**Istio**

**1. install istioctl**

|  |
| --- |
| $ curl -L https://git.io/getLatestIstio | sh -  $ cd istio-1.17.2/ |

**2. install istio**

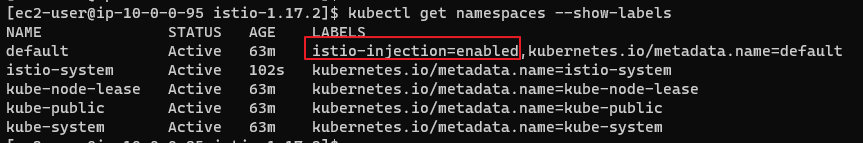
|  |
| --- |
| $ istioctl install |

**3. Envoy 사이드카 Injection label 설정**

|  |
| --- |
| $ kubectl label namespace default istio-injection=enabled |

확인해본다.

|  |
| --- |
| $ kubectl get namespaces --show-labels |



**4. 서비스 배포**

아래 yaml 파일을 가지고 서비스들을 배포해주도록 해준다.

|  |
| --- |
| #nginx-service  apiVersion: v1  kind: Service  metadata:  name: nginx  labels:  app: nginx  service: nginx  spec:  ports:  - port: 80  name: http  selector:  app: nginx  ---  apiVersion: v1  kind: ServiceAccount  metadata:  name: service-nginx  labels:  account: nginx  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx-v1  labels:  app: nginx  version: v1  spec:  replicas: 1  selector:  matchLabels:  app: nginx  version: v1  template:  metadata:  labels:  app: nginx  version: v1  spec:  serviceAccountName: service-nginx  containers:  - name: nginx  image: nginx:latest  imagePullPolicy: IfNotPresent  ports:  - containerPort: 80  ---  # httpdpage services  apiVersion: v1  kind: Service  metadata:  name: httpdpage  labels:  app: httpdpage  service: httpdpage  spec:  ports:  - port: 80  name: http  selector:  app: httpdpage  ---  apiVersion: v1  kind: ServiceAccount  metadata:  name: service-httpdpage  labels:  account: httpdpage  ---  apiVersion: apps/v1  kind: Deployment  metadata:  name: httpdpage-v1  labels:  app: httpdpage  version: v1  spec:  replicas: 1  selector:  matchLabels:  app: httpdpage  version: v1  template:  metadata:  labels:  app: httpdpage  version: v1  spec:  serviceAccountName: service-httpdpage  containers:  - name: httpd  image: httpd:latest  imagePullPolicy: IfNotPresent  ports:  - containerPort: 80  volumeMounts:  - name: tmp  mountPath: /tmp  volumes:  - name: tmp  emptyDir: {}  --- |

**5. 게이트웨이 생성**

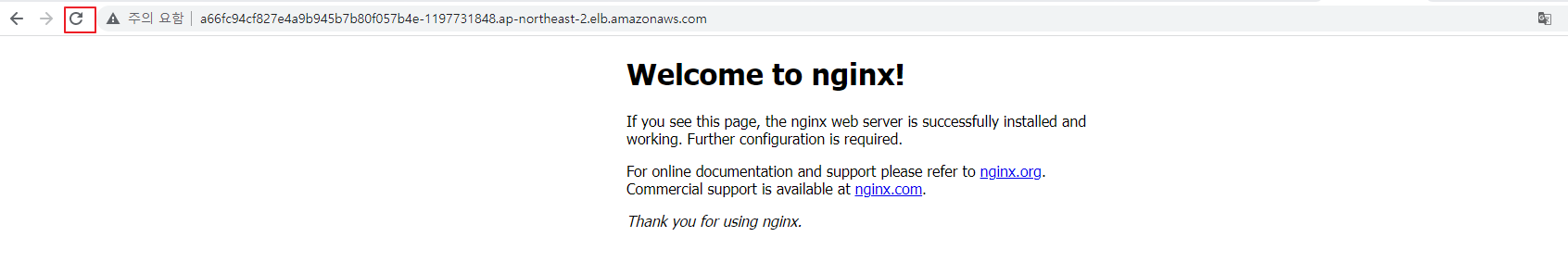
|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: Gateway  metadata:  name: service-gateway  spec:  selector:  istio: ingressgateway  servers:  - port:  number: 80  name: http  protocol: HTTP  hosts:  - "\*"  ---  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: httpdpage-nginx  spec:  hosts:  - "\*"  gateways:  - service-gateway  http:  - route:  - destination:  host: httpdpage  port:  number: 80  weight: 50  - destination:  host: nginx  port:  number: 80  weight: 50 |

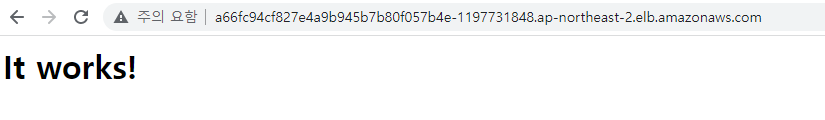
**6. nlb 적용**

|  |
| --- |
| apiVersion: install.istio.io/v1alpha1  kind: IstioOperator  spec:  values:  gateways:  istio-ingressgateway:  serviceAnnotations:  service.beta.kubernetes.io/aws-load-balancer-type: "nlb" |

|  |
| --- |
| $ istioctl upgrade -f nlb.yaml |

nlb 주소로 접근하면 nginx 페이지 또는 httpd 페이지가 표시된 것을 확인할 수 있다.





**7. 새로운 버전의 nginx page 생성**

아래는 hello world가 표시되는 두번째 버전의 nginx이다.

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: nginx-v2  labels:  app: nginx  version: v2  spec:  replicas: 1  selector:  matchLabels:  app: nginx  version: v2  template:  metadata:  labels:  app: nginx  version: v2  spec:  serviceAccountName: service-nginx  containers:  - name: nginx  image: nginx:latest  imagePullPolicy: IfNotPresent  ports:  - containerPort: 80  volumeMounts:  - name: html-volume  mountPath: /usr/share/nginx/html  volumes:  - name: html-volume  configMap:  name: nginx-config  ---  apiVersion: v1  kind: ConfigMap  metadata:  name: nginx-config  data:  index.html: |  <html>  <body>  <h1>Hello, world!</h1>  </body>  </html> |

**8. DestinationRule 작성**

버전 라우팅을 제어하기 위해 DestinationRule을 작성하여 사용가능한 버전을 정의해준다.

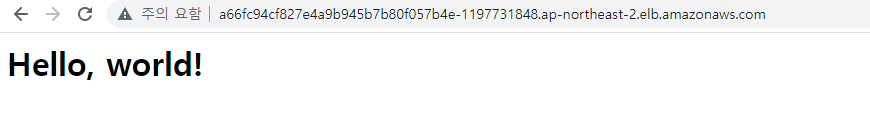
|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: httpdpage  spec:  host: httpdpage  subsets:  - name: v1  labels:  version: v1  ---  apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: nginx  spec:  host: nginx  subsets:  - name: v1  labels:  version: v1  - name: v2  labels:  version: v2 |

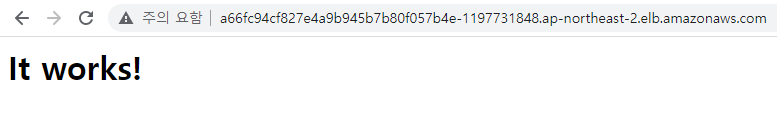
**9. 특정 버전만 서비스 하기**

VirtualService에서 subnet을 지정하면 특정 버전만 서비스 할 수 있게된다. 생성한 nginx는 총 2가지 버전이 있는데 v2만 서비스 할 수 있게끔 구성하였다.

|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: httpdpage-nginx  spec:  hosts:  - "\*"  gateways:  - service-gateway  http:  - route:  - destination:  host: httpdpage  subset: v1  port:  number: 80  weight: 50  - destination:  host: nginx  subset: v2  port:  number: 80  weight: 50 |

nlb 접근시 hello, world 페이지와 httpd 페이지만 표시되는걸 확인할 수 있다.





**10. Circuit breaking**

Circuit breaking은 문제가 되는 기능자체를 동작하지 않도록 하여, 장애가 전파되지 않게 해준다.

Circuit break 대상이 되는 httpbin 앱을 설치해준다. httpbin은 echo 응답앱이다.

|  |
| --- |
| $ kubectl label namespace default istio-injection=enabled  $ kubectl apply -f ./samples/httpbin/httpbin.yaml |

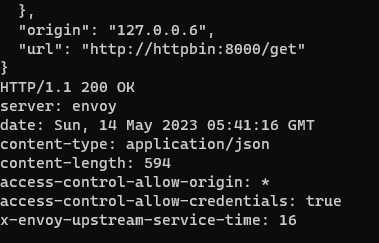
마이크로서비스 로드 테스트 툴인 fortio를 설치한다.

|  |
| --- |
| $ kubectl apply -f samples/httpbin/sample-client/fortio-deploy.yaml |

fortio으로 httpbin 앱에 요청을 전송해준다.

|  |
| --- |
| $ export FORTIO\_POD=$(kubectl get pods -lapp=fortio -o 'jsonpath={.items[0].metadata.name}')  $ kubectl exec "$FORTIO\_POD" -c fortio -- /usr/bin/fortio curl -quiet http://httpbin:8000/get |

응답은 다음과 같다.



Prometheus를 설치한다.

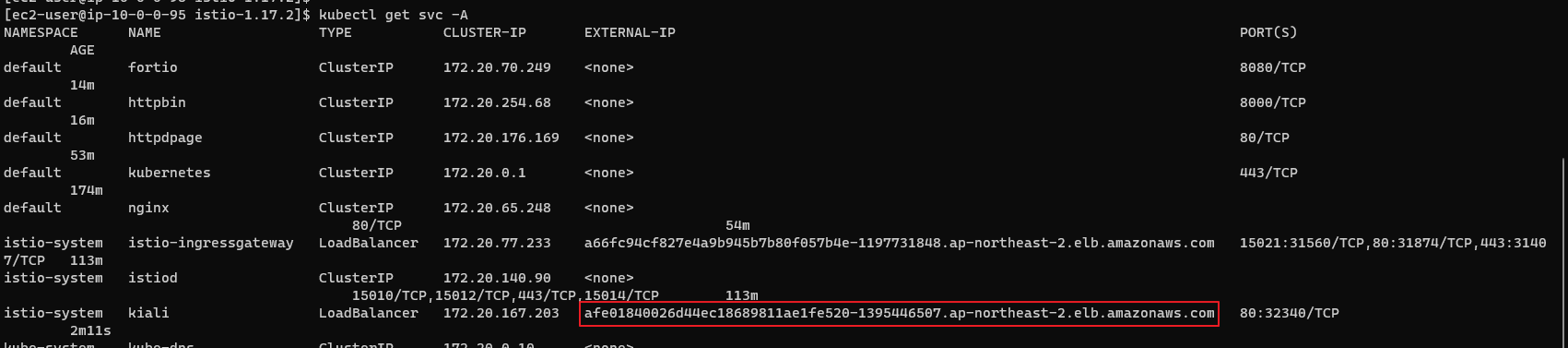
|  |
| --- |
| $ kubectl apply -f https://raw.githubusercontent.com/istio/istio/master/samples/addons/prometheus.yaml |

kiali를 설치하여 확인해준다.

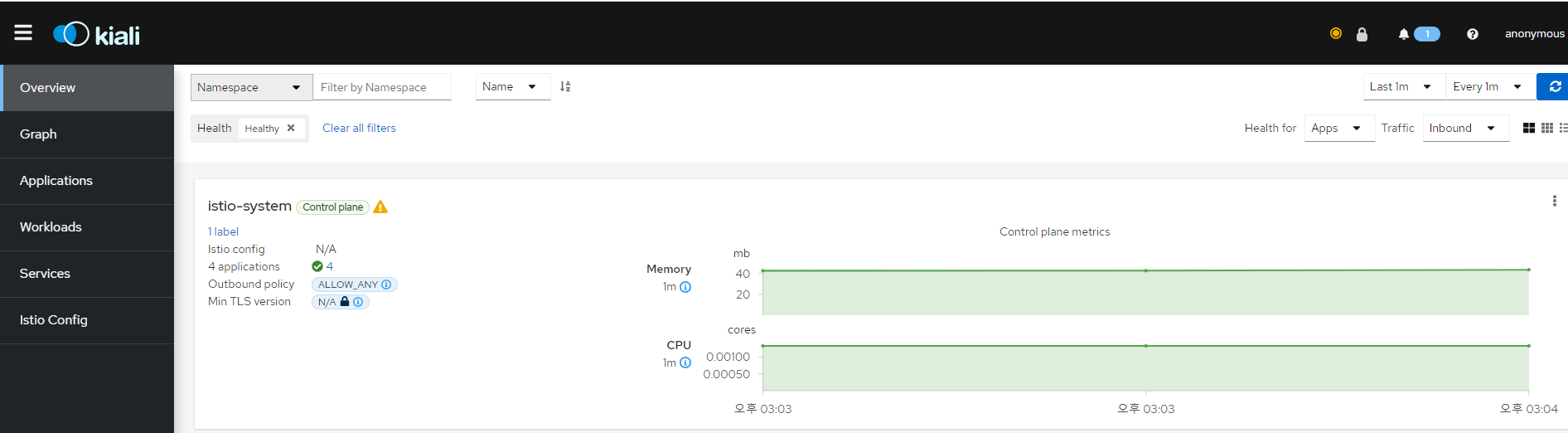
|  |
| --- |
| $ kubectl apply -f https://raw.githubusercontent.com/istio/istio/master/samples/addons/kiali.yaml |

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: kiali  namespace: istio-system  spec:  selector:  app: kiali  type: LoadBalancer  ports:  - protocol: TCP  port: 80  targetPort: 20001 |

kiali에 들어가준다.



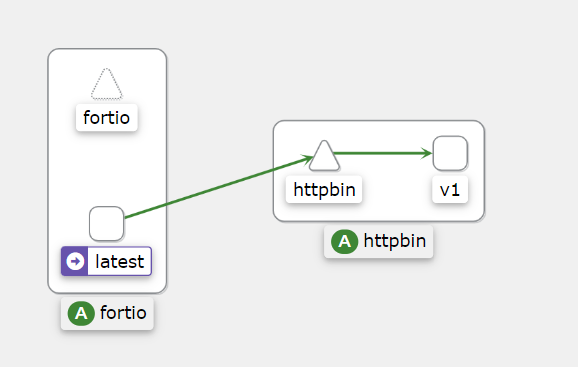
다음 페이지가 표시된다.



그래프에 들어가준다.

테스트를 진행한다.

|  |
| --- |
| $ kubectl exec "$FORTIO\_POD" -c fortio -- /usr/bin/fortio load -c 2 -qps 0 -n 20 -loglevel Warning http://httpbin:8000/get |



그래프의 흐름을 관찰할 수 있다.

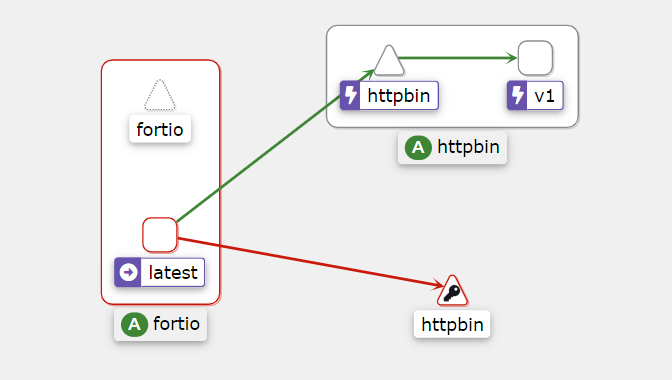
**11. 최소한의 커넥션만 허용하도록 설정**

|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: DestinationRule  metadata:  name: httpbin  spec:  host: httpbin  trafficPolicy:  connectionPool:  tcp:  maxConnections: 1  http:  http1MaxPendingRequests: 1  maxRequestsPerConnection: 1  outlierDetection:  consecutiveErrors: 1  interval: 1s  baseEjectionTime: 3m  maxEjectionPercent: 100 |

테스트 -> 동시 연결수를 최대 3개로 늘리고 30개의 요청을 보낸다.

|  |
| --- |
| $ kubectl exec "$FORTIO\_POD" -c fortio -- /usr/bin/fortio load -c 3 -qps 0 -n 30 -loglevel Warning http://httpbin:8000/get |

Circuit break가 작동되었을 때 다음과 같이 표시된다.

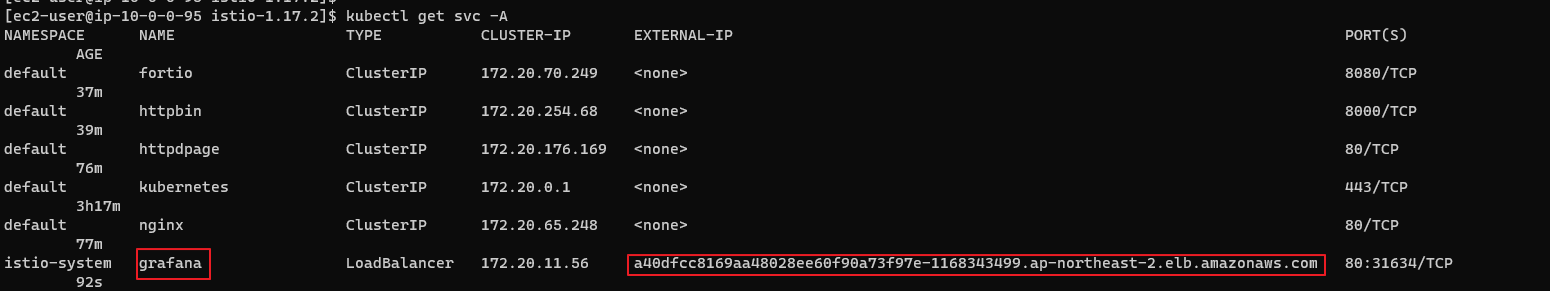


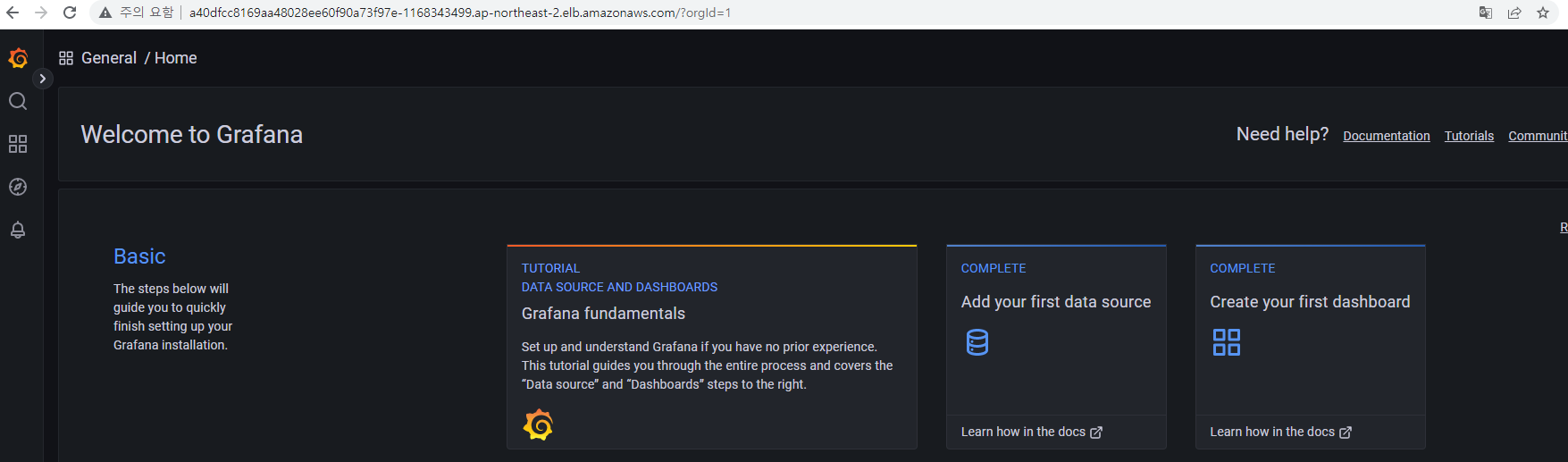
**12. Grafana 설치**

|  |
| --- |
| $ kubectl apply -f https://raw.githubusercontent.com/istio/istio/master/samples/addons/grafana.yaml |

로드벨런서 서비스 생성

|  |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: grafana  namespace: istio-system  spec:  selector:  app: grafana  type: LoadBalancer  ports:  - protocol: TCP  port: 80  targetPort: 3000 |
| $ kubectl get svc -A |





**13. jaeger install**

Jaeger는 end to end 분산 추적 시스템으로 사용자는 복잡한 분산 시스템에서 트랜잭션을 모니터링하고 문제를 해결할 수 있다.

|  |
| --- |
| $ kubectl apply -f https://raw.githubusercontent.com/istio/istio/master/samples/addons/jaeger.yaml |

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: jaeger  namespace: istio-system  labels:  app: jaeger  spec:  selector:  matchLabels:  app: jaeger  template:  metadata:  labels:  app: jaeger  annotations:  sidecar.istio.io/inject: "false"  prometheus.io/scrape: "true"  prometheus.io/port: "14269"  spec:  containers:  - name: jaeger  image: "docker.io/jaegertracing/all-in-one:1.18"  env:  - name: BADGER\_EPHEMERAL  value: "false"  - name: SPAN\_STORAGE\_TYPE  value: "badger"  - name: BADGER\_DIRECTORY\_VALUE  value: "/badger/data"  - name: BADGER\_DIRECTORY\_KEY  value: "/badger/key"  - name: COLLECTOR\_ZIPKIN\_HTTP\_PORT  value: "9411"  - name: MEMORY\_MAX\_TRACES  value: "50000"  - name: QUERY\_BASE\_PATH  value: /jaeger  livenessProbe:  httpGet:  path: /  port: 14269  readinessProbe:  httpGet:  path: /  port: 14269  volumeMounts:  - name: data  mountPath: /badger  resources:  requests:  cpu: 10m  volumes:  - name: data  emptyDir: {}  ---  apiVersion: v1  kind: Service  metadata:  name: tracing  namespace: istio-system  labels:  app: jaeger  spec:  type: LoadBalancer  ports:  - name: http-query  port: 80  protocol: TCP  targetPort: 16686  selector:  app: jaeger  ---  # Jaeger implements the Zipkin API. To support swapping out the tracing backend, we use a Service named Zipkin.  apiVersion: v1  kind: Service  metadata:  labels:  name: zipkin  name: zipkin  namespace: istio-system  spec:  ports:  - port: 9411  targetPort: 9411  name: http-query  selector:  app: jaeger |

서비스를 조회하면 다음과 같은 주소가 표시된다.

|  |
| --- |
| $ kubectl get svc -A |



접속하면 다음과 같은 페이지가 표시된다.

