Strategic Paradigms for Achieving Zero Service Interruption During Application Deployment

Santosh Kumar Nayak

IEEE, Senior Member, Member of IET

Abstract: ***In the past, developers released updates late at night- when users were not active- to minimize disruptions. Updating during slow times ensured users didn't have problems and weren't inconvenienced. Today, users want cloud services available 24/7 in all time zones, which makes finding convenient deployment times challenging. Fortunately, there are two common deployment techniques that can virtually eliminate downtime: blue-green deployments and canary deployments. In a blue-green deployment, you serve the current app on one half of your environment (Blue) and deploy your new application to the other (Green) without affecting the blue environment. In a canary deployment, you deploy to a small subset of systems or users first, before launching the full deployment for everyone to see. This article claims for zero downtime for nay application deployment for reference. 1) Essential for Modern Business: ZDD (Zero Downtime Deployment) prevents revenue loss, improves customer satisfaction, and enables faster, more frequent release cycles without needing to schedule maintenance windows. 2) Not a Single Solution: ZDD is not achieved by a single tool, but rather a combination of architectural choices, and deployment strategies. Popular methods include: a) Blue/Green Deployment: Running two identical production environments to switch traffic instantly. B) Canary Deployment: Gradually rolling out changes to a small subset of users to monitor for issues. C) Rolling Deployment: Updating instances incrementally to maintain continuous service. D) High Complexity and Cost: While beneficial, ZDD requires significant investment in automated testing, sophisticated monitoring, load balancers, and robust CI/CD pipelines. E) Database Challenges: The most challenging aspect of ZDD is managing database schema changes, which often require complex, phased, or backward-compatible migrations. F) Cultural and Technical Shift: True zero-downtime requires a, mature DevOps culture that prioritizes observability, resilience, and automated rollback capabilities.***

Keywords: Blue Green deployment**,** Digital deployment Strategy, canary deployment, zero downtime application, PVT testing, ZDD, Rolling deployment

1. Introduction

Historically, developers brought applications offline when deploying changes and updates, resulting in downtime. Now, **continuous integration and delivery (CI/CD)** allows teams to automate application build, test, and deployment processes. CI/CD pipelines keep environments available as much as possible, while speeding up the deployment process. But deploying an application or updating an environment can still cause downtime and other issues. Most businesses want to release applications and features without affecting the user. This is especially true for those with a modern application setup and a **CI/CD pipeline** in place.

As the name implies, CI/CD is a continuous process that flows seamlessly from developers writing code to making that code ready for users to interact with. This requires deploying changes into production environments, rapidly and safely, without interrupting users who may be performing critical tasks or relying on your systems for mission-critical processes.

Your application and deployment architecture plays a key role in reducing deployment downtime. Generally, your environment should meet the following requirements for both the canary and blue-green deployment methods:

1) A deployment pipeline that can build, perform automated tests, and deploy to specific environments
2) Multiple application nodes or containers distributed behind a load balancer
3) An application that is stateless, allowing any nodes in the cluster to serve requests at any time

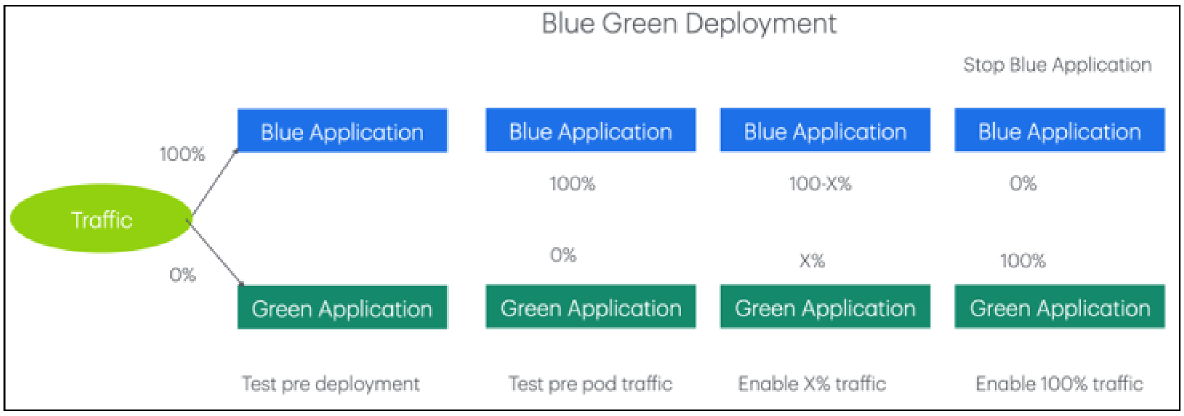
When you make changes to your application, they should be non-destructive to your data layer. This means you should make columns in your datasets optional or nullable. Don't rename or reuse columns for different purposes. Using a non-destructive method in your data layer lets you reverse changes or delete features if there are problems.

With these requirements in mind, let's explore the first zero-downtime deployment option: the blue-green deploy.

Blue Green Deployment Technical Solution to minimize customer Impact

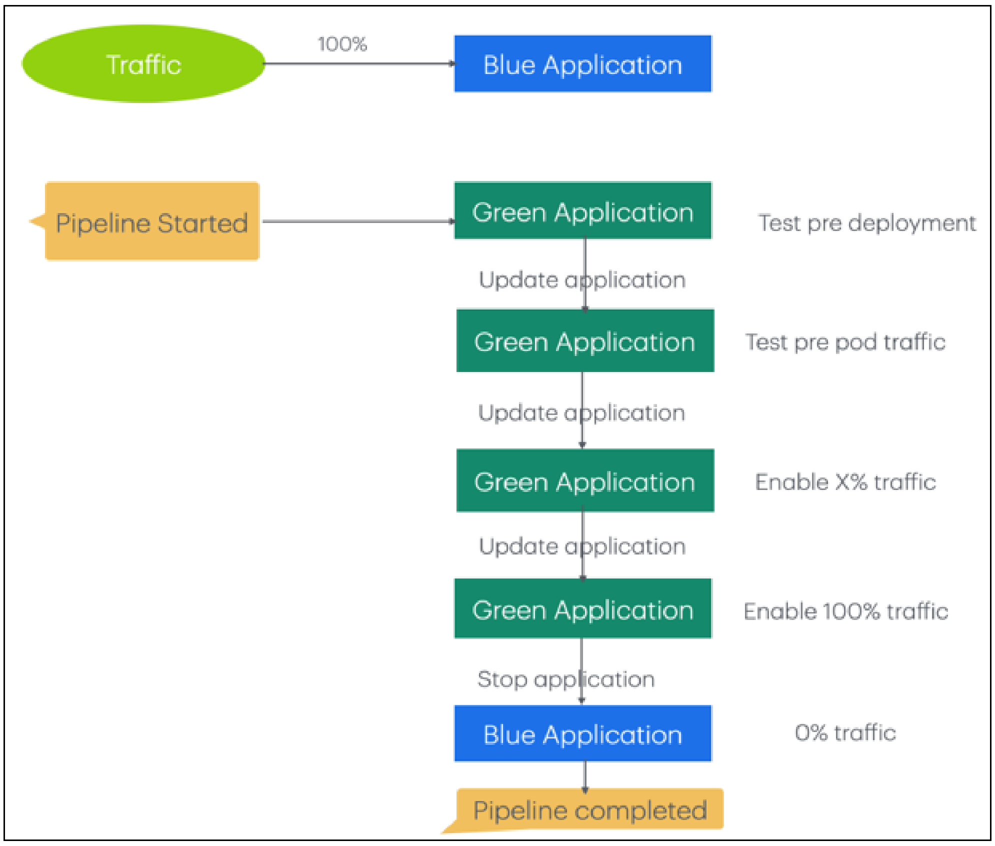
In this article, we will pit canary deployment vs blue-green, comparing and contrasting the differences between the two. We'll explore some scenarios to help you understand which solution is right for your environment.

A blue-green deployment is a software development strategy that uses two identical environments in same system to reduce downtime and deployment risk. One environment runs the current application version, while the other runs the new version.



How it works:
Imagine Blue application is taking 100% traffic and now deploy green application and gradually increase traffic until traffic reaches 100%.

Blue Green application is also suggesting that green application should not refresh new instance which has been tested so same instance need to be enabled traffic which was tested application. Blue Green application is mostly valuable for backward compatibility with upstream and downstream systems.



Process:
1) Initiate pipe line through Jules/Spinnaker
2) Deploy new enhancements green application without taking traffic.
3) Test dummy values and health check to make sure application is good to take traffic.
4) Move pipeline and update green application to take traffic for pre-prod. Mostly all standard software organization has there pre prod environment only for employee before enable real customer.
5) Once pre prod traffic customer tested then update green application to enable small portion of real customer traffic.
6) Confirm that traffic is taking properly without any hiccups.
7) Now update green application to take 100% real customer traffic.
8) Once all users are on the green environment, the blue environment can be keep as in stopped mode? Why because, incase of any emergency production issue comes then with minimal application outage we can rollback to the old application. Here we need to consider that some of new enhanced feature may not work as we are going back to previous application to stop customer impact.

Imagine a situation, where blue application is taking traffic and green application is in stopped mode and application started having some issue (production issue) in that case:

We could have application outage, so here major question comes how long application can be in outage status. We could have 2 approaches to handle this situation:

1st Approach:
1) Issue need to re produce in lower environment.
2) Fix the issue
3) Test in lower environment
4) Raise emergency request to deploy code in production
5) Application could face any infrastructure related issues while deploying Blue green application.
6) Then finally serve the traffic to larger audience.

Disadvantages:
Till time all above 6 points are completed application will be in outage state and it could be more based on different infrastructure outage/glitch.

Advantage:
New enhanced feature could be retained by this approach and user will have same feature as before after deployment.

2nd Approach:
As we keep green application in stopped mode then with help of incident ticket we could enable green application and stop blue (currently taking traffic) application.

Ultimately all the traffic will take by old green application.

Disadvantages:
1) Some of the new enhanced feature will not available for customers until further deployment.
2) If upstream/downstream system has any contact change had happened, old application will not work so we have to follow path forward approach.

Advantages:
1) Customer outage is very minimal
2) Which also can be supported to the approach 1 and extra feature to go back to previous application.

In my view, contract change between upstream or downstream applications are not very often and if at all there are contact change then also without application outage we cannot deploy latest application. However, there could be chance only we deployed latest change with backward compatibility then upstream/downstream make changes in there e application, in that case, we have to take longer outage to move forward and stopped old green application will not work.

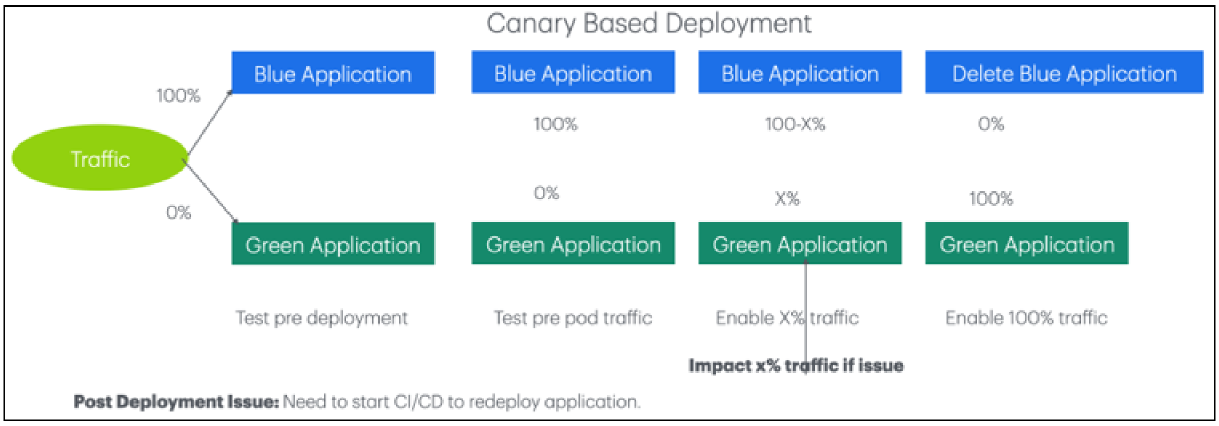
Secondly application owner need to think of the priority, as if we keep old application which is also supporting to go back option.

Benefits Reduces deployment risk, Increases application availability, and Simplifies the rollback process if a deployment fails.

Drawbacks
- Requires maintaining two identical environments, which can be expensive
- Requires more coordination than other deployment strategies
- Can be slower to switch over than other deployment strategies
- Not suitable for all applications

Similar deployment strategies:
- Canary deployment: Similar to blue-green deployment, but offers a more nuanced approach
- Red-black deployment: Similar to blue-green deployment, but handles traffic differently

Canary Based Deployment:
In software deployment, a "canary deployment" involves gradually rolling out a new version of application to a small sunset of users (the “canaries”) before deploying it to the entire user base, allowing for early detection and mitigation of potential issues.



Here's a more detailed explanation:

What is Canary Deployment?
1) **Gradual Rollout:** Instead of deploying a new version to all users at once (like in blue-green deployment), canary deployment introduces the update to a small group of users initially.
2) **Risk Mitigation:** This allows developers to monitor the performance and stability of the new version in a real-world environment before exposing it to the entire user base.

3) **Early Issue Detection:** If problems arise, they can be identified and addressed quickly, minimizing the impact on the broader user base.
4) **Rollback Capability:** If issues are detected, the deployment can be rolled back to the previous version without affecting the majority of users.
5) **Phased Approach:** The new version is gradually rolled out to more users as confidence in its stability increases.

How it Works:
1) **Initial Deployment:** A new version of the application is deployed to a small subset of servers or nodes (the "canary" servers).
2) **Traffic Routing:** Traffic is directed to these canary servers, allowing a small group of users to interact with the new version.
3) **Monitoring and Testing:** Developers monitor the performance and stability of the new version, looking for any issues or unexpected behavior.
4) **Gradual Rollout:** If the new version performs well, traffic is gradually shifted to the new version, increasing the number of users accessing it.
5) **Full Rollout:** Once the new version is deemed stable, traffic is fully redirected to it, and the old version is decommissioned.
6) **Rollback:** If issues arise, the traffic can be redirected back to the old version, and the deployment can be rolled back.

Benefits of Canary Deployment:
1) **Reduced Risk:** Minimizes the impact of potential issues by identifying and addressing them early.
2) **Faster Time to Market:** Allows for quicker release cycles by testing new features in a controlled environment.
3) **Improved User Experience:** Ensures a smoother transition to new versions by identifying and addressing potential issues before they impact a large number of users.
4) **Better Feedback:** Provides an opportunity to gather real-world feedback from users before rolling out the new version to the entire user base.

What is canary deployment?
Instead of maintaining two different blue and green environments, canary deployments select a small number of servers or nodes for the initial deployment. These provide a testing ground for new code before the entire production environment receives a new deployment.

In Canary deployments if something go wrong with a, it will only affect a small subset of users or systems (the canaries) rather than cause problems for the entire production environment (the miners.) You can configure your environment for canary deployments in a variety of ways. The first method involves using a load balancer:
1) Your production environment is set up behind a **load balancer,** with a few extra nodes or servers being reserved as backups.
2) The extra node or server group is the deployment target for the new version of your app. When an update is ready for release, you deploy it to your unused nodes. These "canary" servers are the first to run the new version in the live environment.
3) Using the load balancer, you gradually direct a small percentage of traffic to the canary servers. This allows you to test the update on a small number of users first, monitoring for errors or user feedback to help identify and address any issues.
4) If the new version performs well and no significant issues are detected, you gradually increase the percentage of traffic it receives. This incremental process continues until the new version is handling all traffic.

Disadvantages:
1) If any issues are present in new version of application then that is directly impact to customers which could have handled by using blue green with health check and pre-prod traffic testing so that we will not impact directly to real customers.
2) After 100% traffic enabled, post deployment, if any prod issue comes then we should not have any option to rollback to old application because after canary make 100% traffic, removed old application. This issue can be resolved by using Blue Green deployment by keeping old application to return back to old application with minimum application downtime.

We could resolve this canary issue by deploying old application (Need to keep track of old application) or new application by fixing issues. Here it could take more time to reproduce issue in lower environment, fix issue and then test. After that need to create emergency change request and could have come infrastructure issues while deploying. Keeping it in mind till that time application is having outage.

The other option for configuring a canary deployment is to use a development pattern called feature flags. These tools let you control changes in an application with a configuration switch. They enable you to easily turn features on or off without changing the code. This can be useful for testing new features, rolling out changes gradually, or fixing bugs quickly.

Here's how you execute a canary deployment using feature flags:
1) When developing new features or making significant changes to existing ones, wrap the changes in feature flags. This practice ensures that the new functionality can be enabled or disabled through configuration changes rather than code deployments.
2) Deploy your application with the new features or changes disabled by default. This means the new code is present in your production environment but inactive.
3) Identify a small group of users to act as your canary group. This could be a percentage of your total user base, specific user IDs, or users who meet certain criteria. Enable the feature flags only for this canary group.
4) Closely monitor application performance, user feedback, and any other relevant metrics for issues related to your changes. If the canary phase is successful and the new features are stable, begin gradually increasing the percentage of users with the feature flag enabled. Continue the incremental rollout, monitoring for issues as the number of users increases.

5) Once you’re confident in the stability and performance of the new features, enable the feature flags for all users, completing the rollout. If significant issues are encountered during the canary or rollout phases, use the feature flags to disable the new features.

Canary deployments with feature flags can be challenging to set up, but there are tools that simplify the process, including Flagger and Argo Rollouts. You can incorporate these tools into your CI/CD pipeline, giving you end-to-end visibility into your continuous delivery process and the ability to swiftly roll back changes when issues arise.

2. Conclusion

Zero-downtime deployment (ZDD) has transitioned from a luxury to a necessity in the modern digital landscape, serving as a critical component for maintaining high availability, user trust, and revenue continuity. It is defined as a technique that allows for the update or upgrade of an application without interrupting service for users.

Based on the provided search results, here is the conclusion regarding zero-downtime deployment. In summary, zero-downtime deployment is the "gold standard" for delivering software in the cloud-native era, transforming updates into invisible, low-risk, and routine events. While achieving 100% perfection is difficult, the goal is to eliminate user-facing disruption and drastically reduce the Mean Time To Recovery (MTTR).

References

[1] https: //docs. aws. amazon. com/whitepapers/latest/blue-green-deployments/welcome. html