[**Deep Network GmbH**](https://www.deepnetwork.com/blog/) **Developers' Blog**

**How to Setup an ELK Stack and Filebeat on Kubernetes**

27 Jan 2020

The logs are one of the most critical parts of every infrastructure for monitoring and debugging purposes. In general, there are different types of logs in every infrastructure including third-party, system, application specific logs which have different log formats like json, syslog, text, etc. It is not trivial to handle all these different log formats. But the main challenge is not only the variety of formats but also lots of log producers, especially in cluster environments. It is not possible to perform collection and processing manually. So, to be able to overcome these challenges, you have to utilize the well-known, dedicated tools and frameworks such as [ELK Stack](https://www.elastic.co/what-is/elk-stack), [Filebeat](https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html).

**What is ELK Stack and Filebeat**

ELK is an acronym for three open source projects: Elasticsearch, Logstash and Kibana. Elasticsearch is a real-time, distributed, and scalable search and analytics engine. Logstash is a server‑side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a stash like Elasticsearch. Kibana lets users visualize data with charts and graphs in Elasticsearch. Filebeat is a lightweight shipper for forwarding and centralizing log data. Installed as an agent on your servers. It monitors the log files or locations that you specify, collects log events, and forwards them to either to Elasticsearch or Logstash for indexing. In our blog post, we are going to deploy filebeat as a DaemonSet and forward k8s logs to Logstash.

Before diving into details, if you want to know why we are deploying elasticsearch to the k8s, you can read [this article](https://sematext.com/blog/kubernetes-elasticsearch/).

**Prerequisites**

* Running aks cluster and kubectl.
* If deployments will be performed via helm k8s package manager to a rbac enabled cluster, then you should follow the next section. Otherwise, you can skip to the next section. If you are not sure whether your cluster is rbac enabled or not, please follow [this](https://stackoverflow.com/questions/51238988/how-to-check-whether-rbac-is-enabled-using-kubectl). So, before using helm, we need to give necessary permissions to the helm server side component named Tiller to create k8s resources in all the namespaces.
* In fact, there are many ways deploying elastic stack to k8s for example by official helm chart or [Elastic Cloud on k8s](https://www.elastic.co/guide/en/cloud-on-k8s/current/k8s-quickstart.html) which is pretty easy to install. But in this post, we are going to deploy our stack manually to get better understanding.
* It is very important to deploy same version for all the tools to prevent unxcpected results. In our post, we are going to use 7.5.0 version.

**Enable helm on RBAC enabled AKS Cluster**

* Create service account tiller for the Tiller server in the kube-system namespace
* Bind the cluster-admin role to this Service Account. Since we want Tiller to manage resources in all namespaces, we will use [ClusterRoleBinding](https://kubernetes.io/docs/reference/access-authn-authz/rbac/)
* You can create both of them by using kubectl with separate commands. Also, you can put the resource definitions in a manifest file (for example helm-rbac.yml) and perform kubectl apply command like in the following:

apiVersion: v1

kind: ServiceAccount

metadata:

name: tiller

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: tiller

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: cluster-admin

subjects:

- kind: ServiceAccount

name: tiller

namespace: kube-system

kubectl apply -f helm-rbac.yml

* Now you can setup Tiller to your rbac enabled cluster with the created service account with the following command:

helm init --service-account tiller --upgrade --wait

**Deploy Elasticsearch**

Deployments in k8s do not keep state in their Pods by assuming the application is stateless. Since Elasticsearch maintains state, we need to use StatefulSet which is a deployment that can maintain state. StatefulSets will ensure the same PersistentVolumeClaim stays bound to the same Pod throughout its lifetime. Unlike a Deployment which ensures the group of Pods within the Deployment stay bound to a PersistentVolumeClaim. Other than these, we need a [Headless Service](https://kubernetes.io/docs/concepts/services-networking/service/#headless-services) which is used for discovery of StatefulSet Pods. Elasticsearch can run as a single instance or in a cluster mode. If Elasticsearch instances form a cluster, they might have different [roles](https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-node.html). In our case, all the nodes are equal and share all roles by default and cluster consists of 3 nodes to avoid split-brain problem and provide high availability. You can read further about it by following [this link](https://blog.trifork.com/2013/10/24/how-to-avoid-the-split-brain-problem-in-elasticsearch/). By the way, elasticsearch cluster nodes may be understood as k8s cluster nodes. Actually, they are different and correspond to pods in k8s cluster.

It is important to deploy the Headless Service by setting clusterIP: None, first for discovery of pods. It will define a DNS domain for the elasticsearch pods.

kind: Service

apiVersion: v1

metadata:

name: elasticsearch

labels:

app: elasticsearch

spec:

selector:

app: elasticsearch

clusterIP: None

ports:

- port: 9200

name: rest

- port: 9300

name: inter-node

You can save this manifest to a file and then apply kubectl apply command to deploy. When we associate our Elasticsearch StatefulSet with this service, the service will return DNS records that point to Elasticsearch pods with the app: elasticsearch label.

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: es-cluster

spec:

serviceName: elasticsearch # provides association with our previously created elasticsearch Service.

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

memory: "2Gi"

requests:

cpu: 100m

memory: "2Gi"

ports:

- containerPort: 9200 # for REST API.

name: rest

protocol: TCP

- containerPort: 9300 # for inter-node communication.

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

# sets a list of master-eligible nodes in the cluster.

- name: discovery.seed\_hosts

value: "es-cluster-0.elasticsearch, es-cluster-1.elasticsearch,es-cluster-2.elasticsearch"

# specifies a list of master-eligible nodes that will participate in the master election process.

- name: cluster.initial\_master\_nodes

value: "es-cluster-0,es-cluster-1,es-cluster-2"

- name: ES\_JAVA\_OPTS

value: "-Xms1g -Xmx1g"

# Each init containers run to completion in the specified order.

initContainers:

# By default k8s mounts the data directory as root, which renders it inaccessible to Elasticsearch.

- 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

# To prevent OOM errors.

- name: increase-vm-max-map

image: busybox

command: ["sysctl", "-w", "vm.max\_map\_count=262144"]

securityContext:

privileged: true

# Increase the max number of open file descriptors.

- name: increase-fd-ulimit

image: busybox

command: ["sh", "-c", "ulimit -n 65536"]

securityContext:

privileged: true

# PersistentVolumes for the Elasticsearch pods.

volumeClaimTemplates:

- metadata:

name: data

labels:

app: elasticsearch

spec:

accessModes: [ "ReadWriteOnce" ]

storageClassName: default

resources:

requests:

storage: 100Gi

Again, you can save this manifest to a file and deploy it via kubectl apply command or via helm. To learn more about the deployment settings, please follow the Elasticsearch’s [Notes for production use and defaults](https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html#_notes_for_production_use_and_defaults).

To check the state of the deployment, first forward elasticsearch service to your local environment with the following:

kubectl port-forward svc/elasticsearch 9200

And perform the following requests against the REST API:

curl http://localhost:9200/\_cat/health?v

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

**Deploy Filebeat**

Since we are going to use filebeat as a log shipper for our containers, we need to create separate filebeat pod for each running k8s node by using DaemonSet. The most important thing is the [filebeat configuration](https://www.elastic.co/guide/en/beats/filebeat/current/configuring-howto-filebeat.html) file which describes which file paths are going to be tailed and in which location these collected events are delivered. After determining input and output sources with their settings, it is a straightforward task to deploy it. It is better to divide the deployment steps to understand the process in detailed.

**ServiceAccount & Role Bindings**

Since filebeat is going to be deployed to our rbac enabled cluster, we should first create a dedicated ServiceAccount.

apiVersion: v1

kind: ServiceAccount

metadata:

name: filebeat

labels:

k8s-app: filebeat

Since we want to access container logs in all the namespaces, we should create a dedicated ClusterRole.

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: filebeat

labels:

k8s-app: filebeat

rules:

- apiGroups: [""] # "" indicates the core API group

resources:

- namespaces

- pods

verbs:

- get

- watch

- list

Now we can create a binding between these two with deploying a ClusterRoleBinding.

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: filebeat

subjects:

- kind: ServiceAccount

name: filebeat

roleRef:

kind: ClusterRole

name: filebeat

apiGroup: rbac.authorization.k8s.io

**ConfigMap**

There are lots of supported input and output plugins for filebeat. You can even create your own custom one for specific needs. But, we are going to use [container input plugin](https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-input-container.html) which collects container logs under the given path. Also, to send the events directly to the Logstash, we will use [logstash output plugin](https://www.elastic.co/guide/en/beats/filebeat/current/logstash-output.html).

apiVersion: v1

kind: ConfigMap

metadata:

name: filebeat-config

labels:

k8s-app: filebeat

data:

filebeat.yml: |-

filebeat.inputs:

- type: container

enabled: true

paths:

- /var/log/containers/\*.log

# If you setup helm for your cluster and want to investigate its logs, comment out this section.

exclude\_files: ['tiller-deploy-\*']

# To be used by Logstash for distinguishing index names while writing to elasticsearch.

fields\_under\_root: true

fields:

index\_prefix: k8s-logs

# Enrich events with k8s, cloud metadata

processors:

- add\_cloud\_metadata:

- add\_host\_metadata:

- add\_kubernetes\_metadata:

host: ${NODE\_NAME}

matchers:

- logs\_path:

logs\_path: "/var/log/containers/"

# Send events to Logstash.

output.logstash:

enabled: true

hosts: ["logstash:9600"]

# You can set logging.level to debug to see the generated events by the running filebeat instance.

logging.level: info

logging.to\_files: false

logging.files:

path: /var/log/filebeat

name: filebeat

keepfiles: 7

permissions: 0644

**Deployment**

After creating related ServiceAccount and ConfigMap, we can provide them to our DaemonSet.

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: filebeat

labels:

k8s-app: filebeat

spec:

selector:

matchLabels:

k8s-app: filebeat

template:

metadata:

labels:

k8s-app: filebeat

spec:

# Refers to our previously defined ServiceAccount.

serviceAccountName: filebeat

terminationGracePeriodSeconds: 30

hostNetwork: true

dnsPolicy: ClusterFirstWithHostNet

containers:

- name: filebeat

image: docker.elastic.co/beats/filebeat:7.5.0

args: [

"-c", "/etc/filebeat.yml",

"-e",

]

env:

- name: NODE\_NAME

valueFrom:

fieldRef:

fieldPath: spec.nodeName

securityContext:

runAsUser: 0

# If using Red Hat OpenShift uncomment this:

#privileged: true

resources: # comment out for using full speed

limits:

memory: 200Mi

requests:

cpu: 500m

memory: 100Mi

volumeMounts:

- name: config

mountPath: /etc/filebeat.yml

readOnly: true

subPath: filebeat.yml

- name: data

mountPath: /usr/share/filebeat/data

- name: varlibdockercontainers

mountPath: /var/lib/docker/containers

readOnly: true

volumes:

# Bind previously defined ConfigMap

- name: config

configMap:

defaultMode: 0600

name: filebeat-config

- name: varlibdockercontainers

hostPath:

path: /var/lib/docker/containers

- name: varlog

hostPath:

path: /var/log

# data folder stores a registry of read status for all files, so we don't send everything again on a Filebeat pod restart

- name: data

hostPath:

path: /var/lib/filebeat-data

type: DirectoryOrCreate

After deploying, you should see one filebeat pod for each node in your cluster. If you want to further investigate what is going on with your pod, you can change the logging.level to debug and issue the kubectl logs command to one of your pods. The logs are very descriptive.

**Deploy Logstash**

The deployment is simpler than filebeat and again the most important part is to configure it correctly by following the article [Configuring Logstash](https://www.elastic.co/guide/en/logstash/current/configuration.html). First we need to create the related ConfigMap like we do in the filebeat deployment section.

**ConfigMap**

There are many supported [input plugins](https://www.elastic.co/guide/en/logstash/current/input-plugins.html) and we are going to use [beats](https://www.elastic.co/guide/en/logstash/current/plugins-inputs-beats.html) which will receive the events from the filebeat instances. Also, there lots of supprted [output plugins](https://www.elastic.co/guide/en/logstash/current/output-plugins.html) and we will use [elasticsearch](https://www.elastic.co/guide/en/logstash/current/plugins-outputs-elasticsearch.html) to send events to elasticsearch under pre-defined indexes. Since container logs are in json format, we can use the [json filter plugin](https://www.elastic.co/guide/en/logstash/current/plugins-filters-json.html) to decode them.

apiVersion: v1

kind: ConfigMap

metadata:

name: logstash-config

data:

logstash.conf: |-

input {

beats {

port => "9600"

}

}

filter {

# Container logs are received with variable named index\_prefix

# Since it is in json format, we can decode it via json filter plugin.

if [index\_prefix] == "k8s-logs" {

if [message] =~ /^\{.\*\}$/ {

json {

source => "message"

skip\_on\_invalid\_json => true

}

}

}

# do not expose index\_prefix field to kibana

mutate {

# @metadata is not exposed outside of Logstash by default.

add\_field => { "[@metadata][index\_prefix]" => "%{index\_prefix}-%{+YYYY.MM.dd}" }

# since we added index\_prefix to metadata, we no longer need ["index\_prefix"] field.

remove\_field => ["index\_prefix"]

}

}

output {

# You can uncomment this line to investigate the generated events by the logstash.

# stdout { codec => rubydebug }

elasticsearch {

hosts => "elasticsearch:9200"

template\_overwrite => false

manage\_template => false

# The events will be stored in elasticsearch under previously defined index\_prefix value.

index => "%{[@metadata][index\_prefix]}"

sniffing => false

}

}

**Deployment**

After creating the ConfigMap, we can bind it to our single Logstash pod. The rest is simple. Just create a deployment object and its corresponding service which will interact with filebeat instances.

---

kind: Deployment

apiVersion: extensions/v1beta1

metadata:

name: logstash

spec:

template:

metadata:

labels:

app: logstash

spec:

hostname: logstash

containers:

- name: logstash

ports:

- containerPort: 9600

name: logstash

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

volumeMounts:

- name: logstash-config

mountPath: /usr/share/logstash/pipeline/

command:

- logstash

volumes:

# Previously defined ConfigMap object.

- name: logstash-config

configMap:

name: logstash-config

items:

- key: logstash.conf

path: logstash.conf

---

kind: Service

apiVersion: v1

metadata:

name: logstash

spec:

type: NodePort

selector:

app: logstash

ports:

- protocol: TCP

port: 9600

targetPort: 9600

name: logstash

---

After deployment, if you want to further investigate what is going on with your pod, you can uncomment the line stdout { codec => rubydebug } to display the generated events, redeploy and issue the kubectl logs command to your pod.

**Deploy Kibana**

Kibana deployment is very simple. Just one deployment with one pod replica (you can scale it according to your needs), and one Service object. This time, we can put both into the same file since it is not complicated.

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: kibana

labels:

k8s-app: kibana

spec:

selector:

matchLabels:

k8s-app: kibana

template:

metadata:

labels:

k8s-app: kibana

spec:

containers:

- name: kibana

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

resources:

requests:

cpu: 100m

limits:

cpu: 1000m

env:

- name: ELASTICSEARCH\_URL

value: http://elasticsearch.operations:9200

ports:

- containerPort: 5601

name: ui

protocol: TCP

---

apiVersion: v1

kind: Service

metadata:

name: kibana

labels:

k8s-app: kibana

spec:

ports:

- port: 5601

protocol: TCP

targetPort: ui

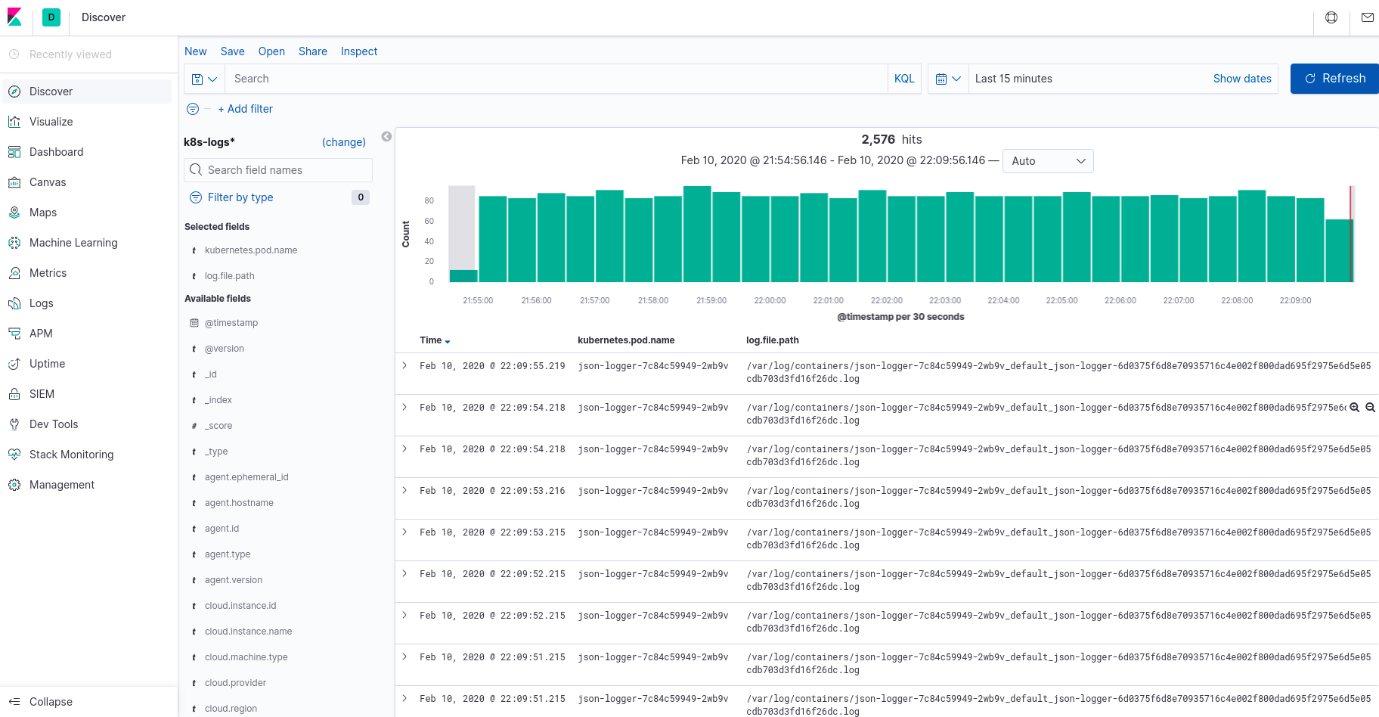
selector:

k8s-app: kibana

To access the Kibana interface, again forward a local port to the Kibana service as we do for Elasticsearch service.

kubectl port-forward svc/kibana 9200

Now you can access to the UI with the URL http://localhost:5601 and start investigating your indexes after creating corresponding index patterns.



**Summary**

In this blog post, sample ELK Stack and Filebeat deployment on k8s cluster is demonstrated. Of course, you can define your own resource requirements and limitations while deploying since this is only for demonstration purposes. As you can see, with the help of this stack, you can easily investigate your containers’ logs with a simple configuration. Also, you can extend your configuration by adding new source of inputs like syslog kernel or system package manager logs and create corresponding indexes to see what is going on behind the scenes.

**Author**

**Haluk Aktas**



Software Developer

**Related Posts**

* [**Password Protected EFK Stack on Kubernetes 13 Mar 2020**](https://www.deepnetwork.com/blog/2020/03/13/password-protected-efk-stack-on-k8s.html)
* [**Kubelet API 13 Jan 2020**](https://www.deepnetwork.com/blog/kubernetes/2020/01/13/kubelet-api.html)
* [**Understanding Networking Options in Azure AKS-Engine [Part 1] 07 Jan 2020**](https://www.deepnetwork.com/blog/2020/01/07/aksengine-advanced-networking-part1.html)