## **Deploy Fluentd Kubernetes Manifests**

****Fluentd is deployed as a daemonset**** since it has to stream logs from all the nodes in the clusters. In addition to this, it requires special permissions to list & extract the pod’s metadata in all the namespaces.

[Kubernetes Service accounts](https://devopscube.com/kubernetes-api-access-service-account/" \t "https://devopscube.com/setup-efk-stack-on-kubernetes/_blank) are used for providing permissions to a component in kubernetes, along with cluster roles and cluster rolebindings. Let’s go ahead and create the required service account and roles.

### **Create Fluentd Cluster Role**

A cluster role in kubernetes contains rules that represent a set of permissions. For ****fluentd****, we want to give permissions for pods and namespaces.  
  
Create a manifest fluentd-role.yaml

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: fluentd

labels:

app: fluentd

rules:

- apiGroups:

- ""

resources:

- pods

- namespaces

verbs:

- get

- list

- watch

Apply the manifest

kubectl create -f fluentd-role.yaml

### **Create Fluentd Service Account**

A service account in kubernetes is an entity to provide identity to a pod. Here, we want to create a service account to be used with fluentd pods.  
  
Create a manifest fluentd-sa.yaml

apiVersion: v1

kind: ServiceAccount

metadata:

name: fluentd

labels:

app: fluentd

Apply the manifest

kubectl create -f fluentd-sa.yaml

### **Creste Fluentd Cluster Role Binding**

A cluster rolebinding in kubernetes grants permissions defined in a cluster role to a service account. We want to create a rolebinding between the role and the service account created above.  
  
Create a manifest fluentd-rb.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: default

Apply the manifest

kubectl create -f fluentd-rb.yaml

### **Deploy Fluentd DaemonSet**

Let us deploy the daemonset now.

Save the following as fluentd-ds.yaml

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: fluentd

labels:

app: fluentd

spec:

selector:

matchLabels:

app: fluentd

template:

metadata:

labels:

app: fluentd

spec:

serviceAccount: fluentd

serviceAccountName: fluentd

containers:

- name: fluentd

image: fluent/fluentd-kubernetes-daemonset:v1.4.2-debian-elasticsearch-1.1

env:

- name: FLUENT\_ELASTICSEARCH\_HOST

value: "elasticsearch.default.svc.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

****Note****: If you check the deployment, we whave use two env vars, "FLUENT\_ELASTICSEARCH\_HOST" & "FLUENT\_ELASTICSEARCH\_PORT". Fluentd uses these Elasticsearch values to ship the collected logs.

Lets apply the fluentd manifest

kubectl create -f fluentd-ds.yaml

### **Verify Fluentd Setup**

In order to verify the fluentd installation, let us start a pod that creates logs continuously. We will then try to see these logs inside Kibana.  
  
Save the following as test-pod.yaml

apiVersion: v1

kind: Pod

metadata:

name: counter

spec:

containers:

- name: count

image: busybox

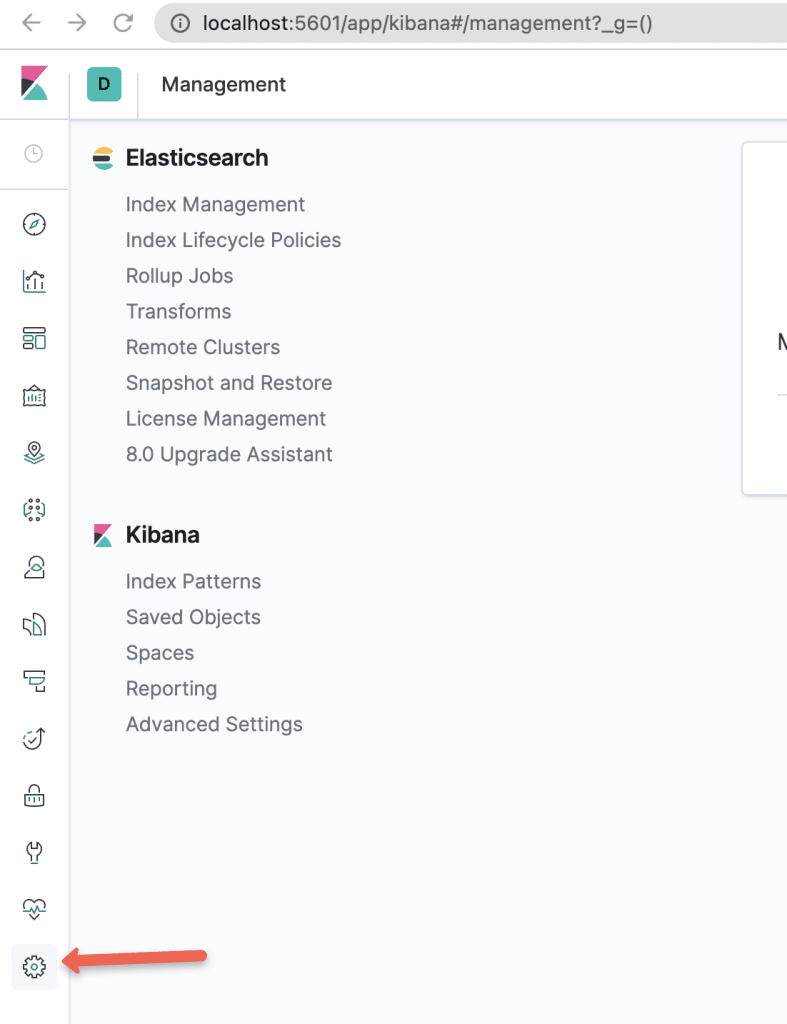
args: [/bin/sh, -c,'i=0; while true; do echo "Thanks for visiting devopscube! $i"; i=$((i+1)); sleep 1; done']

Apply the manifest

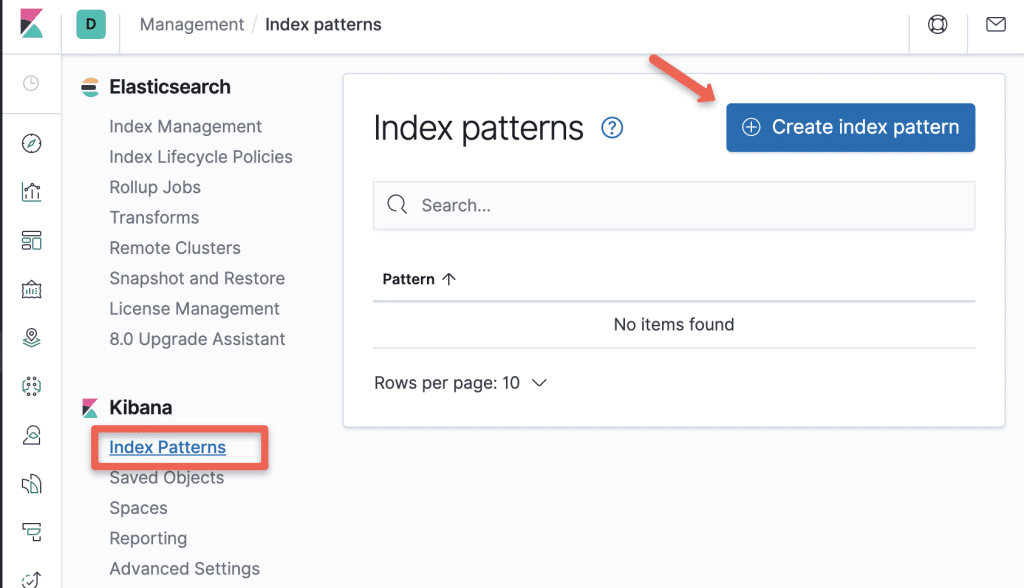
kubectl create -f test-pod.yaml

Now, let’s head to Kibana to check whether the logs from this pod are being picked up by fluentd and stored at elasticsearch or not. Follow the below steps:

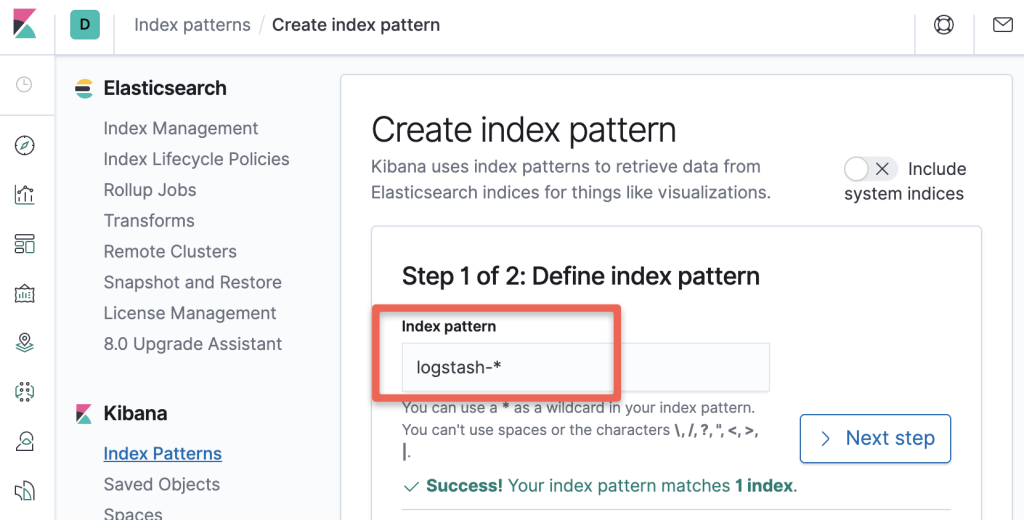
****Step 1:**** Open kibana UI using proxy or the nodeport service endpoint. Head to management console inside it.

[](https://devopscube.com/wp-content/uploads/2021/12/image.png)

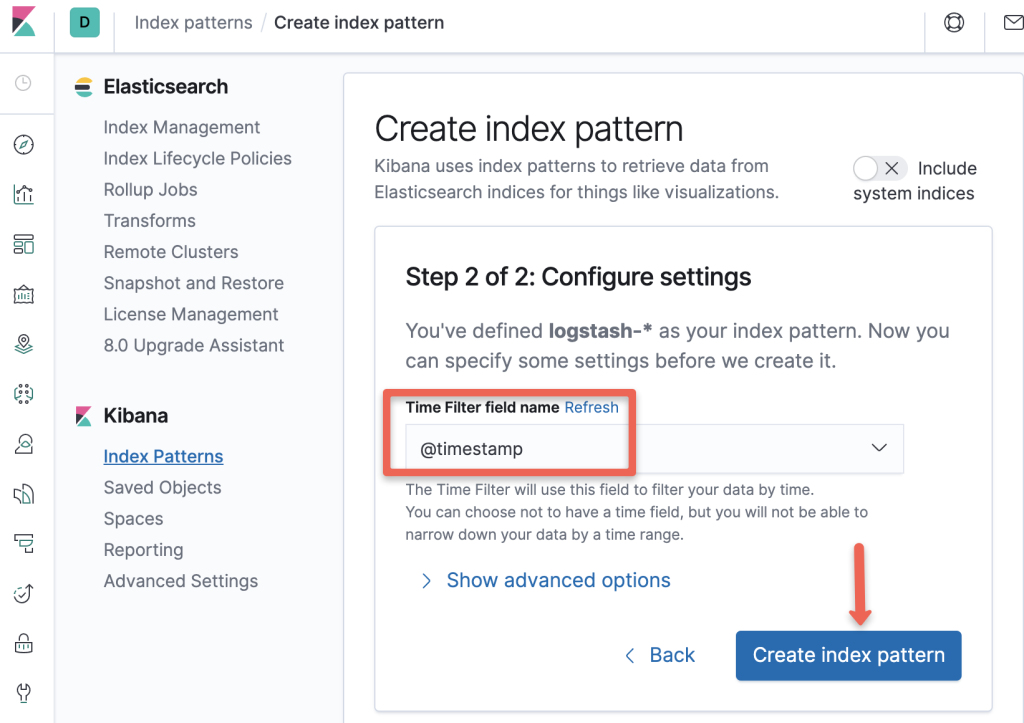
****Step 2:**** Select the “Index Patterns” option under Kibana section.

[](https://devopscube.com/wp-content/uploads/2021/12/image-1.png)

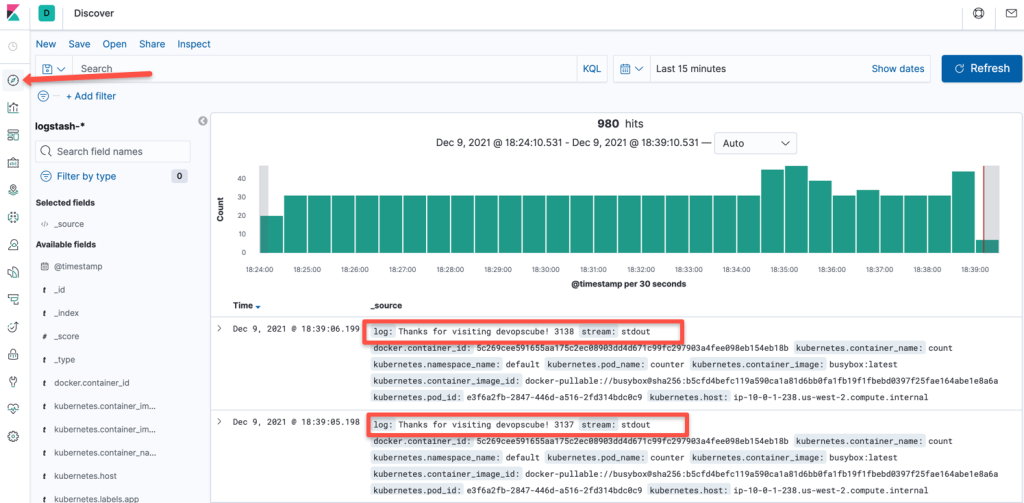
****Step 3:**** Create a new Index Patten using the pattern – “logstash-\*” and

[](https://devopscube.com/wp-content/uploads/2021/12/image-2.png)

****Step 4:**** Select “@timestamp” in the timestamps option.

[](https://devopscube.com/wp-content/uploads/2021/12/image-3.png)

****Step 5:**** Now the index pattern has been created. Head to discover console.Here, you will be able to see all the logs being exported by fluentd like the logs from our test pod as shown in the image below.

[](https://devopscube.com/wp-content/uploads/2021/12/image-4.png)

That’s it!  
  
We have ****covered all the components required for a logging solution in Kubernetes****and also verified each of our components separately. Let us go through the best practices of using EFK stack.

## **Kubernetes EFK Best practises**

1. ****Elasticsearch uses heap memory extensively**** for filtering and caching for better query performances, so ample memory should be available for elastic search.  
     
   Giving more than half of total memory to elasticsearch could also leave too less memory for OS functions which could inturn hamper elasticsearch’s capabilities.  
     
   So be mindful of this! A ****40-50% of total heap space to elasticsearch is good enough****.
2. ****Elastic search indices can fill up quickly**** so it’s important to clean up old indices regularly. Kubernetes cron jobs can help you do this regularly in an automated fashion.
3. Having data replicated across multiple nodes can ****help in disaster recovery****and also ****improve query performance****. By default, replication factor in elasticsearch is set to 1.  
     
   Consider playing around with this values according to your use case. Having atleast 2 is a good practise.
4. Data which is known to be accessed more frequently can be placed in different nodes with more resources allocated. This can be achieved by running a cronjob that moves the indices to different nodes at regular intervals.  
     
   Though this is an advance use case – it is good for a beginner to atleast have knowledge that something like this can be done.
5. In elastic search, you an archive indices to low cost cloud storage such as aws-s3 and restore when you need data from those indices.  
     
   This is a best practise if you need to ****conserve logs for audit and compliance****.
6. Having multiple nodes like master, data and client nodes with dedicated functionalities is good for high availability and fault tolerance.

## **Beyond EFK – Futher Research**

This guide was just a small use case of setting up the Elastic stack on Kubernetes. Elastic stack has tons of other features which help in logging and monitoring solutions.

For example, it can ship logs from virtual machines and managed services of various cloud providers. You can even ship logs from data engineering tools like Kafka into the elastic stack.

The elastic stack has other powerful components worth looking into, such as:

1. ****Elastic Metrics****: Ships metrics from multiple sources across your entire infrastructure and makes it available in elastic search and kibana.
2. ****APM****: Expands elastic stack capabilities and lets you analyze where exactly an application is spending time quickly fixing issues in production.
3. ****Uptime****: Helps in monitoring and analyzing availability issues across your apps and services before they start appearing in the production.