ArgoCD

syncPolicy:

syncOptions:

- ApplyOutOfSyncOnly=true

In ArgoCD, the **syncPolicy** section is used to define policies related to the synchronization of applications. Specifically, the **syncOptions** field allows you to configure various options for how synchronization is performed. In your provided example:

Here's an explanation of what this means:

* **syncPolicy**: This is a section in the ArgoCD Application manifest where you can specify policies related to how the application should be synchronized.
* **syncOptions**: This is a list of options that control how synchronization is performed for the application.
* **ApplyOutOfSyncOnly=true**: This option indicates that during synchronization, ArgoCD should only apply changes if the application is out of sync. In other words, it will skip applying changes if the current state of the application matches the desired state. This can be useful to avoid unnecessary updates when the application is already in the correct state.

For example, if you have made manual changes to the deployed resources (out of sync with the Git repository or desired state), ArgoCD will apply changes during synchronization. If there are no differences between the current state and the desired state, no changes will be applied.

This configuration helps in making the synchronization process more efficient by reducing unnecessary updates and potentially avoiding unintended changes in a live environment.

Top of Form

Bottom of Form

spec:

syncPolicy:

automated:

prune: true

allowEmpty: true

in summary, ApplyOutOfSyncOnly=true focuses on manual intervention, ensuring that changes are applied only when a discrepancy is detected. Automated sync policies, on the other hand, automate the synchronization process and apply changes based on predefined schedules or events, regardless of the current synchronization state. The choice between them depends on your specific deployment and release management requirements.

* **syncPolicy**: This section is used to define synchronization policies for the application.
* **automated**: It specifies an automated synchronization policy. In other words, it configures how automated synchronization should behave.
* **prune: true**: When **prune** is set to **true**, ArgoCD will automatically delete (prune) resources that exist in the cluster but are not defined in the Git repository. This ensures that the actual state in the cluster matches the desired state defined in your version-controlled Git repository.
* **allowEmpty: true**: When **allowEmpty** is set to **true**, ArgoCD will allow the synchronization of an application even if the Git repository contains no manifests (empty repository). This can be useful in scenarios where your application may not have any manifests at a particular point in time, and you still want ArgoCD to proceed with synchronization without errors.

SHA:

* In ArgoCD, the "SHA" typically refers to the Git commit SHA (Secure Hash Algorithm) checksum, which is a unique identifier for a specific commit in a Git repository. The commit SHA is a long hexadecimal string that uniquely identifies the state of the repository at a given point in time.
* ArgoCD uses the commit SHA to track the version of the Git repository that should be deployed. When you define an application in ArgoCD, you often specify a Git repository URL and a target commit SHA to determine which version of the repository should be deployed as the desired state.

spec:

source:

repoURL: https://github.com/example/repo.git

targetRevision: abcdef1234567890

Here, targetRevision represents the specific commit SHA that ArgoCD should use for synchronization. ArgoCD will deploy the application based on the resources defined in the repository at that particular commit.

Using commit SHAs in ArgoCD ensures reproducibility and version control for your deployments, allowing you to roll back or forward to specific states of your application defined in the Git repository.

Bottom of Form

Automated sync:

* Argocd performs automated sync every 3 minutes.
* Automated sync will only performed if the app is out of sync
* Automatic sync will not reattempt if the previous sync attempt same SHA failed.

\*\* when I test with just

spec:

syncPolicy:

automated: {}

>> it syncs automatically but when I deleted the deployment manifest the resources have not deleted.

spec:

syncPolicy:

automated:

prune: true

>> the resources are deleted

spec:

syncPolicy:

automated:

prune: true

allowEmpty: true

>> the resources are deleted

spec:

syncPolicy:

automated:

prune: true

allowEmpty: false

>> the resources are not deleted but the ui shows (x) sign

Automatic Self-Healing[¶](https://argo-cd.readthedocs.io/en/stable/user-guide/auto_sync/" \l "automatic-self-healing" \o "Permanent link)

By default, changes that are made to the live cluster will not trigger automated sync.

To enable automatic sync when the live cluster's state deviates from the state defined in Git

spec:

syncPolicy:

automated:

selfHeal: true

I tested this behavior and noticed that when I scale in the deploy to 1 replicas instead if of 3; it reverted to the state in git which is 3.

SyncOption:

Argo CD allows users to customize some aspects of how it syncs the desired state in the target cluster. Some Sync Options can defined as annotations in a specific resource. Most of the Sync Options are configured in the Application resource spec.syncPolicy.syncOptions attribute. Multiple Sync Options which are configured with the argocd.argoproj.io/sync-options annotation can be concatenated with a , in the annotation value; white spaces will be trimmed.

metadata:

annotations:

argocd.argoproj.io/sync-options: Prune=false

we can put it in the deployment for example and if the resource is deleted from git, it will never be deleted.

## Replace Resource Instead Of Applying Changes

By default, Argo CD executes kubectl apply operation to apply the configuration stored in Git. In some cases kubectl apply is not suitable. For example, resource spec might be too big and won't fit into kubectl.kubernetes.io/last-applied-configurationannotation that is added by kubectl apply. In such cases you might use Replace=truesync option:

apiVersion: argoproj.io/v1alpha1

kind: Application

spec:

syncPolicy:

syncOptions:

- Replace=true

If the Replace=true sync option is set the Argo CD will use kubectl replace or kubectl create command to apply changes.

This can also be configured at individual resource level.

metadata:

annotations:

argocd.argoproj.io/sync-options: Replace=true

Server-Side Apply is a feature in Kubernetes that enables declarative configuration for resources directly on the server, without the need for a client-side tool to compute and send the differences. It is designed to simplify the process of managing and applying configurations to Kubernetes resources.

In the traditional imperative approach, tools like kubectl apply perform a client-side apply. The client computes the differences between the desired state (as specified in the configuration files) and the current state in the cluster, and then sends the changes to the API server.

Server-Side Apply, on the other hand, shifts some of this computation to the server side. When you use Server-Side Apply, you send the entire desired state of the resource to the API server. The server then performs the computation to determine the changes required to move from the current state to the desired state. This helps in reducing the complexity and potential issues related to managing these changes on the client side.

## Create Namespace[¶](https://argo-cd.readthedocs.io/en/stable/user-guide/sync-options/#create-namespace)

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

namespace: argocd

spec:

destination:

server: https://kubernetes.default.svc

namespace: some-namespace

syncPolicy:

syncOptions:

- CreateNamespace=true

The example above shows how an Argo CD Application can be configured so it will create the namespace specified in spec.destination.namespace if it doesn't exist already. Without this either declared in the Application manifest or passed in the CLI via --sync-option CreateNamespace=true, the Application will fail to sync if the namespace doesn't exist.

Note that the namespace to be created must be informed in the spec.destination.namespace field of the Application resource. The metadata.namespacefield in the Application's child manifests must match this value, or can be omitted, so resources are created in the proper destination.