Navigator Documentation

Release 0.0.0

Jetstack Ltd.

Mar 27, 2018

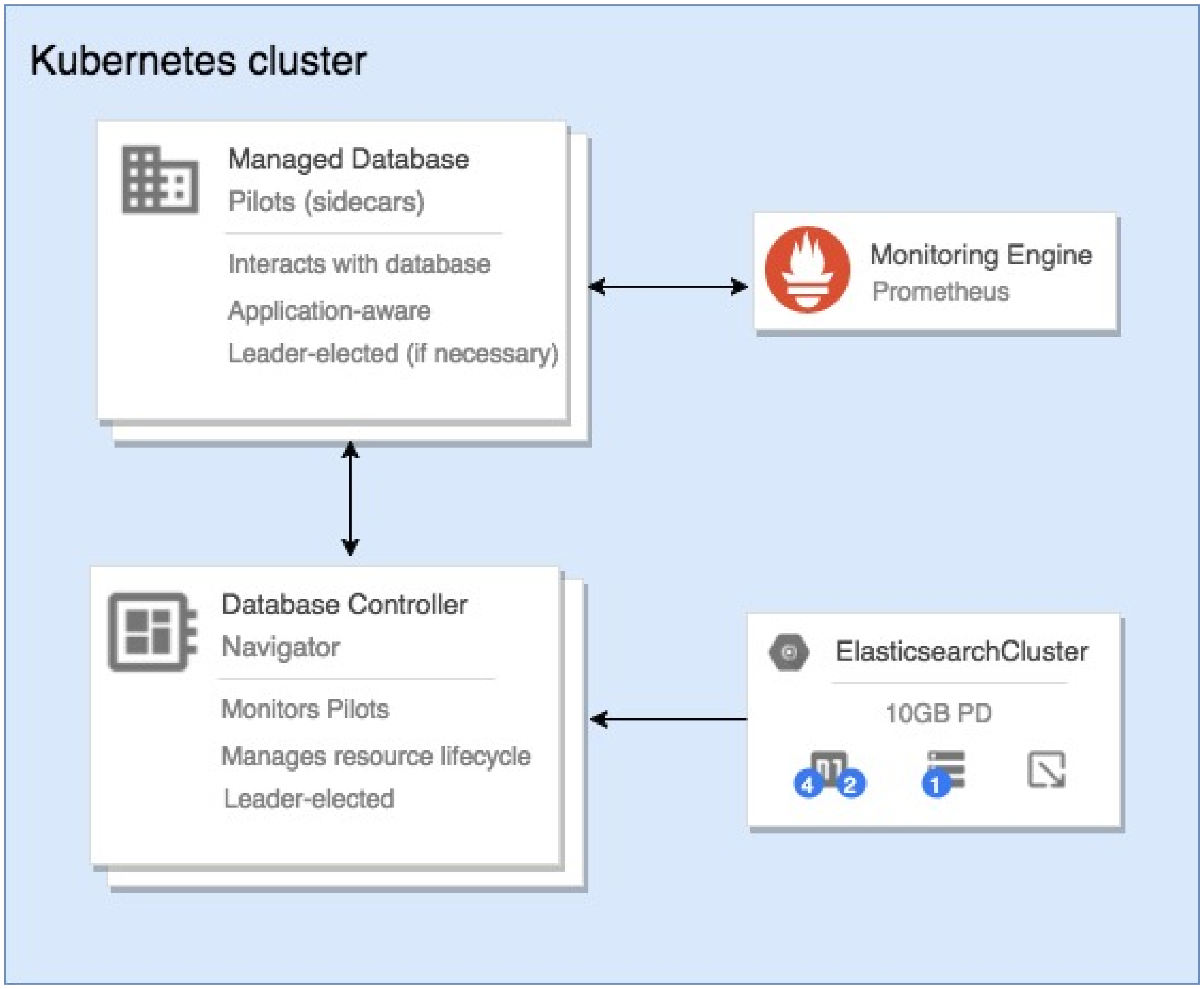
Contents:

Navigator is a Kubernetes extension for managing common stateful services on Kubernetes. It is implemented as a custom apiserver that operates behind kube-aggregator and introduces a variety of new Kubernetes resource types.

As well as following "the operator model", Navigator additionally introduces the concept of 'Pilots' - small 'nanny' processes that run inside each pod in your application deployment. These Pilots are responsible for managing the lifecycle of your underlying application process (e.g. an Elasticsearch JVM process) and periodically report state information about the individual node back to the Navigator API.

By separating this logic into it's own binary that is run alongside each node, in certain failure events the Pilot is able to intervene in order to help prevent data loss, or otherwise update the Navigator API with details of the failure so that navigator-controller can take action to restore service.

- **navigator-apiserver** - this takes on a similar role to `kube-apiserver`. It is responsible for storing and coordinating all of the state stored for Navigator. It requires a connection to an etcd cluster in order to do this. In order to make Navigator API types generally consumable to users of your cluster, it registers itself with kube-aggregator. It performs validation of your resources, as well as performing conversions between API versions which allow us to maintain a stable API without hindering development.
- **navigator-controller** - the controller is akin to `kube-controller-manager`. It is responsible for actually realizing your deployments within the Kubernetes cluster. It can be seen as the 'operator' for the various applications supported by `navigator-apiserver`.
- **pilots** - the pilot is responsible for managing each database process. Currently Navigator has two types: `pilot-elasticsearch` and `pilot-cassandra`.



CHAPTER 1

Quick-start

Here we're going to deploy a distributed and scalable Elasticsearch cluster using the examples provided in this repository. This will involve first deploying Navigator, and then creating an `ElasticsearchCluster` resource. All management of the Elasticsearch cluster will be through changes to the ElasticsearchCluster manifest.

1.1 1) Install Navigator using Helm

```
$ helm install contrib/charts/navigator --name navigator --namespace navigator --wait
```

You should see the Navigator service start in the `navigator` namespace:

```
$ kubectl get pod -n navigator
NAME                         READY     STATUS          RESTARTS    AGE
navigator-745449320-dcgms    1/1       Running         0           30s
```

1.2 2) Prepare your Kubernetes nodes

Elasticsearch requires certain important system configuration settings to be configured on the host operating system i.e. on the Kubernetes node. For this demonstration, it should be sufficient to run `sysctl -w vm.max_map_count=262144`, which increases a particular virtual memory limit. You can quickly run the command on *all* your Kubernetes nodes by installing the following `DaemonSet`:

```
$ kubectl apply -f docs/quick-start/sysctl-daemonset.yaml
```

Or you can log into each node and run the command by hand.

See *System Configuration for Elasticsearch Nodes* for more information.

1.3 3) Create an Elasticsearch cluster

```
$ kubectl create -f docs/quick-start/es-cluster-demo.yaml
```

This will deploy a multi-node Elasticsearch cluster, split into nodes of 3 roles: master, client (ingest) and data. There will be 4 data nodes, each with a 10GB PV, 2 client nodes, and 3 master nodes. All of the options you may need for configuring your cluster are documented on the supported types page.

```
$ kubectl get pod
NAME                                  READY     STATUS    RESTARTS   AGE
es-demo-client-3995124321-5rc6g       1/1       Running   0          7m
es-demo-client-3995124321-9zrv9       1/1       Running   0          7m
es-demo-data-0                        1/1       Running   0          7m
es-demo-data-1                        1/1       Running   0          5m
es-demo-data-2                        1/1       Running   0          3m
es-demo-data-3                        1/1       Running   0          1m
es-demo-master-554549909-00162        1/1       Running   0          7m
es-demo-master-554549909-pp557        1/1       Running   0          7m
es-demo-master-554549909-vjgrt        1/1       Running   0          7m
```

1.4 4) Scale the data nodes

Scaling the nodes can be done by modifying your ElasticsearchCluster manifest. Currently this is only possible using `kubectl replace`, due to bugs with the way ThirdPartyResource's are handled in kubectl 1.5.

Edit your manifest and **increase** the number of replicas in the `data` node pool, then run:

```
$ kubectl replace -f examples/es-cluster-example.yaml
$ kubectl get pod
NAME                                  READY     STATUS    RESTARTS   AGE
es-demo-client-3995124321-5rc6g       1/1       Running   0          9m
es-demo-client-3995124321-9zrv9       1/1       Running   0          9m
es-demo-data-0                        1/1       Running   0          9m
es-demo-data-1                        1/1       Running   0          7m
es-demo-data-2                        1/1       Running   0          5m
es-demo-data-3                        1/1       Running   0          3m
es-demo-data-4                        0/1       Running   0          29s
es-demo-master-554549909-00162        1/1       Running   0          9m
es-demo-master-554549909-pp557        1/1       Running   0          9m
es-demo-master-554549909-vjgrt        1/1       Running   0          9m
```

You should see new data nodes being added into your cluster gradually. Once all are in the Running state, we can try a scale down. Do the same as before, but instead reduce the number of replicas in the `data` node pool. Then run a `kubectl replace` again:

```
$ kubectl replace -f examples/es-cluster-example.yaml
$ kubectl get pod
NAME                                  READY     STATUS        RESTARTS   AGE
es-demo-client-3995124321-5rc6g       1/1       Running       0          10m
es-demo-client-3995124321-9zrv9       1/1       Running       0          10m
es-demo-data-0                        1/1       Running       0          10m
es-demo-data-1                        1/1       Running       0          8m
es-demo-data-2                        1/1       Running       0          6m
es-demo-data-3                        1/1       Running       0          4m
```

(continues on next page)

(continued from previous page)

```
es-demo-data-4                      1/1       Terminating   0          2m
es-demo-master-554549909-00162      1/1       Running       0          10m
es-demo-master-554549909-pp557      1/1       Running       0          10m
es-demo-master-554549909-vjgrt      1/1       Running       0          10m
```

Upon scale-down, the Elasticsearch nodes will mark themselves as non-allocatable. This will trigger Elasticsearch to re-allocate any shards currently on the nodes being scaled down, meaning your data will be safely relocated within the cluster.

CHAPTER 2

Elasticsearch

2.1 Example cluster definition

Example `ElasticsearchCluster` resource:

```yaml
apiVersion: navigator.jetstack.io/v1alpha1
kind: ElasticsearchCluster
metadata:
  name: demo
spec:
  ## Omitting the minimumMasters fields will cause navigator to automatically
  ## determine a quorum of masters to use.
  # minimumMasters: 2
  version: 5.6.2

  securityContext:
    runAsUser: 1000

  pilotImage:
    repository: quay.io/jetstack/navigator-pilot-elasticsearch
    tag: v0.1.0-alpha.1
    pullPolicy: Always

  nodePools:
  - name: master
    replicas: 3

    roles:
    - master

    resources:
      requests:
        cpu: "500m"
        memory: "2Gi"
```

(continues on next page)

(continued from previous page)

```
      limits:
        cpu: "1"
        memory: "3Gi"

    persistence:
      enabled: true
      # size of the volume
      size: 10Gi
      # storageClass of the volume
      storageClass: standard

  - name: mixed
    replicas: 2

    roles:
    - data
    - ingest

    resources:
      requests:
        cpu: "500m"
        memory: "2Gi"
      limits:
        cpu: "1"
        memory: "3Gi"

    persistence:
      enabled: true
      # size of the volume
      size: 10Gi
      # storageClass of the volume
      storageClass: standard
```

2.2 Node Pools

The Elasticsearch nodes in a Navigator `ElasticsearchCluster` are configured and grouped by role and in Navigator, these groups of nodes are called `nodepools`.

Note: Other than the following whitelisted fields, updates to nodepool configuration are not allowed:

- `replicas`
- `persistence`

2.3 Configure Scheduler Type

If a custom scheduler type is required (for example if you are deploying with stork or another storage provider), this can be set on each nodepool:

```
spec:
  nodePools:
```

(continues on next page)

(continued from previous page)

```
  - name: "ringnodes-1"
    schedulerName: "fancy-scheduler"
  - name: "ringnodes-2"
    schedulerName: "fancy-scheduler"
```

If the nodepool field is not specified, the default scheduler is used.

2.4 Managing Compute Resources for Clusters

Each `nodepool` has a `resources` attribute which defines the resource requirements and limits for each database node (pod) in that pool.

In the example above, each database node will request 0.5 CPU core and 2GiB of memory, and will be limited to 1 CPU core and 3GiB of memory.

The `resources` field follows exactly the same specification as the Kubernetes Pod API (`pod.spec.containers[].resources`).

See Managing Compute Resources for Containers for more information.

2.5 Pilots

Navigator creates one `Pilot` resource for every database node. `Pilot` resources have the same name and name space as the `Pod` for the corresponding database node. The `Pilot.Spec` is read by the pilot process running inside a `Pod` and contains its desired configuration. The `Pilot.Status` is updated by the pilot process and contains the discovered state of a single database node.

2.6 Other Supplementary Resources

Navigator will also create a number of supplementary resources for each cluster. For example it will create a `serviceaccount`, a `role` and a `rolebinding` so that pilot pods in a cluster have read-only access the API resources containing cluster configuration, and so that pilot pods can update the status of their corresponding `Pilot` resource and leader election `configmap`.

2.7 System Configuration for Elasticsearch Nodes

Elasticsearch requires important system configuration settings to be applied globally on the host operating system.

You must either ensure that Navigator is running in a Kubernetes cluster where all the nodes have been configured this way. Or you could use node labels and node selectors to ensure that the pods of an Elasticsearch cluster are only scheduled to nodes with the required configuration.

See Using Sysctls in a Kubernetes Cluster, and Taints and Tolerations for more information.

One way to apply these settings is to deploy a `DaemonSet` that runs the configuration commands from within a privileged container on each Kubernetes node. Here's a simple example of such a `DaemonSet`:

```
$ kubectl apply -f docs/quick-start/sysctl-daemonset.yaml
```

```yaml
# Apply sysctl configuration required by Elasticsearch
#
# This DaemonSet will re-run sysctl every 60s on all nodes.
#
# XXX See CronJob daemonset which will allow scheduling one-shot or repeated
# jobs across nodes:
# https://github.com/kubernetes/kubernetes/issues/36601

apiVersion: "extensions/v1beta1"
kind: "DaemonSet"
metadata:
  name: "navigator-elasticsearch-sysctl"
  namespace: "kube-system"
spec:
  template:
    metadata:
      labels:
        app: "navigator-elasticsearch-sysctl"
    spec:
      containers:
      - name: "apply-sysctl"
        image: "busybox:latest"
        resources:
          limits:
            cpu: "10m"
            memory: "8Mi"
          requests:
            cpu: "10m"
            memory: "8Mi"
        securityContext:
          privileged: true
        command:
        - "/bin/sh"
        - "-c"
        - |
          set -o errexit
          set -o xtrace
          while sysctl -w vm.max_map_count=262144
          do
            sleep 60s
          done
```

docs/quick-start/sysctl-daemonset.yaml

CHAPTER 3

Cassandra

3.1 Example cluster definition

Example `CassandraCluster` resource:

```
apiVersion: "navigator.jetstack.io/v1alpha1"
kind: "CassandraCluster"
metadata:
  name: "demo"
spec:
  version: "3.11.1"
  nodePools:
  - name: "ringnodes"
    replicas: 3
    datacenter: "demo-datacenter"
    rack: "demo-rack"
    persistence:
      enabled: true
      size: "5Gi"
      storageClass: "default"
    nodeSelector: {}
    resources:
      requests:
        cpu: "500m"
        memory: "2Gi"
      limits:
        cpu: "1"
        memory: "3Gi"
  image:
    repository: "cassandra"
    tag: "3"
    pullPolicy: "IfNotPresent"
  pilotImage:
    repository: "quay.io/jetstack/navigator-pilot-cassandra"
```

(continues on next page)

(continued from previous page)

```
    tag: "v0.1.0-alpha.1"
```

3.2 Node Pools

The C* nodes in a Navigator `cassandracluster` are configured and grouped by rack and data center and in Navigator, these groups of nodes are called `nodepools`.

All the C* nodes (pods) in a `nodepool` have the same configuration and the following sections describe the configuration options that are available.

Note: Other than the following whitelisted fields, updates to nodepool configuration are not allowed:

- `replicas`
- `persistence`

3.3 Configure Scheduler Type

If a custom scheduler type is required (for example if you are deploying with stork or another storage provider), this can be set on each nodepool:

```
spec:
  nodePools:
  - name: "ringnodes-1"
    schedulerName: "fancy-scheduler"
  - name: "ringnodes-2"
    schedulerName: "fancy-scheduler"
```

If the nodepool field is not specified, the default scheduler is used.

3.4 Cassandra Across Multiple Availability Zones

3.4.1 With rack awareness

Navigator supports running Cassandra with rack and datacenter-aware replication To deploy this, you must run a `nodePool` in each availability zone, and mark each as a separate Cassandra rack.

The nodeSelector field of a nodePool allows scheduling the nodePool to a set of nodes matching labels. This should be used with a node label such as failure-domain.beta.kubernetes.io/zone.

The `datacenter` and `rack` fields mark all Cassandra nodes in a nodepool as being located in that datacenter and rack. This information can then be used with the NetworkTopologyStrategy keyspace replica placement strategy. If these are not specified, Navigator will select an appropriate name for each: `datacenter` defaults to a static value, and `rack` defaults to the nodePool's name.

As an example, the nodePool section of a CassandraCluster spec for deploying into GKE in europe-west1 with rack awareness enabled:

```yaml
nodePools:
- name: "np-europe-west1-b"
  replicas: 3
  datacenter: "europe-west1"
  rack: "europe-west1-b"
  nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "europe-west1-b"
  persistence:
    enabled: true
    size: "5Gi"
    storageClass: "default"
- name: "np-europe-west1-c"
  replicas: 3
  datacenter: "europe-west1"
  rack: "europe-west1-c"
  nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "europe-west1-c"
  persistence:
    enabled: true
    size: "5Gi"
    storageClass: "default"
- name: "np-europe-west1-d"
  replicas: 3
  datacenter: "europe-west1"
  rack: "europe-west1-d"
  nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "europe-west1-d"
  persistence:
    enabled: true
    size: "5Gi"
    storageClass: "default"
```

3.4.2 Without rack awareness

Since the default rack name is equal to the nodepool name, simply set the rack name to the same static value in each nodepool to disable rack awareness.

A simplified example:

```yaml
nodePools:
- name: "np-europe-west1-b"
  replicas: 3
  datacenter: "europe-west1"
  rack: "default-rack"
  nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "europe-west1-b"
- name: "np-europe-west1-c"
  replicas: 3
  datacenter: "europe-west1"
  rack: "default-rack"
  nodeSelector:
    failure-domain.beta.kubernetes.io/zone: "europe-west1-c"
- name: "np-europe-west1-d"
  replicas: 3
  datacenter: "europe-west1"
  rack: "default-rack"
```

(continues on next page)

(continued from previous page)

```
nodeSelector:
  failure-domain.beta.kubernetes.io/zone: "europe-west1-d"
```

3.5 Managing Compute Resources for Clusters

Each `nodepool` has a `resources` attribute which defines the resource requirements and limits for each database node (pod) in that pool.

In the example above, each database node will request 0.5 CPU core and 2GiB of memory, and will be limited to 1 CPU core and 3GiB of memory.

The `resources` field follows exactly the same specification as the Kubernetes Pod API (`pod.spec.containers[].resources`).

See Managing Compute Resources for Containers for more information.

3.6 Connecting to Cassandra

If you apply the YAML manifest from the example above, Navigator will create a Cassandra cluster with three C* nodes running in three pods. The IP addresses assigned to each C* node may change when pods are rescheduled or restarted, but there are stable DNS names which allow you to connect to the cluster.

3.6.1 Services and DNS Names

Navigator creates two headless services for every Cassandra cluster that it creates. Each service has a corresponding DNS domain name:

1. The *nodes* service (e.g. `cass-demo-nodes`) has a DNS domain name which resolves to the IP addresses of **all** the C* nodes in cluster (nodes 0, 1, and 2 in this example).
2. The *seeds* service (e.g. `cass-demo-seeds`) has a DNS domain name which resolves to the IP addresses of **only** the seed nodes (node 0 in this example).

These DNS names have multiple HOST (*A*) records, one for each **healthy** C* node IP address.

Note: The DNS server only includes healthy nodes when answering requests for these two services.

The DNS names can be resolved from any pod in the Kubernetes cluster:

- If the pod is in the same namespace as the Cassandra cluster you need only use the left most label of the DNS name. E.g. `cass-demo-nodes`.
- If the pod is in a different namespace you must use the fully qualified DNS name. E.g. `cass-demo-nodes.my-namespace.svc.cluster.local`.

Note: Read DNS for Services and Pods for more information about DNS in Kubernetes.

3.6.2 TCP Ports

The C* nodes all listen on the following TCP ports:

1. **9042**: For CQL client connections.
2. **8080**: For Prometheus client connections.

3.6.3 Connect using a CQL Client

Navigator configures all the nodes in a Cassandra cluster to listen on TCP port 9042 for CQL client connections. And there are CQL drivers for most popular programming languages. Most drivers have the ability to connect to a single node and then discover all the other cluster nodes.

For example, you could use the Datastax Python driver to connect to the Cassandra cluster as follows:

```
from cassandra.cluster import Cluster

cluster = Cluster(['cass-demo-nodes'], port=9042)
session = cluster.connect()
rows = session.execute('SELECT ... FROM ...')
for row in rows:
    print row
```

Note: The IP address to which the driver makes the initial connection depends on the DNS server and operating system configuration.

3.7 Pilots

Navigator creates one `Pilot` resource for every database node. `Pilot` resources have the same name and name space as the `Pod` for the corresponding database node. The `Pilot.Spec` is read by the pilot process running inside a `Pod` and contains its desired configuration. The `Pilot.Status` is updated by the pilot process and contains the discovered state of a single database node.

3.8 Other Supplementary Resources

Navigator will also create a number of supplementary resources for each cluster. For example it will create a `serviceaccount`, a `role` and a `rolebinding` so that pilot pods in a cluster have read-only access the API resources containing cluster configuration, and so that pilot pods can update the status of their corresponding `Pilot` resource and leader election `configmap`.

3.9 The Life Cycle of a Navigator Cassandra Cluster

Changes to the configuration of an established Cassandra cluster must be carefully sequenced in order to maintain the health of the cluster. So Navigator is conservative about the configuration changes that it supports.

Here are the configuration changes that are supported and the configuration changes which are not yet supported.

3.9.1 Supported Configuration Changes

Navigator supports the following changes to a Cassandra cluster:

- *Create Cluster*: Add all initially configured node pools and nodes.
- *Scale Out*: Increase `CassandraCluster.Spec.NodePools[0].Replicas` to add more C* nodes to a `nodepool`.

Navigator does not currently support any other changes to the Cassandra cluster configuration.

3.9.2 Unsupported Configuration Changes

The following configuration changes are not currently supported but will be supported in the near future:

- Minor Upgrade: Trigger a rolling Cassandra upgrade by increasing the minor and / or patch components of `CassandraCluster.Spec.Version`.
- Scale In: Decrease `CassandraCluster.Spec.NodePools[0].Replicas` to remove C* nodes from a `nodepool`.

The following configuration changes are not currently supported:

- Add Rack: Add a `nodepool` for a new rack.
- Remove Rack: Remove a `nodepool`.
- Add Data Center: Add a `nodepool` for a new data center.
- Remove Data Center: Remove all the `nodepools` in a data center.
- Major Upgrade: Upgrade to a new major Cassandra version.

3.9.3 Create Cluster

When you first create a `CassandraCluster` resource, Navigator will add nodes, one at a time, in order of `NodePool` and according to the process described in *Scale Out* (below). The order of node creation is determined by the order of the entries in the `CassandraCluster.Spec.NodePools` list. You can look at `CassandraCluster.Status.NodePools` to see the current state.

3.9.4 Scale Out

When you first create a cluster or when you increment the `CassandraCluster.Spec.NodePools[i].ReplicaCount`, Navigator will add C* nodes, one at a time, until the desired number of nodes is reached. You can look at `CassandraCluster.Status.NodePools[<nodepoolname>].ReadyReplicas` to see the current number of healthy C* nodes in each `nodepool`.

CHAPTER 4

Development guide

4.1 Setting up

Install minikube and start a cluster with RBAC enabled:

```
minikube start --extra-config=apiserver.Authorization.Mode=RBAC
```

Work around `kube-dns` and helm having problems when RBAC is enabled in minikube:

```
kubectl create clusterrolebinding cluster-admin:kube-system \
    --clusterrole=cluster-admin \
    --serviceaccount=kube-system:default
```

Fetch the docker configuration:

```
eval $(minikube docker-env)
```

Build images in minikube's docker:

```
make BUILD_TAG=dev all
```

Or quicker (skips tests):

```
make BUILD_TAG=dev build docker_build
```

Install helm into the minikube cluster:

```
helm init
```

Install navigator using the helm chart:

```
helm install contrib/charts/navigator \
    --set apiserver.image.pullPolicy=Never \
    --set apiserver.image.tag=dev \
```

(continues on next page)

(continued from previous page)

```
    --set controller.image.pullPolicy=Never \
    --set controller.image.tag=dev \
    --name navigator --namespace navigator --wait
```

Now test navigator is deployed by creating a demo elasticsearch cluster. Edit `docs/quick-start/es-cluster-demo.yaml` to change the pilot image tag to `dev`, and set the `pullPolicy` to `Never`, then create the cluster:

```
kubectl create -f docs/quick-start/es-cluster-demo.yaml
```

4.2 Developing

Edit code, then build:

```
make BUILD_TAG=dev build docker_build
```

Or only for the component you're interested in:

```
make BUILD_TAG=dev controller docker_build_controller
```

Kill the component you're working on, for example the controller:

```
kubectl delete pods -n navigator -l app=navigator -l component=controller
```

CHAPTER 5

Indices and tables

- genindex
- search