A Self-Hosted Approach to Automatic CI/CD Using Open-Source Tools on Low-power Devices

Petr John, Kristýna Zaklová, Juraj Lazúr, Jiří Hynek and Tomáš Hruška
Faculty of Information Technology, Brno University of Technology, Brno, Czechia
Email: {ijohn, zaklova, ilazur, hynek, hruska}@fit.vut.cz

***Abstract*—The process of software development has widely adopted Continuous Integration and delivery (CI/CD), offering a comprehensive approach to streamlining the development process. Both features are commonly offered by third-party services or cloud-hosted environments, like Amazon AWS or GitHub. While these offerings are widely available and perfect for large projects and companies, they may not be suitable for smaller projects. Factors like the desire to maintain code and deployment locally or cost considerations can drive the search for alternative solutions. This paper presents an infrastructure mostly based on open-source services that address these specific needs. It is suitable for both development and the automatic deployment of small, locally deployed projects. The solution underwent six months of testing in a lab environment on a low-power device, namely the Raspberry Pi 4B, which was used to both build new versions of the software and host it. The solution leverages the GitHub self-hosted runner to build Docker images from the code directly on the target device. The images are then pushed into a local Docker Registry. This ensures that the code can be used on multiple architectures, namely the arm64, and amd64, by simply pulling an appropriate image that is cross-built thanks to Docker buildx. The built images can then be automatically updated to deploy or be deployed on different hardware. The solution is suitable for small projects or teams with constrained budgets thanks to the self-hosting of all the components, the low purchase price, and the power efficiency of low-power devices like the Raspberry Pi.**

***Index Terms*—Docker, CI/CD, Continuous Deployment, Raspberry Pi**

I. Introduction

While continuous integration and delivery began as an often discussed and not implemented concept [1] it quickly transitioned into a popular approach that stimulates both agile [2] and test-driven development practices [3]. Tools for this purpose evolved from basic scripts mentioned by Fowler and Foemmel [1] to fully-fledged solutions like the popular tool Travis CI with more than 380 thousand installations[1], Jenkins[2] or cloud-based solutions like GitHub Actions, GitLab CI/CD or the combination of Amazon Web Services (AWS) CodePipeline and CodeBuild among many others [4].

All of these tools can be used for both the testing of the created artifacts and the preparation of built and deployable software, ranging from built binaries to more complex artifacts like Docker images. Then, these artifacts can be used by developers to test different parts of their solution, they can be made available for download, or they can be used to immediately update the existing deployments.

The problem with current approaches to both CI/CD and deployments is often the cost or complexity of the used solutions. This occurs due to the need to repeatedly rebuild the entire solution, which can cost large companies like Google millions of dollars just in computation alone [5]. While some solutions may require no or simple setup from the developer, like cloud-based solutions or online hosting solutions. Online hosting solutions may be too restrictive, limiting the technologies that can be used, such as specific programming languages or certain types of databases. Cloud technologies can be too expensive for small teams. While a large number of cloud providers offer free tiers like AWS[3] and Google Cloud[4] These limits may prove too limited for the needs of the project.

The other option is to create the hosting solution *on-premise* and host the solution on the team's own infrastructure or online virtual machines or Virtual Private Servers (VPS). While there are many solutions to simplify the process of self-hosting the created solution like ISPConfig[5] or servers and extensions like *Nginx* and *Apache2*, the process may still prove to be time-consuming when the updates need to be performed repeatedly.

The aim of this paper is to present an infrastructure that is able to facilitate the need for CI/CD on small teams with limited budgets while reducing the manual effort that is required to both monitor and deploy the created project. This infrastructure aims to simplify both the development and the ability to quickly self-host and automatically update the created deployments through the use of open-source or free-to-use tools. These targets are accomplished by distributing the created artifacts as Docker images. This facilitates the possibility of easily distributing the building and hosting infrastructure, even with nodes that use different operating systems and processor architectures, without the need for a server or high-end components. This was demonstrated by using a single Raspberry Pi 4B device, which serves as both a representative of a low-power device and a device based on the less traditional ARM64 architecture.

This infrastructure empowers small teams to swiftly develop new software, seamlessly host, and automatically update deployments. It facilitates continuous integration of large appli-

[1]Home – Travis-CI https://www.travis-ci.com/

[2]Jenkins Project reports growth of 79% https://cd.foundation/announcement/2023/08/29/jenkins-project-growth/

[3]Amazon Web Services free tier: https://aws.amazon.com/free/.

[4]Google Cloud free tier: https://cloud.google.com/free.

[5]ISPConfig home page: https://www.ispconfig.org/.

cations in order to enable teams to achieve significant cost efficiencies with minimal upfront investment and sustained low costs over time. The approach was used to develop and deploy 3 research projects and several student projects at Brno University of Technology, Faculty of Information Technology in the time span of 6 months in lab conditions, and while the device allows only low performance, resulting in slow build especially when cross-building, it is more than balanced by its low power consumption.

II. Current Solutions

As briefly mentioned in Introduction, there already exists a large number of specialised tools for both the CI/CD process and the deployment of existing projects. The two common approaches to these problems are either outsourcing or self-hosting which come with their own advantages and disadvantages. Both approaches often utilize similar tools, which include the aforementioned Jenkins, Travis CI, and Docker, but may include other tools [6]. The additional tools include project management tools like Maven and npm, or tools that can be used to describe the infrastructure or the deployment like *Terraform*—infrastructure as code tool, *Kubernetes*—a container orchestration platform, or *docker-compose*—a Docker plugin allowing developers to define multi-container applications.

A. Outsourcing

In recent years, cloud providers like Google and AWS have gained more and more popularity, among the outsourced solutions, due to the possibility of both vertical and horizontal scaling, without the need for the improvement of local infrastructure. In some cases, this can be without upfront cost, either by scaling down the solution as needed or by providing the necessary tools through pay-as-you-go offerings. A large number of cloud solutions even provide free trials or free tiers with limits that either allow the user to use the services for free for (1) a limited time, (2) forever, but with restrictions (i.e. limiting the amount of traffic that can be sent or the amount or retention of data), or (3) provide a certain amount of free balance, that can be expended without charge. While all of these situations may result in the quick adoption of a certain technology, they can also result in a vendor lock-in. This has been already identified as a potential threat in the year 2011 [7] with some open-source creators even describing cloud solutions as *traps* [8]. Cloud providers on the other hand disagree with the opinion and view the effort needed to switch between cloud providers as necessary *switching costs* [9].

Currently, Amazon Web Services has been the most popular solution in recent years, having approximately 32% of the market share, with Microsoft Azure and Google Cloud Platform in pursuit, having 23% and 11% respectively according to the statistics of Michalowski [11] The market share trend can be seen in Figure 1. AWS environment provides a large amount of CI/CD-focused tools that can bolster the effectiveness of build/deployment pipelines by providing tools for managing the builds themselves (CodeBuild), managing

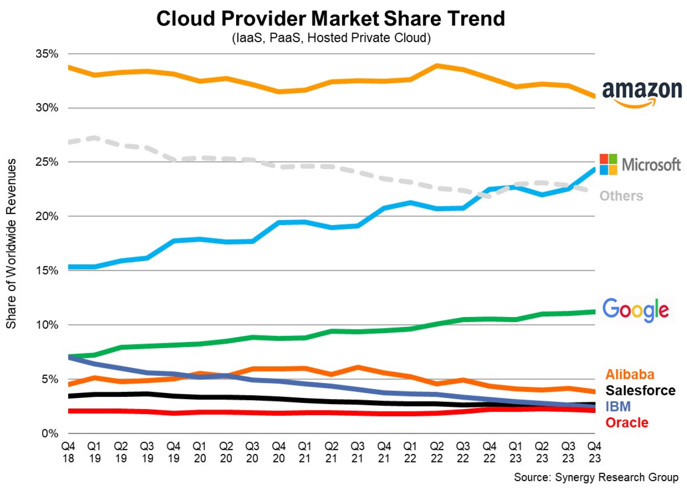

Fig. 1. Cloud Provides Market Share [10]. While Amazon AWS retains the highest market share, Microsoft's offering Azure exhibits the biggest growth.

and orchestrating multiple builds (CodePipeline) or caching dependencies and docker registry (CodeArtifacts and ECR respectively). Both Sinde et al. [12] and Poornalinga et al. [13] used the combination of GitHub for CI and AWS services for CD purposes. Sinde et al. [12] use the combination of GitHub Actions and Amazon EKS to automatically update a Kubernetes cluster, and Poornalinga et al. [13] propose a similar solution, running Jenkins for CI and AWS EC2 for deployment. Many other publications refer to similar CI/CD procedures [14], [15], [16], [17]. Also, cloud environments have the benefit of providing self-healing and self-adapting web-based applications as outlined by Magalhaes et. al. [18] in their SHõWA framework.

Another common approach to outsourcing the hardware requirements is the use of Virtual Private Servers (VPS). VPS are often created by either Docker or LXD containers [19], [20] and allow the provider to create a virtual machine for each customer, thus providing them with dedicated computer resources without the need to buy the bare-metal solution for each customer separately [21]. Thanks to this, the user can use the system in the same way as a traditional bare-metal server, i.e. the system allows the user to use root privileges and install any dependencies that may be needed for the deployment. This leads to reduced costs for the customers, as the same bare-metal server can be reused and the user is billed for only a fraction of the running cost based on the resources that are allocated for his exclusive use. This approach is provided by services like OVHcloud[6] or Hostinger[7], but the aforementioned cloud solutions can be utilized in a similar manner, mainly by using cloud-specific virtual machines like Amazon EC2 or Google Cloud Run.

The last option, but not the least, is the utilization of Web Hosting Panels [21]. These panels include tools like

[6]OVHcloud homepage https://www.ovhcloud.com.

[7]Hostinger VPS hosting offerings https://www.hostinger.com/vps-hosting.

ISPConfig [22], Zpanel, and Virtualmin [23], and provider-specific solutions like GoDaddy[8]. They allow the customer to deploy only code in a specified programming language using the available services. Another major difference between Web Hosting Panels and VPS is that the resources are typically shared between multiple users. While this may initially be perceived as a disadvantage it may lead to lower costs for web pages that don't require as much traffic.

B. *Self-hosting*

The second option for both CI and CD is self-hosting of the related components. While this may be cheaper in some situations (i.e., if low-power devices are used, or the server is hosted on an always-on device like a Network Attached Storage) it provides other difficulties that may not be apparent at first glance. In many cases, the potential user needs to buy hardware used for both CI/CD and pay running costs (i.e. electricity, necessary hardware replacement) and other services, like internet rentals or public IP addresses. On the other hand, the user has the entire solution in their control and can modify the environment to fit their use case exactly.

While this provides the users with the most control over all of the aspects of the environment, they may face difficulties with more advanced tasks, like resource sharing, scaling vertically, or managing the environment. In those cases, it may be beneficial to set up a web hosting panel like the aforementioned ISPConfig. While this may help with some basic issues, it may be beneficial to build a self-hosted and provisioned cloud instead. This can be represented by tools like PlanetIngnite, which aims to create a decentralized and self-assembling cloud [24]. There also exist approaches that can turn multiple machines into a private cloud like Cells [25]. These approaches or tools like Docker, Terraform, or Kubernetes can help the user to reduce the human investment over the lifetime needed to successfully deploy and maintain the package, as they can provide a model-based approach to this problem [26], which leads to a reduction in complexity in the long term when compared to both manual or script-based approaches. While the model-based approaches may require a higher up-front investment and a steeper learning curve, they prove to be better long-term options than even the language-specific approaches.

Although the cost of traditional server computers can be high, there is always the option to use either older desktop computers or lately off-the-shelve single board computers like the popular Raspberry Pi. This approach has already been attempted in previous years in such uses as web hosting [27], [28] or as data storage [29]. These usages did exist before they were limited to simple web hosting in the past, primarily due to the low amount of RAM and the limitation of a 32-bit architecture. In recent years, the hardware improved in both these aspects. This makes the newer devices much more capable, and it is now possible to use them for much more advanced purposes.

[8]GoDaddy Web hosting offering https://www.godaddy.com/en-uk/hosting/web-hosting.

III. Proposed Infrastructure

We required a solution that would satisfy multiple requirements.

1) Firstly, the solution should allow the general public to use the created applications (i.e. be publicly available on the internet, without the use of specialized software like VPN).
2) Secondly, it should allow us to quickly iterate over new versions of the applications with the least amount of effort using automatic CI/CD practices.
3) Thirdly it should prevent vendor lock-in or any restriction on technologies/approaches that should be used.
4) And lastly, it should be cost effective, requiring both a small upfront investment and long-term maintenance costs.

With respect to these requirements, we propose a solution primarily based on open-source technologies and GitHub[9]. We chose GitHub over open-source alternatives like GitLab due to our familiarity with the software, its large community, and security. However, it would be possible to move to any such software that supports self-hosted runners.

All of the self-hosted applications are provided in the form of Docker containers, which allows the infrastructure to be both easily deployed and scaled to more devices if needed. GitHub is used as a code source and it is also responsible for orchestrating both CI/CD actions. This creates a compromise between a fully self-hosted solution and an outsourced one. Due to the low upfront investment being a requirement, we opted for the use of the Raspberry Pi 5B single-board computer. With its launch price of $80 MSRP, it proved to be an ideal choice.

The CI/CD actions are picked up by any of the currently available runners, which are responsible for both building the docker images of the application and running any tests that may be associated with it. Then, the test results get sent back to GitHub, where they can be visualized to the developer. Built Docker images get first pushed into a Docker registry and then picked up by the Deployment Server, which automatically updates the deployed environment. In our case, the Docker registry was deployed alongside the application deployments on the deployment server. The workflow can be seen in Figure 2.

This segregation of responsibilities divides the solution into three main components: the code source, deployment server, and build runners. Each component can be deployed independently. The composition, responsibilities, and interactions of these components are illustrated in Figure 4.

A. *Build Runners*

Build runners are implemented using GitHub self-hosted runners. These runners can be either deployed to the same device as the one responsible for hosting the projects or to a stand-alone device. While GitHub allows runners to be

[9]GitHub self-hosted runners https://docs.github.com/en/actions/hosting-your-own-runners.

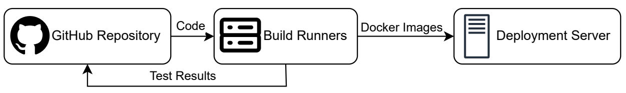

Fig. 2. Simplified Build and Test Workflow.

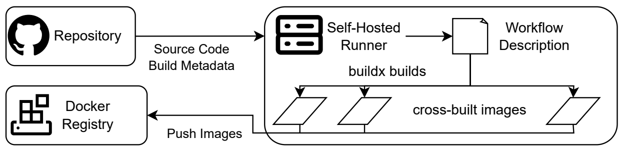

Fig. 3. Build procedure.

added to repositories and organizations, we opted for both approaches. This dual strategy is practical because it enables the maintainers to assign a runner to a specific repository or organization. Additionally, some projects used during testing were located under different organizations or repositories, making it necessary to have the flexibility to assign runners at both levels. While it would be more practical to change this in the future, the diversity of our projects currently doesn't allow this.

The main advantage of the self-hosted runners over their standard counterparts comes in the maximum amount of available minutes. GitHub runners can be used for free only for a limited amount of minutes, currently 2000 minutes per month for free accounts. While this number may be suitable for many projects, larger or more time-consuming tasks like cross-building for multiple target platforms or frequent builds/tests can quickly lead to the exhaustion of this limit. The cost might ramp up quickly since GitHub offers a *pay as you go* offering. The orchestration is then implemented by using GitHub's integrated Action workflows. We chose the GitHub-specific workflow description language, but it is possible to use Jenkins instead, thus reducing the dependence on GitHub.

Then, the triggered actions are responsible for both sending the source code and monitoring the self-hosted runners and build progress. Our typical workflows contain the build instructions for creating new Docker images for the latest update of the built applications. As the target hosting platform is a Raspberry Pi 4B, the image needs to be built for the ARM64 architecture, but we also aim to make the same built image compatible with different architectures based on the project, e.g., the more common AMD64 architecture. This is accomplished by cross-building the images using *buildx*, which is able to create a Docker manifest that contains references to all selected architectures. This allows the target device to pull the correct image without the need for changing the Docker tag. The images are then automatically pushed to a self-hosted Docker registry, which acts as the center point of the entire architecture. This allows all of the machines to pull the containers from a single source. The build procedure can be seen in Figure 3.

B. Deployment Server

The deployment server runs Docker containers based on both publicly available images and the images of the applications being developed. It is responsible for making these applications publicly accessible, which requires a fast internet connection and a public IP address. Each deployment server needs to have access to the Docker registry because otherwise, it would be impossible to retrieve the images. Each deployed application currently uses a *docker-compose* description, which outlines the list of services that are used. This approach simplifies the deployment process significantly. A new deployment can be initiated by transferring the necessary files, such as the *docker-compose* configuration and other run-specific files (e.g., secrets, caches), to a different server. After logging into the registry, a single command like "*docker compose up -d*" is all that's needed to bring the new deployment online.

While this setup meets the requirements for continuous integration, it falls short of achieving continuous delivery. To update a container-based deployment, all containers must be recreated using the new images. This easy process can be quickly automated by either using a simple *Cron* script or dedicated tools like *Watchtower*[10] This open-source project pools the registries containing the used images. It also offers fine-grained control, allowing developers to explicitly label services to be skipped during updates and providing login integrations for registries hosted on cloud environments like AWS.

Watchtower satisfies the continuous delivery as new updates are automatically deployed via the update of the deployment containers. While Docker can recover from some errors by automatically restarting the deployed servers it is necessary to monitor both the state of the device and the state of essential deployed services. This is accomplished by the combination of *Netdata* and *Uptime Kuma* services. *Netdata* monitors the device status and records metrics like the CPU and memory usage. *Uptime Kuma* on the other hand is used for monitoring both APIs and web-based applications and sending notifications to instant messaging applications in case any of the deployed stops responding.

The combination of these services creates a robust, self-healing, and self-monitored solution, which, however, still lacks one very important property—security. While it would be possible to maintain HTTPS certificates and access control on the level of individual publicly available services, it proved beneficial to deploy a single *Nginx* reverse proxy that handles both the SSL certificates and the server mappings. This vastly simplifies the deployment process as it separates responsibility between the team member who is responsible for the maintenance of the reverse proxy and the rest of the team members, who can focus on the application itself. In the test deployment, the Let's Encrypt certification authority in combination with *Certbot* was used to generate the certificates. A simple *Cron* rule can be used to renew the certificates before they run out.

[10]Watchtower homepage: https://containrrr.dev/watchtower/.

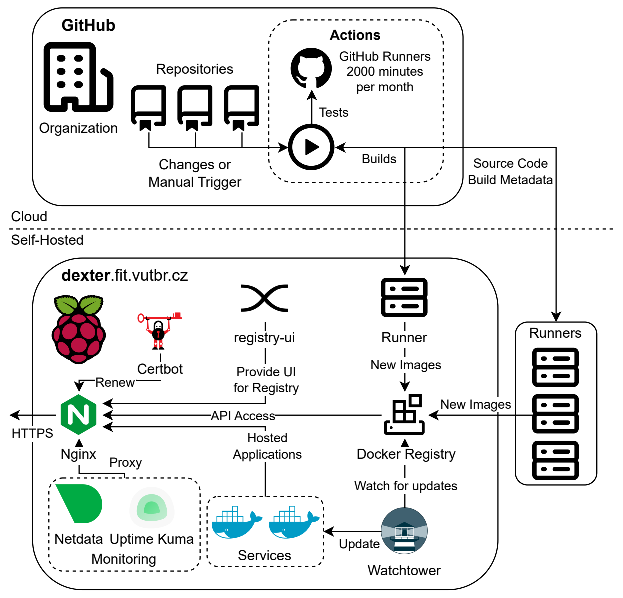

Fig. 4. The proposed Infrastructure: The GitHub architecture hosts source code and orchestrates the build procedure. Quick actions like small tests are evaluated directly within the GitHub architecture, while self-hosted runners handle the remaining tasks. The runners primarily cross-build new images and push them to a Docker registry located on the deployment server, Dexter.

The entire infrastructure with all of the deployed services is shown in Figure 4. The comprehensive technical specifications and configurations files of the infrastructure are documented in a dedicated tutorial, which can be accessed via the following resource[11].

IV. Usage Scenarios

The infrastructure was demonstrated on a research group project server[12] to build and host both the results of the research group members and to demonstrate the result of successful theses of students. Below are some of the examples of the applications that were hosted on the server.

A. Home IoT solution

A solution for a Smart Home installation[13] was developed during a master's thesis [30] and is hosted on the development server. It comprises multiple services, such as *Chirpstack* responsible for the digestion of messages from IoT devices, *Mosquitto*, which acts as a central broker, providing the ingested messages to the rest of the architecture, the NoSQL database system *InfluxDB* responsible for the retention and efficient querying of time-series data and *PostgeSQL* which is used as a metadata store. A combination of a Go-based subscriber is then responsible for sending the data to a backend application based on *Express.js*, which transforms the data into an internal representation and stores them in the selected database systems. Then, the user can use a frontend application based on Angular to visualize the state of the system by the use of specialized dashboards.

This case study showed that even a low-power device like the Raspberry Pi 4B can handle a small home installation of 16 *Zigbee* and 1 *LoRaWAN* device with no issues, even when specialized databases like *InfluxDB* are used. While this might seem like a small installation due to the *Chirpstack* deployment it necessitated the use of 14 containers. Docker simplifies the installation process significantly due to its ability to handle various technologies seamlessly. Furthermore, Docker facilitates easy updates and reinstallation of the setup, ensuring the system remains current and functional with minimal effort. This may be primarily helpful in cases when the deployment needs to be moved to a deployment server with higher performance.

B. System for Analysis of City Council Decision-Making

The system called *Zastupko*[14] aims to increase city councils' transparency by visualizing council meeting results and providing analysis tools primarily aimed at the general public [31]. It was designed as a traditional information system with a client-server architecture. Data is stored in relational databases, and there is always one central database and then one for each municipality. In the basic version, the system only provides data presentations that cannot be edited and does not require any user accounts.

The case study involved deploying the application in several instances, each targeting different users. Each of these deployments required three containers—one for the back-end, one for the front-end, and one for the database management system (DBMS). Two instances were deployed as production, while one needed an extra container due to the separate landing page. Four instances were deployed as development, enabling independent feature development. For example, one instance was dedicated to a front-end redesign, while another focused on adding management features. This approach isolated changes in their respective deployments, simplifying testing new features independently.

In total, six instances of the system consisting of 19 containers were created. All of them have a similar configuration. Creating each subsequent deployment only required copying the existing configuration and changing the image used. The approach used made the whole process very simple. The proposed infrastructure has greatly facilitated and accelerated the deployment of changes in different parts of the system instances.

C. GIS for Public Transport Routes Computing

The system called *LineShaper*[15] designed for obtaining geographically accurate routes of public transport lines [32]

[11]Dexter Tutorial GitHub repository: https://github.com/dexter-fit/dexter-tutorial

[12]Dexter.FIT projects of the research group at Brno University of Technology, Faculty of Information Technology: https://dexter.fit.vutbr.cz/.

[13]Home IoT solution https://jiapl.iot.petr-john.cz/.

[14]System for Analysis of City Council Decision-Making home page https://zastupko.cz/.

[15]GIS for Public Transport Routes Computing home page https://dexter.fit.vutbr.cz/lineShaper.

was tested on the project server during the whole development. The system itself consists of two user mapping applications, a computing server, and a dedicated PostGIS database. The user applications allow editing map data and entering routing tasks. The database then stores the current state of the maps as well as the associated metadata. In terms of computing power requirements, the most demanding part of the whole system is the computing server.

The main task of the computational server is to construct the routes of the lines. Each route represents one computational task, which consists of constructing a graph and browsing it. In the constructed graph, the implemented algorithm then searches and evaluates the individual routes that correspond to the given stop order and traffic rules. From the resulting routes, the one that most closely matches the estimated actual route of a given line is then selected based on the evaluation.

This case study shows two primary findings. The first one is that the solution greatly simplified the deployment of new versions of the system during development, where individual components could be deployed separately as needed without affecting other components, thanks to the use of docker images. The second one shows that it is possible to use a low-power device like the Raspberry Pi 4B for the development of more complex computations lasting a few seconds. On the other hand, the use of low-power devices in production still has some limitations like increased computing time (in this study case around 25% on average compared to the production environment). It is, however, always easy to migrate the developed system to the production environment due to the use of Docker technology.

V. Discussion

Traditional self-hosting or third-party solutions like VPS and cloud environments (e.g. AWS, Azure) provide expected advantages over ARM-based single-board computers or similar low-power devices having limited resources of the single-board computers. This manifests primarily in large projects that are either updated frequently or are used by a large number of users. Also, the traditional solutions are more suitable when specific requirements are needed. This might include a requirement for either a large number of CPUs, storage, or RAM or a need for specialized hardware (GPUs or NPUs for AI tasks) that may not be already included on the device, or the device might lack drivers. Such solutions, however, have non-negligible shortcomings. Dedicated server hardware comes with a much larger upfront investment which is a problem, especially for smaller start-up projects. On the other hand, third-party solutions significantly increase running costs, and there is also the fact that the developers lose some control over their solution which should not be overlooked in some cases.

The infrastructure presented in this paper brings a compromise between self-hosting servers and third-party services. It provides a flexible and cost-efficient solution, which is particularly highly valuable for the development of new projects by small teams of developers. It allows them to keep higher control over their own solution compared to cloud environments and VPS. Also, it offers them a quick and affordable platform to develop their projects with all important tools provided by cloud solutions. The disadvantages of the presented ARM-based platform are outweighed by the flexibility of the solution, as it is easily possible to switch from the low-end device to both cloud and self-hosted servers thanks to the cross-built docker containers. This empowers the team to develop the application, deploy it quickly, and switch to a more powerful, dedicated solution when the need arises. This was the case in the GIS for Public Transport Routes Computing project. The deployment of the project was moved elsewhere, but the deployment server handles a redirect to the new location.

This fact, combined with the extremely low cost and relatively high performance of these ARM-based SBCs, offers promising future opportunities. For example, it would be feasible to deploy multiple build servers or utilize the ARM SBCs for various supporting tasks, such as running automated end-to-end tests with tools like Cypress. Additionally, the lower performance and ARM architecture can be advantageous, as they closely resemble the specifications of traditional mobile phones.

VI. Conclusions

Cheap devices like the ARM-powered Raspberry Pi can be used to great success for small teams but may struggle when some specialized tasks like routing or AI may be required. While this may be a limitation for some projects, it might be worth considering this option for smaller teams that already have access to both a stable internet connection and a public IP address. The usage of such devices allows teams to quickly iterate over new, automatically built, and deployed versions. The main benefit is that the software deployment is not bound to a specific cloud provider or a similar architecture. This allows the developed application to be quickly and without any specifics of the final deployment, as the resulting docker images can be moved to practically any infrastructure.

Acknowledgments

This work was supported by project Smart information technology for a resilient society, FIT-S-23-8209, funded by Brno University of Technology.

References

[1] M. Fowler and M. Foemmel, “Continuous integration,” 2006.

[2] S. Stolberg, “Enabling agile testing through continuous integration,” in *2009 Agile Conference*. IEEE, 2009, pp. 369–374. [Online]. Available: http://ieeexplore.ieee.org/document/5261055/

[3] S. M. Mohammad, “Continuous integration and automation,” vol. 4, no. 3, 2016.

[4] C. Singh, N. S. Gaba, M. Kaur, and B. Kaur, “Comparison of different CI/CD tools integrated with cloud platform,” in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, 2019, pp. 7–12. [Online]. Available: https://ieeexplore.ieee.org/document/8776985/

[5] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. Singapore Singapore: ACM, Aug. 2016, pp. 426–437. [Online]. Available: https://dl.acm.org/doi/10.1145/2970276.2970358

[6] A. Singh, "A comparison on continuous integration and continuous deployment (CI/CD) on cloud based on various deployment and testing strategies," vol. 9, pp. 4968–4977, 2021. [Online]. Available: https://www.ijraset.com/fileserve.php?FID=36038

[7] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing — the business perspective," vol. 51, no. 1, pp. 176–189, 2011. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167923610002393

[8] B. Johnson, "Cloud computing is a trap, warns gnu founder richard stallman," *The Guardian*, Feb. 2017. [Online]. Available: https://www.theguardian.com/technology/2008/sep/29/cloud.computing.richard.stallman

[9] M. Schwartz, "Switching costs and lock-in — amazon web services," Dec. 2018. [Online]. Available: https://aws.amazon.com/blogs/enterprise-strategy/switching-costs-and-lock-in/

[10] S. R. Group, "Cloud market gets its mojo back; ai helps push q4 increase in cloud spending to new highs — synergy research group." [Online]. Available: https://www.srgresearch.com/articles/cloud-market-gets-its-mojo-back-q4-increase-in-cloud-spending-reaches-new-highs

[11] M. Michalowski, "55 cloud computing statistics for 2024," Mar. 2024. [Online]. Available: https://spacelift.io/blog/cloud-computing-statistics

[12] S. P. Sinde, B. Thakkalapally, M. Ramidi, and S. Veeramalla, "Continuous integration and deployment automation in AWS cloud infrastructure," vol. 10, no. 6, pp. 1305–1309, 2022. [Online]. Available: https://www.ijraset.com/best-journal/continuous-integration-and-deployment-automation-in-aws-cloud-infrastructure

[13] K. S. Poornalinga and P. Rajkumar, "Continuous integration, deployment and delivery automation in aws cloud infrastructure," *Int. Res. J. Eng. Technol*, 2016.

[14] N. Kavya and P. Smitha, "Deploying and setting up ci/cd pipeline for web development project on aws using jenkins," *Int. Res. J. Eng. Technol*, 2022.

[15] I. J. Miller, B. Schieber, Z. D. Bey, E. Benner, J. D. Ortiz, J. Girdner, P. Patel, D. G. Coradazzi, J. Henriques, and J. Forsyth, "Analyzing crop health in vineyards through a multispectral imaging and drone system," in *2020 Systems and Information Engineering Design Symposium (SIEDS)*. IEEE, 2020, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9106671/

[16] E. Zheng, P. Gates-Idem, and M. Lavin, "Building a virtually air-gapped secure environment in AWS: with principles of devops security program and secure software delivery," in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*. ACM, 2018, pp. 1–8. [Online]. Available: https://dl.acm.org/doi/10.1145/3190619.3190642

[17] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2020, pp. 0975–0980. [Online]. Available: https://ieeexplore.ieee.org/document/9031195/

[18] J. P. Magalhaes and L. M. Silva, "A framework for self-healing and self-adaptation of cloud-hosted web-based applications," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. IEEE, 2013, pp. 555–564. [Online]. Available: http://ieeexplore.ieee.org/document/6753846/

[19] M. H. Syed and E. B. Fernandez, "The software container pattern," 2015.

[20] D. Ernst, D. Bermbach, and S. Tai, "Understanding the container ecosystem: A taxonomy of building blocks for container lifecycle and cluster management," in *Proceedings of the 2nd International Workshop on Container Technologies and Container Clouds, IEEE*, 2016.

[21] D. M. Firmansyah, A. C. Prihandoko, and S. Bukhori, "The concept of LXD-based web hosting panel," vol. 1211, p. 012043, 2019. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1211/1/012043

[22] Military Technical Academy "Ferdinand I", D. Gîrbea, P. Ciotîrnae, and Military Technical Academy "Ferdinand I", "Efficient response solution for integrated command and control center using automatic interactive voice response system," vol. 4, no. 1, pp. 35–38, 2021. [Online]. Available: https://jmiltechnol.mta.ro/7/SS_6%20GIRBEA_Ciotirnae-min.pdf

[23] H. Jerkovic, P. Vranesic, and G. Slamic, "Implementation and analysis of open source information systems in electronic business course for economy students," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, 2016, pp. 912–917. [Online]. Available: http://ieeexplore.ieee.org/document/7522270/

[24] A. Bavier, R. McGeer, and G. Ricart, "PlanetIgnite: A self-assembling, lightweight, infrastructure-as-a-service edge cloud," in *2016 28th International Teletraffic Congress (ITC 28)*. IEEE, 2016, pp. 130–138. [Online]. Available: http://ieeexplore.ieee.org/document/7809642/

[25] A. Coles, E. Deliot, A. Edwards, A. Fischer, P. Goldsack, J. Guijarro, R. Hawkes, J. Kirschnick, S. Loughran, P. Murray, and L. Wilcock, "Cells: A self-hosting virtual infrastructure service," in *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. IEEE, 2012, pp. 57–64. [Online]. Available: http://ieeexplore.ieee.org/document/6424929/

[26] V. Talwar, D. Milojicic, Qinyi Wu, Calton Pu, Wenchang Yan, and Gueyoung Jung, "Approaches for service deployment," vol. 9, no. 2, pp. 70–80, 2005. [Online]. Available: http://ieeexplore.ieee.org/document/1405978/

[27] M. Runia and K. Gagneja, "Raspberry pi webserver," in *Proceedings of the International Conference on Embedded Systems, Cyber-physical Systems, and Applications (ESCS)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2015, p. 62.

[28] J. Nikhila, "Web based environmental monitoring system using raspberry pi," in *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*. IEEE, 2017, pp. 1074–1080. [Online]. Available: https://ieeexplore.ieee.org/document/8454964/

[29] S. E. Princy and K. G. J. Nigel, "Implementation of cloud server for real time data storage using raspberry pi," in *2015 Online International Conference on Green Engineering and Technologies (IC-GET)*. IEEE, 2015, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7453790/

[30] P. John, "Web-based visualization of data from smart devices in iot," Master's thesis, Brno University of Technology, Faculty of Information Technology, Brno, 2021. [Online]. Available: https://www.vut.cz/studenti/zav-prace/detail/145430

[31] K. Zaklová, J. Hynek, and T. Hruška, "Towards transparent governance: Unifying city councils decision-making data processing and visualization," in *Lecture Notes in Networks and Systems*, ser. Lecture Notes in Networks and Systems, vol. 2024, no. 987. Springer Nature Switzerland AG, 2024, pp. 402–411. [Online]. Available: https://www.fit.vut.cz/research/publication/13126

[32] J. Lazúr, J. Hynek, and T. Hruška, "From data to routes: A comprehensive approach to public transport line routing," in *IEEE Xplore*, ser. 2024 Smart City Symposium Prague (SCSP). Institute of Electrical and Electronics Engineers, 2024, pp. 1–6. [Online]. Available: https://www.fit.vut.cz/research/publication/13183