TECHNIUM

Technium Vol. 4, No. 9 pp.76-88 (2022)
ISSN: 2668-778X
www.techniumscience.com

A crowdsensing cloud-based microservice for weather forecast

Dimitris Selis, Evi Papaioannou, Giorgos Moisiadis, Christos Kaklamanis

University of Patras and CTI "Diophantus", Greece
papaioan@ceid.upatras.gr

Abstract. We present a cloud-based microservice enabling individual users to have access to local air pressure measurements and a proof-of-concept web application providing shot-term weather forecasts based only on local crowdsensed data. Our microservice has been developed using state-of-the-art web technologies and cutting-edge technologies related to cloud computing. Apart from its individual interest and potential usefulness, our microservice and proof-of-concept web application show how modern cloud-based technologies and tools can be nicely combined to improve the community daily routine and wellbeing.

Keywords. Crowdsensing, microservice, container, cloud computing, air-pressure measurements, short-term weather forecast, Go, Docker, Kubernetes, Prometheus, Grafana, html, css, Flexbox.

1. Introduction

The widespread of smartphones has enabled crowdsensing, i.e., the harvesting of large amounts of monitoring data in urban areas generated by a large mass of mobile devices of users visiting those areas. This data can be subsequently channelled to ICT platforms for enabling various services aiming (ideally) to contribute to the community wellbeing. Crowdsensing falls into a subcategory of crowdsourcing where sensors are the actual sources of the data gathered and are distributed to a specific group of people which, while doing their everyday tasks, can contribute effectively [3].

The rapidly evolving Cloud computing model has emerged as a new technology for storing and processing data in real time. It provides easy, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be exploited with minimal effort as far as management and service provider interaction are concerned. In this context, microservices have emerged as a new architecture in which complex applications are structured as a collection of independent services. National Institute of Standards and Technology (NIST) defines that Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [5].

More precisely, Cloud computing relies on virtualization techniques to achieve large-scale resource elasticity. Virtual machines possessed a leading role at the infrastructure level by providing virtualized operating systems [6]. Containers are a similar but lighter virtualization technique. Although virtual machines and containers are both based on similar virtualization approaches, they fall into different domains when it comes to implementation and configuration. Containers act as tools that serve operating systems based on their principles. In contrast, virtual machines are directly related to resource allocation and management. Containers can be used as substitutes for virtual machines,

facilitating the allocation of resources by dividing the workload. For portable, interoperable applications in the cloud, a lightweight distribution of application packages is necessary for development and management. The containers therefore provide a lightweight and portable execution environment, the ability to develop and test applications on a wide range of servers and a solid interconnectivity on which we relied during the deployment process.

Weather forecast based on local information makes a rather popular heuristic approach. Imagine that you are talking on the phone with a friend back in your place of residence while being on vacation at a seaside 100km away. If your friend tells you "it's rainy here", what would be your spontaneous thought/conclusion? Most probably something like "let's get ready for rain in the next couple of hours". Weather forecasts are based on the combination of several factors which include temperature, humidity, air pressure, wind speed and direction. Motivated by this realistic observation and exploiting the modern technological infrastructure described above, we designed and implemented a microservice enabling individual users to have access to air pressure measurements and a proof-of-concept web application providing shot-term weather forecasts currently based only on local crowdsensed air pressure data.

The problem of computing exact location forecast data has attracted the attention of the research community. Recently, several efforts have been made in order to provide a consistent and precise solution. For example, Liu et al. [2], in their work on accessing weather data using deep neural network, developed a computational intelligence technology and, by implementing predictive models, presented forecasts for even thirty years ahead. Furthermore, Montori et al. [4], in their work about smart cities and environmental monitoring, exploited crowdsensing using data collected by smartphones in order to contribute to environmental phenomena prediction.

The novelty of our approach stems from the exploitation of the notion of locality, both in crowdsensed data and obtained short-term predictions, as well as from the emphasis placed on the quality of the user experience. Exploiting the huge number of sensor-equipped mobile devices and cutting-edge technology, we do not seek for obtaining weather forecasts based on global data; we wish to obtain location-based short-term weather predictions by appropriately exploiting only location-relevant data available in the cloud. In addition, our approach draws near the user's perspective, that is, the need for fast, easy and accurate weather forecast with the minimal requirements for resources, i.e., network speed, processing power etc. Apart from its individual interest and potential usefulness, our microservice shows how modern cloud-based technologies and tools can be nicely intertwined to improve the community daily routine and wellbeing.

The rest of the paper is structured as follows. In Section 2, we address in detail the design and implementation of our microservice. In Section 3, we present a use case scenario for demonstrating the microservice and its functionalities. We conclude in Section 4 also discussing future plans.

2. Microservice design and implementation

In what follows we present design and implementation details for our microservice. The proof-of-concept application which utilizes the microservice for air pressure measurements in order to perform sort-term weather prediction is available at http://34.154.30.122:8081/.

2.1. Microservice design

Microservices is an implementation of service-oriented architecture for software development consisting of small services that can be deployed and scaled independently. Each microservice runs in its own process and communicates through lightweight mechanisms, often using application programming interfaces - APIs. Regarding the advantages offered by coherent microservices they implement a predefined and specific number of functions, which makes the code small and inherently limits the scope of a bug. Given their architecture, gradual migration to new versions is feasible. The new version is developed simultaneously with the old one and the services that depend on the latter are gradually modified to interact with the former, thus promoting continuous integration and continuous delivery.

Our microservice, outlined in Fig. 1, has been containerized with Docker [8] providing three replicas sets, which minimize the possibility of system downtime. The containers pull their image from the Docker Hub, which owns a public repository for storing and distributing various Docker images. On top of that, Kubernetes [12] runs, supervising system availability and orchestrating the containers that dockerize our application. Developers have access to the Docker containers through the Docker GUI. Alternatively, they can use the command line tool (cmd, terminal) to interact with the containers using the Docker commands. Docker Deamon is a subprogram that runs in the background and provides all the required actions related to containers. Docker hub refers to the repository owned by Docker. Various images are available to download and install to a container via the command "docker pull". Developers can either download an existing image or upload their image. This is achieved via the registry module, a stateless server allowing for docker images storing and distribution.

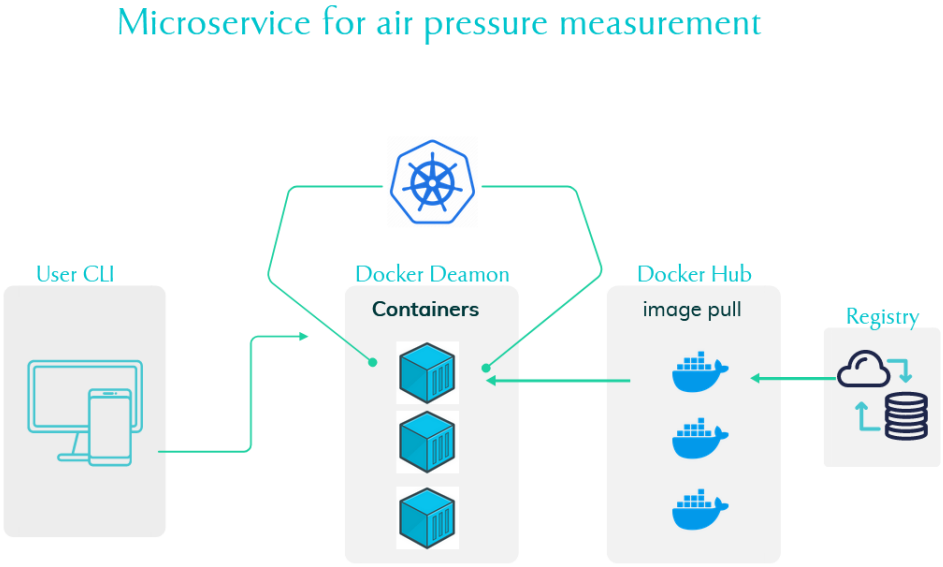

Figure 1: The design of our microservice

Similar microservices can be used for measuring other factors (e.g., temperature, humidity, air pressure, wind speed and direction) whose combination can provide short-term weather forecast based only on local data. Currently, we exploit the air pressure microservice as the underlying data-collection mechanism for a proof-of-concept web application which can provide users with shot-term weather forecasts based on crowdsensed air pressure data within their short-range neighborhood.

2.2. Microservice implementation

In this section, we overview programming languages, frameworks and other software used for the development of our microservice.

The back-end has been developed using Go for compatibility, trouble-free operation and synergy with containers and Kubernetes. Go [9] is expressive, concise, clean, and efficient. Its concurrency mechanisms make it easy to write programs that get the most out of multicore and networked machines, while its novel type system enables flexible and modular program construction. Go compiles quickly to machine code yet has the convenience of garbage collection and the power of run-time reflection. It's a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language. Furthermore, due to the nature of Docker and Kubernetes (both written in Go) the overall process of developing the application becomes swifter. Docker and Kubernetes showcase remarkable synergy regarding container deployment and orchestration and their concurrency allows

for uninterrupted operation of both. Within Docker, user interface containers as well as Kubernetes core elements are presented in detail. Moreover, Docker allows for Kubernetes integration without the need of installing additional dependencies that would mitigate the system's performance.

Kubernetes [11] is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. Kubernetes facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Its services, support, and tools are widely available. Therefore, Kubernetes was used to achieve effective orchestration of the containers.

The front-end is written using pure HTML [15] and CSS [13] along with Flexbox [14]. Firebox makes the responsiveness of the page more vivid and facilitates the overall experience of the user. The Flexbox Layout (Flexible Box) module (a W3C Candidate Recommendation as of October 2017) aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or changes dynamically (thus the word "flex"). The main idea behind the flex layout is to give the container the ability to alter its items' width/height in order to best fill the available space and correctly respond to all displays and screen sizes. A flex container expands items to fill available free space or shrinks them to save space. Flexbox provides a whole user experience without the need of Javascript implementation, thus mitigating the programmer's workload. The front-end is then served through a handler that receives HTTP requests. The overall application is exposed on a desired port. Then, the user's location is obtained and based on that a request is sent for getting the atmospheric pressure from the OpenWeather API [16]. Finally the data that is returned in JSON format is parsed and printed on the home screen. The overall process has been developed in a container so as to be more flexible, scalable and lightweight. For the containerisation of the developed application Docker was used.

Deploying an application within a container runtime is a simplified procedure. Containers offer a stable and consistent environment while aiding the overall deployment and mitigating space and memory needs. Furthermore, Docker allows for detailed monitoring through its interface. It has a smooth synergy with Kubernetes, hence, making continuous delivery and integration possible.

Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs. Moreover, in terms of security, containers provide a safe environment. The containers of the application are managed using Kubernetes which also contributes to the monitoring and the visualisation of the pods along with Prometheus and Grafana.

For the monitoring and the visualization of the Kubernetes pods state, Prometheus [12] and Grafana [10] were used. These two tools offer extensive monitoring and alerting. Prometheus stores data as time series such as metrics information about the pods providing supervision over the application whole state. Grafana consists of numerous utilities which aim to the visualization of the metrics fetched in order for stability to be provided.

2.3. Microservice deployment

The implementation of PreMe as a container through a Kubernetes cluster facilitates the overall process as Kubernetes makes containerized environments possible by acting as the orchestration system. Furthermore, scalability is one of Kubernetes' most significant attributes as applications are able to efficiently scale up and down based on actual demand. This also contributes to cost efficiency. Lastly, it provides the ability to run the application anywhere as every container runtime can be run along with nearly any type of infrastructure [7]. In order to deploy the app in a container, a Dockerfile must be specified. In such files the specifications of the app are provided in order for Docker to be able to proceed with the deployment of the container that builds a Docker image (Fig. 2). The image contains all the dependencies required by the application (Fig. 3). Moving on the deployment of the application in Kubernetes entails the creation of a YAML configuration file (Fig. 4).

```
static — -zsh — 119×36
jsel32@MacBook-Air static % docker build . -t kubmetrics
[+] Building 14.8s (7/7) FINISHED
 => [internal] load build definition from Dockerfile                       0.0s
 => => transferring dockerfile: 89B                                        0.0s
 => [internal] load .dockerignore                                          0.0s
 => => transferring context: 2B                                            0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine            3.8s
 => [1/2] FROM docker.io/library/nginx:alpine@sha256:082f8c10bd47b6acc8e  10.7s
 => => resolve docker.io/library/nginx:alpine@sha256:082f8c10bd47b6acc8ef   0.0s
 => => sha256:082f8c10bd47b6acc8ef15ae61ae45dd8fde0e9f389 1.65kB / 1.65kB   0.0s
 => => sha256:804f9cebfdc58964d6b25527e53802a3527a9ee880e 8.89kB / 8.89kB   0.0s
 => => sha256:2959a35e1b1e61e2419c01e0e457f75497e02d03936 1.57kB / 1.57kB   0.0s
 => => sha256:213ec9aee27d8be045c6a92b7eac22c9a64b4455819 2.81MB / 2.81MB   3.8s
 => => sha256:2546ae67167b3580b7dcc4a4d56e504594bac17e22e 7.40MB / 7.40MB   9.9s
 => => sha256:23b845224e138d383175fc5fb15d93ad0795468b8d6210e 601B / 601B   1.0s
 => => sha256:9bd5732789a330a86ed2257e6bf18928c6ae873107ef0a4 894B / 894B   1.4s
 => => sha256:328309e59ded59f3fc9eb5ade5c0135ec9aae5553ab837e 666B / 666B   2.3s
 => => sha256:b231d02e51502ce72ff0813057a12b7778148c2e629 1.40kB / 1.40kB   4.1s
 => => extracting sha256:213ec9aee27d8be045c6a92b7eac22c9a64b44558193775a   0.2s
 => => extracting sha256:2546ae67167b3580b7dcc4a4d56e504594bac17e22e046b2   0.4s
 => => extracting sha256:23b845224e138d383175fc5fb15d93ad0795468b8d6210ed   0.0s
 => => extracting sha256:9bd5732789a330a86ed2257e6bf18928c6ae873107ef0a40   0.0s
 => => extracting sha256:328309e59ded59f3fc9eb5ade5c0135ec9aae5553ab837ed   0.0s
 => => extracting sha256:b231d02e51502ce72ff0813057a12b7778148c2e629aa21f   0.0s
 => [internal] load build context                                          0.1s
 => => transferring context: 406.94kB                                      0.1s
 => [2/2] COPY . /usr/share/nginx/html                                     0.1s
 => exporting to image                                                     0.0s
 => => exporting layers                                                    0.0s
 => => writing image sha256:5b36f4f2bc69c668517665061df1f15610c3ca9a6daee   0.0s
 => => naming to docker.io/library/kubmetrics                              0.0s
```

Figure 2: Container build using Dockerfile

Dockerfile

```
1  # syntax=docker/dockerfile:1
2
3  FROM golang:latest
4
5  RUN mkdir /app
6
7  ADD . /app
8
9  WORKDIR /app
10
11 RUN go build -o main .
12
13 EXPOSE 8081
14
15 CMD [ "/app/main" ]
```

Figure 3: Dockerfile overview

Technium Vol. 4, No. 9 pp.76-88 (2022)
ISSN: 2668-778X
www.techniumscience.com

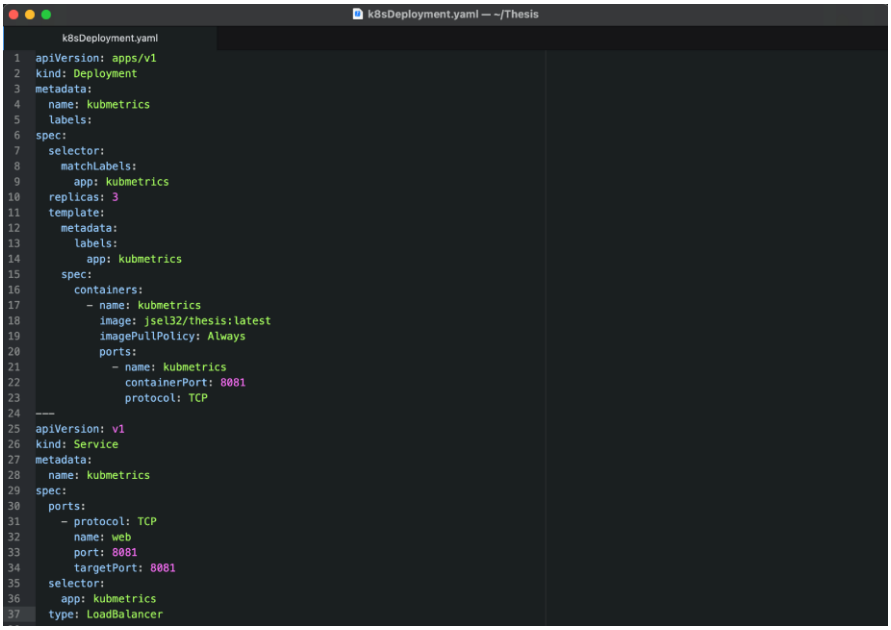

Figure 4: YAML configuration file for Kubernetes

3. Our proof-of-concept application

In this section, we present and overview of the proof-of-concept responsive web application, PreMe, and its functionalities. Fig. 5 shows the GUI of our application.

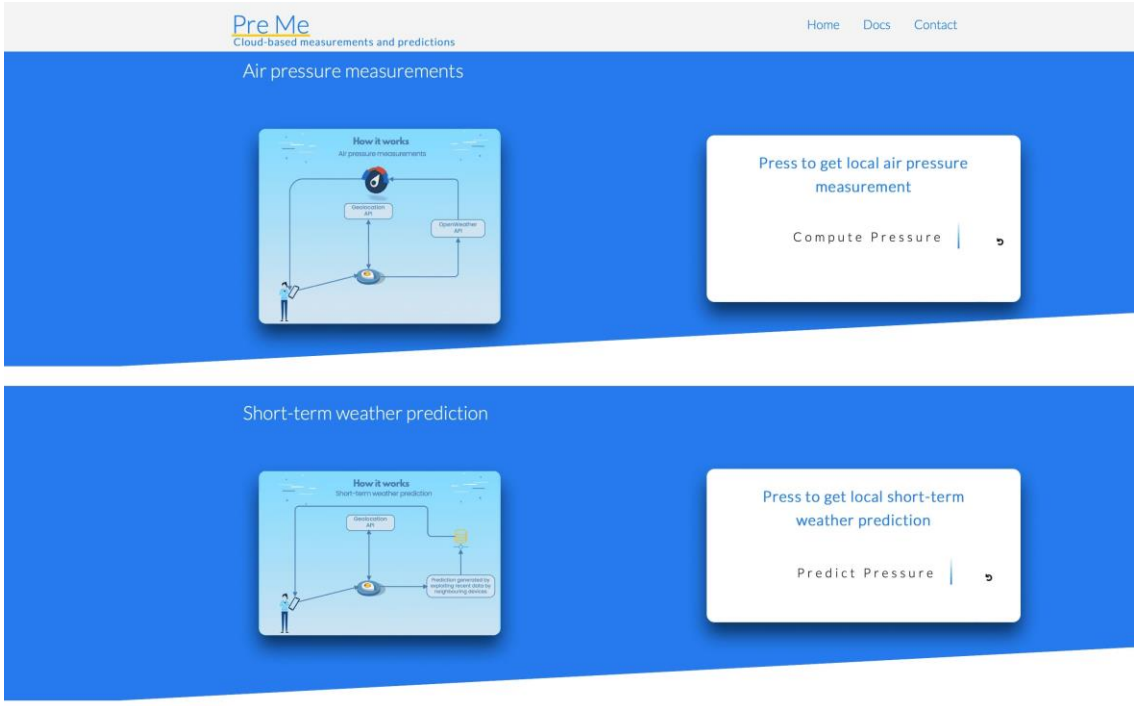

Figure 5: The application GUI

Users are currently offered two functionalities: “Compute air pressure” (Fig 6) and “Predict air pressure” (Fig 7). “Compute air pressure” works as follows. A script, written in Javascript, fetches

information about the user's IP address via which the location of the user is determined according to the corresponding longitude and latitude coordinates. Then, the user's location data, which is processed in the back-end via HTTP requests, is sent to the OpenWeather API [???]. Weather forecast data is retrieved and converted in JSON format. Then, the air pressure data is forwarded to the front-end on the user's screen using Javascript. How this functionality works is illustrated in Fig. 8. "Short-term weather forecast" works as follows. Recently-collected crowdsensed data within a relatively short range around the current user's location is used to provide short-term quantitative estimates for factors affecting the weather, which are then combined to give a single short-term weather prediction. Currently, weather prediction is based only on crowdsensed air pressure data collected during the last hour at a range of 100km around the user's current location. How this functionality works is illustrated in Fig. 9.

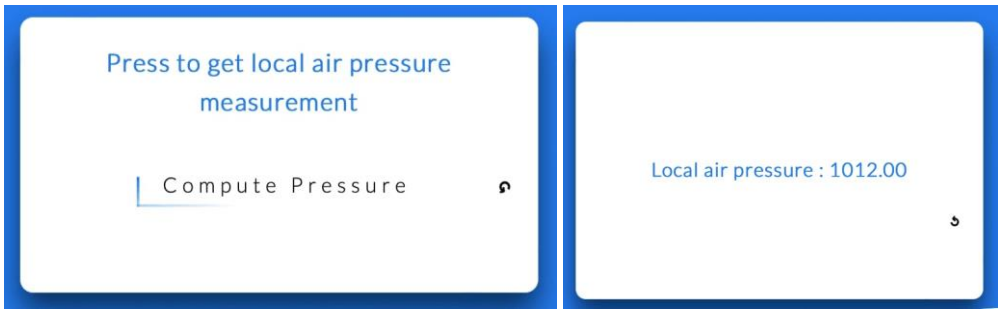

Figure 6: "Compute Pressure" button

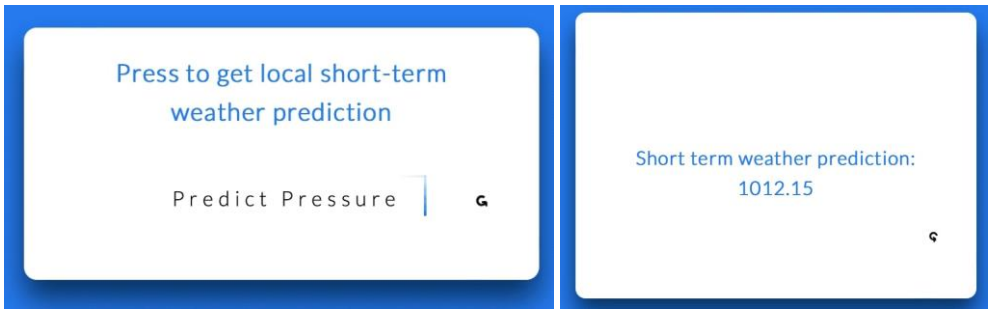

Figure 7: "Short-term weather forecast" button

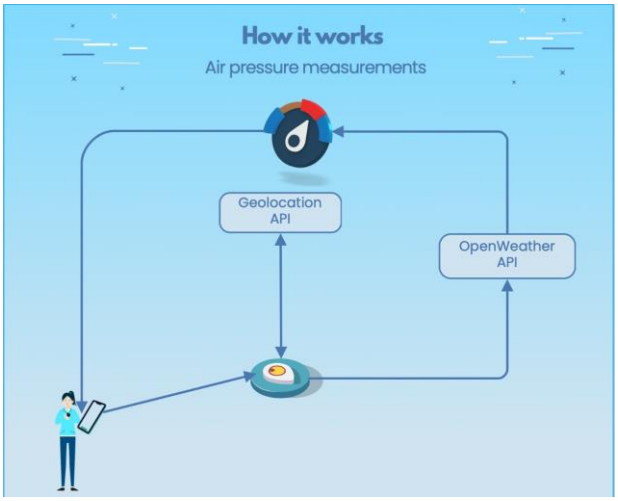

Figure 8: "Compute Pressure" functionality

Technium Vol. 4, No. 9 pp.76-88 (2022)
ISSN: 2668-778X
www.techniumscience.com

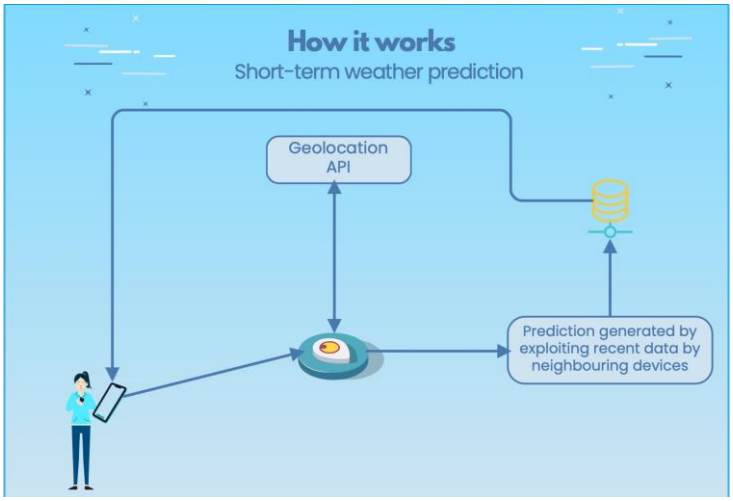

Figure 9: "Short-term weather forecast" functionality

Via the navigation bar at the top-right corner of the homepage, users can access the pages for Documentation and Contact. The Documentation page (Fig. 10, left part) displays information about the application and the way it is deployed through Kubernetes Loadbalancer. Via the contact page (Fig. 10, right part), users can send messages or submit requests for additional information.

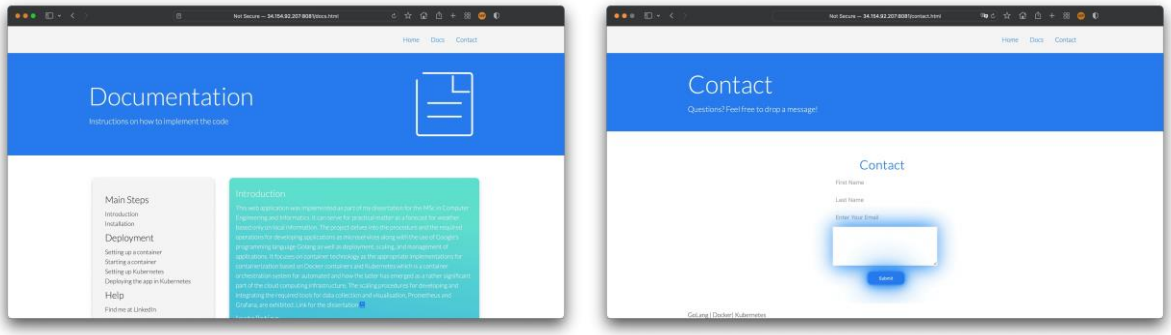

Figure 10: Documentation and Contact

3.1. Monitoring

For the monitoring of the application resources as well as the replicas of the containers, which ensure the consistency of the service and minimize the possibility of downtime, we have used Kubernetes.

Our web application was containerized with Docker. Using the specific image, the application has been deployed through a Kubernetes cluster. This whole procedure facilitates the scalability of the application and provides the ability to collect useful metrics about CPU and memory utilization and network incoming traffic to the container. The Kubernetes Dashboard (Fig. 11) shows detailed information about the deployment of the application and is a rather significant component when deploying cloud native applications.

Monitoring the incoming traffic to a deployed application as well as provisioning resources is among basic concerns of a system administrator. Consequently, regarding the monitoring of our deployments we have used Prometheus. Prometheus has an important synergy with Kubernetes as

Technium Vol. 4, No. 9 pp.76-88 (2022)
ISSN: 2668-778X
www.techniumscience.com

Kubernetes components themselves are instrumented with Prometheus metrics. Fig. 12 shows the total set of pods in the namespace. A pod is the smallest execution unit in Kubernetes and encapsulates one or more applications. Pods are ephemeral by nature, that is, if a pod or the hosting node fails, Kubernetes can automatically create a new replica of that pod to continue operations and ensure service availability. Pods provide environmental dependencies, including persistent storage volumes such as storage that is permanent and available to all pods in the cluster and configuration data needed to run the container within the pod [18].

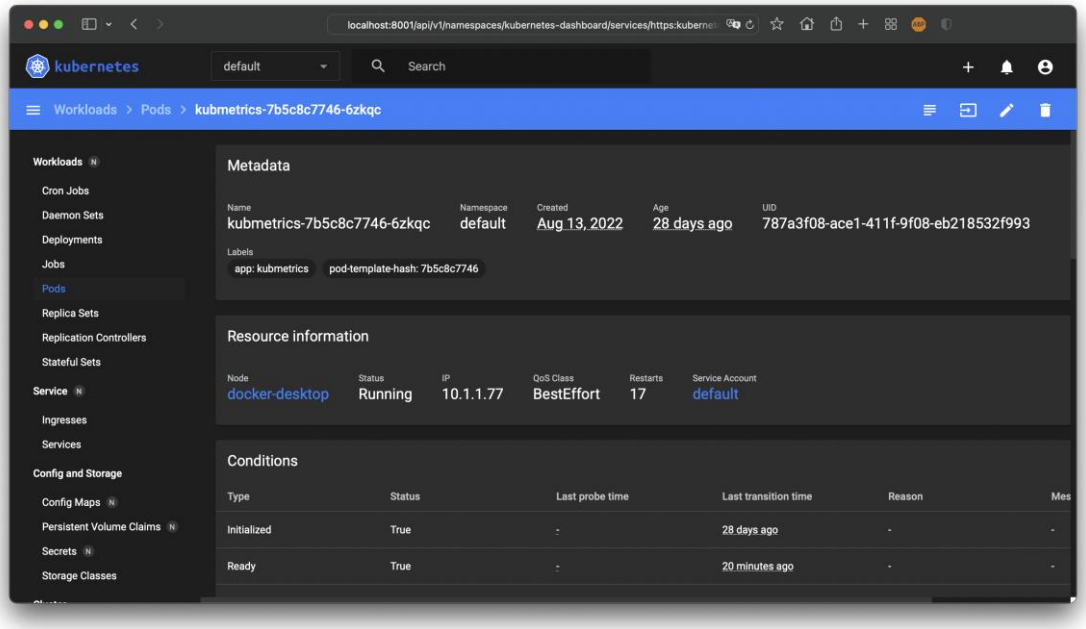

Figure 11: Kubernetes dashboard

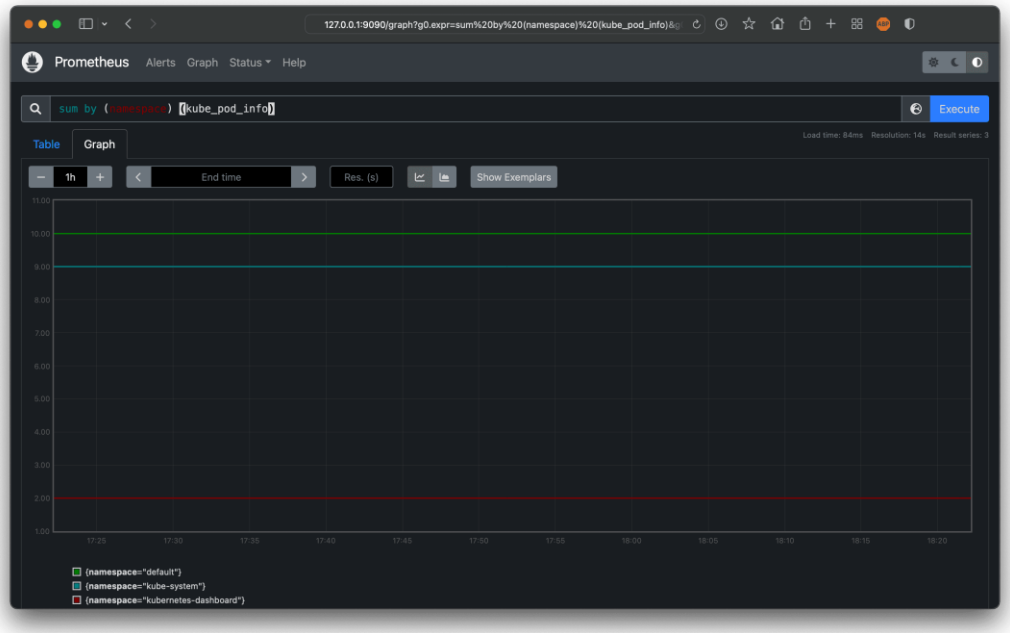

Figure 12: Total of pods

Technium Vol. 4, No. 9 pp.76-88 (2022)
ISSN: 2668-778X
www.techniumscience.com

Additionally another useful metric depicted in Fig. 13 illustrates all restarted pods in the namespace. This is an important metric concerning troubleshooting pods availability and uninterrupted operation. Fig. 14 shows the total usage within the container which also is highly important as a container overcommitted with resources will lead to system failure.

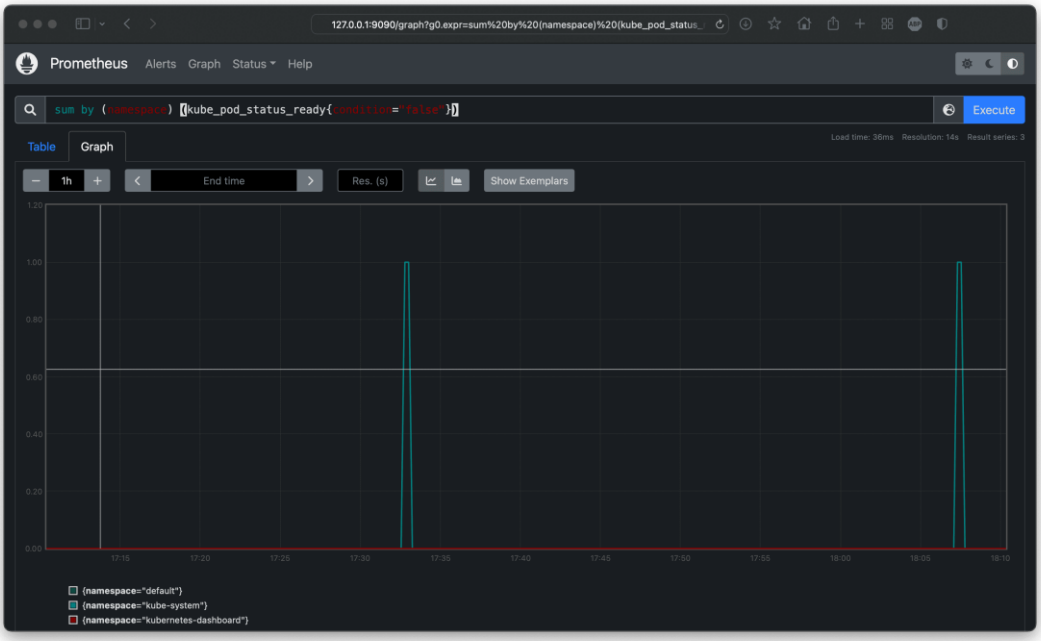

Figure 13: Total of restarted pods

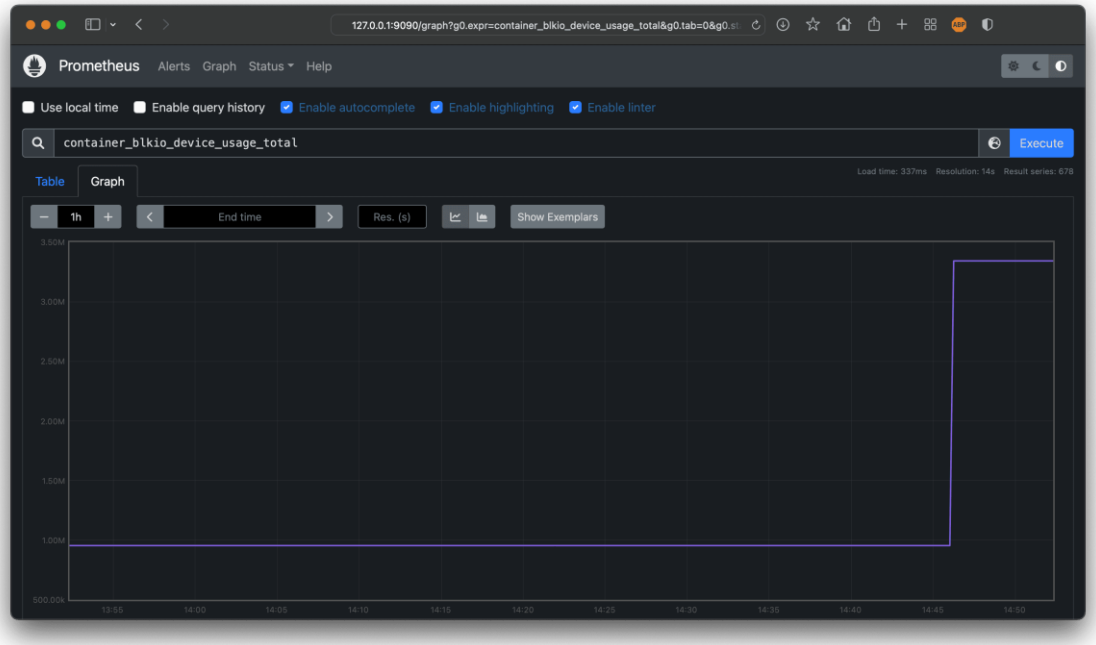

Figure 14: Container device usage

Technium Vol. 4, No. 9 pp.76-88 (2022)
ISSN: 2668-778X
www.techniumscience.com

3.2. *Visualization*

Visualizing the acquired metrics is carried out by Grafana, which is an open source software platform for statistical analysis and visualization of large volumes of data. It provides charts and alerts when connected to supported data sources such as Prometheus. In this particular part of our work, the contribution of Grafana is fundamental since it accounts for visualizing the metrics and time series data collected by Prometheus, which provides important information for the robust operation of the application. We also used Grafana in order to acquire information regarding CPU visualization and other resources. Fig. 15 illustrates the aforementioned metrics.

Fig. 16 illustrates the distribution of CPU usage among the processes of the namespace. Specifically, “kubepods” refer to all the deployed units in Kubernetes. Furthermore, there are several subpaths in “/kubepods”, such as “/kubepods/burstable” which are all Quality of Service (QoS) classes that Kubernetes assigns to pods regarding memory and CPU allocation. Fig. 16 and Fig. 17, respectively, show the subpaths of CPU and memory usage in detail.

A dashboard of Grafana is depicted in Fig. 18, showing total CPU and memory allocation for the whole cluster and provides multiple visualizations, such as gauges, alternate types of figures etc.

Figure 15: Visualization of resource allocation

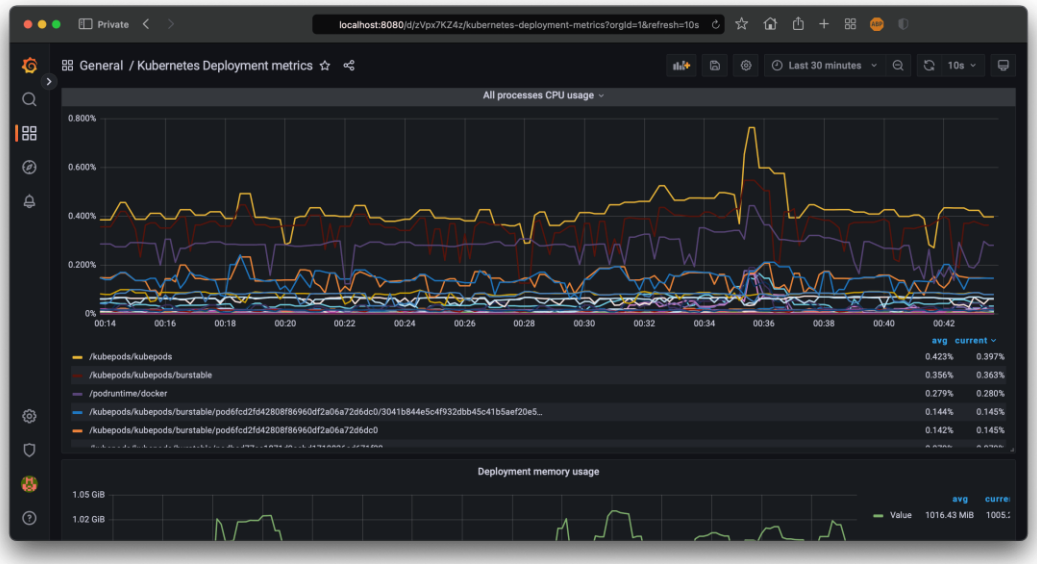

Figure 16: CPU utilization of all processes

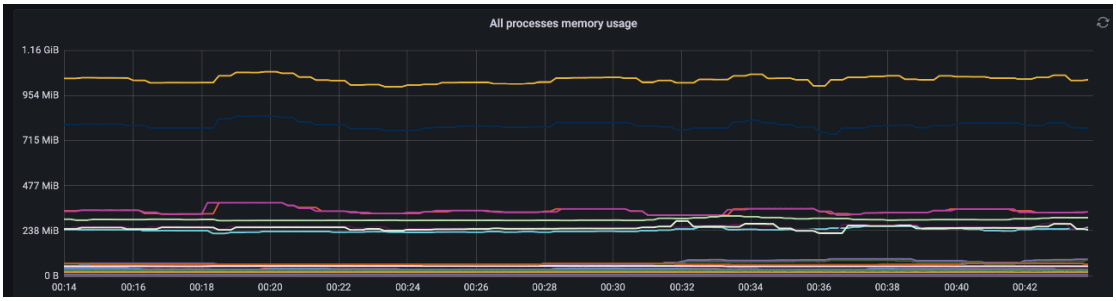

Figure 17: Memory utilization of all processes

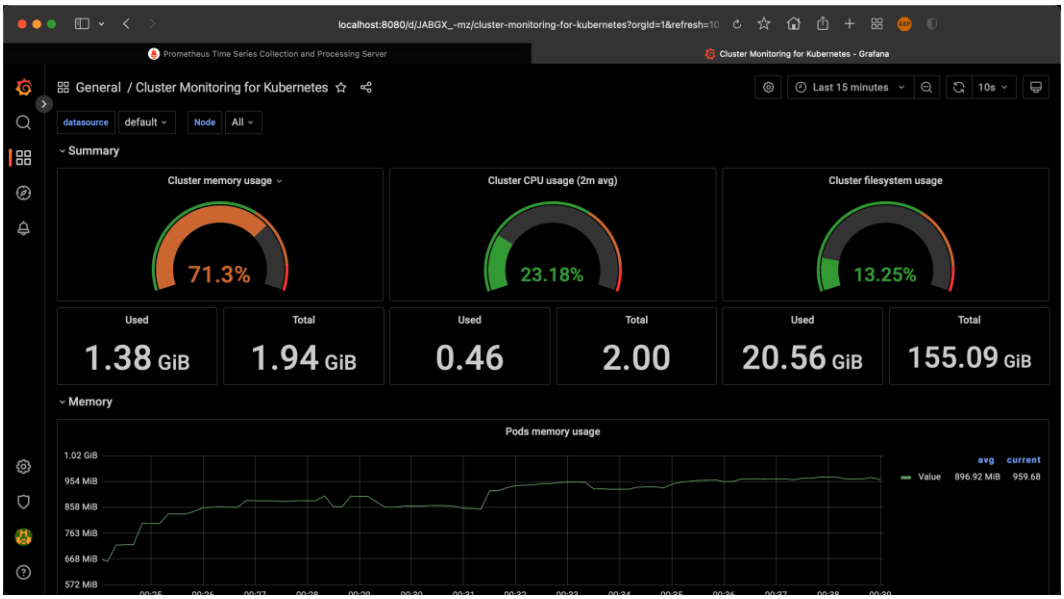

Figure 18: Grafana dashboard

4. Concluding remarks

We designed and implemented a cloud-based microservice for collecting, processing and visualizing crowdsensed air-pressure data generated by user mobile devices within a sort range of a user's current location. We also developed a proof-of-concept application where apart from the air-pressure measurement, a short-term weather prediction currently based only on local crowdsensed air pressure data is also presented. Our approach exploits the notion of locality, both in crowdsensed data and obtained short-term weather predictions and places emphasis on the quality of the user experience. Future plans include the design and implementation of additional microservices for obtaining and visualizing measurements for other factors affecting a short-term weather forecast like temperature, humidity, wind speed and direction. We also plan to use machine-learning algorithms for suggesting crowdsensing-based predictive models for short-term weather forecasting based on local crowdsensed data.

References

[1] P. Bellavista, G. Cardone, A. Corradi, L. Foschini and R. Ianniello. Crowdsensing in Smart Cities: Technical Challenges, Open Issues, and Emerging Solution Guidelines. IGI Global, 2015. DOI: 10.4018/978-1-4666-8282-5.ch015

[2] J.N.K. Liu, Y. Hu, Y. He, P.W. Chan, L. Lai. Deep Neural Network Modeling for Big Data Weather Forecasting. In Pedrycz, W., Chen, SM. (eds) Information Granularity, Big Data, and Computational Intelligence. Studies in Big Data, vol 8, Springer, 2015. https://doi.org/10.1007/978-3-319-08254-7_19

[3] R.K. Ganti, F. Ye; H. Lei. Mobile crowdsensing: Current state and future challenges. IEEE Communications Magazine, vol. 49, pp. 32–39, 2011.

[4] P. Mell and T. Grance. The NIST definition of cloud computing. Communications of the ACM, vol. 53, pp. 800-145, 2011. doi: 10.6028/NIST.SP

[5] F. Montori, L. Bedogni and L. Bononi. A Collaborative Internet of Things Architecture for Smart Cities and Environmental Monitoring. IEEE Internet of Things, vol. 5, pp. 592-605, 2018.

[6] C. Pahl. Containerization and the PaaS Cloud. IEEE Cloud Computing, vol. 2 pp. 24–31, 2015.

[7] ContainIQ. 9 Key benefits of using Kubernetes in 2022. url: https://www.containiq.com/post/benefits-of-kubernetes

[8] Docker. Develop faster. Run anywhere. url: https://www.docker.com

[9] Go. Building fast, reliable and efficient software at scale. url: https://www.go.dev

[10] Grafana Labs. Operational dashboards for your data here, there, or anywhere. url: https://www.grafana.com

[11] Kubernetes. Production-Grade Container Orchestration. url: https://kubernetes.io

[12] Kubernetes. Overview. url: https://kubernetes.io/docs/concepts/overview/

[13] mdn. CSS: Cascading Style Sheets. url: https://developer.mozilla.org/en-US/docs/Web/CSS

[14] mdn. Flexbox. url: https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox

[15] mdn. HTML: HyperText Markup Language. url: https://developer.mozilla.org/en-US/docs/Web/HTML

[16] OpenWeather. url: https://openweathermap.org

[17] Prometheus. From metrics to insight. url: https://prometheus.io

[18] vmware. What are kubernetes pods url: https://www.vmware.com/topics/glossary/