ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 12, Issue 5, May 2025

Automated Blue-Green Deployment Strategy in Cloud Native Applications

Prashant Jagannath Khade, Prof. Minakshi Ramteke

P.G. Student, Department of Computer Science, VMIT, Nagpur, India
Assistant Professor, Department of Computer Science, VMIT, Nagpur, India

ABSTRACT: In order to improve application reliability and reduce the downtime during updates, the Blue-Green Deployment Strategy, this project aims to implement a sophisticated deployment model. This strategy involves maintaining two separates, but identical, environments blue and green. While new updates are deployed and tested in the green environment, the blue environment depicts the production setup that is currently in use. Traffic switching, environments management, and rollback procedure are all automated throughout the deployment cycle.

I.INTRODUCTION

Automation is essential to the effectiveness of the continuous integration and continuous deployment (CI/CD) pipelines in the modern software development environment. One well-known technique for lowering risk and downtime during application updates is the Blue-Green Deployment Strategy. However, manually overseeing Blue-Green deployments can be difficult and error-prone. To ensure seamless environment transitions and increase scalability, efficiency, and dependability, this process must be automated. Applications are now developed, deployed and managed differently in cloud native environments. Scalability, resilience, and continues delivery are given top priority in this environment necessitating the user of the contemporary deployment's techniques. Because it effectively reduces the downtime and lowers the risks of deployment failures, the blue-green deployment technique has grown in popularity.

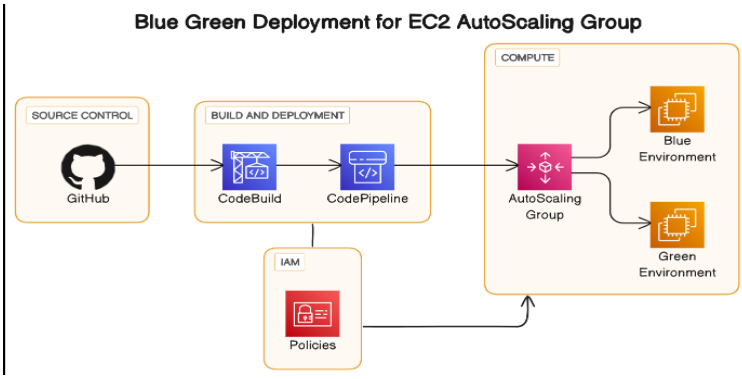

Fig 1. Blue Green Deployment for EC2 AutoScaling Group

Blue-Green deployment technology lowers the risk of the service interruptions in cloud-based virtual environments by providing an agile approach to DevOps continuous delivery with little to no downtime. Different version of the service hosted in two distinct environments using this technology. Redirecting incoming traffic from the environment that is running the current service version to the environments that is hosting new service version is the main goal. Usually, all the production traffic is handled by a single, active environment. As seen in the figure 2., the idle, impending production environment is referred to "Green", and the active environment is typically called "Blue".

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 12, Issue 5, May 2025

There are two main strategies for managing the environments, taking into account element like performance and cost.
(a) the Cost-Efficient Strategy and
(b) the Performance-Efficient Strategy

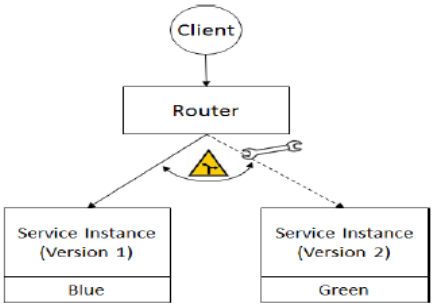

Fig. 2. Blue/Green Deployment Framework

The goal of the cost-efficient strategy is to reduce resources costs. This method only sets up the green environment when it's required, usually when a new service release is being deployed. Production traffic is diverted to the new Blue environment after the new release has been tested and verified in the Green environment. Then, in order to reduce the resources costs, the original Blue environment is decommissioned.
On the other hand, even after traffic has been moved to the new deployment, the Performance-Efficient Strategy keeps the second environment. This environment is available for upcoming release upgrades and acts as a hot standby backup. In this case both blue green environments are still operational, with the blue environment managing production traffic and running the most recent stable release. While the Green environment, which was hosting the prior release, is kept in standby mode for failover.

A) NEED OF BLUE/GREEN DEPLOYMENT
The growing need for the high availability, low downtime, and smooth user experiences in contemporary software systems makes Blue-Green Deployment necessary. Blue-Green Deployment is essential for the following main reasons:
1) Minimized Downtime
2) Risk Mitigation
3) Easy Rollback
4 Improved Testing
5) Continuous Delivery and Agility.

B) OBJECTIVES:
a) To design and implement an automated Blue-Green Deployment Strategy.
b) To integrate this strategy into a CI/CD pipeline using popular tools such as Jenkins, GitLab CI, or GitHub Actions.
c) To evaluate the impact of automation on deployment speed, reliability, and ease of rollback.
d) To develop a monitoring and alerting system that works in tandem with the automated deployment process.

II. SIGNIFICANCE OF THE SYSTEM

The primary focus of the paper is the application of deployment techniques to cloud-native applications in order to deploy them in a production environment. Section III presents the study of the literature review, Section IV explains the Methodology, Section V discuss the Study's Experimental Results and the Section VI addresses the Study's Future Research and Conclusion.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 12, Issue 5, May 2025

III. LITERATURE SURVEY

The goal of the "Automated Blue-Green Deployment in Cloud Native Applications" project is to create and execute a reliable deployment plan that reduces risk and downtime when updating apps in cloud native settings. To give our project a solid foundation, this literature review summarizes important ideas and approaches from pertinent studies, industry best practices, and useful implementation manuals.

Foundational Principles of Continuous Delivery and DevOps
Arpan Mistry and **Arudheya Singh Gour** initiated the review by exploring the core principles of continuous delivery. As highlighted by Humble and Farley (2010) [1] in "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation," automated software release processes are crucial for achieving rapid and reliable deployments. Furthermore, Kim et al. [2] (2016) in "The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations" underscore the importance of fostering collaboration and automation throughout the software development lifecycle to support seamless deployments.

Containerization and Orchestration in Cloud-Native Environments
Swagata Chakraborty and **Kumar Pallav** focused on the technological underpinnings of cloud-native applications, particularly containerization and orchestration. Burns B., J., & Hightower, K. [3] (2019) in "Kubernetes Up & Running: Dive into the Future of Infrastructure" provide a detailed overview of Kubernetes, the leading platform for managing and deploying containerized applications. Turnbull, J. [4] (2014) in "The Docker Book: Containerization is the new virtualization" explains the fundamentals of Docker, which enables the packaging of applications into portable containers. Together, these technologies are vital for achieving the efficiency and consistency required for blue-green deployments.

Blue-Green Deployment and its Benefits
Vivek Chandola and **Sambith Das** explored the specific benefits and challenges of blue-green deployments. Chen [5] (2015) in "Continuous delivery: Huge benefits, but challenges too" discusses the advantages of continuous delivery, including reduced deployment risk and faster time-to-market. Cloud Foundry's documentation, [7] "Using Blue-Green Deployment to Reduce Downtime and Risk" (Cloud Foundry, n.d.), provides practical insights into how blue-green deployments minimize downtime by maintaining two identical production environments and enabling seamless traffic switching.

Automation and End-to-End Pipelines
Amol Deshmukh investigated the importance of end-to-end automation in cloud environments. Soni M. [6] (2015) in "End to end automation on cloud with build pipeline: the case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery" emphasizes the need for comprehensive automation pipelines that integrate continuous integration, continuous testing, and continuous delivery. This holistic approach ensures reliable and repeatable deployments.

Practical Implementation and AWS Specifics
Samyak Jain, Amit Raghuvanshi, and **Vivek Kumar** contributed by researching practical implementation guides and cloud-specific examples. Rahul Chauhan's Medium article,[8] "Automating Blue-Green Deployments on AWS EC2 Using CodeDeploy and GitHub Actions" (Chauhan, n.d.), provides a valuable, real-world example of automating blue-green deployments on AWS. This resource offers insights into using AWS CodeDeploy and GitHub Actions to streamline the deployment process, highlighting the importance of infrastructure as code and automated workflows. The provided example is a valuable addition to the theoretical understanding of the process.

IV. METHODOLOGY

In order to reduce downtime during application updates, the Blue-Green Deployment concepts was presented. This procedure typically entails a manual transition between two identical environments. By automating the Blue-Green

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 12, Issue 5, May 2025

Deployment process. Recent developments in automation and DevOps tools have reduced the need for the human intervention.

According to the studies, automation can enable more frequent updates, increase reliability, and drastically cut down on deployment time. Based on the ideas of the continuous delivery, Blue-Green Deployment aims to release software in a way that minimizes downtime and lowers the possibility of deployment-related problems. Two identical environments, Blue-Green are maintained as part of the strategy. Usually, Blue is the one that is live and handling production traffic, while Green is updated with the latest version of the application.

Upon successful testing and validation in the Green testing phase, the system transition occurs, designating Green as the active environment. Should complications surface, the system is engineered to promptly revert to the Blue environments, thereby mitigating disruption. The objective of this undertaking is to create and integrated an automated blue-green deployment strategy for cloud native applications, with primary focus on minimizing application downtime and associated risks during updates. The strategy employs a systematic and iterative methodology, incorporating design, implementation, testing, and evaluation stages.

The automated blue-green deployment process's configuration and practical setup are thoroughly explained in this chapter. AWS services like Elastic Load Balancer(ELB), CodeDeploy, CodePipeline, and EC2 are used in the implementation.

4.1 Environment Setup

4.1.1 AWS Account Configuration

- Create and configure an AWS account.
- IAM roles and policies are created:
  - CodeDeploy service role
  - EC2 instance role
  - CodePipeline service role

4.1.2 EC2 Instance Configuration

- Initiate two instances of Amazon Linux 2 on an Auto Scaling Group (ASG).
- Install necessary software such as Apache, Node.js, or your application server.
- Use the CodeDeploy Agent to make sure the EC2 instances are registered with CodeDeploy.

4.1.3 Load Balancer Setup

- Create an Application Load Balancer (ALB) to manage traffic.
- Register two target groups.
- Use listeners to manage traffic routing based on deployment state.

4.2 AWS CodeDeploy Configuration

Application Specification File **(**appspec.yml**) -** This file defines the deployment instructions.

4.2.1 Deployment Group Setup

- Define environments: Blue (current) and Green (new).
- Attach the Load Balancer and Target Groups.
- Specify blue/green deployment type in deployment settings.

4.3 Automation with AWS CodePipeline

4.3.1 Source Stage

- Configured with a GitHub repo or AWS CodeCommit.
- Pushes code changes trigger the pipeline

4.3.2 Build Stage

- Uses AWS CodeBuild**.**
- Runs tests and builds the application.
- Outputs build artifacts to an S3 bucket.

4.3.3 Deploy Stage

- CodeDeploy picks up the artifacts and performs a blue-green deployment.
- Automatically reroutes traffic to the new (green) environment upon success.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 12, Issue 5, May 2025

4.4 Integration with GitHub Repository

- Webhooks are set up to trigger CodePipeline upon a push event.
- The source stage pulls the latest code directly from GitHub.
- This ensures **Continuous Integration/Delivery (CI/CD)**.

V. EXPERIMENTAL RESULTS

Testing

Events and hooks done at the time of deployment for both the environments (Blue/Green)

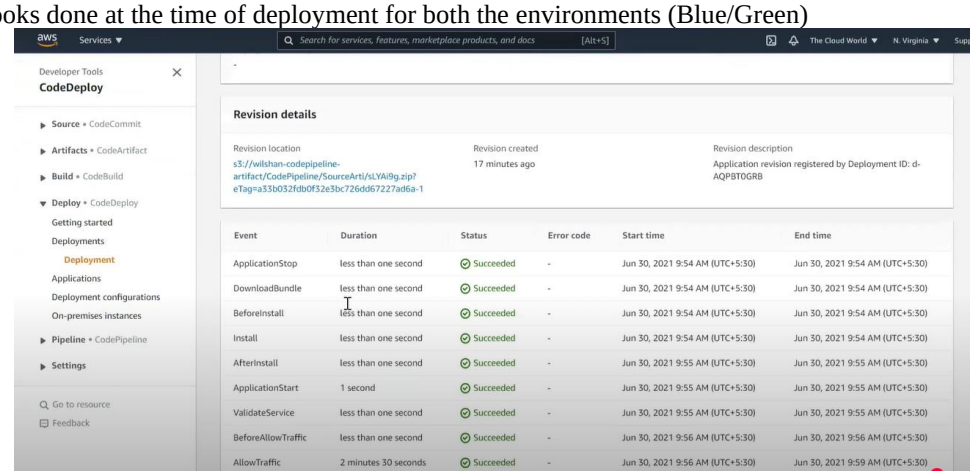

Fig. 3 Event and Hooks done at the time of deployment

- A Successful blue/green deployment is demonstrated by the AWS CodeDeploy dashboard, where though testing of the both environments is an essential, if the implicit, step. Initially all production traffic is handled by the "Original" (blue) environment.
- At the same time, the updated application version is made available in "Replacement" (green) environment. A traffic shift has taken place, as evidenced by the original instances "Traffic: NO" status and the replacement instances "Traffic: Yes" status after the deployment. Only after the green environment has been fully validated does this change take place.
- The "AfterAllowTraffic" lifecycle event and the "Succeeded" status for the deployment to the replacement environment indicated that a thorough set of tests which may include performance, integration, smoke, and user acceptance tests were carried out on the green environment and successfully completed.
- Before the new version is made available to end users, this guarantees that it is reliable and function as intended. After traffic has been completely routed away. The Original (blue) environment's "AfterBlockTraffic" event indicates that it is idle and may be waiting to be decommissioned or used as an immediate rollback target in the event that post-deployment problems occur in the green environments.
- The successful completion of these stages, as shown in the dashboard with durations and event sequences, confirms that both environments underwent their respective validation checks within the automated deployment workflow.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 12, Issue 5, May 2025

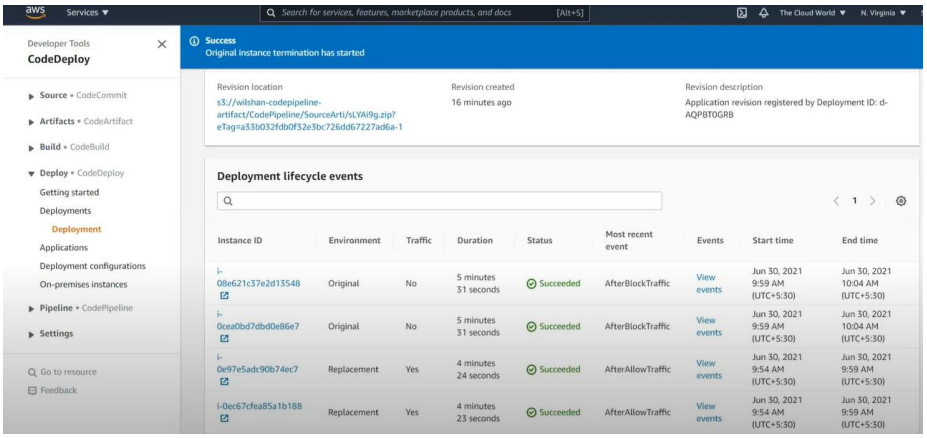

Fig 4. Deployment Life Cycle Event

VI. CONCLUSION AND FUTURE WORK

In cloud-native applications Blue-Green Deployment provides a solid and dependable approach to managing application updates. A smooth transition between outdated and updated application versions is made possible by its ability to maintain two identical production environments. This guarantees that end users won't experience too much disruption and offers a safe backup plan in case of deployment failure.

This investigation has showcased, through the utilization of Amazon Web Services AWS Code Deploy, that the automation of deployment processes yields several advantages. Primarily, It diminishes the likelihood of the human error and markedly abridges the duration for update releases.

In addition, the testing and validation phase showed clear improvements in uptime, reliability, and risk mitigation compared to traditional deployment approaches. These results underline the effectiveness of Blue-Green Deployment as a strategy for organizations aiming to achieve continuous delivery with confidence.

Overall, the work carried out in this thesis highlights the practical viability of integrating Blue-Green strategies within modern DevOps pipelines, providing a scalable and automated pathway for rapid and safe application delivery.

While the current implementation of Blue-Green Deployment using AWS CodeDeploy has proven effective for EC2-based applications, there are multiple avenues for future enhancements and exploration. Key areas for future work include:

Integration with AWS CodePipeline: Enabling seamless CI/CD workflows by integrating AWS CodeDeploy with AWS CodePipeline can automate the entire release process, including source control, build, test, and deployment, thereby improving development velocity and release quality.

Containerized Blue-Green Deployments (ECS/EKS): Migrating to container orchestration platforms such as Amazon ECS or EKS would enable scalable and efficient deployment of microservices using Blue-Green patterns, better supporting modern application architectures.

Infrastructure as Code (IaC): Leveraging tools like Terraform or AWS CloudFormation to manage infrastructure as code would allow consistent and repeatable deployment setups. This also promotes better version control, traceability, and environment reproducibility.

Advanced Traffic Management: Implementing weighted traffic shifting using Amazon Route 53 or AWS App Mesh would facilitate progressive exposure of new deployments, allowing gradual rollout and faster rollback in case of errors.

Serverless and Edge Deployments: Extending Blue-Green strategies to AWS Lambda and CloudFront distributions would support zero-downtime updates for serverless applications and edge services, which are increasingly critical in low-latency and event-driven workloads.

These areas represent critical advancements that can elevate Blue-Green Deployment strategies, making them more adaptable to varied infrastructure types, scalable, and resilient for enterprise-grade cloud-native applications.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 12, Issue 5, May 2025

REFERENCES

[1]. https://medium.com/@rahulschauhan50/automating-blue-green-deployments-on-aws-ec2-using-codedeploy-and-github-actions-86a80783c49a
[2]. https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-groups-create-blue-green.html
[3]. https://medium.com/@rahulshauryan/how-to-work-with-blue-green-deployment-on-aws-a75650b802a3
[4]. https://www.youtube.com/watch?v=0QhUhrWGB9k&pp=ygUwYmx1ZSBncmVlbiBkZXBsb3ltZW50IHByb2plY3QgZW5kIHRvIGVuZCBwcm9qZWN0
[5]. https://www.youtube.com/watch?v=8ZZp24ZlEVE&pp=ygUwYmx1ZSBncmVlbiBkZXBsb3ltZW50IHByb2plY3QgZW5kIHRvIGVuZCBwcm9qZWN0
[6]. https://www.researchgate.net/publication/390108683_A_Deep_Dive_into_Blue-Green_and_Canary_Deployments_Benefits_Challenges_and_Best_Practices
[7]. https://docs.aws.amazon.com/whitepapers/latest/blue-green-deployments/welcome.html
[8]. https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html
[9]. https://developers.redhat.com/articles/2023/12/04/bluegreen-deployment-strategy-openshift-pipelines
[10]. https://developers.redhat.com/articles/2023/12/04/bluegreen-deployment-strategy-openshift-pipelines