Flask-Deployment mit systemd und Caddy

https://gemini.google.com/app/ac1a3720075532db

User prompt: Ich habe eine Python Anwendung mit Flask entwickelt, die ich auf meinem Entwicklungsrechner mit der Kommandozeile 'flask --app cmbnet run --debug --host 0.0.0.0 --port 5001' starte. Zukünftig soll sie auf einem Linux Server gestartet mit systemd und produktiv durch einen kleinen Kreis von Usern genutzt werden. Wie kann ich den Code aus einem git Repository am einfachsten deployen und erstelle mir den passenden Eintrag in die Caddy Konfigurationsdatei für den Reverse Proxy und die passende System Konfigurationsdatei. Erkläre mir, ob ich einen eigenen User erstellen muss und wie ich das tue und wie ich die systemd Konfiguration anwende.

Response: Das Deployment einer Flask-Anwendung auf einem Linux-Server mit **systemd**, **Caddy** (als Reverse Proxy) und einem dedizierten System-User ist ein sauberer und robuster Standardweg für die Produktion.

Hier ist die Schritt-für-Schritt-Anleitung, wie du deine Anwendung am einfachsten aus einem Git-Repository einrichtest, absicherst und automatisierst.

1. Einen eigenen System-User erstellen

Ja, du solltest unbedingt einen eigenen, unprivilegierten System-User erstellen.

- **Warum?** Wenn deine Flask-App (oder eine ausgenutzte Sicherheitslücke in Python/einer Bibliothek) kompromitiert wird, hat der Angreifer nur Zugriff auf diesen spezifischen Benutzer und nicht auf das gesamte System (keine `root`-Rechte).

So erstellst du einen System-User (z. B. `cmbnet`) ohne Login-Shell und mit einem eigenen Home-Verzeichnis (wichtig für Caches, virtuelle Umgebungen etc.):

```
sudo useradd -r -s /bin/bash -m -d /opt/cmbnet cmbnet
```

- `-r`: Erstellt einen System-Benutzer (kleinere UID).
- `-s /bin/bash`: Setzt die Shell (hilft beim Debuggen, kann für reine Daemons auch `/bin/false` sein).
- `-m -d /opt/cmbnet`: Erstellt das Verzeichnis `/opt/cmbnet` als Heimatverzeichnis für die Anwendung.

Wechsle im Anschluss kurz den Kontext oder arbeite als `root`/dein Admin-User weiter, während du die App einrichtest.

2. Code aus dem Git-Repository klonen & Python vorbereiten

Klone dein Git-Repository in das Verzeichnis des neuen Users (oder einen gemeinsamen Pfad wie `/var/www/cmbnet` oder direkt in `/opt/cmbnet/app`). Nehmen wir an, wir klonen es nach `/opt/cmbnet/app`:

```
# Verzeichnis für den Code vorbereiten
sudo mkdir -p /opt/cmbnet/app
sudo chown -R cmbnet:cmbnet /opt/cmbnet

# Als User cmbnet klonen (oder mit sudo -u)
sudo -u cmbnet git clone <DEINE_GIT_URL> /opt/cmbnet/app
```

Richte nun die Python Virtual Environment (venv) ein und installiere die Dependencies:

```
cd /opt/cmbnet/app

# Virtuelle Umgebung als User cmbnet erstellen
sudo -u cmbnet python3 -m venv venv

# Abhängigkeiten installieren (Flask, Gunicorn als WSGI-Server etc.)
sudo -u cmbnet ./venv/bin/pip install --upgrade pip
sudo -u cmbnet ./venv/bin/pip install -r requirements.txt
```

> **Wichtig für die Produktion:** Da du von der Kommandozeile `flask run` (den Entwicklungs-Server) genutzt hast, solltest du für die Produktion einen WSGI-Server wie **Gunicorn** oder **Waitress** einsetzen. Installiere Gunicorn in deiner `venv`:
>
> ```
> sudo -u cmbnet ./venv/bin/pip install gunicorn
> ```

3. Die passende systemd-Konfigurationsdatei erstellen

Erstelle eine systemd-Service-Datei, damit systemd deine Flask-App im Hintergrund startet, überwacht und bei Systemstart automatisch hochfährt.

Erstelle die Datei `/etc/systemd/system/cmbnet.service`:

```
[Unit]
Description=Flask CMBNet Production Application
After=network.target

[Service]
User=cmbnet
Group=cmbnet
```

```
WorkingDirectory=/opt/cmbnet/app
Environment="PATH=/opt/cmbnet/app/venv/bin"
# Beispiel mit Gunicorn (4 Worker, bindet an localhost auf Port 5001)
ExecStart=/opt/cmbnet/app/venv/bin/gunicorn --workers 4 --bind 127.0.0.1:5001 "cmbnet:create_app()"

Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
```

(Hinweis: Ersetze `"cmbnet:create_app()"` durch den korrekten Importpfad deiner Flask-App, falls du die klassische Struktur mit `app = Flask(__name__)` nutzt, schreibst du stattdessen z.B. `gunicorn --workers 4 --bind 127.0.0.1:5001 cmbnet:app`.)

systemd-Konfiguration anwenden:

Damit systemd die neue Datei erkennt und den Dienst startet, führe folgende Befehle aus:

```
# 1. systemd-Daemon neu laden, um neue/geänderte Dateien zu erkennen
sudo systemctl daemon-reload

# 2. Dienst aktivieren, damit er beim Systemstart automatisch startet
sudo systemctl enable cmbnet.service

# 3. Dienst sofort starten
sudo systemctl start cmbnet.service

# 4. Status prüfen, ob alles fehlerfrei läuft
sudo systemctl status cmbnet.service
```

4. Caddy Konfigurationsdatei für den Reverse Proxy

Caddy ist extrem einfach zu konfigurieren und übernimmt automatisch die HTTPS-Zertifikate (Let's Encrypt). Da Caddy als Reverse Proxy vor deiner Flask-App läuft, leitet er externen Traffic (Ports 80/443) an den internen Gunicorn-Server auf Port `5001` weiter.

Öffne deine Caddy-Konfigurationsdatei (üblicherweise unter `/etc/caddy/Caddyfile`):

```
sudo nano /etc/caddy/Caddyfile
```

Trage dort deine Domain (oder IP, falls lokal getestet) ein:

```
deine-domain.de {
    # Leitet alle Anfragen an die Flask-App (Gunicorn) auf Port 5001 weiter
    reverse_proxy 127.0.0.1:5001
}
```

Caddy-Konfiguration anwenden:

Speichere die Datei und lade Caddy neu:

```
sudo systemctl reload caddy
```

Caddy kümmert sich nun vollautomatisch darum, Anfragen anzunehmen, via HTTPS abzusichern und an deine Flask-Anwendung im Hintergrund weiterzureichen.

5. Zukünftiges Deployment (Updates aus Git einspielen)

Wenn du später Änderungen an deinem Code im Git-Repository vornimmst, kannst du sie mit einem kurzen Skript oder manuell in wenigen Schritten auf dem Server aktualisieren:

```
cd /opt/cmbnet/app
sudo -u cmbnet git pull
sudo -u cmbnet ./venv/bin/pip install -r requirements.txt
sudo systemctl restart cmbnet.service
```
