## **Log Aggregation and Analysis with EFK Stack**

EFK stands for ****Elasticsearch, Fluentd, and Kibana****. EFK is a popular and the best open-source choice for the Kubernetes log aggregation and analysis.

****Elasticsearch**** is a distributed and scalable search engine commonly used to sift through large volumes of log data. It is a NoSQL database based on the Lucene search engine (search library from Apache). Its primary work is to store logs and retrive logs from fluentd.

Fluentd is a log shipper. It is an ****open source log collection**** agent which support multiple data sources and output formats. Also, it can forward logs to solutions like Stackdriver, [Cloudwatch](https://devopscube.com/how-to-setup-and-push-serverapplication-logs-to-aws-cloudwatch/" \t "https://devopscube.com/setup-efk-stack-on-kubernetes/_blank), elasticsearch, Splunk, Bigquery and much more. To be short, it is an unifying layer between systems that genrate log data and systems that store log data.

****Kibana**** is UI tool for querying, data visualization and dashboards. It is a query engine which allows you to explore your log data through a web interface, build visualizations for events log, query-specific to filter information for detecting issues. You can virtually build any type of dashboards using Kibana. ****Kibana Query Language (KQL****) is used for querying elasticsearch data. Here we use Kibana to ****query indexed data in elasticsearch****.

Also, Elasticsearch helps solve the problem of separating huge amounts of unstructured data and is in use by many organizations. Elasticsearch is commonly deployed alongside Kibana.

****Note:**** When it comes to Kubernetes, Fluentd is the best choice because than logstash because FLuentd can parse container logs without any extra configurations. Moreover, it is a CNCF project.

## **Setup EFK Stack on Kubernetes**

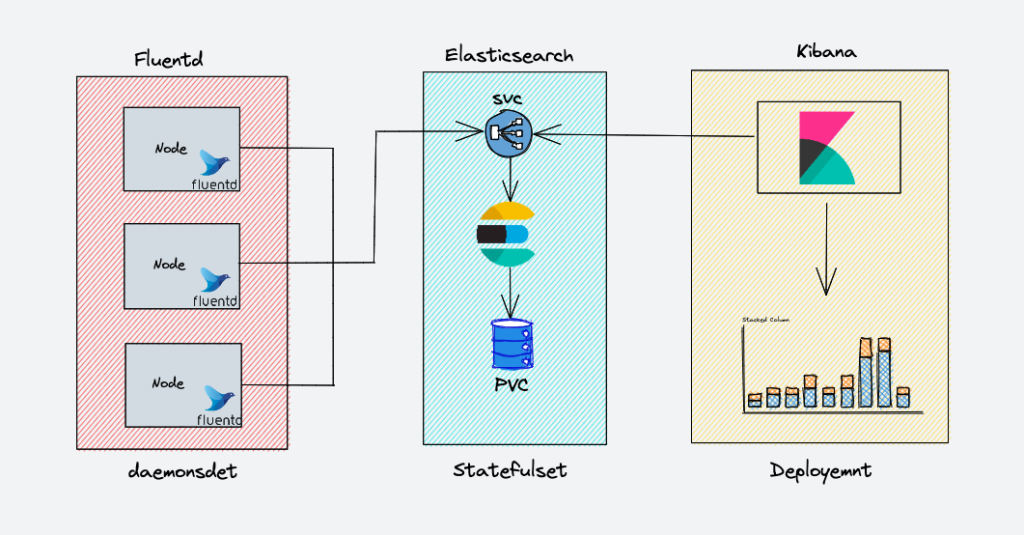
Step-by-step process for setting up EFK using Kubernetes manifests.

git clone https://github.com/scriptcamp/kubernetes-efk

****Note****: All the EFK components get deployed in the default namespace.

## **EFK Archiecture**

The following diagram shows the ****high level architecture of EFK stack**** that we are going to build.

[](https://devopscube.com/wp-content/uploads/2021/12/image-7.png)

EKF components get deployed as follows,

1. ****Fluentd****:- Deployed as daemonset as it need to collect the container logs from all the nodes. It connects to the Elasticsearch service endpoint to forward the logs.
2. ****Elasticsearch****:- Deployed as statefulset as it holds the log data. We also expose the service endpoint for Fluentd and kibana to connect to it.
3. ****Kibana****:- Deployed as deployment and connects to elasticsearch service endpoint.

## **Deploy Elasticsearch Statefulset**

Elasticsearch is ****deployed as a Statefulset**** and the multiple replicas connect with each other using a headless service. The headless svc helps in the DNS domain of the pods.  
  
Save the following manifest as es-svc.yaml

apiVersion: v1

kind: Service

metadata:

name: elasticsearch

labels:

app: elasticsearch

spec:

selector:

app: elasticsearch

clusterIP: None

ports:

- port: 9200

name: rest

- port: 9300

name: inter-node

kubectl create -f es-svc.yaml

Before creating the ****statefulset for elastic search****, let’s recall that a statefulset requires a ****storage class defined****beforehand using which it can create volumes whenever required.

****Note****: Though in a production environment, we need to use 400-500Gbs of volume for elastic search, here we are deploying with 3Gb PVC’s for demonstrations.

Let’s create the Elasticsearch statefulset now. Save the following manifest as es-sts.yaml

****Note****: The statefulset creates the PVC with the default available storage class. If you have a custom storage class for PVC, you can add it in the volumeClaimTemplates by uncommenting the storageClassName parameter.

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: es-cluster

spec:

serviceName: elasticsearch

replicas: 3

selector:

matchLabels:

app: elasticsearch

template:

metadata:

labels:

app: elasticsearch

spec:

containers:

- name: elasticsearch

image: docker.elastic.co/elasticsearch/elasticsearch:7.5.0

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-cluster-2.elasticsearch"

- 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" ]

# storageClassName: ""

resources:

requests:

storage: 3Gi

Let’s create the statefulset.

kubectl create -f es-sts.yaml

### **Verify Elasticsearch Deployment**

After the Elastisearch pods come into the running state, ****verify the Elasticsearch statefulset****. The easiest method to do this is to check the status of the cluster. In order to check the status, port-forward the Elasticsearch pod’s 9200 port.

kubectl port-forward es-cluster-0 9200:9200

To check the health of the Elasticsearch cluster, run the following command in the terminal.

curl http://localhost:9200/\_cluster/health/?pretty

The output will display the status of the Elasticsearch cluster. If all the steps were followed correctly, the status should come up as ‘green’.

{

"cluster\_name" : "k8s-logs",

"status" : "green",

"timed\_out" : false,

"number\_of\_nodes" : 3,

"number\_of\_data\_nodes" : 3,

"active\_primary\_shards" : 8,

"active\_shards" : 16,

"relocating\_shards" : 0,

"initializing\_shards" : 0,

"unassigned\_shards" : 0,

"delayed\_unassigned\_shards" : 0,

"number\_of\_pending\_tasks" : 0,

"number\_of\_in\_flight\_fetch" : 0,

"task\_max\_waiting\_in\_queue\_millis" : 0,

"active\_shards\_percent\_as\_number" : 100.0

}

### **Tip on Elasticsearch Headless Service**

As you know, headless svc does not work as a load balancer and is used to address a group of pods together. There is another use case for headless services.

We can use it to get the address of individual pods. Let’s take an e.g. to understand this.

We have three pods running as part of the Elastic search statefulset.

|  |  |
| --- | --- |
| ****Pod name**** | ****Pod Address**** |
| es-cluster-0 | 172.20.20.134 |
| es-cluster-1 | 172.20.10.134 |
| es-cluster-2 | 172.20.30.89 |

Elasticsearch Pods and their addresses

and a headless svc – “elasticsearch” is pointed to these pods.

If you do a nslookup from a pod running inside the same namespace of your cluster, you’ll be able to get the address of the above pods through the headless svc.

****nslookup es-cluster-0.elasticsearch.default.svc.cluster.local****Server: 10.100.0.10

Address: 10.100.0.10#53

Name: es-cluster-0.elasticsearch.default.svc.cluster.local

Address: 172.20.20.134

The above concept is used very commonly in Kubernetes, so should be understood clearly. In fact, the ****statefulset env vars**** – “discovery.seed\_hosts” and “cluster.initial\_master\_nodes” are using this concept.

Now that we have a running Ealsticsearch cluster, let’s move on to Kibana now.

## **Deploy Kibana Deployment & Service**

Kibana can be created as a simple [Kubernetes deployment](https://devopscube.com/kubernetes-deployment-tutorial/" \t "https://devopscube.com/setup-efk-stack-on-kubernetes/_blank). If you check the following Kibana deployment manifest file, we have an env var ELASTICSEARCH\_URL defined to configure the Elasticsearch cluster endpoint. Kibana uses the endpoint URL to connect to elasticsearch.

Create the Kibana deployment manifest as kibana-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: kibana

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:7.5.0

resources:

limits:

cpu: 1000m

requests:

cpu: 100m

env:

- name: ELASTICSEARCH\_URL

value: http://elasticsearch:9200

ports:

- containerPort: 5601

Create the manifest now.

kubectl create -f kibana-deployment.yaml

Let’s create a service of type NodePort to access the Kibana UI over node IP address. We are using nodePort for demonstration purposes. However, ideally, [kubernetes ingress](https://devopscube.com/kubernetes-ingress-tutorial/" \t "https://devopscube.com/setup-efk-stack-on-kubernetes/_blank)with a ClusterIP service is used for actual project implementation.

Save the following manifest as kibana-svc.yaml

apiVersion: v1

kind: Service

metadata:

name: kibana-np

spec:

selector:

app: kibana

type: NodePort

ports:

- port: 8080

targetPort: 5601

nodePort: 30000

Create the kibana-svc now.

kubectl create -f kibana-svc.yaml

Now you will be able to access Kibana over http://<node-ip>:3000

### **Verify Kibana Deployment**

After the pods come into the running state, let us try and verify Kibana deployment. The easiest method to do this is through the UI access of the cluster.

To check the status, port-forward the Kibana pod’s 5601 port. If you have created the nodePort service, you can also use that.

kubectl port-forward <kibana-pod-name> 5601:5601

After this, access the UI through the web browser or make a request using curl

curl http://localhost:5601/app/kibana

If the Kibana UI loads or a valid curl response comes up, then we can conclude that Kibana is running correctly.