Cert-Manager i Let's Encrypt na Kubernetes

mgr inż. Jakub Woźniak

1 Wprowadzenie

Niniejsze laboratorium obejmuje praktyczne wdrożenie cert-managera – narzędzia do zarządzania certyfikatami TLS w Kubernetes. Uczestnicy nauczą się konfigurować automatyczne wystawianie i odnowienie certyfikatów ze środowiska Let's Encrypt, zarówno staging jak i production.

1.1 Wymagania wstępne

- Minikube
- kubectl skonfigurowany
- Domena w postaci `lab-sec-<nr>.cs.put.poznan.pl`
- Podstawowa wiedza o Kubernetes

1.2 Zmienne laboratorium

Dostosuj poniższe zmienne do swoich danych:

```
COMPUTER_NUMBER=1                    # zmień na swój numer
DOMAIN="lab-sec-${COMPUTER_NUMBER}.cs.put.poznan.pl"
MINIKUBE_IP="192.168.1.100"          # IP maszyny wirtualnej z minikubem
INGRESS_PORT=30443                   # port NodePort dla HTTPS
INGRESS_PORT_HTTP=30080              # port NodePort dla HTTP
EMAIL="infXXXXX@student.put.poznan.pl"     # twój email do Let's Encrypt
```

2 Podstawy cert-managera i Let's Encrypt

2.1 Co to jest cert-manager?

Cert-manager to kontroler Kubernetes, który automatyzuje zarządzanie certyfikatami X.509 w klastrach. Działuje poprzez:

1. **Obserwowanie** zasobów Ingress i Certificate
2. **Komunikowanie się** z Autorytami Certyfikacyjnymi (CA) poprzez protokół ACME
3. **Przechowywanie** wystawionych certyfikatów w Kubernetes Secrets
4. **Automatyczne odnowienie** certyfikatów przed ich wygaśnięciem

2.2 Architektura cert-managera

Cert-manager można logicznie podzielić na kilka elementów:

- **Zasoby użytkownika w klastrze**
  - `Ingress` – opisuje ruch HTTP/HTTPS do aplikacji.
  - `Certificate` – żądanie wystawienia konkretnego certyfikatu.
  - `Issuer` / `ClusterIssuer` – konfiguracja, jak połączyć się z daną CA (np. Let's Encrypt).
- **Kontroler cert-managera**
  - Deployment w namespace `cert-manager`.
  - Obserwuje zasoby `Certificate` / `Ingress` i tworzy obiekty ACME (Order, Challenge).
  - Tworzy obiekty pomocnicze (pody solvera, tymczasowe Ingressy).
- **Sekrety z kluczami i certyfikatami**
  - `Secret` z typem `kubernetes.io/tls` zawiera klucz prywatny i certyfikat.
  - Nazwa Secreta jest podana w `spec.secretName` zasobu `Certificate`.
- **Zewnętrzna CA (Let's Encrypt)**
  - Serwer ACME (staging lub production).
  - Weryfikuje własność domeny i wystawia certyfikat.

Dzięki temu użytkownik pracuje tylko na standardowych obiektach Kubernetes (`Ingress`, `Certificate`), a cała złożoność komunikacji z CA i odnowień jest ukryta w kontrolerze cert-managera.

2.3 Zasób Certificate

Zasób `Certificate` definiuje, jaki certyfikat powinien być wystawiony i od którego wystawcy.

```yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-cert
  namespace: default
spec:
  secretName: example-cert-tls      # gdzie przechować certyfikat
  duration: 2160h                   # czas ważności (90 dni)
  renewBefore: 360h                 # odnów 15 dni przed wygaśnięciem
  commonName: example.com
  dnsNames:
    - example.com
    - "*.example.com"
  issuerRef:
    name: letsencrypt-staging       # wybór wystawcy
    kind: ClusterIssuer
    group: cert-manager.io
```

2.4 Zasoby Issuer i ClusterIssuer

Issuer – zasób ograniczony do namespace'u (tylko certyfikaty w tym namespace) **ClusterIssuer** – dostępny dla wszystkich namespace'ów w klastrze

```yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
```

```
  name: letsencrypt-staging
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    email: admin@example.com
    privateKeySecretRef:
      name: letsencrypt-staging-key
    solvers:
      - http01:
          ingress:
            class: nginx
```

2.5 Podstawowe pojęcia TLS i certyfikatów

- **Certyfikat X.509** – zawiera klucz publiczny, nazwę podmiotu (CN, SAN), okres ważności oraz informację o wystawcy (Issuer). Jest podpisany przez zaufaną jednostkę certyfikującą (CA).
- **Klucz prywatny** – trzymany w klastrze w `Secret` razem z certyfikatem; nie powinien opuszczać klastra ani być logowany.
- **CN vs SAN** – w nowoczesnych przeglądarkach najważniejsze są pola `subjectAltName` (`dnsNames` w zasobie `Certificate`), a nie `commonName`.
- **Łańcuch zaufania** – przeglądarka ufa certyfikatowi końcowemu, jeśli potrafi zbudować łańcuch do którejś z zaufanych CA w swoim magazynie.

2.6 Typy wyzwań ACME

Najczęściej wykorzystywane przez cert-managera typy challenge:

- **HTTP-01** – udowadniasz kontrolę nad domeną poprzez odpowiedź HTTP z określonego URL-a. Wymaga publicznie dostępnego endpointu HTTP.
- **DNS-01** – udowadniasz kontrolę nad domeną poprzez dodanie rekordu TXT w DNS. Pozwala wystawiać certyfikaty wildcard (`*.example.com`), ale wymaga automatycznej integracji z systemem DNS.
- **TLS-ALPN-01** – weryfikacja odbywa się na poziomie ALPN/TLS; używane rzadziej w typowych wdrożeniach.

W tym laboratorium korzystamy z HTTP-01, ponieważ:

- dobrze współpracuje z Ingress + NGINX,
- nie wymaga zmian w konfiguracji zewnętrznego DNS,
- jest łatwy do debugowania (widzisz zwykłe zapytania HTTP).

2.7 Proces wystawiania certyfikatu (ACME HTTP-01)

Poniżej krok po kroku, co dzieje się w typowym scenariuszu z HTTP-01:

1. Użytkownik tworzy `Ingress` z adnotacją `cert-manager.io/cluster-issuer: letsencrypt-staging` (lub `letsencrypt-production`).
2. Cert-manager wykrywa nowy `Ingress` i generuje odpowiadający mu zasób `Certificate`.
3. Cert-manager nawiązuje połączenie z serwerem ACME Let's Encrypt:
   - zakłada (lub używa istniejącego) konto ACME,
   - tworzy `Order` – żądanie wystawienia certyfikatu dla danej domeny.
4. Let's Encrypt odpowiada zestawem `Challenge` typu HTTP-01 – prosi o udowodnienie, że kontrolujesz domenę.
5. Cert-manager tworzy tymczasowy **solver**:
   - tymczasowy `Pod` z prostym serwerem HTTP,
   - `Service` i/lub `Ingress`, który wystawia adres `http://<domena>/.well-known/acme-challenge/<token>`.
6. Serwer Let's Encrypt wykonuje zapytanie HTTP GET do powyższego URL-a i oczekuje poprawnego tokena.
7. Jeżeli weryfikacja się powiedzie, Let's Encrypt generuje certyfikat X.509.
8. Cert-manager pobiera certyfikat i zapisuje go w `Secret` wskazanym w `spec.secretName` zasobu `Certificate`.
9. Kontroler Ingress (np. NGINX) wykrywa zmianę Secreta i zaczyna używać nowego certyfikatu TLS.
10. Po zakończeniu procesu cert-manager usuwa tymczasowe pody solvera oraz obiekty pomocnicze.

11. Cert-manager śledzi datę wygaśnięcia certyfikatu i odpowiednio wcześnie (np. 15 dni przed końcem ważności) ponawia cały proces, automatycznie odnawiając certyfikat.

W rezultacie jedyną rzeczą, którą musisz utrzymywać ręcznie, jest definicja `Ingress` / `Certificate`. Resztę (komunikacja z CA, weryfikacja domeny, tworzenie i odnowienie certyfikatów) wykonuje cert-manager.

2.8 Let's Encrypt: Staging vs Production

2.8.1 Środowisko testowe (staging)

- **URL ACME**: `https://acme-staging-v02.api.letsencrypt.org/directory`
- **Certyfikat wydawany przez**: „(STAGING) Pretend Pear X1" (NIEZAUFANY)
- **Rate limit**: Bardzo hojny (brak praktycznych limitów)
- **Ważność**: 90 dni (jak production)
- **Przypadek użytku**: Testowanie, debug, nauka
- **Przeglądarka**: Wyświetla ostrzeżenie o niezaufanym certyfikacie

```
OSTRZEŻENIE W PRZEGLĄDARCE:
"Twoje połączenie nie jest prywatne"
"(STAGING) Pretend Pear X1"
```

2.8.2 Środowisko produkcyjne

- **URL ACME**: `https://acme-v02.api.letsencrypt.org/directory`
- **Certyfikat wydawany przez**: „Let's Encrypt (trusted)"
- **Rate limit**: 50 certyfikatów na domenę co 7 dni
- **Ważność**: 90 dni
- **Przypadek użytku**: Produkcja, publiczne strony
- **Przeglądarka**: Zielona kłódka, certyfikat zaufany

```
PRAWIDŁOWY CERTYFIKAT:
Wystawiony przez: Let's Encrypt
Ważny dla: lab-sec-01.cs.put.poznan.pl
Status: Certyfikat jest ważny
```

2.9 Ingress z adnotacjami cert-manager

cert-manager obserwuje adnotacje na zasobach Ingress:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-staging"
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - example.com
      secretName: example-tls
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: example-service
                port:
                  number: 80
```

Co robi cert-manager:

1. Widzi adnotację `cert-manager.io/cluster-issuer: letsencrypt-staging`

2. Automatycznie tworzy `Certificate` (nie musisz ręcznie)
3. Wystawia certyfikat
4. Przechowuje w Secret `example-tls`
5. Ingress automatycznie konfiguruję TLS

3 Przygotowanie środowiska

3.1 Krok 1: Uruchomienie Minikube

Korzystając z wiedzy z poprzednich laboratoriów, uruchom `minikube` i skonfiguruj środowisko (`kubectl`, `helm`).

3.2 Krok 2: Instalacja cert-manager

Istnieje wiele sposobów instalacji. Użyjemy Helm (najczystszy):

```
# Dodaj repozytorium Helm
helm repo add jetstack https://charts.jetstack.io
helm repo update

# Utwórz namespace dla cert-manager
kubectl create namespace cert-manager

# Instalacja cert-manager
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --version v1.14.0 \
  --set installCRDs=true

# Czekaj, aż wszystkie pody będą gotowe
kubectl wait --for=condition=ready pod \
  -l app.kubernetes.io/instance=cert-manager \
  -n cert-manager \
  --timeout=300s

# Sprawdzenie
kubectl get pods -n cert-manager
```

Oczekiwany output:

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-569cc87c54-8xvvx              1/1     Running   0          2m
cert-manager-cainjector-5886f7fcdb-wt7vf   1/1     Running   0          2m
cert-manager-webhook-697c6cdd57-tcpb5      1/1     Running   0          2m
```

3.3 Krok 3: Tworzenie ClusterIssuers

3.3.1 ClusterIssuer dla Staging

```
cat <<'EOF' | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    # Let's Encrypt staging server (certyfikaty niezaufane)
    server: https://acme-staging-v02.api.letsencrypt.org/directory

    # Email do wiadomości ważnych od Let's Encrypt
    email: student@cs.put.poznan.pl

    # Secret przechowujący klucz prywatny konta ACME
```

```
    privateKeySecretRef:
      name: letsencrypt-staging-key

    # Solver dla HTTP-01 challenge
    solvers:
      - http01:
          ingress:
            class: nginx
EOF

# Sprawdzenie
kubectl describe clusterissuer letsencrypt-staging
```

3.3.2 ClusterIssuer dla Production

```
cat <<'EOF' | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-production
spec:
  acme:
    # Production server - ZAUFANE certyfikaty
    server: https://acme-v02.api.letsencrypt.org/directory

    email: student@cs.put.poznan.pl

    privateKeySecretRef:
      name: letsencrypt-production-key

    solvers:
      - http01:
          ingress:
            class: nginx
EOF

# Sprawdzenie
kubectl describe clusterissuer letsencrypt-production
```

3.4 Krok 4: Konfiguracja NGINX Ingress Controller

NGINX Ingress Controller już powinien być zainstalowany jako addon, ale musimy go skonfigurować dla NodePort:

```
# Sprawdzenie ingress controllera
kubectl get pods -n ingress-nginx

# Zmiana Service typu na NodePort
kubectl patch svc ingress-nginx-controller -n ingress-nginx \
  -p '{"spec":{"type":"NodePort"}}'

# Sprawdzenie portów
kubectl get svc -n ingress-nginx
```

Oczekiwany output:

```
NAME                                 TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
ingress-nginx-controller             NodePort   10.96.182.14    <pending>
80:30080/TCP,443:30443/TCP
```

3.5 Udostępnienie Ingress Minikube na portach 80 i 443 (VirtualBox)

Przy uruchamianiu Minikube z VirtualBox, kontroler Ingress może nasłuchiwać na wysokim porcie (np. 30213). Aby udostępnić go na zewnątrz, przekierujemy porty 80 i 443 z hosta do maszyny wirtualnej Minikube.

3.5.1 Sprawdź adres IP Minikube

```
minikube ip
```

Przykładowy wynik:

```
192.168.59.101
```

To na ten adres przekierujemy porty.

3.6 Przekierowanie portów przy użyciu iptables

Możesz przekierować ruch bezpośrednio na hoście Linux, bez konfiguracji w VirtualBox:
Najpierw pobierz IP Minikube:

```
MINIKUBE_IP=$(minikube ip)
echo $MINIKUBE_IP
```

Następnie dodaj reguły iptables:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination
$MINIKUBE_IP:30213
sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j DNAT --to-destination
$MINIKUBE_IP:30213

sudo iptables -t nat -A POSTROUTING -p tcp -d $MINIKUBE_IP --dport 30213 -j
MASQUERADE
```

Reguły przekierowują ruch z portów 80/443 hosta do Minikube.

4 Ćwiczenia praktyczne

4.1 Zadanie 1: Testowy Deployment z Certyfikatem Staging

4.1.1 1.1 Utwórz aplikację testową (whoami)

```
cat <<'EOF' | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      containers:
      - name: whoami
        image: traefik/whoami:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
```

```
metadata:
  name: whoami-service
  namespace: default
spec:
  selector:
    app: whoami
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
EOF

# Sprawdzenie
kubectl get pods -l app=whoami
kubectl get svc whoami-service
```

4.1.2 1.2 Utwórz Ingress ze Staging certyfikatem

```
DOMAIN="lab-sec-01.cs.put.poznan.pl"

cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: whoami-ingress
  namespace: default
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-staging"
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - $DOMAIN
      secretName: whoami-staging-tls
  rules:
    - host: $DOMAIN
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: whoami-service
                port:
                  number: 80
EOF

# Sprawdzenie
kubectl describe ingress whoami-ingress
```

4.1.3 1.3 Obserwuj proces wystawiania certyfikatu

```
# W jednym terminalu obserwuj zdarzenia
kubectl get events -w

# W drugim terminalu sprawdzaj status Certificate
kubectl get certificate -w

# Powinno się pojawić:
```

```
# NAME                    READY   SECRET              AGE
# whoami-staging-tls      False   whoami-staging-tls  0s
# (czekaj...)
# whoami-staging-tls      True    whoami-staging-tls  45s
```

4.1.4 1.4 Sprawdzenie certyfikatu

```
# Wylistuj Secret z certyfikatem
kubectl get secrets

# Eksportuj certyfikat
kubectl get secret whoami-staging-tls \
  -o jsonpath='{.data.tls\.crt}' | base64 -d > /tmp/cert.pem

# Sprawdź szczegóły certyfikatu
openssl x509 -in /tmp/cert.pem -text -noout

# Poszukaj linii:
# Issuer: C=US, O=(STAGING) Let's Encrypt, CN=(STAGING) Pretend Pear X1
# Subject: CN=lab-sec-01.cs.put.poznan.pl
```

4.2 Zadanie 2: Przesunięcie na Production

4.2.1 Zmiana adnotacji Ingress'u

```
DOMAIN="lab-sec-01.cs.put.poznan.pl"

cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: whoami-ingress
  namespace: default
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-production"
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - $DOMAIN
      secretName: whoami-production-tls
  rules:
    - host: $DOMAIN
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: whoami-service
                port:
                  number: 80
EOF
```

4.2.2 2.2 Obserwuj proces

```
# Czekaj aż nowy Certificate będzie READY
kubectl get certificate -w
```

```
# Powinna pojawić się nowa certyfikacja
# whoami-production-tls   True    whoami-production-tls   <kilka minut>
```

4.2.3 2.3 Sprawdzenie produkcyjnego certyfikatu

```
# Eksportuj nowy certyfikat
kubectl get secret whoami-production-tls \
  -o jsonpath='{.data.tls\.crt}' | base64 -d > /tmp/cert-prod.pem

# Sprawdź szczegóły
openssl x509 -in /tmp/cert-prod.pem -text -noout

# Poszukaj:
# Issuer: C=US, O=Let's Encrypt, CN=R3
# Subject: CN=lab-sec-01.cs.put.poznan.pl
```

4.3 Zadanie 3: Deployment Redmine z certyfikatem

4.3.1 3.1 Utwórz namespace dla Redmine

```
kubectl create namespace redmine
kubectl config set-context --current --namespace=redmine
```

4.3.2 3.2 Utwórz ConfigMap dla bazy danych

```
cat <<'EOF' | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: redmine-config
  namespace: redmine
data:
  RAILS_ENV: "production"
  DATABASE_ADAPTER: "sqlite3"
EOF
```

4.3.3 3.3 Deployment Redmine

```
DOMAIN="lab-sec-01.cs.put.poznan.pl"

cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redmine
  namespace: redmine
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redmine
  template:
    metadata:
      labels:
        app: redmine
    spec:
      containers:
      - name: redmine
        image: redmine:latest
        ports:
```

```
        - containerPort: 3000
        env:
        - name: REDMINE_DB_MYSQL
          value: "localhost"
        - name: REDMINE_DB_PORT
          value: "3306"
        - name: REDMINE_RELATIVE_URL_ROOT
          value: ""
        - name: RAILS_ENV
          value: "production"
        volumeMounts:
        - name: redmine-data
          mountPath: /usr/src/redmine/files
      volumes:
      - name: redmine-data
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: redmine-service
  namespace: redmine
spec:
  selector:
    app: redmine
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: ClusterIP
EOF

# Czekaj, aż pod będzie Running
kubectl wait --for=condition=ready pod \
  -l app=redmine \
  -n redmine \
  --timeout=300s
```

4.3.4 3.4 Ingress dla Redmine z produkcyjnym certyfikatem

```
DOMAIN="lab-sec-01.cs.put.poznan.pl"

cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: redmine-ingress
  namespace: redmine
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-production"
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - $DOMAIN
      secretName: redmine-tls
  rules:
    - host: $DOMAIN
      http:
```

```
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: redmine-service
                port:
                  number: 80
EOF

# Sprawdzenie
kubectl describe ingress redmine-ingress
```

4.3.5 3.5 Monitoring procesu wystawiania certyfikatu

```
# Obserwuj Certificate
watch kubectl get certificate -n redmine

# Obserwuj zdarzenia
kubectl get events -n redmine -w

# Sprawdzaj pody cert-managera
kubectl get pods -n cert-manager -l app=cert-manager
```

Proces powinien zająć 1-5 minut:

1. cert-manager widzi Ingress i tworzy Certificate
2. Certificate przechodzi w stan „False” (oczekiwanie na weryfikację)
3. cert-manager tworzy challenge solver
4. Let’s Encrypt weryfikuje domenę
5. Certyfikat zostaje wystawiony
6. Certificate przechodzi w stan „True”
7. Secret redmine-tls zawiera certyfikat

4.3.6 Weryfikacja

```
# Sprawdzenie certyfikatu
kubectl get secret redmine-tls -n redmine \
  -o jsonpath='{.data.tls\.crt}' | base64 -d > /tmp/redmine-cert.pem

openssl x509 -in /tmp/redmine-cert.pem -text -noout

# Test HTTPS
DOMAIN="lab-sec-1.cs.put.poznan.pl"
curl https://$DOMAIN:443/

# W przeglądarce (na hoście):
# https://lab-sec-01.cs.put.poznan.pl:30443/
#
# Powinieneś zobaczyć:
# - Zieloną kłódkę (certyfikat zaufany)
# - Redmine dashboard
# - W szczegółach certyfikatu: Let's Encrypt, ważny certyfikat
```

5 Rozwiązywanie problemów

5.1 Sprawdzenie statusu Certificate

```
# Szczegółowe informacje
kubectl describe certificate redmine-tls -n redmine
```

```
# Poszukaj sekcji Status:
# Status:
#   Conditions:
#   - Last Transition Time: 2024-11-24T12:34:56Z
#     Message: Certificate is up to date and has not expired
#     Observed Generation: 1
#     Reason: Ready
#     Status: True
#     Type: Ready
#
#   Renewal Time: 2025-02-22T12:34:56Z
```

5.2 Logi cert-managera

```
# Główny kontroler
kubectl logs -n cert-manager deployment/cert-manager -f

# Webhook
kubectl logs -n cert-manager deployment/cert-manager-webhook -f

# CA Injector
kubectl logs -n cert-manager deployment/cert-manager-cainjector -f

# Poszukaj błędów dotyczących domeny, np:
# "error": "dns :: query timed out"
# "error": "urn:ietf:params:acme:error:connection"
```

5.3 Sprawdzenie Challenge

```
# Listuj wszystkie Challenge'i
kubectl get challenges -A

# Szczegóły Challenge'a (może być w innym namespace)
kubectl describe challenge <name> -n <namespace>

# Challenge powinien przejść przez stany:
# Status:
#   Processing: False
#   Reason: Successfully validated
#   State: valid
```

5.4 Testowanie konfiguracji DNS

```
# Z wewnątrz minikube'a
minikube ssh

# Wewnątrz VM
nslookup lab-sec-01.cs.put.poznan.pl

# Powinna zwrócić IP maszyny hosta
```

5.5 Weryfikacja komunikacji z Let's Encrypt

```
# Sprawdzenie, czy solver pod został utworzony
kubectl get pods -n cert-manager -l app.kubernetes.io/component=http01-solver

# Pody powinny być przez kilka minut, a następnie się usunąć
```

```
# Jeśli nie się pojawił:
# - Sprawdź, że cert-manager controller działa
# - Sprawdź logi: kubectl logs -n cert-manager deployment/cert-manager
```

6 Czyszczenie środowiska

```
# Usunięcie Redmine
kubectl delete namespace redmine

# Usunięcie whoami
kubectl delete ingress whoami-ingress
kubectl delete svc whoami-service
kubectl delete deployment whoami

# Usunięcie cert-managera (opcjonalnie)
helm uninstall cert-manager -n cert-manager
kubectl delete namespace cert-manager

# Zatrzymanie minikube
minikube tunnel  # Ctrl+C w terminalu tunelu
minikube stop
```

7 Podsumowanie i wnioski

7.1 Zdobyta wiedza

1. **Architektura cert-managera**: Jak kontroler obserwuje zasoby i komunikuje się z CA
2. **ACME HTTP-01 Challenge**: Proces weryfikacji domeny przez Let's Encrypt
3. **Staging vs Production**: Kiedy używać każdego środowiska
4. **Automatyzacja**: Certyfikaty są wystawiane i odnawiane bez ręcznej interwencji
5. **Ingress Annotations**: Jak wyzwalać cert-manager przez adnotacje
6. **Port Forwarding**: Konfiguracja dostępu do klastra w VM

7.2 Dobre praktyki

- **Zawsze zacznij od Staging**: Rate limit'y są hojne, mogą Ci oszczędzić problemy
- **Monitoruj Certificate**: Patrz na status, zdarzenia i logi
- **Nie usuwaj Secret'ów**: cert-manager ich odnawiał, jeśli będą potrzebne
- **Email do Let's Encrypt**: Podaj realistyczny email – potrzebny do ważnych wiadomości
- **Tunel vs Ingress**: Na produkcji LoadBalancer, lokalnie NodePort
- **Certyfikaty SSL/TLS**: Zawsze weryfikuj, że certyfikat jest od ufanego CA

7.3 Komendy do pamiętania

```
# Monitoring
kubectl get certificate -A
kubectl describe certificate <name> -n <namespace>
kubectl get events -n <namespace> -w

# Debugowanie
kubectl logs -n cert-manager deployment/cert-manager -f
kubectl get challenges -A
kubectl describe challenge <name> -n <namespace>

# Certyfikaty
openssl x509 -in cert.pem -text -noout
kubectl get secret <name> -o jsonpath='{.data.tls\.crt}' \
  | base64 -d | openssl x509 -text
```

8 Tabela problemów i rozwiązań

Problem	Objawy	Rozwiązanie
Certificate nie przechodzi w stan Ready	Certificate.Ready = False	Sprawdzić logi cert-managera, zdarzenia Challenge'a
„propagation check failed"	Solver pod nie odpowiada	Sprawdzić czy ingress controller działa, firewall DNS
„rate limit exceeded"	Certyfikat się nie wystawia	Przejść na Staging, czekać aż rate limit reset
Secret nie ma certyfikatu	`kubectl get secret` pokazuje inny typ	Czekać, certyfikat może być jeszcze wystawiany
Ingress IP pending	NodePort Service wskazuje `<pending>`	Sprawdzić czy ingress addon jest włączony w minikube
Certyfikat staging w production	OpenSSL pokazuje STAGING	Zmienić issuer, usunąć stary Secret, ponownie tworzyć Certificate
Let's Encrypt DNS timeout	Challenge nie weryfikuje się	Sprawdzić że DNS z hosta widzi domenę, tunel Minikube działa