ArgoCD Configs - Prune Propogation Policy

Prune Propogation Policy

The Prune Propagation Policy in ArgoCD defines how deletions of Kubernetes resources are handled, particularly when dealing with parent-child resource relationships. Pruning is the process of deleting resources that are no longer defined in the Git repository but still exist in the Kubernetes cluster.

ArgoCD supports three prune propagation policies that control how resource deletions (pruning) are handled, especially when dealing with resources that have parent-child relationships. Let's go through all three prune propagation policies:

1. Foreground

Description: Waits for all child resources to be deleted before deleting the parent resource.

How It Works:

- When you delete a parent resource, Kubernetes ensures that all child resources are deleted first.
- Only after all child resources are confirmed to be deleted, the parent resource itself is deleted.

Use Case:

- Ensures that the entire resource hierarchy is cleaned up in a consistent and orderly manner.
- Useful for complex resources with dependencies to avoid leaving orphaned child resources.

Example:

```
1. Deployment (Parent)
2.  |
3.  |-- ReplicaSet (Child)
4.       |
5.       |-- Pods (Child)
```

- **Foreground**: Pods are deleted first, then the ReplicaSet, and finally the Deployment.

Analogy: Think of cleaning up a project. Before you can archive the main project file, you must ensure that all related sub-tasks and documents are completed and removed.

2. Background

Description: Deletes the parent resource immediately, and the garbage collector then deletes the child resources.

How It Works:

- When you delete a parent resource, Kubernetes marks the parent for deletion and removes it immediately.
- Child resources are subsequently deleted by the Kubernetes garbage collector in the background.

Use Case:

- Faster deletion of the parent resource, but the cleanup of child resources happens asynchronously.
- Suitable for scenarios where you don't need immediate and synchronous cleanup of all child resources.

Example:

1. Deployment (Parent)
2. |
3. |-- ReplicaSet (Child)
4. |
5. |-- Pods (Child)

- **Background**: Deployment is deleted immediately, and the garbage collector takes care of deleting the ReplicaSet and Pods in the background.

Analogy: Think of deleting a project file and letting an automated script run in the background to clean up all related sub-tasks and documents.

3. Orphan

Description: Deletes the parent resource but leaves the child resources untouched.

How It Works:

- When you delete a parent resource, Kubernetes deletes the parent but does not delete the child resources.
- Child resources are orphaned and remain in the cluster.

Use Case:

- Useful when you want to retain child resources for further analysis or reuse.
- Suitable for scenarios where child resources need to be preserved even after the parent resource is deleted.

Example:

1. Deployment (Parent)
2. |
3. |-- ReplicaSet (Child)
4. |
5. |-- Pods (Child)

- **Orphan**: Deployment is deleted, but the ReplicaSet and Pods remain in the cluster.

Analogy: Think of deleting a main project file but keeping all the sub-tasks and documents for future reference or repurposing.

Summary of All Three Policies

1. Prune Propagation Policies:
2. - Foreground:
3.   - Parent deleted after all children are deleted.
4.   - Synchronous and orderly cleanup.
5.   - Suitable for complex dependencies.
6.
7. - Background:
8.   - Parent deleted immediately.
9.   - Children deleted asynchronously by garbage collector.
10.   - Faster parent deletion but less immediate cleanup.

11.
12. - Orphan:
13. - Parent deleted but children remain.
14. - Retains child resources for further use or analysis.
15. - Useful when child resources need to be preserved.

Use Case Examples

- **Foreground**: Use when you have complex applications with dependencies, and you want to ensure a clean and orderly deletion process.
- **Background**: Use when you need to delete parent resources quickly and can tolerate asynchronous cleanup of child resources.
- **Orphan**: Use when you want to retain child resources even after the parent resource is deleted, for example, to analyze their state or repurpose them.

By understanding these policies, you can choose the appropriate prune propagation strategy based on your specific needs and resource relationships in your Kubernetes cluster.