OpenLDAP in der Praxis

Das Handbuch für Administratoren

DIE LESEPROBE

3 Installation des ersten OpenLDAP

In der ersten Auflage dieses Buches haben wir die Installation noch für zwei Distributionen beschrieben, da bei den verschiedenen Distributionen ein paar Unterschiede bestanden. Dieses Mal greifen wir auf die Pakete der Firma Symas zurück, daher sind jetzt, bei allen unterstützten Distributionen, die Pfade und Dateinamen identisch, sodass wir hier die Installation distributionsunabhängig beschreiben können.

Hinweis: Bei den Paketen ist der Pfad, in dem die Dateien der Pakete abgelegt werden, immer /opt/symas/. Der Grund dafür ist: So kollidieren die Dateien aus den Symas-Paketen nicht mit den Dateien aus den Paketen der Distribution, falls Sie die Pakete noch installiert haben.

An dieser Stelle gehen wir auch noch einmal auf die Unterschiede zwischen der statischen und dynamischen Konfiguration ein. Im Rest des Buches werden wir dann ausschließlich die dynamische Konfiguration nutzen.

Der Grund ist der, dass die statische Konfiguration auf der Website *https://www.openldap.org* als *deprecated* angegeben wird. Das bedeutet, dass die Möglichkeit der Konfiguration über eine einfache ASCII-Textdatei leider auf Dauer nicht mehr möglich sein wird. In komplexen Installationen mit mehreren Providern ist die Konfiguration über die statische Konfiguration auch nicht mehr zeitgemäß.

Bei der dynamischen Konfiguration werden sowohl die Grundkonfiguration als auch alle weiteren Änderungen über LDIF-Dateien in einer speziellen Datenbank im LDAP gespeichert, und der LDAP-Server verwaltet sich quasi selbst.

3.0.1 Die statische Konfiguration

Bei der statischen Konfiguration handelt es sich um die klassische Variante der Konfiguration. Alle Konfigurationsparameter werden in einer Konfigurationsdatei abgelegt. Immer, wenn Sie eine Änderung durchgeführt haben, ist es notwendig, dass Sie den LDAP-Server neu starten, damit die Änderung wirksam wird. Wenn Sie mehrere LDAP-Server einsetzen, müssen Sie die Konfiguration immer auf allen LDAP-Servern anpassen.

Nicht nur die grundlegende Konfiguration wird in der statischen Konfiguration in der Konfigurationsdatei abgelegt, sondern später auch alle ACLs und die Konfiguration der eventu-

ell verwendeten Overlays (bei Overlays handelt es sich um eine Art Plug-in, das die Funktion des LDAP-Servers erweitert; mehr zum Thema Overlays finden Sie in Kapitel 10, «Einsatz von Overlays»).

Sie können die Konfiguration Ihres LDAP-Servers auch erst statisch beginnen und später auf die dynamische Konfiguration umstellen.

Da die Datei slapd.conf bei allen Distributionen denselben Inhalt hat, möchten wir Ihnen an dieser Stelle die einzelnen Zeilen der Konfiguration erklären. So können Sie in diesem Kapitel den Inhalt der statischen Konfiguration noch sehr gut mit der dynamischen Konfiguration vergleichen. In Listing 3.1 sehen Sie den Inhalt der Datei:

Listing 3.1 Die slapd.conf

```
#
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include         /opt/symas/etc/openldap/schema/core.schema
include         /opt/symas/etc/openldap/schema/cosine.schema
include         /opt/symas/etc/openldap/schema/nis.schema
include         /opt/symas/etc/openldap/schema/inetorgperson.schema

pidfile         /var/symas/run/slapd.pid
argsfile        /var/symas/run/slapd.args

# Read slapd.conf(5) for possible values
loglevel        256

# Load dynamic backend modules:
modulepath      /opt/symas/lib/openldap
moduleload      back_mdb.la
moduleload      argon2.la

# The maximum number of entries that is returned for a search operation
sizelimit 500

# The tool-threads parameter sets the actual amount of cpu's that is used
# for indexing.
tool-threads 1
password-hash {ARGON2}

########################################
# Specific Directives for database #1, of type hdb:
# Database specific directives apply to this databasse until another
# 'database' directive occurs
database        mdb
maxsize         1073741824
# The base of your directory in database #1
suffix          "dc=example,dc=net"

# rootdn directive for specifying a superuser on the database.
rootdn          "cn=admin,dc=example,dc=net"
#rootpw          "geheim"
rootpw          "{ARGON2}$argon2i$v=19$m=4096,t=3,p=1$c2Fs..."
```

```
# Where the database file are physically stored for database #1
directory       "/var/symas/openldap-data"

# Indexing options for database #1
index           objectClass eq

# Save the time that the entry gets modified, for database #1
lastmod         on


# The userPassword by default can be changed
# by the entry owning it if they are authenticated.
# Others should not be able to see it, except the
# admin entry below
# These access lines apply to database #1 only

access to attrs=userPassword,shadowLastChange
        by anonymous auth
        by self write
        by * none

access to *
        by * read

access to dn.base="" by * read
```

Die Konfigurationsdatei besteht aus zwei Teilen: Der erste Teil oberhalb der Linie aus Hashes ist der globale Teil, der für alle Datenbanken, die der OpenLDAP bereitstellt, relevant ist. Der Teil unterhalb betrifft immer eine bestimmte Datenbank. Der Parameter *database mdb* markiert dabei den Beginn einer neuen Datenbank.

- Als Erstes werden die Schematadateien eingebunden, die Reihenfolge ist dabei sehr wichtig. Beim Start des LDAP-Servers werden die Dateien eine nach der anderen abgearbeitet. Da durch die Möglichkeit der Vererbung von Attributen eine Abhängigkeit zwischen den einzelnen Schemata besteht, müssen Sie immer wissen, welche Objektklasse eventuell Attribute einer anderen Objektklasse nutzt. Die Standardobjektklassen, die hier eingebunden sind, befinden sich schon in der richtigen Reihenfolge. Wenn Sie ein eigenes Schema mit eigenen Objektklassen erstellen, ist es wichtig, dass Sie am Anfang Ihres Schemas etwaige Abhängigkeiten kommentieren.
- In den Zeilen *pidfile /var/symas/run/slapd.pid* und *argsfile /var/symas/run/slapd.args* werden die Prozess-ID und die Argumente, die beim Start des OpenLDAP-Servers verwendet werden, abgelegt.
- Das *loglevel* legt fest, welche Ereignisse geloggt werden. Dabei werden die Zahlen nicht einfach hochgezählt, sondern die einzelnen Werte werden binär übergeben. Am Anfang ist ein *loglevel 256 (stats)* eine gute Einstellung, denn da werden alle Zugriffe geloggt. Welche Loglevel Sie verwenden können, finden Sie in der Manpage zur slapd.conf. Im späteren produktiven Betrieb stellt jede Art von Logging immer einen Flaschenhals da. Es ist daher sinnvoll, nachdem der OpenLDAP eingerichtet wurde, das Loglevel auf den Wert *none* zu setzen, dann werden nur noch systemkritische Fehler ins Log geschrie-

ben. Setzen Sie das Loglevel auf *0*, werden auch diese Meldungen nicht mehr ins Log geschrieben.

- Die Parameter *modulepath* und *moduleload* werden immer dann benötigt, wenn Sie zusätzliche Module für zusätzliche Funktionen nutzen wollen. Bei den Symas-Paketen werden alle Overlays über Module bereitgestellt und müssen einzeln geladen werden.
- Der Parameter *sizelimit 500* legt fest, wie viele Antworten bei einer Suche zurückgegeben werden. Gerade wenn Sie einen großen LDAP-Baum mit mehreren Tausend Objekten pflegen, kann die Antwort den Server stark belasten. Deshalb diese Grenze. Über Filter können Sie die Suche einschränken. Die Übersteuerung der Einschränkung der zurückgegebenen Objekte bei der Suche muss immer direkt in der Datenbank definiert werden, für die sie zutreffen soll.
- Wenn Sie sehr viele Indexdatenbanken einsetzen, um die Suche nach Attributen oder Objekten zu beschleunigen, kann es sinnvoll sein, mehr als nur eine CPU für das Indizieren zu verwenden. Testen Sie, ob eine weitere CPU für Sie Vorteile bringt.
- Wir werden hier im Buch nur noch *ARGON2* als Passworthash nutzen. Über den Parameter *password-hash* legen Sie den neuen Passworthash fest. Mehr zum Thema *ARGON2* finden Sie unter *https://de.wikipedia.org/wiki/Argon2*.
- Mit *database mdb* beginnt nicht nur der Datenbankteil, sondern Sie legen auch den Datenbanktyp fest. Seit einiger Zeit wird nur noch der Datenbanktyp *mdb* empfohlen, alle anderen alten Datenbanktypen werden in Zukunft nicht mehr unterstützt.
- Der *suffix* legt die oberste Ebene Ihres LDAP-Baums fest.
- Beim *rootDN* handelt es sich um den Hauptadministrator, der immer alle Rechte hat und nie auf irgendeine Art und Weise beschränkt werden kann. Der Benutzer wird nicht in der Datenbank angelegt.
- Das *rootpw* ist das Passwort für den *rootDN*. Halten Sie dieses Passwort immer unter Verschluss. Mit diesem Passwort können sämtliche Änderungen am LDAP-Baum vorgenommen werden. In der Beispieldatei sehen Sie, dass hier schon *ARGON2* als Passorthash genutzt wird. Anders als bei der Verwendung anderer Passwordhashes nutzen Sie hier das Kommando `echo -n "mein-geheimes-pw" | argon2 "saltsaltsaltsalt" -e`, um den Passworthash zu erzeugen.
- Mit dem Parameter *directory* legen Sie fest, in welchem Verzeichnis die Datenbankdateien abgelegt werden. Für jede Datenbank, die Sie mit dem LDAP-Server verwalten wollen, benötigen Sie ein eigenes Verzeichnis.
- Über den Parameter *index* werden die Indexdatenbanken für diese LDAP-Datenbank festgelegt. Später werden wir noch näher auf die Indexeinträge eingehen.
- *Lastmod on* speichert die Zeit der letzten Änderung für jedes Objekt in der Datenbank.
- Am Schluss folgen die ACLs für die Zugriffe. Auf die ACLs gehen wir in Kapitel 9, «Berechtigungen mit ACLs», näher ein.

Tipp: Haben Sie bis jetzt Ihren LDAP-Server über die statische Konfiguration verwaltet, können Sie diese mit dem Kommando `slaptest -F /opt/symas/etc/openldap/slapd.d -f /etc/ldap/slapd.conf` in die dynamische Konfiguration umwandeln.

3.0.2 Die dynamische Konfiguration

Bei der dynamischen Konfiguration werden alle Einstellungen des LDAP-Servers in einer eigenen Datenbank im LDAP abgelegt: Sowohl die Grundkonfiguration, die Erweiterung des Funktionsumfang durch Overlays als auch die ACLs werden mittels LDIF-Dateien in die Datenbank eingespielt. Kommentare können Sie in dieser Datenbank nicht ablegen, sodass Sie sämtliche Änderungen und Einstellungen extern dokumentieren müssen.

Der große Vorteil der dynamischen Konfiguration ist, dass Sie hier den LDAP-Server nach einer Änderung nicht neu starten müssen; die Änderung ist sofort wirksam, nachdem Sie die LDIF-Datei eingespielt haben. Auch Ihre ACLs passen Sie dann immer über LDIF-Dateien an; auch sie sind nach dem Einspielen sofort wirksam. Wenn Sie mit grafischen Werkzeugen arbeiten wollen, dann achten Sie darauf, ob das Werkzeug die dynamische Konfiguration bearbeiten kann. Wir werden in Kapitel 7, «Grafische Werkzeuge», noch darauf zu sprechen kommen.

Wenn Sie mehrere LDAP-Server einsetzen und diese vielleicht auch noch an unterschiedlichen Standorten stehen und Sie häufig Änderungen an der Konfiguration oder den ACLs vornehmen, dann ist die dynamische Konfiguration auf jeden Fall der bessere Weg. Mit der neuen Version 2.5 und 2.6 funktioniert jetzt auch die Replikation der dynamischen Konfiguration fehlerfrei. So können Sie Änderungen an der Konfiguration auf einem Server einspielen, und die Änderung wird auf alle anderen OpenLDAP-Servern repliziert. Durch den geänderten Replikationsprozess in den neuen Versionen können Sie so einen Multiprovider-Cluster aufbauen und die Konfiguration für den gesamten Cluster mit einer LDIF-Datei anpassen.

3.1 Installation der Symas-Pakete

Eine Besonderheit der Symas-Pakete sei hier gleich am Anfang erwähnt: Der Dienst startet in der Standardeinstellung, nach der Installation, grundsätzlich mit der Benutzerkennung *root*. Das ist aber keine gute Idee und soll hier auch sofort geändert werden.

Aber als Erstes werden jetzt die Pakete installiert. Dafür benötigen Sie die Daten, um das Repository auf Ihrem System einzutragen. Damit Sie nach dem Installieren der Pakete auch gleich die Zugriffsrechte für die verwendeten Verzeichnisse anpassen, sehen Sie in Listing 3.2 ein Skript, mit dem Sie nicht nur die Pakete installieren können, sondern auch gleich einen Benutzer für den Dienst anlegen und die Berechtigungen im Dateisystem setzen:

Listing 3.2 Skript zur Installation der Symas-Pakete

```
#!/bin/bash
DEBIAN_FRONTEND=noninteractive apt install -yq gnupg2 argon2 python3-ldap
wget -O- https://repo.symas.com/repo/gpg/RPM-GPG-KEY-symas-com-signing-key\
     | gpg --dearmor | tee /etc/apt/trusted.gpg.d/symas-com-gpg.gpg \
     > /dev/null
echo "deb [arch=amd64] https://repo.symas.com/repo/deb/main/release26 \
     bullseye main" | tee -a /etc/apt/sources.list.d/symas26.list
```

```
apt update -y
groupadd -r  openldap
useradd -r -g openldap -d /opt/symas/ openldap
DEBIAN_FRONTEND=noninteractive apt install -yq symas-openldap-clients \
     symas-openldap-server
chown openldap:openldap /var/symas/openldap-data/
chmod 770 /var/symas/openldap-data/
chown openldap:openldap /var/symas/run
chmod 770 /var/symas/run
mkdir -m 770 /opt/symas/etc/openldap/slapd.d
chown openldap:openldap /opt/symas/etc/openldap/slapd.d
```

Nachdem die Pakete installiert sind, benötigen Sie jetzt noch ein neues Skript für den *systemd*, um auch den *slapd* mit dem gerade angelegten Benutzer und der Gruppe zu starten. Das Skript aus Listing 3.3 können Sie so übernehmen:

Listing 3.3 Serviceskript zum Starten des slapd

```
[Unit]
Description=Symas OpenLDAP Server Daemon
After=network-online.target
Documentation=man:slapd
Documentation=man:slapd-config
Documentation=man:slapd-mdb

[Service]
Type=notify
EnvironmentFile=/etc/default/symas-openldap
ExecStart=/opt/symas/lib/slapd -d 0 -h ${SLAPD_URLS}  \
          -u ${SLAPD_USER} -g ${SLAPD_GROUP} \
          ${STAT_DYN} ${SLAPD_CONF}
[Install]
WantedBy=multi-user.target
```

Auf einem Debian-System kopieren Sie das Service-Skript in das Verzeichnis /lib/systemd/system/, auf einem Redhat-System in das Verzeichnis /usr/lib/systemd/system/.

Die hier verwendeten Variablen werden in dem Skript /etc/default/symas-openldap definiert. Wenn Sie eine andere Distribution als Debian nutzen, legen Sie das Verzeichnis /etc/default an oder ändern Sie den Pfad passend zu Ihrer Distribution ab.

Fehlt nur noch die Datei symas-openldap mit den vordefinierten Variablen. Den Inhalt der Datei sehen Sie in Listing 3.4:

Listing 3.4 Konfiguration der Variablen für den Start

```
SLAPD_URLS="ldap:/// ldapi:/// ldaps:///"
SLAPD_OPTIONS=""
SLAPD_USER="openldap"
SLAPD_GROUP="openldap"
# Select static "-f" or dynamic "-F"
STAT_DYN="-F"
# To use static configuration
#SLAPD_CONF="/opt/symas/etc/openldap/slapd.conf"
# To use dynamic configuration
SLAPD_CONF="/opt/symas/etc/openldap/slapd.d"
```

Hier wird gleich die dynamische Konfiguration eingestellt, und es werden alle drei möglichen Protokolle für den Zugriff auf den LDAP-Server aktiviert. Auch sehen Sie hier, dass der Benutzer und die Gruppe für den Start des Dienstes festgelegt wird.

Die dynamische Konfiguration

Nachdem Sie im vorigen Abschnitt gesehen haben, wie die Konfiguration über die Datei slapd.conf durchgeführt wird, zeigen wir Ihnen in diesem Abschnitt die dynamische Konfiguration.

Die dynamische Konfiguration wird in Form von LDIF-Dateien im Verzeichnis /opt/symas/etc/openldap/slapd.d abgelegt. In dem Verzeichnis finden Sie, nachdem Sie die Konfiguration eingespielt haben, weitere Unterverzeichnisse und LDIF-Dateien, denn die Konfiguration wird nicht in einem großen LDIF-File abgelegt, sondern setzt sich aus mehreren LDIF-Dateien zusammen. Die LDIF-Dateien sind dort nach Datenbanken und Inhalten in unterschiedlichen Verzeichnissen abgelegt. Beim Neustart des LDAP-Servers wird die Konfiguration aus den LDIF-Dateien geladen.

Jetzt stehen wir aber erst einmal vor dem Henne-Ei-Problem. Der *slapd* startet nicht ohne Konfiguration, und die Konfiguration wird über die ldap-Kommandos in den LDAP geschrieben. Aber der Dienst läuft nicht, weil keine Konfiguration da ist.

Aber zum Glück lässt sich das Problem recht einfach lösen. Mithilfe des Kommandos `slapadd` können Sie Daten in bestehende Datenbanken schreiben oder neue Datenbanken erstellen, ohne dass der *slapd* läuft. Sie benötigen lediglich eine LDIF-Datei, in der die Grundkonfiguration definiert ist, und können diese dann mit `slapadd` einspielen.

Warum klappt das? Die `slap`-Kommandos greifen direkt auf die Datenbankdateien zu und gehen nicht den Weg über das LDAP-Frontend *slapd*. Aus diesem Grund ist es auch wichtig, dass bei der Verwendung von `slapadd` der Dienst nicht läuft, da Sie sonst direkt in geöffnete Datenbankdateien schreiben würden. Daher können wir auch auf diesem Weg die Konfiguration in den OpenLDAP einspielen. Doch bevor wir die Konfiguration einspielen, werfen wir erst einmal einen Blick auf eine Konfigurationsdatei für die Grundkonfiguration und schauen uns einmal an, wo wir, parallel zur statischen Konfiguration, die Informationen finden. Listing 3.5 zeigt die Konfigurationsdatei im LDIF-Format:

Listing 3.5 LDIF-Datei mit den Grundeinstellungen

```
dn: cn=config
objectClass: olcGlobal
cn: config
olcLogLevel: sync
olcLogLevel: stats
olcPidFile: /var/symas/run/slapd.pid
olcArgsFile: /var/symas/run/slapd.args
olcToolThreads: 1

dn: cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema

dn: cn=module{0},cn=config
```

```
objectClass: olcModuleList
cn: module{0}
olcModulePath: /opt/symas/lib/openldap
olcModuleLoad: back_mdb
olcModuleLoad: back_monitor
olcModuleLoad: argon2.la

include: file:///opt/symas/etc/openldap/schema/core.ldif
include: file:///opt/symas/etc/openldap/schema/cosine.ldif
include: file:///opt/symas/etc/openldap/schema/nis.ldif
include: file:///opt/symas/etc/openldap/schema/inetorgperson.ldif

dn: olcDatabase={-1}frontend,cn=config
objectClass: olcDatabaseConfig
objectClass: olcFrontendConfig
olcDatabase: {-1}frontend
olcSizeLimit: 500
olcAccess: {0}to *
  by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth manage
  by * break
olcAccess: {1}to dn=""  by * read
olcAccess: {2}to dn.base="cn=subschema"  by * read
olcPasswordHash: {ARGON2}

dn: olcDatabase={0}config,cn=config
objectClass: olcDatabaseConfig
olcDatabase: {0}config
olcRootDN: cn=admin,cn=config
olcRootPW: {ARGON2}$argon2i$v=19$m=4096,t=3,p=1$cX....
olcAccess: {0}to *
  by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth manage

dn: olcDatabase={1}monitor,cn=config
objectClass: olcDatabaseConfig
olcDatabase: {1}monitor
olcAccess: {0}to dn.subtree="cn=monitor"
  by dn.exact=cn=admin,cn=config read
  by dn.exact=cn=admin,dc=example,dc=net read
  by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth read

dn: olcDatabase={2}mdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcMdbConfig
olcDatabase: {2}mdb
olcSuffix: dc=example,dc=net
olcRootDN: cn=admin,dc=example,dc=net
olcRootPW: {ARGON2}$argon2i$v=19$m=4096,t=3,p=1$cM....
olcDbCheckpoint: 512 30
olcDbDirectory: /var/symas/openldap-data
olcDbIndex: default eq
olcDbIndex: objectClass
olcDbMaxSize: 85899345920
olcAccess: {0} to attrs=userPassword
  by anonymous auth by self write by * none
olcAccess: {1} to attrs=shadowLastChange
```

```
    by anonymous auth by self write by * none
olcAccess: {2} to *
    by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth manage
    by * read
```

Wenn Sie diese Einträge mit der statischen Konfiguration vergleichen, finden Sie hier alle Einstellungen wieder, nur werden jetzt die Informationen aus einer eigenen Datenbank gelesen. Jeder Teilbereich der Konfiguration wird in einem eigenen *DN* beschrieben, die einzelnen Bereiche werden für die folgenden Einstellungen benutzt:

- *dn: cn=config*
  Die Grundkonfiguration der globalen Konfiguration, auch in der statischen Konfiguration finden Sie diese Einträge im globalen Bereich wieder.
- *dn: cn=schema,cn=config*
  Auch die von Ihnen konfigurierten Schemata werden in der Konfigurationsdatenbank abgelegt und erhalten hierfür einen eigenen Bereich in Form eines *DN*.
- *dn: cn=module{0},cn=config*
  Hier handelt es sich um den Teil der Konfiguration, in dem alle benötigten Module eingetragen werden. Sie sehen hier, dass dort auch gleich das Modul für den Passwordhash *ARGON2* eingetragen wird, damit der Hash auch schon für die Passwörter der *rootdn* verwendet werden kann. Im Verlauf des Buchs werden wir die Liste kontinuierlich erweitern.
- *dn: olcDatabase={-1}frontend,cn=config*
  Das ist der zweite Teil der globalen Konfiguration; hier können Sie schon ACLs festlegen, wer diesen Bereich verwalten darf. Auch werden hier ACLs eingetragen, die für alle Datenbanken auf dem LDAP-Server gelten sollen. Der Zugriff auf die oberste Ebene und auf die Schemata muss immer gewährleistet sein. Durch den Eintrag an dieser Stelle muss der Zugriff nicht immer für jede Datenbank einzeln vergeben werden.
- *dn: olcDatabase={1}monitor,cn=config*
  Hier sehen Sie die erste Datenbank, die nicht an eine feste Reihenfolge gebunden ist. Alle vorherigen *DNs* haben grundsätzlich immer dieselbe Nummer. Ab jetzt hängt die Nummerierung der Datenbanken immer davon ab, in welcher Reihenfolge Sie die Datenbank definieren. Deshalb ist es ganz wichtig, dass Sie ab jetzt bei jeder Änderung der Konfiguration genau auf die Nummerierung der Datenbanken achten.

  Bei der Datenbank, die hier definiert wird, handelt es sich um die Datenbank, in der alle Aktionen, die ein Client auf dem LDAP-Server ausführt, protokolliert werden. Diese Daten können Sie in Ihrem Monitoring auswerten, und so haben Sie immer einen guten Überblick über den Zustand Ihres LDAP-Servers.
- *dn: olcDatabase={2}mdb,cn=config*
  Bei diesem *DN* handelt es sich jetzt um die Konfiguration der Objektdatenbank. Auch hier finden Sie alle Parameter aus der statischen Konfiguration wieder.

Eine Sache fällt sofort ins Auge: Die Parameter haben hier andere Namen und beginnen immer mit *olc*. Die Abkürzung steht für *Open LDAP Configuration*. Die Schreibweise der Parameter ist etwas anders; wenn Sie einen bestimmten Parameter suchen, finden Sie die Schreibweise immer in den LDIF-Dateien im Verzeichnis /opt/symas/etc/openldap/schema. Später können Sie auch in der dynamischen Konfiguration nach den Namen suchen, dazu muss der LDAP-Server aber erst laufen.

Hinweis: Wenn Sie bereits Erfahrung mit der dynamischen Konfiguration im OpenLDAP 2.4 gesammelt haben und dort auch schon andere Passworthashes nutzen, gibt es ab der Version 2.5 eine Änderung. Der Parameter *olcPasswordHash* befindet sich jetzt im DN *dn: olcDatabase={-1}frontend,cn=config* und nicht mehr im DN *dn: cn=config*.

Einen ganz wichtigen Punkt haben wir bis jetzt noch nicht angesprochen. In fast jedem Bereich der Konfiguration finden Sie die folgende ACL *by dn.exact=gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth manage*. Was hat es damit auf sich? Diese ACL kann Ihnen das Leben, gerade am Anfang, etwas vereinfachen. Schauen wir uns die einzelnen Teile der ACL mal etwas genauer an:

- *dn.exact*
  Der folgende *DN* muss exakt mit dem hier angegebenen Wert übereinstimmen.
- *gidNumber=0+uidNumber=0*
  Die ACL trifft also auf den Benutzer mit der UID und GID *0* zu, also auf den *root*.
- *cn=peercred,cn=external,cn=auth*
  Hier wird eine im OpenLDAP vorkonfigurierte SASL-Authentifizierung verwendet. Diese Art der Authentifizierung ist nur lokal auf dem Server über den lokalen Socket *ldapi* möglich.
- *manage*
  Der Benutzer erhält den vollen Zugriff auf alle Objekte und Einträge.

Das heißt also, der Benutzer *root* hat direkt auf dem LDAP-Server vollen Zugriff auf alle Einträge, ohne sich zusätzlich mit einem Passwort authentifizieren zu müssen.

Hinweis: Diese Grundkonfiguration können Sie für alle folgenden LDAP-Server als Grundlage nehmen, denn hier sind noch keine rollenspezifischen Einstellungen enthalten.

Nachdem die Grundkonfiguration des OpenLDAP-Servers eingespielt ist, werden wir diese Art der Authentifizierung anhand des Kommandos `ldapsearch` erklären.

Einspielen der dynamischen Konfiguration

Kommen wir jetzt zur Lösung des Henne-Ei-Problems: das Einspielen der Konfiguration in die Datenbank. Wie vorher schon erwähnt, wird dafür das Kommando `slapadd` verwendet. Mithilfe des Kommandos `slapadd` können Sie Daten in verschiedenen Datenbanken des OpenLDAP einlesen. Daher benötigt das Kommando immer eine Zieldatenbank und eine Quelldatei, aus der die Informationen gelesen werden sollen. Neben diesen beiden Parametern wird ein weiterer Parameter benötigt, nämlich die Nummer der Datenbank, in die geschrieben werden soll. Da wir hier die Konfiguration schreiben wollen, ist das immer die Nummer *0*. In Listing 3.6 sehen Sie das Kommando und die Ausgabe nach erfolgreicher Einspielung der Konfiguration:

Listing 3.6 Einspielung der initialen Konfiguration

```
root@provider01:~# slapadd -n0 -F /opt/symas/etc/openldap/slapd.d/ -l config.ldif
Closing DB...
```

13 Loadbalancer mit lloadd

Nachdem Sie in Kapitel 12, «Replikation des OpenLDAP-Baums», eine Landschaft aus zwei oder mehr LDAP-Servern erstellt haben, besitzen Sie eine solide und ausfallsichere Umgebung. Nun können Sie sich dem Thema *Lastverteilung* zuwenden. Ziel dieses Kapitels ist es, in Ihrer Umgebung den bestehenden LDAP-Servern einen *Loadbalancer* vorzuschalten, welcher die Anfragen gleichmäßig auf die einzelnen LDAP-Server verteilt.

13.1 Übersicht über lload

OpenLDAP bietet Ihnen mit *lload* eine solche Lösung der Lastverteilung. Dabei verteilt *lload* nicht die Verbindungen, sondern nimmt die Lastverteilung auf Basis der einzelnen Operationen durch. Daher setzt *lload* identische Daten auf allen LDAP-Servern voraus. Der Algorithmus kontrolliert nicht, welche Operationen der Client im Einzelnen durchführt und kann daher auch keine „intelligente Verteilung“ durchführen. Beim Einsatz von OpenLDAP 2.5 sollte ein Client, welcher eine durchgehende Verbindung z. B. für mehrere Schreiboperationen zu genau einem LDAP-Server benötigt, eine Verbindung direkt mit einem LDAP-Server anstelle des Loadbalancers aufbauen. Ab Version 2.6 existiert die Möglichkeit, den Loadbalancer für einen Client *kohärente* Verbindungen zu einem LDAP-Server aufbauen zu lassen.

13.1.1 Funktionsweise von lload

Schauen wir uns die Funktionsweise von *lload* etwas genauer an. Der Loadbalancer baut beim Start des Dienstes eine Verbindung zum LDAP-Verzeichnis auf. Dazu authentifiziert sich der Dienst mit einem eigens dafür einzurichtenden Benutzer. Verbindet sich nun ein Client mit dem Loadbalancer, leitet dieser die Anfragen des Clients über diese Verbindung an die LDAP-Server weiter. Diese Anfragen finden dabei mit den Rechten des verbundenen Clients statt. Bild 13.1 zeigt Ihnen schematisch den Aufbau.

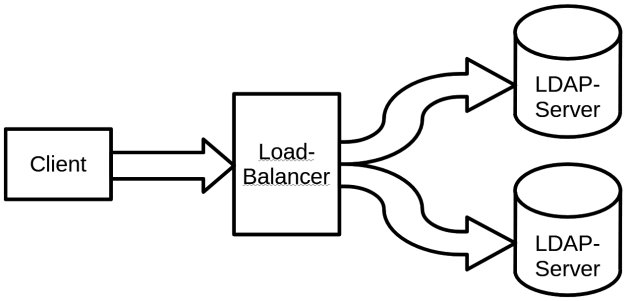

Bild 13.1 Schema eines Loadbalancers

13.1.2 Voraussetzungen

Grundvoraussetzung für den Einsatz eines Loadbalancers ist natürlich, dass die dahinter liegenden LDAP-Server, auf welche der Loadbalancer die Verbindungen verteilt, einen identischen Datenbestand besitzen. Daher ist eine entsprechende Replikation zwischen den LDAP-Servern unerlässlich, um konsistente Ergebnisse zu erhalten.

Bei der Konfiguration geben Sie neben diesem Benutzer – dem *bindDN* – an, um welche Art Verbindung es sich handelt. Neben der Verbindung pro Operation gibt es wie erwähnt ab OpenLDAP 2.6 auch die Möglichkeit einer kohärenten Verbindung. Da es Clients gibt, welche einen hohen Datendurchsatz für Leseoperationen benötigen und daher nicht auf eine kohärente Verbindung angewiesen sind und es andererseits Clients geben wird, welche z. B. für Schreiboperationen eine dauerhafte Verbindung für mehrere Operationen zu einem LDAP-Server benötigen, sollten Sie zwei unterschiedliche Konfigurationen erstellen.

Wie beschrieben, benötigen wir einen weiteren Benutzer im LDAP-Verzeichnis, mit welchen der Loadbalancer sich gegenüber dem LDAP-Verzeichnis beim Start authentifiziert. Die späteren Abfragen werden dann im Namen des Benutzers durchgeführt, welcher die Abfrage vom Client aus absetzt. Dabei wird dessen *bindDN* vom Loadbalancer an den jeweiligen LDAP-Server übertragen, ohne eine weitere Authentifizierung durchzuführen. Eine solche *Proxy-Authentifizierung* wird aber standardmäßig vom LDAP-Server bzw. dem jeweiligen Backend nicht erlaubt, und es ist eine entsprechende Anpassung notwendig.

In diesem Kapitel werden wir den Loadbalancer über einen separaten LDAP-Server beschreiben. Es gibt auch die Möglichkeit, *lload* als eigenen Daemon ohne *slapd*-Service zu betreiben. Allerdings hat die hier beschriebene Methode den Vorteil, dass wir dann auch auf die Leistungsdaten des *monitor*-Overlays zurückgreifen können.

13.2 Vorbereitungen für lload

Wie beschrieben, müssen vor dem Einsatz eines Loadbalancers einige Vorbereitungen getroffen werden, die wir hier nun schrittweise beschreiben.

13.2.1 Proxy-Authentifizierung

Als Erstes sollten Sie die Konfiguration der LDAP-Server durchführen. Beachten Sie, dass Sie dies für alle beteiligten LDAP-Server tun. Wenn Sie die *config*-Datenbank ebenfalls replizieren, ist dies natürlich nur einmal notwendig.
Für die einzelnen Server ist die *Proxy-Authentifizierung* für die Backends zu aktivieren. Dies können Sie zum Beispiel mit dem LDIF aus Listing 13.1 einrichten.

Listing 13.1 Einrichtung von syncprov

```
dn: cn=config
changetype: modify
replace: olcAuthzpolicy
olcAuthzpolicy: to
```

Der Wert für das Attribut *olcAuthzpolicy* kann dabei die Werte *none* (Default) für „keine Proxy-Authentifizierung“, *to* für das Erlauben ankommender, *from* für ausgehende oder *both* für ein-und ausgehende Proxy-Authentifizierungen annehmen. In unserem Fall benötigen wir lediglich eingehende.

13.2.2 Erstellen des Proxy-Benutzers

Nun können Sie den eigenen Benutzer für *lload* in Ihrem LDAP-Verzeichnis einrichten. Listing 13.2 zeigt Ihnen ein Beispiel für eine LDIF-Datei für einen solchen Benutzer. Dieser Benutzer benötigt zusätzlich die Eigenschaft, eine Proxy-Authentifizierung durchzuführen. Dies wird anhand des Attributs *authzTo* realisiert. Dieses Attribut gibt anhand einer LDAP-URI an, für welche Objekte der *lload*-Benutzer eine Proxy-Authentifizierung durchführen darf.

Listing 13.2 lload-Benutzer

```
dn: uid=lloadd,ou=users,dc=example,dc=net
objectClass: account
objectClass: simpleSecurityObject
objectClass: top
uid: lloadd
userPassword: {ARGON2}$argon2i$v=19$m=4096,t=3,p=1$MTIzZmRmZjY3ODY1NDMy$ \
 bFN6MX4smW70QiwRG+jiQLsLv3C69LDQHL+ZE/infR8
authzTo: ldap:///dc=example,dc=net??sub?(|(uid=*)(cn=*))
```

Mit dem Setzen des Attributs *authzTo*, wie in Listing 13.2 gezeigt, können sich Benutzer mit ihrem *cn* oder ihrer *uid* authentifizieren. Eine weitere Möglichkeit für den Wert von *authzTo* ist die Angabe über einen regulären Ausdruck der Form *dn.regex:^uid=[^,]*,dc=example,dc=com$* oder eines einzelnen *Distinguished Names* wie *dn:uid=berater,ou=users,dc=example,dc=net*.

Hinweis: Beachten Sie bei der Bearbeitung mit dem ApacheDirectoryStudio oder einer Suche nach dem Attribut *authzTo*, dass es sich um ein operationales Attribut handelt.

Wichtig: Ein Benutzer mit Schreibrechten auf das Attribut *authzTo* seines Benutzerobjektes kann sich eine Hintertür schaffen, die Identität eines beliebigen anderen Objektes anzunehmen. Daher ist dieses Attribut dringend mit einer entsprechenden ACL zu versehen.

13.2.3 Rechte für den Proxy-Benutzer

Damit der Benutzer eine Proxy-Authentifizierung für einen anderen Benutzer in einer Datenbank durchführen kann, benötigt er noch das *auth*-Privileg für die jeweiligen Attribute, in diesem Fall die Anmeldung mit dem *cn* sowie der *uid*. Listing 13.3 zeigt Ihnen eine LDIF-Datei für die Gewährung eines entsprechenden Rechts.

Listing 13.3 ACL zur Proxy-Anmeldung

```
dn: olcDatabase={2}mdb,cn=config
changetype: modify
add: olcAccess
olcAccess: {0} to attrs=uid,cn
 by dn.exact=uid=lloadd,ou=users,dc=example,dc=net auth
 by *
```

Der neue Benutzer muss nun noch in die Lage versetzt werden, die Benutzer im LDAP-Verzeichnis zu finden. Dazu benötigt er die entsprechenden Rechte in der Datenbank bzw. den entsprechenden Bereichen. In unserem Fall benötigt er das Leserecht auf das Attribut *uid* im gesamten Baum. Die LDIF-Datei in Listing 13.4 gibt ein Beispiel für entsprechende ACLs:

Listing 13.4 Einrichten der ACL für den lload-Benutzer

```
dn: olcDatabase={2}mdb,cn=config
changetype: modify
add: olcAccess
olcAccess: {1} to *
   by dn.exact="uid=lloadd,ou=users,dc=example,dc=net"  read
   by * break
-
add: olcLimits
olcLimits: {0} dn.exact="uid=lloadd,ou=users,dc=example,dc=net"
   time=unlimited
   size=unlimited
```

Wichtig: Beachten Sie, dass die ACL aus Listing 13.4 nach der ACL aus Listing 13.3 steht, da sonst die *auth*-Berechtigung nicht greift.

```
dn: cn=module{0},cn=config
objectClass: olcModuleList
cn: module{0}
olcModulePath: /opt/symas/lib/openldap
olcModuleLoad: {0}back_mdb
olcModuleLoad: {1}back_monitor
olcModuleLoad: {2}argon2.la

[...]
```

Die Option `--rm` gibt an, dass der Container nach Beendigung direkt wieder gelöscht wird. Das verhindert insbesondere bei Tests, dass Sie nach dem Test den Container manuell mit `docker rm` löschen müssen.

Am Ende hier in Listing 18.32 noch eine passende docker-compose.yml-Datei.

Listing 18.32 Statische Konfiguration von syncprov

```
version: "3"
services:
  slapd:
    image: symas:1.0
    volumes:
     - /var/docker/volumes/symas:/var/symas/openldap-data
    ports:
     - "389:389"
     - "636:636"
```

18.3.4 Ausblick

Das ist natürlich erst der Anfang für die Entwicklung eigener Images. Beim Setup lassen sich bereits weitere Module laden, Overlays einrichten, und eine grundlegende Struktur fehlt auch noch. Aber Sie haben nun das Rüstzeug an der Hand, um sich auf den Weg zu machen, mehr mit Docker zu arbeiten. Wandeln Sie die Skripte und LDIF-Dateien ab und ergänzen Sie diese. Das Internet bietet Ihnen zudem etliche Hinweise, Tipps und Tricks für den Bau eigener Images.

Tipp: Was im Büro und in der Werkstatt gilt, hat selbstverständlich auch hier seine Gültigkeit: Räumen Sie ab und zu auf. Der Befehl `docker system prune` hilft Ihnen dabei und entfernt unter anderem beendete Container und ungenutzte Images.

18.4 Kubernetes

Bislang haben wir uns die Container-Lösung nur mit einem einzelnen Host angeschaut. Am Ende dieses Kapitels wollen wir noch einen Blick auf eine Cluster-Lösung für Container werfen: *Kubernetes* oder kurz *K8*. Ein solcher Cluster besteht aus mindestens einem

Master- und mehreren Worker-Knoten. Sämtliche Server sind Server mit einer Container-Lösung wie Docker oder Cri-O. Auf dem Master-Knoten laufen diverse Container zur Verwaltung des Clusters wie beispielsweise ein API-Server, ein Scheduler oder eine Konfigurationsdatenbank. Die Worker-Knoten übernehmen, wie der Name schon sagt, die eigentliche Arbeit. Dort laufen die Webserver, Webservices etc., kurz: die zu veröffentlichenden Dienste. Zur Verwaltung dient u. a. das Kommandozeilentool `kubectl`, welches auf dem Master-Knoten oder über die API-Schnittstelle von einem beliebigen Rechner aus gestartet werden kann. Darüber hinaus liefert Kubernetes ein übersichtliches Dashboard.

18.4.1 minikube

Damit Sie sich zum Kennenlernen von Kubernetes nun keine umfangreiche Rechnerlandschaft aufbauen müssen, können Sie auf *minikube* zurückgreifen. Dieses Projekt liefert Ihnen eine Kubernetes-Umgebung bestehend aus einem einzelnen virtuellen Server bzw. einem einzelnen Container. Unter *https://github.com/kubernetes/minikube/releases/* können Sie sich die aktuelle Version von Minikube herunterladen und installieren. Minikube besitzt Treiber für diverse Virtualisierungsumgebungen wie VirtualBox oder Docker. Über diesen Treiber lässt sich von der Kommandozeile aus eine Minikube-Umgebung einrichten und verwalten. Mit dem Befehl `minikube start -driver=virtualbox` wird beispielsweise ein Minikube-Image für VirtualBox heruntergeladen, eine neue virtuelle Maschine eingerichtet und gestartet.
Im Folgenden zeigen wir Ihnen, wie Sie Minikube unter Debian realisieren. Sollten Sie eine Kubernetes-Umgebung im Zugriff haben, so können Sie die Beispiele natürlich ebenfalls verwenden.

18.4.1.1 Einrichtung unter Debian

Hardware-Voraussetzung für den Einsatz von minikube sind zwei GB RAM, zwei CPUs und 20 GB freier Plattenplatz. Benötigt wird zudem eine Container- oder Virtualisierungsplattform wie Docker, Cri-O, QEMU oder Ähnliches. In unserem Beispiel verwenden wir Docker. Nach der Installation von Docker erfolgt die Installation von *minikube* mit folgenden Befehlen:

- `curl -LO https://storage.googleapis.com/minikube/releases/latest/ \ minikube_latest_amd64.deb`
- `dpkg -i minikube_latest_amd64.deb`
- `minikube start -driver=docker -force`

Beim Start von *minikube* werden sämtliche Docker-Images geladen, welche für Kubernetes benötigt werden. Dies kann daher einige Minuten dauern.

18.4.1.2 Die Kommandozeile

Kubernetes verwendet für die Interaktion in der Hauptsache den Befehl `kubectl`. Unter *minikube* lauten die gleichen Befehle dann `minikube kubectl -`. Beim ersten Aufruf wird der Befehl `kubectl` heruntergeladen. Um sich das Leben einfacher zu machen, können Sie sich natürlich dafür einen Alias `kubectl` erstellen. Einige Befehle für die ersten Gehversuche:

- `kubectl config view` zeigt Ihnen die aktuelle Konfiguration des Clusters.
- `kubectl get pods -all-namespaces` listet Ihnen alle laufenden Pods in allen Namensräumen (s. u.) auf.
- `kubectl logs <Pod>` gibt die Logs des jeweiligen Pods aus.
- `kubectl get events` zeigt Ihnen die aktuellsten Ereignisse.

18.4.1.3 Das Dashboard

Um das Dashboard zu starten, geben Sie den Befehl `minikube dashboard` ein. Auch dabei wird zunächst ein weiteres Image heruntergeladen. Um später auch Leistungsdaten wie die CPU- oder Speicherauslastung im Dashboard zu sehen, können Sie das entsprechende Add-on mit dem Befehl `minikube addons enable metrics-server` einrichten. Nachdem das Dashboard eingerichtet ist, wird Ihnen die entsprechende URL angezeigt, welche Sie aufrufen können, um das Dashboard zu verwenden. Der Befehl läuft im Vordergrund. Daher sollten Sie ihn mit `minikube dashboard &` aufrufen.

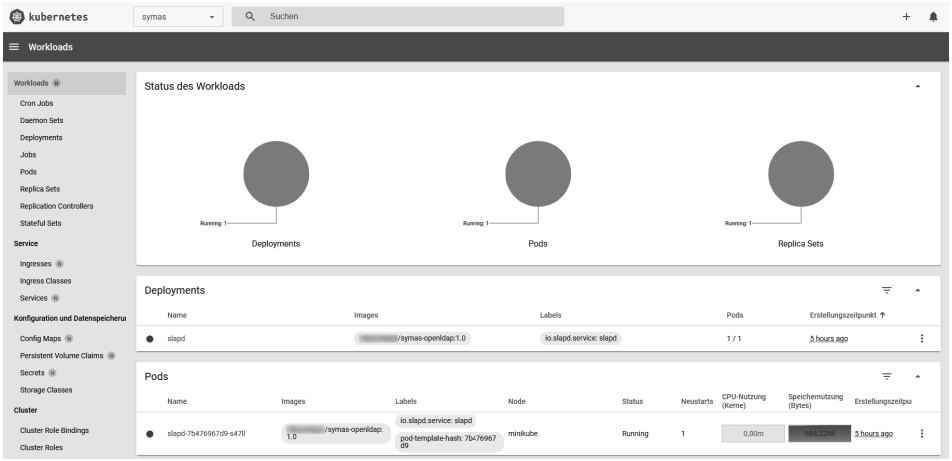

Bild 18.4 Kubernetes Dashboard

18.4.2 Registry

Wollen Sie in den folgenden Beispielen das selbstgebaute Image verwenden, so benötigen Sie wahrscheinlich eine externe Registry. Hier gibt es am Markt diverse Anbieter wie *docker.io*, *canister.io* etc., welche u. a. kostenlosen Platz für Images anbieten. Sie erhalten dort einen eigenen Namensraum und können dann öffentliche oder private Repositorys anlegen.

Nehmen wir an, Sie legen bei *docker.io* ein Konto mit dem Namen *kuberookie* sowie ein Repository *symas-openldap* an. Dann geben Sie Ihrem Image – nennen wir es mal *myopenldap:1.0* – ein neues *Tag* mit dem Befehl
`docker tag myopenldap:1.0 kuberookie/symas-openldap:1.0`.
Dann können Sie dies mit dem Befehl `docker push kuberookie/symas-openldap:1.0`

in die Registry hochladen und später – von jedem anderen Container-Host – wieder darauf zurückgreifen. Diesen Namen werden wir auch im Weiteren verwenden. Vor dem Hochladen müssen Sie sich noch mit `docker login kuberookie` anmelden.

18.4.3 Namespace

Sämtliche Objekte in Kubernetes werden in Namensräumen unterteilt. Bei großen Umgebungen empfiehlt es sich, zusammengehörende Objekte in solchen *Namespaces* zusammenzufassen. Kubernetes-Objekte werden in YAML-Dateien definiert. Ein minimales Beispiel für einen Namensraum finden Sie in Listing 18.33:

Listing 18.33 Definition eines Namespaces

```
kind: Namespace
apiVersion: v1
metadata:
  name: symas
```

Speichern Sie diese Datei unter slapd-namespace.yaml, so können Sie diese im Anschluss mit dem Befehl `minikube kubectl - apply ./slapd-namespace.yaml` einrichten.

Listing 18.34 Anlegen des Namensraumes

```
root@kube01:~/docker# minikube kubectl -- apply -f ./slapd-namespace.yaml
namespace/slapd-namespace created
```

18.4.4 Volume

Um die Daten persistent abzuspeichern, empfiehlt es sich, ein *Volume-Claim* zu erstellen. Damit legen Sie ein Objekt an, welches Speicherbereich zur Ablage von Daten definiert. Ein Beispiel für ein solches Objekt sehen Sie in Listing 18.35.

Listing 18.35 YAML-Repräsentation eines Volume-Claims

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: slapd-database-claim
  namespace: symas
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

In diesem Beispiel wird an die Ressource lediglich die Anforderung gestellt, 100 MebiByte Platz bereitzustellen. Zudem darf das Volume nur von einem Knoten zur gleichen Zeit

verwendet werden (*ReadWriteOnce*). Zusätzlich könnten Sie hier noch definieren, dass das Volume beispielsweise auf einem NFS-Server, einem Ceph-Storage oder einem iSCSI-Target angelegt werden soll. Abgespeichert als slapd-volume.yaml lässt sich das Volume nun mit `minikube kubectl - apply ./slapd-volume.yaml` anlegen.

Listing 18.36 Anlegen des Volumes

```
root@kube01:~/docker# minikube kubectl -- apply -f ./slapd-volume.yaml
volume/slapd-volume created
```

18.4.5 Deployment

Nun sind Sie soweit, den OpenLDAP-Server auf Kubernetes zu starten. Dazu benötigen wir ein *Deployment*. Wie dieses aussehen könnte, sehen Sie in Listing 18.37.

Listing 18.37 Deployment des OpenlDAP-Dienstes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: slapd
  namespace: symas
spec:
  replicas: 1
  selector:
    matchLabels:
      io.slapd.service: slapd
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        io.slapd.service: slapd
    spec:
      containers:
        - image: kuberookie/symas.openldap:1.0
          name: slapd
          ports:
            - containerPort: 389
              hostPort: 389
              protocol: TCP
            - containerPort: 636
              hostPort: 636
              protocol: TCP
          volumeMounts:
            - mountPath: /var/symas/openldap-data
              name: slapd-database
      restartPolicy: Always
      volumes:
        - name: slapd-database
          persistentVolumeClaim:
            claimName: slapd-database-claim
```

Schauen wir uns einige der Parameter genauer an:

- *replicas* legt fest, wieviele Pods gleichzeitig gestartet werden sollen.
- *selector* definiert, welche der weiter unten genannten *Templates* gestartet werden sollen. In diesem Fall wird nach dem *label* ausgewählt.
- *strategy* zeigt an, dass die betroffenen Pods gelöscht und neu erstellt werden sollen, wenn sich ihre Definition ändert. Alternativ könnten Sie den Wert *RollingUpgrade* verwenden, welcher die Pods einer nach dem anderen löscht und neu erstellt, wodurch keine Unterbrechungen entstehen.
- *containerPort* gibt den Port an, unter welchem der Pod den Port des Images präsentiert.
- *hostPort* definiert den Port, den der WorkerNode verwendet und an den Pod weiterreicht.
- *volumeMounts* verbindet das Volume des Images – /VAR/SYMAS/OPENLDAT-DATA – mit einem Volume.
- *volumes* legt fest, welche Volumes des Deployments an welcher Stelle abgelegt werden. In diesem Fall soll der vorher definierte Claim verwendet werden.
- *restartPolicy* gibt an, wie sich die Pods verhalten sollen, wenn sie beendet werden. Alternative Werte sind *OnFailure* – was einen Neustart nur bei einem ungewollten Absturz bewirkt – oder *Never*.

Wenn Sie die Datei unter slapd-deployment.yaml speichern, können Sie das Deployment mit `minikube kubectl - apply ./slapd-deployment.yaml` erstellen:

Listing 18.38 Anlegen des Deployments

```
root@kube01:~/docker# minikube kubectl -- apply -f ./slapd-deployment.yaml
deployment/slapd-deployoment created
```

18.4.6 Ausblick

Im gezeigten Beispiel haben wir natürlich noch einige Stellen, die optimiert oder ausgebaut werden können. Der OpenLDAP-Server ist beispielsweise immer unter der IP-Adresse des jeweiligen Workers erreichbar. Bei einem Schwenk auf einen anderen Worker würde sich diese ändern. Hier kommen dann noch weitere Elemente mit ins Spiel wie der *Services*, welcher Deployments unter Verwendung einer einheitlichen Cluster-IP-Adresse zusammenfasst, sowie Loadbalancer (siehe 13). Zudem liegen die Daten im vorliegenden Beispiel noch lokal auf dem Worker. In der Praxis sollte hier natürlich etwas im Netzwerk bzw. im SAN verwendet werden. Dazu gibt es im Netz etliche Anleitungen sowie z. B. das Buch *Kubernetes in Action* von Marko Lukša.

19 OpenLDAP einrichten mit Ansible

Es muss nicht immer Handarbeit sein, der Trend geht immer mehr in Richtung Automatisierung – so auch bei der Einrichtung von Serversystemen. Deshalb haben wir uns entschlossen, ein Kapitel zur Einrichtung einer OpenLDAP-Umgebung mit Ansible ins Buch auf zu nehmen. Warum gerade Ansible? Ansible gewinnt immer mehr an Bedeutung, da es recht einfach zu erlernen und einzusetzen ist. Es ist genau wie OpenLDAP ein Open-Source-Produkt.

Hier im Buch wollen und können wir keine Einführung in Ansible geben. Wir haben aber trotzdem versucht, das Thema auch für Einsteiger verständlich zu halten. Klar gibt es immer Dinge, die man „besser" machen kann, wenn man Skripte schreibt. Uns ging es aber nicht um die möglichst effektivste Methode, mit Ansible OpenLDAP einzurichten, sondern darum, eine einfache und verständliche Lösung zu zeigen.

Alle Dateien für die Rollen finden Sie in den Downloads zum Buch.

Was wollen wir erreichen?

Mit unserer Ansible-Rolle wollen wir einen OpenLDAP-Cluster mit zwei Providern mit Multi-Provider-Replikation einrichten, wobei wir sowohl die Objektdatenbank als auch die Konfigurationsdatenbank replizieren wollen. Bei der Replikation der Objektdatenbank soll *DeltaSync* zum Einsatz kommen.

Was benötigen Sie?

Damit Sie die Beispiele in diesem Kapitel nachvollziehen können, benötigen Sie zwei Maschinen für die Knoten des LDAP-Clusters und eine Maschine als Ansible-Host.

Auf dem Ansible-Host installieren Sie als Erstes eine aktuelle Version von Ansible. Wir verwenden hier die Ansible-Version 2.9 aus den Debian-Backports.

Alle unsere Playbooks sollen später auf den LDAP-Servern unter einer nichtprivilegierten Benutzerkennung laufen. Zu diesem Zweck haben wir einen lokalen Benutzer *ansible* angelegt. Wichtig ist, dass Sie für den Benutzer einen ssh-key ohne Passphrase erzeugen. Den Key verwenden wir später für die Verbindung zu den LDAP-Hosts. Alle Rollen liegen dann im Basisverzeichnis des Benutzers *ansible*. Im Basisverzeichnis haben wir ein Unterverzeichnis /home/ansible/ldap-buch angelegt. Darin befindet sich die Datei ansible.cfg mit dem Inhalt aus Listing 19.1:

Listing 19.1 Inhalt der Datei ansible.cfg

```
[defaults]
log_path = ~/ldap-buch/logs/ansible.log
inventory = inventories/inventory
roles_path = ./roles
```

Die Datei inventories/inventory enthält die Inhalte aus Listing 19.2:

Listing 19.2 Inhalt der Datei inventory

```
[ldap_server]
ldap-a01
ldap-a02

[ldap_server:vars]
ansible_python_interpreter=/usr/bin/python3
ansible_user=ansible
ansible_become=yes
```

Die Namen in der Serverliste müssen über DNS oder eine Datei /etc/hosts auflösbar sein.

19.1 Rolle zur Vorbereitung

Damit die LDAP-Server später auch über Ansible mit einem nichtprivilegierten Benutzer verwaltet werden können, haben wir eine Rolle *ansible_basic* erstellt. Diese Rolle dient nur zur Voreinstellung der Hosts.

Alle benötigten Variablen sind in der Datei default/main.yml abgelegt. Den Inhalt der Datei sehen Sie in Listing 19.3:

Listing 19.3 Variablen der Rolle ansible_basic

```
# ansible_userid and ansible_user_name will have the same value in all roles
ansible_userid: "1111"
ansible_user_name: "ansible"
# Password created with "mkpasswd -m sha-512 <password>"
# Default "geheim"
ansible_user_pw: "$6$xxXsW/...."
# Path to the .ssh-directory of the ansible-user on the ansible-host
ssh_path: "/home/ansible/.ssh"
# ssh-key from ansible-user on ansible-host
ssh_key_src: "/home/ansible/.ssh/id_rsa.pub"
# Destination of the authorized_keys file on the target host
ssh_key_dest: "/home/ansible/.ssh/authorized_keys"
```

Bei der Rolle werden keine *templates* oder *files* genutzt. Alle Schritte werden direkt über die Tasks durchgeführt.

Die Datei templates/main.yml enthält nur einen Verweis auf die eigene Datei ansible_basic.yml mit dem Task. Den Inhalt der Datei sehen Sie in Listing 19.4:

Listing 19.4 Die Tasks zur Rolle ansible_basic

```
  - name: Check that variables are defined
    assert:
      that:
        - ansible_userid is defined
        - ansible_user_name is defined
        - ansible_user_pw is defined
        - ssh_path is defined
        - ssh_key_src is defined
        - ssh_key_dest is defined

  - name: reread packagelist
    apt:
      update_cache: yes
      cache_valid_time: 86400

#Installing all needed packages
  - name: install package "sudo" and "whois"
    apt:
      name:
        - sudo
        - whois

# Create user ansible, this user will do all changes
# we will use the same user in all other roles
  - name: Add user "ansible" to server
    user:
      name: "{{ ansible_user_name }}"
      comment: "ansible user"
      uid: "{{ ansible_userid }}"
      shell: /bin/bash
      password: "{{ ansible_user_pw }}"
      update_password: on_create

# Create the .ssh directory to store the authorized_keys file
  - name: create /home/{{ansible_user_name}}/.ssh
    file:
      path: "{{ ssh_path }}"
      state: directory
      mode: '0700'
      owner: "{{ ansible_user_name }}"
      group: "{{ ansible_user_name }}"

# Copy the public-key for user ansible from ansible-host
# and create authorized_keys file
  - name: copy public-key from user ansible to host
    copy:
      src: "{{ ssh_key_src }}"
      dest: "{{ ssh_key_dest }}"
      owner: "{{ ansible_user_name }}"
      group: "{{ ansible_user_name }}"
      mode: '644'

# Give user "ansible" root-permission on host with no password
  - name: change "sudoers" on all ldap_server add user ansible
```

```
    ansible.builtin.blockinfile:
      path: /etc/sudoers
      block: |
        # root-permission to ansible-user without password
        {{ansible_user_name}} ALL=(ALL:ALL) NOPASSWD: ALL
      validate: /usr/sbin/visudo -csf %s


# After setting up the configuration revoke root ssh-login
  - name: replace "PermitRootLogin= yes" for ssh
    replace:
      path: /etc/ssh/sshd_config
      regexp: '^PermitRootLogin yes$'
      replace: '#PermitRootLogin prohibit-password'

  - name: Restart service ssh
    ansible.builtin.service:
      name: ssh
      state: restarted
```

Die Einrichtung der Rolle findet hier noch als *root* statt. Während der Einrichtung muss auf den LDAP-Server eine ssh-Verbindung als *root* möglich sein. Das *Playbook* wird diese Möglichkeit am Ende des Skripts wieder deaktivieren.

Für diese Rolle ist es notwendig, dass Sie in der Datei inventory die beiden Zeilen aus Listing 19.5 auskommentieren:

Listing 19.5 Diese Zeilen auskommentieren

```
#ansible_user=ansible
#ansible_become=yes
```

Hinweis: Bevor Sie das Playbook ausführen, verbinden Sie sich einmal vom Ansible-Host zu den LDAP-Server, um die Hostkeys zu laden.

Die Rolle wird über das Playbook playbook/ansible_basic.yml ausgeführt. Dazu geben Sie das Kommando
`ansible-playbook playbook/ansible_basic.yml -u root -ask-pass` ein.

Im Anschluss entfernen Sie die Kommentarzeichen wieder aus der Datei inventory.

Damit sind alle Hosts soweit vorbereitet, dass Sie die Rolle *ldap_basic* vorbereiten und einspielen können.

19.2 Die Rolle zur Einrichtung von OpenLDAP

Nachdem Sie die Hosts vorbereitet haben, geht es jetzt an die Einrichtung von OpenLDAP. Nachdem das Playbook durchgelaufen ist, sind die folgenden Dinge komplett konfiguriert:

1. Auf beiden Servern sind die Symas-Pakete installiert, und der Dienst läuft nicht mehr unter der root-Kennung.
2. Eine Grundkonfiguration wurde erstellt und eingespielt. Dazu wurden die Variablen aus der Datei default/main.yml verwendet.
3. TLS wurde eingerichtet.
4. Die benötigten Objekte für die Administration und Replikation wurden angelegt.
5. DeltaSync für die Objektdatenbank wurde komplett eingerichtet.
6. Die Replikation der Objektdatenbank funktioniert.
7. Die Replikation der Konfiguration funktioniert.

Wichtig bei der Konfiguration ist, dass die Konfiguration auf allen Servern absolut identisch ist. Das schließt auch die Pfade und Namen zu den Zertifikaten mit ein, obwohl jeder Server seine eigenen Zertifikate besitzt.

Nur wenn die Konfiguration am Ende absolut identisch ist, können Sie die Konfiguration später auf einen weiteren LDAP-Server kopieren und mit kleinen Anpassung Ihren LDAP-Cluster erweitern.

Passen Sie die Variablen in der Datei roles/ldap_basic/default/main.yml an Ihre Umgebung an. Wollen Sie gleich mehr als zwei LDAP-Server einrichten, fügen Sie alle Servernamen zur Variable *server_id* hinzu.

Prüfen Sie auch, ob die Liste der Overlays und Schemata ausreichend ist; wenn nicht, erweitern Sie auch hier die Liste.

19.2.1 Vorbereitung für TLS

OpenLDAP und TLS via Ansible einrichten ist etwas komplexer. Am Ende soll ja eine Konfiguration entstehen, die auf allen LDAP-Server absolut identisch ist, inklusive der Dateinamen und Pfade für TLS. Aber jeder Server soll trotzdem seine eigenen Zertifikate erhalten. Natürlich wäre es auch möglich, für alle LDAP-Server ein Wildcard-Zertifikat zu erstellen. Das könnte dann aber auch auf anderen Server genutzt werden, und daher ist das nicht die sicherste Lösung. Um das Problem mit den unterschiedlichen Namen zu lösen, sind wir zu folgender Lösung gekommen:

- Im Verzeichnis files wird für jeden Server, der eingerichtet werden soll, ein eigenes Unterverzeichnis angelegt.
- Der Name des Unterverzeichnisses muss den Hostnamen, der über die Ansible-Variable *inventory_hostname* bereitgestellt wird, haben.
- In dieses Verzeichnis werden dann die Dateien für die Zertifikate der jeweiligen Server abgelegt. Dabei sind die Namen über die Variablen *inventory_hostname* und *key_suffix* für den Schlüssel und *inventory_hostnamecert_suffix* für das Zertifikat festgelegt.
- Der Name des root-Zertifikats Ihrer CA wird über die Variable *tlsca_file* bestimmt.

Während die Tasks laufen, werden die Dateien kopiert. Mehr dazu, wenn wir uns um die entsprechenden Tasks kümmern.