**Deployment Strategies**

One of the phases of Software Development Lifecycle (SDLC) is Deployment. Like any other stage this holds a very important place in the lifecycle. In simple terms project deployment is a process of making the application ready to deliver to the end users. It involves multiple steps, systematic processes before the application is delivered to the customers. The process of deployment involves software release, installation, rigorous testing, and most importantly performance monitoring.

The deployment strategy is a method to alter or enhance an application. The primary concern in deployment is to make the change in such a way that the end-user does not feel any discomfort while using the application. To help overcome any issue while migration, there are several techniques available. The organization needs to ponder on the facts that which suits there requirement the best.

This article is going to focus on deployment strategies weighing each method’s pros and cons to get a clear idea of which strategy works best for you. As I mentioned earlier there are many deployment practices we will focus on below-mentioned approaches.

         Rolling strategy and canary deployments

         Recreate strategy

         Custom Strategy

         Shadow Strategy

         Blue Green Strategy

**Rolling Strategy and Canary Deployments**

The principal idea behind the rolling strategy is to wait for the new pods to become ready. The preparedness of pods is evaluated by a readiness check before taking down the old version. During this period if any serious issue occurs, the rolling deployment can be put on hold.

There is a common practice of employing canary deployments for OpenShift Container Platform. All the new instances are thoroughly tested before replacing them with old versions. If the readiness check does not signal okay, then the deployment is aborted by the team. The check is not an external mechanism; it is inbuilt in the code. The team can always make more stringent checks as per requirement.

**When to use**

         The canary technique carried out when the team is not sure of overall steadiness of the application. To ensure that the new version is working well with the platform it is rolled out in iteration.

**Pros**

         Convenient for performance testing.

         Fast retraction is possible in case of a major issues.

**Cons**

         The application rollout is time-consuming.

**Recreate Strategy**

As the name suggests the strategy works on the idea to deploy a dummy version. The new version is deployed when the old version has been closed down. This approach results in significant downtime as both the instances of applications cannot run simultaneously. Let’s look at the situations which are best suited for, pros and cons.

**When to use**

         The approach is ideal for situations before starting a new development task.

         When you do not support having version A and version B of your program code operating at the same time.

**Pros**

         Easy to put applications together.

         Application gets revamped.

**Cons**

         Downtime of the application is a major drawback.

**Custom Strategy (A/B testing)**

The approach is a little different than the above two strategies discussed above. Instead of rolling out a new version directly a part of the application is introduced to the users. This move is considered more of a business strategy rather than a deployment technique. To make this strategy successful a part of the new application version is rolled out first. Based on the data, statistics, and usage by the end users that part is taken into consideration for final deployment. But for this, we need to make some additional changes in order to distribute the traffic from old to new. Some of them are:

         By using browser cookies

         Query parameters

         Using GPS and IP to get the location of the devices/users.

**Pros**

* Negligible downtime as the new version is tested in conjunction with the old version.
* Traffic among new and old can be efficiently managed.

**Cons**

         Requires a strong load balancing mechanism.

         Troubleshooting can become tricky due to double version management.

**Shadow Deployment**

A shadow deployment principally revolves around the idea of keeping both versions. The old version is not discarded instead the traffic of the old version is diverted to the new version. This way the production is not hampered and the job also gets done. This is particularly effective to test the production load on the new version. Final deployment is done when the application meets all the stability and performance factors.

This approach requires some arrangements to be performed for successful implementation. Let’s consider an example of a payment gateway, one challenge that you can face while implementing is that user might end up paying twice for the same order. This is fairly simple to solve by creating a mocking service you can create a dummy service that replicates the response from the provider.

**Pros**

         No separate performance testing required to be done.

         Minimal impact on the user and production.

**Cons**

         The setup can be quite expensive.

         Additional arrangements like mocking services are required.

**Blue Green Strategy**

Blue-green deployment uses two versions of an application simultaneously. The old version is used is called the green version and the new version is called the blue version. This arrangement requires two configurations. If the testing version clears all the checks then traffic is redirected to a new version. If required, there is also a facility to roll back to the green, version by switching the production back to the previous version.

**Pros**

         Switching back and forth between the versions is easy.

         No versioning issue.

**Cons**

         Expensive as it requires managing dual versions.

         Managing both applications can be tedious task.

There are multiple ways to manage the deployment strategy but it totally depends on the business requirement. So, it is advisable to have in-depth knowledge of application, available resources, end-user requirements, and cost that it is going to incur on the organization. Considering these options you can go for any of the choices listed above.