Implement Datadog APM (using the method of admission controller) in a Kubernetes environment, with the language being node.js.

*Introduction:*

<https://docs.datadoghq.com/tracing/guide/>

<https://docs.datadoghq.com/tracing/guide/tutorial-enable-java-admission-controller/>

<https://docs.datadoghq.com/tracing/guide/tutorial-enable-java-aws-eks/>

- APM (Application Performance Monitoring) is a practice that help monitor and analyze the performance of applications, ensuring they run smoothly and efficiently.

- **How to enable Datadog APM (tracing)?**

Manual instrumentation: You modify your app or container manually, add dd-trace library in code or add NODE\_OPTIONS="--require dd-trace/init” in your YAML.

Automatic instrumentation (via Admission Controller): Kubernetes automatically injects the necessary tracing configuration (environment variables, libraries, annotations) into your Pods, no manual edits to your Deployment.

It is no longer required to create a new image/version of your app just for monitoring purposes, nor is there a need to manually incorporate tracing libraries, as the Admission Controller will handle the injection process. Additionally, you do not have to insert Datadog variables or deploy a new image/version of your app.

- **What is an admission Controller?**

The Datadog Admission Controller is a component of the Datadog Cluster Agent.

If you installed the Datadog Agent using Helm chart or Datadog Operator, the Datadog Cluster Agent is enabled by default.

Datadog Operator is a Kubernetes Operator that enables you to deploy and configure the Datadog Agent in a Kubernetes environment.

- An Admission Controller is a Kubernetes feature that intercepts requests to the API server before objects (like Pods) are created or updated.

You can think of it as a gatekeeper that can mutate (modify) or validate Kubernetes resources on the fly.

There are two types:

Validating Admission Controller – checks if a Pod is allowed to run.

Mutating Admission Controller – modifies the Pod spec before it’s created.

== > Datadog uses the Mutating Admission Controller type.

- How to Instrument your app with Datadog Admission Controller

These are now the steps required to get your application instrumented, especially in K8s environments:

1. Install the Datadog Operator then install datadog agent
2. Install the Datadog Agent using helm
3. Deploy your application
4. Instrument Your Application/Use admission controller to inject Library

- **How it works step-by-step ?**

1. You apply a Deployment YAML that defines your Pod (e.g., your Node.js service).

2. The Kubernetes API receives the request and triggers any active MutatingAdmissionWebhooks.

3.The Datadog Admission Controller webhook intercept the pod creation because of your label/annotation (admission.datadoghq.com/enabled: "true"). It tells the Datadog Admission Webhook to mutate this Pod. Without it, the webhook ignores your Pod.

mutateUnlabelled: false: Enable injecting config without having the pod label: admisson.datadoghq.com/enabled="true"

4. It adds init containers alongside your app container pod to inject library.

admission.datadoghq.com/js-lib.version: v5.72.0: This instructs the Admission Controller to inject the library files (datadog-lib-js-init Init Container).

It will Inject environment variables such as DD\_AGENT\_HOST, DD\_TRACE\_AGENT\_URL and DD\_ENTITY\_ID to configure DogStatsD and APM tracer libraries into your application containers, allowing app to send traces.

It will Inject Datadog Unified Service Tags such as env (DD\_ENV), service (DD\_SERVICE) and version (DD\_VERSION) from application labels into the container environment variables.

admission.datadoghq.com/env: "true": This is useful to have, as it tells the webhook to inject the core networking environment variables and the Unified Service Tagging variables (DD\_ENV, DD\_SERVICE, etc.) if they were not already explicitly defined in your YAML.

6. It **modifies your Pod spec** to add NODE\_OPTIONS="-r dd-trace/init" (via the annotation you provided).

7. Kubernetes then launches the pod **with the injected library and environment variables**, so the app can send traces **without any code changes**.

8.Your application runs normally with Datadog’s tracer (dd-trace) is loaded automatically at startup.

9. The traces are sent to the Datadog Agent, then forwarded to Datadog APM UI

[*Prerequisites*](https://docs.datadoghq.com/tracing/guide/tutorial-enable-java-aws-eks/#prerequisites)*:*

\*\*\*A Datadog account and [organization API key](https://docs.datadoghq.com/account_management/api-app-keys/) (API and Application Keys)

\*\*\*Launch EC2 Instance

Instance Type: t2.medium

AMIs: Ubuntu

\*\*\*Create the IAM role having full access

Go to IAM -> Create role -> Select EC2 -> Give Full admin access "AdministratorAccess" -> Name the role EC2-ROLE-FOR-ACCESSING-EKS-CLUSTER

\*\*\*Attach the IAM role having full access

Go to EC2 -> Click on Actions on the left hand side -> Security -> Modify IAM role

\*\*\*Install Docker

apt install docker.io

\*\*\*Install aws iam authenticator

curl -o aws-iam-authenticator https://amazon-eks.s3.us-west-2.amazonaws.com/1.15.10/2020-02-22/bin/linux/amd64/aws-iam-authenticator

chmod +x ./aws-iam-authenticator

sudo mv ./aws-iam-authenticator /usr/local/bin

Test that the aws-iam-authenticator binary works: aws-iam-authenticator help

\*\*\*Install AWS CLI

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

apt install unzip

unzip awscliv2.zip

./aws/install

aws –version

\*\*\*Configure the CLI with my AWS credentials with

aws configure

I was then required to enter my:

AWS Access Key ID

AWS Secret Access Key

Default region name

Default output format: json

Confirm configuration with: aws configure list

\*\*\*Install and Setup Kubectl (node agent)

curl -LO https://dl.k8s.io/release/$(curl –Ls <https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl>

chmod +x ./kubectl

mv ./kubectl /usr/local/bin/kubectl

kubectl version --client

\*\*\*Install and Setup eksctl

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

mv /tmp/eksctl /usr/local/bin

eksctl version

\*\*\*Creating an Amazon EKS cluster using eksctl

Grant the IAM user the necessary least-privileged permissions in case we are working in production environment or admin permission

Create EKS cluster:

eksctl create cluster --name eks2 --version 1.29 --region eu-west-3 --nodegroup-name worker-nodes --node-type t2.medium --nodes 2 --nodes-min 2 --nodes-max 3

1. Name of the cluster : --eks2

2. Version of Kubernetes : --version 1.29

3. Region : --region eu-west-3

4. Nodegroup name/worker nodes : --nodegroup-name worker-nodes

5. Node Type : --nodegroup-type t2.medium

6. Number of nodes: --nodes 2

7. Minimum Number of nodes: --nodes-min 2

8. Maximum Number of nodes: --nodes-max 3

eksctl will set up an auto-scaling group that starts with 2 "t2.medium" instances, and can scale up to 3 instances if needed, and down to 2 if the load decreases.

in this case eks2 is the name we are giving to our EKS cluster. The EKS control plane for eks2 is managed by AWS It consists of the Kubernetes API server, scheduler, and etcd (the database)

AWS provides the control plane for us. and this instance from which we run this command, It's only used to configure and interact with the EKS cluster, but it does not become part of the control plane.

Verify cluster creation with: eksctl get cluster

kubectl get nodes

IF ANY ERROR ==> aws eks update-kubeconfig --region <region-code> --name <cluster-name>

\*\*\* In case we want to clean Up

eksctl delete cluster --name eks2 --region eu-west-3

\*\*\*Helm - Install by running these commands:

Helm is a package manager for Kubernetes. It is a package that contains all the necessary resource definitions and configurations to deploy an application, tool, or service onto a Kubernetes cluster. Think of it like a pre-packaged application with installation instructions for Kubernetes

curl -fsSL -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

chmod 700 get\_helm.sh

./get\_helm.sh

helm version

\*\*\*Configure Helm by running these commands:

- Add helm chart repository where the chart is located. Helm needs to be configured to know about this repository using this command:

helm repo add datadog <https://helm.datadoghq.com>

- Update helm chart repository

helm repo update

kubectl create namespace datadog

kubectl create secret generic datadog-secret --from-literal api-key=<DATADOG\_API\_KEY> -n datadog

Insert Datadog API key at <DATADOG\_API\_KEY>

\*\*\*Setup the sample node.js application

We will build a very simple HTTP service (Express) that responds to / and /health, with automatic tracing enabled via the dd-trace library

- In the aws instance Create this Project structure with the following content:

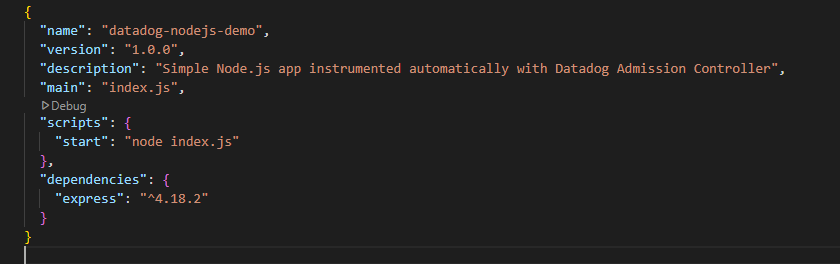
nodejs-app/

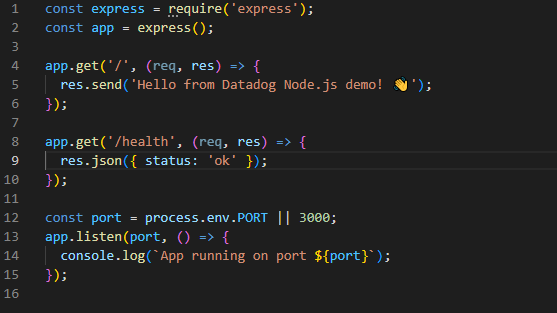
├── Dockerfile

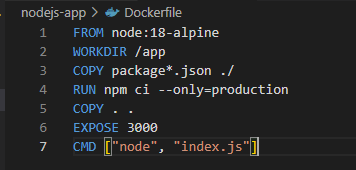
├── package.json

└── index.js

- package.json

- index.js

- - - Dockerfile



\*\*\* Install npm and needed library

apt install npm

cd nodejs-app

npm install

\*\*\*Build and upload the application image

Amazon ECR: a registry for EKS images

- Create ecr repositories:

aws ecr describe-repositories --repository-names nodejs-ecr || aws ecr create-repository --repository-name nodejs-ecr

- Build a Docker image for the sample app:

docker build -t nodejs-app:latest .

- Authenticate with ECR:

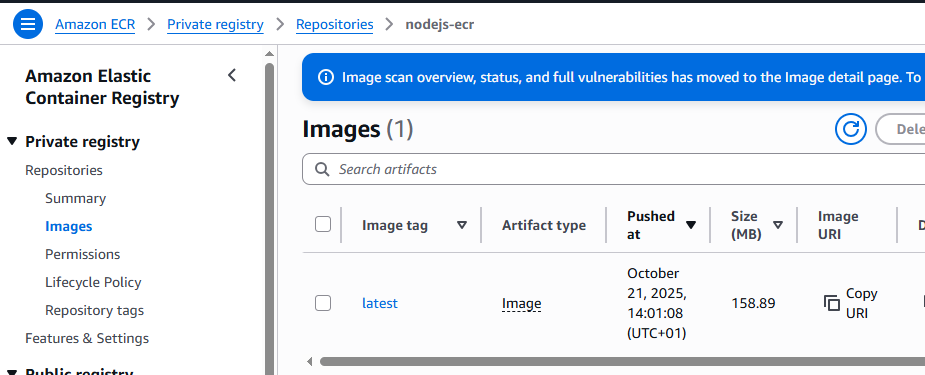
aws ecr get-login-password --region eu-west-3 | docker login --username AWS --password-stdin AWS-account-id.dkr.ecr.eu-west-3.amazonaws.com

- Tag the container with the ECR destination:

docker tag nodejs-app:latest $ECR\_REPOSITORY\_URI /nodejs-ecr:latest

- Upload the container to the ECR registry:

docker push $ECR\_REPOSITORY\_URI /nodejs-ecr:latest



Your application is now containerized and available for EKS clusters to pull.

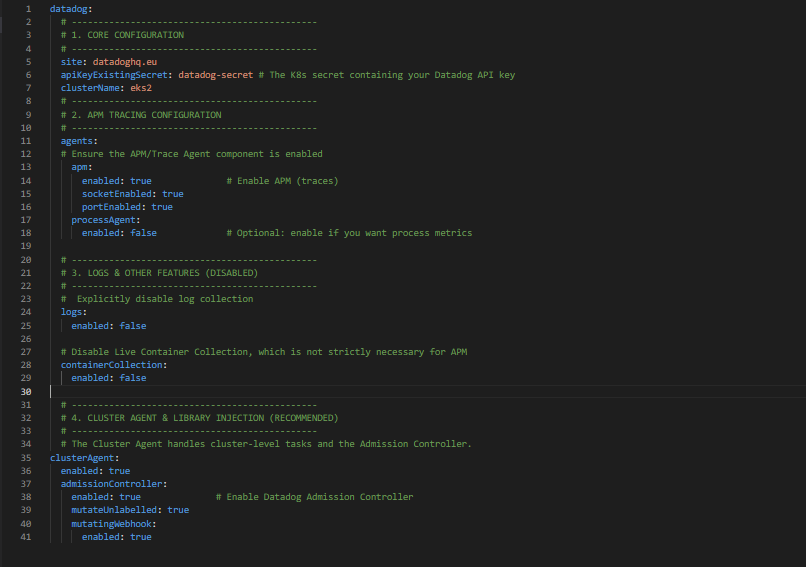
- Since, the focus here is to only turn on APM traces and metrics collection, we have to create a datadog agent values.yaml

We need to check the site configuration where we want our agent to send data it collects.

Configure the Datadog Admission Controller to inject a node.js tracing library to the app container by adding annotation to the pod.

In order to add Datadog standard tags (env, service, version) we have to provide value for this tags using POD labels. In case of using admission controller, datadog will automatically add this variables as env into the POD.

In the the config file, using a socket instead of HTTP or TCP ports is more efficient because it avoids network overhead inside the pod.



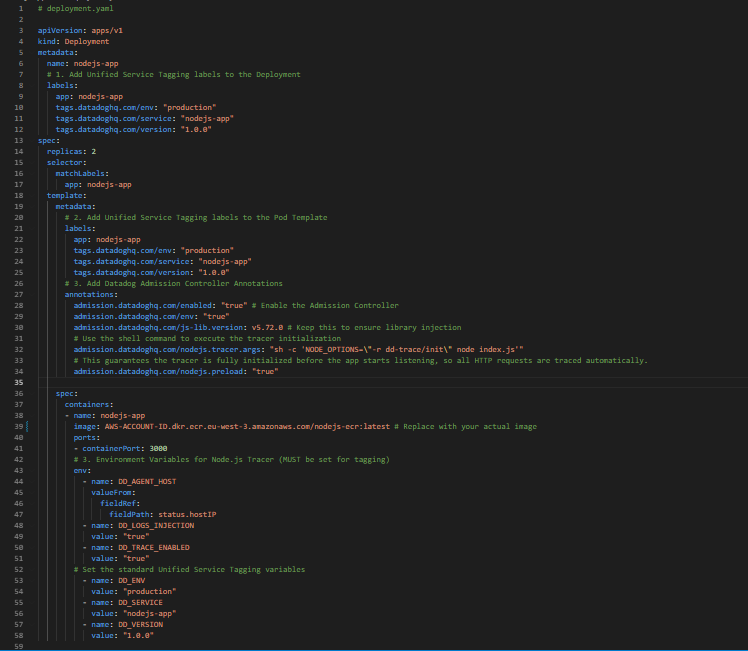
\*\*\*Install the agent and the datadog-values.yaml file contains configuration settings for the Datadog Agent, such as API keys, features to enable, integrations, and resource limits.

helm install datadog-agent -f datadog-values.yaml datadog/datadog --namespace datadog

- Verify the admission controller webhook

kubectl get MutatingWebhookConfiguration datadog-webhook -o yaml

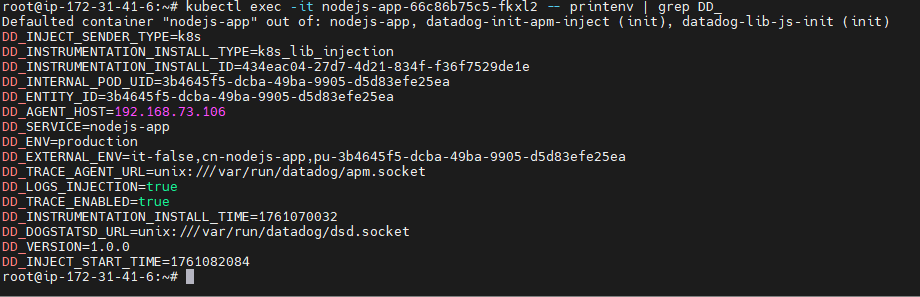
\*\*\*Apply the deployment



kubectl apply -f deployment.yaml

Exec into one of the POD and you will be able to see ENV injected by cluster admission controller. Since, this variables are automatically set, we need not to do anything extra here.

kubectl exec -it <new-nodejs-pod> -- printenv | grep DD\_



- check if the tracer is loaded:

kubectl exec –it POD\_NAME – sh

node -r dd-trace/init -e "const tracer = require('dd-trace'); console.log('Tracer loaded', tracer.\_tracer.\_enabled)"

\*\*\* Verify Metrics in Datadog APM

1. Generate traffic to the pod with “kubectl port-forward pod/<POD\_NAME> <LOCAL\_PORT:POD\_PORT>

the execute this command in another terminal many times: curl -sS http://localhost:3000/

In wait 5-10 min and you will be able to see data in APM section in datadog.

1. Open the Datadog web interface.
2. Navigate to APM > Services.
3. Look for the nodejs-app service

