Сучасний стан наукових досліджень та технологій в промисловості. 2021. № 4 (18) *ISSN 2522-9818 (print)* *ISSN 2524-2296 (online)*

UDC 681.3.06 DOI: https://doi.org/10.30837/ITSSI.2021.18.023

D. ZOLOTARIOV

AUTOMATED DEPLOYMENT OF A SOFTWARE ENVIRONMENT FOR MICROSERVICES IN A RAPIDLY CHANGING TECHNOLOGY STACK

The article is devoted to the development and substantiation of practical recommendations regarding the formation of a mechanism for deploying a software environment for creating and executing microservices in a rapidly changing technological stack. The **subject** of the research is the basics of building a system for automated deployment of a software environment for the development and execution of microservices. The **purpose** of the article is to develop and substantiate practical recommendations for the formation of a mechanism for deploying a software environment for creating and executing microservices in a rapidly changing technological stack. The **task** of the work: to determine the necessary elements of the deployment mechanism of the software environment and provide an analysis of the functional load for each of them, set specific tasks that must be solved when building each of them, propose and justify the choice of tools for their solution. In the course of the study, the **methods** of system analysis were used to decompose a complex system into elements and each element into functional components. As of the study, it was established that such a mechanism should consist of the following elements: a universal server initialization **a result** subsystem for any technological stack and a software environment deployment subsystem for developing or executing an application of a certain type on a certain technological stack. Each element is described in detail, its functional load is shown and its role in the overall system is substantiated. It is shown that such a standardized approach to the deployment of the development and runtime environment allows, among other things, to solve the problem of operating microservices in a tested environment. **Conclusions.** Practical recommendations for the formation of a mechanism for deploying a software environment for creating and executing microservices in a rapidly changing technological stack have been developed and substantiated. This mechanism is automated. It shows its flexibility and versatility in relation to programming languages and other features of the software environment. It is pointed out that when implemented in the shell language, bash does not need any third-party applications for its work. The economic benefit of using the proposed mechanism is shown. The ways of its improvement are shown.

Keywords: deployment of the environment; microservice architecture; distributed infrastructure; technological stack; saving resources and money.

Abbreviations

IoT – Internet of things,

CI / CD – (Continuous integration and continuous delivery) systems of continuous automated testing and updating software products as they are developed,

Production-server – a server with developed software running under design conditions and loads, available to its customers,

Stage-server is a server for testing the developed software under design conditions and loads.

Introduction

The last few years in the IT industry have been marked by the rapid development of distributed data processing and user requests and, as a result, the widespread use of microservice architecture. This allows the simplest way to divide such processing between many separate components, depending on the expected result.

Many works have been published covering both the microservice architecture itself as a basis for the development of new applications [1-5], and the transition to it in existing ones [6-8], a study of its features [9] and applicability depending on the scale of the company and the product [10]. This is also confirmed by the growing popularity of related queries in Google Trends [11].

On its basis, not only the user-server messaging is built, but also inter-server interaction, examples of which are modern high-load media sites and web services [12], as well as the IoT [13].

Analysis of publications

The advantage of microservice architecture is the ability to add and remove elements situationally without changing the system as a whole. For example, the Guardian uses this approach [14] to capture events associated with a specific high-profile sporting event. For this, a specialized microservice is created, which is easily integrated into the media platform, and after losing its relevance, it is simply removed from it. At the same time, a significant load generated by users interested in the indicated event goes to the servers dedicated for this microservice and does not load the main system. For the user, however, this division is imperceptible – he sees one "seamless" website.

From the point of view of economic benefits, the addition of a new situational microservice does not entail changes to the main computing infrastructure, since it is added "on the side", which means that the costs of its deployment are the lowest of all possible.

In addition, an important advantage of the microservice architecture is the fact that, due to the weak connection between the elements, it allows you to use any programming languages for their development. For situational microservices, this means the ability to use the latest languages for the rapid creation of software products with simplified testing, since the life cycle of such a software product is almost always short and the result does not imply development. This also translates into significant cost and resource savings and accelerates time-to-market.

Development technologies are constantly being improved, and this entails the constant emergence of more and more new programming languages [15], a change in their share in development [16], as well as the disappearance of previously created ones due to the loss of community interest in them [17]. Along with this, the technological stack for developing situational microservices can also change.

Despite the advantages of rapid development languages, which include convenience due to more thought-out syntax and speed due to many built-in standard operations, they also have a significant disadvantage due to their "youth" in the form of many not yet identified internal errors.

Therefore, when choosing such a language, it is on the developer's shoulders not only to determine its compliance with a specific task, but also to check all the necessary language tools for errors. This leads to a significant slowdown in the development process at the start.

Formulation of the problem

To fully take advantage of the advantages of rapid development languages and level their shortcomings, a unified centralized system of checking their capabilities, as well as preliminary and load testing, going from the support specialists of the software environment: DevOps specialists and server administrators to the developers, is required, and not vice versa.

The result of such preparation should be a set of requirements for the version of the language and its environment (technological stack), which allows you to create on its basis stably working servers for the development and execution of software products.

The deployment process itself should also be standardized, and the system based on it should be automated and universal. To prevent the developer or DevOps specialist during the server deployment process from changing the version and settings of the software or supporting services, or making a mistake in the sequence of their installation.

Such an attempt at standardization was undertaken in Avito [18], but for the special case of a set of languages (PHP, Golang and Python), which uses the Docker container development approach. The disadvantage is the very narrow specialization of the system - deployment depends only on the programming language, without taking into account auxiliary services and other components of the microservice software environment.

Therefore, ***the purpose*** of this article is to develop and substantiate practical recommendations regarding the formation of a mechanism for deploying a software environment for creating and executing microservices in a rapidly changing technological stack. This mechanism should be automated, easy to use for users and developers, and universal, that is, independent of specific programming languages.

The goal of the article is to determine the necessary elements of such a mechanism, provide an analysis of the functional load for each of them, set specific tasks that must be solved in the construction of each of them, and propose and justify the choice of tools for their solution.

General approach to the construction of the mechanism

To achieve this objective, the mechanism being developed must perform the following tasks:

The first is the initial server configuration, universal for any technological stack: programming language and tools used. Its purpose is to configure access to the developer/administrators/CI/CD system server and, if necessary, also configure the version control system, proxy, web server, attach the domain and make other basic server settings.

The second – provides initial installation and configuration of the necessary software for the development or execution of a certain type of program on a certain technological stack. Its goal is to provide a software environment ready for development (or execution), ideally without the need for manual additional configuration.

Therefore, such a deployment mechanism should consist of the following elements: a universal server initialization subsystem for any technological stack and a software environment deployment subsystem for developing or executing applications on a certain technological stack.

In general, the server deployment mechanism can be presented as the following flowchart (fig. 1).

The figure shows that the universal initialization of the server is started first, after its successful implementation – deployment for a specific technological stack. After the last successful completion, the server deployment process is considered complete. It provides access to the developer, sends the necessary notifications about the successful completion of the process, and restarts the server. In case of any failure, the process is interrupted with sending the necessary alerts about it.

It is worth noting that in order to accelerate the deployment process when using cloud technologies or other infrastructure tools using virtual machines, it is recommended that you have ready-made images for a set of commonly used server types (for example, in Docker). And use the standard deployment process only if there is no such image.

Server configuration data source

To uniquely identify the list of software products and services that must be installed on the server during the initialization and deployment of the environment, as well as the order of their installation and their settings for the needs of a certain type of microservice and technological stack, this information must be stored centrally. It is recommended to use a well-established method – code storage. For this purpose, an environment configuration file (hereinafter referred to as ".conf") is used, the content of which is determined by the type of microservice and the technological stack, and which should contain:

- an ordered list of software to be installed during server initialization, with individual settings,
- an ordered list of software to be installed for the selected microservice type and process stack, with settings,
- an ordered list of software that must be installed as an environment for this microservice: data warehouses, search engines, etc., with settings,
- access to third-party services, databases, instant messengers and other external systems.

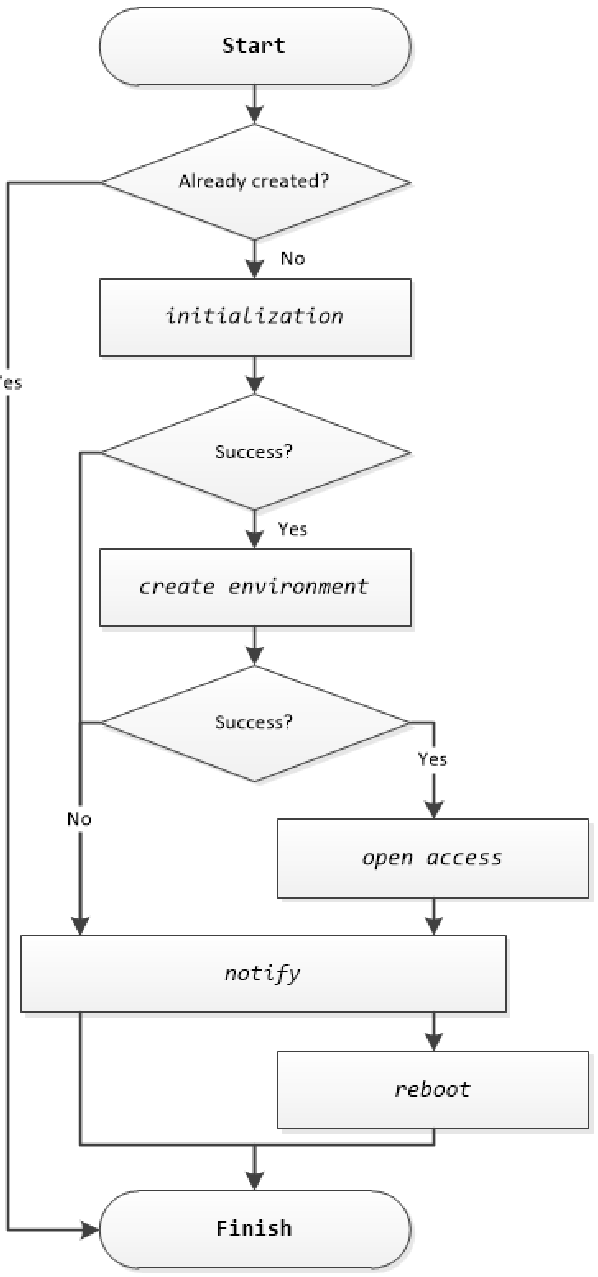

Fig. 1. Flowchart of the server deployment process

This configuration file is then used by the deployment application. As such an application, it is recommended to use standard bash scripts as the easiest and most reliable way. But there are no restrictions on the deployment system itself and the tools used in its development.

All necessary dependencies for the development and testing of the microservice must already be deployed and for all developed the same type of microservice are the same. The use of common dependencies is due to the fact that they are updated in accordance with their improvement, the addition of new data, etc. Using multiple copies of the same dependencies increases the likelihood that the next update will not update all copies correctly. This results in non-test errors that occur only on production servers.

Dependencies can be deployed as a mock server ("plugs"), in particular, in Docker containers, but can be full-fledged.

Server initialization

The purpose of server initialization is to first configure it and set the general set of applications and services required for microservice operation in a certain sequence. A generalized version of such a mechanism can be presented as follows (fig. 2). It should be noted that this is only one possible option: for each type of microservice and development features, this mechanism can both exclude certain stages or steps in them, and include additional ones.

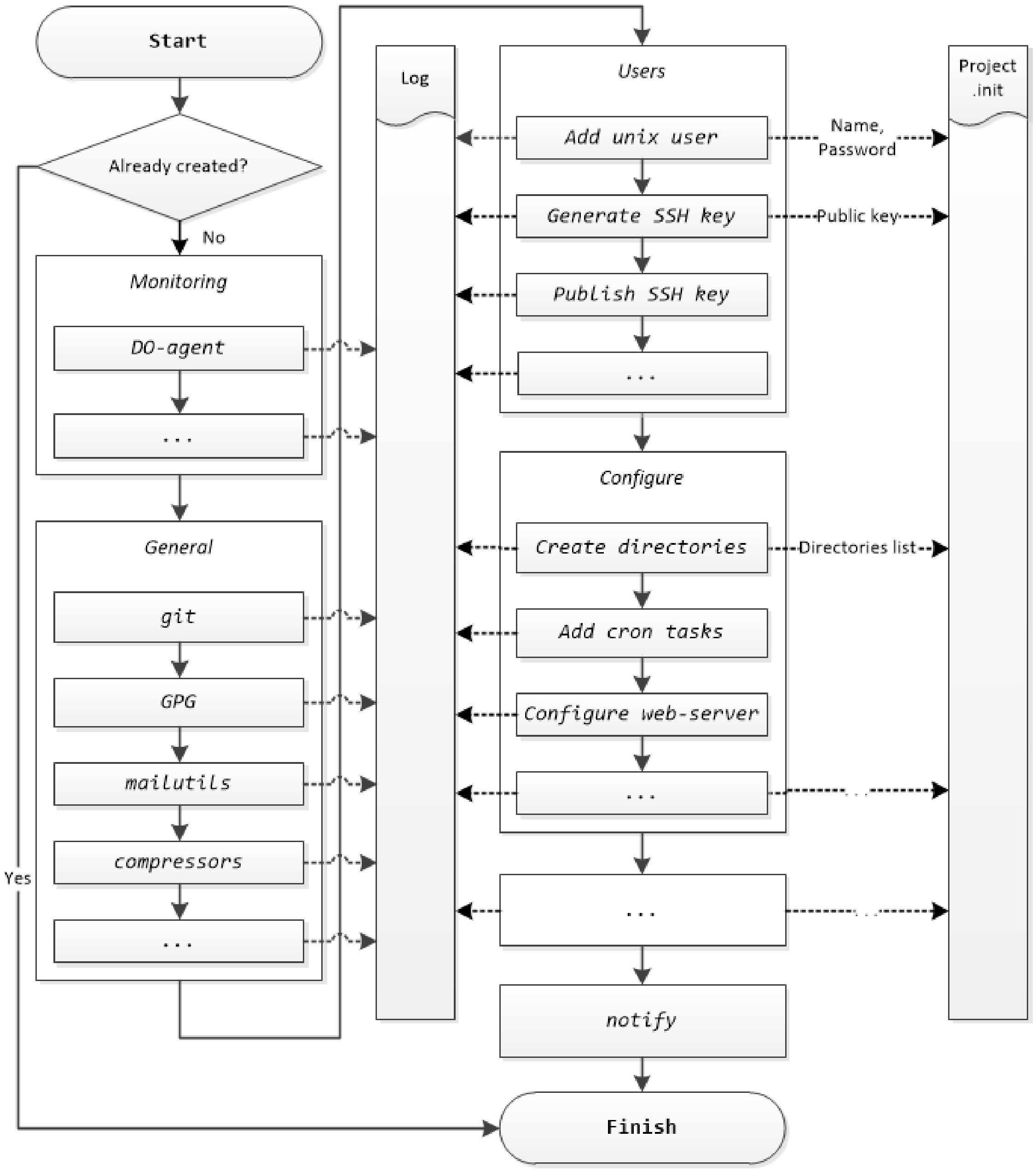

Fig. 2. Flowchart of the server initialization process

The initialization process is divided into several consecutive steps. At the beginning, it is checked whether it has already been produced to avoid re-execution, which almost always leads to a complete reset of settings.

The process begins with the installation of monitoring programs necessary to collect external statistics of the server. These can be programs specifically designed for a specific site of virtual machines (for example, do-agent for DigitalOcean), universal (for example, based on Logstash from Elastic) or individual internal development. Their task is to indicate the state of the server in real time or close to it, as well as provide information for the initial diagnosis of the problem.

Next, a set of utilities is installed to manage the server and fine-tune it. This includes: git, applications responsible for security, programs for working with communication channels, for example, e-mail, archivers and others.

After it, the creation and configuration of the user on whose behalf the microservice process and the generation of SSH keys for it begin. As well as adding a public key to the version control system (Bitbucket, GitLab and the like, including internal ones) for the development server or the CI/CD system for stage- or production-servers.

The next step is responsible for creating project directories, creating and running regular cron tasks, configuring a local web server, and similar general tasks.

In subsequent steps, additional microservice server settings are made, depending on the accepted development standards and the requirements of the server administrator.

The final step, as before, is to publish a notification of the success of the entire process or an error with its detailed description.

During the execution of certain steps that create or modify something, an important result for the developer is saved in the initialization file (label "Project.init"). The following are stored here:

- the name of the created user and his public SSH-key,
- paths to created directories for microservice,
- and other information used in the operation of the server.

For a web server, for example, the added domain and information about its SSL certificate will also be saved here.

Also, during the deployment, the history of each operation ("Log" label) is continuously published to allow you to easily find the location of the error, if it occurs. If an error occurs anywhere in the process, it is interrupted with the publication of a notification to administrators or DevOps specialists.

After the initialization process is complete, the server automatically proceeds to deploy the microservice environment.

Deploying the environment

The goal of deploying a microservice environment on a server is to install a set of necessary applications and services for a microservice to run on a specific technology stack on production and stage servers. In the case of a development server, the necessary tools for development and local testing of the microservice are additionally installed. A schematic block diagram of the mechanism is shown in fig. 3 below.

The entire process is logically divided into two stages, indicated by the "Stage N" labels, consisting of blocks, each of which is responsible for installing and configuring a specific application.

The first stage is the installation of a basic development kit for a specific type of microservice, built on a specific technological stack for a specific programming language: Node.js, Python, Golang, PHP, and the like. If you need to use the local storage of the microservice state, then SQL or NoSQL databases, file storages, queue managers, etc. are also installed for this. Necessary third-party applications are also installed, such as a "headless" browser or GeoIP service. After installing each application, it is fine-tuned for the needs of the microservice.

The second stage carries out installation of utilities intended only during development and fine-tuning of already installed applications for development mode.

As with initialization, the process continuously publishes the execution history of each operation (“Log” label) so that you can easily find the location of the error, if it occurs. Information important for the developer is saved in the initialization file (label "Project.init"). This can be, for example, data for accessing the created local stores of the microservice state or the path to the installed software.

In the event of an error in any step, the process is terminated and a progress report ("Log") and error description is sent to the monitoring system or server administrator. An environment is considered to be successfully deployed only after all stages have been successfully completed.

Each unit installs and configures not only the applications themselves, but also their associated operations, for example, performance monitoring and the like, since within itself it "knows" all the features of the application being installed.

In addition to the blocks themselves, it is also important to precisely set their order within the stage, as well as the order of the stages themselves, which avoids collisions when installing software.

The block approach described above allows the server administrator or specialist to DevOps select the version of the software being installed; determine the most acceptable settings for it, having tested in various options and modes of operation, as well as loads.

It is worth noting that the process of building application installation blocks is continuous. It is recommended to check its relevance when a new version of the software is released, when vulnerability or other deficiency in the existing block is found in it. Therefore, it is recommended to store .conf configuration files in the version control system and version according to the semantic approach [19].

Prospects for further development

The following improvements are seen as prospects for the further evolution of the developed mechanism for automatic deployment of the software environment for creating microservices in the context of a rapidly changing technological stack.

To simplify deployment of the environment, add the second type of configuration files .conf – for "standard" microservices or "standard" technological stack. For them, only the keyword-name is specified in the configuration file, which is a link and is used to load the required full-fledged .conf file or a ready-made server image from the version control system.

To expand the possibilities in the field of CI / CD, add the mechanism after the stages of initialization and deployment of the environment with the preparation of a universal modular pipeline task in Jenkins using the Jenkinsfile for assembly and testing.

Increase the security of the developed deployment mechanism by running it on a third-party server that connects via SSH to the initialized one. Or, provide the deployment mechanism with the ability to completely or partially remove oneself from the deployed server before opening access to the developer in order to hide the details of the settings and increase security.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2021. No. 4 (18)

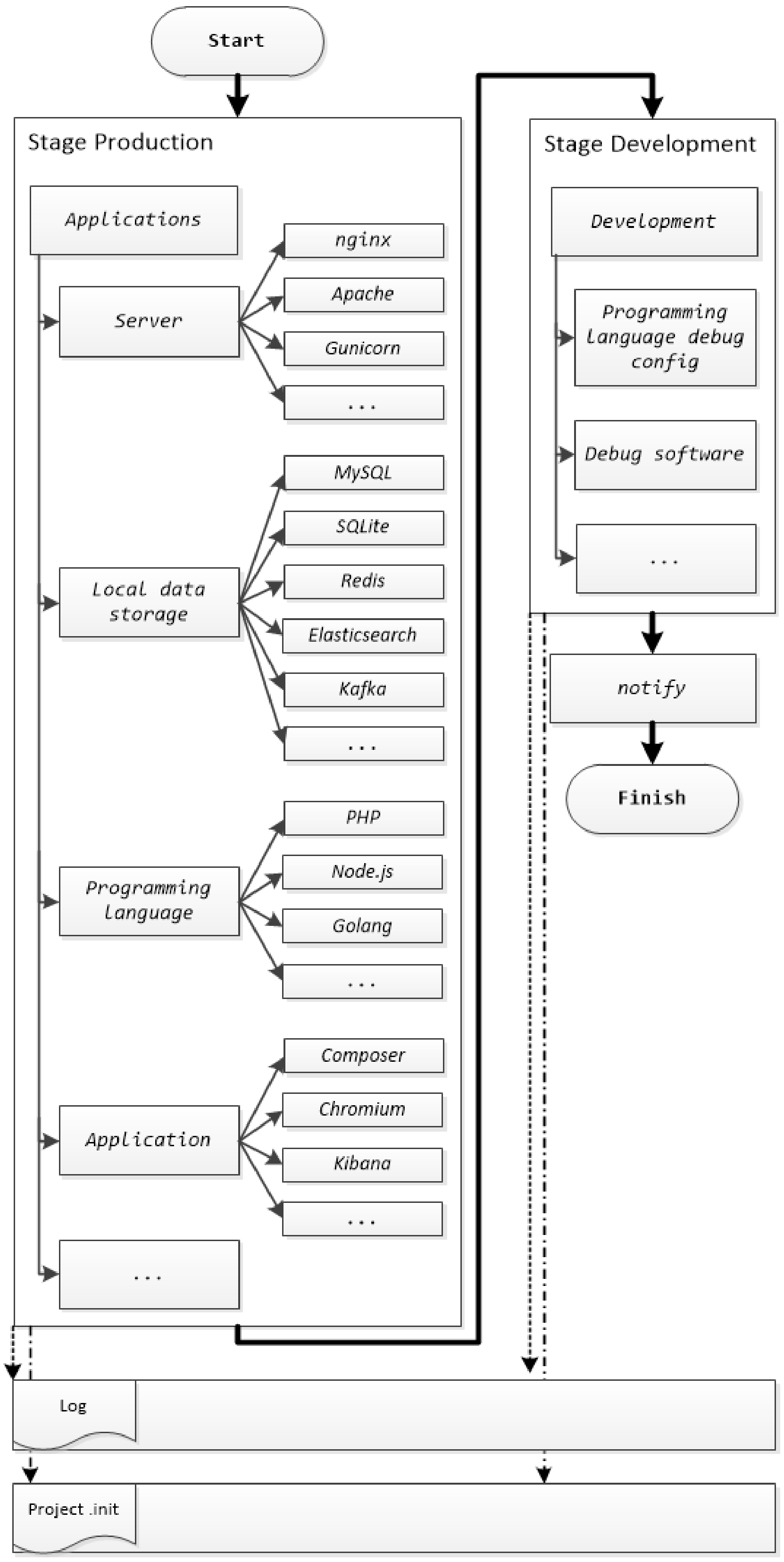

Fig. 3. Flowchart of environment deployment blocks

Conclusions

The work has developed and justified practical recommendations on the formation of a mechanism for deploying a software environment for creating and executing microservices in a rapidly changing technological stack. This mechanism is automated, flexible and versatile, and does not depend on specific programming languages and other features of the software environment. When developing it in the bash shell language, it does not need third-party applications for its work.

It is shown that such a mechanism should consist of the following elements: a subsystem for universal server initialization for any technological stack and a subsystem for deploying a software environment for developing or executing applications of a certain type on a certain technological stack. Each element is described in detail, its functional load is shown and its role in the common system is justified.

It is shown that this standardized approach to deploying the development environment and performing microservices allows, among other things, to solve the problem of operating microservices in a tested environment, where the likelihood of malfunctions is minimized. Hence, the proposed approach also reduces the non-production financial and temporary costs of dealing with the consequences of such failures.

References

1. Rademacher, F., Sachweh, S., Zündorf, A. (2020), "A Modeling Method for Systematic Architecture Reconstruction of Microservice-Based Software Systems", *In: Nurcan S., Reinhartz-Berger I., Soffer P., Zdravkovic J. (eds) Enterprise, Business-Process and Information Systems Modeling. BPMDS 2020, EMMSAD 2020. Lecture Notes in Business Information Processing*, Springer, Cham, Vol. 387. DOI: https://doi.org/10.1007/978-3-030-49418-6_21
2. Zolotariov, D. (2020), "The distributed system of automated computing based on cloud infrastructure", *Innovative Technologies and Scientific Solutions for Industries*, No. 4 (14), P. 47–55. DOI: https://doi.org/10.30837/ITSSI.2020.14.047
3. Zolotariov, D. (2021), "The mechanism for creation of event-driven applications based on Wolfram Mathematica and Apache Kafka", *Innovative Technologies and Scientific Solutions for Industries*, No. 1 (15), P. 53–58. DOI: https://doi.org/10.30837/ITSSI.2021.15.053
4. Zolotariov, D. (2021), "The platform for creation of event-driven applications based on Wolfram Mathematica and Apache Kafka", *Innovative Technologies and Scientific Solutions for Industries*, No. 2 (16), P. 12–18. DOI: https://doi.org/10.30837/ITSSI.2021.16.012
5. Zolotariov, D. (2021), "Microservice architecture for building high-availability distributed automated computing system in a cloud infrastructure", *Innovative Technologies and Scientific Solutions for Industries*, No. 3 (17), P. 13–22. DOI: https://doi.org/10.30837/ITSSI.2021.17.013
6. da Silva, H. H. S., de F. Carneiro G., Monteiro, M. P. (2019), "An Experience Report from the Migration of Legacy Software Systems to Microservice Based Architecture", *In: Latifi S. (eds) 16th International Conference on Information Technology-New Generations (ITNG 2019). Advances in Intelligent Systems and Computing*, Springer, Cham, Vol. 800. DOI: https://doi.org/10.1007/978-3-030-14070-0_26
7. Bucchiarone, A., Dragoni, N., Dustdar, S., et al. (2018), "From monolithic to microservices: an experience report from the banking domain", *IEEE Softw*, No. 35 (3), P. 50–55.
8. Levcovitz, A., Terra, R., Valente, M. T. (2015), "Towards a technique for extracting microservices from monolithic enterprise systems", *In: Proceedings of VEM'15*, P. 97–104.
9. Munari, S., Valle, S., Vardanega, T. (2018), "Microservice-Based Agile Architectures: An Opportunity for Specialized Niche Technologies", *In: Casimiro A., Ferreira P. (eds) Reliable Software Technologies – Ada-Europe 2018. Ada-Europe 2018. Lecture Notes in Computer Science*, Springer, Cham, Vol. 10873. DOI: https://doi.org/10.1007/978-3-319-92432-8_10
10. Sorgalla, J., Sachweh, S., Zündorf, A. (2020), "Exploring the Microservice Development Process in Small and Medium-Sized Organizations", *In: Morisio M., Torchiano M., Jedlitschka A. (eds) Product-Focused Software Process Improvement. PROFES 2020. Lecture Notes in Computer Science*, Springer, Cham, Vol. 12562. DOI: https://doi.org/10.1007/978-3-030-64148-1_28
11. Google Trends (2021), "Microservice architecture – Google Trends", available at: https://trends.google.ru/trends/explore?date=today%205-y&q=%2Fm%2F011spz0k (last accessed 5 February 2021).
12. CIO Dive (2021), "Guardian Life steers tech transformation with microservices", available at: https://www.ciodive.com/news/Cloud-Microservices-Guardian-Life/578732/ (last accessed 5 February 2021).
13. Nasiri, H., Nasehi, S., Goudarzi, M. (2019), "Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities", *Journal of Big Data*, Vol. 6, No. 52. DOI: https://doi.org/10.1186/s40537-019-0215-2
14. Martin Fowler (2014), "Microservices", available at: https://martinfowler.com/articles/microservices.html (last accessed 5 February 2021).
15. Chatley, R., Donaldson, A., Mycroft, A. (2019), "The Next 7000 Programming Languages", *In: Steffen B., Woeginger G. (eds) Computing and Software Science. Lecture Notes in Computer Science,* Springer, Cham, Vol. 10000. DOI: https://doi.org/10.1007/978-3-319-91908-9_15
16. DOU (2021), "Ranking of programming languages 2020: JavaScript is ahead of Java, and Dart entered the first league", available at: https://dou.ua/lenta/articles/language-rating-jan-2020/ (last accessed 5 February 2021).
17. Lee, C. Y. (2019), "Temporal Correlation Analysis of Programming Language Popularity", *J. Korean Phys. Soc*., Vol. 75, P. 755–763. DOI: https://doi.org/10.3938/jkps.75.755
18. Vadim Madison (2018), "What do we know about microservices?" ["Chto my znayem o mikroservisakh?"], *HighLoad ++ 2018, Professional conference for developers of high-load systems*, Moscow.
19. Semantic Versioning (2021), "Semantic Versioning 2.0.0", available at: https://semver.org/lang/ru/ (last accessed 5 February 2021).

Received 23.07.2021

Відомості про авторів / Сведения об авторах / About the Authors

Золотарьов Денис Олексійович – кандидат фізико-математичних наук, Харків, Україна; email: denis@zolotariov.org.ua, ORCID: https://orcid.org/0000-0003-4907-7810.

Золотарёв Денис Алексеевич – кандидат физико-математических наук, Харьков, Украина.

Zolotariov Denis – PhD (Physics and Mathematics Sciences), Kharkiv, Ukraine.

АВТОМАТИЗОВАНЕ РОЗГОРТАННЯ ПРОГРАМНОГО ОТОЧЕННЯ ДЛЯ МІКРОСЕРВІСІВ В УМОВАХ ШВИДКО МІНЛИВОГО ТЕХНОЛОГІЧНОГО СТЕКУ

Стаття присвячена розробці та обґрунтуванню практичних рекомендацій стосовно формування механізму розгортання програмного оточення для створення й виконання мікросервісів в умовах швидко мінливого технологічного стека. **Предметом** дослідження є основи побудови системи автоматизованого розгортання програмного оточення для розробки й виконання мікросервісів. **Метою** статті є розробка й обґрунтування практичних рекомендацій щодо формування механізму розгортання програмного оточення для створення й виконання мікросервісів в умовах швидко мінливого технологічного стека. **Завдання** роботи: виявити необхідні елементи механізму розгортання програмного оточення й надати для кожного з них аналіз функціонального навантаження, поставити конкретні задачі, які повинні бути вирішені при побудові кожного з них, і запропонувати й обґрунтувати вибір інструментів для їхнього розв'язку. У ході дослідження використані **методи** системного аналізу для декомпозиції складної системи на елементи й кожного елемента на функціональні компоненти. У **результаті** дослідження встановлено, що такий механізм повинен складатися із наступних елементів: підсистеми універсальної ініціалізації сервера для будь-якого технологічного стека та підсистеми розгортання програмного оточення для розробки або виконання додатка певного типу на певному технологічному стеці. Кожний елемент детально описаний, показана його функціональне навантаження й обґрунтована його роль у загальній системі. Показано, що такий стандартизований підхід до розгортання середовища розробки й виконання дозволяє, у тому числі, розв'язати задачу експлуатації мікросервісів у протестованому оточенні. **Висновки.** Розроблені й обґрунтовані практичні рекомендації для формування механізму розгортання програмного оточення для створення й виконання мікросервісів в умовах швидко мінливого технологічного стека. Цей механізм є автоматизованим. Показана його гнучкість і універсальність відносно мов програмування й інших особливостей програмного оточення. Зазначено, що при реалізації його мовою командної оболонки bash не має потреби в сторонніх додатках для своєї роботи. Показана економічна вигода від використання запропонованого механізму. Наведені шляхи його вдосконалення.

Ключові слова: розгортання оточення; мікросервисна архітектура; розподілена інфраструктура; технологічний стек; економія ресурсів та коштів.

АВТОМАТИЗИРОВАННОЕ РАЗВЕРТЫВАНИЕ ПРОГРАММНОГО ОКРУЖЕНИЯ ДЛЯ МИКРОСЕРВИСОВ В УСЛОВИЯХ БЫСТРО МЕНЯЮЩЕГОСЯ ТЕХНОЛОГИЧЕСКОГО СТЕКА

Статья посвящена разработке и обоснованию практических рекомендаций относительно формирования механизма развертывания программного окружения для создания и выполнения микросервисов в условиях быстро меняющегося технологического стека. **Предметом** исследования являются основы построения системы автоматизированного развертывания программного окружения для разработки и выполнения микросервисов. **Целью** статьи является разработка и обоснование практических рекомендаций к формированию механизма развертывания программного окружения для создания и выполнения микросервисов в условиях быстро меняющегося технологического стека. **Задача** работы: определить необходимые элементы механизма развертывания программного окружения и предоставить для каждого из них анализ функциональной нагрузки, поставить конкретные задачи, которые должны быть решены при построении каждого из них, предложить и обосновать выбор инструментов для их решения. В ходе исследования использованы **методы** системного анализа для декомпозиции сложной системы на элементы и каждого элемента на функциональные компоненты. В **результате** исследования установлено, что такой механизм должен состоять из следующих элементов: подсистемы универсальной инициализации сервера для любого технологического стека и подсистемы развертывания программного окружения для разработки или выполнения приложения определенного типа на определенном технологическом стеке. Каждый элемент детально описан, показана его функциональная нагрузка и обоснована его роль в общей системе. Показано, что такой стандартизированный подход к развертыванию среды разработки и выполнения позволяет, в том числе, решить задачу эксплуатации микросервисов в протестированном окружении. **Выводы.** Разработаны и обоснованы практические рекомендации для формирования механизма развертывания программного окружения для создания и выполнения микросервисов в условиях быстро меняющегося технологического стека. Этот механизм является автоматизированным. Показана его гибкость и универсальность в отношении языков программирования и других особенностей программного окружения. Указано, что при реализации его на языке командной оболочки bash не нуждается в сторонних приложениях для своей работы. Показана экономическая выгода от использования предложенного механизма. Показаны пути его усовершенствования.

Ключевые слова: развертывание окружения; микросервисная архитектура; распределенная инфраструктура; технологический стек; экономия ресурсов и денег.

Бібліографічні описи / Bibliographic descriptions

Золотарьов Д. О. Автоматизоване розгортання програмного оточення для мікросервісів в умовах швидко мінливого технологічного стеку. *Сучасний стан наукових досліджень та технологій в промисловості*. 2021. № 4 (18). С. 23–30. DOI: https://doi.org/10.30837/ITSSI.2021.18.023

Zolotariov, D. (2021), "Automated deployment of a software environment for microservices in a rapidly changing technology stack", *Innovative Technologies and Scientific Solutions for Industries,* No. 4 (18), P. 23–30. DOI: https://doi.org/10.30837/ITSSI.2021.18.023