Vadym Tkachenko
Yanina Petrova
Gabriel Rafael

Date : 12/06/2023

KUBERNETES CLUSTER UPGRADES WITHOUT DISTRUPTION

Research Document

Fontys
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Revision Table	Description	Date	Author	Checked by	Approved by
V0.1	First draft	21-03-2023	Students	Nathan Keyaerts	
V0.2	Changes made according to the feedback	06-06-2023	Students	Haverkort, Frank	
V0.3	Added main question conclusion	06-12-2023	Students		

Table of Contents

Introduction

Over the past few years, Kubernetes has established itself as the leading container orchestration platform, empowering organizations to efficiently manage and scale their containerized applications. As its popularity grows, it undergoes more rapid development, incorporating different user demands, better security practices and bug fixes with each new version. Upgrading a Kubernetes cluster has become increasingly important for maintaining organization operations and providing consistent services.

Manual cluster upgrades can be complex, time-consuming, and error-prone, requiring meticulous planning, coordination, and human intervention. To address these challenges, there is a growing need to explore automated approaches for upgrading Kubernetes clusters. Automation provides numerous benefits, including reduced operational burden, faster deployment of new features, increased reliability, and minimized downtime.

This research paper aims to investigate the concept of automating Kubernetes cluster upgrades, providing a comprehensive approach to streamline the process while ensuring the smooth functioning of the cluster. The primary objective is to explore the available strategies and techniques that allow clusters to be upgraded seamlessly, eliminating manual intervention, and reducing the risks associated with human error.

To achieve these objectives, this research paper will analyze existing literature, examine use cases, and conduct experiments to evaluate the effectiveness and feasibility of automated Kubernetes cluster upgrades.

Assignment

As Kubernetes continues to evolve, new versions are released, introducing improved features, bug fixes, and enhanced security measures. To ensure the smooth functioning of a Kubernetes cluster, it is essential to research and understand the process of upgrading the cluster to newer versions.

When upgrading a Kubernetes cluster to a newer version, it is important to have a clear understanding of the different components that make up a Kubernetes cluster. The cluster consists of a control plane, responsible for managing the overall cluster state and coordinating operations, which includes the Kubernetes API server, etcd (distributed key-value store), kube-scheduler, and kube-controller-manager. The worker machines in the cluster, known as nodes, handle task execution and run containers. Each node runs kubelet, which communicates with the control plane and manages the containers' lifecycle. Understanding these different components and their interaction within the cluster is essential for successfully upgrading a Kubernetes cluster and ensuring its smooth functioning.

This research aims to explore various strategies, tools, and best practices when it comes to upgrading a Kubernetes cluster version. To assist with our research, we have set the following research question.

The following is our main question:

How to upgrade a Kubernetes cluster with no service disruption?

In addition to the main research question, there are also sub-questions:

- What is the best strategy to use when upgrading a Kubernetes cluster?
- How to measure the availability of the application during the upgrade?
- What tools can be used to upgrade a Kubernetes cluster?
- How to deal with potential data inconsistency issues/loss during an upgrade?
- How to implement rollback in case the test fails?
- How to make the solution cloud-agnostic?

.

Requirements

For the requirements, we used the MoSCoW method to help us narrow down our requirements. The following are the requirements for the solution provided by the customer:

- Must Haves
    - Services must be available during the upgrade.
    - A rollback must be in place in case of failure during the upgrade.
    - Kubernetes must be running at least a web server, API, and Database for the proof of concept.
    - Why an upgrade strategy was chosen must be defined.
    - Evaluate multiple upgrade strategies.
- Should Haves
    - The use of the Blue/Green strategy

    - Testing on a non-production environment before applying on a production environment.

- Could Haves
    - Can be provided a list of clusters to upgrade.

Research Questions

What is the best strategy to use when upgrading a Kubernetes cluster?

Currently, there is no specific strategy in place for updating the version of a Kubernetes cluster. However, there are deployment strategies available for effectively deploying newer versions of software applications. Although these deployment strategies are not specifically designed for updating the version of Kubernetes itself, they can still be leveraged to assist us in achieving our objective. The following are the deployment strategies that best suit our goals for this project.

- Rolling Update
- Blue Green
- Canary

Rolling Update

What is Rolling Update Strategy?
The Rolling update strategy (also known as Ramped strategy) is a deployment strategy used in software development and deployment to update a system in a controlled and gradual manner. The Rolling update strategy gradually updates an instance of an application or system, one at a time, while ensuring that the application remains available and operational throughout the update process. The Rolling update strategy is one of the two strategies available for managing the deployment of an application inside Kubernetes. Rolling Update is the default.

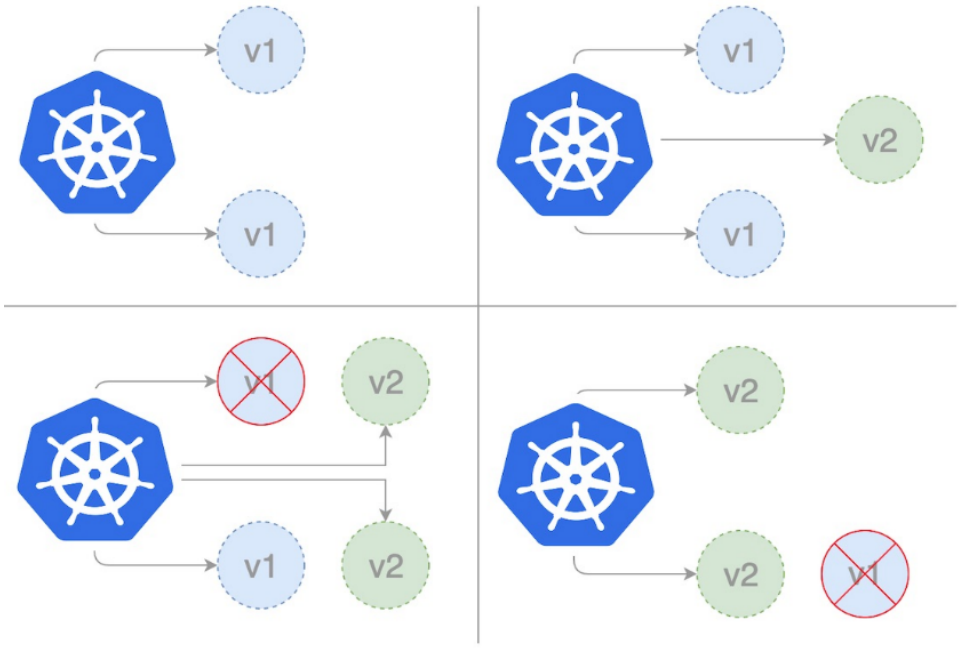

FIGURE 1 ROLLING UPDATE STRATEGY

As it can be seen in Figure 1, rolling update consists of creating a node with the newer version while draining the node with the older version. This process is repeated for each node in the cluster until all nodes are replaced ensuring that the application remains available throughout the cluster update.

How is Rolling Update Strategy used?
Inside Kubernetes, the Rolling Update Strategy is used to manage and update deployments. For this project we need to update the Kubernetes version and not the deployments, how can the Rolling update be used in this scenario then?

Instead of updating deployments, the Rolling Update Strategy will be used to gradually update the Kubernetes control plane and worker nodes. To perform this, we must perform a workload migration. This is necessary to remove workload from the old nodes and move it to the updated ones. This process can be divided into four main points of interest:

1. Draining the node: When it is time to update a node, the Rolling Update Strategy first drains the node by evicting the existing workloads. This ensures that the node is prepared for the update without any active workloads.
2. Scheduling the workload on other nodes: While the node is being drained, the workloads running on that node are automatically scheduled onto other available nodes in the cluster. The Kubernetes scheduler intelligently selects appropriate nodes based on resource availability and other constraints defined in the pod configuration.
3. Updating the drained node: After the workloads have been successfully migrated, the drained node can be updated with the required changes, such as upgrading the Kubernetes version or applying patches. This may involve restarting the node or performing necessary maintenance tasks.
4. Resuming the workload on the updated node: Once the node update is complete, the Rolling Update Strategy allows the workloads to gradually return to the updated node. The Kubernetes scheduler ensures the workloads are distributed across the cluster, considering factors like resource availability and load balancing.

By repeating these steps for each control plane and worker node in a rolling manner, the entire cluster can be updated without causing significant downtime or disruption to the running applications.

*This is a general view

What are the advantages?

- Minimize/zero downtime.
    - The Rolling Update strategy updates one node at a time, ensuring that at least one node is always available.
- Cost
    - The Rolling Update strategy requires extra resources, and extra resources equal extra cost. But compared to other strategies, the Rolling Update strategy is relatively low cost since it only duplicates nodes one at a time and not the cluster at once.
- Scalable
    - The Rolling Update strategy can scale to a larger number of nodes.

What are the disadvantages?

- Rollback duration
    - Because the Rolling Update strategy updates nodes one at a time, it would need to do the same when rolling back. Meaning it will take longer to roll back compared to other strategies.
- Update duration
    - Because the Rolling Update strategy updates nodes one at a time, it takes longer to complete compared to other strategies.
- Complexity
    - The Rolling Update strategy requires a good understanding of the cluster being updated as updating nodes one at a time can be complex.

What are the risks and how to minimize them?

- Downtime
    - The rolling update strategy is designed to minimize downtime, but there will always be a risk of downtime during the update process. To help minimize this risk, it is advised to:
        - Carefully plan and schedule updates.
        - Test the updates in an environment other than production.
- Version Compatibility
    - The rolling update strategy requires compatibility between the old and new versions. If there are compatibility issues between the versions, the update process can fail, causing downtime or other issues. To help minimize this risk, it is advised to:
        - Test the updates in environments other than production.

It is however important to note that backwards compatibility cannot always be guaranteed. Kubernetes releases may introduce changes, deprecations, or removal of features, APIs, or behaviors that could impact existing deployments. Hence why it is of great importance to test the update in a different environment. Such compatibility issues can be avoided by carefully monitoring the process and performing a rollback when necessary.

Blue-Green

What is Blue-Green update strategy?
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 the latest changes have been tested and accepted within the blue environment.

How can the Blue-Green Strategy be used?
The Blue-Green deployment strategy is primarily used for deploying and managing application updates, rather than specifically upgrading the version of a Kubernetes cluster. However, it can indirectly facilitate the process of upgrading a Kubernetes cluster version.

To utilize the Blue-Green strategy for upgrading the Kubernetes cluster version, the following steps can be taken:

- Set up a new Kubernetes cluster:
    - Create a new cluster with the desired upgraded version of Kubernetes alongside the existing cluster.
- Switch traffic to the upgraded cluster:
    - Gradually redirect the traffic from the existing cluster to the upgraded cluster.
        - This can be achieved by updating load balancer settings, DNS records, or service configurations.
- Monitor and validate:
    - Validate that all applications and services are running smoothly.

*This is a general view

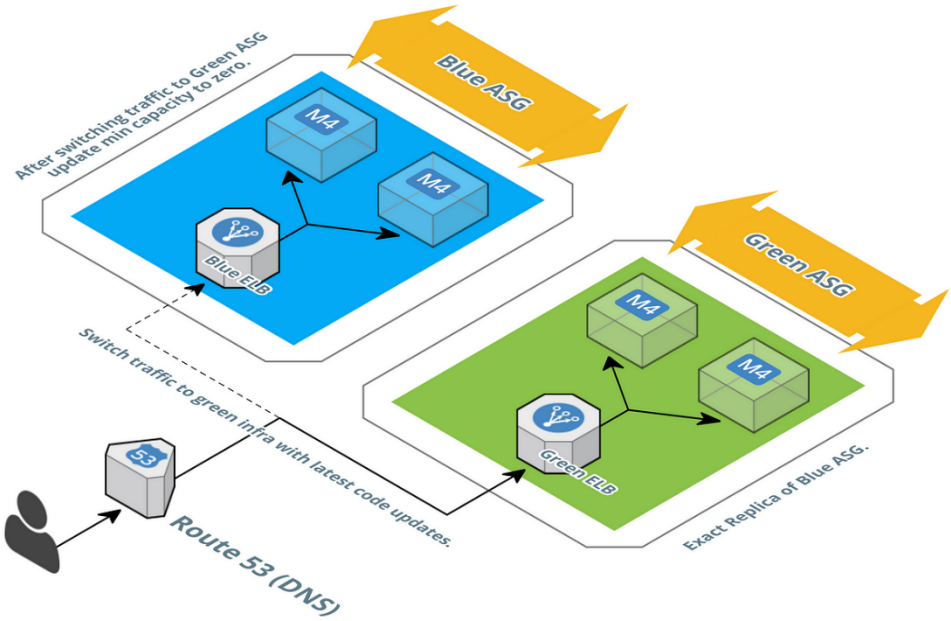

FIGURE 2. BLUE GREEN DEPLOYMENT STRATEGY (ON AWS)

What are the advantages?

- Zero-downtime:
    - The new version of the cluster is deployed in a separate environment (green) alongside the existing version (blue). Traffic is then gradually switched from the blue environment to the green environment, ensuring availability.
- Quick rollback:
    - If any unexpected issues or bugs are discovered in the green environment, rolling back to the blue environment is as simple as redirecting traffic back.
- Testing in a production-like environment
    - Blue-Green strategy allows for testing of the updated version in a production-like environment before fully transitioning traffic to it. This helps identify and address any compatibility, performance, or stability issues before they impact users.

What are the disadvantages?

- Cost:
    - Maintaining two separate environments (blue and green) simultaneously requires additional resources. This additional resource can increase costs and resource utilization, especially for larger deployments.
- Update duration:
    - Gradually transitioning traffic from the blue to the green environment can take time, particularly for larger deployments or applications with extensive user traffic. The extended deployment time may lead to a longer overall update duration.
- Scalability:
    - Blue-Green deployments may pose challenges in managing large-scale environments. When dealing with many nodes or highly distributed systems, duplicating the entire environment can be complex and resource intensive.

What are the risks and how to minimize them?

- Traffic routing issues:
    - Incorrect configuration or issues in traffic routing can lead to service disruptions or incorrect load balancing.
        - To help minimize this risk, it is advised to:
            - implement testing of traffic routing configurations, use traffic management services, and monitor traffic patterns during the transition to identify and resolve any routing issues promptly.
- Data consistency and synchronization:
    - When transitioning between blue and green environments, ensuring data consistency and synchronization can be challenging.
        - To help minimize this risk, it is advised to:
            - implement data replication system, perform testing of data migration processes, and closely monitor data integrity during the transition.

Canary

What is Canary update strategy?

A canary update is a strategy where a new version of an application is initially deployed to a small subset of users or servers, often referred to as the canary group. This group is selected to be representative of the broader user base or production environment. The new version is then tested on this subset of users/servers before it is rolled out to the entire user base or production environment. If any issues are detected during the canary phase, the rollout can be stopped, and the changes can be rolled back.

There are 2 subtypes of Canary deployments:

Side-by-side Canary – similar to the blue/green update strategy, but staged, requires a copy of existing infrastructure, therefore more costly.

Rolling Canary – like rolling update strategies, but an update is rolled out for a small subset of users, called a “canary group”, which is then monitored for a specific period. If no issues are observed, the update is rolled out for the rest of the users.

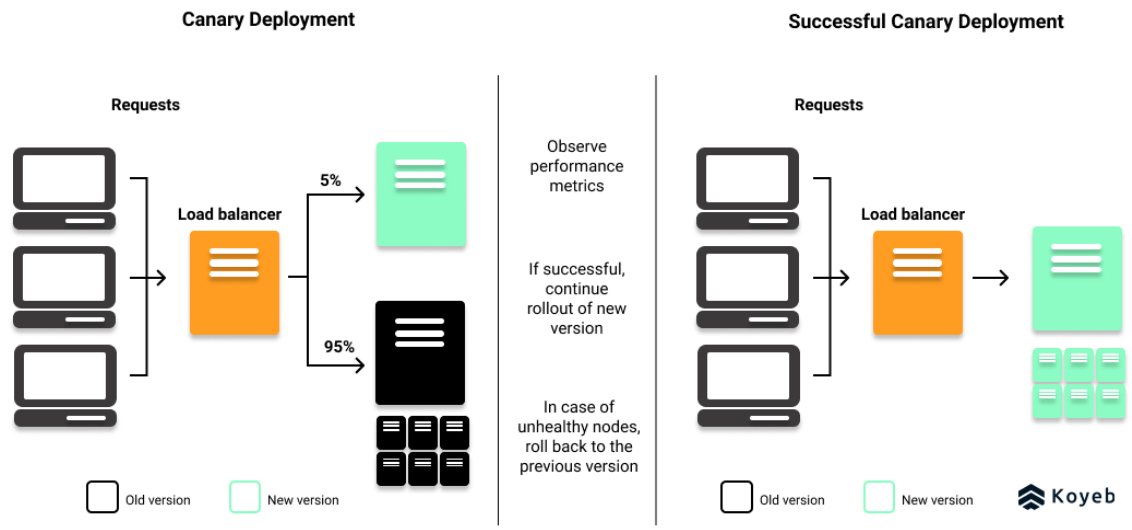

FIGURE 3. CANARY UPGRADE STRATEGY

In Figure 3 we can see a percentage of the users being routed to an updated version of the cluster. Through thorough monitoring of the process and the canary group, it can be determined whether the cluster upgrade should continue, or a rollback should be performed.

What are the advantages?

- Reduced risk:
    - Canary deployment allows for a controlled release of new features or changes to a small group of users or systems, which reduces the risk of any unforeseen issues impacting the entire system.
- Early feedback:
    - Canary deployment allows for early feedback from a small group of users or systems, which can help identify any issues or bugs early on in the deployment process.
- No downtime:
    - Like blue-green deployments, a canary deployment does not generate downtime.

- Easy rollback:
    - If something goes wrong, we can easily roll back to the previous version.

What are the disadvantages?

- Frustration:
    - The first group using the canary will find the worst bugs. What is more, some users may be put off learning they were used as guinea pigs.
- Costs:
    - Side-by-side deployments' cost is higher because we need extra infrastructure.
- Complexity:
    - Canary deployments share the same complexities as blue-green deployments. Having many production machines, migrating users, and monitoring the new system; are complicated tasks.
- Time:
    - Setting up a healthy canary deployment pipeline takes time and effort. On the plus side, once we get it right, we can do more frequent and safer deployments.
- Databases:
    - The problem is the database must simultaneously work with the canary and the control versions during the deployment. So, if we have breaking schema changes, we are in trouble. We need to maintain backward compatibility as we make changes, which adds another layer of complexity.

How to pick users for the “canary group”?

- Randomly:
    - We can send a percentage of users to the canary by random chance.
- By region:
    - Deploy the canary one geographical area at a time. For example, we could choose a follow-the-night strategy and release during each region’s nighttime, when there are the least users online.
- Early adopter program:
    - Giving users a chance to opt-in (or opt-out) to the canary program might lead to the best results. Early adopters are more likely to offer quality feedback.
- Dogfooding:
    - This term relates to the saying “eating your dog food” and involves releasing the canary to internal users and employees first.

How to measure the availability of an application during an upgrade?

There are many things to take into consideration when measuring the availability of an application during an upgrade. First, we need to define what counts as availability.

There are many ways to define an application available, but for this project, we ask the stakeholders a few questions to better define availability in this project. **The conclusion we came to is that availability is having the pod (where an application will be running) running with the status healthy.**

Now that we have availability defined, we can define our key performance indicators (KPIs) that will be used to measure availability. The following are KPIs that can be used and their description:

- Pod status:
    - Pod status indicates the status of the pod, such as whether it is running, pending, or has failed. A healthy pod should have a running status.
- Health checks:
    - Health checks indicate the status of the liveness and readiness probes configured for the pod. A failed liveness or readiness probe can cause the pod to be restarted or taken out of service.
- Restart count:
    - Restart count indicates the number of times a pod has been restarted. A high restart count can indicate that the pod is experiencing issues or failures.
- Network connectivity:
    - Network connectivity indicates whether the pod can communicate with other pods or services in the cluster.

If by the end of the upgrade all KPI (key performance indicators) indicate that the pod is heathy, we can conclude the upgrade a success.

What tools can be used to upgrade a Kubernetes cluster?

There are many different tools out there that can be used to help automate the Kubernetes cluster version upgrade. We narrow down the result to the following three tools:

- Kubeadm
- Kops
- Kubespray

Kubeadm

What is Kubeadm (Key Feature)?
Kubeadm is a tool provided by the Kubernetes project to bootstrap and manage Kubernetes clusters. Kubeadm can be automated to help with many manual tasks involved in cluster initialization, such as configuring the control plane components, securing the cluster, and joining worker nodes.

How does Kubeadm handle upgrade?
Kubeadm provides a straightforward process for upgrading the Kubernetes cluster. It follows a rolling upgrade strategy, which means that control plane components and worker nodes are upgraded one by one while the cluster remains available. Kubeadm handles other the necessary steps, including upgrading the control plane components, updating kubelet on worker nodes,etc.

What are the advantages and disadvantages?
Advantages of Kubeadm:

- Official tool
    - Kubeadm is an official tool from the Kubernetes project, ensuring reliability and alignment with the Kubernetes ecosystem.
- Good documentation
    - The tool provides clear documentation and a well-defined workflow, making it accessible for both beginners and experienced users.

Disadvantages of Kubeadm:

- Limited focus
    - Kubeadm is focused on cluster initialization and upgrades, and it may not cover more advanced cluster management features that other tools offer.
- Complex scripts needed
    - It requires additional manual steps or scripts to handle complex configurations or customizations beyond the basic functionalities provided by Kubeadm.

Kops

What is Kops (Key Feature)?
Kops (Kubernetes Operations) is a powerful command-line tool that automates the creation, upgrade, and management of Kubernetes clusters on AWS and GCP. Its key feature lies in its ability to automate the provisioning and management processes of Kubernetes clusters specifically tailored for the AWS environment. By using Kops, you can define your desired cluster state through configuration files, and the tool handles the creation of the necessary cloud resources, as well as the setup of the Kubernetes control plane and worker nodes.

How does Kops handle upgrades?
When it comes to upgrading a Kubernetes cluster using Kops, the tool supports rolling upgrades. This means that Kops can update each node in the cluster one by one, without causing any downtime. Rolling upgrades allow you to transition your cluster to a new Kubernetes version while ensuring that your applications remain available. Kops manages the process by carefully orchestrating the upgrade steps for control plane components and worker nodes, ensuring no impact on your running workloads.

What are the advantages and disadvantages?
Advantages of Kops:

- Automation:
    - Kops automates the creation and management of Kubernetes clusters, reducing the manual effort and potential for human error.
- Upgrade Management:
    - Kops simplifies the process of upgrading your Kubernetes cluster, ensuring minimal disruption and downtime.

Disadvantages of Kops:

- Complexity:
    - Setting up and managing Kubernetes clusters can be complex, and Kops is no exception. It requires a good understanding of Kubernetes concepts.
- Limited Cloud Provider Support:
    - While Kops is designed for AWS, it may not have the same level of support for other cloud providers. Current supported cloud providers are AWS and GCP, other providers are in beta or alpha (not stable).

Kubspray

What is Kubespray (Key Feature)?
Kubespray is an open-source project that provides a flexible and customizable deployment solution for Kubernetes clusters. Its use of Ansible playbooks, which allow you to define and configure your desired cluster state, separates it from the others. Kubespray supports deploying Kubernetes clusters across various cloud providers, including AWS, Azure, Google Cloud, as well as on bare metal servers. This flexibility makes it a versatile tool for managing Kubernetes deployments in different environments.

How does Kubespray handle upgrades?
Kubespray supports a rolling update strategy. This means that Kubespray orchestrates the upgrade process by updating each component of the cluster incrementally, one node at a time. By following this approach, Kubespray ensures that the cluster remains available during the upgrade, minimizing the impact on running workloads. The rolling update mechanism allows for a smooth transition to the new Kubernetes version while maintaining the availability of your applications.

What are the advantages and disadvantages?
Advantages of Kubespray:

- Customizability:
    - Kubespray provides a high level of customization, allowing you to change your Kubernetes cluster configuration to meet your specific requirements.
- Flexibility:
    - Kubespray is cloud-agnostic and can be used to deploy Kubernetes clusters across various cloud providers and operating systems.
- Extensibility:
    - Kubespray is built on Ansible, a powerful automation tool. This makes it easy to extend and modify the deployment process by leveraging Ansible's vast ecosystem of modules and playbooks.

Disadvantages of Kubespray:

- Complexity:
    - Kubespray flexibility and customizability come at the cost of complexity. It requires a good understanding of Ansible and Kubernetes concepts to effectively use and modify the playbooks.
- Limited support and documentation:
    - As an open-source project, Kubespray requires maintenance and periodic updates to align with the latest versions of Kubernetes and Ansible. This means you need to stay informed about updates and security patches to ensure the stability and security of your cluster.
    - ∂The project has an active community and user base which can be helpful in some cases, but it lacks the extensive support that other commercial tools might offer. The documentation is also not easy to navigate hence requiring a good amount of time put into understanding the tool. This can make it challenging to find timely and accurate solutions for specific issues or scenarios.

How to deal with potential data inconsistency issues/loss during an upgrade?

Upgrading a Kubernetes cluster can be a complex process, and data consistency issues or data loss can be a potential risk during this process. The following are two steps that can be taken to deal with potential data inconsistency issues/loss during a Kubernetes cluster upgrade:

- **Backups**

  Before upgrading the Kubernetes cluster, it is important to create a backup of all important data. This will ensure that in case of any data inconsistency issues or data loss during the upgrade, the data can easily be recovered.

- **Understand upgrade procedures**

  Learn the procedures recommended by the Kubernetes tool you are using. The documentation or guidelines provided by the tool will often include instructions on how to safely perform the upgrade process without causing data inconsistencies.

- **Dry run**

  Before performing the actual upgrade, it is best to perform a dry run to simulate the upgrade process and identify any potential issues or errors which can then be corrected.

How to implement rollback in case the upgrade fails?

Rollback strategies are important when performing Kubernetes cluster upgrades. Depending on the upgrade strategy chosen for the solution, there are rollback procedures that can be implemented for each of them. The following are rollback procedures that can be implemented when using the rolling update or the blue-green strategy.

The rolling update strategy involves gradually replacing the existing nodes with the updated versions. In case of a failed upgrade, the rolling update strategy offers rollback capabilities. Here is how it can work:

1. Version Monitoring:
    - During the upgrade process, monitoring tools can be used to track the status and health of each node being updated. If any failure or abnormal behavior is detected, the upgrade process can be stopped.
2. Pause and Rollback:
    - In the event of a failure, the rolling update strategy allows administrators to pause the upgrade and rollback to the previous version. This can be achieved by reverting the changes made to the updated node and resuming the deployment with the previous version.

The blue-green deployment strategy involves two identical environments, referred to as the "blue" and "green" environments. The blue environment represents the production environment, while the green environment is prepared for the upgrade. Here is how the blue-green strategy can manage a rollback in case of failure:

1. Canary Testing:
    - Before performing the actual upgrade, the green environment serves as a testing ground to make sure the new version's compatibility and functionality. By routing a small portion of traffic to the green environment, administrators can monitor and evaluate the performance and stability of the new version.
2. Switching Traffic:
    - During the upgrade, if issues or failures are detected in the green environment, administrators can simply route the traffic back to the blue environment, which is already running the stable and previous version. This redirection of traffic effectively rolls back the deployment without impacting end-users or causing significant downtime.

How to make the solution cloud-agnostic?

What is cloud-agnostic?
Cloud-agnostic is the ability of a solution to be independent of any specific cloud provider. A cloud-agnostic approach allows the Kubernetes cluster to be deployed and managed consistently across multiple cloud environments, such as AWS, Azure, Google Cloud, or on-premises infrastructure. It ensures that the solution can be migrated or run on different cloud platforms without significant modifications.

What are the challenges and limitations of achieving cloud-agnosticism?

Trying to achieve cloud-agnostic for the solution, there are a lot of challenges to take to consideration:

- Provider-specific features and services:
    - Each cloud provider offers unique features, services, and APIs. Achieving cloud-agnosticism may require avoiding or finding alternatives to provider-specific functionalities, which could limit the utilization of certain cloud capabilities.
- Configuration management complexity:
    - Managing configurations and dependencies for multiple cloud providers can introduce complexity. It may involve creating layers, using templating tools, or using cloud-agnostic configuration management tools to handle variations in configurations between providers.
- Tooling and platform compatibility:
    - Some tools, frameworks, or platforms may be only integrated with specific cloud providers, limiting their usability in a cloud-agnostic context. Finding or developing tools that are compatible and functional across multiple cloud environments can be time-consuming and resource intensive.

Conclusion

How to upgrade a Kubernetes cluster with no service disruption?

Based on all previous research questions we have come to the conclusion that Kubespray would be the ideal tool for the Kubernetes upgrade. This decision was made based on the requirements set by the stakeholders and our extensive research. Kubespray offers a range of features that make it the perfect choice for Kubernetes upgrade. One of its key advantages is its versatility in supporting various cloud providers and on-premises environments. With support for GCP, AWS, Azure, and on-premises deployments, Kubespray ensures seamless integration with different infrastructure setups, allowing for the cloud agnostic nature of this project.

In addition to its cloud flexibility, Kubespray provides a comprehensive set of tools and functionalities specifically designed for cluster management and upgrade processes. Since Kubespray is essentially a collection of Ansible playbooks and Terraform scripts, it also leaves room for potential customization. This of course comes with increased complexity.

Kubespray employes the rolling upgrade strategy which is ideal for keeping the application available while also minimizing costs.