The Helmfile Best Practices Guide

This guide covers the Helmfile's considered patterns for writing advanced helmfiles. It focuses on how helmfile should be structured and executed.

Helmfile .Values vs Helm .Values

Templating engine of Helmfile uses the same pipeline name `.Values` as Helm, so in some use-cases `.Values` of Helmfile and
Helm can be seen in the same file. To distinguish these two kinds of `.Values`, Helmfile provides an alias `.StateValues`
for its `.Values`.

```
app:
  project: {{.Environment.Name}}-{{.StateValues.project}} # Same as {{.Environment.Name}}-{{.Values.project}}

{{`
extraEnvVars:
- name: APP_PROJECT
  value: {{.Values.app.project}}
`}}
```

Missing keys and Default values

helmfile tries its best to inform users for noticing potential mistakes.

One example of how helmfile achieves it is that, `helmfile` fails when you tried to access missing keys in environment values.

That is, the following example let `helmfile` fail when you have no `eventApi.replicas` defined in environment values.

```
{{ .Values.eventApi.replicas | default 1 }}
```

In case it isn't a mistake and you do want to allow missing keys, use the `get` template function:

```
{{ .Values | get "eventApi.replicas" nil }}
```

This result in printing `<no value` in your template, that may or may not result in a failure.

If you want a kind of default values that is used when a missing key was referenced, use `default` like:

```
{{ .Values | get "eventApi.replicas" 1 }}
```

Now, you get `1` when there is no `eventApi.replicas` defined in environment values.

Release Template / Conventional Directory Structure

Introducing helmfile into a large-scale project that involves dozens of releases often results in a lot of repetitions in `helmfile.yaml` files.

The example below shows repetitions in `namespace`, `chart`, `values`, and `secrets`:

```
releases:
# *snip*
- name: heapster
  namespace: kube-system
  chart: stable/heapster
  version: 0.3.2
  values:
  - "./config/heapster/values.yaml"
  - "./config/heapster/{{ .Environment.Name }}.yaml"
  secrets:
  - "./config/heapster/secrets.yaml"
  - "./config/heapster/{{ .Environment.Name }}-secrets.yaml"

- name: kubernetes-dashboard
  namespace: kube-system
  chart: stable/kubernetes-dashboard
  version: 0.10.0
  values:
  - "./config/kubernetes-dashboard/values.yaml"
  - "./config/kubernetes-dashboard/{{ .Environment.Name }}.yaml"
  secrets:
  - "./config/kubernetes-dashboard/secrets.yaml"
  - "./config/kubernetes-dashboard/{{ .Environment.Name }}-secrets.yaml"
```

This is where Helmfile's advanced feature called Release Template comes handy.

It allows you to abstract away the repetitions in releases into a template, which is then included and executed by using YAML anchor/alias:

```
templates:
  default:
    chart: stable/{{`{{ .Release.Name }}`}}
    namespace: kube-system
    # This prevents helmfile exiting when it encounters a missing file
    # Valid values are "Error", "Warn", "Info", "Debug". The default is "Error"
    # Use "Debug" to make missing files errors invisible at the default log level(--log-level=INFO)
    missingFileHandler: Warn
    values:
    - config/{{`{{ .Release.Name }}`}}/values.yaml
    - config/{{`{{ .Release.Name }}`}}/{{`{{ .Environment.Name }}`}}.yaml
    secrets:
    - config/{{`{{ .Release.Name }}`}}/secrets.yaml
    - config/{{`{{ .Release.Name }}`}}/{{`{{ .Environment.Name }}`}}-secrets.yaml

releases:
- name: heapster
  version: 0.3.2
  inherit:
  - template: default
    except:
      - secrets
- name: kubernetes-dashboard
  version: 0.10.0
  inherit:
  - template: default
```

Release Templating supports the following parts of release definition:

- basic fields: `name`, `namespace`, `chart`, `version`

- boolean fields: `installed`, `wait`, `waitForJobs`, `verify` by the means of additional text fields designed for templating only: `installedTemplate`, `waitTemplate`, `verifyTemplate`

```
# ...
  installedTemplate: '{{`{{ eq .Release.Namespace "kube-system" }}`}}'
  waitTemplate: '{{`{{ eq .Release.Labels.tag "safe" | not }}`}}'
# ...
```

- `set` block values:

```
# ...
  setTemplate:
  - name: '{{`{{ .Release.Name }}`}}'
    values: '{{`{{ .Release.Namespace }}`}}'
# ...
```

- `values` and `secrets` file paths:

```
# ...
  valuesTemplate:
  - config/{{`{{ .Release.Name }}`}}/values.yaml
  secrets:
  - config/{{`{{ .Release.Name }}`}}/secrets.yaml
# ...
```

- inline `values` map:

```
# ...
  valuesTemplate:
  - image:
      tag: '{{`{{ .Release.Labels.tag }}`}}'
# ...
```

Previously, we've been using YAML anchors for release template inheritance.
It turned out not work well when you wanted to nest templates for complex use cases and/or you want a fine control over which fields to inherit or not.
Thus we added a new way for inheritance, which uses the `inherit` field we introduced above.

See issue helmfile/helmfile#435 for more context.

You might also find issue roboll/helmfile#428 useful for more context on how we originally designed the release template and what it's supposed to solve.

Layering Release Values

Please note, that it is not possible to layer `values` sections. If `values` is defined in the release and in the release template, only the `values` defined in the release will be considered. The same applies to `secrets` and `set`.

Layering State Files

> See **Layering State Template Files** if you're layering templates.

You may occasionally end up with many helmfiles that shares common parts like which repositories to use, and which release to be bundled by default.

Use Layering to extract the common parts into a dedicated *library helmfiles*, so that each helmfile becomes DRY.

Let's assume that your `helmfile.yaml` looks like:

```
bases:
- environments.yaml

releases:
- name: metricbeat
  chart: stable/metricbeat
- name: myapp
  chart: mychart
```

Whereas `environments.yaml` contained well-known environments:

```
environments:
  development:
  production:
```

At run time, `bases` in your `helmfile.yaml` are evaluated to produce:

```
---
# environments.yaml
environments:
  development:
  production:
---
# helmfile.yaml
releases:
- name: myapp
  chart: mychart
- name: metricbeat
  chart: stable/metricbeat
```

Finally the resulting YAML documents are merged in the order of occurrence,
so that your `helmfile.yaml` becomes:

```
environments:
  development:
  production:

releases:
- name: metricbeat
  chart: stable/metricbeat
- name: myapp
  chart: mychart
```

Great!

Now, repeat the above steps for each your `helmfile.yaml`, so that all your helmfiles becomes DRY.

Please also see the discussion in the issue 388 for more advanced layering examples.

Merging Arrays in Layers

Helmfile doesn't merge arrays across layers. That is, the below example doesn't work as you might have expected:

```
releases:
- name: metricbeat
  chart: stable/metricbeat
---
releases:
- name: myapp
  chart: mychart
```

Helmfile overrides the `releases` array with the latest layer so the resulting state file will be:

```
releases:
# metricbeat release disappeared! but that's how helmfile works
- name: myapp
  chart: mychart
```

A work-around is to treat the state file as a go template and use `readFile` template function to import the common part of your state file as a plain text:

`common.yaml`:

```
templates:
  metricbeat: &metricbeat
    name: metricbeat
    chart: stable/metricbeat
```

`helmfile.yaml`:

```
{{ readFile "common.yaml" }}

releases:
- <<: *metricbeat
- name: myapp
  chart: mychart
```

Layering State Template Files

Do you need to make your state file even more DRY?

Turned out layering state files wasn't enough for you?

Helmfile supports an advanced feature that allows you to compose state "template" files to generate the final state to be processed.

In the following example `helmfile.yaml.gotmpl`, each `---` separated part of the file is a go template.

`helmfile.yaml.gotmpl`:

```
# Part 1: Reused Environment Values
bases:
  - myenv.yaml
---
# Part 2: Reused Defaults
bases:
  - mydefaults.yaml.gotmpl
---
# Part 3: Dynamic Releases
releases:
  - name: test1
    chart: mychart-{{ .Values.myname }}
    values:
      - replicaCount: 1
        image:
          repository: "nginx"
          tag: "latest"
```

Suppose the `myenv.yaml` and `test.env.yaml` loaded in the first part looks like:

`myenv.yaml`:

```
environments:
  test:
    values:
      - test.env.yaml
```

`test.env.yaml`:

```
kubeContext: test
wait: false
cvOnly: false
myname: "dog"
```

Where the gotmpl file loaded in the second part looks like:

`mydefaults.yaml.gotmpl`:

```
helmDefaults:
  kubeContext: {{ .Values.kubeContext }}
  verify: false
  {{ if .Values.wait }}
  wait: true
  {{ else }}
  wait: false
  {{ end }}
  timeout: 600
  recreatePods: false
  force: true
```

Each go template is rendered in the context where `.Values` is inherited from the previous part.

So in `mydefaults.yaml.gotmpl`, both `.Values.kubeContext` and `.Values.wait` are val

environment values inherited from the previous part(=the first part) of your `helmfile.yaml.gotmpl`, and therefore the template is rendered to:

```
helmDefaults:
  kubeContext: test
  verify: false
  wait: false
  timeout: 600
  recreatePods: false
  force: true
```

Similarly, the third part of the top-level `helmfile.yaml.gotmpl`, `.Values.myname` is valid as it is included in the environment values inherited from the previous parts:

```
# Part 3: Dynamic Releases
releases:
  - name: test1
    chart: mychart-{{ .Values.myname }}
    values:
      replicaCount: 1
      image:
        repository: "nginx"
        tag: "latest"
```

hence rendered to:

```
# Part 3: Dynamic Releases
releases:
  - name: test1
    chart: mychart-dog
    values:
      replicaCount: 1
      image:
        repository: "nginx"
        tag: "latest"
```

Re-using environment state in sub-helmfiles

Do you want to decouple the environment state loading from the sub-helmfiles and load it only once?

This example shows how to do this:

```
environments:
  stage:
    values:
    - env/stage.yaml
  prod:
    values:
    - env/prod.yaml
---

helmfiles:
- path: releases/myrelease/helmfile.yaml
  values:
  - {{ toYaml .Values | nindent 4 }}
  # pass the current state values to the sub-helmfile
  # add other values to use overlay logic here
```

and `releases/myrelease/helmfile.yaml` is as DRY as

```
releases:
- name: mychart-{{ .Values.myrelease.myname }}
  installed: {{ .Values | get "myrelease.enabled" false }}
  chart: mychart
  version: {{ .Values.myrelease.version }}
  labels:
```

```
  chart: mychart
values:
- values.yaml.gotmpl
# templated values would also inherit the values passed from upstream
```