**AWS - EKS**

**Pre-requisites:**

**AWS CLI:** The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool to download and configure, you can control multiple AWS services **Used for Eksctl to grab authentication token (version >1.16.156 of CLI)**

**eksctl :** eksctl is a simple CLI tool for creating clusters on EKS - Amazon’s new managed Kubernetes service for EC2. It is written in Go, uses CloudFormation, was created by [Weaveworks](https://www.weave.works/) and it welcomes contributions from the community

**Kubectl :** Kubectl is a command line tool for controlling Kubernetes clusters. kubectl looks for a file named config in the $HOME/.kube directory. You can specify other [kubeconfig](https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/) files by setting the KUBECONFIG environment variable or by setting the [--kubeconfig](https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/) flag.

## **AWS CLI**

|  |
| --- |
| * yum install centos-release-scl * yum install rh-python36 * su - autoteam * sudo yum install python34 python34-pip * scl enable rh-python36 bash * pip3 install --user awscli |

**Configure credentials**

|  |
| --- |
| * mkdir .aws * aws configure set aws\_access\_key\_id <id> * aws configure set aws\_secret\_access\_key <key> * aws configure set default.region us-west-2 * [autoteam@HYD-DEVOPSCOS02 .aws]$ aws --version   aws-cli/1.16.309 Python/3.6.9 Linux/3.10.0-1062.4.1.el7.x86\_64 botocore/1.13.45 |

## **To install or upgrade eksctl on Linux using curl**

Download and extract the latest release of eksctl with the following command.

|  |
| --- |
| * curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest\_release/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp * sudo mv /tmp/eksctl /usr/local/bin * eksctl version |

## **3.Install kubectl on Linux**

|  |
| --- |
| * curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl * chmod +x ./kubectl * sudo mv ./kubectl /usr/local/bin/kubectl * kubectl version |

## **Iam authenticator**

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster through the [AWS IAM Authenticator for Kubernetes](https://github.com/kubernetes-sigs/aws-iam-authenticator). You can configure the  kubectl client to work with Amazon EKS by installing the AWS IAM Authenticator for Kubernetes and modifying your kubectl configuration file to use it for authentication.

|  |
| --- |
| * curl -o aws-iam-authenticator <https://amazon-eks.s3-us-west-2.amazonaws.com/1.14.6/2019-08-22/bin/linux/amd64/aws-iam-authenticator> * chmod +x ./aws-iam-authenticator * mkdir -p $HOME/bin && cp ./aws-iam-authenticator $HOME/bin/aws-iam-authenticator && export PATH=$HOME/bin:$PATH * echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc * aws-iam-authenticator help |

## **Image Registry:**

Commands to push docker images to ECR registry from util server

|  |
| --- |
| * $(aws ecr get-login --no-include-email --region us-west-2) * docker tag 192.168.136.93:5000/ha-api:12.19 534342615230.dkr.ecr.us-west-2.amazonaws.com/eksrepo/mc\_api:12.19.2 * docker push 534342615230.dkr.ecr.us-west-2.amazonaws.com/eksrepo/mc\_api:12.19.2 * Push commands for eksrepo/mc\_web * $(aws ecr get-login --no-include-email --region us-west-2) * #docker build -t eksrepo/mc\_web . * docker tag 192.168.136.93:5000/ha-web:12.19 534342615230.dkr.ecr.us-west-2.amazonaws.com/eksrepo/mc\_web:12.19.2 * docker push 534342615230.dkr.ecr.us-west-2.amazonaws.com/eksrepo/mc\_web:12.19.2 |

Private docker registry:

<http://192.168.136.93:8081/login/auth> (user name: admin/\*\*\*\*\*)

## **EKS Cluster creation:**

Kubernetes source code:

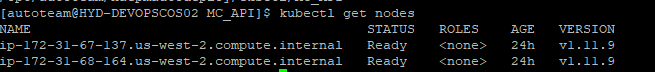
|  |
| --- |
| * git clone [https://<**ID**>@bitbucket.org/hostanalytics/haepmautodeploy.git](https://%3cID%3e@bitbucket.org/hostanalytics/haepmautodeploy.git) * cd haepmautodeploy/eksctl/ * eksctl create cluster -f cluster.yaml |

## **Delete node Group in eks cluster**

|  |
| --- |
| eksctl delete nodegroup --name=dev-m5-4xlarge --cluster=cluster-eks |

## **Create Node group in eks cluster**

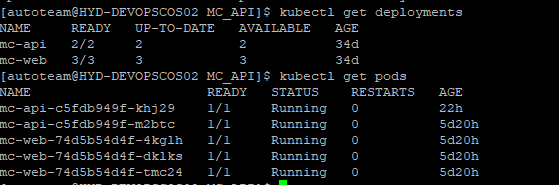
|  |
| --- |
| eksctl create nodegroup -f ../cluster.yaml |



## **Kubernetes Object: Deployment**

**Deployments** represent a set of multiple, identical Pods with no unique identities. A **Deployment** runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive.. **Deployments** are managed by the **Kubernetes Deployment** controller.

|  |
| --- |
| Kubectl apply -f haepmautodeploy/eksctl/MC\_API/deployment.yaml  Kubectl get deployments |



## **Kubernetes Object: Service**

Service is an abstraction which defines a logical set of Pods and a policy by which to access. The set of Pods targeted by a Service is usually determined by a [selector](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/).

There are [four types](https://v1-13.docs.kubernetes.io/docs/concepts/services-networking/#publishing-services-service-types) of Kubernetes services:

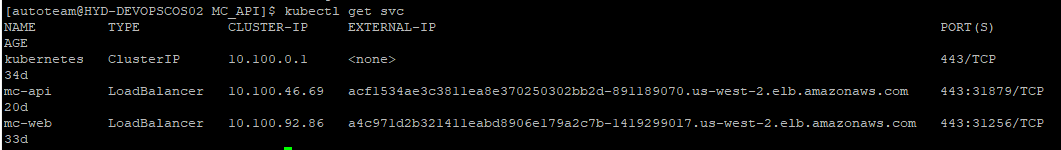
**ClusterIP.** This default type exposes the service on a cluster-internal IP. You can reach the service only from *within* the cluster.

**NodePort.** This type of service exposes the service on each node’s IP at a static port. A ClusterIP service is created automatically, and the NodePort service will route to it. From *outside* the cluster, you can contact the NodePort service by using “<NodeIP>:<NodePort>”.

**LoadBalancer.** This service type exposes the service externally using the load balancer of your cloud provider. The external load balancer routes to your NodePort and ClusterIP services, which are created automatically.

**ExternalName.** This type maps the service to the contents of the externalName field (e.g., foo.bar.example.com). It does this by returning a value for the CNAME record.

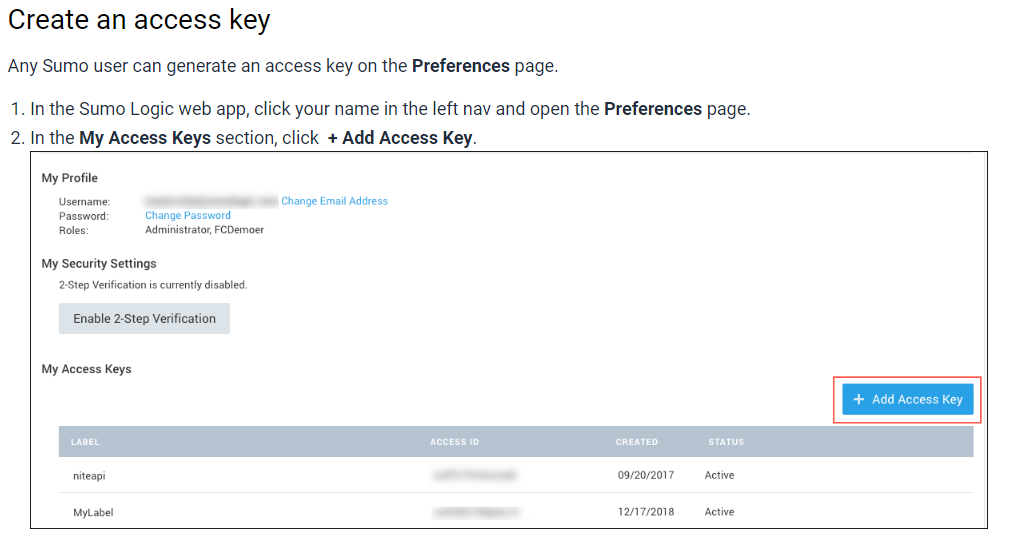
|  |
| --- |
| Kubectl apply -f haepmautodeploy/eksctl/MC\_API/service.yaml  Kubectl get svc |

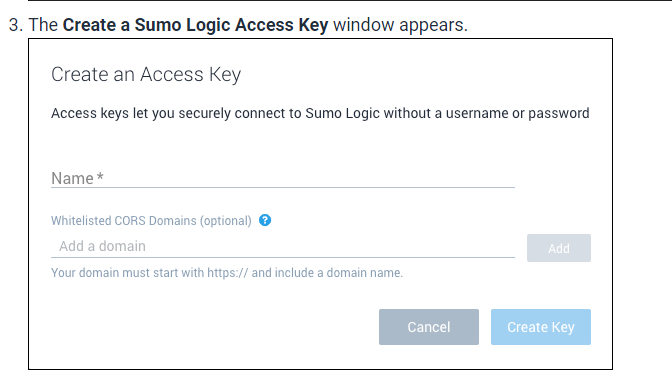


# Sumo logic Integration:

**Generate Access id and Access key:**

you can manage access keys created by other Sumo users on the **Administration > Security > Access Keys** page.





* Enter a name for the access key in the **Name** field.
* you can define one or more domains that may use the access key to access Sumo APIs. Enter a domain in the **Whitelisted CORS Domains** field and click **Add**. (optional)

Example :

Access ID : su8PX9gDauuT4m

Access Key : WEBv71uSAHeKpe6zq9OYcnnw6GbsgpBf00JhIukNi256u5OfLjrLYYE6ll0gTbzH

First, you'll need to set up the relevant [fields](https://help.sumologic.com/Manage/Fields) in the Sumo Logic UI. This is to ensure your logs will be tagged with the correct metadata.

* cluster
* container
* deployment
* host
* namespace
* node
* pod
* service

# 

# chart with Helm 3

1. Helm 3 no longer creates namespaces. You will need to create a Sumo Logic namespace first.

kubectl create namespace sumologic

1. Change your kubectl context to the sumologic namespace. You can use a tool like [kubens](https://github.com/ahmetb/kubectx) or kubectl to do this.

kubectl config set-context --current --namespace=sumologic

1. Install the chart using Helm 3. The helm install command has changed slightly in Helm 3.

|  |
| --- |
| helm install collection sumologic/sumologic --set sumologic.accessId=<SUMO\_ACCESS\_ID> --set sumologic.accessKey=<SUMO\_ACCESS\_KEY> --set prometheus-operator.prometheus.prometheusSpec.externalLabels.cluster="<MY\_CLUSTER\_NAME>" --set sumologic.clusterName="<MY\_CLUSTER\_NAME>" |

helm install collection sumologic/sumologic --set sumologic.accessId=su8PX9gDauuT4m --set sumologic.accessKey=WEBv71uSAHeKpe6zq9OYcnnw6GbsgpBf00JhIukNi256u5OfLjrLYYE6ll0gTbzH --set prometheus-operator.prometheus.prometheusSpec.externalLabels.cluster="cluster-eks" --set sumologic.clusterName="cluster-eks"

Validation:

kubectl --namespace sumologic get pods -l "release=collection"

## **Uninstalling Chart Using Helm 3**

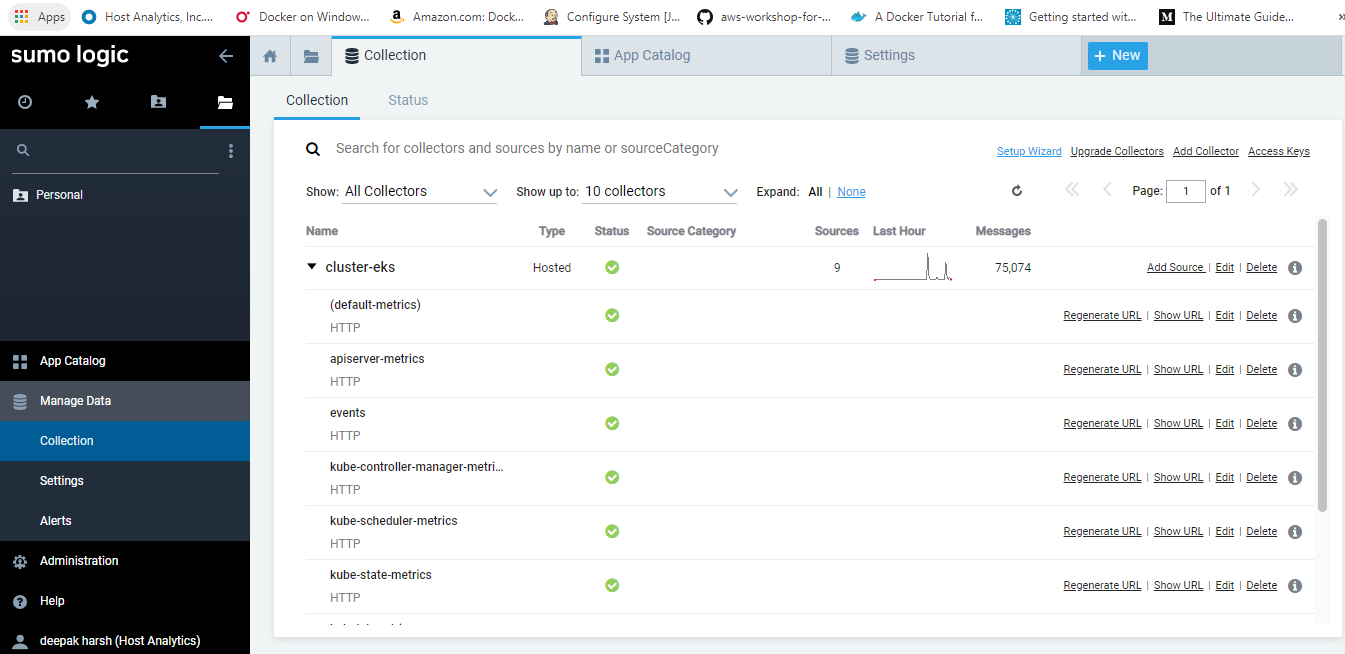
* Delete without preserving history

helm del collection

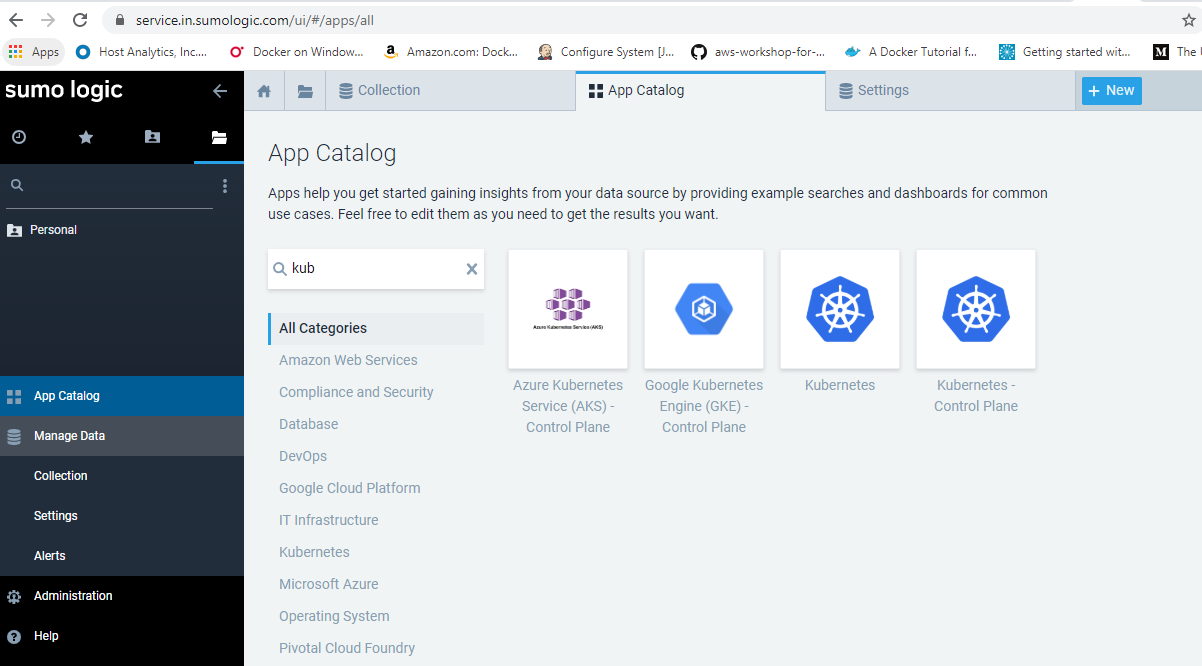
* Delete and preserve history

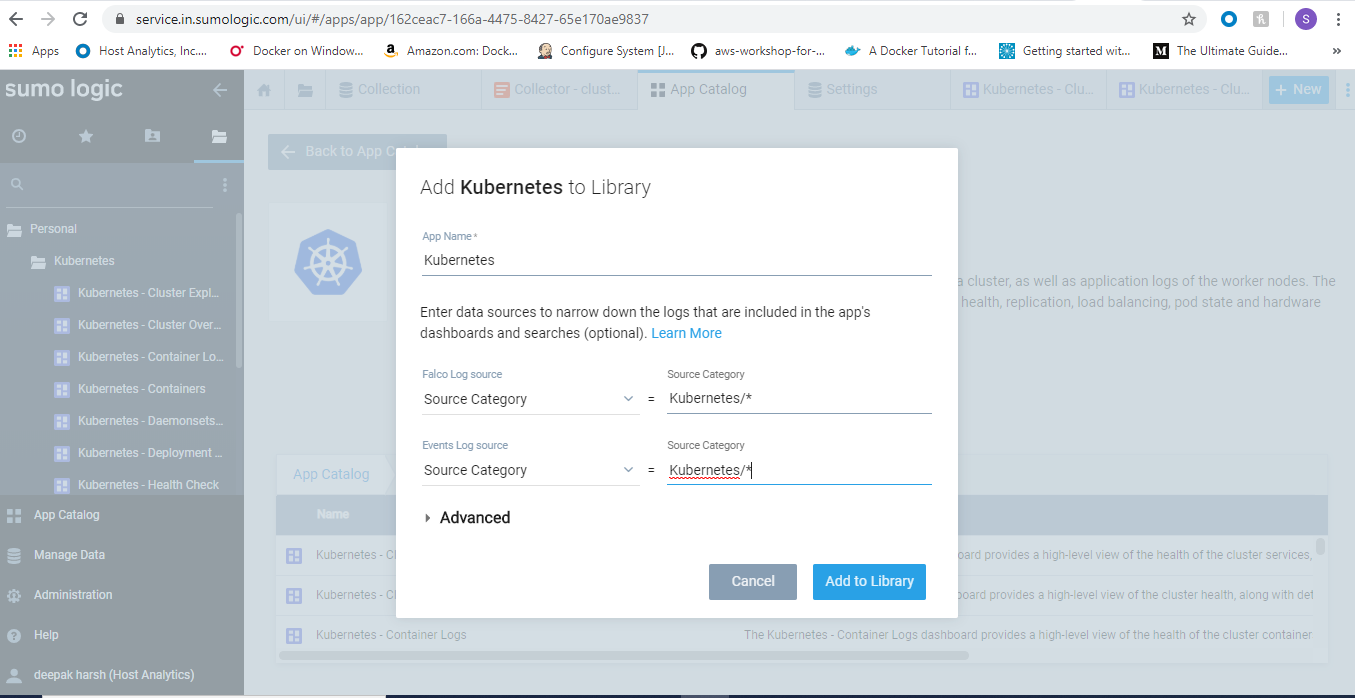
helm del --keep-history collection

**Configure Dashboards to monitor Kubernetes cluster:**

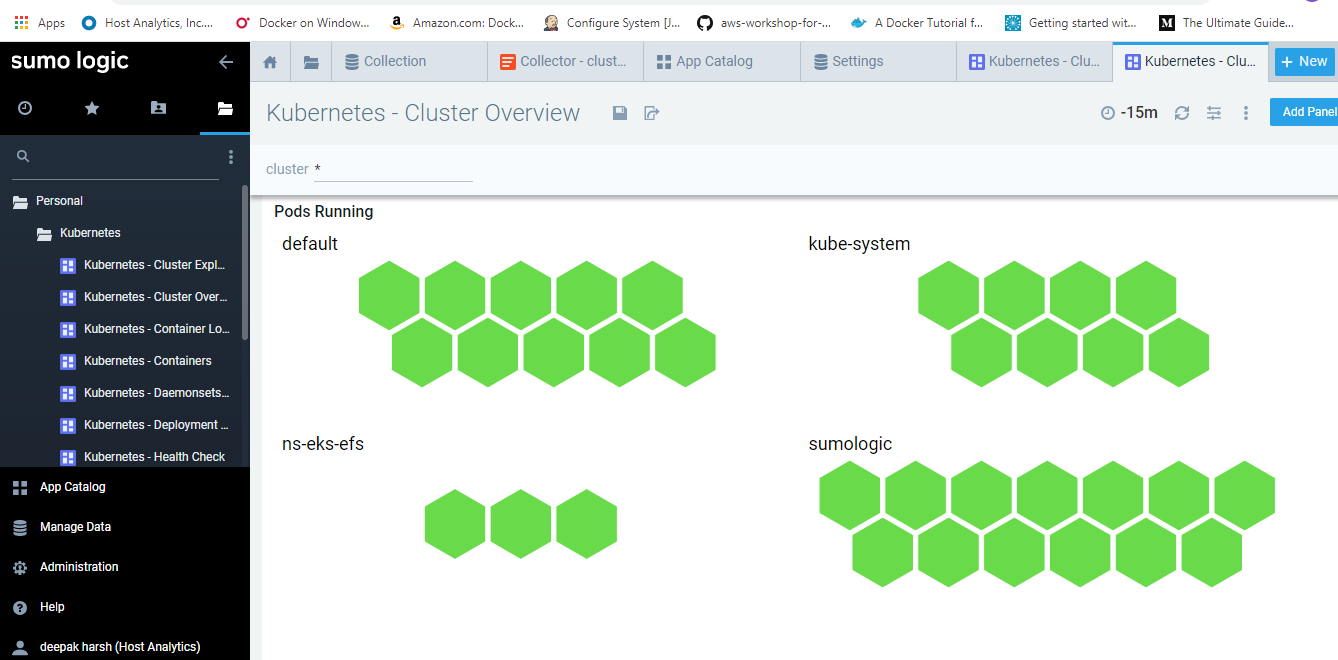


Configure Dashboards to monitor Kubernetes cluster:





**Dashboard**



**Helm 2 install**

curl -LO https://git.io/get\_helm.sh

$ chmod 700 get\_helm.sh

$ ./get\_helm.sh

$ helm version

Client: &version.Version{SemVer:"v2.16.4", GitCommit:"5e135cc465d4231d9bfe2c5a43fd2978ef527e83", GitTreeState:"clean"}

Server: &version.Version{SemVer:"v2.16.4", GitCommit:"5e135cc465d4231d9bfe2c5a43fd2978ef527e83", GitTreeState:"clean"}

<https://v2.helm.sh/docs/using_helm/#installing-helm>

**Context switching:**

kubectl config set-context --current --namespace=sumologic

**Delete Struck NS**

kubectl get namespace "ns-eks-efs" -o json \

| tr -d "\n" | sed "s/\"finalizers\": \[[^]]\+\]/\"finalizers\": []/" \

| kubectl replace --raw /api/v1/namespaces/ns-eks-efs/finalize -f -

SUMO ALERT MANGEMENT

<https://github.com/SumoLogic/sumologic-kubernetes-collection/blob/master/deploy/docs/Helm3.md>  
<https://github.com/SumoLogic/sumologic-kubernetes-collection/blob/master/deploy/docs/Helm3.md>

<https://github.com/SumoLogic/sumologic-kubernetes-collection/blob/master/deploy/docs/Installation_with_Helm.md>

<https://help.sumologic.com/Metrics/Metric-Queries-and-Alerts/Metrics_Monitors_and_Alerts>

<https://help.sumologic.com/Manage/Connections-and-Integrations/Webhook-Connections/Set_Up_Webhook_Connections>

I have checked below is what you can do to get the logs from custom path

See the below link  
[https://github.com/SumoLogic/sumologic-kubernetes-collection/blob/master/deploy/docs/Non\_Helm\_Installation.md#customize-configuration](https://urldefense.proofpoint.com/v2/url?u=https-3A__github.com_SumoLogic_sumologic-2Dkubernetes-2Dcollection_blob_master_deploy_docs_Non-5FHelm-5FInstallation.md-23customize-2Dconfiguration&d=DwMFaQ&c=fabvTBoJTKq8R3bbic--Bj6RE_xWwwtRaR85HeJrISg&r=p7iEt-xwJGGiWlUgPm1QeCpnbm1b5XdosYiCUliGy30&m=6IaCMpgU5Oge2XcT-tdkSRr5L0Gn8423qN2yINcFIhs&s=FpbMoHf5pIAwG-EWoM7zxC5b4wyb0jgUhodBrQvxTUM&e=)

> ###  
>  
> ## Deploy FluentBit  
>  
> In this step, you will deploy FluentBit to forward logs to Fluentd.  
>  
> Run the following commands to download the FluentBit `fluent-bit-overrides.yaml` file and install `fluent-bit`  
>  
> $ curl -LJO [https://raw.githubusercontent.com/SumoLogic/sumologic-kubernetes-collection/v0.13.0/deploy/helm/fluent-bit-overrides.yaml](https://urldefense.proofpoint.com/v2/url?u=https-3A__raw.githubusercontent.com_SumoLogic_sumologic-2Dkubernetes-2Dcollection_v0.13.0_deploy_helm_fluent-2Dbit-2Doverrides.yaml&d=DwMFaQ&c=fabvTBoJTKq8R3bbic--Bj6RE_xWwwtRaR85HeJrISg&r=p7iEt-xwJGGiWlUgPm1QeCpnbm1b5XdosYiCUliGy30&m=6IaCMpgU5Oge2XcT-tdkSRr5L0Gn8423qN2yINcFIhs&s=Lw643HJfLClTdrgGucolRc6oavzbwgeP4pHJqKKjCYk&e=)  
> $ helm fetch stable/fluent-bit --version 2.8.1  
> $ helm template fluent-bit-2.8.1.tgz --name fluent-bit --namespace=sumologic -f fluent-bit-overrides.yaml > fluent-bit.yaml  
> $ kubectl apply -f fluent-bit.yaml  
>  
> \*\*NOTE\*\* Refer to the requirements.yaml for the currently supported version.

Refer to file "[https://raw.githubusercontent.com/SumoLogic/sumologic-kubernetes-collection/v0.13.0/deploy/helm/fluent-bit-overrides.yaml](https://urldefense.proofpoint.com/v2/url?u=https-3A__raw.githubusercontent.com_SumoLogic_sumologic-2Dkubernetes-2Dcollection_v0.13.0_deploy_helm_fluent-2Dbit-2Doverrides.yaml&d=DwMFaQ&c=fabvTBoJTKq8R3bbic--Bj6RE_xWwwtRaR85HeJrISg&r=p7iEt-xwJGGiWlUgPm1QeCpnbm1b5XdosYiCUliGy30&m=6IaCMpgU5Oge2XcT-tdkSRr5L0Gn8423qN2yINcFIhs&s=Lw643HJfLClTdrgGucolRc6oavzbwgeP4pHJqKKjCYk&e=)"

You would see below section in this file

[INPUT]  
Name tail  
\*\*Path /var/log/containers/\*.log\*\*  
Multiline On  
Parser\_Firstline multi\_line  
Tag containers.\*  
Refresh\_Interval 1  
Rotate\_Wait 60  
Mem\_Buf\_Limit 5MB  
Skip\_Long\_Lines On  
DB /tail-db/tail-containers-state.db  
DB.Sync Normal

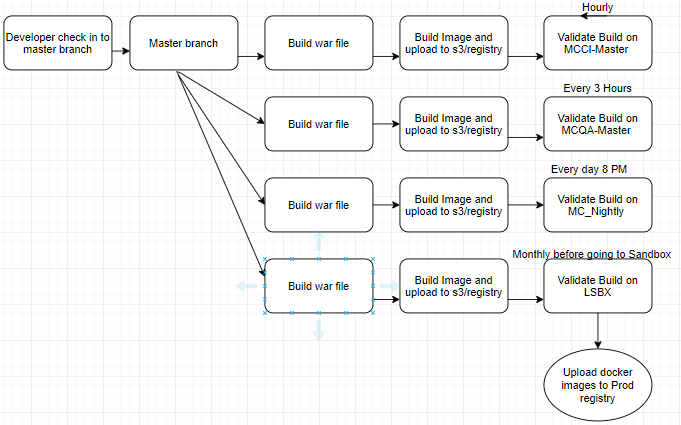
You have change the "path" to your desired custom path. But if you make this change then you will stop seeing the container logs.

In that case if you need to collect the container logs in addition to from local path you will need to add a new [INPUT] section like above.

Again I will emphasis on below  
\*\*Helm 3 compatibility is in the early stages and is not fully tested or supported. Please refer to this guide for more information on Helm 3. We recommend you thoroughly test the use of Helm 3 in your pre-production environments before use.\*\*

Thanks & Regards,  
Shobhit

**Build and Release cycle.**



# Dev Environment Setup

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **WEB** |  | **API** | | **DB** | |
|  | **CORE** | **RAM** | **CORE** | **RAM** | **CORE** | **RAM** |
| **QA** | 4 | 16 | 4 | 40 | 4 | 46 |
| **LSBX** | 4 | 16 | 4 | 40 | 4 | 46 |
| **OPS** | 4 | 16 | 8 | 48 | 12 | 32 |
| **PERF** | 4 | 16 | 8 | 48 | 12 | 32 |
| **TOTAL** | 16 | 64 | 24 | 176 | 32 | 156 |

Assuming Kubernetes cluster is already setup and Metric server is installed.

|  |  |
| --- | --- |
| **Type** | **IP** |
| Kubernetes master/node1 | 192.168.136.16 |
| Node 2 | 192.168.136.20 |
| Node3 | 192.168.136.47 |
| NFS | 192.168.136.83 |
| Registry | 192.168.136.87 |

**Steps to build environment:**

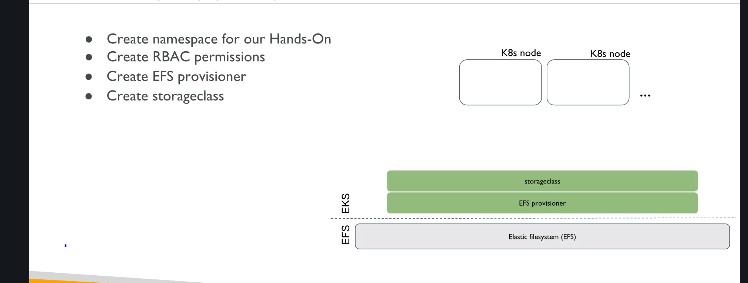
* Build DB VM(12 core and 32 GM RAM) for each Environment.
* Create Name space per environment type ( ex: Perf,ops,Lsbx.QA)
* Create NFS storage class and create persistent volume for both WEB and API specific to namespace.
* Perform Deployment and Add expose application using service type node port/Ingress controller specific to name space.
* Add Horizontal pod Autoscaler to auto scale instances.
* Create a Kubernetes Dashboard.
* Enable logging and monitoring ( Prometheus/grafana and sumologic)

Note: logic monitor is based on number of resources , it is better to use freeware for monitoring.

|  |  |  |
| --- | --- | --- |
| **Environment** | **WEB(2 container)** | **API(2 Container)** |
| Perf | 2 core/8 GB each | 4 core /24 GB each |
| OPS | 2 core/8 GB each | 4 core /24 GB each |
| LSBX | 1 core/8 GB each | 2 core /20 GB each |
| QA | 1 core/8 GB each | 2 core /20 GB each |
|  |  |  |

**Persistent volumes:**

* Enable EFS
* Create namespace and prepare stage
* Deploy application



# create namespace

```

kubectl create namespace modeldev

```

file ID - fs-1c65d2b6

DNS name- fs-1c65d2b6.efs.us-west-2.amazonaws.com

# create storage

The steps to execute are:

1. add RBAC to access EFS

2. create EFS provisioner

replace your \*EFS-ID\* and \*EFS-DNS name\* in file \_create-efs-provisioner.yaml\_

3. create PVCs

The corresponding commands are:

kubectl apply -f create-rbac.yaml --namespace=modeldev

kubectl apply -f create-efs-provisioner.yaml --namespace=modeldev

kubectl apply -f create-storage.yaml --namespace=modeldev