What the HELM !?

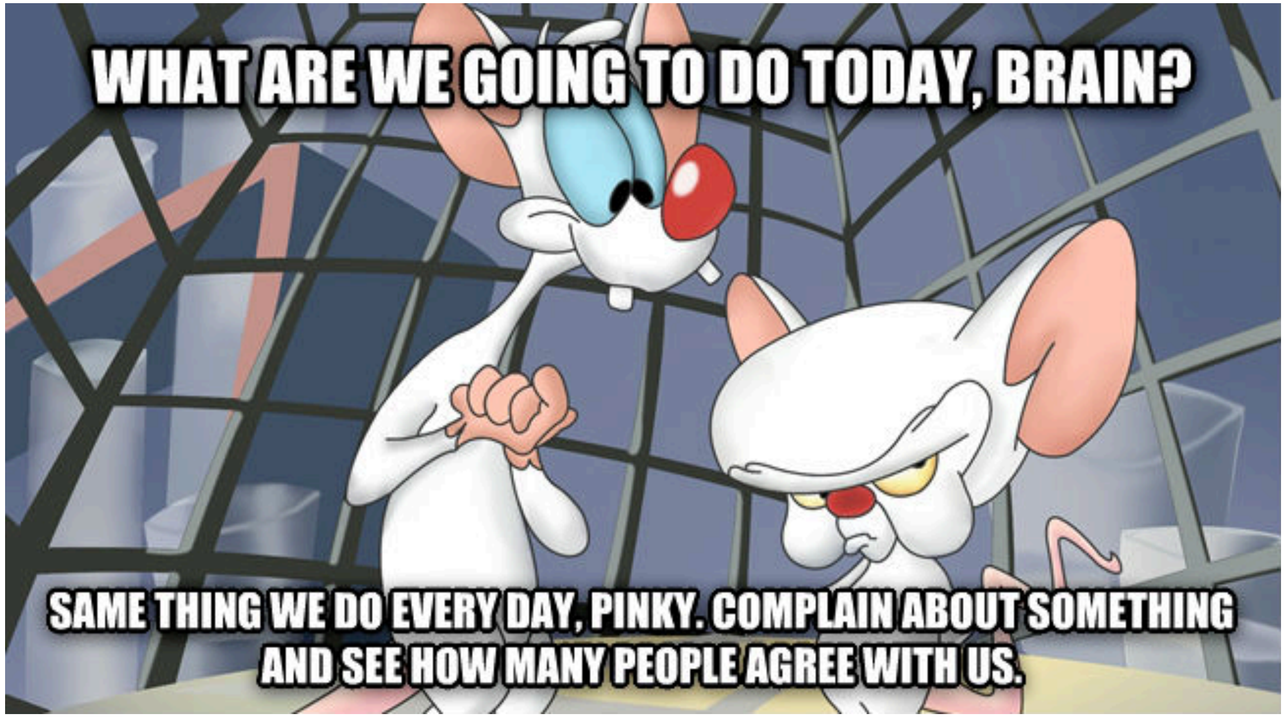
A HELM INTRODUCTION OR WHY EVERYONE SHOULD BE USING HELM

The package manager for Kubernetes

Helm is the best way to find, share, and use software built for Kubernetes.

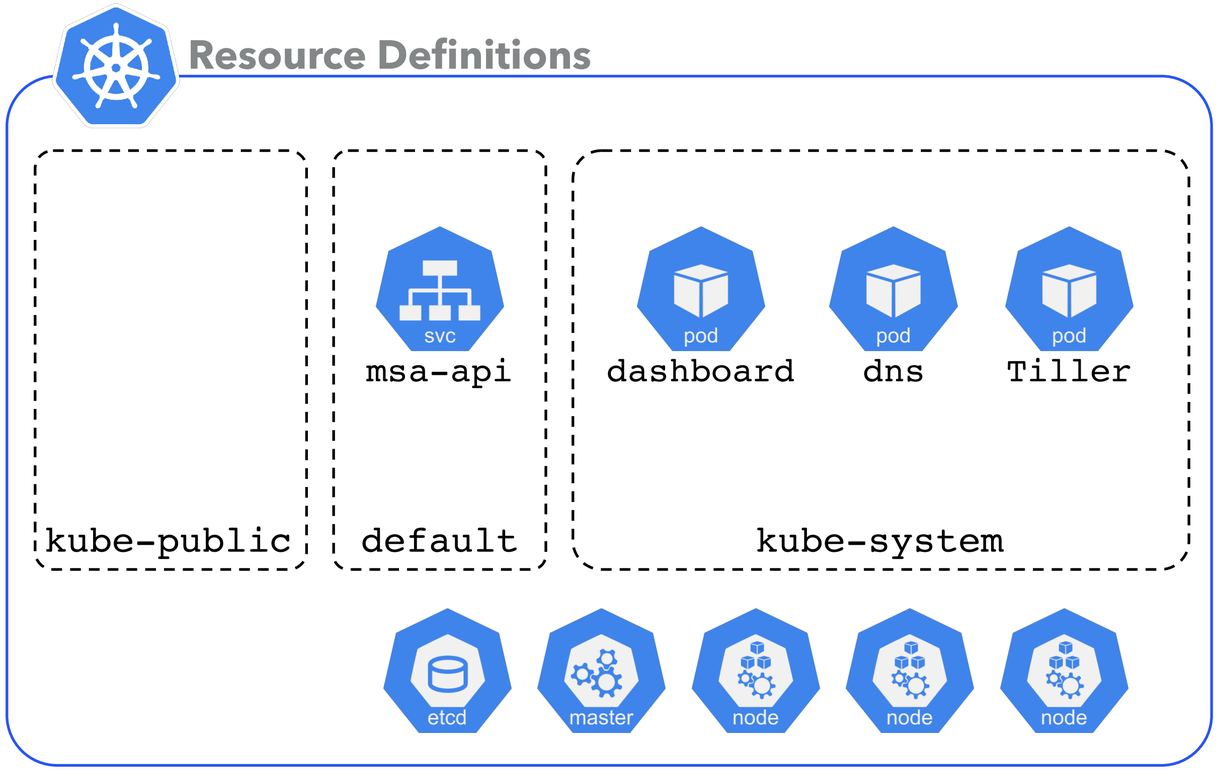
AGENDA

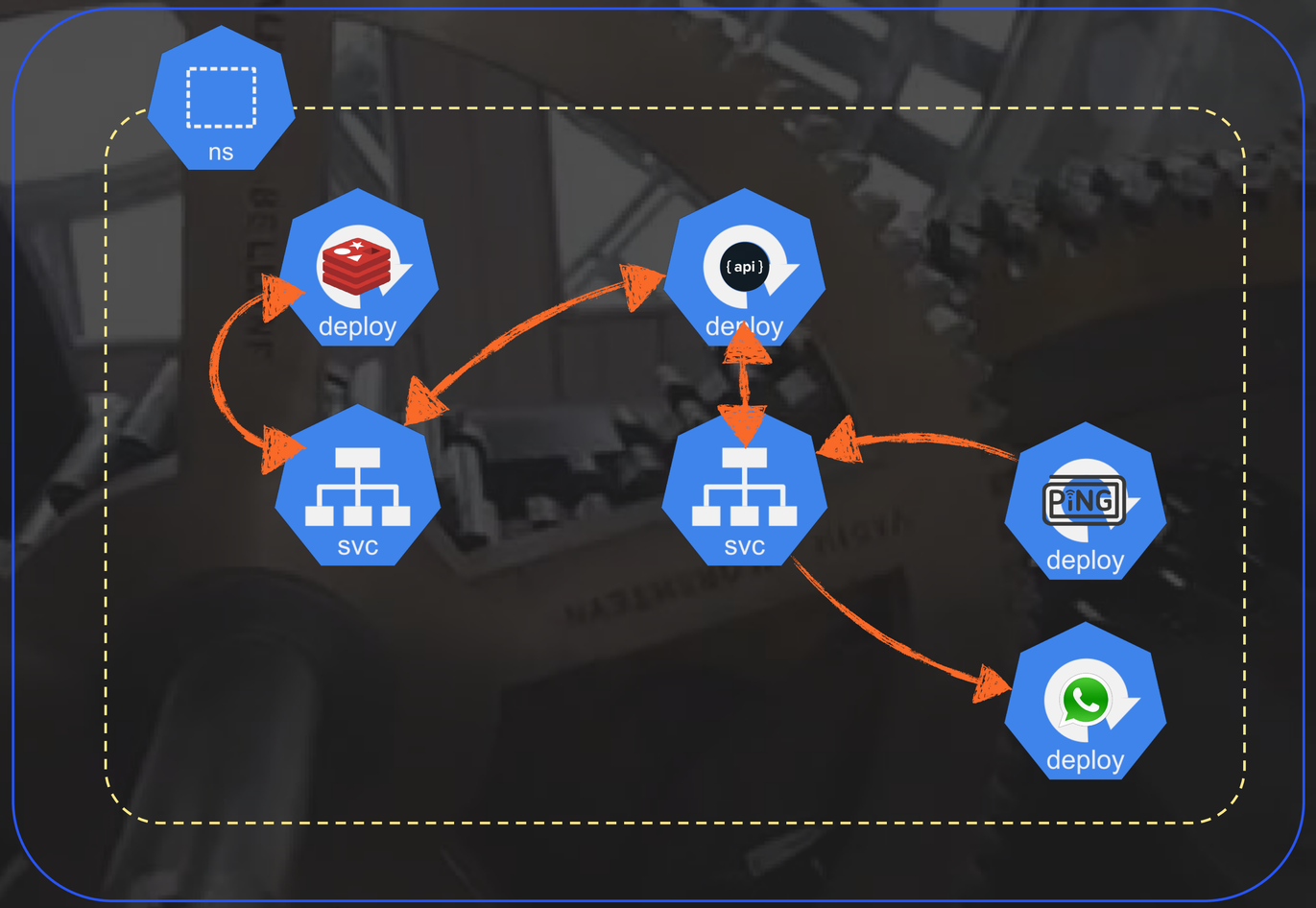
- Introduction to what helm is
  - Life before HELM
    - Intro - sample app
    - demo
  - Life with HELM
  - HELM Architecture
  - Real life examples



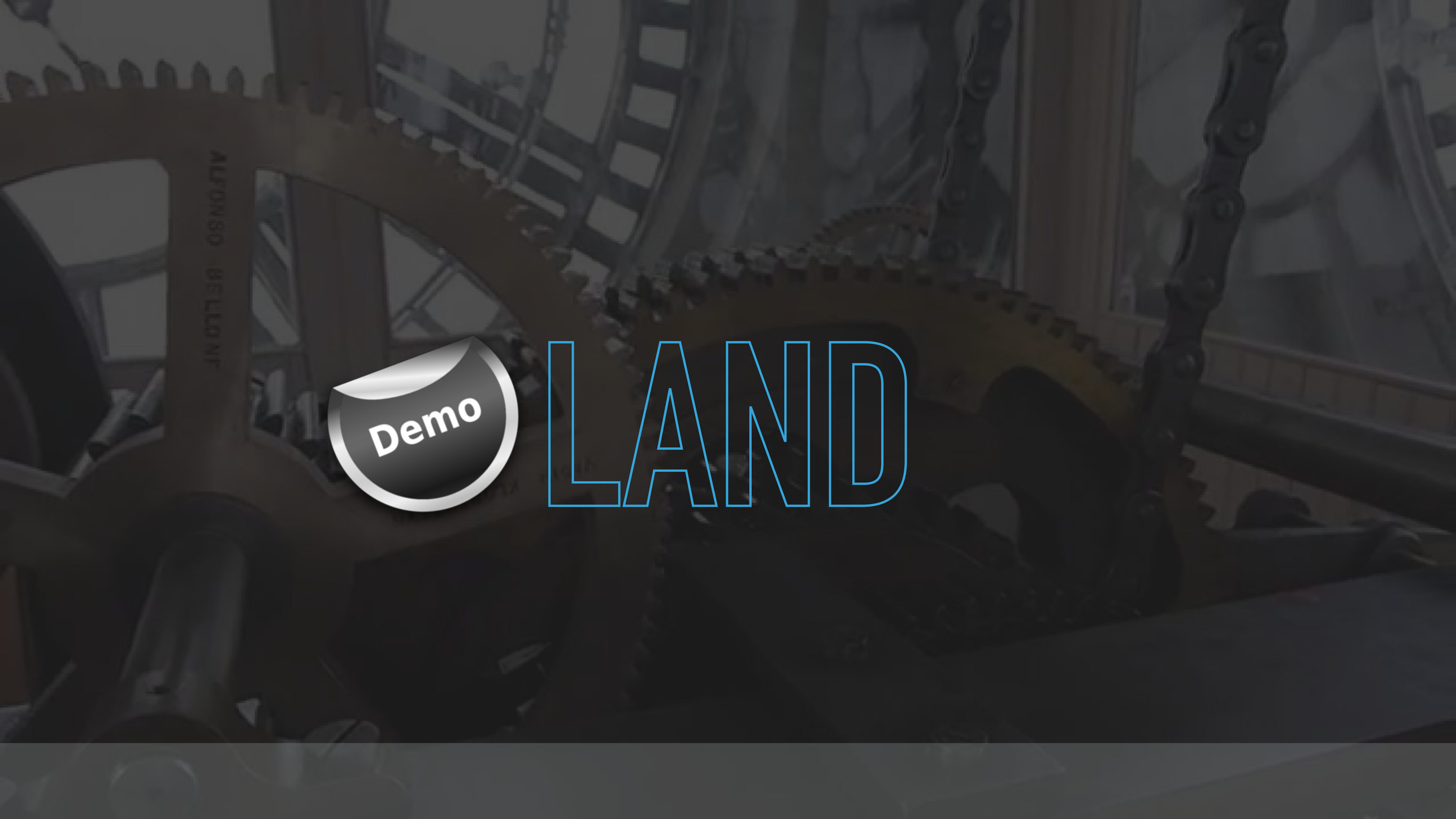
ASSUMPTIONS | PERQUISITES

- You know what kubernetes is:
  - Namespaces [link]
  - Labels & Selectors [link]
  - Deployments [link]
    - Pods [link]
  - Services [link]

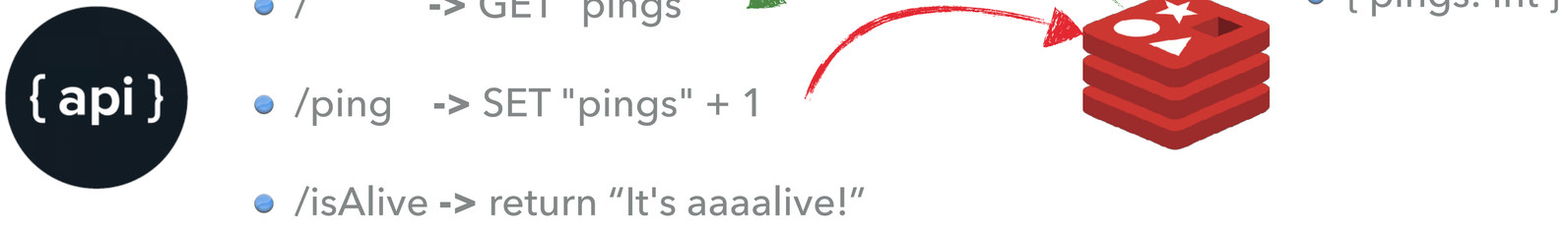




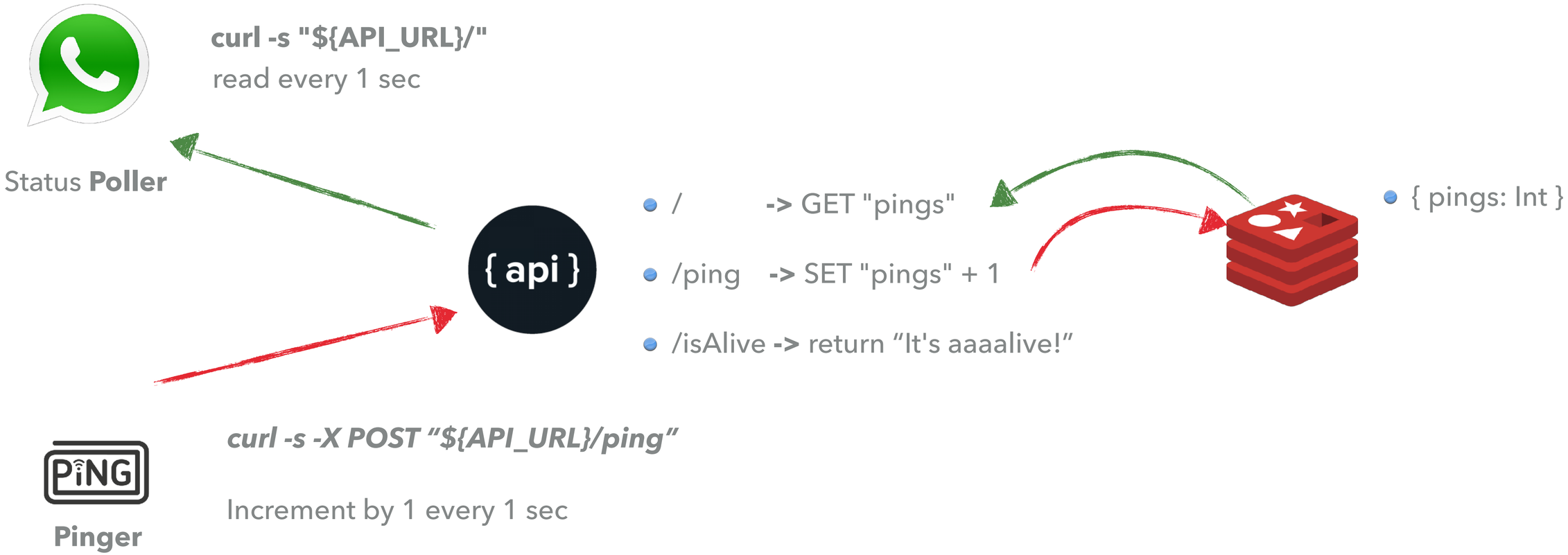
LIFE BEFORE HELM


Demo
LAND

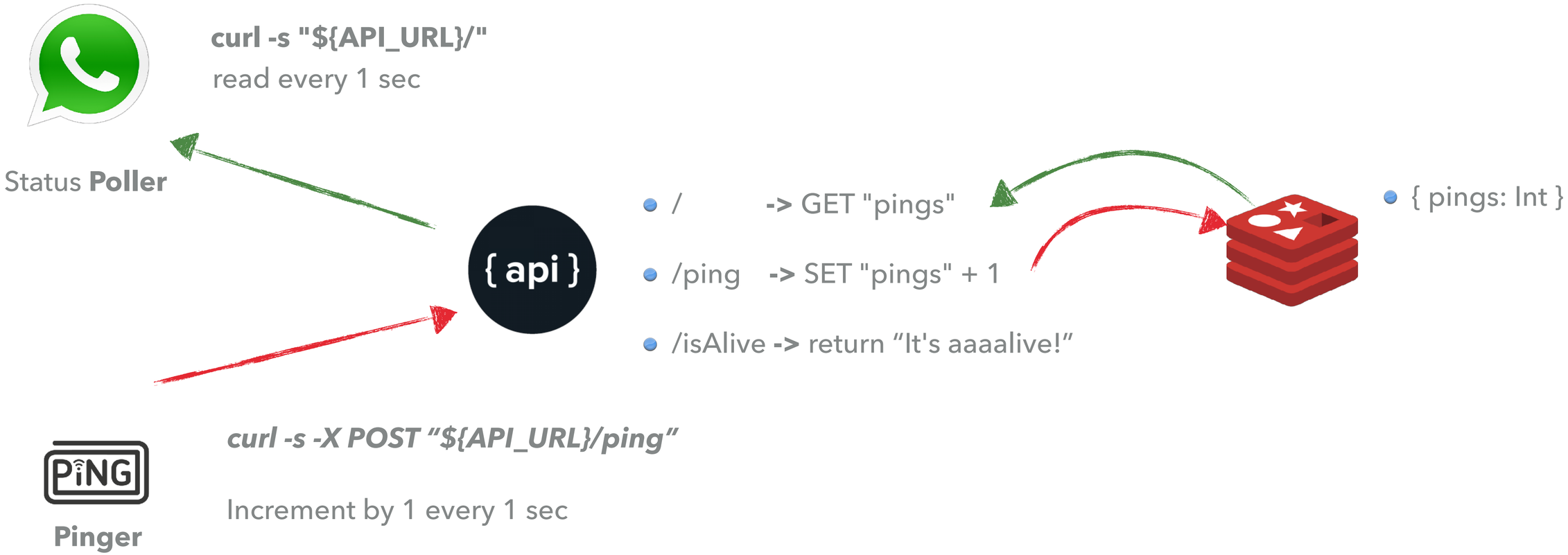
OUR “COMPLICATED” MICRO SERVICE APPLICATION



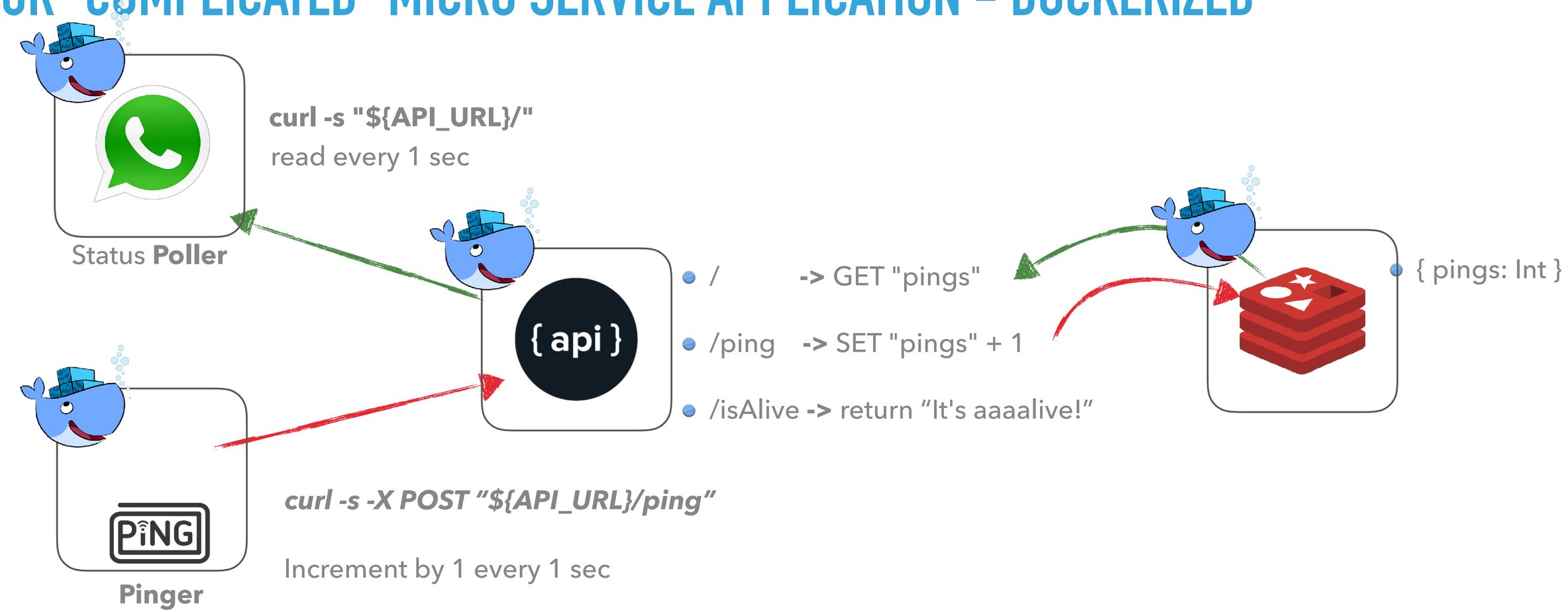
OUR “COMPLICATED” MICRO SERVICE APPLICATION



OUR “COMPLICATED” MICRO SERVICE APPLICATION



OUR “COMPLICATED” MICRO SERVICE APPLICATION – DOCKERIZED



INTRODUCTION TO HELM

PLANNING OUR DEPLOYMENT

- Each micro service has a build process which pushes the container to a docker registry.
- Docker registry is available to the k8s cluster during deployment (with or without authentication depending on the deployment)

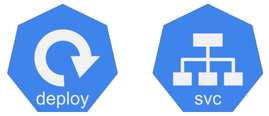
DEPLOYMENT SEQUENCE

DEPLOYMENT SEQUENCE

- ***redis*** deployment + *service* for redis

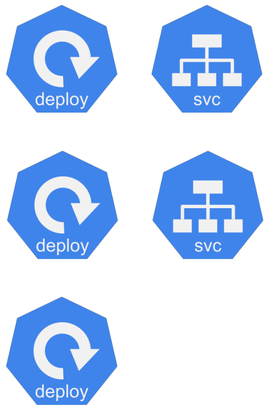
DEPLOYMENT SEQUENCE

- ***redis*** deployment + *service* for redis



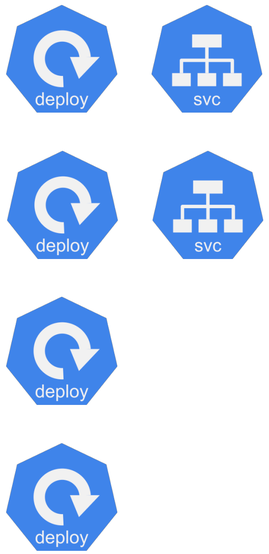
- **msa-api** deployment **+** *service* for the ***msa-api***

DEPLOYMENT SEQUENCE



- ***redis*** deployment + *service* for redis
- **msa-api** deployment **+** *service* for the ***msa-api***
- ***msa-pinger*** deployment

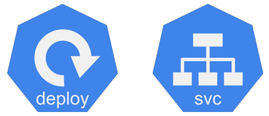
DEPLOYMENT SEQUENCE



- ***redis*** deployment + *service* for redis
- **msa-api** deployment **+** *service* for the ***msa-api***
- ***msa-pinger*** deployment
- ***msa-poller*** deployment

DEPLOYMENT SEQUENCE

- ***redis*** deployment + *service* for redis



- **msa-api** deployment **+** *service* for the ***msa-api***

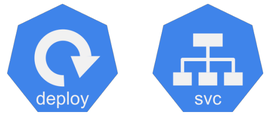
- ***msa-pinger*** deployment

- ***msa-poller*** deployment

- ***msa-demo*** namespace

DEPLOYMENT SEQUENCE

- ***redis*** deployment + *service* for redis



- **msa-api** deployment **+** *service* for the ***msa-api***

- ***msa-pinger*** deployment

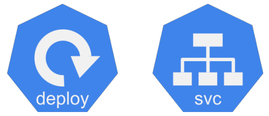
- ***msa-poller*** deployment

- ***msa-demo*** namespace

- ***persistent volume*** for redis data

DEPLOYMENT SEQUENCE

- ***redis*** deployment + *service* for redis



- **msa-api** deployment **+** *service* for the ***msa-api***

- ***msa-pinger*** deployment

- ***msa-poller*** deployment

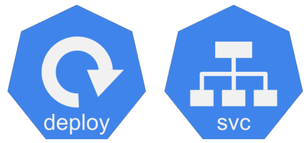
- ***msa-demo*** namespace

- ***persistent volume*** for redis data

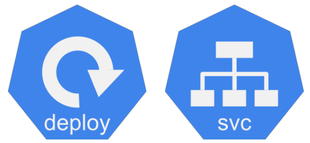
- ***configmaps / secrets*** for all components

KUBERNETES "NATIVE" WAY

- Create a namespace:
  kubectl create namespace ***msa-demo*** --dry-run -o yaml > msa-demo-ns.yml



- Create a deployment + service for redis:
  kubectl run redis --image=redis --port=***6379*** --expose --dry-run -o yaml > redis.yml



- Create a deployment + service for msa-api:
  kubectl run msa-api --image=shelleg/msa-api:config --port=***8080*** –expose --image-pull-policy=Always --dry-run -o yaml > msa-api.yml

KUBERNETES "NATIVE" WAY

- Create a deployment for msa-pinger:

  kubectl run msa-pinger --image=shelleg/msa-pinger:latest --env="API_URL=msa-api:8080" --env="DEBUG=true" --dry-run -o yaml > msa-pinger.yml

- Create a deployment for msa-poller:

  kubectl run msa-poller --image=shelleg/msa-poller:latest --env="API_URL=msa-api:8080" --dry-run -o yaml > msa-pinger.yml


HANDS ON

WE NOW HAVE A BUNCH OF FILES DESCRIBING OUR APPLICATION


```
.
|-- msa-api.yml
|-- msa-demo-ns.yml
|-- msa-pinger.yml
|-- msa-poller.yml
`-- redis.yml

0 directories, 5 files
```


```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  name: msa-api
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    run: msa-api
status:
  loadBalancer: {}
```


```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    run: msa-api
  name: msa-api
spec:
  replicas: 1
  selector:
    matchLabels:
      run: msa-api
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        run: msa-api
    spec:
      containers:
      - image: shelleg/msa-api:config
        imagePullPolicy: Always
```

SO WHY DO WE NEED HELM ?!


```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  name: msa-api
spec:
  ports:
  - port: {{ variable }}
    protocol: TCP
    targetPort: {{ variable }}
  selector:
    run: msa-api
status:
  loadBalancer: {}
```

- We want to better control “what goes where” e.g ports
- Container image versions ?! (is the only change between releases)
- Manage Dependencies

- Update components individually updating things like image versions e.g ***msa-api:latest*** -> ***msa-api:config***
- Write **once** run on any cluster
- **Reproducible & Shareable resource definition specs - TEMPLATES**

TEMPLATES

- Not a foreign concept … Jinja / erb / jsonnet

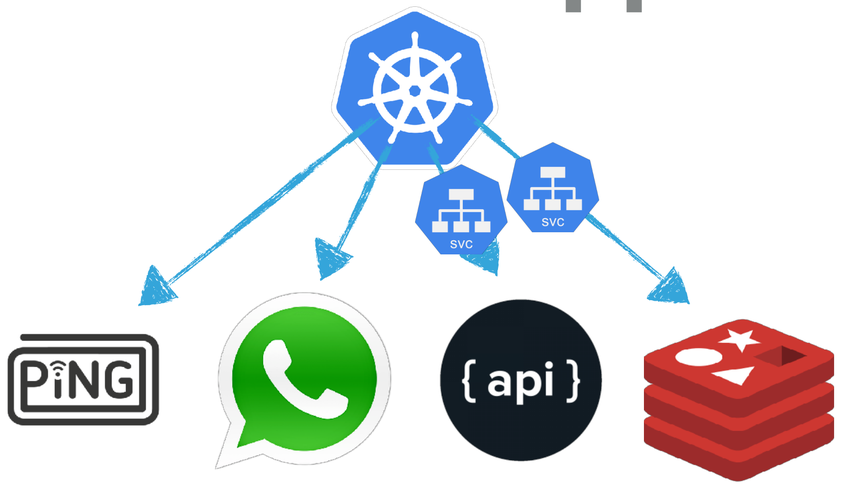

```
{% for key, svc in services.items() %}
upstream {{ key }} {
  least_conn;
{% for server in svc.delegate %}
  server {{ server }}:{{ svc.ports[0].split(':')[0] }} weight=10
max_fails=3 fail_timeout=30s;
{% endfor %}
}
{% endfor %}
```

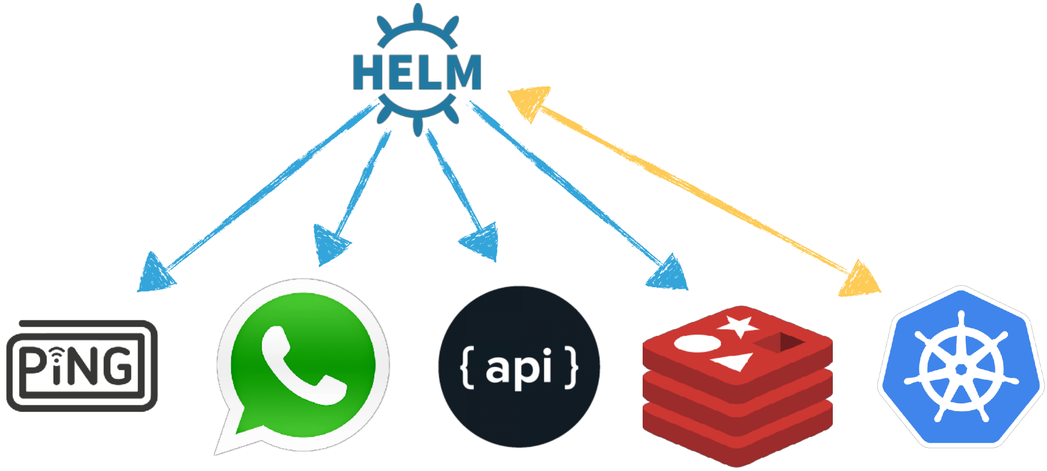
```
// A function that returns an object.
local Person(name='Alice') = {
  name: name,
  welcome: 'Hello ' + name + '!',
};
{
  person1: Person(),
  person2: Person('Bob'),
}
```

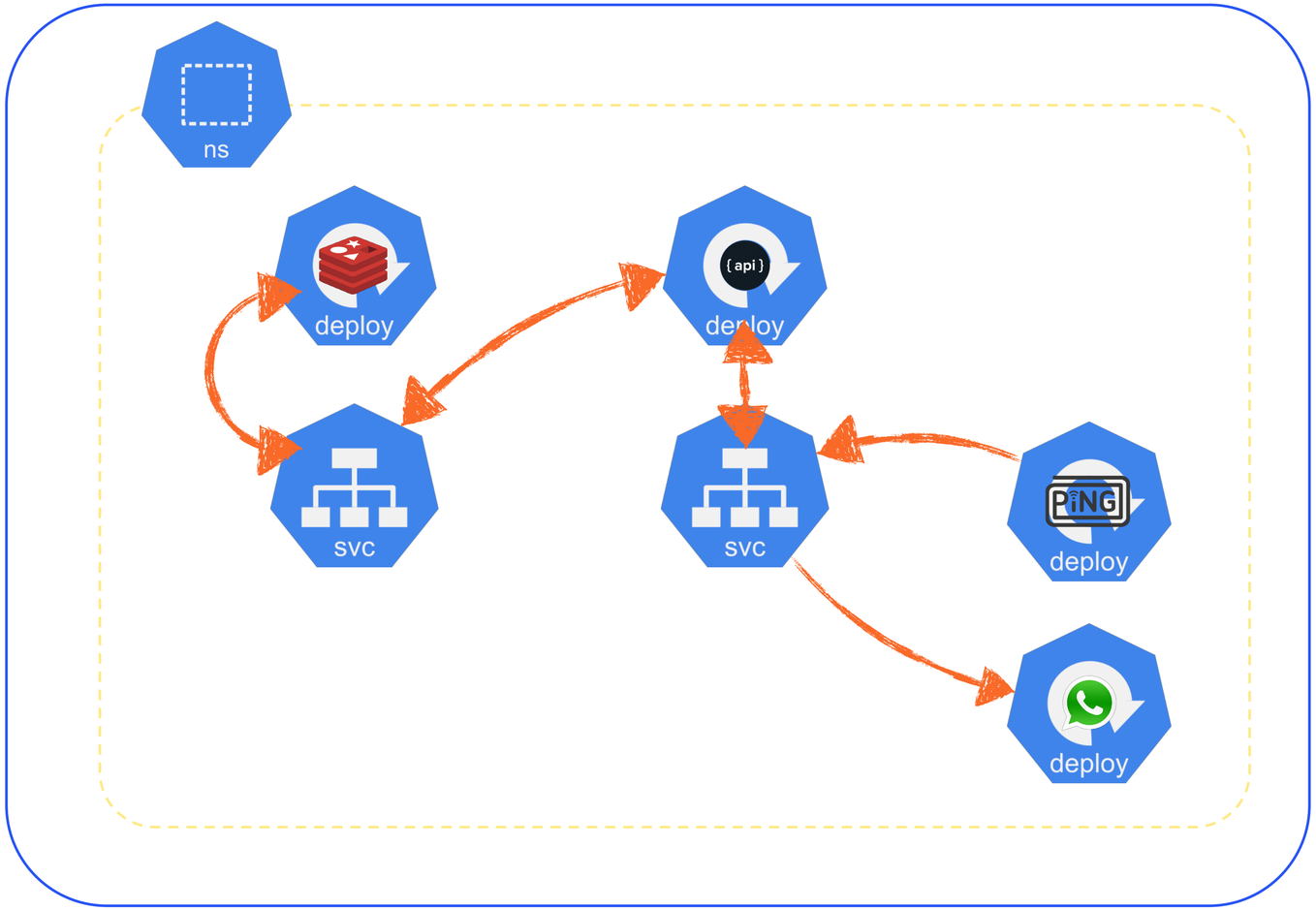
```
<% @sudoers_users.each do |user| -%>
<%= user %>    ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL
<% end -%>

# Members of the sysadmin group may gain root privileges
%sysadmin      ALL=(ALL) <%= "NOPASSWD:" if @passwordless %>ALL
```

SO WHY DO WE NEED HELM ?!

- Kubernetes native discovery mechanism is focused around the **Service** / deployment
- Helm Charts are **Application** focused







LIFE WITH HELM

HELM

IN A NUT SHELL -> HELM MANAGES CHARTS, RELEASES & VERSIONS


```
.
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   `-- service.yaml
`-- values.yaml

2 directories, 6 files
```

- Describe your deployment + service / any other kubernetes Resource Definition

```yaml
apiVersion: v1
description: My Cool Chart
engine: gotpl
name: myCoolChart
version: 0.1.0
```

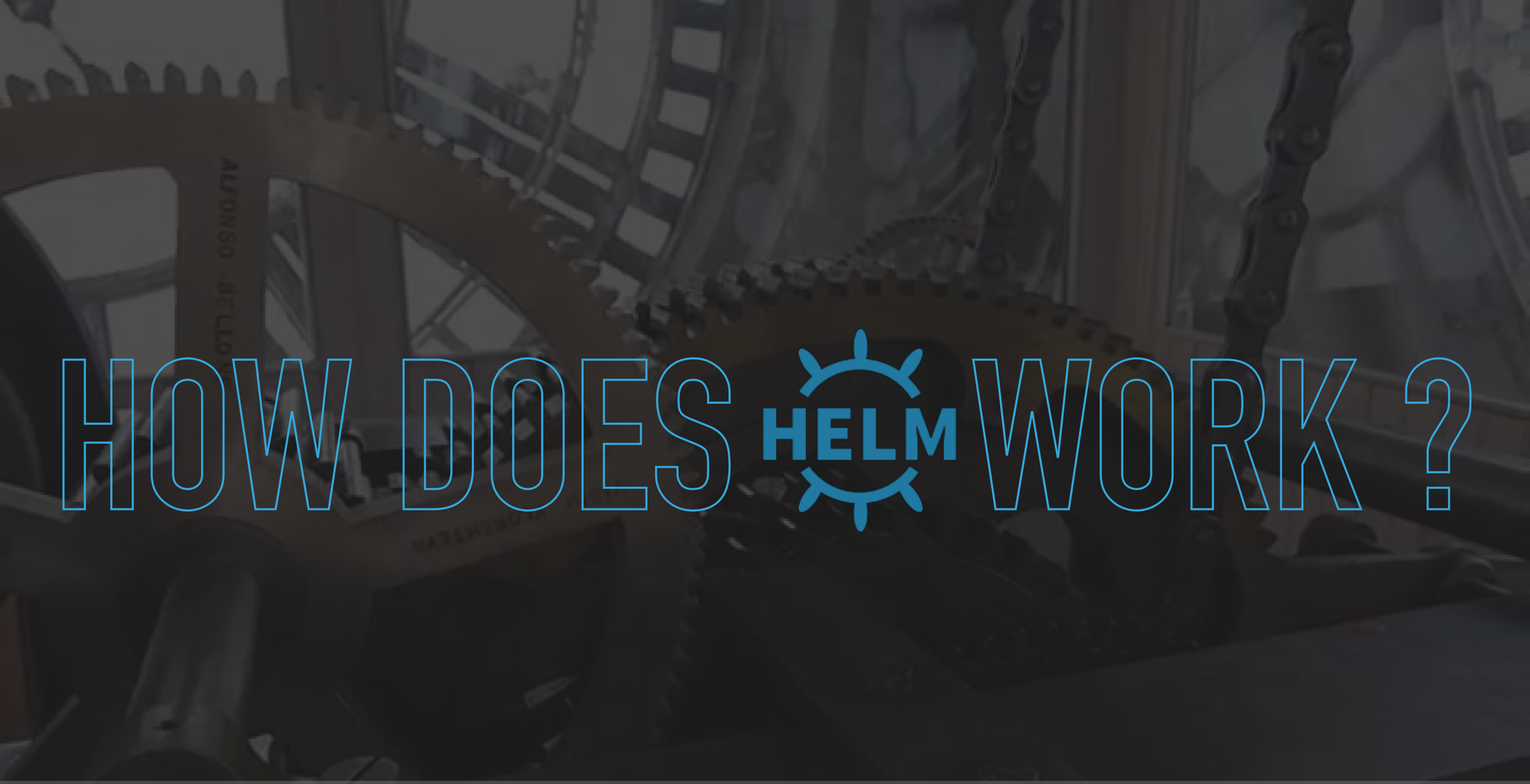
IN A NUT SHELL -> HELM MANAGES CHARTS


```
.
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   `-- service.yaml
`-- values.yaml

2 directories, 6 files
```

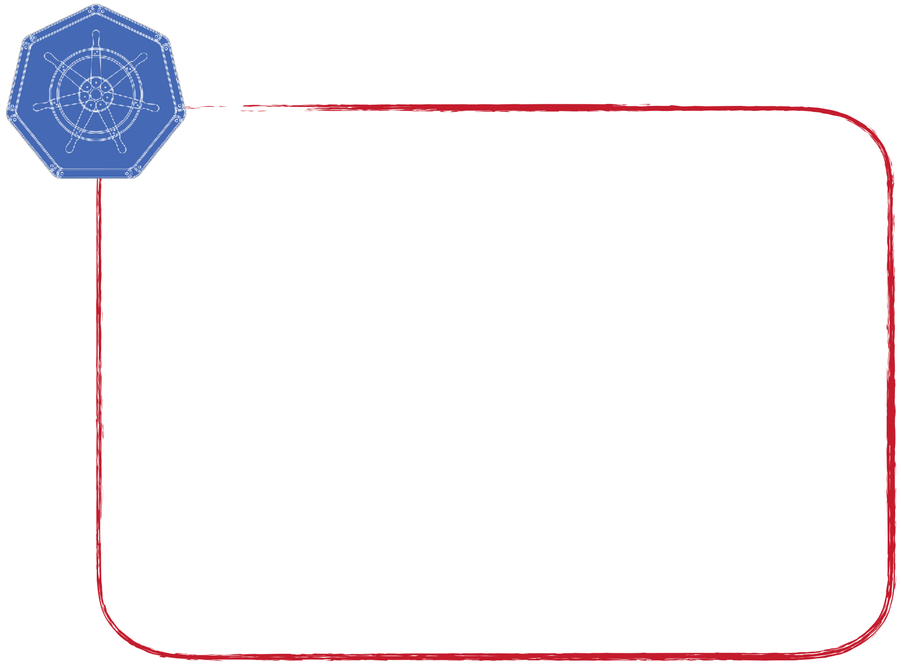
- Use ***values***.yaml

```yaml
image:
  repository: shelleg/msa-api
  tag: config
  pullPolicy: IfNotPresent

service:
  type: ClusterIP
  port: 8080
…
```

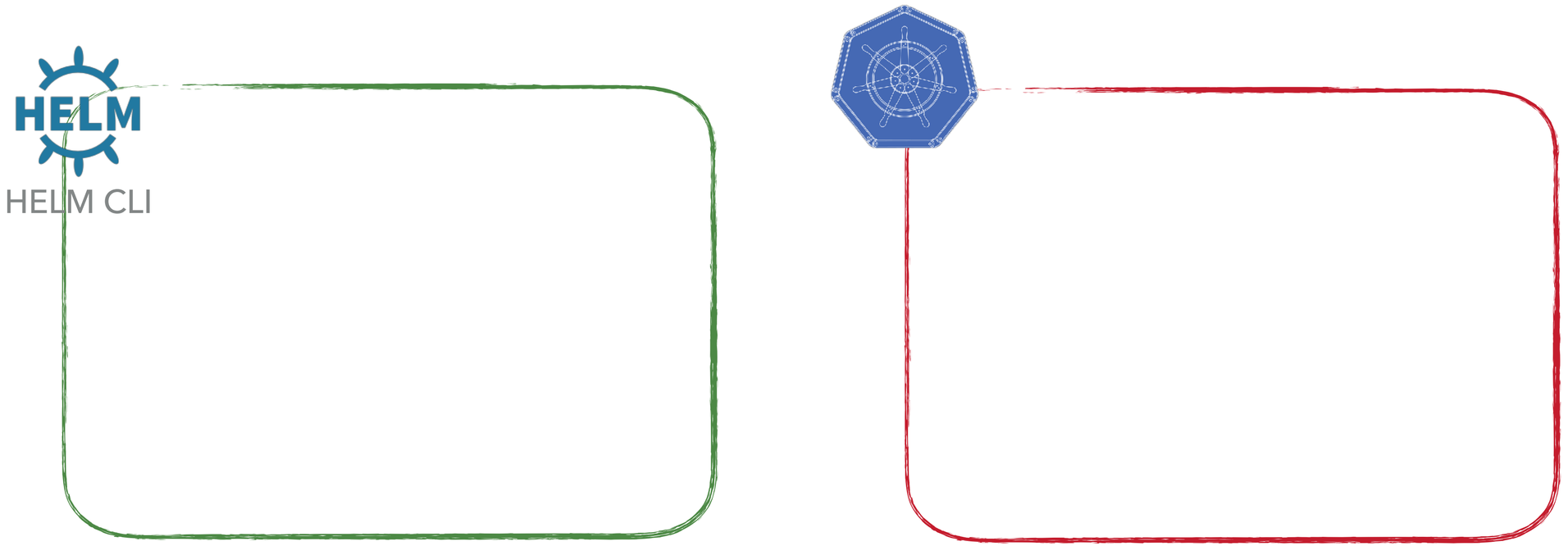
- A rich templating system for making charts generic yet highly customisable


HOW DOES HELM WORK ?

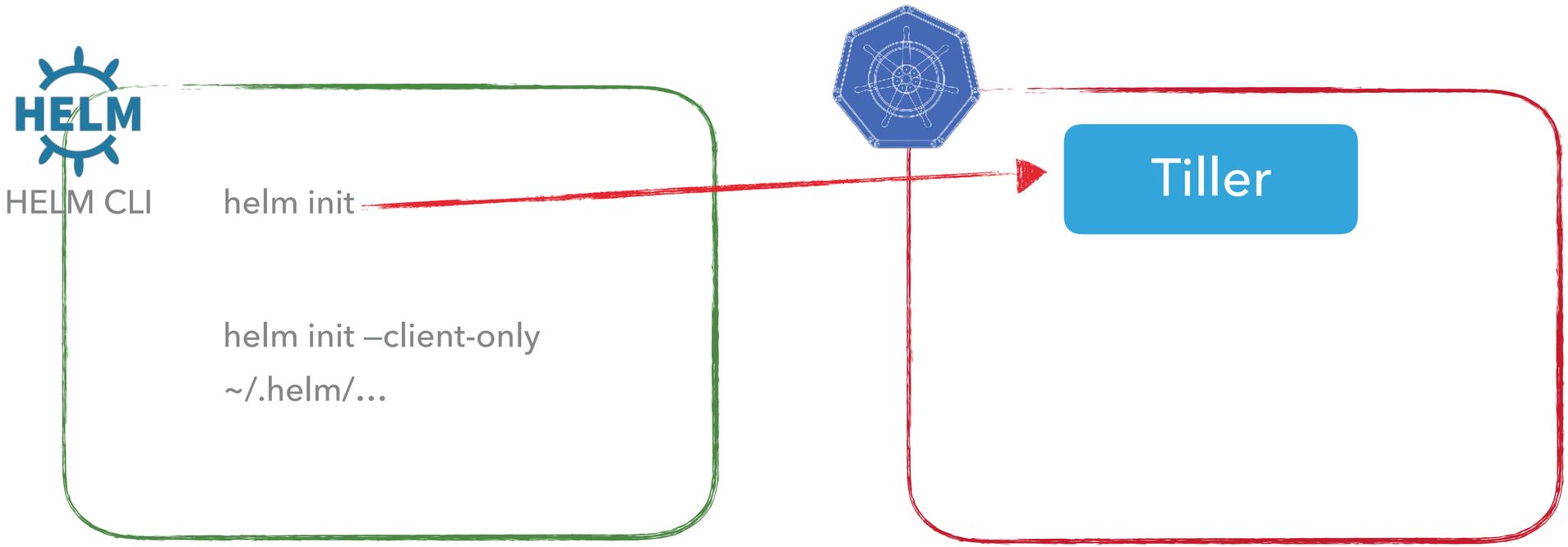
HELM ARCHITECTURE – A KUBERNETES CLUSTER



HELM ARCHITECTURE – A HELMSMAN WORKSTATION



HELM ARCHITECTURE – HELM INIT



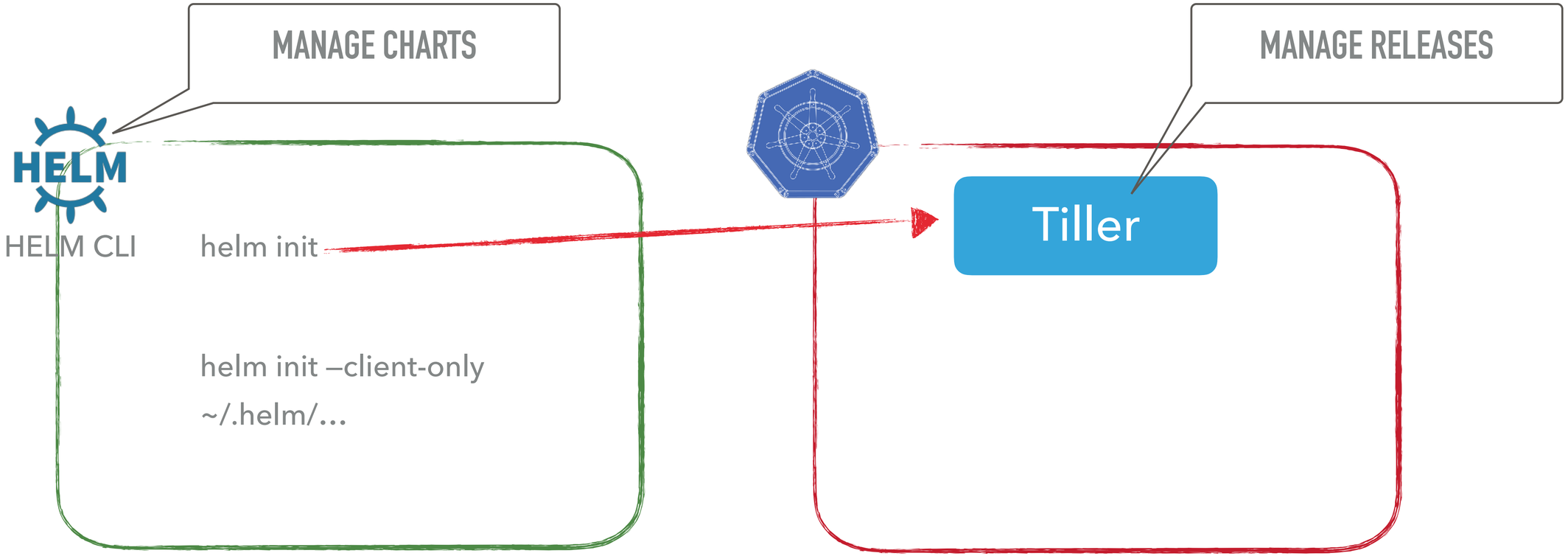
HELM CLIENT

- Local chart development
- Managing repositories
- Interacting with the Tiller server
  - Sending charts to be installed
  - Asking for information about releases
  - Requesting upgrading or uninstalling of existing releases
- It can be used to serve your local charts via http ...
-

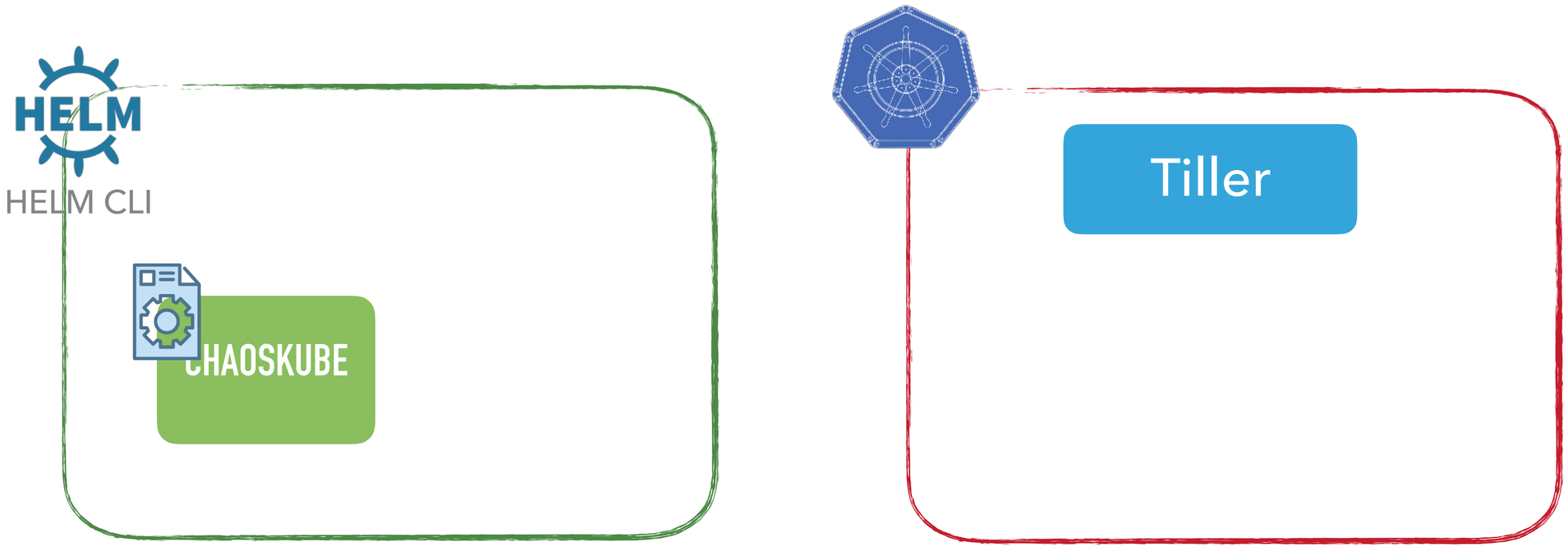
TILLER – "HELM SERVER"

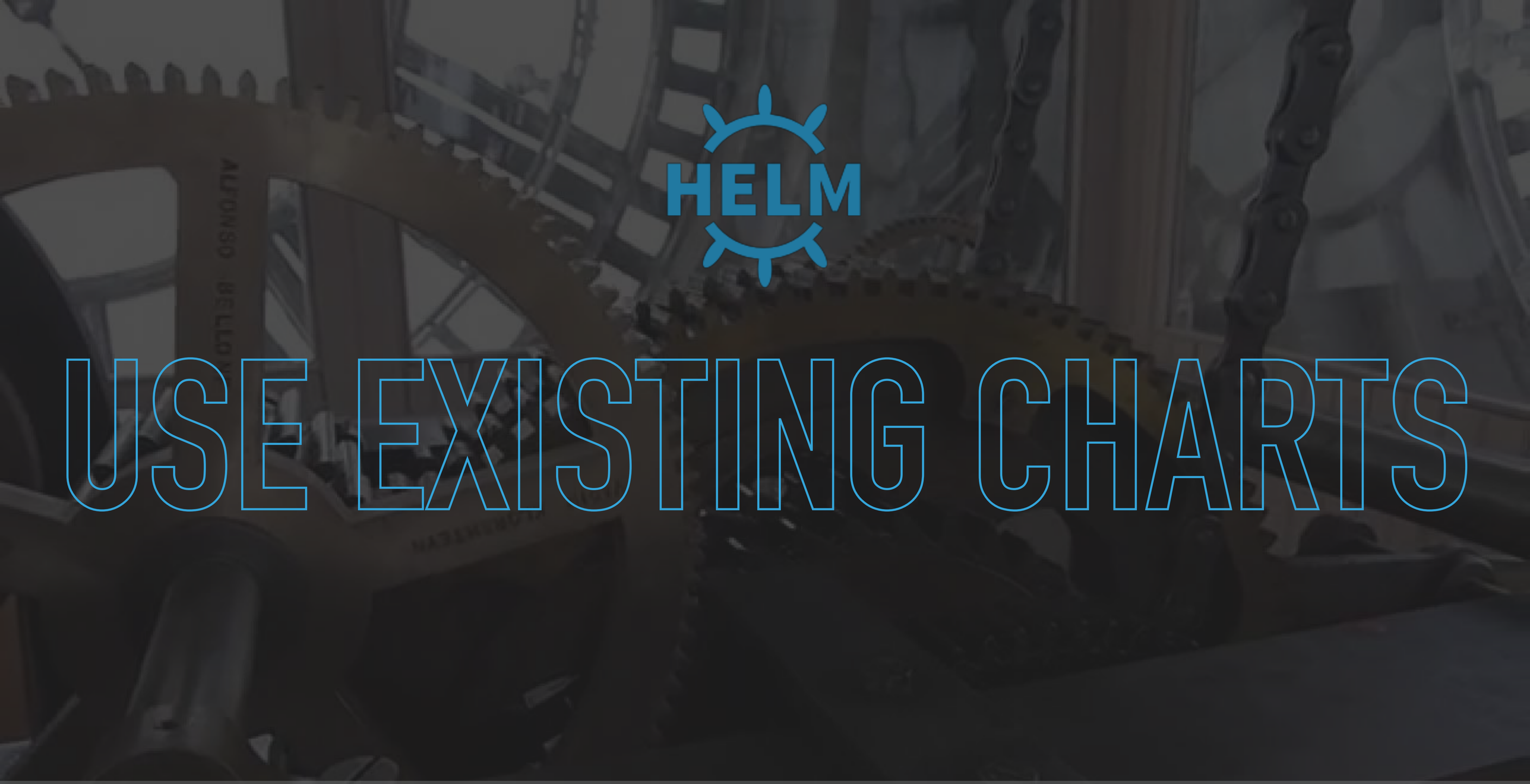
- Listening for incoming requests from the Helm client
- Combining a chart and configuration to build a release
- Installing charts into Kubernetes, and then tracking the subsequent release
- Upgrading and uninstalling charts by interacting with Kubernetes

HELM ARCHITECTURE – HELM INIT



HELM ARCHITECTURE – INSTALL AN EXISTING CHART



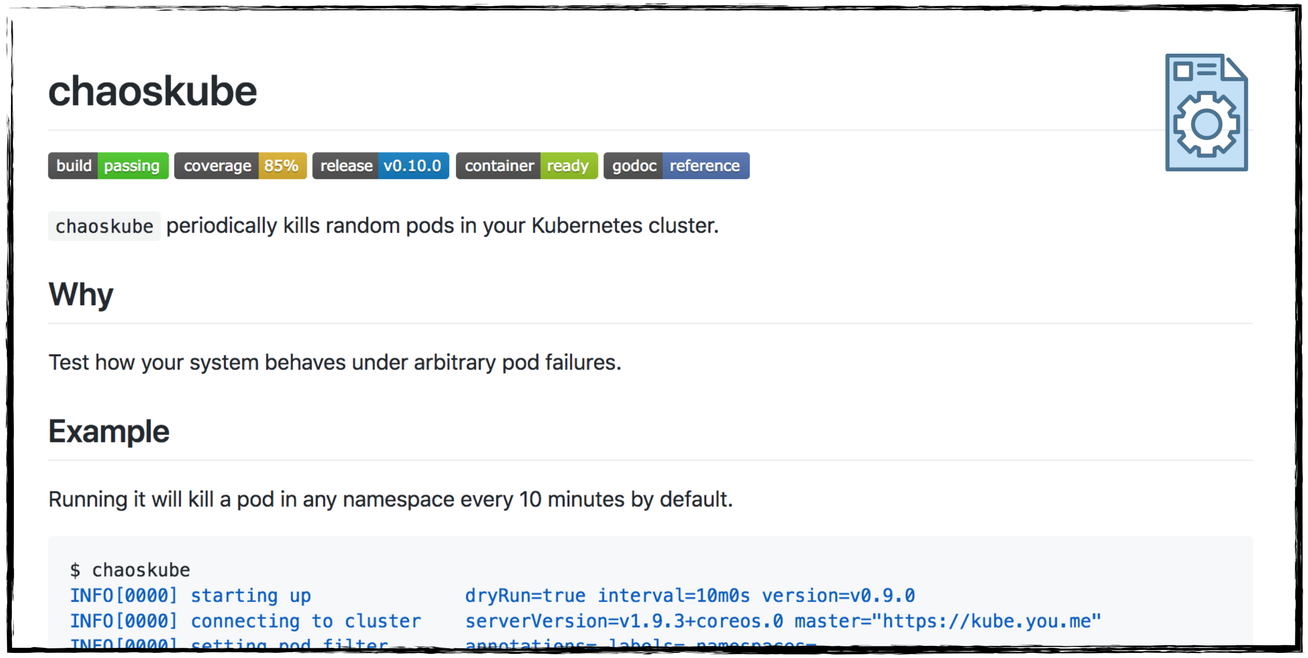


USE EXISTING CHARTS

CHAOSKUBE

- chaoskube is a "***chaos-monkey lite***" it basically takes down pod based on a schedule to test you resilience (and there are some tweaks via configuration)

https://github.com/linki/chaoskube



chaoskube

build passing | coverage 85% | release v0.10.0 | container ready | godoc reference

`chaoskube` periodically kills random pods in your Kubernetes cluster.

Why

Test how your system behaves under arbitrary pod failures.

Example

Running it will kill a pod in any namespace every 10 minutes by default.

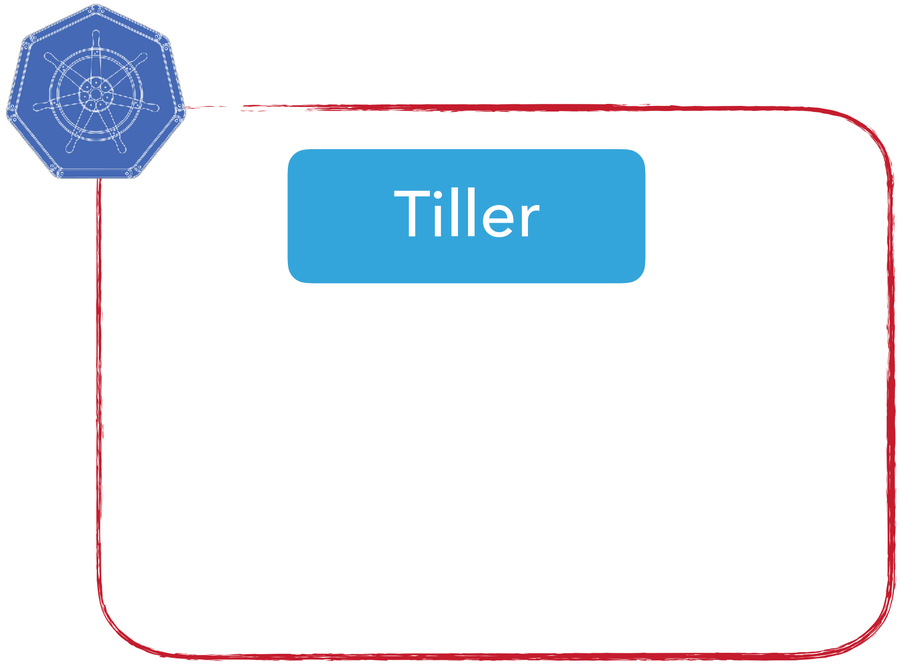
```
$ chaoskube
INFO[0000] starting up              dryRun=true interval=10m0s version=v0.9.0
INFO[0000] connecting to cluster    serverVersion=v1.9.3+coreos.0 master="https://kube.you.me"
INFO[0000] setting pod filter       annotations= labels= namespaces=
```

HELM ARCHITECTURE – WORK WITH REPOS

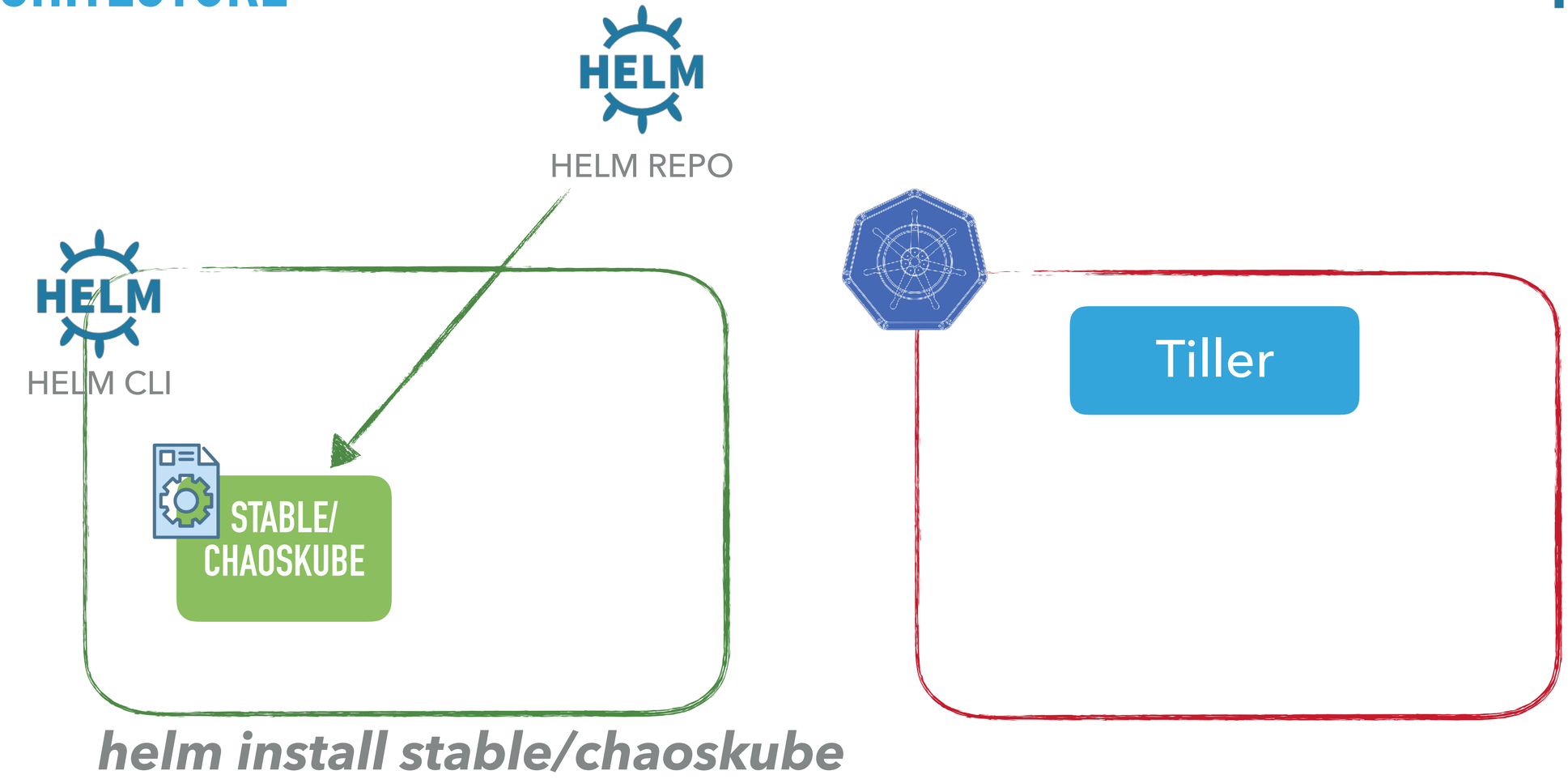
HELM REPO

helm repo list

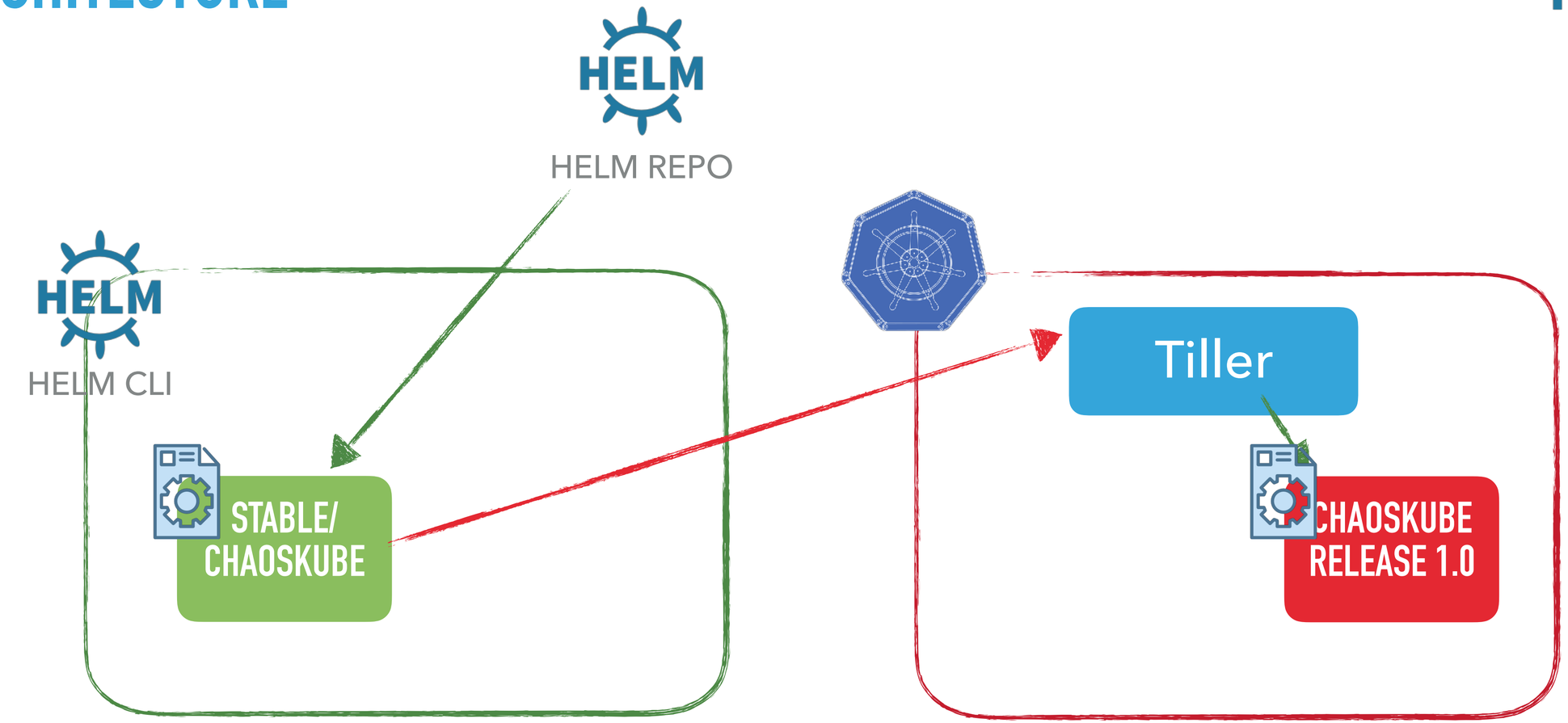
NAME	URL
stable	https://kubernetes-charts.storage.googleapis.com
local	http://127.0.0.1:8879/charts



HELM ARCHITECTURE



HELM ARCHITECTURE



CAHRTS STRUCTURE

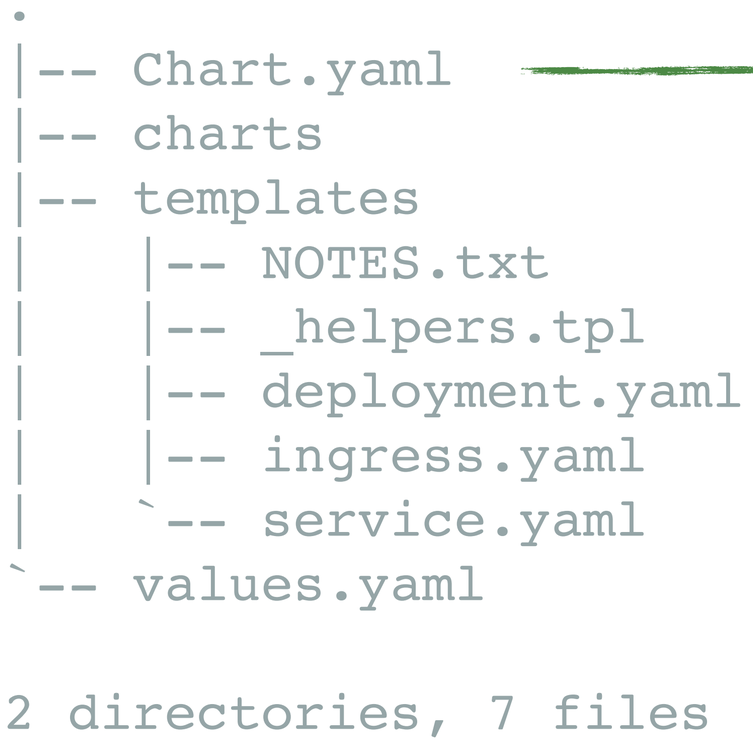
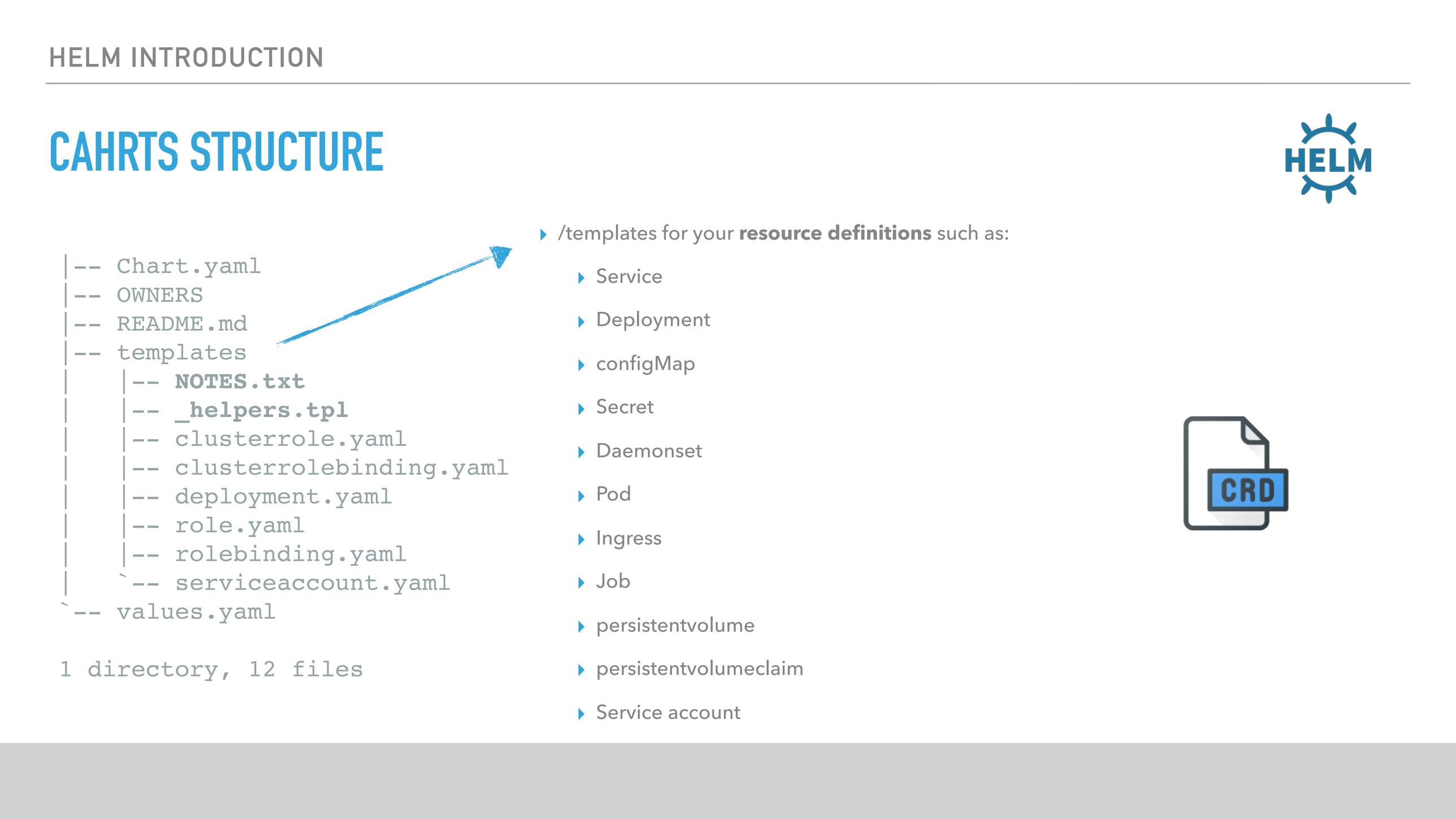


Chart.yaml

```yaml
apiVersion: v1
appVersion: 0.10.0
description: Chaoskube periodically kills random pods in your cluster.
engine: gotpl
home: https://github.com/linki/chaoskube
maintainers:
- email: linki+kubernetes.io@posteo.de
  name: linki
name: chaoskube
sources:
- https://github.com/linki/chaoskube
version: 0.10.0
```

Chart and image version
{not always identical ...}

CAHRTS STRUCTURE

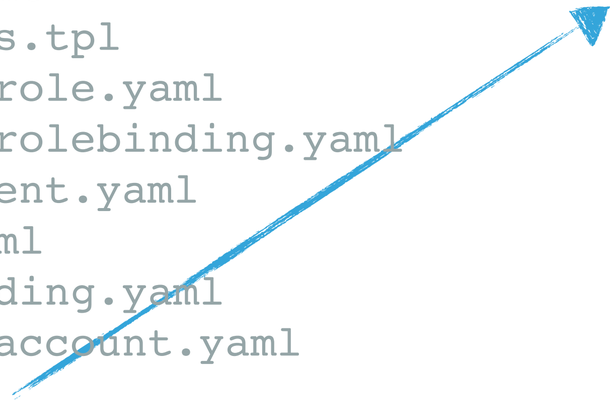



```
|-- Chart.yaml
|-- OWNERS
|-- README.md
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- clusterrole.yaml
|   |-- clusterrolebinding.yaml
|   |-- deployment.yaml
|   |-- role.yaml
|   |-- rolebinding.yaml
|   `-- serviceaccount.yaml
`-- values.yaml

1 directory, 12 files
```

- /templates for your **resource definitions** such as:
  - Service
  - Deployment
  - configMap
  - Secret
  - Daemonset
  - Pod
  - Ingress
  - Job
  - persistentvolume
  - persistentvolumeclaim
  - Service account

CAHRTS STRUCTURE

```
|-- Chart.yaml
|-- OWNERS
|-- README.md
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- clusterrole.yaml
|   |-- clusterrolebinding.yaml
|   |-- deployment.yaml
|   |-- role.yaml
|   |-- rolebinding.yaml
|   `-- serviceaccount.yaml
`-- values.yaml

1 directory, 12 files
```



values.yaml

```yaml
# container name
name: chaoskube

# docker image
image: quay.io/linki/chaoskube

# docker image tag
imageTag: v0.10.0

# number of replicas to run
replicas: 1

# interval between pod terminations
interval: 10m

# label selector to filter pods by, e.g. app=foo,stage!=prod
labels:

# annotation selector to filter pods by, e.g.
chaos.alpha.kubernetes.io/enabled=true
annotations:
…
```

CAHRTS STRUCTURE


```
|-- Chart.yaml
|-- OWNERS
|-- README.md
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- clusterrole.yaml
|   |-- clusterrolebinding.yaml
|   |-- deployment.yaml
|   |-- role.yaml
|   |-- rolebinding.yaml
|   `-- serviceaccount.yaml
`-- values.yaml

1 directory, 12 files
```

Control Structures:

{ render based on boolean }

serviceaccount.yaml

```
{{- if .Values.rbac.create -}}
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
{{ include "labels.standard" . | indent 4 }}
  name: {{ printf "%s-%s" .Release.Name .Values.name }}
{{- end -}}
```

values.yaml

```
# container name
name: chaoskube
```

CAHRTS STRUCTURE

```
.
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   |-- ingress.yaml
|   `-- service.yaml
`-- values.yaml

2 directories, 7 files
```

_helpers.tpl

```
{{/*
Expand the name of the chart.
*/}}
{{- define "chaoskube.name" -}}
{{- default .Chart.Name .Values.nameOverride |
trunc 63 | trimSuffix "-" -}}
{{- 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).
If release name contains chart name it will be
used as a full name.
*/}}
{{- define "chaoskube.fullname" -}}
{{- if .Values.fullnameOverride -}}
{{- .Values.fullnameOverride | trunc 63 |
trimSuffix "-" -}}
{{- else -}}
…
```

Custom functions + set default based on Environment

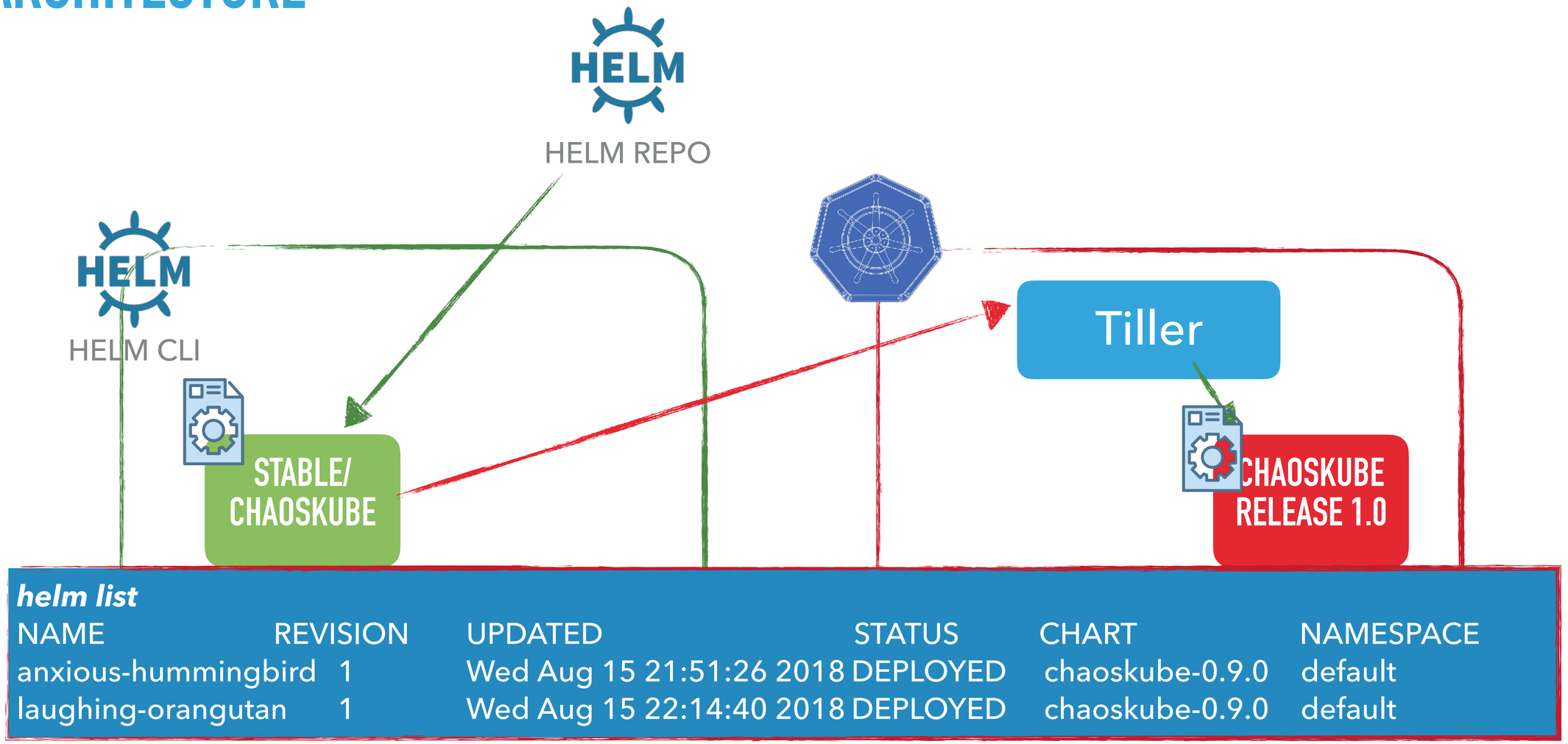
CAHRTS STRUCTURE

- A ***HELM standard way*** of showing information the end user can use in order to access the application.

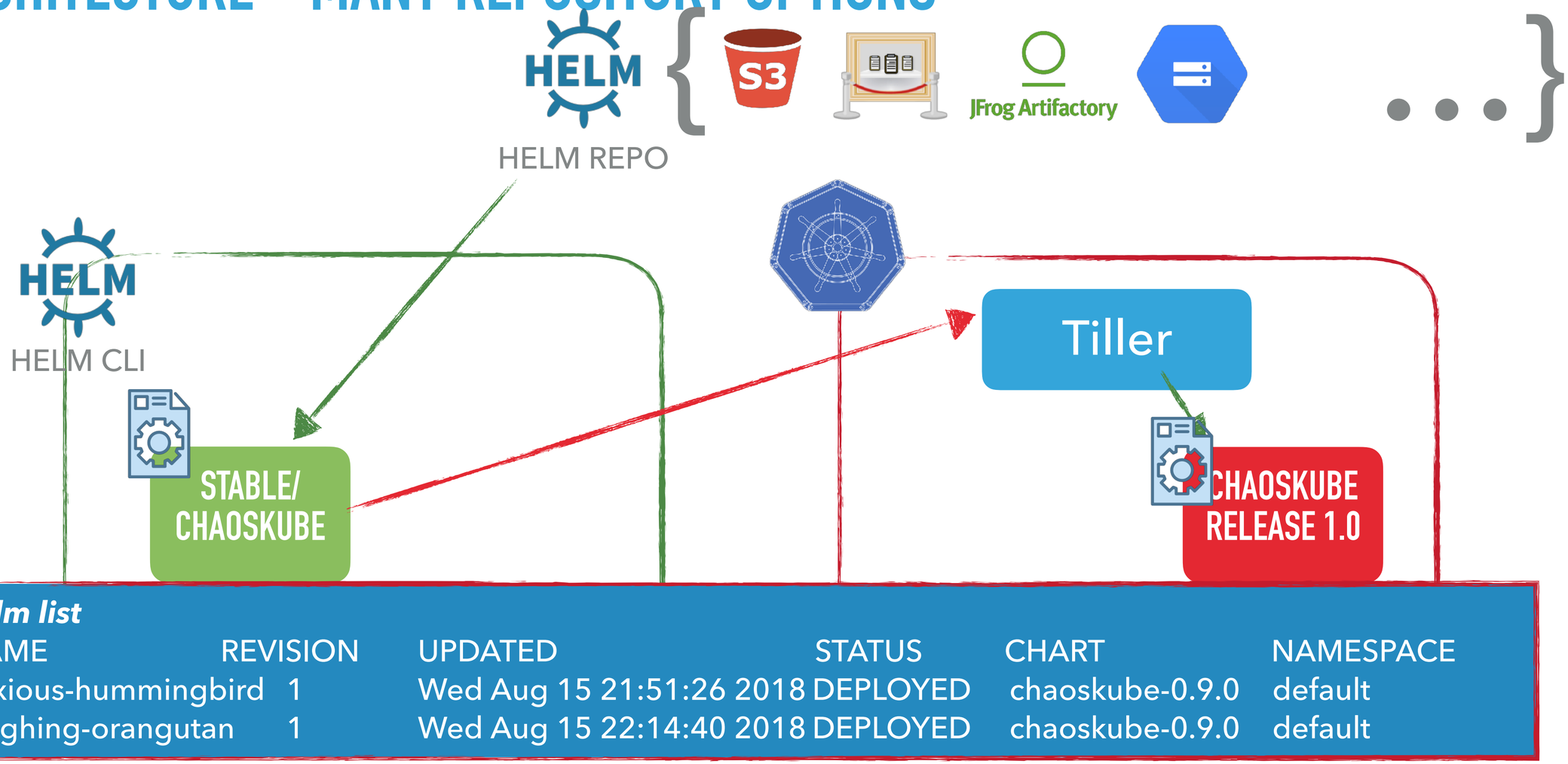
NOTES.txt

```
chaoskube is running and will kill arbitrary pods every
{{ .Values.interval }}.

You can follow the logs to see what chaoskube does:

    POD=$(kubectl -n {{ .Release.Namespace }} get pods -
l='release={{ template "chaoskube.fullname" . }}' --
output=jsonpath='{.items[0].metadata.name}
        ')
    kubectl -n {{ .Release.Namespace }} logs -f $POD
{{ if .Values.dryRun }}
You are running in dry-run mode. No pod is actually terminated.
{{ end -}}
```

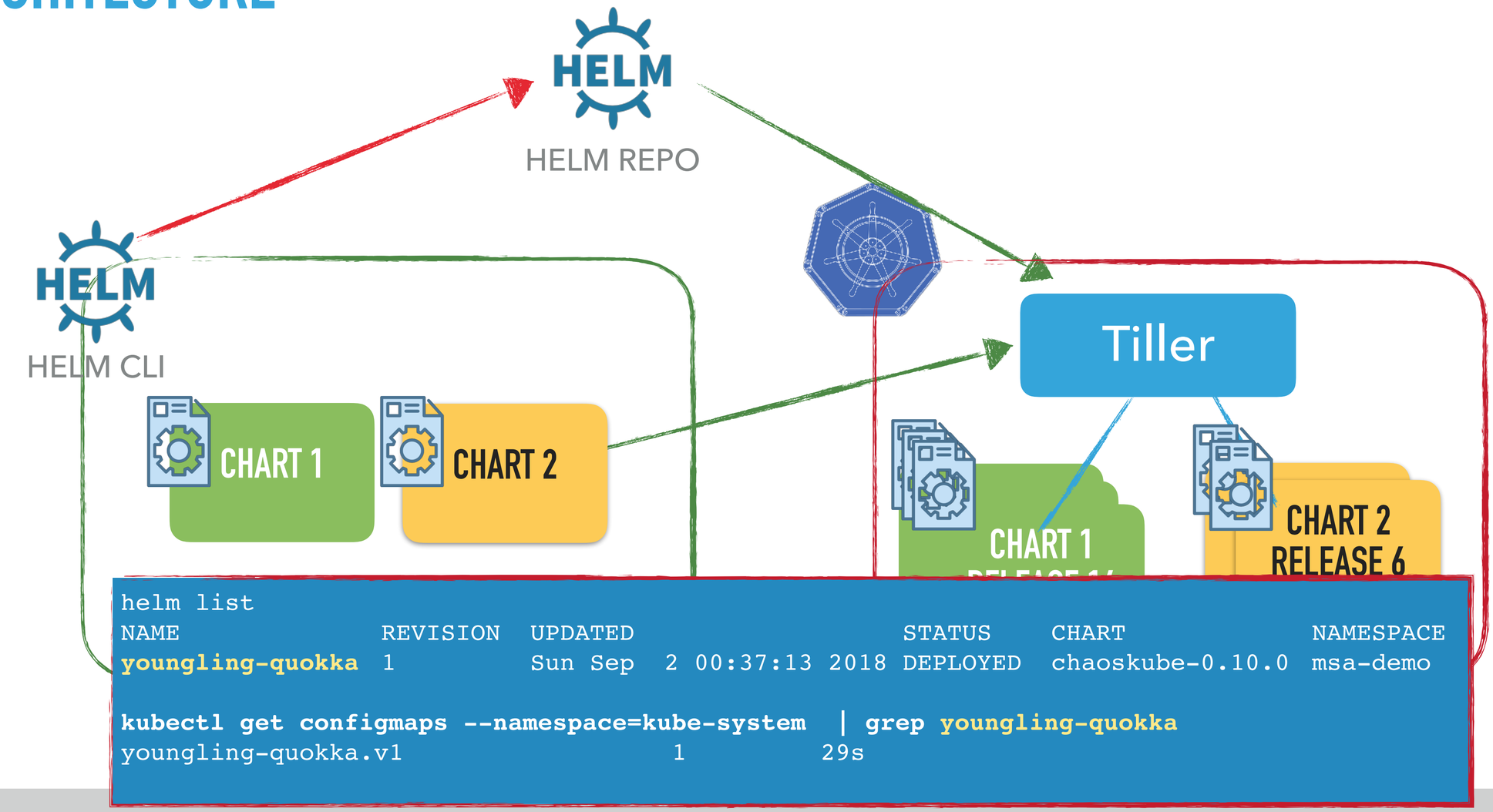
HELM ARCHITECTURE



HELM ARCHITECTURE – MANY REPOSITORY OPTIONS



HELM ARCHITECTURE



"CHARTIFY" (& REUSE) . . .

OUR “COMPLICATED” MICRO SERVICE APPLICATION – DOCKERIZED

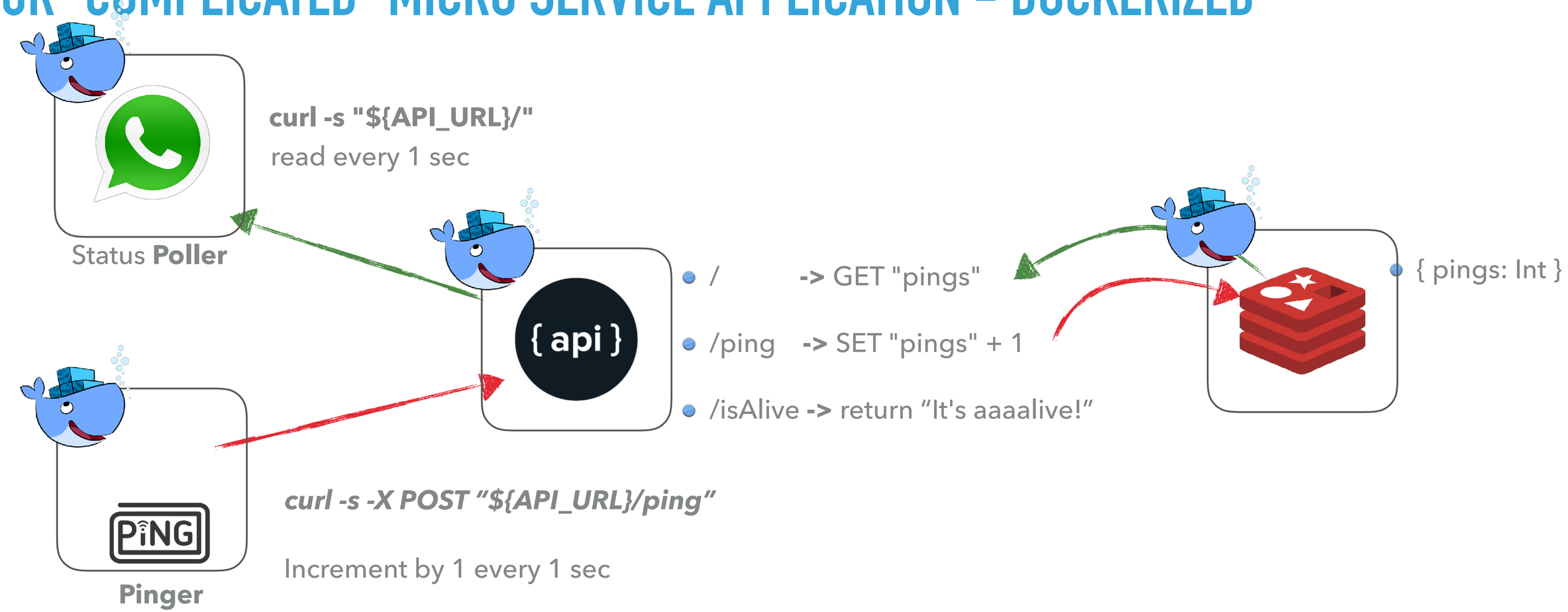
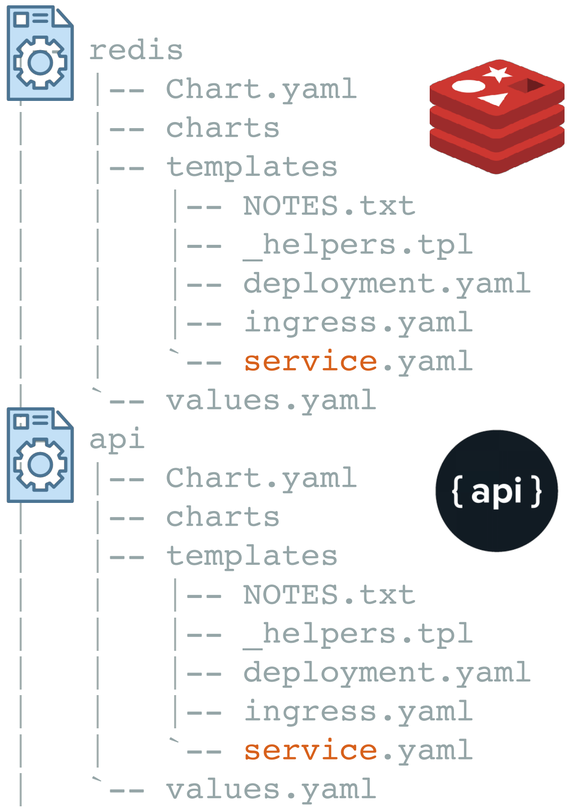


CHART PER COMPONENT



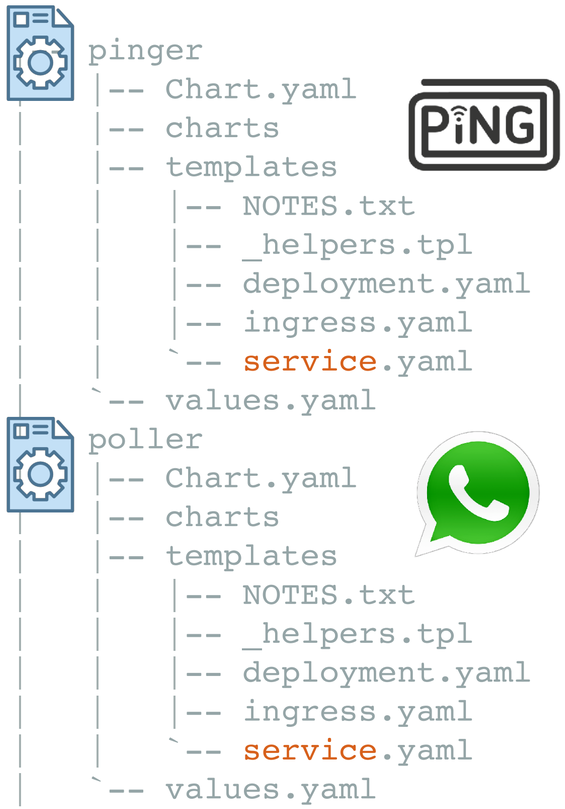
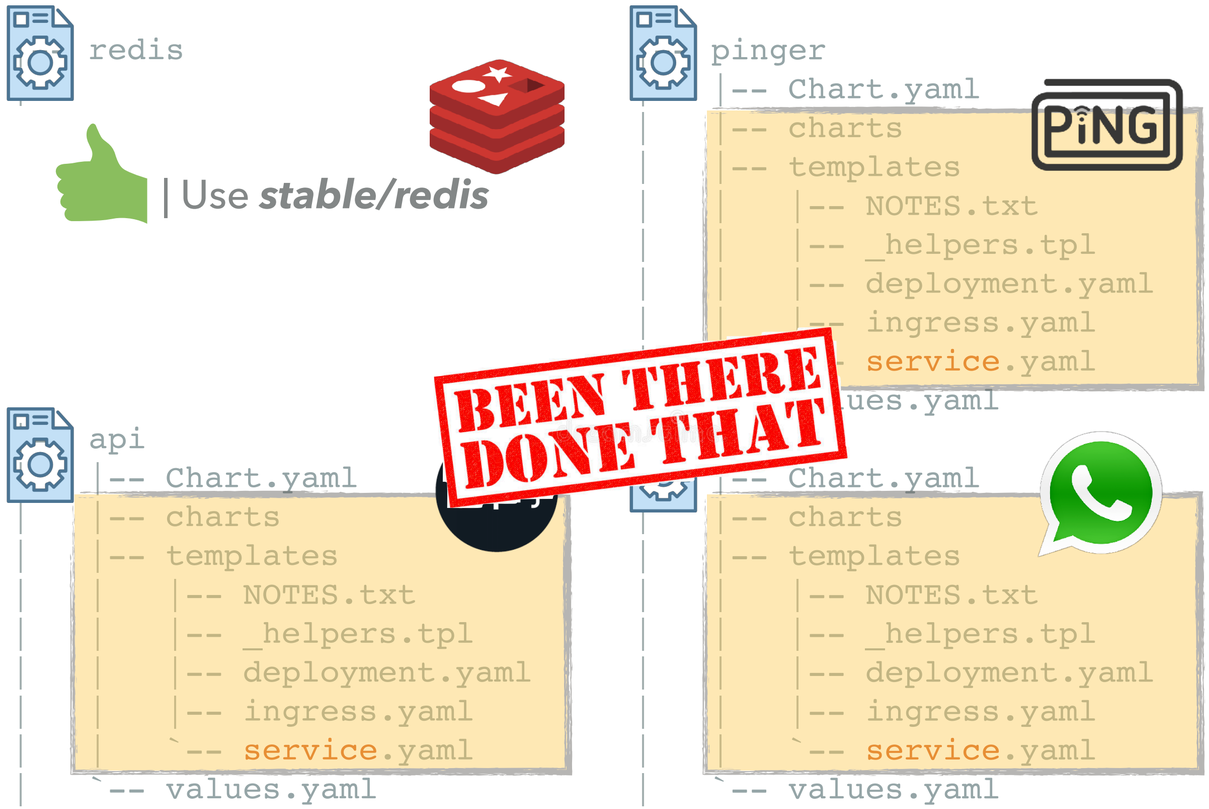


CHART PER COMPONENT



“MSA” CHART

```
.
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- api-deployment.yaml
|   |-- api-service.yaml
|   |-- deployment.yaml
|   |-- pinger-deployment.yaml
|   |-- poller-deployment.yaml
|   `-- service.yaml
`-- values.yaml
```

VALUES

| This works ;) but ...

values.yaml

```yaml
api:
  name: api
  replicaCount: 1
  image:
    repository: shelleg/msa-api
    tag: config
    pullPolicy: Always
    containerPort: 8080
  service:
    enabled: true
    type: ClusterIP
    port: 8080
  resources: {}

  nodeSelector: {}

  tolerations: []

  affinity: {}
```

```yaml
pinger:
  name: pinger
  replicaCount: 1
  image:
    repository: shelleg/msa-pinger
    tag: latest
    pullPolicy: Always

 service:
    enabled: false


 resources: {}

  nodeSelector: {}

  tolerations: []

  affinity: {}
```

```yaml
poller:
  name: poller
  replicaCount: 1
  image:
    repository: shelleg/msa-poller
    tag: latest
    pullPolicy: Always

 service:
    enabled: false


 resources: {}

  nodeSelector: {}

  tolerations: []

  affinity: {}
```

TEMPLATES

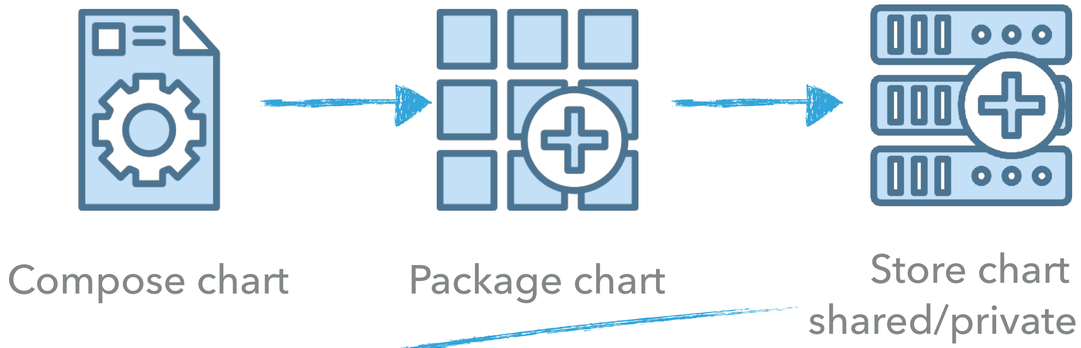
api-deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: {{ .Values.api.name }}
  labels:
    app: {{ .Values.api.name }}
    chart: {{ template "msa.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  replicas: {{ .Values.api.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Values.api.name }}
      release: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ .Values.api.name }}
        release: {{ .Release.Name }}
```

api-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.api.name }}
  labels:
    app: {{ .Values.api.name }}
    chart: {{ template "msa.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  type: {{ .Values.api.service.type }}
  ports:
    - port: {{ .Values.api.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: {{ .Values.api.name }}
    release: {{ .Release.Name }}
```

the better way



| helm install myrepo/msa –set nameOverride: api –set service.enabled=true –name api

| helm install myrepo/msa –set nameOverride: pinger –name pinger

| helm install myrepo/msa –set nameOverride: poller –name poller

HELPERS & VALUES

- nameOverride: [e.g. api | pinger | poller]

HELPERS & VALUES

- nameOverride: [api | pinger | poller]

_helpers.yaml

```
{{- define “chart.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

HELPERS & VALUES

- nameOverride: [api | pinger | poller]

_helpers.yaml

```
{{- define “chart.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

| Make the chart reusable

HELPERS & VALUES

- nameOverride: [api | pinger | poller]
- service.enabled: true

_helpers.yaml

```
{{- define “chart.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

| Make the chart reusable

service.yaml

```
{{- if .Values.service.enabled }}
{{- end }}
```

HELPERS & VALUES

- nameOverride: [api | pinger | poller]
- service.enabled: true

_helpers.yaml

```
{{- define “chart.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

| Make the chart reusable

service.yaml

```
{{- if .Values.service.enabled -}}
{{- end -}}
```

| Make entire RD optional

TEMPLATES

This works ;)
but ...

api-deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: {{ .Values.api.name }}
  labels:
    app: {{ .Values.api.name }}
    chart: {{ template "msa.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  replicas: {{ .Values.api.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Values.api.name }}
      release: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ .Values.api.name }}
        release: {{ .Release.Name }}
```

the better way

This is D.R.Y !

deployment.yaml

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: {{ .Values.name }}
  labels:
    app: {{ .Values.name }}
    chart: {{ template "msa.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Values.name }}
      release: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ .Values.name }}
        release: {{ .Release.Name }}
```

TEMPLATES

This works ;)
but ...

api-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.api.name }}
  labels:
    app: {{ .Values.api.name }}
    chart: {{ template "msa.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  type: {{ .Values.api.service.type }}
  ports:
    - port: {{ .Values.api.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: {{ .Values.api.name }}
    release: {{ .Release.Name }}
```

the better way

This is D.R.Y !

service.yaml

```
{{- if .Values.service.enabled -}}
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.api.name }}
  labels:
    app: {{ .Values.api.name }}
    chart: {{ template "msa.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  type: {{ .Values.api.service.type }}
  ports:
    - port:
{{ .Values.api.service.port }}
      targetPort: http
      protocol: TCP
      name: http
...
{{- end -}}
```

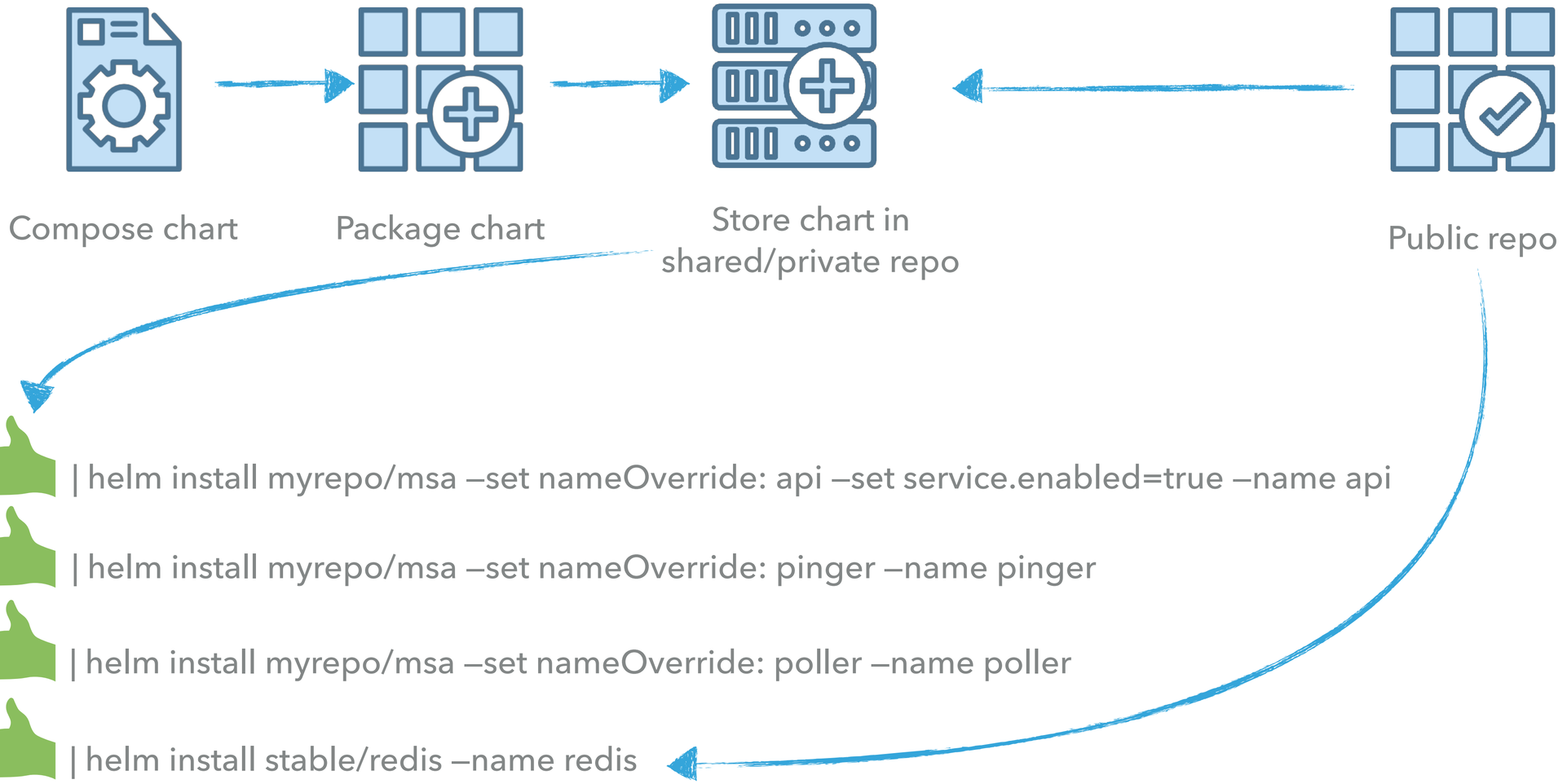
"MSA" CHART

```
.
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   |-- service.yaml
`-- values.yaml
```

my-chart-repo/msa-0.1.0.gz

```yaml
apiVersion: v1
appVersion: "1.0"
description: msg chart for Kubernetes
name: api
version: 0.1.0
```

the better way



MANAGING DEPENDENCIES - A.K.A UMBRELLA CHAT

```
.
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- dependencies.yaml
`-- values.yaml
```

```yaml
dependencies:
  - name: msa
    repository: file://../msa
    version: 0.1.0
  - name: redis
    repository: stable/redis
    version: x.x.x
```


HANDS ON

HOW TO USE THIS LAB ?!

Informational -> you don’t have / need to do it ;)

Get your hands dirty -> follow the instructions

MINIKUBE

MINIKUBE

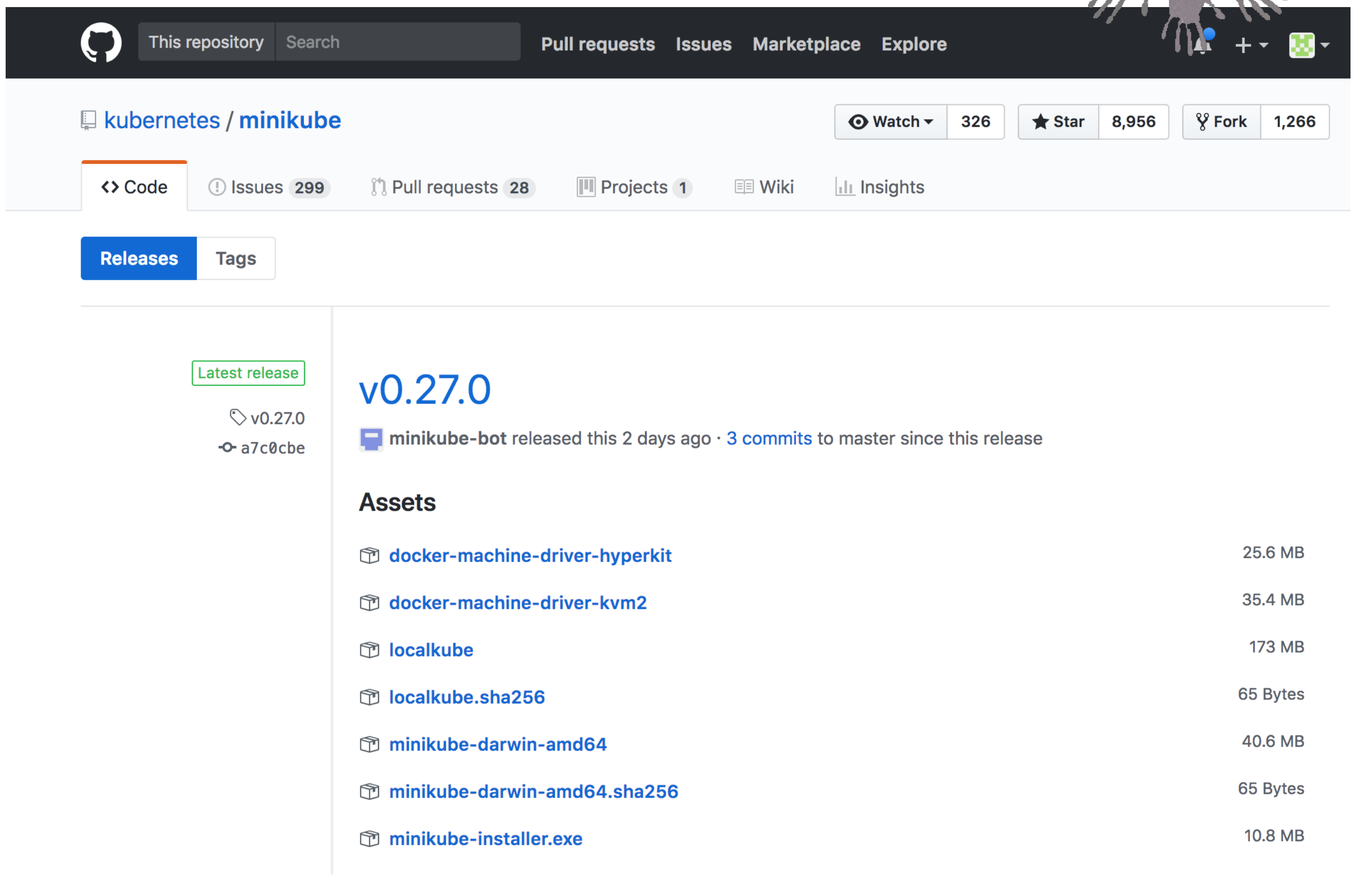
- Minikube supports Kubernetes features such as:
  - DNS
  - NodePorts
  - ConfigMaps and Secrets
  - Dashboards
  - Container Runtime: Docker, rkt and CRI-O
  - Enabling CNI (Container Network Interface)
  - Ingress

WHY

- Good for POC, QuickStarts and Demo’s
- Gives a taste of Kubernetes and how to work with it, without going through a complex setup

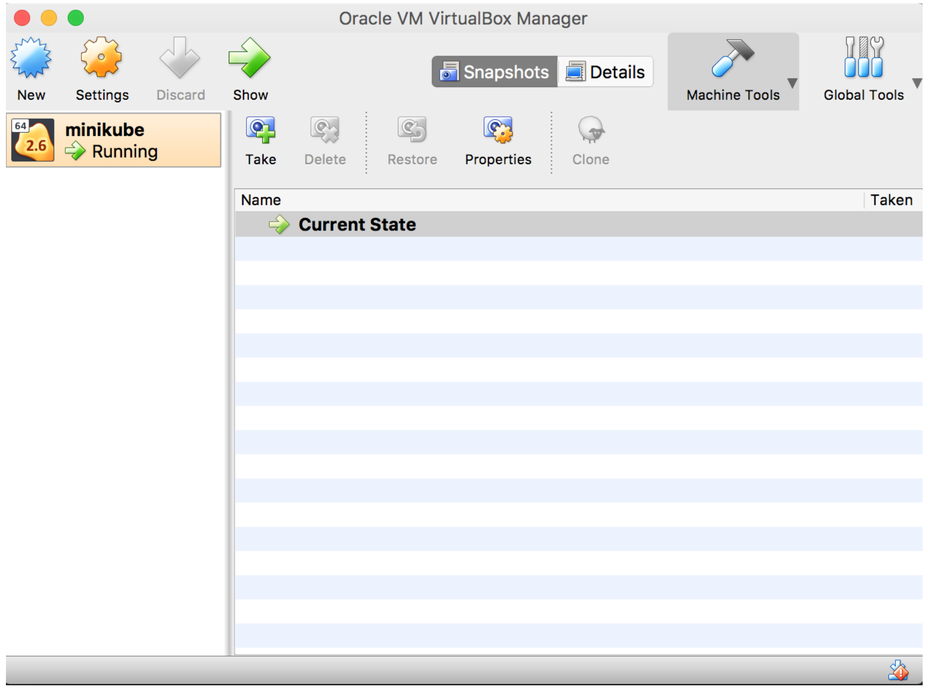
INSTALLING MINIKUBE

- Install ***kubectl*** link
- Install ***minikube*** link



MINIKUBE START

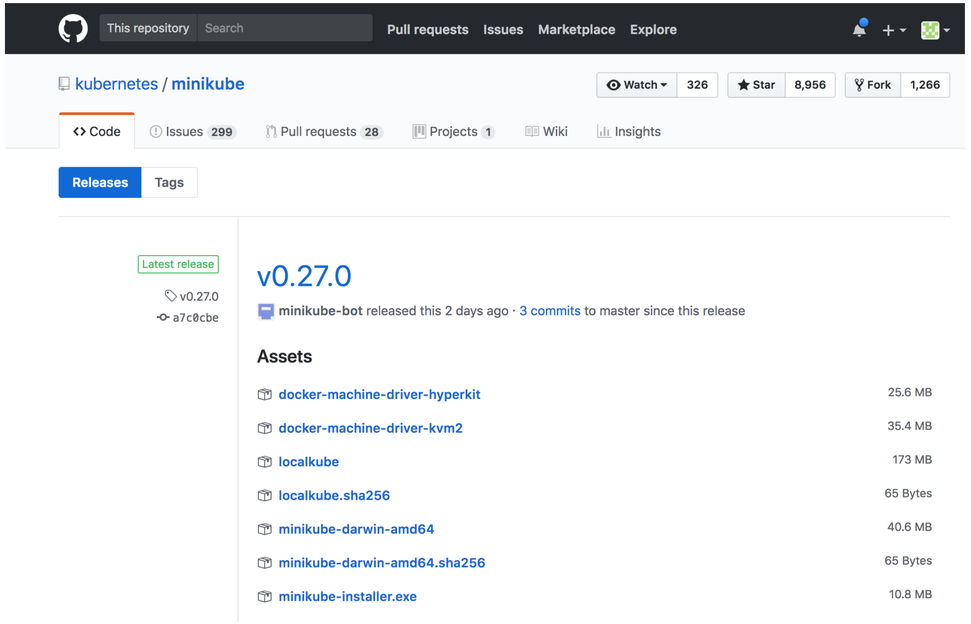
- ***minikube start***



20:31 $ minikube start
Starting local Kubernetes v1.9.0 cluster...
Starting VM...
Getting VM IP address...
Moving files into cluster...
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
Loading cached images from config file.

MACOSX VIA HOMEBREW

```
brew update
brew install kubernetes-cli kubernetes-helm
brew cask install minikube
```



UBUNTU/DEBIAN

```
sudo apt-get update && sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/
sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl

curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.30.0/minikube-
linux-amd64 && chmod +x minikube && sudo cp minikube /usr/local/bin/ && rm minikube
```

CENTOS/RHEL/FEDORA

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpghttps://
packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF

yum install -y kubectl

curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v0.30.0/minikube-linux-
amd64 && chmod +x minikube && sudo cp minikube /usr/local/bin/ && rm minikube
```

WINDOWS

Use the Chocolaty package manager:

```
choco install kubernetes-cli kubernetes-helm
```

Download Minikube from: https://github.com/kubernetes/minikube/releases
Download the minikube-windows-amd64.exe file,
rename it to minikube.exe and add it to your path.


HELM


ON
OFF

INSTALL HELM ON KUBERNETES

helm init --service-account helm --dry-run --debug > helm-init.yml
kubectl create -f helm-rbac-cluster-wide.yml
kubectl create -f helm-init.yml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: helm
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: helm
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: helm
    namespace: kube-system
```

CONFIGURE HELM REPOSITORIES

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com
```

helm / charts

Watch 243 | Star 5,186 | Fork 4,607

Code | Issues 233 | Pull requests 344 | Insights

Branch: master ▾ | charts / stable / | Create new file | Upload files | Find file | History

okgolove and k8s-ci-robot [stable/drone], changed repository name in NOTES (#8628) ··· Latest commit fc4a83c 2 hours ago

..		
acs-engine-autoscaler	Enrich deploy. template for acs-engine-autoscaler (#5662)	5 months ago
aerospike	[stable/aerospike] Add cmd and args options to Aerospike config (#3856)	8 months ago
anchore-engine	add brady todhunter as review/maintainer to anchore-engine (#8514)	5 days ago
apm-server	[stable/apm-server] Elastic APM Server (#6058)	4 months ago
ark	[stable/ark] Quote configuration parameter backupStorageProvider.conf...	8 days ago
artifactory-ha	Deprecate JFrog charts (moved to https://github.com/jfrog/charts) (#7627	a month ago
artifactory	Deprecate JFrog charts (moved to https://github.com/jfrog/charts) (#7627	a month ago
auditbeat	upgrade auditbeat (#8277)	12 days ago
aws-cluster-autoscaler	Typo fix in aws-cluster-autoscaler/README.md (#4297)	7 months ago

INSTALL HELM-S3 PLUGIN

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

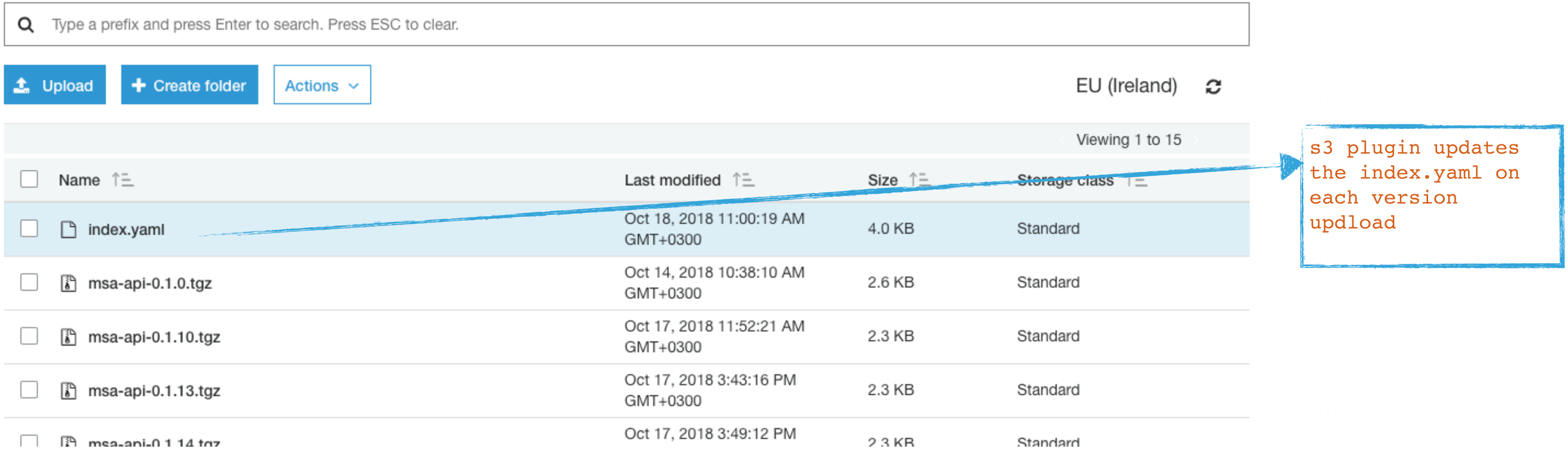
-> If you are planning on pushing to your s3 based helm repo

-> get your own repo and setup the required permissions.

-> I our case you will be using it for read / download purposes which is already public / open source.

ADD THE MSA-CHARTS S3 HELM REPOSITORY

helm repo add msa-charts s3://msa-charts/



DEPLOY YOUR UMBRELLA . . .

```
kubectl create namespace msa-umbrella
helm dep build ./helm/msa-umbrella
helm install --name v1 --namespace msa-umbrella ./helm/msa-umbrella
```

DEPLOY YOUR UMBRELLA . . .

```
kubectl create namespace msa-umbrella
helm dep build ./helm/msa-umbrella
helm install --name v1 --namespace msa-umbrella ./helm/msa-umbrella
```

CHECK YOUR UMBRELLA . . .

```
helm list                                          # view helm releases which are installed

kubectl -n msa-umbrella get po,svc,deploy  # view msa-umbrella pods, services & deployments
```

RECAP

- Helm is awesome -> (completes Kubernetes)
  - Application centric solution
  - Flexible templating language
  - Well integrated into workflows such as GitKube, Scafold and others

USE [DON'T OVER USE] YOUR IMAGINATION

- Custom Resource Definitions { e.g istio }
- Common monitoring RD's { e.g prometheus }
- Common logging RD's { e.g fluentd }
- Ingresses (yes / no)
- Services (type: NodePort, ClusterIp, LoadBalancer)
- Provider specific configs { on-prem vs. cloud }
-

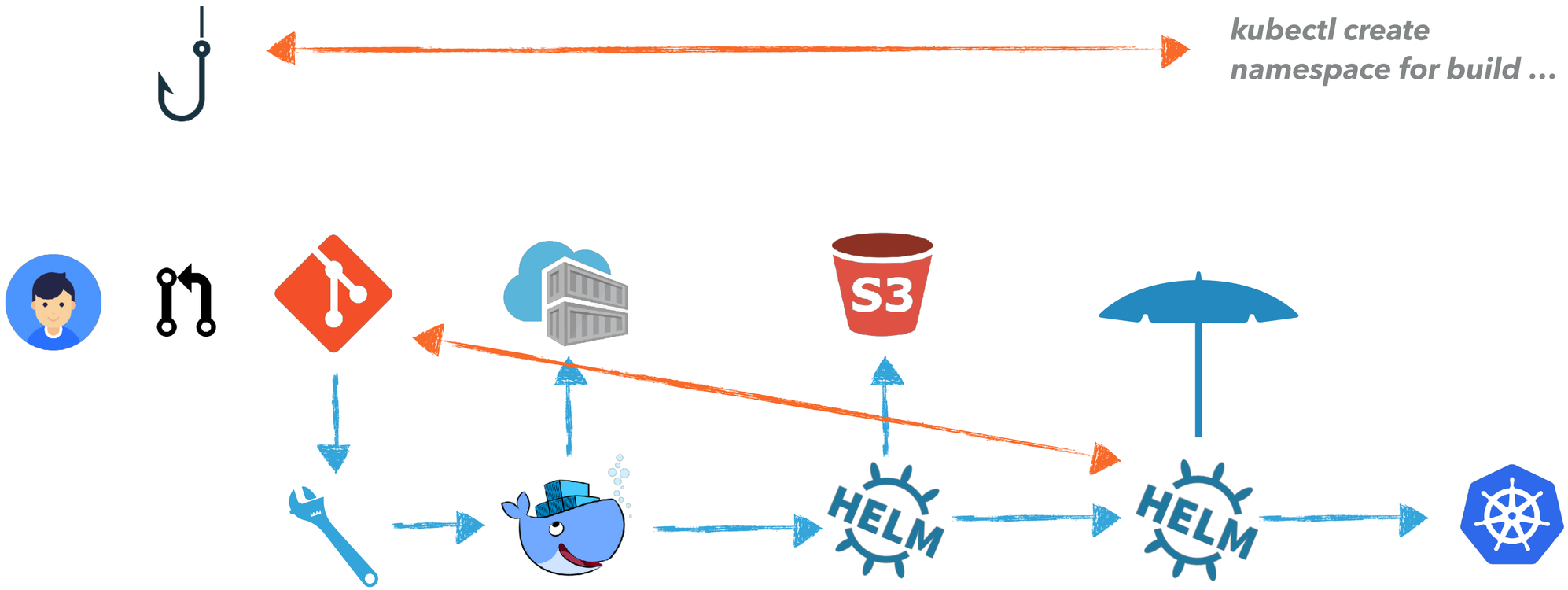
 | {{ if .Values.monitoring.enabled }}

 | {{ if .Values.logging.enabled }}

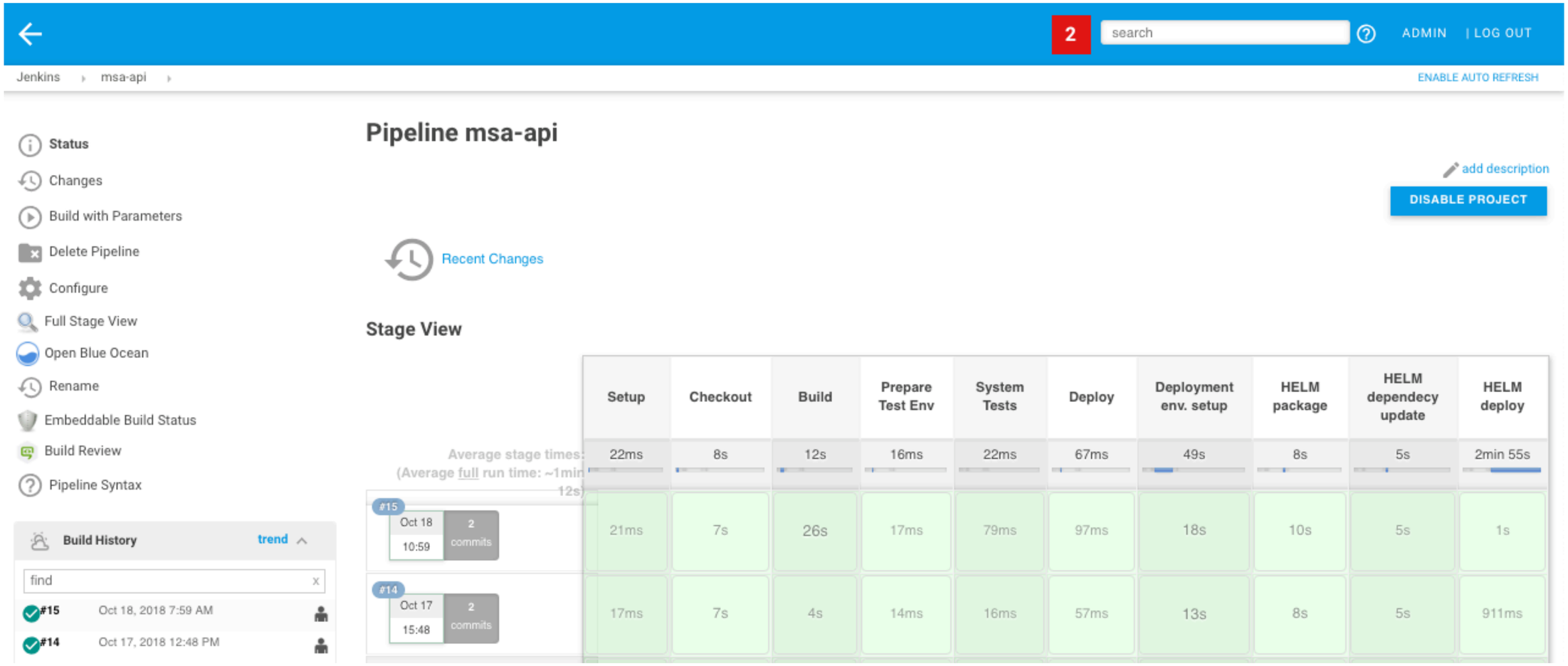
 | {{ if .Values.service.nodeIp }}

So Whats Next ? ...

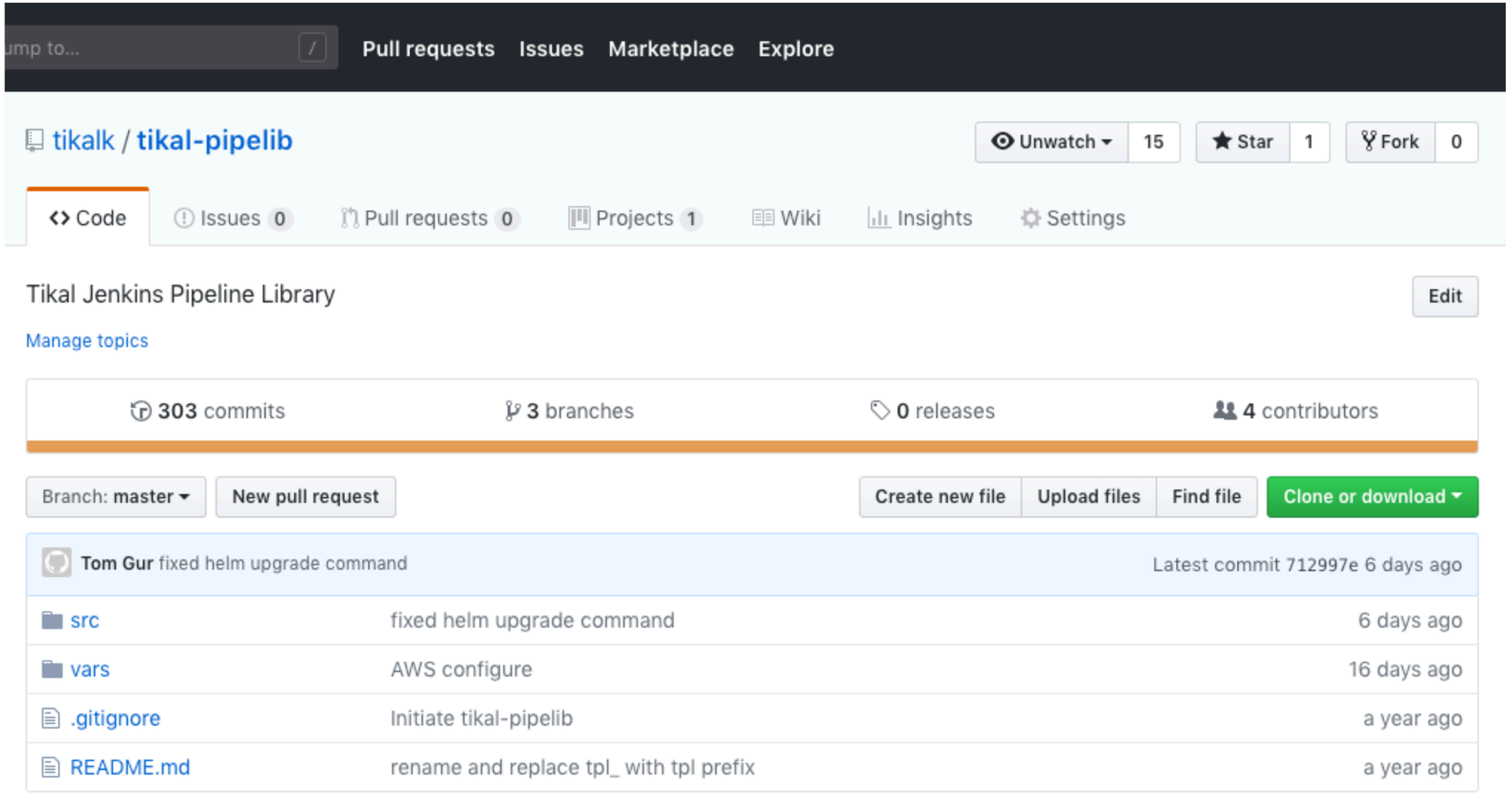
CONTINUES INTEGRATION PROCESS [JENKINS || GITLAB || “FOOBAR”]



JENKINS PIPELINE



PIPELINE LIBRARY – NOW SUPPORTS HELM LIFECYCLE



CREDITS

- Mark Kirshner & Anatoly Rabkin
- Helm documentation -> https://docs.helm.sh/
- ChaosKube helm chart -> https://github.com/helm/charts/tree/master/stable/chaoskube
- Icons -> https://github.com/octo-technology/kubernetes-icons
- Kubernetes docs -> https://kubernetes.io/docs/home/?path=users


Beautiful