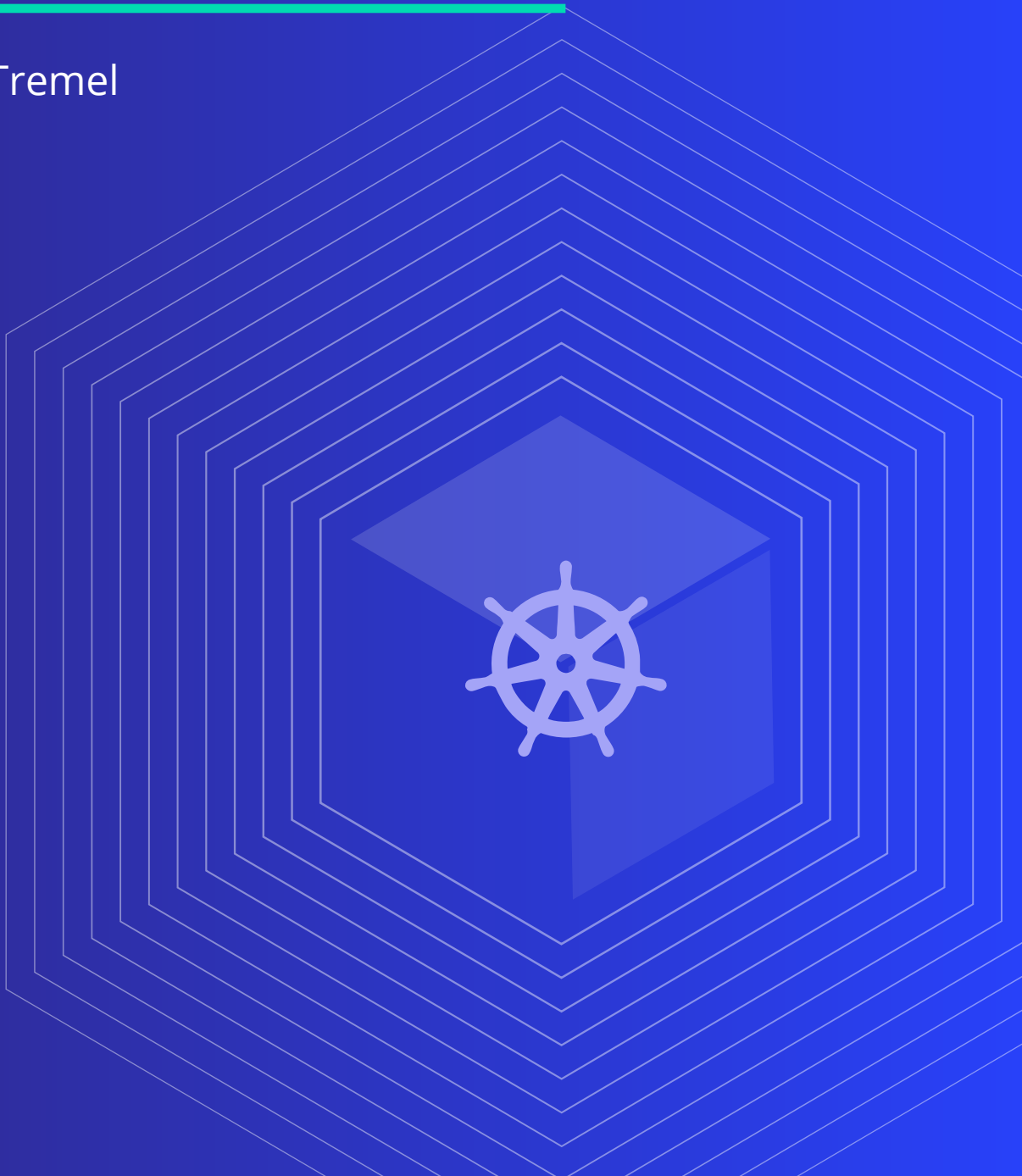
Container Solutions

6 KUBERNETES DEPLOYMENT STRATEGIES

by Etienne Tremel



By Etienne Tremel, software engineer at Container Solutions

In **Kubernetes** there are a few different ways to release an application, it is necessary to choose the right strategy to make your infrastructure reliable during an application update.

Choosing the right deployment

There are a variety of techniques to deploy new applications to production, so choosing the right strategy is an important decision, weighing the options in terms of the impact of change on the system, and on the end-users.

We are going to talk about the following strategies:

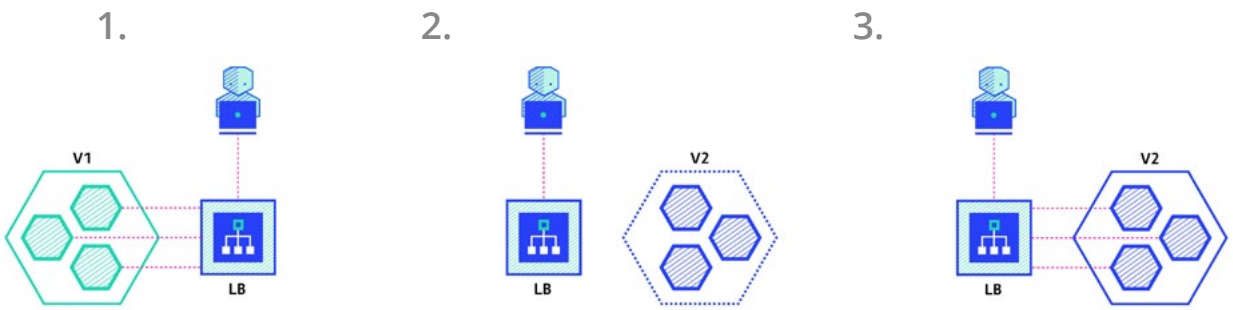
- **Recreate:** Version A is terminated then version B is rolled out.
- **Ramped** (also known as rolling-update or incremental): Version B is slowly rolled out and replacing version A.
- **Blue/Green:** Version B is released alongside version A, then the traffic is switched to version B.
- **Canary:** Version B is released to a subset of users, then proceed to a full rollout.
- **A/B testing:** Version B is released to a subset of users under specific condition.

- **Shadow:** Version B receives real-world traffic alongside version A and doesn't impact the response.

Let's take a look at each strategy and see which strategy would fit best for a particular use case. For the sake of simplicity, we used Kubernetes and tested the example against Minikube. Examples of configuration and step-by-step approaches on for each strategy can be found in this git repository.

Recreate - best for development environment

The recreate strategy is a dummy deployment which consists of shutting down version A then deploying version B after version A is turned off. This technique implies downtime of the service that depends on both shutdown and boot duration of the application.



Pros:

- Easy to setup.
- Application state entirely renewed.

Cons:

- High impact on the user, expect downtime that depends on both shutdown and boot duration of the application.

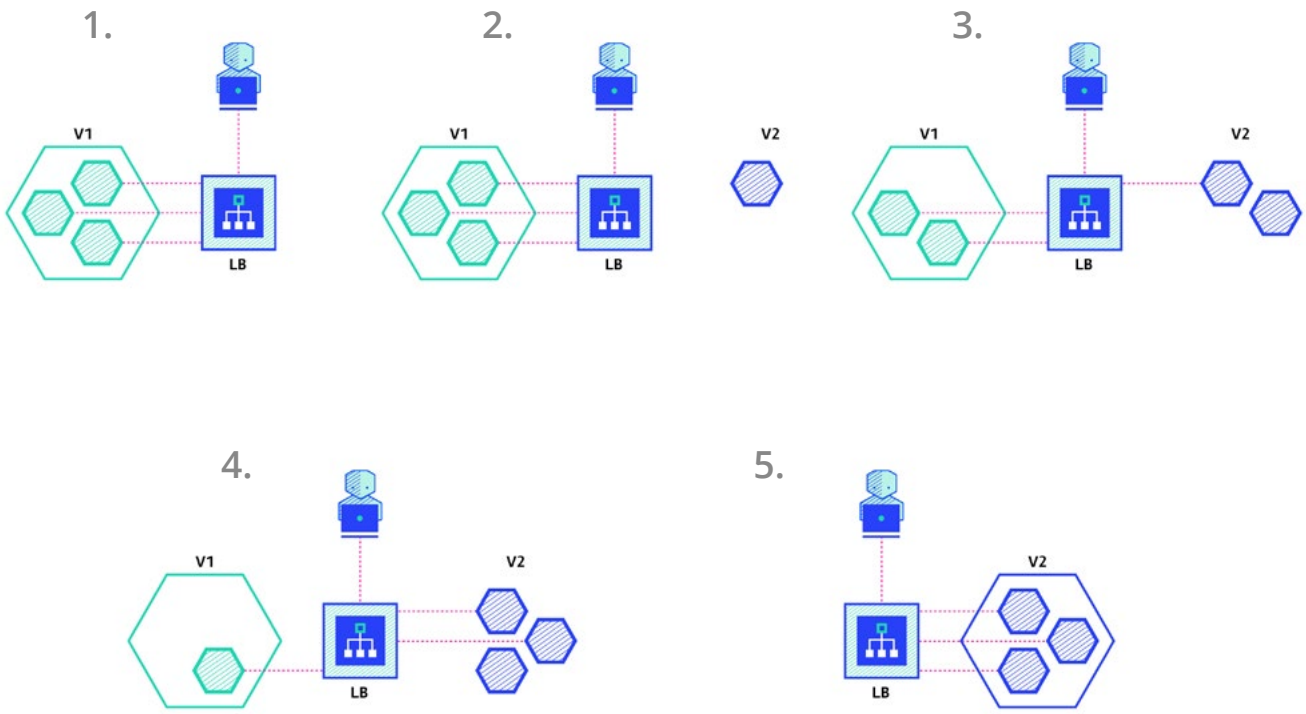
A full example and steps to deploy can be found at https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/recreate

Ramped - slow rollout

The ramped deployment strategy consists of slowly rolling out a version of an application by replacing instances one after the other until all the instances are rolled out. It usually follows the following process: with a pool of version A behind a load balancer, one instance of version B is deployed. When the service is ready to accept traffic, the instance is added to the pool. Then, one instance of version A is removed from the pool and shut down.

Depending on the system taking care of the ramped deployment, you can tweak the following parameters to increase the deployment time:

- Parallelism, max batch size: Number of concurrent instances to roll out.
- Max surge: How many instances to add in addition of the current amount.
- Max unavailable: Number of unavailable instances during the rolling update procedure.



Pros:

- Easy to set up.
- Version is slowly released across instances.
- Convenient for stateful applications that can handle rebalancing of the data.

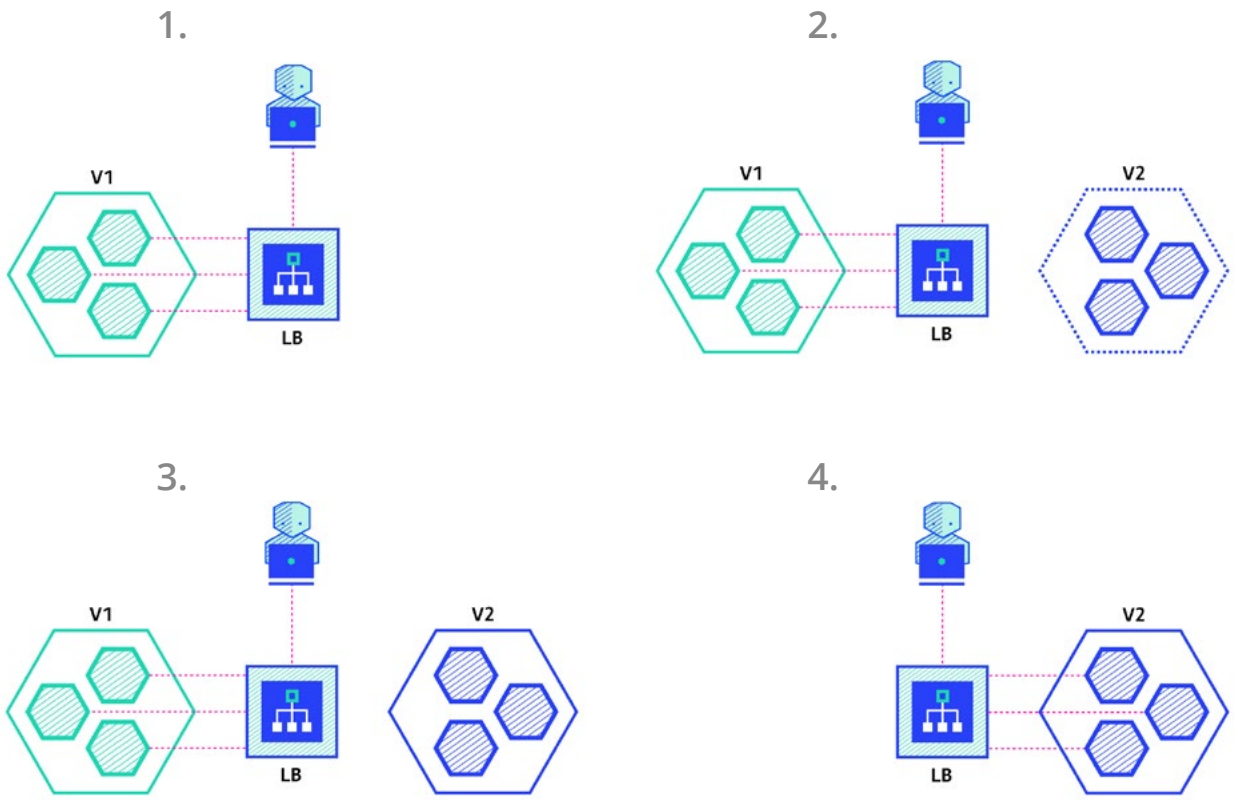
Cons:

- Rollout/rollback can take time.
- Supporting multiple APIs is hard.
- No control over traffic.

A full example and steps to deploy can be found at https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/ramped

Blue/Green - best to avoid API versioning issues

The blue/green deployment strategy differs from a ramped deployment, version B (green) is deployed alongside version A (blue) with exactly the same amount of instances. After testing that the new version meets all the requirements the traffic is switched from version A to version B at the load balancer level.



Pros:

- Instant rollout/rollback.
- Avoid versioning issue, the entire application state is changed in one go.

Cons:

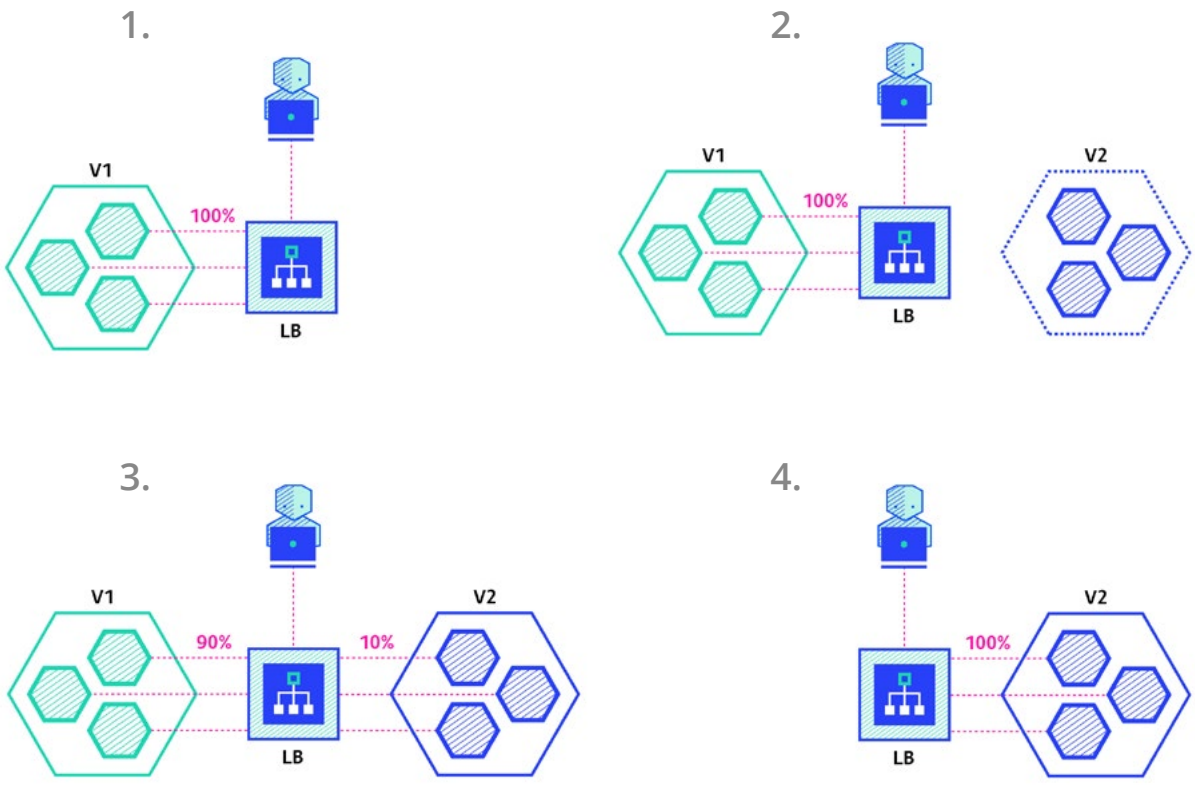
- Expensive as it requires double the resources.
- Proper test of the entire platform should be done before releasing to production.
- Handling stateful applications can be hard.

A full example and steps to deploy can be found at https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/blue-green

Canary - let the consumer do the testing

A canary deployment consists of gradually shifting production traffic from version A to version B. Usually the traffic is split based on weight. For example, 90 percent of the requests go to version A, 10 percent go to version B.

This technique is mostly used when the tests are lacking or not reliable or if there is little confidence about the stability of the new release on the platform.



Pros:

- Version released for a subset of users.
- Convenient for error rate and performance monitoring.
- Fast rollback.

Con:

- Slow rollout.

A full example and steps to deploy can be found at https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/canary

A/B testing - best for feature testing on a subset of users

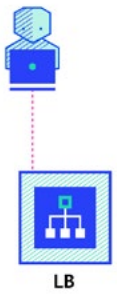
A/B testing deployments consists of routing a subset of users to a new functionality under specific conditions. It is usually a technique for making business decisions based on statistics, rather than a deployment strategy. However, it is related and can be implemented by adding extra functionality to a canary deployment so we will briefly discuss it here.

This technique is widely used to test conversion of a given feature and only roll-out the version that converts the most.

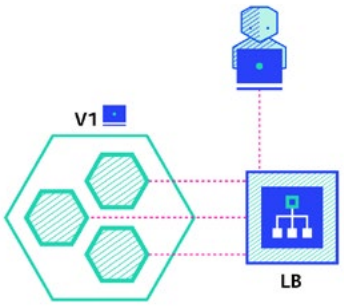
Here is a list of conditions that can be used to distribute traffic amongst the versions:

- By browser cookie
- Query parameters
- Geolocalisation
- Technology support: browser version, screen size, operating system, etc.
- Language

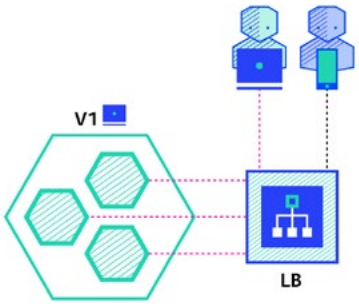
1.



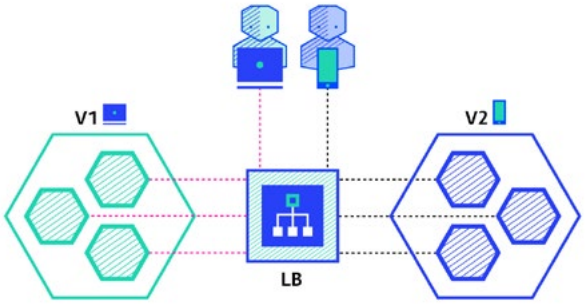
2.



3.



4.



Pros:

- Several versions run in parallel.
- Full control over the traffic distribution.

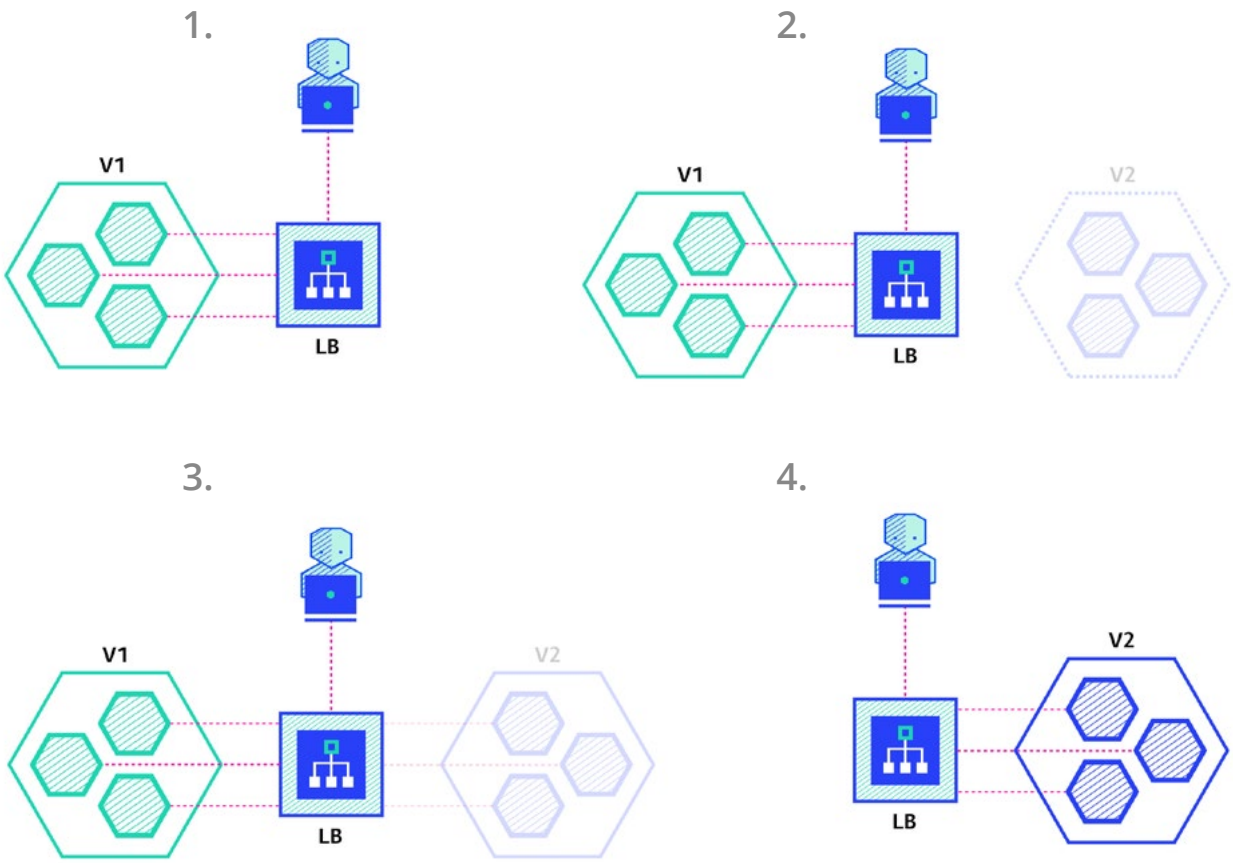
Cons:

- Requires intelligent load balancer.
- Hard to troubleshoot errors for a given session, distributed tracing becomes mandatory.

A full example and steps to deploy can be found at https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/ab-testing

Shadow

A shadow deployment consists of releasing version B alongside version A, fork version A's incoming requests and send them to version B as well without impacting production traffic. This is particularly useful to test production load on a new feature. A rollout of the application is triggered when stability and performance meet the requirements.



This technique is fairly complex to setup and needs special requirements, especially with egress traffic. For example, given a shopping cart platform, if you want to shadow test the payment service you can end-up having customers paying twice for their order. In this case, you can solve it by creating a mocking service that replicates the response from the provider.

Pros:

- Performance testing of the application with production traffic.
- No impact on the user.
- No rollout until the stability and performance of the application meet the requirements.

Cons:

- Expensive as it requires double the resources.
- Not a true user test and can be misleading.
- Complex to setup.
- Requires mocking service for certain cases.

A full example and steps to deploy can be found at https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/shadow

To Sum Up

There are multiple ways to deploy a new version of an application and it really depends on the needs and budget. When releasing to development/staging environments, a recreate or ramped deployment is usually a good choice. When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/ b testing or shadow release can be used. If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

Last but not least, a shadow release is complex and requires extra work to mock egress traffic which is mandatory when calling external dependencies with mutable actions (email, bank, etc.). However, this technique can be useful when migrating to a new database technology and use shadow traffic to monitor system performance under load.

DEPLOYMENT STRATEGIES

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

Container **Solutions**

Want To Read More?

Check out the rest of the Container Solutions blogs:

Deep Dive: Deployment Automation for Applications on Kubernetes (Part 1)

by Philipp Strube

Deep Dive: Deployment Automation for Applications on Kubernetes,(Part 2)

by Philipp Strube

More Public Kubernetes Courses Coming in 2020

by Russell Trow