Anna Akimova

SOFTWARE FOR CBRN MONITORING IN ACCORDANCE WITH ATP45 STANDARD

Bachelor's thesis

Bachelor of Engineering

Degree Programme in Information Technology

2022

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland
University of Applied Sciences

Degree title	Bachelor of engineering
Author(s)	Anna Akimova
Thesis title	Software for CBRN monitoring in accordance with ATP45 standard
Commissioned by	Observis Oy
Year	2022
Pages	43 pages
Supervisor(s)	Timo Hynninen

ABSTRACT

With the growing number of possible kinds of weapons, it is very important to every nation to be able to monitor, detect and to be ready to respond to accidental or intentional attacks. One of the categories of threats that can pose threat to large groups of people are chemical, biological, radiological and nuclear incidents, commonly called CBRN threats. So, it is important to develop software that can summarize information about the as efficient distribution and utilization of correct information can enable timely and properly targeted authority activities in response to CBRN accidents. Also, it is important to spread information in a uniform way, so North Atlantic Treaty Organization (NATO) developed its own standard for reporting information regarding CBRN accidents – ATP45 standard. This thesis work is based on a software development case-study, it involved the research of the ATP45 standard, improvement of the existing in the commissioning company software and addition of further enhanced features.

In theoretical part, the main information about ATP45 standard was summarized. The kinds of threats and data passed in different types of messages were described in more detail. Also, information about basic software requirements and technologies used in ATP 45 software was described and main notions were clarified.

In the practical part of the thesis a real-word case of ATP-45 software implementation developed by Observis Oy was presented. The conversion to more appropriate Spring Boot implementation was described and also addition of extra features to the existing software was explained in detail.

The last part of the thesis presents the summary of the results of the work and reflection on which further improvements can be done to the software.

Keywords: Spring Boot, Docker, PostgreSQL, ATP45 NATO standard, CBRN

CONTENTS

1 INTRODUCTION 4

2 DEVELOPMENT CONTEXT 5

3 OVERVIEW OF JAVA AND SPRING BOOT REST API APPLICATIONS 6

3.1 Java and Spring boot 6

3.2 REST API 7

3.3 PostgreSQL databases, Spring JPA and Hibernate 10

3.4 Docker 11

4 OVERVIEW OF ATP45 NATO STANDARD 13

4.1 ATP45 standard and documentation: ATP45 and AEP45 13

4.2 Definition of CBRN threats 14

4.3 CBRN1-6 messages and data flow overview 15

4.4 CBRN messages and weather reports 18

4.5 Structure of a CBRN message 21

4.5.1 Message heading 21

4.5.2 Message sets – example of CBRN 1,3 AND 4 BIO 22

5 IMPLEMENTATION 25

5.1 Migration to Spring Boot 25

5.2 Exception handling 28

5.3 Logging 29

5.4 PostgreSQL connection 31

5.5 Dockerization: ATP45 and Postgis containers, compose file 33

5.5.1 ATP45 Docker container 33

5.5.2 Postgis container, compose file 34

5.6 CBRN 4 feature 38

6 CONCLUSION AND DISCUSSION 39

7 REFERENCES 40

8 ABBREVIATIONS 42

1 INTRODUCTION

Every country needs to be able to prepare for and respond to accidental or intentional disasters categorized as chemical, biological, radiological or nuclear (CBRN). It is also important to enhance the sophistication and analytic expertise in the process of planning for and responding to CBRN incidents. So, collection and sharing of information are the cornerstones in this process. Efficient distribution and utilization of correct information enable timely and properly targeted authority activities in response to CBRN accidents. Therefore, software which collects, analyses and distributes the data to the relevant users in a timely and clear manner is of an utter importance.

The North Atlantic Treaty Organization (NATO) specifies guidelines to provide such information in its messaging standard ATP-45. The thesis work will be devoted to improvement of the existing in Observis Oy software and addition of some essential features.

In the first chapter the general context of the work will be described. Also, definitions and explanations to the main terms, tools and technologies used in the process of the software development will be given. There will be a brief comparison of Java programming language and Spring Boot framework and why the latter one is more suitable for development of standalone microservice applications. The difference between Spring Boot JPA and Hibernate as tools for managing Java objects in relational databases and the main principles of REST API will be discussed. PostgreSQL databases and Docker containerization will also be mentioned.

The second chapter will explain the idea of ATP-45 standard for CBRN reporting and its goals, and also where Observis Oy stands in the field of CBRN and its situational awareness systems and software. The goals of the work will also be outlined in more detail.

A more detailed overview of the ATP-45 standard will be given including the information about when and how this standard is used and what types of data are collected, processed and distributed.

The third chapter will provide a detailed description of the implementation stage of the work.

A summary of the completed tasks and the outcomes of the work will be made in the conclusion.

2 DEVELOPMENT CONTEXT

Observis Oy (OOY) is aiming to provide world leading situational awareness systems (SAS) to international CBRN monitoring and preparedness markets. To support that purpose, there are certain key features that a SAS system should be capable of providing. One such feature is the support for NATO incident messaging standard ATP-45. Software that implements ATP-45 messaging and reporting standard can be used by authorities responsible for responses to CBRN-related accidents in both NATO member states and any other country which needs clear guidelines and specifications for implementation of sophisticated accident reporting system.

The current version of Observis situational awareness system (ObSAS) software provides a rudimentary support for basic ATP-45 messages, simple predictions based on AEP-45 implementation instructions as well as support for ADAT-3P message format for relaying these messages. However, there are several key features still missing from ObSAS ATP-45 implementation. Furthermore, the usability and integration to available measurement devices is lacking. This work is intended to study and provide solutions to some of those challenges.

The thesis work should cover the following themes:

1. The software providing ATP-45 support should be a standalone solution for a more flexible integration with ObSAS.

2. Integrating ObSAS measurement device data to ATP-45 system by allowing easy generation of ATP-45 CBRN-4 type messages from observed measurement values and location information.

As a result of the thesis the aforementioned topics are discussed, an updated version of ObSAS ATP-45 messaging component is developed providing a solution to these issues.

3 OVERVIEW OF JAVA AND SPRING BOOT REST API APPLICATIONS

This chapter will give a brief description and comparisons of tools and technologies used during the implementation of the thesis work. The original piece of software that supports ATP45 standard was written in Java programming language with OSGi framework. More modern approach with a more concise and functional solution was to use Java with Spring Boot framework instead. Also the use of Doker containers was chosen as it can allow easier transfer of the ATP45 software to further users.

3.1 Java and Spring boot

Spring Boot is a part of Java domain, more exactly it is a framework which assists in orchestrating Java application and dependent libraries and plugins into an executable environment in a more efficient way than plain Java applications. So, Spring Boot is a step forward to simpler architecture and dependency management.

Like mentioned before, one of the most powerful features of Spring Boot compared to basic Java applications is dependency management. Before introduction of the Spring Boot framework, software development required knowledge of the compatible dependencies necessary to make the application up and running. Spring Boot is a tool for automatic managing of dependencies and configurations. With Spring Boot, it is not necessary to mention the version of the dependency as the framework manages them on its own automatically. The second feature is auto-configuration, which means that Spring Boot automatically configures the application according to the dependencies. For example, If

PostgreSQL database dependency is mentioned on the classpath, and if database connection was not configured by the developer, then Spring Boot has an ability to auto-configure an in-memory application database. The third property is the design of standalone applications. Running a plain Java application involves four stages: package the application, choose the type of web server to run an application, configure the web server and organize the deployment process

With Spring Boot this process reduces to two steps, packaging the app and running the app. Spring Boot takes care of configuring and deploying an embedded web server. Another handy characteristic is embedded application servers, which do not require further installation to use. This also provides fast and efficient deployment. Summarizing, Spring Boot framework allows to avoid lot of boilerplate code and reduce the number of manual configurations.

3.2 REST API

First of all, the term API stands for 'application programming interface'. An API is a set of rules that outlines the way for a programmer to arrange the program for effective communication between client applications and a server.

REST stands for Representational State Transfer. REST is an architectural style and methodology frequently used in internet services development (Fielding, 2000). In more technical terms the idea behind REST API is that when a RESTful API is called, the server will transfer a representation of the requested resource's state to the client system (Figure 1).

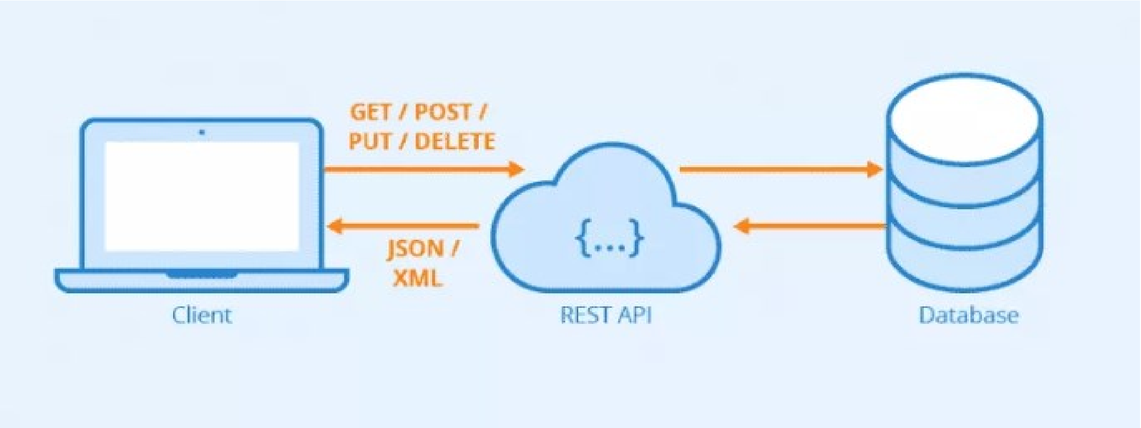


Figure 1. Data flow in a REST API application

REST APIs have several features that are especially advantageous in the case of use with web app services (Naeem, 2020). Firstly, REST APIs fetch data in several data formats (JSON or XML) that allow quicker processing than, for instance, Simple object access protocol (SOAP). SOAP being a protocol, has strict built-in rules which cause SOAP’s complexity and overhead to increase, which, in turn, may result in longer page load times. REST, on the other hand, is not an official protocol, but a set of architectural principles. It doesn’t require so much extra, overhead data, thus less bandwidth is used, making it more appropriate for web services. Secondly, RESTful protocols are weakly coupled, which means that parts of the application are independent, so there is flexibility and re-usability to add, substitute, or adjust them.

There are some terms that we often come across when talking about the REST APIs. REST APIs use the Request-Response system which uses HTTP protocol to fetch and deliver data. The following HTTP requests are used:

- GET (for fetching data)
- PUT (for altering the state of data)
- POST (for initial creation of data)
- DELETE (for elimination of data)

Before moving to the principles of the REST API design, it is important to distinguish the three key terms. First, resource – is any object the API can offer information about. It is the primary abstraction of information in REST. Each resource must have a distinct identifier, it can be either a name or a number. Identifier is important to distinguish resources involved in the communication between different elements. Second, server – is any system that stores the resources (data). Third, client is a hardware or software that uses the API provided by a server and utilizes data sent back to display information to the user.

So, the six principles that guide the REST API design are:
Principle one: client-server architecture. Client and server are separate entities that communicate via HTTP requests and responses. In the meantime, client and server can be modified, developed and scaled separately and independently.

Namely, it should be possible to make changes to the mobile application without impacting either the data structure or the database design on the server.

Principle two: statelessness. Client-server communication should be stateless, which means that client information is not stored between requests, therefore each request must contain all the essential data to complete itself successfully. REST API should not rely on data being stored on the server or sessions to determine what to do with a call, but rather solely rely on the data that is provided in that call itself. When all the state data is stored on the client and not on the server, it reduces memory requirements.

Principle three: cacheable data. The principle of statelessness causes the request overhead due to a large number of requests and responses. So, a REST API design requires to store cacheable data when possible, to reduce the number of interactions with the API. This means that when data is cacheable, the response should indicate that the data can be stored up to a certain time (expires-at), or in cases where data needs to be real-time, that the response should not be cached by the client. It should be noted that caching is done on the client side, the intent is to instruct the client on how it should proceed and whether the client can store the data temporarily.

Principle four: uniform interface. Uniformity implies that information is transferred in a standard form. The key to decoupling client from server is having a uniform interface that allows independent evolution of the application without having the application's services, models, or actions tightly coupled to the API layer itself. The uniform interface lets the client talk to the server in a single language, independent of the architectural backend of either.

Principle five: layered system. A layered system is comprised of layers, with each layer having a specific functionality and responsibility. Taking Model View Controller (MVC) framework as an example to illustrate the layered system: model defines how the data should be formed, the controller focuses on the incoming actions and the view focusing on the output. Thus, layers are separated but also interact with each other.

Principle six (optional): code-on-demand. Although most of the time, a server returns static resource representation in XML or JSON format, it is also possible to send executable code from the server to extend client's functionality.

To sum up, REST API is an architectural style which complies to five basic principles to provide better functionality to internet services. All of these constraints build a flexible API.

3.3 PostgreSQL databases, Spring JPA and Hibernate

The main terms discussed in this part will be Spring JPA, ORM and Hibernate. In most cases an app requires a connection to a database. To achieve that there is Spring Boot Java Persistence API (JPA), which is a Java specification for managing relational data in Java applications. In simple terms, a relational database keeps tables and records, while Java application keeps classes and objects, so it is necessary to convert, or map, one type of data structure to another. So, JPA allows to access and persist data between Java object/class and a relational database. JPA follows Object-Relation Mapping (ORM). As there is no internal feature supported by Java which offers mapping between class objects and database tables. So, when a java application connects to a relational database, when SQL queries are run, the received results need to be converted into Object instances. Apart from mapping Java objects to the database entities, JPA has another task of making queries to the database for all the user operations.

There is various enterprises vendor such as Eclipse, RedHat and Oracle that provide products by adding the JPA in them. There are some popular JPA implementations frameworks and one of them is Hibernate. So, to distinguish between JPA and Hibernate. JPA is a general specification that defines the ways to access, manage, and persist data between Java object and relational database, it defines ORM as a standard approach. Hibernate is one of the JPA implementations and an ORM tool.

Although there is a wide choice of relational database management systems (RDBMSs) - SQLite, MySQL, and PostgreSQL. In the case of ATP-45 software PostgreSQL was chosen.
As opposed to its counterparts, PostgreSQL has a high priority to data integrity (Nguyen, 2021). It implements control to ensure that data remains consistent. Also, PostgreSQL is compatible with a different programming languages. This means that migration of the database to another operating system or integration with a specific tool, is a rather uncomplicated process with a PostgreSQL. Moreover, PostgreSQL has strong support for complex operations like data storage and online transaction processing. Given that ATP-45 functionality may require adjustments to the specific needs of a particular client purchasing the ObSAS solution and sensitivity of the stored data, PostgreSQL is the best choice because of its reliability of data consistency and options for integration with other tools.

3.4 Docker

ATP-45 software requires an easy way to be transferred and installed on different new systems. For easier relocation it is reasonable to use Docker containers. Containerization is a type of virtualization. But as opposed to virtualization, which is the technique of importing guest operating system on top of a host operating system that eliminates the need for extra hardware resource, containerization is the technique of bringing virtualization to the operating system level (Pandit, 2021). While virtualization brings abstraction to the hardware, containerization brings abstraction to the operating system (Figure 2).

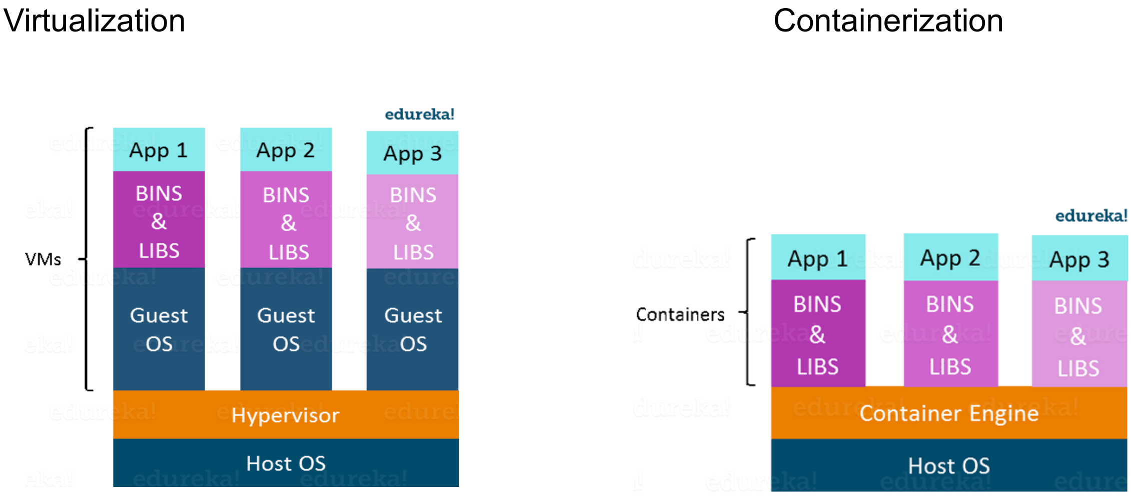


Figure 2. Comparison of virtualization and containerization

This containerization feature guarantees that the application works in any environment. So, a developer can build a container having an application or different applications installed on it and pass it to further users. The users then would only need to run the container to replicate the developer's environment.

A Dockerfile is a text document which contains all the commands to assemble an image. Upon issue of the 'docker build' command by the developer, Docker file becomes a Docker Image. Docker image is a kind of template which is used to create actual Docker container. And when a Docker image is run, it becomes a Docker Container. Docker Container is a running instance of a Docker Image as they hold the entire package needed to run the application (Figure 3) ().

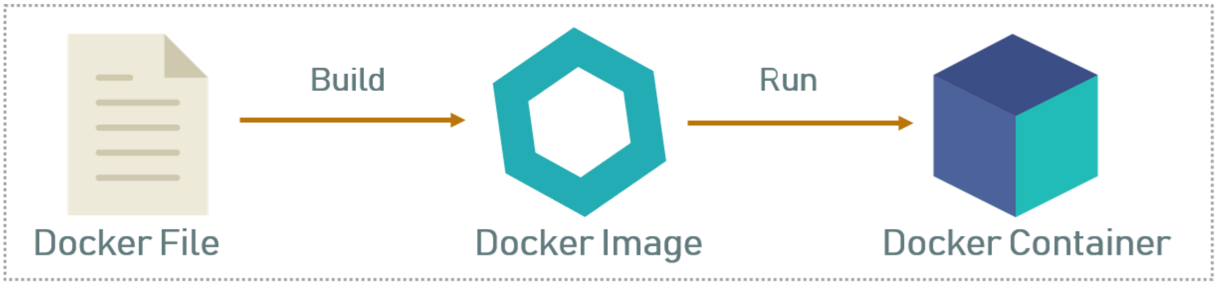


Figure 3. Process of creation of a Docker container from Dockerfile

One of the important notions regarding Docker containers are microservices. The idea behind those microservices is that certain types of applications become easier to build and maintain when they are broken down into smaller, composable pieces which work together (). Each component is developed separately, and the application is then simply the sum of its constituent components. For example, an online shop can be divided into separate microservices for shop user accounts, product catalogues, list of orders and processing statuses. So, if one of the microservices fails, the entire application can continue working and it is much easier to manage in general.

4 OVERVIEW OF ATP45 NATO STANDARD

This chapter describes the idea of NATO ATP45 standard. First, the documents that give definitions to CBRN related terms and procedures are presented. Then, definitions of the threats are given, followed by description of each of six types of CBRN messages. Then data flow between different levels of messages is illustrated and explained. Also, the importance of weather data in assessment of CBRN incident-related hazard areas is highlighted.

4.1 ATP45 standard and documentation: ATP45 and AEP45

Defense against Chemical, Biological, Radiological and Nuclear (CBRN) weapons and incidents have usually been classified as one subject area within NATO military publications. All these kinds of attacks are grouped together because they have similar features: they affect wide geographic areas; there are large numbers of people potentially at risk and certain similarities in protection measures; and the supposed ‘novelty’ of each of these classes of science-derived weapons (ATP45 (operators manual) 2014). The main users of the CBRN related information are national authorities, NATO command centers and command forces as well as local civilian authorities.

To improve coordination between different groups of authorities regarding CBRN activities and spread of data upon an occurrence, some standards were developed. The two of most important documents of the NATO standard are the ‘ATP45 Operators Manual’ and ‘AEP45 Reference Manual’ documents.
The purpose of ‘ATP45 Operators Manual’ is to prescribe the CBRN procedures to be followed by land, air and naval forces for the:

- Reporting of CBR attacks, nuclear detonations and resulting contamination
- Predicting and warning of hazard areas from CBRN incidents.
- Assessment of CBRN information
- Warning of allies about the forthcoming nuclear strikes or interception of an adversary incoming missile
- Alerting about potential CBRN incidents
- Interchange of reports

AEP45 Reference Manual provides requirements and functional specifications for CBRN informational constructs, activities and functionality necessary for the procedures mentioned in the previous paragraph.

Summarizing, ‘ATP45 Operators Manual’ is a higher-level document that describes general approach and procedures of warning and reporting the incidents, while ‘AEP45 Reference Manual’ gives more detailed guidelines for calculations and predictions of hazard areas, speed of spread etc.

4.2 Definition of CBRN threats

CBRN threats refer to hazardous incidents caused by chemical substances (C), biological pathogens (B), radioactive materials (R), nuclear weapons (N) as well as by the misuse of expertise related to these. No one would argue that in case of such serious matters it is important to maintain general situational awareness, detect and prevent intentional and unintentional CBRN incidents. In the case of occurrence, it is crucial to be prepared and to spread information in a concise, clear and timely manner. It is important to all the parties who are in charge of making decisions and implementing risk-reducing measures from higher authorities to the first responders.

Chemical. A chemical weapon or device is any material or item that projects, disperses, or disseminates a chemical substance (ATP45 Operators Manual, 2014).

Biological. A biological weapon or device is any item or material, which projects, disperses, or disseminates a biological agent including arthropod vectors. Arthropod vectors refer to living organisms that transmit an infectious agents from an infected animal to a human or another animal. Such arthropods are frequently mosquitoes, ticks, flies, fleas and lice (ATP45 Operators Manual, 2014).

Radiological. A radiological device is designed to employ radioactive material by disseminating it to cause destruction, damage or injury by means of the radiation produced by the decay of such material (ATP45 Operators Manual, 2014).

Nuclear. A nuclear weapon is a type of weapon that produces nuclear reaction and release of energy after prescribed arming, fusing and firing sequence. Thus, the 2 distinctive features of such weapons are: effect of this kind of weapons is derived from physical energy and some of it is released in the form of radioactivity (as compared to chemical and biological weapons, for example).

4.3 CBRN1-6 messages and data flow overview

Purpose of a CBRN Warning and Reporting System is to provide pre-incident and post-incident information. Pre-incident information may include friendly nuclear strike warnings (STRIKWARN) about an anticipated strike from allies and hazardous material release warnings to friendly Forces (HAZWARN). Post-incident information may include missile intercept reports, operationally pertinent facts about CBRN incidents and their effects and assessment of the CBRN situation and hazard areas.

The next part will describe different types of CBRN messages which will be essentially necessary to understand the working principles of ATP45 software.

As already mentioned before, there are four distinct types of messages depending on the type of threat – chemical substances (C), biological pathogens (B), radioactive materials (R), nuclear weapons (N). Each of these types can be further divided into six levels, level one being the most general information about the hazardous event and level six – the most detailed description.

CBRN 1 is a report giving basic data about an incident that was obtained by a human observer. In simple words, CBRN 1 is the most basic level of CBRN messages, it is some data reported by people in the field about what they saw. For example, an official observer in duty can report blasts, with approximate estimation of the distance and direction of the incident. This report provides the observer’s initial report if the location of the source is known. Processing of

CBRN 1 reports results in identification of clusters of messages that belong to the same incident (clustered by time and location) and generation of CBRN 2 messages from these clusters.

CBRN 2 is a report for passing the evaluated data from collected CBRN 1 reports (ATP45 Operators Manual, 2014). CBRN 2 messages present a minimal analysis of data from CBRN 1 messages. For example, data from two CBRN 1 messages is analyzed to determine a more precise place, time, size and kind of strike and released substance.

CBRN 3 is again a more detailed report than the previous type. Two or more CBRN 2 messages are correlated together to form one CBRN 3 message. It provides data about predicted contamination and hazard areas. Precise instructions for the calculations of contaminated and hazard areas are given in the 'ATP45 Operators Manual'.

CBRN 4 messages are a more sophisticated type of messages. Data for this report is collected by different sensors and detectors. This report is used for two cases. Case one, an attack is not observed, and the first indication of CBRN contamination is by detection. Case two, report measured contamination as a part of a survey or monitoring team.

CBRN 5 is a report for passing information on areas of actual contamination. This report can include areas of possible contamination, but only if actual contamination co-ordinates are included in the report. CBRN 5 messages provide contour information on air or ground contamination. This information allows commanders to determine appropriate contamination avoidance measures, such as protecting troops, or decontamination needs.

CBRN 6 is a report for passing detailed information on CBRN incidents (ATP45 Operators Manual, 2014). It is usually submitted only when requested.

To understand the process of generation of CBRN messages, it is also important to understand the general data flow in a CBRN system – Command Information Systems (CIS). The processing can be either automated or manual or a mix of both. At the highest level the data flows into and out of the Automated CBRN CIS from various sources. First, there are several types of input data:

a. Environment and battlefield data related to specific incidents.

b. Meteorological data on current and forecast conditions

c. Situational information including tactical data and current objectives

d. Specific user inputs and requests

It should be specifically noted that initial CBRN incident data must not be in the form of CBRN messages as these messages are generated by the Automated CBRN system. The initial data is usually entered manually in a user interface which requests particular values for specific phenomena or is sent from devices. Only after that the ATP45 system generates messages in a standard ATP45 format.

The output data can be in three forms: CBRN warnings, CBRN situational information or responses to user inputs and requests.

The diagram in Figure 4 illustrates data flow for radiological (RAD) messages 1-6:

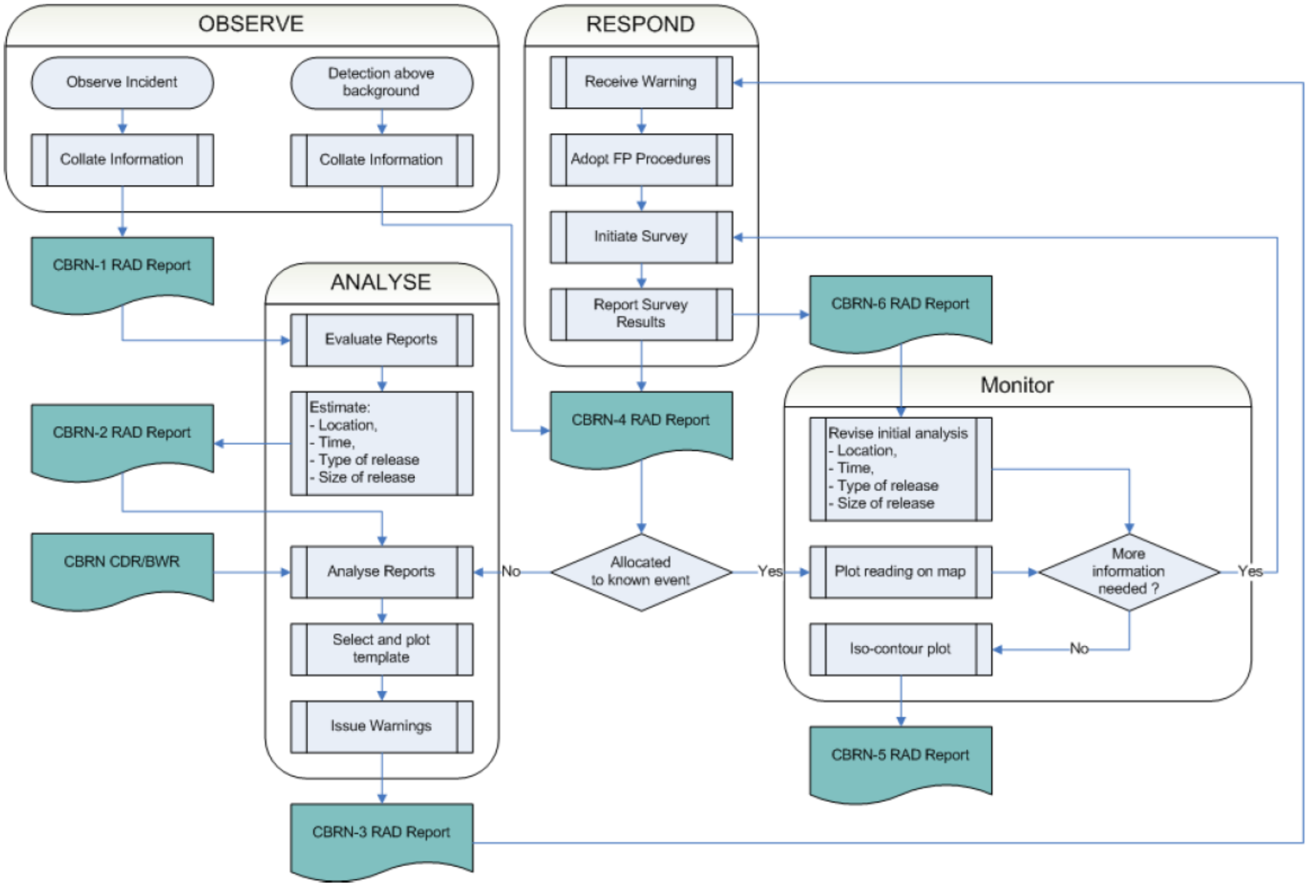


Figure 4. Warning and reporting flow chart (radiological incidents)

4.4 CBRN messages and weather reports

Weather conditions are one of the most important factors when building a prediction model for contaminated and hazard areas and any sort of impact by CBRN incident. Detailed weather information such as temperature, humidity, wind speed, wind direction, precipitation, air stability category, sunlight and air exposure must be obtained to determine the effects of a CBRN hazard on surrounding environment of the incident and how fast and far it will spread (ATP45 Operators Manual, 2014).

In CBRN defense it is essential to have up-to-date meteorological data or weather forecast. This information is essential for calculations of predicted hazard areas and for protection of anyone downwind. Below are the most common meteorological terms within ATP-45:

- Temperature. This indicator is important because the rate of evaporation of a liquid chemical agent or toxic industrial chemical varies with the temperature. High temperatures increase the speed of evaporation and lower temperatures diminish it. Lower temperatures have the opposite effect. Lower temperatures may also reduce or even eliminate casualty potential. However, a contact hazard may remain for several days or weeks. On the other hand, temperature is not expected to have any significant effect on the hazard area resulting from a biological, radiological or nuclear release (ATP45 Operators Manual, 2014).

- Air Stability Category. The air stability describes how easily the released agent can be mixed with the air in the lower atmosphere. There exist three general air stability categories:
(1) Stable. Under stable conditions there is little blending of air and released agent and, thus, higher concentrations of the agent can be detected. Also, the agent cloud will be harmful over long distances.
(2) Neutral. Neutral conditions are the most common type, the define the intermediate range for mixing.
(3) Unstable. Under unstable conditions there is strong mixing and thus smaller hazard areas (ATP45 Operators Manual, 2014).

- Wind. The wind speed and direction impact the spread of contamination. High winds can increase the rate of evaporation of liquid chemical agents and the rate at which chemical clouds are dissipated.

- Downwind direction is the direction towards which, the airborne cloud travels in the hazard area.

- Humidity and Precipitation. Humidity and precipitation change the effects of chemical agents in different ways. High humidity, for instance, increases the effectiveness of blister agents, but will not directly influence the effectiveness of nerve agents. Humidity will alter the effects of biological agents in different ways. Very low humidity decreases the strength by increasing the rate at which agents dry out from atmospheric exposure. Heavy or continuous rain will wash away liquid chemical contamination, and light rain after a liquid release can cause the recurrence of a contact hazard. Rain after a blister or persistent nerve agent release will temporarily increase the evaporation rate, thus increasing the vapour hazard. Snow reduces the evaporation rate of liquid chemical agents, thus reducing the vapour hazard in the release area. Heavy or continuous rain will locally reduce nuclear and biological contamination by washing it out of the air (ATP45 operators manual, 2014).

- Height. In most cases increase in height above the ground level, decreases the concentration of the agent and low concentration is reached at approximately 800 metres altitude. Usually areas at altitudes above 3000 metres above ground are safe.

- Sunlight and Air Exposure. Most biological agents lose their viability or toxicity with time after exposure to the atmosphere. Most biological agents will have a greater rate of loss of viability or toxicity when exposed to bright sunlight (ATP45 Operators Manual, 2014).

Impact of Terrain. The path and speed of airborne clouds is significantly influenced by the terrain in the downwind area. Contaminant clouds can flow over rolling terrain, down valleys and around structures such as buildings. Dangerous concentrations may persist in hollows, depressions and trenches. The contaminant clouds tend to go over or around obstacles such as hills, but tend to be retarded by rough ground, tall grass and bushes. Flat terrain allows for an even, steady movement. (ATP45 operators manual, 2014)

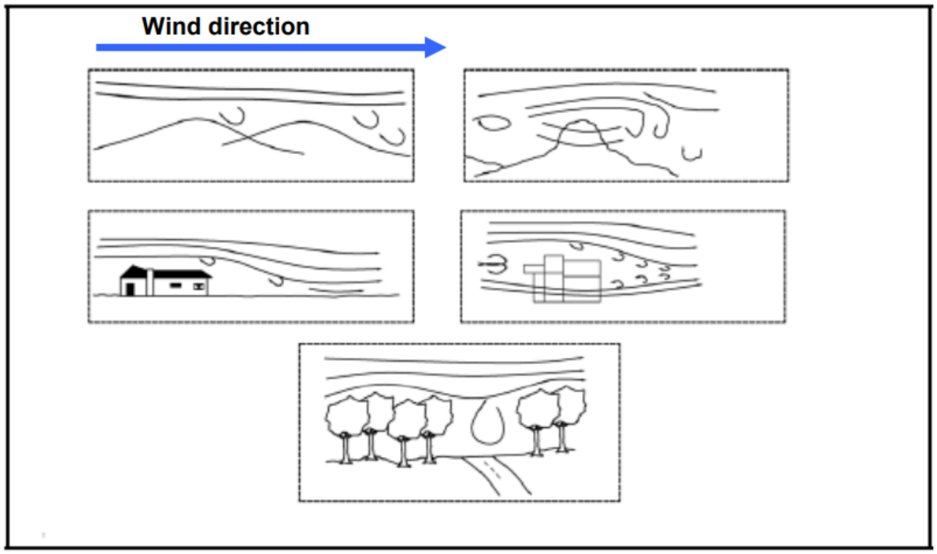


Figure 5. Typical Effects of Terrain and Structures on Wind Patterns

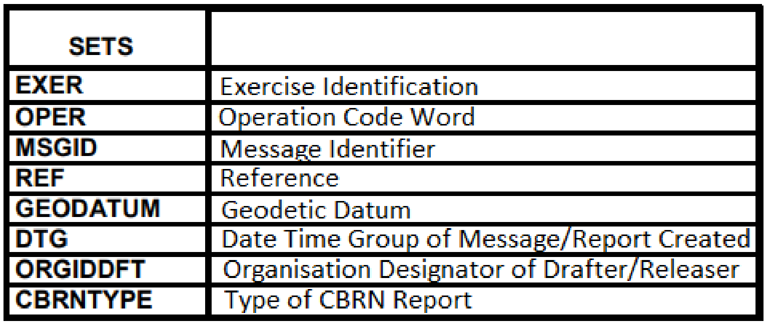
The movements of air are depicted in Figure 5 as lines and hollows, depressions and trenches are illustrated as semi-circles.

4.5 Structure of a CBRN message

Each CBRN message consists of a heading and a collection of data sets. Message heading is the same for every kind of CBRN message and includes the information about location, time and date, type of message and identification items. As opposed to message heading, which is standard for all messages, collection of sets depends on the message type. For example, nuclear incidents of level 4 include a specific set for passing data about detected radiation levels.

4.5.1 Message heading

A standard ATP-45 message consists of a heading and a collection of predefined sets. For different types (chemical, biological, radiological and nuclear) of messages, and for different levels of messages (1 – 6) there are different collections of sets to be used to make reports.
Message heading consists of different SETS, for example ‘EXER’, ‘OPER’, etc. (Figure 6), which has general identification information about the event.

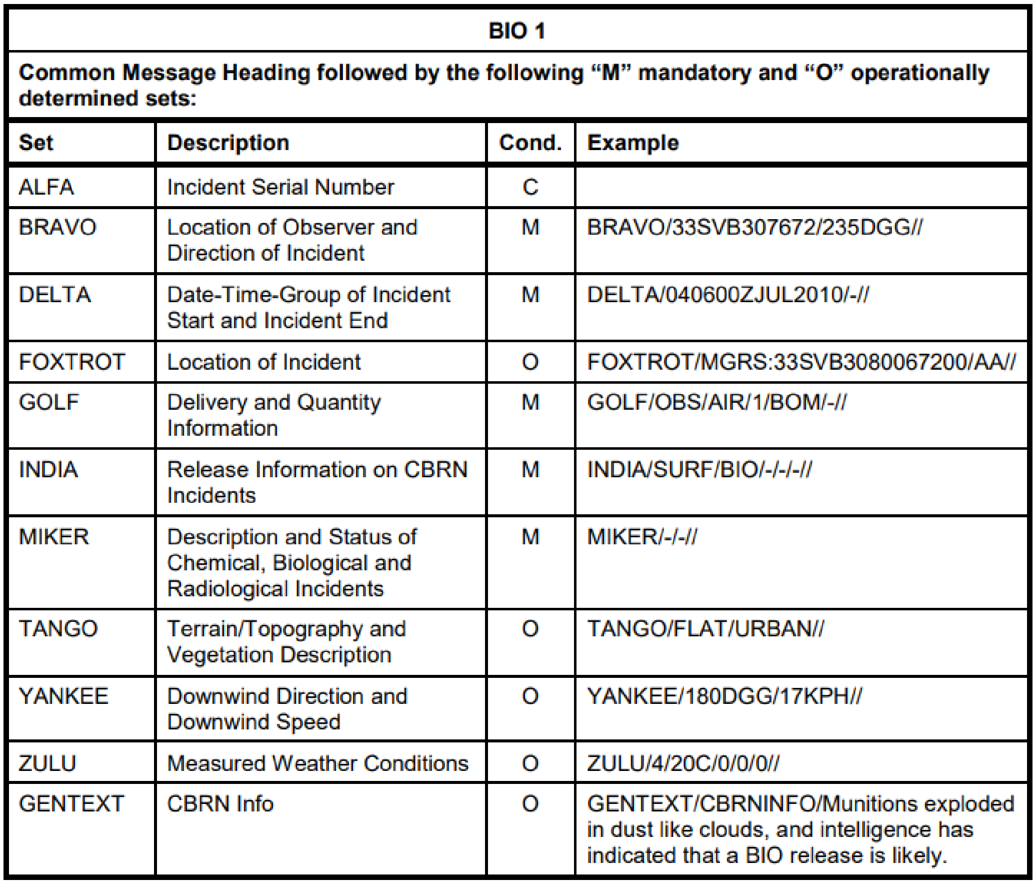


SETS	
EXER	Exercise Identification
OPER	Operation Code Word
MSGID	Message Identifier
REF	Reference
GEODATUM	Geodetic Datum
DTG	Date Time Group of Message/Report Created
ORGIDDFT	Organisation Designator of Drafter/Releaser
CBRNTYPE	Type of CBRN Report

Figure 6. Messages Heading

4.5.2 Message sets – example of CBRN 1,3 AND 4 BIO

Next, CBRN 1 BIO message – observer's initial report about an incident converted into ATP-45 message – is described (Figure 7).



BIO 1

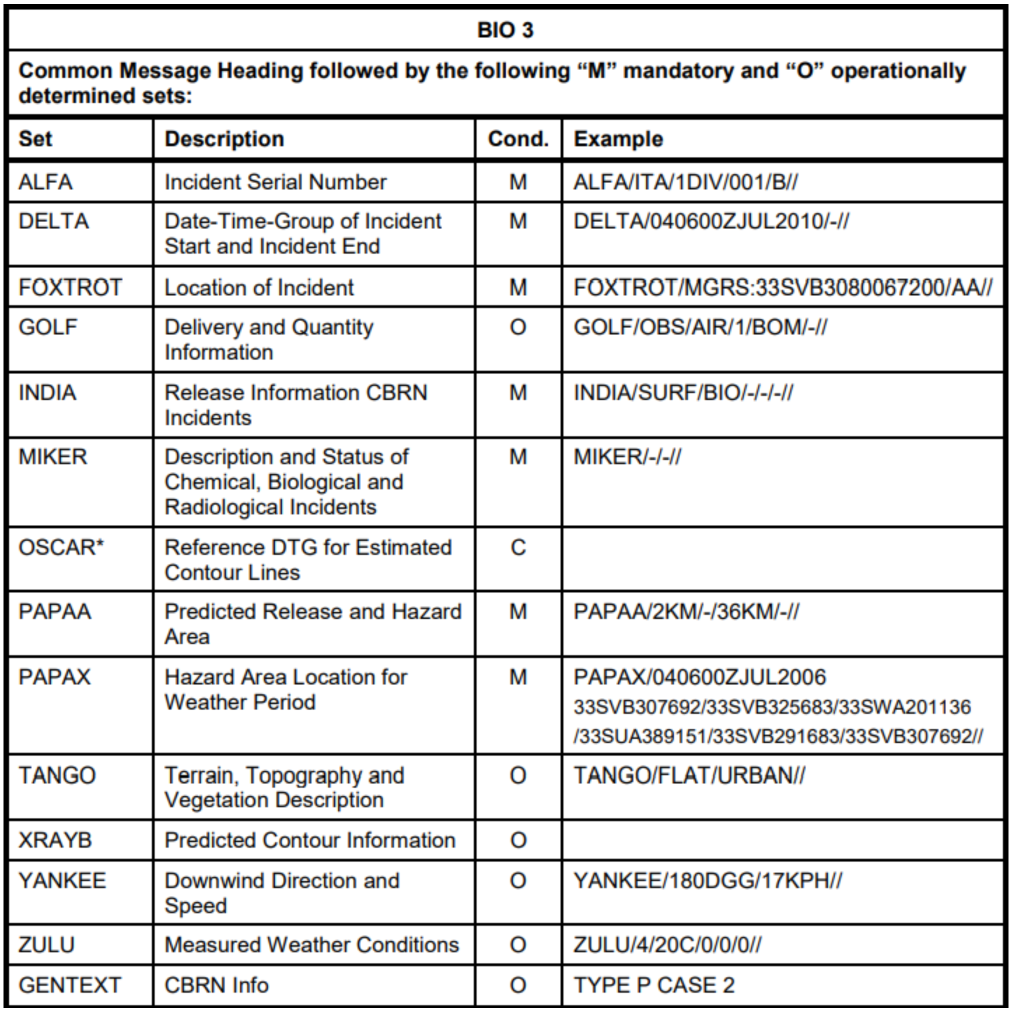
Common Message Heading followed by the following "M" mandatory and "O" operationally determined sets:

Set	Description	Cond.	Example
ALFA	Incident Serial Number	C	
BRAVO	Location of Observer and Direction of Incident	M	BRAVO/33SVB307672/235DGG//
DELTA	Date-Time-Group of Incident Start and Incident End	M	DELTA/040600ZJUL2010/-//
FOXTROT	Location of Incident	O	FOXTROT/MGRS:33SVB3080067200/AA//
GOLF	Delivery and Quantity Information	M	GOLF/OBS/AIR/1/BOM/-//
INDIA	Release Information on CBRN Incidents	M	INDIA/SURF/BIO/-/-/-//
MIKER	Description and Status of Chemical, Biological and Radiological Incidents	M	MIKER/-/-//
TANGO	Terrain/Topography and Vegetation Description	O	TANGO/FLAT/URBAN//
YANKEE	Downwind Direction and Downwind Speed	O	YANKEE/180DGG/17KPH//
ZULU	Measured Weather Conditions	O	ZULU/4/20C/0/0/0//
GENTEXT	CBRN Info	O	GENTEXT/CBRNINFO/Munitions exploded in dust like clouds, and intelligence has indicated that a BIO release is likely.

Figure 7. CBRN 1 BIO Message

Sets in the main part of the message are called I accordance with NATO phonetic alphabet – ALFA, BRAVO, etc. This alphabet is a kind of code where 26 letters of English alphabet are assigned 26 words, so that the names for letters would be as distinct as possible so as to be easily understood by those who exchanged voice messages by radio or telephone, regardless of language differences or the quality of the connection. For instance, set FOXTROT contains data about the location of an incident in a specific format. Set GENTEXT (which is not a part of NATO phonetic alphabet) provides other information, which is not specified in other standard sets.

Next is CBRN 3 BIO message – immediate warning of expected contamination of hazard area (Figure 8). It contains more sets than CBRN 1 BIO (although not of the are mandatory to fill in) which give more information about expected hazard areas, for example in sets PAPAA and XRAYB - ‘Predicted Release and Hazard Area’ and ‘Predicted Contour Information’ correspondingly.



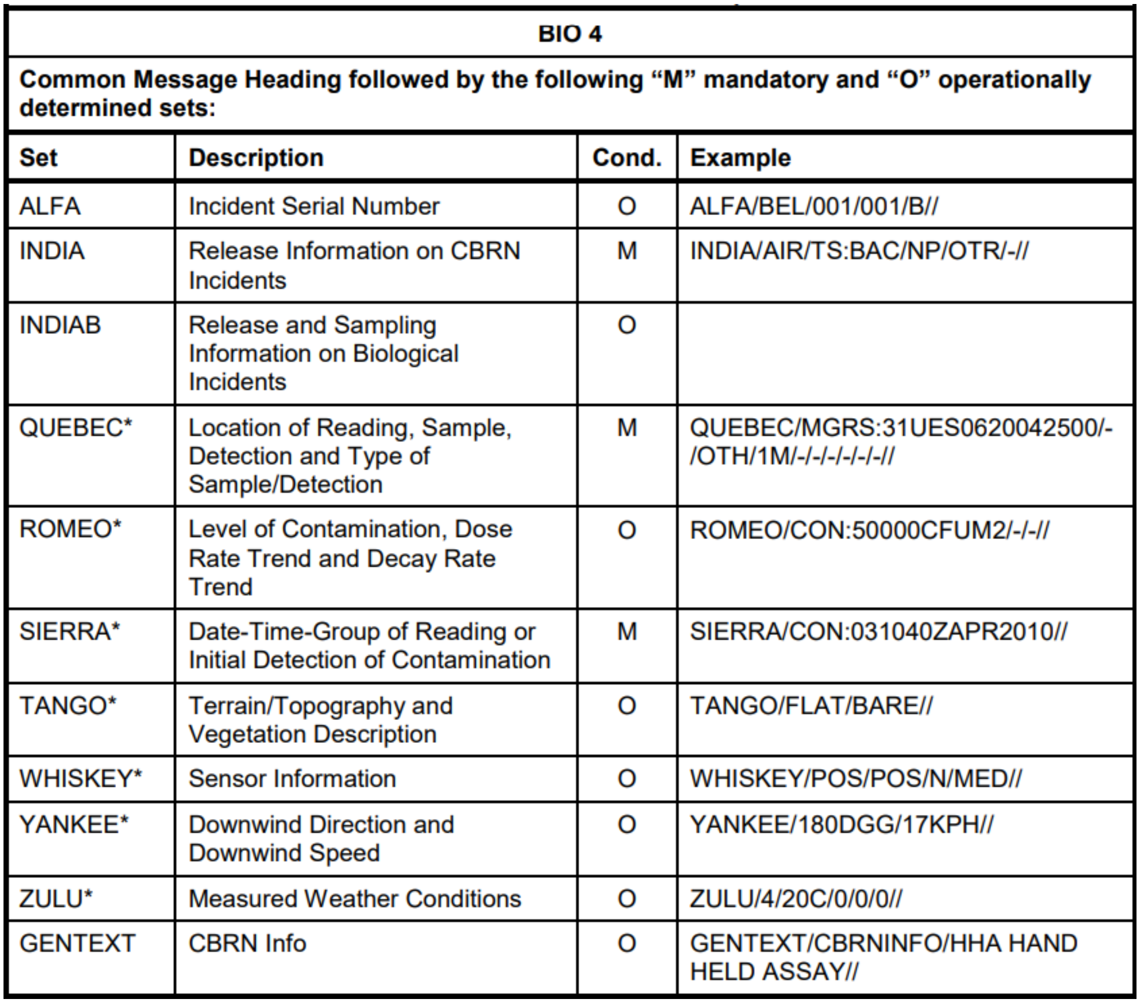
BIO 3

Common Message Heading followed by the following “M” mandatory and “O” operationally determined sets:

Set	**Description**	**Cond.**	**Example**
ALFA	Incident Serial Number	M	ALFA/ITA/1DIV/001/B//
DELTA	Date-Time-Group of Incident Start and Incident End	M	DELTA/040600ZJUL2010/-//
FOXTROT	Location of Incident	M	FOXTROT/MGRS:33SVB3080067200/AA//
GOLF	Delivery and Quantity Information	O	GOLF/OBS/AIR/1/BOM/-//
INDIA	Release Information CBRN Incidents	M	INDIA/SURF/BIO/-/-/-//
MIKER	Description and Status of Chemical, Biological and Radiological Incidents	M	MIKER/-/-//
OSCAR*	Reference DTG for Estimated Contour Lines	C	
PAPAA	Predicted Release and Hazard Area	M	PAPAA/2KM/-/36KM/-//
PAPAX	Hazard Area Location for Weather Period	M	PAPAX/040600ZJUL2006 33SVB307692/33SVB325683/33SWA201136 /33SUA389151/33SVB291683/33SVB307692//
TANGO	Terrain, Topography and Vegetation Description	O	TANGO/FLAT/URBAN//
XRAYB	Predicted Contour Information	O	
YANKEE	Downwind Direction and Speed	O	YANKEE/180DGG/17KPH//
ZULU	Measured Weather Conditions	O	ZULU/4/20C/0/0/0//
GENTEXT	CBRN Info	O	TYPE P CASE 2

Figure 8. CBRN 3 BIO Message

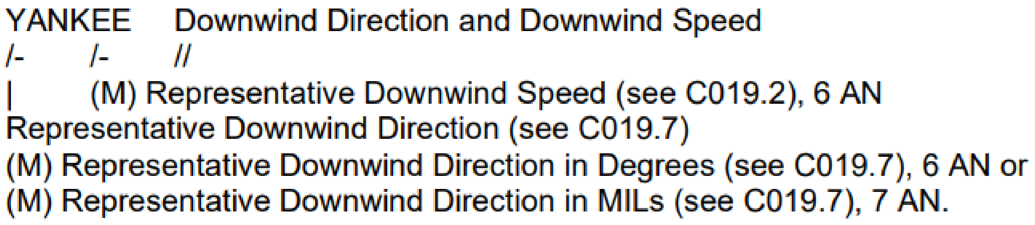
CBRN 4 BIO message is the type of message which reports reconnaissance, monitoring and survey results, which in many cases implies reporting of detector and sensor readings values (Figure 9). So, CBRN 4 BIO report has a set WHISKEY for reporting sensor information. As there may be several kinds of sensors for one incident, this field can repeat in one report up to 20 times. But the sensor data would be rather useless without a corresponding location (set QUEBEC) and timestamp (set SIERRA). Thus, some sets can form ‘segments’ which can repeat only together.



BIO 4			
Common Message Heading followed by the following "M" mandatory and "O" operationally determined sets:			
Set	**Description**	**Cond.**	**Example**
ALFA	Incident Serial Number	O	ALFA/BEL/001/001/B//
INDIA	Release Information on CBRN Incidents	M	INDIA/AIR/TS:BAC/NP/OTR/-//
INDIAB	Release and Sampling Information on Biological Incidents	O	
QUEBEC*	Location of Reading, Sample, Detection and Type of Sample/Detection	M	QUEBEC/MGRS:31UES0620042500/-/OTH/1M/-/-/-/-/-/-//
ROMEO*	Level of Contamination, Dose Rate Trend and Decay Rate Trend	O	ROMEO/CON:50000CFUM2/-/-//
SIERRA*	Date-Time-Group of Reading or Initial Detection of Contamination	M	SIERRA/CON:031040ZAPR2010//
TANGO*	Terrain/Topography and Vegetation Description	O	TANGO/FLAT/BARE//
WHISKEY*	Sensor Information	O	WHISKEY/POS/POS/N/MED//
YANKEE*	Downwind Direction and Downwind Speed	O	YANKEE/180DGG/17KPH//
ZULU*	Measured Weather Conditions	O	ZULU/4/20C/0/0/0//
GENTEXT	CBRN Info	O	GENTEXT/CBRNINFO/HHA HAND HELD ASSAY//

Figure 9. CBRN 4 BIO Message

Each set, in turn, can be divided into field. For example, set YANKEE consists of two mandatory fields – 'Downwind direction' and 'Downwind speed' (Figure 10).



```
YANKEE     Downwind Direction and Downwind Speed
/-      /-      //
|       (M) Representative Downwind Speed (see C019.2), 6 AN
Representative Downwind Direction (see C019.7)
(M) Representative Downwind Direction in Degrees (see C019.7), 6 AN or
(M) Representative Downwind Direction in MILs (see C019.7), 7 AN.
```

Figure 10. Set YANKEE and its fields

Each value in the set has a defined meaning and a particular place, values are separated by a '/' sign.

5 IMPLEMENTATION

This chapter describes a step-by-step process of migration of the existing software written in Java code using OSGi framework to Spring Boot framework. Next, a few specifics about exception handling and logging in Spring Boot are reviewed. Also, the most interesting features of PostgreSQL database connection are emphasized. Next, the process of dockerization is described. Another significant part of the thesis implementation is addition of support for CBRN 4 messages. The most crucial features about the added CBRN 4 functionality are also highlighted.

5.1 Migration to Spring Boot

Apart from configuring dependencies in a .pom file and creating the main method with appropriate Spring Boot annotation - @SpringBootApplication - the first steps to build a Java REST app with Spring Boot is to think of the software structure. There are four layers in Spring boot:

- Presentation layer which consists of the front-end part and handles the HTTP requests. This request is then authenticated and transferred to the business layer.
- Business layer which handles the business logic and performs authentication and validation.
- Persistence layer contains the storage logic that translates the business object to database rows.
- Database layer performs the CRUD (Create, Retrieve, Update and Delete) operations

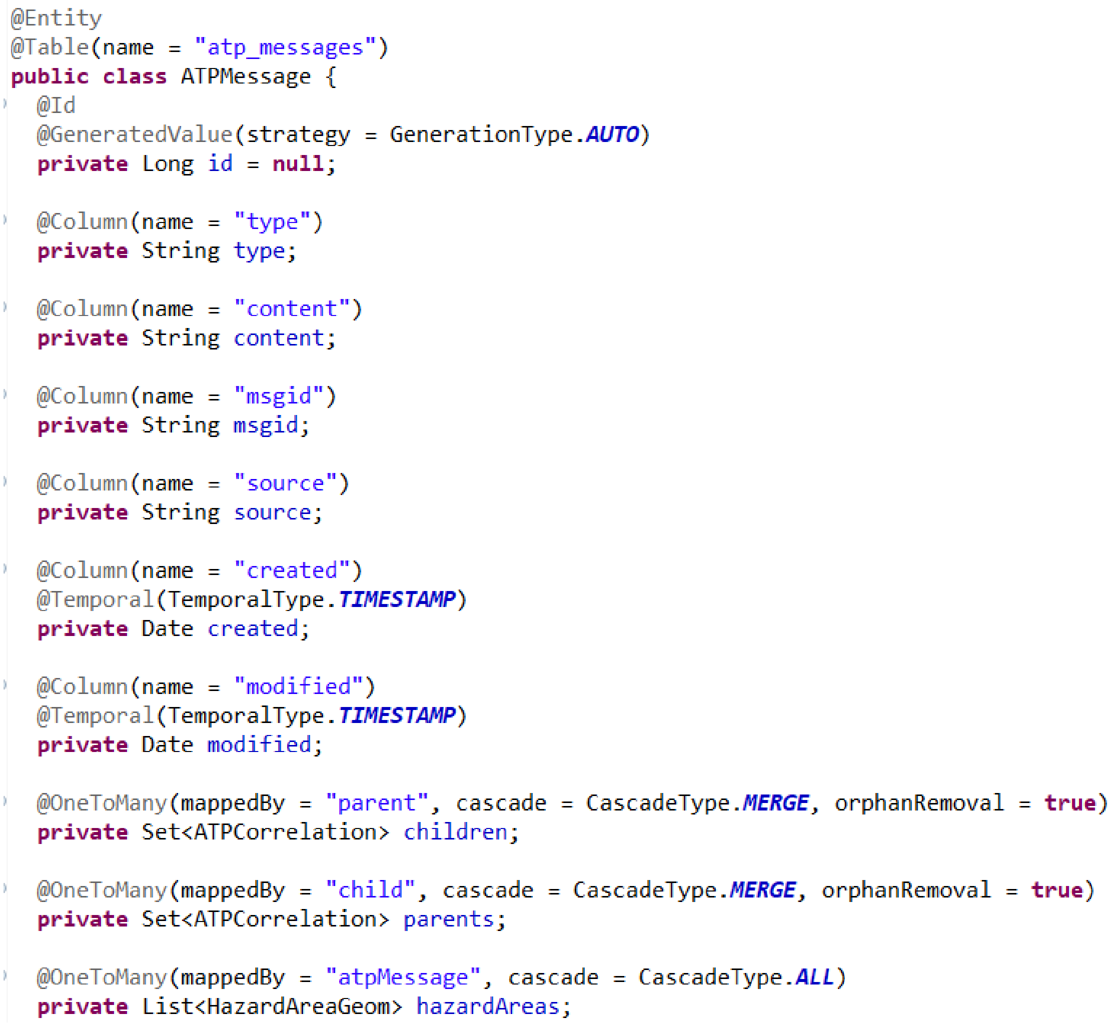
Here is a simplified structure for ATP45 Message endpoint. Same structure is used for other endpoints, for example, ATP45 hazard areas.

```
+- ObSAS

  +- ATP45Application.java

  |

  +- ATPMessage
```

```
|   +- ATPMessageEntity.java

|   +- ATPMessageRepository.java

|   +- ATPMessageService.java

|   +- ATPMessageController.java
```

The cornerstone classes of a Spring Boot app are entities (persistence layer), repository (database layer), controller (presentation layer) and service (business layer).

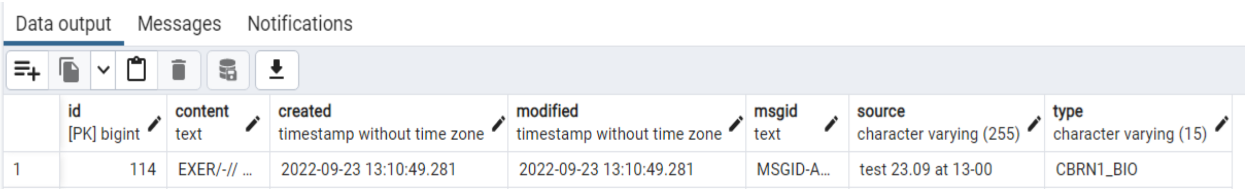
ATP45Message Entity is a structure in which the data about the message will be stored in the database (Figure 11).



```
@Entity
@Table(name = "atp_messages")
public class ATPMessage {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id = null;

  @Column(name = "type")
  private String type;

  @Column(name = "content")
  private String content;

  @Column(name = "msgid")
  private String msgid;

  @Column(name = "source")
  private String source;

  @Column(name = "created")
  @Temporal(TemporalType.TIMESTAMP)
  private Date created;

  @Column(name = "modified")
  @Temporal(TemporalType.TIMESTAMP)
  private Date modified;

  @OneToMany(mappedBy = "parent", cascade = CascadeType.MERGE, orphanRemoval = true)
  private Set<ATPCorrelation> children;

  @OneToMany(mappedBy = "child", cascade = CascadeType.MERGE, orphanRemoval = true)
  private Set<ATPCorrelation> parents;

  @OneToMany(mappedBy = "atpMessage", cascade = CascadeType.ALL)
  private List<HazardAreaGeom> hazardAreas;
```

Figure 11. Entity class

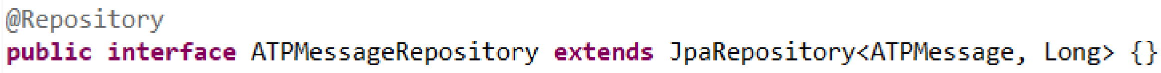
This data structure corresponds to a PostgreSQL database representation (Figure 12):



	id [PK] bigint	content text	created timestamp without time zone	modified timestamp without time zone	msgid text	source character varying (255)	type character varying (15)
1	114	EXER/-// ...	2022-09-23 13:10:49.281	2022-09-23 13:10:49.281	MSGID-A...	test 23.09 at 13-00	CBRN1_BIO

Figure 12. PostgreSQL table

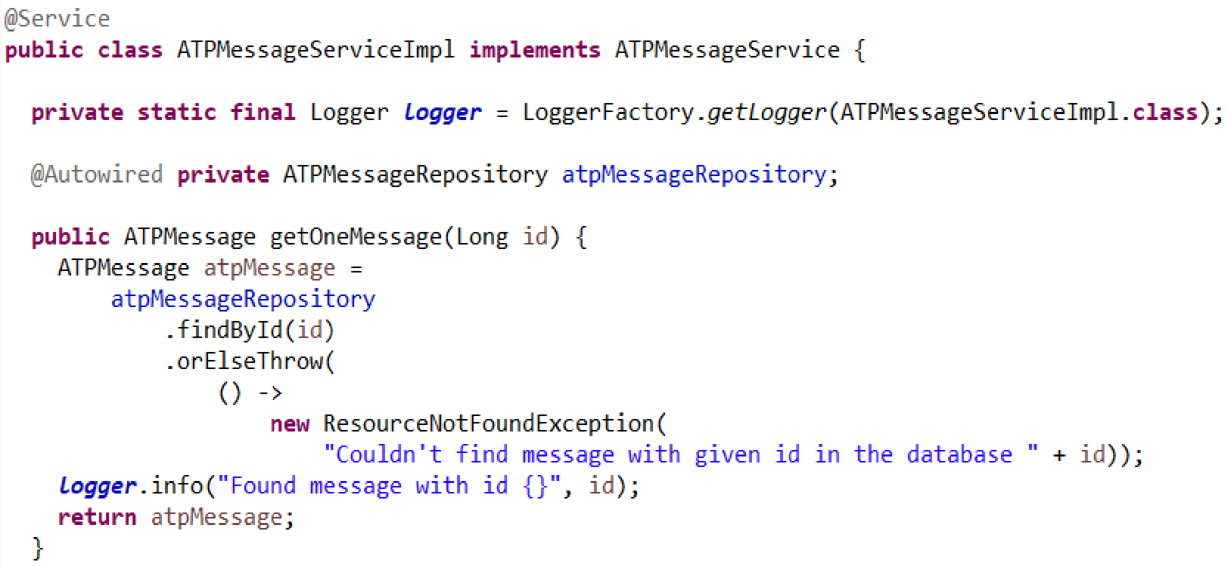
These two things, the Spring Boot class and the database representation are correlated by persistence, repository layer. Repository is a standard class which utilizes the Java Persistence API (JPA) - persistence standard of the Java ecosystem. It allows us to map Java data models directly to the database structures and then gives the flexibility of manipulating objects in the code (Figure 13).



```
@Repository
public interface ATPMessageRepository extends JpaRepository<ATPMessage, Long> {}
```

Figure 13. Repository class

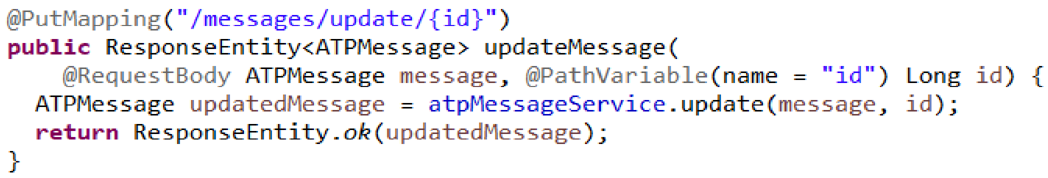
The service layer could be combined with the controller level, because they are basically two steps of the same process. But it is highly recommended to keep them as two separate layers for three reasons - it provides separation of concern, security and loose coupling (Figure 14).



```
@Service
public class ATPMessageServiceImpl implements ATPMessageService {

  private static final Logger logger = LoggerFactory.getLogger(ATPMessageServiceImpl.class);

  @Autowired private ATPMessageRepository atpMessageRepository;

  public ATPMessage getOneMessage(Long id) {
    ATPMessage atpMessage =
        atpMessageRepository
            .findById(id)
            .orElseThrow(
                () ->
                    new ResourceNotFoundException(
                        "Couldn't find message with given id in the database " + id));
    logger.info("Found message with id {}", id);
    return atpMessage;
  }
```

Figure 14. Service class

The pattern with service layer and controller layer separated reduces the risk of exposing database connection to the controller class (Figure 15).



```
@PutMapping("/messages/update/{id}")
public ResponseEntity<ATPMessage> updateMessage(
    @RequestBody ATPMessage message, @PathVariable(name = "id") Long id) {
  ATPMessage updatedMessage = atpMessageService.update(message, id);
  return ResponseEntity.ok(updatedMessage);
}
```

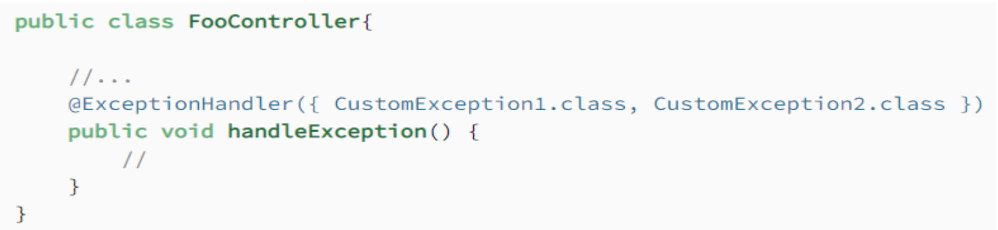
Figure 15. Controller class

5.2 Exception handling

There are several ways to implement exception handling with Spring for a REST API. But the one used in ATP45 software is global exception handling. It ensures that all exceptions are handled in a consistent manner and the response returned is of a specific format.

Before introduction of @ControllerAdvice annotation for global exception handling, there were other methods. Two of those approaches are @ExceptionHandler annotation and the HandlerExceptionResolver, but both of them have some clear downsides.

As for the first method, @ExceptionHandler, it is a controller level method. The method annotated with @ExceptionHandler is only active for that particular Controller, not globally for the entire application. And surely adding the piece of code like the one below to every controller makes it not well suited for a general exception handling mechanism (Figure 16).



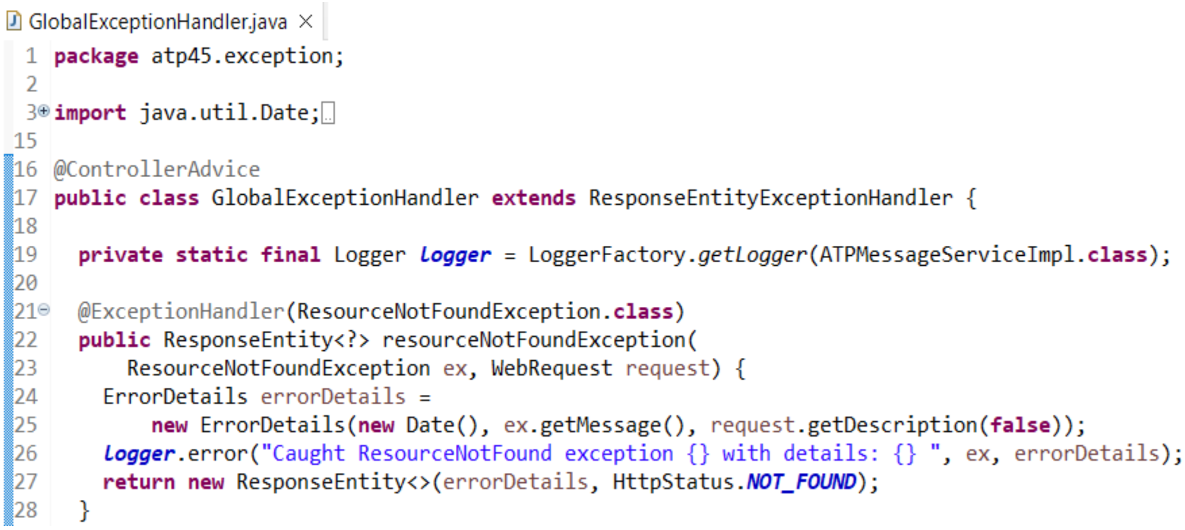
```
public class FooController{

    //...
    @ExceptionHandler({ CustomException1.class, CustomException2.class })
    public void handleException() {
        //
    }
}
```

Figure 16. Exception handler

Regarding the second method, HandlerExceptionResolver, it sets the status code of the Response properly and can handle exceptions across all application, but the limitation is that it doesn't set anything to the body of the Response.

So, the optimum solution is the @ControllerAdvice annotation, which allows to consolidate multiple scattered @ExceptionHandlers into a single, global error handling component.

The mechanism of this option is very simple, but flexible. It gives full control over the body of the response and the status code. It provides mapping of several exceptions to the same method, to be handled together (Figure 17).



```
package atp45.exception;

import java.util.Date;

@ControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

  private static final Logger logger = LoggerFactory.getLogger(ATPMessageServiceImpl.class);

  @ExceptionHandler(ResourceNotFoundException.class)
  public ResponseEntity<?> resourceNotFoundException(
      ResourceNotFoundException ex, WebRequest request) {
    ErrorDetails errorDetails =
        new ErrorDetails(new Date(), ex.getMessage(), request.getDescription(false));
    logger.error("Caught ResourceNotFound exception {} with details: {} ", ex, errorDetails);
    return new ResponseEntity<>(errorDetails, HttpStatus.NOT_FOUND);
  }
```

Figure 17. Global exception handler class

In the picture above there is a part of Global exception handler class, which describes standard methods for handling different types of exceptions. It provides uniform method for information representation and response to all errors occurring in the application.

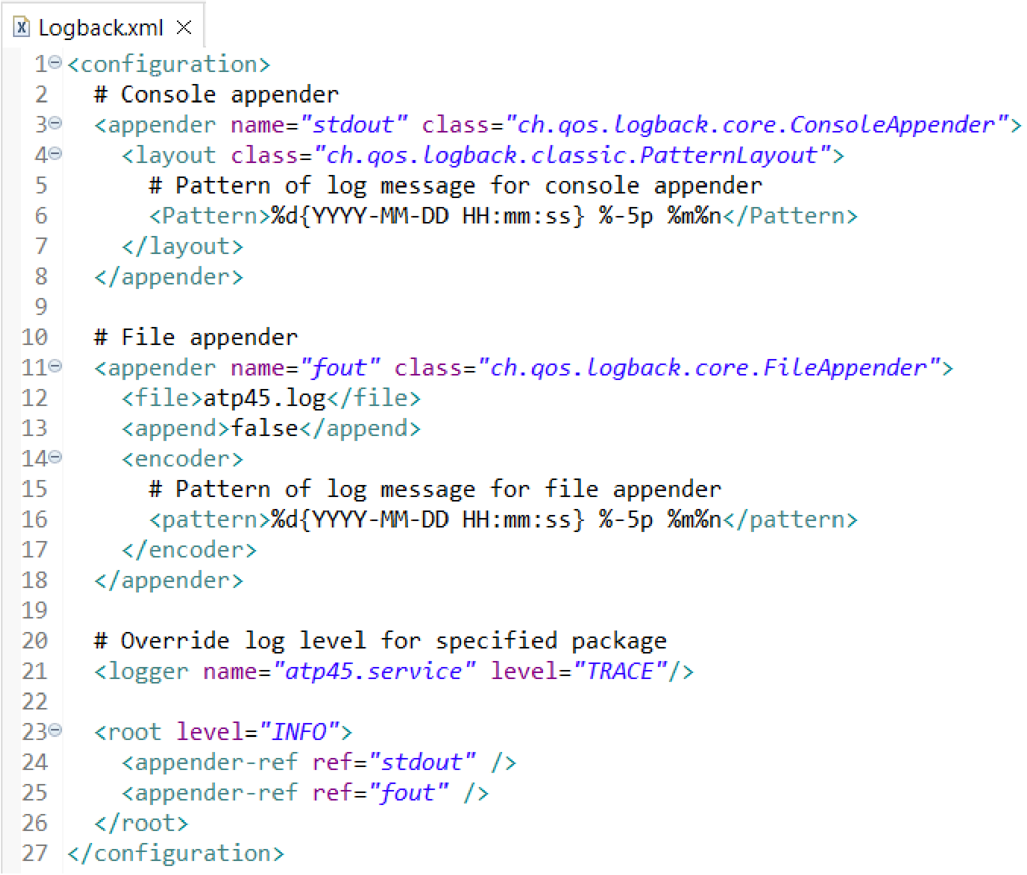
5.3 Logging

Logging is a powerful aid for understanding and debugging program's run-time behavior. Logs also capture and persist the important data and make it available for analysis at any point in time.

While there is always an option for a default Java.Util.Logging present in any Java application, there are other logging frameworks. And to discuss other options, it is important to define some logging-related terms first.

There are three logging components: loggers, appenders, and layouts that need to be configured. Loggers prepare log statements to be written to console or log file.Log levels indicate the type of event that the system logs. You can assign the following levels to Loggers: DEBUG, INFO, WARN, ERROR, and FATAL. Appenders specify the output definitions for loggers. For example, a rolling appender writes log statements to a specified file. Once a size limit for the file is reached, that file is renamed, and a new file is created. Appenders can exist for consoles or files. Console appender writes log statements to the application server console. Rolling appender writes log statements to the specified file. Once the file size limit is reached (5 MB by default), the current file is renamed, and a new file is created. A layout specifies the output format of the log statement, e.g., timestamp and text format, and is always associated with an appender.
In ATP45 Logback logger is used. Configurations for console and file logs are made in Logback.xml file (Figure 18).

In the file below, there are configurations for both console output and for a special log file, which means that log records will be done while the program is running in the console. And a separate file with event records will be kept separately.

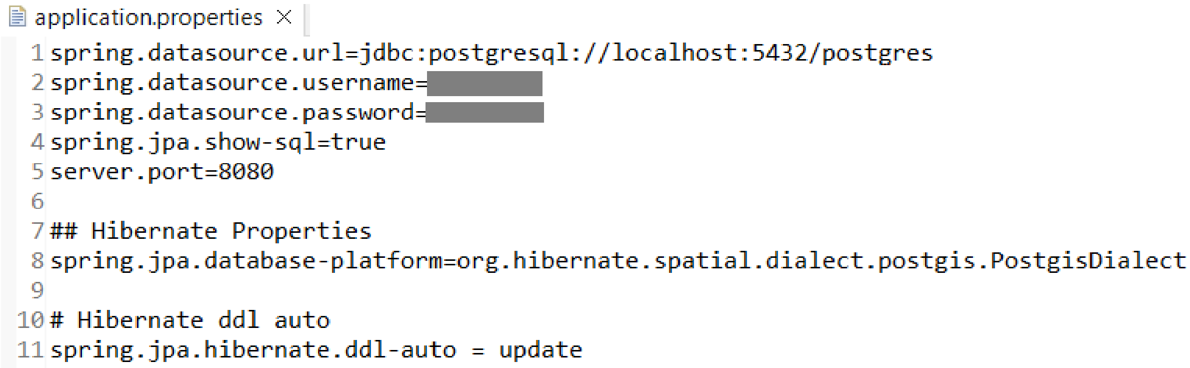


```
<configuration>
  # Console appender
  <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
    <layout class="ch.qos.logback.classic.PatternLayout">
      # Pattern of log message for console appender
      <Pattern>%d{YYYY-MM-DD HH:mm:ss} %-5p %m%n</Pattern>
    </layout>
  </appender>

  # File appender
  <appender name="fout" class="ch.qos.logback.core.FileAppender">
    <file>atp45.log</file>
    <append>false</append>
    <encoder>
      # Pattern of log message for file appender
      <pattern>%d{YYYY-MM-DD HH:mm:ss} %-5p %m%n</pattern>
    </encoder>
  </appender>

  # Override log level for specified package
  <logger name="atp45.service" level="TRACE"/>

  <root level="INFO">
    <appender-ref ref="stdout" />
    <appender-ref ref="fout" />
  </root>
</configuration>
```

Figure 18. Logback configuration file

5.4 PostgreSQL connection

The configuration of database connection is rather straightforward. It takes only a few lines (Figure 19):

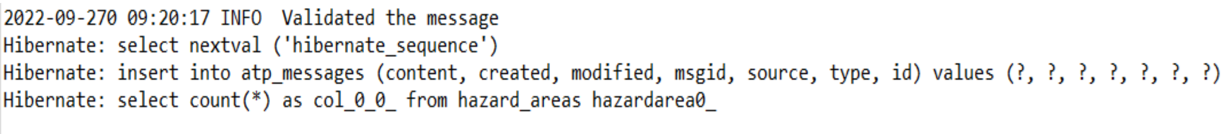


```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=[redacted]
spring.datasource.password=[redacted]
spring.jpa.show-sql=true
server.port=8080

## Hibernate Properties
spring.jpa.database-platform=org.hibernate.spatial.dialect.postgis.PostgisDialect

# Hibernate ddl auto
spring.jpa.hibernate.ddl-auto = update
```

Figure 19. PostgreSQL connection configuration file

Line one determines the path to the database, and initially the application runs in the same machine as PostgreSQL, so the path is to localhost via default database port 5432, which then connects to database called ‘postgres’.

In lines two and three the username and password to this database are stated. Line four allows SQL queries to be recorded in the standard output of the development environment or terminal (Figure 20).



```
2022-09-270 09:20:17 INFO  Validated the message
Hibernate: select nextval ('hibernate_sequence')
Hibernate: insert into atp_messages (content, created, modified, msgid, source, type, id) values (?, ?, ?, ?, ?, ?, ?)
Hibernate: select count(*) as col_0_0_ from hazard_areas hazardarea0_
```

Figure 20. PostgreSQL queries in Eclipse IDE

In line five port on which the application is running on the machine is exposed to the connection to the database.

Configuration in line eight is a bit more complex, here a dialect for the database is defined. Dialect in Hibernate is a class that bridges java language data types and database datatypes. Dialect allows Hibernate to be optimized for generation of SQL queries for a particular relational database type. Hibernate generates queries for the specific database based on the Dialect class. In this case PostGIS dialect is indicated. It adds support for the Postgis spatial types, functions and operators. It is particularly important because ATP45 software needs to keep records of spatial data – locations of incidents and coordinates to calculate and draw on the map expected hazard areas.

Finally, Hibernate property defined in line eleven, ‘ddl-auto’ defines the manner in which tables are created in the database. Here it is set to ‘update’ which is a convenient option in development stage as, in this case, new columns and constraints can be added upon re-run of the application but columns or constraints that existed previously are not removed, even if they are no longer part of the object model from a prior run. It may be a bit problematic as constraints that may exist from previous executions are no longer necessary or interfere with the new code. In production stage, on the other hand, it's often highly recommended you use ‘none’ or not to specify this property.

5.5 Dockerization: ATP45 and Postgis containers, compose file

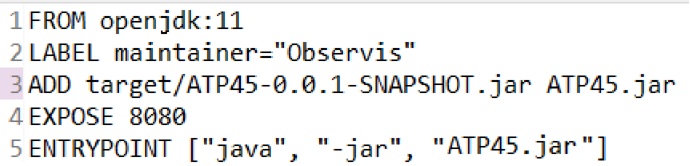
This section discusses the process of ATP45 application containerization. As there are two containers for the ATP45 application to run – the ATP45 itself and a container that runs PostgreSQL database with PostGIS geographical extension – there are several steps to configure smooth work of combined containers. The steps are depicted in detail in the next part.

5.5.1 ATP45 Docker container

After all the changes were applied to convert the original application to a Spring Boot one, it was containerized using Docker for better further transportability.

Dockerization of ATP45 itself is a simple process. Given that Docker is installed and configured on the machine running the application, there are only few small steps. Firstly, the application is run as ‘Maven install’ to create a .jar file

Next, it is necessary to add a Dockerfile into the structure of the application in the development environment and specify some docker build things (Figure 21):



```
FROM openjdk:11
LABEL maintainer="Observis"
ADD target/ATP45-0.0.1-SNAPSHOT.jar ATP45.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "ATP45.jar"]
```

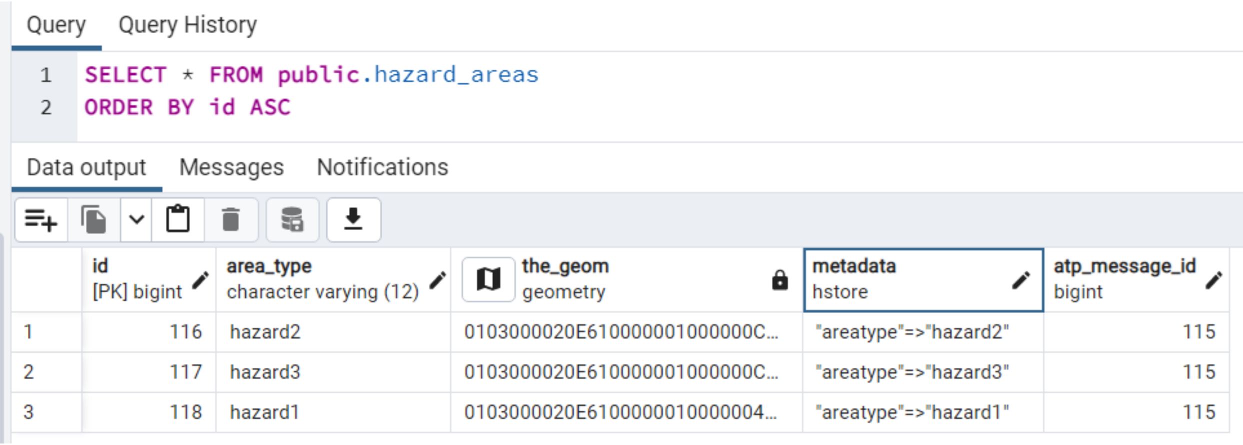
Figure 21. Docker configuration file

The final step in creating an application docker container is to use the command, which creates a container with the name ‘ATP45’: ‘*docker build -t ATP45:latest .’*

5.5.2 Postgis container, compose file

The ATP45 is not the only component for the system to work. This application, running in a container, also requires a database to run in a container. Fortunately, There already exists and official Docker image for a PostgreSQL database, but as already noted before, one of the important features of ATP45 is storage of geographic coordinates of incident locations and hazard areas. Thus, a plain PostgreSQL Docker image does not suffice. However there already is a PostgreSQL image with PostGIS support - postgis/postgis image, which can be found in official Docker Hub container image library. The postgis/postgis image provides tags for running Postgres with PostGIS extensions installed. PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL.

For example, with this extension, it is possible to use some special queries. Here is the initial data in hazard_areas table that correspond to one ATP45 message(Figure 22).

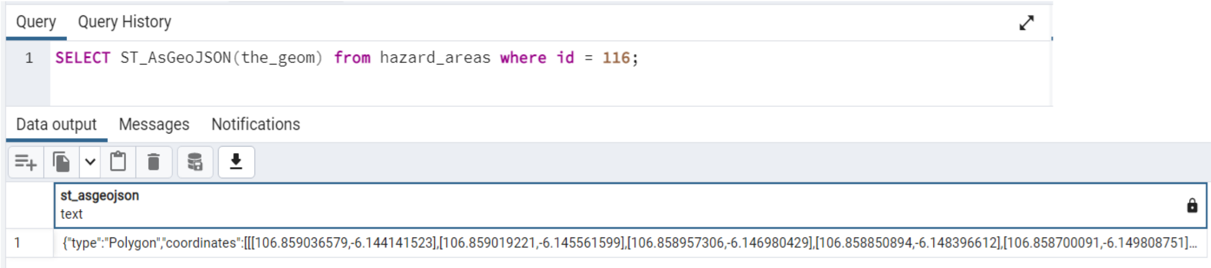


```sql
SELECT * FROM public.hazard_areas
ORDER BY id ASC
```

	id [PK] bigint	area_type character varying (12)	the_geom geometry	metadata hstore	atp_message_id bigint
1	116	hazard2	0103000020E610000001000000C...	"areatype"=>"hazard2"	115
2	117	hazard3	0103000020E610000001000000C...	"areatype"=>"hazard3"	115
3	118	hazard1	0103000020E6100000010000004...	"areatype"=>"hazard1"	115

Figure 22. Geometry data in PostgreSQL

Having PostGIS extension, it is possible to extract coordinates in a JSON format and use them to draw a hazard area (Figure 23 and 24).



```sql
SELECT ST_AsGeoJSON(the_geom) from hazard_areas where id = 116;
```

```
   st_asgeojson
   text
1  {"type":"Polygon","coordinates":[[[106.859036579,-6.144141523],[106.859019221,-6.145561599],[106.858957306,-6.146980429],[106.858850894,-6.148396612],[106.858700091,-6.149808751]...
```

Figure 23. GeoJson query in PostgreSQL

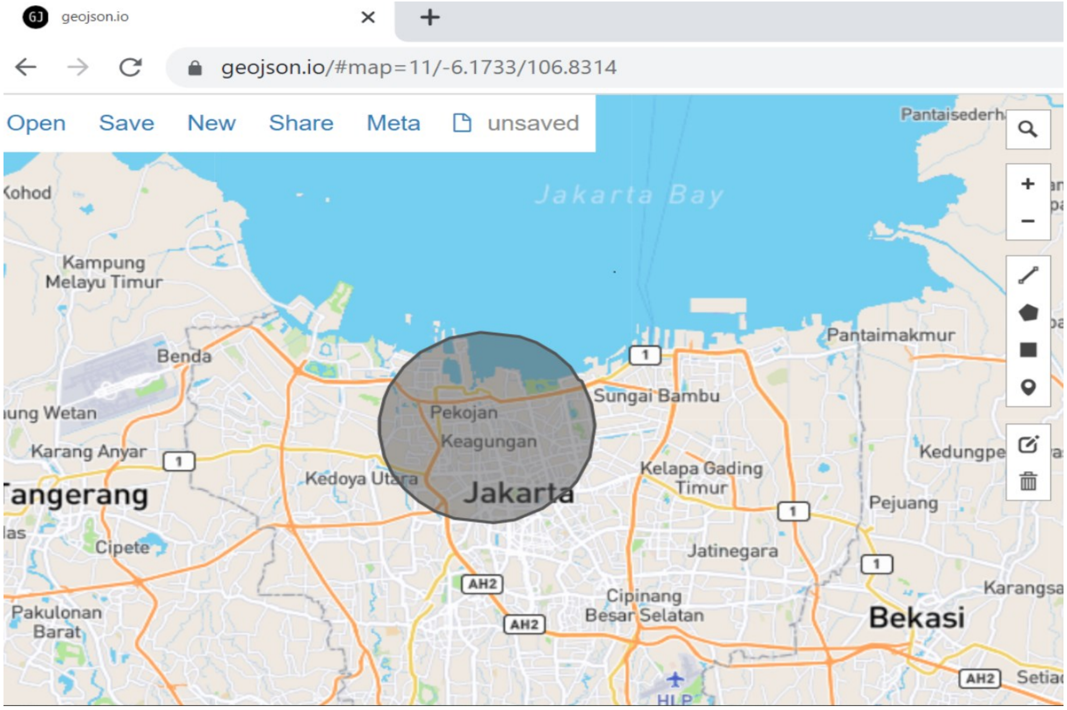
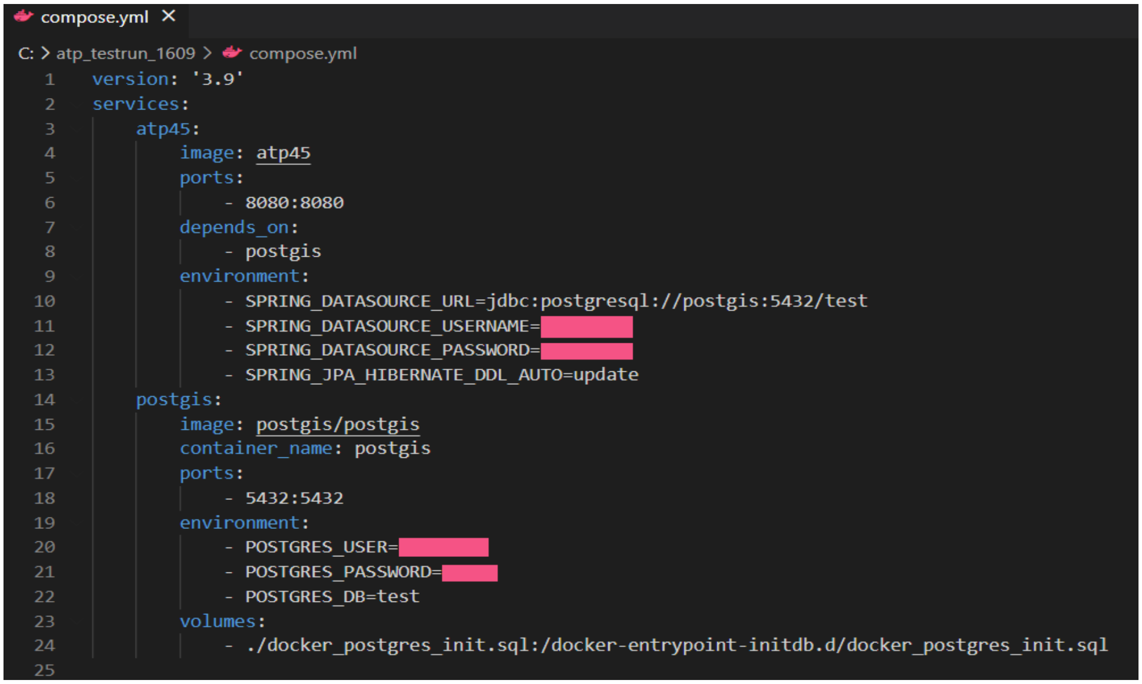


Figure 24. Coordinates from geojson

However, to run two containers as a single system, it is necessary to form a set of instructions. It is done with the use of Docker compose file (Figure 25).



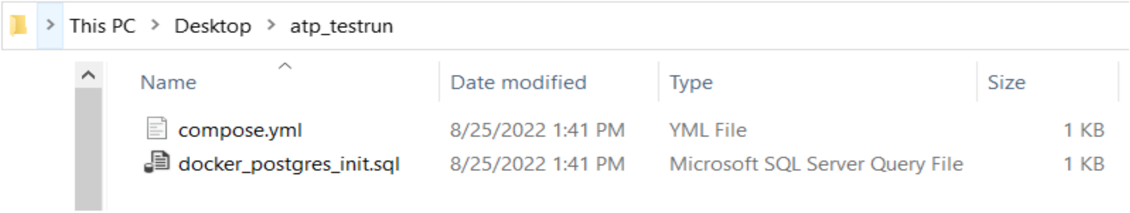
```yaml
1  version: '3.9'
2  services:
3      atp45:
4          image: atp45
5          ports:
6              - 8080:8080
7          depends_on:
8              - postgis
9          environment:
10             - SPRING_DATASOURCE_URL=jdbc:postgresql://postgis:5432/test
11             - SPRING_DATASOURCE_USERNAME=[redacted]
12             - SPRING_DATASOURCE_PASSWORD=[redacted]
13             - SPRING_JPA_HIBERNATE_DDL_AUTO=update
14     postgis:
15         image: postgis/postgis
16         container_name: postgis
17         ports:
18             - 5432:5432
19         environment:
20             - POSTGRES_USER=[redacted]
21             - POSTGRES_PASSWORD=[redacted]
22             - POSTGRES_DB=test
23         volumes:
24             - ./docker_postgres_init.sql:/docker-entrypoint-initdb.d/docker_postgres_init.sql
25
```

Figure 25. Docker compose file

In lines three and fourteen, two containers, here called ‘services’ are specified - ‘atp45’ and ‘postgis’. The exact names of the containers can be checked either in Docker Desktop app or using a command ‘docker ls’ in terminal.

In lines six and eighteen there are ports on which the app and the database run. There is also a dependency, in line eight, of atp45 container on postgis container. This means that the main application cannot run without a connection to a database.

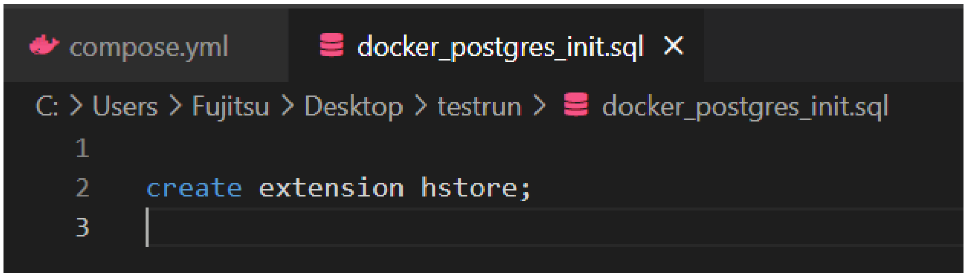
It is also interesting to note line twenty-four in PostGIS image configuration. This line indicates that there is a script that should run upon creation of PostGIS container.



Name	Date modified	Type	Size
compose.yml	8/25/2022 1:41 PM	YML File	1 KB
docker_postgres_init.sql	8/25/2022 1:41 PM	Microsoft SQL Server Query File	1 KB

Figure 26. Folder containing files required for docker compose

Both compose file and initialization script should be placed in the same folder somewhere outside the application (Figure 26). Upon run of docker compose file, docker_postgres_init.sql script is run. It keeps an instruction to run SQL query upon creation (Figure 27).

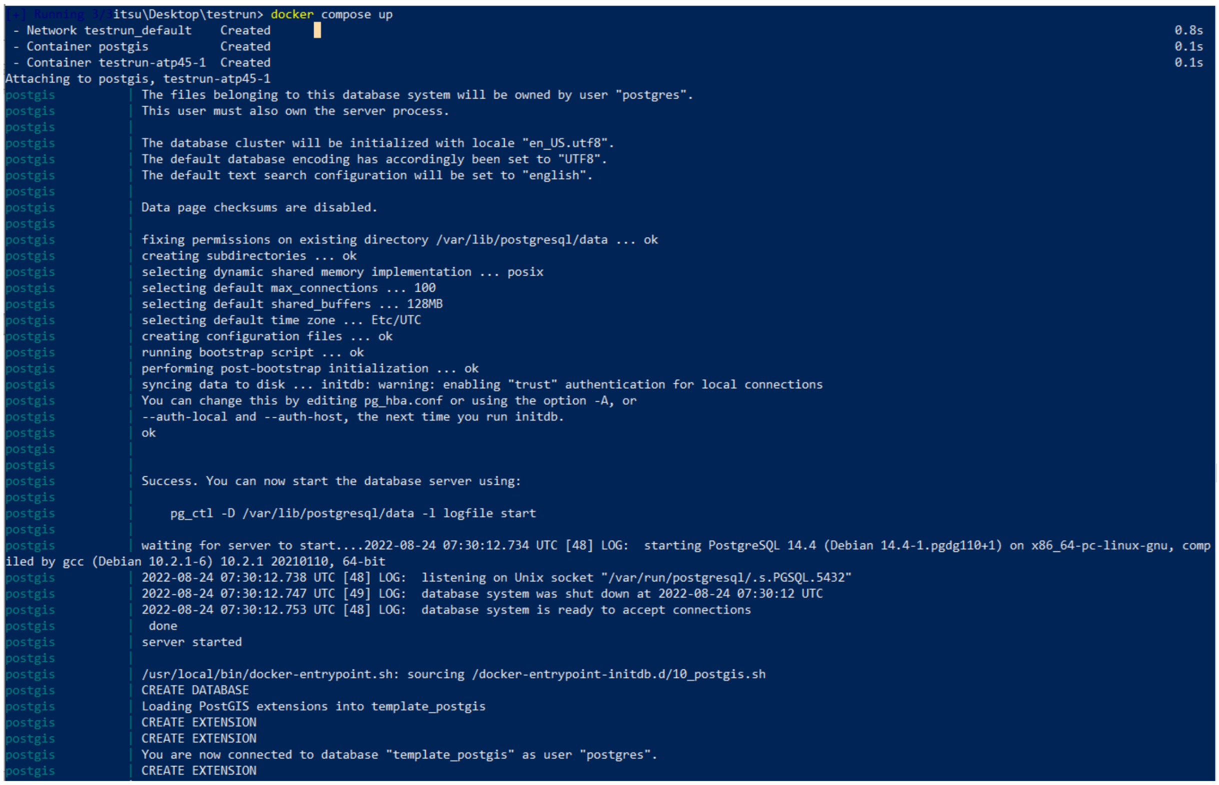

compose.yml
docker_postgres_init.sql ×
C: > Users > Fujitsu > Desktop > testrun > docker_postgres_init.sql

```sql

create extension hstore;

```

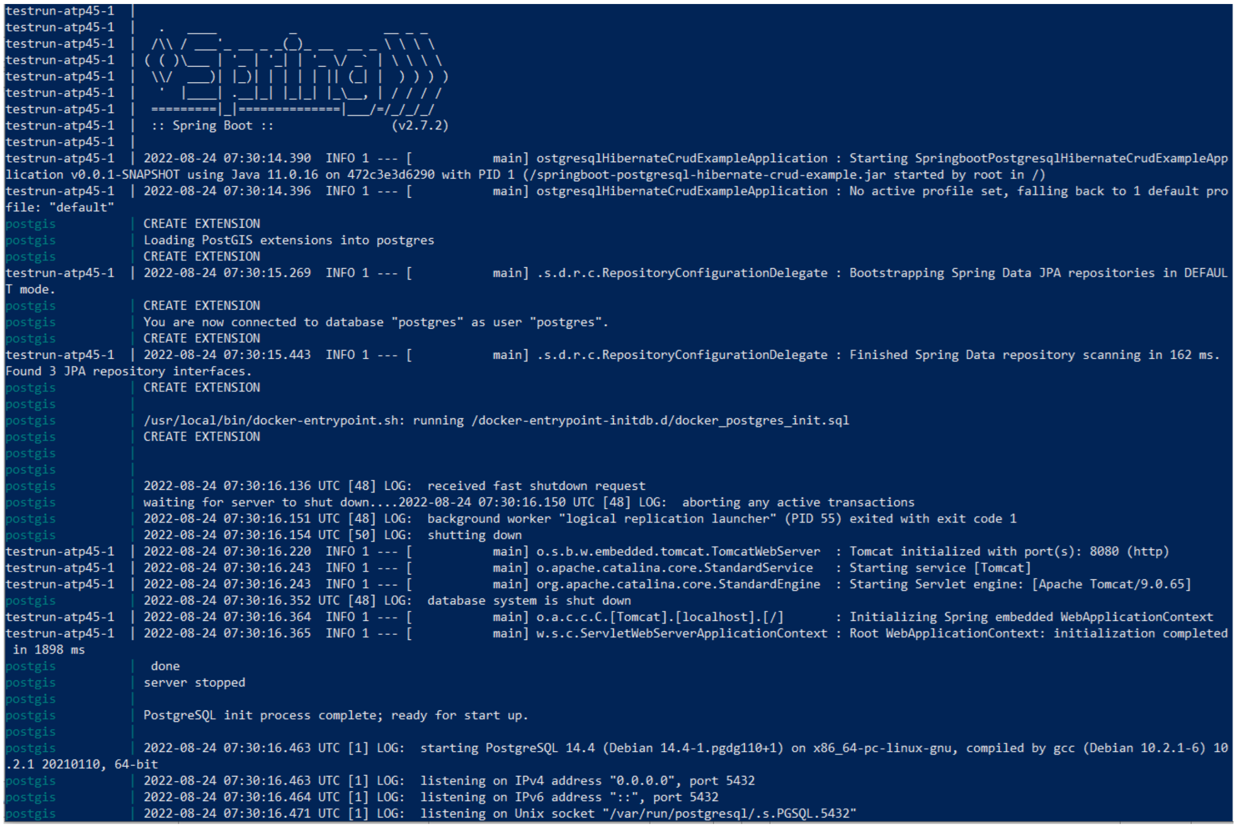
Figure 27. Configuration file for Postgis container

This module implements the hstore data type for storing sets of key/value pairs within a single PostgreSQL value.

Finally, running a ‘docker compose up’ commands brings the set of instruction from Docker compose file and initialization script into action (Figure 28).



```
[+] Running 3/3itsu\Desktop\testrun> docker compose up
 - Network testrun_default    Created                                                                    0.8s
 - Container postgis          Created                                                                    0.1s
 - Container testrun-atp45-1  Created                                                                    0.1s
Attaching to postgis, testrun-atp45-1
postgis          | The files belonging to this database system will be owned by user "postgres".
postgis          | This user must also own the server process.
postgis          |
postgis          | The database cluster will be initialized with locale "en_US.utf8".
postgis          | The default database encoding has accordingly been set to "UTF8".
postgis          | The default text search configuration will be set to "english".
postgis          |
postgis          | Data page checksums are disabled.
postgis          |
postgis          | fixing permissions on existing directory /var/lib/postgresql/data ... ok
postgis          | creating subdirectories ... ok
postgis          | selecting dynamic shared memory implementation ... posix
postgis          | selecting default max_connections ... 100
postgis          | selecting default shared_buffers ... 128MB
postgis          | selecting default time zone ... Etc/UTC
postgis          | creating configuration files ... ok
postgis          | running bootstrap script ... ok
postgis          | performing post-bootstrap initialization ... ok
postgis          | syncing data to disk ... initdb: warning: enabling "trust" authentication for local connections
postgis          | You can change this by editing pg_hba.conf or using the option -A, or
postgis          | --auth-local and --auth-host, the next time you run initdb.
postgis          | ok
postgis          |
postgis          |
postgis          | Success. You can now start the database server using:
postgis          |
postgis          |     pg_ctl -D /var/lib/postgresql/data -l logfile start
postgis          |
postgis          | waiting for server to start....2022-08-24 07:30:12.734 UTC [48] LOG:  starting PostgreSQL 14.4 (Debian 14.4-1.pgdg110+1) on x86_64-pc-linux-gnu, comp
iled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
postgis          | 2022-08-24 07:30:12.738 UTC [48] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
postgis          | 2022-08-24 07:30:12.747 UTC [49] LOG:  database system was shut down at 2022-08-24 07:30:12 UTC
postgis          | 2022-08-24 07:30:12.753 UTC [48] LOG:  database system is ready to accept connections
postgis          |  done
postgis          | server started
postgis          |
postgis          | /usr/local/bin/docker-entrypoint.sh: sourcing /docker-entrypoint-initdb.d/10_postgis.sh
postgis          | CREATE DATABASE
postgis          | Loading PostGIS extensions into template_postgis
postgis          | CREATE EXTENSION
postgis          | CREATE EXTENSION
postgis          | You are now connected to database "template_postgis" as user "postgres".
postgis          | CREATE EXTENSION
```



```
testrun-atp45-1  |
testrun-atp45-1  |   .   ____          _            __ _ _
testrun-atp45-1  |  /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
testrun-atp45-1  | ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
testrun-atp45-1  |  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
testrun-atp45-1  |   '  |____| .__|_| |_|_| |_\__, | / / / /
testrun-atp45-1  |  =========|_|==============|___/=/_/_/_/
testrun-atp45-1  |  :: Spring Boot ::                (v2.7.2)
testrun-atp45-1  |
testrun-atp45-1  | 2022-08-24 07:30:14.390  INFO 1 --- [           main] ostgresqlHibernateCrudExampleApplication : Starting SpringbootPostgresqlHibernateCrudExampleApp
lication v0.0.1-SNAPSHOT using Java 11.0.16 on 472c3e3d6290 with PID 1 (/springboot-postgresql-hibernate-crud-example.jar started by root in /)
testrun-atp45-1  | 2022-08-24 07:30:14.396  INFO 1 --- [           main] ostgresqlHibernateCrudExampleApplication : No active profile set, falling back to 1 default pro
file: "default"
postgis          | CREATE EXTENSION
postgis          | Loading PostGIS extensions into postgres
postgis          | CREATE EXTENSION
testrun-atp45-1  | 2022-08-24 07:30:15.269  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAUL
T mode.
postgis          | CREATE EXTENSION
postgis          | You are now connected to database "postgres" as user "postgres".
postgis          | CREATE EXTENSION
testrun-atp45-1  | 2022-08-24 07:30:15.443  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 162 ms.
Found 3 JPA repository interfaces.
postgis          | CREATE EXTENSION
postgis          |
postgis          | /usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.d/docker_postgres_init.sql
postgis          | CREATE EXTENSION
postgis          |
postgis          |
postgis          | 2022-08-24 07:30:16.136 UTC [48] LOG:  received fast shutdown request
postgis          | waiting for server to shut down....2022-08-24 07:30:16.150 UTC [48] LOG:  aborting any active transactions
postgis          | 2022-08-24 07:30:16.151 UTC [48] LOG:  background worker "logical replication launcher" (PID 55) exited with exit code 1
postgis          | 2022-08-24 07:30:16.154 UTC [50] LOG:  shutting down
testrun-atp45-1  | 2022-08-24 07:30:16.220  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
testrun-atp45-1  | 2022-08-24 07:30:16.243  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
testrun-atp45-1  | 2022-08-24 07:30:16.243  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
postgis          | 2022-08-24 07:30:16.352 UTC [48] LOG:  database system is shut down
testrun-atp45-1  | 2022-08-24 07:30:16.364  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
testrun-atp45-1  | 2022-08-24 07:30:16.365  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
 in 1898 ms
postgis          |  done
postgis          | server stopped
postgis          |
postgis          | PostgreSQL init process complete; ready for start up.
postgis          |
postgis          | 2022-08-24 07:30:16.463 UTC [1] LOG:  starting PostgreSQL 14.4 (Debian 14.4-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10
.2.1 20210110, 64-bit
postgis          | 2022-08-24 07:30:16.463 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
postgis          | 2022-08-24 07:30:16.464 UTC [1] LOG:  listening on IPv6 address "::", port 5432
postgis          | 2022-08-24 07:30:16.471 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
```

Figure 28. Starting containers with ‘docker compose up’ command

5.6 CBRN 4 feature

As noted before, in Chapter 2, CBRN4 is a type of message which combines and reports data from detection devices as opposed to CBRN1, CBRN2 or CBRN 3 messages – which report data collected by human observers who notice an incident, report their observations and add location data, CBRN3 messages also summarize data about expected hazard areas.

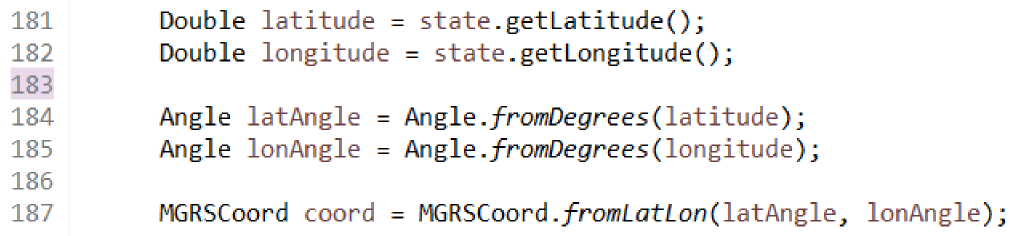
Data flow for a CBRN4 message is more complex. There is usually a collection of devices - detectors of poisonous gases, radiation detectors, equipment for biological analysis, which are assembled either in a stationary base or in a vehicle. Data from the devices is sent by MQTT network protocol over Bluetooth to be collected in a centralized location. This data needs to be fetched by ATP45 application, processed and stored at its own database.

Regarding the code, CBRN 4 message feature has a standard structure, which includes:

- controller level that routes requests to the correct endpoint,

- service layer that provides the logic and validity checks for fetching and combining the data,
- persistence layer (data representation as an entity and storage at the repository)

However, there are a few interesting features in the implementation of the logic for CBRN4. For example, initially location data is stored in form of latitude and longitude in decimal degrees, while for the purpose of ATP45, the location must be expressed in a special coordinate system – Military Grid Reference System (MGRS). Conversions from latitude and longitude are performed using NASA World Wind package (Figure 29).



```
        Double latitude = state.getLatitude();
        Double longitude = state.getLongitude();

        Angle latAngle = Angle.fromDegrees(latitude);
        Angle lonAngle = Angle.fromDegrees(longitude);

        MGRSCoord coord = MGRSCoord.fromLatLon(latAngle, lonAngle);
```

Figure 29. Coordinate conversion feature

6 CONCLUSION AND DISCUSSION

This thesis work deals with the topic implementation of NATO ATP45 standard in software. Such kind of work requires, first, a great deal of investigation of the ATP45 specifications to complete the tasks accurately. Secondly, it requires good understanding of modern architecture of REST API Spring Boot applications and their connections to relational databases. The work itself also included several steps, some of which were not related to each other, but all served to improve the functionality of the broad ATP45 system application.

The major strength of this thesis work is that it allowed me to use a real-world case and to co-operate with employees of Observis Oy. Thus, I acquired not only computer programming skills and enhance my approach towards solving coding tasks, but also took part in the day-to-day life of a software development company developing real-world working skills.

This is also worth mentioning that introduction of Spring Boot brought the existing in the company ATP45 software on a new level of simplicity and applicability as a REST API microservice. Also, a feature of CBRN4 message reporting was added to increase the value of the application for future users.

However, there is still room for improvement of this piece of software – external weather reporting data can be considered when forming CBRN messages and reports. There is also possibility of further development of user interface interactions with the ATP45 system.

7 REFERENCES

Bos, S. 2020. Java Basics: What is Spring Boot? Web page. August 5, 2020. Available at https://www.jrebel.com/blog/what-is-spring-boot [Accessed 12 July 2022].

Council Directive 98/83/EC 3 November 1998. The quality of water intended for human consumption. WWW document. Available at: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31998L0083&from=EN [Accessed 5 February 2017].

EnterpriseDB. 2020. Generating and Managing PostgreSQL Database Migrations (Upgrades) with Spring Boot JPA. Web page. July 20, 2020. Available at: https://www.enterprisedb.com/blog/generating-and-managing-postgresql-database-migrationsupgrades-spring-boot-jpa [Accessed 13 September 2022].

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Dissertation. 2000.

Fishman, R. 2005. The rise and fall of suburbia. Ebook. Chester: Castle Press. Available at: http://libweb.anglia.ac.uk [Accessed 12 May 2010].

Hall, R.S., Pauls, K., Mcculloch, S., Savage, D. OSGi in Action: Creating Modular Applications in Java, Manning Publications (2010).

Hasselbach, S. 2020. Migrating OSGi Plugins to standalone applications. Web page. January 13, 2020. Available at http://hasselba.ch/blog/?p=2865 [Accessed 1 June 2022].

Haroon, M. 2020. RESTful CRUD API using PostgreSQL and Spring Boot – Part 2. Web page. June 16, 2020. Available at https://www.2ndquadrant.com/en/blog/restful-crud-api-using-postgresql-and-spring-boot-part-2/ [Accessed 09 August 2022].

IBM. 2020. Java Spring Boot Learn how Java Spring Boot simplifies development of web applications and microservices with Java Spring Framework. Web page. March 25, 2020. Available at https://www.ibm.com/cloud/learn/java-spring-boot [Accessed 1 June 2022].

Jensen, K. L., Toftum, J. & Friis-Hansen, P. 2009. A Bayesian network approach to the evaluation of building design and its consequences for employee performance and operational costs. Building and Environment 44, 456–462.

Kappagantula, S. 2021. Docker Explained – An Introductory Guide To Docker. Web page. December 15, 2021. Available at: https://www.edureka.co/blog/docker-explained/ [Accessed 19 August 2022].

Naeem, T. 2020. REST API Definition: Understanding the Basics of REST APIs. Web page. January 28, 2020. Available at https://www.astera.com/type/blog/rest-api-definition/ [Accessed 12 July 2022].

NATO publications. 2014. Project on Minimum Standards and Non-Binding Guidelines for First Responders Regarding Planning, Training, Procedure and Equipment for Chemical, Biological, Radiological and Nuclear (CBRN) Incidents GUIDELINES FOR FIRST RESPONDERS TO A CBRN INCIDENT. Online document. August 1, 2014. Available at https://www.nato.int/nato_static_fl2014/assets/pdf/pdf_2016_08/20160802_140801-cep-first-responders-CBR.pdf [Accessed 23 July 2022].

NATO standardization agency (NSA). 2014. ATP-45 warning and reporting and hazard prediction of chemical, biological, radiological and nuclear incidents (operators manual) (edition E version 1). Document. January 2014

NATO standardization agency (NSA). 2018. AEP-45 warning and reporting and hazard prediction of chemical, biological, radiological and nuclear incidents (reference manual). Document. August 2018.

Nguyen, T. 2021 Spring Boot, Hibernate, MySQL example: CRUD app. Web page. May 13, 2021. Available at: https://dev.to/tienbku/spring-boot-hibernate-mysql-example-crud-app-4mhb [Accessed 13 September 2022].

Oracle. n.d. What is a Relational Database (RDBMS)? Web page. Available at: https://www.oracle.com/database/what-is-a-relational-database/ [Accessed 09 August 2022].

Pandit, S. 2021. What is Docker Container? – Containerize Your Application Using Docker. Web page. December 15,2021. Available at: https://www.edureka.co/blog/docker-container/ [Accessed 19 August 2022].

Peräjärvi, K. 2022. Finnish nuclear security detection architecture for nuclear and other radioactive material out of regulatory control. Online document. May, 2022. Available at https://www.julkari.fi/bitstream/handle/10024/144407/STUK-A_267_Finnish_nuclear_security_detection_architecture_for_nuclear_and_other_radioactive_material_out_of_regulatory_control.pdf?sequence=2&isAllowed=y [Accessed 09 August 2022].

Redhat. 2020. What is a REST API? Web page. May 8, 2020. Available at https://www.redhat.com/en/topics/api/what-is-a-rest-api [Accessed 23 July 2022].
Angel, S. 2021. REST API Best Practices and Standards in 2022. Web page. November 1, 2021. Available at https://hevodata.com/learn/rest-api-best-practices/ [Accessed 23 July 2022].

Sirgur, B. 2021. Spring Boot, Hibernate, JPA and H2 Database CRUD REST API Example. Web page. February 05, 2021. Available at https://javatechonline.com/spring-boot-bean-annotations-with-examples/ [Accessed 1 June 2022].

8 ABBREVIATIONS

API – Application Programming Interface

CBRN – Chemical, Biological, Radiological and Nuclear

CIS – Command Information Systems

JPA – Java Persistence API

JSON – JavaScript Object Notation

MGRS – Military Grid Reference System

MQTT – MQ Telemetry Transport (protocol)

MVC – Model View Controller

NATO – North Atlantic Treaty Organization

ObSAS – Observis situational awareness system

ORM – Object-Relation Mapping

RDBMSs – Relational Database Management Systems

REST – Representational State Transfer

SAS – Situational Awareness System

SOAP – Simple object access protocol

XML – Extensible Markup Language