Version 2.1
06/15/20

Prerequisites

- Modern system with at least 8G of memory and 20G free storage
- VirtualBox installed and running
  - http://www.virtualbox.org
- Virtual machine (.ova file) installed in VirtualBox
  - https://www.dropbox.com/s/9z62cdvs0aofvdn/helm-intro.ova?dl=0
- Workshop docs are in https://github.com/brentlaster/safaridocs
- Setup doc is at
  - https://github.com/brentlaster/safaridocs/blob/master/helm-fun-setup.pdf
- Labs doc for workshop
  - https://github.com/brentlaster/safaridocs/blob/master/helm-fun-labs.pdf

Helm Fundamentals

Brent Laster

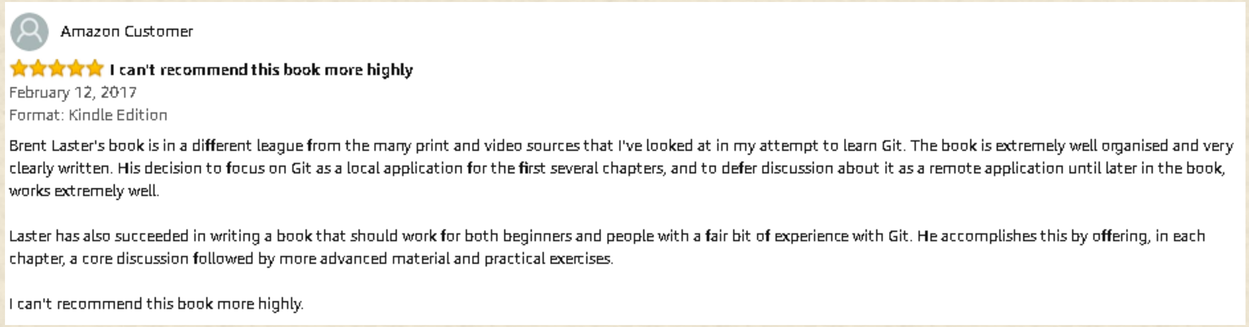
About me

- R&D Director
- Global trainer – training (Git, Jenkins, Gradle, Gerriit, Continuous Pipelines)
- Author -
  - OpenSource.com
  - Professional Git book
  - Jenkins 2 – Up and Running book
  - Continuous Integration vs. Continuous Delivery vs. Continuous Deployment mini-book on Safari
- https://www.linkedin.com/in/brentlaster
- @BrentCLaster

Book – Professional Git

- Extensive Git reference, explanations, and examples
- First part for non-technical
- Beginner and advanced reference
- Hands-on labs

Professional Git 1st Edition
by Brent Laster (Author)
★★★★★ 7 customer reviews
Look inside

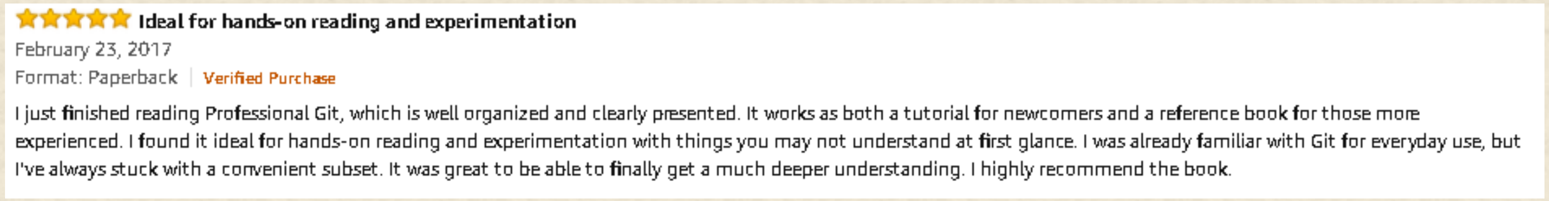

Amazon Customer

★★★★★ **I can't recommend this book more highly**
February 12, 2017
Format: Kindle Edition

Brent Laster's book is in a different league from the many print and video sources that I've looked at in my attempt to learn Git. The book is extremely well organised and very clearly written. His decision to focus on Git as a local application for the first several chapters, and to defer discussion about it as a remote application until later in the book, works extremely well.

Laster has also succeeded in writing a book that should work for both beginners and people with a fair bit of experience with Git. He accomplishes this by offering, in each chapter, a core discussion followed by more advanced material and practical exercises.

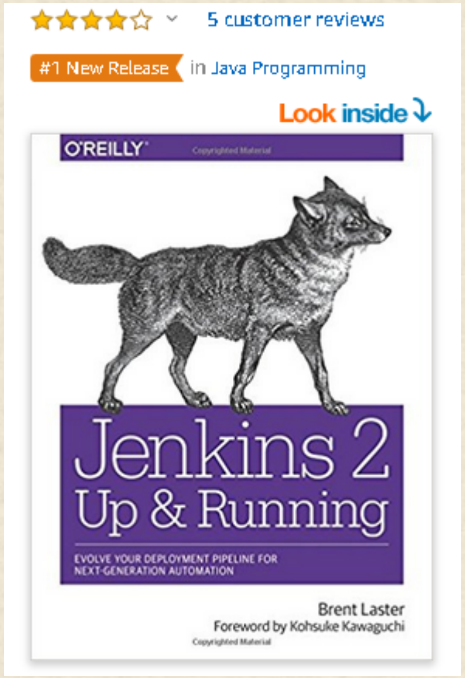
I can't recommend this book more highly.

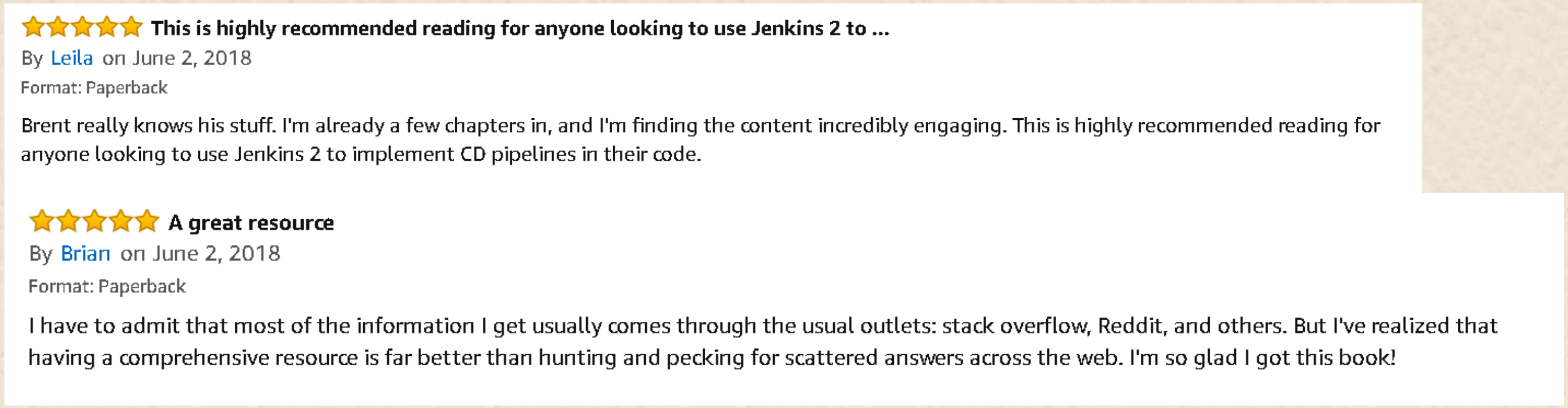

★★★★★ **Ideal for hands-on reading and experimentation**
February 23, 2017
Format: Paperback | Verified Purchase

I just finished reading Professional Git, which is well organized and clearly presented. It works as both a tutorial for newcomers and a reference book for those more experienced. I found it ideal for hands-on reading and experimentation with things you may not understand at first glance. I was already familiar with Git for everyday use, but I've always stuck with a convenient subset. It was great to be able to finally get a much deeper understanding. I highly recommend the book.

Jenkins 2 Book

- Jenkins 2 – Up and Running
- "It's an ideal book for those who are new to CI/CD, as well as those who have been using Jenkins for many years. This book will help you discover and rediscover Jenkins."
  By Kohsuke Kawaguchi, Creator of Jenkins


5 customer reviews
#1 New Release in Java Programming
Look inside
O'REILLY
Jenkins 2
Up & Running
EVOLVE YOUR DEPLOYMENT PIPELINE FOR NEXT-GENERATION AUTOMATION
Brent Laster
Foreword by Kohsuke Kawaguchi

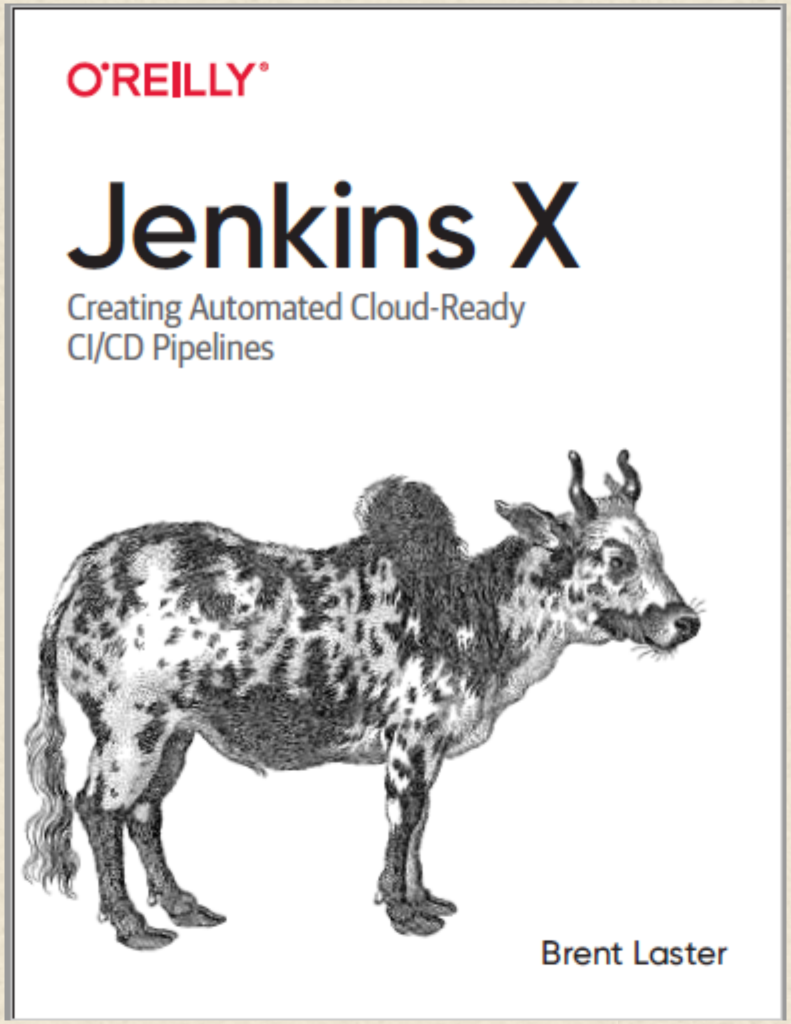


★★★★★ **This is highly recommended reading for anyone looking to use Jenkins 2 to ...**
By Leila on June 2, 2018
Format: Paperback
Brent really knows his stuff. I'm already a few chapters in, and I'm finding the content incredibly engaging. This is highly recommended reading for anyone looking to use Jenkins 2 to implement CD pipelines in their code.

★★★★★ **A great resource**
By Brian on June 2, 2018
Format: Paperback
I have to admit that most of the information I get usually comes through the usual outlets: stack overflow, Reddit, and others. But I've realized that having a comprehensive resource is far better than hunting and pecking for scattered answers across the web. I'm so glad I got this book!

Jenkins X Book

- In progress
- Early release chapters available on O'Reilly


O'REILLY®

Jenkins X

Creating Automated Cloud-Ready CI/CD Pipelines

Brent Laster

Agenda

- Kubernetes objects refresh
- What is Helm?
- Working with repositories
- Understanding charts
- Managing releases
- Templating and values
- Pipelines and functions

What is Kubernetes?

- A portable, extensible platform for managing containerized workloads and services (cluster orchestration system)
- Groups containers that make up an application into logical units for easy management and discovery.
- Goal is to provide a robust platform for running many containers.
- Allows automation of deployment, scaling, and managing containerized workloads.
- Kubernetes provides you with a framework to run distributed systems (of containers) resiliently.
- Takes care of
  - scaling requirements
  - failover
  - deployment patterns

kubernetes

So how do we think about this?

- Analogy: Datacenter for containers
  - If we think of images/containers as being like computers we stage and use
  - We can think of Kubernetes as being like a datacenter for those containers
  - Main jobs of datacenter
    - » Provide systems to service needs (regardless of the applications)
    - » Keep systems up and running
    - » Add more systems / remove systems depending on load
    - » Deal with systems that are having problems
    - » Deploy new systems when needed
    - Provide simple access to pools of systems
    - Etc..

K8s Quick Terminology

- Cluster - an HA set of computers coordinated by k8s to work as a unit.
- Pods – object that contains and manages one or more containers and any attached volumes
- Service – abstraction that groups together pods based on identifiers called labels (or other characteristic)
- Deployment – defines a stateless app with a set number of pod replicas (scaled instances)
- Ingress – resource that lets cluster applications be exposed to external traffic
- Namespace - a logical area that groups k8s items like pods

Data Center Analogy

- Functions: Uptime, scaling, redundancy…

  - Container in a pod ~ server in a rack

  - Pod ~ rack of servers

  - Deployment ~ multiple racks (replicas)

  - Service ~ central control / login server

  - Namespace ~ server room

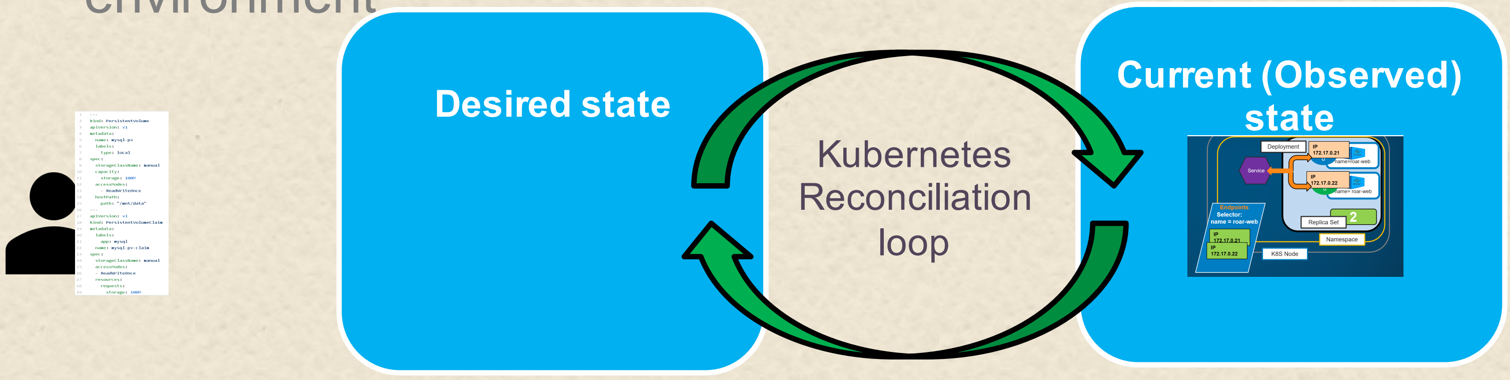
12

Understanding Kubernetes Objects

- To work with k8s objects, you use the k8s API
  - Kubectl command-line tool makes calls to API for you
  - Could also use k8s api client libraries
- K8s objects are persistent entities in the k8s system.
- K8s uses these entities to represent the state of the cluster.
  - They can describe:
    - » What application containers are running and on which nodes.
    - » Resources available to applications.
    - » Policies around how those applications behave.
- Kubernetes object is “record of intent”
  - After creation, k8s will work to ensure object exists
  - Creating an object declares what you want cluster workload to look like
  - Known as cluster’s “desired state”
- Declarative model vs. imperative model

Kubernetes is a Desired-State System

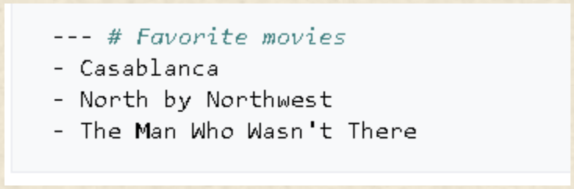
- User supplies desired state via declaring it in manifests
- Kubernetes works to balance the current state and the desired state
  - Desired state – what you want your production environment to be
  - Current (observed) state – current status of your production environment



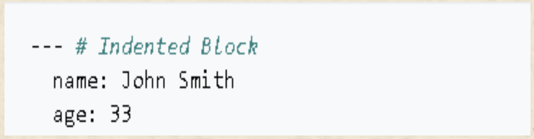
```yaml
---
kind: PersistentVolume
apiVersion: v1
metadata:
  name: mysql-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 100M
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  labels:
    app: mysql
  name: mysql-pv-claim
spec:
  storageClassName: manual
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 100M
```

YAML and K8S specifications

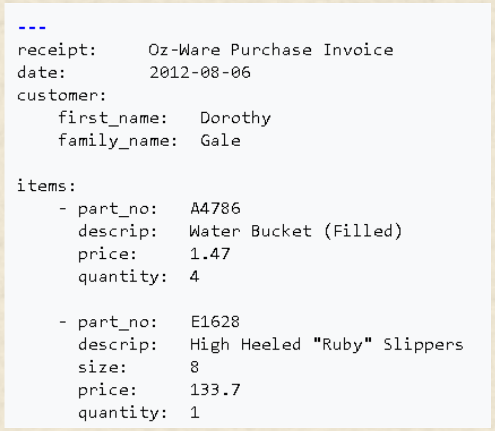
- YAML is a type of markup language to define Kubernetes specs for resources
- Stored in .yaml or .yml text file
- Kubectl apply can take such a file as input and update cluster based on specs
  - Turns yaml specs into resources/objects running in cluster
- Kubectl get –o yaml can be used to dump out spec and status as yaml from running object
- Conventional block format uses a hyphen + space to denote a new item in a list
- Keys are separated from values by a colon + space; indented blocks use indentation and newlines to separate key-value pairs
- Strings do not (generally) require quotation marks
- Data structure hierarchy is maintained by outline indentation



```
--- # Favorite movies
- Casablanca
- North by Northwest
- The Man Who Wasn't There
```



```
--- # Indented Block
  name: John Smith
  age: 33
```

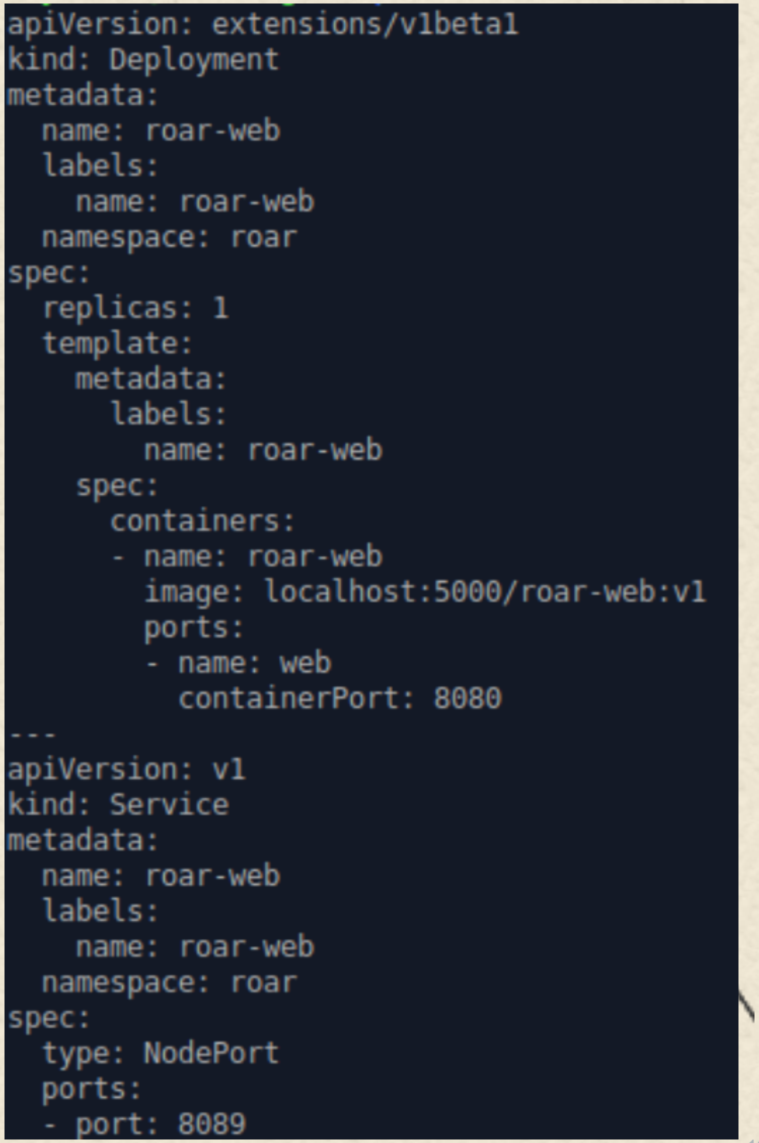


```
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    first_name:   Dorothy
    family_name:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     133.7
      quantity:  1
```

Defining a Kubernetes Object in text

- When creating an object in k8s, have to provide
  - object spec to describe desired
  - basic info, such as a name
- K8s API expects info as JSON in body of request
- But, usually provide it from YAML file
- Kubectl command line converts to JSON for you
- Required fields
  - apiVersion – which version of the k8s API is being used to create the object
  - kind - what kind of object to create
  - Metadata - data that helps uniquely identify the object, such as name, UID, namespace (optional)
  - Ojbect's spec
    - » Format different for every k8s object
    - » Contains nested fields specific to the object
    - » Can find details on specs in the API reference

https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/



```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: roar-web
    spec:
      containers:
      - name: roar-web
        image: localhost:5000/roar-web:v1
        ports:
        - name: web
          containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: roar-web
  labels:
    name: roar-web
  namespace: roar
spec:
  type: NodePort
  ports:
  - port: 8089
```

What is Helm?

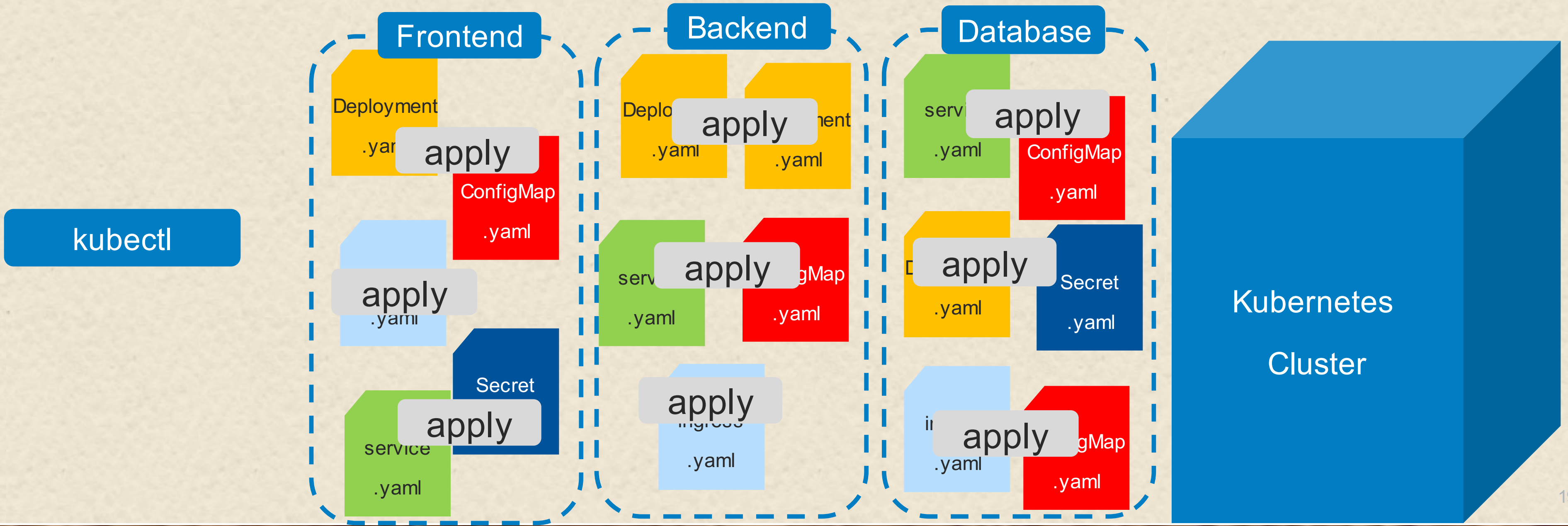
- Package Manager and Lifecycle Manager for K8s
  - Like yum, apt but for K8s
  - Bundles related manifests (such as deployment.yaml, sevice.yaml, etc.) into a “chart”
  - When installing chart, Helm creates a “release”
  - Lifecycle management
    - Create, Install, Upgrade, Rollback, Delete, Status, Versioning
  - Benefits
    - Templating, Repeatability, Reliability, Multiple Environment, Ease of collaboration

Installing Helm

- Location for install
  - https://github.com/Kubernetes/helm/releases
- Can install via
  - curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
- Checking version
  - helm version
- Current state of Helm (locally)
  - helm env
- Uses the cluster/host via config file
  - ~/.kube/config

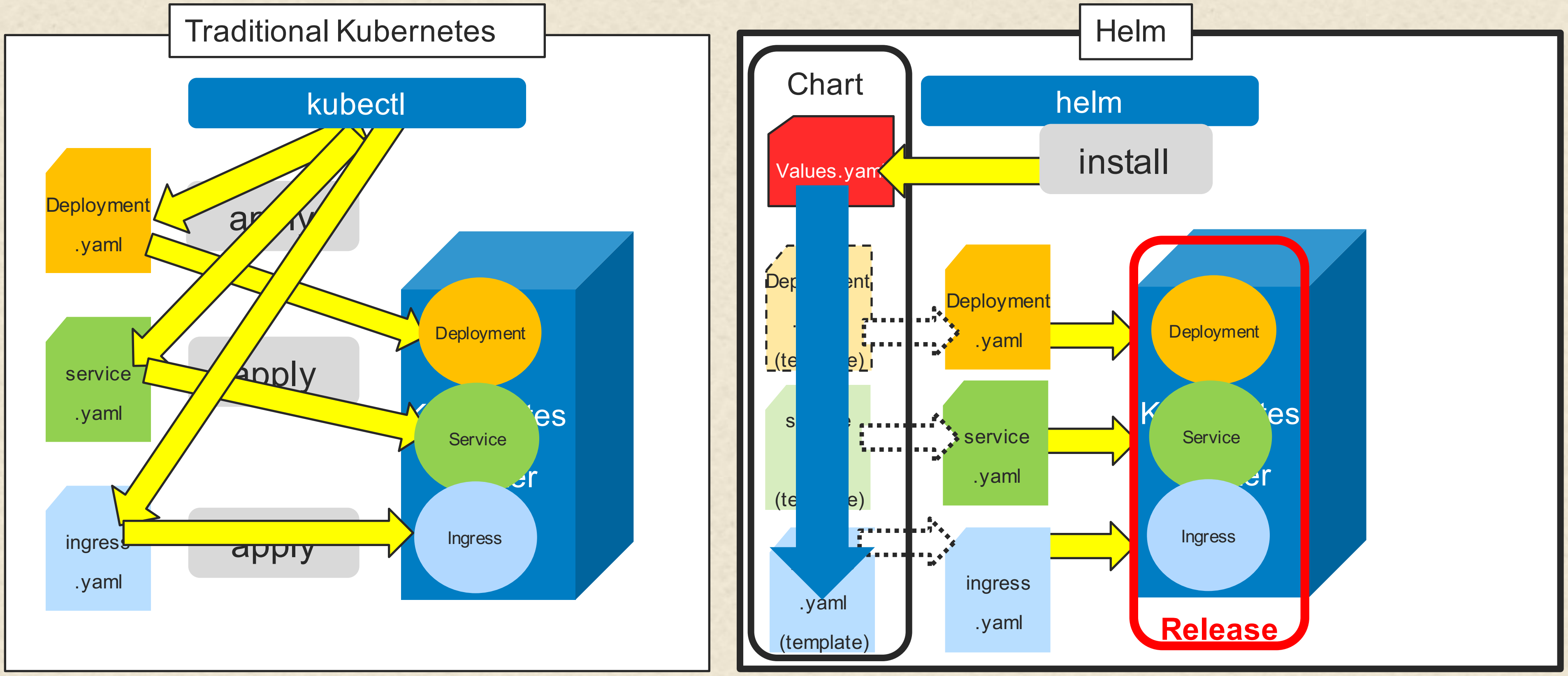
Why do we need something like this?

- Scale and complexity
- Microservice = Pod + Deployment+ ReplicationSet + Ingress + Service times # of microservices
- Duplication of values across objects
- Hard to override values (no parameterization)
- Managing lifecycle of all the objects is challenging



How does Helm simplify things?

- Traditional deployment in Kubernetes is done with kubectl across files into separately managed items
- Helm deploys units called charts as managed releases

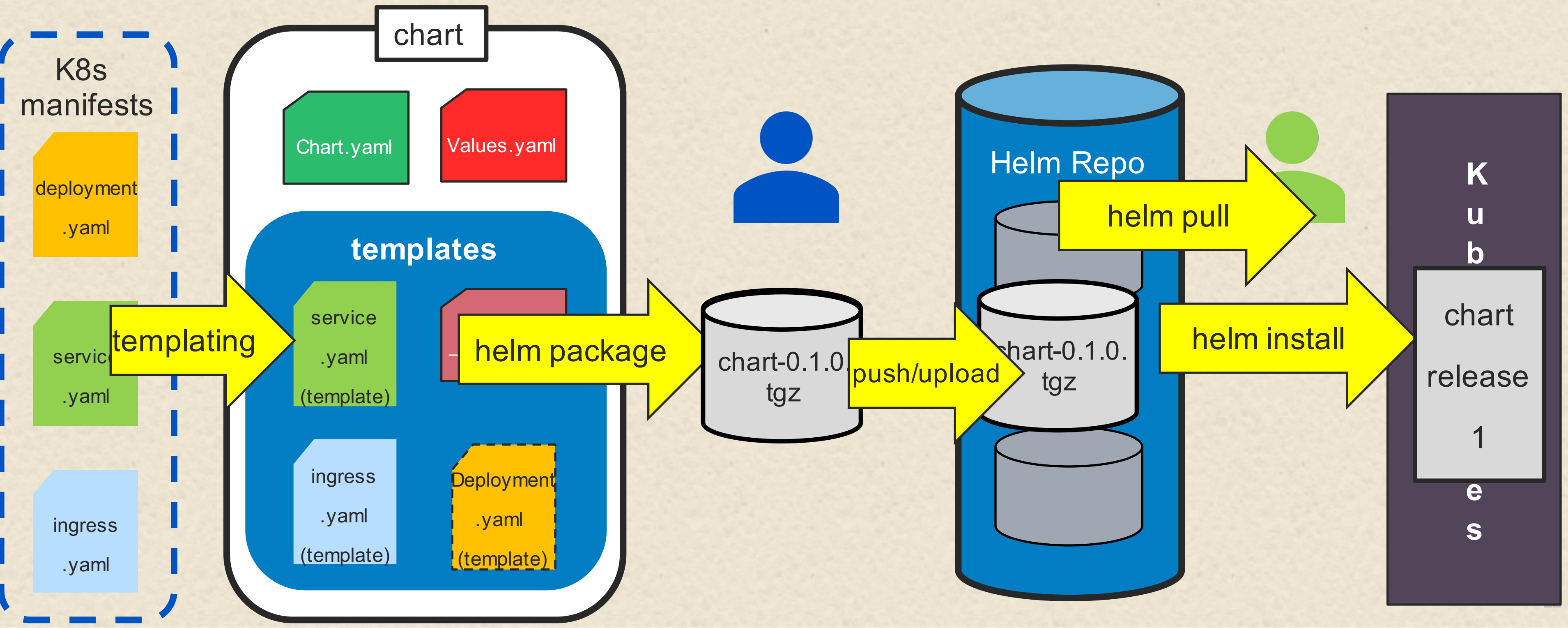


What are the advantages of using Helm?

- Saves having to deploy multiple individual manifests (for multiple Kubernetes objects)
- Allows for reuse via parameterizing (templates)
- Manages releases of Helm packages
- Simplified mechanism for finding and deploying popular software (packaged as Helm charts)
- Makes it easier to share your applications (via charts)

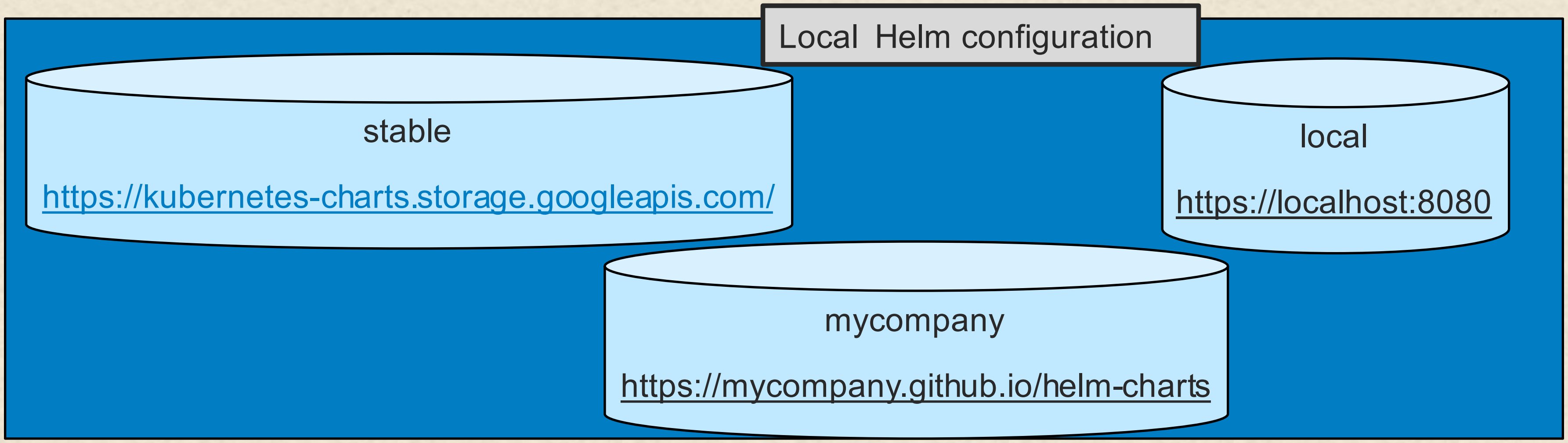
Helm as Package Manager

- Kubernetes files are "templated"
- Templates and related files are structured as a chart
- Charts are packaged
- Packages are stored in repos for easy use by others
- Charts can be pulled (downloaded) and optionally unpacked (untar)
- Charts are installed from repos (with values) as releases into Kubernetes



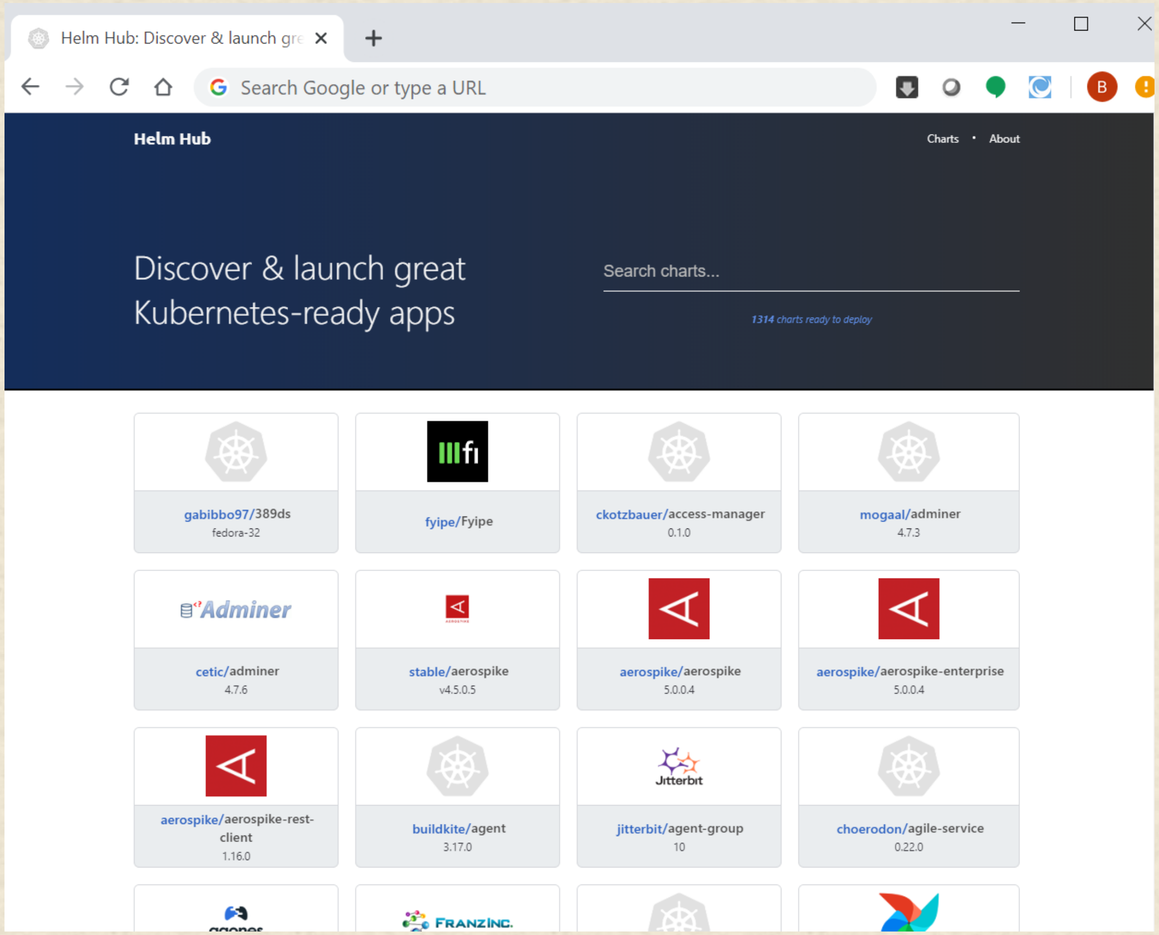
Helm Chart Repositories

- Chart repo
  - Location where Helm charts can be stored and shared
  - Able to serve yaml and tar files
    - » Helm package format is tar file
    - » Index file for repo is yaml
  - Responds to REST API GET requests to get packages
  - Many storage options available – cloud buckets, local storage, GH Pages, etc.
- Helm instance can have many repos defined/added


Local Helm configuration
stable
https://kubernetes-charts.storage.googleapis.com/
local
https://localhost:8080
mycompany
https://mycompany.github.io/helm-charts

Helm Hub

- Default source for charts
- hub.helm.sh
- Built-in repo for helm



Discover & launch great Kubernetes-ready apps

1314 charts ready to deploy

gabibbo97/389ds
fedora-32

fyipe/Fyipe

ckotzbauer/access-manager
0.1.0

mogaal/adminer
4.7.3

cetic/adminer
4.7.6

stable/aerospike
v4.5.0.5

aerospike/aerospike
5.0.0.4

aerospike/aerospike-enterprise
5.0.0.4

aerospike/aerospike-rest-client
1.16.0

buildkite/agent
3.17.0

jitterbit/agent-group
10

choerodon/agile-service
0.22.0

ChartMuseum

- Open-source Helm chart repository
- Available at chartmuseum.com
- Written in GoLang
- Storage backends available for main cloud providers
- Easy to run locally
- Plugins for simplifying packaging and uploading

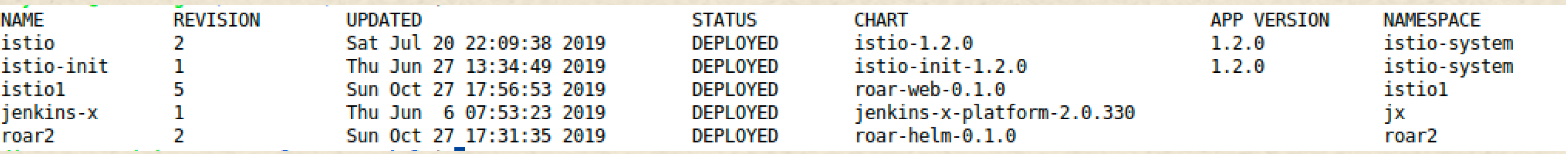
Helm Repo Operations

- Show all repos
  - $ helm repo list
- Add a repo
  - $ helm repo add <repo> url
  - $ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
- Search a repo
  - $ helm search repo <repo> <chartname>
  - $ helm search hub
- Remove a repo
  - $ helm repo rm <repo>

```
$ helm search repo stable | head
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                 7.1.5            1.10.10        Airflow is a platform to programmatically autho...
stable/ambassador              5.3.2            0.86.1         DEPRECATED A Helm chart for Datawire Ambassador
stable/anchore-engine          1.6.9            0.7.2          Anchore container analysis and policy evaluatio...
stable/apm-server              2.1.5            7.0.0          The server receives data from the Elastic APM a...
stable/ark                     4.2.2            0.10.2         DEPRECATED A Helm chart for ark
```

Helm Topology

- Chart – a package; a bundle of K8s resources
- Release – a chart instance loaded into K8s
  - » Same chart can be installed several times into the same cluster
  - » Each such chart will have its own release
- Repository – a repository of published charts
- Template - a K8s configuration file mixed with Go/Sprig templates



```
NAME            REVISION        UPDATED                         STATUS          CHART                           APP VERSION     NAMESPACE
istio           2               Sat Jul 20 22:09:38 2019        DEPLOYED        istio-1.2.0                     1.2.0           istio-system
istio-init      1               Thu Jun 27 13:34:49 2019        DEPLOYED        istio-init-1.2.0                1.2.0           istio-system
istio1          5               Sun Oct 27 17:56:53 2019        DEPLOYED        roar-web-0.1.0                                  istio1
jenkins-x       1               Thu Jun  6 07:53:23 2019        DEPLOYED        jenkins-x-platform-2.0.330                      jx
roar2           2               Sun Oct 27 17:31:35 2019        DEPLOYED        roar-helm-0.1.0                                 roar2
```

Helm Operations

completion	generate autocompletions script for the specified shell (bash or zsh)
create	create a new chart with the given name
dependency	manage a chart's dependencies
env	helm client environment information
get	download extended information of a named release
help	Help about any command
history	fetch release history
install	**install a chart**
lint	examine a chart for possible issues
list	**list releases**
package	package a chart directory into a chart archive
plugin	install, list, or uninstall Helm plugins
pull	download a chart from a repository and (optionally) unpack it in local directory
repo	**add, list, remove, update, and index chart repositories**
rollback	roll back a release to a previous revision
search	**search for a keyword in charts**
show	**show information of a chart**
status	display the status of the named release
template	locally render templates
test	run tests for a release
uninstall	uninstall a release
upgrade	upgrade a release
verify	verify that a chart at the given path has been signed and is valid
version	**print the client version information**

Lab 1: Repos and Charts

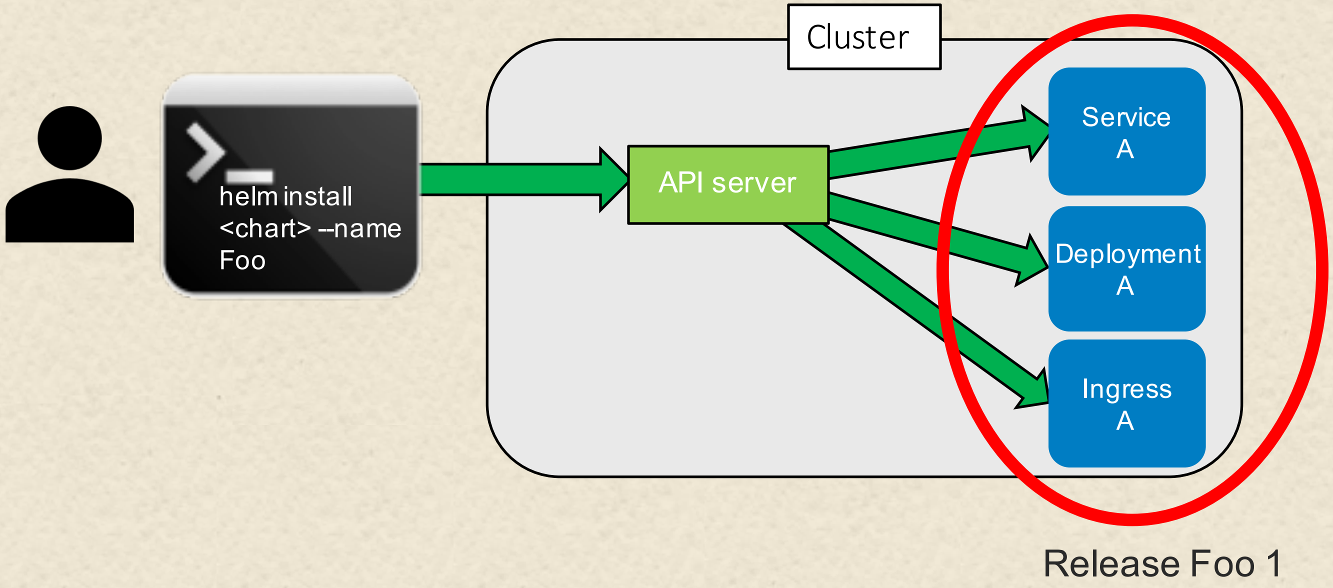
Helm Show (aka inspect) Operations

- helm show - shows information from a chart
- Format
  - $ helm show chart <repo>/<chartname>
- Show a chart's definition
  - $ helm show chart
- Show a chart's README
  - $ helm show readme
- Show a chart's values
  - $ helm show values
- Show all the information from a chart
  - $ helm show all

```
$ helm show chart stable/chartmuseum
apiVersion: v1
appVersion: 0.12.0
description: Host your own Helm Chart Repository
home: https://github.com/helm/chartmuseum
icon: https://raw.githubusercontent.com/helm/chartmuseum/
                master/logo2.png
keywords:
- chartmuseum
- helm
- charts repo
maintainers:
- email: opensource@codefresh.io
  name: codefresh-io
- email: hello@cloudposse.com
  name: cloudposse
- email: chartmuseum@gmail.com
  name: chartmuseum
name: chartmuseum
version: 2.13.0

$ helm show values  stable/chartmuseum |
rollingUpdate:
    maxUnavailable: 0
image:
  repository: chartmuseum/chartmuseum
  tag: v0.12.0
```

Helm Install a Chart

- helm install <chart>



Getting Information about Releases

- A release is an instance of a Helm chart deployed in Kubernetes
- Has a release name that can be different from chart name
- See list of releases
  - $ helm list
- See current status
  - $ helm status RELEASE_NAME [flags]
- See history of release
  - $ helm history RELEASE_NAME [flags]

```
$ helm status local-chartmuseum
NAME: local-chartmuseum
LAST DEPLOYED: Sun Jun 14 15:55:32
 2020
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
** Please be patient while the chart is
being deployed **

Get the ChartMuseum URL by running:
```

```
$ helm history local-chartmuseum
REVISION  UPDATED                   STATUS      CHART                     APP VERSION   DESCRIPTION
1         Sun Jun 14 15:45:38 2020  superseded  chartmuseum-2.13.00.12.0                Install complete
2         Sun Jun 14 15:55:32 2020  deployed    chartmuseum-2.13.00.12.0                Upgrade complete
```

Helm Status

- Shows status of a named release
  - Status fields
    - » Last deployment time
    - » K8s namespace where release is
    - » State of the release (see sidebar)
    - » List of resources, sorted by kind
    - » Details on last test run (if any)
    - » Additional notes provided by chart

```
$ helm status local-chartmuseum
NAME: local-chartmuseum
LAST DEPLOYED: Sun Jun 14 15:55:32 2020
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
** Please be patient while the chart is being deployed **
```

Release States:

unknown

deployed

uninstalled

superseded

failed

uninstalling

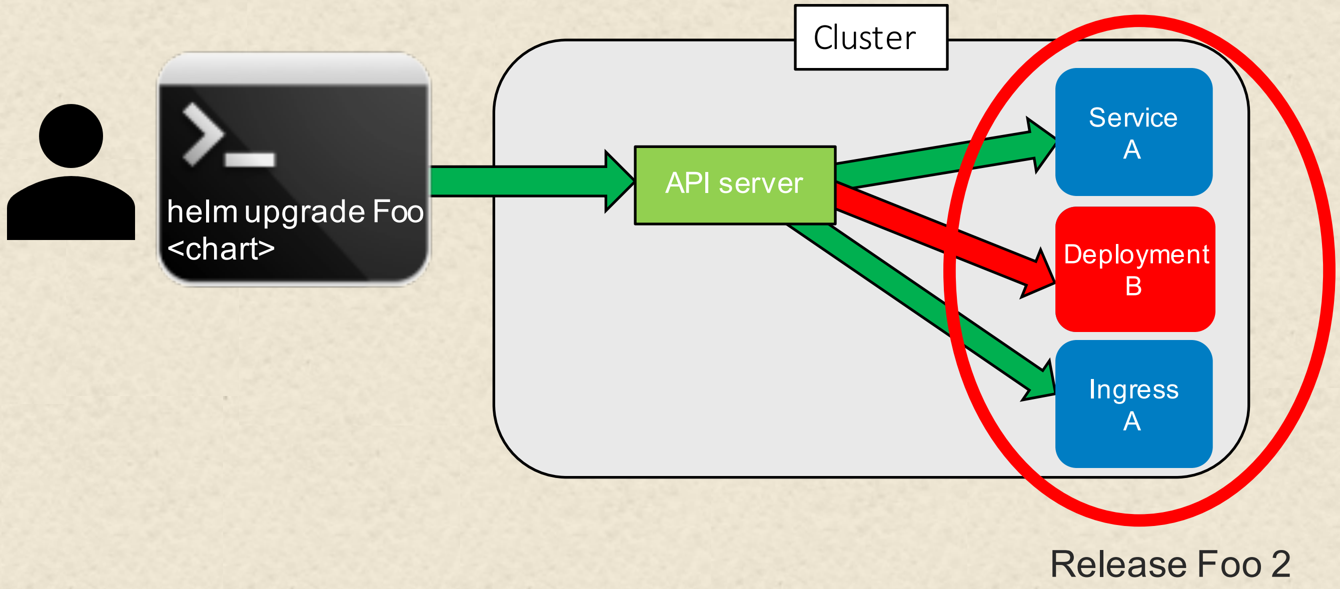
pending-install

pending-upgrade

pending-rollback

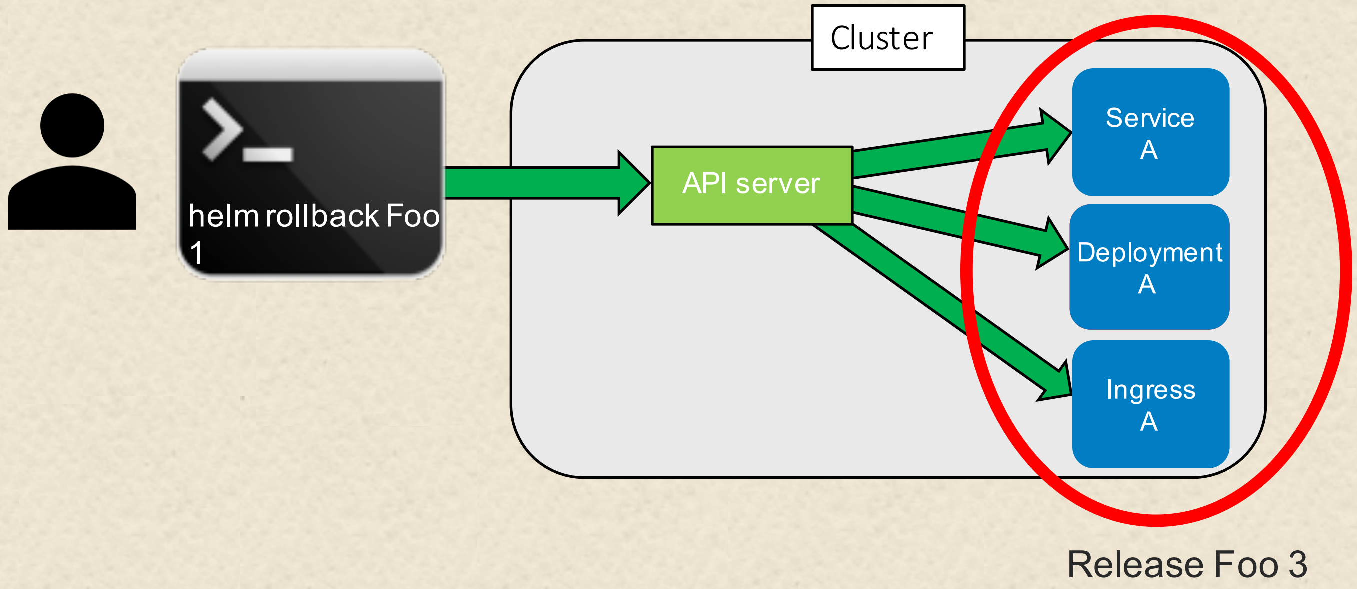
Helm Upgrading a Chart

- helm upgrade <release> <chart>
- use --set to override chart settings



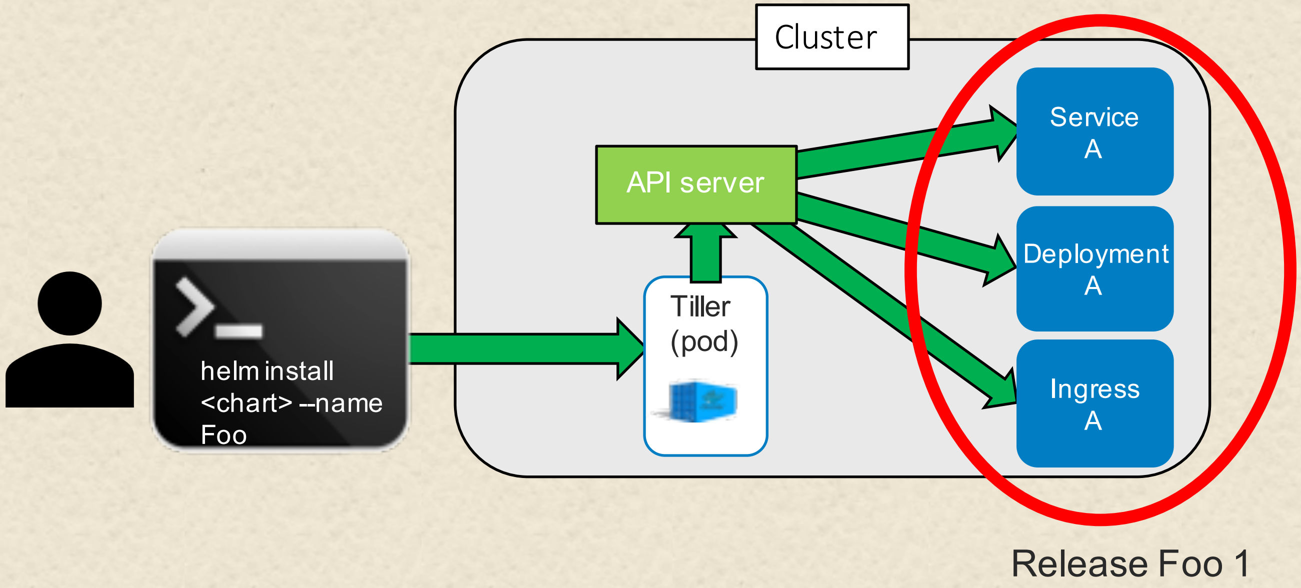
Helm Rollback

- helm history <release> (to see revision numbers)
- helm rollback <release> <revision>



Helm v2 Install a Chart

- helm install <chart>



```
helm install <chart> --name Foo
```

Differences between Helm 2 and Helm 3

- **Removal of Tiller component**
  - Server-side component prior to v3
  - Difficult to secure
  - No longer needed – done directly with K8S
- **Better managing of deltas between current and proposed charts**
  - Helm 2 used 2-way strategic merge patch
    - » For any Helm op, compared most recent chart against proposed
  - Helm 3 uses 3-way strategic merge patch
    - » Incorporates any changes made manually to cluster (such as kubectl edit)
    - » Previously not consider in merge, so manual changes weren't considered for rollbacks, etc.
- **Default storage driver is now secrets instead of configmaps**
  - Use of secrets saves having to encrypt/decrypte
- **JSON schema validation can be enforced**
  - Enforces that values provided by user conform to schema created by chart maintainer
- **Release name now required**
  - No more random names (unless you use –generate-name flag)
- **Helm serve removed**
  - Was used to serve a local chart repo on a system (mostly for dev purposes)
  - Still available as plugin
- **Namespaces not created automatically anymore**
  - Helm 2 used to create automatically when creating a release in a namespace

Helm 3 – Commands removed/replaced/added

- delete: now uninstall (--purge option to remove release history is now implied)
- fetch: now pull
- home: removed (displayed location of $HELM-HOME)
- init: removed (installed Tiller)
- install: now requires release name (or --generate-name)
- inspect: now show
- reset: removed (uninstalled Tiller from cluster)
- template: –x /--execute arg renamed to -s / --show-only
- upgrade: additional arg --history-max – limits # of saved releases

Lab 2: Changing Values

What is a Chart?

- Packaging format for Helm
- Define a way to compose a set of K8S resources and values to make up a deployment
- Deployable unit
  - Can be installed, updated, removed
- "Source code"
  - Can be versioned and managed in source control and compressed in packages
- Charts can include other charts as dependencies

Generating New Charts

- Done via the “helm create” command
- Creates scaffold directory/file structure
- Example: “$ helm create mychart”

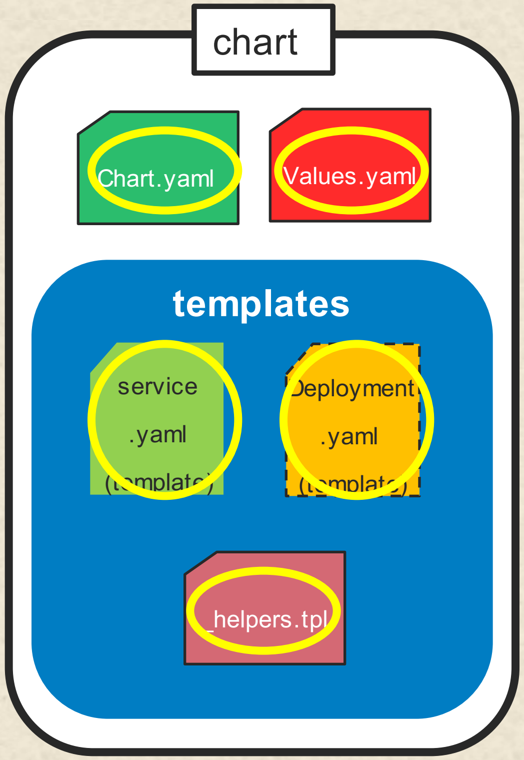
Helm Chart Structure

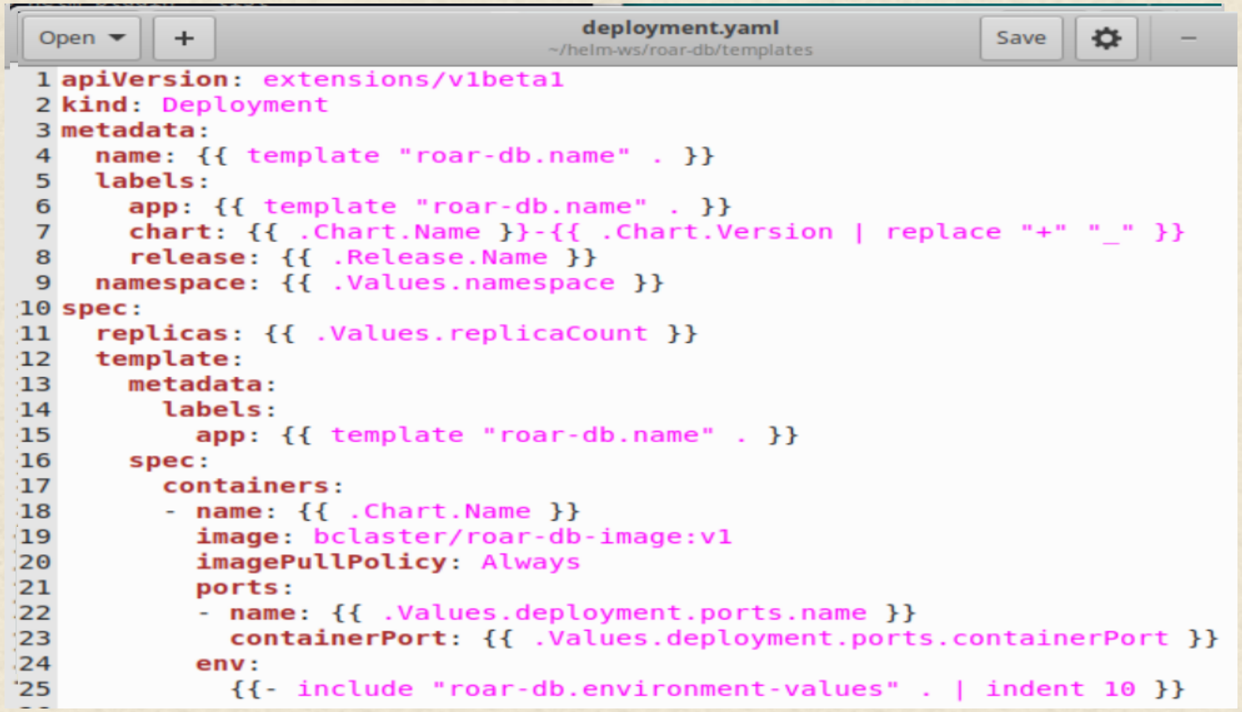
- Helm charts have an explicit structure
- Not all elements have to be present

```
chart-name/
        Chart.yaml          # A YAML file containing information about the chart
        LICENSE             # OPTIONAL: A plain text file containing the license for the chart
        README.md          # OPTIONAL: A human-readable README file
        values.yaml        # The default configuration values for this chart
        values.schema.json  # OPTIONAL: A JSON Schema for imposing a structure on the values.yaml file
        charts/            # A directory containing any charts upon which this chart depends.
        crds/              # Custom Resource Definitions
        templates/        # A directory of templates that, when combined with values,
                          # will generate valid Kubernetes manifest files.
        templates/NOTES.txt    # OPTIONAL: A plain text file containing short usage notes
```

Example Files in a Helm Chart

- Main chart directory
- Chart.yaml – chart description
- Values.yaml – values to be used in chart
- Templates – templated files that will form K8s manifests
- _helpers.tpl – helper functions





```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: bclaster/roar-db-image:v1
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.name }}
          containerPort: {{ .Values.deployment.ports.containerPort }}
        env:
          {{- include "roar-db.environment-values" . | indent 10 }}
```

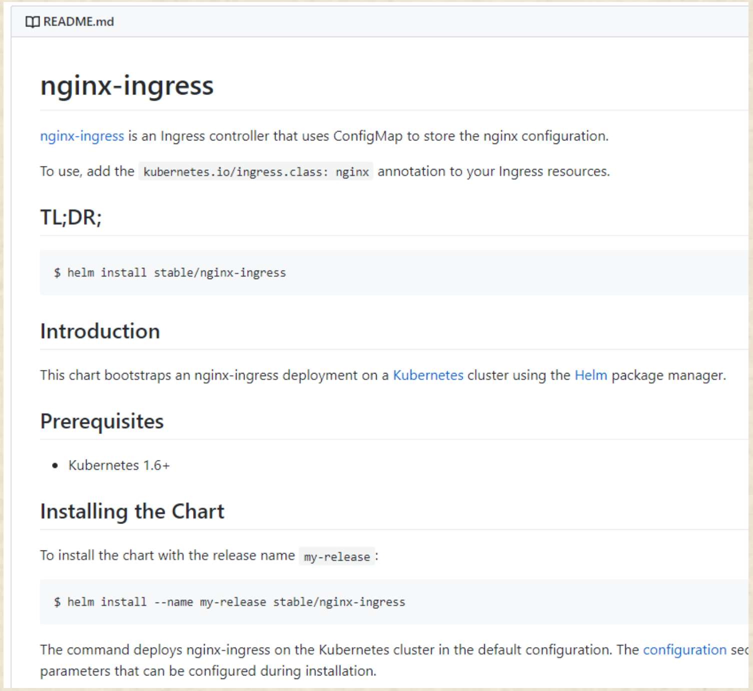
Chart.yaml file structure

- Required file for each chart
- Few fields required, most optional
- Subsections have own fields that may be optional or required (dependencies, maintainers, etc.)
- Type – defines type of chart
  - application – standard
  - library – provides utilities or functions for a chart builder

```
apiVersion: The chart API version (required)
name: The name of the chart (required)
version: A SemVer 2 version (required)
kubeVersion: SemVer range of compatible K8s versions (optional)
description: A single-sentence description of this project (optional)
type: The type of the chart (optional)
keywords:
 - A list of keywords about this project (optional)
home: The URL of this projects home page (optional)
sources:
 - A list of URLs to source code for this project (optional)
dependencies: # A list of the chart requirements (optional)
 - name: The name of the chart (nginx)
   version: The version of the chart ("1.2.3")
   repository: The repository URL or alias ("@repo-name")
   condition: (optional) - boolean for enabling/disabling charts
   tags: # (optional) - for grouping charts and enabling/disabling
   enabled: (optional) Enabled bool determines if chart should be loaded
   import-values: # (optional) - mapping of source values to parent keys
   alias: (optional) Alias to be used for the chart (optional)
maintainers: (optional)
 - name: The maintainers name (required for each maintainer)
   email: The maintainers email (optional for each maintainer)
   url: A URL for the maintainer (optional for each maintainer)
icon: A URL to an SVG or PNG image to be used as an icon (optional).
appVersion: The version of the app that this contains (optional)
deprecated: Whether this chart is deprecated (optional, boolean)
annotations: A list of annotations keyed by name (optional).
```

Optional Supporting Files

- LICENSE – plain text file containing license for the chart
- README.md
  - Markdown format
  - Displayed if user does "helm show readme"
  - Can be used to provide configuration info, urls, explanations, etc.
- templates/NOTES.txt
  - Used to display instructions or meaningful info to users
  - Shown at the end of a helm install or upgrade
  - plain-text, but processed as a template (all template functionality available)
  - Strongly recommended



README.md

nginx-ingress

nginx-ingress is an Ingress controller that uses ConfigMap to store the nginx configuration.

To use, add the `kubernetes.io/ingress.class: nginx` annotation to your Ingress resources.

TL;DR;

```
$ helm install stable/nginx-ingress
```

Introduction

This chart bootstraps an nginx-ingress deployment on a Kubernetes cluster using the Helm package manager.

Prerequisites

- Kubernetes 1.6+

Installing the Chart

To install the chart with the release name `my-release`:

```
$ helm install --name my-release stable/nginx-ingress
```

The command deploys nginx-ingress on the Kubernetes cluster in the default configuration. The configuration sec
parameters that can be configured during installation.

templates/NOTES.txt

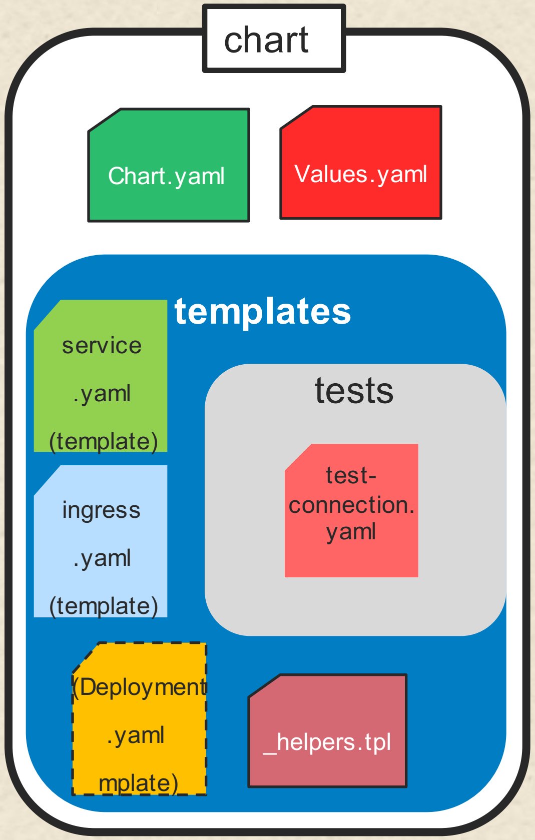
```
Thank you for installing {{ .Chart.Name }}.

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

To learn more about the release, try:

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

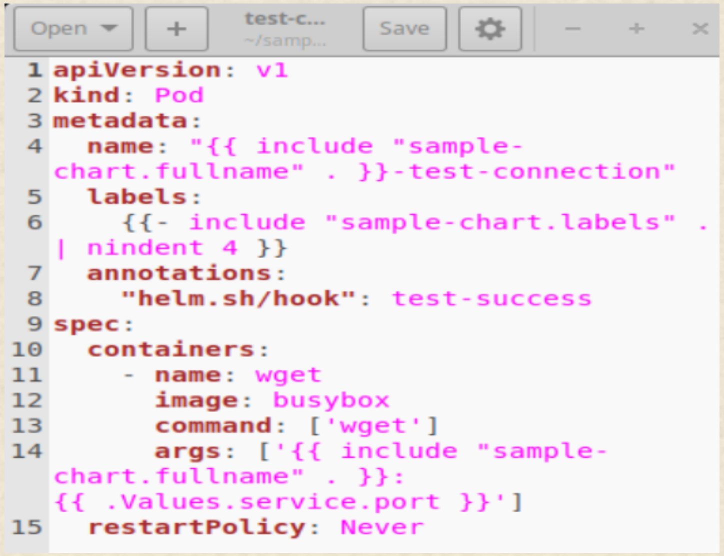
Rendering Templates/Getting Manifests

- Locally rendering templates
  - $ helm template NAME CHART [flags]
  - render templates locally and display output
- Download manifest for a named release
  - $ helm get manifest RELEASE_NAME [flags]
  - manifest = YAML-encoded version of K8s resources generated from chart
- Do a “dry-run” of an install
  - $ helm install NAME   CHART [flags] --dry-run --debug
    - » --dry-run = simulate an install but don’t do it
    - » --debug (global option) = verbose output

Running Tests in Helm

- Basic tests for deployed charts
- Can be included in tests subdir





```
apiVersion: v1
kind: Pod
metadata:
  name: "{{ include "sample-chart.fullname" . }}-test-connection"
  labels:
    {{- include "sample-chart.labels" . | nindent 4 }}
  annotations:
    "helm.sh/hook": test-success
spec:
  containers:
    - name: wget
      image: busybox
      command: ['wget']
      args: ['{{ include "sample-chart.fullname" . }}:{{ .Values.service.port }}']
  restartPolicy: Never
```

Deleting a Release

- Uninstalling a release
  - $ helm uninstall RELEASE_NAME
- Removes all resources that were part of last release
- Also removes release history by default
- Replaces “helm delete” in v2

Lab 3: Creating a Helm Chart

About Helm Versions – part 1

- version
  - Every chart is required to have a version number.
  - version must follow the SemVer2 standard
    - » 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.
  - version numbers used in release package names.
  - package names = name + version (ex: chart-1.2.3.tgz)
  - version # in chart package name must match version in Chart.yaml

Chart.yaml

```
apiVersion:
name:
version:
kubeVersion:
description:
type:
keywords:
home:
sources:
dependencies:
  - name:
    version:
    repository:
    condition:
    tags:
    enabled:
    import-values:
    alias:
maintainers:
  - name:
    email:
     url:
icon:
appVersion:
deprecated:
annotations:
```

About Helm Versions – part 2

- apiVersion
  - v2 for charts needing Helm 3 as minimum or using Helm 3 features
  - v1 for previous Helm versions and backwards-compatibility
- appVersion
  - Unrelated to version field
  - Specifies version of the application managed by chart
  - Information only (no impact on chart version)
- kubeVersion
  - Can be used to define which K8s versions are supported
  - If present, Helm will check when installing chart and fail if version of K8s is not supported
  - Can get very specific (>= 1.12.1 < 1.14.0 || >=1.14.2)

Chart.yaml

```
apiVersion:
name:
version:
kubeVersion:
description:
type:
keywords:
home:
sources:
dependencies:
  - name:
    version:
    repository:
    condition:
    tags:
    enabled:
    import-values:
    alias:
maintainers:
  - name:
    email:
     url:
icon:
appVersion:
deprecated:
annotations:
```

Dependencies

- At some point, may need to pull in another chart/app as a dependency
- Helm allows for identifying other charts that will be pulled in as part of a release
- To define a dependency, add dependencies section in Chart.yaml
- Deprecated method - create requirements.yaml in the chart root directory
- When dependencies are being updated, lockfile is generated so that later fetches of dependencies use a working, known version

Managing Helm Dependencies

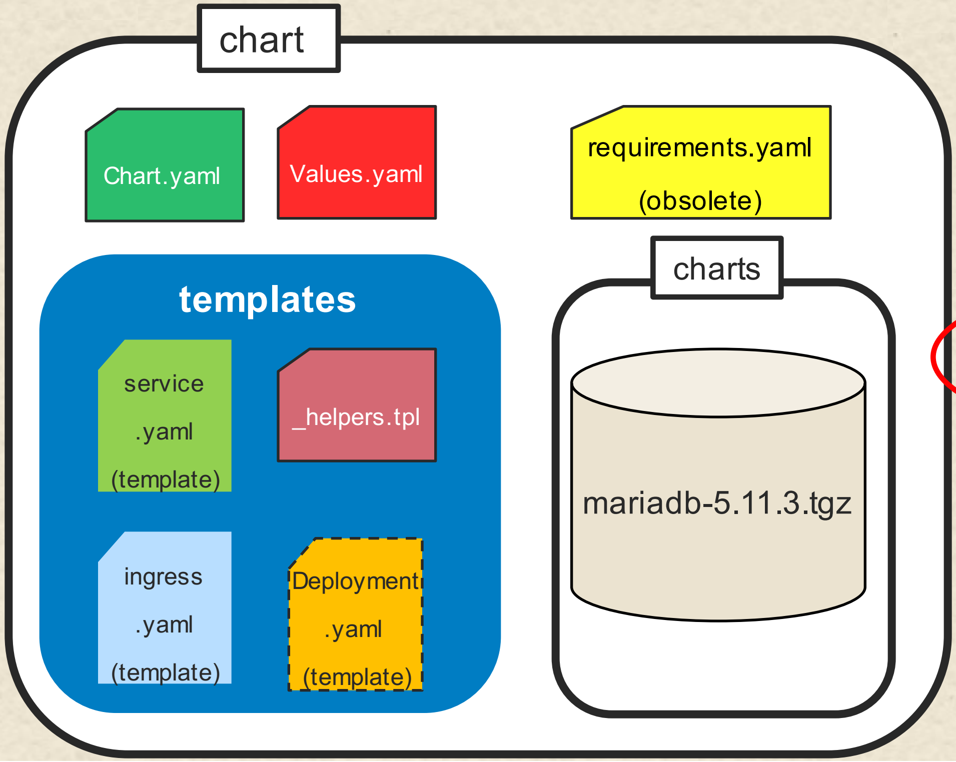
- Helm charts store their dependencies in ‘charts/’
- With v3, dependencies should be declared in “dependencies” block in “Chart.yaml”
- Commands: dependency build, dependency list, dependency update
- $ helm dependency update
  - downloads dependencies defined in requirements.yaml or in dependencies entry in Chart.yaml to the charts folder
- Dependency fields
  - name – name of a chart – must match name in other chart’s ‘Chart.yaml’ file
  - version - semantic version or version range
  - repository – url or path to chart

```
# Chart.yaml
dependencies:
- name: nginx
  version: “1.1.1”
  repository: “https://repo.com/charts”
```

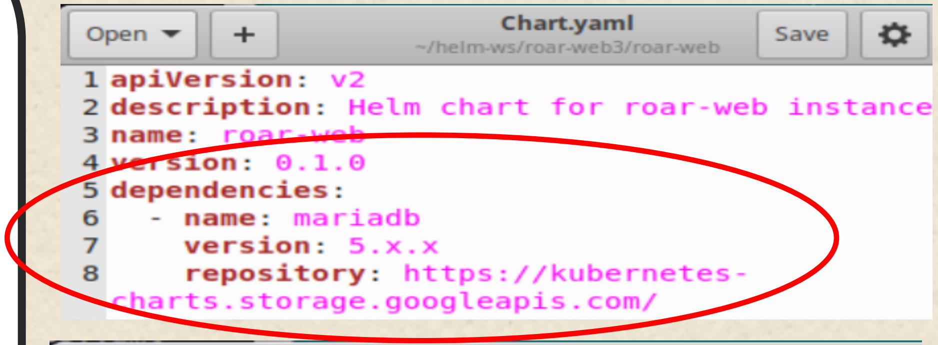
```
# Chart.yaml
dependencies:
- name: nginx
  version: “1.1.1”
  repository: “file://../other-charts/nginx”
```

Managing Dependencies between charts

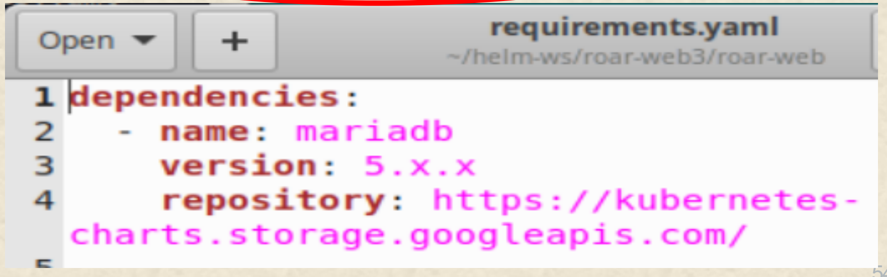
- Previous (but still supported) mechanism via requirement.yaml
- Helm v3 supports "dependencies" section in Chart.yaml
- Package dependencies shown



```
$ helm dep up
```



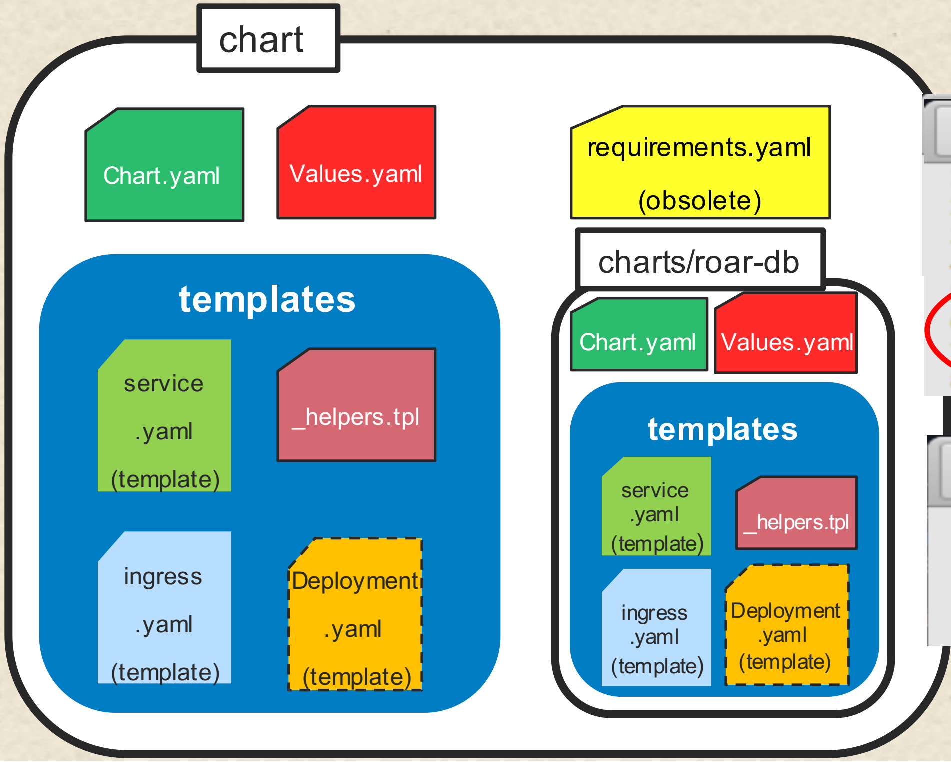
```
apiVersion: v2
description: Helm chart for roar-web instance
name: roar-web
version: 0.1.0
dependencies:
  - name: mariadb
    version: 5.x.x
    repository: https://kubernetes-charts.storage.googleapis.com/
```

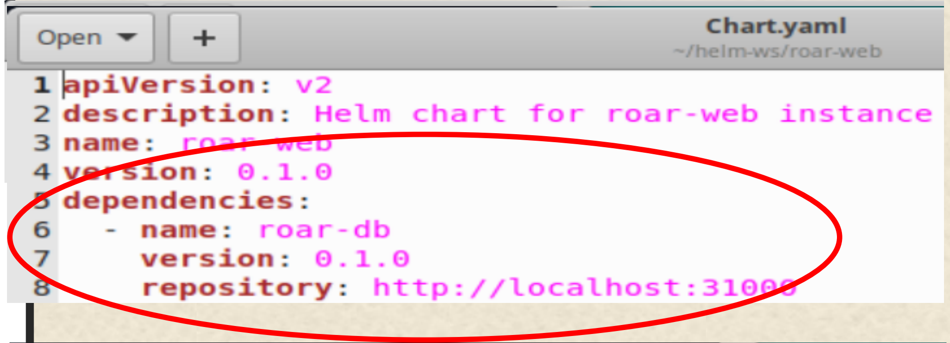


```
dependencies:
  - name: mariadb
    version: 5.x.x
    repository: https://kubernetes-charts.storage.googleapis.com/
```

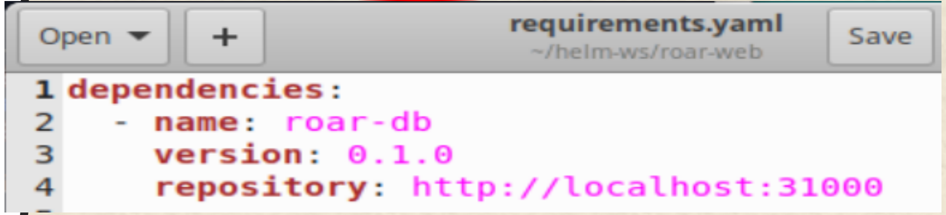
Managing Dependencies between Charts

- Full chart in subdirectory
- Previous (but still supported) mechanism via requirement.yaml
- Helm v3 supports "dependencies" section in Chart.yaml





```
apiVersion: v2
description: Helm chart for roar-web instance
name: roar-web
version: 0.1.0
dependencies:
  - name: roar-db
    version: 0.1.0
    repository: http://localhost:31000
```



```
dependencies:
  - name: roar-db
    version: 0.1.0
    repository: http://localhost:31000
```

Packaging and Uploading Files

- $helm package creates Helm package as tar file
- Can upload directly to repo
- May also require index.yaml generation and uploads
- Plugin for Helm adds “helm push” command to do all of this if using ChartMuseum

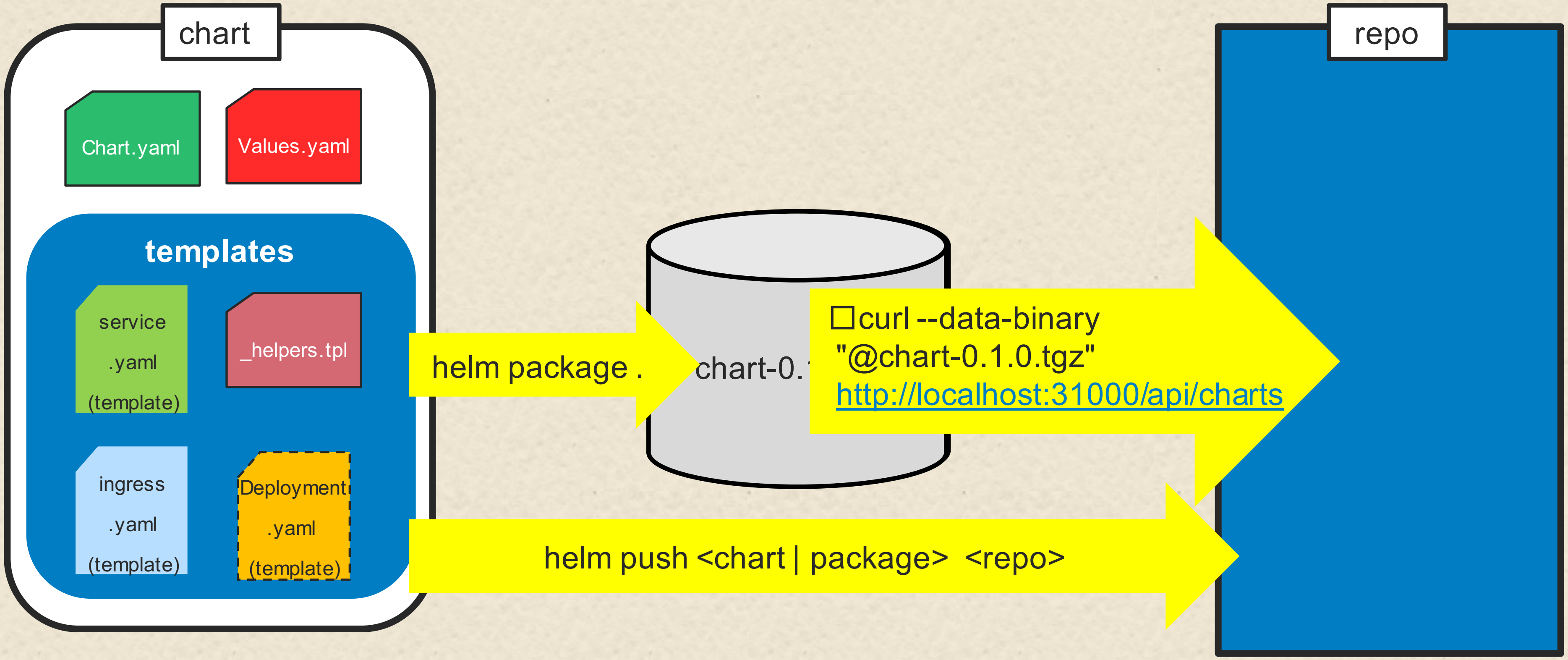


chart
Chart.yaml
Values.yaml
templates
service .yaml (template)
_helpers.tpl
ingress .yaml (template)
Deployment .yaml (template)
helm package .
chart-0.1
☐curl --data-binary "@chart-0.1.0.tgz" http://localhost:31000/api/charts
helm push <chart | package> <repo>
repo

Helm Plugins

- Install plugins
  - $ helm plugin install <path | url>
- List plugins
  - $ helm plugin list
- Update plugin
  - $ helm plugin update <plugin>
- Uninstall plugin
  - $ helm plugin uninstall <plugin>

```
$ helm plugin install https://github.com/chartmuseum/helm-push.git
Downloading and installing helm-push v0.8.1 ...
https://github.com/chartmuseum/helm-push/releases/download/v0.8.1/helm-push_0.8.1_linux_amd64.tar.gz
Installed plugin: push
$ helm | grep push
  push        Please see https://github.com/chartmuseum/helm-push for usage
```

Helm ignore file

- Used to identify files to not be included in a helm chart
- Helm package will ignore files matching patterns in here
- Supports relative paths, unix globbing, and negation with !
- Named .helmignore

.helmignore

```
# Example ignore file
.git
*/temp
*.bak
```

Lab 4: Charts and Dependencies

Templates Directory

- Location where Helm looks for the YAML definitions for Kubernetes objects (deployments, services, etc.)
- These YAML files can be "templated" (have placeholders for reusable values to be inserted)
- Helm runs each file in this directory through a Go template rendering engine
- When placeholders are filled in by running Helm, you end up with a standard K8S manifest

Understanding Templates

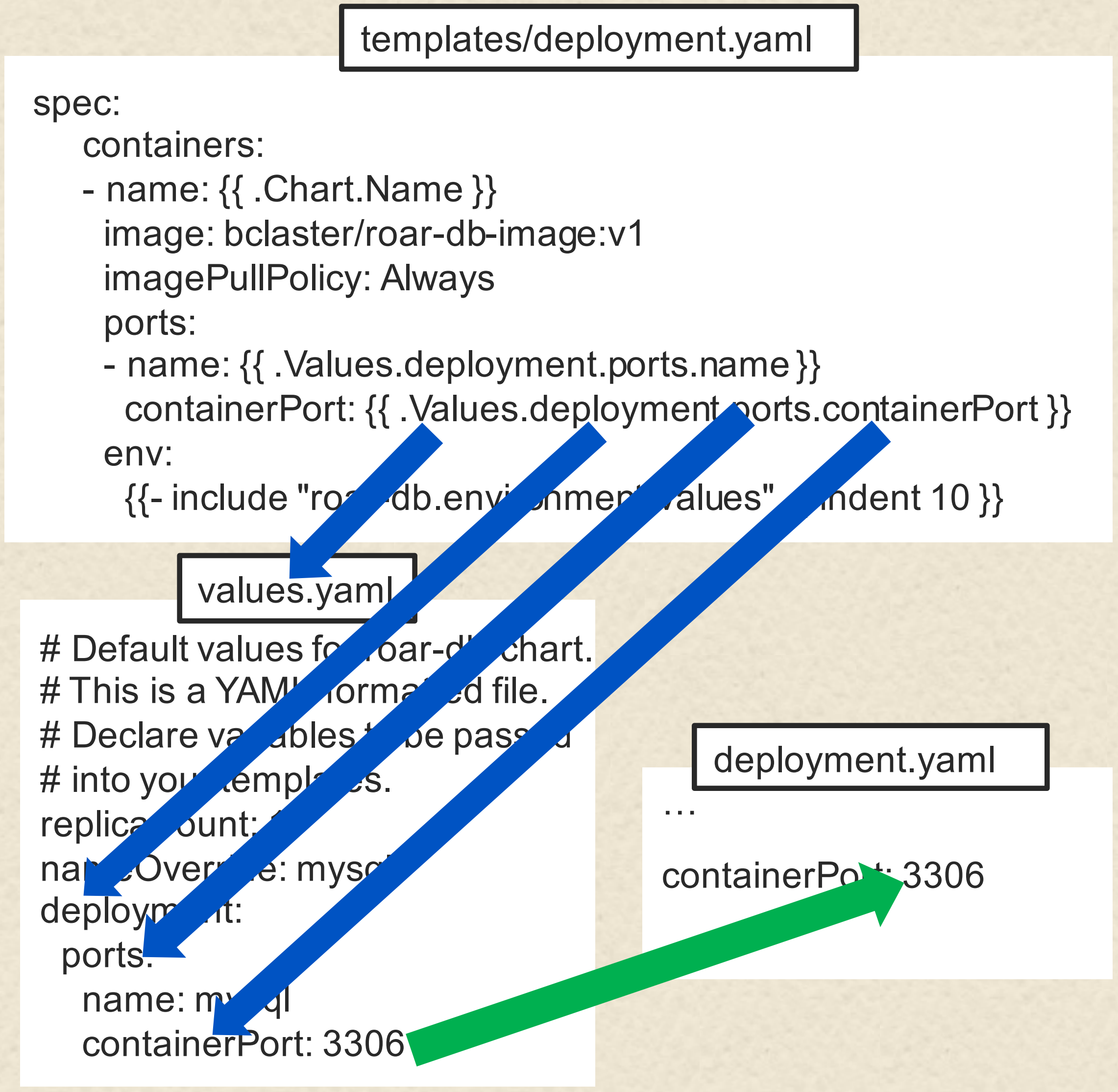
- Template directives (placeholders) are put between "{{" and "}}" blocks.
- Values passed inside the blocks are "namespaced objects"
  - example: {{ Release.Name }} puts the name of the release into the template (. separates levels)
  - Think of Release as top-level and then Name section/value under that
  - Can be thought of as "Start at top, find Release object, then find Name object inside of it"
  - Release object is "built-in" object in Helm
- Use "helm install --dry-run --debug" to see rendered template but not actually install it

Traditional K8s yaml vs Helm template

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: roar-web
  labels:
    app: roar-web
  namespace: roar
spec:
  replicas: 1
  template:
    metadata:
     labels:
       app: roar-web
    spec:
     containers:
     - name: roar-web
       image: localhost:5000/roar-web-v1
       ports:
       - name: web
         containerPort: 8080
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-web.name" . }}
  labels:
    app: {{ template "roar-web.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
     labels:
       app: {{ template "roar-web.name" . }}
    spec:
     containers:
     - name: {{ .Chart.Name }}
       image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
     {{- if .Values.image.pullPolicy }}
       imagePullPolicy: {{ toYaml .Values.image.pullPolicy }}
     {{- end }}
       ports:
       - name: {{ .Values.deployment.ports.name }}
         containerPort: {{ .Values.deployment.ports.containerPort }}
```

How Values are resolved in Helm


templates/deployment.yaml

```
spec:
    containers:
    - name: {{ .Chart.Name }}
      image: bclaster/roar-db-image:v1
      imagePullPolicy: Always
      ports:
      - name: {{ .Values.deployment.ports.name }}
        containerPort: {{ .Values.deployment.ports.containerPort }}
      env:
        {{- include "roar-db.environment.values" . | indent 10 }}
```

values.yaml

```
# Default values for roar-db chart.
# This is a YAML-formatted file.
# Declare variables to be passed
# into your templates.
replicaCount: 1
nameOverride: mysql
deployment:
  ports:
    name: mysql
    containerPort: 3306
```

deployment.yaml

```
…

containerPort: 3306
```

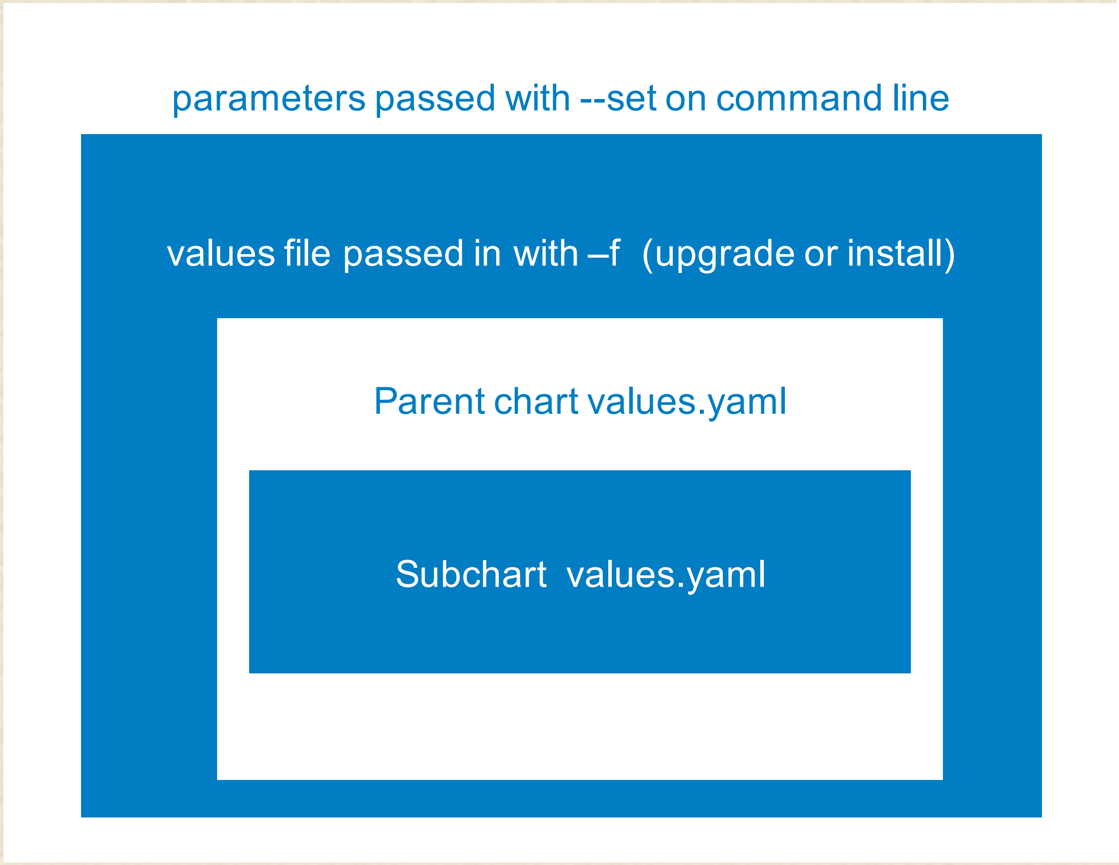
- string indicates hierarchial path
- dot notation separates levels
- .Values refers to top-level values from values.yaml

Top-level Objects

- .Release – describes the release itself
  - Release.Name, Release.Namespace
  - Release.Upgrade/Install – true if operation is upgrade/install
  - Release.Revision – starts at “1” for new install
  - Release.Service – service rendering template – always “helm”
- .Values – values from values.yaml or other specific files
  - empty by default
- .Chart – everything accessible from Chart.yaml
  - example: {{ Chart.Name }}
- .Files – provides functions to access common files in chart
  - example: .Files.Get <filename>
- .Capabilities – gets info about what current cluster supports
  - example: .Capabilities.APIVersions – gets a set of API versions
- .Template – info about current template being executed
  - example: .Template.Name – gets namespaced filepath to current template

Setting Values – Hierarchy/Order of Precedence

- .Values is a top-level object in Helm
- Provides access to items passed into the chart
- Those items can come from:
  - values.yaml file
  - if in a subchart, the values.yaml of a parent chart
  - a different file passed in to a "helm install" or "helm upgrade" with the –f flag as in "helm install –f myvalues.yml ./mychart"
  - parameters passed with –set on the command line
- Items in list above override from top-down. (--set overrides everything)


parameters passed with --set on command line
values file passed in with –f (upgrade or install)
Parent chart values.yaml
Subchart values.yaml

Linting

- Best practice as you develop charts to run through linter
- Ensures good form and good practices
- Invoked via “helm lint <chart>”

Lab 5: Templating

About Objects

- Objects passed in to templates from templating engine
- Objects can be passed around and manipulated by code
- Objects can have one to many values (sub-objects)
- Objects can also have functions that are callable

Template Functions

- Allows you to transform data passed into a template
- Function syntax
  - functionname arg1 arg2 …
- Over sixty functions available to use; provided by
  - Go template language
  - Go sprig template library
- Examples
  - upper, quote, eq
  - custom functions

Anatomy of a Helm Function

templates/deployment.yaml

```
name: {{ template "roar-web.name" . }}
```

templates/_helpers.tpl

```
{{/* vim: set filetype=mustache: */}}
{{/*
Expand the name of the chart. ←- DESCRIPTION
*/}}
                                    ←-FUNCTION
{{- define "roar-web.name" -}}       ←- NAME
{{- default .Chart.Name .Values.nameOverride -}}   ← BODY
{{- end -}}
                          ← END

{{/*
Create a default fully qualified app name.
We truncate at 63 chars because some Kubernetes
name fields are limited to this (by the DNS naming
spec).
*/}}
{{- define "roar-web.fullname" -}}
{{- $name := default .Chart.Name
.Values.nameOverride -}}
{{- printf "%s-%s" .Release.Name $name | trunc 63
-}}
{{- end -}}
```

- Template functions
  - Typically stored in _helpers.tpl
  - Have a description
  - Have a name
  - Have a body

More about Template Functions

- Helm uses Go templates to instrument manifests (mixed with extra functions and wrappers to expose objects to templates)
- Helm has almost all of the Sprig library functions
- Custom functions usually defined in _helpers.tpl
  - Any file that starts with "_" in templates directory is not expected to produce a usable K8s file
- **template function**
  - Allows you to pull in a template in your *.yaml file
  - Example: {{ template "myFunction" . }}
- **include function**
  - Allows you to include a template AND pass results to other functions
  - Example: {{ include "myFunction" . | upper | quote }}
- **required function**
  - Allows you to declare a particular values entry is required for a template
  - Example: {{ required "A valid .Values.foo is required!" .Values.foo }}
- **tpl function**
  - Allows you to evaluate strings as templates within a template
  - Example: Given .Values.template = "{{ .Values.version }}" and .Values.version = "1.0",

    then {{ tpl .Values.template . }} = "1.0"
- **default function**
  - Allows you to specify a default value in a template, in case the value is omitted
  - myItem: {{ .Values.item | default "foo" }}

Pipelines

- Similar to the idea of Unix pipelines
- Allows chaining together multiple template commands
- Simplify doing multiple things in sequence
- When pipelining arguments, result of previous evaluation is sent as last argument to next one

{{ first evaluation | function arg1 }}

equivalent to

{{ function arg1 arg2 }} where “arg2” = “first evaluation”

- Example

```
name:  {{ .Values.user.name | upper | quote }}
```

Dealing with Defaults in Helm

- Can use helm default function

```
item:  {{ .Values.myvalue  | default “default-val”   | quote }}
```

- If you don’t want a default value set, override it with nul

```
helm install mychart   --set <template>.<value>=null
```

- If there’s not a good default value or you want to always have a user-supplied value, use the requires function

```
{{ required “A valid value is required for thing”   Values.thing }}
```

Flow Control

- Conditionally include blocks of text in a template
  - If else
  - false if evaluates to
    - » boolean false
    - » numeric 0
    - » empty string
    - » nil (empty or null)
    - » empty collection
  - else true
- Limit scope
  - with
    - » "." is reference to current scope
    - » can limit scope such as

      {{- with .Values.subitem }}
- Looping
  - range
  - iterates through a collection

```
{{ if VALUE or PIPELINE }}
   # include a block of code
{{ else if OTHER VALUE or PIPELINE }}
   # include a block of code
{{ else }}
   # Default block of code
{{ end }}
```

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

```
{{ range SET }}
   # action
{{ end }}
```

Managing Whitespace

- Because Helm renders yaml files, whitespace is very important
- Helm's rendering engine removes contents of brackets but not newlines and whitespace
- "{{-" and "-}}" removes preceding/following whitespace
- Also available
  - nindent – like indent but adds a new line in rendered output

templates/deployment.yaml

```
image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
    {{- if .Values.image.pullPolicy }}
      imagePullPolicy: {{ toYaml .Values.image.pullPolicy }}
    {{- end }}
      ports:
      - name: {{ .Values.deployment.ports.name }}
        containerPort: {{ .Values.deployment.ports.containerPort }}
```

```
182-        app: roar-web
183-    spec:
184-      containers:
185-      - name: roar-web
186-        image: "bclaster/roar-web-image:v1"
187:        imagePullPolicy: Always
188-        ports:
189-        - name: web
```

Variables

- named reference to another object
- name of variable is $name
- assigned values with ":="
- uses
  - assign variables so that you can reference $name even if object is not in scope
  - in range loops, can be used to get both index and value
- example:

```
{{- $rname := .Release.name -}}
```

Template Use with Range and Variables

templates/deployment.yaml

```
spec:
    containers:
    - name: {{ .Chart.Name }}
      image: bclaster/roar-db-image:v1
      imagePullPolicy: Always
      ports:
      - name: {{ .Values.deployment.ports.name }}
        containerPort: {{ .Values.deployment.ports.containerPort }}
      env:
        {{- include "roar-db.environment-values" . | indent 10 }}
```

←CALL

$ helm template

templates/_helpers.tpl

```
{{/*
Pull in the various environment values.
*/}}
{{- define "roar-db.environment-values" -}}
  {{- range $key, $val := .Values.deployment.env }}
  - name: {{ $key }}
    value: {{ $val }}
  {{- end -}}
{{- end -}}
```

values.yaml

```
deployment:
  ports:
    name: mysql
    containerPort: 3306
  env:
    MYSQL_DATABASE: "registry"
    MYSQL_PASSWORD: "admin"
    MYSQL_ROOT_PASSWORD: "root+1"
    MYSQL_USER: "admin
```

deployment.yaml

```
containerPort: 3306
    env:
      - name: MYSQL_DATABASE
        value: registry
      - name: MYSQL_PASSWORD
        value: admin
     - name: MYSQL_ROOT_PASSWORD
        value: root+1
     - name: MYSQL_USER
        value: admin
```

Lab 6: Using Functions and Pipelines

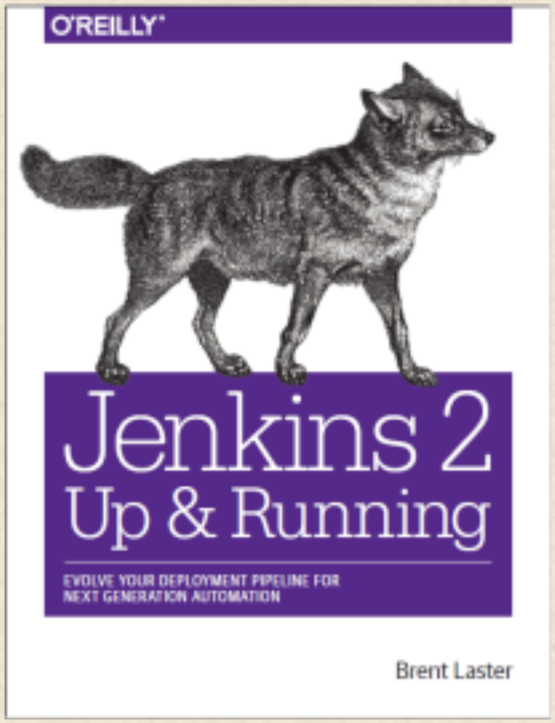
That's all - thanks!

Professional Git 1st Edition

by Brent Laster (Author)

3 customer reviews

Look inside


O'REILLY
Jenkins 2
Up & Running
EVOLVE YOUR DEPLOYMENT PIPELINE FOR NEXT GENERATION AUTOMATION
Brent Laster