Bundesverwaltungsamt

Sicherer Anschlussknoten des Data Providers Integrationsanleitung

Bundesverwaltungsamt

Version 1.0.2, Datum 17.02.2026

Inhaltsverzeichnis

Änderungshistorie

Version	Änderungen
0.1	• Initiale Erstellung des Dokuments
0.2	• Einarbeitung Review Kommentare
0.9	• Dokument finalisiert
1.0	• Dokument abgenommen

1.0.1	• Anpassungen in der Beispiel-Konfigurationsdatei
	  • Hinzufügen von Konfigurationsparametern für das Rate-Limiting der Anwendung
	  • Hinzufügen von Konfigurationsparametern für einen Truststore mit erlaubten Client-Zertifikaten ohne Zertifikatskette (Pinning)
	• Anpassung von neuen Links und Referenzen auf OpenCode

1.0.2	• Hinzufügen der Anleitung für Installation in einem Kubernetes-Cluster inklusive der zugehörigen Konfigurationsparameter

1. Einleitung

Das vorliegende Dokument enthält die Integrationsanleitung für die Anwendung "**Sicherer Anschlussknoten des Data Providers**" (*SAK-DP*) in der jeweils aktuell veröffentlichten Version. Es fungiert als Hilfestellung für den Betrieb der Anwendung.

Das Kapitel Überblick bietet einen groben fachlichen, technischen sowie betrieblichen Überblick über die Anwendung und liefert Verweise auf weitergehende Dokumente.

Das Kapitel Installation der Anwendung für Server-Betrieb beschreibt die traditionelle Installation und Konfiguration der Anwendung in einer Server-Umgebung.
`Relevant für Betrieb und Softwareentwicklung`

Das Kapitel Installation in einem Kubernetes-Cluster beschreibt die Installation und Konfiguration der Anwendung in einem Kubernetes-Cluster.
`Relevant für Betrieb und Softwareentwicklung`

Das Kapitel Regulärer Wirkbetrieb liefert Informationen für den Betrieb der Anwendung und gibt Hilfestellung bei der Fehleranalyse
`Relevant für Betrieb und Softwareentwicklung`

Der Anhang enthält ein fachliches und technisches Glossar sowie die

Beschreibung der Konfigurationsdaten.

2. Überblick

In diesem Kapitel wird aus fachlicher und technischer Sicht ein Überblick über die Geschäftsprozesse gegeben, die mithilfe des SAK-DP durchgeführt werden.

2.1. Fachliche Beschreibung

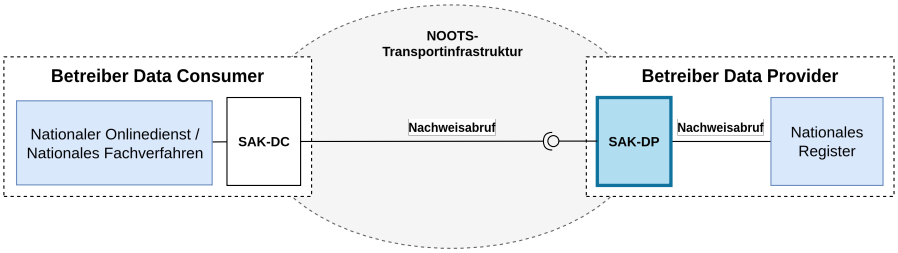

Abbildung 1. SAK-DP: Fachliche Kontextabgrenzung

Die fachlichen Aufgaben von SAK-DP sind:

- Prüfung von Identität und Berechtigung der abfragenden Stelle
- Prüfung und Weitergabe der Abruf-Anfrage an den Data Provider
- Weiterleitung des angefragten Nachweises an die anfragende Stelle

2.2. Technische Beschreibung

SAK-DP stellt Komponenten des NOOTS-Netzwerks einen sicheren Zugang zu einem Data Provider bereit. Hierzu fungiert die Anwendung als "Application Layer Gateway" durch:

- Bereitstellung einer öffentlichen REST-API (Transportprotokoll) für andere SAK-DC im Internet.
- Anbindung des Data Provider via das Anschlussprotokoll.

Zur Absicherung der Zugriffe auf Data Providers via das Anschlussprotokoll kommen "Basic Auth" und "OAuth 2.0" zum Einsatz, zur Absicherung der Netzwerk-Kommunikation "(m)TLS".

Die hierzu benötigten Zugangsdaten und Zertifikate müssen bei der Installation vom SAK-DP bereitgestellt werden.

HINWEIS

- Die Kommunikation des SAK-DP erfolgt direkt mit dessen Data Provider und den SAKs der Dara Consumer.
- Der europäische Datenfluss ist nicht Teil des MVP.

Weitere technische Details können im Anschlusskonzept Data Provider (AD-NOOTS-02)

> nachgelesen werden.

3. Installation der Anwendung für Server-Betrieb

Dieses Kapitel beschreibt die traditionelle Installation der Java-Anwendung auf einem Host-Server.

3.1. Systemvoraussetzungen

Die Systemvoraussetzungen für den Betrieb sind abhängig von der Anzahl der gleichzeitigen Zugriffe auf den SAK-DP. Je nach Mengengerüst können diese erheblich von den Mindestanforderungen abweichen.

Tabelle 1. Systemvoraussetzungen

Name	Mindestanforderungen
Java-JRE	JRE 21, getestet mit Adoptium Temurin
Betriebssystem	nicht vorgegeben, Konformität mit BSI-Grundschutz[1] wird vorausgesetzt
Arbeitsspeicher	1024MB
Speicherplatz	512MB
Prozessor	2 GHz Dual Core, 64-Bit
Netzwerk	Internetverbindung (direkt oder über Proxy) auf Port 443

3.2. Installation

WICHTIG	Halten Sie sich hier an Ihre betrieblichen Vorgaben für einen sicheren Betrieb. Die hier beschriebenen Schritte dienen nur als Beispiel.

Der SAK-DP wird als ausführbare JAR-Datei (Java Archive) bereitgestellt.

Die JAR-Datei sollte in der Laufzeitumgebung abgelegt werden, in unserem Beispiel ist das `/opt/sak-dp`

Kopieren Sie die JAR-Datei mit der entsprechenden Version auf den Zielrechner, beispielsweise mit `scp`:

```
scp sak-dp-VERSION-TAG.jar admin@laufzeitumgebungshost:~
```

Loggen Sie sich per `ssh` auf dem Rechner ein:

```
ssh admin@laufzeitumgebungshost:~
```

Die Berechtigung sollte nur auf das Nötigste beschränkt werden, damit nur

der ausführende User der Serveranwendung Leserechte hat. In unserem Beispiel läuft die Serveranwendung mit dem User `sak-dp` der der Linux Gruppe `sak-dp` angehört.

```
sudo mkdir /opt/sak-dp
sudo chown root:sak-dp /opt/sak-dp
sudo chmod 750 /opt/sak-dp

sudo mv ~/sak-dp-VERSION-TAG.jar /opt/sak-dp

sudo chown root:sak-dp /opt/sak-dp/sak-dp-VERSION-
TAG.jar
sudo chmod 640 /opt/sak-dp/sak-dp-VERSION-TAG.jar
```

WICHTIG	Die Anwendung sollte unter keinen Umständen über den `root`-User des Systems, sondern idealerweise mittels eines eigenen Benutzers mit möglichst minimalen Rechten ausgeführt werden.

3.2.1. Start und Konfiguration der Anwendung

Die Konfigurationen des SAK-DP findet mittels einer bereitgestellten `application.properties` statt, welche auf die Laufzeitumgebung angepasst werden muss. Die Parameter sind in der Datei beschrieben und enthalten Beispielwerte, an denen sich orientiert werden kann. Die Konfigurationsdatei muss als Parameter übergeben werden. Die Dateirechte auf die Konfigurationsdatei sollten auch hier so gesetzt werden, sodass nur die Anwendung darauf zugreifen, aber nicht ändern kann.

```
java -jar sak-dp-VERSION-TAG.jar
--spring.config.additional
-location=application.properties
```

Weiterführende Hinweise zur Konfiguration

Im oben genannten Beispiel wird davon ausgegangen, dass die `application.properties`-Datei im identischen Verzeichnis wie die Anwendung (.jar) selbst liegt. Wenn die Konfiguration aus einem anderen Verzeichnis geladen werden soll, kann dieses sowohl relational, sowie absolut angegeben werden. Außerdem können mehrere Dateien, auch *optional* angegeben werden. Hier muss die Reihenfolge beachtet werden. Gleiche Konfigurationsschlüssel aus Dateien, welche früher in der Kette angegeben werden, werden

durch spätere Dateien überschrieben.

```
java -jar sak-dp-VERSION-TAG.jar
--spring.config.additional-location=\

  #Konfigurations-Datei im gleichen Ordner; muss
vorhanden sein
  application.properties,\

  #externe Konfigurations-Datei ; muss vorhanden
sein
  /opt/default.properties,\

  #externe Konfigurations-Datei ; optional,
produziert keinen Fehler, wenn nicht vorhanden
  optional:/opt/optional.properties
```

Weitere Informationen zu der Angabe von externen Konfigurationen können der offiziellen Spring Boot Dokumentation; Kapitel "Externalized Configuration"; Abschnitt "External Application Properties" entnommen werden.

Die Anwendung sollte nun gestartet sein und folgende Meldung in der Ausgabe erscheinen:

```
Komponente SAK-DP erfolgreich gestartet
```

3.2.2. Endpunkte

In der Standardkonfiguration ist die Serveranwendung über Port **8080** sowie der Managementendpunkt über Port **9000** erreichbar.

WICHTIG	Diese Netzwerkports sollten über geeignete Firewallregeln abgesichert werden, um unbefugten Zugriff zu unterbinden. Halten Sie sich auch hier an Ihre betrieblichen Vorgaben für einen sicheren Betrieb.

In den offiziellen Test- und Produktionsumgebungen von NOOTS ist es verpflichtend, die Endpunkte nicht ungeschützt und unverschlüsselt zu betreiben. Es können hierbei verschiedenen Varianten genutzt werden. Je nach vorhandener Infrastruktur kann auch ein Betrieb hinter einem Proxy sinnvoll sein. In einer lokalen Teststellung ohne Anbindung gegen das NOOTS-System (z.B. mittels Mocks) können die Endpunkte auch weniger abgesichert betrieben werden.

Tabelle 2. Absicherung der Endpunkte

Art	Erklärung
HTTPS	Die Konfiguration erfolgt über die `application.properties`
mTLS	Die Konfiguration erfolgt über die `application.properties`
HTTP Basic-Auth	Die Konfiguration erfolgt über die `application.properties`
Die obigen Methoden können bei Bedarf durch Apache-, Nginx oder einen anderen geeigneten Reverse Proxy abgebildet werden	Die Konfiguration erfolgt über das externe genutzte System

In jedem Fall ist die Nutzung von TLS 1.3 für die Kommunikation zwischen dem Data Provider und dem SAK-DP verpflichtend! Dies wurde in der Technische Richtlinie BSI TR-03190 (Kommentierungsfassung. Version 0.7) im Punkt [DCDP.06] festgelegt.

HINWEIS

Die Technische Richtlinie des BSI liegt aktuell nur in einer Kommentierungsfassung vor. In dieser vorzeitigen Version ist die Nutzung von TLS 1.3 noch wie folgt dokumentiert: „TLS **SOLLTE** in der Version 1.3 verwendet werden.“ In der finalen Richtlinie des BSI wird es allerdings heißen: „TLS **MUSS** [...]“

3.2.3. Start und Stop

Damit die Anwendung nicht immer manuell über die Konsole gestartet werden muss, können Mittel wie zum Beispiel systemd oder init.d genutzt werden.

Wir empfehlen hierzu die offizielle Spring Boot Dokumentation; Kapitel "Installing Spring Boot Applications".

4. Installation in einem Kubernetes-Cluster

4.1. Vorbereitung der Systemumgebung

Dieses Kapitel beschreibt, welche Vorbereitungen für den Betrieb der Anwendung Sicherer Anschlussknoten des Data Providers (SAK-DP) in der Systemumgebung nötig sind.

4.1.1. Kubernetes Laufzeitumgebung

SAK-DP ist für das Deployment auf einer Kubernetes-konformen Container-Orchestrierungsplattform vorgesehen. Die Laufzeitumgebung muss dem BSI-Grundschutz[1] entsprechen, insbesondere der Baustein "APP4.4 Kubernetes" ist zu beachten.

Falls die "Deutsche Verwaltungscloud"[2] genutzt werden soll, sind die entsprechenden FITKO-Richtlinien[3] zu beachten.

4.1.2. Verifzierung der Attestate am Container-Image

Die SBOMs und CVE-Reporte sind als Metadaten am Container-Image attestiert[4]. Vor einem Deployment müssen die Attestate an jedem Image verifiziert werden, um sicherzustellen, dass das Image unverändert und geprüft ausgeliefert wurde. Das folgende Beispiel zeigt die Verifikation mittels **Cosign**. Dabei müssen der Public-Key und der Repository-Pfad des Images angegeben werden.

Verifizierung attestierter Metadaten.

```
cosign verify-attestation \
    --key $PUBLIC_KEY \
    --insecure-ignore-tlog=true \
    --type=vuln \
    $IMAGE_REPOSITORY_PATH
```

> **TIP** Gleichermaßen kann mit der Option `--type=sbom` die SBOM verifiziert werden.

Aktuell wurde der Ablageort des hier verwendeten Public-Keys noch nicht festgelegt. Somit ist die Prüfung mit **Cosign** noch nicht möglich.

4.1.3. Konfiguration des Kubernetes Clusters

Ein Kubernetes Namespace muss für den Sicherer Anschlussknoten des Data Providers angelegt werden. Alternativ, kann der Namespace während der Installation der Anwendung erstellt werden.

Die Helm-Charts von SAK-DP definieren Standard-Ressourcenlimits. Das Cluster muss dementsprechend ausreichend CPU und RAM bereitstellen.

Um die Sicherheit sensibler Daten im Cluster zu gewährleisten, muss die Verschlüsselung des `etcd`-Datenspeichers aktiviert werden. `etcd` ist die zentrale Schlüssel-Wert-Datenbank, in der Kubernetes neben anderen Informationen auch vertrauliche Daten wie Secrets ablegt. Ohne aktivierte Verschlüsselung würden diese Daten im Klartext vorliegen. Für weitere Informationen diesbezüglich kann die Kubernetes Dokumentation[5] herangezogen werden.

4.2. Installation der Anwendung

SAK-DP wird als OCI Container-Image und Helm-Charts bereitgestellt. Diese befinden sich im Auslieferung-OCI-Repository und können von dort bezogen werden.

Die Installation der Anwendung erfolgt mittels des Kommandozeile-Tools `helm`. Angenommen die Helm-Charts für SAK-DP sind in `$RELEASE_REPOSITORY` vorhanden, zeigt das folgende Beispiel die Installation eins der beiden Anwendungen:

Installation der IT-Systeme

```
helm upgrade "$APP_RELEASE"
"oci://$RELEASE_REPOSITORY/$APP_HELM_CHART_NAME" \
    --version "$APP_HELM_CHART_VERSION" \
    --install \
    --namespace "$APP_NAMESPACE" \
    --values "$APP_VALUES"
```

Weitere Optionen können aus der Dokumentation zum `helm upgrade`[6] entnommen werden.

TIP	Durch die Option `--install` kann der Befehl auch für die erstmalige Installation, verwendet werden.

4.3. Konfiguration der Anwendung

Die Konfigurationen von SAK-DP findet mittels einer Helm Values-Datei[7] statt.

Die Helm Values-Dateien können dem Anhang entnommen werden : Anhang D: Helm Chart Values-Datei für Kubernetes-Betrieb

WICHTIG	Mit "(*)" gekennzeichnete Parameter müssen durch eine überschreibende Values-Datei oder per "--set"-Option gesetzt werden. Das Setzen/überschreiben von Parametern, die mit "(+)" gekennzeichnete sind, ist empfohlen, aber nicht obligatorisch. Zur Anpassung des Deployments in der vorgesehenen

Umgebung müssen ggf. auch weitere Parameter angepasst werden.

5. Regulärer Wirkbetrieb

5.1. Logging

Der SAK-DP verwendet standardmäßig die Konsole für Logausgaben. Das Log-Nachrichtenformat entspricht dem Elastic Common Schema (ECS).

5.2. Statusprüfung der Anwendungen

Der SAK-DP stellt die folgenden Endpunkte über den Management Port zur Statusprüfung bereit:

HINWEIS	Diese Endpunkte sollten nur intern (z.B. hinter einer Firewall) betrieben werden. Eine Freigabe für einen öffentlichen, nicht regulierten Zugriff ist zu vermeiden!

Tabelle 3. Status-Endpunkte

Endpunkt	Beschreibung
`/actuator/health`	Gesamtstatus der Anwendung. Liefert `{"status":"UP|DOWN"}` als Antwort

5.3. Monitoring

Der SAK-DP kann zusätzliche Metriken liefern. Es stehen folgende Endpunkte zur Verfügung:

- `/metrics`
- `/prometheus` (Spring Boot Dokumentation; Kapitel "Prometheus (prometheus)")

Die Endpunkte können über die `application.properties` aktiviert werden und sind über den Management-Port erreichbar.

Es werden folgende Standard-Metriken bereitgestellt:

HINWEIS	Diese Endpunkte sollten nur intern (z.B. hinter einer Firewall) betrieben werden. Eine Freigabe für einen öffentlichen, nicht regulierten Zugriff ist zu vermeiden!

Tabelle 4. Standard-Metriken

Metriken	Beschreibung	Path
JVM	Metriken zum Speicherverbrauch, zur Garbage Collection, Threads und geladenen Klassen.	`/actuator/metrics/jvm/*`
System	Metriken zur CPU-Auslastung, dem Dateisystem und der Uptime der Anwendung.	`/actuator/metrics/disk/` `/actuator/metrics/system/` `/actuator/metrics/process/*`
Application Startup	Metriken zur Startzeit der Anwendung.	`/actuator/metrics/application/*`
Logger	Metriken zu Logging-Events.	`/actuator/metrics/logback/events`
REST Services	Metriken zu aufgerufenen REST-Services.	`/actuator/metrics/http/*`

5.4. Backup und Recovery

Wir empfehlen mindestens folgende Daten zu sichern:

- für Host-Server-Betrieb:
  - Angepasste `application.properties` die für den Betrieb verwendet wird.
  - Logs der Anwendung unter Beachtung der geltenden Datenschutzbestimmungen.
- für Container-Betrieb:
  - Angepasste `values.yaml` die für den Betrieb verwendet wird.

5.5. Hinweise für Störungsdiagnose und -behandlung

5.5.1. Debug-Logging

Die Konfiguration wird mit Log-Level `INFO` ausgeliefert, d.h. das Loggen von Debug-Ausgaben ist ausgeschaltet. Es kann aber per Konfigurationsparameter eingeschaltet werden:

- für Host-Server-Betrieb:
  - In `application.properties`: `logging.level.root=DEBUG` setzen.
- für Container-Betrieb:
  - Für den ausführenden POD die Umgebungsvariable `LOGGING_LEVEL_ROOT` auf den Wert `DEBUG` setzen.

ACHTUNG	Das Logging sollte auf der Produktivumgebung nie auf `DEBUG` gestellt werden. Aufgrund der umfangreichen Datenausgabe könnte es zu Performance-Problemen kommen. Gleichzeitig ist nicht sichergestellt, dass keine sicherheitsrelevanten Daten ausgegeben werden.

5.5.2. Bei Ressourcengrenzenüberschreitung des Anwendungscontainers

Für den Fall, dass im Container-Betrieb ein Container an seine Ressourcengrenzen kommt oder diese überschreitet, gibt es keine anwendungsspezifischen Handlungsempfehlungen. Es sind die üblichen Maßnahmen gemäß Betriebsrichtlinien des Anwendungsbetreibers auszuführen.

Anhang

Anhang A: Technisches Glossar

HINWEIS	Das hier hinterlegte Glossar ist übergreifend für alle Dokumente des SAKs identisch

Tabelle 5. Technisches Glossar für SAK-DP

Begriff	Bedeutung
Laufzeitumgebung	Umgebung, in der ein Service oder eine Anwendung betrieben werden kann.

Anhang B: Fachliches Glossar

HINWEIS	Das hier hinterlegte Glossar ist übergreifend für alle Dokumente des SAKs identisch

Tabelle 6. Fachliches Glossar für SAK-DP

Begriff	Bedeutung
Data Consumer	Ein Data Consumer ist ein NOOTS-Teilnehmer, welcher Nachweise aus dem System, konkret den Data Providern, abrufen möchte.
Data Provider	Ein Data Provider ist ein NOOTS-Teilnehmer der den Data Consumern Nachweise zur Verfügung stellt.
Identity Access Management für Behörden (IAM-B)	Das IAM für Behörden ist eine zentrale Komponente des NOOTS. Sie stellt für den Zugriff auf andere NOOTS-Komponenten und Data Provider vertrauenswürdige und zuverlässige Zugriffstoken für Data Consumer aus, die den Zugriff des DC auf die NOOTS-Komponenten und die DPs autorisiert, die Identität des DCs bestätigt und die Berechtigungen des DCs gegenüber den NOOTS-Komponenten und DPs beinhaltet.
Registerdatennavigation (RDN)	Die Registerdatennavigation ist eine zentrale Komponente des NOOTS. Sie liefert auf Anfrage die Information, von welchem technischen Dienst einer Behörde ein gesuchter Nachweistyp abgerufen werden kann. Dazu übermittelt die RDN notwendige Verbindungsparameter an den anfragenden Data Consumer.
Nachweis	Nachweise sind Unterlagen und Daten in elektronischer Form, die zur Ermittlung des Sachverhaltes in Verwaltungsverfahren geeignet sind.

Begriff	Bedeutung
National-Once-Only-Technical-System (NOOTS)	Das NOOTS ist ein gemeinsames informationstechnisches System aus IT-Komponenten, Schnittstellen und Standards, das öffentlichen Stellen den Abruf und die Übermittlung von elektronischen Nachweisen und Daten national und grenzüberschreitend aus Datenbeständen öffentlicher Stellen, zur Erfüllung öffentlicher Aufgaben ermöglicht.
Sicherer Anschlussknoten (SAK)	Ein sicherer Anschlussknoten (SAK) ist eine für Data Consumer - und Provider bereitgestellte NOOTS-Komponente, die dezentral im Verantwortungsbereich der NOOTS-Teilnehmer betrieben wird. Er ermöglicht den NOOTS-Teilnehmern über ein interoperables Anschlussprotokoll einen einfachen und sicheren Zugriff auf die NOOTS Transportinfrastruktur.

Anhang C: Betrieblich-Relevante Konfigurationsparameter für Server-Betrieb

```
# ----------------------------------------
# SAK-DP Transportprotokoll Einstellungen
# ----------------------------------------

# SAK-Server Adresse
server.address = localhost
# SAK-Server Port
server.port = 8443

# max. erlaubte Anzahl von Request, die SAK-DP innerhalb
des gegebenen Zeitfensters (in Millisekunden) annimmt.
#sakdp.ratelimiter.limitForPeriod = 50
#sakdp.ratelimiter.limitRefreshPeriod = 1000

# SAK-Server TLS
# Hardware-basierte Schlüsselspeicher (empfohlen)
spring.ssl.bundle.jks.server.keystore.type = PKCS11
spring.ssl.bundle.jks.server.keystore.password =
spring.ssl.bundle.jks.server.key.alias = sakdp-server-
crt
# Datei-basierte Schlüsselspeicher (nicht empfohlen,
bitte nur PKCS12 in diesem Fall verwenden)
#spring.ssl.bundle.jks.server.keystore.type = PKCS12
#spring.ssl.bundle.jks.server.keystore.location = sakdp-
server-cert.p12
#spring.ssl.bundle.jks.server.keystore.password =
```

```
# Separate TLS Server-Zertifikaten, falls der SAK unter
unterschiedlichen Hostnames exponiert ist
#server.ssl.server-name-bundles[0].server-name = sak-
dp1.example.com
#server.ssl.server-name-bundles[0].bundle = sakdp1
#spring.ssl.bundle.jks.sakdp1.keystore.type = PKCS11
#spring.ssl.bundle.jks.sakdp1.keystore.password =
#spring.ssl.bundle.jks.sakdp1.key.alias = sakdp1-server-
crt
#spring.ssl.bundle.jks.sakdp1.options.enabled-protocols
= ${sakdp.server.ssl.enabled-protocols}
#spring.ssl.bundle.jks.sakdp1.options.ciphers =
${sakdp.server.ssl.enabled-ciphers}

# SAK-Server TLS Client-Auth
# Truststore mit vertrauenswürdigen Client-Cert-CAs
spring.ssl.bundle.jks.server.truststore.type = PKCS12
spring.ssl.bundle.jks.server.truststore.location =
sakdp-server-truststore.p12
spring.ssl.bundle.jks.server.truststore.password =

# Truststore mit erlaubten Client-Zertifikaten ohne
Zertfikatskette (Pinning)
spring.ssl.bundle.jks.sakdcclientcerts.truststore.type =
PKCS12
spring.ssl.bundle.jks.sakdcclientcerts.truststore.locati
on = sakdc-client-certs.p12
spring.ssl.bundle.jks.sakdcclientcerts.truststore.passwo
rd =

# Truststore-Konfiguration für die NOOTS-Zugriffstoken-
Validierung
spring.ssl.bundle.jks.iamsigncert.truststore.type =
PKCS12
spring.ssl.bundle.jks.iamsigncert.truststore.location =
iam-sign-cert.p12
spring.ssl.bundle.jks.iamsigncert.truststore.password =

# NOOTS IAM Hostname
noots.iam.hostname = iam.noots.gov.de

# ---------------------------------------
# SAK-DP Anschlussprotokoll Einstellungen
# ---------------------------------------

# Einstellung für die Verbindung gegen den hinterlegten
Data-Provider
```

```
#
------------------------------------------------------
---------------
# Verbindungsparameter des Data-Providers
# Anfragen gegen den Data-Provider werden mit folgendem
Schema durchgeführt:
# {{dataprovider.url}}{{dataprovider.resource}}/evidence
# Dataprovider Host-URL, ggfs. mit Port
dataprovider.url = https://dp-implementierung-url:8443
# Resource-Path des Request. Dieser wird als Prefix für
die eigentliche URL verwendet.
# Derzeit muss immer ein Prefix-Path angegeben werden.
Leerwerte wie "" oder "/" sind nicht zulässig.
dataprovider.resource = /api

# Optionale Basic-Auth Authentifizierung des Data-
Providers
# Schnittstellen des Data-Provider können laut der
Schnittstellenspezifikation optional via
# Basic-Auth gesichert werden. Sollte dies der Fall
sein, muss hier
# "dataprovider.basicauth.enabled" auf "true" gesetzt,
und Username + Passwort angegeben werden.
dataprovider.basicauth.enabled = true
dataprovider.basicauth.username =
dataprovider.basicauth.password =

# Optionale Proxy-Konfiguration für die Verbindung
zwischen SAK-DP und DP
#
------------------------------------------------------
---------------
#spring.cloud.gateway.server.webflux.httpclient.proxy.ho
st = ""
#spring.cloud.gateway.server.webflux.httpclient.proxy.po
rt = ""
# Optionale Authentifizierung des Proxies
#spring.cloud.gateway.server.webflux.httpclient.proxy.us
ername =
#spring.cloud.gateway.server.webflux.httpclient.proxy.pa
ssword =

# Client-Zertifikate des SAK
# Hardware-basierte Schlüsselspeicher (empfohlen)
spring.ssl.bundle.jks.client.keystore.type = PKCS11
spring.ssl.bundle.jks.client.keystore.password =
spring.ssl.bundle.jks.client.key.alias = sakdp-client-
```

```
crt
# Datei-basierte Schlüsselspeicher (nicht empfohlen,
bitte nur PKCS12 in diesem Fall verwenden)
#spring.ssl.bundle.jks.client.keystore.type = PKCS12
#spring.ssl.bundle.jks.client.keystore.location = sakdp-
client-cert.p12
#spring.ssl.bundle.jks.client.keystore.password =

spring.ssl.bundle.jks.client.truststore.type = PKCS12
spring.ssl.bundle.jks.client.truststore.location =
sakdp-client-truststore.p12
spring.ssl.bundle.jks.client.truststore.password =

# ----------------------------------------
# SAK-DP Allgemeine Anwendungskonfiguration
# ----------------------------------------

# Monitoring-Endpunkte der Anwendung
# Die folgenden Endpunkte werden über die URL
"http://{{management.server.address}}:{{management.serve
r.port}}/actuator/{{endpoint}}" auf dem
# konfigurierten Management-Port standardmäßig zur
Verfügung gestellt und können über den
Konfigurationswert
# "management.endpoints.web.exposure.include" selektiv
aktiviert und damit exponiert werden:
# - health: Der Health-Endpunkt stellt Informationen
über die Readiness und Liveness der Anwendung bereit
# - prometheus: Der Prometheus-Endpunkt stellt eine
Vielzahl an Metriken im Prometheus-Format zur Verfügung
# - info: Der Info-Endpunkt liefert interne
Informationen der Anwendung, z.B. Daten zum Build oder
zur aktuellen Laufzeitumgebung
# - sbom: Der SBOM-Endpunkt liefert eine Auflistung
aller Abhängigkeiten und Bibliotheken im SAK-DP
management.endpoints.web.exposure.include =
health,prometheus
# Verbindungseinstellungen für die Management-Endpoints
management.server.address = localhost
management.server.port = 9000

# Logging-Einstellungen
# Mögliche Log-Level: ERROR, WARN, INFO, DEBUG, TRACE
# In einer Produktivumgebung sollte immer min. WARN
gesetzt werden
# Basis-Level für alle Logmeldungen der Anwendung
# (spezifische Log-Level überschreiben die Basis)
```

```
logging.level.root = INFO
```

Anhang D: Helm Chart Values-Datei für Kubernetes-Betrieb

```
# Standardwerte für das Chart des Sicheren
Anschlussknotens für Data Provider
# Mit @erforderlich gekennzeichnete Parameter müssen
durch eine überschreibende Values-Datei oder per "--
set"-Option gesetzt werden.
# Zur Anpassung des Deployments in der vorgesehenen
Umgebung müssen ggf. auch weitere Parameter angepasst
werden.

# replicaCount -- definiert die Anzahl der Replikate für
das Deployment. Weitere Informationen:
https://kubernetes.io/docs/concepts/workloads/controller
s/replicaset/
replicaCount: 1

# image -- konfiguriert das Container-Image für das
Deployment. @erforderlich
image:
  # image.repository -- definiert das Repository des
Container-Images. @erforderlich
  repository: null
  # image.tag -- definiert den Tag des Container-Images.
@erforderlich
  tag: null
  # image.pullPolicy -- legt die Pull-Richtlinie für das
Container-Image fest.
  pullPolicy: IfNotPresent

# imagePullSecrets -- enthält Referenzen auf Secrets,
die für den Zugriff auf private Container-Image-
Repositories benötigt werden.
imagePullSecrets: null
# nameOverride -- überschreibt den Standardnamen des
Charts.
nameOverride: null
# fullnameOverride -- überschreibt den vollständigen
Namen des Charts.
fullnameOverride: null

# serviceAccount -- konfiguriert den Kubernetes
ServiceAccount, der für das Deployment verwendet wird.
```

```yaml
serviceAccount:
  # serviceAccount.create -- gibt an, ob ein neuer
ServiceAccount erstellt werden soll.
  create: true
  # serviceAccount.automount -- legt fest, ob das
ServiceAccount-Token automatisch in den Pod gemountet
wird.
  automount: false
  # serviceAccount.annotations -- definiert
Annotationen, die dem ServiceAccount hinzugefügt werden.
  annotations: { }
  # serviceAccount.name -- legt den Namen des zu
verwendenden ServiceAccounts fest. Wenn nicht gesetzt
und create auf true steht, wird ein Name generiert.
  name: ""

# podAnnotations -- definiert Kubernetes-Annotationen,
die auf die Pods angewendet werden.
podAnnotations: { }

# podLabels -- definiert Kubernetes-Labels, die auf die
Pods angewendet werden.
podLabels: { }

# podSecurityContext -- konfiguriert den Pod-weiten
SecurityContext, einschließlich Dateisystem- und
Gruppenrechten.
podSecurityContext:
  # podSecurityContext.fsGroup -- legt die Dateisystem-
Gruppe für Dateien im Pod fest.
  fsGroup: 1000
  # podSecurityContext.supplementalGroups -- definiert
zusätzliche Gruppen, die dem Pod zugewiesen werden.
  supplementalGroups:
    - 1000

# securityContext -- konfiguriert den SecurityContext
für den Container, einschließlich Benutzer- und
Gruppenrechten.
securityContext:
  # securityContext.capabilities.drop -- entfernt alle
Linux-Kernel-Fähigkeiten, um erweiterte Systemrechte zu
entziehen.
  capabilities:
    drop:
      - ALL
  # securityContext.readOnlyRootFilesystem -- aktiviert
```

```
ein schreibgeschütztes Root-Dateisystem.
  readOnlyRootFilesystem: true
  # securityContext.runAsNonRoot -- erzwingt, dass der
Container nicht als Root-Benutzer ausgeführt wird.
  runAsNonRoot: true
  # securityContext.runAsUser -- legt die Benutzer-ID
fest, unter der der Container ausgeführt wird.
  runAsUser: 1000
  # securityContext.runAsGroup -- legt die Gruppen-ID
fest, unter der der Container ausgeführt wird.
  runAsGroup: 1000
  # securityContext.allowPrivilegeEscalation --
verhindert Privilegienerhöhungen innerhalb des
Containers.
  allowPrivilegeEscalation: false
  # securityContext.privileged -- verhindert, dass der
Container im privilegierten Modus ausgeführt wird.
  privileged: false

# javatoolOptionsExtra -- definiert zusätzliche
Optionen, die der Anwendung als JAVA_TOOL_OPTIONS
übergeben werden.
javatoolOptionsExtra: ""

# env -- ist eine Liste zusätzlicher Environment-
Variablen, die dem Container übergeben werden können.
env: [ ]

# sakdp -- enthält die Hauptkonfiguration für den
Sicheren Anschlussknoten für Data Provider.
sakdp:
  # sakdp.mtlsAuthentication -- konfiguriert die mTLS-
Authentifizierung für ausgehende Verbindungen.
  mtlsAuthentication:
    # sakdp.mtlsAuthentication.enabled -- aktiviert oder
deaktiviert die mTLS-Authentifizierung.
    enabled: true
    keystore:
      # sakdp.mtlsAuthentication.keystore.data --
enthält die einzeilig Base64-kodierten Keystore-Bytes.
Wenn gesetzt, wird secretRef überschrieben.
      data: ""
      # sakdp.mtlsAuthentication.keystore.secretRef --
ist die Referenz auf ein Secret, das den Keystore
enthält.
      secretRef: "sakdp-mtls-keystore"
      # sakdp.mtlsAuthentication.keystore.secretRefKey
```

```
-- ist der Key innerhalb des Secrets, der den Keystore
enthält.
      secretRefKey: "sakdp-mtls-keystore.p12"
      # sakdp.mtlsAuthentication.keystore.password --
ist das Passwort für den Keystore. Ein leerer String
("") ist ein gültiges Passwort.
      # Um ein bestehendes Secret mittels
'passwordSecretRef' zu nutzen, muss 'password: null'
gesetzt werden.
      password: null
      #
sakdp.mtlsAuthentication.keystore.passwordSecretRef --
ist die Referenz auf ein Secret, das das Passwort für
den Keystore enthält.
      passwordSecretRef: "sakdp-mtlspassword"
      #
sakdp.mtlsAuthentication.keystore.passwordSecretRefKey
-- ist der Key innerhalb des Secrets, der das Passwort
enthält.
      passwordSecretRefKey: "password"

  # sakdp.clientTruststore -- konfiguriert den
Truststore für ausgehende TLS-Verbindungen.
  clientTruststore:
    # sakdp.clientTruststore.data -- enthält die
einzeilig Base64-kodierten Truststore-Bytes. Wenn
gesetzt, wird secretRef überschrieben.
    data: ""
    # sakdp.clientTruststore.enabled -- aktiviert die
Verwendung eines benutzerdefinierten Truststores.
    enabled: true
    # sakdp.clientTruststore.secretRef -- ist die
Referenz auf ein Secret, das den Truststore enthält.
    secretRef: "sakdp-client-truststore"
    # sakdp.clientTruststore.secretRefKey -- ist der Key
innerhalb des Secrets, der den Truststore enthält.
    secretRefKey: "sakdp-client-truststore.p12"
    # sakdp.clientTruststore.password -- ist das
Passwort für den Truststore. Ein leerer String ("") ist
ein gültiges Passwort.
    # Um ein bestehendes Secret mittels
'passwordSecretRef' zu nutzen, muss 'password: null'
gesetzt werden.
    password: null
    # sakdp.clientTruststore.passwordSecretRef -- ist
die Referenz auf ein Secret, das das Passwort für den
Truststore enthält.
```

```
    passwordSecretRef: "sakdp-clienttruststorepassword"
    # sakdp.clientTruststore.passwordSecretRefKey -- ist
der Key innerhalb des Secrets, der das Passwort enthält.
    passwordSecretRefKey: "password"

  # sakdp.rateLimitation -- konfiguriert die
Ratenbegrenzung für Anfragen.
  rateLimitation:
    # sakdp.rateLimitation.limitForPeriod -- legt die
maximale Anzahl von Anfragen pro Zeitraum fest.
    limitForPeriod: 50
    # sakdp.rateLimitation.limitRefreshPeriod -- legt
den Zeitraum in Millisekunden fest, nach dem die
Anfrageanzahl zurückgesetzt wird.
    limitRefreshPeriod: 1000

  # sakdp.whitelistTruststore -- konfiguriert den
Whitelist-Truststore für Client-Zertifikate.
  whitelistTruststore:
    # sakdp.whitelistTruststore.data -- enthält die
Base64-kodierten Truststore-Bytes. Wenn gesetzt, wird
secretRef überschrieben.
    data: ""
    # sakdp.whitelistTruststore.secretRef -- ist die
Referenz auf ein Secret, das den Truststore enthält.
    secretRef: "sakdp-whitelisttruststore"
    # sakdp.whitelistTruststore.secretRefKey -- ist der
Key innerhalb des Secrets, der den Truststore enthält.
    secretRefKey: "truststore.p12"
    # sakdp.whitelistTruststore.password -- ist das
Passwort für den Truststore. Ein leerer String ("") ist
ein gültiges Passwort.
    # Um ein bestehendes Secret mittels
'passwordSecretRef' zu nutzen, muss 'password: null'
gesetzt werden.
    password: null
    # sakdp.whitelistTruststore.passwordSecretRef -- ist
die Referenz auf ein Secret, das das Passwort für den
Truststore enthält.
    passwordSecretRef: "sakdp-
whitelisttruststorepassword"
    # sakdp.whitelistTruststore.passwordSecretRefKey --
ist der Key innerhalb des Secrets, der das Passwort
enthält.
    passwordSecretRefKey: "password"

  # sakdp.accessToken -- konfiguriert die Überprüfung
```

```
von AccessTokens.
  accessToken:
    # sakdp.accessToken.sslClientCertHeader -- ist der
HTTP-Request-Header mit dem Client-Zertifikat aus der
eingehenden TLS-Verbindung.
    sslClientCertHeader: "ssl-client-cert"
    # sakdp.accessToken.sslClientCertHeaderEncoding --
ist die Kodierung des HTTP-Request-Headers mit dem
Client-Zertifikat aus der eingehenden TLS-Verbindung.
    sslClientCertHeaderEncoding: "url"
    # sakdp.accessToken.roleGetEvidence -- definiert die
Rollen, die erforderlich sind, um Nachweise abzurufen.
    roleGetEvidence: "nachweis.national"
    # sakdp.accessToken.algorithms -- gibt den
Algorithmus an, mit dem das Token signiert sein muss.
    algorithms: "ES256"
    # sakdp.accessToken.rolesclaimname -- ist der Name
des Claims im Token, der die Rollen enthält.
    rolesclaimname: "scope"
    # sakdp.accessToken.issuer -- ist der erwartete
Aussteller des Tokens.
    issuer:
    # sakdp.accessToken.audiences -- definiert die
erwartete Audience des Tokens.
    audiences: "urn:regmo:components"
    iambTruststore:
      # sakdp.accessToken.iambTruststore.cabundle --
enthält CA-Zertifikate als PEM. Wenn gesetzt, wird
secretRef überschrieben.
      cabundle: ""
      # sakdp.accessToken.iambTruststore.secretRef --
ist die Referenz auf ein Secret, das den Truststore
enthält.
      secretRef: "sakdp-iamb-truststore"
      # sakdp.accessToken.iambTruststore.secretRefKey --
ist der Key innerhalb des Secrets, der den Truststore
enthält.
      secretRefKey: "sakdp-iamb-cabundle.pem"

iamb:
  # iamb.hostname -- IAM-B Hostname
  hostname:

# dataprovider -- enthält die Konfiguration für den Data
Provider.
dataprovider:
  # dataprovider.url -- ist die Basis-URL des Data
```

```
Providers. @erforderlich
  url:
  # dataprovider.resource -- ist der Pfad innerhalb der
Basis-URL. @erforderlich
  resource:
  credentials:
    basicAuth:
      # dataprovider.credentials.basicAuth.enabled --
aktiviert die Basic-Authentifizierung für den /evidence-
Endpunkt.
      enabled: true
      # dataprovider.credentials.basicAuth.username --
ist der Benutzername für die Basic-Authentifizierung.
      username:
      # dataprovider.credentials.basicAuth.password --
ist das Passwort für die Basic-Authentifizierung.
      password:

# resources -- konfiguriert CPU- und Speicher-Limits
sowie -Requests für den Pod.
resources:
  # resources.limits -- definiert die maximalen
Ressourcen, die der Container verwenden darf.
  limits:
    # resources.limits.cpu -- gibt das obere CPU-Limit
an.
    cpu: 1000m
    # resources.limits.memory -- gibt obere
Arbeitsspeichergrenze an.
    memory: 1024Mi
  # resources.requests -- definiert die reservierten
Ressourcen, die dem Container garantiert werden.
  requests:
    # resources.requests.cpu -- gibt die garantierte
CPU-Menge an.
    cpu: 500m
    # resources.requests.memory -- gibt den garantierten
Arbeitsspeicher an.
    memory: 1024Mi

# proxy -- konfiguriert einen optionalen HTTP-Proxy für
ausgehende Verbindungen.
proxy:
  # proxy.enabled -- aktiviert oder deaktiviert die
Verwendung des Proxys.
  enabled: true
  # proxy.ip -- ist die IP-Adresse des Proxy-Servers.
```

```
  ip: "127.0.0.1"
  # proxy.port -- ist der Port des Proxy-Servers.
  port: "3128"
  # proxy.noProxy -- listet Hosts auf, die ohne Proxy
direkt erreicht werden sollen.
  # proxy.noProxy Listen-Elemente werden durch `|`
getrennt und können Wildcards mit `*` enthalten.
  noProxy: "noots-dp-mock"

# networkPolicy -- konfiguriert die NetworkPolicy-Regeln
für das Deployment.
networkPolicy:
  # networkPolicy.enabled -- steuert, ob NetworkPolicies
im Cluster angewendet werden sollen.
  enabled: true
  # networkPolicy.wiremock -- gibt an, ob der DP Mock
(Wiremock) verwendet wird.
  wiremock: false
  # networkPolicy.ingress -- definiert eingehende Pod-
Verbindungen, die erlaubt sind.
  ingress:
      # networkPolicy.ingress.ports -- definiert die
erlaubten Ports für eingehenden Traffic.
    - ports:
        - protocol: TCP
          port: 8080
      # networkPolicy.ingress.from -- definiert die
erlaubten Quellen für eingehenden Traffic.
      from:
        - podSelector:
            matchLabels:
              app.kubernetes.io/name: rke2-ingress-nginx
          namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-system
  # networkPolicy.egressDns -- definiert ausgehende
Verbindungen zur Namensauflösung.
  egressDns:
    - to:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-system
          podSelector:
            matchLabels:
              k8s-app: kube-dns
      # networkPolicy.egressDns.ports -- definiert die
Ports, die für DNS-Auflösung benötigt werden.
```

```
      ports:
        - protocol: TCP
          port: 53
        - protocol: UDP
          port: 53
  # networkPolicy.egress -- definiert generelle
ausgehende Pod-Verbindungen.
  egress:
      # networkPolicy.egress.ports -- definiert die
erlaubten Ziel-Ports für ausgehenden Traffic.
    - ports:
        - protocol: TCP
          port: 443
  # networkPolicy.ingressPrometheus -- erlaubt
eingehende Prometheus-Verbindungen zum Management-Port.
  ingressPrometheus:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-
prometheus-stack
      ports:
        - protocol: TCP
          port: 9000

# Dies dient zur Einrichtung eines Dienstes. Weitere
Informationen finden Sie hier:
https://kubernetes.io/docs/concepts/services-
networking/service/
service:
  # service.create -- bestimmt, ob ein Service-Objekt
erstellt werden soll.
  create: true
  # service.type -- legt den Typ des Service fest (z. B.
ClusterIP, NodePort).
  type: ClusterIP
  # service.port -- ist der Port, über den der Service
den Traffic zum Pod weiterleitet.
  port: 8080
  # service.managementPort -- ist der Management-Port
(z. B. für Health/Prometheus).
  managementPort: 9000
  # service.exposeDebuggingPort -- steuert, ob ein
Debugging-Port offen gelegt werden soll.
  exposeDebuggingPort: false

# Dieser Block dient zur Einrichtung des Ingress.
```

```
Weitere Informationen finden Sie hier:
https://kubernetes.io/docs/concepts/services-
networking/ingress/
ingress:
  # ingress.enabled -- aktiviert oder deaktiviert das
Erstellen eines Ingress-Objekts.
  enabled: false
  # ingress.className -- legt die Ingress-
Klassenbezeichnung (IngressClass) fest.
  className: ""
  # ingress.annotations -- enthält zusätzliche
Annotationen, die dem Ingress-Objekt hinzugefügt werden.
  annotations: {}
  # ingress.hosts -- listet die Hosts auf, unter denen
die Anwendung erreichbar sein soll.
  hosts:
      # ingress.hosts[0].host -- ist der Hostname für
den ersten Host-Eintrag.
    - host: sicherer-anschlussknoten-dp.local
      # ingress.hosts[0].paths -- definiert Pfade und
Typen für den Host.
      paths:
          # ingress.hosts[0].paths[0].path -- ist der
HTTP-Pfad für den Ingress-Eintrag.
        - path: /
          # ingress.hosts[0].paths[0].pathType -- gibt
den Pfadtyp für das Ingress an.
          pathType: ImplementationSpecific
  # ingress.tls -- konfiguriert TLS-Einträge für das
Ingress-Objekt.
  tls: []

# Der Servicemonitor sowie der Endpunkt
actuator/prometheus wird aktiviert
servicemonitor:
  # servicemonitor.enabled -- schaltet den
ServiceMonitor ein oder aus.
  enabled: false

# Dies dient zur Einrichtung der Liveness-, Readiness-
und Startup-Probes. Weitere Informationen finden Sie
hier: https://kubernetes.io/docs/tasks/configure-pod-
container/configure-liveness-readiness-startup-probes/
livenessProbe:
  # livenessProbe.httpGet -- definiert die HTTP-GET-
Anfrage zur Prüfung der Liveness.
  httpGet:
```

```
    # livenessProbe.httpGet.path -- ist der Pfad, der
für die Liveness-Prüfung aufgerufen wird.
    path: /actuator/health/liveness
    # livenessProbe.httpGet.port -- ist der Port, der
für die Liveness-Prüfung verwendet wird.
    port: 9000
  # livenessProbe.initialDelaySeconds -- legt die
Verzögerung vor der ersten Prüfung fest.
  initialDelaySeconds: 30

# readinessProbe -- konfiguriert die Readiness-Probe des
Containers.
readinessProbe:
  # readinessProbe.httpGet -- definiert die HTTP-GET-
Anfrage zur Prüfung der Readiness.
  httpGet:
    # readinessProbe.httpGet.path -- ist der Pfad, der
für die Readiness-Prüfung aufgerufen wird.
    path: /actuator/health/readiness
    # readinessProbe.httpGet.port -- ist der Port, der
für die Readiness-Prüfung verwendet wird.
    port: 9000
  # readinessProbe.initialDelaySeconds -- legt die
Verzögerung vor der ersten Readiness-Prüfung fest.
  initialDelaySeconds: 30

# startupProbe -- konfiguriert die Startup-Probe des
Containers.
startupProbe:
  # startupProbe.httpGet -- definiert die HTTP-GET-
Anfrage zur Prüfung des Container-Starts.
  httpGet:
    # startupProbe.httpGet.path -- ist der Pfad für die
Startup-Prüfung.
    path: /actuator/health/liveness
    # startupProbe.httpGet.port -- ist der Port, der für
die Startup-Prüfung verwendet wird.
    port: 9000
  # startupProbe.periodSeconds -- legt das Prüfintervall
für die Startup-Probe fest.
  periodSeconds: 30
  # startupProbe.failureThreshold -- legt fest, nach wie
vielen Fehlversuchen der Pod als nicht startbar gilt.
  failureThreshold: 30

# volumes -- definiert zusätzliche Volumes in der Pod-
Spezifikation.
```

```yaml
volumes: [ ]

# volumeMounts -- definiert die Mount-Punkte für
zusätzliche Volumes im Container.
volumeMounts: [ ]

# nodeSelector -- definiert Node-Labels, die zur Auswahl
der Nodes für das Deployment verwendet werden.
# Weitere Informationen:
https://kubernetes.io/docs/concepts/scheduling-
eviction/assign-pod-node/
nodeSelector: { }

# affinity -- konfiguriert Affinity-Regeln für die Pods.
affinity: { }

# tolerations -- definiert Tolerations, um Pods auf
getainteten Nodes zuzulassen.
tolerations: [ ]

# clientCert -- konfiguriert die Erzeugung
selbstsignierter Client-Zertifikate.
clientCert:
  # clientCert.enabled -- aktiviert oder deaktiviert das
Erzeugen von Client-Zertifikaten.
  enabled: false
  # clientCert.metadata.name -- ist der Name für das
generierte Secret/Objekt.
  metadata:
    name:
  # clientCert.annotations.enabled -- gibt an, ob
Annotationen für das Zertifikat gesetzt werden sollen.
  annotations:
    enabled: false
    # clientCert.annotations.annotations -- enthält die
Annotationen als Key-Value-Paare.
    annotations: { }
  # clientCert.subject -- definiert den Subject-Teil des
Zertifikats.
  subject:
  # clientCert.privateKey.algorithm -- bestimmt den
Algorithmus für den privaten Schlüssel.
  privateKey:
    algorithm: ECDSA
    # clientCert.privateKey.size -- gibt die
Schlüssellänge an.
    size: 256
```

```
  # clientCert.usages -- listet die Verwendungszwecke
des Client-Zertifikats auf.
  usages:
    - "client auth"
  # clientCert.issuerRef -- referenziert den Issuer für
die Zertifikatsausstellung.
  issuerRef:
  # clientCert.commonName -- definiert den Common Name
im Zertifikat.
  commonName:

# -- pdb konfiguriert das Pod Disruption Budget für das
Deployment.
pdb:
  # -- pdb.enabled aktiviert die Erstellung eines Pod
Disruption Budgets.
  enabled: false
  # -- pdb.minAvailable definiert die minimale Anzahl an
Pods, die während einer Störung verfügbar bleiben
müssen.
  minAvailable: null
  # -- pdb.maxUnavailable definiert die maximale Anzahl
an Pods, die während einer Störung nicht verfügbar sein
dürfen.
  maxUnavailable: null
```

Quellen

Tabelle 7. Quellen

Name der Quelle	Link	Version
Anschlusskonzept Data Consumer (AD-NOOTS-01)	https://gitlab.opencode.de/noots/public/ad-noots/noots-architektur/ad-noots-release/-/blob/main/AD-NOOTS-XX/AD-NOOTS-01_+Anschlusskonzept+Data+Consumer+_DC_.md	Release Februar 2025
Anschlusskonzept Data Provider (AD-NOOTS-02)	https://gitlab.opencode.de/noots/public/ad-noots/noots-architektur/ad-noots-release/-/blob/main/AD-NOOTS-XX/AD-NOOTS-02_+Anschlusskonzept+Data+Provider+_DP_.md	Release Februar 2025

Name der Quelle	Link	Version
Spring Boot Dokumentation ; Kapitel "Installing Spring Boot Applications"	https://docs.spring.io/spring-boot/3.5/how-to/deployment/installing.html	Version 3.5.5
Spring Boot Dokumentation ; Kapitel "Externalized Configuration"; Abschnitt "External Application Properties"	https://docs.spring.io/spring-boot/reference/features/external-config.html#features.external-config.files	Version 3.5.5
Spring Boot Dokumentation ; Kapitel "Prometheus (prometheus)"	https://docs.spring.io/spring-boot/api/rest/actuator/prometheus.html	Version 3.5.5
Kubernetes Dokumentation ; Kapitel "Pod Lifecycle"; Abschnitt "Types of probe"	https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#types-of-probe	
Technische Richtlinie BSI TR-03190 (Kommentierungsfassung. Version 0.7)	https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03190/BSI-TR-03190.pdf	Stand Januar 2026

[1] https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/IT-Grundschutz-Bausteine/Bausteine_Download_Edition_node.html

[2] https://deutsche-verwaltungscloud.de

[3] https://docs.fitko.de/dvc

[4] https://helm.sh/docs/chart_template_guide/values_files

[5] https://kubernetes.io/docs/concepts/security/secrets-good-practices

[6] https://helm.sh/docs/helm/helm_upgrade

[7] https://helm.sh/docs/chart_template_guide/values_files