

CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF CIVIL ENGINEERING

DIPLOMA THESIS

PRAHA 2019

Bc. Lubomír BUCEK

CZECH TECHNICAL UNIVERSITY IN PRAGUE

FAKULTY OF CIVIL ENGINEERING

STUDY PROGRAMME GEODESY AND CARTOGRAPHY

SPECIALIZATION OF STUDY GEOMATICS



DIPLOMA THESIS

HISTORY AND FORECAST WEATHER AND WAVE DATA
PROCESSING AND VISUALIZATION FRAMEWORK

Supervisor: Doc. Ing. Jiří CAJTHAML, Ph.D.

Department of Geomatics

January 2019

Bc. Lubomír BUCEK



ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: <u>Bucek</u>	Jméno: <u>Lubomír</u>	Osobní číslo: <u>432992</u>
Zadávající katedra: <u>K155 - katedra geomatiky</u>		
Studijní program: <u>Geodézie a kartografie</u>		
Studijní obor: <u>Geomatika</u>		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Rámec pro zpracování a vizualizaci předpovědi a historických dat o počasí a vlnách


Název diplomové práce anglicky: Weather forecast, historic weather and wave data processing and visualization framework

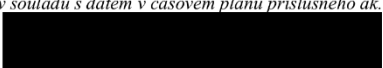
Pokyny pro vypracování:
Cílem diplomové práce je návrh automatizovaného softwarového rámce pro stahování, zpracování a publikování veřejně dostupných celosvětových meteorologických dat o počasí a vlnách pomocí open source technologií. Rámec bude tvořit součást doplňkových datových a vizualizačních služeb webové mapové aplikace pro sledování pohybu lodí Cockpit společnosti Vesseltracker.com.

Seznam doporučené literatury:
GeoServer Cookbook - Stefano Iacovella, Packt Publishing, 2014
Spatial Interpolation for Climate Data - The Use of GIS in Climatology and Meteorology, Hartwig Dobesch, Pierre Dumolard, Izabela Dyras, Wiley.com, 2013
ElasticSearch - An advanced and quick search technique to handle voluminous data, Manda Sai Divya, Shiv Kumar Goyal. COMPUSOFT, 2013, An international journal of advanced computer technology, (171-175)

Jméno vedoucího diplomové práce: doc. Ing. Jiří Cajthaml, Ph.D.

Datum zadání diplomové práce: 1.10.2018 Termín odevzdání diplomové práce: 7.1.2019
Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku


 Podpis vedoucího práce

 Podpis vedoucího katedry

III. PŘEVZETÍ ZADÁNÍ

Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce“ a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.

1.10.2018 Datum převzetí zadání

 Podpis studenta(ky)

Declaration of authorship

I declare that the diploma thesis presented here is, to the best of my knowledge original and the result of my own investigation, except as acknowledged. Formulations, ideas and approaches taken from other sources are cited appropriately

In Prague

.....

Lubomír Bucek

Acknowledgement

I would like to thank to the thesis supervisor Doc. Ing. Jiří Cajthaml, Ph.D. for his guidance and insight. I would like to thank Vesseltracker.com for directing me to a complex, yet engaging topic. I would like to express my gratitude to my family for their lasting support. Last but not the least, I would like to thank deeply to my partner for her aid, patience and insight.

Abstract

Diploma thesis deals with the design and implementation of a framework for downloading, processing and publishing of weather and waves data using open-source technologies. Numerical weather forecasts basics and history are outlined in the text of the thesis together with the introduction of dominant global models. It describes meteorological data formats, used technologies and framework implementation. The practical part of the thesis consists of Python scripts for downloading, processing and storing historical weather and waves data into Elasticsearch, access API in Node.js, Python scripts for downloading, processing and creating appropriate GeoTIFF rasters and complete configuration of GeoServer required to display them. Outputs are present in the E-attachments and will be used as add-ons to the vesseltracker.com GmbH Cockpit web map application.

Keywords

Meteorology data, GRIB, weather, GeoServer, Elasticsearch, Python

Abstrakt

Diplomová práce se zabývá návrhem a implementací rámce pro stahování, zpracování a publikování meteorologických dat o počasí a vlnách pomocí open-source technologií. V textu práce jsou nastíněny souhrnné informace a historie o numerických předpovědích počasí a vln společně s uvedením dominantních světových modelů. Dále jsou popsány meteorologické datové formáty, použité technologie a implementace. Praktická část práce se skládá z Python skriptů pro stahování, zpracování a uložení historických dat o počasí a vlnách do databáze Elasticsearch, přístupové API v Node.js, Python skriptů pro stahování, zpracování a tvorbu příslušných rastrů ve formátu GeoTIFF a kompletní konfiguraci GeoServeru nutnou pro jejich zobrazení. Výstupy jsou uvedeny v přílohách a budou využity jako doplňky webové mapové aplikace Cockpit společnosti vesseltracker.com GmbH.

Klíčová slova

Meteorologická data, GRIB, počasí, GeoServer, Elasticsearch, Python

Contents

Introduction	9
1 Meteorology and data	11
1.1 History of meteorology	11
1.2 Models	14
1.2.1 Global Forecast System	15
1.2.2 ECMWF Integrated Forecasting System	16
1.2.3 Other global models	16
1.2.4 Regional models	18
1.2.5 Ocean models	19
1.3 Data formats	20
1.3.1 NetCDF	21
1.3.2 HDF	22
1.3.3 GRIB	23
2 Storage, processing	25
2.1 Elasticsearch	25
2.2 ecCodes	26
2.3 Alternatives	27
3 Raster overlay serving	28
3.1 OGC Web Services	28
3.1.1 WMS	28
3.1.2 WMTS	28
3.1.3 WCS	29
3.2 GeoServer	29
3.3 GDAL	29
3.4 GeoTIFF	30
4 Implementation of API	31
4.1 Used data	31
4.2 Extracted variables	33
4.3 History weather	35
4.3.1 Download	35
4.3.2 Process	37
4.4 History waves	40
4.4.1 Hourly data	40
4.4.2 Monthly hindcast	41
4.5 Elasticsearch mapping	42
4.6 API in Node.js	43
4.7 Forecast API modifications	45
5 Implementation of overlays	47
5.1 Used data	48
5.2 Download and process	49

5.2.1	Forecast	49
5.2.2	History	51
5.3	GeoServer configuration	53
5.3.1	Workspaces, Stores	53
5.3.2	Layers	54
5.3.3	SLD styles	55
5.3.4	Other	56
6	Further remarks	58
6.1	Storage design	58
6.2	Scripts improvements	60
6.3	Overlay remarks	61
	Conclusion	66
	References	68
	Content of figures	73
	List of abbreviations	74
	E-attachments	75
	Appendix	76

Introduction

Shipping is perhaps the most international of all the world's great industries - and one of the most dangerous. When a ship embarks on its journey, it is influenced by multiple conditions caused by the weather. Waves, wind and currents will have effect on the speed and fuel consumption. Marine safety, mainly high wind and wave avoidance is of great interest for example to ship routing solution providers.

This thesis topic was created in cooperation with vesseltracker.com GmbH, where author had a part-time internship. Vesseltracker.com is a leading provider of global AIS vessel movements and maritime information services with real-time and historical data delivery, port events and a database of over a million vessel and port images (About, 2018). It also operates a shipDB - comprehensive vessel database of specifications, characteristics, equipment, ownership and management information. Vessel positions are regularly updated in a web map application Cockpit.

Weather and wave forecast data systems have already been running at Vesseltracker to some extent before diploma thesis topic was chosen. They included weather and wave forecast API returning current situation and forecast for given place on earth from freely accessible NOAA Global Forecast System data, while using MongoDB as a database. Raster overlay serving system has been composed of high wind and wave tiles rendered from temporary shapefiles for chosen low zooms, both of which can be seen in Attachment 1 and Attachment 2.

Two main objectives were set out for the diploma thesis. First task was to create a history weather and waves downloading, processing, storage and access service and review current and forecast weather and wave API system, since several performance issues have been spotted over time. The history weather and waves API should allow querying for latitude, longitude, timestamp and return closest data point in time and space while listing all contained weather and wave characteristics stored. It should also allow querying for a single point and timestamp in the form of interval. Second objective was to create a new raster overlay serving system presenting several weather and wave variables for history, current and forecast. It would be regularly updated to match new weather and wave model releases.

Raster overlays should then be attachable to Vesseltracker Cockpit web map application, which currently uses Leaflet as a JavaScript web mapping library.

Although a complete freedom was given to the author on the matter of used programs or programming languages, it was expected that both systems will be running on a single instance machine running Ubuntu Linux 16.04. Both objectives should utilize open-source data and software if possible and be launchable on above mentioned Ubuntu system.

There have been multiple reasons why a new history weather API and a raster overlay system should be included in Vesseltracker support and visualization services. As history ship track data are one of main sought-after information data sources, they could be easily augmented with weather and wave information along the track. There have been several cases, where an explanation for a specific vessel accident on open sea connected to weather conditions was needed up to several years in the past. For example, several containers contents were damaged by on-board freezing during the journey. It had to be inspected afterwards whether there was a sudden extreme decrease of outside temperature on any part of the voyage or the crew made errors while controlling the conditions of the goods. The interval history API query could be used for providing another source of weather information inside ports, offshore facilities or stationary ships.

Raster overlays on the other hand offer a convenient way, how to compare weather and waves on a more global scale. Overlays are also an automatically expected feature by customers. The aim of this project on the other side is not to compete with already saturated market of full-stack commercial weather data providers, rather offer an independent, self-reliant and sustainable service based on open-source solutions and datasets.

1 Meteorology and data

Oxford dictionaries propose a definition of meteorology as follows: The branch of science concerned with the processes and phenomena of the atmosphere, especially as a means of forecasting the weather (Meteorology definition, 2018). Four main weather forecasting methods are recognized today: Climatological, Analog method, Numerical Weather Prediction and Persistence and Trends Method (Other Forecasting Methods, 2018).

To delve into a world of a weather forecaster, assume a rotating sphere that is 12,800 kilometers in diameter, has a bumpy surface, is surrounded by a 40-kilometer-deep mixture of different gases whose concentrations vary both spatially and over time, and is warmed, along with its surrounding gases, by a nuclear reactor, 150 million kilometers away. Sphere is revolving around the nuclear reactor and some locations are warmed more during one part of the revolution than the other. The mixture of gases continually receives inputs from the surface below, usually calmly but sometimes through violent and highly localized injections. After watching the gaseous mixture, you are expected to predict its state at one location on the sphere one, two or ten days into the future (Ryan, 1982).

1.1 History of meteorology

As far as current history knowledge can tell, the first documented attempts to predict weather go far into 7th century B.C. (Graham, Parkinson, & Chanine, 2002). At that time Babylonians attempted to predict short-term weather changes based on cloud observations and halo. Meteorology (Meteorologica in Latin), a philosophical treatise containing among other also theories about the formation of rain, clouds, wind, hail, thunder, lightning and tornadoes was written by a Greek Aristotle around 340 B.C. His four-volume text was considered by many to be the authority on weather theory for almost 2000 years and it was not until 17th century that many of his erroneous ideas were corrected or disproved.

By the end of Renaissance, it had become evident that to deepen the understanding of character and phenomena of the atmosphere, instruments were needed to measure its properties, such as moisture, temperature and pressure. A first hygrometer in Europe, an instrument to measure humidity of air, was built by Leonardo da Vinci in the 1400s (Bellis, 2017). Galileo Galilei invented an early thermometer in 1592, Evangelista Torricelli created

the barometer for measuring atmospheric pressure in 1643 (Graham, Parkinson, & Chanine, 2002).

Although technological improvements and new device designs were made in decades and centuries after, it was by the invention of the telegraph and its network in the middle of nineteenth century which allowed transmission of weather observations from observers to compilers and therefore enhanced the overall forecasting process (Graham, Parkinson, & Chanine, 2002). Observing stations started to appear across the planet, which led to first synoptic forecasting, a method based on the compilation and analysis of many observations taken at a given time over a larger area. More data were becoming available for observation-based forecasting in the start of twentieth century from regional and global observation networks. From the invention of radiosonde in 1920, measuring of temperature, moisture and pressure can also be performed in different altitude levels up to 30 kilometers above the surface (Graham, Parkinson, & Chanine, 2002). Small boxes equipped with weather instruments and a radio transmitter are carried into higher levels of the atmosphere by a hydrogen or helium-filled balloon that ascends naturally. Until the moment of balloon burst, the instruments collect measurements and send them back to ground stations.

The idea of describing state of the atmosphere by a set of mathematical equations was formulated by Vilhelm Bjerknes in 1904 and later developed and documented in *Weather Prediction by Numerical Process* by Lewis Fry Richardson in 1922. A result of Richardson's calculations was a six-hour forecast, which however took him several weeks to compute alone. It was clear from his report that many computations needed to be solved very quickly to produce a timely forecast. The first successful simplified weather prediction was made in 1950 using one of the earliest computers by a team of meteorologists and mathematicians at the Institute for Advanced Study in Princeton, New Jersey. By the year 1955, numerical forecasts were already being made on a regular basis in North America.

Over next years, National Meteorological Center models used more advanced equations for atmospheric dynamics and thermodynamics and other input data sources ranging from various satellite measurements to increased number of ground weather stations (Graham, Parkinson, & Chanine, 2002). In addition to providing visual images, satellites also

provide data that allow direct calculation of atmospheric temperature, moisture profiles and other environmental variables, throughout the atmosphere column.



Figure 1 ENIAC - computer used to create the first numerical weather forecasts

Source: https://upload.wikimedia.org/wikipedia/commons/6/6c/ENIAC_Penn1.jpg

Increasing of computing power, never-ending desire for better results of weather forecasts and expanding knowledge of weather influencing factors led to improvements and increasing of complexity of differential equation models in 1970s. Solar radiation effects, moisture, latent heat, feedback effects from rain on convection, sea surface temperature and sea ice influence were added to model variables (Schuman, 1989). During 1980s meteorologists began experimenting with including the interactions of soil and vegetation with the behavior of atmosphere, which also meant more realistic forecasts (Yongkang & Fennessey, 1996). From 1973 models were computed on global scale too, until then they encompassed only Northern Hemisphere (Kalnay, 2003).

For further understanding of history developments overview, a brief introduction into numerical weather forecasting is presented in 1.2. Edward Lorenz made a statement in 1963 that long-range forecasts prepared for more than two weeks in advance fail to predict the state of the atmosphere (or any similar chaotic fluid system). This is caused mainly by the

propagation of multiple model errors and their rapidly increasing influence over time. Generally, four types of errors dominate (Manousov, 2006):

- Model equations do not capture atmosphere processes fully.
- Model resolution (horizontal and vertical) does not suffice to capture all features.
- Initial observations describing the initial state of the model run are not available at every point of the atmosphere.
- Initial observations also cannot be measured to infinite precision.

Complex set of equations called the Liouville Equations contains the uncertainty of the initial conditions (Manousov, 2006). They produce a set of forecasts (distribution of possible forecast states) for an input range of possible initial states of the atmosphere. Both the input and the output of mentioned equations are given in the form of a Probability Density Function. This process is called ensemble forecasting and is very computationally demanding, which was the main reason, why ensemble forecasts started to be produced quite late – in 1992 by the European Centre for Medium-Range Weather Forecasts (ECMWF) and the National Centers for Environmental Prediction (NCEP).

1.2 Models

The basic idea of weather forecasting through numerical models takes the known state of the fluid (atmosphere variables) at a given time and uses equations of fluid dynamics and thermodynamics to estimate the state of the fluid at some time in the future (WRF Introduction, 2007). The observations are processed by various data assimilation algorithms and values at locations usable by the model's mathematical algorithms are obtained (an evenly spaced grid for example). Variables are then used in the model as the starting point for forecast primitive equations. These equations lead to rates of change predictions for the atmosphere state in short time in the future. What usually follows is an iterative process of applying these equations to the always new state until a desired forecast date is reached. The length of the time step is related to the distance between grid points. Many atmospheric and land-soil variables are available within model results datasets.

1.2.1 Global Forecast System

Global Forecast System (GFS) is the main weather forecast model produced by NCEP (Global Forecast System, 2018). Before January 2003 the system was composed of GFS Aviation and GFS Medium Range Forecast. GFS is made up of an atmosphere model, an ocean model, a land/soil model, and a sea ice model, which together try to provide the most accurate picture of the weather. This type of a model is called fully coupled because it tries to solve the full equations for mass and energy transfer and radiant exchange.

Forecasting capabilities of GFS are being improved every now and then, mainly by not so frequent upgrades to existing supercomputers and storage capacity used for calculations allowing more data inputs and higher spatial resolution. These actions happened in greater scale for example after a Superstorm Sandy hit New Jersey in 2012 (Kravets, 2015). While ECMWF model predicted the storm to hit the shore seven days in advance, GFS kept predicting that it will turn back to the sea until four days before the event. The main reason which was afterwards conveyed to the media was the lack of computing resources. In response to that NOAA boosted their computing power ten times from 426 teraflops to 4.6 petaflops. In 2018 a phase three of a multiyear process of improving the GFS was finished (News&Features, 2018). It added 2.8 petaflops of speed, thus increasing NOAA's total operational computing speed to 8.4 petaflops and added 60% storage capacity. Next generation of GFS should be launched during 2019. It should increase the resolution to 9 kilometers and 128 levels out to 16 days into the future, compared to the current 13 kilometers and 64 levels running 10 days into the future.

GFS is licensed under U.S. Government Works - not subject to copyright restrictions (U.S. Government Works, 2018). That would be one of reasons why it is used (with other models) by most of weather forecast display websites. There are more ways how to obtain model history and forecast data from GFS. Gridded data are available for download either by direct HTTPS or FTP access, or through the NOAA National Operational Model Archive and Distribution System (NOMADS), Archive Information Request System (AIRS) or Thematic Real-time Environmental Distributed Data Services (THREDDS) (Global Forecast System, 2018). Further information about the model data used for this project is mentioned in chapter 4.1.

1.2.2 ECMWF Integrated Forecasting System

The comprehensive Earth-system model developed at ECMWF in co-operation with Météo-France has a name The Integrated Forecasting System (IFS) and forms the basis for all data assimilation and forecasting activities. It is composed of a spectral atmospheric model with a terrain-following vertical coordinate system together forming a 4D system. (Andersson & Thépaut, 2008) ECMWF runs the IFS in several configurations and resolutions while other ECMWF member states use ECMWF global forecasts to provide boundary conditions (explained in chapter 1.2.4) for their own limited domain forecasts with desirably higher resolution. The highest resolution configuration (HRES) is run every twelve hours out to ten days, averaged over many forecasts with a 9 km horizontal resolution using 137 vertical layers (Documentation and support, 2018). 50-member ensemble system - ENS is run every twelve hours out to 15 days with 18 km horizontal resolution and 91 vertical layers. Another member of the ensemble CNTL - Control forecast has lower spatial resolution than the HRES but utilizes the most accurate estimate of the current conditions and the best description of the model physics.

ECMWF forecasts are provided free to the national weather services of its members. Commercial users are obliged to pay a fee. Selected variables from the HRES and ENS are available directly to clients under the noncommercial license Creative Commons forbidding derivative work (Licences available, 2018).

1.2.3 Other global models

Global Environmental Multiscale Model has been running since 1991 (Ritchie & Beaudouin, 1994) to serve operational weather-forecasting needs of Canada by the Recherche en Prévision Numérique, Canadian Meteorological Centre and the Meteorological Research Branch (Introduction GEM, 2018). Operational model is currently used for the global data assimilation cycle and medium-range forecasting, outputting forecast for up to ten days. Among main differences from above mentioned models are following:

- Semi-implicit, semi-Lagrangian time discretization scheme, which removes the restrictive time step limitation of a conventional Eulerian scheme,
- Hydrostatic-pressure-type hybrid vertical coordinate system,

CTU in Prague

- Horizontal variable-resolution, cell-integrated, finite-element discretization which reduces the usually staggered finite differences discretization at uniform resolution in spherical geometry.

One of further linked projects plans to develop a single highly efficient model that can be reconfigured at run time to run globally at uniform resolution, or to run with a variable and rotatable resolution over a global domain so that high resolution is focused only over desired area. Output from Global Environmental Multiscale Model is under Crown copyright but is issued under a free license if properly attributed to Environment Canada and is therefore redistributed by many other websites for example by NOMADS (NOMADS, 2011).

Navy Global Environmental Model - a global numerical weather prediction model is run by the United States Navy's Fleet Numerical Meteorology and Oceanography Center in Monterey, California (Naval Oceanography Portal, 2018). It became operational in February 2013, replacing the previous naval global system NOGAPS. It runs four times a day and produces the forecast with three-hour intervals 180 hours into the future. While being licensed under U.S. Government Works, it is not subject to copyright restrictions but is not frequently used by other weather forecast displaying websites.

Unified Model - a NWP model developed by the United Kingdom Met Office has been running since the end of the 1980s (History of the Unified Model, 2007). A single model is used for a range of both timescales and spatial scales. The Unified Model is run operationally in many configurations for weather forecasting at the Met Office. (Numerical Weather Prediction models, 2017) Global deterministic model has a horizontal resolution 10 km, 70 vertical levels, is produced every 6 hours with forecast up to 6 days into the future. Forecasts are available only 72 hours into the future for non-paying users. Unified Model licensing is targeted individually to fulfill specific user needs. (Unified Model, 2018) A regional version of the model containing the main area of the United Kingdom with 1.5 km spatial resolution runs every three hours. The Met Office has the capability to quickly relocate regional models to any interested area worldwide. These Crisis Area Models are run in support of allied military operations and disaster relief operations.

GME, a global numerical weather prediction model run by the German national meteorological service started operational production in December 1999. It was the first

numerical weather prediction model running on an icosahedral grid instead of traditional latitude-longitude grid (ICON, 2018). Compared to those, icosahedral grids provide an almost homogeneous coverage of the earth. This avoids the pole problem, related to the convergence of meridians in traditional grids, which can pose challenges to finding a computationally efficient implementation. Later the model was succeeded by the Icosahedral Nonhydrostatic model (ICON). The global ICON grid has an effective mesh size of about 13 km. It runs its forecast four times per day, 180 hours into the future for the forecasts starting at 00 or 12 hours UTC and 120 hours for those starting at 06 and 18 UTC. If not specially mentioned otherwise, all datetimes in this thesis are using UTC time. To be able to cover small area features for example in the Rhine Valley, Deutscher Wetterdienst additionally operates the high-resolution regional model COSMO-DE with a mesh size of 2.8 km. Another model which provides forecast for the area of whole Europe is a nested area ICON-EU with a mesh size 6.5 km and a forecast range of 120 hours (ICON-EU, 2018). The GME's approach of non-latitude-longitude grid is being currently utilized by the NCEP produced experimental Flow-following, finite-volume Icosahedral Model.

1.2.4 Regional models

Regional models use smaller grid spacing than global models and are focused on a smaller area than global models. They can cover meteorological phenomena of smaller dimensions that are not possible to be represented on the coarser grid. Regional models usually use a global model for determining the initial conditions of the edge of their domain space. Uncertainties and errors are introduced by the global model initial conditions values, as well as regional model itself. As the focus of this thesis output is global, only a short overview of selected regional models is given.

North American Mesoscale Model is run by NCEP to provide mesoscale forecast to public and private sector meteorologists. It is produced four times a day with 12 km horizontal resolution, 84 hours into the future with three-hour temporal resolution (NAM, 2018). Four fixed local nested domains run to 60 hours at 3 km resolution.

Aire Limitée, Adaptation Dynamique, Development International (ALADIN) mesoscale model for France was proposed in 1990 (About Aladin, 2018), first launched in 1995 and is

based on the ECMWF model. It has been used in over 20 European countries to create short-range weather forecasts.

Private Numerical weather prediction (NWP) models are also not that uncommon. One example of a company producing NWP is meteoblue. Their weather models are based on the Nonhydrostatic Meso-Scale Modelling technology (Weather modelling, 2018). Each domain (areas covering parts of continents) is divided into grid cells which are rectangularly arranged and evenly spaced between each other. The average distance between the grid points varies from 1 km to 25 km. Models usually have 55 vertical levels. Currently two types of models are offered to public customers by different licensing methods – NEMS operating from 2013 and older NMM operating from 2007.

1.2.5 Ocean models

Ocean wave modelling tries to predict the evolution of the energy of wind waves using numerical methods. These simulations consider atmospheric wind forcing, non-linear wave interactions, frictional dissipation, sea ice and iceberg influence and ocean or tidal current effects (Wavewatch manual 5.16, 2016). Their outputs are wave heights, wave periods, and dominant directions either only summarized into the significant wave height, which is the average height of the one-third largest waves or divided into appropriate parts - wind waves and up to three swell waves.

After not so successful (in the terms of accuracy) first attempts in 1960s and 1970s to numerically predict ocean waves behavior a second generation of wave models was created (Robinson, 2010). Increased activity of satellites monitoring the earth surface due to high demand for satellite wind measurements for NWP enabled that the wind effect could be weighted more accurately in the wave prediction equations. Second generation models also introduced terms which took energy transfer between spectral components in a simplified way into consideration. With improvements of NWP, it was revealed that describing of what was really happening in the ocean in all possible conditions was still far by optimal state. It led to the creation of third generation of wave models which are being run from 1988 onward. They are depicting the non-linear wave-wave interaction much more accurately. The spectral wave transport equation also describes the change in wave spectrum over ocean topography. Even with the ever-increasing importance of data provided by satellites, since it had a great

influence on the precision of ocean wave models, weather buoys (before the end of World War II also weather ships) have been the largest source of relevant input data for wave predictions.

NCEP operates a third-generation wave model WAVEWATCH III. It is a further development of the model WAVEWATCH, as developed at Delft University and WAVEWATCH II, developed at NASA, Goddard Space Flight Center (Wavewatch III, 2016). It runs four times a day while each run starts with 9, 6 and 3-hour hindcasts (forecast into the past) and produces forecasts with 3 hours interval from the initial time out to 180 hours. WAVEWATCH III has a global domain of approximately 30 km horizontal resolution, with some nested regional domains of interest. WAVEWATCH III has U.S. Government Works license - not subject to copyright restrictions and therefore its forecast multigrid data are often used by many weather forecast visualization websites.

ECMWF runs several wave models. High Resolution Wave Model (HRES-WAM) is coupled with the global model HRES while High Resolution Stand Alone Wave model (HRES-SAW) runs by itself. They offer the same parameters globally, but the resolution and the dissemination schedule are different (Datasets Set II, 2018). HRES-WAM has a global reach with a spatial resolution 0.125° and is produced four times a day with 1-hour forecast step up to 90 hours, 3-hour step up to 144 hours and 6-hour step up to 240 hours into future. HRES SAW is produced two times a day with coarser time resolution of 3-hour step up to 144 hours and 6-hour step up to 240 hours. The type of licensing of ECMWF wave products is shared with their global weather model counterparts.

Among models produced by DWD two wave models can be found. Global GWAM model with 27 km spatial resolution, 3-hour steps and 180 hours of forecast and regional EWAM with 5 km resolution for Europe area with 1-hour steps and 78 hours of forecast are produced twice per day (Ocean wave models, 2018). Winds that drive these wave models are provided by the ICON forecasts.

1.3 Data formats

Never-ending technological advancements mainly in collected data precision lead to increasing demands on storing, reading and analyzing these data. Multidimensional data are

also more common than before. Meteorological data can have up to five dimensions: X, Y, Z coordinates, timestamp and temperature. Suitable data formats needed to be developed to provide a possibility for real-time data processing and assimilation. Among most notable multidimensional data formats used in meteorology are Network Common Data Format (netCDF), Hierarchical data format (HDF) and Gridded Binary (GRIB) (Common Climate Data Formats: Overview, 2013). They are portable (machine independent), self-describing (the user does not have to know file's internal structural details) but also describable by metadata (for example information about units). Each of these file formats has evolved over time to address the needs of various users. Unfortunately, the newer versions of the same format are not always backward compatible. A small difference between the above mentioned is that netCDF and HDF are file formats while GRIB is a record format.

1.3.1 NetCDF

NetCDF is an open-source, Open Geospatial Consortium (OGC) standardized set of software libraries and self-describing data formats that support writing, reading and sharing of array-oriented multidimensional scientific data (Introduction NetCDF, 2018). The project is developed and maintained by Unidata – part of the University Corporation for Atmospheric Research. The main library is written in C, and provides an API for C, C++, two APIs for Fortran and another independent implementation written in Java. Interfaces to netCDF based on the core library are also available for other languages including Python, R, Haskell, Perl, Ruby, MATLAB and more. The API specification is very similar across different languages. Many software packages have been created which make use of netCDF files ranging from command line utilities to graphical visualization packages.

It is highly scalable, allowing access to small subsets of large datasets and shared usage, where a single user is writing to a file while other can read it in the same time. Currently four versions of the netCDF format exist (File Structure and Performance, 2018). The classic format (netCDF 3), which is still the default format for file creation, the 64-bit offset format which was introduced in version 3.6.0, and supports larger variable and file sizes, the netCDF-4/HDF5 format enabled in version 4.0, HDF4 SD format for read-only access and CDF5 format. NetCDF files should have the file name extension .nc.

NetCDF classic dataset is stored as a single file comprising a header, containing all the information about dimensions, attributes, and variables and a data part, containing fixed-size data and variable-size data. The header also contains dimension lengths and information needed to map multidimensional indices for each variable to their offsets. The fixed-size data part following the header contains the data with each variable stored contiguously. The last part, if it is present, consists of variable number of fixed-size records where record data for each variable is stored contiguously in each record. NetCDF-3 does not support compression, string variables or parallel processing, which is a reason why netCDF-4 - a subset of HDF5 with netCDF-3 style API interfaces to create and access the data, was developed.

1.3.2 HDF

HDF is an open-source data format and a set of libraries used for storing voluminous numerical data created by National Center for Supercomputing Applications to address the issue of using many different data formats for the Earth Observation System project. Since 2005 it has been developed and maintained by a non-profit organization HDF Group (History HDF Group, 2016). The HDF Group's public mission is to provide the development of HDF in the long term and the ongoing accessibility of HDF-stored data (Our Mission, 2018).

HDF is self-describing, highly scalable, platform independent, allowing hierarchical search and display of the data in their true form (unlike a relational database). Binary data can be stored too. It allows to specify complex data relationships and dependencies. Multidimensional objects are supported where each element can form a single complex object (Tutorial HDF5, 2017). Objects can be merged into a single file and accessed as a group or independently. HDF can be easily converted to netCDF and vice versa. HDF files can have various file extensions based on the format versions, .hdf, .h4, .hdf4, .he2, .h5, .hdf5, .he5. Currently two incompatible and very different versions exist.

HDF4, being an older but still maintained and used format, has many limitations, which led to development of HDF5. While not being allowed to have data files larger than 2GB because of the usage of 32-bit signed integers for addressing (HDF4 FAQ, 2017), it is not suitable for modern scientific applications. Additionally, with limited parallel input and output support, a single dimension data types and other limitations, it needed to be succeeded.

HDF5 simplifies the file structure to include only two major object types. Those are datasets, multidimensional arrays of a homogeneous type and groups, which are container structures holding datasets and other groups. Metadata attached to groups and datasets appear in a form of user-defined named attributes. More complex storage API representing images and tables can be built. HDF5 performs well for time series data due to the usage of B-trees to index table objects (HDF5 Tutorial, 2017). Many other data formats have been based on HDF5, for example PyTables, which is built on top of the HDF5 library, using the Python programming language and NumPy package (PyTables, 2018).

1.3.3 GRIB

GRIB is a binary data format designed to store results from most NWP systems in 1985. The creation was based on activities of World Meteorological Organization (WMO) and Commission for Basic System. It is standardized as GRIB FM 92, described in WMO Manual on Codes No.306 (Introduction GRIB1-GRIB2, 2003). The reason for the development was a need for a data format allowing fast transfer of large volumes of numerical data and allowing effective storage.

GRIB files are a collection of self-contained records of two-dimensional data. The individual records stand alone as data without any reference to other records or to the schema. Collections of GRIB records can be appended to each other. Each GRIB record has a header (with pointers towards elements in predefined and internationally agreed tables - in the official WMO Manual on Codes) and the actual binary data. The parameter characteristics like name or unit must already be defined in the tables of the WMO Manual on Codes. There are two editions of the GRIB FM 92 currently operational – GRIB 1 and GRIB2.

GRIB 1 contains a collection of entries where each one contains gridded data of specific time and vertical level. A single GRIB 1 message enables the transmission of one single field on a single grid at a single level or layer. The number of entries is limited to 256, while half of them is only for coordinates, second half is for the record and the last one is for the missing parameter (Introduction GRIB1-GRIB2, 2003).

WMO approved the Edition 2 of FM 92 GRIB for an operational WMO code in November 2001 to solve weaknesses of transmission and archiving of specific types of data – mainly

multidimensional and spectral data (FM 92 GRIB Edition 2, 2003). A freeze on new development has been placed on the Edition 1 of FM 92 GRIB on the request of GRIB 1 continuation by International Civil Aviation Organization to extend the possibility of its operational use for their purposes. The Edition 2 can be used in new products, such as the output of ensemble prediction systems and ensemble wave forecasts, long-range forecasts or transport models. GRIB 1 format is still compatible within GRIB 2 though.

GRIB 2 allows coding of multi-grid, multi-product, multi-level or multi-parameter fields in a single message with the maximum number of entries increased to 65536. As for the compression possibilities, data in GRIB 1 are usually converted to integers using offset and scale and then bit-packed, while GRIB 2 has multiple other methods of compression. There is no inviolable rule about file extension of GRIB files, all of these: .grb, .grib, .grb2, .grib2, .GRB, .GRIB, .GRIB2 can be seen frequently.

2 Storage, processing

Raw weather data collected from various sources do not necessarily have a direct value regarding the state of the weather to a human observer. However, results from the NWP in the GRIB format usually contain data which are either directly usable or specific weather characteristics can be easily computed from them (wind speed and direction from u , v wind vector components for example). This chapter introduces two main tools which served the purpose of storage and processing of raw weather data on this project.

2.1 Elasticsearch

Elasticsearch (ES) is an open-source, enterprise-grade search engine based on Apache Lucene, built to handle large volumes of data with high availability and with ability to distribute itself across any number of machines with unlimited horizontal scaling in mind. It has a simple but powerful API that allows applications written in various programming languages to access the database easily (Divya & Goyal, 2013). Elasticsearch is best thought of as an interface to Lucene designed for large volumes of data from the beginning.

It was created by Shay Banon – initially as a search engine for his wife’s list of cooking recipes. He open-sources it in the beginning under Apache License 2.0 which allowed a user community to emerge quickly. The initial public release happened in February 2010 (History of Elasticsearch, 2018). Elasticsearch Inc. was founded in 2012 and it later took two other open-source projects under its wings – a pluggable log ingestion tool Logstash and the UI for data visualization Kibana, together forming the Elastic Stack. In 2015 the company was rebranded to Elastic to represent the growth of the software ecosystem and avoid confusion of the company name with the Elasticsearch engine itself. Elasticsearch and Kibana have been also available as a service for Amazon Web Services (AWS), together named Elastic Cloud. Following the initial Open-source minded attitude, Elastic opened the code of their previously commercially provided packages for security and monitoring, X-Pack to the public in 2018.

Elasticsearch accesses and stores objects through a Representational State Transfer (REST) API by PUT, POST, GET, and DELETE methods. As for the storage type, it is a typical example of a key value store - every piece of data has a defined index and type. Index as a collection of documents or can be thought of as a table in a database. Elasticsearch achieves fast text search responses because instead of searching it directly, an index is looked up

instead (Novoseltseva, 2017). It is called an inverted index, because it inverts a page-centric data structure to a keyword-centric. The documents added to an index have no defined structure and field types, which can be advantageous and disadvantageous in the same time.

From a REST approach, the relative path to the object is `/{index}/{type}/{id}`. Indices and types are created, retrieved and deleted at runtime via a REST API. The sharding (database partitioning that separates large databases into smaller parts called data shards) and data replication (copying of index parts) can be chosen differently for each index (Divya & Goyal, 2013). Elasticsearch performs indexing of stored documents either with a dynamic mapping or user provided mapping. The search API is provided by a `_search` resource at a server level, index level or type level. Elasticsearch offers a domain specific JSON based language to specify complex queries. The main strength of Elasticsearch as an engine lies in text search allowing full-text search, search with spelling errors and with abbreviations considered, while the results can then be filtered, sorted and ranked by various scoring techniques (Elasticsearch, 2018). It is however well suited for storing and searching of other data types like numerical, geodata, timestamps etc.

2.2 ecCodes

EcCodes is a software package created by ECMWF and licensed under the Apache License, Version 2.0 (ecCodes license, 2015). It provides an application programming interface and a set of tools for decoding and encoding messages in the following formats of WMO FM-92 GRIB edition 1 and edition 2, WMO FM-94 BUFR edition 3 and edition 4 and WMO GTS abbreviated header (only decoding) (What is ecCodes, 2016). C, Fortran 90 and Python interfaces provide access to the main ecCodes functionality. It is designed to provide a simple set of functions to access data from several formats with a key/value approach. EcCodes is an evolution of ECMWF software package GRIB-API, but starting from version 2.0.0, it has been selected as the default GRIB encoding and decoding package used at ECMWF and further development of GRIB-API is planned to be discontinued towards the end of year 2018. A low-level API interface is also being developed by ECMWF – currently available for Python 3. It provides almost one to one mappings to the C API functions and uses the NumPy module to handle data values.

Python High level API cfgrid designed to map GRIB files to the NetCDF Common Data Model has been developed jointly by ECMWF and B-Open (cfgrid , 2018). Currently being in development status beta, it aims to provide a reading GRIB backend for xarray module and in the future with limited write capabilities. It is however still dependent on ecCodes C library. Among main features are reading the data lazily and efficiently in terms of both memory usage and disk access and mapping a GRIB 1 or 2 file to a set of N-dimensional variables.

2.3 Alternatives

Reading, writing, modifying or generally working with GRIB files is not only a domain of ecCodes. Wgrib2, a program to manipulate, inventory and decode GRIB 2 files would do the same service (wgrib2, 2016). Source code modules for wgrib2 are either in the public domain or under the GNU license. Other notable functionalities include creating subsets, export to IEEE, text, binary, CSV, NetCDF and MySQL, write of new fields, parallel processing by using threads or flow-based programming, Fortran and C interface.

Database selection is crucial for a design of a weather processing system. There are many alternatives to Elasticsearch among SQL and No-SQL databases, either adapted to work specifically with temporal data or offering general functionality. Discussion about the choice and alternatives is presented in chapter 6.

3 Raster overlay serving

This chapter provides a brief introduction to relevant OGC Web Services which implement the standards for raster data serving, and introduces two main pieces of software, which enabled producing and serving of raster overlays as well as GeoTIFF image format, which is a first candidate when raster format manipulations come to mind.

3.1 OGC Web Services

Open Geospatial Consortium (OGC) is an international industry consortium of companies, public agencies and universities working in the joint effort to develop and maintain publicly available interface standards and encodings for handling geospatial data and systems. OGC Web services include services for data access, data display and data processing (About OGC, 2018). OWS requests are defined using the HTTP protocol and encoded using key-value-pairs structures or Extensible Markup Language (XML), which is more common.

3.1.1 WMS

A Web Map Service (WMS) produces maps of spatially referenced data dynamically from geographic information (Implementation specification WMS, 2006). The standard defines three operations: service-level metadata return, map return and optional third returns information about features shown on a map. WMS operations can be invoked by submitting requests in the form of demanding a specific URL. Produced maps are usually rendered in an image format such as PNG, GIF, JPEG or vector based SVG. The most recent version of the standard is WMS 1.3.0.

3.1.2 WMTS

A Web Map Tile Service (WMTS) standard provides a solution to serve digital maps using predefined image tiles of known size (Implementation specification WMTS, 2010). The service shows the list of tiles that it can provide through a standardized format in a metadata document. The document also declares the communication protocols and encodings which allow clients to interact with the server. The declaration defines the tiles available in each layer, visualization style, output format, scale, coordinate reference system, and geographic

fragment of the total area. The WMTS standard complements the WMS. The most recent version of the standard is WMTS 1.0.0.

3.1.3 WCS

The OGC Web Coverage Service (WCS) supports electronic retrieval of geospatial data as coverages, digital geospatial information representing space and time-varying phenomena (Service implementation, 2018). WCS allows clients to choose portions of server-provided information based on spatial and temporal constraints and other criteria. WCS provides data with their detailed descriptions, defines a syntax for requests against these data and returns them with original semantics, which may be interpreted, extrapolated and not just portrayed. The most recent version of the standard is WCS 2.1.

3.2 GeoServer

GeoServer is an open-source (GPL license) server software that allows users to view, share, process and edit geospatial data and is built on an open-source Java GIS toolkit GeoTools (Documentation, 2018). GeoServer is a OGC compliant implementation of many open standards, enabling reading of a vast number of data formats out of the box and many other by adding community extensions. It uses the Spring Framework to provide a request dispatch architecture for modules. Being a web application, it supports any common servlet container. GeoWebCache, a Java-based tile caching component is also bundled with GeoServer but can be optionally removed. The application provides a REST API implemented using the spring-mvc-framework, which enables for example batch configuration changes thus allowing easy configurations of hundreds of data stores with ease. Further information can be found on project documentation pages, configuration options used in this thesis are described in chapter 5.3.

3.3 GDAL

The Geospatial Data Abstraction Library (GDAL) is an open-source software library (X/MIT license) that allows to perform geographical analysis on vector and raster data. It is usually also built with a variety of command line interface utilities for data translation and processing. Traditionally GDAL was forming the raster part of the library, and OGR the vector

part for Simple Features data model (GDAL, 2018). Starting with GDAL 2.0 they have been more intertwined. It behaves as a swiss knife for raster and vector data and usually forms the basis of geodata manipulation pipeline thanks to the huge number of different data formats (over 200) and sources it can read and some of them also write. The most recent version of GDAL which has been released is 2.3.2. Since version 2.1 GDAL enables a read and write operations on geospatial data stored in Elasticsearch.

3.4 GeoTIFF

GeoTIFF is a very common file format when dealing with geospatial raster data although not really for results of NWP. Due to its versatility, it is as a number one option for storing satellite data, an intermediary file format in raster processing chains or even occasionally raster overlay serving format (although usually JPEG or PNG are preferred). GeoTIFF behaves as a metadata format based on the mature and very versatile TIFF standard and fully backwards compatible (complies with TIFF 6.0 specification) (GeoTIFF Specification 1.0, 2000). It has additional reserved TIFF special tags to allow the storage of georeferencing of the image - projection and coordinate information while not going against the TIFF recommendations or limiting the scope of raster data supported by it.

The popularity of GeoTIFF comes from several reasons: It can store scientific raster data with float or larger integer value types. It allows the use of several methods of compression, such as LZW, Deflate or JPEG as well as the internal tiling of large raster files, thus greatly improving the performance when reading smaller subsets. The different data bands can be interleaved or separate (Sazid Mahammad & Ramakrishnan, 2009).

Due to gradual increase of resolution of provided GeoTIFF imagery from various sources, a proposal for a cloud optimized GeoTIFF was also created. It is targeted to be hosted on a HTTP file server, where internal organization should allow consumption by web clients sending HTTP GET range request for the content like: "bytes: start_offset-end_offset" (cogeotiff/cog-spec, 2018). It contains metadata of the full resolution imagery, followed by the optional overview metadata and the image itself. Therefore, software enabling reading of cloud optimized GeoTIFF can stream just a portion of data, thus improving overall processing times and creating real-time operations on larger raster images, which were previously not feasible. A single file can be also accessed online instead of needing to copy and cache the data.

4 Implementation of API

The following chapter is focused on describing the backend side of an API interface for providing fast access to history weather and wave data for a given latitude, longitude and timestamp input, which will enhance current Vesseltracker Cockpit supportive visualization and auxiliary data tools. API design counts with Elasticsearch as the underlying database and a simple Node.js application controlling access and user input. It is expected that a server with Ubuntu Linux will be used for running both the API and GeoServer for raster overlay serving and all settings were chosen with that in mind.

4.1 Used data

After initial discussion and a brief overview and later review of the existing weather forecast API project (details in chapter 4.7), GRIB2 GFS files for weather and GRIB2 WAVEWATCH III multi grid files for waves have been chosen as data sources for the current project. There are several reasons for that. They are produced regularly and due to recent improvements of the overall NOAA disposed computing power also with denser spatial and temporal resolution than few years in the past. It is provided free of charge licensed under U.S. Government Works in contrast to ECMWF models. There are many ways how and from where to download the data and a user can select which one suits him the best. The unified GRIB2 format allows a single way of reading and processing instead of using a different tool for each new dataset.

History weather data is available as GFS Analysis (GFS-ANL) where on-site measurement data are later used to run the model again with them included as additional parameters. Currently a user can download history data up from 03/2004 from an online storage. For the initial step of the project, only data from 01/2017 onward were selected. There are in total three types of history weather data files with 3-hour time resolution provided four times a day for 00, 06, 12, 18UTC, but not all of them are available for the whole life-span of the releases.

The first model results were published in GFS-ANL 003 GRIB type with 1° spatial resolution where some of the currently desired variables are missing. The filename and URL follow the pattern:

CTU in Prague

https://nomads.ncdc.noaa.gov/data/gfsanl/200701/20070101/gfsanl_3_20070101_1800_003.grb - the _003 part indicates forecast for +3 hours from given time.

From 1.1.2007 onwards the results from the GFS-ANL are also being published in the optimal and for this use-case preferred type GFS-ANL 004 GRIB2 with 0.5° spatial resolution. The filename and URL follow the same pattern, where the only difference is domain version 4: gfsanl_4 and .grb2 file ending.

The last data format type GFS-ANL 003 GRIB2 has been released from 6.4.2017 as a successor to GFS-ANL 003 GRIB files, which have of that date no longer been produced. It has 1° spatial resolution, but all the needed variables are present – as in GFS-ANL 004 GRIB2. GFS-ANL results are currently released irregularly with a variable gap from 2 up to more than 10 days and are always released in a bulk in different day times. It is important to mention that only _003 and _006 forecast run results contain precipitation data (GFS, 2018). For a full history 3-hour spacing coverage, all three _000, _003 and _006 files need to be downloaded and processed. Precipitation data for 0, 6, 12, 18 UTC are taken from the forecast data 6 hours earlier, while for 3, 9, 15, 21 UTC all variables are taken from a single file.

History waves data are available as multi-grid GRIB2 files with spatial resolution 0.5° and appear in two different formats. The first type - WAVEWATCH III Production Hindcast are produced since February 2005 and are carried out in monthly installments where a single file contains all records for one variable from that specific month. For the processing purposes it is important to highlight a minor caveat that wave files from 10/2017 onwards are the result of running the model in two separate 2-week intervals with variable number of days (depending on the number of days in a month) and then concatenating the two files (Production Hindcast Archive, 2018). This change was necessary since NCEP was unable to manage a one month hindcast to run in the maximum allotted operation time. In January 2018 the hindcast production was temporarily discontinued due to lack of resources. After receiving high volume of user feedback showing the demand for the monthly hindcast it was restarted and the data gap was eventually filled. Currently the hindcast is being released with around a month of delay after the end of the previous month, which is a reason why a second type of data for near past is used to bypass the delay. The filename and URL of the FTP resources follow the pattern:

CTU in Prague

`ftp://polar.ncep.noaa.gov/pub/history/waves/multi_1/201701/gribs/multi_1.glo_30m.hs.201701.grb2` - `hs` denoted the variable name, which will be further explained in chapter 4.2.

Second type of wave data used in this project is the WAVEWATCH III forecast data with 1-hour temporal resolution, even though only every third file is downloaded to match the temporal resolution of the hindcast data. WAVEWATCH III Forecast is available on several FTP mirror pages as well as through NOMADS Grib Filter tool, which allows to sub-select only desired specific variables or height levels on the NOMADS server side and therefore decrease the needed storage and bandwidth. For this project the production NCO FTP source was used to download the full GRIB2 file with all variables to allow possible further addition of those data not located inside monthly hindcast (swell and wind waves). The duration of file presence on the production FTP server is around 7 days, so it cannot be downloaded far into the past. The filename and URL of the FTP source follow the pattern:

```
ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/wave/prod/multi_1.20170101/multi_1.glo_30m.t06z.f003.grib2
```

4.2 Extracted variables

Provided history weather files have usually around 250-400 variables depending on the format (GRIB1 having less than GRIB2) and on the choice if `_000` nowcast type of data or `_003/_006` forecasts are being processed. Always only a subset of data was processed and stored in Elasticsearch. The variable selection which would be of any possible interest to the end users was chosen. The decision was partly based on the weather forecast API current usage and demands. Each variable which is redundant would only increase the needed storage size and the Elasticsearch index size and indexing duration. A list of extracted variables for the API is shown in Figure 2.

Figure 2 Extracted variables for history weather source author

Variable	Short name GRIB2	Short name GRIB1	Unit
Temperature at 2m	2t	2t	K
U part of wind vector at 10m	10u	10u	m/s
V part of wind vector at 10m	10v	10v	m/s
Surface temperature	t	t	K
Mean sea level pressure	mslet	msl	Pa

Relative humidity at 2m	2r	r - surface	%
Gust speed	gust	-	m/s
Precipitation rate	prate	prate	kg/m ² s
Sea-ice coverage fraction	ci	icec	0 - 1
Categorical rain	crain	-	0 or 1
Categorical snow	csnow	-	0 or 1
Categorical freezing rain	cfrzr	-	0 or 1
Categorical ice pellets	cicep	-	0 or 1
Total cloud coverage	tcc	tcc	%
Land/sea mask (0 – sea, 1 – land)	lsm	lsm	0 or 1

As for the case of history waves, hindcast FTP archive offers a limited number of variables compared to the production wave forecast. Until the start of year 2017 only significant wave height, dominant wave direction and wave period data were published. Since then swell height, direction and period data have been added. In contrast to that, production forecast additionally contains wind wave and second swell height, period and direction. To enable consistency of production forecast and hindcast for history, swell was left out of the extracted variable list, which is shown in Figure 3.

Figure 3 Extracted variables for history waves source author

Variable	Short name	Unit
Significant wave height	swh	m
Wave direction	dirpw	°
Wave period	perpw	s

Storing information about the motion of wind or water as vectors provides flexibility in how to calculate direction, which is often desired. Oceanographic convention describes wind in terms of the direction towards which the wind is blowing (Butler, 2013). In the oceanographic convention, wind flowing from the south to the north is symbolized by an arrow pointing north. Meteorologists use a special symbol called a wind barb to show the direction from which the wind is blowing. In the meteorological convention, a wind blowing from west to east is symbolized by a barb pointing west. Sea current directions are always symbolized using the oceanographic convention. In order to make it more complicated, wave direction results from numerical wave models follows the meteorological direction (0 – wave coming from north) but the individual wave spectra are archived so that directional information follows the oceanographic convention (Direction for wave fields, 2018).

Note that longitude conversion is needed for further applications because used GRIB files for both weather and waves have a longitude range (0, 360) with Greenwich as a prime meridian, instead of a more common (-180, 180) value range.

4.3 History weather

History weather backend service is contained in a single folder resembling a Python 3 module but not following the recommended module structure, rather being a set of scripts.

Figure 4 historical-weather folder contents source author

```
+---download
+---eccodes
|   +---bin
|   +---include
|   +---lib
|   +---share
+---temp
download.sh
download.py
README.md
.gitignore
elastic.sh
process.py
timetest.py
weather_mapping.json
```

`Download` folder starts empty and will hold all raw downloaded GRIB files. All data processing temporary files are stored in `temp` folder and then deleted from it. As the name implies, `eccodes` folder contains already compiled version of ecCodes software.

4.3.1 Download

During the download part of the procedure, three following scripts are used:

- `timetest.py`: contains a single function `determinefile(datetime)` which based on given input `datetime` object returns the filename and closest `datetime` for data

extraction for both precipitation data and normal data. It is contained in a separate file to avoid code duplication because it is imported in both processing and download part.

Figure 5 Code snippet - finding the closest 3-hour spot to the input time source author

```
def determinefile(time):
    date_time_zero = time.replace(hour=0, minute=0)
    date_times_to_check = [date_time_zero, date_time_zero + timedelta(hours=3),
date_time_zero + timedelta(hours=6), date_time_zero + timedelta(hours=9),
date_time_zero + timedelta(hours=12), date_time_zero + timedelta(hours=15),
date_time_zero + timedelta(hours=18), date_time_zero + timedelta(hours=21),
date_time_zero + timedelta(hours=24)]
    closest_date = min(date_times_to_check, key=lambda d: abs(d - datetime_input))
    ...
```

- `download.py`: contains all download part logic.

It has two mandatory arguments `startdate` and `enddate` in format `YYYY-mm-DDTHH:MM` (2017-01-01T23:45) and a third optional argument `filename` which can be used to redirect list of downloaded datetimes to a different file than default `listofdownloaded` mainly for testing purposes. The example usage of a script with custom datetimes output is:

```
python3 download.py 2017-01-01T01:23 2017-01-11T14:57 mylist.txt
```

The source code with comments can be examined in E-attachments. The operations performed by the script and the motivation are briefly explained further.

After parsing the input datetimes, the script performs a loop from `startdate` to `enddate` while internally always using the closest datetime which has an hour equal to 0, 3, 6, 9, 12, 15, 18 or 21 (times when GRIB files are available either as nowcast or forecast). Inside this loop first the positive presence of files on the remote NOAA server is checked and their size is validated. There is unfortunately no MD5 sum file to control the integrity of downloaded file. The only validity check which is being made is checking the content size in HTTP headers of the file which is to be downloaded. The reason for the seemingly redundant control is that very rarely files are damaged but correctly named. They lack variable amount of data and have variable but usually very small size. The script avoids files, which have their size smaller than three quarters of usual average file size.

Further, in the decreasing order of preference first available of GFSANL4 GRIB2, GFSANL3 GRIB2 or GFSANL3 GRIB files are downloaded if they are not already located on the disk. Always only the highest preference files at disposal are downloaded. It has been quite frequent in the recent past that first GFSANL3 GRB2 files are produced and a few hours or

days later, GFSANL4 GRB2 files are added. This leads to the effect that both files are downloaded and processed but it has no other downside effect than negligible increase in storage demands of raw files. Lastly the successfully downloaded file `datetime` is written to a temporary file `listofdownloaded` for further reading in the processing phase.

- `download.sh`: a small boilerplate bash script

It calls `download.py` to check and possibly download available data for the last 14 days and takes care of redirecting logging output to a separate log file.

4.3.2 Process

The processing part of the import process is performed by two scripts:

- `process.py`: contains all processing logic

It has two mandatory arguments `date` in format YYYY-mm-DDTHH:MM (2017-01-01T23:45) and `eshost` – connection to Elasticsearch host for indexing. The example usage of a script with custom datetimes output is:

```
python3 process.py 2017-01-01T00:00 elasticexample.com
```

The source code with comments can be examined in E-attachments. The operations performed by the script are briefly explained further.

After parsing the input `datetime` the download folder is being checked for all possible files (GFSANL4 GRIB2, GFSANL3 GRIB2 and GFSANL3 GRIB) to find out which types are present and if both needed files (normal and precipitation) are there. If only the precipitation file is available, all variables will be taken from `datetime` 6 hours earlier. If precipitation file is missing, respective variables will be then set to `None`.

EcCodes `grib_copy` tool is called to action to copy selected weather variables into temporary separate GRIB2 files. All ecCodes and other external system tools are called in Python using the `subprocess` module. Temporary files are then merged together into a single GRIB file again by `grib_copy`. GRIB files can be optionally merged simply with a `cat` command:

```
cat file1.grb file2.grb ... fileN.grb > merged.grb
```

To make the compiled ecCodes portable to different Linux machines, processing script locally updates the environment variable of ecCodes definition folder path.

Figure 6 Code snippet - extracts depending on type of GRIB file and available files, function arguments shortName, typeOfLevel and levelType are keys of messages source author

```
extract('2t', 'heightAboveGround', 'sfc')
extract('t', 'surface', 'sfc')
extract('10u', 'heightAboveGround', 'sfc')
extract('10v', 'heightAboveGround', 'sfc')
if fileformat == ".grb2":
    extract('ci', 'surface', 'sfc')
    extract('mslet', 'meanSea', 'sfc')
    extract('2r', 'heightAboveGround', 'sfc')
    extract('gust', 'surface', 'sfc')
if mode in ["normal", "forecastonly"]:
    extractPrecip('prate', 'surface', 'sfc')
if mode in ["normal", "forecastonly"] and fileformat == ".grb2":
    extractPrecip('tcc', 'atmosphere', '10')
    extractPrecip('crain', 'surface', 'sfc')
    extractPrecip('csnow', 'surface', 'sfc')
    extractPrecip('cfrzr', 'surface', 'sfc')
    extractPrecip('cicep', 'surface', 'sfc')
if fileformat == ".grb":
    extract('msl', 'meanSea', 'sfc')
    extract('r', 'heightAboveGround', 'sfc')
    extract('icec', 'surface', 'sfc')
if mode in ["normal", "forecastonly"] and fileformat == ".grb":
    extractPrecip('tcc', 'entireAtmosphere', 'sfc')
```

Another step is to load data into memory and insert it to a dictionary data structure to allow indexing it to ES. Using ecCodes tool `grib_get_data`, the whole `_merged` GRIB file is dumped to subprocess standard output row by row, where a single row contains latitude, longitude, value and additional outputs (here `shortName`). One large dictionary, that will be then indexed has keys in a format `lat_lon_utctimems`, and values forming another dictionary containing all desired variables.

Figure 7 Example of a key and values from weather dictionary source author

```
{1.5_6.0_1533135600000:{"wspd":4.5,"2r":79,"gust":4.5,"lsm":0,"prate":0,"crain":0,"location":{"lat":1.5,"lon":6},"ts":1533135600000,"mslet":1012.8,"ci":0,"tcc":82,"2t":24.8,"t":26.8,"cicep":0,"csnow":0,"cfrzr":0,"wdir":197}, ...}
```

Key does not have to be named in a specific way for Elasticsearch to process, as they will not be queried for later, but key names must be unique, which the current setup fulfils. As can be seen on Figure 7, values are reformatted to have a fixed number of decimal places for each variable, derived variables are created (wind direction and wind speed from u, v vector components) and other variables are converted (temperature from °K to °C, pressure from Pa to hPa, precipitation rate from $\frac{kg}{m^2s}$ to $\frac{mm}{h}$).

If any GFSANL3 file (1° x 1° spatial resolution) is being processed, to ensure fixed resolution of 0.5° x 0.5° for Elasticsearch indices, interpolation is performed using mean of two or four nearest points. If the interpolated point is located on either of integer parallel or integer meridian (-50°, -175.5°) two points are chosen, if not (-50.5°, -175.5°), four points are chosen. Additional emphasis is given to temperature interpolation, as temperature on sea will differ greatly from the one on land just 0.5° further. A land-sea mask was extracted from a GFSANL4 GRIB2 file and saved to a separate .csv file `temp/lsm_005.csv`. These values are then looked up during the interpolation phase. When interpolated position is on land (`land-sea-mask=1`), then mean only from lands around is computed, if there is no land around (island area), oceans are used. The same rule applies for oceans (`land-sea-mask=0`) respectively. This way, mainly the interpolated sea surface temperature will be closer to reality than by using the standard approach. To author's conscience, interpolating other variables than temperature with this method does not have a justified sense.

Last phase is to index the resulting dictionary to Elasticsearch in bulks using the Python `elasticsearch` module. The number of data entries to be sent in a single bulk operation needs to be selected empirically during testing due to different properties (network and disk speed) of each individual created Elasticsearch cluster. Assumption is of course made, that the user launching `process.py` has write access to the cluster.

- `elastic.sh`: a small boilerplate bash script

It loops over all lines in the `listofdownloaded` file from download phase and sends the contained datetimes to `process.py` as an argument. Due to shared usage of downloaded files as GeoServer overlays, additionally `import_weather_history.py` (see chapter 5.2.2) is being run with an argument of contained datetime after `process.py` successfully finishes. Therefore, creating raster overlays and indexing data to Elasticsearch happens in nearly same time. The script also redirects the logging output to a separate log file. Currently used and recommended setup is to create a cron job to launch `download.sh` and `elastic.sh` in a succession once a day during the off-peak hours by adding a following row to `/etc/crontab`

```
30 2 * * * user cd historical-weather && ./download.sh && ./elastic.sh
```

4.4 History waves

As history-waves backend service is in its basics very similar to the ones for history-weather, descriptions of it in the following chapter are much more concise and highlight only differences. As mentioned in chapter 4.1, two different data sources were used for history waves. The folder structure is almost identical as in Figure 4, only `temp` folder is missing and an additional script for monthly hindcast data `download_process_past.py` was added.

4.4.1 Hourly data

On the first glance, parts responsible for downloading and processing wave forecast GRIB files look just like a lightweight version of history weather. During the download phase two following scripts are used:

- `download.py`: contains all download part logic

The only part, where it differs, is a lack of the optional argument `listofdownloaded`. The resulting temporary file containing datetimes has therefore a fixed name. File `timetest.py` containing a function returning the closest run time is missing, because the function is already contained in both `download.py` and `process.py` separately. The example usage is:

```
python3 download.py 2018-11-09T12:34 2018-11-15T17:18
```

- `download.sh`: a small boilerplate bash script.

It looks only 7 days into the past instead of 14 days as this is the usual duration how long GRIB files are stored on the NOAA production FTP and because they are released regularly.

Processing phase consists of launching two following scripts:

- `process.py`: contains all processing logic

It does not have any part checking whether appropriate files are present in download folder and just assumes they are. There can be no such special situation as precipitation file missing or forecast only in waves production data. Additional interpolation never happens because the desired spatial resolution of $0.5^\circ \times 0.5^\circ$ is always met on the data source level. A tool `grib_copy` does not have to be used because all three desired variables can be easily extracted using `grib_get_data` in one run using the `-w "shortName=swh/dirpw/perpw"`

parameter. This would be theoretically possible even for history weather, but the whole query syntax would be very long, cumbersome and fault intolerant, so it was avoided in favour of manually copying single variables to temporary files. Data conversion and value formatting is unnecessary and therefore omitted. The number of dictionaries to be sent to Elasticsearch in a single bulk operation can be increased around five to ten times compared to history weather because the dictionaries sent are five to ten times smaller.

- `elastic.sh`: a small boilerplate script, no difference from history-weather

Recommended setup launches `download.sh` and `elastic.sh` in a succession four times a day after each NOAA run release by adding a following row to `/etc/crontab`

```
40 5,11,17,23 * * * user cd historical-waves && ./download.sh && ./elastic.sh
```

4.4.2 Monthly hindcast

For downloading and specially for processing monthly hindcast data where a single file contains messages for one variable for a whole month a new approach needed to be developed.

- `download_process_past.py` is a single script, which contains both the download and processing part.

It has three mandatory arguments `startdate`, `enddate` in format YYYY-mm and `eshost` – connection to Elasticsearch host for indexing. The example usage for January, February and March 2017:

```
python3 download_process_past.py 2017-01 2017-03 elasticexample.com
```

The source code with comments can be examined in E-attachments. The operations performed by the script are briefly explained further.

After parsing the input datetimes, iterating over each month, it downloads the GRIB2 files, avoiding the files already present in the download folder and then returns a list of datetimes (first day of each successfully downloaded month). Due to the change of data structure which happened in 10/2017 one method how to deal with both formats would be to list the whole file using `ecCodes` tool `grib_get_data`, while displaying additional variables `stepRange` and `date` in each row of output and saving data accordingly. Different method was

in the end chosen. At the start of month import, a check is made directly if there are any entries with `time` equal to middle of month (day, 15-18) and from the knowledge, further approach is inferred accordingly. This poses an advantage of reducing the size of each row output of `grib_get_data`, because additional variables (`stepRange`, `date`) do not have to be outputted.

EcCodes tool `grib_get_data` is used on each of three single variable GRIB file to print their output to subprocess standard output row by row, where a single row contains latitude, longitude, value and additional outputs (here `shortName`). One large dictionary, which will be then indexed has keys in a format `lat_lon_utctimems` with values forming another dictionary containing data variables. Caution must be taken however, the complete dictionary for full one month would probably not fit into system memory, or if it would, all steps would be slowing gradually. Therefore, complete dictionary is sent to Elasticsearch after every processed day of data. Then the dictionary empties and another day can be processed.

Figure 8 Example of a key and values from waves dictionary source author

```
{1.0_6.5_1533135600000:{"swh":1.04,"perpw":12.52,"location":  
{"lon":6.5,"lat":1},"dirpw":205.73,"ts":1533135600000}, ...}
```

4.5 Elasticsearch mapping

Creating an Elasticsearch mapping is a process of defining how a document is stored and indexed, for example which fields it will contain, their expected type, which string fields should be treated as full text, geolocation, numeric or date field specification, custom rules for dynamically added fields and more (Elasticsearch Mapping, 2018). It is not mandatory to create a mapping because ES enables dynamic field mapping where data types are automatically guessed. Nevertheless, the suggested behavior is to define the mapping before data indexing. It is important to say that existing field mappings cannot be updated, and a possible change invalidates the index (except for a few rare exceptions). In an event of discovering that a different mapping is needed for any reason, a new index with the correct mappings needs to be created and data must be reindexed.

Elasticsearch mapping which was created for history waves can be seen in Attachment 3. Due to larger number of variables in history weather, mapping for it was not listed in the appendix and can be seen in E-attachments along with the rest of the source code. There are a few remarks which need to be mentioned about creating the mapping for history weather

and waves. Specifying the `"template": "waves_*` applies the mapping to all indices which have a name starting with `waves_`. To decrease the size of individual indices and improve search speed, each month of data forms its own index, named `"waves_YYYYMM"`. `Settings` option sets the number of shards and replicas. As weather and waves data persistency is not crucial and they can be downloaded and processed again at any time if a failure occurs, number of replicas is set to 0 to reduce storage demands. Sharding option is not important in this case, as the ES cluster is currently using only a single machine and the value was left as default. `Mapping` describes data fields (their name, type and if it should have index or not) along with their metadata (variable description with units in this case). Indexing was enabled only for `location` (`geo_point` data type) and `ts` (time data type). These data fields are then queried in the API.

4.6 API in Node.js

Allowing users to access the Elasticsearch cluster directly would not be wise or rational due to many reasons. Therefore, a small single route access API was built in Node.js. The main reasons justifying this approach were: optimized querying, authentication, access control, input validation and homogenous output. The initial design and implementation of `history-weather-api` was performed by Vesseltracker front-end consultant Ioan Madau. Author of the thesis later made several optimizations and updates, as well as replication of the same design approach for waves history API later.

Figure 9 weather-history-api directory contents source author

```
+---bin
|   www
+---routes
|   index.js
readme.md
.gitignore
app.js
Dockerfile
package.json
elastic-search-client.js
```

In detail description of the API is outside of the scope of this thesis. Emphasis of the description is put on what it does, instead of how it behaves internally. The source code can be seen in E-attachments. As being a Node.js application, the first step before starting the API

CTU in Prague

is ensuring that all needed modules listed in `package.json` are installed – for example using `npm` with a command `npm install` and then started using `npm start`. API answers two different types of requests currently:

- `timeFrom, timeTo` interval query

First intended to be used as a data input for possible weather charts app, showing conditions on a single place on earth from `timeFrom` to `timeTo`. Returns a JSON response with multiple weather data for a given latitude, longitude between `timeFrom` and `timeTo`. A sample request is formatted like:

```
localhost:3000/?lat=1.89&lng=155.27&timeFrom=2018-05-03T03:14:17Z&timeTo=2018-06-18T12:47:47Z
```

- `ts` timestamp query

Returns data for a given combination of latitude, longitude and timestamp as JSON. The response will internally get the data for the interval (`ts - 1.5 hours`, `ts + 1.5 hours`), because `timeFrom` and `timeEnd` are the parameters, which are inserted into the Elasticsearch query. A sample request is formatted like:

```
localhost:3000/?lat=1.89&lng=-155.27&ts=2017-05-02T23:11:47Z
```

Input coordinates are always rounded to the closest 0.5° degree because the spatial resolution is ensured in the data processing phase. All of `ts`, `timeFrom` and `timeTo` must pass ISO_8601 time validation and the coordinates must pass the usual boundary check (-180, 180 for longitude and -90, 90 for latitude) to avoid query errors. The successfully parsed input is then inserted into the Elasticsearch supported JSON query, where location `geo_distance` filter `distance` property is set to 10 metres (if indices contained additional data without guaranteed 0.5° x 0.5° resolution, for example on-site measurements, the distance value would have to be increased to return them). Range `must` query is applied for `ts` variable to be greater than or equal to `timeFrom` and lower than `timeTo`.

The search is not performed on all indices to speed up the query. Only relevant monthly indices are chosen based on the input timestamp, as seen on Figure 10. By using JavaScript `elasticsearch.js` module the query is sent to the Elasticsearch cluster and the response returns to the user without any additional modifications from the API. `Dockerfile` for creating a Docker image of the API was also created. It enables running constantly inside a Docker container and is a recommended way for deploy. It must be ensured that the container running the API has read permission to the Elasticsearch cluster.

Figure 10 Code snippet - function returning string with joined ES index names based on input Moments `timeFrom` and `timeTo` source author

```
function queryIndexPrint(_time1, _time2) {
  var indexes=[];
  var type = "weather_";
  var time1 = _time1.clone().utc();
  var time2 = _time2.clone().utc();
  while (!time2.isBefore(time1)){
    var text = type + time1.format('YYYYMM');
    indexes.push(text);
    time1.add(1, 'months');
  }
  if (time1.month()==time2.month()){
    //catch case when on the edge of month in ts query
    var text = type + time1.format('YYYYMM');
    indexes.push(text);
  }
  var printstring = indexes.join(",");
  return printstring;
}
```

4.7 Forecast API modifications

Forecast weather and waves API with similar functions like the newly created history APIs have already been running at Vesseltracker for over 5 years. This thesis project planned to review the current state and if possible, remove the visible performance issues and errors, improve the functionalities and renew the datasets used, instead of blindly replacing it with a new solution from scratch, since it has been running without major maintenance for a long time. The current setup uses MongoDB as a No-SQL storage, a set of Python 2 scripts for download, process and import, `wgrib2` tool for converting GRIB files to CSV and a similar API, only containing more endpoints and with several output parsing and formatting functions. As the review was performed and the data sources were compared with current development of NOAA dataset production, to save resources and effort, it was decided, that the forecast API framework does not need a replacement, only improvements. Without going into technical detail, the major changes which were made in the scope of the thesis included:

CTU in Prague

- changing wave forecast data source from single grid Wave NWW3 model data with spatial resolution $1.25^\circ \times 1^\circ$ to $0.5^\circ \times 0.5^\circ$ multi grid forecast data, which are described in detail in chapter 4.1
- adding all wave variables to the output (previously only significant wave height)
- increasing spatial resolution of weather data from GFS $0.5^\circ \times 0.5^\circ$ to GFS $0.25^\circ \times 0.25^\circ$ (further information about forecast API used data in chapter 5.1)
- adding ice thickness variable from secondary variables dataset to weather API
- changing time resolution of weather forecast to 1 hour in the first 120 hours, then to 3 hours up to 240 hours
- improving time resolution of waves forecast to 1 hour in the first 120 hours, then to 3 hours up to 180 hours
- fix recent import hangs caused by NOAA strengthening of anti-DDOS attack policies
- adding API endpoints for wave forecast query and for timestamp query for both weather and waves
- modifying the API overall to match data changes mentioned above
- fixing API crash and restart during downtime caused by table renaming in each import
- upgrading MongoDB version to benefit from the increase of geospatial performance over years, for example mentioned in (Zhang & Albertson, 2015)

5 Implementation of overlays

This chapter is focused on describing the second part of the practical task of the thesis - automated framework creating and serving raster overlays for forecast and history weather and waves. As it was created after making the history weather and waves API, previously gained knowledge about NWP model data and their processing could be utilized. The burden of serving overlays was put on the shoulders of GeoServer software. Although neither the author nor any Vesseltracker staff had any previous experience with orchestrating GeoServer, it turned out to be a good decision. The flexibility of used data formats, batch operations using REST API and overall maturity of the GeoServer project outweighed somewhat hard-to-grasp documentation and initial problems with deciphering error log messages. GeoServer users mailing list community is also usually ready to offer clarification and help.

Initial workflow plans expected direct serving of styled GRIB files without any additional preprocessing. GeoServer offers an official plugin for handling GRIB and NetCDF files. Extensive effort was made to visualize them directly but probably due to lack of deeper knowledge of GeoServer rendering transformations and coordinate system operations, author did not manage to visualize any of the GRIB files with longitudes in range (0°, 360°). Vesseltracker Cockpit web application runs on Leaflet and operates in a Web Mercator projection (EPSG:3857) with appropriate bounding box coordinates corresponding to longitudes in range (-180°, 180°). After several attempts, wind arrows, wind barbs and other point symbols generated by GeoServer rendering transformations were created only for longitudes in range (0°, 180°) or (0°, 360°). Raster band color styling worked without problem for all longitudes tested if no rendering transformation was used on them.

Author restrains from any claims that it would not be possible to do server-side point feature rendering transformations for longitude range (-180°, 180°) on GRIB files directly, but he did not manage to make it work properly. Therefore, a workaround in a form of an additional processing phase including converting the GRIB files to GeoTIFF was performed. Using a GDAL software suite, all overlays are being first converted to a multiband GeoTIFF and transformed to a Web Mercator projection (EPSG:3857) to avoid further coordinate reprojections on GeoServer side, thus speed up rendering, as can be often found in recommendations.

5.1 Used data

For serving history weather and waves overlays, data source is identical as for the history-weather and history-waves projects and comes from their respective `download` folders. The exactly same files were used and parsed, even though it does not necessarily mean, that the GeoServer and API projects need to be located on the same machine, as the files can be accessed via network if needed. However, there is no need to download and store the same datasets twice. For their thorough description, please see chapter 4.1. Not all extracted variables for the API, mentioned in chapter 4.2, were used for overlays though. Following weather variables were left out: `crain`, `csnow`, `cicep`, `cfrzr` and `lsm`, mainly because serving them as layers makes no sense. Waves data sources and extracted variables remain the same as for API - both monthly hindcast and hourly forecast.

Wave forecast uses identical data source as wave history. The number of extracted variables is increased because the storage demands are negligible as time resolution spans only 180 hours into the future. The filename and URL of the wave FTP source follow the pattern:

```
ftp://ftpprd.ncep.noaa.gov/pub/data/nccf/com/wave/prod/multi_1.20170101/multi_1.glo_30m.t06z.f003.grib2
```

GFS forecast with spatial resolution of $0.25^\circ \times 0.25^\circ$ was used as a data source for the reviewed forecast API as mentioned in 4.7 and for forecast overlays too. Forecasts of weather and waves are produced by NOAA four times a day (around 4,5 hours after the actual model run for both waves and weather). To speed up the import phase and decrease bandwidth demands, only a selection of weather variables from the GRIB files was downloaded using the NOMADS Grib Filter tool. Ice thickness data for this resolution must be downloaded from secondary parameters category file, because they are not present in the standard file, but their visualization was desired. The download of other variables ended up being divided into two parts, namely variables which will be later converted to another unit and those which will not. Sample download URL for weather forecast data without, with conversion and for separate ice thickness file can be seen below respectively. Variable run times, forecast times and data datetimes are written in bold italics to be clearly distinguished.

```
https://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p25.pl?file=gfs.t06z.pgrb2.0p25.f003&lev_2_m_above_ground=on&lev_entire_atmosphere=on&lev_surface=on&var_ICEC=on&var_RH=on&var_TCDC=on&var_TMP=on&leftlon=0&rightlon=360&toplat=90&bottomlat=-90&dir=%2Fgfs.2017011506
```



```
https://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p25.pl?file=gfs.t06z.pgrb2.0p25.f003&lev_10_m_above_ground=on&lev_surface=on&var_GUST=on&var_PRATE=on&var_PRMSL=on&var_SUNSD=on&var_UGRD=on&var_VGRD=on&leftlon=0&rightlon=360&toplat=90&bottomlat=-90&dir=%2Fgfs.2017011506
```

```
https://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p25.pl?file=gfs.t06z.pgrb2b.0p25b.f003&var_ICETK=on&leftlon=0&rightlon=360&toplat=90&bottomlat=-90&dir=%2Fgfs.2017011506
```

5.2 Download and process

All download and processing tasks are performed by five Python scripts located in a `geoserver-scripts` folder, structure can be seen on Figure 11. Internally, they partly resemble scripts from API project, but usually are more lightweight.

Figure 11 `geoserver-scripts` folder contents source author

```
+---download_waves
+---download_weather
+---logs
+---temp

import_waves_forecast.py
import_waves_history.py
import_waves_grouped.py
import_weather_forecast.py
import_weather_history.py
settings.py
initial_start.py
README.md
.gitignore
```

All source codes from this section with comments can be examined in E-attachments. Operations performed by the scripts and the motivation are briefly explained further. Shared configuration options are stored as variables in `config.py`, which differs for local development version and production server version. Configuration options are then imported in other scripts as needed. Folders `download_waves`, `download_weather` and `temp` are used as intermediate forecast results storage during the import phase. All end products are then moved to appropriate GeoServer image mosaic folders. Contents of the temporary folders are deleted after the end the of the import phase.

5.2.1 Forecast

- `import_weather_forecast.py`: takes care of forecast weather overlay creation

- `import_waves_forecast.py`: takes care of forecast waves overlay creation

Both scripts work very similarly, only weather forecast has additional variable converting steps, which were not needed in case of wave forecast. Scripts do not accept any system arguments. After launch, they determine the current NOAA run time based on current time and start to download data. Due to recent strengthening of NOAA anti-DDOS policies, irregularly some import runs did not complete, because the connection was cut while downloading a random file. An additional check which restarts the download of single timestamp files, if the download operation did not complete in a preset duration was therefore added. For the case of weather, three separate files are downloaded for a single datetime (ice thickness only, then the rest divided into two parts). Not-to-be converted part is merged with ice thickness file using ecCodes tool `grib_copy`. Resulting temporary merged GRIB file is transformed to Web Mercator projection, with fixed spatial resolution 35km x 35 km and internally tiled using `gdalwarp`. Resolution was chosen as a balance between file size and sufficient data accuracy for visualization purposes.

Figure 12 Code snippet for used `gdalwarp` raster operation source author

```
gdalwarp -overwrite -tr 35000 35000 -t_srs EPSG:3857 --config CENTER_LONG  
0 -wo SOURCE_EXTRA=80 -r bilinear -multi -wm 1000 -wo NUM_THREADS=2 -q  
-ot Float32 -dstnodata -9999 -co TILED=YES input.grb2 output.tif
```

Several unit conversions take place in the processing step, where every time a temporary single band .tif file containing a single variable is generated using a tool `gdal_calc.py`. These temporary files are then merged into a single multi-band file with a tool `gdal_merge.py` and transformed to Web Mercator projection the same way as transforming the GRIB files. Both non-converted and converted .tif files are then merged again using `gdal_merge.py`. Currently four variables take part the process of unit conversion. Gust speed from m/s to maritime used knots, pressure from Pa to a more common hPa, precipitation rate from kg/m²s to mm/hour and sunshine duration homogenized from variable step to seconds/one hour. In addition, two new variables are computed – wind speed and wind direction from u and v wind vector components. The reason why the variables are not converted on the fly by GeoServer can be found in chapter 6.3.

Figure 13 Code snippet of wind direction computation (0° = wind comes from north, 180° = wind comes from south) using gdal_calc.py source author

```
gdal_calc.py -A input.tif --A_band=2 -B input.tif --B_band=3
--calc="(180/3.14159)*arctan2(-A,-B)" --NoDataValue=-9999
--outfile=temp.tif
gdal_calc.py -A temp.tif --A_band=1
--calc="(360+A)*(A<0)+(A>=0)*A" --NoDataValue=-9999 --outfile=output.tif
```

At first, overviews allowing additional rendering speed up were added for levels: 2, 4, 8. The visualization comparison on humidity layer with and without overviews can be seen in Attachment 6 and Attachment 7. As the visual side of overview approach rendering was horrible, unsurprisingly, the approach was quickly abandoned. Compression is not used in any form for forecast data because storage capacity gains were negligible on corresponding data volume. It would on the other hand impose increased demands for GeoServer on data loading, which would in addition need to take care of decompression.

All previously mentioned operations are taking place in the respective download folders. Another step involves moving all the processed GeoTIFF to the mosaic folder, overwriting current content. Leftover GRIB files from previous runs older than a day from current run are deleted to avoid folder bloating. Their respective granule (single file of the mosaic) entry in the mosaic index is removed by a dispatched DELETE request to the appropriate coverage store in the REST API. Last operation involves sending POST requests to the REST API to include newly added forecast files to the mosaic index. It means that GeoServer must be running and the script needs to have write access to the coverage store in REST API to correctly index the files to the mosaic.

5.2.2 History

- `import_weather_history`: creates a history weather overlay for a single datetime
- `import_waves_history`: creates a history waves overlay for a single datetime

As both scripts work very similarly, the description will be shared. They accept a single system argument `datetime` in format `YYYY-mm-DDTHH:MM`, the same as `process.py` mentioned in chapter 4.3.2. The weather script first performs a sanity check which files are present in the historical weather download folder, if no precipitation data are present (GRIB1 version or missing precipitation file for GRIB2), the procedure is aborted, and overlay raster is not created. The reason for that is that all visualized data variables are needed for import to secure

CTU in Prague

a fixed number of bands for the resulting GeoTIFF. Following the same procedure of using `grib_copy` as mentioned in chapter 4.3.2, temporary GRIB files are created. First for variables which are to be converted, second for those which stay unconverted. Conversion process using `gdal_calc.py` is the same as mentioned in chapter 5.2.1. After translating the temporary files to Web Mercator projection GeoTIFF with fixed resolution of 35km x 35km and adding a Deflate compression to decrease storage demands, the new granule is created by merging converted and non-converted variables with `gdal_merge.py`, moved from `temp` folder to the mosaic folder and added to the mosaic index by sending a POST request with the file path to the REST API. Overviews are also not being created for history data for obvious reasons – blurriness of the image and increased storage demands.

Wave data from `historical-waves/download` directory for a single timestamp always contain a fixed number of variables available, therefore no additional presence check is needed. Using a `grib_copy` tool, desired variables are extracted to a single temporary GRIB file. The procedure of creating a GeoTIFF using `gdalwarp` and adding a granule to a mosaic stays the same as with history weather overlays, while no conversion step is performed.

- `import_waves_grouped`: creates history waves overlays for each timestamp between start month and end month from hindcast monthly data

After parsing the two system `datetime_start` and `datetime_end` arguments in format YYYY-mm, because of the change of data format from 10/2017, it needs to be determined what type of file it is being dealt with. After figuring that out, the script iterates over the whole month with three-hour step and by using `grib_copy`, it copies all contents for given `stepRange` from all three monthly files (`hs`, `tp`, `dp`) to a temporary GRIB file for a single timestamp and containing all three variables. The rest of the procedure of creating a GeoTIFF using `gdalwarp` and adding a single granule to a mosaic stays the same as with history weather overlays.

Emphasis should be given on the characteristics of recommended setup, that to avoid process duplication, none of history import scripts perform the actual download. GRIB files must at that time already be present in respective download folders of history-weather and history-waves projects. As mentioned in chapters 4.3.2 and 4.4.1, both `import_weather_history.py` and `import_waves_history.py` are being launched always

when new history GRIB files are downloaded and processed to Elasticsearch. Naming convention for result GeoTIFF files created by both wave data processing scripts is identical to enable smooth image mosaic indexing.

5.3 GeoServer configuration

All necessary configurations needed for creation and serving of overlays, which were organized and used to the authors best knowledge are summed in this chapter. It is obligatory to follow the same schema (workspaces, layers) to make scripts from previous chapter work well without changing other files than `config.py`. A folder `geoserver-data`, which can be seen in E-attachments contains all configuration options, which were changed compared to the initial GeoServer install. Its simplified content schema, showing mainly workspace, store and layer dependencies can be seen in Attachment 4. All configurations were first done manually in GeoServer web admin GUI. For migration to a new version of GeoServer or a different machine, `geoserver-data` directory is the main and only content, which should be backed up and then migrated. Batch changes on multiple layers can be done using the REST API if needed.

5.3.1 Workspaces, Stores

Workspace is a container which organizes other items – in GeoServer it often groups similar layers and styles together. Four workspaces were created in total: `waves`, `weather`, `history_waves`, `history_weather`.

A store connects to a data source which contains raster or vector data. A data source can be a file, group of files, a table in a database, a single raster file, a directory and more. The only type of data store used in this project is ImageMosaic – a recommended structure for organizing layers with `time` or `elevation` attribute (ImageMosaic time-series, 2018). It is already bundled into standard GeoServer install, thus no extension needs to be added to provide the functionality on the current stable version (2.14.x). The mosaic operation creates a mosaic from two or more source images in a single folder.

Four coverage stores, one in each workspace, were created to point to the mosaic folders: `waves_forecast_store`, `waves_history_store`, `weather_forecast_store`,

`weather_history_store`. All created GeoTIFF files share the same timestamp naming rules to use the same `timeregex.properties` configuration file in each mosaic folder.

Figure 14 timeregex.properties file to match Python datetime.isoformat() output source author

```
regex=[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}
```

File `indexer.properties` determines which variables will be inserted into the index and was left as default from the manual. Configuration file `datastore.properties` describes the connection to external PostgreSQL database used for storing the mosaic index. There is an option to leave GeoServer to automatically create a shapefile for storing the mosaic index with the first mosaic update. However, this option is generally not suggested for production use, see chapter 6.3.

5.3.2 Layers

In GeoServer a layer forms a raster or vector dataset that represents a collection of features. Raster layers are occasionally called coverages. A layer associated with a workspace in which the layer data source is defined. Layers are then referred to by their workspace name, colon, layer name.

For creating a raster layer from ImageMosaic dataset containing multiband GeoTIFF images, like in this project, the most important task is to correctly select which band number, corresponds to each variable. As GRIB files are messages appended to each other, their respective order as bands in GeoTIFF should stay the same. To determine which band number contains which variable a `gdalinfo -stats` tool was used. Indexing of `gdalinfo` output starts at 1, however indexing of band numbers selection in GeoServer layer coverage creation starts at 0, which needs to be kept in mind while registering a new coverage.

Additionally, `gdalwarp` should retain contained metadata for each band unless `-nomd` (no metadata) option is added to the command. While metadata help to determine what does each GeoTIFF band contain, if GRIB extension would be added to GeoServer, these metadata are read and processed as well. The mosaic is later treated internally as containing only GRIB files by the ImageMosaic plugin, even though files have `.tif` extension and are valid GeoTIFF files. Header contents are prevalent to file extension and format in decision making and an incorrect file accessor (SPI) is then used by GeoTools, which causes the created layer to not be displayed at all. Author could not really decide if it should be treated as ImageMosaic plugin

bug and thus should be reported to GeoServer JIRA error reporting, but it certainly is not expected behavior. A mailing list `geoserver-users` was therefore contacted with the information about the situation to document dubious behavior among users.

All layers were created as multiband Coverage views containing multiple bands together, which can be later styled to provide a single output. Obvious examples being both wind vector components or precipitation rate together with pressure at mean sea level. One of layer creation options, which was modified in layer settings was Adding response cache headers as 600 seconds. Time dimension also needs to be added for each layer with selecting List as a presentation method and smallest domain value as default value. Nearest match option was left switched off, because the interval of data production is regular, not random. It also may be helpful to fill coverage band details with units, name and `NODATA` value for further reference. Each layer was also selected as opaque and a default style was chosen for it. Cached zoom levels were limited in Tile Caching tab and Time parameter had to be added with Regular Expression value `.*` in order to cache it properly in GeoWebCache. Server cache expiration time for tiles was set for each layer as one day for forecast tile layers. All other setting options stayed default.

In total 31 layers were created for raster weather and waves overlays. From these, there were 5 waves forecast, 2 waves history, 13 weather forecast and 11 weather history layers. Their full list can be seen in `geoserver-data` folder contents in Attachment 4.

5.3.3 SLD styles

Styles are used to control the appearance of geospatial data. Styled Layer Descriptor (SLD) is an OGC standard for geospatial styling used by default. Even though it is possible to create advanced SLD styles in specialized software like `sldeditor` (robward-scisys/sldeditor, 2018) or QGIS GeoServer plugin (boundlessgeo/qgis-geoserver-plugin, 2018), as the number of layers created in this project was not that large, the inbuild GeoServer SLD style editor was used instead. As the foundation of most styles, multiple examples from GeoServer manual, together with exploring other online weather and wave model visualization services like Tidemap from a company Tidetech, were used. Visual appearance of all styles can be seen from Attachment 8 to Attachment 21 and their source codes can be seen in E-attachments.

As an example of a simple ColorMap style SLD, CloudCover can be seen on Figure 15.

An example of a more complicated style is WindArrows with RasterAsPointCollection transformation and symbol dimension and rotation modified on the fly based on wind speed and wind direction variables and can be seen in Attachment 5. In total 17 styles were created. The number is lower than the number of layers because history layers share the styles of their forecast counterparts and for example layer `weather:Temperature2m` has the same style as `weather:Temperaturesurface`. On the fly computed and smoothed contours with fixed levels were also used for a few variables, namely wave height, temperature, pressure, humidity and ice thickness. Contours allow easy comprehension of the weather and wave situation without checking up on the legend and confronting the values from it with the map.

Figure 15 CloudCover SLD, part of single RasterSymbolizer example source author

```
<FeatureTypeStyle>
  <Rule>
    <RasterSymbolizer>
      <Opacity>0.5</Opacity>
      <ColorMap>
        <ColorMapEntry color="#fefefe" quantity="0" opacity="0.01" />
        <ColorMapEntry color="#f4f4f4" quantity="20" opacity="0.2" />
        <ColorMapEntry color="#dbdcdd" quantity="40" opacity="0.5" />
        <ColorMapEntry color="#bcbaba" quantity="60" opacity="0.7" />
        <ColorMapEntry color="#bbbcbf" quantity="80" opacity="0.8" />
        <ColorMapEntry color="#8d8f93" quantity="100" opacity="0.8"/>
      </ColorMap>
    </RasterSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

5.3.4 Other

To secure smooth production launch of the visualization framework, some additional GeoServer configurations needed to be changed as opposed to default values or behavior. As the only OGC service, which is utilized for serving the overlays is WMS, all other services have been disabled. WMS configuration options were changed for every workspace. A minor change was limiting supported reference systems to only EPSG:4326 and EPSG:3857 to shorten the GetCapabilities document for each workspace. Originally, all GeoServer provided reference systems were listed, which made the document unnecessarily enormously large. Another change included setting the Default interpolation option from Nearest neighbor to Bilinear and changing the resource consumption limits section, where a maximum rendering time of 20 seconds was added. The time value was chosen with respect to the fact that even though no user is willing to wait for raster image for more than a few seconds and has probably

already given up by then or already zoomed or panned the map a few times, GeoWebCache seeding times can be much higher for smaller zooms.

As both `monitor` and `controlflow` plugins were added into GeoServer, their configuration was changed to match the incoming load and demands. If a request waits in queue for more than a few seconds it's not worth executing, the client has likely given up by then, so the timeout was set to 5 seconds. Because raster data are being served, amount of memory that GeoServer has at disposal matters and was therefore increased from default one quarter of system memory by modifying the `javaOPTS` variable in a configuration shell script `setenv.sh`. Additionally, a property `-Duser.timezone=GMT` should be added to correctly reference timestamps in UTC for both insert and query. The need of changing the master admin password is obvious.

One of many controlled ways how to run the GeoServer in production environment is a Docker container. For this task an already made recipe from a GitHub `thinkWhere/GeoServer-Docker` repository was used and slightly modified (`thinkWhere/GeoServer-Docker`, 2018). Both `setenv.sh` and `geoserver-data` folders should be attached as volumes while running `docker run` command to preserve configuration and raster overlay files even when updating or recreating the Docker container. A suggested method is to use a Docker container for a PostgreSQL instance for mosaic index storage too.

Figure 16 docker run -v options used source author

```
-v $HOME/geoserver-data:/opt/geoserver/data_dir  
-v $HOME/geoserver-data/setenv.sh:/usr/local/tomcat/bin/setenv.sh
```

6 Further remarks

Following chapter aims to point out caveats encountered during the development of both API and overlay framework as well as to offer a discussion about possible alternatives to approaches for certain parts of the workflow that the author has taken.

6.1 Storage design

During the initial historical weather and waves project planning phase, no large discussion about the database system was made. Elasticsearch as an example of a quick No-SQL database was quickly chosen for its scalability, previous good performance and experience. The large initial planned data volume (indices having hundreds of GB per year) only added to it. While relational databases have many features that NoSQL databases do not (robust secondary index support, rich but easily understandable query language, table JOINS), SQL databases are in general difficult to scale (Time-series data, 2017). These advanced options were not planned to be used as the history weather and wave API was meant to be a APPEND and SEARCH only.

An open-source time-series PostgreSQL extension called TimescaleDB developed by a company Timescale is one of aspiring candidates to show that even SQL databases with plugins are highly scalable. It was released into a mature version 1.0.0 on 30.10.2018 (timescaledb, 2018). It is optimized for fast ingest and complex queries. TimescaleDB scales PostgreSQL for time-series data via automatic partitioning across time and space. The primary point of interaction is a so called hypertable, the abstraction of a single continuous table across all space and time intervals, so that it can be queried by SQL (Architecture, 2018). A single deployment can store multiple hypertables, each with different schemas. TimescaleDB could be a suitable replacement for the current Elasticsearch instance, but no performance testing comparing the two options was made.

Even PostgreSQL has recently started to be suitable as a large volume time series data storage as it offers declarative partitioning of data since version 10, which was released in late 2017 (Table partitioning, 2018). With it, there is dedicated syntax to create range and list partitioned tables and their partitions. PostgreSQL 11 removed various limitations that existed in partitioned tables, such as inability to create indexes on the partitioned parent table. PostgreSQL 11 also added hash partitioning. As a possible follow-up of the current work there

could be the in-depth performance comparison of inserts and simple queries between the current Elasticsearch setup and test TimescaleDB instance or partitioned PostgreSQL instance.

During the configuration of used Elasticsearch mappings and initial cluster setup, multiple suggestions regarding tuning for the decreased disk usage and increased search speed were considered. The type that is picked for numeric data can have a significant impact on disk usage. Integers should be stored using an integer type and floating points should either be stored in a scaled float if appropriate or in the smallest type that fits the use-case (Tune for disk usage, 2018). These hints were used mainly for categorical variables. Another important part is to set a custom index only on fields, which will appear to be searched upon.

Documents should be modeled so that search is the cheapest possible on a given hardware. Even though indexing on `time` and `geo_point` variables helps to increase the search speed, another increase was observed when partitioning data into monthly indexes was used instead of having one large index with all historical data. Additionally, searching only in relevant monthly index/indices based on the parsed query datetime on the API side relieves some pressure from Elasticsearch. Queries on date fields that use `now` value are typically not cacheable as the range that changes all the time (Tune for search speed, 2018). Since historical weather and waves API expect very heterogenous time and space queries (no query hotspots), the above mentioned does not pose an issue and Elasticsearch internal query caching will not help very much anyway. One of the case, where caching would probably help a lot would be direct matching of single points of history ship tracks with history weather and wave data. In general, SSD drives are known to perform way better than spinning disks for all on-disk databases, though they are of course more expensive. Therefore, further improvements on search speed can be reached by adding an SSD drive and transfer some parts of the index there, if needed.

Another untested option how to make the API serve weather variables would be to completely avoid the underlying database as a storage and use GeoServer and direct access to raw downloaded file approach. Technically it would be possible to create weather layers as they exist now and using the WFS `GetPropertyValues` query retrieve the value of a feature property or a complex feature property from the data store for a set of features identified using a query expression (WFS reference, 2018). However, there was no real testing made on

the question of feasibility and more importantly response delivery speed under load of such approach. This approach would decrease data duplication significantly.

6.2 Scripts improvements

Setting aside the question of usage of Python as a programming language to manage the download and processing tasks, multiple problems or improvement suggestions will be outlined in this chapter. As work on this project spanned multiple months, author's knowledge of essential Python libraries was very limited at the beginning and work was interrupted a few times by other responsibilities, the overall consistency and similarity of both API and overlay producing script parts is quite low. That could make it potentially harder for someone else to further change and improve parts of it.

Inconsistent usage of libraries and tools providing download of GRIB files can be mentioned as an example. Historical-weather and historical-waves projects use Python `urllib` library for checking the size of file to be downloaded, while the download itself is performed by a `wget` command. The reason for that was originally because `urllib` was hanging in the middle of a download randomly, which was much later discovered probably not be an issue of the implementation but an effect of anti-DDOS measure being gradually introduced and strengthened by a data provider. Overlay creating scripts use `urllib` for downloading of files and Python `Requests` package for sending requests to the GeoServer REST API. As mentioned in the official `urllib` package documentation, The `Requests` package is recommended for a higher-level HTTP client interface and should be homogenously used for all download operations instead.

An alternative way of downloading desired weather GRIB files would be to utilize the fact that in every currently used data source, each GRIB file has its own external index file with the same name and `.inv` or `.idx` file extension. These files usually contain list of all variables, their levels and datetimes along with information on which byte the record starts. As GRIB files are formed of concatenated messages, only those which are of interest can be downloaded using partial HTTP transfer if it is known, which bytes of a file to download (Fast Downloading of GRIB Files, 2006). The HTTP protocol allows random access reading with range header and a program which can utilize it would be for example `curl`. During later parts of the project, it was discovered that an open-source perl script to download parts of GFS files

`get_gfs.pl` was created by a NOAA meteorologist Dr. Wesley Ebisuzaki and other contributors. It could be used instead of NOAA Grib Filter tool in `geoserver-scripts` project, which would pose a friendlier use of NOAA systems resources and should be a preferred way instead of using Grib Filter during full earth coverage download. For local data extracts, Grib Filter would still be the only option (Grib Filter Help, 2018). Optionally, it could be used for extracting history weather variables already in the download part instead of during processing. This would of course decrease storage demands for historical-weather GRIB files more than tenfold. However, since the historical weather project task was set as: download everything and then choose what is needed later and due to fact that a choice of extracted variables was also changing over time, the approach with random access download was not utilized. A simple utility test was done to examine the feasibility of a script anyway and the performance was very good. Surprisingly even though the subset approach should secure fair resource usage for NOAA servers, the connection was randomly broken a few times as anti-DDOS measure anyway, even though a few seconds timeout was added between subsequent file downloads.

Combining bash scripts and Python scripts in the `historical-weather` and `historical-waves` projects adds unnecessary complexity and reduces readability on what is happening. Avoiding the use of additional bash scripts could be achieved by adding other system argument options and enhancing current system argument parsing behavior by packages, which were designed for it, for example widely used and recommended `argparse` package. Bash scripts redirect log messages from standard output to a separate log file, which could be improved by using a `logging` package instead. The increased usage of cron jobs for task scheduling has been recently discouraged in favor of individual Docker containers, which would be another possible further improvement of the project.

6.3 Overlay remarks

As with most open-source software tools in active development, new features and enhancements appear very often. Not being an exception, GeoServer stable version changed from 2.13 to 2.14 during the development of overlay serving project. One of most relevant changes was adding a support of Jiffle, a map algebra scripting language to create and analyze raster images into GeoTools 20 and together with that into GeoServer 2.14.x. Writing of map

algebra operations on the fly can be used as a WPS or as a rendering transformation written in SLD. The approach of not converting variables in the processing phase was successfully tested from speed point of view and readability of SLD point of view. However, during visualization over the boundary of used SRS (over dateline in Web Mercator projection for example), a problem was spotted that the raster was not colorized outside of the boundary. After contacting the mailing list `geoserver-users`, it became clear that rendering transformations generating raster data do not yet benefit from the advanced projection handling and dateline wrapping machinery, which GeoServer contains. Although as with any open-source project, a helpful change of code or feature addition can be performed usually by anyone, the author does not possess any Java knowledge to be able to perform such non-trivial change himself and has therefore at least created an issue report on GeoServer Jira bug reporting system under issue number GEOS-9036. The mentioned condemnation of `Ras:Jiffle` rendering transformation for converting the variables on the fly led to modifying the processing phase of overlay creation and converting variables beforehand using `gdal_calc.py`.

Another issue was spotted and documented on GeoServer Jira bug reporting system under issue number GEOS-9059. On the current setup of using GWC to precache tiles, starting from zoom 4 and higher, the last metatile or tile before dateline lacked colorization, even though the rendering transformations - contours and possibly arrows were rendered correctly. While this issue was not critical, the overall visual appearance was very impaired by that - mainly for layers, which were styled only by raster colorization (cloud, ice cover and other). After testing different methods how to go around the issue, which are documented in the bug report, a surprising workaround was found. It was witnessed, that adding an empty `Ras:Jiffle` rendering transformation handles the raster bounds internally correctly and therefore allows the colorization and fixes the issue – while the issue mentioned in GEOS-9036 is of no concern in this case, therefore the mentioned transformation can be used. As reader can see, the rendering transformations were juggled around until a working solution was found for given data without being necessarily optimal solution.

While enormous number of optimizations for GeoServer are set by default, some extra considerations are to be kept. An important phase of raster overlay serving is determining the right input and output format. Initially the thought was that because GeoServer enables direct

visualization of GRIB files by a plugin, it will be possible to serve them directly just with styling as this saves storage capacity and avoids double storage of the same dataset in different format. As mentioned in chapter 5, the effort was abandoned in favor of converting to GeoTIFF, thus optimal balance of used resolution, compression and internal tiling was then searched. No extensive testing for production use using Java testing suite JMeter was made.

Regarding the output file used as a WMS output for a web mapping application, a choice was based on Paletted Images GeoServer tutorial (Paletted Images, 2018). As transparency option was needed to ensure that the background layer will still be visible under the overlay, the best option in the terms of response speed and size – JPEG had to be left out. Two real options remained to choose from, either standard PNG file or paletted PNG/8. As the latter would be smaller in size due to usage of computed image palette, which is a table of 256 colors to allow for better compression at the expense of longer processing time, it was therefore chosen. The smaller size of paletted images is usually a big gain in both performance and costs, because more data can be served with the same internet connection, and clients will obtain smaller responses faster. There would be a possibility to remove the opacity parameter from SLD styles and serve the images through WMS as JPEG, while opacity would be then chosen on per-layer basis in the receiving web mapping application. That would be much faster overall, however at the expense of having semi-transparent white background for the whole layer even at places with NODATA, which would impair the overall visual performance of some layers significantly and therefore should not be used.

Deciding on tile caching methods was modified during the work on the project. As the forecast layers change quite dynamically - every 6 hours, it was first thought that the weather and waves overlay service would be able to serve requests to users through a non-tiled WMS facilitating only client-side caching. Tile Caching with GeoWebCache GeoServer Training material (Tile Caching with GeoWebCache, 2018) suggests that caching of layers changing quickly or containing more dimensions is generally not a good idea, as the tile revisit rate would be limited and the introduced overhead of writing to disk, keeping track of the age of the files as well as of the size of the cache would be too large.

After the layers were deployed to the Cockpit web map application, it was clear that some sort of server-side caching would be needed, as the performance spikes made it too

heavy for GeoServer to handle separate WMS requests in reasonable response time of up to one second. Even though controlflow plugin and WMS render time limits prevented GeoServer from out of memory error crashes, it started to block new incoming requests under heavy load, because it was still processing running tasks, as instructed by controlflow plugin. When a user starts to pan the map a lot, the above-mentioned solution of sending a single WMS calls was too slow, while cached or on the fly rendered tiles showed better performance.

That meant, GeoWebCache had to be called to action, which made the throughput even without an SSD drive much better overall. Due to the fact, that even though ten days of weather overlays are being served via GeoServer, the Cockpit currently utilizes only current situation through nearest full hour time query. Pre-seeding (creating new cached tiles in advance) only new forecast data six hours into the future was therefore chosen for small zooms. GeoWebCache allows rendering of tiles not one by one, but as a metatile with variable size, which is afterwards cut into separate tiles, which can be then served. This method only helps if enough memory is provided. The advantage of using it can be seen in situations where a label, geometry or point marker lies on a boundary of a tile, which might be truncated or doubled, while this way, the influence of tile stitching is greatly reduced. By default, GeoServer sets a metatile size of 4x4, which should provide a balance between performance, memory usage, and rendering accuracy (Caching defaults, 2018). However, as operational memory is usually in abundance, while disk speed is a limiting factor in case of HDD, it was increased to 12x8 as a new default in this project. With default tile size 256x256 and total metatile size of 3072x2048 pixels it should cover most commonly used monitor resolutions in one or two metatile rendering requests. In fact, using larger metatiling factors in general is one of ways to reduce the time spent by seeding the cache. Smaller inbuild Guava in-memory cache was also enabled for forecast layers. Live traffic in peak hours however also showed that on-demand rendering of metatiles for specific zooms can behave like an effective throughput blocker, due to occasional GeoWebCache blocking behavior. If a single-machine GeoServer instance is used, it is therefore suggested, that low zooms should be pre-cached, while high zooms should be returned on-demand as standard WMS.

In first phases of the project, a shapefile was used for storing a mosaic index, even though it is not generally suggested for production use by the manual. It was discovered that random faults occurred, where one or several single timestamps for a single store were not

CTU in Prague

accessible by the ImageMosaic plugin. The WMS returned an image, which was a styled raster with NODATA everywhere. A thorough check of the shapefile timestamps and boundaries did not show any differences compared to the timestamps, which were rendered correctly. No solution was found other than to use a PostgreSQL Docker container with persistent storage.

Conclusion

Submitted diploma thesis named History and forecast weather and wave data processing and visualization framework presents one of many possible solutions to the task of augmenting current data and visualization services of a ship tracking web map application Cockpit from vesseltracker.com GmbH.

The first part of the thesis is dedicated to introducing the background of numerical weather forecasting, describing characteristics and history of several widely used global weather forecast and wave models with the emphasis on the most prominent - NOAA GFS, NOAA Wavewatch III. and ECMWF IFS. It also gives introduction to commonly used meteorological data formats GRIB, NetCDF and HDF. Theoretical part introduces used technologies, software, data formats and OGC standards, which were at some point utilized in the project.

The second part of the thesis documents the implementation of the project. For fast querying of historical weather and wave data for a user-given combination of latitude, longitude and timestamp a set of Python scripts was created. Selected history weather and wave model data are downloaded, processed using ecCodes GRIB manipulation tools and after unit conversion indexed to Elasticsearch, which serves as the underlying database. A simple Node.js API was built to provide controlled access to the Elasticsearch cluster. Chapter 4 provides detailed information about used data, script functionalities, example usage and Elasticsearch mapping.

Second part of the project was focused on utilizing GeoServer as a platform for serving semi-transparent overlays with history and forecast weather and wave variables. A set of Python scripts was developed to perform downloading and processing, where the last part of the pipeline is creating converted, multi-band GeoTIFF rasters for each timestamp and communicating with GeoServer using the inbuilt REST API interface. Chapter 5 provides detailed information about used data, script functionalities and example usage. Necessary GeoServer configurations for example layer and style creation, startup variables and more are described in detail in chapter 5.3.

Last chapter contains an evaluation of used approaches and description of obstacles encountered during the development of both API and overlays, as well as workarounds found

for some of them. Developed scripts and GeoServer configurations are ready to be used without additional configuration steps and all source codes can be seen in E-attachments.

All proposed project goals were fulfilled and deployed to production before the submission of the thesis. History weather and wave APIs are accessible and currently contain data from start of year 2017 and newer. The initial cold-start response time for a query with an index of mentioned size on an average hardware is a little slower than expected - around one second, but the speed of retrieval increases around ten times for subsequent queries even if they are not that close to previous queries in time and space. Two cronjobs were set-up to regularly download newly released weather and wave model data to the ever-increasing history data index.

GeoServer overlay system was successfully deployed to production and several layers were added to the Cockpit as current weather and wave overlays to test the back-end throughput capabilities. More front-end works will be necessary to enable displaying of user-defined time of forecast or history layers inside the Cockpit, but front-end works were not author's goal. New overlays are scheduled to be regularly created and pre-seeded every six hours when a new forecast run results are published by NOAA, while the ones older than a day are deleted from the folder and mosaic index. Two JIRA bug reports were also filed to document encountered GeoServer errors, as can be seen in chapter 6.3.

Even though modules were deployed to production successfully, there is always space for improvements, mainly in the question of GeoServer optimization, even though a lot of effort was given to decreasing the response time. Although GeoServer documentation is quite extensive, because some parts are updated more frequently than other, it is not that uncommon to encounter invalid or outdated information. Error messages that GeoServer produces are showing in detail, where the error occurred, no matter the depth of the problem, but to determine correctly, why it happened can often be very difficult to a GeoServer novice. Both projects would also benefit from adding global sea current forecasts and history, however current speeds and directions are not included in the datasets, which were utilized in this thesis.

References

- About.* (2018, 12 22). Retrieved from Vesseltracker: https://www.vesseltracker.com/en/static/about_vesseltracker.html
- About Aladin.* (2018, 10 15). Retrieved from ALADIN-OLD: <http://www.umar-cnrm.fr/aladin-old/>
- About OGC.* (2018, 10 30). Retrieved from OGC: <http://www.opengeospatial.org/about>
- Andersson, E., & Thépaut, J.-N. (2008, spring). ECMWF's 4D-Var data assimilation. Retrieved 10 14, 2018, from <https://www.ecmwf.int/sites/default/files/elibrary/2008/17509-ecmwf-4d-var-data-assimilation-system-genesis-and-ten-years-operations.pdf>
- Architecture.* (2018, 10 25). Retrieved from Timescale: <https://docs.timescale.com/v0.12/introduction/architecture>
- Bellis, M. (2017, 4 5). *The History of the Hygrometer*. Retrieved 9 30, 2018, from ThoughtCo.: <https://www.thoughtco.com/history-of-the-hygrometer-1991669>
- boundlessgeo/qgis-geoserver-plugin.* (2018, 11 21). Retrieved from GitHub: <https://github.com/boundlessgeo/qgis-geoserver-plugin>
- Butler, K. (2013, 7 13). *ArcGIS Blog*. Retrieved 11 1, 2018, from Esri: <https://www.esri.com/arcgis-blog/products/product/analytics/displaying-speed-and-direction-symbology-from-u-and-v-vectors/>
- Caching defaults.* (2018, 12 4). Retrieved from GeoServer manual: <https://docs.geoserver.org/latest/en/user/geowebcache/webadmin/defaults.html>
- cfrib .* (2018, 10 21). Retrieved from GitHub: <https://github.com/ecmwf/cfrib>
- cogeoiff/cog-spec.* (2018, 7 6). Retrieved 10 30, 2018, from GitHub: <https://github.com/cogeoiff/cog-spec>
- Common Climate Data Formats: Overview.* (2013, 12 17). Retrieved 10 16, 2018, from NCAR: <https://climatedataguide.ucar.edu/climate-data-tools-and-analysis/common-climate-data-formats-overview>
- Datasets Set II.* (2018, 10 16). Retrieved from ECMWF: <https://www.ecmwf.int/en/forecasts/datasets/set-ii>
- Direction for wave fields.* (2018, 7 26). Retrieved 11 1, 2018, from ECMWF Confluence: <https://confluence.ecmwf.int/pages/viewpage.action?pageId=111155338>
- Divya, M. S., & Goyal, S. K. (2013, 6 1). ElasticSearch - An advanced and quick search technique to handle voluminous data. *COMPUSOFT*, pp. 171-175. Retrieved 10 21, 2018
- Documentation.* (2018, 10 30). Retrieved from GeoServer: <https://docs.geoserver.org/>
- Documentation and support.* (2018, 10 14). Retrieved 10 14, 2018, from ECMWF: <https://www.ecmwf.int/en/forecasts/documentation-and-support>
- ecCodes license.* (2015, 11 19). Retrieved 10 21, 2018, from ECMWF Confluence: <https://confluence.ecmwf.int/display/ECC/License>
- Elasticsearch.* (2018, 10 21). Retrieved from Elastic: <https://www.elastic.co/products/elasticsearch>
- Elasticsearch Mapping.* (2018, 11 13). Retrieved from Elasticsearch: <https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html>
- ENIAC.* (2006, 9 7). Retrieved 9 30, 2018, from Wikimedia commons: <https://commons.wikimedia.org/w/index.php?curid=6557095>
- Fast Downloading of GRIB Files.* (2006, 12 21). Retrieved 12 6, 2018, from NOAA NCEP CPC: http://www.cpc.ncep.noaa.gov/products/wesley/fast_downloading_grib.html

- File Structure and Performance*. (2018, 3 15). Retrieved 10 16, 2018, from UCAR: https://www.unidata.ucar.edu/software/netcdf/docs/file_structure_and_performance.html
- FM 92 GRIB Edition 2*. (2003, 1 1). Retrieved 10 17, 2018, from WMO: https://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/GRIB2_062006.pdf
- GDAL*. (2018, 10 30). Retrieved from GDAL: <https://www.gdal.org/>
- GeoTIFF Specification 1.0*. (2000, 12 28). Retrieved from Maptools: <http://geotiff.maptools.org/spec/contents.html>
- GFS*. (2018, 10 31). Retrieved from NOMADS: <https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/global-forecast-system-gfs>
- Global Forecast System*. (2018, 10 9). Retrieved from National Centers for Environmental Information: <https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/global-forecast-system-gfs>
- Graham, S., Parkinson, C., & Chanine, M. (2002, 2 25). *Weather Forecasting Through the Ages*. Retrieved 9 30, 2018, from Earth observatory NASA: <https://earthobservatory.nasa.gov/Features/WxForecasting/wx2.php>
- Grib Filter Help*. (2018, 12 6). Retrieved from NOAA NOMADS : https://nomads.ncep.noaa.gov/txt_descriptions/grib_filter_doc.shtml
- HDF4 FAQ*. (2017, 6 28). Retrieved 10 17, 2018, from HDF Group: <https://support.hdfgroup.org/products/hdf4/HDF-FAQ.html>
- HDF5 Tutorial*. (2017, 5 2). Retrieved 10 17, 2018, from HDF Group: <https://support.hdfgroup.org/HDF5/Tutor/>
- History HDF Group*. (2016, 10 14). Retrieved 10 17, 2018, from HDF Group: <https://support.hdfgroup.org/about/history.html>
- History of Elasticsearch*. (2018, 10 21). Retrieved from Elastic: <https://www.elastic.co/about/history-of-elasticsearch>
- History of the Unified Model*. (2007, 1 5). Retrieved 10 14, 2018, from British Atmospheric Data Centre: http://artefacts.ceda.ac.uk/badc_datadocs/um/umhist.html
- ICON*. (2018, 10 15). Retrieved from Deutscher Wetterdienst: https://www.dwd.de/EN/research/weatherforecasting/num_modelling/01_num_weather_prediction_modells/icon_description.html
- ICON-EU*. (2018, 10 15). Retrieved from Deutscher Wetterdienst: <https://www.ecmwf.int/sites/default/files/elibrary/2015/13316-new-nwp-forecast-system-dwd-based-icon-icone-eu-and-cosmo-de.pdf>
- ImageMosaic time-series*. (2018, 9 21). Retrieved 11 20, 2018, from GeoServer User Manual: https://docs.geoserver.org/stable/en/user/tutorials/imagemosaic_timeseries/imagemosaic_timeseries.html
- Implementation specification WMS*. (2006, 3 15). Retrieved 10 30, 2018, from OGC WMS: http://portal.opengeospatial.org/files/?artifact_id=14416
- Implementation specification WMTS*. (2010, 4 6). Retrieved from OGC WMTS: http://portal.opengeospatial.org/files/?artifact_id=35326
- Introduction GEM*. (2018, 10 14). Retrieved 10 14, 2018, from Recherche en prevision numerique: http://collaboration.cmc.ec.gc.ca/science/rpn/gef_html_public/INTRODUCTION/gem_intro.html

- Introduction GRIB1-GRIB2*. (2003, 6). Retrieved 10 17, 2018, from WMO: https://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/Introduction_GRIB1-GRIB2.pdf
- Introduction NetCDF*. (2018, 3 15). Retrieved 10 16, 2018, from NetCDF: <https://www.unidata.ucar.edu/software/netcdf/docs/index.html>
- Kalnay, E. (2003). *Atmospheric modeling, data assimilation and predictability*. Cambridge: Cambridge University Press. Retrieved 10 1, 2018, from <http://catdir.loc.gov/catdir/samples/cam033/2001052687.pdf>
- Kravets, D. (2015, 1 5). *National Weather Service will boost its supercomputing capacity tenfold*. Retrieved 10 14, 2018, from Ars Technica: <https://arstechnica.com/science/2015/01/national-weather-service-will-boost-its-supercomputing-capacity-tenfold/>
- Licences available*. (2018, 10 14). Retrieved from ECMWF: <https://www.ecmwf.int/en/forecasts/accessing-forecasts/licences-available>
- Manousos, P. (2006, 7 19). *Ensemble Prediction Systems*. Retrieved 10 9, 2018, from Hydrometeorological Prediction Center: <https://www.wpc.ncep.noaa.gov/ensembletraining/>
- Meteorology definition*. (2018, 9 30). Retrieved from Oxford dictionaries: <https://en.oxforddictionaries.com/definition/meteorology>
- NAM*. (2018, 10 14). Retrieved from NCEP: <http://www.emc.ncep.noaa.gov/index.php?branch=NAM>
- Naval Oceanography Portal*. (2018, 10 14). Retrieved from FNMOC: <https://www.usno.navy.mil/FNMOC>
- News&Features*. (2018, 1 9). Retrieved 10 14, 2018, from NOAA: <https://www.noaa.gov/media-release/noaa-kicks-off-2018-with-massive-supercomputer-upgrade>
- NOMADS*. (2011, 2 23). Retrieved 10 14, 2018, from CMC ensemble dataset: http://nomads.ncep.noaa.gov/txt_descriptions/CMCENS_doc.shtml
- Novoseltseva, E. (2017, 5 16). *Elasticsearch: advantages, case studies and books*. Retrieved 10 21, 2018, from Apiumhub: <https://apiumhub.com/tech-blog-barcelona/elasticsearch-advantages-books/>
- Numerical Weather Prediction models*. (2017, 7 21). Retrieved 10 14, 2018, from Met Office: <https://www.metoffice.gov.uk/research/modelling-systems/unified-model/weather-forecasting>
- Ocean wave models*. (2018, 10 16). Retrieved from DWD: https://www.dwd.de/DE/leistungen/opendata/help/modelle/legend_ICON_wave_EN_pdf.pdf?__blob=publicationFile&v=3
- Other Forecasting Methods*. (2018, 11 15). Retrieved from University of Illinois: [http://ww2010.atmos.uiuc.edu/\(Gh\)/guides/mtr/fcst/mth/oth.xml](http://ww2010.atmos.uiuc.edu/(Gh)/guides/mtr/fcst/mth/oth.xml)
- Our Mission*. (2018, 10 17). Retrieved from HDF Group: <https://www.hdfgroup.org/about-us/our-mission/>
- Paletted Images*. (2018, 12 4). Retrieved 12 8, 2018, from GeoServer Manual: <https://docs.geoserver.org/latest/en/user/tutorials/palettedimage/palettedimage.html>
- Production Hindcast Archive*. (2018, 8 9). Retrieved 10 31, 2018, from NCEP WAVEWATCH III: ftp://polar.ncep.noaa.gov/pub/history/waves/multi_1/00README
- PyTables*. (2018, 10 17). Retrieved from PyTables: <https://www.pytables.org/>

- Ritchie, H., & Beaudouin, C. (1994, 10 1). Approximations and Sensitivity Experiments with a Baroclinic Semi-Lagrangian Spectral Model. *Monthly Weather Review*, p. 2395. Retrieved 10 14, 2018, from <https://journals.ametsoc.org/doi/pdf/10.1175/1520-0493%281994%29122%2C2391%3AAASEWA%2E0.CO%3B2>
- Robinson, I. S. (2010). *Discovering the Ocean from Space: The unique applications of satellite Oceanography*. Heidelberg: Springer.
- robward-scisys/sldeitor. (2018, 11 21). Retrieved from GitHub: <https://github.com/robward-scisys/sldeitor>
- Ryan, R. T. (1982). The weather is changing... or meteorologists and broadcasters, the twain meet. *Bulletin of the American Meteorological Society*, vol. 63, No. 3 (March 1982), 308.
- Sazid Mahammad, S., & Ramakrishnan, R. (2009, 9 1). *GeoTIFF*. Retrieved 10 31, 2018, from Geospatialworld: <https://www.geospatialworld.net/article/geotiff-a-standard-image-file-format-for-gis-applications/>
- Service implementation. (2018, 8 16). Retrieved 10 30, 2018, from OGC WCS: <http://docs.opengeospatial.org/is/17-089r1/17-089r1.html>
- Schuman, F. G. (1989, 9 1). History of Numerical Weather Prediction at the National Meteorological Center. *Weather and Forecasting*, pp. 286–296.
- Table partitioning. (2018, 10 24). Retrieved 10 25, 2018, from PostgreSQL: https://wiki.postgresql.org/wiki/Table_partitioning
- thinkWhere/GeoServer-Docker. (2018, 12 6). Retrieved from GitHub: <https://github.com/thinkWhere/GeoServer-Docker>
- Tile Caching with GeoWebCache. (2018, 12 13). Retrieved from GeoServer Training manual: <https://geoserver.geo-solutions.it/edu/en/enterprise/gwc.html>
- timescaledb. (2018, 12 6). Retrieved from GitHub: <https://github.com/timescale/timescaledb/releases/tag/1.0.0>
- Time-series data. (2017, 4 20). Retrieved 10 25, 2018, from Timescale: <https://blog.timescale.com/time-series-data-why-and-how-to-use-a-relational-database-instead-of-nosql-d0cd6975e87c>
- Tune for disk usage. (2018, 12 7). Retrieved from Elasticsearch Reference: <https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-disk-usage.html>
- Tune for search speed. (2018, 12 7). Retrieved from Elasticsearch Reference: <https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-search-speed.html>
- Tutorial HDF5. (2017, 5 2). Retrieved 10 17, 2018, from HDF Group: <https://support.hdfgroup.org/HDF5/Tutor/>
- U.S. Government Works. (2018, 10 3). Retrieved 10 14, 2018, from USA.gov: <https://www.usa.gov/government-works>
- Unified Model. (2018, 2 13). Retrieved 10 14, 2018, from Met Office: <https://www.metoffice.gov.uk/research/modelling-systems/unified-model>
- Wavewatch III. (2016, 11 3). Retrieved 10 15, 2018, from NCEP: <http://polar.ncep.noaa.gov/waves/wavewatch/>
- Wavewatch manual 5.16. (2016, 10). Retrieved 10 15, 2018, from WAVEWATCH: <http://polar.ncep.noaa.gov/waves/wavewatch/manual.v5.16.pdf>

- Weather modelling.* (2018, 10 15). Retrieved from Meteoblue:
<http://content.meteoblue.com/hu/service-specifications/data-sources/weather-modelling>
- WFS reference.* (2018, 12 4). Retrieved 12 7, 2018, from GeoServer User Manual:
<https://docs.geoserver.org/latest/en/user/services/wfs/reference.html>
- wgrib2.* (2016, 12 29). Retrieved 10 25, 2018, from CPC NCEP:
<http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/>
- What is ecCodes.* (2016, 10 18). Retrieved 10 21, 2018, from ECMWF Confluence:
<https://confluence.ecmwf.int/display/ECC/What+is+ecCodes>
- WRF Introduction.* (2007, 8 14). Retrieved 10 9, 2018, from WRF Variational Data Assimilation System:
<https://web.archive.org/web/20070814044336/http://www.mmm.ucar.edu/wrf/WG4/wrfvar/wrfvar-tutorial.htm>
- Yongkang, X., & Fennessey, M. J. (1996, 3 20). Impact of vegetation properties on U. S. summer weather prediction. *Journal of Geophysical Research*, p. 7419. Retrieved 10 1, 2018, from
<https://web.archive.org/web/20100710080304/http://www.geog.ucla.edu/~yxue/pdf/1996jgr.pdf>
- Zhang, B., & Albertson, K. (2015, 10 9). *Geospatial Performance Improvements in MongoDB 3.2.* Retrieved from MongoDB Blog:
<https://www.mongodb.com/blog/post/geospatial-performance-improvements-in-mongodb-3-2>

Content of figures

Figure 1 ENIAC - computer used to create the first numerical weather forecasts.....	13
Figure 2 Extracted variables for history weather	33
Figure 3 Extracted variables for history waves	34
Figure 4 historical-weather folder contents	35
Figure 5 Code snippet - finding the closest 3-hour spot to the input time.....	36
Figure 6 Code snippet - extracts depending on type of GRIB file and available files, function arguments shortName, typeOfLevel and levelType are keys of messages	38
Figure 7 Example of a key and values from weather dictionary	38
Figure 8 Example of a key and values from waves dictionary	42
Figure 9 weather-history-api directory contents	43
Figure 10 Code snippet - function returning string with joined ES index names based on input Moments timeFrom and timeTo	45
Figure 11 geoserver-scripts folder contents	49
Figure 12 Code snippet for used gdalwarp raster operation	50
Figure 13 Code snippet of wind direction computation (0° = wind comes from north, 180° = wind comes from south) using gdal_calc.py	51
Figure 14 timeregex.properties file to match Python datetime.isoformat() output.....	54
Figure 15 CloudCover SLD, part of single RasterSymbolizer example	56
Figure 16 docker run -v options used.....	57

List of abbreviations

API	Application programming interface
ECMWF	European Centre for Medium-Range Weather Forecasts
GDAL	Geospatial Data Abstraction Library
GFS	Global Forecast System
GRIB	Gridded Binary
ES	Elasticsearch
HDF	Hierarchical Data Format
HRES	Highest resolution configuration of IFS
HRES-WAM	High Resolution Wave Model of ECMWF
HRES-SAW	High Resolution Stand Alone Wave Model of ECMWF
ICON	Icosahedral Nonhydrostatic Model
IFS	Integrated Forecasting System
JSON	JavaScript object notation
NCEP	National Centers for Environmental Prediction
NetCDF	Network Common Data Form
NOAA	National Oceanic and Atmospheric Administration
NOMADS	National Operational Model Archive and Distribution System
NWP	Numerical weather prediction
OGC	Open Geospatial Consortium
PNG	Portable Network Graphics
REST	Representational State Transfer
SLD	Styled Layer Descriptor
WCG	Web coverage service
WMO	World Meteorological Organization
WMS	Web map service
WMTS	Web map tiling service
XML	Extensible markup language

E-attachments

E-attachments consist of following folders and contains all source codes created during the project:

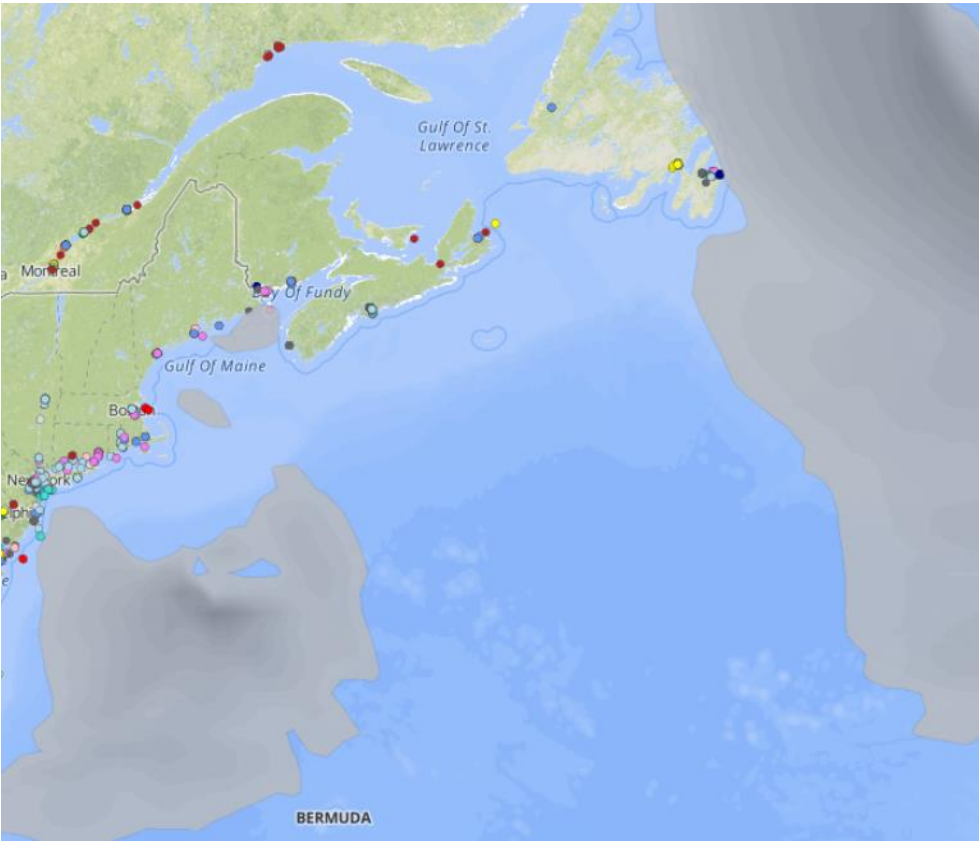
- `historical-weather` with scripts and pre-compiled ecCodes tools
- `historical-waves` with scripts and pre-compiled ecCodes tools
- `weather-history-api` with Node.js API
- `history-waves-api` with Node.js API
- `geoserver-data` with GeoServer configuration settings
- `geoserver-scripts` with all scripts for GeoServer raster overlay creation

Appendix

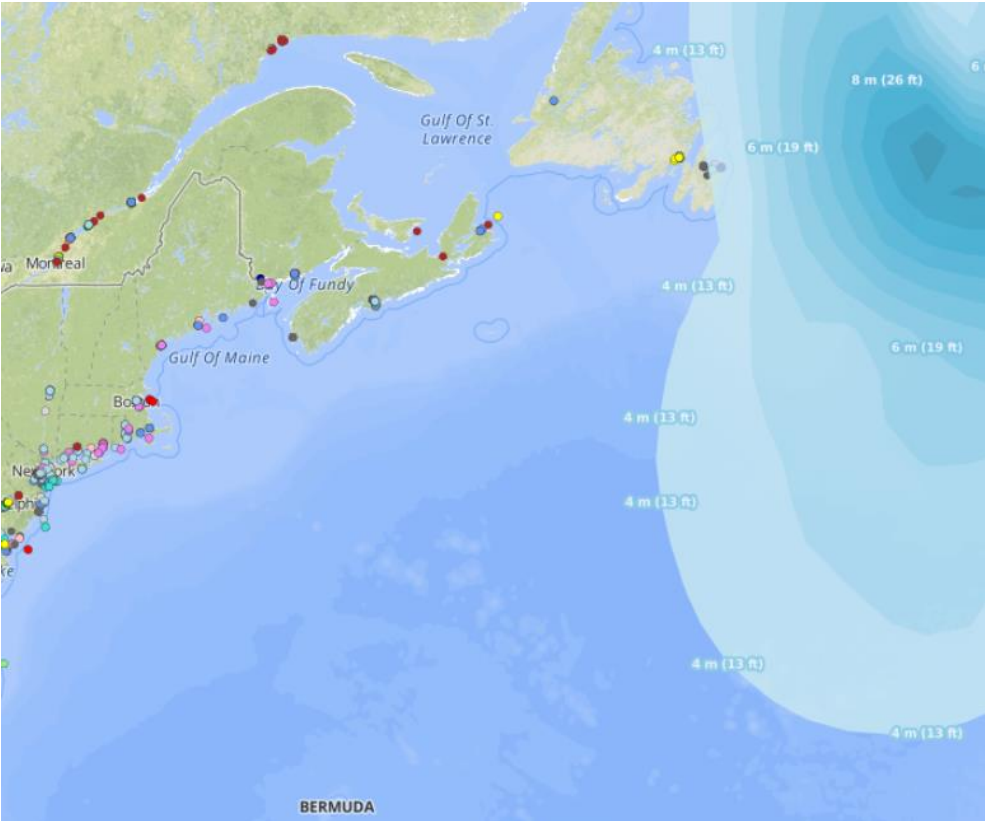
Attachment 1 Old vesseltracker.com current wind tiles	77
Attachment 2 Old vesseltracker.com current wave tiles	77
Attachment 3 Used history waves Elasticsearch mapping waves_mapping.json	78
Attachment 4 Folder geoserver-data showing workspace, store, layer structure	79
Attachment 5 Part of WindArrows SLD, RasterAsPointCollection transformation	80
Attachment 6 WMS of humidity raster with edgy contours (overview approach)	81
Attachment 7 WMS of humidity raster with smooth contours (without overviews)	81
Attachment 8 Wave period style	82
Attachment 9 Significant wave height style	82
Attachment 10 Wind wave height style	83
Attachment 11 Swell wave height style	83
Attachment 12 Wind arrows style	84
Attachment 13 Wind barbs style	84
Attachment 14 Cloud style	85
Attachment 15 Humidity percentage style	85
Attachment 16 Ice thickness style	86
Attachment 17 Ice cover fraction style	86
Attachment 18 Precipitation rate + pressure style	87
Attachment 19 Pressure style	87
Attachment 20 Temperature style	88
Attachment 21 Sunshine style	88

CTU in Prague

Attachment 1 Old vesseltracker.com current wind tiles



Attachment 2 Old vesseltracker.com current wave tiles



Attachment 3 Used history waves Elasticsearch mapping waves_mapping.json

```

{
  "order":0,
  "template":"waves_*",
  "settings":{
    "index":{
      "number_of_shards":"5",
      "number_of_replicas":"0"
    }
  },
  "mappings":{
    "waves":{
      "_meta":{
        "fields":{
          "swh":{
            "description":"Significant wave height"
          },
          "dirpw":{
            "description":"Direction towards which primary wave is coming
in deg"
          },
          "perpw":{
            "description":"Period of primary wave in s"
          }
        }
      },
      "properties":{
        "swh":{
          "index":"no",
          "type":"float "
        },
        "perpw":{
          "index":"no",
          "type":"float "
        },
        "dirpw":{
          "index":"no",
          "type":"float"
        },
        "location":{
          "type":"geo_point"
        },
        "ts":{
          "type":"date"
        }
      }
    }
  },
  "aliases":{
  }
}

```

Attachment 4 Folder geoserver-data showing workspace, store, layer structure

```

+---waves_forecast_mosaic
+---waves_history_mosaic
+---weather_forecast_mosaic
+---weather_history_mosaic
\---workspaces
  +---history_waves
  |   \---waves_history_store
  |       +---History_Wave_primary_height_direction
  |       \---History_Wave_primary_period_direction
  +---history_weather
  |   \---weather_history_store
  |       +---History_Cloud
  |       +---History_Gustarrows
  |       +---History_Gustbarbs
  |       +---History_Humidity
  |       +---History_Icecover
  |       +---History_Prtepressure
  |       +---History_Pressure
  |       +---History_Temperature2m
  |       +---History_Temperaturesurface
  |       +---History_Windarrows
  |       \---History_Windbarbs
  +---waves
  |   +---styles
  |   \---waves_forecast_store
  |       +---Swell_height_direction
  |       +---Swell_height_direction_2
  |       +---Wave_primary_height_direction
  |       +---Wave_primary_period_direction
  |       \---Windwave_height_direction
  \---weather
  |   +---styles
  |   \---weather_forecast_store
  |       +---Cloud
  |       +---Gustarrows
  |       +---Gustbarbs
  |       +---Humidity
  |       +---Icecover
  |       +---Icethickness
  |       +---Prtepressure
  |       +---Pressure
  |       +---Sunshine
  |       +---Temperature2m
  |       +---Temperaturesurface
  |       +---Windarrows
  |       \---Windbarbs

```

Attachment 5 Part of WindArrows SLD, RasterAsPointCollection transformation

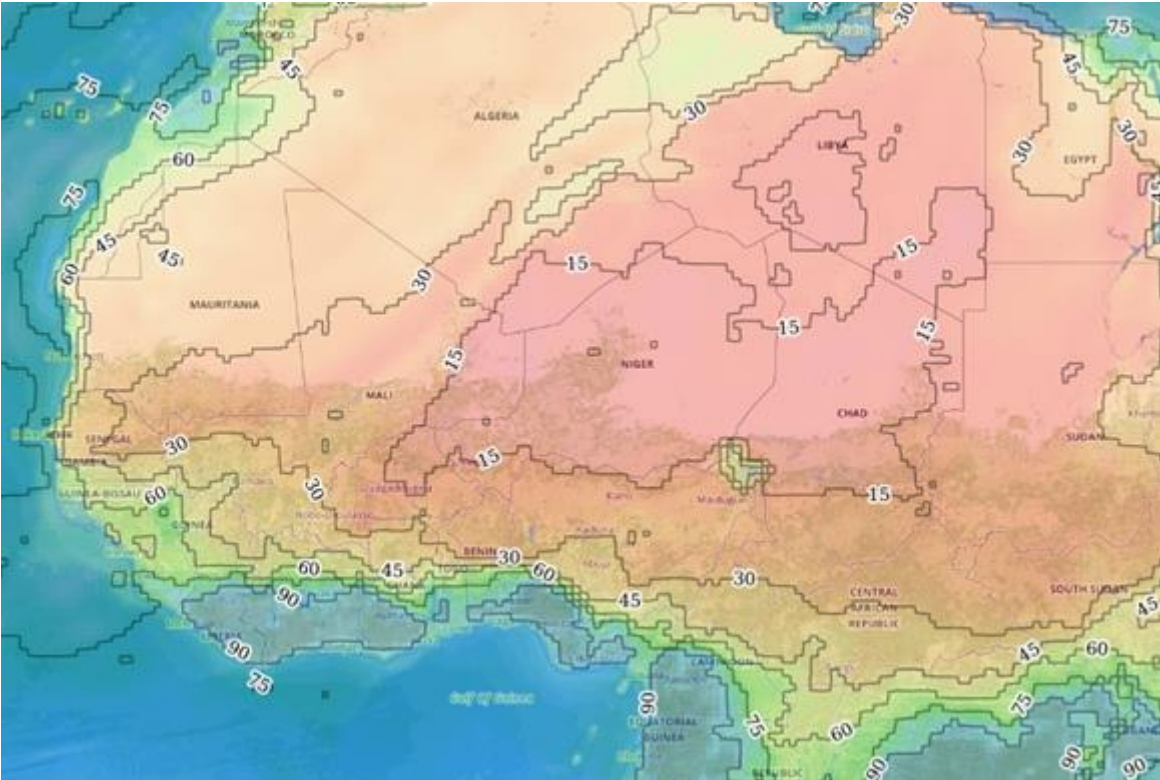
```

<FeatureTypeStyle>
  <Transformation>
    <ogc:Function name="ras:RasterAsPointCollection">
      <ogc:Function name="parameter">
        <ogc:Literal>data</ogc:Literal>
      </ogc:Function>
      <ogc:Function name="parameter">
        <ogc:Literal>interpolation</ogc:Literal>
        <ogc:Literal>InterpolationBilinear</ogc:Literal>
      </ogc:Function>
      <ogc:Function name="parameter">
        <ogc:Literal>scale</ogc:Literal>
        <ogc:Function name="Categorize">
          <ogc:Function name="env">
            <ogc:Literal>wms_scale_denominator</ogc:Literal>
          </ogc:Function>
          <!-- First is scale factor, second wms scale denominator -->
          <ogc:Literal>2.7</ogc:Literal><ogc:Literal>500000</ogc:Literal>
          <ogc:Literal>1.4</ogc:Literal><ogc:Literal>1000000</ogc:Literal>
          <ogc:Literal>0.7</ogc:Literal><ogc:Literal>2500000</ogc:Literal>
          <ogc:Literal>0.4</ogc:Literal><ogc:Literal>5000000</ogc:Literal>
          <ogc:Literal>0.22</ogc:Literal><ogc:Literal>10000000</ogc:Literal>
          <ogc:Literal>0.13</ogc:Literal><ogc:Literal>30000000</ogc:Literal>
          <ogc:Literal>0.07</ogc:Literal><ogc:Literal>60000000</ogc:Literal>
          <ogc:Literal>0.035</ogc:Literal><ogc:Literal>100000000</ogc:Literal>
          <ogc:Literal>0.02</ogc:Literal>
        </ogc:Function>
      </ogc:Function>
    </ogc:Function>
  </Transformation>
  <Rule>
    <PointSymbolizer>
      <Graphic>
        <Mark>
          <WellKnownName>extshape://narrow</WellKnownName>
          <Fill>
            <CssParameter name="fill"><ogc:Literal>#000000</ogc:Literal></CssParameter>
            <CssParameter name="fill-opacity">0.6</CssParameter>
          </Fill>
        </Mark>
        <Size>
          <ogc:Function name="Categorize">
            <ogc:PropertyName>wspd</ogc:PropertyName>
            <!-- First is what (size), second is threshold (speed in knots) -->
            <ogc:Literal>4</ogc:Literal><ogc:Literal>4</ogc:Literal>
            <ogc:Literal>8</ogc:Literal><ogc:Literal>8</ogc:Literal>
            <ogc:Literal>12</ogc:Literal><ogc:Literal>12</ogc:Literal>
            <ogc:Literal>16</ogc:Literal><ogc:Literal>17</ogc:Literal>
            <ogc:Literal>20</ogc:Literal><ogc:Literal>23</ogc:Literal>
            <ogc:Literal>24</ogc:Literal><ogc:Literal>29</ogc:Literal>
            <ogc:Literal>28</ogc:Literal><ogc:Literal>36</ogc:Literal>
            <ogc:Literal>32</ogc:Literal><ogc:Literal>42</ogc:Literal>
            <ogc:Literal>36</ogc:Literal><ogc:Literal>48</ogc:Literal>
            <ogc:Literal>42</ogc:Literal><ogc:Literal>54</ogc:Literal>
            <ogc:Literal>46</ogc:Literal>
          </ogc:Function>
        </Size>
        <Rotation>
          <ogc:Add>
            <ogc:PropertyName>wdir</ogc:PropertyName>
            <ogc:Literal>180</ogc:Literal>
          </ogc:Add>
        </Rotation>
      </Graphic>
    </PointSymbolizer>
  </Rule>
</FeatureTypeStyle>

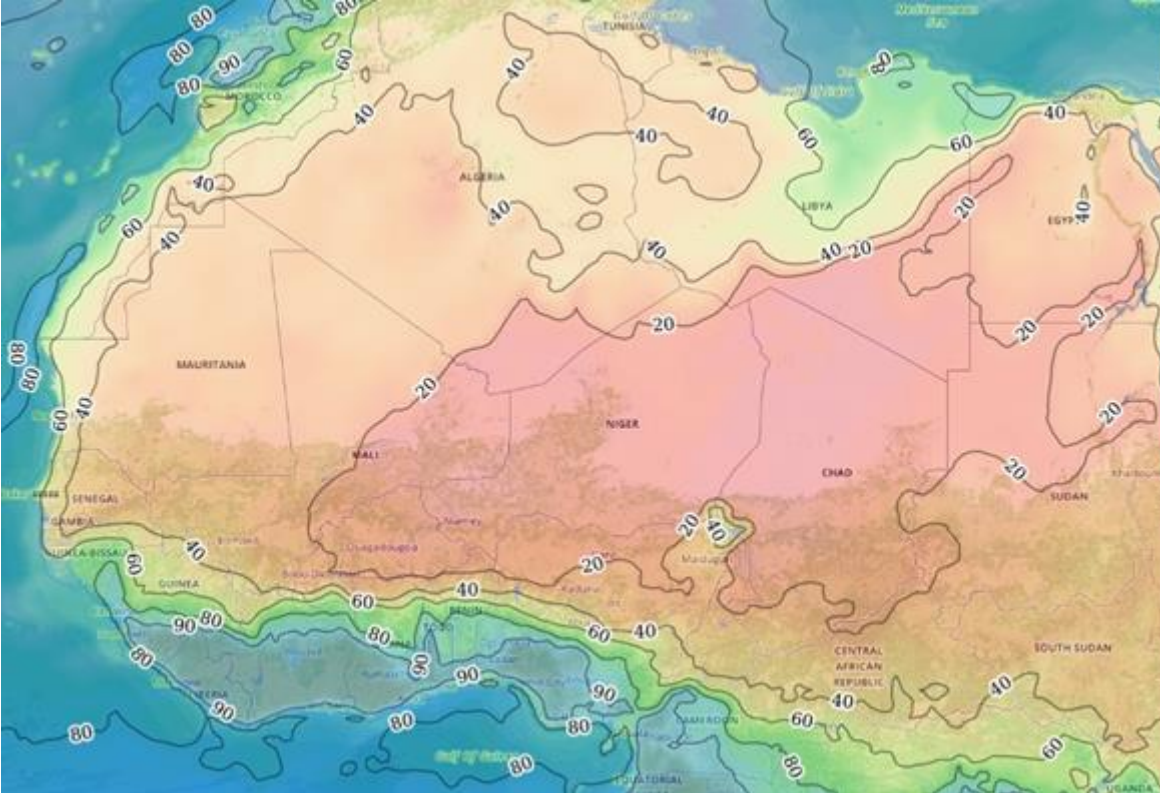
```


CTU in Prague

Attachment 6 WMS of humidity raster with edgy contours (overview approach)

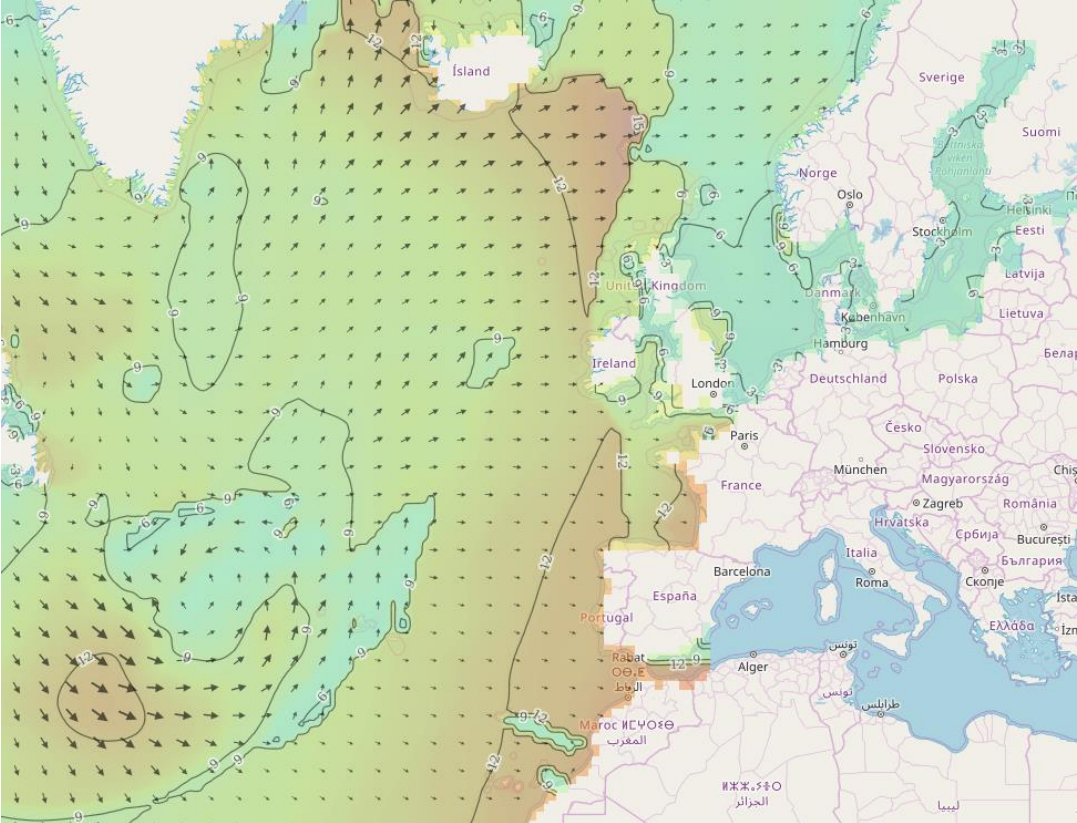


Attachment 7 WMS of humidity raster with smooth contours (without overviews)

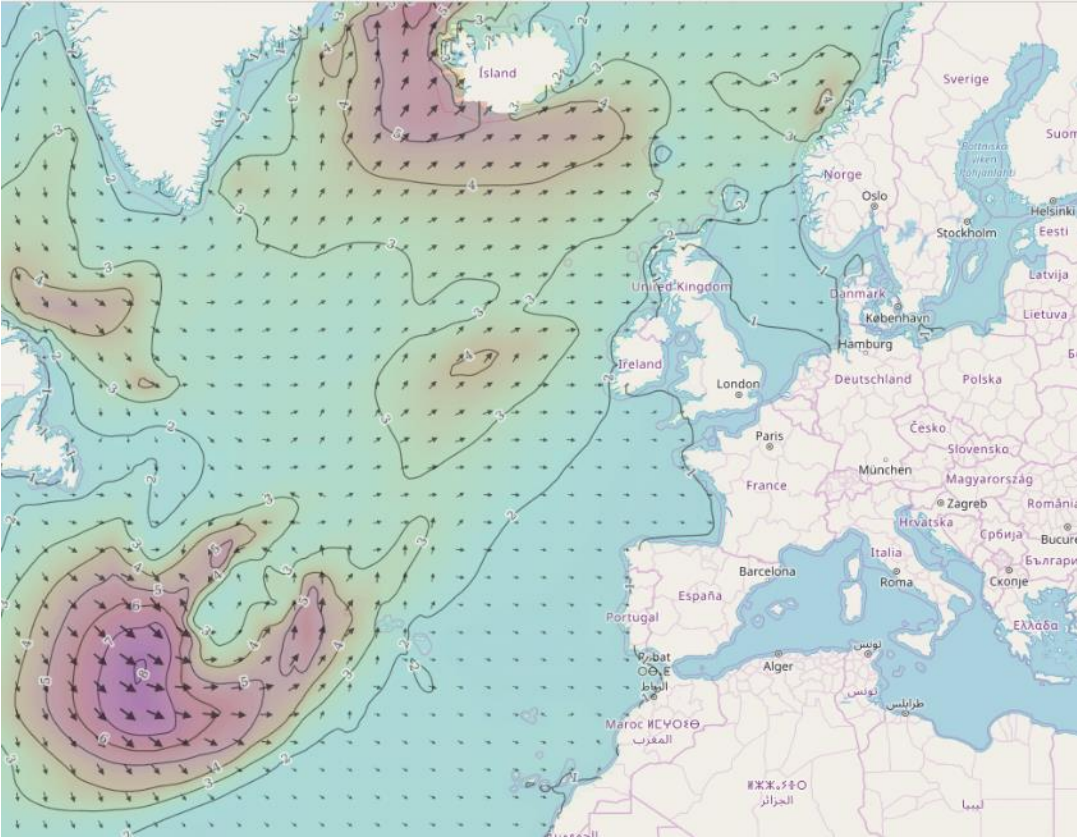


CTU in Prague

Attachment 8 Wave period style

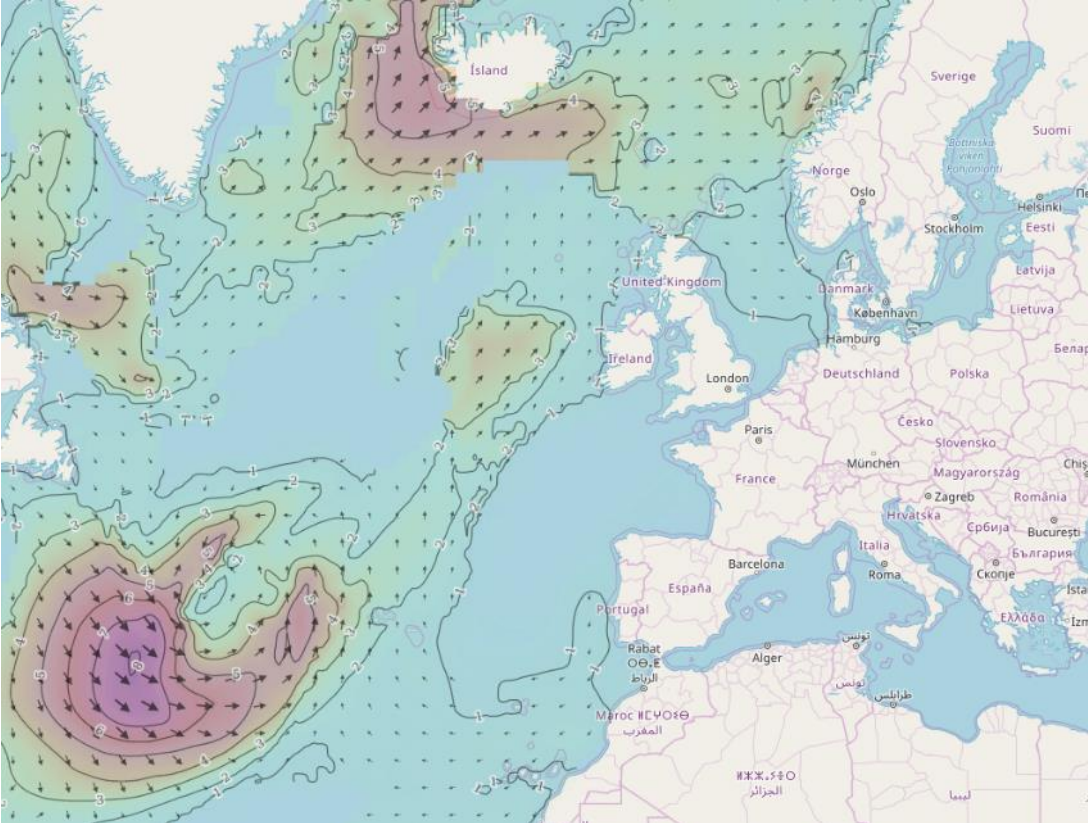


Attachment 9 Significant wave height style

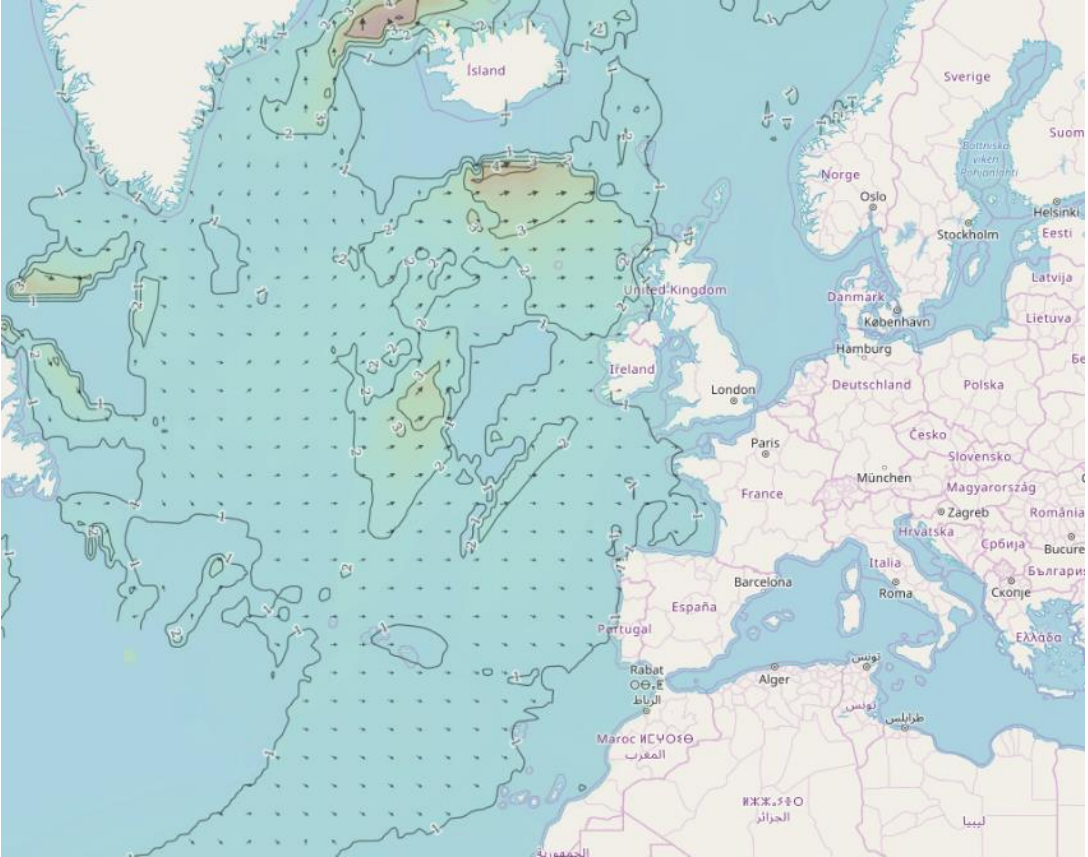


CTU in Prague

Attachment 10 Wind wave height style

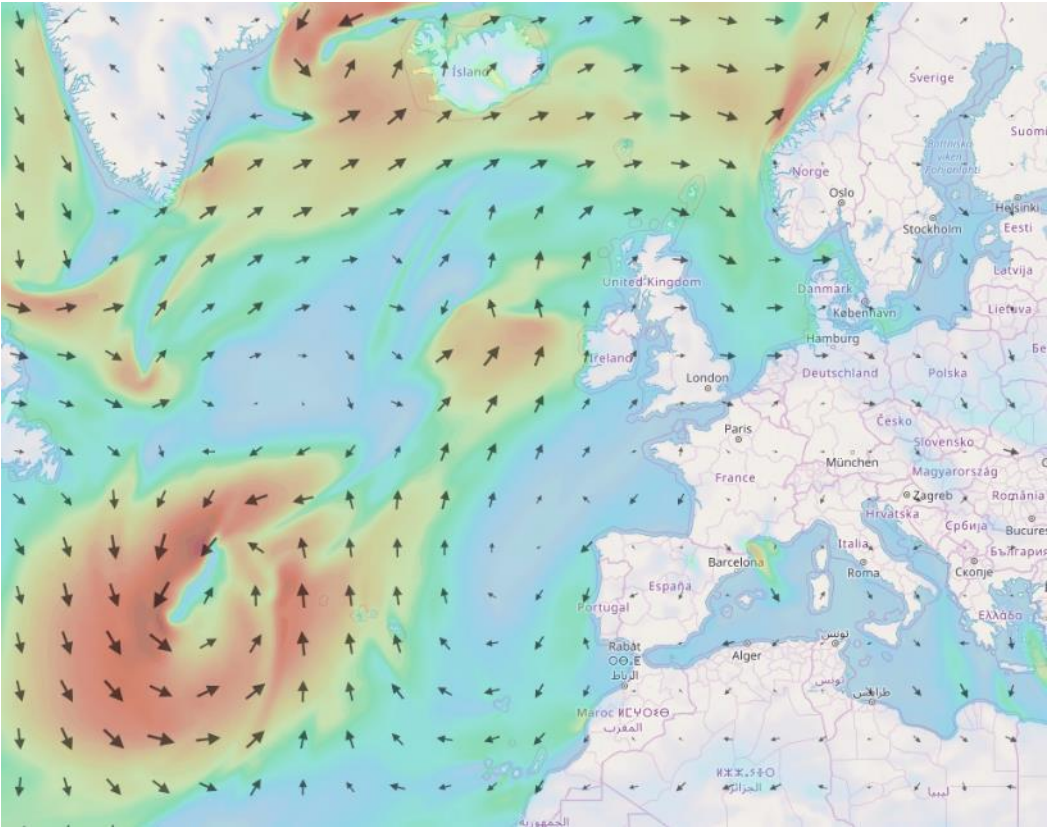


Attachment 11 Swell wave height style

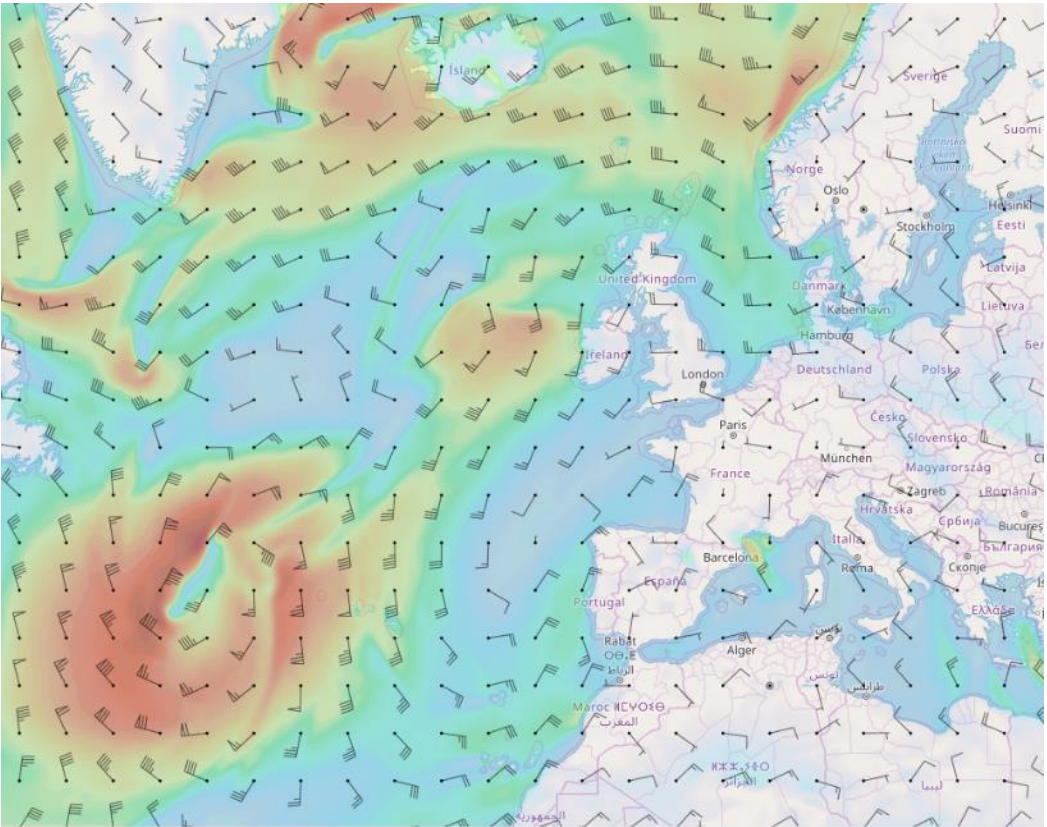


CTU in Prague

Attachment 12 Wind arrows style

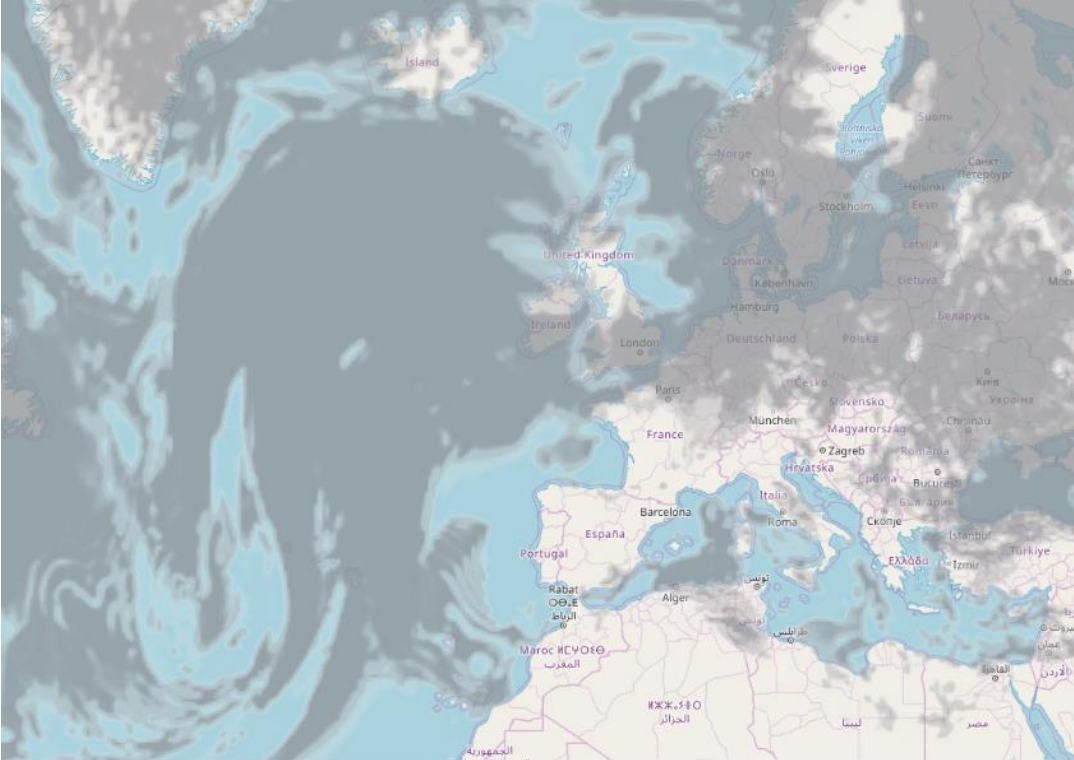


Attachment 13 Wind barbs style

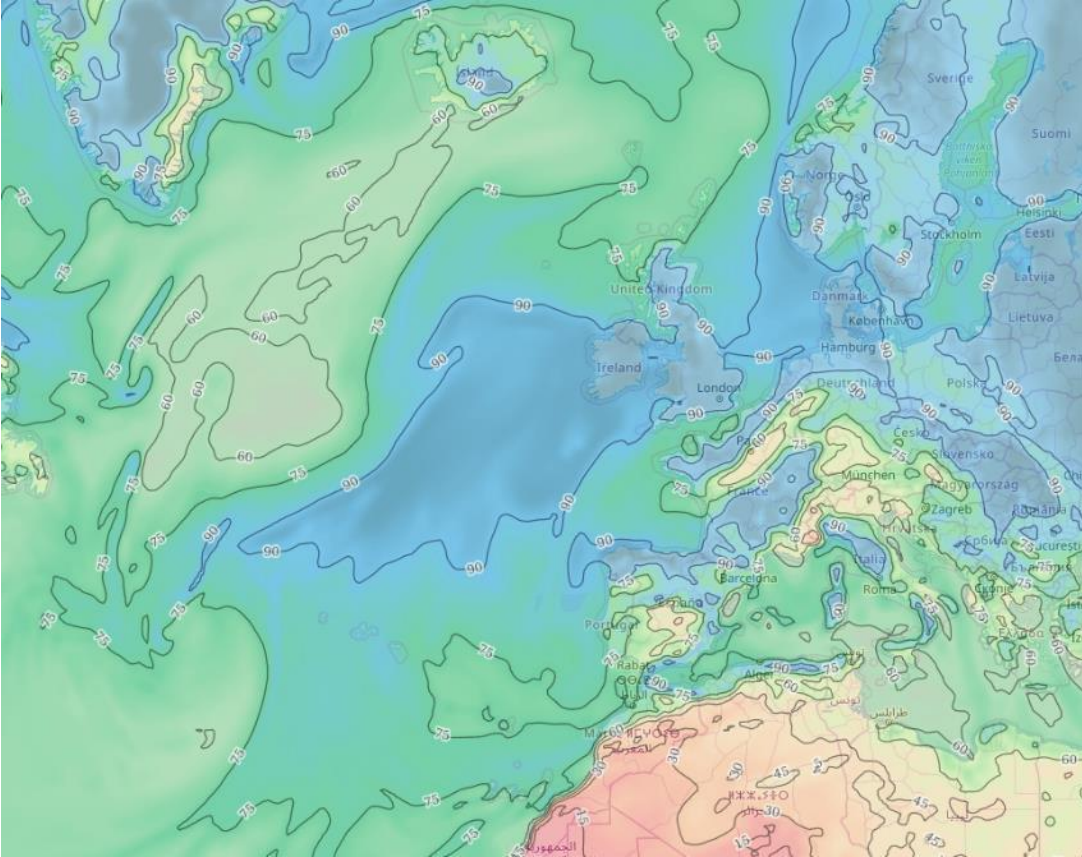


CTU in Prague

Attachment 14 Cloud style

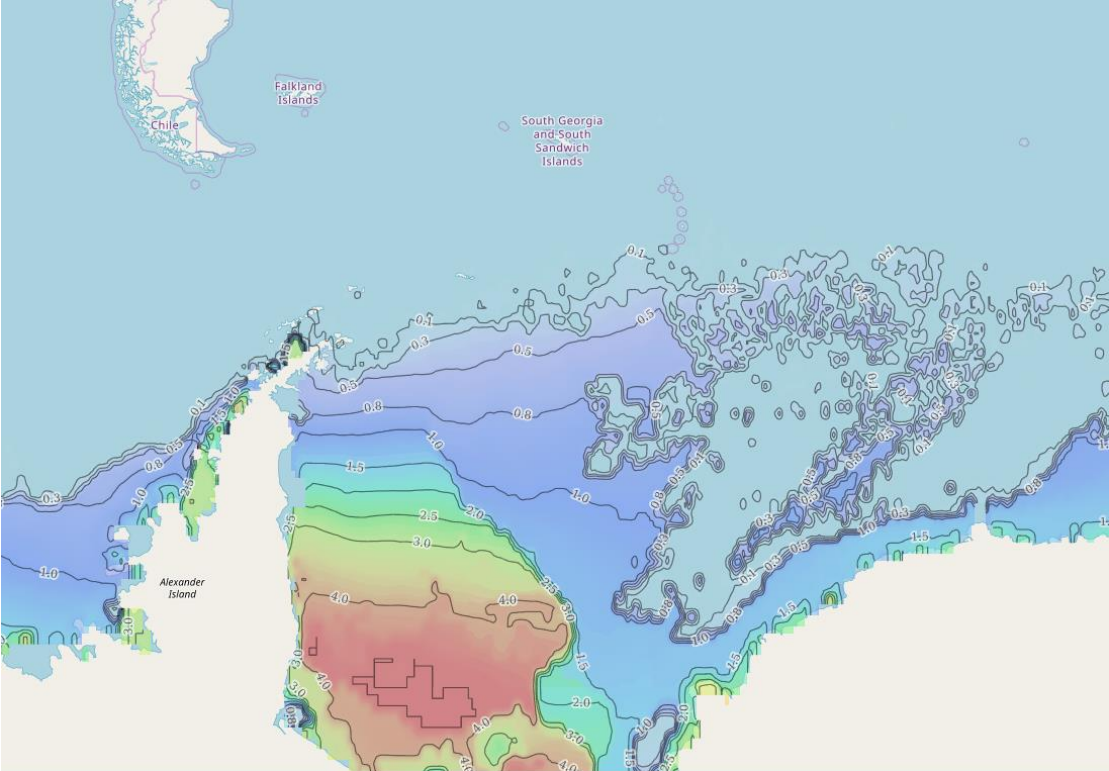


Attachment 15 Humidity percentage style



CTU in Prague

Attachment 16 Ice thickness style

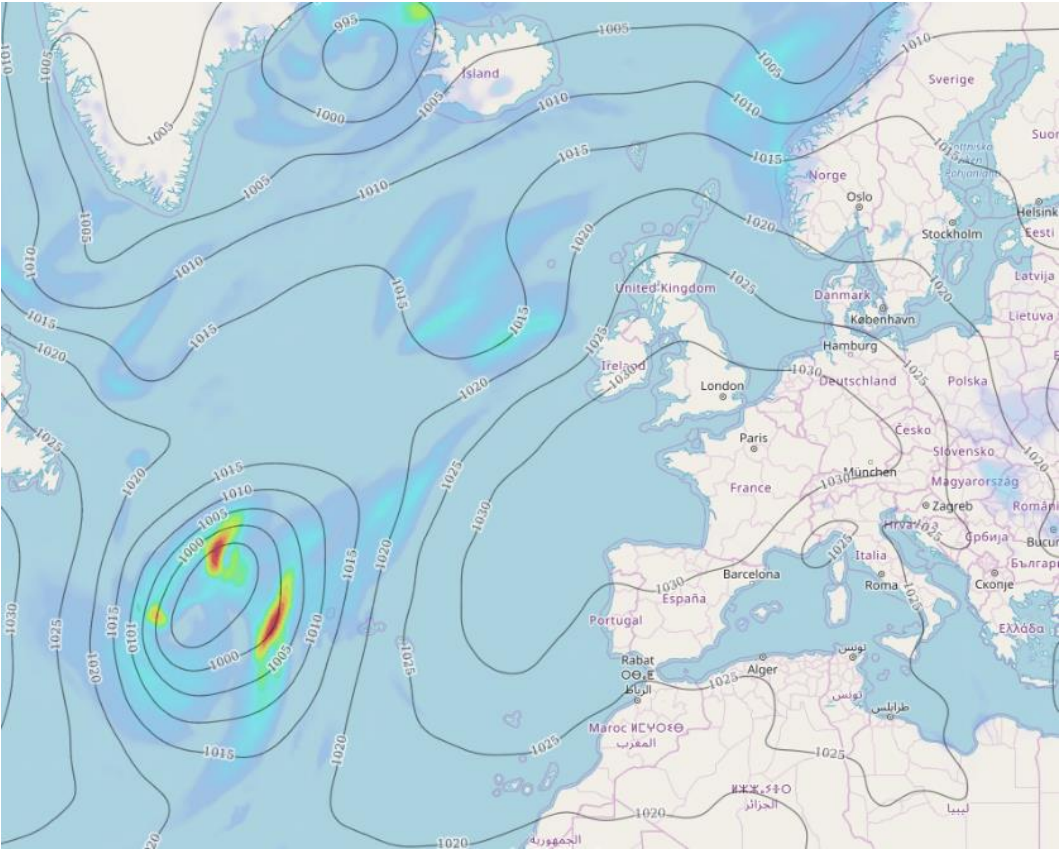


Attachment 17 Ice cover fraction style

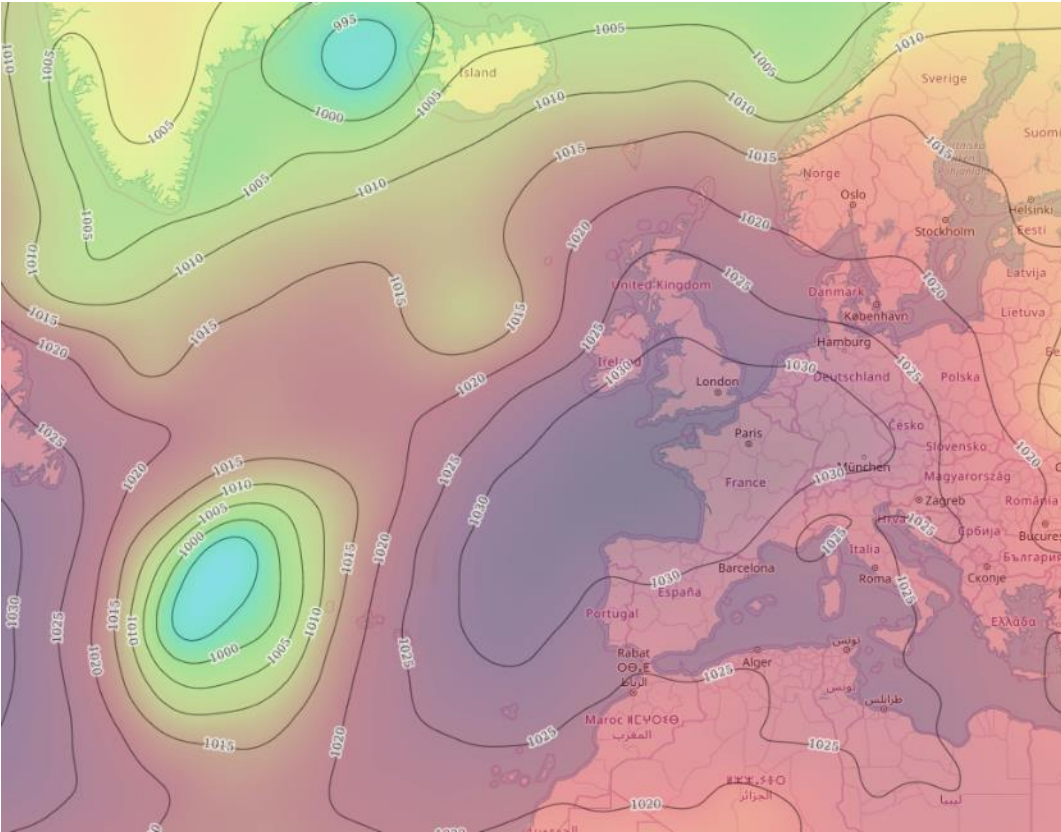


CTU in Prague

Attachment 18 Precipitation rate + pressure style

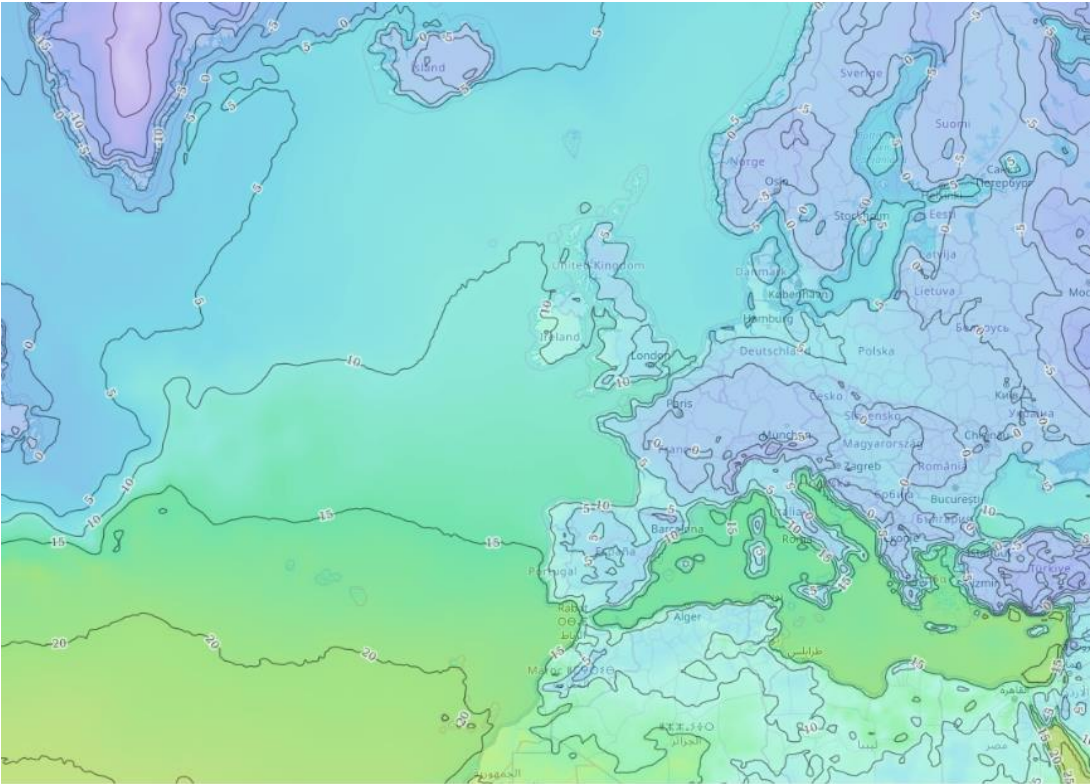


Attachment 19 Pressure style



CTU in Prague

Attachment 20 Temperature style



Attachment 21 Sunshine style

