Kubernetes Cluster Operations Documentation

Release 1.0.0

VMware, Inc.

Sep 27, 2019

CONTENTS:

CHAPTER

ONE

LAB 01: CLUSTER SETUP AND KUBECTL

1.1 Lab Goals

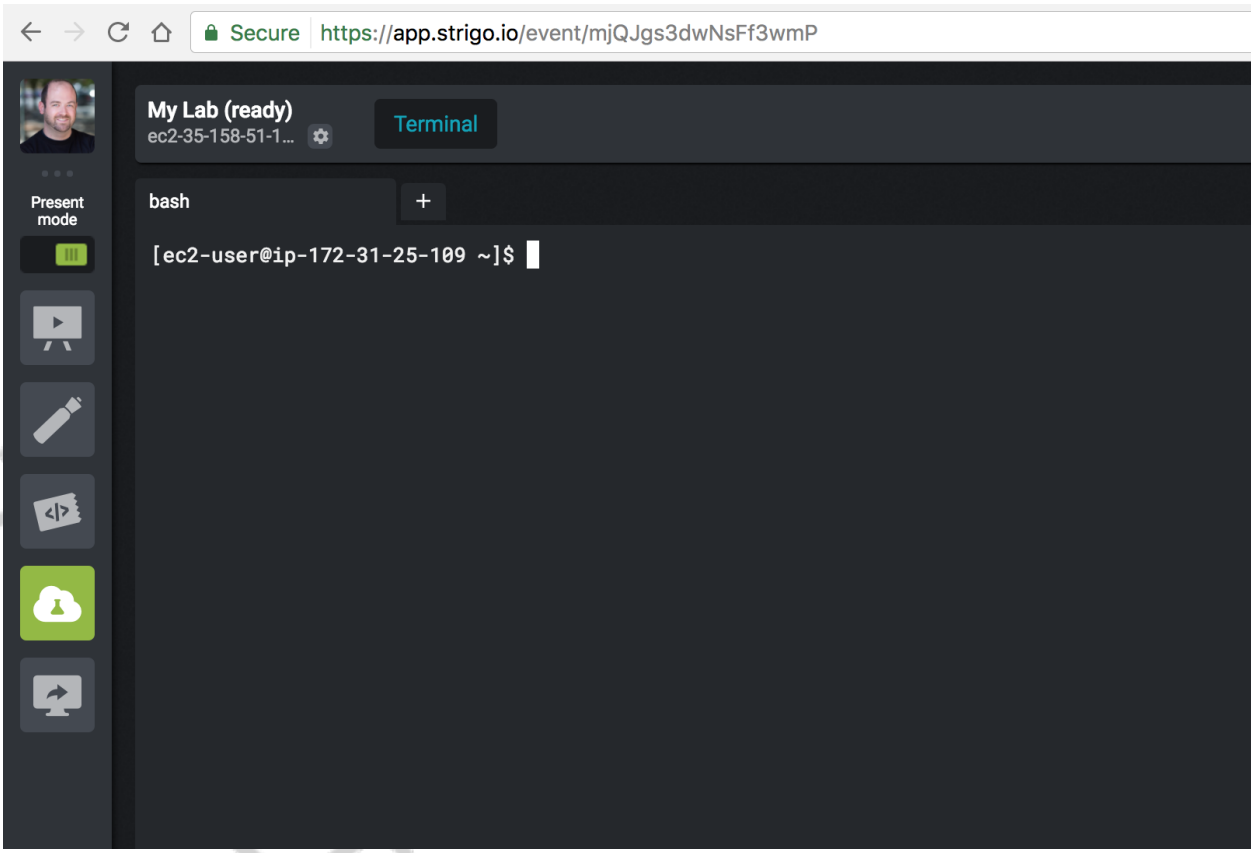
In this lab we work on setting up a small cluster using `kubeadm` By the end of the lab we will have all connectivity established and be familiar with different ways to manipulate kubectl output and settings.

The Strigo learning environment not only provides us with chat and slide sharing, but also provides remote shell access to a Linux machine that will act as our "client machine" and master node for the Kubernetes cluster.

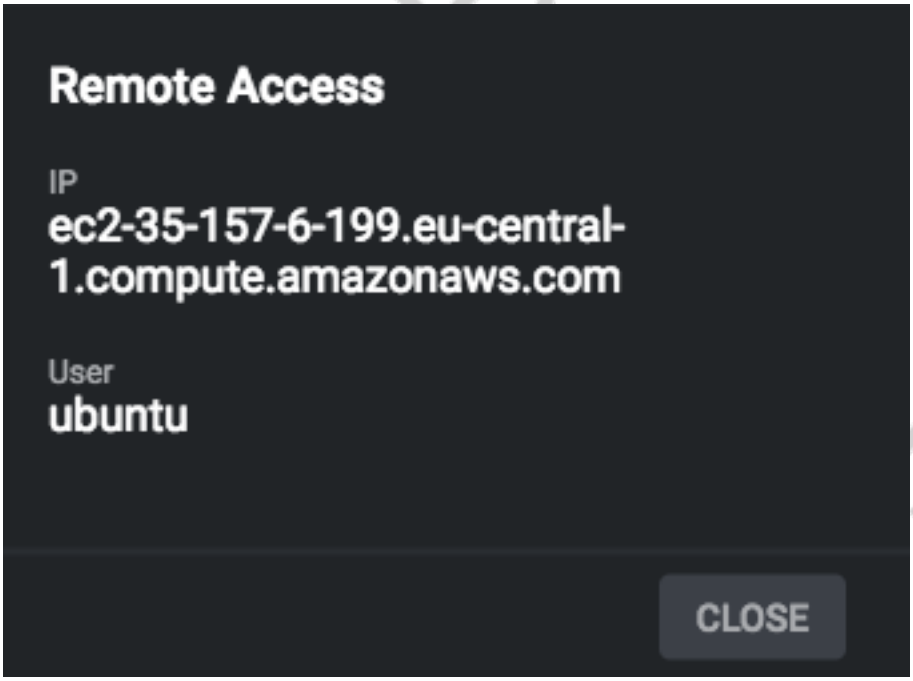
Optionally if you do not want to work from within the Strigo web terminal interface, you can SSH directly to the Client Machine.

1.2 Accessing your client machine

From the Strigo screen click the button on the left navigation that looks like a chemistry beaker. It is highlighted in green on the screenshot below. This will provide you terminal access to your client machine.



Feel free to use the built-in terminal in the browser and use as many tabs as you would like. However if you would prefer to SSH directly to the client machine from your laptop, this is possible as well. First grab the public hostname of the client machine by clicking the gear icon on the top of the screen near the "My Lab (ready)" text and then select "Connect from local" from the menu option. You will be presented with something similar to the following:

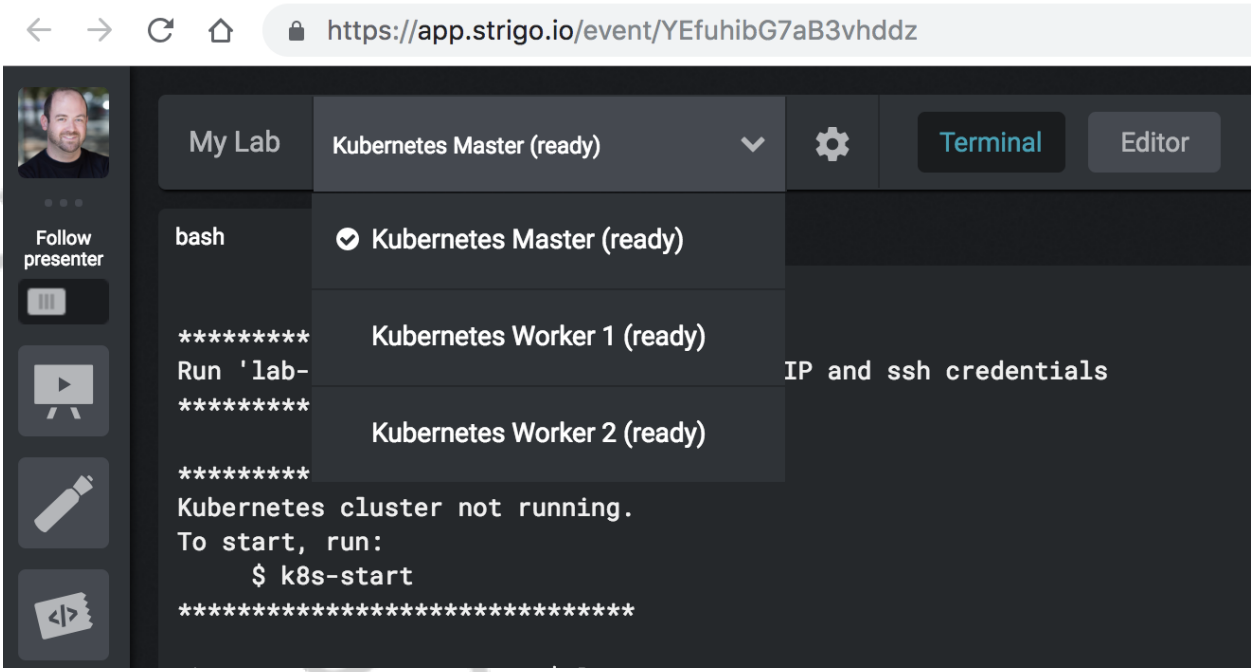


You can then use that information to ssh in the following manner:

```
ssh ubuntu@<public_hostname>
```

The password is H3pt10

There are three different virtual machines in your lab environment. You can switch between these machines by using the dropdown above the top of the terminal as can be seen below:



1.3 Initialize Cluster

In this lab we will setup a three node Kubernetes cluster comprised of one master node and two worker nodes. We'll start by initializing the master node.

```
EXTERNAL_IP=$(curl -s http://169.254.169.254/latest/meta-data/public-ipv4)

sudo kubeadm init --kubernetes-version 1.14.4 \
--pod-network-cidr=192.168.0.0/16 \
--apiserver-cert-extra-sans kubernetes,${EXTERNAL_IP}
```

After the master node is initialized, you should see output indicating a *kubeadm* command that we will run on both worker nodes to join them to the cluster:

```
*** EXAMPLE OUTPUT ****

ubuntu@ip-172-31-30-150:~$ sudo kubeadm init --kubernetes-version 1.14.4 \
         --pod-network-cidr=192.168.0.0/16 \
         --apiserver-cert-extra-sans kubernetes,${EXTERNAL_IP}

[init] Using Kubernetes version: v1.14.4
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
```

(continues on next page)

(continued from previous page)

```
...
Your Kubernetes control-plane has initialized successfully!
```

At the end of the output you will be prompted to perform a few steps.

First you will retrieve a kubeconfig file for the default administrator.

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Second you need to deploy a pod network to the cluster. There are many options for this, but we will deploy Calico for this lab.

```
kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/calico.yaml
```

Finally, we need to join the worker nodes to the cluster. Copy `kubeadm join...` command that is displayed after running `sudo kubeadm token create --print-join-command`.

```
sudo kubeadm token create --print-join-command
#this command creates a brand new token tha tyou may use for a node to join the␣
↪cluster
#even if previous tokens have expired
```

Then paste/run that command on the worker1.

Note: You will need to prefix the `kubeadm join` command with `sudo`.

```
*** EXAMPLE OUTPUT - DO NOT COPY ****

ubuntu@worker1:~$ sudo kubeadm join 172.31.37.134:6443 --token ftdqdf.
↪1slis6rlncc44oaa \
--discovery-token-ca-cert-hash␣
↪sha256:45cbc82819313ed957cac2d18aac417744a3b14862fd9b51a774d9f3227f2ea2

[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster..
...
This node has joined the cluster:
...
Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Copy the same join command and execute it on worker2

After joining both worker nodes, return to the master node's terminal and run the following command to verify that all three nodes (one master and two workers) are running:

```
kubectl get nodes
```

```
*** EXAMPLE OUTPUT ****

kubectl get nodes

NAME          STATUS    ROLES     AGE   VERSION
master        Ready     master    64s   v1.14.4
```

(continues on next page)

(continued from previous page)

```
worker1      Ready    <none>   32s   v1.14.4
worker2      Ready    <none>   24s   v1.14.4
```

1.4 Kubernetes Ingress

Next we'll setup an ingress controller to enable external access to the applications we'll deploy later.

lab-01-ingress.yaml

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: tcp-services
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: udp-services
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: nginx-ingress-serviceaccount
  namespace: ingress-nginx
```

(continues on next page)

(continued from previous page)

```
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: nginx-ingress-clusterrole
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - endpoints
      - nodes
      - pods
      - secrets
    verbs:
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - nodes
    verbs:
      - get
  - apiGroups:
      - ""
    resources:
      - services
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - "extensions"
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - create
      - patch
  - apiGroups:
      - "extensions"
    resources:
      - ingresses/status
```

(continues on next page)

(continued from previous page)

```
    verbs:
      - update

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  name: nginx-ingress-role
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
rules:
  - apiGroups:
      - ""
    resources:
      - configmaps
      - pods
      - secrets
      - namespaces
    verbs:
      - get
  - apiGroups:
      - ""
    resources:
      - configmaps
    resourceNames:
      # Defaults to "<election-id>-<ingress-class>"
      # Here: "<ingress-controller-leader>-<nginx>"
      # This has to be adapted if you change either parameter
      # when launching the nginx-ingress-controller.
      - "ingress-controller-leader-nginx"
    verbs:
      - get
      - update
  - apiGroups:
      - ""
    resources:
      - configmaps
    verbs:
      - create
  - apiGroups:
      - ""
    resources:
      - endpoints
    verbs:
      - get

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
  name: nginx-ingress-role-nisa-binding
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
```

(continues on next page)

(continued from previous page)

```
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: nginx-ingress-role
subjects:
  - kind: ServiceAccount
    name: nginx-ingress-serviceaccount
    namespace: ingress-nginx

---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: nginx-ingress-clusterrole-nisa-binding
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: nginx-ingress-clusterrole
subjects:
  - kind: ServiceAccount
    name: nginx-ingress-serviceaccount
    namespace: ingress-nginx

---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: nginx-ingress-controller
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/part-of: ingress-nginx
  template:
    metadata:
      labels:
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/part-of: ingress-nginx
      annotations:
        prometheus.io/port: "10254"
        prometheus.io/scrape: "true"
    spec:
      serviceAccountName: nginx-ingress-serviceaccount
      hostNetwork: true
      nodeSelector:
        kubernetes.io/hostname: master
      tolerations:
      - key: "node-role.kubernetes.io/master"
        operator: "Exists"
      containers:
```

(continues on next page)

(continued from previous page)

```
      - name: nginx-ingress-controller
        image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.22.0
        args:
          - /nginx-ingress-controller
          - --configmap=$(POD_NAMESPACE)/nginx-configuration
          - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-services
          - --udp-services-configmap=$(POD_NAMESPACE)/udp-services
          - --publish-service=$(POD_NAMESPACE)/ingress-nginx
          - --annotations-prefix=nginx.ingress.kubernetes.io
        securityContext:
          allowPrivilegeEscalation: true
          capabilities:
            drop:
              - ALL
            add:
              - NET_BIND_SERVICE
          # www-data -> 33
          runAsUser: 33
        env:
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
        ports:
          - name: http
            containerPort: 80
            hostPort: 80
          - name: https
            containerPort: 443
            hostPort: 443
        livenessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          initialDelaySeconds: 10
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
        readinessProbe:
          failureThreshold: 3
          httpGet:
            path: /healthz
            port: 10254
            scheme: HTTP
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 1
```

Let's download and apply the above yaml file, making sure to use master node's terminal and not one of the worker nodes that we were setting up above.

```
wget <url_of_yaml_above>

kubectl apply -f lab-01-ingress.yaml
```

1.5 Kubernetes Dashboard

Kubernetes has an optional web-based dashboard that you can deploy to your cluster. Let's set it up now.

Warning: We will use an insecure method of exposing the Kubernetes Dashboard to keep things simple for the lab. In production, you should deploy the dashboard with proper authenticaion and TLS configured.

1.5.1 Deploy the Dashboard

First deploy the dashboard

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/
↪deploy/recommended/kubernetes-dashboard.yaml
```

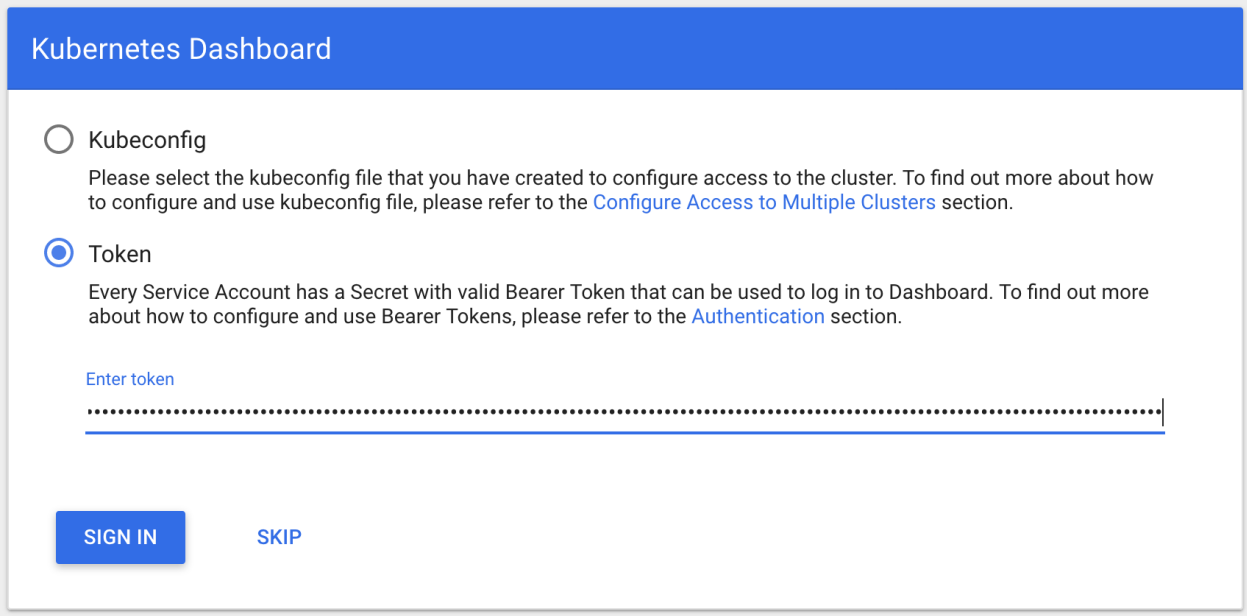
Then expose the dashboard using a NodePort service.

```
kubectl apply -f /opt/kubernetes-dashboard-public.yaml
```

To get the URL to use to connect to it run the following

```
k8s-dashboard
```

When prompted for credentials after browsing to the Dashbaord URL, select the token option and provide the token from `k8s-dashboard` output in the above command.



Warning: If you are using the terminal built into the Strigo web interface, the token may be split across multiple lines. Copy and paste the token into an editor, remove the line breaks, and then paste it into the Dashboard login.

For now, look around to ensure you can connect. Keep this url handy as we'll visit the dashboard in upcoming chapters when more objects have been populated.

1.6 kubectl

1.6.1 Controlling Output

Let's create a quick deployment for testing output and interaction:

```
kubectl create deployment nginx --image=nginx
kubectl scale --replicas=3 deployment nginx
```

Explore the various output of resources by trying the different output options. For example:

```
kubectl get deployment nginx -o [json|yaml|wide]
```

A useful output format is to filter the raw json output through jsonpath. This is handy for piping the output of a `kubectl get` command directly to other tools like sed, awk, grep, etc.

For example, let's figure out the command to see how we can list each namespace name by itself newline separated. Use this reference guide for help: https://kubernetes.io/docs/reference/kubectl/jsonpath

1.6.2 Pod Interaction

Now that we've played around a bit with kubectl and output, let's explore some of the commands that interact with pods and containers. The goal is the following:

- Forward the nginx port 80 from one of the pods onto localhost:8000 of your client machine
- Start a tail with follow of the logs on the same pod used above
- Using curl http://localhost:8000, cause some activity on the nginx server using the local port
- Run a remote command inside the container to for example get the current date and time

1.6.3 Cleaning Up

Let's delete and clean up everything in the default namespace. Ideally this can be done with a single kubectl command and without listing each resource individually. See if you can get the simplest command. You can verify everything is cleaned up by listing everything in the default namespace:

```
kubectl get all
```

1.7 Summary / Takeaways

There was quite a bit of setup in this lab, but now we have a multi node cluster that we can use to schedule pods and know a lot of details around how the `~/.kube/config` file is structured and managed. When all else fails, save a copy of it and if anything happens you can always revert to it.

Feel free to continue exploring kubectl output formatting and other global options, as getting fluent in kubectl is a fundamental skill for quick and efficient cluster management.

CHAPTER

TWO

LAB 02: LOGGING

2.1 Setup Logging Infrastructure

2.1.1 Lab Goals

In this lab we are going to explore and poke around to make sure we know how logs from pods are handled in Kubernetes. Then we'll set up an example centralized logging deployment to help collect all the logs from our cluster so they are searchable. Finally we'll simulate a pod that doesn't log to stdout and set it up so that those logs can still go into our logging setup.

2.1.2 Understanding Logging

First let's explore where to locate logs on each node. To do this we are going to use worker1 terminal from Strigo and run some commands.

Run `docker ps` and checkout all the running containers on the machine. When we are going "under the covers" here we are seeing all containers running regardless of the namespace they are in or whether they are internal Kubernetes system level containers or ones we have deployed.

Next, let's look at the kubelet that is running and look at the various options that are running, etc. Some of this should make sense with our working knowledge of the architecture of Kubernetes

```
sudo ps -ef | grep kubelet
```

Next let's view some of the logs of the kubelet itself:

```
sudo journalctl -xe -u kubelet
```

Now let's see where Kubernetes stores the logs from all the pods/containers that are started by the kubelet:

```
cd /var/log/containers

ls -lah
```

We can see that in this directory are softlinks to each active/live log file containing the stdout from each running pod. There is also a naming convention present here. If we want to look at one specifically. For example:

```
*** EXAMPLE OUTPUT ****

$ sudo tail -f kube-proxy-4fzpr_kube-system_kube-proxy-
↪8f5eaec948c720073158bfeb0b537502525206a8ead132aa8c92baa93c174c4e.log
```

(continues on next page)

(continued from previous page)

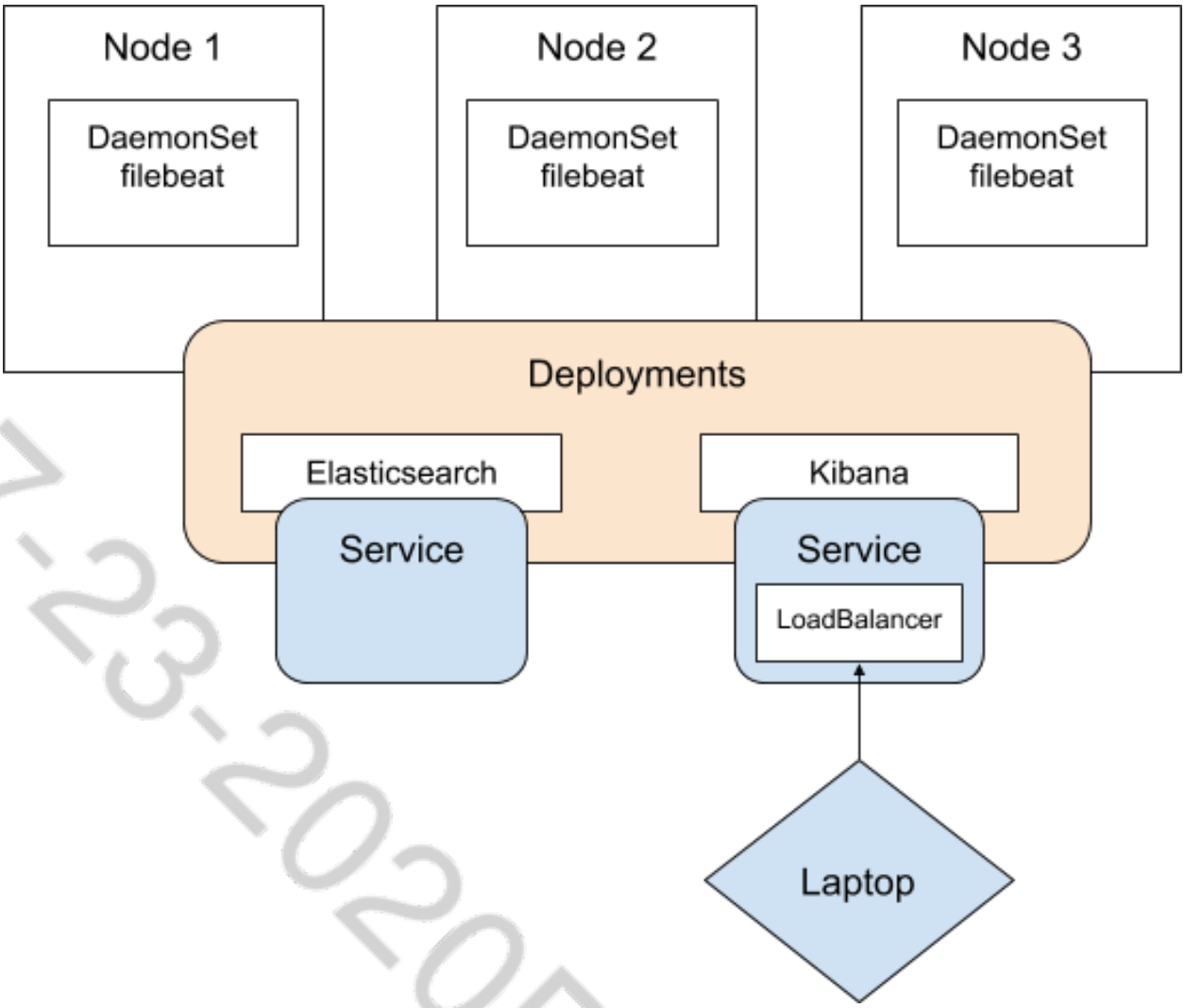
```
{"log":"I0225 20:07:42.484602       1 conntrack.go:52] Setting nf_conntrack_max to␣
↪131072\n","stream":"stderr","time":"2019-02-25T20:07:42.486615139Z"}
{"log":"I0225 20:07:42.485960       1 conntrack.go:83] Setting conntrack hashsize to␣
↪32768\n","stream":"stderr","time":"2019-02-25T20:07:42.486619727Z"}
{"log":"I0225 20:07:42.486136       1 conntrack.go:98] Set sysctl 'net/netfilter/nf_
↪conntrack_tcp_timeout_established' to 86400\n","stream":"stderr","time":"2019-02-
↪25T20:07:42.48662357Z"}
{"log":"I0225 20:07:42.486174       1 conntrack.go:98] Set sysctl 'net/netfilter/nf_
↪conntrack_tcp_timeout_close_wait' to 3600\n","stream":"stderr","time":"2019-02-
↪25T20:07:42.48662745Z"}
```

You may optionally run the commands above on worker2 to check results there.. Now that we have poked around and are familiar with how things are logged behind the scenes. With this knowledge in our hand, let's now put together a centralized logging solution to centralize and search all these pod logs so we don't have to manually log into machines on a regular basis :-)

2.1.3 Setting Up Logging Infrastructure

2.2 Architecture / Background

We're going to set up the following setup as an example of something you may use yourself for a centralized logging solution.



In the above we will deploy Filebeat as a DaemonSet on each node essentially tailing each of the logs from `/var/log/containers` and then shipping that data off to centralized storage using Elasticsearch. Finally there is a UI for Elasticsearch called Kibana that we will expose externally via an Ingress Controller and access it from our laptop.

Another very important part of this is we are placing all of these Services, Deployments, DaemonSets, etc. into a new separate namespace called kube-instrumentation. It's important that we don't pollute the reserved kube-system namespace. The idea here is that we could delete or remove the entire kube-instrumentation namespace and it would not affect any of the actual functioning of the cluster.

Before proceeding to the next step of actually deploying and accessing the above setup, please take some time to look over the supplied configuration to familiarize yourself with it. Ask questions to your instructor or peers if anything is not clear. The deployment YAML for this setup is located below:

`lab-02-logging.yaml`

```
---
kind: Namespace
apiVersion: v1
metadata:
  name: kube-instrumentation
  labels:
    name: kube-instrumentation
```

(continues on next page)

(continued from previous page)

```
---
kind: Service
apiVersion: v1
metadata:
  name: es-k8s-logging
  namespace: kube-instrumentation
  labels:
    app: elasticsearch
    function: logging
spec:
  selector:
    app: elasticsearch
    function: logging
  type: NodePort
  ports:
    - protocol: TCP
      port: 9200
      nodePort: 30920
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: es-k8s-logging
  namespace: kube-instrumentation
spec:
  replicas: 1
  selector:
    matchLabels:
      app: elasticsearch
      function: logging
  template:
    metadata:
      labels:
        app: elasticsearch
        function: logging
    spec:
      containers:
      - name: elasticsearch
        image: docker.elastic.co/elasticsearch/elasticsearch:5.5.1
        command: ["bin/elasticsearch"]
        args: ["-Ehttp.host=0.0.0.0", "-Etransport.host=127.0.0.1", "-Ecluster.
↪name=kubernetes", "-Ebootstrap.memory_lock=true"]
        env:
        - name: ES_JAVA_OPTS
          value: "-Xms512m -Xmx512m"
        ports:
        - containerPort: 9200
---
kind: Service
apiVersion: v1
metadata:
  name: kibana-k8s-logging
  namespace: kube-instrumentation
  labels:
    app: kibana
    function: logging
spec:
```

(continues on next page)

(continued from previous page)

```
  selector:
    app: kibana
    function: logging
  type: ClusterIP
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5601
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: instrumentation-ingress
  namespace: kube-instrumentation
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /kibana/?(.*)
        backend:
          serviceName: kibana-k8s-logging
          servicePort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kibana-k8s-logging
  namespace: kube-instrumentation
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kibana
      function: logging
  template:
    metadata:
      labels:
        app: kibana
        function: logging
    spec:
      containers:
      - name: kibana
        image: docker.elastic.co/kibana/kibana:5.5.1
        env:
          - name: ELASTICSEARCH_URL
            value: http://es-k8s-logging:9200
          - name: SERVER_BASEPATH
            value: /kibana
        ports:
        - containerPort: 5601
---
apiVersion: v1
kind: ConfigMap
metadata:
```

(continues on next page)

(continued from previous page)

```
  name: filebeat-config
  namespace: kube-instrumentation
data:
  filebeat.yaml: |
    name: pod-logs
    filebeat.prospectors:
    - input_type: log
      # This is not ideal as it assumes docker but currently is hardcoded in filebeat.
      # If /var/log/containers/*.log is used then we don't get any "kubernetes.*"␣
↪fields
      # See https://discuss.elastic.co/t/kubernetes-metadata/90865/16
      paths:
        - /var/lib/docker/containers/*/*.log
      symlinks: true
      json.keys_under_root: true
      json.add_error_key: true
      json.message_key: log

    processors:
    - add_cloud_metadata:
    - kubernetes:
        in_cluster: true
        namespace: kube-instrumentation

    output.elasticsearch:
      hosts: ['es-k8s-logging:9200']
      username: elastic
      password: changeme
---
apiVersion: extensions/v1
kind: DaemonSet
metadata:
  name: filebeat-k8s-logging
  namespace: kube-instrumentation
  labels:
    app: filebeat
    function: logging
spec:
  template:
    metadata:
      labels:
        app: filebeat
        function: logging
      name: filebeat
    spec:
      serviceAccountName: filebeat
      containers:
      - name: filebeat
        image: docker.elastic.co/beats/filebeat:6.0.0-alpha2
        command: ["filebeat"]
        args: ["-e", "-c", "/etc/filebeat/filebeat.yaml"]
        resources:
          limits:
            cpu: 50m
            memory: 50Mi
        securityContext:
          privileged: true
```

(continues on next page)

(continued from previous page)

```
          runAsUser: 0
        volumeMounts:
        - name: varlog
          mountPath: /var/log/containers
          readOnly: true
        - name: varlogpods
          mountPath: /var/log/pods
          readOnly: true
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
        - name: config-volume
          mountPath: /etc/filebeat
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log/containers
      - name: varlogpods
        hostPath:
          path: /var/log/pods
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers/
      - name: config-volume
        configMap:
          name: filebeat-config
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: filebeat
subjects:
- kind: ServiceAccount
  name: filebeat
  namespace: kube-instrumentation
roleRef:
  kind: ClusterRole
  name: filebeat
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: filebeat
  labels:
    app: filebeat
rules:
- apiGroups: [""] # "" indicates the core API group
  resources:
  - namespaces
  - pods
  verbs:
  - get
  - watch
  - list
---
```

(continues on next page)

(continued from previous page)

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: filebeat
  namespace: kube-instrumentation
  labels:
    app: filebeat
```

Note: There are many different implementations and products that could be used to implement the general pattern illustrated above. Additionally we are not paying proper attention to setup or a truly HA and scalable in Elasticsearch setup on Kubernetes (or outside of it) as it is out of scope for this lab.

2.3 Deployment of Logging Infrastructure

Let's download and apply the above yaml file, making sure to use your lab client machine and not one of the worker nodes that we were exploring above. We should see something like the following after applying the manifest.

```
# make sure we are back in the home directory
cd

wget <url_of_yaml_above>

kubectl apply -f lab-02-logging.yaml
```

Next make sure the pods have all come up successfully. You will either need to adjust the context to default to the kube-instrumentation namespace or manually specify it. Make sure you can view output something like the following:

```
*** EXAMPLE OUTPUT ****

$ kubectl get pods --namespace kube-instrumentation

NAME                                    READY     STATUS    RESTARTS   AGE
es-k8s-logging-76466469b6-prt5d         1/1       Running   0          1m
filebeat-k8s-logging-67xkz              1/1       Running   0          1m
filebeat-k8s-logging-bjrgm              1/1       Running   0          1m
kibana-k8s-logging-6cfd4fc578-pq5q4     1/1       Running   0          1m
```

All of our pods are up and running that make up our logging infrastructure.

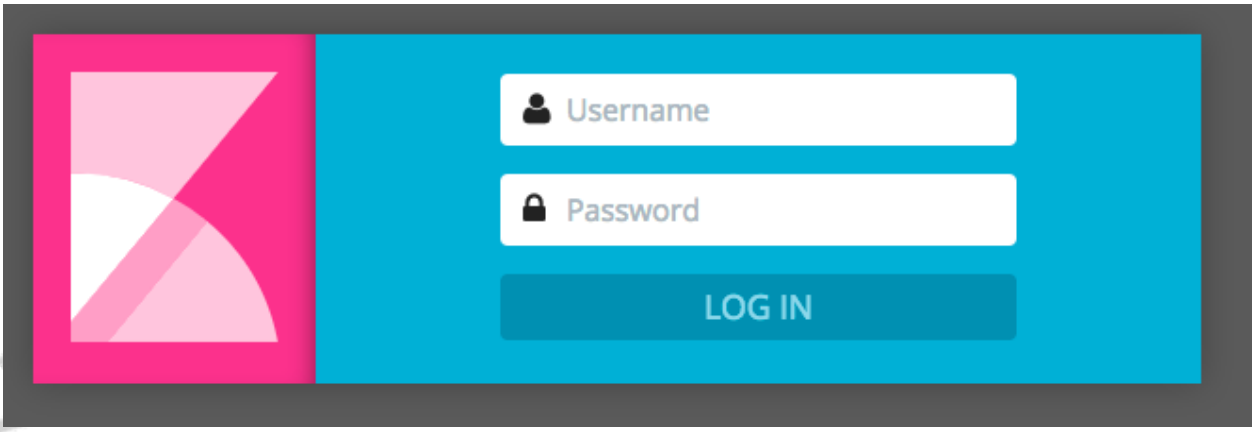
2.3.1 Access Kibana through the Ingress Controller

Note: Kibana will take several minutes to start up. You will initially see 4xx or 5xx errors when trying to connect to the web interface. You can optionally watch the Kibana logs to make sure it has fully started up `kubectl logs -n kube-instrumentation -l app=kibana`

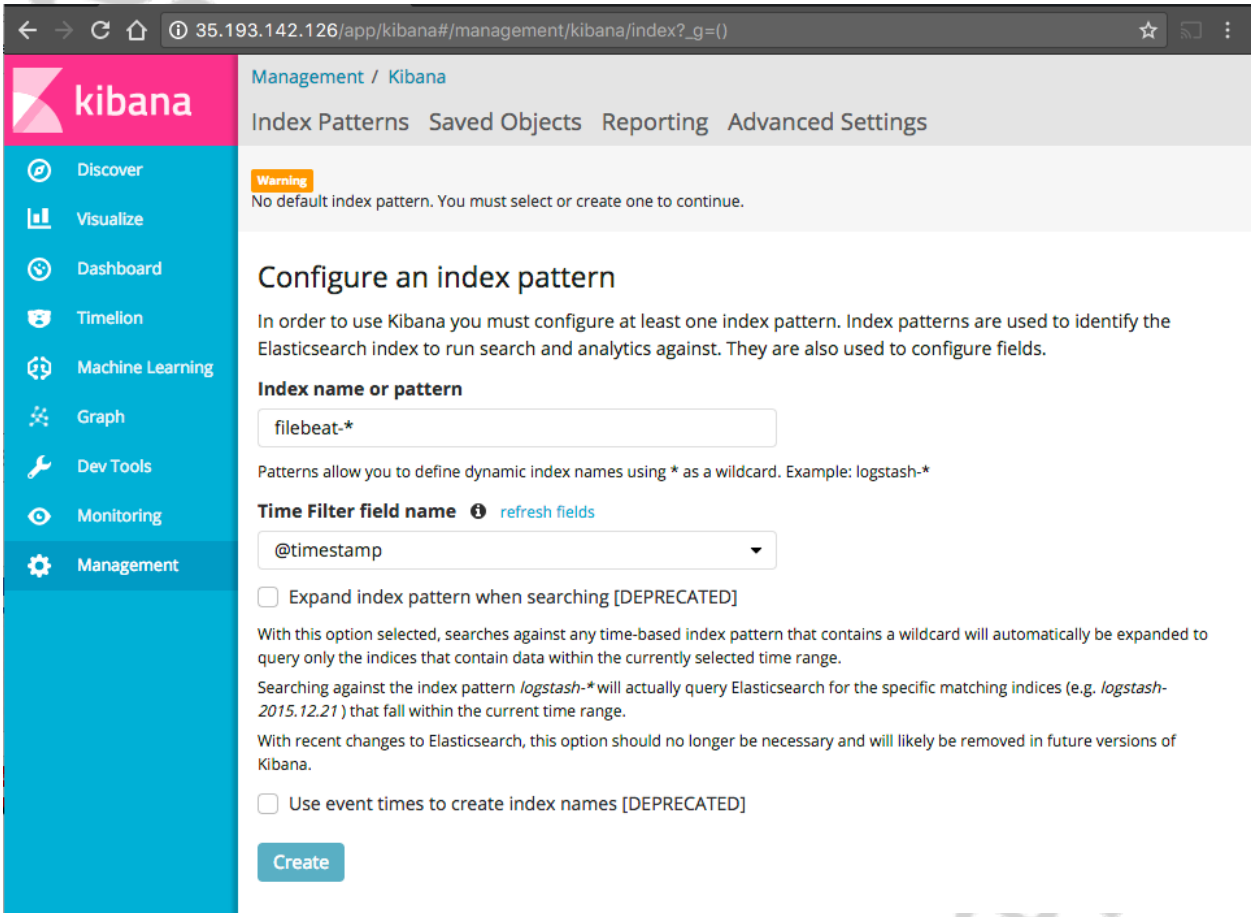
Access Kibana at the following URL: http://<PUBLIC-IP>/kibana

Note: To get the PUBLIC-IP of your host, run the command `lab-info` in your lab terminal

2.4 Using the Kibana UI

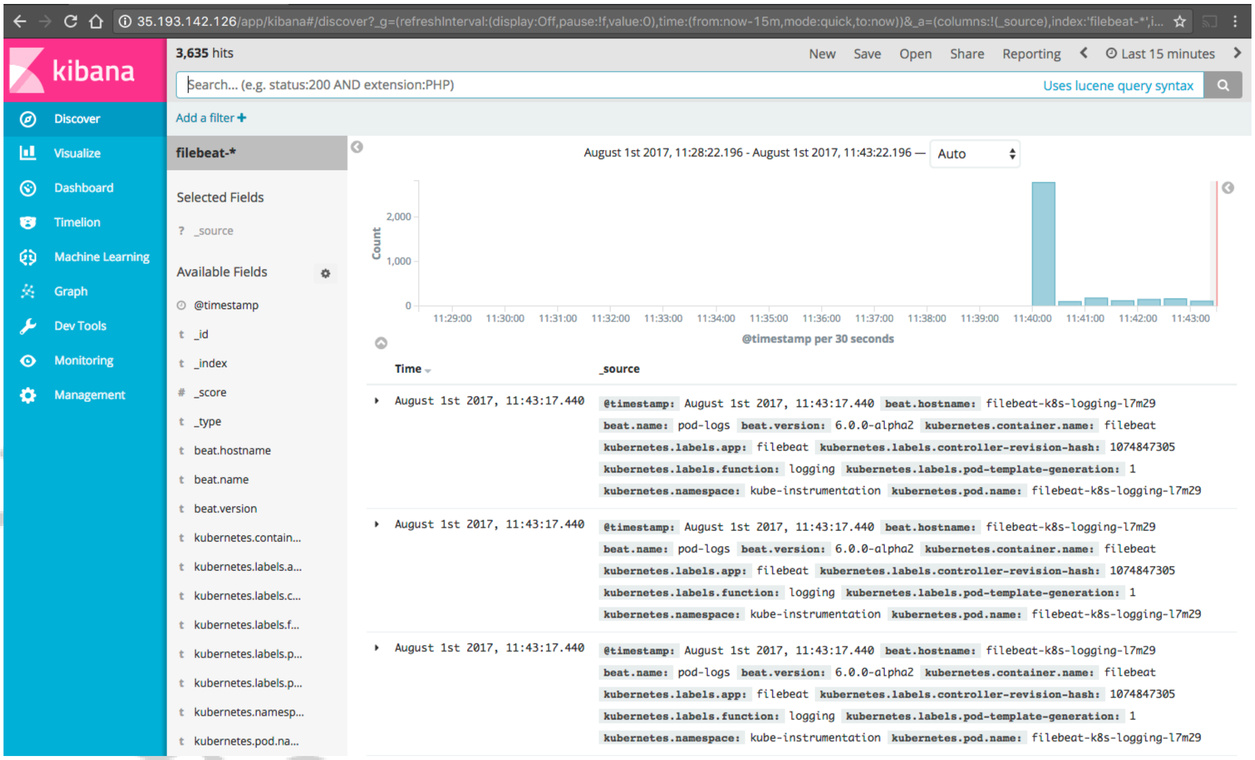


Login to the UI using the credentials `elastic/changeme`. Once logged in on the screen that shows up replace `logstash-*` with `filebeat-*` so that we can view all the logs ingested by filebeat and then click the Create button.

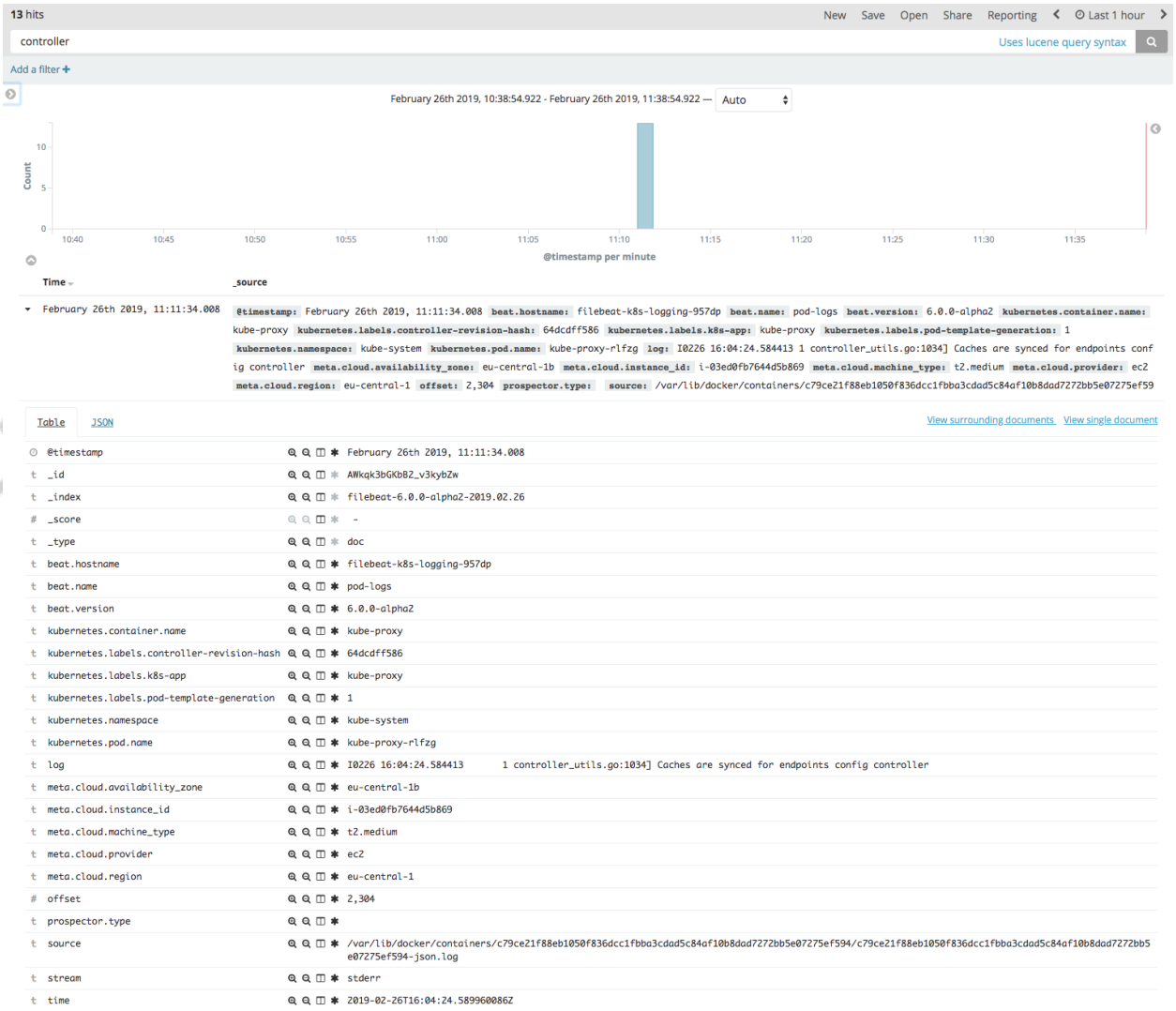


Now that the index pattern has been set, click the 

button on the left menu to look at some logs. We should something like the following:



Next let's search for `controller` and expand the row to look at all the fields. Notice we see not only the logs here from our pods, but we also have all the kubernetes data parsed out into separate fields:



Feel free to continue to explore and play with the Kibana interface to do searches and build visualizations and dashboards. We won't go into the details of how to use Kibana here but there are plenty of resources available for that if you would like to explore more.

2.4.1 Capturing Logs Not Using stdout

So far we have captured logs from images/containers that were designed to have all their logs go to stdout. However if we have an application that is writing to log files within the container, we can use the following approach to collect them. To do this we are going to use a pod with three containers. The main container is writing logs to two different files inside the container. The other two containers will tail those logs and then write them to stdout.

`streaming-sidecar.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
  namespace: default
spec:
```

(continues on next page)

(continued from previous page)

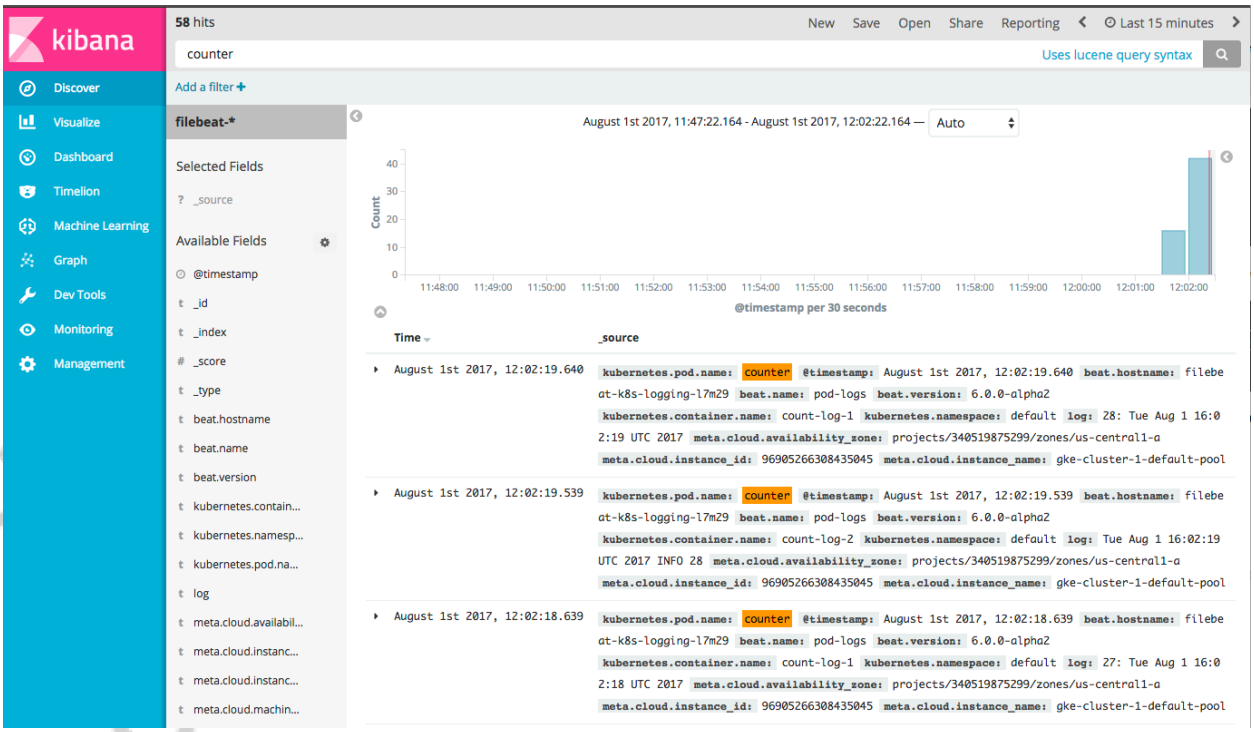
```
  containers:
  - name: count
    image: busybox
    args:
    - /bin/sh
    - -c
    - >
      i=0;
      while true;
      do
        echo "$i: $(date)" >> /var/log/1.log;
        echo "$(date) INFO $i" >> /var/log/2.log;
        i=$((i+1));
        sleep 1;
      done
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-1
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/1.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-2
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/2.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  volumes:
  - name: varlog
    emptyDir: {}
```

Download `streaming-sidecar.yaml` and then deploy it to your cluster.

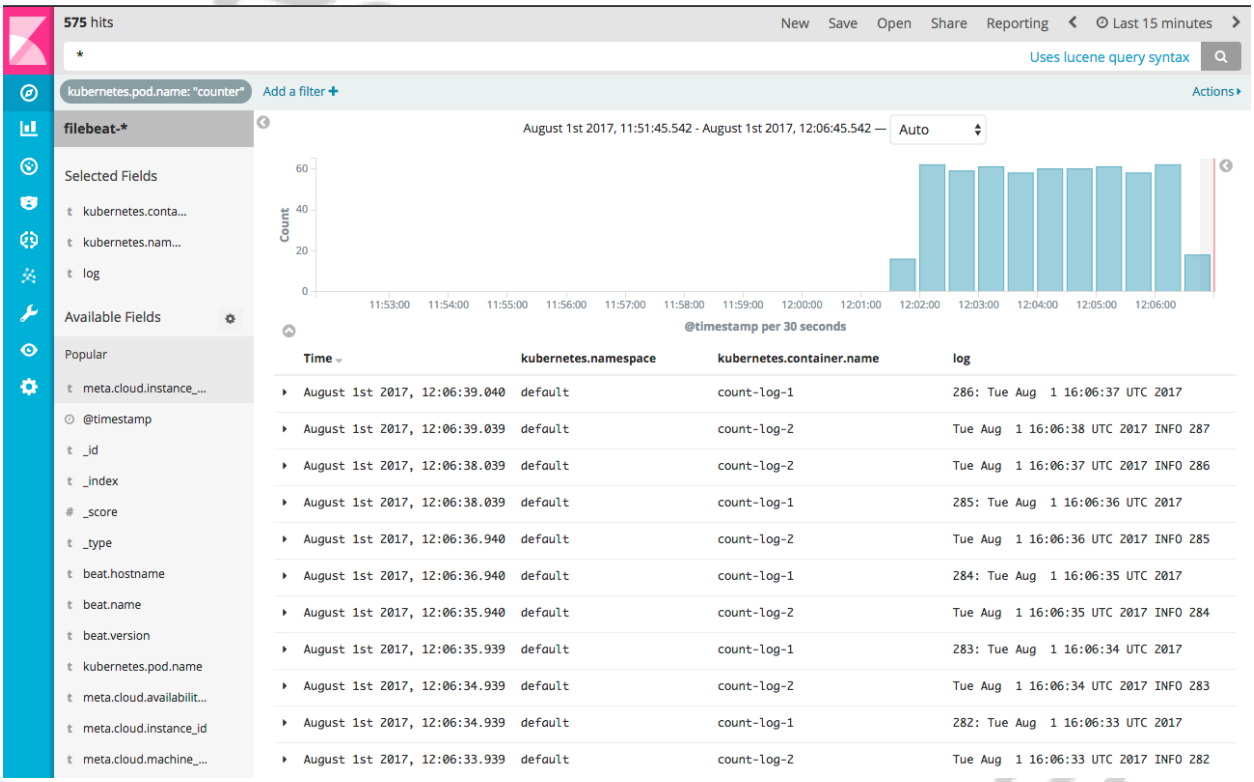
```
wget <url_of_yaml_above>

kubectl apply -f streaming-sidecar.yaml
```

Next head back over to our Kibana interface and do a search for counter and we now see the logs from the files:



We can even get a little fancy with Kibana and call out this information in easier to read ways. Feel free to continue exploring.



2.5 Enable and Explore Audit Logging

2.5.1 Enable Auditing in the API Server

2.6 Step 1: modify the API Server configuration to enable auditing

In order to enable auditing in our Kubernetes cluster, we need to modify the configuration of the API Server.

We need to get a default audit policy file to use so the API server knows what granularity to log items:

Note: The Kubernetes configuration files can only be modified by `root`. You will use `sudo` in the next step to switch to the `root` user.

```
sudo su -

cd /etc/kubernetes/pki/

curl -LO https://raw.githubusercontent.com/kubernetes/website/master/content/en/
↪examples/audit/audit-policy.yaml
```

Using either vi or nano, open the API Server manifest:

```
vi /etc/kubernetes/manifests/kube-apiserver.yaml
```

and add the following to the parameter list for the `kube-apiserver` command:

```
1 - --audit-log-path=/var/log/audit.log
2 - --audit-policy-file=/etc/kubernetes/pki/audit-policy.yaml
```

Setting these parameter will enable API Server audit logging.

2.7 Step 2: restart the API Server

After saving the file, restart the kubelet process which in turn will restart the API Server to pick up the changes we made to its configuration:

```
systemctl restart kubelet
```

With that done, exit from the `root` shell:

```
exit
```

It can take a few moments for the API Server to come back online after restarting the kubelet. Verify availability from the client node by running a `kubectl` command:

```
kubectl get nodes
```

A successful response from this command will indicate that the API Server is back online.

2.7.1 Access the cluster to generate some audit records

As illustrated above, the following command should successfully return output (HTTP status 200):

```
kubectl get nodes
```

2.7.2 Inspect the audit log

2.8 Step 1: retrieve the API Server's pod name

Let's retrieve the name of the API Server pod and store it in a shell local variable:

```
APISERVER_POD=$(kubectl get pods --namespace kube-system \
-l component=kube-apiserver -o jsonpath='{..metadata.name}')
```

2.9 Step 2: search the audit log for activity

As our final step, let's search the audit log on the API Server, looking for `kubernetes-admin` activity around the `nodes` resource:

```
kubectl exec -it $APISERVER_POD --namespace kube-system -- \
grep -A 1 \"kubernetes-admin\" /var/log/audit.log | grep "nodes"
```

Notice the detailed information about API, user, group, etc., along with HTTP return status (200 for success, 403 for failed authorization, etc.) on the following line.

2.9.1 Where to go next

For more information about auditing in Kubernetes, including experimental new features added in Kubernetes, see here:

https://kubernetes.io/docs/tasks/debug-application-cluster/audit/

2.9.2 Summary / Takeaways

We were able to:

- Look behind the scenes of how logging is done in Kubernetes
- Implement a log collection solution using the Node Logging Agent design pattern
- Create and explore a Sidecar Container approach to make logs as files inside containers visible to our log collection solution
- Enable audit logging and see what an audit log entry looks like

What tools are used to implement your logging solution is not the important part, but know how the logging works and putting in a solution that does collect all the logs from your cluster is the most important part and takeaway. We explored one here in hopes that it makes you feel comfortable with the concepts and to implement something on your own as well.

CHAPTER

THREE

LAB 03: MONITORING SETUP

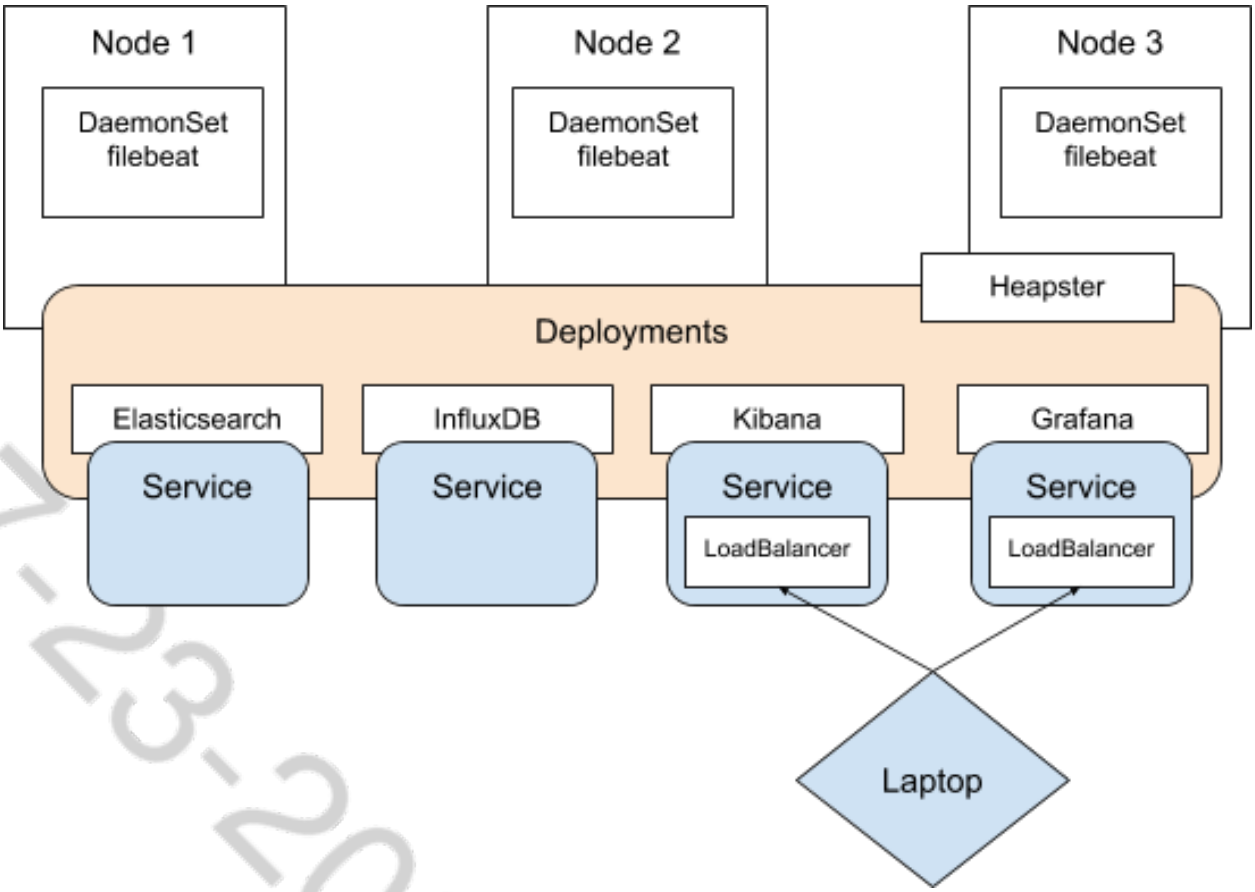
3.1 Lab Goals

In this lab we are going to set up some monitoring agents both inside the cluster (via kubernetes Deployments and DaemonSets) and also externally to make sure we have metrics that are external to the container engine. We will focus on the data collection and viewing but not on alerting.

3.2 Set Up Monitoring Infrastructure

3.2.1 Architecture

In this setup we are going to leverage the same kube-instrumentation namespace as well as our previously setup Elasticsearch deployment. In addition we will also add InfluxDB as another persistence store. Heapster will be deployed as a singleton Deployment in the cluster. It will collect and interpret various signals like compute resource usage, lifecycle events, etc, and exports cluster metrics from Kubernetes and then export those to a persistence store. In this case we will have Heapster write to both Elasticsearch and InfluxDB. Our rough setup once done will look like the following:



Before proceeding to the next step of actually deploying and accessing the above setup, please take some time to look over the supplied configuration to familiarize yourself with it. Ask questions to your instructor or peers if anything is not clear. The deployment YAML for this setup is located below:

`lab-03-monitoring.yaml`

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: monitoring-heapster
  namespace: kube-instrumentation
  labels:
    app: heapster
    function: monitoring
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: heapster-clusteradmin-role
  labels:
    app: heapster
    function: monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
```

(continues on next page)

(continued from previous page)

```
subjects:
- kind: ServiceAccount
  name: monitoring-heapster
  namespace: kube-instrumentation
---
apiVersion: v1
kind: Service
metadata:
  name: monitoring-heapster
  namespace: kube-instrumentation
  labels:
    app: heapster
    function: monitoring
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: heapster
spec:
  ports:
  - port: 80
    targetPort: 8082
  selector:
    app: heapster
    function: monitoring

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: monitoring-heapster
  namespace: kube-instrumentation
  labels:
    app: heapster
    function: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: heapster
      function: monitoring
  template:
    metadata:
      labels:
        app: heapster
        function: monitoring
    spec:
      serviceAccountName: monitoring-heapster
      containers:
      - name: heapster
        image: gcr.io/google_containers/heapster-amd64:v1.3.0
        imagePullPolicy: IfNotPresent
        command:
        - /heapster
        - --source=kubernetes.summary_api:https://kubernetes.default.svc?
↪kubeletHttps=true&kubeletPort=10250&insecure=true
        - --sink=influxdb:http://monitoring-influxdb.kube-instrumentation.svc:8086
        - --sink=elasticsearch:http://es-k8s-logging.kube-instrumentation.svc:9200?
↪sniff=false&healthCheck=false&esUserName=elastic&esUserSecret=changeme
---
```

(continues on next page)

(continued from previous page)

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: influxdb
    function: monitoring
  name: monitoring-influxdb
  namespace: kube-instrumentation
spec:
  ports:
  - port: 8086
    targetPort: 8086
  selector:
    app: influxdb
    function: monitoring
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: monitoring-influxdb
  namespace: kube-instrumentation
  labels:
    app: influxdb
    function: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: influxdb
      function: monitoring
  template:
    metadata:
     labels:
       app: influxdb
       function: monitoring
    spec:
      containers:
      - name: influxdb
        image: gcr.io/google_containers/heapster-influxdb-amd64:v1.1.1
        volumeMounts:
        - mountPath: /data
          name: influxdb-storage
      volumes:
      - name: influxdb-storage
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: monitoring-grafana
  labels:
    app: grafana
    function: monitoring
  namespace: kube-instrumentation
spec:
  type: ClusterIP
  ports:
```

(continues on next page)

(continued from previous page)

```
  - protocol: TCP
    port: 80
    targetPort: 3000
  selector:
    app: grafana
    function: monitoring
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: instrumentation-ingress
  namespace: kube-instrumentation
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /kibana/?(.*)
        backend:
          serviceName: kibana-k8s-logging
          servicePort: 80
      - path: /grafana/?(.*)
        backend:
          serviceName: monitoring-grafana
          servicePort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: monitoring-grafana
  namespace: kube-instrumentation
  labels:
    app: grafana
    function: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
      function: monitoring
  template:
    metadata:
      labels:
        app: grafana
        function: monitoring
    spec:
      containers:
      - name: grafana
        image: gcr.io/google_containers/heapster-grafana-amd64:v4.4.1
        ports:
        - containerPort: 3000
          protocol: TCP
        volumeMounts:
        - mountPath: /var
          name: grafana-storage
```

(continues on next page)

(continued from previous page)

```
        env:
        - name: INFLUXDB_HOST
          value: monitoring-influxdb
        - name: GF_SERVER_HTTP_PORT
          value: "3000"
          # The following env variables are required to make Grafana accessible via
          # the kubernetes api-server proxy. On production clusters, we recommend
          # removing these env variables, setup auth for grafana, and expose the␣
↪grafana
          # service using a LoadBalancer or a public IP.
        - name: GF_AUTH_BASIC_ENABLED
          value: "false"
        - name: GF_AUTH_ANONYMOUS_ENABLED
          value: "true"
        - name: GF_AUTH_ANONYMOUS_ORG_ROLE
          value: Admin
        - name: GF_INSTALL_PLUGINS
          value: raintank-kubernetes-app
        - name: GF_SERVER_ROOT_URL
          value: "/grafana/"
      volumes:
      - name: grafana-storage
        emptyDir: {}
```

Note: There are many different implementations and products that could be used to implement the general pattern illustrated above. Additionally we are not paying proper attention to setup or a truly HA and scalable in Elasticsearch or InfluxDB setup on Kubernetes (or outside of it) as it is out of scope for this lab.

3.2.2 Deployment

Let's download and apply the YAML file for the additional components:

```
wget <url_of_yaml_above>

kubectl apply -f lab-03-monitoring.yaml
```

Next make sure the pods have all come up successfully. You will either need to adjust the context to default to the kube-instrumentation namespace or manually specify it. Now run a command to make sure you can view output something like the following where we now have extra monitoring pods.

```
*** EXAMPLE OUTPUT ****

$ kubectl get pods --namespace kube-instrumentation

NAME                                    READY     STATUS    RESTARTS   AGE
es-k8s-logging-76466469b6-prt5d         1/1       Running   0          52m
filebeat-k8s-logging-67xkz              1/1       Running   0          52m
filebeat-k8s-logging-bjrgm              1/1       Running   0          52m
kibana-k8s-logging-6cfd4fc578-pq5q4     1/1       Running   0          52m
monitoring-grafana-5d7db879db-cll2j     1/1       Running   0          1m
monitoring-heapster-689d4bd5f7-vl2nq    1/1       Running   0          1m
monitoring-influxdb-546f955f87-f6p5g    1/1       Running   0          1m
```

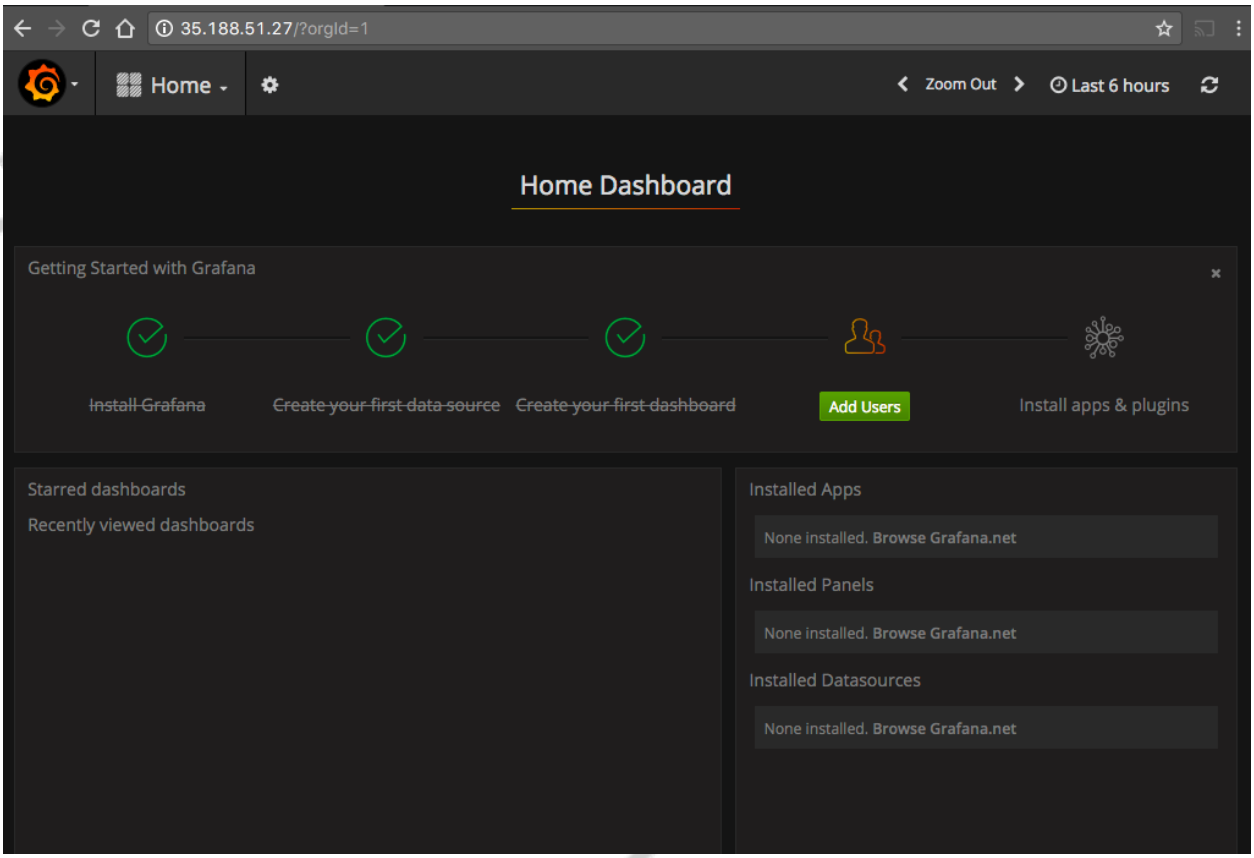
All of our pods are up and running that make up our monitoring infrastructure.

3.3 Access Grafana through the Ingress Controller

Access Grafana at the following URL: http://<PUBLIC-IP>/grafana

Note: To get the PUBLIC-IP of your host, run the command `lab-info` in your lab terminal

If successful after browsing to that URL we should see something like the following:



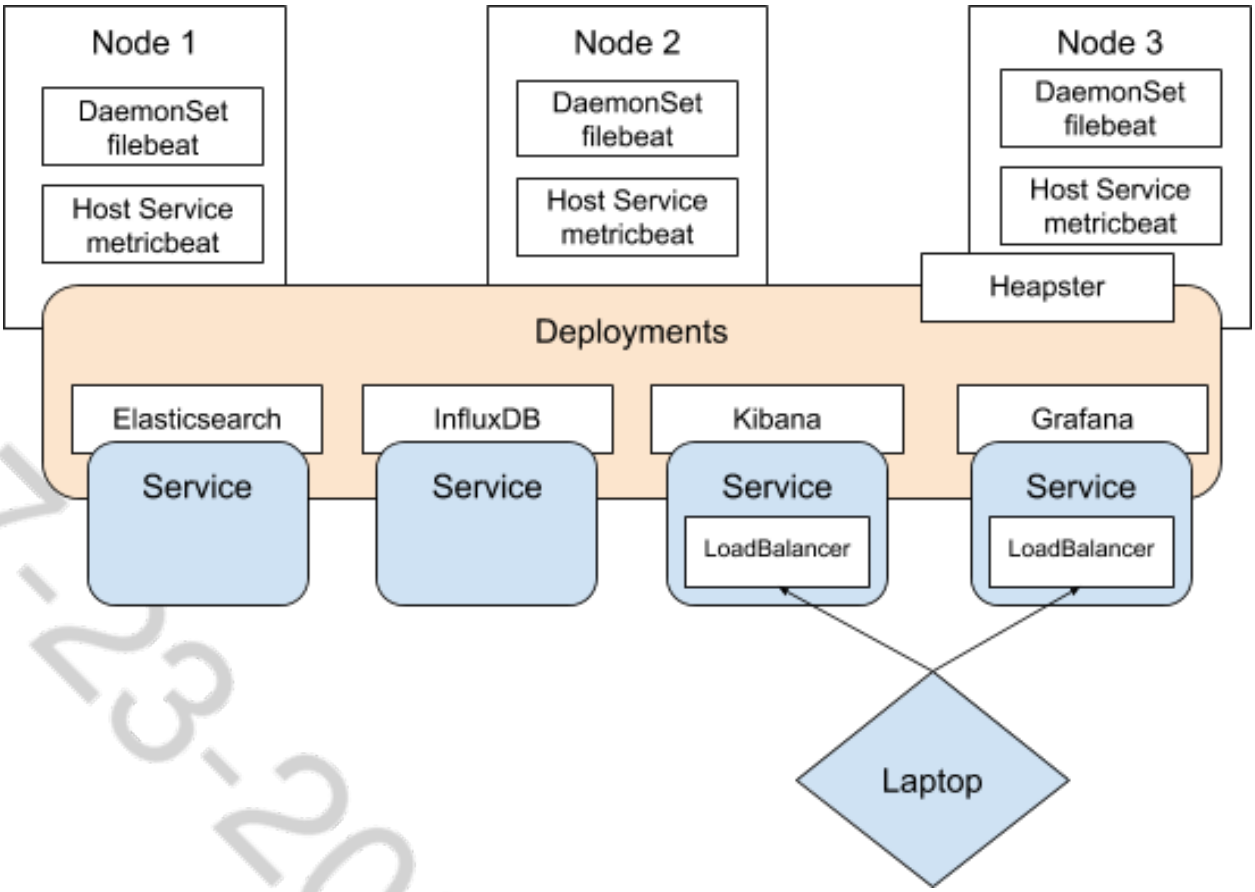
Click the Home button at the top and navigate to Cluster to bring up some of the metrics that we are recording and monitoring:

Note: If you don't see any data, wait a minute and refresh the page. The first batch of metrics are still being collected.

Explore the UI and to see what insights and information you can find.

3.4 External Monitoring

The monitoring setup so far has been internal to the Kubernetes cluster. However it is important to set up logging that is independent of the cluster, kubelet, container engine, etc. In this part of the lab we are going to log into a raw hosts backing your cluster and install an agent outside of the Kubernetes cluster report in metrics. Our rough architecture has now evolved to:



On worker1 from your Strigo lab environment, install an agent that is unaffected by kubectl and the chosen container engine. We will use Metricbeat for this and leverage our existing Elasticsearch install.

Note: We will cheat a little bit here. If you noticed, since we are assuming Elasticsearch is running within the Kubernetes cluster, we have modified the Service definition for it to state that it request a nodePort of 30920. Therefore in the configuration file of each metricbeat process running external to the cluster, we can assume it can access Elasticsearch via localhost:30920.

Once you connect to the terminal of worker1, let's download metricbeat, use a custom configuration file, and start it up.

`lab-03-metricbeat.yaml`

```
###################### Metricbeat Configuration Example #######################

# This file is an example configuration file highlighting only the most common
# options. The metricbeat.full.yml file from the same directory contains all the
# supported options with more comments. You can use it as a reference.
#
# You can find the full configuration reference here:
# https://www.elastic.co/guide/en/beats/metricbeat/index.html

#========================  Modules configuration ============================
metricbeat.modules:

#------------------------------- System Module -------------------------------
```

(continues on next page)

(continued from previous page)

```
- module: system
  enabled: true
  period: 10s
  metricsets:
    - cpu
    - load
    - memory
    #- core
    #- diskio
    - network
    - process_summary
    - process
    #- socket
  processes: ['.*']
  process.include_top_n:
    by_cpu: 5      # include top 5 processes by CPU
    by_memory: 5   # include top 5 processes by memory
- module: system
  enabled: true
  period: 1m
  metricsets:
    - filesystem
    - fsstat
  filters:
    - drop_event.when.regexp.mount_point: '^/(sys|cgroup|proc|dev|etc|host|lib)($|/)'
#------------------------------ kubernetes Module ------------------------------
# Node metrics, from kubelet:
- module: kubernetes
  enabled: true
  metricsets:
    - node
    - system
    - pod
    - container
    - volume
    - event
  period: 10s
  hosts: ["localhost:10255"]
  in_cluster: false
  kube_config: /etc/kubernetes/kubelet.conf

#------------------------------ Docker Module ------------------------------
- module: docker
  metricsets: ["container", "cpu", "diskio", "healthcheck", "info", "memory", "network
↪"]
  hosts: ["unix:///var/run/docker.sock"]
  enabled: true
  period: 10s

#==================== Elasticsearch template setting ==========================
setup.template.settings:
  index.number_of_shards: 1
  index.codec: best_compression
  #_source.enabled: false

#================================ General =====================================

```

(continues on next page)

(continued from previous page)

```
# The name of the shipper that publishes the network data. It can be used to group
# all the transactions sent by a single shipper in the web interface.
#name:

# The tags of the shipper are included in their own field with each
# transaction published.
#tags: ["service-X", "web-tier"]

# Optional fields that you can specify to add additional information to the
# output.
#fields:
#  env: staging

#================================ Outputs =====================================

# Configure what outputs to use when sending the data collected by the beat.
# Multiple outputs may be used.

#-------------------------- Elasticsearch output ------------------------------
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["127.0.0.1:30920"]

  # Optional protocol and basic auth credentials.
  #protocol: "https"
  username: "elastic"
  password: "changeme"

#----------------------------- Logstash output --------------------------------
#output.logstash:
  # The Logstash hosts
  #hosts: ["localhost:5044"]

  # Optional SSL. By default is off.
  # List of root certificates for HTTPS server verifications
  #ssl.certificate_authorities: ["/etc/pki/root/ca.pem"]

  # Certificate for SSL client authentication
  #ssl.certificate: "/etc/pki/client/cert.pem"

  # Client Certificate Key
  #ssl.key: "/etc/pki/client/cert.key"

#================================ Logging =====================================

# Sets log level. The default log level is info.
# Available log levels are: critical, error, warning, info, debug
#logging.level: debug

# At debug level, you can selectively enable logging only for some components.
# To enable all selectors use ["*"]. Examples of other selectors are "beat",
# "publish", "service".
#logging.selectors: ["*"]
processors:
- add_cloud_metadata: ~

setup.dashboards.enabled: true
```

Notice to replace the `wget` URL below with the one of the yaml file linked above.

```
sudo su -
wget https://artifacts.elastic.co/downloads/beats/metricbeat/metricbeat-6.0.0-alpha2-
↪linux-x86_64.tar.gz
tar -xzf metricbeat-6.0.0-alpha2-linux-x86_64.tar.gz
cd metricbeat-6.0.0-alpha2-linux-x86_64
rm metricbeat.yml
wget <url_of_yaml_above>
cp lab-03-metricbeat.yaml metricbeat.yml
./metricbeat -setup &
```

Enable kubelet to expose read-only port 10255 so that metricbeat can access

```
echo "KUBELET_EXTRA_ARGS= \"--read-only-port=10255\"" > /etc/default/kubelet
systemctl restart kubelet
sleep 10
exit
```

This will set everything up to report extra information to our Elasticsearch instance as well as installing some pre-made dashboards.

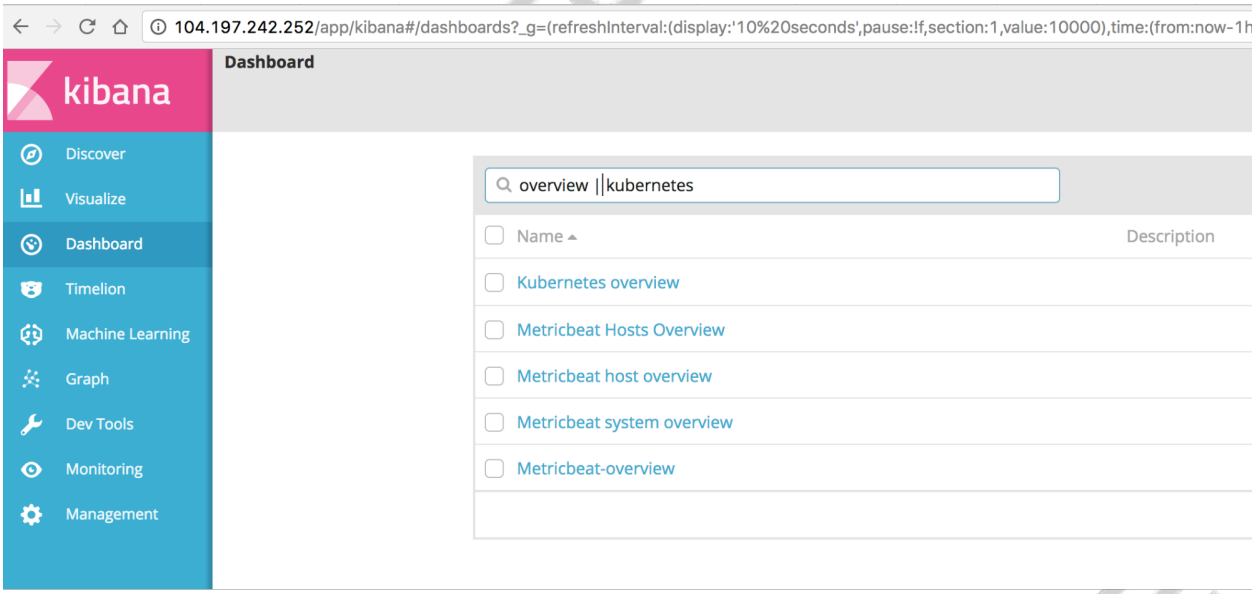
Note:

- This running of the process does not include a script for starting/stopping at boot time of the machine. That would be recommended in an official setup
- We're playing a little trick here. Since we are running Elasticsearch inside the cluster, we expose it with a pre-determined nodePort value so that way our config can be static and talk to localhost to connect to Elasticsearch

---

Next let's go over to Kibana and view some of the extra data being sent in. Switch to your Kibana browser tab from earlier and perform a hard refresh on the browser. Then click the Dashboard button from the left nav to explore some pre-made dashboards.

You can search the existing ones to get a feel for what information is presented. Suggestions would be searching for dashboards that have kubernetes or overview in the title.



Take some time and explore around with this hopefully helpful dashboards.

Additionally, set up metricbeat on worker2 as well so that we can see all hosts are being monitored.

3.5 Summary / Takeaways

In this lab we covered setting up an example monitoring infrastructure for your cluster that is cloud agnostic. The important takeaways are:

- Your monitoring solution may different from what is presented here and that is OK. We want to make sure you understand the best practices and how things work behind the scenes along with one practical example.
- It is important to have agents monitoring your cluster that are external to the Kubernetes infrastructure. This way in case the docker engine or similar experience problems, you still have some monitoring data to come through and aid in your diagnostic

CHAPTER

FOUR

LAB 04: CLUSTER TROUBLESHOOTING

This lab is dedicated to cluster troubleshooting, it assumes you have already ruled out the application as the root cause of the problem. There have a lab dedicated to troubleshooting applications in the Foundations course, this lab will strictly focus on things that may go wrong with a cluster from a cluster administrator's point of view.

Note: Since this lab will have commands to be executed of different nodes, we added (master), (worker#) before commands in the lab steps to reduce confusion.

4.1 Listing your cluster

The first thing to debug in your cluster is if your nodes are all registered correctly.

```
(master) kubectl get nodes
```

Use this to verify that your nodes are in the ready state.

```
(master) kubectl describe nodes worker1
```

Use this to review details about a specific node. Things like cpu, memory and disk pressure can help determine bottlenecks and scaling requirements.

4.2 Looking at logs

Note: Here are the commands to view logs on the lab cluster created by kubeadm, the commands to view logs may be different on other clusters depending on how they were set up.

4.2.1 Step 01: Master node logs

Looking at the api-server log.

```
(master) sudo tail -f /var/log/containers/kube-apiserver*
```

Looking at the scheduler log.

```
(master) sudo tail -f /var/log/containers/kube-scheduler*
```

Looking at the controller manager log.

```
(master) sudo tail -f /var/log/containers/kube-controller*
```

4.2.2 Step 02: Worker nodes logs

Note: Since our master node is also a worker, the commands below can be run on the master as well.

Looking at the kube-proxy log.

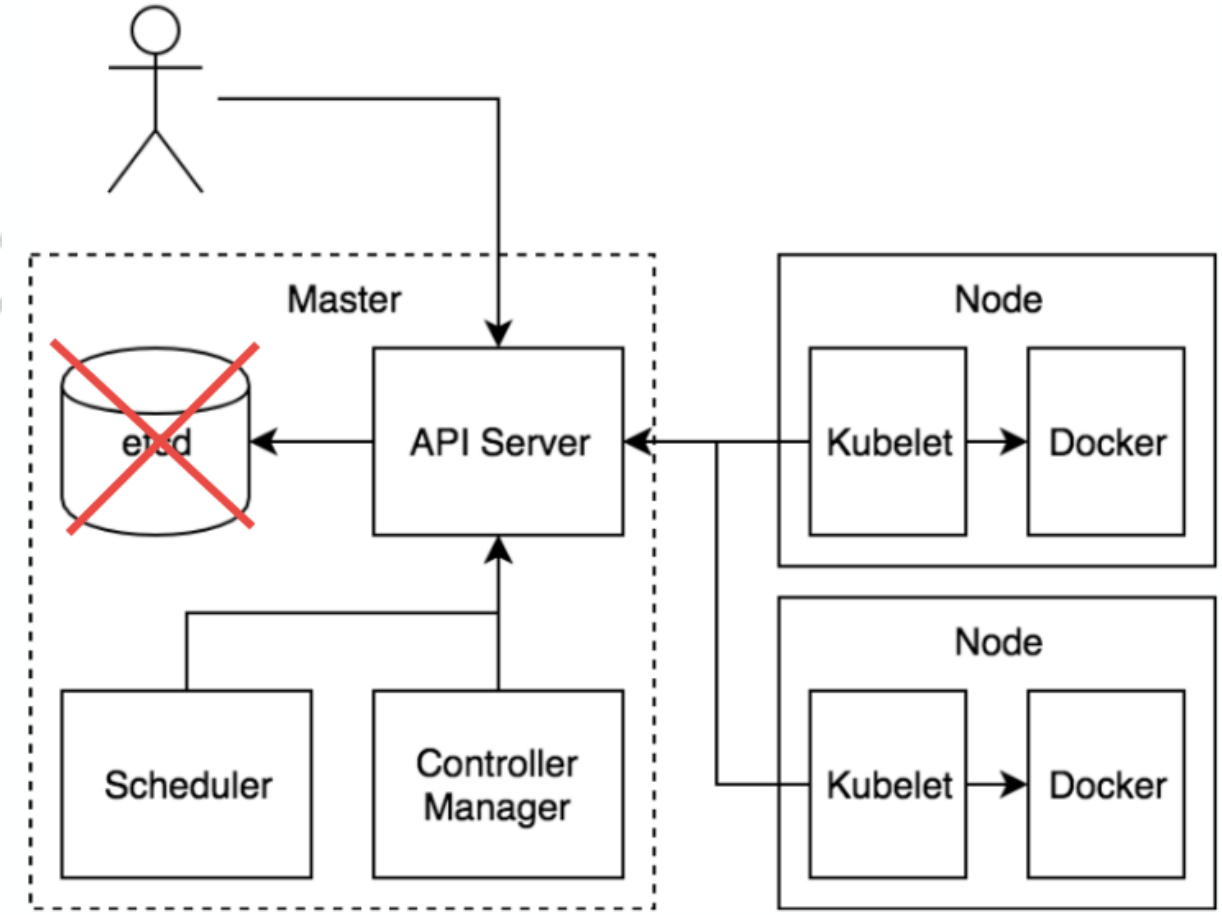
```
(worker1) sudo tail -f /var/log/containers/kube-proxy*
```

Looking at the kubelet log.

```
(worker1) journalctl -u kubelet -f
```

4.3 Cluster Failure Scenarios

4.3.1 Step 01: Cluster Backing Storage (etcd) lost



Note: In a real cluster, this would be caused by etcd crashing or data corruption. In the lab we'll simulate with the command below.

Simulate etcd outage.

```
(master) docker pause k8s_etcd_etcd-master<tab to autocomplete>
```

Symptoms:

- apiserver should fail to come up
- kubelet will not be able to reach it but will continue to run existing pods
- manual recovery necessary before apiserver is restarted

Check api-server log.

```
(master) sudo tail -f /var/log/containers/kube-apiserver*
#it should complain about the backend
```

Check kubectl commands against apiserver.

```
(master) kubectl get nodes
#should time out or fail to connect.
```

Check kubelet log.

```
(worker1) journalctl -u kubelet -f
#should fail to load config
```

Check the kubelet service on any node

```
(worker1) systemctl status kubelet
#should still be running.
```

Check the existing workloads

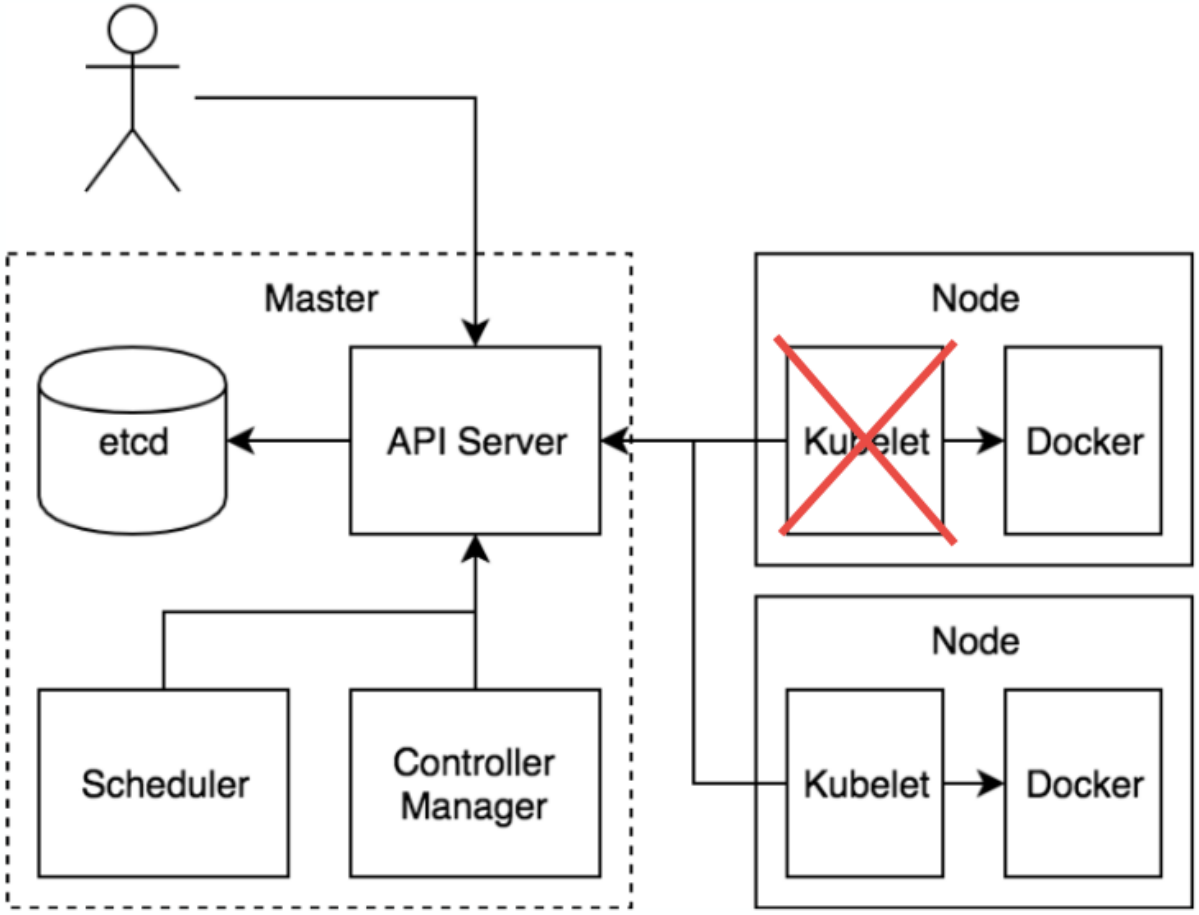
```
(worker1) docker ps
#existing containers and pods continue running
```

Restore to normal:

```
(master)docker unpause  k8s_etcd_etcd-master<tab to autocomplete>
```

Repeat the commands provided above to see output after etcd is restored.

4.3.2 Step 02: kubelet is not running on a worker1



Note: In a real cluster, this would be caused be worker node going offline or kubelet crashing

Simulate kubelet outage, go to worker1 node and run the command below.

```
(worker1) sudo systemctl stop kubelet
```

Symptoms:

- unable to stop, update, or start new pods, services, replication controller

Check the kubelet service on worker1

```
(worker1) systemctl status kubelet
#should be inactive
```

Check kubelet log on worker node.

```
(worker1) journalctl -u kubelet
#it should say main process exited
```

List the cluster.

```
(master) kubectl get nodes
#worker1 status is not ready
```

Check api-server log.

```
(master) sudo tail -f /var/log/containers/kube-apiserver*
#it should not have anything out of the ordinary
```

Run an nginx deployment with 2 replicas, it should deploy only to worker2

```
(master) kubectl run lab04-kubelet --image=nginx --replicas=2
#run get pods to verify they landed on worker2 only instead of one on each node
(master) kubectl get pods -o wide
```

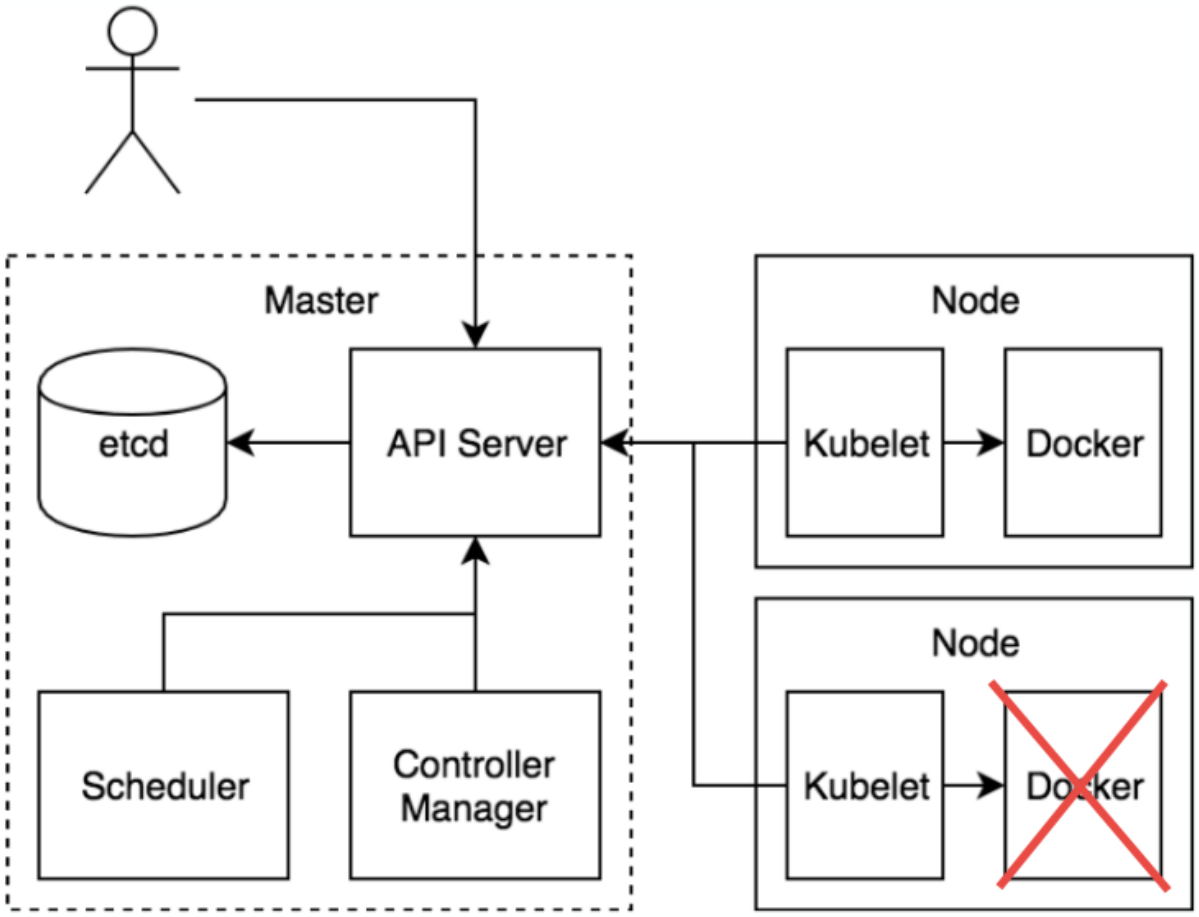
Restore to normal:

```
(worker1) sudo systemctl start kubelet
```

Repeat the commands provided above to see output after kubelet is restored.

```
(master) kubectl get pods -o wide
#the pods are still only on worker 2 since they are not rescheduled during execution
#clean up and redeploy
(master) kubectl scale deploy lab04-kubelet --replicas=0
(master) kubectl scale deploy lab04-kubelet --replicas=2
#there should be 1 pod on each node
#clean up
(master) kubectl delete deploy lab04-kubelet
```

4.3.3 Step 03: Container Engine crashing on worker2



Note: In a real cluster, this would be caused be worker node going offline or docker crashing.

Simulate docker outage.

```
(worker2) sudo systemctl stop docker
```

Symptoms:

- unable to create new workloads. Existing workloads are stopped.

Check the kubelet service.

```
(worker2) systemctl status kubelet
#running, but log should complain about docker
```

Check kubelet log on worker node.

```
(worker2) journalctl -u kubelet -f
#it should complain about docker
```

Check docker ps on worker node.

```
(worker2) docker ps
#it should fail to connect to docker daemon
```

Check kubectl get nodes command.

```
(master) kubectl get nodes
#worker2 should show not ready
```

Check kubectl describe nodes command.

```
(master) kubectl describe nodes worker2
#worker2 should warn about docker
```

Check api-server log.

```
(master) sudo tail -f /var/log/containers/kube-apiserver*
#should not have any related errors
```

Run an nginx deployment with 2 replicas.

```
(master) kubectl run lab04-docker --image=nginx --replicas=2
#run get pods to verify they landed on worker1 only instead of one on each node
(master) kubectl get pods -o wide
```

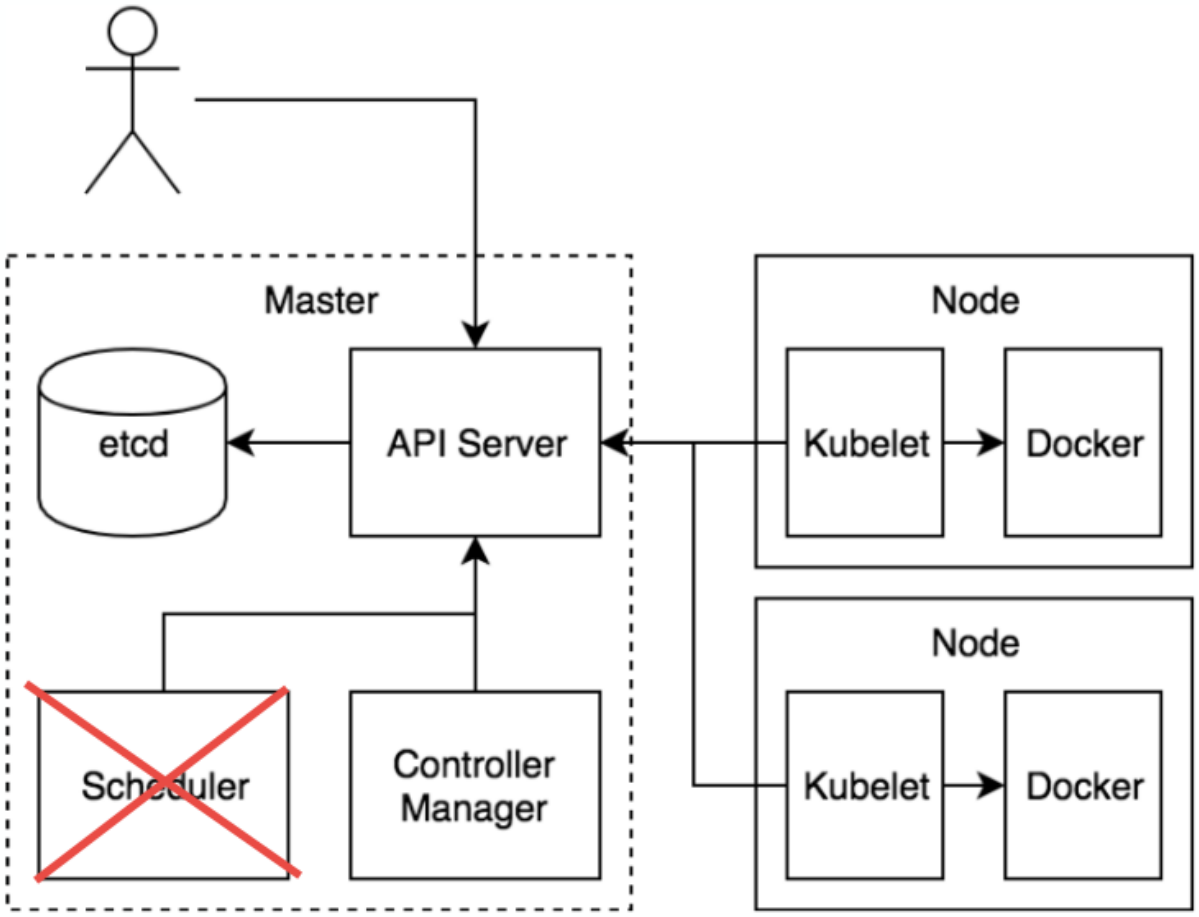
Restore to normal:

```
(worker2) sudo systemctl start docker
```

Repeat the commands provided above to see output after kubelet is restored.

```
(master) kubectl get pods -o wide
#the pods are still only on worker1 since they are no rescheduled during execution
#clean up and redeploy
(master) kubectl scale deploy lab04-docker --replicas=0
(master) kubectl scale deploy lab04-docker --replicas=2
#there should be 1 pod on each node
#clean up
(master) kubectl delete deploy lab04-docker
```

4.3.4 Step 04: Scheduler service outage



Note: In a real cluster, this would be caused by kube-scheduler vm shutdown or kube-scheduler crashing.

Simulate scheduler outage.

```
(master) sudo mv /etc/kubernetes/manifests/kube-scheduler.yaml /home/ubuntu/
```

Symptoms:

- pods get created but will not be scheduled to a node

List the cluster status

```
(master) kubectl get nodes
#everything should be fine as expected
```

Create a pod

```
(master) kubectl run lab04-scheduler --image=nginx --restart=Never
```

Check the pod details

```
(master) kubectl get pod lab04-scheduler
#status should be pending
(master) kubectl describe pod lab04-scheduler
#no event messages
```

Restore to normal:

```
(master)sudo mv /home/ubuntu/kube-scheduler.yaml /etc/kubernetes/manifests/
```

Verify that the pod status moves from pending to running.

```
(master)kubectl get pod lab04-scheduler
#status should be running
(master)kubectl describe pod lab04-scheduler
#event message from scheduler assigning pod to either worker1 or worker2
```

4.4 Lab 04 Conclusion

In this lab we explored the system components running in a Kubernetes cluster on both master and worker nodes. We simulated various failures and worked through the symptoms and how to recover them.

CHAPTER

FIVE

LAB 05: EXTERNAL AUTHENTICATION AND ONBOARDING

Kubernetes does not provide a built-in mechanism for authenticating users. Instead, it relies on integrations with external authentication providers. In this lab you will deploy Dex, which is an OpenID Connect provder and Gangway, which provides a web UI for users to log in and retrieve Kubernetes login information.

5.1 Prepare Configuration Files

There are a number of configuration files which must be populated with the IP and hostname of your master node. To simply things, run the following command to automatically inject the correct information into the files.

```
cd ~
wget http://localhost:8081/_static/lab-files.tar.gz
tar zxvf lab-files.tar.gz
cd ./lab05/code
./generate-config.sh
```

Take a few minutes to review the completed config files.

5.2 Deploy Dex

5.2.1 Step 01: Generate TLS Certificates

```
cd ~/lab05/code/dex/

./gencert.sh
```

5.2.2 Step 02: Copy the Certificate Authority

```
cd ~/lab05/code/dex/ssl

sudo cp ca.pem /etc/ssl/certs/dex-ca.pem
```

5.2.3 Step 03: Create a secret to store the Certificates

```
cd ~/lab05/code/dex/ssl

kubectl create namespace dex

kubectl create secret tls dex-tls --cert=cert.pem --key=key.pem --namespace dex
```

5.2.4 Step 04: Deploy Dex

```
cd ~/lab05/code/dex

kubectl apply -f dex.yaml
```

5.3 Configure kube-apiserver

5.3.1 Step 01: Retrieve the api server arguements for enabling Kubernetes to use Dex

```
cd ~/lab05/code/kubernetes

cat kube-apiserver.yaml
```

Copy the output to an editor. You will need to paste this in the next step.

5.3.2 Step 02: Add arguments to the Kubernetes API server configuration

```
sudo vi /etc/kubernetes/manifests/kube-apiserver.yaml
```

Add the lines from the previous step to the end of spec.containers.command

5.3.3 Step 03: Verify API server availability

The changes to the API server configuration will take effect automatically. If there are any issues with the configuration, the API server will likely crash. Verify that it is still running.

```
kubectl get nodes
```

5.4 Deploy Gangway

5.4.1 Step 01: Create encryption key for client cookies

```
kubectl create namespace gangway

kubectl -n gangway create secret generic gangway-key \
    --from-literal=sesssionkey=$(openssl rand -base64 32)
```

5.4.2 Step 02: Create secret for the Dex Certificate Authority

```
cd ~/lab05/code/dex/ssl

kubectl create secret generic dex-ca --from-file ca.pem --namespace gangway
```

5.4.3 Step 03: Deploy Gangway resources

```
cd ~/lab05/code/gangway

kubectl apply -f .
```

5.5 Test Authentication

5.5.1 Step 01: Access Gangway UI in your browser

Open http://gangway.<EXTERNAL_IP>.nip.io in your browser. Replace <EXTERNAL_IP> with the public IP for your master node.

5.5.2 Step 02: Login

We predefined a user account within Dex for you. You can see it in `~/auth-lab/snippets/dex/dex.yaml` within the dex ConfigMap. Login with those credentials now `admin@example.com` / `password`.

5.5.3 Step 03: Retrieve kubeconfig settings

After logging in, you should be presented with instructions for installing kubectl and configuring it. Since kubectl is already installed, you should skip the first section. Copy the contents of the second section and paste it into the terminal for your master node.

5.5.4 Step 04: Test Access

Run a test command to see if things are working

```
kubectl get nodes
```

This should fail because we haven't granted any permissions to the external user.

5.5.5 Step 05: Grant Access

Create a ClusterRoleBinding that gives `admin@example.com` the `cluster-admin` Cluster Role. Note that we are running the command with the original local administrator context.

```
kubectl create clusterrolebinding admin-example-com --user=admin@example.com --
↪clusterrole=cluster-admin --context=kubernetes-admin@kubernetes
```

5.5.6 Step 06: Test Access Again

Try testing access again.

```
kubectl get nodes
```

This time, it should work now that `admin@example.com` has the correct permissions.

CHAPTER

SIX

LAB 06: CLUSTER MAINTENANCE

This lab will cover worker node maintenance as well as Cluster backup and restore. Node maintenance will enable you to take worker nodes offline gracefully and complete maintenance tasks(Kernel upgrade, Container runtime, hardware changes, etc) Backup and restore will reduce the time to recovery should you lose the cluster state, it can also help with migrating data from one cluster to another.

6.1 Node Maintenance

Kubectl drain can be used to safely evict all of your pods from a node before you perform maintenance on the node (e.g. kernel upgrade, hardware maintenance, etc.). Safe evictions allow the pod's containers to gracefully terminate 4 and will respect the you have specified.

Note: By default kubectl drain will ignore certain system pods on the node that cannot be killed (Daemonsets, etc);

When kubectl drain returns successfully, that indicates that all of the pods (except the ones excluded as described in the previous paragraph) have been safely evicted. It is then safe to bring down the node by powering down its physical machine or, if running on a cloud platform, deleting its virtual machine.

Let's create a sample workload to see how it behaves when the node is drained.

```
kubectl run lab06-drain --image=nginx --replicas=2
```

Verify that a pod has been created on both workers

```
kubectl get pods -o wide
```

Identify the name of the node you wish to drain. You can list all of the nodes in your cluster with

```
kubectl get nodes
```

Drain the node (in our case lets choose worker1):

```
kubectl drain worker1 --ignore-daemonsets --delete-local-data
```

Verify that there are no more pods on worker1, and the pod that was there moved to worker2.

```
kubectl get pods -o wide
```

Once that's done, the node would be ready for maintenance. In a real world situation, that might include reboots, and shutdown. When ready. run the command below to resume scheduling new pods onto the node.

```
kubectl uncordon worker1
kubectl get pods -o wide
#existing pods have been moved to other nodes will stay put
#if we delete and recreate the deployment it will be distributed to worker1
kubectl delete deploy lab06-drain
kubectl run lab06-drain --image=nginx --replicas=2
kubectl get pods -o wide
```

Clean up our drain test deployment.

```
kubectl delete deploy lab06-drain
```

6.2 Backup and Restore

Backing up a Kubernetes cluster enables operators to recover from cluster failures or the accidental deletion of objects. Velero is a free, open-source, tool for backing up and restoring all, or portions of a Kubernetes cluster. In this lab, we'll install Velero, backup our Kubernetes cluster, delete a namespace, and perform a restore.

6.3 Installing Velero

Start by downloading and installing the Velero command line tool.

```
cd ~
wget https://github.com/heptio/velero/releases/download/v1.1.0/velero-v1.1.0-linux-
↪amd64.tar.gz
tar zxvf velero-v1.1.0-linux-amd64.tar.gz
cd velero-v1.1.0-linux-amd64/
sudo cp velero /usr/local/bin
```

Velero requires an object storage provider to store it's backups. It supports many of the major cloud object storage providers including Amazon S3 and Google GCS. For our lab, we will deploy Minio, which is an S3 compatible object store which can be run directly in our lab environment.

```
cd ~/velero-v1.1.0-linux-amd64/
kubectl apply -f examples/minio/00-minio-deployment.yaml
```

Next, create a credentials file that Velero will use to authenticate with Minio.

> **Warning:** Use the key_id and access_key exactly as shown below. These are the default credentials for Minio. While Minio does allow you to change the credentials, we'll use the default ones to keep things simple for now.

```
cd ~/velero-v1.1.0-linux-amd64/

cat <<EOF >>credentials-velero
[default]
aws_access_key_id = minio
aws_secret_access_key = minio123
EOF
```

Now use the Velero command line tool to deploy the Velero server to the Kubernetes cluster.

```
velero install \
--provider aws \
--bucket velero \
--secret-file ./credentials-velero \
--use-volume-snapshots=false \
--backup-location-config region=minio,s3ForcePathStyle="true",s3Url=http://minio.
↪velero.svc:9000
```

6.4 Creating a Backup

Velero can backup an entire cluster or just certain objects. We'll backup the entire cluster now.

```
velero backup create my-cluster-backup
```

Check the status of the backup by running the following.

```
velero backup describe my-cluster-backup
```

When *Phase* shows *Completed*, the backup is complete.

In this case, we manually initiated a backup, but Velero also supports scheduling backups.

6.5 Restoring from a Backup

Now let's simulate a common scenario which would require a restore. Delete the `kube-instrumentation` namespace. This will not only delete the namespace, but all the pods and services we deployed within it.

```
kubectl delete ns kube-instrumentation
```

Note: The namespace can take up to a minute to delete. The terminal will appear to hang during this time.

Verify that the namespace and pods within have been deleted.

```
kubectl get ns

kubectl get pods -n kube-instrumentation
```

Now let's use Velero to restore the deleted namespace.

```
velero restore create --from-backup my-cluster-backup
```

Check the progress of the restore operation by running the `velero restore describe` command from the output of the last command.

Note: You can safely ignore any warnings in the output.

When *Phase* shows *Completed*, the restore is complete. Verify that the namespace and pods within have been restored.

```
kubectl get ns

kubectl get pods -n kube-instrumentation
```

6.6 Summary / Takeaways

This lab demonstrated a fairly simple backup and recovery using Velro. Velero can backup and restore a subset of the cluster as demonstrated in this lab, but can also be used to restore an entire cluster in the event of a major failure. You also ran a single manual backup, but Velero can also be configured to run backups on a schedule.

CHAPTER

SEVEN

LAB 01: SOLUTIONS

7.1 List each namespace name by itself newline separated.

```
kubectl get namespaces -o jsonpath='{range.items[*]}{.metadata.name}{"\n"}{end}'
```

7.2 Forward the nginx port from one of the pods onto localhost:8000 of your client machine

```
kubectl port-forward <pod_id> 8000:80
```

7.3 Start a tail with follow of the logs on the same pod used above

```
kubectl logs -f <pod_id>
```

7.4 Run a remote command inside the container to for example get the current date and time

```
kubectl exec <pod_id> date
```

7.5 Clean up all resources in the default namespace:

```
kubectl delete all --all
```

CHAPTER

EIGHT

LAB 02: SOLUTIONS

No solutions for lab 02.

CHAPTER

NINE

LAB 03: SOLUTIONS

No solutions for lab 03.

CHAPTER

TEN

LAB 04: SOLUTIONS

10.1 Cluster Reset

Since this lab deals with breaking the cluster, there's a small chance that the cluster may get in to a state that is not easily recoverable. In this situation, we maybe need to follow the process below to reset the cluster in order to move on to lab05.

Note: The process below will reset the cluster to scratch and will clear any workloads created in lab02 and lab03. This process is not recomended unless troubleshooting options have been exhausted.

```
#reset the cluster using kubeadm
<master> sudo kubeadm reset
<worker1> sudo kubeadm reset
<worker2> sudo kubeadm reset
#reinitialize the cluster and install CNI. It will ask you to overwrite the admin.conf
#say yes
<master> k8s-start
#copy the join command and add sudo in front of it
<worker1> sudo kubeadm join ......
<worker2> sudo kubeadm join ......
#verify cluster is running
<master> kubectl get nodes
```

Note: if the k8s-start script does not print the join commands, please use kubeadm token create –print-join-command

CHAPTER

ELEVEN

LAB 05: SOLUTIONS

11.1 List all logging services

```
kubectl get services --all-namespaces -l function=logging
```

11.2 List all label key/value pairs on all deployments across all namespaces

```
kubectl get deployment -o jsonpath='{.items[*].metadata.labels}' --all-namespaces |␣
↪sed -e 's/] /\'$'\n/g' | sed 's/map\[//g' | sed 's/ /\'$'\n/g' | sort | uniq
```

11.3 List all non-logging related pods

```
kubectl get pods --all-namespaces -l function!=logging
```

11.4 Resource Quotas:

lab-05-resourceQuota.yaml

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resource
  namespace: default
spec:
  hard:
    pods: "4"
    requests.cpu: "1"
    requests.memory: 512Mi
    limits.cpu: "2"
    limits.memory: 1Gi
```

11.5 Determining Nginx deployment failure:

```
kubectl describe replicaset -l app=nginx

# Check the event log in the output for errors
```

11.6 Add memory request to Elasticsearch:

```
spec:
  containers:
  - name: elasticsearch
    image: docker.elastic.co/elasticsearch/elasticsearch:5.5.1
    resources:
      requests:
        memory: "600Mi"
```

11.7 Add resource requests and limits to Nginx:

```
resources:
  requests:
    memory: "128Mi"
    cpu: 0.5
  limits:
    memory: "256Mi"
    cpu: 1
```

11.8 Node Affinity:

node-affinity2.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.13.0
        ports:
        - containerPort: 80
```

(continues on next page)

(continued from previous page)

```
      # Only run the nginx pods on nodes that do NOT have reserved=customer01
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: reserved
                operator: NotIn
                values:
                - customer01
```

node-affinity3.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.13.0
        ports:
        - containerPort: 80
      # Only run the nginx pods on nodes that have reserved=customer01 or
      # reserved=customer02, but prefer reserved=customer02
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: reserved
                operator: In
                values:
                - customer01
                - customer02
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: reserved
                operator: In
                values:
                - customer02
```

CHAPTER

TWELVE

LAB 06: SOLUTIONS