2.0 Rollback

Overview

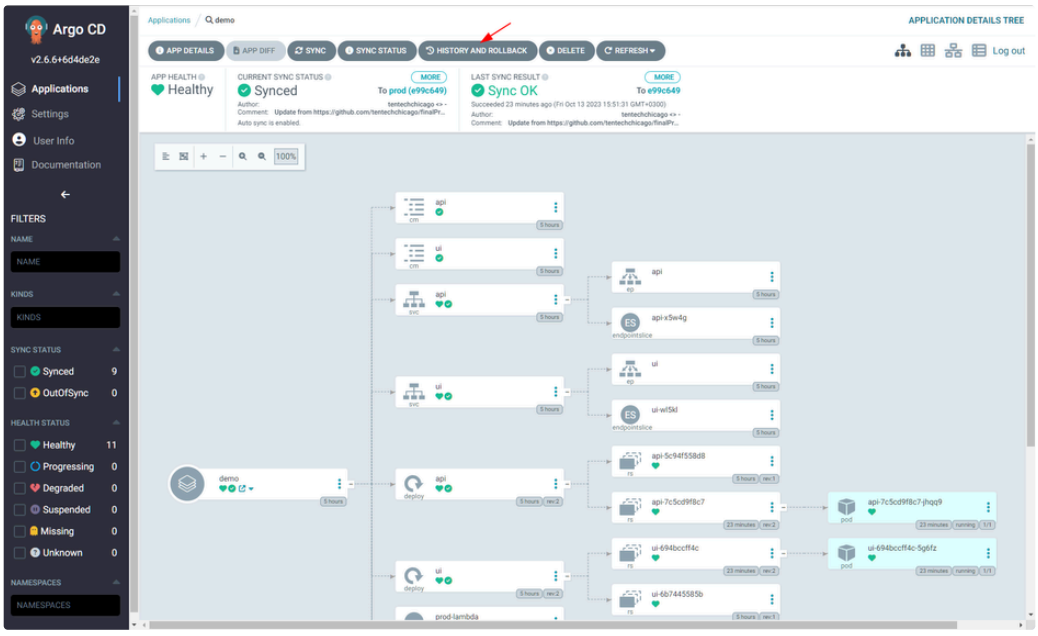
There is often a need to revert to a previous version of an application in order to make fixes or roll back unsuccessful changes. For this purpose, ArgoCD includes a special feature responsible for versioning deployments. All these revisions are associated with commits in our GitOps repository (CD), where all components are declaratively described along with pointers to the required containers. This allows us to easily roll back changes. However, there are nuances related to auto-synchronization that need to be taken into account. It involves a manual component, where you need to disable and enable synchronization during the period when fixes need to be executed.

To revert to the previous version, you need to perform a series of operations:

1. disable auto-sync
2. perform rollback
3. revert/fix the offending commit in git
4. reenable auto-sync

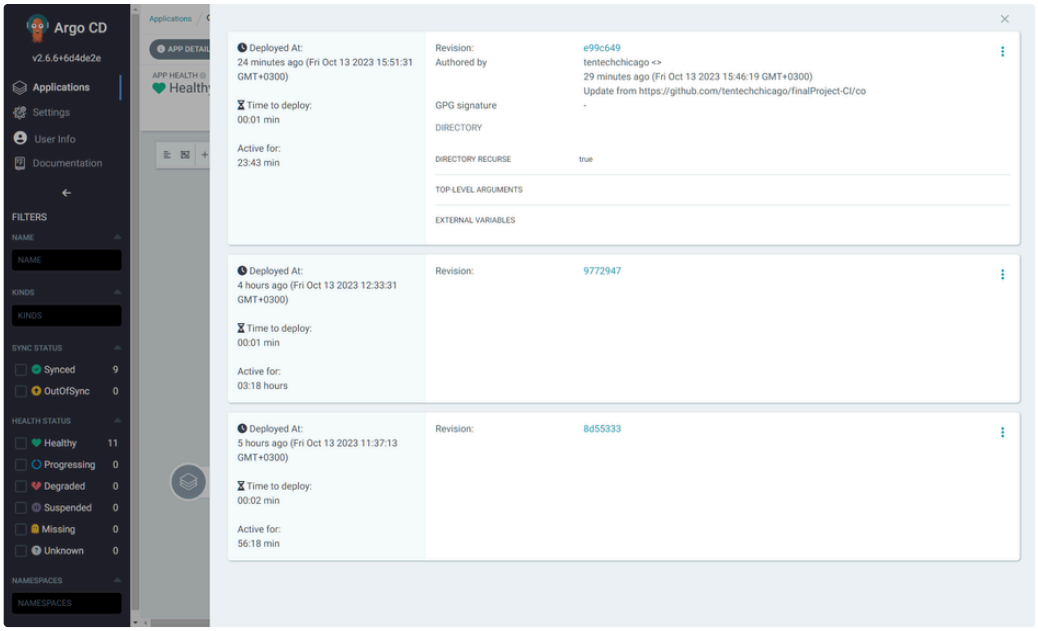
Procedure

To roll back to the previous version in ArgoCD, there is a special feature that allows you to return to the previous state of the application deployment. To do this, you need to select "HISTORY AND ROLLBACK" in the menu.



In this section, all deployments that have occurred during the lifetime of the ArgoCD instance will be displayed.

Each of them is tied to a specific commit in the CD repository.



Let's check using the example of the frontend version of the application, which corresponds to commit 9772947: "version": "0.1.0"

We artificially increased the version of the latest commit e99c649 to 0.1.1 and cosider this version as broken. We need to rollback to previous working and stable version.

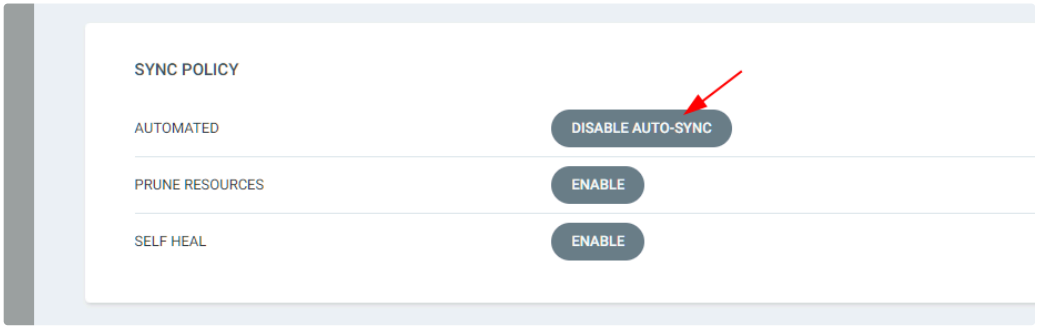
If you go inside the container and look at the `package.json`, you will see this.

```
kubectl -n application exec ui-694bccff4c-5g6fz -- cat package.json | grep -i \"version\"
```

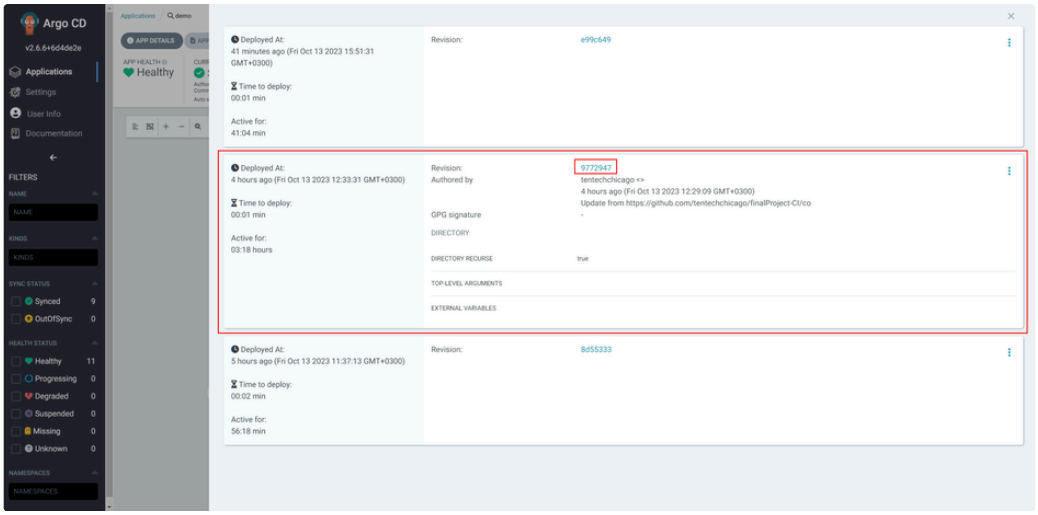
Output:

```
"version": "0.1.1",
```

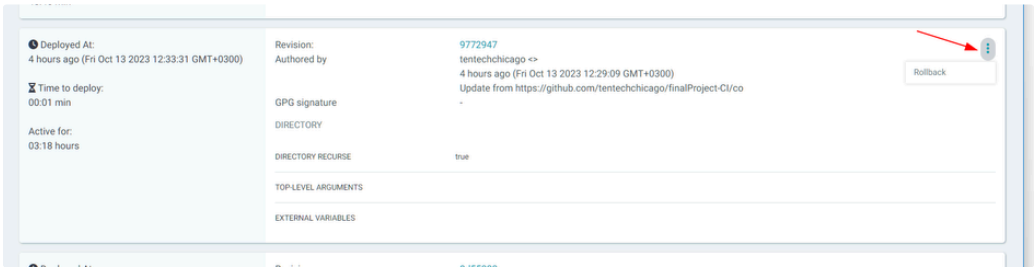
Disabling auto-sync is necessary to prevent automatic redeployment based on a timer. To do this, you need to click on the corresponding button in the project settings.



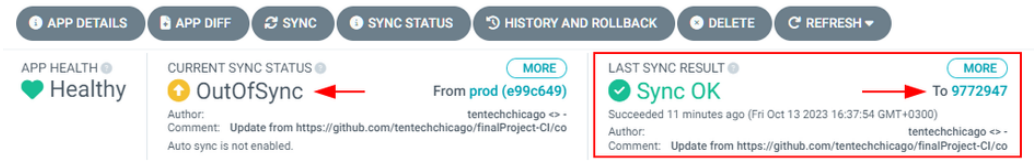
Select the desired commit from the list, choose the previous one, which corresponds to the stable version 0.1.0, and check its functionality.



Then select Rollback from the menu:



After completing all the operations, ArgoCD will display a message indicating that the last sync was performed with the desired commit `9772947`. It will also show a message stating that the current status is not synchronized with the latest commit: OutOfSync.



Now, we check the functionality of our application that corresponds to version 0.1.0, making sure that the current file system of the new container matches this version:

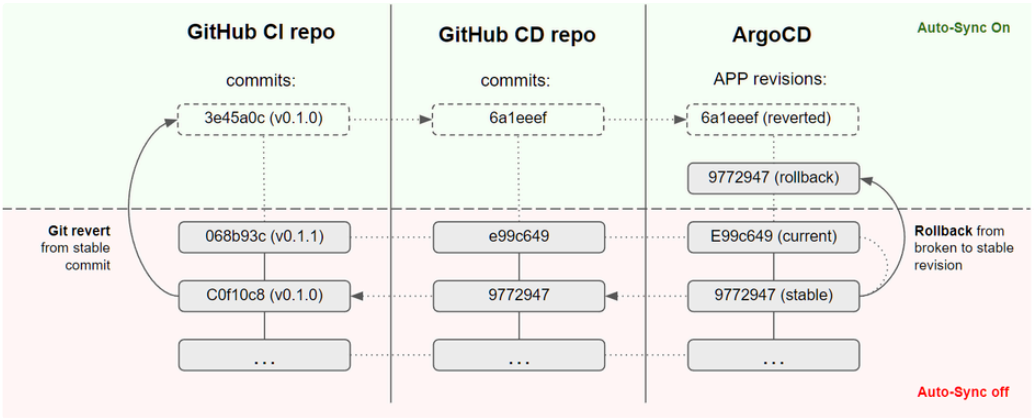
```
kubectl -n application exec ui-6b7445585b-8bf47 -- cat package.json | grep -i \"version\"
```

Output:

```
"version": "0.1.0",
```

Next, it is necessary to fix the last commit or perform a revert. We will perform a revert of changes because this is a test demonstration of version rollback.

It's worth noting that it is correct to perform the revert directly in the CI repository where the source code is stored, rather than in the CD repository, in order to preserve the links between CI and CD.



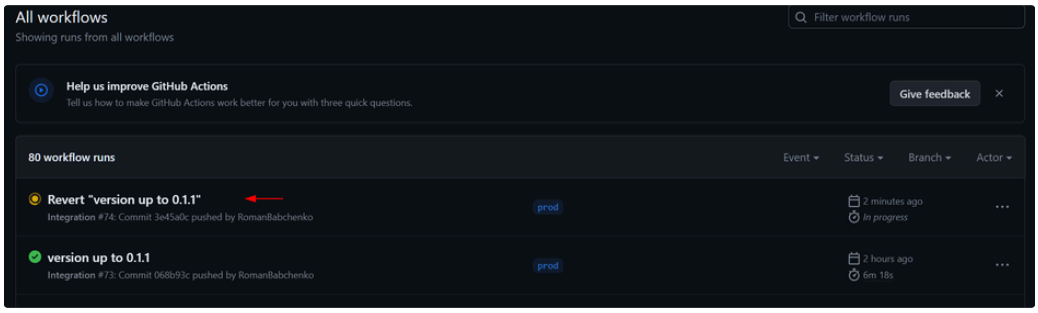
Relations between commits and deployment revisions

Perform `git revert` on the CI repo:

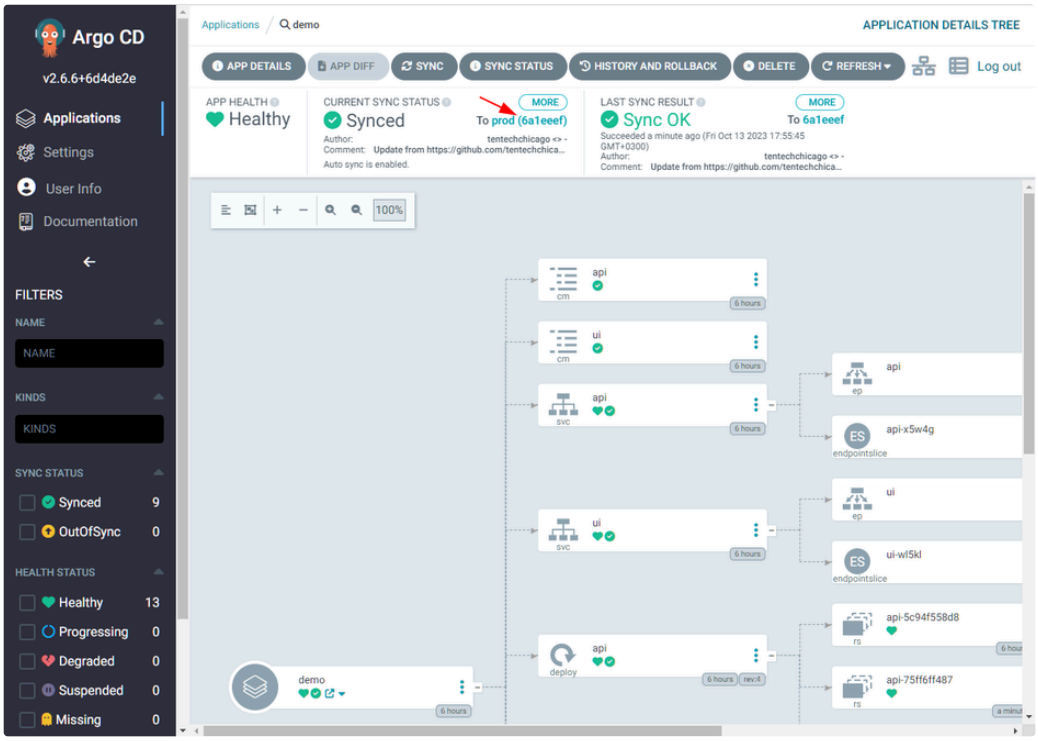
```
git revert c0f10c8218b2ae2ae8dc2f73631c2c97991bc041..068b93c19def7e98a148a82ff6e9fe8e16cc2462To
```

That creates new commit with reverted changes to previous commit

Push and wait for CI process is done:



Turn on **auto-sync** again and initiate synchronization and back to normal state of ArgoCD.



ArgoCD now **auto synchronized** with the commit responsible for the revert to 0.1.0

If you check the package.json on the UI pod again, it should now be at version 0.1.0.

```
kubectl -n application exec ui-66dc666db9-dpd4r -- cat package.json | grep -i \"version\"
```

Output:

```
"version": "0.1.0",
```

Conclusion

This is how rollback procedures are performed on ArgoCD. To some extent, this process is manual, but nonetheless, quickly reverting to any desired commit and deploying the application accordingly is straightforward. It's done swiftly, and you can easily return a malfunctioning application to its previous stable state while making fixes.