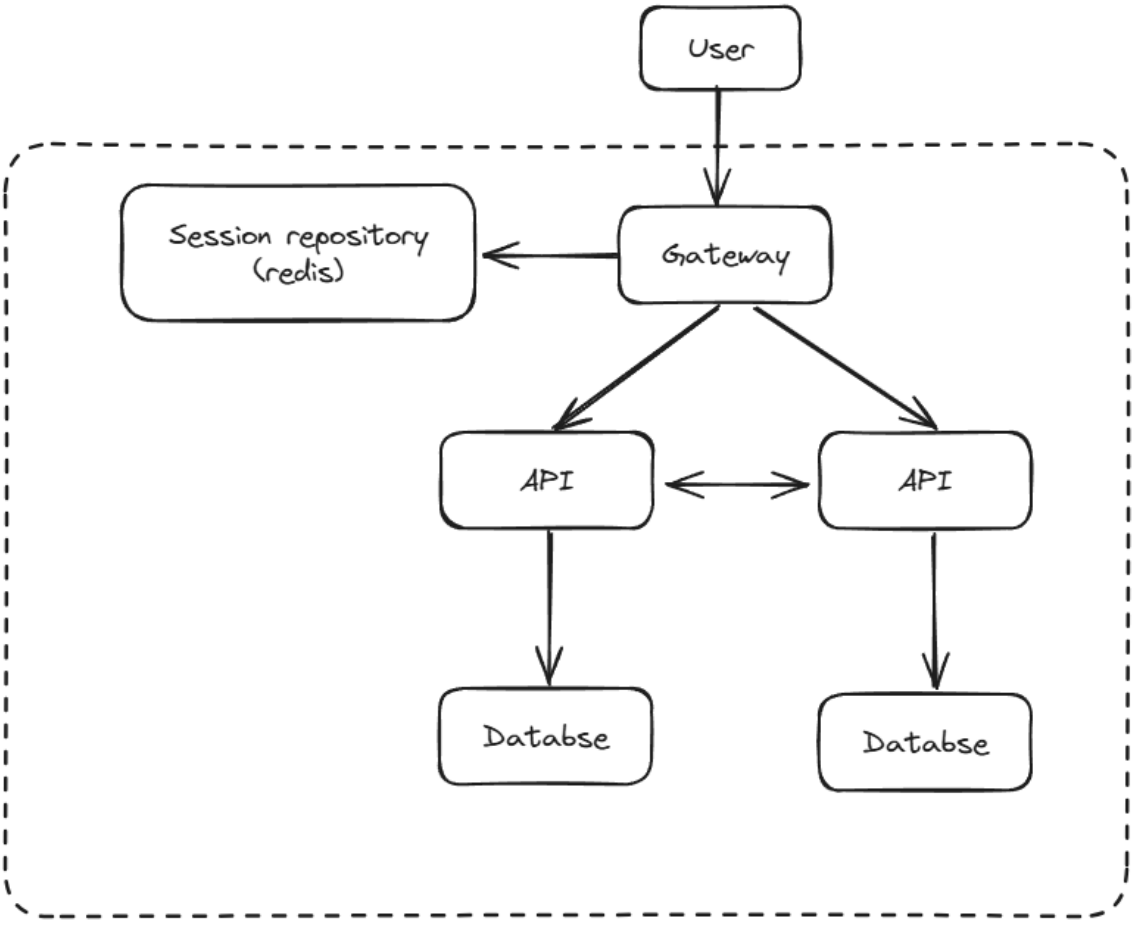
9장 API 게이트웨이

게이트웨이란?

API 게이트웨이는 마이크로 서비스와 같은 분산 아키텍처에서 클라이언트가 직접 내부 API를 연결하지 않도록 하기 위해 일반적으로 사용하는 패턴이다. 시스템에 이러한 진입점을 설정해놓으면 보안, 모니터링 및 복원력과 같이 공통으로 발생하는 이슈를 다루는 데에도 사용할 수 있다.



API 게이트웨이는 시스템의 진입점이다. 마이크로서비스와 같은 분산 시스템에서 클라이언트를 내부 서비스 API로부터 분리하기 위한 편리한 방법인데 이렇게 클라이언트와 분리해놓으면 내부 API는 변경이 자유롭다.

모놀리스에서 마이크로서비스로 옮겨가는 중이라고 가정하면, 이 경우 API 게이트웨이는 모놀리스 교살기로 사용할 수 있는데 레거시 애플리케이션이 새로운 아키텍처로 마이그레이

션이 끝날 때까지 클라이언트가 눈치채지 못하게 레거시 애플리케이션의 앞단에서 클라이언트에게 서비스를 전달해줄 수 있다.

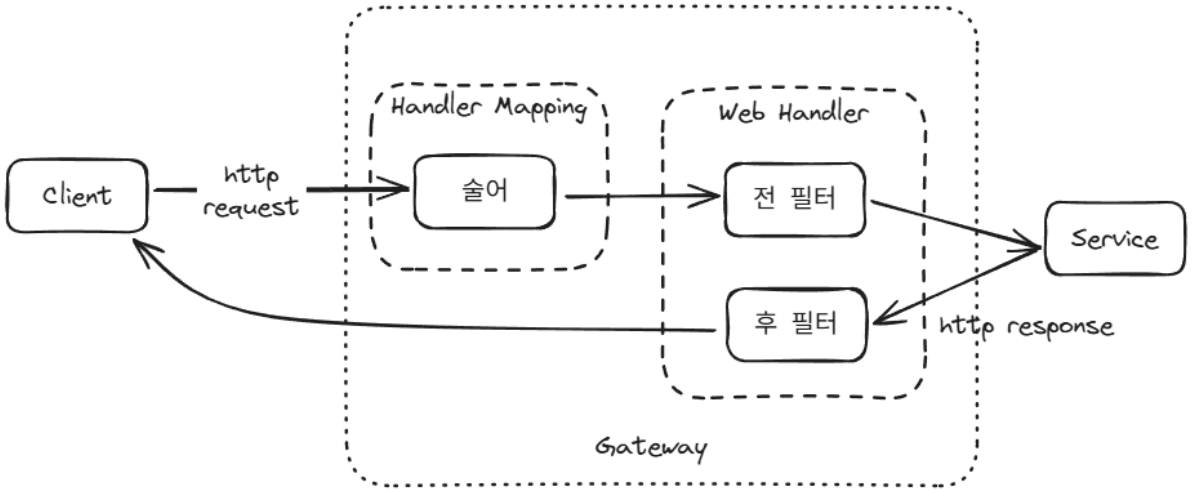
- 다른 서비스를 호출할 때 오류가 확산되는 것을 막기 위해 서킷 브레이커를 설정하거나 내부 서비스에 대한 모든 호출에 재시도 및 타임아웃을 설정
- 진입하는 트래픽을 제어하고 사용자의 회원 등급과 같은 기준에 따라 사용량을 제한하는 정책을 적용
- 서비스에서 인증 및 권한 설정을 구현하고 토큰을 다른 서비스에 전달

그러나 이러면 복잡성이 높아진다. 또한 시스템에 네트워크 홉을 새롭게 추가하기 때문에 결과적으로 응답 시간을 늘린다. 물론 대부분의 경우, 이 비용 증가는 무시해도 될 만한 수준이지만 그래도 염두에 두어야 한다. 게이트웨이는 시스템의 진입 지점이기 때문에 단일 실패 지점이 될 위험을 가지고 있다. 이에 대한 기본적 완화 전력으로, 4장에서 컨피그 서버에 대해 설명한 방식과 같이 복제본을 2개 이상 배포해야 한다.

경로와 술어의 정의

스프링 클라우드 게이트웨이는 3개의 주된 구성 요소로 이루어져 있다.

- 경로: 고유한 ID, 라우트를 따라갈지 여부를 결정하는 술어의 모음, 술어가 허용한 경우 요청을 전달하기 위한 URL, 요청을 다운 스트림으로 전달하기 전이나 후에 적용할 필터의 모음, 이렇게 네 가지 사항으로 고유하게 결정된다.
- 술어: 경로, 호스트, 헤더, 쿼리 매개변수, 쿠키, 본문 등 HTTP 요청의 항목에 해당한다.
- 필터: 요청을 다른 서비스로 전달하기 전이나 후에 HTTP 요청 또는 응답을 수정한다.



- 술어를 통해 하나의 경로와 일치하면 게이트웨이의 HandlerMapping은 이 요청을 게이트웨이의 WebHandler로 보내고 웹 핸들러는 다시 일련의 필터를 통해 요청을 실행한다.
- 필터 체인은 두 가지가 있는데, 첫 번째 필터 체인은 요청을 다운스트림 서비스로 보내기 전에 실행할 필터를 가지고 있고 또 다른 체인은 다른 서비스로부터 받은 응답을 클라이언트에게 전달하기 전에 실행할 필터를 가지고 있다.
- 라우트는 최소한 고유한 ID, 요청을 전달할 URI, 하나 이상의 술어로 구성되어야 한다. 스프링 클라우드 게이트웨이 프로젝트에는 경로 설정에 사용할 수 있는 다양한 술어를 제공하고 있는데, 이를 통해 쿠키, 헤더, 호스트, 메서드, 패스, 쿼리, 리모트주소 등 HTTP 요청의 여러 요소와 일치하는지 확인할 수 있다. 또한 이들을 AND 조건으로 묶어서 사용할 수도 있다.

필터를 통한 요청 및 응답 처리

라우터와 술어만으로도 애플리케이션에서 프록시 역할을 할 수 있지만 스프링 클라우드 게이트웨이를 실제로 강력하게 해주는 것은 바로 필터다.

사전 필터는 들어오는 요청을 다운스트림 애플리케이션으로 전달하기 전에 실행

- 요청의 헤더를 변경
- 사용률 제한 및 서킷 브레이크 적용
- 프락시 요청에 대한 재시도 및 타임아웃 정의
- OAuth2 및 오픈ID 커넥트를 사용한 인증 흐름 시작

사후 필터는 다운스트림 애플리케이션에서 응답을 받은 후에 클라이언트로 보내기 전에 응답에 대해 적용

- 보안 헤더 설정
- 응답 본문에서 민감한 정보를 변경

스프링 클라우드 게이트웨이는 다양한 종류의 작업을 수행할 수 있는 필터를 번들로 제공하는데 이를 통해 요청에 헤더 추가, 서킷 브레이커 설정, 웹 세션 저장, 실패시 요청 재시도, 사용률 제한 활성화 등을 할 수 있다.

Resilience4J로 내결함성 개선

분산 시스템에서는 구성 요소 간의 통합 지점에 서킷 브레이커를 설치할 수 있다. 장애 발생 횟수가 특정 임곗값을 초과하면 서킷 브레이커가 실행되고 회로는 개방 상태로 전환된다.

회로가 열려 있는 동안 게이트웨이와 서비스 간의 통신은 허용되지 않는다. 서비스에 전달해야 하는 요청은 즉시 실패한다. 이때는 클라이언트에게 오류를 반환하거나 폴백 논리를 실행할 수 있다. 시스템이 복구할 수 있는 충분한 시간이 지나면 서킷 브레이커는 반개방 상태로 전환되어 카탈로그 서비스를 호출한다. 이 호출은 해당 서비스를 계속 연결해도 문제가 없을지 확인하기 위한 탐색 단계의 호출이다. 이 호출이 성공하면 서킷 브레이커가 리셋되어 폐쇄 상태로 전환되고 실패하면 서킷 브레이커는 다시 개방 상태로 되돌아간다.

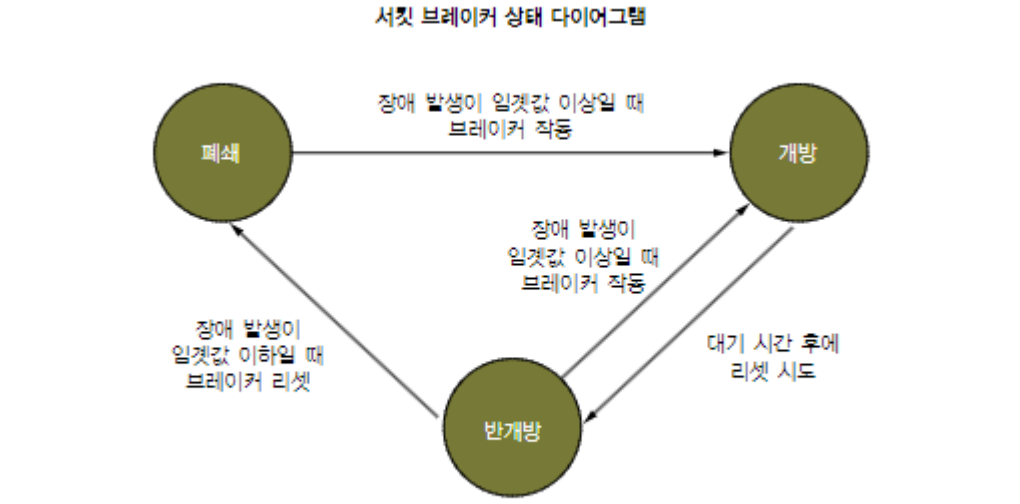


그림 9.4 서킷 브레이커는 서비스에 대해 허용되는 최대 고장 횟수를 초과할 때 서비스로의 통신을 차단함으로써 내결함성을 보장한다. 이 논리는 폐쇄, 개방, 반개방의 세 가지 상태를 기반으로 한다.

재시도와 달리 서킷 브레이커가 작동하면 더 이상 해당 서비스에 대한 호출이 허용되지 않는다. 재시도와 마찬가지로, 서킷 브레이커의 작동은 임곗값과 타임아웃에 따라 다르며, 호출할 폴백 메서드를 정의할 수도 있다. 높은 복원력을 통해 이루고자 하는 목표는 실패가 일어나는 상황에서도 사용자가 시스템을 계속 사용할 수 있도록 유지하는 것이다. 서킷 브레이커 작동과 같은 최악의 경우라도 우아한 성능 저하를 보장해야 한다. 이를 위해 폴백 메서드에 대해 다양한 전략을 채택할 수 있다. 예를 들면 GET 요청의 경우 기본값 혹은 캐시에 마지막으로 저장된 값을 반환할 수 있다.

스프링 클라우드 게이트웨이는 기본적으로 스프링 클라우드 서킷 브레이커와 통합되어 모든 서비스와의 상호작용을 보호하는데 사용할 수 있는 서킷브레이커 게이트웨이 필터를 제공한다.

서킷 브레이커, 재시도 및 시간 제한의 결합

복원력에 대한 여러 개의 패턴을 결합할 때, 이들을 적용하는 순서가 중요하다. 스프링 클라우드 게이트웨이는 TimeLimter(혹은 HTTP 클라이언트의 타임아웃)를 가장 먼저 적용하고, 그다음으로 CircuitBreaker 필터를 적용한 후에 마지막으로 Retry를 시도한다.

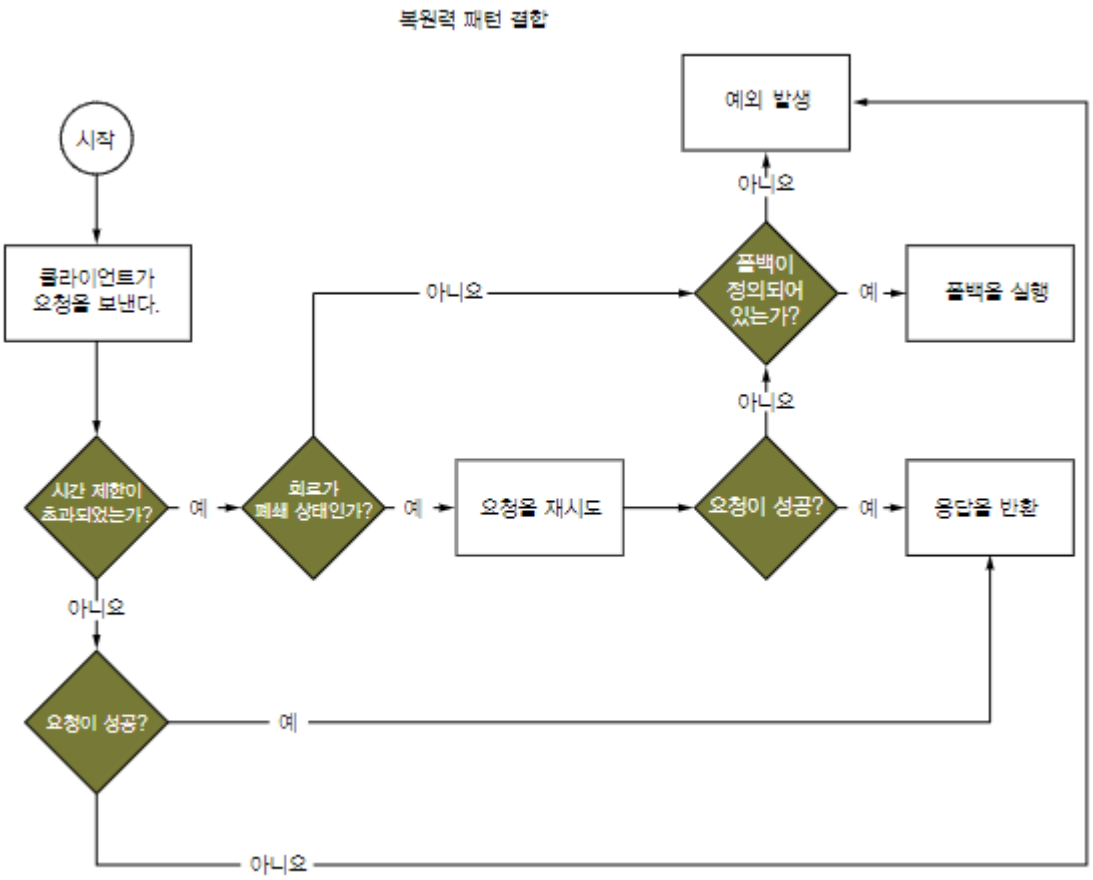


그림 9.5 여러 개의 복원력 패턴이 구현되면 특정 순서로 적용된다.

스프링 클라우드 게이트웨이와 레디스를 통한 요청 사용률 제한

사용률 제한은 애플리케이션이 전송하거나 수신하는 트래픽의 사용률을 제어하기 위해 사용하는 패턴으로, 시스템을 견고하게 만들고 복원력을 높이는 데 도움이 된다.

- HTTP 상호작용의 맥락에서 패턴을 적용하여 네트워크 트래픽을 제어
- 클라이언트 측 사용률 제한은 주어진 기간 동안 다운스트림 서비스로 전송되는 요청 수를 제한하기 위한 것
- 사용률 제한을 통해 DoS도 피할 수 있으나, 벌크헤드 패턴을 사용해 동시 요청 수에 대한 제약 조건을 설정하고 차단된 요청은 큐에 넣는 것이 더 적합 (적응형 벌크헤드를 사용하면, 동시성 제한이 알고리즘에 의해 동적으로 업데이트되기 때문에 클라우드 인프라의 탄력성에 더 잘 적응)
- 서버 측 속도 제한은 주어진 기간에 서비스(또는 클라이언트)가 수신하는 요청 수를 제한하기 위한 것 (DoS 공격에서 보호하기 위해 API 게이트웨이에 구현할 때 유용)

- 사용자가 특정 시간 기간 내에 허용된 요청 수를 초과하면 HTTP 429 -Too Many Requests 상태로 모든 추가 요청이 거부

요청 사용률 제한 설정

요구 사항에 따라 RequestRateLimiter 필터를 모든 라우터에 적용하는 기본 필터 혹은 특정 라우트에만 적용하는 필터로 설정가능

> RequestRateLimiter의 레디스에 대한 구현은 토큰 버킷 알고리즘을 기반으로 한다. 각 사용자에 대해 버킷이 할당되고 시간이 지나면서 토큰이 버킷에 특정한 비율(보충속도)로 떨어진다. 각 버킷은 최대 용량(버스트 용량)이 정해져 있으며 사용자가 요청을 할 때마다 토큰 하나가 버킷에서 삭제된다. 더 이상 토큰이 남아 있지 않으면 요청이 허용되지 않으며 사용자는 토큰이 버킷에 떨어지기를 기다려야 한다.

RequestRateLimiter 설정하고 요청해보면, 헤더에 사용률 제한 설정값과 시간 윈도(1)초 내에 허용된 남은 요청 수를 확인할 수 있음

```
HTTP/1.1 200 OK
Content-Type: application/json
X-RateLimit-Burst-Capacity: 20
X-RateLimit-Remaining: 19
X-RateLimit-Replenish-Rate: 10
X-RateLimit-Requested-Tokens: 1
```

이 정보가 악용될 소지가 있기 때문에 노출하고 싶지 않을 수도 있다. 또는 헤더 이름을 변경할 필요가 있을 수도 있다. spring.cloud.gateway.redis-rate-limiter 속성 그룹을 사용해 이에 대한 작동을 설정할 수 있다.

레디스를 통한 분산 세션 관리

클라우드 네이티브 애플리케이션은 상태를 갖지 않아야 한다

애플리케이션은 확장하거나 축소해야 하는데 만일 애플리케이션이 상태를 가지고 있다면, 인스턴스를 중지하거나 제거할 때마다 상태 정보는 소실될 것이다. 어떤 상태는 반드시 저장해야 한다. 그렇지 않으면 애플리케이션이 쓸모가 없을 수도 있다.

일반적인 아이디어는 애플리케이션은 상태를 갖지 않고 대신 상태를 저장하기 위해 데이터 서비스를 사용하는 것이다. 데이터 서비스는 높은 가용성, 복제 및 내구성을 보장해야 한다.

로컬 환경에서는 이런 측면을 무시할 수 있지만, 프로덕션 환경에서는 PostgreSQL과 레디스와 같은 클라우드 공급 업체가 제공하는 데이터 서비스에 의존하게 된다.

쿠버네티스 인그레스를 통한 외부 액세스 관리

스프링 클라우드 게이트웨이는 시스템의 진입 지점에서 여러 패턴 및 공통 문제를 구현할 수 있는 에지 서비스를 정의하는 데 유용하다. 에지 서비스는 폴라 북숍 시스템의 진입 지점을 나타낸다. 하지만 에지 서비스가 쿠버네티스 클러스터에 배포되면 클러스터 내에서만 액세스할 수 있다. 7장에서는 포트 전달 기능을 사용해 미니큐브 클러스터에 정의된 쿠버네티스 서비스를 로컬 컴퓨터에 노출했다. 개발할 때는 유용한 방법이지만 프로덕션 환경에는 적합하지 않다.

인그레스 API와 인그레스 컨트롤러 이해

인그레스

URI, 호스트네임, 경로 등과 같은 웹 개념을 이해하는 프로토콜-인지형(protocol-aware configuration) 설정 메커니즘을 이용하여 HTTP (혹은 HTTPS) 네트워크 서비스를 사용 가능하게 한다. 인그레스 개념은 쿠버네티스 API를 통해 정의한 규칙에 기반하여 트래픽을 다른 백엔드에 매핑할 수 있게 해준다.

인그레스 컨트롤러

클러스터 내의 인그레스가 작동하려면, 인그레스 컨트롤러가 실행되고 있어야 한다. 적어도 하나의 인그레스 컨트롤러를 선택하고 이를 클러스터 내에 설치한다. 규칙을 적용하고 클러스터 외부의 트래픽을 내부의 애플리케이션으로 라우팅하는 실제 구성 요소는 인그레스 컨트롤러다.

인그레스 컨트롤러 구현은 많이 있기 때문에 핵심 쿠버넷 배포판에는 기본 설정된 인그레스 컨트롤러가 없고 대신 어느 것을 사용할지는 각자에게 달려 있다. 인그레스 컨트롤러는 일반적으로 NGINX, HAProxy 또는 엔보이 같은 리버스 프록시를 사용해 구축하는데, 인그레스 컨트롤러의 예로는 앰배서더 에미서리, 컨투어, 인그레스 NGINX 등이 있다.

인그레스 NGIXN를 통해 배포할 시

- 인그레스 NGINX 사용
- kubectl을 사용해 배포 매니페스트를 클러스터에 적용
- 로컬 쿠버네티스 클러스터 관리는 미니큐브를 통해서 하기 때문에 내장된 애드온을 사용해 인그레스 NGINX를 기반으로 한 인그레스 기능을 활성화

```
minikube start --cpus 2 --memory 4gmb --driver docker --profi
minikube addons enable ingress --profile polar
// ingress 에드온 활성화를 하면 인그레스 nginx가 로컬 클러스터에 배포
kubectl get all -n ingress-nginx
// 배포가 완료되면 인그레스 nginx와 함께 배포된 다른 구성 요소에 대한 정보
```

마지막 명령은 -n ingres-nginx라는 새로운 인수를 사용하는 데 ingress-nginx라는 네임스페이스 내에 존재하는 모든 객체를 가져올 수 있다.

인그레스 객체 사용

에지 서비스는 애플리케이션 라우팅은 처리하지만 기본 인프라나 네트워크 설정에는 관여할 필요가 없다. 인그레스 리소스를 사용하면 이 두 책임을 분리할 수 있다. 개발자는 에지 서비스를 관리, 유지하고 플랫폼 팀은(링커드나 이스티오 같은 서비스 메시를 사용해) 인그레스 컨트롤러 및 네트워크 구성을 관리한다.

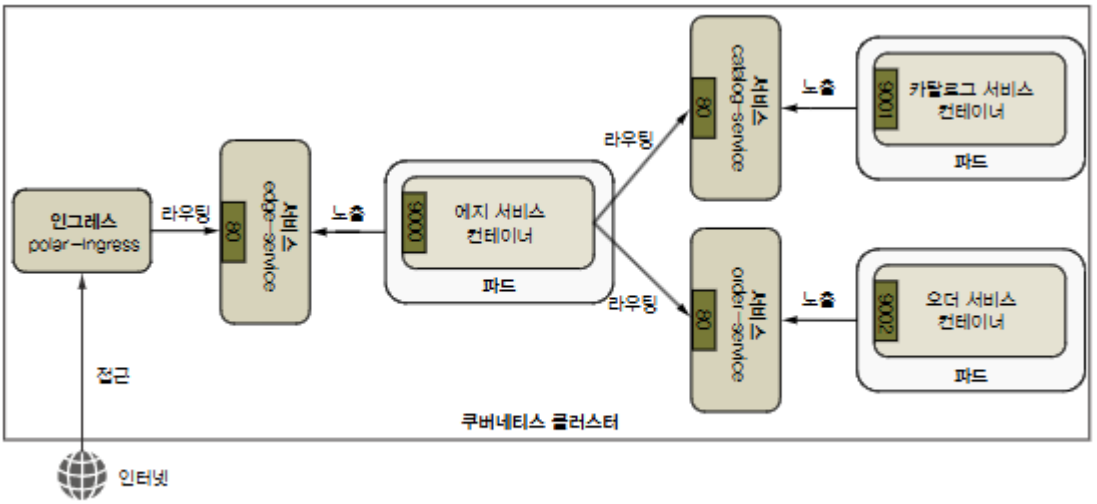


그림 9.6 클러스터에 대한 외부 액세스 관리를 위해 인그레스를 사용하는 경우 폴라 북숍 시스템의 배포 아키텍처

인그레스 라우트 정의와 설정은 HTTP 요청 시 사용하는DNS 이름을 기반으로 하는 것이 일반적이다. 로컬에서 작업하기 때문에 DNS 이름이 없다면 클러스터 외부에서 액세스할 수 있도록 인그레스에 제공된 외부 IP 주소로 호출할 수 있다. 리눅스에서는 미니큐브 클러스터에 할당된 IP 주소를 사용할 수 있다.

```
minikube ip --profile polar // ip주소 검색 명령어
```

macOS나 윈도우에서는 미니큐브가 도커에서 실행할 때 인그레스 애드온이 미니큐브 클러스터의 IP 주소를 사용하는 것을 아직 지원하지 않는다. 대신, minikube teunnel —profile

polar 명령을 사용해 클러스터를 로컬 환경에서 노출한 후에 127.0.0.1 IP 주소를 통해 클러스터를 호출해야 한다. 이것은 kubectl port-foward 명령과 비슷하지만 특정 서비스 대신 전체 클러스터에 적용된다는 점이 다르다.

인그레스 객체 확인 명령어

```
kubectl get ingress
```

맥, 윈도우 로컬 호스트에 미니큐브 클러스터 노출 명령어

```
minikube teunnel –profile polar
```

마지막으로 터미널에서 직접 테스트해볼 수 있다.