Intro To Deployment Strategies

The original article link - https://harness.io/blog/blue-green-canary-deployment-strategies/

Whether we mean to or not, software deployments look different across organizations, teams, and applications. This can make pushing the deployment button feel like playing a game of craps: you roll the dice and try to stay alive. Luckily, there are a few ways to limit the variance in success. This blog post will discuss the different strategies and practices that can help you succeed with your production deployments.

Deployment Strategies to Consider

Deployment strategies are practices used to change or upgrade a running instance of an application. The following sections will explain six deployment strategies. Let’s start with discussing the basic deployment.

The Basic Deployment

In a basic deployment, all nodes within a target environment are updated at the same time with a new service or artifact version. Because of this, basic deployments are not outage-proof and they slow down rollback processes or strategies. Of all the deployment strategies shared, it is the riskiest.

Pros:

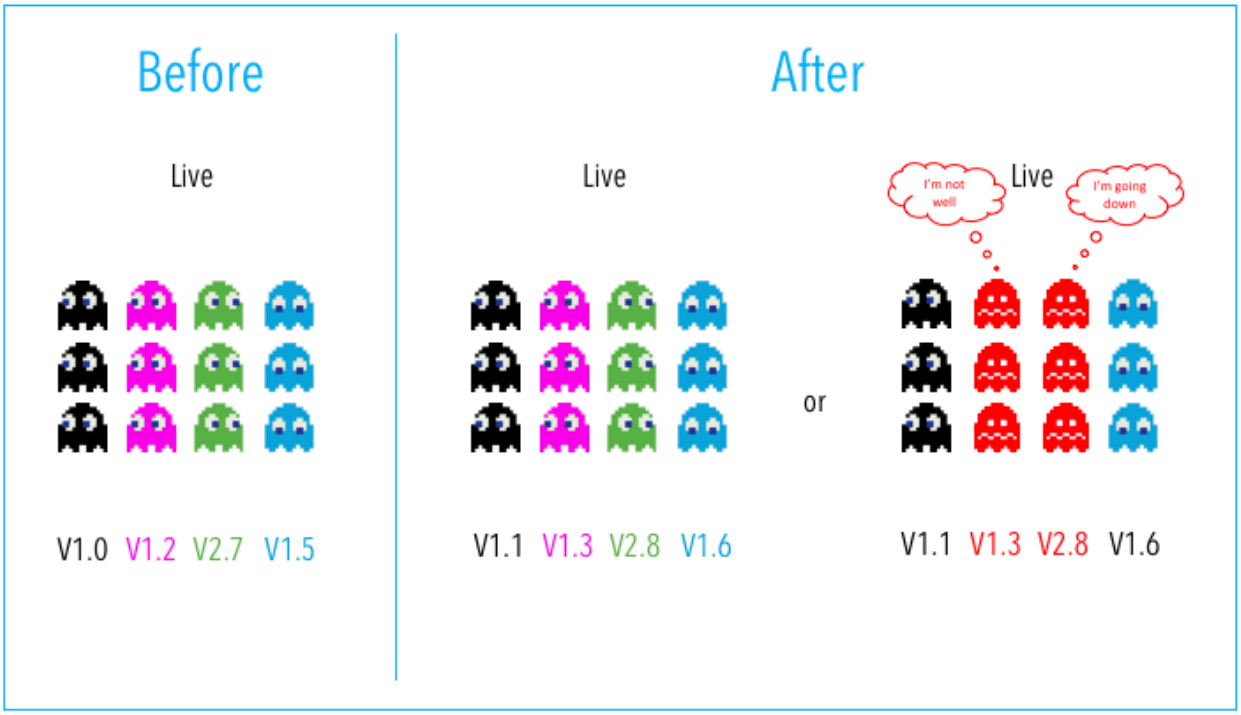
The benefits of this strategy are that it is simple, fast, and cheap. Use this strategy if 1) your application service is not business, mission, or revenue-critical, or 2) your deployment is to a lower environment, during off-hours, or with a service that is not in use.

Cons:

Of all the deployment strategies shared, it is the riskiest and does not fall into best practices. Basic deployments are not outage-proof and do not provide for easy rollbacks.

The Multi-Service Deployment

In a multi-service deployment, all nodes within a target environment are updated with multiple new services simultaneously. This strategy is used for application services that have service or version dependencies, or if you're deploying off-hours to resources that are not in use.



Pros:

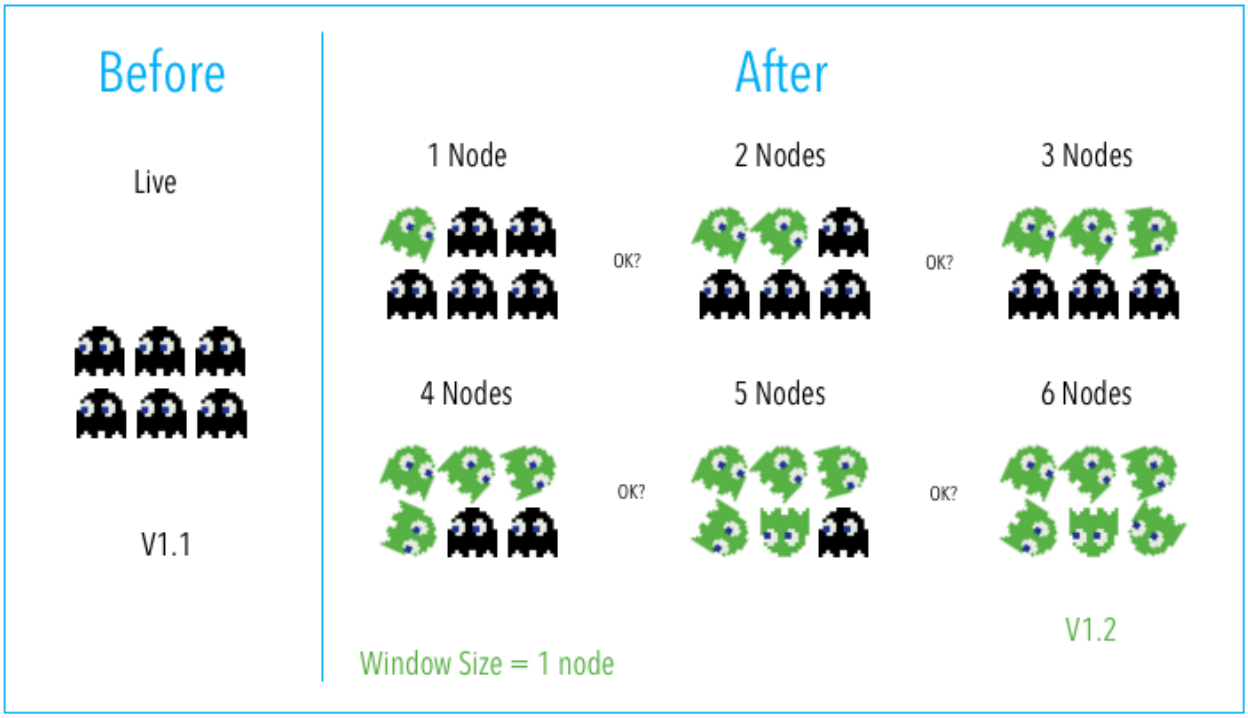
Multi-service deployments are simple, fast, cheap, and not as risk-prone as a basic deployment.

Cons:

Multi-service deployments are slow to roll back and not outage-proof. Using this deployment strategy also leads to difficulty in managing, testing, and verifying all the service dependencies.

Rolling Deployment

A rolling deployment is a deployment strategy that updates running instances of an application with the new release. All nodes in a target environment are incrementally updated with the service or artifact version in integer N batches.



Pros:

The benefits of a rolling deployment are that it is relatively simple to roll back, less risky than a basic deployment, and the implementation is simple.

Cons:

Since nodes are updated in batches, rolling deployments require services to support both new and old versions of an artifact. Verification of an application deployment at every incremental change also makes this deployment slow.

Blue-Green Deployment

Blue-green deployment is a deployment strategy that utilizes two identical environments, a “blue” (aka staging) and a “green” (aka production) environment with different versions of an application or service. Quality assurance and user acceptance testing are typically done within the blue environment that hosts new versions or changes. User traffic is shifted from the green environment to the blue environment once new changes have been testing and accepted within the blue environment. You can then switch to the new environment once the deployment is successful.

Pros:

One of the benefits of the blue-green deployment is that it is simple, fast, well-understood, and easy to implement. Rollback is also straightforward, because you can simply flip traffic back to the old environment in case of any issues. Blue-green deployments are therefore not as risky compared to other deployment strategies.

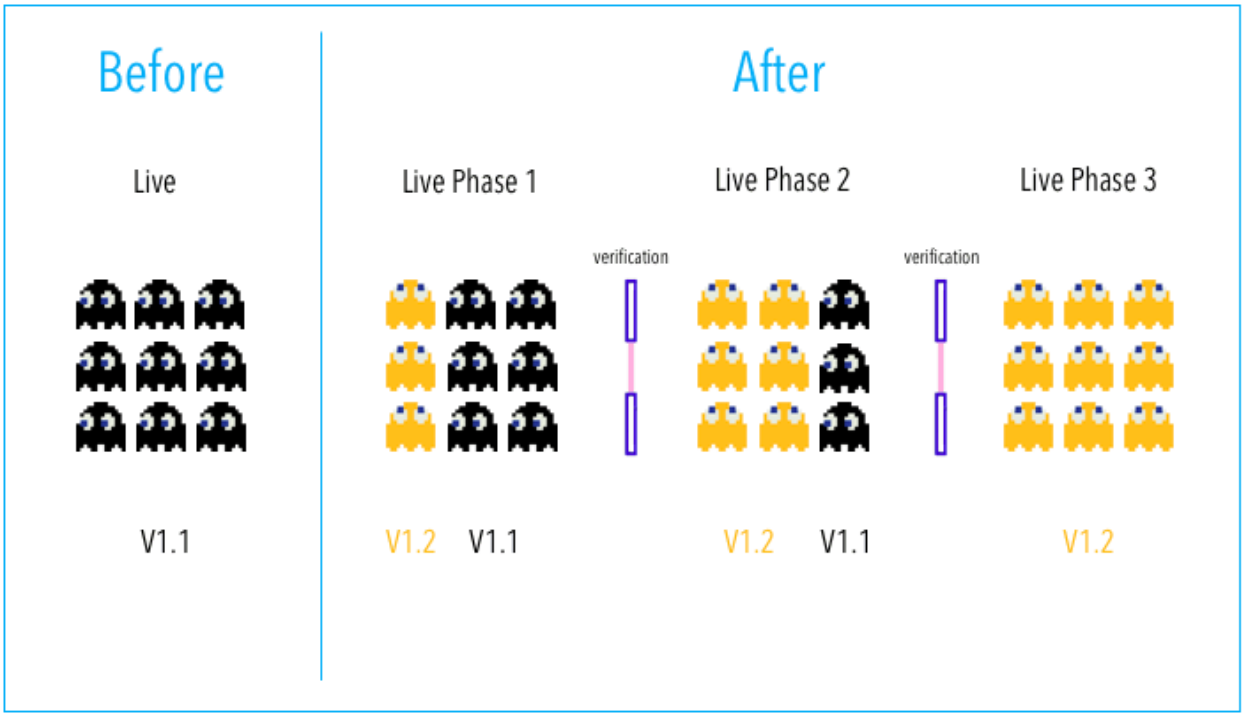
Cons:

Cost is a drawback to blue-green deployments. Replicating a production environment can be complex and expensive, especially when working with microservices. Quality assurance and user acceptance testing may not identify all of the anomalies or regressions either, and so shifting all user traffic at once can present risks. An outage or issue could also have a wide-scale business impact before a rollback is triggered, and depending on the implementation, in-flight user transactions may be lost when the shift in traffic is made.

Canary Deployment

A canary deployment is a deployment strategy that releases an application or service incrementally to a subset of users. All infrastructure in a target environment is updated

in small phases (e.g: 2%, 25%, 75%, 100%). A canary release is the lowest risk-prone, compared to all other deployment strategies, because of this control.



Pros:

Canary deployments allow organizations to test in production with real users and use cases and compare different service versions side by side. It's cheaper than a blue-green deployment because it does not require two production environments. And finally, it is fast and safe to trigger a rollback to a previous version of an application.

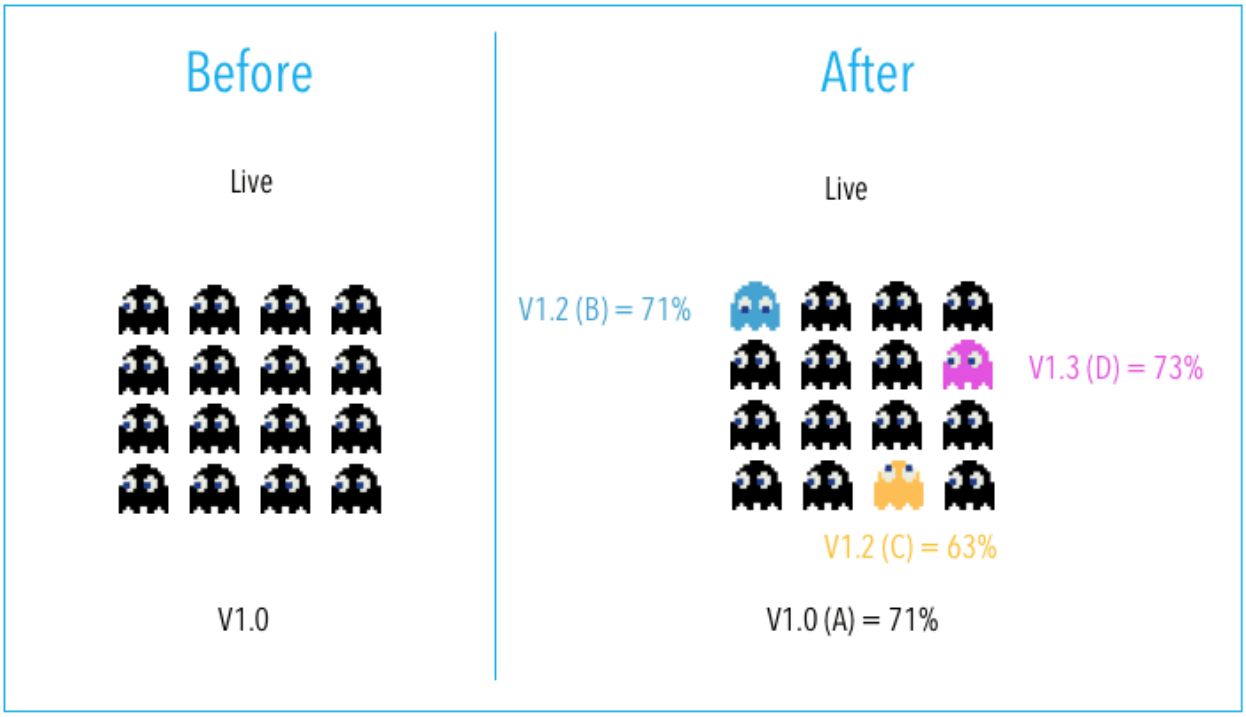
Cons:

Drawbacks to canary deployments involve testing in production and the implementations needed. Scripting a canary release can be complex: manual verification or testing can take time, and the required monitoring and instrumentation for testing in production may involve additional research.

A/B Testing

In A/B testing, different versions of the same service run simultaneously as "experiments" in the same environment for a period of time. Experiments are either

controlled by feature flags toggling, A/B testing tools, or through distinct service deployments. It is the experiment owner's responsibility to define how user traffic is routed to each experiment and version of an application. Commonly, user traffic is routed based on specific rules or user demographics to perform measurements and comparisons between service versions. Target environments can then be updated with the optimal service version.



The biggest difference between A/B testing and other deployment strategies is that A/B testing is primarily focused on experimentation and exploration. While other deployment strategies deploy many versions of a service to an environment with the immediate goal of updating all nodes with a specific version, A/B testing is about testing multiple ideas vs. deploying one specific tested idea.

Pros:

A/B testing is a standard, easy, and cheap method for testing new features in production. And luckily, there are many tools that exist today to help enable A/B testing.

Cons:

The drawbacks to A/B testing involve the experimental nature of its use case. Experiments and tests can sometimes break the application, service, or user experience. Finally, scripting or automating AB tests can also be complex.