Realize your vision

Helm Past, Present, and Future

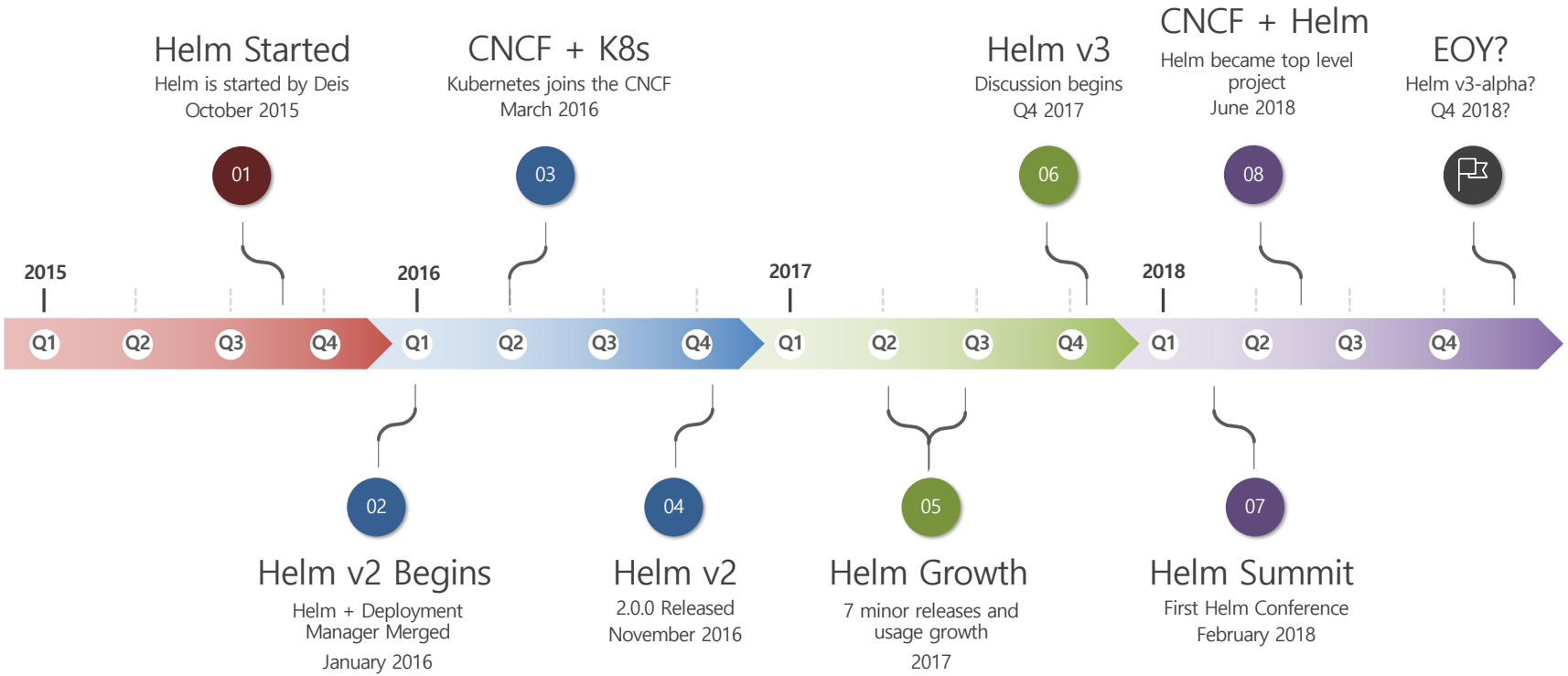
August, 2018

Matt Farina

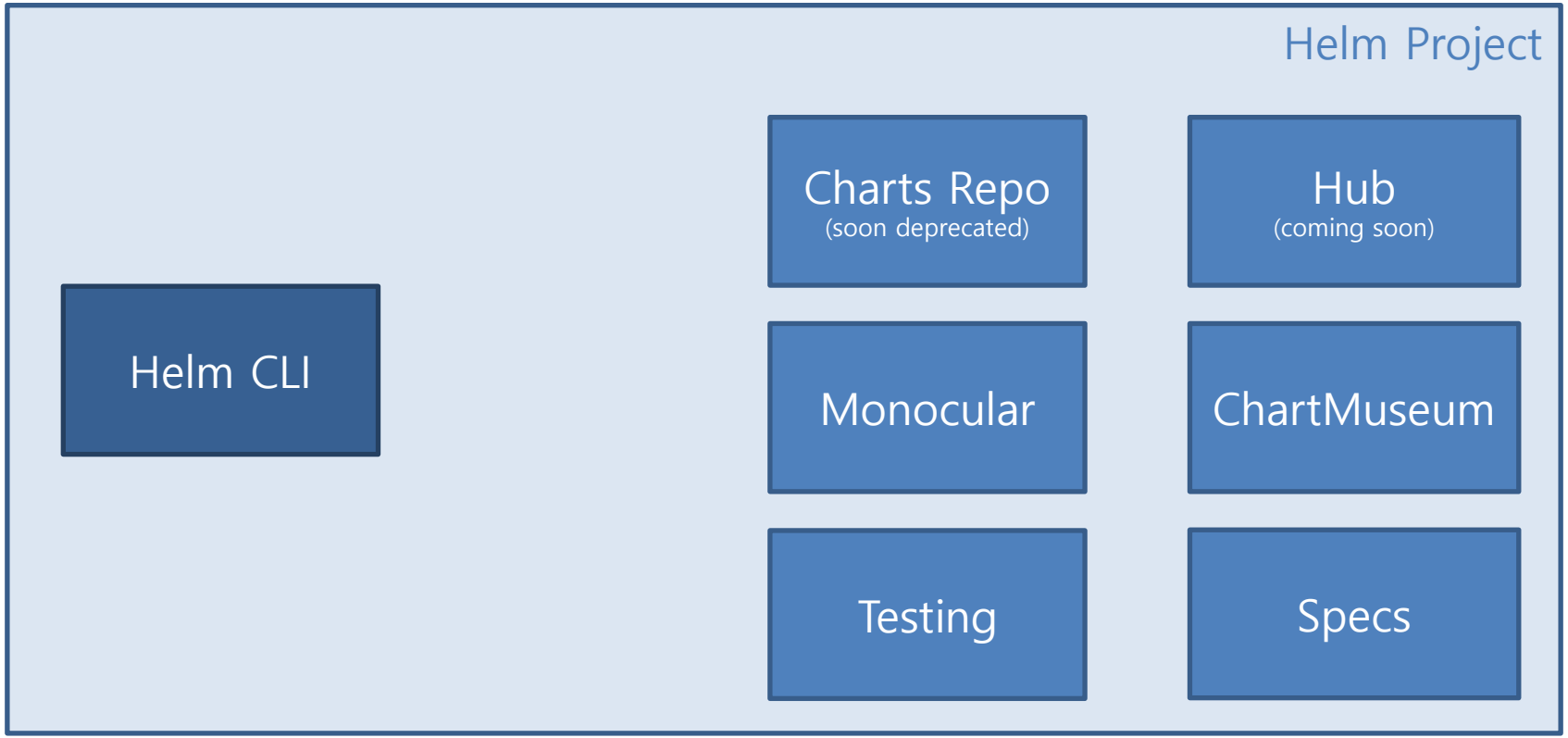
SAMSUNG SDS

Agenda

I. A Brief History of Helm
II. Helm v3 (What's Next For Helm?)
III. Charts: Kubernetes config and application business logic
IV. Repositories
V. Distributed Public Search
VI. Helm and CI/CD


Helm Started
Helm is started by Deis
October 2015
01
CNCF + K8s
Kubernetes joins the CNCF
March 2016
03
Helm v3
Discussion begins
Q4 2017
06
CNCF + Helm
Helm became top level project
June 2018
08
EOY?
Helm v3-alpha?
Q4 2018?
2015
2016
2017
2018
Q1
Q2
Q3
Q4
02
Helm v2 Begins
Helm + Deployment Manager Merged
January 2016
04
Helm v2
2.0.0 Released
November 2016
05
Helm Growth
7 minor releases and usage growth
2017
07
Helm Summit
First Helm Conference
February 2018

Helm Project



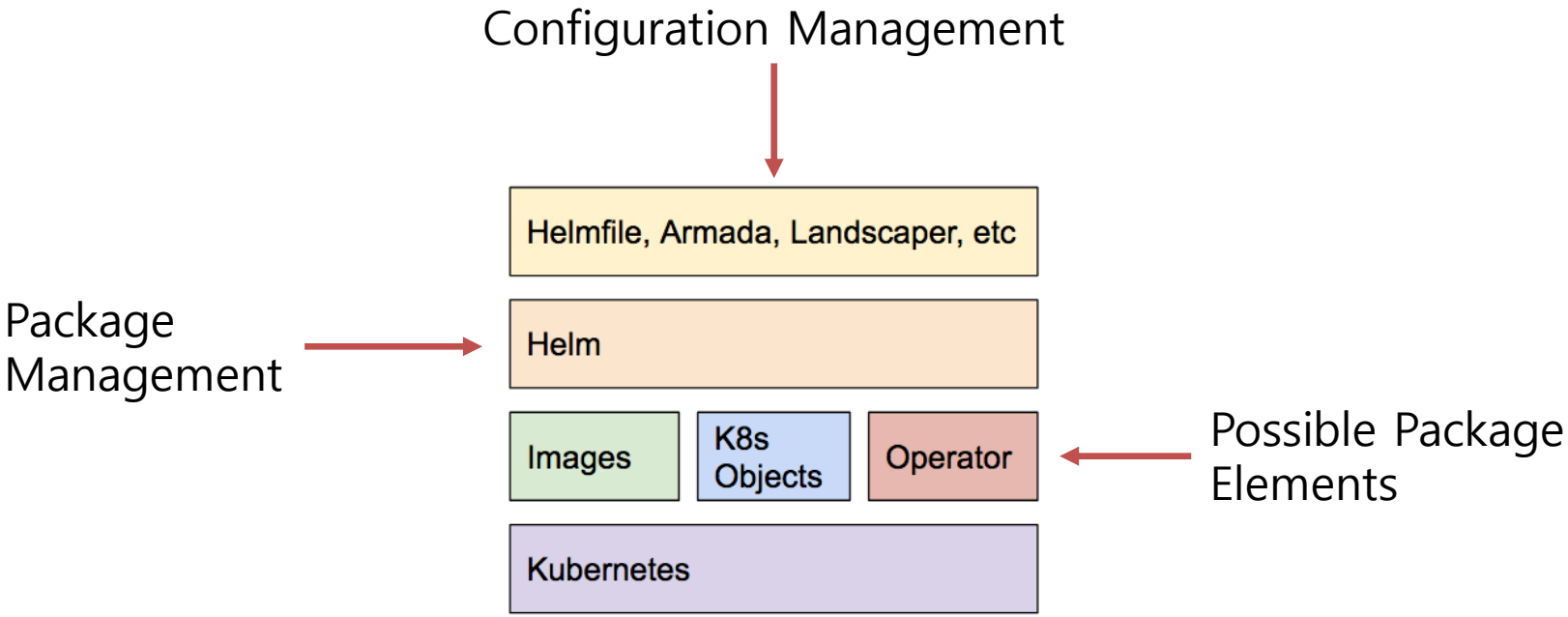
Helm User Roles

We try to understand who users of Helm are and what they do. We have documented several user roles.

In priority order:

1. **Application Operator -** takes an application and operate it within a Kubernetes cluster
2. **Application Distributor -** package an application for someone else to operate
3. **Application Developer -** writes the software for an application
4. **Supporting Tool Developer -** builds tools adjacent to Helm such as a linter, Helm plugin, or even kubectl
5. **Helm Developer -** those who develop Helm itself

Helm And Surrounding Tools



Helm v3 Plan

Major changes based on lessons learned

Changes from v2:

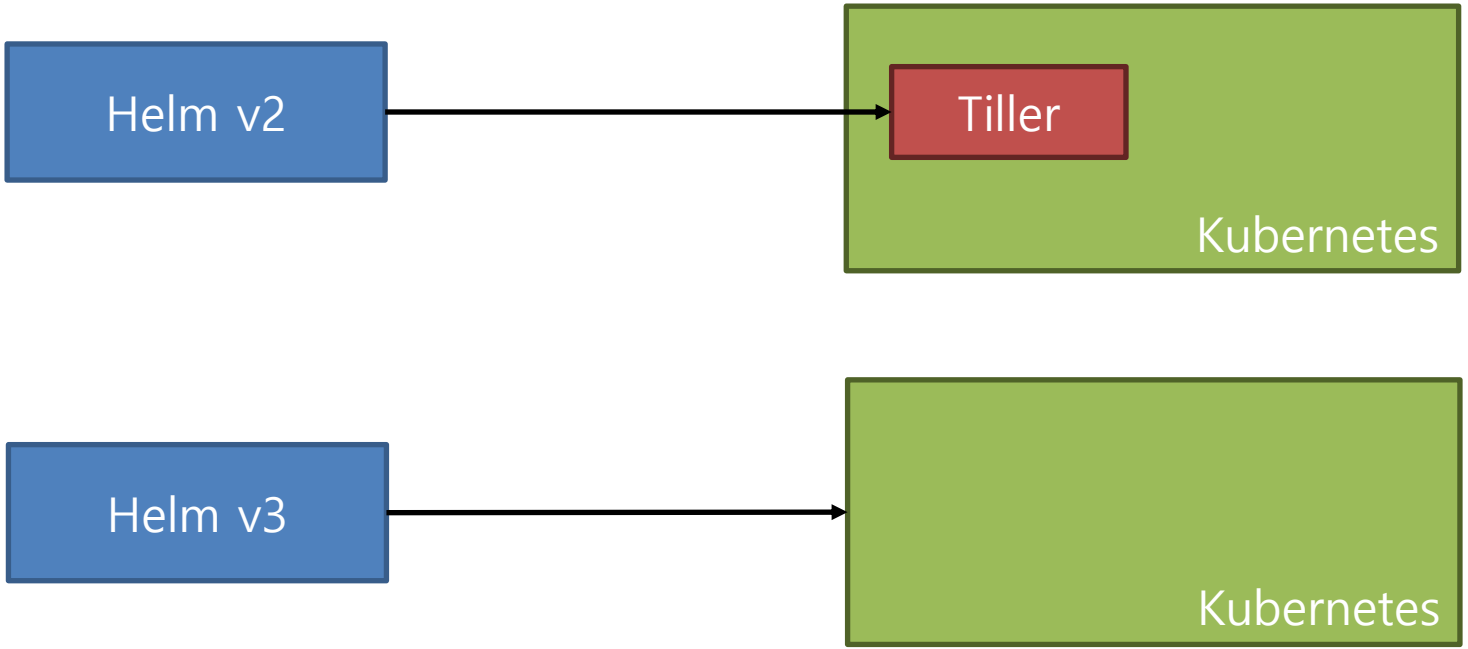
- **No More Tiller**
- **State Storage**
- **Event Driven Architecture**
- **Charts:**
  - Extensions
  - Library Charts
  - Schemas for values files
- **Hook Annotations**
- **Helm controller model**
- **Plugins:**
  - Handling Cross Platforms (like Windows)
  - Plugins in Lua
  - Easier installation
- **Repositories:**
  - No more helm serve
  - Push to repositories
  - Performance improvements

Why Helm v3?

Helm uses Semantic Versioning (semver.org) and aims to provide stable tooling for end users. That means rarely making breaking changes even though Kubernetes moves fast. Now is a time to make a breaking change to use new features and techniques.

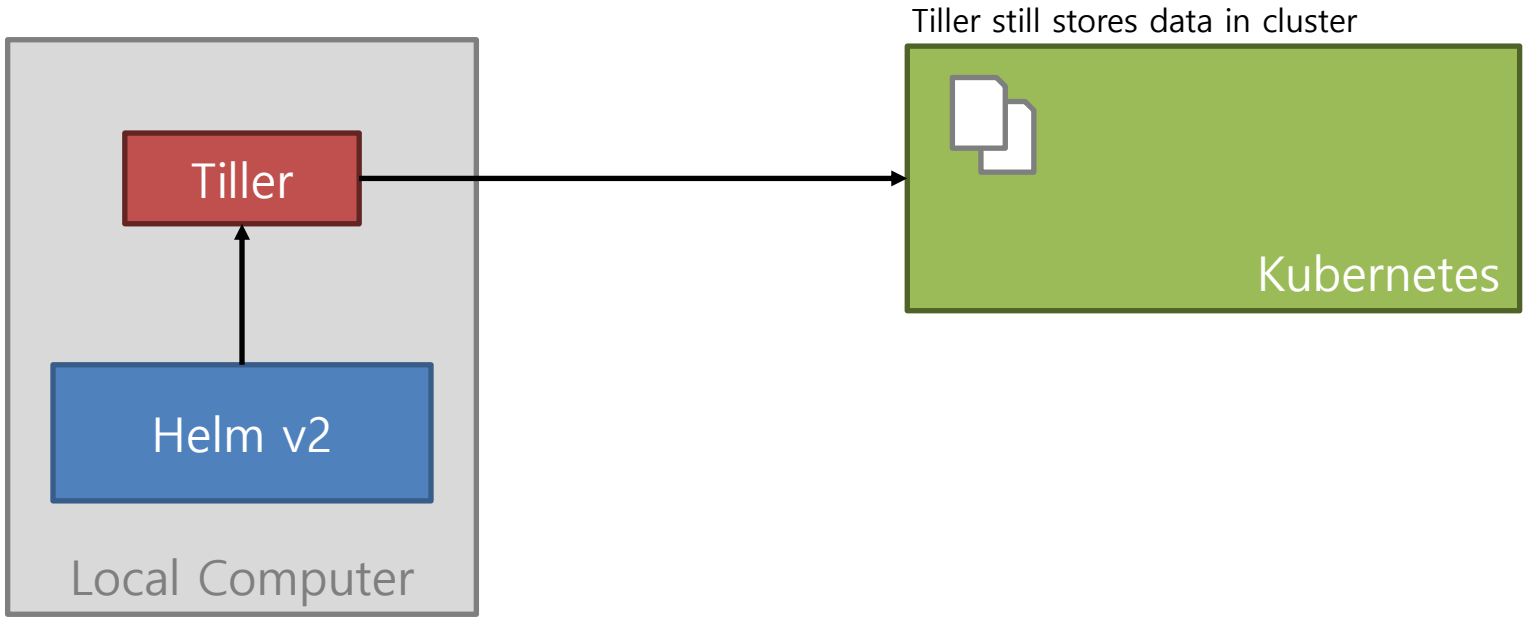
Semantic Versioning Example:

Goodbye Tiller



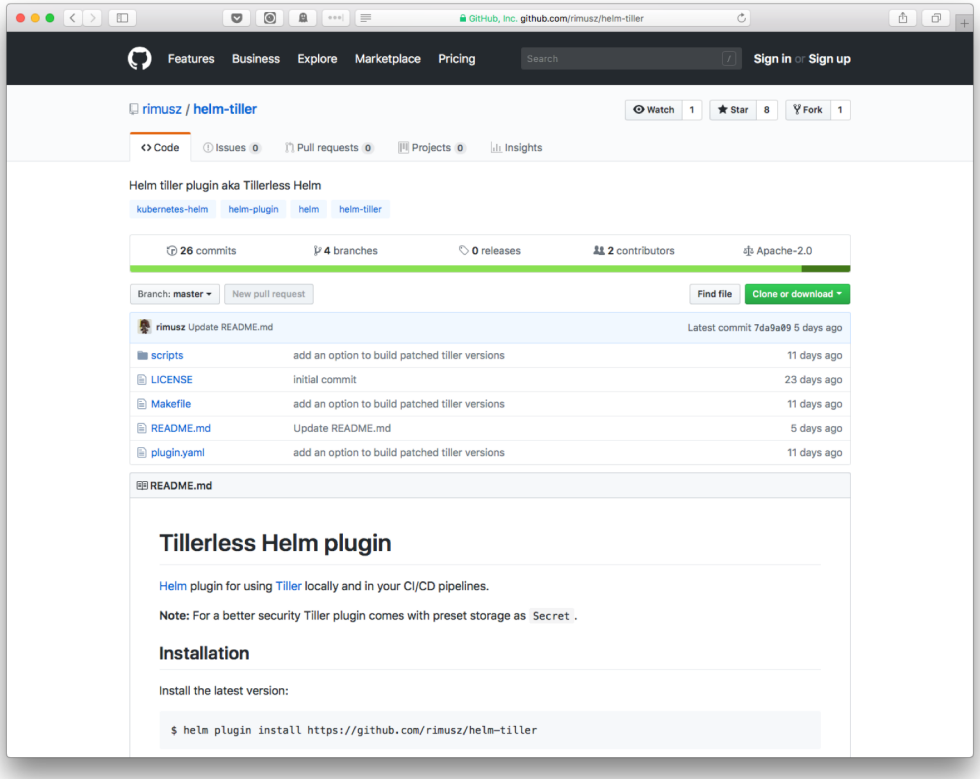
Sidebar: Local Tiller in v2

If you do not want Tiller in the cluster…



Sidebar: Local Tiller in v2

Helm plugin at https://github.com/rimusz/helm-tiller



State Storage

Moving to custom resources and secrets

Event Driven Architecture

The client-server model of Helm 2 is being replaced with an event-driven model.
High level logic will *emit events*, while event responders will *handle events*.

Many Events

These depend on the command being run and include: *pre-create, post-create, pre-delete, pre-dependency-build, post-dependency-build, pre-render, post-render, pre-install, pre-lint, pre-rollback, post-template*...

... and many others

Access to data including:

- Chart data
- Values
- Capabilities
- Files
- Templates
- Dependencies

Charts: Extensions

In a chart the `ext/` directory is reserved for extensions. Lua scripts are placed in `ext/lua`. When a chart is loaded, the script `chart.lua` in `ext/lua` will also be loaded.

A simple example:

```
function init(events) {
  -- Initialize subcharts
  subchart.init(events)

  -- Do other stuff
  events.on("pre-load", function () {
    print("pre-load event")
  })
}
```

The Lua API is still under development

Charts: Extensions – Why Lua?

JavaScript and Java are the most popular language. JavaScript can be used in many places. There are many languages. Why Lua?

Easy To Embed

Requirements:

- Interpreter embedded in Helm
  (Do not rely extra system software)
- Cross platform Helm binaries
  (Windows, macOS, Linux)

Lua is a lightweight language designed primarily for embedded use in applications. Lua was designed for this!

Permission Scheme

The `ext/permissions.yaml` file:

```
lua:
    - network
    - io
```

The Helm CLI will ask for permission to use these libraries. Only permissible libs will be imported.

Charts: Library Charts

Helm 3 supports a class of chart called a "library chart". This is a chart that is shared by other charts, but does not create any release artifacts of its own. Template definitions and extensions can be shared using library charts.

Library charts are noted in the `library:` directive in the `requirements.yaml`:

```
requirements:
  - name: apache
    version: 1.2.3
    repository: http://example.com/charts
  - name: mysql
    version: 3.2.1
    repository: http://another.example.com/charts
libraries:
  - name: common
    version: "^2.1.0"
    repository: http://another.example.com/charts
```

Charts: Schemas for values.yaml files

Schemas enable us to validate values, generate forms, and know other details about the data that can be passed into a chart.

```
title: Values
type: object
properties:
    name:
        description: Service name
        type: string
    protocol:
        type: string
    port:
        description: Port
        type: integer
        minimum: 0
    image:
        description: Container Image
        type: object
            properties:
                repo:
                    type: string
                tag:
                    type: string
required:
    - protocol
    - port
```

Example schema stored in a `values.schema.yaml` file

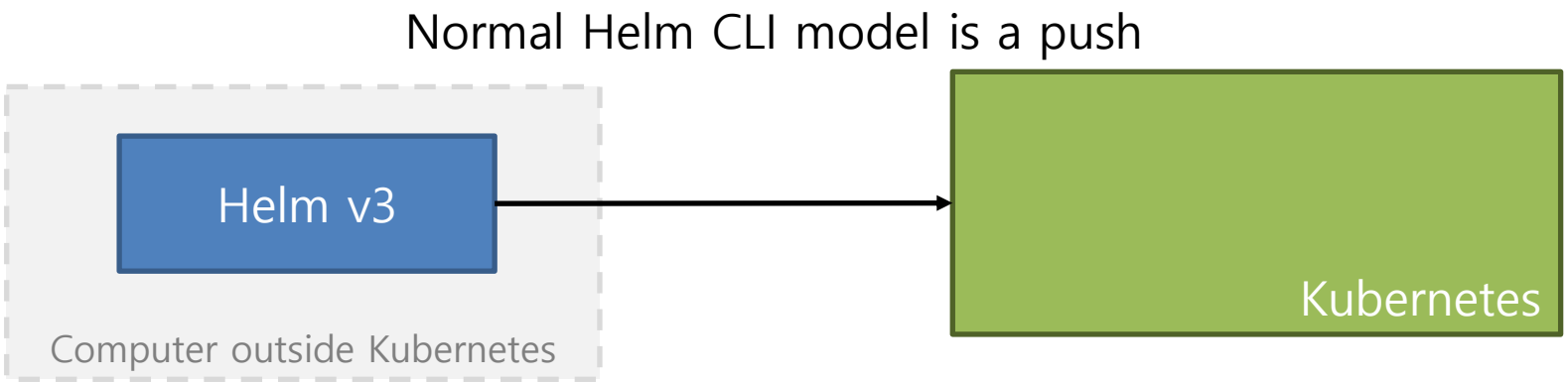
Hook Annotations

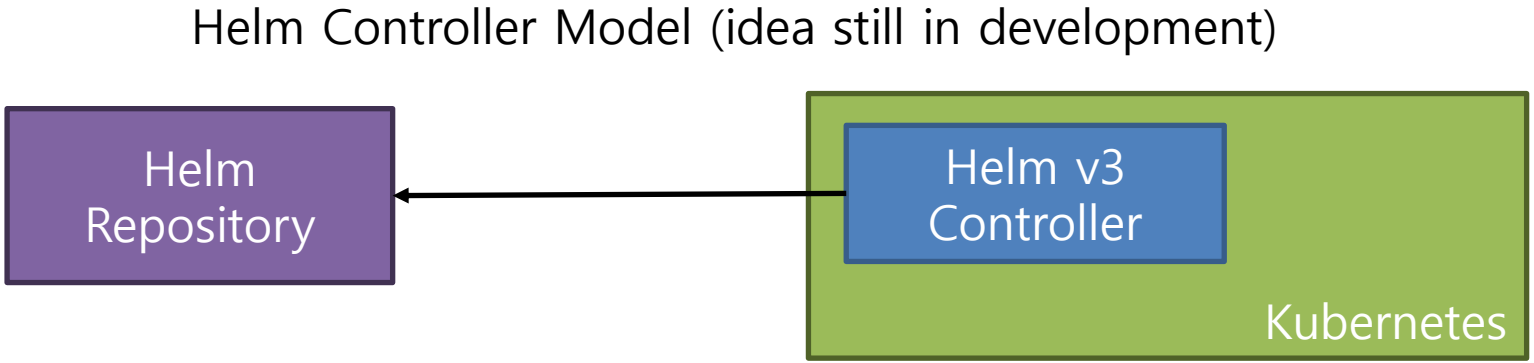
Hooks in Helm v2 were unmanaged. In Helm v3 they will have owner references set and be removed when the chart is deleted.

```
apiVersion: v1
kind: Secret
metadata:
  name: {{ template "tensorflow-notebook.fullname" . }}
  labels:
    app: {{ template "tensorflow-notebook.name" . }}
    chart: {{ template "tensorflow-notebook.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
  annotations:
    "helm.sh/hook": pre-install,pre-upgrade
type: Opaque
data:
password: {{ .Values.jupyter.password | b64enc | quote }}
```

The pre-install and pre-upgrade hook are set

Helm Controller Model





Plugins: Handling Cross Platforms (like Windows)

Not all platforms work the same or even have the same flags for the same utilities. For example, some Linux and macOS/BSD commands are different.

```
name: "last"
version: "0.1.0"
usage: "get the last release name"
description: "get the last release name"
command: "$HELM_BIN --host $TILLER_HOST list --short --max 1 --date -r"
# New part:
platformCommand:
  - os: linux
    arch: i386
    command: "$HELM_BIN list --short --max 1 --date -r"
  - os: windows
    arch: amd64
    command: "$HELM_BIN list --short --max 1 --date -r"
```

Plugins: Lua

The Helm Lua interpreter available to extend charts will also be available to run Lua plugins enabling cross platform scripts as plugins.

Plugins: Easier Installation

Install sets of plugins defined in a file

```
helm init --plugins <file.yaml>
```

An Example Plugins file:

```
plugins:
- name: helm-template
  url: https://github.com/technosophos/helm-template
- name: helm-value-store
  url: https://github.com/skuid/helm-value-store
- name: helm-diff
  url: https://github.com/databus23/helm-diff
```

Repositories: No more helm serve

This feature is being removed due to lack of users and other workflows being more popular

```
$ helm serve
Regenerating index. This may take a moment.
Now serving you on 127.0.0.1:8879
```

Run a local Helm repository

Repositories: Push to repositories

Enable charts to be pushed to repositories, without a plugin, such as ChartMuseum, object storage (e.g., S3), and other API compliant systems such as Artifactory.

```
$ helm login https://repo.example.com
$ helm push mychart-0.1.0.tgz https://repo.example.com
```

The exact commands are still being worked out

Repositories: Performance improvements

Repositories `index.yaml` broken into chart JSON files and an `index.json` file. This will enable smaller downloads, less information passing on change, and less memory usage. Better performance will enable repositories to scale larger.


```
{
  "apiVersion": "v2",
  "entries": {
    "artifactory": {
      "ref": "https://kubernetes-charts-incubator.storage.googleapis.com/artifactory.json",
      "stable": {
        "created": "2017-07-06T01:33:50.952Z",
        "description": "Universal Repository Manager supporting all major packaging formats,₩nbuild tools and CI servers.",
        "digest": "249e27501dbfe1bd93d4039b04440f0ff19c707ba720540f391b5aefa3571455",
        "home": "https://www.jfrog.com/artifactory/",
        "icon": "https://raw.githubusercontent.com/JFrogDev/artifactory-dcos/master/images/jfrog_med.png",
        "keywords": [
          "artifactory",
          "jfrog"
        ],
        "maintainers": [
          {
            "email": "[redacted]",
            "name": "[redacted]"
          }
        ],
        "name": "artifactory",
        "sources": [
          "https://bintray.com/jfrog/product/JFrog-Artifactory-Pro/view",
          "https://github.com/JFrogDev"
        ],
        "urls": [
          "https://kubernetes-charts-incubator.storage.googleapis.com/artifactory-5.2.0.tgz"
        ],
        "version": "5.2.0"
      }
    }
  }
}
```


An example `index.json` file

Charts: Kubernetes Config And Application Business Logic

Charts In The Beginning

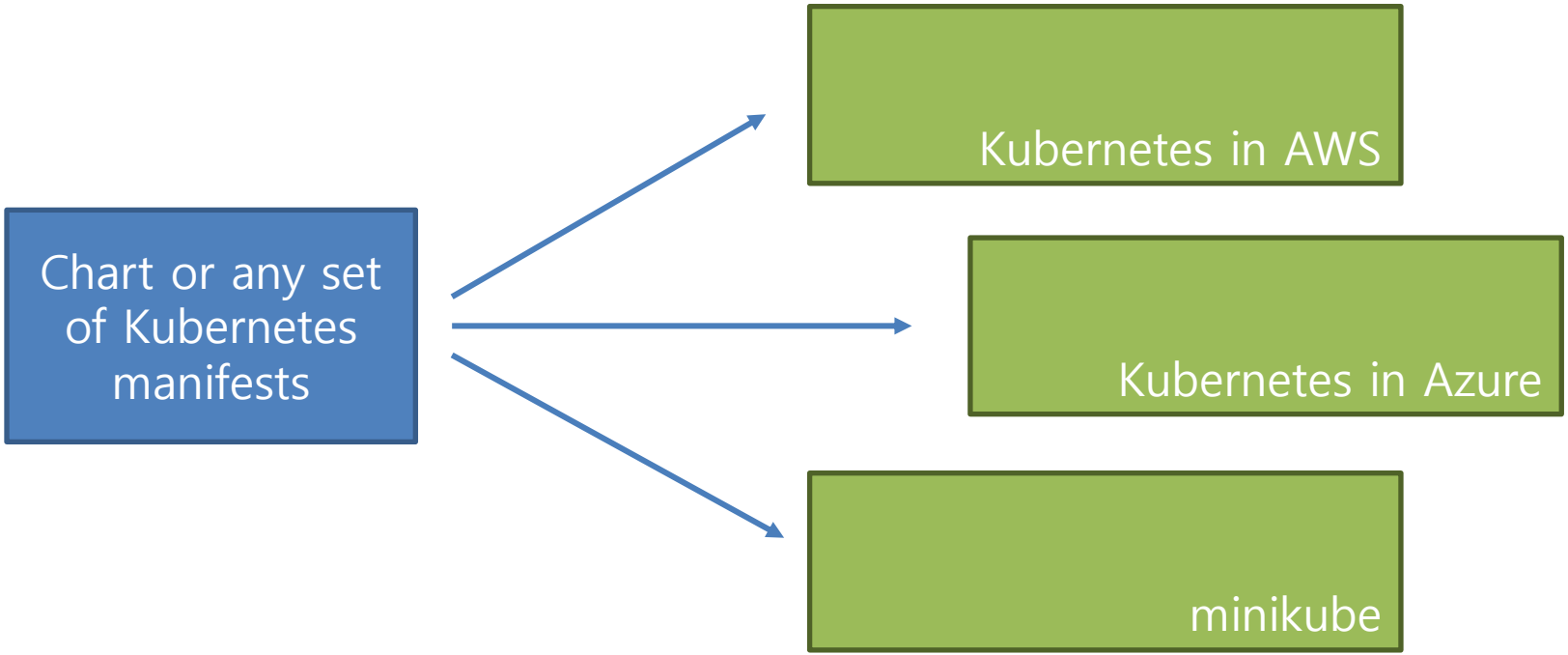
Simple replacements in Kubernetes resource YAML files

An example snippet from phpbb in November 2016:

```
apiVersion: extensions/v1beta1
kind: Deployment
…
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: {{ template "fullname" . }}
    spec:
      containers:
      - name: {{ template "fullname" . }}
        image: "{{ .Values.image }}"
        imagePullPolicy: {{ default "" .Values.imagePullPolicy | quote }}
…
```

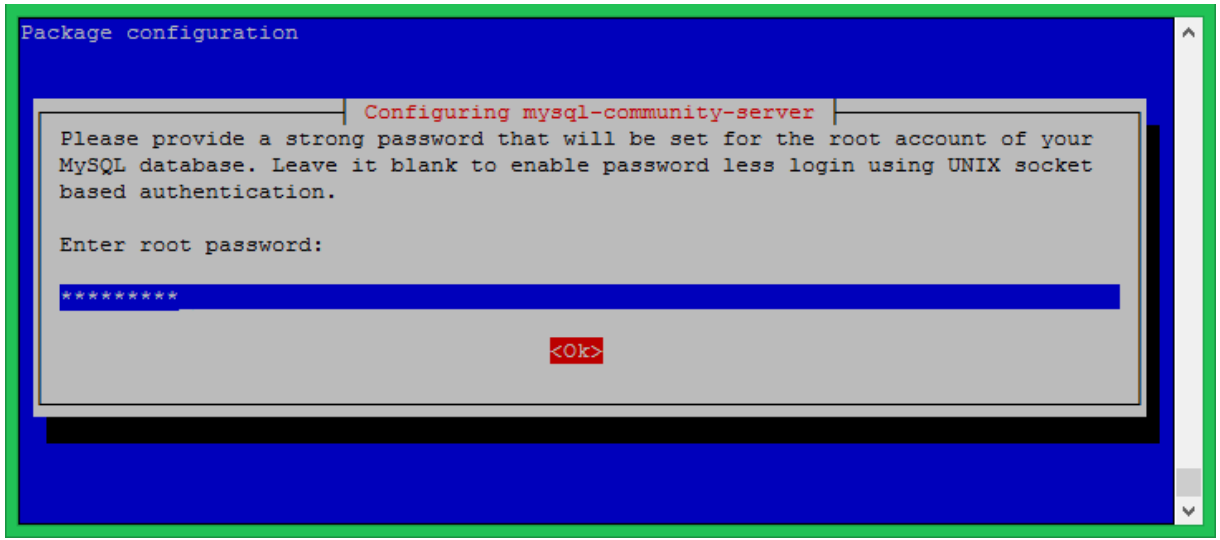
First Problem: Deploying To Different Environments

How can the same package be deployed to different environments with different options (e.g., some places have a load balancer and some do not)



Second Problem: Collecting Application Information

Applications need details at installation time that relate to an application. How can that be collected and passed on with a good experience?



MySQL collecting the root password at install time

Solution: Logic In Charts

Chart developers started, on their own accord, adding logic to charts to handle environments and to collect and react to application configuration

```
{{- if .Values.ingress.enabled }}
{{- range .Values.ingress.hosts }}
apiVersion: extensions/v1beta1
kind: Ingress
…
---
{{- end }}
{{- end }}
```

Example where Chart captures host names and if ingress is enabled. Then builds service definitions as appropriate

Chart Repositories

Many Distributed Chart Repositories

Similar to Debian APT and Homebrew with Taps in distributed style

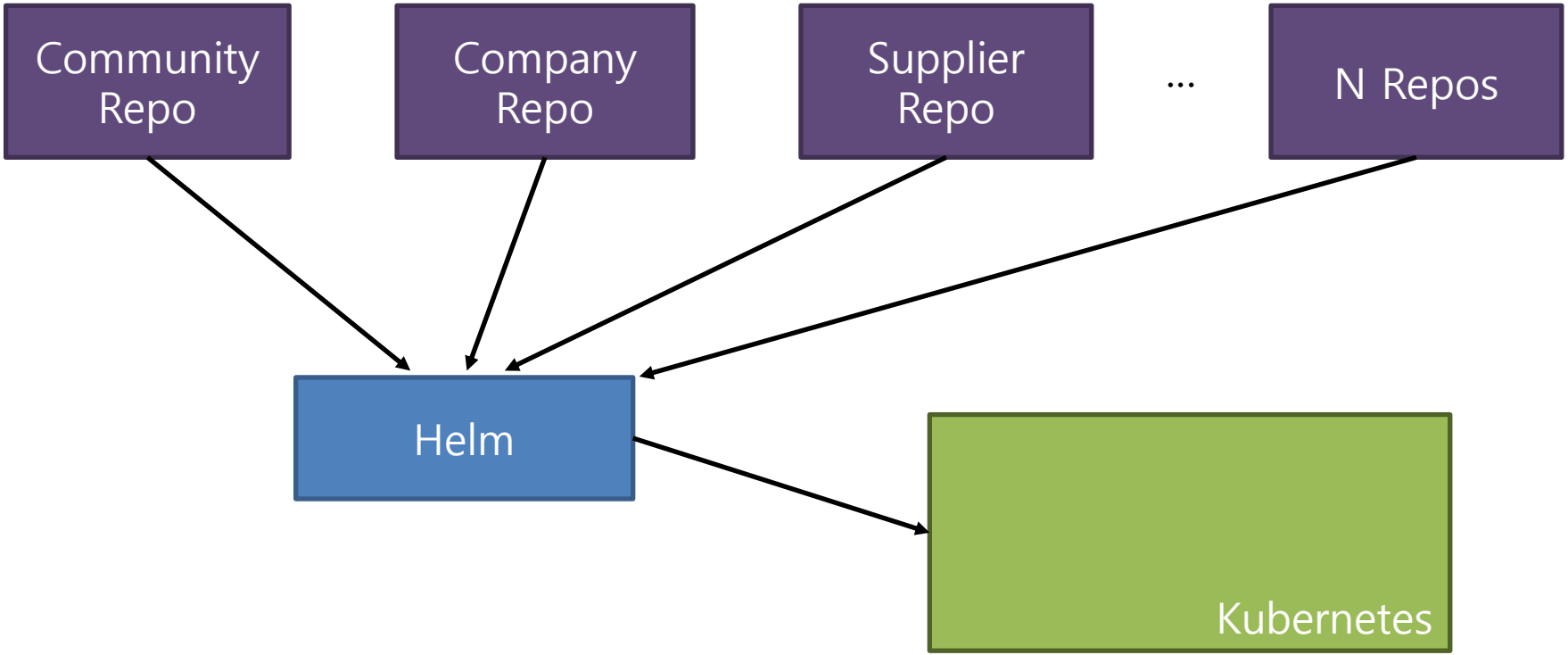


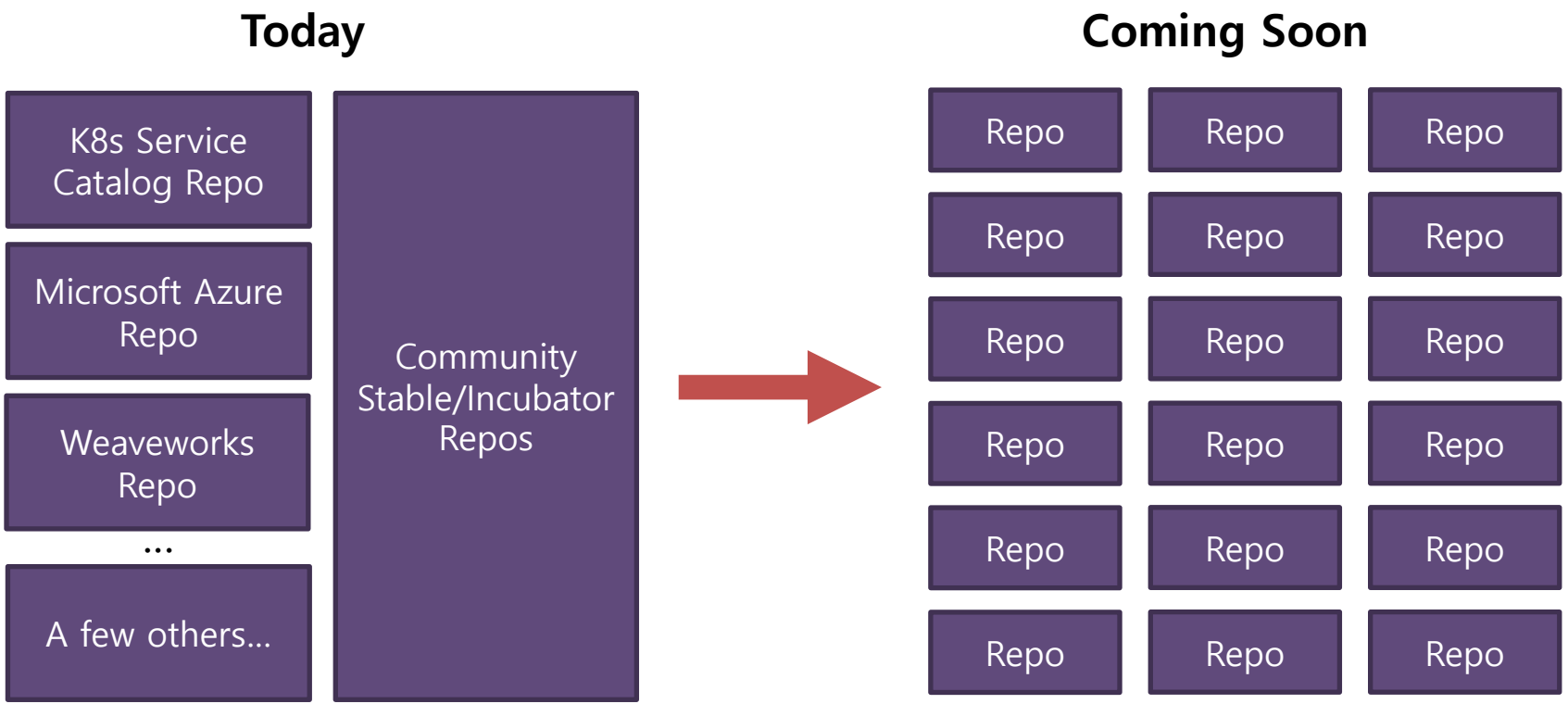
Chart Repository Tools & Services

- **Helm CLI** – Can generate a static repository to use with object storage or a static file server
- **ChartMuseum** – A server side repository service that integrates with object storage from major providers for storage
- **Artifactory**, by jFrog – Can act as a chart repository in addition to repositories for other systems
- **Harbor**, a CNCF container image repository – In addition to hosting container images it can host chart repositories
- **Codefresh** – Part of their DevOps as a Service includes a Helm Registry

Distributed Public Search

Community Chart Repos To Many Public Repos

A transition is coming to shift from central hosting to more distributed repositories



Centralizing Public Search

Publicly consumable charts and repos will be able to get listed in a centralized Helm hosted search powered by Monocular