BLUE-GREEN DEPLOYMENT: AN INTRODUCTION

In the current environment, it is increasingly necessary to use continuous integration and continuous deployment (CI/CD) to stay up to date with software testing, stability, and improvements. There is nearly constant pressure for frequent and functional software updates, and amidst these regular updates, it's necessary for teams to ensure high quality and to provide a good customer experience. These pressures have led to new methods and approaches for software development and deployment, including blue-green deployment.

Blue-green deployment is a deployment method that helps to reduce downtime and risks. It's a software release model that transfers traffic from a previous version to a new version by utilizing two nearly identical production environments, called blue and green. At any given time, only one of the environments is live, and that environment handles all production traffic. This method can be a good solution for businesses looking to avoid downtime while reducing risks during updates.

Specifics of How Blue-Green Deployment Works

Blue-green deployment requires two environments that need to be different but also as close to identical as possible. This can take place with two different pieces of hardware, different virtual machines that run the same hardware, or a single operating environment that is divided into separate zones with separate IP addresses. Additionally, this deployment method requires a router or load balancer that routes traffic to the correct environment.

It's helpful to walk through the details of how this type of deployment would work to better understand this method. Let's start with an example where you're using version 1 of an app. The environment running version 1, we'll say the green version, would be live. The router or load balancer would direct all users to that version. You can then develop and deploy version 2, the blue version, to the same environment while still directing all users to the green version. At this point, you're able to run any smoke tests, ensuring that all aspects of version 2 are working correctly without risks of disruption. Once version 2 is ready for full deployment, you route users to the blue version and disconnect traffic to the green version. At this point, the blue version is live and the green version is in standby.

If any bugs or performance issues arise with version 2, you can always reroute traffic to version 1 without any substantial interruptions or concerns. What's more, once you're comfortable that version 2 is working correctly and is stable, you can update the green version to version 2. This allows for disaster recovery and allows you to begin developing additional updates as needed.

This structure allows for ongoing development, testing, and deployment and gives developers a non-live environment for testing and deployment of the next release. The result is two environments that are consistently transitioning between being live, being the previous version, and offering staging for the next version.

Pros and Cons of Using Blue-Green Deployment

The key benefits that this deployment system offers are the elimination of downtime and a reduction of risks. Because the switch from blue to green happens almost instantaneously, there is no downtime for users. In fact, users won't even notice when the switch happens.

Additionally, because it allows staging, rollback, and disaster recovery, it reduces the risks that are associated with updates. This method allows for tests to take place in production and offers a full environment for testing. This is ideal for developers, making them confident that all aspects of updated software are working appropriately prior to launch. Plus, it's an effective way to avoid surprise errors, which are frustrating for users and development teams. Further, it's reassuring for teams to know that in a worst-case scenario, they can always reroute users back to the original version. Finally, once the new version is stable, the new release can go to standby and can offer recovery in the event of a disaster.

Another important benefit is the ability to deploy at any time. Because this process eliminates downtime, it lets teams deploy whenever they want. Historically, deployments have taken place in the middle of the night or on weekends. Not only does this have the potential of disrupting users, but it's also disruptive for development teams. Being able to deploy updates anytime allows for having the entire team present and fully engage (as opposed to deploying in the middle of the night, with a reduced and tired team). This helps to reduce errors and to ensure a smooth deployment.

While blue-green deployment offers some clear benefits, it also comes with some costs. Because the transition from green to blue happens instantaneously, there is the potential for the transactions or sessions taking place at the time of the switch to be lost. There are ways to potentially avoid this - for example, some designs allow for sending those transactions to both systems and others allow for putting a new version in read-only mode before cutover and then switching to read-write mode. Regardless, this can be a problem area and is something for teams to acknowledge and address before using blue-green deployment.

Another issue with this method is that not all environments have the necessary resources to effectively utilize this method - for example, it requires N-1 data compatibility. Not all environments have the correct uptime requirements or resources for this process. Additionally, there are costs to running two versions at the same time. It can significantly impact the data model and other areas of the system.

Given the pros and cons, it can be hard to determine if this system is right for your organization. Generally, it comes down to what issues are most pressing for your software. For example, if you're concerned about downtime and reducing the risks of software updates, this deployment method is likely a good choice for your team. If these concerns are less pressing, a classic swap and drop deployment might well be a better fit.

Alternative Deployment Systems

Additionally, it's helpful to keep in mind that blue-green deployment is not the only process that offers the benefits of reducing downtime and risks. Another alternative is a canary release, named for the mining practice of sending a canary into a mine to test the air quality. If the canary returned, miners knew that the air was safe. If it did not return, they knew the air was toxic and should be avoided. Utilizing a similar strategy, this method of deployment tests the update on a limited subset of users before making it available to all users. By releasing new software to a limited number of servers and then monitoring it for bugs or issues, it substantially reduces deployment risks.

Another alternative is rolling deployment. This process spreads out deployment across a few phases by using several servers and executing different functions in server clusters. It avoids the issue of taking the app offline. Instead, the load balancer simply stops delivering traffic to a specific server while it's being updated. When it's updated and stable, another server is taken offline and updated. This process continues until the entire system is updated and running smoothly. The benefits of this method are that there's always a stable version available and errors only impact a small subset of users.

With increasing pressures to consistently update software while also maintaining quality and maximizing customers' experiences, it's important for development teams to utilize effective and efficient deployment strategies. Blue-green deployment offers a way to eliminate downtime and reduce risks by utilizing two environments that are consistently transitioning from being live, serving as a fallback, and serving as a space for development, testing, and staging.