Shadow Deployment in Software Engineering: Concepts, Methods, and Practice

Aditya A. Kanikar[1], Ashutosh S. Sarap[2], Ayush S. Sonone[3], Krishna S. Khandare[4], Vaibhav N. Maldhure[5]
[1'2'3'4] *Department of CSE, Prof. Ram Meghe Institute of Technology and Research, Amravati, Maharashtra, India*
[5] *Assistant Professor, Department of CSE, Prof. Ram Meghe Institute of Technology and Research, Amravati, Maharashtra, India*

***Abstract*—Shadow deployment is a gradual deployment strategy where production traffic is diverted to a second environment where the latest version of the application or model is deployed for analyzing the performance and compatibility without affecting the end-user experience. This paper provides an overview of various deployment models, the definition and process involved in shadow deployment, and its implementation techniques. The results indicate that shadow deployment decreases the risks associated with deployment, validates the applications' behavior in real-time conditions, and increases system reliability but consumes more resources. Shadow deployment is an equally vital process in today's distributed systems, whereby one can test the system using actual production traffic without disrupting users in the process, giving it an edge over other deployment methods. Moreover, the study looks at the applicability of shadow deployment to microservices, MLOps, and multi-cloud computing.**

***Index Terms*—Shadow Deployment; DevOps; Continuous Delivery; Microservices; Canary Release; Service Mesh.**

I. INTRODUCTION

The modern software development process tries to strike a balance between delivering fast while maintaining stability in the process. The more an application is scaled in a distributed environment and serves numerous users, the harder it becomes to introduce changes without impacting the application's performance and experience for users. There are several different deployment approaches that have been identified: blue-green deployments, canary deployment, rolling update, and feature toggles.

The blue-green approach involves two identical environments for switching in case of any release. The canary approach involves a gradual release to a few users first before releasing to all users. The rolling update approach involves updating system components in stages without shutting down the services. Feature flags allow runtime configuration of features [7], [8].

The safety aspect of the deployment process becomes crucial when dealing with high-availability applications because even small glitches may cause downtime, data corruption, and loss of performance. These consequences will have both monetary and reputational impacts, making it essential for companies to use deployment strategies that check the application's performance and functionality under realistic conditions [9].

Even though the conventional methods like blue-green or canary deployments help mitigate risks, they still work with actual traffic from end users. The shadow deployment technique extends this method because it uses production traffic to test the system in a completely separate environment. Shadow testing involves replicating live traffic into another environment that does not send back any results to the end-users; thus, there is total isolation involved. This makes the testing of the actual performance, behavior, and accuracy of the system easy in an authentic environment. It is best used for microservices and ML systems [12].

Through the use of shadow deployment, one can compare the outputs of both the production and shadow environments, thus facilitating the identification of any discrepancies that may arise due to differences in latency, accuracy, and functionality. This method is particularly effective in applications where there are high standards for reliability and usability [10], [11].

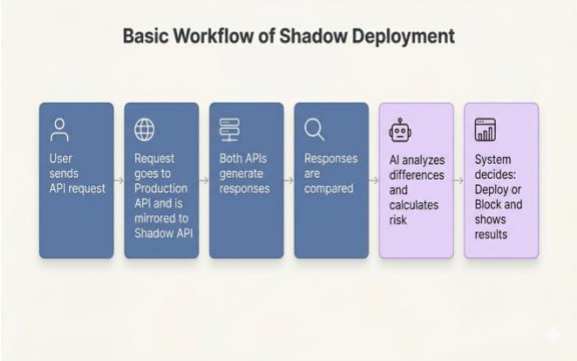

Fig. 1: Basic Workflow of Shadow Deployment

II. LITERATURE SURVEY

Software delivery today is moving from a conventional way of software development towards automation and continuous delivery. Due to the emergence of cloud technology, DevOps, and microservices, today's companies utilize CI/CD pipelines for fast software delivery, while at the same time ensuring that the process is stable and observable. Consequently, deployment methods like blue-green deployment, canary deployment, rolling update, feature flagging, and shadowing became key in modern times [9], [18].

In blue-green deployment, two copies of the environment are used, which enables traffic switching from one environment to another. Although this strategy ensures that there is no downtime during switching, it requires extra resources. In canary release deployment, the updates are rolled out gradually and tested for a smaller group of users. This method allows live monitoring of updates but may cause risks during implementation [19], [21].

Deployment methodologies have grown with the improvements in software development techniques. The earlier methodologies were risky and manual in nature, but with the emergence of DevOps and CI/CD, deployment has become fully automated and robust. Earlier methodologies like Blue-Green and Rolling updates had a focus on uptime and rollback options [17], [18]. Since distributed systems became complicated, canary releases came into existence, enabling incremental roll-outs validated using metrics [19].

Shadow deployment helps overcome the weaknesses of smoke and soak testing by facilitating testing using live production data in an entirely isolated environment. This allows developers to test the efficiency of their software, identify bugs, and assess output differences without interfering with end-users [15], [16]. The adoption of shadow deployment in the CI/CD pipeline enhances reliability and creates feedback loops, enabling data-driven decision-making regarding deployments and reducing the risk of failure in production systems [13], [14].

III. METHODOLOGY

The approach utilized in analyzing the development methods focuses on assessing the impact of shadow deployment on current computer programs. The method entails analytical analysis, comparative assessment, and incorporation into DevOps and CI/CD settings.

Analysis of Modern Deployment Techniques:

The research starts with examining the most common deployment methods applied in today's software development environment. Thanks to cloud computing technologies and microservices architecture, CI/CD tools facilitate frequent releases, thereby requiring greater reliability, ability to roll back, and monitoring. Techniques like blue/green, canary, rolling release, feature flagging, and shadowing have become integral parts of modern deployment processes [9], [18]. The blue-green strategy involves switching smoothly between two identical systems, resulting in high availability but requiring more hardware. The canary release technique enables gradual exposure to a few users for real-time observation [19], [21].

Study of Evolution in Deployment Strategies:

The methodology explores how deployment techniques have advanced from being manual and highly risky to becoming automated and dependable due to DevOps and CI/CD methodologies. With increased complexity, canary deployments allowed developers to test systems using live traffic incrementally. Nonetheless, this technique still requires the use of some real users, making it unsuitable for high-stake environments. Shadow deployments offer an answer to that problem, providing the possibility of testing a system using live traffic without impacting end users [19], [15], [16].

Integration and Evaluation Approach:

The methodology further involves the consideration of the incorporation of shadow deployment in CI/CD pipelines. The performance and other aspects of the system can be assessed through real-world scenarios by directing production traffic to the shadow deployment environment. In terms of advantages, the technique makes use of feedback, data-driven decisions, and risk reduction [13], [14]. Although it is effective, the implementation of this technique presents further infrastructure demands and requires precise traffic replication and monitoring techniques.

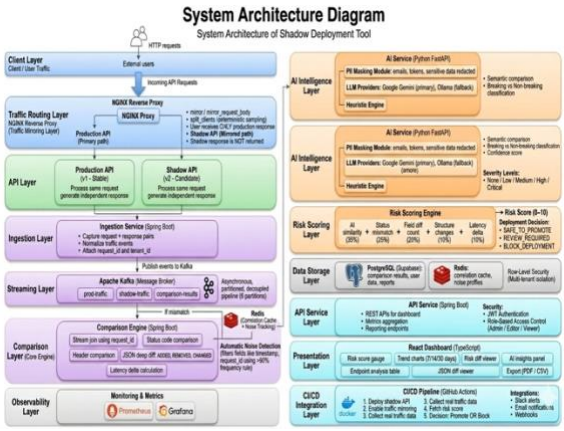

Fig. 2: System Architecture of Shadow Deployment Tool

IV. RESULT AND ANALYSIS

The incorporation of shadow deployment resulted in the enhancement of the validation of the system during heavy load testing. Shadow deployment (v2) worked in conjunction with the live deployment (v1), where the same production load was used for both systems, enabling the comparison of their performance. Observation revealed that v2 was marginally outperforming v1 on p50 latency, meaning that the system coped well with normal load conditions. However, the performance of v2 deteriorated on p99 latency during high load.

The implementation of shadow deployment allowed for a thorough comparison of the outputs from both systems. This exercise highlighted some issues relating to integration with different dependencies because of format differences in the APIs. While no failures occurred, there was a need for adjustments to the processes before full-scale deployment. Additionally, no errors occurred, and therefore, the system did not fail, demonstrating that shadow deployment is a safe process for validation in production settings.

In relation to the recommendation system, shadow testing was utilized to test a new ML model. Through offline testing, the shadow model demonstrated a marginal increase of around 3% in CTR relative to the current one. This confirms that not only was system stability ensured through this method, but model performance was also enhanced. The use of this technique in testing model performance under real-time traffic conditions was very crucial in identifying aspects that cannot be captured through synthetic testing.

An important observation regarding the deployment process was the increased infrastructure burden. Given that the production traffic had been mirrored, the burden on computer resources was almost doubled. The only way to ensure efficient operation was to scale out and have an effective traffic mirroring system. Additionally, data protection issues were addressed through data masking and tokenization to protect the privacy and security of user data.

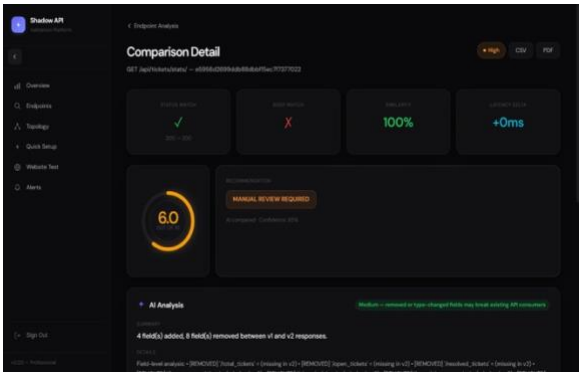

Fig. 3: Comparative Analysis of API Versions (v1 vs v2)

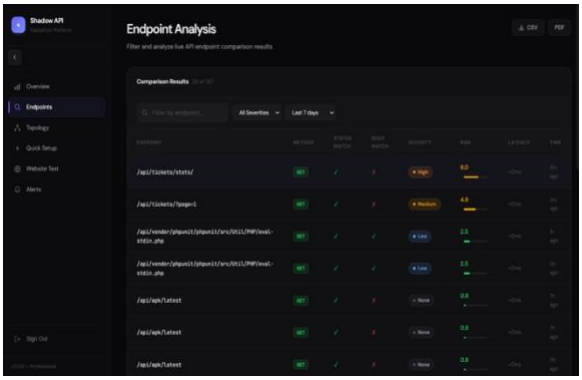

Fig. 4: API Endpoint Comparison Results with Risk and Latency Metrics

V. CONCLUSION

The technique of shadowing gives an ideal method to verify newly developed software versions and machine learning models by using real production data without affecting the end-user experience. Through the use of shadowing, teams can easily identify problems that might not be identified through regular testing methodologies and techniques. Shadowing also becomes an ideal choice for testing complex distributed and data-based applications [9], [17], [18].

While it adds more cost due to increased infrastructure requirements and complexities, these

factors can be addressed by using service meshes, traffic mirroring, and observability platforms. Using automated output comparison, analyzing metrics, and detecting anomalies can help teams make informed decisions on deploying their solutions [13], [14].

When used in conjunction with approaches such as canary release, feature flagging, and CI/CD pipelines, shadow deployment contributes to progressive delivery by providing a constant validation step prior to actual deployment. Shadow deployment is thus instrumental in enhancing deployment security, system resilience, and scalability [15].

REFERENCES

[1] Codefresh, "Shadow Deployment Strategy Explained," Codefresh Blog, 2023.

[2] MLOps Community, "Champion–Challenger Evaluation in Production ML," 2023.

[3] DevOps.com, "Progressive Delivery and Shadow Deployments," 2022.

[4] M. Fowler, "Blue–Green Deployment," martinfowler.com, 2021.

[5] Codefresh, "Safe Release Strategies: Canary, Blue–Green, and Shadow Deployments," 2022.

[6] Zesty.co, "Optimizing Deployment Costs through Traffic Shadowing," 2023.

[7] AWS, "SageMaker Shadow Variant for Model Validation," Amazon Web Services Documentation, 2022.

[8] AWS, "Implementing Shadow Testing in CI/CD Pipelines," AWS Architecture Blog, 2023.

[9] AWS SageMaker, "Production Variant and Shadow Testing," 2022.

[10] M. Fowler, "Feature Toggles and Deployment Patterns," ThoughtWorks, 2021.

[11] Red Hat, "Blue–Green, Canary, and Rolling Update Strategies," Red Hat OpenShift Documentation, 2023.

[12] Google Cloud, "Shadow Traffic Testing in Cloud Run," 2023.

[13] InfoQ, "Advanced Deployment Techniques with Service Mesh," 2023.

[14] Databricks, "Champion–Challenger Model Validation Framework," 2022.

[15] Google Research, "Shadow Testing for ML Pipelines," 2022.

[16] Istio, "Traffic Mirroring Configuration," Istio Documentation, 2023.

[17] Twitter Engineering, "Diffy: Testing Services Without Writing Tests," 2019.

[18] AWS Lambda@Edge, "Traffic Mirroring and Shadow Deployments at the CDN Layer," 2023.

[19] Jenkins X, "Progressive Delivery with Shadow Deployments," 2022.

[20] Envoy Proxy, "Shadowing and Mirroring Traffic," Envoy Documentation, 2023.

[21] GitLab, "Shadow Deployment Pipelines," GitLab Docs, 2023.