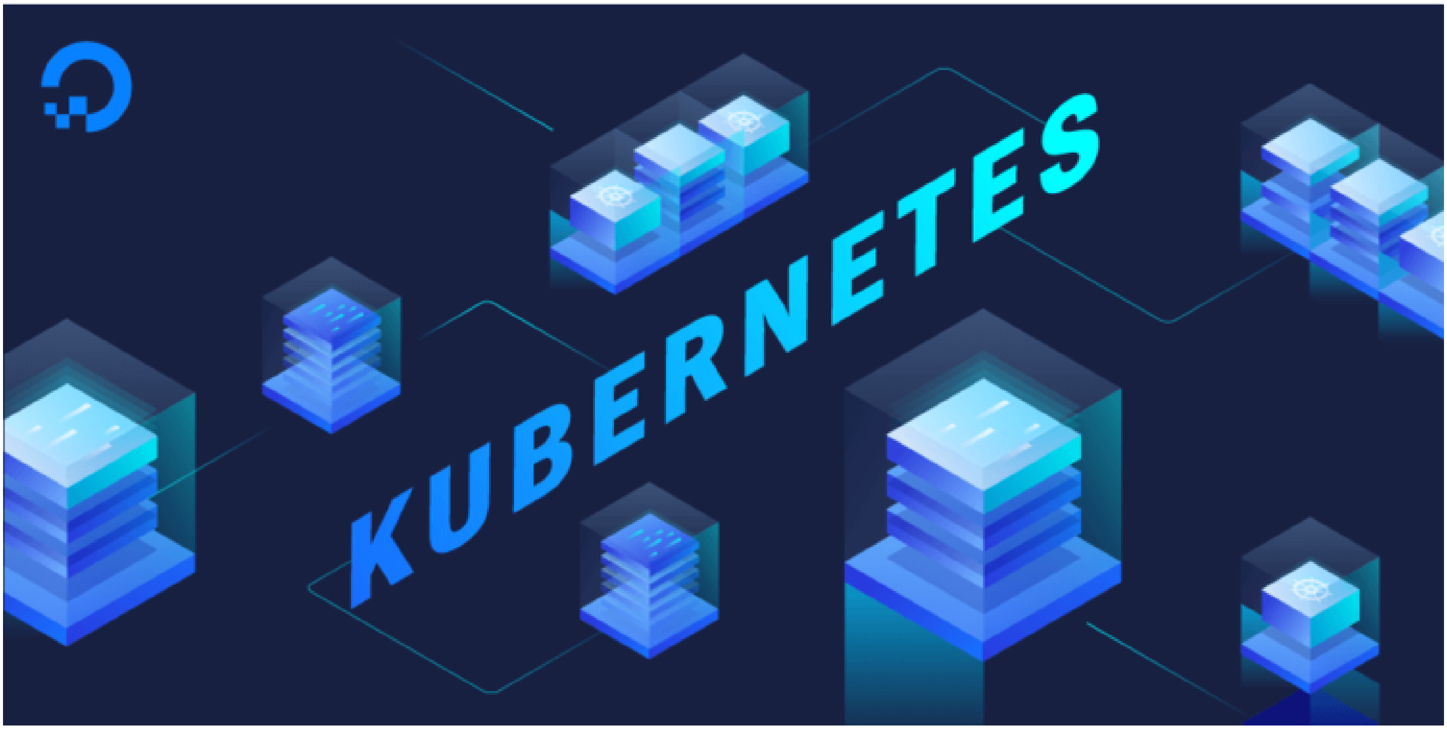


TUTORIAL

How To Set Up an Elasticsearch, Fluentd and Kibana (EFK) Logging Stack on Kubernetes

Logging Elasticsearch Kubernetes Solutions

By Hanif Jetha

Last Validated on March 30, 2020 • Originally Published on November 26, 2018 239.1k

English

Introduction

When running multiple services and applications on a Kubernetes cluster, a centralized, cluster-level logging stack can help you quickly sort through and analyze the heavy volume of log data produced by your Pods. One popular centralized logging solution is the **E**lasticsearch, **F**luentd, and **K**ibana (EFK) stack.

Elasticsearch is a real-time, distribu SCROLL TO TOP search engine which allows for full-text and structured search, as well as analytics. It is commonly used to index and search through large volumes of log data, but can also be used to search many different kinds of documents.

Elasticsearch is commonly deployed alongside **Kibana**, a powerful data visualization frontend and dashboard for Elasticsearch. Kibana allows you to explore your Elasticsearch log data through a web interface, and build dashboards and queries to quickly answer questions and gain insight into your Kubernetes applications.

In this tutorial we'll use **Fluentd** to collect, transform, and ship log data to the Elasticsearch backend. Fluentd is a popular open-source data collector that we'll set up on our Kubernetes nodes to tail container log files, filter and transform the log data, and deliver it to the Elasticsearch cluster, where it will be indexed and stored.

We'll begin by configuring and launching a scalable Elasticsearch cluster, and then create the Kibana Kubernetes Service and Deployment. To conclude, we'll set up Fluentd as a DaemonSet so it runs on every Kubernetes worker node.

Prerequisites

Before you begin with this guide, ensure you have the following available to you:

- A Kubernetes 1.10+ cluster with role-based access control (RBAC) enabled
    - Ensure your cluster has enough resources available to roll out the EFK stack, and if not scale your cluster by adding worker nodes. We'll be deploying a 3-Pod Elasticsearch cluster (you can scale this down to 1 if necessary), as well as a single Kibana Pod. Every worker node will also run a Fluentd Pod. The cluster in this guide consists of 3 worker nodes and a managed control plane.
- The `kubectl` command-line tool installed on your local machine, configured to connect to your cluster. You can read more about installing `kubectl` in the official documentation.

Once you have these components set up, you're ready to begin with this guide.

Step 1 — Creating a Namespace

Before we roll out an Elasticsearch cluster, we'll first create a Namespace into which we'll install all of our logging instrumentation. Kubernetes lets you separate objects running in your cluster using a "virtual cluster" abstraction called Namespaces. In this guide, we'll create a `kube-logging` namespace into which we'll install the EFK stack components.

SCROLL TO TOP

This Namespace will also allow us to quickly clean up and remove the logging stack without any loss of function to the Kubernetes cluster.

To begin, first investigate the existing Namespaces in your cluster using `kubectl`:

```
$ kubectl get namespaces
```

You should see the following three initial Namespaces, which come preinstalled with your Kubernetes cluster:

```
Output
NAME          STATUS    AGE
default       Active    5m
kube-system   Active    5m
kube-public   Active    5m
```

The `default` Namespace houses objects that are created without specifying a Namespace. The `kube-system` Namespace contains objects created and used by the Kubernetes system, like `kube-dns`, `kube-proxy`, and `kubernetes-dashboard`. It's good practice to keep this Namespace clean and not pollute it with your application and instrumentation workloads.

The `kube-public` Namespace is another automatically created Namespace that can be used to store objects you'd like to be readable and accessible throughout the whole cluster, even to unauthenticated users.

To create the `kube-logging` Namespace, first open and edit a file called `kube-logging.yaml` using your favorite editor, such as nano:

```
$ nano kube-logging.yaml
```

Inside your editor, paste the following Namespace object YAML:

kube-logging.yaml

```
kind: Namespace
apiVersion: v1
metadata:
  name: kube-logging
```

SCROLL TO TOP

Then, save and close the file.

Here, we specify the Kubernetes object’s `kind` as a `Namespace` object. To learn more about `Namespace` objects, consult the Namespaces Walkthrough in the official Kubernetes documentation. We also specify the Kubernetes API version used to create the object (`v1`), and give it a `name`, `kube-logging`.

Once you’ve created the `kube-logging.yaml` Namespace object file, create the Namespace using `kubectl create` with the `-f` filename flag:

```
$ kubectl create -f kube-logging.yaml
```

You should see the following output:

```
Output
namespace/kube-logging created
```

You can then confirm that the Namespace was successfully created:

```
$ kubectl get namespaces
```

At this point, you should see the new `kube-logging` Namespace:

```
Output
NAME           STATUS    AGE
default        Active    23m
kube-logging   Active    1m
kube-public    Active    23m
kube-system    Active    23m
```

We can now deploy an Elasticsearch cluster into this isolated logging Namespace.

Step 2 — Creating the Elasticsearch StatefulSet

Now that we’ve created a Namespace to house our logging stack, we can begin rolling out its various components. We’ll first begin by deploying a 3-node Elasticsearch cluster.

In this guide, we use 3 Elasticsearch Pods to avoid the “split-brain” issue that occurs in highly-available, multi-node clusters. [illegible] split-brain” is what arises when one

SCROLL TO TOP

or more nodes can’t communicate with the others, and several “split” masters get elected. With 3 nodes, if one gets disconnected from the cluster temporarily, the other two nodes can elect a new master and the cluster can continue functioning while the last node attempts to rejoin. To learn more, consult A new era for cluster coordination in Elasticsearch and Voting configurations.

Creating the Headless Service

To start, we’ll create a headless Kubernetes service called `elasticsearch` that will define a DNS domain for the 3 Pods. A headless service does not perform load balancing or have a static IP; to learn more about headless services, consult the official Kubernetes documentation.

Open a file called `elasticsearch_svc.yaml` using your favorite editor:

```
$ nano elasticsearch_svc.yaml
```

Paste in the following Kubernetes service YAML:

elasticsearch_svc.yaml

```
kind: Service
apiVersion: v1
metadata:
  name: elasticsearch
  namespace: kube-logging
  labels:
    app: elasticsearch
spec:
  selector:
    app: elasticsearch
  clusterIP: None
  ports:
    - port: 9200
      name: rest
    - port: 9300
      name: inter-node
```

Then, save and close the file.

We define a `Service` called `elasticsearch` in the `kube-logging` Namespace, and give it the `app: elasticsearch` label. We then set the `.spec.selector` to `app: elasticsearch` so that the Se 's with the `app: elasticsearch` label. When we associate our Elastics et with this Service, the Service will

SCROLL TO TOP

return DNS A records that point to Elasticsearch Pods with the `app: elasticsearch` label.

We then set `clusterIP: None`, which renders the service headless. Finally, we define ports `9200` and `9300` which are used to interact with the REST API, and for inter-node communication, respectively.

Create the service using `kubectl`:

```
$ kubectl create -f elasticsearch_svc.yaml
```

You should see the following output:

```
Output
service/elasticsearch created
```

Finally, double-check that the service was successfully created using `kubectl get`:

```
kubectl get services --namespace=kube-logging
```

You should see the following:

```
Output
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)             AGE
elasticsearch   ClusterIP   None         <none>        9200/TCP,9300/TCP   26s
```

Now that we've set up our headless service and a stable `.elasticsearch.kube-logging.svc.cluster.local` domain for our Pods, we can go ahead and create the StatefulSet.

Creating the StatefulSet

A Kubernetes StatefulSet allows you to assign a stable identity to Pods and grant them stable, persistent storage. Elasticsearch requires stable storage to persist data across Pod rescheduling and restarts. To learn more about the StatefulSet workload, consult the Statefulsets page from the Kubernetes docs.

Open a file called `elasticsearch_s` SCROLL TO TOP in your favorite editor:

```
$ nano elasticsearch_statefulset.yaml
```

We will move through the StatefulSet object definition section by section, pasting blocks into this file.

Begin by pasting in the following block:

elasticsearch_statefulset.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: es-cluster
  namespace: kube-logging
spec:
  serviceName: elasticsearch
  replicas: 3
  selector:
    matchLabels:
      app: elasticsearch
  template:
    metadata:
      labels:
        app: elasticsearch
```

In this block, we define a StatefulSet called `es-cluster` in the `kube-logging` namespace. We then associate it with our previously created `elasticsearch` Service using the `serviceName` field. This ensures that each Pod in the StatefulSet will be accessible using the following DNS address: `es-cluster-[0,1,2].elasticsearch.kube-logging.svc.cluster.local`, where `[0,1,2]` corresponds to the Pod's assigned integer ordinal.

We specify 3 `replicas` (Pods) and set the `matchLabels` selector to `app: elasticseach`, which we then mirror in the `.spec.template.metadata` section. The `.spec.selector.matchLabels` and `.spec.template.metadata.labels` fields must match.

We can now move on to the object spec. Paste in the following block of YAML immediately below the preceding block:

elasticsearch_statefulset.yaml

SCROLL TO TOP

```
. . .
    spec:
      containers:
      - name: elasticsearch
        image: docker.elastic.co/elasticsearch/elasticsearch:7.2.0
        resources:
            limits:
              cpu: 1000m
            requests:
              cpu: 100m
        ports:
        - containerPort: 9200
          name: rest
          protocol: TCP
        - containerPort: 9300
          name: inter-node
          protocol: TCP
        volumeMounts:
        - name: data
          mountPath: /usr/share/elasticsearch/data
        env:
          - name: cluster.name
            value: k8s-logs
          - name: node.name
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: discovery.seed_hosts
            value: "es-cluster-0.elasticsearch,es-cluster-1.elasticsearch,es-clus
          - name: cluster.initial_master_nodes
            value: "es-cluster-0,es-cluster-1,es-cluster-2"
          - name: ES_JAVA_OPTS
            value: "-Xms512m -Xmx512m"
```

Here we define the Pods in the StatefulSet. We name the containers `elasticsearch` and choose the `docker.elastic.co/elasticsearch/elasticsearch:7.2.0` Docker image. At this point, you may modify this image tag to correspond to your own internal Elasticsearch image, or a different version. Note that for the purposes of this guide, only Elasticsearch `7.2.0` has been tested.

We then use the `resources` field to specify that the container needs at least 0.1 vCPU guaranteed to it, and can burst up to 1 vCPU (which limits the Pod's resource usage when performing an initial large ingest or dealing with a load spike). You should modify these values depending on your anticipated load and available resources. To learn more about resource requests and limits, consult the official Kubernetes Documentation.

We then open and name ports `9200` and `9300` for REST API and inter-node communication, respectively. We specify a `volumeMount` called `data` that will mount the PersistentVolume named `data` to th path

SCROLL TO TOP

`/usr/share/elasticsearch/data`. We will define the VolumeClaims for this StatefulSet in a later YAML block.

Finally, we set some environment variables in the container:

- `cluster.name`: The Elasticsearch cluster's name, which in this guide is `k8s-logs`.
- `node.name`: The node's name, which we set to the `.metadata.name` field using `valueFrom`. This will resolve to `es-cluster-[0,1,2]`, depending on the node's assigned ordinal.
- `discovery.seed_hosts`: This field sets a list of master-eligible nodes in the cluster that will seed the node discovery process. In this guide, thanks to the headless service we configured earlier, our Pods have domains of the form `es-cluster-[0,1,2].elasticsearch.kube-logging.svc.cluster.local`, so we set this variable accordingly. Using local namespace Kubernetes DNS resolution, we can shorten this to `es-cluster-[0,1,2].elasticsearch`. To learn more about Elasticsearch discovery, consult the official Elasticsearch documentation.
- `cluster.initial_master_nodes`: This field also specifies a list of master-eligible nodes that will participate in the master election process. Note that for this field you should identify nodes by their `node.name`, and not their hostnames.
- `ES_JAVA_OPTS`: Here we set this to `-Xms512m -Xmx512m` which tells the JVM to use a minimum and maximum heap size of 512 MB. You should tune these parameters depending on your cluster's resource availability and needs. To learn more, consult Setting the heap size.

The next block we'll paste in looks as follows:

elasticsearch_statefulset.yaml

```
. . .
      initContainers:
      - name: fix-permissions
        image: busybox
        command: ["sh", "-c", "chown -R 1000:1000 /usr/share/elasticsearch/data"]
        securityContext:
          privileged: true
        volumeMounts:
        - name: data
          mountPath: /usr/share/elasticsearch/data
      - name: increase-vm-max-map
        image: busybox
        command: ["sysctl", "-w", "vm.max_map_count=262144"]
        securityContext:
          privileged: true
      - name: increase-fd-ulimit
```

SCROLL TO TOP

```
        image: busybox
        command: ["sh", "-c", "ulimit -n 65536"]
        securityContext:
          privileged: true
```

In this block, we define several Init Containers that run before the main `elasticsearch` app container. These Init Containers each run to completion in the order they are defined. To learn more about Init Containers, consult the official Kubernetes Documentation.

The first, named `fix-permissions`, runs a `chown` command to change the owner and group of the Elasticsearch data directory to `1000:1000`, the Elasticsearch user's UID. By default Kubernetes mounts the data directory as `root`, which renders it inaccessible to Elasticsearch. To learn more about this step, consult Elasticsearch's "Notes for production use and defaults."

The second, named `increase-vm-max-map`, runs a command to increase the operating system's limits on mmap counts, which by default may be too low, resulting in out of memory errors. To learn more about this step, consult the official Elasticsearch documentation.

The next Init Container to run is `increase-fd-ulimit`, which runs the `ulimit` command to increase the maximum number of open file descriptors. To learn more about this step, consult the "Notes for Production Use and Defaults" from the official Elasticsearch documentation.

> **Note:** The Elasticsearch Notes for Production Use also mentions disabling swapping for performance reasons. Depending on your Kubernetes installation or provider, swapping may already be disabled. To check this, `exec` into a running container and run `cat /proc/swaps` to list active swap devices. If you see nothing there, swap is disabled.

Now that we've defined our main app container and the Init Containers that run before it to tune the container OS, we can add the final piece to our StatefulSet object definition file: the `volumeClaimTemplates`.

Paste in the following `volumeClaimTemplate` block:

elasticsearch_statefulset.yaml

```
. . .
  volumeClaimTemplates:
  - metadata:
```

SCROLL TO TOP

```
      name: data
      labels:
        app: elasticsearch
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: do-block-storage
      resources:
        requests:
          storage: 100Gi
```

In this block, we define the StatefulSet's `volumeClaimTemplates`. Kubernetes will use this to create PersistentVolumes for the Pods. In the block above, we name it `data` (which is the `name` we refer to in the `volumeMount`s defined previously), and give it the same `app: elasticsearch` label as our StatefulSet.

We then specify its access mode as `ReadWriteOnce`, which means that it can only be mounted as read-write by a single node. We define the storage class as `do-block-storage` in this guide since we use a DigitalOcean Kubernetes cluster for demonstration purposes. You should change this value depending on where you are running your Kubernetes cluster. To learn more, consult the Persistent Volume documentation.

Finally, we specify that we'd like each PersistentVolume to be 100GiB in size. You should adjust this value depending on your production needs.

The complete StatefulSet spec should look something like this:

elasticsearch_statefulset.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: es-cluster
  namespace: kube-logging
spec:
  serviceName: elasticsearch
  replicas: 3
  selector:
    matchLabels:
      app: elasticsearch
  template:
    metadata:
      labels:
        app: elasticsearch
    spec:
      containers:
      - name: elasticsearch
```

SCROLL TO TOP

```
        image: docker.elastic.co/elasticsearch/elasticsearch:7.2.0
        resources:
            limits:
              cpu: 1000m
            requests:
              cpu: 100m
        ports:
        - containerPort: 9200
          name: rest
          protocol: TCP
        - containerPort: 9300
          name: inter-node
          protocol: TCP
        volumeMounts:
        - name: data
          mountPath: /usr/share/elasticsearch/data
        env:
          - name: cluster.name
            value: k8s-logs
          - name: node.name
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: discovery.seed_hosts
            value: "es-cluster-0.elasticsearch,es-cluster-1.elasticsearch,es-clus
          - name: cluster.initial_master_nodes
            value: "es-cluster-0,es-cluster-1,es-cluster-2"
          - name: ES_JAVA_OPTS
            value: "-Xms512m -Xmx512m"
      initContainers:
      - name: fix-permissions
        image: busybox
        command: ["sh", "-c", "chown -R 1000:1000 /usr/share/elasticsearch/data"]
        securityContext:
          privileged: true
        volumeMounts:
        - name: data
          mountPath: /usr/share/elasticsearch/data
      - name: increase-vm-max-map
        image: busybox
        command: ["sysctl", "-w", "vm.max_map_count=262144"]
        securityContext:
          privileged: true
      - name: increase-fd-ulimit
        image: busybox
        command: ["sh", "-c", "ulimit -n 65536"]
        securityContext:
          privileged: true
  volumeClaimTemplates:
  - metadata:
      name: data
      labels:
        app: elasticsearch
    spec:
      accessModes: [ "ReadWriteOnce" ]
```

SCROLL TO TOP

```
        storageClassName: do-block-storage
        resources:
          requests:
            storage: 100Gi
```

Once you’re satisfied with your Elasticsearch configuration, save and close the file.

Now, deploy the StatefulSet using `kubectl`:

```
$ kubectl create -f elasticsearch_statefulset.yaml
```

You should see the following output:

```
Output
statefulset.apps/es-cluster created
```

You can monitor the StatefulSet as it is rolled out using `kubectl rollout status`:

```
$ kubectl rollout status sts/es-cluster --namespace=kube-logging
```

You should see the following output as the cluster is rolled out:

```
Output
Waiting for 3 pods to be ready...
Waiting for 2 pods to be ready...
Waiting for 1 pods to be ready...
partitioned roll out complete: 3 new pods have been updated...
```

Once all the Pods have been deployed, you can check that your Elasticsearch cluster is functioning correctly by performing a request against the REST API.

To do so, first forward the local port `9200` to the port `9200` on one of the Elasticsearch nodes (`es-cluster-0`) using `kubectl port-forward`:

```
$ kubectl port-forward es-cluster-0 9200:9200 --namespace=kube-logging
```

Then, in a separate terminal window, perform a `curl` request against the REST API:

SCROLL TO TOP

```
$ curl http://localhost:9200/_cluster/state?pretty
```

You shoulds see the following output:

Output

```
{
  "cluster_name" : "k8s-logs",
  "compressed_size_in_bytes" : 348,
  "cluster_uuid" : "QD06dK7CQgids-GQZooNVw",
  "version" : 3,
  "state_uuid" : "mjNIWXAzQVuxNNOQ7xR-qg",
  "master_node" : "IdM5B7cUQWqFgIHXBp0JDg",
  "blocks" : { },
  "nodes" : {
    "u7DoTpMmSCixOoictzHItA" : {
      "name" : "es-cluster-1",
      "ephemeral_id" : "ZlBflnXKRMC4RvEACHIVdg",
      "transport_address" : "10.244.8.2:9300",
      "attributes" : { }
    },
    "IdM5B7cUQWqFgIHXBp0JDg" : {
      "name" : "es-cluster-0",
      "ephemeral_id" : "JTk1FDdFQuWbSFAtBxdxAQ",
      "transport_address" : "10.244.44.3:9300",
      "attributes" : { }
    },
    "R8E7xcSUSbGbgrhAdyAKmQ" : {
      "name" : "es-cluster-2",
      "ephemeral_id" : "9wv6ke71Qqy9vk2LgJTqaA",
      "transport_address" : "10.244.40.4:9300",
      "attributes" : { }
    }
  },
...
```

This indicates that our Elasticsearch cluster `k8s-logs` has successfully been created with 3 nodes: `es-cluster-0`, `es-cluster-1`, and `es-cluster-2`. The current master node is `es-cluster-0`.

Now that your Elasticsearch cluster is up and running, you can move on to setting up a Kibana frontend for it.

Step 3 — Creating the Kibana SCROLL TO TOP ıd Service

To launch Kibana on Kubernetes, we'll create a Service called `kibana`, and a Deployment consisting of one Pod replica. You can scale the number of replicas depending on your production needs, and optionally specify a `LoadBalancer` type for the Service to load balance requests across the Deployment pods.

This time, we'll create the Service and Deployment in the same file. Open up a file called `kibana.yaml` in your favorite editor:

```
$ nano kibana.yaml
```

Paste in the following service spec:

kibana.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: kibana
  namespace: kube-logging
  labels:
    app: kibana
spec:
  ports:
  - port: 5601
  selector:
    app: kibana
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kibana
  namespace: kube-logging
  labels:
    app: kibana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kibana
  template:
    metadata:
      labels:
        app: kibana
    spec:
      containers:
      - name: kibana
        image: docker.elastic.co/kibana/kibana:7.2.0
        resources:
          limits:
            cpu: 1000m
          requests:
            cpu: 100m
        env:
          - name: ELASTICSEARCH_URL
            value: http://elasticsearch:9200
        ports:
        - containerPort: 5601
```

Then, save and close the file.

In this spec we've defined a service called `kibana` in the `kube-logging` namespace, and gave it the `app: kibana` label.

We've also specified that it should be accessible on port `5601` and use the `app: kibana` label to select the Service's target Pods.

In the `Deployment` spec, we define a Deployment called `kibana` and specify that we'd like 1 Pod replica.

We use the `docker.elastic.co/kibana/kibana:7.2.0` image. At this point you may substitute your own private or public Kibana image to use.

We specify that we'd like at the very least 0.1 vCPU guaranteed to the Pod, bursting up to a limit of 1 vCPU. You may change these parameters depending on your anticipated load and available resources.

Next, we use the `ELASTICSEARCH_URL` environment variable to set the endpoint and port for the Elasticsearch cluster. Using Kubernetes DNS, this endpoint corresponds to its Service name `elasticsearch`. This domain will resolve to a list of IP addresses for the 3 Elasticsearch Pods. To learn more about Kubernetes DNS, consult DNS for Services and Pods.

Finally, we set Kibana's container port to `5601`, to which the `kibana` Service will forward requests.

Once you're satisfied with your Kibana configuration, you can roll out the Service and Deployment using `kubectl`:

```
$ kubectl create -f kibana.yaml
```

SCROLL TO TOP

You should see the following output:

```
Output
service/kibana created
deployment.apps/kibana created
```

You can check that the rollout succeeded by running the following command:

```
$ kubectl rollout status deployment/kibana --namespace=kube-logging
```

You should see the following output:

```
Output
deployment "kibana" successfully rolled out
```

To access the Kibana interface, we’ll once again forward a local port to the Kubernetes node running Kibana. Grab the Kibana Pod details using `kubectl get`:

```
$ kubectl get pods --namespace=kube-logging
```

```
Output
NAME                        READY     STATUS    RESTARTS   AGE
es-cluster-0                1/1       Running   0          55m
es-cluster-1                1/1       Running   0          54m
es-cluster-2                1/1       Running   0          54m
kibana-6c9fb4b5b7-plbg2     1/1       Running   0          4m27s
```

Here we observe that our Kibana Pod is called `kibana-6c9fb4b5b7-plbg2`.

Forward the local port `5601` to port `5601` on this Pod:

```
$ kubectl port-forward kibana-6c9fb4b5b7-plbg2 5601:5601 --namespace=kube-loggi
```

You should see the following output:

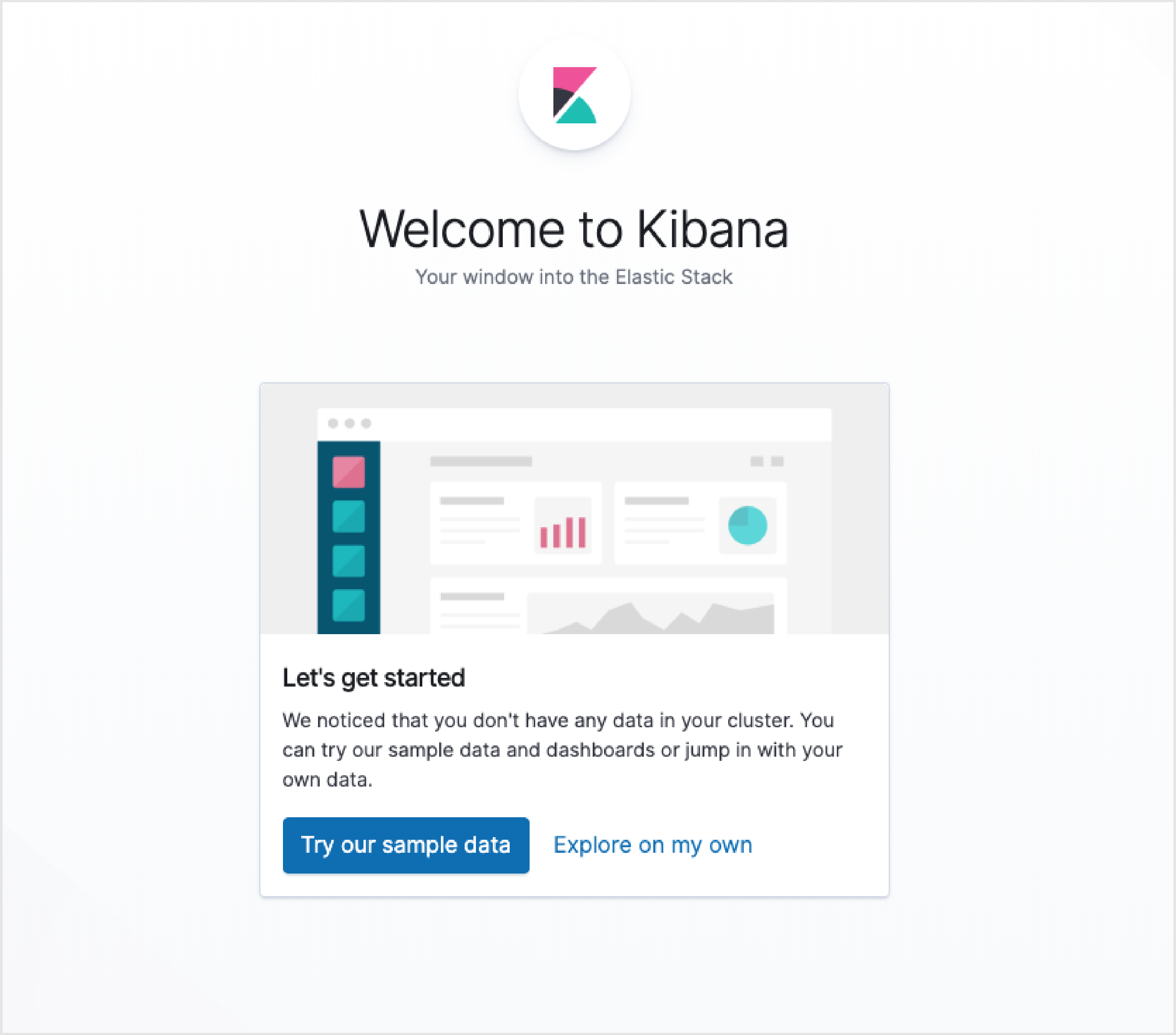
```
Output
Forwarding from 127.0.0.1:5601 -> 5601
Forwarding from [::1]:5601 -> 560
```

SCROLL TO TOP

Now, in your web browser, visit the following URL:

```
http://localhost:5601
```

If you see the following Kibana welcome page, you've successfully deployed Kibana into your Kubernetes cluster:



Welcome to Kibana

Your window into the Elastic Stack

Let's get started

We noticed that you don't have any data in your cluster. You can try our sample data and dashboards or jump in with your own data.

You can now move on to rolling out the final component of the EFK stack: the log collector, Fluentd.

Step 4 — Creating the Fluentd DaemonSet

In this guide, we'll set up Fluentd as a DaemonSet, which is a Kubernetes workload type that runs a copy of a given Pod on each Node in the Kubernetes cluster. Using this DaemonSet controller, we'll roll out a ‾ SCROLL TO TOP ‾agent Pod on every node in our cluster. To learn more about this logging architecture, consult “Using a node logging agent” from the official Kubernetes docs.

In Kubernetes, containerized applications that log to `stdout` and `stderr` have their log streams captured and redirected to JSON files on the nodes. The Fluentd Pod will tail these log files, filter log events, transform the log data, and ship it off to the Elasticsearch logging backend we deployed in Step 2.

In addition to container logs, the Fluentd agent will tail Kubernetes system component logs like kubelet, kube-proxy, and Docker logs. To see a full list of sources tailed by the Fluentd logging agent, consult the `kubernetes.conf` file used to configure the logging agent. To learn more about logging in Kubernetes clusters, consult “Logging at the node level” from the official Kubernetes documentation.

Begin by opening a file called `fluentd.yaml` in your favorite text editor:

```
$ nano fluentd.yaml
```

Once again, we’ll paste in the Kubernetes object definitions block by block, providing context as we go along. In this guide, we use the Fluentd DaemonSet spec provided by the Fluentd maintainers. Another helpful resource provided by the Fluentd maintainers is Kuberentes Fluentd.

First, paste in the following ServiceAccount definition:

fluentd.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluentd
  namespace: kube-logging
  labels:
    app: fluentd
```

Here, we create a Service Account called `fluentd` that the Fluentd Pods will use to access the Kubernetes API. We create it in the `kube-logging` Namespace and once again give it the label `app: fluentd`. To learn more about Service Accounts in Kubernetes, consult Configure Service Accounts for Pods in the official Kubernetes docs.

Next, paste in the following `Cluster`

SCROLL TO TOP

fluentd.yaml

```
. . .
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: fluentd
  labels:
    app: fluentd
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - namespaces
  verbs:
  - get
  - list
  - watch
```

Here we define a ClusterRole called `fluentd` to which we grant the `get`, `list`, and `watch` permissions on the `pods` and `namespaces` objects. ClusterRoles allow you to grant access to cluster-scoped Kubernetes resources like Nodes. To learn more about Role-Based Access Control and Cluster Roles, consult Using RBAC Authorization from the official Kubernetes documentation.

Now, paste in the following `ClusterRoleBinding` block:

fluentd.yaml

```
. . .
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: fluentd
roleRef:
  kind: ClusterRole
  name: fluentd
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: fluentd
  namespace: kube-logging
```

In this block, we define a `ClusterRoleBinding` called `fluentd` which binds the `fluentd` ClusterRole to the `fluentd` Service Account. This grants the `fluentd`

SCROLL TO TOP

ServiceAccount the permissions listed in the `fluentd` Cluster Role.

At this point we can begin pasting in the actual DaemonSet spec:

fluentd.yaml

```
. . .
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
  namespace: kube-logging
  labels:
    app: fluentd
```

Here, we define a DaemonSet called `fluentd` in the `kube-logging` Namespace and give it the `app: fluentd` label.

Next, paste in the following section:

fluentd.yaml

```
. . .
spec:
  selector:
    matchLabels:
      app: fluentd
  template:
    metadata:
      labels:
        app: fluentd
    spec:
      serviceAccount: fluentd
      serviceAccountName: fluentd
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd
        image: fluent/fluentd-kubernetes-daemonset:v1.4.2-debian-elasticsearch-1.
        env:
          - name:  FLUENT_ELASTICSEARCH_HOST
            value: "elasticsearch.kube-logging.svc.cluster.local"
          - name:  FLUENT_ELASTICSEARCH_PORT
            value: "9200"
          - name: FLUENT_ELASTICSEARCH_SCHEME
            value: "http"
          - name: FLUENTD_SYSTE
            value: disable
```

SCROLL TO TOP

Here, we match the `app: fluentd` label defined in `.metadata.labels` and then assign the DaemonSet the `fluentd` Service Account. We also select the `app: fluentd` as the Pods managed by this DaemonSet.

Next, we define a `NoSchedule` toleration to match the equivalent taint on Kubernetes master nodes. This will ensure that the DaemonSet also gets rolled out to the Kubernetes masters. If you don't want to run a Fluentd Pod on your master nodes, remove this toleration. To learn more about Kubernetes taints and tolerations, consult "Taints and Tolerations" from the official Kubernetes docs.

Next, we begin defining the Pod container, which we call `fluentd`.

We use the official v1.4.2 Debian image provided by the Fluentd maintainers. If you'd like to use your own private or public Fluentd image, or use a different image version, modify the `image` tag in the container spec. The Dockerfile and contents of this image are available in Fluentd's fluentd-kubernetes-daemonset Github repo.

Next, we configure Fluentd using some environment variables:

- `FLUENT_ELASTICSEARCH_HOST`: We set this to the Elasticsearch headless Service address defined earlier: `elasticsearch.kube-logging.svc.cluster.local`. This will resolve to a list of IP addresses for the 3 Elasticsearch Pods. The actual Elasticsearch host will most likely be the first IP address returned in this list. To distribute logs across the cluster, you will need to modify the configuration for Fluentd's Elasticsearch Output plugin. To learn more about this plugin, consult Elasticsearch Output Plugin.
- `FLUENT_ELASTICSEARCH_PORT`: We set this to the Elasticsearch port we configured earlier, `9200`.
- `FLUENT_ELASTICSEARCH_SCHEME`: We set this to `http`.
- `FLUENTD_SYSTEMD_CONF`: We set this to `disable` to suppress output related to `systemd` not being set up in the container.

Finally, paste in the following section:

fluentd.yaml

```
. . .
        resources:
          limits:
            memory: 512Mi
          requests:
```

SCROLL TO TOP

```
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Here we specify a 512 MiB memory limit on the FluentD Pod, and guarantee it 0.1vCPU and 200MiB of memory. You can tune these resource limits and requests depending on your anticipated log volume and available resources.

Next, we mount the `/var/log` and `/var/lib/docker/containers` host paths into the container using the `varlog` and `varlibdockercontainers` `volumeMounts`. These `volumes` are defined at the end of the block.

The final parameter we define in this block is `terminationGracePeriodSeconds`, which gives Fluentd 30 seconds to shut down gracefully upon receiving a `SIGTERM` signal. After 30 seconds, the containers are sent a `SIGKILL` signal. The default value for `terminationGracePeriodSeconds` is 30s, so in most cases this parameter can be omitted. To learn more about gracefully terminating Kubernetes workloads, consult Google's "Kubernetes best practices: terminating with grace."

The entire Fluentd spec should look something like this:

fluentd.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluentd
  namespace: kube-logging
  labels:
    app: fluentd
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

SCROLL TO TOP

```
  name: fluentd
  labels:
    app: fluentd
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - namespaces
  verbs:
  - get
  - list
  - watch
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: fluentd
roleRef:
  kind: ClusterRole
  name: fluentd
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: fluentd
  namespace: kube-logging
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd
  namespace: kube-logging
  labels:
    app: fluentd
spec:
  selector:
    matchLabels:
      app: fluentd
  template:
    metadata:
      labels:
        app: fluentd
    spec:
      serviceAccount: fluentd
      serviceAccountName: fluentd
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd
        image: fluent/fluentd-kubernetes-daemonset:v1.4.2-debian-elasticsearch-1.
        env:
          - name:  FLUENT_ELASTI
            value: "elasticsear  SCROLL TO TOP  vc.cluster.local"
          - name:  FLUENT_ELASTICSEARCH_PORT
```

```
            value: "9200"
          - name: FLUENT_ELASTICSEARCH_SCHEME
            value: "http"
          - name: FLUENTD_SYSTEMD_CONF
            value: disable
        resources:
          limits:
            memory: 512Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Once you’ve finished configuring the Fluentd DaemonSet, save and close the file.

Now, roll out the DaemonSet using `kubectl`:

```
$ kubectl create -f fluentd.yaml
```

You should see the following output:

```
Output
serviceaccount/fluentd created
clusterrole.rbac.authorization.k8s.io/fluentd created
clusterrolebinding.rbac.authorization.k8s.io/fluentd created
daemonset.extensions/fluentd created
```

Verify that your DaemonSet rolled out successfully using `kubectl`:

```
$ kubectl get ds --namespace=kube-logging
```

You should see the following status SCROLL TO TOP

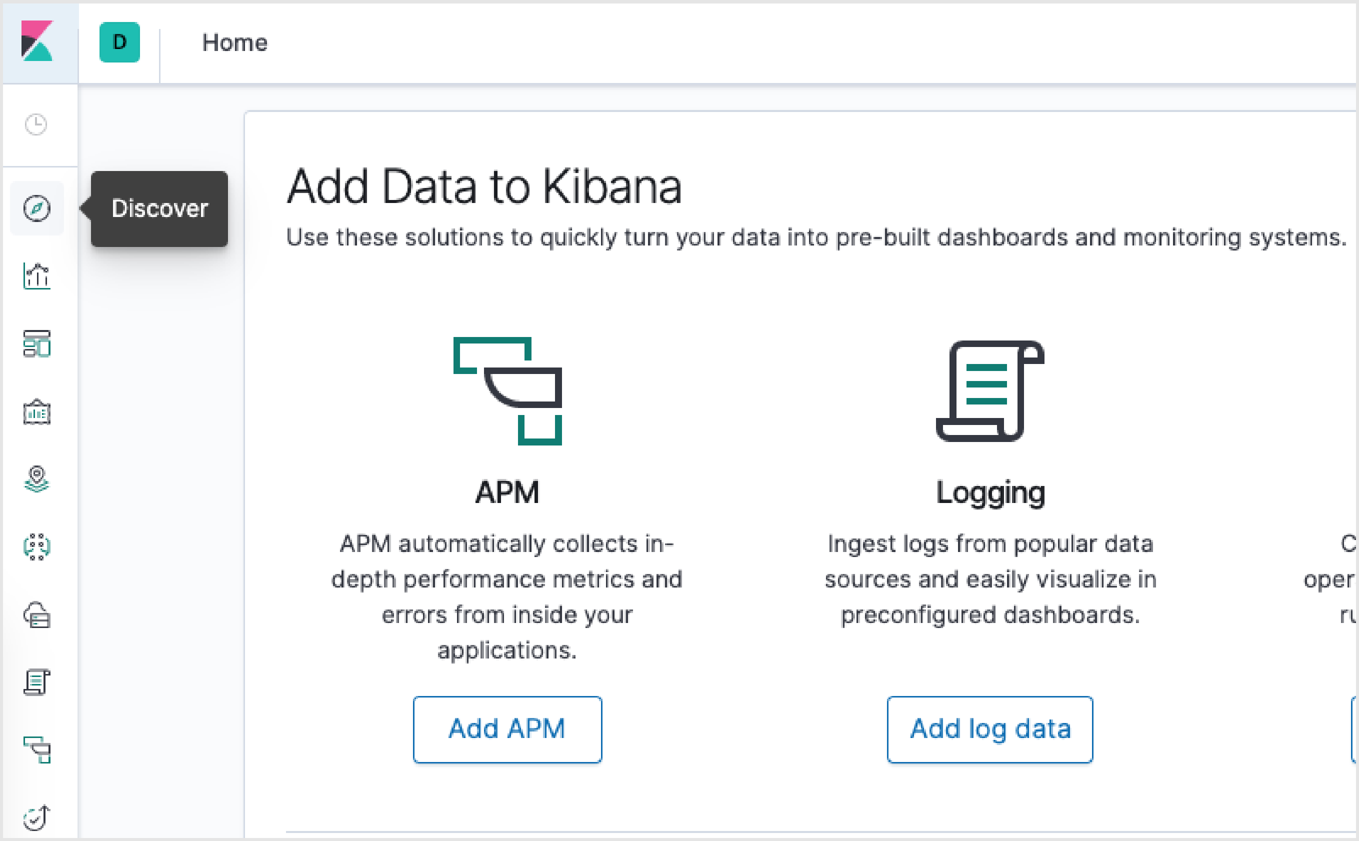
```
Output
NAME      DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGI
fluentd   3         3         3         3            3           <none>          58:
```

This indicates that there are 3 `fluentd` Pods running, which corresponds to the number of nodes in our Kubernetes cluster.

We can now check Kibana to verify that log data is being properly collected and shipped to Elasticsearch.

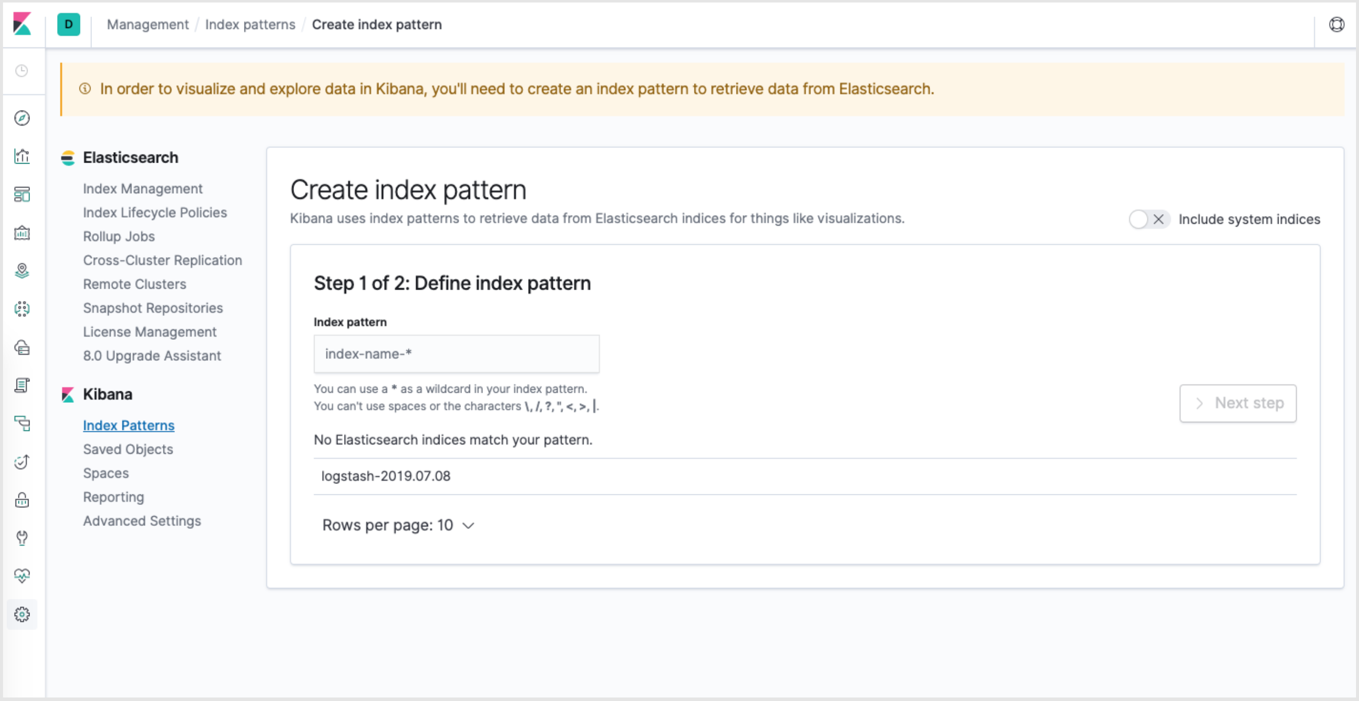
With the `kubectl port-forward` still open, navigate to `http://localhost:5601`.

Click on **Discover** in the left-hand navigation menu:


D
Home
Discover
Add Data to Kibana
Use these solutions to quickly turn your data into pre-built dashboards and monitoring systems.
APM
APM automatically collects in-depth performance metrics and errors from inside your applications.
Add APM
Logging
Ingest logs from popular data sources and easily visualize in preconfigured dashboards.
Add log data

You should see the following configuration window:

SCROLL TO TOP



Management / Index patterns / Create index pattern

In order to visualize and explore data in Kibana, you'll need to create an index pattern to retrieve data from Elasticsearch.

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

Step 1 of 2: Define index pattern

Index pattern: index-name-*

You can use a * as a wildcard in your index pattern.
You can't use spaces or the characters \, /, ?, ", <, >, |.

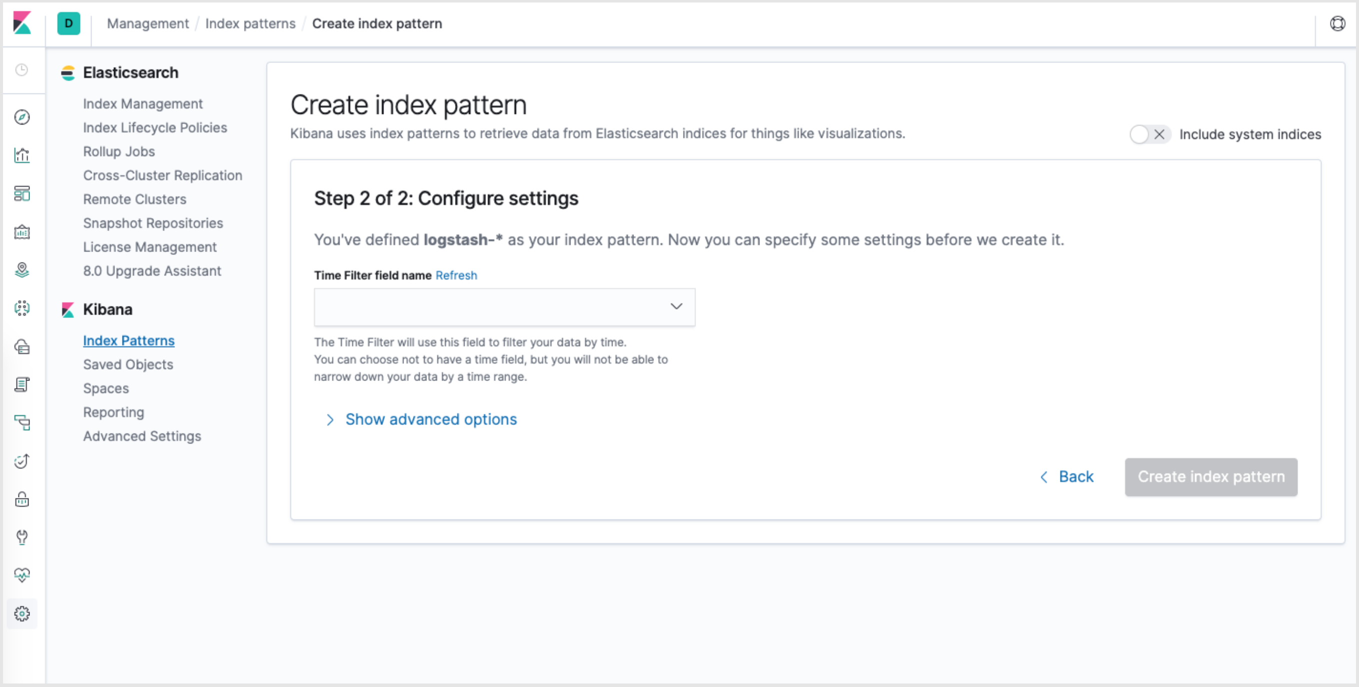
No Elasticsearch indices match your pattern.

logstash-2019.07.08

Rows per page: 10

This allows you to define the Elasticsearch indices you'd like to explore in Kibana. To learn more, consult Defining your index patterns in the official Kibana docs. For now, we'll just use the `logstash-*` wildcard pattern to capture all the log data in our Elasticsearch cluster. Enter `logstash-*` in the text box and click on **Next step**.

You'll then be brought to the following page:



Management / Index patterns / Create index pattern

Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

Step 2 of 2: Configure settings

You've defined **logstash-*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh

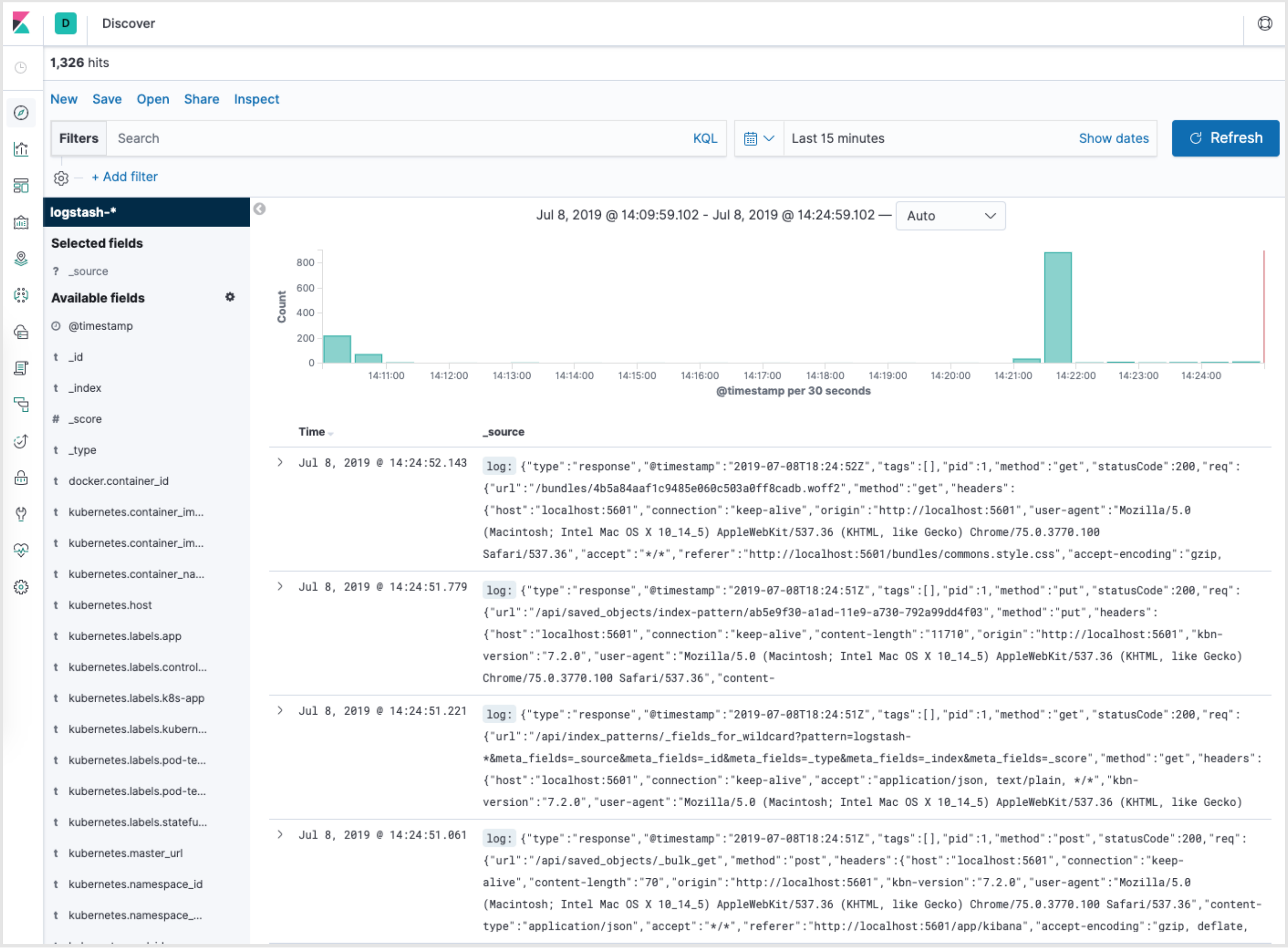
The Time Filter will use this field to filter your data by time.
You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

This allows you to configure which field Kibana will use to filter log data by time. In the dropdown, select the **@timestamp** field, and hit **Create index pattern**.

Now, hit **Discover** in the left hand na

SCROLL TO TOP

You should see a histogram graph and some recent log entries:


Discover
1,326 hits
New Save Open Share Inspect
Filters
Search
KQL
Last 15 minutes
Show dates
Refresh
+ Add filter
logstash-*
Selected fields
? _source
Available fields
@timestamp
t _id
t _index
# _score
t _type
t docker.container_id
t kubernetes.container_im...
t kubernetes.container_im...
t kubernetes.container_na...
t kubernetes.host
t kubernetes.labels.app
t kubernetes.labels.control...
t kubernetes.labels.k8s-app
t kubernetes.labels.kubern...
t kubernetes.labels.pod-te...
t kubernetes.labels.pod-te...
t kubernetes.labels.statefu...
t kubernetes.master_url
t kubernetes.namespace_id
t kubernetes.namespace_...
Jul 8, 2019 @ 14:09:59.102 - Jul 8, 2019 @ 14:24:59.102 —
Auto
Count
800
600
400
200
0
14:11:00
14:12:00
14:13:00
14:14:00
14:15:00
14:16:00
14:17:00
14:18:00
14:19:00
14:20:00
14:21:00
14:22:00
14:23:00
14:24:00
@timestamp per 30 seconds
Time
_source

```
Jul 8, 2019 @ 14:24:52.143  log: {"type":"response","@timestamp":"2019-07-08T18:24:52Z","tags":[],"pid":1,"method":"get","statusCode":200,"req":
                            {"url":"/bundles/4b5a84aaf1c9485e060c503a0ff8cadb.woff2","method":"get","headers":
                            {"host":"localhost:5601","connection":"keep-alive","origin":"http://localhost:5601","user-agent":"Mozilla/5.0
                            (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100
                            Safari/537.36","accept":"*/*","referer":"http://localhost:5601/bundles/commons.style.css","accept-encoding":"gzip,

Jul 8, 2019 @ 14:24:51.779  log: {"type":"response","@timestamp":"2019-07-08T18:24:51Z","tags":[],"pid":1,"method":"put","statusCode":200,"req":
                            {"url":"/api/saved_objects/index-pattern/ab5e9f30-a1ad-11e9-a730-792a99dd4f03","method":"put","headers":
                            {"host":"localhost:5601","connection":"keep-alive","content-length":"11710","origin":"http://localhost:5601","kbn-
                            version":"7.2.0","user-agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko)
                            Chrome/75.0.3770.100 Safari/537.36","content-

Jul 8, 2019 @ 14:24:51.221  log: {"type":"response","@timestamp":"2019-07-08T18:24:51Z","tags":[],"pid":1,"method":"get","statusCode":200,"req":
                            {"url":"/api/index_patterns/_fields_for_wildcard?pattern=logstash-
                            *&meta_fields=_source&meta_fields=_id&meta_fields=_type&meta_fields=_index&meta_fields=_score","method":"get","headers":
                            {"host":"localhost:5601","connection":"keep-alive","accept":"application/json, text/plain, */*","kbn-
                            version":"7.2.0","user-agent":"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko)

Jul 8, 2019 @ 14:24:51.061  log: {"type":"response","@timestamp":"2019-07-08T18:24:51Z","tags":[],"pid":1,"method":"post","statusCode":200,"req":
                            {"url":"/api/saved_objects/_bulk_get","method":"post","headers":{"host":"localhost:5601","connection":"keep-
                            alive","content-length":"70","origin":"http://localhost:5601","kbn-version":"7.2.0","user-agent":"Mozilla/5.0
                            (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36","content-
                            type":"application/json","accept":"*/*","referer":"http://localhost:5601/app/kibana","accept-encoding":"gzip, deflate,
```

At this point you've successfully configured and rolled out the EFK stack on your Kubernetes cluster. To learn how to use Kibana to analyze your log data, consult the Kibana User Guide.

In the next optional section, we'll deploy a simple counter Pod that prints numbers to stdout, and find its logs in Kibana.

Step 5 (Optional) — Testing Container Logging

To demonstrate a basic Kibana use case of exploring the latest logs for a given Pod, we'll deploy a minimal counter Pod that prints sequential numbers to stdout.

Let's begin by creating the Pod. Open up a file called `counter.yaml` in your favorite editor:

```
$ nano counter.yaml
```

SCROLL TO TOP

Then, paste in the following Pod spec:

counter.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox
    args: [/bin/sh, -c,
            'i=0; while true; do echo "$i: $(date)"; i=$((i+1)); sleep 1; done']
```

Save and close the file.

This is a minimal Pod called `counter` that runs a `while` loop, printing numbers sequentially.

Deploy the `counter` Pod using `kubectl`:

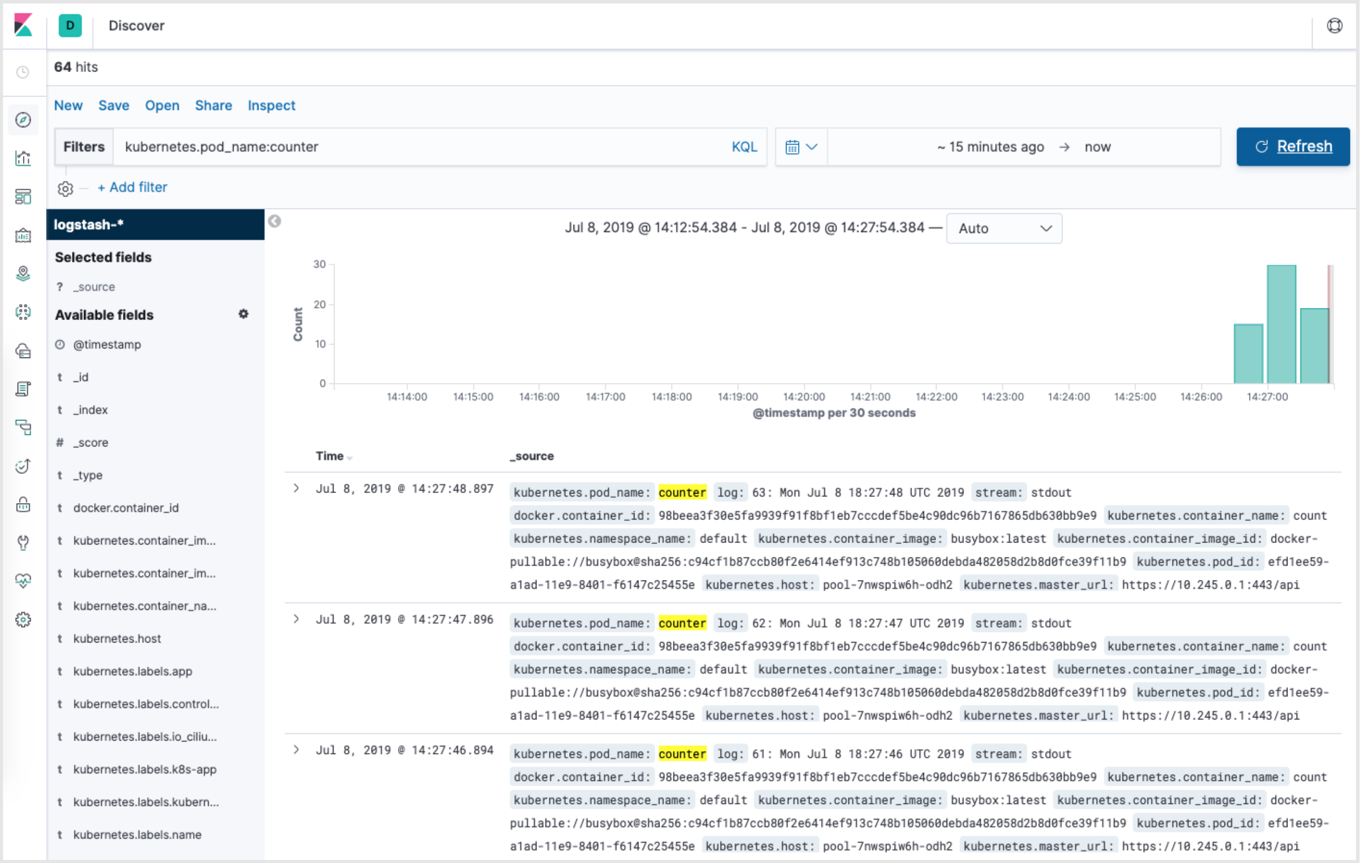
```
$ kubectl create -f counter.yaml
```

Once the Pod has been created and is running, navigate back to your Kibana dashboard.

From the **Discover** page, in the search bar enter `kubernetes.pod_name:counter`. This filters the log data for Pods named `counter`.

You should then see a list of log entries for the `counter` Pod:

SCROLL TO TOP



kubernetes.pod_name:counter

```
Jul 8, 2019 @ 14:27:48.897  kubernetes.pod_name: counter log: 63: Mon Jul 8 18:27:48 UTC 2019 stream: stdout
                            docker.container_id: 98beea3f30e5fa9939f91f8bf1eb7cccdef5be4c90dc96b7167865db630bb9e9 kubernetes.container_name: count
                            kubernetes.namespace_name: default kubernetes.container_image: busybox:latest kubernetes.container_image_id: docker-
                            pullable://busybox@sha256:c94cf1b87ccb80f2e6414ef913c748b105060debda482058d2b8d0fce39f11b9 kubernetes.pod_id: efd1ee59-
                            a1ad-11e9-8401-f6147c25455e kubernetes.host: pool-7nwspiw6h-odh2 kubernetes.master_url: https://10.245.0.1:443/api
Jul 8, 2019 @ 14:27:47.896  kubernetes.pod_name: counter log: 62: Mon Jul 8 18:27:47 UTC 2019 stream: stdout
                            docker.container_id: 98beea3f30e5fa9939f91f8bf1eb7cccdef5be4c90dc96b7167865db630bb9e9 kubernetes.container_name: count
                            kubernetes.namespace_name: default kubernetes.container_image: busybox:latest kubernetes.container_image_id: docker-
                            pullable://busybox@sha256:c94cf1b87ccb80f2e6414ef913c748b105060debda482058d2b8d0fce39f11b9 kubernetes.pod_id: efd1ee59-
                            a1ad-11e9-8401-f6147c25455e kubernetes.host: pool-7nwspiw6h-odh2 kubernetes.master_url: https://10.245.0.1:443/api
Jul 8, 2019 @ 14:27:46.894  kubernetes.pod_name: counter log: 61: Mon Jul 8 18:27:46 UTC 2019 stream: stdout
                            docker.container_id: 98beea3f30e5fa9939f91f8bf1eb7cccdef5be4c90dc96b7167865db630bb9e9 kubernetes.container_name: count
                            kubernetes.namespace_name: default kubernetes.container_image: busybox:latest kubernetes.container_image_id: docker-
                            pullable://busybox@sha256:c94cf1b87ccb80f2e6414ef913c748b105060debda482058d2b8d0fce39f11b9 kubernetes.pod_id: efd1ee59-
                            a1ad-11e9-8401-f6147c25455e kubernetes.host: pool-7nwspiw6h-odh2 kubernetes.master_url: https://10.245.0.1:443/api
```

You can click into any of the log entries to see additional metadata like the container name, Kubernetes node, Namespace, and more.

Conclusion

In this guide we’ve demonstrated how to set up and configure Elasticsearch, Fluentd, and Kibana on a Kubernetes cluster. We’ve used a minimal logging architecture that consists of a single logging agent Pod running on each Kubernetes worker node.

Before deploying this logging stack into your production Kubernetes cluster, it’s best to tune the resource requirements and limits as indicated throughout this guide. You may also want to set up X-Pack to enable built-in monitoring and security features.

The logging architecture we’ve used here consists of 3 Elasticsearch Pods, a single Kibana Pod (not load-balanced), and a set of Fluentd Pods rolled out as a DaemonSet. You may wish to scale this setup depending on your production use case. To learn more about scaling your Elasticsearch and Kibana stack, consult Scaling Elasticsearch.

Kubernetes also allows for more complex logging agent architectures that may better suit your use case. To learn more, consult Logging Architecture from the Kubernetes docs.

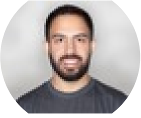
SCROLL TO TOP

Was this helpful? Yes No

54

Report an issue

About the authors



Hanif Jetha

DevOps Engineer, Technical Writer and Editor

Still looking for an answer?

Ask a question

Search for more help

RELATED

DigitalOcean Spaces: Simple Object Storage
Product

How DigitalOcean Manages, Monitors, and Secures Kubernetes for App Platform
Tutorial

How To Install Apache Kafka on Ubuntu 20.04
Tutorial

SCROLL TO TOP

54 Comments

Leave a comment...

Sign In to Comment

♡ 3 **ciaran4d51781530ee70807025** December 4, 2018

Very detailed and helpful tutorial thanks.

Reply Report

♡ 4 **ciaran4d51781530ee70807025** December 4, 2018

However, I'm having an issue on Digital Ocean's Kubernetes Service, I've tried on a new cluster, v1.12.3 and each time the elastic search stateful set fails due to issues with pvc's,

e.g. Warning FailedScheduling 16m default-scheduler pod has unbound immediate PersistentVolumeClaims

Reply Report

> ♡ 2 **leianivey** December 13, 2018
>
> same issue.
>
> Reply Report
>
> ♡ 2 **hjet** December 19, 2018
>
> Hi and thanks for your feedback!!
>
> I just tested this on a DigitalOcean Kubernetes cluster (1.12.3) and was not able to replicate your issue (the ES StatefulSet was successfully created and I was able to `curl` the API) .
>
> Does this occur intermittently for you? How large is your cluster, and what type of Nodes are you using? How much memory and vCPU do those Nodes have? Do you have multiple Node pools?
>
> Thanks!

SCROLL TO TOP

Reply Report

ciaran4d51781530ee70807025 December 23, 2018

2

Hi,

I just retried on 1.12.3 with 3 nodes with 2GB RAM each and it worked. I don’t remember my previous configuration but I suspect I had 1 node pool with 2 nodes with 1GB RAM each and 1 node pool with a single node with 4GB RAM.

Thanks for following up.

Reply Report

radugabor October 26, 2019

0

You need to create StorgeClass and PersistentVolume:

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: data

namespace: kube-logging

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer

Create path /mnt/disk/vol1 with CHMOD 777

apiVersion: v1

kind: PersistentVolume

metadata:

name: data

namespace: kube-logging

spec:

capacity:

storage: 5Gi

accessModes:

- ReadWriteOnce persistentVolumeReclaimPolicy: Retain storageClassName: data local: path: /mnt/disk/vol1 nodeAffinity: required: nodeSelectorTerms:
  - matchExpressions:
    - key: kubernetes.io/hostname operator: In values:
      - your-node-name

Reply Report

BuddyCasino February 28, 2019

1

SCROLL TO TOP

This is the only detailed, correct and up-to-date tutorial on setting up a K8S Elasticsearch cluster on whole internet. Thanks!

Reply Report

1 poojajagdale18 March 14, 2019

curl http://localhost:9200/_cluster/state?pretty gives me output as:
“error” : {
“root*cause” : [*
{
*“type” : “master*not*discovered*exception”,
“reason” : null
}
],
“type” : “master*not*discovered_exception”,
“reason” : null
},
“status” : 503

Reply Report

0 hbinduni May 19, 2019

it seem you forgot to deploy the service:

```
kubectl create -f elasticsearch_svc.yaml
```

Reply Report

0 mishaveldhoen May 28, 2019

This exception also occurs when you try to deploy ES v7 instead of v6 with the current settings. If this is the case, then you need to add the environment variable `cluster.initial_master_nodes`.

Reply Report

0 rohitdharmapuri May 18, 2020

My EFK stack is up and while running the below port forwards
kubectl port-forward kibana-df8d558dd-27j9m 5601:5601 –namespace=logging
kubectl port-forward es-cluster-0 9200:9200 –namespace=logging

I’m accessing my kibana on my chrome at this http://localhost:5601/

I’m getting this error intermittently {“statusCode”:503,“error”:“Service Unavailable”,“message”:“blc *UNAVAILABLE/1/state not recovered /*

SCROLL TO TOP

*initialized];: [cluster*block*exception] blocked by: [SERVICE*UNAVAILABLE/1/state not recovered / initialized];”}

I made sure that I have the Kibana service up and ES has the following configuration

```
containers:
- env:
  - name: cluster.name
    value: k8s-logs
  - name: node.name
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: metadata.name
  - name: discovery.seed_hosts
    value: es-cluster-0.elasticsearch,es-cluster-1.elasticsearch,es
  - name: cluster.initial_master_nodes
    value: es-cluster-0,es-cluster-1,es-cluster-2
  - name: ES_JAVA_OPTS
    value: -Xms512m -Xmx512m
  image: gcr.io/****-****/efk/elasticsearch/elasticsearch:7.7.0
  imagePullPolicy: IfNotPresent
  name: elasticsearch
  ports:
  - containerPort: 9200
    name: rest
    protocol: TCP
  - containerPort: 9300
    name: inter-node
    protocol: TCP```
```

Reply Report

0 jjohnson53b4c0bcd9f28deb44 April 24, 2019

One can also consider Fluent-bit. The bit variance has lighter functionality than Fluentd and for many its smaller size may make more sense.

A Katacoda scenario is also available to walk you through setting up EFK here.

Reply Report

0 johnhalbert May 21, 2019

Very detailed, thank you. Even with the recommended settings for `ES_JAVA_OPTS`, my Elasticsearch containers are dying due to OOM error. Looking at the available resources on the nodes where the pod is schedu SCROLL TO TOP e an issue. Any advice?

Reply Report

0 **psuchy2** May 29, 2019

Hello, try to increase limit on your container resource. I had the same issue:

```
resources:
  limits:
  cpu: 1000m
  memory: 2.5G
```

Reply Report

0 **mishaveldhoen** May 28, 2019

Great tutorial, thanks!

In case you're trying to deploy elasticsearch v7.x.x using these settings, no master node will be found. ES v7 introduces the `cluster.initial_master_nodes` setting, which will need to correspond exactly to the `node.name`, see the ES reference, e.g., the node name must be `es-cluster-#.elasticsearch`.

One way to fix it is by changing

```
- name: node.name
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
```

into

```
- name: NODE_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name
- name: node.name
    value: "$(NODE_NAME).elasticsearch"
- name: cluster.initial_master_nodes
  value: "es-cluster-0.elasticsearch,es-cluster-1.elasticsearch,es-cluster-2.el
```

Reply Report

2 **nsaitov** July 9, 2019

Hi @mishaveldhoen , i am trying to create an elasticsearch_statefulset, but it got stuck on

SCROLL TO TOP

Waiting for 3 pods to be ready…
I follow almost the same yaml file, except part for env as u mentioned:

env:
- name: cluster.name
value: k8s-logs
- name: NODE*NAME*
valueFrom:
fieldRef:
fieldPath: metadata.name
- name: node.name
*value: “$(NODE*NAME).elasticsearch”
- name: cluster.initial*master*nodes
value: “es-cluster-0.elasticsearch,es-cluster-1.elasticsearch,es-cluster-2.elasticsearch”
- name: ES*JAVA*OPTS
value: “-Xms512m -Xmx512m”

Please, let me know if i am doing anything wrong and how to proceed… Thanks in advance!

Reply Report

josesantos October 24, 2019

Me too.
Have you solved?

Reply Report

psuchy2 May 29, 2019

Hello, I followed this tutorial, but after a few days I started to receive following message and no new logs are send to elasticsearch.

```
2019-05-29 12:39:58 +0000 [warn]: emit transaction failed: error_class=Fluent
```

Any tips, how to properly tweak fluentd?

Thanks

Reply Report

jeanandrewfuentes June 25, 2019

Hi great details on the tutorial, however I’m still getting an error.

plugin:elasticsearch@6.4.3 Service Unavailable

SCROLL TO TOP

at the kibana part it says status red too. and upon check of the pods of elastic this is the error.

```
fatal error on the network layer
    at org.elasticsearch.transport.netty4.Netty4Utils.maybeDie(Netty4Utils.j
    at org.elasticsearch.transport.netty4.Netty4MessageChannelHandler.excepti
    at io.netty.channel.AbstractChannelHandlerContext.invokeExceptionCaught(
    at io.netty.channel.AbstractChannelHandlerContext.invokeExceptionCaught(
```

Also this is the logs when I tried to proxy and curl elastic search

```
kubectl port-forward es-cluster-0 9200:9200 --namespace=kube-logging
Forwarding from 127.0.0.1:9200 -> 9200
Forwarding from [::1]:9200 -> 9200
Handling connection for 9200
E0625 10:11:26.617570    7659 portforward.go:400] an error occurred forwardi
```

Reply Report

1 mostafashr July 6, 2019

I have the same error "FailedScheduling 16m default-scheduler pod has unbound immediate".
I test the system and find-out if I use kube-logging namespace it cannot run ES pods and this error occurred but if I apply it on default namespace it works correctly.
how can I fix it and have the pods in logging name-space?
I am using single node cluster. I test it on other service provider's single node cluster and minikube and Kind the problem still persist. I think I'm missing something here.

Reply Report

5 stalinbritto July 12, 2019

Hi Team,
I have followed and when I execute elasticsearch_statefulset.yaml, am getting below output and it's taking a long time.
It couldn't step forward further.

root@hostname:~/kube# kubectl rollout status sts/es-cluster –namespace kube-logging
Waiting for 3 pods to be ready...

Reply Report

2 josesantos October 24, 2019

SCROLL TO TOP

Me too.

Have you solved?

Reply Report

dakadivya July 31, 2019

0 Can u make a video how to store application logs(destination)to EFK (source) server.

Reply Report

jain007neeraj October 2, 2019

1 Awesome Explanation, thanks Hanif, I just have one query how can I configure the `fluentd` to fetch logs from `default` namespace, Since right now it’s just reading from `kube-logging` namespace.

Reply Report

alexdavis0319 April 1, 2021

0 Any solution to this?

Reply Report

Amipb October 7, 2019

0 Great tutorial, but I’m stuck with the same kind of problem stated above : the rollout hangs at “Waiting for 3 pods to be ready”.

Reply Report

josesantos October 24, 2019

0 Me too.

Have you solved?

Reply Report

Amipb October 28, 2019

0 No, not yet.

Reply Report

HasanGoz October 25, 2019

0 Same problem but the output is like this;

Waiting for 3 pods to be ready…

Waiting for 2 pods to be ready…

Waiting for 3 pods to be ready…

Waiting for 2 pods to be ready…

and stuck like this

Reply Report

SCROLL TO TOP

phucphanhong February 14,

0

I had this issue, it is because you have no enough Node to run this infrastructure. Increase the nodes or mark the cluster as scalable.

Reply Report

sysadmin57e164f7fa058bd00e October 26, 2019

0

Hi Hanif,

Thanks for this useful tutorial, i am successfully deployed EFK stack.
But in Kibana Dashboard i can't discover all namespaces that located on my k8s cluster ?!

Reply Report

contraboy9999 November 18, 2019

0

Thanks for the detailed article. I am looking into implementing this but I wanted to know the minimum amount of storage the cluster should have which I need to get started on this. I will be scaling this out after the POC though. Thanks!!

Reply Report

contraboy9999 December 6, 2019

0

HI,
Thankk you for the detailed article:

I am having an issue when executing the following command.
I just get the below output and it goes in hung state right here. Below is the output:

```
kubectl rollout status sts/es-cluster –namespace=kube-logging
```

Output:

```
Waiting for 3 pods to be ready…
```

I have 1 master and 2 nodes in my cluster. Any help would be appreciated.

Reply Report

vinx April 20, 2020

1

Everything works but I had to make a change to use my nfs server as a storage class for elastic search,I always had the error message: "pod has unbound immediate .."
but this was simply a problem of delay in the creation of the pvc, the new attempt should have been solved.
The real problem with was the init container fix-permissions, my volume nfs is with all*squash and root*squash, there is no need to change the owner with choown in the init container (command: ["sh", "-c" , "chown -R 1000: 1000 / usr / share / elasticsearch / data"]), the owner is always nobody, and furthermore this command is blocked, the init container fails and pod not start. I have removed the "fix-permission" init container and everything is working properly.
In your own storage class installation you have to pay attention to this aspect.

SCROLL TO TOP

I hope it will be useful to others

Vinx

Reply Report

shivamsinghal May 7, 2020

0

Thanks you for the detailed article:

I am having an issue when executing the following command.
I just get the below output and it goes in hung state right here. Below is the output:

kubectl rollout status sts/es-cluster –namespace=kube-logging

Output:
Waiting for 3 pods to be ready…

Reply Report

sasidharallnew June 21, 2020

0

i am also facing same issue any luck in solving ?

Reply Report

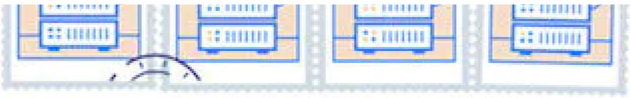
kacperpb96 July 24, 2020

0

describe statefulset of your elasticsearch deployment and make sure you don't have problem with privileged pod permissions. In my case I has to create SecurityAccount/ClusterRole/ClusterRoleBinding and attach it in specs

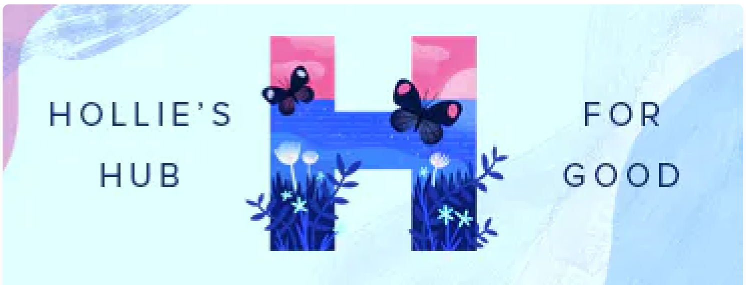
Reply Report

Load More Comments



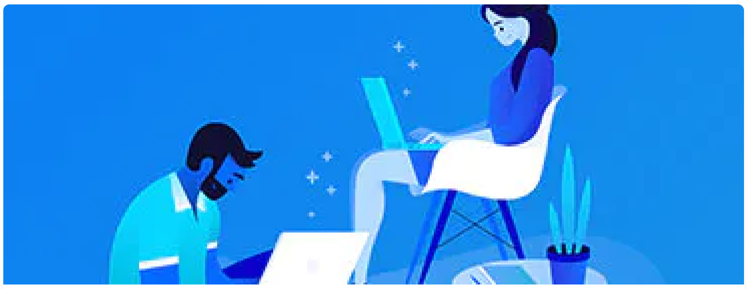
GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.



HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.



BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

Featured on Community Kubernetes Course Learn Python 3 Machine Learning in Python Getting started with Go Intro to Kubernetes

DigitalOcean Products Virtual Machines Managed Databases Managed Kubernetes Block Storage Object Storage SCROLL TO TOP Load Balancers

Company

About

Leadership

Blog

Careers

Partners

Referral Program

Press

Legal

Security & Trust Center

Products

Pricing

Products Overview

Droplets

Kubernetes

Managed Databases

Spaces

Marketplace

Community

Tutorials

Q&A

Tools and Integrations

Tags

Product Ideas

SCROLL TO TOP

Presentation Grants

Contact

Get Support

Trouble Signing In?

Sales

Report Abuse

System Status

Load Balancers
Block Storage
API Documentation
Documentation
Release Notes

Hatch Startup Program
Shop Swag
Research Program
Open Source
Code of Conduct

SCROLL TO TOP