Deploying EFK (Elasticsearch, FluentD and Kibana) in a Gospel GCP cluster

Elasticsearch is a real-time, distributed, and scalable search engine which allows for full-text and structured search, as well as analytics.

Elasticsearch is going to be deployed alongside **Kibana**, a data visualization frontend and dashboard for Elasticsearch. Kibana allows us to explore the Elasticsearch log data through a web interface, and build dashboards and queries.

Fluentd is used 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.

Prerequisites

- A Kubernetes 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.

Step 1 — Creating a Namespace

To create the `kube-logging` Namespace, first open and edit a file called `kube-logging.yaml`

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

Add the following contents to the yaml file:

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

Then run:

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

You should see the following output:

```
Outputnamespace/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:

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

Step 2 — Creating the Elasticsearch StatefulSet

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.

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

```
nano elasticsearch_svc.yaml
```

Paste in the following Kubernetes service 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.

Create the service using `kubectl`:

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

You should see the following output:

```
Outputservice/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:

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

Creating the StatefulSet

Open a file called `elasticsearch_statefulset.yaml` in your favourite editor:

```
nano elasticsearch_statefulset.yaml
```

Paste in the following contents which is going to be using the default storage class for your cluster:

```
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
        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-cluster-2.elas
ticsearch"
          - 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: 100Gi
```

Now, deploy the StatefulSet using `kubectl`:

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

You should see the following output:

```
Outputstatefulset.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:

```
OutputWaiting 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:

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

Step 3 — Creating the Kibana Deployment and 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.

Open up a file called `kibana.yaml` in your favourite editor:

```
nano kibana.yaml
```

Paste in the following service spec:

```
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.

Roll out the Service and Deployment using `kubectl`:

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

You should see the following output:

```
Outputservice/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:

```
Outputdeployment "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

OutputNAME                      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-logging
```

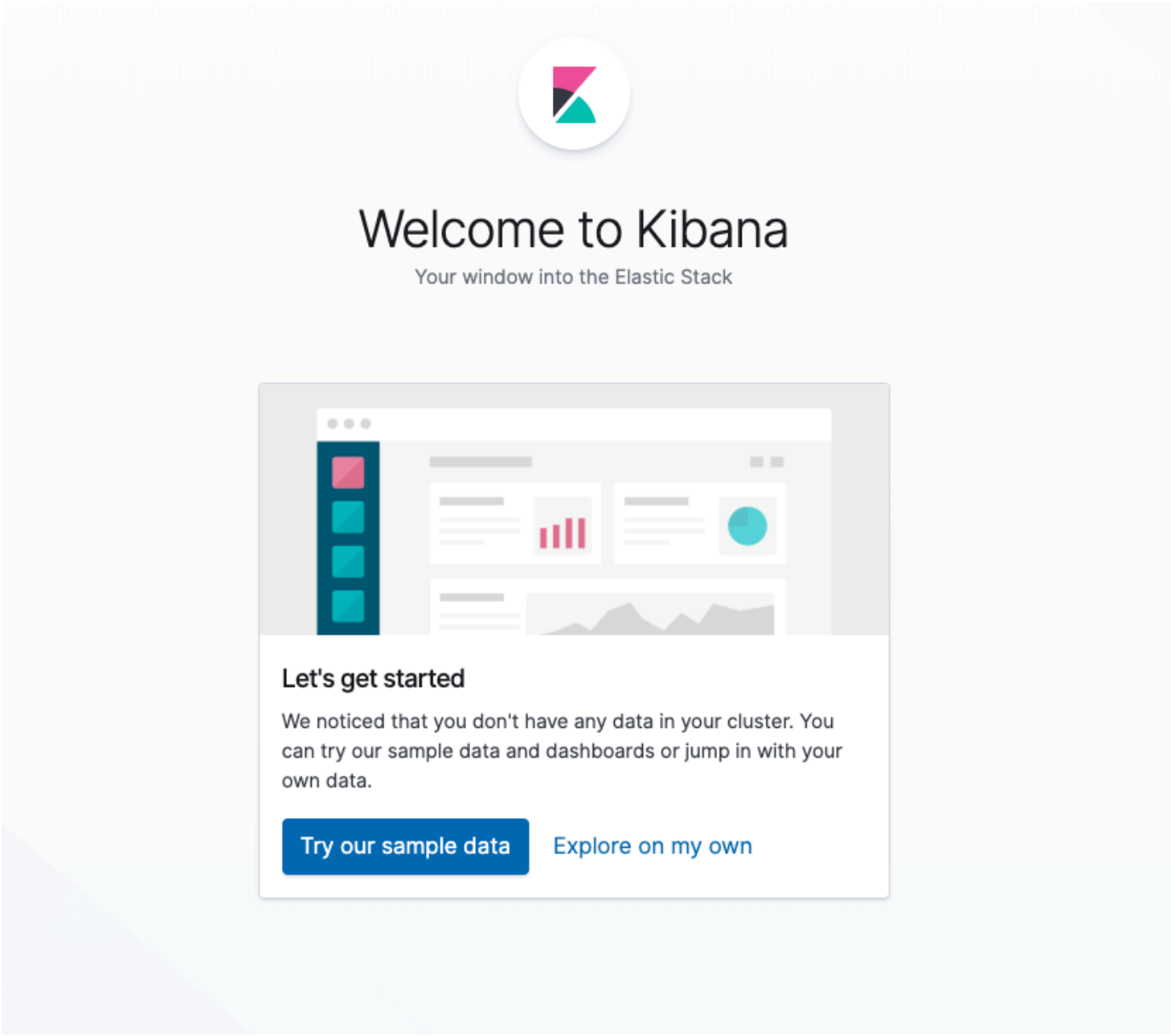
You should see the following output:

```
OutputForwarding from 127.0.0.1:5601 -> 5601
Forwarding from [::1]:5601 -> 5601
```

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:



Step 4 — Creating the Fluentd DaemonSet

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 Fluentd logging agent Pod on every node in our cluster.

In Kubernetes, containerised 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.

Open a file called `fluentd.yaml` in your favourite text editor:

```
nano fluentd.yaml
```

Then paste in the following contents:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluentd
  namespace: kube-logging
  labels:
    app: fluentd
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  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-1.1
        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_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
```

Roll out the DaemonSet using `kubectl`:

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

You should see the following output:

```
Outputserviceaccount/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 output:

```
OutputNAME      DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE
NODE SELECTOR   AGE
fluentd   3         3         3         3            3           <none>
58s
```

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

In order to check Kibana to verify that log data is being properly collected and shipped to Elasticsearch use the following steps:

1. With the `kubectl port-forward` still open, navigate to `http://localhost:5601`.
2. Click on **Discover** in the left-hand navigation menu where you will have a configuration window.
3. This allows you to define the Elasticsearch indices you'd like to explore in Kibana. To learn more, use the 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**.
4. 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 **Cr eate index pattern**.
5. Now, hit **Discover** in the left hand navigation menu: you should see a histogram graph and some recent log entries.

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.

We can begin by creating the Pod. Open up a file called `counter.yaml` in your favourite editor:

```
nano counter.yaml
```

Then, paste in the following Pod spec:

```
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.

Adapted source for GCP (original was targeted for Digital Ocean):

https://www.digitalocean.com/community/tutorials/how-to-set-up-an-elasticsearch-fluentd-and-kibana-efk-logging-stack-on-kubernetes