A Practical Evaluation of Deployment Strategies for Services Running in the Cloud

Vlad-Stefan DIEACONU
National University of Science and Technology Politehnica Bucharest
Bucharest, Romania
vladstefandieaconu@gmail.com

Alexandru Bobe
Ovidius University of Constanta
Constanta, Romania
alexandru.bobe@univ-ovidius.ro

Iosif-Alexandru Selea
CrowdStrike
Bucharest, Romania
alexselea@gmail.com

***Abstract*—Rapid production rollout of new features, vulnerability patches, and critical bug fixes is essential to maintaining a market advantage in today's competitive landscape. However, this need for speed introduces increased risks, as demonstrated by the 2025 Google Cloud incident, where a software deployment containing an undetected bug resulted in cascading service disruptions in major platforms, including Cloudflare, OpenAI, and Microsoft 365, requiring over seven hours for complete system recovery. Given the possible consequences, it becomes mandatory for organisations to select the most appropriate deployment strategy when rolling out new software code changes. This paper provides a comprehensive review of deployment strategies in modern development lifecycles, analysing both the advantages and limitations of each deployment strategy, as well as its practical considerations, such as complexity and cost. Additionally, we propose a framework for evaluating and comparing different approaches to deploying to production. Examining each approach alongside requirements specific to the service itself, such as urgency and risk tolerance, application complexity, and team expertise, organisations can select the most suitable deployment strategy for their use case.**

***Index Terms*—Software deployment, zero-downtime deployment, canary, blue-green, release management**

I. Introduction

In recent years, due to the growing focus on digitalisation, individuals, businesses, and society as a whole have grown increasingly dependent on cloud services, not only for our entertainment or everyday tasks, but also for critical and time-sensitive operations.

A recent comparative review of cloud adoption statistics published by Edge Delta [1] shows us that more than 94% of companies worldwide are using cloud services in their day-to-day operations, with global cloud computing market reaching values of more than $675.4 billion in 2024 [2]. This widespread adoption means that outages of cloud services can no longer be considered isolated incidents, but can cascade to multiple layers and impact millions of users simultaneously. The outage of Google Cloud from June 2025 [3] demonstrates how a faulty code change to a specific internal service, the Service Control, can evolve into a global outage, resulting in over 1.4 million users affected, disrupting more than 50 online services, possibly including but not limited to OpenAI, Spotify, Snapchat, as found out from an Ookla research [4]. This is not an isolated case in which errors due to software deployments resulted in substantial damages, a defective deployment causing Knight Capital to lose over $440 million in less than 45 minutes [5].

Extensive research conducted by I.M. Aldea on a large number of incidents from Verica Open Incident Database [6] revealed that over half of these incidents are caused by deployments of code changes. In contrast, similar studies conducted by large organizations, such as Microsoft and Google, report software deployments as the common culprit for a significantly lower number of failures. An empirical study of high severity incidents in Microsoft Teams [7] shows that 13% of these incidents were caused by software rollouts to production, while a Google research [8] identified approximately 16% of failures to be related to software deployments. These figures, though still substantial, provide strong evidence that in leading cloud organisations, with clear change management policies and techniques in place, software deployments are responsible for a smaller amount of production outages.

Given the possible risks and financial losses associated with impacting clients due to a faulty production deployment, it becomes critical for organisations of all sizes to establish adequate strategies for software deployments. This paper aims to offer a comprehensive analysis of deployment strategies, providing the necessary knowledge and evaluation framework needed to choose the adequate approach to rolling out code changes to production. The selection of an appropriate deployment strategy is further contextualised by broader cloud migration frameworks that guide how diverse applications can leverage cloud services for scalability and efficiency [9], [10]. The architectural evolution from monolithic systems to microservices, aimed at improving resilience and cybersecurity, directly influences this selection, as it requires more sophisticated and granular roll-out techniques [11]. Furthermore, effective strategies must incorporate robust models for remote updates and safe recovery, particularly in distributed environments such as IoT, to ensure system stability and reliability [12]. At the foundational level, the security of the underlying containerization technologies that enable these deployments

is paramount, with user access policies in clustered systems playing a critical role in mitigating the risks associated with privilege escalation [13].

The following section will explore the state of the art in terms of deployment strategies for services running in the cloud, describing its practical implementations using the available AWS cloud technologies[1], such as EC2, Target Groups, Auto Scaling Groups, and Application Load Balancers. We will also analyse each rollout procedure's suitability for different levels of organisations, based on its complexity, speed, associated costs and more.

II. Software Deployment Strategies

A. All-At-Once Deployments (Big-Bang Deployments)

As the name suggests, the All-At-Once deployment approach updates all servers simultaneously. This inevitably causes downtime because there's a required interval to drain existing traffic, deploy new code, and restart services before accepting production traffic again. Typically, organizations manage this downtime through scheduled maintenance windows, notifying users in advance of planned service unavailability.

All-At-Once Deployments can be implemented using both an in-place or a recreate technique. In-place deployments uses the already existing servers[2] to which it deploys a new application version, containing the software code changes. This offers the solution of a simple architecture, with no need for the overhead associated with instance management, like reserving, creating and terminating instances. On the other hand, recreate deployments start by down scaling the previous fleet of instances, to prepare a clean environment to which new instances will be deployed, alongside the code changes.

For a minimal AWS-based architecture supporting the in-place all-at-once deployments, which simply imply that all existing servers are updated simultaneously, we can use the following setup:

- EC2 instances with the application code are deployed and registered to a Target Group, which is attached to an Application Load Balancer receiving traffic;
- During a deployment, to avoid sending traffic to a Target Group with no healthy hosts, the Load Balancer can be configured to redirect traffic to a maintenance mode landing page;
- All EC2 instances are stopped, configured to use the new version containing the code changes we wish to deploy, and then restarted;
- After passing health checks, the Target Group will have available targets which are ready to receive traffic, at this moment the maintenance window can be ended and the Load Balancer can route all traffic to the Target Group.

For recreate deployments, the approach is similar, with the main difference being the need to terminate the old EC2 instances and start fresh instances. The recommended approach to doing this is by using an Auto Scaling Group, which can scale-down old machines, then be configured with a new launch template to reference the application version containing the code changes we wish to deploy.

Although a strategy resembling the procedure of local testing code changes during the development process, All-At-Once deployments are used not only in development or staging environments, but also in production, for specific use-cases in which availability requirements are permissive enough to accept a maintenance window. It may be used for deploying breaking changes across multiple services which are not backwards compatible, requiring a coordinated release of multiple services at the same time. The operational processes presented can be automated using a pipeline which directly interacts with AWS resources, providing organisations with a low-complexity and cost-effective deployment method that can easily be automated. It is important to note that this approach is based on the assumption that software bugs and potential issues will be caught in an incipient phase, in a pre-production environment, as rolling back requires to repeat the complete process again and perform a new deployment, causing significant downtime in case of an incident.

B. Rolling Deployments

Rolling Deployments maintain the idea of a simple, low-complexity and cost-effective architecture from the previously described Big-Bang deployments, while providing a balance between speed and service availability, by removing the need for a maintenance window. Rolling Deployments operate on the assumption that if code changes have been rigorously tested and are free of critical issues, it is possible to achieve a zero-downtime deployment by simply updating the existing machines serving traffic in small incremental phases [14], as opposed to updating all the servers at the same time, as in the All-At-Once strategy.

There are two different techniques to achieving Rolling Deployments, the first one is based on in-place updating of the existing machines, while the other one is based on immutable deployments, in which new instances are always deployed.

The operational cost of in-place rolling deployments does not increase compared to Big-Bang deployments, as rolling deployments are a standard feature of the Auto Scaling Group, that can be configured with a rolling policy, stating the percentage of instances we want to update at a time after changing its template. Immutable rolling deployments can be achieved by creating a new Auto Scaling Group for the application version we wish to deploy, attach it to the Target Group serving live traffic, and then scale-it up. As instances start passing health checks, they will start receiving traffic and we can continue scaling down instances from the previous Auto Scaling Group.

However, in order to benefit from easier rollbacks, it is mandatory to have an automated process that would stop the deployment and revert the Auto Scaling Group's original configurations.

[1]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

[2]https://docs.aws.amazon.com/whitepapers/latest/practicing-continuous-integration-continuous-delivery/deployment-methods.html

The main disadvantage of Rolling Deployments compared to Big-Bang deployments is the requirement for the application to support two different code versions concurrently in production. This implies that the new application version must be fully backward-compatible with the previous one to prevent disruptions during incremental deployment phases. Therefore, careful version management and rigorous backward compatibility testing are essential for successfully applying this technique.

C. Blue-Green Deployments

The Blue-Green deployment strategy achieves improved deployment speed and possibility for instant rollbacks by maintaining two identical, duplicate production environments, hosting two different versions of the application. This rollout strategy aims to deploy the newer version to production with almost zero-downtime using a procedure named cut-over, as stated in a survey of existing Blue-Green deployment techniques [15]. The cut-over implies sending traffic from one environment having the older application version, the Blue, to the other environment, hosting the newer application version, which we will call the Green. Rolling back to the previous version is almost instant, as it only requires cutting-over to the other environment, which remains at all times scaled up in an idle state, without receiving traffic. During the next production deployment, the roles of the two environments are reversed, Blue becomes the environment having the new application version, to which we will perform the traffic cutover from the Green environment. In most use-cases before switching traffic to the environment containing the code changes we wish to deploy, tests are executed on that idle environment, to ensure the application is healthy and ready to receive traffic.

A simple Blue-Green architecture based on AWS resources consists of:

- Two identical Auto Scaling Groups, configured with the two different versions of the application, one for the Blue, and one for the Green;
- Each Auto Scaling Group has a Target Group, which is attached to the same Application Load Balancer;
- The Application Load Balancer uses listener routes to switch traffic from one environment to the other;

This deployment technique is preferred due to its capability for instant traffic switching, allowing for both fast deployments and rapid rollbacks, while opening the possibility to perform testing in a production environment before sending real traffic to the new servers. There is an increased operational complexity associated with automating the deployment process, and an increased infrastructure-related cost due to having duplicate environments. There are possibilities to balance the need for instant rollbacks with the associated costs by scaling down the inactive environment after deployment finishes and enough time has passed to confirm the stability and performance of the new version, which aligns with AWS best practices for Blue-Green deployments [16]. To optimize resource utilization and minimize costs, it is recommended to scale-down the inactive environment after confirming that the new deployment is stable. Automating this scaling process based on monitoring metrics and stability checks can significantly reduce overhead without compromising rollback speed and safety.

D. Canary Deployments

In contrast to the other deployment strategies presented in this paper, the key difference to Canary Deployments is the ability of this deployment approach to provide a reliable mechanism for testing and validating a new application version under real production traffic before committing to a complete roll-out at scale. From an architectural point of view, we can think of canary deployments as an enhanced Blue-Green deployment strategy, where two Target Groups are used for traffic management. One Target Group represents the active side, having the stable, previously tested build version of the application, and the other Target Group is considered the canary side, which will be used to expose the new build of the application, containing the code changes we want to deploy, to an increasingly larger subset of production traffic. An Application Load Balancer is used to send traffic in incremental steps to the canary side, each steps following a validation procedure, which is usually based on analysing observability data related to the health and performance of the application, step which can be automated using a Canary Analysis Service [17].

A comprehensive study of Continuous Integration and Continuous Deployment strategies conducted by Avishek Singh and Vibhakar Mansotra [18] puts canary deployments under the category of testing strategies, for its effectiveness in detecting outliers and anomalies of a build version, when compared to a baseline. Reliable validation through Canary Deployments demands robust observability tools and practices that provide comprehensive telemetry and real-time performance data from production environments. Without detailed monitoring data to accurately detect and respond to anomalies, Canary Deployments cannot fully leverage their benefits and may become indistinguishable from simpler phased rolling or Blue-Green strategies.

E. A/B Testing Deployments

While Canary Deployments focus mainly on minimizing potential risks, by gradually releasing a new application version under limited amounts of production traffic, A/B Testing Deployments imply a completely different methodology, by focusing on experimentation and optimisation, as stated in a recent systematic literature review [19]. It is important to highlight that A/B Testing is best suited for scenarios aimed at optimizing user experience, performance, or business metrics through experimentation. Unlike Canary Deployments, which primarily verify stability and performance, A/B Testing aims explicitly at choosing among competing feature implementations based on user behavior and engagement outcomes. While both approaches involve routing a percentage of production traffic to a different application version, A/B Testing transforms the deployment process to a continuous experiment, by conducting a simultaneous comparison between multiple

application versions. Whereas the primary goal of canary deployments is to determine whether the new application build, containing the intended code changes we need to deploy, maintains acceptable system health and performance compared to the baseline, which is considered the previous version of the application, the goal of A/B Testing Deployments is to identify, from a multitude of different versions of the application, the one that yields the best results, which typically refers to the most favorable user behavior and engagement metrics.

A simple implementation for A/B Testing Deployments, using AWS available resources, can be achieved using:

- An Auto Scaling Group attached to a Target Group for each version of the application;
- An Application Load Balancer that routes equal percentages of traffic to each Target Group;
- Optionally, it is a good practice to ensure that a user will get routed to the same application version every time. This can be done using sticky sessions with Load Balancer cookies[3]. Using this configuration, on a subsequent request, as long as the user's agent (i.e. browser) sends the cookie, the Application Load Balancer will route the request to the same Target Group, enabling the user to always get the same version of the application.

More intelligent approaches can be used, such as using Route53 header-based geographic routing[4], to ensure users from the same regions will get the same version of the application. As concluded by a research paper on A/B testing frameworks used for software feature releases at scale [20], advanced A/B Testing, such as adaptive algorithms and intelligent traffic allocation, can deliver better results, in the terms of faster and reliable rollouts of new feature to the users, but they also introduce significant infrastructure complexity, and operational overhead, which may be unsustainable for smaller organizations.

F. Shadow Deployments (Dark Launches)

Shadow Deployments, or Dark Launches, involve deploying a new application version alongside the current production version but without exposing it directly to end-users. Real production traffic is duplicated and sent to the new, shadow version, which processes requests and generates responses. These shadow responses are discarded, meaning users experience no changes or interruptions. This technique allows extensive risk-free testing under actual production conditions.

Implementing shadow deployments can prove to be particularly challenging due to the inherent overhead. This approach requires a complete duplicate of the production environment, which will serve the mirrored traffic, leading to significantly higher infrastructure costs. Shadow environments should not write to the shared production databases, requiring separate databases or sandboxed resources. The most important aspect is that the application must implement idempotency of operations, ensuring that repeated or duplicate requests, from the mirrored traffic, will have the same effect as a single request. Any integration with a downstream service or external service should be isolated from the production environment or redirected to test environments. This needs to be done in order to prevent unintended side effects, such as data alterations, duplicate transactions or message dispatching. Common techniques to achieve such isolation include service virtualization, which simulates the behavior of dependent services, and the use of mock endpoints that safely emulate external APIs without impacting production systems.

Duplicating or mirroring traffic to the shadow environment can be accomplished using available built-in AWS resources and technologies, such as using Lambda@Edge and the Application Load Balancer's request mirroring, as suggested in an article written by Timothy Patterson [21], or by using the traffic mirroring feature of the VPC(i.e. Amazon Virtual Private Cloud)[5], which allows network packets to be duplicated and sent to the shadow environment.

Using this complex deployment strategy, the live and the shadow application versions run alongside in production, but the new features are kept in the dark, as the name dark launches suggests, being tested and monitored in production without revealing them to the user. In a research paper on the continuous deployment techniques used at Facebook [22], Tony Savor et al. offer a solution to the overhead associated with adding a specific shadow environment, the code changes that we want to dark launch are deployed on all production servers, but configured in a certain way that end users cannot interact with them, allowing to test for performance and scalability issues. A specific example of such an implementation of testing a new feature using dark launches is described in the paper "Development and Deployment at Facebook" [23]. When initially releasing their chat servers, Facebook deployed the feature with its user interface disabled, while the system itself was active and configured to send automated test messages. This approach enabled Facebook to evaluate the system's stability under real production traffic and address scalability challenges before public release. Once the system was validated and optimized, it was exposed to users simply by enabling the interface and disabling the test message functionality.

III. Evaluation Framework for Deployment Strategies

Selecting a deployment strategy requires a clear understanding of available technologies, operational implications, and the specific business needs of each application. To support objective assessments, we evaluate deployment strategies using the following six key criteria, ensuring that each is considered consistently and explicitly across options:

[3]https://docs.aws.amazon.com/prescriptive-guidance/latest/load-balancer-stickiness/alb-cookies-stickiness.html

[4]https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/api-routing-http.html

[5]https://docs.aws.amazon.com/vpc/latest/mirroring/what-is-traffic-mirroring.html

- Deployment Speed/Duration: Time required to fully deploy to production, measured from the start of the rollout until all users are served by the new application version.
- Downtime: Anticipated interruption or service unavailability during deployment.
- Risk Level: Probability of incidents or failures related to the deployment.
- Rollback Speed: Ease and speed of reverting to a previous stable version.
- Infrastructure Cost: Additional resources required for the implementation of the strategy.
- Operational Complexity: Effort needed for setup, automation, and ongoing management.

While establishing clear evaluation criteria is an important step in building an evaluation framework for deployment strategies, selecting an appropriate measurement system is equally critical. During our research for an intuitive scale with universal understanding, we found out that traditional numeric approaches can often create a false sense of precision, ultimately obscuring meaningful distinctions between the options we wish to evaluate, as teams struggle to discern the difference between, for example, a complexity ranking of seven and an eight. Moreover, absolute measurement systems often suffer from calibration inconsistencies, making it difficult to integrate new options into existing assessments. A research paper by Sidky and Gaafar [24] comes to the conclusion that humans are significantly better at making relative comparisons than at scoring based on absolute estimates, further highlighting that relative estimation enhances overall accuracy.

To address these concerns, this paper adopts the t-shirt sizing[6] methodology as the measurement system for evaluating deployment strategies, due to its intuitive, relative comparison scale, balancing simplicity with detailed and unique characteristics [25]. In Table I, we present a comparative evaluation of the deployment strategies described above as part of this paper, by analysing each deployment technique based on multiple key factors (deployment speed, downtime, risk level, rollback speed, infrastructure cost, and complexity), to support objective selection based on relative trade-offs and specific advantages.

Strategy	Speed	Downtime	Risk	Rollback	Cost	Complexity
All-at-Once	L	XL	XL	XS	S	XS
Recreate	M	XL	XL	XS	S	S
Rolling	M	M	L	S	S	XS
Immutable	M	M	L	S	M	S
Blue-Green	XL	M	M	L	L	M
Canary	S	S	S	M	L	L
A/B	S	S	S	XL	XL	XL
Shadow	S	XS	XS	XL	XL	XL

Legend: X=Extra, S=Small, M=Medium, L=Large.

TABLE I: Comparison of Deployment Strategies Based on Operational Criteria (T-Shirt Sizing)

IV. Conclusions

Selecting the most suitable deployment strategy always depends on specific contextual factors. Security strategies within cloud deployment architectures often intersect with graph-based privacy concerns, as explored by Costea et al. [26] in their analysis of differential privacy mechanisms on graph structures. For critical production services with strict availability requirements, safer zero-downtime strategies like Canary, A/B Testing, or Shadow Deployments provide robust mechanisms to manage risk, isolate changes, and enable rapid rollbacks. These benefits typically come with higher complexity and infrastructure costs, but significantly reduce operational risks. In contrast, smaller or internal services that can tolerate scheduled downtime or occasional deployment risks might prefer simpler, lower-cost options like Rolling Deployments, which require less automation effort.

To illustrate the practical utility of the proposed evaluation framework, consider a high-traffic e-commerce platform preparing to launch a new payment feature. The engineering team must choose between a Canary deployment and a Blue-Green deployment. Consulting the t-shirt sizing evaluation table, the team notes that:

- Canary deployment is ranked as S for speed, S for downtime, S for risk, M for rollback speed, L for infrastructure cost, and L for complexity.
- Blue-Green deployment is ranked as XL for speed, M for downtime, M for risk, L for rollback speed, L for cost, and M for complexity.

From these ratings, the team can quickly see that Canary offers lower downtime and risk (both S) while maintaining moderate rollback agility (M), though at a higher infrastructure cost and complexity (L, L) compared to some other strategies. Blue-Green, on the other hand, delivers the fastest rollout (XL) with easy rollback (L), but it comes with moderately higher downtime and risk (M, M) and similar infrastructure cost (L). Using these concise t-shirt size metrics, the team can weigh the trade-offs: Canary may be preferable if minimizing operational risk is the top priority, whereas Blue-Green is the stronger choice if deployment speed and rollback simplicity are more important for the business context.

Moreover, many organizations combine multiple deployment strategies to capitalize on each method's strengths and mitigate risks. In practice, a common industry pattern is to implement a Canary release within a Blue-Green infrastructure: new changes are first deployed to a parallel ("green") environment, and traffic is gradually shifted to the new version using canary analysis. This approach enables teams to validate new releases with a subset of users in the green environment, providing early feedback and risk mitigation, before performing a complete switchover for all users if the canary phase is successful. Such hybrid strategies allow for both rapid, easily reversible rollouts and incremental user exposure, maximizing reliability while maintaining agility.

[6]https://asana.com/resources/t-shirt-sizing

Ultimately, in today's competitive landscape, selecting a deployment strategy requires effective change management planning and a careful balance of factors such as speed, reliability, complexity, and cost. By applying the t-shirt sizing evaluation framework, shown in Table I, businesses and individuals can compare deployment strategies and select the one that best fits their specific use case.

REFERENCES

[1] Edge Delta. (2024, Jun.) How many companies use cloud computing in 2025? [10 statistics and insights]. Accessed: 2025-07-11. [Online]. Available: https://edgedelta.com/company/blog/how-many-companies-use-cloud-computing

[2] Gartner. (2024, May) Gartner forecasts worldwide public cloud end-user spending to surpass $675 billion in 2024. Accessed: 2025-07-11. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2024-05-20-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-surpass-675-billion-in-2024

[3] Google Cloud. (2024, Jun.) Incident 24006: Issue with google cloud infrastructure in us-east5. Accessed: 2025-07-20. [Online]. Available: https://status.cloud.google.com/incidents/ow5i3PPK96RduMcb1SsW

[4] Ookla. (2025, Jun.) Major google cloud outage impacts online services around the globe. Accessed: 2025-07-18. [Online]. Available: https://www.ookla.com/articles/google-cloud-outage-june-2025

[5] N. Popper. (2012, Aug.) Knight capital says trading mishap cost it $440 million. Accessed: 2025-07-18. [Online]. Available: https://archive.nytimes.com/dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/

[6] I. Aldea, "Understanding software failures through incident report analysis: An empirical study of 348 incident reports from the void," Master's thesis, Delft University of Technology, 2025. [Online]. Available: https://repository.tudelft.nl/record/uuid:e8da60fe-3db0-4a02-ac2f-5c7b2859f7ee

[7] S. Ghosh, M. Shetty, C. Bansal, and S. Nath, "How to fight production incidents?: An empirical study on a large-scale cloud service," in *Proceedings of the 13th ACM Symposium on Cloud Computing (SoCC '22)*. New York, NY, USA: ACM, Nov. 2022, pp. 40–53. [Online]. Available: https://doi.org/10.1145/3542929.3563482

[8] K. S. Yim, "Norming to performing: Failure analysis and deployment automation of big data software developed by highly iterative models," in *IEEE International Symposium on Software Reliability Engineering*, 2014, pp. 144–155.

[9] A. Alexandrescu and C. Mironeanu, "A framework for anything-as-a-service on a cloud platform," in *2023 27th International Conference on System Theory, Control and Computing (ICSTCC)*, 2023, pp. 333–338.

[10] O. Agapie, A. Alexandrescu, and D. Turcanu, "Cloud-based distributed solution for optimizing the search for tourist destinations," in *2024 23rd RoEduNet Conference: Networking in Education and Research (RoEduNet)*, 2024, pp. 1–6.

[11] C. Contasel, R. V. Rughiniș, D. C. Trancă, and D. Tsurcanu, "Enhancing e-health cybersecurity and resilience: shifting from monolithic to microservices architecture," *U.P.B. Scientific Bulletin, Series C*, vol. 87, no. 1, pp. 21–34, 2025.

[12] A. Radovici, I. Culic, D. Rosner, and F. Oprea, "A model for the remote deployment, update, and safe recovery for commercial sensor-based iot systems," *Sensors*, vol. 20, no. 16, p. 4393, Aug. 2020.

[13] I. M. Stan, D. Rosner, and S. D. Ciocirlan, "Enforce a global security policy for user access to clustered container systems via user namespace sharing," in *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, 2020, pp. 1–6.

[14] C. K. Rudrabhatla, "Comparison of zero downtime based deployment techniques in public cloud infrastructure," in *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 2020, pp. 1082–1086.

[15] B. Yang, A. Sailer, and A. Mohindra, "Survey and evaluation of blue-green deployment techniques in cloud native environments," in *Service-Oriented Computing – ICSOC 2019 Workshops*, S. Yangui, A. Bouguettaya, X. Xue, N. Faci, W. Gaaloul, Q. Yu, Z. Zhou, N. Hernandez, and E. Y. Nakagawa, Eds. Cham: Springer International Publishing, 2020, pp. 69–81.

[16] Amazon Web Services, "Blue/green deployments on aws," https://docs.aws.amazon.com/whitepapers/latest/blue-green-deployments/introduction.html, Sep. 2021, accessed: 2025-07-18.

[17] Štěpán Davidovič, B. Beyer, N. Desai, and L. Zhang, "Canary analysis service: Automated canarying quickens development, improves production safety, and helps prevent outages," *Queue*, vol. 16, no. 6, pp. 20–39, Jun. 2018. [Online]. Available: https://dl.acm.org/doi/10.1145/3194653.3194655

[18] A. Singh and V. Mansotra, "A comparison on continuous integration and continuous deployment (ci/cd) on cloud based on various deployment and testing strategies," *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, vol. 9, no. VI, pp. 4968–4973, Jun. 2021. [Online]. Available: https://www.ijraset.com/fileserve.php?FID=36038

[19] F. Quin, D. Weyns, M. Galster, and C. C. Silva, "A/b testing: A systematic literature review," *Journal of Systems and Software*, vol. 211, p. 112011, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121224000542

[20] G. Anand, S. K. Lingishetty, and N. Gupta, "Large-scale a/b testing frameworks for improving software feature rollouts," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 13, no. 3, 2025, open Access, Peer-reviewed, Refereed Journal. [Online]. Available: https://ijcrt.org

[21] T. Patterson. (2025) Shadow deployments on aws part 1: Lambda@edge. Accessed: 2025-07-19. [Online]. Available: https://cloudy.dev/article/shadow-deployments/

[22] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at facebook and oanda," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. Austin, TX, USA: IEEE, 2016, pp. 21–30.

[23] D. Feitelson, E. Frachtenberg, and K. Beck, "Development and deployment at facebook," *Internet Computing, IEEE*, vol. 17, pp. 8–17, 07 2013.

[24] A. Sidky and A. Gaafar, "The mindset behind estimating and planning for agile," in *PMI Global Congress 2014—EMEA*, Project Management Institute. Dubai, United Arab Emirates: Project Management Institute, 2014, paper presented at PMI Global Congress 2014–EMEA, Newtown Square, PA.

[25] Teamhub. (2024) Understanding relative sizing in software development: A comprehensive overview. Accessed: 2025-07-19. [Online]. Available: https://teamhub.com/blog/understanding-relative-sizing-in-software-development-a-comprehensive-overview/

[26] S. Costea, M. Barbu, and R. Rughinis, "Qualitative analysis of differential privacy applied over graph structures," in *2013 11th RoEduNet International Conference*, 2013, pp. 1–4.