Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 192 (2021) 3580–3589

Procedia Computer Science

www.elsevier.com/locate/procedia

25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems

Architectural Aspects of a Data-Intensive System: A Covid-19 Related Case Study

Piotr Dunin-Kęplicz[a,*], Michał Iwański[b], Marek Niezgódka[b], Piotr Wiśniewski[c]

[a] *keplicz.net, Warsaw, Poland*
[b] *CNT Center, Cardinal Wyszyński University, 01-938 Warsaw, Poland*
[c] *TensorCode, Warsaw, Poland*

Abstract

The Covid-19 pandemic caused serious turbulences in most aspects of humans activities. Due to the need to address the epidemic developments at extreme scales, ranging from the entire population of the country down to the level of individual citizens, a construction of adequate mathematical models faces substantial difficulties caused by lacking knowledge of the mechanisms driving transmission of the infections and the very nature of the resulting disease. Therefore, in modeling Covid-19 and its effects, a shift from the knowledge-intensive systems paradigm to the data-intensive one is needed. The current paper is devoted to the architecture of ProME, a data-intensive system for forecasting the Covid-19 and decision making support needed to mitigate the pandemics effects. The system has been constructed to address the mentioned challenges and to allow further relatively easy adaptations to the dynamically changing situation. The system is mainly based on open-source solutions so can be reproduced whenever similar challenges occur.

Keywords: data-intensive systems, architecture, knowledge engineering

1. Introduction and Motivations

An epidemic development that affects public life on countrywide level not only generates direct harm to all dimensions of public life but also causes long-term post-epidemic negative effects. Due to high complexity of the social interactions affecting transmission of the infections and their dynamic nature, any modeling effort to construct a dependable system that not only would produce accurate predictions of the epidemic spread but also practical tools for decision making support in combating the epidemic faces heavy difficulties.

* Corresponding author.
E-mail address: piotr@keplicz.net, m.iwanski@uksw.edu.pl, m.niezgodka@uksw.edu.pl, piotr@tensorcode.eu

A primary challenge is to master simultaneous barriers that prevent data access in any places and for any (often far from explainable) reasons. Then, because of the need to address the epidemic developments at extreme scales, ranging from the entire population of the country down to the level of individual citizens, a construction of adequate mathematical models faces substantial difficulties caused by lacking knowledge of the mechanisms driving transmission of the infections and the very nature of the resulting disease.

ProME is a research project aimed at setting up a modular system that will support monitoring and decision making for rational actions combating the Covid-19 epidemic spreading. The system developed within ProME is flexible so as to apply, up to quite limited adaptive actions, also to a range of future epidemic situations. The adaptivity of the system refers also to the multi-scale nature of the epidemic processes: depending on the precision of input data, the system can be applied at various scales of resolution.

It still remains to mention that the ultimate goal of ProME is to support multi-scenario decision making in combating epidemic developments of the range comparable to those caused by SARS-CoV-2, accounting not only for the diversity of activating agents but also for strong spatial differences in the process dynamics.

In this paper we expose the structure and architecture of the ProME system. Beyond the software-driven specification, we also present an outline of the hardware set-up. Of course, data-intensive architectures have been discussed in the literature (see, e.g., [8, 15, 22] and references there). The architecture presented in the current paper uses containerization technologies as the main tool for dealing with a variety of environments needed for implementing models. We also show how diverse state-of-the-art technologies can be combined for packaging and deploying large-scale data-intensive applications in a flexible and easily extendable manner.

The paper is structured as follows. In Section 2 we briefly describe models developed for the ProME system together with their use in decision processes. Section 3 is devoted to a high level system's architecture. Next, in Sections 4 and 5 we present data layer and the interface between models and presentation layer. In Section 6 we describe the physical and base software components of the system. Section 7 is devoted to the deployment of the system's components on nodes. Finally, Section 8 concludes the paper with final remarks.

2. ProME Models

The major focus of the ProME project has been set on modeling the Covid-19 epidemic and reacting on its impact on the healthcare system. The developed models can be divided into three groups:

- Predictive models for forecasting the main indicators of the epidemic intensity and the demand for medical procedures for people with non–Covid-19 related diseases;
- Simulation models allowing both for forecasting the epidemic as well as for comparing the effects of preventive measures intended to reduce the infection rate and other negative factors caused by the epidemic;
- Decision support models for relocating medical procedures for non-Covid-19 related diseases.

Let us now briefly describe models developed and selected for further use in the ProME system. Note that the models mainly report their results at the level of districts or larger regions.[1]

2.1. Predictive/Forecasting Models

There are three predictive models in the ProME system:

- *Nearest Neighborhood Model* NN [21] employs the *forecasting by analogy* method. It assumes that the epidemics share the same patterns of behavior in similar geographic regions but perhaps shifted in time. It compares regions using similarities between trajectories representing the Covid-19 dynamics. Given a region of interest, one selects a number of regions with similar characteristics in the past and applies the past trajectories to forecast the future course of the epidemic in the given region.

[1] The administrative division of Poland is based on three levels of subdivision: voivodeships (*województwo*) further divided into counties or districts (*powiat*), and these in turn are divided into communes or municipalities (*gmina*).

- *Neural Networks* model, Neural, uses neural networks for predicting the future values of indicators such as infection and death rates at the level of districts. It also aggregates the results to larger geographical regions. The neural networks have been implemented using TensorFlow [25].
- The *Demand for Medical Procedures* ProCME model [17] allows one to forecast the demand for non-Covid-19-related medical procedures. The model applies a Bayesian forecasting method of [24]. Currently it covers the urology sector only. Given the procedure type (classified according to the WHO ICD System), the month, and the district, the model returns the expected demand of medical procedures.

2.2. *Simulation Models*

The following simulation models have been developed:

- *Multiagent Simulation System* ProMES [9] develops a well-known Covasim simulator [7] and adapts it to the Polish context. The extended version includes some functionalities not present in Covasim. It allows one to simulate a variety of preventive measures and scenarios intended to reduce the risk of Covid-19 transmissions. The measures include physical or social distancing, quarantining, the use of face masks, etc.
- *Cellular Automata Model* ProMECA [11] allows one to simulate the pandemics dynamics using cellular automata, where cells may represent people or geographical regions. The model allows one to simulate the Covid-19 transmission and to predict selected indicators characterizing the Covid-19 dynamics.

2.3. *Decision Support Models* ProMELP, OptiLoc

Decision Support Models, ProMELP [12] and OptiLoc [17] address optimal relocation of medical procedures. Covid-19 may seriously affect the healthcare system and cause a substantial reduction of non-Covid-19-related procedures in particular regions. The models allow one to plan the relocations of procedures to medical facilities in other regions and to verify the feasibility of completing the procedures within the specified time period. Optimality may be measured with respect to various criteria, including minimization of patients' risks, time required for completing the procedures, overall costs involved, etc. The models use mixed integer linear programming techniques with the Cplex library [6].

3. High-Level Architecture of ProME System

First and foremost, the principal rule for building architecture for the ProME project was to choose open-source software components that have commercial equivalents. The idea behind this approach was to be able to redeploy the prototype solution to the environment with full software vendor support with little or no additional effort.

The only exception to this rule is the CATO computing system[2] which works with its native software stack. However, in the line of the Machine Learning trends, some of the software components available and used in this system to some extent intertwine with the open source space. These are software components that are frequently used by commercial companies and research institutions.

Another assumption behind the developed architecture in the context of the conducted research was its scalability. In line with this assumption, at the present stage of the prototype we developed a technologically coherent system that performs its functions based on the acquired data sets but ready for further extensions.

The ProME system consists of four main logical components (see Fig. 1):

- A data lake and a warehouse to provide input data for the models.
- A predictive, simulation and decision support models repository.
- An API that delivers the computations and selected parts of the database to the clients.

[2] CATO is a proprietary of Digital Science and Technology Center Cardinal Stefan Wyszynski University solution consisting of hardware and software components developed and manufactured by IBM, which architecture is modeled on Summit and Sierra supercomputers [5]

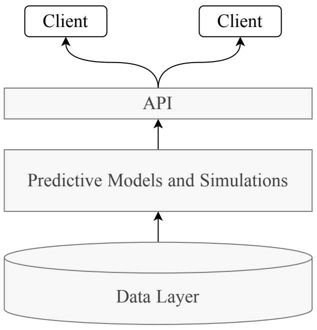

Fig. 1. The ProME architecture.

- A presentation layer consisting of two client types that deliver the results of computations: a web application and a desktop application.

4. Data Layer

4.1. Main Requirements

The accessibility and quality of the data is a necessary condition for faultless functioning of the ProME system. Any irregularities in the data could cause inaccurate learning of the models which in turn would affect the quality of responses provided by the system. Since each of the models requires specific data to train there need to be dedicated data marts with already curated and validated data. In order to ensure audibility and effective monitoring of the progress in the quality of models, all data entering the system should be versioned. Due to the fact that data comes from various sources, the system should be ready to accept them in any format.

4.2. High-Level View of the Architecture

To ensure the ease of loading new data into the system and the high quality of this data for consumers the subsystem responsible for data management is divided into two key components:

- Data Lake which is the entry point to our system and contains raw, unprocessed data in many formats.
- Data Warehouse which is a component that contains all the data in processed form, ready for use by other elements of the system.

Any data flow between the two components is done using ETL (Extract, Load, Transform) processes. Thanks to this abstraction layer any inaccuracies in the data entering the system will not directly affect their consumers. The Data Warehouse consists of strictly defined entities created on the basis of analyzes of how the data is used in the system. To further facilitate the use of data, all objects in the database are described in special tables that store metadata.

4.3. Data Collection and Curation

As mentioned before the data enters the system as files of various formats and must then undergo a series of transformations before it can end up in the data warehouse. The key steps are:

- *Verification*: all the data is validated with previously created schemas and constraints.
- *Anonimization*: the process of eliminating any sensitive data entering the system. In the case of detecting such cases the data has to be hashed.
- *Normalization*: unnecessarily tangled data is decomposed. It simplifies the process of creating final entities in the Data Warehouse

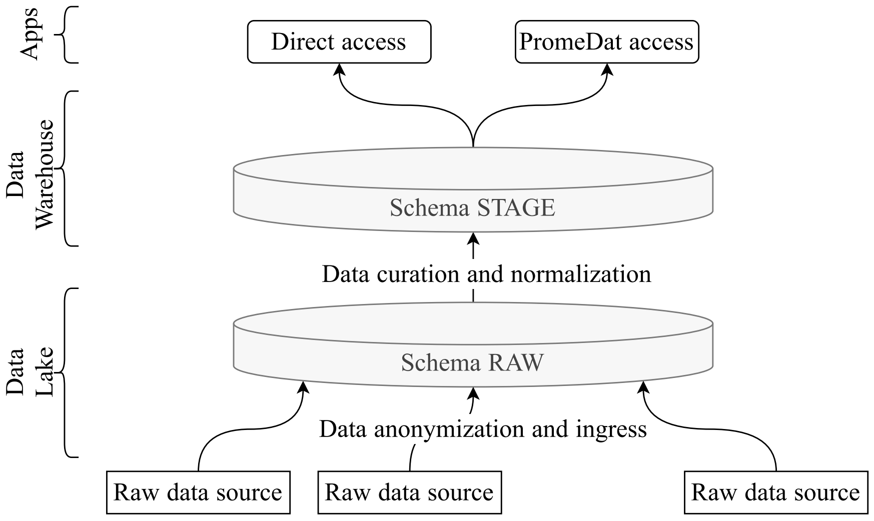

Fig. 2. The data layer schema.

In order to keep flexibility when it comes to data collecting from various sources we heavily rely on PostgreSQL functions which enable creating tables on top of raw files in many formats like CSV, JSON or XML.

4.4. PromeDat Library as a Data Access and Snapshot Layer

In order to facilitate data access, a helper library called PromeDat was developed. It relies on the SQLAlchemy [23] toolkit to access the main ProME database and reflect the structure of the tables, views, and columns.

PromeDat introduces a common format of the metadata to describe the curated input data and allows to load additional table and column descriptions from the comments attached to the PostgreSQL objects. Data scientists and developers can easily connect and browse the metadata in the JSON format.

With metadata loaded to memory, PromeDat provides an instant access to:

- SQLAlchemy table objects that allow to prepare and execute SQL queries in a safe and compatible manner;
- Pandas [14] frames for flexible data access, analysis, and manipulation within Python applications.

Moreover, PromeDat allows to define and create snapshots of the data located in the ProME database. These snapshots – data-marts – are created in separate schemas of the PostgreSQL database as either table copies or view definitions (normal or materialized). This way developers can ensure that computations use a consistent and well defined dataset extracted from the ever evolving data lake.

5. PromeAPI: the API for the Presentation Layer

5.1. Main Requirements

In order to fill the gap between the raw data calculated by the models described in 2.1 and the presentation layer comprised of the VisNow data visualization tool (see [20]) and a web browser client, an API layer had to be provided.

Such an API layer should provide a uniform interface for all the models with similarly structured request parameters and response schemas, and a common data format. The API layer should also facilitate cooperation between authors of the computational models and the presentational layer by allowing to quickly upgrade the API codebase. We also need to take into account that different modules of the ProME environment may require different (possibly conflicting) sets of tool libraries. Last, but not least, the API layer should be easily scalable so that the requested computations, both CPU- and memory-intensive, can be performed simultaneously and in a timely manner.

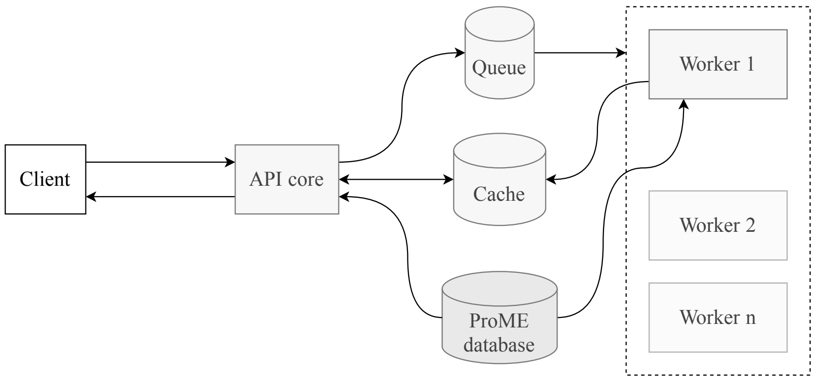

Fig. 3. The PromeAPI architecture.

5.2. *High-Level View of the Architecture*

The PromeAPI (see Fig. 3) application consists of:

- The core FastAPI application that receives client requests and validates them;
- Multiple workers to execute predictive models computations;
- Two databases (a relational database to cache requests and responses, and an in-memory data store to control task queue).

The workers can also access the ProME database layer, either directly or by means of the PromeDat library (see Section 4.4). All of the components are deployed to the infrastructure as Docker containers (see Section 5.6).

5.3. *FastAPI and Pydantic*

The core of the API layer uses FastAPI [3], a Python web framework for OpenAPI [13] compliant services. While there are several Python web frameworks available, most of them are designed for feature rich server-side web applications. On the other hand, FastAPI provides a simple way to define data-centric API endpoints, the input and output schemas and their textual description, and automatically provides an OpenAPI schema (that can be used to create client-side data access library), as well as a human readable documentation for client application developers.

The input and output schemes are built using the Pydantic library [18]. This library extends class definitions using Python 3.5+ type annotations with metadata and additional validators. Coupled with the FastAPI core, this tool automatically checks and validates input and output requests and provides meaningful error messages in case of any inaccuracies. It can also cast the data from JSON to native Python objects and vice-versa.

5.4. *ProME Models*

Most of the ProME models, described in Section 2, are implemented as Python libraries. They provide similarly structured API entrypoints together with the input and output data schemata using Pydantic. These schemata are not only automatically translated to proper OpenAPI definitions, but also allow to validate and cast JSON data to Python objects.

For a model not implemented in Python (the Nearest Neighborhood Model) an alternative way of execution is provided. It can be launched as a subprocess with the input and output data (specified in JSON) provided on their *stdin* and *stdout* file descriptors.

5.5. Asynchronous Computations and Cache

The computation can take anywhere between seconds and hours to complete. Since it's not feasible to lock the client for an unknown amount of time, the requests are first put into a queue, computed in background, and only after these computations are complete, the client can repeat the request and expect the full response with output data ready.

In order to provide such functions, the following components are needed:

- A database to store request parameters and computed responses. PromeAPI uses a local PostgreSQL [16] database for persistent storage of such data;
- An in-memory database acting as a queue of tasks to compute. PromeAPI uses a local Redis [19] instance to hold such queue;
- A Celery [1] broker that controls job execution using the task queue.
- Celery workers that process incoming tasks and store computed results in the PostgreSQL database.

The full operation sequence is as follows:

1. The client sends a request.
2. The server validates the request. In case of an error, it returns with a HTTP status [10] 422 "Unprocessable Entity" response.
3. The server looks up any previous API calls with the same parameters.
    - When found, the previously computed results are returned as a HTTP status 200 "OK" response;
    - Otherwise, a job definition is created and put in the local PostgreSQL database, and a Celery task is initialized. The client receives a HTTP status 202 "Accepted" response with an identifier of the job that has been initialized.

In order to check the job status, the client can either repeat the HTTP request and check its status code or use a status API endpoint that returns the job information (and, possibly, a computed response). Note that this execution model acts as a cache and in fact several clients can perform the same request and share the results calculated only once.

5.6. Container Services

Containerization techniques have become increasingly popular in the recent years. Lightweight virtualization services, such as Docker [2], bring many benefits to the development process:

- Fine grained control over build process (base operating system requirements, libraries and settings);
- Consistent, repeatable builds;
- Isolation from the underlying operating system that allows to deploy and test contained services without modifying existing host configuration;
- Ease of vertical scaling by deploying additional containers with duplicated services;
- Increased security by isolating the services from each other and control over shared resources.

These benefits are amplified by the fact that modern development team tools such as GitLab [4] heavily rely on containerization of the applications being developed. *Continuous Integration and Continuous Deployment* processes (usually referred to as CI/CD) performed by a central GitLab instance replay application build stages, execute tests, and allow to deploy production ready images to the infrastructure. Some parts of the ProME are covered by such processes.

As mentioned earlier, containerization enables fine grained control over the execution environment. In fact while many components of the ProME system were developed using Python version 3.8, one of the prediction models uses Python version 3.7 due to some incompatibilities in the recent versions of the TensorFlow library. With Docker containers it is straightforward to prepare, test, and deploy a modified version of the PromeAPI worker that uses specific versions of the Python environment and its libraries.

Using Docker images also allows us to deploy several Celery workers to handle concurrent computations on several independent nodes. Additional Docker containers can be deployed and controlled with minimal configuration on new host nodes.

6. Computational Resources

The computational infrastructure for modeling in the ProME project includes full stack hardware and software components. In addition to models and software developed for the purposes of the ProME project, the entire system is a combination of standard hardware and software products, such as:

- x86_64 servers, disk array, FC and Ethernet networks;
- open source software;
- the CATO system, designed for large-scale computing, extended with the tasks related to artificial intelligence and high-performance data analysis based on large data sets.

The base layer for runtime environments is oVirt, an open source virtualization management platform. It offers many features characteristic to enterprise solutions while remaining free. It allows one to administer hosts, disk spaces, logical networks and virtual machines. It is also the foundation for the Red Hat Enterprise Virtualization environment. The operating system layer consists of Linux distributions belonging to the Red Hat family.

In the ProME computing infrastructure environment for the modeling purposes, the CATO system plays a significant role. The CATO, serving as a computing platform for models developed as part of the project, hosts those models that require considerable computing power for efficient execution. As part of the project the so-called Resource Groups and Instance Groups were created and configured in the CATO system with the use of IBM Spectrum Conductor according to the project participants' needs. For configuration purposes two types of resources were used: hardware resources (AC922 nodes) and software resources using different versions of the installed packages in different virtual environments. As part of the Instance Group, software developed as part of the ProME project was created and launched using tools such as Jupyter Notebook, Spark, Watson Machine Learning Accelerator along with the Deep Learning Impact component.

Striving to build a system in the microservices paradigm, individual software components are migrated to the OKD cluster space which provides a complete Open Source platform for container applications. It allows one to manage container clusters and is enhanced with application lifecycle management and DevOps tools.

An important role of the entire infrastructure solution is played by GitLab, serving not only as a repository service based on the version control system, but also as a platform for the DevOps methodology and CI/CD. The use of these components is aimed at building a fully flexible and automated environment, so that the system is easily adaptable to various situations and allows for easy extensions by new models and modifications of the existing ones.

While discussing the computing infrastructure for modeling and project implementation purposes, it is impossible not to mention the security aspect. The access to production environments for project participants is possible through a VPN tunnel only, and the management of access and authorizations to individual subsystems is carried out through the LDAP directory service.

7. Production Architecture

The ProME system has been implemented as a set of virtual machines operating under the oVirt hypervisor. In virtual system instances, separate environments have been prepared addressing particular system's layers (see Fig. 4):

- The *access zone*, consisting of the PromeEXT subsystem, separates the external data uploads from the rest of the system's virtual environments. This separation was imposed by security demands.

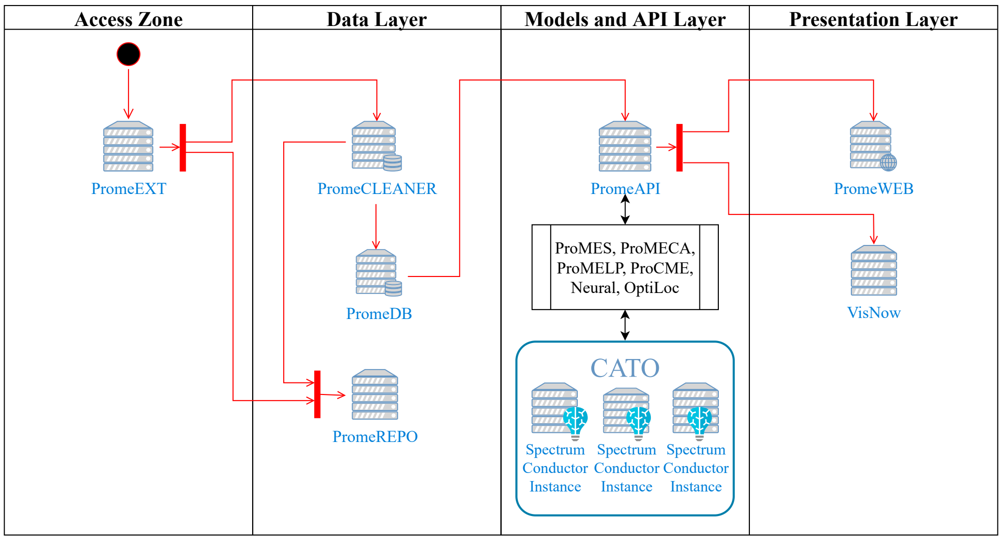
Access Zone
Data Layer
Models and API Layer
Presentation Layer
PromeEXT
PromeCLEANER
PromeDB
PromeREPO
PromeAPI
ProMES, ProMECA, ProMELP, ProCME, Neural, OptiLoc
CATO
Spectrum Conductor Instance
Spectrum Conductor Instance
Spectrum Conductor Instance
PromeWEB
VisNow

Fig. 4. The ProME production architecture.

- The *data layer* was based on separate database environments built for transformation and cleaning raw data as well as a data warehouse environment directly used by the system models. The access to database runtime environments has been restricted for the project teams according to their needs and scope of activities and responsibilities. Maintaining separate environments and a granular access to databases was a solution addressing various data sensitivity levels. Data cleaning and transformation is performed on the PromeCLEANER virtual environment, whereas data directly accessed by models are stored in the PromeDB system. Besides these environments, a raw data repository — the PromeREPO — is being maintained.
- The *models and API layer* is launched as a subsystem consisting of Docker containers located in a dedicated virtual machine PromeAPI. It contains a set of API features and running models developed for the project purposes. Models are trained using the CATO environment.
- The *data presentation layer* was created as a separate virtual environment with a running HTTP server. The server provides dynamically generated pages containing the results delivered by models for registered and logged in users. Dynamic content is created using the PHP server-side interpreter and additional packages and libraries such as Bootstrap, jQuery, LESS-CSS, Composer, and OpenLayers. The website allows user registration and provides great possibilities of query parameterization. Due to the security aspects, user accounts related to the data presentation layer are stored in a separate database.
  The website is integrated with the system's API and the parametrization and interpretation of the data is carried out using the JSON format. The whole environment is maintained on the PromeWEB virtual system.

For the sake of architectural clarity, some components have been intentionally removed from the production architecture diagram. At the present, the environment is being transferred to the microservices architecture based on the OKD cluster applying the full CI/CD process using the GitLab instance.

8. Conclusions

In the paper we have discussed the architecture of ProME, a data-intensive system for forecasting Covid-19 pandemics and supporting actions mitigating its effects. We address both software and hardware components. The system mainly uses open-source solutions and can be reproduced in many environments.

In the future we plan to further develop the architecture to make it as self-adaptable as possible, in particular allowing for fully automatic monitoring, and at least semi-automatic data collection and curation as well as retraining models whenever needed. However, such developments do not affect the designed architectural foundations discussed in this paper.

Acknowledgment

The research has been supported by the Polish Ministry of Science and Higher Education under project ProME "Predictive Modeling of the Covid-19 Epidemics". We would also like to thank ProME Team members for discussions and support.

References

[1] Celery, 2021. Celery: a distributed task queue. URL: https://docs.celeryproject.org/en/stable/.

[2] Docker, 2021. What is a container? URL: https://www.docker.com/resources/what-container.

[3] FastAPI, 2021. FastAPI: a web framework for building APIs with Python 3.6+ based on standard Python type hints. URL: https://fastapi.tiangolo.com/.

[4] GitLab, 2021. Gitlab: the open devops platform. URL: https://about.gitlab.com/.

[5] IBM, 2019. Meet two of the most powerful supercomputers on the planet. URL: https://www.ibm.com/thought-leadership/summit-supercomputer/.

[6] IBM, 2021. IBM Cplex Optimizer. https://www.ibm.com/analytics/cplex-optimizer/.

[7] Kerr, C., Stuart, R., Mistry, D., Abeysuriya, R., Hart, G., Rosenfeld, K., Selvaraj, P., Nunez, R., Hagedorn, B., George, L., Izzo, A., Palmer, A., Delport, D., Bennette, C., Wagner, B., Chang, S., Cohen, J., Panovska-Griffiths, J., Jastrzebski, M., Oron, A., Wenger, E., Famulare, M., Klein, D., 2020. Covasim: an agent-based model of Covid-19 dynamics and interventions. medRxiv .

[8] Kleppmann, M., 2017. Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media.

[9] Latkowski, R., Dunin-Kęplicz, B., 2021. An agent-based Covid-19 simulator: Adjusting Covasim to the Polish context. Procedia Computer Science 25th Int. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems. This volume.

[10] MDN, 2021. HTTP response status codes. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

[11] Nguyen, H.S., Podolski, P., 2021. Cellular automata in Covid-19 prediction. Procedia Computer Science 25th Int. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems. This volume.

[12] Nguyen, L.A., Szałas, A., 2021. Optimization models for medical procedures relocation. Procedia Computer Science 25th Int. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems. This volume.

[13] OpenAPI, 2017. OpenAPI specification, version 3.0.0. URL: https://spec.openapis.org/oas/v3.0.0.

[14] Pandas, 2021. Pandas: a fast, powerful, flexible and easy to use data analysis and manipulation tool. URL: https://pandas.pydata.org/.

[15] Pavlo, A., Paulson, E., Rasin, A., Abadi, D., Dewitt, D., Madden, S., Stonebraker, M., 2009. A comparison of approaches to large-scale data analysis, in: Proceedings of the 35th SIGMOD International conference on Management of Data, pp. 165–178.

[16] PostgreSQL, 2021. Postgresql: the world's most advanced open source relational database. URL: https://www.postgresql.org/.

[17] Powała, A., Woźnica, A., 2021. Optimal relocation of medical procedures under the Covid-19 regime: an urology case study. Procedia Computer Science 25th Int. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems. This volume.

[18] Pydantic, 2021. Pydantic: data validation and settings management using Python type annotations. URL: https://pydantic-docs.helpmanual.io/.

[19] Redis, 2021. Redis: an open source, in-memory data structure store, used as a database, cache, and message broker. URL: https://redis.io/.

[20] Regulski, P., Wendykier, P., Kantiem, K., Murdzek, W., 2021. Advanced methods of visual analysis and visualization of various aspects of the Covid-19 outbreak in Poland. Procedia Computer Science 25th Int. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems. This volume.

[21] Rusin, T.M., 2021. Forecasts of Covid-19 evolution by nearest epidemic trajectories detection. Procedia Computer Science 25th Int. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems. This volume.

[22] Sabek, I., Chandramouli, B., Minhas, U.F., 2019. CRA: Enabling data-intensive applications in containerized environments, in: Proc. 2019 IEEE 35th Int. Conf. on Data Engineering (ICDE), pp. 1762–1765. doi:10.1109/ICDE.2019.00192.

[23] SQLAlchemy, 2021. Sqlalchemy: the database toolkit for Python. URL: https://www.sqlalchemy.org/.

[24] Taylor, S., Letham, B., 2018. Forecasting at scale. The American Statistician 72(1), 37–45.

[25] TensorFlow, 2021. TensorFlow Federated: Machine learning on decentralized data. https://www.tensorflow.org/federated.