**1. Helm에 eks-chart 추가**

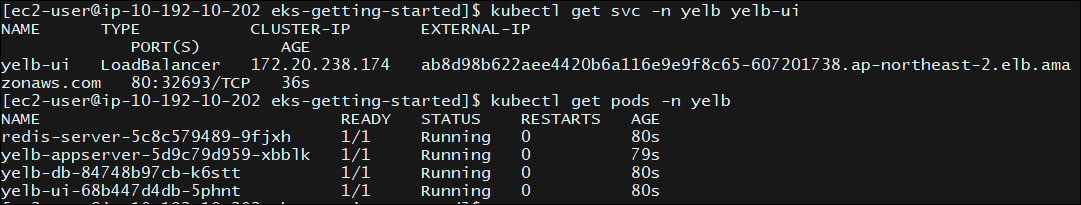
|  |
| --- |
| $ helm repo add eks <https://aws.github.io/eks-charts> |

**2. App Mesh Controller를 설치하기**

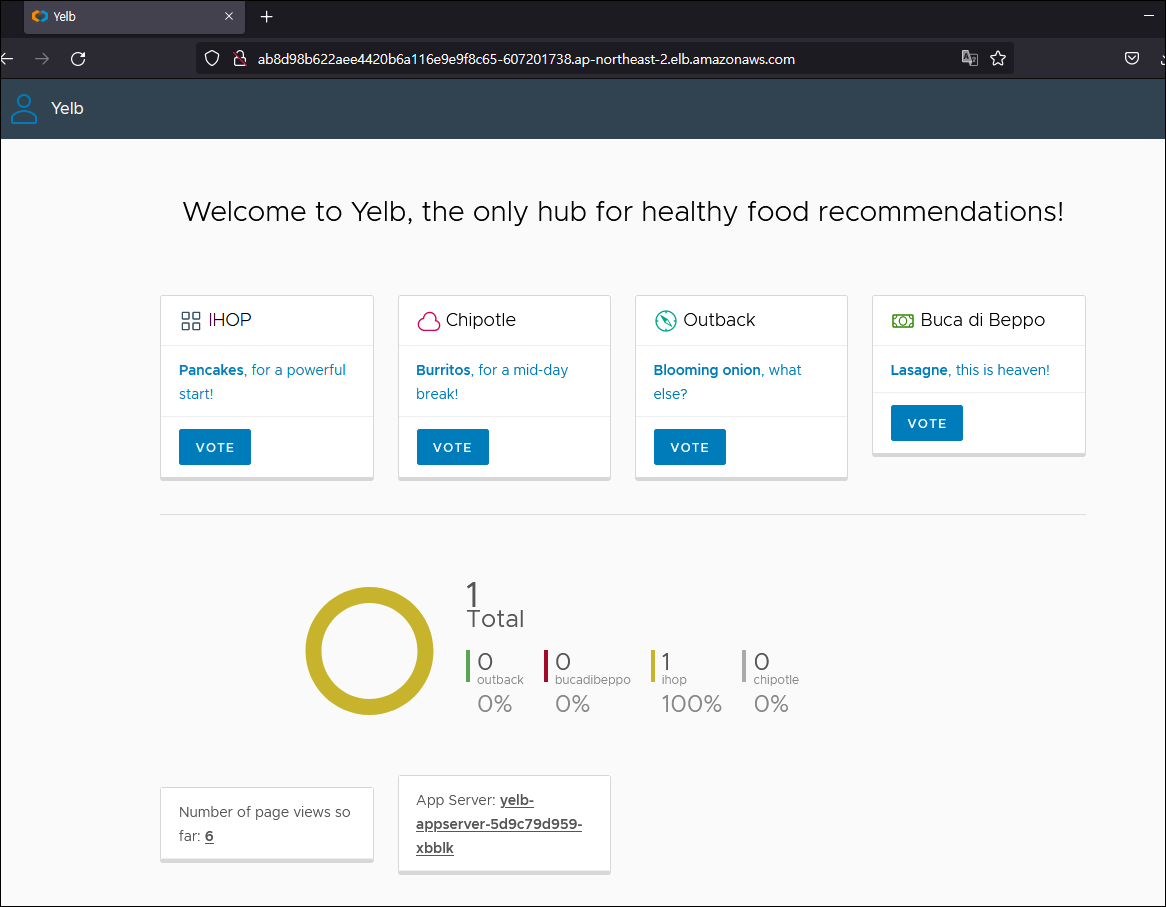
|  |
| --- |
| $ kubectl create ns appmesh-system  $ helm upgrade -i appmesh-controller eks/appmesh-controller -n appmesh-system  $ kubectl get pods -n appmesh-system |

**3. Deploy Demo Application**

|  |
| --- |
| $ git clone https://github.com/aws/aws-app-mesh-examples.git  $ kubectl create ns yelb  $ cd aws-app-mesh-examples/walkthroughs/eks-getting-started/  $ kubectl apply -f infrastructure/yelb\_initial\_deployment.yaml  $ kubectl get pods -n yelb  $ kubectl get svc -n yelb yelb-ui |



그리고 LoadBalancer의 External-IP로 접근해보자.



위와 같은 사이트가 출력되는 것을 볼 수 있다.

**4. Namespace에 sidecar injection label 적용하기**

app에 envoy sidecar를 적용하려면 해당 app이 설치된 namespace에 label을 적용시켜줘야 한다. 아래와 같이 애플리케이션에 적용될 mesh 이름과 사이트카 Injector webhook을 활성화하는 Label을 적용해주자.

|  |
| --- |
| $ kubectl label namespace yelb mesh=yelb  $ kubectl label namespace yelb appmesh.k8s.aws/sidecarInjectorWebhook=enabled  $ kubectl get namespaces --show-labels | grep yelb |

**5. App Mesh 컴포넌트 등록**

sample app에 mesh를 적용하려면 먼저 Application을 각각의 서비스로 추상화하는 app mesh 컴포넌트들을 생성해야한다.

우선 아래와 같이 mesh를 생성해주자.

|  |
| --- |
| apiVersion: appmesh.k8s.aws/v1beta2  kind: Mesh  metadata:  name: yelb  spec:  namespaceSelector:  matchLabels:  mesh: yelb |

그러면 아래와 같이 app mesh가 생성된 것을 볼 수 있다.



그리고 이제 모든 구성요소 들을 추상화해주자.

|  |
| --- |
| # redis  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualNode  metadata:  name: redis-server  namespace: yelb  spec:  awsName: redis-server-virtual-node  podSelector:  matchLabels:  app: redis-server  listeners:  - portMapping:  port: 6379  protocol: tcp  serviceDiscovery:  dns:  hostname: redis-server.yelb.svc.cluster.local  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualService  metadata:  name: redis-server  namespace: yelb  spec:  awsName: redis-server  provider:  virtualNode:  virtualNodeRef:  name: redis-server  ---  # DB  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualNode  metadata:  name: db-server  namespace: yelb  spec:  awsName: db-virtual-node  podSelector:  matchLabels:  app: yelb-db  listeners:  - portMapping:  port: 5432  protocol: tcp  serviceDiscovery:  dns:  hostname: yelb-db.yelb.svc.cluster.local  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualService  metadata:  name: db-server  namespace: yelb  spec:  awsName: db-server  provider:  virtualNode:  virtualNodeRef:  name: db-server  # appserver  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualNode  metadata:  name: app-server  namespace: yelb  spec:  awsName: app-virtual-node  podSelector:  matchLabels:  app: yelb-appserver  listeners:  - portMapping:  port: 4567  protocol: tcp  serviceDiscovery:  dns:  hostname: yelb-appserver.yelb.svc.cluster.local  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualRouter  metadata:  namespace: yelb  name: app-server  spec:  awsName: app-server-virtual-router  listeners:  - portMapping:  port: 4567  protocol: http  routes:  - name: route-to-yelb-appserver  httpRoute:  match:  prefix: /  action:  weightedTargets:  - virtualNodeRef:  name: app-server  weight: 1  retryPolicy:  maxRetries: 2  perRetryTimeout:  unit: ms  value: 2000  httpRetryEvents:  - server-error  - client-error  - gateway-error  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualService  metadata:  name: app-server  namespace: yelb  spec:  awsName: app-server  provider:  virtualNode:  virtualNodeRef:  name: app-server  # UI  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualNode  metadata:  name: ui-server  namespace: yelb  spec:  awsName: ui-virtual-node  podSelector:  matchLabels:  app: yelb-ui  listeners:  - portMapping:  port: 80  protocol: http  serviceDiscovery:  dns:  hostname: yelb-ui.yelb.svc.cluster.local  backends:  - virtualService:  virtualServiceRef:  name: ui-server  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualService  metadata:  name: ui-server  namespace: yelb  spec:  awsName: ui-server  provider:  virtualNode:  virtualNodeRef:  name: ui-server |

그리고 적용 시켜주면 아래와 같이 Virtual Service와 Virtual Node 등이 생성된 것을 볼 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

**5. Envoy SideCar 주입**

Mesh의 모든 구성 요소가 준비되었으면 sidecar를 주입해주자. 위에서 app mesh controller를 namespace에 설정하였으므로, Pod를 새로 띄우면 자동으로 주입된다. READY가 1/1에서 2/2로 변경되는 것을 볼 수 있다.

|  |
| --- |
| $ kubectl rollout restart deployment -n yelb |

Envory Sidecar 주입 전

텍스트이(가) 표시된 사진

자동 생성된 설명

Envoy Sidecar 주입 후

텍스트이(가) 표시된 사진

자동 생성된 설명

**6. Create VIrtual Gateway & Virtual Router**

가상 게이트웨이를 적용하려면 namespace에 label을 명시해줘야한다.

|  |
| --- |
| $ kubectl label namespace yelb gateway=yelb-gateway |

|  |
| --- |
| apiVersion: appmesh.k8s.aws/v1beta2  kind: VirtualGateway  metadata:  name: yelb-gateway  namespace: yelb  spec:  namespaceSelector:  matchLabels:  gateway: yelb-gateway  podSelector:  matchLabels:  app: yelb-gateway  listeners:  - portMapping:  port: 8088  protocol: http  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: GatewayRoute  metadata:  name: yelbui-gatewayroute  namespace: yelb  spec:  httpRoute:  match:  prefix: "/"  action:  target:  virtualService:  virtualServiceRef:  name: ui-server  ---  apiVersion: appmesh.k8s.aws/v1beta2  kind: GatewayRoute  metadata:  name: yelbapp-gatewayroute  namespace: yelb  spec:  httpRoute:  match:  prefix: "/api"  action:  target:  virtualService:  virtualServiceRef:  name: app-server |

그리고 Envoy로 gateway 생성해서 외부에 서비스 노출하자

**7. X-Ray Daemon 적용하기**

우선 Worker Node IAM Role에 아래와 같은 권한을 부여해주고 진행한다.



|  |
| --- |
| $ helm upgrade -i appmesh-controller eks/appmesh-controller -n appmesh-system --set tracing.enabled=true --set tracing.provider=x-ray  $ kubectl rollout restart deployment -n yelb  $ kubectl -n yelb get po |

텍스트이(가) 표시된 사진

자동 생성된 설명