Helm chart

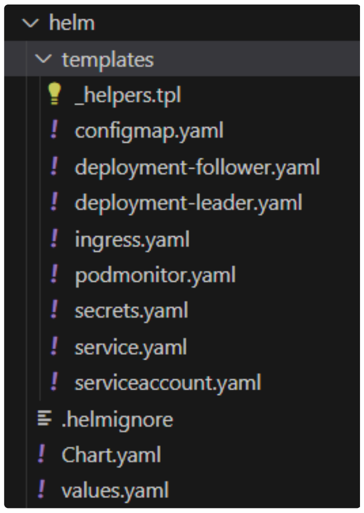
The helm chart is the primary deployment artifact used to deploy Mendix applications. It's written in Golang and uses standard Go templating style for its templating engine of YAML files. For those familiar with Python, you can consider it like Jinja except the syntax being slightly different. This page aims to explain the structure of the Helm chart itself, not how Helm itself works, nor how Kubernetes works. For the basics of learning about Helm you can read the following page to get started: https://helm.sh/docs/intro/using_helm/. It's advised you are familiar with Helm before continuing to read this page.

General structure

The helm chart follows standard conventions that Helm charts typically have, so if you are already familiar with using Helm you will find it to be familiar. Every kubernetes object type lives in a different YAML file with a fairly standard helpers file (this concept is used in pretty much all Helm charts and is a standard way of doing things). The values.yaml has been commented, and for the sake of not updating documentation on two places it's not going to be documented here but the key ones will be mentioned.

⚠ This Helm chart is written with the assumption you override the default values with specifics for each application. Just installing the Helm chart without providing valid values will be accepted by Helm but will cause a failure to start the Mendix Runtime. Use `generate_helm_values.py` found in the repository to generate a `values_generated.yaml` and use that to override the default values.

At this time of writing, the general structure looks like this:



```
helm
  templates
    _helpers.tpl
    configmap.yaml
    deployment-follower.yaml
    deployment-leader.yaml
    ingress.yaml
    podmonitor.yaml
    secrets.yaml
    service.yaml
    serviceaccount.yaml
  .helmignore
  Chart.yaml
  values.yaml
```

An overview of the Helm chart layout

The first thing that may draw your attention is that there are two deployments; one for the leader and one for the follower. This is a Mendix concept and relies on the Mendix Clustering functionality. When adding a follower, cluster mode must be enabled (by setting the value of *mendix.clustering* in the `values.yaml` to *true*, default is *false*). Both the leader and the follower pod utilize two containers; one nginx and one Mendix runtime. The reason for running an nginx is that a Mendix application consists of static HTML and Javascript, as well as the Java-based runtime. The Javascript sends XMLHttpRequests to the nginx, which in terms proxies the request to the Mendix runtime in addition to serving static HTML. It also does HTTP URL rewriting and Header injection (most notably for CORS). This setup is also how the original Mendix buildpack does things (albeit a bit more complicated than that), and we did not want to deviate from this. In the future this may be further optmized but at this time of writing there is nothing planned for this.

The leader and the follower have a PodAntiAffinity set so that both are not running on the same host. This is taken from the original Kubernetes Manifests as they exist in the *aws-mendix-deployment* repository, which is used by the old cluster.

There is a `configmap.yaml` that contain the shared configuration for both leader and follower; each deployment also has an additional configmap that basically defines the clustering settings only. Everything else comes out of the shared `configmap.yaml`. This may be slightly confusing but this is the simplest way to do things for the clustering specifically in conjunction with how Mendix works and starts. The configmaps mounts the various configfiles at specific locations inside of the container, which the container entrypoint expects to be there. It will fail to start without these files present.

The *values.yaml* also has a *mendix.extra_settings*, which is a map for mapping arbitrary Mendix runtime settings to the Mendix runtime. See Runtime Customization | Mendix Documentation for a list of supported parameters

There is a podmonitor in place that will be installed if the Prometheus CRDs are installed in the cluster. At this time of writing this podmonitor is used to expose JVM metrics for Mimir to scrape, and later on can be expanded to scrape Mendix metrics as well (only recent versions of Mendix support the Metrics system)

The serviceaccount has support for an IAM role but using IAM roles is only supported from Mendix 9.23+, which at this time of writing has hardly any adoption amongst the Mendix applications we run. This is also the reason why IAM users are used over IAM roles.

The ingress object works off a bunch of annotations as used by the ALB Load Balancer Controller. It uses a bunch of sane defaults. The `group.name` annotation is important for us as it allows for grouping of ingress objects to load balancers. This is where the **team name** comes into play; they are grouped based off a team to have a reasonable consolidation of ALBs and is a way to reduce costs.

The value of *ingress.http_headers* is **not** set on the ingress but on the nginx of the mendix pod.

The other components are fairly self explanatory if you read through them.

Helpers

As you may know, Helm ships with a bunch of standard helpers to prevent common errors such as names being too long for the Kubernetes API server or setting labels. One helper function that was added to the Helm chart was setting the FQDN of the Mendix application. At this time of writing this is not actively used but was added to prepare support for a multi-cluster deployment where each cluster would require unique URLs.

```
{{/*
Create an ingress URL name. If the ingress.suffix parameter is set this is appended to the URL to
create unique URLs so multiple kubernetes clusters do not clash
*/}}
{{- define "mendix-app.fqdn" -}}
{{- if .Values.ingress.suffix }}
{{- printf "%s-%s.%s" (include "mendix-app.fullname" .) .Values.ingress.suffix .Values.ingress.dns_domain }}
{{- else }}
{{- printf "%s.%s" (include "mendix-app.fullname" .) .Values.ingress.dns_domain }}
{{- end }}
{{- end }}
```

How this works is that if the value `.Values.ingress.suffix` is set, it will use this to define the url. Let's say the Helm chart name (so this translates to my mendix application name) is *supermarketone,* the suffix is *cluster-01*, and the dns_domain is *mx.dsm.app* then the fqdn becomes *supermarketone-cluster-01.mx.dsm.app*. If the `.Values.ingress.suffix` is not set, it falls back to the standard value of *supermarketone.mx.dsm.app*

Adding new parameters

If you want to add a new parameter it is fairly straightforward for a simple string addition. For sake of demonstration, let's say we want to change the `com.mendix.storage.s3.Region` parameter, found under the deployment of both the leader and follower, from being hardcoded to `eu-west-1` to something configurable. Let's go over it in steps:

1. Decide what category the parameter falls under in the *values.yaml*. If it has to do with Mendix itself, put it under `mendix`, if it's database related, put it under `database`, if it has to do with the ingress, add it under `ingress` etc.
2. Our example parameter is an interesting one as you can both consider it as the `mendix` category (it's a Mendix parameter) but it's also an `aws` category as a region is an AWS region. In this case it makes more sense to use it as an `aws` parameter as it can be commonly reused for other configuration parameters that require a configurable bucket.
3. Based on point 2, we add the following to the `values.yaml` (the bucket_name parameter is mentioned here for brevity only, only line 5 and 6 are actual new additions) :

```
1 aws:
2   # The name of the S3 bucket that Mendix uses for its storage
3   bucket_name: helm-default-bucket-name
4 
5   # The AWS region that the Mendix application runs on
6   region: eu-west-1
```

4. Now we need to find the hardcoded values and instruct Helm to use this value on all places where we need. For sake of this example we will only change `configmap.yaml`

```
[snip]
    mxruntime:
      ApplicationRootUrl: {{ include "mendix-app.fqdn" . }}
      com.mendix.core.StorageService: "com.mendix.storage.s3"
      com.mendix.storage.s3.BucketName: {{ .Values.aws.bucket_name }}
      com.mendix.storage.s3.AccessKeyId: {{ .Values.aws.iam_access_key_id }}
      com.mendix.storage.s3.SecretAccessKey: {{ .Values.aws.iam_secret_access_key }}
      com.mendix.storage.s3.Region: "eu-west-1"
[snip]
```

5. Replace the hardcoded value with the templated value (line 8 is the only one that changes):

```
1 [snip]
2     mxruntime:
3       ApplicationRootUrl: {{ include "mendix-app.fqdn" . }}
4       com.mendix.core.StorageService: "com.mendix.storage.s3"
5       com.mendix.storage.s3.BucketName: {{ .Values.aws.bucket_name }}
6       com.mendix.storage.s3.AccessKeyId: {{ .Values.aws.iam_access_key_id }}
7       com.mendix.storage.s3.SecretAccessKey: {{ .Values.aws.iam_secret_access_key }}
8       com.mendix.storage.s3.Region: {{ .Values.aws.region }}
9 [snip]
```

6. Save and commit your work. Now if no value is given to the `helm install` command it will use the default value we specified, namely `eu-west-1`

Overriding the new parameter

Now that we have a default value we also want to be able to set this. There are two ways to achieve this. First will take the SSM approach as that is the most commonly used approach for us, and secondly we will describe the generic way of doing this via Helm directly. The latter is discouraged as it makes the deployment process more convoluted but technically very much possible to do.

For SSM and following the convention used in the SSM parameter store, our new parameter can be overridden by `generate_helm_values.py` by inserting a new SSM parameter under the following path: `/dsm/mendix/apps/<app_name>/config/aws/region`. This is because the `aws` maps to the `aws` category as outlined in step 1, 2 and 3 mentioned above, and `region` is the newly added parameter in the *values.yaml*. So for each application that needs to have a different value for its `region` can set this SSM parameter with a value of the region that you are running in. For example, this could have a value of `ap-southeast-1` while the default is `eu-west-1`.

The helm native way is to provide the value for the parameter either via a separate yaml file (this is what we already use in the pipeline for the generated values.yaml done by `generate_helm_values.py`) or provide the value directly to the `helm install` command by adding a `--set` flag to it. As we have a lot of values, it would mean a lot of `--set` parameters that we have to pass along. Using a second YAML file is better hence we created `generate_helm_values.py` to automate this process according to a convention.

So if in doubt, just add an SSM parameter unless you have a good reason to pass it directly to the `helm install` command.

> ⚠ The above example assumes you are using simple strings. For lists things are a bit more complicated and, depending on the format of the list, may require you to modify `generate_helm_values.py` to add support for the format you are adding. Read on for more info on this

Adding List or Dict parameters

These are a bit more complicated as it likely means it cannot be natively loaded and will produce an incorrect format. Make sure you develop and test this locally first by executing `generate_helm_values.py` locally on your laptop and ensure that the `values_generated.py` is in the expected format before adding new parameters to (production) apps.

The safest thing to do for a multi-value parameter is to serialize it as JSON and store the JSON string as a simple text / string parameter in SSM. You can then modify the `merge_to_helm_format` to add support for this. In the future this may change to add auto detection support for this and try to load it, at this time of writing this is not in place and you are required to add it there (like how `extra_settings` is done). StringLists, which are comma separated strings in SSM, would have to be converted to a python list first, but this is also not supported at this time of writing. You will have to add some code for that (or perhaps you are lucky and by the time you read this this was already added).