Helm 101: Tame the chaos of your Kubernetes apps with Helm charts

Sahdev Zala
spzala@us.ibm.com

Henry Nash
henry.nash@uk.ibm.com

Martin Hickey
martin.hickey@ie.ibm.com

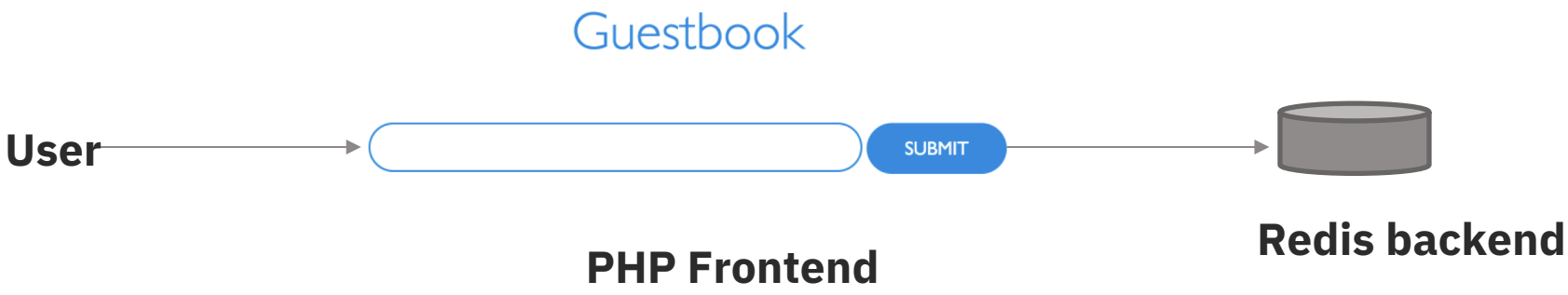
I want to be able to deploy and share my app everywhere consistently, and operate on its constituent parts as one?

Agenda

- Deploying app on a running Kubernetes cluster
  - *kubectl* way
  - Helm way
- Helm in action
- Learn more about Helm
  - How it works?
  - Charts
  - Repository
  - Release
  - Installation
- Where is Helm?
- Helm 3
- Summary
  - Kubernetes vs Helm

Guestbook App

Sample multi-tier web application which stores guest entries



Find out more:
https://github.com/IBM/guestbook/tree/master/v1/guestbook

Deploying Guestbook – *kubectl* Way

Let's see what it takes to deploy this app on a running Kubernetes cluster

- Total 6 YAML Kubernetes manifest files
  - Application deployment and service configuration
  - Redis master deployment and service configuration
  - Redis slaves deployment and service configuration
- Using the Kubernetes client, *kubectl*
  - Create Deployment
  - Manage Deployment
- Check out files at: https://github.com/IBM/guestbook/tree/master/v1

Deploying Guestbook – *kubectl* Way

Pain points

- CI/CD pipeline
  - *kubectl* deployments are not easy to configure, update and rollback
    - Deploying app to dev/test/production may require different configuration
      - Update deployment e.g. update with a new image
      - Change the configuration based on certain conditions
      - A different serviceType is needed in different environments (e.g. NodePort/LoadBalancer)
      - Need for rollback
      - Need of having multiple deployments (e.g. multiple Redis deployments)
  - Requires to track your deployment and modify YAML files (can be error prone)
  - Does not allow multiple deployments without updating metadata in manifest files
- Share your deployment configurations with your friend, team or customer?
  - You need to share many files and related dependencies
  - Your users are required to have knowledge of deployment configuration

Deploying Guestbook – Desired Way

- Templatize YAML files which allows providing configuration values at runtime and eliminate the need of modifying YAML files for
  - Scaling
  - Update
  - Rollback
- Package YAML files and all other dependencies
  - Easily share the package with others
  - Install the packaged app by providing desired configuration at runtime
    - Eliminate the need for user to learn details of Kubernetes resources
- Deploy same workload multiple times

Here Comes Helm

Deploying Guestbook – Helm Way

- No expertise of Kubernetes deployment needed as Helm hides Kubernetes domain complexities
- Helm packages all dependencies
- Desired configuration can be passed at runtime as key-value
- Helm tracks deployment making it easy to update and rollback
- Same workload can be deployed multiple times
  - Helm allows assigning workload release names at runtime
- Easy to share

So, What is Helm?

- Helm is a tool that streamlines installation and management of Kubernetes applications
  - Helm became a CNCF project in mid 2018
- It uses a packaging format called **charts**
  - A chart is a collection of files that describe Kubernetes resources
  - Think of Helm like apt/yum/homebrew for Kubernetes
- Helm is available for various operating systems like OSX, Linux and Windows
- Run Helm anywhere e.g. laptop, CI/CD etc.

So, What Helm is NOT

– A fully fledged **system** package manager

– A configuration management tool like Chef, puppet etc.

– A Kubernetes resource lifecycle controller

Five Keywords

helm

– While *Helm* is the name of the project, the command line client is also named helm. By convention, when speaking of the project, ***Helm*** is capitalized. When speaking of the client, ***helm*** is in lowercase.

Tiller

– Tiller is the Helm server. It interacts directly with the Kubernetes API server to install, upgrade, query, and remove Kubernetes resources. It is installed in the Kubernetes cluster.

Chart

– It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster. A chart is basically a package of pre-configured Kubernetes resources.

Release

– An instance of a chart running in a Kubernetes cluster

– Same chart can be deployed multiple time

Repository

– Place where charts reside and can be shared with others

Let’s See Helm in Action

Demo – Guestbook Chart Deployment

- Check existing installation of Helm chart
  - ***helm ls***
- Check what repo do you have
  - ***helm repo list***
- Add repo
  - ***helm repo add helm101*** ***https://ibm.github.io/helm101/***
- Verify that helm101/guestbook is now in your repo
  - ***helm repo list***
  - ***helm search helm101***
- Install
  - ***helm install helm101/guestbook --name myguestbook --set serviceType=NodePort*** – follow the output instructions to see your guestbook application
- Verify that your guestbook chart is installed
  - ***helm ls***
- Check chart release history
  - ***helm history myguestbook***

Demo – Guestbook Upgrades and Rollback

First let's see what we have

- ***helm history myguestbook***
  - REVISION UPDATED STATUS CHART DESCRIPTION
    1 Thu May 17 21:54:29 2018 DEPLOYED guestbook-0.1.0 Install complete

Upgrade

- ***helm upgrade myguestbook helm101/guestbook***
- ***helm history myguestbook***
  - REVISION UPDATED STATUS CHART DESCRIPTION
    1 Thu May 17 21:54:29 2018 SUPERSEDED guestbook-0.1.0 Install complete
    2 Fri May 18 09:08:10 2018 DEPLOYED guestbook-0.1.0 Upgrade complete

Rollback

- ***helm rollback myguestbook 1***
  - ***helm history myguestbook***
    - REVISION UPDATED STATUS CHART DESCRIPTION
      1 Thu May 17 21:54:29 2018 SUPERSEDED guestbook-0.1.0 Install complete
      2 Fri May 18 09:08:10 2018 SUPERSEDED guestbook-0.1.0 Upgrade complete
      3 Fri May 18 09:11:25 2018 DEPLOYED guestbook-0.1.0 Rollback to 1

Demo – Clean Up

- Remove repo
  - **helm repo remove helm101**
- Remove chart completely
  - **helm delete --purge myguestbook**
    - Delete all Kubernetes resources generated when the chart was instantiated

Guestbook Chart

https://github.com/IBM/helm101/tree/master/charts/guestbook

Let’s Learn More About Helm

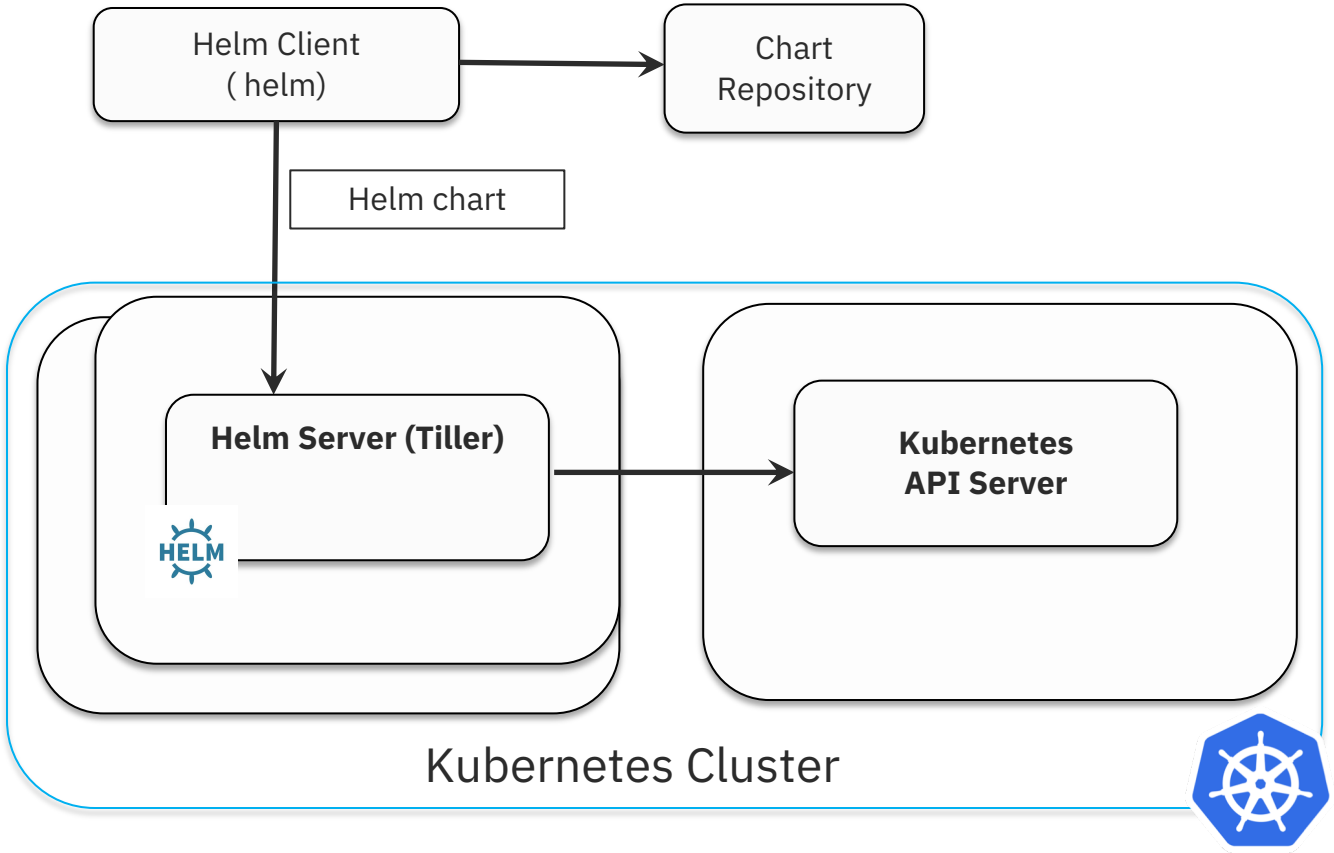
Helm Architecture

The Helm Client (helm)

- Command-line client for end users
- Interacts with the Tiller server sending charts to be installed/upgrade/uninstall
- Interacts with the chart repository

The Helm Server (Tiller)

- Interacts with the Helm client and interfaces with the Kubernetes API server
- Renders templates to Kubernetes manifest files
- Install/Upgrade/Uninstall charts into Kubernetes, and then tracks the subsequent release



What’s the Chart all about?

- A collection of files that describe a related set of Kubernetes resources
- Files laid out in a directory tree structure
- Templates based on "Go template language" + functions from “Sprig” lib + specialized functions:
  - values, functions, pipelines, operators, flow control, variables, built-in objects
- Templates rendered by Helm template engine into Kubernetes manifest files
- Can be packaged into versioned objects for sharing (repos)
- Version numbers (SemVer2 ‘major.minor.path’) used as release markers
- Charts can have dependencies on other charts

Chart Template Snippet

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "guestbook.fullname" . }}
  labels:
    app.kubernetes.io/name: {{ include "guestbook.name" . }}
    helm.sh/chart: {{ include "guestbook.chart" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
spec:
    replicas: {{ .Values.replicaCount }}
    selector:
.....................................................
      spec:
        ports:
        - name: http-server
          containerPort: {{ .Values.service.port }}
```

Chart Structure

- A chart is organized as a collection of files inside of a directory
- The directory name is the name of the chart e.g. guestbook.
- Inside of the directory, the expected file structure is

Required files:

Chart.yaml - A YAML file containing information about the chart.

One of the **charts** or **templates** directory:

- **charts/** - A directory containing any charts upon which this chart depends. (static linked)
- **templates/** - A directory of templates with Kubernetes manifest files or that will generate valid Kubernetes manifest files when combine with values.yaml.

Optional files:

LICENSE - A plain text file containing the license for the chart
README.md - A human-readable README file
requirements.yaml - A YAML file listing dependencies for the chart (dynamic linked)
values.yaml - The default configuration values for this chart
templates/NOTES.txt - A plain text file containing short usage notes
template/_helpers.tpl – template helpers that you can re-use throughout the chart

```
Sahdevs-MBP:guestbook sahdevzala$ tree
.
├── Chart.yaml
├── LICENSE
├── README.md
├── charts
├── templates
|   ├── NOTES.txt
|   ├── _helpers.tpl
|   ├── guestbook-deployment.yaml
|   ├── guestbook-service.yaml
|   ├── redis-master-deployment.yaml
|   ├── redis-master-service.yaml
|   ├── redis-slave-deployment.yaml
|   └── redis-slave-service.yaml
└── values.yaml
```

Helm Repository

- A chart repository is an HTTP server that houses an **index.yaml** file and some packaged charts
  - The index file contains information/metadata about each chart in the repository
- The preferred way of sharing chart is by uploading them to a chart repository
- Use cloud provider, third party repository or Create your own
  - A valid chart repository must have an index.yaml file
  - The ***helm repo index*** will generate an index file based on a given local directory that contains packaged charts
- Add repo to your local environment
  - ***helm repo add myguestbook https://ibm.github.io/helm101/***
    - Where myguestbook is the name of local repo and https:// url provides the remote repository
    - This downloads charts locally and allows you to search and install any of the charts without using the repository URL
    - ***helm repo list*** provides list of available repositories locally

Where are the Charts?

- Charts reside in:
  - Helm upstream charts repository
    - https://github.com/helm/charts
    - Contains several official Helm charts.
    - The Charts in this repository are organized into two folders:
      - Stable – ready to use
      - Incubator – under active development
  - IBM provided Helm charts
    - https://github.com/IBM/charts
    - Charts for IBM and third party applications
  - Various cloud provider repositories e.g. Google cloud storage
  - Individually owned repository e.g. GitHub
  - Local repository
  - File system

Helm Release

- When a chart is installed, Helm creates a *release* to track that installation
  - A single chart may be installed many times into the same cluster using different releases
    - Let's say you need multiple, independent, instance of database on a same cluster
      - » You can reuse the same chart that can deploy a database by specifying different release names
        - ***helm install --name redis1 stable/redis***
        - ***helm install --name redis2 stable/redis***
  - A single release can be upgraded or rolled back multiple times
    - A sequential counter is used to track releases as they change
    - After a first helm install, a release will have *release number* 1
    - Each time a release is upgraded, the release number will be incremented
    - Since release history is stored, a release can also be *rolled back* to a previous release

Upgrade and Rollback

- Upgrade
  - ***helm upgrade <RELEASE> <CHART>***
  - This will upgrade existing deployment of specified release
- Rollback
  - ***helm rollback <RELEASE> <REVISION>***
  - This will rollback a helm deployment to the specified revision number
  - To see revision numbers, run ***helm history <RELEASE>***
- Get the history of upgrades and rollbacks
  - ***helm history*** **< RELEASE>**

Installing Helm

- Things to consider
  - There are two parts to Helm install:
    - Helm Client (helm)
    - Helm Server (Tiller)
  - Both of them are separate installs
  - Client version installed should not be later than the server version
  - Securing Helm requires additional steps: https://docs.helm.sh/using_helm/#securing-your-helm-installation
  - Find out more about installation: https://docs.helm.sh/using_helm/#installing-helm

Helm Client Install

- Install Helm Client (helm)
  - Binary downloads of the Helm client can be found for the following OSes:
    - OSX
    - Linux
    - Windows
    - Helm releases: https://github.com/helm/helm/releases
  - You can also use package manager:
    - macOS homebrew users
      - ***brew install kubernetes-helm***
    - Windows chocolatey users
      - ***choco install kubernetes-helm***
    - Linux snap users
      - ***sudo snap install helm --classic***

Helm Server Install

- Install Helm Server (Tiller)
  - Tiller can be installed in one of two ways:
    - In the Kubernetes cluster (recommended way)
    - Running locally (in some dev scenarios)
  - Install/Upgrade in the Kuberentes cluster:
    - ***helm init [–upgrade]***
    - Command also initializes the CLI
  - Run locally:
    - ***tiller***
    - ***export HELM_HOST=localhost:44134*** (for helm client. Alternatively, can pass *–host* when running client command)

Where is Helm?

- GitHub repository - https://github.com/helm/helm
- Channels on the Kubernetes Slack workspace - https://kubernetes.slack.com
  - #helm-users
  - #helm-dev
  - #charts
- Mailing list - https://lists.cncf.io/g/cncf-kubernetes-helm
- Developer call Every Thursdays at 9:30-10:00 Pacific - https://zoom.us/j/696660622
- Helm Documentation - https://docs.helm.sh
- Interested in contributing to the Helm?
  - Refer to the Helm Community - https://github.com/helm/community

Future

Helm v3

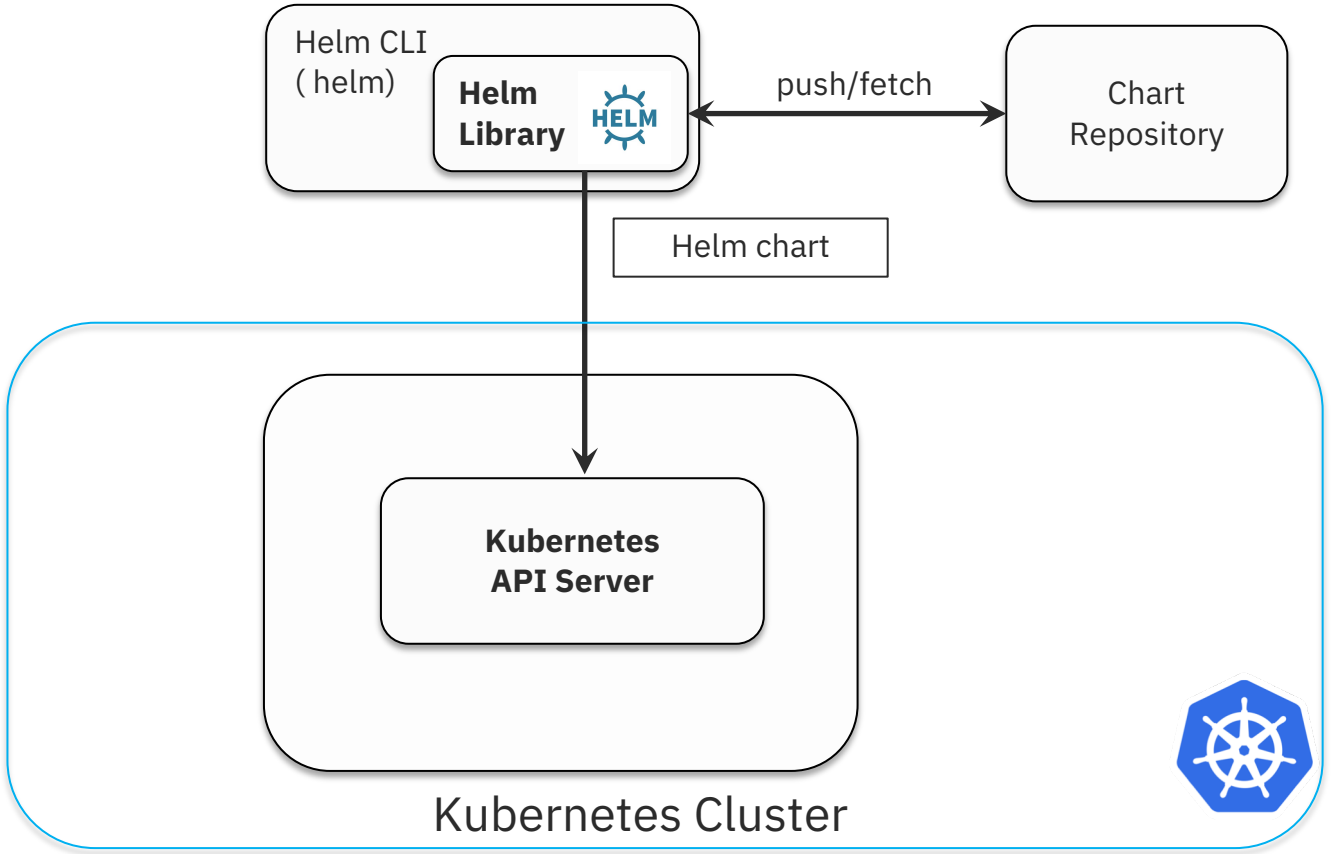
Key Changes

- No Server component (Tiller) – everything done via client
  - All operations under user credentials
- Charts are updated with:
  - Libraries that allow “define” attributes to be shared across charts
  - Schema to control values
  - Reserved ‘*ext*’ directory for extensions/scripts
- Helm will use a "lifecycle events" emitter/handler model
- Hooks are still supported, and will now be managed
- Helm has an embedded Lua engine for scripting some event handlers - scripts stored in charts
- For pull-based DevOps workflow, a new Helm Controller project will be started
  - This is not required for a deployment and will be optional
- Cross platform plugins in Lua that only have a runtime dependency on Helm
- A complementary command to ’*helm fetch’,* to push packages to a repository

Helm v3 Architecture

The Helm v3 Client (helm) is a command-line client façade that uses the (new) helm library to:

- Interface with the Kubernetes API server directly
- Supports all the functionality of helm v2
  - With a few exceptions that don't make sense with the new architecture
- Stores release state in the same namespace as the application destination
- Can push as well as fetch charts to/from a chart repository



Helm v3 Time Frame

- Provisional dates:
  - Alpha version: 2018 EOY
  - Release: Mid 2019
- More detail on v3 functionality
  - https://github.com/helm/community/blob/master/helm-v3/000-helm-v3.md
- Milestones can be tracked
  - https://github.com/kubernetes/helm/milestones
- A second use of the (new) Helm library is proposed as a “Helm Controller” that could be installed into the cluster to enable pull operations. Details of this are, as yet, not defined. However, this would NOT act on behalf of the v3 Helm client, rather have its own TBD interface.

Summary

Deployment - Kubernetes vs Helm

	Deployment
kubectl	• You need to know the manifest YAML file(s) you are deploying • *$ kubectl create –f *.yaml* • Can not deploy same workload without modifying YAML files. • Requires valid manifest upfront because it lacks templating mechanism like helm. With helm, a valid manifest can be produced at runtime combining YAML templates and values.yaml file.
helm	• No need to know the which YAML files to use. Install with chart name, helm install charts/guestbook --name guestbook1 • Same chart can be deployed multiple times by simply providing different release names at runtime. • Templating provides robustness of generating manifest files at runtime.

Upgrade/Rollback - Kubernetes vs Helm

	Upgrade/Rollback with new values

kubectl	• Multiple ways but each can add complexities
	  • Modify the values in the YAML files OR
	  • Create ConfigMap OR
	  • Set environmental variables
	• Other users can not get your environment.
	• User need to know the configuration for rollback.

helm	• No need to touch Kuberntes manifest files. Change the values in values.yaml or provide on command line. • For example, $ *helm install charts/guestbook --name guestbook1 --set serviceType=NodePort* • Configuration files are saved by Helm release which makes rollback easy.

Share Configuration Files - Kubernetes vs Helm

	Share
kubectl	• Not as easy as Helm chart unless for a basic deployments with a single YAML manifest file. • User need to download multiple files/dependencies.

helm	• Easy to share by uploading charts to remote repository
	• For example, several ready to use charts are available at:
	  • Helm official chart repo: https://github.com/helm/charts
	  • IBM chart repo: https://github.com/IBM/charts

Thank you

twitter.com/sp_zala

twitter.com/henrynash

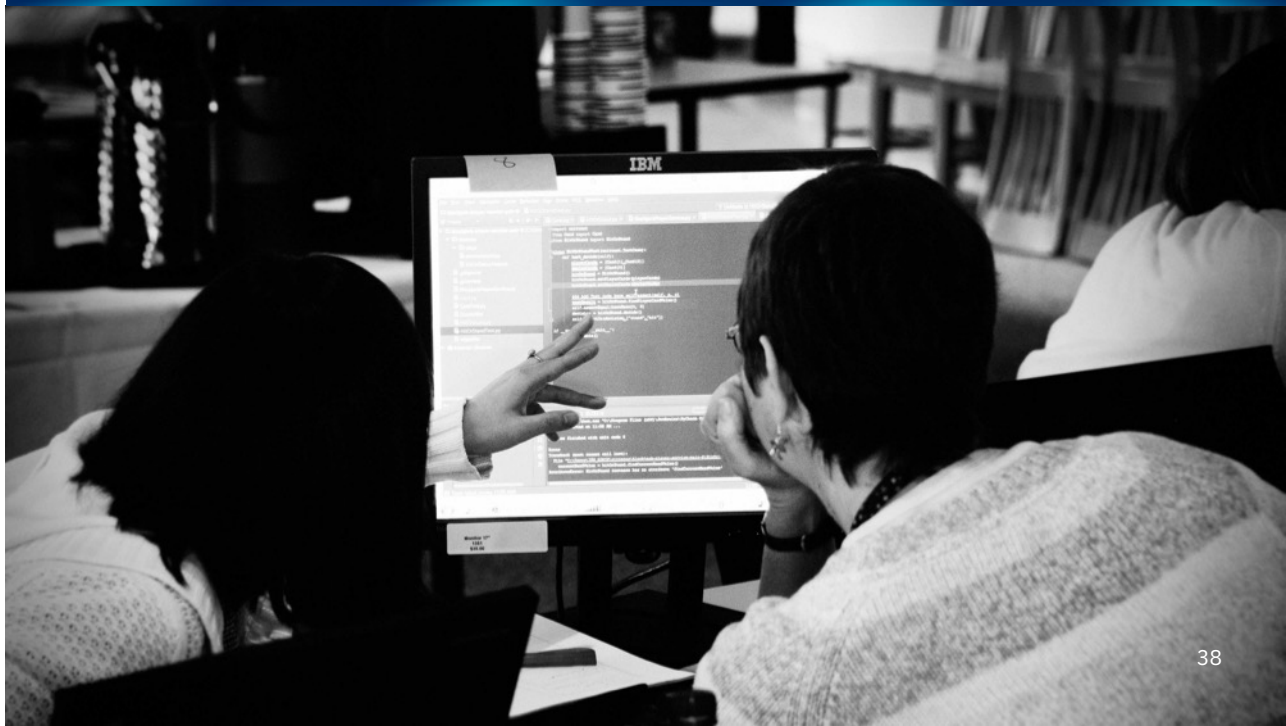
twitter.com/mhickeybot

github.com/spzala

github.com/henrynash

github.com/hickeyma

developer.ibm.com



Backup

Demo – Chart Deployment

Check existing installation of Helm chart

– ***helm ls***

Check what repo do you have

– ***helm repo list***

Add repo

– ***helm repo add helm101 https://ibm.github.io/helm101/***

Verify that helm101/guestbook is now in your repo

– ***helm repo list***

– ***helm search helm101***

Install

– ***helm install helm101/guestbook --name myguestbook --set serviceType=NodePort*** – follow the output instructions to see your guestbook application

Verify that your guestbook chart is installed

– ***helm ls***

Check chart release history

– ***helm history myguestbook***

Demo – Upgrades and Rollback

First let's see what we have

- ***helm history myguestbook***

REVISION	UPDATED	STATUS	CHART	DESCRIPTION
1	Thu May 17 21:54:29 2018	DEPLOYED	guestbook-0.1.0	Install complete

Upgrade

- ***helm upgrade myguestbook helm101/guestbook***
- ***helm history myguestbook***

REVISION	UPDATED	STATUS	CHART	DESCRIPTION
1	Thu May 17 21:54:29 2018	SUPERSEDED	guestbook-0.1.0	Install complete
2	Fri May 18 09:08:10 2018	DEPLOYED	guestbook-0.1.0	Upgrade complete

Rollback

- ***helm rollback myguestbook 1***
  - ***helm history myguestbook***

REVISION	UPDATED	STATUS	CHART	DESCRIPTION
1	Thu May 17 21:54:29 2018	SUPERSEDED	guestbook-0.1.0	Install complete
2	Fri May 18 09:08:10 2018	SUPERSEDED	guestbook-0.1.0	Upgrade complete
3	Fri May 18 09:11:25 2018	DEPLOYED	guestbook-0.1.0	Rollback to 1

Demo – Clean Up

Remove repo

– **helm repo remove helm101**

Remove chart completely

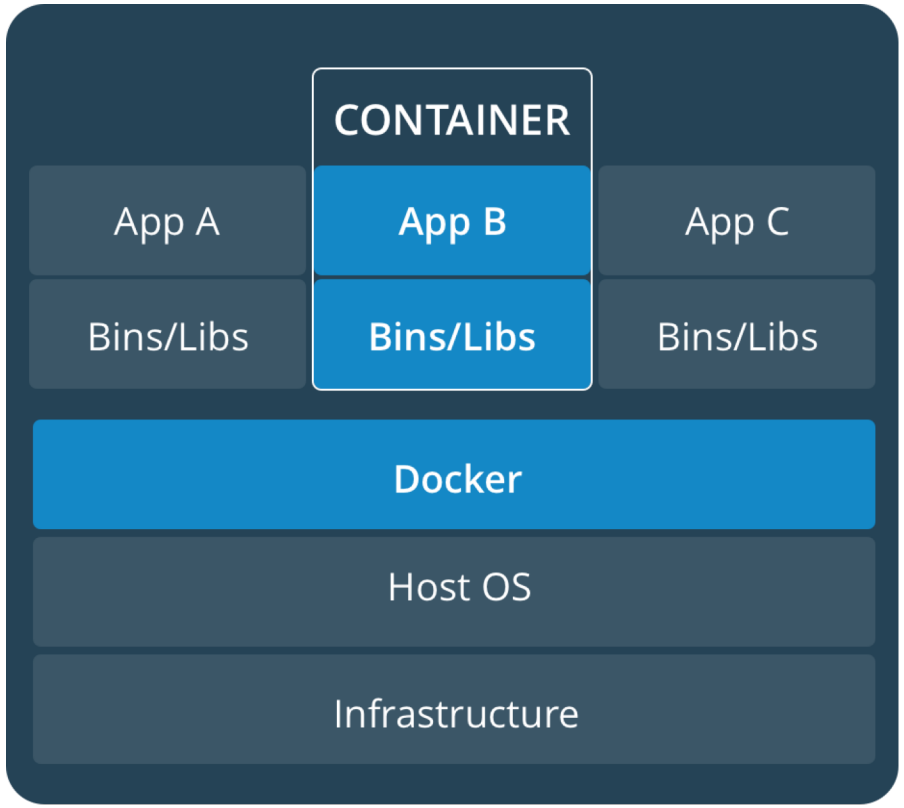
– **helm delete --purge myguestbook**

– Without –purge Helm will not completely remove your chart

Overview of Containers

•Abstraction at the app layer that packages code and dependencies together

- Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space



Learn more about containers here -
https://developer.ibm.com/courses/all/docker-essentials-extend-your-apps-with-containers/

Why Helm?

Helm provides probably the easiest way to use Kubernetes

– You can:

- Use existing charts created by others
- Create your own charts and share with others
- Easily manage your Kubernetes manifest files, configuration values and related resources as a package
- Release charts and manage releases

Configuring deployment with user values

Active development with a strong community behind it

– There is even a dedicated Helm Summit

Used widely in IBM and in the industry

Overview of Kubernetes

•Enterprise level container orchestration

•Provision, manage, scale applications (containers) across a cluster

- Manage infrastructure resources needed by applications
    - Compute
    - Volumes
    - Networks
    - And many many many more...
- Declarative model
    - Provide the "desired state" and Kubernetes will make it happen
- What's in a name?
    - Kubernetes (K8s/Kube): "Helmsman" in ancient Greek

Learn more about Kubernetes here - **https://github.com/IBM/kube101**

Kubernetes vs Helm deployments

	Deployment	Upgrade/Rollback with new values	Share

kubectl	• You need to know the manifest YAML file(s) you are deploying
	  • *$ kubectl create –f *.yaml*
	• Can not deploy same workload without modifying YAML files.
	• Requires valid manifest upfront because it lacks templating mechanism like helm. With helm, a valid manifest can be produced at runtime combining YAML templates and values.yaml file.
		• Multiple ways but each can add complexities
		  • Modify the values in the YAML files OR
		  • Create ConfigMap OR
		  • Set environmental variables
		• Other users can not get your environment.
		• User need to know the configuration for rollback.
			• Not as easy as Helm chart unless for a basic deployments with a single YAML manifest file.
			• User need to download multiple files.

helm	• No need to know the which YAML files to use. Install with chart name, helm install charts/guestbook --name guestbook1 • Same chart can be deployed multiple times by simply providing different release names at runtime. • Templating provides robustness of generating manifest files at runtime.	• No need to touch Kuberntes manifest files. Change the values in values.yaml or provide on command line. • For example, $ *helm install charts/guestbook --name guestbook1 --set serviceType=NodePort* • Configuration files are saved by Helm release which makes rollback easy.	• Easy to share by uploading charts to remote repository • For example, several ready to use charts are available at the https://github.com/IBM/charts

Helm Security Considerations

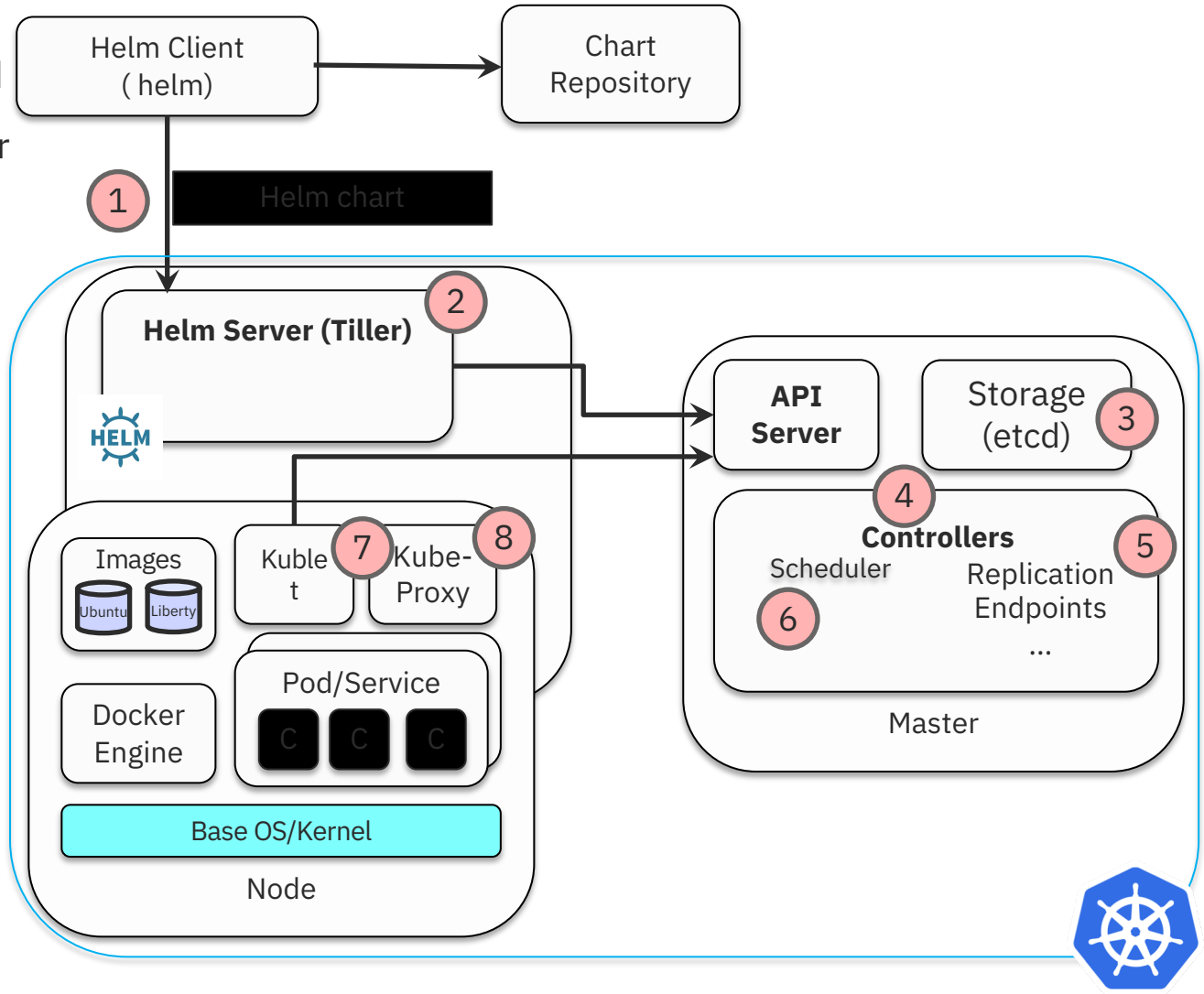
The main difference between non-secure and secure install is security of Tiller or Helm server

Four main areas to consider

- Role-based access control (RBAC)
  - Create a cluster with RBAC enabled
- Protecting the Tiller endpoint with TLS
  - Configure each Tiller gRPC endpoint to use a separate TLS certificate
- Tiller release information
  - Tiller stores its release information in ConfigMaps by default
  - Recommendation is to change default to Secrets
- Helm charts
  - Validate all software you install yourself *before* you install it.

Helm Chart Installation Flow

1. User via ”helm" installs a new chart
2. The Helm server (Tiller) will resolve chart templates and dependencies into a Kubernetes Manifest files and send a request to the Kubernetes API server. The Helm server will track the deployment as a Helm release.
3. API server receives the request and stores it in the DB (etcd)
4. Watchers/controllers detect the resource changes and act upon it
5. ReplicaSet watcher/controller detects the new app and creates new pods to match the desired # of instances
6. Scheduler assigns new pods to a kubelet
7. Kubelet detects pods and deploys them via the container runing (e.g. Docker)
8. Kubeproxy manages network traffic for the pods – including service discovery and load-balancing



Note: Steps 3 onwards are a normal Kubernetes flow

Using Helm

Some other common helm commands are

- create - create a new baseline chart with the given name
- delete - given a release name, delete the release from Kubernetes
- list - list releases of charts
- inspect - inspect a chart
- search - search for charts
- status - displays the status of the named release
- repo list – list local repositories
- version – helm version information

Run ***helm --help*** to find out all the commands