Kubernetes ElasticSearch Environment Installation (Kubeapps / Helm)

Table of Contents

[Introduction 2](#_Toc536613281)

[GitHub Repository (smj-kubeapps) 2](#_Toc536613282)

[Kubernetes 'kubectl' Command-Line Utility 2](#_Toc536613283)

[Kubenetes 'helm' Command-Line Utility 2](#_Toc536613284)

[Kubeapps Overview 2](#_Toc536613285)

[Installing Kubeapps : smj-kubeapps Repository Scripts 3](#_Toc536613286)

[Installation Script : kubeapps/install\_kubeapps.sh 3](#_Toc536613287)

[Service Script : kubeapps/serve\_kubeapps.sh 6](#_Toc536613288)

[Kubernetes Dashboard Deployment (KubeApps) 7](#_Toc536613289)

[Kubernetes Dashboard Deployment (Helm) 10](#_Toc536613290)

[ElasticSearch Cluster Deployment (KubeApps) 12](#_Toc536613291)

[Cerebro Deployment 16](#_Toc536613292)

[Kibana Deployment 19](#_Toc536613293)

# Introduction

This document describes the steps required to build an ElasticSearch environment on Kubernetes using the 'kubeapps' application.

Kubeapps is an Open Source utility which provides a GUI onto Kubernetes Helm charts which allows installation/configuration of various applications.

It assumes that a working Kubernetes cluster environment is already available and the 'kubectl' command can access the cluster.

It also assumes that the 'helm' command-line utility is installed and available as the installation scripts depend on this.

**NOTE: In an environment with no internet access the KubeApps application web interface will not load correctly as it relies on external CSS so alternate installation instructions are also provided which use the Helm command line utility directly to install the various Helm Charts.**

# GitHub Repository (smj-kubeapps)

The Helm 'charts' and various supporting bash shell scripts can be found in the following GitHub repository.

[**https://github.com/smjgithub/smj-kubeapps**](https://github.com/smjgithub/smj-kubeapps)

**NOTE: This is a private repository but will be migrated to an internally available repository once it is set up. (Please contact steve@webtura.co.uk) for access.**

# Kubernetes 'kubectl' Command-Line Utility

[**https://kubernetes.io/docs/tasks/tools/install-kubectl**](https://kubernetes.io/docs/tasks/tools/install-kubectl/)

[**https://kubernetes.io/docs/reference/kubectl/overview**](https://kubernetes.io/docs/reference/kubectl/overview/)

# Kubenetes 'helm' Command-Line Utility

[**https://helm.sh**](https://helm.sh/)

# Kubeapps Overview

Kubeapps is an application dashboard for deploying to Kubernetes. The official project documentation can be found here:-

[**https://kubeapps.com**](https://kubeapps.com/)

Kubeapps is effectively a 'wrapper' around a set of Helm charts, each of which has a '**Chart.yaml'** file to define the chart, a package containing the contents and a **'values.yaml**' file which contains configuration options which can be modified as the application is deployed.

The package file can be produced for a given chart by changing directory to the folder of that chart and issuing the following command:-

helm package .

Each of these charts is described in a global **'index.yaml**' file which can be rebuilt by issuing the following command in the repository's root folder.

helm repo index .

The content and structure of the charts and their config files is beyond the scope of this document but more can be found at:-

[https://docs.helm.sh/chart\_template\_guide](https://docs.helm.sh/chart_template_guide/)

# Installing Kubeapps : smj-kubeapps Repository Scripts

Assuming that 'helm' and 'kubectl' command-line utilities are installed there are 2 bash shell scripts in the smj-kubeapps repository that can be used to install the 'kubeapps' application into the Kubenetes cluster environment.

**NOTE: The KubeApps application has a dependency on an external CSS resource so if deployed inside an environment without internet access then the web GUI will not work correctly. The Kubeapps application is not strictly necessary to install the various Helm charts but makes the process significantly easier than directly issuing Helm commands in a terminal but both approaches are covered in this document. It is possible to 'live' edit the HTML in Chrome developer tools to include the CSS in the browser but that is beyond the scope of this document and only intended as a quick fix. Some future work may build a version of Kubeapps without this dependency but this is currently a Work In Porgress...**

## Installation Script : kubeapps/install\_kubeapps.sh

**#!/usr/bin/env bash**# This ClusterRoleBinding allows kubernetes-dashboard higher level permissions  
*kubectl* apply -f config/dashboard-admin.yaml  
  
*kubectl* -n kube-system create sa tiller  
*kubectl* create clusterrolebinding tiller --clusterrole cluster-admin --serviceaccount=kube-system:tiller  
  
# Initialise the helm system by creating the Tiller control pod  
*helm* init --service-account tiller  
  
*echo* "Waiting 10 seconds for Tiller pod to initialise"  
**for** LOOP **in** {1..10}  
**do** *echo* -n "."  
 *sleep* 1  
**done***echo* ""  
*echo* "Installing Kubeapps..."  
  
# Install the kubeapps chart with initial custom chart repository  
*helm* install --name kubeapps --namespace kubeapps -f config/custom-values.yaml bitnami/kubeapps  
  
*kubectl* create serviceaccount kubeapps-operator  
*kubectl* create clusterrolebinding kubeapps-operator --clusterrole=cluster-admin --serviceaccount=default:kubeapps-operator

This script initially performs a 'helm init' to create the 'tiller' orchestration pod within the k8s environment after which Helm can be used to install the 'kubeapps' chart and all of its associated pods.

If everything works as expected you should see something similar to the following in the terminal:-

Steves-MBP-5:kubeapps stevejones$ ./install\_kubeapps.sh

clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created

serviceaccount/tiller created

clusterrolebinding.rbac.authorization.k8s.io/tiller created

$HELM\_HOME has been configured at /Users/stevejones/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.

To prevent this, run `helm init` with the --tiller-tls-verify flag.

For more information on securing your installation see: https://docs.helm.sh/using\_helm/#securing-your-helm-installation

Happy Helming!

$HELM\_HOME has been configured at /Users/stevejones/.helm.

Warning: Tiller is already installed in the cluster.

(Use --client-only to suppress this message, or --upgrade to upgrade Tiller to the current version.)

Happy Helming!

Waiting 10 seconds for Tiller pod to initialise

..........

Installing Kubeapps...

NAME: kubeapps

LAST DEPLOYED: Wed Dec 19 12:09:33 2018

NAMESPACE: kubeapps

STATUS: DEPLOYED

RESOURCES:

==> v1/ConfigMap

NAME AGE

kubeapps-internal-dashboard-config 19s

kubeapps-frontend-config 19s

==> v1beta1/CustomResourceDefinition

apprepositories.kubeapps.com 15s

==> v1beta1/ClusterRoleBinding

kubeapps-internal-apprepository-controller 15s

==> v1/Service

kubeapps-mongodb 15s

kubeapps-internal-chartsvc 15s

kubeapps-internal-dashboard 15s

kubeapps 15s

kubeapps-internal-tiller-proxy 15s

==> v1beta1/Deployment

kubeapps-mongodb 15s

==> v1/ServiceAccount

kubeapps-internal-apprepository-controller 19s

kubeapps-internal-tiller-proxy 19s

==> v1beta1/ClusterRole

kubeapps-internal-apprepository-controller 15s

==> v1beta1/Role

kubeapps-internal-apprepository-controller 15s

kubeapps-internal-tiller-proxy 15s

==> v1beta1/RoleBinding

kubeapps-internal-apprepository-controller 15s

kubeapps-internal-tiller-proxy 15s

==> v1beta2/Deployment

kubeapps-internal-apprepository-controller 15s

kubeapps-internal-chartsvc 15s

kubeapps-internal-dashboard 15s

kubeapps 15s

kubeapps-internal-tiller-proxy 15s

==> v1/Pod(related)

NAME READY STATUS RESTARTS AGE

kubeapps-mongodb-5898f98f66-ld7cw 0/1 Running 0 15s

kubeapps-internal-apprepository-controller-56894c9b47-q2sk9 1/1 Running 0 15s

kubeapps-internal-chartsvc-8574b469d9-47dx6 0/1 ContainerCreating 0 15s

kubeapps-internal-chartsvc-8574b469d9-5j77r 0/1 ContainerCreating 0 15s

kubeapps-internal-dashboard-7cf84f6fd-c6wmg 1/1 Running 0 15s

kubeapps-internal-dashboard-7cf84f6fd-rktbw 0/1 Running 0 15s

kubeapps-57f7f76c4-6phrz 0/1 ContainerCreating 0 15s

kubeapps-57f7f76c4-8pw87 0/1 Running 0 15s

kubeapps-internal-tiller-proxy-6f8d8c957b-2q4xz 1/1 Running 0 15s

kubeapps-internal-tiller-proxy-6f8d8c957b-tp4hs 0/1 ContainerCreating 0 15s

NOTES:

\*\* Please be patient while the chart is being deployed \*\*

Tip:

Watch the deployment status using the command: kubectl get pods -w --namespace kubeapps

Kubeapps can be accessed via port 80 on the following DNS name from within your cluster:

kubeapps.kubeapps.svc.cluster.local

To access Kubeapps from outside your K8s cluster, follow the steps below:

1. Get the Kubeapps URL by running these commands:

echo "Kubeapps URL: http://127.0.0.1:8080"

export POD\_NAME=$(kubectl get pods --namespace kubeapps -l "app=kubeapps" -o jsonpath="{.items[0].metadata.name}")

kubectl port-forward --namespace kubeapps $POD\_NAME 8080:8080

2. Open a browser and access Kubeapps using the obtained URL.

serviceaccount/kubeapps-operator created

clusterrolebinding.rbac.authorization.k8s.io/kubeapps-operator created

## Service Script : kubeapps/serve\_kubeapps.sh

Once Kubeapps is installed it can be exposed via HTTP on a port and accessed through a web browser using the 'serve\_kubeapps.sh' script:-

**#!/usr/bin/env bash***export* SECRET=**$***(kubectl get secret $(kubectl get serviceaccount kubeapps-operator -o jsonpath='{.secrets[].name}') -o jsonpath='{.data.token}' | base64 --decode)  
  
echo* "Secret Token = $SECRET"  
  
*echo* "Kubeapps URL: http://127.0.0.1:8080"  
*export* POD\_NAME=**$***(kubectl get pods --namespace kubeapps -l "app=kubeapps" -o jsonpath="{.items[0].metadata.name}")  
kubectl* port-forward --namespace kubeapps $POD\_NAME 8080:8080

If all works OK then you should see something similar to the following in your terminal:-

Steves-MBP-5:kubeapps stevejones$ ./serve\_kubeapps.sh

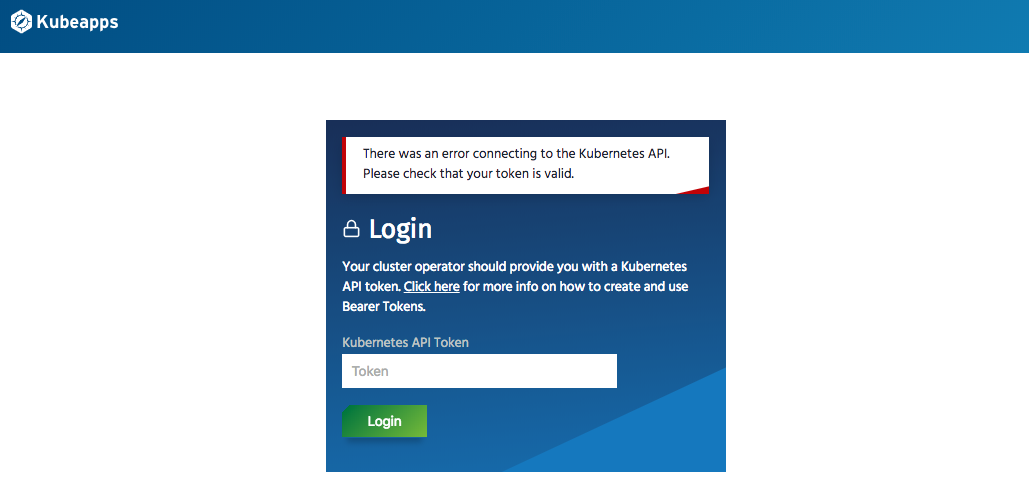
Secret Token = eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9..Ek3QxJ1OAxefonT8IjALkhS243sLklvWigEjVmJcdGdqS4B83oEVMC501nwT0WQVBTcrFApNpcnxyt8kP\_WI\_VjZluWDBWbRHDciTyUsGTKL-AzZfYfIfdN6KMjVI0Mk6qo0cR2SdYhtYtiaZn-2fA12pmyXlAPfPQVIz2ozy2wO59epypmlyNfSDGJ0cLXFHkCL0JFVWyNfl4s0JuWUbW6KgCqr6oVB6FLaiAV17d7RMeISw0BuNntX\_MaXOWFG95k7sVGuuTwEuQdbGsDs5K6Xi8yCgJp2jUTLKrkgD5OD6BAp\_5YnL\_WM6MZ0268uwRGfVv1G3fioTIZxOORVug

Kubeapps URL: http://127.0.0.1:8080

Forwarding from 127.0.0.1:8080 -> 8080

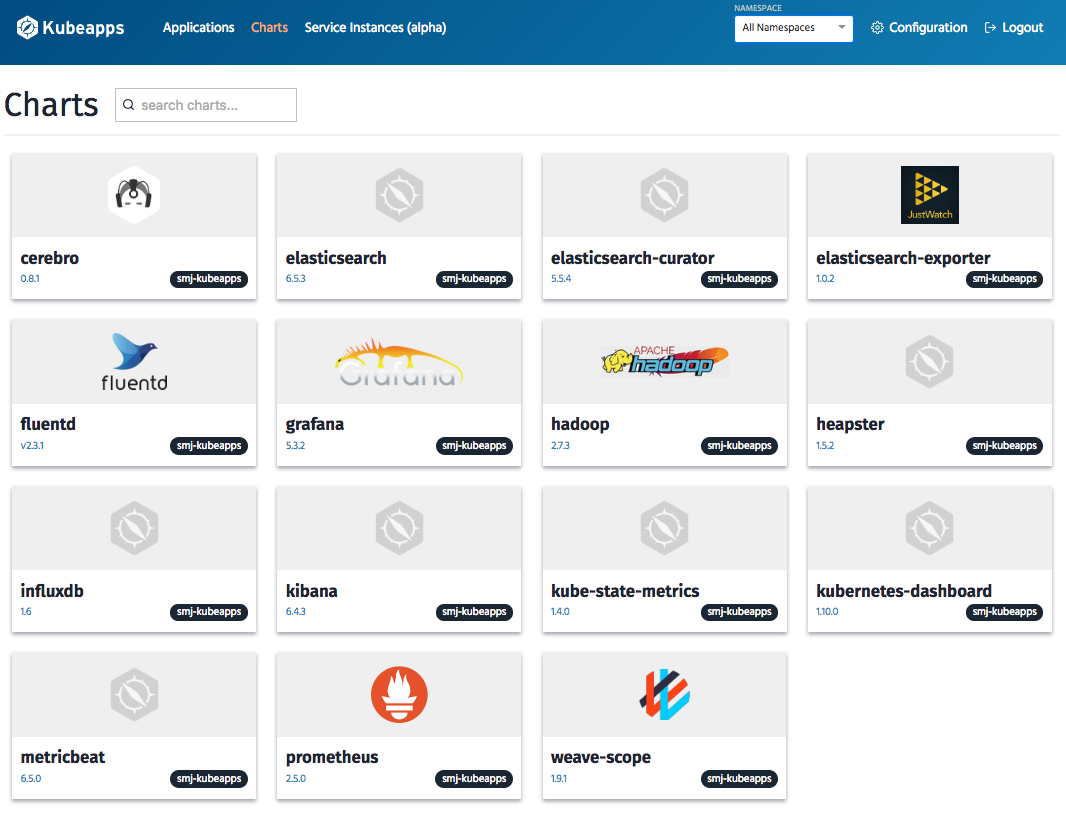
The 'Secret Token' value should be copied to the clipboard as this will now be need to log in to the Kubeapps GUI.

Launch the following URL in your web browser: **http://localhost:8080**



Paste the 'Secret Token' into the form and 'Login'...

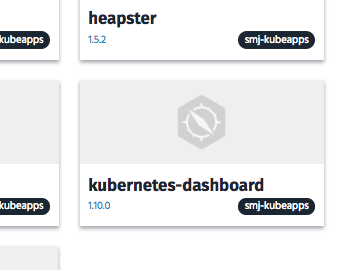
Select the 'Charts' menu option from the header bar and you should see the following which is a table of all the available Helm charts currently installed in Kubeapps that can be deployed:-



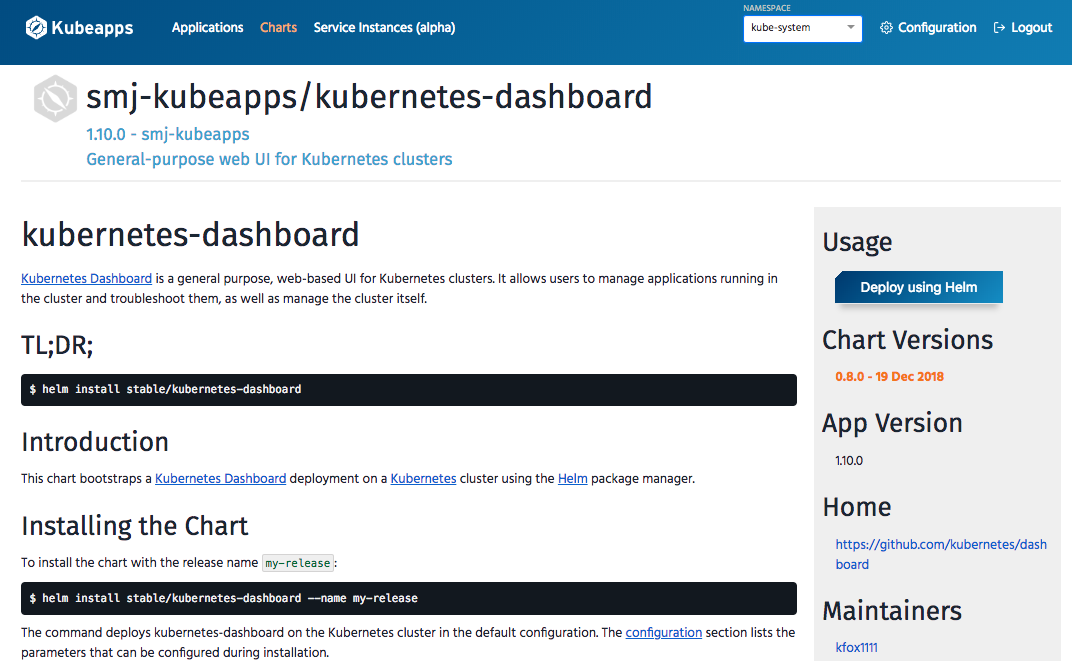
# Kubernetes Dashboard Deployment (KubeApps)

Kubernetes provides an API through which every aspect of configuration and monitoring can be performed but to give easier visibility and control there is an application called 'kubernetes-dashboard' which can be deployed to the cluster to provide a user-friendly web interface onto this.

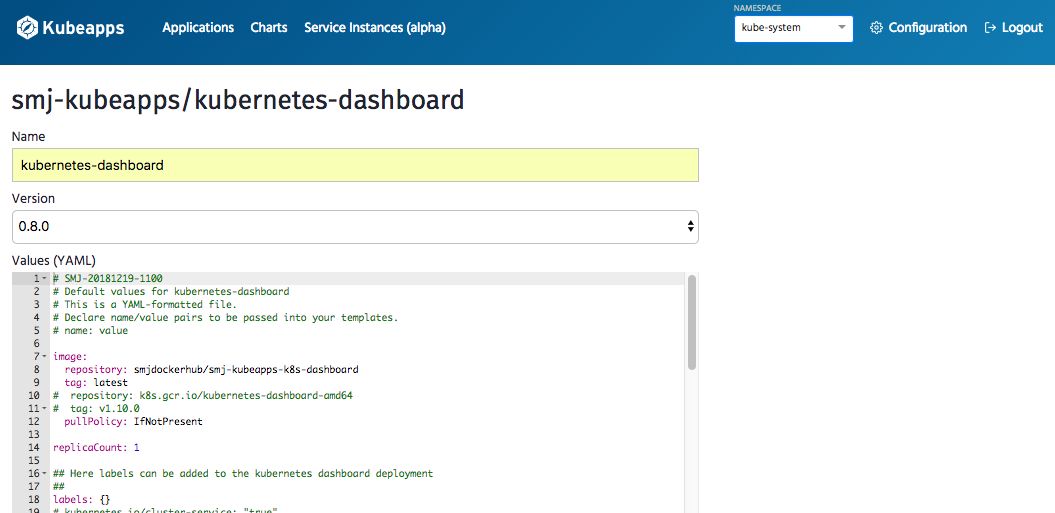
This application is available in the 'smj-kubeapps' respository and can be accessed directly from Kubeapps by clicking the button below:-



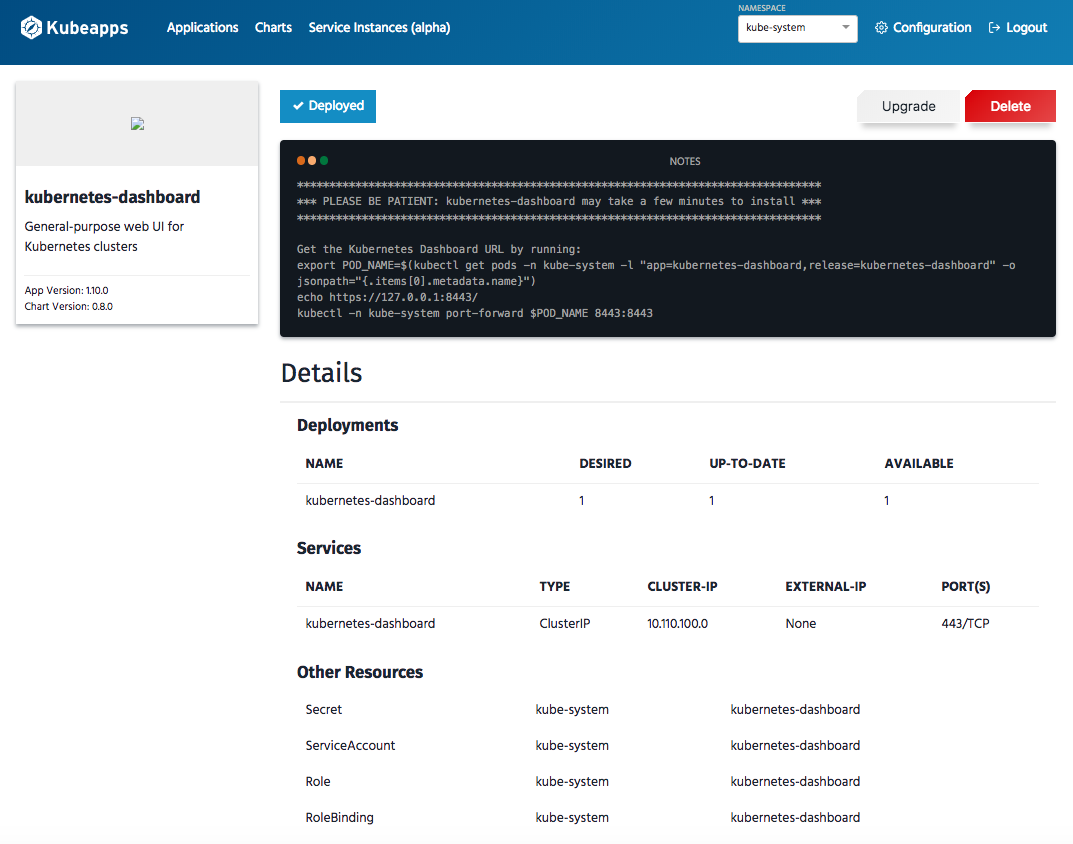
Switch to the 'kube-system' namespace and click the 'Deploy using Helm' button as shown below:-



Type in the name 'kubernetes-dashboard' into the 'Name' field and click 'Submit'...



The display will briefly show 'Deploying...' but should then look as follows:-



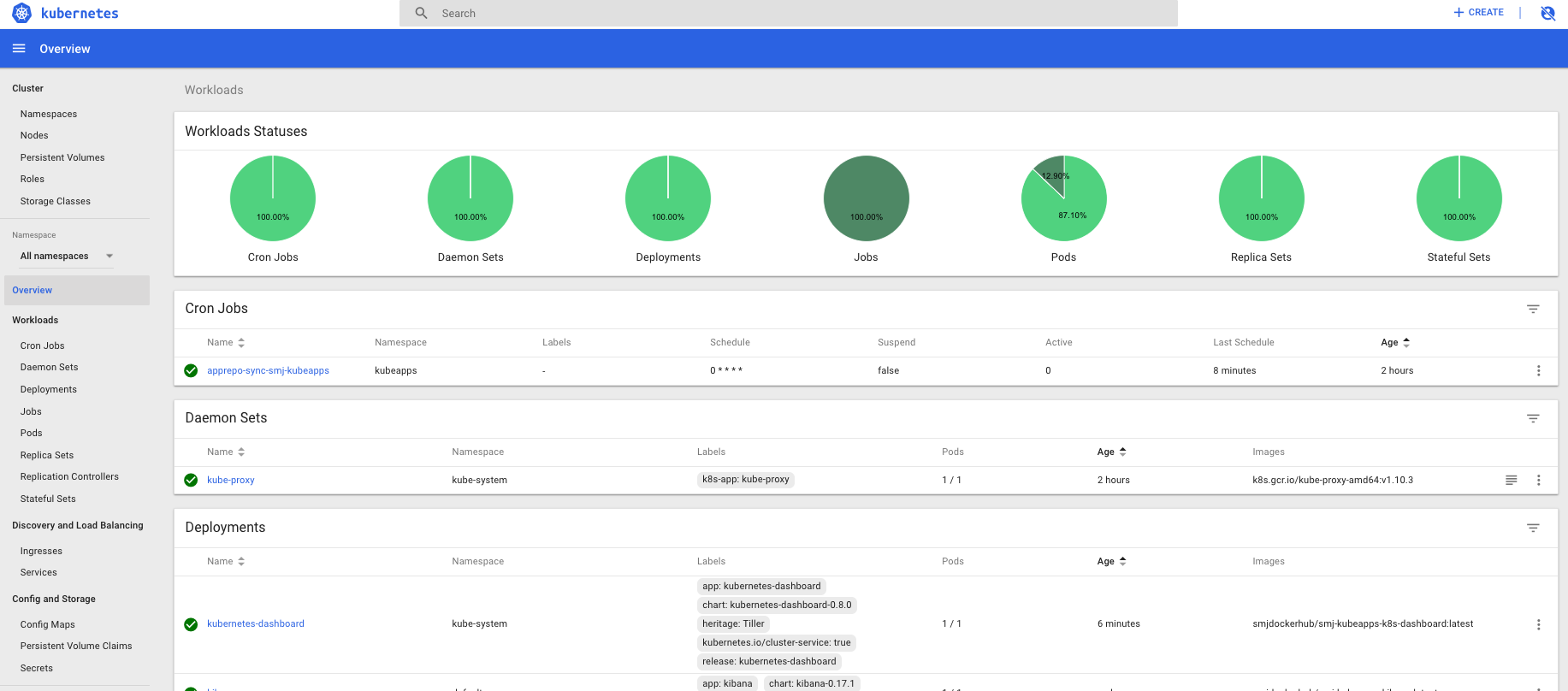
The Kubernetes Dashboard is now deployed and can be accessed by setting up a 'proxy' to the Kubernetes API as follows:-

Steves-MBP-5:kubeapps stevejones$ kubectl proxy

Starting to serve on 127.0.0.1:8001

Now launch the following URL in a web browser:-

[http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/ - !/overview?namespace=\_all](http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/overview?namespace=_all)



You can now explore your Kubenetes cluster in detail and even modify and change configuration, scale resources, edit, delete, etc...

NOTE: USE WITH CAUTION!!!!

# Kubernetes Dashboard Deployment (Helm)

If the KubeApps GUI is not working correctly due to the CSS dependency issue mentioned earlier then Kubernetes Dashboard can be directly deployed on the command line using the Helm CLI utility.

To do this first open a terminal and changge directory into the 'kubernetes-dashboard' folder inside the 'smj-kubeapps' repository.

The following command will then install Kubernetes Dashboard:-

helm install --name kubernetes-dashboard --namespace kube-system -f values.yaml .

This will install the Kubernetes Dashboard into the 'kube-system' namespace and will use configuration values found in the local 'values.yaml' file and also use the chart found in the local '.' folder. If all works as expected the following output should be seen:-

NAME: kubernetes-dashboard

LAST DEPLOYED: Mon Jan 28 11:35:10 2019

NAMESPACE: kube-system

STATUS: DEPLOYED

RESOURCES:

==> v1beta1/RoleBinding

NAME AGE

kubernetes-dashboard 0s

==> v1/Service

kubernetes-dashboard 0s

==> v1beta1/Deployment

kubernetes-dashboard 0s

==> v1/Pod(related)

NAME READY STATUS RESTARTS AGE

kubernetes-dashboard-7d47c7b6c9-2tcxp 0/1 ContainerCreating 0 0s

==> v1/Secret

NAME AGE

kubernetes-dashboard 0s

==> v1/ServiceAccount

kubernetes-dashboard 0s

==> v1beta1/Role

kubernetes-dashboard 0s

NOTES:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* PLEASE BE PATIENT: kubernetes-dashboard may take a few minutes to install \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Get the Kubernetes Dashboard URL by running:

export POD\_NAME=$(kubectl get pods -n kube-system -l "app=kubernetes-dashboard,release=kubernetes-dashboard" -o jsonpath="{.items[0].metadata.name}")

echo https://127.0.0.1:8443/

kubectl -n kube-system port-forward $POD\_NAME 8443:8443

NOTE: The 'NOTES' info in the output has not yet been updated as this is a work in progress so do not take this as definitive documentation. This will be updated during production hardening of the environments.

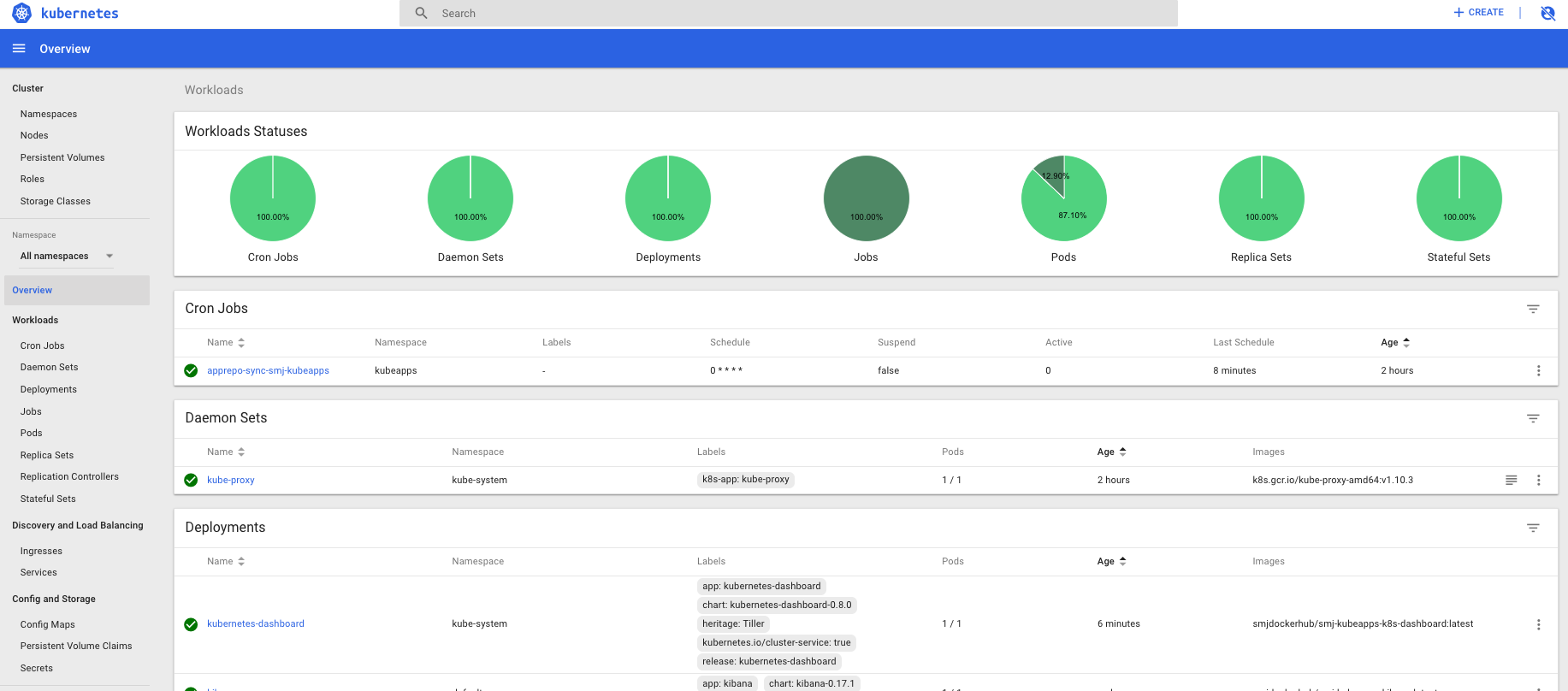
You will now be able to 'proxy' to the Kubernetes API server and serve the Kubernetes Dashboard for viewing in the browser:-

Steves-MBP-5:kubeapps stevejones$ kubectl proxy

Starting to serve on 127.0.0.1:8001

Now launch the following URL in a web browser:-

[http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/ - !/overview?namespace=\_all](http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/overview?namespace=_all)



You can now explore your Kubenetes cluster in detail and even modify and change configuration, scale resources, edit, delete, etc...

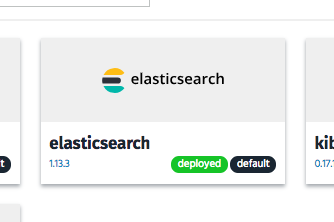
NOTE: USE WITH CAUTION!!!!

# ElasticSearch Cluster Deployment (KubeApps)

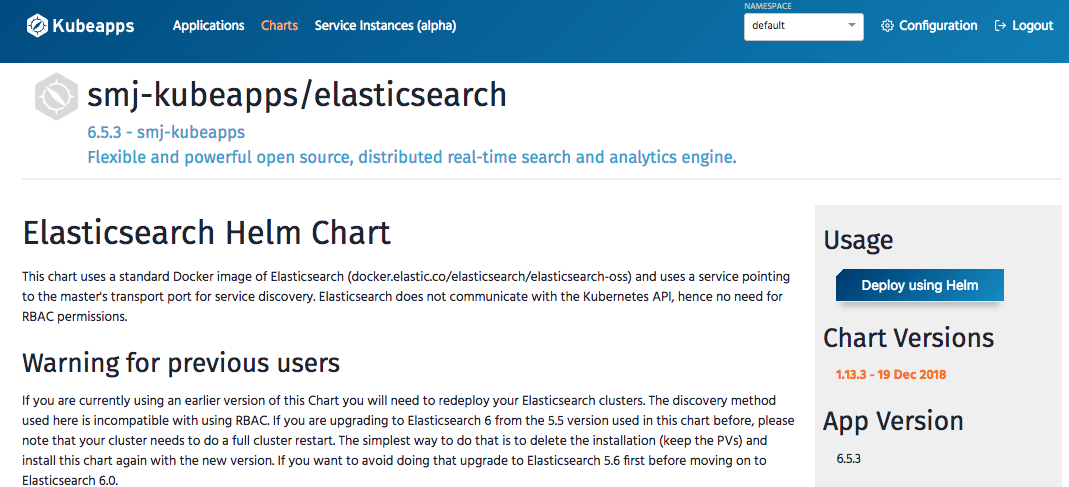
The base 'kubeapps' application is now running and served via HTTP and also we have a 'kubectl proxy' into the Kubernetes Dashboard so we are now ready to install applications.

In the case of ElasticSearch we will deploy this into the 'default' namespace for now but this may well be deployed into a specific 'elastic' namespace in future along with any other applications which form part of its eco-system, e.g. Cerebro, Kibana, MetricBeat or whatever is decided.

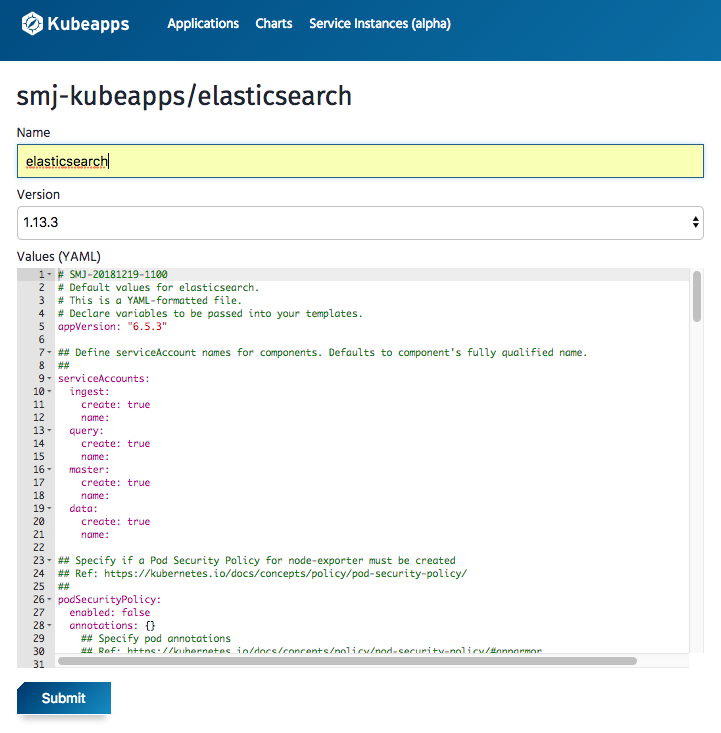
To begin the ElasticSearch deployment click on the 'elasticsearch' button as shown below:-



Click on the 'Deploy using Helm' button on the following screen:-

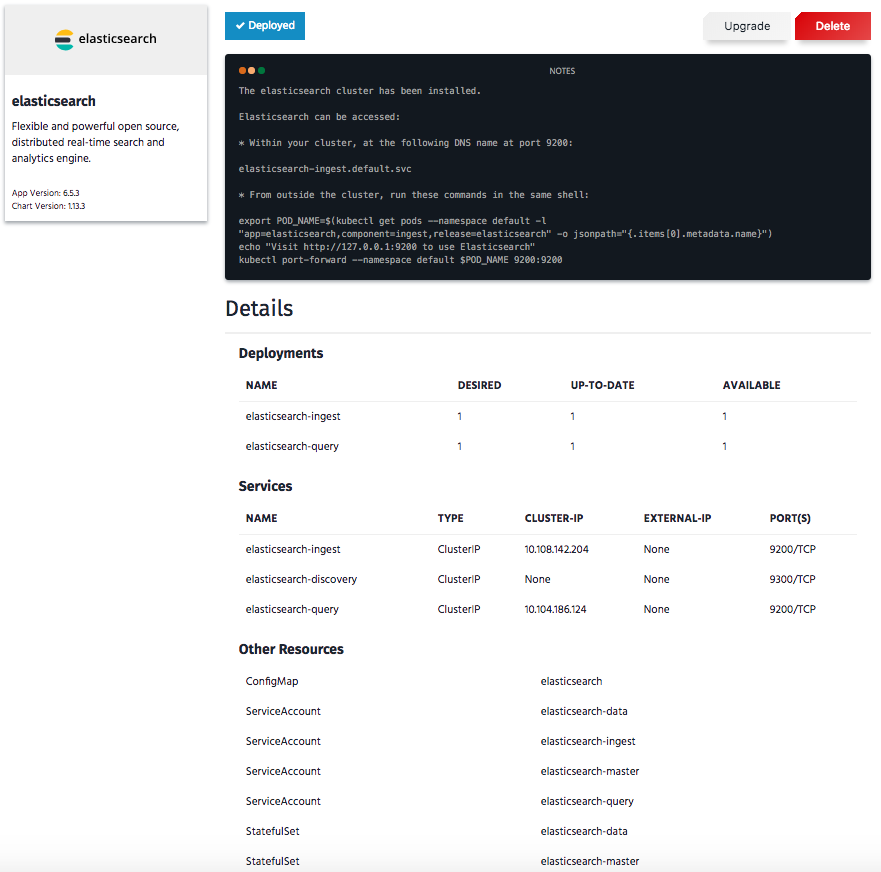


Fill in the 'Name' field with release name of 'elasticsearch' and click 'Submit'...

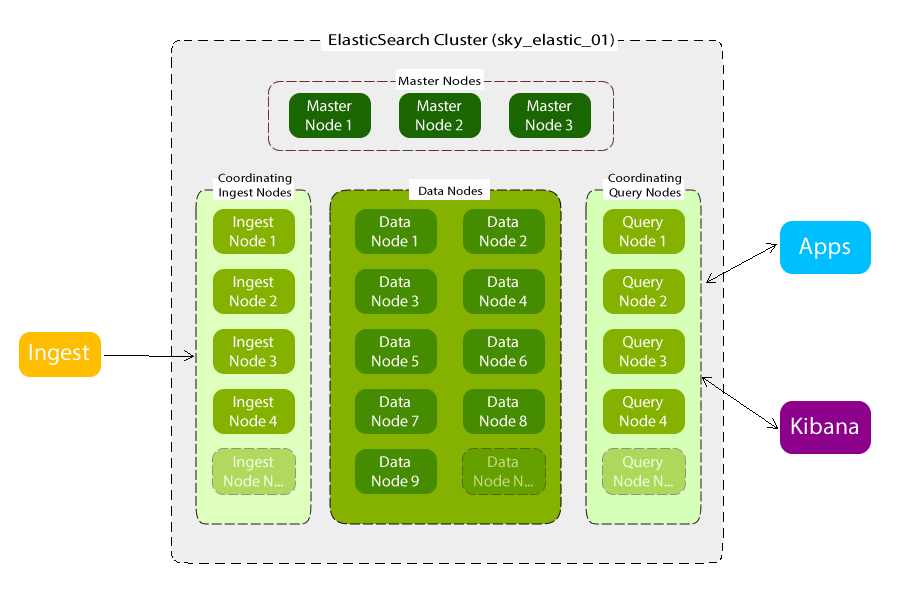


This deployment will take a little while as it has a number of ElasticSearch nodes to create including allocation of storage space for the cluster and also various services to provide inter-node communication and also exposure to external services.

The progress of this can be checked in the Kubernetes Dashboard. Once complete the following should be seen:-



This deployment is a very limited deployment with minimal CPU, storage and memory specified to enable it to run on a basic environment, e.g. a laptop. This can be scaled up once the deployment is complete but the following diagram shows the basic structure of the ElasticSearch cluster and node types.



In the initial basic deployment there are 3 Master Nodes, 2 Data Nodes, 1 Ingest Node and 1 Query Node. The Ingest and Query nodes are exposed via LoadBalancer services on ports 9201 and 9202 so that the cluster can be accessed externally.

All index writes (ingest) should be directed to the 'elasticsearch-ingest' service and applications and Kibana should direct queries to the 'elasticsearch-query' service.

# ElasticSearch Cluster Deployment (Helm)

If the KubeApps GUI is not working correctly due to the CSS dependency issue mentioned earlier then ElasticSearch can be directly deployed on the command line using the Helm CLI utility.

To do this first open a terminal and changge directory into the 'elasticsearch' folder inside the 'smj-kubeapps' repository.

The following command will then install Kubernetes Dashboard:-

helm install --name elasticsearch --namespace default -f values.yaml .

This will install ElasticSearch into the 'default' namespace and will use configuration values found in the local 'values.yaml' file and also use the chart found in the local '.' folder. If all works as expected the following output should be seen:-

NAME: elasticsearch

LAST DEPLOYED: Mon Jan 28 11:40:26 2019

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/ConfigMap

NAME AGE

elasticsearch 0s

==> v1/ServiceAccount

elasticsearch-data 0s

elasticsearch-ingest 0s

elasticsearch-master 0s

elasticsearch-query 0s

==> v1/Service

elasticsearch-ingest 0s

elasticsearch-discovery 0s

elasticsearch-query 0s

==> v1beta1/Deployment

elasticsearch-ingest 0s

elasticsearch-query 0s

==> v1beta1/StatefulSet

elasticsearch-data 0s

elasticsearch-master 0s

==> v1/Pod(related)

NAME READY STATUS RESTARTS AGE

elasticsearch-ingest-74457576bd-6hjnv 0/1 Init:0/1 0 0s

elasticsearch-query-7b557c4bc-zsmhb 0/1 Init:0/1 0 0s

elasticsearch-data-0 0/1 Pending 0 0s

elasticsearch-master-0 0/1 Pending 0 0s

NOTES:

The elasticsearch cluster has been installed.

Elasticsearch can be accessed:

\* Within your cluster, at the following DNS name at port 9200:

elasticsearch-ingest.default.svc

\* From outside the cluster, run these commands in the same shell:

WARNING: You have likely exposed your Elasticsearch cluster direct to the internet.

Elasticsearch does not implement any security for public facing clusters by default.

As a minimum level of security; switch to ClusterIP/NodePort and place an Nginx gateway infront of the cluster in order to lock down access to dangerous HTTP endpoints and verbs.

NOTE: It may take a few minutes for the LoadBalancer IP to be available.

You can watch the status of by running 'kubectl get svc -w elasticsearch-ingest'

export SERVICE\_IP=$(kubectl get svc --namespace default elasticsearch-ingest -o jsonpath='{.status.loadBalancer.ingress[0].ip}')

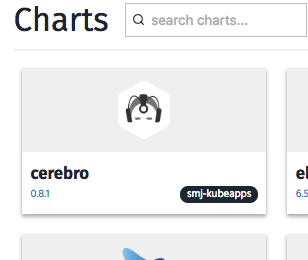
echo http://$SERVICE\_IP:9200

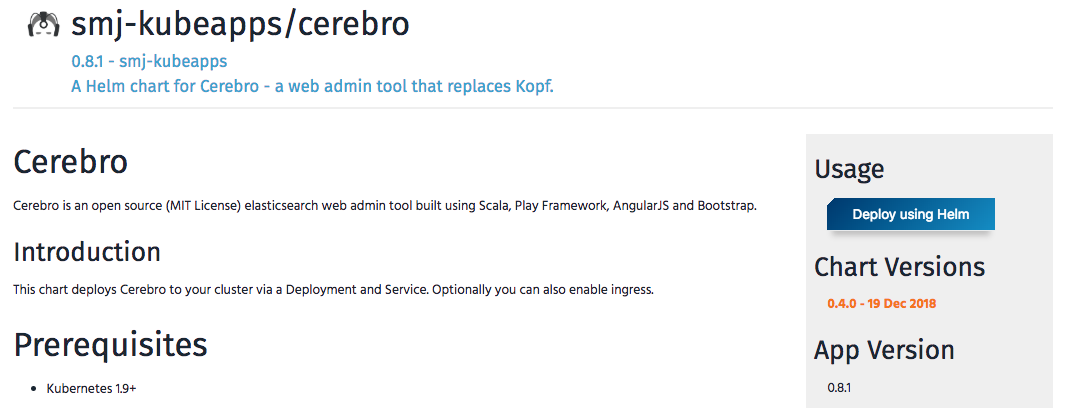
NOTE: The 'NOTES' info in the output has not yet been updated as this is a work in progress so do not take this as definitive documentation. This will be updated during production hardening of the environments.

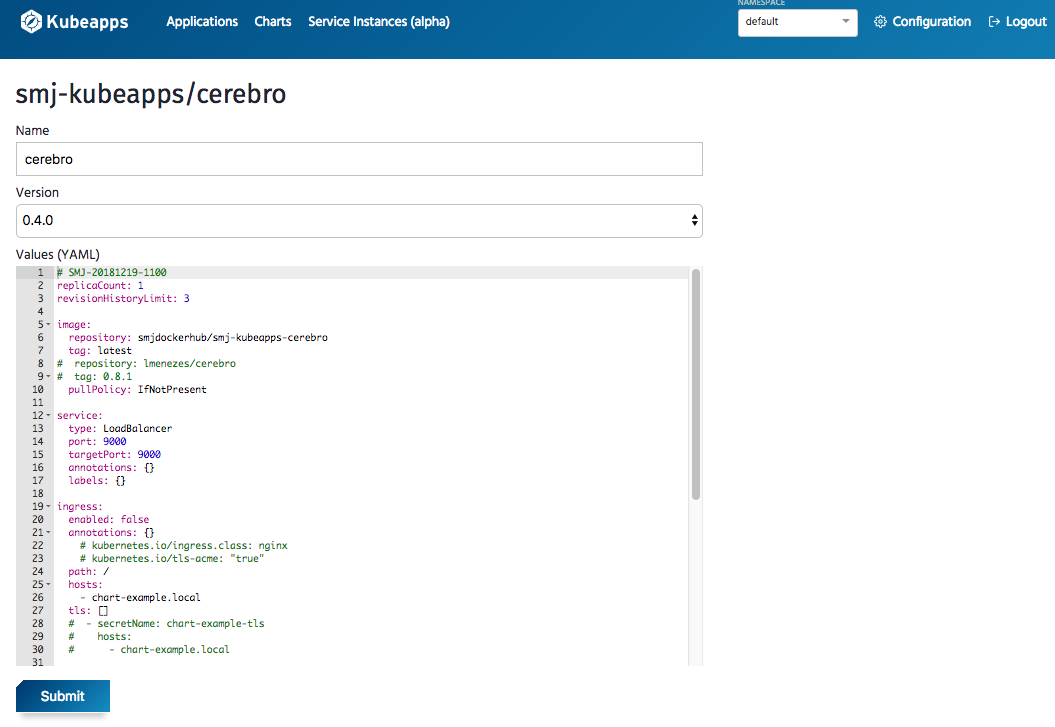
# Cerebro Deployment (KubeApps)

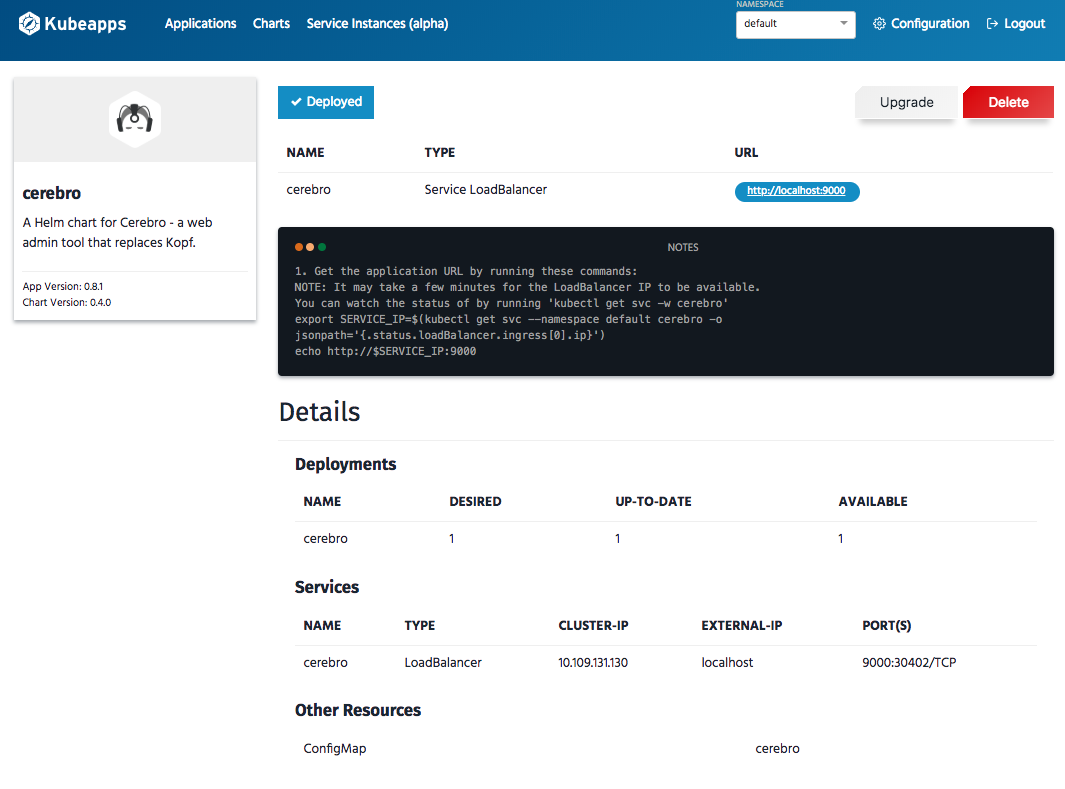
Cerebro is an Open Source application which is very popular for connecting to ElasticSearch clusters and provides a number of useful facilities which are effectively wrappers around the ElasticSearch HTTP RESTful API.

It can be deployed from Kubeapps using the following chart:-

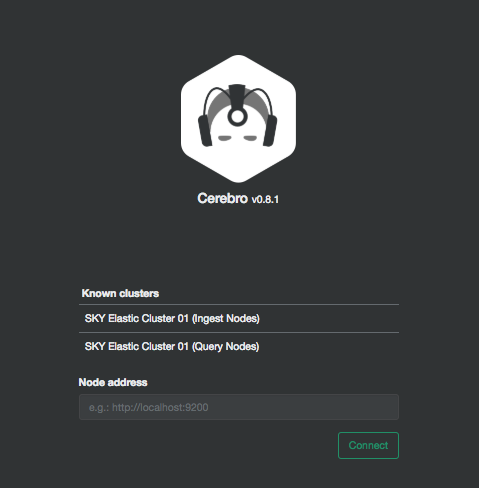






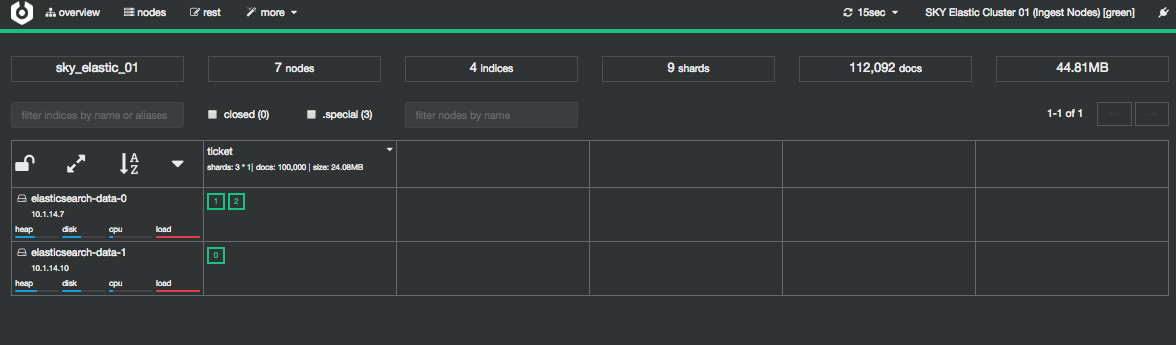


The Cerebro deployment exposes port 9000 which can be connected to from a web browser simply by clinking the link on the deployment status page:-



For convenience 2 connection profiles are already configured in the deployment which will connect either to the 'elasticsearch-ingest' service or the 'elasticsearch-query' service which are both access points to the same cluster.

Click on either to connect to the Sky Elastic Cluster 01 (sky\_elastic\_01) cluster in Cerebro.



For more information on this application please refer to the author's documentation...

<https://github.com/lmenezes/cerebro>

# Cerebro Deployment (Helm)

If the KubeApps GUI is not working correctly due to the CSS dependency issue mentioned earlier then Cerebro can be directly deployed on the command line using the Helm CLI utility.

To do this first open a terminal and changge directory into the cerebro folder inside the 'smj-kubeapps' repository.

The following command will then install Kubernetes Dashboard:-

helm install --name cerebro --namespace default -f values.yaml .

This will install Cerebro into the 'default' namespace and will use configuration values found in the local 'values.yaml' file and also use the chart found in the local '.' folder. If all works as expected the following output should be seen:-

NAME: cerebro

LAST DEPLOYED: Mon Jan 28 11:50:42 2019

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Service

NAME AGE

cerebro 0s

==> v1/Deployment

cerebro 0s

==> v1/Pod(related)

NAME READY STATUS RESTARTS AGE

cerebro-f7658b85d-t6vk9 0/1 Init:0/1 0 0s

==> v1/ConfigMap

NAME AGE

cerebro 0s

NOTES:

1. Get the application URL by running these commands:

NOTE: It may take a few minutes for the LoadBalancer IP to be available.

You can watch the status of by running 'kubectl get svc -w cerebro'

export SERVICE\_IP=$(kubectl get svc --namespace default cerebro -o jsonpath='{.status.loadBalancer.ingress[0].ip}')

echo http://$SERVICE\_IP:9000

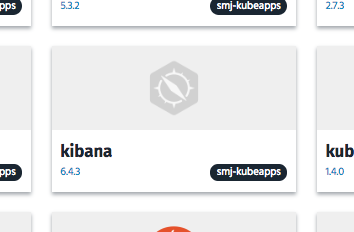
Browser: http://localhost:9000

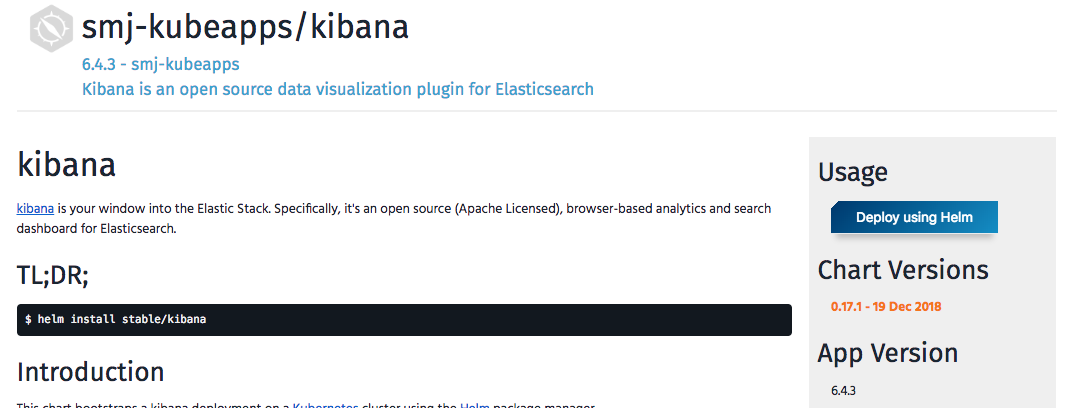
NOTE: The 'NOTES' info in the output has not yet been updated as this is a work in progress so do not take this as definitive documentation. This will be updated during production hardening of the environments.

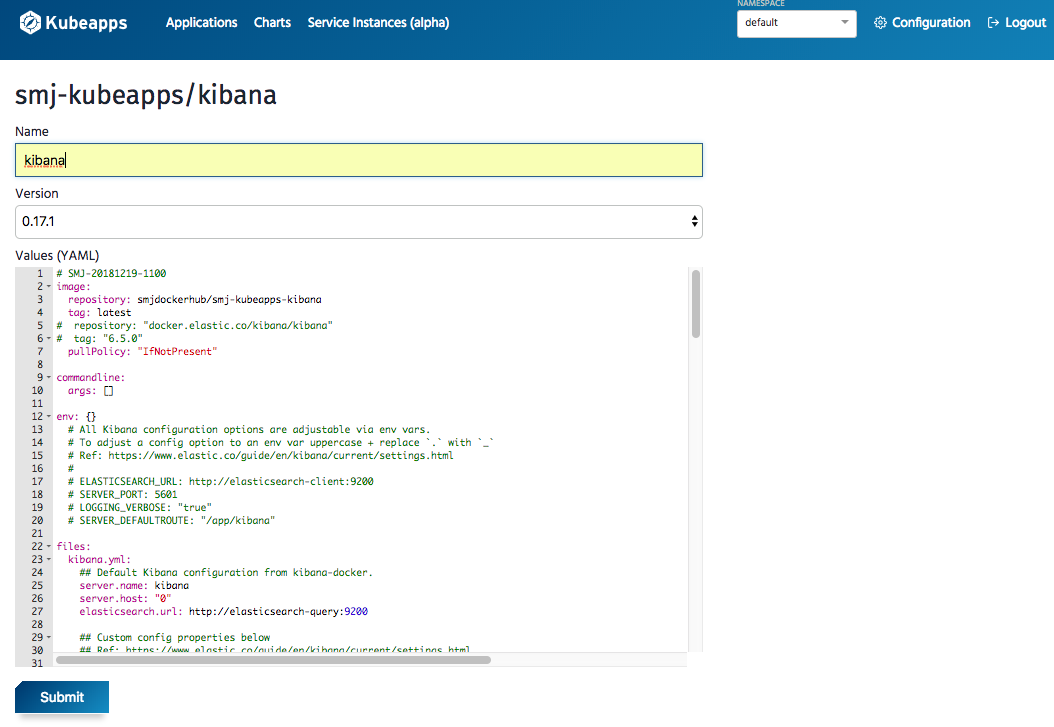
# Kibana Deployment (KubeApps)

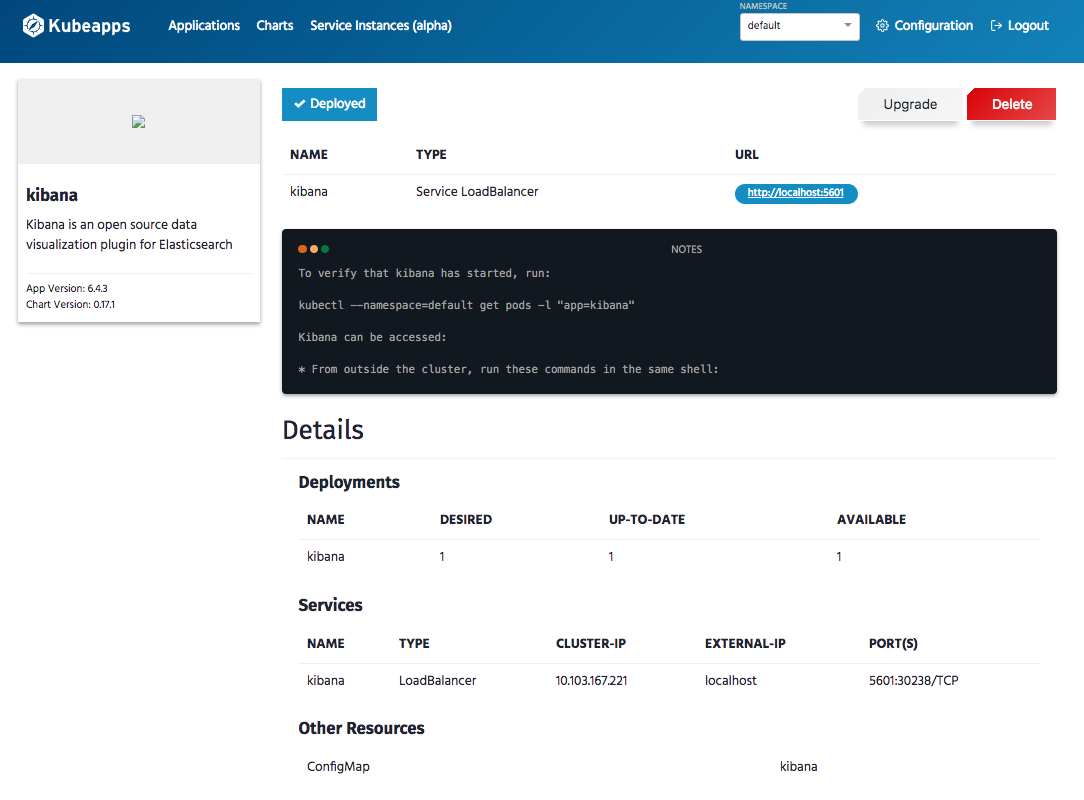
The final deployment in this initial basic ElasticSearch environment is Kibana.

The deployment can be launched from the following button:-

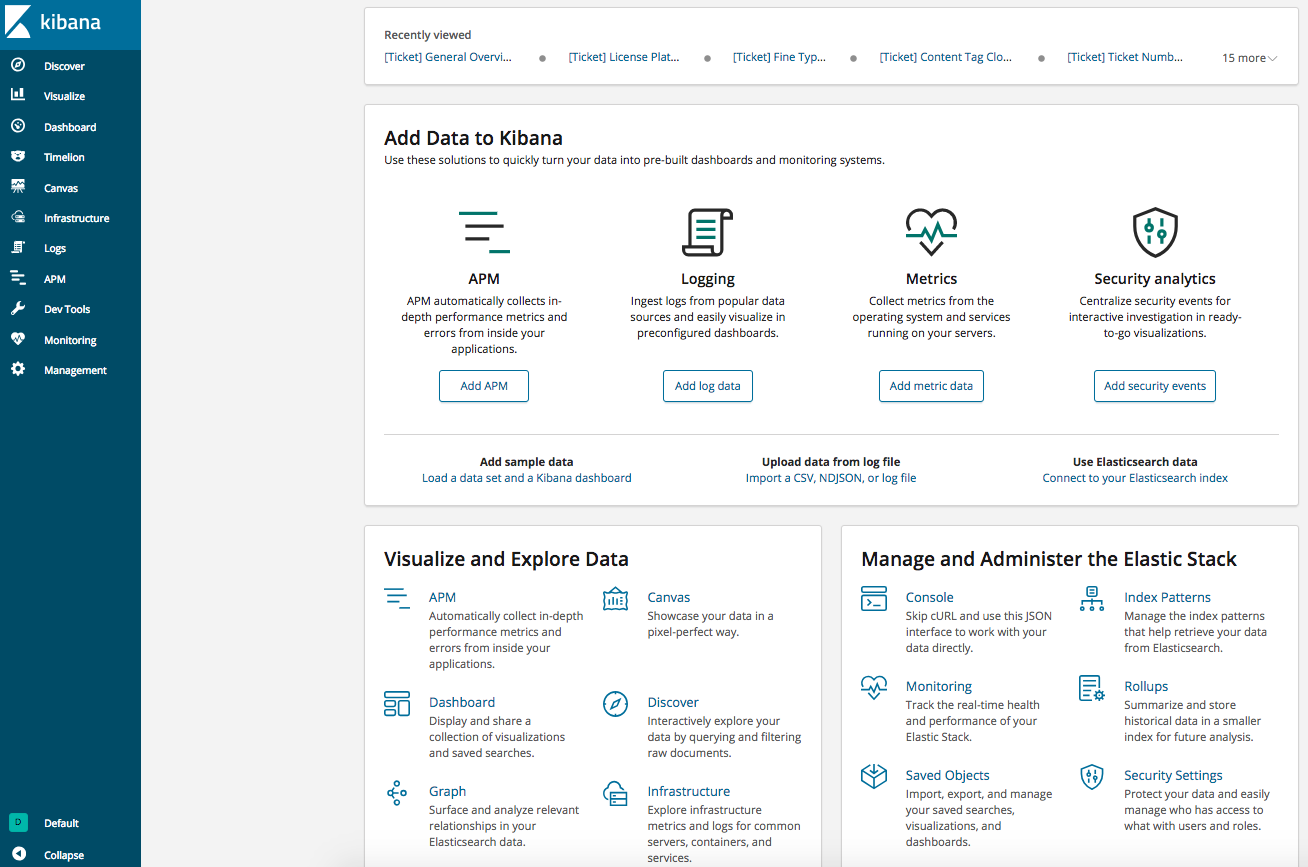






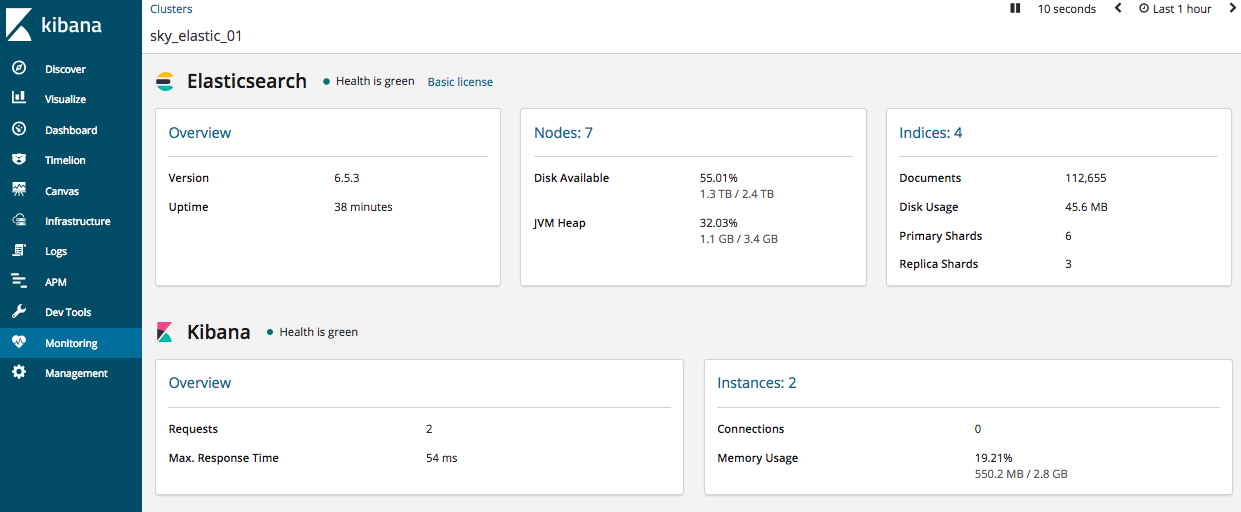


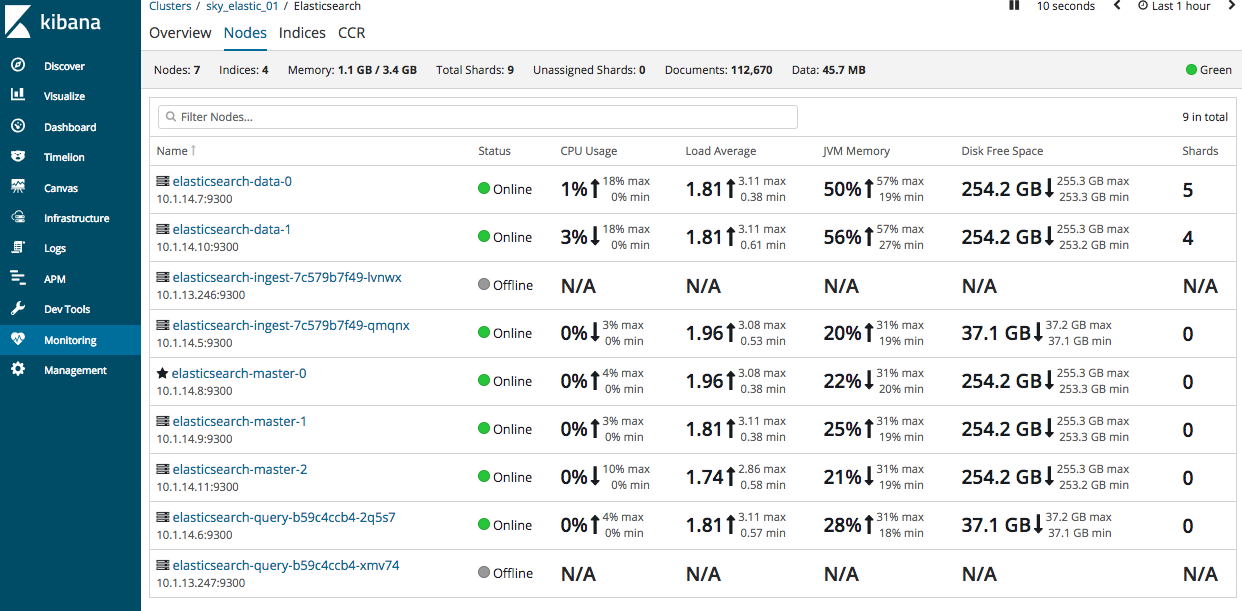
The Kibana deployment exposes port 5601 which can be connected to from a web browser. Click the <http://localhost:5601/> link to launch Kibana.



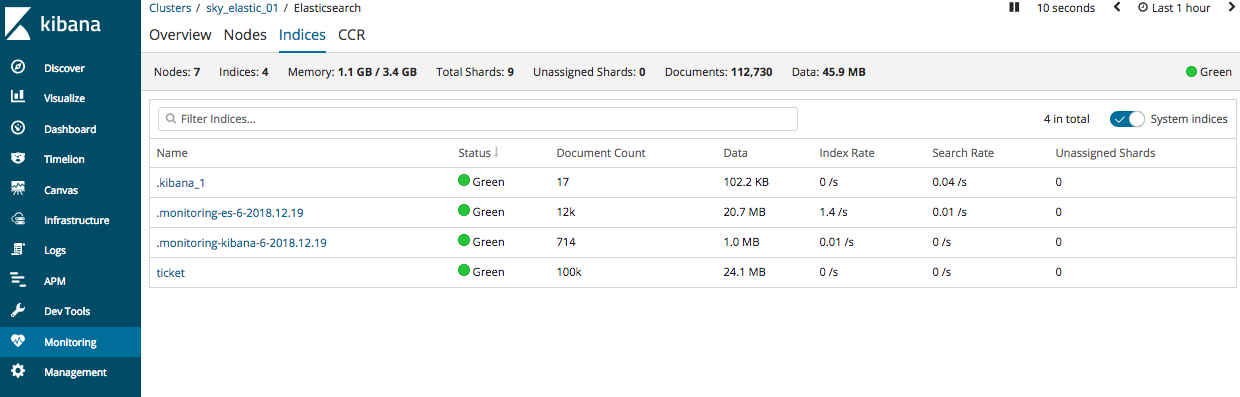
This deployment is configured to connect to the 'elasticsearch-query' service within the Kubernetes cluster so is ready to go and communicating with the sky\_elastic\_01 cluster.

X-Pack Monitoring is also configured so clicking on the 'Monitoring' link on the left navigation bar will allow you to also see the status of the cluster.









All other Kibana features are also available so the cluster is ready to use.....!!!!

# Kibana Deployment (Helm)

If the KubeApps GUI is not working correctly due to the CSS dependency issue mentioned earlier then Kibana can be directly deployed on the command line using the Helm CLI utility.

To do this first open a terminal and change directory into the kibana folder inside the 'smj-kubeapps' repository.

The following command will then install Kubernetes Dashboard:-

helm install --name kibana --namespace default -f values.yaml .

This will install Kibana into the 'default' namespace and will use configuration values found in the local 'values.yaml' file and also use the chart found in the local '.' folder. If all works as expected the following output should be seen:-

NAME: kibana

LAST DEPLOYED: Mon Jan 28 11:51:55 2019

NAMESPACE: default

STATUS: DEPLOYED

RESOURCES:

==> v1/Service

NAME AGE

kibana 0s

==> v1beta1/Deployment

kibana 0s

==> v1/Pod(related)

NAME READY STATUS RESTARTS AGE

kibana-5dd6894cc9-2sdrk 0/1 ContainerCreating 0 0s

==> v1/ConfigMap

NAME AGE

kibana 0s

NOTES:

To verify that kibana has started, run:

kubectl --namespace=default get pods -l "app=kibana"

Kibana can be accessed:

\* From outside the cluster, run these commands in the same shell:

Browser: http://localhost:5601

NOTE: The 'NOTES' info in the output has not yet been updated as this is a work in progress so do not take this as definitive documentation. This will be updated during production hardening of the environments.