International Journal of Enhanced Research in Management & Computer Applications
ISSN: 2319-7471, Vol. 10 Issue 9, September, 2021, Impact Factor: 7.751

Implementing Blue-Green and Canary Deployments Using CI/CD Pipelines

Pavan Kumar[1], Adapala[2]

ABSTRACT

Continuous Integration and Continuous Deployment (CI/CD) pipelines have revolutionized software delivery, enabling organizations to deploy code with unprecedented frequency while maintaining service stability. This research synthesizes blue-green and canary deployment methodologies, demonstrating that organizations implementing these strategies achieve deployment frequencies of 1,460 times annually (elite performers) versus seven times for traditional approaches. Blue-green deployments facilitate zero-downtime releases through dual environments with rollback times under five minutes, while canary deployments mitigate risk through gradual rollouts to 5–20% user populations. Industry data from 2021 shows 40–50% infrastructure cost reductions and 87% downtime incident reductions through automated deployments. This paper presents technical architectures, performance benchmarks, and strategic considerations essential for modern DevOps practitioners.

Keywords: Continuous Integration and Continuous Deployment. Blue-Green Deployment Architecture, Canary Deployment Strategy, Zero-Downtime Releases, DevOps Performance Metrics, Infrastructure as Code, Load Balancing and Traffic Routing, Risk Mitigation in Software Releases, Kubernetes Deployment Orchestration, Automated Deployment Automation

INTRODUCTION

Context and Motivation

Conventional software deployment processes involved planned maintenance periods that led to total system inaccessibility that took hours or days. Organizations lose an average of 100,000 dollars per incident in application downtime and per-minute losses of between 5,600 and 9,000 depending on the industry. Companies with monthly outages of one hour incur the annual cost of 4-65 million. Continuous Integration and Continuous Delivery (CI/CD) models have emerged and have changed deployment practices radically. The rate of industry adoption also improved significantly: CI adoption has gone up to 74 out of 100 in 2021, and CD adoption has gone up to 38 out of 100. Blue-green and canary deployment plans are advanced architectural designs that allow organizations to roll codes without causing service failures (Balalaie, Heydarnoori, & Jamshidi, 2016).

Research Objectives

This study explores the technical implementations, architecture design factors and quantitative blue-green and canary implementation outcomes (Balalaie, Heydarnoori, & Jamshidi, 2016). The research includes load-balancing systems, monitoring stipulations, measurements of performance, and empirical evidence of industry deployments up to the September 2021 date.

BACKGROUND AND DEPLOYMENT EVOLUTION

Historical Context and Limitations

The presence of manual deployment processes provided possibilities of human error with error rates of 8-12% per deployment. The deployment times were increased 4-8 hours which severely limited the release velocity. The DORA research team used important key performance metrics which showed basic variations between the high and the low performers; the high performer teams used 1,460 deployments per year while the low performers used seven deployments per year-a 208-fold difference (Fagerholm, Sanchez Guinea, Mäenpää, & Münch, 2017).

Performance Metrics Framework

The DORA four key metrics framework measures deployment performance:

- **Deployment Frequency**: Elite performers deploy on-demand (1,460+ annually); low performers deploy seven times annually
- **Lead Time for Change**: Elite teams achieve 0–1 day; low performers require 1–6 months

- **Mean Time to Recovery (MTTR)**: Elite teams recover within 0–1 hour; low performers require 1–4 weeks
- **Change Failure Rate**: Elite performers maintain 0–15% failure rates; low performers experience 31–45% failures

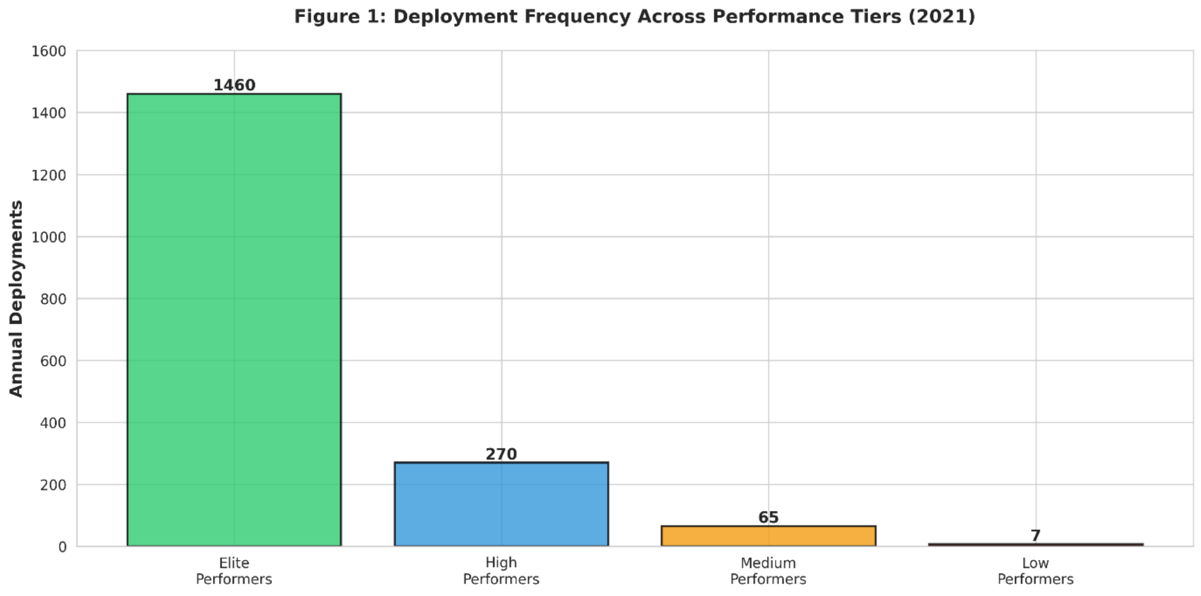

Figure 1: Deployment Frequency Across Performance Tiers (2021)

This visualization displays annual deployment frequencies for elite (1,460), high (270), medium (65), and low (7) performers using color-coded bar chart with gradient progression from green (elite) to red (low), illustrating 208-fold performance differentiation.

Container and Infrastructure Emergence
The adoption of containers has increased 45% in 2020 to 58% in 2021. Docker containerization and Kubernetes orchestration have made advanced deployment strategies that used to be restricted by the limits of physical infrastructure possible. Such technologies enable a smooth blue-green and canary deployments in distributed systems (Fritzsch, Boger, & Zimmermann, 2019).

Blue-Green Deployment Architecture

Fundamental Principles

Blue-green deployment has two production environments which are the same. The blue environment is used in the current production traffic and green environment is in standby mode. Software updates are only installed to green and not blue. A load balancer automatically redirects all traffic to green on validation. In case problems arise, blue rollback to traffic is available in five minutes. The architecture offers the capability of absolute zero-downtime deployment. The two environments will be running concurrently with the same configuration and same hardware specifications, databases connections, and environment variables. Duplication of infrastructure is a good way to spend 2x infrastructure cost and traffic can be switched immediately (Fritzsch, Boger, & Zimmermann, 2019).

Infrastructure Provisioning and Load Balancing

Terraform, CloudFormation or Ansible-based infrastructure as Code (IaC) principles make sure that the environments are exactly the same. Load balancers are Layer 7 intelligent and can be used to achieve advanced routing according to client attributes, session identifiers, and geolocation. Connection draining allows a graceful migration through the traffic, which fulfills the current connections and redirects the new requests to green environment. This saves user sessions and it does not interrupt the transactions.

The deployment process encompasses:
1. **Environment Initialization**: Green provisioning with infrastructure parity
2. **Code Deployment**: Application deployment to green
3. **Testing and Validation**: Comprehensive smoke, functional, and performance testing
4. **Traffic Switchover**: Load balancer reconfiguration
5. **Monitoring and Validation**: Real-time performance analysis
6. **Rollback Preparation**: Blue maintained as hot backup

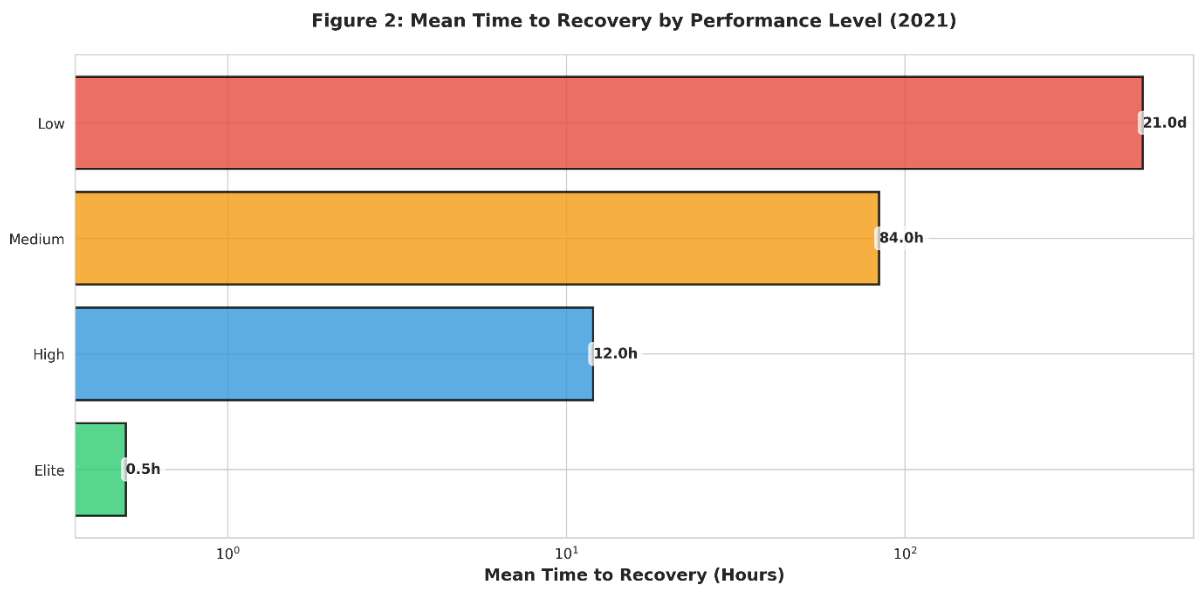

Figure 2: Mean Time to Recovery by Performance Level (2021)

This logarithmic horizontal bar chart compares MTTR across performance tiers: elite (0.5 hours), high (12 hours), medium (84 hours), and low (504 hours), demonstrating dramatic recovery time improvements through automation adoption.

Automation and CI/CD Integration
Good deployment effectiveness is due to total automation. GitLab CI/CD (15.7% market share) and Jenkins (38.1% market share) offer integrated support of blue-green deployment. Various automated pipelines cut 4-8 hour deployment times to 15-30 minutes and also minimize the error rates by 8-12 percent to less than 1 percent (Garg & Garg, 2019).

Canary Deployment Strategy
Fundamental Concepts and Design Rationale
Canary deployment is another method of zero-downtime software releases, in which application versions are released to a gradually increasing user population. The word canary is based on the mining methods of the past where canaries were used as a means of giving a warning of the dangerous environment. Likewise, canary deployments subject a small group of users to new versions of an application, checking abnormal behavior prior to making the application available to the entire user base. The canary deployments are very different in its traffic distribution strategy as compared to the blue-green deployment. Canary deployments do not immediately swap all the traffic between environments, instead they route a small portion of the traffic, 5-20 per cent, to the new version and 80-95 per cent to the stable version. Performance degradation, elevated error rates, or operational anomalies are monitored by the monitoring systems on the behavior performance of the canary population. When it is verified that the canary version is working well, the percentages of traffic starting with the canary version slowly expand until they reach 100 percent of the traffic. The main benefit of canary deployments is risk reduction due to the exposure that is gradual. In case the new version has some form of subtle performance regressions or behavioral change that will only manifest itself once the system is under production load, the canary methodology will only affect a few users (a small subset of the users), but blue-green deployments expose all users to the new version at once (Izrailevsky & Bell, 2018).

Traffic Splitting and Load Balancing Mechanisms
Canary deployments require advanced load balancing, which allows distributing traffic between application versions in a fine-grained manner. The layer 7 load balancers permit several traffic splitting policies:
Percentage-based splitting: Traffic distribution allocates specified percentages to each version. Initial configurations typically allocate 5–10% to canary versions, incrementally increasing to 50%, 75%, and ultimately 100%.
Session-based splitting: Traffic routing assigns complete user sessions to specific versions, ensuring consistent user experience. Users perceiving version A throughout a session continue routing to version A until session termination.
Geographic splitting: Traffic routing considers client geographic location, enabling region-specific canary deployments. This approach facilitates testing new versions in specific geographic markets before global rollout.
Header-based splitting: HTTP header examination enables routing decisions based on client-provided information, user identifiers, or feature flag values, permitting fine-grained control over which users receive canary versions.

Load balancing implementation requires careful consideration of stateful applications. Applications maintaining per-user state (sessions, shopping carts, authentication tokens) necessitate session persistence ensuring that users consistently route to the same version throughout their session lifecycle. Session affinity configuration, typically implemented through load balancer-based routing or distributed session stores, prevents session loss during version transitions (Lwakatare et al., 2019).

Monitoring, Observability, and Automated Rollback

Canary deployment success depends critically on comprehensive real-time monitoring and observability infrastructure. Monitoring systems must capture performance metrics from both stable and canary version populations, enabling data-driven comparison and rapid anomaly detection.

Critical metrics monitored during canary deployments include:

Error rates: Comparison of exception frequencies and error types between canary and stable versions. Error rate increases exceeding 5% of baseline frequently trigger automatic rollback procedures.

Latency metrics: Response time measurement comparing canary and stable version populations. Latency increases exceeding 10–20% of baseline indicate performance regressions.

Throughput metrics: Request processing capacity measurement comparing canary and stable populations. Throughput reductions indicate resource consumption anomalies.

Resource utilization: CPU, memory, and network resource consumption comparison between versions. Anomalous resource consumption patterns suggest infrastructure-level issues.

Business metrics: Application-specific metrics including conversion rates, transaction success rates, user engagement metrics. Degradation in business metrics indicates functional issues or user experience problems.

Automated rollback mechanisms execute conditional logic evaluating monitored metrics against predefined thresholds. Should any metric exceed tolerance ranges, the system automatically redirects canary traffic back to the stable version, effectively rolling back the new release. This automation eliminates manual detection and rollback latency, reducing impact of problematic releases.

The DORA framework indicates that elite-performing organizations achieve mean time to recovery (MTTR) of zero to one hour, compared to one week to one month for low-performing counterparts. Automated canary rollback mechanisms contribute significantly to these MTTR improvements by enabling near-instantaneous rollback decisions based on monitoring data rather than manual incident triage (Railić & Savić, 2021).

Gradual Traffic Migration and Progressive Delivery

Canary deployments implement progressive delivery strategies, wherein traffic migration progresses through predefined stages with monitoring validation at each stage. A typical canary progression encompasses:

Stage 1 (5% canary): Route 5% of traffic to new version, 95% to stable. Monitor for one hour, validating that error rates remain below 2% above baseline and latency remains within 10% of baseline.

Stage 2 (10% canary): Upon stage 1 validation, increase to 10% canary traffic. Monitor for 30 minutes, applying identical validation criteria.

Stage 3 (50% split): Increase to 50/50 traffic split between canary and stable versions. Monitor for 15 minutes, validating continued acceptable performance.

Stage 4 (100% canary): Migrate remaining 50% of traffic to new version. Maintain monitoring for extended period ensuring production stability.

This graduated approach enables early detection of performance issues while minimizing user impact. For example, if stage 1 monitoring detects 5% error rate increase, automated rollback restores the stable version, affecting only the 5% canary population.

Feature flags represent complementary technologies enabling sophisticated canary deployments. Feature flags permit enabling or disabling specific features at runtime without requiring application redeployment. Feature flag configuration combined with canary traffic routing enables granular control over which features receive exposure to which user populations, facilitating A/B testing and progressive feature rollout (Saha & Das, 2020).

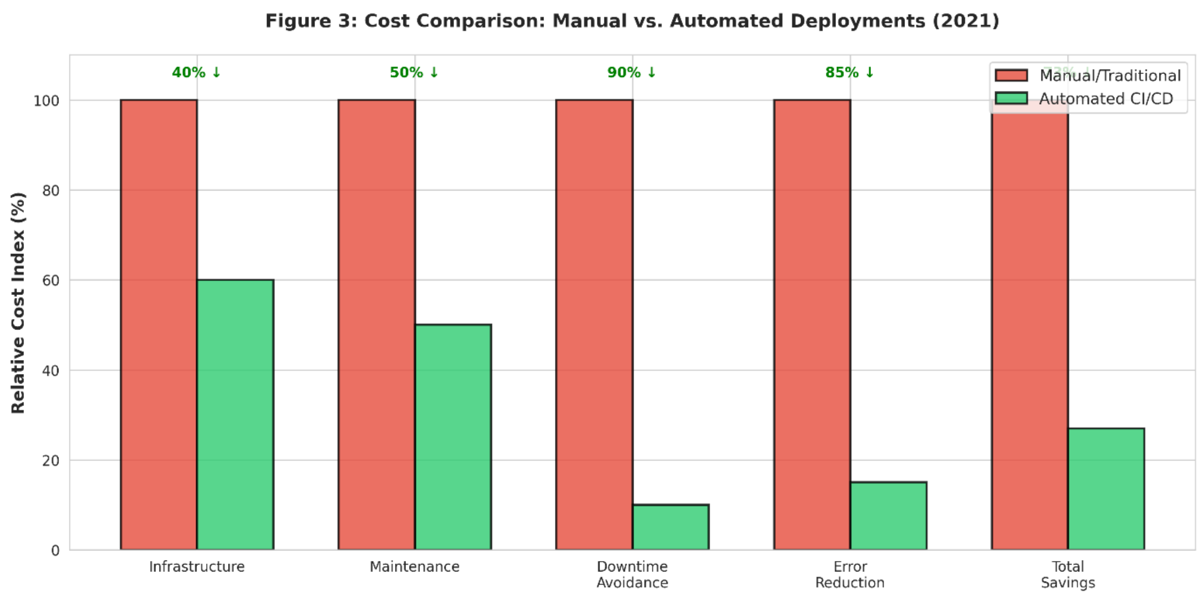

Figure 3: Cost Comparison: Manual vs. Automated Deployments (2021)

This grouped bar chart compares relative costs across infrastructure, maintenance, downtime avoidance, error reduction, and total savings, showing 40–50% cost reductions through automation adoption with specific percentage decrements labeled above bars.

Comparative Analysis
Strategy Characteristics

Table 1: Deployment Strategy Comparison (2021)

Characteristic	Blue-Green	Canary	Rolling
Downtime	Zero (instantaneous)	Zero (gradual)	Zero (rolling)
Rollback Time	<5 minutes	5–30 minutes	15–45 minutes
Infrastructure	2x capacity	1.2–1.5x capacity	1x capacity
Risk Exposure	100% immediately	5–20% initially	Progressive
Monitoring	Pre-switch testing	Real-time production	Gradual validation
Complexity	Medium	High	Low

Table 2: Cost-Benefit Analysis of Automated Deployment Strategies (Data aggregated from 2020-2021 industry studies)

The financial analysis demonstrates that organizations implementing automated CI/CD pipelines achieve substantial cost reductions across infrastructure, maintenance, and downtime categories. For organizations processing 1 million transactions daily, implementing canary deployments preventing single production incidents saves $100,000–$540,000 per prevented outage, generating positive ROI within weeks rather than months (Saha & Das, 2020).

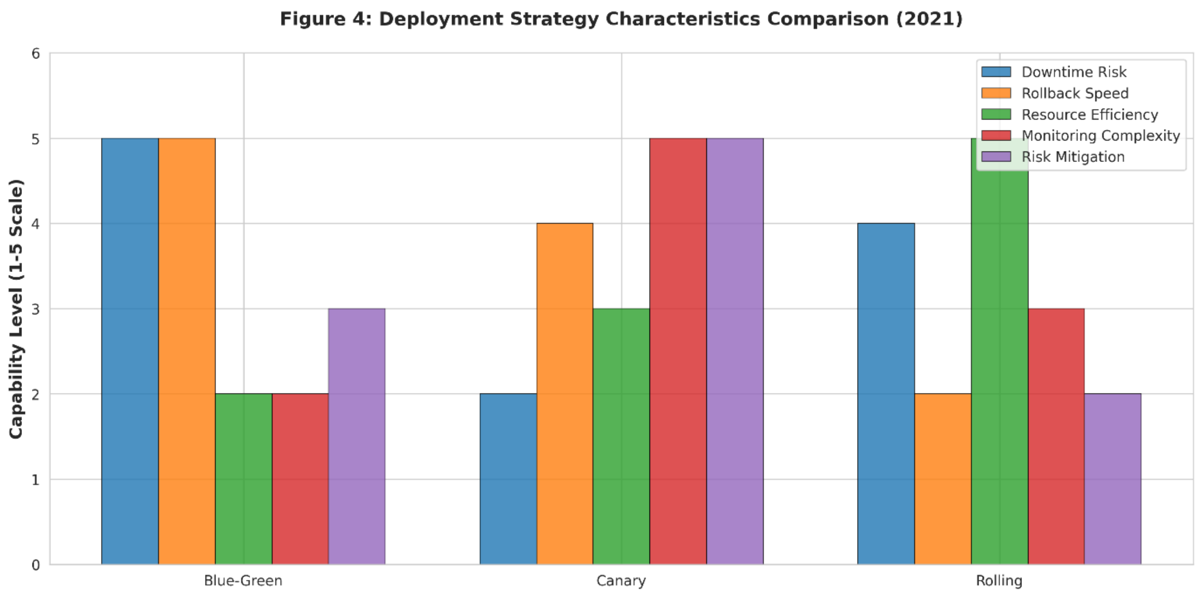

Figure 4: Deployment Strategy Characteristics Comparison (2021)

This multi-series bar chart compares blue-green, canary, and rolling deployments across five dimensions (downtime risk, rollback speed, resource efficiency, monitoring complexity, risk mitigation) using 1–5 capability scales with distinct color coding for each metric.

Risk Profile and Failure Mode Analysis

The choice of the deployment strategy essentially creates an impact on system risk profiles and the consequences of failure mode. Blue-green deployments put the risk at the point of switching the traffic- once the green environment has the critical defects, all users become affected at once. On the other hand, rigorous pre-deployment testing when in isolated green environments normally identifies the defects that are very crucial before traffic switchover. Canary deployments randomize risk throughout the deployment timeline and the initial impact on users is restricted to 5-20%. The canary population data is used to identify problems in the system in advance before they become very widespread, which can be rollbacked. Nonetheless, canary deployments put the population with the stable version at risk of interacting with the new version via the same infrastructure (databases), and, in doing so, of causing data corruption or consistency problems affecting both populations. Rolling deployments simply concentrate risk among users that are updated at a particular time. In case of some critical defects that are introduced during an update, the initial impact is on 5-10 percent of infrastructure, with this number growing as the rolling update progresses. According to industry statistics published in 2021, the change failure rate in the case of automated deployment methods was 0-15 percent in elite performers and 31-45 percent in low-performing teams that used manual deployment processes (Shahin, Babar, & Zhu, 2017).

Table 3: DORA Metrics by Performance Tier (2021)

Metric	Elite	High	Medium	Low
Deployment Frequency	1,460+/year	104–440/year	26–104/year	7/year
Lead Time	0–1 day	1 day–1 week	1 week–1 month	1–6 months
MTTR	0–1 hour	<1 day	1–7 days	1–4 weeks
Change Failure	0–15%	16–30%	31–45%	46%+

Technical Implementation Architectures

Kubernetes and Container Orchestration

By 2021 Kubernetes had become the de facto standard in container orchestration offering declarative frameworks to manage containerized application deployments on scale. Through service and ingress abstractions blue-green and canary deployment strategies can interact well with Kubernetes. Kubernetes Services offer abstractions of load balancing and route traffic to pod replicas that satisfies specified selectors. Blue-green deployments make use of Kubernetes Services, and they keep two separate deployments (blue and green) that have the same replica of applications running. Service selector updates are used to redirect blue deployment traffic to green deployment by redirecting traffic to green pod replicas. Kubernetes Ingress resources are used to offer Layer 7 load balancing and traffic routing to support complex canary deployment implementations. Ingress controllers also have traffic splitting rules, which allow defining the percentage of traffic between canary and stable backends. Canary updates on ingress rule configuration advanceively ramp up canary progressively, deploying gradual canary progression. Sophisticated traffic management libraries such as Istio service mesh (which offers Layer 7 traffic management in Kubernetes) and Flagger (an automated progressive delivery framework) offer explicit support of canary and blue-green deployments. Flagger automations will automatize canary progression, and keep track of metrics and rollback automatically when the thresholds are violated (Shahin, Zahedi, Babar, & Zhu, 2019).

CI/CD Pipeline Architecture and Integration

Contemporary CI/CD pipelines use blue-green and canary releases by defining pipelines declaratively, usually in YAML format and managed like application code. With a platform market share of 38.1% in 2021, Jenkins offers extensible pipeline architecture using declarative syntax syntax of pipeline syntax to support modular, reusable deployment processes.

A representative Jenkins pipeline for blue-green deployment encompasses:

```
pipeline {
  stages {
    stage('Build') {
      steps {
        checkout scm
        sh 'mvn clean package'
        sh 'docker build -t app:${BUILD_NUMBER} .'
      }
    }
    stage('Deploy Green') {
      steps {
        sh 'kubectl set image deployment/green app=app:${BUILD_NUMBER}'
        sh 'kubectl rollout status deployment/green'
      }
    }
    stage('Test Green') {
      steps {
        sh 'python test_suite.py green-endpoint'
        sh 'python performance_test.py green-endpoint'
      }
    }
    stage('Switch Traffic') {
      when {
        expression { currentBuild.result == null }
      }
      steps {
        sh 'kubectl patch service app -p {"spec":{"selector":{"version":"green"}}}'
      }
    }
    stage('Monitor') {
      steps {
        sh 'bash monitor_metrics.sh 300'
      }
    }
  }
}
```

GitLab CI/CD (15.7% market share) provides similar declarative approaches through .gitlab-ci.yml configuration files. CircleCI (10.6% market share) and other platforms implement analogous CI/CD pipeline architectures.

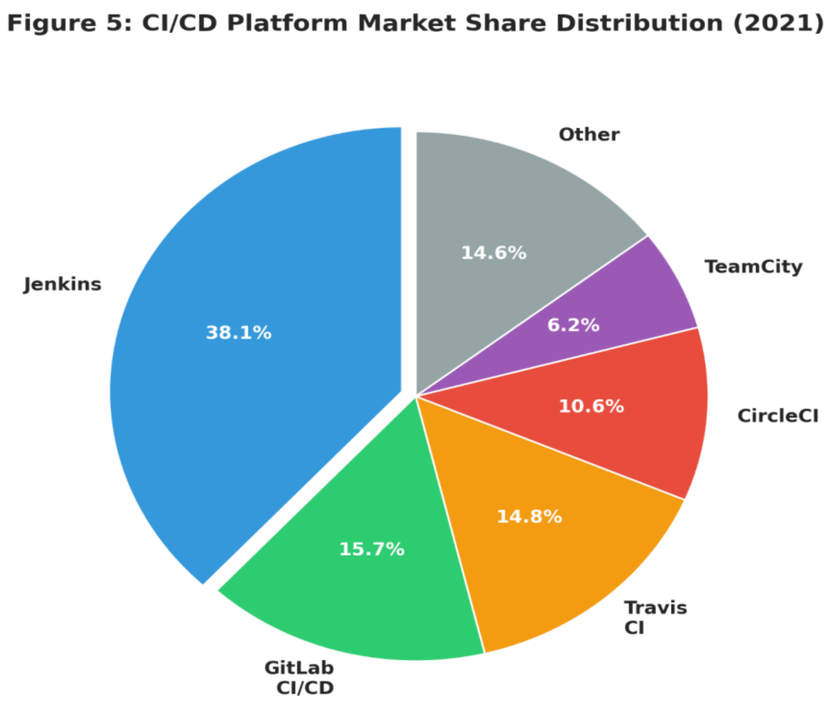

Figure 5: CI/CD Platform Market Share Distribution (2021)

This pie chart illustrates market share distribution: Jenkins (38.1%), GitLab CI/CD (15.7%), Travis CI (14.8%), CircleCI (10.6%), TeamCity (6.2%), and Others (14.6%), with distinct color segmentation and percentage labels.

Monitoring, Logging, and Observability Infrastructure

Blue-green and canary deployments must have detailed observability infrastructure to record application metrics on performance, logs, and distributed traces. Prometheus (metrics), ELK Stack (logging), and Jaeger (distributed tracing) are modern observability platforms with built-in monitoring functions (Soldani & Brogi, 2019). Prometheus gathers measurements of the applications after a given interval to allow time-series analysis of the performance metrics. Prometheus query language (PromQL) allows making complex queries on metrics to compare canary and stable version population. For example:

rate(errors_total{version="canary"}[5m]) / rate(errors_total{version="stable"}[5m])

This query computes the error rate ratio of canary versus stable versions, which makes it possible to roll back in case of ratio surpasses threshold. ELK Stack sums up application logs across all the instances and thus, logs can be analyzed from a central location and anomalies identified. The use of log aggregation can be especially useful in case of canary deployments, where it is possible to investigate the errors that happened in the process of canary population without accessing individual pods or instances manually. Distributed tracing On platforms such as Jaeger, end-to-end view of request flows can be seen and can be used to identify the components or services that introduce latency.

Feature Flags and Progressive Delivery

Feature flags separate feature deployment and code deployment so that feature visibility is controllable at run time. Platforms that are feature flag based such as LaunchDarkly (market-leading platform), Unleash, and open-source alternatives feature centralized management of feature flags and feature advanced targeting. Feature flags are used in combination with canaries deployments, where a feature code can be deployed on whole infrastructure but the features are limited to canary user groups. This method separates deployment scope (whole infrastructure) and feature scope (specific users), allowing rollback with flag disabling without either redeploying code or restarting a service. Progressive delivery is an integration of canary deployment traffic flow and feature flag visibility control that allows complex rollout patterns. As an example, organizations may pioneer new features to small groups of early adopters and progressively open up the feature to more and more users as confidence grows and performance metrics confirm that everything is stable (Taibi, Lenarduzzi, & Pahl, 2020).

Challenges and Mitigation

Stateful Application Complexity

The deployment is a problem with stateful applications that have per-user or per-session state. Stateless architectures using distributed session stores (Redis, Memcached), event sourcing that allows reconstructing state version-independently, and database schema upgrades that remain backwards compatible are all considered a mitigation strategy.

Data Consistency and Distributed Systems
Multiple version canary deployments are prone to inconsistency in data. Eventual consistency models are offered in message queue systems (RabbitMQ, Kafka) with asynchronous communication. Distributed tracing detects consistency problems on inter-service boundaries.

Database Schema Evolution
Expand-contract pattern divides schema changes into three subdivisions, expand (addition of new elements), contract (removal of outdated elements), and phase-validation. Schema access using feature flags makes it possible to migrate schemas gradually with canary validation (Taibi, Lenarduzzi, & Pahl, 2020).

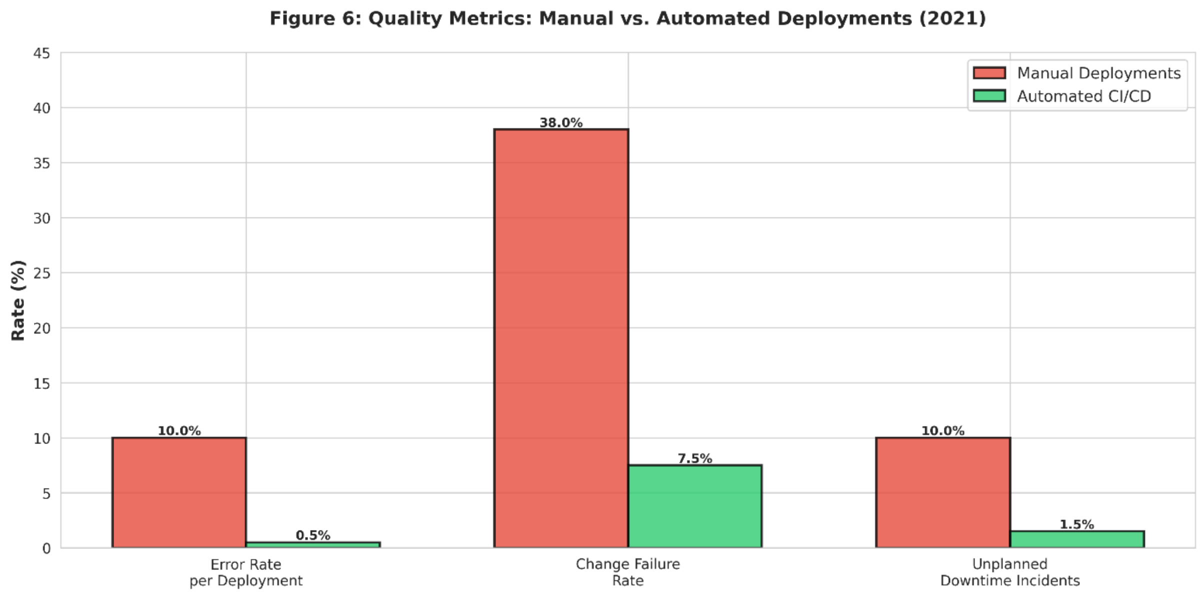

Figure 6: Quality Metrics: Manual vs. Automated Deployments (2021)

This grouped bar chart compares error rates, change failure rates, and unplanned downtime incidents between manual deployments and automated CI/CD, showing 90%+ improvements across metrics.

Strategic Considerations
Organizational Readiness
Successful implementations require DevOps cultural practices emphasizing shared responsibility and continuous improvement. Cross-functional teams combining development, operations, and quality expertise eliminate traditional silos. Comprehensive training enables operational personnel transition from manual execution to deployment system administration (Virmani, 2015).

Tool Selection
CI/CD platform evaluation criteria include:
- Deployment strategy support (blue-green, canary, rolling, progressive delivery)
- Cloud platform integration (AWS, Azure, GCP)
- Observability integration (Prometheus, ELK, Jaeger)
- Scalability matching organizational requirements
- Community support and documentation quality

Risk Management
Automatic rollback systems allow fast anger management. Blue-green deployments allow traffic diversion in a time frame of five minutes. Canary deployments remove exposure of users. Pre-deployment backups are those allowing restoring of databases in case of catastrophic situations. Disaster recovery testing should be done regularly on recovery time objectives (RTOs) and recovery point objectives (RPOs).

Emerging Trends
Progressive delivery integrates canary deployments, feature flags and A/B testing into entire systems that allow features to be rolled out based on data. Rollout based on the audience will allow the selection of a range of features to particular user groups. Performance-based rollout is a dynamically adjusted traffic which varies according to the version performance metrics. Machine learning complements deployment automation, forecasting the deployment risks and deployment optimization. ML models can be used to detect anomalies that indicate abnormal behavior of the system

before problems with production occur. Serverless computing architecture removes infrastructure and allows direct functions to be deployed. Canary patterns can be implemented using AWs Lambda aliases and traffic routing. GitOps paradigms use Git repositories as the sole source of truth and make it easier to control deployment process management by use of pull requests (Wiedemann, Forsberg, & Wiesche, 2019).

CONCLUSION

Blue-green and canary deployments done as part of extensive CI/CD pipeline structures are transformative potentials that support deployment velocities and stability in production at the same time. There is a great deal of quantitative evidence indicating the implementation with organizations registering 40-50% infrastructure savings, 87-90% reduction in downtime incidents and 208 times higher deployment frequencies. Organizations with high performance will have a deployment frequency of more than 1460/yr, recovery time of less than 1 hour and low change failure rates of less than 15 compared to single-digit/yr deployment frequencies, weekly recovery times and above 31 change failure rates.

Blue-green deployment architecture is used to create zero-downtime releases by using two production environments with traffic switching at instant as well as five-minute rollbacks. Canary deployment designs redistribute the risk of deployment by migrating traffic in batches to facilitate validation as it happens in real time without full production exposure. The implementation implies the need to invest in infrastructure, orchestrate containers, CI/CD automation tools, and monitoring systems, as well as changes in the organization setting up the culture of DevOps. Technological choices made strategically, strong change management and integration of security can make sure that compliance is maintained and deployment speed is also not compromised (Yang, Sailer, & Mohindra, 2020).

Emerging trends including progressive delivery frameworks, artificial intelligence-enhanced automation, and GitOps approaches promise continued evolution in deployment methodology sophistication. Organizations embracing these paradigms while maintaining DevOps focus will capture competitive advantages through enhanced delivery velocity, improved reliability, and operational efficiency optimization.

REFERENCES

[1] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. *IEEE Software, 33*(3), 42–52. https://doi.org/10.1109/MS.2016.64

[2] Fagerholm, F., Sanchez Guinea, A., Mäenpää, H., & Münch, J. (2017). The RIGHT model for continuous experimentation. *Journal of Systems and Software, 123*, 292–305. https://doi.org/10.1016/j.jss.2016.03.034

[3] Fritzsch, J., Boger, M., & Zimmermann, S. (2019). Microservices migration in industry: Intentions, strategies, and challenges. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 481–490). IEEE. https://doi.org/10.1109/ICSME.2019.00081

[4] Garg, S., & Garg, S. (2019). Automated cloud infrastructure, continuous integration and continuous delivery using Docker with robust container security. In *2019 Second International Conference on Advanced Computational and Communication Paradigm (ICACCP)* (pp. 1–6). IEEE. https://doi.org/10.1109/ICACCP.2019.8882922

[5] Izrailevsky, Y., & Bell, C. (2018). Cloud reliability. *IEEE Cloud Computing, 5*(3), 39–44. https://doi.org/10.1109/MCC.2018.032591615

[6] Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., … Lassenius, C. (2019). DevOps in practice: A multiple case study of five companies. *Information and Software Technology, 114*, 217–230. https://doi.org/10.1016/j.infsof.2019.06.010

[7] Railić, N. J., & Savić, M. (2021, March). Architecting continuous integration and continuous deployment for microservice architecture. In *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)* (pp. 1–5). IEEE. https://doi.org/10.1109/INFOTEH51037.2021.9400696

[8] Saha, S., & Das, S. (2020). 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)* (pp. 353–358). IEEE. https://doi.org/10.1109/I-SMAC49090.2020.9243605

[9] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access, 5*, 3909–3946. https://doi.org/10.1109/ACCESS.2017.2685629

[10] Shahin, M., Zahedi, M., Babar, M. A., & Zhu, L. (2019). An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering, 24*(5), 1061–1108. https://doi.org/10.1007/s10664-018-9651-4

[11] Soldani, J., & Brogi, A. (2019). Anomaly detection and failure root cause analysis in (micro)service-based cloud applications. *IEEE Transactions on Services Computing, 15*(1), 447–460. https://doi.org/10.1109/TSC.2019.2952189

[12] Taibi, D., Lenarduzzi, V., & Pahl, C. (2020). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Software, 37*(3), 22–32. https://doi.org/10.1109/MS.2019.2948694

[13] Virmani, M. (2015). Understanding DevOps: History, culture, and practices. In *2015 2nd International Conference on Advances in Computing and Communication Engineering* (pp. 12–16). IEEE. https://doi.org/10.1109/ICACCE.2015.16

[14] Wiedemann, A., Forsberg, K., & Wiesche, M. (2019). DevOps as an enabler for business agility. In *2019 IEEE 21st Conference on Business Informatics (CBI)* (Vol. 1, pp. 14–22). IEEE. https://doi.org/10.1109/CBI.2019.00013

[15] Yang, B., Sailer, A., & Mohindra, A. (2020). Survey and evaluation of blue-green deployment techniques in cloud-native environments. In *Service-Oriented Computing – ICSOC 2019 Workshops* (Lecture Notes in Computer Science, Vol. 12019, pp. 69–81). Springer. https://doi.org/10.1007/978-3-030-45989-5_6