Bundesverwaltungsamt

Sicherer Anschlussknoten des Data Consumers 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 • 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 Consumers**" (*SAK-DC*) 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-DCs durchgeführt werden.

2.1. Fachliche Beschreibung

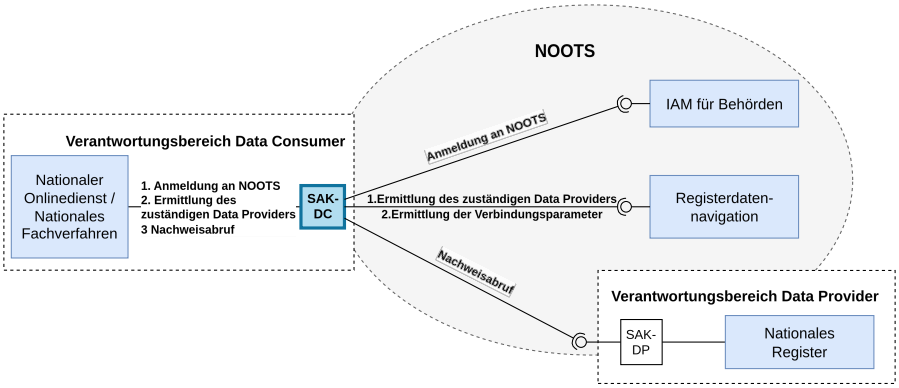

Abbildung 1. SAK-DC: Fachlicher Kontext

Die fachlichen Aufgaben des SAK-DCs sind die Durchführung des Nachweisabrufs unter Einhaltung des obligatorischen Sicherheitsprotokolls.

Im Einzelnen:

- der Abruf des NOOTS-Zugriffstokens vom IAM für Behörden
- die Abfrage des Data Providers und der benötigten Verbindungsparameter
- der eigentliche Nachweisabruf.

2.2. Technische Beschreibung

Der SAK-DC ermöglicht via das Anschlussprotokoll (bestehend aus REST-APIs) den sicheren Zugang zum NOOTS-Netzwerk für einen Data Consumer (*DC*).

- Zur Absicherung der Zugriffe auf SAK-DC kommen **Basic Auth** und **OAuth 2.0** zum Einsatz
    - Bei einer gleichzeitigen Nutzung von **Basic Auth** und **OAuth 2.0** (Zugriffstoken) auf den Schnittstellen */evidence-offer* und */evidence* müssen beide Header-Werte in **einem** HTTP-Header angegeben werden:

    Beispiel 1. Nutzung von **Bearer**- *und* **Basic**-*Schemas in einem Header*

    ```
    Authorization: Bearer [[Zugriffstoken]],Basic
    ```

```
[[Basic-Auth]]
```

- Zur Absicherung der Netzwerk-Kommunikation zwischen DC und SAK-DC kommt **(m)TLS** zum Einsatz

In der folgenden Matrix sind die Sicherheitsmechanismen in einer Matrix aufgeschlüsselt. Hier kann eingesehen werden, welcher Sicherheitsmechanismus wann eingesetzt werden sollte oder muss.

Tabelle 1. Aktivierung / Erzwingung der Sicherheitsmechanismen je Endpunkt

Endpunkt	mTLS	OAuth 2.0	Basic Auth
`/token`	optional *(empfohlen)*	-	optional
`/de/evidence-offer`	optional *(empfohlen)*	erforderlich	optional
`/de/evidence`	optional *(empfohlen)*	erforderlich	optional

Die hierzu benötigten Zugangsdaten und Zertifikate müssen bei der Installation bereitgestellt und konfiguriert werden. Die benötigten Zugriffstoken für die OAuth 2.0 Authentifizierung müssen zur Laufzeit über die `/token`-Schnittstelle abgerufen werden.

HINWEIS

- Die Übertragung der Nachweisdaten erfolgt ausschließlich über den geschützten Weg zwischen SAK-DC und SAK-DP
- Der Anbindung an den EU-Kontext ist derzeit nicht Teil der Anwendung

Weitere technische Details können im Anschlusskonzept Data Consumer (AD-NOOTS-01) 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-DC. Je nach Mengengerüst können diese erheblich von den Mindestanforderungen abweichen.

Tabelle 2. 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-DC wird als ausführbare JAR-Datei (Java Archive) bereitgestellt.

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

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

```
scp sak-dc-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-dc` der der Linux Gruppe `sak-dc` angehört.

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

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

sudo chown root:sak-dc /opt/sak-dc/sak-dc-VERSION-
TAG.jar
sudo chmod 640 /opt/sak-dc/sak-dc-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-DC 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-dc-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-dc-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-DC erfolgreich gestartet`

3.2.2. Proxykonfiguration

Wird für die Kommunikation nach Außen ein Proxyserver benötigt, muss dieser beim Start der Anwendung konfiguriert werden, nicht in der `application.properties`.

```
java -Dhttps.proxyHost=IP/HOSTPROXY
-Dhttps.proxyPort=PROXYPORT
-Dhttp.nonProxyHosts=PROXYAUSNAHMEN -jar sak-dc-VERSION-
TAG.jar --spring.config.additional
-location=application.properties
```

3.2.3. 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 3. 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

HINWEIS

> Für HTTPS-Verbindungen und die mTLS-Authentifizierung, bei welchen der `SAK-DC` als Server agiert, dürfen die in der `application.properties` angegebenen Schlüsselspeicher keine Schlüssel und Zertifikate auf Basis der Brainpool-Kurven enthalten.

In jedem Fall ist die Nutzung von TLS 1.3 für die Kommunikation zwischen dem Data Consumer und dem SAK-DC 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** [...]“

3.2.4. 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 Consumers (SAK-DC) in der Systemumgebung nötig sind.

4.1.1. Kubernetes Laufzeitumgebung

SAK-DC 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 Consumers angelegt werden. Alternativ, kann der Namespace während der Installation der Anwendung erstellt werden.

Die Helm-Charts von SAK-DC 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-DC 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-DC 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-DC 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-DC verwendet standardmäßig die Konsole für Logausgaben. Das Log-Nachrichtenformat entspricht dem Elastic Common Schema (ECS).

5.2. Statusprüfung der Anwendungen

Der SAK-DC 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 4. Status-Endpunkte

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

5.3. Monitoring

Der SAK-DC 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 5. 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 6. Technisches Glossar für SAK-DC

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 7. Fachliches Glossar für SAK-DC

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-DC Transportprotokoll Einstellungen
# ----------------------------------------
# Zugewiesene NOOTS-Komponenten-ID des Sicheren
Anschlussknotens
sakdc.id =

# SAK-Client TLS-Auth
# 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 = sakdc-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 = sakdc-
client-cert.p12
#spring.ssl.bundle.jks.client.keystore.password =

# Truststore mit Zertifikaten, die der SAK bei
ausgehenden Aufrufen vertrauen darf.
# Optional; das System-Truststore des JRE wird
verwendet, falls nicht gesetzt.
#spring.ssl.bundle.jks.client.truststore.type = PKCS12
#spring.ssl.bundle.jks.client.truststore.location =
sakdc-client-truststore.p12
```

```
#spring.ssl.bundle.jks.client.truststore.password =

# 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 =
#spring.ssl.bundle.pem.iamsigncert.truststore.certificat
e = iam-sign-cert.crt
#spring.ssl.bundle.pem.iamsigncert.truststore.certificat
e = -----BEGIN CERTIFICATE-----....

# Truststore-Konfiguration für die NOOTS-Zuständigkeit-
/Verbindungstoken (RDN-Token) Validierung
spring.ssl.bundle.jks.rdnsigncert.truststore.type =
PKCS12
spring.ssl.bundle.jks.rdnsigncert.truststore.location =
rdn-sign-cert.p12
spring.ssl.bundle.jks.rdnsigncert.truststore.password =
#spring.ssl.bundle.pem.rdnsigncert.truststore.certificat
e = rdn-sign-cert.crt
#spring.ssl.bundle.pem.rdnsigncert.truststore.certificat
e = -----BEGIN CERTIFICATE-----....

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

# ---------------------------------------
# SAK-DC Anschlussprotokoll Einstellungen
# ---------------------------------------

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

# Optionale Basic-Auth Authentifizierung der
Schnittstellen des SAKs
# Schnittstellen des SAKs können laut der
Schnittstellenspezifikation optional via
# Basic-Auth gesichert werden. Sollte dies der Fall
sein, muss hier
# "basicauth.token.active" muss auf "true" gesetzt
```

```
werden, um den Token-Endpunkt abzusichern.
basicauth.token.active = true
# "basicauth.other.active" muss auf "true" gesetzt
werden, um die Evidence- und Evidence-Offer-Endpunkte
mit Basic-Auth abzusichern.
basicauth.other.active = true
# Username + Passwort muss angegeben werden, sobald
mindestens einer der oberen Werte auf "true" gesetzt
sind.
# Sofern aktiviert, werden fuer die Basic
Authentifizierung, keine leeren Credentials
unterstuetzt.
basicauth.username =
basicauth.password =

# 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 = sakdc-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 = sakdc-
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-
dc1.example.com
#server.ssl.server-name-bundles[0].bundle = sakdc1
#spring.ssl.bundle.jks.sakdc1.keystore.type = PKCS11
#spring.ssl.bundle.jks.sakdc1.keystore.password =
#spring.ssl.bundle.jks.sakdc1.key.alias = sakdc1-server-
crt
#spring.ssl.bundle.jks.sakdc1.options.enabled-protocols
= ${sakdc.server.ssl.enabled-protocols}
#spring.ssl.bundle.jks.sakdc1.options.ciphers =
${sakdc.server.ssl.enabled-ciphers}

# SAK-Server TLS Client-Auth
# Einkommentieren, falls DCs ohne Client-Zertifikat den
SAK aufrufen
#server.ssl.client-auth = none
# Truststore mit vertrauenswürdigen Client-Cert-CAs
```

```
# Auskommentieren, falls DCs ohne Client-Zertifikat den
SAK aufrufen
spring.ssl.bundle.jks.server.truststore.type = PKCS12
spring.ssl.bundle.jks.server.truststore.location =
sakdc-server-truststore.p12
spring.ssl.bundle.jks.server.truststore.password =

# SAK-Server TLS deaktivieren (nicht empfohlen, bitte
sicherstellen, dass DCs nur über verschlüsselten Wege
den SAK aufrufen).
# ACHTUNG: 'server.port' wird standardmäßig auf 8080
gesetzt.
# In diesem Fall, alle 'spring.ssl.bundle.jks.server.*'
Properties auskommentieren.
#spring.profiles.active = plain-http

# ----------------------------------------
# SAK-DC Allgemeine Anwendungskonfiguration
# ----------------------------------------

# Arbeitsordner des Webservers
# Dieser muss für die Anwendung lesbar und beschreibbar
sein.
# Andere User und Anwendungen sollen nicht darauf
zugreifen können.
server.tomcat.basedir = /var/run/sak-dc

# 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-DC
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

# Rate-Limiting
# Anzahl der Requests je Periode die erlaubt sind
resilience4j.ratelimiter.instances.globalRateLimiter.lim
itForPeriod = 50
# Periode in der die obige Anzahl an Requests erlaubt
sind.
# Gültige Einheiten sind Sekunden, 's' und Nanosekunden,
'n'
resilience4j.ratelimiter.instances.globalRateLimiter.lim
itRefreshPeriod = 1s
```

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

```
# Standardwerte für das Chart des Sicheren
Anschlussknotens für Datenkonsumenten
# 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. Weitere Informationen:
https://kubernetes.io/docs/concepts/containers/images/
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.
Weitere Informationen:
https://kubernetes.io/docs/concepts/security/service-
accounts/
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. Weitere
Informationen:
https://kubernetes.io/docs/concepts/overview/working-
with-objects/annotations/
podAnnotations: { }
```

```yaml
# podLabels -- definiert Kubernetes-Labels, die auf die
Pods angewendet werden. Weitere Informationen:
https://kubernetes.io/docs/concepts/overview/working-
with-objects/labels/
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: [ ]

# sakdc -- enthält die Hauptkonfiguration für den
Sicheren Anschlussknoten (Sak DC).
sakdc:
  # sakdc.id -- ist die eindeutige ID des Sak DC.
  id: "b233caa1-f582-414a-aca1-ad3da4168de5"

  # sakdc.mtlsAuthentication -- konfiguriert die mTLS-
Authentifizierung für ausgehende Verbindungen.
  mtlsAuthentication:
    keystore:
      # sakdc.mtlsAuthentication.keystore.data --
enthält die einzeilig Base64-kodierten Keystore-Bytes.
Wenn gesetzt, wird secretRef überschrieben.
      data: ""
      # sakdc.mtlsAuthentication.keystore.secretRef --
ist die Referenz auf ein bestehendes Secret mit Client
Zertifikat zu Diensten wie IAMB, RDN, etc.
      secretRef: "sakdc-mtls-keystore"
      # sakdc.mtlsAuthentication.keystore.secretRefKey
-- ist der Key innerhalb des Secrets, der den Keystore
enthält.
      secretRefKey: "sakdc-mtls-keystore.p12"
      # sakdc.mtlsAuthentication.keystore.password --
ist das Passwort für das Client-Zertifikat. Ein leerer
String ("") ist ein gültiges Passwort.
      # Um ein bestehendes Secret mittels
'passwordSecretRef' zu nutzen, muss 'password: null'
gesetzt werden.
      password: null
      #
sakdc.mtlsAuthentication.keystore.passwordSecretRef --
ist die Referenz auf ein Secret, das das Passwort für
das Client-Zertifikat enthält.
      passwordSecretRef: "sakdc-mtlspassword"
      #
sakdc.mtlsAuthentication.keystore.passwordSecretRefKey
```

```
-- ist der Key innerhalb des Secrets, der das Passwort
enthält.
      passwordSecretRefKey: "password"

  # sakdc.clientTruststore -- Konfiguriert den
Truststore für ausgehende TLS-Verbindungen.
  clientTruststore:
    # sakdc.clientTruststore.data -- Enthält die
einzeiligen Base64-kodierten Keystore-Bytes. Wenn
gesetzt, wird secretRef überschrieben.
    data: ""
    # sakdc.clientTruststore.enabled -- Aktiviert die
Verwendung eines benutzerdefinierten Truststores.
    enabled: false
    # sakdc.clientTruststore.secretRef -- Ist die
Referenz auf ein Secret, das den Truststore enthält.
    secretRef: "sakdc-client-truststore"
    # sakdc.clientTruststore.secretRefKey -- Ist der Key
innerhalb des Secrets, der den Truststore enthält.
    secretRefKey: "sakdc-client-truststore.p12"
    # sakdc.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
    # sakdc.clientTruststore.passwordSecretRef -- Ist
die Referenz auf ein Secret, das das Passwort für den
Truststore enthält.
    passwordSecretRef: "sakdc-clienttruststorepassword"
    # sakdc.clientTruststore.passwordSecretRefKey -- Ist
der Key innerhalb des Secrets, der das Passwort enthält.
    passwordSecretRefKey: "password"

  # sakdc.accessToken -- Konfiguriert die Überprüfung
von AccessTokens.
  accessToken:
    # sakdc.accessToken.roleEvidenceOffer -- Definiert
die Rollen, die erforderlich sind, um das
Nachweisangebot abzurufen.
    roleEvidenceOffer: "noots.nachweisangebot"
    # sakdc.accessToken.roleGetIdNr -- Definiert die
Rollen, die erforderlich sind, um die ID-Nummer
abzurufen.
    roleGetIdNr: "noots.identifikationsnummer"
    # sakdc.accessToken.roleGetEvidenceDataprovider --
```

```
Definiert die Rollen, die erforderlich sind, um
Nachweise abzurufen.
    roleGetEvidenceDataprovider:
"noots.verbindungsparameter,
noots.abstrakteberechtigung, nachweis.national"
    # sakdc.accessToken.algorithms -- Gibt den
Algorithmus an, mit dem das Token signiert sein muss.
    algorithms: "ES256"
    # sakdc.accessToken.rolesclaimname -- Ist der Name
des Claims im Token, der die Rollen enthält.
    rolesclaimname: "scope"
    # sakdc.accessToken.issuer -- Ist der erwartete
Aussteller des Tokens.
    issuer:
    # sakdc.accessToken.audiences -- Definiert die
erwartete Audience des Tokens.
    audiences: "urn:regmo:components"
    # sakdc.accessToken.iambTruststore -- Konfiguriert
den Truststore für die Kommunikation mit dem IAMB.
    iambTruststore:
      # sakdc.accessToken.iambTruststore.enabled --
Aktiviert die Verwendung eines benutzerdefinierten
Truststores für den IAMB.
      enabled: false
      # sakdc.accessToken.iambTruststore.cabundle --
Enthält die CA-Zertifikate als PEM. Sie ersetzen das
secretRef und zugehörigen secretRefKey.
      cabundle: ""
      # sakdc.accessToken.iambTruststore.secretRef --
Ist die Referenz auf ein Secret, das den Truststore für
den IAMB enthält.
      secretRef: "sakdc-iamb-truststore"
      # sakdc.accessToken.iambTruststore.secretRefKey --
Ist der Key innerhalb des Secrets, der den Truststore
für den IAMB enthält.
      secretRefKey: "sakdc-iamb-cabundle.pem"
    # sakdc.accessToken.rdnTruststore -- Konfiguriert
den Truststore für die Kommunikation mit dem RDN.
    rdnTruststore:
      # sakdc.accessToken.rdnTruststore.enabled --
Aktiviert die Verwendung eines benutzerdefinierten
Truststores für den RDN.
      enabled: false
      # sakdc.accessToken.rdnTruststore.cabundle --
Enthält die CA-Zertifikate als PEM. Sie ersetzen das
secretRef und zugehörigen secretRefKey.
      cabundle: ""
```

```
      # sakdc.accessToken.rdnTruststore.secretRef -- Ist
die Referenz auf ein Secret, das den Truststore für den
RDN enthält.
      secretRef: "sakdc-rdn-truststore"
      # sakdc.accessToken.rdnTruststore.secretRefKey --
Ist der Key innerhalb des Secrets, der den Truststore
für den RDN enthält.
      secretRefKey: "sakdc-rdn-cabundle.pem"

  # sakdc.ratelimit -- Konfiguriert das Rate Limiting
für die Endpunkte.
  ratelimit:
    # sakdc.ratelimit.period -- Definiert den Zeitraum,
in dem die konfigurierte Anzahl an Requests nicht
überschritten werden darf.
    period: 1s
    # sakdc.ratelimit.requestForPeriod -- Definiert die
Anzahl an Requests, die je konfigurierter Zeiteinheit
erlaubt sind.
    requestForPeriod: 50

# credentials -- Konfiguriert die Basic-
Authentifizierung für bestimmte Endpunkte.
credentials:
  basicAuth:
    # credentials.basicAuth.token -- Aktiviert die
Basic-Authentifizierung für den /token-Endpunkt.
    token: true
    # credentials.basicAuth.other -- Aktiviert die
Basic-Authentifizierung für andere Endpunkte wie
/de/evidence-offer.
    other: true
    # credentials.basicAuth.username -- Ist der
Benutzername für die Basic-Authentifizierung.
@erforderlich, wenn token oder other aktiviert sind.
    username: ""
    # credentials.basicAuth.password -- Ist das Passwort
für die Basic-Authentifizierung. @erforderlich, wenn
token oder other aktiviert sind.
    password: ""

# resources -- Definiert die Ressourcenanforderungen und
-limits für den Container.
resources:
  # resources.limits -- Definiert die maximalen
Ressourcen, die der Container verwenden darf.
  limits:
```

```
    cpu: 1000m
    memory: 1024Mi
  # resources.requests -- Definiert die minimalen
Ressourcen, die der Container benötigt.
  requests:
    cpu: 500m
    memory: 1024Mi

# proxy -- Konfiguriert einen HTTP(S)-Proxy für
ausgehende Verbindungen.
proxy:
  # proxy.enabled -- Aktiviert oder deaktiviert die
Verwendung eines Proxys.
  enabled: false
  # proxy.ip -- Ist die IP-Adresse des Proxy-Servers.
  ip: "127.0.0.1"
  # proxy.port -- Ist der Port des Proxy-Servers.
  port: "8080"
  # proxy.noProxy -- Enthält eine Liste von Hosts, die
direkt erreichbar sind und den Proxy umgehen.
  noProxy: ""

# networkPolicy -- Konfiguriert die Kubernetes
NetworkPolicy für die Pods.
networkPolicy:
  # networkPolicy.enabled -- Aktiviert die Anwendung
einer NetworkPolicy auf die Pods.
  enabled: true
  # networkPolicy.ingress -- Definiert die eingehenden
Verbindungen zu den Pods.
  ingress:
    - ports:
        - protocol: TCP
          port: 8080
      from:
        - podSelector:
            matchLabels:
              app.kubernetes.io/name: rke2-ingress-nginx
          namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-system
  # networkPolicy.egress -- Definiert die ausgehenden
Verbindungen von den Pods.
  egress:
    - to:
        - namespaceSelector:
            matchLabels:
```

```
              kubernetes.io/metadata.name: kube-system
          podSelector:
            matchLabels:
              k8s-app: kube-dns
      ports:
        - protocol: TCP
          port: 53
        - protocol: UDP
          port: 53
    - ports:
        - protocol: TCP
          port: 443
  # networkPolicy.ingressPrometheus -- Definiert die
eingehenden Verbindungen für Prometheus zum Management
Port.
  ingressPrometheus:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: kube-
prometheus-stack
      ports:
        - protocol: TCP
          port: 9000

# service -- Definiert den Kubernetes-Service für die
Anwendung.
service:
  # service.create -- Gibt an, ob ein Service erstellt
werden soll.
  create: true
  # service.type -- Legt den Typ des Services fest, z.
B. ClusterIP, NodePort oder LoadBalancer.
  type: ClusterIP
  # service.port -- Ist der Port, unter dem die
Anwendung erreichbar ist.
  port: 8080
  # service.managementPort -- Ist der Port für
Management-Endpunkte.
  managementPort: 9000
  # service.exposeDebuggingPort -- Legt fest, ob ein
Debugging-Port exponiert werden soll.
  exposeDebuggingPort: false

rdn:
  # rdn.baseUrl -- RDN Basis-URL
  baseUrl:
```

```
  # rdn.hostname -- RDN Hostname
  hostname:

iamb:
  # iamb.baseUrl -- IAM-B Basis-URL
  baseUrl:
  # iamb.hostname -- IAM-B Hostname
  hostname:

# Dieser Block dient zur Einrichtung des Ingress.
Weitere Informationen finden Sie hier:
https://kubernetes.io/docs/concepts/services-
networking/ingress/
ingress:
  # ingress.enabled -- Aktiviert die Erstellung eines
Ingress-Objekts.
  enabled: false
  # ingress.className -- Legt den Klassennamen für den
Ingress-Controller fest.
  className: ""
  # ingress.annotations -- Definiert Annotationen, die
dem Ingress hinzugefügt werden.
  annotations: { }
  # ingress.hosts -- Definiert die Hosts, für die der
Ingress gilt.
  hosts:
    - host: sicherer-anschlussknoten-dc.local
      paths:
        - path: /
          pathType: ImplementationSpecific
  # ingress.tls -- Konfiguriert die TLS-Einstellungen
für den Ingress.
  tls: [ ]

# servicemonitor -- Konfiguriert die Überwachung der
Anwendung durch Prometheus.
servicemonitor:
  # servicemonitor.enabled -- Aktiviert die Erstellung
eines ServiceMonitors für Prometheus.
  enabled: false

# livenessProbe -- Konfiguriert die Liveness-Probe für
den Pod.
livenessProbe:
  httpGet:
    path: /actuator/health/liveness
    port: 9000
```

```
  initialDelaySeconds: 30

# readinessProbe -- Konfiguriert die Readiness-Probe für
den Pod.
readinessProbe:
  httpGet:
    path: /actuator/health/readiness
    port: 9000
  initialDelaySeconds: 30

# startupProbe -- Konfiguriert die Startup-Probe für den
Pod.
startupProbe:
  httpGet:
    path: /actuator/health/liveness
    port: 9000
  periodSeconds: 30
  failureThreshold: 30

# volumes -- Definiert zusätzliche Volumes, die im Pod
verwendet werden.
volumes: [ ]

# volumeMounts -- Definiert zusätzliche VolumeMounts,
die im Pod verwendet werden.
volumeMounts: [ ]

# nodeSelector -- Dient der Einschränkung des
Deployments auf bestimmte Clusterknoten.
nodeSelector: { }

# affinity -- Konfiguriert die Affinität und Anti-
Affinität für das Deployment.
affinity: { }

# tolerations -- Konfiguriert die Tolerations für das
Deployment, um auf tainted Nodes zu laufen.
tolerations: [ ]

# clientCert -- Konfiguriert die Erstellung und
Verwaltung von selbstsignierten Zertifikaten.
clientCert:
  # clientCert.enabled -- Aktiviert die Erstellung eines
selbst signierten Zertifikats.
  enabled: false
  # clientCert.metadata -- Definiert Metadaten für das
Zertifikat.
```

```
  metadata:
    name:
  # clientCert.annotations -- Konfiguriert Annotationen
für das Zertifikat.
  annotations:
    enabled: false
    annotations: { }
  # clientCert.subject -- Definiert das Subject des
Zertifikats.
  subject:
  # clientCert.privateKey -- Konfiguriert den privaten
Schlüssel für das Zertifikat.
  privateKey:
    # clientCert.privateKey.algorithm -- Gibt den
Algorithmus für den privaten Schlüssel an.
    algorithm: ECDSA
    # clientCert.privateKey.size -- Gibt die Größe des
privaten Schlüssels an.
    size: 256
  # clientCert.usages -- Definiert die Verwendungszwecke
des Zertifikats.
  usages:
    - "client auth"
  # clientCert.issuerRef -- Konfiguriert den Issuer für
das Zertifikat.
  issuerRef:
  # clientCert.commonName -- Definiert den gemeinsamen
Namen für das 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 8. 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
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	

Name der Quelle	Link	Version
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