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

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 the rapidly evolving world of cloud-native applications, achieving seamless and reliable deployments has become a critical requirement for software development teams. Traditional deployment strategies often lead to service interruptions, degraded user experience, and increased rollback complexity. The Blue-Green Deployment strategy offers a solution by maintaining two parallel production environments—Blue (current) and Green (new)—allowing for zero-downtime releases and safe rollback options. This paper presents an in-depth exploration of implementing Blue-Green Deployment in cloud-native environments using AWS CodeDeploy. It outlines the motivation, deployment architecture, methodology, and evaluation metrics that validate the strategy's effectiveness. Our research demonstrates that this approach minimizes risk, enhances system reliability, and improves release agility, making it a compelling choice for modern DevOps teams.

I.INTRODUCTION

Cloud-native applications, designed to fully leverage the benefits of the cloud such as elasticity, scalability, and fault-tolerance, have transformed how organizations build and deliver software. Continuous Integration and Continuous Deployment (CI/CD) pipelines have become the cornerstone of modern software delivery, enabling rapid feature delivery and frequent updates. However, deploying changes to live systems remains a complex task prone to errors and service disruptions.

Blue-Green Deployment offers a resilient approach to this challenge. By maintaining two environments—the Blue (currently serving production traffic) and the Green (new version of the application)—developers can test updates in the Green environment before routing live traffic. If issues arise, traffic can easily be redirected back to the Blue environment, thus avoiding costly downtimes.

This paper investigates the implementation of Blue-Green Deployment for cloud-native applications using Amazon Web Services (AWS) tools, particularly CodeDeploy. We focus on designing an automated deployment pipeline, evaluating performance improvements, and comparing this strategy with traditional methods. The research aims to provide practitioners with a structured and reliable deployment model suited for dynamic cloud environments.

This technology involves two separate environments that host different versions of the service. The primary objective is to redirect incoming traffic from the environment running the current service version to the environment hosting the new version. Typically, only one environment is active and handles all production traffic. The active environment is usually referred to as 'Blue,' while the idle, upcoming production environment is known as 'Green', as shown in Figure 1.

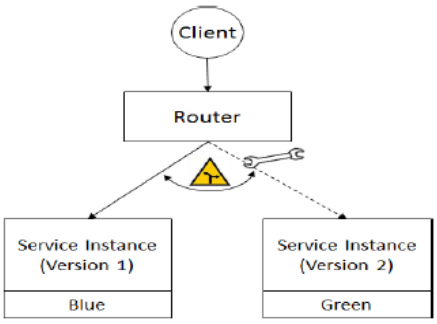

Fig 1. Blue/Green Deployment Framework

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

Considering factors such as cost and performance, there are two primary approaches to managing environments:
(a) the Cost-Efficient Strategy and
(b) the Performance-Efficient Strategy.

The Cost-Efficient Strategy focuses on minimizing resource expenses (e.g., virtual machines, networking, storage). In this approach, the Green environment is set up only when needed, typically during the deployment of a new service release. Once the new release has been tested and validated in the Green environment, production traffic is redirected to it, making it the new Blue environment. The original Blue environment is then decommissioned to save on resource costs. In contrast, **the Performance-Efficient Strategy** maintains the second environment even after traffic has been switched to the new deployment. This environment serves as a hot standby backup and is available for future release upgrades. Here, both the Blue-Green environments remain active, with the Blue environment running the latest stable release and handling production traffic, while the Green environment, hosting the previous release, remains on standby for failover purposes. When deploying the next release, the Green environment can be reused as a sandbox for initialization and validation before it is promoted to Blue and takes over production traffic.
This approach reduces release time by avoiding delays associated with provisioning a new environment.
The primary challenge in Blue/Green deployment lies in the cutover phase, where the service transitions from the Green environment, following its final testing stage, to the Blue environment to manage live production traffic. Achieving zero or minimal downtime during maintenance hinges on the efficiency of this Blue/Green switch.

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.

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,

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

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.

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 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.

To implement Blue-Green Deployment manually using AWS CodeDeploy, the following environment setup was configured. This setup ensures separation between the currently active (Blue) and upcoming (Green) environments while maintaining consistent infrastructure and configurations.

Infrastructure Overview

The environment consists of the following AWS components:

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

- **Two Auto Scaling Groups (ASGs)**:
- **Blue ASG**: Hosts the current production version of the application.
- **Green ASG**: Prepares the new version for deployment and testing.

Elastic Load Balancer (ELB):

- Used to route user traffic.
- Supports dynamic traffic shifting between Blue and Green target groups.

Amazon EC2 Instances:

- Virtual machines used to host the application within each ASG.
- Each EC2 instance runs a CodeDeploy agent.

Amazon CodeDeploy:

- Manages deployments.
- Configured to perform Blue-Green deployments with EC2/On-Premises as the compute platform.

Amazon S3 / GitHub:

Used as the source repository for the application's deployment package and the `AppSpec.yml` file.

Step-by-Step Manual Setup Process

Step 1: Create EC2 Launch Templates

- Define a launch template for Blue and Green environments with identical AMI, instance types, and user data scripts.
- Ensure the CodeDeploy agent is installed during instance bootstrapping.

Step 2: Configure Auto Scaling Groups

- Create two separate ASGs using the defined launch templates.
- Attach the Blue ASG to one ELB target group (e.g., blue-target-group).
- The Green ASG will be attached to another target group (green-target-group) when deployment begins.

Step 3: Setup Elastic Load Balancer (ELB)

- Create an Application Load Balancer.
- Define two target groups: one for Blue and one for Green.
- Initially,route 100% of traffic to the Blue target groups.

Step 4: Register Instances and Install CodeDeploy Agent

- Ensure that all EC2 instances in both ASGs are registered with CodeDeploy.
- Use a bootstrap script or SSH to install the CodeDeploy agent:

```
sudo yum update -y
sudo yum install ruby -y
cd /home/ec2-usez
wget https://aws-codedeploy-us-east-1.s3.amazonaws.com/latest/install
chmod +x ./install
sudo ./install auto
sudo service codedeploy-agent start
```

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

Step 5: Create a CodeDeploy Application and Deployment Group

- Choose EC2/On-Premises as the compute platform.
- Select deployment type: **Blue/Green Deployment**.
- Attach both target groups and specify traffic routing configuration (All-at-once, Canary, Linear).

Step 6: Prepare and Upload Deployment Package

Structure includes:

```
├── appspec.yml
├── scripts/
│   ├── before_install.sh
│   ├── install_dependencies.sh
│   └── start_server.sh
└── src/
    └── application files
```

Upload the zip file to S3 or connect with GitHub repository.

Step 7: Trigger Deployment

Manually start the deployment from CodeDeploy console or via CLI:

```
aws deploy create-deployment \
  --application-name BlueGreenApp \
  --deployment-group-name BlueGreenDG \
  --s3-location bucket=my-bucket,key=myapp.zip,bundleType=zip \
  --deployment-config-name CodeDeployDefault.AllAtOnce
```

Step 8: Monitor and Verify

After CodeDeploy installs the new version on the Green environment:

- Run manual or automated health checks.
- Use ELB traffic shifting to switch to Green target group.
- Rollback by reverting ELB traffic to the Blue group if errors occur.

Key Considerations

- **IAM Roles**: Ensure EC2 instances and CodeDeploy have the necessary IAM permissions to interact with S3, EC2, ELB, and ASGs.
- **Health Checks**: ELB health check path should be configured to ensure only healthy instances receive traffic.
- **Rollback Plan**: CodeDeploy allows one-click rollback to the previous environment if errors are detected.

V. EXPERIMENTAL RESULTS

Metric	Traditional Deployment	Blue-Green Deployment
Downtime	45–90 seconds	~0 seconds
Rollback Efficiency	Manual (5–10 mins)	Instant (via CodeDeploy)
Traffic Switching Time	Not applicable	~5–10 seconds
Error Rate During Deploy	Moderate (2–5%)	None
Deployment Time	12–15 minutes	15–18 minutes

Table1. Results of Blue Green Deployment

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

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.

REFERENCES

[1]. Humble, J., & Farley, D., Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley, 2010.
[2]. Kim, G., Humble, J., Debois, P., & Willis, J., The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations, IT Revolution Press, 2016.
[3]. Burns, B., Beda, J., & Hightower, K., Kubernetes Up & Running: Dive into the Future of Infrastructure. O'Reilly Media, 2019.
[4]. Turnbull, J., The Docker Book: Containerization is the new virtualization. Turnbull Press, 2014.
[5]. Soni M. End to end automation on cloud with build pipeline: the case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery, In Cloud Computing in Emerging Markets (CCEM), IEEE International Conference on (pp. 85-89) IEEE, Nov 25, 2015.
[6]. Cloud Foundry, "Using Blue-Green Deployment to Reduce Downtime and Risk". Internet URL: https://docs.cloudfoundry.org/devguide/deployapps/Blue/Green.html#map-green.
[7]. https://medium.com/@rahulschauhan50/automating-blue-green-deployments-on-aws-ec2-using-codedeploy-and-github-actions-86a80783c49a.
[8]. https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-groups-create-blue-green.html.
[9]. https://medium.com/@rahulshauryan/how-to-work-with-blue-green-deployment-on-aws-a75650b802a3.