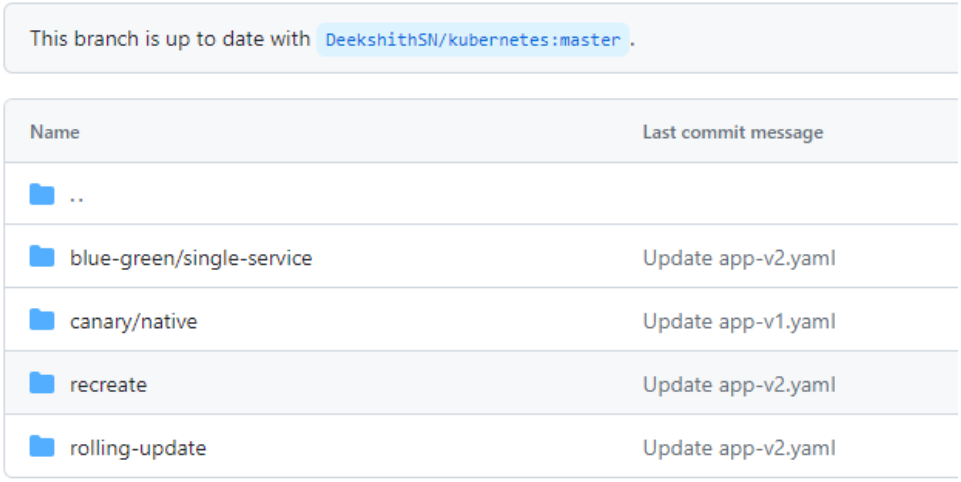
AmazingDevOps

Deployment Strategies



Blue-Green/Single-service

Blue-green deployment is a strategy used in software development to release new versions of an **application without causing downtime or disruptions for users.**

Imagine you have a web application called "my-app" with an initial version (v1.0.0) running. In blue-green deployment, you create a duplicate environment, which is identical to the existing one. This **duplicate environment is called the "green" environment**, while the existing one **is called the "blue" environment.**

To release **a new version (v2.0.0) of your application**, you set up the new version in the green environment. However, initially, the traffic **is still routed to the blue environment**, so users continue to use the **old version without any interruption**.

Once the green **environment is ready and you have tested the new version thoroughly**, you switch the traffic from the **blue environment to the green environment**. Now, users start accessing the new version of the application.

The blue environment is kept as a backup in case any issues arise with the new version. If any problems occur, **you can quickly switch back to the blue environment**, which still has the old version running. This allows you **to rollback easily without causing significant disruptions for your users**.

The configuration you provided seems to be an example of blue-green deployment in Kubernetes. **It includes the creation of two services ("my-app") and two deployments ("my-app-v1" and "my-app-v2") for the two versions of the application.** The services help route the traffic to the respective deployments based on **their labels and selectors**.

Blue-green deployment **is sometimes referred to as "single service" because it involves using a single service endpoint to handle traffic for both the blue and green environments**.

In a blue-green deployment, you have two environments: **the blue environment**, which represents the existing or currently active version of the application, and **the green environment**, which represents the new version being deployed. Both environments have their **own deployment and associated resources**.

However, from the perspective of the users or client applications accessing the application, **there is only one service endpoint exposed**. This service endpoint handles the incoming requests and directs them either to the blue environment or the green environment based on the routing configuration.

Canary Deployment

A canary deployment is a strategy used in software development to test a new version of an application in a controlled manner before rolling it out to all users. Imagine you have an application called "my-app" with an existing version (v1.0.0) running. In a canary deployment, you introduce a small percentage of users or traffic to the new version (v2.0.0) while most users still use the old version.

To set up a canary deployment, you create a separate environment for the new version, often with a smaller number of instances or replicas compared to the existing version. This environment is called the "canary environment." Initially, only a small portion of incoming requests or users are directed to the canary environment, while the majority still go to the existing version. This allows you to monitor the behavior and performance of the new version with a limited audience. If the canary environment performs well and shows no issues or errors, you can gradually increase the traffic directed to it. This way, you can observe how the new version behaves under higher loads and gather more feedback.

However, if any problems are detected, such as increased error rates or performance issues, you can quickly redirect the traffic back to the existing version, minimizing the impact on most users.

We have two configurations that include the creation of two services ("my-app") and two deployments ("my-app-v1" and "my-app-v2") for the two versions of the application. The canary version (v2.0.0) has fewer replicas and receives a smaller portion of traffic compared to the existing version (v1.0.0).

Rolling updates

Rolling update is a deployment strategy that allows you to update your application without causing any downtime or disruptions to the users.

Imagine you have an application called "my-app" running with multiple instances or replicas. In a rolling update, you want to release a new version of the application while keeping the application available and accessible to users throughout the update process.

With rolling update, you define the desired number of replicas for your application, and then the deployment system updates the instances gradually, one by one. This means that only a small portion of instances are updated at a time, while the rest continue running the older version.

The process typically involves these steps:

1. Start with a set number of replicas of the current version running and serving user requests.
2. Introduce a new version of the application by creating updated instances, one by one, alongside the existing instances.
3. The deployment system monitors the health and status of each updated instance, ensuring it is ready to handle requests before moving on to the next one.
4. Once an updated instance is confirmed to be healthy and ready, the deployment system gradually scales down the replicas of the older version.
5. The process continues until all instances are running the new version, and the update is considered complete.

The rolling update strategy allows for a smooth transition from the old version to the new version, ensuring that the application remains available and responsive during the update process. **It reduces the risk of downtime and allows for easy rollback in case any issues are encountered.**

Our configuration specifies the desired number of replicas (5) for the "my-app" deployment and defines the rolling update strategy. The strategy includes parameters such as the maximum number of additional instances that can be added at a time (maxSurge) and the maximum number of instances that can be unavailable during the update (maxUnavailable). **This ensures controlled and gradual updates without overwhelming the system or causing service disruptions.**

What can you use to control the traffic routing in canary deployment

To control traffic routing in a canary deployment, you can use various mechanisms and tools. Here are a few commonly used options:

1. Load Balancers: Load balancers can be configured to distribute traffic between different versions of your application. You can set up rules or weights to control the proportion of traffic that is directed to the canary version versus the stable version.
2. Ingress Controllers: Ingress controllers act as an entry point for incoming traffic to your cluster. They often support routing rules and can be configured to route traffic based on various criteria such as path-based routing, HTTP headers, or source IP addresses. By defining appropriate rules, you can direct traffic to the canary version or the stable version of your application.

3. Service Meshes: Service meshes like Istio or Linkerd provide advanced traffic control capabilities. They allow you to define traffic routing rules, conduct A/B testing, and implement canary releases. Service meshes can intercept network traffic between microservices and enable fine-grained control over routing decisions based on factors like request headers, source, or destination service.
4. **Kubernetes Native Features: Kubernetes itself offers several native features for traffic control. You can use labels and selectors to route traffic based on pod labels or annotations. Kubernetes also provides the concept of "Service" resources, which act as a stable endpoint for your application and can be used to control traffic routing to different versions.**
5. Custom Application Logic: In some cases, you may implement custom application logic to control traffic routing. This could involve using feature flags or conditional logic within your application code to selectively route requests to the canary version or the stable version based on certain conditions or percentages.

The choice of which tool or mechanism to use depends on your specific requirements, the complexity of your infrastructure, and the level of control you need over traffic routing. It's common to use a combination of these approaches to achieve the desired traffic control in a canary deployment.