아래와 같이 istioctl를 설치해준다.

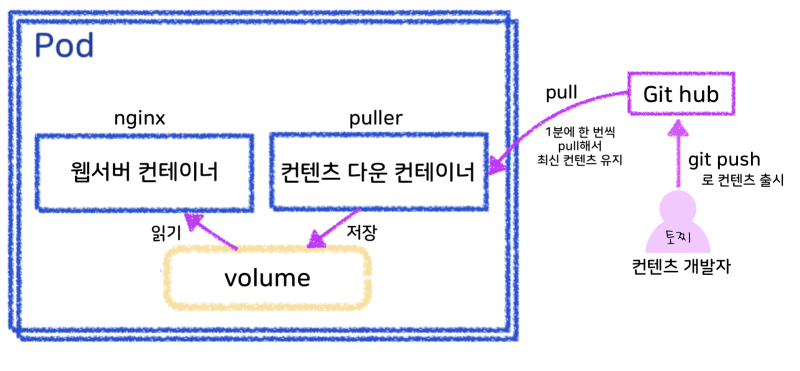
|  |
| --- |
| $ cd ~  $ curl -L https://istio.io/downloadIstio | ISTIO\_VERSION=1.12.1 TARGET\_ARCH=x86\_64 sh -  $ cd istio-1.12.1  $ sudo ln -s $PWD/bin/istioctl /usr/bin/istioctl  또는  $ export PATH=$PWD/bin:$PATH |

istioctl을 설치했으면 이제 istio를 설치해주자.

|  |
| --- |
| $ istioctl install --set profile=demo -y |

그리고 어떤 서비스 구축에 앞서 istio의 sidecar injection 기능을 활성화해야한다. istio는 Pod에 sidecar(Envoy)를 Proxy기반으로 트래픽을 컨트롤하는 구조이기 때문에 Pod가 생성될 때 자동으로 sidecar가 생성되도록 injection 기능을 활성화하는 것이다.

* sidecar란: 원래 사용하려던 기본 컨테이너의 기능을 확장하거나 강화하는 용도의 컨테이너를 추가하는 패턴이다.



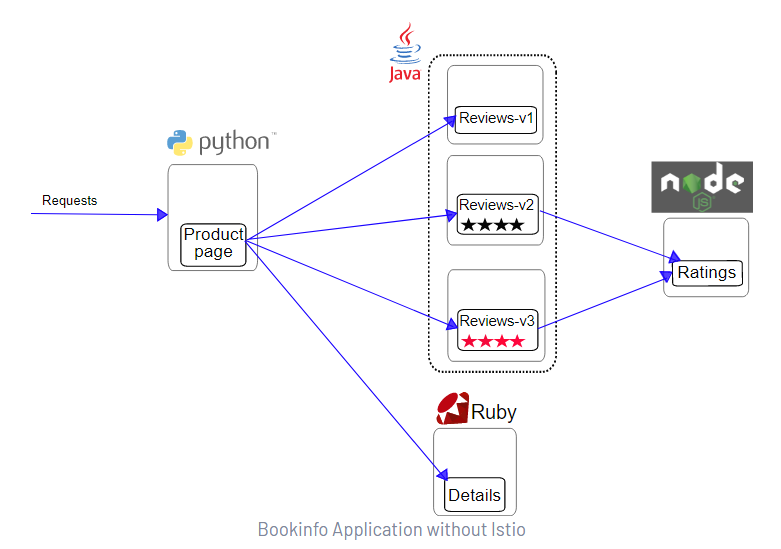
이제 default namespace에 sidecar label을 추가해주자.

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

이제 bookinfo 샘플 애플리케이션을 사용하여 istio에 대해서 구축해보자.

|  |
| --- |
| $ cd ~  $ git clone https://github.com/istio/istio.git ; cd istio  $ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml |

우선 bookinfo 서비스는 아래와 같이 구성되어있다.



텍스트이(가) 표시된 사진

자동 생성된 설명

우선 위와 같이 productpage, detail, rating, review (v1~v3)이 배포되었음을 알 수 있다.

그리고 또 SVC를 보면 아래와 같이 모든 서비스들은 ClusterIP로 구성되어있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이는 내부접근만 가능하고, 외부에서는 접근이 불가능한 상태이다. (API Gateway를 생성해줘야겠다.)

우선 외부접근을 열어주자.

istio는 서비스를 외부로 노출시키기 위해서 istio의 gateway를 사용한다. (k8s의 ingress, service를 미사용)

istio의 gateway는 k8s 커스텀 리소스 타입으로, istio로 들어오는 traffic의 통로가 된다. 또한 gateway는 pod 형식이며 Load Balancer 타입으로 서비스된다. istio gateway를 등록 후 gateway를 통해 서비스할 호스트를 Virtual Service로 등록된다.

|  |  |
| --- | --- |
| 구분 | 데이터 흐름 |
| 기본 | Client -> Istio Gateway -> Istio Virtual Service -> K8s Service |
| Bookinfo  상세 | Client -> Istio-ingressgateway (kind: Service) -> instio-ingressgateway (kind: Pod) -> bookinfo-gate (kind: gateway) -> bookinfo (kind: Virtual Service) -> productpage (kind: service) -> productpage (kind: pod) |

한 번 booking gateway의 manifest 파일을 분석 해보자.

|  |
| --- |
| apiVersion: networking.istio.io/v1alpha3  kind: Gateway <- 커스텀 Kind  metadata:  name: bookinfo-gateway  spec:  selector:  istio: ingressgateway # use istio default controller  servers:  - port:  number: 80 <- 80번으로 리슨  name: http  protocol: HTTP <- http로 client 트래픽 받음  hosts:  - "\*" <- 모든 호스트 허용  ---  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService < 위의 gateway 트래픽을 수용해 라우팅 룰이 적용  metadata:  name: bookinfo  spec:  hosts:  - "\*"  gateways:  - bookinfo-gateway  http:  - match: < 매치되는 URL을 확인해 route 정책 된다.  - uri:  exact: /productpage  - uri:  prefix: /static  - uri:  exact: /login  - uri:  exact: /logout  - uri:  prefix: /api/v1/products  route:  - destination: < 매치된 모든 URL이 모드 productpage의 9080으로 포워딩  host: productpage  port: |

이제 Gateway를 생성해주자.

|  |
| --- |
| $ vim gateway.yaml |
| apiVersion: networking.istio.io/v1alpha3  kind: Gateway  metadata:  name: bookinfo-gateway  spec:  selector:  istio: ingressgateway # use istio default controller  servers:  - port:  number: 80  name: http  protocol: HTTP  hosts:  - "\*"  ---  apiVersion: networking.istio.io/v1alpha3  kind: VirtualService  metadata:  name: bookinfo  spec:  hosts:  - "\*"  gateways:  - bookinfo-gateway  http:  - match:  - uri:  exact: /productpage  - uri:  prefix: /static  - uri:  exact: /login  - uri:  exact: /logout  - uri:  prefix: /api/v1/products  route:  - destination:  host: productpage # Service Name  port:  number: 9080 |

|  |
| --- |
| $ kubectl apply -f gateway.yaml |

그리고 정상 적으로 적용이 됐는지 확인해주자.

|  |
| --- |
| $ istioctl analyze    $ kubectl get gateway    $ kubectl get pod -n istio-system -l istio=ingressgateway    $ kubectl get svc istio-ingressgateway -n istio-system --show-labels |

그리고 ALB를 사용하기 위해서 아래와 같은 작업을 진행 해준다.

|  |
| --- |
| $ helm repo add eks https://aws.github.io/eks-charts  $ helm install aws-load-balancer-controller eks/aws-load-balancer-controller --set clusterName=wsi-korea-cluster -n kube-system |

그리고 istio 설정을 변경해주자.

Istio의 ingress gateway의 service type을 nodeport로 변경하고 alb가 istio ingress gateway로 health check를 할 수 있도록 관련 정보를 넣어준다.

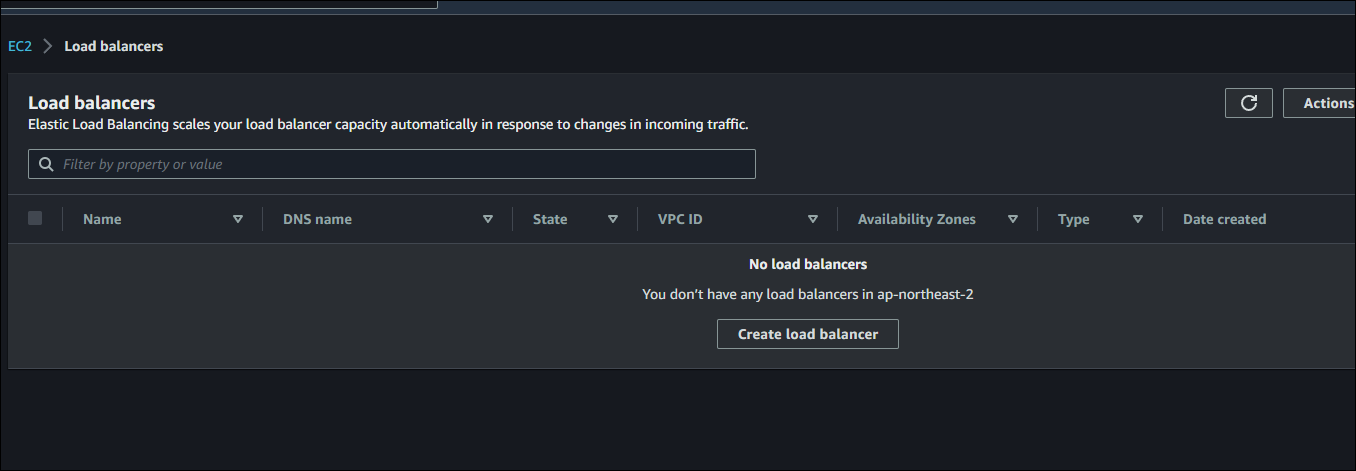
|  |
| --- |
| $ kubectl get service istio-ingressgateway -n istio-system -o jsonpath='{.spec.ports[?(@.name=="status-port")].nodePort}' |

우선 Port Number를 알아와주고 아래와 같이 istiooperator에 아래와 같이 추가해준다.

|  |
| --- |
| $ vim operator.yaml  apiVersion: install.istio.io/v1alpha1  kind: IstioOperator  metadata:  namespace: istio-system  name: installed-state  spec:  profile: default  components:  egressGateways:  - name: istio-egressgateway  enabled: true  k8s:  hpaSpec:  minReplicas: 2  ingressGateways:  - name: istio-ingressgateway  enabled: true  k8s:  hpaSpec:  minReplicas: 2  service:  type: NodePort # ingress gateway 의 NodePort 사용  serviceAnnotations: # Health check 관련 정보  alb.ingress.kubernetes.io/healthcheck-path: /healthz/ready  alb.ingress.kubernetes.io/healthcheck-port: "31622" # 위에서 얻은 port number를 사용  pilot:  enabled: true  k8s:  hpaSpec:  minReplicas: 2  meshConfig:  enableTracing: true  defaultConfig:  holdApplicationUntilProxyStarts: true  accessLogFile: /dev/stdout  outboundTrafficPolicy:  mode: REGISTRY\_ONLY |

|  |
| --- |
| $ istioctl install -f operator.yaml |

그러면 기존 service type이 loadbalancer에서 nodeport로 변경했기에 아래와 같이 CLB가 삭제되는 것을 볼 수 있다.



그리고 ALB Ingress Controller를 생성해주자.

|  |
| --- |
| apiVersion: networking.k8s.io/v1  kind: Ingress  metadata:  name: ingress-alb  namespace: istio-system  annotations:  kubernetes.io/ingress.class: alb  alb.ingress.kubernetes.io/scheme: internet-facing  spec:  rules:  - http:  paths:  - pathType: Prefix  path: /  backend:  service:  name: istio-ingressgateway  port:  number: 80 |

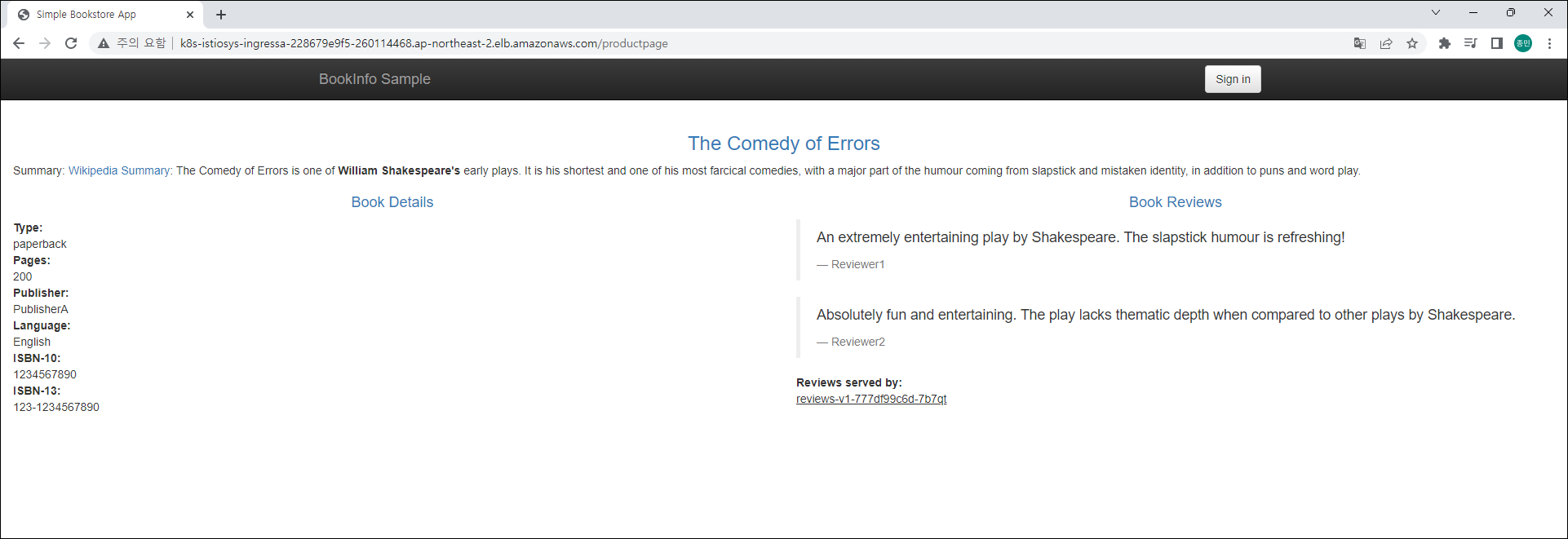
그리거면 ALB가 생긴 것을 볼 수 있다.

텍스트, 스크린샷, 모니터, 검은색이(가) 표시된 사진

자동 생성된 설명

그리고 해당 ALB 주소로 접근 해보자.

http://k8s-istiosys-ingressa-228679e9f5-260114468.ap-northeast-2.elb.amazonaws.com/productpage



위와 같이 정상 라우팅 되는 것을 볼 수 있다.