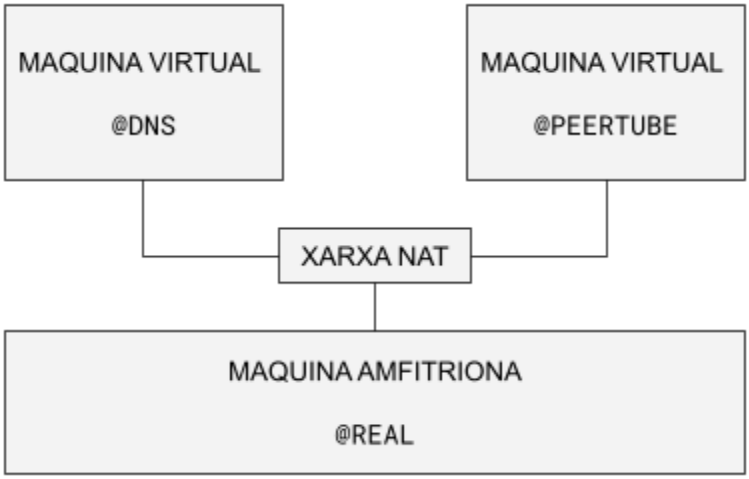
Ansible

Instal·lació del servidor Peertube mitjançant un playbook Ansible

En aquesta pràctica escriurem un playbook Ansible per a la instal·lació automatitzada del servidor Peertube.

El servidor on s'instal·larà Peertube serà una màquina virtual (vm), i també hi ha haurà una altra vm per al servidor DNS. Aquestes màquines les connectarem amb una xarxa nat a la màquina amfitriona.



Prerrequisits

1. Comprovarem que tenim connexió de xarxa entre les tres màquines.
2. Configurem el servidor DNS per a que resolgui el nom `peertube.edt` amb un registre `A` a l'adreça IP de la vm `@PEERTUBE`

```
@DNS: /etc/bind/named.conf.default-zones
```

```
zone "edt" {
```

```
    type master;
    file "/etc/bind/db.edt";
};
```

```
@DNS: /etc/bind/db.edt
```

```
@ SOA ns root 1 4 4 4 4
   NS ns
ns A 192.168.122.119

peertube A 192.168.122.209
```

```
192.168.122.119  <=  IP DE LA MAQUINA VIRTUAL @DNS
192.168.122.209  <=  IP DE LA MAQUINA VIRTUAL @PEERTUBE
```

3. Configurem la resolució de noms a la màquina `@REAL` (amb NetworkManager o amb /etc/resolv.conf) per a que utilitzi el servidor `@DNS`

4. Comprovem que la màquina `@REAL` resol correctament el nom `peertube.edt`

```
user@REAL:~$ host peertube.edt
peertube.edt has address 192.168.122.209
```

5. Comprovem que tenim accés SSH amb clau pública des de la màquina `@REAL` a la màquina `@PEERTUBE`. L'accés ha de ser com a usuari "normal" a la `@REAL`, i accedint com a root a la màquina `@PEERTUBE`:

```
user@REAL:~$ ssh root@peertube.edt
```

Ansible

Què és Ansible?

TLDR; Amb Ansible podem instal·lar i configurar software en altres màquines. Escrivim en la nostra màquina un playbook (o sigui, un script), i ansible connectarà per SSH a les màquines clients que li diguem i executarà el playbook en elles.

Ansible és una eina d'automatització i orquestració de codi obert per al subministrament de programari, la gestió de la configuració i el desplegament de programari. Ansible pot executar i configurar fàcilment sistemes semblants a Unix, així com sistemes Windows per proporcionar infraestructura com a codi. Conté el seu propi llenguatge de programació declaratiu per a la configuració i gestió del sistema.

Ansible és popular per la seva senzillesa d'instal·lació, facilitat d'ús pel que fa a la connectivitat amb els clients, la seva manca d'agent per als clients d'Ansible i la multitud d'habilitats. Funciona connectant-se mitjançant SSH als clients, de manera que no necessita un agent especial al costat del client, i en empènyer els mòduls als clients, els mòduls s'executen localment al costat del client i la sortida es retorna al servidor Ansible.

Com que utilitza SSH, es pot connectar molt fàcilment als clients mitjançant SSH-Keys, simplificant tot el procés. Els detalls del client, com ara els noms d'amfitrió o les adreces IP i els ports SSH, s'emmagatzemen en fitxers anomenats fitxers d'inventari. Un cop hàgiu creat un fitxer d'inventari i l'hagueu emplenat, ansible el pot utilitzar.

Conceptes importants utilitzats a Ansible

Servidor Ansible (Control node):
La màquina on està instal·lat Ansible i des de la qual s'executaran totes les tasques i playbooks.
En el nostre cas el "Control node" és la màquina `@REAL`

Client (Managed node):
Les màquines que es gestionen amb Ansible. Els nodes gestionats també s'anomenen "hosts". Ansible no ha d'estar instal·lat als nodes gestionats.
El nostre "Managed node" serà la vm `@PEERTUBE`

Inventory:
Fitxer que conté la llista de clients.

Fact:
Informació obtinguda dels clients (sistema operatiu que utilitzen, o la memòria ram que tenen)

Module:
Bàsicament, un mòdul és una ordre destinada a ser executada als clients.

Els mòduls s'organitzen en col·leccions, els trobaràs tots aquí: https://docs.ansible.com/ansible/latest/collections/index.html

Per exemple, hi ha el mòdul `apt`, que permet instal·lar software en sistemes que utilitzin apt, com Debian o Ubuntu.
També hi ha, per exemple, el mòdul `user`, que permet administrar usuaris en sistemes Linux, o `win_user` per a Windows.

Playbook:
Llista ordenada de tasques a executar. S'escriuen en format YAML. Són els *scripts* Ansible.

Role:
Una manera d'organitzar les tasques i els fitxers relacionats que poden ser cridats més tard en un Playbook.

Task:
Una tasca és una secció que consta d'un únic procediment a realitzar. És a dir, la execució d'un mòdul als clients.

Handler:
Tasca que només s'executa si hi ha un notificador.

Notifier:
Secció atribuïda a una tasca que crida a un handler si hi ha un canvi a la seva sortida

Tag:
Nom definit per a una tasca que es pot utilitzar més endavant per executar només aquesta tasca o grup de tasques específics.

El primer Playbook

Els playbooks són *scripts* en format YAML on es defineixen les tasques a executar i en quins clients s'han d'executar.

Escriurem el playbook **d'exemple** dintre d'un directori anomenat `peertube_playbook`:

```
user@REAL:~$ mkdir peertube_playbook
user@REAL:~$ cd peertube_playbook
```

El primer pas serà definir l'inventari, és a dir, en quins clients volem executar el playbook. Crearem un fitxer anomenat `hosts` en el qual inclourem la màquina `@PEERTUBE`:

`@REAL: ~/peertube_playbook/hosts`

```
peertube.edt
```

Ara escriurem el Playbook. Per a aquesta pràctica escriurem tot el Playbook en un sol arxiu (també es pot escriure en distints arxius i directoris, organitzats per **roles**).

Tasks

Com hem vist, un playbook és en essència una llista de tasques a realitzar. Cada tasca correspón a un mòdul Ansible que s'ha d'executar amb uns paràmetres. Cada mòdul té els seus propis paràmetres.

Crearem un playbook que tindrà una única tasca que serà instal·lar NGINX en els clients. Utilitzarem el mòdul apt per a aquesta tasca.

Crea l'arxiu `peertube_playbook.yaml` amb el següent contingut:

`@REAL: ~/peertube_playbook/peertube_playbook.yaml`

```yaml
---
- name: Install Peertube
  hosts: all
  remote_user: root

  tasks:
  - name: Install NGINX server
    apt:
      name: nginx
      state: present
```

En aquest playbook hem definit:

Que el nom del playbook és "Install Peertube":
`- name: Install Peertube`

Que s'ha d'executar en tots els clients de l'inventory

```
  hosts: all

Que s'ha de fer la connexió ssh als clients com a usuari remot root
  remote_user: root

La llista de tasques a realitzar:
  tasks:

Una tasca anomenada "Install NGINX server
  - name: Install NGINX server

Que aquesta tasca executa el mòdul apt als clients
    apt:

Que el nom del paquet que ha d'administrar el mòdul apt és nginx
      name: nginx

Que el paquet definit abans estigui present (instal·lat) als clients
      state: present
```

Executa el playbook:

```
user@REAL:~/peertube_playbook$ ansible-playbook -i hosts peertube_playbook.yaml
```

Ara pots comprovar que s'ha instal·lat NGINX a la màquina `peertube.edt` (l'única màquina de l'inventari `hosts`).

Hem vist doncs, que per cada tasca que vulguem executar en els clients, haurem d'utilitzar un mòdul Ansible. Instal·lar software, crear usuaris, crear bases de dades, descarregar arxius, descomprimir arxius, crear directoris, i un llarguíssim etcètera.

Templates

Un mòdul que també s'utilitza molt és el mòdul `template`. Aquest mòdul permet copiar arxius als clients però amb valors dinàmics basats en variables. Són molt útils per a parametritzar els arxius de configuració que s'han de copiar als clients.

Veiem un exemple:

Suposem que un cop instal·lat en NGINX a un client, volem crear en aquest client un arxiu de configuració NGINX amb les directives `port`, `server_name` i `root`.
Però és clar, el valor d'aquestes directives canviarà segons qui utilitzi el playbook i haurà de personalitzar aquests valors.

Aleshores, podem crear una plantilla (amb sintaxi Jinja2) on els valors d'aquestes directives siguin variables:

Crea la plantilla de configuració d'NGINX:

```
@REAL: ~/peertube_playbook/nginx.conf.j2
```

```
server {
    listen {{ port }};
    server_name {{ domain }};
    root {{ root_directory }};
}
```

L'objectiu és que aquest arxiu es copiï als clients, però substituint les variables pels seus valors. Aquests valors els definirem en un altre arxiu que l'usuari del playbook podrà personalitzar.

Crea l'arxiu de variables:

```
@REAL: ~/peertube_playbook/vars.yaml
```

```
port: 80
domain: mysuperdomain.com
root_directory: /var/www/html/mysuperdomain
```

Ara cal indicar al playbook que utilitzi aquest arxiu de variables, i usar el mòdul template per a copiar-lo al client.

```
@REAL: ~/peertube_playbook/peertube_playbook.yaml
```

```
---
- name: Install Peertube
  hosts: all
  remote_user: root
  vars_files:
    - vars.yaml

  tasks:
```

```
  - name: Install NGINX server
    apt:
      name: nginx
      state: present

  - name: Create NGINX config file
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/sites-enabled/default
```

Hem afegit al playbook:

Que utilitzi l'arxiu de variables "vars.yaml"
```
  vars_files:
    - vars.yaml
```

Que executi la tasca anomenada "Create NGINX config file"
```
  - name: Create NGINX config file
```

Que utilitzi el mòudl "template"
```
    template:
```

Per a copiar l'arxiu "nginx.conf.j2" del servidor Ansible
```
      src: nginx.conf.j2
```

A la ruta "/etc/nginx/sites-enabled/default" dels clients (substiuïnt les variables, és clar)
```
      dest: /etc/nginx/sites-enabled/default
```

Les variables es poden utilitzar tant a les templates com al codi del Playbook. Veure Simple Variables.
Es poden definir en el mateix Playbook, o en un altre arxiu i fer referència a aquest arxiu en el playbook. Veure Where to set variables.

Escalada de privilegis

Ansible utilitza sistemes d'escalada de privilegis existents als clients per executar tasques amb privilegis root o amb els permisos d'un altre usuari. Com que aquesta característica ens permet "convertir-nos" en un altre usuari, diferent de l'usuari que va iniciar sessió a la màquina (usuari remot), s'anomena `become` (convertir-se). La paraula clau `become` utilitza eines d'escalada de privilegis existents als clients com *sudo, su, pfexec, doas, pbrun, dzdo, ksu, runas, machinectl* i altres.

Sovint ens trobem amb tasques que s'han d'executar amb un usuari determinat (root o un altre usuari). En aquestes tasques activarem l'escalada de privilegis amb `become: yes` i indicarem l'usuari amb `become_user: usuaridesitjat`

En aquest primer playbook d'exemple que estem realitzant, anem a copiar una pàgina html al client. Però aquest arxiu html no volem que sigui propietat de l'usuari root, que és l'usuari amb que s'executen les tasques d'aquest playbook (recorda que hem indicat al principi `remote_user: root`). Així doncs, haurem de crear un usuari al client, i executar la tasca de copiar l'arxiu html com aquest usuari.

Primer crearem l'arxiu html al servidor ansible:

```
@REAL: ~/peertube_playbook/index.html
```

```
<!DOCTYPE html>
<h1>This is an example page</h1>
```

Després afegirem les tasques de creació de l'usuari, i de còpia de l'arxiu al client:

```
@REAL: ~/peertube_playbook/peertube_playbook.yaml
```

```
---
- name: Install Peertube
  hosts: all
  remote_user: root
  vars_files:
    - vars.yaml

  tasks:
  - name: Install NGINX server
    apt:
      name: nginx
      state: present

  - name: Create NGINX config file
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/sites-enabled/default

  - name: Create user
    user:
      name: mysuperuser
      home: "{{ root_directory }}"

  - name: Copy html file
```

```
    become: true
    become_user: mysuperuser
    copy:
      src: index.html
      dest: "{{ root_directory }}/index.html"
```

Amb el mòdul user podem crear un usuari
```
    user:
```

amb el username "mysuperuser"
```
      name: mysuperuser
```

i crear el seu directori "home" a la ruta especificada a la variable `root_directory` (o sigui `/var/www/html/mysuperdomain`)
```
      home: "{{ root_directory }}"
```

La següent tasca "Copy html file" l'executem amb l'usuari "mysuperuser"
```
  - name: Copy html file
    become: true
    become_user: mysuperuser
```

La tasca utilitza el mòdul copy per a copiar l'arxiu "index.html" del servidor a la carpeta `root_directory` del client
```
    copy:
      src: index.html
      dest: "{{ root_directory }}/index.html"
```

Loops

Hi ha vegades que volem executar un tasca vàries vegades però amb paràmetres diferents. Per a això, Ansible ofereix la paraula clau `loops`.

En aquest playbook d'exemple, podríem voler copiar dos arxius als clients. Podríem fer dues tasques iguals però canviant el nom de l'arxiu, o podem utilitzar un loop.

Per exemple, volem copiar aquest altre arxiu:

```
@REAL: ~/peertube_playbook/other.html
```

```
<!DOCTYPE html>
<h1>Super useful!</h1>
```

Podem escriure el `loop` així:

@REAL: ~/peertube_playbook/peertube_playbook.yaml

```yaml
---
- name: My first playbook
  hosts: all
  remote_user: root
  vars_files:
    - vars.yaml

  tasks:
  - name: Install NGINX server
    apt:
      name: nginx
      state: present

  - name: Create NGINX config file
    template:
      src: nginx.conf.j2
      dest: /etc/nginx/sites-enabled/default

  - name: Create user
    user:
      name: mysuperuser
      home: "{{ root_directory }}"

  - name: Copy html file
    become: true
    become_user: mysuperuser
    copy:
      src: "{{ item }}"
      dest: "{{ root_directory }}/{{ item }}"
    loop:
      - index.html
      - other.html
```

La tasca "Copy html file" s'executarà dues vegades, una per cada element del "loop"

```yaml
    loop:
      - index.html
      - other.html
```

La primera vegada que s'executi, la variable "item" serà igual al primer element "index.html". I el mòdul "copy" s'executarà així:

```yaml
    copy:
      src: "index.html"
```

```
      dest: "{{ root_directory }}/index.html"
```

La segona vegada, la variable "item" serà "other.html":

```
    copy:
      src: "other.html"
      dest: "{{ root_directory }}/other.html"
```

Facts

Els facts són variables amb informació obtinguda dels clients que podem utilitzar en els playbooks

Podem veure tot el diccionari de facts dels clients amb una tasca debug:

```
tasks:
- name: Print all available facts
  debug:
    var: ansible_facts
```

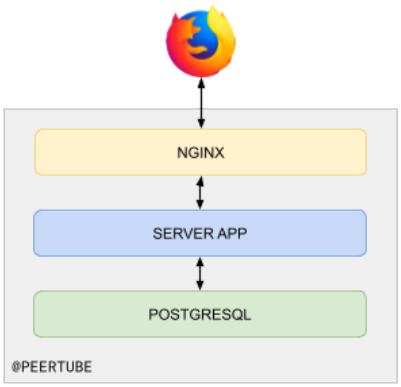
O podem utilitzar una variable concreta del diccionari de facts:

```
tasks:
- name: Print distribution release of clients
  debug:
    msg: "La distro linux es {{ ansible_facts['distribution_release'] }}"
```

El playbook Peertube

La pràctica consisteix en afegir les tasques al playbook per a realitzar totes les passes indicades en el següent document https://github.com/Chocobozzz/PeerTube/blob/develop/support/doc/dependencies.md

L'arquitectura de l'aplicació web Peertube és bàsicament 3 components: un proxy invers NGINX, una server-app en nodejs i una base de dades postgres. (Podeu trobar tots els detalls aquí). Així doncs, totes les tasques de la instal·lació van encaminades a enllestir aquests components.



A continuació es resumeixen les passes del tutorial d'instal·lació, i una petita indicació de com es podrien traslladar a **ansible tasks**. L'única cosa diferent serà que no s'utilitzarà HTTPS per a connectar al proxy NGINX, sinò HTTP.

Dependències

Instal·lar utilitats bàsiques

```
sudo apt install curl sudo unzip vim
```

Es pot utilitzar el mòdul `apt`. El paràmetre `name` accepta una llista.

Instal·lar NodeJS–14

```
curl -fsSL https://deb.nodesource.com/setup_14.x | sudo -E bash -
sudo apt install -y nodejs
```

Observant l'script "setup_14.x" veiem que bàsicament el que fa és dues coses:

```
# 1 DESCARREGAR LA CLAU GPG
node_key_url="https://deb.nodesource.com/gpgkey/nodesource.gpg.key"
local_node_key="/usr/share/keyrings/nodesource.gpg"

if [ -x /usr/bin/curl ]; then
   exec_cmd "curl -s $node_key_url | gpg --dearmor | tee $local_node_key"
else
   exec_cmd "wget -q -O - $node_key_url | gpg --dearmor | tee $local_node_key"
fi

# 2 AFEGIR EL REPOSITORI
exec_cmd "echo 'deb [signed-by=$local_node_key] https://deb.nodesource.com/${NODEREPO} ${DISTRO} main' >
/etc/apt/sources.list.d/nodesource.list"
exec_cmd "echo 'deb-src [signed-by=$local_node_key] https://deb.nodesource.com/${NODEREPO} ${DISTRO} main' >>
/etc/apt/sources.list.d/nodesource.list"
```

Per fer aquestes dues tasques podem utilitzar els mòduls `apt_key` i `apt_repository`
Però cal anar amb compte amb els **requeriments** de cada mòdul, ja que alguns requereixen que els clients tinguin instal·lats alguns paquets, i per tant s'haura d'usar el mòdul `apt` per a instal·lar-los prèviament.

La versió de node es pot posar literal a la `14.x` i la `${DISTRO}` es pot obtenir dels facts del client.

Instal·lar yarn

```
sudo npm install --global yarn
```

La instal·lació de `yarn` es pot fer similar a l'anterior. Podeu veure un exemple aquí:
https://github.com/angristan/ansible-yarn/blob/master/tasks/main.yml

Instal·lar python

```
sudo apt install python3-dev python-is-python3
```

Es pot utilitzar el mòdul `apt`.

Instal·lar dependències comunes

```
sudo apt install certbot nginx ffmpeg postgresql postgresql-contrib openssl g++ make redis-server git cron wget
```

Es pot utilitzar el mòdul `apt`.

Reiniciar redis i postgres

```
sudo systemctl start redis postgresql
```

Es pot utilitzar el mòdul `service`.

👷 Afegir usuari peertube

Crear l'usuari i establir-li un password

```
sudo useradd -m -d /var/www/peertube -s /bin/bash -p peertube peertube
sudo passwd peertube
```

Es pot utilitzar el mòdul `user`.
Cal parar atenció a la "**FAQ entry**" amb els detalls per a generar contrasenyes

🗃️Crear la base de dades

Crear l'usuari de la base de dades

```
cd /var/www/peertube
sudo -u postgres createuser -P peertube
```

(Aquí es posa un password que després s'haurà de posar a l'arxiu production.yaml)

Crear la base de dades

```
sudo -u postgres createdb -O peertube -E UTF8 -T template0 peertube_prod
```

Afegir extensions necessàries

```
sudo -u postgres psql -c "CREATE EXTENSION pg_trgm;" peertube_prod
sudo -u postgres psql -c "CREATE EXTENSION unaccent;" peertube_prod
```

Per a aquestes tasques es pot utilitzar la col·lecció `comunity.postgresql`.

Per a poder utilitzar aquesta col·lecció s'ha d'instal·lar al servidor ansible. Vejeu Using collections

També cal tenir en compte que mòduls d'aquesta col·lecció com `postgresql_user`, `postgresql_db` o `postgresql_ext` requereixen tenir instal·lat prèviament al client el paquet psycopg2. Aquest paquet es pot instal·lar amb `apt` i s'anomena `python3-psycopg2`

S'ha de mirar també que aquestes comandes s'executen amb `sudo -u postgres`.

Instal·lar peertube

Obtenir el número de l'última versió de peertube

```
VERSION=$(curl -s https://api.github.com/repos/chocobozzz/peertube/releases/latest | grep tag_name | cut -d '"' -f 4) && echo
"Latest Peertube version is $VERSION"
```

Aquesta comanda és per a obtenir el número de la última versió de Peertube. Caldria penser si pot ser més convenient que aquest número de versió estigués definit en una variable.

Crear arbre de directoris necessaris

```
cd /var/www/peertube
sudo -u peertube mkdir config storage versions
```

```
sudo -u peertube chmod 750 config/
```

Per a crear directoris es pot fer un loop amb el mòdul `file`.

Atenció a l'usuari i als permisos.

Descarregar peertube, descomprimir i eliminar el zip

```
cd /var/www/peertube/versions
sudo -u peertube wget -q "https://github.com/Chocobozzz/PeerTube/releases/download/${VERSION}/peertube-${VERSION}.zip"
sudo -u peertube unzip -q peertube-${VERSION}.zip && sudo -u peertube rm peertube-${VERSION}.zip
```

Per a descarregar arxius es pot utilitzar el mòdul `get_url`.

Per a descomprimir es pot utilitzar el mòdul `unarchive`. Cal tenir en compte que l'arxiu a descomprimir ja estarà descarregat, i per tant ja existirà en el client.

La versió, com hem dit abans, es pot agafar d'una variable.

Instal·lar peertube

```
cd /var/www/peertube
sudo -u peertube ln -s versions/peertube-${VERSION} ./peertube-latest
cd ./peertube-latest && sudo -H -u peertube yarn install --production --pure-lockfile
```

Per a crear symlinks es pot utilitzar el mòdul `file`.

Per a executar yarn es pot utilitzar el mòdul `community.general.yarn`. Cal instal·lar la col·lecció `community.general`.
El mòdul requereix que estigui instal·lat yarn al client, però ja l'hem instal·lat al principi.

Atenció al paràmetre `production`.

El paràmetre `--pure-lockfile` es pot obviar, ignorar.

Configurar peertube

Copiar la configuració per defecte

```
cd /var/www/peertube
sudo -u peertube cp peertube-latest/config/default.yaml config/default.yaml
```

Aquest arxiu no s'ha de personalitzar ni res, per tant es pot copiar l'arxiu tal qual s'ha descarregat al client. Es pot copiar amb el mòdul `copy`. Compte amb el paràmetre `remote_src`.

Copiar la plantilla de configuració d'exemple

```
cd /var/www/peertube
sudo -u peertube cp peertube-latest/config/production.yaml.example config/production.yaml
```

Per als arxius de configuració el més recomanable és descarregar-lo al servidor i fer una plantilla Jinja2 posant els placeholders per a les variables.

Aquest arxiu `production.yaml` el podem trobar en aquesta url:
https://github.com/Chocobozzz/PeerTube/blob/develop/config/production.yaml.example

En aquest arxiu caldrà canviar la configuració de les opcions corresponents al proxy invers per a que utilitzi HTTP/80 en lloc d'HTTPS/443

```
webserver:
  https: false
  hostname: 'peertube.edt'
  port: 80
```

Un cop ja es té el fitxer, es crea la plantilla i s'utilitza el modul `template` per a copiar-la al client. El `hostname` haurà de ser un placeholder que serà subtituït per la variable corresponent.

Servidor web

Copiar la plantilla de configuració

```
sudo cp /var/www/peertube/peertube-latest/support/nginx/peertube /etc/nginx/sites-available/peertube
```

Fem com abans, descarregem l'arxiu al servidor, el personalitzem i fem el template.
https://github.com/Chocobozzz/PeerTube/blob/develop/support/nginx/peertube

Per a canviar la configuració d'HTTPS a HTTP, caldrà fer les següents modificacions:

```
#server {
#  listen 80;
#  listen [::]:80;
#  server_name peertube.edt;#

#  location /.well-known/acme-challenge/ {
#     default_type "text/plain";
#     root /var/www/certbot;
#  }
#  location / { return 301 https://$host$request_uri; }
#}

server {
  listen 80;
  #listen 443 ssl http2;
  #listen [::]:443 ssl http2;
  server_name peertube.edt;

  access_log /var/log/nginx/peertube.access.log;
  error_log  /var/log/nginx/peertube.error.log;

  ##
  # Certificates
  # you need a certificate to run in production. see https://letsencrypt.org/
  ##
  # ssl_certificate     /etc/letsencrypt/live/peertube.edt/fullchain.pem;
  # ssl_certificate_key /etc/letsencrypt/live/peertube.edt/privkey.pem;
```

Establir el `[peertube-domain]` al fitxer de configuració

```
sudo sed -i 's/${WEBSERVER_HOST}/[peertube-domain]/g' /etc/nginx/sites-available/peertube
sudo sed -i 's/${PEERTUBE_HOST}/127.0.0.1:9000/g' /etc/nginx/sites-available/peertube
```

La variable `${WEBSERVER_HOST}` de la plantilla es pot substituir per una variable Jinja2, i que Ansible agafi el seu valor de l'arxiu de variables.

La variable `${PEERTUBE_HOST}` es pot posar literal com a `127.0.0.1:9000`

Activar el lloc web peertube

```
sudo ln -s /etc/nginx/sites-available/peertube /etc/nginx/sites-enabled/peertube
```

Mòdul `file`.

Reiniciar NGINX

```
sudo systemctl reload nginx
```

Mòdul `service`.

Servei systemd

Copiar la plantilla de configuració

```
sudo cp /var/www/peertube/peertube-latest/support/systemd/peertube.service /etc/systemd/system/
```

Mòdul `copy` o `template`.

Reiniciar el dimoni systemctl

```
sudo systemctl daemon-reload
```

Activar el servei peertube

```
sudo systemctl enable peertube
```

Aquestes dues últimes es poden fer amb una sola task, amb el mòdul `service`.

Resolució de problemes

Durant l'execució del playbook Ansible anirà mostrant el registre de cada tasca. Si alguna tasca ha tingut errors, es pot veure ahí mateix.

Si les tasques s'executen correctament, però l'aplicació no funciona, aleshores cal anar al client (@PEERTUBE) i investigar la causa. Pot ser d'utilitat mirar els logs del proxy invers i de la server-app de peertube:

```
root@PEERTUBE:~# cat /var/log/nginx/peertube.error.log
root@PEERTUBE:~# journalctl -u peertube.service
```