login token command directly by selecting the link shown in the image above. The link will open a new Rahti2 login page for authentication. Copy-paste given token in your operating system command prompt or terminal.

For a more detailed understanding of how the Rahti2 environment works, it is recommended to refer to the Rahti2 manual at the following address: <https://docs.csc.fi/cloud/rahti/>

## Eclipse Mosquitto

Eclipse Mosquitto consists of the following objects in the Rahti2 environment

* ConfigMap (Stores the Mosquitto configuration (mosquito.conf) file.
* Deployment (Manages the lifecycle of the Mosquitto pod)
* Secrets (Stores MQTT Broker certificates)
* Service (Exposes Mosquitto pod for other objects)
* Route (Exposes Mosquitto service to external network)

Mosquitto configuration currently listens ports 1883 and 8883 but is only routed to 8883 security connections due to limitation of Rahti2.

Start by creating the Secret object first. We give name for this object “mosquito-certs”. Make sure the certs directory contains the files ca.crt, server.crt, and server.key. **These certificates are also shared with Telegraf**. **The certificates must also be the same as the ones used in the client application that sends data to the MQTT broker!**

**Secret creation commands:**

**Linux/MacOs/Gitbash:**

oc create secret generic mosquitto-certs \

--from-file=certs/server.crt \

--from-file=certs/server.key \

--from-file=certs/ca.crt

**Windows Powershell:**

oc create secret generic mosquitto-certs `

--from-file=server.crt="$(Get-Location)\certs\server.crt"

--from-file=server.key="$(Get-Location)\certs\server.key"

--from-file=ca.crt="$(Get-Location)\certs\ca.crt"

**ConfigMap creation command:**

oc create -f mosquito-configmap.yaml

**Deployment creation command:**

oc create -f mosquito-deployment.yaml

**Service creation command:**

oc create -f mosquito-service.yaml

**Route creation command:**

oc create -f mosquito-route.yaml

After creation, the MQTT Broker service should be running.

## Telegraf

Telegraf consists of the following objects in the Rahti2 environment

* ConfigMap (Stores the configuration file for Telegraf
* Deployment (Manages the lifecycle of the Telegraf pod)

Inside the Telegraf ConfigMap YAML file, the configuration is stored in the data section. This part can be modified, for example the interval can be changed. Right now, Telegraf is parsing incoming data using the configuration. If incoming data format changes, the parsing section needs to be modified. Configuration input (and deployment volume) references the certs from the Mosquitto, make sure the certs Secret is deployed. Configuration output uses variables from the Deployment file, these include INFLUX\_URL, INFLUX\_TOKEN, INFLUX\_ORG & INFLUX\_BUCKET. These variables reference the InfluxDB set-up ConfigMap, so make sure it is deployed as well.

**ConfigMap creation command:**

oc create -f telegraf-configmap.yaml

**Deployment creation command:**

oc create -f telegraf.yaml

After this, Telegraf should be deployed

## InfluxDB

InfluxDB consists of the following objects in the Rahti2 environment

* ConfigMap (Stores normal setup parameters for InfluxDB)
* Secret (Stores confidential setup parameters, username and password for InfluxDB. They are BASE64 decoded in the YAML file)
* Service (Exposes InfluxDB pod for other objects)
* Route (Exposes InfluxDB service to external network)
* PersistentVolumeClaim (Stores InfluxDB data)
* PersistentVolumeClaim (Stores InfluxDB setup config)
* Deployment (Manages the lifecycle of the InfluxDB pod)

The InfluxDB data PVC and setup config PVC are on the same YAML file but create different PVC, they can’t be stored inside the same PVC. They can be in separate YAML files but for simplicity’s sake they are combined. The retention policy of InfluxDB Graphical UI is currently infinite, it can be changed if the client wants to change it to be a definite time.

**ConfigMap creation command:**

oc create -f influxdb-config.yaml

**Secret creation command:**

oc create -f influxdb-secret.yaml

**Service creation command:**

oc create -f influxdb-service.yaml

**Route creation command:**

oc create -f influxdb-route.yaml

**PersistentVolumeClaim creation command (both)**

oc create -f influxdb-pvc.yaml

**Deployment creation command:**

oc create -f influxdb.yaml

After this, InfluxDB should be deployed.

## Grafana

Grafana consists of the following objects in Rahti2 environment:

* Secret (Stores confidential setup parameters such as username and password)
* PersistentVolumeClaim (Save Grafana data between restarts)
* Service (Exposes Grafana pod for other objects)
* Route (Exposes Grafana service to external network)
* Deployment (Manages the lifecycle of the Grafana pod)

Grafana dashboards and settings are saved permanently via PVC.

**Secret creation command:**

oc create -f grafana-secret.yaml

**PersistentVolumeClaim creation command:**

oc create -f grafana-pvc.yaml

**Service creation command:**

oc create -f grafana-service.yaml

**Route creation command:**

oc create -f grafana-route.yaml

**Deployment creation command:**

oc create -f grafana-deployment.yaml

After this, Grafana should be deployed.

## OpenStreetMap

OpenStreetMap is the solution of the previous ConnectiCar V1 group, and no changes have been made except for transferring it to a new Github repository managed by the customer and ensuring compatibility with the current InfluxDB configuration. This was achieved by removing the previous hardcoded values and modifying the Rahti2 configuration so that all organization and bucket settings are automatically retrieved from the default InfluxDB settings in the Rahti2 environment. Unlike before, the customer no longer needs to read hardcoded bucket and organization data from the source code.

In a case where the OpenStreetMap configuration needs to be redone, proceed as follows:

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Verkkosivusto

Tekoälyn generoima sisältö voi olla virheellistä.

In Rahti2 enter Developer state and select “+Add” and select from “Git Repository” function “Import from Git”. This builds a container image and deploys the application into the Rahti2 environment.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Verkkosivusto

Tekoälyn generoima sisältö voi olla virheellistä.

Provide Git Repo URL. In this case use customer provided GitHub <https://github.com/VishakaB/ConnectiCar2.0> as Git Repo URL. Rahti2 detects it as node.js project.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, numero, Fontti

Tekoälyn generoima sisältö voi olla virheellistä.

Unlike the previous group’s implementation, we now provide all variable data explicitly and reference the InfluxDB ConfigMap.

These are current settings as add value:

Name: Value:  
  
INFLUX\_URL <https://influxdb-connecticar.2.rahtiapp.fi>

INFLUX\_TOKEN <provide here InfluxDB API read-only token>

These are current settings from ConfigMap (select “Add from ConfigMap or Secret)

INFLUX\_ORG ConfigMap: influxdb-setup-parameters

org

INFLUX\_BUCKET ConfigMap: influxdb-setup-parameters

bucket

Kuva, joka sisältää kohteen teksti, kuvakaappaus, numero

Tekoälyn generoima sisältö voi olla virheellistä.

Finally, you can set hostname. Currently Rahti2 can be given in form **<yourhostname>**.2.rahtiapp.fi

Currently it is set as **connecticar.2.rahtiapp.fi**

For the certificates there is no need for configuration, since Rahti2 will configure them automatically. **Please do not confuse this with the MQTT Broker certificates, as OpenStreetMap and the MQTT Broker are entirely unrelated.**

Finally in the end select “Create”. Build takes about 3-5 minutes. Rahti2 may display error messages related to container creation, but this is expected behavior once the build process has been completed.

# Troubleshooting

Several connectivity issues were observed during the final phase of testing, specifically related to the Eclipse Mosquitto MQTT Broker itself. Many of the issues were resolved by simply restarting the MQTT Broker. The easiest way to restart it is through the Rahti2 web interface by temporarily scaling the number of pods down to zero and then back up to one. This can be done in the Rahti2 environment:

Administrator -> Workloads -> Deployment -> mosquitto -> Decrease the Pod count -> Increase the Pod count.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, numero

Tekoälyn generoima sisältö voi olla virheellistä.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Tietokonekuvake

Tekoälyn generoima sisältö voi olla virheellistä.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, Verkkosivusto

Tekoälyn generoima sisältö voi olla virheellistä.

After scaling up, Eclipse Mosquitto MQTT Broker should function again.