# cert-manager für interne PKI konfigurieren

Um cert-manager in Kubernetes so zu konfigurieren, dass er eine eigene, interne Public Key Infrastructure (PKI) aufbaut und verwaltet, müssen Sie im Wesentlichen zwei Schritte durchführen:

1. **Schritt 1:** Einen selbstsignierten Root-CA-Zertifikatsaussteller (Issuer) erstellen: Dieser Issuer generiert ein Root-Zertifikat und den dazugehörigen privaten Schlüssel. Dieses Zertifikat bildet die Vertrauensbasis Ihrer internen PKI.
2. **Schritt 2:** Einen CA-Issuer erstellen, der das Root-Zertifikat verwendet: Dieser zweite Issuer nutzt das in Schritt 1 erstellte Root-Zertifikat, um End-Entity-Zertifikate (z. B. für Ihre Webserver oder Dienste) zu signieren.

Hier sind die Details und Beispiel-YAML-Konfigurationen:

### Schritt 1: Root-CA erstellen (mit einem SelfSigned Issuer)

Zuerst definieren Sie ein **Certificate**-Objekt, das als Ihre Root-CA fungieren soll. Dieses Zertifikat wird von einem temporären **SelfSigned** Issuer ausgestellt. Das resultierende Zertifikat und der Schlüssel werden in einem Kubernetes-**Secret** gespeichert.

```yaml
apiVersion: cert-manager.io/v1
kind: Issuer # Oder ClusterIssuer für clusterweiten Geltungsbereich
metadata:
  name: selfsigned-issuer # Name des temporären Issuers
  namespace: cert-manager # Oder wo immer Sie es platzieren möchten
spec:
  selfSigned: {} # Definiert diesen als SelfSigned Issuer
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-internal-ca # Name für Ihr Root-CA-Zertifikat
  namespace: cert-manager # Gleicher Namespace wie der Issuer oben
spec:
  # Das Secret, in dem das CA-Zertifikat und der Schlüssel gespeichert
werden
  secretName: root-ca-secret
  # Dauer der Gültigkeit des CA-Zertifikats (z.B. 10 Jahre)
  duration: 87600h # 10 * 365 * 24h
  renewBefore: 720h # Erneuern 30 Tage vor Ablauf (optional für Root-CA)

  # WICHTIG: Markiert dieses Zertifikat als Certificate Authority
  isCA: true

  # Identifizierende Informationen für Ihre CA
  commonName: "My Internal Root CA"
  subject:
    organizations:
```

```
      - "My Company"

  # Referenz zum SelfSigned Issuer, der dieses CA-Zertifikat erstellt
  issuerRef:
    name: selfsigned-issuer # Name des oben definierten Issuers
    kind: Issuer # Muss dem Kind des Issuers entsprechen (Issuer oder
ClusterIssuer)
    group: cert-manager.io
```

**Wichtige Punkte zu Schritt 1:**

- `isCA: true`: Dies ist entscheidend, um das Zertifikat als CA zu kennzeichnen.
- `secretName: root-ca-secret`: Merken Sie sich diesen Namen. Er enthält das Herzstück Ihrer PKI.
- `issuerRef`: Verweist auf den `selfsigned-issuer`, der nur dazu dient, dieses eine CA-Zertifikat zu erstellen.
- `Issuer` vs. `ClusterIssuer`: Wenn die Root-CA und der spätere CA-Issuer clusterweit (in allen Namespaces) verfügbar sein sollen, verwenden Sie `ClusterIssuer` statt `Issuer` und entfernen Sie die `namespace`-Angabe aus den `metadata`-Abschnitten der Issuer/ClusterIssuer. Das Secret (`root-ca-secret`) muss dann in einem Namespace existieren, auf den der cert-manager Zugriff hat (oft der `cert-manager`-Namespace selbst).

**Schritt 2: CA-Issuer erstellen, der das Root-Zertifikat verwendet**

Nun erstellen Sie den eigentlichen Issuer (oder ClusterIssuer), den Ihre Anwendungen verwenden werden, um Zertifikate anzufordern. Dieser Issuer verwendet das in `root-ca-secret` gespeicherte CA-Zertifikat und den Schlüssel zum Signieren neuer Zertifikate.

```
apiVersion: cert-manager.io/v1
kind: Issuer # Oder ClusterIssuer für clusterweiten Geltungsbereich
metadata:
  name: internal-ca-issuer # Name des Issuers, der Ihre CA nutzt
  namespace: cert-manager # Oder ein anderer Namespace, wenn gewünscht
spec:
  ca:
    # Referenz zum Secret, das das Root-CA-Zertifikat und den Schlüssel
enthält
    secretName: root-ca-secret
```

**Wichtige Punkte zu Schritt 2:**

- **kind: Issuer** / **ClusterIssuer**: Entscheiden Sie wieder basierend auf dem gewünschten Geltungsbereich. Wenn Sie **ClusterIssuer** verwenden, können Anwendungen in *jedem* Namespace diesen Issuer referenzieren, um Zertifikate zu erhalten.
- **spec.ca.secretName: root-ca-secret**: Dies verknüpft den Issuer direkt mit dem in Schritt 1 erstellten CA-Geheimnis.

### Zertifikate mit dem neuen CA-Issuer anfordern

Nachdem Sie diese beiden Schritte durchgeführt haben, können Sie nun **Certificate**-Ressourcen erstellen, die auf Ihren `internal-ca-issuer` verweisen:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-service-tls
  namespace: my-application # Der Namespace Ihrer Anwendung
spec:
  secretName: my-service-tls-secret # Secret für das Zertifikat des Dienstes
  duration: 2160h # 90 Tage
  renewBefore: 360h # 15 Tage
  dnsNames:
  - my-service.my-application.svc
  - my-service.example.com # Falls extern erreichbar
  # etc.
  issuerRef:
    name: internal-ca-issuer # Verweist auf den in Schritt 2 erstellten
Issuer
    kind: Issuer # Oder ClusterIssuer, je nachdem, was Sie in Schritt 2
erstellt haben
    group: cert-manager.io
```

### Zusammenfassung

1. Erstellen Sie einen `SelfSigned` Issuer (oder ClusterIssuer).
2. Erstellen Sie ein **Certificate**-Objekt mit `isCA: true`, das vom `SelfSigned` Issuer ausgestellt wird und dessen Ergebnis in einem Secret (`root-ca-secret`) gespeichert wird.
3. Erstellen Sie einen **CA** Issuer (oder ClusterIssuer), der auf `root-ca-secret` verweist (`spec.ca.secretName`).
4. Verwenden Sie diesen **CA** Issuer in den `issuerRef`-Abschnitten Ihrer Anwendungs-**Certificate**-Ressourcen.

### Wichtiger Hinweis: Vertrauen der CA

Damit haben Sie eine interne PKI mit cert-manager aufgebaut. Beachten Sie, dass Clients (wie Browser oder andere Dienste), die auf mit dieser internen CA signierte Zertifikate zugreifen, dieser CA vertrauen müssen. Das bedeutet in der Regel, dass Sie **das öffentliche Zertifikat der Root-CA** (`ca.crt` aus dem `root-ca-secret`) **in den Trust Store der jeweiligen Clients importieren müssen**.