Deploying Linkerd Service Mesh on AWS EKS for Secure Microservices

By

Mahendran Selvakumar

https://devopstronaut.com/

https://www.linkedin.com/in/mahendran-selvakumar/

What is a Service Mesh?

A **service mesh** is a tool designed to enhance **security**, **reliability**, and **observability** in cloud-native applications. It achieves this by transparently inserting these features at the **platform layer**, instead of requiring changes to the application layer.

This abstraction allows developers to focus on business logic while the service mesh handles concerns like traffic control, encryption, and monitoring. It is increasingly becoming a **standard component** of the cloud-native stack, particularly for Kubernetes adopters, where it simplifies complex service-to-service communication and ensures robust performance at scale.

Key Features of a Service Mesh

1. **Service-to-Service Communication:**
   - Manages how services within a cluster discover and communicate with one another.
   - Handles retries, timeouts, and load balancing automatically.
2. **Observability:**
   - Provides insights into service performance through metrics, logs, and distributed tracing.
   - Tools like Prometheus, Grafana, and Jaeger are often integrated.
3. **Traffic Management:**
   - Enables advanced traffic control, such as canary deployments, blue-green deployments, and traffic splitting.
4. **Security:**
   - Implements mutual TLS (mTLS) to encrypt traffic between services.
   - Supports policies for secure communication and access control.
5. **Resilience:**
   - Handles service failures gracefully with retries, circuit breaking, and fault injection.

What is Linkerd Service Mesh?

Linkerd is a lightweight, open-source service mesh specifically designed for Kubernetes environments. It focuses on providing **security, observability, and reliability** for cloud-native applications by managing service-to-service communication within a cluster. Developed as a CNCF (Cloud Native Computing Foundation) project, Linkerd is known for its simplicity, efficiency, and ease of use.

Key Features of Linkerd

1. **Security:**
   - **mTLS (Mutual TLS):** Encrypts traffic between services automatically, ensuring secure communication without modifying application code.
   - Automatic certificate rotation and strong identity guarantees.
2. **Observability:**

   - Provides **golden metrics** (latency, success rate, and request volume) for services.
   - Tools for real-time inspection of service behavior, such as linkerd stat and linkerd tap.
   - Integration with observability tools like Prometheus, Grafana, and Jaeger for metrics and tracing.
3. **Reliability:**
   - Traffic policies for fine-grained control over communication between services.
   - Automatic retries and timeouts to handle transient failures.
   - Traffic splitting for blue-green or canary deployments.
4. **Lightweight:**
   - Minimal resource overhead compared to other service meshes like Istio.
   - Uses Rust-based proxies for efficiency and performance.
5. **Ease of Use:**
   - Simple installation and configuration process with a CLI (linkerd install).
   - Pre-configured defaults for Kubernetes clusters, making it beginner-friendly.

Use Cases for Linkerd

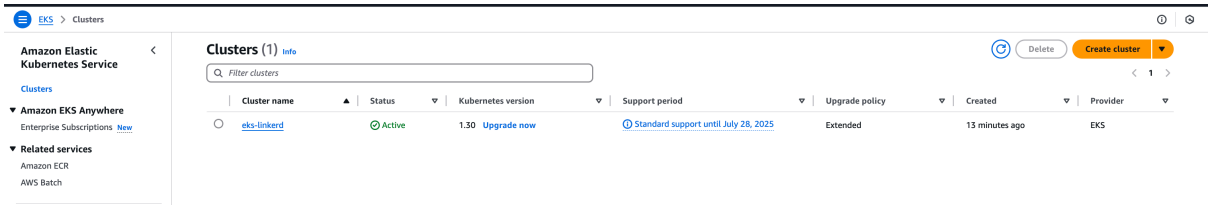
- **Enhancing Security**: Automatic encryption of inter-service traffic using mTLS.
- **Traffic Management**: Blue-green deployments, canary releases, and traffic shaping.
- **Monitoring and Debugging**: Golden metrics and real-time traffic inspection for better observability.
- **Resilience and Recovery**: Automatic retries, circuit breaking, and failure handling.

Step 1: Create the EKS Cluster Without Any Node Groups

Create an EKS cluster without a node group using the eksctl command **eksctl create cluster --name=eks-linkerd --region=eu-west-1 --without-nodegroup.** By default, eksctl creates a node group with m5.large instances, so we used the — without-nodegroup option to skip creating a default node group

```
mahendranselvakumar@Mahendrans-MBP ~ % eksctl create cluster --name=eks-linkerd --region=eu-west-1 --without-nodegroup
2024-12-30 10:55:25 [ℹ]  eksctl version 0.196.0
2024-12-30 10:55:25 [ℹ]  using region eu-west-1
2024-12-30 10:55:25 [ℹ]  setting availability zones to [eu-west-1c eu-west-1b eu-west-1a]
2024-12-30 10:55:25 [ℹ]  subnets for eu-west-1c - public:192.168.0.0/19 private:192.168.96.0/19
2024-12-30 10:55:25 [ℹ]  subnets for eu-west-1b - public:192.168.32.0/19 private:192.168.128.0/19
2024-12-30 10:55:25 [ℹ]  subnets for eu-west-1a - public:192.168.64.0/19 private:192.168.160.0/19
2024-12-30 10:55:25 [ℹ]  using Kubernetes version 1.30
2024-12-30 10:55:25 [ℹ]  creating EKS cluster "eks-linkerd" in "eu-west-1" region with
2024-12-30 10:55:25 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=eu-west-1 --cluster=eks-linkerd'
2024-12-30 10:55:25 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "eks-linkerd" in "eu-west-1"
2024-12-30 10:55:25 [ℹ]  CloudWatch logging will not be enabled for cluster "eks-linkerd" in "eu-west-1"
2024-12-30 10:55:25 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=eu-west-1 --cluster=eks-linkerd'
2024-12-30 10:55:25 [ℹ]  default addons vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons
2024-12-30 10:55:25 [ℹ]
2 sequential tasks: { create cluster control plane "eks-linkerd",
    2 sequential sub-tasks: {
        1 task: { create addons },
        wait for control plane to become ready,
    }
}
2024-12-30 10:55:25 [ℹ]  building cluster stack "eksctl-eks-linkerd-cluster"
2024-12-30 10:55:26 [ℹ]  deploying stack "eksctl-eks-linkerd-cluster"
2024-12-30 10:55:56 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 10:56:26 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 10:57:26 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 10:58:26 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 10:59:27 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 11:00:27 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 11:01:27 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 11:05:48 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-cluster"
2024-12-30 11:05:50 [!]  recommended policies were found for "vpc-cni" addon, but since OIDC is disabled on the cluster, eksctl cannot configure the requested permissions; the recommended
 way to provide IAM permissions for "vpc-cni" addon is via pod identity associations; after addon creation is completed, add all recommended policies to the config file, under `addon.PodI
dentityAssociations`, and run `eksctl update addon`
2024-12-30 11:05:50 [ℹ]  creating addon
2024-12-30 11:05:50 [ℹ]  successfully created addon
2024-12-30 11:05:50 [ℹ]  creating addon
2024-12-30 11:05:51 [ℹ]  successfully created addon
2024-12-30 11:05:51 [ℹ]  creating addon
2024-12-30 11:05:51 [ℹ]  successfully created addon
2024-12-30 11:07:52 [ℹ]  waiting for the control plane to become ready
2024-12-30 11:07:52 [✔]  saved kubeconfig as "/Users/mahendranselvakumar/.kube/config"
2024-12-30 11:07:52 [ℹ]  no tasks
2024-12-30 11:07:52 [✔]  all EKS cluster resources for "eks-linkerd" have been created
2024-12-30 11:07:52 [ℹ]  kubectl command should work with "/Users/mahendranselvakumar/.kube/config", try 'kubectl get nodes'
2024-12-30 11:07:52 [✔]  EKS cluster "eks-linkerd" in "eu-west-1" region is ready
mahendranselvakumar@Mahendrans-MBP ~ %
```

Go to the EKS console to verify that the cluster was successfully created using eksctl



Step 2: Create a Managed Node Group

Add a node group using the following separate **eksctl command eksctl create nodegroup --name eks-linkerd-ng --cluster eks-linkerd --region eu-west-1 --nodes 2 --nodes-min 1 --nodes-max 3 --node-type t3.medium**

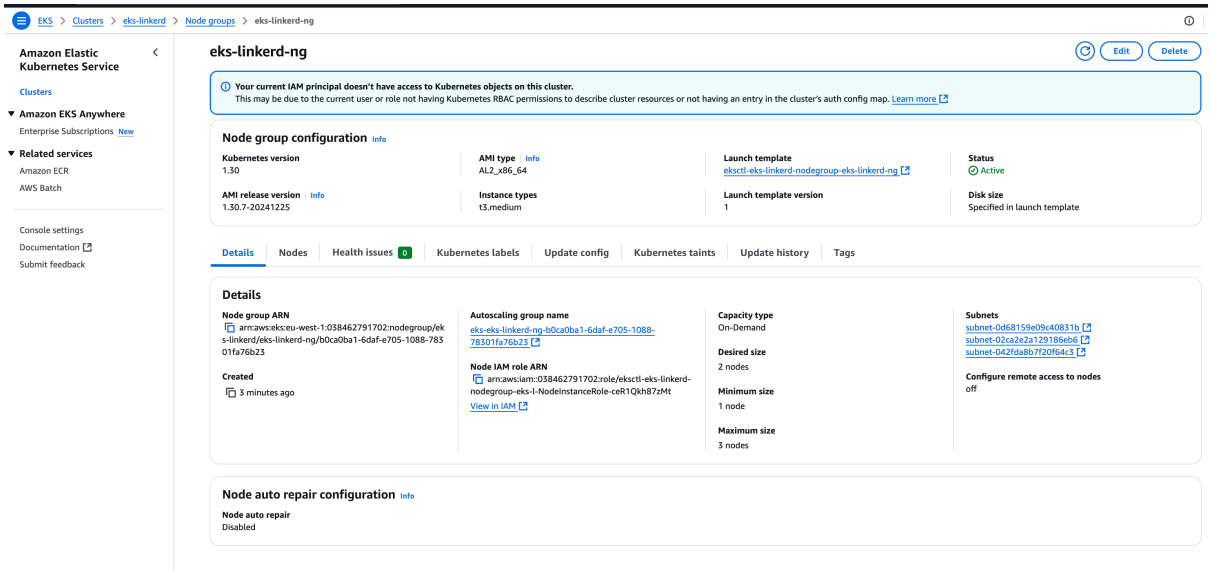
```
mahendranselvakumar@Mahendrans-MBP ~ % eksctl create nodegroup --name eks-linkerd-ng --cluster eks-linkerd --region eu-west-1 --nodes 2 --nodes-min 1 --nodes-max 3 --node-type t3.medium
2024-12-30 11:10:21 [ℹ]  will use version 1.30 for new nodegroup(s) based on control plane version
2024-12-30 11:10:23 [ℹ]  nodegroup "eks-linkerd-ng" will use "" [AmazonLinux2/1.30]
2024-12-30 11:10:23 [ℹ]  1 nodegroup (eks-linkerd-ng) was included (based on the include/exclude rules)
2024-12-30 11:10:23 [ℹ]  will create a CloudFormation stack for each of 1 managed nodegroups in cluster "eks-linkerd"
2024-12-30 11:10:23 [ℹ]
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create managed nodegroup "eks-linkerd-ng" } }
}
2024-12-30 11:10:23 [ℹ]  checking cluster stack for missing resources
2024-12-30 11:10:23 [ℹ]  cluster stack has all required resources
2024-12-30 11:10:24 [ℹ]  building managed nodegroup stack "eksctl-eks-linkerd-nodegroup-eks-linkerd-ng"
2024-12-30 11:10:24 [ℹ]  deploying stack "eksctl-eks-linkerd-nodegroup-eks-linkerd-ng"
2024-12-30 11:10:24 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-nodegroup-eks-linkerd-ng"
2024-12-30 11:10:54 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-nodegroup-eks-linkerd-ng"
2024-12-30 11:11:28 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-nodegroup-eks-linkerd-ng"
2024-12-30 11:13:26 [ℹ]  waiting for CloudFormation stack "eksctl-eks-linkerd-nodegroup-eks-linkerd-ng"
2024-12-30 11:13:26 [ℹ]  no tasks
2024-12-30 11:13:26 [✔]  created 0 nodegroup(s) in cluster "eks-linkerd"
2024-12-30 11:13:26 [ℹ]  nodegroup "eks-linkerd-ng" has 2 node(s)
2024-12-30 11:13:26 [ℹ]  node "ip-192-168-3-52.eu-west-1.compute.internal" is ready
2024-12-30 11:13:26 [ℹ]  node "ip-192-168-39-110.eu-west-1.compute.internal" is ready
2024-12-30 11:13:26 [ℹ]  waiting for at least 1 node(s) to become ready in "eks-linkerd-ng"
2024-12-30 11:13:26 [ℹ]  nodegroup "eks-linkerd-ng" has 2 node(s)
2024-12-30 11:13:26 [ℹ]  node "ip-192-168-3-52.eu-west-1.compute.internal" is ready
2024-12-30 11:13:26 [ℹ]  node "ip-192-168-39-110.eu-west-1.compute.internal" is ready
2024-12-30 11:13:26 [✔]  created 1 managed nodegroup(s) in cluster "eks-linkerd"
2024-12-30 11:13:26 [ℹ]  checking security group configuration for all nodegroups
2024-12-30 11:13:26 [ℹ]  all nodegroups have up-to-date cloudformation templates
mahendranselvakumar@Mahendrans-MBP ~ %
```

Explanation of the flags:

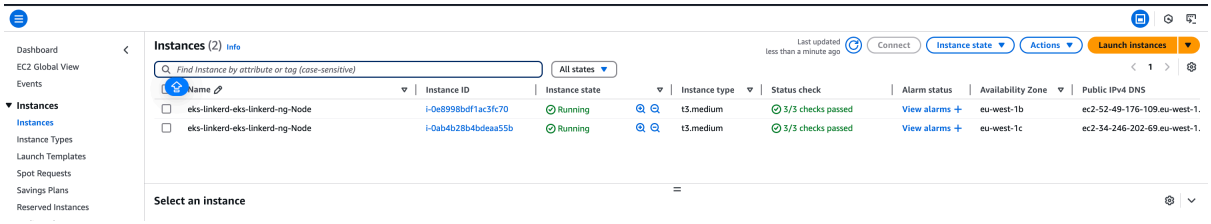
- — cluster: Specifies the name of the existing EKS cluster to which the node group will be added.

- — name: Names the node group for easy identification.
- — region: Specifies the AWS region.
- — nodes: Sets the initial desired number of nodes (in this case, 2).
- — nodes-min and — nodes-max: Define the minimum and maximum number of nodes for auto-scaling.
- — node-type: The EC2 instance type for the nodes (e.g., m5.large)

If you check in the console, you'll see that the node group has been created



In the instances section of the EC2 console, you can view the Instances created for the node group



Step 3: Configure Context for EKS Cluster

Set the Kubernetes context for the EKS cluster using the following command: aws eks --region eu-west-1 update-kubeconfig --name eks-linkerd

```
mahendranselvakumar@Mahendrans-MBP ~ % aws eks --region eu-west-1 update-kubeconfig --name eks-linkerd
Added new context arn:aws:eks:eu-west-1:038462791702:cluster/eks-linkerd to /Users/mahendranselvakumar/.kube/config
mahendranselvakumar@Mahendrans-MBP ~ %
```

Use **kubectl get ns** to view namespaces and **kubectl get nodes** to check the status of the nodes in the cluster, verifying that the setup is complete

```
mahendranselvakumar@Mahendrans-MBP ~ % kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
ip-192-168-3-52.eu-west-1.compute.internal    Ready    <none>   6m26s   v1.30.7-eks-59bf375
ip-192-168-39-110.eu-west-1.compute.internal  Ready    <none>   6m23s   v1.30.7-eks-59bf375
mahendranselvakumar@Mahendrans-MBP ~ % kubectl get ns
NAME              STATUS   AGE
default           Active   17m
kube-node-lease   Active   17m
kube-public       Active   17m
kube-system       Active   17m
mahendranselvakumar@Mahendrans-MBP ~ %
```

Step 4: Install the Linkerd CLI

Download the latest version of the Linkerd CLI using this command **curl --proto '=https' --tlsv1.2 -sSfL https://run.linkerd.io/install-edge | sh**

Note: The **install-edge** URL is for downloading the edge (pre-release) version of Linkerd. Use **install** for the stable release unless you specifically need the edge version

```
mahendranselvakumar@Mahendrans-MBP ~ % curl --proto '=https' --tlsv1.2 -sSfL https://run.linkerd.io/install-edge | sh

Downloading linkerd2-cli-edge-24.11.8-darwin-arm64...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 60.4M  100 60.4M    0     0  6924k      0  0:00:08  0:00:08 --:--:-- 7524k
Download complete!

Validating checksum...
Checksum valid.

Linkerd edge-24.11.8 was successfully installed 🎉


Add the linkerd CLI to your path with:

  export PATH=$PATH:/Users/mahendranselvakumar/.linkerd2/bin

Now run:

  linkerd check --pre                         # validate that Linkerd can be installed
  linkerd install --crds | kubectl apply -f - # install the Linkerd CRDs
  linkerd install | kubectl apply -f -        # install the control plane into the 'linkerd' namespace
  linkerd check                               # validate everything worked!

You can also obtain observability features by installing the viz extension:

  linkerd viz install | kubectl apply -f -  # install the viz extension into the 'linkerd-viz' namespace
  linkerd viz check                         # validate the extension works!
  linkerd viz dashboard                     # launch the dashboard

Looking for more? Visit https://linkerd.io/2/tasks

mahendranselvakumar@Mahendrans-MBP ~ %
```

Run this command **export PATH=$PATH:$HOME/.linkerd2/bin** to add the Linkerd CLI to your system's PATH

```
mahendranselvakumar@Mahendrans-MBP ~ % export PATH=$PATH:$HOME/.linkerd2/bin

mahendranselvakumar@Mahendrans-MBP ~ %
```

Verify the installation using this command **linkerd version**

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd version
Client version: edge-24.11.8
Server version: unavailable
mahendranselvakumar@Mahendrans-MBP ~ %
```

Note: If the output shows **Server version: unavailable**, it means the Linkerd control plane has not yet been installed on your cluster

Step 5: Validate Your Kubernetes Cluster

Run the pre-check command **linkerd check –pre** to ensure your Kubernetes cluster meets Linkerd's requirements

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd check --pre
kubernetes-api
--------------
√ can initialize the client
√ can query the Kubernetes API

kubernetes-version
------------------
√ is running the minimum Kubernetes API version

pre-kubernetes-setup
--------------------
√ control plane namespace does not already exist
√ can create non-namespaced resources
√ can create ServiceAccounts
√ can create Services
√ can create Deployments
√ can create CronJobs
√ can create ConfigMaps
√ can create Secrets
√ can read Secrets
√ can read extension-apiserver-authentication configmap
√ no clock skew detected

linkerd-version
---------------
√ can determine the latest version
√ cli is up-to-date

Status check results are √
mahendranselvakumar@Mahendrans-MBP ~ %
```

Step 6: Install Linkerd on your EKS cluster

Run this command **linkerd install --crds | kubectl apply -f –** to install Linkerd's Custom Resource Definitions (CRDs),which must be installed before the control plane

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd install --crds | kubectl apply -f -
Rendering Linkerd CRDs...
Next, run `linkerd install | kubectl apply -f -` to install the control plane.

customresourcedefinition.apiextensions.k8s.io/authorizationpolicies.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/egressnetworks.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/httplocalratelimitpolicies.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/httproutes.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/meshtlsauthentications.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/networkauthentications.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/serverauthorizations.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/servers.policy.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/serviceprofiles.linkerd.io created
customresourcedefinition.apiextensions.k8s.io/httproutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/grpcroutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/tlsroutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/tcproutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/externalworkloads.workload.linkerd.io created
mahendranselvakumar@Mahendrans-MBP ~ %
```

Install the Linkerd control plane using this command **linkerd install | kubectl apply -f -**

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd install | kubectl apply -f -
namespace/linkerd created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-identity created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-identity created
serviceaccount/linkerd-identity created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-destination created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-destination created
serviceaccount/linkerd-destination created
secret/linkerd-sp-validator-k8s-tls created
validatingwebhookconfiguration.admissionregistration.k8s.io/linkerd-sp-validator-webhook-config created
secret/linkerd-policy-validator-k8s-tls created
validatingwebhookconfiguration.admissionregistration.k8s.io/linkerd-policy-validator-webhook-config created
clusterrole.rbac.authorization.k8s.io/linkerd-policy created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-destination-policy created
role.rbac.authorization.k8s.io/remote-discovery created
rolebinding.rbac.authorization.k8s.io/linkerd-destination-remote-discovery created
role.rbac.authorization.k8s.io/linkerd-heartbeat created
rolebinding.rbac.authorization.k8s.io/linkerd-heartbeat created
clusterrole.rbac.authorization.k8s.io/linkerd-heartbeat created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-heartbeat created
serviceaccount/linkerd-heartbeat created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-proxy-injector created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-proxy-injector created
serviceaccount/linkerd-proxy-injector created
secret/linkerd-proxy-injector-k8s-tls created
mutatingwebhookconfiguration.admissionregistration.k8s.io/linkerd-proxy-injector-webhook-config created
configmap/linkerd-config created
role.rbac.authorization.k8s.io/ext-namespace-metadata-linkerd-config created
secret/linkerd-identity-issuer created
configmap/linkerd-identity-trust-roots created
service/linkerd-identity created
service/linkerd-identity-headless created
deployment.apps/linkerd-identity created
service/linkerd-dst created
service/linkerd-dst-headless created
service/linkerd-sp-validator created
service/linkerd-policy created
service/linkerd-policy-validator created
deployment.apps/linkerd-destination created
cronjob.batch/linkerd-heartbeat created
deployment.apps/linkerd-proxy-injector created
service/linkerd-proxy-injector created
secret/linkerd-config-overrides created
mahendranselvakumar@Mahendrans-MBP ~ %
```

Now check the version again using this command **linkerd version** and you should now see both the **Client version** and the **Server version** displayed

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd version
Client version: edge-24.11.8
Server version: edge-24.11.8
mahendranselvakumar@Mahendrans-MBP ~ %
```

Validate the installation using **linkerd check** command and Ensure that all checks pass successfully to confirm that the Linkerd control plane is properly installed and functioning

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd check
kubernetes-api
--------------
√ can initialize the client
√ can query the Kubernetes API

kubernetes-version
------------------
√ is running the minimum Kubernetes API version

linkerd-existence
-----------------
√ 'linkerd-config' config map exists
√ heartbeat ServiceAccount exist
√ control plane replica sets are ready
√ no unschedulable pods
√ control plane pods are ready
√ cluster networks contains all pods
√ cluster networks contains all services

linkerd-config
--------------
√ control plane Namespace exists
√ control plane ClusterRoles exist
√ control plane ClusterRoleBindings exist
√ control plane ServiceAccounts exist
√ control plane CustomResourceDefinitions exist
√ control plane MutatingWebhookConfigurations exist
√ control plane ValidatingWebhookConfigurations exist
√ proxy-init container runs as root user if docker container runtime is used

linkerd-identity
----------------
√ certificate config is valid
√ trust anchors are using supported crypto algorithm
√ trust anchors are within their validity period
√ trust anchors are valid for at least 60 days
√ issuer cert is using supported crypto algorithm
√ issuer cert is within its validity period
√ issuer cert is valid for at least 60 days
√ issuer cert is issued by the trust anchor

linkerd-webhooks-and-apisvc-tls
-------------------------------
√ proxy-injector webhook has valid cert
√ proxy-injector cert is valid for at least 60 days
√ sp-validator webhook has valid cert
√ sp-validator cert is valid for at least 60 days
√ policy-validator webhook has valid cert
√ policy-validator cert is valid for at least 60 days

linkerd-version
---------------
√ can determine the latest version
√ cli is up-to-date

control-plane-version
---------------------
√ can retrieve the control plane version
√ control plane is up-to-date
√ control plane and cli versions match

linkerd-control-plane-proxy
---------------------------
√ control plane proxies are healthy
√ control plane proxies are up-to-date
√ control plane proxies and cli versions match

linkerd-extension-checks
------------------------
√ namespace configuration for extensions

Status check results are √
mahendranselvakumar@Mahendrans-MBP ~ %
```

Step 7: Install the viz extension on EKS Cluster

The Linkerd dashboard provides a high level view of what is happening with your services in real time. It can be used to view “golden metrics” (success rate, requests/second and latency), visualize service dependencies and understand the health of specific service routes.

Run this command **linkerd viz install | kubectl apply -f –** to install viz extensions which includes a Prometheus instance, metrics-api, tap, tap-injector, and web components. These components work together to provide an on-cluster metrics stack

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd viz install | kubectl apply -f -
namespace/linkerd-viz created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-metrics-api created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-metrics-api created
serviceaccount/metrics-api created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-prometheus created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-prometheus created
serviceaccount/prometheus created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap-admin created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap-auth-delegator created
serviceaccount/tap created
rolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-tap-auth-reader created
secret/tap-k8s-tls created
apiservice.apiregistration.k8s.io/v1alpha1.tap.linkerd.io created
role.rbac.authorization.k8s.io/web created
rolebinding.rbac.authorization.k8s.io/web created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-check created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-check created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-admin created
clusterrole.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-api created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-linkerd-viz-web-api created
serviceaccount/web created
service/metrics-api created
deployment.apps/metrics-api created
server.policy.linkerd.io/metrics-api created
authorizationpolicy.policy.linkerd.io/metrics-api created
meshtlsauthentication.policy.linkerd.io/metrics-api-web created
networkauthentication.policy.linkerd.io/kubelet created
configmap/prometheus-config created
service/prometheus created
deployment.apps/prometheus created
server.policy.linkerd.io/prometheus-admin created
authorizationpolicy.policy.linkerd.io/prometheus-admin created
service/tap created
deployment.apps/tap created
server.policy.linkerd.io/tap-api created
authorizationpolicy.policy.linkerd.io/tap created
clusterrole.rbac.authorization.k8s.io/linkerd-tap-injector created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-tap-injector created
serviceaccount/tap-injector created
secret/tap-injector-k8s-tls created
mutatingwebhookconfiguration.admissionregistration.k8s.io/linkerd-tap-injector-webhook-config created
service/tap-injector created
deployment.apps/tap-injector created
server.policy.linkerd.io/tap-injector-webhook created
authorizationpolicy.policy.linkerd.io/tap-injector created
networkauthentication.policy.linkerd.io/kube-api-server created
service/web created
deployment.apps/web created
serviceprofile.linkerd.io/metrics-api.linkerd-viz.svc.cluster.local created
serviceprofile.linkerd.io/prometheus.linkerd-viz.svc.cluster.local created
mahendranselvakumar@Mahendrans-MBP ~ %
```

Run **linkerd viz check** command to validate the viz installation and Ensure all checks pass successfully to confirm that the Viz extension is properly installed and functioning

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd viz check
linkerd-viz
-----------
√ linkerd-viz Namespace exists
√ can initialize the client
√ linkerd-viz ClusterRoles exist
√ linkerd-viz ClusterRoleBindings exist
√ tap API server has valid cert
√ tap API server cert is valid for at least 60 days
√ tap API service is running
√ linkerd-viz pods are injected
√ viz extension pods are running
√ viz extension proxies are healthy
√ viz extension proxies are up-to-date
√ viz extension proxies and cli versions match
√ prometheus is installed and configured correctly
√ viz extension self-check

Status check results are √
mahendranselvakumar@Mahendrans-MBP ~ %
```

Step 8: Access the Linkerd Dashboard

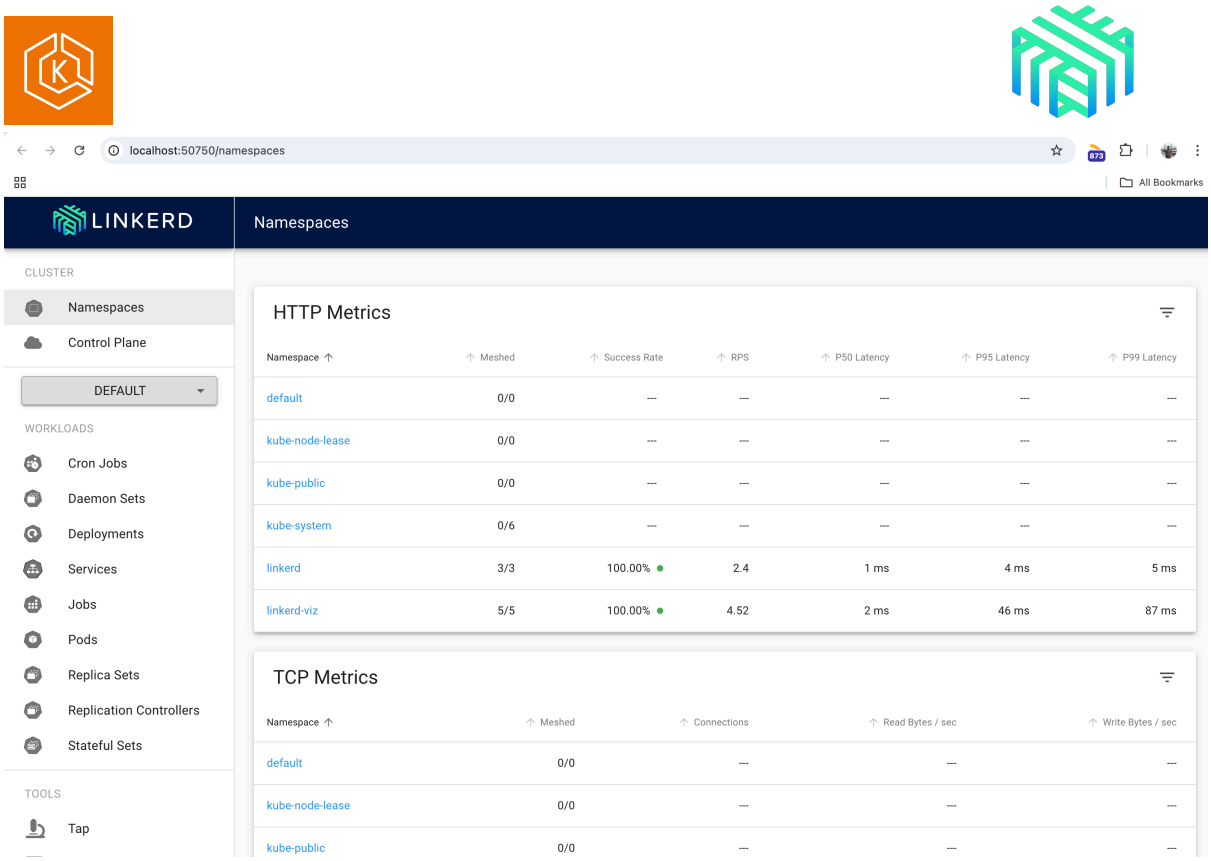
Launch the Linkerd dashboard using this command **linkerd viz dashboard.** This will open the Linkerd dashboard in your default web browser, providing a visual interface to monitor metrics and observe your meshed workloads

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd viz dashboard
Linkerd dashboard available at:
http://localhost:50750
Grafana dashboard available at:
http://localhost:50750/grafana
Opening Linkerd dashboard in the default browser
```

Once the Linkerd dashboard is launched, you can view the **golden metrics** for your meshed workloads. Once the Linkerd dashboard is launched, you can view the **golden metrics** for your meshed workloads. These metrics include:

- **Success Rate**: The percentage of successful requests between services.
- **Requests per Second (RPS)**: The volume of traffic handled by each service.
- **Latency**: The time it takes for requests to be processed, broken down into P50, P95, and P99 percentiles.

These metrics provide valuable insights into the performance and health of your microservices.



Step 9: Install the Linkerd-Jaeger extension on EKS Cluster

Run this command **linkerd jaeger install | kubectl apply -f –** to install linkerd-jaeger extension for distributed tracing

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd jaeger install | kubectl apply -f -

namespace/linkerd-jaeger created
deployment.apps/jaeger-injector created
service/jaeger-injector created
server.policy.linkerd.io/jaeger-injector-webhook created
authorizationpolicy.policy.linkerd.io/jaeger-injector created
networkauthentication.policy.linkerd.io/kube-api-server created
serviceaccount/collector created
clusterrole.rbac.authorization.k8s.io/collector created
clusterrolebinding.rbac.authorization.k8s.io/collector created
clusterrole.rbac.authorization.k8s.io/linkerd-jaeger-injector created
clusterrolebinding.rbac.authorization.k8s.io/linkerd-jaeger-injector created
serviceaccount/jaeger-injector created
secret/jaeger-injector-k8s-tls created
mutatingwebhookconfiguration.admissionregistration.k8s.io/linkerd-jaeger-injector-webhook-config created
serviceaccount/jaeger created
configmap/collector-config created
service/collector created
deployment.apps/collector created
service/jaeger created
deployment.apps/jaeger created
server.policy.linkerd.io/collector-otlp created
server.policy.linkerd.io/collector-otlp-http created
server.policy.linkerd.io/collector-opencensus created
server.policy.linkerd.io/collector-zipkin created
server.policy.linkerd.io/collector-jaeger-thrift created
server.policy.linkerd.io/collector-jaeger-grpc created
server.policy.linkerd.io/collector-admin created
authorizationpolicy.policy.linkerd.io/collector-otlp created
authorizationpolicy.policy.linkerd.io/collector-otlp-http created
authorizationpolicy.policy.linkerd.io/collector-opencensus created
authorizationpolicy.policy.linkerd.io/collector-zipkin created
authorizationpolicy.policy.linkerd.io/collector-jaeger-thrift created
authorizationpolicy.policy.linkerd.io/collector-jaeger-grpc created
server.policy.linkerd.io/jaeger-grpc created
authorizationpolicy.policy.linkerd.io/jaeger-grpc created
server.policy.linkerd.io/jaeger-admin created
authorizationpolicy.policy.linkerd.io/jaeger-admin created
server.policy.linkerd.io/jaeger-ui created
authorizationpolicy.policy.linkerd.io/jaeger-ui created
mahendranselvakumar@Mahendrans-MBP ~ %
```

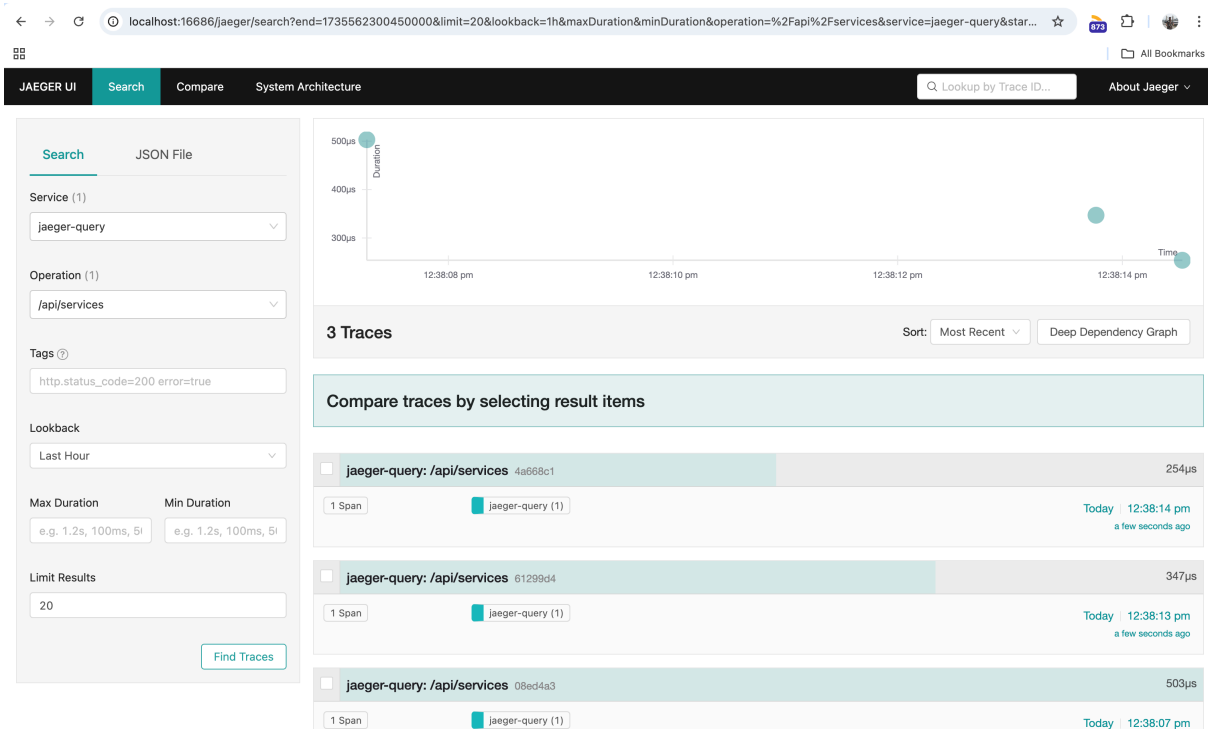
You can verify that the Linkerd-Jaeger extension was installed correctly by running this command **linkerd jaeger check.**Ensure all checks pass successfully to confirm that the Jaeger extension is properly installed and functional

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd jaeger check

linkerd-jaeger
--------------
√ linkerd-jaeger extension Namespace exists
√ jaeger extension pods are injected
√ jaeger injector pods are running
√ jaeger extension proxies are healthy
√ jaeger extension proxies are up-to-date
√ jaeger extension proxies and cli versions match

Status check results are √
mahendranselvakumar@Mahendrans-MBP ~ %
```

Run this command **linkerd jaeger dashboard** to explore the Jaeger dashboard for distributed tracing. This will open the Jaeger dashboard in your default web browser, allowing you to visualize and analyze trace data for your services

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd jaeger dashboard

Jaeger extension dashboard available at:
http://localhost:16686
```

This command will launch the Jaeger UI dashboard and you can access the jaeger dashboard with this URL **http://localhost:16686.**



Here, you can visualize and analyze the traces for your meshed services

Step 10: Uninstall Linkerd Extensions

Uninstall the Viz Extension using this command **linkerd viz uninstall | kubectl delete -f –**

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd viz uninstall | kubectl delete -f -
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-viz-metrics-api" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-viz-prometheus" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-viz-tap" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-viz-tap-admin" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-viz-web-api" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-viz-web-check" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-tap-injector" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-metrics-api" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-prometheus" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-tap" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-tap-auth-delegator" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-web-admin" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-web-api" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-web-check" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-tap-injector" deleted
role.rbac.authorization.k8s.io "web" deleted
rolebinding.rbac.authorization.k8s.io "linkerd-linkerd-viz-tap-auth-reader" deleted
rolebinding.rbac.authorization.k8s.io "web" deleted
apiservice.apiregistration.k8s.io "v1alpha1.tap.linkerd.io" deleted
mutatingwebhookconfiguration.admissionregistration.k8s.io "linkerd-tap-injector-webhook-config" deleted
namespace "linkerd-viz" deleted
authorizationpolicy.policy.linkerd.io "metrics-api" deleted
authorizationpolicy.policy.linkerd.io "prometheus-admin" deleted
authorizationpolicy.policy.linkerd.io "tap" deleted
authorizationpolicy.policy.linkerd.io "tap-injector" deleted
server.policy.linkerd.io "metrics-api" deleted
server.policy.linkerd.io "prometheus-admin" deleted
server.policy.linkerd.io "tap-api" deleted
server.policy.linkerd.io "tap-injector-webhook" deleted
mahendranselvakumar@Mahendrans-MBP ~ %
```

Uninstall the Jaeger Extension using this command **linkerd jaeger uninstall | kubectl delete -f –**

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd jaeger uninstall | kubectl delete -f -

clusterrole.rbac.authorization.k8s.io "collector" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-jaeger-injector" deleted
clusterrolebinding.rbac.authorization.k8s.io "collector" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-jaeger-injector" deleted
mutatingwebhookconfiguration.admissionregistration.k8s.io "linkerd-jaeger-injector-webhook-config" deleted
namespace "linkerd-jaeger" deleted
mahendranselvakumar@Mahendrans-MBP ~ %
```

Check that their associated namespaces have been removed using **kubectl get namespaces** command

```
mahendranselvakumar@Mahendrans-MBP ~ % kubectl get namespaces
NAME              STATUS   AGE
default           Active   5h
emojivoto         Active   3h33m
kube-node-lease   Active   5h
kube-public       Active   5h
kube-system       Active   5h
linkerd           Active   4h31m
mahendranselvakumar@Mahendrans-MBP ~ %
```

Step 11: Uninstall the Linkerd Control Plane

Run this **linkerd uninstall | kubectl delete -f -** command to uninstall the Linkerd control plane

```
mahendranselvakumar@Mahendrans-MBP ~ % linkerd uninstall | kubectl delete -f -
clusterrole.rbac.authorization.k8s.io "linkerd-heartbeat" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-destination" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-identity" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-linkerd-proxy-injector" deleted
clusterrole.rbac.authorization.k8s.io "linkerd-policy" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-destination-policy" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-heartbeat" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-destination" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-identity" deleted
clusterrolebinding.rbac.authorization.k8s.io "linkerd-linkerd-proxy-injector" deleted
role.rbac.authorization.k8s.io "ext-namespace-metadata-linkerd-config" deleted
role.rbac.authorization.k8s.io "linkerd-heartbeat" deleted
role.rbac.authorization.k8s.io "remote-discovery" deleted
rolebinding.rbac.authorization.k8s.io "linkerd-destination-remote-discovery" deleted
rolebinding.rbac.authorization.k8s.io "linkerd-heartbeat" deleted
customresourcedefinition.apiextensions.k8s.io "authorizationpolicies.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "egressnetworks.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "externalworkloads.workload.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "grpcroutes.gateway.networking.k8s.io" deleted
customresourcedefinition.apiextensions.k8s.io "httplocalratelimitpolicies.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "httproutes.gateway.networking.k8s.io" deleted
customresourcedefinition.apiextensions.k8s.io "httproutes.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "meshtlsauthentications.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "networkauthentications.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "serverauthorizations.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "servers.policy.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "serviceprofiles.linkerd.io" deleted
customresourcedefinition.apiextensions.k8s.io "tcproutes.gateway.networking.k8s.io" deleted
customresourcedefinition.apiextensions.k8s.io "tlsroutes.gateway.networking.k8s.io" deleted
mutatingwebhookconfiguration.admissionregistration.k8s.io "linkerd-proxy-injector-webhook-config" deleted
validatingwebhookconfiguration.admissionregistration.k8s.io "linkerd-policy-validator-webhook-config" deleted
validatingwebhookconfiguration.admissionregistration.k8s.io "linkerd-sp-validator-webhook-config" deleted
namespace "linkerd" deleted
mahendranselvakumar@Mahendrans-MBP ~ %
```

Verify that the linkerd namespace is removed

```
mahendranselvakumar@Mahendrans-MBP ~ % kubectl get namespaces
NAME              STATUS   AGE
default           Active   5h4m
emojivoto         Active   3h37m
kube-node-lease   Active   5h4m
kube-public       Active   5h4m
kube-system       Active   5h4m
mahendranselvakumar@Mahendrans-MBP ~ %
```

Step 12: Remove Linkerd CRDs

To ensure all Linkerd Custom Resource Definitions (CRDs) are removed using this command **kubectl delete crds -l linkerd.io/control-plane-ns=linkerd**

```
mahendranselvakumar@Mahendrans-MBP ~ % kubectl delete crds -l linkerd.io/control-plane-ns=linkerd

No resources found
mahendranselvakumar@Mahendrans-MBP ~ %
```

Conclusion

Linkerd is a reliable and lightweight service mesh that adds security, observability, and reliability to your cloud-native applications. This guide covered the steps to set up Linkerd on AWS EKS, including installing the control plane, enabling observability with the Viz extension, and setting up distributed tracing with the Jaeger extension.

With Linkerd installed, your microservices are now equipped with features like automatic mTLS encryption, real-time metrics, and distributed tracing, helping you better manage and monitor your Kubernetes workloads. These tools make it easier to ensure secure, efficient, and reliable communication between services.

This setup provides a solid foundation for managing microservices in a Kubernetes environment, with the flexibility to scale and extend as needed.

Keep Learning, Keep Mesh-ing!!

Feel free to reach out to me, if you have any other queries or suggestions Stay connected on LinkedIn: Mahendran Selvakumar

Stay connected on Medium: https://devopstronaut.com/