

Software

- [Keycloak](#)
- [Mailsetup](#)
 - [Mailinglisten](#)

Keycloak

Dies ist die technische Anleitung primär zum Einbinden neuer Dienste. Suchst Du vielleicht die [Anleitungen/Single Sign-On?](#)

Keycloak

[Keycloak](#) ist eine Identity Management Software, in Java geschrieben. Wir setzen sie seit August 2025 auf auth.ping.de ein.

Deployed als docker container auf hafen. Für die User aus der Ping- und Prima-Mitglieder Datenbank haben wir den Realm PING angelegt.

Der Datenexport aus den Datenbanken erfolgt über Perl-Scripte, die hier auch mal dokumentiert werden sollten.

Unter der URL <https://auth.ping.de/realms/PING/account> können Mitglieder z.B. 2FA konfigurieren.

Es gibt in dem Realm zwei Gruppen "ping" und "prima", so dass gewisse Dienste auf diese Gruppen eingeschränkt werden können (wenn Daten nicht existieren für alle User u.s.w.).

Neue Anwendung zu Keycloak hinzufügen

Wenn man dann eine Software hat, die sich via [OIDC](#) an ein [Single Sign-on](#) dranflanschen lässt, dann geht das wie folgt:

1. Im Keycloak **im Realm PING** einen neuen Client anlegen.
Meist findet man eine Anleitung, welche Parameter dabei zu beachten sind.
Vermutlich wollt ihr unter "capability config" den Punkt "client authentication" einschalten.
Darum geht es schließlich.
Tragt auch die richtige URL des Clients unter *Root URL* ein.
2. Nachdem ihr den Client angelegt habt, findet ihr in dessen *Client details* unter *Credentials* das *Client secret*. Das braucht ihr für die Anwendung, die angebunden werden soll.

3. In der Anwendung könnt ihr bei der Konfiguration des OIDC vielleicht den Erkennungsendpunkt angeben. Der lautet

<https://auth.ping.de/realms/PING/.well-known/openid-configuration>

Gruppen aus Keycloak in der Anwendung

Wenn ihr jetzt noch Rollen für besondere Berechtigungen vom Keycloak in eure Anwendung übernehmen wollt, dann könnt ihr im Keycloak noch Client Roles definieren, Usern diese zuweisen und dann in einem *Client scope* für den Client die Rollen mappen. Dazu legt ihr einen *token mapper* vom Typ *user client roles* an. Dort ist vermutlich die Checkbox "Add to ID token" wichtig, also aktiviert die.

Viel Erfolg.

Im Keycloak konfigurierte Dienste:

- conf.ping.de BigBlueButton greenlight-v3 UI 2026-04
- wiki.ping.de Bookstack 2026-03
- poz.ping.de wordpress 2026-04
- weiterbildung.ping.de wordpress 2026-04
- matrix anmeldung 2026-03
- code.ping.de forgejo git repo
- milla apache geschützter Bereich 2026-03
- cloud.ping.de nextcloud
- ki.ping.de open-webui

Mailsetup

Das PING Mailsetup

Mailinglisten

Die Mailinglisten werden über Mailman für die Domains lists.ping.de und lists.ping.ruhr bereitgestellt.

Ausgeführt wird das ganze als Docker-Container auf der VM mail.ping.ruhr. Dafür wird der [docker-mailman](#) Stack ausgeführt. Das Repo ist dabei unter `/opt/docker-mailman` ausgecheckt und in der Variante mit PostgreSQL als Datenbank ausgeführt.

Setup

Der Container-Stack besteht aus dem Mailman-Backend (`mailman-core`) und mehreren Frontends (`mailman-web`). Über die Frontends können die Listen administriert und ein Archiv eingesehen werden: <https://lists.ping.ruhr/>

Die Anbindung an die Mailcow geschieht über ein geteiltes Docker internes Netzwerk `mailman` (bzw. `docker-mailman_mailman`).

Single Sign-On Setup

Im Keycloak wurde der Client wie immer angelegt mit ID `mailman-web`.

Mailman-web Konfiguration

Im Verzeichnis `/opt/mailman/web/` die Datei `settings_local.py` wie folgt konfigurieren:

```
# locale
LANGUAGE_CODE = 'de-de'

# Verhindert lokale Registrierung komplett
#ACCOUNT_ADAPTER = 'django_mailman3.views.user_adapter.DisableSignupAdapter'

# Erlaubt OIDC-Account-Erstellung
#SOCIALACCOUNT_ADAPTER = 'django_mailman3.views.user_adapter.EnableSocialSignupAdapter'

# Erlaubt den SSO-Start durch Apache-Redirect (ohne CSRF-Token)
SOCIALACCOUNT_LOGIN_ON_GET = True

# disable social authentication
MAILMAN_WEB_SOCIAL_AUTH = ['allauth.socialaccount.providers.openid_connect']
```

```

# ping keycloak config
SOCIALACCOUNT_PROVIDERS = {
    "openid_connect": {
        "APPS": [
            {
                "provider_id": "mailman-web",      # Interne ID für die Callback-URL
                "name": "PING Single Sign-On",    # Beschriftung des Buttons auf der Website
                "client_id": "mailman-web",
                "secret": "redacted",
                "settings": {
                    "server_url": "https://auth.ping.de/realms/PING/.well-known/openid-
configuration",
                }
            }
        ]
    }
}
# change it
DEFAULT_FROM_EMAIL = 'mailman@ping.ruhr'

DEBUG = False

```

Apache2 Konfiguration:

Weil mailman-web das reguläre Login-Formular unterhalb des SSO-Login-Buttons nicht ausblenden kann behelfen wir uns damit die Seite im Apache zu umgehen. Damit das klappt, benötigten wir in der vorherigen Datei auch die gesetzte Option `SOCIALACCOUNT_LOGIN_ON_GET`.

In `/etc/apache2/sites-available/mailman.conf` im Virtualhost für SSL:

```

# Fängt die Login-Seite ab und leitet zum SSO
RewriteEngine On
RewriteRule ^/accounts/login/?$ /accounts/mailman-web/login/?process=login [R=307,L,QSA]
# dito signup seite
RewriteRule ^/accounts/signup/?$ /accounts/mailman-web/login/?process=login [R=307,L,QSA]

```

Mailman hacks

Falls man mal was mit den Usern machen muss gibt es die "manage.py shell".

Pruning einer bestimmten Mailingliste, Löschen von mails > XX tage

```
root@mail:/opt/docker-mailman# docker compose exec mailman-web python3 manage.py shell
```

```
from django.utils import timezone
from datetime import timedelta
from hyperkitty.models import Email, Thread, MailingList

# --- KONFIGURATION ---
# Trage hier die exakte Listen-ID ein (meist im Format: listenname.deinedomain.de)
ZIEL_LISTE = "root@lists.ping.de"
TAGE_BEHALTEN = 1
# -----

stichtag = timezone.now() - timedelta(days=TAGE_BEHALTEN)

try:
    # 1. Prüfen, ob die Liste überhaupt existiert
    target_list = MailingList.objects.get(name=ZIEL_LISTE)

    # 2. E-Mails nur für diese spezifische Liste löschen
    geloeschte_emails = Email.objects.filter(
        mailinglist=target_list,
        date__lt=stichtag
    ).delete()
    print(f"Erfolg: {geloeschte_emails[0]} E-Mails (inkl. Anhänge) aus '{ZIEL_LISTE}'
gelöscht.")

    # 3. Leere Threads aufräumen, die nur zu dieser Liste gehören
    geloeschte_threads = Thread.objects.filter(
        mailinglist=target_list,
        emails__isnull=True
    ).delete()
    print(f"Erfolg: {geloeschte_threads[0]} verwaiste Threads aus '{ZIEL_LISTE}' bereinigt.")

except MailingList.DoesNotExist:
    print(f"ABBRUCH: Eine Liste mit dem Namen '{ZIEL_LISTE}' wurde in der Datenbank nicht
gefunden.")
    print("Tipp: Prüfe die genaue Schreibweise (meist name.domain.tld).")
```

```
exit()
```

User zum Admin machen

```
from django.contrib.auth.models import User

# Benutzer anhand der E-Mail-Adresse suchen
user = User.objects.get(email="user@example.com")

# Rechte vergeben
user.is_superuser = True
user.is_staff = True

# Änderungen in der Datenbank speichern
user.save()

exit()
```

Mailadresse eines Users ändern

```
from django.contrib.auth.models import User
from allauth.account.models import EmailAddress

alte_mail = "alteadresse@example.org"
dummy_mail = "neueadresse@example.com"

# 1. Ändere die Adresse im Haupt-Benutzeraccount
user = User.objects.get(email=alte_mail)
user.email = dummy_mail
user.save()

# 2. Ändere die Adresse in der Allauth-Verwaltung
email_record = EmailAddress.objects.get(email=alte_mail)
email_record.email = dummy_mail
email_record.save()

exit()
```

Mailanhänge im Archiv löschen

(noch ungeprüft)

```
from django.utils import timezone
from datetime import timedelta
from hyperkitty.models import MailingList, Attachment

# --- KONFIGURATION ---
ZIEL_LISTE = "deine-liste@deinedomain.de"
TAGE_BEHALTEN = 90
# -----

stichtag = timezone.now() - timedelta(days=TAGE_BEHALTEN)

try:
    target_list = MailingList.objects.get(name=ZIEL_LISTE)

    # Der Trick: Mit "email__" springen wir in der Datenbank
    # von der Anhang-Tabelle rüber zur E-Mail-Tabelle, um nach Datum und Liste zu filtern
    anhaenge_zum_loeschen = Attachment.objects.filter(
        email__mailinglist=target_list,
        email__date__lt=stichtag
    )

    anzahl = anhaenge_zum_loeschen.count()

    print("-" * 30)
    print(f"ANALYSE FÜR LISTE: {target_list.display_name}")
    print(f"Stichtag (Anhänge älter als): {stichtag.strftime('%Y-%m-%d')}")
    print(f"Gefundene Anhänge zum Löschen: {anzahl}")
    print("-" * 30)

    if anzahl > 0:
        ergebnis = anhaenge_zum_loeschen.delete()
        print(f"ERFOLG: {ergebnis[0]} Objekte wurden aus der Datenbank entfernt.")
        print("Hinweis: Die E-Mail-Texte sind im Archiv weiterhin lesbar, nur die
        Büroklammer/Download-Links verschwinden.")
    else:
        print("Keine alten Anhänge gefunden. Nichts zu tun.")

except MailingList.DoesNotExist:
```

```
print(f"FEHLER: Die Liste '{ZIEL_LISTE}' existiert nicht.")
```

```
exit()
```