Container update process

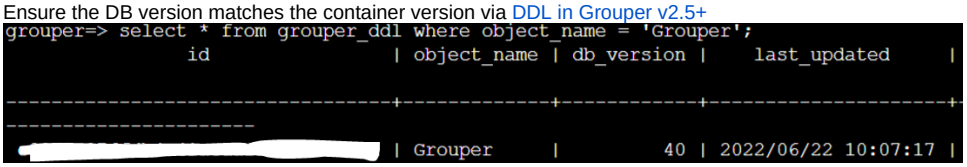
The following describes our container upgrade process. We developed this process, including the rollback process, in order to ensure that during a container upgrade nothing breaks along the way. Our container upgrade process goes through three phases:

1. Plan
2. Do
3. Review

Plan

This phase of the container upgrade process includes knowing what we need to do during the upgrade, checking current versions, and ensures everything works prior to the upgrade.

1. Review the release notes https://spaces.at.internet2.edu/display/Grouper/v2.6+Release+Notes and understand the upgrade process https://spaces.at.internet2.edu/display/Grouper/v2.6+Upgrade+Instructions+from+v2.6
2. Check the current DDL version
    a. SSH into the container
    b. Connect to the backend DB via the container; if need be install postgresql
    c. Check the DDL version via this query: select * from grouper_ddl where object_name = 'Grouper';
    d. Ensure the DB version matches the container version via DDL in Grouper v2.5+

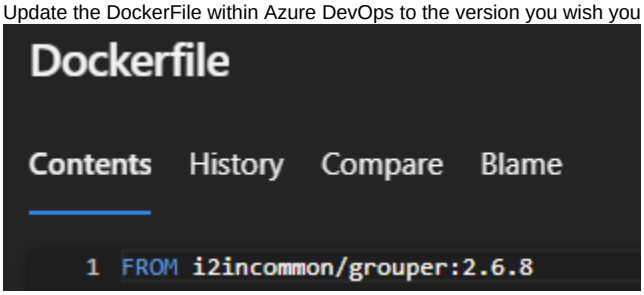


3. Start GSH (we want to make sure there are no errors before we upgrade)
4. Backup all of the configuration and post to Azure DevOps file repo
5. Check that all daemons are green before upgrading
6. Test all external systems

Do

This phase of the upgrade process actually upgrades the container image.

1. Update the DockerFile within Azure DevOps to the version you wish you run



2. Create a new Azure DevOps container release (see below for details about how we release containers to the Azure container registry)
3. Restart the Grouper Daemon container

Review

This phase of the upgrade process ensures what we did worked and that nothing is broken.

1. Watch the Grouper Daemon Container image log stream to ensure it is pulling the updated container
2. Perform any upgrade specific instructions as documented
3. SSH into the container and run GSH to ensure there are no upgrades post error
4. Check the DDL version as described above in the Plan phase to ensure the DB was indeed upgraded
5. Assuming no errors, proceed with restarting the UI container
6. Ensure that within the UI, under Configure the correct version of Grouper is shown
7. Check daemon jobs to ensure they are all green (note it might take a while for them to turn green)
8. Run a simple loader
9. Run a simple provisioner
10. Test all external systems
11. Test anything specific to the release
12. Backup all of the configuration files to Azure DevOps

How to revert

Our revert process basically rolls back the container image one upgraded to and restores the DB prior to the upgrade.

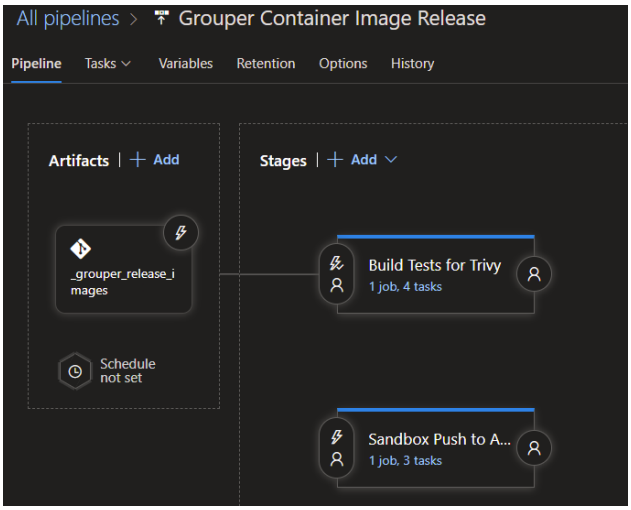
1. Start the Azure Postgres database restore (pick a date/ time prior to the upgrade); this will take a while.
2. Revert the change made within the DockerFile to the version you would like to go back to
3. Stop the Grouper daemon and UI containers
4. Create a new Grouper container image release to push the container to the Azure Container registry
5. Within the Azure app service daemon and UI containers, go to configuration and change the GROUPER_DATABASE_URL to the restored DB server name.
6. Restart the Grouper daemon and UI container
7. Verify the restore worked by checking the Grouper container version via the UI and check the backend DB version using the process described above

Azure DevOps release pipeline for container images

The following describes how we update our container images using Azure DevOps release pipelines.

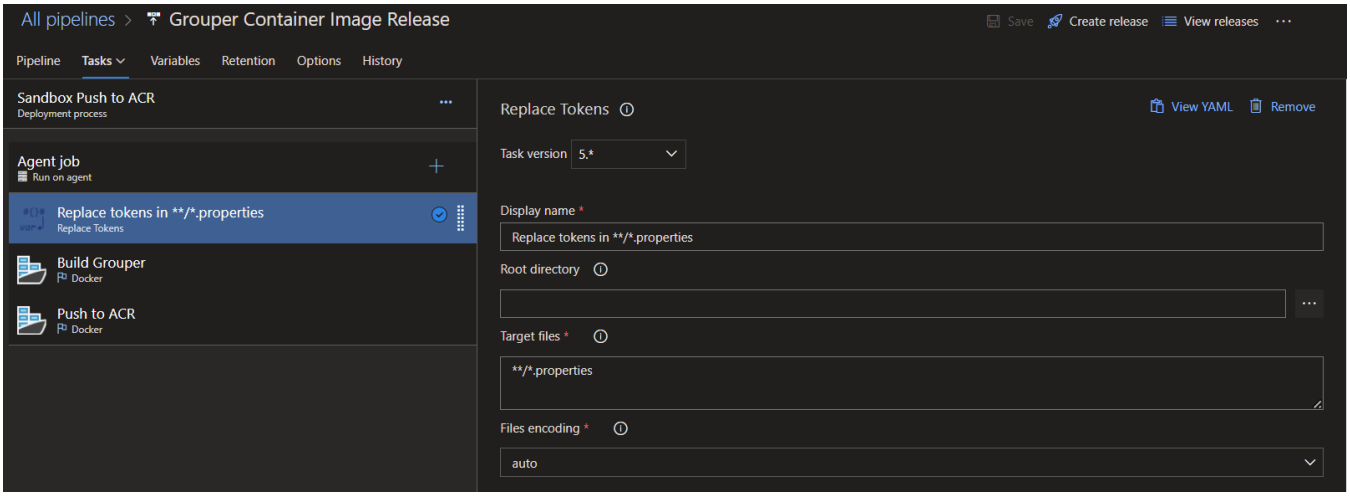
Our Grouper container files are stored within an Azure DevOps repo; i.e. we have a Dockerfile and some overlay files.

When one starts a new release pipeline, a Grouper container is built using the Dockerfile and overlay files. This immediately gets scanned via Trivy for security vulnerabilities.



One can review the results of the Trivy scan and then one has to manually deploy to one or many environments.

For a given stage that actually pushes to an Azure container registry, we have the following tasks:



The first task performs token replacement to swap out tokens within the overlay files that are environment specific.

Here is an example of some of our variables.

grouper.is.ui.basicAuthn	false
grouperPasswordConfigOverride_UI_GrouperSystem_pass	********
terraform_version	1.1.5

Next within the stage, we build the Grouper container image and push it to the ACR. We connect to portal.azure.com via a service principal connection.