Deployment Patterns for Confidence

Joseph W. Yoder, The Refactory, Inc. – USA
Ademar Aguiar, Faculdade de Engenharia, Universidade do Porto – Portugal
Paulo Merson, Federal Court of Accounts (TCU), Brasilia, Brazil
Hironori Washizaki, Waseda University – Japan

Many software development processes such as Agile and Lean focus on the delivery of working software that meets the needs of the end users. Many of these development processes help teams respond to unpredictability through incremental, iterative work cadences and through empirical feedback. There is a commitment to quickly deliver reliable working software that has the highest value to those using or benefiting from the software. DevOps has become a common practice to assist with quality delivery in these practices, specifically when developing using the microservices architectural style. Delivery options have evolved from the "big bang" approach to those that release new features, or small pieces of them, more safely and reliably, i,.e. with more confidence, through techniques such as "Blue-Green" and "Canary" deployments. This paper will focus on these two techniques presenting patterns for each.

Categories and Subject Descriptors

• Software and its engineering ~ Agile software development • Social and professional topics
• Software and its engineering ~

General Terms

Agile, Sustainable Delivery, Patterns, Deployment, Confidence,

Additional Keywords and Phrases

Agile Software Development, Continuous Integration, Continuous Delivery, Innovation, Reliability, Delivery, Deployment

ACM Reference Format:

Yoder, J.W., Aguiar A., Washizaki, H., 2019. "Deployment Patterns for Confidences". HILLSIDE Proc. of 8th Asian Conf. on Pattern Lang. of Prog. (March), 8 pages.

Author's email address: joe@refactory.com, ademar.aguiar@fe.up.pt, paulomerson@gmail.com, washizaki@waseda.jp

Introduction

Nowadays, releases need to happen more frequently, sometimes many times per day, in order to meet growing business objectives. This helps organizations to be more agile by making changes that deliver business value faster which includes ways to rapidly experiment with new ideas, testing the impact on the business, and quickly addressing any issues that arise.

DevOps as a software engineering practice unifies software development (Dev) and software operation (Ops). To assist with quality delivery in these practices you need to provide a "*Quality Delivery Pipeline*" [ref SLPLoP paper] to help assure the delivery meets the requirements and proper validation and checks are done before releasing into full production. At the end of the pipeline the validated system will be deployed into production. There are various deployment techniques to help successfully and reliably deploy more quickly. The goal is to give confidence by providing "reliable, working software" to the user (making the user confident in the system). Also, the teams will have more confidence the system is working.

Monolith architectures generally use a "big bang" deployment approach that updates most of the application at one time, sometimes including database updates as well. This has been the de facto release approach for decades. Big bang deployment approaches required a development and operations team to do extensive development and testing before release. This big bang approach can be slow, error-prone, and less agile.

Feature toggles[1] have become popular with "big bang" deployment to help address some of these problems, especially with reliability. With feature toggles you can enable or disable features at runtime. This is especially useful for releases where new features might cause issues, thus you can turn off a specific feature and address the eventual issues more quickly. However, there are still maintenance issues with using feature toggles and quite often the toggles become a legacy to the system.

The challenges in any types of deployments is that if you break something, the deployment could negatively affect reliability or customer experience. Another scenario would be that you are testing new features with end users and unstable new features would negatively affect regular users while power users would want to have them (e.g. stable versus beta versions), so you have parallel development streams. Therefore having alternatives deployment techniques for releasing can provide many benefits. Recently, deployment techniques in modern software development that have become more popular are *blue-green deployment* and *canary deployment*.

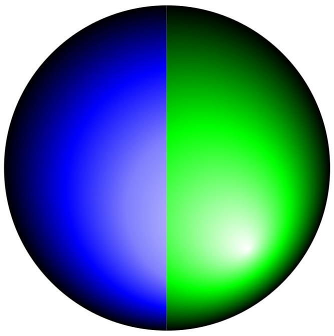
Blue-green deployment is where you have two "almost" identical environments. One is always live. You release to the non-live environment and validate the release with testing. After verification, you switch all traffic to the newly released environment and the previous environment is idle and available for rollback or a new release.*Canary deployment* deploys a new application code in a limited part of the production area with limited users. You can then have these users test and validate the application and if there are no problems, the system can slowly be rolled out to the rest of your users.

This paper will start out by looking at the "*Blue-Green Deployment*" and the "*Canary* and *Rolling Deployment"* patterns and how they assist development teams in more reliably releasing more quickly. The context and the forces for these patterns have similarities as they are both addressing deployment issues. However, there is a variance in the problems they are addressing. Therefore the patterns duplicate these similar context and forces, though some forces have been added as they relate to the different patterns.

[1] https://en.wikipedia.org/wiki/Feature_toggle

Blue-Green Deployment

aka Red-Black or A/B Deployment



You are doing continuous delivery as part of DevOps by making deployment as automated and quick as possible. You are releasing to a live environment that has potentially many users.

How can we deploy reliably and with confidence of not negatively impacting many users?

❖ ❖ ❖

Building and releasing into production environments can be tedious and error prone.

Automating the testing and deployment process is challenging, due to its complexity and uncertainty, thus requiring a lot of expertise and effort.

Lack of validation or testing of your release can be dangerous and costly.

It can be expensive to duplicate the entire production environments. Any other alternatives will be only a simpler replica, possibly not emulating all the issues of production.

New releases have various risks that can negatively impact the business (e.g. lost of funds), if problems arise. Problems with releases need to be able to be rolled back quickly and reliably.

❖ ❖ ❖

Therefore, when releasing, have two environments that are nearly identical. One is the live environment. Release into the non-live environment, after validating the release, switch all network traffic to the new environment disabling the previous live environment.

This type of deployment process is referred to as "*blue-green deployment*". *Blue-green deployments* has two nearly identical production environments (called "blue" and "green") where deployments are made The two environments need to be as identical as possible. They can be, for example, different pieces of hardware, or different virtual machines running on the same (or different) hardware, to mention a few.

At any time only one of the blue-green environments is live. For example (see Figure 1), let's say the green environment is currently being used for live production. When you have a new release, you deploy your system and do your final testing in the other (blue) environment. Once the software is working in the new environment (blue), you switch all live access so that all incoming requests now go to new tested (blue) environment. The previous production environment (green) is now idle and ready either for a rollback, for emergency use, or for the next release. As you have new releases, you continue to switch back and forth between the blue and the green environment.

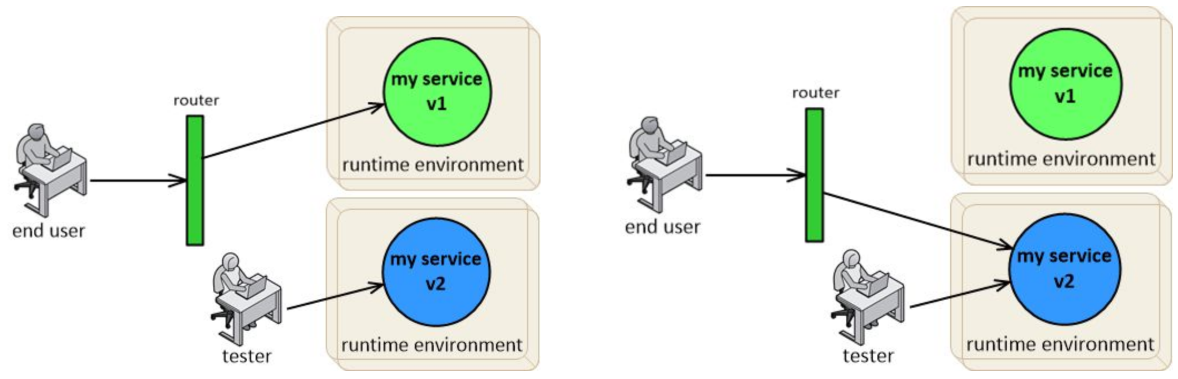


Figure 1: Blue-Green Deployment

Rollback can still be complicated as you have to make sure that there are no missed transactions during the transition. Your design might handle this through some replication or possible putting the systems into a read only stage during the transition.

Advantages of Blue-Green Deployments

Blue-green deployment gives the advantages of a staging environment: when the green environment is active, blue becomes the existing staging environment for the next deployment, and vice-versa.

Blue-green deployment gives the advantage of rollback: after deploying to one environment, if a problem is discovered you can easily roll back and start using the previous environment. You may have to cleanup some transactions that happened during the rollout of the failed environment.

Finally, *blue-green deployments* offers some advantages for disaster recovery: you always have a backup environment ready in case of any disasters.

Disadvantages of Blue-Green Deployments

Blue-green deployments require organizations to have two identical sets of deployment environments, which can lead to significant added costs and overhead without actually adding capacity or improving utilization. As an alternative, there are other strategies that can help such as *canary* or *rolling deployments*. *Canary deployment* releases the new system to a small limited number of users, while *rolling deployment* staggers the rollout of new code across servers, usually to a servers with limited number of users first.

* * *

Blue-green Deployment can use *Feature Toggles* to emulate a form of *Canary Deployment*, by toggling on/off certain features for certain users roles.

Blue-green Deployment can be used in conjunction with *Canary Deployment*. In other words you can push the release completely to the new server. And then move a few users from the old server to the new server in a canary fashion. *Feature toggles* can be a way to enable the features selectively for users.

Canary Deployment
aka Staged Deployment

You are doing continuous delivery as part of DevOps by making deployment to different environments as automated and as quick as possible. You are releasing to a live environment that has potentially many users.

How can we get feedback on the new release, verify if it is working properly, and get early reactions from users?

❖ ❖ ❖

Building and releasing into production environments can be tedious and error prone.

Automating the testing and deployment process is challenging and requires a lot of expertise.

Lack of validation or testing of your release can be dangerous and costly.

New releases have various risks that are important to validate before released to all users..

Getting feedback and acceptance on new features but can be risky if certain users are not happy or have problems with the new features.

❖ ❖ ❖

Therefore, first deploy the change to a limited number of users or servers to test and validate the release. This could include verifying the release works properly and/or getting acceptance feedback from your users. After you have validated the release, then roll the change out to all servers or users.

This limited release is called *Canary Deployment*. Canaries used to be used in coal mining as a warning system making sure there were no toxic gases before miners entered the mine. In a sense we are doing the same thing with *canary deployment*. Before we release to a wide audience, we first deploy to one or more canary servers (see figure 2). These might be for trusted internal users. Then after we validate the release, we can release to other users and servers.

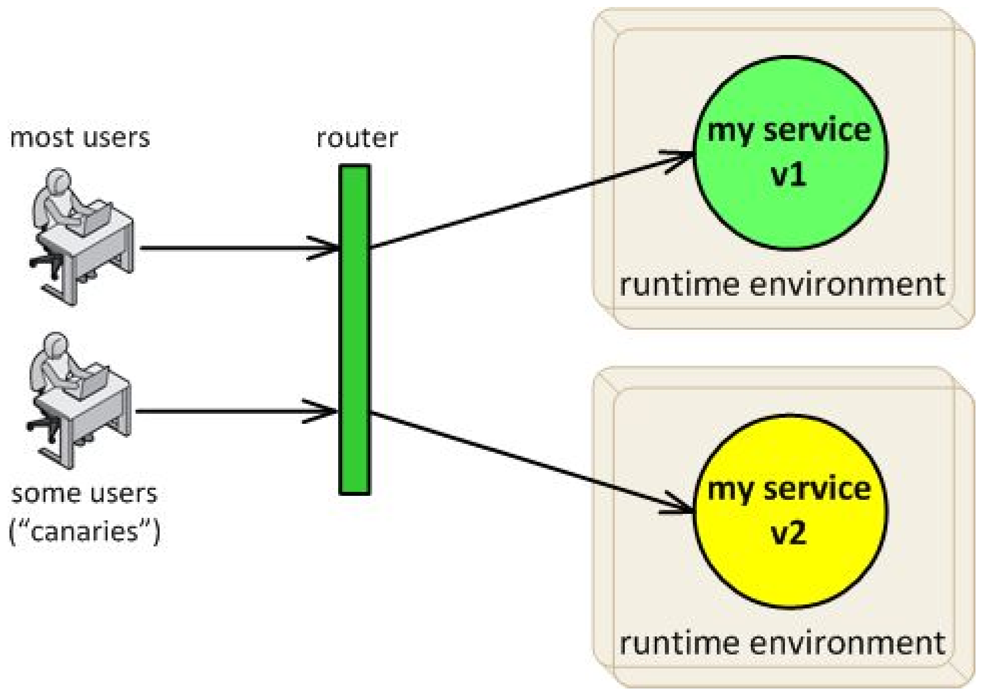


Figure 2: Canary Deployment

Advantages of Canary Deployments

Canary deployment allows for the exploration of the new system within the environment finding problems before being pushed out to all of your users.

Disadvantages of Canary Deployments

Canary deployments require you to control who sees the canary release and, for example, maybe to build the infrastructure for moving users from the original system to the canary system, depending on the authentication mechanism used.

Alternatives to Canary Deployments

An alternative to canary deployment is to deploy the new version to everyone with *feature toggles* turned for the new features turned off. Then you selectively enable features to different users. As you validate the release, you increase the number of users access to the new features. This requires an implementation using "*feature toggles*".

* * *

Big-bang Deployment can use *Feature Toggles* to emulate a *Canary Deployment* by selectively accessing the new features for designated users..

You can also use *Blue-Green Deployment* to push the release out to the green server for example, and then only move a few users from the blue server to the green server for the *Canary Deployment*. Then as you validate the release you can move more and more users to the green server.

Summary

There is not a one-size fits all approach that works for all instances. There are tradeoffs and you can use variations or a mix of the deployment strategies. For example, you could use *feature toggles* with *canary* and/or *blue-green deployment* strategies. Big-bang can use feature toggles to get a canary effect. You can also use feature toggles with blue-green for a canary effect.

Acknowledgements

We'd like to thank our shepherd Michael Weiss for his valuable comments and feedback during the AsianLoaf PLoP 2019 shepherding process. Finally we'd like to thank our 2019 AsianPLoP Writers Workshop Group, XXX, YYY, and ZZZ for their valuable comments and suggestions.

References ***NOTE TO Writers' Workshop...TO BE UPDATED***

[YWW] Yoder J., Aguilar A., and Washizaki H., "Quality Delivery Pipeline," 12th Latin American Conference on Patterns of Programming Language (SugarLoafPLoP 2018), Valparaíso, Chile, 2018.