Edvard Shalaev

Containerized Software Development for Industrial Environment

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Bachelor's Thesis

12 October 2021

Abstract

Author:	Edvard Shalaev
Title:	Containerized software development for industrial environment
Number of Pages:	27 pages
Date:	12 October 2021
Degree:	Bachelor of Engineering
Degree Programme:	Media Technology
Professional Major:	Information and Communications Technology
Supervisors:	Jyrki Keinänen, Project Manager Patrick Ausderau, Senior Lecturer

The main objective of this thesis was to migrate an Agent-application to containerized environment and studying benefits of application containerization and Industrial Edge environment. Application was developed using NodeJS and TypeScript, and packaged with Docker.

NodeJS is an asynchronous event-driven runtime for JavaScript which enables development of JavaScript back-end applications. TypeScript is a programming language that is used for development of strict JavaScript applications. Container is an isolated environment of operating system for running program code. Docker is an open platform containerization engine, which enables isolation of the applications for better security and scaling, quicker deployment and development. Agent is a cyclic application, the purpose of which is to fetch data from Sinema- and Sinec NMS-servers and upload it to the MindSphere.

The process of the project was divided into three parts. In the first part a simple application was developed from scratch as a proof of concept and containerized using Docker. It was published to the IEM from where it was installed to IED. In the second phase the existing agent application was modified to work in containerized environment. It was uploaded and tested on the IED. The uploaded agent application was tested with real data.

Agent application was successfully containerized, migrated and tested. However, future improvements could include more user-friendly agent configuration, logs upload to the MindSphere and possible utilization of the IE data busses.

Keywords:
Docker, containerization, NodeJS, MindSphere, Industrial Edge

Tiivistelmä

Tekijä:	Edvard Shalaev
Otsikko:	Ohjelmistokehitys konttialustalle teollisuusympäristöön
Sivumäärä:	27 sivua
Aika:	12.10.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mediatekniikka
Ohjaajat:	Projektipäällikkö Jyrki Keinänen Lehtori Patrick Ausderau

Opinnäytetyön päätavoitteena oli siirtää agenttisovellus konttiympäristöön ja tutkia sovellusten säiliöinnin ja Industrial Edge -ympäristön etuja. Sovellus kehitettiin käyttäen NodeJS-ajoympäristö, TypeScript-ohjelmointikieli ja pakattiin Docker-konttimoottorin avulla.

NodeJS on JavaScriptin asynkroninen tapahtumapohjainen ajoympäristö, joka mahdollistaa JavaScript-taustaohjelmien kehittämisen. TypeScript on ohjelmointikieli, jota käytetään strict-JavaScript-sovellusten kehittämiseen. Kontti on käyttöjärjestelmän eristetty ympäristö ohjelmakoodin suorittamista varten. Docker on avoimen alustan konttimoottori, joka mahdollistaa sovellusten eristämisen paremman tietoturvan ja skaalauksen sekä nopeamman käyttöönoton ja kehityksen varmistamiseksi. Agentti on syklinen sovellus, jonka tarkoituksena on noutaa tietoja Sinema- ja Sinec NMS -palvelimilta ja ladata ne MindSphereen. Mindsphere on Siemensin kehittämä teollinen IoT-pilvipohjainen käyttöjärjestelmä.

Projektin prosessi jaettiin kolmeen osaan. Ensimmäisessä osassa kehitettiin yksinkertaista sovellusta konseptin todisteeksi ja pakattiin Dockerilla. Sovellus julkaistiin Industrial Edge Managmentiin, josta se asennettiin Industrial Edge-laitteelle. Toisessa vaiheessa olemassa olevaa agenttisovellusta muutettiin toimimaan konttiympäristössä. Se ladattiin ja testattiin Industrial Edge-laitteella ensin valetiedolla, minkä jälkeen se testattiin todellisilla tiedoilla.

Agenttisovellus säiliöitiin, siirrettiin ja testattiin onnistuneesti. Tulevat parannukset voivat kuitenkin sisältää käyttäjäystävällisemmän agentin kokoonpanon, lokien lataamisen MindSphereen ja mahdollisen Industrial Edge-tietoväylien hyödyntämisen.

Avainsanat:
Docker, säiliöinti, konttiteknologia, NodeJS, MindSphere, Industrial Edge

Contents

List of Abbreviations

API:	Application Programming Interface.
GUI:	Graphic User Interface. Used to allow user visually comprehend application and control it.
IDE:	Integrated Development Environment.
IE:	Industrial Edge.
IEM:	Industrial Edge Management.
IED:	Industrial Edge Device.
IoT:	Internet of Things.
OTCM:	Operational Technology Condition Monitor.
SDK:	Software Development Kit.
UTC:	Coordinated Universal Time.
UI:	User Interface. Allows user interaction with the application.
NMS:	Network Management System.

1 Introduction

This project was completed in the interests of Siemens Oy's industrial department. Siemens Oy is a limited company that is a part of the Siemens AG conglomerate. Siemens Oy was founded in 1919 in Espoo. It has multiple departments, such as Industry, Energy, Healthcare, Infrastructure and Cities.

Siemens produces products for industrial environment and automation, two of which were used in this project namely MindSphere, and Industrial Edge (IE).

MindSphere is an Internet of Things (IoT) service solution built on the Mendix application platform. IE, on the other hand, is a platform for the industrial applications that acts as a on field computer. Both products are essential for the project, as MindSphere was used for analytical data storage and IE was used for hosting the agent application.

The main idea of this thesis is to containerize an existing back-end Agent application for IE and to study its interaction with the MindSphere. The project is considered a success if a working and stable Agent application is converted, dockerized and hosted on the Industrial Edge platform with MindSphere integration.

The flow of this project’s development will be described in next chapters Theory, Background, Development, Conclusion.

The theory chapter covers history and description of solutions used in this project. The background chapter covers previous version of an Agent- and Operational Technology Condition Monitor (OTCM) -application that works with it in parallel, and their dataflow and synergy principles. The development chapter describes with examples how development environment was prepared, and how project was planned, coded, and tested. The conclusion part describes the project’s overview: what was the goal, what was achieved, possible improvements and which should be the next steps.

2 Theory

Software, also known as a program, is a collection of instructions for a computer to execute. Software can be written in different programming languages. A programming language is an invented language that can be used to program a device to perform various functions. The first programming language and the first piece of software were invented by Ada Lovelace in the 19th century, while the first upper-level programming language was invented by Konrad Zusen in the 1940s. [1.]

There are multiple choices for what language should be used when writing an application. The choice is influenced by such criteria as simplicity of learning the language, simplicity of writing, environment’s backing for language and support of the application in the future.

2.1 Back-end

Software can be controlled by the user through ‘user interface’ also referred as front-end. In a back-end applications, on the other hand, the user does not control the program, as back-end applications are mostly located out of reach from clients. The program controls itself regarding of what it is instructed to do and what data it receives or through API (Application Programming Interface) a software’s interface that allows other software to communicate with it. [24.]

A back-end application, also referred to as server, is a software whose function is to provide functionality such as parsing data, forwarding data, storing data, to other applications or devices. Such hardware or software are called clients. This architecture is called the client-server model and is shown in figure 1.

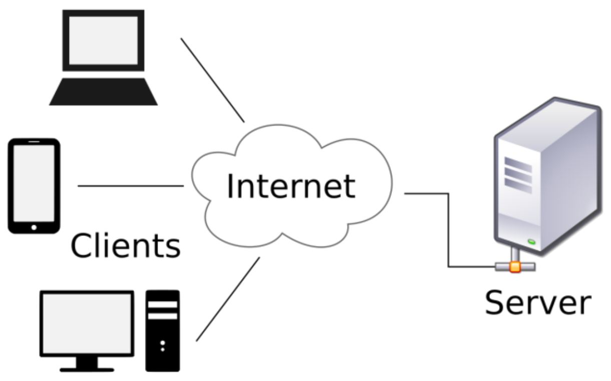


Figure 1. Client-server model [2].

2.1.1 JavaScript

Javascript is a programming language developed by Brenda Eich and published in 1995. Originally, JavaScript was used to dynamically process information on web pages, but today it is used in many different tasks. JavaScript is widely used in the world because it is simple, well supported and one of the most popular programming languages. [3, 4.] Due to its popularity, many tools and libraries are created for it.

Currently, JavaScript is not only a client-side language, but it is also used in server-side program development. All JavaScript code works on a V8 machine, which is a JavaScript and WebAssembly machine written in C++. It is used in Chrome and NodeJS. [6.]

NodeJS (also referred as Node.js) is an asynchronous event-driven runtime for JavaScript. It enables possible development of scalable back-end programs with JavaScript. [7.]

2.1.2 TypeScript

TypeScript is a typified programming language that is built around JavaScript. It was developed by Microsoft and launched for the public in 2012. [8.] Main idea behind TypeScript is to bring more strictness to development of JavaScript applications. Strictness of the TypeScript brings better understanding of the code and greater debugging possibilities. [9.] More strictness is achieved by utilizing concept of types in code.

In JavaScript all variables by default can be of different types which means that any kind of data could be assigned to the variable. In case of TypeScript, these types are assigned to the variables by the user, or they are dynamically depending on data type that is assigned to the variable. With TypeScript integration IDE (Integrated development environment) throws an error if wrong type of data is assigned to the variable.

```
// JavaScript variable
let num = 1;

// Assigning string to a num variable doesn’t throw an error
number = “text”;


// TypeScript variable
// In this case IDE treats the num variable to be of the number type
let num = 1;

// IDE throws the wrong type error
num = “text”;
```

Listing 1. Example of the JavaScript and TypeScript code.

For example, if two identical variables were created, but one was in JavaScript and other in TypeScript, as shown on listing 1, JavaScript variable can be assigned any type of data, but in case of the TypeScript if wrong type of data is assigned to the variable, an IDE will immediately throw an error which makes it possible to do necessary fixes even before launching the application.

2.1.3 MindSphere

MindSphere is an Industrial IoT cloud-based operating system developed by Siemens. It was made for all kind of Industrial IoT use cases. Data upload to the MindSphere cloud services is possible by utilizing MindSphere's open APIs. [10, 11.]

MindSphere is divided to multiple different services. Those services are accessible through straight API requests or through MindConnect Software Development Kit (SDK). [12, 13.]

2.1.4 Siemens Industrial Edge

Siemens Industrial Edge (IE) is the Siemens's platform for hosting applications close to the shopfloor. This allows for better and safer interactions with automation systems as the Industrial Edge Device (IED) could be located behind plant's firewall.

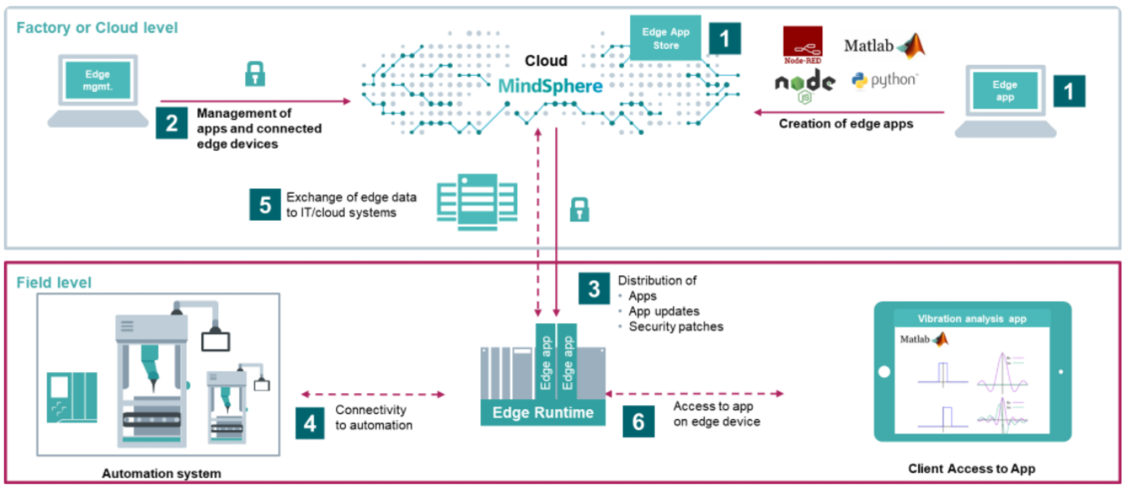


Figure 2. Diagram showing basic idea behind IE platform [14].

As shown in figure 2, IED is usually located on the plant itself and Industrial Edge Management (IEM) is in the cloud. IED runs operating system based on Linux. One of the key features of the IED is the installation of the containerized applications and runtime log collection. Installation of these applications happens through IEM. In this project IED works as a server – a device that provides functionality for other programs or devices. [14.]

2.2 Containers

Container is an isolated environment of operating system for running program code. While containers resemble virtual machines, they are different. One of the key differences is that each of the virtual machines creates a separate entity with an own operating system. [15.] Containers on the other hand share the host's operating system, as shown on in figure 3.

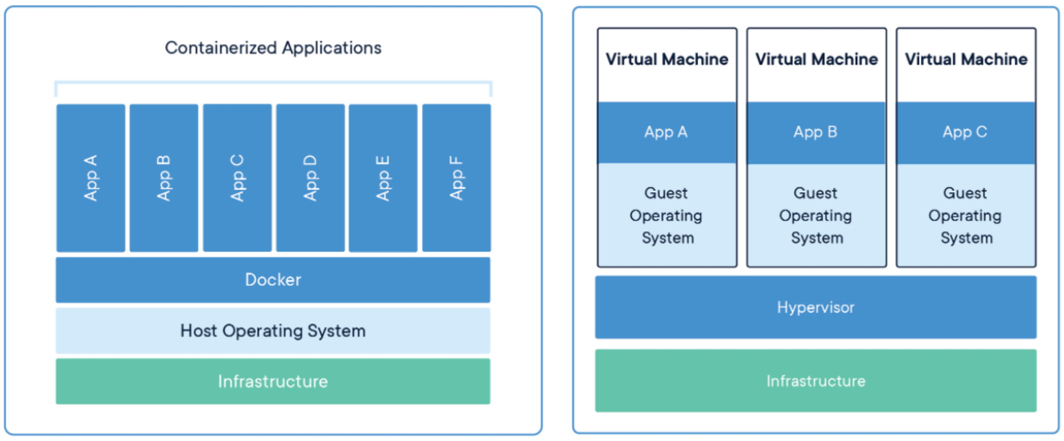


Figure 3. Difference between containers' and virtual machines' infrastructures [15].

This makes containers more lightweight and reduces their start up times compared to virtual machines. There are many different containerization solutions as Kubernetes, Mesos and Docker. Docker was used as a containerization engine in this project as it was required for the IE application development. [16.]

Docker is an open platform containerization engine. It enables isolation of the applications for better security and scaling, quicker deployment and development. Docker ensures that containerized applications run the same way on all devices. [15, 17, 18, 19.]

3 Background

This project is a part of a logical continuation of the existing OTCM environment's development. This environment consists of Agent and OTCM applications and is connected to MindSphere. Those applications will be described in this chapter.

3.1 Agent application

Agent application is a cyclic application that is installed on a plant, in the same network or behind same firewall as Sinema or Sinec Network Management System (NMS) servers. Sinema or Sinec server is a monitoring server for observing and diagnosing of on plant machinery. [20.]

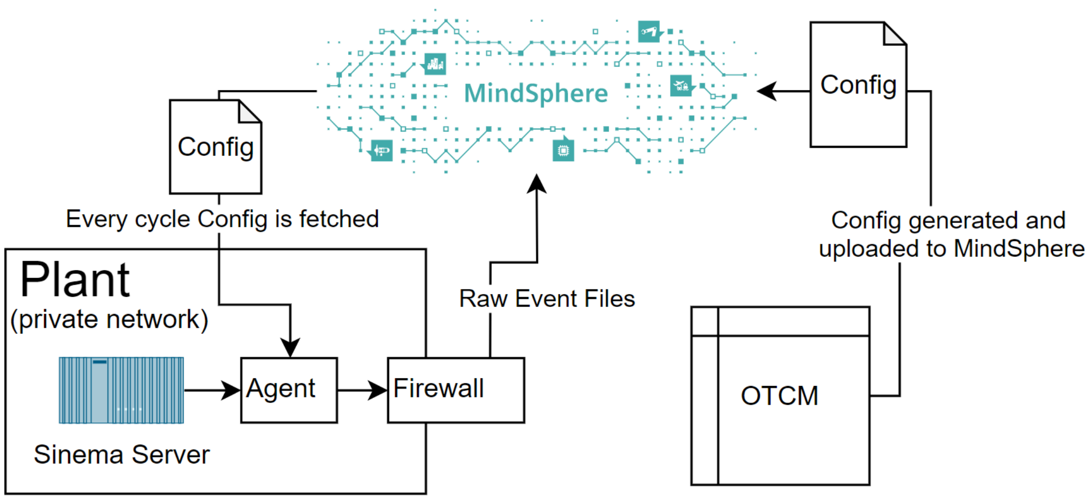


Figure 4. Graph that depicts Agent's environment and dataflow.

Agent's purpose is to fetch data from said servers and upload it to the MindSphere. In case there are errors or no data available, Agent uploads event's to MindSphere with exception description. Agent is configured from OTCM and config is updated from MindSphere. Agent's surrounding environment and dataflow is shown on figure 4.

3.2 OTCM

OTCM is Siemens's private application used for clients' network state and health monitoring, and support purposes. Development of this application is currently in continuous development.

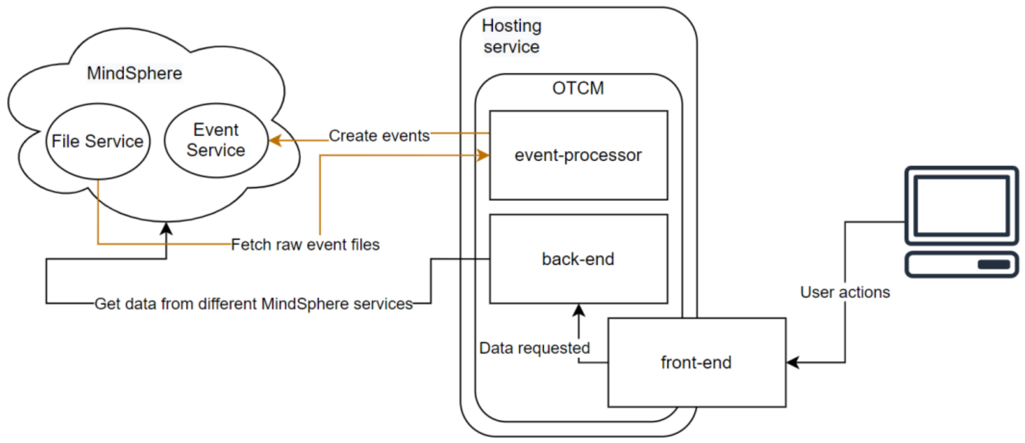


Figure 5. Diagram that shows OTCM application's data flow.

Application has three main components

- Front-end
- Back-end
- Event-processor.

Front-end component displays data to user and allows for user friendly interaction with the application, while back-end processes user requests and returns processed data to the front-end component. Some of the simple front-end's requests that do not require heavy data parsing are redirected straight to the MindSphere's services. Event-processor is a data processor that runs in background in cycles. Its job is to fetch raw event files from MindSphere and parse those into events and upload back to MindSphere's Event Service. Communication between those components is shown in figure 5.

4 Project Development

Migration of the agent application was done in three phases. The first phase was to develop a simple application as a proof of concept. After said application was uploaded on the IED and was running, the second phase of development should commence. The second phase of development was to remove unused parts from the code and packages, and upload a containerized version of the application to the IED. Third phase was to connect this Agent that is running on the IED to the Sinec server on the same network. This phase simulated real-world situation to test the application's behaviour. Based on results from testing, the code was refactored final time and containerized.

After three phases of development were passed, the final version of the application was released on the Gitlab in a separate branch, but in the same repository as previous versions of the Agent.

Git is a version control system. It was developed by Linus Torvalds. [21.] GitLab is the DevOps platform created for streamlining and speeding up development. [22.]

Development was carried out on a notebook provided by Siemens, it carried Windows 10 x64 operating system. Before development could start, the dev-environment should have been prepared. Visual Studio Code was selected for development as IDE (Integrated Development Environment) of choice because it was most familiar, opensource and is considered as one of the best IDEs for TypeScript and JavaScript application development. List of plugins were installed for better development experience

- Auto Import
- Better Comments
- Prettier - Code formatter.

Git was installed locally, and git bash was used as the main terminal application. Docker was used for containerization.

4.1 First phase of development (proof of concept)

As a proof-of-concept, a simple application was developed from scratch. The idea behind this application was to test IE environment's behaviour and to go through a complete deployment process before deploying the real application.

The application did not have any UI and its functionality was limited to reading config file, writing to console and one HTTP GET request. The application was written in TypeScript and after compiled to JavaScript, for local testing cfg-data folder was created with template config.json which was red if application was not launched in production mode. In case errors happened while reading config, simple error catcher was added. To be able to track if the application has successfully read the config file, config object logging was added. Before containerizing the application, it was tested by running it locally.

```
import express from "express";
import fs from "fs";

let config;

try {
  config = JSON.parse(
    fs.readFileSync(
      process.env.PROD ? "/cfg-data/config.json" : "./cfg-data/config.json",
      "utf-8"
    )
  );
} catch (error) {
  console.log(error);
}

const app = express();

app.get("/", (req, res) => {
  return res.json({
    message: "Everything worked!",
  });
});

console.log("wow app 0.0.2?", config);

app.listen(2000, () => {
  console.log("App working on port:", 2000);
});
```

Listing 2. Test application's code from index.ts file.

Test were considered successful when config object was logged to console, as seen in line 25 of listing 2. Also GET request, in lines 19-23 of listing 2, should return desired message.

4.1.1 Application containerization

After testing was complete Dockerfile, contents of which are shown in listing 3, was created in the root of the application.

```
FROM node:12-alpine

#! Env

#! Create Directory for the Container
WORKDIR /app

#! Only copy the package.json file to work directory
COPY package.json .

#! Install all Packages
RUN npm i

#! Copy all other source code to work directory
ADD . /app

#! TypeScript
RUN npm run build

#! Start
CMD [ "npm", "start" ]
EXPOSE 2000
```

Listing 3. Applications Dockerfile.

The final version of the test application was containerized by running the following command: `docker build . -t docker-test:0.0.4`. Node:12-alpine was used as it is considered smallest node 12 image. [25.]

4.1.2 Project and app registration on IEM

Next step was to create a project on IEM. It was possible by accessing IEM via internet browser by visiting `https://*ip_of_edge_management*:9443/pp/app/home` and signing in, in this case Google Chrome was used as the web browser. After sing-in user is greeted with multiple Edge Management options illustrated in figure 6.

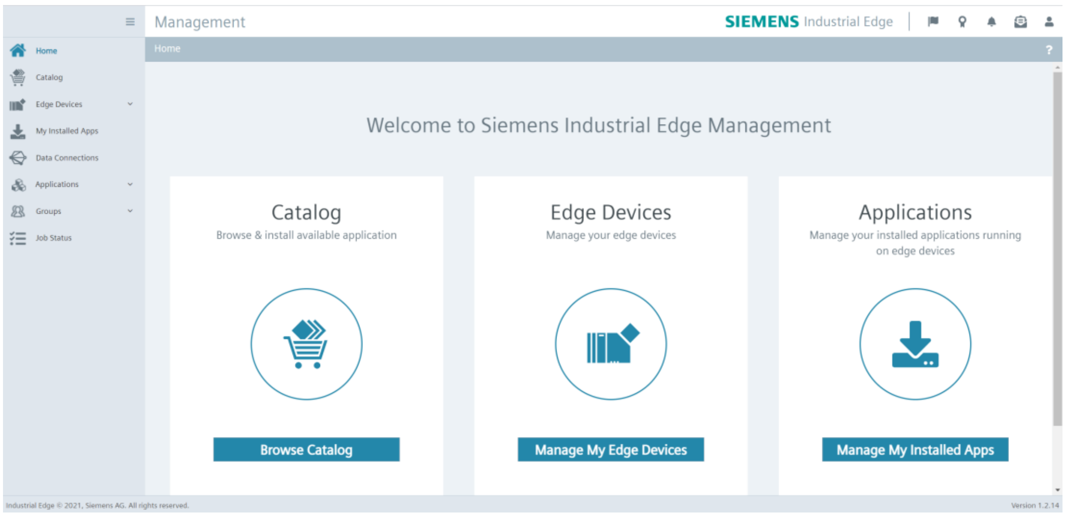


Figure 6. IEM home page.

A new project was registered on My Projects page. To access said page, the user should select the Applications folder from side-menu and click on My Projects button. First, the user needs to create a new project by clicking on Create Project button in right top corner, after which s/he is greeted with a modal window as shown in figure 7, where project name, description of a project and a company for which project is created should be specified. If the company is not available for selection on the modal window, it should be added manually by clicking the + icon and filling required fields.

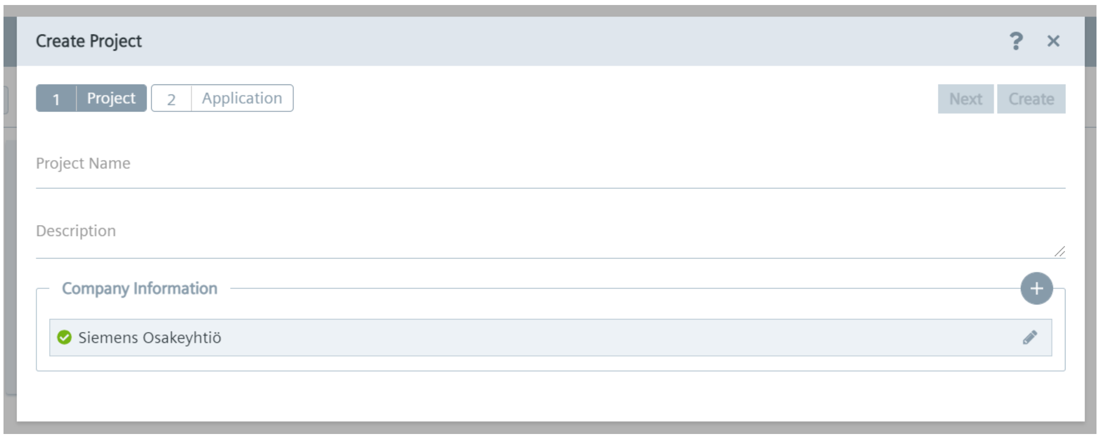
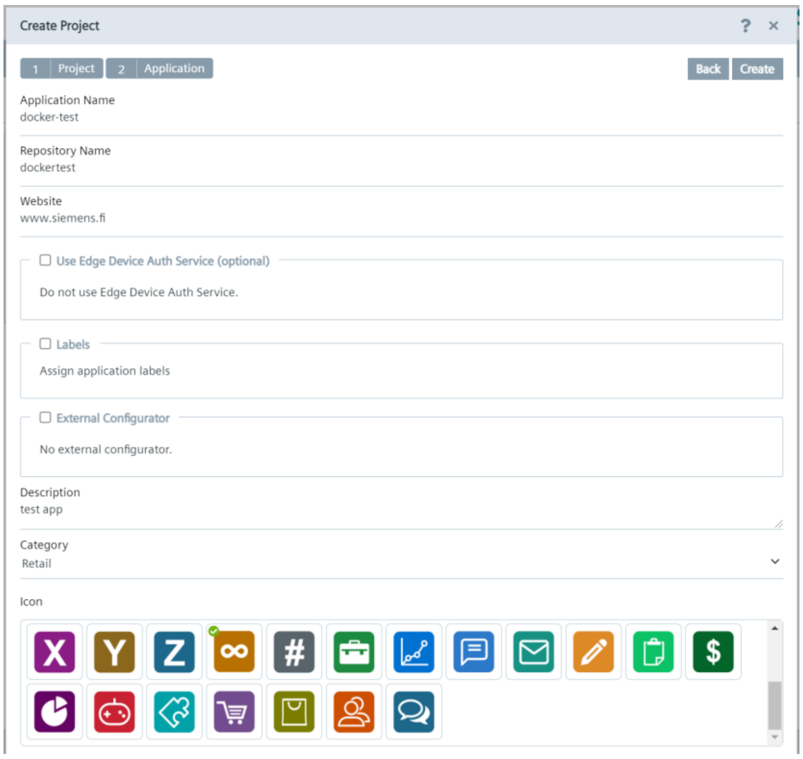


Figure 7. Project creation modal window.

After filling all fields, it is possible to create an empty project by clicking the Create button or, as in this project, continue to the creation of a new application straight away by clicking the Next button. When the button is clicked the application creation modal window is opened as shown in figure 8. Fields as the Application name, the Repository name, the Website and the Description need to be filled and an icon needs to be selected from the existing or added new icon by the user.



Application Name
docker-test

Repository Name
dockertest

Website
www.siemens.fi

☐ Use Edge Device Auth Service (optional)
Do not use Edge Device Auth Service.

☐ Labels
Assign application labels

☐ External Configurator
No external configurator.

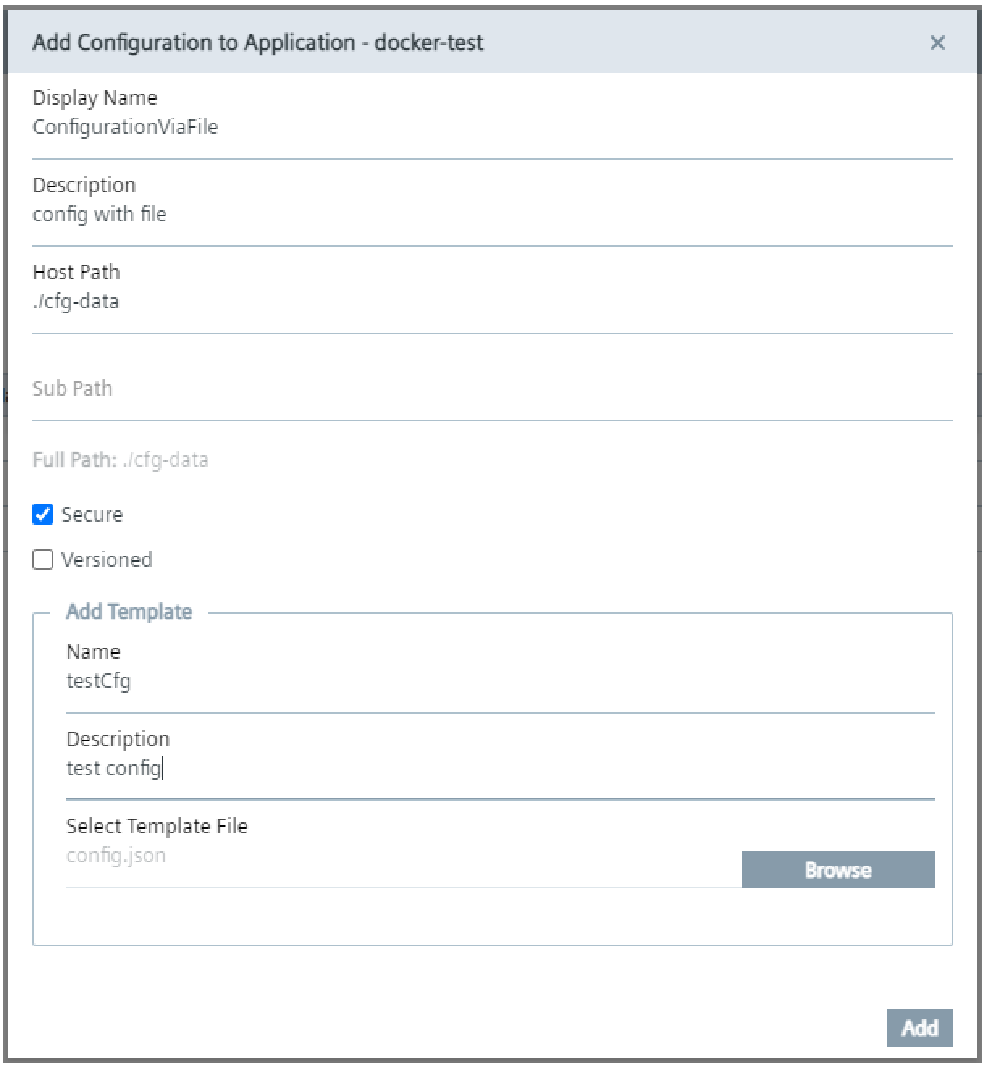
Description
test app

Category
Retail

Icon

Figure 8. Application creation modal window.

After everything is completed, a new application can be created. The created application will appear on the My Projects page as a clickable icon. The Developers user group was added to the application by clicking an icon of the application and clicking on the plus button on the right from the User Groups. Adding the Developers user group was essential for the deployment of the application as for the privacy settings of the IE device.



Add Configuration to Application - docker-test

Display Name
ConfigurationViaFile

Description
config with file

Host Path
./cfg-data

Sub Path

Full Path: ./cfg-data

[✓] Secure

[ ] Versioned

Add Template

Name
testCfg

Description
test config

Select Template File
config.json

Figure 9. Configuration template creation modal window.

Configuration template was added on the same page. It was possible by pressing the Configurations button and filling all the required fields and selecting the template file from the local files, as shown in figure 9. Saving the new configuration template enables application configuration with the file.

4.1.3 App Publisher setup

For application uploading App Publisher – Industrial Edge application was used. Application needs to be setup before uploading applications to the IEM.

Docker Engine needs to be connected to the publisher. Connection can be established by clicking + *Docker Engine* button on top right corner and filling

Docker engine ip and port. Also, the publisher needs to be connected to IEM. To connect App Publisher to IEM, user must click *Go Online* button and fill in URL of the IEM, after which user is redirected to IEM login page. On successful login App Publisher connects to the IEM and setup of the App Publisher is done.

4.1.4 Application upload

Before starting the upload process, the application needs to be containerized locally. To upload application first it needs to be selected from My Projects list of applications. After selection list of application's versions is displayed, a new application's version must be created to initialize upload process.

To create a new version *+Add New Version* button was clicked, which opened a modal window with docker compose version selection; 2.4 version of docker compose was selected at it was newest version available for selection. On selection the user is redirected on a new page for application configuration before upload.

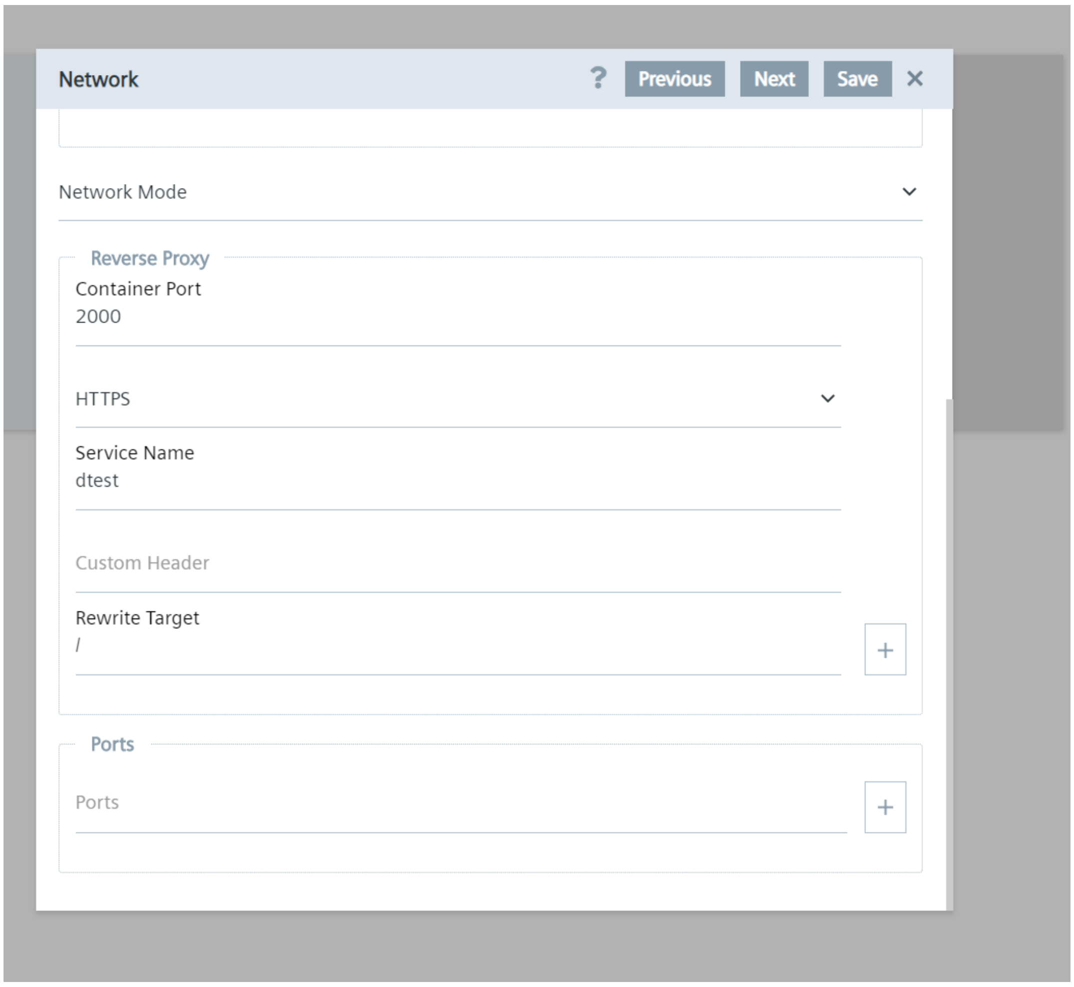
The user can manually configure application or YAML file can be uploaded and finalized manually if required. In case of developed test application YAML file was created beforehand, and its content is shown in listing 4. YAML was imported by pressing *Import YAML* button in top right corner and selecting YAML file from opened explorer window.

```
version: "3"

services:
  docker-test-app:
    image: "docker-test:0.0.4"
    environment:
      - PROD=true
    mem_limit: 200mb
```

Listing 4. Code from docker-compose.yaml

As the application has web API, reverse proxy had to be configured manually on upload. Reverse proxy was needed for communication between client and containerized application. To configure it, the configuration menu was opened by pressing pen icon in top right corner. After the menu opened, Network tab was selected and under Reverse Proxy headline all fields were filled, except custom headline, as shown in figure 10. After filling a new rule was added by pressing + button and saved in top right corner of the modal window.



Network Mode

Reverse Proxy

Container Port
2000

HTTPS

Service Name
dtest

Custom Header

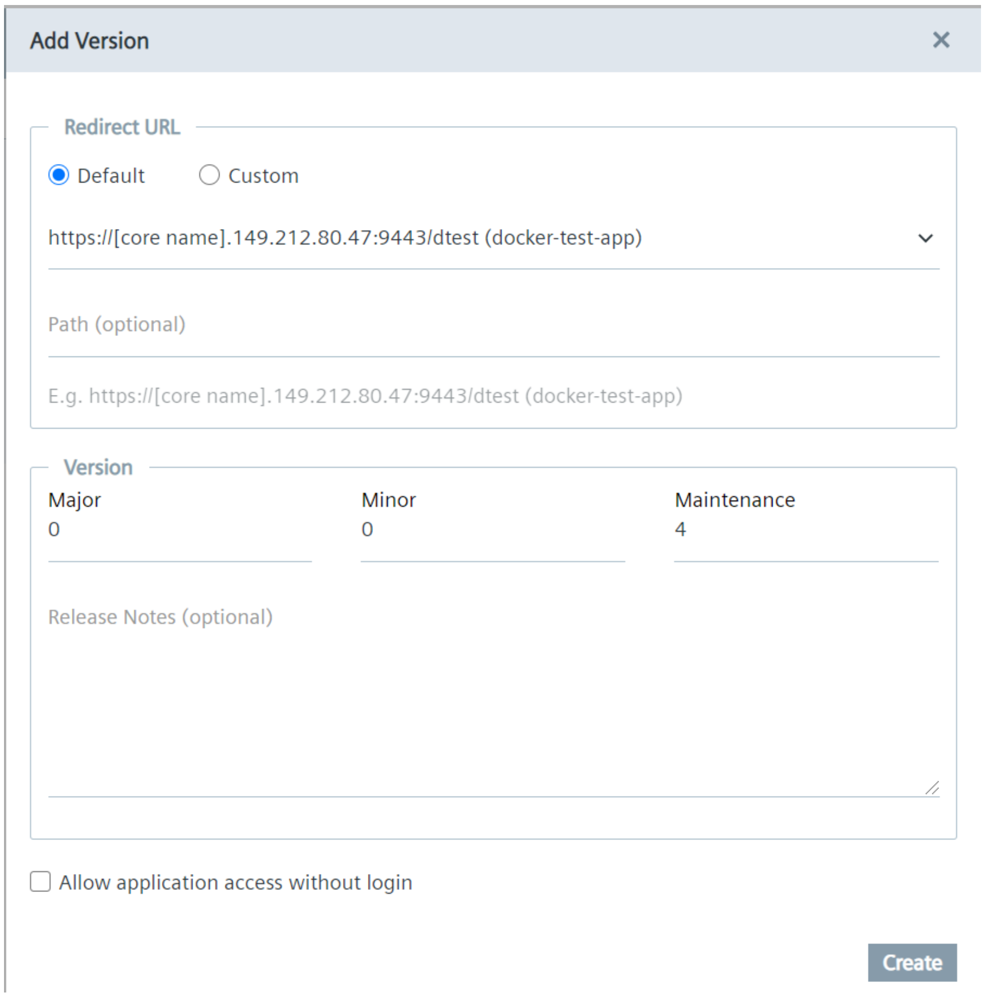
Rewrite Target
/

Ports

Ports

Figure 10. Reverse proxy configuration.

When configuration was done it was saved by pressing “Review” button and after “Validate & Create” button. On newly opened modal window, as shown on figure 11, the version was changed to be 0.0.4 as the third iteration of test app was being uploaded.


Add Version

Redirect URL

Default [✓] Custom

https://[core name].149.212.80.47:9443/dtest (docker-test-app)

Path (optional)

E.g. https://[core name].149.212.80.47:9443/dtest (docker-test-app)

Version

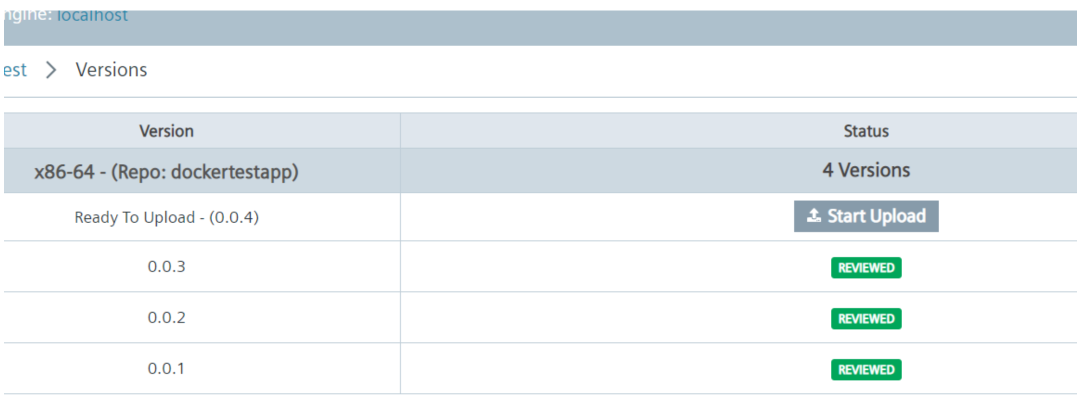
Major	Minor	Maintenance
0	0	4

Release Notes (optional)

☐ Allow application access without login

Figure 11. Version selection before creating new version of application.

After creation of a new version, the user is redirected back to the version selection list with the new version on the top of the list, as seen in figure 12.


ngine: localhost

est > Versions

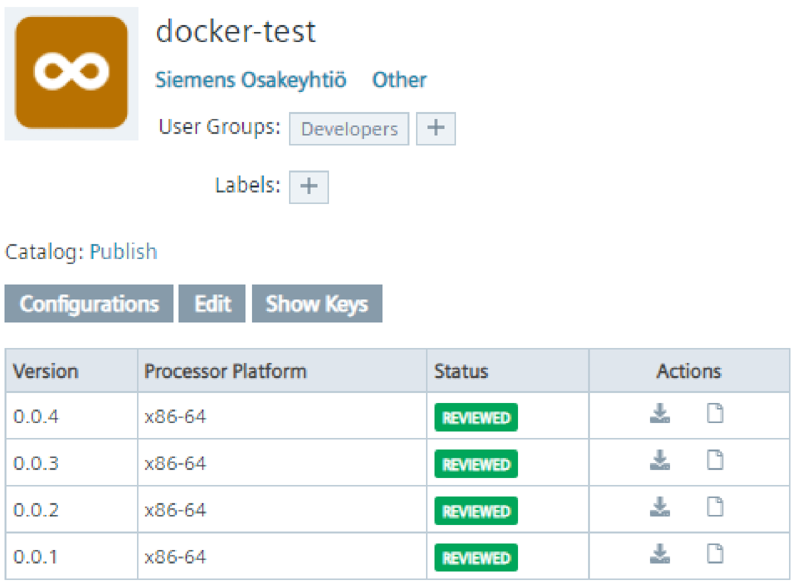
Version	Status
x86-64 - (Repo: dockertestapp)	4 Versions
Ready To Upload - (0.0.4)	Start Upload
0.0.3	REVIEWED
0.0.2	REVIEWED
0.0.1	REVIEWED

Figure 12. List of application’s versions in App Publisher with newly created version ready for upload.

In the new version's row on the second column a Start Upload button appeared. Clicking it uploads application to the IEM, from where it could be installed to IE device.

4.1.5 Application installation on IE device

Application installation happens from application's page. After uploading a new version of the application to IEM it should be shown in the list of app's versions, which is shown in figure 13.


docker-test

Siemens Osakeyhtiö Other

User Groups: Developers

Labels:

Catalog: Publish

Version	Processor Platform	Status	Actions
0.0.4	x86-64	REVIEWED	
0.0.3	x86-64	REVIEWED	
0.0.2	x86-64	REVIEWED	
0.0.1	x86-64	REVIEWED	

Figure 13. Application's page with list of its versions.

To install the desired version of the application, install button should be clicked. Next the modal window with a two-step installation opens. In first step, the configuration template should be selected, if such is required by the app. It could be instantly modified by inputting desired code and saving it. In case of this test project, ConfigurationViaFile template was selected and modified to contain code from listing 5.

```
{
  "PORT": 2000,
  "MESSAGE": "IF THIS IS WORKING; THEN IT IS! :D"
}
```

Listing 5. Code imputed to configuration file.

When config was changed and selected by clicking on it, the second step was to select device on which to install the app. As only one device was provided for the project it was selected and Install Now button was clicked. As an app being installed does not contain trusted certificate, a warning message appears, and the installation process continues by pressing allow button.

4.1.6 Reviewing application after installation

When reviewing application after installation it is important to determine

- Is the application running?
- Is the application running correctly?
- Is the application configured correctly and is configuration reachable?

The application could be reviewed from the said device's web-UI. To access devices UI, `https://*ip_of_edge_device*/device/edge/app/apps` URL should be opened from internet browser. All applications installed are listed on *Apps* page.

To determine if an installed application is running and if it is running correctly, a decision to view its logs was made. Logs were downloaded by clicking three-dots-icon under the icon of the application and selected Download Logs, as seen in figure 14.

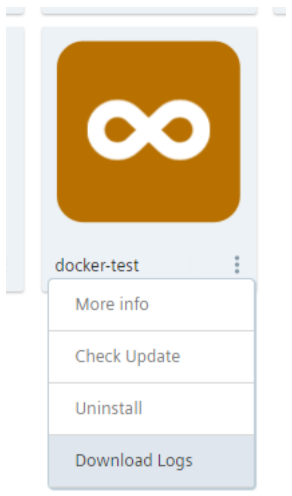


Figure 14. Button to download application’s logs.

Clicking said button initiated download of tar file containing logs, part of which is shown in listing 6. After unpacking the downloaded file and opening logs containing a file it was determined that back-end of the application was running correctly, and the new configuration was readable as its logs matched text inputted on installation.

```
{"log":"\n","stream":"stdout","time":"2021-08-24T12:33:35.147939578Z"}
{"log":"\u003e docker-test@1.0.0 start /app\n","stream":"stdout","time":"2021-
08-24T12:33:35.148210098Z"}
{"log":"\u003e node dist/index.js\n","stream":"stdout","time":"2021-08-
24T12:33:35.148269738Z"}
{"log":"\n","stream":"stdout","time":"2021-08-24T12:33:35.148324027Z"}
{"log":"wow app 0.0.2? { PORT: 2000, MESSAGE: 'IF THIS IS WORKING; THEN IT IS!
:D' }\n","stream":"stdout","time":"2021-08-24T12:33:36.522541455Z"}
{"log":"App working on port: 2000\n","stream":"stdout","time":"2021-08-
24T12:33:36.640613337Z"}
```

Listing 6. Application’s logs.

The front-end of the application was tested by simply clicking the application’s icon on Apps page. On click, a new tab opened with json, which is shown in listing 7.

```
{
  "message": "Everything worked!"
}
```

Listing 7. Json returned from the application.

Thus, it was determined that both back-end and front-end of the application were working. After successful testing it was decided that instead of writing Agent from scratch, the existing Agent application with minimal changes should be containerized, as it was deemed less time consuming and a simpler solution.

4.2 Second phase of development (refactoring, containerization, deployment)

To start the second phase of development, the code of the Agent application needed to be stored locally and all required applications should be installed. Development started by cloning online repository to local. Authorization process on the web page is seamless, as it only requires the user's smart card to be read from card reader.

After successful authorization, it was possible to find the needed repository and copy the cloning link. Repository was cloned by executing command, which is illustrated in listing 8, before locating to the desired folder in terminal.

```
$ git clone git@<host>:<path> && cp -r ./sinema-mdsp-agent/* . && rm -rf
sinema-mdsp-agent/

Cloning into 'sinema-mdsp-agent'...
remote: Enumerating objects: 517, done.
remote: Counting objects: 100% (517/517), done.
remote: Compressing objects: 100% (242/242), done.
remote: Total 517 (delta 306), reused 452 (delta 263), pack-reused 0
Receiving objects: 100% (517/517), 267.37 KiB | 1.60 MiB/s, done.
Resolving deltas: 100% (306/306), done.
```

Listing 8. Terminal log after repository cloning command execution.

For running the application while development packages were installed and to install packages `npm install` command was executed. List of installed dependencies is shown in listing 9.

```
"devDependencies": {
    "@types/cron": "redacted",
    "@types/moment-timezone": "redacted",
    "@types/node": "redacted",
    "@types/prompts": "redacted",
    "pkg": "redacted",
    "standard-version": "redacted",
    "ts-node": "redacted",
    "tslint": "redacted",
    "tslint-config-prettier": "redacted",
    "typescript": "redacted",
 },
 "dependencies": {
    "@mindconnect/mindconnect-nodejs": "redacted",    "@types/winston":
"redacted",
    "axios": "redacted",
    "cron": "redacted",
    "moment-timezone": "redacted",
    "winston": "redacted",
    "winston-daily-rotate-file": "redacted"
 }
```

Listing 9. Project dependencies, with redacted version numbers, before starting migration.

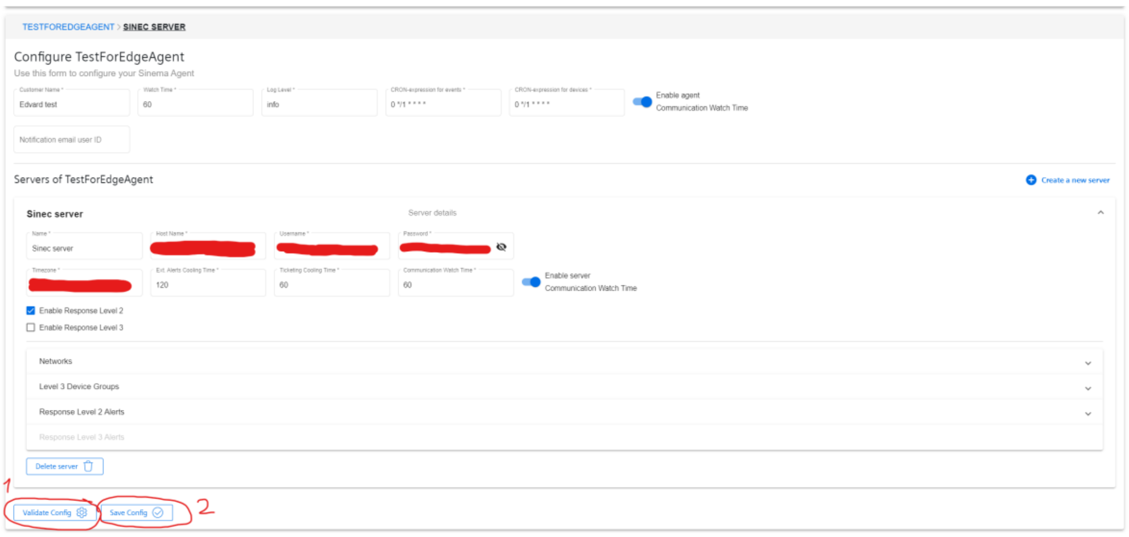
After package installation, a new branch was created and named *container-development*. Migration process started by creating three essential docker files as `Dockerfile`, `docker-compose.yaml`, `.dockerignore`.

Power Shell scheduled job creation code was removed as it was not used in the new environment and code line determining config path was refactored to read config file from `/cfg-data` location when application is running in container.

`pkg` package and `exe` script were removed as they were no longer needed. `start` script was changed to launch the built application and `dev` script was added to test application during development. After all changes were committed and pushed to remote, the application had to be tested locally.

4.2.1 Configuration

A new asset was created. `soydev.sinemaserveragent` asset type was given to the asset and `AppVersion` variable was set to 3, so it would be visible in the newest version of OTCM application, from where the agent was configured.


TESTFOREDGEAGENT > SINEC SERVER

Configure TestForEdgeAgent

Use this form to configure your Sinema Agent

Customer Name *	Watch Time *	Log Level *	CRON-expression for events *	CRON-expression for devices *
Edvard test	60	info	0 */1 * * * *	0 */1 * * * *

Enable agent Communication Watch Time

Notification email user ID

Servers of TestForEdgeAgent

Create a new server

Sinec server

Server details

Name *	Host Name *	Username *	Password *
Sinec server	[redacted]	[redacted]	[redacted]

Timezone *	Ext. Alerts Cooling Time *	Ticketing Cooling Time *	Communication Watch Time *
[redacted]	120	60	60

Enable server Communication Watch Time

[✓] Enable Response Level 2

[ ] Enable Response Level 3

Networks

Level 3 Device Groups

Response Level 2 Alerts

Response Level 3 Alerts

Delete server

1

Validate Config

Save Config

2

Figure 15. Web-view of agent’s config in the OTCM.

Sinec server’s address and credentials were added to the agent configuration to be able to fetch data from it. After agent’s config was done it was validated and saved as shown in figure 15.

4.2.2 Local testing

Before uploading the application, it needed to be tested locally. To do it, the application was containerized. `mindConnectConfig.json` was manually added to the container for testing purposes. Said file was added when building container by adding `COPY mindConnectConfig.json` line to the Dockerfile and removing said file from *.dockerignore* file.

After the container was built without errors it was launched in Docker GUI and it successfully connected to the MindSphere. The Agent was able to fetch config and tried to fetch data from Sinec server. As Sinec server did not exist in local network, it was not reachable by the Agent. This generated constant flow of network errors which was considered correct behaviour at the current situation. Local testing was deemed successful.

4.2.3 Application registration, upload and installation

Before uploading the application, it was registered in IEM. Application upload was done in a similar way as for the proof of concept with the only difference that reverse proxy was not configured, as the application did not have a user interface or any API.

After the application was visible in the list of uploaded applications it was installed on the IE device. When installing the application, MindConnect token was pasted into config, which was copied from the MindSphere agent’s page.

4.2.4 Reviewing application after installation

To review the application’s state after installation logs were fetched and examined. After quick examination of the logs it was clear that the application is working and is trying to fetch data from Sinec server from the configuration.

4.3 Third phase of development (testing with real-data, final fixes)

In the third phase of testing real Sinec server with real data was used. It was configured by a separate engineer. The same version of Agent used in the second phase of development was used. As Agent was already configured and already trying to fetch data from Sinec serve,r no additional actions were required regarding Agent’s configuration. After Sinec server’s configuration the Agent immediately connected to it, but no events were fetched. The reason for this was wrong time-zone, as default time-zone for Docker container was UTC. [23.]

```
{"log":"{\"message\":\"Fetching events from server: Sinec server for time
frame of 2021-09-06 12:17:00 .. 2021-09-06
14:16:45\",\"level\":\"debug\",\"timestamp\":\"2021-09-06
14:17:00\"}\n","stream":"stdout","time":"2021-09-06T11:17:00.211481802Z"}
{"log":"{\"message\":\"Got 200 from Sinec
server\",\"level\":\"info\",\"timestamp\":\"2021-09-06
14:17:02\"}\n","stream":"stdout","time":"2021-09-06T11:17:02.417116039Z"}
```

Listing 10. Agent’s logs of fetched events.

This bug was fixed by adding a new line, which sets the time-zone of the container to the Dockerfile of the Agent. Going through building, uploading and installation process was required to update the application on the IE device. After update, the Agent was able to fetch events from the Sinec server as seen in listing 10.

5 Conclusion

Core idea of the project was to migrate an Agent application from windows environment to work on IE device for creating product for IE, development process optimization and experience. Docker and IE environment were studied and utilized in this project. As a result of this project an Agent application was successfully containerized, migrated, and tested.

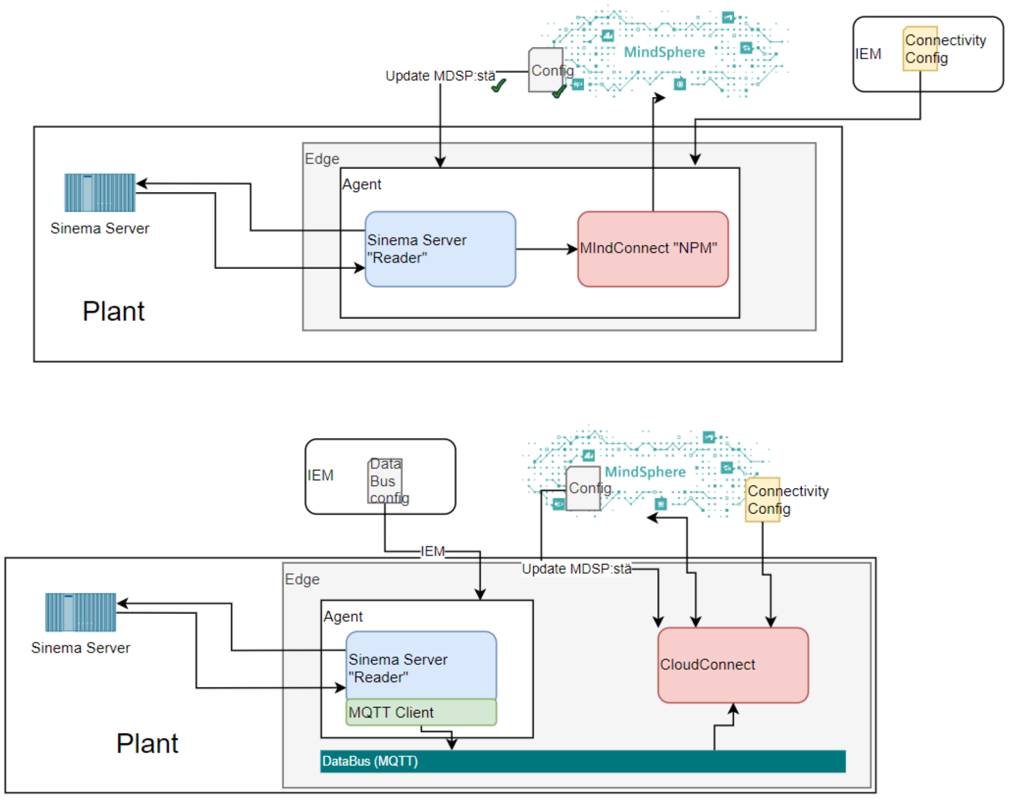


Figure 16. Diagrams with Agent’s current environment (top) and improved environment (bottom).

As described in the third phase of development, time-zone was hardcoded to the Dockerfile of the Agent application. In the future this should be configurable from the OTCM application. For future improvements, the Agent should be able to utilize IE device's data bus for event fetching, as shown in figure 16, and log fetching should be available from OTCM application for ease of the development.

References

1 Computer Programming Languages. 2020. Online material. ComputerScience.org. <https://www.computerscience.org/resources/computer-programming-languages/>. Updated 24.11.2020. Read 28.9.2020.

2 Mgyugcha. A computer network diagram of clients communicating with a server via the Internet. 2011. Image. Wikipedia.org. <https://en.wikipedia.org/wiki/Client%E2%80%93server_model#/media/File:Client-server-model.svg>. Updated 5.12.2014. Downloaded 30.9.2020.

3 2020. Online material. quirksmode. <https://www.quirksmode.org/js/intro.html>. Accessed on 25.11.2020.

4 Kamaruzzaman, Md. 2020. Top 10 In-Demand programming languages to learn in 2020. Online material. Towards Data Science. <https://towardsdatascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e>. Updated 4.2.2020. Accessed on 5.10.2020.

5 <https://miro.medium.com/max/2452/1*8YmyAEb3raC2gqj1AeVcvQ.png>. Downloaded 4.6.2021.

6 What is V8?. Online material. V8. <https://v8.dev/>. Accessed on 5.10.2020.

7 About Node.js®. Online material. NodeJS. < https://nodejs.org/en/about/>. Accessed on 26.09.2021.

8 TypeScript. Online material. Cleverism. <https://www.cleverism.com/skills-and-tools/typescript/>. Accessed on 26.09.2021.

9 What is TypeScript?. Online material. TypeScript. <https://www.typescriptlang.org/>. Accessed on 23.09.2021.

10 TechTarget Contributor. 2018. MindSphere. Online material. WhatIs.com. <https://whatis.techtarget.com/definition/MindSphere>. Updated 10.2018. Accessed on 27.09.2021.

11 MindSphere. Online material. Siemens. <https://siemens.mindsphere.io/en>. Accessed on 27.09.2021.

12 MindSphere Services. Online material. Siemens. <https://developer.mindsphere.io/apis/index.html>. Accessed on 20.09.2021.

13 MindConnect-NodeJS. Online material. npm. <https://www.npmjs.com/package/@mindconnect/mindconnect-nodejs>. Updated 08.09.2021. Accessed on 20.09.2021.

14 Posey, Brian. 2021. What is a Server?. Online material. WhatIs.com. <https://whatis.techtarget.com/definition/server>. Updated 07.2020. Accessed on 20.09.2020.

15 Online material. Docker. <https://www.docker.com/resources/what-container>. Accessed on 26.09.2021.

16 Industrial Edge apps. Online material. Siemens. <https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10364120>. Accessed on 12.09.2021.

17 What are containers?. Online material. NetApp. <https://www.netapp.com/devops-solutions/what-are-containers/>. Accessed on 28.09.2021.

18 Introduction to Containers. 2021. Online material. Unity. <https://fullstackopen.com/en/part12/introduction_to_containers>. Accessed on 28.09.2021.

19 Docker overview. Online material. Docker Docs. <https://docs.docker.com/get-started/overview/>. Accessed on 28.09.2021.

20 SINEMA Server – Making your network transparent. 2018. Online material. Siemens. <https://assets.new.siemens.com/siemens/assets/api/uuid:2a0ce858-ee1d-43d1-9692-8f5ea16088e8/di-ca-pi-rcm-sinema-server-brochure-en.pdf>. Accessed on 28.09.2021.

21 What is Git. Online material. Atlassian Bitbucket. <https://www.atlassian.com/git/tutorials/what-is-git>. Accessed on 30.09.2021.

22 What is GitLab?. Online material. GitLab. <https://about.gitlab.com/what-is-gitlab/>. Accessed on 30.09.2021.

23 Modifying a Container Time-Zone. Online material. TIBCO Software Inc. . <https://docs.tibco.com/pub/om-ll/5.0.0/doc/html/GUID-0618A976-3E0A-4750-B44E-F329452C05CE.html>. Accessed on 03.10.2021.

24 Application Programming Interface (API). 2020. Online material. IBM Cloud Education. <https://www.ibm.com/cloud/learn/api>. Accessed on 10.10.2021.

25 Garcia Perilla Julia. 2020. Alpine, Slim, Stretch, Buster, Jessie, Bullseye — What are the Differences in Docker Images?. Online material. Medium. < https://medium.com/swlh/alpine-slim-stretch-buster-jessie-bullseye-bookworm-what-are-the-differences-in-docker-62171ed4531d>. Accessed on 10.10.2021.