Dockerized Architecture for a Progressive Web App: An Italian Collocations Dictionary

Damiano Perri[1(✉)], Osvaldo Gervasi[1], Sergio Tasso[1], Irene Fioravanti[2], Fabio Zanda[2], Stefania Spina[2], and Luciana Forti[2]

[1] Department of Mathematics and Computer Science, University of Perugia, Perugia, Italy
{damiano.perri,osvaldo.gervasi,sergio.tasso}@unipg.it
[2] University for Foreigners of Perugia, Perugia, Italy
{irene.fioravanti,fabio.zanda,stefania.spina,luciana.forti}@unistrapg.it

Abstract. The aim of this work is to realise a scalable architecture for the provision of certain services to students and learners. During the course of a PRIN research project, an IT infrastructure is being set up for the consultation of Italian lexical combinations by students and learners interested in studying Italian. The system is proposed on an experimental basis, by means of a Progressive Web App that allows users to access and consult the developed contents independently of the operating system they use, without necessarily requiring the installation of an application developed in the native machine language of the operating system. This article explains in detail how the backend of the system is realised, with a focus on the development and testing phase. All the services that will be explained have been containerised via Docker, in order to guarantee portability, scalability of the system and future improvements.

Keywords: Mobile Developing · Artificial Intelligence · Progressive Web App

1 Introduction

The realisation of a computer architecture always requires a preliminary study, an analysis of the requirements, an estimate of the costs that will have to be borne in order to calibrate the initial investment and the implementation time with the expectations of those requiring the service and the final result to be obtained. This article presents a development methodology for the realisation of an infrastructure capable of scaling the computational power that can be delivered in order to guarantee the proper functioning of the service even with a high number of users that varies dynamically throughout the day, while maintaining a low cost.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15891, pp. 33–43, 2026.
https://doi.org/10.1007/978-3-031-97617-9_3

The proposed infrastructure has the task of managing the operation of an Italian lexical collocation dictionary [1] that must be used by students or language learners belonging to various levels of competence, and that can allow these people to consult in a simple and fast way without necessarily installing a native application on their device.

To achieve this result, it was first necessary to decide what type of product to create, in particular the choice fell on Progressive Web Apps (PWA). PWAs are web applications that offer a user experience similar to that of native applications, but are accessible via a browser and do not require separate installation. Developing using a technology of this type allows to drastically reduce the time necessary to create the application, since instead of referring to multiple programming languages, such as Java for Android and Swift for iOS, it is possible to use a single programming language, JavaScript, which allows to create multi-platform applications, simultaneously supporting Android, iOS, Windows, MacOS and Linux.

Furthermore, there are also other advantages related to the use of PWAs, such as being free from the policies of the official smartphone Stores, not necessarily having to go through an approval process for the publication of the application and the possibility of updating the application in real time without having to wait for users to install the new version. Costs are further reduced by the fact that it is not necessary to purchase the license for the publication of the application, which is instead mandatory for native applications. The article is organized as follows: Sect. 2 presents an analysis of the current scientific literature that addresses similar topics to the one proposed in this article, Sect. 3 describes the architecture of the system, within it you can find SubSect. 3.1 describes the proposed development environment, SubSect. 3.2 describes the backend for managing user requests. Section 4 summarizes the conclusions and future developments.

2 Related Works

The use of Docker is now considered a standard for the creation of applications to be delivered via cloud infrastructures [2,3]. Its characteristic is that it allows the creation of microservices, or small containers within which one or more processes are running. Even if containerized, experimental results show that applications such as Apache, nginx or MySQL maintain very high performance [4,5] and are perfectly comparable to the performance they would have if they were run natively on the machine. This is because Docker does not use complete virtualization but isolates at the namespace level to keep the applications' workspaces separate. This allows you to run multiple instances of the same container without necessarily having to create a virtual machine for each of them.

Furthermore, using Docker allows developers to create a scalable system, as it is possible to increase or decrease the number of containers running based on the number of requests that the system must handle [6–8].

Developing an application that can run on modern smartphones is a complex operation that can be done by following two main paths. The first is to create

the application natively, that is, using the specific programming languages of each operating system, such as Java or Kotlin for Android and Swift for iOS [9,10]. This first phase requires a considerable amount of time, as it is necessary to create two separate applications and the code that can be reused is very limited. Unless developers have very specific needs, such as the needs of using high computational power from the GPU, or access to particular sensors, it is possible to create a cross-platform application using a single programming language like as JavaScript [11,12]. In fact, there are frameworks, i.e. Ionic[1], React Native[2], Flutter[3] that allow you to write an application and see it running in real time within your browser [18,19].

Ionic for example allows Capacitor[4] or Cordova[5] integration. These two tools provide the capability to produce Hybrid Apps: in case a native application is needed, Cordova exports the HTML and JavaScript code into two separate projects, Android and iOS, all starting from a web application [22,23]. What must be taken into account is the publication phase of the applications, which takes a considerable amount of time, as the code submission phase, the review phase, and the cost of the licence to publish on the store must be considered.

3 Discussion

In this section, the system architecture is presented, which has been designed to be scalable and easily upgradable.

The infrastructure utilises the potential of the cloud, which allows you to pay a monthly fee based on the use of the provider's services. The formula is that of pay-per-use, i.e. you pay according to the services you use, and it is not necessary to buy a physical server, but you can use the virtual servers made available by the provider. Based on this, it appears essential to organise the system in such a way as to avoid waste and at the same time guarantee a quality service for end users. First of all, two subnets were created, the first being used for the development and testing of the application, the second being dedicated to production, i.e. the management of users accessing the service. Access to the network must take place via the infrastructure's central gateway, which not only sorts requests towards the production backend, but also implements appropriate firewall rules to guarantee network security.

The architecture that is described is represented in Fig. 1.

The description of the development environment is given in SubSect. 3.1, while SubSect. 3.2 describes the backend that handles user requests. All servers and services that will be described are based on Docker containers.

[1] https://ionicframework.com/.
[2] https://reactnative.dev/.
[3] https://flutter.dev/.
[4] https://capacitorjs.com/.
[5] https://cordova.apache.org/.

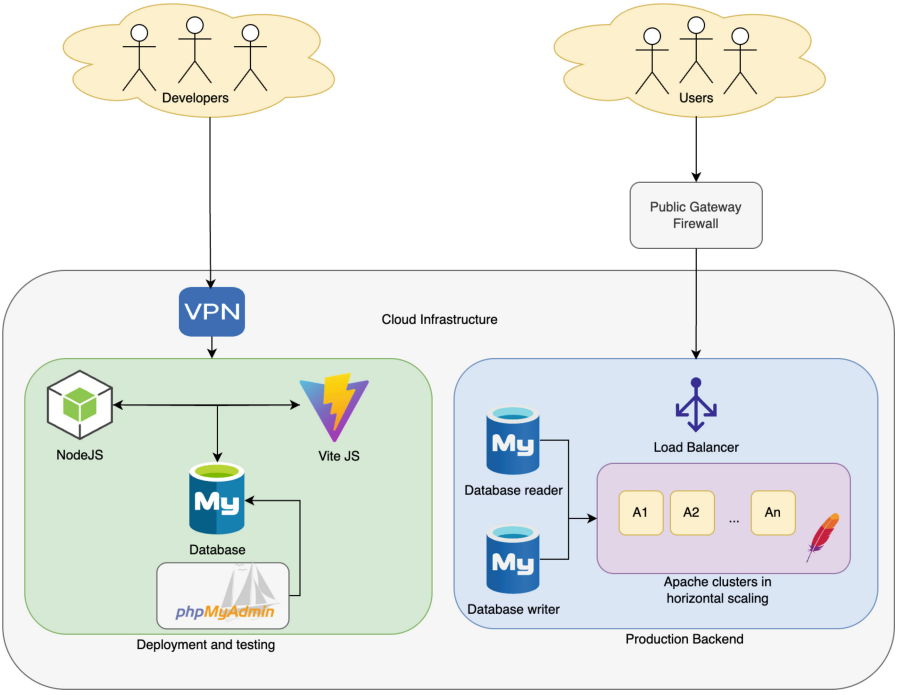

Fig. 1. System Architecture.

3.1 Development Environment

The test environment is fundamental to the realisation of an application, it must be separate from the production environment and in general must not be accessible to end users. Its purpose is to enable the development of applications and their testing in order to guarantee the absence of problems and malfunctions. The application envisages the creation of a dictionary that can be consulted via a web browser, The user connecting to the page must be able to type a word and will receive in output a list of lexical combinations that contain the word typed in, selecting the one of interest the user will have to obtain the information he or she requires. To solve this problem, first of all a database is required, in our case a MySQL relational database was selected, which by design guarantees a very high speed in the consultation phase and the possibility of making useful queries using SQL syntax.

The container containing the database was built using the official MySQL image[6] [24]. Among the precautions necessary for its realisation, we mention the fact of not exposing port 3306, which is the default port for the MySQL database, outside of the test network, as it is not necessary for users to be able to access the database directly, in order to improve the security of the system.

[6] https://hub.docker.com/_/mysql.

A second container based on a phpMyAdmin[7] image had to be created to manage the database [25]. The functionalities provided by this container allow the database to be easily managed via a web page. In fact, developers can connect via a browser to the database administration platform and, after authenticating themselves with their credentials, will have access to the tables and contents of the database itself. To guarantee maximum system security, the connection is only made via VPN.

As far as the code development environment is concerned, a solution based on two docker containers was chosen, the first containing Node.js[8], Ionic and Ionic Capacitor, and the second containing the web server[9].

The first container aims to provide a development environment containing all the necessary packages for building JavaScript-based web applications that are also graphically faithful to the native design of the devices. This is ensured by the use of IONIC, which provides a series of graphical components, such as buttons, table styles, text styles, and many others, realised in such a way as to adapt dynamically to the device that is using the application. In this way, a user using an Android smartphone will have a different graphic appearance to a user using an iPhone, but the source code will remain the same. The implementation of this first container was carried out using a custom image. This was made possible by defining a Dockerfile containing the instructions needed to install the basic packages of NodeJS, Ionic and Capacitor; in addition to the basic libraries needed to run these applications.

The second container was realised using the official Vite[10] image. The use of a Vite server allows the development phase of the application to be greatly facilitated, since changes to the code are reported in real time on all devices that are connected to the server. Suppose we have an Android smartphone and a laptop and we connect both to the web page containing the code we are developing to create our application. As soon as we make a change to the source code, this will be reflected in real time on both devices, which will automatically reload the page and immediately show the new code. Thanks to this functionality, it is possible to test changes on several devices at the same time in a very short time.

The development environment, realised with the technologies described, allows services to be easily encapsulated, packages and libraries used by the applications to be easily updated, allows easy backup and roll back of the images that manage the containers, and allows deployment to be performed quickly and easily. In addition, the possible addition of further docker containers, e.g. for testing a possible web server, does not require any substantial changes to the infrastructure, as docker containers are by nature independent of each other.

[7] https://hub.docker.com/r/phpmyadmin/phpmyadmin.
[8] https://hub.docker.com/_/node.
[9] https://vitejs.dev/.
[10] https://hub.docker.com/r/vitejs/vite.

3.2 The Backend

The purpose of the production environment is to handle the requests of users connecting to the service. It is crucial that this portion of the infrastructure is able to deliver the required services quickly and without errors. The realisation of the production infrastructure adopts an approach derived from the High Availability theory, which involves the creation of several instances of the same service, so as to guarantee continuity of service even in the event of failure of one of them.

As for the database, this was configured using a MySQL cluster [26,27]. A cluster is composed of several nodes, in the case presented here we have a total of two nodes of which one will have the task of performing reads and writes, while the other will only have the task of performing reads. In other words, two identical MySQL containers are created, and they are put in communication and synchronisation with each other. This means that when data is updated on the main node, it is replicated in real time on the secondary node. The primary node is the one that can perform the writes, while the secondary node is the one that can only perform the reads. Such an infrastructure is robust in that a failure of one of the two nodes does not compromise the service, as the other node is able to continue providing the required services. In the event of a failure of the primary node, the secondary node can be promoted to the primary node and thus be able to perform both writes and reads to the database. This type of approach also allows upgrades to be performed safely without compromising service delivery, since in the event that there is a need to shut down the main node to upgrade libraries and packages, the secondary node will continue to provide the services required by users. Having multiple nodes also allows user requests to be distributed, so since the workload related to readings is distributed between the two nodes, there is an improvement in overall system performance.

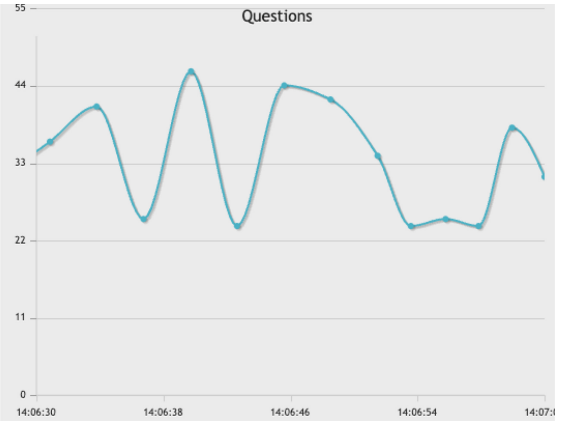

Fig. 2. Graph of queries received by the database cluster in real time.

As far as the web page accessed by users is concerned, this is managed by a cluster of Apache containers [28]. The Apache containers are all identical to each other, as they are generated from the same docker image. The cluster can dynamically adapt the number of containers according to the number of requests the cluster receives. The greater the number of user requests, the greater the number of nodes. This ensures that low running costs are maintained at times of low load. For example, at night, when there are few users connecting to the system, the number of Apache containers will be reduced to a minimum, while during daylight hours, when there are many users connecting to the system, the number of Apache containers can be increased to ensure a fast and error-free service. This technique, also known as horizontal scalability, can be performed while the system is switched on and is completely transparent to users.

Access to the data contained in the apache cluster is via a server a load balancer [29] whose task is to sort requests evenly distributed among the various nodes using the Round Robin algorithm [30,31].

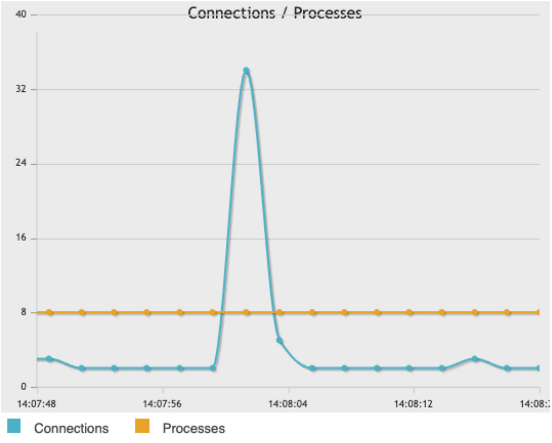

Fig. 3. Graph of connections received by the database cluster in real time.

Upstream of the infrastructure is the gateway, which not only provides the access port to the system, but also manages the firewall rules to guarantee network security. These rules include, for example, those for denying connections to unauthorised ports. This infrastructure is capable of delivering user services and also allows it to be further expanded should the numbers of those who wish to use the dictionary increase as it will always be possible to increase the number of nodes within the Apache cluster and the MySQL cluster. In addition, if the dictionary and its functionalities were to become extremely more complex in the future, it might also be possible to implement vertical scalability, i.e. to increase the hardware resources of the already existing nodes in order to guarantee ever higher performance.

Figures 2, 3 and 4 show the graphs of the queries, connections and data usage received by the database cluster in real time. The graphs were obtained by means of phpmyadmin, which provides many functionalities: in addition to managing the contents of the database, it is also possible to monitor its utilisation in real time and to obtain vital statistics to understand whether there are user peaks or problems to be analysed. It is also important to note that PhpMyAdmin is a free and open source software maintained by an active user community on GitHub[11].

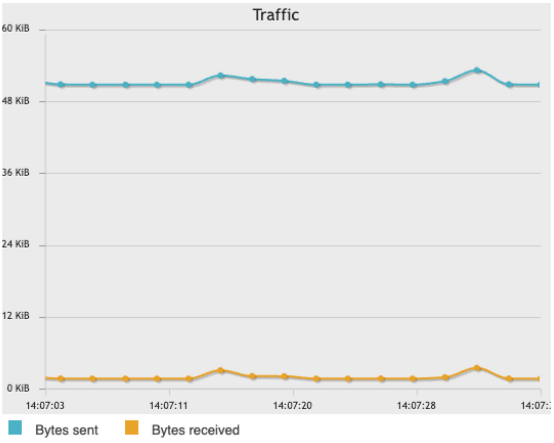

Fig. 4. Graph of data usage in real time.

4 Conclusions

This article presents a scalable architecture for developing and hosting a dictionary of Italian lexical collocations, which can be provided to Italian language students and learners.

Scalable architecture is now essential to reduce management costs and guarantee a correct user experience. It allows the computational power made available by the IT infrastructure that manages the service to be dynamically adapted based on the number of users connected to the system at the same time, thus avoiding a waste of resources. Using Docker both for the development phase and the creation of the backend was fundamental: from the point of view of IT security, it allowed the application to be broken down into multiple microservices, each of which was containerized and, therefore, isolated from the others, preventing a possible cyber attack from compromising the entire system. Furthermore, the use of docker allows updates and improvements to be made to the system by creating new images, which, in the circumstance that these have problems, can be replaced by restoring the previous images.

[11] https://github.com/phpmyadmin/phpmyadmin.

The testing, experimentation and improvement phase is still in progress; future improvements may be integrated into the system, also analyzing the user feedback that will arrive over the next few months, but for the moment it is possible to affirm that the first version of the proposed infrastructure described in this article is able to provide the requested service and thanks to its structure it is easily expandable and improvable.

Acknowledgements. The research has been funded by PRIN 2022 - PNNR M4.C2.1.1. - ERC secotr SH4, title "*DICI-A Dizionario delle Collocazioni Italiane per Apprendenti A Learner Dictionary of Italian Collocations*". Financed by the European Union NextGenerationEU. CUP J53D23008060006.

The authors thank Grammarly Inc., Quillbot Inc., Deepl SE, and Google LLC for using their platforms to improve the concepts they wanted to formulate in English.

References

1. Perri, D., Fioravanti, I., Gervasi, O., Spina, S.: Combining Grammatical and Relational Approaches. A Hybrid Method for the Identification of Candidate Collocations from Corpora, pp. 138–146 (2024). https://aclanthology.org/2024.mwe-1.18/
2. Garg, S., Garg, S.: Automated cloud infrastructure, continuous integration and continuous delivery using docker with robust container security. In: 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), pp. 467–470 (2019). https://doi.org/10.1109/MIPR.2019.00094
3. Antonova, V.M., Egorov, M.A., Blinov, V.P., Malikova, E.E., Malikov, A.Y.: Studying the principles of infocommunication network virtualisation using the docker platform. In: 2024 Systems of Signals Generating and Processing in the Field of on Board Communications, pp. 1–5 (2024). https://doi.org/10.1109/IEEECONF60226.2024.10496739.
4. Kithulwatta, W.M.C.J.T., Jayasena, K.P.N., Kumara, B.T.G.S., Rathnayaka, R.M.K.T.: Performance evaluation of docker-based apache and nginx web server. In: 2022 3rd International Conference for Emerging Technology (INCET), pp. 1–6 (2022). https://doi.org/10.1109/INCET54531.2022.9824303
5. Perri, D., Simonetti, M., Tasso, S., Ragni, F., Gervasi, O.: Implementing a scalable and elastic computing environment based on cloud containers. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12949, pp. 676–689 (2021). https://doi.org/10.1007/978-3-030-86653-249
6. Wang, W.: Research on using docker container technology to realize rapid deployment environment on virtual machine. In: 2022 8th Annual International Conference on Network and Information Systems for Computers (ICNISC), pp. 541–544 (2022). https://doi.org/10.1109/ICNISC57059.2022.00112
7. Nurwarsito, H., Sejahtera, V.B.: Implementation of dynamic web server based on operating system-level virtualization using docker stack. In: 2020 12th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 33–38 (2020). https://doi.org/10.1109/ICITEE49829.2020.9271710

8. Kumar, K., Kurhekar, M.: Economically efficient virtualization over cloud using docker containers. In: 2016 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pp. 95–100 (2016). https://doi.org/10.1109/CCEM.2016.025
9. Golhar, R.V., Vyawahare, P.A., Borghare, P.H., Manusmare, A.: Design and implementation of android base mobile app for an institute. In: 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), pp. 3660–3663 (2016). https://doi.org/10.1109/ICEEOT.2016.7755391
10. Hauser, D.: Test-Driven iOS Development with Swift: Write Maintainable, Flexible, and Extensible Code Using the Power of TDD with Swift 5.5. Packt Publishing (2022). ISBN 180323248X
11. Koram, N., Garg, R.: Review on mobile app development: tools and techniques. In: 2023 IEEE World Conference on Applied Intelligence and Computing (AIC), pp. 260–266 (2023). https://doi.org/10.1109/AIC57670.2023.10263908
12. Pinto, C.M., Coutinho, C.: From native to cross-platform hybrid development. In: 2018 International Conference on Intelligent Systems (IS), pp. 669–676 (2018). https://doi.org/10.1109/IS.2018.8710545.
13. Babich, L.V., Svalov, D.A., Smirnov, A.L., Babich, M.V.: Neurorehabilitation control application: development using ionic framework. In: 2018 26th Telecommunications Forum (TELFOR), pp. 420–425 (2018). https://doi.org/10.1109/TELFOR.2018.8611880
14. Yang, Y., Zhang, Y., Xia, P., Li, B., Ren, Z.: Mobile terminal development plan of cross-platform mobile application service platform based on ionic and cordova. In: 2017 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), pp. 100–103 (2017). https://doi.org/10.1109/ICIICII.2017.28
15. Dabit, N.: React Native in Action: Developing iOS and Android Apps with JavaScript (2019). ISBN 1617294055
16. Novac, O.C., Novac, C.M., Ciora, B., Gordan, C.E., Gordan, M.I., Bujdos'o, G.: The rise of mobile development: a comparison between Ionic and Flutter. In: 2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), pp. 1–10 (2022). https://doi.org/10.1109/ECAI54874.2022.9847460
17. Lohani, D.: Taking Flutter to the Web: Learn how to build crossplatform UIs for web and mobile platforms using Flutter for Web (2022)
18. Behl, K., Raj, G.: Architectural pattern of progressive web and background synchronization. In: 2018 International Conference on Advances in Computing and Communication Engineering (ICACCE), pp. 366–371 (2018). https://doi.org/10.1109/ICACCE.2018.8441701
19. Gowri, S., Pappa, C.K., Tamilvizhi, T., Nelson, L., Surendran, R.: Intelligent analysis on frameworks for mobile app development. In: 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT), pp. 1506–1512 (2023). https://doi.org/10.1109/ICSSIT55814.2023.10060902
20. Bosnic, S., Papp, I., Novak, S.: The development of hybrid mobile applications with Apache Cordova. In: 2016 24th Telecommunications Forum (TELFOR), pp. 1–4 (2016). https://doi.org/10.1109/TELFOR.2016.7818919
21. Vladimirovich, B.M., Andreyevich, S.D., Vladimirovna, B.L., Leonidovich, S.A.: Development of the mobile application for control of the neuro-electrostimulator of type 'SYMPATHOCOR-01' using apache cordova. In: Ural Symposium on Biomedical Engineering. Radioelectronics and Information Technology (USBEREIT), 2018, pp. 72–75 (2018). https://doi.org/10.1109/USBEREIT.2018.8384553

22. Malavolta, I.: Web-based hybrid mobile apps: state of the practice and research opportunities. In: 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 241–242 (2016). https://doi.org/10.1145/2897073.2897133
23. Uniyal, S.P., Joshi, K., Singh, V.K., Aggarwal, A., Chhabra, G., Kumar, A.: Comparative analysis of app size variations between react native and apache cordova powered android applications. In: 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), pp. 1697–1702 (2023). https://doi.org/10.1109/ICAISS58487.2023.10250551
24. Malhotra, R., Bansal, A., Kessentini, M.: Vulnerability analysis of docker hub official images and verified images. In: 2023 IEEE International Conference on Service-Oriented System Engineering (SOSE), pp. 150–155 (2023). https://doi.org/10.1109/SOSE58276.2023.00025
25. Gabarro, S.A.: Using PhpMyAdmin. In: Web Application Design and Implementation: Apache 2, PHP5, MySQL, JavaScript, and Linux/UNIX, pp. 151–158 (2007). https://doi.org/10.1109/9780470083963.ch12
26. Tummalapalli, S., Machavarapu, V.R.: Managing MySQL cluster data using Cloudera impala. Procedia Comput. Sci. **85** (2016). International Conference on Computational Modelling and Security (CMS 2016), pp. 463–474. ISSN 1877-0509. https://doi.org/10.1016/j.procs.2016.05.193
27. Perri, D., Simonetti, M., Gervasi, O.: Deploying efficiently modern applications on cloud. In: Electronics (Switzerland) **11**(3) (2022). All Open Access, Gold Open Access, Green Open Access. https://doi.org/10.3390/electronics11030450, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85123753434&doi=10.3390%2felectronics11030450&partnerID=40&md5=48cb887b0ef5347839dfea38121158df
28. Tasso, S., Cal'o, S., Gervasi, O., Perri, D.: Integration of an artificial intelligence model into a smartphone flutter application to solve a live image classification problem. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 14815, pp. 39–55 (2024). https://doi.org/10.1007/978-3-031-65154-03, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85200998470&doi=10.1007%2f978-3-031-65154-03&partnerID=40&md5=4ee5be56b8dcb78d8ebf534f6ad4b9cd
29. Rahman, M., Iqbal, S., Gao, J.: Load balancer as a service in cloud computing. In: 2014 IEEE 8th International Symposium on Service Oriented System Engineering, pp. 204–211 (2014). https://doi.org/10.1109/SOSE.2014.31
30. Pramono, L.H., Buwono, R.C., Waskito, Y.G.: Round-Robin algorithm in HAProxy and Nginx load balancing performance evaluation: a review. In: 2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), pp. 367–372 (2018). https://doi.org/10.1109/ISRITI.2018.8864455
31. Kaur, S., Kumar, K., Singh, J., Ghumman, N.S.: Round-robin based load balancing in software defined networking. In: 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 2136–2139 (2015). https://ieeexplore.ieee.org/document/7100616