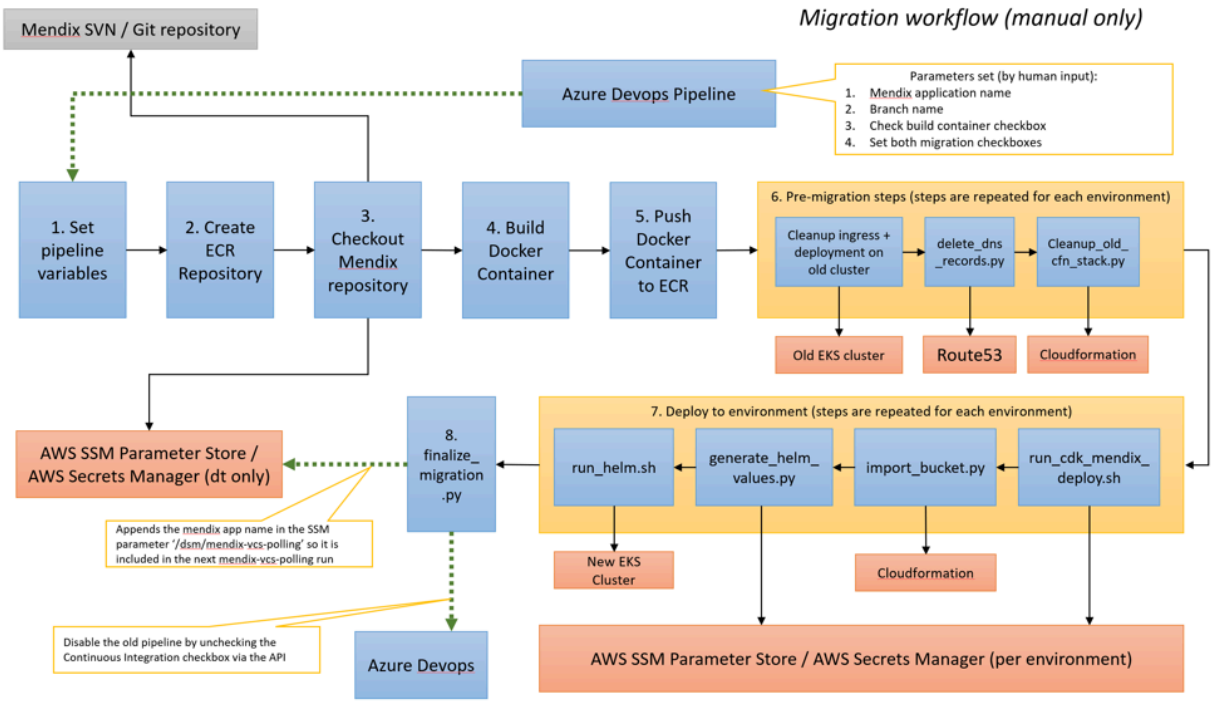
Migration pipeline run

This page explains the migration pipeline run. It uses pretty much all of the steps also found in the normal pipeline run, so please read that page first to understand that process: [Normal pipeline run - MAAS Team Space - Confluence (atlassian.net)](). The migration pipeline is only meant to be running when migrating from the `eksctl` based Kubernetes cluster, including the needed Cloudformation state, to the CDK created Kubernetes cluster. Once this has completed, this pipeline serves no more purpose and you must use the normal mode instead.

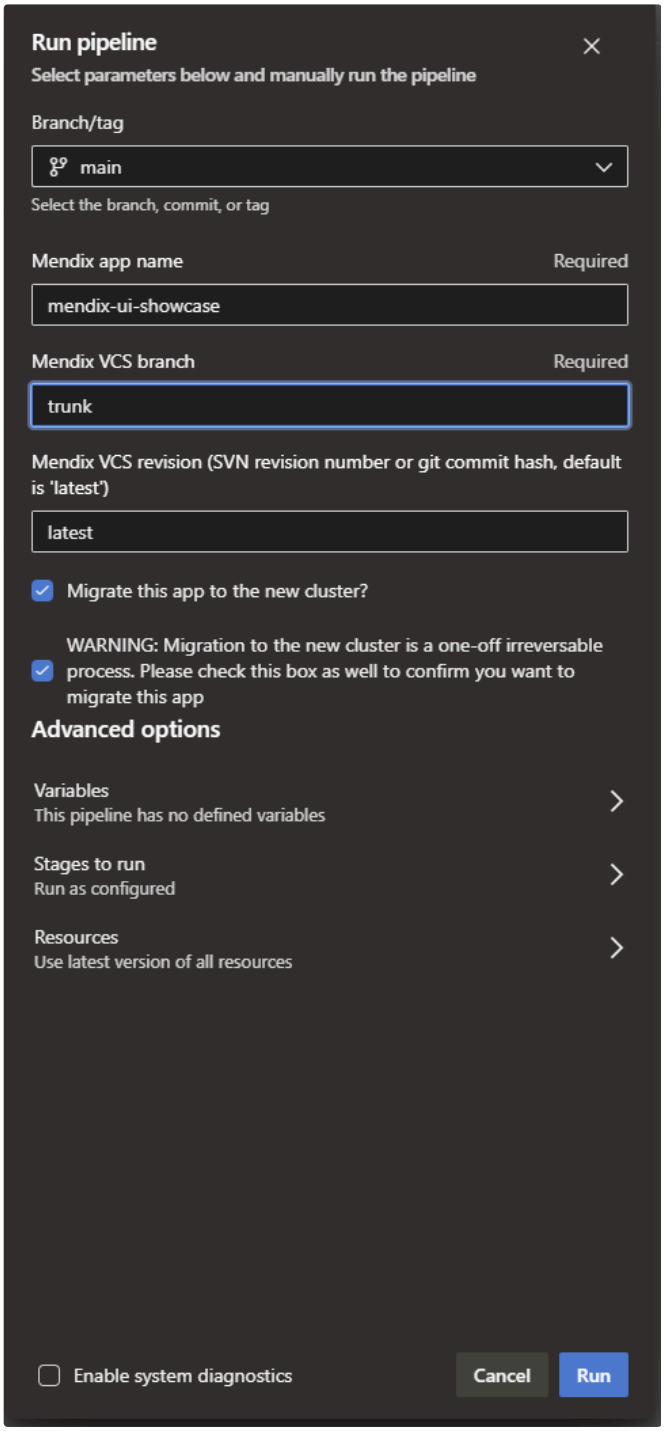


Phase 1: Starting pipeline

This pipeline run is the same as starting the normal pipeline except for setting two more checkboxes: to enable migration mode and one more to confirm it (to prevent you from accidently doing it). You can also optionally disable the building of the container if you don't need to do it, make sure you set the image tag to the right value then, of course.

⚠ Once you start the migration for an environment, rolling back is not a trivial thing to do. Only start migration when you are sure you can do it. It is expected to do a fail-forward approach instead in case of problems, and be prepared to make changes to the old

pipeline to make it possible to do the rollback (most notably you have to use the no-bucket cloudformation template to avoid recreating the s3 bucket)



In this screen we select both checkboxes in the build pipeline to enable migration mode. This is propagated to the release pipeline

Phase 2: Build and push

Refer to the Normal pipeline run - MAAS Team Space - Confluence (atlassian.net) for more info on how this works.

Phase 3: Pre-migration steps

In this phase the migration itself actually starts. Basically the migration is a two-step rocket as we first need to cleanup and decommission old resources on the old cluster before starting to use the new ones. In this phase we cleanup old resources and effectively bring down the application. The following steps are one in this phase:

1. Cleanup of resources on the old cluster, which is achieved by:
    a. Patching the ingress object to remove the finalizer. This to avoid waiting for reconciliation that fails due to incorrect parameters of the ingress objects. Without this patching the ingress will wait forever
    b. Delete the ingress object
    c. Delete the leader and follower Kubernetes deployment
2. Deletion of DNS records. This is done because external-dns will not take ownership of existing records of the new cluster (the owner does not match). This is actually a good thing because it prevents bouncing of DNS records between the old and the new cluster. So we force a deletion of the DNS records after deleting the ingress object so that the new ingress object on the new cluster can just create a fresh set of DNS records.
3. Cleanup old cloudformation stack. This is where the tricky bit comes in:
    a. It deletes the old cloudformation stack and waits until the status is in DELETE_FAILED. This is because the S3 bucket is not empty and cannot be deleted.
    b. This DELETE_FAILED state is intentional and we do another delete with retaining the S3 bucket. At this stage the old CF resources are gone except for the bucket (the bucket is now unmanaged by Cloudformation)
    c. To prevent buckets being deleted which happen to empty, a migrate.txt file is uploaded to the bucket before calling Cloudformation. This way we can assume the CF stack will always get in a DELETE_FAILED state.

At this point in time we are in a state that everything is cleaned up that has Cloudformation state and uses AWS resources directly (things like ACM certificates are not handled by Cloudformation and will remain. A different problem for another time).

Phase 4: Deployment and importing of the S3 bucket

In this phase we follow the majority of the normal pipeline run (see link at Phase 2) with a few caveats. In here we only highlight the differences over the normal run:

1. Execute `run_cdk_mendix_deploy.sh` in migration mode. This is a special mode that creates a Cloudformation stack with everything a normal pipeline run also creates, but **without** the S3 bucket resource. Instead, an S3 Cloudformation template is synthesized but **not used**. This template must be used in the following step. The reason why this process is so clunky because this is how Cloudformation imports work; you need to provide a Cloudformation template to import into another template. This template will be modified in step 2 to match the existing S3 bucket that was previously part of the old Cloudformation template.
    a. A special note here: It also checks if the bucket exists. If no bucket exists this script cannot run in migration mode, and instead falls back to normal mode to create a new bucket as if it were a new application.
2. Execute `import_bucket.py`. This takes the CF template, specifically for an S3 bucket, generated in the previous step and modifies it before passing it onto Cloudformation to run the IMPORT_RESOURCE action within Cloudformation. This effectively merges this temporary cloudformation template into the newly created Cloudformation stack that has been created by the CDK. As this is an asynchronous call the pipeline will continue will importing the bucket in the background.
    a. This script also checks if a bucket exists and exits if a bucket does not exist. This makes it idempotent for applications that have not been deployed on all environments.
3. Execute the remainder of the pipeline as if it were a normal pipeline.

At this stage the migration for this environment is complete. The application runs on the new cluster and the Bucket belongs to the cloudformation stack created by the CDK. Any additional pipeline runs in normal mode will now operate correctly on the existing bucket. Of course, phase 3 and 4 must be repeated for every environment.

Phase 5: Finalize migration

When deployment is done across all 3 environments, there are two more steps to perform:

1. We disable the old Azure Devops pipeline. This is done by unchecking the continuous integration checkbox as part of the old build pipeline. This disables the polling and prevents future builds from happening in this pipeline.

2. Add the application name to the SSM parameter `/dsm/mendix-vcs-polling`. This is used by the polling script to determine which applications must be included for polling, and if a change is found, trigger the normal pipeline run.

Once this is completed you are done. The new pipeline will be used for any subsequent commits and will build and deploy onto the CDK managed cluster.