Log Management and Monitoring on EKS

Table of Contents

Overview ***3***

Assumptions ***3***

Quick Review ***3***

Amazon EKS control plane logging **3**

EKS CloudTrail Logging **5**

Log Management ***5***

ContainerInsights **6**

Fluentd 6

CloudWatch Agent 6

Installation **6**

Monitoring ***9***

Container Insights **9**

Control Plane **10**

Prometheus 10

Grafana 10

Installation 11

Overview

By default EKS ships with two kinds of logging mechanisms; Amazon EKS control plane logging and EKS CloudTrail Logging. Although useful, these two kinds of logs by themselves are not enough to fully understand the state of the cluster at any given point in time. In such a scenario application and system logs become highly important for us to understand what is happening inside our cluster. These logs are particularly useful for debugging problems and monitoring cluster activity.
As kubernetes is a complex system with many moving pieces, it becomes all the more important for us to have a solid monitoring system in place in addition to logging.

In this document we will look beyond out of the box logging strategies mentioned above and dive deeper into different logging and monitoring solutions which can be installed on to our EKS cluster as add-on's. This document in intended to be an in-depth and comprehensive guide to logging and monitoring solutions on EKS. Following are the outcomes of this tutorial.

1. Introduction to Container Insights as logging mechanisms and integrate them into AWS Elasticsearch cluster via AWS CloudWatch log groups. We will also run Kibana dashboard and view metrics at different hierarchy such as cluster, namespace, pod, container etc.
2. Introduction to Prometheus and Grafana as monitoring systems and installation on EKS cluster.

Assumptions

This tutorial makes following assumptions

- An EKS cluster is already up and running in your AWS account.
- Kubectl client is installed and running on your machine and is able to connect to your cluster successfully. Your machine can be your local system/Cloud9/EC2 instance.
- GIT is installed and running on your machine.
- Homebrew is installed and running

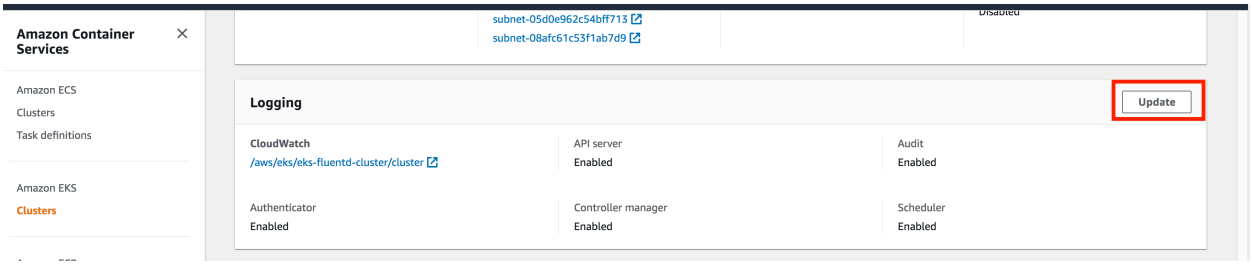
```
Admin:~/workspace/eks-monitoring/monitoring (master) $ kubectl version
Client Version: version.Info{Major:"1", Minor:"14+", GitVersion:"v1.14.7-eks-1861c5", GitCommit:"1861c597586f84f1498a9f2151c78d8a6bf47814", GitTreeState
:"clean", BuildDate:"2019-09-24T22:12:08Z", GoVersion:"go1.12.9", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"14+", GitVersion:"v1.14.6-eks-5047ed", GitCommit:"5047edce664593832e9b889e447ac75ab104f527", GitTreeState
:"clean", BuildDate:"2019-08-21T22:32:40Z", GoVersion:"go1.12.9", Compiler:"gc", Platform:"linux/amd64"}
Admin:~/workspace/eks-monitoring/monitoring (master) $ git --version
git version 2.14.5
Admin:~/workspace/eks-monitoring/monitoring (master) $ brew --version
Homebrew 2.1.15
Homebrew/linuxbrew-core (git revision 87d4f0; last commit 2019-10-30)
Admin:~/workspace/eks-monitoring/monitoring (master) $
```

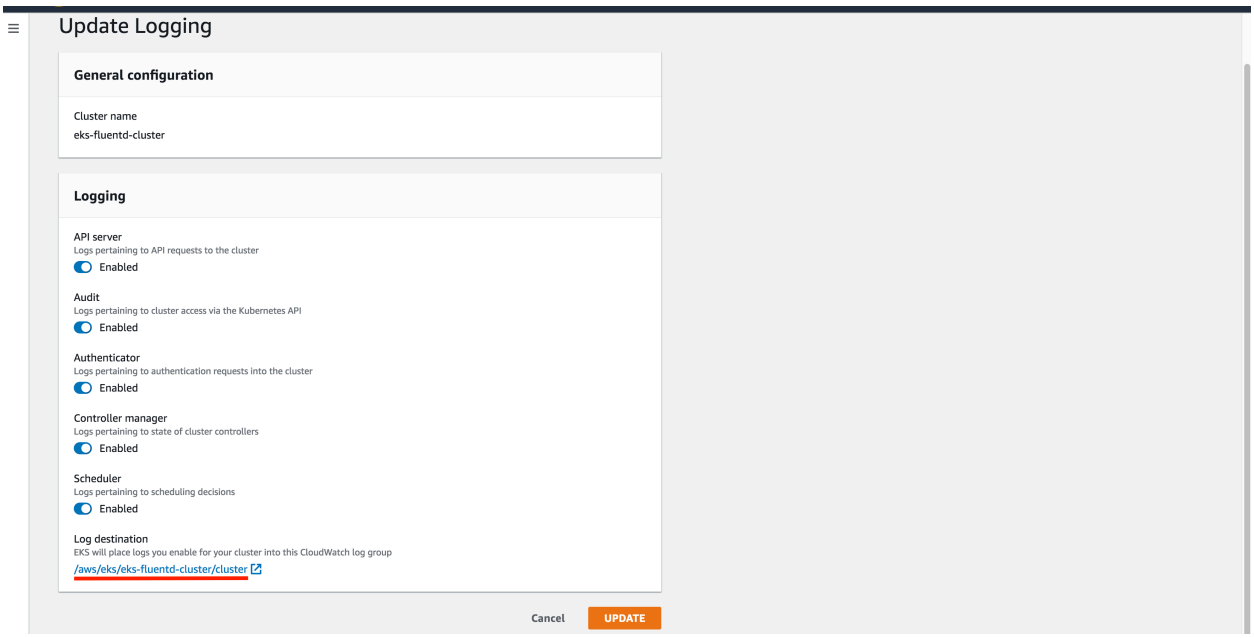
Quick Review

This section is a quick review of what EKS already has in store for us in terms of log management.

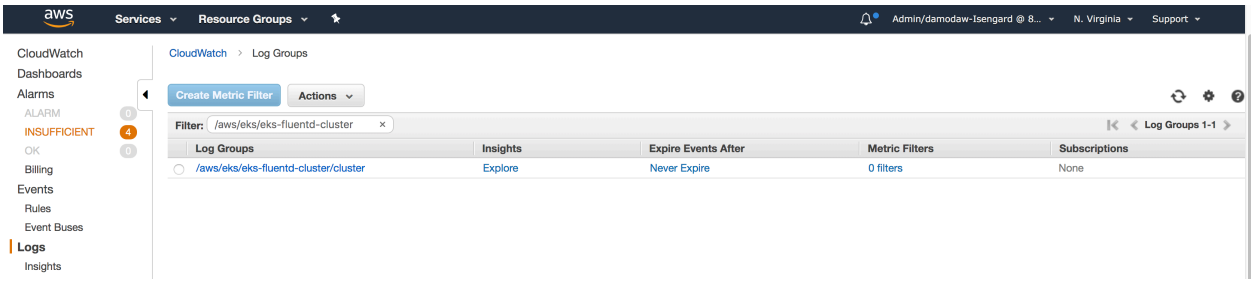
Amazon EKS control plane logging

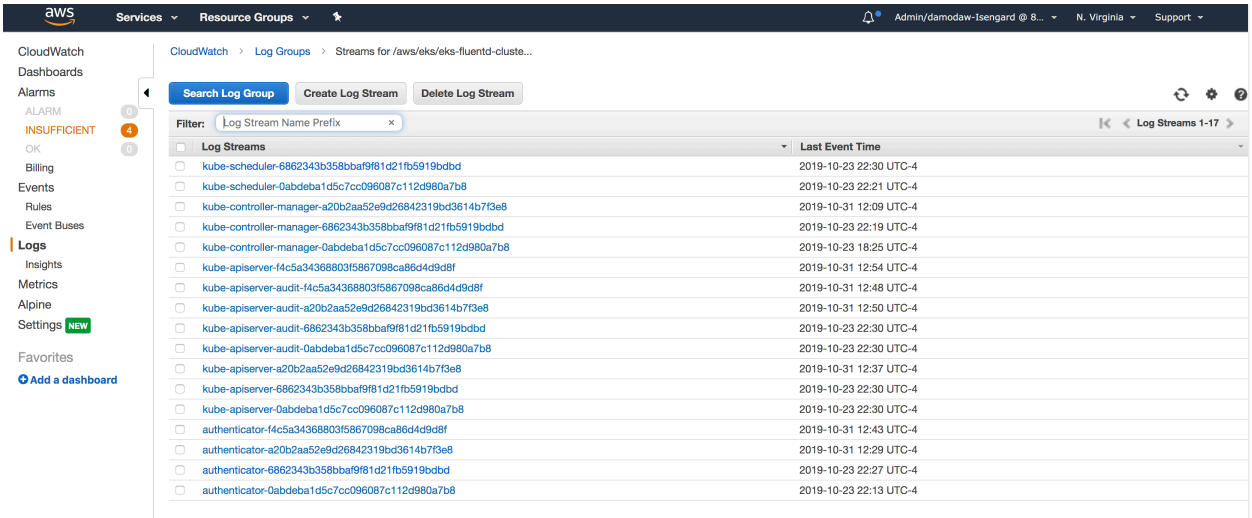
Provides audit and diagnostic logs directly from the Amazon EKS control plane to CloudWatch Logs in your account. These logs make it easy for you to secure and run your clusters. They can be enabled or disabled on the AWS console for EKS cluster as shown below.





When enabled these logs become visible as a log group on CloudWatch as shown below, where you can do things like applying metric Filters and subscription filters etc.

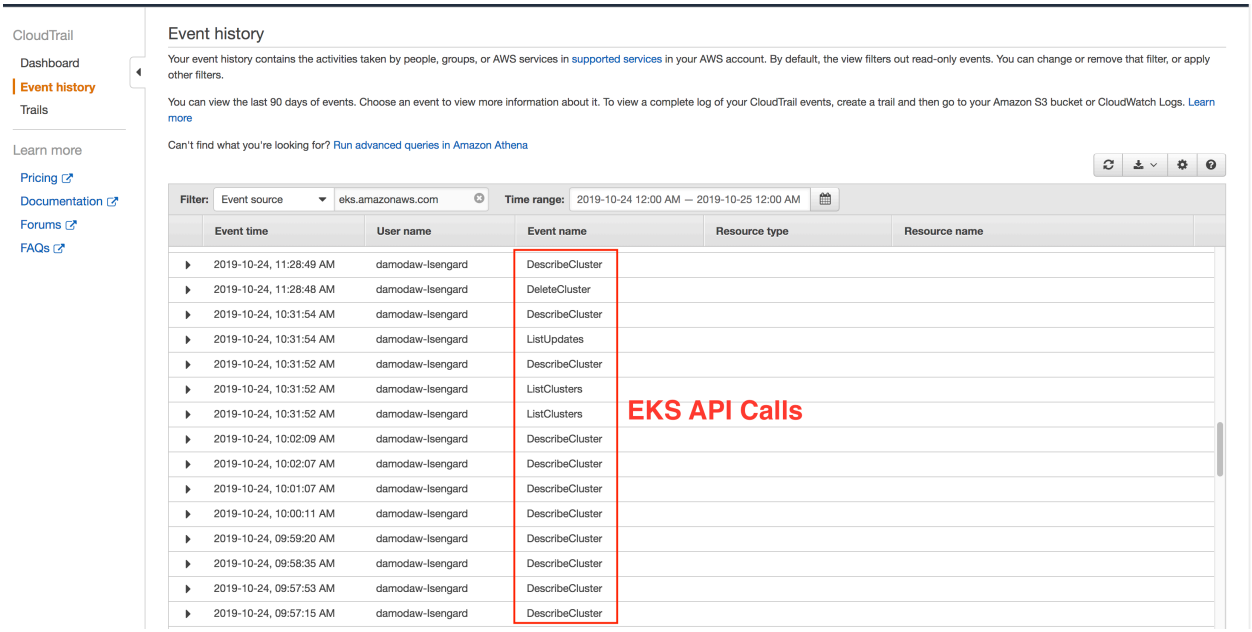




More info on this can be found at **Amazon EKS Control Plane Logging**

EKS CloudTrail Logging

Amazon EKS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon EKS. CloudTrail captures all API calls for Amazon EKS as events. The calls captured include calls from the Amazon EKS console and code calls to the Amazon EKS API operations as shown below. More info on this can be found at **Logging Amazon EKS API Calls with AWS CloudTrail**.



Log Management

Most modern applications have some kind of logging mechanism; as such, most container engines are likewise designed to support some kind of logging. The easiest and most embraced logging method for containerized applications is to write to the standard output and standard error streams.

Native functionality provided by a container engine or runtime is usually not enough for a complete logging solution. For example, if a container crashes, a pod is evicted, or a node dies, you'll usually still want to access your application's logs. As such, logs should have a separate storage and lifecycle independent of nodes, pods, or containers. This concept is called cluster-level-logging. Cluster-level logging requires a separate backend to store, analyze, and query logs. Kubernetes provides no native storage solution for log data, but you can integrate many existing logging solutions into your Kubernetes cluster.
This is where container insights comes into picture.

ContainerInsights

- Used to collect, aggregate, and summarize metrics and logs from your containerized applications and microservices. The metrics include utilization for resources such as CPU, memory, disk, and network.
- Container Insights also provides diagnostic information, such as container restart failures, to help you isolate issues and resolve them quickly.
- CloudWatch alarms can be set on metrics that Container Insights collects.
- More info can be found at **Using Container Insights**

Fluentd

- Fluentd is an open source data collector, which lets you unify the data collection and consumption for a better use and understanding of data.
- It tries to structure data as JSON as much as possible, this allows it to unify all facets of processing log data: collecting, filtering, buffering, and outputting logs across multiple sources and destinations. In our case the destination is **AWS CloudWatch** Logs.
- It runs as DaemonSet on our EKS cluster, collects and pushes logs to specific AWS CloudWatch log groups.
- Data is collected in the following log groups
  /aws/containerinsights/cluster_name/application
  /aws/containerinsights/cluster_name/dataplane
  /aws/containerinsights/cluster_name/host
- More details can be found on the page **Set Up FluentD as a DaemonSet to Send Logs to CloudWatch Logs**

CloudWatch Agent

- CloudWatch agent runs as a DaemonSet on our EKS cluster
- It collects metrics that become available on CloudWatch automatic dashboards. List of metrics collected can be found at **Amazon EKS and Kubernetes Container Insights Metrics**
- Data is collected as performance log events in a log group
  /aws/containerinsights/cluster_name/performance
- More details can be found on the page **Set Up the CloudWatch Agent to Collect Cluster Metrics**

Installation

Step 1: Clone GIT repo

Open terminal window

Clone the GIT repo *<custom git hub repo>* and navigate to folder *logging*

$ git clone <repository>

$ cd eks-monitoring

$ git checkout –b feature/eks-monitoring

$ git pull origin feature/eks-monitoring

$ cd logging

Step 2: Install Prerequisites

Before we begin installation, we need to make sure that the worker nodes have right set of permissions to enable them to send metrics and logs to CloudWatch. If your worker node already has *arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy* attached, then you may skip this step.

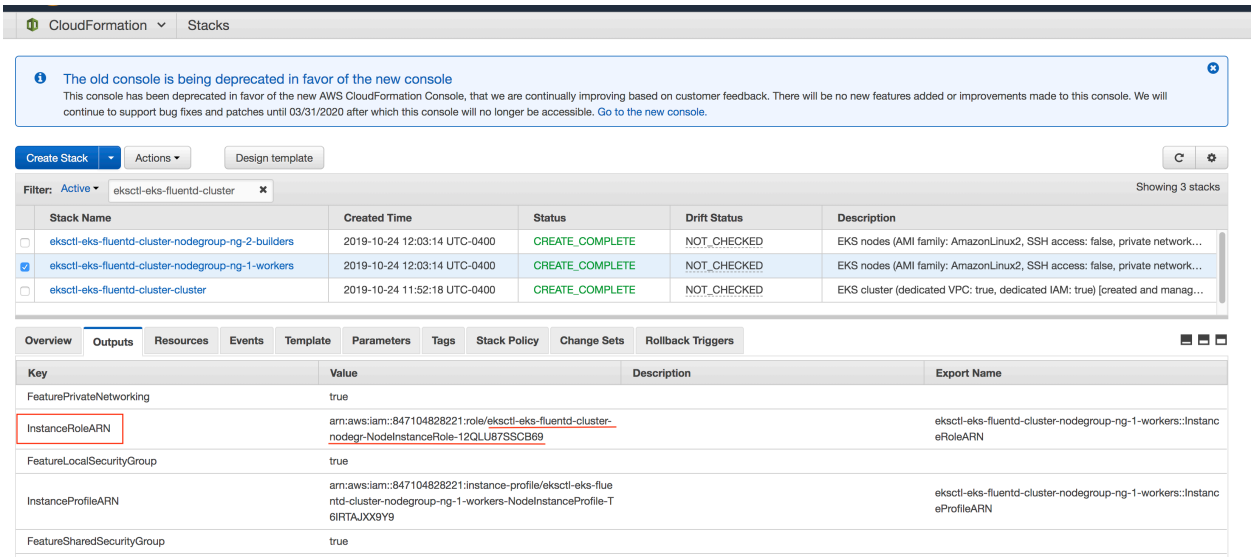
Execute following commands. Update <node-role-name> with actual worker node role name such as *eksctl-eks-fluentd-cluster-nodegr-NodeInstanceRole-174JRSLDL4WUB*
This will ensure that the worker node role has correct policy attached *arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy*

$ chmod +x update-node-role.sh

$ sh update-node-role.sh <node-role-name>

```
bash - *ip-192-168-9 ×   Immediate (Javascrij ×
Admin:~/workspace/eks-cluster-setup/monitoring (master) $ chmod +x update-node-role.sh
Admin:~/workspace/eks-cluster-setup/monitoring (master) $ sh update-node-role.sh eksctl-eks-fluentd-cluster-nodegr-NodeInstanceRole-12QLU87SSCB69
Attaching arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy policy to worker node instance role eksctl-eks-fluentd-cluster-nodegr-NodeInstanceRole-12Q
LU87SSCB69
Admin:~/workspace/eks-cluster-setup/monitoring (master) $ sh update-node-role.sh eksctl-eks-fluentd-cluster-nodegr-NodeInstanceRole-174JRSLDL4WUB
Attaching arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy policy to worker node instance role eksctl-eks-fluentd-cluster-nodegr-NodeInstanceRole-174
JRSLDL4WUB
Admin:~/workspace/eks-cluster-setup/monitoring (master) $
```

If you created your cluster using **eksctl** then you can find worker role node arn in CloudFormation stack outputs for the respective worker node group as shown below. Since I have two node groups I am attaching the policy to both roles. If you have a single node group then you can stick to running the script only for your worker node role.



Step 3: Install ContainerInsights

Here we install Fluentd and CloudWatch agent DaemonSet onto our eks cluster. Execute following commands to do so.

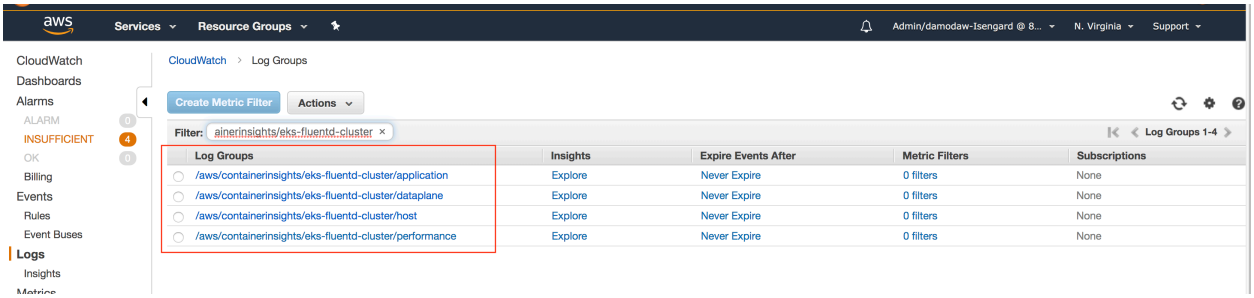
$ chmod +x install-containerinsights.sh
$ sh install-containerinsights.sh "<cluster-name>" "<region>" <metrics_collection_interval> <force_flush_interval>

The parameters supplied to the script are highlighted in the screenshot below. These parameters define the frequency at which the CloudWatch agent collects data from the cluster. There are more configurations we can potentially do here, details of which can be found in under section "*Create a ConfigMap for the CloudWatch Agent*" on **this** page.

```
Admin:~/workspace/eks-cluster-setup/monitoring (master) $ chmod +x install-containerinsights.sh
Admin:~/workspace/eks-cluster-setup/monitoring (master) $ sh install-containerinsights.sh "eks-fluentd-cluster" "us-east-1" 60 20
Creating namespace amazon-cloudwatch
namespace/amazon-cloudwatch created
Creating configmap called cluster-info defining cluster name and log region.
configmap/cluster-info created
Installing fluentd DaemonSet. Manifest used here is vended by AWS and can be found at https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-co
ntainer-insights/master/k8s-yaml-templates/fluentd/fluentd.yaml
serviceaccount/fluentd created
clusterrole.rbac.authorization.k8s.io/fluentd-role created
clusterrolebinding.rbac.authorization.k8s.io/fluentd-role-binding created
configmap/fluentd-config created
daemonset.extensions/fluentd-cloudwatch created
NAME                      READY   STATUS     RESTARTS   AGE
fluentd-cloudwatch-4nnnx  0/1     Init:0/2   0          0s
fluentd-cloudwatch-8fp6q  0/1     Init:0/2   0          0s
fluentd-cloudwatch-jnrvd  0/1     Init:0/2   0          0s
fluentd-cloudwatch-pqqf4  0/1     Init:0/2   0          0s
fluentd-cloudwatch-wcq72  0/1     Init:0/2   0          0s
fluentd-cloudwatch-xln8c  0/1     Init:0/2   0          0s
Creating service account for cloudwatch agent. Manifest used here is vended bu AWS and can be found at https://raw.githubusercontent.com/aws-samples/ama
zon-cloudwatch-container-insights/master/k8s-yaml-templates/cwagent-kubernetes-monitoring/cwagent-serviceaccount.yaml
serviceaccount/cloudwatch-agent created
clusterrole.rbac.authorization.k8s.io/cloudwatch-agent-role unchanged
clusterrolebinding.rbac.authorization.k8s.io/cloudwatch-agent-role-binding unchanged
Updating cloudwatch agent configmap. Basic configmap can be found at https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/
master/k8s-yaml-templates/cwagent-kubernetes-monitoring/cwagent-configmap.yaml
configmap/cwagentconfig created
Installing CloudWatch Agent DaemonSet. The manifest is vended by AWS and can be found at https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch
-container-insights/master/k8s-yaml-templates/cwagent-kubernetes-monitoring/cwagent-daemonset.yaml
daemonset.apps/cloudwatch-agent created
NAME                      READY   STATUS              RESTARTS   AGE
cloudwatch-agent-9c5c9    0/1     ContainerCreating   0          0s
cloudwatch-agent-gjmd7    0/1     ContainerCreating   0          0s
cloudwatch-agent-kmhb2    0/1     ContainerCreating   0          0s
cloudwatch-agent-kx7dh    0/1     ContainerCreating   0          0s
cloudwatch-agent-pbntd    0/1     ContainerCreating   0          0s
cloudwatch-agent-xbp7d    0/1     ContainerCreating   0          0s
```

<cluster-name> <region> <force_flush_interval> <metrics_collection_interval>

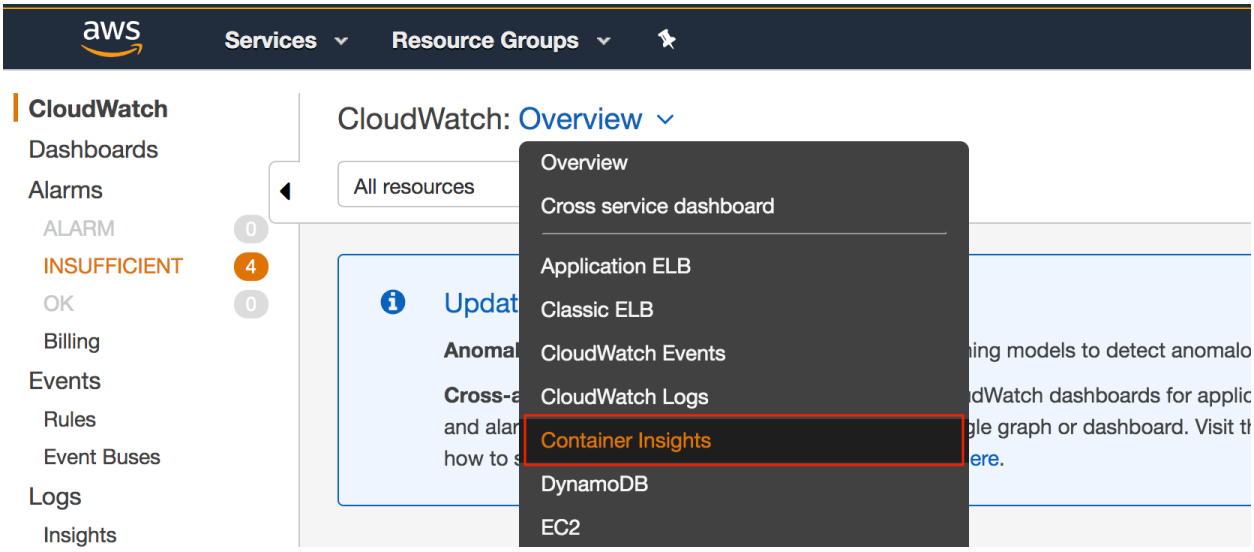
Upon executing the above scripts, fluentd will be installed first followed by the cloudwatch agent. You can run following kubectl command to check if all the pods are installed and they are running.
Once all pods are up and running as shown above, you will see the appropriate log groups forming in AWS CloudWatch as shown below.



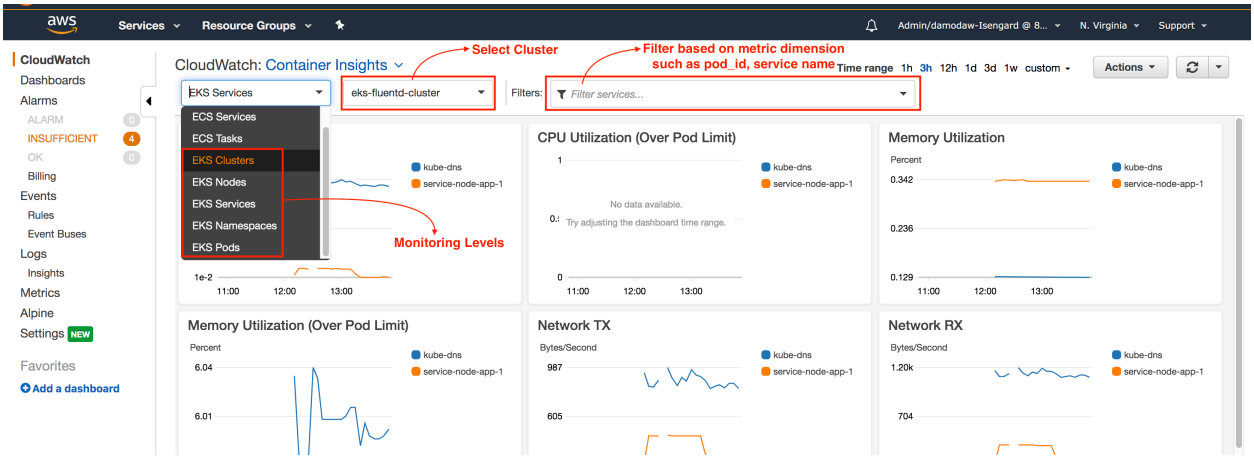
Monitoring

Container Insights

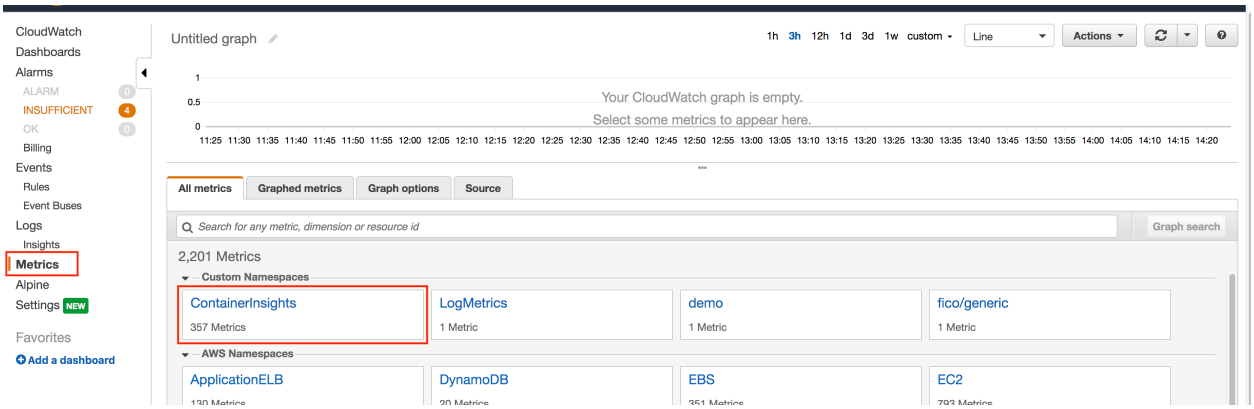
CloudWatch agent DaemonSet installed in the previous section provides monitoring capabilities via Container Insights AWS CloudWatch automatic dashboard. This can be categorized as Data Plane monitoring as the agents running on Data Plane extract data directly from there.
Upon successful installation ***Container Insights*** link appears in ***CloudWatch -> Overview*** dropdown list as shown below. Using this link you can navigate to CloudWatch automatic dashboard where you will see the monitoring charts pertaining to your EKS cluster.



Container insights provides detailed monitoring of EKS cluster at various different levels. Monitoring levels available are Pod, Namespace, Service, Node and Cluster.



The metrics collected are available in the **Metrics** section of AWS CloudWatch under **ContainerInsights** custom namespace as shown below. Routine CloudWatch Alerts can then be created for these metrics.



Control Plane Monitoring

The Kubernetes API server also exposes a number of metrics that are useful for monitoring and analysis. These metrics are exposed internally through a metrics endpoint that refers to the /metrics HTTP API. Like other endpoints, this endpoint is exposed on the Amazon EKS control plane.

Prometheus

- Open-source systems monitoring and alerting toolkit
- A multi-dimensional data model with time series data identified by metric name and key/value pairs
- Consists of main Prometheus server which scrapes raw data from /metrics control plane API and stores time series data
- Comes with alertmanager to handle alerts
- More details can be found at **prometheus.io** and **Prometheus Metrics**

Grafana

- Visualization tool that can query, visualize, alert on and understand your metrics from multiple data sources.
- Built-in support for Prometheus data source. It also supports CloudWatch as the data source.
- More details can be found at **Grafana Docs**

Installation

Step 1: Clone GIT repo

Open terminal window
If you have already cloned the repo in the installation step 1 in logging section then you do not need to clone it again. Simply navigate to folder *monitoring*
$ cd ../monitoring

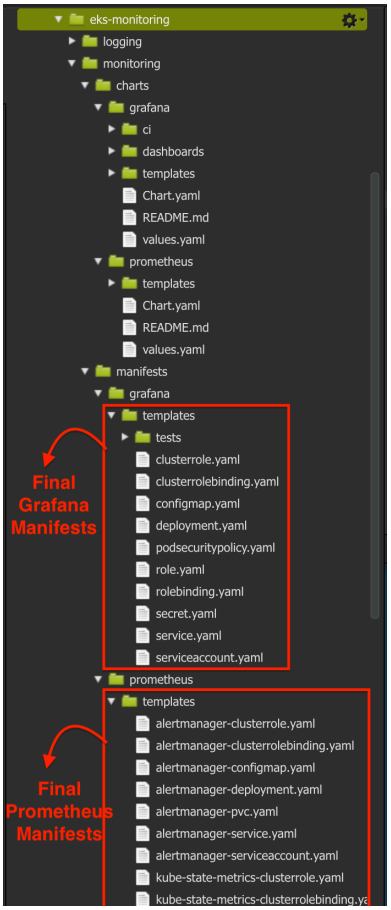
Step 2: Install Prometheus and Grafana without Helm-Tiller

Because of security concerns with ***helm*** and ***tiller*** not everyone is inclined to use them especially in controlled production environment. Thus, we are going to be taking a different approach to installing Prometheus and Grafana. ****You will still need to install helm on the client. Command "brew install Kubernetes-helm" should do it for you.*

De-Constructing Helm Charts:
First we need to generate Prometheus and Grafana manifests to be installed. For this we download helm charts for Prometheus and Grafana and untar them. ****Note that the downloaded charts have been checked into the source repo. The De-Construction of helm chart has been used as one-time process. You may directly use the charts from repo.* Although, if you want to download the charts and untar them yourself then use following commands.
$ chmod +x deconstructing-helm-charts.sh
$ sh deconstructing-helm-charts.sh

```
Admin:~/workspace/eks-monitoring/monitoring (master) $ chmod +x deconstructing-helm-charts.sh
Admin:~/workspace/eks-monitoring/monitoring (master) $ sh deconstructing-helm-charts.sh
Create charts folder to hold downloaded charts
Download helm chart stable/prometheus
Download helm chart stable/grafana
Admin:~/workspace/eks-monitoring/monitoring (master) $
```

Next, we render templates locally and apply them to the cluster using following script. Final manifests will be temporarily available under ***manifests*** folder as shown below. Once the manifests are applied they will automatically be deleted by the script.



At the time of rendering template you may choose to add or override certain configurations via command line options. Following script adds certain configurations when rendering templates. Detailed documentation on additional command line options for Prometheus are provided **here**. Command line options for Grafana can be found **here**. If there are any changes to be made to the configurations then update the script and run it again to render and apply manifests to the cluster.
Note that Grafana will be installed with an internal load balancer. More often than not, organizations want their monitoring dashboards to be private so this will accomplish that goal. Grafana has also been preconfigured with 2 dashboards **Kubernetes Pods** and **Kubernetes All Nodes**

Run following commands to render and install manifests
$ chmod +x install-prometheus-grafana-manifests.sh
$ sh install-prometheus-grafana-manifests.sh

```
Admin:~/workspace/eks-monitoring/monitoring (master) $ chmod +x install-prometheus-grafana-manifests.sh
Admin:~/workspace/eks-monitoring/monitoring (master) $ sh install-prometheus-grafana-manifests.sh
Create manifests folder to hold rendered templates
Creating prometheus namespace
namespace/prometheus created
Render prometheus manifests
wrote ./manifests/prometheus/templates/alertmanager-configmap.yaml
wrote ./manifests/prometheus/templates/server-configmap.yaml
wrote ./manifests/prometheus/templates/alertmanager-pvc.yaml
wrote ./manifests/prometheus/templates/server-pvc.yaml
```

Render Prometheus Manifests Locally

```
wrote ./manifests/prometheus/templates/kube-state-metrics-deployment.yaml
wrote ./manifests/prometheus/templates/pushgateway-deployment.yaml
wrote ./manifests/prometheus/templates/server-deployment.yaml
Applying prometheus templates to cluster
clusterrole.rbac.authorization.k8s.io/prometheus-alertmanager created
clusterrolebinding.rbac.authorization.k8s.io/prometheus-alertmanager created
configmap/prometheus-alertmanager created
deployment.extensions/prometheus-alertmanager created
persistentvolumeclaim/prometheus-alertmanager created
service/prometheus-alertmanager created
serviceaccount/prometheus-alertmanager created
```

```
deployment.extensions/prometheus-server created
persistentvolumeclaim/prometheus-server created
service/prometheus-server created
serviceaccount/prometheus-server created
Patching config map to set up alerts
configmap/prometheus-server patched
```

```
Creating grafana namespace
namespace/grafana created
Rendering grafana manifests
wrote ./manifests/grafana/templates/podsecuritypolicy.yaml
wrote ./manifests/grafana/templates/tests/test-podsecuritypolicy.yaml
wrote ./manifests/grafana/templates/secret.yaml
wrote ./manifests/grafana/templates/configmap.yaml
wrote ./manifests/grafana/templates/dashboards-json-configmap.yaml
wrote ./manifests/grafana/templates/tests/test-configmap.yaml
wrote ./manifests/grafana/templates/serviceaccount.yaml
wrote ./manifests/grafana/templates/tests/test-serviceaccount.yaml
wrote ./manifests/grafana/templates/clusterrole.yaml
wrote ./manifests/grafana/templates/clusterrolebinding.yaml
wrote ./manifests/grafana/templates/role.yaml
wrote ./manifests/grafana/templates/tests/test-role.yaml
wrote ./manifests/grafana/templates/rolebinding.yaml
wrote ./manifests/grafana/templates/tests/test-rolebinding.yaml
wrote ./manifests/grafana/templates/service.yaml
wrote ./manifests/grafana/templates/tests/test.yaml
wrote ./manifests/grafana/templates/deployment.yaml
Applying grafana templates to cluster
clusterrole.rbac.authorization.k8s.io/grafana-clusterrole configured
clusterrolebinding.rbac.authorization.k8s.io/grafana-clusterrolebinding unchanged
configmap/grafana created
configmap/grafana-dashboards-default created
deployment.apps/grafana created
podsecuritypolicy.policy/grafana configured
role.rbac.authorization.k8s.io/grafana created
rolebinding.rbac.authorization.k8s.io/grafana created
secret/grafana created
service/grafana created
serviceaccount/grafana created
configmap/grafana-test created
podsecuritypolicy.policy/grafana-test configured
role.rbac.authorization.k8s.io/grafana-test created
rolebinding.rbac.authorization.k8s.io/grafana-test created
serviceaccount/grafana-test created
pod/grafana-test created
Remove menifests folder
Admin:~/workspace/eks-monitoring/monitoring (master) $
```

Use port forwarding to view Prometheus dashboard at http://localhost:9090

$ kubectl --namespace=tiller port-forward deploy/prometheus-server 9090

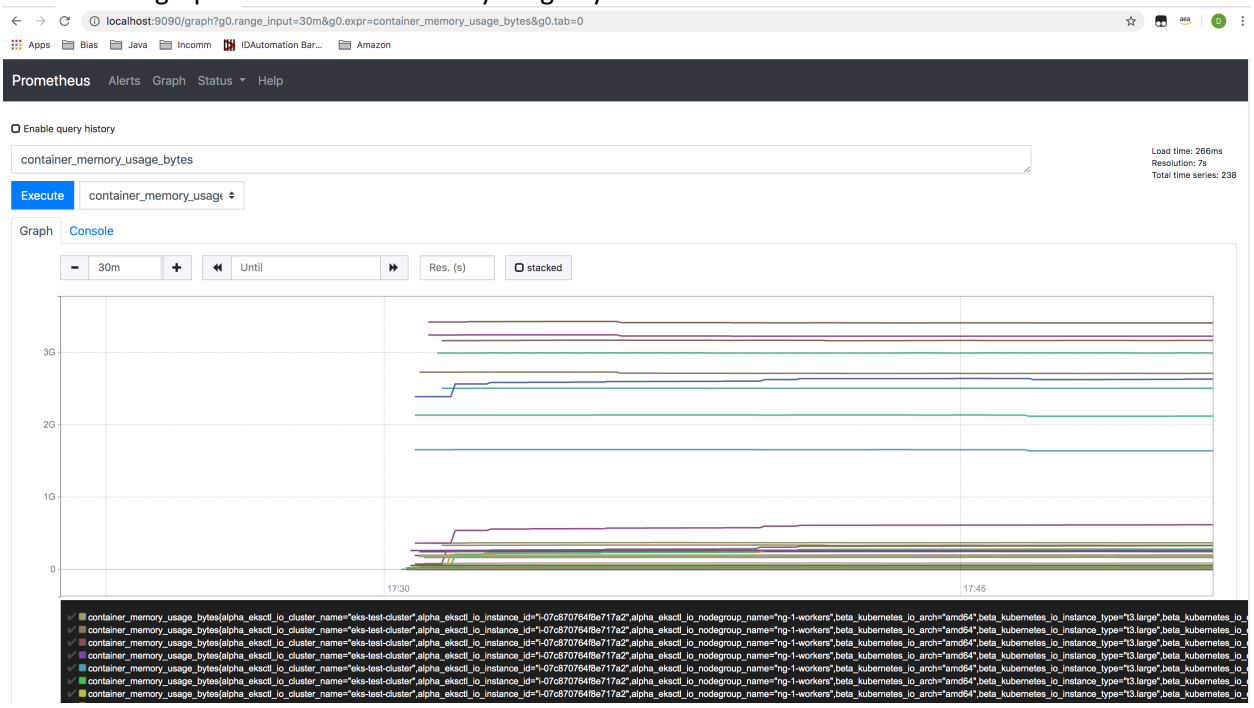
```
38f9d377667a:Downloads damodaw$ kubectl --namespace=tiller port-forward deploy/prometheus-server 9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
```

Use port forwarding to view Alert Manager dashboard at http://localhost:9093

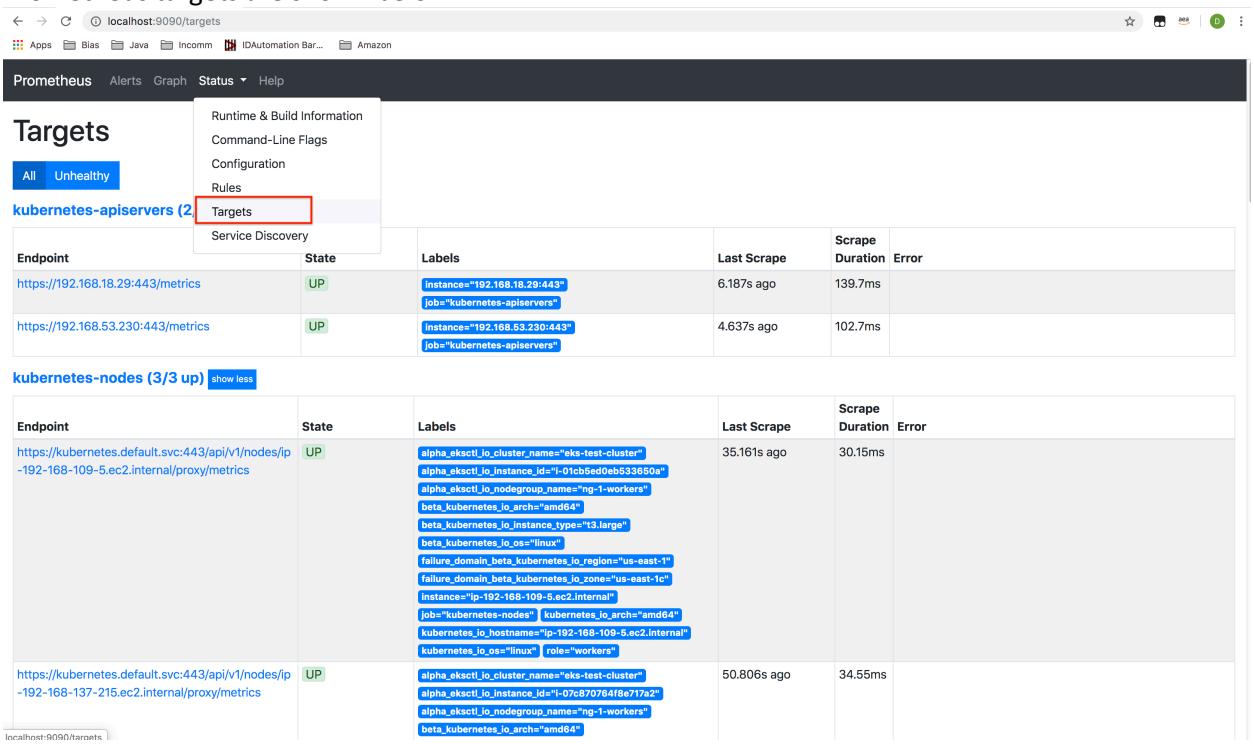
$ kubectl --namespace=tiller port-forward deploy/prometheus-alertmanager 9093

```
38f9d377667a:~ damodaw$ kubectl --namespace=tiller port-forward deploy/prometheus-alertmanager 9093
Forwarding from 127.0.0.1:9093 -> 9093
Forwarding from [::1]:9093 -> 9093
```

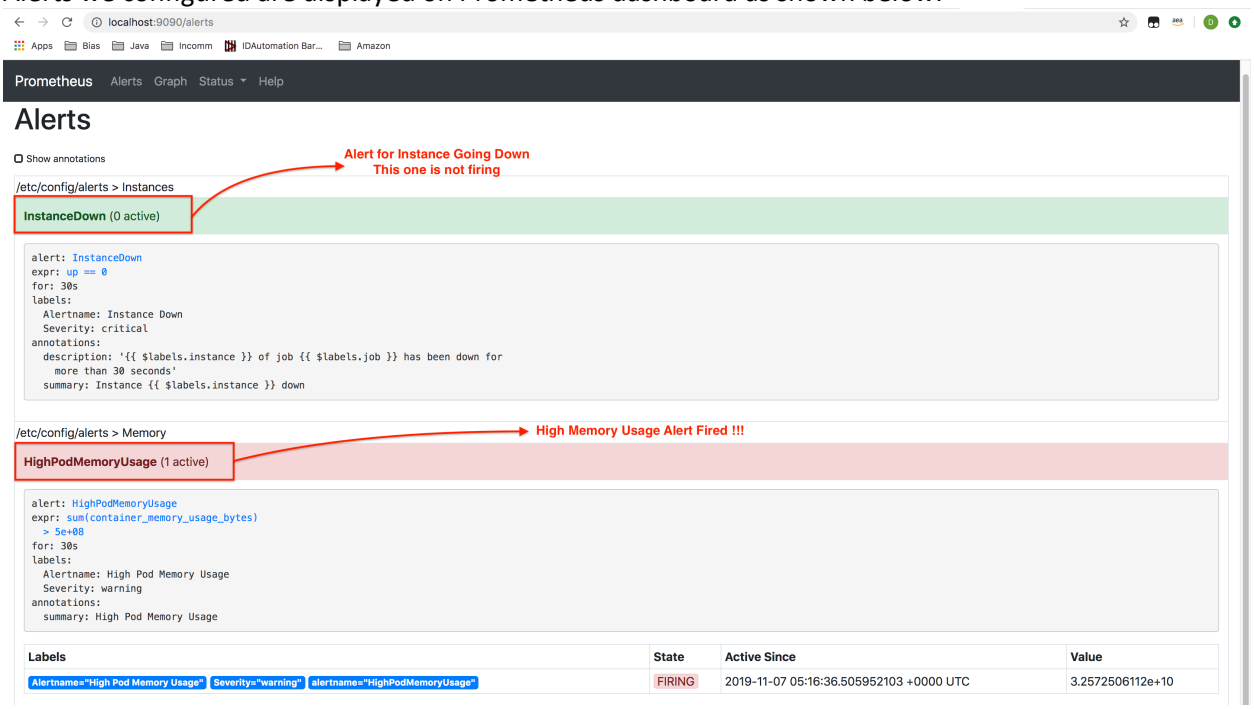
Prometheus graph for container memory usage bytes is shown below



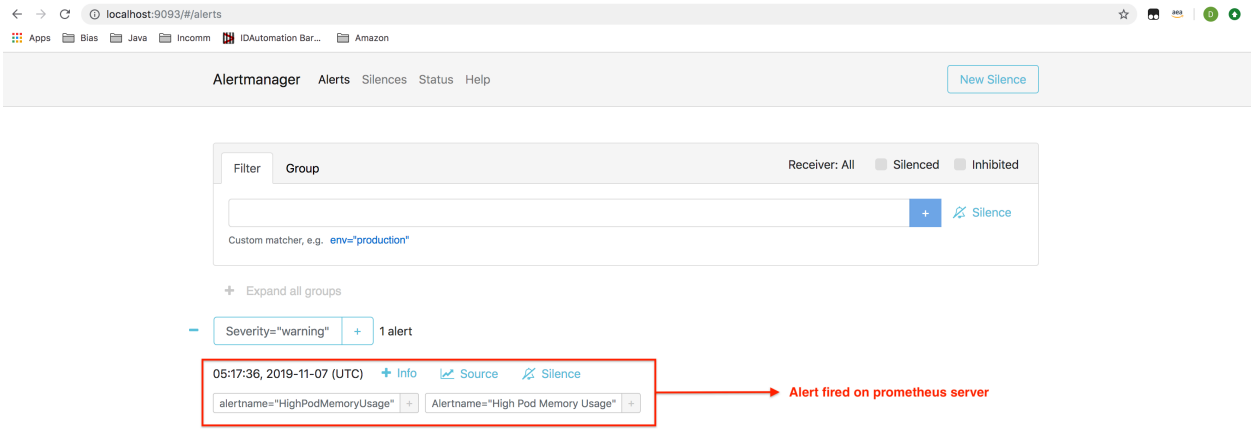
Prometheus targets are shown below



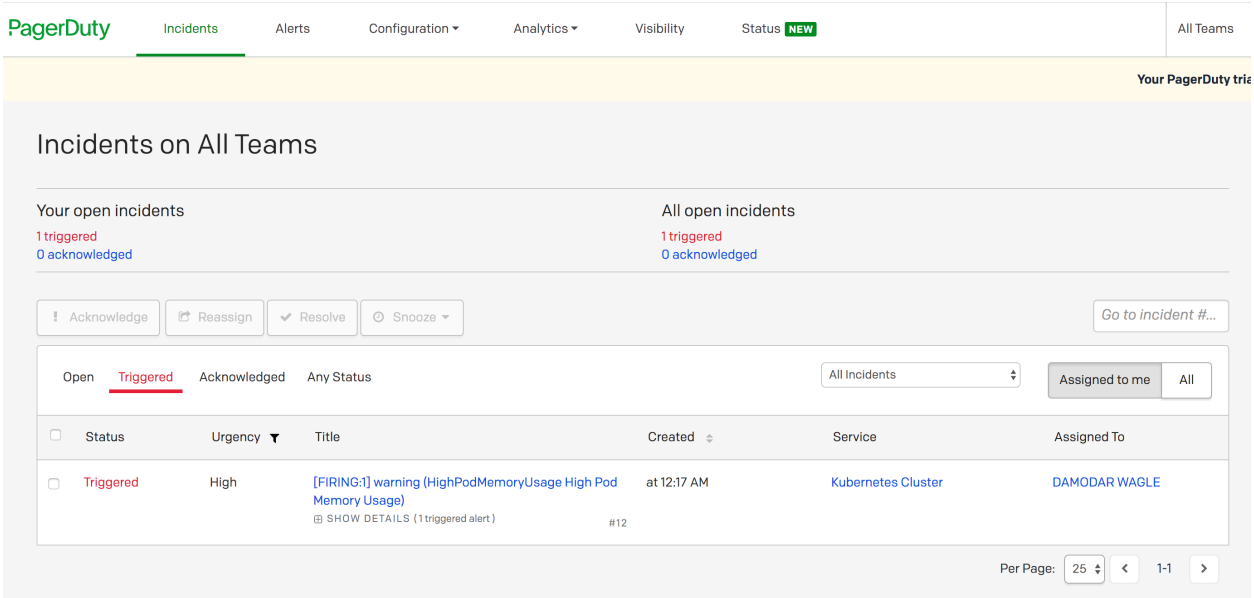
Alerts we configured are displayed on Prometheus dashboard as shown below.



When alerts fire, Prometheus server pushes them to Alert Manager which handles delivery channel for those alerts. In our case the delivery channel is Pager Duty. Below screenshot shows fired alert on Alert Manager dashboard.



Alert Manager send the alert to pagerduty.com where you should be able to see it as shown below.



For Grafana dashboard we have an internal load balancer. So I will use a windows EC2 machine to access it as shown below. User name for dashboard is admin. Password can be retrieved using following command.

$ kubectl get secret --namespace tiller grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

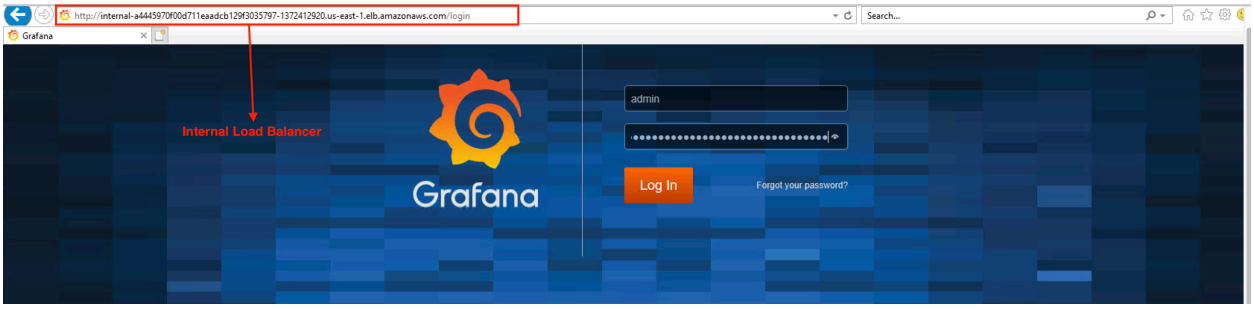
```
Admin:~/workspace/eks-monitoring/monitoring (master) $ kubectl get secret --namespace tiller grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
3zpcI3LYn0ZcJt8Yj2A4mRSSltAuVJ9MeUsDb9OJ
```

Load balancer url can be obtained using following command

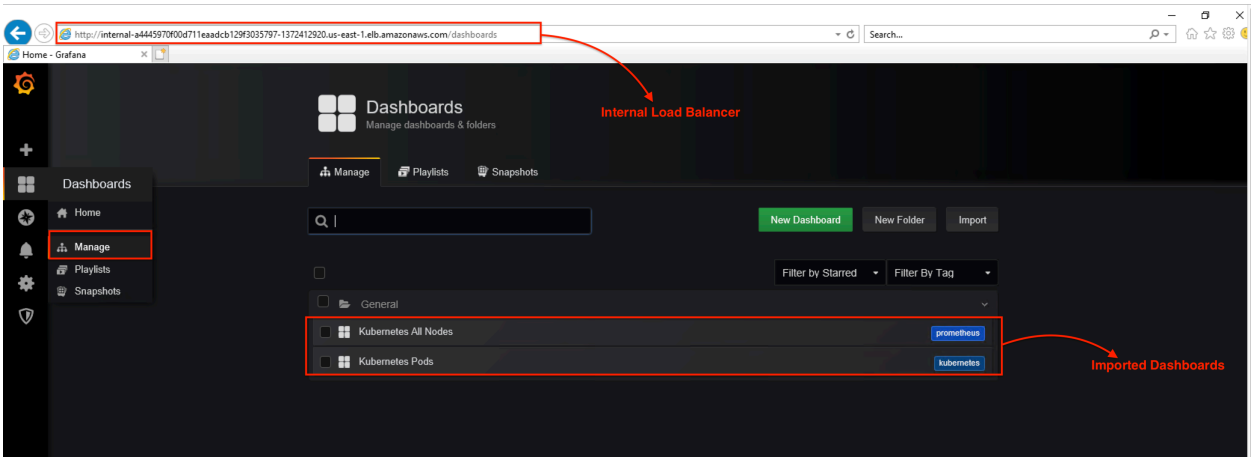
$ export ELB=$(kubectl get svc -n tiller grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
$ echo http://$ELB

```
Admin:~/workspace/eks-monitoring/monitoring (master) $ export ELB=$(kubectl get svc -n tiller grafana -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')
Admin:~/workspace/eks-monitoring/monitoring (master) $ echo "http://$ELB"
http://internal-a17ffa1a700bb11eaadcb129f3035797-1110862669.us-east-1.elb.amazonaws.com
Admin:~/workspace/eks-monitoring/monitoring (master) $
```

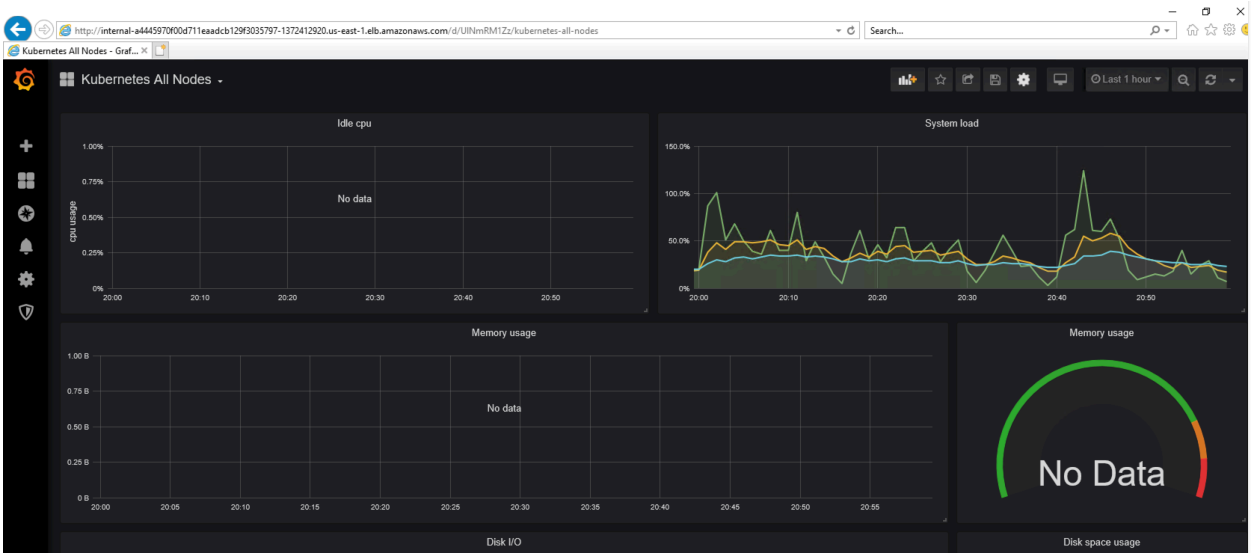
Grafana Landing Page



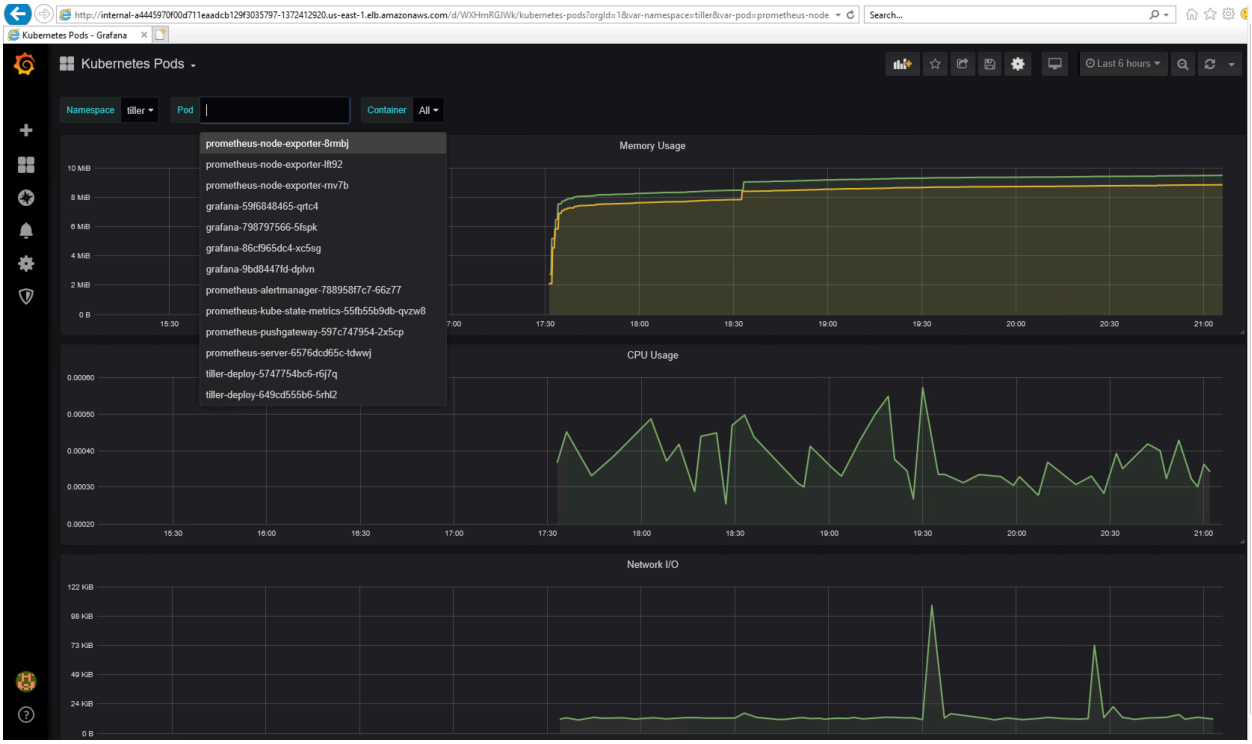
Grafana has been pre-configured to import 2 dashboards **Kubernetes Pods** and **Kubernetes All Nodes**
These dashboards should be displayed under Dashboards -> Manage



Kubernetes All Nodes Dashboard



Kubernetes Pods Dashboard



Here I would like to highlight important configuration updates that we performed via scripts to set up the alerts. Following generated manifests had some changes.
prometheus/templates/alertmanager-configmap.yaml
prometheus/templates/server-configmap.yaml

These changes pertain to setting up alerts to trigger pager duty. There is an additional step for signing up with www.pagerduty.com which we are not going to be covering as part of this tutorial. Please follow instructions on their website to retrieve ***service_key*** that you would need to get alerts from pager-duty on your phone. There is a 14-day trial period you can leverage. Prometheus integration guide can be found **here**.

Changes to above files are highlighted as shown in the screenshots below.

install-without-helm.s × prometheus.yaml × Untitled1 alertmanager-configı ×

```yaml
---
# Source: prometheus/templates/alertmanager-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    component: "alertmanager"
    app: prometheus
    release: prometheus
    chart: prometheus-9.2.0
    heritage: Tiller
  name: prometheus-alertmanager
data:
  alertmanager.yml: |-
    global:
      resolve_timeout: 5m
    receivers:
    - name: pagerduty-receiver
      pagerduty_configs:
      - service_key: [redacted]
    route:
      group_by: ['Severity']
      group_interval: 5m
      group_wait: 10s
      receiver: pagerduty-receiver
      repeat_interval: 5m
```

Add pager-duty receiver and global route for all alerts

service_key can be retrieved from your account on pager-duty.com

```yaml
---
# Source: prometheus/templates/server-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    component: "server"
    app: prometheus
    release: prometheus
    chart: prometheus-9.2.0
    heritage: Tiller
  name: prometheus-server
data:
  alerts: |-
    groups:
    - name: Instances
      rules:
      - alert: InstanceDown
        annotations:
          description: '{{ $labels.instance }} of job {{ $labels.job }} has been down
            for more than 30 seconds.'
          summary: Instance {{ $labels.instance }} down
        expr: up == 0
        for: 30s
        labels:
          Severity: critical
          Alertname: Instance Down
    - name: Memory
      rules:
      - alert: HighPodMemoryUsage
        expr: sum(container_memory_usage_bytes) > 500000000
        for: 30s
        labels:
          Severity: warning
          Alertname: High Pod Memory Usage
        annotations:
          summary: High Pod Memory Usage

  prometheus.yml: |-
    global:
      evaluation_interval: 1m
      scrape_interval: 1m
      scrape_timeout: 10s
```

Add Alert Rules

```
205        - action: replace
206          source_labels:
207          - __meta_kubernetes_pod_name
208          target_label: kubernetes_pod_name
209
210      alerting:
211        alertmanagers:
212        - kubernetes_sd_configs:
213            - role: pod
214          tls_config:
215            ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
216          bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
217          relabel_configs:
218          - source_labels: [__meta_kubernetes_namespace]
219            regex: prometheus
220            action: keep
221          - source_labels: [__meta_kubernetes_pod_label_app]
222            regex: prometheus
223            action: keep
224          - source_labels: [__meta_kubernetes_pod_label_component]
225            regex: alertmanager
226            action: keep
227          - source_labels: [__meta_kubernetes_pod_container_port_number]
228            regex:
229            action: drop
230
231      alerting:
232        alertmanagers:
233        - scheme: http
234          api_version: v1
235          path_prefix: /
236          static_configs:
237          - targets:
238            - "prometheus-alertmanager:80"
239
240    rules: |-
241      {}
242
```

Remove this alerting section

Add this alerting section