A Taste of Elastic Cloud on Kubernetes

By *Adam Quan*

Abstract

(Blog requirement: 160 characters maximum, including spaces)

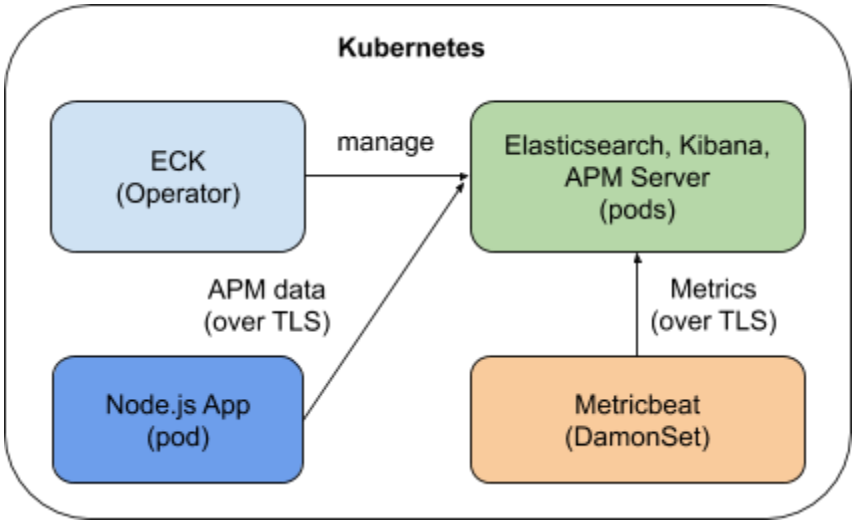
Built on the Operator pattern, ECK extends Kubernetes to support the setup and management of Elasticsearch, Kibana, and streamlines many critical operations.

Introduction

Event though still in alpha, recent announcement of Elastic Cloud on Kubernetes (ECK) has stirred up significant interests from both the Elasticsearch community and the Kubernetes community. Developed and supported by the creator of the Elastic Stack, ECK is the best way to deploy Elastic on Kubernetes. The goal of this blog is to walk you through the step-by-step process of:

1. Deploying ECK into a Kubernetes cluster (Minikube or GKE)
2. Deploying the Elastic Stack into ECK
3. Deploying a sample application instrumented with Elastic APM and sending APM data to the ECK managed Elasticsearch cluster
4. Scaling and upgrading Elasticsearch, Kibana inside ECK
5. Deploying Metricbeat to Kubernetes as a DamonSet and securely connect Metricbeat to the ECK managed Elasticsearch cluster

At the end of the blog, you will end up with a system like this:



You can also refer to the quickstart page or the product page for more up-to-date information about ECK.

What is ECK?

Kubernetes Operator is the Kubernetes way of packaging, deploying and managing a Kubernetes application. It allows you to extend the Kubernetes platform by "Putting Operational Knowledge into Software".

ECK is built on the Kubernetes Operator pattern, for the Elastic Stack. However, ECK is much more than just a Kubernetes Operator. It goes beyond just simplifying the task of deploying Elasticsearch and Kibana on Kubernetes, by streamlining many other critical operations like:

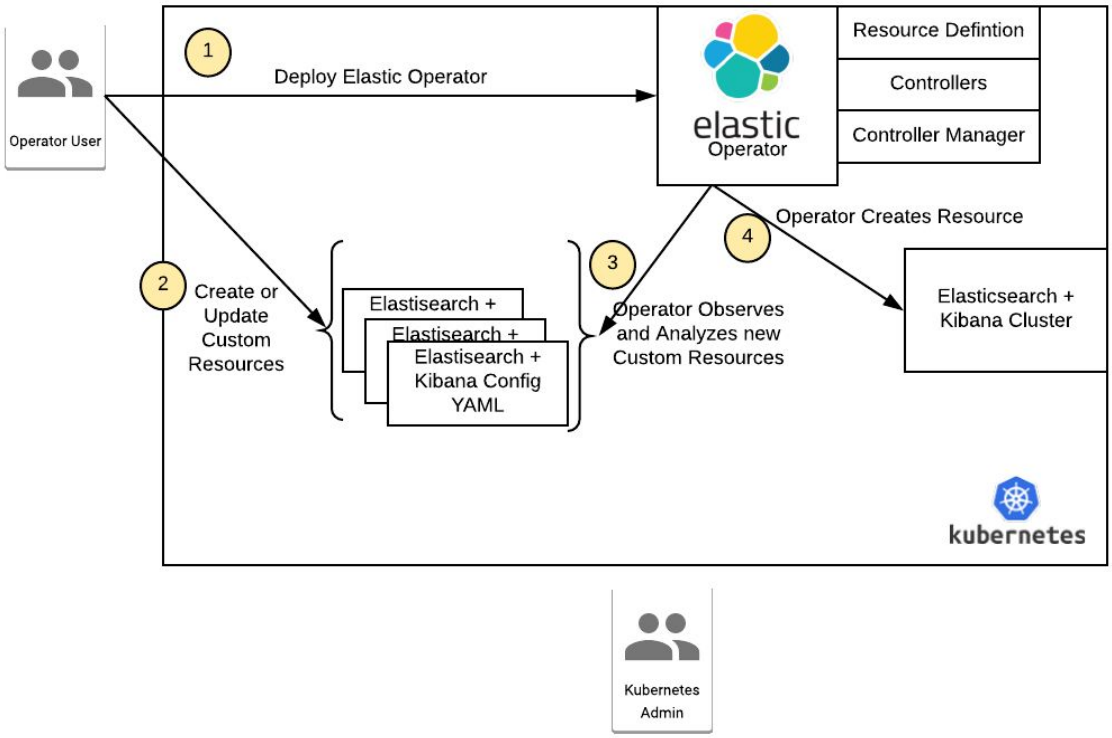
- Managing and monitoring multiple clusters
- Upgrading to new stack versions with ease
- Scaling cluster capacity up and down
- Cluster configuration changes
- Dynamically scaling local storage
- Scheduling backups
- Securing clusters with TLS certificates
- Setting up hot-warm-cold architectures with availability zone awareness

The list goes on. As you can see, it focuses on streamlining the entire experience of deploying and managing Elastic Stack on Kubernetes, in addition to automating all the operational and cluster administration tasks. As pointed out by Anurag, the vision for ECK is to provide an official way to orchestrate Elasticsearch on Kubernetes and provide a SaaS-like experience for Elastic products and solutions on Kubernetes. So, expect to see more and more exciting functionalities built into ECK!

ECK Architecture

From an architecture aspect, ECK is made up of a few different components:

- **Custom Resource Definitions** (CRDs): These are Kubernetes objects: include Elasticsearch, Kibana, APM Server. These CRDs require cluster admin level permissions to install. A custom resource is an object that extends the Kubernetes API. A CRD file defines your own object kinds and lets the API Server handle the entire lifecycle.
- **Elastic Operator** (StatefulSet): Made up of a controller and a webhook that watch for new objects to be defined. All the Elastic magic happens here.
- **Deployments** (pods): These are deployments of Elasticsearch and Kibana.
- **Certificates** (secrets): Certificates generated by the operator and contained as Kubernetes secrets
- **Default user** (secrets): The default user created to gain access to Elasticsearch and Kibana



Create your k8s cluster

Before deploying ECK and your Elastic Stack, you have to have a Kubernetes cluster ready. For the purpose of this blog, either Minikube or Google Kubernetes Engine (GKE) will do. We also support other Kubernetes distributions like OpenShift, Amazon EKS and Microsoft AKS. Because we are going to deploy ECK, Elasticsearch, Kibana, APM Server and a sample application, I recommend you give 16GB of RAM and 4 cores to your cluster. You might be able to go as small as 8GB RAM for your Kubernetes cluster but 16GB will for sure give you a smooth experience.

Minikube

If you are going to use Minikube, you can follow the installation instructions to install minikube. Once installed, configure minikube with the resources needed and start it up:

```
minikube config set memory 16384
minikube config set cpus 4
minikube start
```

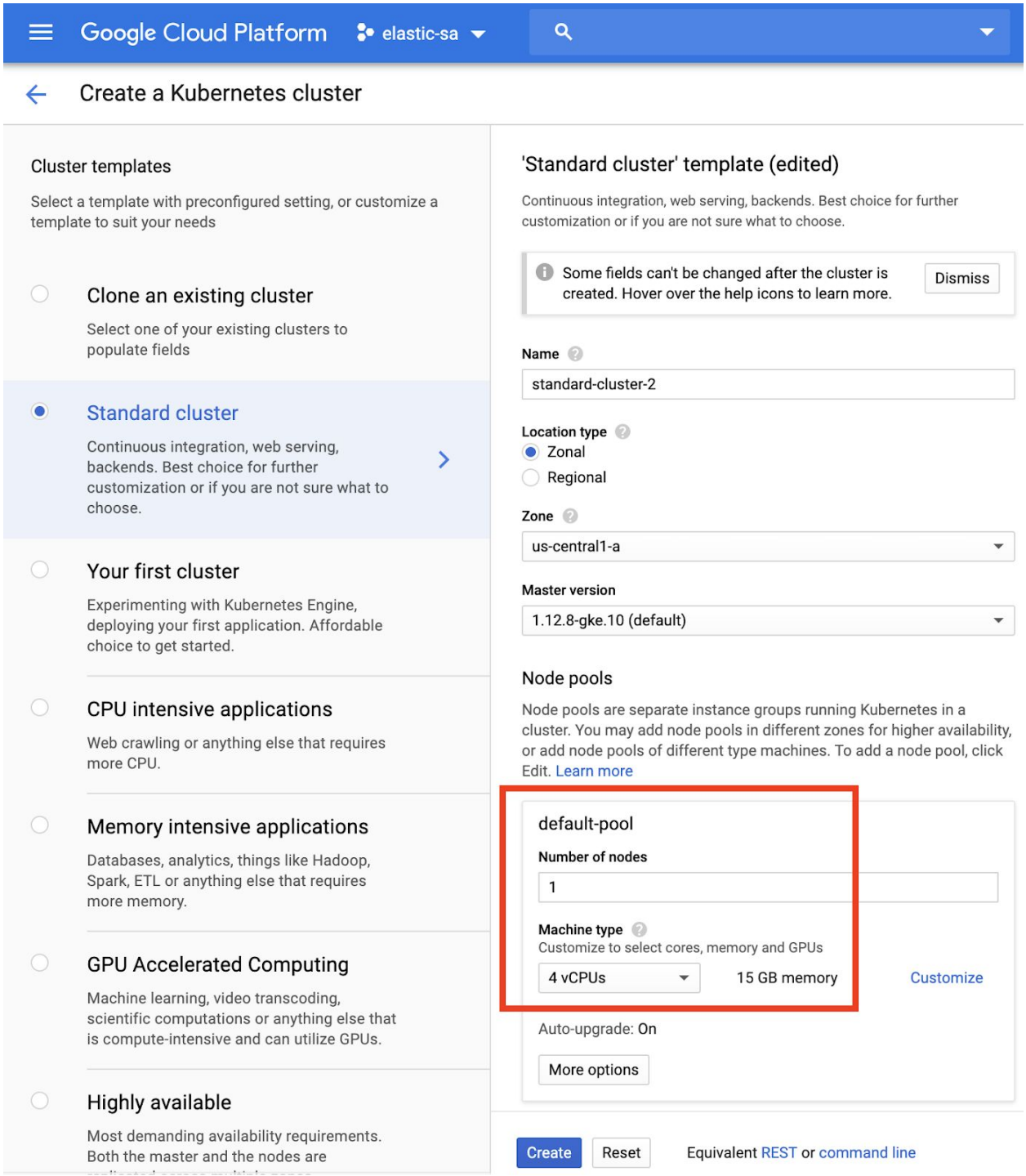
The Kubernetes dashboard is extremely convenient for monitoring and managing your Kubernetes cluster. You can enable the dashboard for Minikube with one command:

```
minikube dashboard
```

A browser window should open automatically with the Kubernetes dashboard displayed. That's it, you are ready to go.

Google Kubernetes Engine (GKE)

If you are going to use GKE, please follow google's documentation to create a GKE cluster. For your reference, here is a screenshot of my cluster configuration. As you can see, I'm creating a cluster consisting of one node of 15GB RAM and 4 vCPUs. Again, you can choose a smaller cluster but nothing below 8GB RAM.



To work with GKE, you need the gcloud command-line tool. Please see instructions from google on how to install and use it.

Once your GKE cluster is created and up running, use the following command to connect to your cluster and set the required privileges. With GKE, you have to have cluster-admin

permissions. For more information, see Prerequisites for using Kubernetes RBAC on GKE. Make sure you replace the **project name**, **zone name**, **cluster name** and **user ID** with yours when running these commands.

```
gcloud config set project elastic-sa
gcloud config set compute/zone us-central1-a
gcloud config set container/cluster aquan
gcloud auth login
gcloud container clusters get-credentials aquan --zone us-central1-a
--project elastic-sa
kubectl create clusterrolebinding cluster-admin-binding
--clusterrole=cluster-admin --user=adam.quan@elastic.co
kubectl create clusterrolebinding adam.quan-cluster-admin-binding
--clusterrole=cluster-admin --user=adam.quan@elastic.co
```

Finally, we will be using the Kubernetes command-line tool to interact with either Minikube or GKE. Please follow the instructions to install `kubectl`. Make sure that you have `kubectl` version 1.11+ installed.

Download all the artifacts we will be using for this blog from this GitHub repository.

Deploy the Elasticsearch Operator

Will you believe me if I tell you that you only need one command to deploy ECK? Yes, it does only take one command! Run the following command to deploy the latest ECK 0.9.0 as of this writing. Simple and easy.

```
kubectl apply -f
https://download.elastic.co/downloads/eck/0.9.0/all-in-one.yaml
```

Once deployed, you can use this command to monitor logs from the operator to make sure everything is going well. It should just take a few moments for ECK to be up running.

```
kubectl -n elastic-system logs -f statefulset.apps/elastic-operator
```

If you are interested, take a peek at the custom resource definition inside `all-in-one.yaml` for Elasticsearch, Kibana, ApmServer etc. You will see that lots of things happen there. No need to try to understand all of that.

Deploy the Elastic Stack

With ECK running, we are ready to deploy the Elastic Stack. It will also be one command! Here is the content of the modified `apm_es_kibana.yaml` file for your reference. Notice how

simple it is to configure Kibana and the APM server so that they can communicate with each other?

```
# This sample sets up a an Elasticsearch cluster along with a Kibana
instance
# and an APM server, configured to be able to communicate with each
other
apiVersion: elasticsearch.k8s.elastic.co/v1alpha1
kind: Elasticsearch
metadata:
  name: elasticsearch-sample
spec:
  version: 7.1.0
  nodes:
  - nodeCount: 1
    podTemplate:
      spec:
        containers:
          - name: elasticsearch
            resources:
              limits:
                memory: 2Gi
    volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 2Gi
---
apiVersion: apm.k8s.elastic.co/v1alpha1
kind: ApmServer
metadata:
  name: apm-server-sample
spec:
  version: 7.1.0
  nodeCount: 1
  elasticsearchRef:
    name: "elasticsearch-sample"
---
apiVersion: kibana.k8s.elastic.co/v1alpha1
kind: Kibana
metadata:
  name: kibana-sample
spec:
  version: 7.1.0
```

```
  nodeCount: 1
  elasticsearchRef:
    name: "elasticsearch-sample"
```

As you can see:

1. We only have one node in your Elasticsearch cluster to save resources.
2. Elasticsearch, Kibana and APM server are all 7.1.0, so that we can showcase the upgrade process later.
3. We are using persistent storage for the cluster.

ECK is flexible in storage configuration. It supports the following type of storage:

- EmptyDir/HostPath: This is on by default. But be careful, this can lead to data loss! Only use this option for learning purposes
- Persistent Storage: Local PV, Cloud vendor specific
- Container Storage Interface (CSI): Future of Kubernetes storage

There are many other ways to customize your deployment, such as:

- Pod Template
- JVM heap size
- Node configuration
- Volume claim templates
- HTTP settings & TLS SANs
- Virtual memory
- Custom HTTP certificate
- Secure settings
- Custom configuration files and plugins
- Init containers for plugin downloads
- Update strategy
- Advanced Elasticsearch node scheduling

Deploy the stack with the following command. Your pod will be Pending if your cluster does not have enough resources.

```
kubectl apply -f apm_es_kibana.yaml
```

You can check the status of different components either using `kubectl` or the Kuberentes dashboard.

Check the state of the Elasticsearch, Kibana and APM server pods for all of them to get ready and green.

```
kubectl get elasticsearch,kibana,apmserver
```

Once all three components become green, your Elasticsearch cluster is up running, ready for data to come in! ECK performs many tasks for us. Before moving on, let's say a few things about what ECK has actually done for our cluster:

- **Security** is turned on by default for ECK. This means that all Elastic Stack resources deployed by ECK are secured by default. The built-in basic authentication user `elastic` was provisioned, and network traffic to, from, and within your Elasticsearch cluster are secured with TLS.
- **Certificates**: By default, a self-signed CA certificate is used for each cluster. Custom HTTPS certificate can be configured.
- **Default service exposure**: A ClusterIP Service is automatically created for your cluster, and Kibana. You can obviously configure them to be LoadBalancer type if needed.

With Elasticsearch, Kibana and APM server running, let's see it from Kibana.

Access Kibana

By default, Kibana is exposed as a service with a cluster IP accessible from within the cluster but is not exposed to the internet. However, you can use the following port forwarding command to make Kibana accessible from your laptop. Once again, you could have deployed a load balancer service for Kibana too.

```
kubectl port-forward service/kibana-sample-kb-http 5601
```

Before we can log into Kibana, we need to get the default `elastic` user's credentials. You can run `kubectl get secrets` and then inspect the secret to find the JSON path of the secret. Here is the correct secret and path for our case:

```
echo `kubectl get secret elasticsearch-sample-es-elastic-user
-o=jsonpath='{.data.elastic}' | base64 --decode`
```

The password for the `elastic` user should be displayed. Copy and save the password into a notepad.

Now, open a browser and connect to `https://localhost:5601`. Log into Kibana using the `elastic` user and the password you just retrieved. You should be taken to the homepage of Kibana. Our cluster is totally ready!

Next, we will deploy a simple application instrumented with Elastic APM into our Kubernetes cluster to generate some APM data. Exciting!

Deploy an application instrumented with Elastic APM

We instrumented an extremely simple Node.js application from this blog using the Elastic APM Node.js agent. The application is a static website that uses the Express framework. You can download the source code of this application from this GitHub repository.

To properly configure the APM agent, we need to obtain some information about the APM server we just deployed. Specifically, we will need:

- APM server token
- APM server URL

APM Server token

We can decode the token with the following command:

```
echo `kubectl get secret apm-server-sample-apm-token
-o=jsonpath='{.data.secret-token}' | base64 --decode`
```

The APM server token should be displayed. Copy and save the token into your notepad.

APM Server URL

The APM server is exposed as a Kuberenetes service. You can find the cluster IP address of the service using the following command:

```
kubectl get services | grep apm
```

The output should look something like:

```
apm-server-sample-apm-http     ClusterIP   10.0.41.22    <none>
8200/TCP   17m
```

So, the APM server URL in this case is:

```
https://10.0.41.22:8200
```

Notice that the protocol is `https`? The APM server is secured with TLS by default, which means that the agents need to connect to it using https.

Deploy an APM Application

Deployment information for the sample application is inside the deployment file `shark-node.yaml`. Make sure you edit the environment section with information you found out above. The APM agent gets its configuration information from these environment variables.

```
---
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: shark
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: shark-demo-deployment
  namespace: shark
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: shark-demo
    Spec:
      containers:
      - name: shark-demo
        image: adamquan/nodejs-image-demo
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
        env:
        - name: ELASTIC_APM_SECRET_TOKEN
          # Put the decoded token here in single quotes
          value: 'srj59kt4gbdm8xnxm2mdqd9d'
        - name: ELASTIC_APM_SERVER_URL
          # Put the URL for the APM service here.
          value: https://10.0.41.22:8200
        - name: ELASTIC_APM_SERVICE_NAME
          # Name that will appear in the APM UI
          value: "Shark"
        - name: ELASTIC_APM_VERIFY_SERVER_CERT
          value: "false"
```

Also notice that we are setting **`ELASTIC_APM_VERIFY_SERVER_CERT`** to `false`, because the cluster was created with self-signed certificate, server certificate validation has to be turned off. Otherwise, you will get errors.

Deploy the sample application into your Kubernetes cluster:

```
kubectl apply -f shark-node.yaml
```

Check the logs with the following command. If you see an error, then the application is not able to connect to the APM server. Double check your token and URL.

```
kubectl logs deployment/shark-demo-deployment -n shark
```

The output should be:

```
Example app listening on port 8080!
```

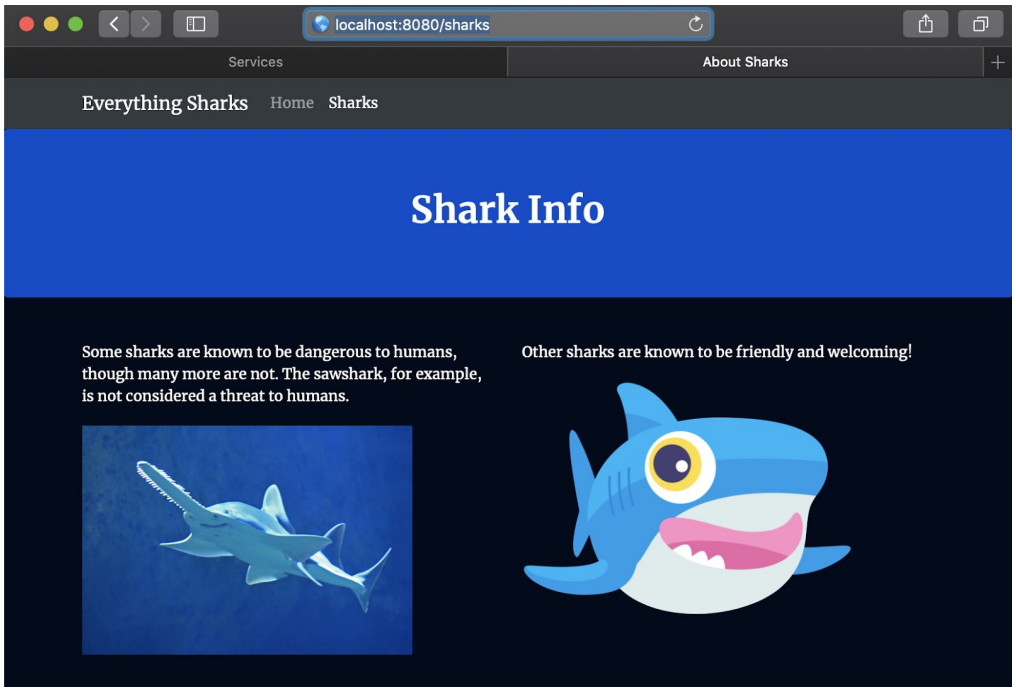
Let's expose the application with a service so that you can access it from your laptop with port forwarding:

```
kubectl expose deployment shark-demo-deployment -n shark
```

Forward the port for the application to your laptop so that we can access the application from your laptop, like what you have done for the Kibana service..

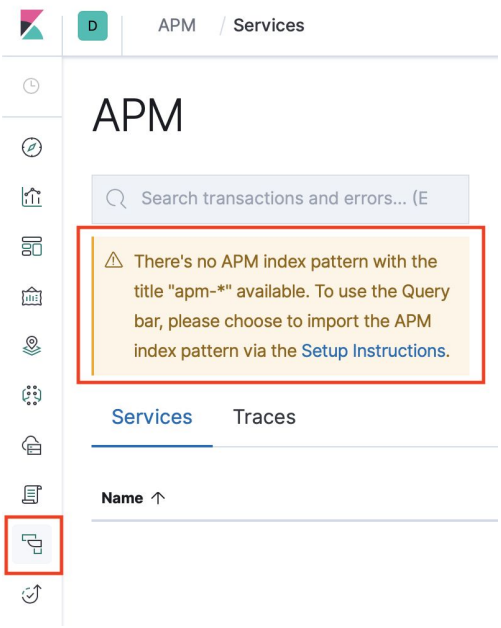
```
kubectl port-forward service/shark-demo-deployment 8080 -n shark
```

Navigate through the pages of the app: `http://localhost:8080`, and also type in an endpoint that does not exist: `http://localhost:8080/foo` to generate some errors. The application looks like this:

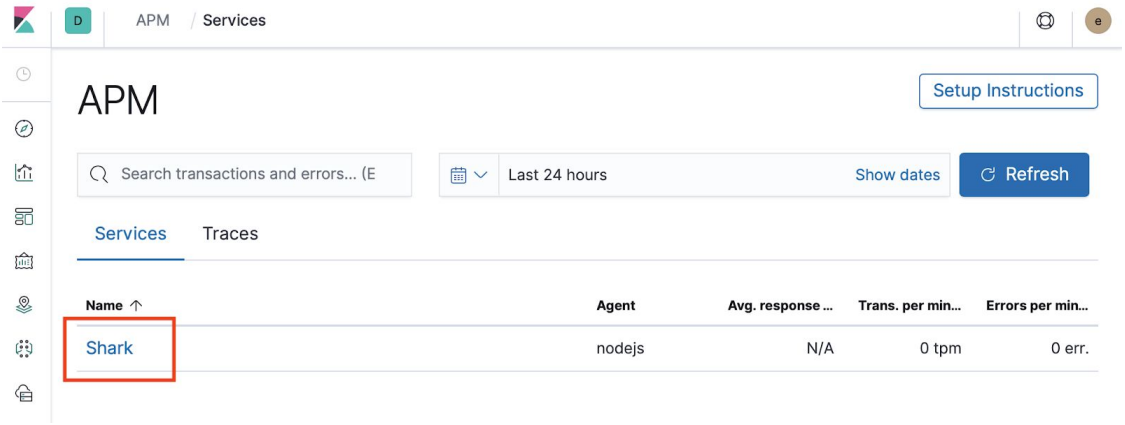


Look at APM tracing data

Open the APM UI. If you see a message like this, click on the "Setup Instructions" link and follow the instructions to setup APM. Make sure you click on "Load Kibana objects" at the end.



In the APM UI, You should see the “Shark” service listed. Click on the **Shark** service. Look at the transactions, check for errors, and look at the metrics.



Detailed APM walkthrough is beyond the scope of this blog. But, yeah, we got APM data flowing into an ECK managed Elasticsearch cluster! If you are interested in Elastic APM, a couple of Elastic blogs I wrote will help you:

- A Sip of Elastic RUM (Real User Monitoring)
- Distributed Tracing, OpenTracing and Elastic APM

Scaling and Upgrading

Scaling and upgrading your Elasticsearch cluster is as simple as modifying the deployment configuration and applying it with `kubectl`. Do keep in mind that you have to use the same

name for Elasticsearch, Kibana and APM server deployments. Otherwise, you will be creating a new deployment, instead of scaling or upgrading your existing ones.

Scaling

Modify your deployment manifest `apm_es_kibana.yaml` by changing the memory size of your elasticsearch node to 3GB from 2GB. The section you change is:

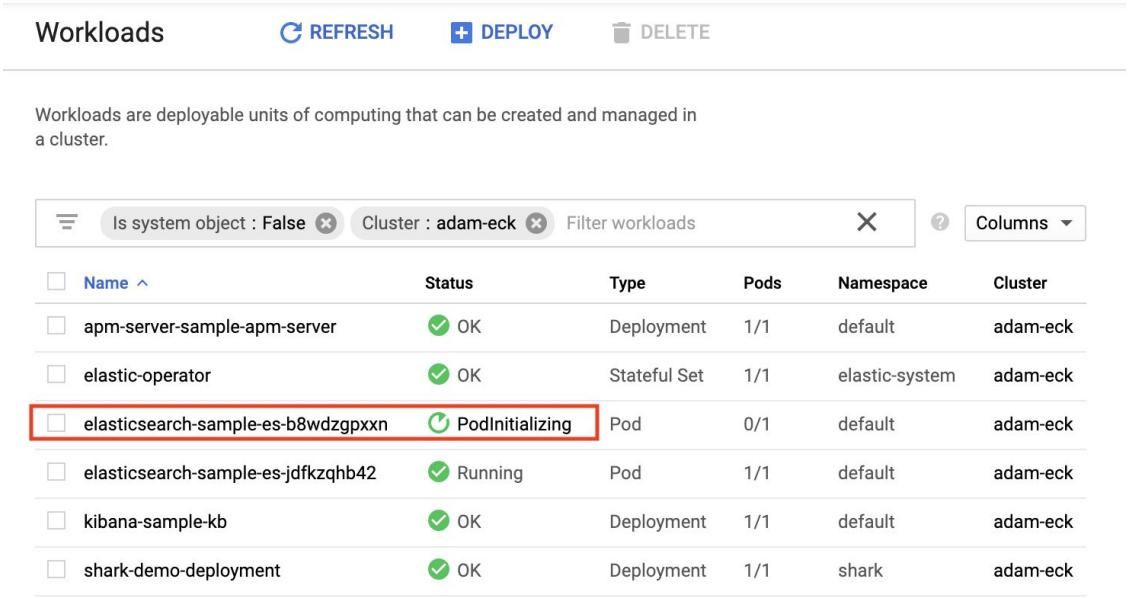
```
podTemplate:
      spec:
        containers:
          - name: elasticsearch
            resources:
              limits:
                memory: 3Gi
```

Save the file and scale your node with this command:

```
kubectl apply -f apm_es_kibana.yaml
```

Immediately switch to your Kubernetes console, you should see a new Elasticsearch pod being created. ECK is scaling your Elasticsearch node with zero downtime! In a few moments, you should see the new pod running and the original pod disappears. Data from the old pod was also seamlessly moved over.

You can obviously scale your cluster other ways, like adding more nodes to it. However way you do it, it's a simple `kubectl apply` command!



Upgrading

Now, let's upgrade the cluster to 7.2.0 by simply changing the version for Elasticsearch, Kibana and APM server from 7.1.0 to 7.2.0 inside the deployment manifest file `apm_es_kibana.yaml`. Save and apply the upgrade with the following command:

```
kubectl apply -f apm_es_kibana.yaml
```

Similarly, from the Kubernetes console, you can see that ECK is seamlessly upgrading Elasticsearch, Kibana and APM server with no downtime.

Once the upgrading process is done, check to make sure you are running 7.2.0 version of the Elastic stack. Convinced that ECK is pretty powerful by now? I hope you are.

Deploy Metricbeat

We really have done a lot so far. Let's also deploy Metricbeat into your Kubernetes cluster and have it send metrics data to your ECK managed Elasticsearch cluster. The main purpose is to show you how to get hold of the cluster certificate secret so that Metricbeat can connect to Elasticsearch securely. Recall that our Elasticsearch cluster has TLS enabled by default?

We need a couple of things for Metricbeat to connect to Elasticsearch securely via SSL:

1. The certificate
2. Connection information for Elasticsearch and Kibana: URL, user name, password

Create a Kubernetes ConfigMap with the cert

The cert is added to a secret. Extract the cert using the following command.

```
kubectl get secret elasticsearch-sample-es-http-ca-internal
-o=jsonpath='{.data.tls\.crt}' | base64 --decode
```

Edit the manifest `cert.yaml` and replace the sample with the decoded `tls.crt`. **Note**: Indent the cert like it is in the sample.

Create the ConfigMap with the following command:

```
kubectl create -f cert.yaml
```

Create secrets with connection information

Next, we will put all the information about our Elasticsearch cluster into a manifest file `secrets.yaml` so that we can create the secrets. Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.

Use the following command to list all the services so that you can find out the cluster IP of the Elasticsearch service and the Kibana service:

```
kubectl get services
```

The output should look like:

```
NAME                          TYPE      CLUSTER-IP  EXTERNAL-IP PORT(S)    AGE

apm-server-sample-apm-http    ClusterIP 10.0.41.22  <none>      8200/TCP   127m

elasticsearch-sample-es-http ClusterIP 10.0.33.116 <none>      9200/TCP   127m

kibana-sample-kb-http         ClusterIP 10.0.38.69  <none>      5601/TCP   127m

kubernetes                    ClusterIP 10.0.32.1   <none>      443/TCP    3d23h
```

In this case,

- Elasticsearch URL is: `https://10.0.33.116:9200`
- Kibana URL is `https://10.0.38.69:5601`

Base64 encrypt all connection information and put them into the file `secrets.yaml`. Make sure you use **your** Elasticsearch password and URLs.

- Elastic user name
  `echo -n 'elastic' | base64`
- Elastic user password
  `echo -n 'bhsd6djttf6ff8zn8njgdm8m' | base64`
- Elasticsearch URLs
  `echo -n ''["https://10.0.33.116:9200"]'' | base64`
- Kibana URL
  `echo -n 'https://10.0.38.69:5601' | base64`

Create the Kubernetes secret with the following command:

```
kubectl apply -f secrets.yaml -n kube-system
```

Deploy kube-state-metrics

`kube-state-metrics` is a simple service that listens to the Kubernetes API server and generates metrics about the state of the objects. It does not focus on the health of individual Kubernetes components, but rather on the health of the various objects inside, such as

deployments, nodes and pods. We will use `kube-state-metrics` to get metrics of these components.

Download and deploy `kube-state-metrics` with the following commands:

```
git clone https://github.com/kubernetes/kube-state-metrics.git
kube-state-metrics

kubectl create -f kube-state-metrics/kubernetes
```

Confirm it's deployed properly and running with the following command:

```
kubectl get pods --namespace=kube-system | grep kube-state
```

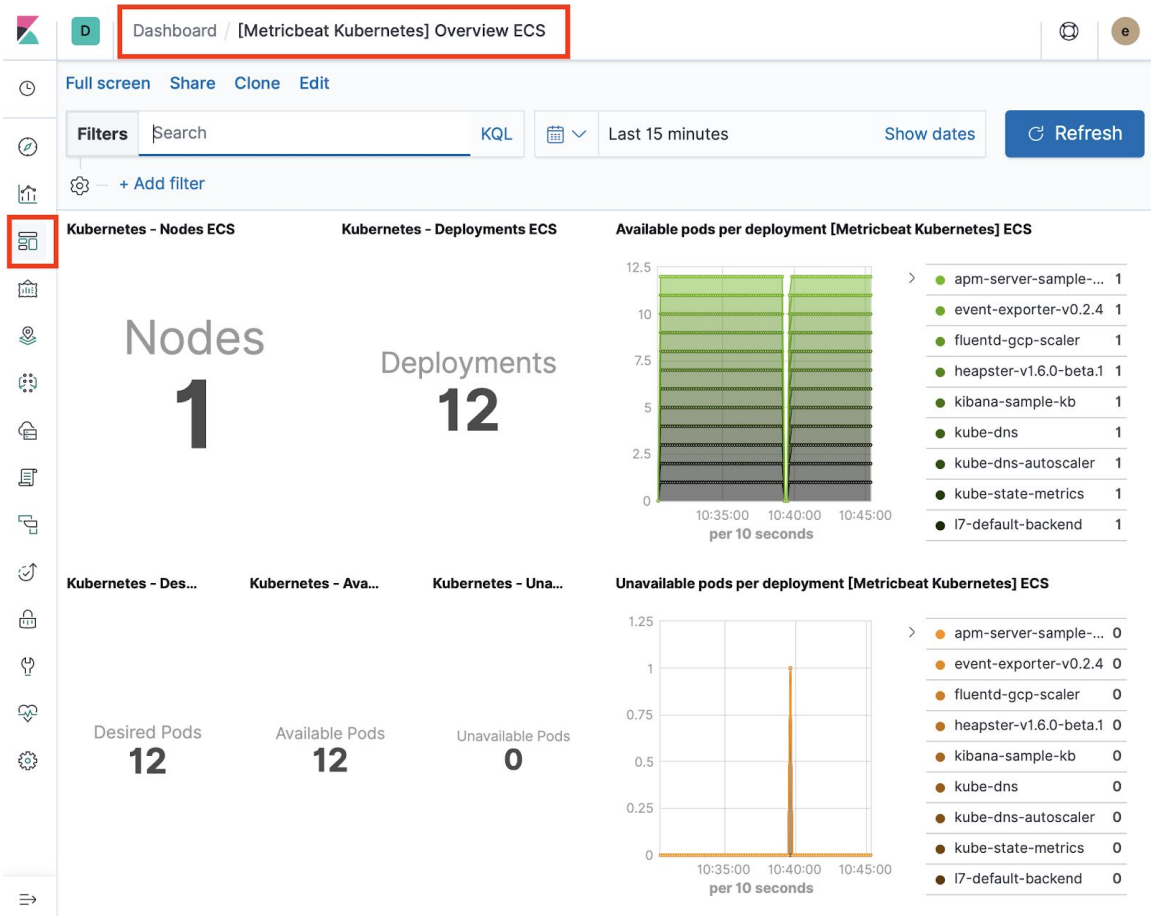
Deploy Metricbeat

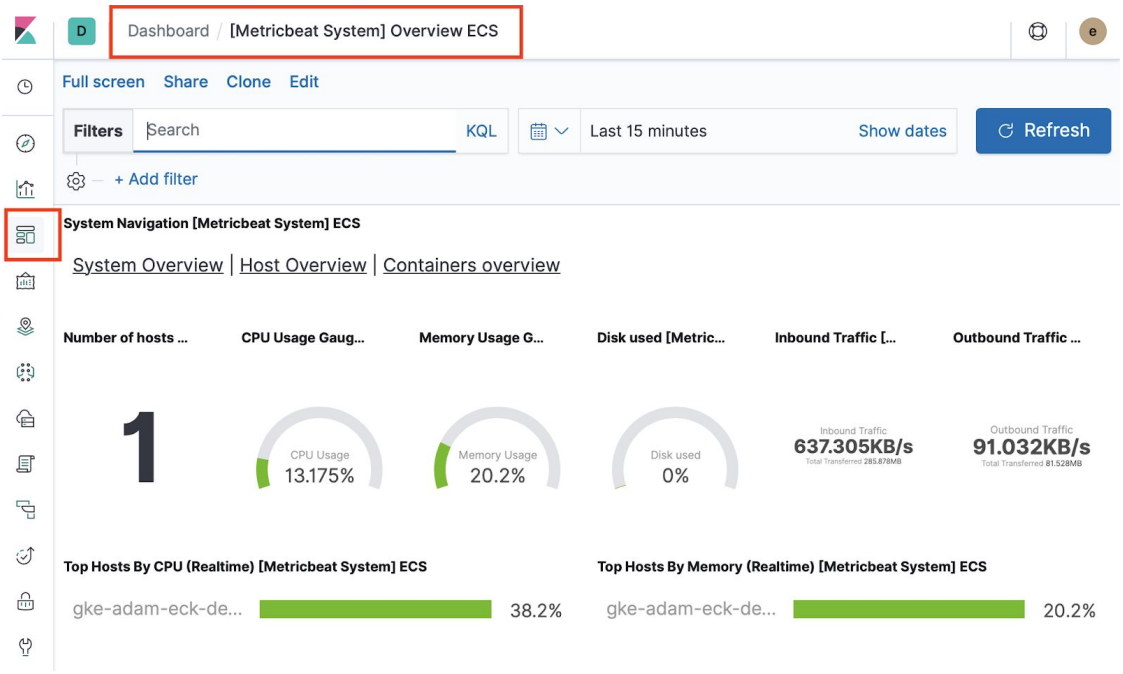
Now, we are ready to deploy Metricbeat! Since we are running a 7.2 Elasticsearch cluster, we will also deploy the same version of Metricbeat:

```
kubectl apply -f metricbeat-kubernetes.yaml
```

That's it! Metricbeat should be sending data to your ECK managed Elasticsearch cluster. If you are interested, take a look at the deployment manifest file and see how Metricbeat is deployed into Kubernetes.

Go to Kibana and check out a couple of out-of-the-box Metricbeat dashboards for your Kubernetes cluster, hosts and containers etc. Enjoy!


Dashboard / [Metricbeat Kubernetes] Overview ECS
Full screen
Share
Clone
Edit
Filters
Search
KQL
Last 15 minutes
Show dates
Refresh
+ Add filter
Kubernetes - Nodes ECS
Nodes
1
Kubernetes - Deployments ECS
Deployments
12
Available pods per deployment [Metricbeat Kubernetes] ECS
12.5
10
7.5
5
2.5
0
10:35:00
10:40:00
10:45:00
per 10 seconds
apm-server-sample-... 1
event-exporter-v0.2.4 1
fluentd-gcp-scaler 1
heapster-v1.6.0-beta.1 1
kibana-sample-kb 1
kube-dns 1
kube-dns-autoscaler 1
kube-state-metrics 1
l7-default-backend 1
Kubernetes - Des...
Desired Pods
12
Kubernetes - Ava...
Available Pods
12
Kubernetes - Una...
Unavailable Pods
0
Unavailable pods per deployment [Metricbeat Kubernetes] ECS
1.25
1
0.75
0.5
0.25
0
10:35:00
10:40:00
10:45:00
per 10 seconds
apm-server-sample-... 0
event-exporter-v0.2.4 0
fluentd-gcp-scaler 0
heapster-v1.6.0-beta.1 0
kibana-sample-kb 0
kube-dns 0
kube-dns-autoscaler 0
kube-state-metrics 0
l7-default-backend 0

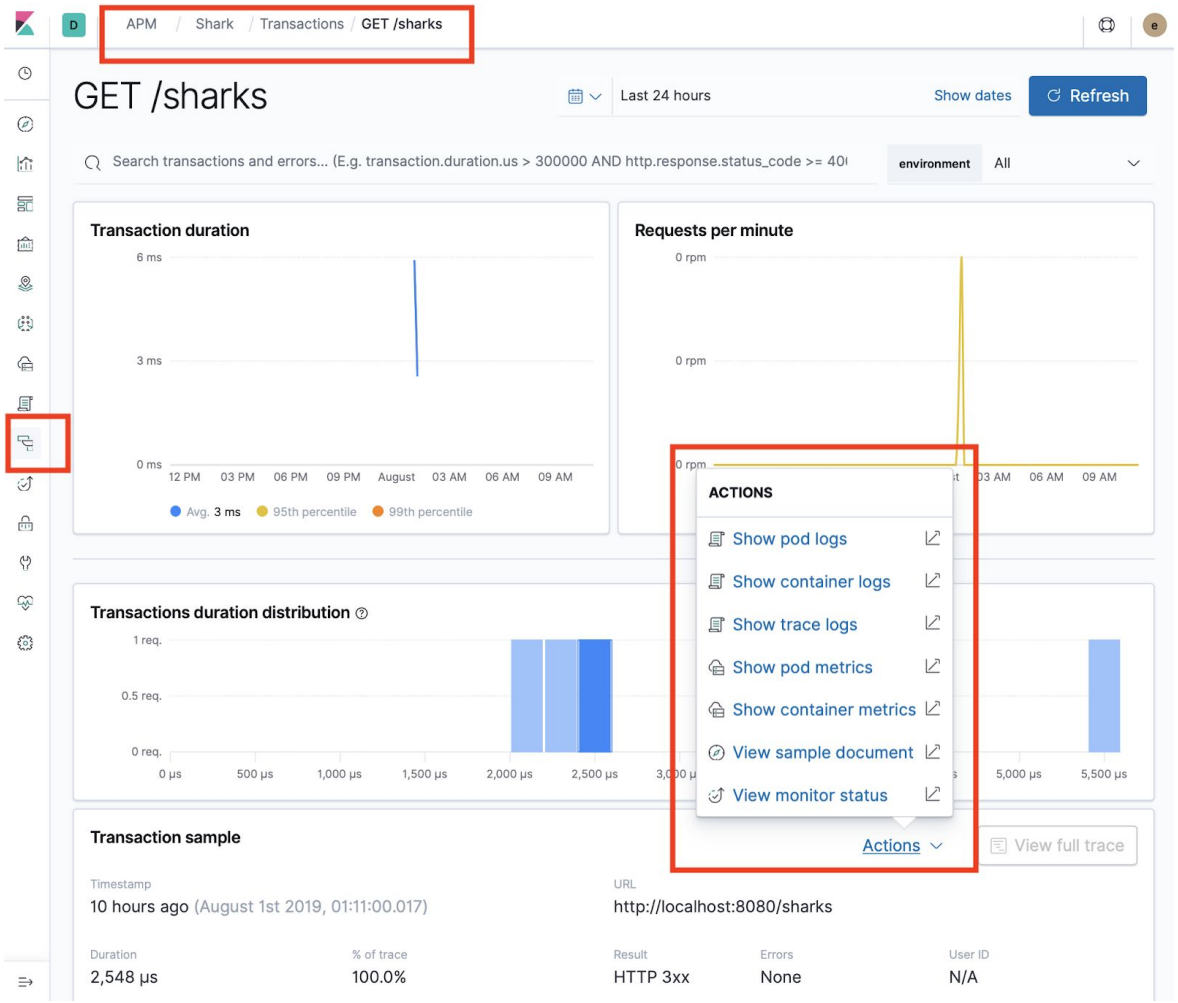

Dashboard / [Metricbeat System] Overview ECS
Full screen
Share
Clone
Edit
Filters
Search
KQL
Last 15 minutes
Show dates
Refresh
+ Add filter
System Navigation [Metricbeat System] ECS
System Overview | Host Overview | Containers overview
Number of hosts ...
1
CPU Usage Gaug...
CPU Usage
13.175%
Memory Usage G...
Memory Usage
20.2%
Disk used [Metric...
Disk used
0%
Inbound Traffic [...
Inbound Traffic
637.305KB/s
Outbound Traffic ...
Outbound Traffic
91.032KB/s
Top Hosts By CPU (Realtime) [Metricbeat System] ECS
gke-adam-eck-de...
38.2%
Top Hosts By Memory (Realtime) [Metricbeat System] ECS
gke-adam-eck-de...
20.2%

Bonus

With both Metricbeat and Elastic APM running inside ECK, let me give you a final bonus!

When Elastic APM is deployed inside Kubernetes, a lot of Kubernetes metadata are added by the kubernetes processor, like Kubernetes pod name, node name, namespace etc. These metadata can be used to correlate logs, metrics and APM tracing data inside the Elastic Stack. Super powerful.

Go back to the APM UI and look at the "**GET /sharks**" transaction page. Click on the "**Actions**" link, you should see something like the screenshot below. Notice you can navigate to pod and container logs and metrics directly from an APM trace? That is the power and convenience of the Elastic Stack! Imagine how convenient this is when you are trying to troubleshoot production issues!



Summary

Hopefully you enjoyed the quick tour of Elastic Cloud on Kubernetes and convinced that ECK is the way to run the Elastic Stack on Kubernetes. As we evolve the technology, there will be more productivity tools, like a web UI, to make a lot of the tasks easier. Enjoy!