Deployment Strategies: 6 Explained in Depth

For every software development firm, there comes a time when their software will need to be changed or upgraded. It can come in the form of a total version change or fixing a bug found in the original version.

Now that the new version is ready, here comes the real task the firm's DevOps teams must undertake: How do we deploy this new upgrade seamlessly without complicating issues, bearing in mind that the user experience of the end-users is a top priority?

To solve this humongous task, the DevOps team integrates **deployment strategies** into their operating procedures. Over the years, different techniques have been developed to help software firms successfully deploy new application and code versions. These techniques are called **deployment strategies**.

In this article, we'll look at what deployment strategies are and the various types of deployment strategies, and their method of usage. We'll also cover each technique's advantages and disadvantages and when to use which deployment strategy in your work as a DevOps engineer.

What Is a Deployment Strategy?

A deployment strategy is any technique employed by DevOps teams to successfully launch a new version of the software solution they provide. These techniques cover how network traffic in a production environment is transitioned from the old version to the new version. Based on the firm's specialty, a deployment strategy can influence downtime and the company's operational cost.

Various Types of Deployment Strategies

Now that we understand what deployment strategies are, let's explore the various types of deployment strategies.

Blue/Green Deployment

In this type of deployment strategy, the new version of the software runs alongside the old version. Note that you can also refer to this as **red/black deployment strategy** in some cases.

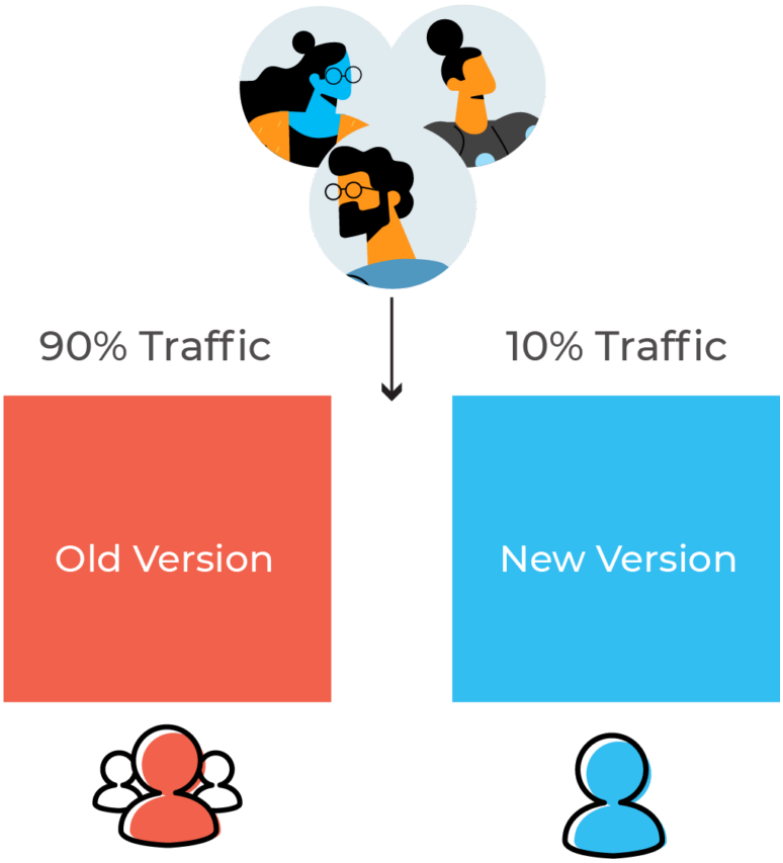
Here, the stable or the older version of the application is always **blue or red**, while the newer version is **green or black**.

After the new version has been tested and certified to meet all the requirements, the load balancer automatically switches the traffic from the older version to the newer version.

The major advantage of this strategy is that it avails a quick update or rollout of a new application version. However, its main disadvantage is that it is costly because you must run both the new and old versions concurrently. Engineers mostly use this method in mobile app development and deployment.

Canary Deployment

In canary deployment, the deployment team sets up the new version and then gradually shifts the production traffic from the older version to the newer version. For example, at a point in time during the deployment process, the older version might retain 90% of all traffic for the software while the newer version hosts 10% of the traffic. This deployment technique helps the DevOps engineers test the stability of the new version. It uses live traffic from a subset of the end-users at different levels that varies as production occurs.

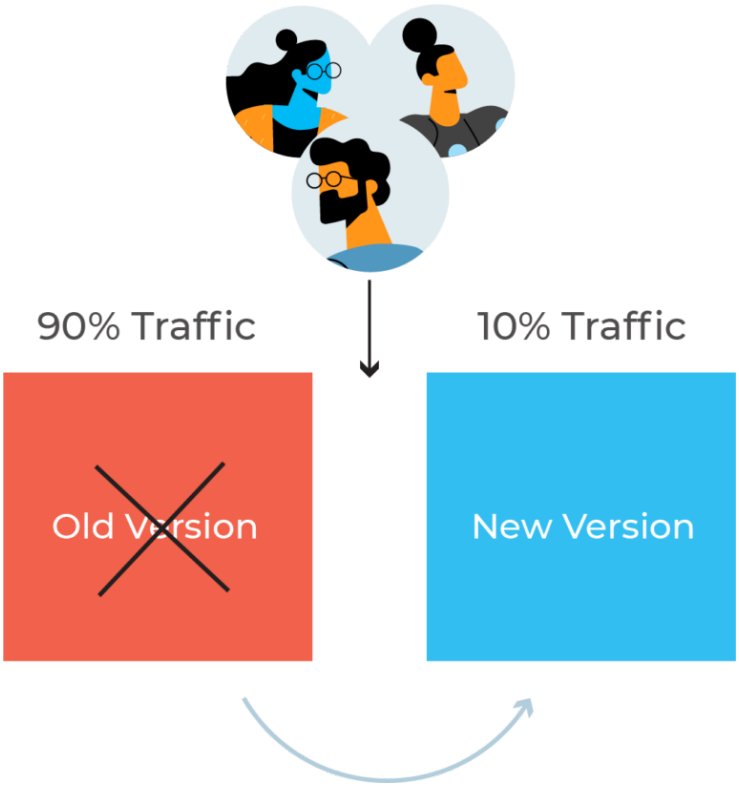


Canary deployment enables better performance monitoring. It also aids in a faster and better rollback of the software if the new version fails. However, it has a slow nature and a more time-consuming deployment cycle.

Recreate Deployment

In this deployment strategy, the dev team shuts down the old version of the application entirely, deploys the new version, and then reboots the whole system. This deployment technique creates a system downtime between shutting down the old software and booting the new one.

It is cheaper and mainly used when the software firm wants to change the application from scratch. It doesn't require a load balancer as there's no shifting of traffic from one version to another in the live production environment.

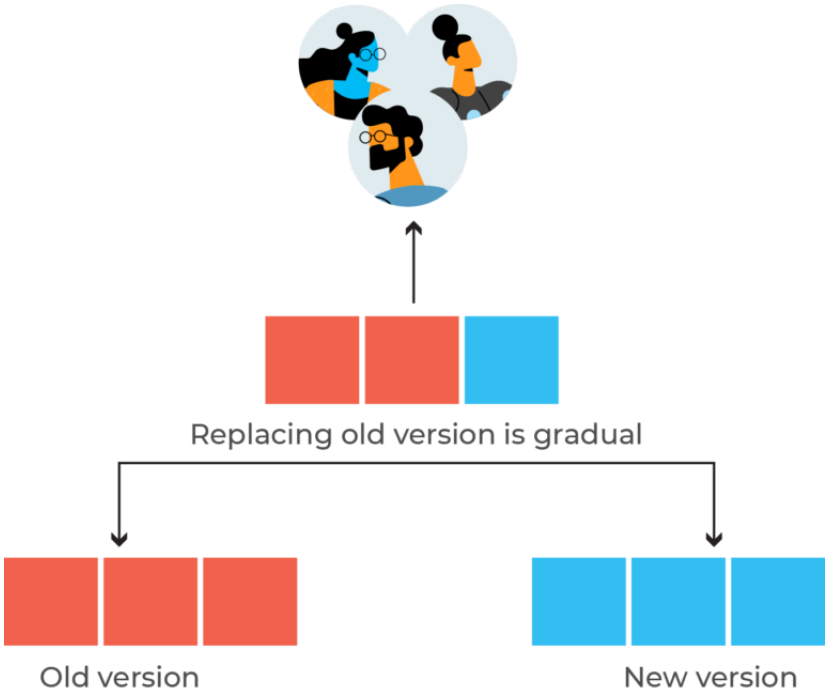


But this strategy dramatically affects the end users due to the downtime. Users must wait until the application goes live again to use the software. As a result, not many developers use this strategy unless they have no other choice.

Ramped Deployment

The ramped deployment strategy gradually changes the older version to the new version. Unlike canary deployment, the ramped deployment strategy makes its switch by replacing instances of the old application version with the instances from the new application version one instance at a time. You can also call this method the **rolling upgrade deployment strategy**.

When developers replace all instances of the older version, they shut down the older version. The new version then controls the whole production traffic.



This strategy gives zero downtime and also enables performance monitoring. Nevertheless, the rollback duration is long in case there is an unexpected event. This is because the downgrading process to the initial version follows the same cycle, one instance at a time.

Shadow Deployment

In this deployment strategy, developers deploy the new version alongside the old version. However, users can't access the new version immediately. Just as the name suggests, the latest version hides in the shadows. Developers send a fork or copy of the requests the old version receives to the shadow version to test how the new version will handle the requests when live.

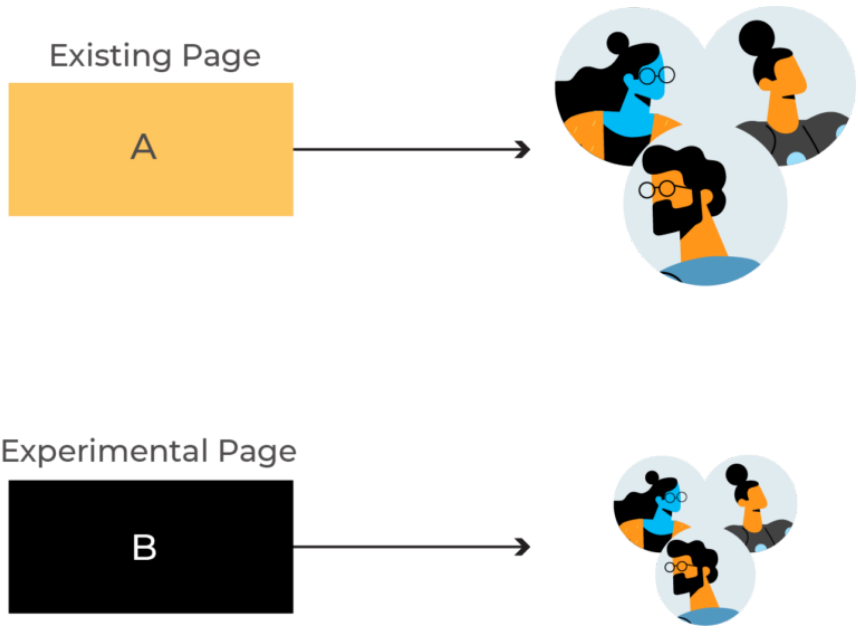
This technique is very complex, and as such, the DevOps engineer should be careful so that the forked traffic does not create a duplicate live request since two versions of the same

system are running.

Shadow deployment lets engineers monitor system performance and conduct stability tests. But it's expensive and complex to set up and can create serious issues.

A/B Testing Deployment

In A/B testing deployment, developers deploy the new version alongside the older version. However, the new version is only available to a subset of users. These users are selected based on specific conditions and parameters the engineers choose. These parameters can be the user's location, type of device, UI language, and operating system.



This method measures the effectiveness of the application's new functionality. After gathering statistics from performance monitoring, developers roll out the version that yielded the best results to all users.

Real-time statistical data can help developers make important and favored decisions about their software. However, A/B testing is very complex to set up. It also requires a highly intelligent (and pricey) load balancer.

Better Deployment with Tools

Successfully launching new features with any deployment strategy isn't an easy task. DevOps teams need to understand what values they'll be shipping to their users. Luckily, Plutora offers tools can make it easier for DevOps teams to launch or release new features with any deployment strategies.

For example, release management tools allow firms to plan and manage their release, adopting a deployment strategy and tracking analytics. Another tool that's great to have when planning a deployment is the deployment planning tool. This tool allows a streamlined deployment process to minimize risk.

Software deployment tools save time as they can manage CI/CD (continuous integration/continuous delivery), automating deployment. These tools can also enhance security as they can integrate role-based access control.

Conclusion

You can deploy a new version of your software using any of these strategies. Each of these strategies has its merits and demerits, and each is good in different scenarios. The only question now is, what makes the most sense for your DevOps team to use? Keep in mind your team's, project's, and company's needs and business goals. Also, note how much downtime your company can afford and other cost constraints. Check out Plutora and make use of our deployment management tools and release management platform to aid all your software deployment activities. Sign up for a personalized demo today!

For every software development firm, there comes a time when their software will need to be changed or upgraded. It can come in the form of a total version change or fixing a bug found in the original version.
