Mendix Generic CICD Pipeline & AWS Infrastructure

This page describes the working of the Generic CICD pipeline for Mendix applications. It describes the rationale, design goals & decisions, current working on a high level and will go through each possible flow in detail. We are assuming basic knowledge of the tools and technologies used, but references are made to external pages for further reading on topics should these be unfamiliar.

Background information & rationale

At this time of writing (2 October 2023), the current workflow for building and deploying the Mendix applications on AWS is primarily based around Azure Devops. Within Azure Devops, the so-called “classic” pipelines are used. They are non-versioned, non-codified (and manually created) pipelines created via the Azure Devops UI. There are two pipelines for each Mendix application; one build and one release pipeline (both are considered to be “classic”). Each pipeline is effectively a copy of one-another and there is no central place to update all pipelines in one go. There is an `aws-mendix-deployment` repository that houses both the Kubernetes manifests and the Cloudformation templates. These files are templates utilizing an Azure Devops only templating functionality and are not standard industry tools. The actual values are coming out of Azure Devops variables, which must be defined for each pipeline separately. With the `aws-mendix-deployment` repository there are copies of Kubernetes manifests and Cloudformation files as well for specific application’s needs.

Mendix applications themselves are built using the Mendix Buildpack. The concept of a buildpack comes from Cloudfoundry, which is something Mendix uses internally for their Mendix Cloud offering. With this, several issues have been identified in addition to the ones mentioned above. First of all, the buildpack is expecting to be running on Cloudfoundry and does not work on a Kubernetes environment. Mendix released an additional wrapper around this called the Docker Mendix Buildpack. This basically prepares the container environment to mimic running on Cloudfoundry. All of this uses a lot of custom Python code to make it possible; this causes us to lose direct control over whatever happens during build time. It also makes it difficult for platform / devops engineers unfamiliar with Mendix to understand

how applications are built and deployed. On top of that, Mendix does infrequent updates to these projects without any form of support.

The infrastructure itself is created via Cloudformation templates and is combined with the build & release pipeline mentioned above. Apart from earlier mentioned problems, the EKS cluster itself is created and updated via `eksctl`, a tool provided by AWS. This means the EKS cluster itself has little to no Cloudformation state and has a separate process. Whatever infrastructure is defined in Cloudformation for EKS will have significant state drift. In addition, there are some custom scripts that use boto3 directly and therefore have no state at all.

Combining the three problems mentioned above means that updating the Mendix Buildpack, or any of the core infrastructure processes is a labor intensive, time-consuming thing to do which requires significant effort. Rather than trying to patch up the existing processes, we set out to rectify a lot of problems currently faced but not really noticable outside of the team.

Design goals & decisions

In order to overcome the problems mentioned, we settled on the following decisions on defining the new pipeline, infrastructure and build & deploy process:

- One generic CI/CD pipeline for all Mendix applications. Pipeline code must live in Git and should not be manually created.
- Azure Devops variables to make place for a generic system to remove the tight coupling to Azure Devops. Other applications must be able to integrate with this.
- Use industry standard tools to streamline onboarding of new devops / platform engineers
- Remove the need for duplicating manifests / templates. All logic must live inside code.
- Reduce the pipeline logic itself as much as possible; move them to generic scripts that can operate by themselves. The pipeline should only be a coordinator as much as possible.
  - This makes it easier to troubleshoot and modify specific steps of the pipeline.
  - This also makes it easier to move to a different CICD system in the future, should that be necessary
- All infrastructure to be created via the AWS Cloud Development Kit (CDK).
  - This was chosen over Terraform for the simple reason that there is knowledge of CDK within DSM-Firmenich and none of Terraform, and Cloudformation is of course already used.
- Remove stateless scripts from pipeline and must be using CDK instead
- Remove Azure Devops templating in favor of Helm templates

- Deployments to acceptance and production must have a manual approval stage
- Replace Mendix buildpack with a Kubernetes native workflow
  - This still needs the tools the buildpack uses, like mxbuild and m2ee, but the objective is to strip away all of the unneeded Python code to give all control to us and not rely on Mendix's upstream and unsupported project.

We settled on the following tools to fulfill the above requirements:

- AWS CDK for infrastructure-as-code and Typescript as the programming language used
- Helm as the tool to do deployments to Kubernetes as well as templating of the Kubernetes deployments
- AWS SSM Parameter store for configuration variables of Mendix application and state for active processes (like mendix-vcs-polling, see paragraph below for more info).
  - This replaces the Azure Devops variables
- AWS Secrets Manager for storing passwords, tokens, license keys etc.
  - This replaces the secrets portion of the Azure Devops variables.
- The Azure Devops pipeline is a YAML-based pipeline which is considered to be "new". This comes with some limitations; please refer to the other pages to learn more on those.

Repository	URL	Purpose
maas	https://dsmplatform.visualstudio.com/aws-mendix-deployment/_git/maas	CDK code and pipeline for the Kubernetes Cluster
kubernetes-mendix-deployment	https://dsmplatform.visualstudio.com/aws-mendix-deployment/_git/kubernetes-mendix-deployment	CDK code, Helm chart, Azure pipeline plus supporting scripts for Mendix application deployment

Migration

In order to move to this new pipeline, a one-time migration needs to take place. On a high level, the migration is done in the following steps (step 1 till 8 are part of the migration

pipeline)

0. Populate the SSM parameter store with the needed parameters. At the very least the VCS branch and URL must be registered in SSM. Typically the Azure Devops variables are also stored in SSM a part of this step. Visit the SSM parameter store page for more details on this. This step is not part of the pipeline.
    a. There is a script in the *kubernetes-mendix-repository* to populate SSM with Azure Devops variables. It's called `populate_aws_with_azure_devops_vars.py` - This script is also not part of the pipeline and is standalone.
1. Build and push a new container to a new ECR repo. If the build fails the migration is aborted (at this point in time no change has been made and can be safely repeated)
2. Shut down the application on the old cluster and cleanup lingering resources
3. Delete the old Cloudformation Stack(excluding the S3 bucket)
4. Run the CDK code for this application to create new resources.
    a. This will reset the RDS database password, IAM Access Key & secret and Mendix Admin password
5. Import the S3 bucket into the newly created Cloudformation Stack as is generated by the CDK code.
6. Deploy the Mendix application via Helm
7. Repeat step 2-6 for all remaining environments
8. Finalize the migration by disabling the old pipeline and registering this Mendix application in the mendix-vcs-polling (see paragraph below for more info on this).

This implies that there will be a downtime of the application during the migration, which is estimated to be around 10-15 minutes for an environment (it also depends on how quickly DNS caches are expired since the DNS records also change). Other than the short downtime, no changes should be apparent for the end users of the migrated application.

Please refer to the Migration pipeline run - MAAS Team Space - Confluence (atlassian.net) page for more detail on how the pipeline works.

⚠ Once the migration process has started, the old pipeline will fail to work for the environment which has been migrated. For example, if you just migrated development, doing a deployment to acceptance and production on the old cluster will only be possible by directly deploying onto these environments and bypassing the development environment.

Mendix VCS polling

In the current pipeline there is a polling mechanism that is provided by Azure Devops itself. This polling mechanism checks for upstream changes on the upstream Mendix Subversion / Git repository and if there are, trigger the pipeline. This mechanism no longer exists in the YAML pipeline world (only for Azure hosted Git repositories), thus we had to mimic this ourselves. This is where mendix-vcs-polling comes in (VCS standing for Version Control System, as both git and svn are used by Mendix). It basically relies on the SSM parameter store for input, loops over a list of apps, and polls for changes. The upstream commit is compared with what it has stored (also in the SSM parameter store) and, if it's different, it triggers the pipeline by using the Azure Devops REST API with the required parameters.

At this time of writing it runs in the Kubernetes dev cluster as a cronjob. This may change in the future so please validate if this information is still accurate, especially if you are reading this in the future. Finally, This functionality does not work without the required SSM parameters. Please refer to SSM Parameter Store & Secrets Manager reference - MAAS Team Space - Confluence (atlassian.net) for a list of SSM parameters that this functionality requires.

Further reading

Please refer to subpages of this page for additional information on specific topics. For a general introduction for the tools used, please use these links as a starting point to get familiar with them. It's important to better learn these tools so you have a good understanding how the tools are used:

What is the AWS CDK? - AWS Cloud Development Kit (AWS CDK) v2 (amazon.com)

AWS CDK Intro Workshop | AWS CDK Workshop

AWS Systems Manager Parameter Store - AWS Systems Manager (amazon.com)

What is AWS Secrets Manager? - AWS Secrets Manager (amazon.com)

What is a Helm Chart? A Tutorial for Kubernetes Beginners (freecodecamp.org)

Helm | Using Helm

Azure Pipelines New User Guide - Key concepts - Azure Pipelines | Microsoft Learn (note, look specifically at YAML based pipelines, some things only apply to classic pipelines)

YAML schema reference | Microsoft Learn