WORKING WITH HELM

DevelopIntelligence

WELCOME

Develop Intelligence

Mark Your Attendance:

- ➢ Go to the Course Calendar Invite
- ➢ Click on the Course Event Link
- ➢ Press Check-In

VIRTUAL TRAINING

What we want

...what we've got

VIRTUAL TRAINING EXPECTATIONS FOR ME

I pledge to:

- Make this as interesting and interactive as possible
- Ask questions in order to stimulate discussion
- Use whatever resources I have at hand to explain the material
- Try my best to manage verbal responses so that everyone who wants to speak can do so
- Use an on-screen timer for breaks so you know when to be back

VIRTUAL TRAINING EXPECTATIONS FOR YOU

- Arrive and Return on Time
- Mute unless speaking
- Use chat or ask questions verbally

ABOUT DEVELOPINTELLIGENCE

Our purpose…

We help organizations learn and adopt new technologies.

…Impacts you daily.

When you talk on the phone, watch a movie, connect with friends on social media, drive a car, fly on a plane, pay with a credit card, shop online, and order a latte with your mobile app, you are interacting with technology developed by one of our customers.

TECHNOLOGIES WE TEACH

Ruby · Jenkins · akka · Android · ANGULAR · ANSIBLE · Apache Spark

aws · Azure · C# · cassandra · CHEF · D3 · docker

Git · GO · Google Cloud · GraphQL · iOS · Java · JS

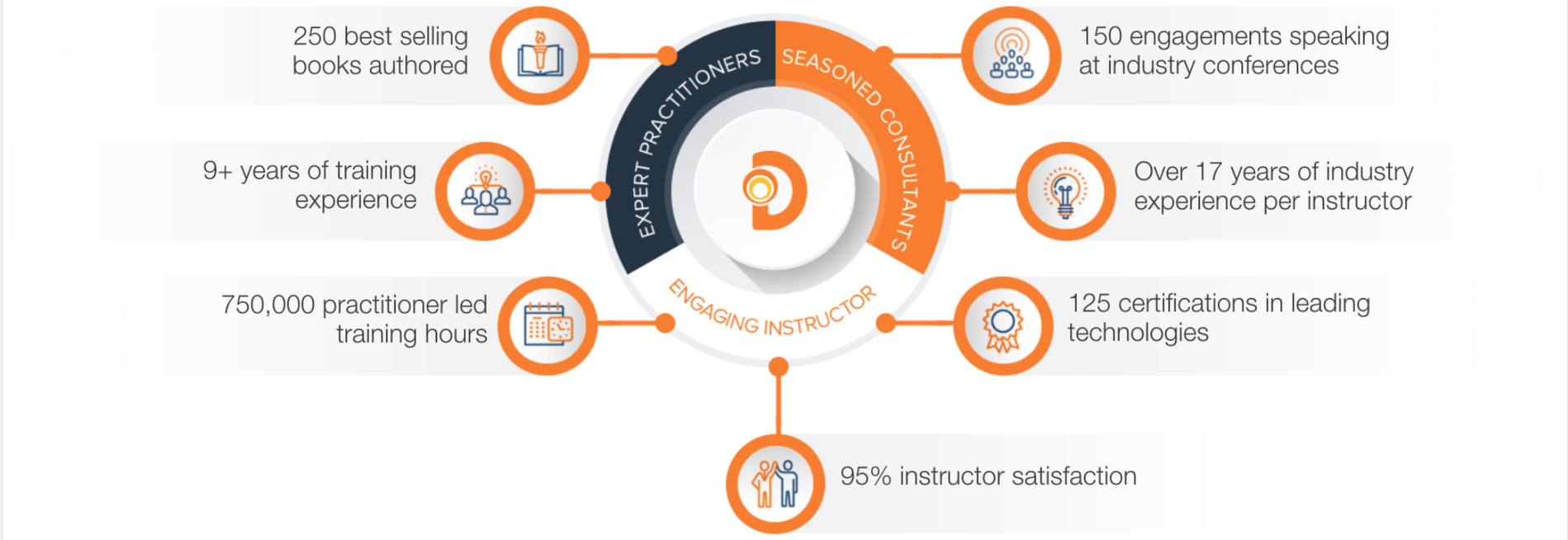
kafka · Kubernetes · mongoDB · MySQL · node.js · puppet · python

Rust · React · React Native · redis · Scala · spring · Swift

Kotlin · Terraform · TS · Vue

AND MANY OTHER TRENDING TECHNOLOGIES

OUR PRACTITIONERS



COURSE OVERVIEW

- The Helm Fundamentals course is designed to be an introduction to understanding and working with the Helm Kubernetes package manager.
- It will start with a journey to understand why Helm exists and its purpose.
- The course will then navigate a path towards how to work with it, including
  - Building charts
  - Publishing charts
  - Using them to release and manage Kubernetes workloads
  - Various capabilities Helm offers via its CLI

AUDIENCE

Developers, Service Owners, Ops

PURPOSE

To gain a comprehensive understanding of Helm and how to use it to package and manage Kubernetes workloads

PREREQUISITES FOR THE CLASS

- Knowledge and working experience with Kubernetes. This includes understanding common resources and usage of things like Deployments, Config Maps, Secrets, Pods, Container Spec, etc.
- Some level of comfort with the command line (kubectl, piping output, navigating folders, etc)

OBJECTIVES

- Understand the purpose of Helm, why it exists
- Create Helm charts of your own ready for distribution
- Test helm in local, and how to validate your changes
- Understand all features and practices around Helm chart source code
- Feel comfortable with best practices around using Helm as well as gotchas of Helm

OBJECTIVES CONTINUED

- Know how Helm charts are commonly distributed and stored, and work with Helm chart repos
- Understand the architecture of Helm, where it is today, and where it's headed
- Install the Helm CLI, initialize the client and server

OBJECTIVES CONTINUED

- Use the various features of the CLI including packaging charts, installing and using plugins, installing and upgrading charts.
- Have awareness of development and testing strategies around Helm charts, and be able to use these strategies
- How to use Helm securely

INTRODUCTION

PURPOSE

- Manages Kubernetes Applications
- Templatizes Kubernetes Resources
- Repeatable Application Installation
- Upgrade Applications
- Reusability
- Share Charts on Repositories
- Rollback to Previous Applications

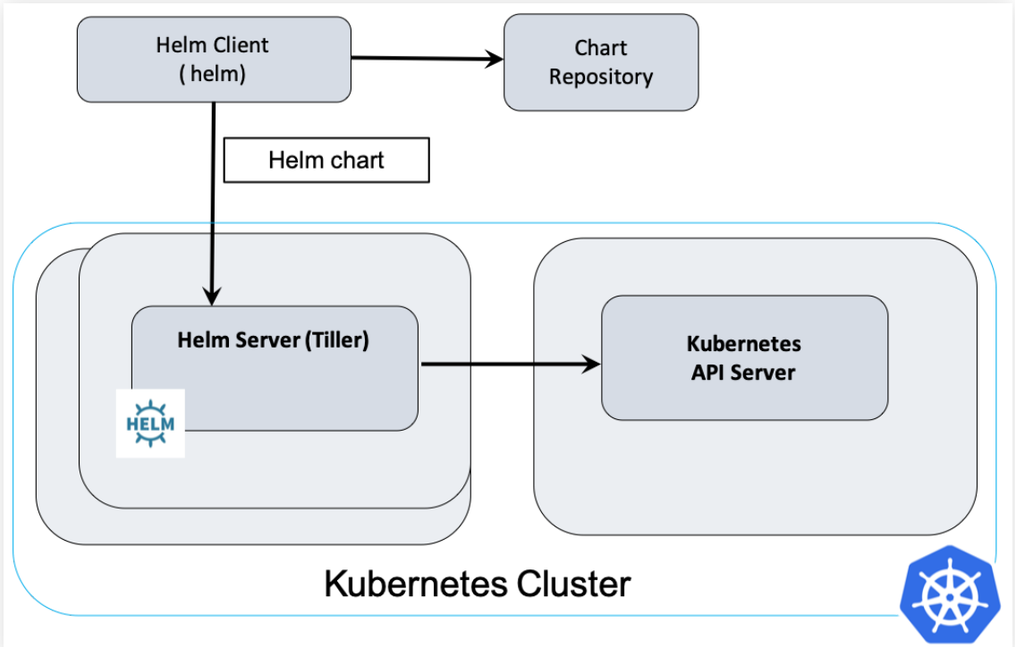
ARCHITECTURE

- Discuss the components of Helm
- Discuss how Helm is implemented

COMPONENTS

Helm has two components:

- Client
- Tiller



HELM CLIENT

- Command Line for End Users
- Local Chart Development
- Managing Repositories
- Interacting with the Tiller Server
  - Sending charts to be installed
  - Asking for information about releases
  - Requesting upgrading or uninstalling of existing releases

TILLER SERVER

- In Kubernetes Cluster that interacts with the Helm Client
- Interfaces with Kubernetes API Server
- The server is responsible for the following:
  - Listening for incoming requests from the Helm client
  - Combining a chart and configuration to build a release
  - Installing charts into Kubernetes, and then tracking the subsequent release
  - Upgrading and uninstalling charts by interacting with Kubernetes

IMPLEMENTATION

- Helm is written in the Go Language
- gRPC is used to interact with the server
- Uses the Kubernetes client library to communicate with Kubernetes
- The Tiller server stores information in *ConfigMaps* located inside of Kubernetes. It does not need its own database.

INSTALLING

In this section:

- Installing Kubectl
- Installing Hypervisor and Minikube
- Installing Docker and Kind
- Installing Helm
- Installing Tiller

INSTALLING `kubectl` ON MAC

```
% brew install kubectl
```

INSTALLING `kubectl` ON LINUX

Run the following curl download

```
curl -LO https://storage.googleapis.com/kubernetes-
release/release/$(curl -s https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl
```

Make `kubectl` binary executable

```
% chmod +x ./kubectl
```

Move the binary to the `PATH`

```
% sudo mv ./kubectl /usr/local/bin/kubectl
```

CHECKING IF YOU HAVE VIRTUALIZATION

Having Virtualization is required if you wish to use Minikube

ON LINUX

```
% grep -E --color 'vmx|svm' /proc/cpuinfo
```

ON MACOSX

```
% sysctl -a | grep -E --color 'machdep.cpu.features|VMX'
```

ON WINDOWS

```
C:/> systeminfo
```

If you see the following, virtualization is supported

```
Hyper-V Requirements:      VM Monitor Mode Extensions: Yes
                           Virtualization Enabled In Firmware: Yes
                           Second Level Address Translation: Yes
                           Data Execution Prevention Available: Yes
```

IF YOU DO NOT HAVE VIRTUALIZATION

Install one of the following virtualization packages

- Hyperkit
- VirtualBox
- VMware Fusion

INSTALLING DOCKER

Download and install the latest docker client: https://docs.docker.com/get-docker/

INSTALL KUBERNETES

We are going to be deploying applications on Kubernetes, therefore we need a cluster Typically, we will manage remote clusters, but for class, and experimentation we will use Minikube There are other solutions:

- Remote Kubernetes
  - AWS
  - GCP
  - More…
- Local Kubernetes
  - Minikube
  - Kind
  - K3S

INSTALLING MINIKUBE

```
% brew install minikube
```

VIEWING THE MINIKUBE DASHBOARD

```
% minikube dashboard
```

INSTALLING KIND

```
% cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
        authorization-mode: "AlwaysAllow"
  extraPortMappings:
  - containerPort: 80
```

ALLOWING US TO DEPLOY TO KIND

```
% kubectl create clusterrolebinding add-on-cluster-admin --
clusterrole=cluster-admin --serviceaccount=kube-system:default
```

INSTALLING HELM

- Binary Source
- Homebrew (MacOSX)
- Chocolatey (Windows)
- Snap (Linux)

HOMEBREW INSTALL OF HELM ON MACOSX

```
% brew install helm
```

LINUX HELM INSTALL

- Download from https://github.com/helm/helm/releases
- Unpack it (`tar -zxvf helm-v2.0.0-linux-amd64.tgz`)
- Find the helm binary in the unpacked directory, and move it to its desired destination (`mv linux-amd64/helm /usr/local/bin/helm`)

OTHER INSTALLATION INSTRUCTIONS

https://v2.helm.sh/docs/install/

STOPPING `minikube`

```
% minikube stop
```

CLEARING `minikube` STATE

```
% minikube delete
```

STOPPING KIND

```
% kind delete cluster
```

BASIC OPERATIONS

GLOSSARY OF TERMS

CHART DEFINITION

- A Helm package.
- Contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster.
- Think of it like the Kubernetes equivalent of a Homebrew formula, an Apt dpkg, or a Yum RPM file.

REPOSITORY DEFINITION

- A Repository is the place where charts can be collected and shared.
- It’s like Perl’s CPAN archive, Fedora Package Database, or Maven Central but for Kubernetes packages.

RELEASE DEFINITION

- A Release is an *instance* of a chart running in a Kubernetes cluster.
- One chart can often be installed many times into the same cluster.
- Each time it is installed, a new release is created.
- For Example:
  - If you want two databases running in your cluster, you can install that chart twice.
  - Each one will have its own release, which will in turn have its own release name.

GLOSSARY REVIEW

Helm installs charts into Kubernetes, creating a new release for each installation. And to find new charts, you can search Helm chart repositories.

INITIALIZING HELM

- To set up using helm, call `helm init`
- Establishes plugins, repositories, starters, and more
- Installs tiller onto your Kubernetes Cluster

RUNNING `helm init`

```
% helm init
```

Responds with:

```
% helm init
Creating /Users/sue/.helm
Creating /Users/sue/.helm/repository
Creating /Users/sue/.helm/repository/cache
Creating /Users/sue/.helm/repository/local
Creating /Users/sue/.helm/plugins
Creating /Users/sue/.helm/starters
Creating /Users/sue/.helm/cache/archive
Creating /Users/sue/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-
charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /Users/sue/.helm.
```

`HELM_HOME` contains cached resources and configuration

DETERMINING `tiller`

```
% kubectl get deploy
```

SETTING UP A REPOSITORY

REPOSITORY UPDATE

```
% helm repo update
```

FINDING CHARTS

- When you first use Helm it comes by default with a carefully curated repository, called `stable`
- You can search for charts using `helm search`

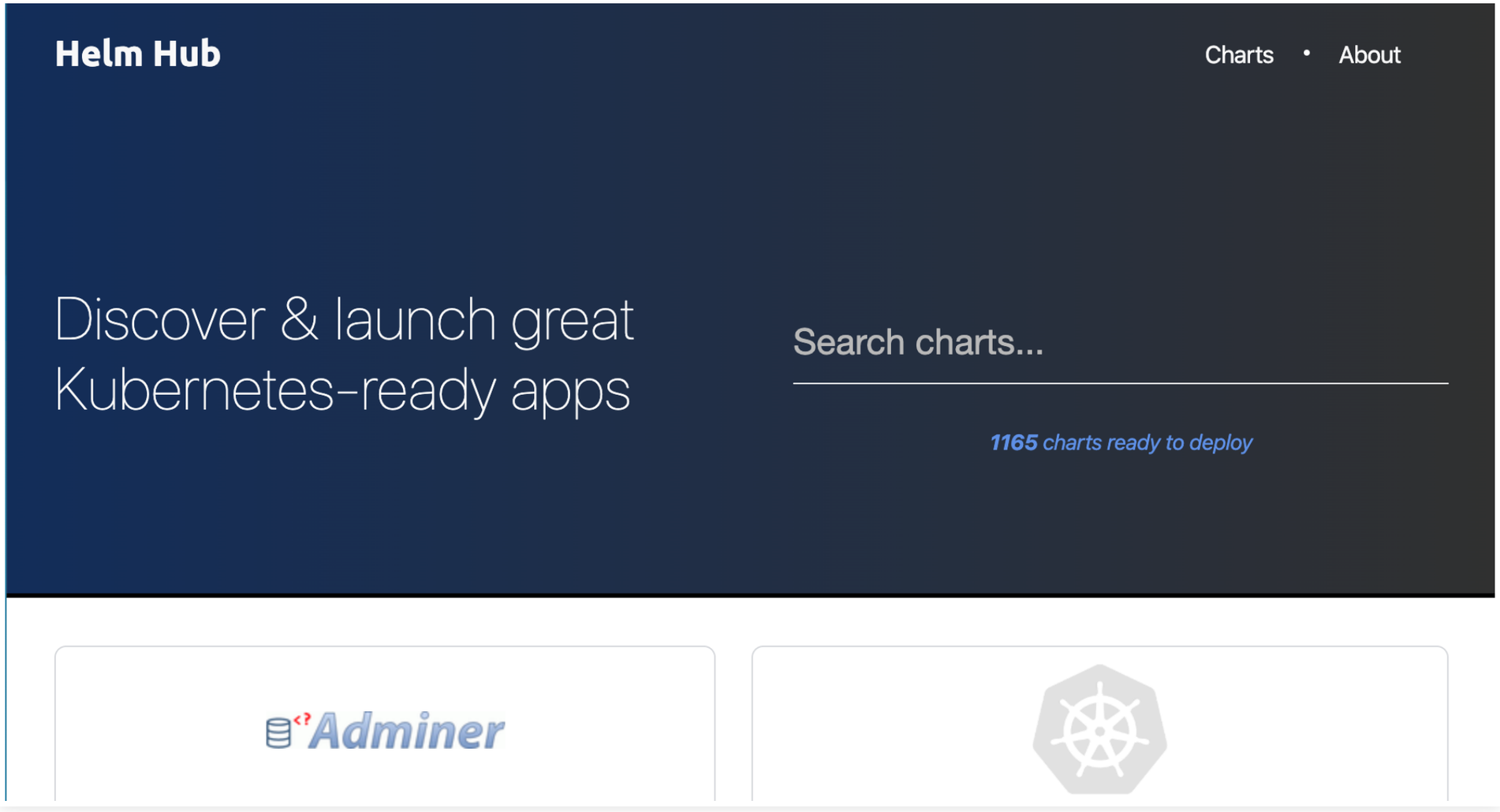
```
% helm search
```

Returns for example:

```
NAME                                    CHART VERSION   APP VERSION
DESCRIPTION
stable/acs-engine-autoscaler            2.2.2           2.1.1
DEPRECATED Scales worker nodes within agent pools
stable/aerospike                        0.3.2           v4.5.0.5
A Helm chart for Aerospike in Kubernetes
stable/airflow                          6.5.0           1.10.4
Airflow is a platform to programmatically author, schedul...
stable/ambassador                       5.3.1           0.86.1
A Helm chart for Datawire Ambassador
stable/anchore-engine                   1.5.0           0.7.0
Anchore container analysis and policy evaluation engine s..
```

HELM CHARTS

Another way to find Charts is Helm Hub: https://hub.helm.sh/



SEARCH FOR SPECIFIC CHARTS

```
% helm search hadoop
```

```
%  helm search hadoop
danno@DannoAir
NAME            CHART VERSION   APP VERSION     DESCRIPTION
stable/hadoop   1.1.2           2.9.0           The Apache Hadoop
software library is a framework that al...
stable/luigi    2.7.5           2.7.2           Luigi is a Python module
that helps you build complex pip...
```

Now you will only see the results that match your filter

INSPECT YOUR SEARCH

- You can view information about your chart including
  - Maintainers
  - Instructions
  - Values

```
% helm inspect <chart>
```

INSPECT YOUR SEARCH EXAMPLE

```
% helm inspect stable/hadoop
```

Renders:

```
apiVersion: v1
appVersion: 2.9.0
description: The Apache Hadoop software library is a framework that
allows for the
  distributed processing of large data sets across clusters of computers
using simple
  programming models.
home: https://hadoop.apache.org/
icon: http://hadoop.apache.org/images/hadoop-logo.jpg
```

INSTALL CHART

```
% helm install stable/mysql
```

VIEW INFORMATION ON A RELEASE

```
% helm inspect stable/mysql
```

- Each release gets its own name
- Therefore each can be managed separately

SHOW ALL RELEASES

```
% helm ls
```

To show all deployed releases

```
% helm list
```

UNINSTALL A RELEASE

```
% helm delete --purge <release-name>
```

For example:

```
% helm delete --purge righteous-fiber
```

DETERMINING THE STATUS

- Deployments need time to download containers
- Check the status with the following
  - This will also show a NOTES.txt, basic set of chart instructions

```
% helm status righteous-fiber
```

QUERYING VALUES TO OVERRIDE

Helm charts are extremely flexible to change configuration

```
% helm inspect values <chart>
```

```
helm inspect values stable/mariadb
```

GETTING HELP

```
% helm get -h
```

LAB 1: INSTALL CHART

Open your lab book, *lab_book.html*, and do "Lab 1: Install a public chart"

YAML PRIMER

Agenda:

- Understand Quickly YAML Specifications

Source: https://v2.helm.sh/docs/chart_template_guide/#yaml-techniques

YAML LIST

```
sequence:
- one
- two
- three
```

YAML MAP

```
map:
one: 1
two: 2
three: 3
```

INTEGER SCALARS

If an integer or float is an unquoted bare word, it is typically treated as a numeric type

```
count: 1
size: 2.34
```

If they are quoted, they are treated as strings:

```
count: "1" # <-- string, not int
size: '2.34' # <-- string, not float
```

BOOLEAN SCALARS

```
isGood: true   # bool
answer: "true" # string
```

```
null
```

```
isAvailable: null
```

COERCING YAML VALUES

- You can coerce YAML values using `!!`
- Below:
  - `!!str` tells the parser that age is a string, even if it looks like an `int`.
  - `port` is treated as an `int`, even though it is quoted.

```
coffee: "yes, please"
age: !!str 21
port: !!int "80"
```

YAML STRINGS

Three inline ways to declare a string

```
way1: bare words
way2: "double-quoted strings"
way3: 'single-quoted strings'
```

- Bare words are unquoted, and are not escaped. For this reason, you have to be careful what characters you use.
- Double-quoted strings can have specific characters escaped with `\`.
  - For example `"\"Hello\", she said"`.
  - You can escape line breaks with `\n`.
- Single-quoted strings are “literal” strings, and do not use the `\` to escape characters.
  - The only escape sequence is `''`, which is decoded as a single `'`.

MULTILINE STRINGS

- Multiline strings can be declared using `|`
- The following is equivalent to: `Latte\nCappuccino\nEspresso\n`

```
coffee: |
  Latte
  Cappuccino
  Espresso
```

The first line has be correctly indented

MULTILINE WITH TRIMMING

- The previous slide will render a multiline string with an additional newline
- To trim the extra newline, use a dash –
- The following is equivalent to: `Latte\nCappuccino\nEspresso`

```
coffee: |-
  Latte
  Cappuccino
  Espresso
```

INDENTATION IS PRESERVED

- Indentation inside of a text block is preserved, this includes line breaks
- In the following, `coffee` is `Latte\n 12 oz\n 16 oz\nCappuccino\nEspresso`

```
coffee: |-
  Latte
    12 oz
    16 oz
  Cappuccino
  Espresso
```

MAINTAINING ALL TRAILING WHITESPACES

- Using `|+` maintains all trailing whitespace
- The following evaluates to: `Latte\nCappuccino\nEspresso\n\n\n`

```
coffee: |+
  Latte
  Cappuccino
  Espresso


another: value
```

FOLDED MULTILINE STRINGS

- Sometimes you want to represent a string in your YAML with multiple lines, but want it to be treated as one long line.
- This is called “folding”.
- To declare a folded block, use `>` instead of `|`
- The following will render `coffee` as `Latte Cappuccino Espresso\n`

```
coffee: >
  Latte
  Cappuccino
  Espresso
```

FOLDING AND TRIMMING

- Given the previous slide, it still maintains the last `\n`
- We can then use `>-` to fold and trim all the newlines
- The following produces for `coffee`: `Latte\n 12 oz\n 16 oz\nCappuccino Espresso`

```
coffee: >-
  Latte
    12 oz
    16 oz
  Cappuccino
  Espresso
```

EMBEDDING MULTIPLE DOCUMENTS

- You can embed multiple yaml files in a single file
- This is done by prefixing: _, and ending with ...
- In many cases they can be omitted since it would be up to the parser

```
---
document:1
...
---
document: 2
...
```

YAML ANCHORS

- YAML spec provides a way to store a reference to a value and later refer to that value by reference.
- YAML refers to this as “anchoring”
- Use `&` as the reference
- Use `*` as the pointer

```
coffee: "yes, please"
favorite: &favoriteCoffee "Cappucino"
coffees:
  - Latte
  - *favoriteCoffee
  - Espresso
```

⚠ References is expanded and then discarded. The anchors in Helm and K8s will be lost

CREATE BASIC CHART

Agenda:

- Creating a chart
- Understanding templates
- Performing Dry Runs
- Understanding Values

CREATING A CHART

```
% helm create mychart
```

This will create:

```
.
└── mychart
    ├── Chart.yaml
    ├── charts
    ├── templates
    │   ├── NOTES.txt
    │   ├── _helpers.tpl
    │   ├── deployment.yaml
    │   ├── ingress.yaml
    │   ├── service.yaml
    │   ├── serviceaccount.yaml
    │   └── tests
    │       └── test-connection.yaml
    └── values.yaml
```

Chart.yaml

A default YAML file containing information about the chart

```
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for Kubernetes
name: mychart
version: 0.1.0
```

apiVersion	Chart API version
appVersion	Version that app contains
description	Single sentence description
name	Name of the project
version	SemVer 2 version

`charts` DIRECTORY

The `charts` directory is a directory containing any charts upon which this chart depends

`templates` DIRECTORY

A directory of templates that, when combined with values, will generate valid Kubernetes manifest files

- `_helpers.tpl` - helper templates
- Kubernetes files
- tests - unit

ADDITIONAL FIELDS

`kubeVersion`	SemverRange of compatible K8s versions
`keywords`	List of keywords
`home`	Project's Home Page
`sources`	List of URLs of source code
`maintainers`	List of maintainers (see maintainer object)

ADDITIONAL FIELDS CONTINUED

`engine`	Name of the template engine; default: Go Template Engine
`icon`	SVG or PNG representing the chart
`deprecated`	Whether this chart is deprecated
`tillerVersion`	Version of Tiller (Semver)

MAINTAINERS:

The maintainers, previous slide looks like the following

`name`	Maintainer's name
`email`	Maintainer's email
`url`	Maintainer's url

SEMVER STANDARD

- https://semver.org/
- Given a version number MAJOR.MINOR.PATCH, increment the:
  - **MAJOR** version when you make incompatible API changes,
  - **MINOR** version when you add functionality in a backwards compatible manner, and
  - **PATCH** version when you make backwards compatible bug fixes.
  - Additional labels for pre-release and build metadata are available as extensions to the **MAJOR.MINOR.PATCH** format.
- There are additional rules, see website for details

CHARTS AND VERSIONING

- Versions must follow SemVer2 Versioning
- Example the tar ball that represents nginx, given the name `nginx` and version `1.2.3`

```
nginx-1.2.3.tgz
```

- Version field is in the *chart.yml* is used by various tools. All version numbers *must match*.

appVersion

- App version is for users' use, purely informational
- Doesn’t follow SemVer rules

DEPRECATING CHARTS

- The optional `deprecated` field in *Chart.yaml* can be used to mark a chart as deprecated
- If the latest version of a chart in the repository is marked as deprecated, then the chart as a whole is considered to be deprecated.
- The chart name can later be reused by publishing a newer version that is not marked as `deprecated`

LICENSE, README, _NOTES.TXT

- *LICENSE* is the licensing used in Helm
- *README* is Markdown information on how to use the chart
- *template/NOTES.txt* used to display the status of the release when using `helm package` or `helm status`

CHART DEPENDENCIES

requirements.yaml is used to link one chart with another chart

```
dependencies:
- name: apache
  version: 1.2.3
  repository: http://example.com/charts
- name: mysql
  version: 3.2.1
  repository: http://another.example.com/charts
```

- The `name` field is the name of the chart you want.
- The `version` field is the version of the chart you want.
- The `repository` field is the full URL to the chart repository.

You must also use helm repo add to add that repo locally.

DOWNLOADING DEPENDENCIES

```
% helm dep up foochart
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "local" chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "example" chart repository
...Successfully got an update from the "another" chart repository
Update Complete.
Saving 2 charts
Downloading apache from repo http://example.com/charts
Downloading mysql from repo http://another.example.com/charts
```

HOW DEPENDENCIES ARE STORED

The tarballs will be installed in the *charts/* directory

```
charts/
  apache-1.2.3.tgz
  mysql-3.2.1.tgz
```

USING `alias`

- Use `alias` in cases where they need to access a chart with other name
- The following is using the same chart three different times

```
dependencies:
  - name: subchart
    repository: http://localhost:10191
    version: 0.1.0
    alias: new-subchart-1
  - name: subchart
    repository: http://localhost:10191
    version: 0.1.0
    alias: new-subchart-2
  - name: subchart
    repository: http://localhost:10191
    version: 0.1.0
```

Which creates:

```
subchart
new-subchart-1
```

USING CONDITIONS

- The condition field holds one or more YAML paths (delimited by commas).
- If this path exists in the *parent's values* and resolves to a boolean value,
- the chart will be enabled or disabled based on that boolean value.
- Only the first valid path found in the list is evaluated and if no paths exist then the condition has no effect.
- For multiple level dependencies the condition is prependend by the path to the parent chart.

USING TAGS

- The tags field is a YAML list of labels to associate with this chart.
- In the top parent's values, all charts with tags can be enabled or disabled by specifying the tag and a boolean value.

PARENT CHART

```
# parentchart/requirements.yaml
dependencies:
  - name: subchart1
    repository: http://localhost:10191
    version: 0.1.0
    condition: subchart1.enabled
    tags:
      - front-end
      - subchart1

- name: subchart2
  repository: http://localhost:10191
  version: 0.1.0
  condition: subchart2.enabled
```

SUBCHART 2 REQUIREMENTS

```
# subchart2/requirements.yaml
dependencies:
  - name: subsubchart
    repository: http://localhost:10191
    version: 0.1.0
    condition: subsubchart.enabled
```

PARENT CHART VALUES

```
subchart1:
  enabled: true
subchart2:
  subsubchart:
    enabled: false
tags:
  front-end: false
  back-end: true
```

- All charts with `front-end` would be disabled, but
- Conditions override tags, `subchart1` is enabled, therefore `subchart1` is enabled.

TAG & CONDITION

- Conditions (when set in values) always override tags.
- The first condition path that exists wins and subsequent ones for that chart are ignored.
- Tags are evaluated as ‘if any of the chart’s tags are true then enable the chart’.
- Tags and conditions values must be set in the top parent’s values.
- The tags: key in values must be a top level key. Globals and nested tags: tables are not currently supported.

LAB 2: CREATE A BASIC CHART

Open your lab book, *lab_book.html*, and do "Lab 2: Create a basic chart"

TEMPLATES

CREATING `chart`

```
% helm create chart
```

Creates the following directory

```
mychart/
  Chart.yaml
  values.yaml
  charts/
  templates/
  ...
```

ABOUT TEMPLATES

- `templates` directory contain the template files
- When Tiller evaluates it will send all the files in the `templates` directory through the template engine
- Tiller collects the results and sends it through Kubernetes

`values.yml`

- Contains default values for charts
- Then can be overriden by
  - `helm install`
  - `helm upgrade`

`Chart.yml`

- Contains a description of the chart
- It can be accessed from the template
- Charts can create other charts called *subcharts*

INSIDE `templates`

- `NOTES.txt` - "Help text" - Displayed when they run `helm install`
- `deployment.yaml` - A basic manifest for creating a Kubernetes deployment
- `service.yaml` - A basic manifest for creating a service endpoint for your deployment
- `_helpers.tpl`: A place to put template helpers that you can re-use throughout the chart

EXAMPLE `template`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mychart-configmap
data:
  myvalue: "Hello World"
```

> Convention, use `.yaml` extension for yaml files, and `.tpl` for helpers

VIEWING THE TEMPLATE

We can install it using `helm install mychart`

```
% helm install ./mychart

NAME: full-coral
LAST DEPLOYED: Tue Nov  1 17:36:01 2016
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME                  DATA      AGE
mychart-configmap     1         1m
```

`full-coral` is the release name, it is either manually created or chosen randomly

RETRIEVING THE RELEASE

```
% helm get manifest full-coral


---
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mychart-configmap
data:
  myvalue: "Hello World"
```

SETTING A RELEASE NAME

- Best Practice is the replace the `Name:` field
- We can create a Template call of the name
- The dots are called *namespace objects*

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
```

RUNNING THE TEMPLATE

- Installing the chart

```
% helm install ./mychart
```

- Getting the Manifest

```
% helm get manifest ./mychart
```

DEBUGGING THE CONTENT

```
% helm install ./mychart --debug --dry-run
```

`--dry-run` will make it easier to test your code

OBJECTS

- Passed into the template engine
- Can be created within the template engine
- Can simple with one value
- Contain other objects and functions

`Release` OBJECT

- `Release`: This object describes the release itself. It has several objects inside of it:
- `Release.Name`: The release name
- `Release.Time`: The time of the release
- `Release.Namespace`: The namespace to be released into (if the manifest doesn't override)
- `Release.Service`: The name of the releasing service (always Tiller).
- `Release.Revision`: The revision number of this release. It begins at 1 and is incremented for each helm upgrade.
- `Release.IsUpgrade`: This is set to true if the current operation is an upgrade or rollback.
- `Release.IsInstall`: This is set to true if the current operation is an install.

`Values` OBJECT

- Values passed into the template from
  - The *values.yaml* file
  - From user-supplied files.
  - By default, `Values` is empty.

`Chart` OBJECT

- The contents of the *Chart.yaml* file
- Any data in Chart.yaml will be accessible here
- e.g. `{{.Chart.Name}}-{{.Chart.Version}}`

`Files` OBJECT

- This provides access to all non-special files in a chart.
- While you cannot use it to access templates, you can use it to access other files in the chart.
- `Files.Get` is a function for getting a file by name (.Files.Get config.ini)
- `Files.GetBytes` is a function for getting the contents of a file as an array of bytes instead of as a string.

Capabilities

- This provides information about what capabilities the Kubernetes cluster supports
- `Capabilities.APIVersions`- Set of versions.
- `Capabilities.APIVersions.Has $version` -
  - Indicates whether a version (e.g., batch/v1) or resource (e.g., apps/v1/Deployment) is available on the cluster.
- `Capabilities.KubeVersion`
  - Provides a way to look up the Kubernetes version.
  - It has the following values: `Major`, `Minor`, `GitVersion`, `GitCommit`, `GitTreeState`, `BuildDate`, `GoVersion`, `Compiler`, and `Platform`.

`Capabilities` CONTINUED

- `Capabilities.TillerVersion`
  - Provides a way to look up the Tiller version.
  - It has the following values:
    - SemVer
    - GitCommit
    - GitTreeState

`Template`

Contains information about the current template that is being executed

- `Name` - A namespaced filepath to the current template (e.g. `mychart/templates/mytemplate.yaml`)
- `BasePath` - The namespaced path to the templates directory of the current chart (e.g. `mychart/templates`)

CONVENTION SPELLING

- Choose lower case for `Values` objects
- Reserve upper case for built-in objects

VALUES FILES

Values comes from the following in descending order:

- The *values.yaml* file in the chart
- If this is a subchart, the *values.yaml* file of a parent chart
- A values file is passed into helm install or helm upgrade with the `-f` flag (`helm install -f myvals.yaml ./mychart`)
- Individual parameters passed with `--set` (such as `helm install --set foo=bar ./mychart`)

EXAMPLE OF GETTING AN ATTRIBUTE

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favoriteDrink }}
```

RENDERING THE ATTRIBUTE

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: geared-marsupi-configmap
data:
  myvalue: "Hello World"
  drink: coffee
```

OVERRIDING THE ATTRIBUTE

```
% helm install --dry-run --debug --set favoriteDrink=slurm ./mychart
```

Renders:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: solid-vulture-configmap
data:
  myvalue: "Hello World"
  drink: slurm
```

MULTI-LEVEL VALUES.YAML

```
favorite:
  drink: coffee
  food: pizza
```

Using the template:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink }}
  food: {{ .Values.favorite.food }}
```

Keep trees shallow in `values.yaml` as much as possible

DELETING A DEFAULT KEY

To delete a key, override the value with `null`

```
livenessProbe:
  httpGet:
    path: /user/login
    port: http
```

```
% helm install ./chart --set livenessProbe.httpGet=null
```

TEMPLATE FUNCTIONS

- Allows us to transform the values using functions
- Works much like functional programming
- Functions work with the syntax: `functionName arg1 arg2…`

quote

Places a quote (") around the food when evaluated

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | quote }}
  food: {{ .Values.favorite.food | quote }}
```

RENDERING quote

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: trendsetting-p-configmap
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "PIZZA"
```

OVER 60 FUNCTIONS AVAILABLE

Functions are available:

- Go Template Library
- Sprig template library

PIPELINES

- Analogous to Pipeline in UNIX shell scripting
- Each function is applied to the tail to perform actions
- The following is equivalent to the previous slide

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | quote }}
  food: {{ .Values.favorite.food | quote }}
```

CHAINING MULTIPLE PIPELINES

The following will convert `favorite.food` to upper case and then add a quote

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
```

RENDERING `quote` AND `upper`

Here is the result:

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: trendsetting-p-configmap
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "PIZZA"
```

ORDERING DOES MATTER

Getting the order down is important, determine the following:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | repeat 5 | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
```

ANSWER: ORDERING DOES MATTER

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: melting-porcup-configmap
data:
  myvalue: "Hello World"
  drink: "coffeecoffeecoffeecoffeecoffee"
  food: "PIZZA"
```

`default` FUNCTION

- The default function is: `default DEFAULT_VALUE GIVEN_VALUE`
- Running it will provide us with the following:
- Although works with static values, works well with functions

```
drink: {{ .Values.favorite.drink | default "tea" | quote }}
```

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: virtuous-mink-configmap
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "PIZZA"
```

REMOVING coffee

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: fair-worm-configmap
data:
  myvalue: "Hello World"
  drink: "tea"
  food: "PIZZA"
```

OPERATORS

- Operators that return a boolean value. To use `eq`, `ne`, `lt`, `gt`, `and`, `or`, `not`
- To chain operators, separate by surrounding them with parenthesis

`and` and `eq`

```
(and .Values.fooString (eq .Values.fooString "foo"))
```

`or` and `not`

```
(or .Values.anUnsetVariable (not .Values.aSetVariable))
```

FLOW CONTROL

We will cover basic flow control:

- `if`/`else` for creating conditional blocks
- `with` to specify a scope
- `range`, which provides a “for each”-style loop

And wade our way into templates

- `define` declares a new named template inside of your template
- `template` imports a named template
- `block` declares a special kind of fillable template area

if/else

```
{{ if PIPELINE }}
  # Do something
{{ else if OTHER PIPELINE }}
  # Do something else
{{ else }}
  # Default case
{{ end }}
```

"FALSY" VALUES

A pipeline is `false` if a value is:

- a `boolean` false
- a numeric zero
- an empty string
- a `nil` (empty or null)
- an empty collection (`map`, `slice`, `tuple`, `dict`, `array`)

`if/else` EXAMPLE

- The following checks for an absence of a value
- The determines the equality using `eq`
- If so, the create `mug:true`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | default "tea" | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
  {{ if and .Values.favorite.drink (eq .Values.favorite.drink "coffee")
}}mug: true{{ end }}
```

CONTROLLING WHITESPACE

- The following creates an `error converting YAML to JSON` error due to indentation
- `mug:true` is indented too far right, and is an isolated key

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | default "tea" | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
  {{if eq .Values.favorite.drink "coffee"}}
    mug: true
  {{end}}
```

RENDERING THE EXTRA SPACE

```
data:
  myvalue: "Hello World"
drink: "coffee"
food: "PIZZA"
  mug: true
```

ATTEMPT TO FIX SPACE ERROR

As an attempt we could flush the `mug:true` up to the `if`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | default "tea" | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
  {{if eq .Values.favorite.drink "coffee"}}
  mug: true
  {{end}}
```

WHAT THAT RENDERS

We get an extra carriage return, where the `{{if}}` would be

```
data:
myvalue: "Hello World"
drink: "coffee"
food: "PIZZA"

mug: true
```

USING SPECIAL CURLY-BRACED SYNTAX

- `{{-` (with the dash and space added) indicates that whitespace should be chomped left
- `-}}` means whitespace to the right should be consumed.
- Be careful! Newlines are whitespace!

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  drink: {{ .Values.favorite.drink | default "tea" | quote }}
  food: {{ .Values.favorite.food | upper | quote }}
  {{- if eq .Values.favorite.drink "coffee"}}
  mug: true
  {{- end}}
```

REMINDER: `indent` FUNCTION

You can apply the same technique with the `indent` function

```
{{indent 2 "mug:true"}}
```

MODIFYING WITH `with`

- `with` is like a mixin in languages like Ruby
- Provides a way to set a scope from a value, like `.Values.favorites`
- May seem similar to `import static` in Java

```
{{ with PIPELINE }}
  # restricted scope
{{ end }}
```

USING `with`

Using `with`, we can reference `.drink`, `.food`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  {{- end }}
```

EXITING THE `with` APPROPRIATELY

- `Release.Name` is not in `.Values.favorite`
- This will return an error

```
{{- with .Values.favorite }}
drink: {{ .drink | default "tea" | quote }}
food: {{ .food | upper | quote }}
release: {{ .Release.Name }}
{{- end }}
```

CORRECTING THE `with` ERROR

```
{{- with .Values.favorite }}
drink: {{ .drink | default "tea" | quote }}
food: {{ .food | upper | quote }}
{{- end }}
release: {{ .Release.Name }}
```

LOOPING WITH `Range`

Given the following, we would like to "iterate" all the pizza toppings

```
favorite:
  drink: coffee
  food: pizza
pizzaToppings:
  - mushrooms
  - cheese
  - peppers
  - onions
```

USING `Range`

- `range` will iterate over the `pizzaToppings`
- `.` will represent each value
- `{{ . | title | quote }}` will send each value into `title` case and then `quote`

```
toppings: |-
    {{- range .Values.pizzaToppings }}
    - {{ . | title | quote }}
    {{- end }}
```

RESULT OF USING `Range`

```
toppings: |-
  - "Mushrooms"
  - "Cheese"
  - "Peppers"
  - "Onions"
```

⚠ Notice `|` – is a YAML Multiline String

VARIABLES

- Temporary Variables are available
- Using a previous example that failed because `.Release.Name` is not in `.Values.favorite`
- A workaround is to introduce a variable

```
{{- with .Values.favorite }}
drink: {{ .drink | default "tea" | quote }}
food: {{ .food | upper | quote }}
release: {{ .Release.Name }}
{{- end }}
```

USING `Variables`

- Uses the form `$name`
- Assigned using a special operator: `:=`
- Notice the variable is bound *before* the `with` block

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- $relname := .Release.Name -}}
  {{- with .Values.favorite }}
  drink: {{ .drink | default "tea" | quote }}
  food: {{ .food | upper | quote }}
  release: {{ $relname }}
  {{- end }}
```

AFTER RUNNING THE EXAMPLE

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: viable-badger-configmap
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "PIZZA"
  release: viable-badger
```

USING VARIABLES WITH `range`

- Variables can be used as an index and value variable for lists
- Below, `index` and `topping` are used to display element number, `:`, and the element

```
toppings: |-
    {{- range $index, $topping := .Values.pizzaToppings }}
      {{ $index }}: {{ $topping }}
    {{- end }}
```

USING `range` FOR KEY VALUE PAIRS

- Since `./Values.favorite` are a list of key value pairs, we can iterate
- $key will be `drink`, $val will be `coffee`, etc.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  myvalue: "Hello World"
  {{- range $key, $val := .Values.favorite }}
  {{ $key }}: {{ $val | quote }}
  {{- end}}
```

RESULT OF USING `range` FOR KEY VALUE PAIRS

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: eager-rabbit-configmap
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "pizza"
```

GLOBAL VARIABLES

- `$` represents the global variable
- This is useful in a `range` loop you need a value by navigating from the top

```
{{- range .Values.tlsSecrets }}
apiVersion: v1
kind: Secret
metadata:
  name: {{ .name }}
  labels:
    # Many helm templates would use <code>.</code> below, but that will
not work,
    # however <code>$</code> will work here
    app.kubernetes.io/name: {{ template "fullname" $ }}
    # I cannot reference .Chart.Name, but I can do $.Chart.Name
    helm.sh/chart: "{{ $.Chart.Name }}-{{ $.Chart.Version }}"
    app.kubernetes.io/instance: "{{ $.Release.Name }}"
    # Value from appVersion in Chart.yaml
```

NAMED TEMPLATES

- A named template (sometimes called a partial or a subtemplate)
- A template defined inside of a file, and given a name
- Declare and manage `define`, `template`, `include` and `block`

PARTIALS AND _ FILES

- We can create named embedded templates, that can be accessed by name elsewhere
- Some Conventions:
  - Most files in *templates/* are treated as if they contain Kubernetes manifests
  - The *NOTES.txt* is one exception
  - Files whose name begins with an underscore (_) are assumed to not have a manifest inside.

DECLARING A TEMPLATE

The define action allows us to create a named template inside of a template file.

```
{{ define "MY.NAME" }}
  # body of template here
{{ end }}
```

EXAMPLE OF A TEMPLATE

```
{{- define "mychart.labels" }}
  labels:
    generator: helm
    date: {{ now | htmlDate }}
{{- end }}
```

EMBEDDING A TEMPLATE

```
{{- define "mychart.labels" }}
  labels:
    generator: helm
    date: {{ now | htmlDate }}
{{- end }}
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  {{- template "mychart.labels" }}
data:
  myvalue: "Hello World"
  {{- range $key, $val := .Values.favorite }}
  {{ $key }}: {{ $val | quote }}
```

TEMPLATE NAMES ARE GLOBAL

- Template names are global.
- If you declare two templates with the same name, whichever one is loaded last will be the one used.
- Convention is to prefix each defined template with name of the chart
- Using the specific chart name as a prefix we can avoid any conflicts that may arise due to two different charts that implement templates of the same name.

```
{{ define "mychart.labels" }}
```

SETTING SCOPE OF A TEMPLATE

The following will add `Chart Name` and `Version` albeit incorrectly

```
{{/* Generate basic labels */}}
{{- define "mychart.labels" }}
  labels:
    generator: helm
    date: {{ now | htmlDate }}
    chart: {{ .Chart.Name }}
    version: {{ .Chart.Version }}
{{- end }}
```

WHAT IT RENDERS

- Notice that `chart` and `version` are missing
- They were not in the scope

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: moldy-jaguar-configmap
  labels:
    generator: helm
    date: 2016-11-02
    chart:
    version:
```

PASSING IN THE SCOPE

Instead of:

```
{{ define "mychart.labels" }}
```

We will include . :

```
{{ define "mychart.labels" . }}
```

RESULTING IN THE FOLLOWING

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  {{- template "mychart.labels" .}}
data:
  myvalue: "Hello World"
  {{- range $key, $val := .Values.favorite }}
  {{ $key }}: {{ $val | quote }}
  {{- end }}
```

`include` FUNCTION

If provided the following:

```
{{- define "mychart.app" -}}
app_name: {{ .Chart.Name }}
app_version: "{{ .Chart.Version }}+{{ .Release.Time.Seconds }}"
{{- end -}}
```

CALLING `template`

- Substituting it into template using `template`
- Notice the location of the `template`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
    {{ template "mychart.app" .}}
data:
  myvalue: "Hello World"
  {{- range $key, $val := .Values.favorite }}
  {{ $key }}: {{ $val | quote }}
  {{- end }}
{{ template "mychart.app" . }}
```

THE INCORRECT OUTPUT

- Notice the location of `app_name` and `app_version` in both `labels` and `data`
- `template` will organize in it's own text alignment
- `template` is an *action*, not a function

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: measly-whippet-configmap
  labels:
    app_name: mychart
app_version: "0.1.0+1478129847"
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "pizza"
app_name: mychart
app_version: "0.1.0+1478129847"
```

`include` IS A FUNCTION

- `include` is a function, instead of an action
- Since it is a function, it is available to be chained to other functions

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
    {{- include "mychart.app" . | nindent 4 }}
data:
  myvalue: "Hello World"
  {{- range $key, $val := .Values.favorite }}
  {{ $key }}: {{ $val | quote }}
  {{- end }}
  {{- include "mychart.app" . | nindent 2 }}
```

WHAT IT RENDERS

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: edgy-mole-configmap
  labels:
    app_name: mychart
    app_version: "0.1.0+1478129987"
data:
  myvalue: "Hello World"
  drink: "coffee"
  food: "pizza"
  app_name: mychart
  app_version: "0.1.0+1478129987"
```

Prefer `include` over `template`

ACCESSING FILES

- You can also bring in files into Helm
- Files must be less than 1M
- Files in *template/* directory cannot be accessed
- Files excluded in *.helmignore* cannot be accessed

CREATING STANDARD FILES

config1.txt

```
message = "Hello from config 1"
```

config2.txt:

```
message = "This is config 2"
```

config3.txt

```
message = "Goodbye from config 3"
```

USING `range` TO INCLUDE THE FILES

Here, we will use a `range list` to get all the values in range and print them

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  {{- $files := .Files }}
  {{- range list "config1.txt" "config2.txt" "config3.txt" }}
  {{ . }}: |-
    {{ $files.Get . }}
  {{- end }}
```

RESULTS OF `Files` AND `range`

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: quieting-giraf-configmap
data:
  config1.txt: |-
    message = "Hello from config 1"

  config2.txt: |-
    message = "This is config 2"

  config3.txt: |-
    message = "Goodbye from config 3"
```

PATH HELPERS

- Helm imports many of the functions from Go's `path`
- They use the same names, but instead of capital letters it uses small letters
- `Base` → `base`

Imported Functions

- Base
- Dir
- Ext
- IsAbs
- Clean

GLOB PATTERNS

- `Glob`
  - Allows you to filter files in glob patterns
  - Returns a `Files` type, so you may call any of the `Files` methods on the returned object

GIVEN THE FILES

```
foo/:
  foo.txt foo.yaml

bar/:
  bar.go bar.conf baz.yaml
```

USING `Glob`:

- Notice the `$path` and `$byte` are included with `Glob`
- We can enrich the content of `$path` and `$byte` with Go functions

```
{{ $root := . }}
{{ range $path, $bytes := .Files.Glob "foo/*" }}
{{ base $path }}: '{{ $root.Files.Get $path | b64enc }}'
{{ end }}
```

CONFIGMAP AND SECRETS UTILITY

- Helps with configmaps and secrets
- Helm Provides some utilities
- Useful to use these methods in conjunction with the `Glob` method.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: conf
data:
  {{- (.Files.Glob "foo/*").AsConfig | nindent 2 }}
---
apiVersion: v1
kind: Secret
metadata:
  name: very-secret
type: Opaque
data:
  {{- (.Files.Glob "bar/*").AsSecrets | nindent 2 }}
```

ENCODING

You can apply Base64 encoding using `b64enc`

```
apiVersion: v1
kind: Secret
metadata:
  name: {{ .Release.Name }}-secret
type: Opaque
data:
  token: |-
    {{ .Files.Get "config1.toml" | b64enc }}
```

LINES

To iterate each line in a template, you can use `.Files.Lines`

```
data:
  some-file.txt: {{ range .Files.Lines "foo/bar.txt" }}
    {{ . }}{{ end }}
```

NOTES.TXT

- Provides help to end users
- Is in plain text
- Has template functions available

```
Thank you for installing {{ .Chart.Name }}.

Your release is named {{ .Release.Name }}.

To learn more about the release, try:

  $ helm status {{ .Release.Name }}
  $ helm get {{ .Release.Name }}
```

SUBCHARTS

- Charts can have dependencies called subcharts
- They have their own values and templates

SUBCHART DETAILS

- Subcharts are considered "stand-alone"
- Subchart can never
  - Explicitly depend on its parent chart
- Parent Chart can override values of subchart
- Helm has a concept of global values that can be accessed by all

CREATING A SUBCHART

- Change into a directory that is already a chart
- Create a subchart in that directory using `helm create mysubchart`
- Clean out templates if you desire to define your own

```
% cd mychart/charts
% helm create mysubchart
% rm -rf mysubchart/templates/*.*
```

ADDING VALUES AND TEMPLATE TO SUBCHART

In the `subchart`, there is a *Values.yaml* file where we can include:

```
dessert: cake
```

USING `Values` FROM SUBCHART'S OWN `values.yaml`

We can then create a ConfigMap template in
mychart/charts/mysubchart/templates/configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-cfgmap2
data:
  dessert: {{ .Values.dessert }}
```

WHAT IT RENDERS

Because the charts works as is, it becomes the following:

```
# Source: mysubchart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: newbie-elk-cfgmap2
data:
  dessert: cake
```

OVERRIDING THE DEFAULTS

- `mychart` is now the parent of `mysubchart`
- Since this relationship exists, we can specify configuration at `mychart` and pushed to `mysubchart`
- In other words *mychart/values.yaml*

```
favorite:
  drink: coffee
  food: pizza
pizzaToppings:
  - mushrooms
  - cheese
  - peppers
  - onions

mysubchart:
  dessert: ice cream
```

The last two lines will have information for the subchart

RENDERING THE RESULTS

- Remember that our favorite dessert was `cake`
- The parent now has an override defined in `configmap.yaml`
- The following shows the results of the ConfigMap defined in `mysubchart`

```
# Source: mychart/charts/mysubchart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: unhinged-bee-cfgmap2
data:
  dessert: ice cream
```

GLOBAL CHART VALUES

- Global values are values that can be accessed from any chart or subchart by exactly the same name
- Globals require explicit declaration
- You can’t use an existing non-global as if it were a global

CREATING A GLOBAL SECTION IN VALUES

- Both of the following should be able to access the value `global.salad` as {{ .Values.global.salad}}.
  - *mychart/templates/configmap.yaml*
  - *mychart/charts/mysubchart/templates/configmap.yaml*

```
favorite:
  drink: coffee
  food: pizza
pizzaToppings:
  - mushrooms
  - cheese
  - peppers
  - onions

mysubchart:
  dessert: ice cream

global:
  salad: caesar
```

CONFIGURING PARENT CHART'S CONFIG MAP

In *mychart/templates/configmap.yaml*, notice the `salad` value

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
data:
  salad: {{ .Values.global.salad }}
```

CONFIGURING SUBCHARTS' CONFIG MAP

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-cfgmap2
data:
  dessert: {{ .Values.dessert }}
  salad: {{ .Values.global.salad }}
```

THE PARENT CHART'S RESULT

```
# Source: mychart/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: silly-snake-configmap
data:
  salad: caesar
```

THE CHILD CHART'S RESULT

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: silly-snake-cfgmap2
data:
  dessert: ice cream
  salad: caesar
```

SHARING TEMPLATES WITH SUBCHARTS

- All templates to include are stored by unique name
- Therefore they are possible to include in any chart or subchart
- Be sure to favor `include` over `template`

.HELMIGNORE FILE

- Used to specify files you don't want to include in your helm chart.
- If the file exists, the `helm package` command will ignore all the files that match the pattern specified in the *.helmignore* file while packaging your application.
- This can help in avoiding unnecessary or sensitive files or directories from being added in your helm chart.
- The *.helmignore* file supports Unix shell glob matching, relative path matching, and negation (prefixed with !).
- Only one pattern per line is considered.

```
# comment
.git
*/temp*
*/*/temp*
temp?
```

DEBUGGING TEMPLATES

- While things may look well on the client
- When it is sent to the Kubernetes API, it may reject
- To help debug, you can use the following:
  - `helm lint` is your go-to tool for verifying that your chart follows best practices
  - `helm install --dry-run --debug` render your templates and eyeball
  - `helm get manifest` see what templates have already been installed on the server

LAB 3: TEMPLATIZING A CHART

Open your lab book, *lab_book.html*, and do "Lab 3: Templatizing"

PLUGINS

- A plugin is a tool that can be accessed through the helm CLI
- Not part of the built-in Helm
- Wide array of a list of plugins
- Also available at Github

PLUGIN OVERVIEW

- Integrate with Helm CLI
- Provide a way to extend the core of Helm
- Without requiring every new feature to be written in Go

PLUGIN FEATURES

- Can be added and removed from a Helm installation without impacting the core Helm tool
- Written in any programming language
- Will show up with `helm help`

INSTALLING A PLUGIN

- You can pass in a path to a plugin, either
  - Local file system
  - Url of a remote VCS repo
- Plugins get cloned in `$(helm home)/plugins`

```
% helm plugin install <path|url>
```

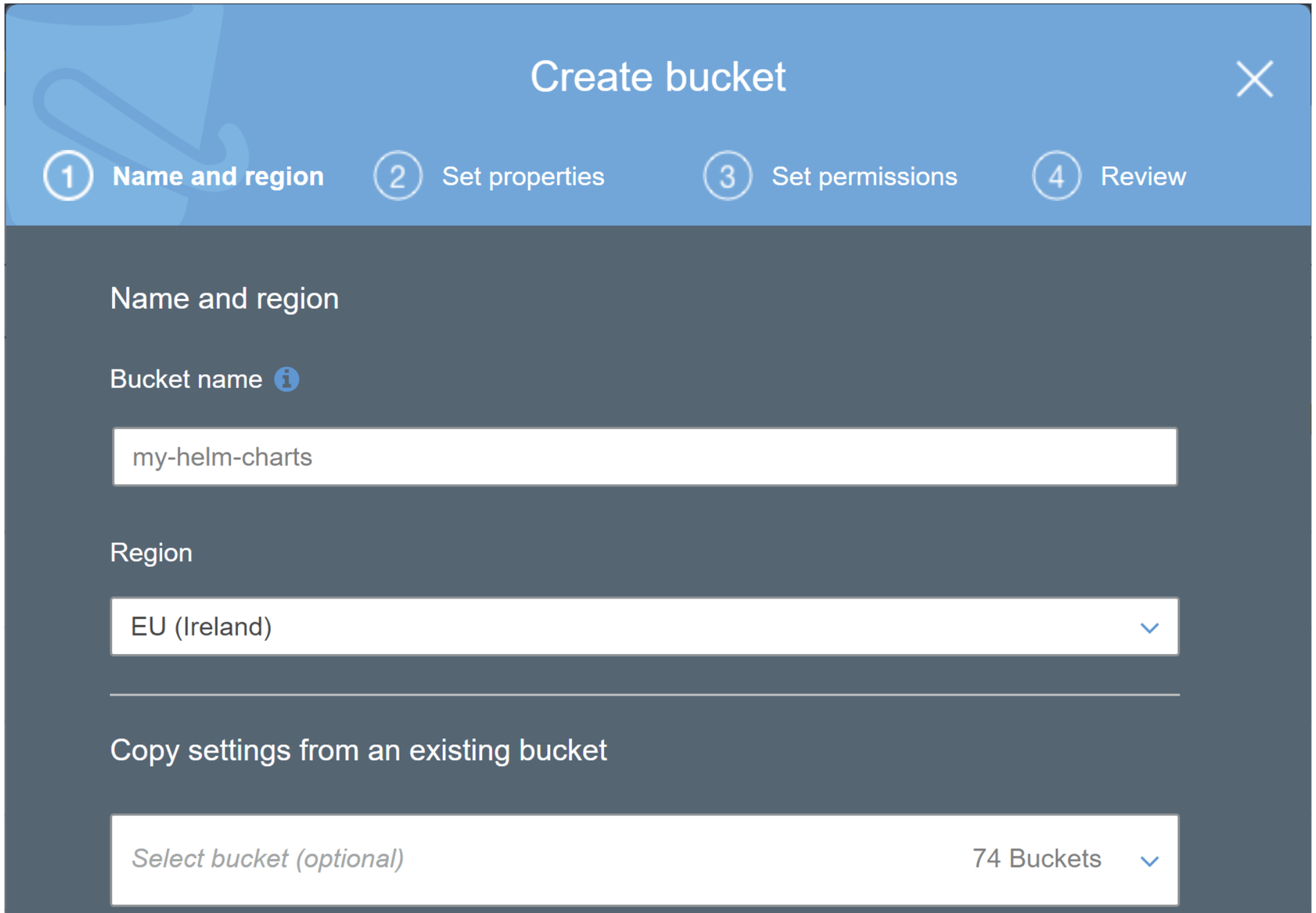
INSTALLING VIA TARBALL

- If you have a plugin tar distribution, simply untar the plugin into the $(helm home)/plugins directory.
- You can also install tarball plugins directly from url by issuing helm plugin install http://domain/path/to/plugin.tar.gz

EXAMPLE PLUGIN: S3

- An Example Plugin is the S3 Plugin[https://github.com/hypnoglow/helm-s3]
- It can use AWS S3 as a chart repository

CREATING THE AWS BUCKET



INSTALLING THE S3 PLUGIN

```
% helm plugin install https://github.com/hypnoglow/helm-s3.git
```

INITIALIZING THE S3 PLUGIN

- We initialize the plugin using `S3 init`

```
% helm S3 init s3://my-helm-charts/charts
```

- After we can then introduce the URL as a repository

```
% helm repo add my-charts s3://my-helm-charts/charts
```

DISPLAY ALL THE REPOS

```
% helm repo list
NAME                    URL
stable                  https://kubernetes-charts.storage.googleapis.com
local                   http://127.0.0.1:8879/charts
my-charts               s3://my-helm-charts/charts
```

CREATING A SIMPLE HELM CHART

```
% helm create test-chart
Creating test-chart
# Remove the initial cruft
$ rm -rf test-chart/templates/*.*
# Create a ConfigMap template at test-chart/templates/configmap.yaml
$ cat > test-chart/templates/configmap.yaml << EOL
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-chart-configmap
data:
  myvalue: "Hello World"
EOL
```

INSTALL THE CHART

```
% helm install ./test-chart
NAME:   zeroed-armadillo
LAST DEPLOYED: Fri Feb  9 17:10:38 2018
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME                 DATA  AGE
test-chart-configmap  1     0s
```

REMOVE IT AGAIN

- Since we wanted to ensure that it works
- We can remove it from the K8S cluster

```
# --purge removes the release from the "store" completely
% helm delete --purge zeroed-armadillo
release "zeroed-armadillo" deleted
```

UPLOAD THE TEST CHART

```
% helm package ./test-chart
Successfully packaged chart and saved it to: ~/test-chart-0.1.0.tgz
```

ONCE COMPLETE, SEND IT TO S3

- Specify the `tgz` file
- Specify the name of the repository
- Ensure that you version with SemVer2
- Ensure that you treat each release as immutable

```
% helm s3 push ./test-chart-0.1.0.tgz my-charts
```

TESTING HELM

UNIT TESTING HELM

https://github.com/lrills/helm-unittest

- Write test file in pure YAML
- Render locally with no need of tiller
- Create nothing on your cluster
- Define values and release options
- Snapshot testing

INSTALL AS PLUGIN

```
% helm plugin install https://github.com/lrills/helm-unittest
```

BUILDING A TEST

- Create a `tests` folder
- Add `tests` directory to your chart
- Add file, for example, to *$YOUR_CHART/tests/deployment_test.yaml*

EXAMPLE TEST

```
suite: test deployment
templates:
  - deployment.yaml
tests:
  - it: should work
    set:
      image.tag: latest
    asserts:
      - isKind:
          of: Deployment
      - matchRegex:
          path: metadata.name
          pattern: -my-chart$
      - equal:
```

RUNNING THE TEST

To run the tests

```
% helm unittest $YOUR_CHART
```

If you wish to use a different folder, you can use `-f`, or `--file`

```
% helm unittest -f 'my-tests/*.yaml' -f 'more-tests/*.yaml' my-chart
```

SNAPSHOT TESTING

- You can take a snapshot of the entire manifest or portions with a snapshot
- This avoids the need assert every single element
- This will match an element against the snapshot taken the previous time
- Snapshots are located in the `*snapshot/*test.yaml.snap*` directory
- Run `-u` or `--update-snapshot` to update the snapshot

SNAPSHOT EXAMPLE TEST

```
templates:
  - deployment.yaml
tests:
  - it: pod spec should match snapshot
    asserts:
      - matchSnapshot:
          path: spec.template.spec
# or you can snapshot the whole manifest
- it: manifest should match snapshot
  asserts:
    - matchSnapshot: {}
```

TEST SUITE DETAIL

```
suite: test deploy and service
templates:
  - deployment.yaml
  - service.yaml
tests:
  - it: should test something
    ...
```

- `suite`: *string, optional*. The suite name to show on test result output.
- `templates`: *array of string, recommended*. The template files scope to test in this suite, only the ones specified here is rendered during testing. If omitted, all template files are rendered. File suffixed with .tpl is added automatically, you don't need to add them again.
- `tests`: *array of test job, required*. Where you define your test jobs to run, check Test Job.

TEST JOB

The test job is the base unit of testing. Your chart is rendered each time a test job run, and validated with assertions defined in the test. You can setup your values used to render the chart in the test job with external values files or directly in the test job definition.

TEST DETAILS

```
tests:
  - it: should pass
    values:
      - ./values/staging.yaml
    set:
      image.pullPolicy: Always
      resources:
        limits:
          memory: 128Mi
release:
  name: my-release
  namespace:
  revision: 9
  isUpgrade: true
```

- `it` - Define the name of the test with TDD style or any message you like.
- `values` - The values files used to renders the chart, like *Values.yaml*
- `set` - Set the values directly in suite file, analogous to `--set`

VARIOUS ASSERTIONS

- Equality: `equal`, `notEqual`
- Regex: `matchRegex`, `notRegexMatch`
- Contains: `contains`, `notContains`
- Null: `isNull`, `isNotNull`
- Empty: `isEmpty`, `isNotEmpty`
- APIVersion: `apiVersion` of Manifest
- Number of Documents: `hasDocuments` Expected number of documents rendered
- Match Snapshot: Path that should not change from one run to another

INTEGRATION TESTING

```
% helm test [RELEASE] [flags]
```

LAB 4: TESTING

Open your lab book, *lab_book.html*, and do "Lab 4: Testing"

SECURITY

- Tiller authentication model uses client-side SSL certificates
- Tiller itself verifies these certificates using a certificate authority
- Client also verifies Tiller's identity by certificate authority

⚠ Tiller requires that the client certificate be validated by its CA

GENERATE A CERTIFICATE AUTHORITY

This will generate both secret key and CA

```
% openssl genrsa -out ./ca.key.pem 4096
% openssl req -key ca.key.pem -new -x509 -days 7300 -sha256 -out
ca.cert.pem -extensions v3_ca
```

GENERATING THE CERTIFICATE FOR TILLER

- Generate certificate is for Tiller
- You will want one of these per tiller host that you run

```
% openssl genrsa -out ./tiller.key.pem 4096
```

GENERATING THE CERTIFICATE FOR EACH CLIENT

For each client that you wish to run

```
% openssl genrsa -out ./helm.key.pem 4096
```

CREATE CERTIFICATES REQUESTS

```
% openssl req -key tiller.key.pem -new -sha256 -out tiller.csr.pem
% openssl req -key helm.key.pem -new -sha256 -out helm.csr.pem
```

SELF SIGN EACH OF THE CERTIFICATES

```
% openssl x509 -req -CA ca.cert.pem -CAkey ca.key.pem -CAcreateserial -
in tiller.csr.pem -out tiller.cert.pem -days 365
% openssl x509 -req -CA ca.cert.pem -CAkey ca.key.pem -CAcreateserial -
in helm.csr.pem -out helm.cert.pem  -days 365
```

TAKING STOCK OF THE FILES

```
# The CA. Make sure the key is kept secret.
ca.cert.pem
ca.key.pem
# The Helm client files
helm.cert.pem
helm.key.pem
# The Tiller server files.
tiller.cert.pem
tiller.key.pem
```

INSTALLING TLS ON TILLER

```
% helm init --dry-run --debug --tiller-tls --tiller-tls-cert
./tiller.cert.pem --tiller-tls-key ./tiller.key.pem --tiller-tls-verify
--tls-ca-cert ca.cert.pem
```

INSTALLING TLS ON CLIENT

Manually running the TLS certs

```
% helm ls --tls --tls-ca-cert ca.cert.pem --tls-cert helm.cert.pem --
tls-key helm.key.pem
```

Adding the certificates to the client in the helm home

```
% cp ca.cert.pem $(helm home)/ca.pem
% cp helm.cert.pem $(helm home)/cert.pem
% cp helm.key.pem $(helm home)/key.pem
```

Then run `helm ls --tls`

COMMANDS

- Commands in Helm
- For a Complete List visit the website for details

helm completion

Generate autocompletions script for the specified shell (bash or zsh)

`helm inspect`

- This command inspects a chart and displays information.
- It takes a chart reference (‘stable/drupal’), a full path to a directory or packaged chart, or a URL.
- Inspect prints the contents of the *Chart.yaml* file and the *values.yaml* file.

Also available:

- `helm inspect chart` - shows inspect chart
- `helm inspect readme` - shows inspect readme
- `helm inspect values` - shows inspect values

`helm list`

- This command lists all of the releases.
- By default, it lists only releases that are deployed or failed. Flags like `-deleted` and `-all` will alter this behavior. Such flags can be combined: `-deleted -failed`.
- By default, items are sorted alphabetically. Use the `-d` flag to sort by release date.
- If an argument is provided, it will be treated as a filter. Filters are regular expressions (Perl compatible) that are applied to the list of releases. Only items that match the filter will be returned.

`helm package`

- Package a chart directory into a chart archive
- This command packages a chart into a versioned chart archive file.
- If a path is given, this will look at that path for a chart (which must contain a Chart.yaml file) and then package that directory.

`helm plugin`

- Manage helm plugins, including:
  - `helm plugin install` - Install one or more Helm plugins
  - `helm plugin list` - List installed Helm plugins
  - `helm plugin remove` - Remove one or more Helm plugins
  - `helm plugin update` - Update one or more Helm plugins

`helm repo`

- Manage repositories to pull charts and plugins
  - `helm repo add` - Add a chart repository
  - `helm repo index` - Generate an index file given a directory containing packaged charts
  - `helm repo list` - List chart repositories
  - `helm repo remove` - Remove a chart repository
  - `helm repo update` - Update information of available charts locally from chart repositories

`helm upgrade`

- Upgrades a release to a specified version of a chart and/or updates chart values.
- You can use `--set` for ad-hoc changes
- Include the release name: `helm upgrade [RELEASE] [CHART] [flags]`
- All upgrades are available to rollback

`helm rollback`

- Rolls back a release to a previous revision.
- The first argument of the rollback command is the name of a release
- Second is a revision (version) number.
- To see revision numbers, run ‘helm history RELEASE’
- If you’d like to rollback to the previous release use ‘helm rollback [RELEASE] 0’.

helm version

- Display the version

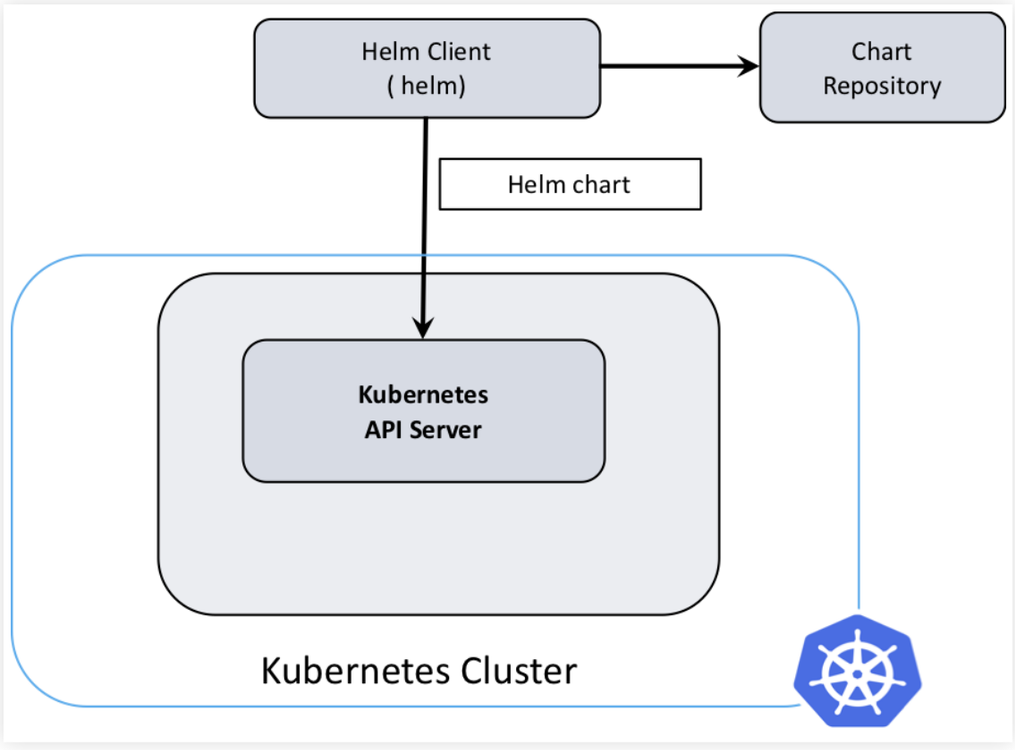
LAB 5: MORE CLIENT COMMANDS

Open your lab book, *lab_book.html*, and do "Lab 5: More Client Commands"

WHAT IS NEW IN HELM V3?

CLIENT-ONLY ARCHITECTURE

- Helm 3 doesn't require Tiller on K8s, unlike Helm 2
- The main benefit of removing Tiller is that security is now on a per-user basis



XDG BASE DIRECTORY SPECIFICATION

- Helm 3 doesn't use `.helm` directory
- Uses XDG Standard, Platform Independent, Directories
- XDG Standards: https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html

NO NEED TO `helm init`

- `helm init` is removed, no need to install
- Helm state is created when required

MORE SPECIFICS

Source: https://developer.ibm.com/technologies/containers/blogs/kubernetes-helm-3/

WHERE TO GET MORE INFORMATION

https://helm.sh/blog/2019-10-22-helm-2150-released