I7 Teamverwaltung - Entwicklerdokumentation

Inhaltsverzeichnis

1. Einführung

Dieses Dokument beinhaltet detaillierte Informationen zum Aufbau der Software und ihrer Komponenten, um die Einarbeitung anderer Entwickler zu erleichtern. Dieses Dokument wird durch eine HTML Seite für die Dokumentation der Python Django Module erweitert (mehr dazu unter Punkt Django).

2. Frameworks

Folgende Frameworks und Bibliotheken wurden für die Python Entwicklung verwendet:

- wheel
- django
- mysqlclient
- django-mysql
- django-widget-tweaks
- ortools

- django-auth-ldap
- gunicorn
- django-debug-toolbar (nur für Debugging)
- sphinx (nur für Python Dokumentation)
- sphinx-rtd-theme (nur für Python Dokumentation)

3. Architecture Notebook

3.1. Zweck

Dieses Dokument beschreibt die Philosophie, Entscheidungen, Nebenbedingungen, Begründungen, wesentliche Elemente und andere übergreifende Aspekte des Systems, die Einfluss auf Entwurf und Implementierung haben.

3.2. Architekturziele und Philosophie

Das System ist eine Webanwendung, welche zur Verwaltung und automatisierten Erstellung der Projektteams sowie der Bereitstellung und Bearbeitung des Fragebogens im Modul Software Engineering I an der HTW-Dresden dient. Die Anwendung ist nur im HTW-internen Netz verfügbar.

Das System arbeitet mit personenbezogenen Daten wie Bibliotheksnummer, Name und Studiengang. Deshalb ist es wichtig, dass die Daten datenschutzrechtlichen Belangen entsprechend konform und geschützt vor Zugriff von Unbefugten abgespeichert werden.

Um den organisatorischen Aufwand der Planung der Teams möglichst gering zu halten, ist eine einfache und intuitive Bedienbarkeit sowohl für Studenten als auch für Dozenten erforderlich. Neben guter Responsivität soll dies mit einer klaren Menüführung und einem an andere bekannte Websites angelehntes Design erzielt werden. Des Weiteren sollen Icons und kurze Nutzeranweisungen an entscheidenden Stellen eingesetzt werden.

3.3. Annahmen und Abhängigkeiten

Annahmen:

- Jeder Benutzer besitzt einen Zugang ins HTW-interne Netzwerk.
- Jeder Benutzer verfügt über eine stabile Verbindung mit genügend Bandbreite zu unserem Webserver.
- Die von uns eingesetzten Frameworks funktionieren ohne schwerwiegende Fehler und werden in Zukunft weiter gepflegt.
- Jeder Benutzer verfügt über einen HTML5 fähigen Webbrowser.
- Jeder Student im Modul Software Engineering besitzt einen gültigen HTW-Login für Studenten.

Abhängigkeiten:

- Wir sind auf das Rechenzentrum der HTW angewiesen, da die Dienste unseres Servers fehlerfrei bereitgestellt werden müssen.
- Wir sind darauf angewiesen, dass die von uns verwendeten Frameworks in Zukunft weiter gepflegt werden.

3.4. Architektur-relevante Anforderungen

Nicht Funktional:

Systemweite Anforderung:

[SWFA-2] [SWFA-3] [SWFA-5] [SWFA-6] [SWFA-7] [SWFA-8]

Benutzbarkeit (Usability):

[NFAU-1] [NFAU-3] [NFAU-4]

Zuverlässigkeit (Reliability):

[NFAR-1] [NFAR-2] [NFAR-3]

Effizienz (Performance):

[NFAP-3]

3.5. Entscheidungen, Nebenbedingungen und Begründungen

1. Wir nutzen Python, da
   a. Python reich an verschiedenen weit verbreiteten und gut gepflegten Frameworks ist.
   b. einfach zu verstehen bzw. zu programmieren ist, wodurch der Code leichter nachvollziehbar und erweiterbar ist.
2. Wir nutzen das Framework Django, da
   a. es eine weit verbreitetes und gut dokumentiertes Web-Framework ist.
   b. es die Verbindung zur Datenbank für uns übernimmt
   c. es eine sehr einfache Möglichkeit der Datenabfrage ohne spezifische SQL Befehle durch den mitgelieferten ORM mit dem Datenbank-Server ermöglicht.
3. Wir nutzen MySQL als Datenbank, da
   a. es eine weit verbreitetes und gut dokumentiertes DBMS ist.
   b. es offiziell mit Django kompatibel ist.
4. Wir nutzen Bootstrap das Interface, da
   a. es simpel in Django integrierbar und nutzbar ist.
   b. es uns durch vorgefertigte Design-Patterns erlaubt, schnell eine moderne Nutzeroberfläche

ohne viele manuelle Anpassungen zu erstellen.

5. Wir nutzen einen Constraint Solver für den Zuordnungsalgorithmus, da
   a. dieser die besten Zuordnungsergebnisse in kurzer Zeit liefert.
   b. der Algorithmus damit relativ einfach um neue Restriktionen erweitert werden kann.
6. Wir nutzen das Framework OR-Tools für den Constraint Solver, da
   a. dies gut dokumentiert ist.
   b. das Hinzufügen von Restriktionen sehr simpel ist.
7. Wir nutzen Apache als Webserver, da dieser eine schnelle und zuverlässige Bereitstellung der HTML-Files ermöglicht.

3.6. Architekturmechanismen

Doku "Concept: Architectural Mechanism"

1. Zur dauerhaften Speicherung der Fragebogen- und Zuordnungsdaten greifen wir auf die relationale Datenbank MySQL zurück. Diese Entscheidung haben wir getroffen, da sich die zu sichernden Daten gut in einem relationalen Schema darstellen lassen. MySQL bietet dabei eine weit verbreitete Lösung zur zuverlässigen und sicheren Speicherung der Daten. Außerdem kann durch das individuell einstellbare Rechtesystem in der Datenbank der Zugriff vor Unbefugten geschützt werden (siehe nicht funktionale Anforderungen).
2. Wir haben die Entscheidung getroffen, dass die Funktionalität der Buttons einheitlich visuell dargestellt werden soll ([NFAU-4]), da dies die Usability für die Nutzer erhöht. Dabei hatten wir entweder die Möglichkeit das User Interface manuell per CSS von Grund auf neu zu entwickeln oder ein Framework mit vorgefertigten Design Elementen zu wählen. Das manuelle Designen mittels CSS bietet zwar sehr großen Freiraum und nahezu unbegrenzte Gestaltungsmöglichkeit, jedoch kostet dieses Vorgehen auch sehr viel Zeit. Deshalb haben wir uns für die zweite Möglichkeit entschieden und das Design mit Bootstrap umgesetzt. Dies schränkt zwar die Gestaltungsmöglichkeiten ein, bietet für unsere Anforderungen jedoch ausreichend Freiraum und erspart uns sehr viel Zeit.
3. Für die Teamzusammenstellung werden folgende Methodiken in Betracht gezogen: Zuordnen & Vertauschen Algorithmus, genetischer Algorithmus und Algorithmus mittels Constraint Solver. Letztendlich haben wir uns für einen Algorithmus mittels Constraint Solver entschieden, da dieser es uns ermöglicht schnell neue Restriktionen (Constraints) hinzuzufügen. Dies erleichtert die Entwicklung und auch eine eventuelle Weiterentwicklung unseres Projekts in Zukunft. Außerdem schnitt der Algorithmus mit Constraint Solver im Vergleich zu den anderen Varianten in Sachen Lösungsqualität als auch Lösungszeit am besten ab.

3.7. Wesentliche Abstraktionen

- Student: Enthält Stammdaten (s-Nummer, Name, Studiengang und weiteres) zu den belegbearbeitenden Studenten und verweist auf deren Fragebogenergebnisse. In Verbindung mit der s-Nummer können die Logindaten der Studenten abgeglichen werden.
- Projektthema: Enthält mehrere Textinhalte zur Beschreibung des Projekts sowie einen Link,

unter welchem eine ausführliche Info-pdf abgelegt werden kann.

- Teamzuordnung: Enthält Informationen darüber welcher Student welchem Projekt und welcher Rolle zugeteil ist.

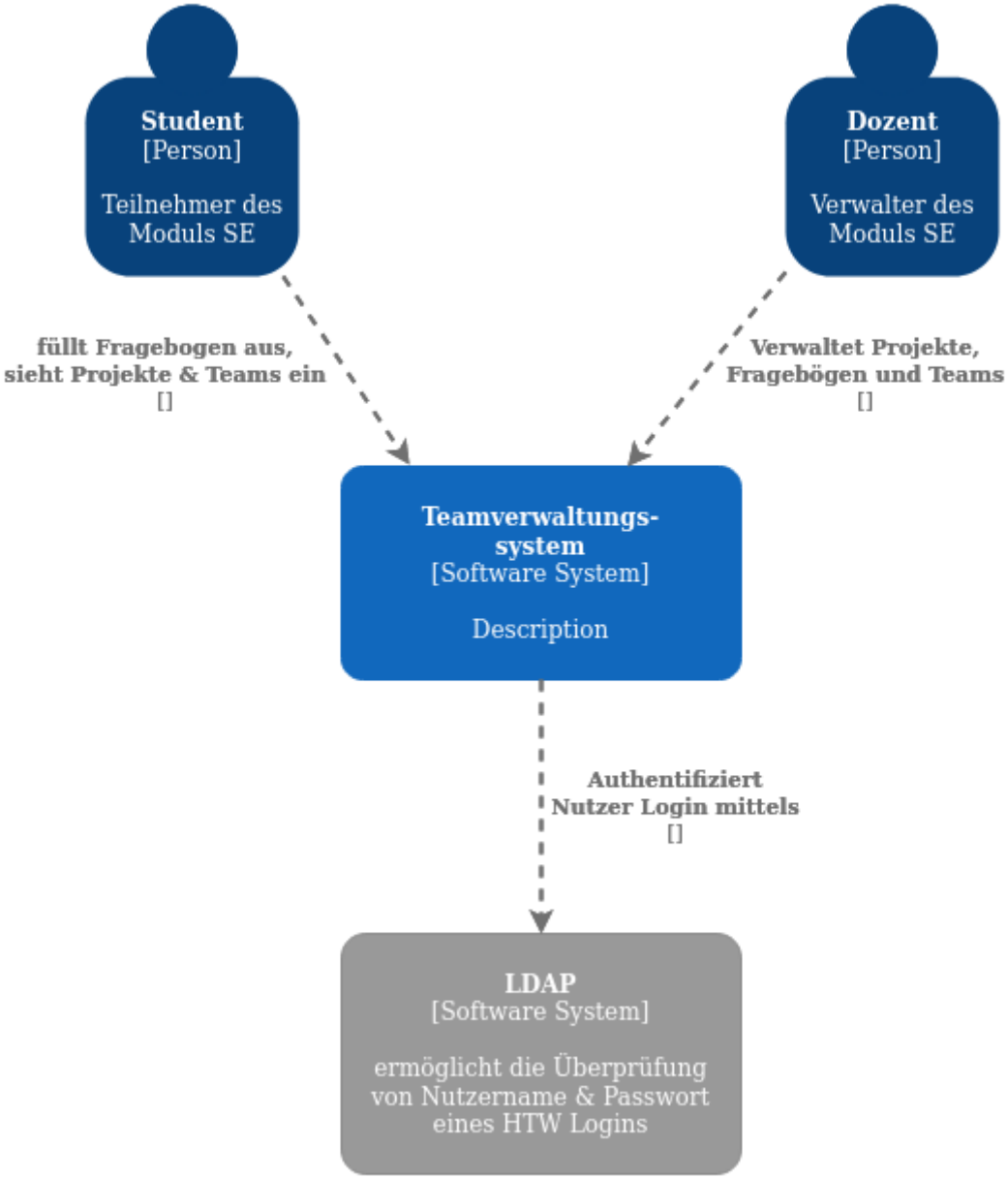
3.8. Architektursichten (Views)

Im Nachfolgenden werden die Architektursichten anhand der ersten beiden Levels des C4 Modells erläutert.

3.8.1. Logische Sicht

Wie im 1. Level des C4 Modells zu erkennen, dient das Teamverwaltungssystem als zentraler Interaktionspunkt für die beiden Akteuere (Dozenten und den Student). Die einzige externe Schnittstelle besteht zum LDAP Server der HTW Dresden. Diese Schnittstelle erlaubt es, Studenten mit ihrer s-Nummer und ihrem HTW Login zu authentifizieren. Die Authentifizierung der Studenten läuft dabei über das Teamverwaltungssystem ab. Diese Entscheidung hat den Grund, dass auf Weiterleitungen zu externe Websites verzichtet werden soll und somit eine konsistente User Experience garantiert wird. Das heißt der Nutzer kann alle Funktionen des Software Systems nutzen, ohne die Teamverwaltungswebsite zu verlassen.

C4 Modell - Level 1:


Student
[Person]
Teilnehmer des Moduls SE
Dozent
[Person]
Verwalter des Moduls SE
füllt Fragebogen aus, sieht Projekte & Teams ein
[]
Verwaltet Projekte, Fragebögen und Teams
[]
Teamverwaltungs-system
[Software System]
Description
Authentifiziert Nutzer Login mittels
[]
LDAP
[Software System]
ermöglicht die Überprüfung von Nutzername & Passwort eines HTW Logins

3.8.2. Physische Sicht (Betriebssicht)

Wie im 2. Level des C4 Modells sichtbar, besteht das Teamverwaltungssystem aus drei Containern:

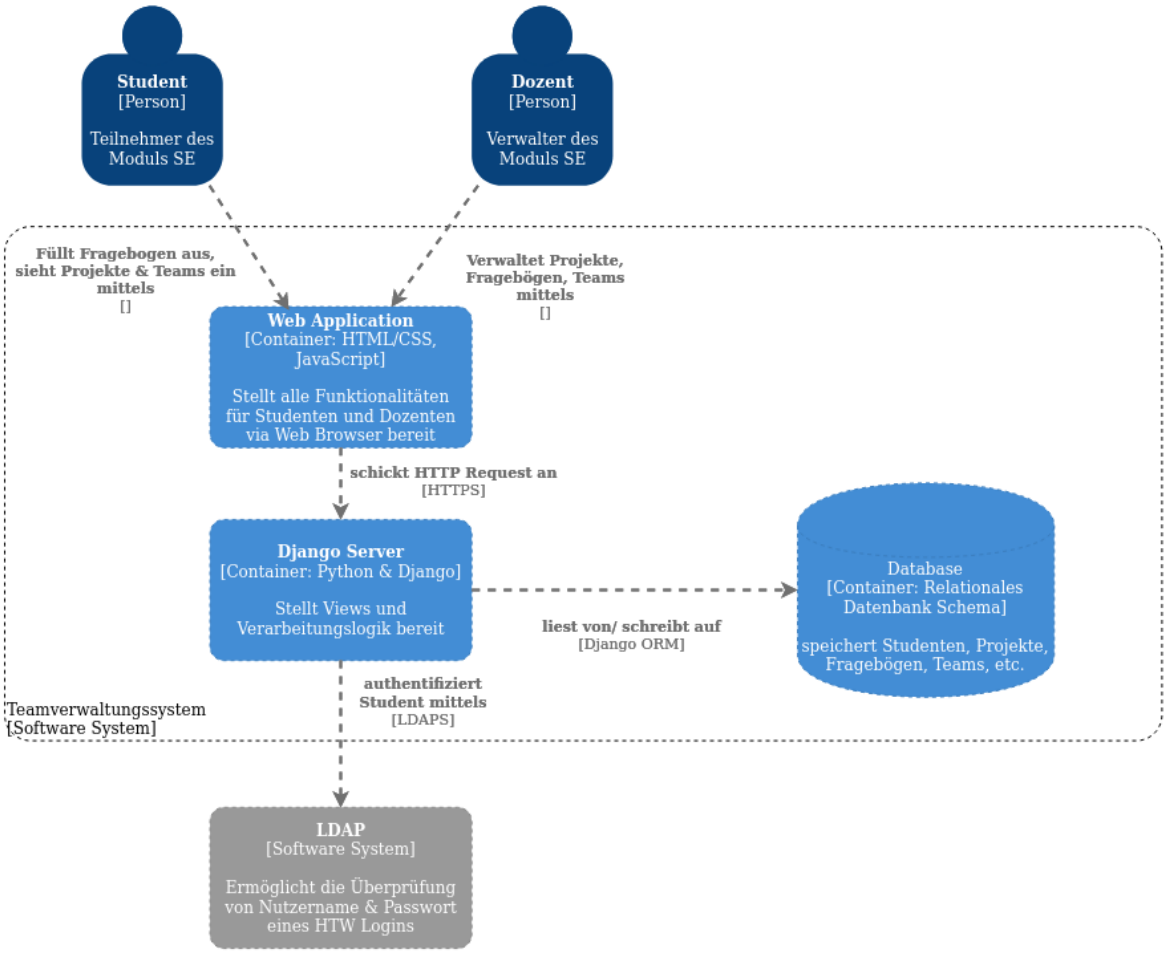
- Web-Application
- Django Server
- Datenbank

Dieser Grundaufbau ist durch das von uns verwendete Framework Django mit dem Model View Controller vorgegeben. Die Web Application, welche mithilfe von HTML/CSS und JavaScript umgesetzt wurde läuft dabei im Browser des Nutzers. Die Web Application ist somit der zentrale Interaktionspunkt für die beiden Akteure (Student und Dozent). Durch diese können alle vom Teamverwaltungssystem bereitsgestellten Funktionen wie z.B. das Ausfüllen des Fragebogens oder das Verwalten der Projekte, genutzt werden.

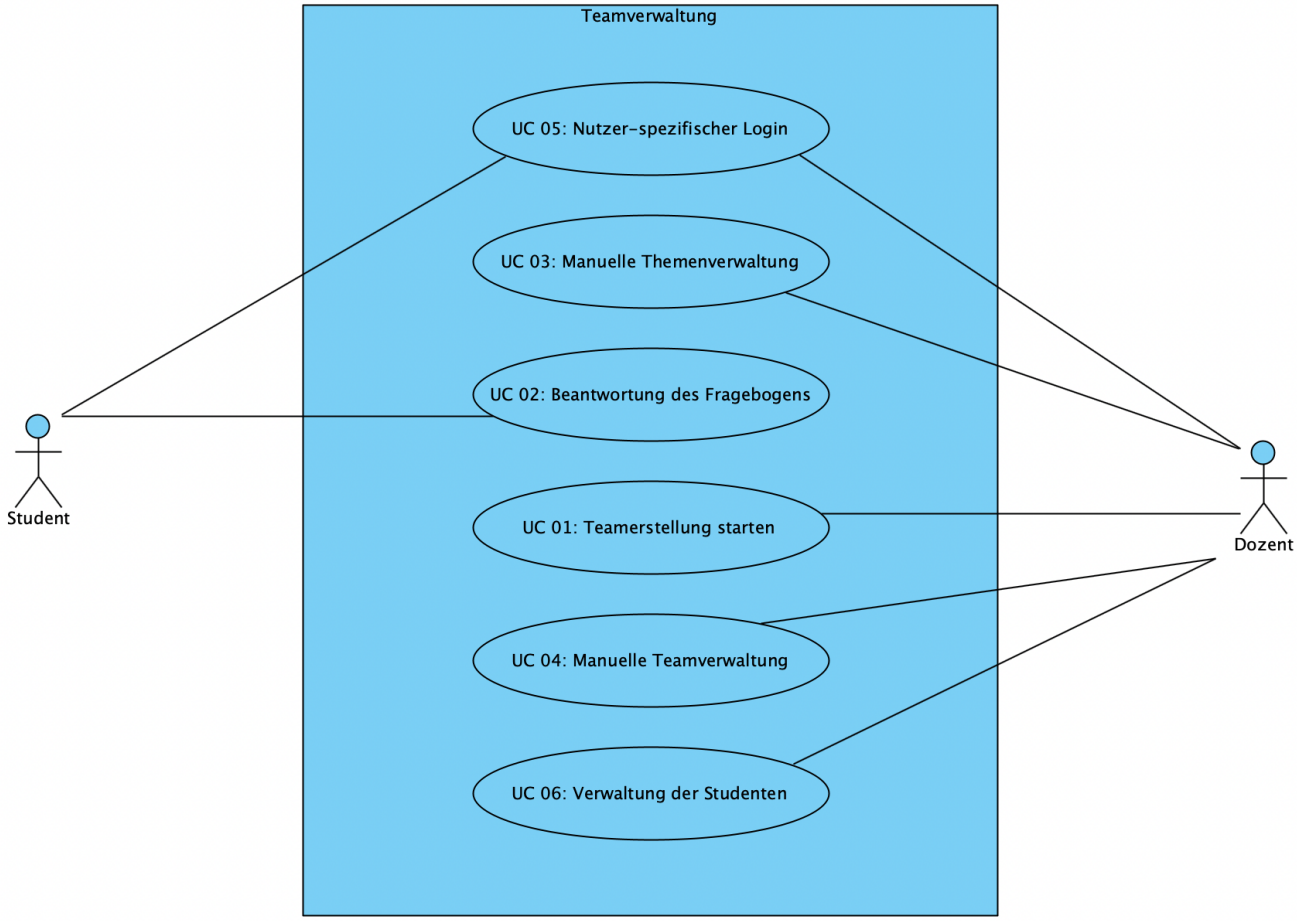
Der Django Server stellt das Backend mitsamt seiner Logik für die Teamverwaltungssoftware bereit. Die Web Application kommuniziert mithilfe von HTTP Requests bzw. HTTP Responses mit dem Django Server. Der Django Server rendert dabei entweder die HTML Views und schickt diese an die Web Application oder er dient als API für JSON Anfragen mittels Ajax (genaueres dazu ist in der Entwicklerdokumentation zu finden). Durch die Requests seitens der Application an den Django Server können mithilfe der Views gewisse Logiken wie z.B. das Abfragen von Daten oder das Ausführen des Zuordnungsalgorithmus getriggert werden. Zusätzlich dazu kommuniziert der Django Server noch mit dem LDAP Server der HTW, um die Studenten zu authentifizieren. Ein LDAP Plugin für Django erlaubt das automatisierte Handling der LDAP Anfragen.

In der Datenbank werden grundsätzlich alle Daten abgelegt, welche persistent gespeichert werden müssen. Das beinhaltet bspw. die Projekte inkl. Kurzbeschreibung, die Antworten aus den Fragebögen oder auch die Login Daten des Dozenten Accounts, usw. Der Dozent wird nicht wie Studenten mittels LDAP, sondern via Django Authentication authentifiziert. Diese Entscheidung rührt daher, dass durch LDAP nicht eindeutig festgestellt werden kann, welcher Nutzer ein berechtigter Dozent oder Mitarbeiter des Moduls Software Engineering I/II ist und dieses verwalten darf. Durch einen einheitlichen Dozenten Account wird diese Problematik gelöst. Das Verwenden von Django Authentication hat zusätzlich den Vorteil, dass Django alle nötigen Sicherheitsvorkehrungen, wie Hashing des Passwortes oder die Möglichkeit zum Ändern des Passwortes bereitstellt. Alle Abfragen bzw. Schreiboperationen, die auf dem Django Server durch die Views getriggert werden laufen über den mitgelieferten Object Relational Mapper (ORM).

C4 Modell - Level 2 (Teamverwaltungssystem):


Student
[Person]
Teilnehmer des Moduls SE
Dozent
[Person]
Verwalter des Moduls SE
Füllt Fragebogen aus, sieht Projekte & Teams ein mittels
[]
Verwaltet Projekte, Fragebögen, Teams mittels
[]
Web Application
[Container: HTML/CSS, JavaScript]
Stellt alle Funktionalitäten für Studenten und Dozenten via Web Browser bereit
schickt HTTP Request an
[HTTPS]
Django Server
[Container: Python & Django]
Stellt Views und Verarbeitungslogik bereit
liest von/ schreibt auf
[Django ORM]
Database
[Container: Relationales Datenbank Schema]
speichert Studenten, Projekte, Fragebögen, Teams, etc.
authentifiziert Student mittels
[LDAPS]
Teamverwaltungssystem
[Software System]
LDAP
[Software System]
Ermöglicht die Überprüfung von Nutzername & Passwort eines HTW Logins

3.8.3. Use cases



4. Server

In diesem Abschnitt wird jede Komponente des Servers und ihre Rolle erläutert.

Im Beispiel läuft der Server unter Ubuntu, aber dieses Dokument kann auch analog für andere Distro (Arch Linux, OpenSuse, Fedora, usw.) verwendet werden. Für die Installation des benötigen Pakets folgen Sie bitte den Anweisungen der Distribution. Außerdem kann alles, was in Docker läuft, auch ohne Docker laufen.

4.1. Bestandteile des Servers

4.1.1. Uncomplicated Firewall

Uncomplicated Firewall schützt den Webserver vor unerwünschten Netzwerkzugriffen und ist unkompliziert und einfach zu bedienen. Installieren Sie das Paket `ufw` und aktivieren `ufw.service`, um es beim Booten verfügbar zu machen. Danach führen Sie die folgenden Kommandos aus, um standardmäßig alles zu verweigern, Port 80 und 443 und ratenbeschränkten SSH-Verkehr jedoch von überall zu zulassen.

```
ufw default deny
ufw allow 80
ufw allow 443
ufw limit ssh
ufw enable
```

4.1.2. Open-SSH

Nach dem Setup muss sich der SysAdmin oder Developer regelmäßig mit dem Server mittels ssh verbinden. Normalerweise muss zuerst eine VPN-Verbindung hergestellt werden, aber ab OpenSSH-Version 7.3 existiert ein weiterer Weg namens ProxyJump. Es ist zu beachten, dass die Pfade zu der Datei unter *nix (Linux, MacOS) und unter Windows 10/11 unterschiedlich sind. Die Kommandos verbleiben aber gleich. In Windows 10/11 muss die SSH-Dateien z.B. unter C:\User\Flick\.ssh gespeichert werden.

Angenommen, der Server ilux150.informatik.htw-dresden.de ist ilux150 und der Webserver ise1.

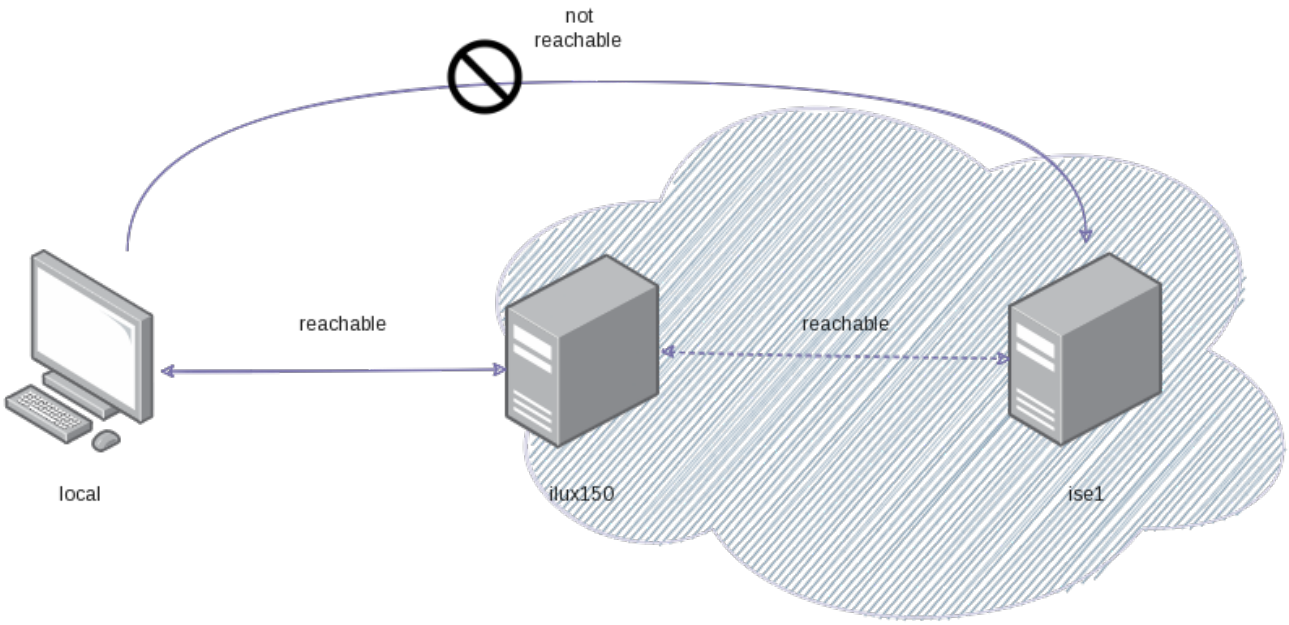


Abbildung 1. Pseudo-Netzwerk

Erzeugen SSH-Schlüsselpaar

Authentifizierung unter Einsatz von Public-Key-Kryptographie ist nicht anfällig für Brute-Force-Angriffe und wenn der Server kompromitiert wurde, bekommt der Angreifer keine Anmeldendaten. Zwei SSH-Schlüsselpaare können durch Ausführen des Befehles ssh-keygen erzeugt werden. Einer für ilux150 und der andere für ise1.

```
mkdir -p ~/.ssh/keys
ssh-keygen -t ed25519 -f ~/.ssh/keys/ilux150
ssh-keygen -t ed25519 -f ~/.ssh/keys/ise1
```

Der öffentliche Schlüssel hat den gleichen Namen wie der private Schlüssel, nur dass er mit der Erweiterung .pub versehen ist. Außerdem findet sich das SSH-Schlüsselpaar unter **~/.ssh/keys**

SSH-Config

Der Client kann so kofiguiert werden, dass er allgemeine Optionen und Hosts speichert. Alle Optionen können global oder auf bestimmte Hosts beschränkt angegeben werden. (~/.ssh/config)

```
host *
    SetEnv TERM=xterm-256color
    AddKeysToAgent yes
    PubKeyAuthentication yes
    IdentitiesOnly yes

Match host ilux150
    Hostname ilux150.informatik.htw-dresden.de
    User s82053
    #Port 22
    IdentityFile ~/.ssh/keys/ilux150
```

```
    PasswordAuthentication no

Match host ise1
    Hostname iseproject01.informatik.htw-dresden.de
    User seproject
    #Port 22
    ProxyJump ilux150
    IdentityFile ~/.ssh/keys/ise1
```

Kopieren des öffentlichen Schlüssel auf den Server

Dann muss der öffentliche Schlüssel auf den Server kopiert werden, damit dieser die SSH-Schlüsselauthentifizierung verwenden kann. Beachten Sie, dass der private Schlüssel auf dem lokalen Rechner verbleiben muss.

```
ssh-copy-id -i ~/.ssh/keys/ilux150.pub ilux150
ssh-copy-id -i ~/.ssh/keys/ise1.pub ise1
```

Alternative ist

```
cat ~/.ssh/keys/ilux150.pub | ssh ilux150 "mkdir ~/.ssh; chmod 700 ~/.ssh; cat >>
~/.ssh/authorized_keys"
cat ~/.ssh/keys/ise1.pub | ssh ise1 "mkdir ~/.ssh; chmod 700 ~/.ssh; cat >>
~/.ssh/authorized_keys"
```

Jump Hosts

Da keine direkte Verbindung zu ise1 besteht, ist die Verwendung eines Jump-Servers erforderlich. Deswegen versuchen wir, zwei oder mehr SSH-Tunnel miteinander zu verbinden. Dies ist mit Hilfe des SSH-Agent-Forwarding (-A) und der Pseudo-Terminal-Allocation (-t) möglich, die die lokalen Schlüsseln weiterleiten.

```
ssh -J ilux150 ise1
```

```
ssh -J jump1,jump2,jump3 dest
```

Ein Äquivalent zu dem Flag -J in der SSH-Config-Datei ist die Option `ProxyJump`.

```
ssh ise1
```

Verschlüsselter SOCKS-Tunnel

Mit Hilfe von SSH kann die Webseite ohne VPN-Verbindung erreichbar sein. Außerdem ist diese Technik sehr nützlich für Rechner, die über unsichere Verbindungen verbunden sind.

```
ssh -TND 4711 ise1
```

Konfiguriert den Webbrowser (oder andere Programme)

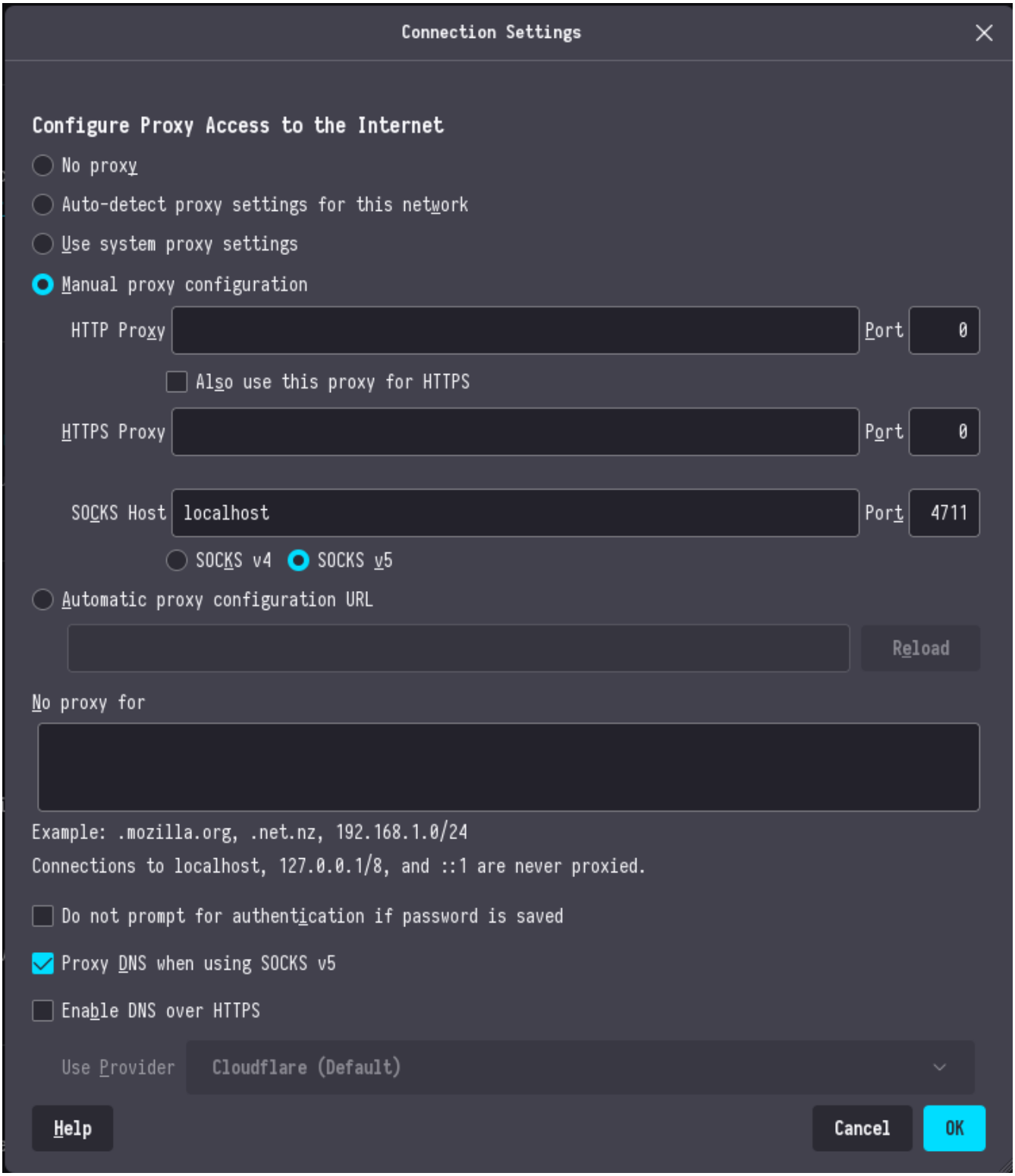


Abbildung 2. Firefox

4.1.3. Docker

Um Docker und Docker Compose aufzubauen und zum Laufen zu bringen, werden Docker-Engine, Docker CLI und Docker Compose benötigt. In Windows und MacOS gibt es nur eine Methode dazu,

welche Docker Desktop heißt. Docker Desktop ist eine proprietäre Anwendung, die alle wesentliche Komponenten enthält und innerhalb einer virtuellen Linux-Maschine ausführt. Deswegen gibt es für Windows-Nutzer eine andere Alternative Windows Subsystem for Linux, welche Docker unter WSL lauffähig macht. In Linux können das Paket `docker` und `docker-compose` einfach mittels Package Management installiert werden. Anschließend starten und aktivieren Sie `docker.service`.

Mit Docker ist es möglich, schnell bereitzustellen und einfach zu testen. Wenn Docker-Compose lokal ohne Fehler läuft, dann läuft er auch ohne Fehler beim Server.

```
systemctl enable --now docker.service
```

Normalerweise darf nur der Root-User docker ausführen. Damit fügen Sie Ihren aktuellen User zur docker (user group) hinzu, melden sich erneut an und starten erneut `docker.service`.

```
usermod -aG docker username
systemctl restart docker.service
```

Docker Compose ist eine alternatives CLI-Frontend für die Docker-Engine. Dies ist für die komplexe Konfiguration von Docker Containern nützlich.

```
docker-compose up --build -d
```

Dieses Kommando startet Container und baut diese bei Bedarf auf.

```
docker-compose down
```

Deshalb ermöglichen Docker und Docker-Compose die schnelle Bereitstellung eines komplizierten Systems. Entwickler können auch Docker verwenden, um zu testen, ob die neue Version des Quellcodes in der Produktion funktioniert, bevor sie dieser auf den Server übertragen wird. Es ist auch einfach die Module (Service) auszutauschen, hinzufügen oder zu entfernen.

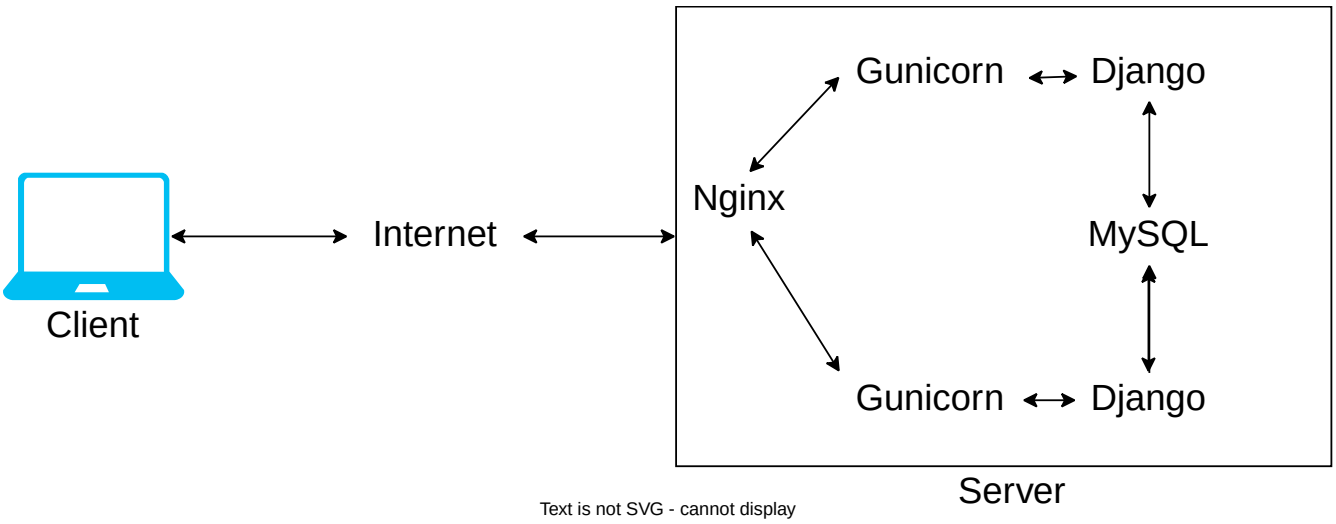


Abbildung 3. Docker Compose für die Webseite Teamverwaltung

Der untere Docker-Compose-Code stellt die obige Abbildung dar.

```
version: "3.8"

services:
  proxy:
    image: nginx
    container_name: nginx-proxy
    restart: on-failure
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./docker/nginx/:/etc/nginx/:ro
      - static_volume:/var/www/static:ro
    depends_on:
      - web
  web:
    build:
      context: .
      dockerfile: ./docker/django/Dockerfile
    restart: on-failure
    command: bash -c "/usr/local/bin/wait && /usr/local/bin/start"
    volumes:
      - ./src/teamverwaltung/:/var/django/app:ro
      - ./docker/nginx/ssl/:/var/www/ssl/:ro
      - ./docker/django/gunicorn.py:/var/gunicorn/gunicorn.py:ro
      - ./docker/django/wait:/usr/local/bin/wait:ro
      - ./docker/django/start:/usr/local/bin/start:ro
      - static_volume:/var/www/static:rw
    deploy:
      mode: replicated
      replicas: 2
    depends_on:
      - db
  db:
    image: mysql
    container_name: mysql-db
    restart: on-failure
    volumes:
      - ./docker/db/django.sql:/docker-entrypoint-initdb.d/django.sql:rw
      - mysql-data:/var/lib/mysql:rw
    environment:
       MYSQL_DATABASE: 'django'
       MYSQL_USER: 'dev'
       MYSQL_PASSWORD: 'Er3dyWTM#a8_HE#'
       MYSQL_ROOT_PASSWORD: '05912065344aae'

volumes:
  static_volume:
```

```
  mysql-data:
```

Der Client kann nur mit Nginx auf Port 80 und 443 kommunizieren, wenn er Nginx einen Request schickt, leitet Nginx ihn an Gunicorn weiter, der der aktuelle Webserver ist. Gunicorn übersetzt die Anfrage in etwas, das Django verstehen kann. Django arbeitet mit MySQL und sendet Gunicorn die Antwort, der diese weiter an Nginx befördert. Danach schickt Nginx dem Client die Antwort.

Warum muss das System sehr kompliziert sein? Django ist nur ein Framework, das einen Großteil der Webentwicklung abnimmt, so dass sich die Programmierer auf die Entwicklung der Anwendung konzentrieren können, ohne das Rad neu erfinden zu müssen. Aber er hat einen Nachteil, dass er nicht ein aktueller Webserver ist, d.h. die Sicherheit ist wirklich gering und es ist nicht für den Einsatz in der Produktion geeignet. Deshalb wird ein aktueller Webserver (Gunicorn) benötigt. Die schnelle Entwicklung hat einen Preis: Manchmal ist Django nicht schnell genug. Das bedeutet, dass der Benutzer einige Minuten warten muss, bis die Seite geladen ist. Die Lösung gegen diese Langsamkeit sind Threads und Lastverteilung. Anwendungen, die Multicore- oder Multi-CPU-Systeme nutzen möchten, können Multithreading verwenden, um Daten und Aufgaben in parallele Teilaufgaben aufzuteilen und die zugrunde liegende Architektur die Ausführung der Threads verwalten zu lassen, entweder gleichzeitig auf einem Kern oder parallel auf mehreren Kernen. Mittels Lastverteilung (englisch: Load Balancing) werden große Mengen von Anfragen auf mehrere parallel arbeitende Systeme verteilt mit dem Ziel, ihre gesamte Verarbeitung effizienter zu gestalten. Gunicorn unterstützt Threads und Nginx ist die einfachste Antwort der Lastverteilung.

4.1.4. Nginx

Nginx ist ein Reverse-Proxy, das vor den Backend-Anwendungen sitzt und Cient-Anfragen an diese weiterleitet. Eine Aufgabe kann ein Reverse-Proxy erfüllen, der die Anfragen entgegennimmt, um die Geschwindigkeit zu verbessern und funktionell zu erweitern. Die an den Client zurückgegebenen Antworten sehen aus, als kämen sie vom Webserver selbst. Damit kann Reverse-Proxy die Existenz und die Eigenschaften von Ursprungsservern verbergen. Außerdem verwendet die Webseite ihn zusammen mit anderen Techniken, um die Last auf die internen Server zu verteilen. Reverse-Proxy kann statische Inhalte auf dem Cache speichern, um die Belastung dieser internen Server und des internen Netzes weiter zu verringern. Es ist auch üblich, dass Komprimierung oder TLS-Verschlüsselung zwischen dem Client und dem Reverse-Proxy eingebaut werden. Bei sicheren Webseiten kann es vorkommen, dass die Webseite die TLS-Verschlüsselung nicht selbst durchführt, sondern diese Aufgabe an einen Reverse-Proxy überträgt. Es gibt eine Technik, die "Spoon feeding" heißt, deswegen kann eine dynamische Seite (HTML, CSS, JS, usw.) einmal generiert und an den Proxy Server geschickt werden. Danach sendet der Proxy Server sie an den Client zurück und muss die aktuelle Webseite nicht göffnen, das heißt die Serverressourcen werden entlastet. Mittels Reverse-Proxy kann ein Webserver, der über keine Authentifizierung verfügt, eine Zugangsauthentifizierung hinzufügen.

Um nginx zu verwenden, installieren Sie das Paket `nginx` und starten das `nginx.service`. Der zweite Schritt besteht darin, die Konfigurationsdatei `/etc/nginx/nginx.conf` zu ändern. In dem Repository unter `docker/nginx/nginx.conf` finden Sie ein Beispiel.

```
# Source: https://docs.nginx.com/nginx/admin-guide/web-server/
worker_processes auto;
worker_cpu_affinity auto;
```

```
events {
    multi_accept on;
    accept_mutex on;
    worker_connections 1024;
}

http {
    sendfile on;
    sendfile_max_chunk 1m;
    ssl_ciphers
"EECDH+CHACHA20:EECDH+AES128:RSA+AES128:EECDH+AES256:RSA+AES256:EECDH+3DES:RSA+3DES:!M
D5";
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3;
    add_header Strict-Transport-Security "max-age=15768000; includeSubDomains" always;
    include mime.types;
    default_type application/octet-stream; # fallback in case we can't determine a
type
    client_max_body_size 5G;
    error_log /var/log/nginx/error.log warn;
    access_log /var/log/nginx/access.log combined;
    proxy_cache_path /tmp/cache keys_zone=mycache:10m;
    #limit_conn_zone $binary_remote_addr zone=ip_addr:10m; # Let's prevent DoS

    upstream gunicorn.django {
        least_conn;
        server web:8000;
        #server i7_teamverwaltung-web-3:8000 backup;
    }

    server {
        # if no Host match, close the connection to prevent host spoofing
        listen 80 default_server;
        return 444;
    }

    server {
        listen 80;
        listen [::]:80;
        server_name iseproject01.informatik.htw-dresden.de;
        keepalive_timeout 5;
        return 301 https://$host$request_uri;
    }

    server {
        listen 443 ssl http2;
        listen [::]:443 ssl http2;
        server_name iseproject01.informatik.htw-dresden.de;
        proxy_cache mycache;
        keepalive_timeout 5;
        ssl_certificate ssl/server.crt;
```

```
        ssl_certificate_key ssl/server.key;
        #limit_conn ip_addr 20;

        location / {
            proxy_set_header   Host $host;
            proxy_set_header   X-Forwarded-Proto $scheme;
            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_redirect     off;
            proxy_connect_timeout   300;
            proxy_send_timeout      300;
            proxy_read_timeout      300;
            proxy_cache_valid any 1m;
            proxy_cache_min_uses 3;
            proxy_pass https://gunicorn.django;
        }
        location /static {
            autoindex on;
            alias /var/www/static/;
        }
    }
}
```

Nginx hört auf Port 80 und wenn ein Client einen Request an diesen Port schickt, gibt Nginx den Status-Code 301 zurück, der Move Permanently bedeutet. In diesem Fall befindet sich die Webseite auf Port 443 (HTTPS). Die Einstellung `worker_processes` legt fest, wie viele Verbindungen Nginx akzeptiert und wie viele Prozessoren es nutzen kann. Standardmäßig wickelt Nginx die Dateiübertragung selbst ab und kopiert die Datei in den Puffer, bevor sie gesendet wird. Durch die Einstellung `sendfile` entfällt der Schritt des Kopierens der Daten in den Puffer und ermöglicht das direkte Kopieren. Um zu verhindern, dass eine schnelle Verbindung den Worker-Prozess vollständig auslastet, wird `sendfile_max_chunk` verwendet. Sie begrenzt die in einem einzigen sendfile()-Aufruf übertragene Datenmenge. Die Verwendung des Lastausgleich des Verkehrs auf eine Gruppe von Servern oder Services ist in `upstream` definiert. Der Algorithmus `Round Robin` wird standardmäßig verwendet. Die Anfragen werden gleichmäßig auf die Server verteilt, wobei die Servergewichtung `weight` berücksichtigt wird. In dem Beispiel wird die Methode `Least Connections` genutzt. Hier wird eine Anfrage an den Server mit der geringsten Anzahl aktiver Verbindungen gesendet, wobei auch die Servergewichtung berücksichtigt wird. Wenn einer der Server vorübergehend aus der Lastausgleichsrotation herausgenommen werden muss, kann er mit `down` markiert werden, z.B. `server i7_teamverwaltung-web-2:8000 down;`. Die Anfragen, die von diesem Server bearbeitet werden sollten, werden automatisch an den nächsten Server in der Gruppe weitergeleitet. Der Server `i7_teamverwaltung-web-3` ist als Backup-Server gekennzeichnet und erhält nur dann Anfragen, wenn beide anderen Server nicht verfügbar sind. Außerdem verhindert die Einstellung `slow_start`, z.B.: `slow_start=30s` zum langsamen Starten des Servers, dass ein kürzlich wiederhergestellter Server von Verbindung überflutet wird, die ein Zeitüberschreitung verursachen und dazu führen können, dass der Server erneut als ausgefallen markiert wird. Die Optionen `proxy_cache_valid any 1m` und `proxy_cache_min_uses 3` bedeuten, dass die Anfragen nur eine Minute lang gültig sind und erst dann zwischengespeichert werden, wenn dieselbe Anfrage drei Mal gestellt wird. Um einen HTTPS-Server einzurichten, fügen Sie in `nginx.conf` den ssl-Parameter ein und geben dann die Speicherorte der Server-Zertifikats- und privaten Schlüsseldateien an. Sie können das Zertifikat von HTW bekommen.

4.1.5. Gunicorn / Django

Django ist ein auf Python basierendes Framework für Backend-Webanwendungen. Eine der Django-Philosophien (Don't Repeat Yourself) bedeutet, dass Entwickler vorhandenen Code wiederverwenden und sich so auf das Einzigartige konzentrieren können.

Nginx ist der Außenwelt zugewandt. Es wird die statischen Dateien (Bilder, CSS, usw.) direkt aus dem Dateisystem und aus dem Cache bereitstellen. Allerdings kann er nicht mit Django kommunizieren; er braucht etwas, das Django ausführt, es mit Anfragen aus dem Web füttert, Antworten zurückgibt, auf viele Web-Anfragen gleichzeitig reagiert und die Last verteilt. Das ist die Aufgabe von Gunicorn. Wer bearbeitet dann die Anfragen? Die einfache Antwort ist Gunicorn. Die vollständige Antwort ist, dass sowohl Nginx als auch Gunicorn die Anfragen bearbeiten. Grundsätzlich empfängt Nginx die Anfrage, und wenn es sich um eine dynamische Anfrage handelt, dann gibt er sie an Gunicorn weiter. Gunicorn verarbeitet, d.h. er betreibt eine Pool von Threads, und gibt eine Antwort an Nginx zurück, der die Antwort an den urspünglichen Client weiterleitet.

```
import multiprocessing
workers = multiprocessing.cpu_count() * 2 + 1
worker_connection = 1000
```

4.1.6. TLS

Um die Kommunikation über das Netz zu sichern, verwenden wir TLS zur Verschlüsselung der Verbindung und zur Authentifizierung. Bei TLS prüft und signiert eine der Zertifizierungsstellen (CAs) die Echtheit eines Zertifikats mit öffentlichen Schlüsseln eines Servers. Ein Client, der sich über TLS mit dem Server verbindet, kann die Echtheit seines Zertifikats anhand einer digitalen Signatur der CA überprüfen. Um die digitale Signatur zu prüfen, muss ein Client über einen öffentlichen Schlüssel der CA verfügen, der auf einem separaten Weg beschafft und als selbstsigniertes Zertifikat gespeichert wurde.

Um ein Zertifikat zu erhalten, muss zunächst ein privater Sclüssel erstellt werden. Nachdem der Schlüssel generiert wurde, kann ein Zertifikat von einer Zertifizierungsstelle mit einem Certificate Signing Request (CSR) angefordert oder ein Zertifikat selbst signiert werden. Selbstsignierte Zertifikate lassen sich zwar leicht erstellen, werden aber von den Clients standardmäßig abgelehnt, d.h. jeder Client muss so konfiguiert werden, dass er dem selbstsignierten Zertifikat vertraut.

Selbstsigniertes Zertifikat

```
openssl req -x509 -newkey rsa:4096 -days days -keyout key_filename -out cert_filename
```

HTW Zertifikat

Wir können das Zertifikat auch direkt von der HTW beziehen. Folgen Sie den Anweisungen des Links und wir werden ein echtes Zertifikat haben.

4.2. Dokumente

1. ssh(1)
2. ssh_config(5)
3. ssh-keygen(1)
4. Proxy Jump
5. Docker (Linux)
6. Docker Desktop
7. docker(1)
8. docker-build(1)
9. docker-image(1)
10. docker-ps(1)
11. docker-run(1)
12. Dockerfile(5)
13. Nginx
14. Gunicorn
15. SSL/TLS
16. HTW Zertifikat

5. Django

Die Entwicklerdokumentation aller Python Dateien findet sich unter dem folgendem Link: Entwicklerdokumentation