Method for deploying enterprise applications in a K8s cluster

Andriy Yerokhin[†], Oleh Zolotukhin[†], Valentin Filatov[†], Maryna Kudryavtseva[*,†] and Denys Kalinin[†]

Kharkiv National University of Radio Electronics, Nauky av. 14 61166, Kharkiv, Ukraine

Abstract

The paper analyzes widely used DevOps practices for continuous integration and continuous deployment. Continuous Integration/Continuous Delivery (CI/CD). The result of this study is an analysis of the most popular CI/CD methods for use in microservices projects with the Kubernetes orchestration system for interoperability of containers. To test the results of the study, an open source test project was identified that best fits the business project. A method aimed at optimizing the process of deploying Enterprise applications in a Kubernetes environment has been improved. After determining the optimal methods in the context of continuous integration and continuous deployment (CI/CD) have been implemented on business project. For each of the selected solutions, pipelines have been developed aimed at testing functionality and debugging the system to ensure their efficiency and correct functionality. During testing, an alternative to the three selected modern solutions was proposed, namely their combination (Spinnaker, Jenkins, Helm) due to the impossibility of using only one solution. The results of the study can be used when selecting CI/CD solutions for Enterprise-level projects to address the needs and challenges of large corporations or organizations. These projects are characterized by a variety of technical, financial, and business aspects and may include the development and implementation of complex systems, strategic planning, technology integration, as well as ensuring a high level of scalability and security. Using the improved method the time to implement new versions has decreased to 50%, to improve the efficiency to implement new versions by 50%, reducing the number of errors detected during deployment by 30%, to increase the reliability of the deployment by 89%, to increase the security of deployment by 99.99%. To increase the efficiency of deploying a new software version in a production environment with minimal risks, downtime, and impact on users, it is recommended to additionally use the LLM (Large Language Model) module.

Keywords

Big Data, CI/CD, computer science, data visualization, ensemble learning, enterprise applications, helm, jenkins, kubernetes, neural network, python, stacking [1]

1. Introduction

Many corporate structures, even those not involved in information technology, have identified the presence of their own digital services that are subject to systematic implementation and updating. This process includes adding innovations, correcting identified shortcomings, eliminating vulnerabilities and activating new functions. An important characteristic of this development cycle is the need to ensure continuous functionality during updates, ensuring uninterrupted operation. The need for speed of implementation of changes and flexibility of applications has caused transformation architectural approaches in the field of software, leading to the rejection of monolithic structures in favor of microservice architecture.

To simplify the transfer of microservices from test to production environments and improve development efficiency, these microservices were packaged into containers. As applications gradually improved, the number of microservices and containers increased, creating a need for

[*] *AISSLE-2025: The International Workshop on Applied Intelligent Security Systems in Law Enforcement, October, 30–31, 2025, Vinnytsia, Ukraine.*

[1,*] Corresponding author.

[†] These authors contributed equally.

andriy.yerokhin@nure.ua (A. Yerokhin); oleg.zolotukhin@nure.ua (O. Zolotukhin); valentin.filatov@nure.ua (V. Filatov); maryna.kudryavtseva@nure.ua; (M. Kudryavtseva); denis.kalinin@teamdev.com (D. Kalinin)

0000-0002-8867-993X (A. Yerokhin); 0000-0002-0152-7600 (O. Zolotukhin); 0000-0002-3718-2077 (V. Filatov); 0000-0003-0524-5528 (M. Kudryavtseva); 0009-0004-4431-7728 (D. Kalinin)

optimized management and configuration of the software development process. To address these challenges, Kubernetes technology emerged and evolved, providing centralized container management, accelerating and simplifying the process of introducing new products to the market and creating an efficient development and testing cycle.
The paper explores the most popular CI/CD methods for use in projects with microservices on Kubernetes. For practical implementation, a business project was selected for which it is necessary to choose a CI/CD solution, as well as to transfer existing components to microservices that will be deployed on a Kubernetes cluster for testing.

2. The purpose of the work

The purpose of the work is to analyze the possibilities of using methods in the process of designing CI/CD infrastructure and improving the existing method of deploying applications in Kubernetes in order to effectively implement enterprise applications in this environment. It is important to find and research methods that would be most suitable for solving the task.

The goal requires solving the following scientific problems:

- research into the current state of the technological landscape related to the deployment of Enterprise applications in a Kubernetes cluster;
- analysis of existing approaches and methods used to deploy applications in a Kubernetes environment, as well as identifying the advantages and limitations of each of the existing methods;
- improvement and description of a method aimed at optimizing the process of deploying Enterprise applications in the Kubernetes environment;
- use of artificial intelligence methods to speed up the deployment process;
- implementation of an experiment using an improved method of deploying applications in a real or simulated Kubernetes cluster;
- comparison of results and with existing deployment methods and recommendations for an improved method for deploying Enterprise applications in Kubernetes.

3. Related Works

Methods for deploying applications in a Kubernetes cluster can include various approaches and tools depending on the needs and requirements of the project. The architecture of a Kubernetes cluster is given in Fig. 1.

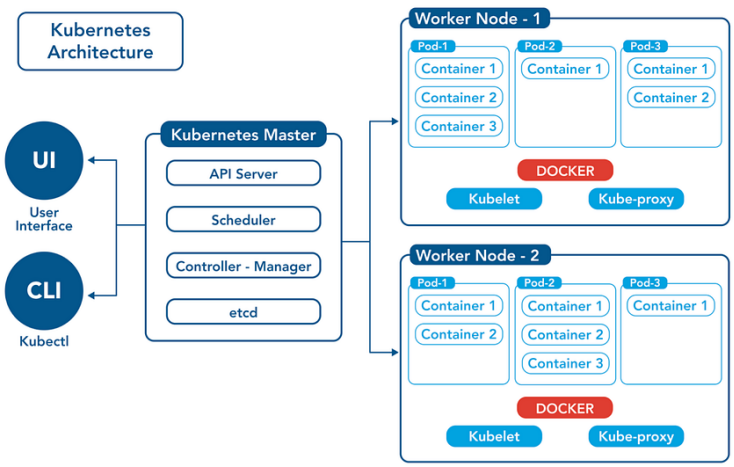

Figure 1: Kubernetes cluster architecture

In general, all these methods can be classified into the following main types [1]:

- Imperative.
- Declarative.
- Developing templates and using tools such as Helm and other package managers.
- GitOps.
- Kubernetes operators.
- CI/CD pipelines.

Let’s consider application deployment methods in more detail.

Additionally, in this context, an important element of the Agile approach is the use of Continuous Integration and Continuous Deployment (CI/CD). Continuous integration (CI) and continuous deployment (CD) are popular software development practices for automation and reducing feedback time. However, improperly set up CI/CD pipelines can cause development delays [2].

The imperative method allows you to use the Kubernetes API without requiring configuration files or YAML manifests. In this approach, Kubernetes takes responsibility for determining what needs to be done to achieve the expected result.

The declarative method of deploying an application in Kubernetes is that the developer describes the desired state of the system in the form of configuration files, rather than specifying specific steps to achieving this state [3]. The basic idea is to specify what the system should look like as a result of deployment, and leave the task of determining the optimal path to this state to Kubernetes itself.

Helm is a package management tool for Kubernetes that simplifies the deployment and management of applications and their dependencies in a Kubernetes environment. Using Helm and other package managers allows you to describe, manage, and deploy applications on a Kubernetes cluster [4].

GitOps is a methodology for managing infrastructure and software deployment in which all configuration and state descriptions of the infrastructure are stored in a version control system such as Git. The basic idea is that the state of the system should reflect the state defined in the Git repository. GitOps is based on the principles of declarative configuration, version control, automated deployment, and self-healing [5].

Kubernetes operators are special controllers that extend the functionality of Kubernetes by enabling automation of routine tasks and application management. They leverage the native capabilities of Kubernetes to provide automated deployment, scaling, and application management [6].

CI/CD pipelines allow integrate with Continuous Integration and Continuous Deployment to automate the processes of building, testing, and deploying code to a Kubernetes cluster.

In today's information environment, efficient application deployment is a critical component for ensuring stable and scalable software operation. In the context of container orchestration, Kubernetes has become an integral part of the infrastructure. However, to optimally utilize the power of Kubernetes, it is important to choose the right deployment methods, including the use of the Helm package manager and CI/CD approaches [7].

The Helm package manager is becoming an important tool for standardizing and automating application deployment. Using Helm charts, developers can package and distribute their applications, and administrators can perform configuration management [8].

Fundamentally an important stage is Continuous Integration, which involves automated integration of code into the repository. Testing, building, and verifying the code help ensure high software quality.

Continuous Delivery extends CI by automating the deployment process to test and production environments. Using Helm in CI/CD pipelines helps automate the deployment of Helm charts and configuration management.

Helm can be integrated into CI/CD pipelines to automate the deployment and configuration of applications in Kubernetes. Helm charts are packaged, deployed, and configured with the appropriate resources.

The advantages of using CI/CD and Helm are a high level of automation, making the tasks of developers and DevOps engineers easier.

Deploying applications in Kubernetes is a complex task that requires choosing the right methods. Using Helm and CI/CD approaches helps standardize, automate, and effectively manage applications in a Kubernetes environment. This not only ensures reliable deployment, but also provides a path to creating scalable and fault-tolerant systems [9].

After conducting a preliminary analysis of the subject area and considering modern methods and strategies that can be used in the process of deploying enterprise applications in a Kubernetes cluster, a corresponding conclusion can be formulated. At this stage of evolution, the selection of methods is based on a comprehensive analysis of numerous popular solutions and taking into account the requirements for the test project [10].

Therefore, it is relevant to solve the problem of improving existing methods and improving a universal method suitable for solving problems in designing CI/CD infrastructure using the most popular technologies, strategies, and approaches. This will be the result of the research.

4. Proposed model and technique

For effective project implementation, it is recommended to provide for an analysis of all components of the project system. The project includes various services and subsystems (monitoring solutions with visualization, services used to develop analytical models that are subsequently used in business analytics, APIs that directly interact with clients who constantly need access to the data warehouse, existing relational and non-relational data warehouses) [11].

Figure 2 presents the general principle of the configuration of the future system (see Fig.2).

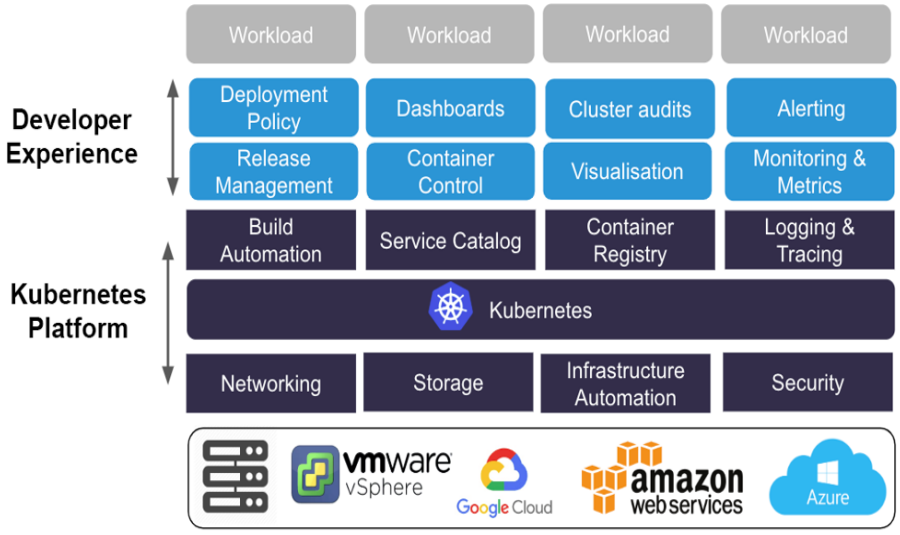

Figure 2: The principle of building a future system

After reviewing the entire system and identifying the technologies used, it is advisable to use the Jenkins, Spinnaker, and Helm tools, because the selected solutions have advantages [12, 13].

As one of the pioneering open source continuous integration servers, Jenkins remains a recognized leader in the industry. It exhibits significant flexibility and has a user-centric interface that makes it extremely attractive to use [14, 15].

The flexible nature of Jenkins can be considered an advantage or a disadvantage, depending on the requirements of a particular project. In the context of the project under study, it is important to

have a graphical interface that facilitates understanding of the tool. The use of various modules in Jenkins allows you to visualize different types of reports, such as test results, code coverage, and other aspects that were subjected to static analysis.

The convenience of combining and orchestrating tasks in a GUI (Graphical User Interface) is relatively simplified compared to using YAML (YAML Ain't Markup Language) files. However, using YAML files to define a build pipeline has the advantage of being clean and structured. This allows you to document changes to the pipeline directly in the repository as part of the source code. This approach allows you to use different YAML files for different release branches, thus providing granular configuration tracking for each branch.

Regarding the use of Spinnaker in a project, it provides several important advantages [16, 17]:

1. Cross-platform. Spinnaker is a cross-platform tool and supports various cloud platforms such as AWS, Google Cloud, Microsoft Azure, and Kubernetes. This makes it an ideal choice for projects that use different cloud infrastructures or a combination of cloud and on-premises computing.
2. Continuous Delivery and Deployment. Spinnaker simplifies the continuous delivery and deployment process by automating the entire cycle from package creation to production deployment, resulting in faster and more secure development cycles
3. Secure Deployments. Spinnaker allows for a variety of deployment strategies, such as staged deployments or testing and disaster recovery strategies. This helps reduce risk and ensures more secure software releases.
4. Visualize the delivery process. Spinnaker provides an intuitive graphical interface that allows you to visualize the entire software delivery process, from build to deployment. This makes it easy to track and analyze each stage of the process.
5. Integration with other tools. Spinnaker easily integrates with other popular development tools such as Jenkins, Git, and Slack. This allows you to build a powerful tool stack for development and delivery automation.
6. Extensibility: Spinnaker is an extensible tool that can be easily adapted to the specific needs of a project. It supports the creation of custom extensions and plugins, allowing you to ensure compliance with specific requirements and infrastructure.

Overall, using Spinnaker allows you to increase development efficiency and provides tools to manage and automate the software delivery process in any deployment area.

Using Helm on a project has several key benefits [18, 19]:

1. Automate deployment and configuration management. Helm allows you to automate application deployment and configuration using well-defined packages (charts). Helm Chart provides an orderly structure for organizing and managing application configuration.
2. Standardization and reuse. Helm allows you to standardize configuration and reuse it across different environments (development, testing, operation). Also Helm Chart has a standardized structure that allows for easy sharing and use of ready-made solutions.
3. Modularity and extensibility. Helm allows you to create and use different Helm Charts for different components of the application and easily extend or modify these charts to meet specific project needs.
4. Dependency Management. The built-in dependency management system allows you to effectively manage dependencies between applications. A Helm Chart can contain dependencies, making it easier to manage other Helm Charts or resources.
5. Versioning and Release History. Helm provides versioning of Helm Charts and an easy way to manage release history. Helm also has versioning support, which allows you to easily track and revert to specific versions of configurations.

6. Umbrella Chart for managing multiple Helm Charts. Umbrella Chart allows you to combine multiple Helm Charts to manage their deployment as a single system. This allows you to organize a complex application into subprojects, where each Helm Chart corresponds to a separate component of the application.
7. Simplify microservice architecture. Using an Umbrella Chart can simplify the management and deployment of a microservice architecture, where each Helm Chart corresponds to a separate service.

The overall benefit of using Helm on a project is that Helm, Helm Chart, and Umbrella Chart help create more structured, automated, and easily manageable configurations for deploying applications at various stages of development and operation.

The CI/CD diagram is shown in Figure 3 (see Fig.3).

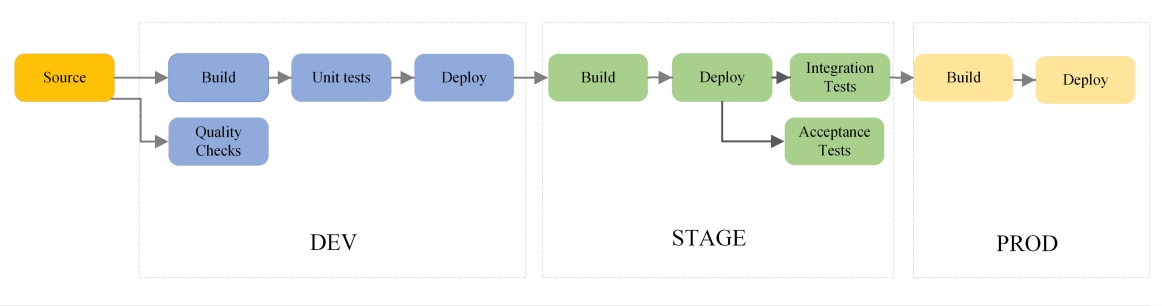

Figure 3: CI/CD diagram

Thus, project deployment can be significantly improved by implementing a joint solution of Jenkins, Spinnaker, and Helm, as these are quite powerful tools.

The stages of the CI/CD pipeline can be represented as follows [20].

1. Quality Checks. When a developer creates code and publishes it to a repository, the system will immediately be prompted to start a further code analysis process. The code must be checked for static code policies.
2. To compile, you need to configure a docker container as a build agent. Then the CI/CD tool pulls in the latest code changes.
3. Testing. During testing, several types of CI tests are performed:

- Code quality review may or may not occur, depending on when the formal CI process begins.
- Unit Tests are fundamental tests that are performed when new features are added or developed.
- Integration Tests. This cross-module testing of the application will be the main focus of integration testing in the context of continuous integration.
- Acceptance Tests are testing to ensure that the software meets the requirements established during its development phase.

4. Upload Docker Image and Helm Chart to the repository. The executables and packages created in the previous step are assembled into a Docker Image and delivered to the repository. Based on the new Docker Image, a new component diagram (Helm Chart) is assembled and uploaded to the repository (e.g. AWS S3) and a top-level diagram (Umbrella Chart) is assembled.
5. Deployment. Application deployment involves creating isolated environments, Dev, Stage, Prod. The Dev environment configuration should have the minimum resources needed only to run the application. The Prod and Stage environments should be as similar as possible.

A diagram of of CI/CD Pipeline Integration with Jenkins, Spinnaker, Helm, and Kubernetes is presented in Figure 4 (see Fig.4). It visually demonstrates how Jenkins and Spinnaker work together to automate the build, testing, packaging, and deployment of applications to multiple Kubernetes clusters using Helm charts.

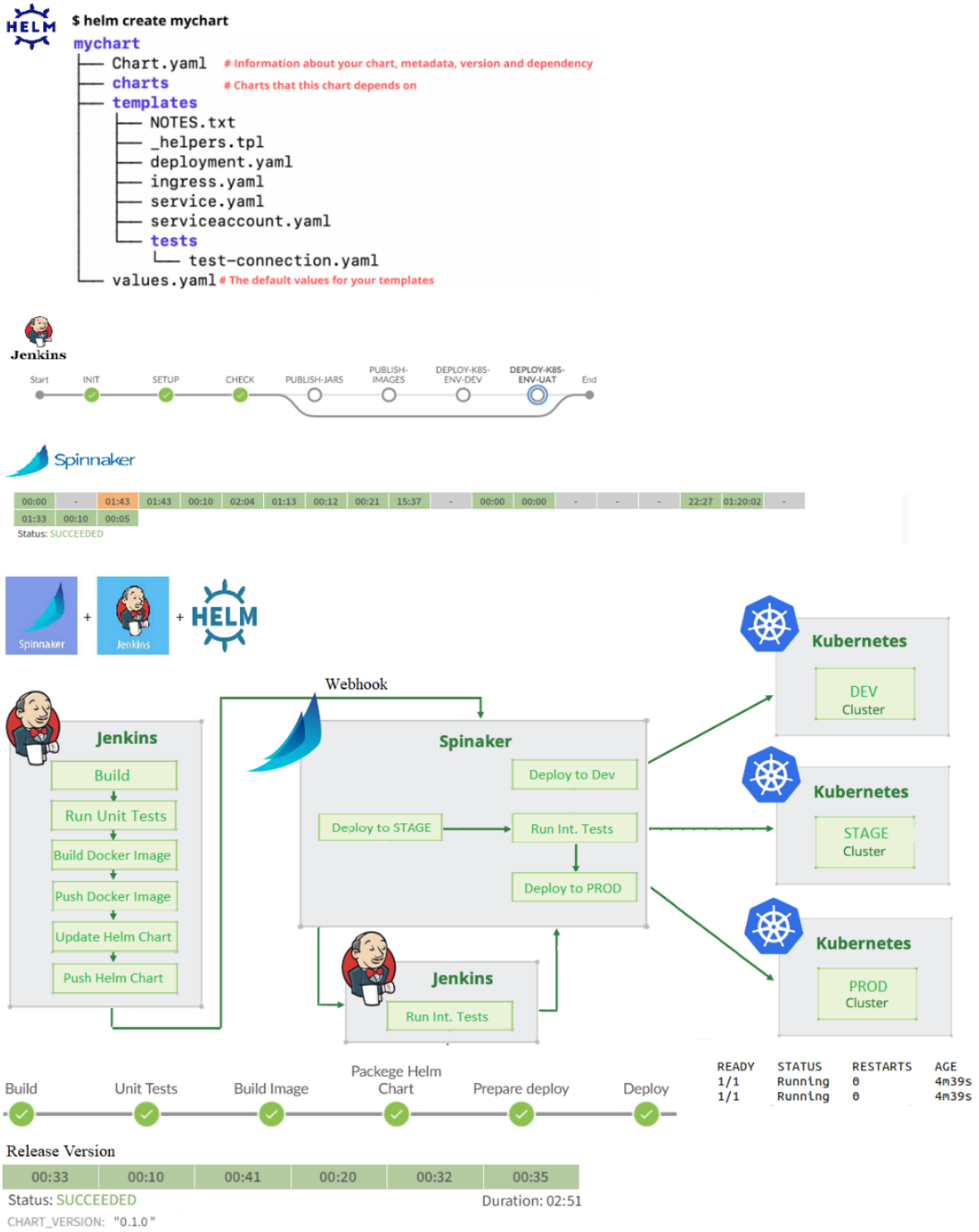

Figure 4: Diagram of CI/CD Pipeline Integration with Jenkins, Spinnaker, Helm, and Kubernetes

To create a mathematical model of deploying an Enterprise application to a Kubernetes cluster using automated delivery (CI/CD), we can formulate this process as follows.

Iterations and steps in Jenkins, Helm, Spinnaker:

1. Stages in Jenkins:
– from Git G select the code, $C_{source}=GitPull(G)$;
– compile the code into a Docker image, $D=Build(C_{source})$;
– run tests:
- Unit tests, $T_{unit}=RunUnitTests(C_{source})$;
- Acceptance tests, $T_{acceptance}=RunAcceptanceTests(C_{source})$;
- Integration tests, $T_{integration}=RunIntegrationTests(C_{source})$;
- Quality tests, $T_{quality}=RunQualityTests(C_{source})$.

2. Creating Helm charts:
– creating a Helm Chart for the application, $H_{app}=CreateHelmChart(C_{source})$;
– creating an Umbrella Chart, $H_{umbrella}=CreateUmbrellaChart(H_{app})$.
3. Deploying to Kubernetes using Spinnaker:
– updating chart versions, $(H_{app}, H_{umbrella})$;
– uploading Helm charts to the Docker image repository UploadCharts(Happ, Humbrella, U);
– choice of environment:
- DeployToK8s(Happ,Cdev,S) – for the DEVELOPMENT environment;
- DeployToK8s(Happ,Cstage,S) – for the STAGE environment;
- DeployToK8s(Humbrrella,Cprod,S) – for PRODUCTION environment,

where input parameters:
G - Git repository with the application source code;
C - Kubernetes cluster, divided into separate environments: DEVELOPMENT (C_{dev}), STAGE (C_{stage}), PRODUCTION (C_{prod});
J - Jenkins server;
D - Docker application image;
U - Docker image repository (registry);
H - Helm charts;
S - Spinnaker for deployment in C_{dev}, C_{stage}, *and* C_{prod} clusters.
Mathematical concepts:
$\{C_{source}, D, T_{unit}, T_{acceptance}, T_{integration}, T_{quality}, H_{app}, H_{umbrella}, U, S\}$ – represent variables describing the state of the system at each step;
$\{GitPull(G), Build(C_{source}), RunUnitTests(C_{source}), RunAcceptanceTests(C_{source}), RunIntegrationTests(C_{source}), RunQualityTests(C_{source}), CreateHelmChart(C_{source}), CreateUmbrellaChart(H_{app}), UploadCharts(H_{app},H_{umbrrella},U), DeployToK8s(H,C,S)\}$ – functions that define the transition to a new system state.
Conditions:
6. successful completion of each step is ensured by running appropriate tests and creating the necessary artifacts (Docker images, Helm charts);
7. deployment in C_{dev}, C_{stage}, C_{prod} clusters can be performed after successful completion of all previous steps.

To increase the efficiency of deploying a new version of software in a production environment with minimal risks, downtime, and impact on users it is recommended to additionally use artificial intelligence tools in the Helm Chart setup, for example, using the LLM module (Large Language Model).

The LLM module is a component that uses the capabilities of large language models and allows you to read the application specification and generate a basic configuration file in the Helm Chart based on it, which contains all the variable parameters (settings) for deploying the application in Kubernetes or suggestions for it (resources, configuration). Then the final stage of the CI/CD process is performed, such asautomatic validation (checking by the system of the correctness and consistency of the configuration, code or environment) and testing, which ensures the reliability and stability of software deployment before the new version of the application enters the production environment.

An advanced method aimed at optimizing the process of deploying Enterprise applications in a Kubernetes environment has the following advantages:

- Automated testing of each microservice allows you to increase the reliability and security of deployment, as well as reduce the time required to implement new versions.
- Using Docker containers as build agents simplifies and automates the process of building microservices.
- Using AWS S3 cloud storage for storing assemblies allows for reliable and scalable assembly storage.
- Deploying an application in isolated environments reduces the risk of an unsuccessful deployment impacting the environment.
- The result of implementing the LLM module is an acceleration of the CI/CD process, a reduction in the load on specialists, and a reduction in human errors.

5. Results

To develop the information system of the project on the use of neural networks in business (using ensemble learning methods), a popular cloud computing provider was chosen - Amazon Web Services. Elastic Kubernetes Service and Amazon Simple Storage Service (Amazon S3) were used to deploy the cluster. The GitHub version control system was used to store the service code [10].

At the stage of executing the business project, a task is created on the Jenkins server, which has a structured appearance, including sequential stages: "Prepare", "Checkout project", "Build", "Unit tests", "Build Image", "Package Helm Chart", "Prepare deploy", "Deploy", and" Acceptance Tests" (See Fig. 5).

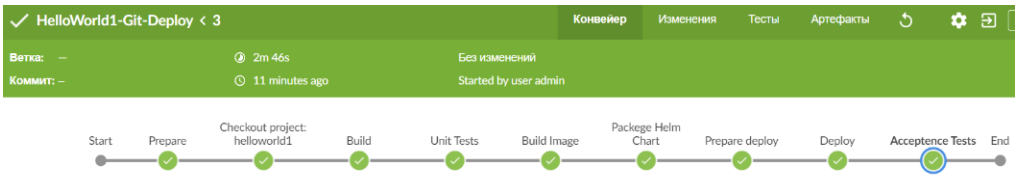

Figure 5: Diagram of an improved method aimed at optimizing the process of deploying Enterprise applications in a Kubernetes environment

This approach allows you to effectively implement changes in a multi-service corporate system in parallel and release new product versions using continuous integration/continuous delivery tools and the Helm package manager for project deployment.

The implementation time of new versions of an enterprise application can be described using the following model:

$$T = T1 + T2 + T3 + T4, \quad (1)$$

where T is the total time for implementing new versions; T1 is the time required to complete the Quality Checks stage; T2 is the time required to execute the Build phase; T3 is the time required to complete the Testing stage; T4 is the time required to complete the Deployment phase.

The effectiveness of the improved application deployment method can be determined using the following models:

$$E1 = (T1 - T1') / T1, \quad (2)$$

where E1 is the efficiency coefficient in terms of reducing the time for implementing new versions; T1 is the time to implement new versions using the traditional method; T1' is the time of implementation of new versions using the improved method.

$$E2 = (N1 - N1') / N1, \quad (3)$$

where E2 is the efficiency coefficient in terms of reducing the number of errors detected during deployment; N1 is the number of errors detected during deployment using the traditional method; N1' is the number of errors detected during deployment using

$$E3 = (R' - R) / R, \quad (4)$$

where E3 is the efficiency coefficient in terms of increasing deployment reliability; R' is the deployment reliability using the improved method; R is the reliability of deployment using the traditional method.

$$E4 = (S' - S) / S, \quad (5)$$

where E4 is the efficiency coefficient in terms of increasing deployment safety; S' is the deployment security using the improved method; S is the deployment security using the traditional method.

The time to implement new versions using the conventional application deployment method is: T = 0.3 + 2 + 1.32 + 0.45 = 4.47 minutes. Using the improved method, the time to implement new versions is: T' = 0.15 + 1.18 + 0.32 + 0.15 = 2.20 minutes.

The efficiency coefficient in terms of reducing the time to implement new versions will be: E1 = (4.47 − 2.20) / 4.47 = 50%. That is, the improved method allows you to improve the efficiency to implement new versions by 50%.

The efficiency coefficient in terms of reducing the number of errors detected during deployment is E2 = (3 - 2) / 3 = 30%. That is, the improved method allows reducing the number of errors detected during deployment by 30%.

In the traditional method of implementing new versions of a corporate application, the probability of a failed deployment is 10%. The improved method allows you to increase the reliability of the deployment to 99%. Then the efficiency coefficient in terms of increasing the reliability of the deployment will be E3 = (99 - 10) / 10 = 89%. That is, the improved method allows you to increase the reliability of the deployment by 89%.

In the traditional method of implementing new versions of a corporate application, the probability of a security breach during deployment is 1%. The improved method allows you to increase the security of deployment to 99.99%. Then the efficiency coefficient in terms of increasing the security of deployment will be E4 = (99.99 - 1) / 1 = 99.99%. That is, the improved method allows you to increase the security of deployment by 99.99%.

6. Discussion

Experiments have shown that the improved method allows:

- To reduce the time required to implement new versions by 50%.
- To reduce the number of errors detected during deployment by 30%.
- To improve the efficiency to implement new versions by 50%.
- Increase the reliability of the deployment by 89%.
- To increase the security of deployment by 99.99%.

An advanced method for automating the deployment of an enterprise application to a Kubernetes cluster is an effective way to improve the quality and reliability of deployment. The method can be used for any enterprise application consisting of a large number of microservices.

7. Conclusions

The results of the study can be used when selecting CI/CD solutions for Enterprise-level projects to solve the problems and challenges of large corporations, especially those using artificial intelligence technologies, especially neural networks, in business. These projects are characterized by a variety of technical, financial, and business aspects and may include the development and implementation of complex systems, strategic planning, technology integration, as well as ensuring a high level of complexity of scalability and security.

In further research, it is advisable to analyze the effectiveness of the improved method under different conditions using AI technology. For example, one can study the impact of application size, test complexity, and load on the production environment on deployment time and quality.

Therefore, it is relevant in the future to improve the proposed method aimed at optimizing the process of deploying Enterprise applications in the Kubernetes environment through a detailed analysis of the most common solutions, their testing, and research into possible integrations between them.

Acknowledgements

This paper is part of the DIOR project that has received funding from the European Union's MSCA RISE program under grant agreement No. 10100828.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, Omega, and Kubernetes, Communications of the ACM 59 (2016) 50–57. https://doi.org/10.1145/2890784.

[2] K. Hightower, B. Burns, J. Beda, Kubernetes: Up and Running, 2nd ed., O'Reilly Media, Sebastopol, CA, 2019.

[3] D. Bernstein, Containers and cloud: From LXC to Docker to Kubernetes, IEEE Cloud Computing 1 (2014) 81–84. https://doi.org/10.1109/MCC.2014.51

[4] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed., O'Reilly Media, Sebastopol, CA, 2021.

[5] A. Vayghan, S. Buchanan, M. Khazaei, K. Beznosov, Kubernetes-native architecture for cloud applications: An empirical study, Future Generation Computer Systems 117 (2021) 25–39. https://doi.org/10.1016/j.future.2020.11.001

[6] M. Villamizar, O. Garcés, H. Castro, M. Verano, R. Salamanca, L. Casallas, S. Gil, Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud, in: 10th Computing Colombian Conference (10CCC), IEEE, Bogota, 2015, 583–590. https://doi.org/10.1109/ColumbianCC.2015.7333476

[7] S. G. Kulkarni, C. Xu, KubeEdge: An edge-native Kubernetes framework for IoT applications, in: 21st International Middleware Conference, ACM, New York, 2020, 1–14. https://doi.org/10.1145/3423211.3425686

[8] C. Pahl, Containerization and the PaaS cloud, IEEE Cloud Computing 2 (2015) 24–31. https://doi.org/10.1109/MCC.2015.51

[9] A. Kratzke, P. Quint, Understanding cloud-native applications after 10 years of cloud computing – A systematic mapping study, Journal of Systems and Software 126 (2017) 1–16. https://doi.org/10.1016/j.jss.2017.01.001

[10] D. Merkel, Docker: Lightweight Linux containers for consistent development and deployment, Linux Journal 239 (2014) 2–11.

[11] J. Turnbull, The Docker Book, 3rd ed., James Turnbull Publishing, San Francisco, CA, 2018.

[12] H. Kang, M. Le, S. Tao, Container and microservice driven design for cloud infrastructure DevOps, in: IEEE International Conference on Cloud Engineering (IC2E), IEEE, Berlin, 2016, 202–211. https://doi.org/10.1109/IC2E.2016.26

[13] L. Villamizar, F. Castro, Deployment strategies in Kubernetes, in: IEEE International Conference on Software Architecture Companion (ICSA-C), IEEE, Hamburg, 2019, 23–26. https://doi.org/10.1109/ICSA-C.2019.00011

[14] M. Choi, J. Kim, J. Song, Performance analysis of Kubernetes-based edge computing cluster, in: International Conference on Information Networking (ICOIN), IEEE, Chiang Mai, 2018, 910–915. https://doi.org/10.1109/ICOIN.2018.8343274

[15] R. Dua, A. R. Raja, D. Kakadia, Virtualization vs containerization to support PaaS, in: IEEE International Conference on Cloud Engineering (IC2E), IEEE, Boston, 2014, 610–614. https://doi.org/10.1109/IC2E.2014.41

[16] V. Filatov, V. Semenets, O. Zolotukhin, Synthesis of semantic model of subject area at integration of relational databases, in: IEEE 8th International Conference on Advanced Optoelectronics and Lasers (CAOL), 2019, 598–601. https://doi.org/10.1109/CAOL46282.2019.9019532

[17] O. Zolotukhin, V. Filatov, A. Yerokhin, M. Kudryavtseva, V. Semenets, An approach to the selection of behavior patterns autonomous intelligent mobile systems, in: IEEE 8th Int. Conf. on Problems of Infocommunications, Science and Technology (PIC S&T), Kharkiv, 2021, 349–352. https://doi.org/10.1109/PICST54195.2021.9772110

[18] V. Filatov, O. Zolotukhin, A. Yerokhin, M. Kudryavtseva, The methods for the prediction of climate control indicators in IoT systems, CEUR Workshop Proceedings (2021) 391–400. https://doi.org/10.5281/zenodo.14526027

[19] Y. Bodyanskiy, P. Otto, I. Pliss, S. Popov, An optimal algorithm for combining multivariate forecasts in hybrid systems, in: V. Palade, R. J. Howlett, L. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems (KES 2003), LNCS 2774, Springer, 2003. https://doi.org/10.1007/978-3-540-45226-3_132

[20] Y. Bodyanskiy, S. Popov, Fuzzy selection mechanism for multimodel prediction, in: M. G. Negoita, R. J. Howlett, L. C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems (KES 2004), LNCS 3214, Springer, 2004. https://doi.org/10.1007/978-3-540-30133-2_101