International Journal of Computer Applications (0975 - 8887)
Volume 186 - No.50, November 2024

Innovative CI/CD Pipeline Optimization through Canary and Blue-Green Deployment

Sudheer Amgothu
Technology Professional, Department of Computer Science, Pega Systems Inc, USA
7 Gorham St unit 18 Chelmsford MA 01924 USA

ABSTRACT

In modern software development, managing the risks associated with new code releases is critical to maintain system stability and performance. **Canary** and **Blue-Green deployments** are two effective strategies used to achieve this balance. **Canary deployments** allow for incremental rollouts of new features to a small subset of users, providing real-time performance monitoring and risk mitigation before a full-scale release to the production environment while **Blue-Green deployments** involve maintaining two separate environments, blue and green and switching between them to ensure that new changes do not disrupt the production environment. This paper explores the methodologies behind Canary and Blue-Green deployments, their integration with Jenkins CI/CD pipelines, and the benefits, experimental set up, challenges, and future advancements of these strategies. By leveraging Jenkins to manage these deployment approaches, organizations can enhance their deployment processes, reduce the risk of disruptions, and improve overall system stability.

Keywords

Deployment Strategies, Canary Deployment, Blue-Green Deployment, CI/CD, Continuous Integration, Continuous Delivery, Jenkins, Containerization, Kubernetes, Safe Releases.

1. INTRODUCTION

The adoption of CI/CD pipelines has revolutionized how organizations develop, test, and deploy software. These pipelines automate and streamline the release process, enabling rapid updates and enhancements. But, the expanded frequency of deployments introduces ability risks, which includes bugs, performance issues, and provider disruptions. To mitigate these risks, Canary and Blue-Green deployment strategies offer structured approaches for managing and validating software changes.
A canary release allows new features to be released to a small subset of users before they are fully released to production, allowing teams to evaluate performance and stability. On the other hand, the blue-green layout is to maintain two separate environments (blue and green) and transfer traffic between them to distribute the changes with minimal risk. Integrating these strategies into Jenkins CI/CD pipelines increases the efficiency and security of the deployment process.
While CI/CD pipelines streamline the release process, they also necessitate robust deployment strategies to manage the risks associated with frequent changes. Canary and Blue-Green deployments provide complementary methods for safely deploying new features and infrastructure.

2. LITERATURE REVIEW

Deployment strategies such as Canary and Blue Green have become increasingly popular in modern CI/CD pipelines due to their ability to reduce risk during system deployment. These methods are designed to balance the need for continuous delivery with the requirement for system stability.
Canary distribution, first formalized by Humble and Farley [6], involves releasing a new version to a small subset of users before expanding it to a wider audience. This approach minimizes the impact of potential problems by isolating them in the operating environment. According to Raili et al. [9], Canary deployments are particularly valuable in microservices architectures, where individual service failures can be caught early, preventing system-wide issues.
Blue-Green Deployment, also detailed by Humble , Farley and Michael [8] [6], uses two identical environments—Blue (current) and Green (new)—to switch user traffic between versions. If issues arise, traffic is reverted to the Blue environment, ensuring near-zero downtime. Yang et al [12] highlighted the simplicity of Blue-Green for mission-critical systems, although it can be resource-intensive due to the need for duplicate environments.
A study, such as Raili et al [9], found that Canary deployments offer better risk mitigation for gradual rollouts, while Blue-Green deployments are more efficient when rapid rollbacks are critical. Both strategies, however, require sophisticated monitoring and infrastructure to be fully effective, as noted by Yury et al. [7].
In summary, Canary deployments excel at gradual exposure, reducing the risk of widespread failures, while Blue-Green deployments ensure quick recovery and minimal downtime, making both strategies vital components of modern CI/CD pipelines.

3. METHODOLOGIES

3.1 Canary Deployment Methodology

Canary deployment is a strategy for gradually releasing new features or updates to a small subset of users, allowing teams to test the changes in a real-world environment before making them available to the entire user base. The process begins by identifying a ”canary group”—a small, representative set of users who will receive the new version of the application while the rest of the users continue using the previous version. [1]This controlled rollout allows the team to monitor how the update performs in terms of user experience, system performance, and error rates. By leveraging monitoring tools and gathering user feedback, teams can evaluate the stability and success of the update in real-time. If the canary deployment shows positive results, the feature is gradually rolled out to a larger group until it's eventually deployed across the entire user base. However, if issues are detected, a rollback mechanism allows the team to quickly revert the canary group to the previous version, minimizing the impact of potential problems. This methodology helps organizations deploy updates with minimal risk by testing them incrementally in a production-like environment.

3.2 Blue-Green Deployment Methodology

Blue-Green deployment [5] [10]is another effective strategy designed to reduce downtime and deployment risks by operating two identical environments—often referred to as ”blue” and ”green.” In this setup, the blue environment is the active, live version that serves all user traffic, while the green environment is used to stage and test the upcoming release. The deployment process begins with the new version being deployed to the green environment, where it undergoes comprehensive testing and validation to ensure functionality, performance, and reliability. This includes running automated tests, performing manual checks, and validating that the configuration is consistent. When the green environment is fully validated and ready for production, traffic is seamlessly switched from the blue environment to the green environment using load balancers or DNS adjustments. This approach makes the transition invisible to users, providing a smooth cutover with little to no downtime. After the switch, the team monitors the green environment closely for any unforeseen issues. If issues arise, they can immediately revert traffic back to the blue environment, offering a safe rollback option. Blue-Green deployments offer a robust way to ensure continuity and reliability by providing an easy transition path between old and new versions, making it ideal for applications that require high availability.

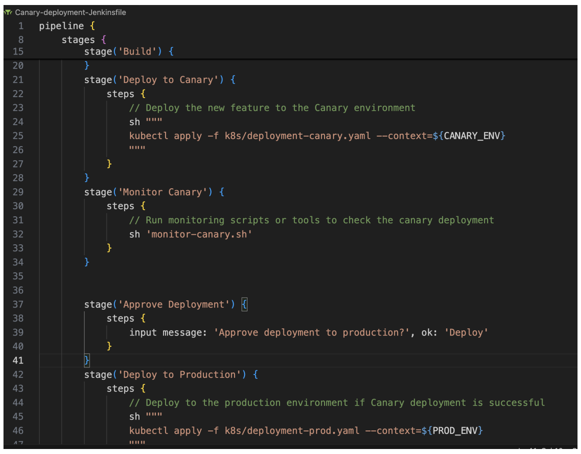
4. INTEGRATING CANARY AND BLUE-GREEN DEPLOYMENTS WITH JENKINS CI/CD

Jenkins is a popular open-source automation provider that helps implement CI/CD pipelines [2]. [3]It presents robust assistance for automating the build, test, and deployment strategies. Integrating Canary and Blue-green deployment strategies with Jenkins can decorate the reliability and performance of software program releases.

4.1 Canary Deployment with Jenkins

In a Canary deployment, new features are released to a small subset of users before a full-scale rollout. Jenkins can automate this process by defining a pipeline that manages incremental feature releases and integrates monitoring and rollback mechanisms. [4]
High level steps during the Jenkins Set up:

(1) **Checkout Code:** Checks out the latest code from version control.
(2) **Build:** Builds the application (e.g., using Maven).
(3) **Deploy to Canary:** Deploys the application to the Canary environment using Kubernetes.
(4) **Monitor Canary:** Runs monitoring scripts to validate the Canary deployment.
(5) **Approve Deployment:** Waits for manual approval before proceeding to the production deployment.

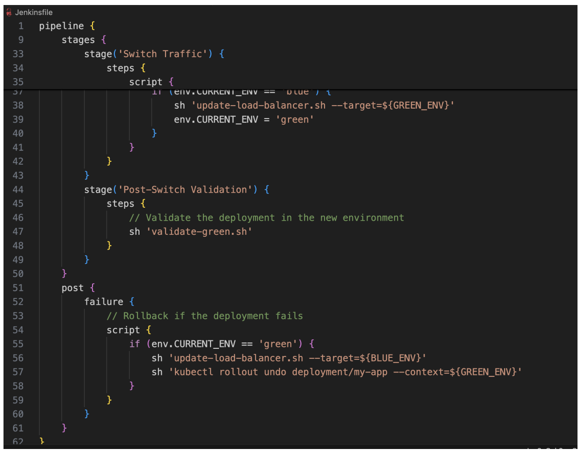


Canary-deployment-Jenkinsfile

```
pipeline {
    stages {
        stage('Build') {
        }
        stage('Deploy to Canary') {
            steps {
                // Deploy the new feature to the Canary environment
                sh """
                kubectl apply -f k8s/deployment-canary.yaml --context=${CANARY_ENV}
                """
            }
        }
        stage('Monitor Canary') {
            steps {
                // Run monitoring scripts or tools to check the canary deployment
                sh 'monitor-canary.sh'
            }
        }


        stage('Approve Deployment') {
            steps {
                input message: 'Approve deployment to production?', ok: 'Deploy'
            }
        }
        stage('Deploy to Production') {
            steps {
                // Deploy to the production environment if Canary deployment is successful
                sh """
                kubectl apply -f k8s/deployment-prod.yaml --context=${PROD_ENV}
                """
```

Fig. 1. **Jenkinsfile for Canary Deployment**



Jenkinsfile

```
pipeline {
    stages {
        stage('Switch Traffic') {
            steps {
                script {
                    if (env.CURRENT_ENV == 'blue') {
                        sh 'update-load-balancer.sh --target=${GREEN_ENV}'
                        env.CURRENT_ENV = 'green'
                    }
                }
            }
        }
        stage('Post-Switch Validation') {
            steps {
                // Validate the deployment in the new environment
                sh 'validate-green.sh'
            }
        }
    }
    post {
        failure {
            // Rollback if the deployment fails
            script {
                if (env.CURRENT_ENV == 'green') {
                    sh 'update-load-balancer.sh --target=${BLUE_ENV}'
                    sh 'kubectl rollout undo deployment/my-app --context=${GREEN_ENV}'
                }
            }
        }
    }
}
```

Fig. 2. **Jenkinsfile for Blue-Green Deployment**

(6) **Deploy to Production**: Deploys the application to the production environment if Canary testing is successful.
(7) **Post-Deployment Verification:** Verifies the production deployment.
(8) **Failure Handling:** Rolls back the deployment if any stage fails.

4.2 Blue-Green Deployment with Jenkins

In a Blue-Green deployment, two identical environments (blue and green) are maintained. Traffic is switched between these environments to deploy new features with minimal disruption. Jenkins can automate the management of these environments and the traffic switch.
High level steps during the Jenkins Set up:

(1) **Checkout Code:** Checks out the latest code from version control.
(2) **Build:** Builds the application (e.g., using Maven).
(3) **Deploy to Green**: Deploys the application to the Green environment.
(4) **Verify Green:** Runs validation scripts to ensure the Green environment is working correctly.

(5) **Switch Traffic**: Updates the load balancer to direct traffic to the Green environment.
(6) **Post-Switch Validation:** Verifies that the Green environment is handling the traffic correctly.
(7) **Failure Handling:** Rolls back to the Blue environment if the Green deployment fails.

5. EXPERIMENTAL SETUP

5.1 Introduction to the experiment

In this experiment, we evaluate the performance of Canary and Blue-Green deployment strategies in a CI/CD pipeline by deploying a new version of a web application. The objective is to assess key performance indicators (KPIs) such as system uptime, error rate, rollback rate, and user experience impact during the deployment of the new application version.

5.2 Environment and Tools

The experimental environment is designed to simulate real-world conditions under which the deployment strategies are tested. The following tools and infrastructure were used: [11]

(1) **CI/CD Pipeline:** Jenkins for continuous integration and deployment.
(2) **Containerization & Orchestration:** Kubernetes was used to run application container microservices.
(3) **Monitoring Tools:** Prometheus was integrated with Grafana to view system metrics such as system uptime, error rate, and CPU/memory usage.
(4) **Load Balancer:** Nginx to manage traffic distribution between old and new application versions.
(5) **Cloud Infrastructure:** The application was deployed on AWS using EC2 instances and Elastic Load Balancing for Blue-Green deployment and traffic management.

5.3 Application Setup

The web application is comprised of three key microservices: authentication, data retrieval, and user interface. The new version of the application included improvements to the UI and updates to the data retrieval service's API structure.

(1) **Old version:** Deployed and running in production with 100% of user traffic.
(2) **New version:** Prepared for gradual rollout and testing during the experiment.

5.4 Deployment Strategies

Canary Deployment
The objective of this deployment was to roll out the new version incrementally, minimizing risk and identifying potential issues early on. The process began in **Week 1**, with the new version deployed to 10% of users, while the remaining 90% continued using the previous version. This initial step allowed the team to observe the performance of the new version on a small user group. In **Week 2**, the traffic was increased to an even 50% split, with half of the users on the new version and half on the old version, enabling the team to gather more comprehensive data on the update's stability. Finally, in **Week 3**, the new version was rolled out to 100% of users, completing the deployment.

Throughout each stage, key metrics such as error rate, system uptime, rollback rate, and user feedback were closely monitored. Prometheus was used to scrape metrics from both versions, enabling the team to track and respond to any increases in errors, downtime, or user complaints. This gradual rollout and thorough monitoring helped ensure a stable transition while reducing the risk of widespread issues.

Blue-Green Deployment
The objective of this deployment was to perform a complete switch from the old version to the new version, with the flexibility to immediately roll back if any issues arose. The procedure began on **Day 1**, when the new version (Blue) was deployed in parallel to the existing version (Green), while 100% of user traffic continued to run on the Green version. On **Day 2**, the team shifted all user traffic to the Blue version, making it the live environment. By **Day 3**, the team closely monitored the performance of the Blue environment, with the capability to revert traffic back to Green if any critical issues emerged.
During this period, essential metrics such as error rate, system uptime, rollback rate, and user feedback were tracked to ensure the stability of the new deployment. Grafana dashboards provided a clear comparison of metrics between the Blue and Green environments throughout the switch, enabling the team to swiftly detect and address any discrepancies. This careful monitoring process helped ensure a seamless transition and minimized the risk of disruptions.

5.5 Traffic Management

For Canary deployment, Nginx was configured to manage incremental traffic distribution between the old and new versions. In the Blue-Green deployment, AWS Elastic Load Balancer managed the switch of all user traffic between environments.

5.6 KPIs Monitored

Throughout the experiment, the following KPIs were collected to evaluate both deployment strategies:

(1) **Error Rate:** Percentage of failed requests during the deployment phases.
(2) **System Uptime:** Percentage of time the system remained operational without downtime.
(3) **Rollback Rate:** Percentage of times a rollback was triggered due to critical errors.
(4) **User Experience Impact:** Measured through user feedback and reported issues post-deployment.

5.7 Testing Procedure

The tests were carried out over a three-week period for Canary deployment and a three-day period for Blue-Green deployment. The following steps were repeated for each phase:

(1) **Traffic Routing:** Adjusted traffic levels between the old and new versions according to the deployment strategy.
(2) **Metrics Collection:** Collected system performance data from Prometheus every 10 minutes.
(3) **Issue Monitoring:** Monitored user feedback and bug reports to assess the impact on user experience.
(4) **Rollback Triggers:** In case of significant errors, rolled back the new version to ensure system stability.

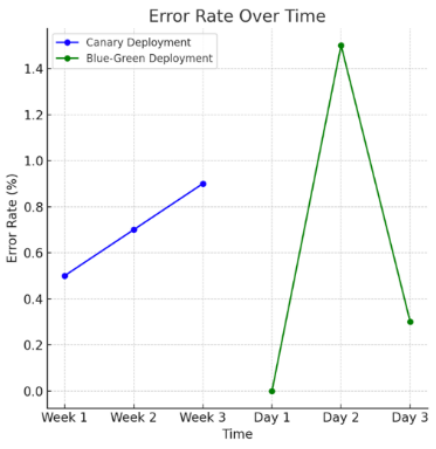


Fig. 3. **Error Rate Over Time for Canary and Blue-Green Deployment**

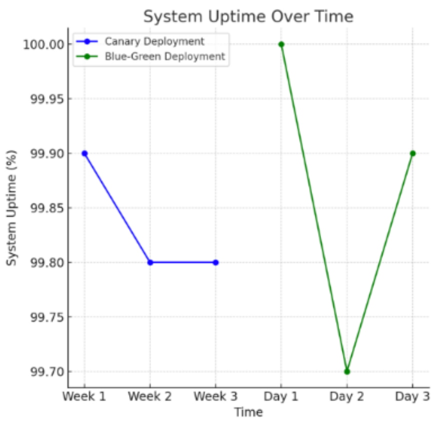


Fig. 4. **System Uptime Over Time for Canary and Blue-Green Deployment**

5.8 Data Collection

All metrics were collected using Prometheus and visualized through Grafana dashboards. The error rate, system uptime, and rollback rate were recorded for each stage of the experiment. Users experience feedback was collected through in-app reporting and external monitoring tools that measured the frequency and nature of complaints.
The graphs below depict the error rate and system uptime across different deployment strategies over time.

5.9 Results and Comparison

The table(1) summarizes the real-time outcomes of both Canary and Blue-Green deployments for a web application:
Overall, the integration of Canary and Blue-Green deployments through Jenkins CI/CD pipelines provides a robust framework for managing software and infrastructure changes. The software company's ability to roll out new features and upgrades with confidence and stability was significantly enhanced, leading to operational efficiencies and improved user experiences.

6. CHALLENGES AND SOLUTIONS

6.1 Challenges

In the experimental deployment setup using Canary and Blue-Green strategies, several technical and operational challenges were observed:
Infrastructure duplication (blue-green): Maintaining two parallel environments significantly increased infrastructure costs. The cost of cloud resources for replication environments can be prohibitive for long-term use, especially for small teams or projects with limited budgets. **Dynamic Routing and Load Balancing (Canary):** Canary deployment relies on accurate traffic routing and load balancing. It is difficult to achieve traffic distribution in the Canary version without affecting performance or creating network bottlenecks. **Monitoring and release decision:** Continuous monitoring is important in canary and blue-green distributions, but it is difficult to set the right thresholds for release decisions. It should be fine-tuned to determine whether deployment is sustainable, or regressing based on real-time metrics such as failure rate and deployment response time. **User Experience Impact:** Although the Canary application helped isolate the issues, users are still at risk with the new release. This has led to concerns about customer satisfaction for those in the Canary team who may experience poor performance or errors. **Delayed Launch (Canary):** The rolling nature of Canary releases can delay the entire release, especially when multiple changes are required to resolve issues. This slowed down the overall speed of the release and delayed the release of critical updates.

6.2 Solutions

To mitigate these challenges, the following solutions were implemented in the experimental setup:
Dynamic resource allocation: For green-blue deployments, a dynamic resource allocation mechanism was implemented to handle the increase in traffic as the environment changes, and to reduce infrastructure costs. This was achieved by using Kubernetes organized container environments, which enabled rapid scaling and resource sharing.
Automated Traffic Management (Canary): Using service mesh technologies (such as Istio) enables automated traffic management and simplifies the process of increasing traffic to a Canary deployment. This ensured a smooth transition with minimal performance degradation.
Advanced monitoring and automatic rollback: Implemented a comprehensive monitoring system using Prometheus for real-time metrics collection and Grafana for visualization. In addition, automatic recovery triggers were set based on pre-defined error thresholds to ensure rapid decompression without manual intervention.
Parallel release strategy: To reduce the duration of Canary releases, parallel release strategies were used for critical updates. This includes running Canary and Blue Green at the same time for different parts of the system, to speed up the overall deployment and maintain control over potential risks.
User Feedback Loops: To reduce user impact, user feedback loops were incorporated into the monitoring system, allowing for early detection of performance or performance issues and provide more accurate feedback triggers based on real-world user experiences.

Table 1. Deployment Data Summary

Deployment Type	Timeframe	Traffic Distribution	Error Rate	System Uptime	Rollback Rate	User Experience Impact
Canary Deployment	Week 1	10 % new version, 90% old version	0.5%	99.9%	5% rollback for canary users	Minimal Small group affected
Canary Deployment	Week 2	50% new version, 50% old version	0.7%	99.8%	No rollback	Slight increase in reporter issues
Canary Deployment	Week 3	100% new version	0.9%	99.8%	No rollback	Full rollout successful
Blue-Green Deployment	Day 1	100% on green (old version)	0%	100%	N/A	No Impact
Blue-Green Deployment	Day 2	100% traffic switched to Blue (new version)	1.5%	99.7%	10% rollback to Green	Moderate – initial surge in errors
Blue-Green Deployment	Day 3	100% on Blue	0.3%	99.9%	No rollback	Stable post-fix deployment

7. FUTURE WORK

To further optimize deployment strategies, several avenues for future work were identified:
AI/ML Integration for Rollout Optimization: Implement machine learning models to predict deployment outcomes based on historical metrics and user behavior data. This would allow automated decision-making regarding the pace of rollout and whether to proceed or rollback.
Hybrid Deployment Models: Explore the potential of hybrid deployment strategies that combine the benefits of both Canary and Blue-Green. For instance, Canary could be used for early-stage testing, followed by a Blue-Green approach for full rollout, minimizing the risks of both methods.
Chaos Engineering Practices: Introduce Chaos Engineering to test the resilience of both Canary and Blue-Green deployments under extreme conditions, such as network outages, traffic spikes, or system failures. This helps improve system robustness and reduce deployment risk.
Serverless Infrastructure: Explores the use of serverless technologies for delivery, especially in blue-green strategies. This can eliminate the need for additional environments, reduce costs and simplify the deployment process.

8. CONCLUSION

The experimental setup demonstrated the relative strengths and weaknesses of both Canary and Blue-Green deployments in a CI/CD pipeline. The release of the Canary is useful for more visibility so that issues can be identified and resolved. However, the length of the entire production process and the difficulty of managing traffic are challenges. On the other hand, the blue-green release made the release of all versions more efficient but required large infrastructure resources and increased errors after the change.
These challenges are mitigated by implementing advanced traffic management, automatic backup mechanisms and dynamic resource allocation. Future work will focus on AI-based distribution optimization and hybrid distribution models, to enable more efficient, effective and cost-effective distribution strategies. The results of this experiment show the importance of creativity in optimizing the CI/CD pipeline to meet the different needs of modern software development.

9. REFERENCES

[1] Azeem Ahmad, Kristian Sandahl, Daniel Hasselqvist, and Pontus Sandberg. Information needs in continuous integration and delivery in large scale organizations: An observational study. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, SAC '24, page 1262–1271, New York, NY, USA, 2024. Association for Computing Machinery.

[2] Sudheer Amgothu. An end-to-end ci/cd pipeline solution using jenkins and kubernetes. *International Journal of Science and Research (IJSR)*, 13(8):1576–1578, 2024.

[3] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A software architect's perspective*. Addison-Wesley Professional, 2015.

[4] Ahmed Mateen Buttar, Adeel Khalid, Mamdouh Alenezi, Muhammad Azeem Akbar, Saima Rafi, Abdu H Gumaei, and Muhammad Tanveer Riaz. Optimization of devops transformation for cloud-based applications. *Electronics*, 12(2):357, 2023.

[5] M Fowler. Inversion of control containers and the dependency injection pattern": http://www. martinfowler. com/articles/injection. html. *Captured on July 19th*, 2006.

[6] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[7] Yury Izrailevsky and Charlie Bell. Cloud reliability. *IEEE Cloud Computing*, 5(3):39–44, 2018.

[8] Michael Nygard. Release it!: design and deploy production-ready software. *torrossa*, 2018.

[9] Njegoš Railić and Mihajlo Savić. Architecting continuous integration and continuous deployment for microservice architecture. In *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–5. IEEE, 2021.

[10] Petar Rajković, Dejan Aleksić, Andjelija Djordjević, and Dragan Janković. Hybrid software deployment strategy for complex industrial systems. *Electronics*, 11(14):2186, 2022.

[11] Giridhar Kankanala Sudheer Amgothu. Sre and devops: Monitoring and incident response in multi-cloud environments. *International Journal of Science and Research (IJSR)*, 12(9):2214–2218, 2023.

[12] Bo Yang, Anca Sailer, and Ajay Mohindra. Survey and evaluation of blue-green deployment techniques in cloud native environments. In *Service-Oriented Computing–ICSOC 2019 Workshops: WESOACS, ASOCA, ISYCC, TBCE, and STRAPS, Toulouse, France, October 28–31, 2019, Revised Selected Papers 17*, pages 69–81. Springer, 2020.