

# kubernetes\_pod\_port\_debug



9s 동안 생각함

- If we're facing a connection refused error on port 80 in Minikube, let's first check whether the container is actively listening. Try commands like `kubectl describe pod`, check readiness, liveness, ports per container, and inspect network policies. The issue might be that the service isn't binding to `0.0.0.0`. Finally, use port-forwarding or `kubectl exec` to test connectivity inside the pod.
- To resolve minikube connection issues, first check that the app inside the container is listening on the correct port, such as 8080 or 3000. Use `kubectl exec` to inspect the container's listening ports and check if it's binding to 0.0.0.0. Also, review the Deployment spec and logs for more insights.

☑ 완료

접속 거절(Connection refused)이면 “네트워크가 막힘”보다는 “그 포트에 듣는 프로세스가 없음” 쪽이 유력해요. 즉, Pod `10.244.1.247` 내부에서 80/tcp가 안 떠 있다는 뜻입니다. 빠르게 진단-해결 순서 드릴게요.

## 1) Pod 안에서 정말 80이 뜨는지 확인

# 이미지/컨테이너 확인

```
kubectl get pod etc-guestbook-ui-deployment-564dfbffb-7rzb5 -o  
jsonpath='{.spec.containers[*].image}{"\n"}{.spec.containers[*].ports}{"\n"}'
```

# 포트 리스닝/자체 접속 테스트

```
kubectl exec -it etc-guestbook-ui-deployment-564dfbffb-7rzb5 -- sh -lc '  
(command -v ss >/dev/null && ss -lntp) || netstat -lntp || true  
echo "--- curl localhost:80 ---"  
(curl -sS -v --max-time 3 http://127.0.0.1:80 || true)  
echo "--- env | grep -i port ---"  
env | grep -i port || true  
'
```

- `ss/netstat`에 `*:80`(또는 `0.0.0.0:80`)이 없으면 앱이 80으로 안 뜰 겁니다. 보통 3000/8080으로 뜨는 경우가 많아요.
- `curl localhost:80`도 실패하면 확정.

## 2) Deployment 스펙이 실제 앱 포트와 맞는지

```
kubectl get deploy etc-guestbook-ui-deployment -o yaml | sed -n '1,200p'
```

확인 포인트:

- `containers[].ports[].containerPort: 80` 으로 **잘못** 적혀 있지 않은가?
- 애플리케이션이 `PORT` 환경변수(예: 3000, 8080)를 요구하는데 안 줬는가?
- `args/command`로 다른 포트로 실행 중 아닌가?

필요하면 포트를 맞춰 수정:

```
# (예시) 앱이 3000으로 뜬다면
containers:
- name: ui
  image: your/image
  ports:
  - containerPort: 3000
```

## 3) 서비스/접속 경로를 올바르게 사용

Pod IP로 직접 치는 건 테스트 용도일 뿐이고, 보통은 Service를 통해 접근합니다.

### (A) 간단히 노드 포트로 노출

```
kubectl expose deploy etc-guestbook-ui-deployment --name ui-svc --port 80 --target-port 3000 --type=NodePort
```

#                                    ^ Service 포트                                    ^ 실제 앱 포트에 맞춰주세요(예: 3000/8080)

```
kubectl get svc ui-svc -o wide
minikube service ui-svc --url    # 접속 가능한 URL 자동 출력
```

## (B) 포트포워딩으로 바로 확인

```
kubectl port-forward deploy/etc-guestbook-ui-deployment 8080:3000
# 브라우저에서 http://localhost:8080
```

## 4) 로그로 오류 힌트 확인

앱이 구동은 되는데 바로 죽었다가 재시작 → 잠깐 뜨다 끊기는 경우도 있어요.

```
kubectl logs etc-guestbook-ui-deployment-564dfbffb-7rzb5 --tail=200
```

- 포트 바인딩 실패, 권한 문제, 프레임워크 기본 포트(예: 3000) 안내 메시지가 보일 수 있어요.

## 5) “minikube ssh — curl PodIP:80”이 실패하는 흔한 원인들

- **실제 앱 포트 ≠ 80**: 가장 흔함 (예: 앱은 3000으로 리스)
- **127.0.0.1에만 바인딩**: 컨테이너가 `localhost`로만 듣는 경우 Pod IP로는 거절됨  
→ 앱 실행 옵션을 `0.0.0.0`으로 바꾸세요.
- **NetworkPolicy**: 보통은 막히면 “타임아웃” 성격이지만, 혹시 정책이 있다면 서비스로 접근해 보세요.  
(지금은 “거절”이라서 리스닝 문제 쪽이 더 가능성 큼)

정리해서 액션

1. 위 1) 명령으로 컨테이너 내부 리스닝 포트를 확정
2. Deployment의 `containerPort`/실행 포트와 정렬
3. `kubectl port-forward` 또는 `minikube service`로 정상응답 확인
4. 필요 시 NodePort/Ingress는 그다음 단계

원하시면 지금 사용하는 Deployment/Service YAML을 붙여 주시면, 실제 포트 매칭을 바로 잡아 드릴게요.



보셨죠—`GUESTBOOK_UI_SERVICE_PORT=8086`. 지금 이 네임스페이스에 **port 8086**짜리 **Service(guestbook-ui)**가 있어요. 그런데 방금은 Pod IP의 **80**으로 때렸으니 거절난 겁니다. 다음처럼 확인해보세요.

## 1) Pod 안에서 실제 리스닝 포트 확인

```
kubectl exec -it etc-guestbook-ui-deployment-564dfbffb-7rzb5 -- sh -lc '(command -v ss >/dev/null && ss -lntp) || netstat -lntp || true
for p in 80 8080 8086; do
  echo "--- curl 127.0.0.1:$p ---"
  (curl -sS -v --max-time 2 http://127.0.0.1:$p >/dev/null || true)
done
'
```

- `*:8086`, `*:8080`, `*:80` 중 뭐가 뜨는지 보세요.
- `curl 127.0.0.1:8086`가 통하면 컨테이너는 8086으로 잘 떠있는 겁니다.

## 2) Pod IP로 바로 테스트(빠른 검증)

```
# 80 말고 8086로
```



```
minikube ssh -- curl -I 10.244.1.247:8086
```

### 3) Service/targetPort 정렬

서비스가 8086을 노출한다면 **targetPort**가 컨테이너가 실제로 듣는 포트와 같아야 합니다. 예시:

```
apiVersion: v1
kind: Service
metadata:
  name: guestbook-ui
spec:
  type: NodePort          # minikube에서 외부접속하려면 편함
  selector:
    app: guestbook-ui
  ports:
    - name: http
      port: 8086           # 클러스터 IP에서 보이는 포트
      targetPort: 8086     # ← 컨테이너가 실제로 듣는 포트로 맞추기(8080이면 8080으로)
      nodePort: 30086      # 선택(원하면 고정)
```

Deployment도 맞춰 주면 더 깔끔합니다:

```
containers:
- name: ui
  image: <이미지>
  ports:
    - containerPort: 8086    # 실제 리스 포트
# (중요) 앱이 127.0.0.1 말고 0.0.0.0로 바인딩하도록 설정
```

### 4) 손쉬운 접속 두 가지

- 포트포워딩

```
kubect1 port-forward deploy/etc-guestbook-ui-deployment 8080:8086
# 브라우저: http://localhost:8080
```

- **minikube service**

```
kubectl get svc guestbook-ui -o wide
minikube service guestbook-ui --url
```

## 참고로

- ``NGINX_HTTP_PORT_NUMBER=`` 가 비어 보이지만, 이미지에 따라 기본이 8080일 수도 있습니다. 그래서 8080/8086 둘 다 찍어보는 거예요.
- 앱이 ``127.0.0.1``로만 바인딩하면 Pod IP로는 접속이 거절됩니다. 꼭 ``0.0.0.0``로 바꿔주세요.

필요하시면 사용 중인 Deployment/Service YAML 붙여 주세요. 실제 포트 매칭 바로 잡아드릴게요.



### 5s 동안 생각함

- 서비스 셀렉터와 파드 레이블 간 불일치 문제를 해결해야 합니다. 파드는 8080 포트를 리스닝할 가능성이 있지만, 서비스는 80 포트를 타겟으로 하고 있어 연관이 맞지 않아 연결이 거부된 것 같습니다. NGINX는 기본적으로 8080을 사용하므로 서비스의 targetPort를 8080으로 바꿔야 할 수도 있습니다.

☑ 완료

좋아요, 두 군데가 어긋나 있어서 접속이 안 됐네요.

### 1. 포트 불일치

- ``bitnami/nginx``는 기본이 **8080**(비루트)로 떠요. 지금 ``containerPort: 80``으로 적으셨고, Service도 ``targetPort: 80``이라 Pod 안에서 80이 안 떠서 거절된 겁니다.

### 2. 라벨 불일치

- Deployment는 ``labels.app=guestbook-ui`` 인데, Service의 selector는 ``app=etc-guestbook-ui-deployment`` 로 되어 있어요. 이러면 Service가 Pod를 못 잡습니다.

아래처럼 정리해서 쓰면 바로 됩니다.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: etc-guestbook-ui-deployment
spec:
  replicas: 1
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: guestbook-ui
  template:
    metadata:
      labels:
        app: guestbook-ui
    spec:
      containers:
        - name: c-etc-guestbook-ui
          image: bitnami/nginx:latest
          env:
            # 명시적으로 8080으로 띄우기(이미 기본이지만 헛갈림 방지)
            - name: NGINX_HTTP_PORT_NUMBER
              value: "8080"
          ports:
            - name: http
              containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: guestbook-ui
spec:
  selector:
    app: guestbook-ui          # ← 라벨 맞춤
  ports:
    - name: http
      port: 8086                # 클러스터IP로 보이는 포트
      targetPort: 8080          # ← 컨테이너 실제 리스 포트와 일치
# minikube에서 밖에서 직접 보려면 아래 주석 해제
# type: NodePort
# ports:

```

```
# - name: http
#   port: 8086
#   targetPort: 8080
#   nodePort: 30086          # 원하는 고정 값
```

## 적용 & 확인 순서

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml

# 서비스가 Pod를 제대로 붙잡는지
kubectl get endpoints guestbook-ui

# Pod 내부 포트 체크(8080이 떠야 OK)
kubectl exec -it deploy/etc-guestbook-ui-deployment -- sh -lc '
  (ss -lntp || netstat -lntp) 2>/dev/null | grep -E ":8080\\b" || true
  curl -I 127.0.0.1:8080 || true
'

# 클러스터 내부에서 서비스로 확인
kubectl run tmp --rm -it --image=curlimages/curl --restart=Never -- \
  curl -I http://guestbook-ui:8086

# 로컬 브라우저로 바로 보고 싶으면(둘 중 택1)
kubectl port-forward deploy/etc-guestbook-ui-deployment 8080:8080 #
http://localhost:8080
# 또는
minikube service guestbook-ui --url
```

이렇게 맞추면 `minikube ssh -- curl <PodIP>:80`은 여전히 실패(80은 안 엽니다)하고,  
`minikube ssh -- curl <PodIP>:8080` 또는 `curl <ServiceIP>:8086`은 성공해야 정상이에요.

필요하시면 NodePort/Ingress까지 깔끔하게 이어서 잡아드릴게요.





## 27s 동안 생각함

- 사용자가 물어본 내용은 Bitnami/nginx 컨테이너에서 curl을 설치할 수 있는지 여부입니다. Bitnami/nginx는 보통 Debian 기반으로 apt를 사용하지만, non-root 환경에서는 'apt-get'이 작동하지 않습니다. root 권한 없이 curl을 설치하려면 임시 디버그 컨테이너를 실행해야 할 수 있습니다.
- Kubernetes에서 'kubectl debug' 명령을 사용하여 임시 컨테이너를 실행하여 curl 테스트를 할 수 있습니다. 임시 컨테이너는 동일한 네트워크를 공유하며, 이를 활용한 포트 테스트 가능. 대신 포트나 볼륨을 열 수 없다는 점은 유의. 비-루트 환경에서는 다른 방법들로 대체 가능.
- 대부분의 비트나미/nginx 컨테이너는 root 권한이 없고 패키지 관리자가 없어서 curl을 설치할 수 없습니다. 또한 설치 후에도 변경 사항은 재시작 시 사라집니다. 대신, 다음 옵션들을 제안할 수 있습니다: 1) 임시 디버그 컨테이너 사용, 2) BusyBox를 이용한 curl 실행.
- 사용자가 최근에 설정한 Minikube 환경에서, curl을 사용하기 위한 방법을 제시해야 합니다. 이때 임시 컨테이너를 실행하는 명령어를 통해 Pod의 네트워크를 공유하며, `kubectl debug` 명령어로 curl을 실행할 수 있습니다. 또한, Pod 이름을 모를 경우 라벨 셀렉터를 사용해 찾을 수도 있습니다.

🏁 완료

짧게 결론부터요: **실행 중인 컨테이너 안에 `curl`을 설치하는 건 보통 불가능하거나 비권장입니다.** (대부분 **비루트 계정**으로 돌아가고, 패키지 매니저가 없거나 있어도 권한이 없고, 설령 설치돼도 **재시작하면 사라짐.**)

대신 표준적인 디버깅 방법 3가지를 드릴게요.

## 1) Ephemeral container로 같은 Pod 안에서 `curl` 쓰기(추천)

Pod 네트워크 네임스페이스를 공유하므로 **\*\*컨테이너 내부 포트(127.0.0.1:8080)\*\***를 바로 찌를 수 있습니다.

```
# 1) Pod 이름 잡기 (라벨 기준)
POD=$(kubectl get po -l app=guestbook-ui -o
jsonpath='{.items[0].metadata.name}')
```

```
# 2) 에페멀 컨테이너 붙여서 바로 요청 보내기
kubectl debug -it pod/$POD \
```

```
--image=curlimages/curl:8.8.0 \  
--target=c-etc-guestbook-ui -- \  
sh -lc 'curl -I http://127.0.0.1:8080'
```

“성공이면 200/301 같은 헤더가 보입니다. 실패면 앱이 8080으로 안 뜨거나 `127.0.0.1`에만 바인딩되지 않았는지 다시 봐야 해요.”

## 2) 임시 `curl` 파드로 Service/PodIP 테스트

컨테이너 내부가 아니라 클러스터 내부에서 외부처럼 때리기가 목적일 때 유용합니다.

```
# 한 번 쓰고 지우는 임시 파드  
kubectl run -it --rm testcurl --image=curlimages/curl --restart=Never -- \  
sh -lc '  
    echo "[Svc]"; curl -I http://guestbook-ui:8086 || true  
    echo "[PodIP]"; curl -I http://10.244.1.247:8080 || true  
'
```

## 3) 로컬에서 포트포워딩

로컬 터미널에서 바로 확인하고 싶다면:

```
kubectl port-forward deploy/etc-guestbook-ui-deployment 8080:8080  
# 다른 터미널에서  
curl -I http://127.0.0.1:8080
```

## 왜 컨테이너 안 설치는 피하냐?

- 컨테이너는 **불변 이미지**가 원칙이라 런타임 설치는 재시작 시 유실.
- ``bitnami/nginx``는 **\*\*비루트(1001)\*\***로 동작 → ``apt-get`` 권한 없음. ``sudo``도 보통 없음.
- 디버깅은 에페멀 컨테이너/임시 파드가 정석이에요.

필요하시면 ``kubectl debug``가 안 붙는 경우(버전/권한 문제) 우회 방법도 바로 잡아드릴게요.



좋은 질문이에요 😊

쿠버네티스에서 **에페멀(ephemeral) 컨테이너**란, 이미 실행 중인 Pod에 일시적으로 붙여 쓰는 디버깅 용 컨테이너를 말합니다.

---

## 핵심 특징

- **일시적**: Pod 스펙(Deployment, ReplicaSet 등)에는 기록되지 않음. Pod가 재시작되면 사라집니다.
- **디버깅 목적**: ``kubectl debug`` 명령으로 주로 추가하며, ``curl``, ``bash``, ``tcpdump`` 같은 도구를 넣어 “옆에서 관찰”하는 역할을 합니다.
- **네트워크/IPC 공유**: 같은 Pod 안 네임스페이스를 공유하기 때문에 원래 컨테이너가 듣고 있는 포트(127.0.0.1:8080 등)에 그대로 접근할 수 있습니다.
- **제약**: Pod 내부에서 애플리케이션 실행 환경을 바꾸는 용도는 아님. 오직 **진단/검사**용.

---

## 일반 컨테이너와 비교

구분	일반 컨테이너	에페멀 컨테이너
생성 방식	Deployment/Pod YAML에 정의됨	`kubect1 debug` 명령으로 런타임 추가
라이프사이클	Pod와 함께 생성·재시작·관리	재시작 안 됨. Pod 끝나면 같이 끝남
목적	실제 서비스 실행	디버깅/관찰/네트워크 테스트

## 예시

```
# Pod 이름 확인
POD=$(kubect1 get pod -l app=guestbook-ui -o
jsonpath='{.items[0].metadata.name}')

# curl 도구가 들어있는 에페멀 컨테이너를 추가
kubect1 debug -it $POD --image=curlimages/curl --target=c-etc-guestbook-ui
```

들어가면 바로 ``curl``, ``sh`` 등을 써서 원래 컨테이너 옆에서 네트워크 확인이 가능합니다.

👉 정리하면: **“실행 중인 Pod 안에 잠깐 들어가서 디버깅할 수 있게 해주는, 덧붙이는 컨테이너”**라고 보시면 됩니다.

혹시 원하시면, 지금 쓰고 계신 Minikube 환경에서 바로 ``kubect1 debug``로 테스트할 수 있는 간단 실습도 짜드릴까요?



‘ephemeral’이라는 영어 단어는 “잠시 존재하는, 덧없는, 일시적인”이라는 뜻이에요.

- 어원: 그리스어 *epi* (위에) + *hemera* (하루) → “하루살이 같은” 느낌
  - 비슷한 단어: *temporary* (임시의), *transient* (순간적인), *short-lived* (단명한)
- 

📌 그래서 쿠버네티스에서 **ephemeral container** 라는 말을 쓴 거예요.

“잠깐 붙였다가 사라지는 컨테이너”

딱 본래 뜻 그대로 쓰인 거죠.

원하시면 이 단어가 IT 말고도 일상에서 쓰이는 문장 예시도 보여드릴까요?



좋은 디테일 잡으셨네요 👍

‘sh -lc ‘...’`에서 ‘-l`하고 ‘-c`는 **sh(=shell)**의 옵션입니다.

---

‘-c`

- **command** 옵션.
  - 뒤에 오는 문자열(‘curl -I http://127.0.0.1:8080’)을 “실행할 명령어”로 해석합니다.
  - 없으면 그냥 sh 대화형 셸이 열려 버려요.
-

`-1`

- **login shell** 옵션.
- 마치 새로 로그인한 것처럼 셸을 띄워요.
- 이때 `/etc/profile`, `~/.profile` 같은 로그인용 초기화 스크립트를 읽습니다.
- 그래서 PATH, 환경 변수 등이 평소와 같게 세팅돼요.

## 합치면

```
sh -lc 'curl -I http://127.0.0.1:8080'
```

= “로그인 셸 환경을 불러온 다음, `curl -I http://127.0.0.1:8080` 명령을 실행해라”

즉, 환경 변수가 깨끗이 안 잡히는 컨테이너 안에서도 **PATH 문제 없이 원하는 명령만 실행**하게 하는 안전한 방식이에요.

혹시 원하시면 `-1`` 을 뺀 경우와 넣은 경우 환경변수가 어떻게 다른지 직접 비교하는 작은 실습 예제도 보여드릴까요?



그 손맛 알죠 😊 속도 붙을 때가 제일 재밌습니다.

지금 흐름 이어가 볼까요? 두 가지 중 하나 골라주세요:

1. 에페멀 컨테이너로 네트워크 진단 미니 실습 세트

## 2. `curl / ss(netstat) / kubectl debug / port-forward` 디버깅 치트시트 1페이지

원하시는 쪽으로 바로 만들어 드릴게요.



13s 동안 생각함

- 우리는 (1) 임시 컨테이너 네트워크 진단을 위한 미니랩과 (2) curl, ss, netstat, kubectl 디버깅/포트 포워딩에 대한 요약을 작성해야 해요. 사용자분은 구조화된, 단계별 접근을 선호하니 이에 맞춰 간결한 방식으로 작성할게요. PDF는 요청되지 않았지만 문서 생성은 유용할 수 있으니 필요시 고려하겠습니다.
- 우리는 필수 요소를 체크할 거예요: kubectl 디버그 지원, API 서버의 임시 컨테이너 기능, 임시 컨테이너 생성 권한(RBAC). 함께, nsenter 또는 사이드카를 통한 디버깅 대체 방법도 추가하겠습니다. 각 섹션에 명령어, 예상 출력, 문제 해결법, 클린업 방법 등을 제공할 예정이에요. 시스템 특성에 따라 적합한 도구도 제시할 거예요.
- 먼저, `label app=guestbook-ui`로 포드를 식별하고, `c-etc-guestbook-ui` 컨테이너를 확인합니다. 그런 후 임시 컨테이너 실행: `kubectl debug -it pod/$POD --image=curlimages/curl` 또는 셸 실행 `kubectl debug -it pod/$POD --image=nicolaka/netshoot -- sh`. 임시 컨테이너는 파일 시스템을 공유하지 않으므로 네트워크/IPC만 사용. Kubernetes 1.23 이상에서는 `kubectl debug` 명령어를 사용해야 해요.

🕒 완료

좋습니다. 바로 씁니다—실습 세트(에페멀 컨테이너) + 1페이지 디버깅 치트시트.

# 1) 에페멀 컨테이너로 네트워크 진단 - 미니 실습

“목표: 실행 중인 Pod 옆에 `curl`/도구가 들어있는 컨테이너를 잠깐 붙여서(에페멀) 포트/응답을 확인합니다.

전제: 앞서 정리한 대로 `app=guestbook-ui`, 컨테이너 포트는 8080을 예로 듭니다.”

## A. 준비 (Pod/컨테이너 확인)

# Pod 이름

```
kubectl get po -l app=guestbook-ui -o wide
POD=$(kubectl get po -l app=guestbook-ui -o
jsonpath='{.items[0].metadata.name}')
```

# 대상 컨테이너명(Deployment에서 지정한 name)

```
kubectl get po $POD -o jsonpath='{.spec.containers[*].name}{"\n"}'
```

# 예) c-etc-guestbook-ui

```
TARGET=c-etc-guestbook-ui
```

# 에페멀 컨테이너 기능이 설명 가능한지(참고용)

```
kubectl explain pod.spec.ephemeralContainers >/dev/null && echo "ephemeral OK"
```

## B. 가장 빠른 원샷 테스트 (응답 헤더만)

```
kubectl debug -it pod/$POD \
--image=curlimages/curl:8.8.0 \
--target=$TARGET -- \
sh -lc 'curl -I http://127.0.0.1:8080'
```

- 200/301/302 등 헤더가 보이면 컨테이너 내부에서 **8080** 리스 **OK**.
- `Connection refused`면 해당 포트에 리스닝 프로세스 없음(또는 127.0.0.1이 아닌 인터페이스에만 바인딩).

## C. 셸 열어서 여러 포트/도구 점검

# 도구 풍부한 이미지(조금 큼). 네임서버/라우팅/포트까지 종합 점검 가능

```
kubectl debug -it $POD --image=nicolaka/netshoot --target=$TARGET -- sh
```

# (셸 안) 흔한 체크들

```
ss -lntp 2>/dev/null || netstat -lntp
```

```
for p in 80 8080 8086 ; do
```

```
echo "=== curl 127.0.0.1:$p ==="
```



```
curl -sS -I --max-time 2 http://127.0.0.1:$p || true
done
ip addr ; ip route
nslookup guestbook-ui || dig guestbook-ui
exit
```

## D. 같은 Pod 네트워크에서 Service 경유 확인

```
kubectl debug -it $POD --image=curlimages/curl:8.8.0 --target=$TARGET -- \
sh -lc 'curl -I http://guestbook-ui:8086'
```

- OK면 **Service → Endpoints → Pod** 흐름이 정상.

## E. 정리(자동으로 사라지지만, 참고)

- 에페멀 컨테이너는 Pod 재시작 시 사라집니다. 별도 삭제 불필요.
- 에페멀 컨테이너가 붙지 않을 때:
  - RBAC/클러스터 정책으로 막힌 경우가 있습니다(회사망 환경). 그땐 아래 임시 파드 방식으로 대체.

## F. 대체: 임시 파드로 내부에서 때리기

```
kubectl run -it --rm curltest --image=curlimages/curl --restart=Never -- \
sh -lc 'curl -I http://guestbook-ui:8086'
```

# 2) 네트워크 디버깅 치트시트 (1페이지)

## A. 무엇을 볼 것인가

- **Connection refused**: 해당 포트에 프로세스가 **안** 듣는 경우(혹은 127.0.0.1 바인딩).
- **Connection timed out**: 경로/정책/서비스 연결(Endpoints) **막힘/없음** 가능성 큼.
- **503/404** 등: 애플리케이션/리버스 프록시 레벨 문제.

## B. 핵심 명령 10선

# 1) 파드/서비스 상태

```
kubectl get po -o wide
kubectl get svc -o wide
kubectl get endpoints guestbook-ui
```

# 2) 파드 로그

```
kubectl logs <pod> --tail=200
```

# 3) 컨테이너 내부 리슨 포트 확인 (있으면 ss, 없으면 netstat)

```
kubectl exec -it <pod> -- sh -lc '(ss -lntp || netstat -lntp) 2>/dev/null || true'
```

# 4) 에페멀 컨테이너로 내부 포트 직접 타격

```
kubectl debug -it pod/<pod> --image=curlimages/curl:8.8.0 --target=<container>
-- \
sh -lc 'curl -I http://127.0.0.1:<PORT>'
```

# 5) 임시 파드로 Service/PodIP 확인

```
kubectl run -it --rm curltest --image=curlimages/curl --restart=Never -- \
sh -lc 'curl -I http://guestbook-ui:<SERVICE_PORT>'
```

# 6) 포트포워딩 (로컬에서 바로 확인)

```
kubectl port-forward deploy/etc-guestbook-ui-deployment 8080:8080
# -> curl -I http://127.0.0.1:8080
```

# 7) Service ↔ Pod 연결(Endpoints) 빠르게 스캔

```
kubectl get endpoints guestbook-ui -o jsonpath='{.subsets[*].addresses[*].ip}{": "}{.subsets[*].ports[*].port}{"\n"}'
```

# 8) 라벨 셀렉터 불일치 탐지

```
kubectl get svc guestbook-ui -o yaml | yq '.spec.selector'
```

```
kubectl get po -l app=guestbook-ui # 셀렉터가 진짜 붙는지
```

# 9) minikube에서 외부 접근

```
minikube service guestbook-ui --url
```

```
# (LoadBalancer 타입이면) minikube tunnel
```

# 10) NodePort로 빠르게 외부 노출

```
kubectl patch svc guestbook-ui -p '{"spec":{"type":"NodePort"}}'
```

```
kubectl get svc guestbook-ui -o wide
```

```
# -> NODE-PORT 확인 후: curl -I http://$(minikube ip):<NODEPORT>
```

## C. 증상 ↔ 원인 매칭

- **PodIP에 80은 거절, 8080은 OK** → 앱이 8080 리스. Service `targetPort`를 8080으로.
- **Service 접속 실패 & Endpoints 비어있음** → 라벨 셀렉터 불일치(가장 흔함).
- **Service는 OK, PodIP 직접은 거절** → 앱이 `127.0.0.1` 바인딩. `0.0.0.0`로 바꾸기.
- 타임아웃 → Endpoints 없음, NetworkPolicy/방화벽, 잘못된 ClusterIP/네임해결 문제.
- 리디렉션만 계속(**301/302**) → Nginx/앱 리다이렉트 설정 확인(호스트/스킴).

## D. 빠른 체크 리스트

- ☐ 실제 리스 포트(컨테이너 내부) = `ss/netstat`, `curl 127.0.0.1:<port>`
- ☐ Service `targetPort` ↔ 실제 포트 일치
- ☐ Service `selector` ↔ Pod `labels` 일치
- ☐ Endpoints 채워짐 확인
- ☐ 필요 시 `port-forward` / `minikube service`로 외부에서 확인

- ☐ 앱 바인딩: `0.0.0.0` (로컬 전용이면 `127.0.0.1`)
- 

원하시면 위 두 묶음을 **PDF 한 장씩**으로도 내드릴 수 있어요. 지금 내용으로 충분하면 그대로 쓰시고, 더 깊게(예: Ingress/NetworkPolicy까지) 확장판도 만들 수 있습니다.